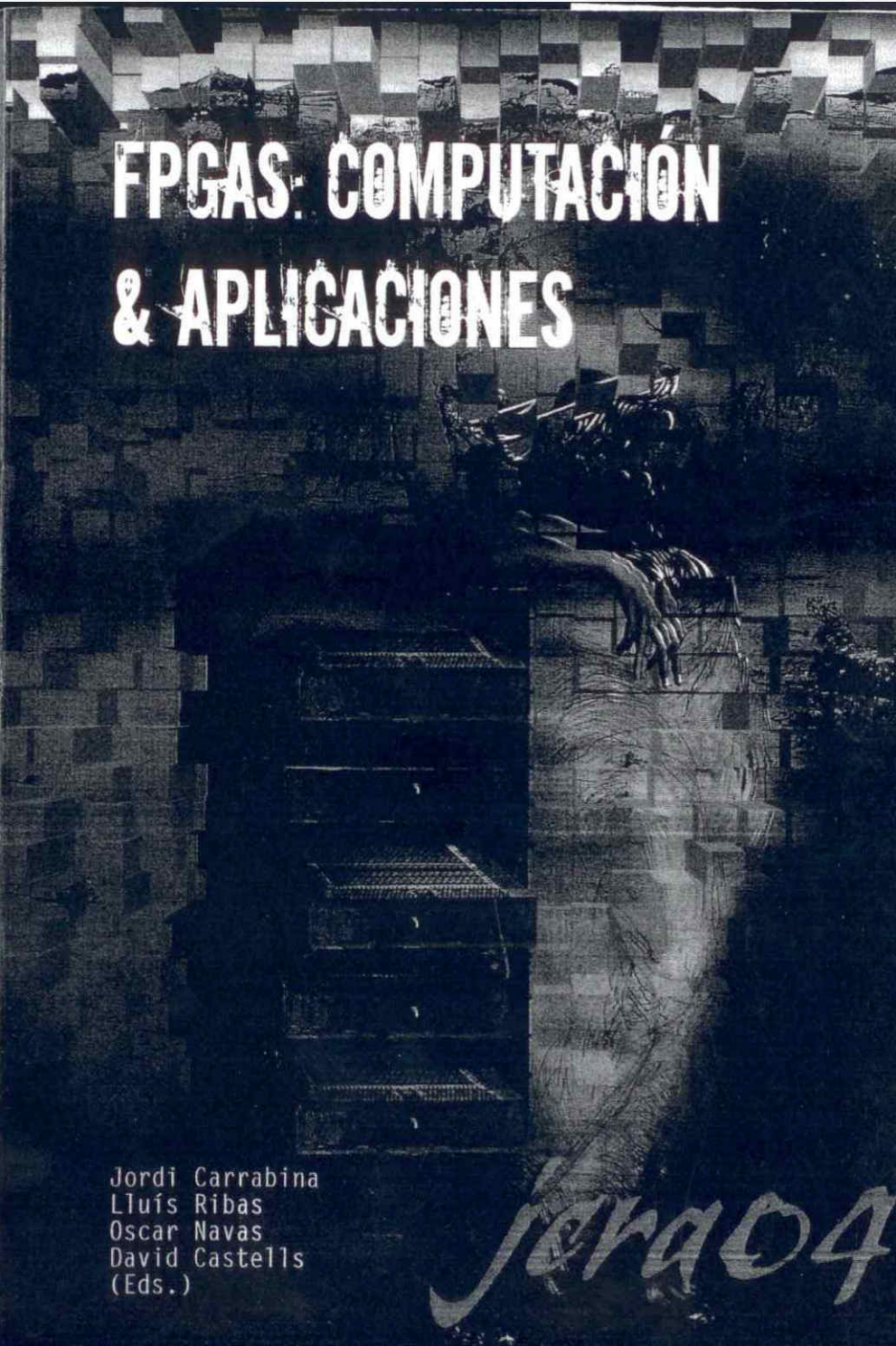
FPGAS: COMPUTACIÓN & APLICACIONES



Jordi Carrabina
Lluís Ribas
Oscar Navas
David Castells
(Eds.)

