VNIVERSITAT DE VALÈNCIA

UNIVERSITAT DE VALÈNCIA

Arquitectura de procesamiento de imágenes basada en lógica reconfigurable para navegación de vehículos autónomos con visión foveal

Memoria para optar al grado de doctor presentada
al Dpto. de Informática
Facultad de Física

Jose Antonio Boluda Grau

Dirección

Dr. Fernando Pardo Carpio
Dr. Joan Pelechano Fabregat

A qui tant hem estimat i ja no està amb nosaltres.
A la meua iaia Carmen.

Índice General

Índice de Tablas

Índice de Figuras

Agradecimientos

Son muchas las personas que de una u otra forma han contribuido a que haya podido llevar a cabo el trabajo de investigación que se expone en la presente memoria, y a todas ellas quiero agradecer sinceramente su ayuda.

En primer lugar quisiera agradecer a mis directores, el Dr. Fernando Pardo y el Dr. Joan Pelechano, su inestimable ayuda técnica y su amistad tantas veces puesta a prueba.

Quisiera de la misma manera agradecer al Dr. Gregorio Martín el haber confiado en mi como becario de investigación en el proyecto FASST de tolerancia a fallos. A mis compañeros de proyecto: Germán Fabregat, Rafa Martínez, Fernando Pardo y Carlos Pérez debo agradecer su ayuda y su amistad, tanto en mis inicios como investigador, como en la actualidad. De la misma manera al resto de compañeros del antiguo LISITT debo agradecer su cordialidad y ayuda durante tantos años de trabajo. Dentro del Departament d'Informàtica he contado siempre con el aliento de, tanto mis compañeros de área, como del resto de colegas de otras áreas. A ellos quisiera agradecer su interés y sus ánimos tan sinceros.

No quiero dejar de agradecer al Dr. Nandhakumar su gran ayuda en el desarrollo de los primeros trabajos del presente trabajo de investigación en la Universidad de Virginia. Él inicialmente y mucha gente adicional tanto en Virginia como en España, tuvieron en esos complicados meses de mi vida un comportamiento exquisito que nunca podré agradecer suficientemente.

Quisiera también agradecer al Dr. Juan Domingo sus consejos y su inagotable capacidad de trabajo. Sinceramente, espero poder agradecerle alguna vez todo el tiempo que ha invertido en mí. Análogamente quisiera agradecer al resto del grupo de control y robótica móvil del Instituto de Robótica su ayuda en la fase de experimentación, especialmente al Dr. Francisco Vegara por su colaboración desinteresada. Análogamente quisiera agradecer a Francisco Blasco su inestimable ayuda en la fase de depuración de los elementos de proceso.

No sería justo olvidar las fuentes de financiación del presente trabajo de investigación. En primer lugar a la CICYT por su financiación con el proyecto CICYT TAP95-1086-CO2-02 dirigido por el Dr. Joan Pelechano, co-director del presente trabajo de

investigación. Posteriormente la financiación fue continuada por la Generalitat Valenciana con el proyecto GV97-TI-05-27 dirigido por el Dr. Filiberto Pla, al que quisiera agradecer su tiempo y acertados comentarios. Finalmente la financiación ha continuado con el proyecto CICYT TIC98-1026 dirigido por el Dr. Juan Domingo.

Finalmente quisiera agradecer a la gente que me ha dado algo más importante que las ayudas y consejos científicos: el afecto y el cariño sin el que no habría podido llevar a cabo este trabajo. A mis padres quisiera agradecer el haber creído tanto en mi, haberme dedicado su vida y haber renunciado a tantas cosas para que yo y mis hermanos Susi y Vicente pudiéramos tener tantas otras. A Alicia le quiero agradecer su infinita paciencia y cariño. Su amor ha sido el estímulo más dulce para la finalización del presente trabajo de investigación.

Resumen

La sensorización visual ha mostrado su utilidad en la navegación de plataformas móviles autónomas al conseguir simultáneamente largo alcance y precisión. Sin embargo, como desventaja de los métodos de sensorización visuales habituales, se tiene la gran cantidad de información a procesar. A este hecho hay que añadir el alto coste computacional de muchos algoritmos de visión artificial que serían útiles como ayuda a la navegación, más las características de tiempo real típicamente asociadas al control del robot. Todas estas restricciones plantean soluciones a medida para el procesamiento de la información visual. Por otra parte, es necesario poder seleccionar el algoritmo de procesado de la información visual dependiendo de la tarea específica a realizar. La naturaleza autónoma de este tipo de vehículos dificulta la posibilidad de que se dispongan de todos los recursos *hardware* necesarios por razones de tamaño y consumo.

Los dispositivos lógicos programables combinan simultáneamente la velocidad del *hardware* con una cierta programabilidad, con lo que se ha planteado la utilización de un sistema basado en lógica reconfigurable en la etapa de procesamiento de la información visual.

Simultáneamente, se ha planteado la utilización del sensor log-polar CMOS recientemente desarrollado como sensor visual, ya que el formalismo log-polar reduce selectivamente la cantidad de información a procesar, proporcionando gran resolución en el centro de la imagen y menos en la periferia. De manera adicional, el sistema coordenado log-polar simplifica la utilización de diversos algoritmos cuando el centro del sensor coincide con el centro de expansión óptica.

Los algoritmos diferenciales de visión artificial son un tipo de algoritmos que se benefician significativamente de la reducción de información. Las características de este tipo de algoritmos, particularizadas al formalismo log-polar, han definido las bases de la arquitectura del módulo reconfigurable de procesamiento de imágenes log-polares. La arquitectura consiste en un cauce segmentado donde cada etapa es implementada por un elemento de proceso. Todos los elementos de proceso son idénticos, teniendo una única interfase y un mismo protocolo de intercambio de datos. Su funcionalidad está definida por el dispositivo lógico programable que es el núcleo del elemento de proceso. La reconfiguración de cada etapa permite la programación del algoritmo de visión seleccionado por la plataforma autónoma. La especial orientación de la arquitectura a

los algoritmos diferenciales de visión artificial permite el cálculo eficiente de diferencias temporales en secuencias de imágenes.

La arquitectura desarrollada es independiente de la implementación realizada y es escalable y ampliable en el futuro. La disponibilidad de dispositivos reconfigurables más rápidos o la combinación de más de uno de estos dispositivos en un sólo elemento de proceso mejorará las prestaciones del módulo reconfigurable. En función de las restricciones de la plataforma móvil y de la necesidad de cálculo de cada algoritmo se puede reducir o ampliar el número de elementos de proceso incluidos en el cauce reconfigurable.

Partiendo de la arquitectura del módulo reconfigurable se ha definido una metodología genérica de diseño de algoritmos diferenciales. Utilizando esta metodología se han implementado dos algoritmos diferenciales de visión artificial que se benefician especialmente del uso del sistema coordenado log-polar. El primero de ellos, explícitamente desarrollado para su utilización en el módulo reconfigurable, es capaz de descartar automáticamente el desplazamiento de la imagen debido al movimiento propio del vehículo autónomo. El segundo algoritmo consiste en la aplicación del algoritmo de cálculo del tiempo al impacto en coordenadas log-polares desarrollado por Tistarelli y Sandini. El pequeño tamaño de las imágenes log-polares, combinado con la segmentación de los algoritmos y el aprovechamiento de la capacidad del cauce reconfigurable para calcular diferencias temporales, hace que se consiga procesar una gran cantidad de imágenes por segundo.

La alta velocidad de procesamiento (en términos de imágenes por segundo) y la naturaleza diferencial de los algoritmos ha obligado a parametrizar, en función de las características del sensor y de la escena, el intervalo de adquisición mínimo que garantice diferencias entre imágenes. La posterior experimentación con ambos algoritmos a mostrado su utilidad y ha permitido averiguar bajo qué condiciones se obtienen unos buenos resultados con las implementaciones realizadas. De la misma manera, en el caso en el que los resultados no han sido los esperados se ha encontrado el origen de este error.

Los resultados del presente trabajo de investigación plantean aplicaciones futuras en diversos campos. La utilización del módulo reconfigurable puede extenderse a otros entornos en los cuales sea necesario un procesamiento de imágenes con alta velocidad. La reducción selectiva de información conseguida con el formalismo log-polar ha mostrado ser especialmente útil en los problemas abordados. De la misma manera, la utilización de arquitecturas reconfigurables dentro de la robótica apunta resultados prometedores.

Abstract

The use of visual sensors has shown to be useful in the navigation of autonomous platforms, since they are simultaneously high distance effective and accurate enough. Nevertheless, the big quantity of data to be processed appears as a disadvantage of the most common visual sensorization methods. In the same way, it should be taken into account the high computational cost of many artificial vision algorithms useful for robotic navigation, and the real-time restrictions usually associated to the robot control. In the other hand, the autonomous platform should be able to choose the better image processing algorithm in order to perform a specific task. The autonomous nature of this kind of vehicles makes it difficult for the platform to transport all the hardware resources possible due to size and power consumption reasons.

The programmable logic devices combine simultaneously hardware speed and a certain degree of programmability. In this way, a system based in reconfigurable logic has been proposed for the image processing system.

Simultaneously, the use of the log-polar CMOS sensor recently developed has been elected, since the log-polar formalism selectively reduces the amount of data to be processed, yielding more resolution at the center of the sensor and less at the periphery. Moreover, the log-polar formalism simplifies the utilization of several algorithms when the sensor center is at the optical expansion center.

Differential algorithms of image processing are a class of algorithms which can quickly obtain benefit of the information reduction. The characteristics of these kind of algorithms, customized to the log-polar formalism, have been useful for determining the basis of the reconfigurable module architecture. The architecture is a pipeline which stages have been implemented using a single processing element. All the processing elements are identical, having a common interface and data-exchange protocol. The processing element functionality is determined by the programmable logic device. The reconfiguration of each stage allows the programmability of the image processing algorithm selected by the autonomous platform. The special architecture orientation to image processing differential algorithms allows the efficient computation of temporal derivatives image sequences.

The developed architecture is independent of the carried out implementation and is scalable and extensible. The availability of faster programmable logic devices, or the

combination of more that one of this kind of devices in a single processing element, will improve the reconfigurable module performance. It is possible to reduce, or to extend, the number of processing elements included in the reconfigurable module taking into account the vehicle constrains and the computation needs of the algorithms.

Starting from the reconfigurable module architecture, a generic differential algorithms design methodology has been established. Using this methodology two image processing differential algorithms have been implemented, which specially obtain benefit from the use of the log-polar coordinate system. The first algorithm, explicitly developed to be implemented into the reconfigurable module, discards automatically the image displacement due to the camera ego-motion. The second one consists of the implementation of the method developed by Tistarelli and Sandini for the time-to-impact computation. The small size of the log-polar images, combined with the algorithm pipelining and the exploitation of the reconfigurable pipeline characteristics for computing temporal derivatives, makes possible a high throughput.

The high processing speed of the reconfigurable system (in terms of images per second) and the differential nature of the algorithms, force the parameterization, in function of the sensor and scene characteristics, of the minimum interval acquisition in order to guarantee differences between images. Several experiments with these two algorithms have shown the conditions to obtain good results. In the same way they have been found in which cases the results differ from the expected results, founding the source of this mismatch.

The conclusions of this research appoint future applications in several fields. The utilization of the reconfigurable module can be extended to other environments where a high speed image processing system is required. The selective reduction of information achieved with the log-polar formalism has shown to be a powerful approach for the proposed algorithms. In the same way, the use of reconfigurable architectures in the field of robotics is showing promising results.

Prólogo

El origen del presente trabajo de investigación parte de la confluencia de varias líneas en el seno del Departament d'Informàtica. A mediados de 1995 ya existía una cierta experiencia en el uso de dispositivos lógicos programables tras su utilización en diversos proyectos de investigación y desarrollo dentro del laboratorio **LISITT**. Este hecho coincidió temporalmente con la disponibilidad del sensor log-polar CMOS desarrollado por el Dr. Fernando Pardo, co-director del presente trabajo de investigación, en el laboratorio **IMEC** de Bélgica. El diseño e implementación del sensor se realizó en el marco del proyecto internacional IBIDEM que consistía en el desarrollo de un sistema de comunicación visual entre sordomudos. El desarrollo de aplicaciones reales que sacaran partido de la reducción de la información se inició tras la disponibilidad del sensor. Por otra parte, antes de la disponibilidad de la utilización de un sensor log-polar real se han realizado diversos estudios teóricos que muestran las ventajas en ciertos casos de este formalismo. Diversos grupos han mostrado la simplificación de varios algoritmos de visión artificial. Entre ellos cabe destacar el grupo **LIRA** (*Laboratorio Integrato di Robotica Avanzata*) del profesor Giulio Sandini, grupo en el que se inició la colaboración del Dr. Fernando Pardo en el proyecto IBIDEM.

Por otra parte, y dentro del *Institut de Robòtica* de la Universitat de València, se inició el desarrollo de una plataforma autónoma móvil con la financiación del proyecto CICYT TAP95-1086-C02-02: *Planificación de movimientos para vehículos autónomos en entornos industriales basados en percepción sensorial.* Proyecto dirigido por el Dr. Joan Pelechano, co-director del presente trabajo de investigación. Es entonces cuando se plantea la posibilidad de utilizar el formalismo log-polar desde un punto de vista práctico para sensorizar la plataforma autónoma.

En el **Machine Vision Lab.** de la Universidad de Virginia se entra en contacto con el trabajo del profesor Nandhakumar y se aborda la adaptación de algoritmos diferenciales para su utilización con el formalismo log-polar. En este entorno se inician los primeros trabajos que van a definir la arquitectura del módulo de sensorización log-polar.

Posteriormente se inicia la colaboración con el Departament d'Informàtica de la **Universitat Jaume I** en el desarrollo de algoritmos de visión foveal. Esta colaboración se realiza bajo el proyecto de la Generalitat Valenciana GV97-TI-05-27: *Seguimiento de*

objetos mediante visión activa foveal. Aplicación al control de vigilancia, que financia la implementación física del módulo reconfigurable.

Finalmente, y como continuación al proyecto CICYT que sirvió para desarrollar la plataforma autónoma, se inició el proyecto CICYT TIC98-1026: *Algoritmos de visión para navegación de robots móviles: desarrollo, implantación hardware e integración en las arquitecturas de control usando μkernels de reparto variable.* Durante este proyecto se ha realizado el diseño de los algoritmos en el módulo reconfigurable, su depuración y experimentación. Así mismo, se está realizando la integración del módulo reconfigurable en la plataforma móvil. De manera adicional, a partir de los resultados del presente trabajo de investigación se ha planteado la extensión del trabajo realizado en el proyecto de reciente aprobación GV99-116-1-14: *Técnicas y equipos de procesamiento de imágenes foveales en tiempo real para el análisis del entorno frontal de vehículos circulando a gran velocidad.*

El trabajo y estudio realizado en los diversos campos implicados en el presente trabajo de investigación han sido muy enriquecedores para el autor. La intención es que la investigación realizada responda a las necesidades que han justificado su inicio.

Parte I

Introducción y fundamentos

Capítulo 1

Introducción

1.1 Descripción del problema

La navegación de vehículos autónomos en entornos no estructurados es una tarea que cobra cada vez mayor importancia. El desarrollo de robots móviles autónomos viene justificado por la realización de tareas que o bien son peligrosas o simplemente son imposibles para operadores humanos. El desarrollo de nuevas plataformas autónomas o robots para realizar tareas determinadas lleva implícito una mejora de los mecanismos de navegación en entornos no estructurados. De esta manera, la plataforma autónoma debe ser capaz de detectar obstáculos en su trayectoria para poder evitarlos y detectar otros vehículos móviles tanto estando parada como en movimiento. El objetivo será siempre que la plataforma móvil tenga una sensorización que garantice la correcta navegación, y por tanto, pueda realizar de forma segura las tareas asignadas.

El amplio rango de sensores útiles para plataforma móviles incluye sensores de ultrasonidos e interruptores de contacto, pero sin duda los sensores más útiles por su rango de alcance y la cantidad de información que se puede extraer de sus medidas, son los sensores visuales.

La forma habitual de realizar esta sensorización visual es mediante la incorporación de una o varias cámaras CCD a la plataforma móvil. A la cámara CCD hay que añadir una etapa de digitalización y almacenamiento de las imágenes y una etapa que procese dichas imágenes e implemente algoritmos visuales útiles para navegación robótica.

La aproximación mediante sensores CCD es muy costosa en términos de utilización de recursos hardware dedicados a la sensorización. Las etapas de adquisición y procesado de la imagen implican, si se quiere realizar un procesado con una resolución (y por tanto precisión) aceptable, una gran cantidad de información a procesar. Si además existen limitaciones de tiempo real, como típicamente pasa en la navegación robótica,

toda esta información debe de ser procesada muy rápidamente.

Estas restricciones de velocidad de proceso y cantidad de información obligan a una implementación de los algoritmos de sensorización visual con hardware hecho a medida. Cabe hacer notar que esta solución implica una pérdida de flexibilidad en los módulos de sensorización visual. Si la aplicación de un cierto algoritmo requiere una circuitería realizada expresamente para implementar dicho algoritmo, entonces se están utilizando recursos exclusivamente para una aplicación concreta. Esta circuitería será un peso muerto innecesario cuando dichos algoritmos no sean útiles o aplicables.

Se han realizado paralelamente otras aproximaciones interesantes al procesado de imágenes poniendo énfasis en otras formas de representar la información. La representación Cartesiana, heredada de los sensores CCD, puede ser interesante para resolver ciertos problemas de visión artificial pero ha mostrado no ser la más adecuada para problemas de visión activa. En esta línea de razonamiento ha sido estudiada y utilizada la representación log-polar, conocida por que es el sistema de visión que incorporan muchos seres vivos, entre ellos el hombre.

La visión log-polar tiene una representación en la que la resolución es función de la proximidad al centro de la imagen. De esta manera se tiene una mayor resolución en el centro de la imagen manteniendo un amplio campo de visión. Es decir, se mantiene una resolución alta en la zona de interés, lo que permite un análisis de esa zona, combinada con un amplio campo de visión, lo que permite que se puedan apreciar cambios en la periferia de la imagen. Esta interesante forma de representación de la información visual consigue reducir de forma significativa la cantidad de información a procesar. Además de esta importante característica, el formalismo log-polar tiene otras propiedades que se mostrarán en el capítulo 3.

Se ha escogido el sistema de coordenadas log-polar por sus interesantes propiedades para la navegación robótica y por la reciente disponibilidad de un sensor log-polar que hace posible su utilización real. Algunos algoritmos útiles para navegación robótica se simplifican sobremanera al utilizar este formalismo, otros directamente se han desarrollado basándose en este sistema coordenado.

Por otra parte es interesante una cierta flexibilidad en la elección del algoritmo a implementar mediante visión foveal. Si la plataforma está parada puede ser interesante detectar objetos que se muevan a su alrededor, si se está moviendo detectar objetos que se muevan independientemente de ella o incluso hacia ella, o puede ser interesante calcular el tiempo hasta que se produzca el impacto con un objeto móvil o estático. Realizando un estudio de los diversos algoritmos que pueden ser interesantes para navegación robótica se puede realizar una clasificación y observar las características comunes de los **algoritmos diferenciales.** Estos algoritmos se caracterizan por que

utilizan básicamente la diferenciación para extraer información relevante. Se utilizan tanto derivadas espaciales o gradientes como derivadas temporales de orden n.

En cualquier caso, la necesidad de realizar diversas tareas con objetivos similares, (algoritmos de procesamiento de imágenes útiles para la navegación), pero con soluciones distintas (cada algoritmo concreto) plantea de nuevo el problema de que una solución hardware para cada caso no es viable por cuestiones de espacio, consumo de potencia y coste.

La reciente aparición de lógica programable, reconfigurable en tiempo de ejecución, posibilita tener simultáneamente la velocidad del hardware con la flexibilidad del software. Este objetivo sería muy deseable para una plataforma móvil que pudiera reconfigurar el hardware disponible para realizar la tarea más adecuada en cada momento.

Estos objetivos se hicieron patentes en la colaboración con el **Instituto de Robótica de la Universitat de València** en el proyecto CICYT TAP95-1086-C02-02 con el título: *Planificación de Movimientos para Vehículos Autónomos en Entornos Basados en Percepción Sensorial* dirigido por el Dr. Joan Pelechano. Por otra parte la disponibilidad a mediados de 1995 del sensor log-polar CMOS desarrollado por el Dr. Fernando Pardo en el IMEC (Bélgica) ofrecía la posibilidad real de incorporar procesamiento de imágenes log-polares en la plataforma móvil. Los recientes avances de la lógica programable y la experiencia en su utilización sugirió la utilización de estos dispositivos. La colaboración en posteriores proyectos de investigación fijó el diseño de la arquitectura reconfigurable y financió su implementación.

De la misma manera la colaboración con la **Departamento de Informática de la Universitat Jaume I** en el proyecto Generalitat Valenciana GV97-TI-05-27 con el título: *Seguimiento de Objetos Mediante Visión Activa Foveal. Aplicación al Control de Vigilancia*, dirigido por el Dr. Filiberto Pla. Esta colaboración permitió definir e implementar la arquitectura reconfigurable. Adicionalmente la colaboración con el Instituto de Robótica ha continuado con el diseño de algoritmos de visión dentro de la arquitectura reconfigurable, trabajo enmarcado en el proyecto CICYT TIC98-1026 con el título: *Algoritmos de Visión para Navegación de Robots Móviles: Desarrollo, implantación hardware e integración en la arquitecturas de control usando μkernels de reparto variable.* Proyecto dirigido por el Dr. Juan de Mata Domingo

El planteamiento de la necesidad de dotar a una plataforma móvil, con una sensorización visual adecuada para la navegación robótica, es el núcleo de los objetivos del presente trabajo de investigación, que a continuación se desarrollan de forma pormenorizada.

1.2 Objetivos

A partir de la descripción del problema realizada en la sección anterior se pueden concluir que el objetivo del presente trabajo de investigación será:

El desarrollo de un módulo reconfigurable de sensorización visual, para navegación de robots móviles en tiempo real, que incluya un amplio rango de algoritmos útiles. Desarrollo de algoritmos de visión bajo esta arquitectura, con la posibilidad de que la plataforma móvil seleccione el algoritmo más adecuado, según la tarea concreta a realizar.

Para la realización de este objetivo se han desarrollado las siguientes etapas o fases:

- Elección del sistema de sensorización visual más adecuado, en este caso el sistema log-polar. Esto se ha argumentado en la sección anterior y se justificará de forma detallada en el capítulo 3.
- Estudio y selección inicial de un conjunto de algoritmos útiles para navegación de vehículos autónomos. Extracción de las características comunes a todos ellos.
- Definición y diseño de la arquitectura reconfigurable que implementará los algoritmos del rango definido, y los algoritmos diseñados expresamente para esta arquitectura.
- Diseño de una metodología para la realización de los algoritmos que cumplan las especificaciones propuestas bajo la arquitectura reconfigurable. Tanto los algoritmos iniciales que han servido para definir la arquitectura, los expresamente diseñados para ser implementados en la arquitectura, como otros que estén dentro del rango cubierto, deben ser implementables en esta arquitectura siguiendo la metodología de diseño propuesta.
- Diseño de algoritmos diferenciales en la arquitectura propuesta útiles para navegación de plataformas autónomas. En particular se diseñaran dos algoritmos diferenciales que se simplifican especialmente en coordenadas log-polares.
  1. Diseño de un algoritmo de detección de movimiento que detecte movimiento independiente del movimiento de la cámara.
  2. Diseño de un algoritmo de cálculo de tiempo al impacto.
- Implementación física de la arquitectura propuesta y de los algoritmos diseñados para navegación robótica.
- Comprobación experimental del funcionamiento del módulo reconfigurable y de los algoritmos propuestos. Parametrizar su funcionamiento y limitaciones.

Cada una de las etapas propuestas se basa en la realización con éxito de la an-

terior. En la siguiente sección se describe la organización de la presente memoria de investigación.

1.3 Contenidos

La presente memoria ha sido dividida en cuatro partes, que recogen el trabajo de investigación realizado. La primera parte recoge el planteamiento del problema a resolver y el estado de la investigación. La segunda parte describe el diseño de la arquitectura propuesta y de los algoritmos realizados bajo esta arquitectura. La tercera parte recoge las simulaciones y los resultados experimentales obtenidos con el módulo reconfigurable ya implementado. La cuarta parte recoge las conclusiones, aportaciones que se han realizado y trabajo futuro que se propone realizar.

La quinta parte incluye las citas bibliográficas a las que se hace referencia, más las publicaciones que ya se han realizado del presente trabajo de investigación. Por último, en la sexta parte, se incluyen los apéndices con el código VHDL diseñado, esquemas, y todo el material que se ha considerado oportuno incluir aquí para hacer más ligera la lectura de la presente memoria.

Cada parte a su vez ha sido organizada en capítulos que describen los diversos objetivos y fases del trabajo de investigación.

En el **capítulo 1** se realiza una breve introducción y descripción general del problema a resolver, así como los objetivos del trabajo de investigación.

En el **capítulo 2** se revisan las aproximaciones al procesamiento de imágenes mediante componentes estándar. También se revisan las principales máquinas reconfigurables existentes, para estudiar su posible aplicación como módulo de sensorización visual.

En el **capítulo 3** se expone el formalismo log-polar, se muestran las propiedades que lo hacen tan útil para navegación robótica y se justifica la elección de este sistema coordenado.

En el **capítulo 4** se hace una revisión del rango de algoritmos que sería interesante cubrir con el módulo reconfigurable. A partir de este estudio, se extraen las limitaciones y posibilidades que tendrá la arquitectura a implementar. Basándose en las características anteriores, se diseña la arquitectura segmentada con los elementos de proceso que van a realizar cada etapa del cauce, se muestra la escalabilidad de la arquitectura, y se diseña el flujo de información entre los módulos de proceso.

En el **capítulo 5** Se propone una metodología genérica de diseño de algoritmos en la arquitectura propuesta. Se muestra la similitud de la plataforma móvil con un sistema

empotrado y se propone la utilización de técnicas de codiseño binario.

En el **capítulo 6** se diseña de forma teórica un algoritmo de detección de movimiento independiente del movimiento de la plataforma móvil en coordenadas log-polares. Posteriormente se particulariza su diseño a la arquitectura propuesta con la metodología expuesta en el capítulo 5.

En el **capítulo 7** se expone un algoritmo de detección de tiempo al impacto desarrollado en coordenadas log-polares, y se muestra su diseño mediante la metodología propuesta.

En el **capítulo 8** se parametrizan los algoritmos desarrollados, se obtienen ecuaciones analíticas que expresan la magnitud de los errores en los algoritmos , y se proponen estrategias para reducirlo. Así mismo se estudia la relación entre la geometría del sensor y su respuesta a cambios dinámicos de la escena.

En el **capítulo 9** se comprueba experimentalmente el correcto funcionamiento del módulo reconfigurable programado con el algoritmo diseñado en el capítulo 6. Adicionalmente, se automatiza la elección de los dos parámetros del algoritmo (intervalo de adquisición y umbral) en función de las características de la escena.

En el **capítulo 10** se comprueba experimentalmente el correcto funcionamiento del módulo reconfigurable programado con el algoritmo diseñado en el capítulo 7. Se evalúa en qué casos el comportamiento es bueno y en que casos no lo es.

En el **capítulo 11** se resumen los resultados, las conclusiones del trabajo de investigación y las aportaciones realizadas. Así mismo se exponen las posibles líneas de extensión de la investigación y el trabajo futuro a realizar basándose en las conclusiones de esta tesis.

Finalmente se adjuntan referencias bibliográficas del presente trabajo. Adicionalmente se incluyen las publicaciones ya realizadas. Los **apéndices** recogen listados de las fuentes de VHDL de los diversas etapas de los algoritmos implementados y otros elementos que han sido ubicados ahí para hacer más ligera la lectura de esta tesis.

Capítulo 2

Arquitecturas de adquisición y procesamiento de imágenes

2.1 Introducción: La aproximación clásica

El procesamiento de imágenes es un campo de investigación ampliamente estudiado. Son muchas las aplicaciones que utilizan la visión artificial como mecanismo de sensorización de entrada de información a los actuadores. Para realizar el procesado de estas imágenes, extraer la información que se considera relevante, e interpretar esta información son necesarias dos etapas: el sistema de adquisición y el sistema de procesamiento.

El sistema de adquisición se encarga de transformar la información visual externa, en una representación comprensible por un sistema digital. De esta manera un sistema de adquisición consta de un sensor visual, una etapa de digitalización (si es necesaria) de la señal del sensor y un dispositivo de almacenamiento de las imágenes.

Dentro del rango de sensores, los más habituales, por la calidad de la imagen suministrada, son los sensores CCD [Sch87]. Estos sensores suministran una señal analógica que debe ser digitalizada. Adicionalmente, debido a la estructura particular de la tecnología CCD, debe leerse toda la imagen aunque sólo haya interés en analizar una parte. Estas características, de alta resolución y acceso secuencial, obligan a que la etapa de adquisición deba tener suficiente memoria para almacenar las imágenes.

La etapa de procesamiento clásica suele ser un computador de propósito general con tarjetas aceleradoras. Estos módulos implementan, mediante procesadores de señal digital (DSPs), funciones que serían muy costosas en tiempo de ejecución para un procesador secuencial. El problema de esta aproximación es la gran cantidad de información que debe ser procesada típicamente con restricciones de tiempo real. Las tarjetas acele-

radoras con procesadores de señal digital aumentan la velocidad de proceso (comparado con un procesador secuencial de propósito general) pero pueden no ser suficiente para tareas complejas [BFBB92].

Otro problema de la aproximación clásica es la gran cantidad de información que puede ser necesario almacenar temporalmente. Muchos algoritmos se dividen en etapas que producen resultados (imágenes) parciales que a su vez son entradas a otras etapas. Si además se quiere tener una precisión aceptable en el análisis de la zona de interés, las imágenes deben tener una resolución suficiente. Esto implicará que se debe almacenar una gran cantidad de información que además debe fluir muy rápidamente en la etapa de procesamiento.

La aproximación clásica ha sido superada tanto desde el punto de vista de la adquisición como desde el punto de vista del procesado [Col95]. En las siguientes secciones se muestran algunos desarrollos novedosos, tanto para la etapa de adquisición como para la etapa de procesamiento.

2.2 Sensores avanzados

Se han desarrollado sensores de visión que superan en determinadas prestaciones las del CCD para procesamiento de imágenes. La tecnología CCD sigue siendo la que ofrece una mejor calidad de imagen desde el punto de vista *humano* de la palabra. La alta relación señal/ruido y otros parámetros como el contraste y los niveles de gris de dicha tecnología, ofrecen la mejor calidad visual hasta la fecha [Hyn97], [SNO+97].

Para aplicaciones robóticas se tendrá una buena calidad de imagen cuando la imagen sea lo suficientemente buena para realizar la tarea destinada. De esta manera no es tan importante para un robot la *calidad* de la imagen, siendo importantes otros parámetros como pueden ser: acceso aleatorio a cualquier zona de la imagen, mayor resolución en zonas de interés de la imagen, preprocesado interno en el sensor, o que el sensor directamente suministre un valor digital en una escala de grises.

La tecnología CMOS no ofrece tanta *calidad* de imagen desde el punto de vista humano, pero se han desarrollado una serie de sensores que incorporan características que los hacen muy interesantes desde el punto de vista de la visión artificial. Una de estas características tiene relación con una de las fotocélulas diseñada con tecnología CMOS. Se pueden realizar elementos fotosensibles de conversión de la luz incidente en corriente eléctrica, en lugar de integrar la luz, consiguiéndose de esta manera un control más sencillo del sensor y un acceso aleatorio a las celdas [DSM+96]. De esta manera se puede acceder solamente a la parte de la imagen que se quiera analizar, no siendo necesaria la adquisición y almacenamiento de toda la imagen.

Otra característica de algunas celdas visuales CMOS es que las fotocélulas tienen una respuesta logarítmica a la luz [RD92], como en el ojo humano, lo cual las puede hacer interesantes para trabajar en entornos naturales y con alto contraste [RD93b].

Finalmente, otra característica que está haciendo que la tecnología CMOS se vaya utilizando cada vez más en visión artificial, es que se puede incorporar dentro del mismo circuito fotosensible, una etapa de procesado de la imagen. Este procesado interno puede ser la aplicación de algoritmos sencillos como detección de esquinas [HKL95a], detección simple de movimiento [KMB+95], o incluso llegar a implementar un **microsistema** sencillo de forma completa [BCM+96]. Se han llegado a completar sensores visuales CMOS que incluyen un etapa de procesado que implementa algún algoritmo útil para navegación robótica como es el cálculo del flujo óptico [RP98].

Recientemente se han desarrollado sensores CMOS con matrices analógicas que pueden realizar un procesamiento continuo sencillo de la información visual de entrada [LFE+99]. La incorporación de procesado dentro del sensor visual consigue liberar gran cantidad de espacio y hacer un sistema de proceso más rápido. Como desventaja de esta aproximación, se tiene la falta de configurabilidad del circuito hecho a medida para resolver una tarea concreta.

La tecnología CMOS ha sido la elegida para sensorizar el módulo reconfigurable, debido al reciente desarrollo de sensores CMOS con las ventajas anteriormente descritas (acceso aleatorio y respuesta logarítmica a la luz). Se justificará en el capítulo 3 la elección del sistema coordenado log-polar y se revisará el sensor log-polar CMOS que se ha escogido.

2.3 Arquitecturas especiales para procesamiento de imágenes

Se puede realizar una división de los algoritmos de visión artificial en función del nivel de complejidad del procesamiento [Hir93], tal como se muestra en la figura 2.1. El nivel bajo en el procesamiento de imágenes transforma imágenes en imágenes, típicamente realizando alguna operación sencilla de filtrado. El nivel medio extrae características interesantes de la imagen, como son bordes, circularidades o texturas. Esta etapa transforma imágenes en características. El nivel más alto de procesado tiene como entrada las características extraídas en la etapa anterior y como salida otra función característica más compleja. Esta es la etapa que suministra la información necesaria a un sistema de toma de decisiones que controle unos actuadores.

Algunas tareas típicas del nivel alto de procesado son la segmentación y reconoci-

miento de patrones. Un algoritmo de visión artificial general puede tener etapas en los tres niveles descritos, y una arquitectura genérica debe ser capaz de cubrir los tres niveles.

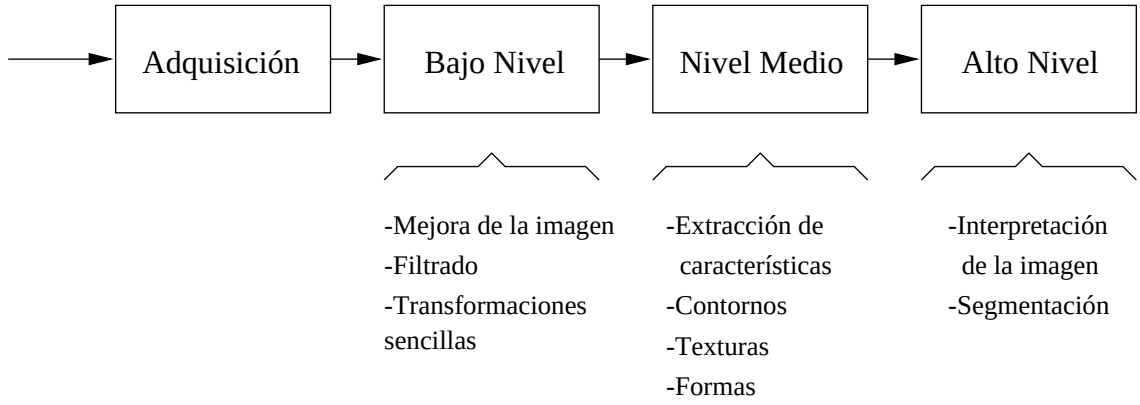


Figura 2.1: *Niveles de procesamiento para una aplicación típica de visión artificial*

La aproximación para la etapa de procesamiento de un solo procesador realizando tareas de forma secuencial ha sido ampliamente superada por las técnicas de procesamiento paralelo [Pit93] [GHHS96]. Máquinas para el procesamiento de imágenes basadas en arquitecturas SIMD y MIMD con buses estándar han sido desarrolladas con prestaciones de tiempo real.

Arquitecturas paralelas con componentes estándar

Dentro del programa *Esprit long term research*, un consorcio Europeo ha desarrollado una arquitectura heterogénea MIMD-SIMD llamada CC/IPP [JV97].

La filosofía de esta plataforma de procesado de imágenes ha sido utilizar tarjetas comerciales en una arquitectura escalable. Cada nodo de proceso MIMD es una tarjeta madre con un procesador comercial (PowerPC o Pentium) conectado mediante un enlace rápido de red a un encaminador. Estas tarjetas tienen a su vez una interfase PCI que permiten su conexión con matrices comerciales SIMD y con otros elementos para la adquisición y almacenamiento de imágenes. En la figura 2.2 se puede observar un esquema de la arquitectura CC/IPP.

Esta plataforma se está utilizando en la actualidad para la realización de tareas complejas en tiempo real con éxito. La ventaja de esta aproximación es la utilización de tarjetas y circuitos comerciales de forma escalable, beneficiándose de las rápidas mejoras que se producen en el mercado. Otra ventaja es la flexibilidad de este enfoque conseguida con su programabilidad. Utilizando las tarjetas SIMD y MIMD con sus correspondientes librerías de programación, el rango de aplicaciones de esta arquitectura

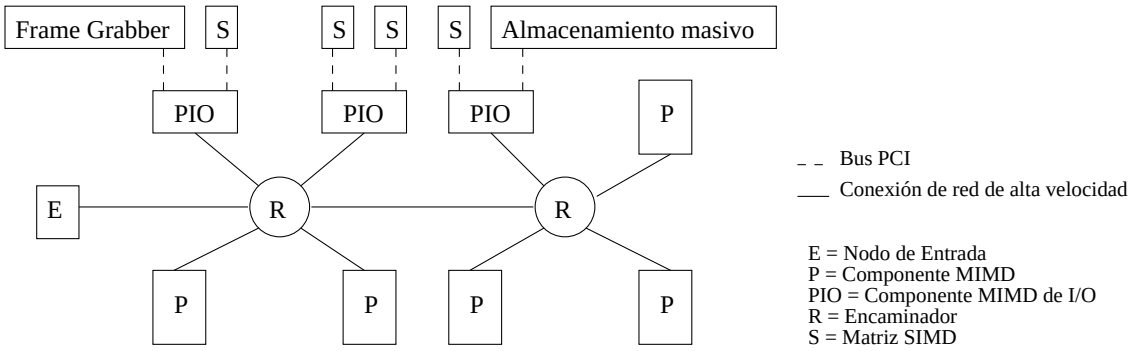


Figura 2.2: *Arquitectura CC/IPP*

es muy grande.

Siguiendo la misma política de utilización de tarjetas comerciales PCI se ha desarrollado la máquina GFLOPS [HFB97]. Esta máquina explota el paralelismo de los algoritmos de visión mediante el uso de utilidades software como PVM (*Parallel Virtual Machine*) y una red especial de interconexión entre los diversos módulos de proceso. De nuevo se tienen las ventajas del software, es decir, flexibilidad y reconfigurabilidad.

Las desventajas de estas aproximaciones son el alto coste y el tamaño físico que ocuparía una configuración medianamente potente. Si se desea incorporar un sistema realizado con componentes estándar como módulo de sensorización visual sensorial, las prestaciones serán proporcionales a los recursos utilizados. En el caso de una plataforma móvil, la utilización masiva de tarjetas de proceso y buses de interconexión entre ellas es prohibitiva. No es posible dedicar tanto espacio, peso y potencia para una tarea de sensorización.

Redes neuronales y transputers

Existen otros enfoques distintos al problema del procesamiento de imágenes: las redes neuronales y los Transputers.

Las redes neuronales han sido ampliamente estudiadas y aplicadas al campo del procesamiento de imágenes [JMM96], [Sme95], y más concretamente al procesamiento de medio [RD93a] y alto nivel [XIM91], [IXAM92a].

La aplicación de las redes neuronales a problemas concretos de segmentación da buenos resultados en ciertos casos [PP93]. Desafortunadamente la solución de los problemas de la navegación robótica no consiste en un simple problema de segmentación. Tal como se argumentará en el capítulo 3, no es posible reducir el problema a computar un mapa de vectores de flujo óptico y segmentar la imagen a partir de estos vectores. Las redes neuronales sólo cubrirían la parte de alto nivel del algoritmo, y otras dificul-

tades como son el aprendizaje, y su costosa implementación hardware, desaconsejan su uso.

Los transputers también se han utilizado en tareas de visión artificial y procesamiento de imágenes [CMRU93], [BUR96b]. La posibilidad de explotar la concurrencia y el paralelismo de un algoritmo en una red de transputers es algo que se realiza con éxito en tareas concretas [MWD91]. De nuevo, al igual que con las redes neuronales, tenemos que se han realizado implementaciones de algoritmos específicos, pero no se ha utilizado una red de transputers que pueda implementar simultáneamente varios algoritmos útiles para navegación de plataformas móviles autónomas. Estos procesadores tienen el problema del alto coste, lo cual dificulta su introducción como parte de un sistema de sensorización.

Lo más deseable para un sistema digital es que tenga la reconfigurabilidad del software y la rapidez del hardware. La aparición en los años 80 de los primeros dispositivos lógicos programables ha hecho posible optimizar el silicio utilizado en un sistema digital. Hasta la aparición de los primeros dispositivos lógicos programables las únicas opciones para diseñar hardware digital eran, o bien el diseño de circuitos integrados hechos a medida, o bien la utilización de componentes estándar LSI-MSI conectados entre si.

En el apéndice A se resumen las características básicas de los diferentes tipos de dispositivos lógicos programables. De este análisis y de las particularidades del módulo reprogramable se obtienen las características de los dispositivos lógicos programables que van a implementar el módulo reconfigurable.

En el caso del módulo reconfigurable, la tecnología debe ser la SRAM para permitir simultáneamente la programabilidad en el sistema y una razonable escala de integración.

Tras realizar en el apéndice A un análisis de las diferentes clases de dispositivos se puede concluir que los dispositivos con arquitecturas híbridas son los más adecuados. En estos dispositivos se intenta mantener la velocidad de las CPLDs, manteniendo simultáneamente la granularidad de las FPGAs.

2.4 Arquitecturas reconfigurables

Las primeras máquinas reconfigurables se desarrollan a principios de los 90, cuando ya se dispone de FPGAs suficientemente complejas. Es entonces cuando se empieza a plantear la posibilidad de realizar máquinas totalmente reconfigurables basadas en FPGAs. La lista de este tipo de máquinas es muy grande, (más de 70 a la escritura de este capítulo), y va incrementándose rápidamente [Guc99].

Existe una gran variedad de máquinas reconfigurables, con arquitecturas específicas

diseñadas para ellas. A partir de la cantidad de recursos hardware reconfigurables de que disponen se puede realizar una clasificación en tres grandes grupos:

1. Sistemas modulares multitarjeta.
2. Redes de FPGAs monotarjeta.
3. Tarjetas coprocesador.

La mayoría de estas máquinas son de propósito general y de arquitecturas muy abiertas. Su utilización para visión por computador en tiempo real se plantea debido a que se consigue la velocidad del hardware para los algoritmos de visión hechos a medida. A continuación se analizan algunos ejemplos significativos de cada grupo, sopesando su posible utilización para el módulo de sensorización visual reconfigurable. Finalmente se introducirá la aparición del hardware que puede evolucionar como una de las últimas líneas de investigación en máquinas reconfigurables.

2.4.1 Sistemas modulares multitarjeta

Han sido desarrollados distintos sistemas modulares reconfigurables multitarjeta, todos ellos con arquitecturas a medida escalables y modulares. La gran potencia de cálculo obtenida combinando la configurabilidad del hardware y la modularidad, tiene como desventaja el tamaño de estos sistemas. Estos grandes sistemas pueden trabajar como *maestros*, aunque existe siempre la comunicación con una estación de trabajo para programar las FPGAs y realizar tareas de control menores.

Para una plataforma móvil, con las limitaciones habituales de espacio, peso y consumo de potencia de este tipo de vehículos, no es posible cargar con todo un *rack* de tarjetas con múltiples FPGAs. La máquina reconfigurable de propósito general modular más potente y popular de la actualidad es Splash-2, desarrollada en el *Institute for Defense Analyses. Supercomputing Research Center*. La tecnología desarrollada a partir del proyecto Splash, y su secuela Splash-2, ha generado a su vez múltiples sistemas reconfigurables.

Splash-2: Una máquina reconfigurable de propósito general

Una de las máquinas reconfigurables más potentes y utilizadas es Splash-2 [BAK96]. Esta máquina no tiene una arquitectura específica orientada al procesamiento de imágenes. De hecho se ha utilizado con éxito en aplicaciones distintas el cálculo numérico [PA96]. En cualquier caso va a ser analizada porque se han realizado un gran número de aplicaciones de visión artificial sobre ella [AA95], consiguiéndose procesar flujos de decenas de imágenes por segundo.

La arquitectura de Splash-2 puede observarse en la figura 2.3. La máquina consta de hasta 16 tarjetas conectadas a una tarjeta especial que hace de interfase con una estación de trabajo Sparc. La interfase proporciona un ancho de banda de hasta 50 Mbytes/sec sobre 3 canales bidireccionales de DMA.

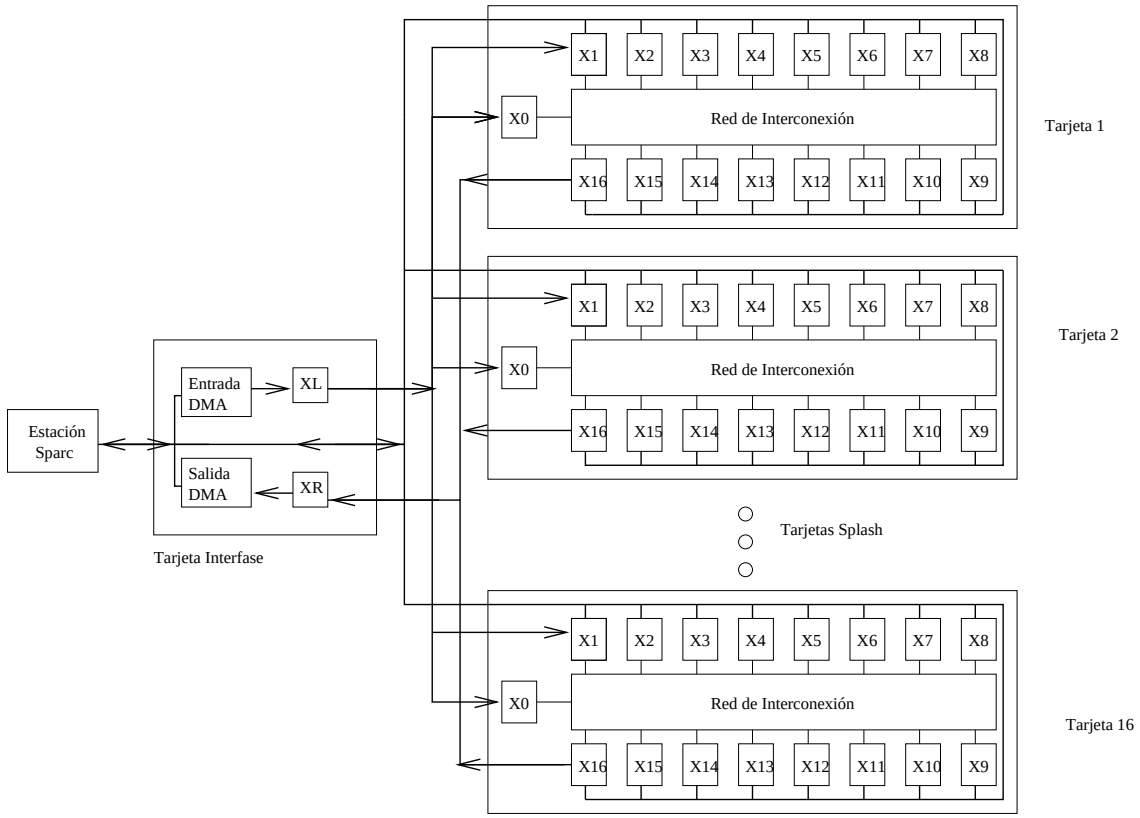


Figura 2.3: *Arquitectura del sistema Splash-2*

Cada una de las 16 tarjetas tienen a su vez 16 elementos de proceso (EPs) ordenados en una red lineal y totalmente conectados mediante una red de interconexión de 16x16. Cada EP es una FPGA XilinX 4010 y tiene 1/2 Mbyte de memoria directamente accesible por el *host*.

La red de interconexión y el flujo de datos puede configurarse como un cauce segmentado, como una matriz sistólica, o como una matriz SIMD (*Single Instruction Multiple Data*) con múltiples redes de interconexión. Los algoritmos de procesado de imágenes pueden implementarse en Splash-2 siguiendo dos aproximaciones distintas:

- Aproximación SIMD con la imagen distribuida en los EPs, con cada EP responsable de una porción de la imagen.
- Aproximación en la que se construye un cauce segmentado con los EPs. La imagen fluye a través del cauce y el procesado se divide en diversas etapas.

La ventaja de esta arquitectura es la potencia y la flexibilidad que se consigue con la red de interconexión reconfigurable y los múltiples EPs. La desventaja, al igual que otras arquitecturas basadas en componentes estándar, va a ser la cantidad de recursos y tarjetas necesarias para conseguir tiempo real. De nuevo la complejidad de diversos algoritmos interesantes y el tamaño de las imágenes imponen muchos recursos hardware, aunque estos sean reconfigurables.

Otro gran sistema reconfigurable es el DFFC, *Data Flow Functional Computer*, desarrollado al LIMSI-CNRS de Francia.

DFFC: Una plataforma para el rápido prototipado del hardware

El propósito de esta arquitectura reconfigurable es el rápido desarrollo de hardware para visión en tiempo real [QKSZ94].

El sistema consiste en una matriz regular 3D de FPGAs especialmente desarrolladas para este sistema llamadas FPOAs, (*Field-Programmable Operator Arrays*). Las FPOAs tienen una arquitectura similar a la de las FPGAs híbridas descritas en la sección A.3 y sus características se resumen en la tabla 2.1.

En la figura 2.4 se observa la arquitectura de matriz 3D de la máquina DFFC. El sistema puede contener desde 1 hasta 8 tarjetas de 8x8 FPOAs. Directamente a la red de FPOAs se pueden conectar las entradas y salidas de vídeo digitalizadas. Una estación de trabajo SPARC2 se encarga de la programación de las FPOAs.

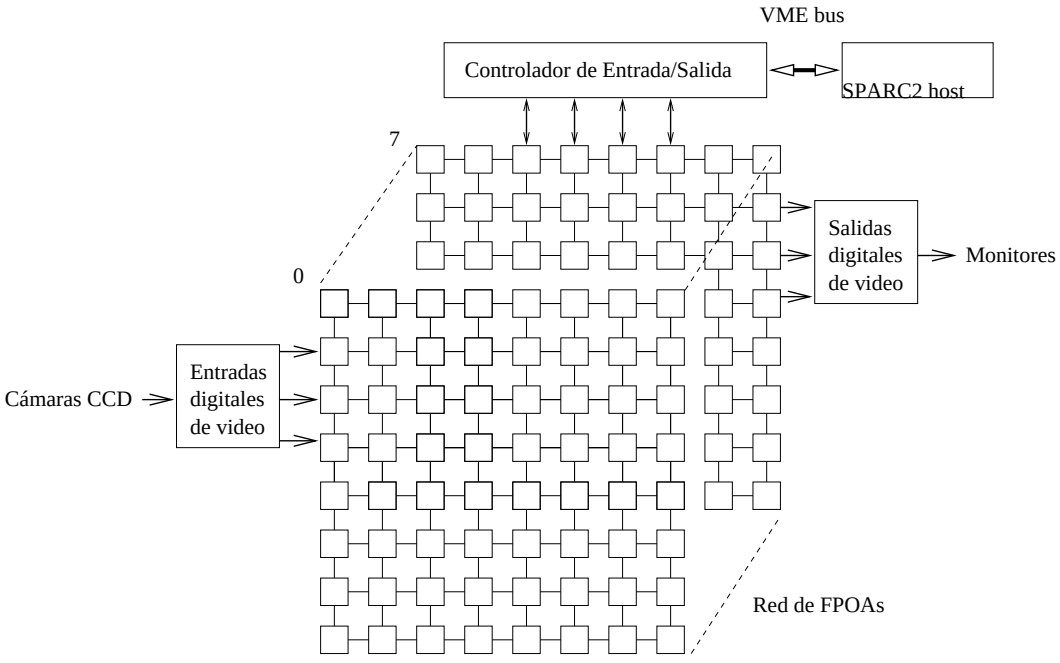


Figura 2.4: *Arquitectura del sistema DFFC*

Encapsulado	144 pines PGA
Tecnología	1 μm CMOS
Área	8.9 mm x 9.6 mm
Frecuencia de funcionamiento	25 MHz
Pines de Entrada/Salida	100 (distribuidos en 10 puertos)
Tecnología de programación	SRAM
Bits de configuración	10.294
Puertas lógicas equivalentes	33.000
Memoria interna	8.5 Kbits

Tabla 2.1: *Características de las FPOAs*

Se han desarrollado herramientas y librerías específicas de programación de los algoritmos. A partir de la prueba de la eficiencia de un algoritmo *hardware* programado en el DFFC se puede derivar de forma automática la implementación en circuitos integrados hechos a medida.

Con esta máquina se han desarrollado con éxito diversas aplicaciones de filtrado, detección de esquinas y algoritmos no muy complejos. La principal ventaja de esta arquitectura es el rápido diseño de los algoritmos de visión artificial, su verificación experimental, y su translación a circuitos hechos a medida. La principal desventaja de esta arquitectura viene a ser de nuevo el excesivo tamaño. Además su no optimización para tareas de navegación robótica, hace que no sea posible su utilización como módulo de sensorización visual de una plataforma autónoma.

Se han estudiado otras arquitecturas modulares multitarjeta reconfigurables. Todas tienen una gran capacidad de cálculo pero todas tienen las mismas desventajas en cuanto a dimensiones y utilidad para una plataforma móvil.

Como ejemplos más interesantes cabe citar la plataforma *ArMen* desarrollada en el *Laboratoire d'Informatique* de Brest [CPP+94]. Esta plataforma, conectada a una estación de trabajo mediante SBUS, combina FPGAs y transputers T805 siguiendo la filosofía de conectividad de los transputers.

Análogamente se ha desarrollado en la Universidad de Mannheim un sistema multiprocesador basado en FPGAs con una filosofía similar al DFFC [HKL+95b]. En este caso las tarjetas con FPGAs contienen 16 XC4013 + 11XC5005H, además de memoria SRAM y conectores para transputers. Este sistema se está utilizando para procesado en tiempo real de los datos suministrados por el acelerador de partículas ATLAS del CERN.

2.4.2 Redes de FPGAs monotarjeta

La siguiente categoría de máquinas reconfigurables serían máquinas de tamaño medio, implementadas en una sola tarjeta. Habitualmente no son escalables ni modulares, salvo pequeñas ampliaciones internas de memoria o cambio/adición de unas pocas FPGAs. Estas tarjetas siempre actúan como esclavos de un sistema maestro para el que realizan alguna tarea específica.

Suelen tener un buen rendimiento para aplicaciones de poca complejidad como filtrado, y convoluciones. El principal problema de esta categoría de sistemas es su falta de escalabilidad y capacidad de proceso para implementar algoritmos complejos de visión. Se gana en ligereza del sistema, con respecto a los sistemas escalables, pero se pierden prestaciones. La utilización de estas tarjetas es adecuada para aplicaciones muy concretas, pero no cubre las necesidades de los algoritmos de visión útiles para una plataforma móvil.

WILDFORCE: Una tarjeta reconfigurable para bus PCI

Recientemente, a partir de la tecnología transferida desde el proyecto Splash2, la empresa Annapolis Micro Systems ha desarrollado una tarjeta reconfigurable para bus PCI llamada WILDFIRE [FDP96]. La arquitectura WILDFIRE es similar a las tarjetas de Splash-2, incorporando cinco EPs, colas FIFO, y conectores especiales de entrada salida para matrices SIMD, tal como se muestra en la figura 2.5.

Existen diversos modelos de la tarjeta WILDFIRE, en función del tamaño de las FPGAs que sirven como elementos de proceso. La FPGA más pequeña que se utiliza en la familia WILDFIRE es la XC4013E con 1536 flip-flops y la mayor es la XC4036EX con 3168 flip-flops. Con esta tarjeta se han desarrollado con éxito diversos algoritmos de procesado de imágenes en tiempo real como convoluciones, filtros, y otros algoritmos de bajo nivel.

Las prestaciones conseguidas para estas aplicaciones son realmente buenas, llegando a las 30 imágenes por segundo y frecuencias de reloj de hasta 40 MHz. De nuevo se centra la solución del problema en intentar procesar las imágenes lo más rápidamente posible, pero siempre con la representación Cartesiana de gran tamaño. No se tiene constancia de que se haya podido utilizar esta tarjeta para implementar algoritmos de nivel medio o alto, como cálculo de flujo óptico, o segmentación de imágenes. Para la solución de estos algoritmos más complejos sería necesario una arquitectura con mayor cantidad de recursos, como por ejemplo Splash-2.

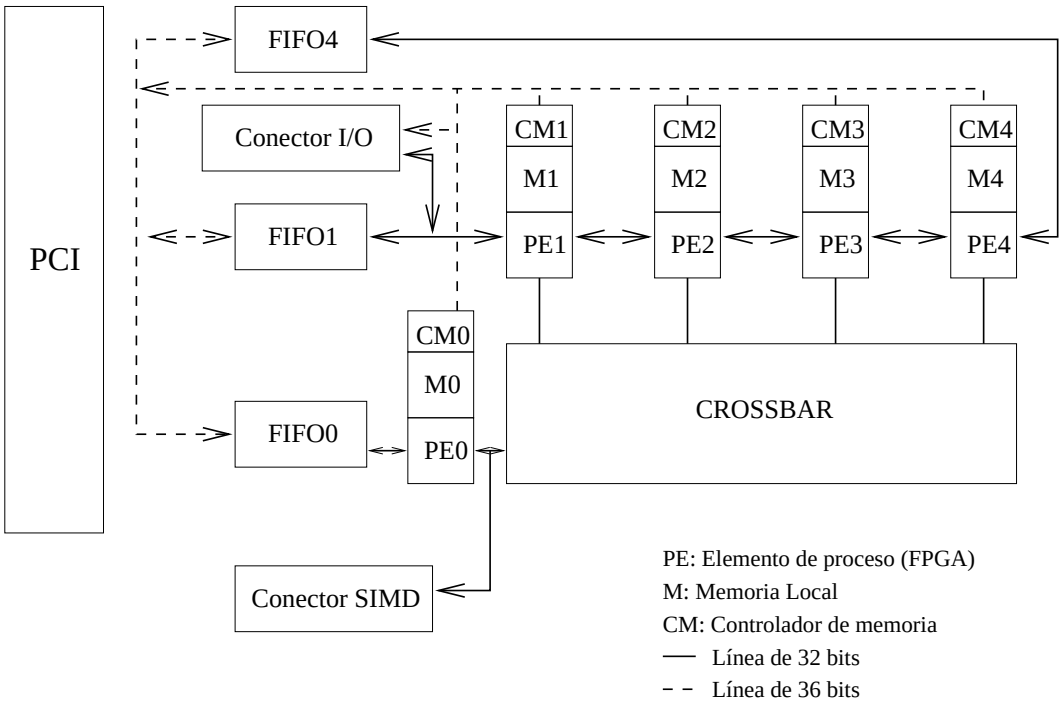


Figura 2.5: *Diagrama de la arquitectura WILDFIRE*

Spectrum RC: Una plataforma reconfigurable para el procesado de vídeo

Otro ejemplo de una reciente máquina de tamaño medio, reconfigurable y orientada al procesado de imágenes de vídeo es el reciente sistema *Spectrum RC* de Giga Operations Corporation [Tay96].

El sistema Spectrum puede soportar hasta 16 pequeños módulos de proceso llamados XMODS conectados a la placa madre. Cada XMOD contiene 2 FPGAs XC4010E y 2 bancos de 1 M x 16-bit DRAM (8 MB), aunque se están desarrollando XMODs con diferentes FPGAs y diferentes configuraciones de memoria. El sistema ha sido desarrollado para implementar filtrado digital complejo y aplicaciones de procesado de imágenes en tiempo real con múltiples fuentes de vídeo (hasta 16 NTCS/SVGA).

En esta plataforma se han implementado múltiples aplicaciones de filtrado de vídeo, convoluciones y filtrado no lineal. Por su naturaleza esta plataforma es en realidad una máquina de procesado de vídeo en tiempo real. No se tiene noticia de aplicaciones de más alto nivel implementadas en la plataforma y, por su arquitectura específica, la resolución de un problema más complejo desbordaría la capacidad de la tarjeta, limitada por el número y capacidad de XMODs.

