Arquitectura Basada en Tecnología FPGA para la Estimación y Análisis de Información de Flujo Óptico en Tiempo Real

Mauricio de Jesús Vanegas Hernández

Departamento de Arquitectura y Tecnología de computadores

Universidad de Granada

Tesis para optar al grado de

Doctor en Filosofía (PhD)

Junio 2010

Editor: Editorial de la Universidad de Granada
Autor: Mauricio de Jesús Vanegas Hernández
D.L.: GR 3443-2010
ISBN: 978-84-693-5230-4

Declaración

Profesor Dr. Eduardo Ros Vidal, Catedrático de Universidad, y Dr. Javier Díaz Alonso, Profesor Ayudante Doctor de Universidad, ambos del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada,

CERTIFICAN:

Que la memoria titulada "Arquitectura Basada en Tecnología FPGA para la Estimación y Análisis de Información de Flujo Óptico en Tiempo Real", ha sido realizada por D. Mauricio de Jesús Vanegas Hernández bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor por la Universidad de Granada.

Granada, 2 de Junio 2010

Fdo. Eduardo Ros Vidal

Fdo. Javier Díaz Alonso

A mis padres que siempre me han dado libertad para soñar, bondad para compartir con la sociedad, sabiduría para sopesar mis problemas, soporte para enfrentar mis errores y amor infinito para, sin egoísmo, dejar que me aleje de ellos en busca de un sueño.

A mi futura esposa por su apoyo incondicional en mis momentos más difíciles, por su alegría en mis momentos más tristes, por su alma de niña que contagia sueños y por su amor que llena mi vida.

A mis queridos amigos por soportar los altibajos de mi forma de ser tan difícil, por echar un chiste en los momentos de máximo estrés, por compartir sus ideas en los momentos en los que yo carecía de ellas y por haberme dado el mejor de los regalos, ¡su amistad!.

"Vivir sin leer es peligroso, porque te obliga a conformarte con la vida"
Michel Houellebecq.

Agradecimientos

Quiero agradecer a mis padres Germán y Rubia, porque gracias a todos sus esfuerzos hoy puedo defender mi tesis doctoral; a mis hermanos Andrés, Diana y Elizabeth, porque han sido un constante apoyo, su amor y su cariño siempre me acompañan. En general a todos ellos porque yo soy el reflejo de sus corazones.

De igual forma quiero agradecer a mis directores Eduardo y Javier, pues han tenido una paciencia infinita para guiarme en los difíciles quehaceres de una tesis doctoral, asimismo por su apoyo moral cuando tuve problemas personales y en general por confiar en mis capacidades. También quiero agradecer muy especialmente el gran apoyo recibido de Leonardo durante el desarrollo de la tesis, gracias a su gran ayuda puedo entregar los resultados que se muestran en esta tesis.

A mi futura esposa Silvia le debo su incondicional apoyo, ha sido ella quien me ha mantenido positivo en estos últimos días de trabajo en la tesis doctoral, sin sus consejos y sin su ayuda no habría terminado a tiempo el trabajo que hoy está en consideración.

Mis amigos, a ellos también les debo mis agradecimientos; no quiero dar nombres por temor a omitir su gran aporte en esta empresa que hoy termina con la sustentación de esta tesis doctoral.

Abstract

This Ph.D. dissertation proposes a middle level vision system that allows analysing otpical flow cues in real time. Since the European project DRIVSCO was the framework in which this thesis was done, several contributions were done in different areas of interest of the computer vision problem. Generally speaking, a computer vision problem can be identified three different levels in which are defined all the computer vision tasks. The low level vision is characterized by simple mathematical operations applied to a large quantity of data (pixelwise computation). The middle level vision is characterized by complex mathematical operations applied to a not so large quantity of data. Finally, the high level vision is characterized by very complex mathematical operations applied to a little quantity of data. This thesis is focused just in the low and middle level vision.

We have done a significant contribution in the low level vision by developing a memory controller suitable for parallel processing systems accessing to shared data massively. Likewise, we have designed a framework in which were developed the low level vision systems in the DRIVSCO project.

In the middle level vision we have developed a system for the analysis of optical flow cues. This system estimates the ego-motion of a camera in a rigid environment in a robust manner and in real time. Likewise, this system can be used for detecting independent moving objects in a scene. According to the system architecture, we can assert that our system is suitable for standalone navigation applications.

The results obtained in this thesis show that the proposed system is the starting point for future works in the middle and high level vision.

This work has been supported by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, scholarship

No.[E06D101749CO], EU research project DRIVSCO (IST-016276-2), and Spanish Grant (P06-TIC-02007).

Resumen

Esta tesis doctoral propone una aproximación a un sistema en el nivel de media visión que permite analizar información de flujo óptico en tiempo real. Dado que el trabajo realizado estuvo enmarcado en el proyecto Europeo DRIVSCO, se hicieron aportes en diferentes puntos de interés en el problema de la visión por computador. En el problema de visión se pueden identificar tres niveles en los cuales se definen tareas específicas, el nivel de baja visión que se caracteriza por esquemas de procesamiento simples pero sobre una cantidad grande de datos (computación a nivel de píxeles), el nivel de media visión que se caracteriza por esquemas de procesamiento un poco más complejos pero sobre una cantidad de datos que más reducida y el nivel de alta visión que se caracteriza por esquemas de procesamiento muy complejos pero sobre una pequeña cantidad de datos. Esta tesis se centra sólo en los niveles bajo y medio de visión.

En el nivel de baja visión hacemos un aporte significativo al desarrollar un controlador de memoria apto para sistemas con accesos masivos a datos por medio de múltiples entidades realizando tareas en paralelo. Asimismo, se diseña una plataforma que permitió el desarrollo de sistemas en el nivel de baja visión.

En el nivel de media visión se desarrolla un sistema para el análisis de flujo óptico. Este sistema calcula el movimiento propio de una cámara en un entorno rígido de manera robusta y en tiempo real. De igual forma, este sistema puede ser utilizado para la detección de objetos con movimiento independiente en una escena. Dadas sus características, este sistema es idóneo para su aplicación en sistemas de navegación.

Los resultados que se muestran a lo largo de esta tesis demuestran que el sistema desarrollado es un punto de partida para futuros trabajos en el

nivel de media y alta visión.

Esta tesis fue realizada con el apoyo del Programa Alβan, Programa de Becas de Alto Nivel de la Unión Europea para América Latina, beca nº [E06D101749CO], con el apoyo del proyecto Europeo DRIVSCO (IST-016276-2) y con el apoyo del proyecto de la Junta de Andalucía (P06-TIC-02007).

Contenidos

Lista de Figuras

Lista de Tablas

Glosario

AAP por su acrónimo del inglés *Abstract Access Port*, así hemos denominado cada uno de los puertos de conexión de la MCU.

ARM por su acrónimo del inglés *Advanced RISC Machine*, es un procesador desarrollado por ARM Holdings que se caracteriza por tener un set de instrucciones reducido de 32 bits y un conjunto de instrucciones de arquitectura. ARM y Xilinx han anunciado una colaboración conjunta para la incorporación de estos procesadores, y su conexión a periféricos, en las FPGAs de Xilinx.

ASIC por su acrónimo del inglés *Application Specific Integrated Circuit*, es un circuito hecho a medida para una aplicación específica.

AU por su acrónimo del inglés *AAP-Arbitration Unit*, en la arquitectura de la MCU es la entidad encargada de resolver los accesos múltiples a memoria y garantizar el acceso de todos los AAP.

C o estándar ANSI C.

C++ o estándar ISO C++.

DDR por su acrónimo del inglés *Double data rate synchronous Dynamic Random access memory*, es un tipo de memoria RAM que es capaz de utilizar los flancos positivo y negativo del reloj para realizar transacciones de datos.

DDR3 es la versión más actual de la tecnología DDR, se caracteriza por ser de muy alta velocidad, por ejemplo: si se utiliza un reloj de 100 MHz. entonces es capaz de entregar datos a 6400 Mb/s (Mb: mega bytes).

DRIVSCO proyecto europeo llamado "Learning to emulate perception action cycles in a driving school scenario".

FIFO por su acrónimo del inglés *First In, First Out*, es un conjunto de registros que sirven para generar una cola, y en este sentido, para definir el orden de precedencia de los datos.

FPGA por su acrónimo del inglés *Field Programmable Gate Array*, son dispositivos lógicos programables de gran capacidad.

GPU por su acrónimo del inglés *Graphics Processing Unit*, son las tarjetas gráficas utilizadas en los computadores convencionales.

HDL por su acrónimo del inglés *Hardware Description Language*, es comúnmente utilizado para referirse a los lenguajes de descripción de hardware.

IDE por su acrónimo del inglés *Integrated Drive Electronics*, es un estándar para la conexión de dispositivos de almacenamiento masivos.

IMO por su acrónimo del inglés *Independent Moving Object*, son objetos que tienen un movimiento propio y diferente al movimiento de la escena.

LUT por su acrónimo del inglés *Look-Up Table*, son estructuras de datos usadas para reemplazar rutinas de cálculo.

MCU por su acrónimo del inglés *Memory-Control Unit*. Controlador de memoria capaz de abstraer los accesos a memorias externas en sistemas con múltiples unidades de procesamiento trabajando en paralelo. Esta arquitectura es una propuesta novedosa en el marco de esta tesis.

MMU por su acrónimo del inglés *Memory Management Unit*, es el hardware encargado de controlar los accesos a memoria requeridos por un procesador.

MPMC por su acrónimo del inglés *Multi-Port Memory Controller*, es un controlador de memoria DDR desarrollado por la empresa Xilinx para su aplicación en FPGA.

PC-Card por su acrónimo del inglés *Personal Computer Memory Card*, es un estándar para la conexión de dispositivos de almacenamiento en computadores personales, sin embargo, su uso fue extendido para soportar tarjetas de red y módem. Este estándar ha sido desplazado por la aparición del estándar USB.

PCI por su acrónimo del inglés *Peripheral Component Interconnect*, es una especificación estándar para la conexión de componentes en computadores.

PCIe por su acrónimo del inglés *Peripheral Component Interconnect Express*, es una interfaz de conexión estándar para la conexión de componentes en computadores que permite conexiones punto a punto, eliminando la necesidad de un arbitraje en el bus del procesador.

PLB por su acrónimo del inglés *Peripheral Local Bus*, es el bus de datos desarrollado por IBM y adoptado por Xilinx para la conexión de periféricos al PowerPC.

PowerPC es un procesador desarrollado por la alianza entre IBM, Apple y Motorola que se caracteriza por tener un set de instrucciones reducido. IBM y Xilinx hicieron una alianza por medio de la cual algunas FPGAs tienen embebido este tipo de procesadores dentro de sus arquitecturas.

RAM por su acrónimo del inglés *Random Access Memory*, es comúnmente utilizado para referirse a aquellas memorias que son volátiles.

RANSAC es un algoritmo utilizado para estimar de forma iterativa y precisa los parámetros de un modelo matemático a partir de un conjunto de datos observados y en los cuales existen valores atípicos.

RTL por su acrónimo del inglés *Register Transfer Level*, es un nivel descriptivo de los sistemas digitales caracterizado por definir un circuito a través de las conexiones entre las entidades más simples que lo componen, generalmente registros.

SoC Sistema en un Chip.

SPM por su acrónimo del inglés *Scratchpad Memory*, es una simplificación de la memoria cache que usan los microcontroladores para solventar los problemas de accesos lentos a memoria externa y de la existencia de un número muy limitado de registros en el microcontrolador.

VERILOG es un lenguaje usado para la descripción, verificación e implementación de hardware.

VGA por su acrónimo del inglés *Video Graphics Array*, es un estándar de representación gráfica, se caracteriza por definir el tamaño de una imagen como una matriz de puntos de 640x480.

VHDL por su acrónimo del inglés *Very high speed integrated circuit Hardware Description Language*, es un lenguaje usado para descripción, verificación e implementación de hardware.

VLSI por su acrónimo del inglés *Very Large Scale of Integration*, es una clasificación de los circuitos integrados que indica el número de transistores usados en la fabricación de un chip, en este caso representa un chip con billones de transistores.

ZBT por su acrónimo del inglés *Zero-Bus Turnaround*, es un tipo de memoria SSRAM capaz de realizar transacciones a dato por ciclo de reloj.

1

Introducción

En este capítulo se hace un análisis de la problemática al respecto de la visión por computador y de la incidencia que tiene la tecnología en la solución de la misma. Basados en los avances tecnológicos y en las necesidades cotidianas, proponemos un esquema de trabajo para la construcción de un sistema de procesamiento de vídeo en tiempo real capaz de analizar información de flujo óptico y extraer, a partir de este análisis, el movimiento propio de una cámara, así como detectar objetos con movimiento independiente dentro de la escena.

1.1 Motivación

La visión por computador es un campo de estudio en constante evolución. Se caracteriza por ser una disciplina diversa que cubre una gran cantidad de tópicos y que, por su gran diversidad, no tiene una formulación global estandarizada o única. En sus inicios en la década de los setenta, fue una tarea difícil poder procesar la información contenida en imágenes, sin embargo el avance tecnológico, le ha permitido permear otras disciplinas, dificultando aún más una definición general del problema a tratar. De forma simplificada, podemos diferenciar hoy día dos formas de abordar el problema de la visión por computador. Por un lado están los científicos que se ocupan de generar modelos matemáticos de sistemas de visión y por el otro se encuentran quienes aplican estos modelos en los diferentes escenarios de la vida cotidiana.

Esta tesis doctoral se centra en el desarrollo de sistemas de visión para su aplicación en tareas específicas, por tal motivo se aborda el problema desde las dificultades tecnológicas que dificultan la utilización de los complejos modelos matemáticos existentes en tareas como: asistencia en la conducción de vehículos para la disminución de accidentes de tránsito, navegación autónoma de robots y vídeo vigilancia. Estas tareas son de interés actual debido a que benefician, en gran medida, la seguridad de las personas. A continuación, en la sección 1.2, se hará una descripción detallada de lo que significa visión por computador, haciendo una síntesis sobre los problemas que se presentan y la forma en que han sido solucionados, más adelante, en la sección 1.3, se habla de los sistemas en un sólo chip (**SoC**), de su aplicabilidad y de los aspectos más relevantes a tener en cuenta al momento de diseñarlos; antes de finalizar el capítulo, en el apartado 1.4, se describen los objetivos perseguidos en esta tesis y por último, en la sección 1.5, se hará la definición de la hipótesis de trabajo en la que se enmarca este trabajo de tesis.

1.2 Visión por Computador

En su definición más global, visión por computador hace referencia a la ciencia y la tecnología que juntas pueden proveer de un sistema de visión a cualquier máquina. Una simple cámara de vídeo puede proveer información visual a cualquier ente, sin embargo, está muy lejos de poder ser catalogado como un sistema de visión por computador, pues

carece de la capacidad de interpretar su entorno. El sistema visual humano está compuesto por el ojo, capaz de captar la luz de su entorno y enviarla al cerebro a través de impulsos nerviosos para ser procesada en la corteza visual. La visión por computador trata de emular el sistema visual humano, por lo que dos problemas, a priori, tendrán que ser resueltos: el diseño de un transductor de luz que emule la retina humana (aunque sea a un nivel de mero sensor de imágenes, como las cámaras) y el diseño de sistemas capaces de procesar la información entregada por estos transductores. Esta tesis se enmarca dentro del segundo problema e intenta construir un sistema de procesamiento de vídeo en tiempo real.

David Marr [2] propuso que el cerebro procesa la información en tres estados diferentes o niveles de abstracción. El primer estado o *nivel de baja visión*, es el encargado de la extracción de características fundamentales de la escena tales como bordes, formas, etc. El segundo estado o *nivel de media visión*, se encarga de la detección de texturas sobre la escena, según Marr en esta etapa se genera una representación tridimensional de la escena en un plano. Finalmente, el tercer estado o *nivel de alta visión* genera un modelo continuo 3-D de la escena por medio del cual es posible su interpretación. Una definición más detallada, desde el punto de vista de algoritmos, del problema de visión por computador fue entregada por Charles Weems [3] y por Ratha y Jain [4], ellos definen el problema de visión por computador con los mismos tres niveles de visión pero, a diferencia de David Marr, lo hacen en términos de la cantidad de información procesada y de los operadores aritméticos utilizados en cada una de ellas, así, el nivel de baja visión se caracteriza por procesar una gran cantidad de píxeles y operadores simples aplicados a pequeñas vecindades, en el nivel de media visión se realiza un agrupamiento de píxeles tales como la segmentación y etiquetado de regiones, este nivel se caracteriza por un acceso local de datos pero con operaciones sobre píxeles más complejas y, finalmente, el nivel de alta visión realiza tareas orientadas a la toma de decisiones, en este nivel se hacen tareas de reconocimiento de texturas que se caracterizan por un acceso no localizado de datos y por utilizar algoritmos muy complejos y no deterministas. Influenciados por estas teorías, la comunidad científica que estudia la visión por computador ha definido una serie de problemas en cada una de las etapas de visión.

El nivel de baja visión ha sido quizá el más estudiado, en éste se han definido las tareas más clásicas en visión por computador. El problema se centra en la extracción de descriptores locales (detección de bordes, orientación local y energía), así como su

movimiento (flujo óptico) y profundidad (disparidad) en un fotograma. En esta etapa el proceso se ejecuta a nivel de píxel, es decir, una serie de filtros espacio-temporales son aplicados a cada píxel de la imagen de entrada con el fin de extraer información relevante de cada uno de ellos. La capacidad de cómputo necesaria para abordar estas tareas es muy alta, lo que ha motivado una gran cantidad de trabajos a fin de obtener sistemas de procesamiento en tiempo real. Algunos trabajos han sido implementados en procesadores de propósito general [5], otros utilizando tarjetas gráficas (**GPU**) [6] y otros mediante el diseño de circuitos hechos a medida (**ASIC**) [7] o en dispositivos lógicos programables (**FPGA**) [8, 9, 10].

Para detectar las texturas en una escena, en el nivel de media visión, se siguen dos esquemas totalmente diferentes; por un lado hay quienes plantean la solución de forma local, evaluando las características de cada píxel en una pequeña vecindad y por el otro están aquellos que insisten que las texturas en una imagen deben ser definidas evaluando las características del píxel en toda la escena. Aunque la comunidad científica aún no tiene un consenso sobre en qué nivel ubicar las tareas de procesamiento, el análisis e interpretación de la información del flujo óptico puede decirse que se realiza en este nivel. El flujo óptico contiene información de la estructura tridimensional de una escena (si la cámara se mueve), así como de los objetos con movimiento independiente (**IMO**) dentro de la misma; Bruss y Horn [11] introdujeron uno de los primeros modelos matemáticos para la estimación del *movimiento propio* de una cámara (conocido generalmente por su término inglés de *ego-motion*). Ellos definen el movimiento de la cámara como una rotación y una traslación que sumadas dan como resultado el vector de movimiento de la cámara; con esta hipótesis de partida generan un problema de minimización en el que utilizan el cálculo de flujo óptico para ajustar una línea de tendencia. Basados en este modelo, muchos otros científicos han intentado solucionar el problema de minimización desde muchas perspectivas, sin embargo, éste sigue siendo un tópico de investigación abierto. Es de resaltar que el nivel de media visión condensa la información que proviene del nivel de baja visión, lo que a priori permite la utilización de elementos de cómputo de menor capacidad.

Completando el marco de la definición de los problemas en visión por computador, están los desafíos del nivel de alta visión. Forsyth y Ponce [12] definen cinco desafíos principalmente: reconocimiento, segmentación, racionalización de relaciones, construcción de conocimiento y generalización. El cerebro humano es capaz de segmentar una

imagen y reconocer todos y cada uno de los elementos que la componen, clasificando estos elementos desde la racionalización de lo que se observa; asimismo es capaz de utilizar esta información para generar conocimiento adicional y finalmente, generalizar este conocimiento para poder utilizarlo posteriormente. Esto es una tarea muy compleja que aún está en sus inicios y que no está relacionada con el trabajo realizado en el marco de esta tesis pero que se nutre y requiere del análisis de las características que aquí se realizan (en concreto en lo referente al análisis del movimiento).

1.3 Sistemas en un Chip (SoC)

En su definición más general, un **SoC** es la integración de todos los componentes funcionales necesarios para ejecutar una determinada tarea en un sólo circuito integrado. En el marco de visión por computador, un **SoC** debe contener, entre otros elementos, circuitos para el control de memorias externas de gran capacidad de almacenamiento y de gran ancho de banda, circuitos para la adquisición de vídeo y circuitos para el procesamiento de la información.

Actualmente existen en el mercado una gran cantidad de **SoC** para aplicaciones multimedia. Marvell[1], por ejemplo, ha puesto en el mercado un **SoC** para el procesamiento de vídeo en formato de alta definición; este sistema cuenta con dos procesadores Sheeva[2] que pueden trabajar con relojes de hasta 1.2 GHz., decodificadores de vídeo y audio para diferentes estándares, y entidades para el procesamiento de imágenes que ya traen implementados algunos algoritmos como por ejemplo: escalado de imágenes, entrelazado 3D, mejora del contraste en forma adaptativa, etc. Fujitsu[3], por su parte, ha introducido un **SoC** basado en un procesador con una **MMU** y dos buses de datos que permite una conexión fácil con las interfaces **PCI**, **IDE** y **PC-Card**. Este **SoC** es utilizado en aplicaciones de compresión de datos de audio y vídeo **VGA** en tiempo real. La capacidad de cómputo que entregan los sistemas antes mencionados es muy alta, sin embargo, todos ellos han sido desarrollados para ser utilizados en computadores convencionales como plataformas aceleradoras en el procesamiento de información multimedia.

[1] Marvell® Technology Group Ltd.

[2] Sheeva™ 88SV131 ARM v5TE, es un procesador con una unidad de control de memoria (**MMU**) integrada y niveles de cache L1/L2.

[3] Fujitsu Microelectronics Asia Pte Ltd.

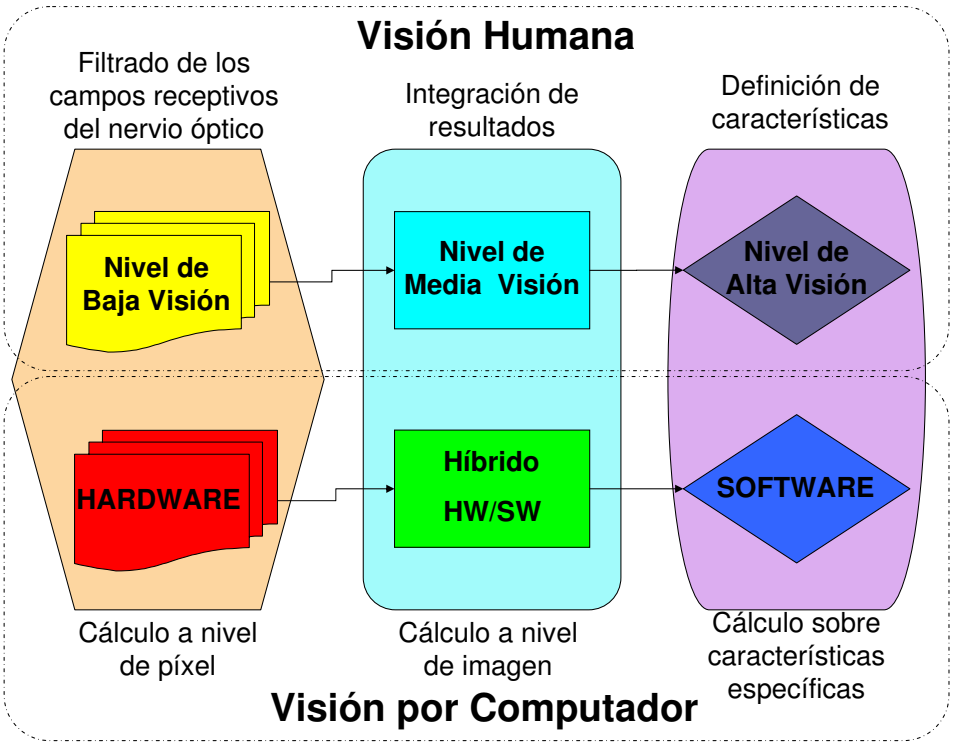


Figura 1.1: Esquema de los sistemas de visión.

En el campo de la visión por computador, los **SoC** representan un punto de partida para el desarrollo de sistemas complejos que puedan ser utilizados en aplicaciones donde el uso de computadores convencionales no es posible (en el marco de sistemas empotrados), como por ejemplo la robótica o la asistencia a la conducción. Debido a su bajo nivel de consumo de potencia y a su versatilidad, hemos encaminado nuestros esfuerzos, en el marco de esta tesis, para obtener un **SoC** capaz de procesar información en los diferentes niveles de visión; en el nivel de baja visión procesar datos por medio de hardware hecho a medida, en el nivel de media visión procesar datos por medio de procesadores empotrados en el mismo chip con apoyo de máquinas aceleradoras y así dejar una herramienta disponible para futuras aplicaciones en el nivel de alta visión, Figura 1.1.

Teniendo en cuenta el esquema de la Figura 1.1, es necesario estudiar el tipo de tecnología que mejor se ajuste a los requerimientos del problema de visión por computador. Aunque las **GPU** tienen una capacidad de procesamiento enorme y se ha demostrado que en ellas es posible ejecutar la gran mayoría de modelos en los niveles

bajo y medio de visión en tiempo real [13], su utilización en aplicaciones como la asistencia a la conducción aún no es posible; su alto consumo de potencia y su incapacidad para ser sistemas autónomos las releva a tareas de coprocesamiento en computadores convencionales; actualmente, las **GPU** se utilizan en la etapa de validación de modelos en tiempo real. Los **ASIC** representan el tipo de tecnología que mejor se adapta para el diseño de sistemas homologables por criterios estandarizados de calidad. Su capacidad de cómputo puede ser igual o superior a la entregada por las **GPU** y aún así su consumo de potencia es el mínimo posible, sin embargo, el tiempo de diseño de circuitos utilizando esta tecnología es muy elevado y, adicionalmente, su implementación es muy costosa. En una forma alternativa es posible utilizar las **FPGA** como tecnología de desarrollo; aunque su capacidad de cómputo es inferior a la capacidad de las tecnologías antes mencionadas y en particular los **ASIC** superan cualquier diseño hecho en una **FPGA**, no existe una tecnología que mejor integre la versatilidad de las **GPU** y el rendimiento de los **ASIC**. En la última década el gap entre los **ASIC** y las **FPGA** ha disminuido enormemente [14], permitiendo que las **FPGA** puedan ser utilizadas, en lugar de los **ASIC**, en muchos campos de aplicación tanto en la academia como en la industria. Algunas de las **FPGA** actuales cuentan con procesadores empotrados dentro de sus arquitecturas, lo cual potencia aún más la posibilidad de escoger estos dispositivos para el diseño de un **SoC** que pueda ser utilizado en aplicaciones de tiempo real.

Finalmente, para la descripción del sistema es necesario utilizar herramientas de síntesis que permitan obtener diseños que ejecuten su función al máximo rendimiento posible, esto es: máxima velocidad de ejecución con la mínima cantidad posible de recursos hardware utilizados. En el marco de esta tesis se utilizaron dos lenguajes de descripción de hardware, el **VHDL** por ser un estándar mundialmente utilizado y que se caracteriza por permitir descripciones de hardware en diferentes niveles de abstracción, y *Handel-C*[1] por ser un lenguaje que permite el diseño de algoritmos complejos en hardware mediante una descripción simple en un lenguaje similar al lenguaje **C**. El **VHDL** es utilizado en la descripción de circuitos con restricciones de tiempo muy específicas, generalmente se ha utilizado en la descripción de interfaces con elementos

[1]Lenguaje de descripción de hardware desarrollado en la universidad de Oxford. Se puede definir como un subconjunto de sentencias tomadas del lenguaje C para su aplicación en la definición de circuitos.

externos a la **FPGA**. La descripción de los circuitos que hacen parte del nivel de baja visión se hizo mediante el uso de *Handel-C* por ser algoritmos más complejos. Ortigosa et al. [15] afirmaron tener buenos resultados en sus descripciones de hardware mediante el uso de *Handel-C*, en su trabajo se compararon varios lenguajes de descripción de hardware evaluando sus prestaciones y el consumo de recursos.

1.4 Objetivos de la Tesis

La hipótesis de trabajo de esta tesis doctoral se basa en que aunque se pueden desarrollar circuitos de muy altas prestaciones para modelos del nivel de baja visión en los que se realiza computación masiva a nivel de píxel, no se puede adoptar la misma estrategia de diseño para modelos del nivel de media visión como la segmentación y la estimación del patrón de *movimiento propio*, ya que son inherentemente iterativos. La implementación eficiente de estos modelos de visión como sistemas en un sólo chip requiere de la integración de componentes hardware de propósito específico con procesadores de propósito general ejecutando módulos software inherentemente iterativos. Esto requiere aplicar eficientemente técnicas de diseño híbrido hardware/software y adaptar los modelos de visión para aprovechar de forma eficiente los recursos computacionales disponibles en los dispositivos **FPGA**; por lo tanto, en esta tesis doctoral se abordan concretamente los siguientes objetivos fundamentales:

- Desarrollo de una plataforma hardware/software para el procesamiento de imágenes en tiempo real.
- Implementación de un modelo de estimación de *movimiento propio* utilizando estrategias de diseño híbrido hardware/software, es decir, implementando los módulos de cálculo intensivo local como componentes hardware (IP cores) y los elementos de cálculo iterativo como módulos software en procesadores empotrados en el mismo chip.
- Implementación de un modelo sencillo de detección de **IMO**s (Objetos con Movimiento Independiente).

Además se acometen también dos objetivos importantes para que los resultados obtenidos se utilicen más allá del marco de esta tesis doctoral, estos objetivos estratégicos son:

- Implementación de un controlador eficiente de memoria *SSRAM*, que abstraiga al diseñador de sistemas complejos de la temporización fina en los accesos a soporte de memoria externa. Esto es fundamental para la implementación de sistemas de procesamiento de imágenes que requieran utilización intensiva de recursos de memoria externa para almacenamiento temporal de datos.

- Integración de la arquitectura desarrollada en la plataforma del proyecto europeo **DRIVSCO** en el cual se enmarca el presente trabajo.

