UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE INGENIERÍA

CENTRO DE INVESTIGACIÓN Y ESTUDIOS DE POSGRADO

"Implementación de Algoritmos de Visión en un Procesador Digital de Señales "

por

Ing. Ezequiel Herrera Flores

Asesor

Dr. Juan Antonio Cárdenas Galindo

TESIS

PARA OBTENER EL GRADO DE MAESTRÍA EN

INGENIERÍA ELÉCTRICA

ESPECIALIDAD EN CONTROL AUTOMÁTICO

San Luis Potosí, S.L.P., Junio de 2010

Índice general

Dedicatoria

A la memoria de mis Abuelos.

A mis padres.

Lux In Tenebris Lucet

Agradecimientos

A Dios.

A mis Padres y Hermano.

Al Dr. Juan Antonio Cárdenas Galindo, por la atención prestada a este trabajo de tesis, así también por sus valiosos conocimientos, comentarios, observaciones y correcciones dentro del proceso de obtención del grado.

Al comité evaluador de esta tesis.

Al Posgrado de Ingeníeria Eléctrica de la UASLP por el apoyo material y de su cuerpo académico.

Al CONACYT por la beca otorgada para la realización de estos estudios.

A mis amigos:

- Isela Bonilla Gutiérrez.
- María del Carmen Rodríguez Liñán.
- Marco Octavio Mendoza.
- Josue Reyes Malanche.
- Francisco Villalobos Pía.

Por su valioso apoyo y ayuda.

A mis amigos y compañeros de generación, Fabiola Campos Cornejo, Ulises Daniel Cuevas Martinez, Hugo César Hernández Juárez, Marco Antonio Hernández Sánchez, Daniel Martinez Escareño, Juan Antonio Mejía Vázquez, Octavio Vega Hernández. Gracias por su ayuda, amistad y compañía. A todos los amigos y compañeros del posgrado, en especial a Diego Rivelino, Manuel Flota, Marco Gallegos, Ambrocio Loredo, Mario Hernández y Víctor Mora. Gracias por los sanos momentos de esparcimiento.

A mis amigos y compañeros ajedrecistas de San Luis Potosí, por mantener en mí el sano espíritu de competencia, deportividad y constancia. Gracias totales. *Gens una summus* (Somos una familia)

Resumen

Entre las áreas de investigación de la visión computacional se encuentran las aplicaciones para el control basado en visión de manipuladores móviles. Para la mayoría de los sistemas de control de estos robots, se utilizan comúnmente computadoras estándar, con poderosas herramientas de software, haciendo que los procesos para la implementación de la tarea sean fácilmente desarrollados. Tales sistemas mantienen su compatibilidad con versiones anteriores, de esta manera el software puede ser reimplementado tan pronto como una nueva y más rápida plataforma está disponible en el mercado. Desafortunadamente una solución basada en una PC estándar no es viable en la gran mayoría de los casos para el procesamiento en tiempo real. En los años recientes, han sido desarrolladas plataformas especiales de hardware para satisfacer los requerimientos de procesamiento en tiempo real, con lo cual se ha podido obtener soluciones más flexibles y asequibles basándose en dispositivos programables, principalmente en Procesadores Digitales de Señales (DSP) y Arreglos de Compuertas Programables en Campo (FPGA). La tasa de desempeño/costo para tales dispositivos se ha incrementado convirtiéndolos en una alternativa valida para el diseño de soluciones en tiempo real al nivel de plataforma. Este trabajo de tesis presenta, un análisis de los dos tipos de dispositivos programables mencionados, la selección de uno de ellos, la selección de un fabricante y modelo específico de dispositivo y un esquema de programación que implementa la etapa de procesamiento digital de imágenes en la plataforma TMS320DM637 de Texas Instruments (TI),que fue la que resultó seleccionada, como un primer paso en la etapa de construcción de un robot móvil con una velocidad de desempeño mucho mas eficiente.

Capítulo 1

Introducción

1.1. Motivación

El control basado en visión ha sido utilizado en varios campos de la ingeniería. Uno de ellos es el relacionado con la robótica móvil, donde uno de sus objetivos principales es dotar de autonomía a los robots usando cámaras, ya que la necesidad de que el operador humano tenga un papel dominante en el control y la manipulación de estos robots ha limitado el uso de esta clase de dispositivos a aplicaciones en las que el nivel de desempeño requerido es relativamente bajo. Una de las etapas fundamentales en este tipo de control reside en la extracción de la información de la imagen; la información extraída, servirá para la construcción de un modelo que llevará a una interpretación y toma de desiciones basadas en el algoritmo de control implementado. En este sentido el procesamiento digital de la imagen se convierte en un factor que impacta decisivamente en la dinámica del sistema. Dada la gran cantidad de información que se tiene que procesar, este tipo de sistemas presentan una velocidad de respuesta lenta. Para un espacio de trabajo fijo y delimitado a una área previamente definida, el impacto de una respuesta lenta no es crítica, pero para un robot móvil, en el cual su espacio de trabajo cambia conforme se desplaza en él, una respuesta lenta limita signifcativamente al robot en el desempeño de su tarea,

de tal manera que la necesidad de disponer de un procesamiento en tiempo real es alta para mejorar substancialmente la respuesta del sistema móvil. La mayoría de los sistemas de procesamiento de imágenes (y de control de robots) basan su implementación en una computadora personal; sin embargo en muchas ocasiones estos equipos no brindan el requerimiento de alta velocidad de procesamiento en tiempo real. En este punto, dado el gran desarrollo de la electrónica, especialmente en el área de Procesadores Digitales de Señales, este trabajo de tesis propone el uso de esta plataforma para implementar los algoritmos de visión computacional.

1.2. Objetivos de la tesis

El objetivo principal del trabajo de tesis es:

- Implementar algoritmos de Visión Computacional en una plataforma DSP que por sus características brinde gran capacidad de procesamiento de la información.

Los objetivos particulares están definidos para poder alcanzar el objetivo principal de este trabajo, y son:

- Evaluación de la arquitectura de procesamiento, debido a que las plataformas más poderosas para el procesamiento digital se dividen en dos arquitecturas principales, procesadores digitales de señales (DSP) y dispositivos de arreglos de compuertas de campo programable (FPGA), se hace necesario una evaluación para determinar cual de estas dos arquitecturas es la más adecuada para la implementación de los algoritmos de Visión.
- Definición de la plataforma a trabajar una vez definida la arquitectura. Existen varios fabricantes de la arquitectura seleccionada, cuyas plataformas reúnen varias características distintas entre sí, por lo tanto es necesario definir cual de

todas reúne los requerimientos necesarios tanto en hardware como en software para poder cumplir con el objetivo principal de la tesis.

- Análisis de los diferentes tipos de algoritmos de procesamiento digital de imágenes, ya que existen diferentes tipos de algoritmos es necesario determinar cuales son los adecuados en función la complejidad de su implementación y su velocidad de procesamiento.

- Portabilidad del algoritmo implementado, debido a que constantemente se presentan avances en la investigación de nuevos esquemas de control o desarrollo de hardware, es necesario que la implementación sea flexible y portable para que pueda ser reconfigurada y adaptada a plataformas de procesamiento más modernas.

- Desarrollo de una comparativa de desempeño de ejecución de algunos algoritmos de procesamiento digital de imágenes en una plataforma DSP y una PC.

1.3. Estado del arte

La implementación de los algoritmos de procesamiento digital hace su aparición en el campo computacional tardíamente, debido a que primeramente se hubo de desarrollar los esquemas de software y hardware que permitieran llevarlo a cabo. En cuanto al software, los esquemas de procesamiento digital de imágenes son muy elaborados y complejos, en la actualidad existen muchas aplicaciones de software que permiten dicho procesamiento y utilizan técnicas y algoritmos que son bien conocidos por la comunidad que trabaja en ellos, aunque existen variantes ó técnicas nuevas que están poco documentadas. Por el lado del hardware, el rápido desarrollo de la tecnología de circuitos integrados ha estimulado el desarrollo de plataformas de procesamiento potentes, pequeños, baratos y de propósito general. Estas plataformas han permitido construir sistemas digitales altamente sofisticados, capaces de

realizar funciones y tareas de procesado que anteriormente eran difíciles o caras con circuitería y sistemas de procesado de señales analógicas.

En el caso de la presente propuesta de tesis, el área que se va a trabajar, tiene sus antecedentes desde 1993 con la implementación de algoritmos iterativos de recuperación y captura de imágenes en máquinas DSP en paralelo en arreglos de 8x8 bits y el sistema AT&T pixel[1]. A partir de ahí la literatura recoge cientos de reportes de trabajos que incorporan de manera directa o indirecta la utilización de procesadores digitales de señales. Las diferente aplicaciones van desde el ámbito industrial con un procesado de visión artificial para el control automático de impresión de tintas implementado en un FPGA de Xilinx [2] y visión en 3D para el control autónomo de soldaduras usando el sistema UNSHADES de Xilinx [3], pasando por aplicaciones como la interacción hombre-máquina usando gestos manuales en tiempo real desarrollada en la tarjeta ADSP BF-533 EzKit Lite de Analog Devices que está basada en el análisis multiresolución a través de las ondeletas de Haar[4] , hasta aplicaciones de cámara CMOS como sensor en FPGA [5] y en el DSP SHARC de Analog Devices [6]. En la literatura existente, se han presentado diversas estructuras especializadas de cálculo para control de robots móviles [7, 8, 9, 10] las cuales utilizan dispositivos configurables tales como DSP y FPGA, en diferentes configuraciones. La utilización de un conjunto de procesadores para alcanzar velocidades importantes de procesamiento, requiere el desarrollo de protocolos de comunicación y sincronización complejos. El diseño de la plataforma que se presenta en este trabajo de tesis permite implementar todo el sistema de procesamiento de imagen. Este sistema DSP compuesto de varios subsistemas es tecnología de última generación en procesadores de punto fijo y de la cual hay pocos trabajos reportados en la literatura especializada [11] y en la cual no fue posible encontrar trabajos reportados en el área de robótica móvil.

1.4. Estructura de la tesis

El capítulo dos presenta un descripción de las dos principales arquitecturas para el procesamiento digital, se mencionan a los principales fabricantes de dichas arquitectura y sus diferentes familias de dispositivos, después se realiza un análisis de las diferentes arquitecturas de plataformas de procesamiento digital, el análisis se basa en dos puntos, la facilidad en el diseño de la aplicación en software y potencia en hardware. Una vez seleccionada la arquitectura, se realiza un comparativo entre los diferentes fabricantes de dicha arquitectura y se selecciona uno de ellos. Una vez seleccionado el fabricante se realiza una comparativa entre sus diversos dispositivos y se selecciona el dispositivo final para la implementación de los algoritmos de procesamiento.

El capítulo tres describe brevemente el Método de Manipulación en Espacio de Cámara (CSM) y establece las técnicas de procesamiento digital de imágenes asociadas a él. También se realiza una comparativa entra las diferentes técnicas de extracción de información de una imagen y se seleccionan las más adecuadas para ser llevadas acabo en el DSP, la selección se realiza en base a dos puntos principales, su bajo coste computacional y su velocidad de procesamiento.

El capítulo cuatro describe la arquitectura en hardware y software del dispositivo DSP seleccionado en el capítulo dos, también la implementación de los algoritmos de visión y sus resultados experimentales, para terminar el capítulo cinco que contiene las conclusiones y el trabajo a futuro.

Capítulo 2

Análisis y Selección de la plataforma de Procesamiento Digital de Señales

Como se mencionó anteriormente, las imágenes en tiempo real y el procesamiento digital de señales tienen un papel cada vez más importante en la actualidad en muchas áreas. En el área del control basado en visión de robots móviles, donde una solución en tiempo real es requerida, se imponen limitantes considerables en cuanto a tamaño, disipación de potencia y costo de la misma. Una limitante adicional que se presenta continuamente es la flexibilidad de la solución, ésta debe permitir que se reconfigure conforme se avance en la investigación de nuevos esquemas de control o desarrollo de hardware. La necesidad de implementar un esquema reconfigurable se opone al uso de esquemas de hardware dedicados dejando la posible solución a esquemas basados en microprocesadores de propósito general, procesadores digitales de señales (DSP) o dispositivos de arreglos de compuertas de campo programable (FPGA). Los microprocesadores de alto desempeño tienden a disipar mucha energía como para ser considerados soluciones de baja potencia, y generalmente, requieren de complicadas interfaces para comunicarse con el resto del sistema. Los DSP y FPGA, sin embargo, disipan menos potencia manteniendo un alto desempeño de procesamiento y poseen interfaces que permiten conectarlos directamente con otros

dispositivos del sistema.

En este capítulo, se presenta un análisis de las plataformas FPGA y DSP en base a sus características, el análisis de las diferentes fabricantes y familias de procesadores digitales de señales y se menciona la plataforma seleccionada.

2.1. Arreglos de Compuertas de Campo Programable (FPGA)

2.1.1. Antecedentes

Los primeros dispositivos programables de arreglos de compuertas aparecieron en los primeros años de la década de los ochenta , y fueron llamados Arreglos Lógicos Programables (PAL por sus siglas en inglés), los cuales facilitaban una manera de implementar la lógica de programación de los antiguos diseños. Esto reducía significativamente el número de partes distintas y necesarias para completar un diseño y permitía darle un mantenimiento más fácil que a un sistema más complejo.

La siguiente generación de dispositivos combinaba tantas PAL, como fuera posible, dentro de dispositivos más sencillos llamados Dispositivos Lógicos Programables (PLD por sus siglas en inglés), Dispositivos Lógicos Programables Borrables (EPLD) o Dispositivos Lógicos Programables Complejos (CPLD en inglés) siendo estos últimos, el antecedente de los FPGA. Funciones más complejas pueden ser ahora implementadas en un chip programable sencillo, lo que los hace candidatos a sustituir ASIC o circuitos impresos.

Los FPGA fueron inventados en el año de 1984 por Ross Freeman, co-fundador de Xilinx, y surgen como una evolución de los CPLD. La idea fue combinar la densidad y versatilidad de los arreglos de compuertas con la rápida rotación y las conveniencias de los dispositivos programables. Así, los FPGA dan al desarrollador

las ventajas de altos niveles de integración sin los gastos de desarrollo personalizado que se presenta en los ASIC.

Los FPGA son dispositivos basados en la arquitectura de las PROM de campo programable (EPROM y EEPROM), de ahí su nombre.

2.1.2. Arquitectura

Los dispositivos FPGA [12] tienen un arreglo de compuertas con una matriz de células lógicas conectadas por los llamados recursos de interconexión y rodeadas con células de entrada/salida como se muestra en la figura 2.1. Esta configuración puede llegar a ser muy diferente de un fabricante a otro, pero el concepto general es el mismo. En el centro está un arreglo de compuertas lógico. Cada célula está compuesta de flip-flops y lógica Booleana. Las células lógicas son conectadas por interconexiones programables, las cuales realizan el trazado de la ruta de las señales entre las células. Algunos de los dispositivos para el trazado de rutas contienen buffers de tres estados, que permiten buses internos bidireccionales reduciendo el número de interconexiones. Los bloques de entrada/salida poseen buffers de tres estados con suficiente potencia para manejar señales, ya sea hacia adentro o hacia afuera del FPGA. Estos bloques también contienen flip-flops para sincronizar las señales entrantes y retenedores (latchs) para para las señales de salida.

Actualmente hay 4 diferentes categorías de FPGA disponibles comercialmente: *arreglos simétricos, arreglos basados en filas, jerarquía de PLD* y *mar de compuertas.* En todas esas categorías las interconexiones y la manera de programarlos varía. También existen cuatro tecnologías en uso actualmente: *Células RAM estáticas, Anti-fuse, EPROM y EEPROM de transitores.* Dependiendo de la aplicación, cada tecnología FPGA puede tener características deseables para la misma.

La tabla 2.1 muestra la relación entre las arquitecturas, el tipo de block lógico del que están compuestas y la tecnología en la que están basadas, como puede verse,

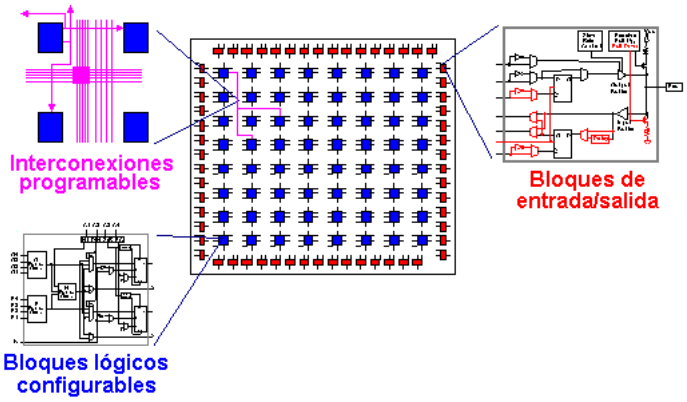

Figura 2.1: Arquitectura de un FPGA

un mismo block lógico puede ser usado en diversas tecnologías y arquitecturas de FPGA.

2.1.3. Herramientas de Diseño

Existen varias compañías que ofrecen herramientas de diseño para los FPGA. La fig.2.2 muestra una típica herramienta de diseño basada en interfaces gráficas de usuario, que permite ver los árboles de directorios de los diferentes archivos que maneja el proyecto. Cuenta además, con una pantalla de edición donde se escribe el código del programa, y diferentes ventanas que permiten ver las variables que está procesando un FPGA. Estas herramientas permiten diseñar diferentes tipos de aplicaciones para diferentes tipos de fabricantes de FPGA. También dan libertad al diseñador para completar el diseño antes de que sea implementado en el hardware. Así como, permitir que una vez que la aplicación está implementada en hardware, mejorarla y actualizarla sin hacer un rediseño extensivo. Dependiendo de la complejidad del diseño, muchos de éstos pueden existir independientemente del hardware y, en algunos casos, pueden ser sintetizados para diferentes dispositivos.

La más común de las herramientas usadas, hoy en día, son el Verilog y el VHDL. Verilog es más comúnmente usado para aplicaciones comerciales, y es similar, en

Tabla 2.1: Arquitecturas, bloques lógicos y tecnologías de los FPGAS

Arquitectura	**Block Lógico**	**Tecnología**
Filas	Multiplexor	anti-fuse
Mar de compuertas	Multiplexor y Compuertas básicas	RAM estática
Jerarquía PLD	PLD	EPROM
Arreglo simétrico	Multiplexor	anti-fuse
Arreglo simétrico	Look-up table	RAM estática

estructura y sintaxis, a los lenguajes C y C++. VHDL son las siglas de Lenguaje de Descripción de Hardware para VHSIC, y VHSIC a su vez es una abreviación de Circuito Integrado de Muy Alta Velocidad, por sus siglas en inglés. El VHDL era hasta hace algún tiempo la herramienta más comúnmente usada para aplicaciones militares y gubernamentales y es similar en estructura y sintaxis al lenguaje de programación ADA. Tanto el Verilog como el ADA son portables entre varios compiladores y herramientas de simulación, lo cual permite hasta cierto punto que el diseño o la aplicación siga vigente, aún cuando las herramientas que se usaron para crearla sean actualizadas o cambiadas.

Muchas características están disponibles en las últimas herramientas de diseño para tomar ventaja del gran volumen de recursos disponibles en el FPGA. Algunos fabricantes proveen herramientas para generar máquinas de estado, donde los diagramas de estado son convertidos a Lenguaje de Descripción de Hardware (HDL, por sus siglas en inglés). Este tipo de herramientas son muy valiosas en la creación de diseños muy complejos, ya que permite que sean aplicaciones autodocumentadas. Así mismo, el código generado puede ser unido a otro código en HDL para terminar

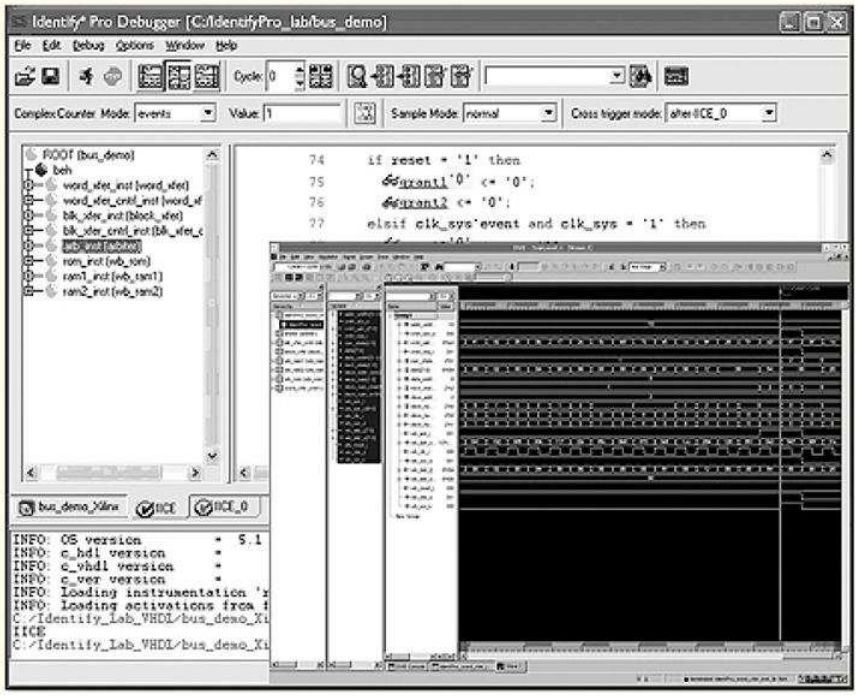

Figura 2.2: Ejemplo de herramienta de diseño.

de implementar el diseño completo.

