IDENTIFICACIÓN E INDIVIDUALIZACIÓN EN TIEMPO REAL DE ESTRELLAS PRESENTES EN IMÁGENES DE CÚMULOS ESTELARES

PROYECTO PARA OBTAR AL TÍTULO DE INGENIERO ELECTRÓNICO:

ADOLFO ALEXÁNDER CARDONA C.
LEONARDO FABIO GARCÍA R.

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA ELECTRÓNICA

PEREIRA

2016

IDENTIFICACIÓN E INDIVIDUALIZACIÓN EN TIEMPO REAL DE ESTRELLAS PRESENTES EN IMÁGENES DE CÚMULOS ESTELARES

PROYECTO PARA OBTAR AL TÍTULO DE INGENIERO ELECTRÓNICO:

ADOLFO ALEXÁNDER CARDONA C.
LEONARDO FABIO GARCÍA R.

Dirigido por:
Ing. Edwin Andres Quintero S

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

PROGRAMA DE INGENIERÍA ELECTRÓNICA

PEREIRA

2016

Agradecimientos

Primero que todo queremos agradecer a Dios quien nos proporciono los medios para hacer posible este sueño, el cual marca una gran etapa de nuestras vidas.

En segundo lugar agradecemos a nuestras familias que no solo nos apoyaron durante toda nuestra carrera sino también a lo largo de toda nuestra vida y ademas siempre se esmeraron por inculcarnos grandes valores.

En tercer lugar agradecemos a nuestro director de tesis el ingeniero Edwin Quintero, en el que siempre tuvimos un apoyo incondicional.

Y por ultimo a los profesores que nos acompañaron a lo largo de este camino y a las personas que por un motivo u otro influenciaron de cierto modo en el desarrollo de nuestra carrera.

Gracias a todos, y Dios los bendiga.

Adolfo Alexander Cardona, Leonardo Fabio Garcia

Dedicatoria

Este trabajo va dedicado a nuestras familias por siempre brindarnos su amor, enseñarnos los valores y principios en los que se forjaron los pilares de lo que somos y además por apoyarnos de forma incondicional a lo largo de las diferentes etapas de nuestras vidas.

Adolfo Alexander Cardona, Leonardo Fabio Garcia

Resumen

En esta tesis de investigación formativa se presenta una solución al problema de la identificación e individualización de estrellas en imágenes de cúmulos estelares en tiempo real, para la Universidad Tecnológica de Pereira y su Grupo de Investigación en Astroingeniería Alfa Orión, basándose en técnicas de procesamiento digital de imágenes, que son implementadas en la DSP BlackfinBF533 de Analog Devices.

Este documento se estructuró en cinco capítulos, donde en el primer capítulo se define el problema, se abordan el objetivo y objetivos específicos, junto con la justificación, finalizando con el estado del arte donde se demarcan los antecedentes históricos. En el segundo capítulo se explican los algoritmos implementados y a partir de que criterios se seleccionaron las diferente técnicas a usar. Posteriormente se mostraron los resultados obtenidos del sistema y por último se dan conclusiones, aportes y recomendaciones.

Para llevar a cabo este sistema se desarrolló un algoritmo basado en técnicas de procesamiento digital de imágenes, y se implementaron en la tarjeta DSP Blackfin BF533. El Grupo de Investigación Alfa Orión aportó las imágenes de los cúmulos estelares para el desarrollo y análisis de los resultados, las capturas fueron realizadas por medio del telescopio del observatorio de la Universidad Tecnológica de Pereira.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Preliminares

1.1. Definición del problema

La astronomía es el área de la ciencia encargada del estudio de los cuerpos celestes. Sin embargo esta presenta grandes desafíos ya que no es posible tener un contacto directo con el objeto de análisis. Por tal motivo es necesario aprovechar las tenues señales provenientes desde el espacio para comprender el comportamiento y la composición de dichos astros.

Entre los objetos más analizados por los grupos de investigación en astronomía se encuentran los cúmulos estelares. Estos objetos son claves, entre otras cosas, para comprender la formación de las galaxias. Sin embargo debido a la distancia que separa estos objetos del investigador, ellos aparecen generalmente en las imágenes como débiles manchas luminosas, lo cual dificulta enormemente la tarea de la individualización e identificación de cada una de las estrellas que componen el cúmulo objeto de interés. Esta situación dificulta en gran medida el estudio de los cúmulos estelares, lo cual impide a los investigadores conocer a fondo el comportamiento de los mismos; conceptos claves para la comprensión de la evolución estelar.

Así mismo la implementación de algoritmos para la identificación de estrellas en cúmulos estelares implica un alto costo computacional y un tiempo considerable, debido a la gran cantidad de estrellas que se encuentran en ellos y además a la gran cantidad de fotografías que se toman para el estudio de estos objetos.

Por otra parte son pocos los estudios que en el país se encaminan al procesamiento de imágenes y de señales astronómicas, por lo tanto se evidencia la necesidad de generar proyectos desde la ingeniería que traten estas temáticas, con el fin de iniciar los estudios de estas áreas en el país, específicamente en el observatorio de la Universidad Tecnológica de Pereira.

Considerando lo anterior se plantea el siguiente interrogante: ¿Es posible desarrollar un sistema capaz de identificar e individualizar en tiempo real estrellas presentes en imágenes de cúmulos estelares?

1.2. Justificación

Como bien es sabido, la investigación en astronomía en mayor parte se realiza a través de imágenes que se toman en diferentes telescopios ubicados alrededor del planeta. Es por esta razón que se evidencia la necesidad de utilizar técnicas de procesamiento digital de imágenes, tales como la implementación de algoritmos de reconocimiento de patrones, reducción de ruido, etc. Sin embargo, es claro que la implementación de este tipo de algoritmos de identificación e individualización de estrellas requiere de un hardware potente que trabaje en tiempo real, por lo cual como primera medida se descarta el emplear un computador de propósito general. Esto se debe a las múltiples tareas que este debe realizar simultáneamente durante su operación. Por ende es significativa la importancia que tiene un sistema de uso especifico, de alto rendimiento como lo es un sistema embebido.

Así mismo, las opciones de diseño en un sistema son influenciadas por el costo, rendimiento y requerimientos de este; es por esto que resulta de gran utilidad un procesador DSP que provee características muy especificas, como lo son permitir el procesamiento de señales de audio y vídeo en tiempo real. Esto hace que sea el sistema dedicado ideal para implementar algoritmos de procesamiento de imágenes que conllevan un gran peso computacional, lo que permitirá dar una solución rentable y rápida sin necesidad de costosos componentes externos, para el desarrollo del sistema que permita individualizar e identificar en tiempo real las estrellas presentes en imágenes de cúmulos estelares.

Por otra parte, actualmente en la Universidad Tecnológica de Pereira no se cuenta con un sistema capaz de identificar e individualizar estrellas presentes en imágenes de cúmulos estelares en tiempo real. Además, este trabajo se propone como el seguimiento al proceso de investigación astronómica que viene realizando el grupo Alfa Orión.

1.3. Objetivos general y específicos

1.3.1. Objetivo general

Desarrollar un sistema capaz de identificar e individualizar en tiempo real estrellas presentes en imágenes de cúmulos estelares.

1.3.2. Objetivos específicos

- Elaborar un estado del arte acerca de las técnicas propias del procesamiento de imágenes enfocada en la identificación de objetos.
- Recopilar información acerca del procesamiento digital de imágenes en tiempo real sobre la tarjeta ADSP BF-533.
- Implementar en la tarjeta ADSP BF-533, el algoritmo capaz de identificar e individualizar estrellas en imágenes de cúmulos estelar.
- Verificar los tiempos de respuesta del sistema, durante la etapa de identificación e individualización de estrellas, en imágenes de los cúmulos estelares establecidos como muestra.

1.4. Marco histórico y antecedentes

Diferentes algoritmos han sido propuestos para la tarea de la detección de estrellas. Un algoritmo que detecta un nivel de umbral óptimo es uno de los enfoques más usados para dicha labor. En este algoritmo todos los píxeles por encima de cierto nivel son considerados estrellas y todos los otros píxeles son considerados el fondo[1]. Dos principales inconvenientes se han detectado en la literatura respecto a la identificación de estrellas basándose en un método de umbral. El primer problema consiste en que la intensidad del fondo no siempre es constante, por tal motivo, en una parte de la imagen se podría dar una detección correcta, mientras en otra, los píxeles del fondo podrían estar por encima del nivel encontrado como óptimo y dar una identificación errónea. Como segundo problema, en las imágenes astronómicas siempre hay cierta cantidad de ruido presente, que podría ser identificado como estrella[2]. Por esta razón han nacido nuevas técnicas para el procesamiento digital de imágenes, entre las cuales se encuentra la detección de bordes, técnica que ha sido parte esencial de muchos sistemas de visión por computador, gracias a que este proceso sirve para simplificar el análisis de imágenes, reduciendo drásticamente la cantidad de datos a procesar, mientras al mismo tiempo conserva información estructural útil acerca del objeto[3].

Las tareas de segmentación no suelen dar un resultado exacto de la delimitación de los objetos o regiones de interés. Aparecen píxeles mal clasificados, bordes imprecisos de los objetos o regiones que están solapadas. Por tanto, antes de extraer más características de medio nivel se requiere de una etapa de post-procesamiento. En esta fase se suele emplear el tratamiento morfológico. Esta es una técnica de procesamiento no lineal de la señal, caracterizada en realzar la geometría y forma de los objetos. Su fundamento matemático se basa en la teoría de conjunto. Aunque en un principio se aplicará sobre imágenes binarizadas, luego se extenderá a imágenes en niveles de grises. Este uso a niveles de grises permitirá vislumbrar que el procesamiento morfológico también se puede utilizar como técnica de procesado de la señal. Concluyendo, estas nuevas herramientas se pueden emplear tanto en el procesado, como en las etapas de segmentación pos procesado o en fases de mayor nivel de información visual. Actualmente, se pueden encontrar aplicaciones en la restauración de imágenes, en la detección de bordes, en el análisis de texturas, en el aumento del contraste y hasta en la compresión de imágenes[4].

Hasta recientemente los software de identificación de estrellas estaban enfocados en la reducción de carga computacional y de memoria, tanto como fuera posible, porque el rendimiento electrónico era bajo. En los últimos años esta necesidad se ha reducido por los avances que ha tenido el procesamiento en hardware. Otras características han adquirido más importancia como la efectividad y confiabilidad de los algoritmos de detección.

En el año 2007 en [5] se realizó un enfoque analítico no recursivo con técnicas enfocadas a la determinación del centroide de una estrella. Este método está basado en la aplicación de la campana de Gauss o distribución de Gauss, la cual permite

por medio de información de píxeles vecinos, estimar la localización del centro como una función de la relación de las intensidades. Para poner a prueba este método, se adquirieron 50 imágenes del cielo nocturno, en alrededor de una hora, con un tiempo aproximado de exposición para cada imagen de 0,05 s. Donde se halló el rendimiento estático del centroide de una estrella patrón, estimando la desviación del centro en las diferentes imágenes. En esta se aplicó una solución cuadrática (con $x = 0{,}25$) la cual tiene una precisión de 0,012 de un ancho de píxel. Dando como resultados una separación máxima del centroide entre imágenes de $3{,}5 * 10^{-5} rad$. Los resultados experimentales indican que la estimación del centroide se puede generar con gran precisión con un poco de calibración empírica. Además, la técnica es adaptable a una gran variedad de cámaras CCD que pueden ser usadas en diferentes aplicaciones. La teoría desarrollada ofrece una base para el cálculo de desviación típica del centroide de una estrella, además de proporcionar fundamentos y procedimientos prácticos para la calibración de cámaras estelares en el laboratorio y en observatorios astronómicos.

En el año 2008 en [2] se desarrolla un novedoso método basado en la umbralización de imágenes, mejorando el histograma combinándolo con técnicas que tienen como objetivo buscar dentro de objetos difusos. Inicialmente se crea una matriz análoga donde cada celda guarda un "1" si el correspondiente píxel es etiquetado como estrella, luego se crea otra matriz donde cada columna representa un identificador del objeto y cada fila representa datos relevantes como cantidad de píxeles que conforman el objeto y coordenada espacial en la imagen. Se crea además una variable que almacena la cantidad de objetos identificados, concluyendo con marcar con otro color sobre la imagen original los objetos detectados. Las simulaciones fueron realizadas sobre 4 tipos de imágenes (ideales creadas en Paint, semi-ideales obtenidas desde alguna aplicación con tratamientos ya establecidos, imágenes en escala de grises, y por último imágenes RGB). Fueron inyectados también cuatro tipos de ruidos diferentes (El Gaussiano, Poisson, Sal y Pimienta, y el moteado o con puntos). Fueron comparados con el método de Otsu y el de Tsai, además con programas disponibles en la red como el de Misao y GCX, revelando mejores resultados respecto a la detección de estrellas además que es robusto frente al ruido, aunque reportó tiempos más altos de ejecución. Este método demostró ser adecuado para la identificación de estrellas, cancelación de ruido, y mejoramiento de imágenes borrosas, además es implementado por bloques lo que hace que sea escalable y de fácil mejoramiento y entendimiento.

En el año 2008 en [6] se implementó un algoritmo rápido de adquisición de estrellas, que incluye tanto el análisis de sus características, como de conectividad, además de realizar segmentación y el método del centroide. En orden de obtener un objeto estrella correctamente, inicialmente es necesario calcular la magnitud y el centroide de la estrella. Este proceso afecta directamente la efectividad en el reconocimiento y la eficiencia computacional. Para realizar la segmentación se basaron en hallar un apropiado umbral tanto para el objeto como para el fondo, para luego determinar el centroide realizando un análisis conectivo del sector objetivo. Otro tipo de enfoque para realizar la segmentación se basa en la extracción de bordes. Entre este tipo de

segmentación se encuentra principalmente los métodos de Prewitt, Sobel, Roberts, entre otros. En general de los métodos de segmentación se puede destacar que los basados en un umbral son adecuados para extraer objetos de gran tamaño. Mientras los métodos basados en la extracción de bordes son adecuados para objetos más pequeños. Los autores, para hacer el análisis de sus métodos, tomaron como muestra una imagen compuesta por 5 objetos, donde determinaron que 3 de ellos eran estrellas, otro parece ser un objeto de gran tamaño como lo podría ser un cometa o una nebulosa, mientras el otro objeto es un punto aislado el cual tiene una tenue energía. Este algoritmo no solo demostró ser rápido sino que también fue eficiente, siendo este robusto a la ahora de extraer las características de los objetos detectados.

En el 2009 en [7] se plantea el diseño de un algoritmo para la identificación de estrellas en imágenes astronómicas. Este nace de la necesidad de realizar procesamiento digital de imágenes de gran tamaño en tiempo real. Para realizar dicha labor se utiliza la teoría de distribución Gaussiana, la cual permite hallar los diferentes valores de umbral de cada píxel perteneciente a una vecindad. De esta forma se podrá determinar en donde se encuentra el píxel con mayor intensidad, y así determinar el centro del objeto. El investigador plantea tres algoritmos para realizar dicha labor. Inicialmente, propone un esquema de escaneo, donde se recorre toda la imagen en busca de estrellas, pero este es poco eficiente debido a que recorre zonas donde no se detecta ningún tipo de intensidad. Por tal motivo propone la implementación de un segundo algoritmo, al que llama acelerador, debido a que permite omitir parte del recorrido en zonas difusas. Finalmente, propone un algoritmo excavador que permite encontrar estrellas más tenues. El análisis de estos algoritmos se realizó sobre dos imágenes astronómicas de $2048 * 2048 * 16$ bits, donde inicialmente se comparó el esquema de escaneo y el acelerador, dando como resultado que el acelerador mejoró los tiempos de respuesta en alrededor de 19 veces. Como segunda medida se comparó el algoritmo acelerador con un algoritmo de morfología propuesto en [8], donde se verifica una mayor efectividad en la detección de estrellas con menores tiempos de respuesta. Este diseño demostró ser eficaz y eficiente en la detección de estrellas en imágenes de gran tamaño como lo son las imágenes astronómicas, siendo muy útil en muchas áreas de la ingeniería tales como la segmentación gráfica, el reconocimiento de objetos irregulares, etc.

La observación de las estrellas es ampliamente utilizada para la determinación de posición de naves espaciales. En el año 2011 en [9] se implementó un proceso de determinación de posición que constaba de cuatro pasos. Primero, por medio de una cámara CCD se toman las imágenes de las estrellas de una localización arbitraria. Segundo, un microprocesador opera un algoritmo de procesamiento de imágenes sobre las fotografías de las estrellas mejorando la calidad de las mismas. Tercero, las características de las estrellas son extraídas de acuerdo a la estrategia de coincidencia, además estas son almacenadas en una base de datos, para luego ser comparadas con otra base de datos previamente almacenada. Finalmente, la posición y ángulo de las estrellas en la imagen son calculados para estimar el sistema de control de posición. En la etapa de pre-procesado se aplica un filtro de Gauss para la cancelación de ruido. En la etapa siguiente se calcula el gradiente de la imagen usando

un método de detección de bordes difuso, y solo entonces se calcula el centroide del histograma de los bordes. Este paso es quizá el más crucial, porque la efectividad de la identificación y el tiempo consumido son los parámetros más importantes en el rendimiento del algoritmo. Las simulaciones fueron realizadas sobre 50 imágenes de $128 * 128$ píxeles en el entorno de Matlab. Se comparó con el algoritmo de tipo triángulo dando como resultado una mejora en la fiabilidad del reconocimiento pasando del 91,3 % al 96,4 %, aunque afectó el tiempo de reconocimiento pasando de 1525 ms a 1756 ms. Este método es adecuado para la identificación de estrellas, cancelación de ruido, y mejoramiento de imágenes borrosas.

En el año 2010 en [10] se reportaron los resultados obtenidos de los recientemente instalados GPU en Alemania y China usando las tarjetas NVIDIA C870 y C1070 respectivamente. Desde septiembre de 2008 se instaló KOLOB en la Universidad Heidelberg de Alemania, contando con 40 nodos, cada nodo con características de tarjeta NVIDIA C870. El instituto realiza simulaciones numéricas en formación de estrellas y evolución de cúmulos estelares con este grupo de GPU. Un nuevo GPU está ahora instalado en China desde febrero de 2010 y este cuenta con 85 nodos, cada uno con 2 NVIDIA tesla C1070, que la ubica entre los 5 supercomputadores más rápidos del mundo. El código fue desarrollado en C++. Sobre este algoritmo se realizó una evaluación comparativa con código de simulación de N-Body, usando un esquema de alto orden de integración Hermite. Para la evaluación comparativa se utilizaron 164 GPU. El mayor rendimiento se alcanzó con 51,2 $Tflop/s$. De estos resultados se concluye que se tiene una velocidad sostenida por tarjeta de 312 $Gflop/s$. Simultáneamente en Alemania el grupo Kolob con 40 tarjeta NVIDIA C870 obtiene una velocidad de 6,5 $Tflop/s$, es decir 162,5 $Gflop/s$ por cada tarjeta. Este grupo de GPU ha demostrado ser muy adecuado para aplicaciones de análisis de formación estelar y evolución de cúmulos globulares, además puede ser utilizado en simulaciones de densos sistemas gravitacionales como lo son los N-Body. En el futuro el grupo de GPU será utilizada en una gama más amplia de aplicaciones, incluyendo FFT, SPH, y procesamiento de datos.

En el año 2012 en [11] se propone la implementación de un algoritmo en un Hardware con una arquitectura tipo pipeline, para la detección de estrellas y el cálculo del centroide. La arquitectura contiene varias unidades, incluyendo (1) unidad de filtrado, (2) unidad de etiquetado y de la memoria, (3) la unidad de control, y (4) Unidad de cálculo de los centroides. La unidad de filtrado aplica los filtros de mediana y Gauss para eliminar el ruido pico y a su vez suavizar la imagen. La unidad de etiquetado y de memoria aplica técnicas de umbralización para detectar los píxeles que corresponden a estrellas de la imagen y posteriormente etiquetarlos. La unidad de control es la responsable de generar los tiempos requeridos y las señales de control para la activación de las diferentes unidades. Por último la unidad de cálculo remueve estrellas pequeñas basándose en el número de píxeles que las componen. Además tomando información de los bloques de memoria calcula para las estrellas restantes de la imagen el centroide. Para el análisis del algoritmo, se comparó el código en Matlab, con el código HDL implementado en un ASIC, dando como resultados igual cantidad de estrellas detectadas, pero la arquitectura tipo hardware incrementó en

57,5 veces la velocidad de procesamiento. Este método es revolucionario en el sentido de que utiliza hardware para la identificación de estrellas, siendo rápido y reduciendo considerablemente el uso de memoria, además requiere de un solo escaneo de la imagen para determinar todas sus salidas.

Capítulo 2

Sistema de procesamiento digital de imágenes

Un sistema de procesamiento digital de imágenes puede entenderse como un conjunto de elementos que se relacionan entre si para lograr dos objetivos principales, como lo son el de mejorar una imagen para una fácil interpretación por los humanos o como el procesamiento de la imagen para el almacenamiento, la transmisión y representación para la interpretación automática de máquinas.

Para el desarrollo de un sistema de procesamiento se puede tomar ventaja de un "framework" que se define, en términos generales, como un conjunto estándar de conceptos, prácticas y criterios que enfocan un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. Dicho framework es ampliamente utilizado en la literatura por lo que se tomará como referencia durante todo el desarrollo del sistema para la identificación e individualización de estrellas en cúmulos estelares.

La Figura 2.1 representa el marco de trabajo que se distingue por tres niveles principales de acuerdo a la complejidad de implementación. En el primer nivel se ubican las etapas de adquisición, digitalización y preprocesamiento. En estas etapas se pretende obtener una imagen por medio de un instrumento dedicado a dicha labor como una cámara, satélite, video-grabadora, etc. Luego un digitalizador convierte esta señal eléctrica, en una señal digital a la que se conoce como imagen digital, concluyendo este nivel con una serie de técnicas que permiten mejorar la calidad de la imagen además de reducir el ruido que pudo ser adquirido en las fases anteriores, el mejoramiento de la imagen se logra a través de técnicas que modifican su contraste, color, contorno, o características especificas. El segundo nivel esta conformado por las fases de segmentación, representación y descripción. La segmentación es una de las fases más complejas en un sistema de procesamiento puesto que en gran medida el éxito o fracaso del sistema autónomo dependerá de esta fase, en ella se pretende dividir la imagen en regiones más pequeñas que contengan el objeto de interés, esto con el fin de facilitar el estudio de la imagen, concluyendo con las técnicas que pretenden describir y representar cada una de las áreas segmentadas, en datos estadísticos, como por ejemplo forma y tamaño. Por último aparecen las fases de

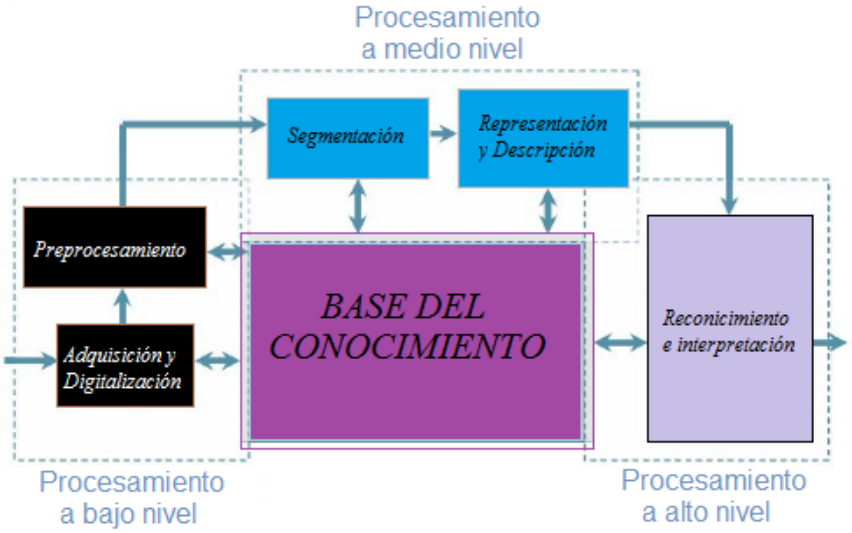


Figura 2.1: Esquema de un sistema de procesamiento digital de imágenes.

alto nivel como lo son las de reconocimiento e interpretación, donde se pretende identificar los objetos de la imagen (una estrella, un planeta, etc), e interpretar los datos estadísticos hallados en el punto anterior.

2.1. Adquisición y digitalización de la imagen

La adquisición y digitalización es una etapa muy importante, ya que toma las señales análogas y las transforma en señales digitales, quienes son totalmente manipulables por un procesador o DSP. La adquisición de imágenes partiendo desde la captura de la señal por el sensor, hasta la digitalización, está conformada por varias etapas, aquí se nombrarán algunas a continuación:

- Captura de la señal.
- Acondicionamiento de la señal.
- Tarjeta de adquisición de datos.

En la siguiente Figura 2.2 se muestra como funciona el proceso.

Figura 2.2: Esquema para la adquisición de imágenes.

2.1.1. Captura de la señal

La captura de la señal se realiza a través de los sensores quienes a su vez son dispositivos capaces de transformar señales físicas (análogas), como por ejemplo la temperatura, la presión, humedad, espectro electromagnético, etc en señales eléctricas como voltaje y corriente, facilitando de esta forma que otros dispositivos como procesadores puedan interpretar y analizar dichas señales. Convirtiéndose en el elemento inicial que conforma un sistema de adquisición.

Actualmente en el mercado se encuentran sensores que pueden generar una salida en tensión y/o corriente, o bien, modifican o extraen una propiedad que pueda ser evaluada de forma eléctrica, de esta manera, y con el debido acondicionamiento, la señal de salida puede ser tratada por una equipo de adquisición de datos o procesamiento de señales como lo pueden ser una DSP, una FPGA o un micro-controlador entre otros.

Las señales del mundo real son generalmente, analógicas y variantes en el tiempo, por tal motivo para que un procesador trabaje con la señal captada por el sensor, esta debe ser digitalizada.

2.1.2. Acondicionamiento de la señal

El acondicionamiento de la señal se realiza con el fin de transformar la señal captada por los sensores en una señal compatible con las características de entrada del sistema de adquisición. Dentro de las características principales de la tarjeta de adquisición se pueden encontrar amplitud de la señal de voltaje, acoplación de señales para protección, linealización, filtrado de ruido entre otros. A continuación se nombran algunas de las fases de la etapa de acondicionamiento:

- **Transformación:** Las tarjetas de adquisición aceptan como entrada señales de voltaje. Ya que los sensores entregan diferentes tipos de señales eléctricas como corriente, voltaje, resistencia, se hace necesario transformar dichas señales a una diferencia de potencial equivalente a la señal inicial.
- **Amplificación:** La tarjeta de adquisición acepta normalmente voltajes comprendidos entre los rangos de -15V hasta los +15V mientras que los sensores suelen entregar pequeños valores en el rangos de los milivoltios, es por esta razón que esta señal debe ser amplificada para una correcta lectura en la tarjeta de adquisición. Esta etapa suele inyectarle ruido a la señal.
- **Conversión por medio de opto acopladores:** Generalmente en equipos médicos se suele aislar el sistema eléctrico de los sensores del sistema eléctrico de la tarjeta de adquisición, esto con el fin de proteger el paciente de alguna señal no deseada proveniente de la etapa de adquisición. Este aislamiento se suele llevar acabo con opto acopladores que convierten las señales eléctricas en señales ópticas.

- **Filtrado:** Los filtros usualmente son los encargados de eliminar las señales no deseadas, esto con el fin de que el sistema de adquisición no pierda resolución y tome la señal lo más limpia posible.
- **Excitación:** Algunos tipos de transductores por su estructura interna y para su correcto funcionamiento, necesitan de una excitación externa de voltaje o corriente.
- **Linealización:** Debido a que algunos sensores brindan una respuesta no lineal en algunos parámetros, se hace necesario realizar una etapa de linealización para un manejo adecuado de la señal.

2.1.3. Tarjeta de adquisición de datos

Las tarjetas de adquisición de datos se encargan de la conversión de las señales analógicas a digitales, esto lo hace a través de conversores ADC. También el otro objetivo de las tarjetas de adquisición es efectuar la comunicación serial con el procesador o DSP.

Entre las características más comunes entre las diferentes tarjetas de adquisición se encuentran las siguientes:

- **Número de canales analógicos:** Esto indica la cantidad de sensores que se pueden conectar a la misma tarjeta. Comúnmente las tarjetas disponen de un único ADC y gracias a un multiplexor analógico se pueden obtener datos de los diferentes canales.
- **Velocidad de muestreo:** La Velocidad de muestreo hace referencia a la cantidad de datos o muestras que se deben tomar para obtener la señal original lo mejor posible, comúnmente se utiliza el Teorema de Nyquist quien dice que la velocidad de muestreo debe ser siempre mayor que el doble de la frecuencia de la señal que se desea muestrear, se puede decir entonces que cuanto mayor sea la velocidad de muestreo mejor sera la representación de la señal analógica.
- **Resolución:** La resolución se refiere la la cantidad de bits que se utilizarán para representar la magnitud de la señal, a mayor número de bits del ADC la tarjeta tendrá la capacidad de detectar hasta las más mínimas variaciones en magnitud, brindando una mejor captación de la señal, el número de niveles en que se divide la señal a convertir por el ADC viene dada por 2^n, donde n es la longitud de la palabra que maneja el conversor.
- **Rango de entrada:** Este indica el rango o valores máximos y mínimos permitidos por la tarjeta de adquisición para su correcto procesamiento. Las tarjetas de adquisición de datos suelen dar varias opciones al programador ya sea por hardware o por software.
- **Capacidad de temporización:** En los últimos años se ha vuelto indispensable que las tarjetas de adquisición cuenten con temporizadores para realizar

tareas como contar eventos, para generar pulsos y crear bases de tiempo, generar formas de onda de acuerdo al reloj entre otras, permitiendo al procesador realizar tareas adicionales sin que se interrumpa el flujo normal.

2.2. Pre-procesamiento

Las técnicas de pre-procesamiento se utilizan por dos objetivos principales, el primero es eliminar el ruido presente como consecuencia de la etapa de formación de la imagen y el segundo pretende mejorar o realzar las propiedades de la imagen, esto con el fin de facilitar las operaciones de las subsiguientes etapas.

Las principales fuentes de inclusión del ruido se presentan en las etapas de captación y transmisión, por falta de iluminación uniforme que produce la sensación de que la imagen estuviera formada por la multiplicación entre dos imágenes, además de la saturación de carga que recibe un píxel ya sea por exceso o por defecto que se conoce como " ruido sal y pimienta", como también de la sensibilidad de las cámaras al infrarrojo que genera objetos "calientes" en la imagen.

El trabajo para el desarrollador se encuentra en determinar la mejor técnica para realizar el pre-procesamiento, y este es verdaderamente difícil ya que no se cuenta con un conocimiento profundo de como la imagen fue degradada. Cuando se necesita mejorar una imagen, hay que tener en mente de que no hay una regla general para determinar cual es la mejor técnica para aplicar, casi siempre es un proceso empírico, además que el método apropiado para un determinado sistema puede ser totalmente inapropiado para otro.