Jera04

FPGAs: COMPUTACIÓN & APLICACIONES

Artículos seleccionados de las IV Jornadas de Computación Reconfigurable y Aplicaciones Bellaterra, 13-15 de Septiembre de 2004

Jordi Carrabina
Lluís Ribas
Oscar Navas
David Castells

Editores

Centre de Prototips i Solucions HW/SW
Escola Tècnica Superior d'Enginyeries
Universitat Autònoma de Barcelona

Título: FPGAs: COMPUTACIÓN & APLICACIONES

I.S.B.N.: 84-688-7667-4
Depósito Legal: 40.194-2004

Editores: J. Carrabina, Ll. Ribas, O. Navas, D. Castells
Centre de Prototips i Solucions HW/SW
Escola Tècnica Superior d'Enginyeries
Universitat Autònoma de Barcelona
08193 Bellaterra
cephis@cephis.uab.es
http://cephis.uab.es

Impresión: Format Digital S.L.
C/ Berlin 11-13, Baixos
08014 Barcelona
Tel: 934090005
formatdigital@telefonica.net

6.3 Algoritmos de DSP

7.1 Visión (I)

Cómputo de flujo óptico en tiempo real mediante FPGAs

Javier Díaz[1], Eduardo Ros[1], Sonia Mota[1], Eva M. Ortigosa, Richard Carrillo[1], Rodrigo Agís[1]

[1] Departamento de Arquitectura y Tecnología de Computadores, E.T.S.I. Informática, Universidad de Granada, Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain
{jdiaz, eros, smota, eva, rcarrillo, ragis}@atc.ugr.es

Resumen. Utilizando dispositivos FPGA, describimos un sistema que actúa como un *sensor virtual de movimiento*. La reconfigurabilidad y modularidad del diseño nos permite adaptar fácilmente el dispositivo a distintas condiciones de luminosidad, velocidad del movimiento u otras condiciones del entorno. El sistema es denominado *sensor virtual de movimiento* por que consiste en una cámara convencional como sensor de entrada y un elemento de procesamiento (FPGA) que implementa el frame-grabber, el circuito de estimación de flujo óptico así como la lógica de control, almacenamiento y visualización de la salida. Abordamos también el estudio de la precisión del sistema, así como su rendimiento y escalabilidad proporcionando un análisis completo del sistema y su comparación con la implementación software.

1 Introducción

El estudio del movimiento presente en secuencias de imágenes es un problema extensamente estudiado en la visión por computador y tiene numerosas aplicaciones para la segmentación de objetos en movimiento, cálculo de tiempos de impacto, reconocimiento de objetos, análisis 3-D de la imagen, compresión de video, etc. Aun siendo una tarea compleja, existen estudios relativos a la exactitud del flujo óptico obtenido por las distintas aproximaciones utilizando secuencias de video sintéticas en las que el flujo óptico es conocido [1].

Nuestro trabajo trata sobre la implementación en tiempo real del método de estimación de Lucas y Kanade (L&K) [1,2]. Algunos autores han resaltado su buena relación entre exactitud y eficiencia, lo que es un factor clave para su elección como método para implementar en hardware. Por ejemplo, en la evaluación realizada en [1] L&K muestra resultados aceptables, Liu et al. en [3] estudian la relación entre eficiencia y exactitud de diferentes algoritmos y indican que el de L&K es una buena opción para su implementación en tiempo real. Finalmente, McCane et al. [4] también indica sus buenas cualidades así como que requiere una potencia computacional abordable. Todo ello ha motivado que el presente método haya sido muy estudiado por distintos autores [5, 6].