2.1.4. Fabricantes y familias de FPGA

En los últimos años, el mercado de los FPGA se ha colocado en un estado donde hay dos productores de FPGA de propósito general que están a la cabeza del mismo, y un conjunto de otros competidores quienes se diferencian por ofrecer dispositivos de capacidades únicas. A continuación, se mencionan las principales fabricantes de FPGA y algunas de las características que aportan sus productos al mercado. La tabla 2.2 muestra las principales familias de dispositivos que los fabricantes producen actualmente.

Xilinx es uno de los dos grandes líderes en la fabricación de FPGA [13] . Altera es el otro gran líder [14]. Lattice Semiconductor proveedor líder en tecnología no

volátil, lanzó al mercado dispositivos FPGA con tecnología de 90nm basados en tecnología Flash, además de productos de 130nm. Actel tiene FPGA basados en tecnología Flash reprogrammable. También ofrece FPGA que incluyen mezcladores de señales basados en Flash. QuickLogic tiene productos basados en antifusibles (programables una sola vez). Atmel es uno de los fabricantes cuyos productos son reconfigurables (el Xilinx XC62xx fue uno de éstos, pero no están siendo fabricados actualmente). Ellos se enfocaron en proveer microcontroladores AVR con FPGA, todo en el mismo encapsulado. Achronix Semiconductor tienen en desarrollo FPGA muy veloces. MathStar, Inc. ofrecen FPGA que ellos llaman FPOA (Arreglo de objetos de matriz programable).

2.2. Procesadores Digitales de Señales

2.2.1. Antecedentes

Un procesador digital de señales es un microprocesador especializado y diseñado específicamente para procesar señales digitales en tiempo real. En su núcleo, un DSP es altamente numérico y repetitivo. A la vez que cada dato llega, éste debe ser multiplicado, sumado y, además, transformado de acuerdo a fórmulas complejas. Lo que permite realizar todo ello es la velocidad del dispositivo [15].

En 1978, INTEL lanzó el 2920 como un "procesador analógico de señales". Éste poseía un chip ADC/DAC con un procesador de señales interno, pero no poseía un multiplicador de hardware. El 2920 no tuvo éxito en el mercado. En 1979, AMI lanza el S2811, el cual fue diseñado como un microprocesador periférico; al igual que el 2920, no tuvo gran éxito en el mercado. En el mismo año, BELL LABS introduce el primer chip procesador digital de señales (DSP), The Mac 4 Microprocessor. Luego, en 1980 fueron presentados en el ISSCCs'80 los primeros DSP completos: el PD7710 de NEC y el DSP1 de AT&T; ambos procesadores fueron inspirados en las investigaciones de PSTN Telecomunicaciones. En ese mismo año, NEC comenzó la

producción del PD7710, la primera producción de DSP completos en el mundo. El primer DSP producido por TEXAS INSTRUMENTS, el TMS32010, probó ser un suceso mayor.

2.2.2. Arquitectura

Una de las características más importantes de un DSP es su capacidad de realizar operaciones de multiplicación y acumulación (MAC) en sólo un ciclo de reloj. No obstante, es necesario que el dispositivo posea la característica de manejar aplicaciones críticas en tiempo real. Esto requiere de una arquitectura que soporte un flujo de datos a alta velocidad hacia y desde la unidad de cálculo y memoria. Esta ejecución a menudo requiere el uso de unidades DMA (Direct Memory Acess) y generadores de direcciones duales (DAG) que operan en paralelo con otras partes del chip.

Los DGA realizan los cálculos de direcciones, permitiendo al DSP buscar dos datos distintos para operar con ellos en un sólo ciclo de reloj, de tal forma que es posible ejecutar algoritmos complejos en tiempo real.

Es importante para DSP tener un mecanismo efectivo de salto para la ejecución de lazos de programación ("loops"), ya que el código generalmente programado es altamente repetitivo. La arquitectura permite realizar estos "loops" sin instrucciones adicionales ni demoras, las que al ejecutarse millones de veces pueden generar retardos significativos.

Las arquitecturas de los computadores actuales están comúnmente clasificadas como RISC (Reduced Instruction Set Computers) y CISC (Complex Instruction Set Computers). Estos últimos tienen un gran número de instrucciones sumamente poderosas, mientras que la arquitectura RISC posee pocas instrucciones y realiza movimientos de datos entre registros en un ciclo de máquina. Como puede observarse en la fig 2.3 la arquitectura CISC agrupa una gran cantidad de componentes en menos bloques, con el bus de direcciones y de datos también agrupados en un

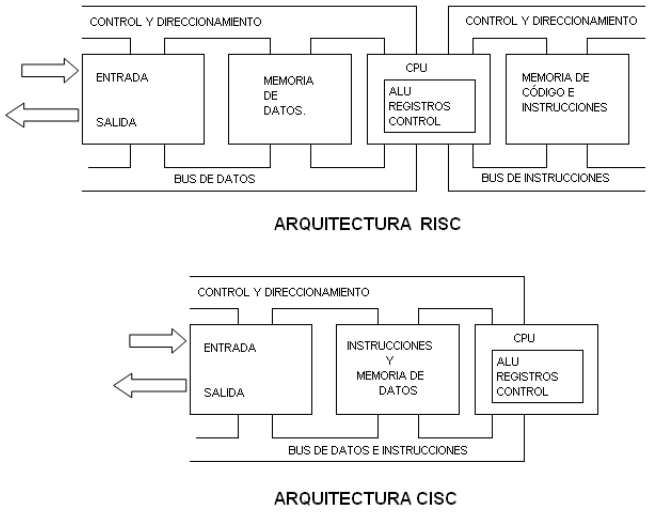

Figura 2.3: Principales Arquitecturas de DSP

sólo canal. Sin embargo los computadores RISC alcanzan un rendimiento más alto por medio del uso de un eficiente compilador como a través de la ejecución de instrucciones simples en forma ordenada.

Los DSP estándares tienen mucho rasgos de una arquitectura tipo RISC, pero son procesadores de propósitos específicos cuya arquitectura es especialmente diseñada para operar en ambientes de alta necesidad de cálculo. Un DSP estándar ejecuta varias operaciones en paralelo mientras que un RISC usa unidades funcionales altamente eficientes que pueden iniciar y completar una instrucción simple en uno o dos ciclos de reloj.

2.2.3. Herramientas de diseño

Los principales fabricantes de DSP ofrecen sus propias herramientas de diseño llamadas Entornos de Desarrollo Integrados (IDEs por sus siglas en inglés). Estas herramientas como en el caso de los FPGA permiten diseñar diferentes tipos de aplicaciones. Un IDE consiste normalmente de código fuente, un compilador, her-

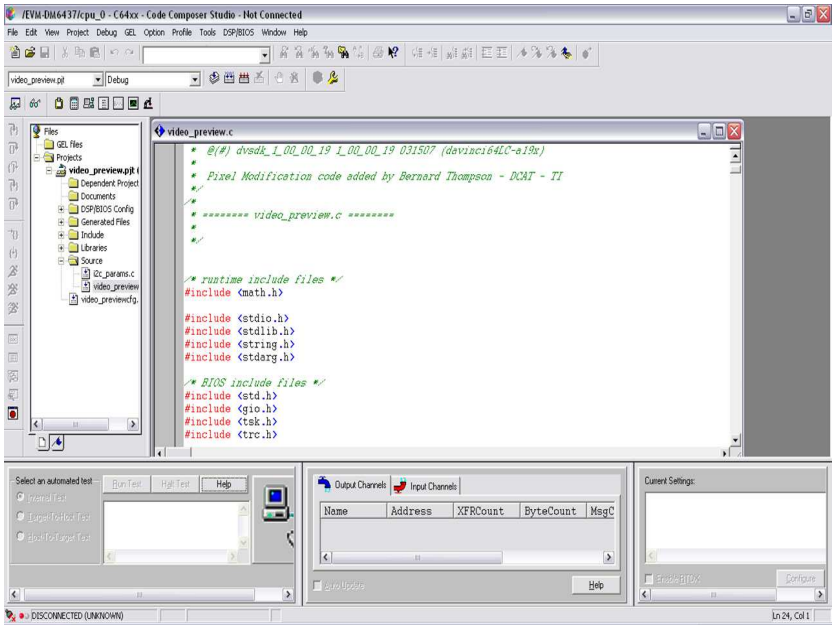

Figura 2.4: Entorno de Desarrollo Integrado para los DSP

ramientas de construcción automática y un depurador como se puede apreciar en la fig 2.4.

Están diseñadas para maximizar la productividad del programador facilitándole varios componentes en una interface gráfica de usuario y están dedicadas a un lenguaje específico, el cual en la mayoría de los casos es C/C++, afín de proporcionar un conjunto de características que más se ajusten a los paradigmas de programación del lenguaje. Los IDEs suelen presentarse como un programa único en el que todo el desarrollo se realice. Típicamente ofrece muchas características para modificar, compilar,depurar y desplegar el software. El objetivo es abstraer las configuraciones necesarias para unir todas las utilidades de la línea de comandos en una unidad coherente y que en teoría reduce el tiempo para aprender el lenguaje de programación e incrementa, como ya se dijo, la productividad del desarrollador. Por ejemplo, el código puede ser compilado mientras se escribe, proporcionando información instantánea acerca de los errores de sintaxis.

Todo lo anterior, más el uso del lenguaje C/C++ hace que las aplicaciones sean

portables entre varias plataformas en la mayoría de los casos, lo cual permite una vigencia del diseño a pesar de las actualizaciones del hardware donde fue implementado.

2.2.4. Fabricantes y familias de DSP

Existen diversos fabricantes, cada uno con un tipo especial y particular de arquitectura, uso y/o aplicación. A continuación se mencionan los principales fabricantes de DSP y algunas de las características que aportan sus productos al mercado. La tabla 2.3 muestra las principales familias de dispositivos que los fabricantes producen actualmente.

Texas Instruments es la compañía que mantiene el liderazgo en el área de procesadores digitales de señales con la serie TMS320C, y es el tercer fabricante de semiconductores en el mundo después de Intel y Samsung [16] . Como se puede ver en la tabla 2.3 Texas Instruments mantiene 3 principales familias, que van de la menos compleja con número 2000 hasta la más compleja con número 6000. Motorola, es el otro gran fabricante de microprocesadores y procesadores digitales de señales, debido a la alta especialización y por motivos económicos creó la empresa Freescale Semiconductor en el 2004. En los últimos años ha incorporado la tecnología Star-Core con dos o tres núcleos en un DSP. La tabla 2.3 muestra, además, que ha introducido arquitecturas de 16 y 24 bits [17]. Analog Devices es el otro competidor en el mercado de los DSP's, actualmente, apuesta por la la integración de componentes a gran escala en sus procesadores [18], y es capaz de fabricar dispositivos semiconductores muy pequeños en el orden de los 0.09-3 micrómetros.

2.3. Comparativa FPGA-DSP

Como se expuso en los temas anteriores, las arquitecturas y herramientas de diseño entre un dispositivo FPGA y un dispositivo DSP varían considerablemente. Por lo tanto, es difícil establecer una comparación directa en ese sentido. La premisa básica de comparación serán las ventajas que el hardware aporta a las aplicaciones y el esfuerzo de programación necesario para llevarlo a cabo; todo esto, en el marco de los objetivos de esta tesis.

2.3.1. Tendencias en la arquitectura de los DSP

Los DSP son ampliamente utilizados en la solución de diseños que implican el trabajo con señales digitales, y aunque son programables a través de software su arquitectura de hardware no es flexible. Por lo tanto, se encuentran limitados por su arquitectura fija, que por ejemplo, crea problemas en el desempeño de sus buses de datos cuando estos tienen que soportar el manejo de una gran cantidad de datos. Otras limitantes inherentes al problema del hardware son el número fijo de bloques de acumulación y multiplicación (MAC) y de bloques aceleradores de hardware. Estas características no lo hacen adecuado para ciertas aplicaciones que requieran de un diseño y configuración de hardware específicas.

Por otra parte, los FPGA proveen una solución reconfigurable para implementar aplicaciones de procesamiento digital de señales que pretende dar un mayor rendimiento y potencia en el procesamiento de datos en bruto, que un DSP. Desde que los FPGA son configurables en su hardware, pueden ofrecer configuraciones completas y personalizables mientras se implementan varios diseños. Por lo tanto, los diseños para el procesamiento digital implementados en un FPGA pueden tener características particulares en su arquitectura, en su estructura de buses de datos, tamaño de memoria, bloques de aceleración de hardware y un número variable de bloques MAC.

A pesar de estos beneficios, una de las mayores razones por la cuales los FPGA no han encontrado una amplia aceptación en el mercado es la poca posibilidad de implementar un diseño basado en lenguaje de nivel intermedio como el lenguaje C y que no requiera conocimientos previos de la arquitectura del FPGA, ni del lenguaje de descripción de hardware (HDL). Históricamente los programadores acostumbrados a diseños y aplicaciones de procesamiento digital basados en software se enfrentan a una barrera de diseño al cambiar a una solución basada en FPGA, lo cual no es deseable.

2.3.2. Diseño de software

La figura 2.5, muestra el típico diseño de software que usan los programdores de DSP. Los programadores usan herramientas de diseño de algoritmos tales como MATLAB para optimizarlos y SIMULINK para el modelado a nivel del sistema. Después los algoritmos y el modelo se implementan en C/C++ o en lenguaje Ensamblador, mediante un ambiente de desarrollo integrado estándar como el Code Composer Studio de Texas Instruments, que proporciona diseño, simulación, depuración y herramientas de verificación en tiempo real. Además de que pueden usar librerías basadas en ANSI C para acortar los ciclos de diseño y obtener los beneficios de la reutilización.

Por otro lado, como se muestra en la fig 2.6, el diseño de software empleado para la implentación de soluciones en FPGA conlleva más pasos que para un DSP, el desarrollo de un algoritmo y el modelo de sistema aunque puede hacerse con herramientas de diseño de algoritmos la implementación se realiza siempre en HDL. Una vez que se tiene el modelo, la optimización del hardware casi siempre requiere el uso de Funciones de Aceleración, para lo cual se requiere un amplio conocimiento de la arquitectura del FPGA, una vez diseñadas la Funciones de Aceleración se integran al modelo del Sistema en la etapa que se denomina Construcción y se realiza igual con HDL, las funciones de simulación y depuración están disponibles

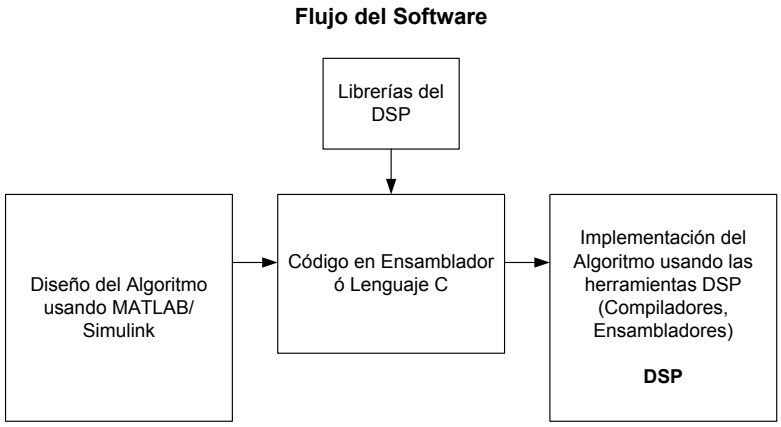

Figura 2.5: Diseño de software en DSP

para los FPGA a través de Librerías de software. La construcción de estas librerías constituyen, también, una tarea importante en vista de la mejora de la aplicación desarrollada. Una vez realizados estos pasos, la implementación es cargada en el FPGA.

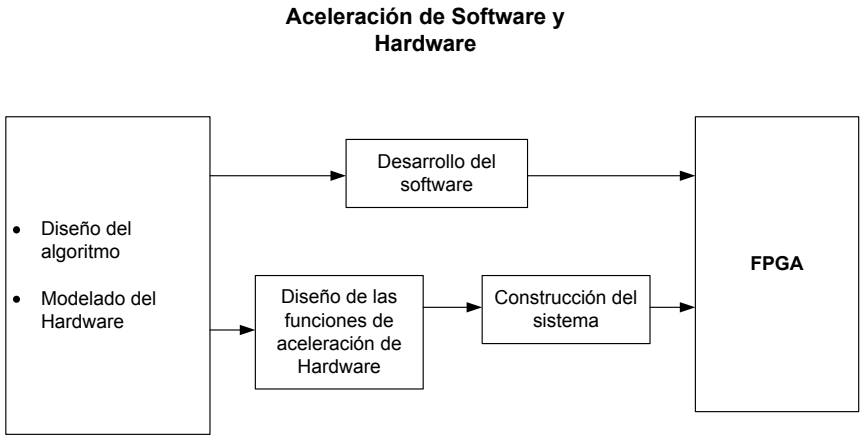

Figura 2.6: Diseño de software en FPGA

De los puntos anteriores, se pueden exponer los siguientes argumentos a favor para seleccionar un Procesador Digital de Señales:

1. Con respecto al hardware, las arquitecturas de hardware que tienen los DSP's, en la actualidad, están suficientemente especializadas para soportar en velocidad de procesamiento, los algoritmos que se implementan en ellas como parte de esta tesis. Es decir, no se necesita diseñar una arquitectura especial para el desarrollo de los algoritmos, como tampoco implementar algunas de las ya conocidas. Esto es un esfuerzo de diseño que siempre estará presente cuando se trabaja con un FPGA.

2. Con respecto al software, se requiere un conocimiento detallado de la arquitectura del FPGA y de un lenguaje de programación especial, para poder desarrollar un proyecto o aplicación. Como puede inferirse, el uso de Funciones Aceleradoras de Hardware conlleva el diseño de arquitecturas paralelas consumiendo así más tiempo que el que podría destinarse a la programación del DSP. Para lograr el objetivo de que la aplicación o proyecto en un FPGA pueda llevarse de una plataforma a otra, se requiere de la implementación de lo llamados buffers lineales, que tienen como principal característica que pueden ser reprogramados o actualizados sin restricción de tamaño y direccionamiento de la memoria, como puede inferirse, la implementación, o en casos particulares, el diseño de los buffers lineales, representa un esfuerzo de tiempo y programación agregado.

Dados los puntos anteriores, se propone el uso de la plataforma DSP porque el proceso de desarrollo e implementación es más rápido, así como las características de su arquitectura y programación permiten cumplir con los objetivos de portabilidad establecidos en este trabajo de tesis.

2.4. Selección de la Plataforma

Después de analizar las características de las diferentes familias, los DSP de Texas Instrument surgen con ciertas ventajas sobre los otros.

Una de las principales ventajas del fabricante mencionado es que a través de su desarrollo ha acumulado un gran nivel de conocimiento, sus procesadores se han utilizado para resolver problemas reales y han dado resultados positivos. Reúne, en su página web, documentos y aplicaciones de utilidad para quienes trabajan en nuevos desarrollos y/o para quienes recién empiezan a trabajar con microprocesadores. Por otra parte, los precios de los procesadores permiten acceder a tecnologías de última generación a un costo aceptable, además de que el posgrado donde se desarrolló el presente trabajo de tesis ha obtenido experiencia en el manejo de los mismos, con resultados satisfactorios en otros trabajos de investigación.

Texas Instruments ofrece kits y tarjetas de desarrollo divididos en tres grupos [20]:

Starter Kits. DSP Starter Kits ofrecen paquetes de evaluación completos y de bajo costo que incluyen herramientas de desarrollo y hardware.

Módulos de Evaluación (EVMs por sus siglas en inglés). Con un amplio rango de tarjetas desarrolladas por TI y terceros.

Plataformas de Desarrollo para aplicaciones específicas. Soportan aplicaciones de video e imagen, audio digital, telefonía, seguridad, control, etc. Estas plataformas combinan el uso de un software y hardware específico en un ambiente de fácil desarrollo para la creación de prototipos.

Como complemento a estos kits de desarrollo TI ofrece las *Daughter Cards,* que son módulos de hardware que se conectan las tarjetas de desarrollo. Texas Instruments maneja 171 plataformas de DSP divididas en 6 grandes grupos. Se han elegido las plataformas de la familia C6000 y DM por ser las de más alto rendimiento en el manejo de los datos [21]. La tabla 2.4 muestra una tabla comparativa entre las plataformas de desarrollo.

Se seleccionó y evaluó un representante de cada familia con base en tres criterios: La velocidad de procesamiento, la cantidad de memoria externa que puede direc-

cionar y que está incluida en la misma plataforma y la conectividad a dispositivos externos. Las plataformas seleccionadas y con sus respectivas características fueron:

DM6437 Plataforma para Desarrollo de Video Digital. Provee a los desarrolladores hardware y software para una rápida implementación de un amplio rango de aplicaciones basadas en video digital como sistemas de visión de máquina, robótica, seguridad de video y video telefonía. Características: Procesador basado en la tecnología Da Vinci, conectividad múltiple a la PC a través dePCI, Ethernet, UART, CAN. Entrada y salida de video NTSC/PAL y YPbPr/RGB, Códecs para imagen, video y audio. Es por ello que se considera la mejor plataforma a bajo costo para la implementación de imagen y video digital .