1.5 Motivación y Contenido de la Tesis

Esta tesis tiene sus cimientos en el proyecto europeo **DRIVSCO**[1]. En este proyecto uno de los objetivos fundamentales era la implementación de un sistema de visión por computador en tiempo real con el fin de potenciar su aplicación en sistemas de ayuda a la conducción. Algunos de los participantes del proyecto estaban directamente relacionados con la industria automovilística y fueron ellos quienes definieron un marco de posibles aplicaciones para los sistemas de visión. También se ha colaborado con los proyectos nacionales DINAM-VISION (DPI2007-61683) y MULTIVISION (TIC-3873), para evaluar los resultados de esta tesis en otros campos de aplicación como videovigilancia.

Díaz [16] mostró con múltiples ejemplos que el uso de las **FPGA** en sistemas para la visión por computador es viable, en su trabajo diseñó un sistema capaz de estimar movimiento y profundidad en tiempo real, entre otras características que el sistema puede estimar. Como continuación de su trabajo hemos propuesto como hipótesis de trabajo que, dada la evolución de los dispositivos lógicos programables (**FPGA**) y de las herramientas de diseño de hardware, es posible desarrollar bloques de cómputo que permitan sacar el máximo provecho del paralelismo inherente de las **FPGA**, aplicándolos al procesamiento de fotogramas a nivel de píxel, es decir, en el nivel de baja visión; asimismo, dada la coexistencia de procesadores y recursos hardware de la **FPGA**, es posible construir una arquitectura computacional basada en procesadores en la que se conecten estos bloques de cómputo masivo con un procesador y así sacar el máximo provecho de la flexibilidad de los procesadores, ejecutando en ellos los modelos del nivel

[1] Proyecto número IST-016276-2 desarrollado durante el período comprendido entre enero de 2006 y julio de 2009.

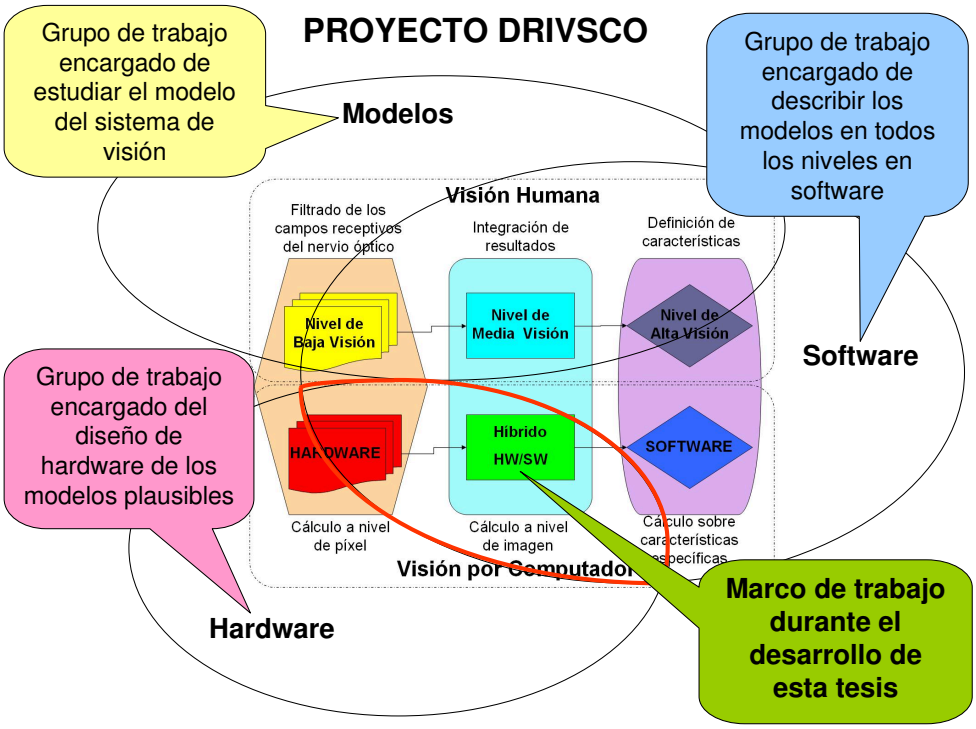


Figura 1.2: Esquema de trabajo dentro del proyecto **DRIVSCO**.

de media visión; y de esta forma es posible construir un sistema de visión por computador en un sólo chip (**SoC**) que pueda ser utilizado en aplicaciones como robots autónomos, vídeo vigilancia y asistencia en la conducción de vehículos.

Son muchos los retos que deben ser encarados para conseguir el sistema de visión de la Figura 1.1. El problema del diseño debe ser abordado desde tres perspectivas diferentes, por un lado es necesario generar modelos matemáticos de visión, de otra parte, es necesario describir estos modelos en software para verificar su adecuado comportamiento y por último implementar estos modelos en hardware (**SoC**). Un esquema de trabajo en el marco del proyecto **DRIVSCO** es mostrado en la Figura 1.2; la Universidad de Granada, como socia del proyecto, era la encargada de la implementación de modelos en hardware y es dentro de este marco en el que se realizó el trabajo de esta tesis.

Como ya ha sido expresado antes, la **FPGA** fue escogida como plataforma para el desarrollo del sistema de visión y con ella se escogió también *Handel-C* como el lenguaje a utilizar para la descripción de hardware; como ya se dijo antes, este lenguaje permite una descripción funcional de algoritmos muy sencilla, sin embargo, la descripción de

interfaces con elementos externos a la **FPGA**, en particular con memorias externas, es ineficiente y poco recomendable si el objetivo es obtener sistemas de alto rendimiento. Adicionalmente, la ventaja de trabajar con la **FPGA** es su capacidad de cómputo en paralelo y es precisamente ésto lo que permite que procesos masivos de cálculo, como los realizados en el nivel de baja visión, puedan ser ejecutados de forma eficiente en la **FPGA**; este paralelismo se pierde cuando es necesario utilizar elementos secuenciales como las memorias en el proceso de cómputo, es por esto que una parte muy importante en el desarrollo de esta tesis fue el diseño de una arquitectura novedosa para el acceso a memoria en sistemas con múltiples entidades de cálculo accediendo a memoria compartida de forma concurrente (**MCU**). Este trabajo permitió a los demás integrantes del grupo de la Universidad de Granada centrarse sólo en el desarrollo de los circuitos de cálculo abstrayéndose de las complejas interfaces con memorias externas a la **FPGA**; asimismo, esta **MCU** fue incorporada en el lenguaje *Handel-C* como funciones de escritura y lectura de memorias con el fin de facilitar su uso. Finalmente, gracias a los dos dominios de reloj con los que cuenta la **MCU** es posible utilizar la memoria externa a su máximo rendimiento aunque las entidades de cálculo utilicen relojes mucho más lentos.

Una vez solucionado el problema de los accesos a memoria, el trabajo se centró en la construcción de un arquitectura basada en un procesador conectado a periféricos capaces de estimar y analizar movimiento a partir de fotogramas en tiempo real. En este punto del trabajo se obtuvo una colaboración enorme de otros compañeros del equipo de trabajo en **DRIVSCO** (IST-016276-2), DINAM-VISION (DPI2007-61683) y MULTIVISION (TIC-3873), pues se utilizaron las entidades de procesamiento que ellos construyeron en el marco de sus trabajos de tesis. Ellos se centraron en la descripción en hardware de los modelos del nivel de baja visión y, utilizando parte de sus trabajos, se pretende conseguir una plataforma de visión por computador capaz de procesar modelos en el nivel de media visión, en particular la estimación de **movimiento propio** y la detección de los **IMO**s presentes en secuencias de fotogramas, todo esto en tiempo real.

Las contribuciones más importantes del trabajo desarrollado en esta tesis para el proyecto **DRIVSCO** se pueden ver en el diagrama que se muestra en la Figura 1.3 y se enumeran a continuación:

- Diseño e implementación de la interfaz hardware del sistema **DRIVSCO**.

- Diseño e implementación del controlador de memoria **MCU**.
- Diseño e implementación de un módulo de compensación de movimiento y de rectificación de pares de imágenes estereoscópicas usando interpolación bilineal.
- Diseño e implementación de una arquitectura híbrida hardware/software para la estimación de *movimiento propio* y la detección de **IMO**s.

Notemos que, como se desprende de las lista de las tareas anteriores, el hilo conductor de la tesis consisten en el diseño de sistemas mixtos hardware/software de altas prestaciones. Asociado a ello y dada la limitada capacidad de cómputo de los procesadores empotrados, existe la necesidad de desarrollar módulos de gestión de memoria y comunicaciones de altas prestaciones. Finalmente, los módulos de compensación de movimiento y de rectificación son un ejemplo claro de esta problemática ya que requieren un acceso aleatorio a un gran volumen de datos y por ello la gestión eficiente de la memoria se hace primordial para alcanzar procesamiento de imágenes en tiempo real.

Es necesario remarcar que muchas personas trabajaron en el proyecto **DRIVSCO** dentro de la Universidad de Granada y que cada una tenía un papel específico dentro del esquema de trabajo; hemos intentado dejar claro, en los párrafos anteriores, cual es el objetivo de esta tesis y para dar cuenta de la forma en que estos objetivos fueron cumplidos, hemos dividido el texto a continuación de la siguiente manera: el capítulo 2 inicia con un análisis detallado de los esquemas de control de memoria más usados en los sistemas de procesamiento actuales, indicando cuál es la característica más representativa de cada uno de ellos; una vez analizado los pros y contras de los esquemas convencionales de control de memorias en sistemas de cómputo paralelo, se hace una propuesta de cómo debe ser el control de memoria más adecuado para arquitecturas de cálculo basadas en caminos paralelos de procesamiento de datos; este enfoque particular del problema es debido al estudio anterior que se hizo en grupo sobre el tipo de arquitectura más eficiente para el computo de algoritmos en el nivel de baja visión; en el resto del capítulo se hace una descripción detallada de la arquitectura del controlador de memoria propuesto y, al final del mismo, se hace un análisis del comportamiento del controlador en una aplicación real.

En el capítulo 3 se hace una descripción breve del modelo matemático para la estimación del *movimiento propio*, haciendo referencia al modelo de flujo óptico utilizado y las simplificaciones hechas para obtener una arquitectura **SoC** viable para aplicaciones

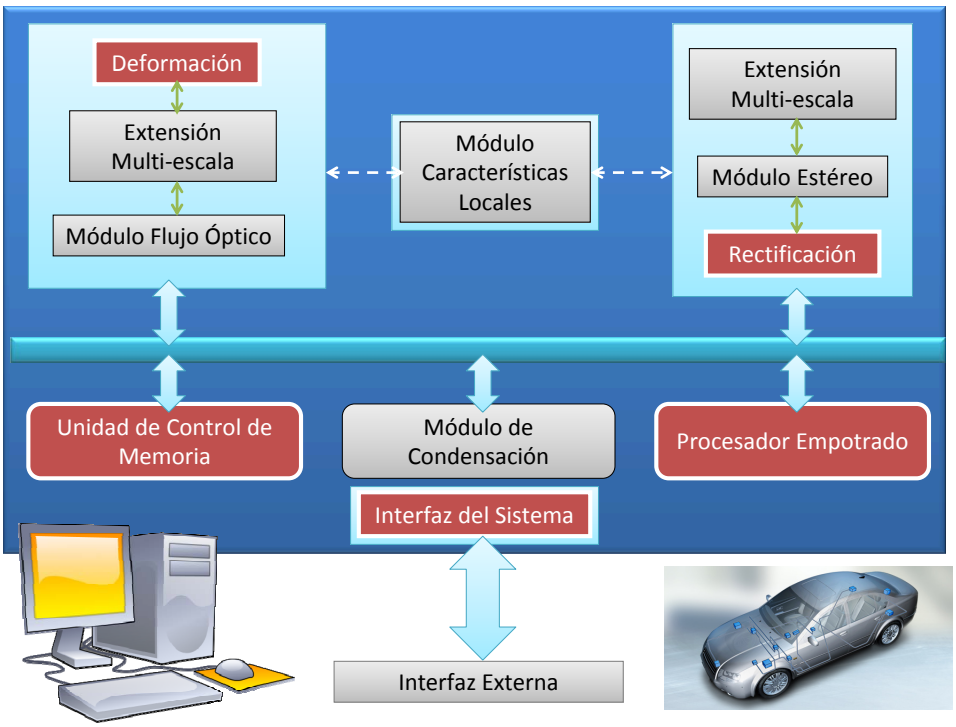


Figura 1.3: Diagrama en bloques del sistema desarrollado en el marco del proyecto **DRIVSCO**.

en tiempo real; asimismo se presenta un modelo simple para la estimación de los **IMO**s a partir del cálculo de *movimiento propio*.

El capítulo 4 describe en detalle la arquitectura **SoC** del sistema propuesto, partiendo del estudio de los sistemas actuales de visión por computador, su aplicación más frecuente y sus debilidades; una vez descrito el estado del arte, hacemos una primera propuesta de la arquitectura para la estimación de *movimiento propio* y evaluamos su rendimiento en términos de velocidad de ejecución, consumo de recursos y precisión; finalmente, hacemos un estudio del código **C** para determinar las secciones de código con alta carga computacional y así poder construir bloques de aceleración de procesos en hardware con el fin de mejorar el rendimiento del **SoC**.

Como capítulo final tenemos la discusión de los resultados, el capítulo 5 contiene una serie de conclusiones sobre el trabajo realizado, en ellas tratamos de mostrar las ventajas del sistema propuesto, sus posibles aplicaciones, pero también mostramos las diferencias y los puntos susceptibles de mejoras; por último mostramos las líneas de

estudio que se abren una vez concluido este trabajo.

2

Controlador de Memoria

Dada la gran cantidad de datos que deben ser procesados por los sistemas de procesamiento de vídeo, hemos propuesto un controlador de memoria con múltiples puertos capaz de aprovechar al máximo el ancho de banda de las memorias **RAM** estáticas. En este capítulo se hace una descripción detallada de la arquitectura del controlador de memoria que se basa en un dominio cruzado de relojes que permite, por un lado trabajar a la frecuencia de reloj de la memoria y por el otro proveer datos en cada canal a la frecuencia de las entidades conectadas a éstos. Finalmente, por medio de un caso de estudio se evalúa el rendimiento del controlador de memoria propuesto y se demuestra su versatilidad de uso.

2.1 Introducción

En los últimos años, las **FPGA**s han dejado de ser utilizadas como simples plataformas de prueba, para ser utilizadas como sistemas de cómputo. Debido a que el gap entre las tecnologías **ASIC** y **FPGA** ha disminuido significativamente [14], los **ASIC**s han sido reemplazados por **FPGA** en algunos campos de la industria electrónica durante la última década. En el campo de las redes de datos, por ejemplo, los conmutadores de datos utilizan las **FPGA**s con el fin de disminuir los tiempos de diseño y los costos asociados a éstos, de esta forma se garantiza un acceso al mercado mucho más rápido. Actualmente, las **FPGA**s están siendo usadas en el diseño de **SoC** [17, 18] debido a la cantidad de unidades funcionales que las conforman, algunas de ellas por ejemplo, tienen procesadores incluidos como unidades funcionales previamente definidas dentro de su circuito.

En las **FPGA**s modernas es posible diseñar caminos de datos finamente segmentados usando además unidades superescalares[1] de procesamiento (ver por ejemplo [8, 9]). Además, algunos algoritmos para el procesamiento de datos siguen patrones simples que permiten aprovechar la capacidad de computación en paralelo que ofrecen las **FPGA**s. Desafortunadamente, los conocimientos y habilidades necesarios para lograr sistemas complejos mediante el uso de las **FPGA**s es mayor que los necesarios para desarrollar el mismo sistema pero mediante el uso de las **GPU**s; esto se debe, entre otros factores, a que las **GPU**s ya tienen definidos los esquemas de acceso a memoria, así como sus interfaces de conexión.

El rendimiento de las plataformas de cómputo es muy sensible al comportamiento de los sistemas de memoria y a sus limitaciones. Tradicionalmente, los sistemas basados en procesadores utilizan esquemas con jerarquía de memoria en los cuales, las memorias de poca capacidad de almacenamiento y tiempos cortos de acceso están ubicadas muy cerca de los procesadores, mientras que las memorias de gran capacidad de almacenamiento y con grandes tiempos de acceso están más alejadas [19]. En la práctica, bloques enteros de datos son movidos desde las memorias de gran capacidad hacia aquellas de menor capacidad basándose en principios de localidad y/o temporalidad de datos [20, 21]; esto le permite a los procesadores tener un acceso muy rápido a los datos. Aunque estos

[1] Superescalar es el término utilizado para designar un tipo de arquitectura de procesador capaz de ejecutar más de una instrucción por ciclo de reloj.

principios funcionan bien en la mayoría de los algoritmos, el rendimiento de este tipo de sistemas se ve degradado cuando es necesario un acceso aleatorio a los datos; de hecho, las técnicas para optimización de código se basan en la estructura de los datos que se desean procesar [5]. Un ejemplo de ello es que, en el procesamiento de vídeo en tiempo real, utilizar esquemas de compresión no conlleva a un incremento en el rendimiento del procesador al acceder a una menor cantidad de datos, pues el acceso a éstos es totalmente dependiente de datos anteriores.

Múltiples entidades de procesamiento trabajando en paralelo son posibles en los dispositivos **FPGA** actuales [22], sin embargo, este paralelismo se pierde en el momento en que todas estas entidades necesitan acceder a datos contenidos en memorias externas. El funcionamiento inherentemente secuencial de las memorias puede limitar el rendimiento de sistemas de procesamiento en paralelo; este potencial cuello de botella puede ser controlado de forma efectiva mediante la implementación de accesos a memoria por ventanas de tiempo [10]. Sin embargo, dado que el acceso a memoria es una tarea crucial en cualquier tipo de sistema, sería muy útil poder abstraer este acceso mediante una arquitectura general que facilite el diseño de sistemas con múltiples entidades con acceso intensivo a memoria. En este capítulo se presenta la arquitectura de un controlador de memoria especialmente diseñado para **FPGA** y ser usado en **SoC**. Sin detrimento de lo anterior, la arquitectura del controlador de memoria puede ser fácilmente implementada en un **ASIC** debido a su descripción **RTL**.

Hoy en día es común el uso de herramientas de diseño de alto nivel de abstracción, tanto en la academia como en la industria. Particularmente el diseño de circuitos para implementación en **FPGA**, se hace mediante el uso de este tipo de herramientas con el fin de disminuir los tiempo de diseño. Actualmente, es posible generar diseños complejos a través de la conexión esquemática de cajas en un entorno visual mediante el uso de la herramienta *System Generator* [23] o hacer una descripción algorítmica mediante el uso de lenguajes de descripción de hardware basados en **C** [24, 25, 26] o en **C++** [27]. En la última década, estos lenguajes y herramientas han sido evolucionados y ampliamente utilizados; se presentan como una alternativa para que personas no expertas en el diseño de hardware puedan utilizar **FPGA**s o, en muchas ocasiones, para disminuir los tiempos de diseño de sistemas complejos. Sin embargo, aunque estos lenguajes aceleran el proceso de diseño y permiten obtener sistemas con rendimientos aceptables, no es posible generar interfaces óptimas con periféricos, de hecho, las interfaces de control con

memorias externas se define, normalmente, con lenguajes de descripción de hardware de bajo nivel. En este capítulo se describe un controlador de memoria capaz de abstraer el acceso a la misma en diseños con múltiples entidades de procesamiento trabajando en paralelo, así como mostrar las ventajas de su aplicación en sistemas diseñados con herramientas de alto nivel.

Este capítulo está organizado de la siguiente manera: en la sección 2.2 se hace un análisis de las soluciones presentadas hasta el momento con respecto a los sistemas de memoria; en la sección 2.3 se describe en detalle la arquitectura de la **MCU**, así como los resultados de su implementación. Finalmente, en la sección 2.4 se evalúa el rendimiento de la **MCU** en un sistema real.

2.2 Trabajos Relacionados

En la literatura se han presentado una gran cantidad de esquemas para mejorar el acceso a memoria en sistemas multimedia. Liu et al. [28] ha desarrollado una técnica para la optimización de accesos, en memorias tipo *scratchpad* (SMP), durante la ejecución de ciclos anidados mediante la reutilización de datos. Ranganathan et al. [29] han desarrollado en una **FPGA** un tipo de memoria *cache*, para ser utilizada en sistemas basados en procesadores, que permite la configuración dinámica de sus parámetros; esta configuración dinámica aplicada en diferentes fases de cualquier proceso incrementa el rendimiento de los sistemas que la usan. Sohi et al. [30] ha incrementado el paralelismo de datos mediante el uso de memorias *cache* con múltiples puertos. En un trabajo diferente, Heithecker et al. [31] propusieron e implementaron, en una **FPGA**, un controlador de memoria que se basa en un acceso programado a los datos, de esta forma son capaces de disminuir las latencias inherentes de las memorias y permitir un acceso concurrente desde múltiples unidades de procesamiento. En un trabajo reciente, Su-Shin et al. [32] propusieron una arquitectura híbrida que utiliza una memoria *cache* y una memoria **SPM** juntas con el fin de sacar el máximo beneficio a la reutilización de datos, incluso, en sistemas con acceso aleatorio. Todos estos modelos propuestos son dependientes del tipo de aplicación en el que se usen para aprovechar su máximo rendimiento, por tal motivo sería muy útil una solución más general que no dependa de la aplicación en la que se utilice para entregar su máxima funcionalidad.

El problema de acceder de forma óptima a los datos contenidos en una memoria

es aún más complejo en sistemas con múltiples unidades funcionales. En este caso, la solución más común es utilizar memorias *cache* de poca capacidad de almacenamiento en cada uno de los procesadores y compartir memorias *cache* de alta capacidad [33]. Por otra parte, cuando las aplicaciones requieren de grandes volúmenes de transferencia de datos, entonces la solución es utilizar arquitecturas similares a la mencionada anteriormente, pero con memorias de gran ancho de banda como las usadas en las **GPU**s. Por ejemplo, una **GPU** trabajando en la interpretación de imágenes de alta resolución, requiere memorias con gran ancho de banda [34, 35]; una Tesla NVIDIA utiliza 16 bancos de memoria **DDR3** de 32 bits con relojes a 1,6 GHz. El ancho de banda de estas memorias es impresionante y es el responsable de rendimiento de este tipo de arquitecturas.

Lo antes mencionado muestra la importancia de un arquitectura de control de memoria adecuada. Desafortunadamente, el acceso a memoria en el campo de los sistemas empotrados es, generalmente, el que limita el rendimiento del sistema. De hecho, las plataformas de diseño no cuentan, usualmente, con memorias de gran ancho de banda y la cantidad de memorias disponibles, sea cual sea su tipo, es limitado con el fin de reducir el costo y la disipación de potencia [36]. Actualmente, los fabricantes de las **FPGA**s desarrollan controladores para memorias de gran ancho de banda (**DDR**, **DDR3**, etc.) y las ofrecen a sus clientes con el fin de facilitar el diseño de sistemas complejos. La empresa Xilinx [37] ofrece el **MPMC** con ocho puertos de conexión independientes; cada uno de ellos trabaja a frecuencias más bajas que la utilizada en la interfaz directa con la memoria **DDR**. Esta estrategia usada por los diseñadores de Xilinx es muy útil, debido a que la gran mayoría de los diseños complejos que son implementados en **FPGA**, trabajan a frecuencias mucho menores que las utilizadas por la memoria.

Hasta ahora, todos los trabajos han sido enfocados en transferencias masivas de datos con el fin de aprovechar al máximo el ancho de banda de las memorias **DDR**, y de igual forma han enfocado sus esfuerzos en encontrar patrones espaciales y temporales de localidad de datos para incrementar el rendimiento de sus sistemas. Sin embargo, la **MCU** propuesta en este capítulo es una solución mejor para sistemas con acceso aleatorio a datos, en los que las transferencias masivas y la localidad de datos no se puede aprovechar.

2.3 Implementación de la Unidad Controlador de Memoria (MCU)

Para abordar el problema de múltiples entidades que requieren acceso a datos compartidos, hemos propuesto una arquitectura con múltiples puertos abstraídos **AAP** para acceder a una memoria externa. Teniendo en cuenta que los diseños sobre **FPGA** son generalmente más lentos que las memorias, es posible construir un esquema de accesos programado mediante el arbitraje de puertos, basados en un controlador de memoria trabajando a una mayor frecuencia que la utilizada por cada **AAP**; por consiguiente, el sistema tendrá dos dominios de reloj, CD_s que define la frecuencia de trabajo de los **AAP** y CD_f que define la frecuencia de trabajo del controlador de la memoria externa. De lo anterior podemos suponer que el acceso a memoria a través de los **AAP** puede hacerse de forma paralela en el dominio de reloj de éstos si la razón entre CD_f y CD_s es mayor o igual que el número de puertos **AAP**.

Para obtener un mejor nivel de abstracción, la **MCU** ha sido equipada con dos tipos de **AAP** diferentes, uno para lectura y otro para escritura. Por lo tanto, la **MCU** consta de un controlador de memoria y varios **AAP** de acuerdo a los requerimientos del sistema. Cada **AAP** trabaja de forma independiente; es la **MCU** la que sincroniza todas las peticiones hechas por los **AAP**, asimismo, garantiza una distribución uniforme del ancho de banda entre los **AAP** activos en tiempo de ejecución. Un diagrama de bloques es presentado en la Figura 2.1, en ella se ilustra la **MCU** con dos puertos, uno para lectura y otro para escritura; nótese que el **AAP** de lectura tiene dos interfaces en contraste con el **AAP** de escritura que sólo utiliza una interfaz.

La arquitectura **MCU** propuesta fue sintetizada e implementada en una **FPGA** *XC4VFX60-10* de Xilinx [37]. Esta **FPGA** se incluye en la plataforma de desarrollo *Xirca-V4* de la empresa *Seven Solutions* [38]. La plataforma utilizada cuenta con 4 memorias **ZBT** de 72 mega bits (2M x 36 bits) totalmente independientes una de la otra en cuanto a las señales de control y al bus de datos y de direcciones; las entradas de reloj son compartidas. Cada memoria **ZBT** tiene su propia unidad de control de memoria **MCU** diseñada en **VHDL**.

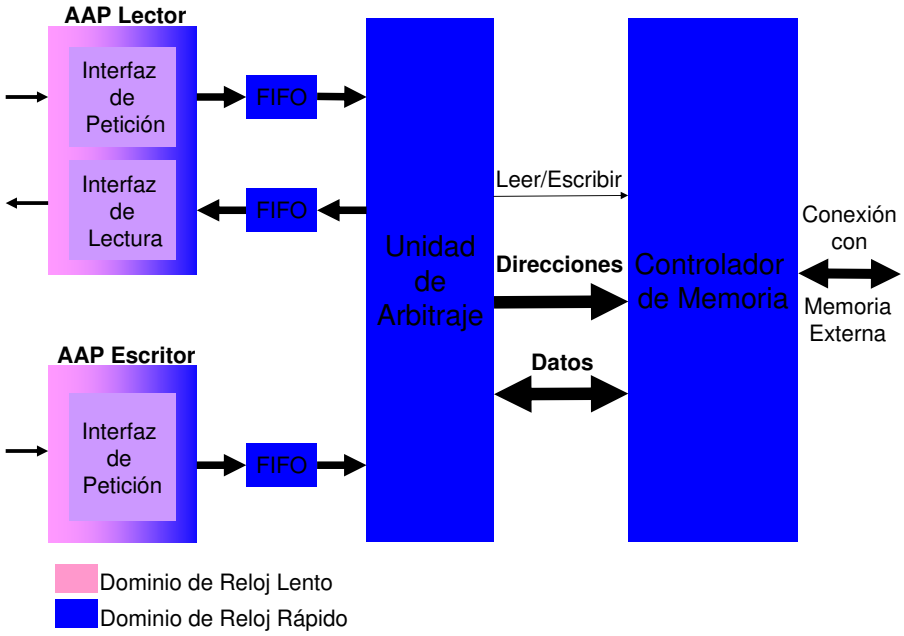


Figura 2.1: Diagrama en bloques de una **MCU** con dos **AAP**.

2.3.1 Diseño de los Puertos de Acceso (AAP)

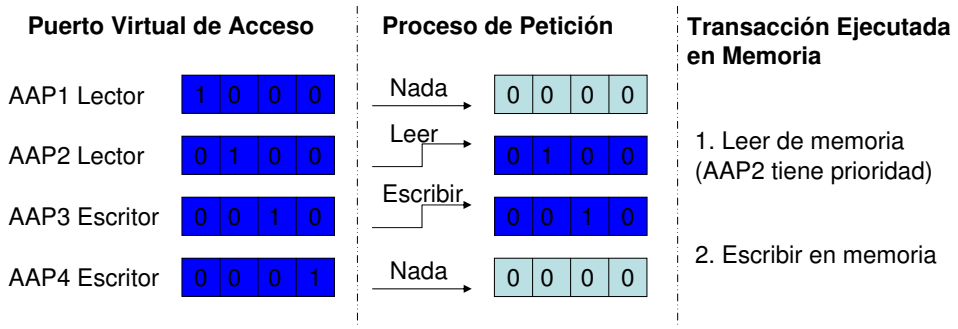
Los **AAP** son bloques críticos en la etapa de diseño debido a su dominio cruzado de reloj. Los datos atraviesan por dos dominios de reloj, el primero es el reloj de la entidad conectada al **AAP** y el segundo es el reloj de la **MCU**. La mejor forma de tratar con dominios cruzados de reloj en una **FPGA** es utilizar una **FIFO**. Por medio de la herramienta *Core Generator*^{TM} de Xilinx [37] es posible definir una **FIFO** optimizada para trabajar en un dominio de reloj cruzado.

Para solucionar el problema de los dominios cruzados de reloj hemos utilizado las **FIFO**s como interfaces de petición en el **AAP**. Estas **FIFO**s ofrecen una fácil sincronización de procesos y al mismo tiempo separan los procesos de petición de las transacciones en memoria. Como resultado, las entidades de cálculo pueden hacer sus peticiones de acceso en procesos diferentes a los usados en efecto para leer o escribir en memoria. Para hacer efectiva esta separación, en el **AAP** de lectura es necesario utilizar una interfaz adicional con el fin de almacenar los datos pedidos. Como se puede ver en la Figura 2.1, el **AAP** de lectura tiene dos interfaces, una para solicitar datos y otra para obtener efectivamente el dato. La sincronización entre el proceso de petición y el

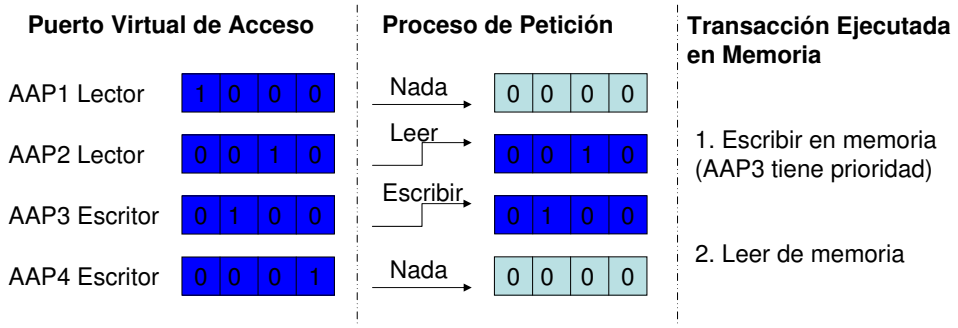
subsecuente proceso de lectura es realizada por las señales de control de la **MCU**, como consecuencia, el diseñador sólo necesita pedir y leer de forma paralela en su diseño.