En este articulo describimos nuestra implementación en tiempo real del algoritmo de L&K. Aunque hay otros autores que han realizado implementaciones en tiempo real de otros algoritmos de flujo óptico [7, 8, 9], la mayoría no proporcionan resultados acerca del rendimiento del sistema es decir, de su exactitud y eficiencia. El sistema que describimos es capaz de procesar en tiempo real imágenes de 320x240 pixeles con 30 cuadros por segundo y como veremos, la degradación de la implementación hardware respecto al software es pequeña.

2. Descripción del modelo de estimación flujo óptico

Originariamente el algoritmo de L&K fue propuesto como un método para estimación de la disparidad de pares de imágenes estereo [2]. Nosotros partimos de la descripción de Barron et al.[1] del algoritmo que es utilizada para la estimación de movimiento. Como principal modificación cabe destacar el uso de filtros IIR en vez de FIR, tal y como es descrito en [10]. Los filtros recursivos son más apropiados para su implementación en sistemas embebidos debido a que en ellos los requerimientos de memoria externa son menores y además, independendientes de la extensión temporal del filtro, cosa que no ocurre con filtros FIR.

En las próximas ecuaciones (1), (2) tratamos de describir brevemente las diferentes etapas del algoritmo de L&K que son las que darán lugar a la arquitectura hardware. Usamos una ventana ponderada con $W(x)$ que es una función monótona decreciente si nos alejamos del píxel central que se extiende en una ventana espacial Ω para el ajuste por mínimos cuadrados.

La solución final de este sistema es:

$$\vec{v} = \left[A^T W^2 A\right]^{-1} A^T W^2 \vec{b} \tag{1}$$

$$\begin{aligned} A &= \left[\nabla I(x_1),....,\nabla I(x_n)\right] \\ W &= diag\left[W(x_1),....,W(x_2)\right] \\ \vec{b} &= -\left[I_t(x_1),....,I_t(x_n)\right] \end{aligned} \tag{2}$$

En la ecuación (3), hemos añadido una pequeña constante a la diagonal de la matriz izquierda de (1) (ver [11]) que nos permite obtener estimaciones de velocidad para situaciones donde aparezca el problema de la apertura.

$$A^T W^2 A = \begin{bmatrix} \sum_{x\in\Omega} W^2 I_x^2 + \alpha & \sum_{x\in\Omega} W^2 I_x I_y \\ \sum_{x\in\Omega} W^2 I_x I_y & \sum_{x\in\Omega} W^2 I_y^2 - \alpha \end{bmatrix} \tag{3}$$

Resumiendo, nosotros debemos computar la matriz 2x2 de la ecuación (3), su inversa y la matriz 2x1 de la ecuación (4).

$$B = A^T W^2 \vec{b} = \begin{bmatrix} -\sum_{x\in\Omega} W^2 I_x I_t \\ -\sum_{x\in\Omega} W^2 I_y I_t \end{bmatrix} \tag{4}$$

El uso filtros FIR temporales requiere del orden de 15 imágenes previas almacenadas, lo que es un requerimiento bastante costoso para sistemas embebidos. Es por ello que hemos implementado filtros temporales IIR tales como los descritos en [10] que necesitan solo el almacenamiento de 3 imágenes anteriores y que además permiten, sin modificar los requerimientos de memoria, modificar la extensión temporal con sólo el cambio de la constante de tiempo del filtro.

3. Implementación hardware

Actualmente debido al rápido aumento de las prestaciones de los procesadores de los PCs actuales el cómputo software de flujo óptico para imágenes pequeñas es posible en tiempo real. El problema aparece si pretendemos utilizar esta aproximación para sistemas embebidos y debido a esto nosotros usamos FPGAs. Otras tecnologías posibles basadas por ejemplo en ASICs pueden obtener rendimientos iguales o superiores pero con un mayor coste de desarrollo así como pérdida de flexibilidad. DSPs son una solución óptima si poseen suficiente potencia de cálculo pero cuando no, arquitecturas basadas en sistemas multiprocesador son empleadas y en esos casos comienzan a perder su principal ventaja, la flexibilidad del sistema y el rápido tiempo de desarrollo [12]. Como vemos a continuación, el sistema desarrollado usa dispositivos reconfigurables para implementar un sistema capaz de actuar como un coprocesador hardware o como un sensor embebido para sistemas empotrados.