Otras tarjetas de redes de FPGAs reconfigurables son de aplicación más específica si cabe. Así por ejemplo el sistema GANGLION es una tarjeta que implementa redes neuronales utilizando 24 FPGAs 3090 de Xilinx [CB92].

Otro ejemplo de arquitectura de matrices de FPGAs de utilización específica en una sola tarjeta se puede encontrar en [Box94]. En este caso se trata de una red matricial de $4x4$ FPGAs 4013 de Xilinx. Este sistema se utiliza para realizar prototipado hardware de sistemas previamente a su implementación física.

2.4.3 Tarjetas coprocesador

Este grupo de máquinas reconfigurables tienen de arquitecturas más sencillas y menor número de FPGAs, típicamente en el rango 1-3.

Estos sistemas siempre son esclavos de un maestro central que los programa, y para el que realizan alguna tarea hardware, liberando al procesador central de trabajo. Es por esto que el nombre adoptado para estos sistemas reconfigurables sencillos sea de tarjetas coprocesador. Aplicaciones típicas de estos sistemas son algoritmos muy sencillos, como filtrados simples, operaciones sistemáticas de las que liberar al maestro, y utilización en docencia.

Estas tarjetas sencillas no son escalables internamente. Siempre se puede conectar más de una de estas tarjetas al maestro mediante un bus estándar, pero no son escalables internamente. Claramente este tipo de sistemas reconfigurables quedan descartados por no poder implementar algoritmos suficientemente complejos, como son los útiles para navegación robótica.

Ejemplos de estas tarjetas sencillas son la APS-X208 [AD98], BORG y BORG-II, con 2 FPGAs de la familia Xilinx 4000 [CSM92]. en estos 3 casos una de las 2 FPGAs sirve como interfaz con el maestro y la otra FPGA implementa la funcionalidad de la tarjeta.

Otras tarjetas son interesantes plataformas en las que se experimentan técnicas de codiseño binario. Una plataforma que combina técnicas de codiseño para diseñar algoritmos combinando el DSP TMS320C40 con 2 FPGAs X4013E, es el sistema desarrollada en el LIRMM (*Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier*) [PTRC96].

Las técnicas de codiseño binario también se utilizan de forma experimental en la tarjeta HARP, pero esta vez como procesador se tiene un transputer T805 [Pag94], combinado con una FPGA Xilinx 3195.

Todas estas aproximaciones son útiles sólo para aplicaciones muy sencillas o para docencia o experimentación básica de codiseño. En cualquier caso estas tarjetas no parecen útiles para sensorizar una plataforma autónoma.

2.4.4 Futuras tendencias: *Evolvable Hardware*

El hardware que evoluciona (traducción de la expresión inglesa *Evolvable Hardware*), se refiere a un tipo particular de máquina reconfigurable cuya arquitectura, estructura y función cambia dinámicamente y autónomamente con el objetivo de mejorar su rendimiento en ciertas tareas [Yao99] [SMS99]. La combinación de los principios de los algoritmos genéticos, la computación adaptativa y las FPGAs reconfigurables están originando nuevos horizontes en el desarrollo del hardware digital.

Aunque los conceptos implicados son prometedores y las primeras investigaciones están dando resultados alentadores, este tipo de máquinas siguen siendo un proyecto de futuro. El desarrollo de aplicaciones eficientes basadas en estos principios se reducen a casos muy concretos de hardware adaptativo (filtrado digital, redes neuronales, etc.) [Hig99].

2.5 Conclusiones

La sensorización clásica con un sensor CCD puede quedar descartada porque no es necesaria una alta calidad de la imagen desde el punto de vista *humano*. Una calidad suficiente para un robot autónomo será aquella que le permita realizar con éxito las tareas encomendadas. La tecnología CCD, al ser de integración, obliga a un control complejo del sensor y a una adquisición completa de la imagen aunque sólo se pretenda analizar una parte de ella. Esto implica un tiempo adicional innecesario y memoria de almacenamiento para las imágenes.

Los recientes desarrollos de la tecnología CMOS, han mostrado que es posible realizar sensores de visión que combinen las dos propiedades interesantes para un sensor visual robótico. Por una parte suficiente calidad para realizar tareas de visión artificial y por otra parte acceso aleatorio, conseguido gracias a celdas fotosensibles de conducción.

La ventaja adicional de la tecnología CMOS de poder incorporar procesamiento interno al sensor, podría ser interesante si éste fuera reconfigurable. No es deseable una etapa de preproceso que sistemáticamente aplique un algoritmo sobre las imágenes ya que puede no ser interesante para ciertas aplicaciones. Por tanto, esta flexibilidad deberá conseguirse fuera del sensor.

Esta flexibilidad se puede conseguir mediante la utilización de una aproximación con módulos estándar combinada con software. El problema es que ninguna de estas aproximaciones reconfigurables *vía software* parece que pueda resolver el problema del procesamiento en tiempo real en una plataforma móvil.

Existen máquinas con arquitecturas orientadas a la visión artificial que consiguen

prestaciones de tiempo real, pero a costa de una gran cantidad de procesadores interconectados en buses y redes especiales de interconexión. Esto hace que un sistema completo de procesamiento de imágenes pueda llegar a ser una máquina realmente grande y costosa, cosa que no es deseable para una plataforma móvil.

El problema de la gran cantidad de hardware necesario aparece debido a la gran cantidad de datos a procesar en tiempo real. Esta restricción combinada con la necesidad de flexibilidad impone sistemas con estaciones de trabajo y tarjetas especiales basadas en DSPs. Un enfoque no posible para un módulo de sensorización visual de una plataforma móvil por cuestiones de espacio y peso. De esta manera queda descartada la utilización de módulos estándar programables vía software. La flexibilidad para implementar múltiples algoritmos de visión no se puede conseguir a través de recursos que hacen que la plataforma deje de ser autónoma.

La utilización de silicio hecho a medida queda descartada por la falta de flexibilidad y el alto coste de esta solución. La rapidez de proceso tiene el coste de la pérdida de programabilidad, con lo que de nuevo para implementar cada algoritmo, harían falta circuitos hechos a medida que sólo serian útiles en ese caso.

La reconfigurabilidad del software con la rapidez del hardware se puede conseguir con la lógica programable. El estado actual de estos dispositivos hace recomendable la utilización de una arquitectura mixta entre CPLDs y FPGAs.

Los dispositivos lógicos programables que van a implementar los algoritmos de visión artificial deben soportar una cierta complejidad. Esto descarta las CPLDs, que a pesar de sus prestaciones, tienen un número de puertas lógicas equivalentes relativamente reducido. Dentro de las familias con gran número de puertas y flip-flops, se escogerá una que permita tener contadores y punteros (generadores de direcciones de memoria), sin grandes retrasos. Esto se justificará en la definición y diseño de la arquitectura reconfigurable (secciones 4.2 y 4.3).

La tecnología de programación del dispositivo debe permitir la programación en el sistema (ISP o *In System Programability*). La reprogramación la realizará la plataforma móvil escogiendo el algoritmo de visión artificial a aplicar. Es decir, la flexibilidad del software combinada con la rapidez del hardware, lo cual obliga a la utilización de tecnología SRAM o FLASH.

La posibilidad de utilización de arquitecturas reconfigurables para procesado de imágenes existe desde principio de los 90. Haciendo una revisión del estado del arte actual de máquinas reconfigurables no se ha encontrado ninguna máquina que cumpla los requisitos de flexibilidad, modularidad, escalabilidad y pequeñas dimensiones, necesarios para una plataforma móvil autónoma.

La representación con imágenes Cartesianas hace que la aproximación reconfigurable

sea otra vez muy costosa en recursos y tiempo de proceso. En el capítulo 3 se planteará la representación log-polar que reduce la cantidad de datos a ser procesados y simplifica de forma significativa varios algoritmos interesantes para navegación robótica.

Capítulo 3

Visión foveal y navegación robótica

3.1 Visión foveal

3.1.1 La representación log-polar

El sistema de visión humano, y el de muchos otros seres vivos, tiene unas características bastante diferentes a los sistemas de visión artificial o robótica usuales. Debido a las características de los sensores visuales, generalmente CCD, y por facilidad tecnológica, la representación de la información visual suele ser Cartesiana.

A cada par de valores (x, y) se le asocia un valor $f(x, y)$ discreto que, en el caso de imágenes en blanco y negro, corresponde a un valor en una escala de grises. En realidad se suele hablar de funciones de 3 variables añadiendo la dependencia con el tiempo, obteniendo de esta manera funciones $f(x, y, t)$. En esta representación con coordenadas Cartesianas la información está distribuida uniformemente a través de la imagen, teniendo siempre la misma resolución en todas las zonas de la imagen.

Esta propiedad puede ser considerada como una ventaja, si se quieren tratar problemas con cámaras estáticas donde no se sabe a priori qué zona de la imagen va a ser interesante. También puede ser interesante cuando el formalismo Cartesiano facilite la aplicación de ciertos algoritmos de visión, cosa que ocurre cuando se pretende la extracción de información Cartesiana u ortogonal (como detección de bordes o esquinas).

Sin embargo, este método de representación de la información visual no es el que el paso del tiempo ha elegido para los seres vivos más evolucionados, incluyendo al hombre. En la figura 3.1 se muestra la representación cartesiana frente a la representación log-polar donde cada celda sería una fotocélula.

La representación log-polar tiene una distribución no uniforme de elementos foto-

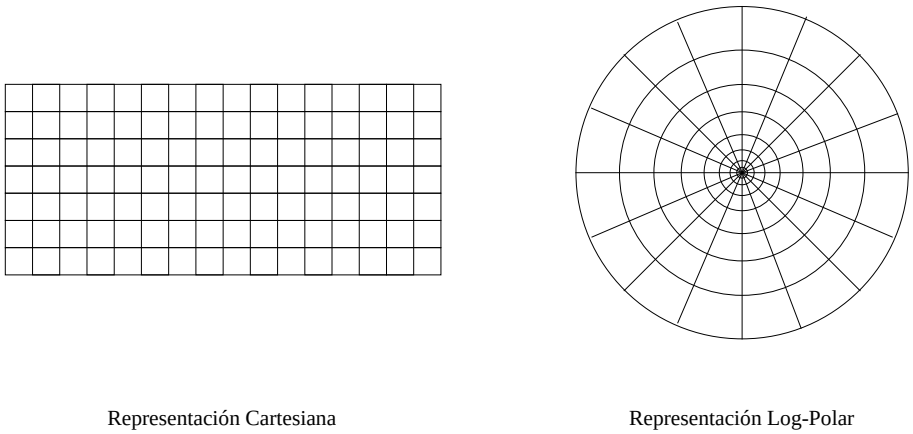


Figura 3.1: *Representación de imágenes Cartesiana y log-polar con igual número de celdas y área*

sensibles en la superficie del sensor. La concentración de células fotosensibles crece con la cercanía al centro del sensor, consiguiéndose una mayor resolución en la zona central. Simultáneamente se obtiene un amplio campo de visión en la periferia con poca resolución.

Con este razonamiento se puede observar la propiedad más interesante e importante de este método de representación de la información visual, la **reducción selectiva de información**.

Se puede observar en la figura 3.1 cómo en el centro del sensor log-polar hay una menor distancia entre las fotocélulas, éstas son más pequeñas y por tanto hay más celdas (mayor resolución). Si se quisiera tener la misma resolución que hay en el centro del sensor log-polar en toda la imagen cartesiana, la imagen tendría una cantidad total de celdas mucho mayor, por lo tanto, el tamaño de la imagen sería mucho mayor.

La figura 3.2 muestra las ecuaciones de conversión entre la representación log-polar y la cartesiana, o equivalentemente la relación entre el plano retínico y el cortical. Cabe hacer notar la singularidad del origen, es decir cuando $r = 0$ la resolución será infinita. Posteriormente en la sección en la que se presenta el sensor log-polar CMOS se comentará la solución tecnológica a este problema matemático.

Las propiedades de esta representación han sido ampliamente estudiadas [TS92], [FJPG95] encontrándose, además de la reducción selectiva de la información, las siguientes interesantes propiedades básicas:

- **Rotaciones:** Las rotaciones en el plano retínico se convierten en simples translaciones en el plano cortical, cuando el eje óptico coincide con el eje del sensor. Esta interesante propiedad ha sido explotada para realizar clasificación de patrones invariante a rotaciones [IXAM92b].

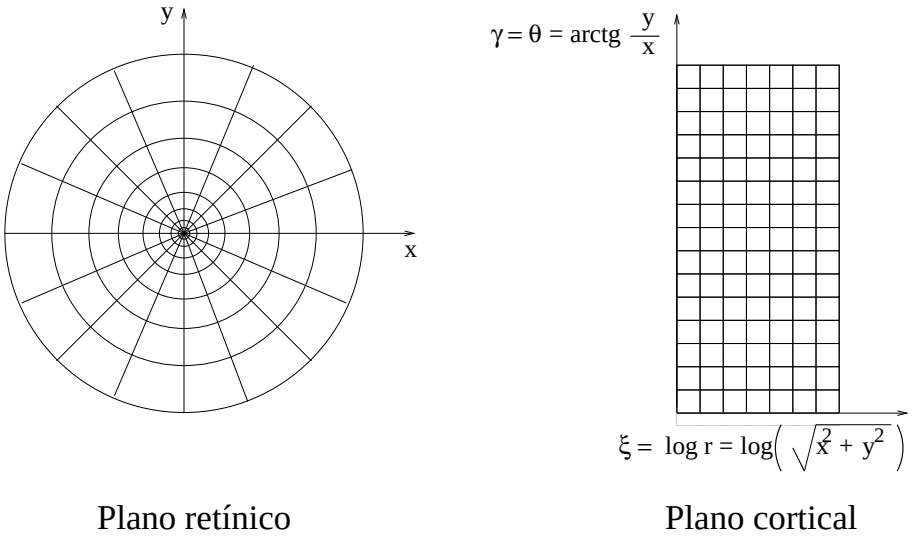


Figura 3.2: *Plano retínico y cortical: Equivalencia entre representaciones*

- **Escalados:** De nuevo, si el eje óptico coincide con el centro del sensor una operación de escalado se convierte en una simple traslación a través del plano cortical. En la sección 6 se utilizará esta propiedad para asimilar el escalado en el plano imagen debido al movimiento propio del robot.

En la sección 3.2.2 se hace una revisión de la simplificación de algunos algoritmos de visión al utilizar el sistema coordenado log-polar. Muchos de estos algoritmos son especialmente interesantes para la navegación de plataformas móviles, como son el cálculo de flujo óptico y el cálculo del tiempo al impacto.

En la sección 6 se rediseñará, usando el formalismo log-polar, un algoritmo de detección de movimiento desarrollado en coordenadas Cartesianas. Se justificará de nuevo de esta manera, la idoneidad de este sistema coordenado para la navegación de vehículos autónomos.

3.1.2 El sensor log-polar CMOS

Sensores log-polares previos

La utilidad del formalismo log-polar ha justificado, desde bastante antes de la existencia de un sensor físico, la utilización *teórica* de sensores que realizaban la transformación log-polar [WC79], [IMHM86] y [IMH+86].

El primer sensor log-polar CCD documentado fue construido en IMEC (Bélgica) fruto de la colaboración entre varios institutos de Estados Unidos y de Europa [VKC+89], [KVB+90]. Este sensor presenta algunos problemas en cuanto a continuidad de la ima-

gen por la existencia de una *zona ciega*. Esta zona con forma de cuña viene impuesta por la necesidad de extraer las cargas en la tecnología CCD, lo cual impone una discontinuidad de la imagen y una pérdida de información.

Otra discontinuidad de la imagen viene impuesta por la discontinuidad en la zona cercana al origen o **fóvea**. Esta zona no puede seguir la ley logarítmica de decrecimiento del área de los *pixels*, ya que estos deben tener un tamaño mínimo, impuesto por la tecnología de fabricación. La zona exterior del sensor que sí que sigue la ley log-polar se denomina **retina**. Estos nombres, fóvea y retina, se toman por la similitud de estos sensores de estado sólido con los sensores visuales biológicos.

La retina del sensor de IMEC tiene una resolución de 30 circunferencias concéntricas de 64 puntos cada una, lo que da una resolución de 64x30=1.920 puntos. La solución al problema de la fóvea adoptada en el sensor de IMEC fue realizar una matriz cuadrada de 102 celdas CCD en el centro del sensor. Esto resuelve el problema de la singularidad pero de nuevo plantea dificultades en la continuidad de las imágenes y, por tanto, complica la utilización de este sensor para aplicar algoritmos de visión artificial. Finalmente, la fóvea del sensor CCD de IMEC no funcionó correctamente, lo cual la convertía en una zona ciega.

A los problemas estructurales de este sensor hay que añadir las desventajas de la tecnología CCD utilizada. Se debe adquirir necesariamente toda la imagen de forma secuencial, aunque sólo se desee procesar una zona concreta. En la parte izquierda de la figura 3.3 se puede observar una micro-fotografía de este sensor.

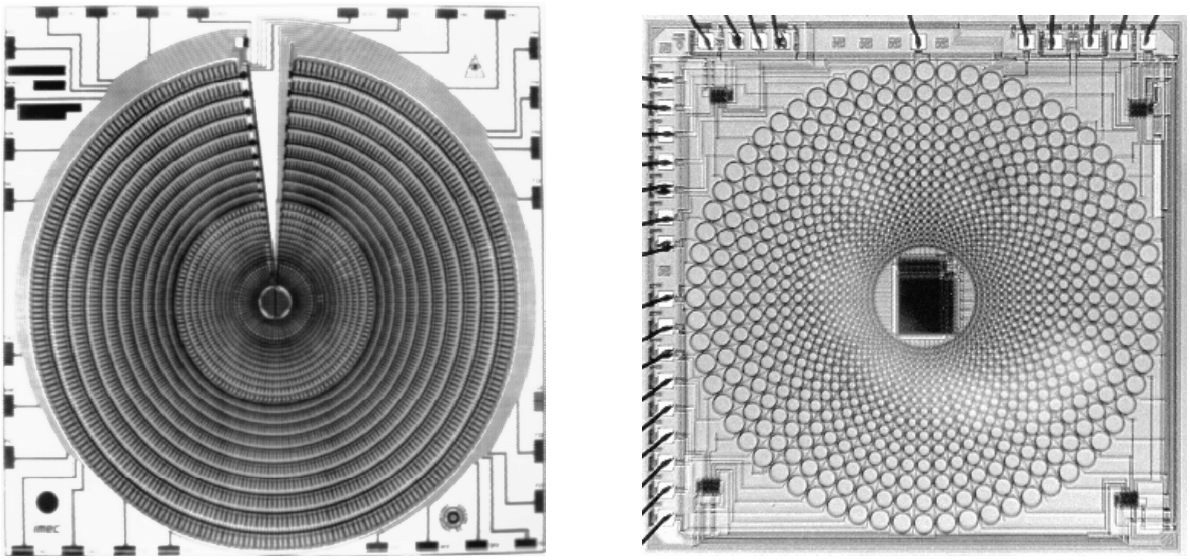


Figura 3.3: *Micro fotografías de los sensores log-polares CCD de IMEC y CMOS de la Universidad McGill*

Otro sensor log-polar, desarrollado en la Universidad McGill de Canadá con tec-

nología CMOS, se muestra también en la parte derecha de la figura 3.3 [WRL95]. La resolución en la retina de este sensor es de 16 anillos con 64 celdas cada anillo, obteniéndose de esta manera 1.024 puntos de resolución retínica. De nuevo aparece la discontinuidad en la fóvea, que es una matriz de 40 filas por 52 columnas. El problema tecnológico de las celdas demasiado pequeñas se resuelve mediante una matriz rectangular, lo que complica la utilización de algoritmos de visión artificial.

Además de los problemas comentados anteriormente, este sensor tampoco tiene acceso aleatorio porque utiliza una celda visual de integración, igual que las fotocélulas CCD.

Un problema común en ambos sensores es que en ellos la resolución no muy alta en la zona retínica, que es donde se van a explotar las ventajas del formalismo log-polar. Otro problema común a ambos sensores aparece debido al diferente tamaño de las fotocélulas. Diferente tamaño implica diferente respuesta, siendo necesario un escalado de la señal de cada celda, ralentizando de esta manera el proceso de adquisición y complicando el control del sensor.

Como respuesta a este déficit tecnológico en cuanto a sensores log-polares se diseño y construyó en IMEC (Bélgica) el sensor que a continuación se describe.

El sensor log-polar CMOS

El sensor log-polar CMOS de IMEC [Par97] resuelve muchos de los problemas que presentaban los anteriores sensores log-polares. En la tabla 3.1 se compara el sensor log-polar utilizado frente a los sensores log-polares anteriormente descritos. Las principales características del sensor log-polar son:

- **Mayor resolución:** La retina tiene una resolución de 56 anillos con 128 celdas por anillo, es decir 7.168 puntos de resolución para la retina, la mayor resolución de los sensores log-polares existentes [PBP+96a].
- **Continuidad de la fóvea con la retina:** La fóvea no es es una matriz cuadrada sino que hay continuidad en la estructura de la retina. Después del anillo más interno de la retina de 128 celdas está el anillo más externo de la fóvea con 64 celdas, justo la mitad de 128. Hay un total de 10 anillos de 64 celdas, después 5 anillos de 32, 2 de 16 celdas, 1 de 8, 1 de 4 y un *pixel* central. Esta aproximación sigue la estructura polar de la retina dando una continuidad a la imagen, útil para el procesamiento de imágenes, y una resolución de 845 puntos para la fóvea [PDS98].
- **Celda de conversión logarítmica:** La célula fotosensible que se utiliza es de conducción en vez de integración, al contrario que en los otros 2 sensores precedentes. Esto permite un acceso aleatorio a la información visual, no siendo necesaria una

Característica	CCD IMEC	CMOS McGill	CMOS IMEC
Resolución retina	1.920	1.204	7.168
Resolución fóvea	102	2080	845
Estructura fóvea	Matriz	Matriz	Polar
Celda fotosensible	Integración	Integración	Conducción

Tabla 3.1: *Comparativa entre los últimos sensores foveales desarrollados*

lectura de toda la imagen si solamente se quiere analizar una zona concreta de ésta. Adicionalmente la celda utilizada tiene una relación de conversión de la luz incidente logarítmica, lo cual minimiza el problema del escalado debido al distinto tamaño de las celdas en función de su distancia al radio [PBP+96b], [PDS97].

Estas características hacen que el sensor CMOS log-polar de IMEC sea el elegido como sensor del módulo reconfigurable. También ha sido importante el hecho de que uno de los directores del presente trabajo de investigación perteneciese al grupo de diseño del sensor en IMEC. Adicionalmente, el grupo de arquitecturas para la percepción visual, al que el doctorando pertenece, ha desarrollado hardware adicional de soporte para este sensor [BPK+96], [Bla98].

En la figura 3.4 puede observarse la estructura de la retina y de la fóvea del sensor log-polar CMOS. La retina sigue la distribución log-polar y la fóvea sigue una escala lineal, disminuyendo el número de celdas por anillo de forma gradual, para así poder mantener la continuidad de la imagen. La ecuación (3.1) muestra la equivalencia entre el plano cortical y el plano retínico para ambas zonas del sensor.

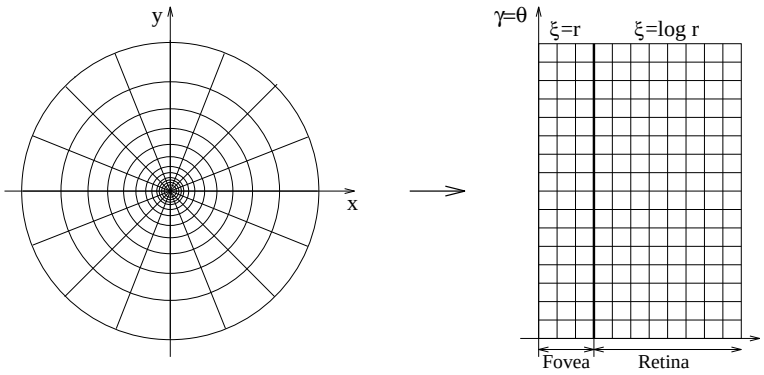


Figura 3.4: *El sensor log-polar CMOS de IMEC*

$$\begin{cases} \xi = \begin{cases} r & r \le r_o \quad \text{(fovea)} \\ \log r & r > r_o \quad \text{(retina)} \end{cases} \\ \theta = \theta \end{cases} \qquad \textit{donde} \quad \begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \arctan\left(\frac{y}{x}\right) \end{cases} \tag{3.1}$$

3.2 Navegación robótica

3.2.1 Aproximación clásica

La rápida evolución de la tecnología está incrementando la utilización de plataformas móviles en múltiples tareas, que van desde la utilización de éstas en misiones espaciales [CK97], hasta la exploración de fondos marinos [AAV97], requiriendo cada tarea específica una sensorización adecuada.

La tareas elementales que debe de realizar una plataforma móvil en un **entorno no estructurado** son:

- **Navegación y Exploración:** La plataforma debe ser capaz de moverse en su medio, adquiriendo información útil para realizar la tarea encomendada.
- **Modelado Espacial:** La información obtenida del entorno debe combinarse para realizar un modelo espacial que facilite la navegación del robot.
- **Auto-localización:** Basándose en el modelo espacial, la plataforma debe tener control sobre su posición, y así tomar decisiones que le permitan completar la tarea encomendada.

El presente trabajo de investigación se centra en conseguir un módulo de sensorización reconfigurable que sea útil para la navegación de vehículos autónomos. De una sensorización adecuada dependerá la precisión y el éxito de estas tareas.

La sensorización de una plataforma móvil es necesaria no sólo para la navegación en entornos no estructurados, donde es preciso detectar colisiones con objetos estáticos y móviles, sino también en entornos estructurados debido al inevitable error de los sistemas odométricos de medida [Bor96].

Una de las metodologías de sensorización más habituales para plataformas móviles es la utilización de sensores de ultrasonidos. La desventaja de este método de sensorización es la pobre información que se puede extraer de ellos (sólo detección de obstáculos), el rango limitado de distancia operativa, y los errores frente a cuerpos con superficies absorbentes u oblicuas [GT94].

Estas desventajas recomiendan la utilización de sensores visuales que extraerán una información más precisa del medio. Un ejemplo de una aproximación que combina ambos tipos de sensores y realizan una fusión de los datos sensoriales se puede encontrar en [IMRS94]. De nuevo aparece el problema de la gran cantidad de datos a procesar y el coste computacional de los algoritmos necesarios. Esto hace que esta aproximación tenga separada la plataforma móvil del módulo de procesamiento y control, una estación de trabajo con múltiples tarjetas de adquisición, y tarjetas de procesamiento MIMD. Es

entonces cuando aparece el problema del *cordón umbilical* o conexión entre el módulo móvil sensorial y el módulo estático de procesamiento y control, haciendo que estas plataformas no sean estrictamente hablando autónomas.

El mismo problema aparece en otros experimentos de robots guiados sólo por visión [FGS+92]. Se ha evitado el cordón umbilical en algunos casos mediante una conexión por radio a una red de *workstations*, consiguiéndose de esta manera movilidad a costa de un gran dedicación de recursos hardware [Jar97], y perdiéndose el significado estricto de la palabra *autónomo*.

Otras aproximaciones de guiado visual más novedosas han conseguido realizar una navegación realmente autónoma en entornos débilmente estructurados. En [FCS98] se presenta una plataforma móvil en la que se reducen la cantidad de datos visuales a procesar teniendo en cuenta la cantidad de información externa que representan los pixels en una imagen. De esta manera se tiene en cuenta que los *pixels* de la zona inferior de la imagen representan el suelo más cercano, y si el *pixel* está en una zona superior se corresponde con una zona más alejada y que representa una mayor área externa.

La plataforma móvil presentada en [PH94] incorpora una tarjeta con 4 transputers T800 para calcular el campo de flujo óptico. En este caso el movimiento de la plataforma debe ser conocido y constante y el escenario debe ser estático. Otro problema de esta aproximación es que la técnica utilizada para el cálculo del flujo óptico se basa en la detección y encaje de esquinas y bordes, limitando de esta manera la navegación a entornos con la presencia adecuada en la imagen de estas características. La experimentación con esta aproximación ha sido realizada en entornos interiores *preparados* de la manera adecuada, es decir en entornos semi-estructurados. La utilización de una método de cálculo de campo de flujo óptico más general, ha probado ser muy costoso computacionalmente si se quieren resultados suficientemente precisos [BB95].

Otros experimentos de navegación visual no basados en el cálculo del flujo óptico se basan también en la navegación en entornos estructurados de alguna manera. Así por ejemplo, en [MOY97] se presenta una plataforma autónoma capaz de navegar totalmente guiada por visión, aunque incorpora odometría y sensores de ultrasonidos como ayuda para la navegación. La plataforma detecta marcas especiales en árboles y en el suelo para recorrer un camino en un parque. En este caso la plataforma incorpora un sistema con múltiples CPUs una de las cuales está dedicada al procesamiento de imágenes.

Se puede observar como la aproximación *clásica* o con sensorización visual cartesiana impone equipos muy grandes para realizar la navegación en tiempo real, que habitualmente no pueden ser llevados en la plataforma móvil. La otra posibilidad para realizar navegación visual es imponer restricciones en la escena y estructurar de cierta

manera el entorno con puntos de referencia. De nuevo se plantea la necesidad de reducir los datos a procesar desde los sensores visuales y simplificar los algoritmos utilizados para la navegación. En la siguiente sección se presente el estado de la investigación en la visión log-polar y los algoritmos interesantes para la navegación basada en esta representación.

3.2.2 Visión y navegación Log-Polar

Las propiedades de la visión log-polar han sido ampliamente utilizadas en **visión activa**. El término *activo* tiene que ver con la interacción dinámica del observador con el medio. La visión activa intenta resolver el problema de *donde mirar a continuación*, intentando mantener la atención del observador sobre la zona de interés. La representación espacio variante puede ser muy útil en este problema por tener mayor cantidad de información en el centro de la imagen, donde estará la zona de interés.

La interesante propiedad de reducción selectiva de la información, combinada con la distribución radial y el crecimiento logarítmico, han sido ampliamente estudiadas en algoritmos útiles para navegación robótica.

Cálculo del flujo óptico y detección de movimiento

El flujo óptico es una poderosa aproximación para la detección de movimiento y análisis dinámico de imágenes. La aproximación clásica, desarrollada en coordenadas cartesianas, parte de la constancia de la intensidad de un punto en una imagen en un instante t y del mismo punto desplazado en un instante $t + \Delta t$. Partiendo de esta base, si las coordenadas iniciales del punto eran (x, y) y el punto desplazado en la imagen tiene unas coordenadas $(x + \Delta x, y + \Delta y)$ se puede plantear la ecuación (3.2).

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{3.2}$$

Diferenciando la ecuación y desarrollándola por series de Taylor, se obtiene la ecuación diferencial (3.3) conocida como la **ecuación de Horn**.

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{3.3}$$

La representación log-polar no presenta especiales ventajas para el cómputo del flujo óptico, en el caso más general, salvo la reducción selectiva de información. Esto se debe a que se plantea la resolución de la ecuación de Horn equivalentemente en una matriz cuadrada de puntos, que en este caso seria el plano cortical. A partir de la ecuación

(3.3) y utilizando la transformación log-polar (3.1) se puede obtener la ecuación de Horn equivalente en coordenadas log-polares (3.4).

$$\frac{\partial I}{\partial \xi}\frac{d\xi}{dt} + \frac{\partial I}{\partial \theta}\frac{d\theta}{dt} + \frac{\partial I}{\partial t} = 0 \tag{3.4}$$

En [Dan95a], [Dan95b] y [Dan96] se muestran las propiedades interesantes que puede tener el cálculo de flujo óptico para determinar parámetros relacionados con el movimiento propio del observador. De esta manera se simplifica la determinación del foco de expansión, en el caso de translación pura, y se simplifica el cómputo de la distancia de los objetos de la escena. Como conclusión se tiene que es interesante la representación log-polar, no solamente por la reducción de la información para el cálculo del flujo óptico y parámetros dinámicos relacionados con él, sino por la simplificación de los extensos cálculos en algunos algoritmos útiles para navegación de plataformas móviles.

Los métodos para la detección de movimiento se pueden agrupar en tres grandes bloques [HZM93]:

- **Cálculo del flujo óptico:** Cabe hacer notar que la ecuación (3.3) tiene dos incógnitas que corresponden a las dos componentes de la velocidad en el plano bidimensional de la imagen. La resolución de la ecuación de Horn necesita de algún supuesto adicional, que impone en definitiva limitaciones y hace que sea tan complejo este cálculo como preciso se desee hacer [BFB93], [BB95]. A partir del campo de vectores de flujo óptico se puede realizar una segmentación de la imagen y agrupar estos vectores para detectar objetos en distintas profundidades.
- **Extracción de características y encaje de puntos relevantes:** Estas técnicas se basan en dos etapas. En la primera de ellas se realiza una extracción de puntos relevantes, como esquinas, circularidades o bordes. En la segunda fase se realiza un encaje entre estas características en dos imágenes consecutivas de la misma escena, de esta manera y realizando una posterior segmentación de los puntos relevantes, se puede realizar la estimación del movimiento en objetos [Sha95]. Esta técnica es también muy costosa computacionalmente y además es menos general que la de flujo óptico. Se deben conocer las características de la escena para saber qué información a extraer va a ser relevante.
  Un ejemplo de la utilidad de estas técnicas se puede ver en [Día96] y [DAD97], donde se detectan y siguen vehículos desde cámaras estáticas en curvas de carreteras.
- **Métodos diferenciales:** Se parte de la base de que ha habido movimiento si dos imágenes consecutivas de una misma escena no son iguales. De esta manera se puede realizar una diferenciación y detectar los cambios. La ventaja de los algoritmos diferenciales es claramente la simplicidad de los cálculos.

La arquitectura del módulo reconfigurable estará orientada, pero no limitada, a la realización de algoritmos diferenciales de forma segmentada, tal como se justificará en el capítulo 4.2.

Como parte del presente trabajo de investigación se ha desarrollado en el sistema log-polar, un algoritmo de detección de movimiento propio independiente del movimiento de la cámara pero con ciertas restricciones. Un análisis inicial sólo para la retina puede encontrarse en [BDPP97b] y el trabajo completo, desarrollado tanto para la fóvea como para la retina, se puede encontrar en [BDPP97a]. En el capítulo 6 se realiza el diseño del algoritmo diferencial bajo la arquitectura reconfigurable propuesta.

Cálculo del tiempo al impacto

La utilización del formalismo log-polar simplifica el cálculo del tiempo al impacto, algoritmo especialmente interesante para navegación robótica. En la figura 3.5 puede observarse de forma simplificada la relación entre el tamaño real de un objeto r', la distancia focal f, la distancia del objeto hasta la focal z, y el tamaño del objeto en el plano imagen r.

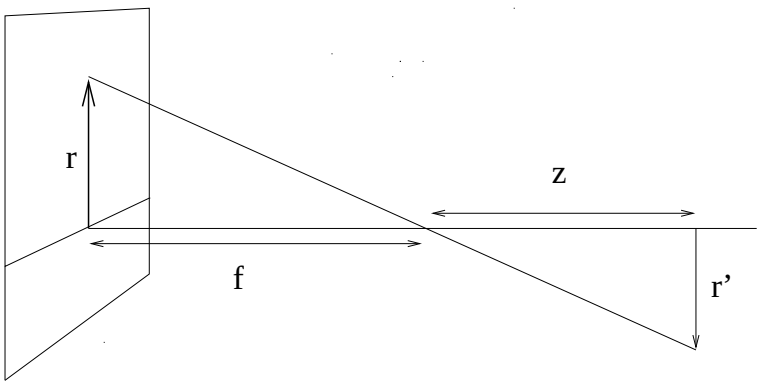


Figura 3.5: *Crecimiento de un objeto en el plano imagen*

Este tamaño viene expresado por la ecuación (3.5).

$$\frac{f}{r(t)} = \frac{z(t)}{r'} \tag{3.5}$$

Derivando respecto del tiempo se obtiene para los valores absolutos:

$$V(t)\frac{f}{r^2(t)} = \frac{W(t)}{r'} \tag{3.6}$$

Donde $V(t)$ es la velocidad radial en el plano de la imagen y $W(t)$ es la velocidad de aproximación del objeto al sensor.

Combinando ambas ecuaciones se puede obtener la relación entre la distancia del objeto al foco y su velocidad que se conoce como **tiempo al impacto** representada en la ecuación (3.7) como τ.

$$\tau = \frac{z(t)}{W(t)} = \frac{r(t)}{V(t)} \tag{3.7}$$

Realizar los cálculos de la ecuación (3.7) en el plano cartesiano no es sencillo. Se debe calcular la velocidad de cada punto en el plano imagen, (es decir el flujo óptico) cuya determinación es tan costosa como precisa. Además aparece la dependencia con el radio, lo que implica la aparición de raíces cuadradas y operaciones costosas temporalmente.

Si se utiliza la ecuación de transformación log-polar (3.1) y la ecuación del flujo óptico en coordenadas log-polares (3.4) se obtendrá el tiempo al impacto en coordenadas log-polares. Para la retina se expresa en la ecuación (3.8) y para la fóvea en la ecuación (3.9).

$$\tau = \frac{-\frac{\partial I}{\partial \xi}}{B\frac{\partial I}{\partial t}} \tag{3.8}$$

$$\tau = \frac{-\xi\frac{\partial I}{\partial \xi}}{\frac{\partial I}{\partial t}} \tag{3.9}$$

Puede observarse en ambas ecuaciones cómo se simplifica el cálculo del tiempo al impacto. En el caso de la retina basta con calcular el gradiente en la dirección radial y realizar el cociente con la derivada temporal. Es decir, sólo mediante la diferenciación espacial y temporal se puede realizar el cálculo del tiempo al impacto, tarea muy costosa computacionalmente en coordenadas cartesianas. La constante B es el coeficiente de crecimiento del sensor log-polar y depende de la geometría del sensor. En [TS91b] y [TS91a] se puede encontrar un rigurosa justificación de las ventajas del formalismo log-polar para el cálculo del tiempo al impacto.

Visión binocular y seguimiento de objetos

Dentro de la *visión activa* aparece la necesidad de redireccionar el centro de la imagen hacia la zona de interés. Esta tarea realizada en cabezas de robots móviles ha mostrado ser especialmente sencilla utilizando visión log-polar. En [BSV96a], [BSV96b] y en [CPS97] se muestran algoritmos de control de convergencia y correlación con visión

log-polar. Adicionalmente, es posible el planteamiento de tareas más complejas como son la detección de formas y su posterior seguimiento con visión log-polar [Jur99].

Existe una relación de utilidad mutua entre la representación log-polar y el seguimiento de objetos. La representación polar hace que sea más sencillo buscar el centro de atención y redirigir la cámara móvil de un robot. De la misma manera la posibilidad de mover la cámara y buscar el centro de atención hace que la representación log-polar sea más útil porque se puede tener la máxima resolución en la zona de interés.

Esta interesante relación de la visión log-polar con la visión activa hace de nuevo recomendable la utilización de la visión log-polar como etapa de sensorización del módulo reconfigurable.

3.3 Conclusiones

La representación log-polar tiene diversas ventajas muy interesantes para la navegación robótica. La más importante sin duda es la reducción selectiva de la información.

La distribución log-polar proporciona mayor resolución en la zona a analizar manteniendo un campo de visión suficiente para apercibirnos de los cambios. Este es el modo de sensorización con el que la naturaleza ha dotado a los seres vivos más evolucionados. Esta reducción de información implica una disminución del tiempo de proceso y, por tanto, un menor tiempo de respuesta.

Además de la reducción selectiva de información se tiene una simplificación significativa en varios algoritmos de visión interesantes para navegación robótica. Adicionalmente, si la cámara está montada sobre una cabeza móvil, y se puede orientar hacia la dirección del movimiento, se elimina la componente polar, quedando sólo la dependencia radial del flujo óptico.

La distribución log-polar hace que se simplifique también el cálculo del tiempo al impacto cuando, de nuevo, coincide el foco de expansión con el centro del sensor. Esta simplificación evita la realización de costosos cálculos y elimina el cálculo indirecto del flujo óptico en cada punto.

Como parte del presente trabajo de investigación, se ha realizado el estudio de diversos algoritmos que serían interesantes que implementase el módulo reconfigurable. Las conclusiones preliminares muestran que los algoritmos diferenciales tienen un bajo coste computacional. De manera adicional, muchos de los algoritmos útiles para navegación robótica que se simplifican quedan reducidos a cálculos de derivadas espaciales y temporales.

Los sistemas que utilizan guiado visual, o bien navegan en entornos semi-estructurados

(marcas o limitaciones en el movimiento), o bien el equipo para procesar los datos visuales no reside localmente en el módulo que navega. Es decir, hay una conexión por radio o cable desde la plataforma móvil hasta los módulos de proceso y control. Esto se debe a que la mayoría de veces la plataforma móvil no puede transportar los recursos hardware necesarios para realizar la navegación visual robótica en tiempo real. El tamaño de estos equipos limita la movilidad de las plataformas autónomas.

La simplificación de los algoritmos interesantes para navegación, debido a la utilización de coordenadas log-polares, reduce la complejidad de las operaciones a realizar. La reducción selectiva de la información a procesar también se hace interesante para mejorar el tiempo de respuesta del sistema. El dotar a una plataforma **autónoma** (estrictamente hablando) de un sistema de guiado visual basado en cámaras CCD no es posible en entornos no estructurados. La complejidad de los algoritmos, añadida a la cantidad de información a procesar, requieren unos equipos que pueden ser demasiado voluminosos para una plataforma móvil. Los ejemplos que se han encontrado en la bibliografía de guiado visual de plataformas móviles, son o bien en entornos semiestructurados, (lo que reduce la complejidad de los algoritmos de navegación, o bien la plataforma móvil simplemente contiene los sensores visuales y el procesado se realiza en una estación remota, conectada a la plataforma móvil por cable o radio.

Otro hecho decisivo para la elección del sistema coordenado log-polar ha sido la existencia de un sensor CMOS útil para implementaciones reales. Este sensor, basado en celdas de conversión, permite además un acceso aleatorio a cualquier zona de la imagen.

La combinación de la velocidad del hardware y la programabilidad del software, (conseguida con la lógica programable), con la reducción del tamaño de las imágenes y la simplificación de los algoritmos de navegación, (conseguida con el sensor log-polar), van a ser el punto de partida para el diseño del módulo de sensorización reconfigurable.

Parte II

Diseño y programación del módulo reconfigurable

Capítulo 4

Diseño del módulo reconfigurable

4.1 Introducción

En el capítulo 1 se ha descrito el problema que se quiere abordar: el diseño de un módulo de sensorización visual útil para navegación robótica. Posteriormente, en los capítulos 2 y 3, se han revisado las arquitecturas y estrategias existentes para la navegación con sensorización visual. Las aproximaciones estudiadas han mostrado ser mejorables para la navegación de vehículos autónomos. La gran cantidad de recursos hardware, necesarios para procesar en tiempo real gran cantidad de datos visuales, limita la movilidad de la plataforma autónoma.

Se ha presentado, de la misma manera, la visión log-polar como una solución parcial al problema de la gran cantidad de información que debe ser procesada. Así mismo se ha justificado la utilización de la visión log-polar por la simplificación de varios algoritmos útiles para la navegación de plataformas móviles.

Desde otro punto de vista se ha planteado la necesidad de obtener una velocidad *hardware* para cumplir restricciones de tiempo real, pero simultáneamente es deseable tener la flexibilidad *software* para cubrir distintos algoritmos útiles para navegación robótica. Estas ideas han orientado la investigación hacia la utilización de lógica reconfigurable para conseguir ambos objetivos. En esta línea se han revisado los diversos dispositivos lógicos programables y las máquinas reconfigurables existentes en la actualidad, sin encontrar una máquina que se ajuste a las necesidades del módulo reconfigurable.

En este capítulo se van a plantear los requisitos que debe cumplir el módulo reconfigurable y, en función de ellos, las soluciones propuestas para dicho módulo. Con estas soluciones se afrontará el diseño de la arquitectura propuesta. Adicionalmente se propondrá la metodología a seguir para diseñar un algoritmo de visión artificial en el

módulo reconfigurable.

4.2 Objetivos del módulo reconfigurable

4.2.1 Requisitos tecnológicos

En primer lugar para conseguir el requisito de la reconfigurabilidad del hardware, y la flexibilidad del software, se utilizan dispositivos lógicos programables. De las diversos dispositivos existentes en la actualidad se seleccionan los que:

- **Permitan la reconfigurabilidad en el sistema.** Este requisito es conocido como **ISP** (*In System Programmability*). Esto implica tecnología SRAM o FLASH.
- **Permitan la implementación de algoritmos complejos.** Este requisito elimina a las CPLDs complejas y selecciona las FPGAs.
- **Tengan prestaciones de alta velocidad.** La utilización de FPGAs clásicas impondrá grandes retrasos cuando el sistema, y por tanto el conexionado, sea muy complejo. Esto impone la selección de una familia de FPGAs de arquitectura híbrida.

Entre las disponibles en el mercado, al inicio de la implementación física de este trabajo de investigación, la familia FLEX8000 de Altera cumple los requisitos tecnológicos anteriormente descritos.

4.2.2 Requisitos a nivel de sistema

Es también un requisito para el módulo reconfigurable que cubra un amplio margen de algoritmos de diferente complejidad, incluso algoritmos de visión artificial que no han sido previstos inicialmente en el módulo. Es por tanto aconsejable una **arquitectura escalable**, pero teniendo en cuenta las restricciones de espacio, peso, bajo coste y consumo deseables para una una plataforma móvil.

Adicionalmente, con el objetivo de conseguir alta velocidad de proceso (decenas de imágenes por segundo sin pérdida de información relevante), se utilizan imágenes log-polares para reducir la cantidad de datos a procesar, y se utiliza el paralelismo inherente en los algoritmos de visión artificial [Pit93]. La arquitectura debe incorporar **paralelismo temporal** y **paralelismo espacial**. Esto implica dividir los algoritmos en etapas que puedan formar un **cauce segmentado de datos**, permitiendo simultáneamente la paralelización en cada etapa. El diseño de una red de elementos de proceso escalable con una topología determinada implicaría un alto coste, no deseable para el módulo de sensorización reconfigurable. Se asume entonces la realización de un cauce segmentado

que implementa los algoritmos de visión artificial en el módulo reconfigurable.

La combinación de la división de cada algoritmo en etapas, y la escalabilidad de la arquitectura sugieren una **arquitectura modular**. Cada etapa de proceso del módulo reconfigurable, que se denomina **Elemento de proceso** (**EP** en adelante), realiza una etapa del algoritmo segmentado. Como el algoritmo que implementa el módulo reconfigurable va a cambiar, el número de etapas del cauce segmentado puede cambiar, con lo que el número de etapas es variable y, además cada etapa o EP debe ser totalmente reconfigurable.

A estas condiciones hay que añadir que, cada EP debe tener externamente la misma interfase de conexión con los EPs vecinos del cauce (EP anterior y posterior). Así se evita tener que diseñar un EP específico para cada etapa. De esta manera, se realiza una sola interfase externa de intercambio de datos, permaneciendo la reconfigurabilidad dentro de la FPGA de cada EP. Es decir, cada EP es totalmente reconfigurable manteniendo la misma interfaz externa. La reconfigurabilidad se realiza incorporando la funcionalidad del EP en una FPGA. La interfaz externa de transmisión de datos entre EPs, (las diversas etapas del cauce segmentado), se consiguen definiendo:

- Un protocolo de intercambio de datos entre EPs independiente del algoritmo y de la etapa.
- Una sola asignación de pines para el módulo que incorpora la funcionalidad (la FPGA).
- Un solo pequeño circuito impreso que implemente un EP y cuya funcionalidad se define programando la FPGA con el código adecuado.

Con estos principios y diseñando un solo circuito impreso que incorpore la funcionalidad de un EP, añadiendo más EPs se consigue aumentar la complejidad de los algoritmos y, por tanto, la profundidad de la segmentación. Estos EPs son idénticos externamente estando su funcionalidad definida con la programación de las FPGAs.

Es deseable que el EP tenga una sola interfaz externa que permita el diseño de cada etapa independientemente del resto. Lo único a tener en cuenta debe ser el tipo de entradas del EP, (que vienen del EP anterior), la tarea a realizar con estos datos, y la salida del EP, (que es la entrada del EP siguiente). El protocolo de intercambio de datos debe ser el mismo para todos los EPs y está íntimamente relacionado con la interfase externa del EP.

4.2.3 Requisitos de los algoritmos de visión

Un requisito del módulo reconfigurable es que sea capaz de operar como una función cuyas entradas sean imágenes, y cuya salida sea una estructura compleja de datos. Un caso particular de estructura compleja de datos puede ser una imagen log-polar procesada, pero la salida podría ser un solo byte expresando un valor medio, o un conjunto de bytes como coordenadas de objetos detectados. De esta manera, el módulo reconfigurable debe ser capaz de obtener imágenes de un módulo de adquisición de imágenes y suministrar al robot el resultado del análisis realizado.

El algoritmo completo de visión puede no estar completamente implementado en el módulo reconfigurable de sensorización visual, y estar dividido en dos etapas: Una hardware (implementada en el cauce segmentado), y una software que realizará el procesador del sistema. En el capítulo 5 se analiza esta interacción hardware-software.

Globalmente el módulo reconfigurable tiene como entrada imágenes log-polares, y como salida unos datos estructurados. Para diseñar cada etapa del algoritmo hay que tener en cuenta los recursos hardware disponibles en cada EP, el tipo de datos de entrada que provienen del EP anterior, y la salida que es la entrada del EP siguiente.

A las señales necesarias para implementar una transmisión de bytes independiente del algoritmo, hay que añadir señales para indicar que se ha finalizado la transmisión de una estructura de datos. La salida de un EP puede ser una secuencia de bytes de longitud no determinada *a priori*. De esta manera, un EP puede transmitir o bien imágenes completas, o bien una simple colección de bytes.

Finalmente cabe hacer notar que el balance de entrada/salida del cauce no tiene por que ser siempre de uno a uno. Es decir, por cada imagen que entre en el cauce el resultado no tiene porque salir una imagen o estructura de datos. Para obtener cada resultado se pueden consumir varias imágenes log-polares, con una latencia no despreciable.

En la sección 3.2.2 se agrupan los diversos algoritmos de estimación de movimiento en tres grandes bloques: Cálculo de flujo óptico, extracción y encaje de puntos relevantes y algoritmos diferenciales. De estos tres distintos enfoques, se ha razonado en 3.2.2, que los algoritmos diferenciales son los de implementación más sencilla y funcionamiento más rápido.

Para implementar algoritmos diferenciales en el tiempo es necesario el almacenamiento de imágenes de distintos instantes en las etapas. De esta manera, una etapa del cauce que calcule la primera derivada temporal debe tener acceso a la imagen grabada en los instantes $t-1$ y t simultáneamente. Esto implica la utilización de memoria de almacenamiento de imágenes parciales en un EP accesible por el siguiente EP. Al

Requisitos del módulo reconfigurable	Soluciones propuestas
Flexibilidad del software y velocidad del hardware	Lógica programable
Reconfigurabilidad en el sistema	Tecnología SRAM
Complejidad de los circuitos + prestaciones	FPGAs híbridas
Complejidad de los algoritmos	Arquitectura escalable
Alta velocidad de proceso	Reducción de datos log-polar
	Paralelismo temporal
	Paralelismo espacial
Facilidad de diseño de los algoritmos	Interfase única para los EPs
	Protocolo único
Algoritmos diferenciales temporales	Memoria entre los EPs
Algoritmos diferenciales espaciales	Memoria local a cada EP

Tabla 4.1: *Requisitos y soluciones propuestas al módulo reconfigurable*

utilizar imágenes log-polares, cada imagen ocupa sólo 9.728 bytes, lo que hace posible almacenar una imagen entera en un circuito de memoria de 16 Kbytes, quedando incluso espacio de memoria libre.

La diferenciación espacial también requiere de una memoria local a cada EP, para almacenar los datos de filas o columnas con los que se va a operar. También esta memoria local se utiliza para almacenar resultados intermedios del EP que no se puedan almacenar en la FPGA. De nuevo, el pequeño tamaño de las imágenes log-polares, permite reducir la necesidad de memoria local del EP a un solo circuito integrado.

En la tabla 4.1 se muestra un resumen de los objetivos o requisitos para el módulo reconfigurable y las soluciones propuestas. En las siguientes secciones se describirá el cauce segmentado en EPs, la estructura de los EPs y el flujo de datos entre ellos.

4.3 El cauce segmentado

En el módulo reconfigurable se implementa un cauce segmentado (paralelismo temporal), donde cada etapa del cauce es un EP que puede implementar paralelismo espacial [BPP98]. Para diseñar la interfase externa de los EPs hay primero que definir el protocolo de flujo de datos dentro del cauce segmentado.

4.3.1 Flujo de datos

En un cauce segmentado la velocidad del flujo de datos viene determinada por la etapa más lenta del cauce [HP90]. En el caso del módulo reconfigurable cada etapa se implementa en un EP, con lo cual el EP cuya tarea sea más costosa ralentiza el flujo de datos a través del cauce. Por tanto la división de un algoritmo en etapas debe ser equilibrada en coste temporal para evitar cuellos de botella.

Un algoritmo de visión artificial se divide en varias etapas, en función de su complejidad. De esta manera varios algoritmos pueden tener varias etapas y las etapas, implementadas cada una en un EP, pueden tener diversos costes temporales. Además, estos costes temporales pueden ser variables en función de los datos suministrados y el resultado de operar con estos. Como conclusión, se tiene que la temporización del intercambio de datos no puede estar fijada, sino que se debe adaptar a la complejidad de las operaciones de cada EP y la dependencia temporal con el resultado de estas operaciones.

La principal idea va a ser definir un protocolo de intercambio de datos, y por tanto unas señales, que sean independientes de un algoritmo concreto. Un requisito básico de los EPs es que el tiempo de proceso en cada etapa puede no estar fijado a priori y depender de los datos que se estén procesando. Esto sugiere un protocolo asíncrono de intercambio de datos con señales de envío / reconocimiento de datos. Por otra el diseño de máquinas síncronas dentro de las FPGAs de cada EP tiene ventajas en cuanto a facilidad de diseño, comprobación y automatización en el proceso de diseño. En [Boe95] se puede encontrar un análisis detallado de las ventajas y desventajas de circuitos síncronos *versus* autotemporizados. En el caso del EP se ha optado por realizar un cauce segmentado síncrono, con un protocolo de intercambio de datos con señales de envío / reconocimiento que también evolucionan síncronamente.

Cada EP evoluciona síncronamente con el reloj global del sistema. Idealmente, para que no haya una etapa que ralentice a las demás y evitar cuellos de botella, las diversas etapas de un algoritmo deben tener el mismo tiempo de proceso. Esto no es siempre posible debido a la diferente complejidad de cada etapa y los recursos necesarios para implementarla. La figura 4.1 muestra el flujo de datos a través de la arquitectura segmentada que se realiza con la máquina básica de estados que se muestra en la figura 4.3.

La solución a estos requisitos de flexibilidad es utilizar un protocolo simple de intercambio de datos basado en señales de envío/reconocimiento de dato. En el cronograma que aparece en la figura 4.2 se muestra la comunicación del elemento de proceso i-ésimo con la etapa anterior y posterior. En la figura, el EP $i+1$ es el cuello de botella en el cauce de datos ya que a pesar de tener el EP i-ésimo el dato listo, el EP siguiente en el

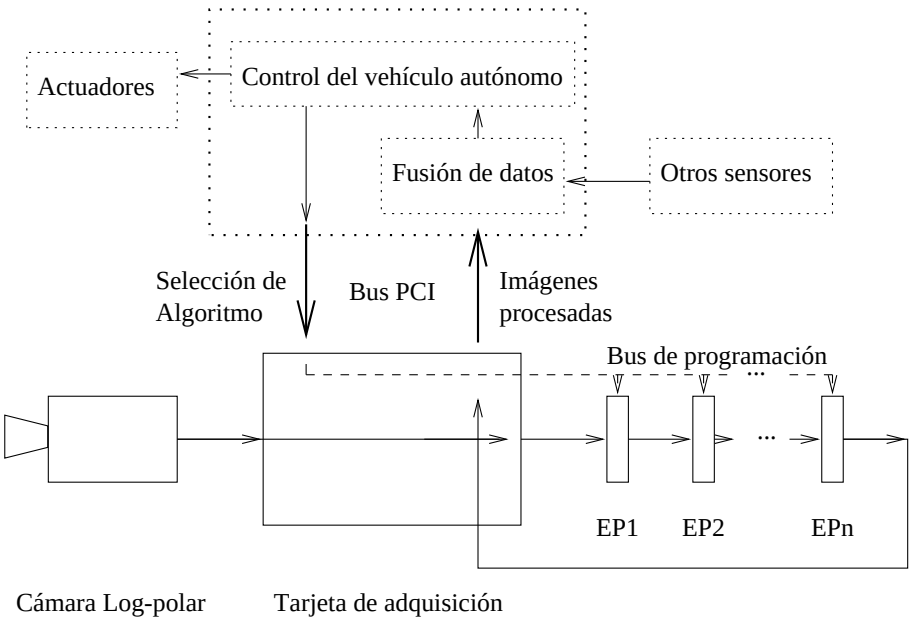


Figura 4.1: *Flujo de datos a través del cauce segmentado*

cauce no lo acepta hasta procesar y entregar el dato anterior.

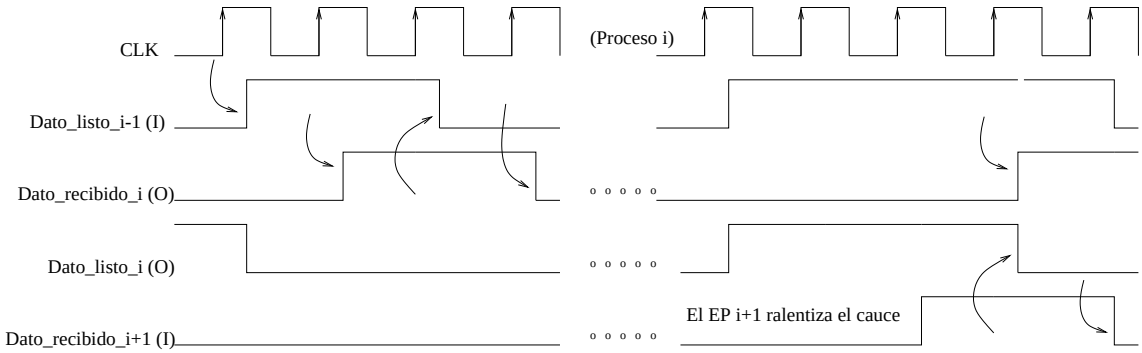


Figura 4.2: *Transmisión de datos a través del cauce segmentado*

Para indicar el fin de una estructura de datos, que puede ser una imagen, y el inicio de la siguiente, el EP i-ésimo tiene una señal de entrada img_i-1 y una señal de salida img_i. Así por ejemplo, si img_i-1 vale 0 y en la siguiente validación de dato con $Dato\_listo\_i-1$ pasa a valer 1, esto indica que este nuevo dato es el primero de una nueva estructura de datos. El EP i-ésimo empieza a procesar esta nueva estructura de datos (o imagen) que como resultado dará una nueva estructura de datos (o imagen). Cuando se empiece a transmitir el primer byte de esta nueva estructura, se validará de nuevo con la señal $Dato\_listo\_i$ a la vez que cambiará el valor de la señal img_i

Para la adquisición de las imágenes y transmisión de éstas al primer EP, se utiliza una tarjeta de adquisición de imágenes para bus PCI [Bla98]. Dicha tarjeta ha sido

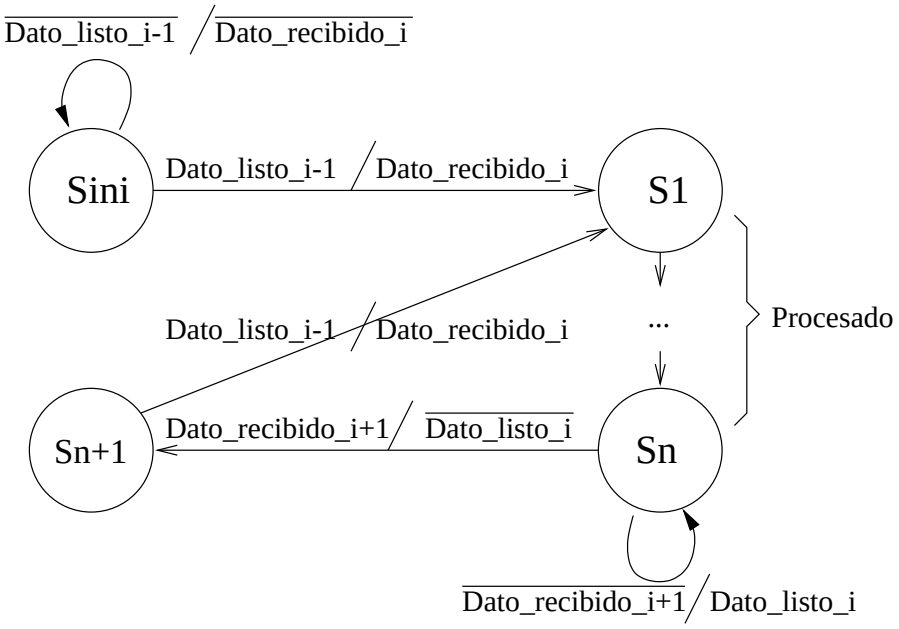


Figura 4.3: *Máquina de estados para el intercambio de datos*

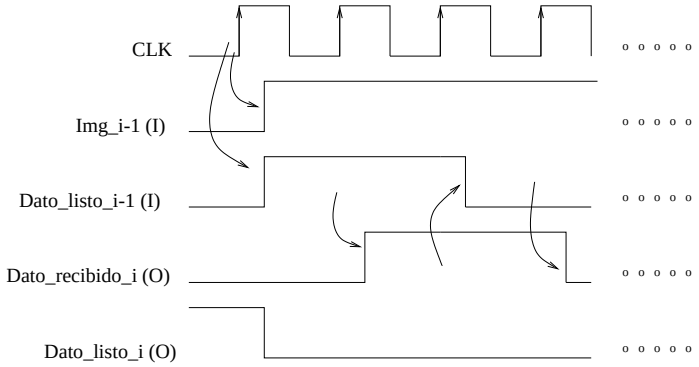


Figura 4.4: *Transmisión de datos con cambio de estructura de datos*

desarrollada en paralelo al módulo reconfigurable y tiene, además de la funcionalidad normal de una tarjeta de adquisición de imágenes, un conector con la misma interfase que los EPs para el flujo de datos. Finalmente la tarjeta también dispone de un conector por donde se reciben los datos resultado del procesado a través del cauce. La tarjeta de adquisición de imágenes se encarga de almacenar las imágenes o datos resultado en una zona de memoria accesible por el control de la plataforma móvil. El EP i-ésimo tiene la señales de intercambio de datos que se indican en la tabla 4.2.