Debido a que los **AAP** han sido implementados por medio de las **FIFO**s, la manipulación de las interfaces es muy sencilla, cada interfaz de petición tiene una señal que indica si la **FIFO** está llena; cuando esta señal se pone a un valor lógico alto, cualquier petición adicional es rechazada y la entidad de cálculo debe detener su proceso de petición hasta que la señal tenga un valor lógico bajo. Para una petición de escritura, la entidad de cálculo debe poner la dirección y el dato que desea almacenar en memoria; para una petición de lectura, la entidad de cálculo debe poner sólo la dirección donde ha de ejecutarse la transacción. Adicionalmente, la interfaz de lectura del **AAP** de lectura tiene una señal que indica cuando la **FIFO** está vacía. Un valor lógico bajo en esta señal indica que hay datos disponibles y que pueden ser leídos en el mismo orden en el que fueron pedidos a través de la interfaz de petición (para el caso de peticiones consecutivas). De acuerdo a lo mencionado anteriormente, cuando una petición de lectura es realizada, no hay necesidad de esperar a leer el dato para pedir datos adicionales; las interfaces de petición y lectura deben usarse de forma concurrente para conseguir el máximo rendimiento del **AAP**.

Dado que las interfaces son independientes, la ejecución en paralelo de todas las interfaces es recomendada para obtener el máximo rendimiento de la **MCU**; el sistema entero genera muchos ciclos de reloj de latencia, pero aún así es capaz de leer y escribir por ciclo de reloj. La **MCU** no puede garantizar un adecuado tratamiento a los datos cuando una escritura y una lectura sobre la misma dirección de destino es realizada de forma concurrente, es decir, ambas peticiones son tratadas de forma correcta, pero el orden en el que se ejecutan depende del nivel de jerarquía de los **AAP** en el momento de permitir el acceso a la memoria; el nivel de jerarquía de los **AAP** es controlado por la unidad de arbitraje **AU**. La Figura 2.2 muestra de forma esquemática la forma en que es tratada una petición concurrente de lectura y escritura. Leer y escribir la misma dirección en el mismo ciclo de reloj puede hacerse de dos formas diferentes, primero lee y luego escribe de acuerdo a lo mostrado en la Figura 2.2a o primero escribe y luego lee de acuerdo a lo mostrado en la Figura 2.2b; la diferencia radica en el valor Id de los **AAP** en el momento de la ejecución; por consiguiente, el diseñador debe tener cuidado y garantizar la coherencia de los datos en su diseño.



(a) El **AAP** de lectura tiene prioridad



(b) El **AAP** de escritura tiene prioridad

Figura 2.2: Esquema de accesos concurrentes a memoria.

De acuerdo a lo mencionado anteriormente, los **AAP** pueden ser fácilmente inmersos en herramientas de síntesis de alto nivel, como por ejemplo en el lenguaje *SystemC* [39], y ser usados como funciones de acceso a memoria. El lenguaje *SystemC* se está convirtiendo en un estándar industrial para el modelado, diseño y verificación de sistemas. Este lenguaje es cada vez más usado en el modelado de sistemas híbridos que incluyen parte de hardware y parte de software.

2.3.2 Descripción de la unidad de arbitraje (AU)

El propósito de la **AU** es vaciar todas las **FIFO**s usadas como interfaces en el menor tiempo posible en el dominio CD_f. La **AU** está muy relacionada con el tipo de memoria externa que se use en el sistema, es decir, la velocidad para servir una petición está definida por la velocidad de escritura y lectura de la memoria. El objetivo es conseguir el máximo rendimiento posible en sistemas con acceso aleatorio a datos, por esta razón la mejor tecnología que puede ser utilizada es la **ZBT** ya que es capaz de realizar una lectura o una escritura por ciclo de reloj. Para utilizar esta tecnología la máximo de su rendimiento, la **AU** debe tener la capacidad de resolver los accesos a memoria por ciclo de reloj.

Un esquema de división fija del ancho de banda disponible puede ser útil para atender todas las peticiones del sistema, sin embargo, un esquema de este tipo no representa la solución más óptima. En realidad un sistema óptimo debe ser capaz de dividir uniformemente el ancho de banda de la memoria entre todos los **AAP** activos. Este mecanismo será explicado a continuación.

Una colisión virtual es debida a una petición concurrente de dos o más **AAP**. En la **MCU** cada **AAP** tiene un identificador que es usado para definir un nivel de prioridad de acceso. Los identificadores son representados mediante una codificación *One-Hot*[1]. Para una **FPGA**, Sutaone y Badwaik [40] evaluaron cuatro tipos de codificación dentro de sus diseños a saber, *One-Hot*, *Johnson*[2], *Binary* y *Gray*[3]. Ellos concluyeron que la codificación *One-Hot* es la más adecuada para optimizar área y velocidad en el diseño de circuitos digitales. Cassel y Kastensmidt [41] sostienen que la codificación *One-Hot*

[1]Es un tipo de codificación binaria en la que sólo son válidas las combinaciones en las que sólo uno de los bits es "1" y el resto es "0".

[2]Es un tipo de codificación progresiva donde sólo cambia de estado un bit a la vez.

[3]También conocido como codificación reflejo, se caracteriza por cambiar de estado sólo un bit entre estados sucesivos

es la que mejor compensa el consumo de área, el rendimiento y la confiabilidad de los circuitos digitales. Asimismo, Xilinx aconseja el uso de la codificación *One-Hot* para evitar una implementación ineficiente en términos de velocidad y consumo de recursos, según la guía de simulación y diseño [42]. En nuestro diseño, el uso de la codificación *One-Hot* permite reducir la lógica asociada al circuito encargado de resolver las colisiones virtuales.

Cuando un **AAP** hace una petición de acceso, lo hace activando su identificador, de lo contrario su identificador permanece en cero. El valor del identificador representa una petición de acceso y al mismo tiempo identifica al **AAP** que ha hecho la petición. Una única petición de acceso a memoria no activa el mecanismo para la resolución de las colisiones virtuales, pero sí que garantiza el acceso a memoria del **AAP** que realizó la petición. Múltiples peticiones de acceso en forma concurrente, activan el mecanismo para la resolución de colisiones virtuales y genera la secuencia de eventos que se explica a continuación.

Asumamos una **MCU** con cuatro **AAP**, los identificadores en codificación *One-Hot* son: $Id1 = 1000$, $Id2 = 0100$, $Id3 = 0010$ y $Id4 = 0001$, donde $Id1$ tiene la máxima prioridad y así sucesivamente. La Figura 2.3 muestra el procedimiento que es ejecutado por la **AU** cuando una única petición es hecha. Tres estados son mostrados, el estado *Identificadores Iniciales*, el estado *Proceso de Petición* y el estado *Entregando el Acceso a Memoria*. En el estado *Identificadores Iniciales* son exhibidos todos los valores Id. En el estado *Proceso de Petición* el puerto AAP2 hace una petición mostrando su Id, mientras que los demás mantienen sus identificadores en cero. Finalmente en el estado *Entregando el Acceso a Memoria*, se representa el acceso a memoria desde el puerto AAP2 y se resalta que no hubo cambios en los identificadores de ningún **AAP**. Si sólo un **AAP** hace peticiones de acceso a memoria, todo el ancho de banda es entregado a su servicio, sin embargo, la transacción se lleva a cabo en el dominio CD_s.

Múltiples peticiones de acceso hacen necesario el mecanismo de resolución de colisiones virtuales. Este mecanismo garantiza el acceso a memoria al **AAP** con el máximo nivel de prioridad y cambia los identificadores de todos de acuerdo a su precedencia. Esto quiere decir que el **AAP** con el acceso concedido obtendrá la mínima prioridad y los demás incrementarán su orden de prioridad para competir por el siguiente acceso. La Figura 2.4 muestra un diagrama que representa los diferentes estados para la solución de colisiones virtuales. De igual forma que para peticiones únicas, la secuencia de

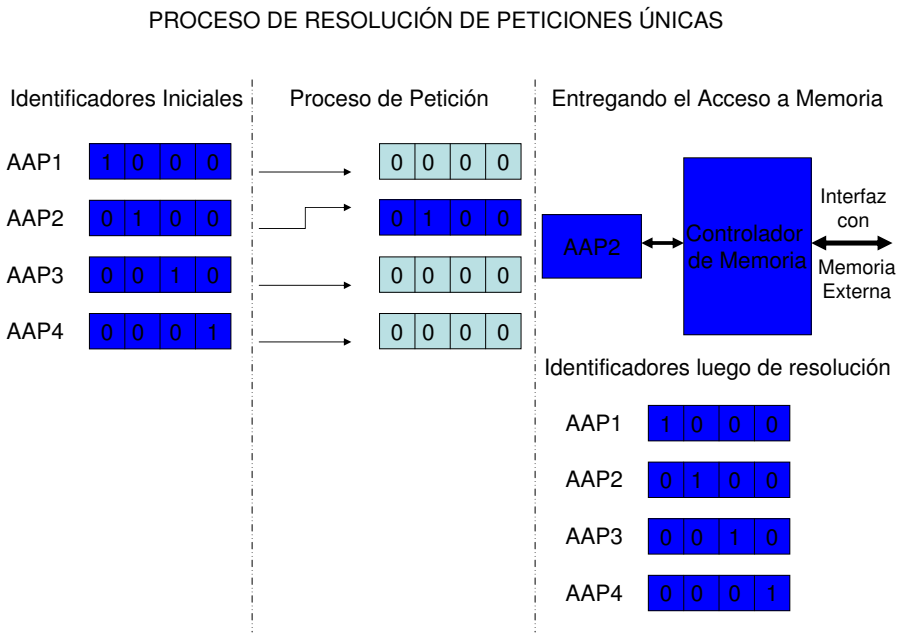


Figura 2.3: Diagrama de acceso a memoria para una única petición.

eventos para definir el acceso a memoria es de tres estados. En la parte izquierda de la figura se muestran todos los valores Id. Si una colisión virtual necesita ser solucionada, la **AU** utiliza estos valores Id como niveles de jerarquía. En la parte media de la figura se muestra que tanto el AAP2 y el AAP3 han hecho una petición de acceso, por eso sus Id son diferentes de cero. Debido a la colisión virtual, en el estado *Entregando el Acceso a Memoria* se modifican todos los valores de identificación; se nota que el AAP2 obtuvo el acceso a memoria por tener la máxima prioridad entre quienes hicieron la petición de acceso, también puede verse que, el Id de AAP2 es modificado y obtiene la mínima prioridad, los Id de AAP3 y AAP4 incrementan su nivel de jerarquía al ser modificados y por último, el AAP1 mantiene su máxima prioridad. Es importante resaltar que las colisiones virtuales son resueltas en cada ciclo de reloj, de esta forma el acceso a memoria se concede por ciclo de reloj.

2.3.3 Diseño del Controlador de Memoria

Escoger la mejor tecnología de memoria para una aplicación específica es una tarea difícil y depende de la plataforma de procesamiento que se utilice. Cuando se utilizan

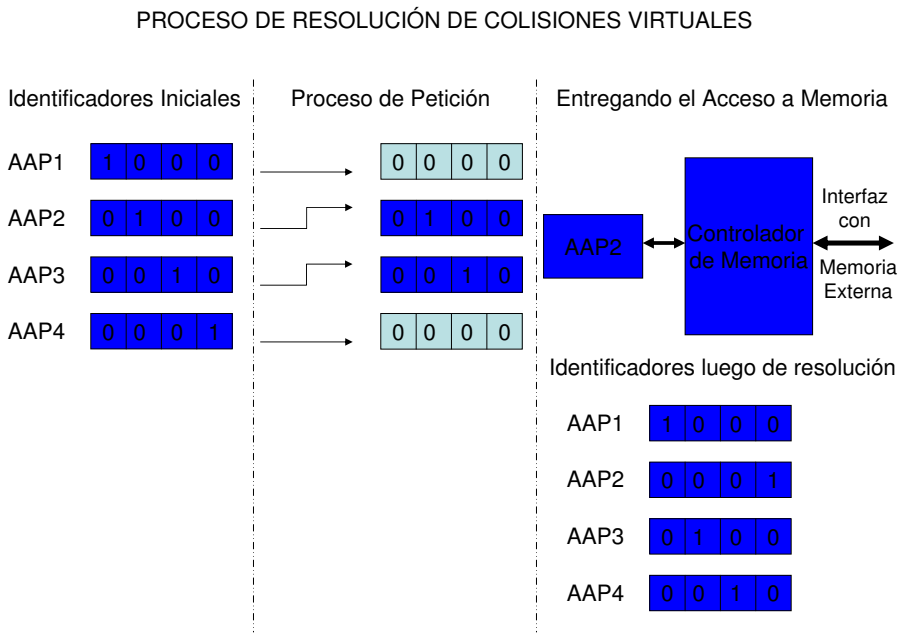


Figura 2.4: Diagrama de acceso a memoria para múltiples peticiones.

las **FPGA**s como plataformas de procesamiento, son ellas las que limitan el rendimiento de los sistemas y, en la mayoría de los casos, los periféricos conectados a éstas son más rápidos; de hecho, los diseño de sistemas complejos que utilizan memorias externas son siempre más lentos que las memorias y por tal motivo no aprovechan al máximo el ancho de banda que éstas ofrecen. Aunque las memorias **ZBT** tienen la desventaja de usar frecuencias más bajas que las memorias **DDR**, siendo incluso de menor capacidad que éstas, ellas cumplen satisfactoriamente con el rendimiento de cualquier sistema complejo diseñado en las **FPGA**s, incluso siguen siendo más rápidas. El objetivo del controlador de memoria propuesto es utilizar las memorias **ZBT** a su máximo rendimiento.

La tecnología **ZBT** es capaz de cambiar del estado de escritura al estado de lectura por ciclo de reloj y viceversa; las **ZBT** están disponibles en dos versiones, la versión *pipelined* puede operar a una mayor frecuencia que la versión *flowthrough* pero esta última tiene una menor latencia de operación. La **MCU** controla una memoria **ZBT** de la versión *flowthrough* que trabaja a 133 MHz. y por tal motivo la **MCU** debe trabajar a 133 MHz. con el fin de alcanzar el máximo rendimiento de la memoria.

Debido a que la **AU** y la memoria **ZBT** pueden realizar sus operaciones por ciclo

de reloj, es necesario que el controlador de memoria pueda realizar su tarea por ciclo de reloj. Como la **ZBT** tiene una interfaz de control sencilla, entonces el controlador de memoria fue diseñado de tal forma que las señales de control se sincronizan con la memoria externa en un camino largo de datos con una latencia grande, pero puede trabajar a 200 MHz. para una **FPGA** *Virtex4* de Xilinx.

La arquitectura de la **MCU** fue diseñada para aceptar cualquier dispositivo de memoria **RAM**. Existen en Internet una gran cantidad de proyectos **VHDL** de acceso libre [43], allí se pueden encontrar una gran variedad de controladores de memoria para muchos tipos de memoria, asimismo, los diferentes proveedores de **FPGA** ofrecen librerías para el control de memorias **RAM**, y es de resaltar que cualquiera de estos controladores puede ser utilizado dentro de la arquitectura de la **MCU**.

2.3.4 Resultados de la Implementación: Recursos y Prestaciones

El porcentaje de ocupación de la **FPGA**, la frecuencia de trabajo de la **MCU** y el ancho de banda disponible en cada **AAP** son todos afectados por el número de puertos que tenga el sistema; asimismo, el tipo de **AAP** (de lectura o escritura) también influye en el resultado final. La Figura 2.5 muestra la evolución de la frecuencia de trabajo de la **MCU** al incrementar el número de puertos **AAP**. Como puede verse, un buen compromiso entre rendimiento y número de **AAP**s se consigue al configurar la **MCU** con 4 **AAP**s, de esta forma la frecuencia de trabajo es muy cercana a la frecuencia de trabajo de la memoria **ZBT** (aproximadamente 125 MHz.). También se puede apreciar que una configuración con más **AAP**s de escritura presenta una leve mejora en la frecuencia de reloj del sistema, sin embargo, estas diferencias no son significativas y el usuario puede escoger la configuración de la **MCU** más conveniente de acuerdo a la aplicación en la que sea usada.

Es importante explicar la forma en que los datos fueron recolectados; si el número de **AAP**s en la **MCU** es par, entonces un número igual de puertos de lectura y escritura fue escogido; si por el contrario, el número de **AAP**s es impar, entonces dos medidas fueron tomadas, una cuando el número de puertos de lectura era mayor y otra cuando el número de puertos de escritura era mayor, estas dos mediadas dan origen a las columnas *Lectores* > *Escritores* y *Lectores* < *Escritores* en las Tablas 2.1, 2.2 y 2.3, también a las series *más Lectores* y *más Escritores* en las Figuras 2.5 y 2.6.

La Tabla 2.1 resume el porcentaje de ocupación de la **FPGA** para las diferentes

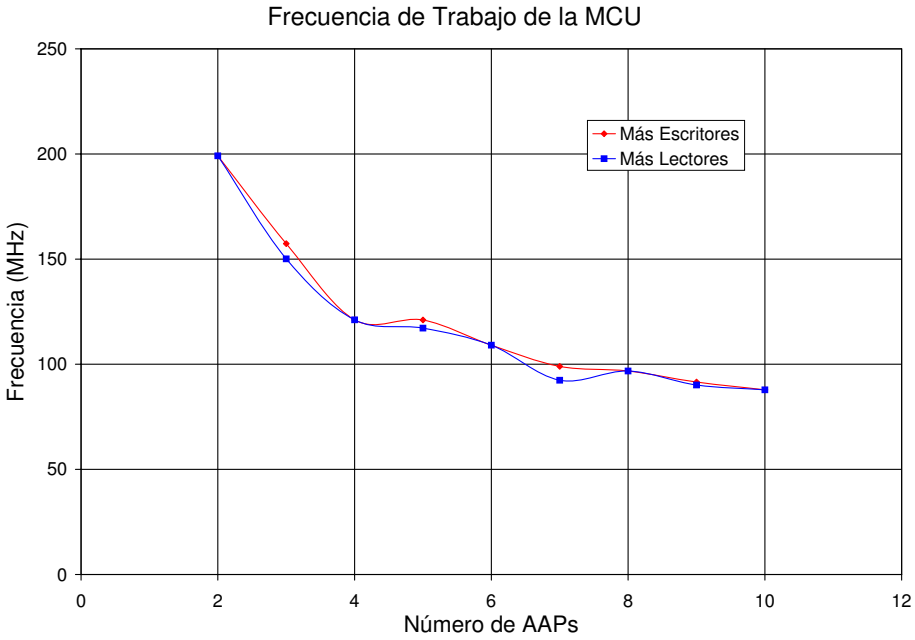


Figura 2.5: Frecuencia de trabajo de la **MCU** para varias configuraciones.

configuraciones en términos del número de **AAP**s que constituyen la **MCU**. Es de anotar que una **MCU** con más **AAP** escritores consume una menor cantidad de recursos hardware que una **MCU** con más **AAP** de lectura, esto es debido a la interfaz adicional que es usada en todos los puertos de lectura; de cualquier modo esta diferencia no es representativa en el rendimiento final de la **MCU**.

La Figura 2.6 se obtiene de los datos contenidos en la Tabla 2.1 y en ella se muestra una tendencia lineal en el uso de los recursos hardware cuando el número de **AAP**s es incrementado en la **MCU**. La **MCU** fue sintetizada y probada en una **FPGA** *Virtex4 XC4VFX60-10* de Xilinx la cual tiene 25.000 *slices*[1].

Los resultados presentados en las Tablas 2.2 y 2.3 son la recopilación de los anchos de banda alcanzados en cada **AAP** cuando se hace uso masivo de todos los **AAP**s disponibles. La **MCU** fue diseñada para distribuir equitativamente el ancho de la memoria externa, sin embargo, su rendimiento se ve afectado por el número de **AAP**s en el sistema y por el número de peticiones concurrentes hechas por ciclo de reloj. El

[1]Es la estructura de cómputo más simple y por media de la cual se mide la cantidad de recursos hardware que tiene una FPGA de la familia Virtex4 de la empresa Xilinx

Tabla 2.1: Consumo de hardware para una **FPGA** *Virtex4 XC4VFX60*.

# AAPs	(# Slices) (Lectores > Escritores)	(# Slices) (Lectores < Escritores)	Ocupación de la **FPGA** (%)
2	300	300	1
3	484	397	1
4	621	621	2
5	836	726	3
6	1.010	1.010	3
7	1.189	1.120	4
8	1.298	1.298	5
9	1.560	1.449	6
10	1.733	1.733	6

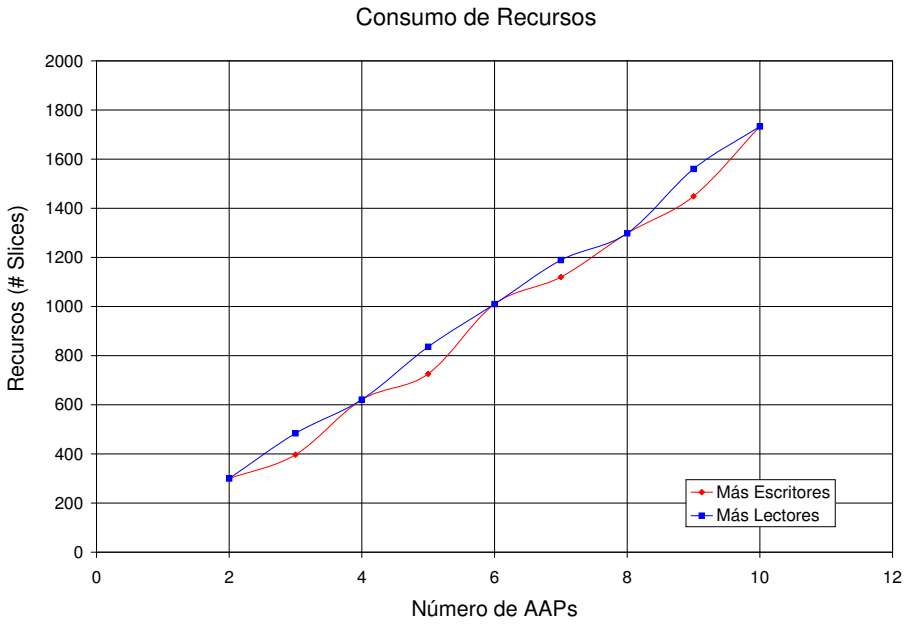


Figura 2.6: Línea de tendencia de ocupación de la **FPGA** de acuerdo al número de **AAP**s.

Tabla 2.2: Ancho de banda por **AAP** para el caso *Lectores* < *Escritores*.

# **AAP**s	Ancho de Banda por **AAP** (Mbps) (*Lectores* < *Escritores*)	Frecuencia de trabajo de la **MCU** (MHz)
2	1.592,93	199,12
3	839,22	157,35
4	484,53	121,13
5	387,38	121,06
6	290,82	109,06
7	226,23	98,98
8	193,54	96,77
9	162,74	91,54
10	140,50	87,81

Tabla 2.3: Ancho de banda por **AAP** para el caso *Lectores* > *Escritores*.

# AAPs	Ancho de Banda por **AAP** (Mbps) (*Lectores* > *Escritores*)	Frecuencia de trabajo de la **MCU** (MHz)
2	1.592,93	199,12
3	800,55	150,10
4	484,53	121,13
5	375,00	117,12
6	290,82	109,06
7	211,16	92,38
8	193,54	96,77
9	160,24	90,14
10	140,50	87,81

ancho de banda en cada **AAP** fue calculado por medio de la ecuación (2.1).

$$f = c\left(\frac{w_f}{n}\right) \tag{2.1}$$

Donde n es el número de **AAP**s en el sistema, w_f es la frecuencia de trabajo de la **MCU**, c es la longitud del bus de datos de la memoria externa y f es el ancho de banda asignado a cada **AAP** en el peor caso para una configuración con n puertos; este ancho de banda no tiene en cuenta la latencia inherente a cada **AAP**. El bus de datos de la memoria **ZBT** usada es de 36 bits de longitud (4 bytes más 1 bit de paridad por byte), sin embargo, para las estimaciones se ha utilizado $c = 32$.

2.4 Ejemplo de Aplicación: Procesamiento de Vídeo

La **MCU** fue construida para ser usada en sistemas empotrados con múltiples unidades de procesamiento intentando acceder a la misma memoria externa y al mismo tiempo. Quizá el área más interesante para mostrar los beneficios de la **MCU** es el procesamiento de vídeo en tiempo real, ya que la cantidad de datos que deben ser procesados es enorme. En particular, los algoritmos más interesantes son aquellos en los cuales el acceso a memoria es dependiente de los datos, es decir, el acceso a un dato en la memoria depende de los datos leídos previamente. Para esta aplicación nos hemos enfocado en el proceso de deformación de imágenes, ya que es ampliamente usado en una gran variedad de algoritmos para el procesamiento de imágenes y vídeo.

Algunos métodos para el cálculo de flujo óptico utilizan una estrategia de control *piramidal* para incrementar el rango de movimiento que puede ser detectado [44, 45] (métodos con aproximación de varias escalas). Esta estrategia de control también es usada en algoritmos para la estimación de la disparidad en sistemas de visión estéreo como el propuesto por Nishihara en [46]. Este esquema *piramidal* requiere la deformación del movimiento y/o la disparidad en cada escala para sintonizar adecuadamente el detector de movimiento y/o disparidad; para realizar este proceso de deformación es necesario un acceso irregular a los datos, es decir, un acceso a datos que no tiene un patrón preestablecido. La precisión en los resultados de la visión estéreo depende de un apropiado esquema de rectificación; el algoritmo para la transformación lineal directa propuesto por Abdel-Aziz et. al [47] y el algoritmo para la estimación de líneas sobre los polos propuesto por Papadimitriou et. al [48] son ampliamente usados en sistemas de visión estéreo para la calibración de las cámaras.

Todos los métodos descritos anteriormente tienen en común el uso del proceso de deformación de imágenes y es por esto que hemos decidido utilizarlo como caso de estudio para evaluar el rendimiento de la **MCU**. El objetivo es demostrar las mejoras asociadas a la utilización de la **MCU** en sistemas que requieren una cantidad grande de accesos aleatorias a memoria; para tal fin, en el apartado 2.4.1 se describe el proceso de deformación basado en tablas de búsqueda **LUT**, en el apartado 2.4.2 se ilustra la forma en que se introduce la **MCU** en el entorno de diseño del lenguaje *Handel-C*, aunque se puede utilizar igualmente con cualquier otro lenguaje de descripción de alto nivel como *SistemC* o *Impulse*, y finalmente en el apartado 2.4.3, se explica la implementación en

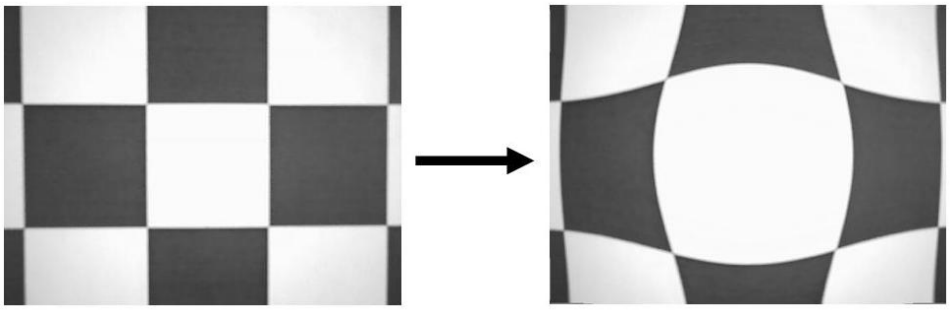


Figura 2.7: Ejemplo ilustrativo de una imagen deformada.

hardware del proceso de deformación y su conexión con la **MCU**, asimismo se hace un análisis del rendimiento del todo el sistema.

2.4.1 Descripción del Algoritmo de Deformación de Imágenes

La deformación de imágenes es el proceso mediante el cual se mueven los píxeles de una imagen de acuerdo a una transformación geométrica del sistema de coordenadas original [49]. Dado que este proceso requiere un cálculo a nivel de sub-píxel, hemos utilizado la interpolación bilineal para realizar el cómputo de la deformación. La Figura 2.7 muestra un ejemplo ilustrativo de una imagen deformada, esta imagen fue obtenida de una entrada de vídeo procesada por medio del sistema propuesto con la deformación de ojo de pez contenida en las **LUT**s.