Nuestra plataforma de desarrollo ha sido la placa RC1000-PP de Celoxica [13] y como lenguaje de descripción hardware hemos usado uno de alto nivel tal y como es el Handel C [14]. Esta placa es conectada al PC mediante el bus PCI y permite el desarrollo de circuitos de coprocesamiento para el computador o bien el prototipado de sistemas. Contiene una FPGA Virtex-E XCVE2000-4 y cuatro bancos de 2 MB de memoria SRAM accesibles en paralelo.

3.1 Descripción general del sistema

Para una implementación eficiente del sistema dentro de la FPGA debemos hacer un uso eficiente de sus posibilidades de paralelismo y segmentación de cauce. Nosotros hemos diseñado una arquitectura segmentada que es descrita en la Fig. 1. Las etapas de procesamiento ilustradas en ella son:

S_0. El Frame-Grabber recibe las imagines de la cámara y las almacena en la memoria externa utilizando un doble buffer para evitar problemas de sincronización entre el PC y la FPGA en el acceso a los bancos externos.

S_1. Etapa de suavizado espacial de la imagen con filtros Gaussianos.

S_2. Procesamiento temporal de la imagen con filtros IIR para suavizado y computo de la derivada.

S_3. Estimación espacial de las derivadas vertical y horizontal de la imagen.

S_4. Computo de los elementos de las matrices (2) y (3).

S_5. Desarrollo de la unidad de cómputo flotante para la estimación final de la velocidad. Debido a que esta etapa requiere costosas operaciones con datos con número de bits muy alto, una representación con datos en punto fijo se hace

inviable. Es por ello que hemos desarrollado una unidad específica de cómputo con datos flotantes de precisión seleccionable.

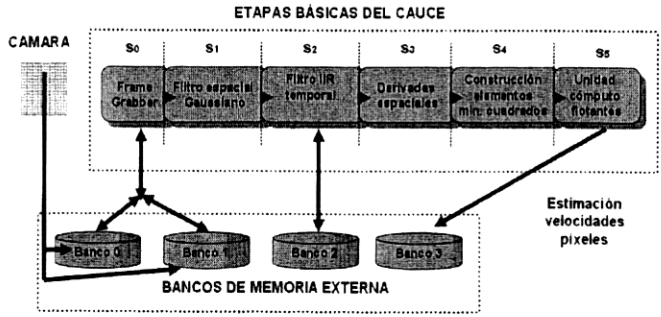


Fig. 1. Etapas básicas del cauce propuesto para el sistema de procesamiento.

Respecto a los parámetro útiles para caracterizar el sistema tenemos la latencia, (L) y en máximo numero de ciclos (MNC) consumidos en el procesamiento por la etapa más lenta, siendo este parámetro crítico a la hora de considerar el rendimiento del sistema global. Esta arquitectura segmentada nos proporciona una relación básica entre el número de píxeles procesados por Segundo (pps) y la frecuencia de trabajo del sistema (f_{clk}) que puede ser calculada usando la ecuación $\mathbf{pps=f_{clk}/MNC}$.

Debido a que las etapas más complejas y costosas de procesamiento son la 5 y la 6, en los dos siguientes apartados nos centramos en estudiar su arquitectura y posibilidades de paralelismo.

3.2. Construcción de las matrices para ajuste por mínimos cuadrados

Esta es una etapa crítica respecto al consumo de la FPGA en la que podemos ilustrar el compromiso entre compartición de recursos, paralelismo y coste hardware. Para la construcción de las matrices de las ecuaciones (2) y (3) necesitamos computar los productos: I_x^2, I_y^2, I_xI_y, I_xI_t, I_yI_t. Hecho esto, promediamos sus valores en una vecindad Ω de tamaño w_x por w_y. Debido a los limitados recursos de memoria embebida, almacenamos en ella sólo los valores de las derivadas I_x, I_y y I_t en vez de sus productos cruzados. Un esquema de esta arquitectura es mostrado en la Fig. 2.