En general las técnicas de mejoramiento se pueden dividir en tres categorías:

- *Métodos en el dominio espacial* están enfocados en manipular directamente los píxeles.
- *Métodos en el dominio de la frecuencia* operan a través de la transformada de Fourier o cualquier otro dominio de frecuencia de la imagen.
- *Híbridos* que combinan los dos tipos de métodos anteriores.

Para llevar a cabo cualquiera de estas técnicas se pueden catalogar además cuatro tipos de procesos diferentes:

- *Proceso puntual* son operaciones enfocadas a modificar cada píxel basándose únicamente en el valor de este.
- *Proceso local* son operaciones enfocadas a la vecindad del píxel a través de una matriz de convolución.
- *Proceso global* son operaciones enfocadas a manipular un píxel dependiendo del valor de toda la imagen.

- *Proceso regional* es del tipo global pero este se enfoca en un área más pequeña, donde la imagen podría ser dividida en zonas de igual tamaño para mejorar el comportamiento del proceso anterior.

2.2.1. Filtrado

Son las técnicas o transformaciones necesarias en imágenes para facilitar la visualización, como la interpretación, además de ser fundamentales en los sistemas de tiempo real, ya que cuando se dispone de tiempos tan reducidos por imagen no es posible realizar una optimización completa, sino más bien es necesario actuar sobre los píxeles según se reciben. Aquí interviene precisamente el concepto de filtro.

Un filtro es un operador que realiza una transformación sobre una señal de entrada produciendo una señal de la misma naturaleza en su salida. En las tareas de visión artificial, el filtrado toma una o varias imágenes para producir otra en la que se han resaltado o atenuado ciertas características.

Un paso habitualmente necesario, tanto para tratar automáticamente las imágenes como para aportarlas a un operador humano, es la atenuación del ruido, que puede proceder de diversas fuentes. Este puede ser introducido por el canal de transmisión, siendo en ese caso de naturaleza aditiva, independiente de la intensidad de la señal.

$$f(x, y) = f'(x, y) + n(x, y) \tag{2.1}$$

Donde $f'(x, y)$ denota a la imagen hipotéticamente sin ruido, $n(x, y)$ es el ruido distribuido por la imagen, y $f(x, y)$ es la imagen que nos llega. Para minimizar este tipo de ruido se puede acudir a estimadores estadísticos o a técnicas de filtrado ad hoc: filtros lineales paso-bajo o no lineales con el mismo comportamiento.

Otra fuente de ruido puede ser el medio de almacenamiento o de digitalización, dependiendo habitualmente del valor de la señal. En cada aplicación será necesario estimar el tipo de ruido que se presenta y la mejor manera de atenuarlo, no existiendo métodos generales que sirvan en todos los entornos [12].

Filtros

Son utilizados para eliminar ruido o detalles pequeños de poco interés puesto que sólo afecta a zonas con muchos cambios. La frecuencia de corte se determina por el tamaño del kernel y sus coeficientes.

El mecanismo de funcionamiento de un filtro a través de un Kernel es ilustrado en la Figura 2.3, el proceso consiste simplemente en mover la máscara del filtro (kernel, ventana), píxel a píxel a través de la imagen $i(x, y)$.

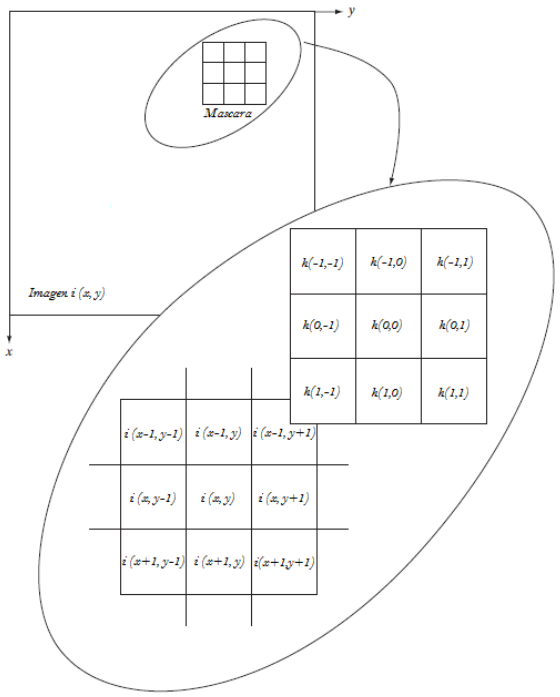


Figura 2.3: Funcionamiento de la ventana (kernel).

Básicamente una máscara es una matriz bidimensional pequeña cuyos valores de los elementos son escogidos para detectar una propiedad de la imagen. En cada punto (x, y), la respuesta es dada por la suma del producto de los coeficientes del filtro y los correspondientes píxeles de la imagen en el área abarcada por la máscara del filtro.

La principal técnica usada para definir la máscara $h(s, t)$, es usar una sub-imagen cuadrada centrada en (x, y), el centro de la sub-imagen se mueve de píxel a píxel en $i(x, y)$ empezando en la esquina superior izquierda y al aplicar el operador en cada píxel se obtiene el valor de cada punto en la nueva imagen $r(x, y)$. Donde $r(x, y)$ es la imagen procesada, y $h(s, t)$ es el operador definido sobre alguna vecindad del punto (x, y).

Dada una imagen $i(x, y)$ y una máscara $h(s, t)$, la imagen resultante $r(x, y)$ consiste en realizar la operación:

$$r(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s, t) i(x + s, y + t) \tag{2.2}$$

Por lo tanto un filtro es una ecuación matemática (denominada producto de convolución) que permite modificar el valor de un píxel según los valores de los píxeles contiguos, con coeficientes por cada píxel de la región a la que se lo aplica. El filtro se representa en una tabla (matriz) que se caracteriza por sus dimensiones y coeficientes, cuyo centro corresponde al píxel en cuestión. Los coeficientes de la tabla determinan las propiedades del filtro.

2.2.2. Filtro del Promedio

Este tipo de filtro generalmente se usa para atenuar el ruido de una imagen, ya que ataca directamente los componentes de la imagen con alta frecuencia (píxeles oscuros); por eso usualmente se lo denomina filtro de suavizado. Los filtros promedio son un tipo de filtros de paso bajo cuyo principio es sacar el promedio de los valores de los píxeles contiguos. Con este filtro se obtiene una imagen más borrosa.

$$Promedio(x,y) = \frac{1}{(2a+1)(2b+1)} \sum_{s=-a}^{a} \sum_{t=-b}^{b} i(x+s, y+t) \tag{2.3}$$

Al reemplazar cada píxel de la imagen por el promedio de la vecindad definida por la máscara se obtiene como resultado la reducción de los detalles o cambios abruptos en los niveles de gris. La aplicación más obvia es la reducción del ruido. Sin embargo, los bordes de los objetos también son transiciones abruptas de los tonos de gris por lo que los bordes resultan emborronados después de la aplicación de estos filtros.

Para un caso particular 3x3 donde en el filtro promedio a = b = 1 se obtiene la matriz:

$$h(s,t) = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.4}$$

Por lo que de (2.3) se obtiene (2.5) específica para un filtro 3*3.

$$Promedio(r,c) = N(r,c) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} i(r+s, c+t) \tag{2.5}$$

Un área donde resulta realmente importante la aplicación del filtro del promedio es la astronomía donde se vuelve rutina encontrar imágenes con muy bajo nivel de luz, causando frecuentemente que el ruido se mezcle con la imagen[8].

2.2.3. Filtro de la Mediana

Son filtros de orden estadístico, de tipo espacial no lineal cuya respuesta se basa en el estatus o posicionamiento de los píxeles contenidos en el área de la imagen que abarca el kernel que efectúa la ejecución del filtro. El mejor ejemplo conocido de este tipo es el filtro de mediana, el cual, como su nombre lo sugiere, reemplaza el valor de un píxel por la mediana de los niveles de gris en la vecindad incluyendo a este. Los filtros de la mediana son muy populares puesto que, para varios tipos de ruido aleatorio, proporciona excelentes capacidades de reducción de este, con una cantidad inferior de recursos que los filtros de suavizado lineal de tamaño similar.

$$Mediana(r,c) = N(r,c) = Vcentral(Orden(i(r+s, c+t))) \tag{2.6}$$

Los filtros de la mediana son particularmente eficaces ante la presencia de ruido de impulso, también llamado ruido de sal y pimienta, quien es llamados así debido

a su aspecto como puntos blancos y negros superpuestos en una imagen.

La mediana, m, de un conjunto de valores tal que, la mitad de los valores en el conjunto son menores que o iguales a m, y la mitad son mayores que o iguales a m. Con el fin de realizar el filtrado de la mediana en un punto dado en una imagen, en primer lugar se ordenan los valores de la vecindad contenida en la ventana, para así determinar la mediana y posteriormente asignar este valor al píxel central de la ventana. Por ejemplo, en una vecindad 3x3, la mediana es el quinto valor más grande, en una vecindad 5x5 el treceavo valor más grande corresponde a la mediana, y así sucesivamente. Si se supone una vecindad 3x3 con valores (30, 10, 90, 60, **20**, 50, 80, 40, 70) se ordenan como (10, 20, 30, 40, **50**, 60, 70, 80, 90), la mediana será el nivel 50. En consecuencia, el ruido que tiene un valor atípico en el entorno quedará colocado en los extremos de la ordenación. Véase cómo el nivel 20 queda desplazado y la imagen toma el valor de 50 después de la ordenación.

Por lo tanto, la idea fundamental con los filtros de la mediana es forzar los valores que se encuentran dentro de la imagen y se alejan demasiado de sus vecinos a tener valores aproximados entre ellos, de esta manera se logra eliminar los valores aislados con picos de intensidad. Aunque el filtro de la mediana es de lejos el filtro más útil de orden estadístico en el procesamiento de imágenes, no es de ninguna manera el único. Pero es de recordar que dentro de las estadísticas básicas se clasifican muchas otras posibilidades. Por ejemplo, utilizando los resultados en el denominado filtro máx, que es útil en la búsqueda de los puntos más brillantes en una imagen. También se encuentra el filtro min, utilizado para el propósito opuesto.

2.2.4. Filtro Gaussiano

El filtro Gaussiano es otro filtro de suavizado por lo que sirve para eliminar ruido y produce el efecto de difuminar las imágenes. Su construcción es similar al filtro de la media y solo cambia la máscara o kernel. En la siguiente matriz 3*3 se observa el modelamiento para la función Gaussiana:

$$h(s,t) = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix} \tag{2.7}$$

Para una varianza dada utilizando una discretización de la función de la densidad normal de media cero $N(0,\sigma^2)$ se obtiene la matriz de convolución de un filtro Gaussiano.

$$h_\sigma(x,y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{2.8}$$

Donde la varianza σ determina la respuesta en frecuencia o ancho de banda de estos filtros. Son filtros lineales y como todo filtro binomial sus máscaras bidimensionales pueden ser construidas a partir de sus máscaras unidimensionales. Cabe resaltar que la principal ventaja de estos filtros es que se puede manipular la varianza, de modo que si se considera una varianza grande se le estará dando importancia a

los vecinos más lejanos, mientras que al considerar una varianza pequeña se le estaría dando mayor importancia a los vecinos más cercanos. Normalmente un desarrollador pretende tener una varianza pequeña ya que desea dar mayor peso al centro y sus vecinos más cercanos.

Para evitar la formación de picos de intensidad en la imagen se suele tomar como regla general que el el operador w, debe ser al menos de:

$$w \geq 3c \tag{2.9}$$

Siendo c el tamaño del lóbulo central y dependiente de la varianza.

$$c = 2\sqrt{2\sigma^2} \tag{2.10}$$

En Matlab modelan el filtro de Gauss como:

$$H_g(n_1, n_2) = e^{-({n_1}^2+{n_2}^2)/2\sigma^2} \tag{2.11}$$

Donde el H se determinaría:

$$H(n_1, n_2) = H_g(n_1, n_2) / \sum_{n_1} \sum_{n_2} H_g \tag{2.12}$$

2.3. Segmentación

La segmentación abarca las técnicas que permiten agrupar los elementos fundamentales de una imagen por algún criterio de homogeneidad, logrando de esta manera particionar la imagen en segmentos o regiones de interés, esto con el fin de hacer la imagen más fácil de analizar. Entonces segmentar significa subdividir una imagen en regiones que constituyan el objeto de investigación. La segmentación debe detenerse cuando los objetos de interés en una aplicación han sido aislados. El contraste, textura, color, luminancia, entre otras, son algunas de las características bajo los cuales se realiza el agrupamiento de píxeles. Después de realizada la segmentación la imagen estará preparada para la representación y descripción, ya no se hablará de los píxeles como elemento más básico sino que lo serán los nuevos agrupamientos de píxeles que conforman los objetos.

Dentro del procesamiento digital de imágenes la segmentación toma gran relevancia ya que el eventual éxito o fracaso del algoritmo dependen de este, de hecho muy pocas veces se logra obtener resultados 100 % satisfactorios por lo que el desarrollador tiene que tomar medidas y buscar métodos alternativos que combinados le den mejores resultados, pero recordar que nada es gratis por lo que el precio lo termina pagando el procesador, adquiriendo mayores tiempos de procesamiento.

El proceso para determinar los algoritmos a implementar para la fase de segmentación incluye dos pasos básicamente, el primero seria analizar si la imagen tiene bien definido el fondo con respecto a los objetos, esto implicaría que se tienen cambios bruscos en la escala de grises por lo que bastaría simplemente con aplicar técnicas

de umbrales, si es así la imagen resultaría muy sencilla de trabajar, pero lo normal es que esto no ocurra por lo que se tendría que ejecutar al siguiente paso que abarca mayor complejidad, aquí se tendrían que recurrir a técnicas que trabajen con gradientes de luminosidad. A continuación se nombran algunos de los métodos usuales para realizar tareas de segmentación de acuerdo a lo descrito anteriormente.

2.3.1. Umbralización

La umbralización abarca muchas de las técnicas de segmentación más utilizadas, esto por su sencillez y bajo costo computacional. Supone un umbral donde los valores por encima de este son el objeto y por debajo de este son el fondo, esto quiere decir que el objeto puede ser extraído del fondo mediante una simple operación que compare los valores de la imagen con el valor del umbral T.

Hay diversos métodos de umbralización entre los que se encuentran métodos globales que dependen de $i(x, y)$ basados en histogramas de los niveles de grises de toda la imagen, métodos regionales donde el umbral depende de alguna similitud o continuidad del píxel en una región dada $r(x, y)$ pueden ser utilizados los mismos métodos globales pero por regiones. Bajo esta característica se considera que la imagen esta formada por n regiones, y por último métodos locales donde el umbral depende del píxel en su propia vecindad $l(x, y)$.

$$T(x, y) = T(i(x, y), r(x, y), l(x, y)) \tag{2.13}$$

Al aplicar estas técnicas sobre una imagen en escala de grises el resultado es una imagen binaria, donde los píxeles con valores de intensidad igual a 1 corresponden al objeto deseado; mientras que los píxeles con valor 0, corresponder al resto de la imagen (fondo).

La imagen $b(x, y)$, resultante de aplicar la umbralización, viene definida por:

$$b(x, y) = \left\{ \frac{t_0 \Rightarrow Si, i(x, y) \prec T}{t_0 \Rightarrow Si, i(x, y) \succ T} \right\} \tag{2.14}$$

1. **Umbralización global:** En la umbralización global se modifica el valor del píxel actual, de acuerdo al valor de todos y cada uno de los píxeles que conforman la imagen. Existen determinados métodos para realizar una umbralización global, la mayoría de ellos se basa a su histograma. Para crear el histograma de una imagen se requiere ordenar los valores de intensidad de los píxeles de acuerdo a la frecuencia relativa de aparición. Con el histograma resulta fácil calcular un umbral para distinguir las dos clases[13].

   En un caso ideal, el histograma de intensidad de una imagen (niveles de gris) tendría bien marcado los dos picos para objeto y fondo Figura 2.4, y se puede decir que el umbral óptimo es aquel valor T que separa ambas regiones.

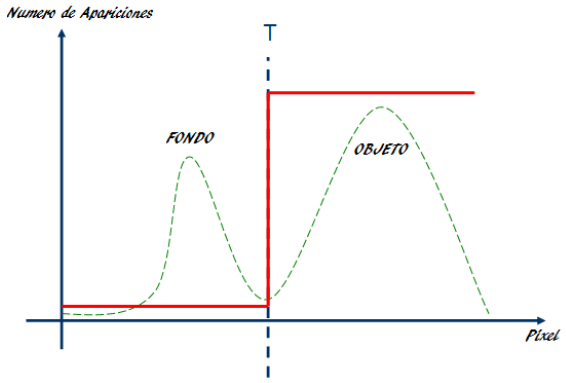


Figura 2.4: Umbral óptimo de separación entre clases.

A continuación se nombran algunas técnicas globales que se basan en el histograma bimodal:

a) Método del Triángulo: Para hallar el umbral adecuado mediante el método del triángulo, se debe trazar una línea recta entre el valor máximo del pico del fondo "P1max" y el valor máximo del objeto "P2max" en la gráfica del histograma. El umbral se determina mediante otra línea recta la cual debe cortar perpendicular a la primera y rodear el histograma hasta encontrar la máxima distancia "d". El punto donde la recta toca el histograma corresponde al umbral deseado "T". En la Figura 2.5 se ilustra el procedimiento cuando sobre una curva suavizada del histograma se determina el método del triángulo.

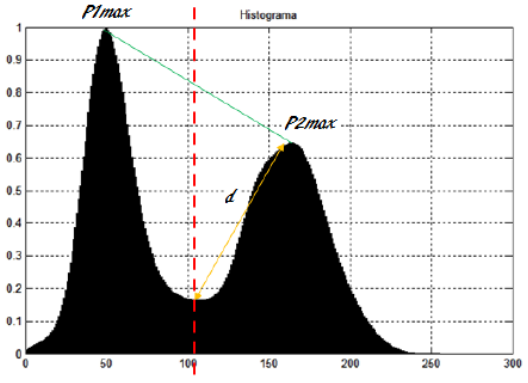


Figura 2.5: Determinación del umbral por el método del triángulo.

b) Método de la Entropía: Se usa la entropía H como una medida del contenido de la información. El umbral T separa la información en dos clases correspondientes al fondo y al objeto, determinando la entropía asociada a ellas. El umbral T se determina para aquel valor donde la entropía es máxima. La entropía se puede calcular a través de:

$$H = -\sum_{K=0}^{T} p_I(K) * \log_2(p_I(K)) - \sum_{K=T+1}^{L-1} p_I(K) * \log_2(p_I(K)) \quad (2.15)$$

c) Método de Otsu: Esta técnica itera a lo largo del histograma calculando para cada valor estimado posible de T [0 hasta 255] la varianza de pesos dentro de cada clase. El umbral óptimo se logra cuando la varianza entre clases entrega un valor mínimo. La varianza entre clases se calcula mediante:

$$\sigma_w^2(t) = P_1(t) * \sigma_1^2(t) + P_2(t) * \sigma_2^2(t) \quad (2.16)$$

Donde las probabilidades de cada clase se determinan por medio de:

$$P_1(t) = \sum_{i=0}^{t} p(i) \quad (2.17)$$

$$P_2(t) = \sum_{i=t+1}^{255} p(i) \quad (2.18)$$

Y los promedios de clase a través de:

$$\mu_1(t) = \frac{\sum_{i=0}^{t} i * p(i)}{P_1(t)} \quad (2.19)$$

$$\mu_2(t) = \frac{\sum_{i=t+1}^{G-1} i * p(i)}{P_2(t)} \quad (2.20)$$

Con las varianzas de cada clase presentadas mediante:

$$\sigma_1^2(t) = \sum_{i=0}^{t} [i - \mu_1(t)]^2 \frac{p(i)}{P_1(t)} \quad (2.21)$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{255} [i - \mu_2(t)]^2 \frac{p(i)}{P_2(t)} \quad (2.22)$$

2. **Umbralización regional:** En la umbralización regional se modifica el valor del píxel actual, de acuerdo al valor de todos y cada uno de los píxeles que conforman la región a la que pertenece, donde la región es una sub-imagen n perteneciente a la imagen i, ver Figura 2.6. En este método, primero, se divide una imagen en sub-imágenes rectangulares solapadas, y se obtienen los histogramas para por último calcular los umbrales globales de dichas sub-imágenes.

   Las regiones deben ser lo suficientemente grandes como para abarcar el fondo y el objeto al tiempo. Siempre y cuando las sub-imagen tengan un histograma

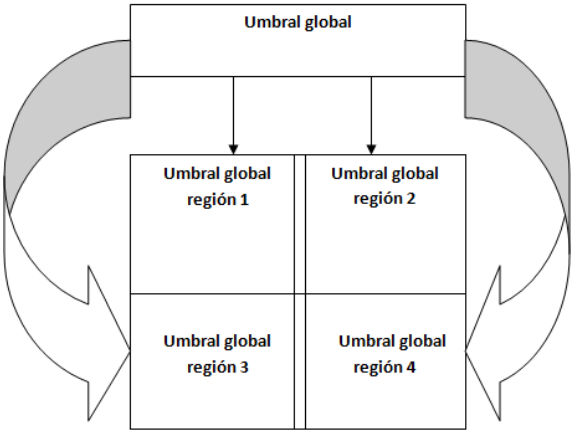


Figura 2.6: Umbralización regional.

bimodal, se pueden utilizar las mismas técnicas que aplican para determinar un umbral global.

En muchas aplicaciones, no se puede obtener un umbral global para un histograma, es decir, no se puede obtener una buena segmentación con un único umbral para toda la imagen. Esto ocurre cuando el fondo no es constante y el contraste de los objetos varía en la imagen, por lo que la umbralización daría buenos resultados en una parte de la imagen, pero para el resto de la imagen, la segmentación no sería la adecuada.

Si las variaciones del fondo de la imagen pueden ser descritas a través de funciones conocidas, dependientes de la posición en la imagen, se podría intentar corregir la segmentación utilizando la corrección de niveles de gris; para que, posteriormente, la aplicación de un único umbral a toda la imagen, diera buenos resultados en la segmentación. Otra solución sería el uso de la umbralización local.

3. **Umbralización local:** En este tipo de técnicas el píxel de salida depende del píxel de entrada o valores en una vecindad de ese píxel. Algunos ejemplos son técnicas de detección de bordes (Canny, Sobel, Prewitt, Roberts), filtros estadísticos básicos (Promedio, Mediana, Máximos y Mínimos), y técnicas avanzadas (Sauvola, Niblack, Laplaciano, Gradiente). Estas operaciones pueden ser regionales ya que los resultados dependen de los valores de los píxeles particulares encontrados en cada región de la imagen[14].

   a) **Sauvola** La técnica de Sauvola[15] calcula un valor de umbral óptimo $T(r, c)$ para el píxel $p(r, c)$. Donde $lmean(r, c)$ es el promedio en la vecindad del píxel $p(r, c)$, R es el máximo valor de la desviación estándar posible, siendo la intensidad máxima 255 entonces $R = 128$, K es un

valor que varía entre 0,2 y 0,5, y $lstd(r,c)$ es la desviación estándar hallada en la vecindad del píxel $p(r,c)$ centrada en (r,c). Según (2.23) el uso de $lmean(r,c)$ como factor multiplicativo de K y R tiene un efecto de amplificar la contribución de $lstd(r,c)$ de forma adaptativa.

$$T(r,c) = lmean(r,c)\left[1 + K\left(\frac{lstd(r,c)}{R} - 1\right)\right] \tag{2.23}$$

Para el cálculo de la desviación estándar $lstd(r,c)$ ó σ se puede utilizar:

$$lstd(r,c) = \sqrt{C\sum_{s=-n}^{n}\sum_{t=-m}^{m}(p(r+s,c+t) - lmean(r,c))^2} \tag{2.24}$$

Para el cálculo del promedio $lmean(r,c)$ se puede utilizar:

$$lmean(r,c) = C\sum_{s=-n}^{n}\sum_{t=-m}^{m}p(r+s,c+t) \tag{2.25}$$

Donde C es:

$$C = \frac{1}{(2n+1)(2m+1)} \tag{2.26}$$

b) **Niblack** La técnica de Niblack[16] calcula un valor de umbral óptimo $T(r,c)$ para el píxel $p(r,c)$, Donde $lmean(r,c)$ es el promedio en la vecindad, $lstd(r,c)$ es la desviación estándar hallada en la vecindad de $p(r,c)$ centrada en (r,c), y K toma valores de 0,2 para objetos brillantes y de -0,2 para objetos opacos. Niblack considera que K es un valor que debe estar entre -1 y 0.

$$T(r,c) = lmean(r,c) + K * lstd(r,c) \tag{2.27}$$

c) **Máximos y Mínimos** Para realizar esta técnica se requiere inicialmente hallar la intensidad máxima de la ventana $lmax(r,c)$ y la intensidad mínima de la ventana $lmin(r,c)$, y por último se halla el umbral óptimo $T(r,c)$ de máximos y mínimos como:

$$T(r,c) = \frac{lmax(r,c) + lmin(r,c)}{2} \tag{2.28}$$

La umbralización local es computacionalmente más costosa que la global. Es muy útil a la hora de segmentar objetos en fondos no homogéneos, y para extraer regiones muy pequeñas y dispersas.

4. **Operaciones geométricas:** la salida en un píxel sólo depende de los niveles de entrada en algunos otros píxeles definidos por las transformaciones geométricas. Operaciones geométricas son diferentes de las operaciones globales, de tal

manera que la entrada es sólo de algunos píxeles específicos sobre la base de la transformación geométrica, y no requieren la entrada de todos los píxeles para hacer su transformación.

Las operaciones morfológicas se encuentran entre esta clasificación, ya que son técnicas de procesamiento no lineal de la señal, caracterizadas en realzar la geometría y forma de los objetos. Su fundamento matemático se basa en la teoría de conjunto. Aunque en un principio se aplicará sobre imágenes binarizadas, luego se extenderá a imágenes en niveles de grises. Actualmente, se puede encontrar aplicaciones en la restauración de imágenes, en la detección de bordes, en el análisis de texturas, en el aumento del contraste y hasta en la compresión de imágenes. Los operadores básicos de la morfología son la erosión, la dilatación, la apertura y el cierre que trabajan en base a un elemento estructurante "EE" también conocido en imágenes como Kernel.

El elemento estructurante es un conjunto de puntos matriciales que determinan la estructura de la imagen sobre la que se aplica la operación morfológica. Este es el responsable de la forma y el tamaño de los objetos que forman la imagen. Todo EE se expresa con respecto a un origen llamado elemento director Figura 2.7.

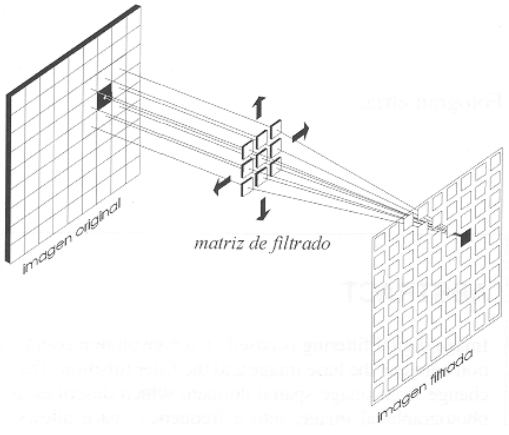


Figura 2.7: Elemento estructurante.

a) Erosión: La transformación de la erosión es el resultado de comprobar si el elemento estructurante B está completamente incluido dentro del conjunto X. Cuando no ocurre, el resultado de la erosión es el conjunto vacío.

$$\epsilon_B(X) = X \ominus B = x|B_x \subseteq X \tag{2.29}$$

Cuando los objetos de la escena sean menores que el elemento estructurante, éstos desaparecerán. Otra interpretación de la erosión supone tomar el valor mínimo de la imagen en el entorno de vecindad definido por el elemento estructurante.

b) Dilatación: es la transformación dual a la erosión. El resultado de la dilatación es el conjunto de puntos del elemento estructurante B, tal que al menos uno está contenido en algún elemento de X, cuando el elemento estructurante se desplaza sobre el conjunto X.

$$\delta_B(X) = X \oplus B = x|X \cap B_x \neq \phi \tag{2.30}$$

La dilatación también se interpreta como el valor máximo del entorno de vecindad definido por el elemento estructurante.

c) Apertura (Opening): Usando los operadores elementales de erosión y dilatación, se pueden diseñar operaciones de realce de las formas de los objetos. La erosión binaria suele utilizarse para eliminar pequeños objetos, práctica que suele ser utilizada en la fase de post-procesado, después de la segmentación. Sin embargo, tiene el inconveniente de disminuir el tamaño del resto de los objetos. Este efecto puede ser subsanado con una aplicación en cascada de erosión y dilatación binaria con igual elemento estructurante.

$$\gamma_B(X) = X \circ B = \delta_B(\epsilon_B(X)) \tag{2.31}$$

d) Cierre (Closening): Por el contrario, la dilatación binaria opera engrandando los objetos, cerrando los agujeros y las grietas. El ensanchamiento de los objetos puede ser reducido mediante la aplicación seguida de una erosión. La operación combinada de dilatación y erosión es llamada cierre (closing):

$$\varphi_B(X) = X \bullet B = \epsilon_B(\delta_B(X)) \tag{2.32}$$

El cierre binario morfológico produce que la dilatación rellene las estructuras que la erosión no puede separar. Los contornos de los objetos también serán suavizados, pero habiendo rellanado las fisuras.

2.4. Representación y descripción

La representación y descripción de objetos o regiones constituye la etapa siguiente a la segmentación, y se convierte en la entrada para la etapa de reconocimiento de objetos, y de esta manera obtener un sistema de procesamiento de imágenes automático. Son muchos los métodos que se pueden utilizar en esta fase, pero la selección de uno u otro se basa unicamente en el problema a solucionar, su objetivo radica unicamente en capturar la esencial diferencia entre objetos, regiones o clases, mientras se mantienen características importantes como su posición, tamaño y orientación[8].

Una representación puede ser dada en características externas como contorno, forma o puede ser dada en características internas como color, textura. Seleccionar el esquema de representación es solo una parte de la tarea de hacer los datos útiles para un procesador.

1. **Representación:** Aunque los objetos algunas veces pueden ser descritos inmediatamente después de la etapa de segmentación, una buena práctica seria realizar la representación para compactar los datos, siendo más útil la representación en comparación con los descriptores respecto al procesador.

   En esta etapa se encuentran técnicas enfocadas a cadenas de píxeles, perímetros poligonales, esqueléticos, puntos de inflexión de una curva, morfología, entre otras.

   a) **Cadena:** Este tipo de técnicas se usan para representar un contorno de acuerdo a una secuencia de segmentos de líneas conectadas en una dirección y longitud especifica. Típicamente esta representación es basada en la conectividad. La dirección de cada segmento es codificada usando un esquema de numeración Figura 2.8.