En el ejemplo de aplicación, el efecto de ojo de pez fue previamente calculado y su transformación geométrica fue almacenada en las **LUT**s. La interpolación bilineal es una función que puede implementarse fácilmente en hardware. Esta función necesita leer cuatro píxeles de la imagen original para calcular un píxel de la imagen procesada. Las ecuaciones que definen la interpolación bilineal son basadas en la interpolación lineal en la dirección x y luego en la dirección y de forma separada. Para encontrar el valor desconocido de la función f en el punto $P = (x, y)$, si se asume que se conocen los puntos $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$ y $Q_{22} = (x_2, y_2)$ sobre una cuadrícula y los correspondientes valores de la función en esos puntos $f(Q_{11})$, $f(Q_{12})$, $f(Q_{21})$ y $f(Q_{22})$, entonces es posible interpolar en la dirección x utilizando las ecuaciones (2.2) y (2.3) para encontrar $f(x, y_1)$ y $f(x, y_2)$:

$$f(R_1) \approx \left(\frac{x_2 - x}{x_2 - x_1}\right) f(Q_{11}) + \left(\frac{x - x_1}{x_2 - x_1}\right) f(Q_{21}), \tag{2.2}$$

donde $R_1 = (x, y_1)$.

$$f(R_2) \approx \left(\frac{x_2 - x}{x_2 - x_1}\right) f(Q_{12}) + \left(\frac{x - x_1}{x_2 - x_1}\right) f(Q_{22}), \tag{2.3}$$

donde $R_2 = (x, y_2)$.
Luego, los resultados de las ecuaciones (2.2) y (2.3) son usados para calcular la interpolación en la dirección y de acuerdo a la ecuación (2.4).

$$f(P) \approx \left(\frac{y_2 - y}{y_2 - y_1}\right) f(R_1) + \left(\frac{y - y_1}{y_2 - y_1}\right) f(R_2) \tag{2.4}$$

Las ecuaciones previas permiten la estimación de $f(x, y)$. Como la imagen tiene una cuadrícula definida a partir de números enteros y la vecindad del punto $P = (x, y)$ se define como $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$ y $Q_{22} = (x_2, y_2)$ tal que $x_2 - x_1 = 1$ y $y_2 - y_1 = 1$, entonces la ecuación (2.4) puede simplificarse escogiendo un sistema de coordenadas adecuado en el cual los cuatro puntos donde f es conocida son $Q_{11} = (0, 1)$, $Q_{12} = (0, 1)$, $Q_{21} = (1, 0)$ y $Q_{22} = (1, 1)$. Esto se puede escribir matricialmente como se indica en la ecuación (2.5).

$$f(x, y) \approx [(1 - x) \;\; x] \begin{bmatrix} f(0,0) & f(0,1) \\ f(0,1) & f(1,1) \end{bmatrix} \begin{bmatrix} 1 - y \\ y \end{bmatrix} \tag{2.5}$$

De acuerdo al nuevo sistema coordenado, x y y están en el intervalo cerrado [0,1] ya que en el sistema de coordenadas original se tiene $|x - x_1| \leq 1$, $|x - x_2| \leq 1$, $|y - y_1| \leq 1$ y $|y - y_2| \leq 1$.

La ecuación (2.5) muestra una operación aritmética de punto fijo que puede ser implementada fácilmente en hardware ya que sólo requiere sumas y multiplicaciones. En un sistema basado en **LUT**s, el proceso de deformación es realizado a partir de dos matrices de información, la matriz x que contiene la posición en la dirección x y la matriz y que contiene la posición en la dirección y de la imagen deformada; con esta información se ejecuta la interpolación bilineal para obtener cada píxel de la imagen deformada.

Se puede describir el proceso de deformación como una ejecución recurrente en la que se utiliza la ecuación (2.5) para estimar cada píxel de la imagen deformada. La combinación de los elementos de las matrices x y y definen los puntos $P(x, y)$ en los que serán estimadas las deformaciones. En resumen, el punto $P(x, y)$ entrega los coeficientes para la interpolación bilineal y la posición en la que el sistema coordenado

está localizado. Cada dato contenido en las **LUT**s tiene una longitud de 12 bits, 8 bits de parte entera (ubicación del sistema coordenado) y 4 bits de parte fraccionaria (coeficientes para la interpolación).

2.4.2 Librerías para el Entorno *Handel-C*

Normalmente los diseños hechos utilizando lenguajes de descripción de alto nivel son más lentos que aquellos hecho utilizando **VHDL** o **VERILOG** [15], haciéndose más evidente cuando se abordan diseños complejos, sin embargo, el tiempo que toma el desarrollo del sistema es mucho menor. Con el fin de utilizar la **MCU** en el entorno *Handel-C* [50], se propone encapsular el diseño hecho en *Handel-C* en una caja negra dentro del diseño hecho en **VHDL**, en este sentido, los *netlists*[1] que se obtienen del diseño en *Handel-C* serán utilizados como módulos interiores del sistema completo hecho por medio del **VHDL**; estos módulo interiores son conectados a través de interfaces específicas que satisfacen las restricciones de tiempo del sistema completo. Por otro lado, en el entorno de *Handel-C* se crea una librería que permita la utilización de las interfaces que definen los **AAP**s.

La **MCU** necesita definir tres interfaces en *Handel-C* con el fin de encapsular las tres interfaces posibles en los **AAP**s (ver Figura 2.1), por consiguiente, tres macro procesos son descritos en *Handel-C* de la siguiente forma:

macro proc RequestData (*PtrInterface*, *addr*, *ChanNum*, *bank*);
macro proc ReadFromRAM (*PtrInterface*, *data*, *ChanNum*, *bank*);
macro proc WriteToRAM (*PtrInterface*, *addr*, *data*, *ChanNum*, *bank*);

donde *PtrInterface* es la conexión existente entre el diseño en *Handel-C* y el diseño en **VHDL**, *addr* es la dirección de memoria, *data* puede ser el dato de lectura o el dato de escritura, *ChanNum* es el número del puerto **AAP** que se va a utilizar y *bank* es el número de la memoria **ZBT** externa a la cual está conectada la **MCU**. Estos macro procesos son similares a las funciones en el lenguaje **C** con la pequeña diferencia que cuando se usan varias en paralelo el hardware se replica; ésto quiere decir que si se usan dos instancias de la función*RequestData* en paralelo, entonces será necesario usar dos **AAP**s diferentes tal como se muestra en el siguiente ejemplo en el lenguaje *Handel-C*:

[1] Archivo que describe la conexión de un sistema electrónico.

```
par{
    RequestData (PtrInterface, addr1, 0, bank0);
    RequestData (PtrInterface, addr2, 1, bank0);
    ReadFromRAM (PtrInterface, data1, 0, bank0);
    ReadFromRAM (PtrInterface, data2, 1, bank0);
}
```

Es de anotar que el uso de la sentencia *par* en *Handel-C* implica que todas las funciones deben ejecutarse en paralelo, sin embargo las funciones declaradas en *Handel-C* no necesitan una sincronización explícita, el diseñador sólo debe tener en cuenta el número de **AAP**s disponibles y usarlos de acuerdo a la configuración de la **MCU**. Adicionalmente, debido a que la **MCU** trabaja con un reloj de hasta 120 MHz. para una configuración con 4 **AAP**s (ver 2.5) y que la frecuencia de los diseños en *Handel-C* no supera los 80 MHz. en la mayoría de los casos, entonces el sistema es capaz de ejecutar las dos funciones de forma paralela en el dominio de reloj del diseño en *Handel-C*, tal como fue explicado en el apartado 2.3. A través de estos tres macro procesos es posible leer o escribir en la memoria externa de forma concurrente en *Handel-C* y a la vez aprovechar el alto rendimiento que ofrecen los diseños hechos en **VHDL**.

2.4.3 Aproximación a la Implementación del Sistema

Para el cálculo de cada píxel deformado es necesario leer desde las matrices x y y las coordenadas (x, y) que corresponden a cada punto P. La parte entera de (x, y) es usada para obtener de la memoria los cuatro píxeles de la imagen original que conforman la vecindad del punto P y la parte fraccionaria de (x, y) es usada para calcular la deformación del píxel tal como se plantea en la ecuación (2.5).

El proceso de deformación necesita hacer seis accesos a memoria por ciclo de reloj para calcular un píxel deformado y obtener el máximo rendimiento. De forma demostrativa se han diseñado dos tipos de arquitectura diferente, en la primera se ha utilizado una **MCU** con 4 **AAP**s con el fin de aprovechar la ventaja del máximo rendimiento de la memoria **ZBT**, ver Figura 2.5, y la segunda utiliza una **MCU** con 5 puertos con el fin de aumentar el acceso a datos por ciclo de reloj; ambos diseños fueron hechos por medio de estructuras segmentadas. La configuración de 4 puertos utiliza diferentes **AAP** de lectura para las matrices x y y y para la imagen original, el puerto restante

es un puerto de escritura que es utilizado para almacenar la imagen deformada. En la configuración de 5 puertos, la imagen original es leída a través de dos **AAP**s diferentes, uno para leer los píxeles Q_{11} y Q_{12} y el otro para leer los píxeles Q_{21} y Q_{22}, los demás puertos se utilizan de igual forma que en el caso anterior.

La **MCU** maneja datos con una longitud de 36 bits, por lo que cuatro píxeles pueden ser leídos en un sólo acceso a memoria para el caso de datos de la imagen original; para el caso de las matrices x y y, sólo tres datos son leídos por acceso a memoria. Teniendo en cuenta lo anterior, es evidente que en el mejor caso un acceso a memoria entrega dos píxeles Q_{11} y Q_{12} o Q_{21} y Q_{22}, sin embargo, los píxeles Q_{11}, Q_{12}, Q_{21} y Q_{22} están en palabras de memoria diferentes cada cuatro accesos a direcciones consecutivas de la imagen original. Esta restricción permite calcular 4 píxeles de la imagen deformada por cada 10 accesos a memoria para el sistema con 4 **AAP**s y 4 píxeles de la imagen deformada por cada 5 accesos a memoria para el sistema con 5 **AAP**s. El acceso sobre las matrices x y y es totalmente regular y no impone ninguna restricción adicional al sistema.

La Tabla 2.4 muestra el funcionamiento de las diferentes arquitecturas en un sistema que realiza la distorsión de ojo de pez en tiempo real para imágenes con una resolución de 640x480. El sistema fue probado en un computador de escritorio que tiene un procesador *Intel Core 2 CPU 6600 @ 2.4 GHz.* y 2 giga bytes de memoria **RAM** y una cámara web *Philips SPC1300NC*. La plataforma *Xirca-V4* es conectada al procesador por medio de una interfaz **PCIe**, esta plataforma ha sido desarrollada para la evaluación de sistemas de procesamiento de imágenes implementados en **FPGA**s [51]. Los datos fueron recolectados durante 10 minutos de ejecución de la distorsión ojo de pez, el resultado promedio fue calculado a partir de 200 datos. Los resultados para la arquitectura secuencial se obtuvieron a partir del mismo sistema pero usando sólo un **AAp**.

Como se muestra en la Tabla 2.4, la utilización de la **MCU** mejora significativamente el rendimiento de sistemas de procesamiento paralelo con accesos masivos a memoria y simplifica la descripción de los mismos al permitir el uso de la **MCU** en entornos de descripción de alto nivel como el *Handel-C*; estas mejoras en el rendimiento se deben principalmente a dos razones:

- La descripción de la **MCU** usando **VHDL** permite velocidades de reloj más altas

Tabla 2.4: Funcionamiento de las diferentes configuraciones de la **MCU**.

	Promedio (fps)	Varianza
MCU 125 MHz 5 AAP	48,5	0,5054
MCU 100 MHz 5 AAP	47,7	0,6724
MCU 125 MHz 4 AAP	45,7	0,6415
Secuencial	32,6	0,7105

que las conseguidas por medio de lenguajes de descripción de hardware de alto nivel de abstracción.

- Una descripción de alto nivel al usar una interfaz de memoria secuencial requerirá tantos ciclos de reloj para hacer su procesamiento como datos necesarios para realizar el cálculo. La utilización de la **MCU** permite utilizar varios puertos de conexión con la memoria y de esta forma se pueden realizar accesos en paralelos a memoria en el dominio de reloj en *Handel-C*; por ejemplo en la aplicación anterior, una interfaz secuencial requeriría 5 ciclos de reloj de una frecuencia de 50 MHz. para calcular un píxel de la imagen deformada, sin embargo, al utilizar la **MCU** con un reloj de frecuencia 125 MHz. con 4 **AAP**s (3 de lectura y 1 de escritura) es posible calcular un píxel de la imagen deformada usando sólo 2 ciclos del reloj de 50 MHz., esto representa un factor de mejora de $5/2 = 2.5$ y justifica la utilización de la **MCU**.

Es de resaltar que la utilización de los puertos **AAP**, más allá de incrementar el rendimiento, ayudan a reducir el tiempo de diseño, pues los diseñadores no necesitan tratar con complejos esquemas de acceso a memoria.

3

Métodos para Estimación y Análisis de Movimiento en Secuencias de Imágenes

Para diseñar un sistema de procesamiento de vídeo en tiempo real es necesario identificar los métodos matemáticos que han sido desarrollados y estudiar la viabilidad de su implementación en hardware. Este capítulo muestra un estudio detallado de los métodos matemáticos más representativos en la literatura y de su aplicabilidad en hardware. Asimismo, una vez determinado qué métodos son plausibles de ser implementados en hardware, se describen las simplificaciones realizadas sobre los mismos para lograr su máximo rendimiento en hardware. Finalmente, se muestran las pruebas de ejecución de los métodos con sus respectivas simplificaciones y se escoge el método que mejores resultados, en términos de velocidad de ejecución y precisión, ha entregado.

3.1 Introducción

El objetivo es desarrollar un **SoC** para el análisis de flujo óptico por medio del cual se puedan estimar características de movimiento tales como el *movimiento propio* y los **IMO**s, asimismo utilizar esta información para filtrar zonas de interés en las imágenes. De este objetivo se desprenden dos tareas a realizar, por una parte es necesario estimar el flujo óptico y por la otra analizarlo. El problema de la estimación de flujo óptico ha sido ampliamente estudiado y muchos métodos han sido propuestos, Horn y Schunck [52] propusieron uno de los primeros métodos que se basó en un patrón espacio-temporal de la intensidad de una imagen, a partir de este trabajo muchos otros fueron propuestos y su principal objetivo ha sido obtener un flujo óptico denso y con un error muy bajo. En un trabajo posterior Barron et al. [53] evaluaron 9 métodos para la estimación de flujo óptico y concluyeron que los métodos basados en la fase obtenían mejores resultados que aquellos basados en la intensidad, asimismo concluyeron que aquellos que presentaron peores resultados podían incrementar su precisión mediante un método basado en una estimación de varias escalas por medio del cual es posible incrementar el rango de estimación del movimiento. En el apartado 3.2 se hace una pequeña descripción del modelo de flujo óptico utilizado, pero un estudio detallado al respecto de la problemática del flujo óptico ha sido realizado por Tomasi [1].

Paralelo al problema de estimar el flujo óptico, está el problema de analizar éste para obtener las características de movimiento que subyacen dentro de estos campos de flujo óptico, Bruss y Horn [11] introdujeron uno de los primeros modelos matemáticos para la estimación de *movimiento propio* habiendo estudiado a fondo la geometría de la estructura de movimiento; de su trabajo se deriva una ecuación que actualmente se conoce como la *restricción bilineal* y sobre ésta ellos definen un sencillo problema de minimización que resuelven por medio de técnicas de optimización no lineal. La gran mayoría de algoritmos propuestos para la estimación de *movimiento propio* basados en flujo óptico, plantean modelos lineales y no lineales para la minimización de la *restricción bilineal*. Heeger y Jepson [54] construyeron un subespacio en el que eliminan la dependencia de la *restricción bilineal* con respecto a la rotación y a la profundidad, de esta forma evitan la estimación de estas variables durante la estimación de la variable de traslación. Los modelos planteados por Bruss y Horn y por Heeger y Jepson para

la minimización de la *restricción bilineal* son altamente sensibles al ruido de las estimaciones de flujo óptico y por tal motivo son conocidos como modelos "no óptimos".

En un escenario más complejo donde, además del ruido inherente de las estimaciones de flujo óptico, haya presencia de **IMO**s, los modelos "no óptimos" son poco robustos. En la literatura podemos encontrar una serie de métodos que pueden ser considerados como "óptimos" debido a su significativa inmunidad al ruido, su robustez ante mínimos locales y a la mínima varianza en las estimaciones de *movimiento propio* en presencia de **IMO**s. Chiuso et al. [55] argumentó que el *movimiento propio* no sólo puede ser modelado geométricamente, sino que también debe modelarse el comportamiento del ruido con el fin de obtener resultados que puedan ser utilizables en aplicaciones reales; al igual que Bruss y Horn, Chiuso et al. definieron un problema de minimización, pero a diferencia de éstos utilizaron una proyección esférica en vez del modelo *pin-hole*[1]; el problema de minimización lo denominaron *mínimos cuadrados esféricos* y se basa en una estimación iterativa de la velocidad de traslación, seguida de la estimación de la velocidad de rotación y la profundidad, y así sucesivamente.

Zhang y Tomasi [56] utilizaron el algoritmo de Gauss-Newton para obtener iterativamente las estimaciones de rotación, profundidad y traslación sobre la *restricción bilineal*. Pauwels y Van Hulle [57] utilizaron el mismo algoritmo que Zhang y Tomasi para ilustrar su método denominado *gradually-unweight*; este método se basa en ajustar gradualmente la relevancia de cada vector de flujo óptico al momento de aplicar el algoritmo de Gauss-Newton, esto se verá con más detalle en el apartado 3.3. Pauwels y Van Hulle argumentan que su método es mucho más robusto contra mínimos locales que los métodos propuestos por Bruss y Horn, Heeger y Jepson, Zhang y Tomasi, ellos demuestran que el parámetro de regularización que ellos proponen permite identificar adecuadamente los vectores de flujo óptico adecuados y en ese sentido su algoritmo tiene una varianza en la estimación de *movimiento propio* acotada en presencia de **IMO**s presentes en la secuencia de fotogramas.

Los modelos denominados "óptimos" son algoritmos recurrentes que necesitan ejecutarse una gran cantidad de veces con valores iniciales diferentes a fin de encontrar la convergencia; esto requiere de una capacidad de cómputo muy alta que, actualmente,

[1]Es un modelo de cámara comúnmente utilizado que describe la relación matemática entre un sistema de coordenadas tridimensional y su proyección en un plano a través de una abertura (o *pin-hole*); la luz pasa a través de un punto sin necesidad de un lente que enfoque la luz.

sólo puede ser alcanzada por medio de **GPU**s (o *clusters*[1]) en aplicaciones de tiempo real, sin embargo, las **GPU** no pueden ser utilizadas como plataformas independientes. Con el fin de poder obtener un **SoC** que pueda estimar el *movimiento propio* en tiempo real, es necesario encontrar un algoritmo por medio del cual sea posible hacer las estimaciones en un número restringido de iteraciones y que su resultado tenga un buen compromiso entre eficiencia y precisión.

De acuerdo a la anterior premisa, hemos centrado nuestro sistema en dos modelos ampliamente conocidos, el modelo de Pauwels y Van Hulle [57] antes mencionado y un modelo alternativo propuesto por Raudies y Neumann [58] que, a diferencia de Pauwels y Van Hulle, trata el problema de la minimización de la *restricción bilineal* de forma lineal y por tanto no requiere de diferentes inicializaciones para alcanzar resultados óptimos. Con el fin de explicar el modelo del sistema de visión, este capítulo se divide en cuatro apartados, el apartado 3.2 en el cual se describe, de forma breve, el algoritmo para el cálculo de flujo óptico y las respectivas modificaciones hechas para su adecuada implementación en hardware, el apartado 3.3 en el cual se explican, de forma detallada, los dos métodos para la estimación de *movimiento propio*, el apartado 3.4 en el que se evalúa el desempeño de éstos con el fin de determinar las modificaciones necesarias para lograr un sistema en tiempo real y finalmente en el apartado 3.5 se hace una aproximación muy simple para la detección de **IMO**s con el sistema de visión propuesto.

3.2 Algoritmo para el Cálculo de Flujo Óptico

Fleet y Jepson [59] mostraron que la evolución temporal de los contornos de fase constante permiten una mejor aproximación a la velocidad local que los contornos de amplitudes constantes. También demostraron que los contornos de la fase son más robustos con respecto a sombras suaves y variaciones de iluminación, asimismo más estables con respecto a pequeñas desviaciones en la traslación [44, 60]. Fleet y Jepson propusieron un rastreo de los contornos de fase constante a través del cálculo del gradiente espacio-temporal de la fase en las imágenes; este cálculo genera singularidades en las estimaciones que fueron tratadas por medio de una sencilla ecuación de restricción. Gautama y Van Hulle [61] propusieron una variación al modelo introducido por Fleet

[1]Grupo de computadores conectados que trabajan en paralelo una misma tarea.

y Jepson en la cual definen una estimación del gradiente espacial, en vez del gradiente espacio-temporal, utilizando filtros en cuadratura; de esta forma es posible transformar las singularidades en medidas no lineales que, por medio de una medida de confianza, pueden ser utilizadas para desechar estimaciones poco fiables.

En el algoritmo original propuesto por Gautama y Van Hulle se calculan 8 orientaciones espaciales y se usa un esquema de resolución múltiple en el cual, una red neuronal combina la información de cada orientación espacial en cada nivel de resolución para incrementar el rango de movimiento que puede ser detectado. En un trabajo posterior, Pauwels y Van Hulle [45] modificaron el esquema de resolución múltiple del modelo propuesto por Gautama y Van Hulle y utilizaron la estrategia de control *piramidal* introducida por Fleet y Jepson [44], asimismo propusieron un método para la estabilización de las imágenes que les permitió una mejor precisión en las estimaciones de flujo óptico. El modelo que ha sido escogido para la estimación de flujo óptico, en el marco de esta tesis, es un simplificación del modelo propuesto por Pauwels y Van Hulle [45].

Con el fin de ahorrar recursos hardware de la **FPGA** se ha eliminado la estrategia de control *piramidal*, por lo que se puede pensar que nuestro modelo es la estimación de flujo óptico en una sola escala. Tal como se muestra en la Figura 3.1, la arquitectura hardware para la estimación de flujo óptico tiene diferentes estados de procesamiento que han sido desarrollados a través de caminos de datos segmentados. Estos caminos de procesamiento comienzan con el proceso de convolución en el cual 8 filtros de Gabor en cuadratura, sintonizados a diferentes orientaciones de fase, pueden ejecutarse en paralelo. Luego, la salida del estado anterior pasa a través de un camino de datos en el cual se estima la fase para cada orientación (esta etapa incluye el cálculo de la función arco tangente). A continuación, en la siguiente etapa, se ajustan los valores de fase considerando su periodicidad en las estimaciones de la derivada temporal, tres fotogramas son necesarios. En la penúltima etapa es estimada la componente de velocidad (o flujo óptico); esta componente de velocidad se calcula a partir de la fase ϕ en cada orientación θ como es indicado en la ecuación (3.1).

$$V_{C,\theta}(x) = \frac{-\phi_{t,\theta}(x)}{2\pi(f_{x,\theta}^2 + f_{y,\theta}^2)}(f_{x,\theta}, f_{y,\theta}) \tag{3.1}$$

Finalmente, para cada componente de velocidad estimada se calcula un valor de confianza que, de superar un umbral, valida la estimación. Las estimaciones válidas en cada

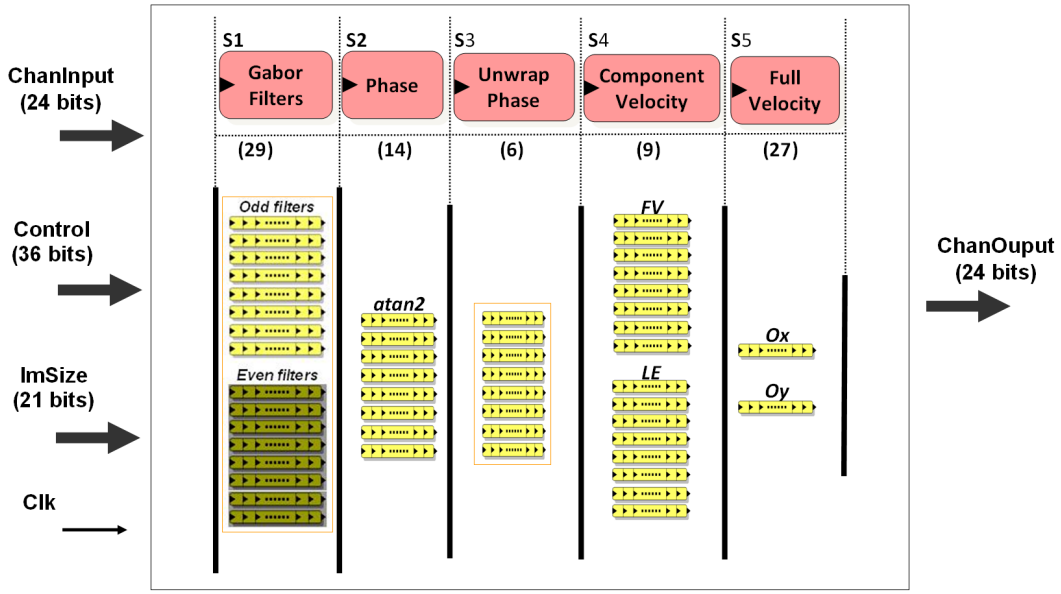


Figura 3.1: Diagrama en bloques de la entidad para el cálculo de flujo óptico. Figura tomada de la tesis doctoral de Tomasi [1].

orientación se combinan para obtener el vector de velocidad final tal como se muestra en la ecuación (3.2).

$$V^*(x) = \arg\min_{V(x)} \sum_{\theta \in O(x)} \left(\|V_{C,\theta}(x)\| - V(x)^T \frac{V_{C,\theta}(x)}{\|V_{C,\theta}(x)\|} \right)^2 \tag{3.2}$$

Un valor de confianza sobre la estimación final del flujo óptico es calculado y se utiliza para decidir si se escribe el valor de flujo óptico estimado o si se pone una etiqueta que significa que ese píxel en particular no tiene una medición válida. Operaciones adicionales de regularización, no mostradas en la Figura 3.1, son hechas a las estimaciones de flujo óptico; estas operaciones son filtros de mediana sencillos aplicados en ventanas de 3x3 que permiten reducir el número de valores falsos que se producen típicamente por datos ruidosos o regiones de oclusión.

El diseño e implementación de esta entidad de procesamiento fue realizado en el marco del proyecto **DRIVSCO**, para más detalles véase: [1, 62, 63, 64].

3.3 Algoritmo para el Análisis de Flujo Óptico: Estimación de *Movimiento Propio*

Longuet-Higgins y Prazdny [65] demostraron que un observador puede determinar la estructura de una escena rígida y su dirección de movimiento relativa a ésta (*movimiento propio*) a partir de la medida de la velocidad instantánea de la retina. Su modelo se basa en el modelo de cámara *pin-hole* con movimiento arbitrario en un ambiente estático. La cámara tiene una longitud focal f y proyecta puntos del espacio (X, Y, Z) a un plano. Si se asume una proyección en perspectiva, el sistema de coordenadas (x, y) del punto P en el plano de la imagen se define como:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z} \tag{3.3}$$

El movimiento en cualquier instante puede ser descrito como la contribución de dos componentes: la velocidad de traslación del *pin-hole* relativa a la escena y la velocidad angular del hemisferio alrededor del *pin-hole* con respecto de la escena; así, el *movimiento propio* genera un desplazamiento instantáneo que puede escribirse como:

$$(\dot{X}, \dot{Y}, \dot{Z})^T = -(t_x, t_y, t_z)^T - (w_x, w_y, w_z)^T \times (X, Y, Z)^T \tag{3.4}$$

donde ^{T} denota la transposición del vector, los puntos definen la primera derivada temporal, $T = (t_x, t_y, t_z)$ es el vector de velocidad de traslación y $W = (w_x, w_y, w_z)$ es el vector de velocidad angular.

El flujo óptico en cada punto del plano de la imagen es la velocidad instantánea del patrón de percepción del ojo en cada punto; por lo tanto, el flujo óptico $U = (u, v)$ se puede obtener al derivar en el tiempo la ecuación (3.3) como sigue:

$$u = \dot{x} = f\frac{\dot{X}}{Z} - f\frac{X\dot{Z}}{Z^2}, \qquad v = \dot{y} = f\frac{\dot{Y}}{Z} - f\frac{Y\dot{Z}}{Z^2} \tag{3.5}$$

Substituyendo el resultado de la ecuación (3.4) en la ecuación (3.5) se obtiene la siguiente expresión:

$$U^T = \frac{1}{Z}AT^T + BW^T \tag{3.6}$$

donde

$$A = \begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix} \tag{3.7}$$

$$B = \begin{bmatrix} xy/f & -(x^2+f^2)/f & y \\ (y^2+f^2)/f & -xy/f & -x \end{bmatrix} \tag{3.8}$$

La matriz A mostrada en la ecuación (3.7), proyecta el vector de velocidad espacial en un plano ortogonal a los ejes de la óptica de la cámara y el parámetro Z de la ecuación (3.6), define la profundidad de la escena. Dado que T y Z aparecen relacionados proporcionalmente, entonces no es posible determinar sus magnitudes absolutas. Bruss y Horn [11] utilizaron el término $1/Z$ como una función de restricción de forma tal que primero determinan qué valor de Z minimiza la formulación de mínimos cuadrados y luego utilizan este resultado para escribir la bien conocida ecuación de *restricción bilineal* en traslación y rotación (3.9).

$$\frac{1}{\|AT^T\|}(AT^T)^T(U - BW^T) = 0 \tag{3.9}$$

Pauwels y Van Hulle [57] reescribieron la ecuación (3.9). Originalmente la *restricción bilineal* tenía una función de peso $\left\|AT^T\right\|$ que impedía considerar de igual forma a todos los vectores de flujo óptico; para solucionar este problema ellos propusieron un método que asigna un peso específico a cada vector de flujo óptico utilizado en la formulación de mínimos cuadrados. Los pesos están definidos como $\frac{1}{\|A\hat{T}^T\|}$ donde $\hat{T}$ es el vector de velocidad de traslación obtenido en una iteración anterior. Asimismo, para suprimir sesgo sistemático ellos proponen, sin perder generalidad en el método, utilizar la *restricción bilineal* en ausencia de la componente de rotación tal como se muestra en la ecuación (3.10).

$$\hat{T}(X_i \times U) = 0 \tag{3.10}$$

Una vez que ha sido obtenida la estimación de la traslación $\hat{T}$, se utiliza para la estimación de los pesos y de la rotación por medio de la ecuación (3.11). Debido a que este algoritmo es muy sensible a los vectores $\hat{T}$ de inicialización, es necesario realizar una gran cantidad de ejecuciones con inicializaciones diferentes hasta obtener la solución óptima, esto tiene un costo computacional muy alto.

$$((\hat{T} \times X_i) \times X_i)^T W - \hat{T}^T(X_i \times U) = 0 \tag{3.11}$$

Raudies y Neumann [58] han introducido recientemente un método lineal preciso que requiere una carga computacional menor que la necesaria para ejecutar el algoritmo propuesto por Pauwels y Van Hulle. Ellos ha propuesto un subespacio lineal a partir

de la *restricción bilineal*; este método es de nuevo un problema de optimización en el cual se obtienen de forma iterativa el vector de traslación seguido por la estimación del vector de rotación. La transformación algebraica propuesta sobre la *restricción bilineal* es:

$$T^T(M - HW^T) = 0 \tag{3.12}$$

donde

$$M = \begin{bmatrix} fv \\ -fu \\ yu - xv \end{bmatrix} \tag{3.13}$$

y

$$H = \begin{bmatrix} -(f^2+y^2) & xy & fx \\ xy & -(f^2+x^2) & fy \\ fx & fy & -(x^2+y^2) \end{bmatrix} \tag{3.14}$$

El problema de optimización lineal se plantea a partir del vector $E = (-(f^2 + y^2), xy, fx, -(f^2 + x^2), fy, -(x^2 + y^2))$, que es una parte del polinomio base H linealmente independiente, y el vector $K = (t_x \cdot w_x, t_x \cdot w_y, t_x \cdot w_z, t_y \cdot w_x, t_y \cdot w_y, t_y \cdot w_z, t_z \cdot w_x, t_z \cdot w_y, t_z \cdot w_z)$, que representa las variables auxiliares para la traslación y la rotación; de esta forma es posible plantear el problema de optimización lineal con respecto a E y K de la siguiente manera:

$$\int_{\Omega_x} [T^T M + K^T E] \cdot E^T dx = 0 \tag{3.15}$$

$$\int_{\Omega_x} [T^T M + K^T E] \cdot [M + \frac{\partial (K^T E)}{\partial T}]^T dx = 0 \tag{3.16}$$

Solucionando la ecuación (3.15) con respecto a K y reemplazando este resultado en la ecuación (3.16) se obtiene el sistema de ecuaciones lineal y homogéneo como se muestra en la ecuación (3.17). Una solución no trivial se encuentra para el eigenvector que corresponde con los eigenvalores más pequeños de la matriz $3x3$ de dispersión C [11].

$$0 = T^T \int_{\Omega_x} L_i L_j dx =: T^T C, \qquad i,j = 1..3, \qquad donde \tag{3.17}$$

$$L_i := M_i - (DE)^T \int_{\Omega_x} E M_i dx, \qquad i,j = 1..3, \qquad y$$

$$D := [\int_{\Omega_x} E E^T dx]^{-1} \in \Re^{6x6}.$$

Para lograr un algoritmo robusto en presencia de valores atípicos, Raudies y Neumman utilizaron **RANSAC** para ajustar de mejor forma los vectores de flujo óptico y

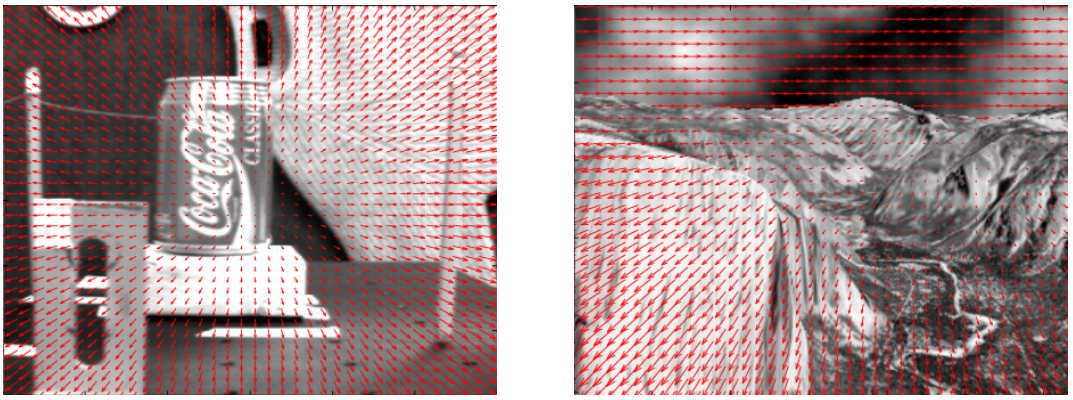
así obtener una mejor estimación del *movimiento propio*. Dado que **RANSAC** es un algoritmo iterativo y con una carga computacional considerable, nosotros hemos propuesto un método diferente para dar robustez frente a valores atípicos, este método será explicado y evaluado a continuación, apartado 3.4.