TMS320C6416 DSP Starter Kit (DSK). Es una plataforma de bajo costo para un rápido desarrollo de aplicaciones de alto desempeño, y que requieren de un intensivo manejo de memoria tales como video, imagen y sistemas de multicanales. Características. Dentro de sus ventajas se encuentran la alta velocidad del procesador TMS320C6416 a 1 GHz capaz de desarrollar 8000 MIPs. Un ambiente de desarrollo integrado (IDE por sus siglas en inglés). Compilador y ensamblador para C/C++, software Code Composer y herramientas de diagnóstico.

Módulo de Evaluación C6424. Es una solución a bajo costo que permite a los desarrolladores evaluar e implementar rápidamente sistemas prototipo basados en los poderosos procesadores C642x. Características. Múltiples tipos de memoria: compact Flash, ATA, SD, DDR. Conexión a la computadora a través de PCI, Ethernet y USB. Ambiente de desarrollo integrado.

Como criterio de decisión para la plataforma se consideraron 3 factores: la velocidad de procesamiento, la memoria RAM y el manejo de periféricos. Del análisis de estos factores se seleccionó la plataforma DM6437, basados en las siguientes ventajas: procesador con la nueva tecnología Da Vinci, bajo costo, además de que cuenta con casi todo el hardware necesario para la aplicación y desarrollo de sistemas de video e imagen digital.

En este capítulo se presentaron las principales características de los dispositivos FPGA y DSP, tanto en el área de arquitectura de hardware como en su arquitectura de software. Se establecieron los criterios para la selección de la plataforma más adecuada para el presente trabajo de tesis, así como la plataforma seleccionada.

Por su especialización y características propias de la arquitectura, y por su diseño lineal en el desarrollo del software, los Procesadores Digitales de Señales se convierte en la plataforma elegida. De los fabricantes de DSP, se escogió la marca Texas Instruments por proporcionar una amplia variedad de dispositivos especializados en el procesamiento digital de imagen y video, además de proporcionar soporte e información para el desarrollo de la implementación.

De todas los dispositivos de Texas Instruments se optó por la plataforma DM6437 de tecnología DaVinci, dado que su precio es accesible y es una plataforma especializada que permite trabajar con lenguajes y diseños de nivel alto e intermedio para la implementación de los algoritmos de visión que se presentan en el siguiente capítulo, además de que, como se expone en el capítulo cuatro, su arquitectura de software y hardware nos permite cumplir con el objetivo de portabilidad del proyecto.

Tabla 2.2: Fabricantes y familias de FPGA

Fabricantes	**Familias**
Xilinix	Glue Logic Bajo Coste Alto Rendimiento
Altera	Alto Rendimiento Bajo Costo
Lattice Semiconductor	Non-Volatile High Value System Level
Actel	Low-Power IGLOO ProASIC3 Mixed-Signal Fusion
Quick Logic	ArticLink ArticLink II PolarPro II
Atmel	AT40KAL (5v) AT40KAL (3v)
Achronix	Speedster
MathStar	MDOA2440D

Tabla 2.3: Fabricantes y Familias de DSP.

Fabricantes.	**Familias.**
Texas Instruments	TMS320C6000
	TMS320C5000
	TMS320C2000
Freescale(Motorola)	StarCore 16 Bits.
	Symphony DSP56x 24bits.
	DSP563xx 24 bits.
	56800 16 bits.
Analog Devices	Black Fin.
	SHARC.
	Tiger SHARC.
	ADSP-21xx.

Tabla 2.4: Plataforma de Desarrollo de Texas Instruments.

	Starter Kits	**Módulos de Evaluación (EVMs)**	**Plataformas de desarrollo para aplicaciones específicas**	**Tarjetas hijas**
Plataformas TMS320 DSP				
Procesadores DaVinci		*	*	*
TMS320DM644x DSP		*	*	
TMS320DM643x DSP		*	*	
TMS320DM64x DSP		*	*	*
DSP de Alto desempeño C6000	*	*	*	
TMS320C645x DSP	*	*		
TMS320C6414T/15T/16T DSP	*		*	
DSP Performance Value C6000		*	*	*
TMS320C642x DSP		*		
TMS320C6410/12/13/18 DSP		*	*	*
TMS320C62x DSP		*	*	*

Capítulo 3

Técnicas de procesamiento digital de imágenes aplicadas al Método de Manipulación en Espacio de Cámara

Los sistemas robóticos pueden ser divididos en dos partes, el hardware y el software. El hardware hace referencia al cuerpo físico del robot, es decir, las partes mecánicas (motores, eslabones, ruedas, pinzas, etc) y dispositivos de sensado que permiten al robot interactuar con el medio que lo rodea (cámaras, encoders, sensores de posición, sensores ultrasónicos, etc). Por software se puede entender a la versión programada de la técnica de control por ejemplo, el control Servo Visual, Mapeo y Localización Simultáneos (SLAM por sus siglas en inglés) y Calibración, sólo por mencionar algunas de ellas. Se puede considerar que el estado actual de desarrollo tecnológico del hardware permite desarrollar un gran número de tareas que los humanos suelen realizar en la industria y algunas labores que se muestran en la ciencia ficción. Sin embargo, el mayor reto dentro del área de robótica se encuentra en el desarrollo del software. La programación del software no es el reto en sí, lo es más bien la técnica de control que se ha de desarrollar antes de ser implementada como un algoritmo computacional. Una de las técnicas desarrolladas en las

últimas tres décadas es el control basado en visión, tratando de emular a los sistemas biológicos. Sin embargo, aún no se ha podido entender completamente cómo la percepción visual humana extrae y clasifica la información del medio, y mucho menos replicarla en forma de un algoritmo computacional práctico. Es por eso que se han buscado técnicas que permitan establecer un punto medio entre la información necesaria y suficiente que necesita ser proporcionada a un robot por medio de los sensores de visión para la realización de una tarea y toda la información contenida en una imagen. Algunas son técnicas del control "clásico" aplicadas al área de visión, tal como el control Servo Visual, y otras son una combinación de los puntos fuertes de técnicas ya existentes, como lo es el caso de la técnica de Manipulación en Espacio de Cámara (CSM por sus siglas en inglés). El hecho de que este tipo de sistemas basados en visión aparecieran en las últimas 3 décadas se debe a que los algoritmos y técnicas de optimización que han tenido que desarrollarse para procesar digitalmente las imágenes son sofisticados y elaborados, además de que había que desarrollar el hardware y los sistemas operativos que permitieran aplicarlos de manera práctica, ya que las computadoras estándar, en esos momentos, no poseían la velocidad y capacidad de memoria requerida. Como se mencionó anteriormente los sistemas de control de robots basados en visión computacional requieren cierta información extraída del flujo de imágenes que contiene una secuencia de video. La manera como se extrae esta información es a través del procesamiento digital de cada una de las imágenes que componen dicha secuencia.

En este capítulo se aborda el análisis e implementación de algunos de los métodos de procesamiento digital de imágenes involucrados en procesos de visión de bajo nivel y nivel medio que por su bajo coste computacional permitan una alta velocidad en su procesamiento, sean óptimas de implementar en la plataforma DSP y además adecúan la información, que servirá como la entrada del sistema, una vez procesada esta información se utiliza para llevar la planta al estado deseado a través del CSM, este método se explica en la primera parte de este capítulo.

3.1. Método de Manipulación en Espacio de Cámara

El método de manipulación de espacio de cámara [22] es una técnica que usa visión por computadora y que como se mencionó en la introducción de este capítulo integra una combinación de los puntos fuertes de las técnicas ya existentes tales como: a)no requerir de un proceso de calibración, ni del manipulador ni del sistema de visión, b)aplicar la intervención humana para la definición de la tarea a realizar por medio de una interfaz en la computadora de control del proceso y c)el uso de métodos de estimación para calcular los parámetros de control de las juntas rotacionales del robot. Este método hace uso de marcas visuales detectadas en las imágenes que guían al robot desde su posición inicial hasta la posición final de la maniobra que debe realizar para completar su tarea. Esta correspondencia entre el espacio de trabajo tridimensional y el espacio bidimensional que se obtiene por medio de las cámaras, se puede generar usando un modelo de cámara de orificio descrito por las siguientes ecuaciones:

$$x_{ci} = f\frac{X_i}{Z_i}, \quad y_{ci} = f\frac{Y_i}{Z_i} \tag{3.1}$$

Donde f representa la distancia focal de la cámara, (X, Y, Z) las coordenadas tridimensionales y (x, y) las coordenadas de imagen. Ya que la correspondencia entre el espacio tridimensional de trabajo y el espacio bidimensional de la imagen es una transformación entre dos sistemas coordenados se debe definir una matriz de transformación entre los dos sistemas, la cual queda definida por la siguiente ecuación:

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \begin{bmatrix} {e_1}^2 + {e_2}^2 - {e_3}^2 - {e_4}^2 & 2\left(e_2e_3 - e_1e_4\right) & 2\left(e_2e_4 - e_1e_3\right) & X_0 \\ 2\left(e_2e_3 - e_1e_4\right) & {e_1}^2 - {e_2}^2 + {e_3}^2 - {e_4}^2 & 2\left(e_3e_4 - e_1e_2\right) & Y_0 \\ 2\left(e_2e_4 - e_1e_3\right) & 2\left(e_3e_4 - e_1e_2\right) & {e_1}^2 - {e_2}^2 - {e_3}^2 + {e_4}^2 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{3.2}$$

Donde las cantidades $e_1 \ldots e_4$ representan los cuatro parámetros de Euler, los cuales satisfacen la restricción $e_1^2 + e_2^2 + e_3^2 + e_4^2 = 1$ y la columna (X_0, Y_0, Z_0) nos permite establecer la correspondencia entre el origen del sistema de coordenadas (x, y, z) y el sistema de coordenadas en la cámara (X, Y, Z). De acuerdo a una simplificación del modelo de cámara aquí presentado [52] se forma un vector de parámetros $C = [C_1 \ldots\ldots\ldots C_6]^T$ que reciben el nombre de parámetros de visión, los cuales definen la relación entre la posición de un punto en el espacio tridimensional y su correspondiente posición en el espacio bidimensional (el espacio asociado a la imagen). Ya que estos parámetros definen la relación de posición entre los dos espacios, se asocian a la configuración angular del robot para definir un vector de posición angular de la siguiente forma: $(h_y\left(R_X(\Theta), R_Y(\Theta), R_Z(\Theta), C\right))$. Este vector describe la localización de cada punto al cual el robot debe llegar a posicionarse a través de su modelo cinemático y donde $\Theta = (\theta_1, \theta_2 \ldots\ldots \theta_n)$representa la configuración angular de un robot de n grados de libertad.

El correcto posicionamieto del robot en el punto al que se le quiere llevar se logra cuando el valor del vector $(h_y\left(R_X(\Theta), R_Y(\Theta), R_Z(\Theta), C\right))$ es igual al valor de la coordenada en el plano bidimensional de la imagen y queda dado por la función de la ecuación 3.3, esto significa que el correcto posicionamiento se obtiene cuando el valor de la función es igual a cero. Dado que es una función parametrizada, se hace necesario recurrir a métodos de estimación para lograr que la función converja a cero. Dado que el número mínimo de cámaras para implemetar el CSM es de dos, se pueden involucrar muchas más camaras y se puede definir que para la j^{esima} cámara del CSM, los parámetros de visión C son estimados por la minimización de

la siguiente función:

$$\begin{aligned} J(C) &= \sum_{k=1}^{m} \sum_{i=1}^{n} \left[x_{ci} - h_x \left(R_X(\Theta), R_y(\Theta), R_Z(\Theta), C \right) \right]^2 \\ &+ \left[y_{ci} - h_y \left(R_X(\Theta), R_Y(\Theta), R_Z(\Theta), C \right) \right]^2 W_i W_k \end{aligned} \tag{3.3}$$

Donde m es el número de pares de juntas de rotación y muestras de la cámara, n es el número de puntos visuales encontrados en cada toma de la cámara, y W_i y W_k son factores de peso.

Las coordenadas x_{ci} y y_{ci} en el plano de imagen son las coordenadas de las marcas visuales que identifican la posición del efector final del robot, las coordenadas por si mismas deben ser determinadas en función de la información visual que las marcas presentan en la imagen que los sensores (en este caso, cámaras de video) están captando constantemente. Esta información puede ser representada o estar presente como ciertas formas o figuras que resalten del entorno de trabajo o que pueden asumirse también como una extensión del mismo robot. Sin embargo, no existe un algoritmo computacional que pueda abarcar la identificación de todas y cada una de las maneras en que la información visual está presente en el campo de trabajo del robot tal y como la visión humana puede hacerlo. La visión humana puede lograr la identificación de las formas, marcas, dibujos, colores y perspectivas basada en el conocimiento adquirido a través de vastas experiencias de aprendizaje que se le presentan a lo largo de su vida. Sin embargo, los sistemas de visión no tienen este tipo de conocimientos basados en la experiencia, por lo que la información que se les proporcione debe estar específicamente definida en función a la labor a realizar, ya que una simple labor como la colocación de una herramienta en un punto determinado puede llegar a convertirse en una tarea complicada sin la correcta especificación de la información necesaria para poder lograrla. Para poder especificar esa información se requiere reducir el número de variables que se le deben entregar al sistema de visión y que provienen de la imágenes que está captando. De manera que es necesario establecer cuál es la mejor información específica que se le

puede entregar; uno de los mejores datos que componen esa información proviene de los contrastes entre texturas de diferentes colores, donde las áreas de interés se diferencian de las otras áreas emplazadas en el mismo ambiente de trabajo [55].

Es por eso que las imágenes en un sistema de visión son sujetas a un proceso de análisis. Sin embargo, las imágenes están sujetas a muchas distorsiones provenientes del medio, tales como la iluminación, defectos en las lentes, superficies irregulares, ambientes en constante cambio, etc., lo que reduce la eficiencia de los algoritmos de procesamiento y, por lo tanto, la precisión de la información que se extrae de las imágenes. Antes de continuar con la discusión acerca de los algoritmos que extraen la información de la imagen se definirá lo que es un sistema de visión, que niveles de visión computacional existen y cuales son los más adecuados para el CSM.

3.2. Sistemas de Visión por Computadora

Como se muestra en la figura 3.1, en la práctica, un sistema que involucre visión por computadora constaría de los siguientes procesos: a) la percepción, proceso a través de cual se obtiene una imagen visual, b) la construcción de una representación densa, la cual incluye técnicas de preprocesamiento tales como la reducción de ruido y el realce de detalles, c) la extracción de información importante, es el proceso que divide una imagen en objetos que sean de interés, d) construcción de un modelo parcial, donde se obtienen características convenientes no solo para diferenciar un tipo de objeto de otro, si no que, una vez diferenciados, estos también puedan ser identificados. e) la interpretación y toma de decisiones, lo cual corresponde a la asociación de un significado a los objetos reconocidos y, a partir de esos significados, generar una acción a realizar.

Por conveniencia, se agrupan estas diversas áreas de acuerdo con la complicación o facilidad con la que se lleva a cabo su implementación. De esta manera, se consideran tres niveles de procesamiento: visión de bajo, medio y alto nivel. Los incisos

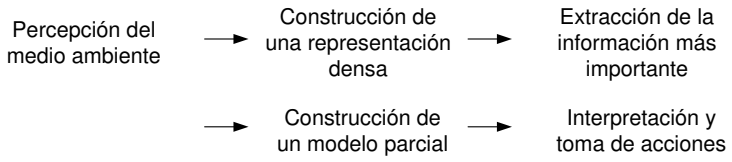

Figura 3.1: Sistema de visión por computadora.

a) y b) pueden clasificarse como una visión de bajo nivel, los incisos c) y d) como de nivel intermedio, y el inciso e) como visión de alto nivel [23]. Los sistemas de visión alto nivel comúnmente están asociados al área de la Inteligencia Artificial. El CSM no es un método que use regularmente Inteligencia Artificial, y aunque se han desarrollado trabajos en esa dirección [24], para el caso de este proyecto de tesis se abordaran las técnicas de procesamiento digital de imágenes presentes en la visión de bajo nivel y nivel medio que a continuación se definirán.

3.2.1. Visión de bajo nivel y nivel medio

Se puede definir la visión de bajo nivel como aquellos procesos que son primarios, en el sentido de que pueden ser considerados como automáticos, es decir, no requieren la emulación de ningún tipo de inteligencia, en nuestro caso la información visual se convierte a señales eléctricas por medio de sensores visuales como pueden ser scanners y cámaras. En ambos casos se obtiene una representación densa, la cual consiste en un arreglo matricial con un número relativamente grande de renglones y columnas. Cada elemento de este arreglo recibe el nombre de pixel y su valor está asociado al nivel de gris para imágenes en blanco y negro o al porcentaje RGB en las imágenes a color. Una vez realizada esta acción se pasa a las compensaciones del preprocesamiento, tales como reducción de ruido, para finalmente pasar a la obtención de características de la imagen. La visión de nivel medio esta asociada a aquellos procesos que extraen, caracterizan y etiquetan componentes de la imagen que se obtiene de la visión de bajo nivel.

3.2.2. Extracción de información de una imagen.

Una parte muy importante de un sistema que involucre visión por computadora lo constituye la parte dedicada a la extracción de información o rasgos distintivos de una imagen. Este proceso cae dentro de las acciones clasificadas como visión de nivel intermedio. Por otro lado, existe un número significativo de técnicas para obtener los rasgos distintivos de una imagen bidimensional, las cuales están clasificadas en dos grupos principales: a)Técnicas en el dominio del espacio, y b)técnicas en el dominio de la frecuencia [25].

Es necesario por lo tanto definir los criterios principales que lleven a definir cual de estas dos técnicas es la más adecuada para ser aplicadas al CSM. El CSM utiliza marcas visuales para simplificar el procesamiento de la información visual y estas pueden ser típicamente de dos tipos, puntos laser sobre una superficie y marcas en forma de anillo concéntrico, de dos tipos, con el centro obscuro y con el centro claro como pueden verse en la figura 3.2. Este tipo de marcas visuales permiten reducir la cantidad de información a extraer y por lo tanto debe redundar en una alta velocidad de procesamiento computacional. De esta manera es posible establecer dos criterios principales: bajo costo computacional de implementación y velocidad de procesamiento alta. Una vez definidos los criterios, se expondrán las principales características de ambas técnicas para poder ver cual de las dos se ajusta más a los criterios que se han establecido. Se inicia por las técnicas en el dominio de la frecuencia.

3.3. Técnicas en el dominio de la frecuencia

Una imagen es mapeada al dominio de la frecuencia como resultado de haberle sido aplicada el par de transformadas de Fourier bidimensionales para una imagen de $N \times N$y que están definidas como:

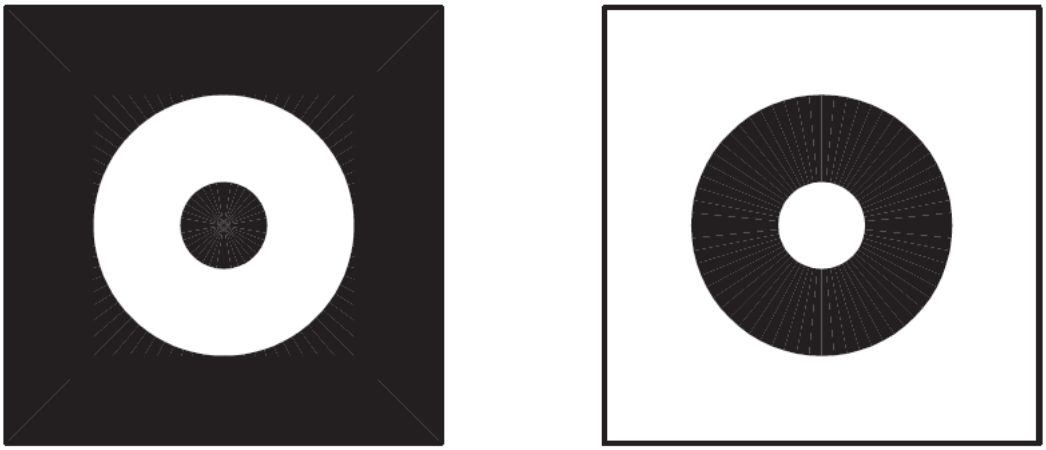

Figura 3.2: Marcas visuales en forma de anillo.

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux+vy)/N} \tag{3.4}$$

para $u, v = 0, 1, 2..., N-1$ y

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux+vy)/N} \tag{3.5}$$

para $x, y = 0, 1, 2, ..., N-1$

El dominio de frecuencia trabaja con conjuntos de pixeles complejos, resultado de aplicar la transformada de Fourier a una imagen. Debido a la gran cantidad de procesamiento que se requiere, para los procesadores actuales, los métodos en el dominio de la frecuencia no están tan extendidos en las aplicaciones de control de robots basadas en visión como las técnicas en el dominio del espacio [25]. No obstante la transformada de Fourier juega un papel importante en áreas como el análisis de movimiento de objetos y su descripción. Por otra parte, muchas técnicas especiales de realce y restauración se fundan en conceptos cuyo origen es la transformada de Fourier.

La transformada discreta bidimensional de Fourier tiene abundantes aplicaciones

en la reconstrucción de imágenes, realce y recuperación, aunque, como se mencionó, la utilización de esta herramienta en sistemas de visión en tiempo real está todavía bastante restringida debido al alto costo computacional que se requiere para implementarla en aplicaciones de este tipo.