Un esquema del funcionamiento de una máquina de estados, que implementa este protocolo de envío / reconocimiento de datos para el EP i-ésimo, puede observarse en el grafo de la figura 4.3. El funcionamiento de esta máquina viene descrito en los pasos:

- El EP i-ésimo espera que el EP anterior tenga un dato válido para procesar.

Nombre	Tipo	Comentarios
Dato_listo_i	Salida	Dato válido en la salida del EP i-ésimo
Dato_recibido_i	Salida	Dato capturado en la entrada del EP i-ésimo
img_i	Salida	Indica cambio de estructura de salida
Dato_listo_i-1	Entrada	Dato válido en la entrada del EP i-ésimo
Dato_recibido_i+1	Entrada	Dato capturado en la entrada del EP i+1
img_i-1	Entrada	Indica cambio de estructura de entrada

Tabla 4.2: *Señales para intercambio de datos del EP i-ésimo*

- El EP $i-1$ ya ha terminado, tiene un dato listo, y activa la señal $Dato\_listo\_i-1$.
- El EP i-ésimo lo captura y advierte al EP $i-1$ que ya lo ha capturado activando $Dato\_recibido\_i$
- El EP i-ésimo procesa este dato junto con la información almacenada de otras imágenes previas y la imagen actual.
- Cuando el EP i-ésimo termina de procesar este dato lo pone en su salida de datos, que es la entrada del EP $i+1$, y la valida activando la señal $Dato\_listo\_i$.
- Se mantiene el dato en el bus hasta que el EP $i+1$ lo ha capturado, indicándolo activando la señal $Dato\_recibido\_i+1$. Una vez ha ocurrido esto se vuelve al principio del ciclo.

En el caso en que sea preciso realizar una petición de más de un byte al EP $i-1$ simplemente se repite el proceso de espera de dato hasta tener los datos necesarios. Pueden haber casos en los cuales sea más eficiente transmitir sin el protocolo de envío / reconocimiento un grupo de bytes ya procesados entre etapas del cauce, y una vez residan completamente en el EP, iniciar el procesado. Las señales de la tabla 4.2 también son útiles en este caso. Se diseñará el EP i-ésimo de manera que cuando se active la señal $Dato\_listo\_i-1$ se reciban de forma síncrona el conjunto de bytes desde el EP $i-1$. Esta imagen se almacena en memoria local, pudiéndose acceder de forma completa a toda la imagen.

4.3.2 El elemento de proceso

A partir del protocolo que marca el flujo de datos en la arquitectura segmentada, y teniendo en cuenta la orientación (que no limitación) de la arquitectura a algoritmos diferenciales, es posible definir los requisitos del elemento de proceso.

- Los *pixels* de las imágenes log-polares están codificados en una escala de grises de 0 a 255. El tamaño del bus de datos será por tanto de 8 bits.

- El EP tiene una FPGA que evolucionará síncronamente con el reloj global del sistema.
- El EP tiene una memoria local para poder almacenar una imagen log-polar completa y los resultados intermedios de las computaciones.
- Para acelerar el cálculo de derivadas temporales, cada EP tiene una memoria de almacenamiento intermedio accesible por el EP siguiente. En esta memoria se almacena una imagen log-polar completa.
- Para evitar conflictos de lectura y escritura simultáneas en el mismo banco, se tienen 2 bancos de memoria. En uno la FPGA del EP i-ésimo está escribiendo la imagen actual (instante t) que le está suministrando al EP $i+1$. En el otro banco está la imagen que se escribió en el instante $t-1$ y que puede leer simultáneamente el EP $i+1$.
- Para evitar contenciones de bus y, teniendo en cuenta que los datos fluyen en una sola dirección, las memorias de almacenamiento de imágenes intermedias son de doble puerto. En el puerto de la izquierda escribe la FPGA del EP i-ésimo y en el de la derecha lee el EP $i+1$.
- Para facilitar la programación de la FPGA del EP hay un bus de programación común a todos los EPs. A cada EP se le asigna una dirección física mediante interruptores que una pequeña PAL decodifica.

Para posibilitar que el EP i-ésimo acceda a la imagen almacenada en la memoria de doble puerto del EP $i-1$, genera la dirección del dato al que quiere acceder mediante la señal $direccion\_izda\_i[13..0]$. Este bus de direcciones es compartido por ambas memorias. El EP selecciona la memoria de la que se quiere leer activando la señal de habilitación de salida $OE\_izda\_i[1..0]$. Para evitar que simultáneamente ambas memorias vuelquen sus datos en el bus, provocando un cortocircuito, se ha tenido el cuidado de diseñar estas dos señales de habilitación de salida siendo una complementaria de la otra. De esta manera, incluso cuando se esté realizando un *reset*, siempre una salida de las dos memorias está en modo triestado. En estas memorias, el puerto de la derecha está siempre seleccionado en modo lectura, ya que en ese puerto sólo se va a leer.

Análogamente, en un EP, cuando la FPGA quiera escribir en su memoria de doble puerto, debe generar la dirección de escritura en el puerto de la izquierda. Activando la señal interna al EP CE adecuada se selecciona la memoria donde se va a escribir. En este puerto solamente se va a escribir, por lo que está siempre seleccionado en modo escritura.

En la figura 4.5 se puede observar la estructura interna del EP i-ésimo. Se ha utilizado una FPGA de la familia FLEX 8000 de Altera con 820 registros. Esta familia cum-

ple con los requisitos de arquitectura híbrida y programación en el sistema (tecnología SRAM). Para realizar la programación se ha implementado un bus de programación con todas las señales en paralelo a todas las FPGAs. El proceso de programación se describe en la sección 4.3.3.

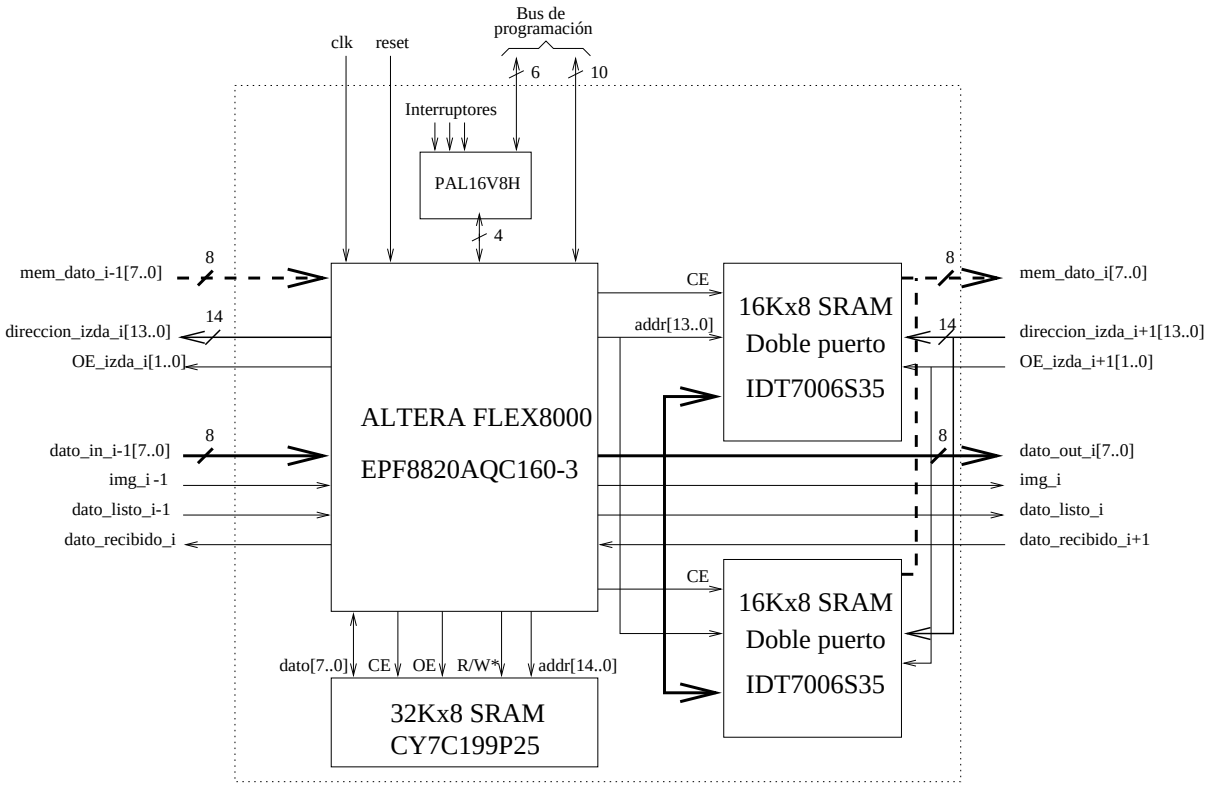


Figura 4.5: *Elemento de proceso (EP) i-ésimo*

Las memorias de doble puerto son de 16 Kbytes. De esta manera cabe una imagen log-polar completa en cada una de ellas. El puerto de la izquierda de las memorias de doble puerto está siempre en modo escritura con la señal $R/\overline{W}$ siempre a nivel bajo. El bus de direcciones que entra en el puerto de la izquierda lo comparten ambas memorias. Mediante un flanco ascendente en la señal CE se graba el dato de salida en la memoria adecuada. De esta manera, el EP siguiente tiene acceso a la imagen presente que le esté suministrando el EP previo del cauce, y a la imagen anterior almacenada en este mismo EP. Con esta sencillo cauce de datos se puede acceder en paralelo a la imagen del instante t y a la imagen del instante $t-1$, facilitando el computo de diferencias temporales.

Para poder realizar este acceso, el puerto de la derecha de las memorias de doble puerto está siempre en modo lectura. El EP puede acceder a la memoria de doble puerto del EP de su izquierda para leer una imagen generando la dirección de 14 bits y activando la señal OE adecuada.

Esquema de configuración	Acrónimo	Fuente del programa
Serie activo	AS	EPROM de configuración serie de Altera
Paralelo activo	AP	EPROM paralela
Serie pasivo	PS	Fuente de datos serie
Paralelo pasivo síncrono	PPS	Controlador inteligente
Paralelo pasivo asíncrono	PPA	Controlador inteligente

Tabla 4.3: *Esquemas de configuración de la familia FLEX8000*

La memoria local es de 32 Kbytes, siendo necesario el control de todas sus señales por parte de la FPGA para poder realizar accesos de lectura y escritura de datos intermedios.

Con este esquema se ha realizado un pequeño PCB que implementa un EP. El circuito impreso, para el que se han utilizado 2 planos de alimentación y 2 capas de trazado, se muestra a tamaño real en la figura 4.6.

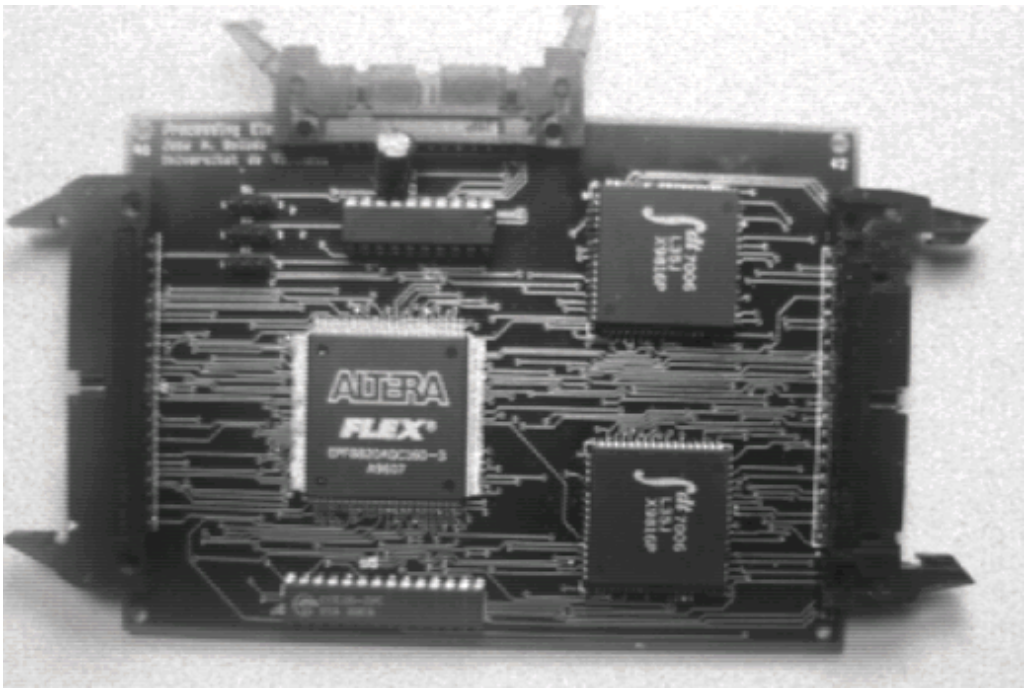


Figura 4.6: *Circuito impreso del elemento de proceso*

4.3.3 Programación de las FPGAs

Hay diversos métodos de programación de las FPGAs de la familia FLEX8000 de Altera que se resumen en la tabla 4.3

El requisito de que el propio robot sea capaz de seleccionar el algoritmo en el módulo reconfigurable, sin intervención humana, elimina los esquemas que necesitan una EPROM. Es necesario que un controlador inteligente (en este caso la tarjeta de adquisición de imágenes log-polares) genere las señales de programación de la FPGA. El robot indica a la tarjeta de adquisición el programa que debe suministrar a cada EP, para configurar de forma completa un algoritmo. Por cuestiones de flexibilidad se ha escogido el esquema asíncrono PPA.

Para evitar realizar un conector de programación por EP se ha diseñado un bus común de programación de las FPGAs de los EPs. A cada EP se le asigna una *dirección física* mediante tres interruptores. Con esta opción de diseño se limita a 8 el número de etapas del cauce para poder realizar una implementación física de la arquitectura propuesta. La metodología expuesta en la presente tesis no tiene una limitación teórica en cuanto al número de etapas, pero por razones tecnológicas de propagación de las señales globales (fundamentalmente reloj), espacio y complejidad de los algoritmos, se ha considerado que 8 son un número máximo razonable de etapas. Los algoritmos diseñados en los capítulos 6 y 7 hacen uso de 3 etapas cada uno de ellos.

Al realizar la programación se pone la dirección del EP que se va a programar en las lineas $A[2..0]$. La PALCE16V8H simplemente hace de interfase entre el bus y la FPGA, habilitando la programación cuando la dirección del bus se corresponda con la dirección fijada por los interruptores. Este circuito es una mera función combinacional de decodificación que ha sido diseñado utilizando PALASM, un lenguaje de programación de PALs. La figura 4.7 describe el conexionado de las señales del bus de programación con la PALCE16V8 y la FPGA.

Para programar la FPGA del EP en primer lugar la tarjeta de adquisición y programación seleccionará en el bus $A[2..0]$ la dirección del EP que se va a programar. La PAL comparará la dirección del bus con su propia dirección de la tarjeta. Si coincide pondrá la señal CS a uno, habilitando el chip para programación.

Estando la dirección estable, la señal nWS pasará a cero para hacer un $reset$ de la FPGA. Cuando esta señal pase a uno se inicia el proceso de programación, indicándolo la FPGA poniendo la señal $nSTATUS$ a uno y $CONF\_DONE$ a cero. Con la señal nWS se activan pulsos bajos de una amplitud mínima de 500 ns. En el flanco de subida de la señal nWS se captura el dato presente en el bus de datos y la señal $RDYnBSY$ cae indicando que la FPGA está ocupada serializando el byte, (durando este pulso un máximo de 4μs). Cuando de nuevo la señal $RDYnBSY$ sube ya se puede activar otro pulso negativo en la señal nWS y otro dato en el bus. Cuando el proceso acaba la FPGA activa la señal $CONF\_DONE$ a uno. En la figura 4.8 se describe mediante un cronograma este proceso de programación. Los tiempos asociados a este cronograma se

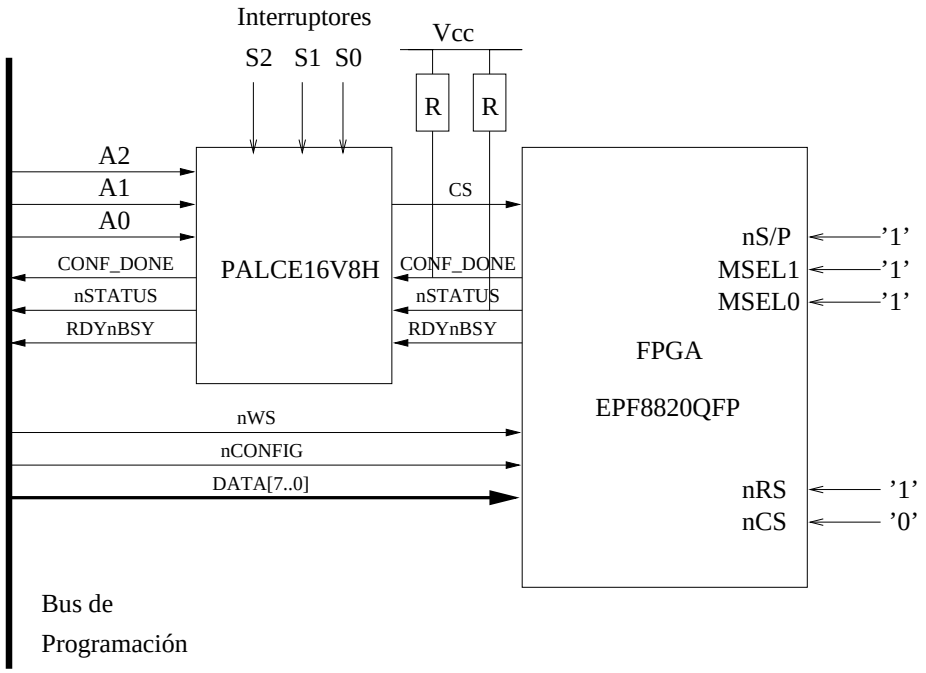


Figura 4.7: *Esquema del bus de programación del EP*

muestran en la tabla 4.4.

El tamaño del fichero describiendo las conexiones de la programación de la FPGA EPF8820 es de 16 Kbytes. La FPGA necesita como máximo 4 μs para serializar los datos por byte. Teniendo en cuenta este tiempo y el resto de tiempos de establecimiento y mantenimiento necesarios en el protocolo de programación se obtiene que el tiempo de programación será de 75 *ms* como máximo por EP.

Es posible incluso realizar una programación parcial de una sola etapa manteniendo el resto de etapas del cauce funcionando. Es por esto que se ha utilizado el adjetivo

Símbolo	Parámetro	Min	Max	Unidades
tCF2WS	nCONFIG alto hasta 1er flanco de nWS	5		μs
tDSU	SETUP del dato antes del flanco de nWS	50		ns
tDH	Mantenimiento del dato después de nWS	0		ns
tCCSU	Selección de circuito antes de nWS	50		ns
tWSP	Ancho de pulso de nWS	500		ns
tW2SB	Tiempo desde nWS hasta RDYnBSY bajo		50	ns
tBUSY	Ancho de pulso RDYnBSY bajo		4	μs
tRDY2WS	tiempo de flanco de RDYnBSY a nWS	50		ns

Tabla 4.4: *Parámetros del esquema de programación pasivo paralelo asíncrono*

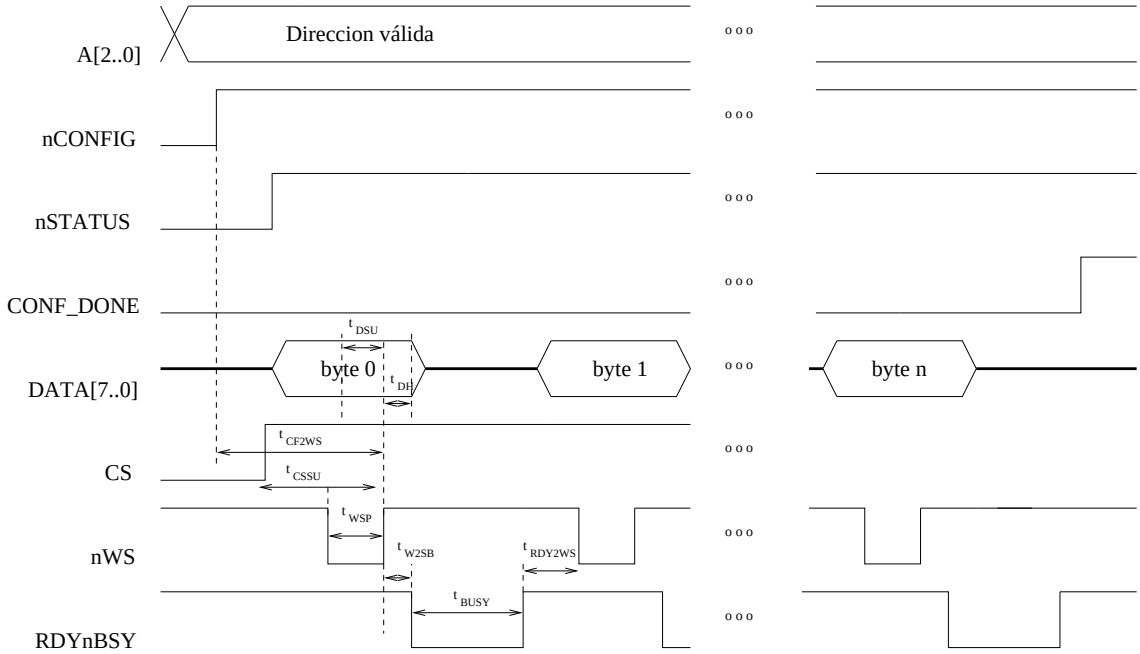


Figura 4.8: *Cronograma de programación pasivo paralelo asíncrono del EP*

reconfigurable para el módulo en lugar de **programable** a pesar de utilizarse lógica programable y no lo que se entiende comúnmente por lógica reconfigurable (dispositivos reprogramables parcialmente). El hecho de que se pueda reprogramar cada etapa del cauce por separado emula la reprogramación parcial que se consigue en los dispositivos reconfigurables. Sin embargo si una etapa del cauce está siendo reprogramada no atenderá al protocolo de intercambio de datos, paralizando de esta manera el cauce segmentado. En cualquier caso la ventaja que se obtendrá será la de que si hay etapas que son iguales en dos algoritmos, éstas no se tendrán que reprogramar, acelerando la programación de un algoritmo. Un interesante análisis de de la reconfiguración en tiempo de ejecución RTR (*Run-time reconfiguration*) puede encontrarse en [Sch97].

4.4 Conclusiones

A partir de la definición del problema planteada en el capítulo 1, y del estado de la investigación revisada en los capítulos 2 y 3, se han planteado los requisitos que debe cumplir el módulo reconfigurable.

Para cumplir con los requisitos de tipo tecnológico, (reconfigurabilidad en el sistema, complejidad de los EPs y prestaciones), se han utilizado FPGAs de arquitectura híbrida de la familia FLEX8000. Estos dispositivos lógicos programables son de tecnología SRAM, configurables en el sistema, y tienen una gran cantidad de registros y

una arquitectura híbrida entre FPGAs y CPLDs. Dentro de la familia FLEX8000 se ha escogido el dispositivo FLEX8820QFP de 820 registros. Esta FPGA presenta un compromiso razonable entre prestaciones y precio.

Los requisitos a nivel de sistema, (alta velocidad en términos de imágenes por segundo y sencillez de diseño de los algoritmos), se han abordado mediante la utilización de paralelismo temporal y espacial y mediante la utilización de imágenes log-polares. El paralelismo temporal se ha planteado mediante el diseño de un cauce segmentado donde cada etapa del algoritmo la realiza un EP. Dentro de cada EP existe la posibilidad de realizar paralelismo espacial, en función de la complejidad de la etapa y de los recursos disponibles en la FPGA.

El hecho de utilizar imágenes log-polares reduce la cantidad de datos a transmitir y procesar, permitiendo la utilización de pocos circuitos de almacenamiento y procesamiento. Además este formalismo simplifica algunos algoritmos de visión interesantes para la plataforma móvil.

La facilidad de diseño se ha planteado con la realización de un sólo protocolo de intercambio de datos, junto con una interfase de comunicación común a todas las etapas. El intercambio de datos no tiene fijada una temporización y permite un tiempo de proceso variable en cada etapa. La transmisión entre EPs puede ser de imágenes completas log-polares, o bien de estructuras de datos. Es necesaria una señal, que permutará cada vez que el dato que se esté transmitiendo pertenezca a una nueva estructura de datos.

Finalmente los requisitos a nivel de algoritmos de visión, (optimización del cauce para algoritmos diferenciales), se consiguen mediante la utilización de memoria en los EPs. Dos bancos de memoria de doble puerto en cada EP permiten la escritura de una imagen en uno de los circuitos mientras se esta leyendo la imagen anterior del otro circuito de memoria. La utilización del doble puerto acelera la transmisión de datos al ser el flujo de datos en una sola dirección, fijando el puerto izquierdo como de sólo escritura y el puerto derecho como de sólo lectura. Esta estructura permite un cálculo eficiente de derivadas temporales.

La diferenciación espacial se facilita mediante un almacenamiento local de la imagen, permitiendo de esta manera el calculo de derivadas espaciales (gradientes). Un ejemplo de esta capacidad se muestra en el capítulo 7, donde en una de las etapas se calcula el gradiente radial de las imágenes log-polares.

La señal de reloj global de los EPs vendrá limitada por la implementación física concreta. El hecho de utilizar FPGAs de grado 3 ha limitado el periodo de reloj global de las FPGAs a cerca de 28 MHz. De manera adicional se pretende automatizar del proceso de diseño de algoritmos en el módulo reconfigurable con lo que se ha evitado editar el emplazado y realizar el conexionado manualmente para mejorar caminos críticos. En

cualquier caso el retraso más importante viene impuesto por las memorias cuyo tiempo acceso es de 25 ns para el caso de la memoria local, y de 35 ns para la memoria de doble puerto. De esta manera el reloj global del sistema se ha fijado a 60 ns.

A partir del diseño del cauce segmentado con los elementos de proceso reconfigurables, se ha de definir una metodología de diseño a modo de *manual* del usuario. Con esta metodología se facilitará el diseño de algoritmos dentro de la arquitectura reconfigurable.

Capítulo 5

Metodología de diseño para el módulo reconfigurable

5.1 Introducción

El módulo reconfigurable ha quedado definido en el capítulo 4 como un cauce segmentado en el que cada EP realiza una etapa del cauce. La utilización de las memorias de doble puerto como memorias de almacenamiento intermedio facilita la diferenciación temporal. De la misma manera, la utilización de una memoria local que almacena una imagen completa, permite la realización de la diferenciación espacial y operaciones globales.

La visión del módulo reconfigurable como un módulo acelerador hardware de los algoritmos de visión es restringida, y limita la potencialidad de la arquitectura desarrollada. En realidad, el módulo reconfigurable está conectado a un *host*, en el cual hay un procesador ejecutando el software de control de los sensores y actuadores que componen la plataforma móvil. El estudio de la arquitectura de capas del robot móvil no es objeto del presente trabajo de investigación, pero la interacción del software con el hardware específico si que puede ayudar a optimizar los algoritmos de visión a implementar.

En las siguientes secciones se presenta la plataforma móvil como un sistema empotrado. De esta manera pueden aplicarse en el diseño de algoritmos de visión artificial, técnicas de codiseño y cosíntesis que optimicen la utilización de recursos, y aceleren la generación del código de programación de las FPGAs. A partir de esta abstracción se sugiere la utilización de una metodología global de codiseño binario en plataformas móviles, línea de trabajo que será retomada en el futuro.

Finalmente, y basándose en la arquitectura propuesta, se definirá una metodología

de diseño para implementar algoritmos de procesado de imágenes diferenciales para navegación de plataformas móviles Esta metodología será utilizada en las secciones 6 y 7 para diseñar dos algoritmos, interesantes para la plataforma móvil, en la arquitectura propuesta.

5.2 La plataforma móvil como un sistema empotrado

Un sistema **empotrado** o *embedded system* es un sistema que tiene como tarea encomendada el control de algún tipo de máquina. El término *empotrado* hace referencia a que este sistema es a su vez una parte de una sistema mayor, donde realiza alguna tarea de control [Mic94].

Un sistema empotrado mixto es aquel en el que, además de un procesador de propósito general, se tiene un hardware específicamente diseñado. Un ejemplo de un sistema empotrado mixto se puede observar en la figura 5.1.

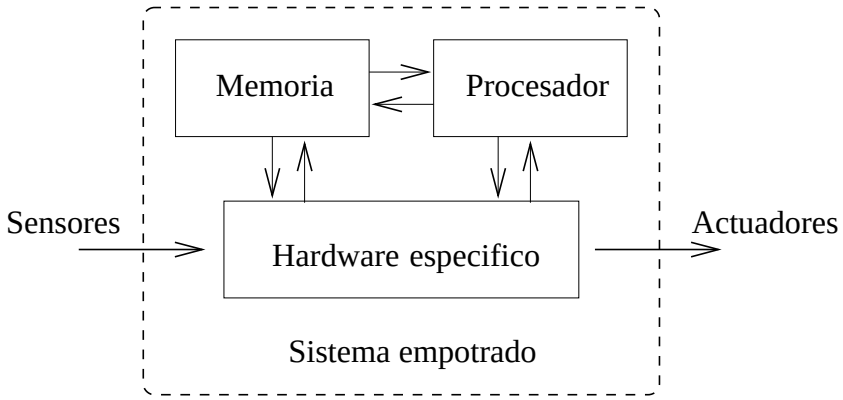


Figura 5.1: *Esquema básico de un sistema empotrado mixto*

Los sensores, que en el caso de la plataforma móvil son el sensor foveal y sensores de ultrasonidos, extraen información del entorno. El sistema compuesto por el hardware específico y el software accionará los actuadores, que en este caso serán los motores de ambas ruedas. Un sistema empotrado como el de la figura 5.1 es un típico ejemplo donde puede ser útil la aplicación de técnicas de **codiseño, cosimulación y cosíntesis**.

Una definición de codiseño puede ser la siguiente [Knu95]:

Codiseño es la metodología mediante la cual, a partir de la funcionalidad requerida para un sistema, se derivan las tareas que el sistema debe realizar, se definen las diversas partes de las cuales constará el sistema, y se asignan las tareas a las diferentes partes del sistema de manera simultánea.

A partir de la definición anterior se puede deducir el significado de *cosimulación* y *cosíntesis*. Por cosimulación se entenderá el proceso en el cual se simula de forma conjunta el sistema previamente a la asignación software-hardware, y la cosíntesis será la generación simultánea del hardware y del software del sistema.

Definir un sistema basado en computador de la manera habitual, es un proceso con lazos de rediseño que lo pueden hacer muy lento y costoso para sistemas complejos, tal como se muestra en la figura 5.2. A partir de las especificaciones que debe cumplir el sistema, el ingeniero decide qué partes se implementarán por software y qué otras partes por hardware. Es entonces cuando se escogen los componentes hardware y/o se diseña el hardware específico necesario. Una vez realizado esto se diseña el software y sólo entonces se puede simular y comprobar si el diseño cumple con la funcionalidad propuesta. De hecho puede ocurrir que los resultados no sean los previstos, obligando a un rediseño del sistema, que puede llegar a ser total.

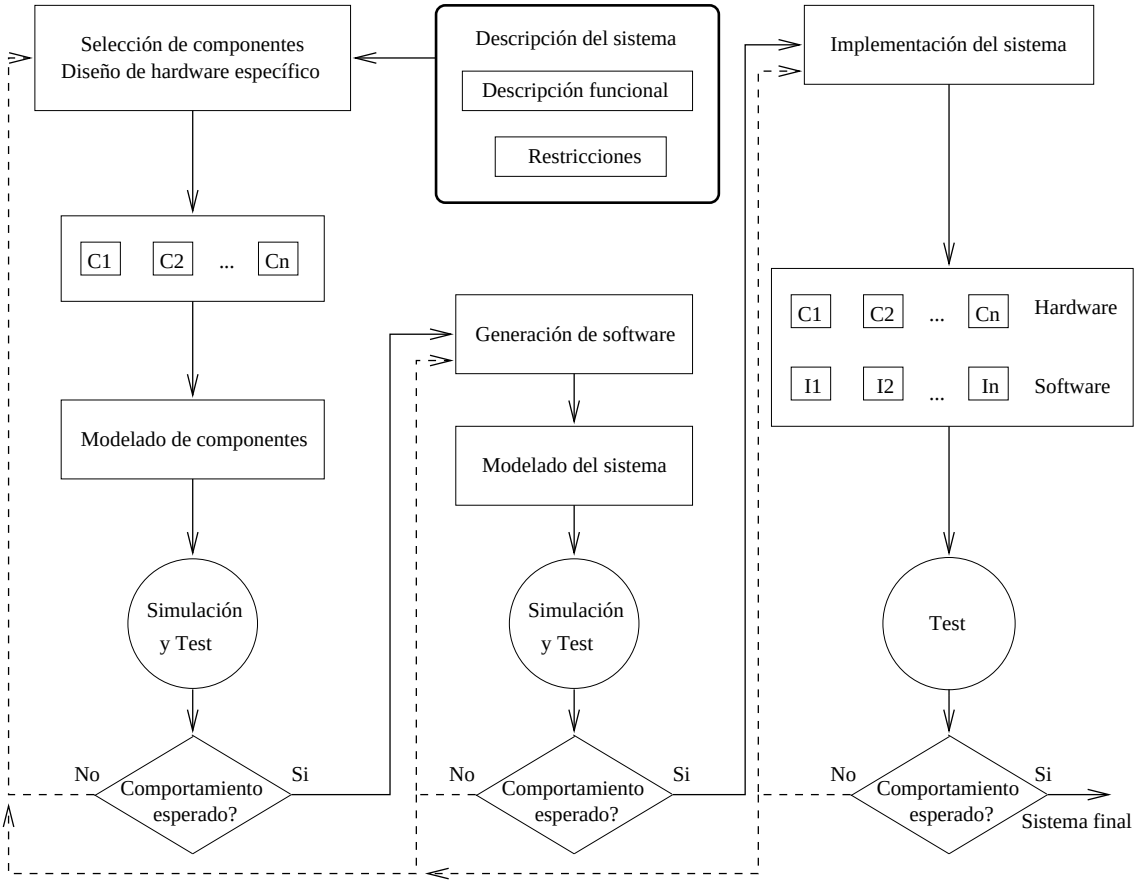


Figura 5.2: *El ciclo de diseño clásico*

La aproximación de codiseño estructurado intenta posponer la decisión sobre la distribución de tareas en el sistema hasta la fase de implementación [IJ95]. Esto se puede realizar describiendo la funcionalidad del sistema con un lenguaje de alto nivel indepen-

diente del sistema como puede ser VHDL o ADA. De hecho a partir de aproximaciones al codiseño desde ambos lenguajes se están desarrollando metodologías que ponen al descubierto su similitud [MP98].

Se puede verificar la correcta funcionalidad del sistema usando métodos formales de cosimulación en esta etapa inicial. Una vez comprobado que la descripción verbal se ajusta a la descripción formal se utilizan técnicas de transformación formales de síntesis que mantienen la correcta funcionalidad del sistema (cosíntesis). De esta manera se evitan muchos lazos de rediseño.

Resolver este problema para un sistema genérico es hoy por hoy inabordable a pesar de que se restrinjan las arquitecturas y se optimice por ejemplo sólo el coste. Sin embargo, en algún caso particular, como el de los sistemas reactivos empotrados[1], se han desarrollado herramientas de codiseño, cosimulación [KL92] y cosíntesis [BCE+97].

Con el entorno POLIS se aborda la cosíntesis binaria de sistemas reactivos descritos en el lenguaje ESTEREL [Ber97]. La cosíntesis binaria reduce la complejidad del problema genérico de cosíntesis mediante una arquitectura más simple, con un módulo de hardware específico comunicado con un procesador estándar mediante un interfase, tal como muestra la figura 5.3.

Figura 5.3: *Arquitectura del codiseño binario clásico*

Observando la arquitectura binaria de la figura 5.3 se pueden observar las similitudes con la arquitectura de la plataforma móvil. El hardware específico sería el módulo reconfigurable, y el canal de comunicación con el procesador que gobierna el robot sería la memoria compartida.

Desde este punto de vista es posible afrontar el diseño de un algoritmo de visión artificial como un problema de codiseño binario. En este caso, y ya con la arquitectura del módulo de proceso definida, hay que estudiar para cada algoritmo particular, qué tareas realizará el módulo reconfigurable, y qué tareas realizará la etapa software.

[1]Un sistema reactivo es aquel cuyo comportamiento viene unívoca y directamente determinada por los estímulos externos

5.3 Descripción y diseño de los algoritmos

Tal como se ha indicado anteriormente, sería deseable realizar una descripción global de los algoritmos propuestos con un lenguaje independiente del sistema. VHDL es un lenguaje que soporta concurrencia y paralelismo de forma natural [IEE93], pudiendo ser utilizado para la descripción de los algoritmos de visión artificial.

El diseño de las herramientas de codiseño y cosíntesis binaria que, a partir de la descripción de los algoritmos en VHDL, generen el código de programación de las FPGAs y el código del procesador estándar, queda para un trabajo futuro de investigación. En este trabajo se indica qué tareas deberá realizar el módulo reconfigurable y que tareas quedan para el procesador. A partir de aquí se indican las líneas a seguir para generar el código VHDL para las FPGAs.

Un algoritmo de visión artificial **F**, implementable en diversas etapas sin realimentaciones, puede abstraerse como una función que opera con elementos de L, donde L es el conjunto de las imágenes log-polares posibles. Como resultado se puede obtener una nueva imagen log-polar transformada o una estructura de datos del conjunto D. Esta función **F** se puede a su vez descomponer en funciones más simples, que operarán de manera encadenada sobre las imágenes resultado de las funciones anteriores, tal como se muestra en la figura 5.4.

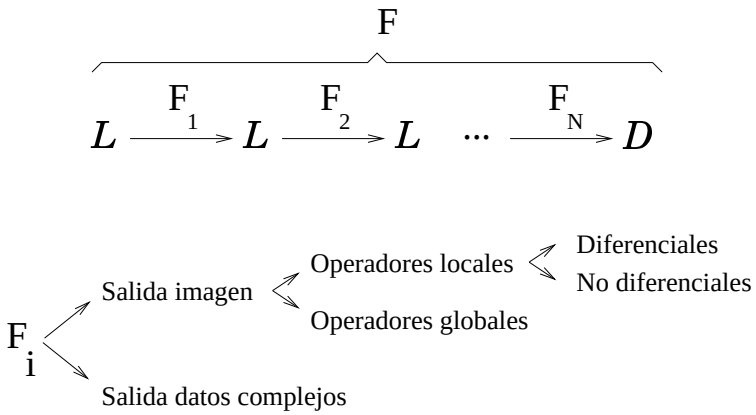


Figura 5.4: *Descomposición y clasificación de los algoritmos por etapas de visión artificial*

A partir de esta estructura se pueden clasificar las subfunciones F_i teniendo en cuenta en primer lugar si el resultado es una imagen o un conjunto de datos. Las funciones cuya salida es una imagen pueden a su vez clasificarse en función de la localidad de las operaciones. Si para obtener un punto resultado solamente intervienen puntos vecinos (en el espacio y en el tiempo) la función se considerará local. Si por el contrario, para un punto resultado intervienen cantidades extraídas globalmente de toda la imagen, la

función se considerará global. Dentro de las funciones con operadores locales aparecen las funciones diferenciales, tanto en el espacio como en el tiempo.

El módulo reconfigurable puede realizar *a priori* cualquier algoritmo de visión artificial de la clasificación propuesta. La escalabilidad de la arquitectura, combinada con la comunicación con el procesador estándar, suministran flexibilidad y capacidad de computación para abordar los algoritmos de visión artificial propuestos.

La comunicación y procesado byte a byte es eficiente cuando existe alguna etapa diferencial. Es entonces cuando se obtiene el máximo rendimiento del cauce segmentado ya que simultáneamente se produce la transmisión y cálculo en los EPs. Esto se verá con más detalle en los algoritmos ejemplo que se han diseñado en los capítulos 6 y 7. En este tipo de algoritmos cada subfunción diferencial será implementada por un EP o etapa del cauce segmentado. Quedará para el procesador estándar una última posible etapa de cálculos globales a partir de la última imagen procesada. De esta manera quedan asignadas al módulo reconfigurable (partición hardware) las funciones locales diferenciales y al procesador estándar (partición software) la última etapa de operaciones globales (si la hay).

Cuando se trata de operadores locales no diferenciales, dependerá del problema concreto y de la implementación concreta la optimización de la partición hardware-software.

Con estas consideraciones, y dejando como línea abierta para trabajo futuro el codiseño binario en la arquitectura reconfigurable en el caso más global, se plantea la metodología de diseño en las etapas del cauce reconfigurable.

5.3.1 Descripción de las etapas y flujo de información

Un algoritmo genérico se descompondrá en funciones simples, de manera que cada función se podrá implementar en un EP. La descripción de la funcionalidad de los EPs se realizará mediante el lenguaje VHDL. El lenguaje VHDL es un lenguaje estándar de descripción del hardware orientado a modelado y simulación. La ventaja de utilizar un lenguaje de descripción del hardware es la portabilidad entre herramientas de síntesis lógica. A pesar de ser un lenguaje de gran complejidad, es posible utilizar un subconjunto bastante grande de instrucciones y estructuras con el fin de sintetizar hardware en lógica programable [Ska96] [GS95] [Sca97].

Cada etapa del cauce implementará la máquina de estados de la figura 4.3 para realizar la transmisión/recepción de datos. Las operaciones incluidas en el procesado de los datos no deben ser excesivamente complejas, para no incluir caminos de datos que ralenticen la frecuencia de reloj de la máquina. La velocidad de las memorias de

doble puerto de los EPs (35 ns de tiempo de acceso), el reloj de la tarjeta de adquisición (30 ns), y pruebas de compilación en las FPGAs que generaban máquinas de estados con 40ns de ciclo de reloj, han hecho elegir la frecuencia de reloj como la mitad de la frecuencia de la tarjeta de adquisición, fijando un periodo de 60 ns. Sería posible realizar una edición del emplazado y del rutado para intentar optimizar el conexionado y por tanto la velocidad de las FPGAs. Esta edición a bajo nivel del código generado por el compilador podría aumentar las prestaciones del sistema, pero iría en contra del objetivo final de automatización del proceso de diseño de algoritmos en el módulo reconfigurable

El código VHDL, básicamente comportamental, de los dos algoritmos que se han diseñado en los capítulos 6 y 7, se muestra en el apéndice B.

La máquina de estados de las etapas vendría a ser un $PROCESS$ de las señales $RESET$ y CLK con un estado inicial donde se entraría al realizar el $RESET$ y del que no se saldría hasta que el EP anterior suministre el primer dato. Esta funcionalidad se implementa con un código tal como este:

```
BEGIN
 PROCESS (clk, reset)
   BEGIN
   IF reset='1' THEN estado<=Sini;
   ELSIF (clk'event AND clk='1') THEN
     CASE estado IS
        WHEN Sini => --Inicializaciones previas.
             .....     .....     .....
             IF data_ready0='1' THEN data_received1<='1'; estado<=S0;
                                ELSE data_received1<='0'; estado<=Sini;
```

Cuando el EP anterior valida el anterior dato éste se captura y se pasa al siguiente estado donde se inicia el procesado. Al terminar el procesado se suministra el dato al siguiente EP del cauce y la máquina queda paralizada hasta que el siguiente EP reconoce haberlo recibido. Es entonces cuando se pasa a esperar el siguiente dato y se cierra el bucle.

```
        WHEN Si => --Se ha terminado de procesar el dato.
                   --Se espera a que el siguiente EP lo capture.
             IF data_received2='1' THEN estado<=Si+1; data_ready1<='0';
                                   ELSE estado<=Si; data_ready1<='1';
             END IF;
        WHEN Si+1 => --Se espera siguiente dato listo.
             IF data_ready='1' THEN estado<=S0; data_received1<='1';
                               ELSE estado<=Si+1;  data_received1<='0';
             END IF;
```

El número de ciclos de cada etapa del cauce debe estar equilibrado ya que la etapa más lenta será la que marque la velocidad de salida de datos en el cauce segmentado.

Siguiendo esta sencilla metodología y tratando de realizar el mayor número de acciones en paralelo para disminuir el número de ciclos, se han diseñado dos algoritmos útiles para navegación robótica que se describen en los siguientes capítulos.

5.3.2 Cálculo diferencial temporal y espacial

Cuando la entrada y la salida de un EP son imágenes log-polares, se puede sistematizar de una manera sencilla el cálculo de de derivadas temporales. Sea f una función parte de un algoritmo de procesamiento de imágenes. Dicha función tiene como entradas las imágenes log-polares $l(t)$ y $l(t-1)$, imágenes tomadas en los respectivos instantes de tiempo. Sea $S_{ij}(t)$ el punto resultado de aplicar dicha función para obtener el punto de las coordenadas (i, j).

$S_{ij}(t)$ será función de los puntos de un entorno de $a_{ij}(t)$ y de un entorno de $a_{ij}(t-1)$. Cuando aparece una dependencia temporal y $S_{ij}(t)$ es función de $a_{ij}(t-1)$ hay que realizar una diferenciación temporal de las imágenes. Se utilizan las memorias de doble puerto para almacenar de forma completa una imagen. En la figura 5.5 se puede observar como el EP n-ésimo accede a la imagen $t-1$ almacenada en el banco inferior del EP $n-1$. Simultáneamente el EP $n-1$ almacena la imagen que está procesando en el banco de arriba y se la suministra directamente al EP n-ésimo.

Con la transmisión de datos a nivel de byte se utilizará, además de las señales *dato_listo* y *dato_recibido*, la señal *img* para señalar el cambio de imagen. De esta manera, si para la imagen que se esta transmitiendo *img* vale 0, cuando se transmita la siguiente imagen *img* valdrá 1. El valor de la señal *img* debe ser el correcto cuando se valide el dato con la señal *dato_listo*.

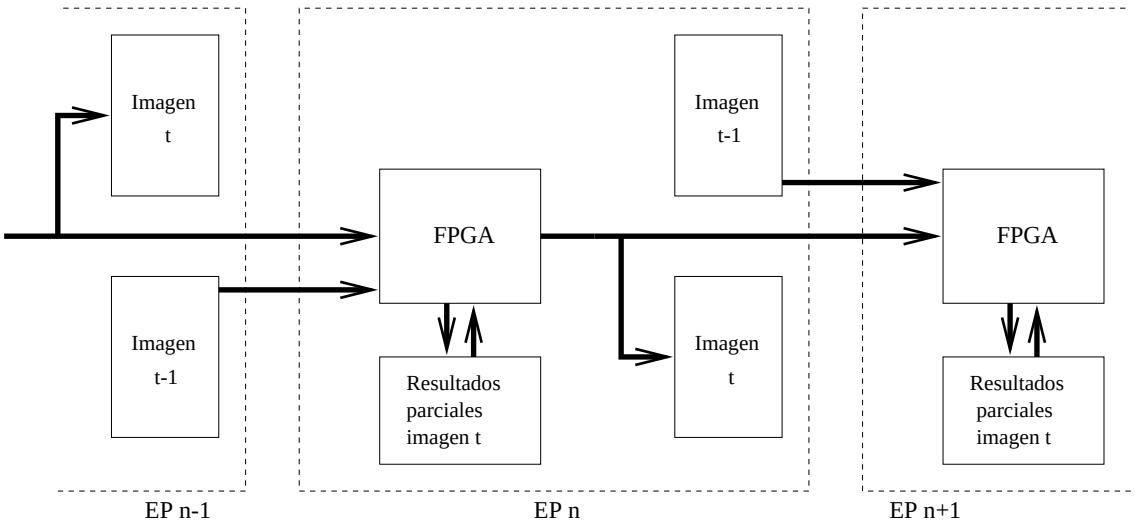


Figura 5.5: *Ejemplo en el que el EP n-ésimo calcula la primera derivada temporal*

Si el elemento $S_{ij}(t)$ depende de n puntos de la imagen presente, entonces con la transmisión/procesado a nivel de byte, a la vez que un EP recibe los datos del EP anterior los almacena en memoria local. De esta manera cuando ya ha recibido la parte de la imagen necesaria, puede calcular el elemento $S_{ij}(t)$ y transmitirlo al EP siguiente.

Esta metodología es especialmente útil si se quiere realizar una diferenciación espacial sobre la imagen que se esta recibiendo. Un ejemplo es el cálculo de gradientes, caso que se ejemplifica en el capítulo 7.

5.4 Conclusiones

A partir del diseño del módulo reconfigurable se hace necesario establecer una guía de diseño de algoritmos de visión artificial bajo la arquitectura propuesta. La combinación del hardware específico del módulo reconfigurable, con el procesador estándar que gobierna la plataforma móvil, hace plantearse de qué manera se pueden dividir las tareas de los algoritmos de visión de manera eficiente entre ambas partes. Este es un típico problema de codiseño binario del sistema empotrado en el que se pueden aplicar técnicas de codiseño para optimizar la parte que implementará la partición hardware, y la parte que implementará la partición software.

El problema del codiseño, cosimulación y cosíntesis es un problema extremadamente complejo. Aún acotándolo a la síntesis binaria de un elemento hardware y un elemento software, sólo con grandes restricciones se puede plantear un codiseño binario práctico. En el caso del problema abordado por el presente trabajo de investigación, los aspectos de codiseño ligados a la implementación de algoritmos genéricos útiles para navegación robótica, se han dejado para posteriores trabajos de investigación.

La definición de la arquitectura realizada en el capítulo 4 y las restricciones a los algoritmos de visión artificial que van a ser útiles, plantean la realización de la partición hardware/software *a priori*, sin temor a una gran pérdida de eficiencia. Se han implementado en el módulo reconfigurable (partición hardware) las etapas con operadores locales, tanto espaciales como temporales. Queda para el procesador estándar (partición software) el análisis global de las imágenes ya procesadas por el cauce segmentado, y la posterior extracción de parámetros globales de relieve en la navegación de la plataforma.

En las siguientes secciones se plantea el diseño de dos algoritmos, útiles para navegación robótica, siguiendo esta metodología de trabajo.

Capítulo 6

Diseño del algoritmo de detección de movimiento

6.1 Introducción

Dentro del amplio rango de algoritmos interesantes para una plataforma autónoma móvil es interesante la detección de objetos en movimiento. Una plataforma móvil debe ser capaz de detectar el movimiento de otros objetos de su entorno para planificar su trayectoria y evitar choques.

Este problema, que se reduce al problema clásico de detección de objetos móviles si la plataforma permanece estática y con la posición de la cámara fijada, es un problema complejo cuando el vehículo tiene movimiento propio. Descartar el desplazamiento en las imágenes debido al movimiento propio, y aislar el desplazamiento de las imágenes debido a movimientos externos, es una tarea con un gran coste computacional. Una manera de afrontar este problema consiste en calcular el flujo óptico de las imágenes y posteriormente, a partir del mapa de vectores obtenido, realizar una segmentación de la imagen. De esta manera, y eliminando de la imagen la componente debida al movimiento propio, se pueden identificar grupos de vectores paralelos e identificar objetos.

La elegante aproximación de cálculo de flujo óptico y posterior segmentación de la imagen, es muy costosa computacionalmente y requiere una gran cantidad de recursos para poder implementarla en tiempo real [BB95]. De la misma manera, la opción de extracción de puntos relevantes y posterior encaje de éstos también tiene los problemas de alto coste computacional y dependencia del problema, tal como se ha justificado en el capítulo 3.

Como opción alternativa se han estudiado algoritmos diferenciales de detección de

movimiento, que una vez adaptados al formalismo log-polar, ofrezcan unas buenas prestaciones. Siguiendo la metodología de diseño expuesta en el capítulo 5 se ha diseñado el algoritmo en el módulo reconfigurable que se expone a continuación.

6.2 El algoritmo original en coordenadas cartesianas

Chen y Nandhakumar demuestran en [CN93] unas interesantes propiedades que cumplen los bordes de los objetos en movimiento en las secuencias de imágenes que cumplen una serie de condiciones. El planteamiento dice así:

Sea $E(x, y, t)$ una secuencia de imágenes que varían con el tiempo, $T(\Sigma)$ la proyección de la superficie Σ en el plano de proyección Π, y $\delta T(\Sigma)$ los puntos del borde de $T(\Sigma)$. Los supuestos para el algoritmo son:

1. E es lineal a tramos respecto a x e y en cualquier punto que pertenezca a la proyección de una superficie en el plano imagen. Esto significa que:

$$\frac{\partial^2 E}{\partial x^2} = \frac{\partial^2 E}{\partial y^2} = 0 \tag{6.1}$$

   en todos los puntos de $T(\Sigma) - \partial T(\Sigma)$.

2. El movimiento de la escena es suave respecto al tiempo, lo cual significa que $\forall x, y \in T(\Sigma)$

$$\frac{d^2 x}{dt^2} = \frac{d^2 y}{dt^2} = 0 \tag{6.2}$$

Con estos supuestos y cumpliéndose la ecuación del flujo óptico para la secuencia $E(x, y, t)$

$$v_x \frac{\partial E}{\partial x} + v_y \frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} = 0 \tag{6.3}$$

Se demuestra que $\forall x, y \in T(\Sigma) - \partial T(\Sigma)$

$$\frac{\partial^2 E}{\partial t^2} = 0 \tag{6.4}$$

Es decir, la derivada segunda temporal será nula en los puntos de la imagen que pertenecen a la superficie del objeto, siendo distinta de cero en los puntos de los bordes.

Cabe hacer notar que siempre que se cumpla la suavidad en el movimiento y en los tonos de gris de la imagen la derivada segunda será distinta de cero sólo en los bordes de la imagen. La variación de la imagen debida a que la cámara se mueve queda totalmente eliminada detectándose sólo el movimiento propio del objeto.

El supuesto de suavidad en la imagen no es tan restrictivo como pudiera parecer. De hecho, en imágenes de entornos interiores, compuestos fundamentalmente por objetos hechos por el hombre, y bajo iluminación difusa, es un modelo razonable. En cualquier caso, puede conseguirse de forma aproximada aplicando un filtro pasa-baja. El supuesto de linealidad en el movimiento de la cámara se puede conseguir también de forma aproximada en tres imágenes consecutivas, si las imágenes se toman muy rápidamente. Esta aproximación ha mostrado ser muy útil por la simplicidad de las computaciones a realizar y ha sido aplicada a la detección de vehículos móviles desde imágenes aéreas.

6.3 Adaptación al formalismo log-polar

Partiendo de estos resultados, y como parte de este trabajo de investigación, se ha desarrollado el algoritmo log-polar de detección de movimiento que a continuación se describe y cuyos resultados teóricos han sido publicados. En [BDPP97b] se plantea el desarrollo en el formalismo log-polar puro, y en [BDPP97a] se adapta al sensor log-polar CMOS, con la transformación particularizada a la fóvea y a la retina.

En este caso la secuencia de imágenes que varían con el tiempo sería $E(\xi, \theta, t)$, y las condiciones en el plano retínico equivalentes a las cartesianas serán:

1. Suavidad con respecto a ξ y θ en cualquier punto que pertenezca a la proyección de una superficie en el plano imagen. Es decir:

$$\frac{\partial^2 E}{\partial \xi^2} = \frac{\partial^2 E}{\partial \theta^2} = 0 \tag{6.5}$$

2. El movimiento debe cumplir la condición de suavidad $\forall\ (\xi, \theta) \in T(\Sigma)$:

$$\frac{d^2 \xi}{dt^2} = \frac{d^2 \theta}{dt^2} = 0 \tag{6.6}$$

Utilizando la ecuación de transformación del sensor log-polar (3.1) tanto para la fóvea como para la retina y la ecuación de Horn (6.3), se puede calcular la ecuación equivalente log-polar:

$$v_\xi \frac{\partial E}{\partial \xi} + v_\theta \frac{\partial E}{\partial \theta} = -\frac{\partial E}{\partial t} \tag{6.7}$$

Bajo estas condiciones, se cumple que $\forall \xi, \theta \in T(\Sigma) - \partial T(\Sigma)$

$$\frac{\partial^2 E}{\partial t^2} = 0 \tag{6.8}$$

La equivalencia es total al ser el plano cortical equivalente al plano cartesiano. Simplemente queda interpretar la equivalencia de las condiciones del teorema a supuestos log-polares. La condición de suavidad en la imagen se resolverá de nuevo aplicando un suavizado, en este caso en el plano log-polar.

La condición más interesante es la que se refleja en el movimiento descrito en la ecuación (6.6). Si se quiere averiguar qué tipo de movimiento puede realizar la cámara, de tal modo que quede automáticamente descartado por el algoritmo, no hay mas que resolver la ecuación 6.6. Para ello utilizaremos la ecuación de transformación del sensor log-polar 3.1, obteniendo las ecuaciones diferenciales:

$$\begin{cases} (x\dot{x} + y\dot{y})^2 = (x^2 + y^2)(\dot{x}^2 + \dot{y}^2 + x\ddot{x} + y\ddot{y}) & \text{para la fóvea} \\ (x\dot{x} + y\dot{y})(2x\dot{x} + 2y\dot{y}) = (x^2 + y^2)(\dot{x}^2 + \dot{y}^2 + x\ddot{x} + y\ddot{y}) & \text{para la retina} \\ (x\dot{y} - y\dot{x})(2x\dot{x} + 2y\dot{y}) = (x^2 + y^2)(x\ddot{y} - y\ddot{x}) & \text{para ambas zonas} \end{cases} \tag{6.9}$$

Se puede comprobar que las soluciones generales a (6.9) para la retina y la fóvea son respectivamente:

$$\begin{cases} x = Ae^{b+vt}\cos(\theta_o + \omega t) \\ y = Ae^{b+vt}\sin(\theta_o + \omega t) \end{cases} \qquad \begin{cases} x = (r_o + vt)\cos(\theta_o + \omega t) \\ y = (r_o + vt)\sin(\theta_o + \omega t) \end{cases} \tag{6.10}$$

Estas ecuaciones representan espirales exponenciales y lineales donde A, b, v, θ_o, ω y r_o son constantes. La constante ω se interpreta como una velocidad angular constante. La otra velocidad se puede interpretar como una velocidad de translación de la cámara. r_o y θ_o son constantes relacionadas con la posición inicial

En el caso de la componente angular se concluye que un movimiento de auto-rotación sobre el eje óptico, con una velocidad angular constante se incluye como un movimiento automáticamente no detectable por el algoritmo. Sin embargo, relacionar el movimiento de traslación que aparece en la fóvea y en la retina, con un movimiento de la cámara que sea útil que no detecte el robot no es tan inmediato.