3.4 Análisis de Comportamiento de los Modelos para la Estimación de *Movimiento Propio*

Al analizar los modelos matemáticos que se han propuesto en la literatura, podemos establecer dos entidades de procesamiento bien definidas, una para la estimación de flujo óptico y la otra para resolver el problema de minimización que supone la *restricción bilineal*. Como ya se dijo en el apartado 3.3, el hardware para la estimación de flujo óptico, que hace parte del nivel de baja visión, fue desarrollado dentro del proyecto **DRIVSCO** y nuestro trabajo consistió en introducir este bloque de procesamiento dentro del sistema de visión que se propone en el marco de esta tesis. Aunque la arquitectura del sistema de visión se explicará en detalle en el capítulo 4, podemos decir en esta instancia que el sistema propuesto tiene un procesador **PowerPC** al cual iría conectado el módulo para la estimación de flujo óptico, de esta forma se aprovechan los recursos hardware de la **FPGA** para realizar las tareas del nivel de baja visión (flujo óptico) y se utiliza el procesador para realizar las tareas del nivel de media visión, en particular para ejecutar el algoritmo para la solución del problema de minimización de la *restricción bilineal*.

Dado que, en principio, en el **PowerPC** se ejecutarán los algoritmos propuestos por Raudies y Neumann y por Pauwels y Van Hulle para la estimación de *movimiento propio* y teniendo en cuenta que el sistema de visión que se quiere obtener debe funcionar en tiempo real, no es recomendable utilizar algoritmos iterativos en los cuales no sea posible determinar un número máximo de iteraciones para obtener un resultado aceptable; de igual forma es interesante poder determinar, al menos empíricamente, cuál es el número mínimo de vectores de flujo óptico que pueden ser utilizados en el problema de minimización sin que este proceso se vea afectado en precisión o robustez, pues el número de vectores de flujo óptico afecta de forma directa la cantidad de fotogramas que pueden ser procesados por segundo.

Tomando como referencia dos secuencias de vídeo reales bien conocidas, la secuencia

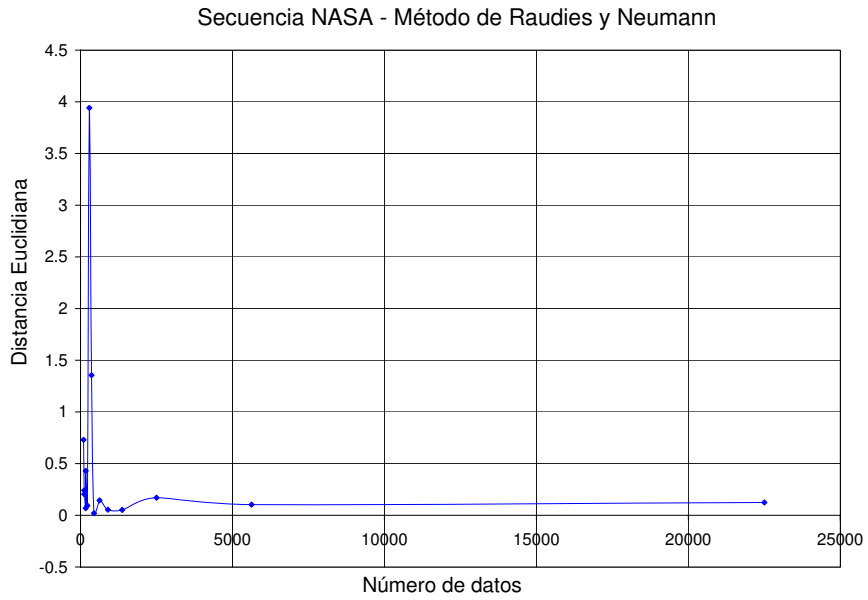


(a) Secuencia NASA [66]. **(b)** Secuencia YOSEMITE [67].

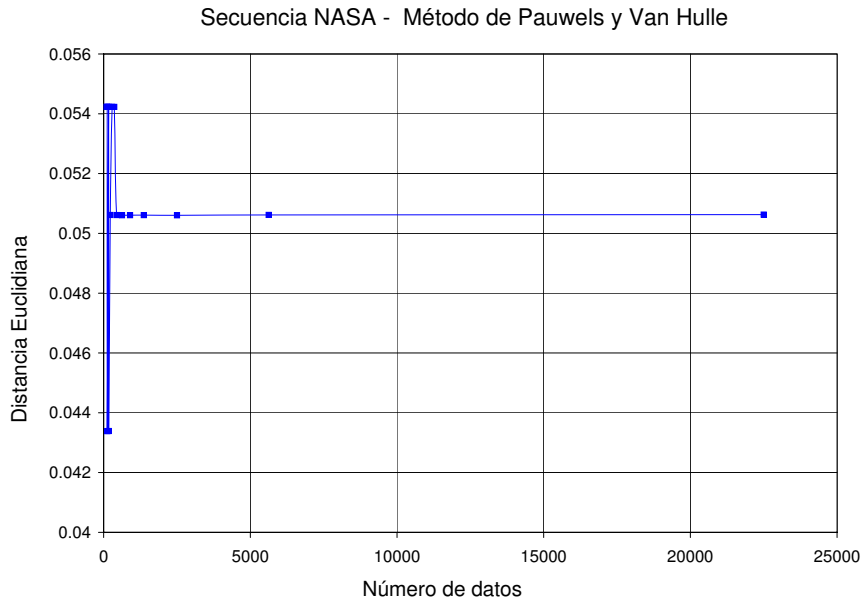
Figura 3.2: Secuencias reales con vectores de flujo óptico conocidos.

de la NASA Figura 3.2a y la secuencia de YOSEMITE Figura 3.2b, se han ejecutado los modelos antes mencionados variando el número de vectores de flujo óptico. Como se quiere estimar el número de vectores de movimiento necesarios para que el problema de minimización de la *restricción bilineal* sea estable, hemos utilizado los modelos de Raudies y Neumann, y Pauwels y Van Hulle en su forma más simple, es decir, sin tener en cuenta los segmentos de código que son usados para ajustar los modelos; en el caso de Raudies y Neumann sin usar **RANSAC** y en el caso de Pauwels y Van Hulle ejecutando sólo una inicialización de las 15 que proponen de media en su trabajo [57].

Los vectores de flujo óptico de la Figura 3.2 se consideran exactos y sin componentes de ruido, por tal motivo los resultados de las Figuras 3.3 y 3.4 obedecen específicamente al comportamiento de los modelos ante variaciones en el número de datos usados. En la Figura 3.3a se puede observar el comportamiento del modelo de Raudies y Neumann aplicado a los vectores de flujo óptico de la secuencia de la NASA, en ella puede verse que para este conjunto de datos el número mínimo de vectores para que las estimaciones sean precisas es de 5.000. De otro lado, en la Figura 3.3b se observa el comportamiento del modelo de Pauwels y Van Hulle para la secuencia de la NASA y puede decirse, a priori, que necesita sólo 1.000 vectores de flujo óptico para obtener estimaciones estables, sin embargo, su algoritmo se basa en un método no lineal para solucionar el problema de minimización de la *restricción bilineal*, por lo cual se ve altamente afectado por el valor inicial que se use para hacer la estimación; en su trabajo Pauwels y Van Hulle [57] reconocen que para obtener un resultado fiable es necesario realizar, en promedio, 15



(a) Modelo de Raudies y Neumann.



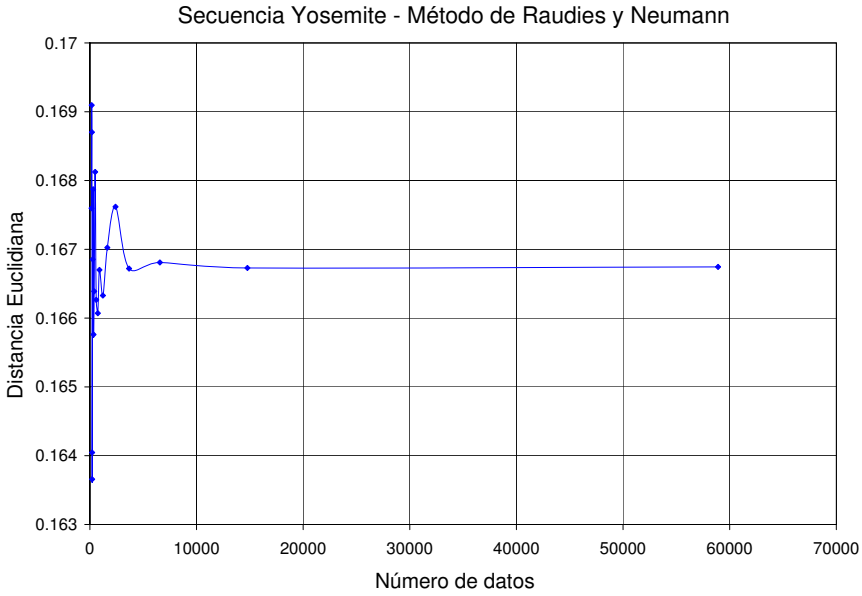
(b) Modelo de Pauwels y Van Hulle.

Figura 3.3: Comportamiento de los modelos para la estimación de *movimiento propio* para la secuencia de la NASA.

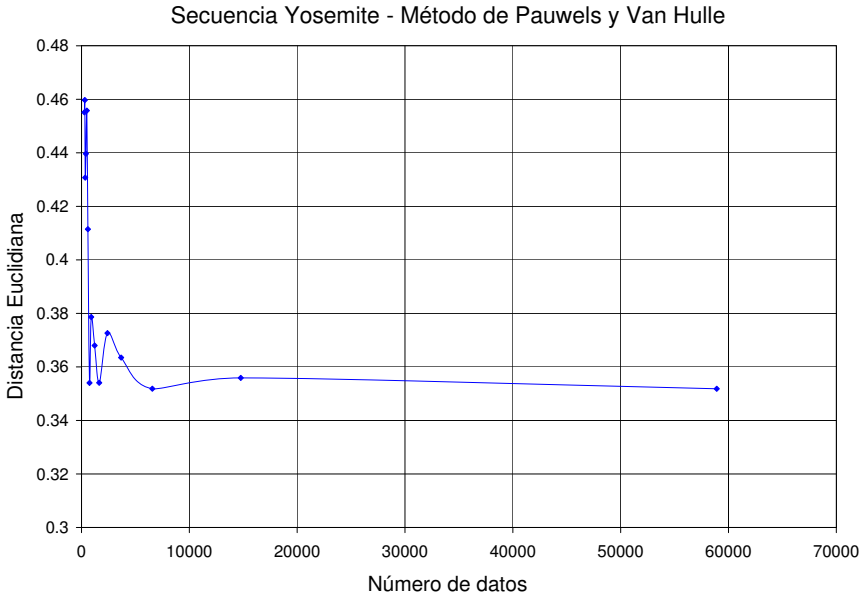
inicializaciones diferentes distribuidas de forma equidistante en el espacio de soluciones (vectores de traslación). Este último es otro elemento que debe tenerse en cuenta al momento de decidir cuál algoritmo es más plausible de ser utilizado en el sistema de visión en tiempo real propuesto, pues dadas las limitaciones de procesamiento que tiene el **PowerPC**, y la arquitectura en sí misma, no es posible ejecutar varias inicializaciones en paralelo.

La secuencia de YOSEMITE presenta un conjunto de vectores de flujo óptico un poco más complejo y se refleja en los resultados mostrados en la Figura 3.4. En la Figura 3.4a se observa el comportamiento del modelo de Raudies y Neumann y en la Figura 3.4b el comportamiento del modelo de Pauwels y Van Hulle, en ambos casos es necesario utilizar más de 15.000 vectores de flujo óptico para calcular el *movimiento propio* de forma precisa. De estas gráficas se puede establecer que es necesario ejecutar el algoritmo para la minimización de la *restricción bilineal* con al menos 15.000 vectores de flujo óptico para que los resultados sean fiables, sin embargo, la fiabilidad de las estimaciones se pierde, incluso tomando todo el conjunto de datos, cuando en los cálculos se utilizan campos de flujo óptico ruidosos y en presencia de **IMO**s.

Como ya se ha mencionado antes, para proveer robustez contra valores atípicos Raudies y Neumman, y Pauwels y Van Hulle propusieron algoritmos iterativos en los cuales no es posible determinar un número máximo de iteraciones. Llegados a este punto, aunque la arquitectura hardware propuesta puede ejecutar cualquiera de los dos algoritmos sin que ello implique cambios en la misma, es necesario decidir cuál de los algoritmos es el más adecuado para un sistema en tiempo real. En su trabajo Raudies y Neumman [58] demostraron que su algoritmo es más rápido que el propuesto por Pauwels y Van Hulle, y aún así la precisión de sus resultados es muy similar a la obtenida por estos. Teniendo en cuenta que, para su aplicación en navegación, los cambios en el *movimiento propio* no pueden ser bruscos y más si se tiene en cuenta que para obtener tiempo real es necesario estimar el *movimiento propio* en 25 fotogramas en un segundo, y que el método de Raudies y Neumman es más rápido, entonces es posible concebir un método de tan sólo dos iteraciones a partir de la solución del problema de minimización de la *restricción bilineal* propuesto por Raudies y Neumman. El método que se propone recibe el nombre de *R2-Iterativo* y se basa en una primera iteración del algoritmo de Raudies y Neumman con pocos puntos (de acuerdo a lo que se muestra en las Figuras 3.3 y 3.4) en la que se estima un *movimiento propio* inicial (T_i, W_i) que



(a) Modelo de Raudies y Neumann.



(b) Modelo de Pauwels y Van Hulle.

Figura 3.4: Comportamiento de los modelos para la estimación de *movimiento propio* para la secuencia de YOSEMITE.

se usa para filtrar el conjunto de vectores de flujo óptico; luego, una vez eliminados los valores atípicos en el conjunto de vectores de flujo óptico, se ejecuta una segunda iteración que entrega como resultado el *movimiento propio* (T, W) de la cámara.

Para entender el proceso de filtrado de los vectores de flujo óptico es necesario revisar de nuevo la ecuación (3.6), debemos recordar que el *movimiento propio* está definido por los vectores de traslación $T = (t_x, t_y, t_z)$ y rotación $W = (w_x, w_y, w_z)$. Luego de realizar la primera iteración del método *R2-Iterativo* obtenemos los vectores T_i y W_i con los cuales podemos determinar los vectores de flujo que definen el movimiento de cada píxel en la imagen de acuerdo al movimiento de la cámara, al reemplazar estos valores en la ecuación (3.6) y asumiendo, sin pérdida de generalidad, el plano de proyección en $Z = 1$, obtenemos los vectores de flujo óptico $U_i = (u_i, v_i)$ tal como se muestra en la ecuación (3.18).

$$U_i^T = AT_i^T + BW_i^T \tag{3.18}$$

Estos vectores indican cómo se debieron mover los píxeles en la imagen, es por esto que utilizamos U_i para realizar el filtrado de los vectores de flujo óptico estimados. Utilizando un umbral de confianza Th eliminamos los vectores de flujo óptico que no cumplan con la ecuación (3.19).

$$\begin{aligned} \theta &= arccos\left(\frac{\vec{U}\cdot\vec{U_i}}{\left|\vec{U}\right|\left|\vec{U_i}\right|}\right) \\ |\theta| &\leq Th \end{aligned} \tag{3.19}$$

De esta forma obtenemos un conjunto de vectores de flujo óptico filtrados en el cual sigue existiendo ruido, pero los valores atípicos debidos principalmente a **IMO**s son eliminados. Una buena elección del valor de umbral Th permite obtener una estimación del *movimiento propio* más precisa, sin embargo, si el umbral es muy pequeño entonces se corre el riesgo de tener una cantidad pequeña de vectores de flujo óptico al momento de ejecutar la segunda iteración del método *R2-Iterativo*, lo que impediría, paradójicamente, que los resultados fueran precisos. Empíricamente hemos determinado un valor de umbral $Th = 0.8rad$.

3.5 Modelo Simple para la Detección de IMOs

Los **IMO**s pueden dividirse en categorías y de acuerdo a éstas podemos definir el grado de dificultad para detectarlos. En una primera categoría están los objetos que se mueven de forma diferente al movimiento del observador (cámara en movimiento); en este contexto es posible detectar el **IMO** basados en el flujo óptico y en el *movimiento propio*. Para detectar los **IMO**s en esta situación es necesario calcular el flujo óptico y estimar el *movimiento propio* de forma robusta como ha sido explicado en el apartado 3.4. El *movimiento propio* define un patrón de flujo óptico que debe cumplirse para todos los objetos en la imagen a excepción de aquellos con movimiento independiente. De esta forma, cualquier vector de flujo que no cumpla con este patrón es susceptible de ser un **IMO**.

La segunda categoría, y en principio más compleja que la primera, es cuando el objeto y el observador tiene un movimiento idéntico pero el objeto se mueve más rápido que el observador. En este contexto nuestro sistema no puede ser utilizado pues es necesario tener adicionalmente la estructura tridimensional de la escena a partir del movimiento (más conocido por sus traducción al inglés *structure from motion*); esta información adicional nos permitiría conocer la profundidad de la escena; esta profundidad será positiva en todos los puntos excepto en aquellos pertenecientes a **IMO**s.

En la tercera categoría encontramos el escenario más complicado, en esta categoría el movimiento del objeto y del observador es idéntico pero con sentidos contrarios. Por ejemplo en carretera esta situación se presenta cuando el observador se acerca a un vehículo que se encuentra en frente, este escenario es muy complejo y ni el *movimiento propio*, ni la estructura tridimensional a partir del movimiento son suficientes para detectar el **IMO**.

De acuerdo a lo antes mencionado, nuestro sistema sólo puede detectar los **IMO**s catalogados en la primera categoría. Una vez determinado el *movimiento propio* es posible filtrar los vectores de flujo óptico que se corresponden con aquellos del *movimiento propio*, tal como se hace en el método *R2-Iterativo* con la ecuación (3.19). Una vez etiquetados los píxeles que tienen un movimiento diferente al descrito por el *movimiento propio*, se define una vecindad 3x3 que nos permite determinar si en efecto corresponde a un **IMO** o simplemente es un píxel aislado. La figura 3.5 ilustra la detección de **IMO**s

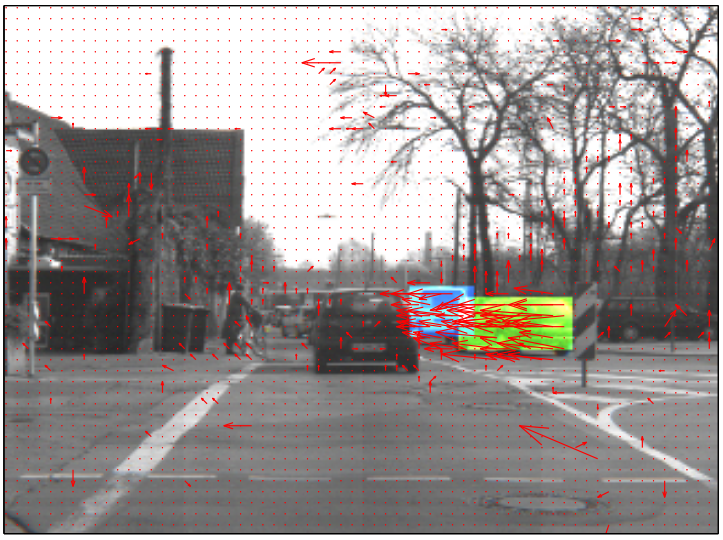


Figura 3.5: Ilustración de la detección de **IMO**s para una secuencia real. La detección de los **IMO**s se hace con un etiquetado y una regularización en una vecindad de 3x3.

en una secuencia real, los **IMO**s se detectan por medio del método descrito anteriormente; en esta figura sólo los puntos relacionados con vehículos representan **IMO**s. En general son zonas con alguna coherencia espacial de **IMO**, es decir, puntos aislados que no se corresponden con el patrón de *movimiento propio* serían catalogados como erróneos, mientras "nubes de puntos" en una vecindad y que no se corresponden con el patrón de *movimiento propio* si que se catalogarían como **IMO**s.

Una ilustración adicional se hace en la figura 3.6, en esta figura los **IMO** se detectan mediante el etiquetado de los vectores de flujo óptico que no corresponden con el *movimiento propio* sin utilizar la regularización que se hace por medio de la vecindad de 3x3. Se puede captar del análisis de estas figuras que la regularización por medio de una vecindad permite eliminar algunos vectores de flujo aislados. Como línea de trabajo futuro queda el desarrollo de un método más robusto que permita mejorar el modelo simple que acá se propone y que además pueda detectar **IMO**s en las demás categorías antes mencionadas.

En conclusión podemos resumir el modelo del sistema de visión como un sistema híbrido hardware/software en el que se ejecuta la modificación *R2-Iterativo* propuesta

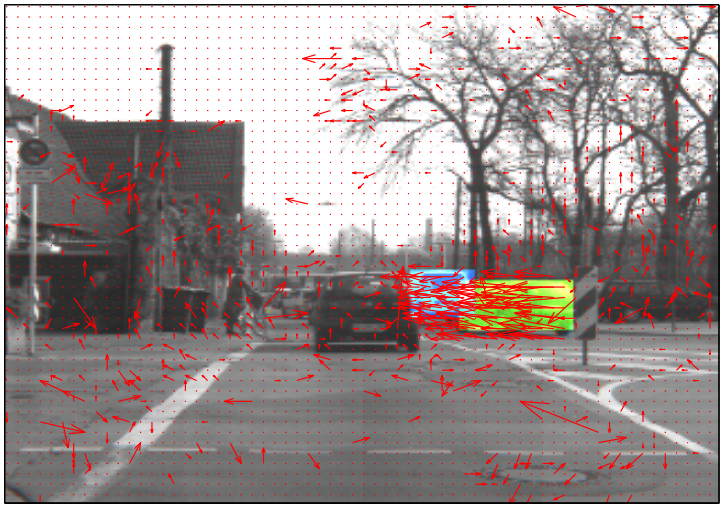


Figura 3.6: Ilustración de la detección de **IMO**s para una secuencia real. La detección de los **IMO**s se hace sólo con el etiquetado.

del modelo desarrollado por Raudies y Neumman para la estimación del *movimiento propio*, asimismo se aprovecha el método *R2-Iterativo* para etiquetar los vectores de flujo óptico que son susceptibles de pertenecer a **IMO**s. En el capítulo 4 se explicará detalladamente la arquitectura propuesta, así como algunas modificaciones hechas a la misma con el fin de acelerar la ejecución del software.

4

Arquitectura Hardware del Sistema

En este capítulo se describe en detalle la arquitectura **SoC** del sistema de visión propuesto. Este sistema se basa en un procesador **PowerPC** al cual se conectan, como periféricos, unas entidades de cálculo específicas que sirven para acelerar la ejecución del modelo matemático escogido en el capítulo 3. Debido a que el **PowerPC** tiene una velocidad de acceso a memoria muy pequeña, hemos construido dos tipos de arquitectura, estas arquitecturas son descritas en detalle, así como los factores que motivaron el desarrollo de las mismas. Finalmente, se muestran los resultados de la implementación hardware y de la velocidad de ejecución de ambas arquitecturas.

4.1 Introducción

La creciente necesidad de sistemas de ejecución independiente en campos de aplicación como la conducción asistida, la robótica y en algunos casos la medicina, hace relevante la investigación en sistemas de procesamiento de vídeo [68, 69], esta tesis tiene como objetivo construir un sistema independiente capaz de procesar información en el nivel de baja y media visión, cálculo de flujo óptico y de *movimiento propio* respectivamente. En la última década se han generado grandes avances en sistemas independientes capaces de calcular flujo óptico en tiempo real, algunos de estos sistemas han sido implementados en procesadores convencionales [5, 70, 71] mediante técnicas de optimización de código y aprovechando al máximo la arquitectura de los procesadores utilizados, otros han sido implementados en **GPU**s [6], pero la gran mayoría han sido implementados en **FPGA**s [72, 73, 74, 75]. Aunque en la última década se han desarrollado muchos sistemas de tiempo real en el nivel de baja visión, son muy poco los que se han desarrollado en el nivel de media visión; la gran mayoría de sistemas capaces de estimar la estructura de movimiento han sido implementados en software [76, 77], sin embargo, David Nistér [78] es el primero en desarrollar un sistema robusto y en tiempo real capaz de estimar el movimiento de una cámara en un escenario rígido. Hasta donde hemos podido investigar, sólo un sistema hardware se ha publicado a la fecha, Ania Mitros [79] ha desarrollado un sensor analógico capaz de estimar el *movimiento propio* en tiempo real, ella ha realizado un diseño **VLSI** tanto del sensor óptico como de los circuitos para la estimación del *movimiento propio*.

Fife y Archibald [80] mostraron en su trabajo las ventajas que tienen las **FPGA**s en el campo de la navegación autónoma, dado su capacidad de procesamiento en un sólo chip y a su bajo consumo de potencia. Claus y Stechele [81] han desarrollado un **SoC** basado en **FPGA** que permite la configuración parcial y durante la ejecución del sistema de funciones de procesamiento de vídeo en la **FPGA**, de esta forma el sistema es capaz ejecutar varios algoritmos de procesamiento de vídeo en hardware, pero reutilizando recursos.

El sistema que se explica a continuación es un **SoC** capaz de estimar el *movimiento propio* de una cámara en tiempo real, ha sido implementado en una **FPGA** y, dado que su arquitectura es basada en un procesador, permite un procesamiento híbrido hardware/software que lo hace idóneo para aplicaciones independientes y autónomas.

Las tareas desarrolladas tienen una carga computacional muy alta y requieren de modelos computacionales muy complejos para ser capaces de analizar el movimiento de forma robusta y ello ha motivado que pocos autores hayan trabajado en su implementación en un sistema empotrado; por tanto, el sistema presente es, hasta donde conocemos, el único **SoC** existente capaz de estimar flujo óptico y analizarlo para determinar la estructura de movimiento de una cámara en tiempo real.

4.2 Arquitectura del Sistema

El rendimiento del sistema propuesto **SoC** es altamente dependiente del comportamiento de los elementos de memoria utilizados. El **SoC** está compuesto por dos unidades de procesamiento y varias memorias **RAM**. Cada unidad de procesamiento utiliza su propia memoria **RAM** de acuerdo a sus necesidades, por una parte el **PowerPC** utiliza algunos bloques de memoria **RAM** embebidas en la **FPGA** como memorias *cache*, de instrucciones y datos respectivamente, para incrementar su rendimiento y de otra parte la entidad para la estimación de flujo óptico utiliza memorias externas **ZBT** conectadas a través de varias **MCU**s. Adicionalmente, el sistema cuenta con una memoria *DDR SDRAM* de propósito general que es controlada a través del controlador de memoria con múltiples puertos **MPMC** desarrollado por Xilinx [82]. El **MPMC** tiene 8 puertos independientes que pueden ser configurados de acuerdo a los tipos de interfaces definidas por Xilinx.