Relativo a la relación existente entre coste de la arquitectura y eficiencia de la misma, los parámetros principales a seleccionar son el área de promediado, el número de unidades de convolución por filas y el número de unidades de convolución por columnas. Por ejemplo, para el caso descrito en la Fig. 1 que usa una vecindad 3x3, el número de multiplicadores hardware necesarios para el cómputo de los productos cruzados varía entre 1 y 45. Podemos utilizar entre 1 y 15 unidades de convolución por filas y hasta 5 por columnas. Diferentes elecciones son mostradas en la sección 4

en la que vemos cómo, dependiendo del paralelismo requerido se consumen más o menos recursos hardware.

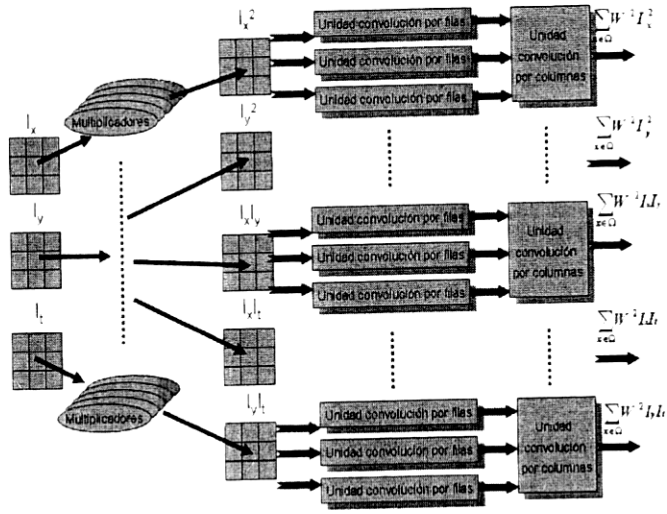


Fig. 2. Arquitectura del sistema de construcción de las matrices para ajuste de mínimos cuadrados en una vecindad de 3x3 elementos.

3.3. Cómputo final de la velocidad usando una unidad flotante personalizada

Esta etapa realiza el cómputo descrito en la ecuación (1). En la etapa anterior los datos están representados en punto fijo de hasta 24 bits, lo cual nos asegura una alta precisión pero a costa de un alto coste hardware. Esto ha motivado el desarrollo de esta unidad de cómputo con coma flotante. Tal y como se desprende de la ecuación (1), las operaciones a realizar en esta etapa son la inversión de una matriz (2x2) más la multiplicación de ésta por un vector de (2x1) usando para ello como operaciones aritméticas la resta, multiplicación y la división.

En nuestra unidad de cómputo con flotantes hemos definido circuitos que procesan la información en tan solo un ciclo de reloj con excepción de la operación de división en la que se ha definido una estructura iterativa debido a su bajo coste hardware. El número de ciclos requeridos en este caso para realizar la división es entonces de 21 para datos en representación flotante de 25 bits y ello nos limita de manera excesiva el rendimiento del sistema. Es por ello que para reducir estos problemas hemos diseñado una unidad superescalar de 3 vías para el cómputo de la división. Dependiendo de los

requerimientos del sistema podemos utilizar más o menos vías y sintetizar más o menos unidades convolucionales.

4. Configuraciones y consumo de recursos

El sistema ha sido diseñado de manera modular para poder modificar de manera sencilla el grado de paralelismo así como la longitud de bits de las diferentes etapas. El uso de un lenguaje de alto nivel como Handel-C [14] proporciona una herramienta muy sencilla para el manejo de estos parámetros.