En la figura 3.5 se observa como crece la imagen de un objeto que se aproxima a la cámara. $r = f\frac{r'}{z}$ donde r' es el tamaño del objeto, f es la distancia focal de la cámara, z es la distancia desde el objeto al punto focal y r es la imagen en el plano focal.

Si se asume una translación lineal de la cámara a lo largo de la dirección del eje óptico, el parámetro z decrecerá linealmente como $z = z_o - v_t t$. Del modelo de la figura 3.5 se obtiene:

$$r = f\frac{r'}{z_o - v_t t} \tag{6.11}$$

Donde v_t es la velocidad de traslación a lo largo de la dirección del eje óptico. En el capítulo 8 se analiza el error introducido en este algoritmo cuando el eje óptico no coincide exactamente con la dirección de movimiento. La dependencia radial es exponencial para la solución de la retina y lineal para la fóvea. Si se compara este resultado con la aproximación lineal de un objeto expresada en la ecuación (6.11), se observa que estas cantidades son muy similares. De hecho, normalizando y desarrollando la solución para la retina (6.10) en serie de Taylor alrededor del origen, los polinomios son muy similares.

$$\begin{aligned} \tfrac{1}{1-x} &= \textstyle\sum_{n=0}^{\infty} n! x^n = 1 + x + 2x^2 + \ldots \\ e^x &= \textstyle\sum_{n=0}^{\infty} x^n = 1 + x + x^2 + \ldots \end{aligned} \tag{6.12}$$

Para la fóvea esta aproximación es correcta inicialmente. La diferencia aparece en la retina el término de tercer orden. El supuesto realizado por la transformación log-polar propuesta consiste en considerar que el crecimiento del objeto en el plano retínico (o plano imagen), corresponde a la solución de (6.9), para la retina y la fóvea. Esta aproximación es suficientemente válida tal como se muestra teóricamente en el capítulo 8, y se comprueba en los resultados experimentales del capítulo 9. En cualquier caso, esta aproximación se podría anular forzando un movimiento del robot que siga la ecuación (6.10). Planificar la realización de movimientos particulares para que el robot tenga una percepción del entorno más precisa es una tarea que se abordará en otros trabajos de investigación.

Como conclusión, la adaptación del algoritmo diferencial cartesiano a coordenadas log-polares permite descartar, de manera automática, el desplazamiento en la imagen debido a:

1. Auto-rotaciones de la imagen alrededor del eje óptico.
2. Desplazamiento de la cámara en la dirección del eje óptico.

El segundo caso es realmente muy útil, ya que simplemente orientando la cámara log-polar en la dirección de movimiento, se podrá detectar movimiento independientemente del movimiento propio de la plataforma móvil. El desplazamiento de los objetos en la imagen (crecimiento/decrecimiento) debido a la traslación de la cámara quedará automáticamente descartado. Esta orientación se podrá conseguir sin problemas de

modo aproximado al tener la plataforma móvil una la cámara montada en una cabeza orientable.

Cabe finalmente hacer notar que en realidad no bastará con comprobar aquellos puntos de la imagen log-polar en los que la segunda derivada temporal es distinta de cero. El hecho de que no se cumplan de forma absoluta ambas condiciones de suavidad expresadas en (6.5) y (6.6) y la aproximación (6.12) , hacen que la condición (6.8) no se cumpla de forma exacta. Habrá que aplicar un margen umbral, de manera que si la derivada segunda temporal sale fuera de este margen, el punto será seleccionado como poseedor de movimiento propio independiente del movimiento de la cámara.

El supuesto de linealidad en el movimiento de la cámara se podrá conseguir con 3 imágenes consecutivas suficientemente próximas en un desplazamiento lineal. En las experiencias del capítulo 9 la plataforma móvil se ha fijado con una velocidad constante rectilínea para que este supuesto sea cierto.

Ya sólo queda encajar este algoritmo diferencial de visión artificial dentro de la arquitectura del módulo reconfigurable mediante la metodología propuesta. Para ello se han diseñado las etapas del cauce que se presentan a continuación.

6.4 Etapas en el cauce reconfigurable

El algoritmo anteriormente descrito presenta una gran utilidad *a priori* para una plataforma móvil. Simplemente orientando la cámara en la dirección de movimiento se descarta el desplazamiento en las imágenes debido a la auto-translación. En este caso el formalismo log-polar, además de reducir la cantidad de datos a procesar, plantea una interesante aplicación del algoritmo. La geometría radial del sensor permite que, en el caso de que el movimiento sea a través del eje óptico, se simplifiquen los cálculos. La escala exponencial y lineal se puede aproximar a la solución del crecimiento en plano imagen debido al desplazamiento a lo largo de el eje óptico.

El algoritmo de detección de movimiento se puede desglosar en las siguientes etapas:

1. Adquisición de las imágenes $E(\xi, \theta, t)$.
2. Filtrado de la imagen para realizar un suavizado.
3. Cálculo de la primera derivada temporal.
4. Cálculo de la segunda derivada temporal a partir de la etapa anterior y binarización.

Todas las etapas tienen como entrada y salida una imagen log-polar. Además, este algoritmo es totalmente implementable en la partición hardware, dejando el procesador dedicado a otras tareas de la plataforma móvil. No hay ninguna operación global, las

etapas realizan operaciones locales (convolución), o diferenciales (derivadas temporales). Quedaría para el procesador la interpretación de la imagen binarizada agrupando puntos cercanos en objetos, y tras un seguimiento de varias imágenes y conociendo la posición propia, planificar las modificaciones a la trayectoria.

Nótese que se efectuará el cálculo de la segunda derivada temporal a partir del cálculo de la primera derivada. De esta manera se realiza un mejor ajuste en la arquitectura propuesta y se aprovecha al máximo el paralelismo temporal del cauce segmentado.

6.4.1 Etapa de suavizado de la imagen

En esta etapa la entrada al EP será un byte de la imagen log-polar tomada por la tarjeta de adquisición y la salida una imagen log-polar suavizada. Existen diversos métodos para realizar un suavizado de la imagen y eliminar *picos* o funciones escalón. Por simplicidad de las computaciones se ha escogido la aplicación de una simple máscara unitaria de convolución.

Es una elección comúnmente aceptada tomar una máscara cuadrada, con un tamaño mínimo de 3x3 y con coeficientes gaussianos [Can96]. Para simplificar las computaciones y acelerar el cálculo se va a realizar una convolución con coeficientes unitarios. De esta manera sólo se tendrá que sumar el valor de gris de 6 *pixels* vecinos. Así se evita la implementación de multiplicadores, que ocuparían una gran cantidad de celdas lógicas y habría que segmentar entre varios EPs para aumentar su rapidez.

En la figura 6.1 se muestra la solución adoptada. Se ha implementado un registro de desplazamiento de 128 bytes (el tamaño de una fila log-polar) con la memoria local. Dentro de la FPGA se realiza un desplazamiento a lo largo de 3 bytes, de manera que cuando llega un nuevo byte suministrado por la cámara el que sale de la FPGA se guarda en memoria local. Análogamente hay otra secuencia de 3 registros que recuperan los 3 bytes correspondientes de la fila anterior, simplemente accediendo a memoria local. Este acceso a la fila que se ha recibido con anterioridad es posible, porque con la memoria local se ha implementado un registro de desplazamiento de 125 bytes, que sumado a los 3 de la FPGA completan una fila.

Teniendo en cuenta que el valor máximo de un byte suministrados por la cámara es de 255 en la escala de grises (punto totalmente blanco), el valor máximo que se puede alcanzar como resultado de la suma es de $255 * 6 = 1.530$. Para almacenar este valor binario son necesarios 11 bits, con los que se puede codificar hasta 2.048 valores en la escala de gris.

Para evitar tener que dividir por 6 el valor resultado de la suma y tener que implementar un divisor, circuito que ocuparía muchas celdas lógicas, se trunca el resultado

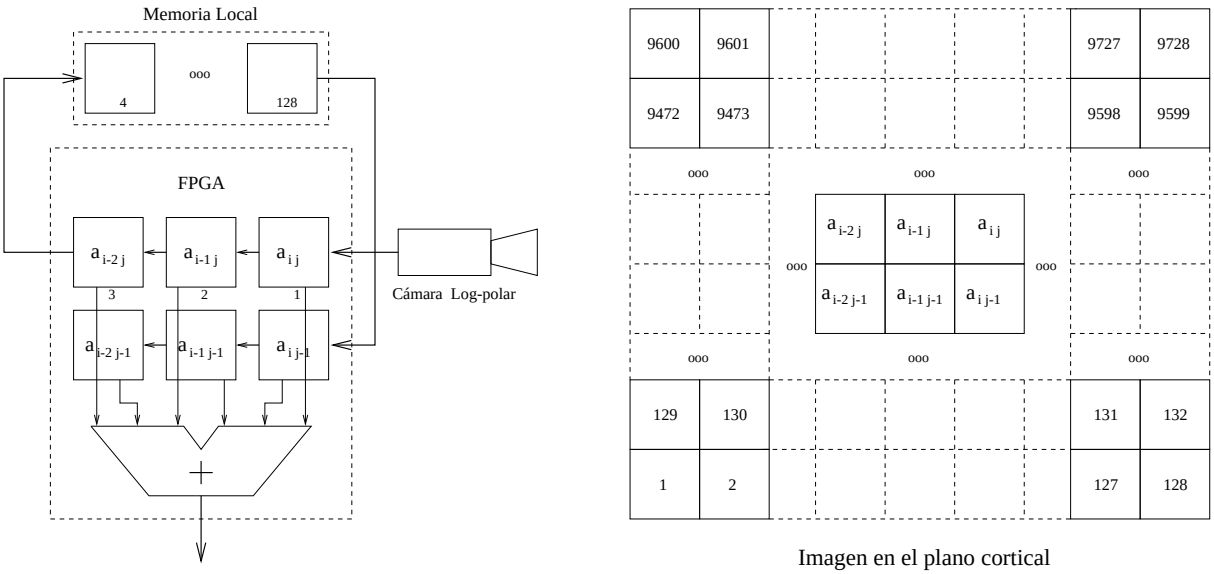


Figura 6.1: *Esquema del flujo de datos en el EP que realiza el suavizado*

de 11 bits seleccionando los 7 bits más significativos y desechando los 4 menos significativos. Esto es equivalente a realizar una división entera por 16 en lugar de dividir por 6 que sería el valor exacto.

Se han seleccionado los 7 bits más significativos, en vez de seleccionar 8 y realizar una división por 8 que sería más aproximado a 6, porque se pretende que el resultado de 7 bits sea una representación en complemento a 2. Este resultado, al tener el bit más significativo siempre a cero, será siempre positivo.

Este re-escalado para dejar el resultado de la convolución como un número positivo de 7 bits, es necesario porque la siguiente etapa operará con estos valores y el resultado podrá ser negativo, obteniendo un número de 8 bits en complemento a 2.

Si como resultado de la suma de los bytes de la máscara se obtiene el valor más alto (1.530) y se seleccionan los 7 bits más significativos se obtiene el nivel 95 en una escala de 127 valores de gris. El nivel 16 daría un valor una vez re-escalado de 1 y cualquier valor resultado de la suma menor que 16 dará un valor 0.

Es posible concluir que mediante esta aproximación para realizar el suavizado de la imagen se gana en rapidez al no tener que realizar multiplicaciones ni divisiones, pero se pierde en precisión en la escala de grises. Básicamente se oscurecen las imágenes al quedar con valores menores en la escala de grises. Es pues recomendable tomar imágenes iniciales con una iluminación correcta. Los resultados experimentales de la sección 9 justificarán las aproximaciones realizadas.

De esta manera, y solamente sumando el valor de los 6 bytes de la máscara de dentro de la FPGA, se realiza un suavizado del punto a_{ij} obteniéndose el punto S_{ij}

cuyo resultado expresado analíticamente sería:

$$S_{ij} = \lceil \frac{a_{ij} + a_{i-1j} + a_{i-2j} + a_{ij-1} + a_{i-1j-1}a_{i-2j-1}}{16} \rceil \quad 128 \geq i \geq 1,\ 76 \geq j \geq 1 \quad (6.13)$$

Cuando $i = 1$ claramente el resultado será incorrecto porque se estará realizando la media con unos valores que se leerán de memoria de la fila anterior, que no existe (por simplicidad de la máquina no se han tenido en cuenta estos casos límite). Esto es poco importante porque la primera fila corresponde sólo al *pixel* central del sensor, con lo que toda la fila incorrecta no es mas que el *pixel* central incorrecto.

Con $i > 1$ y $j = 1$ la máscara estará un poco desplazada, pero realizándose la convolución con valores cercanos al elemento a_{ij}. El formalismo log-polar tiene como ventaja adicional que elimina los bordes laterales de la imagen, siendo la columna 1 del plano cortical consecutiva a la columna 128.

La máquina de estados de esta etapa básicamente se comunicará con la tarjeta de adquisición, para obtener las imágenes, siguiendo el protocolo de intercambio de datos descrito en el capítulo 4. Accederá a memoria local para ir almacenando las filas para realizar la convolución, tal como se ha descrito en esta sección, y generará las señales adecuadas para guardar la imagen suavizada en memoria de doble puerto intermedia. La etapa siguiente accederá la imagen almacenada en esta memoria simultáneamente a la nueva imagen generada por este módulo para calcular la derivada primera temporal.

Paralelizando estas acciones al máximo (petición de datos, acceso a memoria local y escritura en memoria de doble puerto) se obtiene una máquina con un ciclo de 4 estados. El código VHDL de esta etapa se muestra en el apéndice B.

6.4.2 Etapa de cálculo de la primera derivada temporal

La entrada de esta etapa serán los bytes de la imagen suavizada de la etapa anterior. La salida será la primera derivada temporal calculada realizando la resta de la imagen que en un instante le suministre el EP anterior $Img(t)$, menos la imagen almacenada anteriormente en la memoria de doble puerto $Img(t-1)$.

En la figura 6.2 se muestra la situación de los datos con varias imágenes al realizarse la primera diferenciación. La imagen C se ha tomado en el instante t, la imagen B en $t-1$ y la A en el instante $t-2$. El primer EP está realizando el suavizado de la imagen C mientras la escribe y se la suministra al EP que calcula la derivada temporal. En el banco donde se está escribiendo la imagen C estaba con anterioridad la imagen A suavizada.

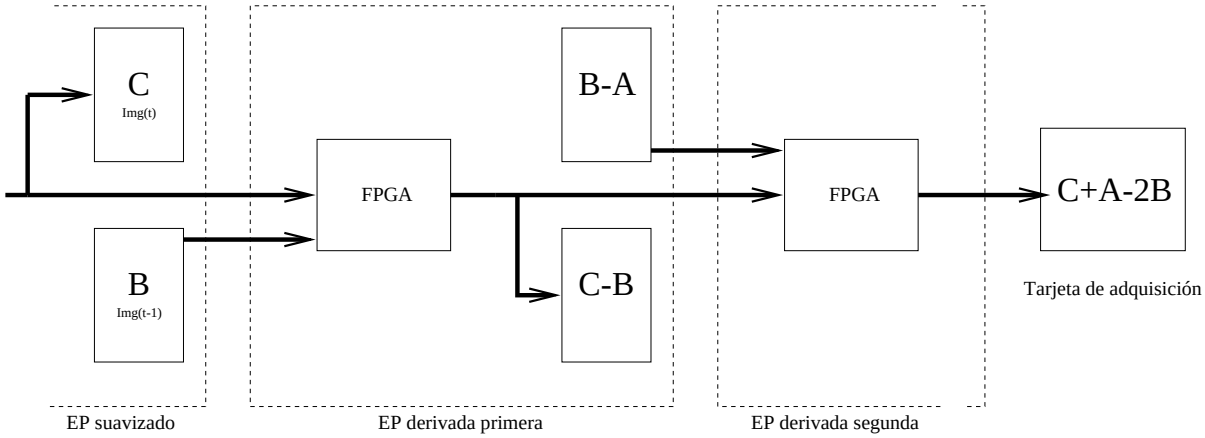


Figura 6.2: *Flujo de imágenes entre la etapa de calculo de la primera derivada y la etapa siguiente*

La etapa que realiza la resta de imágenes está calculando y escribiendo en su memoria de doble puerto la imagen $C - B$. Esta imagen, a la vez que se escribe en memoria de doble puerto, se suministra al EP siguiente del cauce para que calcule la derivada segunda. En el otro banco permanece almacenada la imagen $B - A$, con la que está operando el siguiente EP (cálculo de la derivada segunda).

El resultado de realizar la resta puede ser un número negativo, sin desbordamiento, que se representará como complemento a 2. La expresión analítica del byte resultado S_{ij} en función de los bytes a_{ij} de las imágenes en los instantes t y $t-1$ será la expresada en la ecuación (6.14).

$$S_{ij} = a_{ij}(t) - a_{ij}(t-1) \quad 128 \geq i \geq 1, 76 \geq j \geq 1 \tag{6.14}$$

El bucle de esta máquina de estados es de nuevo de 4 estados. En B se muestra el código VHDL de esta etapa, que es la más sencilla de este algoritmo.

6.4.3 Etapa de cálculo de la segunda derivada temporal y binarización

En esta etapa del algoritmo la entrada serán bytes de la imagen derivada primera y la salida serán los puntos marcados como poseedores de movimiento propio independientemente de la plataforma móvil. La entrada vendrá del EP descrito anteriormente y la salida tendrá como destino la tarjeta de adquisición, que la almacenará en memoria accesible por el robot. La plataforma móvil extraerá de los puntos, marcados como poseedores de movimiento propio, la información estadística necesaria (medias temporales

y espaciales) para planificar su trayectoria. En posteriores trabajos de investigación se estudiará la integración sensorial y la planificación dinámica de trayectorias utilizando estos datos. En el capítulo 9 se evaluarán las prestaciones de este algoritmo.

Como se ha justificado en la sección 6.3, no se puede cumplir exactamente la exigencia de la ecuación (6.8). La no perfección en el suavizado (ecuación (6.5)) y las aproximaciones en el movimiento (expresadas en las ecuaciones (6.6) y (6.12)) hacen que, en vez de exigir que la derivada temporal de la imagen sea simplemente distinta de cero, su valor absoluto supere un cierto margen umbral.

El valor de este margen umbral viene determinado por las condiciones de iluminación, las características de la escena y la *calidad* de las aproximaciones. La experimentación del capítulo 9 ha sido realizada con diversos márgenes umbral, automatizándose la elección del umbral en función de la velocidad de la plataforma móvil. La ecuación 6.15 expresa de forma analítica el resultado S_{ij} en función de los bytes a_{ij} de las imágenes en los instantes t y $t-1$.

$$\begin{cases} S_{ij} = 255 & \text{si } \mid a_{ij}(t) - a_{ij}(t-1) \mid \geq umbral \\ S_{ij} = 0 & \text{en otro caso} \end{cases} \qquad 128 \geq i \geq 1,\ 76 \geq j \geq 1 \quad (6.15)$$

En la figura 6.2 se muestra el flujo de información para calcular la derivada segunda sobre la secuencia de imágenes A, B y C, tomadas en los instantes $t-2$, $t-1$, y t. Esta imagen binarizada se guarda en la tarjeta de adquisición siguiendo el protocolo descrito en la sección 4.3. Si el resultado de calcular la derivada segunda no supera la ventana marcada por el valor umbral se marca como 0. Si lo supera, el byte pasa a valer un valor de gris (200), con lo que los puntos negros sobre fondo gris marcarán los puntos con movimiento independiente de la plataforma móvil.

La máquina de estados que realiza estas operaciones tiene un ciclo de 4 estados y el código VHDL que la describe se puede observar en el apéndice B

6.5 Simulación y estimación de prestaciones

A partir de los ficheros SCF con los retrasos de las señales generados por el compilador de VHDL de ALTERA se ha realizado la simulación de la temporización del algoritmo. Las FPGAs de los tres EPs de los que consta el algoritmo pueden funcionar con un periodo de reloj de 30 ns, pero debido al tiempo de acceso de las memorias de doble puerto (35 ns) y la velocidad del reloj de la tarjeta de adquisición PCI (30 ns), se ha optado por dividir la frecuencia de la tarjeta PCI por 2 y fijar un periodo de 60 ns. Estas velocidades no son muy altas ya que se han utilizado FPGAS de grado 3, memorias no

muy rápidas y el emplazado y rutado se ha realizado de manera totalmente automática con la herramienta MAX+PLUS II. Se ha evitado una optimización manual de la síntesis ya que se ha pretendido que el proceso de generación del código de las FPGAs sea lo más automático posible con la intención de integrarlo en un marco más amplio con técnicas de codiseño. En cualquier caso, es interesante realizar una estimación de las prestaciones evaluada en ciclos de reloj necesarios por byte de la imagen log-polar.

En la figura 6.3 se realiza una simulación con imágenes de una longitud hipotética de 2 bits para poder observar el intercambio en el flujo de imágenes. El primer EP realiza la convolución de los datos que le suministra la tarjeta de adquisición a través de $Dato\_in\_0$. En el fragmento simulado estos datos, una vez realizada la convolución con la fila anterior, transmite un valor de 25 y 26 en la escala de grises al módulo que calcula la derivada primera.

Se puede observar como en el cambio de imagen se intercambia el origen de la imagen de la memoria de doble puerto cambiando las señales OE_2. Análogamente cambiando las señales CE_1 se permuta la escritura por parte del EP que realiza el suavizado a su memoria de doble puerto. Se puede comprobar como no se efectúa una lectura y escritura simultánea en el mismo circuito de memoria.

De la misma manera opera la etapa que calcula la derivada segunda y luego realiza la binarización de la imagen. De nuevo se realiza un acceso a la memoria de doble puerto del módulo anterior con las señales OE_3 y generando la dirección adecuada.

En este caso la máquina de estados de esta etapa permanece paralizada en $Sini$ (estado de configuración inicial) hasta que el módulo anterior activa por primera vez $Dato\_listo\_2$. Esto ocurre cuando se tiene disponible el primer byte de la segunda imagen, ya que el módulo siguiente para calcular la derivada segunda, necesitará que ya esté una imagen previamente almacenada en memoria. A partir de esta segunda imagen que pasa por el módulo RESTA, sale una imagen binarizada por cada imagen que entra en el cauce.

La figura 6.4 muestra esta latencia inicial de dos imágenes log-polares completas para que el cauce empiece a suministrar imágenes binarizadas correctas al empezar a procesar la tercera imagen. En la simulación se comprueba cómo se produce un dato correcto cada 7 ciclos de reloj, ya que la tarjeta de adquisición lo suministra cada 7 ciclos de reloj. La tarjeta de adquisición de imágenes es capaz de suministrar un dato nuevo cada 4 ciclos de reloj de 30 ns, lo que supone un dato nuevo cada 2 ciclos de los EPs. Las etapas de este algoritmo tienen un bucle de 4 ciclos de reloj, que sumado a los dos ciclos de la comunicación/reconocimiento de dato, suponen un velocidad máxima de 6 ciclos de reloj.

En este algoritmo de detección de movimiento el cauce suministrará un punto de la

Operación por byte		Coste en ciclos
Acceso a dato imagen nueva		5 ciclos
Suavizado (máscara de 3x2)		
	Suma entera nueva	1 ciclo
	Accesos a caché	$1 * 5 = 5$ ciclos
	Desplazamiento	1 ciclo
Acceso a imágenes de caché	(Resta)	$1 * 2 = 2$ ciclos
Restas enteras		$1 * 3 = 3$ ciclos
Comparación con umbral		1 ciclo
Escritura		5 ciclos
Total		23 ciclos

Tabla 6.1: *Costes estimados del algoritmo de detección de movimiento en un PENTIUM PRO*

imagen cada 6 ciclos de $60ns$. Teniendo en cuenta que una imagen log-polar completa son 9.728 puntos, lo que da un tiempo de proceso de 3.5 ms por imagen, se podrán procesar hasta 285 imágenes por segundo.

En realidad, la limitación de la velocidad de proceso vendrá marcada por la tarjeta de adquisición, que puede adquirir hasta un máximo de 200 imágenes por segundo, pero por cuestiones de calidad de la imagen se limitará a 100 imágenes por segundo.

La latencia del algoritmo es de 2 imágenes, con lo que teóricamente la limitación impuesta por la velocidad del cauce seria de 7 ms. Al ser la tarjeta de adquisición la que limita la velocidad de procesado, la latencia será de 20 ms.

Realizar el mismo algoritmo con un procesador escalar es posible, con un mayor número de ciclos. La tabla 6.1 da cuenta del coste aproximado en número de ciclos que serian necesarios para realizar las mismas operaciones con un procesador Pentium Pro.

De la tabla se desprende que el número de ciclos es 4 veces mayor, con lo que teniendo un procesador que funcione con una frecuencia de reloj 4 veces más rápida que la del cauce segmentado, las prestaciones serían las mismas.

El cauce segmentado, en este algoritmo concreto y con la implementación realizada se podría hacer funcionar a 33 MHz cambiando las memorias de doble puerto de 35 ns por otras equivalentes más rápidas. De la misma manera, la utilización de FPGAs más rápidas incrementaría significativamente el rendimiento. En cualquier caso en la actualidad se dispone de procesadores que funcionan a frecuencias mucho mayores de 4x33 MHz=133 MHz. La justificación del uso de esta arquitectura reconfigurable para la realización de este algoritmo viene en la escalabilidad de la arquitectura y su

reconfigurabilidad. Solamente añadiendo EPs se puede aumentar la profundidad del cauce, disminuir los ciclos de reloj por etapa, y aumentar la velocidad de proceso. De hecho se pueden diseñar EPs con FPGAs más rápidas y memorias más rápidas que mejoren las prestaciones del cauce sin cambios en la arquitectura.

6.6 Conclusiones

Se ha diseñado, siguiendo la metodología propuesta en el capítulo 5 un algoritmo de detección de movimiento independientemente del movimiento de la cámara en el cauce reconfigurable. La localidad de todas las operaciones hace que este algoritmo sea totalmente implementable mediante la partición hardware. Queda a la partición software la interpretación del mapa binario con los puntos en movimiento marcados en negro, su agrupamiento y posterior modificación de la trayectoria. Esta última tarea está íntimamente relacionada con la programación de tareas del robot y la organización de su comportamiento por capas.

La conversión a coordenadas log-polares, junto con la aproximación de la ecuación (6.12) hace que el movimiento descartado sea el producido en la imagen cuando la cámara se mueve en el sentido del eje óptico. De esta manera se puede aislar el movimiento propio de los objetos del desplazamiento de la imagen inducido por la translación de la cámara.

El formalismo log-polar, además de auto-descartar este movimiento, reduce significativamente la cantidad de datos a procesar. De una imagen típica de $256*256 = 65.536$ puntos se pasa a sólo 9.728 puntos, con una resolución similar en el centro de la imagen.

El algoritmo segmentado, adaptado al formalismo log-polar, se ha diseñado en un cauce de 3 etapas. Las 3 etapas tienen como entrada y salida una imagen log-polar. La primera etapa realiza una convolución con una ventana de 6 puntos y unos coeficientes unitarios para evitar la utilización de multiplicadores. La división para calcular el valor medio se hace de nuevo de forma aproximada al truncar el resultado manteniendo sólo los 7 bits más significativos. De esta manera la etapa de suavizado es una sencilla máquina con un ciclo de 4 estados. Para realizar la convolución de manera eficiente se ha implementado, con la memoria local al EP, un registro de desplazamiento del tamaño de una fila (128 puntos).

Las etapas siguientes, (que calculan la derivada primera y la derivada segunda + binarización), no utilizan la memoria local, pero sí que acceden a la memoria de doble puerto del EP anterior. De esta manera se calcula de forma rápida la derivada primera, y a partir de ésta, la derivada segunda. Ambas etapas de nuevo tienen un bucle de 4 ciclos por byte, quedando un cauce segmentado con etapas muy equilibradas.

Simulando de forma exhaustiva la temporización del algoritmo, se pueden obtener imágenes procesadas con un *ratio* teórico de 285 imágenes por segundo. Esta velocidad está limitada por la velocidad a la que la tarjeta de adquisición suministre imágenes al cauce que será de 100 imágenes por segundo.

Comparando el número de ciclos (6) necesarios para procesar un punto de la imagen en el cauce reconfigurable, con el número de ciclos de un procesador escalar, se obtiene un mayor número de ciclos para este último. Puede ser más favorable utilizar para este caso concreto un procesador con una frecuencia al menos 4 veces más rápida, pero se pierden todas las ventajas de la arquitectura propuesta. La escalabilidad y la velocidad no viene limitada por la arquitectura. Siempre se puede aumentar el número de EPs o etapas, aumentando la velocidad. Así mismo siempre se puede realizar un diseño de un EP con una FPGA y con memorias más rápidas. Además, mientras el módulo reconfigurable realiza este algoritmo, libera al procesador central de una tarea muy costosa computacionalmente.

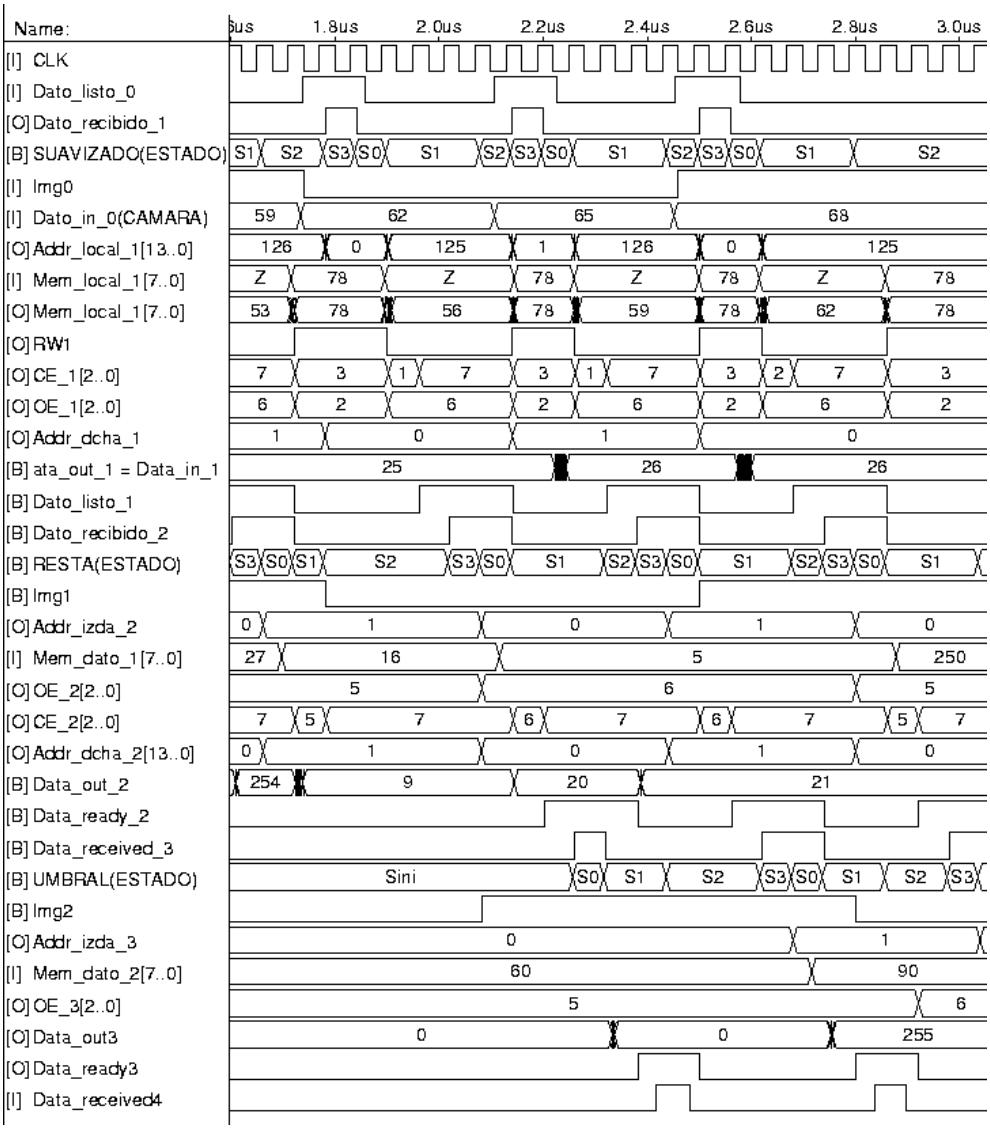


Figura 6.3: *Simulación del algoritmo de detección de movimiento con imágenes de 2 bits*

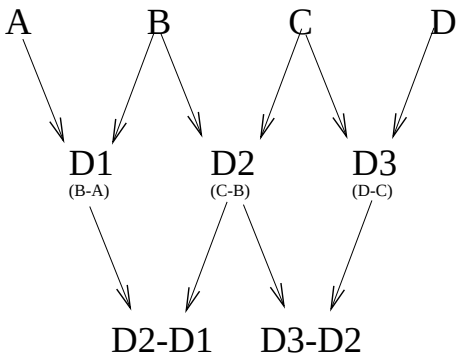


Figura 6.4: *Orden en el que se van procesando y generando las imágenes*

Capítulo 7

Diseño del algoritmo de cálculo de tiempo al impacto

7.1 Introducción

La estimación del tiempo al impacto del observador con un objeto, es de vital importancia para los seres vivos y lo puede ser también para una plataforma móvil navegando en entornos no estructurados. Es interesante dotar a un robot de la capacidad de prever colisiones y modificar la trayectoria para poder evitarlas, mediante el cálculo del tiempo al impacto de los diversos objetos de la escena, a partir de imágenes tomadas desde el mismo robot.

Las ventajas del cálculo del tiempo al impacto mediante el formalismo log-polar han sido estudiadas y desarrolladas por Tistarelli y Sandini en [TS91a] y [TS91b]. Además de la reducción selectiva de información, el formalismo log-polar simplifica el cálculo del tiempo al impacto cuando el centro de expansión coincide con el centro del sensor log-polar. Esta condición es bastante restrictiva, pero puede conseguirse si la cámara log-polar está fijada en la parte frontal del vehículo y se realiza un movimiento rectilíneo uniforme para poder aplicar este algoritmo.

En el presente capítulo se desarrolla el cálculo del tiempo al impacto log-polar en la arquitectura propuesta. Se diseña en el módulo reconfigurable o partición hardware las etapas necesarias, siguiendo la metodología propuesta en el capítulo 5. La partición software interpretará y analizará el mapa de tiempos al impacto.

La posterior simulación de prestaciones del algoritmo muestra cómo se pueden procesar imágenes en la etapa hardware a una velocidad media del orden de 100 imágenes por segundo. La velocidad de producción de mapas de tiempo al impacto dependerá

del número de divisiones que haya que realizar, y esto dependerá de la calidad de las aproximaciones realizadas (suavizado y movimiento centrado).

7.2 El algoritmo en coordenadas log-polares

En la sección 3.2.2 se describe, a partir de la figura 3.5 cual es la expresión del crecimiento de un objeto en el plano imagen si el movimiento es a lo largo del eje focal. Derivando y reescribiendo la ecuación (3.5) de manera adecuada se obtiene la expresión del tiempo al impacto τ, que se expresa en la ecuación (7.1). En el capítulo 8 se analiza el error que se introduce en esta expresión cuando el eje óptico no coincide exactamente con la dirección de movimiento. Este error se parametriza en función del ángulo β, y de los parámetros de la escena.

$$\tau = \frac{z(t)}{W(t)} = \frac{r(t)}{V(t)} \tag{7.1}$$

Tal como se ha justificado en la sección 3.2.2, Realizar los cálculos de la ecuación (7.1) en el plano cartesiano no es sencillo. Se debe calcular la velocidad de cada punto en el plano imagen, es decir se debe calcular el flujo óptico en cada punto, cuyos costes ralentizan este método. Sin embargo la utilización de las coordenadas log-polares simplifica la expresión, obteniéndose para la retina la ecuación:

$$\tau = K \frac{-\frac{\partial I}{\partial \xi}}{\frac{\partial I}{\partial t}} \tag{7.2}$$

Donde K es una constante que depende de los parámetros concretos de la transformación log-polar que realiza el sensor. En el caso del sensor log-polar utilizado esta constante tiene un valor de $K = 1/B \approx 20$. La expresión para la fóvea es algo más compleja, incluyendo la dependencia radial:

$$\tau = \frac{-\xi \frac{\partial I}{\partial \xi}}{\frac{\partial I}{\partial t}} \tag{7.3}$$

Estas expresiones son más sencillas de calcular que sus equivalentes en coordenadas cartesianas al desaparecer el calculo del flujo óptico en dos coordenadas. A partir de estas expresiones se puede diseñar un algoritmo de cálculo de tiempo al impacto que, por simplicidad, se va a realizar sólo en la retina.

7.3 Etapas en el cauce reconfigurable

El calculo del tiempo al impacto, a partir de la ecuación log-polar (7.2), puede ser dividido en las siguientes etapas [BPBP98]:

- **Adquisición de las imágenes** $E(\xi, \theta, t)$. Etapa que realizará la tarjeta de adquisición de imágenes en los instantes indicados por el procesador del robot.
- **Suavizado de la imagen.** La eliminación de picos en la imagen (funciones escalón), mejora el cálculo de las etapas posteriores al cumplirse la condición de continuidad.
- **Cálculo de la derivada espacial.** El cálculo de $\frac{dI}{d\xi}$ se va a realizar restando valores de gris de los puntos de filas consecutivas.
- **Cálculo de la primera derivada temporal.** El cálculo de $\frac{dI}{dt}$ se realizará siguiendo la técnica habitual ya descrita para el cálculo de derivadas temporales.
- **Cálculo de τ.** Para ello hay que realizar la división del gradiente entre la derivada temporal.

En este algoritmo la entrada de la primera etapa son imágenes log-polares. La salida de la primera etapa también es una imagen log-polar suavizada. Sin embargo, la salida de la segunda etapa es un mapa log-polar con 2 cantidades por punto de entrada: la derivada temporal y la derivada espacial. La última etapa produce un mapa de valores proporcionales a τ que tampoco es una imagen estrictamente hablando. De esta manera se genera un mapa de tiempos al impacto, cuya interpretación y extracción de valores estadísticos (operaciones globales) se dejan a la partición software.

Cabe hacer notar que la etapa de suavizado, diseñada con anterioridad en la sección 6.4.1, va a ser utilizada de nuevo sin ningún cambio adicional. De esta manera se prueba la validez de la modularidad de la arquitectura. Una vez diseñado un EP, éste se puede reutilizar para diferentes algoritmos, simplemente conociendo las estructuras de datos de la entrada y salida de la etapa.

Las etapas de cálculo de la derivada espacial y temporal pueden agruparse en una sola etapa dentro de un solo EP. Esto es posible al no haber conflicto entre los recursos utilizados (memoria local y de doble puerto), y no existir dependencia de datos entre estos valores.

Una vez obtenida la derivada espacial y temporal, se debe realizar una división entre estos valores. Esta etapa, al ser muy costosa computacionalmente, también se realiza mediante un EP en el cauce segmentado. De nuevo por simplicidad, y para tratar de encajar toda esta etapa en un solo EP, se ha realizado un algoritmo de división entera. De esta manera se obtendrá un mapa de bytes proporcionales al tiempo al impacto.

Finalmente, como tarea para el software, se escalarán adecuadamente los valores numéricos en función de las características del sensor, se realizarán medias entre los puntos del mapa, se agruparán puntos para formar objetos y, en definitiva, se interpretará el mapa de bytes extrayendo la información adecuada.

Las etapas que han sido diseñadas como EPs en el cauce reconfigurable son las que a continuación se describen con mayor detalle.

7.3.1 Etapa de cálculo del gradiente radial y primera derivada

El cálculo de la derivada espacial o gradiente radial, y el cálculo de la primera derivada puede realizarse en paralelo en un solo EP al no haber conflicto en los recursos necesarios.

De nuevo la estructura de entrada será una imagen log-polar suavizada, y la estructura de salida serán 2 bytes consecutivos por punto de la imagen log-polar. Primero se transmitirá la derivada temporal y después el gradiente correspondiente al punto *presente* de la imagen procesada.

La derivada temporal se calcula de la manera habitual, ya descrita en el capítulo 6. El módulo de suavizado de la imagen guardará una imagen inicial en la memoria intermedia de doble puerto. Al realizar el suavizado de la segunda imagen, se validará cada dato con el protocolo de transmisión de datos habitual, accediendo simultáneamente a esta segunda imagen *presente* y a la primera imagen guardada en memoria intermedia. El cambio entre imágenes se realiza de la misma manera.

El cálculo de la derivada espacial o gradiente radial se realiza restando 2 bytes de la misma columna, pero de líneas consecutivas. Para ello es necesario almacenar en memoria local la última línea transmitida, y restar a cada *pixel* presente el *pixel* del anillo inmediatamente inferior, ya recuperado de la memoria local. Posteriormente se almacenará este valor en memoria local para que esté disponible este dato para el cálculo del gradiente de la siguiente línea. En realidad es como si se implementa una cola FIFO de 128 registros para realizar el desplazamiento de una línea. La utilización de recursos en el algoritmo de cálculo de tiempo al impacto se muestra en la figura 7.1. Una vez se recibe un *pixel* éste se almacena en una dirección llevada por un contador. Posteriormente, se recupera el *pixel* apuntado por la dirección a la que apunta el puntero menos 128. De esta manera se recupera el *pixel* de la misma columna, pero de la fila inmediatamente inferior, y se puede calcular el gradiente radial.

El gradiente de la primera linea resulta una cantidad absurda ya que se efectúa la resta con un valor de memoria que no corresponde a la línea anterior. Este efecto de borde, al igual que en el EP de suavizado, es despreciable al corresponder toda la

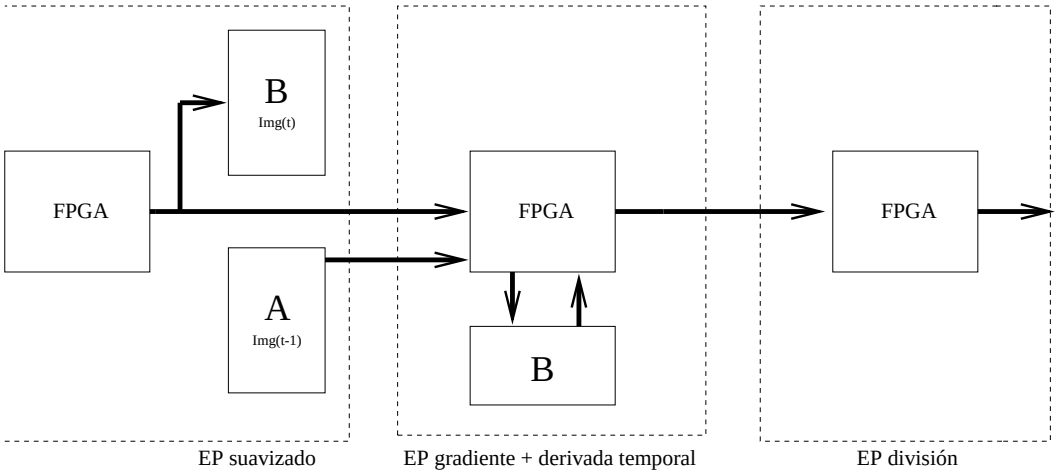


Figura 7.1: *Utilización de recursos de los EPs en el calculo del tiempo al impacto*

primera línea a un solo punto.

Con estas expresiones la derivada temporal del punto a_ij sería S_{ij}^t y la derivada espacial sería S_{ij}^e, cuyas expresiones analíticas vendrán dadas por la ecuación (7.4).

$$\begin{cases} S_{ij}^t = a_{ij}(t) - a_{ij}(t-1) & 128 \geq i \geq 1,\ 76 \geq j \geq 1 \\ S_{ij}^e = a_{ij}(t) - a_{ij-1}(t) & 128 \geq i \geq 1,\ 76 \geq j \geq 1 \end{cases} \tag{7.4}$$

Al ser los números suministrados por el módulo anterior de 7 bits, el resultado es 8 bits en complemento a 2 sin desbordamiento. Después, en la etapa siguiente, se tendrá en cuenta el signo del gradiente y de la derivada temporal para efectuar o no la operación de división.

Las operaciones de comunicación con el EP anterior, comunicación con el EP siguiente, acceso a memoria local y memoria de doble puerto se han realizado con una máquina de estados con un bucle de 4 ciclos. En el apéndice B se adjunta el código VHDL del EP que realiza estas operaciones.

7.3.2 Etapa de cálculo de la división entera

A partir de la derivada espacial y temporal es posible obtener una valor proporcional al tiempo al impacto τ dividiendo ambos valores, tal como indica la ecuación 7.3. Para realizar este cálculo se ha implementado en un EP el algoritmo de división entera sin restauración.

Han sido estudiadas otras arquitecturas de división entera más rápidas, como divisiones enteras de *radix* o base mayor que 2 o matrices de división enteras. Sin embargo,

la complejidad de la lógica necesaria para realizar las conversiones de base, o para implementar las matrices de elementos de proceso, han hecho desestimar la implementación de una algoritmo más rápido ya que se ha intentado encajar toda la etapa de división en un solo EP.

El algoritmo de división entera sin restauración, ofrece una velocidad de procesado suficiente, tal como se mostrará en la sección 7.4. Se puede encontrar un estado del arte actualizado de la aritmética digital de división en [OF97].

El EP recibe en primer lugar la derivada temporal y a continuación la derivada espacial del módulo anterior, siguiendo el protocolo de intercambio de datos habitual. Después se realizan una serie de operaciones para comprobar que es posible realizar la división. Si el divisor o el numerador son cero, o el denominador es mayor que el numerador no se realiza la división y el resultado será un número que marcará como incorrecto el valor de *tau* calculado para ese punto. De la misma manera, si ambos números son de signo contrario el resultado sería negativo, lo cual no tiene sentido para objetos que se están acercando, por tanto tampoco se realiza la división, siendo el resultado cero de nuevo. El resultado de esta división entera será siempre un número positivo entre 0 y 127 y la expresión analítica del resultado de esta etapa para el punto $S(i,j)$viene expresado en la ecuación 7.5 en función de la correspondiente derivada espacial a_{ij}^e, y de la derivada temporal a_{ij}^t.

$$\begin{cases} S_{ij} = \lceil \frac{|a_{ij}^e|}{|a_{ij}^t|} \rceil & a_{ij}^t \neq 0 \text{ y signo}(a_{ij}^t) \neq \text{signo}(a_{ij}^e) \\ S_{ij} = 0 & \text{en otro caso} \end{cases} \qquad 128 \geq i \geq 1,\ 76 \geq j \geq 1 \quad (7.5)$$

Una vez obtenido el mapa de tiempos al impacto, se realizará por software el escalado y agrupamiento de estos datos, obteniendo de esta manera una segmentación de la imagen en objetos con diferentes τ. Resultados experimentales de la evaluación de este algoritmo se muestran en la sección 10.

7.4 Simulación y estimación de prestaciones

Se han simulado separadamente las etapas del cauce segmentado que implementan el algoritmo de cálculo del tiempo al impacto. En la figura 7.2 se muestra una simulación global del flujo de datos a través del cauce segmentado en este algoritmo.

En la figura 7.2 se observa como el segundo EP del cauce suministra los valores numéricos 243 y 123 validándolos mediante la señal *dato_listo_2*. El primer valor (negativo en complemento a 2) sería la representación del 13, correspondiente al denominador. El siguiente valor es positivo y corresponderá al numerador.

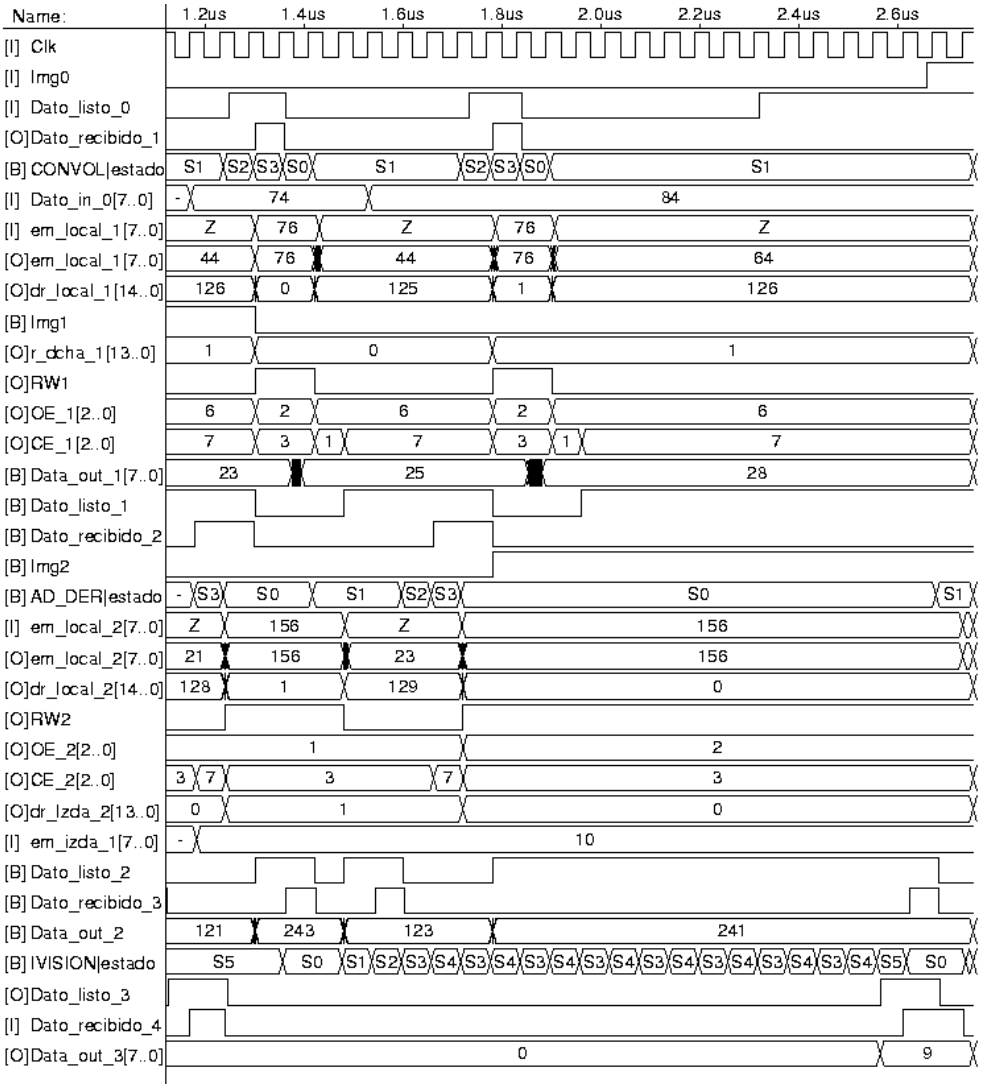


Figura 7.2: *Simulación del cauce en el algoritmo de cálculo de tiempo al impacto*

Al cumplirse que ambos valores son de signo contrario, el módulo del numerador es mayor que el módulo del denominador, y que el denominador es distinto de cero, entonces se efectúa la división sin restauración. El número de ciclos de este algoritmo es de $2 * n$ donde n es el número de bits utilizado en la representación.

Tras $2 * 8 = 16$ ciclos ya se dispone del resultado correcto de la división y se valida mediante *dato_listo_3*. En este caso el resultado será de $\lceil \frac{123}{13} \rceil = 9$. Posteriormente se realizará un escalado de este valor, se agrupará con los valores vecinos similares, y se realizará una media, cuyo resultado estadístico será más preciso que realizar un cálculo en un solo punto mediante una división entera.

En este caso la división sí que se realiza, pero en otros casos, debido a que el resultado no tendría sentido, no se realiza, siendo el tiempo de proceso del EP que realiza la

división variable. La flexibilidad en la transmisión/recepción de datos permite evitar retrasos inútiles. De esta manera, sólo quedan los bloqueos inherentes a los cauces segmentados, debidos a la disponibilidad de los datos en cada etapa.

Con el cauce configurado para realizar el algoritmo de cálculo de τ, en el peor caso, se puede producir un byte del mapa del tiempo al impacto cada 21 ciclos de reloj. En el caso en que no haya que realizar la división se produce un dato procesado cada 8 ciclos de reloj.

Al ser las imágenes de 9.728 puntos, un mapa de tiempos al impacto tardará, en el peor caso, 0.012 segundos (siendo el periodo del reloj de 60 ns). Esto da un resultado de 81 imágenes por segundo.

En el caso que no se realice la división de ningún punto se pueden obtener más de 200 imágenes por segundo. Suponiendo que mayoritariamente sí que se realiza la etapa más lenta de división en el último EP, se prevé una velocidad de proceso de unas 100 imágenes por segundo, al igual que la velocidad de suministro de las imágenes por la etapa de adquisición. En el capítulo 10 se evalúan, de forma experimental, las prestaciones de este algoritmo.

De nuevo hay que comparar el coste temporal de implementar este algoritmo mediante un procesador escalar de propósito general. En la tabla 7.1 se estiman los costes de las mismas operaciones que realiza el cauce reconfigurable para calcular el valor proporcional a τ. Para tener en cuenta el coste de la división se ha considerado la más rápida (simple precisión) al no existir una unidad de división entera.

El coste es el doble en número de ciclos, por tanto de nuevo un procesador que funcione a más del doble de frecuencia que el cauce reconfigurable efectuará las operaciones más rápidamente.

La justificación del uso de esta arquitectura reconfigurable para la realización de este algoritmo viene de nuevo por la escalabilidad de la arquitectura. Añadiendo más EPs se puede aumentar la profundidad del cauce, disminuyendo los ciclos de reloj por etapa, y aumentando la velocidad de proceso.

En este algoritmo claramente el cuello de botella es la etapa de división entera. Se puede evitar este cuello de botella mediante el diseño de un EP que realice esta misma funcionalidad con menos ciclos de reloj. Este diseño se realizaría con una FPGA con más celdas lógicas, pero siguiendo la arquitectura del módulo reconfigurable. El cauce reconfigurable permite esta ampliabilidad y mejora de los EPs, en función del abaratamiento y mejora de prestaciones de la lógica reconfigurable.

Operación por byte		Coste en ciclos
Acceso a dato imagen nueva		5 ciclos
Suavizado (máscara de 3x2)		
	Suma entera nueva	1 ciclo
	Accesos a caché	$1 * 5 = 5$ ciclos
	Desplazamiento	1 ciclo
Acceso a imágenes de caché		$1 * 2 = 2$ ciclos
Restas enteras		$1 * 2 = 2$ ciclos
División (Simple precisión)		17 ciclos
Escritura		5 ciclos
Total		38 ciclos

Tabla 7.1: *Costes estimados del algoritmo de cálculo de tiempo al impacto en un PENTIUM PRO*

7.5 Conclusiones

La capacidad de calcular el tiempo al impacto es una funcionalidad que puede resulta interesante para un vehículo autónomo. El formalismo log-polar, además de reducir la cantidad de datos a procesar, presenta una serie de importantes ventajas para el cálculo del tiempo al impacto. La reducción de dos variables de la velocidad a la sola velocidad radial, en el caso de movimiento a lo largo del eje óptico, permite evitar todos los problemas de aproximaciones y cálculos complejos que lleva implícito el cálculo del flujo óptico.

El cálculo del tiempo al impacto es de nuevo un algoritmo diferencial que puede ser implementado de forma ventajosa en la arquitectura reconfigurable. Se reutiliza el diseño de la etapa de suavizado de la imagen de la sección 6. Se utilizan las memorias de doble puerto intermedias para realizar el cálculo de las derivadas temporales, y la memoria local para calcular el gradiente radial en la siguiente etapa. Este EP suministra al EP siguiente 2 bytes por cada *pixel* recibido: la derivada espacial y la derivada temporal. Posteriormente ambos valores se dividen de forma entera, siendo esta última etapa el cuello de botella de la partición hardware de este algoritmo.

A partir del mapa de cantidades proporcionales a τ suministrado por el cauce reconfigurable, se deja a la etapa *software* el agrupamiento de estos puntos en objetos que se mueven, y la extracción de medias y otros parámetros estadísticos.

Las prestaciones de la implementación realizadas son claramente mejorables utilizando un EP con más recursos, en el que sea posible implementar una matriz de división entera que acelere el algoritmo. Queda para el trabajo futuro, en función de la rela-

ción precio/prestaciones de las FPGAs futuras, el desarrollo de EPs más potentes que muestren de manera más directa las ventajas de la arquitectura propuesta.

En cualquier caso, y tal como se mostrará en el capítulo 10, unos resultados precisos en este algoritmo requerirán de un análisis estadístico de los datos que ralentizará el cálculo del tiempo al impacto, siendo el cuello de botella del algoritmo la partición *software*.

Capítulo 8

Parametrización y limitaciones de los algoritmos

8.1 Introducción

En los capítulos 6 y 7, se han diseñado dos algoritmos distintos interesantes para navegación robótica. Estos algoritmos prueban la utilidad y reconfigurabilidad de la arquitectura propuesta en el capítulo 4, y la metodología de diseño propuesta en el capítulo 5.

En ambos algoritmos se hacen una serie de supuestos que cabe analizar para prever los errores que se producirán en caso de que no se cumplan. Las aproximaciones realizadas en estos algoritmos introducen una diferencia o error entre el valor teórico y el valor real obtenido.

En el algoritmo de detección de movimiento, diseñado expresamente en el capítulo 6 para su utilización en el módulo reconfigurable, se realizan varias aproximaciones y supuestos. La aproximación de la similitud entre la ecuación (6.10), solución de la ecuación diferencial (6.9), con el crecimiento en el plano imagen debido al acercamiento de un cuerpo expresado en la ecuación (6.11), introduce un error que debe ser evaluado. Para ello es interesante encontrar expresiones analíticas que, en función de ciertos parámetros, permitan evaluar la magnitud del error y minimizarlo.

De la misma manera, tanto en este algoritmo como en el algoritmo de cálculo de tiempo al impacto, se parte de la base de que el eje óptico coincide con la dirección de movimiento. Esta condición, combinada con la estructura polar del sensor, hace que se simplifiquen los algoritmos implementados. Condición que se puede conseguir debido a que el vehículo autónomo posee una cabeza móvil orientable. De nuevo cabe

la posibilidad de que este alineamiento no sea perfecto y haya una divergencia entre ambos ejes de un cierto ángulo β. Es también interesante por tanto estudiar cual será el error introducido en los algoritmos cuando el alineamiento no es perfecto.

Finalmente se ha planteado la cuestión de cuáles deben ser las condiciones dinámicas de la escena para que un cambio externo produzca un cambio detectable en el sensor. La relación entre la velocidad externa de los objetos, la distancia inicial, y las características geométricas del sensor deben ser tenidas en cuenta para capturar las imágenes con un intervalo de tiempo mínimo. Estos conceptos se relacionan y se analizan en la parte final de este capítulo.

8.2 Errores en el algoritmo de detección de movimiento

En el capítulo 6 se diseña un algoritmo de detección de movimiento, independiente del movimiento propio de la cámara, siempre que se cumplan una serie de condiciones que a continuación se resumen.

1. El eje óptico debe coincidir con la dirección de movimiento.
2. El movimiento debe ser suave, es decir $\forall\ (\xi, \theta) \in T(\Sigma)$:

$$\frac{d^2\xi}{dt^2} = \frac{d^2\theta}{dt^2} = 0 \tag{8.1}$$

3. La imagen no debe tener picos en sus niveles de gris, es decir $\forall\ (\xi, \theta) \in T(\Sigma)$:

$$\frac{\partial^2 E}{\partial \xi^2} = \frac{\partial^2 E}{\partial \theta^2} = 0 \tag{8.2}$$

La tercera condición exigiría la construcción de un filtro óptimo para que se cumpliese en todos los puntos de la imagen [CN93]. Este complicado proceso se puede evitar mediante un suavizado que, aunque no haga que se cumpla la ecuación (8.2) totalmente, experimentalmente funciona tras el ajuste del valor umbral para la binarización [BDPP97a].

La primera aproximación requiere un estudio un poco más detallado para determinar la influencia de un posible ángulo β entre el eje de movimiento y el eje óptico. La no coincidencia de la ecuación que expresa el crecimiento de un objeto en el plano imagen, con la solución de la ecuación diferencial (8.1) requiere también un análisis más cuidadoso.

8.2.1 Divergencia entre el eje óptico y el vector de movimiento

De la figura 8.1 se puede extraer la ecuación (8.3), que expresa el crecimiento en módulo de un objeto r' con unas coordenadas (x', y', z') en el plano imagen (x, y). Adicionalmente, el movimiento relativo entre el objeto y el sensor es un vector $\vec{v}$ de módulo constante v_0 que, estando contenido en el plano (x, z), forma un ángulo constante β con el eje óptico.