El sistema completo fue implementado en la tarjeta de desarrollo *XIRCA-V4* de la empresa Seven Solution S.L. [38]. Esta tarjeta tiene dos versiones, una está equipada con una **FPGA** *XC4VFX60-10* que cuenta con 25.280 bloques funcionales y la otra con una **FPGA** *XC4VFX100-10* que cuenta con 42.176 bloques funcionales, ambas **FPGA**s son de Xilinx y tienen dos procesadores **PowerPC** embebidos, además estas tarjetas cuentan con dos bancos de memoria **DDR** de 512 Mb. cada uno, cuatro bancos de memoria **ZBT** de 72 Mb. (2M x 36 bits) cada uno, una interfaz **PCIe** y un puerto serie *RS-232* entre otros elementos; estas dos versiones son necesarias ya que la complejidad del sistema requiere de una gran cantidad de recursos hardware de la **FPGA** y hace que, según el consumo de recursos de la arquitectura diseñada, sea necesario utilizar una u otra. La Figura 4.1 muestra el diagrama del **SoC** propuesto; el sistema está conectado a un computador de escritorio a través de la interfaz **PCIe** con el fin de

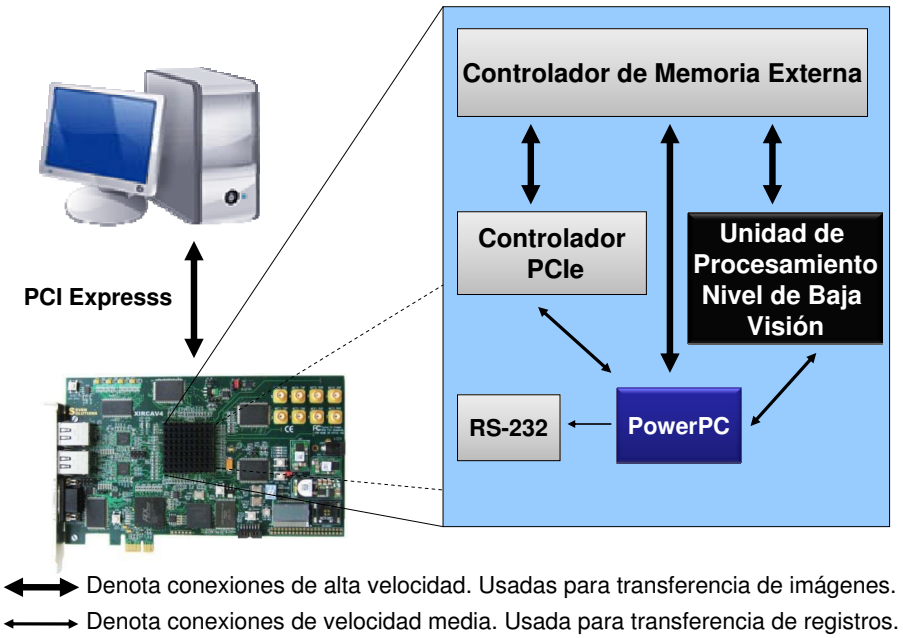


Figura 4.1: Diagrama en bloques de la arquitectura del **SoC**.

que sea éste el encargado de capturar y mostrar las imágenes, tanto capturadas como las procesadas, en este sentido el computador de escritorio es utilizado como interfaz visual exclusivamente.

Todas las unidades de procesamiento en el sistema pueden leer y escribir de forma autónoma la memoria **DDR**, así, el computador de escritorio captura imágenes de una cámara de vídeo y las escribe en la memoria **DDR** a través de la interfaz **PCIe**, la unidad de control lee estas imágenes y calcula el flujo óptico y por último, el **PowerPC** lee la información de flujo óptico y estima el *movimiento propio*. Es de resaltar que el **PowerPC** se encarga de sincronizar todos los procesos de cálculo por medio de una asignación dinámica de la memoria **DDR**, es decir, mientras el computador de escritorio está escribiendo la imagen del tiempo de proceso t, la entidad para el cálculo de flujo óptico está procesando el flujo de la imagen del tiempo de proceso $t-1$. Como el **PowerPC** necesita los datos de flujo óptico, entonces éste espera a tenerlos para estimar el *movimiento propio*; una vez a terminado su proceso vuelve a sincronizar cambiando la asignación de la memoria.

En la Figura 4.2 se presenta un diagrama de conexión del **SoC** en el que se mues-

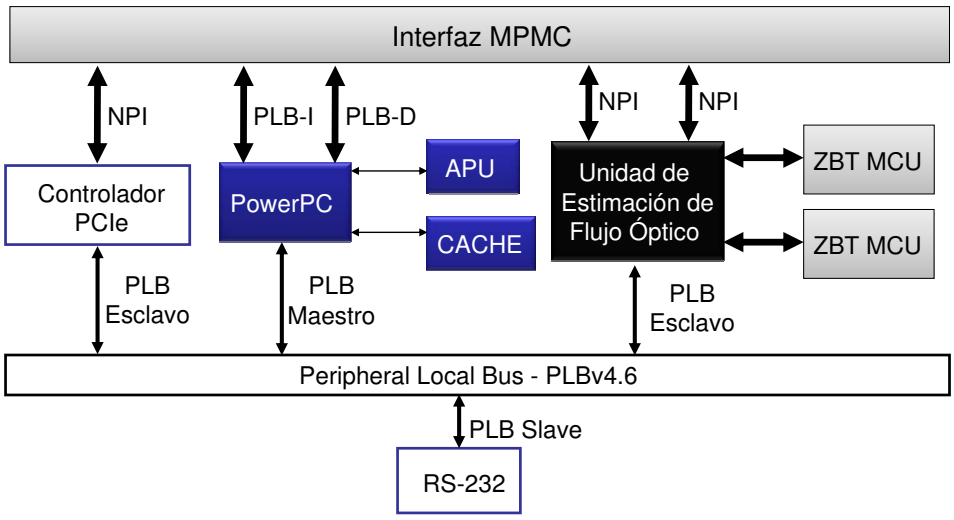


Figura 4.2: Diagrama de conexiones de la arquitectura del **SoC**.

tra de forma esquemática la conexión de todas las unidades de procesamiento; como ya se ha mencionado antes, la memoria **DDR** se comparte totalmente y por ello cada unidad de procesamiento puede leer la información que cualquiera otra pudo escribir, en este sentido la **DDR** se utiliza como interfaz de transferencia masiva de datos. Sin detrimento de lo anterior, el sistema cuenta con un bus de datos (**PLB**) que sirve para conectar efectivamente todas las unidades de procesamiento. Por medio del **PLB** es posible mapear, utilizando registros, todos las entidades que constituyen el **SoC**. Dentro del sistema sólo el **PowerPC** es conectado como maestro al **PLB** y el resto de entidades se conectan como esclavas, de esta forma se tiene un sistema de procesamiento centralizado en el que el **PowerPC** es el encargado de asignar tareas y de identificar el momento en que deben realizarse.

También pueden verse algunas particularidades del **SoC** en la Figura 4.2, por un lado el **PowerPC** tiene una unidad auxiliar de procesamiento (APU) que realiza operaciones aritméticas en punto flotante; esta APU se conecta al **PowerPC** con el fin de aumentar el rendimiento de éste cuando muchas operaciones aritméticas necesitan ser ejecutadas, además el modelo para la estimación de *movimiento propio* exige operaciones aritméticas en punto flotante. En contraste, la unidad para el cálculo de flujo óptico utiliza una aritmética de punto fijo debido a que es posible predecir la profundidad de bits necesaria para cada píxel de la imagen. Las componentes del flujo óptico

utilizan un tamaño de palabra de 12 bits cada una, 8 bits de parte entera y 4 de parte fraccionaria (ver (O_x, O_y) en la Figura 3.1).

En este punto es importante resaltar que la **MCU** [83] tiene un papel muy importante dentro del diseño de los circuitos que estiman primitivas en nivel de baja visión; debemos recordar que este nivel requiere de un procesamiento en el que múltiples entidades trabajan en paralelo con datos compartidos. En este sentido la **MCU** entrega un acceso a datos compartido de forma óptima en arquitecturas con múltiples caminos de procesamiento de datos como el que se presenta en el esquema de la Figura 3.1 (unidad para la estimación de flujo óptico). Además de las ventajas ofrecidas por la **MCU** en el rendimiento del sistema, subyace en su utilización la inherente abstracción de los accesos a memoria de tal forma que para el diseñador es de fácil integración.

A continuación, en el apartado 4.3, se muestran los resultados de la validación del **SoC**; debemos recordar que según nuestra metodología de trabajo este sistema combina el nivel de baja visión (hardware) con el nivel de media visión (software o híbrido hardware/software); esta primera aproximación al sistema definitivo ejecuta la componente de media visión totalmente en software con el fin de analizar que puntos son susceptibles de mejora para obtener el máximo rendimiento del sistema.

4.3 Validación del Sistema

La validación del sistema debe hacerse en dos instancias, primero es necesario demostrar que el sistema puede estimar el *movimiento propio* de forma precisa incluso con campos de flujo óptico complejos y después demostrar que el sistema puede hacerlo, como mínimo, a una velocidad de 20 fotogramas por segundo (*fps*). En el apartado 4.3.1 se evaluará la precisión del método que hemos propuesto en el capítulo 3 y en el apartado 4.3.2 se evaluará la velocidad de procesamiento del sistema en el que la estimación del *movimiento propio* se hace totalmente en software.

4.3.1 Evaluación del Método *R2-Iterativo*

Dado que el **SoC** propuesto presenta un rendimiento limitado por la velocidad de acceso a memoria que impide la ejecución de múltiples iteraciones del algoritmo de Raudies y Neumman, hemos propuesto el método *R2-Iterativo* por medio del cual podemos estimar de forma precisa el *movimiento propio* incluso ante vectores de flujo óptico

Tabla 4.1: Error angular absoluto (*EAA*) de 16 secuencias de flujo óptico sintéticas (con ruido y en presencia de **IMO**s) ante variaciones en el número de vectores de flujo óptico utilizando el método *R2-Iterativo*.

Secuencia	*EAA*	*EAA*	*EAA*	*EAA*	*EAA*
Flow 1	0.0033	0.0022	0.0026	0.0028	0.0028
Flow 2	0.0140	0.0157	0.0157	0.0157	0.0188
Flow 3	0.0184	0.0200	0.0218	0.0215	0.0216
Flow 4	0.0028	0.0023	0.0024	0.0025	0.0026
Flow 5	0.0015	0.0012	0.0011	0.0013	0.0014
Flow 6	0.0044	0.0054	0.0057	0.0058	0.0061
Flow 7	0.0024	0.0013	0.0018	0.0018	0.0021
Flow 8	0.0029	0.0023	0.0029	0.0029	0.0029
Flow 9	0.0129	0.0126	0.0129	0.0127	0.0127
Flow 10	0.0102	0.0114	0.0115	0.0119	0.0119
Flow 11	0.0090	0.0090	0.0092	0.0091	0.0091
Flow 12	0.0072	0.0070	0.0072	0.0075	0.0085
Flow 13	0.0093	0.0093	0.0092	0.0091	0.0111
Flow 14	0.0330	0.0340	0.0334	0.0335	0.0342
Flow 15	0.0406	0.0411	0.0416	0.0422	0.0418
Flow 16	0.0181	0.0195	0.0198	0.0195	0.0198
Número de Datos	300	1200	4800	19200	76800

ruidosos y con presencia de **IMO**s. Para demostrar el comportamiento adecuado de nuestro modelo, hemos construido campos de flujo óptico sintéticos, estos campos son estimados mediante el uso del modelo de *movimiento instantáneo*, tal como se describe en la ecuación (3.6) (Z se escoge de forma aleatoria y W se hace cero), y se definen a partir de un vector de traslación tomado de una esfera unitaria, los parámetros α, β definen dicho vector. La tabla 4.1 muestra los errores angulares absolutos de 16 secuencias sintéticas procesadas con el método *R2-Iterativo*, en ella se puede apreciar que el método es suficientemente estable y con errores angulares muy pequeños.

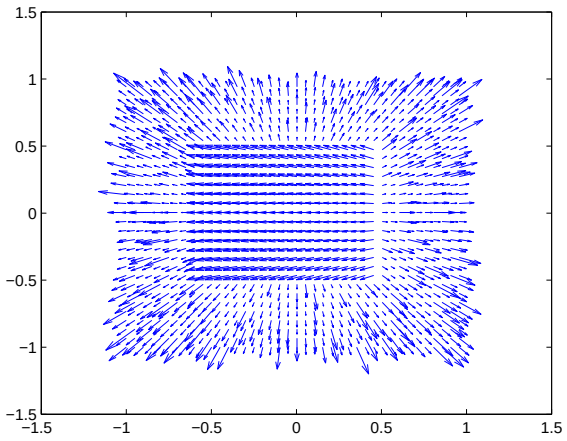
De los 16 campos de flujo antes mencionados hemos escogido dos, por su nivel de complejidad, para demostrar el adecuado funcionamiento del método *R2-Iterativo*, *Flow 1* que está definido por los parámetros $\alpha = 0°$ y $\beta = 0°$ y *Flow 2* que está definido por $\alpha = 30°$ y $\beta = 0°$. También hemos añadido **IMO**s que corresponden al 25% del tamaño de la imagen (320x240) y su movimiento es definido, una vez más, a partir de un vector de parámetros α, β tomado de una esfera unitaria. La Figura 4.3 muestra

los dos campos de flujo óptico cada uno con un **IMO** del 25%; la Figura 4.3a tiene un **IMO** definido por los parámetros $\alpha = 30°$ y $\beta = 0°$ y la Figura 4.3b un **IMO** definido por $\alpha = 0°$ y $\beta = -30°$. En todos los casos antes mencionados se tiene una componente de rotación igual a cero, es decir, tanto la cámara como los **IMO**s deben su movimiento sólo a la traslación.

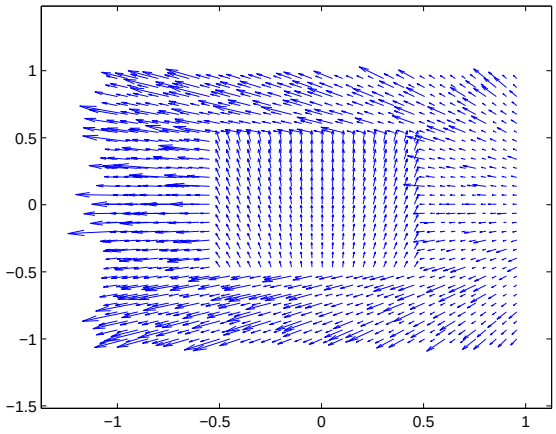
Para determinar la precisión del modelo hemos reducido gradualmente el número de vectores de flujo óptico que son utilizados en la estimación del *movimiento propio* desde el total disponible (320x240) hasta el 1% de éstos, así, además de estimar la robustez del método ante variaciones en el número de datos usados para realizar la minimización de la *restricción bilineal*, podremos analizar el número de fotogramas que pueden ser analizado por segundo ejecutando el algoritmo totalmente en software en el **PowerPC**.

Los vectores de flujo óptico que son utilizados en la estimación del *movimiento propio* son escogidos de forma equidistante y los resultados de las estimación se muestran en la Figura 4.4. La *Serie 1* corresponde a la primera iteración del algoritmo de Raudies y Neumman y la *Serie 2* corresponde a la segunda iteración del algoritmo luego de filtrar los vectores de flujo óptico que no se ajustan al modelo según la ecuación (3.19). Los resultados de la *Serie 2* muestran que el modelo *R2-Iterativo* funciona bien incluso cuando el número de vectores de flujo óptico utilizados es bajo. Se puede notar en la Figura 4.4a que el error angular absoluto *EAA* en la *Serie 1* disminuye cuando el número de vectores de flujo óptico decrece, esto se debe a que al disminuir el número de vectores también se disminuyen aquellos que pertenecen al **IMO**, de cualquier modo, los resultados en la *Serie 2* son más regulares y su *EAA* es muy bajo.

Los resultados de la Figura 4.4 demuestran que el modelo propuesto funciona adecuadamente con campos de flujo óptico con valores atípicos (**IMO**s), sin embargo es necesario utilizar campos de flujo óptico ruidosos para verificar la robustez del modelo; para ello hemos añadido ruido a los campos de flujo óptico antes mencionados. Hemos escogido un ruido aditivo normalmente distribuido con una desviación estándar de 0.1 y lo hemos añadido a los campos de flujo óptico como se muestra en la Figura 4.5. Estos vectores de flujo óptico se utilizan una vez más para demostrar la precisión del método propuesto *R2-Iterativo*; siguiendo el mismo procedimiento anterior, hemos estimado el *movimiento propio* utilizando un conjunto de vectores de flujo óptico de tamaño diferente cada vez y hemos escogido el conjunto de vectores de flujo utilizando una distribución normal; los resultados se muestran en la Figura 4.6.

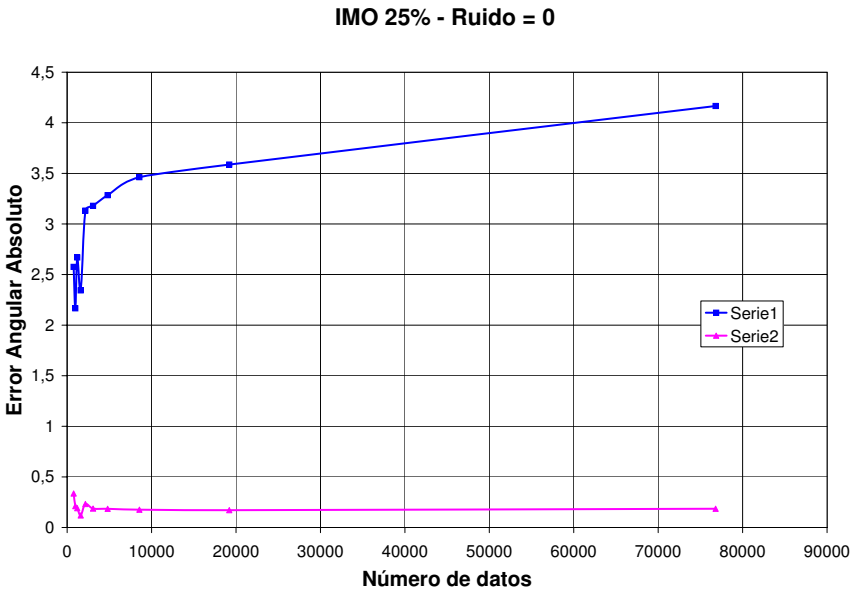


(a) *Flow 1* con parámetros $\alpha = 0°$ y $\beta = 0°$. El **IMO** está definido por los parámetros $\alpha = 30°$ y $\beta = 0°$.

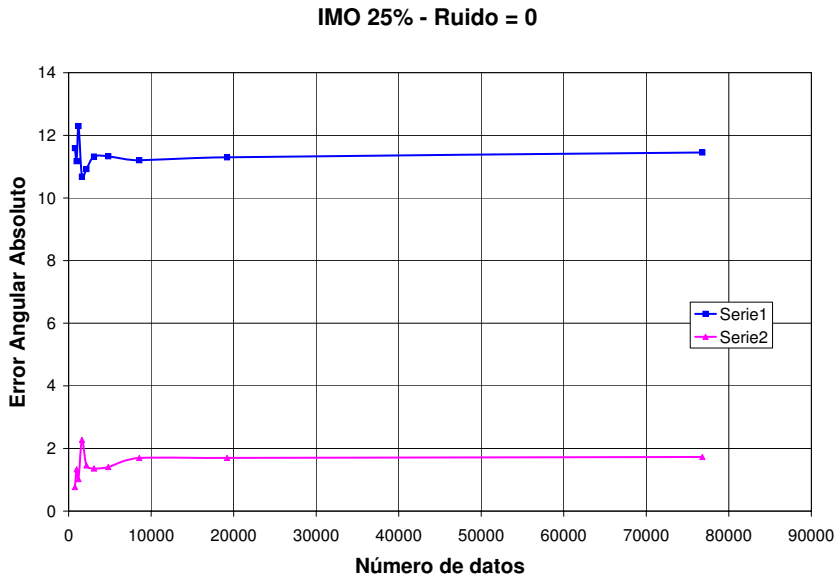


(b) *Flow 2* con parámetros $\alpha = 30°$ y $\beta = 0°$. El **IMO** está definido por los parámetros $\alpha = 0°$ y $\beta = -30°$.

Figura 4.3: Campos de flujo óptico sintéticos. El **IMO** corresponde con el 25% del tamaño de la imagen (320x240).



(a) *Flow 1.*



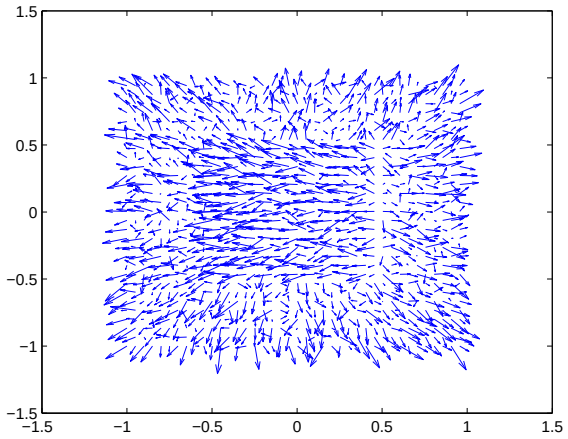
(b) *Flow 2.*

Figura 4.4: Error angular absoluto *EAA*. Estimación del *movimiento propio* con vectores de flujo óptico sin ruido y con un **IMO** del 25%.

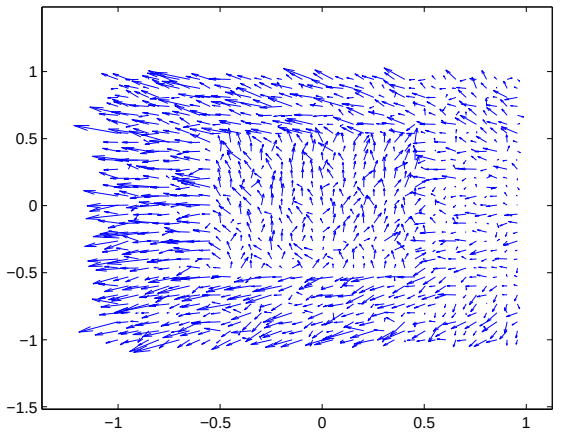
Para hacer un análisis exhaustivo de la precisión del sistema, es necesario primero validar que el campo sintético de flujo óptico ruidoso que hemos construido es representativo y que, en efecto, demuestra que nuestro **SoC** es preciso. Como ya se explicó antes, el campo de flujo óptico sintético se obtiene a partir del modelo de *movimiento instantáneo* y por tal motivo el movimiento de cada píxel de la imagen está determinado por el vector de traslación de acuerdo a la ecuación (3.6), el término de la ecuación que se refiere a la rotación no se tiene en cuenta debido a que hemos escogido un movimiento de rotación igual a cero. Con esta información podemos construir una secuencia de imágenes sintéticas por medio de las cuales evaluaremos la precisión de la unidad de cálculo de flujo óptico y así poder concluir si el ruido que hemos añadido al campo de flujo óptico sintético es representativo.

Para construir la secuencia de imágenes sintéticas utilizamos la textura de la conocida secuencia de YOSEMITE, esta decisión se basa en que la textura de esta secuencia es bastante compleja y es una de las más utilizadas para evaluar la precisión de modelos matemáticos para la estimación de flujo óptico. Cada píxel de la imagen de YOSEMITE fue movido de acuerdo al mapa de movimiento que se obtiene de la ecuación (3.6) y la correspondiente transformación del vector de traslación de coordenadas esféricas (con parámetros α y β) a coordenadas cartesianas (con parámetros t_x, t_y y t_z). Una vez obtenidas las imágenes sintéticas con la textura de YOSEMITE, fue posible estimar el flujo óptico real por medio de la unidad de cálculo de flujo óptico del sistema. Tal como se muestra en la tabla 4.2, el error añadido a los campos de flujo óptico sintéticos corresponden a los errores típicos que maneja la unidad de cálculo de flujo óptico del sistema para densidades muy cercanas al 100%, lo cual valida los resultados de la Figura 4.6, sin embargo, es posible mejorar en gran medida el error que entrega la unidad de cálculo de flujo óptico si disminuimos la densidad; al diminuir la densidad estamos disminuyendo el número de vectores de flujo óptico disponibles, pero como ya se ha demostrado antes, nuestro modelo de estimación del *movimiento propio* funciona adecuadamente incluso con un pequeño subconjunto de vectores de flujo óptico.

Los resultados de la Figura 4.6 y los contenidos en la tabla 4.1 validan el método *R2-Iterativo* y muestran un comportamiento muy estable incluso con vectores de flujo óptico ruidosos; asimismo queda validado que nuestro método es preciso incluso cuando los datos tienen valores atípicos como en el caso de campos de flujo óptico en presencia de **IMO**s.

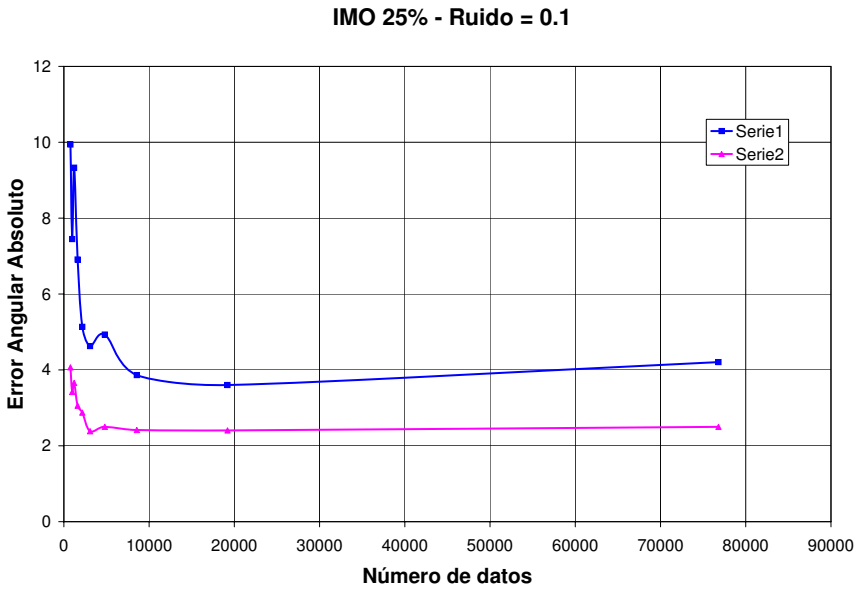


(a) *Flow 1* con parámetros $\alpha = 0°$ y $\beta = 0°$. El **IMO** está definido por los parámetros $\alpha = 30°$ y $\beta = 0°$.

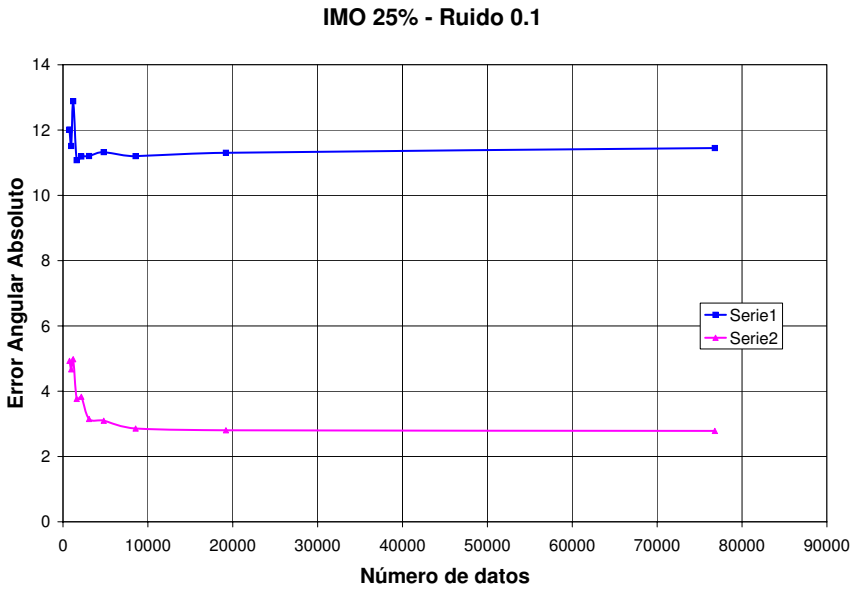


(b) *Flow 2* con parámetros $\alpha = 30°$ y $\beta = 0°$. El **IMO** está definido por los parámetros $\alpha = 0°$ y $\beta = -30°$.

Figura 4.5: Campos de flujo óptico sintéticos con ruido aditivo normalmente distribuido con una desviación estándar de 0.1. El **IMO** corresponde con el 25% de la imagen (320x240).



(a) *Flow 1.*



(b) *Flow 2.*

Figura 4.6: Error angular absoluto *EAA*. Estimación del *movimiento propio* con vectores de flujo óptico con ruido y con un **IMO** del 25%.

Tabla 4.2: Error angular medio (*EAM*) y densidad de datos. Calculados para el campo sintético de flujo óptico ruidoso y para la estimación hardware.

Secuencia de datos	*EAM* datos hardware	*EAM* datos sintéticos	densidad datos hardware (%)	densidad datos sintéticos (%)
Flow 1	8.3934	10.5389	97.9150	100
Flow 2	11.2993	10.6515	96.7631	100

Tabla 4.3: Error angular absoluto (*EAA*) de 16 secuencias de flujo óptico sintéticas (con ruido y en presencia de **IMO**s) ante variaciones en el número de vectores de flujo óptico utilizando un filtrado adicional luego de la ejecución del método *R2-Iterativo*.

Secuencia	*EAA*	*EAA*	*EAA*	*EAA*	*EAA*
Flow 1	0.0033	0.0022	0.0026	0.0028	0.0028
Flow 2	0.0140	0.0157	0.0157	0.0157	0.0188
Flow 3	0.0184	0.0200	0.0218	0.0215	0.0216
Flow 4	0.0028	0.0023	0.0024	0.0025	0.0026
Flow 5	0.0015	0.0012	0.0011	0.0013	0.0014
Flow 6	0.0044	0.0054	0.0057	0.0058	0.0061
Flow 7	0.0024	0.0013	0.0018	0.0018	0.0021
Flow 8	0.0029	0.0023	0.0029	0.0029	0.0029
Flow 9	0.0129	0.0126	0.0129	0.0127	0.0127
Flow 10	0.0102	0.0114	0.0115	0.0119	0.0119
Flow 11	0.0090	0.0090	0.0092	0.0091	0.0091
Flow 12	0.0072	0.0070	0.0072	0.0075	0.0085
Flow 13	0.0093	0.0093	0.0092	0.0091	0.0111
Flow 14	0.0330	0.0340	0.0334	0.0335	0.0342
Flow 15	0.0406	0.0411	0.0416	0.0422	0.0418
Flow 16	0.0181	0.0195	0.0198	0.0195	0.0198
Número de Datos	300	1200	4800	19200	76800

De otra parte, las tablas 4.1 y 4.3 contienen los errores angulares absolutos en las estimaciones del *movimiento propio* ante variaciones en el número de vectores de flujo óptico utilizados. La tabla 4.1 muestra los resultados con el método *R2-Iterativo* y la tabla 4.3 muestra los resultados al realizar un filtrado de vectores de flujo óptico adicional al realizado en el método *R2-Iterativo*, estos resultados demuestran que no existe ventaja alguna con filtrar dos veces los vectores de flujo óptico y que el método *R2-Iterativo* es óptimo.