3.4. Técnicas en el dominio del espacio

El dominio del espacio se refiere al conjunto de pixeles que componen una imagen, y las técnicas del dominio espacial son procedimientos que operan de forma directa en estos pixeles. Las funciones de preprocesamiento en el dominio espacial se pueden expresar como:

$$g(x,y) = h[f(x,y)] \tag{3.6}$$

donde $f(x,y)$ es la imagen de entrada, $g(x,y)$ es la imagen que se obtiene y h es un operador definido sobre el entorno de vecindad. Se puede hacer también que h opere sobre un conjunto de de imágenes centradas, como la suma de pixel-a-pixel de K imágenes para la reducción de ruido. Este tipo de técnicas hace uso de operaciones aritméticas/lógicas y de operaciones compuestas tales como la suma de productos, para poder extraer cierta información de la imagen. Este tipo de operaciones caen dentro de los criterios que se establecieron en la sección 3.2.2 ya que a diferencia de las operaciones en el dominio de la frecuencia, donde se trabaja con números y coeficiente complejos, en el dominio espacial se trabajan con operaciones básicas, como son la suma, la resta, la multiplicación y las operaciones lógicas, lo que no representan un gran costo computacional. Por otro lado el tipo de datos de entrada/salida que manejan, así como la propuesta de usar una plataforma DSP para su procesamiento, debe permitir cumplir con el objetivo de alta velocidad de procesamiento. Existen además varios tipos de técnicas en el domino espacial entre las cuales destacan: Borrado gaussiano, el realce, la obtención de nitidez, el suavizado, la diferenciación de imágenes y la detección de bordes. Dado el tipo de marcas

visuales y la información que se requiere adquirir, como lo son las coordenadas de sus centroides, se eligieron las técnicas de diferenciación de imágenes y detección de bordes, las cuales se explican a continuación.

3.4.1. Diferenciación de Imágenes

Dentro de la técnicas en el dominio del espacio existen las que hacen uso de operaciones aritméticas/lógicas, estas operaciones son desarrolladas básicamente entre dos o mas imágenes (eso excluye al operador NOT que puede ser aplicado en una sola imagen). La diferencia entre dos imágenes $l(x,y)$ y $m(x,y)$ expresada como:

$$q(x,y) = l(x,y) - m(x,y) \tag{3.7}$$

es obtenida computando las restas entre todos los pares de pixeles que se corresponden de l y m. El resultado es una imagen cuyo pixel en las coordenadas (x,y) corresponde a la resta de los pixeles en la misma localización en las dos imágenes que han sido restadas.

Cabe mencionar que la técnica diferenciación de imágenes es usada en el área de segmentación. Básicamente, la técnica de segmentación consiste en subdividir la imagen dentro de varias regiones, basadas en un criterio específico. La diferencia de imágenes es utilizada cuando el criterio está en constante "cambio". De esta manera, en una tarea de seguimiento de vehículos en movimiento en una secuencia de imágenes, la diferenciación es usada para remover todos los componentes estacionarios en una imagen. El resultado es que sólo se conservan los elementos que se mueven en la imagen más algo de ruido [26].

Para el caso particular de esta tesis, el procedimiento consiste en obtener una imagen con el láser encendido y una segunda imagen de la misma superficie pero esta vez con el láser apagado, tal como se muestra en la figura 3.3. La diferencia de las dos imágenes consiste en un fondo obscuro en el que resaltan de manera muy

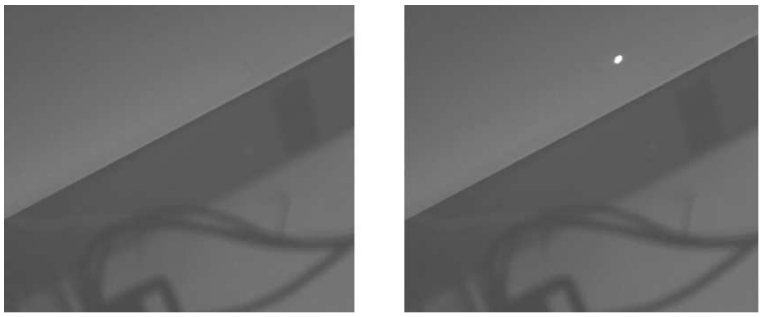

Figura 3.3: Láser encendido y láser apagado.

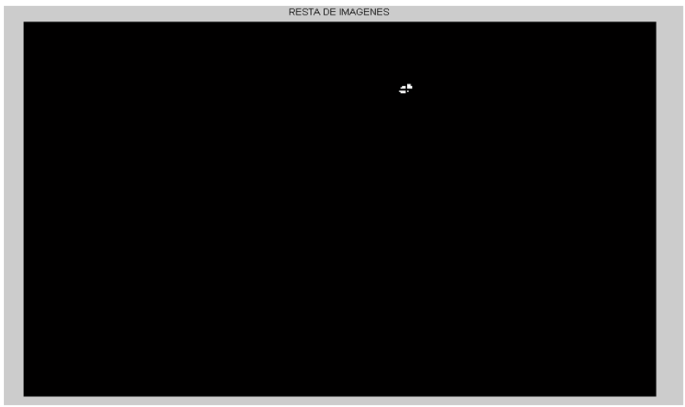

Figura 3.4: Diferencia de imágenes.

evidente las manchas producidas por la fuente luminosa. El siguiente paso es calcular las coordenadas del centroide de la mancha para obtener su ubicación en el plano de la imagen cuyo resultado se muestra en la figura 3.4.

3.4.2. Detección de bordes

Un borde en una imagen se puede detectar como un cambio en el contraste o en la intensidad de la escala de grises, normalmente este cambio es asociado a una discontinuidad. Discontinuidades de este tipo en una imagen tal y como se pueden apreciar en la figura 3.5 pueden ser [27]:

De tipo escalón. En este caso la intensidad en la imagen cambia de manera abrup-

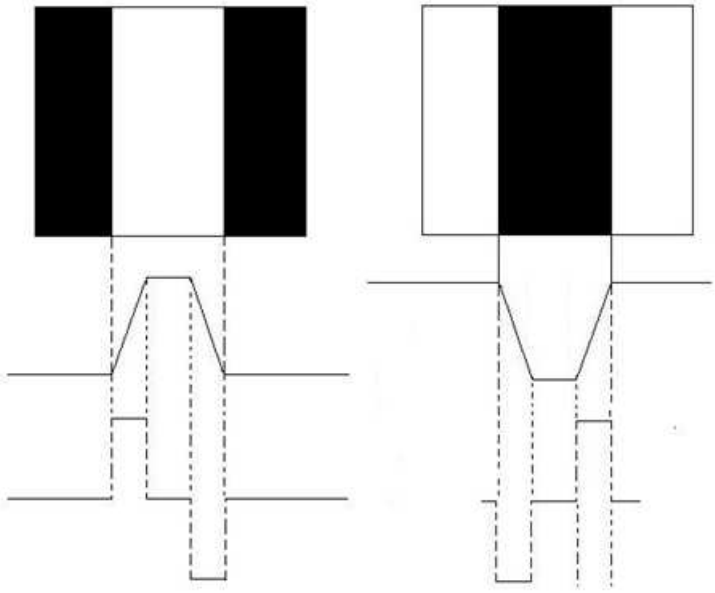

Figura 3.5: Tipo de bordes.

ta de un valor en un lado de la discontinuidad a otro valor muy diferente en el lado opuesto.

Lineales. En este caso la intensidad en la imagen cambia de nuevo abruptamente de valor pero regresa al valor inicial en una distancia corta.

Debido a las componentes de baja frecuencia en la señal o al alisamiento introducido por la mayoría de los dispositivos de captación, rara vez se presentan discontinuidades agudas en una imagen real. Los bordes de tipo paso o escalón aparecen como bordes tipo rampa, mientras que los borde tipo línea aparecen como bordes tipo techo. Los cambios de intensidad no se dan de manera instantánea sino a lo largo de distancias finitas.

El modelado de estos bordes, como anteriormente se ha descrito, permite que su primera derivada sea cero en todas las regiones de intensidad constante y tenga un valor constante en toda la transición de intensidad. Basándonos en esto, es evidente que el valor de la primera derivada puede utilizarse para detectar la presencia de un

borde.

La detección de bordes se logra con el cálculo de un operador de derivada local para formar la máscara de convolución. Una máscara de convolución (plantilla, ventana o filtro) es, básicamente, una pequeña matriz bidimensional, cuyos coeficientes se eligen de forma que podamos detectar una propiedad dada para una imagen.

3.4.3. Operadores de primera derivada y primera aproximación al gradiente

Como se mencionó anteriormente, se puede usar el cómputo de la primera derivada de la señal para detectar un borde. El gradiente como se sabe, es el equivalente de la primera derivada y se define como el vector

$$\mathbf{G}\left[f\left(x,y\right)\right] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{3.8}$$

cuya magnitud viene dada por

$$G\left[f(x,y)\right] = \sqrt{G_x^2 + G_y^2}, \tag{3.9}$$

que como se sabe, iguala la tasa máxima de crecimiento de $G\left[f(x,y)\right]$ por unidad de distancia en la dirección G.

En el caso de una imagen digital, las derivadas en la ecuación pueden aproximarse de varias maneras, una de ellas es a través de primeras diferencias, como sigue:

$$G_x = \frac{\partial\left(x,y\right)}{\partial x} = \triangle_x f\left(x,y\right) \cong f\left(x+1,y\right) - f\left(x,y\right) \tag{3.10}$$

$$G_y = \frac{\partial f\left(x,y\right)}{\partial y} = \triangle_y f\left(x,y\right) \cong f\left(x,y+1\right) - f\left(x,y\right) \tag{3.11}$$

Estas aproximaciones pueden llevarse a cabo a través de las siguientes máscaras de convolución:

$$G_x = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ y } G_y = [-1, 1]$$

No es difícil darse cuenta que estas aproximaciones producen valores en la posición interpolada $\left(x + \frac{1}{2}, y + \frac{1}{2}\right)$ y no en la posición deseada (x, y).

El operador cruzado de Roberts, la máscara básica para efectos de procesamiento computacional, proporciona una aproximación sencilla para el cálculo de la magnitud del gradiente y está dado por:

$$G[f(x,y)] = f(x,y) - f(x+1,y+1) + f(x+1,y) - f(x,y+1) \tag{3.12}$$

Al usar máscaras de convolución, este operador se transforma en:

$$G[f(i,j)] = G_x + G_y \tag{3.13}$$

donde G_x y G_y son calculados usando las siguientes máscaras:

1	0
0	-1

0	1
-1	0

3.4.3.1. Operador de Sobel

Para evitar el cálculo del gradiente en una posición interpolada entre pixeles, se usa una máscara basada en una vecindad de 3×3, llamada operador de Sobel, mostrado a continuación:

a_0	a_1	a_2
a_7	(x, y)	a_3
a_6	a_5	a_4

En este caso, la magnitud del gradiente de Sobel puede obtenerse como sigue:

$$M = \sqrt{s_x^2 + s_y^2} \tag{3.14}$$

Para este caso las derivadas parciales de s_x y s_y son calculadas de la siguiente manera:

$$s_x = (a_6 + ca_5 + a_4) - (a_0 + ca_1 + a_2) \tag{3.15}$$

$$s_y = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{3.16}$$

donde la constante c toma el valor de 2. Para obtener la dirección del gradiente se utiliza la expresión:

$$\theta(x, y) = \arctan \frac{G_y}{G_x} \tag{3.17}$$

Este operador tiene la propiedad añadida de suavizar la imagen, eliminando parte del ruido y, por consiguiente, minimizando la aparición de falsos bordes debidos al efecto de magnificación del ruido por parte del operador derivada como puede verse en la figura 3.6.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

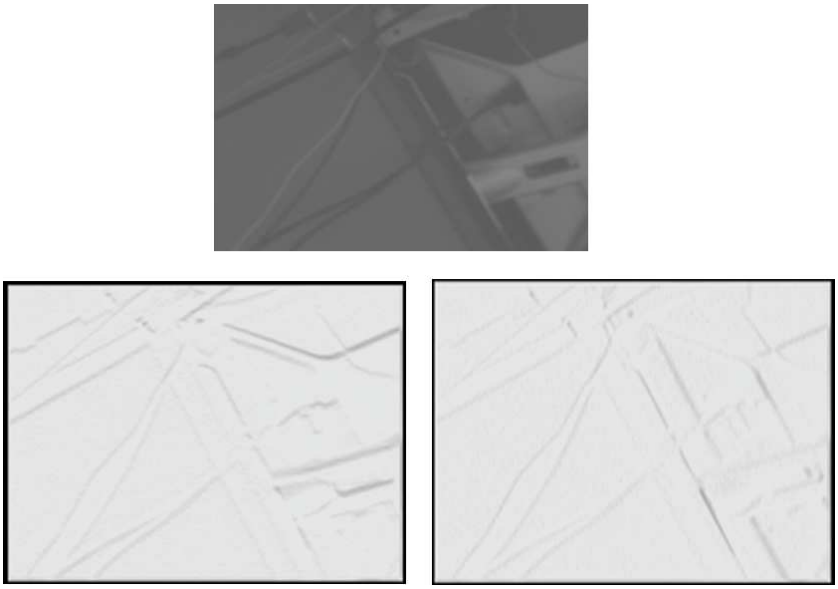

Figura 3.6: Resultado de aplicar la máscara de Sobel.

3.4.3.2. Operador de Prewitt

El operador de Prewitt usa las mismas ecuaciones que el operador de Sobel, excepto que la constante $c = 1$. El par de máscaras a través de las cuales puede ser utilizado este operador son las siguientes:

-1	-1	-1
0	0	1
1	1	1

-1	0	1
-1	0	1
-1	0	1

El operador de Prewitt otorga el mismo peso a los pixeles contiguos en vertical y horizontal, que a los contiguos en diagonal, y el de Sobel duplica el coeficiente de los primeros teniendo en cuenta que sus centros están más próximos al pixel central. El operador de Prewitt es el medio adecuado para estimar la magnitud y la orientación de un borde. Esta información se obtiene de aplicar directamente la máscara a la imagen. Con este operador también es posible detectar la orientación de un borde en ocho posibles direcciones. Ocho máscaras se pueden utilizar para esta operación. El conjunto de esas máscaras se produce mediante la adopción de

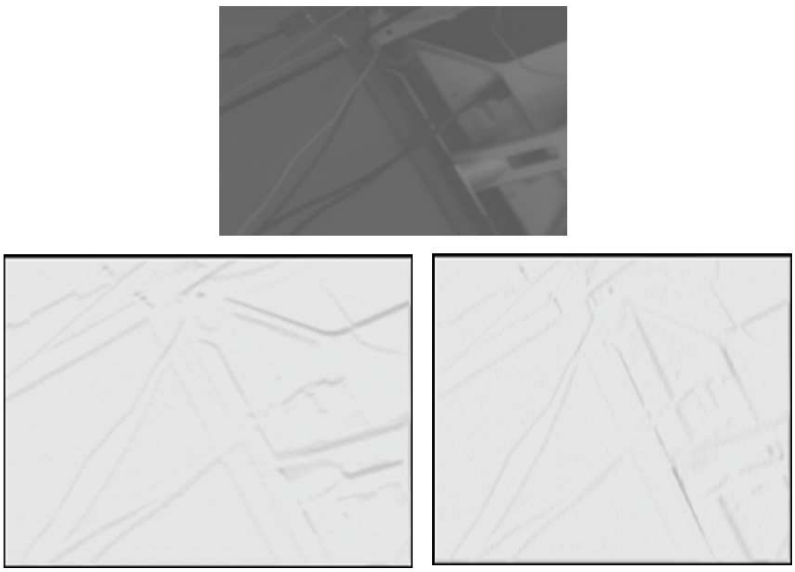

Figura 3.7: Resultado de aplicar la máscara de Prewitt.

una de las máscaras (en general la que detecta bordes en dirección horizontal) y la rotación de sus coeficientes circularmente. Cada una de las máscaras resultantes es sensible a la orientación de un borde que van desde 0° a 315° en pasos de 45°, donde 0° corresponde a un borde vertical. El máximo de respuesta para cada pixel es el valor del pixel correspondiente en la magnitud de salida de imagen. Los valores de orientación de la salida de imagen se encuentran entre 1 y 8, dependiendo de cuál de los 8 núcleos produjo la respuesta máxima Un ejemplo del resultado del procesamiento aplicando mascaras horizontales y verticales de Prewitt puede verse en la figura 3.7.

3.4.3.3. Operador de Kirsch

Una generalización de los dos operadores antes mencionados son las máscaras de Kirsch. El operador de Kirsch se conoce como un operador "brújula", lo cual puede entenderse como que detecta bordes orientados en diferentes direcciones en la imagen. Para esto requiere el uso de ocho máscaras que representan las ocho direcciones angulares más comunes. Dichos operadores se pueden definir a diferentes

tamaños como 2x2, 3x3, 5x5. Para cada punto de la imagen se obtienen 8 valores, resultantes de la convolución con cada una de las máscaras, el valor del módulo del gradiente resulta ser el máximo de esos 8 valores, mientras que la dirección queda determinada por el ángulo asociado a la máscara que ha generado dicho valor máximo. Las máscaras para la dirección vertical y horizontal se muestran a continuación y los resultados del procesamiento se muestran en la figura 3.8

5	-3	-3
5	0	-3
5	-3	-3

5	5	5
-3	0	-3
-3	-3	-3

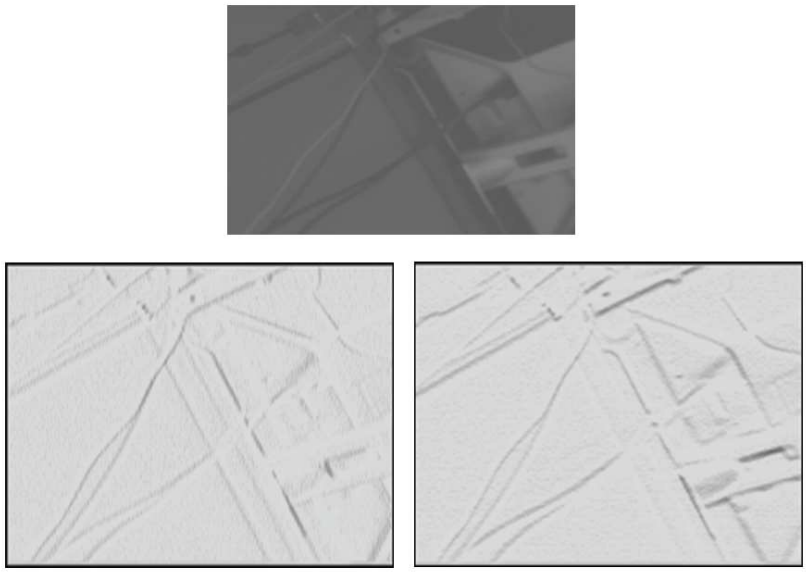

Figura 3.8: Resultado de aplicar la máscara de Kirsch.

Una vez que se han descrito las técnicas que se han elegido para extraer la información necesaria de las imágenes para el sistema de visión del CSM, se hace necesario analizar si la información que requieren como entrada y la información que arrojan como salida puede llegar a comprometer el desempeño de la plataforma DSP que se propuso en el capítulo 2. La última sección de este capítulo da una justificación de por que estas técnicas son viables de implementar con resultados adecuados de desempeño en plataformas DSP.

3.5. Justificación del uso de los algoritmos en el DSP

El gran reto del procesamiento digital de imágenes es claro cuando se considera que la retina humana puede desarrollar 10 mil millones de operaciones por segundo aproximadamente. La corteza visual que analiza toda esta información tiene una gran capacidad computacional, esto contrasta con la velocidad actual de procesamiento de una computadora personal. De hecho, esa cantidad de información podría representar el límite superior en cuanto a velocidad de procesamiento de las imágenes. Los actuales procesadores se encuentran limitados en procesamiento por el uso de dispositivos periféricos y la administración del hardware propio. Esto motivó a proponer el uso de las plataformas DSP como una alternativa de procesamiento.

Las aplicaciones multimedia con limitantes en tiempo real, demandan una gran velocidad de procesamiento y manejo de datos a las arquitecturas de procesadores digitales de señales. Para resolver este tipo de requerimientos los procesadores de TI emplean una palabra de instrucción muy grande (VLIW), la cual es usada en combinación con múltiples unidades aritméticas, permitiendo el uso de múltiples-datos en una sola instrucción sencilla (SIMD). Con el fin de justificar la selección de los algoritmos de procesamiento de imágenes usados en esta aplicación, en base a lo anteriormente mencionado, a continuación se muestra una clasificación de los algoritmos que se usan en aplicaciones multimedia acorde a [28].

a) Algoritmos de bajo nivel: presentan características predeterminadas independientes del control y flujo de los datos. Los datos de entrada y salida adquieren el formato icono (por ejemplo los datos pueden ser interpretados como imágenes). Debido a esto, tales algoritmos ofrecen un muy alto potencial para el uso de arquitecturas VLIW. Ejemplos de estos algoritmos son los filtros, las transformadas, los estimadores de movimiento y los algoritmos morfológicos.

b) Algoritmos de nivel medio, como los algoritmos de bajo nivel: su entrada son

datos que pueden ser clasificados como iconos y que describen una imagen, pero los datos de salida se interpretan simbólicamente, ya que describen las características de una imagen. Los algoritmos de nivel medio usualmente tienen una dependencia débil del flujo de control de datos (por ejemplo, una cantidad limitada de saltos condicionales). Ejemplo de ellos son los seguidores de contornos en imágenes binarias o los histogramas de niveles de gris.

c) Algoritmos de alto nivel: operan con entradas y salidas simbólicas y los flujos de datos y control son altamente dependientes de los datos que manejan. Los algoritmos de alto nivel son usualmente considerados no paralelizables a nivel de datos. Ejemplos típicos son los algoritmos de codificación, como es el caso de la codificación aritmética. De la anterior clasificación se puede inferir que los algoritmos de bajo nivel son candidatos para aprovechar las arquitecturas VLIW. Debido al hecho de que pueden ser independientes del control que se aplica a la captura de los mismos, permiten su extracción estática, evitando la programación y ejecución de órdenes fuera de la arquitectura del DSP.