Se ha considerado este caso particular teniendo en cuenta que la plataforma móvil se mueve sobre el plano horizontal, al igual que el resto de objetos. En este caso se cumple que

$$r = \frac{r'F}{Z - F} = F\frac{\sqrt{(x'_0 + v_0 t sen\beta)^2 + y'^2_0}}{z_0 - F - v_0 t cos\beta} \tag{8.3}$$

donde la posición inicial del extremo del vector es (x'_0, y'_0, z_0). Claramente las dimensiones en el plano imagen (x, y) dependen de la posición inicial, del módulo de la velocidad v_0, y del ángulo β entre ambos ejes. Cuando $\beta = 0$ la expresión se simplifica y se obtiene la ecuación (6.11), ecuación que se ha aproximado a la solución de la ecuación (8.1).

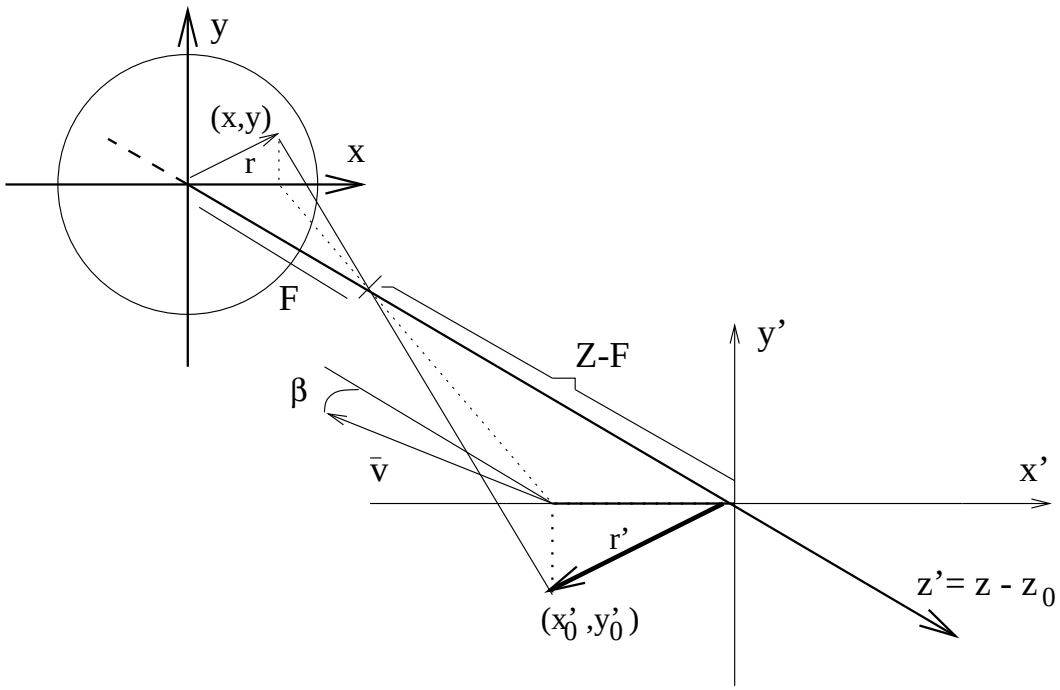


Figura 8.1: *Crecimiento de un objeto, con movimiento no paralelo, en el plano imagen*

Se puede expresar la diferencia Δr entre el nuevo valor de r' calculado en la ecuación (8.3) que tiene en cuenta el posible ángulo β, y la ecuación (6.11) que no lo tiene en

cuenta. Esta diferencia Δr será función del tiempo t, la distancia del objeto al sensor z_0, la velocidad v_0, y el ángulo β. A partir de este valor Δr, se puede calcular la variación porcentual $100\frac{\Delta r}{r}$ debido a que el vector de movimiento y el eje óptico ya no son paralelos.

En las figuras 8.2 y 8.3 se muestra la variación porcentual $\frac{100\Delta r}{r}$ o error del módulo de r en función de β, v_0 y z_0. En cada una de las tres gráficas se han fijado dos de estos tres parámetros y se ha estudiado la variación del error en función del otro parámetro y del tiempo. El tiempo t es el que transcurre entre la primera imagen ($t = 0$), y la siguiente imagen capturada por la tarjeta de adquisición.

En las tres gráficas se ha supuesto un objeto de dimensiones originales $r' = 0.5\ m$ y una focal $F = 50\ mm$.

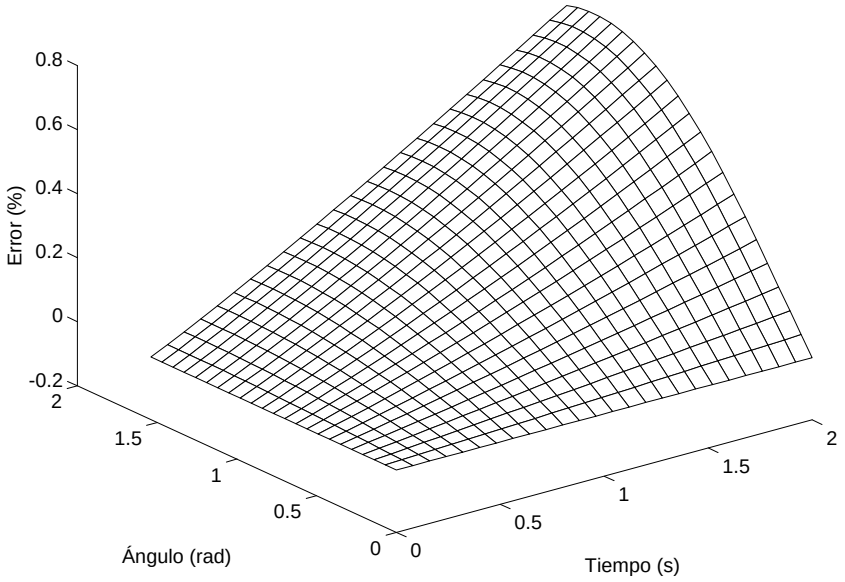
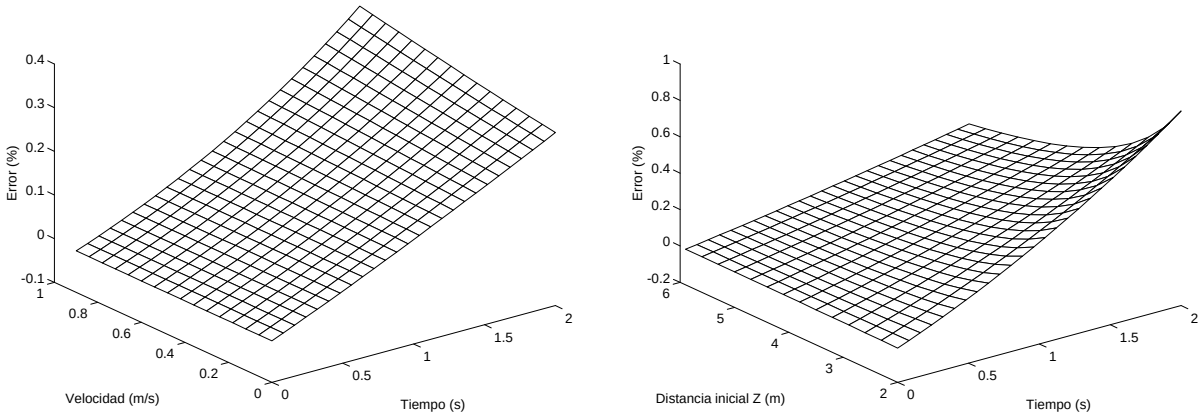


Figura 8.2: *Evolución del error porcentual de r en el plano imagen en función del ángulo y el tiempo*

En la figura 8.2 se han fijado los valores de la velocidad $v_0 = 0.25\ m/s$, y la distancia inicial $z_0 = 5\ m$, representándose la variación del error en función del ángulo β y del tiempo. En esta gráfica se obtiene el comportamiento esperado. Para $\beta = 0\ rad$ el error es 0 y no varía con t. Dado un cierto ángulo β el error crece con el tiempo, y dado un cierto instante fijo t el error crece con el ángulo, llegando a ser máximo con $\beta = \pi/2\ rad$. Con un ángulo de unos pocos grados el error, debido a esta divergencia, es tolerable no llegando a valores del 1%.

En la figura 8.3(a), se ha supuesto un ángulo de desviación constante $\beta = \pi/8\ rad$, y se ha representado la variación del error en función del módulo de la velocidad v_0 y el tiempo. De nuevo los resultados son los esperados. Para velocidades pequeñas el error



(a) (b)

Figura 8.3: *Influencia de la velocidad (a) y la distancia (b) en el error*

en las dimensiones crece linealmente de forma lenta con el tiempo. Para velocidades relativas más altas el crecimiento del error empieza a ser cuadrático con el tiempo.

Por último en la figura 8.3(b), se han fijado los valores de $\beta = \pi/8\, rad$, y de $v_0 = 0.25$ m/s, dejando el error en función de la distancia inicial z. En este caso se observa como el error es menor cuanto más alejado está el cuerpo, tal como se podía prever. Si un cuerpo está muy alejado de la cámara apenas se apreciarán las variaciones de su imagen en el sensor.

Como regla general se observa como el error crece con el intervalo de captura entre las imágenes. Es interesante entonces, reducir este intervalo y adquirir imágenes a la máxima velocidad posible. En la sección 8.4 se relacionará además este concepto con la granularidad del sensor.

8.2.2 Tipo de movimientos no detectados

La ecuación diferencial (8.1) describe el tipo de movimiento que será automáticamente descartado en el plano imagen. Las funciones solución de esta ecuación son espirales exponenciales que, expresadas en coordenadas cartesianas, tienen la forma ya expresada en la ecuación (6.10). Si se agrupa este resultado y se expresa en función de r en vez de (x, y), se obtiene la ecuación (8.4) que no tiene dependencia angular.

$$r = Ae^{b+vt} \tag{8.4}$$

Siendo A y b constantes, que pueden ser substituidas al cumplirse que en $t = 0$ $r = r_0 = r'_0 F/(z_0 - F)$, con lo que la ecuación (8.4) se puede expresar como:

$$r = \frac{r'_0 F}{z_0 - F} e^{vt} \tag{8.5}$$

En la sección 6.3 se ha realizado la aproximación de la ecuación (6.10), al crecimiento de un objeto en el plano imagen, cuando el eje óptico coincide con la dirección de movimiento. Cabe evaluar la influencia de la no coincidencia del eje óptico con la dirección de movimiento en esta aproximación, y en qué condiciones será válida. Para ello de nuevo, y utilizando la ecuación (8.3), se puede computar la diferencia Δr o error entre el crecimiento en el plano imagen solución a la ecuación diferencial, y el crecimiento *real*. Esta diferencia se expresa en la ecuación (8.6) y es función de β, v_0, z_0 y el tiempo t.

$$\Delta r = \frac{r'_0 F}{z_0 - F} e^{vt} - F \frac{\sqrt{(x'_0 + v_0 t sen\beta)^2 + y'^2_0}}{z_0 - F - v_0 t cos\beta} \tag{8.6}$$

Análogamente a la representación realizada en las figuras 8.2 y 8.3, se ha representado la variación porcentual o $100\frac{\Delta r}{r}$ fijando dos parámetros y viendo la dependencia con el tiempo y un parámetro.

La figura 8.4 es análoga a la figura 8.2 y se pueden extraer las mismas conclusiones que antes. La diferencia estriba en que ahora a pesar de que el ángulo β sea 0, existe un error. Este error crece con el tiempo y en el ejemplo mostrado, con los mismos valores numéricos para F y r' que antes, no supera el 1.5% con 2 segundos de intervalo temporal entre imágenes. De nuevo la mayor divergencia se produce cuando el movimiento es más ortogonal al eje, pero produciéndose un crecimiento más suave que en el caso anterior.

En la figura 8.5 se estudia la dependencia de este error en función de la velocidad en la gráfica (a), y la posición inicial en la gráfica (b). En el caso de la dependencia con la velocidad se observa como crece este error para velocidades altas y bajas, siendo posible encontrar una velocidad (función del experimento) que hace que este error apenas crezca con el tiempo. Este resultado puede resultar interesante en el caso de entornos semi-estructurados en los que se sabe, si aparece un cuerpo con movimiento propio, las características de éste. En cualquier caso los errores previstos son admisibles al no superar el 0.5%.

La dependencia del error con la posición es también similar al resultado de la figura 8.3. En este caso de nuevo el error se neutraliza bastante con la distancia, siendo posible encontrar una distancia inicial, también dependiente del experimento, con la que el error apenas crece con el tiempo.

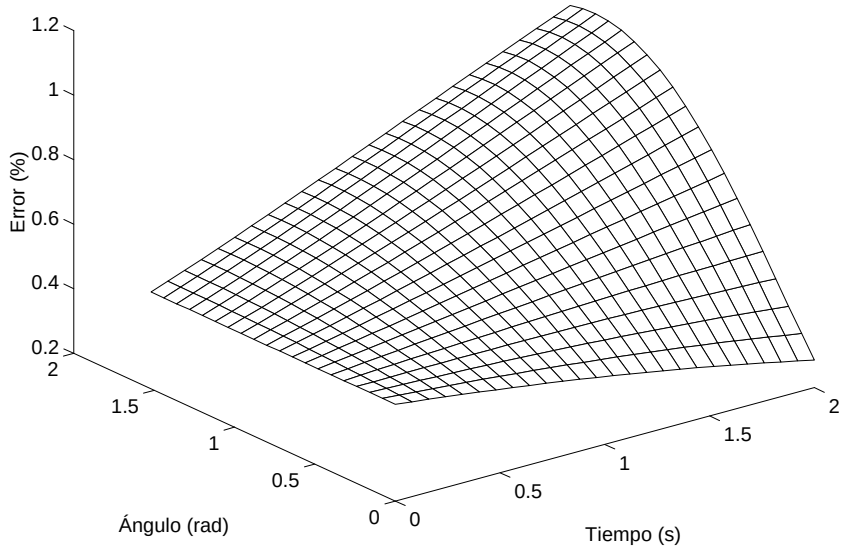


Figura 8.4: *Error en el algoritmo de detección de movimiento en función del ángulo*

8.3 Errores en el algoritmo de cálculo de tiempo al impacto

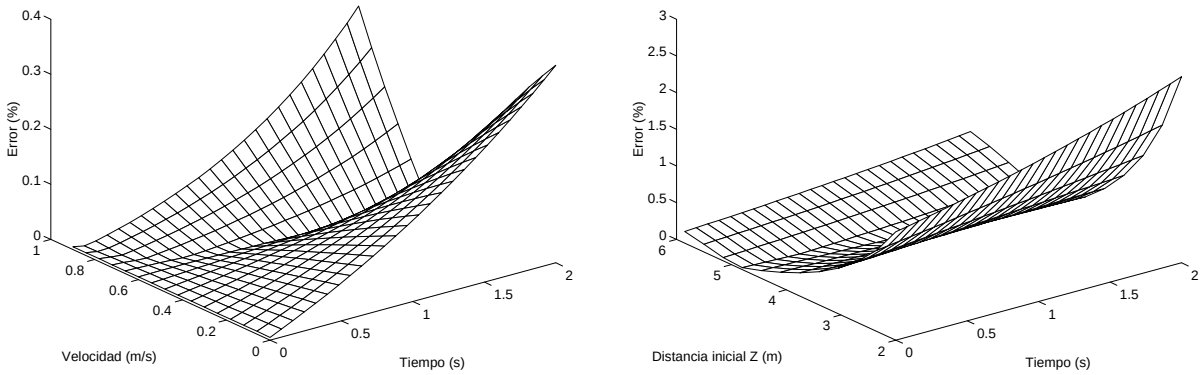
En el algoritmo de cálculo de tiempo al impacto desarrollado en la sección 7 se ha realizado el supuesto de que el eje de movimiento coincidía con el eje óptico del sensor. En el caso, mostrado en la figura 8.1, de que esto no sea así, el algoritmo dará resultados erróneos.

La aparición de la dependencia angular hace que en el plano imagen el crecimiento del objeto ya no sea radial, no cumpliéndose que $d\theta/dt = 0$, con lo que no se puede utilizar la ecuación de Horn (3.4) para simplificar la ecuación (3.7). En este caso aparecen ambas velocidades, la radial $d\xi/dt$ y la angular $d\theta/dt$, con lo que aparece el problema del cálculo del flujo óptico, con sus aproximaciones y costes prohibitivos.

No se puede obtener una ecuación analítica sencilla, como la ecuación (3.8), que permita el cálculo del tiempo al impacto a partir de las derivadas temporales y espaciales del mapa de puntos $E(\xi, \theta, t)$.

Sin embargo, sí que se puede evaluar, mediante la simulación con sencillos experimentos, la dependencia del error porcentual en el cálculo de τ. Para ello, si se reescribe la ecuación original de τ en función de β, v_0, z y el tiempo t entre imágenes, se obtiene la ecuación:

$$\tau' = \frac{z(t)}{V(t)} = \frac{z_0 - v_0 t cos\beta}{v_0 t cos\beta} \tag{8.7}$$



(a) (b)

Figura 8.5: *Error en el algoritmo de detección de movimiento en función de la velocidad (a) y la distancia (b)*

Si se calcula la variación porcentual $100\frac{\Delta\tau}{\tau}$ debida a la aparición de un ángulo β, se obtiene que:

$$100\frac{\Delta\tau}{\tau} = 100\frac{z_0(1-cos\beta)}{z_0 - v_0 t cos\beta} \tag{8.8}$$

En la figura 8.6 se representa la variación del error porcentual expresado en la ecuación (8.8) en función de β y t, con $z_0 = 5\ m$ y $v_0 = 0.25\ m/s$. En esta figura se puede observar como el error crece fuertemente con el ángulo β, tal como cabía esperar. En este caso para un ángulo β dado, el crecimiento del error con el tiempo es suave y casi lineal. La mayor dependencia aparece con el ángulo.

La figura 8.7 muestra la dependencia del error que se produce en el cálculo de τ en función de la velocidad y la distancia del objeto. En este caso el error introducido en función de v_0 y z_0, es de nuevo mayor que en el algoritmo de detección de movimiento. La evolución de las figuras es similar, es decir, a mayor velocidad y menor distancia inicial mayor error en τ, que crece con el intervalo entre imágenes.

En ambas gráficas aparece un error fijo, aunque la velocidad sea muy pequeña o la distancia muy grande, debido al ángulo β.

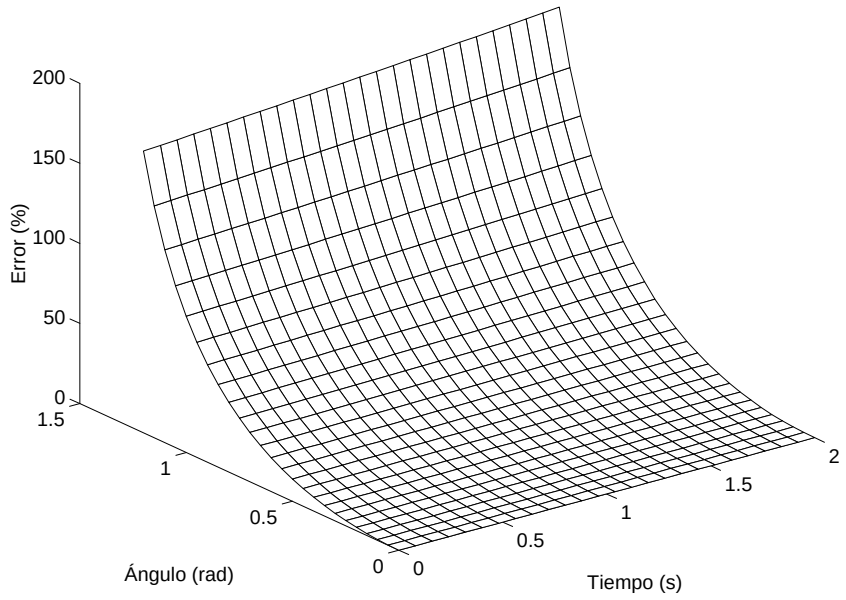


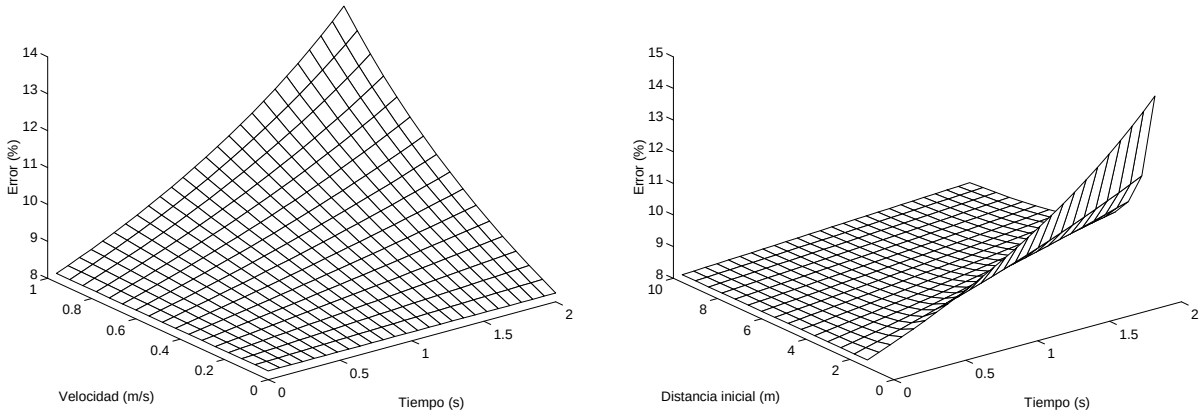
Figura 8.6: *Error en el cálculo del tiempo al impacto función del ángulo*

8.4 Limitaciones relacionadas con la estructura del sensor

El análisis de los errores debido al ángulo β y a las aproximaciones, muestran en que condiciones los algoritmos implementados en el módulo reconfigurable son fiables. Como regla general, el error crece conforme el intervalo de adquisición entre dos imágenes es mayor. Es por tanto interesante adquirir las imágenes muy rápidamente, para reducir este error.

La alta velocidad de adquisición de imágenes de la tarjeta de adquisición log-polar [Bla98], combinada con el rápido procesado conseguido en el cauce reconfigurable pueden no ser siempre una ventaja. Estas características plantean la posibilidad de que, si dos imágenes se adquieren en un intervalo de tiempo muy pequeño, y la evolución dinámica de la escena no es muy rápida, quizás no se produzca ningún cambio apreciable en los puntos de la imagen.

Esta cuestión obliga a un análisis en el que se relacione las características geométricas del sensor, con la velocidad de adquisición, la velocidad relativa robot-objetos y la distancia de la plataforma a éstos.



(a) (b)

Figura 8.7: *Error en el cálculo del tiempo al impacto función de la velocidad (a) y la distancia (b)*

8.4.1 Características geométricas del sensor log-polar CMOS

La retina ocupa cerca del 99% de la superficie del sensor log-polar CMOS [Par97], por ello (además de porque es la zona donde se cumple la ley log-polar), el análisis se realizará sólo en la retina.

La transformación en la zona retínica viene dada por las ecuaciones:

$$\begin{cases} rad = Ae^{B \cdot cir} \\ A = \frac{MINSIZEX \cdot LIMR}{2\pi} \\ B = \log\left(1 + \frac{MINSIZEY}{A}\right) \end{cases} \tag{8.9}$$

siendo rad la distancia al centro del *pixel* y cir es el número del anillo, cantidad ésta que varía entre 0 y 55 para generar los 56 anillos. El valor de las cantidades constantes tienen que ver con el proceso tecnológico utilizado, y las características del sensor, y se definen en la tabla 8.1.

Con las ecuaciones (8.9) y los parámetros de la tabla 8.1, se puede obtener el crecimiento radial de la retina. Para ello sólo hay que hacer variar el parámetro cir entre 0 y 55 para generar, de forma completa los 56 anillos del sensor.

8.4.2 Relación entre el sensor y los parámetros de la escena

Existen dos posibles análisis relacionando la geometría del sensor y los parámetros de la escena. Por una parte, se puede considerar que no se modifica el plano imagen, hasta

Constante	Descripción	Valor
$MINSIZEX$	Anchura del *pixel* más pequeño	13.8 μm
$MINSIZEY$	Altura del *pixel* más pequeño	14.0 μm
$LIMR$	Número de *pixels* por circunferencia	128
$LIMC$	Número de circunferencias	56
radioret	Radio de la retina	4274.12 μm
A	Radio de la fóvea	281.13 μm
B	Coeficiente de crecimiento	0.048598523

Tabla 8.1: *Parámetros de la retina*

que el movimiento del objeto hace que se alcance un *pixel* vecino en el plano imagen. Este análisis puede considerarse como modificación entre *pixels*.

Por otra parte, la evolución del perfil de un objeto dentro de un *pixel* en el plano imagen, hace que su porcentaje de participación en el tono de gris de este *pixel* varíe. Si hay una diferencia apreciable entre el tono de gris del objeto, y el tono de gris del fondo, esto puede producir un cambio de intensidad suficiente para ser detectado como cambio en el *pixel*. En este caso se producirá un cambio en la imagen sin haberse alcanzado el *pixel* siguiente. Este análisis puede considerarse como modificación interna a un *pixel*.

Detección de modificación entre *pixels*

A partir de la figura 8.8 se puede obtener la relación entre el crecimiento en el plano imagen del vector de módulo r y los parámetros dinámicos de la escena, que se expresan con la siguiente igualdad:

$$r = A\, e^{B \cdot c} = \frac{Fr'}{z - F} = \frac{Fr'}{z_0 - v_0 t - F} \tag{8.10}$$

En $t = 0$ el extremo del vector r en el plano imagen estará situado sobre el anillo c_1, de manera que particularizando la ecuación (8.10) a este caso, se obtiene:

$$r_1 = A\, e^{B \cdot c_1} = \frac{Fr'}{z_0 - F} \tag{8.11}$$

La condición para tomar la siguiente imagen es que el extremo del vector r en el plano del sensor haya avanzado al menos hasta el siguiente anillo $c_1 + 1$. Es entonces cuando se apreciará un cambio en la imagen, y se podrán aplicar los algoritmos descritos en los capítulos 6 y 7.

El tamaño del vector $r(\Delta t) = r_2$ en el plano imagen expresado en función del anillo

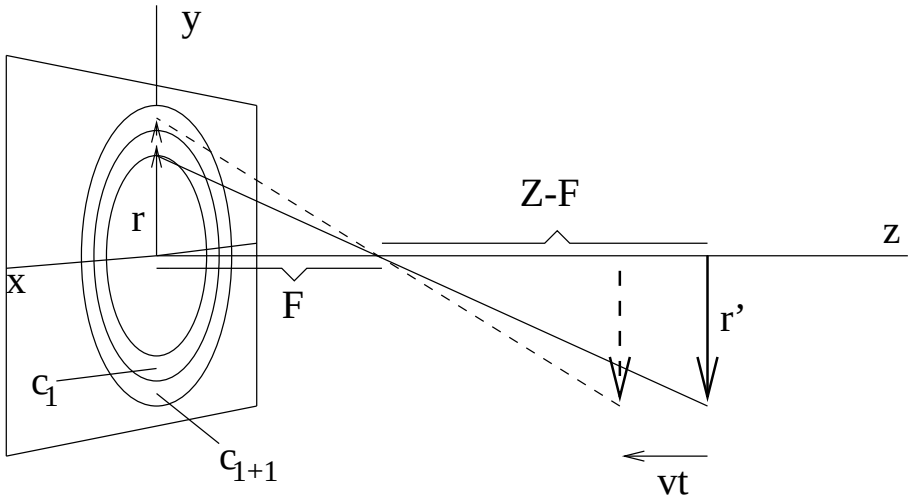


Figura 8.8: *Influencia de la resolución en el crecimiento de un objeto en el plano imagen*

$c_1 + 1$, deberá ser superado (o al menos ser igual), que el crecimiento del vector debido a su acercamiento. Es decir,

$$r_2 = A\, e^{B\cdot(c_1+1)} \leq \frac{Fr'}{z_0 - v_0\Delta t - F} \tag{8.12}$$

Utilizando la ecuación (8.11) se obtiene la relación

$$\Delta t \geq \frac{(e^B - 1)}{e^B} \cdot \frac{(z_0 - F)}{v_0} \tag{8.13}$$

que expresa el tiempo mínimo que debería transcurrir para tomar dos imágenes consecutivas, en función de los parámetros dinámicos de la escena y las características del sensor. Esta ecuación relaciona el coeficiente de crecimiento del sensor, la velocidad relativa entre el sensor y el objeto, (v_0 supuesta constante), y la distancia inicial del objeto z_0.

Para interpretar mejor esta dependencia se ha representado en la figura 8.9 la variación de Δt expresada en la ecuación (8.13). La constante B es la del sensor log-polar, expresada en la tabla 8.1, y la distancia focal de nuevo es 50 mm, aunque la influencia de este último parámetro es despreciable frente a z_0.

Los resultados obtenidos son los esperados. Este intervalo mínimo, para que se aprecien los cambios en la imagen, crece con la distancia y con la lentitud relativa de aproximación del objeto, siendo infinito para velocidad 0.

De la ecuación (8.13) se puede extraer una regla general para obtener la velocidad de programación máxima de adquisición de imágenes. El coeficiente $(e^B - 1)/e^B$ tiene un valor fijado para el sensor log-polar CMOS de 0.04743726012. El cociente $z_0 - F/v_0$

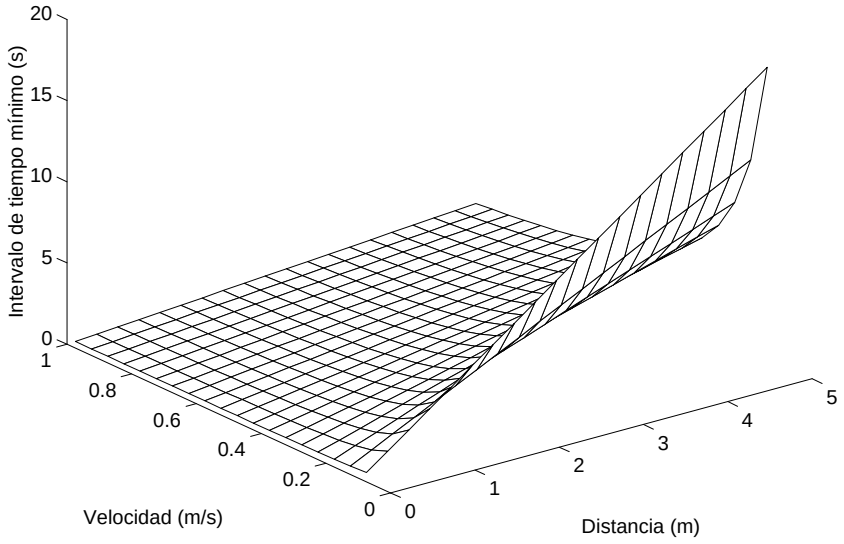


Figura 8.9: *Intervalo mínimo de adquisición entre imágenes consecutivas en función de la velocidad y la distancia*

expresa, (si $z_o \gg F$), el tiempo al impacto τ o el tiempo que se tarda en recorrer la distancia que separa al sensor del objeto de interés. Por tanto se puede concluir que el intervalo temporal de adquisición de dos imágenes consecutivas deberá ser mayor que aproximadamente la veinteava parte de τ. Este resultado sirve para prever, teniendo acotado el rango de velocidades relativas y distancias entre los objetos de un entorno, cual deberá ser la velocidad de adquisición.

Detección de la modificación en un *pixel*

Los resultados anteriores muestran la evolución de un borde entre *pixels* distintos, en función de la geometría del sensor, y los parámetros dinámicos de la escena.

Este análisis puede considerarse el peor de los casos, para limitar la velocidad de adquisición de imágenes log-polares. El posible cambio de intensidad de un *pixel*, debido a la evolución de la ocupación de un *pixel* debe ser tenida en cuenta.

En la figura 8.10 se muestra la ocupación de un *pixel* por parte de la imagen de un objeto en un instante inicial $t = 0$, y tras un intervalo de tiempo Δt. En este modelo, por simplicidad, se ha evitado tener en cuenta que el lado L en realidad debería ser un arco. Esta aproximación simplificará las expresiones, mostrando de la misma manera, la dependencia radial del crecimiento.

El valor de la intensidad de gris T_{GP} del *pixel* de la figura 8.10, se puede expresar en función de la superficie de ocupación del *pixel* por parte de la imagen del objeto S_O, de la intensidad del tono de gris del objeto T_{GO}, del superficie del fondo S_F, y del tono

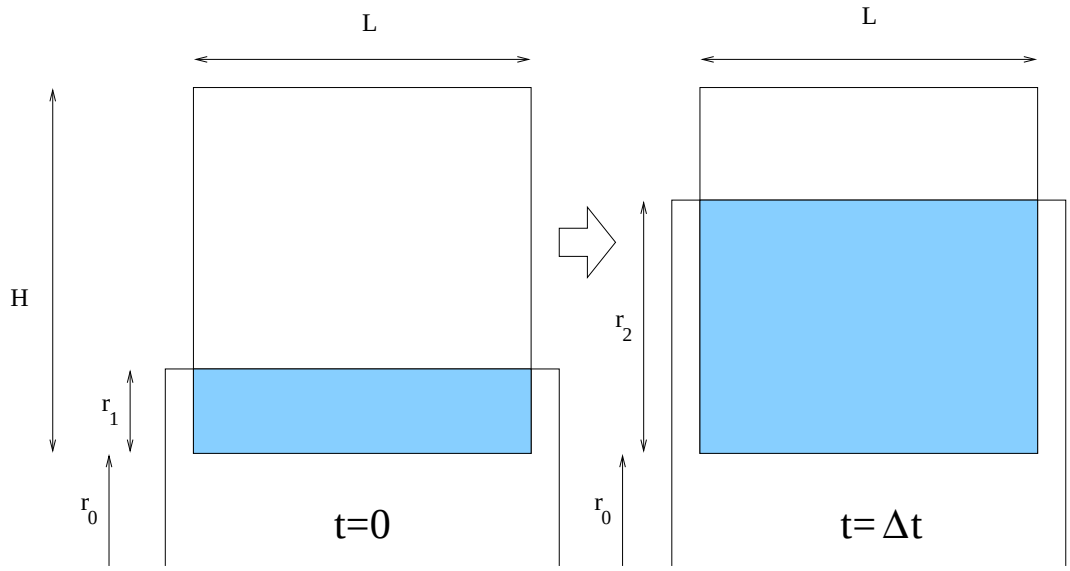


Figura 8.10: *Evolución de la imagen de un objeto en un pixel*

de gris de este fondo T_{GF}.

$$T_{GP} = \frac{S_O T_{GO} + S_F T_{GF}}{S_O + S_F} = \frac{r_i(T_{GO} - T_{GF}) + HT_{GF}}{H} \tag{8.14}$$

Donde r_i es la longitud radial de ocupación del *pixel* por parte de la imagen del objeto.

Por otra parte, la altura H del *pixel* se puede expresar a partir de la ecuación (8.9), en función del número del anillo c_1, y de las constantes del sensor A y B, como

$$H = Ae^{Bc_1}(e^B - 1) \tag{8.15}$$

Particularizando la ecuación (8.14) para los instantes $t = 0$ y $t = \Delta t$, utilizando la ecuación (8.3) con $\beta = 0$ y la ecuación (8.15), se puede obtener la ecuación (8.16) que expresa la diferencia de tono de gris ΔT_{GP} en un *pixel* entre los instantes $t = 0$ y $t = \Delta t$.

$$\Delta T_{GP} = \frac{v\Delta t(T_{GO} - T_{GF})}{(e^B - 1)(z_0 - v\Delta t - F)} > U \tag{8.16}$$

Donde U es el umbral a partir del cual se considerará que se ha producido un cambio en el *pixel*. Este umbral es necesario para evitar que pequeños cambios de iluminación en la escena, afecten a la detección de cambio en un *pixel*.

La ecuación (8.16) tiene la forma que cabría esperar. Por una parte hay un dependencia lineal con la diferencia de tono de gris entre el fondo y el objeto. A mayor

contraste entre el objeto y el fondo, el tono de gris del *pixel* variará más rápidamente. De hecho la ecuación (8.16) se puede expresar como un coeficiente que multiplica la diferencia de tono de gris entre el objeto y el fondo, tal como se indica en la ecuación (8.17). Si esta diferencia es mayor que el umbral U, cuanto más cercano a 1 sea este coeficiente, mejor se detectará la variación de gris en el *pixel*.

$$\Delta T_{GP} = Coeff(z_0, \Delta t, v)\Delta T_G > U \tag{8.17}$$

Por el contrario, cuanto menor sea este coeficiente, menor será la variación del tono de gris del *pixel* aunque el objeto tenga un gran contraste con el fondo.

En la figura 8.11 se representa la variación de este coeficiente para un valor fijo de $z_0 = 5\ m$, una focal $F = 50\ mm$ y con la constante B del sensor log-polar CMOS. Se puede observar como el coeficiente, tal como cabe esperar, crece con el intervalo de tiempo entre imágenes y la velocidad. Para valores de la velocidad de más de 1 m/s, e intervalos de tiempo entre imágenes de más de 0.4 s, se empiezan a obtener valores del coeficiente del orden de 0.5. Esto permite detectar cambios la imagen, suponiendo un umbral U, si existe un contraste entre el objeto y el fondo del doble del valor umbral.

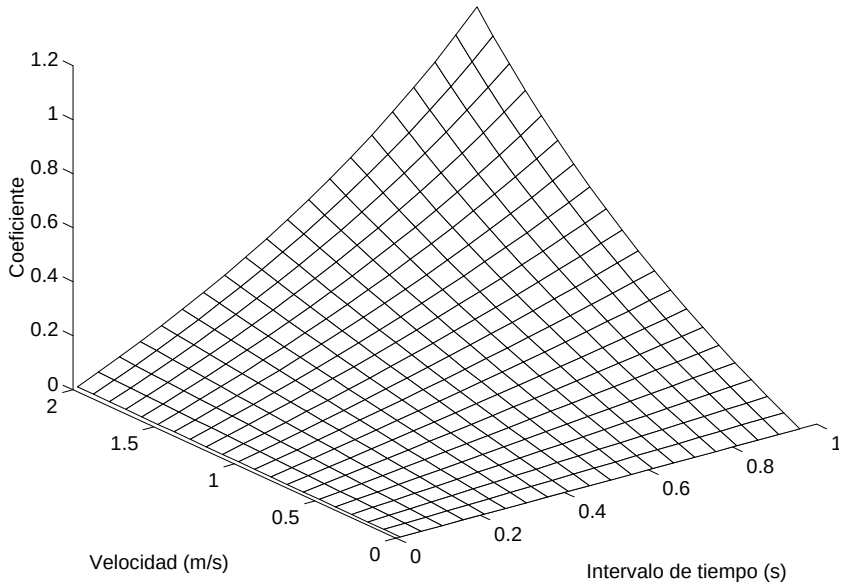


Figura 8.11: *Dependencia con el tiempo y la velocidad del coeficiente $Coeff$ de multiplicación del contraste objeto-fondo*

Como ejemplo numérico un objeto situado a 5 m, acercándose a una velocidad relativa de 0.25 m/s, tiene un tiempo al impacto de 20 s. Según la regla deducida en la sección anterior, para que se produzca un crecimiento en la imagen que afecte al *pixel* vecino, debería de transcurrir al menos la veinteava parte de τ, es decir un segundo.

Aplicando la ecuación (8.17) a este mismo ejemplo numérico, con intervalos temporales de Δt se obtienen coeficientes de aproximadamente el mismo valor numérico que Δt. De esta manera, si la escena tiene un alto contraste, se pueden tomar imágenes con un intervalo temporal menor que un segundo.

8.5 Conclusiones

Los algoritmos implementados en el cauce reconfigurable utilizan la distribución polar de las celdas del sensor para simplificar su cálculo. Para ello es necesario que el vector de movimiento relativo entre la plataforma móvil, y el objeto de interés, coincida con el eje óptico.

En este capítulo se ha estudiado la influencia de la existencia de un cierto ángulo β en el error porcentual entre ambos algoritmos. Los resultados obtenidos muestran, como cabía esperar, que para distancias grandes y velocidades pequeñas este error es menor.

Se han establecido ecuaciones analíticas sólo para la retina ya que esta ocupa el 99% del sensor. Estas ecuaciones permiten establecer una estimación previa del error. De esta manera, se podrá dar fiabilidad o no a la aplicación de los dos algoritmos desarrollados, en función de los parámetros de la escena.

Por regla general estos errores crecen con el intervalo de tiempo entre la adquisición de las imágenes, con lo que es interesante adquirir las imágenes a la mayor velocidad posible.

La tarjeta de adquisición permite la adquisición a velocidades cercanas a 200 imágenes por segundo, aunque se limite a un máximo de 100 para conservar la calidad de la imagen. Se ha planteado que una velocidad de adquisición excesivamente alta, podría hacer que no se produjeran cambios apreciables en la imagen formada en el sensor. A partir de un modelo sencillo de crecimiento radial, se han planteado dos modelos analíticos que indican en que condiciones dinámicas se producirán cambios en la escena.

El primer modelo atiende al cambio entre *pixels*, llegándose a la conclusión de que la velocidad máxima de adquisición es del orden de $\tau/20$, siendo τ el tiempo al impacto.

Este resultado no es definitivo, puesto que el crecimiento de la imagen de un objeto tiende a cambiar el nivel de gris de un *pixel* sin llegar necesariamente a invadir el siguiente. Se ha concluido que la velocidad de adquisición puede ser más rápida, si la diferencia de tono de gris entre el objeto y el fondo es apreciable.

Se ha llegado a una ecuación que permite calcular el límite máximo en la velocidad de adquisición, en función de los parámetros de la escena, y del umbral de cambio U.

Estos resultados permiten ajustar al máximo la velocidad de adquisición de la tarjeta, suponiendo un cierto margen de velocidades relativas entre los objetos y la plataforma móvil, de distancias relativas iniciales, y un margen mínimo de contraste entre tonos de gris de los objetos con el fondo. Reduciendo este intervalo temporal entre imágenes se reducirán los errores debidos a la existencia de un ángulo β, y a las aproximaciones realizadas en los algoritmos utilizados.

En los siguientes capítulos se evaluarán de manera experimental estos algoritmos y la parametrización de estos errores.

Parte III

Experimentación

Capítulo 9

Algoritmo de detección de movimiento

9.1 Introducción

En el capítulo 6 se ha diseñado un algoritmo diferencial para la detección de movimiento en coordenadas log-polares. Este algoritmo tiene la particularidad de que automáticamente descarta el desplazamiento en la imagen debido al movimiento de la cámara cuando este movimiento es a lo largo del eje óptico.

En el capítulo 8 se ha estudiado bajo qué condiciones se debe esperar un correcto funcionamiento del algoritmo. Una conclusión esperada del capítulo ha sido que el ángulo de desviación entre la dirección de movimiento y el eje óptico debe ser lo menor posible. Análogamente se ha concluido que habrá un error inevitable debido a:

1. la diferencia entre el cumplimiento ideal de las condiciones de linealidad expresadas en las ecuaciones (6.5) y el que se consigue con el suavizado de la imagen log-polar implementado.
2. la diferencia entre el cumplimiento de las condiciones de movimiento en el plano log polar expresada en las ecuaciones (6.6), y el movimiento en el plano log-polar expresado en la ecuación (6.11).

Estas conclusiones implican que los puntos de la imagen sin movimiento propio no cumplirán de forma exacta que su segunda derivada temporal sea cero, aunque tendrán un valor absoluto pequeño. Es por tanto necesario determinar un umbral no nulo a partir del cual se considerará que la segunda derivada es nula. El umbral dependerá de la escena y del movimiento relativo entre el vehículo que soporta la cámara y los objetos. En el caso límite de que el vehículo esté quieto no habrá variación en las

imágenes debida al movimiento del vehículo. La variación se deberá exclusivamente a móviles externos, siendo en este caso de aplicación inmediata el algoritmo.

El caso en el que los móviles externos tengan una velocidad del orden de magnitud de la velocidad de la plataforma móvil respecto al fondo estático será usado para testear el algoritmo. En este caso la variación de la imagen debida al movimiento de la cámara debe quedar totalmente descartada y sólo debe detectarse el movimiento del objeto. Es también interesante realizar experiencias con diversas direcciones relativas de movimiento del objeto respecto a la dirección de avance del robot. El caso en el que el movimiento del objeto sea totalmente perpendicular a la dirección de avance del robot, que coincide con el eje óptico, la variación en el plano imagen será mayor que si este movimiento es oblicuo o totalmente paralelo. Este último caso será la prueba más complicada de las que se usaran para validar el algoritmo.

De manera adicional en el capítulo 8 se ha planteado cuál ha de ser el intervalo mínimo de adquisición de imágenes para apreciar variación en ellas en función de las características de movimiento de la cámara. La ecuación (8.13) muestra este intervalo mínimo en el caso de modificación entre *pixels*, aunque como se razona en el capítulo anterior y se muestra en este capítulo experimentalmente, un intervalo más pequeño también muestra variaciones en la imagen debido a la modificación del nivel de gris en un *pixel*. Esta diferencia en las imágenes debida al movimiento de avance del robot sobre el eje óptico de la cámara quedará eliminada con la aplicación del algoritmo tras la deducción del umbral que se realizará en la sección 9.4.3. En las siguientes secciones se describirán los experimentos realizados con el cauce reconfigurable y los resultados obtenidos para determinar la validez del algoritmo de detección de movimiento propio.

9.2 Depuración del cauce reconfigurable

Previamente a la fase de experimentación ha sido necesario realizar una depuración de los EPs, comprobando que el conexionado y funcionamiento de cada uno de los componentes era correcto. En primer lugar ha sido necesario garantizar que las señales globales de reloj y reset eran correctas y carecían de ruido. Posteriormente se ha verificado que el proceso de programación de las FPGAs era correcto y que si el cauce no funcionaba no se debía a una mala programación de las FPGAs. Con esta intención ha sido de utilidad usar y reservar el pin de la FPGA `CONF_DONE` que pasa a valer 1 si la programación se ha realizado con éxito. Este pin se ha conectado a un LED con lo que una mera inspección visual es suficiente para comprobar la correcta programación de cada etapa del cauce.

Una vez se ha garantizado que la programación de los EPs se realiza de manera

correcta se ha pasado a comprobar la funcionalidad de los componentes de los EPs que intervendrán en ambos algoritmos. Con esta intención se han diseñado en VHDL numerosas etapas que, siguiendo el protocolo de intercambio de datos descrito en la sección 4.3, acceden a los recursos de los EPs.

De esta manera en primer lugar se ha comprobado la conectividad del cauce programando todas las etapas con un algoritmo que simplemente transmite los datos. Posteriormente se ha realizado código VHDL para escribir una imagen log-polar completamente en la memoria local y en las memorias de doble puerto de cada EP. Este acceso a los recursos de los EPs se ha realizado de manera aislada inicialmente (accediendo a cada recurso por separado y de forma secuencial), y con un cauce inicialmente formado por un solo EP y aumentando el número de EPs del cauce posteriormente. Esta metodología ha sido muy costosa temporalmente pero ha sido útil para garantizar el correcto funcionamiento de la totalidad de cada EP previamente a la experimentación con los algoritmos.

9.3 Descripción de los experimentos

Se han realizado varios experimentos con secuencias reales tomadas desde el robot móvil desarrollado en el Institut de Robòtica de la Universitat de València [Veg99]. Esta plataforma se muestra en la figura 9.1 y en todas las experiencias se ha mantenido en lo posible el eje óptico de la cámara coincidente con la dirección de movimiento. Se han obtenido secuencias de 190 imágenes, a 12 imágenes por segundo, con una velocidad constante del robot de avance a lo largo del eje óptico de 0.05 m/s, para de esta manera poder determinar el intervalo Δt de adquisición en función de la velocidad del robot.

Claramente hay una relación entre Δt y las velocidades de la cámara y de los objetos para garantizar que se produzcan diferencias. De esta manera, procesando imágenes a intervalos constantes de entre todas la imágenes adquiridas, se puede analizar la relación entre la velocidad de la plataforma y el intervalo mínimo de adquisición temporal Δt.

Con la intención de parametrizar en función de la velocidad el intervalo de tiempo para procesar imágenes y la elección del umbral en función de este intervalo, se ha realizado el procesamiento en el cauce reconfigurable conectado al puerto paralelo del PC mediante una tarjeta especialmente diseñada para estos experimentos. Esta tarjeta incluye el bus de programación de los EPs, la fuente de datos del primer EP y la conexión de recepción del último EP. De esta manera es posible programar los EPs a partir del fichero **rbf** generado para las FPGAs, enviar imágenes log-polares al inicio del cauce y recibir las estructuras de datos resultado de este procesamiento. Simultáneamente se han desarrollado funciones en C que permiten la gestión del puerto paralelo en

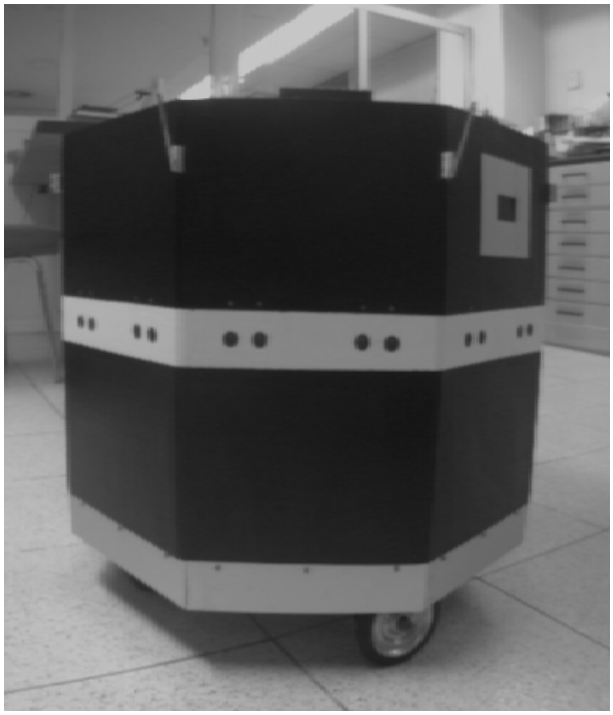


Figura 9.1: *Plataforma móvil desde la que se han tomado las imágenes*

modo EPP para la programación del cauce reconfigurable y el envío/recepción de las imágenes log-polares. La tarjeta suministra de manera adicional la alimentación al cauce reconfigurable y las señales globales de reset y de reloj de 16MHz. La figura 9.2 muestra el banco de pruebas conectado al cauce reconfigurable en el que se han realizado los experimentos.

Con las secuencias de imágenes log-polares almacenadas en el disco duro es posible estudiar el comportamiento del algoritmo en función de la velocidad sin más que procesar imágenes a intervalos constantes.

Si se adquieren imágenes muy rápidamente y la velocidad de los objetos y del robot no es muy alta puede ocurrir que no se produzca variación apreciable entre dos imágenes. De esta manera en el capítulo 8 se ha deducido una primera relación que define el intervalo mínimo para la adquisición de imágenes log-polares en función de los parámetros de la escena. El intervalo de adquisición mínimo Δt depende de la constante de crecimiento del sensor B, la distancia del objeto z_0, la focal de la cámara F y de la velocidad relativa entre cámara y objeto v_0.

$$\Delta t \geq \frac{(e^B - 1)}{e^B} \cdot \frac{(z_0 - F)}{v_0} \tag{9.1}$$

Para la deducción de esta ecuación se ha impuesto que la variación de la imagen

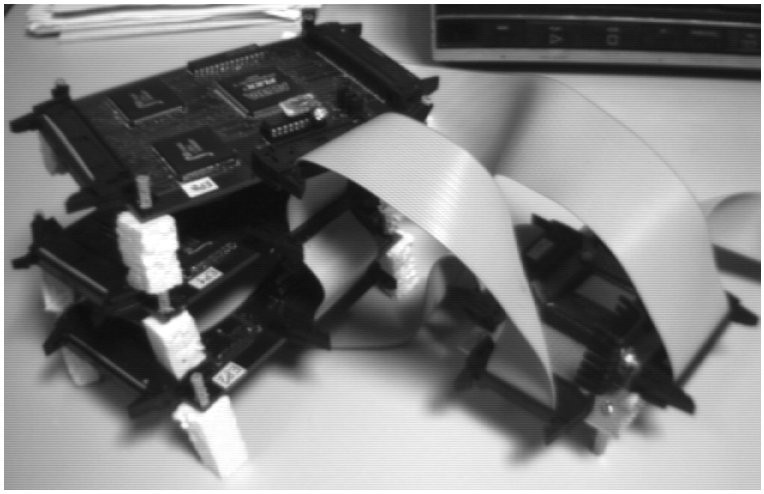


Figura 9.2: *Cauce reconfigurable con 3 EPs y la tarjeta de programación y entrada/salida de imágenes*

en el plano log-polar se deba a que la imagen de un objeto pase de un anillo polar al anillo inmediatamente siguiente. Este intervalo temporal puede ser incluso menor para que se aprecien cambios en la imagen, ya que el avance en el plano imagen log-polar dentro de un *pixel* sin invadir el siguiente puede ser suficiente para que se produzcan cambios, tal como se expresa con la ecuación (8.16). A partir de esta ecuación se obtiene la expresión de Δt, en función de los mismos parámetros descritos anteriormente, el tiempo al impacto τ, y la diferencia entre el tono de gris del objeto y el tono de gris del fondo $T_{GO} - T_{GF}$.

$$\Delta t \geq \frac{\tau}{\frac{T_{GO}-T_{GF}}{e^B-1}+1} \tag{9.2}$$

En la sección 9.4.2 se analizará esta relación entre los parámetros de la escena y la velocidad de adquisición que garantice variación entre imágenes. Se observará como la velocidad fija el intervalo Δt mínimo de adquisición de imágenes.

Con el objeto de determinar el valor del umbral en función de Δt se ha realizado una primera secuencia en la que no hay objetos móviles y la variación en las imágenes se debe exclusivamente al movimiento de la cámara, tal como se muestra en la figura 9.3. El algoritmo debe garantizar la no aparición de falsos positivos, o lo que es lo mismo, no debe detectar puntos con movimiento propio ya que en este caso sólo se mueve la cámara. En la sección 9.4.3 se propone una metodología para que la plataforma móvil determine de manera unívoca y automática el umbral a aplicar.

A partir de la determinación del umbral se han realizado experiencias con un objeto con una velocidad lineal constante de aproximadamente 0.055 m/s respecto al fondo

Figura 9.3: *Imágenes de la secuencia 1: sólo hay movimiento del robot*

estático y con diversos ángulos de avance respecto a la dirección de movimiento del robot. En la secuencia 2 que se muestra en la figura 9.4, el pequeño coche móvil avanza en diagonal formando un ángulo de 45^o con la dirección de avance del robot. La secuencia 3 que se muestra en la figura 9.5, es el caso más favorable ya que la dirección del vehículo móvil es totalmente perpendicular a la dirección de movimiento del robot. Finalmente la secuencia 4 es el caso más desfavorable, ya que el vehículo móvil avanza frontalmente hacia el robot.

Figura 9.4: *Imágenes 10, 50 y 90 de la secuencia 2: la dirección de movimiento del objeto forma un ángulo de* 45^o *con la dirección del robot*

9.4 Resultados

Con la intención de observar el resultado de cada etapa del algoritmo de detección de movimiento, se ha implementado una etapa que únicamente transmite los datos, sin transformarlos. De esta manera, se puede analizar el resultado parcial del procesamiento en cada etapa experimentando con imágenes log-polares en el banco de pruebas descrito.

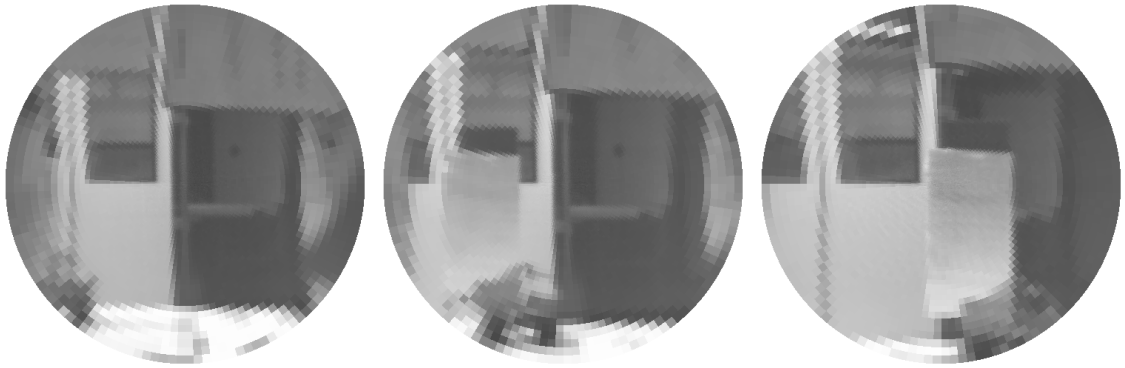


Figura 9.5: *Imágenes 10, 50 y 90 de la secuencia 3: la dirección de movimiento del objeto es perpendicular a la del robot*

Figura 9.6: *Imágenes 10, 50 y 90 de la secuencia 4: el objeto avanza frontalmente hacia el robot*

9.4.1 Etapa de suavizado de la imagen

En la sección 6.4.1 se ha descrito cómo se ha implementado la etapa de suavizado de la imagen, necesaria para cumplir la condición de suavizado expresada en la ecuación (6.5). Con intención de acelerar el suavizado se ha utilizado una máscara unitaria de convolución de 6 puntos, seleccionando para cada punto 3 puntos del anillo anterior y los 2 puntos inmediatamente anteriores del anillo presente. Se realiza una suma de estos 6 puntos, escogiendo los bits más significativos para, de esta manera, realizar un división por una potencia de 2 en lugar de por 6 que sería el valor más correcto. De esta manera la etapa de suavizado procesa un punto cada 4 ciclos.

Al realizarse una división por una potencia de dos se puede realizar ésta por 8 que sería la potencia más próxima. Si cada *pixel* toma su valor máximo (255 que correspondería a totalmente blanco), el valor máximo de la suma sería $255 \cdot 6 = 1.530$, resultado almacenado en un registro de 11 bits. Si se escogen los 8 bits más significativos se obtiene que el tono de gris más alto que se obtendrá será el de la división entera

de 1.530 entre 8, que es 191. Esto implica que las imágenes suavizadas se oscurecerán ligeramente, tal como se muestra en la secuencia de la figura 9.8, que es el resultado de suavizar la secuencia original de la figura 9.7.

Figura 9.7: *Imágenes originales de la secuencia que muestra el suavizado*

Figura 9.8: *Imágenes suavizadas y almacenadas en 8 bits*

La figura 9.8 muestra un suavizado de la imagen a la vez que un cierto oscurecimiento de la misma. En el caso de que fuera necesario un suavizado más perfecto, o que no fuera tolerable el oscurecimiento de la imagen, se podría ampliar el tamaño de la máscara y/o dividir exactamente por el tamaño de la máscara. Esto implicaría un mayor número de registros (EPs más grandes) y/o un número de ciclos mayor para esta etapa, lo cual ralentizaría el funcionamiento del cauce en el algoritmo de detección de movimiento.

De manera adicional se ha de tener en cuenta que las etapas siguientes realizan una diferenciación y, por tanto, cabe la posibilidad de que el resultado sea negativo. Si se realiza la diferenciación de números originalmente de 8 bits, almacenando el resultado en complemento a 2, serían necesarios 9 bits para evitar un desbordamiento y por tanto un resultado incorrecto en la diferenciación. Con intención de evitar un desbordamiento en las etapas posteriores se ha fijado la salida de la etapa de suavizado en 7 bits, o equivalentemente dividir por 16 el resultado de la suma de la máscara de 6 bits en

lugar de la división por 8 propuesta inicialmente. Esta aproximación oscurece más las imágenes ya que al dividir por 16 el valor más alto representable será 95, quedando el bit más alto del resultado siempre a cero. La figura 9.9 muestra el suavizado de la secuencia de la figura 9.7 realizado con 7 bits para evitar el desbordamiento en etapas posteriores.

Figura 9.9: *Imágenes suavizadas y almacenadas en 7 bits*

Claramente el resultado de la figura 9.9 sería inaceptable en el caso de que la intención de estas imágenes fuera mostrar la secuencia original simplemente suavizada. La aproximación produce una pérdida de información la imagen visualmente intolerable al reducir los niveles de gris. Al ser el objetivo final de la aplicación del algoritmo la detección de movimiento esta aproximación será válida si los resultado experimentales son correctos. En el caso de los experimentos realizados con una iluminación normal la aproximación ha demostrado ser suficientemente buena, evitándose de esta manera una etapa lenta que ralentizase el cauce en este algoritmo.

9.4.2 Etapas de diferenciación de la imagen

La velocidad de adquisición y procesado de las imágenes log-polares es relativamente alta comparada con la velocidad del robot y de los objetos móviles en los experimentos. Es interesante que las imágenes a procesar estén relativamente próximas temporalmente. De esta manera se puede asumir como mínimo el error producido al discretizar las derivadas temporales pasando de ∂t a Δt. Por otra parte, puede ocurrir que si las velocidades de la escena son comparativamente mucho menores que la velocidad de adquisición no se produzca ninguna variación apreciable en la imagen, con lo que la derivada sería nula. Es por tanto interesante fijar el intervalo temporal mínimo que garantice diferencia entre imágenes.