En conclusión, podemos asegurar que con un conjunto de vectores de flujo óptico mayor a 22.000 datos, el método *R2-Iterativo* es robusto a valores atípicos y significativamente inmune al ruido.

4.3.2 Evaluación de la Velocidad del Sistema

Antes de evaluar en detalle la velocidad de ejecución del sistema propuesto con el método *R2-Iterativo*, es interesante mostrar la velocidad de ejecución del sistema con los métodos de Pauwels y Van Hulle, y Raudies y Neumman ejecutados totalmente en el **PowerPC**, la figura 4.7 muestra el número de fotogramas por segundo que son evaluados por nuestro sistema; los métodos de Pauwels y Van Hulle, y Raudies y Neumman han sido simplificados de acuerdo a lo explicado en el apartado 3.4. Teniendo en cuenta estos resultados y basándonos en lo expresado en el apartado 3.4 se sustenta la escogencia del método de Raudies y Neumman para la implementación de nuestro sistema.

En la Figura 4.8 se muestra la velocidad del sistema para una sola iteración del algoritmo de Raudies y Neumman; como puede verse, la velocidad del sistema software no permite una ejecución en tiempo real del modelo *R2-Iterativo* y por ello es necesario diseñar un módulo hardware para acelerar el procesamiento del **PowerPC**.

Para definir qué parte del código debe ser acelerado hicimos un análisis de rendimiento del código ejecutado y los resultados se muestran en la tabla 4.4. El código para la estimación del *movimiento propio* los dividimos en 9 rutinas, tal como se muestra en la figura 4.9, que se repiten en cada una de las dos iteración del método *R2-Iterativo*. Cada una de las rutinas ha sido escogida bien sea porque las operaciones aritméticas que realiza son complejas o porque es un ciclo *for*. De acuerdo a los resultados mostrados en la tabla 4.4 podemos concluir que son los ciclos *for* (rutinas *2, 4 y 5*) que recorren todo el conjunto de vectores de flujo óptico (320x240) los que más tiempo tardan en ejecutarse, casi el 100% del tiempo. Al analizar estos resultados y sabiendo que el procesador está

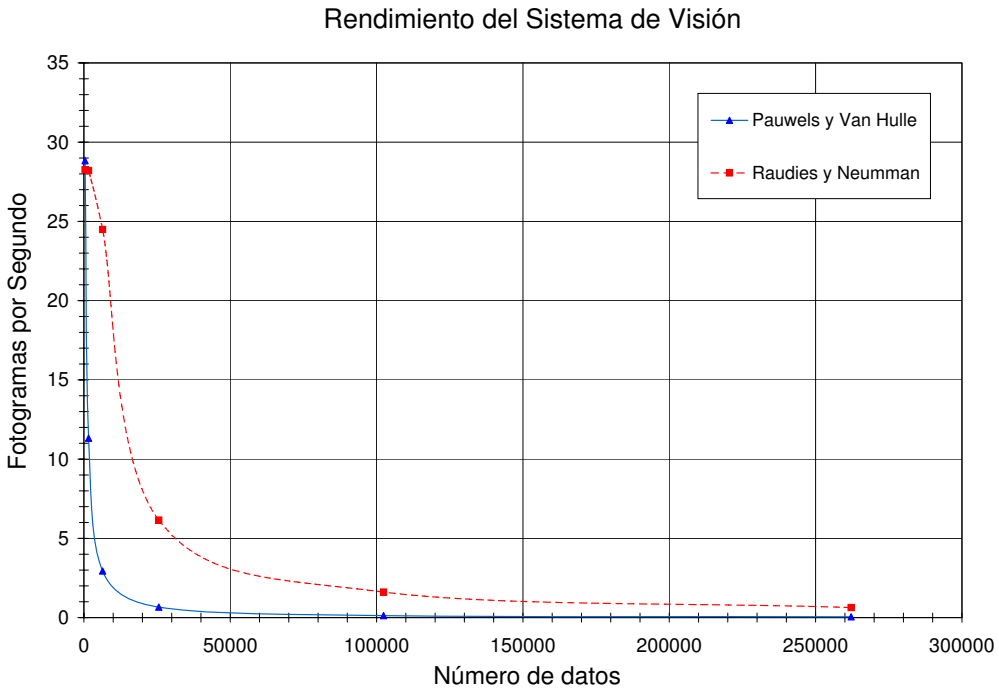


Figura 4.7: Rendimiento del sistema de visión propuesto (**SoC**) en fotogramas por segundo (*fps*) de acuerdo al número de vectores de flujo óptico. Se ejecutan los algoritmos de Pauwels y Van Hulle, y Raudies y Neumman en su versión simple.

trabajando con un reloj de 200 MHz. llegamos a la conclusión que el procesador está la mayor parte del tiempo esperando a recibir datos de la memoria, por esta razón es necesario diseñar una entidad que tenga acceso directo a los datos contenidos en la memoria **DDR** y que realice las operaciones aritméticas de las rutinas *2, 4 y 5*.

El método *R2-Iterativo* además de ejecutar dos veces el código anterior, necesita filtrar el conjunto de vectores de flujo óptico y para ello se tarda 530 milisegundos; en total el **SoC** tarda 2.5 segundos aproximadamente en procesar el conjunto de vectores de flujo óptico de una secuencia de tamaño 320x240 sí la estimación del *movimiento propio* se realiza totalmente en software. Aunque hemos utilizado algunas técnicas de optimización de código para intentar acelerar el procesamiento, no es posible acelerarlo todo lo que necesitamos, incluso utilizando un conjunto de vectores de flujo óptico reducido (como mínimo 22.000 vectores de flujo según el estudio del apartado 4.3.1), por tal motivo es necesario diseñar un bloque de cálculo que, a través de una conexión directa a la memoria **DDR**, realice las operaciones aritméticas del algoritmo de esti-

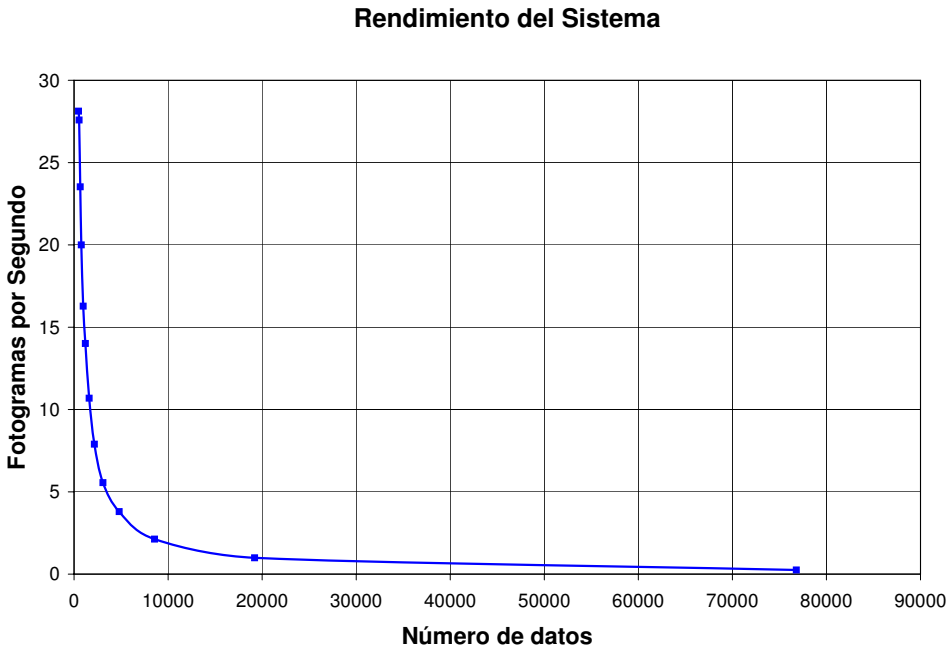


Figura 4.8: Rendimiento del **SoC** para el modelo software de estimación del *movimiento propio*.

mación del *movimiento propio*, de esta forma es posible liberar al **PowerPC** de carga computacional y permitir que sea usado en otras funciones del nivel de media y alta visión.

4.4 Aceleración Hardware del Algoritmo para la Estimación del *Movimiento Propio*

El principal problema del **SoC** es el acceso tan lento que tiene el **PowerPC** a la memoria **DDR**. Este es una restricción de funcionamiento estructural de la arquitectura de la **FPGA** que estamos utilizando, por tal motivo no es posible realizar ningún cambio en nuestro sistema con el fin de aumentar la velocidad de acceso a memoria del **PowerPC**. La forma más sencilla de incrementar la velocidad de proceso es acceder directamente a memoria e implementar las operaciones aritméticas en una entidad de soporte del **PowerPC**. Esta unidad se conecta al procesador a través del **PLB** para recibir instrucciones de ejecución, pero el acceso a memoria lo realiza autónomamente a través

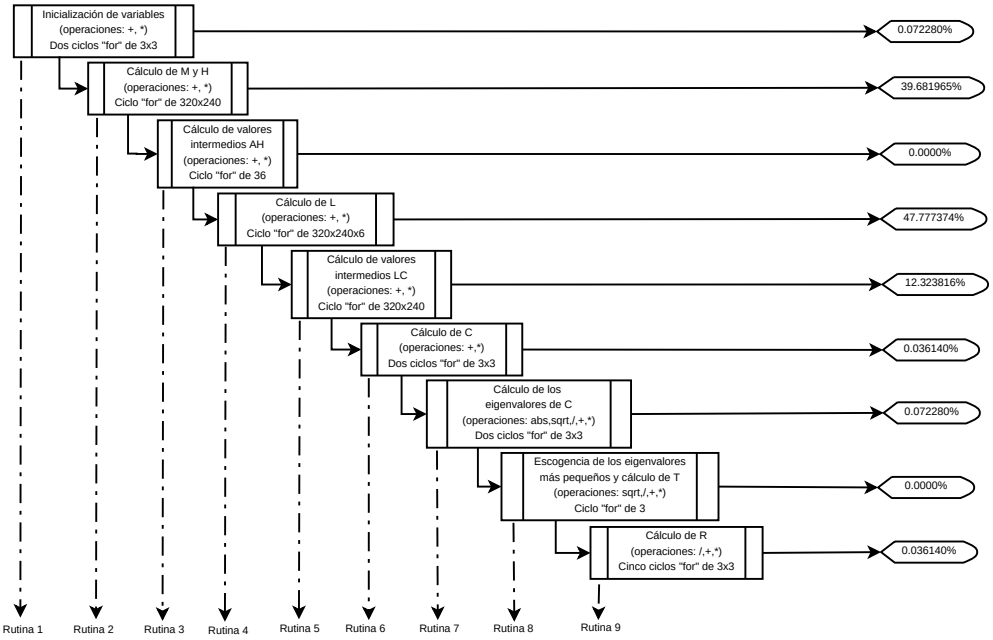


Figura 4.9: Organigrama de las rutinas que se ejecutan para estimar el *movimiento propio*. Estimación de carga computacional de cada rutina.

de la conexión con el **MPMC**. El diagrama de conexiones de la nueva arquitectura se muestra en la Figura 4.10; la entidad de soporte, a diferencia de las demás unidades de procesamiento, está en constante comunicación con el **PowerPC**, pues, aunque es la entidad de soporte la que ejecuta la tarea, es el **PowerPC** el que guía la secuencia de operaciones que deben ser realizadas de acuerdo al código implementado. Debido a la complejidad de esta nueva arquitectura y a su consumo de recursos, fue necesario utilizar la versión de *XIRCA-V4* que tiene la **FPGA** *XC4VFX100-10*.

La unidad de aceleración hardware realiza las rutinas *2, 4 y 5*, asimismo, ejecuta el filtrado del conjunto de vectores de flujo óptico. Todas estas rutinas las realiza de forma secuencial, es decir, el procesador activa la unidad de aceleración para que ejecute una de las rutinas y espera que ésta le devuelva el resultado; el resultado es entregado en la memoria **RAM** y por esta razón está disponible a todas las unidades de procesamiento del sistema. Esta unidad de aceleración sólo realiza sumas, restas y multiplicaciones en punto flotante, ya que su implementación en hardware se puede hacer por medio de caminos segmentados de procesamiento de datos, lo que permite aprovechar al máximo la arquitectura de la **FPGA**. Las operaciones de división y raíz cuadrada, utilizadas

Tabla 4.4: Análisis del código software para la estimación del *movimiento propio*. Cada iteración del algoritmo tarda 1 segundo en evaluar los vectores de flujo de una imagen de 320x240 y no tiene en cuenta el tiempo que tarda en filtrar el conjunto de vectores de flujo óptico.

Rutinas	Tiempo de ejecución (%)	Cantidad de datos procesados
Rutina 1	0.072280	45
Rutina 2	39.681965	307200
Rutina 3	0.000000	18
Rutina 4	47.777374	460800
Rutina 5	12.323816	230400
Rutina 6	0.036140	27
Rutina 7	0.072280	45
Rutina 8	0.000000	6
Rutina 9	0.036140	30

en el cálculo de jacobianos, las sigue realizando el **PowerPC**, pues la *APU* tiene implementadas de forma óptima estas operaciones y, adicionalmente, su utilización no es masiva.

Por medio de la unidad de aceleración hardware hemos podido obtener un sistema en tiempo real para la estimación de *movimiento propio*, en la Figura 4.11 se puede ver la velocidad de ejecución del sistema con el método *R2-Iterativo*. Esta figura demuestra que el sistema propuesto es capaz de estimar el *movimiento propio* con una buena precisión y en tiempo real. Dado que la unidad de aceleración hardware realiza la mayor parte de cálculos matemáticos, es posible utilizar el **PowerPC** en otras funciones, esto supone un punto de partida interesante para seguir desarrollando sistemas empotrados para su aplicación en visión por computador.

Finalmente, en la tabla 4.5 se muestra el consumo de recursos de las dos versiones implementadas del **SoC**, asimismo, se muestra el consumo de potencia de las dos arquitecturas; es necesario aclarar que la arquitectura con la unidad de aceleración hardware ha sido implementada en una **FPGA** *XC4VFX100-10* de Xilinx.

Varias conclusiones se pueden sacar a la luz de los resultados obtenidos. Del estudio realizado en el apartado 4.3.1 queda claro que el modelo *R2-Iterativo* funciona correctamente con una pequeña cantidad de vectores de flujo óptico, incluso, en presencia de ruido; adicionalmente, la unidad para el cálculo de flujo óptico entrega resultados

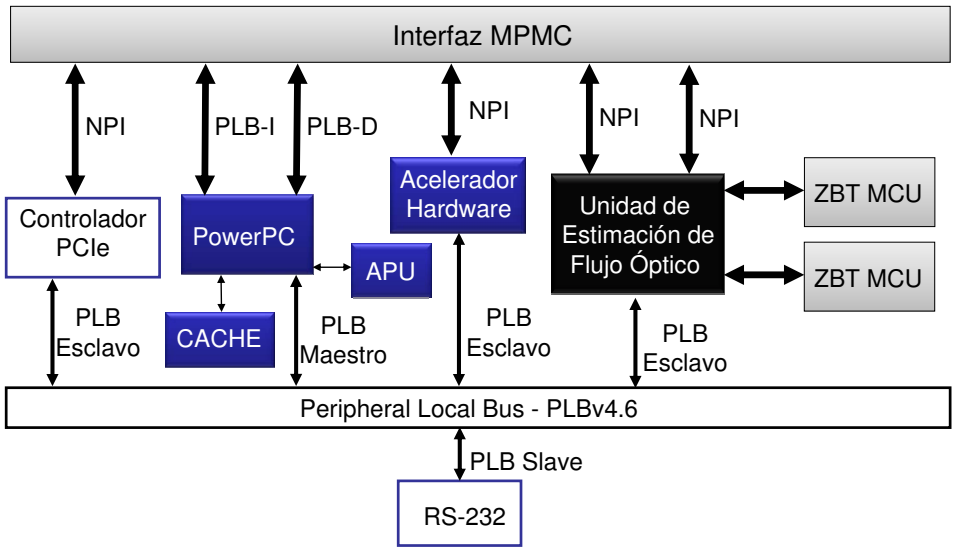


Figura 4.10: Diagrama de conexiones de la arquitectura del **SoC** con la unidad de aceleración hardware.

más precisos si la densidad de datos se disminuye, es decir, si escogemos un umbral de precisión muy alto, entonces la unidad para el cálculo de flujo óptico entrega unos resultados mucho más precisos pero a expensas de una menor cantidad de vectores de flujo óptico o lo que es lo mismo, menos vectores de flujo pero con errores más bajos. Gracias al módulo acelerador hemos aumentado el número de puntos que se pueden computar por segundo en un factor de 25. Aunque la unidad de aceleración hardware ha incrementado significativamente la velocidad de ejecución del sistema, no podemos decir que el sistema puede ejecutar el modelo *R2-Iterativo* en tiempo real cuando el número de vectores de flujo supera los 22.000 (ver Figura 4.11), sin embargo, este número de vectores de flujo óptico es suficiente para decir que nuestro **SoC** es capaz de estimar el *movimiento propio* en tiempo real y está sustentado en lo antes mencionado. Asimismo, los resultados contenidos en la tabla 4.5 demuestran que nuestro sistema consume muy poca potencia en comparación con sistemas similares implementados en ordenadores convencionales o en **GPU**s, así, basados en estos resultados y en que el sistema está totalmente implementado en un sólo chip, podemos decir que nuestro sistema es idóneo para ser utilizado en aplicaciones empotradas como por ejemplo de automoción, aviónica o robótica.

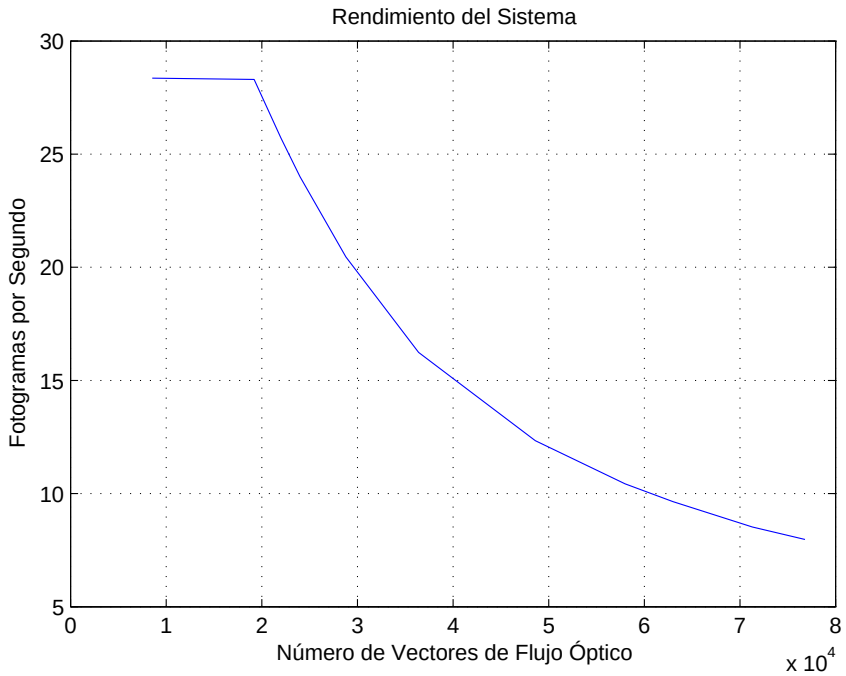


Figura 4.11: Rendimiento del **SoC** definitivo y ejecutando el modelo *R2-Iterativo*.

Tabla 4.5: Consumo de potencia y de recursos de las dos versiones del sistema para la estimación de *movimiento propio*.

Sistema	Versión de la **FPGA**	Recursos Hardware			Consumo de Potencia
		Entidad	Utilizados	Disponibles	
Sin Aceleración	*XC4VFX60-10*	*Flip-Flops*	36.062	50.560	3,5 *W*
		LUTs	40.614	50.560	
		Slices	25.090	25.280	
		RAMB16s	127	232	
		DSP48s	120	128	
Con Aceleración	*XC4VFX100-10*	*Flip-Flops*	67.856	84.352	4,1 *W*
		LUTs	78.512	84.352	
		Slices	41.754	42.176	
		RAMB16s	239	376	
		DSP48s	158	160	

5

Discusión de Resultados

En este capítulo se hace un resumen de los resultados obtenidos durante el desarrollo de esta tesis; se enumeran las publicaciones que se desprenden del trabajo realizado, se describen las sinergias suscitadas por el trabajo en grupo realizado en el marco de los proyectos **DRIVSCO**, DINAM-VISION y MULTIVISION y se hace un compendio de todos los diseños hardware que hemos realizado en el marco de esta tesis.

5.1 Resultados

En el desarrollo de esta tesis se abordaron diferentes desafíos y se presentaron soluciones a cada uno de ellos. El trabajo realizado en el marco de los proyectos **DRIVSCO** (IST-016276-2), DINAM-VISION (DPI2007-61683) y MULTIVISION (TIC-3873) permitió encarar los desafíos desde diferentes perspectivas gracias al trabajo en grupo, las sinergias dieron como resultado sistemas más complejos y robustos que se reflejan en las publicaciones realizadas a la fecha, a continuación se hace un compendio de todas las publicaciones, algunas enviadas y otras aceptadas, por orden de relevancia.

Revistas:

[1] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multi-port abstraction layer for FPGA intensive memory exploitation applications, Journal of Systems Architecture In Press, Accepted Manuscript (2010) –, URL `http://www.sciencedirect.com/science/article/B6V1F-50412G1-2/2/47c37daf412ec52d7e185359ebf85965`. `doi:DOI:10.1016/j.sysarc.2010.05.007`

[2] M. Tomasi, F. Barranco, M. Vanegas, J. Díaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, accepted and under publication in IEEE Trans. on CSVT. (2010)

[3] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing. (2009)

[4] M. Tomasi, F. Barranco, M. Vanegas, J. Díaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture. (2009)

Congresos:

[1] M. Vanegas, L. Rubio, M. Tomasi, J. Díaz, E. Ros, On-chip ego-motion estimation based on optical flow, submitted to International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. SAMO X. (2010)

[2] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, ISIE2010. Accepted for IEEE Internetional Symposium on Industrial Electronics, Bari (Italy). (July 2010)

[3] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 295–304

[4] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Díaz, Entorno software para visualización y configuración de procesamientode imágenes en tiempo real con plataformas reconfigurables, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 327–336

[5] J. Gómez López, M. Vanegas Hernández, C. Morillas Gutiérrez, M. López Górdo, F. Pelayo Valle, Implementación hardware-software de operadores morfológicos en powerpc, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 517–526

Es de resaltar que esta tesis abordó temas de baja y media visión, sin embargo, en el nivel de baja visión nuestro aporte se centró en el diseño de una plataforma que permitiera el procesamiento de algoritmos con accesos masivos a memoria, pero el estudio y el desarrollo de arquitecturas específicas ha sido realizado por otros integrantes del grupo de trabajo. La Figura 1.3 muestra de forma esquemática el trabajo realizado en el marco de esta tesis. Nosotros, a lo largo del texto, hemos descrito nuestro aporte desde la perspectiva del nivel de media visión, es decir, este texto da cuenta de la arquitectura desarrollada para la estimación del *movimiento propio* de una cámara en un escenario rígido, pero más allá de esta contribución, las contribuciones que más han trascendido a los demás integrantes del grupo han sido el diseño del controlador de memoria **MCU**, este componente se ha realizado pensando en los modelos hardware de los demás integrantes ya que cumple con los requerimientos impuestos por los complejos algoritmos en el nivel de baja visión, y de las interfaces de conexión con periféricos externos como la **PCIe** y las memorias **DDR**; asimismo, fue diseñada una unidad de procesamiento encargada de deformar imágenes (*unidad de deformación*) y que es utilizada como módulo de deformación en la arquitectura de múltiples escalas para el cálculo de flujo óptico [63] y como módulo de rectificación de imágenes en la arquitectura

para la estimación de disparidad en sistemas de visión estéreo [84].

En general, las funciones de acceso a memoria y de conexión con el computador a través de la interfaz **PCIe** sobre la plataforma de desarrollo *Xirca-V4*, en la que se basan los proyectos **DRIVSCO**, DINAM-VISION y MULTIVISION fueron desarrolladas en el marco de esta tesis, pero hemos centrado el desarrollo de este texto en la metodología utilizada para obtener un sistema de procesamiento de vídeo en el nivel de media visión, en el cual es posible aprovechar las ventajas del procesamiento paralelo que ofrecen las **FPGA** y de flexibilidad de descripción que ofrecen los procesadores, para estimar el *movimiento propio* en tiempo real. Con el fin de exponer adecuadamente los resultados de esta tesis, hemos dividido el texto en cinco partes, en el apartado 5.2 se mostrarán las ventajas del controlador de memoria para accesos paralelos y masivos que hemos desarrollado, en el apartado 5.3 se discutirá al respecto de los resultados obtenidos con la arquitectura de procesamiento que hemos desarrollado para la estimación de *movimiento propio*, en el apartado 5.4 se describe la formación transversal adquirida durante el trabajo realizado, en el apartado 5.5 se describen los trabajos futuros que se desprenden de esta primera aproximación a un sistema de procesamiento independiente y en tiempo real para el procesamiento de algoritmos en el nivel de media visión y para finalizar, en el apartado 5.6, se realizan las conclusiones del presente trabajo y en el apartado 5.7, se enumeran las principales aportaciones de esta tesis.

5.2 Controlador de Memoria (MCU)

Se ha demostrado la versatilidad de usar la **MCU** en sistemas con accesos a memoria que son dependientes de datos. La **MCU** utiliza interfaces sencillas que son muy útiles en arquitecturas con caminos de datos segmentados. Asimismo, no es necesario utilizar métodos complicados para integrar la **MCU** en cualquier tipo de sistema hardware.

Hemos diseñado una **MCU** capaz de abstraer los accesos a memorias externas de múltiples entidades de procesamiento trabajando en paralelo. Hemos diseñado un eficiente programador de accesos que es capaz de garantizar los accesos a memoria por ciclo de reloj. La alta frecuencia de trabajo de la **MCU** permite a todas las entidades de procesamiento conectadas tener un acceso optimizado a la memoria **SSRAM** de tipo **ZBT**.

Los caminos de datos segmentados y con unidades de procesamiento superescalares

pueden beneficiarse mucho del acceso a memoria mediante los puertos **AAP**; la forma en que estas interfaces fueron hechas permiten ajustarse adecuadamente a las arquitecturas de procesamiento segmentadas, lo que confiere una gran flexibilidad de uso a una gran variedad de sistemas.

El sistema de procesamiento de imágenes escogido para testear la **MCU** demuestra la versatilidad del sistema, la capacidad para cambiar de estado, lectura o escritura, por ciclo de reloj de la **MCU** la convierte en una solución muy interesante para el control de memorias externas en sistemas basados en **FPGA**s. Asimismo, la integración de la **MCU** en el entorno de diseño del lenguaje *Handel-C* se muestra como un ejemplo y como una prueba de la fácil utilización de ésta en entornos de diseño basados en lenguajes de descripción de hardware de alto nivel.

Finalmente, hemos ilustrado, con un ejemplo de procesamiento de imágenes (la unidad de deformación), como el uso de la **MCU** puede conseguir mejoras significativas en el rendimiento de sistemas con accesos masivos y paralelos a memorias externas y, además de las mejoras en rendimiento, su utilización reduce apreciablemente el tiempo de diseño de arquitecturas con múltiples caminos segmentados de procesamiento con accesos paralelos a memoria, ya que el diseñador no tiene que preocuparse por programar explícitamente los accesos a memoria.

5.3 Estimación de *Movimiento Propio*

Hemos introducido una novedosa arquitectura para un sistema de procesamiento de vídeo en el nivel de baja y media visión. Este sistema puede ejecutar cualquier algoritmo para la estimación de *movimiento propio* basado en estimaciones de flujo óptico, ya que por su arquitectura modular es capaz de estimar flujo óptico por medio de entidades hardware y el *movimiento propio* por medio de software que se ejecuta en un **PowerPC**. Su versatilidad ha sido demostrada ejecutando los algoritmos de Pauwels y Van Hulle [57], y Raudies y Neumman [58].

El **SoC** propuesto inicialmente ha sido modificado para incrementar su capacidad computacional, por medio de una entidad de soporte matemático ha sido posible incrementar el rendimiento del sistema, además es posible acceder de forma más rápida a los datos en memoria **DDR**.

La capacidad de cómputo del sistema propuesto no puede compararse con la capacidad de las **CPU**s y **GPU**s actuales, sin embargo, hemos demostrado que nuestro sistema consume muy poca energía para realizar su tarea, esto lo hace idóneo para ser utilizado en sistemas empotrados en aplicaciones de navegación en general.

Hemos demostrado que nuestro modelo para la estimación de *movimiento propio* es preciso y adecuado para implementaciones hardware, ya que permite definir desde el método mismo el número máximo de iteraciones para solucionar el problema de la minimización de la *restricción bilineal*.

Aunque muy superficialmente, hemos demostrado que nuestro sistema puede detectar **IMO**s, sin embargo, el problema de la detección de éstos es muy complejo y requiere de información que, a priori, no puede entregar nuestro sistema, como por ejemplo la reconstrucción 3-D de una escena bien sea a partir del flujo o de un sistema de visión estéreo.