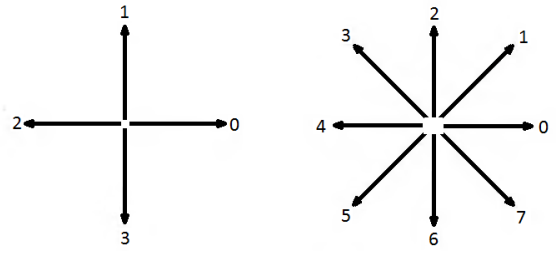


Figura 2.8: Esquema de numeración.

Usualmente la imágenes son adquiridas y procesadas en formato cuadricula, con igual espaciamiento en las dos direcciones x, y, a diferencia de la técnica de la cadena que puede ser procesada siguiendo un contorno (suponerse en el sentido de las manecillas del reloj) y asignando una dirección a cada segmento que conecta un par de puntos Figura 2.9.

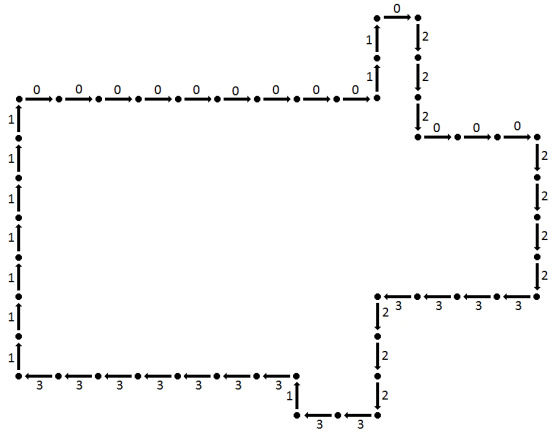


Figura 2.9: Representación por el método de la cadena.

Este método generalmente no se puede trabajar por dos razones principales: primero el algoritmo de cadena resultante tiende a ser bastante largo, y segundo es que pequeñas alteraciones a lo largo de la frontera debido al ruido o imperfecta segmentación causan cambios que no están relacionados con el objeto.

b) **Aproximación poligonal:** Un contorno puede ser aproximado por un polígono con una precisión arbitraria. Para una curva cerrada, la aproximación es exacta cuando el número de segmentos en el polígono es igual al número de puntos en el contorno, de manera que cada par de puntos adyacentes define un segmento en el polígono. En la práctica, el objetivo de la aproximación poligonal es capturar la esencia del contorno del objeto con el menor número posible de segmentos poligonales. Estas técnicas poseen problemas ya que pueden rápidamente convertirse en métodos costosos computacionalmente. Sin embargo, varias técnicas de aproximación poligonal son adecuadas para aplicaciones de procesamiento de imagen gracias a menor complejidad y requisitos de procesamiento. Ver Figura 2.10

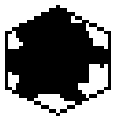


Figura 2.10: Aproximación poligonal.

c) **División sucesiva:** Esta técnica se basa en realizar la división de un segmento en dos partes sucesivamente hasta que se cumpla con un criterio establecido. Se debe tener cuidado que la máxima distancia perpendicular desde un segmento del contorno hasta la línea formada por dos puntos no sobrepase un umbral previamente hallado. Si es así, el más lejano punto desde la línea se convierte en un vértice y entonces se tendría que dividir el actual segmento en dos sub-imágenes. Esta técnica debe tomar ventaja de los puntos de inflexión, regularmente el mejor comienzo es buscar los puntos más alejados del contorno.

En la Figura 2.11 se explica el funcionamiento de esta técnica, donde en (1) se tiene un contorno de un objeto, en (2) se observa que una línea continua marca los puntos más alejados de dicho contorno, el punto c es el más alejado desde la perspectiva de la distancia perpendicular con respecto a a, de esa misma manera el punto c es el más alejado con respecto a la recta, de la misma forma sucede con d, es el punto más distante, en (3) se muestra los resultados de aplicar una técnica de divisiones sucesivas con un umbral de 0,25 veces la longitud de la línea ab, como ningún punto en el nuevo segmento tiene una perpendicular que satisfaga el umbral, el procedimiento termina con el polígono mostrado

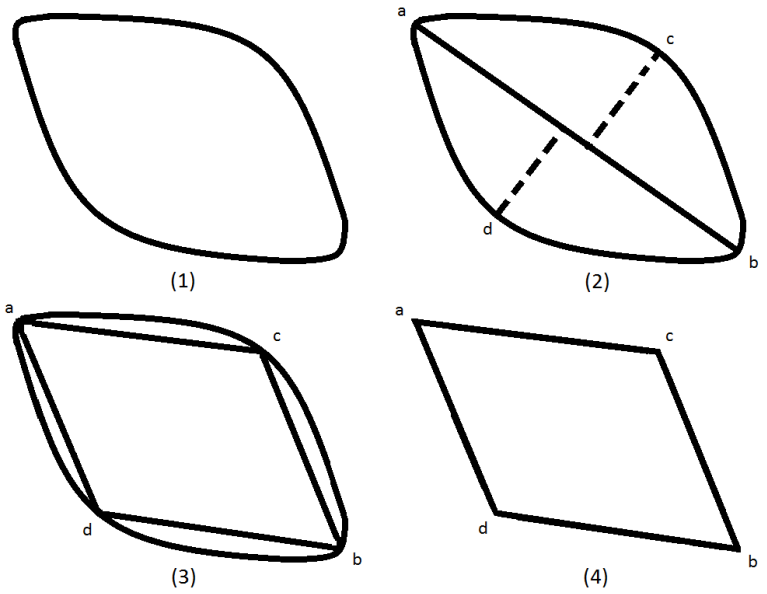


Figura 2.11: Divisiones sucesivas.

en (4).

A continuación se introducen algunos conceptos básicos referentes a diferentes relaciones entre píxeles:

a) **Vecindad de un píxel:** Un píxel p en las coordenadas (r, c) tiene dos vecinos horizontales $(r, c-1), (r, c+1)$ y dos vecinos verticales $(r-1, c), (r+1, c)$, a este conjunto de píxeles se les conoce como $N_4(p)$, además un píxel p en las coordenadas (r, c) tiene cuatro vecinos diagonales $(r-1, c-1), (r-1, c+1), (r+1, c-1), (r+1, c+1)$, al que se le conoce como $N_D(p)$, si se mezclan los anteriores conjuntos se obtiene una vecindad $N_8(p)$.

b) **Conectividad:** La conectividad entre píxeles es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas en una imagen. Para establecer si dos píxeles están conectados, primero se debe determinar si son vecinos y luego si satisfacen algún criterio de similitud en la escala de grises.

c) **Adyacencia:** En escala de grises el rango de posibles valores para un píxel va de 0 a 255, para conocer si dos píxeles p y q son adyacentes se debe determinar primero si pertenecen a algún sub-conjunto V de estos posibles 256 valores, y si además cumplen con alguno de los siguientes tres criterios:

- **4-Adyacencia:** Dos píxeles p y q de un sub-conjunto de V, son adyacentes si y solo si q pertenece a la vecindad $N_4(p)$.

- **8-Adyacencia:** Dos píxeles p y q de un sub-conjunto de V, son adyacentes si y solo si q pertenece a la vecindad $N_8(p)$.

- **m-Adyacencia:** Dos píxeles p y q de un sub-conjunto de V, son adyacentes si y solo si q pertenece a la vecindad $N_4(p)$ ó $N_D(p)$ y la intersección de los conjuntos $N_4(p)$ y $N_4(q)$ no comparte valores en el sub-conjunto V.

d) **Región:** S representa un subconjunto de píxeles de una imagen I. Se llama "Componente Conectado" a todo el subconjunto de píxeles conectados a p dentro de S. Se le llama a S "Conjunto Conectado" si solo contiene un "Componente Conectado". R representa un subconjunto de píxeles de una imagen I, se le llama a R una región de la imagen si este es un "Conjunto Conectado".

e) **Contorno, Borde:** Se le llama borde de una región R al conjunto de píxeles que tiene uno o más vecinos que no están conectados a la región R.

2. **Descripción:** Esta sección pretende describir el objeto, región o clase de acuerdo a la representación seleccionada, esto permitirá clasificar una imagen de acuerdo a sus características y empezar a darle sentido a uno grupo de píxeles y así saber que representan. Por ejemplo un objeto puede ser representado por su contorno, el cual puede ser descrito por características como longitud y orientación. A continuación se verán dos de los descriptores más usados en la literatura como lo son los de contorno y regiones, esto es así ya que son los que mejor representan la forma como los humanos ven el mundo.

a) **Descriptores de contorno:** Este tipo de descriptores se usa unicamente cuando interesa conocer aspectos de como esta formado el objeto.

Longitud: Uno de los más simples descriptores es la longitud, el número de píxeles a lo largo de un borde es una aproximación de su longitud. Por ejemplo para calcular la longitud de un contorno representado por una técnica cadena, estaría dada por la suma de todas las componentes horizontales y verticales más $\sqrt{2}$ las componentes diagonales. El diámetro de un contorno B es dado por:

$$Diam(B) = max_{i,j}[D(p_i, p_j)] \tag{2.33}$$

Donde D es la distancia medida y p_i y p_j son puntos dados en el contorno, el valor del diámetro y la dirección del segmento de línea que conecta los dos puntos extremos se conoce como el "Eje Mayor". El "Eje Menor" del contorno es definido como la línea perpendicular al eje mayor, y con los cuatro puntos que resultan se forma una caja

que contiene todo el objeto, este descriptor es llamado "Rectángulo básico", y el punto de intersección entre el eje mayor y eje menor es llamado la "Excentricidad del Contorno".

Fourier: Este tipo de descriptor se usa unicamente cuando el contorno del objeto puede expresarse como una función periódica, para poder aplicarse la transformada de Fourier. La descripción quedará dada por todos los coeficientes de dicha transformada.

Aproximaciones poligonales: Este tipo de descriptor se usa por medio de segmentos de línea que se unen a través del contorno del objeto, por lo forma visual que se obtiene al final recibe su nombre.

b) **Descriptores de regiones:** Este tipo de descriptores se usan cuando la idea principal es obtener propiedades especificas de las regiones como luminosidad, color, textura, etc. Debe tenerse en cuenta que en la mayoría de los casos su usan al mismo tiempo tanto los descriptores de contorno como los de regiones.

Área: Esta descripción es de las más simples puesto que solo indica el número de píxeles que pertenecen a un objeto, si se desea hallar el área exacta de un objeto, debe obtenerse la relación de cada píxel con el mundo real. El perímetro se entiende como la longitud de un contorno, otros de los más simples descriptores son los basados en el promedio, la mediana y máximos y mínimos de una región en escala de grises.

Textura: Esta descripción pretende cuantificar la información relativa a la textura, aunque no existe una definición exacta de textura se pueden catalogar algunas propiedades básicas como suavidad, aspereza y regularidad de la superficie, hay tres enfoques principales mediante los cuales se puede realizar la descripción por textura y son estadísticos, espectrales y estructurales.

2.5. Reconocimiento e interpretación

El reconocimiento e interpretación es la etapa final en un sistema de procesamiento digital de imágenes, después de concluida esta etapa los objetos habrán sido reconocidos. La esencia del reconocimiento radica en el concepto de enseñanza a partir de un patrón de muestra.

El enfoque de reconocimiento de patrones es dividido en dos áreas principales, una es el área de decisión teórica que toma ventaja de los descriptores cuantitativos como los de longitud, área y textura, y la otra área es de decisiones estructurales que

se basan en los descriptores cualitativos como los relacionales cuyo objetivo principal es hallar la forma de reescribir las reglas básicas de los patrones repetitivos en un límite o región.

En la literatura de reconocimiento de patrones es frecuentemente usado el termino "característica" para denotar a un descriptor, un patrón es entonces una disposición de descriptores, y de la misma manera una clase paterna es una familia de patrones que comparten características similares.

El reconocimiento de objetos realizado por máquinas involucra técnicas para asignar patrones a las clases automáticamente y con la menor cantidad de intervención humana posible.

Los tres principales arreglos de patrones usados son los vectores, los tipo cadena y tipo árbol, los vectores para descriptores cuantitativos, y el tipo árbol y cadena para descriptores estructurales.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ \dots \\ x_n \end{bmatrix} ; \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ \dots \\ y_n \end{bmatrix} ; \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ \dots \\ z_n \end{bmatrix}$$

(a)

Figura 2.12: Vectores de patrones.

En la Figura 2.12 numeral (a) se observa los componentes x_i,y_i,z_i, donde i representa cada uno de los descriptores y n representa el número total de patrones en cada vector.

Cada vector patrón puede ser expresado como:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \tag{2.34}$$

Otra forma de expresar un vector de patrones es por medio de su traspuesta.

$$\mathbf{x} = (x_1 \ x_2 \ \dots \ x_n)^T \tag{2.35}$$

En la Figura 2.13 numeral (b) se observa el modelo de descriptor en escalera, el cual se puede representar como una cadena encargada de genera los descriptor que representa los contornos o bordes del objeto que se este analizando. La estructura de

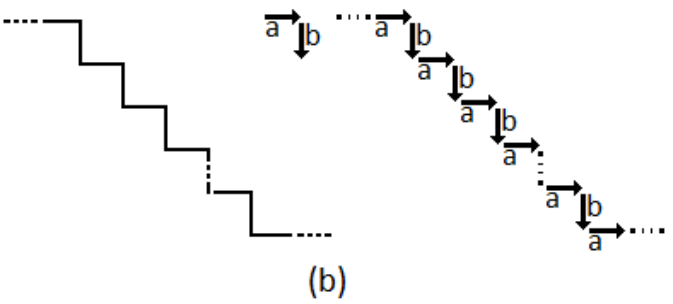


Figura 2.13: Descriptores por escaleras.

este modelo consiste en repetir dos elementos primitivos que se representan con las letras (a) y (b) como se observa en (2.36) realizando la conectividad cabeza a cola. y por seguir solo símbolos alternantes. Esta construcción estructural es aplicable a escaleras de cualquier longitud pero se excluye de otros tipos de estructuras, que podrían ser generadas por otras combinaciones de primitivas a y b.

$$w = ...\ ababababab\ ... \tag{2.36}$$

Los descriptores cadena generan adecuadamente patrones de objetos y otras entidades cuya estructura se basa en la conectividad relativamente simple de primitivas, por lo general asociados con la forma del contorno.

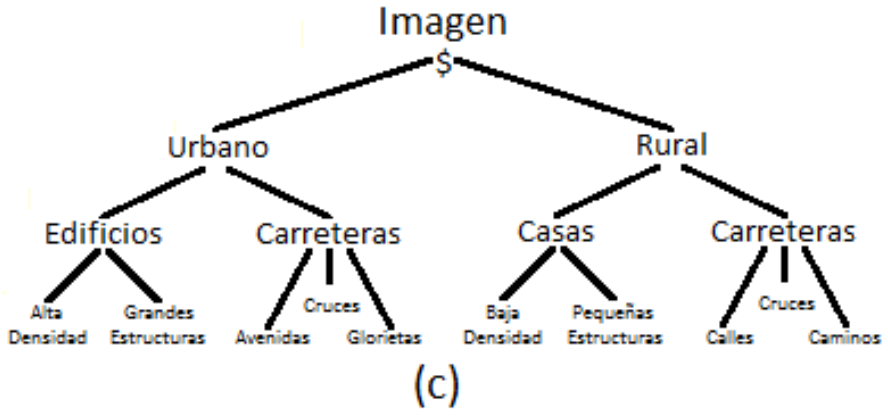


Figura 2.14: Descriptores tipo árbol.

Un enfoque más potente para muchas aplicaciones es el uso de los descriptores tipo árbol Figura 2.14 numeral (c). Básicamente, la mayoría de los esquemas de ordenamiento jerárquico conducen a estructuras de árbol.

Teniendo en cuenta las anteriores representaciones se puede clasificar las técnicas de interpretación en dos grandes métodos, el método de reconocimiento en decisión teórica y el método estructural.

1. **Métodos de reconocimiento por decisión teórica:** Estos tipos de enfoques se basan en el uso de funciones de decisión o discriminantes. Si $x = (x_1\ x_2\ ...\ x_n)^T$, donde x representa un vector patrón de dimensión n, para las clases patrones W, w_1, w_2, ... , w_W, el problema básico del enfoque de decisión teórica es encontrar las funciones de decisión W, $d_1(x)$, $d_2(x)$, ... , $d_W(x)$ con la propiedad de que, si un patrón x pertenece a la clase w_i, entonces:

$$d_i(x) \succ d_j(x)\ j = 1, 2, ..., W;\ j \neq i. \tag{2.37}$$

En otras palabras, se dice que un patrón desconocido x pertenece a la clase de patrón i-nésimo, si tras sustituir x en todas las funciones de decisión $d_i(x)$, este es el mayor valor numérico. Los empates se resuelven de manera arbitraria. La frontera de decisión que separa la clase w_i de w_j, está dada por los valores de x para los que $d_i(x) = d_j(x)$ o, equivalentemente, para los valores de x para los cuales:

$$d_i(x) - d_j(x) = 0 \tag{2.38}$$

Una buena práctica seria identificar la frontera de decisión entre dos clases, utilizando una sola función $d_{ij}(x) = d_i(x) - d_j(x) = 0$. Por lo tanto $d_{ij}(x) \succ 0$ para los patrones de clase w_i y $d_{ij}(x) \prec 0$ para los patrones de clase w_j. El objetivo principal de estos enfoques es desarrollar diversas técnicas para encontrar las funciones de decisión que satisfacen (2.37).

   a) **Coincidencia:** Las técnicas de reconocimiento basadas en la coincidencia sirven para representan cada clase en base a un patrón prototipo. A cada clase se le asigna un patrón desconocido, quien es el más cercano en términos de una métrica predefinida. Una de las técnicas más simples es aplicando el clasificador de distancia mínima, quien, como su nombre lo indica, calcula la distancia (Euclidiana) entre el patrón desconocido y cada uno de los vectores prototipo, eligiendo el prototipo con la distancia más pequeña. Otra de las técnicas basadas en coincidencia se basa en la correlación, quien se puede formular en términos de imágenes y resulta bastante intuitiva.

   b) **Clasificadores estadísticos óptimos:** Este tipo de enfoque de reconocimiento es probabilístico. Como ocurre en la mayoría de los campos que tienen que ver con la medición e interpretación de los fenómenos físicos, estas técnicas que consideran la probabilidad, han tomado gran importancia en el reconocimiento de patrones, debido a la aleatoriedad que se genera normalmente las clases de patrones. Por lo tanto, es posible derivar un enfoque de clasificación que es óptimo en el sentido de que se produce la menor probabilidad de cometer errores.

   c) **Redes neuronales:** Los dos enfoques analizados anteriormente se basan en el uso de patrones de muestra para estimar los parámetros estadísticos

de cada clase. El clasificador de distancia mínima se describe completamente por el vector medio de cada clase. De manera similar, el clasificador de Bayes para las poblaciones de Gauss se representan completamente por el vector medio y la matriz de covarianza de cada clase. En este enfoque los patrones utilizados para estimar los parámetros de forma general se denominan como patrones de entrenamiento, y el conjunto de patrones de cada clase se le denomina "conjunto de entrenamiento" y al proceso por el cual se utiliza un conjunto de entrenamiento para obtener las funciones de decisión se le conoce como aprendizaje o entrenamiento.

2. **Métodos estructurales:** Los enfoques de decisión teórica se basan en patrones cuantitativos y en gran parte ignoran las relaciones estructurales inherentes a la forma del patrón. Los métodos estructurales, sin embargo, tratan de lograr el reconocimiento de patrones mediante la capitalización precisamente de este tipo de relaciones. Este tipo de métodos requieren de autómatas o reconocedores apropiados para cada clase considerada. En situaciones sencillas, el autómata suele ser suficiente, pero en casos más complicados, puede ser necesario un algoritmo para el aprendizaje de los autómatas a base de patrones de muestra (tales como cadenas o árboles).

   a) **Coincidencia de formas:** Este tipo de técnicas son análogas a las técnicas de distancia mínima, donde los vectores patrón puede ser formulado por la comparación de fronteras de región que se describen en términos de números de forma. Por lo tanto el grado de similitud k entre dos límites de región, se define como el orden más grande para que sus números de forma aún coincidan. Por ejemplo, a y b denotan números de forma de contornos cerrados representados en 4-direcciones por códigos cadena.

   b) **Sintaxis de reconocimiento por cadenas:** Las técnicas de cadena tipo sintáctico proporcionan una metodología unificada para el manejo de problemas de reconocimiento estructural. Básicamente, la idea detrás del reconocimiento de patrones sintáctica es la especificación de un conjunto de patrones primitivos, un conjunto de reglas (tipo gramática) que rigen su interconexión, y un reconocedor sintáctico (llamado autómata), cuya estructura está determinada por el conjunto de reglas.

   c) **Sintaxis de reconocimiento por árboles:** Al igual que los métodos anteriores las técnicas de árbol tipo sintáctico proporcionan una metodología unificada para el manejo de problemas de reconocimiento estructural. Donde se especifican un conjunto de elementos primitivos para expresar una región u objeto de interés en forma de árbol, por medio de un conjunto de reglas, y un autómata.

Capítulo 3

Descripción de los algoritmos implementados

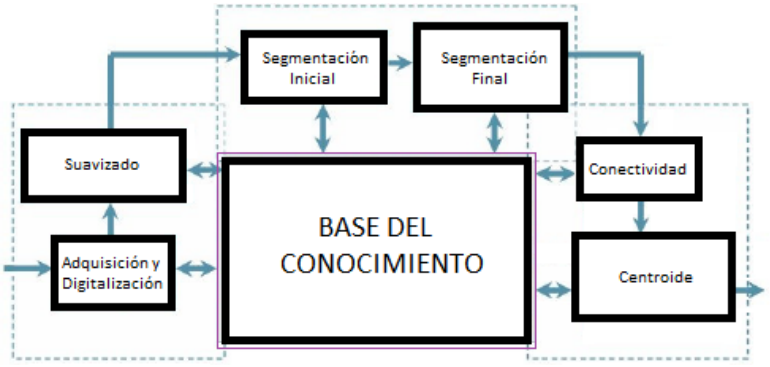


Figura 3.1: Diagrama algoritmos implementados.

Este capítulo pretende dar a conocer las diferentes técnicas a utilizar en las etapas de procesamiento, esto con el fin de obtener en tiempo real una correcta identificación e individualización de estrellas presentes en cúmulos estelares.

Para lograr el objetivo anterior se procedió a evaluar diferentes técnicas en cada una de las diferentes etapas, donde en cada etapa se midieron determinados parámetros con el fin de obtener la técnica más apropiada, factores como fácil implementación en la tarjeta, como además mejores tiempos de ejecución jugaron también un papel importante en la selección. Cabe señalar también que ninguna técnica es mejor que otra, solo en un contexto dado o ambiente controlado una tendrá un comportamiento más adecuado que otra.

En el algoritmo se busca dar solución a 4 etapas que se identificaron, teniendo en cuenta que la etapa de adquisición de la imagen no se aborda en este proyecto debido a que partimos de anteriores trabajos realizados en ese campo. Se inicia entonces determinando la técnica idónea para suavizar la imagen con el fin de reducir ruido, sigue con la etapa de segmentación donde se pretende separar los objetos de interés del fondo de la imagen, aquí se dividió esta etapa en dos secciones debido a que las

imágenes de cúmulos globulares presentan un comportamiento diferente en el centro (alta concentración de estrellas) con la relación a los alrededores (zonas menos densas), tercero se procede a buscar un algoritmo que permita identificar el área que comprende una estrella, y finalmente una técnica que permita individualizar dicha estrella de las demás.

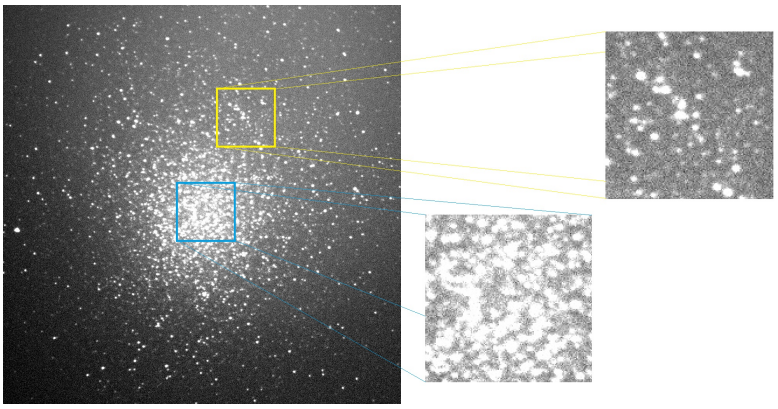


Figura 3.2: Imagen original del cúmulo Omega Centauri.

En la Figura 3.2 se observa el cúmulo Omega Centauri, con una resolución de 1024*1024*8 bits en escala de grises, en ella están demarcadas dos zonas, que se tendrán en cuenta durante todo este capítulo. la zona amarilla demarca una parte del exterior del cúmulo (poca concentración de estrellas) y la parte azul demarca el centro del cúmulo (gran cantidad de estrellas).

3.1. Suavizado de la imagen

Esta es la etapa inicial y pretende suavizar la imagen o disminuir el ruido o efectos negativos que se presentan como consecuencia del sistema de adquisición. Esta etapa requiere que cualidades propias como contraste, borde y tamaño de la estrella no sean alteradas ya que en técnicas posteriores se hará uso de estas características.

Para determinar la técnica adecuada, se realizaron pruebas con tres diferentes tipos de filtros en una sub-región de la imagen del cúmulo Omega Centauri o NGC-5139 de la constelación de Centaurus. Los tres filtros fueron el Gaussiano, la mediana y el promedio. A los tres tipos de filtros se le hicieron pruebas con kernels de diferentes tamaños 3*3, 5*5 y 7*7[13].

En la Figura 3.3 en el numeral *a*,*b* y *c*, se aborda el filtro del promedio con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *a* ventana 3+3 se observan muy buenos resultados a la hora de reducir ruido a pesar de ser una técnica muy básica, las estrellas pierden muy poco contraste aunque se ve pérdida de los bordes en las estrellas de la imagen. El tamaño de las estrellas prácticamente es el mismo. Esta técnica tomó un tiempo aproximado de 170 ms en ejecutarse. En la Imagen *b*, filtro promedio 5*5 se observa que se empiezan a perder los bordes,

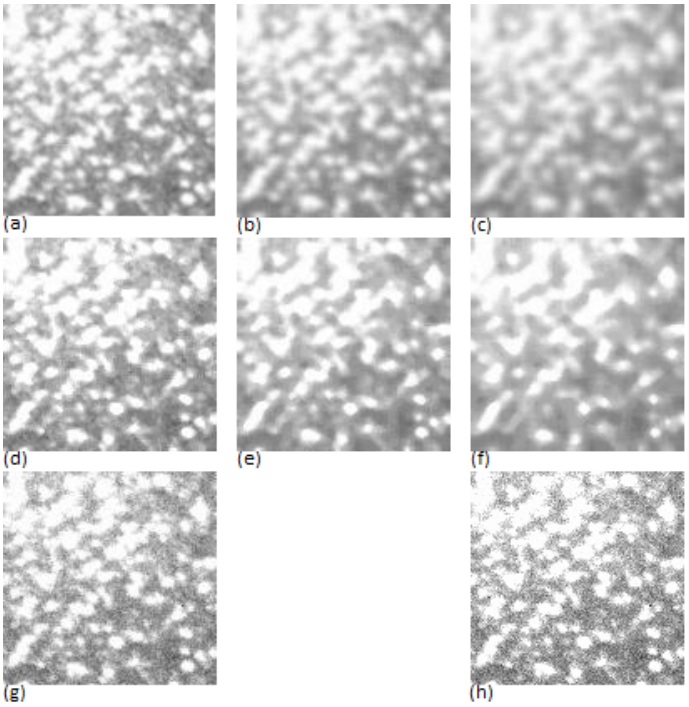


Figura 3.3: Técnicas de suavizado.

la imagen se difumina demasiado, el ruido se elimina por completo pero deja como consecuencia el aumento de tamaño de las estrellas. Esta técnica tomó un tiempo aproximado de 186 ms en ejecutarse. La Imagen *c*, filtro promedio 7*7 Elimina el ruido por completo, aumenta el tamaño de las estrellas hasta prácticamente fundir unas con otras, los bordes son prácticamente imperceptibles consecuencia de un muy bajo contraste resultante. Esta técnica tomó un tiempo aproximado de 209 ms en ejecutarse. Para el filtro del promedio en todas sus ventanas los tiempos de ejecución son muy bajos debido a sus operaciones sencillas y pocos cálculos requeridos.

En la Figura 3.3 en el numeral *d*, *e* y *f*, se aborda el filtro de la Mediana con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *d* ventana 3*3 se observan buenos resultados a la hora de reducir ruido, el contraste sigue siendo muy bueno con bordes muy notorios, pero el tamaño de las estrellas ha crecido en gran medida. Esta técnica se ejecutó en un tiempo aproximado de 126 ms. En la Imagen *e*, filtro mediana 5*5 se observa que el contraste y bordes siguen siendo muy buenos, el ruido prácticamente se ha llevado a cero y las estrellas siguen creciendo hasta fusionar algunas. Esta técnica se ejecutó en un tiempo aproximado de 148 ms. La Imagen *f*, filtro mediana 7*7 elimina el ruido por completo, aumenta el tamaño de las estrellas hasta prácticamente fundir unas con otras, los bordes y el contraste siguen siendo muy buenos. Esta técnica se ejecutó en un tiempo aproximado de 184 ms.

Se abordó una técnica un poco más compleja que prometía ser muy buena para la reducción de ruido como lo es el filtro Gaussiano Figura 3.3 numeral *g* donde se observa muy buen contraste que conlleva a tener unos muy buenos bordes, las es-

trellas prácticamente conservan su tamaño pero a diferencia de lo que se pensaba la reducción de ruido fue muy pobre. Esta técnica se ejecutó en un tiempo aproximado de 9 ms.

En general con los filtros de suavizado se obtuvieron tiempos muy similares, por lo que este aspecto no se hace relevante en la selección de la técnica a utilizar para la etapa de pre-procesado.

Después de analizar los resultados que brindan las diferentes técnicas de filtrado de imágenes se decide utilizar como etapa de pre-procesamiento el filtro del promedio 3*3 el cual permitió reducir el ruido casi por completo cualidad a la que se le dio más peso para esta etapa, además de no afectar el tamaño de las estrellas, motivo que hizo descartar casi de inmediato el filtro de la mediana, ya que es crítico que las estrellas se fusionen y no permita la correcta individualización en etapas posteriores. El filtro Gaussiano no cumplió con el principal objetivo que era la reducción de ruido.