La ecuación (9.1) muestra el intervalo temporal mínimo entre imágenes en el caso de que la variación en la imagen se deba a que la imagen del objeto haya cambiado

de *pixel*. En el caso del experimento 1, en el que la plataforma móvil avanza a una velocidad de 0.05 m/s y los objetos de la escena están a una distancia aproximada de 1.5 m, este intervalo temporal llega a ser de 1.4 segundos. Claramente este es el peor caso y no es necesario este gran intervalo temporal. El avance de la imagen del objeto en el plano del sensor hace que el movimiento dentro de un solo *pixel* sea suficiente para modificar el nivel de gris del *pixel*, y por tanto producir variación en el *pixel*. La expresión (9.2) muestra este intervalo mínimo en función de las características geométricas del sensor y los parámetros de la escena. Este valor depende de la diferencia entre el tono de gris del objeto y del fondo $T_{GO} - T_{GF}$. Para una diferencia mínima (valor 1) de tono de gris entre el objeto y el fondo y para los valores de la escena indicados anteriormente el intervalo temporal es de nuevo del orden de 1.4 segundos. Si esta diferencia entre tonos de gris es algo mayor, el intervalo temporal para producir variación en el plano imagen será menor, y por tanto se podrán tomar imágenes más próximas. Esta diferencia supuesta de tonos de grises entre los objetos y el fondo se puede generalizar a un valor distinto de la unidad pero suficientemente pequeño. En el caso de los experimentos realizados la diferencia mínima entre el tono de gris del objeto $T_{GO} - T_{GF}$ se ha supuesto de un valor de al menos 5 unidades. Si la diferencia de tono de gris entre objeto y fondo es mayor entonces la diferencia entre los niveles de gris de imágenes consecutivas en el plano imagen también será mayor. Con este valor y los parámetros de la escena apuntados anteriormente se obtiene que $\Delta t \approx 0.33$ segundos. En el caso de los experimentos realizados tomando 12 imágenes por segundo esto indica que una de cada cuatro imágenes tiene que ser procesada, con lo que $\Delta t = 0.\hat{3}$.

Si la diferencia de tonos de gris es mayor que el valor mínimo de 5, supuesto el intervalo de tiempo mínimo será menor, con lo que el intervalo temporal de 0.3 segundos garantizará en cualquier caso diferencias mayores en el plano imagen. De esta manera se pretende que la plataforma autónoma escoja de manera automática el intervalo Δt de las imágenes que va a procesar en función de su velocidad lineal y de la velocidad de procesamiento del cauce reconfigurable. Este intervalo se podrá cumplir siempre que no se supere la velocidad a la que se pueden suministrar las imágenes desde la etapa de adquisición del sistema (del orden 100 imágenes/segundo). En el caso del algoritmo de detección de movimiento se ha realizado de manera adicional un estudio de la influencia del intervalo Δt en la elección del umbral.

Las figuras 9.10 y 9.11 muestran parte de la secuencia original del experimento 2 tomando 3 imágenes por segundo. En la secuencia el robot avanza frontalmente y el objeto avanza en una dirección que forma un ángulo de 45^o con la del robot. De esta manera se produce un desplazamiento radial de los objetos estáticos en la escena y un desplazamiento del objeto de izquierda a derecha. En la secuencia aparece como más apreciable la variación en la imagen producida por el objeto con movimiento propio

que la debida al avance de la plataforma móvil a pesar de ser del mismo orden (0.05 m/s de la plataforma y 0.055 m/s del objeto). Este efecto producido al coincidir la dirección de movimiento con el eje óptico se puede deducir de la ecuación (9.1) ya que el movimiento se aprecia en la imagen cuando un *pixel* avanza hasta el siguiente. En el caso de los objetos estáticos, este desplazamiento se debe al avance de la plataforma móvil y tal como se ha indicado en el caso de este experimento será necesario más de 1 segundo. Este movimiento se aprecia al ver las diferencias entre la primera imagen de la secuencia y la quinta (1.333 segundos más tarde).

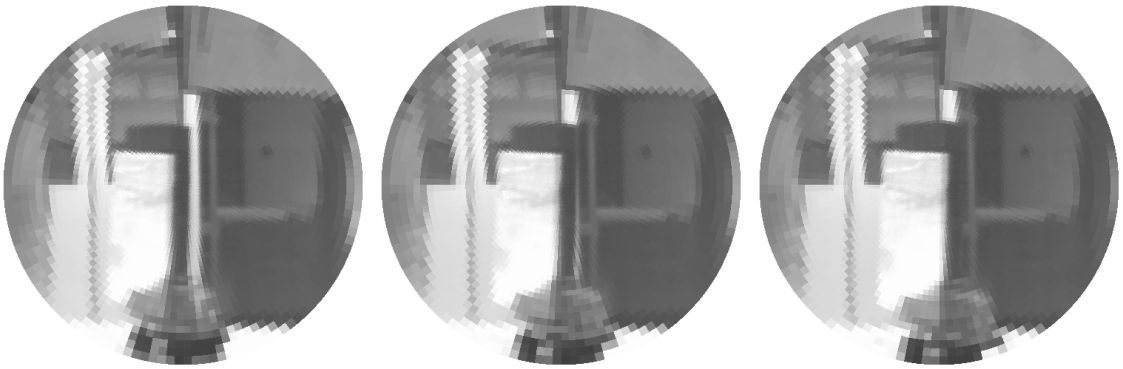


Figura 9.10: *Imágenes originales para diferenciar tomadas cada 0.333 segundos. Parte I*

Figura 9.11: *Imágenes originales para diferenciar tomadas cada 0.333 segundos. Parte II*

La figura 6.2 muestra el resultado de calcular la derivada primera de la secuencia formada por las figuras 9.10 y 9.11 y después de atravesar la etapa de suavizado. A partir de las 6 imágenes originales se obtendrán 5 imágenes resta o primera derivada temporal, ya que hay una latencia inicial de una imagen en la salida de la segunda etapa del algoritmo. En la figura 9.12 se muestran las 4 últimas imágenes de la serie de las 5 derivadas primeras.

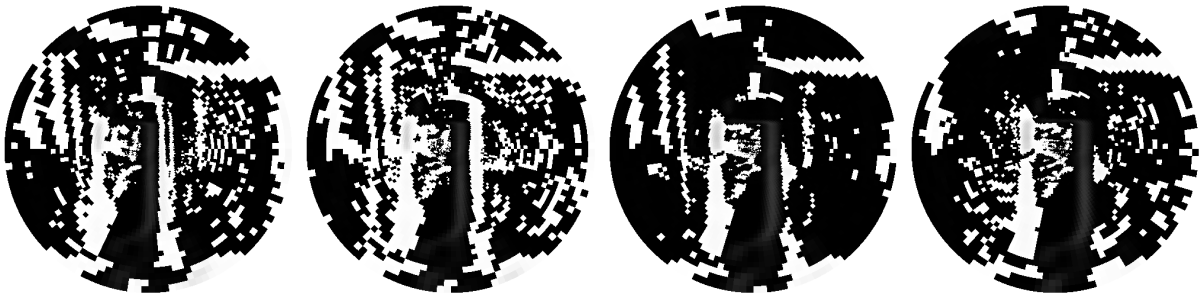


Figura 9.12: *Primera derivada temporal de la secuencia de imágenes de las figuras 9.10 y 9.11*

Las imágenes muestran el comportamiento esperado. Mayoritariamente los puntos tienen valores muy oscuros o casi blancos. El tono de gris totalmente negro corresponde al valor 0 y los valores muy oscuros a valores positivos muy cercanos a 0. Estas diferencias indican que, o bien no se ha producido variación, o bien la variación ha sido mínima. Análogamente el blanco absoluto corresponderá al máximo valor del *pixel* (255) que corresponde en complemento a 2 al valor -1. Los valores muy claros corresponderán a valores cercanos a 255 e inferiores. Estos niveles de gris en complemento a 2 expresan valores negativos cercanos a cero. Las zonas donde se observan tonalidades grises intermedias corresponden a valores positivos o negativos más alejados de cero, lo que indica que se ha producido una variación más brusca.

La última etapa del algoritmo de detección a su vez diferencia las imágenes que son la derivada primera para obtener la derivada segunda. Las imágenes correspondientes a la secuencia ejemplo se muestran en la figura 9.13. En este caso la secuencia original de 6 imágenes se reduce a sólo 4 imágenes al tener una latencia inicial de dos imágenes el cómputo de la derivada segunda.

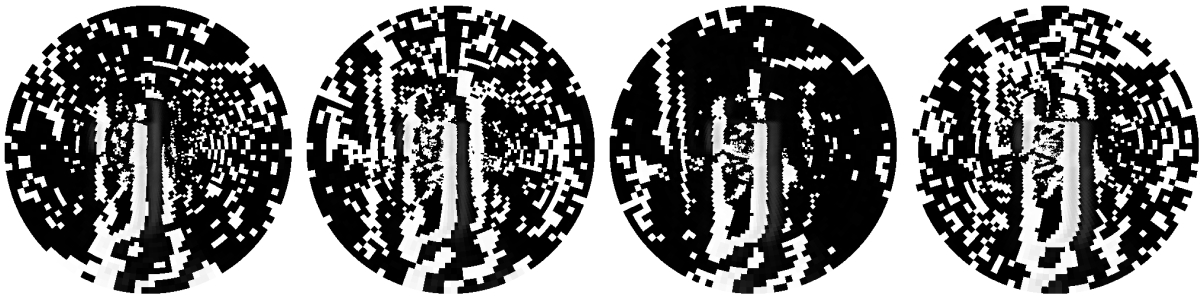


Figura 9.13: *Segunda derivada temporal de la secuencia de imágenes de las figuras 9.10 y 9.11*

El objetivo de realizar un segunda diferenciación consiste en eliminar los gradientes o diferencias constantes que hayan aparecido tras la primera derivada. De esta manera

la segunda derivada eliminará la aparición de diferencias debido al avance de superficies con un gradiente constante (imágenes suavizadas), permaneciendo preferentemente los bordes de los objetos. Tras esta segunda diferenciación será necesario determinar el umbral que va a ser útil para detectar los puntos que pertenecen a objetos con movimiento propio. Idealmente la secuencia de la figura 9.13 debería mostrar solamente con tonos de gris los puntos con movimiento propio, siendo el resto de puntos totalmente negros (sin diferencias). Al no cumplirse de manera exacta las condiciones del algoritmo es necesario determinar un umbral que fijará a partir de qué valor se considerará que un punto ha variado y por tanto tiene movimiento propio.

9.4.3 Determinación del umbral

Para la determinación del umbral es necesario tener en cuenta la velocidad de la plataforma autónoma y la velocidad de adquisición de imágenes. En la sección 9.4.2 se ha analizado como elegir la velocidad de adquisición de las imágenes para garantizar que se produzca variación entre imágenes consecutivas y de esta manera tenga sentido la diferenciación. La ecuación (9.2) expresa el valor mínimo de este intervalo en función de los parámetros dinámicos de la escena y de la diferencia de tono de grises entre el objeto y el fondo de la imagen ($T_{GO} - T_{GF}$). Se ha supuesto una diferencia mínima de tono de gris de 5, para fijar automáticamente el intervalo Δt en función de la velocidad v. Una vez la plataforma móvil fija esta velocidad de adquisición debe automáticamente fijarse el umbral que descarte el desplazamiento en la imagen debido al movimiento propio.

Para observar la evolución del umbral en función del intervalo temporal de adquisición se ha realizado el experimento 1 en el que la cámara avanza en la dirección del eje óptico sin objetos móviles, tal como se muestra en la secuencia de la figura 9.3. Al aplicar el suavizado, calcular la primera derivada y segunda derivada de las imágenes log-polares, el algoritmo tiene que garantizar que no habrá puntos con movimiento propio en las imágenes procesadas. Claramente habrá una relación entre la elección de la velocidad de adquisición Δt, el umbral escogido y los falsos positivos.

La tabla 9.1 muestra el valor medio del número de falsos positivos obtenidos en el experimento 1 en el que se han adquirido 190 imágenes en intervalos de $0.08\hat{3}$ segundos. El número de falsos positivos se muestra en función del intervalo de imágenes seleccionadas para procesar, N, y del valor del umbral aumentado cada 5 unidades.

El número medio de falsos positivos crece generalmente con el intervalo entre imágenes y decrece aumentando el umbral. La figura 9.14 muestra de forma gráfica los falsos positivos de la tabla 9.1. El objetivo final es automatizar la elección del umbral en función de la velocidad, que a su vez ha fijado el rango de intervalos de adquisición de imágenes.

		Intervalo de imágenes N									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Umbral	**5**	57	206	385	376	450	534	785	1.253	1.431	1.879
	10	0	7	76	29	58	59	189	449	664	845
	15	0	0	0	0	1	1	26	189	177	380
	20	0	0	0	0	0	0	2	46	9	194
	25	0	0	0	0	0	0	0	1	1	65
	30	0	0	0	0	0	0	0	0	0	3
	35	0	0	0	0	0	0	0	0	0	0
	40	0	0	0	0	0	0	0	0	0	0

Tabla 9.1: *Número medio de falsos positivos en el experimento 1 en función del intervalo de imágenes (columnas) y del umbral (filas)*

A partir de la tabla 9.1 es posible representar el umbral mínimo para que no se produzcan falsos positivos en función del intervalo de imágenes seleccionadas para enviar al cauce reconfigurable. La figura 9.15 muestra estos puntos y la recta, obtenida por mínimos cuadrados, que mejor se ajusta a estos puntos experimentales. Cabe hacer notar que es equivalente la representación del umbral en función del intervalo temporal de adquisición sin más que multiplicar N por 0.083 para obtener el valor en segundos. De esta manera se ha obtenido una regla para que de manera automática la plataforma móvil fije el umbral del algoritmo en función del intervalo de adquisición. A su vez el intervalo de adquisición viene fijado por la velocidad propia de la plataforma, con lo que queda automatizado el proceso de selección de imágenes para procesar y de elección del umbral en función de la velocidad de la plataforma.

La recta que expresa el nivel de gris umbral obtenida tras la regresión lineal que se muestra en la figura 9.15 es:

$$umbral = 2.6364 \cdot N + 7.0000 \tag{9.3}$$

En el caso del experimento 1 la velocidad de la plataforma es de 0.05 m/s, velocidad que fija un intervalo de adquisición equivalente a procesar una de cada 4 imágenes adquiridas fijando $N = 4$. De esta manera el umbral fijado tras aplicar la ecuación (9.3) será (truncando la parte entera) igual a 17. Este umbral garantizará para una velocidad de 0.05 m/s y un intervalo temporal de adquisición de $0.\hat{3}$ segundos la no aparición de falsos positivos siempre que se mantengan las condiciones de iluminación, requisito no muy restrictivo y de aplicabilidad en entornos interiores.

Es posible aplicar el umbral fijado por la velocidad del vehículo a la secuencia ejemplo utilizada para ilustrar las etapas de suavizado y diferenciación. La figura 9.16

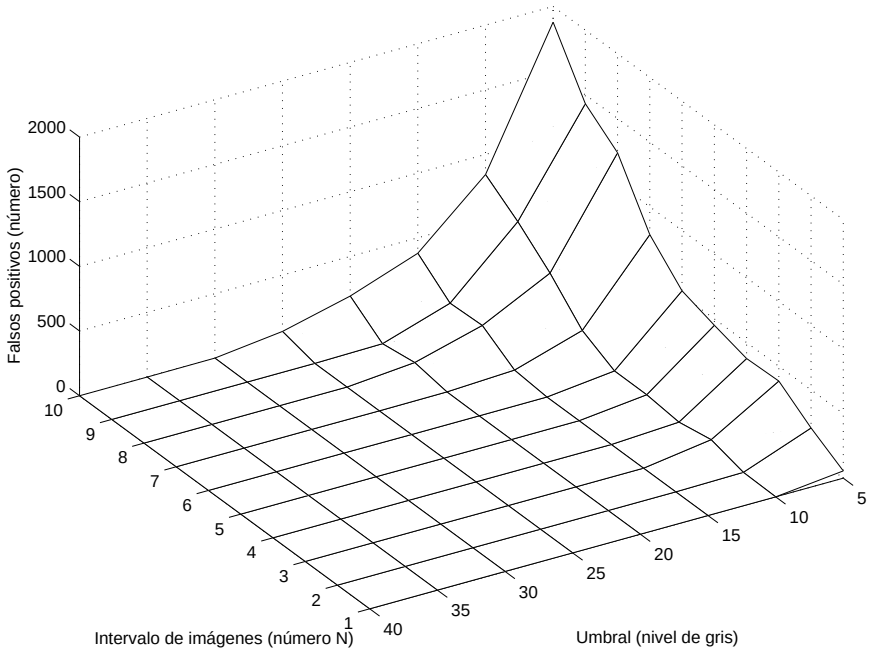


Figura 9.14: *Número medio de falsos positivos en función del intervalos de imágenes tomadas y del umbral para el experimento 1*

muestra los puntos detectados con movimiento propio obtenidos con un umbral de 17 para la secuencia de las figuras 9.10 y 9.11. Cabe hacer notar cómo se han eliminado los falsos positivos y los puntos detectados sólo pertenecen a los bordes del vehículo en movimiento.

Para observar de qué manera el umbral elimina de forma efectiva los falsos positivos es interesante representar un histograma de las derivadas segundas conjuntamente con el umbral. La figura 9.17 muestra los histogramas de las 2 primeras imágenes correspondientes a las derivadas segundas temporales que se muestran en la figura 9.13. El eje Y ha sido cortado para que la escala permita ver con detalle las zonas interesantes de la gráfica. En esta figura se muestra la frecuencia de aparición de los niveles de gris de la derivada segunda obteniéndose cantidades muy grandes para diferencias pequeñas, tanto positivas (valores cercanos a 0), como negativas (valores cercanos a 255). El umbral es la línea vertical que aparece en el valor 17 y -17 (239 en complemento a 2). Los puntos a la izquierda del 17 y a la derecha del 239 son los descartados por el algoritmo como falsos positivos y los situados a la derecha del 17 y a la izquierda del 239 son los marcados como positivos en las 2 primeras imágenes de la figura 9.16. Los falsos positivos, debidos a la no idealidad de las condiciones de suavizado y de solución de la ecuación diferencial impuestas por el algoritmo, se eliminan con la aplicación del umbral, permaneciendo los picos debidos a los bordes del objeto con movimiento propio.

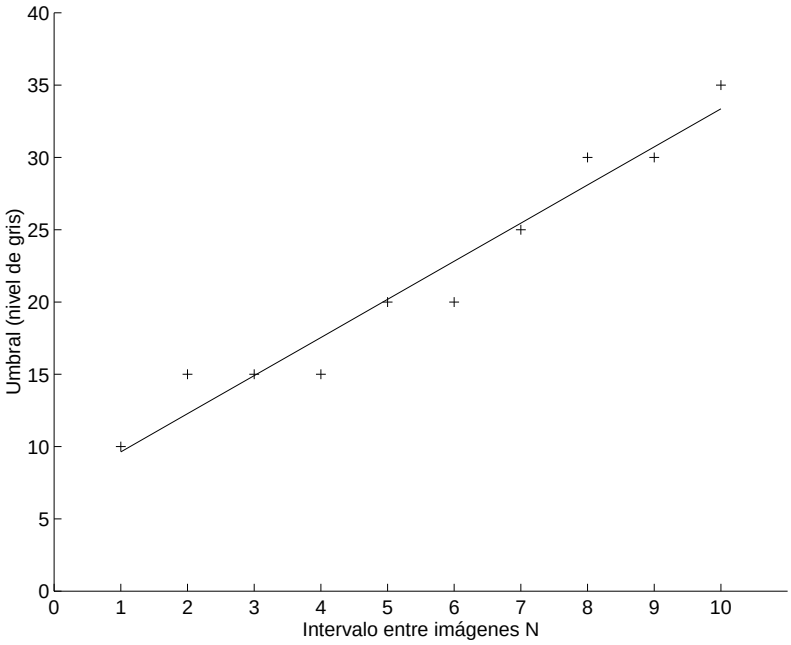


Figura 9.15: *Umbral mínimo para evitar los falsos positivos en función del intervalo de imágenes procesadas N*

9.4.4 Detección de movimiento propio

Se ha justificado la elección del intervalo de adquisición de imágenes y del umbral en función de la velocidad. A partir de estos resultados se ha aplicado el algoritmo de detección de movimiento a las secuencias de los experimentos descritos en la sección 9.3. Se han eliminado las primeras imágenes y las últimas de las secuencias de 190 imágenes en los experimentos para dejar secuencias en las que aparezca el móvil de una manera efectiva.

De la misma manera se muestra una imagen del inicio de la secuencia, una imagen del final de la secuencia y una imagen intermedia, para no hacer muy extensa la des-

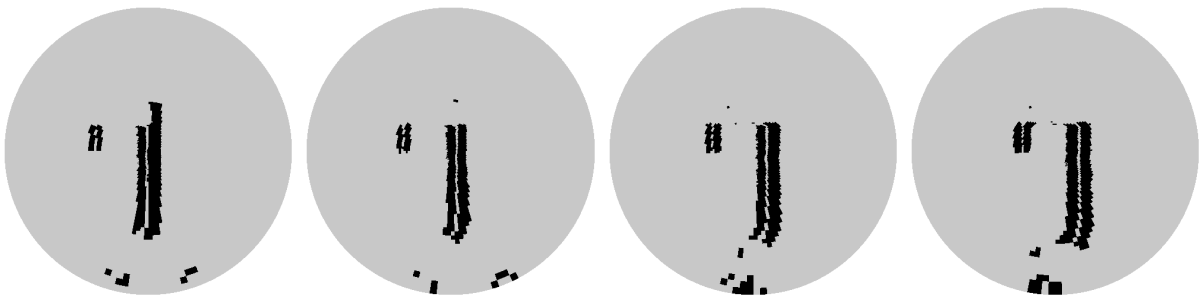


Figura 9.16: *Positivos obtenidos con un umbral de 17 de la secuencia de las figuras 9.10 y 9.11*

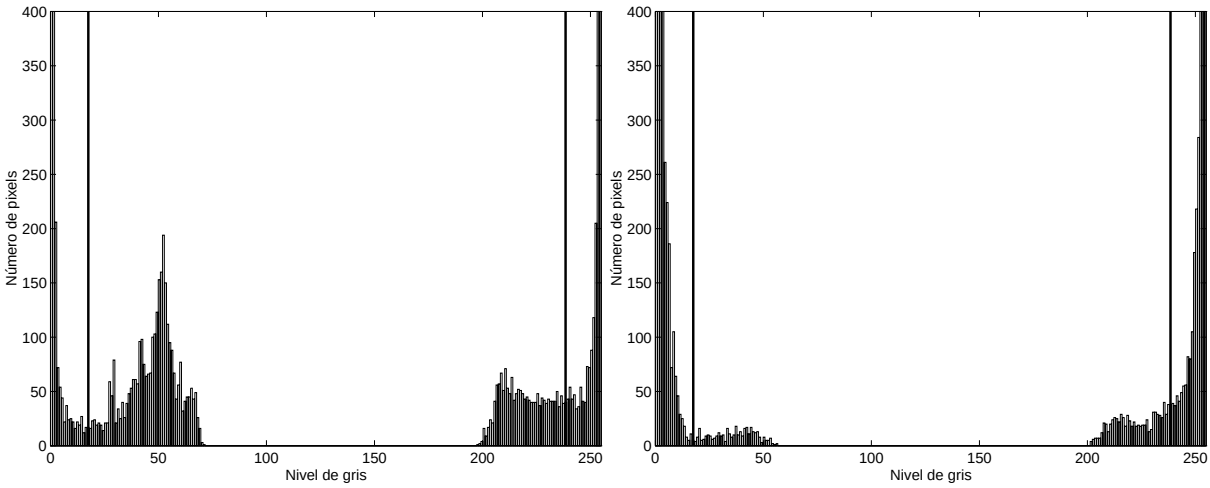


Figura 9.17: *Histogramas ejemplos de la la segunda derivada temporal para el experimento 2*

cripción de los resultados experimentales. Sí que se ha considerado interesante incluir la evolución de la cantidad de positivos en función de la imagen. Estos datos, conjuntamente con las imágenes iniciales intermedias y finales, muestran el resultado de los experimentos.

Experimento 2

En esta secuencia la dirección de movimiento del móvil forma un ángulo de 45^o con el eje óptico de la cámara, que marca a su vez la dirección de avance de la plataforma móvil. El pequeño objeto móvil (el coche con la caja que se puede observar en la figura 9.18) avanza con una velocidad de 0.055 m/s, avanzando la plataforma con una velocidad de 0.05 m/s.

La tabla 9.2 muestra los puntos detectados con movimiento propio para la secuencia del experimento 2. Al haberse procesado una de cada 4 imágenes aparecen en intervalos de 4. Inicialmente se detectan pocos puntos al estar el objeto parcialmente oculto y algo alejado, con lo que el movimiento refleja un menor desplazamiento en la imagen. Posteriormente van aumentando de forma gradual los puntos detectados al acercarse el objeto a la cámara hasta que empieza de nuevo a desaparecer. La figura 9.18 muestra las imágenes correspondientes a las imágenes 22, 62 y 102.

Nótese que si los puntos detectados están en la fóvea o en la parte central de la retina apenas ocupan superficie en el plano retínico a pesar de ser una gran cantidad de puntos en el plano cortical. Esto se ve con claridad en las imágenes centrales de la figura 9.18. Para la imagen 62 se han detectado casi 5.000 puntos con movimiento propio, lo

Imagen	Positivos	**Imagen**	Positivos	**Imagen**	Positivos
10	70	14	61	18	63
22	83	26	167	30	306
34	204	38	289	42	350
46	473	50	887	54	1.518
58	3.393	62	4.923	66	3.911
70	3.390	74	1.871	78	1.227
82	907	86	975	90	711
94	991	98	618	102	621
106	796	110	577	114	263

Tabla 9.2: *Puntos detectados en la secuencia 2*

Imagen	Positivos	**Imagen**	Positivos	**Imagen**	Positivos
58	121	62	155	66	114
70	2.037	74	2.702	78	1.650
82	959	86	583	90	538
94	541	98	1.869	102	2.780
106	2.263	110	1.070	114	690
118	628	122	457	132	549

Tabla 9.3: *Puntos detectados en la secuencia 3*

cual corresponde a más del 50% de la cantidad de puntos de la imagen foveal. Estos puntos al estar mayoritariamente en el centro de la imagen pertenecen a la fóvea y a la parte central de la retina, y por tanto ocupan una superficie relativamente pequeña (que en ningún caso llega a ser el 50% de la imagen).

Experimento 3

En este experimento la dirección de movimiento del móvil forma un ángulo de 90^o con el eje óptico de la cámara, que marca a su vez la dirección de avance de la plataforma móvil. El objeto móvil avanza con una velocidad de aproximadamente 0.055 m/s, y la plataforma con una velocidad constante de 0.05 m/s. Éste es el caso más sencillo para el algoritmo, ya que el desplazamiento en la imagen debido al movimiento perpendicular del coche será mayor que el desplazamiento en la imagen debido al movimiento del vehículo.

La tabla 9.3 muestra parte de los puntos detectados con movimiento propio en el experimento 3, observándose el resultado del algoritmo en la figura 9.19 .

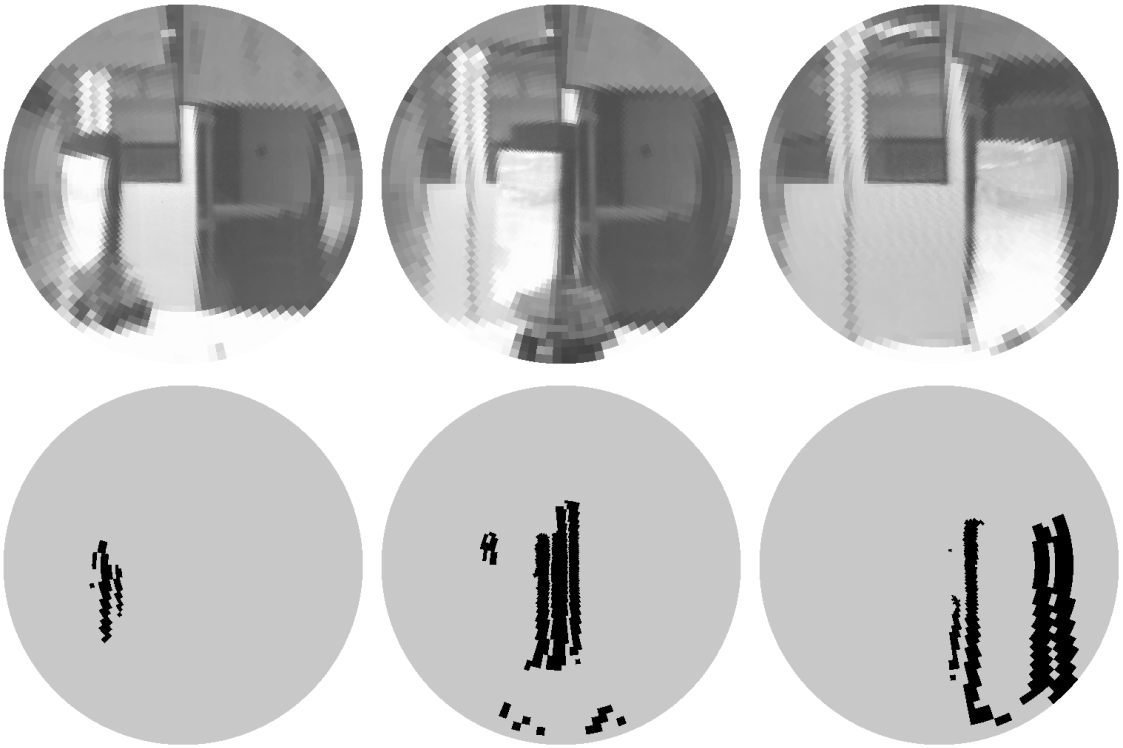


Figura 9.18: *Imágenes 22, 62 y 102 del experimento 2*

Experimento 4

Este último experimento será la prueba más difícil del algoritmo de detección de movimiento propio ya que en este caso el objeto avanza hacia la plataforma móvil, superponiéndose este movimiento al propio del robot que avanza hacia el objeto. En este caso cabe pensar que el algoritmo eliminará totalmente el movimiento del objeto sin detectarlo, ya que la dirección de movimiento relativo entre el objeto y la plataforma coincide con el eje óptico.

La eliminación de la componente transversal reduce en gran medida los puntos detectados, pero se siguen detectando puntos con movimiento propio. Estos puntos se detectan ya que el umbral está ajustado para eliminar el desplazamiento en la imagen cuando la velocidad es a lo largo del eje y de 0.05 $m/4$. Sin embargo la velocidad relativa entre objeto y plataforma móvil es mayor (de 0.103 m/s), por lo que el desplazamiento radial en la imagen será mayor que el que elimina el algoritmo de forma automática.

La tabla 9.4 muestra la cantidad de positivos obtenidos, que es mucho menor que los positivos obtenidos en los casos anteriores, pero siguen mostrando la presencia de un objeto con movimiento propio. En cualquier caso solamente en el caso de que la velocidad no tenga componente perpendicular a la dirección de avance del robot se producirá esta reducción de puntos detectados.

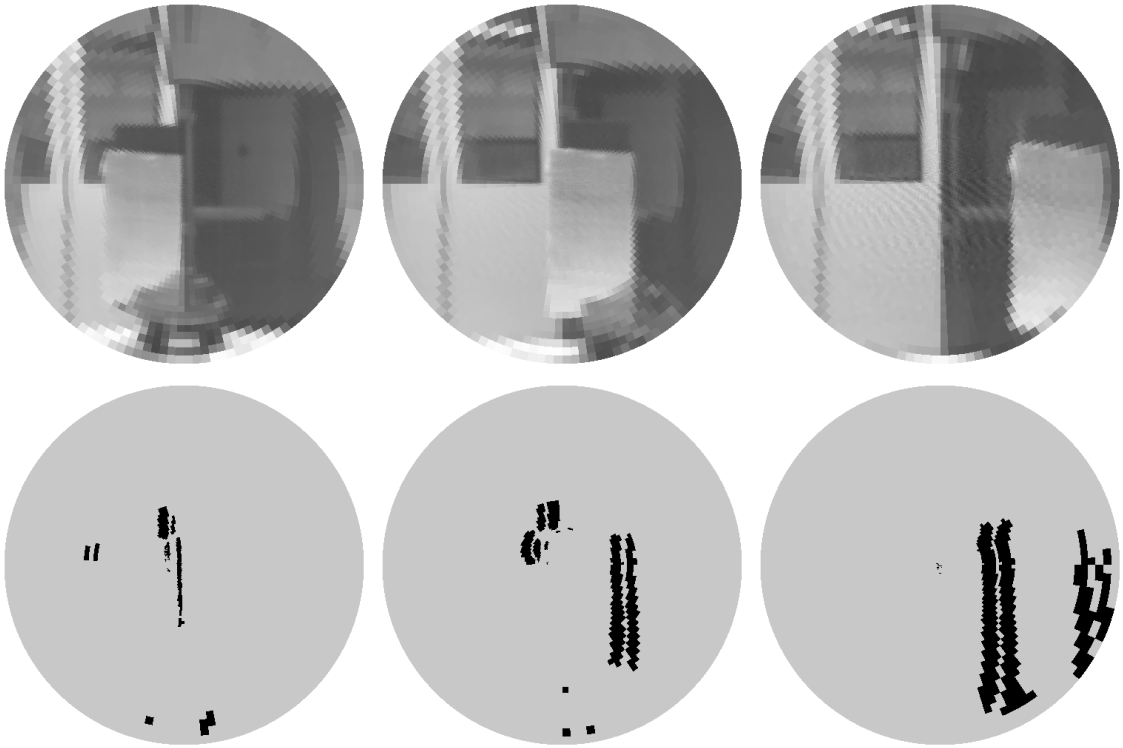


Figura 9.19: *Imágenes 66, 90 y 114 del experimento 3*

9.5 Conclusiones

El algoritmo diferencial en coordenadas log-polares desarrollado teóricamente en el capítulo 6 realiza varias aproximaciones cuya validez es necesario comprobar experimentalmente. El hecho de que el suavizado no sea lo suficientemente bueno (máscara de 6 bits de valor unitario) y de que se haya realizado la aproximación de la solución de movimiento suave en la ecuación (6.12) hace necesaria la aparición de un umbral en el algoritmo.

Se han mostrado los resultados parciales de la etapa de suavizado con 8 bits y 7 bits. Esta etapa de forma colateral oscurece la imagen, ya que utiliza una máscara de 6 bits y sin embargo se divide la suma entre 16 para almacenar el resultado en 7 bits.

Por otra parte, el hecho de que se puedan adquirir y procesar imágenes con muy altas velocidades debe ser tenido en cuenta al utilizar etapas diferenciales. Si el intervalo de adquisición entre imágenes es relativamente pequeño comparado con la velocidad del vehículo puede ocurrir que dos imágenes consecutivas sean casi iguales, con lo que la diferenciación no tiene sentido. De manera adicional la especial geometría log-polar con tamaño de *pixel* espacio-variante requiere el análisis realizado en la sección 8.4. A partir de las ecuaciones (9.1) y (9.2) se ha caracterizado cual es la relación entre la

Imagen	Positivos	**Imagen**	Positivos	**Imagen**	Positivos
82	6	86	235	90	150
94	61	98	43	102	161
106	439	110	340	114	250
118	465	122	1.132	126	1.102
130	356	134	237	138	225
142	177	146	266	150	295
154	364	158	438	162	473

Tabla 9.4: *Puntos detectados en la secuencia 4*

velocidad de la plataforma móvil y el intervalo de adquisición de imágenes.

Con la intención de parametrizar la validez del algoritmo en función de la velocidad se han tomado secuencias de imágenes desde la plataforma móvil con velocidad constante. De esta manera a una velocidad de la plataforma de 0.05 m/s se han tomado 12 imágenes por segundo con lo que Δt =0.08$\hat{3}$ segundos. Procesar una imagen de cada 2 es equivalente a procesar todas las imágenes pero con el vehículo avanzando al doble de velocidad. La relación es por tanto biunívoca.

La ecuación (9.1) indica el intervalo de adquisición en el caso más estricto en el que la imagen de un objeto avance entre *pixels*. No es necesario este tiempo para que se produzca variación entre imágenes. La ecuación (9.2) indica el intervalo temporal de adquisición en función del nivel de grises del objeto y del fondo $T_{GO} - T_{GF}$. Según esta ecuación este intervalo temporal depende de cada experimento concreto, siendo el caso en el que la diferencia de tono de grises entre el objeto y el fondo sea mínima (valor 1) el caso en el que será necesario más tiempo para que se produzca la variación en la imagen log-polar. Es por tanto interesante realizar un supuesto genérico de diferencia de tono de grises y a partir de este supuesto obtener un Δt. Esta diferencia será mayor que 1, pero no muy grande para no imponer que el contraste entre objeto y fondo en los experimentos sea muy grande. En cualquier caso la ecuación (9.2) muestra la relación que se tiene que cumplir entre la velocidad del objeto v, la diferencia del nivel de grises $T_{GO} - T_{GF}$ y Δt. Cabe hacer notar que esta ecuación ha sido deducida para que produzca una diferencia de nivel de gris mayor que 1 en el plano imagen. Si la diferencia real de nivel de gris entre el objeto y el fondo es mayor que la diferencia supuesta, entonces la diferencia producida en el nivel de gris de la imagen será mayor que 1 y por tanto también detectada por el algoritmo diferencial.

Con estas consideraciones y teniendo en cuenta el resto de parámetros de la escena se ha escogido un valor mínimo de $T_{GO} - T_{GF}$ de 5 unidades. Este valor es suficientemente pequeño (el objeto muestra una diferencia mayor de tono de gris con el fondo) y hace

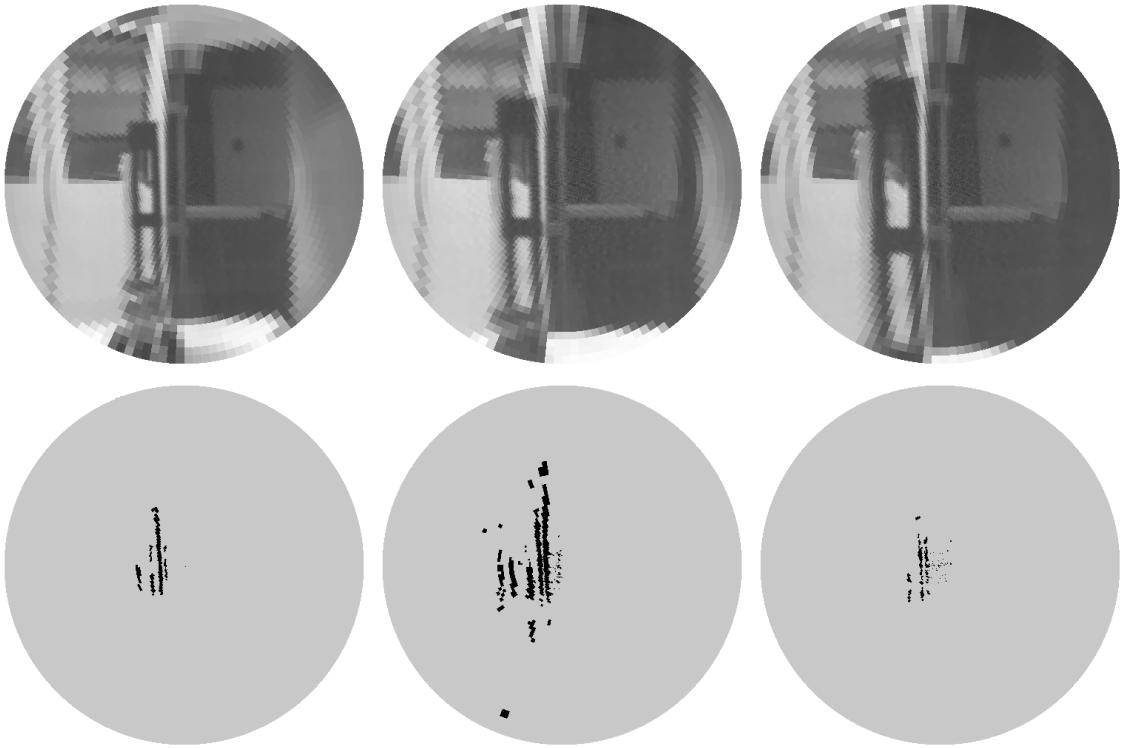


Figura 9.20: *Imágenes 86, 122 y 158 del experimento 4*

que deba procesarse una imagen cada 0'3 segundos, es decir, se procese una de cada 4 imágenes. El intervalo temporal entre imágenes queda de esta manera automáticamente fijado en función de la velocidad de la plataforma móvil si se supone esta diferencia mínima entre tonos de gris. De esta manera se ha mostrado el resultado parcial de la segunda y tercera etapa (prescindiendo de la binarización).

El algoritmo tiene que eliminar de forma automática el desplazamiento en la imagen debido al avance de la plataforma. Esto se consigue mediante la fijación de un umbral de manera que si las diferencias son menores que dicho umbral se elimine ese desplazamiento. La magnitud del desplazamiento en la imagen debido al movimiento propio del robot dependerá de la velocidad de éste. De esta manera al haber fijado el intervalo temporal de adquisición en función de la velocidad del robot se podrá parametrizar el umbral en función de dicho intervalo.

La forma de fijar el umbral ha sido mediante el experimento 1 en el que no había objetos con movimiento propio y solamente avanza la plataforma móvil con velocidad constante. En ese caso el algoritmo tiene que eliminar este desplazamiento en las imágenes procesadas, estando fijado el intervalo de adquisición de manera que se garantice una variación en las imágenes originales. Se ha encontrado una manera de sistematizar la elección del umbral y se ha aplicado a diversos experimentos.

Si la velocidad del objeto es similar a la del vehículo autónomo, y ambos se mueven perpendicularmente, el objeto será detectado con claridad sin detección de falsos positivos. En el caso de que las direcciones de movimiento sean paralelas la cantidad de puntos detectados es menor, aunque el algoritmo sigue evitando la aparición de falsos positivos.

El algoritmo será capaz de detectar objetos en movimiento propio independiente del movimiento del vehículo autónomo siempre que se oriente el eje óptico de la cámara hacia la dirección de avance. Se ha estudiado la elección del intervalo de adquisición y del umbral a partir de la velocidad de la plataforma móvil, llegando a ecuaciones que sistematizan esta elección.

En trabajos posteriores se integrará la información suministrada por el algoritmo de detección con la información suministrada por otros sensores. A partir de esta información se estudiará la planificación de trayectorias en entornos no estructurados. Es entonces cuando se valorará el peso real de la información suministrada por el algoritmo de detección de movimiento. En cualquier caso los resultados obtenidos apuntan a que esta información puede ser muy valiosa para que la plataforma móvil pueda detectar objetos en movimiento estando ésta moviéndose. De esta manera sería posible la utilización de estos datos para planificar la trayectoria y evitar choques.

Capítulo 10

Algoritmo de cálculo del tiempo al impacto

10.1 Introducción

En el capítulo 7 se ha propuesto una implementación del algoritmo diseñado por Tistarelli y Sandini para el cálculo del tiempo al impacto en coordenadas log-polares [TS91b]. La coincidencia del centro del sensor con el centro de expansión óptica debido al avance de la plataforma móvil permite la eliminación de la componente angular del campo de velocidades en el sensor óptico. De esta manera sólo es necesario computar la componente radial de la velocidad v_ξ o equivalentemente, utilizando la ecuación del flujo óptico, calcular el cociente entre el gradiente radial de la imagen $\partial I/\partial \xi$ y la derivada temporal $\partial I/\partial t$.

Esta aproximación se ha implementado en el cauce reconfigurable utilizando tres EPs que realizan respectivamente: el suavizado de la imagen, el cálculo del gradiente radial y de la derivada temporal, y por último la división entera de estos valores.

Diversos experimentos han mostrado las limitaciones del presente algoritmo así como bajo qué condiciones se pueden esperar resultados suficientemente precisos como para ser utilizados en situaciones reales. En la sección 10.2 se describen las correcciones del algoritmo que han sido realizadas para obtener resultados más fiables. En la sección 10.3 se muestran los resultados obtenidos realizando estos ajustes y se realiza un análisis detallado de estos datos.

10.2 Ajuste del algoritmo y descripción de los experimentos

Previamente al diseño e implementación del algoritmo en el cauce reconfigurable se han realizado diversas experiencias con imágenes sintéticas simulando los resultados. Estos experimentos han mostrado la necesidad de realizar un suavizado que elimine de forma casi total la presencia de picos en la imagen log-polar. La utilización de la ecuación del flujo óptico para reducir el cálculo de τ (tiempo al impacto) a una división entre derivadas hace necesario que estas parciales estén bien definidas en cada punto de la imagen log-polar. Esto implica la no existencia de funciones escalón en la imagen. Análogamente es necesario garantizar que exista una cierta variación en la imagen, tanto espacial como temporal, para de esta manera poder calcular ambas derivadas. Experiencias con imágenes sintéticas utilizando el cauce reconfigurable han mostrado una gran precisión con objetos con un gradiente radial constante.

Una fuente de error es que el algoritmo realiza por simplicidad una división entera de dos derivadas, estando de esta manera discretizados los valores posibles de τ. Por otra parte, la propia expresión de τ hace que errores pequeños en la derivada temporal, al estar ésta en el denominador, puedan producir errores grandes en τ.

El cálculo de un valor numérico más preciso de τ puede ser abordado realizando la media de los valores obtenidos de una superficie. Este promedio lo realizará la partición software, sumando el tiempo al impacto de cada punto en el que ha sido calculado, y dividiendo (división con números reales) esta cantidad entre el número de puntos que han contribuido.

Los valores de τ se calculan internamente en el último EP mediante la división del gradiente espacial entre la derivada temporal. El hecho de que esta división sea entera hace que se pierda precisión en los valores de τ calculados por el último EP. Para evitar este efecto y ganar precisión es necesario multiplicar la derivada temporal por un factor constante. Después, en la partición software se tendrá en cuenta este factor de escalado en los valores de τ y se realizará la división (en este caso con la precisión de los números reales). Sin embargo esta constante de multiplicación previa a la división no debe desbordar el resultado en el numerador. Una inspección de los valores del gradiente obtenidos y diversas pruebas han mostrado que valores adecuados para la constante de multiplicación están entre 10 y 20. Para acelerar la multiplicación la constante se ha elegido como 16. De esta manera esta multiplicación se realiza en el mismo EP que calcula la división sin más que desplazar 4 bits a la izquierda el gradiente. De manera adicional, la expresión de τ para la retina no consiste solamente en el cociente del gradiente radial entre la derivada temporal. También aparece como

constante de división el coeficiente de crecimiento del sensor log-polar $B \approx 0'05$, o equivalentemente una constante de multiplicación $B^{-1} \approx 20$. De esta manera el mapa de los valores de τ que se obtendrán del tercer EP tendrán un valor no muy lejano al valor correcto. Posteriormente se realiza por software el ajuste multiplicando por 1'25 (20/16).

Se han realizado diversas experiencias con el robot aproximándose hacia objetos y superficies irregulares en el laboratorio obteniéndose resultados poco precisos para τ. El hecho de que el suavizado que realiza el primer EP no sea excesivamente bueno tiene una gran influencia en la dispersión de los valores calculados para τ, resultando unos valores generalmente decrecientes pero de un valor numérico incorrecto. Un suavizado mayor, realizado con más EPs o con una etapa más lenta, generaría mapas de τ con menor dispersión y resultados más correctos, tal como han mostrado los experimentos con imágenes sintéticas.

En cualquier caso, y con la intención de probar la validez del algoritmo, se ha diseñado un experimento en el que la plataforma móvil se aproxima hacia una superficie con un cierto gradiente especialmente preparado para la experiencia. Se ha medido la velocidad del robot, la distancia inicial cuando se empezó a tomar imágenes, adquiriendo una imagen cada 0.08 segundos, para de esta manera poder probar la exactitud del valor numérico obtenido para τ. La figura 10.1 muestra de forma esquemática los parámetros del experimento realizado.

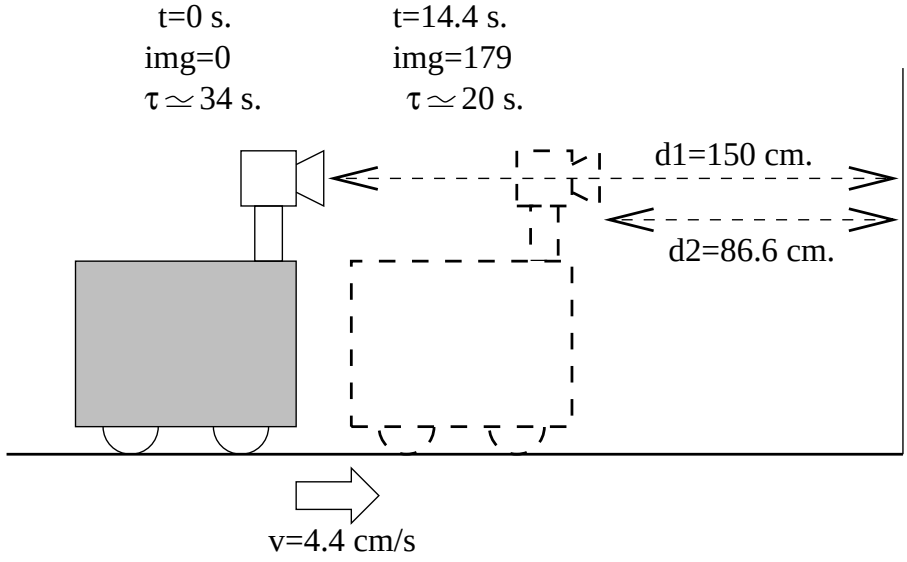


Figura 10.1: *Esquema del experimento del cálculo del tiempo al impacto*

La plataforma móvil avanza hacia la pared con una velocidad constante de aproximadamente 4.4 cm/s. En el instante en el cual la distancia hasta la pared es de 1.5 metros se inicia la adquisición de imágenes, a razón de 12.5 imágenes por segundo

(o lo que es lo mismo una imagen cada 0.08 segundos). Esta adquisición finaliza tras haber almacenado un total de 180 imágenes, estando éstas numeradas desde la imagen 000 hasta la 179. La figura 10.2 muestra 3 imágenes de la secuencia en las que se puede observar el aspecto de esta superficie. En las siguientes secciones se presentan los resultados parciales tras el procesado de cada etapa, y los resultados globales del cálculo del tiempo al impacto en este experimento.

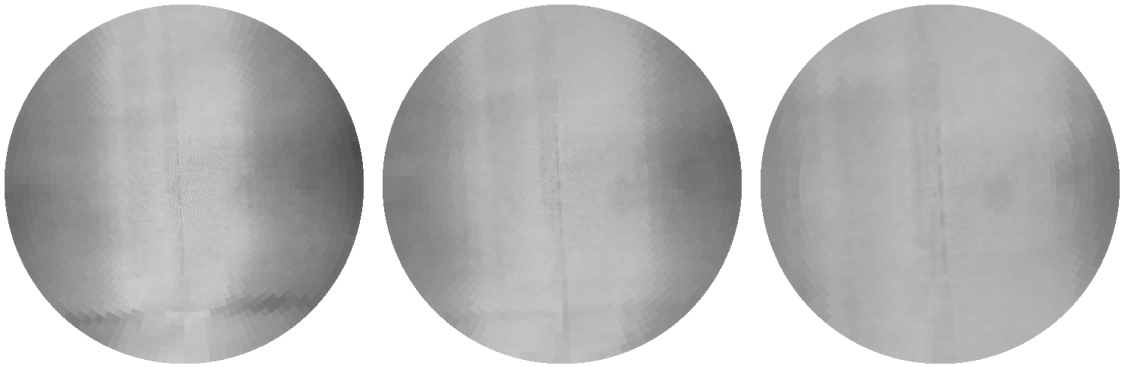


Figura 10.2: *Secuencia original del experimento del cálculo del tiempo al impacto. Imágenes 8, 92 y 176 de la secuencia*

10.3 Resultados

10.3.1 Etapa de suavizado

La imagen original está suficientemente suavizada, ya que la superficie hacia la que se acerca la plataforma móvil ha sido especialmente diseñada para el experimento. La superficie original muestra un gradiente horizontal de manera que la imagen es más clara en el centro de la imagen que en los extremos. Se ha mantenido la etapa del suavizado radial descrita en la sección 6.4.1, utilizándose en este caso un suavizado seleccionando 8 bits para la salida, ya que no se va a producir desbordamiento al calcular el gradiente y la derivada temporal (imagen original muy suavizada). Adicionalmente se ha repetido el experimento de avance hacia esta superficie eliminando la etapa de suavizado y los resultados han sido muy similares. En cualquier caso se ha mantenido esta etapa, ya que la aplicación de este algoritmo requiere de un fuerte suavizado y no ralentiza la obtención de resultados. La figura 10.3 muestra la salida de esta primera etapa.

Figura 10.3: *Resultado de realizar un suavizado de 8 bits de la secuencia de la figura 10.2*

10.3.2 Cálculo de la derivada espacial y temporal

De nuevo se plantea cuál ha de ser el intervalo de adquisición mínimo que garantice la variación entre imágenes. En este caso las ecuaciones (9.1) y (9.2) que definen el intervalo mínimo de adquisición de imágenes para que se produzca variación no son aplicables, ya que no hay un borde de un objeto distinto del fondo que invada el *pixel* del anillo siguiente o avance dentro del propio *pixel*. En cualquier caso se ha utilizado inicialmente el mismo intervalo de adquisición de imágenes que en el experimento del capítulo 9 (una de cada cuatro imágenes o equivalentemente una imagen cada 0.32 segundos). Este intervalo produce una variación temporal en una cantidad suficiente de puntos para calcular τ. Intervalos temporales ligeramente mayores (hasta una de cada 10 imágenes o equivalentemente una cada 0.8 segundos) no han producido variación apreciable en los resultados experimentales, ya que se garantiza que la variación se produce entre *pixels* vecinos. Un intervalo mayor de adquisición suministra resultados incorrectos al no corresponder el cálculo del gradiente (entre *pixels* de anillos vecinos) con el de la derivada (entre *pixels* que habrán avanzado más de un *pixel*). Intervalos de adquisición menores han suministrado una pequeña cantidad de puntos con variación temporal, con lo que el cálculo de τ se reduce a unos pocos puntos cuyo valor medio no es muy correcto.

El segundo EP calcula simultáneamente la derivada espacial y temporal a partir de la imagen suavizada y se la suministra al EP que va a realizar la división. Se han diseñado especialmente etapas que calculan sólo el gradiente radial y sólo la derivada temporal con el objeto de mostrar el resultado parcial de cada operación a partir de la secuencia del experimento. En la figura 10.4 se muestra el gradiente radial de la secuencia original de la figura 10.2. En esta figura se puede observar como las diferencias son pequeñas entre los diversos puntos, al obtenerse casi únicamente puntos negros

(diferencia nula), casi negros (diferencia positiva pequeña) o puntos blancos (diferencia negativa pequeña). Claramente la zona central se corresponde con una zona con pequeñas variaciones positivas radiales del nivel de gris (puntos mayoritariamente negros). Sin embargo, la zona más externa se corresponde con pequeñas variaciones radiales negativas del nivel de gris, ya que la secuencia original presenta un oscurecimiento gradual en la periferia. En esta zona, al realizar la resta del valor de gris de un *pixel* con el del anillo inmediatamente inferior (más claro y por tanto con un nivel de gris mayor), el resultado es negativo con lo que en complemento a 2 se obtienen valores cercanos a 255 (casi blancos). La derivada temporal tiene un aspecto similar al ser las diferencias entre imágenes pequeñas. Todos los pixels presentan un nivel de gris cercano a cero, o bien positivo, o bien negativo.

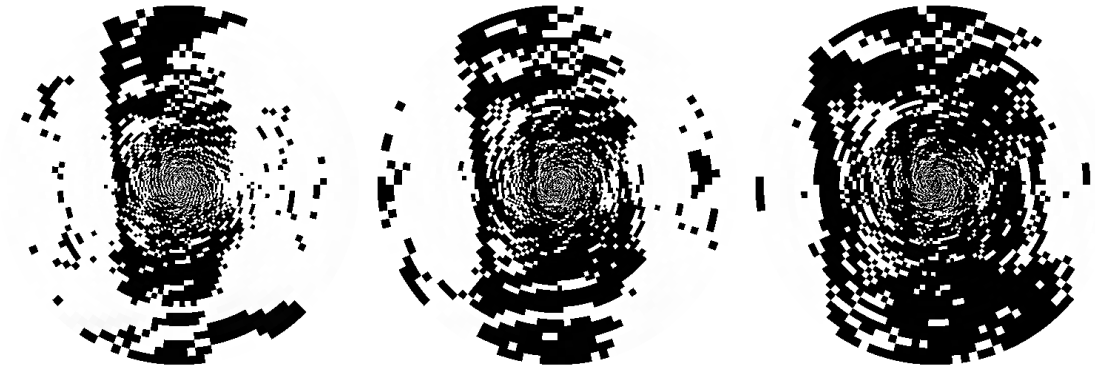


Figura 10.4: *Resultado de calcular el gradiente radial de la secuencia de la figura 10.2*

10.3.3 Mapas de tiempo al impacto

La salida del último EP genera los mapas de los valores calculados para el tiempo al impacto. Valores enteros calculados en los puntos en los que se ha podido calcular, ya que se han considerado como puntos no válidos aquellos cuyo valor del tiempo al impacto fuese negativo o no estuviese bien definido (gradiente o derivada nula). Cabría esperar que los mapas de tiempo al impacto de las imágenes iniciales fueran más claros, ya que el tiempo al impacto es mayor, y que los mapas de tiempo al impacto de las imágenes finales fueran más oscuros. De hecho, debido a la gran dispersión de valores obtenidos, con una mera inspección visual de los mapas de tiempo al impacto no se muestra este comportamiento. Posteriormente, en la sección 10.3.4, se realizará un análisis más detallado de los mapas del tiempo al impacto mostrando los histogramas obtenidos.

La figura 10.5 muestra los mapas de tiempo al impacto correspondientes a la secuencia de la figura 10.2. Cabe hacer notar que el tono de gris constante que se puede

apreciar en el fondo corresponde a los puntos en los que τ no ha sido calculado y que, por tanto, no han contribuido al valor medio de τ para cada imagen. Los puntos de la fóvea igualmente han sido eliminados, ya que en ellos no se sigue la distribución log-polar y, por tanto, el resultado del valor de τ calculado por el último EP para estos puntos no es correcto.

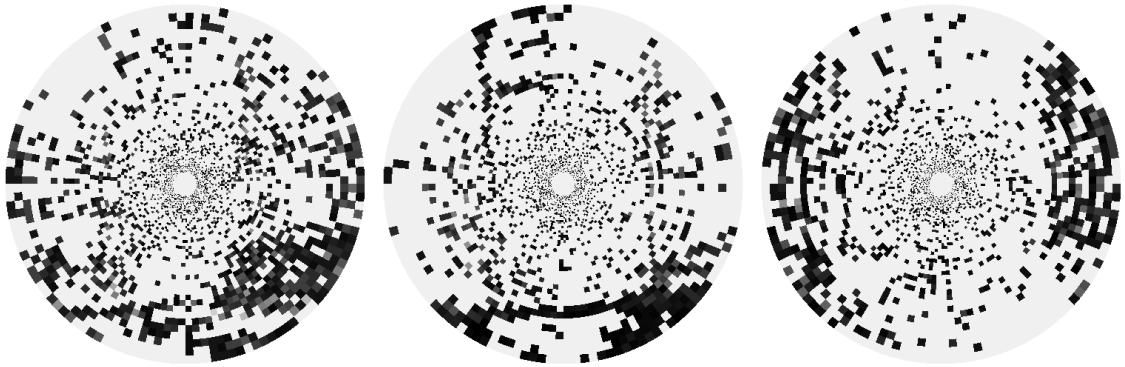


Figura 10.5: *Mapas de tiempo al impacto correspondientes a la secuencia de la figura 10.2*

A partir de los mapas de tiempo al impacto suministrados por el cauce reconfigurable, se ha realizado por software el escalado correcto, y se ha calculado el valor medio para cada imagen. La figura 10.6 muestra los resultados experimentales al calcular la media de todos los puntos en los que se ha calculado τ. La línea discontinua corresponde al resultado teórico y la línea continua muestra el ajuste de los datos experimentales a una recta mediante regresión lineal.

La recta que se muestra en la figura 10.6 muestra los valores medios de τ calculados con cerca de 2000 puntos para cada imagen. De hecho, los mapas de tiempo al impacto que se muestran en la figura 10.5 corresponden a las imágenes 8, 92 y 176. Los valores medios de τ en estos casos han sido calculados con 2007, 1669 y 1643 puntos. La tabla 10.1 muestra el número de puntos que han contribuido para el valor medio de τ en función del número de la imagen.