Lo expuesto en este capítulo permite concluir que, por un lado, la información que los algoritmos entregan es la necesaria para que el CSM pueda realizar la tarea que se le ha encomendado; por otra parte, gracias a los datos numéricos puros que manejan, estos algoritmos permiten, en teoría, ser manejados adecuadamente por la arquitectura del hardware que se propuso en el capítulo dos para la implementación de este trabajo de tesis, lo cual debe permitir incrementar el desempeño y la velocidad del procesamiento.

Una vez descrito el método CSM y los algoritmos, se muestra a continuación como está constituida y configurada la plataforma digital de desarrollo considerada para la implementación del método.

Capítulo 4

Programación de algoritmos en el DSP y resultados experimentales

La implementación de algoritmos de procesamiento de imágenes y visión es una tarea que representa un reto. Cuando se trabaja con aplicaciones fuera de línea se utilizan comúnmente computadoras estándar, donde poderosas herramientas de software están disponibles, haciendo que los procesos para la implementación de la tarea sean fácilmente desarrollados. Tales sistemas mantienen su compatibilidad con versiones anteriores, de esta manera, el software puede ser reimplementado tan pronto como una nueva y más rápida plataforma esté disponible en el mercado. Desafortunadamente una solución basada en una PC estándar no es viable en muchos casos donde se requiera procesamiento en tiempo real. Sin embargo, como se discutió en el capítulo dos, en los años recientes, plataformas especiales de hardware han sido desarrolladas para satisfacer los requerimientos de procesamiento en tiempo real, con lo cual soluciones más flexibles y asequibles son basadas en dispositivos programables, principalmente en Procesadores Digitales de Señales (DSP's) y Arreglos de Compuertas de Campo Programable (FPGA's). La razón desempeño/costo para tales dispositivos se ha incrementado convirtiéndolos en una alternativa válida para el diseño de soluciones en tiempo real al nivel de plataforma. Además, permiten

el desarrollo de sistemas programables de propósito general, los cuales pueden ser adoptados por los desarrolladores de software para implementar algoritmos "personalizados", haciendo que sean ampliamente usados cuando se requieren prototipos de procesamiento muy veloces. Como también se mencionó en el capítulo dos, se presentan dos objeciones al uso de plataformas programables: el conocimiento altamente especializado del hardware que se usa y el uso de lenguaje ensamblador que dificulta la obtención de código optimizado. El sistema desarrollado, y que se presenta en este capítulo, se implementa en el DSP DM6437 de Texas Instruments el cual permite soslayar las anteriores objeciones al proveer librerías y marcos de trabajo que hacen posible configurarlo sin un conocimiento detallado del mismo. Posee un Ambiente Integrado de Desarrollo con una poderosa y flexible Interfaz Gráfica de Usuario, y un moderno compilador que permite el uso de un lenguaje de nivel medio, como lo es el lenguaje **C**, para obtener implementaciones con código optimizado. El sistema también captura una señal de video de una fuente analógica, la digitaliza, lee las variables de interés que después procesa, extrae y manipula, y despliega de forma analógica el resultado del procesamiento. La manera de implementar los algoritmos es flexible, permitiendo programar por rutinas separadas cada uno de los procesos necesarios dentro del DSP. Lo anterior, aunado a la arquitectura del software de programación permite cumplir el objetivo de portabilidad del proyecto expuesto en este documento de tesis.

4.1. Descripción del Hardware

El Módulo de Evaluación de Video (EVM por sus siglas en inglés) DM6437 es una plataforma que permite al usuario desarrollar y evaluar aplicaciones para la familia de procesadores Da Vinci de Texas Instruments. El EVM está provisto de un conjunto de dispositivos que le permiten ser aplicado en una amplia variedad de ambientes, para una descripción mas completa de los componentes del EVM ver [29].

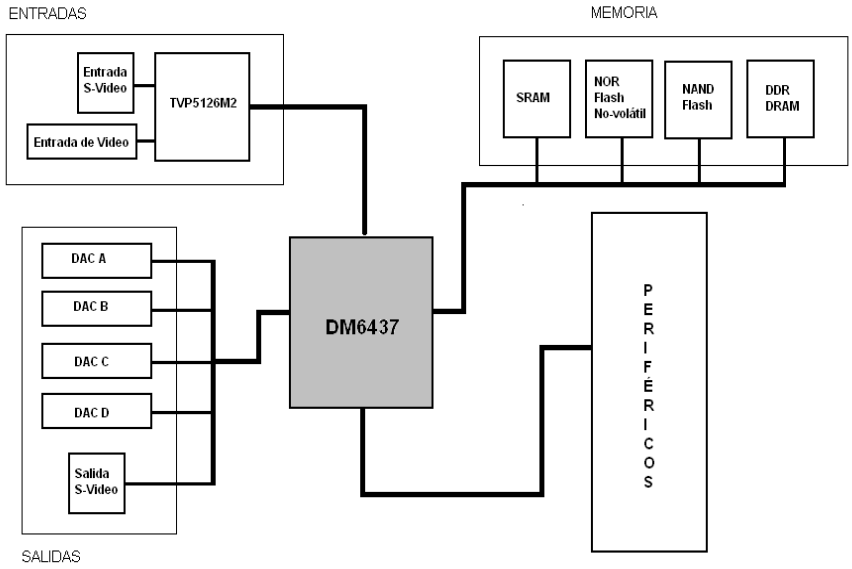

Figura 4.1: Principales módulos del EVM

Los dispositivos que componen el EVM se pueden agrupar en 5 módulos, como se muestra en la fig. 4.1.

- (i) DM6437,
- (ii) Memoria,
- (iii) Entrada de video,
- (iv) Salida de video,
- (v)Periféricos.

El DM6437 es a su vez un amplio sistema de procesamiento que incluye: el sistema de control, el subsistema de procesamiento de video (VSS por sus siglas en inglés), los periféricos y el procesador TMS320C64x+, para una descripción mas detallada ver [30]. El DSP TMS320C64x+ es el núcleo del sistema y la plataforma de más alto desarrollo de la generación de DSP de la familia TMS320C6000 de punto fijo. EL DM6437 está basado en la tercera generación de alto desempeño desarrollada por

TI y que consta de arquitecturas VelociTI y VLIW (very long instruction-word), haciendo de este DSP una excelente opción para aplicaciones de medios digitales, y cuyas principales características se muestran en la tabla 4.1.

El módulo de memoria contiene cuatro bancos: 1) 16 Mbytes de memoria Flash no-volátil, la cual es usada para almacenar permanentemente el *firmware* de la aplicación en el DSP; 2) 64 Mbytes de NAND Flash para almacenar grandes cantidades de datos, se usa generalmente en conjunto con la memoria Flash no-volátil; 3) 2 Mbytes de SRAM, usada principalmente para almacenamiento de datos a alta velocidad; 4) 128 Mbytes DDR DRAM, usada para almacenar los datos de la imagen y la configuración del DSP (esto último durante la etapa de emulación de los algoritmos).

La entrada y salida de video pueden agruparse como una etapa de procesamiento analógico. La entrada consta de un chip video decodificador TVP5146M2, que soporta video compuesto o S video. La función del TVP5146M2 es convertir las señales de video analógico en señales digitales y los datos digitales en formato YUV 4:2:2. La etapa de salida consta de 4 DACs uno para cada componente de video RGB y uno para el video compuesto, además de una salida de S vídeo.

4.2. Arquitectura de Software del DSP

Los DSPs son comúnmente programados como microprocesadores "tradicionales" embebidos. Esto es, son programados con una mezcla de lenguaje **C** y ensamblador, acceden directamente al hardware, y, por razones de desempeño, casi siempre tienen poco o nada de soporte de un sistema operativo estándar. Sin embargo, los DSPs están diseñados para correr sofisticados algoritmos de procesamiento pero que por causa de diseño no es posible usarlos en más de un sistema sin un trabajo significativo de reingeniería. Es por esta razón que TI decidió estandarizar una arquitectura de software que permitiera a los programadores integrar uno o varios algoritmos en una

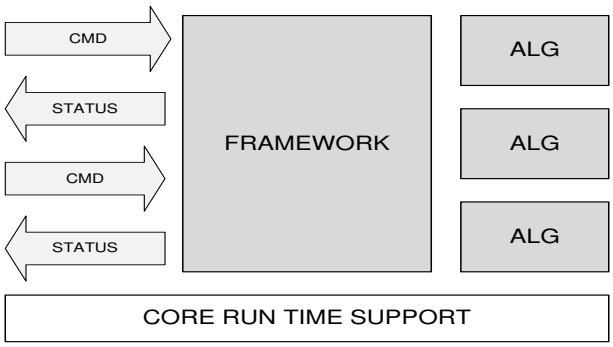

Figura 4.2: Arquitectura de software.

o varias plataformas de características similares.

La mayoría de las arquitecturas de los sistemas pueden ser divididas como se muestra en la figura 4.2, en donde se observa que los algoritmos son transductores de datos "puros", por ejemplo, ellos simplemente toman datos de los buffers de entrada y producen algunos datos para los buffers de salida. el *core run time support* hace el intercambio entre las memorias y contiene las funciones que habilitan o deshabilitan interrupciones. El *framework* es el elemento que integra los algoritmos con las fuentes de datos en tiempo real y los une usando el *core run time support* para crear un subsistema completo. Los *frameworks* interactúan con los periféricos en tiempo real (incluyendo otros procesadores que pudieran agregarse al hardware del sistema) y definen las interfaces de I/O para los componentes de los algoritmos.

Con la ayuda del IDE *Code Composer Studio* que fue facilitado por Texas Instruments, y el Paquete de Soporte para Plataforma (PSP por sus siglas en inglés) [32] fue suministrado por la empresa Spectrum Digital, se conformó el marco de trabajo necesario para implementar la arquitectura de software descrita en la fig 4.2. Los dos principales componentes de la arquitectura son el DSP/BIOS y los manejadores PSP tal como se muestra en la fig 4.3.

Estableciendo un paralelismo con la arquitectura de la fig 4.2, se hace notar que el DSP/BIOS toma el lugar del *Core run time support*. El DSP/BIOS es un kernel de

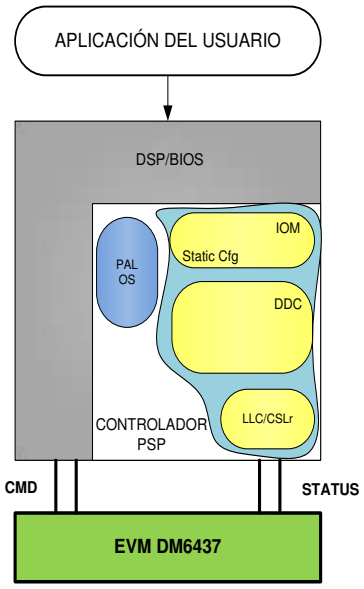

Figura 4.3: Arquitectura de software de la aplicación.

tiempo real, escalable y multi-tarea (es decir, un mini sistema operativo) diseñado específicamente para la plataforma DSP que se usa en este proyecto, para una información mas detallada ver [31]. Por otro lado los manejadores PSP toman el papel del *framework*. Estos no solo permiten enlazar el algoritmo con todos los periféricos existentes en el hardware, sino que nos permiten, además, configurarlos, convirtiéndose en una poderosa herramienta para optimizar el desempeño de las aplicaciones que se programaron. El PSP Drive está dividido en distintos subcomponentes mostrados en la figura 4.3 y son descritos a continuación.

- Capa específica del dispositivo, compuesta a su vez de dos subsistemas más; Capa de Registro CSL, compuesta de un grupo de constantes simbólicas (#defines) que exponen los detalles de los registros de configuración del hardware; y Capa LLC que es la capa del controlador de bajo nivel (*low level contoller*), compuesta de un conjunto de funciones que configuran todas las características soportadas por el hardware.

- *Device Driver Core* (DDC), compuesta de un conjunto de funciones que al ser

implementadas constituyen una interface funcional para el driver y que soportan operaciones que inician/finalizan el dispositivo de hardware, operaciones que actualizan el estado de los dispositivos, operaciones para programar los registros de dispositivos de hardware para cambiar los parámetros de su configuración, y operaciones de monitoreo de los estados actuales de los dispositivos de hardware.

- Capa IOM,ésta tiene la obligación de mantener los parámetros para el DDC durante su etapa de creación. El IOM implementa un conjunto de funciones que adaptan el driver al Sistema Operativo. Debe hacerse notar que la capa DDC no existe por sí misma, ésta es implementada por la capa IOM cuando el driver del dispositivo es cargado por el sistema.

- Capa Abstracción de Plataforma para servicios de SO (PALOS). Debe de hacerse notar que no todos los servicios de adaptación del Sistema Operativo (SO) son solicitados por el DDC por razones de desempeño, el resto de los servicios son separados en unidades de compilación y sólo son cargadas por la PALOS aquellas que se necesitan para resolver todas las referencias cruzadas de las diferentes instancias de los *drivers*.

4.3. Flujo de datos del sistema

El trabajo de nuestro sistema involucra muestreo de la imagen, su transmisión, almacenamiento y procesamiento, y el despliegue del resultado, tal como se muestra en la figura 4.4. La descripción de cada paso se expone a continuación.

(1) La información analógica de la imagen se obtiene de una videocámara. La salida de ésta, ya sea en formato NTSC o PAL, es transformada en señales digitales con formato YUV 4:2:2 por el chip decodificador de video TVP5146M2.

(2) La transmisión de los datos de la imagen entre el DSP y el chip decodificador

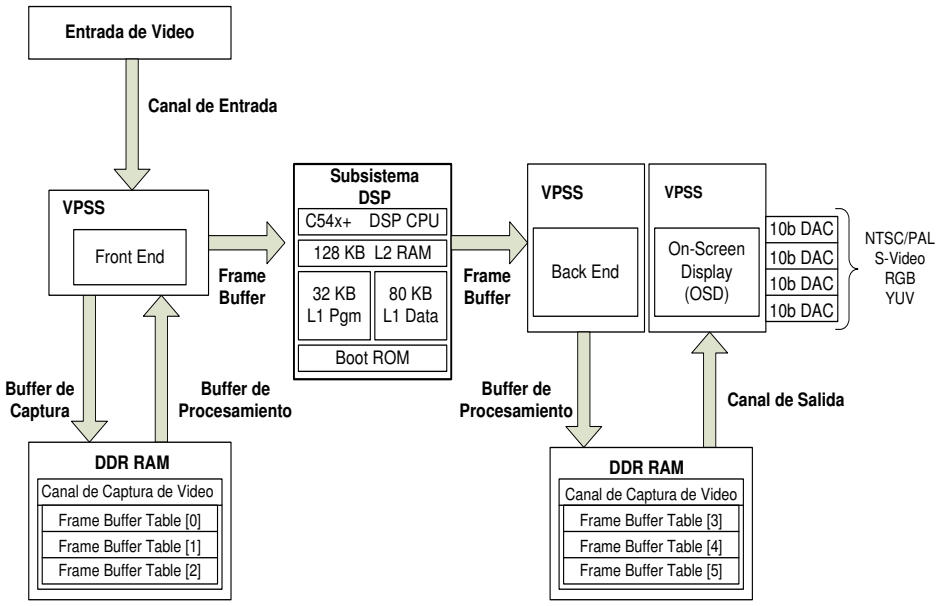

Figura 4.4: Flujo de datos de la aplicación.

de video se realiza por medio del la interfaz controladora de video CCD, que se encuentra en el módulo Front End del Subsistema de Procesamiento de Vídeo (VPSS) del DM6437. Este controlador habilita dos buffers o canales para la transmisión de los datos digitales de la imagen: el canal de captura y el canal de procesamiento. El VPPS a través de la interfaz CCD coloca por medio del canal de captura la información digital en la memoria DDR del EVM, esta información es puesta en 3 buffers continuamente. Esto se hace con el fin de que exista un flujo continuo de información entre la que se adquiere y la que es puesta en el canal de procesamiento, y no haya pérdida súbita de la misma. Cuando el subsistema DSP solicita información para procesar al VPSS, éste habilita el canal de procesamiento de la interfaz CCD, obtiene los datos del primer buffer que fue cargado en la memoria DDR y deposita la información en un buffer dentro del Switch Central de Recursos llamado FrameBuffer y de ahí es enviada al subsistema DSP para su correspondiente procesamiento. Este procedimiento se repite con los otros dos buffers cargados en la memoria DDR y nuevamente comienza el ciclo.

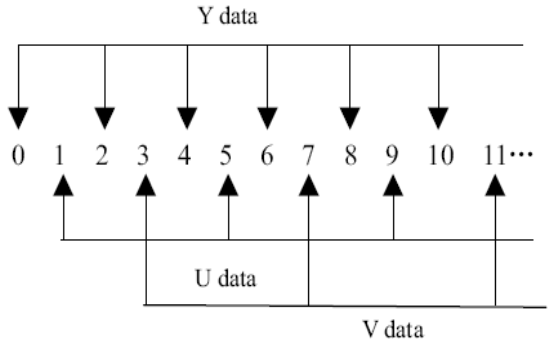

Figura 4.5: Orden de los datos YUV en una fila de la imagen.

(3) El tercer punto a mencionar es el almacenamiento de los datos de la imagen. Un frame con los datos de la imagen está dividido en campos pares y campos impares, la dirección de inicio del almacemamiento de la imagen en la memoria DDR es 0x80000000 y el formato de imagen de los datos, como se mencionó anteriormente es el YUV 4:2:2. El orden de almacenamiento se muestra en la figura 4.5 donde puede observarse que cualquier pixel tiene un dato Y, y dos pixeles adyacentes solamente comparten un grupo de datos UV. El tamaño de la imagen de acuerdo al estándar NTSC es de 480x720 pixels, por lo que el tamaño del dato en una fila es de 720x2bytes.

(4) El subsistema DSP es el encargado de implementar los algoritmos de procesamiento de imágenes. Se implementaron los algoritmos que se discutieron en el capítulo tres.

(5) El despliegue de resultados se logra al invalidar los datos de la memoria caché L2 del subsistema DSP. Esto produce que los datos procesados en el CPU C64x+ sean puestos en el FrameBuffer y retornados al VPSS. Los datos procesados son depositados en el módulo BackEnd, en específico en el Controlador de Despliegue en Pantalla (OSD). El OSD manda a los datos de nuevo a la memoria DDR y los coloca en otros tres buffers, tal y como lo hace el controlador CCD. Una vez ahí, el mismo OSD se encarga de tomarlos y enviarlos a los DACs para desplegar la información en forma analógica. El uso de los tres buffers de datos está justificado para evitar

una pérdida de información súbita cuando la imagen está siendo proyectada, y que se vería reflejada como un parpadeo.

4.4. Implementación

A diferencia de otras familias de DSP de TI, la configuración y coordinación de los distintos subsistemas presentes en el DM6437 hace necesario recurrir a la arquitectura de software mencionada en la sección 4.2, esto debido a la gran complejidad de los mismos y al enorme flujo de datos que el procesamiento digital de imágenes implica.

El primer paso para la implementación del sistema consiste, como se muestra en la fig. 4.7, en la inicialización de los archivos del DSP/BIOS como de los manejadores PSP. Dado que se trabaja en lenguaje **C**, esto se logra con los archivos de inclusión (include files) y que, como se mencionó anteriormente, son provistos por el fabricante. Estos archivos contienen las definiciones de las estructuras de datos que configuran y sincronizan los diferentes componentes de la plataforma.

Como segundo paso se declara e inicializa la sección de memoria llamada Frame-Buffer Table, que consta de seis buffers, tres que se servirán como canal de entrada y tres que servirán como canal de salida. En este mismo paso se declara el arreglo bidimensional que nos servirá para convertir el buffer unidimensional en una matriz con los valores de los pixeles de la imagen y el cual tiene la siguiente estructura:

```
typedef struct pixelContent {
unsigned char y;
unsigned char c;
} pixelContent;
```

El tercer paso consta de la creación de los canales de entrada y salida. Esto conlleva realizar dos procedimientos, se configura el controlador de video CCD que se encuentra en el Video Front End y el controlador del OSD que se encuentra

en el Video Back End. La configuración del controlador de video CCD se logra a través de la estructura de datos PSPVPFECcdcConfigParams [33] que se encuentra en el archivo psp_vpfe.h y que a su vez es un parámetro de una estructura aún mayor llamada PSPVPFEChannelParams, y tiene la estructura que se muestra a continuación.

```
static PSP_VPFECcdcConfigParams vpfeCcdcConfigParams = {
PSP_VPFE_CCDC_YCBCR_8, /* dataFlow (input mode) */
PSP_VPSS_FRAME_MODE, /* ffMode */
480, /* height */
720, /* width */
(720 *2), /* pitch */
0, /* horzStartPix */
0, /* vertStartPix */
NULL, /* appCallback */
{
PSP_VPBE_TVP5146_Open, /* extVD Fxn */
PSP_VPFE_TVP5146_Close,
PSP_VPFE_TVP5146_Control,
},
0 /*segId */
};
```

La descripción de cada uno de los parámetros mencionados se encuentra en la tabla 4.2, la cual contiene 8 parámetros a configurar: dataflow, ffMode,height, width, pitch, horzStartPix, vertStartPix, appCallBack. La configuración del OSD se realiza de la misma manera que el controlador CCD, es decir, a través de una estructura de control llamada PSP_VPBEOsdConfingParams [34], estructura que se encuentra en el archivo psp_vpbe.h y contiene los elementos que se muestran a continuación y su descripción se puede observar en la tabla 4.3.

```
static PSP_VPBEOsdConfigParams vpbeOsdConfigParams = {
PSP_VPSS_FRAME_MODE, /* ffMode */
PSP_VPSS_BITS16, /* bitsPerpixel */
PSP_VPSS_YCbCr422, /* colorMode */
(720 *(16/8u)), /* pitch */
0, /* leftMargin */
0, /* topMargin */
720, /* width */
480, /* height */
0, /*segId */
PSP_VPBE_ZOOM_IDENTITY, /* hscaling */
PSP_VPBE_ZOOM_IDENTITY, /* vscaling */
PSP_VPBE_EXP_IDENTITY, /* hExpansion */
PSP_VPBE_EXP_IDENTITY, /* vExpansion */
NULL /* appCallback */
};
```

La figura 4.6 muestra los principales valores asociados a la imagen y que son configurados a través de las estructuras antes mencionadas.