Dada la problemática que se deriva de la estimación de flujo óptico y que en principio ha sido solventado con el método *R2-Iterativo* que es en gran medida inmune al ruido y robusto ante valores atípicos, nuestro sistema ha sido desarrollado de tal forma que diferentes métodos para la estimación de flujo óptico pueden ser utilizados. De las sinergias del grupo de trabajo de **DRIVSCO**, DINAM-VISION y MULTIVISION varias entidades, basadas en diferentes modelos para la estimación de flujo óptico, han sido desarrolladas, entre ellas se encuentran el método de fase utilizando filtros *steerable* que utiliza una menor cantidad de recursos hardware que los filtros de *Gabor* y el método de Lukas y Kanade [89] que utiliza la menor cantidad de recursos de todos los anteriores y que se basa en la intensidad. Adicionalmente, todos estos métodos están implementados también en su versión con múltiples escalas lo que permite una mayor precisión en las estimaciones de flujo óptico. Nuestro sistema puede utilizar indistintamente cualquiera de estas entidades y sacar provecho de lo que cada una ofrece, es decir, si utilizamos por ejemplo Lukas y Kanade podemos implementar nuestro sistema en una **FPGA** más pequeña, o en su defecto utilizar los recursos hardware liberados para conectar una entidad de proceso adicional, todo esto al coste de una menor precisión en las estimaciones de flujo óptico.

5.4 Formación Transversal

El trabajo desarrollado en esta tesis ha implicado una amplia formación en el campo de visión por computador (modelos de estimación de flujo óptico, patrón de *movimiento propio*, **IMO**s) y en el campo de diseño de arquitecturas de propósito específico basadas en **FPGA**s. Además el diseño de un sistema híbrido que integra módulos hardware y software ha permitido explorar técnicas de co-diseño (por ejemplo en el diseño del módulo acelerador hardware tras evaluar la carga computacional de submódulos del algoritmo objetivo). La arquitectura propuesta en este texto es de alta complejidad y sus prestaciones dependen de elementos de computación y, en gran medida, de control de memoria externa. Para ello además ha sido necesario el desarrollo de componentes de comunicación de altas prestaciones como la unidad **MCU** para memoria externa **SSRAM**. EL trabajo que se presenta aquí ha requerido la integración de módulos hardware de propósito específico (circuito de estimación de flujo óptico) de altas prestaciones con procesadores empotrados en un mismo chip. Todo ello constituye un **SoC** de alta complejidad. Además se ha evaluado cómo las prestaciones del sistema dependen fundamentalmente de qué tan bien se acoplen módulos implementados en hardware con módulos software. El trabajo en este campo ha requerido una formación intensa a nivel técnico con distintas herramientas de diseño de circuitos, simulación, evaluación de modelos matemáticos, etc. Por todo lo anterior, se ha adquirido experiencia con plataformas como MATLAB (para evaluación de modelos de visión), programación en C plano (para implementación de módulos software), plataformas de diseño de circuitos (como el ISE de Xilinx), plataforma de diseño de sistemas empotrados en FPGA (herramienta EDK de Xilinx), herramientas de simulación (como MODELSIM), etc. De hecho, durante la realización de la tesis, gran parte del trabajo ha sido técnico de puesta a punto de herramientas, validación de circuitos, etc.

Por último, resaltar que la intensa labor de trabajo en equipo realizada en los proyectos **DRIVSCO**, DINAM-VISION y MULTIVISION ha exigido ir más allá del mero diseño de los circuitos, ha sido necesaria su integración con otros componentes desarrollados por otros miembros del grupo de investigación, documentación para entregables de proyectos (para revisión de proyectos), evaluación exhaustiva de prestaciones, etc. Todo esto ha exigido una metodología de trabajo con plazos de entrega precisos, integración de componentes de distintos investigadores, etc., es decir, una metodología de trabajo

en grupo intenso encaminada a unos objetivos comunes en el marco de proyectos de investigación nacionales y Europeos; todo esto no es fácilmente visible en los resultados obtenidos, pero sí en la formación transversal adquirida.

5.5 Trabajos Futuros

La estabilización de imágenes es un problema de investigación actual y en el cual nuestro sistema puede aportar mucho. Morimoto y Chellappa [90] propusieron un método para estabilizar imágenes basados en la estimación de movimiento con el método de varias escalas de resolución; a diferencia de este trabajo, nuestro sistema aporta la estimación de *movimiento propio* por medio de la cual se puede restringir el movimiento de las imágenes sabiendo la cantidad de fotogramas por segundo que se analizan. Ertürk [91] propuso estabilizar imágenes por medio de filtros de Kalman, si utilizamos como hipótesis de partida que filtrar todos los píxeles de una imagen como lo hace Ertürk es muy costoso computacionalmente y que en cambio filtrar los vectores T y R es en principio menos costoso, nuestro sistema podría realizar esta tarea sin necesidad de adicionar más hardware. La implementación del filtro de Kalman para los vectores T y R se puede hacer en el **PowerPC** y esta información se puede transmitir de regreso al módulo para la estimación de flujo óptico para compensar el movimiento.

Un trabajo que queda a medio camino es la detección de **IMO**s, como se ha explicado en el apartado 3.5 la detección de objetos con movimiento independiente tiene diferentes grados de dificultad y, de acuerdo al tipo de aplicación, necesita de sistemas complejos. En nuestro caso, y debido a las sinergias dentro del grupo de trabajo de **DRIVSCO**, DINAM-VISION y MULTIVISION, es algo inmediato intentar construir un sistema por medio del cual poder detectar cualquier tipo de **IMO**. Para esto es necesario utilizar una entidad adicional que nos permita estimar la profundidad de la escena, dentro del proyecto **DRIVSCO** se ha diseñado también un bloque capaz de estimar la disparidad a partir de un sistema de visión estéreo, si incorporamos este bloque a nuestro sistema podremos detectar de forma robusta **IMO**s. El problema radica en que el sistema que hemos implementado ocupa todo el espacio disponible de la **FPGA** *XC4VFX100-10*, por tal motivo es necesario evaluar si el método de Lukas y Kanade permite obtener los mismos resultados que hemos obtenido con nuestro sistema, de confirmarse los resultados, entonces podríamos ahorrar una gran cantidad de recursos hardware que pueden

ser utilizados para implementar el bloque para la estimación de la disparidad.

A futuro, y de concretarse el anuncio hecho por Xilinx y ARM Holdings en el cual dicen que Xilinx está trabajando en una nueva arquitectura basada en un procesador **ARM** *Cortex-9* central con conexiones de hardware a través de buses especiales de alta velocidad, nuestro sistema podrá ver incrementado su rendimiento simplemente por tener un procesador mucho más potente y con una velocidad de hasta 800 MHz. El diseño ha sido realizado totalmente en **VHDL**, exceptuando la entidad para el cálculo de flujo óptico, y utilizando bloques estándar lo que permite una fácil adaptación a diferentes **FPGA**s. Adaptar esta arquitectura a las nuevas **FPGA**s de Xilinx nos permitirá ser mucho más competitivos y a su vez realizar diseños mucho más ambiciosos.

5.6 Conclusiones

En este trabajo se ha desarrollado una plataforma híbrida Hardware/Software para procesamiento de imágenes en tiempo real. Se ha implementado un modelo de estimación de *movimiento propio* (ego-movimiento) a partir de estimaciones de flujo óptico extraídas de un circuito hardware de propósito específico (IP core). Este modelo permite además segmentar **IMO**s (Objetos con Movimiento Independiente) en tiempo real. Para ello se ha adoptado una estrategia de diseño híbrido hardware/software implementando los componentes de procesamiento intensivo local en hardware específico y los elementos de cálculo iterativo como módulos software en procesadores empotrados en el mismo chip.

El desarrollo de esta plataforma ha requerido el diseño de un controlador de memoria *SSRAM* que abstrae al diseñador de circuitos con caminos de datos segmentados de la necesidad de temporizar adecuadamente los accesos a memoria. Este componente es crítico para arquitecturas de cálculo intensivo con necesidad de soporte de memoria externa para almacenamiento temporal de datos.

5.7 Aportaciones Principales

En el marco de esta tesis se destacan las siguientes aportaciones principales:

- Diseño e implementación de la plataforma **DRIVSCO**, en la cual se basó el desarrollo de los diferentes sistemas para el procesamiento de vídeo en tiempo

real.

- Diseño e implementación de un controlador de memoria de altas prestaciones para aplicaciones con accesos masivos y aleatorios a memorias **SSRAM** externas en plataformas basadas en **FPGA**.

- Diseño e implementación de un sistema para la deformación de imágenes que puede ser utilizado como módulo para rectificación de imágenes y como módulo de compensación de movimiento en sistemas de procesamiento de múltiples escalas.

- Diseño e implementación de un sistema híbrido hardware/software para el procesamiento de imágenes en tiempo real. Este sistema permite calcular flujo óptico por medio de hardware y analizar esta información por medio de un procesador empotrado en el mismo chip (**FPGA**).

- Diseño e implementación de hardware acelerador para procesadores **PowerPC**. Para ello se ha analizado la carga computacional de cada submódulo del modelo de estimación de *movimiento propio* y se han implementado en hardware específico los módulos que requieren mayor carga computacional. Por medio de este hardware es posible incrementar la velocidad de ejecución de los procesadores empotrados en una **FPGA**. En concreto este módulo hardware aumenta la capacidad de cómputo por segundo del sistema en un factor de 13.5, esto se puede concluir de las figuras 4.8 y 4.11.

- Diseño e implementación de una conexión eficiente **PCIe**. Este módulo permite desarrollar sistemas híbridos hardware/software, donde los módulos software pueden ejecutarse en el procesador del computador en el que se aloja la placa de co-procesamiento basada en **FPGA**.

- Diseño e implementación de un sistema para la estimación del *movimiento propio* de una cámara en tiempo real en un entorno rígido. El sistema es capaz de procesar 25 imágenes por segundo a una resolución de 320x240 y estimar el campo de flujo óptico y a partir de éste, tomando un subconjunto con 22.000 vectores de flujo óptico, es capaz de estimar el *movimiento propio*. En conclusión, el sistema es capaz de estimar el *movimiento propio* para imágenes de 320x240 a una velocidad de 25 fotogramas por segundo.

- Diseño e implementación de un sistema para la detección de **IMO**s. Este sistema es capaz de detectar vecindades cuyo patrón de movimiento no se corresponde con el *movimiento propio* y clasificarlas como **IMO**s. Esto es factible para **IMO**s sencillos (fácilmente separables del patrón de *movimiento propio*).

Bibliografía

[1] M. Tomasi, Pyramidal architecture for stereo vision and motion estimation in real-time FPGA-based devices, Ph.D. thesis, University of Granada (2010). xvii, 40, 44

[2] D. Marr, Vision: A Computational Investigation into the Human Representation and Processing of Visual Information, Henry Holt and Co., Inc., New York, NY, USA, 1982. 3

[3] C. Weems, Architectural requirements of image understanding with respect to parallel processing, Proceedings of the IEEE 79 (4) (1991) 537 –547. doi:10.1109/5.92046. 3

[4] N. Ratha, A. Jain, Computer vision algorithms on reconfigurable logic arrays, Parallel and Distributed Systems, IEEE Transactions on 10 (1) (1999) 29 –43. doi:10.1109/71.744833. 3

[5] M. Anguita, J. Díaz, E. Ros, F. J. Fernández-Baldomero, Optimization strategies for high-performance computing of optical-flow in general-purpose processors, IEEE Trans. Cir. and Sys. for Video Technol. 19 (10) (2009) 1475–1488. doi:http://dx.doi.org/10.1109/TCSVT.2009.2026821. 4, 17, 58

[6] K. Pauwels, M. M. Van Hulle, Realtime phase-based optical flow on the GPU, in: Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on, 2008, pp. 1–8. doi:10.1109/CVPRW.2008.4563090. 4, 58

[7] Y. Murachi, Y. Fukuyama, R. Yamamoto, J. Miyakoshi, H. Kawaguchi, H. Ishihara, M. Miyama, Y. Matsuda, M. Yoshimoto, A VGA 30-fps realtime optical-flow

processor core for moving picture recognition, IEICE Transactions on Electronics E91 (4) (2008) 457 – 464.
URL http://ietele.oxfordjournals.org/cgi/content/short/E91-C/4/457 4

[8] J. Díaz, E. Ros, F. Pelayo, E. M. Ortigosa, S. Mota, FPGA-based real-time optical-flow system, Circuits and Systems for Video Technology, IEEE Transactions on 16 (2) (2006) 274–279. doi:10.1109/TCSVT.2005.861947. 4, 16

[9] J. Díaz, E. Ros, S. P. Sabatini, F. Solari, S. Mota, A phase-based stereo vision system-on-a-chip, Biosystems 87 (2-3) (2007) 314 – 321, papers presented at the Sixth International Workshop on Information Processing in Cells and Tissues, York, UK, 2005 - IPCAT 2005, Information Processing in Cells and Tissues. doi:DOI:10.1016/j.biosystems.2006.09.028.
URL http://www.sciencedirect.com/science/article/B6T2K-4KVP1BT-8/2/67c98151d57bfbcbf81916f14571ab54 4, 16

[10] J. Díaz, E. Ros, R. Agís, J. Bernier, Superpipelined high-performance optical-flow computation architecture, Computer Vision and Image Understanding 112 (3) (2008) 262–273. doi:10.1016/j.cviu.2008.05.006. 4, 17

[11] A. R. Bruss, B. K. P. Horn, Passive navigation, Computer Vision, Graphics, and Image Processing 21 (1) (1983) 3–20. 4, 40, 46, 47

[12] D. A. Forsyth, J. Ponce, Computer Vision: A Modern Approach, Prentice Hall Professional Technical Reference, 2002. 4

[13] V. Vineet, P. J. Narayanan, CUDA cuts: Fast graph cuts on the GPU, Computer Vision and Pattern Recognition Workshop 0 (2008) 1–8. doi:http://doi.ieeecomputersociety.org/10.1109/CVPRW.2008.4563095. 7

[14] I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, in: FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays, ACM, New York, NY, USA, 2006, pp. 21–30. doi:http://doi.acm.org/10.1145/1117201.1117205. 7, 16

[15] E. M. Ortigosa, A. C. nas, E. Ros, P. M. Ortigosa, S. Mota, J. Díaz, Hardware description of multi-layer perceptrons with different abstraction levels, Microprocessors and Microsystems 30 (7) (2006) 435 – 444. doi:10.1016/j.micpro.2006.03.004.
URL http://www.sciencedirect.com/science/article/B6V0X-4JTR4JY-1/2/ac09ca29a75b144779fe9a4d23640b8f 8, 35

[16] J. Díaz Alonso, Multimodal bio-inspired vision system. high performance motion and stereo processing architecture, Ph.D. thesis, University of Granada (2006). 9

[17] Z. Guzik, R. Jacobsson, Beam phase and intensity monitor (BPIM) for the LHCb experiment. oai:cds.cern.ch:989402. proposal for a beam phase and intensity monitor for the LHCb experiment, Tech. Rep. LHCb-2006-055. CERN-LHCb-2006-055, CERN, revised version submitted on 2006-10-26 18:44:08 (Oct 2006). 16

[18] R. Jacobsson, Building integrated remote control systems for electronics boards, in: Real-Time Conference, 2007 15th IEEE-NPSS, 2007, pp. 1–6. doi:10.1109/RTC.2007.4382741. 16

[19] W. Stallings, Computer Organization and Architecture: Designing for Performance, 7/E, seventh Edition, Prentice Hall, Boston, MA, USA, 2005. 16

[20] S. Carr, K. S. McKinley, C.-W. Tseng, Compiler optimizations for improving data locality, SIGPLAN Not. 29 (11) (1994) 252–262. doi:http://doi.acm.org/10.1145/195470.195557. 16

[21] V. Phalke, B. Gopinath, An inter-reference gap model for temporal locality in program behavior, in: SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, ACM, New York, NY, USA, 1995, pp. 291–300. doi:http://doi.acm.org/10.1145/223587.223620. 16

[22] P. Cobos, F. Monasterio, FPGA implementation of camus correlation optical flow algorithm for real time images, in: Vision Interface Proceedings VI2001, 14th International Conference on Vision Interface, Canada, 2001, pp. 32–38. 17

[23] System Generator, real time system models, an easy way to perform very early testing and verification of hardware and software integration (2000) [cited 9 march 2009].
URL http://www.arm.com/products/DevTools/ 17

[24] Mitrion SDK 2.0.2, mitrion software development kit, software Acceleration Platform with FPGA acceleration coprocessor modules from leading vendors to deliver solutions for running and developing accelerated applications (2001) [cited 29 april 2009].
URL http://www.mitrionics.com/ 17

[25] Agility design solutions corporate, AGILITY, shorten time to develop and deploy complex image and signal processing systems (2008) [cited 3 October 2008].
URL http://www.agilityds.com 17

[26] Impulse C™, software-to-hardware compile (2003) [cited 30 april 2009].
URL http://www.impulseaccelerated.com/ 17

[27] Catapult C synthesis, high Level Synthesis tool for ASIC and FPGA hardware designers (1981) [cited 14 november 2008].
URL http://www.mentor.com/products/ 17

[28] Q. Liu, G. Constantinides, K. Masselos, P. Cheung, Automatic on-chip memory minimization for data reuse, in: Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on, 2007, pp. 251–260. doi:10.1109/FCCM.2007.18. 18

[29] P. Ranganathan, S. Adve, N. P. Jouppi, Reconfigurable caches and their application to media processing, in: Computer Architecture, 2000. Proceedings of the 27th International Symposium on, Farmington, Pennsylvania, USA, 2000, pp. 214–224. 18

[30] G. S. Sohi, M. Franklin, High-bandwidth data memory systems for superscalar processors, SIGOPS Oper. Syst. Rev. 25 (Special Issue) (1991) 53–62. doi:http://doi.acm.org/10.1145/106974.106980. 18

[31] S. Heithecker, A. do Carmo Lucas, R. Ernst, A mixed QoS SDRAM controller for fpga-based high-end image processing, in: Signal Processing Systems, 2003. SIPS 2003. IEEE Workshop on, 2003, pp. 322–327. 18

[32] S.-S. Ang, G. A. Constantinides, W. Luk, P. Y. K. Cheung, Custom parallel caching schemes for hardware-accelerated image compression, Journal of Real-Time Image Processing 3 (4) (2008) 289–302. doi:10.1007/s11554-008-0082-0. 18

[33] Intel ® Core™ i7 processor, 45nm Hi-k next generation intel core microarchitecture, intel's latest-generation microarchitecture (2008) [cited 03 April 2009].
URL http://www.intel.com/technology/architecture-silicon/ 19

[34] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, NVIDIA tesla: A unified graphics and computing architecture, IEEE Micro 28 (2) (2008) 39–55. doi:http://doi.ieeecomputersociety.org/10.1109/MM.2008.31. 19

[35] NVIDIA ® tesla™ GPU, tesla computing solution, the world's first teraflop many-core processor. (2009) [cited 18 may 2009].
URL http://www.nvidia.com/object/tesla_computing_solutions.html 19

[36] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, P. G. Kjeldsberg, Data and memory optimization techniques for embedded systems, ACM Trans. Des. Autom. Electron. Syst. 6 (2) (2001) 149–206. doi:http://doi.acm.org/10.1145/375977.375978. 19

[37] Xilinx Inc., Xilinx, fPGA manufacturer. (1984) [cited 10 January 2007].
URL http://www.xilinx.com 19, 20, 21

[38] Seven solution s.l., sevensols, seven Solutions is a technology-based company specialized in design of processing systems (2006) [cited 23 October 2008].
URL http://www.sevensols.com/index.php?seccion=262&subseccion=268 20, 59

[39] Open SystemC initiative, SystemC™, the Open SystemC Initiative (OSCI) (2005) [cited 12 November 2008].
URL http://www.systemc.org 24

[40] M. Sutaone, S. Badwaik, Performance evaluation of VHDL coding techniques for optimized implementation of ieee 802.3 transmitter, IET Conference Publications 2008 (CP535) (2008) 287–293. doi:10.1049/cp:20080199.
URL http://link.aip.org/link/abstract/IEECPS/v2008/iCP535/p287/s1 24

[41] M. Cassel, F. L. Kastensmidt, Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs, in: IOLTS '06: Proceedings of the 12th IEEE International Symposium on On-Line Testing, IEEE Computer Society, Washington, DC, USA, 2006, pp. 139–144. doi:http://dx.doi.org/10.1109/IOLTS.2006.32. 24

[42] Xilinx Inc, Synthesis and Simulation Design Guide, Chapter 5, Encoding State Machines. (2000). 25

[43] Open cores community, OpenCores, hardware Projects Repository (1999) [cited 20 July 2008].
URL http://www.opencores.org/browse.cgi/filter/category_memory 28

[44] D. J. Fleet, A. D. Jepson, M. R. M. Jenkin, Phase-based disparity measurement, CVGIP: Image Underst. 53 (2) (1991) 198–210. doi:http://dx.doi.org/10.1016/1049-9660(91)90027-M. 32, 42, 43

[45] K. Pauwels, M. M. Van Hulle, Optic flow from unstable sequences containing unconstrained scenes through local velocity constancy maximization, in: British Machine Vision Conference (BMVC 2006), Edinburgh, Scotland, 2006, pp. 397–406. 32, 43

[46] H. K. Nishihara, Practical real-time imaging stereo matcher, Optical engineering 23 (5) (1984) 536 – 545. 32

[47] Y. I. Abdel-Aziz, H. M. Karara, Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry, in: Proc. ASP/UI Symp. Close-Range Photogrammetry, Urbana, IL, USA, 1971, pp. 1–18. 32

[48] D. V. Papadimitriou, T. J. Dennis, Epipolar line estimation and rectification for stereo image pairs, Image Processing, IEEE Transactions on 5 (4) (1996) 672–676. doi:10.1109/83.491345. 32

[49] G. Wolberg, Digital Image Warping, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994. 33

[50] Celoxica Company, Handel-C Language Reference Manual, Version 3.1 (2003). 35

[51] Open rtVision (real-time visualization environment for image processing reconfigurable devices), open source platform developed to evaluate FPGA implementations of image processing architectures. (2009) [cited 23 February 2010].
URL http://code.google.com/p/open-rtvision/ 37

[52] B. K. Horn, B. G. Schunck, Determining optical flow, Artificial Intelligence 17 (1-3) (1981) 185 – 203. doi:DOI:10.1016/0004-3702(81)90024-2.
URL http://www.sciencedirect.com/science/article/B6TYF-47YY2FX-4F/2/a770df320b9401cc331ad3dfcf8e29b0 40

[53] J. L. Barron, D. J. Fleet, S. S. Beauchemin, Performance of optical flow techniques, International Journal of Computer Vision 12 (1) (1994) 43 – 77. doi:DOI:10.1.1.126.2350.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.126.2350 40

[54] D. J. Heeger, A. D. Jepson, Subspace methods for recovering rigid motion I: algorithm and implementation, Int. J. Comput. Vision 7 (2) (1992) 95–117. doi:http://dx.doi.org/10.1007/BF00128130. 40

[55] A. Chiuso, R. Brockett, S. Soatto, Optimal structure from motion: Local ambiguities and global estimates, Int. J. Comput. Vision 39 (3) (2000) 195–228. doi:http://dx.doi.org/10.1023/A:1026563712076. 41

[56] T. Zhang, C. Tomasi, Fast, robust, and consistent camera motion estimation, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 1 (1999) 1164. doi:http://doi.ieeecomputersociety.org/10.1109/CVPR.1999.786934. 41

[57] K. Pauwels, M. M. V. Hulle, Optimal instantaneous rigid motion estimation insensitive to local minima, Computer Vision and Image Understanding 104 (1) (2006) 77–86. doi:DOI:10.1016/j.cviu.2006.07.001.

URL http://www.sciencedirect.com/science/article/B6WCX-4KPFKW2-1/2/408805b3b92b00ce3c73b8a188fa20ae 41, 42, 46, 49, 83

[58] F. Raudies, H. Neumann, An efficient linear method for the estimation of ego-motion from optical flow, in: Pattern Recognition, Vol. 5748 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 11–20. 42, 46, 51, 83

[59] D. J. Fleet, A. D. Jepson, Computation of component image velocity from local phase information, International Journal of Computer Vision 5 (1) (1990) 77 –104. doi:10.1007/BF00056772. 42

[60] D. Fleet, A. Jepson, Stability of phase information, Pattern Analysis and Machine Intelligence, IEEE Transactions on 15 (12) (1993) 1253 –1268. doi:10.1109/34.250844. 42

[61] T. Gautama, M. Van Hulle, A phase-based approach to the estimation of the optical flow field using spatial filtering, Neural Networks, IEEE Transactions on 13 (5) (2002) 1127–1136. doi:10.1109/TNN.2002.1031944. 42

[62] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 295–304. 44

[63] M. Tomasi, F. Barranco, M. Vanegas, J. Díaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, accepted and under publication in IEEE Trans. on CSVT. (2010). 44, 81

[64] M. Tomasi, F. Barranco, M. Vanegas, J. Díaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture. (2009). 44

[65] H. C. Longuet-Higgins, K. Prazdny, The interpretation of a moving retinal image, Royal Society of London Proceedings Series B 208 (1980) 385–397. 45

[66] Nasa sequence, Data repository. [cited 3 February 2009].
URL ftp://ftp.uniovi.es/pub/vision/TESTDATA/NASA_DATA/ 49

[67] Yosemite sequence, Data repository. [cited 3 February 2009].
URL ftp://ftp.uniovi.es/pub/vision/TESTDATA/YOSEMITE_DATA/ 49

[68] Z. Sun, G. Bebis, R. Miller, On-road vehicle detection: A review, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006) 694–711. doi:http://doi.ieeecomputersociety.org/10.1109/TPAMI.2006.104. 58

[69] X. Baró, S. Escalera, J. Vitrià, O. Pujol, P. Radeva, Traffic sign recognition using evolutionary adaboost detection and forest-ecoc classification, Trans. Intell. Transport. Sys. 10 (1) (2009) 113–126. doi:http://dx.doi.org/10.1109/TITS.2008.2011702. 58

[70] M. Correia, A. Campilho, Real-time implementation of an optical flow algorithm, in: ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02), Vol. 4, 2002, pp. 247 – 250 vol.4. doi:10.1109/ICPR.2002.1047443. 58

[71] A. Bruhn, J. Weickert, T. Kohlberger, C. Schnoerr, A multigrid platform for real-time motion computation with discontinuity-preserving variational methods, International Journal of Computer Vision 70 (3) (2006) 257–277. 58

[72] H. Niitsuma, T. Maruyama, High speed computation of the optical flow, in: ICIAP, 2005, pp. 287–295. 58

[73] J. C. Sosa, R. Gomez-Fabela, J. A. Boluda, F. Pardo, Change-driven image architecture on FPGA with adaptive threshold for optical-flow computation, in: Reconfigurable Computing and FPGA's. ReConFig 2006. IEEE International Conference on, 2006, pp. 1 –8. doi:10.1109/RECONF.2006.307775. 58

[74] Z. Wei, D.-J. Lee, B. E. Nelson, J. K. Archibald, B. B. Edwards, FPGA-based embedded motion estimation sensor, International Journal of Reconfigurable Computing 2008 (2008) 1 – 8. 58

[75] G. Botella, A. Garcia, M. Rodriguez-Alvarez, E. Ros, U. Meyer-Baese, M. C. Molina, Robust bioinspired architecture for optical-flow computation, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on PP (99) (2009) 1 –1. doi:10.1109/TVLSI.2009.2013957. 58

[76] P. A. Beardsley, A. Zisserman, D. W. Murray, Sequential updating of projective and affine structure from motion, Int. J. Comput. Vision 23 (3) (1997) 235–259. doi:http://dx.doi.org/10.1023/A:1007923216416. 58

[77] P. H. S. Torr, A. W. Fitzgibbon, A. Zisserman, The problem of degeneracy in structure and motion recovery from uncalibrated image sequences, Int. J. Comput. Vision 32 (1) (1999) 27–44. doi:http://dx.doi.org/10.1023/A:1008140928553. 58

[78] D. Nistér, Preemptive RANSAC for live structure and motion estimation, in: ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision, IEEE Computer Society, Washington, DC, USA, 2003, p. 199. 58

[79] A. K. Mitros, A compact system for self-motion estimation, Ph.D. thesis, California Institute of Technology (2006).
URL http://etd.caltech.edu/etd/available/etd-05252006-145119/ 58

[80] W. S. Fife, J. K. Archibald, Reconfigurable on-board vision processing for small autonomous vehicles, EURASIP J. Embedded Syst. 2007 (1) (2007) 33–46. doi:http://dx.doi.org/10.1155/2007/80141. 58

[81] C. Claus, W. Stechele, Dynamically Reconfigurable Systems, Springer Science, 2010. doi:10.1007/978-90-481-3485-4_18.
URL http://www.springerlink.com/content/n174672p01784217 58

[82] Xilinx Inc., Multi-Port Memory Controller (MPMC) (2009).
URL http://www.xilinx.com/support/documentation/ipembedprocess_memoryinterface_mpmc.htm 59

[83] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multi-port abstraction layer for FPGA intensive memory exploitation applications, Journal of Systems Architecture In Press, Accepted Manuscript (2010) –, URL http://www.sciencedirect.com/science/article/B6V1F-50412G1-2/2/47c37daf412ec52d7e185359ebf85965. doi:DOI:10.1016/j.sysarc.2010.05.007. 62

[84] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing. (2009). 82

[85] M. Vanegas, L. Rubio, M. Tomasi, J. Díaz, E. Ros, On-chip ego-motion estimation based on optical flow, submitted to International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. SAMO X. (2010).

[86] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, ISIE2010. Accepted for IEEE Internetional Symposium on Industrial Electronics, Bari (Italy). (July 2010).

[87] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Díaz, Entorno software para visualización y configuración de procesamientode imágenes en tiempo real con plataformas reconfigurables, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 327–336.

[88] J. Gómez López, M. Vanegas Hernández, C. Morillas Gutiérrez, M. López Górdo, F. Pelayo Valle, Implementación hardware-software de operadores morfológicos en powerpc, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 517–526.

[89] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: Proceedings of Imaging Understanding Workshop, 1981, pp. 121 – 130.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.201984

[90] C. Morimoto, R. Chellappa, Fast electronic digital image stabilization for off-road navigation, Real-Time Imaging 2 (5) (1996) 285 – 296. doi:DOI:10.1006/rtim.1996.0030.
URL http://www.sciencedirect.com/science/article/B6WPR-45NJS7K-3/2/849f98ceac3b646e3bbed864569a0470 86

[91] S. Ertürk, Real-time digital image stabilization using kalman filters, Real-Time Imaging 8 (4) (2002) 317 – 328. doi:DOI:10.1006/rtim.2001.0278.
URL http://www.sciencedirect.com/science/article/B6WPR-46WF1H5-6/2/943138636e107098c32c36a30e3f5f47 86