Se puede observar como el error relativo es inicialmente pequeño, ya que cuando el tiempo al impacto es grande la divergencia entre el valor teórico y el valor experimental obtenido es pequeña (menor del 6%). Sin embargo se observa como la diferencia (y por tanto el error relativo) crece cuando τ disminuye, llegando al final de la secuencia a ser del 25%. Es necesario por tanto un estudio más detallado de los mapas de tiempo al impacto para poder analizar el origen de este error.

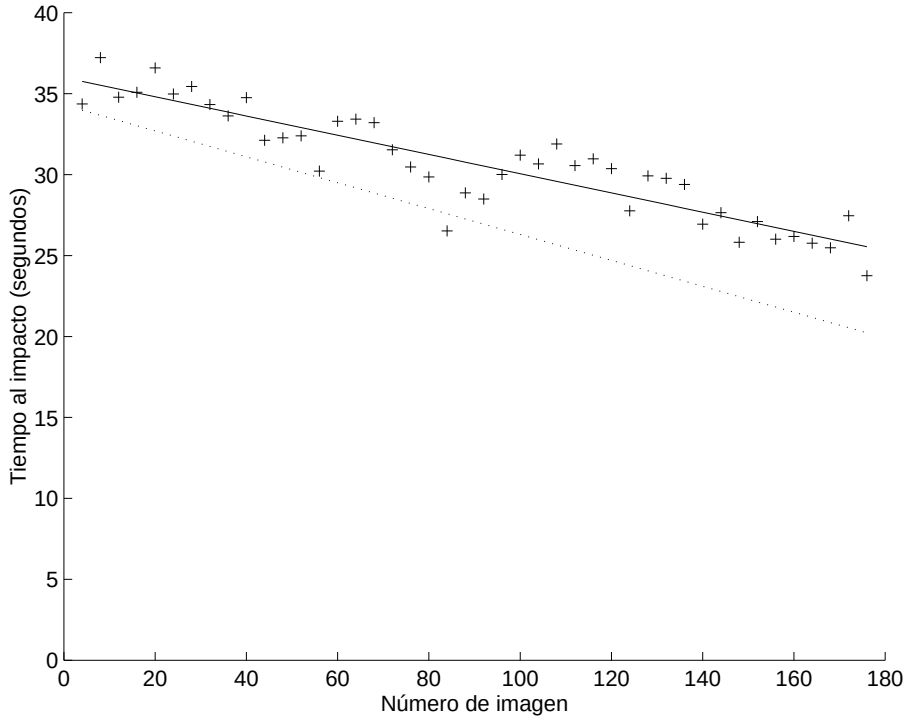


Figura 10.6: *Resultados experimentales y teóricos (línea discontinua) del tiempo al impacto en función del número de imagen*

10.3.4 Análisis de los resultados

Los valores de τ experimentales que se muestran en la figura 10.6 han sido calculados con la contribución de cerca de 2000 puntos para cada uno de ellos. Cabría por tanto suponer que el valor obtenido debería ser bastante correcto. Si se analiza el histograma de los valores obtenidos se podrá entender el comportamiento del valor medio de τ. En la figura 10.7 se muestran los histogramas correspondientes a los mapas del tiempo al impacto de la imagen 8, y de la imagen 176.

Cabe hacer notar cómo la dispersión de los valores es bastante grande, presentándose un decrecimiento exponencial en la frecuencia de aparición de valores altos. Los picos en los valores múltiplos de 10 (especialmente en el valor máximo 20) se corresponde con valores de τ calculados con gradientes de 1, 2, 3 etc, y valores de la derivada temporal unitarios multiplicados por la constante B^{-1}. El hecho de que, internamente ya se haya realizado una multiplicación por 16 previa a la división del gradiente entre la derivada temporal, permite que se obtengan valores intermedios a los múltiplos de 16. Posteriormente, y ya fuera del cauce reconfigurable, se reajusta el escalado multiplicando por $\frac{16}{B}$ para que el resultado sea correcto.

Si se realiza un análisis comparativo de los dos histogramas que se muestran en la

Img	Nº de ptos	**Img**	Nº de ptos	**Img**	Nº de ptos	**Img**	Nº de ptos
4	2319	8	2007	12	2285	16	2079
20	1963	24	1878	28	1970	32	1980
36	1840	40	2085	44	1848	48	1791
52	1653	56	2169	60	1708	64	1943
68	1762	72	1773	76	2167	80	1810
84	1682	88	2399	92	1669	96	1720
100	2016	104	1847	108	1812	112	1724
116	1781	120	1968	124	1695	128	1707
132	1799	136	1666	140	1689	144	1798
148	1703	152	1673	156	1757	160	1677
164	1688	168	1649	172	1641	176	1643

Tabla 10.1: *Número de puntos de los mapas de tiempo al impacto que contribuyen a la media experimental de los valores de τ que se muestra en la figura 10.6*

figura 10.7 se puede observar como la disminución del valor medio de τ se debe a que aumenta la cantidad de puntos que contribuyen con valores pequeños. Sin embargo se sigue manteniendo una cantidad de puntos no despreciable que contribuyen con valores altos al cálculo de τ. Estos valores pueden considerarse como erróneos y se deben a que valores grandes del gradiente han sido divididos por valores muy pequeños de la derivada temporal.

Adicionalmente es posible observar como en el segundo histograma, al aumentar la frecuencia de aparición de valores pequeños, la contribución de la parte baja de la curva se ve cortada al llegar al valor cero. De hecho, cabe pensar que si la distribución estuviera desplazada más hacia la derecha (multiplicados los valores por un valor mayor que 20) los valores más bajos de τ compensarían los valores más altos debidos a las derivadas temporales pequeñas.

A partir de los histogramas de la figura 10.7 y tras haber encontrado que el origen de la dispersión de los valores se debe a valores de la derivada temporal que son incorrectamente pequeños, se puede plantear limitar la contribución de los puntos con derivada temporal pequeña. De esta manera se reducirá el número de puntos que contribuyen con gran peso, cortándose el decrecimiento exponencial de la distribución.

Se ha modificado el último EP del cauce para que realice la división solamente si el módulo de la derivada temporal (M) es mayor que una cierta cantidad. En la figura 10.8 se muestran las diversas rectas obtenidas para τ tras la regresión lineal si $M \geq 2$, $M \geq 4$ y $M \geq 6$. No se han mostrado las rectas obtenidas con valores intermedios $M \geq 3$ y $M \geq 5$ con la intención de poder mostrar los datos experimentales con mayor

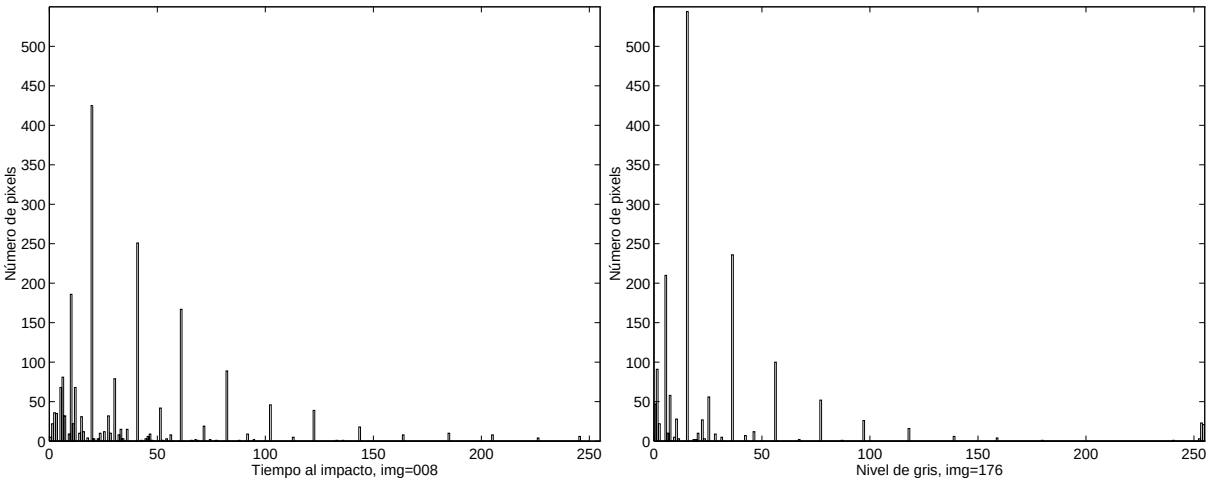


Figura 10.7: *Histogramas de tiempo al impacto de las imágenes 8 y 176*

claridad.

En la figura 10.8 la línea discontinua muestra el comportamiento teórico esperado para el tiempo al impacto, y las líneas continuas muestran los resultados experimentales del tiempo al impacto en función del número de imagen y, de abajo a arriba para $M \geq 2$, $M \geq 4$ y $M \geq 6$. El comportamiento de los datos experimentales en los que se ha dividido si $M \geq 2$ muestra una recta con una pendiente similar a la línea teórica, con un error relativo inicial del 6% que llega a ser del 15% al final de la secuencia. Cabe hacer notar como el error absoluto se mantiene casi invariable para cada recta, siendo los valores experimentales menores que el valor teórico.

La recta en la que se tienen en cuenta los puntos sólo si $M \geq 2$ muestra unos errores relativos inicialmente iguales que en la recta que se muestra en la figura 10.6, pero cuando avanza la secuencia los errores relativos son apreciablemente mejores que en el caso de ésta. Cabría esperar unos resultados mejores si sólo se divide cuando el módulo de la derivada temporal es aun mayor. El resto de rectas experimentales de la figura 10.8 muestran como la pendiente apenas varía, pero los valores experimentales son más pequeños que los teóricos. De esta manera, conforme disminuye el valor de M los valores experimentales difieren más de los teóricos. Este comportamiento se puede analizar observando los histogramas de la figura 10.9. En esta figura se muestra como el decrecimiento exponencial en la distribución de los valores de τ ha sido eliminado. Sin embargo se tiene el efecto negativo de que disminuye significativamente la cantidad de puntos que contribuyen al cálculo de τ, y no se compensan suficientemente los valores bajos de la distribución al haberse eliminado los valores altos. De hecho, se ha mantenido el escalado en la figura 10.9 igual al de la figura 10.7 para que se puedan comparar ambas gráficas.

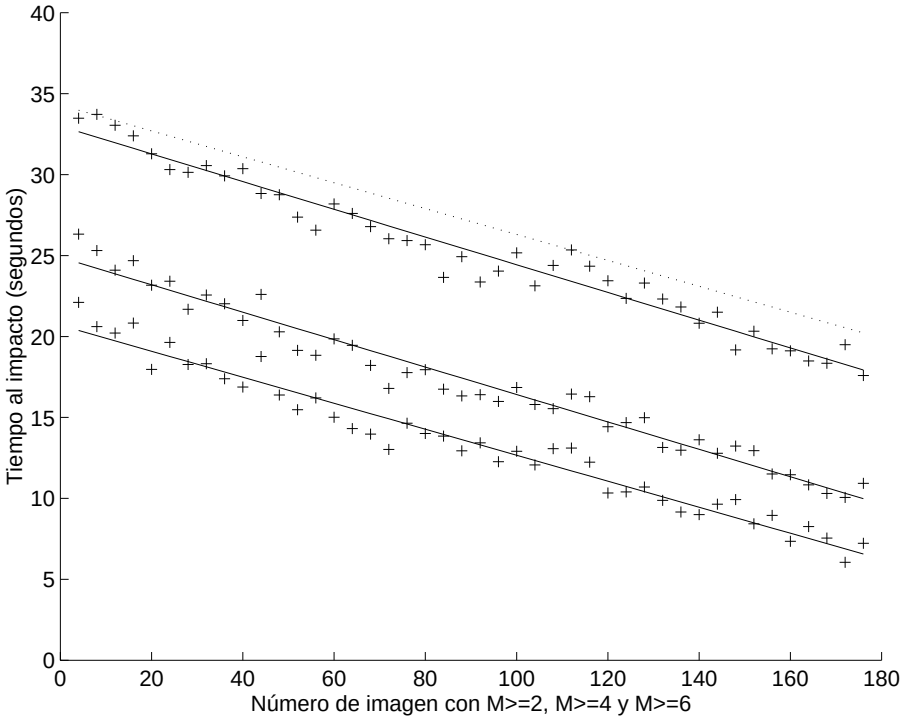


Figura 10.8: *Resultados teóricos y experimentales del tiempo al impacto en función del número de imagen. Módulo de la derivada temporal* $\geq$ *2,* $\geq$ *4 y* $\geq$ *6 (respectivamente y de arriba a abajo)*

10.4 Conclusiones

Los diversos experimentos previos han sido útiles para evaluar la sensibilidad del algoritmo ante imágenes no suficientemente suavizadas. Experiencias con secuencias de imágenes en las que la plataforma móvil se aproxima a superficies irregulares, han mostrado que el suavizado realizado por el primer EP no genera valores numéricos para τ suficientemente correctos. Sin embargo, experiencias con imágenes log-polares sintéticas (muy suavizadas) probadas con el cauce reconfigurable han mostrado que el algoritmo suministra valores numéricos bastante exactos de τ.

Es posible diseñar una etapa que realice un suavizado mayor de la imagen que el realizado por el EP que se utiliza en el algoritmo de detección de movimiento. A pesar de esto, y con la intención de probar la validez del algoritmo de cálculo de tiempo al impacto con la etapa de suavizado ya diseñada, se ha realizado un experimento en el que la plataforma móvil se acerca hacia una superficie con un gradiente horizontal. La secuencia total de imágenes es de 14.4 segundos, habiéndose tomado un total de 180 imágenes.

Los resultados obtenidos con la mera aplicación del algoritmo para obtener los mapas

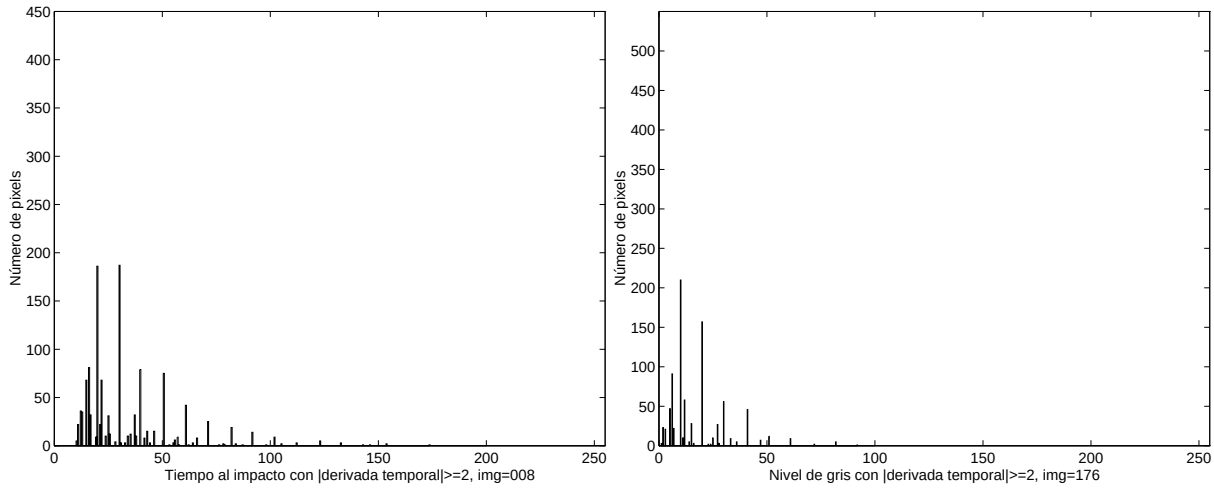


Figura 10.9: *Histogramas de tiempo al impacto de las imágenes 8 y 176. Módulo de la derivada temporal* $\geq$ *2*

de tiempo al impacto, y la posterior media de los puntos calculados, presenta un error absoluto de cerca de 2 segundos (error relativo del 6%) al inicio de la secuencia y de cerca de 5 segundos (error relativo del 25%) al final de la secuencia.

Un análisis detallado de los histogramas de los mapas del tiempo al impacto obtenidos muestran el origen de este error. La disminución exponencial de la distribución de los valores hace que exista una cantidad no despreciable de valores de τ grandes. Cuando la plataforma avanza y el tiempo al impacto debe disminuir, el peso de estos valores incorrectos hace que τ no decrezca de manera adecuada. Si se analiza la forma de la ecuación del tiempo al impacto, se observa que si la derivada temporal tiene un valor incorrectamente pequeño para valores grandes del gradiente, el valor de τ será anormalmente grande. Esta contribución errónea se puede eliminar si se obliga a que el error relativo de la derivada temporal sea pequeño, o equivalentemente, el valor absoluto de la derivada temporal sea mayor que un cierto valor. De esta manera los resultados experimentales han mostrado ser más correctos si sólo se realiza el cálculo de τ en los puntos en los que la derivada temporal sea mayor que 2. Sin embargo, si se aumenta aun más el valor mínimo de la derivada temporal para realizar el cálculo de τ, se reduce el número de puntos que contribuyen a este cálculo a unos pocos cientos. El error en la derivada espacial tiene un un menor peso al estar en el numerador en la expresión de τ.

El algoritmo de tiempo al impacto, implementado en el cauce reconfigurable, muestra resultados correctos en un entorno en el que las paredes tengan un cierto gradiente, sin presentar irregularidades en la imagen. La corrección en la que se calculan los puntos solamente en el caso en el que el módulo de la derivada temporal sea mayor o igual que 2, ha planteado unos resultados con errores relativos aceptables si la plataforma

móvil está lejos de la superficie.

Como conclusión global el algoritmo implementado en el cauce reconfigurable ha mostrado ser útil en ciertos casos. La mejora del comportamiento genérico del algoritmo se puede plantear desde diversos frentes: por una parte realizar un suavizado más perfecto de las imágenes log-polares; por otra parte un estudio estadístico más detallado de los mapas de tiempo al impacto puede suministrar un valor de τ más correcto que la simple media de los puntos calculados. Este análisis, a realizar por la partición software, podría requerir un tiempo de CPU no despreciable que ralentizaría la atención a otros procesos del robot. En cualquier caso un análisis complejo de los datos podría disminuir el *ratio* de imágenes procesadas por segundo.

Parte IV

Conclusiones globales

Capítulo 11

Conclusiones y trabajo futuro

En este capítulo se resume el trabajo de investigación ya realizado, se enumeran las aportaciones que se han realizado al estado de la investigación, y se plantean las futuras líneas de trabajo como continuación de la investigación presentada.

11.1 Sumario

11.1.1 Origen y planteamiento del trabajo de investigación

El presente trabajo de investigación parte de la confluencia de varias líneas de trabajo en el seno del Departamento de Informática de la Universitat de València. Por una parte la experiencia en dispositivos lógicos programables y lenguajes de descripción del hardware. Por otra parte la línea de visión log-polar introducida por el Dr. Fernando Pardo tras el desarrollo del sensor log-polar CMOS en el IMEC (Bélgica). El contacto en el seno de la misma universidad con el grupo emergente de navegación de vehículos autónomos, ha planteado el desarrollo de sistemas útiles para navegación de plataformas móviles.

Dentro de los posibles métodos de sensorización de una plataforma autónoma destaca la sensorización visual, ya que con este tipo de sensorización se puede conseguir simultáneamente largo alcance y precisión. Sin embargo, como desventaja de los métodos de sensorización visuales habituales, se tiene la gran cantidad de información a procesar. Si a esto se le añade la complejidad y el coste computacional de algunos algoritmos de visión artificial más (en muchos casos) la necesaria respuesta en tiempo real, se obtiene que es deseable una solución con *hardware* específico.

Por otra parte, es deseable una cierta flexibilidad *software* en los módulos de sensorización visual, combinada simultáneamente con una velocidad de proceso *hardware*. Los

dispositivos lógicos programables combinan simultáneamente la velocidad del *hardware* con una cierta programabilidad, con lo que se ha planteado la utilización de un sistema basado en lógica programable en la etapa de procesamiento de la información visual. El módulo de procesamiento de la información visual ha tomado el nombre de **módulo reconfigurable** debido a que es posible programar cada etapa del cauce segmentado por separado, análogamente a los dispositivos programables reconfigurables que son reprogramables parcialmente.

Se ha realizado una revisión de los dispositivos lógicos programables existentes al inicio de la investigación, encontrando unas características comunes deseables para el módulo reconfigurable. Por una parte es necesaria una tecnología de programación que permita la programabilidad y reconfigurabilidad en el sistema. Por otra parte son deseables características de las FPGAs clásicas (alta densidad) y de las CPLDs clásicas (altas prestaciones). Los dispositivos con arquitecturas híbridas han mostrado tener una combinación adecuada de ambas características.

Existen una gran variedad de máquinas reconfigurables, algunas de ellas orientadas al procesamiento de imágenes. Una revisión del estado de la investigación en estas máquinas no ha servido para encontrar una máquina que se ajustara a las necesidades de capacidad de procesamiento y de consumo de recursos de un vehículo autónomo. En el caso de máquinas reconfigurables orientadas al procesamiento de imágenes, la gran cantidad de información suministrada por las cámaras CCD cartesianas normales, unida a la complejidad de los algoritmos útiles para navegación de plataformas autónomas, dificulta la construcción de sistemas reconfigurables que puedan incorporarse a un sistema autónomo. De la necesidad de reducir la cantidad de información a procesar, seleccionando la información relevante y útil para la plataforma autónoma, surge la utilización de la visión log-polar.

La representación espacio-variante ha sido utilizada en estudios teóricos que han mostrado su utilidad en ciertos algoritmos de visión artificial, especialmente de visión activa. Sin embargo, la reciente disponibilidad de un sensor log-polar ha planteado su utilización para realizar la sensorización visual de un vehículo autónomo. La reducción selectiva de la información, conjuntamente con la simplificación de varios algoritmos de visión artificial útiles en robots, han determinado la utilización del sensor log-polar CMOS como entrada del módulo reconfigurable.

11.1.2 Diseño teórico del cauce reconfigurable y de los algoritmos

Un estudio de qué tipo de algoritmos sería interesante implementar en el módulo reconfigurable ha definido las características de la arquitectura del módulo. Los algoritmos diferenciales extraen de manera sencilla información relevante de secuencias de imágenes. De manera adicional suelen ser algoritmos en los cuales se aplica de forma sistemática una serie de operaciones sobre toda la imagen. Esta característica hace que sean algoritmos altamente paralelizables y que se beneficien rápidamente de la reducción en el tamaño de la imagen. En la misma línea, la segmentación del flujo de datos y la provisión de memorias que almacenen imágenes log-polares entre las etapas intermedias, permite el cálculo diferencial de manera eficiente. De esta forma se ha diseñado el módulo reconfigurable como un cauce segmentado síncrono, en el que cada etapa esta formada por un *Elemento de Proceso* (EP). Se ha diseñado un solo tipo de EP realizando varias copias del mismo. Todos los EPs tienen una interfase externa de conexión idéntica y disponen de 2 memorias de doble puerto (útiles para la diferenciación temporal), además de memoria local útil para la diferenciación espacial. La programación de los EPs se realiza a través de un bus común de programación. Cada EP tiene una dirección física distinta asignada con interruptores y circuitería adicional para facilitar la programación. En cualquier caso las dimensiones y consumo de los EPs son reducidas cumpliendo los requisitos de portabilidad de las plataformas autónomas.

A partir de la arquitectura del módulo reconfigurable se ha definido una metodología genérica para implementar algoritmos diferenciales de manera eficiente. El algoritmo se divide en etapas en función de la utilización de recursos. Cada etapa se implementa en un EP, configurando todos los EPs la plataforma autónoma para la ejecución completa del algoritmo. De hecho el diseño de algoritmos en el conjunto módulo reconfigurable/procesador de propósito general, puede ser planteado como un problema de codiseño binario. Por una parte, el módulo reconfigurable es útil para implementar las etapas que incluyen operaciones sistemáticas sobre las imágenes log-polares. Por la otra, el procesador de propósito general extrae información global útil para la toma de decisiones y la activación de los actuadores.

Siguiendo la metodología de programación en el módulo reconfigurable se han diseñado dos algoritmos diferenciales distintos útiles para la navegación de vehículos autónomos. El primero de ellos parte del trabajo originalmente desarrollado en coordenadas cartesianas por Chen y Nandhakumar y ha sido desarrollado por el doctorando para ser implementado en el cauce reconfigurable usando imágenes log-polares. En este primer algoritmo se detecta movimiento independiente del movimiento de la cámara. Para ello es necesario que tanto la imagen log-polar como el movimiento del robot sean

suaves. La primera condición se consigue cuando el movimiento de la plataforma es rectilíneo y uniforme con una velocidad de adquisición suficientemente rápida. La segunda condición se puede conseguir mediante un suavizado de la imagen log-polar. En este caso, si el centro del sensor coincide con el centro de expansión óptica debido al movimiento del robot, el desplazamiento de la imagen debido al movimiento del robot queda eliminado, detectándose solamente el borde de los objetos que se mueven respecto del fondo estático. Este algoritmo ha sido implementado con tres EPs: el primero realiza un suavizado, posteriormente el segundo EP calcula la primera derivada temporal y finalmente el último EP calcula la segunda derivada temporal y umbraliza el resultado. Simulaciones detalladas han mostrado que la implementación concreta realizada en el cauce reconfigurable de este algoritmo puede procesar hasta 285 imágenes por segundo.

El segundo algoritmo implementado ha sido la aplicación del método desarrollado por Tistarelli y Sandini para el cálculo del tiempo al impacto con coordenadas log-polares. De nuevo se supone que el centro del sensor coincide con el centro de expansión óptica debido al movimiento del robot. Se han utilizado igualmente en este caso tres EPs siendo el primer EP el mismo EP que realiza el suavizado en el algoritmo anterior; el siguiente EP calcula el gradiente espacial y la derivada temporal, siendo el último EP el encargado de realizar una división de estos valores. En este caso las simulaciones han mostrado un procesamiento de aproximadamente 80 imágenes por segundo.

11.1.3 Implementación física, experimentación y metodología de trabajo

Se ha diseñado y fabricado el circuito impreso del EP construyendo 3 EPs, que es el número de EPs que se usan en ambos algoritmos, y una tarjeta de programación del cauce reconfigurable. La señal de reloj global de los EPs vendrá limitada por la implementación física concreta. El hecho de utilizar FPGAs de grado 3 ha limitado el periodo de reloj global de las FPGAs a cerca de 33 MHz tras sintetizar el código VHDL de todas las etapas de los dos algoritmos implementados. De manera adicional se pretende automatizar del proceso de diseño de algoritmos en el módulo reconfigurable con lo que se ha evitado editar el emplazado y realizar el conexionado manualmente para mejorar el retraso en los caminos críticos. En cualquier caso el retraso más importante viene impuesto por las memorias cuyo tiempo acceso es de 25 ns para el caso de la memoria local, y de 35 ns para la memoria de doble puerto. De esta manera el reloj global del sistema se ha fijado a 60 ns. Posteriores implementaciones con FPGAs y memorias más rápidas mejorarán la frecuencia de la señal global de reloj.

Se han diseñado etapas en VHDL para comprobar por separado el funcionamiento

correcto de los recursos de cada EP. El trabajo de depuración de cada EP y del cauce en su conjunto ha sido costoso temporalmente pero el haber abordado de manera secuencial el funcionamiento de cada recurso a permitido aislar los errores físicos en el cauce. De hecho se ha elaborado una librería de etapas en VHDL útiles para la depuración de más EPs en el cauce reconfigurable.

Una vez se ha depurado la implementación física del cauce reconfigurable han aparecido más problemas, éstos no relacionados con el funcionamiento correcto de los EPs. La naturaleza diferencial de los algoritmos combinada con la rápida velocidad de adquisición/procesamiento y la estructura particular del sensor, plantea cuál debe ser el intervalo mínimo entre dos imágenes consecutivas para garantizar que las diferencias no sean nulas. De esta manera ha sido necesario garantizar en los dos algoritmos diseñados (ambos utilizan diferencias temporales) que se producen diferencias entre imágenes.

La experimentación se ha realizado en condiciones controladas para de esta manera determinar la bondad de ambos algoritmos. Se han tomado secuencias con muchas más imágenes de las que se han procesado para de esta manera simular diferentes velocidades de la plataforma móvil.

En el caso del algoritmo de detección de movimiento se ha llegado a establecer una relación para determinar de forma automática los parámetros del algoritmo: intervalo de adquisición y umbral. Diversos experimentos han mostrado la habilidad del algoritmo para descartar el desplazamiento de la imagen debido al movimiento del robot cuando la velocidad del objeto externo era del orden de magnitud de la velocidad del robot. De hecho, y tal como cabía esperar, la cantidad de puntos detectados es mayor cuando el movimiento del objeto externo es más transversal al movimiento de la plataforma móvil.

La implementación del algoritmo de Tistarelli y Sandini no ha mostrado unos resultados tan interesantes como los del algoritmo anterior. Se ha constatado la fuerte dependencia de la precisión del tiempo al impacto con el suavizado de la imagen. Los bordes y picos en las imágenes presentan derivadas mal definidas, con lo que no se puede calcular el tiempo al impacto. Experimentos con entornos no-preparados en el laboratorio no han suministrado valores correctos. Por contra, experimentos con imágenes sintéticas con gradientes constantes y sin bordes han mostrado un resultado muy preciso para τ (tiempo al impacto). Las opciones de aplicación del algoritmo en entornos reales no estructurados pasan por la realización de un mejor acondicionamiento de la imagen, que puede consistir en un suavizado más perfecto, lo cual implica o bien aumentar la complejidad y el número de ciclos de esta etapa si se mantiene en un sólo EP, o bien dividirla en varias etapas implementadas por más de 1 EP. Se ha comprobado como en los casos en los que la derivada no esté bien definida aparece un error que puede

generar una dispersión grande en los valores finales obtenidos. Es por tanto importante garantizar que las derivadas calculadas como simples diferencias tengan valores correctos.

En cualquier caso, y con la intención de probar la utilidad del algoritmo, se ha realizado un experimento en el cual se ha estructurado el entorno al preparar una pared con un cierto gradiente y sin bordes. Los resultados de la aplicación del algoritmo muestran ser adecuados si sólo se tienen en cuenta los puntos en los cuales el error relativo de la derivada temporal no es grande. Esto se consigue al haber rediseñado el último EP para que efectúe la división entera sólo si la derivada temporal es mayor que un cierto umbral. En estos casos se consiguen estimaciones del tiempo al impacto que no difieren demasiado del valor esperado. Estimaciones más precisas requerirán de un análisis estadístico más sofisticado de los mapas del tiempo al impacto que el simple cálculo de la media.

Toda la fase de experimentación ha permitido establecer una metodología experimental que acelerará la implementación futura de más algoritmos en el cauce reconfigurable. En primer lugar la fase de depuración de la implementación física de los EPs se ha automatizado mediante el diseño de etapas VHDL que acceden a los recursos de cada EP por separado. Por otra parte, la implementación de un algoritmo diferencial en el cauce reconfigurable se realizará siguiendo la metodología desarrollada en el presente trabajo de investigación. Una simulación previa de cada etapa del cauce que implementa el algoritmo no es suficiente para garantizar el funcionamiento final correcto de un determinado algoritmo. A la simulación comportamental y post-síntesis de las etapas del cauce en la herramienta de diseño de PLDs hay que añadir una simulación previa con imágenes log-polares sintéticas y reales. Posteriormente en el cauce reconfigurable se experimentará con imágenes sintéticas que serán útiles para acotar y parametrizar las fuentes de error en el resultado final de los algoritmos.

En definitiva y como conclusión global, se ha diseñado e implementado un módulo de procesamiento de imágenes log-polares útil para la navegación de vehículos autónomos. La combinación de visión log-polar y la adecuación de la arquitectura segmentada a los algoritmos diferenciales, consigue una alta tasa de imágenes procesadas por segundo. Los dos algoritmos diseñados prueban la flexibilidad de la arquitectura propuesta, así como la adecuación de la utilización de dispositivos lógicos reconfigurables en problemas de visión autónoma. El trabajo de investigación presentado ha realizado una serie de aportaciones que se resumen en la siguiente sección. De la misma manera, la investigación realizada ha planteado varias líneas de trabajo futuras en los campos de la arquitectura de computadores, la visión artificial y la robótica que se enumeran en la sección 11.3.

11.2 Aportaciones

La naturaleza multidisciplinar del presente trabajo de investigación aborda diversos aspectos del procesamiento de imágenes, de la arquitectura y tecnología de computadores y de la robótica. Las aportaciones realizadas se enumeran de forma resumida a continuación.

- Se ha realizado un arquitectura reconfigurable específica para la navegación de vehículos autónomos. Los resultados obtenidos, que combinan flexibilidad *software* y prestaciones *hardware*, apuntan a que éste es un buen campo de aplicación de las máquinas reconfigurables.
- Se ha diseñado una arquitectura orientada al procesamiento de algoritmos diferenciales de visión artificial basados en visión espacio-variante. La utilización de la visión log-polar, junto con la orientación de la arquitectura reconfigurable al cálculo diferencial, permiten una gran capacidad de proceso. De manera adicional, al formar los elementos de proceso un cauce segmentado se acelera más aun la capacidad de cálculo. La arquitectura definida es independiente de la implementación realizada. De hecho, al ser todos los elementos de proceso idénticos, la conectividad viene definida por el protocolo de intercambio de datos y por la interfase común. La posterior disponibilidad de componentes más rápidos permitirá aumentar las prestaciones de los elementos de proceso de forma significativa, pudiendo utilizar simultáneamente las etapas anteriormente diseñadas.
- Se ha definido una metodología de diseño de algoritmos diferenciales en el cauce reconfigurable. La sencillez de la arquitectura permite sistematizar el diseño de las etapas que forman un algoritmo completo. De hecho, se ha ejemplificado la utilidad de la metodología con el diseño de dos algoritmos diferenciales distintos.
- Se ha diseñado, para su utilización en la arquitectura reconfigurable, un algoritmo de detección de movimiento independiente del movimiento de la cámara. La aplicación del trabajo de Chen y Nandhakumar a las coordenadas log-polares, ha mostrado ser útil para descartar la variación de las imágenes debida al movimiento de la cámara cuando el centro del sensor coincide con el centro de expansión óptica. Se han diseñado en VHDL, utilizando la metodología genérica de diseño en el cauce reconfigurable, las etapas que implementan el algoritmo de detección de movimiento. La posterior experimentación con este algoritmo ha permitido automatizar la elección de los parámetros más importantes del algoritmo.
- Se ha implementado en el cauce reconfigurable el algoritmo de cálculo del tiempo al impacto en coordenadas log-polares desarrollado por Tistarelli y Sandini. La posterior experimentación con la implementación de este algoritmo ha mostrado la

poca precisión de los valores numéricos del tiempo al impacto si no se realiza un fuerte suavizado de las imágenes. A pesar de estos resultados, la implementación del algoritmo en el cauce reconfigurable ha mostrado ser útil en entornos estructurados especialmente (paredes con gradiente).

- Se han deducido ecuaciones que parametrizan el intervalo temporal necesario para producir variaciones en los *pixels* del sensor en función de la geometría del sensor y los parámetros de la escena. La naturaleza diferencial de los algoritmos implementados en el cauce reconfigurable obliga a garantizar que estas diferencias se produzcan y estén bien definidas. La alta velocidad de proceso del cauce reconfigurable (en términos de imágenes por segundo) ha tenido que ser limitada con estas ecuaciones que definen el intervalo mínimo temporal que debe transcurrir entre dos imágenes consecutivas.
- Se ha implementado el cauce reconfigurable con la intención de validar de forma experimental el diseño realizado. Se ha diseñado un solo circuito impreso con el que se han construido los 3 elementos de proceso que implementan las 3 etapas de los dos algoritmos que se han diseñado posteriormente.

11.3 Trabajo futuro

La presente investigación plantea numerosas líneas de trabajo futuras en diversos campos. Algunas de estas líneas serán abordadas directamente por el doctorando y por el grupo de trabajo al cual pertenece. Otras continuaciones serán realizadas conjuntamente con otros grupos de investigación, tanto de la Universitat de València como externos, con los que se ha colaborado en los proyectos que han financiado la presente investigación.

La continuación más inmediata consiste en la integración del módulo reconfigurable como etapa de sensorización en el robot móvil recientemente desarrollado en el **Institut de Robòtica** de la Universitat de València. La integración de los datos suministrados por el módulo reconfigurable en un robot autónomo, y la fusión de datos con datos procedentes de otros métodos de sensorización plantean la extensión más cercana.

El desarrollo e implementación en el cauce reconfigurable de otros algoritmos útiles para la navegación de plataformas móviles autónomas es también una línea de trabajo a tomar de manera inmediata. Posteriormente, y con una librería más extensa de algoritmos para el cauce reconfigurable, se evaluará realmente la eficiencia de la reconfigurabilidad de la arquitectura.

La arquitectura reconfigurable ha sido especialmente desarrollada para su integra-

ción en una plataforma autónoma, pero puede ser utilizada en otros ámbitos de aplicación. Dentro de la colaboración con el Departament d'Informàtica de la **Universitat Jaume I** de Castellón se planteará el desarrollo de algoritmos de visión log-polar útiles para vigilancia, siendo el paso siguiente su implementación en el cauce reconfigurable.

Otro uso que se va a realizar del módulo reconfigurable es su utilización para el procesado de imágenes log-polares obtenidas desde vehículos en carretera. La posibilidad de la arquitectura desarrollada de procesar cerca del centenar de imágenes log-polares por segundo será totalmente utilizada en este caso. El diseño y aplicación de algoritmos diferenciales basados en visión log-polar, útiles para el control de vehículos, se va a desarrollar bajo el proyecto recientemente concedido de la Generalitat Valenciana GV99-116-1-14. En este proyecto se colaborará con el **INTRAS** (Instituto de Tráfico y Seguridad Vial de la Universitat de València).

Por otra parte, la interacción del módulo reconfigurable con el procesador de propósito general de la plataforma autónoma, ha planteado la similitud de este problema con el problema del codiseño binario. La partición *hardware* correspondería al módulo reconfigurable y la partición *software* al procesador que adicionalmente ejecuta el resto de tareas de la plataforma autónoma. Cuanto mayor sea el porcentaje del algoritmo total que se encargue de realizar el módulo reconfigurable más liberada estará la CPU del robot, por tanto será posible una atención mayor a otros subsistemas y se tendrá un funcionamiento más óptimo. Por el contrario, el coste y consumo de potencia será mayor al utilizar una mayor cantidad de elementos de proceso y componentes. Existirán casos en los que será absolutamente necesario utilizar la total potencialidad del cauce reconfigurable para computar algoritmos diferenciales de visión log-polar por cuestiones de velocidad y porque la CPU no pueda abordarlos por si misma. En los casos en los que no sea así habrá que llegar a un compromiso entre el tiempo de CPU que se puede dedicar a procesar las imágenes log-polares y el coste/consumo que se puede asumir para la plataforma móvil.

De manera adicional, se planteará el diseño de un conjunto de herramientas que automaticen la generación del código de programación de las FPGAs de los elementos de proceso, y el código del procesador de propósito general. Se planteará la descripción mediante VHDL de algoritmos diferenciales y la generación del código de ambas particiones en función de las restricciones de tiempo real y de las limitaciones de la plataforma móvil.

En paralelo, y como un ejercicio de mejora tecnológica, se ha planteado el desarrollo de elementos de proceso con una mayor capacidad de cálculo. La utilización de FPGAs más rápidas, con una mayor cantidad de puertas lógicas equivalentes, o la integración de varias FPGAs en un sólo elemento de proceso plantea mejoras que serán función de

las necesidades de las etapas de los algoritmos y de la evolución tecnológica.

Parte V

Bibliografía

Bibliografía

[AA95] P. Athanas y L. Abbott. Real-time image processing on a custom computing platform. *IEEE Computer*, 28(2):16–24, 1995.

[AAV97] Josep Amat, Joan Aranda, y Ricard Villà. Underwater hidrojet explorer camera controlled by vision. En *Proceedings of the Fifth International Symposium on Experimental Robotics*, páginas 487–497, Junio 1997.

[AD98] APS-Documentation. Aps-x208 data sheet, 1998.

[APE99] *APEX 20K Programmable Logic Device Family Data Sheet*, preliminary information edición, Febrero 1999.

[BAK96] Duncan Buell, Jeffrey Arnold, y Walter Kleinfelder. *Splash2: FPGAs for Custom Computing Machines*. IEEE Computer Society Press, 1996.

[BB95] S.S. Beauchemin y J.L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–467, 1995.

[BCE+97] Felice Balarin, Massimiliano Chiodo, Daniel Engels, Paolo Giusto, Wilsin Gosti, y Harry Hsieh. *POLIS A design environment for control-dominated embedded systems*, 1997.

[BCM+96] J. Bausell, J. Carrabina, A. Merlos, J. Sáiz, S. Bota, y J. Samitier. Mechanical sensors integrated in comercial CMOS technology. En *Proceedings of the XI Conference of Design of Integrated Circuits and Systems*, páginas 107–110, Noviembre 1996.

[BDPP97a] Jose A. Boluda, Juan Domingo, Fernando Pardo, y Joan Pelechano. Detecting motion independent of the camera movement through a log-polar differential approach. En *Computer Analysis of Images and Patterns. 7th International Conference CAIP'97*, volumen 1296, páginas 702–710. Lectures Notes on Computer Science. Springer-Verlag, Septiembre 1997.

[BDPP97b] Jose A. Boluda, Juan Domingo, Fernando Pardo, y Joan Pelechano. Motion detection independent of the camera movement with a log-polar sensor. En *13th International Conference on Digital Signal Processing, DSP'97*, páginas 809–812, Santorini, Grecia, Julio 1997.

[Ber97] Gerard Berry. *The Esterel V5 Languaje Primer*, 1997. Ecole des Mines and INRIA.

[BFB93] J.L. Barron, D.J. Fleet, y S.S. Beauchemin. Performance of optical flow techniques. Informe Técnico RPL-TR-9107, Dpto. of Computing and Information Science, Queen's University, Kingston, Ontario, Julio 1993.

[BFBB92] J.L. Barron, D.J. Fleet, S.S. Beauchemin, y T.A. Burkitt. Performance of optical flow techniques. En *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, páginas 236–242, Junio 1992.

[BKV96] Stephen Brown, Muhammad Khellah, y Zvonko Vranesic. Minimizing FPGA interconnect delays. *IEEE Design and Test of Computers*, páginas 16–23, Winter 1996.

[Bla98] Francisco Blasco. Entorno de adquisición de imágenes log-polares a alta velocidad. Tesis de Maestría, Ingenería Informàtica. Universitat de València, 1998. Proyecto fin de Carrera.

[Boe95] Eduardo I. Boemo. *Contribución al Diseño de Arrays VLSI con paralelismo de grano fino.* Tesis Doctoral, Dpto. de Ingeniería Electrónica. ETSI-UPM, Madrid, Noviembre 1995.

[Bor96] Johann Borenstein. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880, 1996.

[Box94] B. Box. Field programmable gate array based reconfigurable preprocessor. En D. A. Buell y K. L. Pocek, editores, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machings*, páginas 40–48, Napa, CAEEUU, Abril 1994.

[BPBP98] Jose A. Boluda, Fernando Pardo, Francisco Blasco, y Joan Pelechano. Una arquitectura segmentada para el cálculo del tiempo al impacto con visión log-polar. En *XIX Jornadas de Automática*, páginas 281–285, Septiembre 1998.

[BPK+96] Jose A. Boluda, Fernando Pardo, Tomas Kayser, Juan J. Pérez, y Joan Pelechano. A new foveated space-variant camera for robotic applications. En *IEEE, International Conference on Electronics Circuits And Systems, ICECS'96*, páginas 680–683, Rodos, Grecia, Octubre 1996.

[BPP98] Jose A. Boluda, Fernando Pardo, y Joan Pelechano. Reconfigurable architectures for machine perception. an approach for autonomous vehicle navigation. En *Workshop on European Scientific and Industrial Collaboration on promoting advanced technologies in manufacturing. WESIC'98*, páginas 359–363, Girona, España, Junio 1998.

[BR96] Stephen Brown y Jonathan Rose. FPGA and CPLD architectures: A tutorial. *IEEE Design and Test of Computers*, páginas 42–56, Summer 1996.

[Bro96] Stephen Brown. FPGA architectural research: A survey. *IEEE Design and Test of Computers*, páginas 9–15, Winter 1996.

[BSV96a] A. Bernardino y J. Santos-Victor. Correlation based vergence control using log-polar images. En *Int. Symposium on Intelligent Robotics Systems, SIRS'96*, Lisboa, Portugal, Julio 1996.

[BSV96b] A. Bernardino y J. Santos-Victor. Vergence control for robotic heads using log-polar images. En *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'96*, Osaka, Japón, Noviembre 1996.

[Bur96a] Dave Bursky. Programmable arrays mix FPGA and ASIC blocks. *Electronic Design*, 44(21):69–74, 1996.

[BUR96b] J. Byrne, P. Undrill, y P. Ross. Communication dependent image analysis using a network of grid connected processors. *Transputer Communications*, 3(2):101–115, Abril 1996.

[Can96] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8(6), 1996.

[CB92] C. E. Cox y W. E. Blanz. GANGLION - a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, Marzo 1992.

[CK97] Fabio Cozman y Eric Krotkov. Automatic detection and pose estimation for teleoperation of lunar rovers. En *Proceedings of the Fifth International Symposium on Experimental Robotics*, páginas 164–172, Junio 1997.

[CMRU93] R. Creutzburg, T. Minkiwitz, M. Roggenbach, y T. Umland. Parallel fft-like trasnsform algorithms on transputers. En *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, páginas 53–89, 1993.

[CN93] W. Chen y N.Nandhakumar. A simple scheme for motion boundary detection. En *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1993.

[Col95] Philip Colet. NSP vs. DSP: The battle for image processing supremacy. *DSP and Multimedia Technology*, páginas 21–26, Julio 1995.

[CPP+94] J. Champeau, L. Le Pape, B. Pottier, S. Rubini, E. Gautrin, y L. Perraudeau. Flexible parallel fpga-based architectures with armen. En *Annual Hawaii International Conference on System Sciences (ACM-IEEE)*, Hawaii, EEUU, Enero 1994.

[CPS97] C. Capurro, F. Panerai, y G. Sandini. Dynamic vergence using log-polar images. *International Journal of Computer Vision*, 24(1):79–94, 1997.

[CSM92] P.K. Chan, M. Schlag, y M. Martin. Borg: A reconfigurable prototyping board using fpgas. En *1992 ACM International Workshop on Field- Programmable Gate Arrays*, páginas 47–51, Febrero 1992.

[Día96] Elena Díaz. *Estimación de movimiento en secuencias de imágenes mediante la detección y encaje de puntos relevantes. Aplicación al seguimiento de vehículos para el análisis de trayectorias.* Tesis Doctoral, Departament de Informàtica. Universitat de València, 1996.

[DAD97] Juan Domingo, Guillermo Ayala, y Elena Diaz. A method for multiple rigid-object motion segmentation based on detection and consistent matching of relevant points in image sequences. En *Proceedings of the IEEE international Conference on Acoustics, Speech and Signal Processing*, volumen 4, páginas 3021–3025, Abril 1997.

[Dan95a] K. Daniilidis. Computation of 3D-motion parameters using the log-polar transform. En *Int. Conf. on Computer analysis of images and patterns, CAIP'95*, páginas 82–89, 1995.

[Dan95b] K. Daniilidis. Optical flow computation in the log-polar plane. En *Int. Conf. on Computer analysis of images and patterns, CAIP'95*, páginas 65–72, 1995.

[Dan96] K. Daniilidis. Attentive visual motion processing computation in the log-polar plane. *Computing*, (11):1–20, 1996.

[DSM+96] Bart Dierickx, Danny Scheffer, Guy Meynants, Werner Ogiers, y Jan Vlummens. Random addressable active pixel image sensors. En *SPIE European*

Symposium on Advanced Imaging and Network Technologies, AFPAEC'96, Berlin, Alemania, Octubre 1996.

[FCS98] Josep Fernández, Alícia Casals, y Jordi Saludes. Adaptative approach for autonomous visual navigation. En *Proceedings of the 1998 IEEE International Conference of Intelligent Vehicles*, volumen 1, páginas 219–224, Alemania, Octubre 1998.

[FDP96] Bradley Fross, Robert Donaldson, y Douglas Palmer. PCI-based wildfire reconfigurable computing engines. En *High-Speed Computing, Digital Signal Processing, and Filtering using Reconfigurable Logic*, volumen 2914 de *Proceedings of the SPIE*, páginas 170–179, Noviembre 1996.

[FGP+97] F. Ferrero, M. González, M. Pérez, F. Fernández, y J. González. Lógica programable: Arquitecturas de fpgas de altas prestaciones. *Mundo Electrónico*, (274):28–33, Febrero 1997.

[FGS+92] M. Fossa, E. Grosso, G. Sandini, M. Zapendouski, F. Ferrari, y M. Magrassi. A visually guided mobiles robot acting in indoor environments. En *Proceedings of the IEEE workshop on applications of computer vision*, páginas 308–316, Noviembre 1992.

[FJPG95] F.Ferrari, J.Nielsen, P.Questa, y G.Sandini. Space variant imaging. *Sensor Review*, 15(2):17–20, 1995.

[GHHS96] Jeffrey Gealow, Frederick Hermann, Lawrence Hsu, y Charles Sodini. System design for pixel-parallel image processing. *IEEE Transactions on Very large Scale of Integration (VLSI) Systems*, 4(1):32–41, 1996.

[GS95] Michael Gachwind y Valentina Salapura. A VHDL design methodology for FPGAs. En *Proceesings of the 5th International Workshop on Field Programmable Logic and Applications*, Oxford, Inglaterra, 1995.

[GT94] Francisco García y Luis Tomás. Aplicación de ultrasonidos para determinar entornos no estructurados. *Mundo Electrónico*, (248):30–33, 1994.

[Guc99] Steven Guccione. List of FPGA-based computing machines. Web page at http:www.io.com, 1999.

[HFB97] Dominique Houzet, Abdelkrim Fatniand, y Jean-Luc Basille. Image processing PCI-based shared memory architecture design. En *Proceedings of Computer Architectures for Machine Perception*, volumen 1997, páginas 244–252, Octubre 1997.

[Hig99] Tetsuya Higuchi. Evolvable hardware chips for industrial applications. *Communications of the ACM*, 42(4):60–69, 1999.

[Hir93] Ernest Hirsch. *Parallel Algorithms and Computer Vision*, páginas 353–390. Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks. WILEY, 1993.

[HKL95a] J. G. Harris, C. Koch, y J. Luo. *A Two-Dimensional Analog VLSI Circuit for Detecting Discontinuites in Early Vision*, páginas 221–223. VISION CHIPS. Implementing Vision Algorithms with Analog VLSI Circuits. IEEE Computer Sociey Press, 1995.

[HKL+95b] H. Högl, A. Kugel, J. Ludvig, R. Männer, K.Ñoffz, y R. Zoz. Enable++: A second generation fpga processor. En *3rd IEEE Symp. on FPGAs for Custom Computing Machines*, Napa, CA, EEUU, 1995.

[HP90] John L. Hennessy y Davis A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kauffmann Publishers, Inc., 1990.

[Hyn97] Jaroslav Hynecek. Low-noise and high-speed charge detection in high resolution CCD image sensors. *IEEE Transactions on Electron Devices*, 44(10):1679–1688, 1997.

[HZM93] M. Holden, M. J. Zemerly, y J.-P. Muller. Parallel stereo and motion estimation. En *Parallel algorithms for digital image processing, computer vision and neural networks*, Series in Parallel Computing, páginas 175–232. Wiley Professional Computing, 1993.

[IEE93] IEEE. *IEEE Standard VHDL Languaje Reference Manual, Standard 1076-1993.* IEEE Standards Press, 1993.

[IJ95] Tarek B. Ismail y Ahmed A. Jerraya. Synthesis steps and design models for codesigns. *IEEE Computer*, 28(2):44–53, Febrero 1995.

[IMH+86] R.M. Iñigo, E.S. McVey, C. Hsin, Z. Rahman, y J. Minnix. A new sensor for machine vision. En *Proceedings of the IFAC Symposium LCA'86*, páginas 83–88, Noviembre 1986.

[IMHM86] R.M. Iñigo, J.L. Minnix, C. Hsin, y E.S. McVey. A biological-structure visual sensor for robotic applications. En *Proceedings of the 16th International Symposium on Industrial Robots*, Octubre 1986.

[IMRS94] C. Innocenti, G. Mondino, P. Regis, y G. Sandini. Trajectory planning and real-time control of an autonomous mobile robot equipped with vision and ultrasonic sensors. En *Proceedings of IROS'94*, Septiembre 1994.

[IXAM92a] R.M. Iñigo, C.Q. Xu, B.C. Arrúe, y E.S. McVey. Hardware-implementable neural network for rotation-scaling invariant pattern classification. *SPIE Journal of Electronic Imaging*, 1(3):293–312, Julio 1992.

[IXAM92b] R.M. Iñigo, C.Q. Xu, B.C. Arrúe, y E.S. McVey. Hardware-implementable neural network for rotation-scaling invariant pattern classificati on. *SPIE Journal of Electronic Imaging*, 1(3):293–312, Julio 1992.

[Jar97] Ray Jarvis. Etherbot - an autonomous mobile robot on a local area network radio tether. En *Proceedings of the Fifth International Symposium on Experimental Robotics*, páginas 151–163, Junio 1997.

[JMM96] A. Jain, J. Mao, y K. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, 1996.

[Jur99] Frederic Jurie. A new log-polar mapping for space variant imaging. application to face detection and tracking. *Pattern recognition*, 32(5):865–875, Mayy 1999.

[JV97] Pieter Jonker y Jan Vogelbruch. The CC/IPP, an MIMD-SIMD architecture for image processing and pattern recognition. En *Proceedings of Computer Architectures for Machine Perception*, volumen 1997, páginas 33–39, Octubre 1997.

[KL92] A. Kalavade y E. A. Lee. Hardware/software co-design using ptolemy - a case study. En *Proc. of the IFIP International Workshop on Hardware/Software Co-Design*, 1992.

[KMB+95] C. Koch, A. Moore, W. Bair, T. Horiuchi, B. Bishofberger, y J. Lazzaro. *Computing Motion Using Analog VLSI Vision Chips: An Experimental Comparison Among Four Approaches*, páginas 301–313. VISION CHIPS. Implementing Vision Algorithms with Analog VLSI Circuits. IEEE Computer Sociey Press, 1995.

[Knu95] Peter Voigt Knudsen. *Fine-grain partitioning in codesing.* Tesis Doctoral, Technical University of Denmark, 1995.

[KVB+90] G. Kreider, J. Van der Spiegel, I. Born, C. Claeys, I. Debusschere, G. Sandini, P. Dario, y F. Fantini. A retina like space variant CCD sensor.

SPIE, Charge-coupled Devices and Solid State Optical Sensors, 1242:133–140, 1990.

[LFE+99] G. Liñan, P. Foldesy, S. Espejo, R. Dominguez-Castro, E. Roca, y A. Rodríguez-Vázquez. Real-time image processing using a vlsi mixed signal analog array processor. En *Proceedings of XIV Conferece on Design of Circuits and Integrated Systems*, páginas 229–234, November 1999.

[Mic94] Giovanni De Micheli. Computer-aided hardware-software codesign. *IEEE Micro*, 14(4):10–16, 1994.

[MOY97] Shoichi Maeyama, Akihisa Ohya, y Shin'ichi Yuta. Long distance outdoor navigation of an autonomous mobile robot by playback of perceived route maa. En *Proceedings of the Fifth International Symposium on Experimental Robotics*, páginas 141–150, Junio 1997.

[MP98] Mike Mills y Greg Peterson. Hardware/software codesign: VHDL and ADA95 code migration and integrated analysis. *ADA Letters*, 18(4):18–26, December 1998.

[MWD91] M. Mirmehdi, G. West, y G Dowling. Label inspection using the hough transform on transputer networks. *Microprocessors and Mycrosystems*, 15(3):167–173, 1991.

[OF97] Stuart F. Oberman y Michael J. Flynn. Division algorithms and implementations. *IEEE Transactions on Computers*, 46(8):833–854, 1997.

[PA96] Kevin Paar y Peter Athanas. Implementation of a finite difference method on a custom computing platform. En *High-Speed Computing, Digital Signal Processing, and Filtering using Reconfigurable Logic*, volumen 2914 de *Proceedings of the SPIE*, páginas 44–53, Noviembre 1996.

[Pag94] Ian Page. The harp reconfigurable computing system. Informe técnico, Oxford Hardware Compilation Research Group, 1994.

[Par97] Fernando Pardo. *Sensor retínico espacio variante basado en tecnología CMOS.* Tesis Doctoral, Departament d'informàtica, Facultat de Física, Universitat de València, Marzo 1997.

[PBMP94] Fernando Pardo, Jose A. Boluda, Rafael J. Martínez, y Carlos Pérez. Lógica programable, criterios de selección. *Mundo Electrónico*, páginas 27–31, Junio 1994.

[PBP+96a] Fernando Pardo, Jose A. Boluda, Juan J. Pérez, Bart Dierickx, y Danny Scheffer. Design issues on CMOS space-variant image sensors. En *SPIE Proceedings: Advanced Focal Plane Arrays and Electronic Cameras, AFPAEC'96*, volumen 2950, páginas 98–107, Berlin, Alemania, Octubre 1996.

[PBP+96b] Fernando Pardo, Jose A. Boluda, Juan J. Pérez, Santiago Felici, Bart Dierickx, y Danny Scheffer. Response properties of a foveated space-variant CMOS image sensor. En *IEEE, International Symposium on Circuits And Systems, ISCAS'96*, volumen 1, páginas 373–376, Atlanta, EEUU, Mayo 1996.

[PDS97] F. Pardo, B. Dierickx, y D. Scheffer. CMOS foveated image sensor: Signal scaling and small geometry effects. *IEEE Transactions on Electron Devices*, 44(10):1731–1737, Octubre 1997.

[PDS98] F. Pardo, B. Dierickx, y D. Scheffer. Space-variant non-orthogonal structure CMOS image sensor design. *IEEE Journal of Solid State Circuits*, 33(6), Junio 1998.

[PH94] Simon Perkins y Gillian Hayes. Real-time optical flow based range sensing in mobile robots. En *Proceedings of the International Symposium on Intelligent Robotic Systems*, páginas 303–310, Julio 1994.

[Pit93] Ioannis Pitas, editor. *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*. Series in Parallel Computing. WILEY, 1993.

[PP93] N. Pal y S. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[PTRC96] S. Pillement, L. Torres, M. Robert, y G. Cambon. Lirmm : prototyping platform for hardware/software codesign. Informe Técnico 96061, Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier, 1996.

[QKSZ94] G.M. Quénot, I.C. Kraljic, J. Sérot, y B. Zavidovique. A reconfigurable compute engine for real-time vision automata prototyping. En *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, EEUU, Abril 1994.

[RD92] Nico Ricquier y Bart Dierickx. Pixel structure with logarithmic response for intelligent and flexible imager architectures. *Microelectronic Engineering*, 19:631–634, 1992.

[RD93a] T. Reed y J. Dubuf. A review of recent texture segmentation and feature extraction techniques. *CVGIP - Image Understanding*, 57(3):359–372, 1993.

[RD93b] Nico Ricquier y Bart Dierickx. Addressable imager with a logarithmic response for machine vision. En *ISIIROM*. Brussels, 1993.

[RP98] Thomas Röwekamp y Liliane Peters. A compact sensor for visual motion detection. *Videre: Journal of Computer Vision Research*, 1(2):36–54, 1998.

[Sca97] Frank A. Scarpino. *VHDL and AHDL Digital System implementation.* Prentice Hall, 1997.

[Sch87] Dieter K. Schroder. *Advanced MOS Devices.* Modular Series on Solid State Devices. Addison-Wesley Publishing Company, 1987.

[Sch97] Herman Schmit. Incremental reconfiguration for pipelined applications. En *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, páginas 47–55, 1997.

[Sha95] L. H. Shapiro. *Affine Analysis of image Sequences.* Cambridge University Press, 1995.

[Ska96] Kevin Skahill. *VHDL for Programmable Logic.* Addison Wesley, 1996.

[Sme95] Yu. Smetanin. Neural networks as systems for pattern recognition: a review. *Pattern Recognition and Image Analysis*, 5(2):254–293, 1995.

[SMS99] Moshe Sipper, Daniel Mange, y Eduardo Sanchez. Quo vadis evolvable hardware? *Communications of the ACM*, 42(4):50–56, 1999.

[SNO+97] N. Sakaguchi, N.Nakumara, S. Oshava, Y. Endo, Y. Matsunaga, K. Ooi, y O. Yoshida. Dark current fixed pattern noise reduction for the 2/3-in two million pixel HDTV STACK-CCD imager. *IEEE Transactions on Electron Devices*, 44(10):1658–1666, 1997.