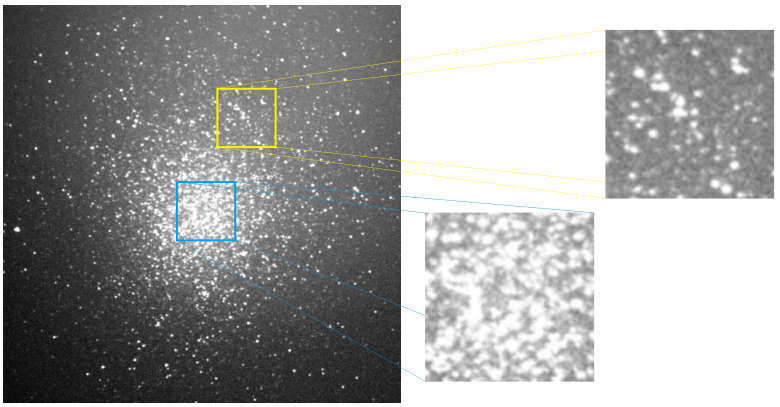


Figura 3.4: Suavizado del cúmulo Omega Centauri.

En la Figura 3.4 se observa los resultados finales de la técnica seleccionada del filtro promedio 3*3, este fue aplicado sobre la imagen del cúmulo globular Omega Centauri, con resultados especialmente buenos a la hora de reducir ruido, como también la conservación del tamaño original de las estrellas presentes en la imagen, además de generar cambios casi imperceptibles en el contraste y bordes. Esto quiere decir que el filtro promedio 3*3 demostró ser la técnica más adecuada para realizar la etapa de pre procesamiento en el algoritmo de identificación e individualización en tiempo real de estrellas presentes en cúmulos estelares. Se obtuvieron tiempos en Matlab de 9772,49 ms y en la tarjeta ADSP BF-533 de 846,7927 ms.

Para mayor entendimiento ver diagrama de flujo y código en Matlab y Visual en el Anexo .9.

3.2. Segmentación

Esta es la segunda etapa del proceso de identificación e individualización de estrellas presentes en cúmulos estelares, donde se pretende identificar los píxeles que hacen parte del objeto de interés y los píxeles que hacen parte del fondo.

Se dividió esta etapa en dos secciones, en la primera se pretendía además de la extracción de los objetos, hacer que los algoritmos posteriores fueran más rápidos, al encontrarse frente a una imagen más adecuada ya que todo píxel que no perteneciere a un objeto se lleva a cero. Con la segunda sección se pretendía atacar el centro de los cúmulos globulares donde hay alta concentración de estrellas, de manera que se lograra separar un poco unas de otras.

3.2.1. Segmentación inicial "Eliminando el fondo"

Es claro que las técnicas de segmentación pretenden separar los píxeles pertenecientes al objeto de los píxeles pertenecientes al fondo de una imagen, aquí adicionalmente se pretende llevar todo el fondo a un valor igual a cero, de manera que en técnicas posteriores estos píxeles puedan ser ignorados, permitiendo que dichos algoritmos se efectúen de manera más rápida.

En las técnicas de binarización comúnmente se utiliza un umbral para determinar que píxel tomará el valor de "1" ó que píxel tomará el valor de "0", aquí se usa un umbral global en la imagen en escala de grises para determinar que píxel tomará el valor de "0" o que píxel no se modifica si este pertenece a un objeto, cabe resaltar que aquí no se trabaja con imágenes binarias ya que se pretende sacar el mayor provecho de las características propias de las estrellas.

Debido al comportamiento de los cúmulos en especial globulares donde se observan grandes concentraciones de estrellas en el centro se determina que la imagen debe ser seccionada para realizar una mejor extracción, por lo que se realiza un análisis detallado de las diferentes técnicas de segmentación, se determina que es necesario usar una técnica global pero por regiones, debido a los cambios de intensidad que se detectan en las estrellas en diferentes zonas de la imagen, como también la luminosidad no homogénea que se denota en estas a causa del reflejo de la luna, con el fin de tener los mejores resultados de acuerdo a las características que presentan estas imágenes se toman regiones de 16*16.

Se plantea que la estrella más grande nunca debe superar el tamaño de la ventana seleccionada, para garantizar siempre tanto valores de fondo como de objeto en la ventana y así determinar el umbral adecuado, entonces para obtener el tamaño de las ventana apropiado, se seleccionaron visualmente alrededor de 100 estrellas aleatorias del cúmulo Omega Centauri, tanto del interior como del exterior, contemplando diferentes tamaños, en la Figura 3.5 se observan 8 de estas estrellas. Cabe recordar que esta tesis trabaja con imágenes de muy baja resolución por lo que se encontró que la estrella más grande ocupaba alrededor de 15*15 píxeles, por lo que

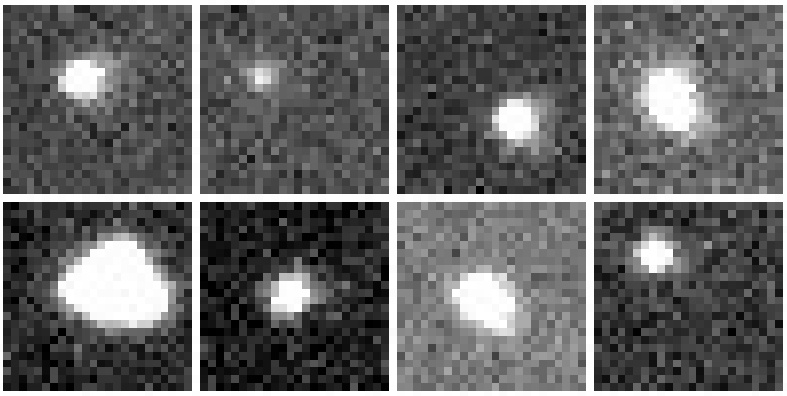


Figura 3.5: Determinando las ventanas.

una ventana 16*16 seria suficiente para abarcar estrellas de este tipo, si se desea aplicar este algoritmo en imágenes de mucha mayor resolución donde se puede encontrar estrellas de tamaños hasta de 128*128 píxeles o más, se debe ampliar esta ventana, también se recomienda que la ventana sea cuadrada perfecta, además se debe buscar un número que al aplicarlo como divisor del tamaño de la imagen el resultado de un número entero para facilitar el desarrollo del algoritmo $Imagen/Ventana = Entero$ $(1024/16 = 64)$. Este algoritmo se entiende como un acelerador ya que próximos algoritmos simplemente omitirán píxeles con valores iguales a cero para este algoritmo se obtuvieron tiempos de ejecución en Matlab de 806,95 ms y en la tarjeta ADSP BF-533 de 409,6837 ms.

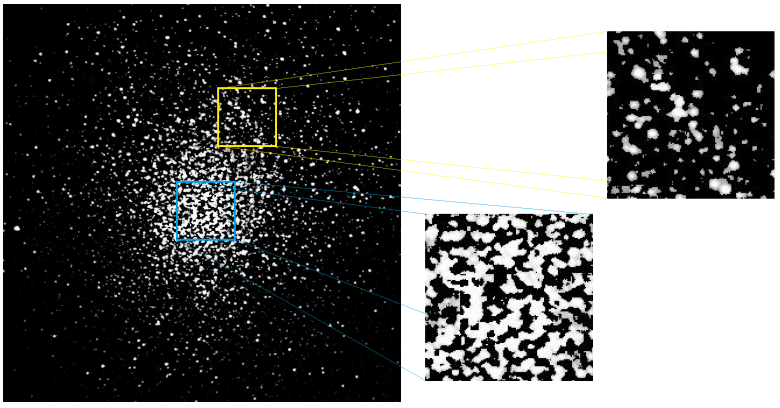


Figura 3.6: Segmentación inicial del cúmulo Omega Centauri.

Para mayor entendimiento ver diagrama de flujo y código en Matlab y Visual en el Anexo .12.

3.2.2. Segmentación final "Separando el centro"

Como se comentó en la sección anterior el objetivo principal de la técnica de segmentación es separar los píxeles pertenecientes al objeto de los píxeles pertenecientes al fondo de una imagen, pero esta técnica adicionalmente tiene por objeto

generar discontinuidades en las estrellas pertenecientes al centro de los cúmulos globulares, alterando lo menos posible las estrellas de las afueras o bordes del cúmulo, ya que en el centro se encuentran una gran cantidad de estrella con un alto nivel de luminancia y además se presentan estrellas traspuestas unas con otras, al igual que pequeños grupos de individuos que hacen parecer que fuesen un solo todo. Por ello se desea hallar una técnica capaz de percibir los pequeños cambios que se encuentren entre una estrella y otra para acentuar al máximo ese cambio, y así poder generar una separación, que pueda ser detectada en las subsiguientes etapas de descripción e interpretación.

Con el fin de obtener una buena segmentación acorde a los requerimientos de las imágenes de cúmulos estelares, se hace necesario evaluar diferentes métodos y así determinar el más adecuado para aplicar. Se realizaron pruebas con tres diferentes tipos de técnicas en una sub-región de la imagen del cúmulo Omega Centauri o NGC-5139 de la constelación de Centaurus. Las tres técnicas fueron Sauvola[15], máximos y mínimos[14], Niblack[16] y , se analizaron los tres métodos con kernel de tamaño 3*3, 5*5 y 7*7.

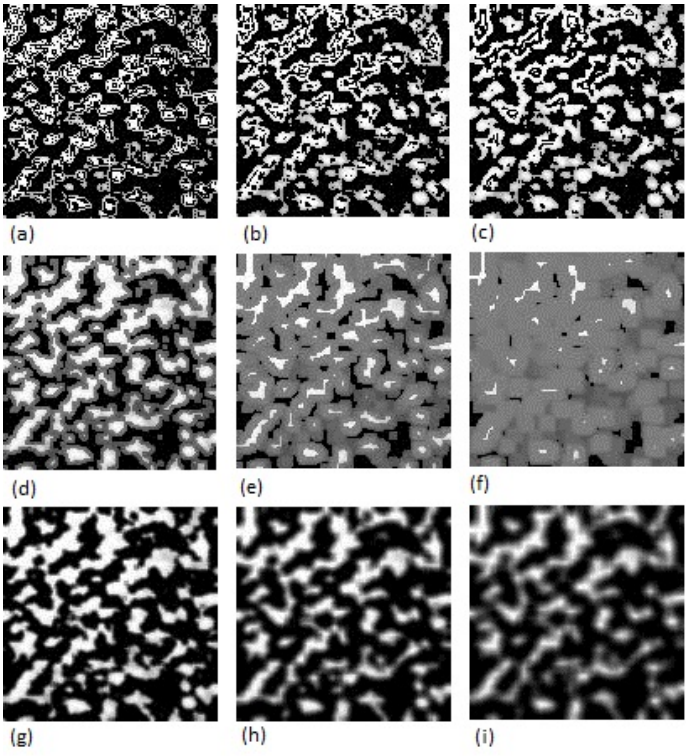


Figura 3.7: Técnicas de la segmentación final (centro).

En la Figura 3.7 en el numeral *a*, *b* y *c*, se aborda el método de Sauvola con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *a* ventana 3*3 se observa que las estrellas en el centro del cúmulo sufren una serie de discontinuidades, ya que las estrellas han perdido píxeles en los bordes generando separaciones entre ellas, las estrellas perdieron un poco de tamaño al igual que los bordes se vieron un

poco afectados y además el contraste no fue muy alterado. Esta técnica tomó un tiempo aproximado de 330 ms. En el numeral *b* donde se usa un kernel 5*5 se observa que las estrellas en el centro del cúmulo sufren discontinuidades, consiguiendo que las estrellas pierdan píxeles más hacia el centro y no en los bordes, lo cual es poco beneficioso a la hora de querer separar las estrellas, realmente las estrellas no pierden tamaño a pesar de perder algunos píxeles y los bordes no se ven afectados, además el contraste es poco alterado. Esta técnica tomó un tiempo aproximado de 342 ms. En el numeral *c* usando un kernel 7*7 se observa que las estrellas en el centro del cúmulo sufren algunas discontinuidades, consiguiendo que las estrellas pierdan píxeles directamente en el centro y no en los bordes, esto no presta ningún beneficio a la hora de pretender separar las estrellas, prácticamente las estrellas no pierden tamaño y además sus bordes se conservan al igual que el contraste. Esta técnica tomó un tiempo aproximado de 348 ms.

En la Figura 3.7 en el numeral *d*, *e* y *f*, se aborda el método de Máximos y Mínimos con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *d* ventana 3*3 se observa que las estrellas en el centro del cúmulo sufren un pequeño aumento de tamaño, ya que las estrellas ganan píxeles en los bordes generando un aumento de bordes, y además el contraste fue alterado medianamente. Esta técnica tomó un tiempo aproximado de 81 ms. En el numeral *e* donde se usa un kernel 5*5 se observa que las estrellas toman gran parte de sus píxeles como bordes desapareciendo mucho las estrellas lo cual es poco deseado, además pareciere que aumentaran su tamaño y afecta un poco el contraste. Esta técnica tomó un tiempo aproximado de 101 ms en ejecutarse. En el numeral *c* usando un kernel 7*7 se observa que las estrellas prácticamente desaparecen, los bordes aumentan al punto de formar cuadros de píxeles en tono gris de diferentes tamaños, además de afectar el contraste. Esta técnica tomó un tiempo aproximado de 120 ms en ejecutarse.

En la Figura 3.7 en el numeral *g*, *h* e *i*, se aborda el método de Niblack con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *g* ventana 3*3 se observa que las estrellas conservan su tamaño, los bordes se conservan y el contraste se afecta un poco, en términos generales no se ve ningún efecto que sirva para una segmentación en este tipo de imágenes. Esta técnica tomó un tiempo aproximado de 711 ms en ejecutarse. En el numeral *h* usando un kernel 5*5 se observa que las estrellas pierden algo de su tamaño, también los bordes se vuelven algo robustos y el contraste se ve afectado al punto de emborronar un poco la imagen. Esta técnica tomó un tiempo aproximado de 731 ms en ejecutarse. En el numeral *h* usando un kernel 7*7 se observa que las estrellas disminuyen mucho su tamaño, también los bordes se hacen más robustos y el contraste se ve afectado al punto de emborronar mucho la imagen. Esta técnica tomó un tiempo aproximado de 745 ms en ejecutarse.

En la Figura 3.8 perteneciente a los bordes del cúmulo en el numeral *a*, *b* y *c*, se aborda el método de Sauvola con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *a* donde se usa un kernel 3*3, se logra detectar que se pierden algunos píxeles en los bordes entre estrellas, ayudando a separar a las que quizás estén juntas, esto no es tan crítico ya que se pierden muy pocos píxeles. Esta técnica

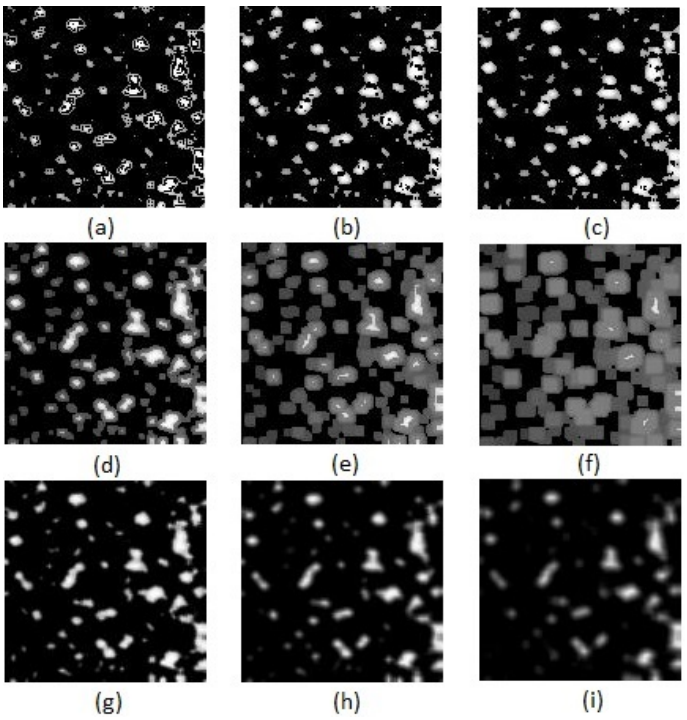


Figura 3.8: Técnicas de la segmentación final (bordes).

se ejecutó en un tiempo de 132 ms. En el numeral *b* con un kernel 5*5 se pierden algunos píxeles en las estrellas afectándolas muy poco, y en los casos de haber alguna juntas no se podría percibir. Esta técnica se ejecutó en un tiempo de 139 ms. Por último en el numeral *c* con un kernel 7*7 se puede observar que la técnica no generó ningún efecto conservando todas las características originales de la imagen. Esta técnica se ejecutó en un tiempo de 141 ms.

En la Figura 3.8 perteneciente a los bordes del cúmulo en el numeral *d*, *e* y *f*, se aborda el método de Máximos y mínimos con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *d* donde se usa un kernel 3*3, se logra detectar que las estrellas ganan algo de tamaño, también los bordes se hacen un poco más anchos, además el contraste se ve un poco afectado. Esta técnica se ejecutó en un tiempo de 79 ms. En el numeral *e* con un kernel 5*5 las estrellas se hacen mucho más grandes y en un tono gris como el de los bordes, además los bordes se hacen más ancho al punto de hacer casi desaparecer las estrellas. Esta técnica se ejecutó en un tiempo de 100 ms. En el numeral *f* con un kernel 7*7 las estrellas se hacen más grandes, además toman una forma cuadrada, esta forma que toman se debe al aumento del grosor en los bordes y a la adición de píxeles que se puede percibir, también el contraste se ve afectado un poco. Esta técnica se ejecutó en un tiempo de 120 ms.

En la Figura 3.8 perteneciente a los bordes del cúmulo en el numeral *g*, *h* y *i*, se aborda el método de Niblack con diferentes ventanas 3*3, 5*5 y 7*7 respectivamente. En el numeral *g* ventana 3*3 se observa que las estrellas conservan su tamaño, los bordes se conservan y el contraste se afecta un poco, en términos generales no se ve

ningún efecto que sirva para una segmentación en este tipo de imágenes. Esta técnica tomó un tiempo aproximado de 699 ms en ejecutarse. En el numeral h usando un kernel 5*5 se observa que las estrellas pierden algo de su tamaño, también los bordes se vuelven algo robustos y el contraste se ve afectado al punto de emborronar un poco la imagen. Esta técnica tomó un tiempo aproximado de 721 ms en ejecutarse. En el numeral h usando un kernel 7*7 se observa que las estrellas disminuyen mucho su tamaño, también los bordes se hacen más robustos y el contraste se ve afectado al punto de emborronar mucho la imagen. Esta técnica se ejecutó en un tiempo de 740 ms.

En general con las técnicas de segmentación se obtuvieron tiempos muy diferentes, aunque el tiempo es importante no se tomó como un factor relevante para la elección de la técnica más adecuada para la segmentación, ya que lo realmente importante es que cumpla con los requerimiento necesarios para obtener una buena segmentación, además los tiempos obtenidos en las diferentes técnicas no son críticos acorde a las necesidades.

Después de analizar los resultados que brindan las diferentes técnicas de segmentación de imágenes se decide utilizar en la etapa de segmentación final la técnica de Sauvola 3*3 gracias a que esta genera discontinuidades entre las estrellas, esto sucede debido al alto nivel de sensibilidad que esta técnica presenta ante los pequeños cambios entre los niveles de luminancia de las estrellas, y de esta forma permitiendo obtener una segmentación notable en la parte central de los cúmulos globulares, dicha cualidad hace que esta etapa de mucho peso en el proceso para cumplir el objetivo principal de esta tesis. A pesar que de las tres técnicas la que mejores tiempos conseguía era la de máximos y mínimos, no lograba conseguir una buena separación entre las estrellas del centro de los cúmulos estelares y además alteraba mucho el tamaño de los bordes, la técnica de Niblack se pudiese decir que en este tipo de segmentaciones prácticamente funciona como un filtro, ya que solo conseguía emborronar las imágenes a medida que se aumentaba el tamaño del kernel, afectando el contraste y la nitidez de la imagen. Sin embargo, Sauvola disminuye su rendimiento cuando la imagen tiene bajo contraste o exceso de ruido como el que se puede presentar en este tipo de imágenes astronómicas.

Sauvola presenta un algoritmo para la binarización de imágenes usando un enfoque adaptable para gestionar diferentes situaciones en una imagen. La técnica propuesta utiliza un análisis rápido de la imagen para la selección y adaptación del algoritmo de acuerdo con el contenido de esta y este algoritmo tardó un tiempo en Matlab de 4,23395 s para ejecutarse mientras que en la tarjeta de desarrollo ADSP BF-533 el tiempo fue de 2,233518 s.

Para mayor entendimiento ver diagrama de flujo y código en Matlab y Visual en el Anexo .15.

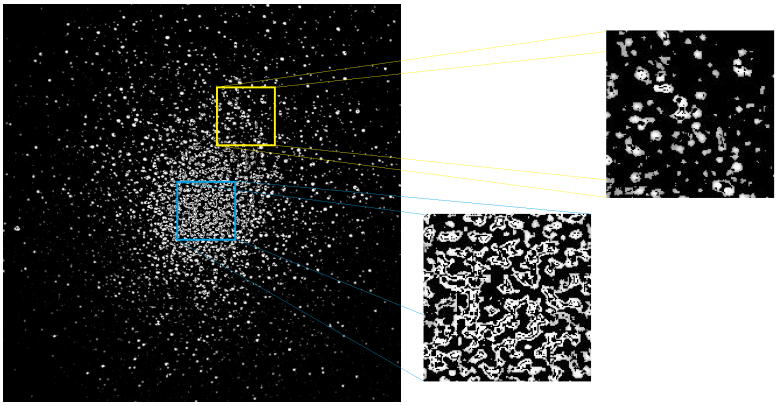


Figura 3.9: Segmentación final del cúmulo Omega Centauri.

3.3. Conectividad

Hasta ahora las técnicas han trabajado sobre el contraste, bordes, brillo entre otras propiedades básicas de la imagen sin tener en cuenta las características y patrones de comportamiento propios que caracterizan las estrellas. Es por esta razón que en esta etapa se pretende realizar el proceso clave para la identificación de las estrellas en cúmulos estelares, a partir de analizar este tipo de patrones únicos que suelen tener las estrellas en cuanto a su forma, tamaño y desvanecimiento en sus niveles de luminancia. Es por este motivo que en las anteriores etapas se obtuvo como salida una imagen, mientras que en esta etapa se obtendrán una serie de coordenadas que representaran el contorno donde se encuentran las estrellas[8].

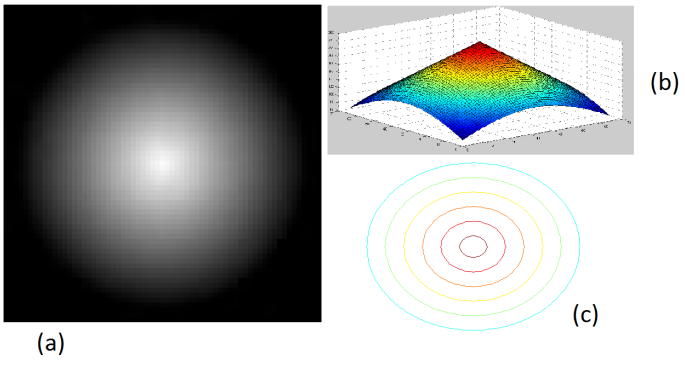


Figura 3.10: Estrella ideal.

En la Figura 3.10 numeral (a) se observa una estrella ideal, en (b) como se vería esta en un plano de 3-dimensiones, y en (c) se lleva a un plano en 2-dimensiones, de la ultima figura numeral (c), se nota como la estrella va perdiendo brillo a medida que se aleja del centro que es el punto de mayor luminosidad, a continuación se hace un estudio similar al efectuado en la etapa de segmentación donde se detalló el máximo tamaño para estrellas del cúmulo Omega Centauri en una imagen de tamaño 1024*1024, pero en esta etapa se analizara el comportamiento de los píxeles

a medida que se alejan del centro de las estrellas.

7,1	6,4	5,8	5,4	5,1	5	5,1	5,4	5,8	6,4	7,1
6,4	5,7	5	4,5	4,1	4	4,1	4,5	5	5,7	6,4
5,8	5	4,2	3,6	3,2	3	3,2	3,6	4,2	5	5,8
5,4	4,5	3,6	2,8	2,2	2	2,2	2,8	3,6	4,5	5,4
5,1	4,1	3,2	2,2	1,4	1	1,4	2,2	3,2	4,1	5,1
5	4	3	2	1	0	1	2	3	4	5
5,1	4,1	3,2	2,2	1,4	1	1,4	2,2	3,2	4,1	5,1
5,4	4,5	3,6	2,8	2,2	2	2,2	2,8	3,6	4,5	5,4
5,8	5	4,2	3,6	3,2	3	3,2	3,6	4,2	5	5,8
6,4	5,7	5	4,5	4,1	4	4,1	4,5	5	5,7	6,4
7,1	6,4	5,8	5,4	5,1	5	5,1	5,4	5,8	6,4	7,1

Figura 3.11: Radios al píxel central.

Para el análisis posterior se tendrán en cuenta la distancia entre el píxel central y sus alrededores, para ello se formaron niveles que relacionan una serie de píxeles de acuerdo a la distancia hasta el centro, ver la Figura 3.11.

En base a lo anterior para realizar una correcta descripción de las estrellas, se elaboró un trabajo estadístico para determinar el comportamiento característico de las estrellas, para el desarrollo de dicho estudio se tomaron de modo aleatorio 60 estrellas contenidas en la imagen del cúmulo globular Omega Centauri, la toma de las estrellas se realizó de modo subjetivo de acuerdo al brillo de las mismas, esto con el fin de garantizar que se contase con una gran variedad de estrellas y poder hallar el comportamiento que mejor represente a la mayoría. Para efectos de este documento solo se analizarán dos estrellas, una perteneciente al centro del cúmulo (donde se espera tener niveles muy similares por su alto brillo y su cercanía con otras estrellas que pueden sumar brillo a la estrella patrón), y otra estrella perteneciente a los bordes del cúmulo (donde se espera tener una estrella adecuada, debido a que se encuentra aislada del resto).

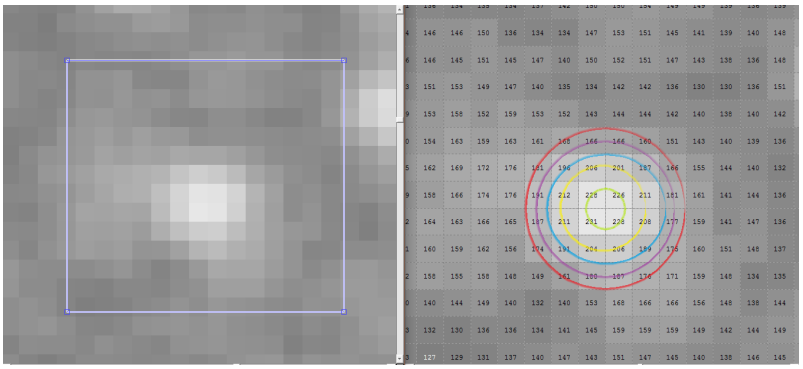


Figura 3.12: Estrella aleatoria del centro del cúmulo.

En la Figura 3.12 se demarcan 5 niveles con los siguientes promedios 228, 207, 191, 179, 170, donde se observa unos porcentajes de acuerdo al primer nivel de 100 %,

90,78 %, 83,77 %, 78,5 %, 74,56 %, lo que da como resultado una caída de intensidad o brillo promedio entre niveles de 6,36 %, además del primero al quinto nivel se tiene una caída superior al 25 %.

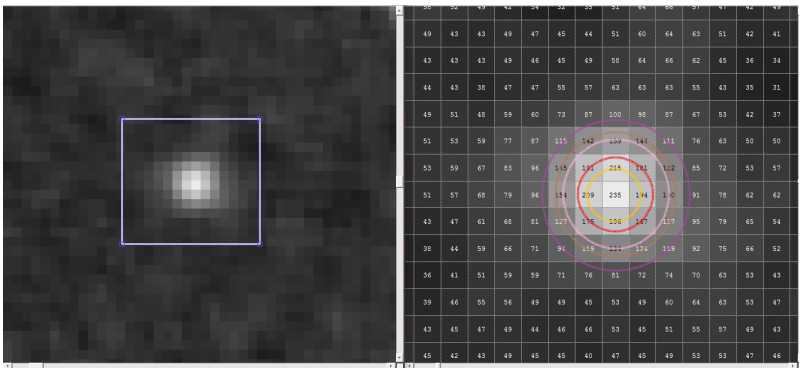


Figura 3.13: Estrella aleatoria del borde del cúmulo.

En la Figura 3.13 se demarcan 6 niveles con los siguientes promedios 235, 203, 178, 147, 132, 107, donde se observa unos porcentajes de acuerdo al primer nivel de 100 %, 86,38 %, 75,74 %, 62,55 %, 56,17 %, 45,53 %, lo que da como resultado una caída de intensidad o brillo promedio entre niveles de 10,89 %, además del primero al quinto nivel se tiene una caída superior al 25 %.

Gracias al análisis estadístico se concluye entonces que cada estrella independiente de su lugar de ubicación siempre cumple con una caída superior al 25 % entre su primer nivel y el quinto. Además se propone que una estrella para ser catalogada como tal debe contar al menos con cinco niveles o lo que es lo mismo una estrella de aproximadamente 5*5 píxeles donde este quinto píxel podría ser el fondo.

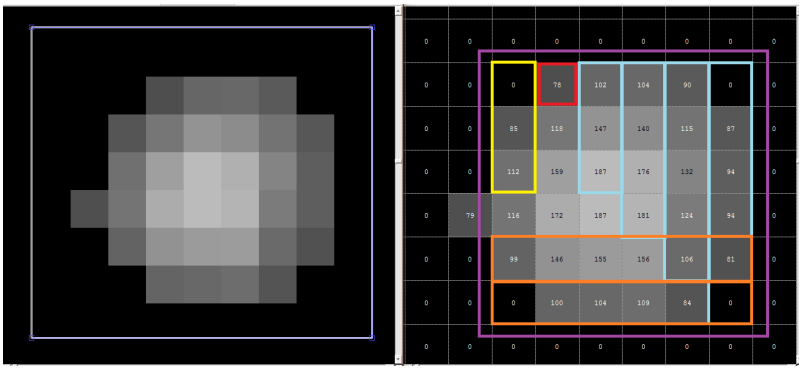


Figura 3.14: Método de representación.

En la implementación del algoritmo no se conoce el píxel central o píxel de nivel 1, por lo que básicamente se recorre la imagen de izquierda a derecha y de arriba a abajo buscando un píxel diferente de cero (aprovechando el trabajo realizado por los algoritmos pasados). Cuando este es encontrado se detiene a analizar todos sus vecinos de acuerdo al siguiente método, ver Figura 3.14.

1. ***Derecha:*** A partir del píxel encontrado (recuadro rojo), se desplaza un píxel a la derecha en la imagen, iniciando con tres píxeles. Luego se desplaza nuevamente un píxel a la derecha y se aumenta un píxel verticalmente, este proceso se repite siempre y cuando no se cumpla con el criterio de caída del 25 % (recuadros azules).