Como cuarto paso se tiene la configuración del chip decodificador de video. Tiene tres parámetros a configurar: format, mode, y su habilitación.

El quinto paso consiste en colocar en la memoria DDR el contenido del Frame-Buffer Table. Por medio de un apuntador al FrameBuffer, se obtienen los datos que van a ser dirigidos al subsistema de procesamiento para ser procesados de acuerdo al algoritmo seleccionado. Esto corresponde al sexto paso.

Como séptimo paso, una vez obtenidos los datos del FrameBuffer se pasan al arreglo bidimensional para que tomen forma de matriz con los valores de los pixeles. Una vez en el arreglo, se implementan en ellos los algoritmos seleccionados como octavo paso.

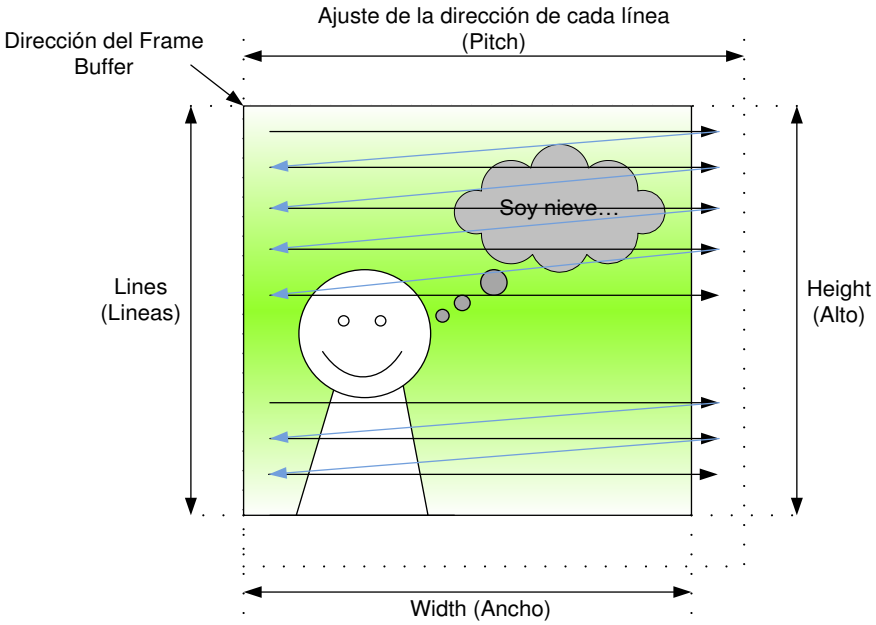

Figura 4.6: Principales valores configurados en la imagen.

Noveno paso: una vez realizado el procesamiento de los datos, estos son enviados de vuelta del arreglo bidimensional al FrameBuffer al invalidar la memoria cache.

Una vez en el frame buffer los datos son enviados al canal de salida, como se explicó en la sección de flujo de datos. Una vez ahí son enviados al convertidor digital-analógico para su posterior despliegue. Esto se realiza como un ciclo infinito en el DSP.

4.5. Resultados

El Módulo de Evaluación DM6437 es un sistema que, como su nombre lo indica, permite al programador medir el desempeño de esta plataforma tanto a nivel de su programación como del manejo de su hardware. El sistema fue probado con una cámara como entrada analógica de video, las imágenes fueron fueron adquiridas con una resolución de 480x720, como se mencionó anteriormente, de acuerdo al estándar

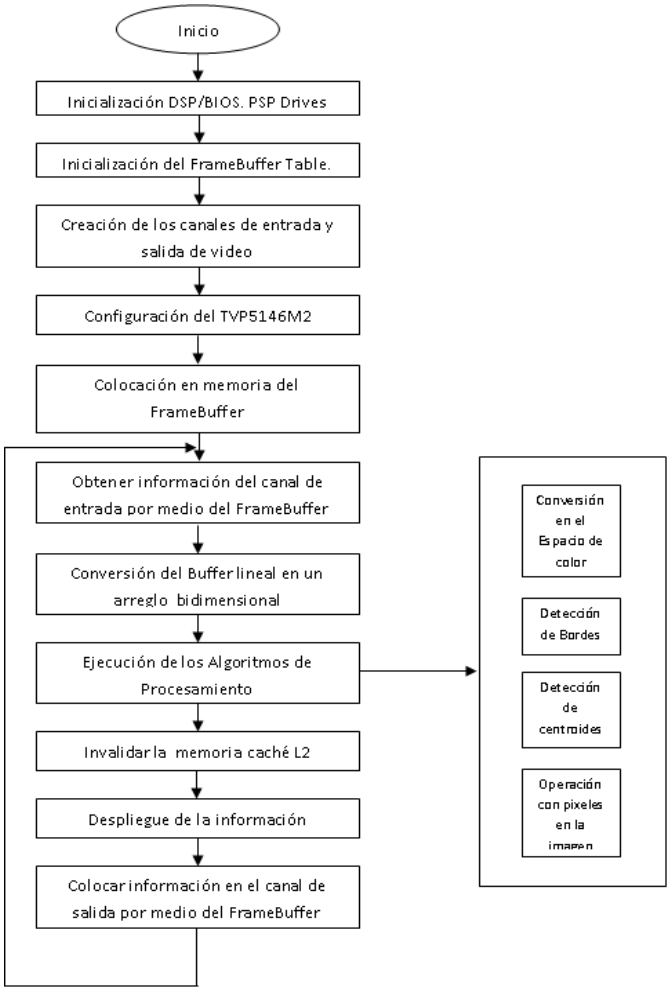

Figura 4.7: Diagrama de Flujo de la aplicación.

NTSC para América (en caso de utilizar el estándar PAL la resolución sería de 576x720), 16 bits/pixel y en formato YUV 4:2:2. La fig 4.7 muestra el diagrama de flujo para el desarrollo del software. De la fig 4.7 podemos ver que todos los procesos, excepto los algoritmos de procesamiento de imágenes, son los mismos. Los resultados de los algoritmos son los siguientes:

- Detección de bordes, fig 4.8. Uno de los más sencillos e importantes procesos es el de la detección de bordes. Sencillo por que los operadores de detección de bordes son máscaras de convolución, e importantes por que los bordes contienen mucha de la información de la imagen como es: posición del objeto,

Figura 4.8: Resultado de la detección de bordes.

forma, tamaño y en algunos casos texturas, un procedimiento de detección facilita la elaboración de las fronteras de los objetos.

Conversión de color a blanco/negro, fig 4.9. Este algoritmo permite obtener los componentes de en escala de grises en la imagen. Esto facilita el manejo de la información y acelera el procesamiento de los algoritmos ya que reduce la información de color contenida en 3 matrices, una para cada componente R,G y B, a una sola matriz.

- Diferenciación, binarización y detección de centroides, fig 4.10. El proceso de diferenciación, como se mencionó en el capítulo 3, se usa para eliminar los elementos estáticos de un ambiente en movimiento, dando como resultado que se conserven los elementos en movimiento mas ruido. La binarización permite eliminar el ruido presente en la imagen diferenciada.

- Detección de marcas por umbralado, fig 4.11. Una de las maneras más económicas para detectar marcas en una imagen es asociar un determinado valor de

Figura 4.9: Resultado de la conversión de color a blanco/negro

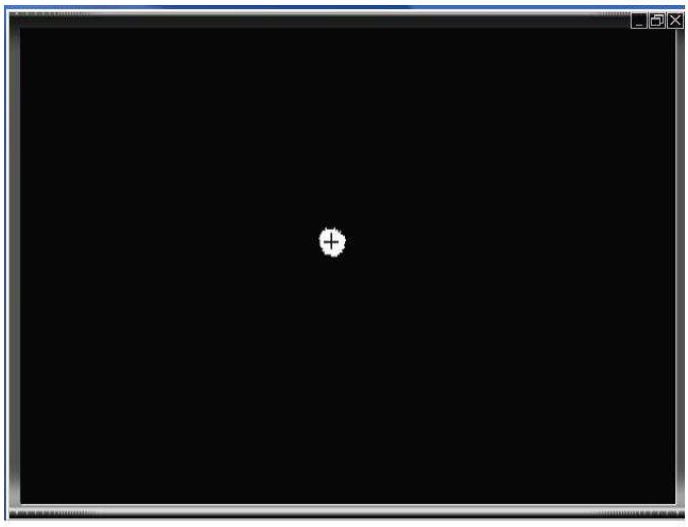

Figura 4.10: Resultado del cálculo de centroides

Figura 4.11: Resultado de la detección de marcas por umbralado

umbral a esas marcas y empezar a buscar pixeles con valores diferentes al marcado por el umbral y agruparlos para poder determinar la posición de esos pixeles en el plano de la imagen.

- Una vez obtenidos los resultados como se muestran en la tabla 4.4, es necesario obtener una medida del desempeño del DSP en cuanto a su velocidad de procesamiento. Para evaluar tal desempeño se realizó una comparación, para ello se seleccionó una computadora con las siguientes características: procesador Intel Core 2 Duo corriendo a 2.83 GHz, 1.95 Gb de memoria RAM en un bus de datos PCI a 600 Mhz y 32 bits, con Windows XP como sistema operativo y los algoritmos programados en C/C++. La razón de escoger un equipo como el que se describió anteriormente se debe a que son equipos que se están usando comúnmente en el mercado y son la alternativa de hardware de procesamiento para los algoritmos que se han programado en esta tesis. Las características de las imágenes a procesar en la computadora son las siguientes: resolución de 480 x 720, las imágenes en blanco y negro se presentan en escala de 256 tonos de gris, y las imágenes a color en formato YUV.

La tabla de comparación 4.5, nos muestra el resultado de desempeño en tiempo de los algoritmos para operaciones en espacio de color. Como puede apreciarse el DSP consume poco más del doble de lo que la computadora personal requiere. Estos resultados pueden explicarse desde el punto de vista del hardware que realiza el proceso, al tener la computadora un núcleo doble y un bus de datos más rápido que el DSP, y al contar con mayor memoria caché interna, el resultado es que la computadora procesa los datos a mayor velocidad aún con las desventajas que el sistema operativo representa. Por otro lado desde el punto de vista del software se puede hacer una comparativa entre los esquemas de programación. El código que realiza la tarea es exactamente el mismo:

```
for (renglon=0; renglon<renglones; renglon++)
for(columna = 0; columna < columnas; columna++) {
imagenbw[renglon][columna].y = 0x80;
imagenbw[renglon][columna].c = imagen1[renglon][columna].c;
}
```

Sin embargo la forma en la que se obtienen la imagen y como están declaradas las funciones no es la misma. Mientras que para el la computadora se usa la siguiente función:

```
void leeArchivoImagen (char *nombreArchivo, char **bloqueImagen) {
ifstream archivo; ifstream::postype inicio, fin, tamanio;
archivo.open (nombreArchivo, ios::in | ios::binary);
archivo.seekg (0, ios::beg);
inicio = archivo.tellg();
archivo.seekg (0, ios::end);
fin = archivo.tellg();
tamanio = fin - inicio;
*bloqueImagen = new char [tamanio];
```

```
archivo.seekg (0, ios::beg);
archivo.read (*bloqueImagen, tamanio);
archivo.close(); }
```

Para el DSP se usa el siguiente método:

```
void copia_buffer_arreglo ( void* frameActual, int renglones, int
columnas, pixelContent arreglo[480][720]) {
int renglon = 0;
int columna = 0;
int consecutivo = 0;
for (renglon=0; renglon<renglones; renglon++)
for(columna = 0; columna < columnas; columna++) {
arreglo[renglon][columna].y = *(((unsigned char*)frameActual)+
consecutivo++);
arreglo[renglon][columna].c = *(((unsigned char*)frameActual)+
consecutivo++); }}
```

Puede verse que en el primer código se utilizan dos apuntadores, uno al archivo de la imagen y otro doble apuntador al segmento de memoria donde se va a guardar la imagen. También se aprecia que se usan funciones propias del lenguaje de programación para su uso dentro del compilador de la PC. Por otro lado se observa que en el método que se usa para obtener la imagen del DSP, se tienen dos apuntadores, uno apunta a la sección de memoria y otro al buffer de datos de donde se obtiene la imagen, además de usar dos ciclos "for" para obtener el arreglo de memoria donde serán guardados los datos de la imagen. Se puede apreciar por los valores de tiempo de ejecución, que las funciones que usa el lenguaje C/C++ están optimizadas y obtienen un buen rendimiento haciendo uso del potente hardware en el que son procesadas. En el DSP, a diferencia de la PC, se infiere que las operaciones de comparación en los ciclos "for", más el uso de apuntadores genera un conjunto de instrucciones que exigen un esfuerzo mayor al hardware y por lo tanto aumentan el tiempo de procesamiento.

La tabla 4.6, nos muestra el tiempo que consume para ambos sistemas la detección de marcas por umbralado. Como se puede deducir de la tabla de comparación 4.5, el DSP sigue presentando mayor tiempo en el procesamiento. Lo interesante es observar que para la computadora el nuevo algoritmo no representó un incremento muy significativo en el tiempo de procesado, pero para el el DSP se puede ver que sí lo fue. La explicación radica en que este nuevo proceso involucra comparaciones para el DSP y como puede apreciarse, estas operaciones a nivel de hardware empiezan a mostrar un incremento notable en el procesamiento. Las operaciones de comparación introducidas por los condicionales "if" generan un código más grande cuantitativamente hablando y exigen el uso de registros de comparación a nivel del hardware del DSP. Estas nuevas instrucciones van generando dependencias, es decir que las variables deben ser cargadas en los registros antes de ser comparadas y por la tanto ese tipo de dependencia inhibe el uso de la arquitectura para operaciones en paralelo que el DSP posee, también este tipo de instrucciones dificultan al compilador por si solo, determinar que accesos a la memoria son independientes, por ejemplo, una vez cargadas las variables saber cuales pueden permanecer en los registros sin ser movidas para ser nuevamente utilizadas y cuales no [59]. Este tipo de cuestiones es lo que va generando que no haya una proporción constante en el tiempo de ejecución de los algoritmos en el DSP. Con respecto a la computadora las funciones que usa para comparaciones, como se mencionó anteriormente, se encuentran optimizadas y los cambios en el tiempo de ejecución del programa no muestran incrementos muy desproporcionados. El código usado para obtener las marcas, en la PC y el DSP es el siguiente:

```
int renglon = 0; int columna = 0;
for (renglon=0; renglon<renglones; renglon++)
for(columna = 0; columna < columnas; columna++) {
```

```
if (puntoEncontrado(imagen[renglon][columna])) {
imagen[renglon][columna].y=0x80;
if(columna%2!=0)
imagen[renglon][columna].c=0x00;
else imagen[renglon][columna].c=0xFF; imagen[renglon][columna].y=0x80;
}}
int puntoEncontrado (pixelContent punto) {
if (punto.y < UMBRALDETECCION)
return 1; else return 0; }
```

Es de hacer notar en este algoritmo el uso y manejo de la memoria en el DSP. Mientras que en una computadora se puede acceder a mutiples valores localizados consecutivamente en la memoria, bajo este esquema de programación no se tiene la certeza de que los datos sean accesados de manera múltiple en la memoria del DSP. Es decir, si se pudieran acceder de manera múltiple se podrían almacenar datos de hasta 16 bits en la parte baja y alta de registros de 32 bits, de tal forma que cuando sea necesario acceder a esos datos se utilice una palabra de 32 bits, para leer los dos valores de 16 bits al mismo tiempo, este método es llamado procesamiento de paquetes de datos, el cual hace uso de las funciones intrínsecas descritas en [59]. El esquema actual de programación no permite el acceso directo a los registros de memoria del hardware por si mismo, pero el uso de las funciones mencionadas si lo permite. Sin embargo esta tarea de optimización esta más cercana al bajo nivel, que al esquema de programación en nivel medio-alto con el que este trabajo de tesis fue desarrollado. En la tabla 4.7, es posible observar la comparación entre ambas plataformas para la detección de bordes. Es interesante ver como se dispara el tiempo de procesamiento para el DSP; estas mediciones llevaron a tratar de entender que estaba pasando en el DSP cuando el procesamiento se estaba llevando acabo. La manera en como se ha programado el DSP es codificando el algoritmo en ANSI C, esto permite tener un código portable para poderlo migrar en el momento en que una plataforma DSP más moderna aparezca, lo cual es uno de los objetivos expuestos en el capítulo uno de esta tesis. Sin embargo, la tarea del compilador que convierte de

lenguaje C a lenguaje Ensamblador, es en este punto crítica, ya que debe permitir el menor número de instrucciones para que el procesamiento sea más rápido y por lo resultados observados, puede verse que, por si solo el compilador no es capaz de dar el conjunto óptimo de instrucciones.

La forma de implementar este filtro es la siguiente:

```
for (renglon=0; renglon< renglones-4; renglon++){
for(columna=0; columna<columnas-4; columna++){
imagenbord[renglon+1][columna+1].y=0;
for (s=(-a+2); s<(a+2); a++){
for(t=(-b+2);t<(b+2);b++){
imagenbord[renglon+1][columna+1].y=
imagenbord[renglon+1][columna+1].y+
(mascara[s][t]*imagen1[renglon+s][columna+t].y);
imagenbord[renglon+1][columna+1].c=imagen1[renglon+1][columna+1].c;
}}}}
```

Como se puede apreciar del código para este procesamiento los ciclos "for" incrementan las operaciones de comparación que el DSP debe realizar en su estructura de hardware. Como se vio en el fragmento de código referente a la obtención de la imagen, se trabaja con apuntadores para obtener la información del buffer de datos del DSP. De la misma manera, con un apuntador se devuelven los datos procesados al buffer, como se muestra en el código siguiente:

```
void copia_arreglo_buffer (void*frameActual, int renglones, int
columnas, pixelContent arreglo[480][720]) {
int renglon = 0;
int columna = 0;
int consecutivo = 0;
for (renglon=0; renglon<renglones; renglon++)
```

```
for(columna = 0; columna < columnas; columna++){
*(((unsigned char*)frameActual)+ consecutivo++) =
arreglo[renglon][columna].y;
*(((unsigned char*)frameActual)+ consecutivo++) =
arreglo[renglon][columna].c; }}
```

Estos procesos se realizan continuamente mientras el algoritmo se esta ejecutando. De esto se infiere que al no indicarle de manera directa al compilador qué variables son independiente y cuáles no, se va creando un acumulación de registros con direcciones de memoria a las cuales se apunta para hacer la comparación y mover datos entre el buffer y el arreglo de memoria con el que se trabaja. Dado que el compilador no marca un error en la memoria, el algoritmo realiza su función, pero el impacto se ve reflejado en el tiempo que transcurre para llevarse a cabo. Es necesario por lo tanto entrar a una optimización de bajo nivel [59].

La tabla 4.8, muestra la comparativa para el algoritmo de detección de centroides. Se puede ver que existe una diferencia de tiempos entre las dos plataformas de procesamiento. Esto se debe a que la detección involucra operaciones de división, y el compilador no es capaz nuevamente de construir un conjunto de instrucciones mas eficiente para el aprovechamiento del hardware del DSP. Estos resultados ponen de manifiesto que las herramientas de compilación por sí solas no proveen la optimización necesaria debido a que esta, es muy dependiente del código particular que se esta tratando de desarrollar. El código que realiza la operación se muestra a continuación:

```
float xc1=0; float yc1=0;
int xc=0; int yc=0; int area=0;
int renglon = 0; int columna = 0;int i= 0;
for (renglon=0; renglon<renglones; renglon++)
for(columna=0; columna<columnas; columna++){
```

```
centroide[renglon][columna].y=imagenbinaria[renglon][columna].y;
centroide[renglon][columna].c=imagenbinaria[renglon][columna].c;}
for (renglon=0; renglon<renglones; renglon++)
for(columna = 0; columna < columnas; columna++) {
if ( centroide[renglon][columna].c == 0xFF)
{
xc= xc+columna;
yc= yc+renglon;
area=area+1;
} }
xc1= _roundf(xc/area);
yc1= _roundf(yc/area);
renglon= yc1;
columna= xc1;
```

Del código para la realización de la detección de centroides, se ve que las operaciones de división generan una cantidad significativa de instrucciones, que se suman a las de los ciclos "for" y a la instrucción condicional "if". Lo anteriormente mencionado acerca de la dependencia y el acumulamiento de registros también es aplicable a esta parte del código. El código genera un conjunto de instrucciones muy grande, con acumulaciones de registros y dependencias entre las variables que hace que los tiempos de ejecución se disparen.