[Tay96] Brad Taylor. DSP filters in FPGAs for image processing applications. En *High-Speed Computing, Digital Signal Processing, and Filtering using Reconfigurable Logic*, volumen 2914 de *Proceedings of the SPIE*, páginas 100–109, Noviembre 1996.

[TS91a] M. Tistarelli y G. Sandini. Direct estimation of time-to-impact from optical flow. En *IEEE workshop on visual motion*, Princeton, New Jersey EEUU, Octubre 1991.

[TS91b] M. Tistarelli y G. Sandini. On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow. *IEEE Trans. on PAMI*, PAMI-15, No. 4:401–410, 1991.

[TS92] M. Tistarelli y G. Sandini. Dynamic aspects in active vision. *CVGIP: Image Understanding*, 56 No.1:108–129, 1992.

[Veg99] Francisco Vegara. *Comparación test y diseño de arquitecturas de control para la generación de comportamientos eficientes por robots móviles en entornos no estructurados.* Tesis Doctoral, Facultat de Física. Universitat de València, Jun 1999.

[VKC+89] J. Van der Spiegel, G. Kreider, C. Claeys, I. Debusschere, G. Sandini, P. Dario, F. Fantini, P. Belluti, y G. Soncini. *A Foveated Retina-Like Sensor using CCD Technology.* Kluwer Academic, Boston, 1989.

[WC79] C. Weiman y G. Chaikin. Logarithmic spiral grids for image processing and display. *Computer Graphics and Image Processing*, 11:197–226, 1979.

[WRL95] Robert Wodnicki, Gordon Roberts, y Martin Levine. A foveated image sensor in standard CMOS technology. En *Custom Integrated Circuits Conference*, Santa Clara, CaliforniaEEUU, Mayo 1995.

[XIM91] C.Q. Xu, R.M. Iñigo, y E.S. McVey. A combined approach for large-scaling pattern recognition with traslational, rotational and scal ing invariances. *SPIE, Automatic Object Recognition*, 1471:378–389, 1991.

[Yao99] Xin Yao. Following the path of evolvable hardware. *Communications of the ACM*, 42(4):47–49, 1999.

Publicaciones relacionadas con el presente trabajo

[PVBF95] F. Pardo, F. Vegara, J.A. Boluda, and S. Felici. Sensores CMOS para robótica e industria: Sensor retínico espacio variante y visión activa. En *Cuarto Congreso AER-ATP*, pages 101-106, Zaragoza, Octubre 1995.

[BPP+96] J.A. Boluda, F. Pardo, J.J. Pérez, J. Domingo y J. Pelechano. Seguimiento de objetos mediante una cámara log-polar con movimiento propio. En *XVII Jornadas de Automática*, pages 179-184, Santander, Septiembre 1996.

[BPK+96] J.A. Boluda, F. Pardo, T. Kayser, J.J. Pérez, and J. Pelechano. A new foveated space-variant camera for robotic applications. En *IEEE, International Conference on Electronics Circuits And Systems, ICECS'96*, páginas 680-683, Rodos, Grecia, Octubre 1996.

[BDPP97a] J.A. Boluda, J.J. Domingo, F. Pardo and J. Pelechano. Motion Detection Independent of the Camera Movement with a Log-Polar Sensor. En *13th International Conference on Digital Signal Processing, DSP'97*, pages 809-812, Santorini, Grecia, Julio 1997.

[BDPP97b] J.A. Boluda, J.J. Domingo, F. Pardo and J. Pelechano. Detecting motion independent of the camera movement through a log-polar differential approach. En *7th International Conference on Computer Analysis of Images and Patterns CAIP'97*, Proceedings publicados en *Springer lecture notes in computer sciences series*, volumen 1296, página 702-710, Kiel, Alemania, Septiembre 1997.

[PB97] F. Pardo y J.A. Boluda. Cámaras CMOS y visión foveal. En *Seminario Anual de Automática, Electrónica Industrial e Instrumentación, SAAEI'97*, páginas 316-321, Valencia, Septiembre 1997.

[BPP98] J.A. Boluda, F. Pardo and J. Pelechano. Reconfigurable architectures for machine perception. An approach for autonomous vehicle navigation. In *Workshop on European Scientific and Industrial Collaboration on pro-*

moting advanced technologies in manufacturing, WESIC'98, páginas 359-363, Girona, Junio 1998.

[BPBP98] J.A. Boluda, F. Pardo, F. Blasco and J. Pelechano. Una arquitectura segmentada para el cálculo del tiempo al impacto con visión log-polar. En *XIX Jornadas de Automática*, páginas 281-285, Madrid, Septiembre 1998.

[BPB98] F. Blasco, F. Pardo and J.A. Boluda. Sistema de adquisición de imágenes log-polares con alta velocidad basado en bus PCI. En *XIX Jornadas de Automática*, páginas 277-280, Madrid, Septiembre 1998.

[BPBP99] J.A. Boluda, F. Pardo, F. Blasco and J. Pelechano A pipelined reconfigurable architecture for visual-based navigation. En *EUROMICRO'99*, páginas 71–74, Milan, Italia, Septiembre 1999.

[BBP99] J.A. Boluda, F. Blasco and F. Pardo A scalable reconfigurable FPGA-based architecture for robotic navigation. En *XIV Design of Circuits and Integrated Systems Conference. DCIS'99*, páginas 831-836, Palma de Mallorcas, Noviembre 1999.

[BPB99] F. Blasco, F. Pardo and J.A. Boluda A FPGA-based PCI bus interface for a real-time log-polar image processing system. En *XIV Design of Circuits and Integrated Systems Conference. DCIS'99*, páginas 379–384, Palma de Mallorcas, Noviembre 1999.

Parte VI

Apéndices

Apéndice A

Lógica programable

A.1 Introducción: Tecnologías de programación

Hasta la aparición de los primeros dispositivos lógicos programables las únicas opciones para diseñar hardware digital eran, o bien el diseño de circuitos integrados hechos a medida, o bien la utilización de componentes estándar LSI-MSI conectados entre si. La solución del circuito integrado hecho a medida es la preferible en cuanto a optimización de silicio, velocidad, tamaño y consumo de potencia. Los problemas de esta opción son la falta de flexibilidad y el alto coste de la fabricación de los circuitos *fullcustom* (a medida). La opción de la utilización de circuitos digitales estándar tiene el problema de la falta de flexibilidad para un diseño hardware y el excesivo silicio no utilizado que ocupa espacio y consume potencia.

Los dispositivos lógicos programables, o PLDs, son programables en el sentido de que en el circuito se tienen difundidos de forma fija los dispositivos lógicos, puertas lógicas y flip-flops. Lo que se *programa* son las interconexiones entre estos dispositivos lógicos. De esta manera se puede tener la velocidad del hardware sin haber hecho un costoso circuito a medida. Basta con programar las conexiones internas de manera adecuada para que la PLD realice la función deseada.

Hay diversas formas de programar una PLD. Las diversas tecnologías de programación están íntimamente relacionadas con la arquitectura del dispositivo lógico programable. En la tabla A.1 se resumen las principales características, ventajas y desventajas de cada tecnología. Cabe hacer notar que una misma característica, por ejemplo la volatilidad, puede ser una ventaja o desventaja dependiendo de la aplicación. La característica ISP es la programabilidad en el sistema o *In System Programability*, característica fundamental si se desea que el sistema sea programable de forma automática. De manera adicional, se suele diferenciar entre lógica programable y lógica reconfigurable ya que

Tecnología	Ventajas	Desventajas
Antifusible	No volatilidad, Alta integración	No reprogramabilidad
EEPROM	No volatilidad	No ISP
EPROM	No volatilidad	No ISP
SRAM	Reprogramabilidad, ISP	Volatilidad,
FLASH	Reprogramabilidad, No volatilidad, ISP	Baja integración

Tabla A.1: *Tecnologías de programación ventajas y desventajas*

esta última se entiende que engloba a los dispositivos lógicos que permiten una reconfiguración parcial. En el caso del módulo de procesado de imágenes diseñado se ha optado mayoritariamente por el adjetivo reconfigurable ya que las etapas del cauce son programables por separado. En el caso del módulo reconfigurable, la tecnología debe ser la SRAM para permitir simultáneamente la programabilidad en el sistema y una razonable escala de integración.

Se debe realizar un análisis de las diferentes clases de dispositivos para tener un criterio de selección del más adecuado [PBMP94].

A.2 PALs, CPLDs y FPGAs clásicas

Los dispositivos programables más simples son las matrices lógicas programables, conocidas como PALs (Programmable And Logic). La arquitectura de una PAL genérica se muestra en la figura A.1.

Las salidas no son más que una función OR de varias líneas AND cableadas. Las lineas horizontales cruzan completamente la PAL posibilitando una conexión en cada cruce con una línea vertical. Las salidas a su vez se realimentan para hacer posible funciones más complejas que el número de minitérminos que caben en la función OR. En las salidas pueden haber biestables, útiles para realizar sistemas secuenciales sencillos.

A partir de esta simple arquitectura se definen las PALs universales. La arquitectura de la matriz de ANDs cableadas y interconexiones con realimentación apenas cambia, la única diferencia estriba que en las salidas en vez de tener flip-flops simples se tienen macroceldas. Las macroceldas a su vez incluyen un biestable (de tipo D generalmente) y lógica para multiplexar la salida y la entrada a él. El ejemplo más popular de PAL universal es la PAL $22V10$, en la que se pueden implementar máquinas de estados y sistemas secuenciales sencillos.

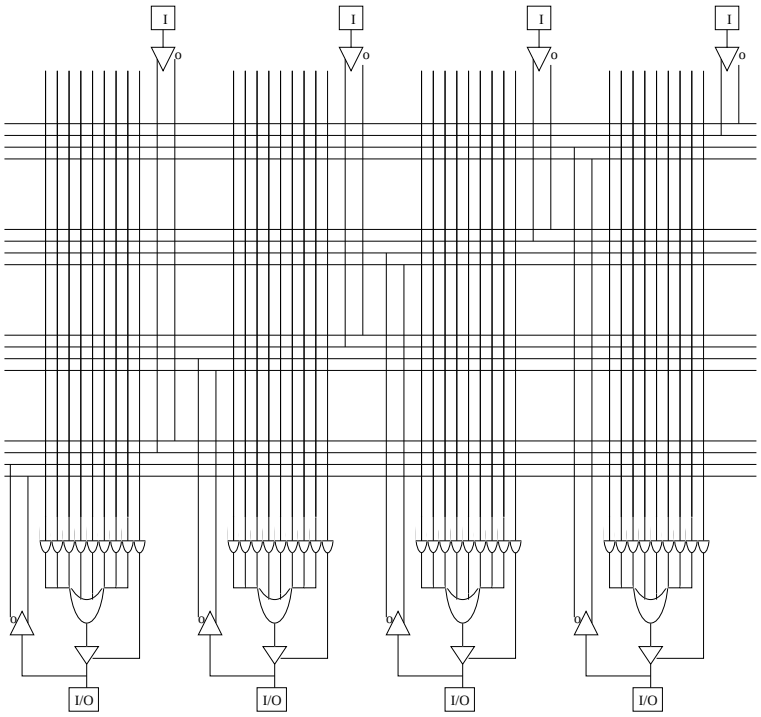


Figura A.1: *Arquitectura de una PAL genérica*

A.2.1 Dispositivos programables lógicos complejos: CPLDs

A partir de las PALs, que son las PLDs más sencillas, se construyeron las PLDs complejas o CPLDs. En la figura A.2 se puede observar la arquitectura genérica de las CPLDs de la familia Max7000 de Altera, siendo todas las arquitecturas de CPLDs actuales muy similares [BR96].

Las CPLDs tienen unos bloques internos, llamados bloques de matrices lógicas o LABs (*logic array blocks*), con una estructura similar a la arquitectura PAL mostrada en la figura A.1. Además de estas pequeñas PALs, las CPLDs tienen una matriz de interconexión programable, que permite el conexionado de los LABs entre si y, por tanto, la realización de sistemas más complejos que con las PALs. Junto con los LABs, las CPLDs tienen unos bloques especiales de entrada/salida que incorporan salidas triestado y también útiles para suministrar corriente.

Las CPLDs son dispositivos realmente complejos que pueden llegar a incorporar más de 20.000 puertas lógicas equivalentes con más de 50 LABs. El mercado de CPLDs cambia muy rápidamente, ofreciendo los fabricantes cada vez CPLDs más complejas y con mayores prestaciones de velocidad, integración y menor consumo de potencia. Cualquier análisis del estado de la tecnología a la escritura de este capítulo sería superado unos pocos meses después.

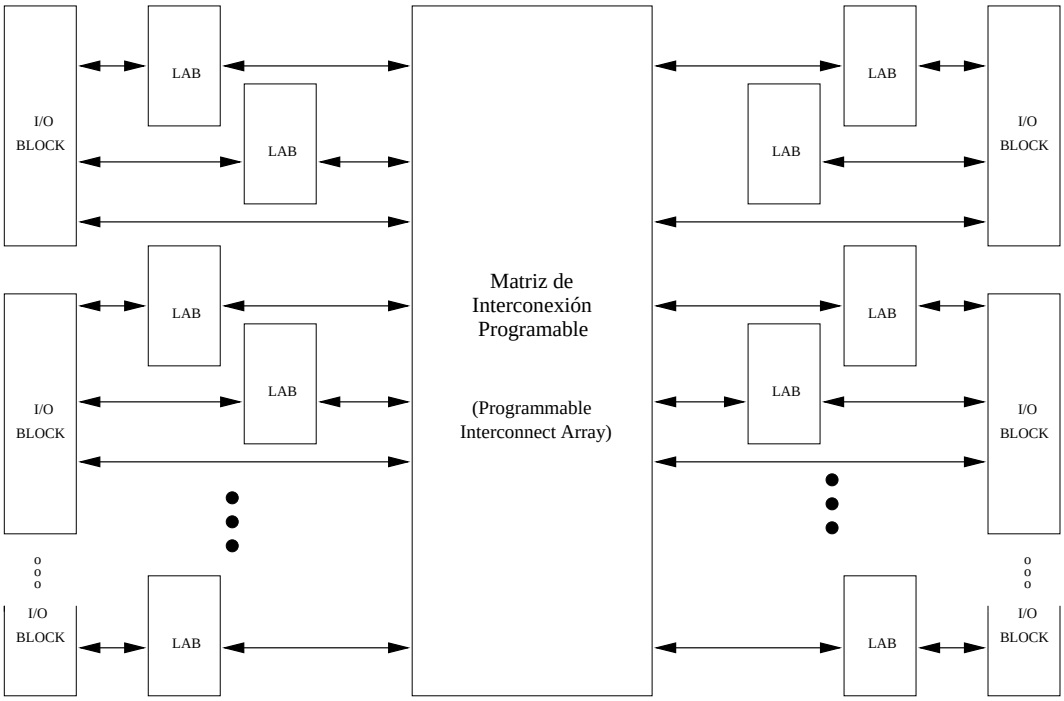


Figura A.2: *Arquitectura de las CPLDS de la familia Max 7000 de ALTERA*

La principal característica de las CPLDs son sus prestaciones. La arquitectura de las CPLDs hace que se puedan implementar funciones lógicas complejas con pocos niveles de realimentación, y por tanto, pocos retrasos [BR96]. La arquitectura de las CPLDs también hace que se puedan predecir los retrasos de una manera sencilla y así el diseñador puede prever las prestaciones de su diseño (velocidad de funcionamiento) incluso antes de implementarlo.

A.2.2 Redes de puertas lógicas programables: FPGAs

Paralelamente al desarrollo de las CPLDs se han desarrollado las redes de puertas lógicas programables o FPGAs (*Field Programmable Gate Arrays*). En la figura A.3 puede observarse la arquitectura genérica de una FPGA clásica

Al contrario de las CPLDs, aquí no se tienen varios bloques complejos como los LABs y una matriz de interconexión localizada. En las FPGAs se tienen muchas pequeñas celdas lógicas distribuidas regularmente por su superficie. Cada celda lógica incluye uno o dos flip-flops y unas pocas pequeñas LUTs (*lookup tables*) que pueden implementar funciones combinacionales sencillas. Las celdas lógicas suelen incorporar también algún multiplexor para interconectar las LUTs con los flip-flops. Una buena revisión de los principios generales de la arquitectura genérica de las FPGAs clásicas puede encontrarse en [Bro96] y [Bur96a].

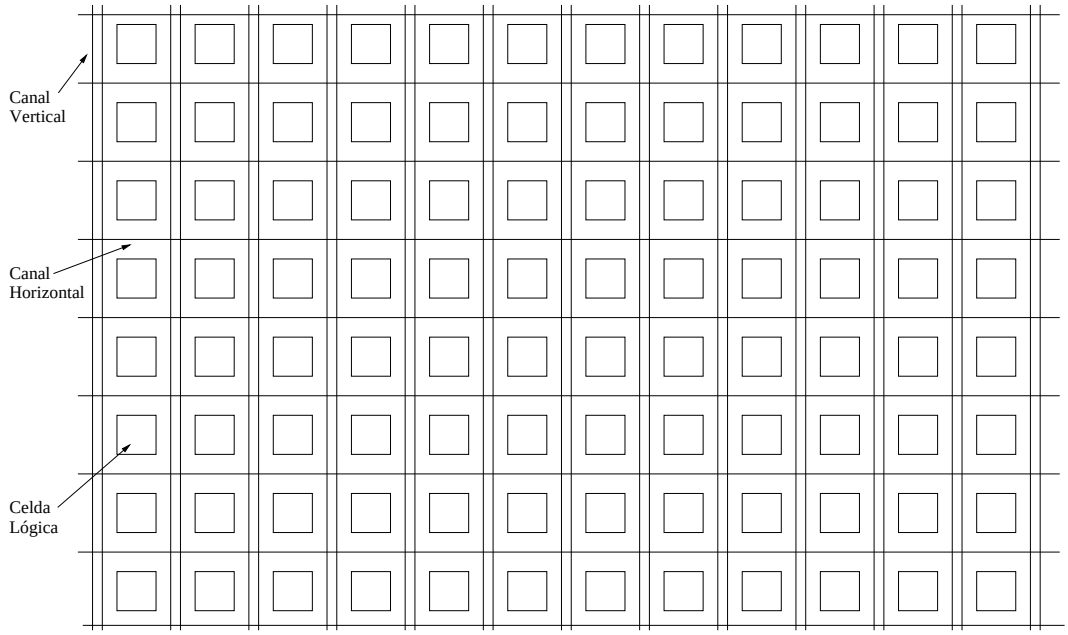


Figura A.3: *Arquitectura de una FPGA genérica*

El conexionado entre las diversas celdas lógicas no se centraliza en una matriz de interconexión sino que se realiza mediante canales de interconexión. Estos canales están distribuidos uniformemente entre las celdas lógicas y cruzan la superficie de la FPGA horizontal y verticalmente.

La ventaja de esta arquitectura regular es que, aprovechando la alta escala de integración de la tecnología VLSI, se pueden realizar dispositivos muy complejos, bastante más que las CPLDs más avanzadas. A la fecha de la escritura del presente capítulo las FPGAs más grandes podían tener una complejidad de hasta 250.000 puertas equivalentes con más de 10.000 flip-flops y celdas lógicas.

La desventaja de la arquitectura FPGA parece clara a la vista de la figura A.3; para realizar un sistema complejo se deben realizar muchas conexiones entre las celdas lógicas, generando caminos realmente largos y realimentaciones profundas, por tanto grandes retrasos impredecibles a priori [BKV96].

Como conclusión a la revisión de la arquitectura genérica de las FPGAs y las CPLDs se tiene que a mayor complejidad (FPGAs) se obtiene una mayor perdida de velocidad de funcionamiento.

A.3 Arquitecturas híbridas

El problema de la pérdida de prestaciones con las FPGAs al implementar sistemas complejos ha sido abordado por los diferentes fabricantes de PLDs. La nueva filosofía de diseño consiste en realizar arquitecturas híbridas que intentan combinar los beneficios de ambas arquitecturas. Se intenta mantener la velocidad de las CPLDs mediante una nueva arquitectura de conexionado, agrupando las celdas lógicas elementales en estructuras más grandes. Por otra parte se intenta mantener la granularidad de las FPGAs difundiendo un gran número de estos bloques más grandes entre las líneas de interconexión [FGP+97].

En la figura A.4 se muestra, como ejemplo, la arquitectura híbrida de la familia FLEX 8000 de Altera.

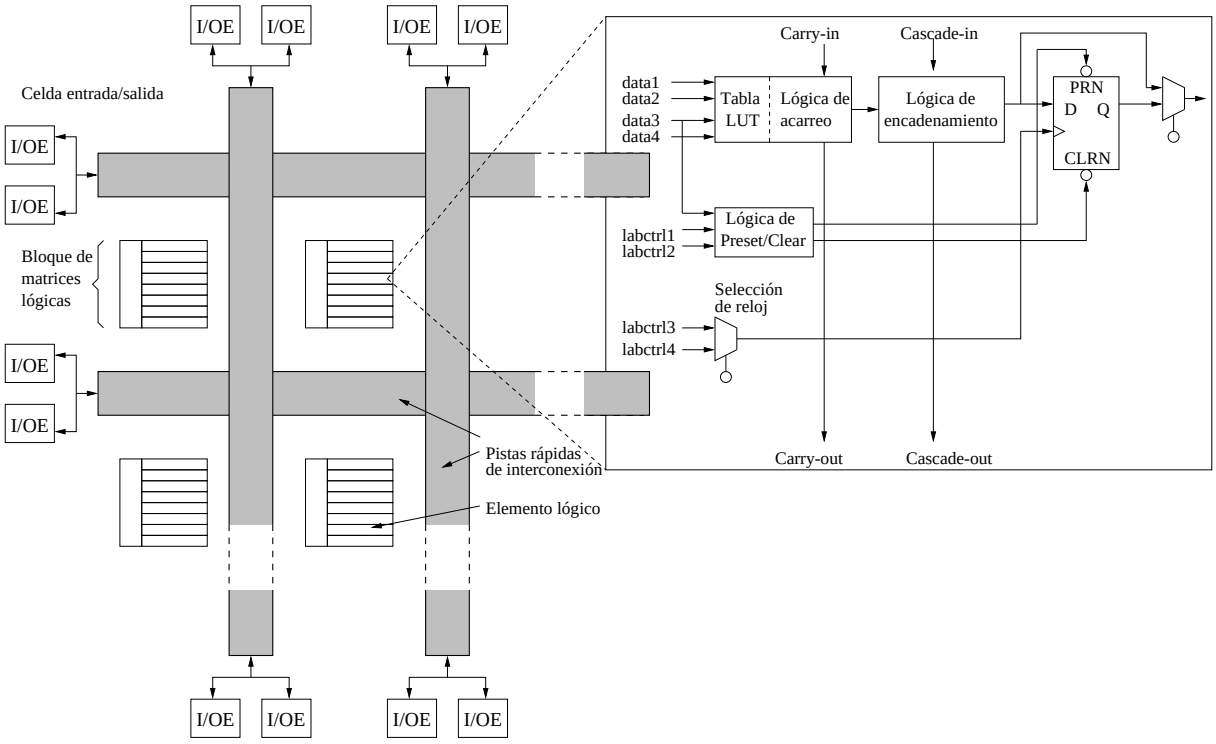


Figura A.4: *Arquitectura de la familia FLEX 8000 de Altera*

Se puede observar que en esta arquitectura las celdas lógicas están agrupadas en bloques, que en el caso de la figura, son de 8 elementos lógicos. Esta agrupación por bloques facilita el agrupamiento de las señales y la aparición de buses durante la síntesis. En el caso de que se traten señales de 8 bits, un solo bloque cubriría una señal, evitando la aparición de retrasos distintos en los diferentes bits de la señal.

Además, la estructura de los elementos lógicos incluye líneas especiales de acarreo adelantado y de encadenamiento. De esta manera se mejora la velocidad de contadores

y otros dispositivos lógicos con realimentaciones y acarreo. La interconexión de los bloques de matrices lógicas se realiza mediante pistas rápidas de interconexión similares a las matrices de interconexión de las CPLDs.

Se puede concluir que las arquitecturas híbridas combinan cierta granularidad y flexibilidad, al igual que las FPGAs, con la agrupación por bloques, y por tanto rapidez, de las CPLDs.

De nuevo el mercado de los circuitos híbridos o FPGAs avanzadas, cambia con gran rapidez. Cabe hacer notar que de nuevo se está hablando de una escala de integración, a la escritura de esta tesis, de 250.000 puertas equivalentes.

A.4 Mega-estructuras

A partir de las nuevas arquitecturas híbridas, y como consecuencia de la alta escala de integración que es está consiguiendo con la tecnología SRAM, se han desarrollado recientemente (principios de 1999) dispositivos programables más complejos [APE99].

Las mega-estructuras desarrolladas por Altera con su familia APEX20k y Xilinx con su familia VIRTEX XCV00 son un paso más en la complejidad de los dispositivos programables. Tal como se muestra en la figura A.5, la alta escala de integración consigue agrupar los bloques lógicos de las arquitecturas híbridas en un nivel jerárquico superior llamado megabloques. La interconexión de los bloques lógicos en megabloques se realiza por pistas rápidas de conexión, de la misma manera que la interconexión entre megabloques.

Esta agrupación de bloques lógicos (16 en la familia APEX) con pistas rápidas permite la aparición de buses con unas altas prestaciones. La alta escala de integración permite en estas familias llegar a 200 megabloques, más de 2 millones de puertas equivalentes, más de 40.000 elementos lógicos y medio millón de bits de RAM, con encapsulados de más de 700 pines de entrada/salida de usuario.

Estas cifras de integración, junto con las altas prestaciones conseguidas (del orden de 100 MHz), provocarían una disipación de potencia inaceptable, lo que ha obligado a desarrollar estas familias como lógica programable de baja tensión (desde 1'8V hasta 3'3V).

El principal problema de estas familias a la escritura del presente capítulo es su elevado precio, lo cual desaconseja su utilización para el módulo reconfigurable.

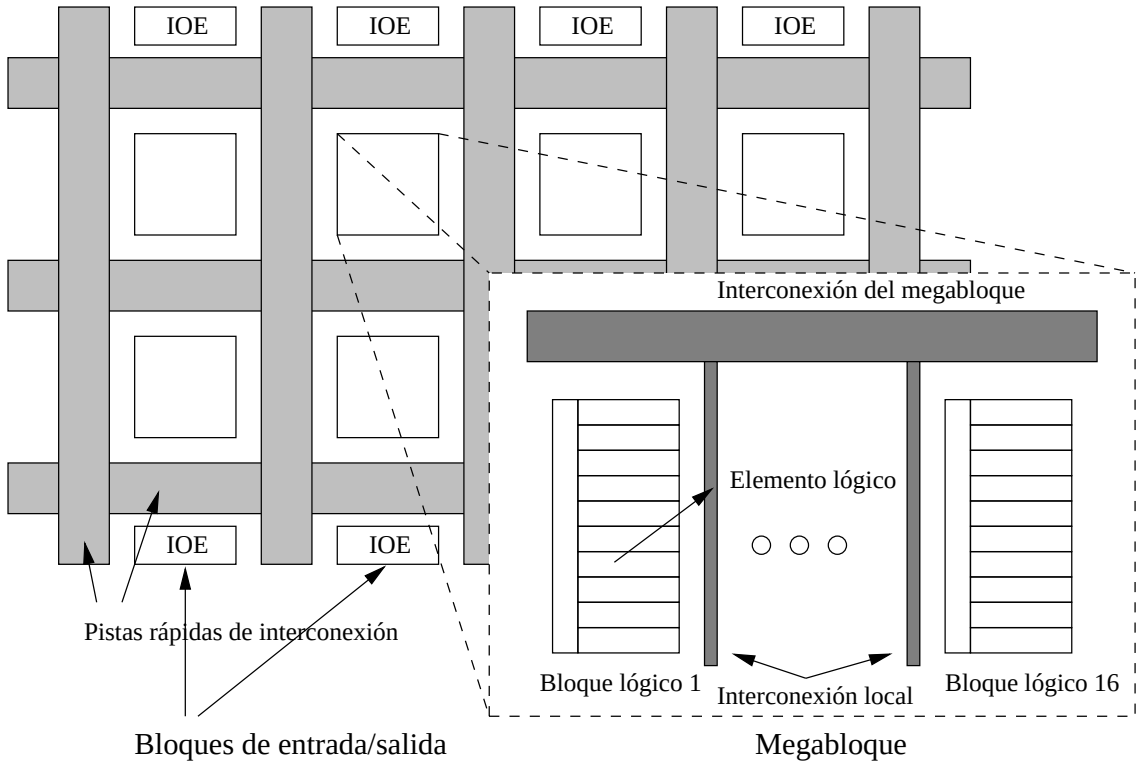


Figura A.5: *Arquitectura de la familia APEX20K de Altera*

Apéndice B

Código VHDL

El código VHDL que se muestra en este apéndice corresponde a la versión final de las etapas de los dos algoritmos de programación del módulo reconfigurable que se han descrito en el presente trabajo de investigación. Se ha realizado mucho más código con el objeto de depurar los EPs y el correcto funcionamiento de cada etapa del algoritmo.

Los programas VHDL que se muestran en las siguientes secciones son básicamente una máquina de estados implementada utilizando la metodología descrita en el capítulo 5. La interfase descrita en la entidad del código VHDL de todas las etapas no cambia ya que se ha realizado un solo circuito impreso para los EPs. De esta manera la compilación en el dispositivo programable se ha realizado con la misma asignación de pines en todas las etapas.

Entre las diversas etapas en la entidad sólo varía en la nomenclatura el número que indica el número de etapa en el cauce en el caso de que sea una salida (o bidireccional), y el número del EP del que proviene en el caso de que sea una entrada. De esta manera para la primera etapa las salidas de la FPGA (y por extensión del EP) tienen asociadas el número 1 al final de la etiqueta correspondiente al pin. Como ejemplo en la primera etapa la señal de salida que indica al EP siguiente que el dato ya está listo tiene la etiqueta `data_ready1`. La señal de entrada proveniente del siguiente EP que indica que el dato ya ha sido capturado tiene la etiqueta `data_received2`. Solamente las señales globales a todos los EPs del cauce (`clk` y `reset` no tienen número en la etiqueta identificativa.

B.1 Código VHDL de la etapa de suavizado

Tanto el algoritmo de detección de movimiento independiente del movimiento de la cámara, como el algoritmo de cálculo de tiempo al impacto, tienen como primera etapa un EP que realiza el suavizado de la imagen log-polar. En esta sección se muestra el código VHDL que realiza este suavizado en ambas etapas, tal como se describe en la sección 6.4.1.

El código implementa básicamente en un `PROCESS` una máquina con un ciclo de 4 estados, correspondiendo el estado `Sini` al estado de inicialización o reset activo a nivel bajo. El byte transmitido de la imagen log-polar se almacena en el registros `d1`, produciéndose un desplazamiento de manera que `d1<=data_in0; c1<=d1; i1<=c1;` y `mem_local_out<=i1;`. El dato almacenado en `mem_local_out` se almacenará en memoria local en la posición apuntada por `puntero`. Simultáneamente a la recepción del byte de la imagen log-polar proveniente de la tarjeta de adquisición, se lee de la posición apuntada por `puntero` + 125, para de esta manera completar una fila log-polar de 128 bytes. Este dato se almacena en `d2`, produciéndose un desplazamiento de manera que `d2<=mem_local_in; c2<=d2;` y `i2<=c2;`. La figura 6.1 muestra el desplazamiento de los bytes para realizar la convolución con la máscara unitaria de 6 bytes. Claramente la convolución de la primera fila será incorrecta, pero esto tiene poca importancia ya que la primera fila corresponde a un solo *pixel*.

Simultáneamente a la escritura en memoria local se escribe en memoria de doble puerto para que el siguiente EP pueda calcular la derivada primera. Se tiene en cuenta si el byte que se va a escribir corresponde a un cambio de imagen verificando si ha variado el valor del pin `img0`. Este control de cambio de imagen es útil para seleccionar la memoria de doble puerto en la que se escribe la imagen mediante `CE1`. Cabe hacer notar como está previsto en la máquina de la etapa de suavizado no activar la señal `data_ready1` hasta que se haya procesado completamente una imagen (la señal `primera` pasa a valer 0).

Fuera del `PROCESS` permanece la parte meramente combinacional de suma de los valores de la máscara de convolución. El valor sumado se guarda en la señal `suma` de 11 bits, seleccionándose los 7 bits más significativos para el byte de salida y escritura en la memoria de doble puerto.

```
-- Modulo de suavizado de la imagen. Hace una convolución de 3x2 usando máscara unitaria.

LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_arith.all;

ENTITY convoluc IS PORT(
  clk, reset, data_received2, data_ready0, img0 : IN std_logic;
  data_in0, mem_izda0 : IN unsigned(7 DOWNTO 0);
  mem_local1 : INOUT unsigned(7 DOWNTO 0);
  addr_dcha1 : BUFFER unsigned( 13 DOWNTO 0);
  addr_izda1 : OUT unsigned( 13 DOWNTO 0);
  addr_local1 : OUT unsigned( 14 DOWNTO 0);
  data_out1 : OUT unsigned(7 DOWNTO 0);
  data_ready1, data_received1, RW1 : OUT std_logic;
  CE1 : OUT std_logic_vector(2 DOWNTO 0);     --Activas a nivel bajo
  OE1 : BUFFER std_logic_vector(2 DOWNTO 0);  --Activas a nivel bajo
  img1 : BUFFER std_logic);
END convoluc;

ARCHITECTURE transferencias OF convoluc IS
  TYPE tipo_estados IS (Sini, S3, S2, S1, S0);
  SIGNAL estado : tipo_estados;
  SIGNAL d2, d1, c2, c1, i1, i2, mem_local_in, mem_local_out : unsigned(7 DOWNTO 0);
  SIGNAL suma : unsigned(10 DOWNTO 0);
  SIGNAL primera, OE_mem, img_anterior : std_logic;
  SIGNAL puntero : unsigned(14 DOWNTO 0);
  CONSTANT desplazamiento : unsigned(13 DOWNTO 0):="00000001111101";

BEGIN
 PROCESS (clk, reset)
   BEGIN
   IF reset='0' THEN estado<=Sini;
   ELSIF (clk'event AND clk='1') THEN
     CASE estado IS
        WHEN Sini => --Inicializaciones previas.
                     --Si data_ready0='1' se lee de la mem local y del frame-grabber
                     CE1<="011"; OE1(2)<='0'; OE1(1)<='1'; RW1<='1'; img1<='0';
                     data_ready1<='0'; data_received1<='0'; addr_izda1<=(OTHERS=>'0');
                     puntero<=(OTHERS=>'0'); addr_dcha1<=(OTHERS=>'0');
                     img_anterior<='0'; OE_mem<='0';
                     addr_local1<=(OTHERS=>'0'); primera<='1';
                     IF data_ready0='1' THEN d1<=data_in0; d2<=mem_local_in;
                                             i2<=c2; i1<=c1; c2<=d2; c1<=d1;
                                             estado<=S0; data_received1<='1';
                                        ELSE estado<=Sini;
                     END IF;

        WHEN S0 => --Se prepara escritura en mem_local y mem_derecha
                   OE_mem<='1'; mem_local_out<=i1; RW1<='0'; OE1(2)<='1';
                   addr_local1<=puntero+desplazamiento;  data_received1<='0';
                   IF img1='0' THEN CE1<="001";
                               ELSE CE1<="010";
                   END IF;
                   estado<=S1;

        WHEN S1 => --Se escribe
                   IF primera='1' THEN data_ready1<='0'; ELSE data_ready1<='1'; END IF;
                   CE1<="111";  mem_local_out<=i1;
                   IF data_received2='1' OR primera='1' THEN estado<=S2;
                                                        ELSE estado<=S1;
                   END IF;

        WHEN S2 => --Se cierra el bloque preparando lectura
                   OE_mem<='0'; OE1(2)<='0'; OE1(1)<='1';
                   CE1<="011"; RW1<='1'; data_ready1<='0';
                   IF (data_ready0='1' AND img_anterior/=img0)
                                          THEN img_anterior<=NOT(img_anterior);
```

```
                                                addr_local1<=(OTHERS=>'0');
                                                addr_dcha1<=(OTHERS=>'0');
                                                puntero<=(OTHERS=>'0');
                                                IF img1='0' THEN img1<='1';
                                                           ELSE img1<='0';
                                                END IF;
                                                estado<=S3; data_received1<='1';
                    ELSIF (data_ready0='1' AND img_anterior=img0)
                                              THEN addr_dcha1<=addr_dcha1+1;
                                                data_received1<='1';
                                                addr_local1<=puntero+1;
                                                puntero<=puntero+1;
                                                estado<=S3;
                                              ELSE estado<=S2;
                    END IF;

          WHEN S3 => --Saco la espera de data_ready del estado anterior
                     --para evitar incremento de punteros
                     IF (primera='1' AND img0='1') THEN primera<='0'; END IF;
                     d1<=data_in0; d2<=mem_local_in; i2<=c2;
                     i1<=c1; c2<=d2; c1<=d1; data_received1<='0';
                     estado<=S0;

      END CASE;
    END IF;
  END PROCESS;

  suma<=d2+d1+c2+c1+i1+i2;
  data_out1(6 DOWNTO 0)<=suma(10 DOWNTO 4); data_out1(7)<='0'; --Dejo 7 bits
  mem_local_in<=mem_local1; --Datos mem local puerto bidireccional
  mem_local1<=mem_local_out WHEN OE_mem='1' ELSE (OTHERS=>'Z');
  OE1(0)<=NOT(OE1(1));      --Así evito que OE2 sea 000 durante el reset

END transferencias;
```

B.2 Código VHDL de la etapa del cálculo de la primera derivada temporal

La etapa de cálculo de la segunda derivada temporal accede simultáneamente a los datos suministrados por la FPGA y a los datos almacenados en la memoria de doble puerto del EP anterior. Para ello selecciona mediante la señal Œ2 la memoria de la cual debe leer la imagen anterior. Cuando la señal `img1` cambia su valor respecto al valor que tenía en el byte anterior se entiende que se ha producido un cambio de imagen y se permuta la selección de la memoria de lectura.

Una vez se ha capturado el byte de la imagen presente suministrada por la FPGA y el byte de la imagen almacenada en la memoria de doble puerto se valida como buena la resta de ambos. Cabe hacer notar que la resta es totalmente combinacional y se realiza fuera del `PROCESS`, almacenándose en el registro intermedio `intermedia`. El valor de esta diferencia se expresa en complemento a 2 y no tiene desbordamiento al tener los datos originales 7 bits y almacenarse el resultado en 8 bits.

Una vez la diferencia es correcta y se valida como tal se escribe simultáneamente en memoria de doble puerto para que el tercer y último EP calcule la derivada segunda. La selección de la memoria en la que se va a escribir se realiza tras verificar que la imagen ha cambiado comprobando la señal de entrada `img1`. Si esto ocurre se permuta la memoria en la que se va a escribir (y leer) y se indica este cambio al EP siguiente invirtiendo la señal de salida `img2`. De nuevo se tiene en cuenta que debe haber procesado una imagen log-polar completa antes de activar la señal `data_ready2` ya que el último EP necesitará de ésta para calcular la derivada segunda.

```
-- Calculo de la Primera derivada temporal. Resto la imagen producida por el modulo
-- de suavizado a la almacenada en memoria. No se usa la memoria local.

LIBRARY ieee;
       USE ieee.std_logic_1164.all;
       USE ieee.std_logic_arith.all;

ENTITY resta1 IS PORT(
        clk, reset, data_received3, data_ready1, img1 : IN std_logic;
        data_in1, mem_izda1 : IN unsigned(7 DOWNTO 0);
        mem_local2 : IN unsigned(7 DOWNTO 0);
        addr_dcha2, addr_izda2 : BUFFER unsigned( 13 DOWNTO 0);
        addr_local2 : OUT unsigned( 14 DOWNTO 0);
        data_out2 : OUT signed(7 DOWNTO 0);
        RW2, data_ready2, data_received2 : OUT std_logic;
        CE2 : OUT std_logic_vector(2 DOWNTO 0);
        OE2 : BUFFER std_logic_vector(2 DOWNTO 0);
        img2 : BUFFER std_logic);
END resta1;


ARCHITECTURE maquina OF resta1 IS
 TYPE tipo_estados IS (Sini, S3, S2, S1, S0);
 SIGNAL estado : tipo_estados;
 SIGNAL intermedia, data_in_aux, mem_izda_aux : unsigned(7 DOWNTO 0);
 SIGNAL img_anterior, img_aux, primera : std_logic;
 BEGIN
   PROCESS (clk, reset)
     BEGIN
       IF reset='0' THEN estado<=Sini;
       ELSIF (clk'event AND clk='1') THEN
         CASE estado IS
                WHEN Sini => --Inicializaciones previas
                            data_ready2<='0'; img2<='0'; addr_dcha2<=(OTHERS=>'0');
                            addr_izda2<=(OTHERS=>'0'); addr_local2<=(OTHERS=>'0');
                            addr_local2<=(OTHERS=>'0'); RW2<='0'; img_anterior<='1';
                            OE2(1)<='0'; OE2(2)<='1';
                            IF data_ready1='1' THEN  data_in_aux<=data_in1; img_aux<=img1;
                                                     data_received2<='1'; estado<=S0;
                                               ELSE  estado<=Sini;
                            END IF;

                WHEN S0 => --Cojo el Dato de mem_izda y preparo escritura.
                           data_received2<='0'; mem_izda_aux<=mem_izda1;
                           IF img_aux='1' THEN CE2<="101";
                                          ELSE CE2<="110";
                           END IF;
                           estado<=S1;

                WHEN S1 => --La resta es buena. Escribo.
                           CE2<="111";
                           IF primera='1' THEN data_ready2<='0';
                                          ELSE data_ready2<='1';
                           END IF;
                           IF (primera='1' OR data_received3='1') THEN estado<=S2;
                                                                  ELSE estado<=S1;
                           END IF;

                WHEN S2 => --Espero siguiente dato
                           data_ready2<='0';
                           IF data_ready1='1' THEN data_in_aux<=data_in1; data_received2<='1';
                                                   img_aux<=img1; estado<=S3;
                                              ELSE estado<=S2;
                           END IF;

                WHEN S3 => --He cambiado de imagen? Actualizo punteros.
                           IF img_aux/=img_anterior THEN addr_dcha2<=(OTHERS=>'0');
                                                         addr_izda2<=(OTHERS=>'0');
```

```
                                                    img_anterior<=NOT(img_anterior);
                                                    primera<='0';
                                                    IF img2='1' THEN img2<='0';
                                                                ELSE img2<='1';
                                                    END IF;
                                               ELSE addr_izda2<=addr_izda2+1;
                                                    addr_dcha2<=addr_dcha2+1;
                          END IF;
                          IF img_aux='1' THEN OE2(1)<='0';
                                         ELSE OE2(1)<='1';
                          END IF;
                          estado<=S0;

        END CASE;
     END IF;
   END PROCESS;

   intermedia<=data_in_aux-mem_izda_aux;
   data_out2<=CONV_SIGNED (intermedia, 8);
   OE2(0)<=NOT(OE2(1)); --Así evito que OE2 sea 000 durante el reset
END maquina;
```

B.3 Código VHDL de la etapa del cálculo de la segunda derivada y binarización

Esta etapa es la última del algoritmo de detección de movimiento y accede a las imágenes suministradas por la FPGA y a las imágenes almacenadas en las memorias de doble puerto del EP anterior de la manera descrita anteriormente. De nuevo el código es una máquina VHDL con un ciclo de 4 estados, permaneciendo fuera del `PROCESS` la etapa combinacional. En esta etapa combinacional se evalúa que el valor de la segunda derivada sea mayor que el módulo del umbral definido por la variable `treshold`. Si es así se marca el punto con valor negro mediante la instrucción `data_out3<=(OTHERS=>'1')`.

```
--Calculo de la segunda derivada temporal restando a la imagen que viene de resta1
--y le aplico un umbral.

LIBRARY ieee;
        USE ieee.std_logic_1164.all;
        USE ieee.std_logic_arith.all;

ENTITY umbral IS PORT(
        clk, reset, data_received4, data_ready2, img2 : IN std_logic;
       data_in2, mem_izda2, mem_local3 : IN signed(7 DOWNTO 0);
       addr_dcha3, addr_izda3 : BUFFER signed( 13 DOWNTO 0);
       addr_local3 : OUT signed(14 DOWNTO 0);
       data_out3 : OUT signed(7 DOWNTO 0);
       data_ready3, data_received3, RW3 : OUT std_logic;
       CE3 : OUT std_logic_vector(2 DOWNTO 0);
       OE3 : BUFFER std_logic_vector(2 DOWNTO 0);
       img3 : BUFFER std_logic);
END umbral;

ARCHITECTURE maquina OF umbral IS
  TYPE tipo_estados IS (Sini, S3, S2, S1, S0);
  SIGNAL estado : tipo_estados;
  SIGNAL intermedia, data_in_aux, mem_izda_aux : signed(7 DOWNTO 0);
  SIGNAL img_anterior, img_aux : std_logic;
  CONSTANT treshold : integer := 17;
  BEGIN
   PROCESS (clk, reset)
     BEGIN
      IF reset='0' THEN estado<=Sini;
      ELSIF (clk'event AND clk='1') THEN
          CASE estado IS

                WHEN Sini => --Inicializaciones previas
                           data_received3<='0'; data_ready3<='0'; img3<='0';
                           addr_izda3<=(OTHERS=>'0'); img_anterior<='1';
                           OE3(2)<='1'; OE3(1)<='0'; RW3<='0'; CE3<="111";
                           addr_dcha3<=(OTHERS=>'0'); addr_local3<=(OTHERS=>'0');
                           IF data_ready2='1' THEN  data_in_aux<=data_in2;
                                                    data_received3<='1'; img3<='1';
                                                    img_aux<=img2; estado<=S0;
                                                ELSE estado<=Sini;
                           END IF;

                WHEN S0 => --Cojo data memoria Izda.
                          data_received3<='0'; mem_izda_aux<=mem_izda2;
                          estado<=s1;

                WHEN S1 => --Valido dato
                          data_ready3<='1';
```

```
                        IF data_received4='1' THEN estado<=S2;
                                              ELSE estado<=S1;
                        END IF;

                WHEN S2 =>
                        data_ready3<='0';
                        IF data_ready2='1' THEN data_in_aux<=data_in2;
                                                data_received3<='1';
                                                img_aux<=img2; estado<=S3;
                                           ELSE estado<=S2;
                        END IF;
                        IF img2='1' THEN OE3(1)<='0';
                                    ELSE OE3(1)<='1';
                        END IF; --Si img1=1 Leo de abajo.

                WHEN S3 => --Compruebo el cambio de imagen;
                        IF img_aux/=img_anterior THEN addr_izda3<=(OTHERS=>'0');
                                                      img_anterior<=NOT(img_anterior);
                                                      IF img3='1' THEN img3<='0';
                                                                  ELSE img3<='1';
                                                      END IF;
                                                 ELSE  addr_izda3<=addr_izda3+1;
                        END IF;
                        IF img_aux='1' THEN OE3(1)<='0';
                                       ELSE OE3(1)<='1';
                        END IF; --Si img1=1 Leo de abajo.
                        estado<=S0;

            END CASE;
        END IF;
       END PROCESS;

   intermedia<=data_in_aux-mem_izda_aux;
   data_out3<=(OTHERS=>'1') WHEN (intermedia>treshold OR intermedia<-treshold) ELSE (OTHERS=>'0');
   OE3(0)<=NOT(OE3(1));
END maquina;
```

B.4 Código VHDL de la etapa del cálculo del gradiente y de la primera derivada temporal

La implementación del algoritmo de cálculo del tiempo al impacto se utilizan tres etapas implementadas en otros tantos EPs. La primera etapa corresponde exactamente a la misma etapa de suavizado descrita al inicio de este apéndice. La estrategia de almacenamiento de imágenes seguida es igual de válida ya que la etapa siguiente debe de calcular igualmente la derivada primera temporal.

A continuación se muestra el código VHDL correspondiente a la segunda etapa del algoritmo de cálculo de tiempo al impacto. Tal como se ha descrito en el capítulo 7 esta segunda etapa calcula la derivada primera temporal y el gradiente radial de las imágenes log-polares suavizadas. La máquina de estados implementada tiene un ciclo de 4 estados, accediendo a la memoria de doble puerto del EP anterior de la misma manera que la segunda etapa del algoritmo de detección de movimiento. De manera adicional simultáneamente se accede a memoria local para acceder al dato correspondiente a la fila anterior y calcular el gradiente radial.

En primer lugar se hace la diferencia entre la imagen suministrada por la FPGA del EP anterior y la imagen log-polar almacenada en memoria de doble puerto. El cambio de imagen se indica mediante un cambio en la señal `img1`. Una vez se ha validado por parte del siguiente EP la recepción de la derivada temporal se suministra la diferencia radial al siguiente EP.

```
--Calculo del gradiente y de la primera derivada temporal.
LIBRARY ieee;
        USE ieee.std_logic_1164.all;
        USE ieee.std_logic_arith.all;

ENTITY grad_der IS PORT(
        clk, reset, data_received3, data_ready1, img1 : IN std_logic;
        data_in1,  mem_izda1 : IN unsigned(7 DOWNTO 0);
        mem_local2 : INOUT unsigned(7 DOWNTO 0);
        addr_izda2 : BUFFER unsigned(13 DOWNTO 0);
        addr_dcha2 : OUT unsigned(13 DOWNTO 0);
        addr_local2 : BUFFER unsigned(14 DOWNTO 0);
        data_out2 : OUT unsigned(7 DOWNTO 0);
        data_ready2, data_received2, RW2 : OUT std_logic;
        CE2 : OUT std_logic_vector(2 DOWNTO 0);
        OE2 : BUFFER std_logic_vector(2 DOWNTO 0);
        img2 : BUFFER std_logic);
END grad_der;

ARCHITECTURE maquina OF grad_der IS
 TYPE tipo_estados IS (Sini, S0, S1, S2, S3);
 SIGNAL estado : tipo_estados;
 SIGNAL img_anterior, img_aux, OE_mem : std_logic;
 SIGNAL data_in1_aux, mem_local2_in_aux, mem_local2_in, mem_local2_out : unsigned(7 DOWNTO 0);
 SIGNAL puntero : unsigned(14 DOWNTO 0);
 CONSTANT desplazamiento : unsigned(14 DOWNTO 0):="000000010000000"; -- Mide cada fila 128
 BEGIN
 PROCESS (clk, reset)
```

```
   BEGIN
   IF reset='0' THEN estado<=Sini;
   ELSIF (clk'event AND clk='1') THEN
        CASE estado IS
             WHEN Sini => --Inicializaciones previas. Cojo 1er dato.
                          data_ready2<='0';  addr_izda2<=(OTHERS=>'0'); img2<='0';
                          img_anterior<='1'; OE_mem<='0'; OE2(2)<='0'; OE2(1)<='0';
                          RW2<='1'; CE2<="011"; addr_local2<=(OTHERS=>'0');
                          puntero<=(OTHERS=>'0'); addr_dcha2<=(OTHERS=>'0');
                          IF data_ready1='1' THEN  data_in1_aux<=data_in1;
                                                   data_received2<='1'; img_aux<=img1;
                                                   estado<=S0;
                                              ELSE estado<=Sini; data_received2<='0';
                          END IF;

             WHEN S0 => --Cojo el Dato de mem_izda y mem local.
                        --Pongo la derivada temporal y la valido
                        data_out2<=mem_izda1-data_in1_aux ;
                        mem_local2_in_aux<=mem_local2_in;
                        OE_mem<='0'; data_received2<='0';
                        IF img_aux/=img_anterior THEN img_anterior<=NOT(img_anterior);
                                                      img2<=NOT(img2);
                        END IF;
                        IF data_received3='1' THEN data_ready2<='0'; estado<=S1;
                                              ELSE data_ready2<='1'; estado<=S0;
                        END IF;

             WHEN S1 => --Preparo escritura en mem local pongo el gradiente y lo valido
                        OE_mem<='1'; RW2<='0'; addr_local2<=puntero+desplazamiento;
                        mem_local2_out<=data_in1_aux;
                        data_out2<=data_in1_aux-mem_local2_in_aux;
                        IF data_received3='1' THEN data_ready2<='0'; estado<=S2;
                                              ELSE data_ready2<='1'; estado<=S1;
                        END IF;

             WHEN S2 => --Escribo en mem local y espero nuevo dato
                        CE2<="111";
                        IF data_ready1='1' THEN data_in1_aux<=data_in1;
                                                data_received2<='1';
                                                img_aux<=img1; estado<=S3;
                                           ELSE estado<=S2; data_received2<='0';
                        END IF;

             WHEN S3 => -- Preparo la nueva lectura de mem_local, mem_izda
                        CE2<="011"; RW2<='1'; OE2(2)<='0'; OE_mem<='0';
                        IF img_aux='1' THEN OE2(1)<='0';
                                       ELSE OE2(1)<='1';
                        END IF;
                        IF img_aux/=img_anterior THEN addr_izda2<=(OTHERS=>'0');
                                                      addr_local2<=(OTHERS=>'0');
                                                      puntero<=(OTHERS=>'0');
                                                 ELSE addr_izda2<=addr_izda2+1;
                                                      addr_local2<=puntero+1;
                                                      puntero<=puntero+1;
                        END IF;
                        estado<=S0;

        END CASE;
   END IF;
 END PROCESS;

 mem_local2_in<=mem_local2;
 mem_local2<=mem_local2_out WHEN OE_mem='1' ELSE (OTHERS=>'Z');
 OE2(0)<=NOT(OE2(1));

END maquina;
```

B.5 Código VHDL del algoritmo de división entera

El código que se muestra a continuación corresponde a la implementación del algoritmo clásico de división entera sin restauración. La máquina de estados tiene un bucle de `2*8=16` ciclos de reloj por byte. En primer lugar se recibe el divisor (la derivada temporal) que se almacena en el registro `M`. Posteriormente se recibe la derivada espacial que se almacena en el registro `SRQ`. Cabe hacer notar que este valor se almacena directamente desplazado 4 bits, con lo que se realiza una multiplicación del dividendo por 16.

Previamente al inicio del bucle se comprueba que se obtendrá un valor no negativo y bien definido con sentido para τ. Con esta intención se verifica que la derivada temporal y espacial tengan signos opuestos comparando los bits de signo. De manera adicional se comprueba que el divisor no sea cero y que su valor en módulo sea mayor que una cierta cantidad, tal como se ha justificado en la sección 10.3.4. Esto se realiza mediante la condición `IF (SRQ(11)=M(7) OR (SRQ(11 DOWNTO 4)=0) OR (M<2) OR (M>254))`. Si no se cumple la condición la salida será un valor de gris prefijado (en este caso el valor 11110000) que la partición software interpretará como valor de τ no válido y no lo utilizará para calcular la media del tiempo al impacto. Una vez se ha comprobado que los signos de divisor y dividendo son distintos se transforma la cantidad negativa en positiva, ya que el algoritmo de división sin restauración divide solamente cantidades positivas.

```
-- Coge la derivada y el gradiente del módulo 2 y los divide.
-- Uso el algoritmo de non-restoring division radix-2.
-- El coste es 2*n+2 en el peor caso.

LIBRARY ieee;
        USE ieee.std_logic_1164.all;
        USE ieee.std_logic_arith.all;

ENTITY division IS PORT(
        clk, reset, data_received4, data_ready2, img2 : IN std_logic;
        data_in2, mem_izda2, mem_local3 : IN unsigned(7 DOWNTO 0);
        addr_dcha3, addr_izda3 : BUFFER unsigned( 13 DOWNTO 0);
        addr_local3 : OUT unsigned(14 DOWNTO 0);
        data_out3 : OUT unsigned(7 DOWNTO 0);
        data_ready3, data_received3, RW3 : OUT std_logic;
        img3 : BUFFER std_logic;
        CE3 : OUT std_logic_vector(2 DOWNTO 0);
        OE3 : BUFFER std_logic_vector(2 DOWNTO 0));
END division;

ARCHITECTURE algoritmo OF division IS
 TYPE tipo_estados IS (Sini, S0, S1, S2, S3, S4, S5);
 SIGNAL estado : tipo_estados;
 SIGNAL M : unsigned(7 DOWNTO 0);
 SIGNAL SRQ : unsigned(16 DOWNTO 0);
 SIGNAL img_aux : std_logic;
 SIGNAL count : integer RANGE 0 TO 8;
```

```
BEGIN
 PROCESS (clk, reset)
  BEGIN
   IF reset='0' THEN estado<=Sini;
   ELSIF (clk'event AND clk='1') THEN
         CASE estado IS
               WHEN Sini => --Inicializaciones previas.
                            --Se carga la derivada temporal (divisor).
                            addr_local3<=(OTHERS=>'0'); OE3(2)<='1'; OE3(1)<='1';
                            CE3<="111"; RW3<='1'; addr_izda3<=(OTHERS=>'0');
                            img3<='0';  count<=0; addr_dcha3<=(OTHERS=>'0'); data_ready3<='0';
                            IF data_ready2='1' THEN data_received3<='1'; M<=data_in2;
                                                    img_aux<=img2; estado<=S0;
                                                ELSE data_received3<='0'; estado<=Sini;
                            END IF;

               WHEN S0 => --Estado necesario para transmitir el gradiente
                          data_received3<='0';
                          IF data_received4='1' THEN data_ready3<='0';
                          END IF;
                          IF data_ready2='0' THEN  estado<=S1;
                                             ELSE estado<=S0;
                          END IF;

               WHEN S1 => --Se carga el gradiente multiplicado por 16!!!.
                          SRQ(15 DOWNTO 12)<="0000"; SRQ(3 DOWNTO 0)<="0000";
                          IF data_ready2='1' THEN SRQ(11 DOWNTO 4)<=data_in2; img_aux<=img2;
                                                  estado<=S2; data_received3<='1';
                                             ELSE estado<=S1;
                          END IF;

               WHEN S2 => --Si el divisor es cero, si son del mismo signo,
                          --si el divisor es más grande que el numerador, o si son valores
                          --pequeños acabo la división. Paso el numero negativo a positivo.
                          data_received3<='0';
                          IF  (SRQ(11)=M(7) OR (SRQ(11 DOWNTO 4)=0) OR (M<2) OR (M>254))
                                        THEN data_out3<="11110000"; data_ready3<='1';
                                             img3<=img_aux; estado<=S5;
                          ELSIF (M(7)='1' AND (256-M<=SRQ(11 DOWNTO 0)))
                                        THEN M<=256-M; estado<=S3;
                          ELSIF (M(7)='0' AND (M<=(256-SRQ(11 DOWNTO 4))&"0000"))
                                        THEN SRQ(11 DOWNTO 4)<=256-SRQ(11 DOWNTO 4);
                                             estado<=S3;
                                        ELSE  data_out3<=(OTHERS=>'0'); data_ready3<='1';
                                              estado<=S5; img3<=img_aux;
                          END IF;

               WHEN S3 => --Hago el desplazamiento a la Izda.
                          SRQ(15)<=SRQ(14); SRQ(14)<=SRQ(13); SRQ(13)<=SRQ(12);
                          SRQ(12)<=SRQ(11); SRQ(11)<=SRQ(10); SRQ(10)<=SRQ(9);
                          SRQ(9)<=SRQ(8); SRQ(8)<=SRQ(7); SRQ(7)<=SRQ(6);
                          SRQ(6)<=SRQ(5); SRQ(5)<=SRQ(4); SRQ(4)<=SRQ(3);
                          SRQ(3)<=SRQ(2); SRQ(2)<=SRQ(1); SRQ(1)<=SRQ(0);
                          estado<=S4;

               WHEN S4 => --Pongo el bit bajo a lo que toca y resto si toca.
                          count<=count+1;
                          IF count<7 THEN estado<=S3;
                                      IF SRQ(15 DOWNTO 8)>=M THEN SRQ(0)<='1';
                                                             SRQ(15 DOWNTO 8)<=SRQ(15 DOWNTO 8)-M;
                                                                ELSE SRQ(0)<='0';
                                      END IF;
                                     ELSE estado<=S5; data_ready3<='1'; img3<=img_aux;
                                          data_out3(7 DOWNTO 1)<=SRQ(7 DOWNTO 1);
                                          IF SRQ(15 DOWNTO 8)>=M THEN data_out3(0)<='1';
                                                                 ELSE data_out3(0)<='0';
                                          END IF;
                          END IF;
```

```
                WHEN S5 => --Verifico que se ha recibido el dato
                        count<=0;
                        IF data_received4='1' THEN data_ready3<='0'; END IF;
                        IF data_ready2='1' THEN data_received3<='1';
                                                M<=data_in2; img_aux<=img2;
                                                estado<=S0;
                                           ELSE data_received3<='0'; estado<=S5;
                        END IF;

          END CASE;
        END IF;
  END PROCESS;

  OE3(0)<=NOT(OE3(1));
END algoritmo;
```