2. ***Izquierda:*** Se repite el proceso realizado a la derecha pero esta vez se realiza a la izquierda (recuadros amarillos).

3. ***Abajo:*** A partir de los dos pasos anteriores se tienen las coordenadas máximas izquierdas y derechas y simplemente se mueve en la imagen hacia abajo respetando estos límites hasta encontrar el criterio de caída del 25 % (recuadros naranjas).

4. ***Coordenadas:*** Se almacenan la coordenada superior izquierda del recuadro violeta, junto con la longitud máxima horizontal (para el ejemplo = 6), y la longitud máxima vertical (para el ejemplo = 6).

5. ***Limpiar:*** Con el fin de no realizar el mismo procedimiento sobre este grupo de píxeles, se deben llevar a valor de cero todos los pertenecientes al recuadro violeta.

Después del anterior método se obtiene una serie de coordenadas (píxel superior izquierdo, longitud en x, longitud en y) que representan el área que podría demarcar una estrella.

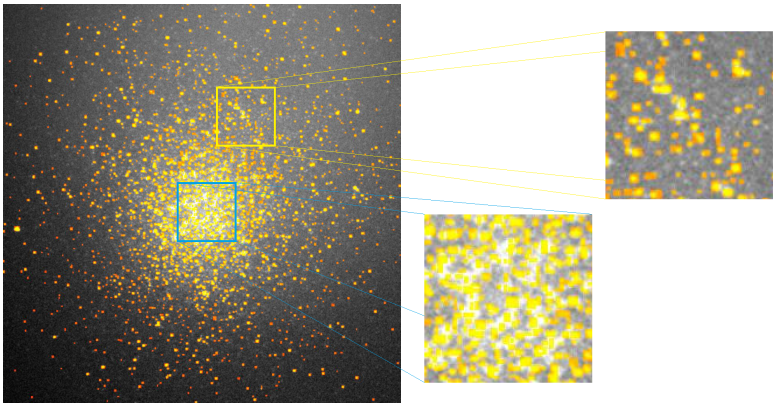


Figura 3.15: Conectividad de las estrellas en el cúmulo Omega Centauri.

Para el fácil entendimiento humano se han traspuesto los recuadros encontrados en la imagen original suavizada (Figura 3.15), recordemos que este tipo de algoritmos no dan como salida una imagen sino un tipo de datos que representan o describen el objeto. Este algoritmo tardó un tiempo en Matlab de 551 ms para ejecutarse mientras que en la tarjeta de desarrollo ADSP BF-533 el tiempo fue de 181 ms

Para mayor entendimiento ver diagrama de flujo y código en Matlab y Visual en el Anexo .18.

3.4. Centroide

Ahora partiendo de la etapa anterior donde se obtuvieron las coordenadas de dirección del contorno donde se encuentra representadas las estrella, se pretende obtener el centro de gravedad o de masa de cada estrella. Esto con el fin de obtener una coordenada que permita individualizar las estrellas en las imágenes de cúmulos estelares[8].

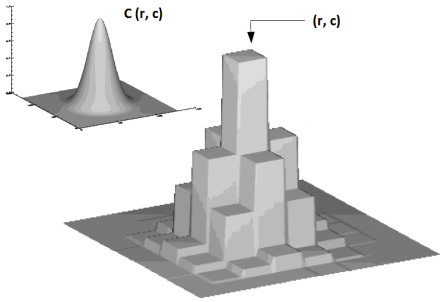


Figura 3.16: Imagen centroide.

Centroide: Para calcular el centroide generalmente se usa el método del centro de gravedad, quien es el más simple y directo para calcular la posición del centro en una circunferencia simétrica.

$$\hat{X}_{CoG} = \frac{\sum x I_{x,y}}{\sum I_{x,y}} \tag{3.1}$$

Para explicar de forma más clara la anterior ecuación, en (3.2), (3.3) se muestra como se aplica (3.1) para las dos componentes en r y c:

$$C_r = \sum_{r=1}^{m} r \sum_{c=1}^{n} Imagen(r,c) / \sum_{r=1}^{m} \sum_{c=1}^{n} Imagen(r,c) \tag{3.2}$$

$$C_c = \sum_{c=1}^{n} c \sum_{r=1}^{m} Imagen(r,c) / \sum_{r=1}^{m} \sum_{c=1}^{n} Imagen(r,c) \tag{3.3}$$

Para observar como se aplica (3.2), (3.3) obsérvese el ejemplo de la Figura 3.17:

En la Figura 3.17 se observa que para realizar el cálculo del centroide se efectúa una sumatoria de los valores correspondientes a cada fila, que a la vez se multiplica por su número de índice correspondiente según la columna de ubicación en la que se encontrase. Luego se procede a realizar una sumatoria de los valores obtenidos y posteriormente se divide por la suma ponderada de los valores correspondientes a lo píxeles de dicha imagen. Así de esta forma se obtiene el valor correspondiente de coordenada en r. De igual forma se realiza lo respectivo para las columnas dando como resultado la coordenada en c, consiguiéndose así la coordenadas (r, c) donde se encuentra el centro de masa de la estrella en la imagen. Este método fue muy

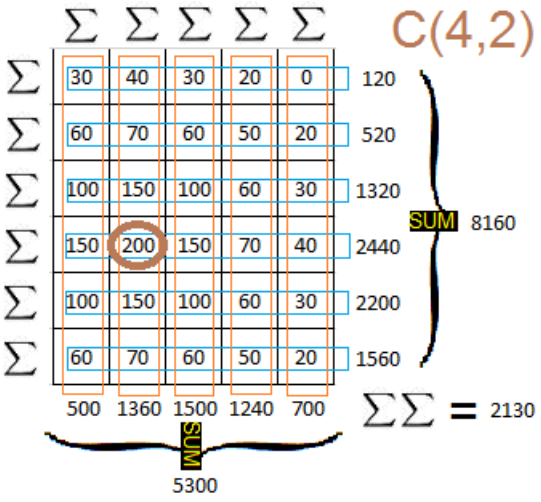


Figura 3.17: Ejemplo centroide.

efectivo para determinar el centro en las estrellas presentes en imágenes astronómicas, puesto que las estrellas ofrecen factores que facilitan el trabajo como lo son; su comportamiento, variabilidad entre los niveles de luminancia y forma simétrica.

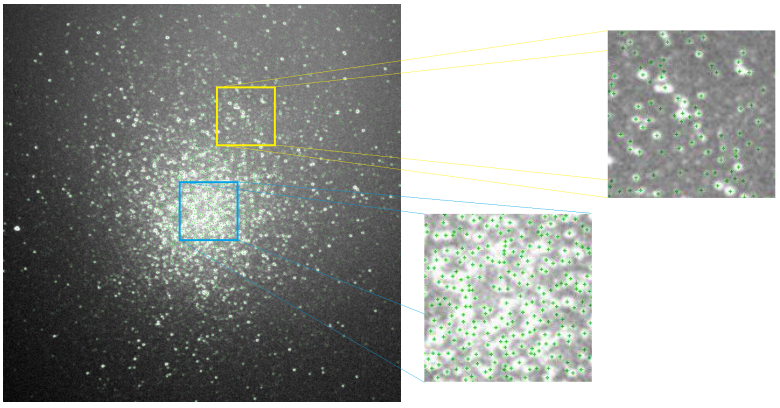


Figura 3.18: Centroide de las estrellas en el cúmulo Omega Centauri.

Para el fácil entendimiento humano se ha dibujado una cruz de color verde en las coordenadas donde se halló el centro y se ha transpuesto en la imagen original suavizada (Figura 4.1), recordemos que este tipo de algoritmos no dan como salida una imagen sino un tipo de datos que representan o describen el objeto. Este algoritmo tardó un tiempo en Matlab de 115 ms para ejecutarse mientras que en la tarjeta de desarrollo ADSP BF-533 el tiempo fue de 23 ms.

Para mayor entendimiento ver diagrama de flujo y código en Matlab y Visual en el Anexo .21.

Capítulo 4

Resultados obtenidos

Este capítulo pretende mostrar el rendimiento en general de todos los algoritmos implementados en la tarjeta ADSP BF-533 para la correcta identificación e individualización de estrellas en imágenes de cúmulos estelares en tiempo real. Para esto se realizaron pruebas sobre tres imágenes de cúmulos estelares de 1024*1024 píxeles de tamaño en escala de grises. Las imágenes pertenecen al cúmulo Omega Centauri o NGC 5139 (cúmulo globular situado en la constelación de Centaurus), al cúmulo NGC 6819 (cúmulo abierto situado en la constelación de Cygnus) y por último al cúmulo M22 o NGC 6656 (cúmulo globular situado en la constelación de Sagitario).

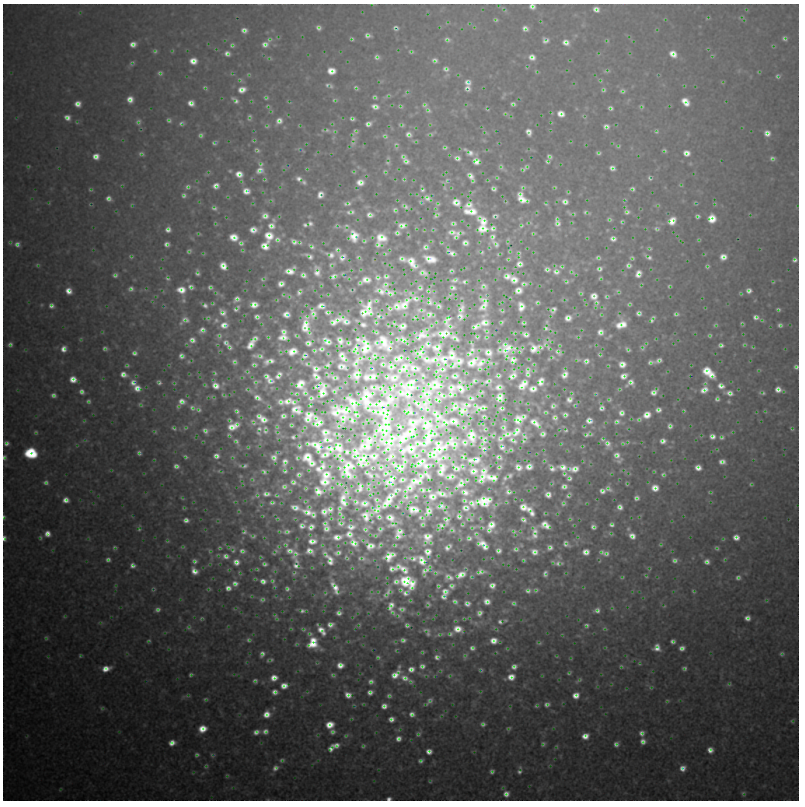


Figura 4.1: Cúmulo globular Omega Centauri.

En la Figura 4.1 se evidencian los resultados obtenidos sobre la imagen del cúmulo Omega Centauri en la cual se logran identificar con la DSP Blackfin BF-533 una cantidad de 2435 estrellas en un tiempo total de 4,010 s, mientras que en Matlab se obtuvieron un total de estrellas identificadas de 2376 en un tiempo total de 15,12855 s. En relación a la DSP, Matlab presentó un 2,42 % menos de identificación de estrellas.

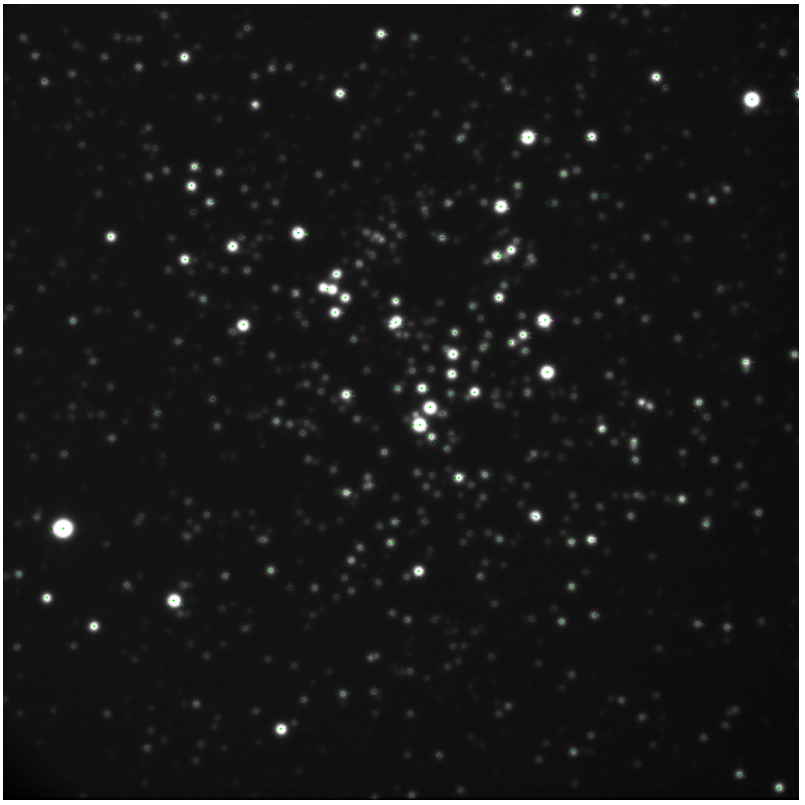


Figura 4.2: Cúmulo abierto NGC 6819.

En la Figura 4.2 se evidencian los resultados obtenidos sobre la imagen del cúmulo NGC 6819 en el cual se logran identificar una cantidad de 465 estrellas en un tiempo total de 2,584 s, mientras que en Matlab se obtuvieron un total de estrellas identificadas de 453 en un tiempo total de 11,67993 s, En relación a la DSP, Matlab presentó un 2,58 % menos de identificación de estrellas.

En la Figura 4.3 se evidencian los resultados obtenidos sobre la imagen del cúmulo M22 en el cual se logran identificar una cantidad de 656 estrellas en un tiempo total de 3,045 s, mientras que en Matlab se obtuvieron un total de estrellas identificadas de 646 en un tiempo total de 12,6031 s. En relación a la DSP, Matlab presentó un 1,52 % menos de identificación de estrellas.

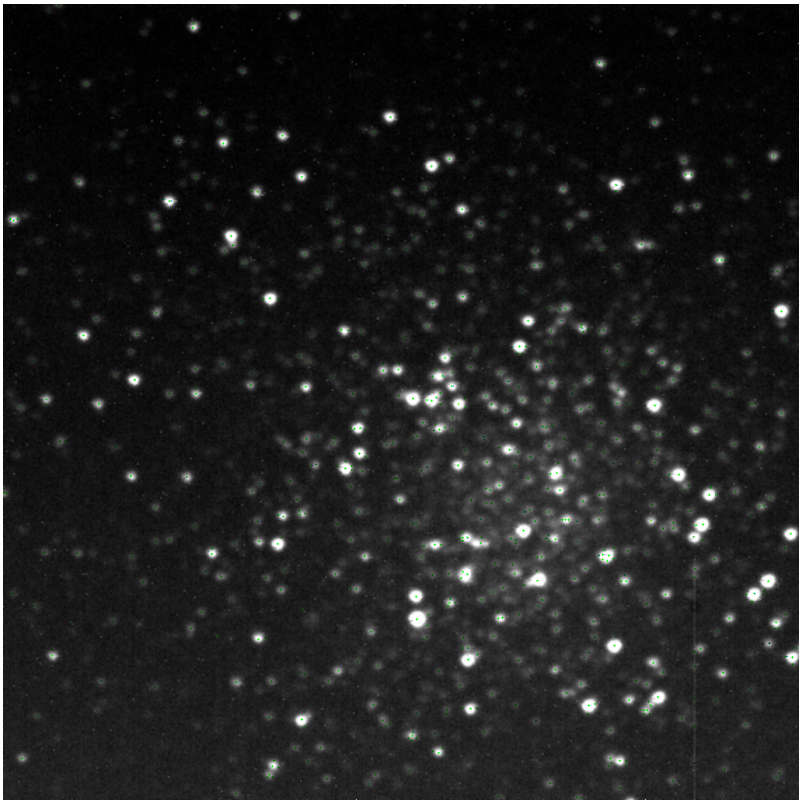


Figura 4.3: Cúmulo globular M22.

Cúmulo	**Estrellas Visual DSP**	**Estrellas Matlab**	**% Diferencia de Detección**
NGC-5139	2435	2376	2,42
NGC-6819	465	453	2,58
NGC-6656	656	646	1,52

Tabla 4.1: Cantidad de estrellas detectadas

Cúmulo	**Tiempo Visual DSP**	**Tiempo Matlab**	**% Reducción de Tiempo**
NGC-5139	4,010 s	15,128 s	73,4938
NGC-6819	2,584 s	11,6799 s	77,8753
NGC-6656	3,045 s	12,6031 s	75,8392

Tabla 4.2: Reducción de tiempo entre plataformas

Capítulo 5

Conclusiones, aportes y recomendaciones

5.1. Conclusiones

El algoritmo implementado sobre la ADSP BF-533 demostró ser adecuado para una correcta identificación e individualización de estrellas, lo cual se logra evidenciar en las imágenes de prueba a través del etiquetamiento de las estrellas en los cúmulos estelares.

Al iniciar este proyecto se deseaba contar con un sistema capaz de identificar e individualizar estrellas presentes en cúmulos estelares utilizando los recursos disponibles de la Universidad Tecnológica de Pereira, además de que se pudieran analizar resultados dentro de la misma noche de observación (que son tan difíciles de conseguir), por lo tanto el sistema desarrollado se ejecutó en tiempo real debido a que se obtuvieron resultado en unos cuantos segundos, tiempo poco relevante si se considera que en una noche de observación se puede contar con algunas horas para el análisis de resultados.

La tarjeta ADSP BF-533 demostró ser una herramienta que trabaja en tiempo real, esto se debe en gran medida a que es un sistema dedicado, de que cuenta con opciones de ser programada, tanto en lenguaje C++, como en lenguaje ensamblador al mismo tiempo, lo que permite al desarrollador realizar funciones complejas en C++ o en ensamblador cuando el tiempo sea un factor crítico, adicionalmente la DSP cuenta con un entorno de desarrollo que permite optimizar los tiempos de ejecución gracias a sus múltiples herramientas de depuración, mientras que un computador de propósito general debe realizar múltiples tareas de forma simultánea durante su operación, desfasando los tiempos de ejecución de una tarea especifica. Incluso esto se nota a la hora de medir los tiempos de ejecución de los algoritmos realizados en Matlab, donde se tubo que recurrir a métodos que ayudaran a obtener un tiempo más o menos correcto.

Se considera que la diferencia de tiempos entre ambas herramientas no es tan significativo si se habla de tiempo real, cabe resaltar que el sistema desarrollado es

tan solo una etapa del macro proyecto de investigación astronómica del grupo Alfa Orión donde se encuentran ya otros sistemas implementados bajo esta plataforma, por lo que se hace indispensable el uso de esta.

5.2. Aportes

El principal aporte de esta investigación fue generar un sistema capaz de identificar e individualizar estrellas presentes en imágenes de cúmulos estelares en tiempo real.

Esta tesis servirá como herramienta para futuros investigadores, así como para el grupo de investigación Alfa Orión de la Universidad Tecnológica de Pereira que deseen examinar características especificas de las estrellas presentes en cúmulos que antes eran imposibles debido a que se veían el cúmulo como un solo todo.

Se aportan una guía sobre el manejo de la ADSP BF-533, su set de instrucciones y entorno de programación que servirá de base para futuros investigadores que deseen incursionar en el manejo de sistemas dedicados, para que de esta manera puedan enfrentarse a este tipo de herramientas sin temor a equivocarse (revisar anexos).

Se aporta un nuevo enfoque de manejo de la programación de la DSP-BF533 donde se saca el máximo provecho a la utilidad de poder mezclar diferentes tipos de lenguajes, donde C++ se utiliza para funciones difíciles de implementar y ensamblador en funciones donde el tiempo se vuelve un factor crítico. Esto se hace evidente ya que anteriores investigadores se veían muy reacios a incursionar en lenguajes más complejos como el ensamblador.

Se propicia el desarrollo de investigaciones astronómicas sobre los cúmulos estelares en la Universidad Tecnológica de Pereira, así como en el país en general. Lo anterior ya que existen pocos estudios sobre el análisis astronómico, en el país y más específicamente el observatorio de la Universidad.

5.3. Recomendaciones

Se recomienda a futuros investigadores explorar al máximo las instrucciones en lenguaje ensamblador, ya que este brinda una cantidad de herramientas que facilitan la optimización de los algoritmos además de tiempos muy reducidos de ejecución que se vuelven indispensables cuando se tienen funciones donde el tiempo es un factor muy crítico.

Se recomienda seguir generando proyectos desde la ingeniería que estén encaminados en el área astronómica y de procesamiento digital de imágenes que permitan seguir avanzando el el macro proyecto del Grupo de Investigación en Astroingeniería Alfa Orión de la Universidad Tecnológica de Pereira.

A partir de este trabajo se pueden desarrollar diferentes aplicaciones a futuro, como por ejemplo realizar un algoritmo de corroboración de datos obtenidos, a partir de las zonas etiquetadas e implementando algoritmos que faciliten de la detección de diferentes característica de las estrellas.

Glosario

UART: (Universal Asynchronous Receiver-Transmitter).

PWM: Temporizadores con PWM (Modulación por Ancho de Pulso).

Watchdog Timer: Temporizador de vigilancia.

Imagen: Una imagen puede ser definida como una función de dos dimensiones $f(x, y)$, donde x e y son coordenadas espaciales, y la amplitud de f en cualquier par de coordenadas (x, y) se llama intensidad o nivel de gris de la imagen en ese punto. Donde x, e y los valores de amplitud de f son todos cantidades finitas y discretas, que se pueden llamar como imagen o imagen digital. Tenga en cuenta que una imagen digital se compone de un número finito de elementos, cada uno de los cuales tiene un valor y localización particular, cada elemento es llamado Píxel o Pel denominado del término "Picture Element". Píxel es el término más utilizado para referirse a los elementos de una imagen de una cámara digital.[8]

Procesamiento de Imágenes: A pesar de que el término "procesamiento de imágenes"a menudo se utilizan indistintamente con el de "edición de imágenes", se introducen las siguientes definiciones más precisas. Edición de imágenes digitales, o, como a veces se hace referencia, imagen digital, es la manipulación de imágenes digitales mediante una aplicación de software existente, como Adobe Photoshop o Corel Paint. Procesamiento de imágenes digitales, por otro lado, es la concepción, el diseño, el desarrollo y la mejora de los algoritmos de procesamiento digital de imágenes.[18]

Umbralización: Es uno de los métodos que abarca la segmentación de imágenes digitales. Las técnicas de umbralización global buscan obtener un valor de umbral que permita binarizar a la imagen separando adecuadamente el fondo (background) y el objeto a separar (foreground). Muchas de las técnicas de umbralización están basadas en la información estadística que brinda el histograma, sobre todo en aquellas imágenes donde los objetos tienen una superficie o textura homogénea y el fondo es más o menos uniforme. El problema de la umbralización es encontrar el valor T (umbral) adecuado entre los valores de grises.[4]

Sistema Embebido: Los sistemas embebidos o empotrados son generalmente parte de sistemas más grandes y complejos y suelen aplicarse en hardware dedicado con software asociado para formar un motor de cálculo que eficientemente

realice una función específica. El hardware dedicado (o procesador embebido) con el software asociado está integrado en muchas aplicaciones. A diferencia de computadoras de propósito general, que están diseñados para realizar muchas tareas generales, un sistema embebido es un sistema informático especializado que suele ser integrado como parte de un sistema mayor. Por ejemplo, una cámara digital toma de una imagen, y el procesador incorporado en el interior de la cámara comprime la imagen y la almacena en la memoria. En algunas aplicaciones de instrumentos médicos, el procesador integrado está programado para registrar y procesar datos médicos tales como la frecuencia del pulso y la presión sanguínea y utiliza esta información para controlar un sistema de soporte del paciente.[21]

DSP: El término DSP se aplica a cualquier chip que trabaje con señales representadas de forma digital, Los DSP o procesadores digitales de señal son microprocesadores específicamente diseñados para el procesado digital de señal. Algunas de sus características más básicas como el formato aritmético, la velocidad, la organización de la memoria o la arquitectura interna hacen que sean o no adecuados para una aplicación en particular, así como otras que no hay que olvidar, como puedan ser el coste o la disponibilidad de una extensa gama de herramientas de desarrollo.

Cúmulos Estelares: Son condensaciones locales de estrellas unidas por fuerzas gravitacionales. Comparten características similares como su composición química y edad relativa, debido a que se forman a partir de la misma nube de gas o nebulosa. En nuestra galaxia los astrónomos las han clasificado en dos grandes grupos:

Los cúmulos abiertos se encuentran en el disco galáctico, y están caracterizados por ser relativamente jóvenes y por poseer una densidad estelar cien veces más elevada que la que se encuentra en las regiones que rodean al Sol; sin embargo, las estrellas que los componen están relativamente dispersas. El diámetro medio de los cúmulos abiertos es de aproximadamente 10 años-luz y el número de estrellas que contienen varía desde algunas decenas a algunos miles.

Los cúmulos globulares están caracterizados por una elevada densidad estelar y por una alta concentración de estrellas en la parte central del cúmulo, hasta el punto que en muchos casos resulta imposible, incluso con un potente telescopio, distinguir cada una de las estrellas presentes en dicho centro. El diámetro medio de los cúmulos globulares es de aproximadamente 70 años-luz.

Centroide El algoritmo de promediado estadístico (por ejemplo el centro de masa) es el más utilizado de acuerdo a la definición de Centroide. El método convencional del algoritmo para la detección de centroide se basa en el cálculo del centro de masa , y es conocido como el algoritmo de peso promediado de píxeles (PPP) y a partir de éste método se han desarrollado los métodos de: PPP de umbral (PPPU), en el cual se utiliza el umbral para mejorar la razón de señal a ruido (SNR) y el método PPP

de potencias (PPPP), en el cual el valor de la razón de señal a ruido se mejora mediante el uso de potencias. Además, se han utilizado métodos adaptativos de detección de centroide mediante un método de filtrado morfológico sobre la imagen binaria.

Bibliografía

[1] Nobiyuki Otsu: *A Threshold Selection Method from Gray Level Histograms*, Tokyo Japan Texas A M University, IEEE 1979

[2] Alejandro Cristo, Antonio Plaza, David Valencia: *A novel thresholding method for automatically detecting stars in astronomical images*, Spain University of Extremadura, IEEE 2008

[3] John Canny: *A Computational Approach to Edge Detection*, Usa Artificial Intelligence Laboratory Massachusetts,IEEE 1986

[4] Carlos Platero Dueñas: *Apuntes de Visión Artificial*, España , Dpto. Electrónica, Automática e Informática Industrial, 2009

[5] Brendan M. Quine , Valery Tarasyuk, Henok Mebrahtu, Richard Hornsey: *Determining star image location: A new sub-pixel interpolation technique to process image centroids*, Department of Space Science and Engineering, York University, ElSevier 2007

[6] Yufeng Li, Rongkun Xue, and Fei Liang: *Rapid Star Acquisition algorithm in Star Sensor*, China Shenyang institute of aeronautical engineering, IEEE 2008

[7] Jingchang Pan, Caiming Zhang : *An Efficient Object Detection Method For Large CCD Astronomical Images*, School of Computer Science and Technology, Shandong University and School of Information Engineering, Shandong University at Weihai, IEEE 2009

[8] Rafael C. Gonzalez, Richard E. Woods: *Digital Image Processing*, Library of Congress Cataloging in Pubblication Data., 2th Edition, 2001

[9] Shahin Sohrabi, Ali Asghar Behshti Shirazi: *A New Star Identification Algorithm Based on Fuzzy Algorithms*, Iran University of Science and Technology, Springer 2011

[10] R. Spurzem, P. Berczik, K. Nitadori, G. Marcus, A. Kugel , R. Manner, I. Berentzen, R. Klessen, R. Banerjee : *Astrophysical Particle Simulations with Custom GPU Clusters* , University of Heidelberg, IEEE 2010

[11] Mohsen Azizabadi, Alireza Behrad, M. B. Ghaznavi Ghoushchi: *VLSI implementation of star detection and centroid calculation algorithms for star tracking applications*, Tehran, Iran Shahed University, Springer 2012

[12] Santiago de Pablo Gómez: *Arquitecturas y algoritmos para el tratamiento y la segmentación de imágenes en tiempo real* , España-Valladolid, Universidad de Valladolid, Diciembre de 1995, Pag. 30

[13] J. A. Cortés Osorio , A. Muriel, J.A. Mendoza Vargas : *Comparación cualitativa y cuantitativa de las técnicas básicas de umbralización global basadas en histogramas para el procesamiento digital de imágenes*, Universidad Tecnológica de Pereira, Scientia et Technica, 2011

[14] J. A. Cortés Osorio , J.A. Mendoza Vargas, J.A. Chaves Osorio: *Comparación cualitativa y cuantitativa de las técnicas básicas de umbralización local para el procesamiento digital de imágenes*, Universidad Tecnológica de Pereira, Scientia et Technica, 2012

[15] J. Sauvola, M. PietikaKinen: *Adaptive document image binarization*, Finland University of Oulu, Elsevier 1999

[16] W.Niblack, M. PietikaKinen: *An Introduction to Digital Image Processing*, Englewood Cliff, Prentice Hall 1986

[17] Michael D. Inglis: *A Field Guide to Deep-Sky Objects*, Suffolk County Community College, Springer, Selden, NY, USA, Second Edition

[18] Wilhelm Burger, Mark J. Burge: *Principles of Digital Image Processing, Core Algorithms*, 2nd ed., Springer-Verlag, London.

[19] Analog Devices Inc., *ADSP-BF533 Blackfin® Processor Hardware Reference*, Rev 3.4, April 2009

[20] Jaime Andrés Aranguren Cardona: *Blackfin: DSP para la era multimedia*, Medellin, Colombia

[21] Woon-Seng Gan, Sen M. Kuo: *Embedded Signal Processing with the Micro Signal Architecture*, A John Wiley y Sons, Inc., Publication, 2007

[22] Analog Devices Inc., *Blackfin® Processor Instruction Set Reference*, Rev 2.0, May, 2003

Anexos

Cúmulos estelares

Una mirada casual en el cielo nocturno puede llevar a creer que las estrellas son objetos aislados, solitarios, pero en realidad ninguna estrella ha nacido en el aislamiento. El proceso de nacimiento estelar tiene lugar en inmensas nubes oscuras, las cuales son enormes concentraciones de frío polvo cósmico, que en sus partes más densas y por efecto de la gravedad, se contraen, calientan y empiezan a brillar, dando lugar a las estrellas. Dependiendo del tamaño de las nubes oscuras, las misteriosas y polvorientas "guarderías", estas pueden dar lugar al nacimiento de cualquier cantidad de estrellas, variando desde unas pocas docenas a varios miles de nuevas y jóvenes estrellas. Con el tiempo, sin embargo, esta guardería estelar se dispersará gradualmente. La teoría predice que las estrellas masivas tienen ciclos de vida mucho más cortos que las más pequeñas y las menos masivas, de manera que las estrellas más masivas no viven lo suficiente para escapar de su lugar de nacimiento, mientras que una estrella más pequeña, por ejemplo, de la masa solar, será en la mayoría de los casos quien lograra migrar fácilmente de su lugar de origen[17].