Un punto importante de nuestro diseño es la posibilidad de configuración del mismo con distintos grados de paralelismo y rendimiento dependiendo de nuestras necesidades. Nosotros hemos explorado diversas configuraciones basadas en una misma arquitectura pero modificando estos parámetros. La tabla 1 resume los resultados principales encontrados para las distintas versiones. La versión que usa una vecindad de promediado de 5x5 para el ajuste por mínimos cuadrados la hemos denominado version de alta calidad *(high quality HQ)* y la que usa una vecindad de 3x3 la denominamos calidad media *(medium quality, MQ)*. Otros parámetros seleccionables son el tamaño de los filtros de suavizado y cómputo de derivadas. Las versiones anteriores utilizan núcleos derivativos de 5 etapas y filtros Gaussianos de tamaño 9. Hemos sintetizado también una versión de bajo coste *(low cost, LQ)* que usa derivadas y suavizado de tamaño 3.

Tabla 1. Rendimiento y coste hardware de las distintas configuraciones estudiadas para una Virtex 2000-E-6 FPGA con un total de 43200 LCs (aproximadamente, un LC de Xilinx equivale a 1 LE de Altera, su principal competidor). (Kpps → kilopixels por segundo, Fps → frames por segundo). Como medida de referencia usamos una frecuencia de reloj para todos los diseños de f_{clk}=27MHz. Notemos que las medidas Kpps y Fps están sub-estimadas ya que en el tiempo de procesamiento también se han tenido en cuenta el de transmisión vía PCI de la placa de prototipado. Veáse [16] para un estudio de la degradación del rendimiento debido al PCI.

Versión	% dispositivo ocupado	% Memoria embebida	Kpps	Resolución imagen	Fps (f_{clk}=27MHz)	Max. f_{clk} (MHz)
HSHQ	99	17 31	1776	160x120 320x240	95 24	35
HSMQ	65	16 31	1776	160x120 320x240	97 24	35
MSMQ	43	16	625	160x120	33	35
LSLQ	36	8	400	120x90	38	35

Si fijamos la calidad del flujo óptico estimado, otro parámetro que podemos modificar es el grado de paralelismo del sistema. La versión más rápida tiene un valor de MNC de 10. Es posible aumentar este valor mediante la compartición de recursos. Por ejemplo, podemos sintetizar una versión rápida *(high-speed, HS)* con MNC=10 ciclos usando una unidad superescalar de 3 vías para la división y el máximo paralelismo para las distintas etapas. Una versión más lenta es posible disminuyendo el número de vías así como el número de unidades de convolución para el promediado

en la etapa de construcción de matrices. A esta versión la denominamos velocidad media *(medium speed, MS)*. Finalmente, la versión de baja velocidad *(low-speed, LS)* realiza compartición de gran número de elementos de procesamiento.

En lo referente a la máxima frecuencia de reloj prevista por ISE, podemos afirmar que las estimaciones provenientes del *Xilinx timing analyser* no son exactas. La frecuencia máxima de trabajo permitida por el sistema ha sido experimentalmente medida en la placa y es del orden de 10-20 MHz más rápida de lo previsto por ISE. Ello ocurre debido a que el analizador de caminos usa los caminos lógicos estáticos en vez de los dinámicos (ver [15]) y es por esto que las medidas de frecuencia de reloj utilizadas por nosotros son las experimentales.

5. Evaluación de la exactitud del sistema

Tal y como comentamos al principio, la medida de la exactitud del flujo óptico proporcionado por una implementación es difícil de medir para secuencias reales por lo que suele recurrirse a secuencias de imágenes sintéticas. Siguiendo el trabajo realizado por Barron et al. [1] y para medir la degradación del sistema debido a la reducida longitud de los datos utilizados, hemos medido la exactitud del flujo estimado para la secuencia de Yosemite con la implementación HSHQ y los resultados son un error angular medio de 15.91° para la versión software y de 18.30° para el hardware. Esta degradación es justificable debida a la pérdida de precisión de los datos usados que, a diferencia del software, no son flotantes de 64 bits. Además hemos utilizado para medir la degradación del sistema secuencias de movimiento de rejillas sinusoidales con diferentes frecuencias espaciales (f_0=0.02 y f_0=0.05) y velocidades (V=0.25 ppf y V=1 ppf). Con ellas la implementación hardware obtiene resultados muy similares a los provistos por el software, con una desviación inferior al 5% en la estimación de la velocidad.