La figura 4.12, muestra una comparativa entre el tiempo de ejecución versus la cantidad de cantidad aproximada de operaciones que se tiene que realizar por pixel para cada algoritmo, de acuerdo a los tiempos mostrados en las tablas:4.5, 4.6, 4.7, 4.8, puede apreciarse que el límite de tiempo considerado como aceptable para la aplicación de control de robots móviles es de 40 ms, para algoritmos con mucho mayor número de operaciones, de acuerdo a lo explicado en párrafos anteriores, el conjunto de instrucciones no optimizadas consume mucho más tiempo de ejecución en el DSP a diferencia de la computadora. A simple vista se puede considerar que en la etapa de procesamiento de imágenes la computadora representa una opción más

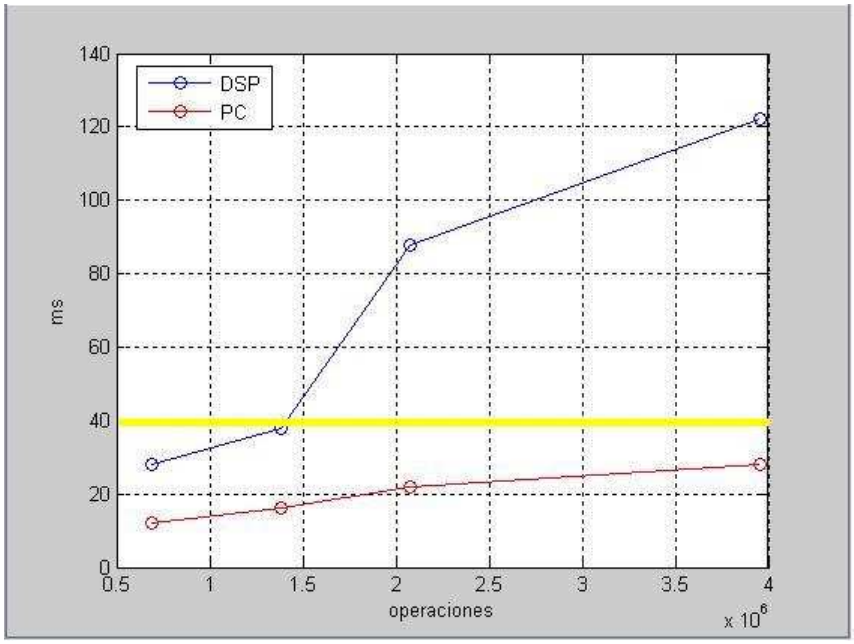

Figura 4.12: Comparación en tiempos DSP vs PC.

rápida y que consume menos tiempo. Sin embargo se debe tener en cuenta que la programación en el DSP puede optimizarse a partir de estructuras de bajo nivel, y por lo tanto el tiempo puede reducirse considerablemente con respecto al esquema de programación usado en este trabajo de tesis. Por lo tanto es necesario investigar formas de optimizar la programación y ejecución de los algoritmos en el DSP para que su tiempo de ejecución quede dentro del rango de los 40 ms aceptables.

Los resultados muestran que existe un conflicto entre el compromiso de mantener un código que sea portable y un código que sea eficiente. El lenguaje ANSI C nos permite obtener código portable de una plataforma a otra, pero queda muy limitado por el compilador, en cuestiones de darnos el mejor conjunto de instrucciones en lenguaje ensamblador, así como el mejor manejo de memoria para el DSP. Por otro lado un código escrito en ensamblador puede que sea más eficiente y nos permita un mejor manejo de memoria, pero queda anclado al hardware propio del DSP, lo que impide que ese código sea portable y fácil de entender. Por otro lado se puede observar que las diferencias en cuanto a potencia de hardware entre una y

otra plataforma son significativas, permitiendo un procesamiento mucho más rápido en la computadora personal a pesar de las desventajas que un sistema operativo multitareas representa.

Con respecto al hardware se puede mencionar que el DM6437 es una plataforma con capacidad de integración muy grande dada la cantidad de dispositivos periféricos que posee, es decir su arquitectura en hardware no está limitada solo al procesamiento de video. La capacidad de manejar un buen número de periféricos en un solo chip tiene como consecuencia que la velocidad de procesamiento se vea limitada, ya que se considera un caso crítico de disipación de potencia el hecho de que todos los periféricos trabajen casi simultáneamente y el procesador también trabaje a su velocidad máxima de procesamiento. En este sentido, con el hardware también existe un compromiso entre la capacidad de manejar una gran cantidad de periféricos en un mismo chip y la velocidad de procesamiento que de él se puede obtener. Es decir, existen DSP de la misma familia 64x que corren a velocidades de 1.2 GHz pero solamente son unidades de proceso, no tienen integrados tantos periféricos en un mismo chip, lo que hace que además de desarrollar un algoritmo se tenga que desarrollar los métodos de comunicación con los periféricos. Eso no sucede en el DM6437, ya que la arquitectura de software mencionada permite una rápida configuración y comunicación entre todos sus componentes.

En la búsqueda de esquemas de permitieran hacer más eficiente el desempeño del DSP, se mantuvo un intercambio de consultas e información con el fabricante, lo cual dio como resultado información acerca de opciones para mejorar el manejo de la memoria y la optimización de código que están presentes en el compilador, pero el uso de estas opciones solo permite trabajar con mayor número de fotogramas por segundo del DM6437 sin significar que pueda ser más rápido que una computadora personal.

En este capítulo se presentó la implementación de los algoritmos en el DSP, así como una comparativa de desempeño en función del tiempo de procesamiento con respecto a una computadora personal de reciente salida al mercado. Los resultados

de esta comparación permitieron establecer las limitantes que el esquema propuesto de programación tiene. Al no tener un acceso directo al hardware, y depender del compilador, no es posible tener la optimización requerida para un procesamiento capaz de competir con una computadora personal.

Tabla 4.1: Descripción Técnica del DSP.

Tecnología	Metal Process (CMOS)
Area	$15mm\,x\,15mm$
Consumo de Energía	1.2 V, 524 mA @ 600 Mhz
Velocidad Procesamiento	600MHz Max.
Instrucciones por ciclo de reloj	$1.67ns$
Instrucciones de 32bits por ciclo de reloj	8
MIPS	4800
Multiplicaciones Acumuladas por ciclo	2400 Millones
Registros de propósito general de 32bits	64
Multiplicadores de 32 bits	2
ALUs	6
Memoria Interna	-Caché L1 (110 Kbytes). -Caché L2 (128 Kbytes)

Tabla 4.2: Descripción de los parámetros de PSPVPFECcdcConfigParams.

Parámetro	**Descripción**
inpmode	Especifica la entrada ya sea formato RAW o formato YCbCr. YCbCr para el caso de nuestra aplicación.
ffMode	Del inglés FILED/FRAME Mode. Esto hace referencia al modo Progresive o Frame del formato de flujo de datos Interlaced. Frame Mode para la aplicación
height	Altura del frame. Es decir cuántas lineas verticales debe tener. 480 para la aplicación
width	Ancho del Frame. En número de pixeles en dirección horizontal.
pitch	Ajuste de la imagen
horzStartPix	Primer píxel horizontal
vertStartPix	Inicio de la línea vertical
appCallBack	Habilitación de la llamada de la función.
extVD Fxn	Interfaz decodificadora externa de video. Indica que tipo de interface se esta utilizando
segID	Sirve para indicar que pueden colocarse los parámetros en memoria cuando esta en 0.

Tabla 4.3: Descripción de los parámetros de PSP_VPBEOsdConfingParams.

Parámetro	**Descripción**
ffMode	Del inglés FILED/FRAME Mode. Esto hace referencia al modo Progresive o Frame del formato de flujo de datos Interlaced.
bitsPerpixel	El número de bits que conforman la información asociada a un pixel.
colorMode	Se refiere al formato de color de salida, ya sea en YCbCr o RGB888
pitch	Ajuste de la imagen.
leftMargin	Margen izquierdo de la imagen en la ventana OSD.
topMargin	Margen máximo del la ventana OSD hacia arriba
width	Ancho del Frame en el OSD. En número de pixeles en dirección horizontal
height	Altura del frame en el OSD. Es decir cuántas lineas verticales debe tener. 480 para la aplicación
seglD	Sirve para indicar que pueden colocarse los parámetros en memoria cuando está en 0.
hscaling	Escalamiento en dirección horizontal
vscaling	Escalamiento en dirección vertical
hExpansión	Expansión del tamaño del pixel en dirección horizontal.
vExpansión	Expansión del pixel en dirección vertical.
appCallBack	Habilitación de la llamada de la función.

Tabla 4.4: Datos de desempeño para los algoritmos seleccionados en el DSP.

Algoritmo	**Descripción**	**Tiempo de procesamiento @ 600mhz**
Operación en el Espacio de Color (conversión a blanco y negro)	480 x 720 Tamaño de la imagen de entrada. 256 niveles de gris.	28.4ms
Operación con pixeles en la imagen. (Detección de marcas por umbralado)	480 x 720 Tamaño de la imagen de entrada.	38 ms
Detección de bordes	480 x 720 Tamaño de la imagen de entrada. Filtros verticales y horizontales de Sobel sin supresión máxima.	122 ms
Cálculo de centroides	480 x 720 Tamaño de la imagen de entrada.	88 ms

Tabla 4.5: Comparación de desempeño para operaciones en espacio de color.

	DM6437	C/C++
Velocidad del reloj del procesador	600 MHz	2.83 GHz
Tiempo de procesamiento	28.4ms	12 ms

Tabla 4.6: Comparación de desempeño para operaciones con pixeles en la imagen.

	DM6437	C/C++
Velocidad del reloj del procesador	600 MHz	2.83 GHz
Tiempo de procesamiento	38 ms	16 ms

Tabla 4.7: Comparación de desempeño para detección de bordes

	DM6437	C/C++
Velocidad del reloj del procesador	600 MHz	2.83 GHz
Tiempo de procesamiento	122 ms	28 ms

Tabla 4.8: Comparación de desempeño para cálculo de centroides

	DM6437	C/C++
Velocidad del reloj del procesador	600 MHz	2.83 GHz
Tiempo de procesamiento	88 ms	22 ms

Capítulo 5

Conclusiones

A lo largo de este trabajo se presentó una plataforma de procesamiento digital de señales que permite ser reconfigurada de una manera flexible, la plataforma DSP TMS320DM6437 de Texas Instruments, esto como una opción que permite implementar algoritmos de procesamiento de imágenes para aplicaciones de control basado en visión de robots, donde la flexibilidad de programación de la plataforma, la velocidad de procesamiento, el tamaño, la disipación de potencia y el costo de la misma son restricciones de los robots móviles. En este trabajo también se describió la arquitectura y el software de la plataforma; se reportan los resultados de la implementación de algoritmos de procesamiento de imágenes de bajo nivel que, como se presenta en el capítulo tres, son candidatos naturales para aprovechar la arquitectura VLIW de los DSP.

De acuerdo a los resultados obtenidos en las comparaciones realizadas en el capítulo 4, se puede concluir con respecto a la arquitectura de software que el fabricante ofrece junto con la plataforma, que permite una rápida configuración y puesta a punto del hardware del DSP, lo que hace posible que los desarrolladores no necesiten conocer a fondo los detalles de la arquitectura del DSP. Todo se realiza a tráves de las estructuras que se describieron en la sección 4.4, lo anterior es con el fin de que el desarrollador se enfoque más en las tareas de diseño del algoritmo e implementación.

Las comparaciones que se llevaron a cabo entre un el DM6437 bajo la arquitectura de hardware y software mencionadas en el capítulo cuatro y la computadora personal nos llevan a la conclusión que, bajo ese esquema dependiente del compilador sin tomar completamente el mando de todo el hardware del DSP, las cifras no presentan en lo general nada fuera de lo común considerando que la computadora tiene un reloj casi 5 veces más rápido, dos núcleos en el procesador, más memoria caché interna y un bus de memoria externa mucho más rápido.

Las mayores contribuciones del trabajo se resumen en los siguiente puntos:

- Se evaluaron dos arquitecturas de procesamiento digital, los procesadores digitales de señales (DSPs por sus siglas en inglés) y los arreglos de compuerta de campo programable (FPGAs) con un criterio de selección basado en las ventajas que el hardware aporta al desarrollo de la aplicación y el esfuerzo de programación necesario para llevarla a cabo, como se explicó en el capítulo dos. Esto dió como resultado escoger una plataforma de procesamiento digital de señales ya que aunque las ventajas en hardware del FPGA son un poco más significativas, el esfuerzo de programación en un DSP es menor.

- De el conjunto de fabricantes y familias de DSPs se seleccionó la plataforma de Texas Instruments DM6437 por su capacidad de integración de periféricos y su capacidad para procesar imágen y vídeo además de otras características expuestas en los capítulos dos y cuatro de esta tesis.

- Se analizaron los diferente algoritmos de procesamiento de imágenes como se explica en el capítulo tres de la tesis. Se determinó que los algoritmos de bajo nivel, son los más adecuados para el aprovechamiento de la arquitectura del DM6437.

- Dadas las características del compilador proporcionado por el fabricante del DSP, se pudo generar un código en lenguaje ANSI C que es portable de un procesador a otro, bajo la arquitectura de software en la que se desarrolló.

- Bajo el esquema de software y hardware con el que se trabajó, se infiere que la propuesta de un producto como el DM6437 no es ser más potente que una computadora estándar, si no ser más eficiente, en particular en su menor consumo de energía y bajo costo de producción mientras siga teniendo un buen rendimiento en aplicaciones de vídeo.

El presente trabajo de tesis generó los siguientes productos:

- Estancia de investigación en la "Summer School on Image and Robotic" en el Centro de Investigación en Cómputo del IPN, junio-julio 2007
- Artículo para el Congreso Mexicano de Róbotica 2008: Implementación del Método de Manipulación en Espacio de Cámara en una plataforma DSP.
- Artículo para el Congreso Mexicano de Róbotica 2009: Plataforma de procesamiento digital de imágenes para aplicaciones de control de robots móviles.
- Poster para exposición de proyectos apoyados por el FAI: Control basado en visión de robots manipuladores usando una plataforma DSP.
- Poster para exposición de proyectos apoyados por el PIFI: Implentación de un método de manipulación en espacio de cámara en una plataforma DSP para control basado en vision de manipuladores móviles.

En términos generales se puede concluir, que bajo el esquema de software y hardware con el que se trabajó, los objetivos propuestos en este trabajo fueron cumplidos, al seleccionar una plataforma con una alta capacidad de integración de perifericos, bajo costo y manteniendo un buen rendimiento en procesamiento de imagen y vídeo, generando un código portable lo que la hace viable para implementar un control de robots móviles basado en visión en un solo DSP.

5.1. Trabajo a futuro.

Dado que este trabajo de tesis se realizó bajo un esquema software de nivel medio-alto, se propone que en el futuro se trabaje con lenguajes de bajo nivel como el ensamblador, con el fin de comprobar cuanto es la mejora en cuestión de tiempos de procesamiento sobre los tiempos que este trabajo de tesis reportó. También daría una mejor perspectiva para comparar con los tiempos generados por una computadora personal con las características aquí descritas.

Como una recomendación previa al trabajo en ensamblador, se considera necesario una buena experiencia en el manejo de dicho lenguaje y un conocimiento a detalle de la arquitectura del hardware, dado que los procesadores como el DM6437 son de alta complejidad y requiere de una cantidad significativa de tiempo de desarrollo. Todo lo anterior con el objeto de evitar colisiones de hardware, dañarían el dispositivo DSP.

También se propone que los algoritmos de procesamiento de imágenes que se implementen en la computadora personal se trabajen en C/C++ apoyados de las funciones de procesamiento de imágenes incluidas en las librerias IPP (Integrated Performance Primitives, por sus siglas en inglés) de Intel, las cuales son funciones diseñadas para obtener un desempeño máximo de la arquitectura de los procesadores Intel de doble núcleo. Esto con el objetivo de tener datos que nos permitan una medida más fiable de la potencia de los procesadores en cuestión del procesamiento de imagen. Si los resultados de tiempo con estas librerias son significativamente superiores a los del DSP aún siendo programado en ensamblador, quiere decir que en cuestiones de procesamiento de imagen los procesadores son superiores, dejando a los DSP como la mejor opción para sistemas embebidos, donde el costo de producción y el consumo de potencia son tan significativos como la velocidad de procesamiento de los datos.

Por último se propone la implementación de un sistema de control basado en visión de robots móviles, como una manera de comprobar la capacidad de integración de esta plataforma y medir su eficiente consumo de potencia.