Vale la pena señalar, que en cúmulos que contienen estrellas de masa aproximadamente igual a la del Sol, y que contienen varios miles de estos objetos, la atracción gravitatoria combinada de tantas estrellas puede ralentizar la dispersión del grupo. Realmente depende tanto de la densidad de la estrella como de la masa del cúmulo en particular, por lo que se puede deducir que los cúmulos más densos o apretados, que contienen estrellas de masa del tamaño del sol, serán aquellos de los que contengan la población más antigua de las estrellas, mientras que los grupos más dispersos posiblemente contengan la población estelar más joven.

También cabe destacar que las estrellas presentes en un cúmulo, debido a que nacen de la misma nube de gas o polvo cósmico, comparten características similares como su edad y composición química, algo que las podría diferenciar entre si, es su masa. Se considera que la mayoría de estas agrupaciones se mueven en el espacio como un solo cuerpo debido a su atracción gravitacional, es por este motivo que para determinar la distancia desde la tierra a cada estrella perteneciente a un cúmulo, todos y cada uno de los integrantes de dicho grupo, se analizan con los mismos datos de referencia.

Los cúmulos estelares se pueden dividir en dos grandes grupos, los cúmulos abiertos y los cúmulos globulares, donde ambos tipos de cúmulos son de los objetos astronómicos más estudiados, ya que permiten analizar y comprobar teorías de la evolución estelar, como también proporcionan indicios acerca de la formación del

universo. Para la observación se encuentran disponibles una rica y diversa selección de cúmulos de estrellas, que con la ayuda de instrumentos tales como los binoculares o telescopios se facilita su respectiva comprensión.

Los cúmulos abiertos están compuestos por una pequeña cantidad de estrellas, de entre unas decenas (cúmulo Mariposa o M6 en Escorpión, alrededor de 83 estrellas) y unos miles (cúmulo del Pato Salvaje M11 en Scutum de 2900 estrellas), este tipo de cúmulo contiene estrellas muy jóvenes, tal vez de unos pocos millones de años. Estos grupos no presentan suficiente atracción gravitatoria para mantener a sus estrellas juntas indefinidamente, por lo que en algún momento puede llegar a perder varios de sus integrantes. El sol por ejemplo, que es una estrella solitaria ahora, nació probablemente en un grupo de estos, quienes también se conocen como cúmulos galácticos. Muchos cúmulos de este tipo son fácilmente identificables a simple vista en una noche clara.

Por otro lado, en el caso de los cúmulos globulares que pueden llegar a contener varios miles de estrellas (cúmulo M22 en Sagitario alrededor de 70000 estrellas) o, en algunos casos millones (Omega Centauri o NGC 5139 de alrededor de 10M de estrellas), se cree que pueden llegar a tener edades cercanas a la del universo. Los cúmulos globulares son colecciones masivas de estrellas en forma esférica, que son firmemente unidas por la gravedad y puesto que están muy lejos de la Tierra brindan un espectro demasiado débil para ser observados a simple vista.

.1. Cúmulos abiertos

Los cúmulos abiertos, o cúmulos galácticos, como también se les llama, son conjuntos de estrellas jóvenes, que contienen tal vez una docena de miles de miembros, o algunos de estos, como por ejemplo, Messier 11 en Scutum (pequeña constelación situada en la Vía Láctea entre Aquila y Serpens), contiene un impresionante número de estrellas, que incluso iguala la cantidad de estrellas de algunos cúmulos globulares como por ejemplo M13 con alrededor de 8500 estrellas. Luego están los pequeños grupos, quienes no son más que aparentemente varias estrellas compactas, estas agrupaciones tienen una relación de débil contraste en relación al campo de las estrellas del fondo, como es el caso de IC 4996 en Cygnus. El tamaño de un cúmulo abierto puede variar entre unas pocas docenas de años luz de diámetro, como en el caso de NGC 255 en Casiopea, o hasta unos 70 años luz de diámetro, como en cualquiera de los componentes de Caldwell 14 o el doble cúmulo de Perseo.

Tal es la variedad en formas y tamaños de los cúmulos abiertos, que pueden ser observados por binoculares aunque estos tienen un campo de visión limitado, si se desea tener un impacto pleno se hace necesario el uso de telescopios. La razón de la variedad y disparidad de los cúmulos abiertos es proveniente de las circunstancias de su nacimiento, ya que la nube interestelar es quien determina el número y tipo de estrellas que nacen en su interior, así como también factores tales como el tamaño, la densidad, la turbulencia, la temperatura, y el campo magnético, juegan un papel decisivo en los parámetros característicos referentes al nacimiento de las estrellas. En el caso de las nubes moleculares gigantes (GMCs) las condiciones pueden dar lugar al nacimiento de estrellas tipo O y tipo B (estrellas gigantes), junto con las estrellas enanas de tipo solar, mientras que en las pequeñas nubes moleculares (SMCs) se formarán muchas estrellas de tipo solar, entre las cuales muy pocas, o ninguna en absoluto, tendran la luminosidad de las O ú las estrellas de tipo B. Un ejemplo de un SMCs es la nube oscura Taurus, que se encuentra justo detrás de las Pléyades. Un aspecto interesante de los cúmulos abiertos es su distribución en el cielo nocturno. Se puede pensar que se distribuyen al azar a través del cielo, pero las observaciones muestran más de mil grupos descubiertos, de los cuales unos pocos se encuentran a distancias superiores de 25° por encima o por debajo del ecuador galáctico.

En 1930, Harlow Shapley ideó un sistema muy sencillo de clasificación de cúmulos abiertos, que describe la riqueza del número de estrellas y la concentración del cúmulo. Consiste simplemente en una letra, de la “a” a la “g”.

- a, Irregularidades de campo
- b, Asociaciones estelares
- c, Cúmulos irregulares y muy levemente ligados
- d, Cúmulos levemente ligados
- e, Cúmulos con riqueza y concentración intermedia

- f, Cúmulos bastante concentrados
- g, Cúmulos con una gran riqueza y concentración

En el mismo año, Robert Trumpler ideó un sistema de clasificación de cúmulos abiertos mucho más complejo. Según dicho sistema, cada cúmulo recibe tres carácteres: el primero de ellos, en numeración romana, puede oscilar entre I y IV e indica su concentración y tamaño hasta la estrella más cercana (de mayor a menor), el segundo se escribe en numeración arábiga, pudiendo variar entre 1 y 3, y revela información acerca de la luminosidad de sus miembros (de menos a más), y el último carácter puede ser una p, una m, o una r, e indica si el cúmulo es pobre (menos de 30), medio (entre 50 y 100), o rico (más de 100) en estrellas, respectivamente. Además, si el cúmulo se encuentra dentro de una nebulosa, al final se le añade la letra n. En 1990 se publicó un compendio de todos los cúmulos abiertos de nuestra galaxia conocidos hasta entonces, todos ellos clasificados con el sistema de Trumpler.

Las Pléyades, bajo el sistema de clasificación de Trumpler, queda catalogado como "I3rn"(muy concentrado y luminoso, rico en población de estrellas, e incluido dentro de una nebulosa), mientras que la clasificación de las Híades es "II3m"(más disperso y con pocas estrellas en su haber).

.2. Cúmulos globulares

Diseminados por todos los halos de las galaxias en el cosmos, los cúmulos globulares parecen ser los compañeros constantes de todos ellos. Parece que estos grupos son comunes en todas las galaxias de masas suficientemente grandes como para acogerles. Las edades de estas agrupaciones son probablemente similares a la del universo en sí, hecho quien trae consigo todo tipo de cuestiones acerca de sus orígenes y de sus galaxias anfitrionas. Algunas galaxias parecen tener una proporción desmesurada de cúmulos globulares, como es el caso de las galaxias elípticas quienes con frecuencia contienen hasta mil o más. Algunos colosos galácticos, como la gigante elíptica M87, pueden tener hasta 500 veces o más el número de cúmulos globulares que la Vía Láctea.

La mayoría de las estrellas que comprenden los cúmulos globulares son notables por su bajo contenido de metal (es decir, los elementos más pesados que el helio). Como la población II de estrellas antiguas, que siempre han sido considerados como algunos de los objetos más antiguos del universo (quienes se encuentran en un rango de entre al menos 9 y 12 billones de años, frente a la edad típica de cúmulos abiertos, que tienden a ser tan poco como unos 1,000 veces más jóvenes). Ellos pueden deber su composición química presente ahora en parte a los efectos que agotan su edad avanzada, a pesar de que se formaron en la época en que los elementos que se encuentran en las estrellas de población I eran escasos o incluso inexistentes. Sin embargo, parece que la composición metálica que tienen representa subproductos de los procesos de fusión aún más tempranos presentes en las estrellas de población III.

Recordemos que los cúmulos abiertos son grupos de estrellas que suelen ser jóvenes, tienen un tamaño angular apreciable y pueden tener unos pocos cientos de componentes. Mientras los cúmulos globulares, como se les conoce, son grupos que son muy viejos, son compactos y pueden contener hasta medio millón de estrellas, en algunos casos incluso más. Las estrellas que componen estos cúmulos se les llaman estrellas de población II, estas son estrellas pobres en metales y por lo general se encuentran en una distribución esférica alrededor del centro galáctico en un radio de unos 200 años luz. Por otra parte, el número de cúmulos globulares aumenta significativamente cuanto más nos acercamos al centro de la galaxia, esto significa tener constelaciones que se encuentran en dirección hacia el bulbo galáctico y tienen una alta concentración de cúmulos globulares dentro de ellos, como Sagitario y Escorpión. Por lo tanto, la primavera y el verano son los momentos más favorables para la visualización de estos esquivos objetos.

El origen y la evolución de un cúmulo globular es muy diferente de la de un cúmulo abierto. Todas las estrellas de un cúmulo globular son longevas, con el resultado de que cualquier estrella antes de las de tipo G o F han dejado la secuencia principal y, o bien se encuentran en la fase de gigante roja, y, para las de tipo O y las de tipo B han desarrollado desde hace mucho tiempo la fase de supernova y más allá, dejaran tras de sí una estrella de neutrones o incluso tal vez un agujero negro, de hecho, la nueva formación estelar ya no tiene lugar en ninguno de los cúmulo globulares de la vía láctea, y estos grupos se cree que son las estructuras más antiguas formadas en la galaxia de la vía láctea. Bien puede ser que el más joven de los cúmulos globulares es aún mucho más antiguo que el cúmulo abierto más antiguo. El origen de los cúmulos globulares es un tema de intenso debate y la investigación con los modelos actuales predicen que los cúmulos globulares pueden haberse formado dentro de las nubes proto-galaticas que posiblemente formaron esta galaxia.

Hay alrededor entre 150 y 1586 cúmulos globulares en esta galaxia que varían en tamaños de 60 a 150 años luz de diámetro. Todos ellos se encuentran en vastas distancias del Sol, cerca de 60.000 años luz de la del plano galáctico, y se encuentran en lo que se llama el halo de la galaxia. Los cúmulos globulares más cercanos, por ejemplo, Caldwell 86 en Ara, se encuentra a una distancia de más de 6.000 años luz, y por lo tanto estos grupos son objetos difíciles de observar con pequeños telescopios. Incluso el globular más brillante y más grande tendrá aberturas de por lo menos 15 cm para las estrellas individuales que deben resolverse. Sin embargo, si se utilizan grandes telescopios de abertura, enseguida podrá observar que, estos objetos son verdaderamente magníficos. Algunos cúmulos globulares tienen concentraciones densas hacia su centro, mientras que otros pueden aparecer como cúmulos abiertos compactos. En algunos casos, es difícil decir donde el cúmulo globular se apaga paulatinamente y donde comienzan las estrellas del fondo .

Al igual que en el caso de los cúmulos abiertos, existe un sistema de clasificación, la clase de concentración de Shapley-sawyer, donde la clase I cúmulos globulares densos son los con más estrellas, mientras que la clase XII es el menos denso en estrellas.

Procesamiento digital de imágenes

Los antecedentes indican que, la primera vez que se dio uso de técnicas para la manipulación de imágenes digitales fue en los años veinte, cuando se logro transmitir fotografías a través de cable submarino, entre las ciudades de Nueva York y Londres. Este avance permitió reducir los tiempos entre emisión y recepción de las imágenes, de una semana por barco a tres horas por cable, dicho logro dio pie a desarrollar durante algunas décadas, técnicas que permitieran agilizar la codificación y reproducción de las imágenes.

Sin embargo el procesamiento digital de imágenes nace a partir del momento en que se cuenta con los recursos tecnológicos capaces de captar y manipular enormes cantidades de información en forma matricial, es decir de carácter espacial. Por tal motivo el procesamiento de imágenes digitales se asocia a la base de conocimientos tecnológicos de la ciencia de la computación.

A pesar de que el término "edición de imágenes"a menudo se utiliza indistintamente con el de "procesamiento de imágenes" aquí se desea hacer énfasis en su diferencia, donde edición de imágenes se entenderá como la manipulación de imágenes digitales mediante una aplicación de software existente, como Adobe Photoshop o Corel Paint, y por otro lado el termino procesamiento de imágenes digitales, se entenderá como la concepción, el diseño, el desarrollo y la mejora de los algoritmos de procesamiento digital de imágenes[18].

.3. Representación de una imagen

.3.1. Imagen

Una imagen se define como una función bidimensional de la luz y la intensidad. que se representa como $f(x, y)$, para cada punto de la imagen (x, y), la función da como resultado el valor de la intensidad de la imagen en dicho punto.

Una imagen representa una señal correspondiente al espectro siendo una forma de energía, que por lo tanto ha de ser finita y estrictamente mayor que cero, teniéndose por tanto que $f(x, y)$ toma valores enteros que se encuentran en el rango $0 \succ f(x, y) \prec n$, siendo n un valor finito. La diferencia entre una imagen analógica y una digital, es más que clara dado que la primera de ellas es de tipo continuo y la

segunda es de tipo discreto. Esto quiere decir que una señal digital esta representa por un número concreto de valores, mientras que una señal análoga se representa a través de una función de puntos infinitos. Por tal motivo, la digitalización de una imagen es solo una aproximación a la señal original.

Las imágenes digitales están formadas por una matriz de elementos llamados "píxeles", cada uno de estos contiene un valor correspondiente al color o intensidad el cual viene asociado a cada elemento de la matriz, según sus coordenadas espaciales. El origen de coordenadas de una imagen esta dado por la posición $(0, 0)$ de $f(x, y)$ es decir en la esquina superior izquierda de la imagen[8].

Para entender más claro como se forman las imágenes, se deben tener claros conceptos básicos referentes a la luz y su espectro, al igual que el concepto de píxel y el espacio del color RGB de los cuales se dará una breve introducción a continuación.

.3.2. Píxeles

Las imágenes digitales están compuestas por entre decenas a millones de puntos llamados píxeles, nombre que viene de la contracción de "picture element" (elemento de imagen), y representa la mínima porción de la imagen que puede ser manipulada directamente. Cada uno de estos posee características propias, que son otorgadas por el color y la intensidad.

A la cantidad de píxeles que contiene una imagen se le otorga el nombre de resolución, y generalmente se denomina con un par de números que hacen referencia a los ejes horizontal y vertical, es decir, una resolución de 720*256, indica que la imagen sera representada por una matriz de 720 píxeles o puntos horizontales y 256 verticales, lo cual da un total de 184,320 píxeles que representaran la imagen a procesar, por lo que, a mayor resolución, mayor cantidad de píxeles, mejor calidad de imagen y por consiguiente, mayor sera la cantidad de valores a procesar, a mostrar en pantalla y almacenar en memoria.

La relación de aspecto o también llamada "aspect ratio" de la imagen, es la relación que existe entre la cantidad de píxeles en el eje horizontal y con los pertenecientes al eje vertical. La relación de aspecto más común en las pantallas de los computadores es 4:3, son calibrados a esta relación para que las imágenes geométricas aparezcan de manera proporcional.

.3.3. Resolución de bits

Concepto también conocido como profundidad de píxel, este es quien proporciona una medida del número de bits de información que puede almacenar un píxel. Es decir, ofrece al usuario información sobre el color que proporciona cada píxel de la imagen al igual que su capacidad de soporte de colores. esto indica que, a mayor profundidad de píxel, se pueden obtener más colores y una amplia representación de los mismos y por ende de la imagen. Un píxel con profundidad 1 tiene dos valores

posibles: 255 ó 0, blanco/negro, esto análogo a, 1 ó 0, un píxel con profundidad 8 tiene 256 valores posibles, como ocurre con las imágenes en escala de grises o color indexado, es decir se tiene la posibilidad de contar con 256 colores.de igual forma para un píxel con profundidad 24 tiene 16.000.000 valores posibles, este tipo es el que permite obtener imágenes representadas en millones de colores.

.3.4. El espacio del color RGB

Es el modelo más conocido y utilizado. En el modelo RGB cada color está representado por tres valores de rojo (R), verde (G) y azul (B), situados a lo largo de los ejes del sistema de coordenadas cartesianas sobre un cubo. Los valores de RGB van en un rango de [0,1] o de [0, 255]. De esta manera el negro se indica como (0, 0, 0), el blanco como (1, 1, 1) o (255, 255, 255). El color negro y el blanco están representados por dos esquinas opuestas del cubo que se pueden definir por los ejes de R, G, B, del sistema de coordenadas cartesianas. Las otras puntas del cubo representan el rojo, el verde, el azul, el cyan, el magenta y el amarillo. Los colores en escala de grises se representan con componentes idénticas de R, G, B.

Estos espacios de color son parte integral de las normas que rodean el grabado, el almacenamiento, la transmisión y visualización de señales de televisión. YIQ y YUV son los métodos fundamentales de codificación de color para algunas formas de la televisión análoga $NTSC$ y PAL, y $YCbCr$ forma parte de las normas internacionales que rigen la televisión digital. Todos estos espacios de color tienen en común la idea de separar la componente de luminancia "Y" y a partir de dos componentes de croma determinan los colores de la imagen. De esta manera se logra la compatibilidad con los sistemas anteriores en blanco y negro. En el espacio del color YIQ, "Y" representa la intensidad (luma) que corresponde al brillo de la imagen en sus valores óptimos adecuados para la percepción del ojo humano,"I" representa la fase (phase), mientras que "Q" corresponde a la cuadratura (quadrature), en referencia a las componentes utilizadas en la modulación de amplitud en cuadratura.

Tarjeta de desarrollo ADSP BF-533

La tarjeta de desarrollo ADSP BF-533 es un miembro de la familia Blackfin®, producto incorporado por Analog Devices®, Inc. Intel Micro. Basados en arquitectura de micro señal (MSA), y desarrollados en conjunto con Intel®, los procesadores Blackfin combinan una serie de instrucciones de 32 bits RISC trabajando de forma similar a los micro-controladores de propósito general. Además, cuentan con una funcionalidad de procesamiento de señal de multiplicación y acumulación (MAC) de 16 bits dual.

Todos los miembro de la familia Blackfin® comparten el mismo núcleo, lo único que los diferencia es la capacidad de memoria, la velocidad de operación, el consumo de potencia y el número de periféricos integrados; lo que hace que estos procesadores sean muy utilizados en dispositivos electrónicos y en equipos de telecomunicaciones y procesamiento de video de alto desempeño[19].

El procesador ADSP BF-533 es soportado por un completo conjunto de herramientas de desarrollo tanto de software como de hardware, incluyendo emuladores de Analog Devices y el entorno de desarrollo VisualDSP++® que deja al programador desarrollar y depurar una aplicación. Este entorno incluye un fácil uso del lenguaje ensamblador, quien se basa en una sintaxis algebraica, una de las principales ventajas de este entorno es el uso eficiente de código C/C++, el compilador ha sido desarrollado para una traslación eficiente entre código ensamblador y C/C++

La Familia Blackfin emplea un conjunto de instrucciones en lenguaje ensamblador, las instrucciones han sido especialmente diseñadas para mejorar la flexibilidad, también proveen completamente características multi-funcionales permitiendo al programador el uso de todos los recursos del núcleo en una simple instrucción.

.4. Hardware ADSP BF-533

El procesador ADSP BF-533 es un miembro mejorado de la familia Blackfin que ofrecen significativamente más alto rendimiento con menos consumo de potencia que previos procesadores Blackfin, mientras al mismo tiempo conserva los beneficios de compatibilidad y el fácil uso de código, además es completamente compatible en pines con el ADSP BF-531 y el ADSP BF-532 diferenciándose solo de estos por su más alto desempeño y su más alta capacidad de memoria en el interior del nucleo.

La arquitectura del nucleo del procesador combina un motor de procesamiento de doble MAC, con un conjunto de instrucciones tipo RISC (conjunto de instrucciones reducido), con características SIMD (multiples datos con una única instrucción).

Los procesadores Blackfin poseen una caracteristica que le permite manejar dinamicamente su potencia, por su habilidad de variar tanto su frecuencia como su voltaje, optimizando su consumo dependiendo de su tarea actual. En la estructura del ADSP BF-533 se observan claramente dos grandes bloques, Los periféricos del sistema y el núcleo del procesador. Figura 1

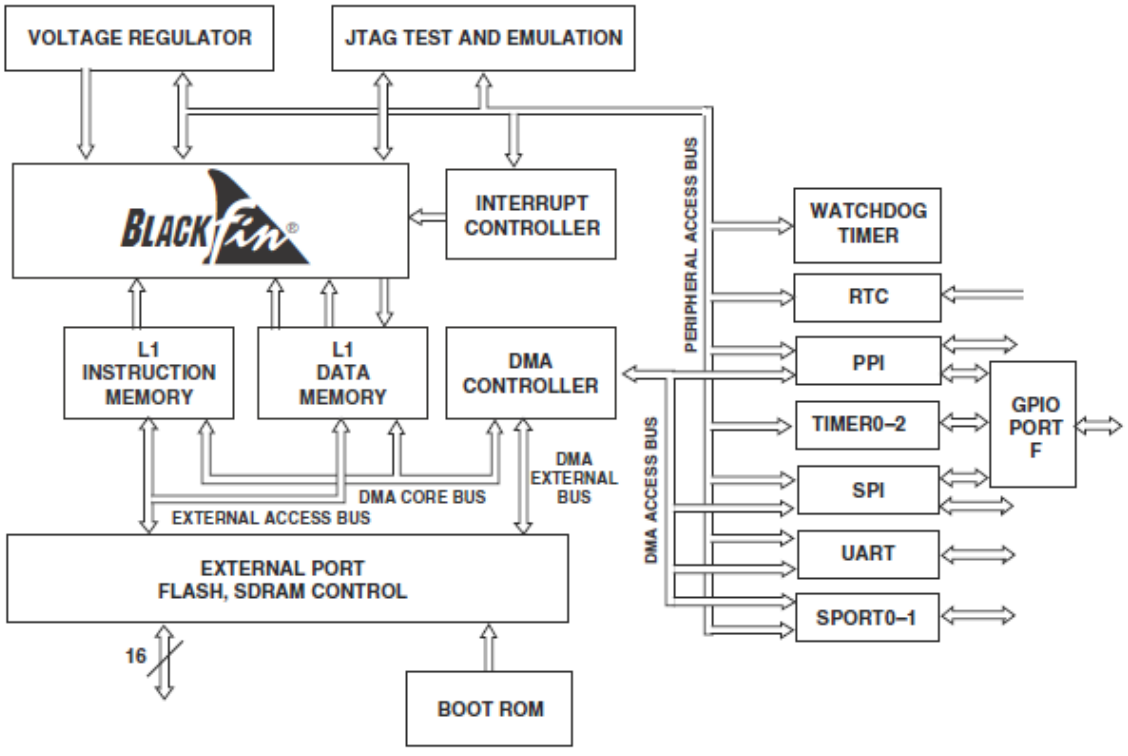


Figura 1: Diagrama de bloques ADSP BF-533.

.4.1. Periféricos del sistema

El ADSP BF-533 contiene una gran variedad de periféricos conectados al núcleo a través de buses de alto ancho de banda, entre los periféricos de propósito general encontramos: Un puerto UART, un puerto SPI, dos puertos serie (SPORTs), cuatro temporizadores de uso general (tres con capacidad PWM), un reloj de tiempo real, un temporizador "perro guardián", y una interfaz paralela. Las especificaciones de rendimiento y configuraciones de memoria de la tarjeta en cuestión se muestran en la Tabla 1.

Características	**ADSP BF-533**
SPORTs	2
UART	1
SPI	1
GP Timers	3
WatchDog Timers	1
RTC	1
Parallel Peripheral Interface	1
GPIOs	16
L1 Istruction SRAMCache	16K bytes
L1 Istruction SRAM	64K bytes
L1 Data SRAMCache	32K bytes
L1 Data SRAM	32K bytes
L1 Scratchpad SRAMCache	4K bytes
L3 Boot ROM	1K bytes
Maximum Speed Grade	600MHz
Package Options:	
CSP_BGA	160-Ball
Plastic BGA	169-Ball
LQFP	176-Lead

Tabla 1: Especificaciones de memoria y características generales del procesador Blackfin® ADSP BF-533.

El ADSP BF-533 contiene puertos seriales y paralelos de alta velocidad para aplicaciones que utilicen audio y video, un controlador de interrupciones para un manejo flexible de interrupciones ocurridas en el interior del núcleo, como también de las producidas por los recursos externos. Todos los periféricos, a excepción de los de propósito general de E/S en tiempo real y temporizadores, son apoyados por una estructura flexible DMA. También hay un canal DMA de memoria independiente dedicada a la transferencias de datos entre el procesador en distintos espacios de memoria, incluyendo memoria SDRAM de memoria externa y asincrónica.

SPORT's

El procesador ADSP BF-533 incorpora dos puertos seriales síncronos (SPORT0, SPORT1), para múltiples protocolos de comunicación, además de permitir la conexión con otros procesadores en una estructura multiprocesador.

Los puertos seriales SPORT's proveen unas interfaces de E/S para la comunicación con una gran variedad de dispositivos, cada SPORT tiene un grupo de pines para transmitir (Dato primario, Dato secundario, Reloj, y Sincronización de trama) y un segundo grupo para recibir. Las funciones de recepción y transmisión son programadas de forma independiente, lo que lo hace un sistema full-duplex ya que es capaz de transferir datos en ambas direcciones de forma simultanea. Además cada puerto puede ser programada su tasa de bit's, trama de sincronización y número de bit's por palabra.

UART

El Procesador ADSP BF-533 provee un puerto universal asíncrono de transmisión y recepción (UART) full-duplex, totalmente compatible con protocolos PC. El UART convierte datos entre serial y paralelo, permite también diferentes longitudes de palabras, bits de parada, y opciones para generar paridad, maneja interrupciones por hardware, las interrupciones pueden ser generadas con 12 eventos diferentes.

SPI

El procesador tiene una interface serial que provee una amplia comunicación con una gran variedad de dispositivos. El puerto SPI tiene 4 líneas que consisten en 2 pines de datos, 1 pin de selección del dispositivo, y un pin del reloj. El puerto suporta modos esclavo, maestro y multi-maestros, generalmente los puertos SPI se usan para comunicarse con dispositivos como Codec's, Convertidores A/D, Convertidores D/A, Display LCD, FPGA's, otros CPU's o micro-controladores.

Interfaz de periféricos paralelos (PPI)

Los procesadores ofrecen una interfaz de periféricos paralelos (PPI) que se puede conectar directamente en paralelo con ADCs y DACs, codificadores de vídeo y descodificadores, y otros periféricos de uso general. La PPI contiene en un pin de entrada de reloj, hasta tres pines de sincronización, y hasta 16 pines de datos. El reloj de la entrada soporta velocidades de datos en paralelo de hasta la mitad de la velocidad del reloj del sistema, y las señales de sincronización se pueden configurar como entradas o salidas. El PPI se utiliza para conectar LCDs, codificadores de video, decodificadores de video, sensores CMOS o CCDs, y dispositivos genéricos paralelos de alta velocidad. El puerto PPI puede alcanzar frecuencias de hasta 75 MHz y puede configurarse en un rango de 8 a 16 bits.

Reloj en tiempo real (RTC)

El reloj en tiempo real (RTC), ofrece un conjunto de características como un reloj digital, incluyendo el tiempo actual, cronómetro, y alarma. El periférico RTC es registrado por un cristal externo al procesador de 32.768 KHz. El RTC dedica pines de alimentación para que pueda permanecer encendido, incluso cuando el resto del procesador se encuentra en estado de baja alimentación. El RTC ofrece interrupción programable, incluyendo interrupción por segundo, minuto, hora, días o impulsos de reloj.

Los 32.768 KHz de frecuencia de entrada se dividen en frecuencias de hasta un 1 Hz usando el prescaler. La función del tiempo de conteo consiste en cuatro contadores organizados de la siguiente manera: Un contador de 60 segundos, un contador de 60 minutos, un contador de 24 horas y un contador de 32.768 días.

Temporizador de vigilancia (Watchdog Timer)

El procesador ADSP BF-533 incluye un temporizador de 32 bits que puede ser utilizado para implementar una función de vigilancia del software. Un organismo de control de software puede mejorar la disponibilidad del sistema, forzandolo a un estado conocido a través de la generación de un restablecimiento por hardware, interrupción no enmascarable (NMI), o interrupciones de propósito general. Si el temporizador expira antes de que se restablezca por software, el programador inicializa el valor del contador en el reloj, habilita la interrupción apropiada, para luego poder habilitar el reloj. A partir de entonces, el software debe encargarse de cargar el contador antes de que llegue a cero, esto para proteger el sistema de que llegue a un estado desconocido. Si está configurado para generar un restablecimiento de hardware, el temporizador de vigilancia restablece tanto el núcleo, como los periféricos del procesador ADSP BF-533.

Después de un reset, el software puede determinar si el "Perro Guardián" fue la fuente del reinicio, esto interrogando el bit de estado en el registro de control de temporizador de vigilancia.

Temporizadores de propósito general

Hay cuatro unidades de temporización programable de uso general en el procesador ADSP BF-533. Tres temporizadores tienen un pin externo que puede ser configurado ya sea como un modulador de ancho de pulso (PWM) o de salida del temporizador, como una entrada al reloj del temporizador, o como un mecanismo para medir anchos de pulso y los períodos de eventos externos. Estos temporizadores pueden ser sincronizados a una entrada de reloj externo a la clavija de PF1, una entrada de reloj externo a la clavija PPI_CLK, o a la SCLK interna.

Las unidades de temporizador se puede utilizar en conjunto con el UART para medir la anchura de los pulsos en la corriente de datos para proporcionar una función de detección automática de baudios para un canal serie. Los temporizadores

pueden generar interrupciones en el núcleo del procesador que proporciona eventos periódicos para la sincronización, ya sea al reloj del sistema o a un recuento de las señales externas.