Finalmente, hemos comparado el rendimiento software con el hardware usando para ello un versión programada en C estándar del algoritmo. Con ella podemos procesar hasta 30 fps de imágenes de tamaño 160x120 píxeles en un AMD 1800+ frente a la versión hardware que va unas 4 veces más rápido.

6. Conclusiones

Hemos descrito un sistema para cómputo de flujo óptico en tiempo real basado en hardware reconfigurable programado como un DSP de uso específico. La arquitectura descrita es modular, escalable y permite el procesamiento en tiempo real de imagines a velocidad de video de hasta 320x240 píxeles.

Respecto a nuestros objetivos futuros, planeamos dos modificaciones principales del sistema. La primera es un estudio más riguroso de la profundidad de bits de los datos usados dependiendo de la aplicación a la que vaya dirigido el sistema. Nuestro segundo objetivo consiste en el cómputo de flujo óptico usando la aproximación multi-escala, que nos permite solventar en gran medida el problema de la existencia

4. Resultados experimentales y conclusiones

En esta experiencia mostramos un sistema robótico, basado en hardware reconfigurable (Véase Tabla 1), que sirve de base de estudio para sistemas de control adaptativo futuros. El objetivo a medio plazo pasa por dos líneas de investigación: (a) Ajuste dinámico del PID optimizando el consumo y error de posición utilizando algoritmos genéticos. (b) Sustitución del PID por una red neuronal de pulsos.

Módulos	% Consumo *Slices*	Frecuencia máx. Mhz
Comunicaciones	5.64	139.25
Lector de posición	0.97	127.68
Controlador PID	0.99	132.22
Ajuste de cero	0.97	127.68
Sistema de visión	34.3	25.62
Generador PWM	3.78	88.37
Lógica de apoyo (*)	1.85	121.62
TOTAL 1 (**)	37.9	33.82
TOTAL 2 (**)	64.98	22.54

Tabla 1. Porcentaje de ocupación y frecuencia máxima realizada con compilaciones parciales (aislamiento parcial de módulos). Software de desarrollo y síntesis DK-2 [9] e ISE-5.1i [10]. Nº *slices* de la FPGA : 5120. (*) Parpadeo de *leds, displays* etc. (**): Sin y con sistema de visión.

Referencias

1. SpikeFORCE, (EU Project: IST-2001-35271). http://www.spikeforce.org
2. EcoVision, (EU Project: IST-2001-32114). http://www.pspc.dibe.unige.it/ecovision
3. Ziegler-Nichols (1942): *Tuning rules and limitations.* http://www.automationtechies.com/sitepages/pid1373.php
4. Regulador PID: http://iaci.unq.edu.ar/materias/Identificacion_y_Control_Adaptativo/Apuntes/15-PID.pdf
5. Gerstner, W., Kistler, W.; Spiking Neuron Models. Cambridge University Press, (2002)
6. Jahnke, A.; Schoenauer, T.; Roth, U.; Mohraz, K.; Klar, H.: *Simulation of Spiking Neural Networks on Different Hardware Platforms.* ICANN97, LCNS, pp. 1187-1192 (1997)
7. National Semiconductors: http://www.national.com
8. Control de velocidad mediante PWM (Frenado regenerativo): http://www.isa.cie.uva.es/proyectos/servos/info/PWM/PWM.htm
9. Celoxica, http://www.celoxica.com
10. Xilinx, http://www.xilinx.com
11. Handel-C language referent manual. Celoxica 2003.
12. Roberto Sanchos: Implementación digital de controladores PID: http://www.tec.uji.es/asignatura/dir_asignaturas/3/67/Tema1.pdf
13. James Brusey and Lin Padgham. *Techniques for obtaining robust, real-time, colour-based vision for robotics.* 1999. In *IJCAI-99 Proceedings of the Third International Workshop on Robocup.* Stockholm, Sweden.