Apéndice

Apéndice A

Código del DSP

Como se mencionó en el capítulo cuatro, el DSP fue programado en lenguaje C, en el entorno de desarrollo "Code Composer Studio V3.3" proporcionado por Texas Instruments. El código implementado para configurar el DSP y los algoritmos de procesamiento se muestra en las siguientes páginas.

```
/* * ======== videopreview.c ======== * */

/* runtime include files */

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdarg.h>

/* BIOS include files */

#include <std.h>
```

```
#include <gio.h>

#include <tsk.h>

#include <trc.h>

/* PSP include files */

#include <pspi2c.h>

#include <pspvpfe.h>

#include <pspvpbe.h>

#include <fvid.h>

#include <psptvp5146extVidDecoder.h>

#include <FVIDevmDM6437.h>

/* CSL include files */

#include <soc.h>

#include <cslrsysctl.h>

#define FRAMEBUFFCNT 6

/*Configuración del controlador de vídeo a través de las estructuras
de datos*/

static CSLSysctlRegsOvly sysModuleRegs = (CSLSysctlRegsOvly )CSLSYSOREGS;

static PSPVPFETVP5146ConfigParams tvp5146Params =

{

TRUE, /* enable656Sync */

PSPVPFETVP5146FORMATCOMPOSITE, /* format */
```

```
PSPVPFETVP5146MODEAUTO /* mode */
};
static PSPVPFECcdcConfigParams vpfeCcdcConfigParams =
{
PSPVPFECCDCYCBCR8, /* dataFlow */
PSPVPSSFRAMEMODE, /* ffMode */
480, /* height */
720, /* width */
(720 *2), /* pitch */
0, /* horzStartPix */
0, /* vertStartPix */
NULL, /* appCallback */
{
PSPVPFETVP5146Open, /* extVD Fxn */
PSPVPFETVP5146Close,
PSPVPFETVP5146Control, },
0 /*segId */
};
static PSPVPBEOsdConfigParams vpbeOsdConfigParams =
{
```

```
PSPVPSSFRAMEMODE, /* ffmode */

PSPVPSSBITS16, /* bitsPerPixel */

PSPVPBEYCbCr422, /* colorFormat */

(720 * (16/8u)), /* pitch */

0, /* leftMargin */

0, /* topMargin */

720, /* width */

480, /* height */

0, /* segId */

PSPVPBEZOOMIDENTITY, /* hScaling */

PSPVPBEZOOMIDENTITY, /* vScaling */

PSPVPBEEXPIDENTITY, /* hExpansion */

PSPVPBEEXPIDENTITY, /* vExpansion */

NULL /* appCallback */

};

static PSPVPBEVencConfigParams vpbeVencConfigParams =

{

PSPVPBEDISPLAYNTSCINTERLACEDCOMPOSITE /* Display Standard */

};

/*Declaración de apuntadores a los registros de memoria caché L2*/
```

```
volatile unsigned int * L2IBARregPtr = (unsigned int*) 0x01844018;

volatile unsigned int * L2IWCregPtr = (unsigned int*) 0x0184401C;

volatile unsigned int * L2WBARregPtr = (unsigned int*) 0x01844000;

volatile unsigned int * L2WWCregPtr = (unsigned int*) 0x01844004;

volatile unsigned int * L2WBregPtr = (unsigned int*) 0x01845000;

volatile unsigned int * L2INVregPtr = (unsigned int*) 0x01845008;

volatile unsigned int * L2WBINVregPtr = (unsigned int*) 0x01845004;

#define UMBRALDETECCION 110 // definicion del valor del umbral

typedef struct pixelContent { // Declaración de la estructura de memoria
para la imagen

unsigned char y;

unsigned char c; } pixelContent;

/*Declaración de los arreglos bidimensionales para las imágenes que
se van a trabajar en base a la

estructura de memoria */

pixelContent imagen1[480][720];

pixelContent imagen2[480][720];

pixelContent imagenDiferencia[480][720];

pixelContent imagenbw[480][720];

pixelContent imagenbinaria[480][720];

pixelContent centroide[480][720];
```

```
/*Declaración de los prototipos de funciones a utilizar

para el procesamiento*/

void copiabufferarreglo ( void* currentFrame, int yRows, int xPixels,
pixelContent

arreglo[480][720]);

void copiaarreglobuffer ( void* currentFrame, int yRows, int xPixels,
pixelContent

arreglo[480][720]);

void imagenbw (int renglones, int columnas, pixelContent imagen[480][720],
pixelContent

imagenbw[480][720]);

void binarizacion (int renglones, int columnas, pixelContent

imagenbw[480][720], pixelContent imagenbinaria[480][720]);

void calc_centroide(int renglones, int columnas,

pixelContent imagenbinaria[480][720],

pixelContent centroide[480][720]);

int puntoEncontrado (pixelContent punto);

void marcaPuntos (int renglones, int columnas,

pixelContent imagen[480][720]);

/* * ======== main ========

*/
```

```
void main()

{

printf("Video Preview Application\n");

fflush(stdout);

/* Workaround to BIOS/LOG issue: SDSCM00013785 */

TRCdisable(TRCLOGCLK);

/* VPSS PinMuxing */

/* CI10SEL - No CI[1:0] */

/* CI32SEL - No CI[3:2] */

/* CI54SEL - No CI[5:4] */

/* CI76SEL - No CI[7:6] */

/* CFLDSEL - No CFIELD */

/* CWENSEL - No CWEN */

/* HDVSEL - CCDC HD and VD enabled */

/* CCDCSEL - CCDC PCLK, YI[7:0] enabled */

/* AEAW - EMIFA full address mode */

/* VPBECKEN - VPBECLK enabled */

/* RGBSEL - No digital outputs */

/* CS3SEL - LCDOE/EMCS3 disabled */

/* CS4SEL - CS4/VSYNC enabled */
```

```
/* CS5SEL - CS5/HSYNC enabled */

/* VENCSEL - VCLK,YOUT[7:0],COUT[7:0] enabled */

/* AEM - 8bEMIF + 8bCCDC + 8 to 16bVENC */

sysModuleRegs -> PINMUX0 &= (0x005482A3u);

sysModuleRegs -> PINMUX0 |= (0x005482A3u);

/* PCIEN = 0: PINMUX1 - Bit 0 */

sysModuleRegs -> PINMUX1 &= (0xFFFFFFFEu);

sysModuleRegs -> VPSSCLKCTL = (0x18u);

return;

}

/* * ======== videopreview ======== */

void videopreview(void)

{

PSPVPSSSurfaceParams *frameBuffTable[FRAMEBUFFCNT];

PSPVPSSSurfaceParams *frameBuffPtr;

GIOHandle hGioVpfeCcdc;

GIOHandle hGioVpbeVid0;

GIOHandle hGioVpbeVenc;

int status = 0;

int done = 0;
```

```
int result;

int i;

/* init the frame buffer table */

for (i=0; i<FRAMEBUFFCNT; i++)

{

frameBuffTable[i] = NULL;

}

/* create video input channel */

if (status == 0)

{

PSPVPFEChannelParams vpfeChannelParams;

vpfeChannelParams.id = PSPVPFECCDC;

vpfeChannelParams.params = (PSPVPFECcdcConfigParams*)&vpfeCcdcConfigParams;


hGioVpfeCcdc = FVIDcreate("/VPFE0",IOMINOUT,NULL,&vpfeChannelParams,NULL);


status = (hGioVpfeCcdc == NULL ? -1 : 0);

}

/* create video output channel, plane 0 */

if (status == 0) {

PSPVPBEChannelParams vpbeChannelParams;
```

```
vpbeChannelParams.id = PSPVPBEVIDEO0;

vpbeChannelParams.params = (PSPVPBEOsdConfigParams*)&vpbeOsdConfigParams;


hGioVpbeVid0 = FVIDcreate("/VPBE0",IOMINOUT,NULL,&vpbeChannelParams,NULL);


status = (hGioVpbeVid0 == NULL ? -1 : 0);

}

/* create video output channel, venc */

if (status == 0)

{

PSPVPBEChannelParams vpbeChannelParams;

vpbeChannelParams.id = PSPVPBEVENC;

vpbeChannelParams.params = (PSPVPBEVencConfigParams *)&vpbeVencConfigParams;


hGioVpbeVenc = FVIDcreate("/VPBE0",IOMINOUT,NULL,&vpbeChannelParams,NULL);


status = (hGioVpbeVenc == NULL ? -1 : 0);

}

/* configure the TVP5146 video decoder */

if (status == 0)

{

result = FVIDcontrol(hGioVpfeCcdc, VPFEExtVDBASE+PSPVPSSEXTVIDEODECODERCONFIG,
```

```
&tvp5146Params);
   status = (result == IOMCOMPLETED ? 0 : -1);
   }
   /* allocate some frame buffers */
   if (status == 0) {
   for (i=0; i<FRAMEBUFFCNT && status == 0; i++)
   {
   result = FVIDalloc(hGioVpfeCcdc, &frameBuffTable[i]);
   status = (result == IOMCOMPLETED && frameBuffTable[i] != NULL ? 0 :
-1);
   }
   }
   /* prime up the video capture channel */
   if (status == 0)
   {
   FVIDqueue(hGioVpfeCcdc, frameBuffTable[0]);
   FVIDqueue(hGioVpfeCcdc, frameBuffTable[1]);
   FVIDqueue(hGioVpfeCcdc, frameBuffTable[2]);
   }
   /* prime up the video display channel */
   if (status == 0)
```

```
{ FVIDqueue(hGioVpbeVid0, frameBuffTable[3]);
FVIDqueue(hGioVpbeVid0, frameBuffTable[4]);
FVIDqueue(hGioVpbeVid0, frameBuffTable[5]);
}
/* grab first buffer from input queue */
if (status == 0)
{
FVIDdequeue(hGioVpfeCcdc, &frameBuffPtr);
}
/* loop forever performing video capture and display */
while (!done && status == 0)
{
/* grab a fresh video input frame */
FVIDexchange(hGioVpfeCcdc, &frameBuffPtr);
///// Here we can do operations on the video frame buffer...
copiabufferarreglo( (void*)(frameBuffPtr->frame.frameBufferPtr), 480,
720, imagen1);
imagenbw (480, 720,imagen1, imagenbw);
binarizacion(480, 720, imagenbw,imagenbinaria);
calc_centroide(480, 720, imagenbinaria, centroide);
copiaarreglobuffer( (void*)(frameBuffPtr->frame.frameBufferPtr), 480,
```

```
720, centroide);
   *L2WBINVregPtr = 0x1; //initiate a writeback/invalidate to all to the
L2 cache
   while( *L2WBINVregPtr != 0);
   /* display the video frame */
   FVIDexchange(hGioVpbeVid0, &frameBuffPtr);
   }
   }
   /****** funciones de manejo de memoria y procesamiento de imágenes******/
   void copiabufferarreglo ( void* frameActual, int renglones, int columnas,
pixelContent arreglo[480][720])
   {
   int renglon = 0; int columna = 0; int consecutivo = 0;
   for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
< columnas; columna++)
   {
   arreglo[renglon][columna].y = *(((unsigned char*)frameActual)+
   consecutivo++);
   arreglo[renglon][columna].c =*(((unsigned char*)frameActual)+
   consecutivo++);
   }
```

```
} // copiabufferarreglo()

void copiaarreglobuffer ( void* frameActual, int renglones, int columnas,
pixelContent

arreglo[480][720])

{

int renglon = 0; int columna = 0; int consecutivo = 0;

for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
< columnas; columna++)

{

*(((unsignedchar*)frameActual)+ consecutivo++) = arreglo[renglon][columna].y;


*(((unsigned char*)frameActual)+ consecutivo++) = arreglo[renglon][columna].c;


}

} // copiaarreglobuffer()

/*Detección de marcas por umbralado*/

void marcaPuntos (int renglones, int columnas, pixelContent imagen[480][720])

{

int renglon = 0; int columna = 0;

for (renglon=0; renglon<renglones; renglon++)

for(columna = 0; columna < columnas; columna++)

{
```

```
   if (puntoEncontrado(imagen[renglon][columna]))
   {
   imagen[renglon][columna].y=0x80;//valor original 255
   if(columna%2!=0)
   imagen[renglon][columna].c=0x00;//valor original 255
   else imagen[renglon][columna].c=0xFF; imagen[renglon][columna].y=0x80;

   }
   }
   }
   int puntoEncontrado (pixelContent punto)
   {
   if (punto.y < UMBRALDETECCION) //es Y en el orignal return 1; else return
0;
   }
   /* Conversion en el espacio de color*/
   void imagenbw (int renglones, int columnas, pixelContent
   imagen1[480][720],pixelContent imagenbw[480][720])
   {
   int renglon = 0; int columna = 0;
   for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
```

```
< columnas; columna++)
   {
   imagenbw[renglon][columna].y = 0x80; imagenbw[renglon][columna].c =
imagen1[renglon][columna].c;
   }
   }
   /*binarización*/
   void binarizacion (int renglones, int columnas, pixelContent
   imagenbw[480][720], pixelContent
   imagenbinaria[480][720])
   {
   int renglon = 0; int columna = 0;
   for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
< columnas; columna++)
   {
   if ( imagenbw[renglon][columna].c < UMBRALDETECCION)
   {
   imagenbinaria[renglon][columna].y= imagenbw[renglon][columna].y;
   imagenbinaria[renglon][columna].c = 0x00;
   }
   else
```

```
{
imagenbinaria[renglon][columna].y= imagenbw[renglon][columna].y;
imagenbinaria[renglon][columna].c = 0xFF;
}
}
}
/* Cálculo del centroide*/
void calc_centroide ( int renglones, int columnas, pixelContent
imagenbinaria[480][720],
pixelContent centroide[480][720])
{
float xc1=0;
float yc1=0;
int xc=0;
int yc=0;
int area=0;
int renglon = 0;
int columna = 0;
int i= 0;
for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
< columnas; columna++)
```

```
   {
   centroide[renglon][columna].y=imagenbinaria[renglon][columna].y;
   centroide[renglon][columna].c=imagenbinaria[renglon][columna].c;
   }
   for (renglon=0; renglon<renglones; renglon++) for(columna = 0; columna
< columnas; columna++)
   {
   if ( centroide[renglon][columna].c == 0xFF)
   {
   xc= xc+columna; yc= yc+renglon; area=area+1;
   }
   }
   xc1= roundf(xc/area);
   yc1= roundf(yc/area);
   renglon= yc1;
   columna= xc1;
   for (i=-10; i < 10; i++)
   { centroide[renglon+i][columna].c=0x00;
   }
   for (i=-10; i < 10; i++)
   {
```

```
centroide[renglon][columna+i].c=0x00;

}

}
```

Bibliografía

[1] R. L. Stevenson, B. G. Adams y H L. Jamieson, *Parallel implementation for iterative image restoration algorithms on a parallel DSP machine*, The Journal of VLSI Signal Processing. Kluwer Academic Publishers. Vol. 5, No. 2, 1993, pp. 261 262.

[2] P. Martí, R. Reig, M. Serra y J. Bover, *Prototipo de procesado de visión artificial para el control automático de impresión de tintas*, Dept. d'Electrònica i Telecomunicacions, Universitat de Vic., 2003.

[3] I. Ibáñez, M.A. Aguirre y J.N. Tombs , *Una plataforma de bajo coste para visión 3D para control de robots autónomos de soldaduras*, Escuela Superior de Ingenieros. Universidad de Sevilla, 2004.

[4] N. Balsero, D. Botero, J. Zuluaga y C.A. Parra, *Interacción hombre-máquina usando gestos manuales en texto real*, Ingeniería y Universidad, julio-diciembre año/vol. 9, numero 002, Pontificia Universidad Javeriana, Bogotá, Colombia, 2005.

[5] K. Shimizu y S. Hirai, *Realtime and Robust Motion Tracking by Matched Filter on CMOS+FPGA Vision System*, IEEE International Conference on Robotics and Automation Roma, Italy, 10-14 April, 2007.

[6] P. Egan, F. Lakestani, M.P. Whelan y M.J. Connelly, *Three-dimensional machine vision utilising optical coherence tomography with a direct read-out CMOS*

camera, Optical Communications Research Group, University of Limerick, Ireland; Photonics Sector, Institute for Health and Consumer Protection, European Commission Joint Research Centre, Ispra, Italy 2005, pp. 2230-2237.

[7] D.C. M. Bilsby, R.L. Walker y R. W. M. Smith, *Comparation of a Programmable DSP and FPGA for real-timemultiscale convolution*, IEE Colloquium on Volume, Issue, , Feb 1998. pp. 4/1 - 4/6.

[8] Z. Zhang, *A stereovision system for a planetary rover: Calibration, correlation, registration, and fusion*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems Human Robot Interaction and Cooperative Robots, IROS 96, April 1996.

[9] K. Konolige, *Small vision systems: Hardware and implementation*, Proceedings of Eight International Symposium on Robotics, ISR 97, Hayama, Japan, October 1997, pp. 111116.

[10] S. Mahlknecht, R. Oberhammer y G. Novak, *A Real-Time Image Recognition System for Tiny Autonomous Mobile Robots*, Springer Netherlands. Volumen 29, Números 2-3, Marzo del 2005, pp. 247261.

[11] J. Fainguelernt, G. Reith y R. Sikora, *An integrated environment for developing real-time DSP applications*, ICASSP 2008. IEEE International Conference March 31-April 4, 2008, pp. 2653- 2656.

[12] http://home.mit.bme.hu/~szedo/FPGA/fphahw.htm. Consultada en febrero del 2009.

[13] http://www.xilinx.com/. Consultada en agosto del 2007

[14] Altera Data Book, 2002

[15] R.H. Cortes, *Introduccion a los DSPs*, Universidad Técnica Federico Santa María, Departamento de Electrónica, Valparaiso Chile febrero 2004.

[16] http:/ / focus.ti.com/dsp/docs/dsphome. Consultada en agosto del 2007.

[17] www.freescale.com. Consultada en agosto del 2007.

[18] http:/ /www.analog.com/embedded-procesing-dsp/pocessors. Consultada en agosto del 2007.

[19] J.C. Paz, *Generación de Imágenes*, Publicaciones, Universidad de Ciudad Juárez, México, 2002.

[20] DSP Selection Guide, Texas Instrument, 2007.

[21] Texas Instruments Technical Group, *DSP Selection Guide C6000, Texas Instruments*, Texas Instruments, EUA, 2007.

[22] S.B. Skaar, W. H. Brockman y R. Hanson, *Camera Space Manipulation*, International Journal of Robotics Researchs, Vol. 6, No. 4, pp. 20-32.

[23] E.J. González Galván, *Notas sobre Teoría de Estimación aplicada a la Robótica*, Centro de Investigación y Estudios de Posgrado. Facultad de Ingeniería. Universidad Autónoma de San Luis Potosí.

[24] F.E. Martínez Pérez, *Interfaz Portable utilizando Internet y Tecnología Orientada a Aspectos para la definición de Tareas Robotizadas basadas en Visión Computacional*, Tesis de Maestría en Ingeniería de la Computación, 2005.

[25] K.S. Fu, R.C. González y C.S.G. Lee, *Robótica: Control, detección, visión e inteligencia*, McGraw Hill Primera Edición,1988.

[26] R.C. González R. y E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company. First Edition, 1992.

[27] J.H. Sossa Azuela, *Rasgos Descriptores para el Reconocimiento de Objetos*, Colección: Ciencia de la Computación.Editado por el IPN. Primera Edición, 2006.

[28] W. Hinrichs, P. Jens, L. Hanno. *A1.3 GOPS Parallel DSP for Hig Performance Image Processing Aplicattions*. IEEE Journal of Solid State Circuits, Vol., 35, No 7, Julio 2000.

[29] Spectrum Digital Technical Group, *DM6437 Technical Reference*, Spectrum Digital, EUA, 2007.

[30] Texas Instruments Technical Group, *TMS320DM6437 Digital Media Processor (Rev D)*, Texas Instruments, EUA, 2007.

[31] Texas Instruments Technical Group, *TMS320 DSP Algorithm Standard Rules and Guidelines*, Texas Instruments, EUA, 2007.

[32] Spectrum Digital Technical Group, *PSP Drivers User Guide,* Spectrum Digital, EUA, 2007.

[33] Spectrum Digital Technical Group, *DSP/BIOS VPFE User Guide,* Spectrum Digital, EUA, 2007.

[34] Spectrum Digital Technical Group, *DSP/BIOS VPBE User Guide,* Spectrum Digital, EUA, 2007.

[35] J. Li, K. Yuan y W. Zou, *A DSP Based Non-Vision Sensor Data Acquisition System for Autonomous Mobile Robot*, Hi-tech Innovation Center Institute of Automation Chinese Academy of Sciences Beijing, China Proceedings of the 6th World Congress on Intelligent Control and Automation, Dalian, China, June 21 - 23, 2006.

[36] M.G. Morrow, T.B. Welch y C.H.G. Wright, *A host port interface board to enhace the TMS320C6713 DSK*, ICASSP, 2006.

[37] J.D. Martínez, J.Toledo, R.Esteve, F.J. Mora, A.Sebastiá, J.M. Benlloch, M.Fernández, E.N. Giménez, M. Giménez, C.W. Lerche y N. Pavón, *Fully Digital Trigger and Pre-Processing Electronics for Planar Positron Emission Mammography (PEM) Scanners*, Department of Electronics Engineering, Universidad Politécnica de Valencia IEEE, 2004.

[38] V. Gemignani, F. Faita, M. Giannoni y A. Benassi, *A DSP-Based Platform for Rapid Prototyping of Real Time Image Processing Systems*, CNR, Pisa ITALY 3d International Symposium on Image and Signal Processing an Analysis, 2003.

[39] J. Proffitt, W. Hammond, S. Majewski, V. Popov, R.R. Raylman, A.G. Weisenberger y R. Wojcik, *A Flexible High-Rate USB2 Data Acquisition System for PET and SPECT Imaging*, Thomas Jefferson National Accelerator Facility, Newport News, VA Department of Radiology, Center for Advanced Imaging, West Virginia University, Morgantown, WV Nuclear Science Symposium Conference Record, IEEE, 2005.

[40] Z. Song, Q. Wang, X. D y Y. Wang, *A High Speed Digital Ultrasonic Flaw Detector Based on PC and USB*, Dept. of Control Science and Engineering, Harbin Institute of Technology, Harbin, P. R. China Instrumentation and Measurement Technology Conference - IMTC 2007 Warsaw, Poland, May 1-3, 2007.

[41] U. Muehlmam, M. Ribo, P. Lag y A. Pinz, *A New High Speed CMOS Camera for Real-Time Tracking Applications*, Institute of Electrical Measurement and Measurement Signal Processing and Christian Doppler Laboratory for Automotive Measurement Research Graz University of Technology, Austria 2004 IEEE International Conference on Robotics L Automation New Orleans, LA April 2004.

[42] Y. Zhu, B.R. Hayes-Gill, S.P. Morgan y N.C. Hoang,*An FPGA Based Generic Prototyping Platform employed in a CMOS Laser Doppler Blood Flow Camera*, IEEE FPT 2006.

[43] P. Cobos Arribas y F. Monasterio-Huelin Macia,*FPGA Board for Real Time Vision Development Systems*, Universidad Politecnica de Madrid 4fth IEEE International Caracas Conference on Device, Circuits and Systems.

[44] H. Lim, S-Y. Park y S-J. Kang y W-H. Cho,*FPGA Implementation of Image Watermarking Algorithm for a Digital Camera*, IEEE, 2003.

[45] C. Murphy, D. Lindquist, A.M. Rynning T. Cecil, S. Leavitt, M.L. Chang y F.W. Olin,*FLow-Cost Stereo Vision on an FPGA*, International Symposium on Field-Programmable Custom Computing Machines, 2007.

[46] A. Linares-Barranco, F. Gómez-Rodríguez, A. Jiménez Fernández. T. Delbrück y P. Lichtensteiner,*Using FPGA for visuo-motor control with a silicon retina and a humanoid robot*, IEEE, 2007.

[47] J.H. Park, D.R. Lee, D.K. Kim y J.W. Jeon, *Implementation of a Stand-alone Color Image Transfer Circuit Using the USB 2.0 Interface*, International Conference on Control, Automation and Systems, COEX, Seoul, Korea, Oct. 17-20, 2007.

[48] L.F. Rodríguez-Ramos, A. Alonso, F. Gago, J.V. Gigante, G. Herrera y T. Viera, *Adaptive optics real-time control using FPGA*, Institute of Astrophysics of the Canary Islands, España, IEEE, 2006.

[49] Q.L. Jiang, *Sistema de video vigilancia de múltiples canales en plataforma DSP+FPGA Embebida*, Quian Liang Jiang Productos Electronicos, China, No. 11, 2007.

[50] P. Ling, *Tecnología Digital Visual Talkback*, Productos Electronicos, China, No. 11, 2007.

[51] D.C. M. Bilsby, R.L. Walker, R. W. M. Smith, *Comparation of a Programmable DSP and FPGA for real-timemultiscale convolution*, High Performance Architectures for Real Time Imaging Processing. (Ref. No. 1998/197), IEE Colloquium on Volume, Issue, 12 Feb 1998, pp. 4/1 - 4/6.

[52] A. Cárdenas, B. Goodwine, S. Skaar y M. Seelinger,*Vision Based Control of a Mobile Base and OnBoard Arm*, The International Journal of Robotichs Research, Sage Publications, Vol. 22, No. 9, 2003, pp. 677-698.

[53] Spectrum Digital Technical Group, *TMS320C6416 DSK Technical Reference*, Spectrum Digital, EUA, 2007.

[54] Texas Instruments Technical Group, *TMS320C6424 Fixed Point Digital Signal Processor Rev (C)*, Texas Instruments, EUA, 2007.

[55] S. Skaar, G. Del Castillo, *Revisualizing Robotics: New DNA for Surviving a World of Cheap Labor*, DNA Press, Primera Edición, EUA, 2006.

[56] Texas Instruments Technical Group, *EDMA v2.0 to EDMA v3.0 (EDMA3) Migration Guide*, Texas Instruments, EUA, 2007.

[57] Texas Instruments Technical Group, *xDAIS-DM User Guide*, Texas Instruments, EUA, 2007.

[58] Texas Instruments Technical Group, *TMS320 DSP Algorithm Standard Rules API Reference*, Texas Instruments, EUA, 2007.

[59] Texas Instruments Technical Group, *TMS320C6000 Programmer's Guide*, Texas Instruments, EUA, 2007.