Además de los tres temporizadores programables de propósito general, también se proporciona un cuarto temporizador. Este temporizador adicional es sincronizado por el reloj interno del procesador y se utiliza normalmente como un reloj de tic para la generación de interrupciones periódicas del sistema operativo.

.4.2. Núcleo del procesador ADSP BF-533

El núcleo del procesador Blackfin esta conformado por tres unidades o bloques funcionales, la unidad aritmética de datos que a su ves esta conformada por dos multiplicadores de 16-bits, dos acumuladores de 40-bits, dos ALU's 40-bit, cuatro ALU's de vídeo, y una unidad de corrimiento de 40 bits. Además de la unidad Aritmética de direccionamiento que esta conformada por dos DAG's, ocho registros apuntadores de 32-bits de propósito general (P0-P5,SP,FP), consta de cuatro conjuntos de registros de 32-bits, 4 índices (I0-I3), 4 modificadores (M0-M3), 4 de longitud (L0-L3), y 4 de base (B0-B3). Y por último la unidad de control conformada por el programa secuenciador que se encarga de controlar el flujo de ejecución de las instrucciones, incluyendo instrucciones de alineación y decodificación. Para el control de flujo del programa, el secuenciador soporta PC, relativos y saltos condicionales indirectos, interrupciones y excepciones, inactividad o idle y llamadas a subrutinas.

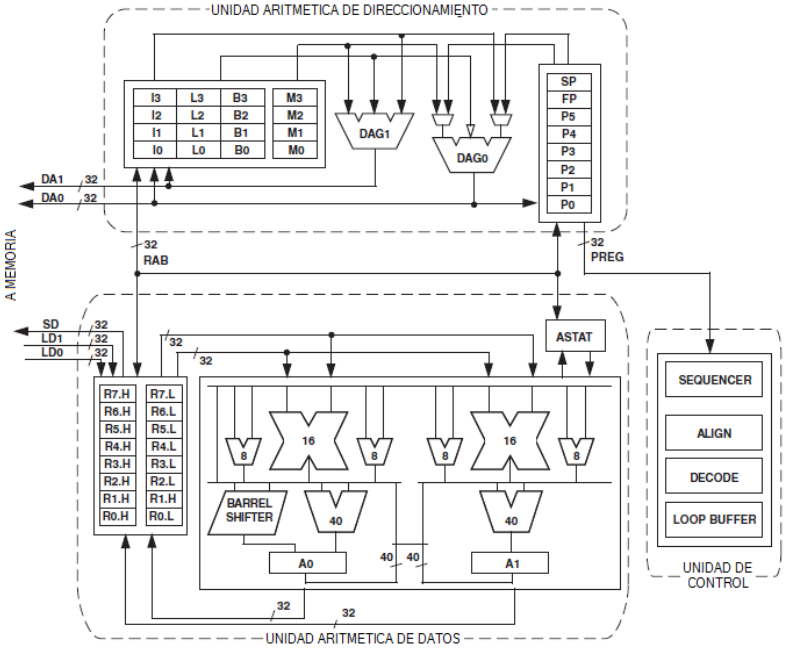


Figura 2: Estructura del núcleo del procesador Blackfin ADSP BF-533.

Unidad aritmética de datos

Las ALU's realizan un conjunto tradicional de operaciones tanto aritméticas, como lógicas sobre los datos de 16 bits o de 32 bits. Además, cuenta con muchas instrucciones especiales para acelerar las tareas de procesamiento de señales. Entre estas encontramos, operaciones de bits, tales como extracción de campo y conteo de población, un módulo de multiplicación2^{32}, división primitiva, saturación, redondeo, y detección de signo/exponente.

El conjunto de instrucciones de vídeo incluye las operaciones de alineación de bytes y de empaquetamiento, promedio de 8 bits, valor absoluto, acumulador (SAA) y resta. Para ciertas instrucciones, dos operaciones de ALU de 16 bits se pueden realizar simultáneamente a través de registros pares(16-bits de la parte baja de un registro y 16-bits de la parte alta del otro registro). Y si además se utiliza la segunda ALU serian cuatro operaciones que se podrían realizar simultáneamente.

El modulo de corrimiento o desplazamiento de 40 bits puede ser utilizado para realizar corrimiento, rotación y para apoyar la normalización.

Cada MAC puede realizar una multiplicación de 16 bits por 16 bits y acumular el resultado a 40-bits en cada ciclo. Además soporta operaciones con signo y sin signo como también saturación y redondeo.

La unidad procesa datos de 8 bits, 16 bits o 32 bits provenientes de los ocho registros de 32-bits de propósito general. Cuándo se realizan operaciones de cálculo con datos de 16 bits, los registros funciona como 16 registros independientes de 16 bits.

Unidad aritmética de direccionamiento

Los registros punteros de 32 bits (P0-P5) permiten el direccionamiento aritmético, proporcionando dos direcciones simultáneas para obtener datos de la memoria. Estos pueden a su vez ser usados como registros de datos para operaciones aritméticas muy simples con un reducido conjunto de operaciones, tampoco afectan las banderas en el registro de status (ASTAT)

El conjunto de registros DAG's de 32 bits que consiste en 4 grupos de 4 registros (I0-I3,M0-M3,L0-L3,B0-B3) son principalmente usados para el direccionamiento, los registros I contienen la dirección efectiva, los registros M contienen un offset para ser sumado o restado de un registro I, los registros B y L definen ciclos, el B contiene la dirección de inicio del bucle y el registro L define la longitud del mismo. Cada par de registros B y L debe ser asociado con su correspondiente registro I, aunque los registros M pueden trabajar con cualquiera. En ocasiones estos registros pueden ser utilizados para salvaguardar un dato como si fuera un registro de datos, sin permitir realizar operaciones sobre estos.

Unidad de control

En el procesador el programa secuenciador controla el flujo del programa, proveyendo constantemente la dirección de la siguiente instrucción hacer ejecutada por otras partes del procesador, el flujo del programa es lineal aunque esta puede ser alterada por diferentes tipos de estructuras como:

- **Bucles:** Una secuencia de instrucciones se ejecuta muchas veces.
- **Subrutinas:** El procesador temporalmente interrumpe el flujo normal de la instrucciones para atender otras en una parte diferente de la memoria.
- **Saltos:** El flujo del programa se transfiere definitivamente para otra parte de la memoria.
- **Interrupciones y excepciones:** Un evento o interrupción obligan a la atención de una subrutina.
- **Idle:** Una instrucción puede obligar al procesador a parar toda actividad, y entonces tener que esperar la normalización, a menos que una interrupción ocurra, donde el procesador atendería el servicio de dicha interrupción y continuaría con el normal funcionamiento.

.4.3. Arquitectura de memoria

La memoria de los procesadores Blackfin esta configurada como un espacio unificado de 4G bytes, con direccionamiento a través de registros de 32-bits. Todos los recursos, incluyendo la memoria interna, memoria externa, y los registros de control de E/S, ocupan secciones separadas de este espacio común. Las porciones de la memoria se organizan en una estructura jerárquica para proporcionar una buena relación coste/rendimiento, de baja latencia en la memoria interna (como la caché o SRAM), con menor costo y rendimiento en la memoria externa. Tabla 2

El sistema de memoria SRAM L1, corre a la misma frecuencia de reloj, para un mayor rendimiento de memoria disponible en el procesador. Las memorias de instrucciones y de datos son independientes y están conectadas al núcleo por medio de buses dedicados, que permiten el rápido intercambio de datos entre el núcleo y la memoria L1. La memoria externa, se accede a través de la unidad de interfaz de bus externo(EBIU), esta ofrece la expansión de SDRAM, memoria flash, y SRAM, opcionalmente acceder hasta 132M bytes de memoria física. El controlador DMA de memoria proporciona una capacidad alta de ancho de banda para transferencias de bloques de códigos o datos entre la memoria interna y externa. [20]

Memoria interna (ON-CHIP)

El ADSP BF-533 tiene tres bloques de memoria en el interior del chip que facilitan el acceso al nucleo debido a su gran ancho de banda. La primera es la memoria de instrucciones L1, que consta de 80K bytes de SRAM, la cual puede ser configurada

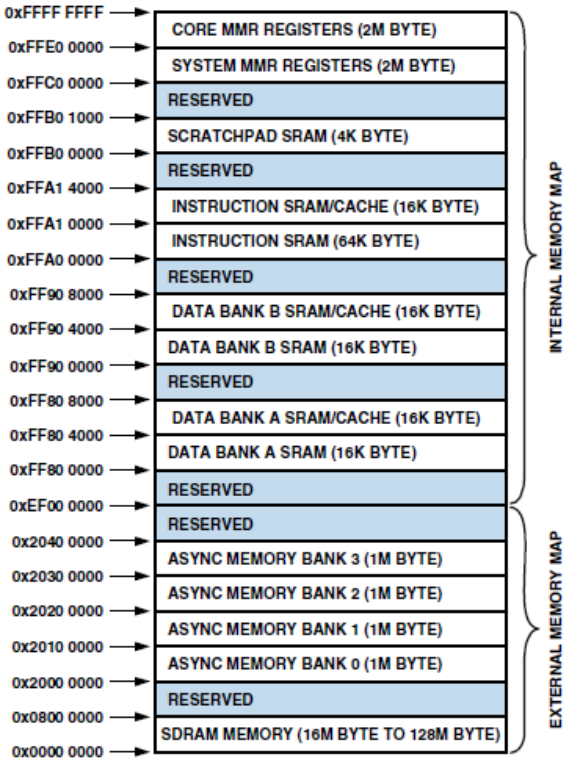


Tabla 2: Mapa de memoria interna/externa Blackfin ADSP BF-533.

como una mezcla de memoria SRAM y de caché. El segundo bloque de memoria en el chip es la memoria de datos L1, que consiste en dos bancos de 32K bytes cada uno. Cada banco de memoria es configurable, opción ofrecida tanto por la memoria caché, como por la SRAM. El otro bloque de memoria es un bloc de notas de 4K bytes de SRAM que funciona a la misma velocidad que la memoria L1, pero sólo es accesible como SRAM de datos y no se puede configurar como memoria caché.

Memoria externa (Off-CHIP)

La interfaz de bus externo se puede utilizar tanto con dispositivos asíncronos tales como SRAM, FLASH, EEPROM, ROM, dispositivos E/S, y dispositivos sincrónicos como SDRAM. La anchura del bus es siempre 16-bits. Periféricos de 8 bits deberían abordarse como si fueran dispositivos de 16 bits, donde sólo los ocho bits inferiores de los datos serian utilizados.

El controlador de memoria asíncrona puede ser programado para controlar hasta cuatro bancos de dispositivos con los parámetros de tiempo muy flexible para una amplia variedad de dispositivos. Cada banco ocupa un segmento de 1Mbyte, independientemente del tamaño del dispositivo utilizado, por lo que estos bancos sólo serán contiguos, si cada uno de ellos está completamente poblado con 1M byte de memoria.

Espacio de memoria E/S

Los procesadores Blackfin no definen un espacio de E/S independiente. Todos los recursos son mapeados a través de banderas de 32-bits. Los dispositivos de E / S interna tienen sus registros de control(MMR) asignados en la parte superior de la memoria de 4GBytes. Estos están separados en dos bloques más pequeños, uno de los cuales contiene los MMR de control para todas las funciones básicas del núcleo, y el otro de los cuales contiene los registros necesarios para configurar y controlar los periféricos fuera del núcleo. Los MMR son accesibles sólo en el modo supervisor y aparecen como espacio reservado en los periféricos internos.

Arranque"Booting"

El procesador ADSP BF-533 contiene un pequeño Kernel, que es el encargado del adecuado arranque de la tarjeta. Si el procesador ADSP BF-533 está configurado para iniciar desde un espacio de la memoria ROM, el procesador comienza a ejecutar a partir del espacio de memoria ROM de arranque.

El procesador ADSP BF-533 tiene dos mecanismos Tabla 3 para cargar automáticamente la memoria interna de instrucciones L1 después de un reinicio. Un tercer modo se proporciona para ejecutar desde una memoria externa, sin pasar por la secuencia de arranque.

BMODE 1-0	Descripción
00	Ejecuta desde una memoria externa de 16-bits
01	Arranca desde memoria flash de 8-bits o 16-bits
10	Arranca desde el terminal SPI en modo esclavo
11	Arranca desde el SPI serial EEPROM

Tabla 3: Modos de arranque.

Manejo de eventos

El controlador de eventos en el procesador ADSP BF-533 maneja todos los eventos asíncronos y síncronos. El procesador ADSP BF-533 soporta tanto eventos de anidación y priorización. Anidamiento permite múltiples servicios de atención a las rutinas al mismo tiempo. Priorización asegura que el servicio de un evento de mayor prioridad tiene preferencia sobre el servicio de un evento de menor prioridad. El controlador proporciona soporte para cinco diferentes tipos de eventos:

- **Emulación:** Un evento emulación hace que el procesador para entrar en modo de emulación, lo que permite el mando y control del procesador a través de la interfaz JTAG.
- **Restablecer:** Este evento pone a cero el procesador.
- **NMI(Interrupción no enmascarable):** El evento NMI puede ser generado por el temporizador de vigilancia (Watchdog Timer) de software o por la señal

de entrada NMI al procesador. El evento NMI se utiliza con frecuencia como un indicador de apagado para iniciar un apagado ordenado del sistema.

- **Excepciones:** Eventos que se producen sincrónicamente al flujo del programa (es decir, se tomarán la excepción antes de permitir que las instrucciones se completen). Las condiciones tales como violaciones de alineación de datos e instrucciones indefinidas causan excepciones.

- **Interrupciones:** Eventos que se producen de forma asíncrona para el flujo del programa. Son causadas por los pines de entrada, temporizadores y otros periféricos, así como por una instrucción de software explícito.

Cada tipo de evento tiene un registro asociado para guardar la dirección de retorno, y una instrucción asociada de retorno. Cuando se activa un evento, el estado del procesador se guarda en la pila del supervisor.

El controlador de eventos del procesador ADSP BF-533 consta de dos etapas, el controlador de eventos del núcleo (CEC) y el controlador de interrupciones del sistema (SIC). El controlador de eventos núcleo trabaja con el controlador de interrupciones del sistema para priorizar y controlar todos los eventos del sistema. Conceptualmente, las interrupciones de los periféricos entran en el SIC, y estas luego se encaminan directamente a las interrupciones de propósito general del CEC.

Controlador de eventos del núcleo (CEC)

La CEC soporta nueve interrupciones de propósito general (IVG15-7), además de las excepciones e interrupciones dedicadas. De estas interrupciones de propósito general, se recomiendan dejar libres las dos interrupciones de menor prioridad (IVG15-14) para el manejar las interrupciones por software, dejando a siete interrupciónes priorizadas para manejar los periféricos del procesador ADSP BF-533.

Sistema controlador de interrupciones (SIC)

El sistema controlador de interrupciones proporciona el mapeo y enrutamiento de los acontecimientos de las muchas fuentes de interrupciones periféricas, priorizando las de propósito general del CEC. Aunque el procesador ADSP BF-533 ofrece un mapeo por defecto, el usuario puede modificar las asignaciones y prioridades de los eventos de interrupción, escribiendo los valores adecuados en los registros de asignación de interrupciones (IAR).

.4.4. Controladores DMA

El procesador ADSP BF-533 tiene múltiples canales DMA independientes que soportan transferencias de datos de una forma automatizada disminuyendo la sobrecarga en el núcleo del procesador. Este controlador permite trasferir bloques de código o datos entre memorias internas del procesador y cualquiera de sus periféricos con capacidad para DMA, con una mínima disipación de calor en el núcleo. Además, las transferencias se puede lograr entre cualquiera de los periféricos con capacidad

DMA y dispositivos externos conectados a la interfaz de memoria externa, incluyendo el controlador de memoria SDRAM y controlador de memoria asincrónico. Especialmente, los procesadores tienen canales DMA dedicados para cada periférico permitiendo alta velocidad en el intercambio de información para aplicaciones que necesiten tomar ventaja de codificación y decodificación de video en tiempo real. Los periféricos con capacidades DMA incluyen los SPORT's, el puerto SPI, la unidad UART, y el PPI.

.5. Software para ADSP BF-533

El entorno de desarrollo VisualDSP++ permite a los programadores elaborar y depurar un programa, este entorno incluye tanto un lenguaje ensamblador de fácil uso (basado en una sintaxis algebraica), como también un enlazador, un archivador, un compilador de lenguaje C/C++, un simulador y una librería que incluye funciones matemáticas y de DSP en lenguaje C/C++; El compilador ha sido desarrollado para una eficiente interpretación del código C/C++ en el procesador ensamblador.

El depurador de VisualDSP++ tiene ciertas características que lo hacen muy atractivo, como la representación gráfica de los datos, que permiten al programador determinar rápidamente el rendimiento de un algoritmo. A medida que se desarrolla un algoritmo, este va creciendo en complejidad, esta cualidad puede tener cada vez mayor importancia en el calendario de desarrollo del diseñador. Esta propiedad de VisualDSP ++, permite al desarrollador de software de forma pasiva reunir importantes métricas de ejecución de código, sin afectar el funcionamiento en tiempo real del programa. Esencialmente, el desarrollador puede identificar cuellos de botella en el software de forma rápida y eficiente. Al utilizar el perfilador, el programador puede centrarse en esas áreas del programa que tienen un alto impacto negativo en el rendimiento y de esta manera puede tomar acciones correctivas.

Los depuradores de VisualDSP++, tanto el de código C/C++ y ensamblador le permiten al programador, realizar su algoritmo con una mezcla de estos dos lenguajes, insertar “break points”, configurar “break points” condicionales en registros y memorias, realizar un trazado lineal y estadístico de la ejecución del programa, mostrar mediante gráficos la información contenida en la memoria.

.5.1. Descripción del entorno

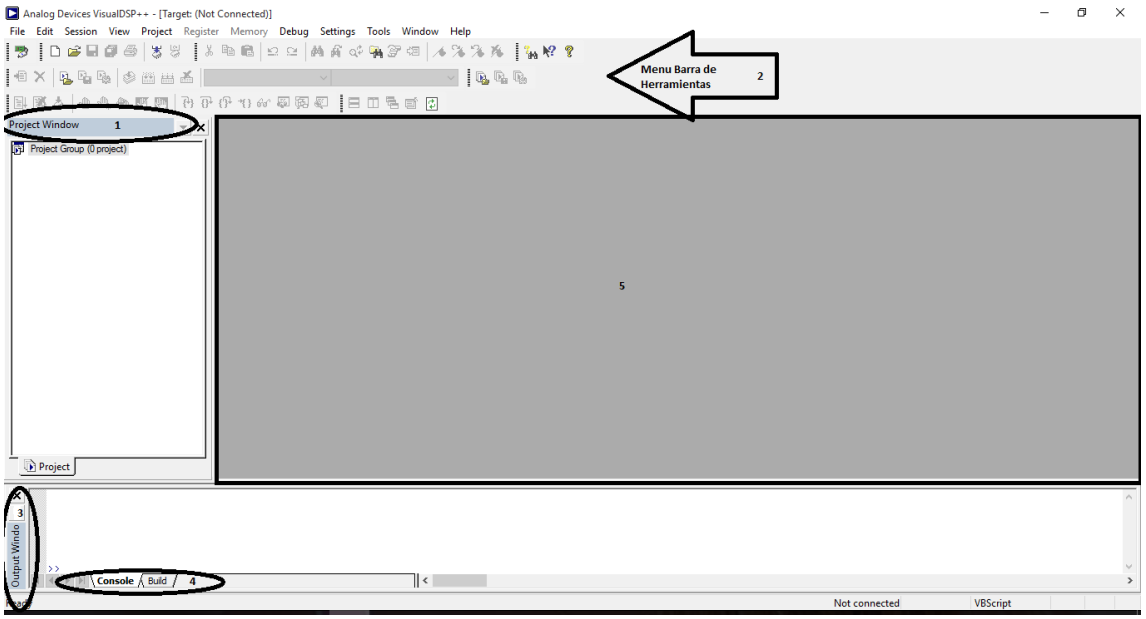


Figura 3: Entorno VisualDSP++.

Cuando se ingresa al entorno de VisualDSP++ por primera vez Figura 3, se pueden apreciar 3 ventanas principales, como también una barra de menú(2) y una barra de herramientas(2). Entre las ventanas se encuentran la ventana de proyectos(1), la ventana de salida(3) y la ventana de edición(5), adicionalmente el usuario puede personalizar el entorno de acuerdo a sus necesidades, para ello puede tomar ventaja de ventanas adicionales como la ventana "stack/frame pointer" que permite ver el contenido de estos registros en tiempo real. A estas ventanas adicionales se les conoce como ventanas de depuración[21].

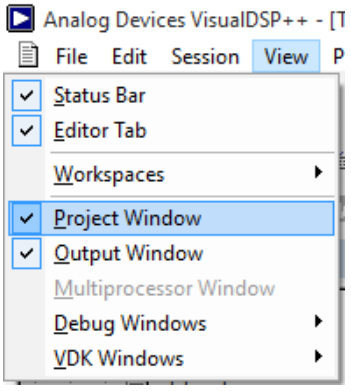


Figura 4: Ruta ventana de proyectos.

Para abrir la ventana de proyectos debe dirigirse a la ruta "View/Project Window", Figura 4 con esto conseguirá tener control total de sus proyectos, alli observara

carpetas que contienen los archivos de dicho proyecto, entre los archivos más comunes se encuentran archivos de cabecera(.h, .hpp, .hxx), archivos enlazadores(.ldf, .dlb, .doj), archivos fuente(.asm,.c, .cpp, .cxx, .dsp, .s) y archivos kernel(.vdk), pero para incluir estos últimos deberá activar el VDK. Si desea adicionar más proyectos en la ventana debe dirigirse a la siguiente ruta "File/Open/Project".

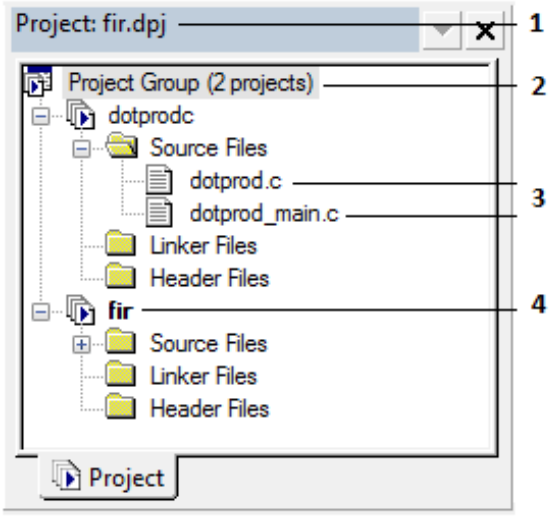


Figura 5: Ventana de proyectos.

En la Figura 5 se describe la estructura básica de la ventana en 4 puntos, el punto 1 muestra el nombre del proyecto activo, el número 2 indica el número de proyectos contenidos en la ventana, el 3 muestra algunos archivos que están contenidos en determinado proyecto, y por último el 4 indica con negrita que este proyecto se encuentra activo, teniendo en cuenta lo anterior se deducen algunas reglas básicas de dicha ventana como lo son: tener varios proyectos abiertos al mismo tiempo pero solo uno puede estar activo, pueden incluirse cualquier cantidad de diferentes tipos de archivos pero los archivos de extensión desconocida serán omitidos por el compilador, dos archivos no pueden tener el mismo nombre si se encuentran en la misma carpeta, y para finalizar solo se permite un archivo enlazador(.ldf).

La ventana de edición permite abrir, crear o editar los archivos que se incluirán en el proyecto, para abrir un archivo dirigirse a "File/Open/File", y se selecciona el archivo deseado, otra manera de abrir un archivo seria desde la ventana de proyectos dando doble clic sobre el archivo deseado o con el acceso rápido(ctrl + O).

El editor puede ser personalizado ya que maneja algo conocido como sintaxis de color esto quiere decir que todo el entorno editor puede tomar diferentes colores de acuerdo a si son comentarios, si es código, si son letras, números o símbolos, etc. Para ello debe seguir la siguiente ruta "Setting/Preferences/Editor". Adicionalmente el editor cuenta con varias herramientas básicas como copiar, cortar, pegar.

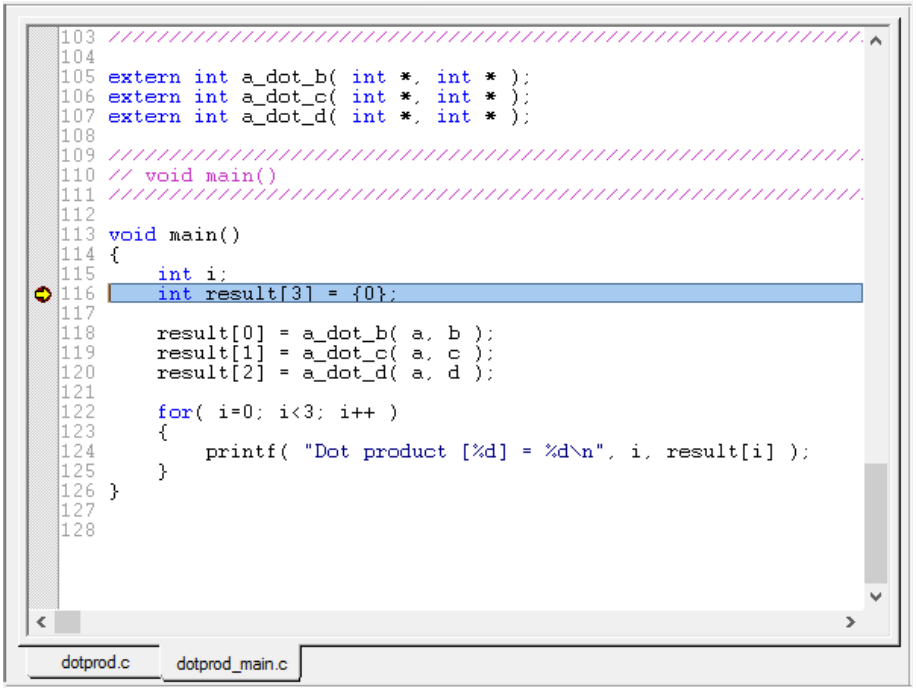


Figura 6: Ventana de edición.

Además el editor permite abrir múltiples archivos fuente, y para brindar un acceso fluido entre estos, el editor crea una pestaña para cada uno de ellos. El editor también cuenta con la opción de poder observar el número de cada línea de código, herramienta que da la posibilidad de ir a un número de línea especifica en caso de ser necesaria una modificación o algo que se requiera en dicho momento, también en la parte exterior de dicho editor pueden incluirse varios íconos como los de interrupción, favoritos o actual posición del PC. Observarse la Figura 6

Para abrir la ventana de salida dirigirse a “View/Output Window”, Esta ventana es la responsable de informar al programador a través de mensajes de texto estándar de entrada y salida el estado de compilación del algoritmo que se encuentre desarrollando en dicho instante, informándole si la compilación del código es exitosa o no, también esta ventana da información del estado y cantidad de errores que se cometieron en el código compilado, esta ventana maneja toda la información por medio de dos pestañas quienes son la de consola y la de construcción.

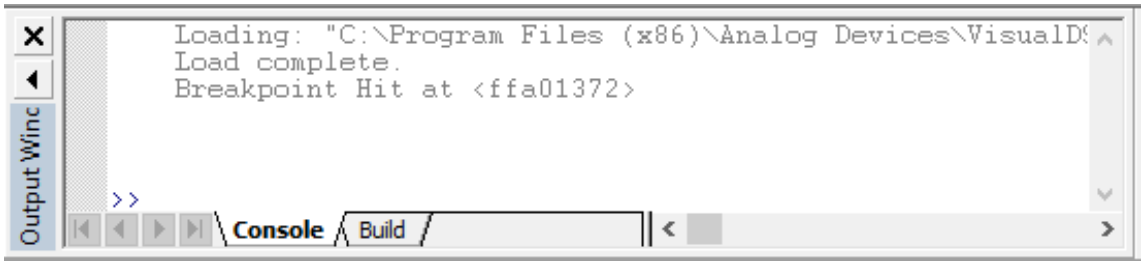


Figura 7: Pestaña de consola.

La pestaña de "Console" tiene una visión semejante a una consola de Windows o Linux que permite ejecutar comandos script, como tambien autocompletarlos a medida que se escriben, o moverse entre los comandos anteriores por medio del teclado, sirve además para ver mensajes de Usuario a través de las librerías "Stdio". Ver Figura 7.

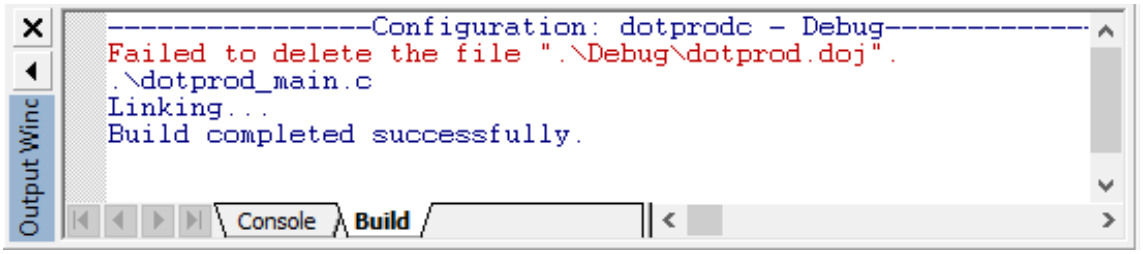


Figura 8: Pestaña de construcción.

La pestaña "Build" sirve para ver los mensajes de error durante la construcción y compilación de los algoritmos, para corregir los errores generados basta con dar clic sobre dicho error en la pestaña y eso te llevara hasta la ventana del editor donde se encuentre dicho error, si fue más de un error generado puede ir moviéndose entre errores con los íconos de "Next Error" y "Prev Error". Ver Figura 8.

Los mensajes que genera la herramienta de desarrollo están catalogados en una serie de niveles, esto ayuda a determinar si el estado del error es crítico o no, también advertencias referentes al uso de diferentes instrucciones del lenguaje de programación que se este usando. De forma general estos errores están catalogados en error fatal, error, advertencia y observación.

Las ventanas de depuración se encuentran en diferentes menús, la ruta más común seria por medio del menú "Register"(Timers, UARTs, Cycles), pero también se pueden encontrar algunas en el menú "View" (Image Viewer) o tambien en el menu "Memory" (BLACKFIN Memory). Las diferentes ventanas tienen por objetivo ayudar al diseñador a corregir y mejorar el código en tiempo real, generalmente estas ventanas son usadas por personas que trabajan con el lenguaje ensamblador, esto no quiere decir que no se puedan usar con los demás lenguajes de programación.

Entre las que tienen uso más frecuente en la depuración se encuentran el "Data Register File" quien puede ser llamado por medio de la ruta "Register/Core/Data Register File"' en este se observan los datos que se encuentran almacenados en los registros de uso especifico R0-R7, otro ejemplo seria la que se usa para observar las imágenes "Image Viewer" a quien se accede a través de la ruta "View/Debug Windows/Image Viewer".

La barra de menú y herramientas proporciona entre el entorno gran variedad de menús e íconos respectivamente, para facilitarle al programador la forma de acceder a los diferentes recursos del entorno y así poder realizar las tareas en forma eficaz y

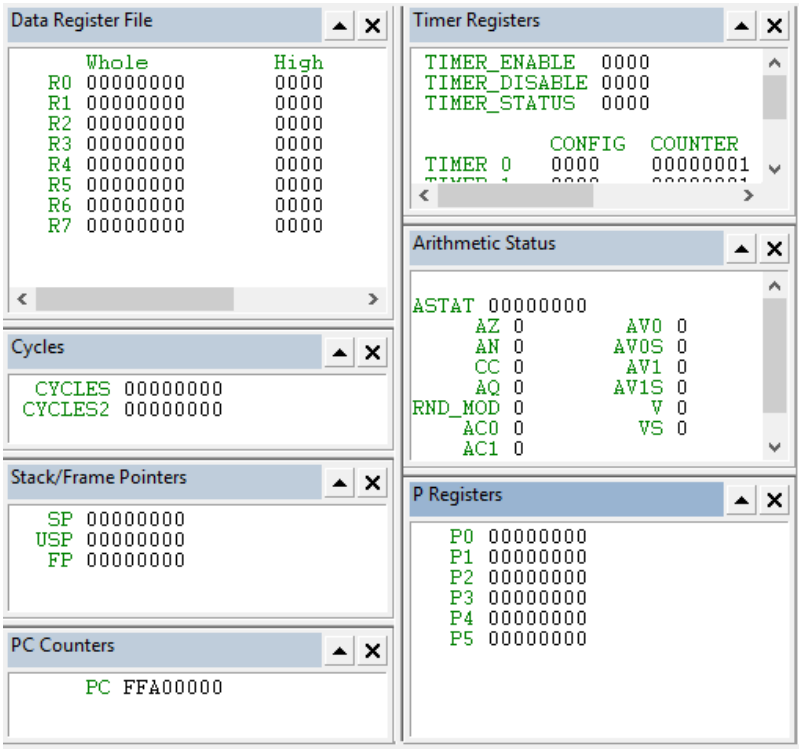


Figura 9: Ventanas de depuración.

eficiente.

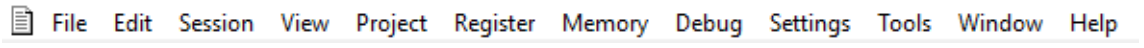


Figura 10: Barra de menú.

En La Figura 10. se pueden observar los diferentes componentes de la barra de menú, dentro de ellos podemos encontrar diferentes submenus que van desde realizar una simple tarea como guardar , hasta realizar tareas tan complejas como correr el código, por lo general muchos componentes que están contenidos en la barra de menú se hacen visibles como íconos en la barra de herramientas.

En la barra de herramientas se pueden encontrar una variedad de íconos que tienen diferentes funciones como la de compilación y ejecución del algoritmo con los ícono de "build proyect, run", además de contar con funciones que brindan la opción de ejecutar el algoritmo paso paso con los "break points" por medio de los íconos "step into, step over", también se encuentran herramientas que brindan la oportunidad de hacer algunos tipos de personalización que ofrece el entorno por defecto con los íconos "tile horizontal, tile vertical, cascade windows", y por supuesto también se encuentran los íconos de uso común como lo son "cut, paste , copy, save, open" entre otros como de observa en la Figura 11.

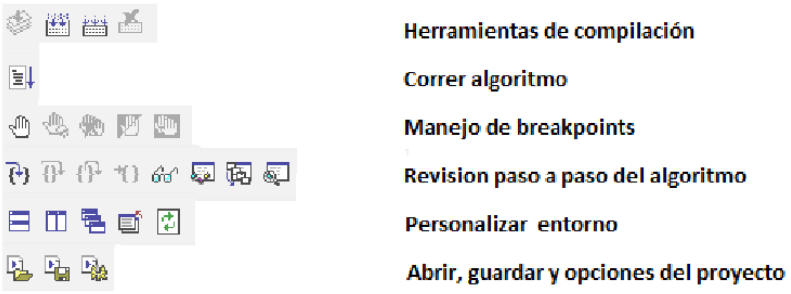


Figura 11: Principales íconos de la barra de herramientas.

.5.2. Crear sesión

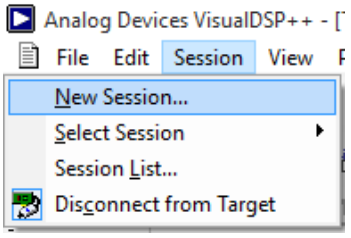


Figura 12: Ruta crear sesión.

Para inicializarse con VisualDSP++, como primera instancia se debe crear una "Sesión" si esta no ha sido creada con anterioridad, para ello se debe llamar al asistente de configuración de sesión dirigiéndose a "Session/New Session". Figura 12.

Ya en el asistente como primera medida se debe seleccionar la tarjeta que se utilizara en dicho proyecto, para este caso en particular se seleccionara la tarjeta ADSP BF-533. Ver Figura 13.

VisualDSP++ permite crear diferentes tipos de sesiones, en unas se puede trabajar directamente sobre el procesador y en otras sobre un modelo virtual de la tarjeta (simulador) que trae el software por defecto. Para esta implementación se trabajara en el simulador durante las etapas de depuración para manejo de errores y demás, como también se creara una sesión donde se utilice la tarjeta Blackfin BF-533 para pruebas finales y verificación del objetivo principal de tiempo real. Esto quiere decir que se crearan dos sesiones diferentes tanto de simulador como de hardware y se pasara de una a la otra dependiendo de la etapa en que se encuentre este trabajo. En la Figura 14 se ilustra como crear la sesión para utilizar el hardware ADSP BF-533. Cabe destacar que la ventaja más importante que trae el simulador es que se puede utilizar a plenitud las herramientas de depuración que trae el entorno pero a su vez el simulador afectara enormemente el tiempo de ejecución del programa.

Llegada a la opción "Select Platform" se debe seleccionar la opción ADSP BF-533 EZ-KIT Lite via Debut Agent en el caso que se este trabajando sobre la tarjeta o la opción ADSP BF-5xx Single Processor Simulator en el caso de que se trabajando

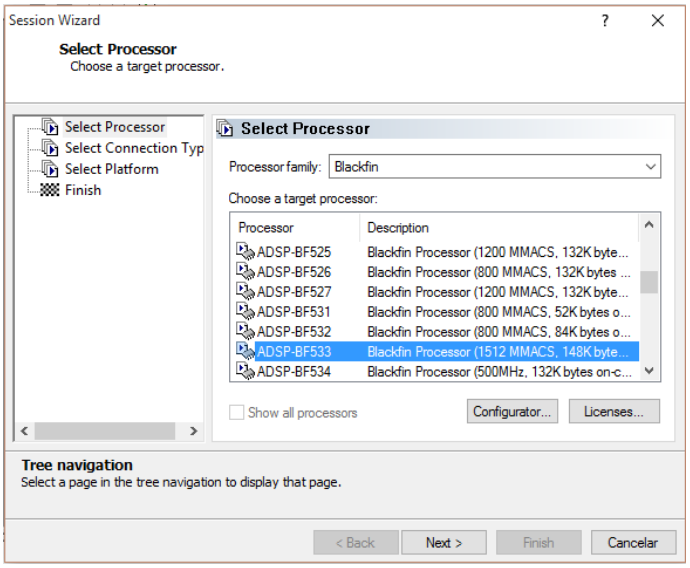


Figura 13: Seleccionando la tarjeta.

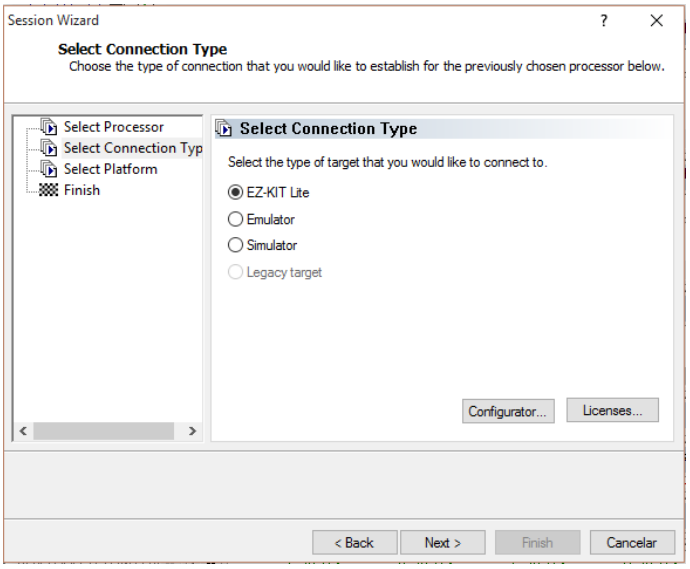


Figura 14: Seleccionando el tipo de sesión.

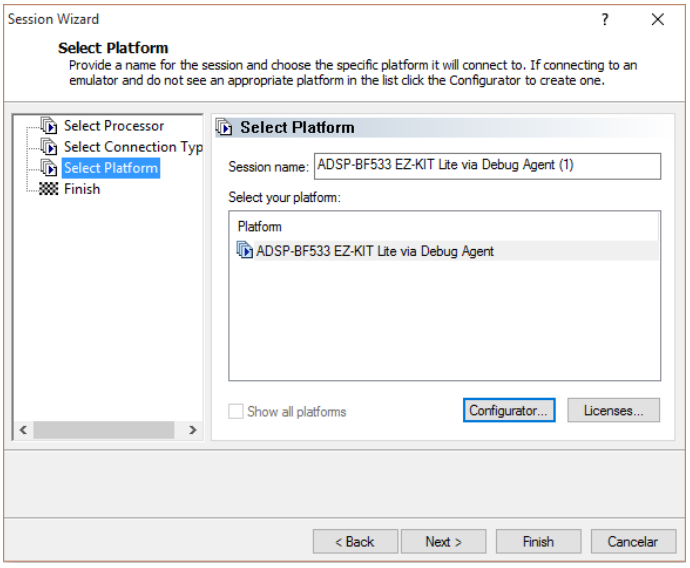


Figura 15: Seleccionando la plataforma.

sobre el simulador, Vease Figura 15 donde se ilustra para hardware.

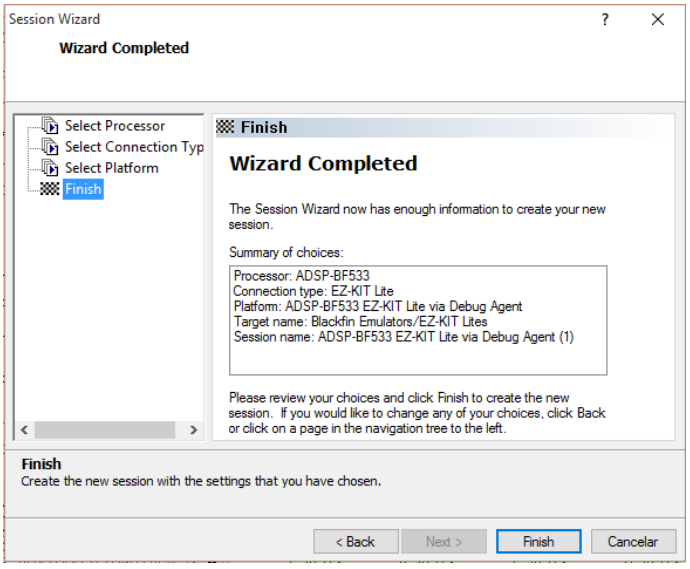


Figura 16: Verificando configuraciones de la sesión.

Por último el asistente mostrara todas las configuraciones seleccionadas, donde se puede verificar que todo este correcto y se puede proceder a finalizar con la creación de la sesión, seleccionando la opción “Finish”. Figura 16

En la Figura 17, se puede observar la forma que toma el entorno de VisualDSP++ después de creada una sesión. A partir de este momento ya se esta en capacidad de empezar a crear un proyecto.

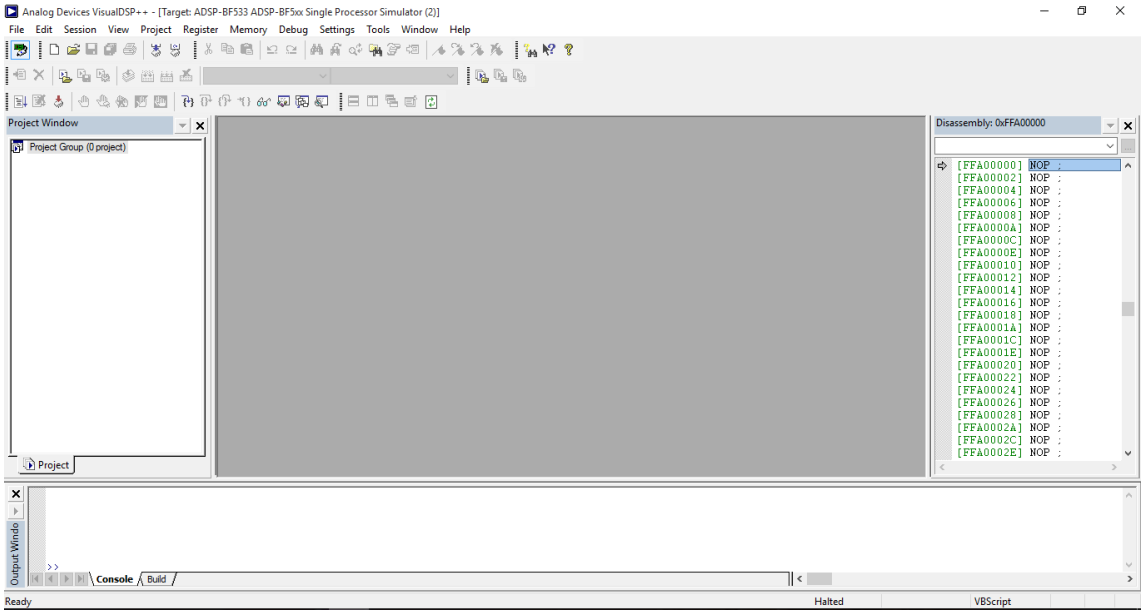


Figura 17: Sesión creada correctamente.

.5.3. Crear proyecto

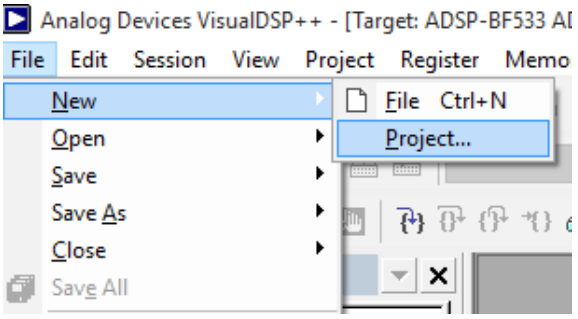


Figura 18: Ruta crear proyecto.

Para crear un proyecto dirigirse a la pestaña “File/New/Project” como se muestra el la Figura 18, esto abrirá el asistente de configuración de proyectos.

Después de abierto el asistente pedirá que se especifique que tipo de proyecto se desea crear, para este ejemplo simplemente se creara una aplicación estándar, adicionalmente se debe dar un nombre al proyecto y seleccionar la ruta donde se desea almacenar dicho proyecto, observar Figura 19.

Después de especificado el tipo de proyecto se da en siguiente y se procederá a seleccionar el tipo de procesador a utilizar en el proyecto, para este ejemplo se selecciona el ADSP BF-533 como se ilustra en la Figura 20.

Luego el asistente preguntara si se desea incluir un archivo en el proyecto y de que tipo (Assembly, c, c++). Figura 21.

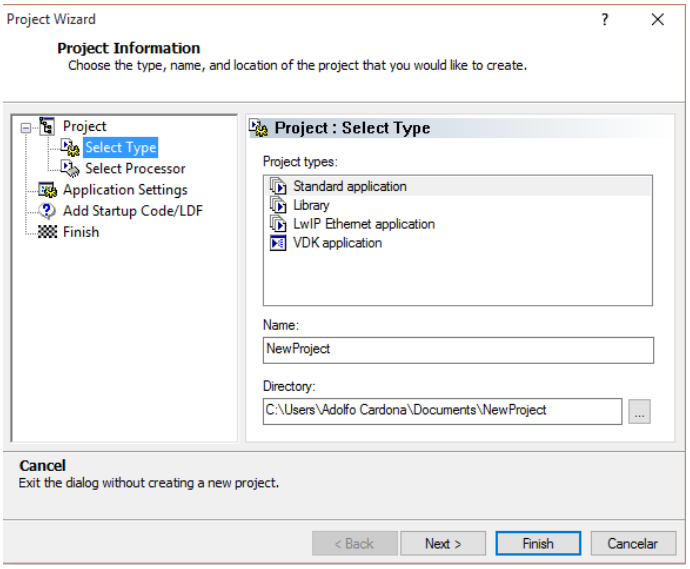


Figura 19: Seleccionando tipo de proyecto.

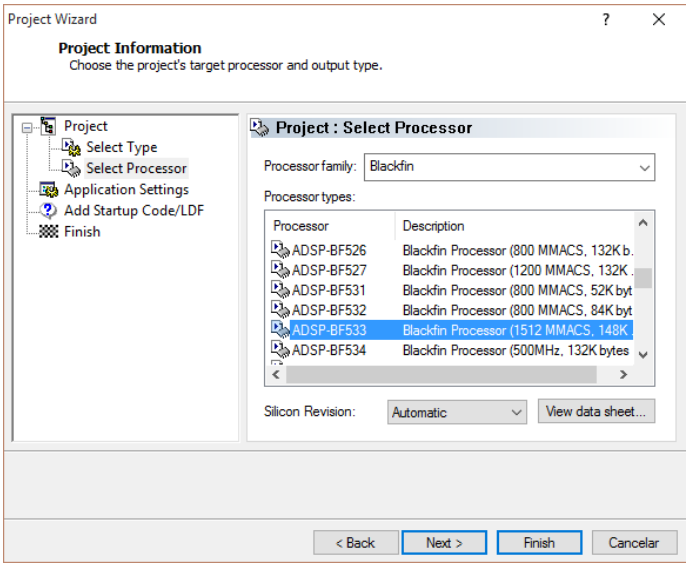


Figura 20: Seleccionando tipo de procesador.

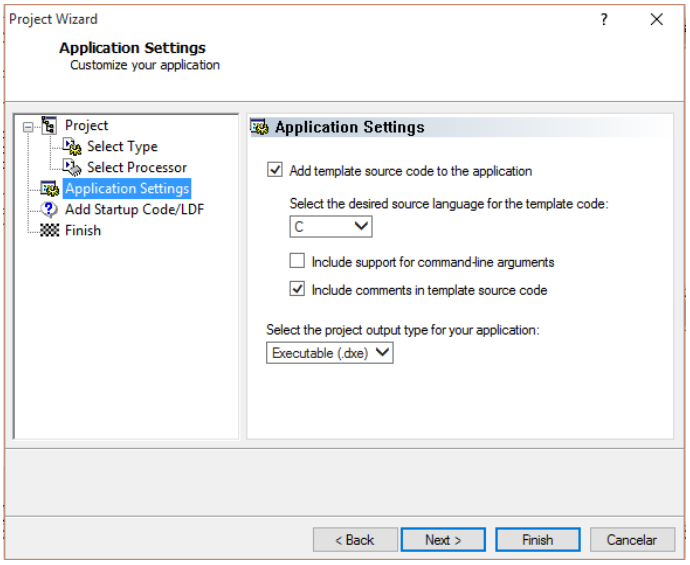


Figura 21: Configurando archivos del proyecto.

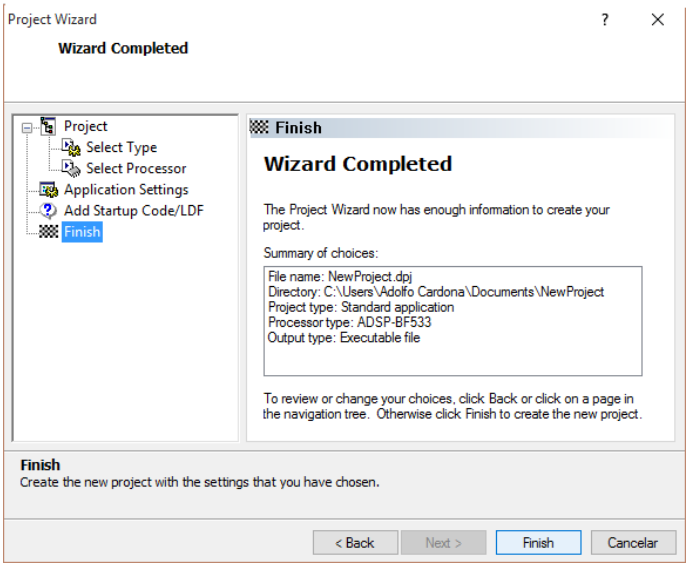


Figura 22: Verificando configuraciones del proyecto.

En la Figura 22, el asistente de configuración del proyecto pedirá revisar todas las configuraciones anteriores, si todo es correcto se procederá a finalizar el asistente y crear el proyecto.

.5.4. Crear nuevo archivo

Para desarrollar un algoritmo, en primera instancia se crea un archivo mediante la siguiente ruta " file/new/file"como se muestra en la Figura 23.

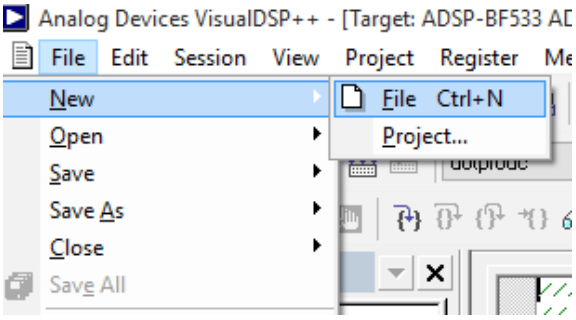


Figura 23: Ruta crear archivo.

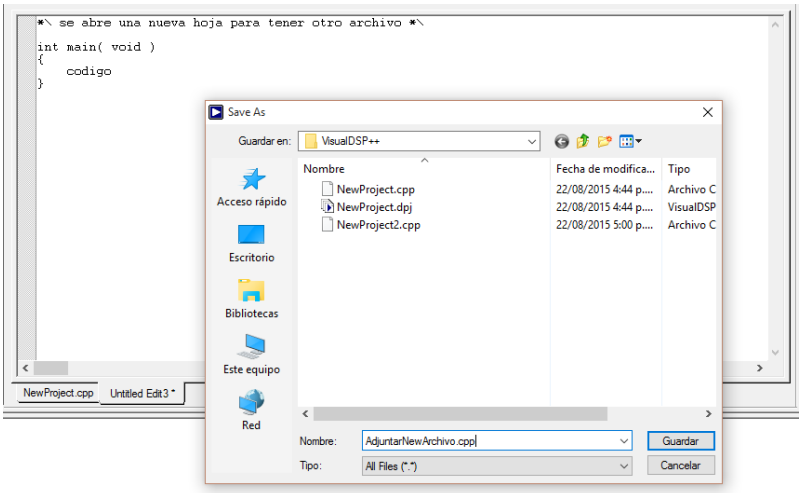


Figura 24: Guardando archivo en carpeta.

Posteriormente se procede a guardar el archivo para darle un nombre con la ayuda del ícono "save as", quien genera una ventana asistente a través de la cual se le da el nombre al archivo como se muestra en la siguiente Figura 24, ya teniendo creado el archivo se procede a trabajar en el editor es decir se comienza a implementar el código del algoritmo en cuestión.

.5.5. Adjuntar nuevo archivo

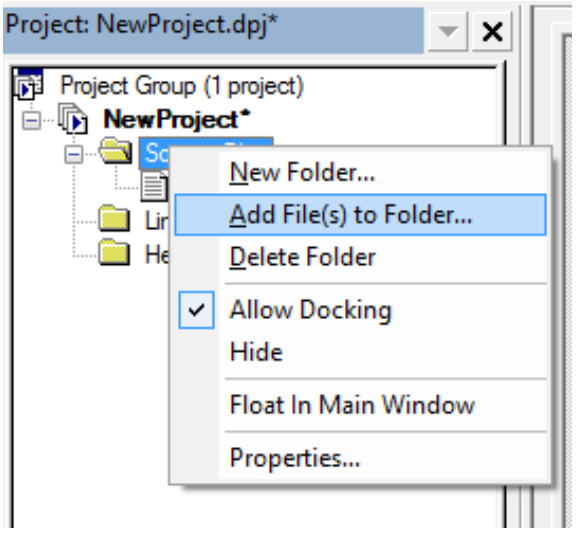


Figura 25: Adjuntando archivo al proyecto.

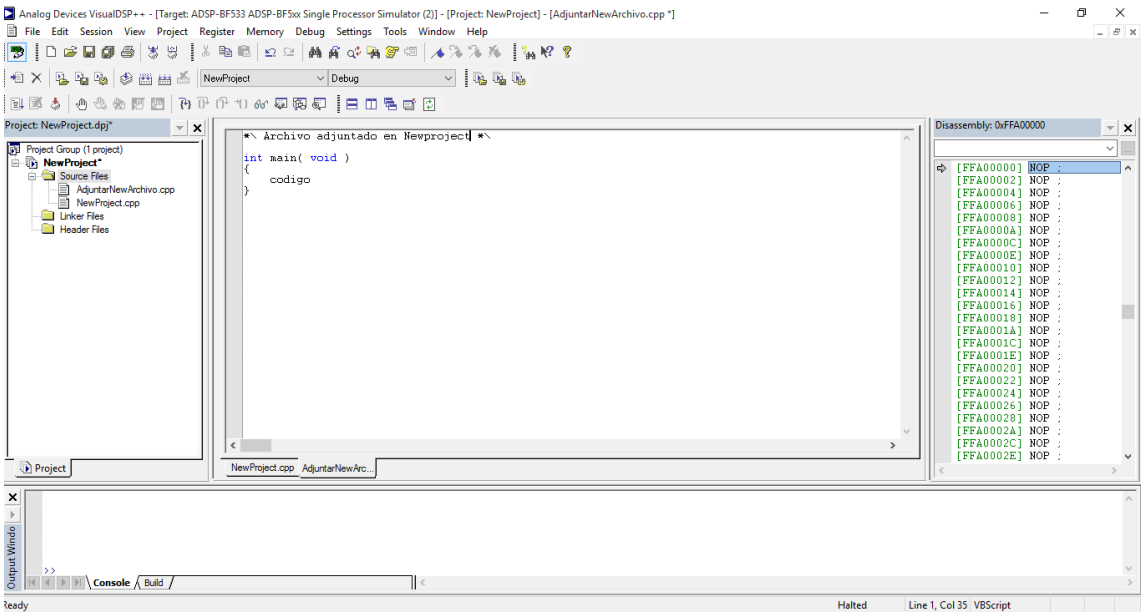


Figura 26: Adjuntar archivo.

Para adjuntar un nuevo archivo al proyecto, se debe seleccionar el archivo que se encuentra en la ventana "proyect window", quien ya debe tener como nombre el nombre del proyecto. Dentro de la ventana se selecciona la carpeta de proyecto y después se le da clic derecho, donde se desprende un menú como aparece en la siguiente Figura 25. Desde este menú se da clic izquierdo sobre "Add file(s) to folder", y aparecerá la ventana asistente para seleccionar el archivo que se desee adjuntar, después de seleccionado se abre el archivo en la ventana editor en una nueva hoja como se muestra en la Figura 26.

.5.6. Compilación y ejecutar el programa

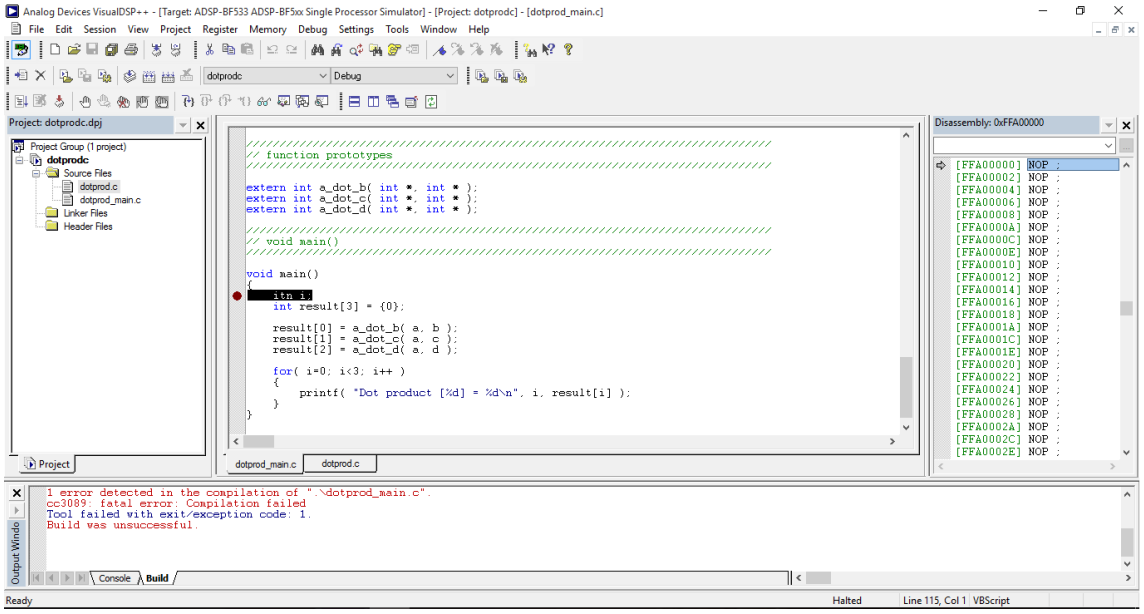


Figura 27: Fallo de compilación.

A medida que se va desarrollando el algoritmo la herramienta de compilación también puede ser muy práctica, puesto que ayuda al desarrollador a detectar con facilidad los errores con mensajes de texto que se muestran mediante la ventana de salida en la pestaña "build", como se observa en la Figura 27.

Figura 28: Compilación correcta.

De igual forma esta herramienta también nos indica cuando ejecutamos un algoritmo si fue exitosa la compilación, como se muestra en la Figura 28

.6. Conjunto de instrucciones

La sintaxis de las instrucciones es caso insensible, esto quiere decir que las letras mayúsculas y minúsculas pueden ser usadas y mezcladas arbitrariamente, el ensamblador trata los nombres de los registros y las instrucciones de esta misma manera, por lo que R3.L, r3.l, R3.l, r3.L; significan exactamente lo mismo.

La entrada del ensamblador es de formato libre, por lo que una instrucción puede aparecer en cualquier parte de la línea, o puede extenderse por múltiples líneas, o más que una instrucción puede aparecer en la misma línea.

Las instrucciones deben terminar con un punto y coma " ; " varias instrucciones pueden ser colocadas en la misma línea a discreción del programador solo debe finalizar cada una con " ; ", para el caso que se deseen ejecutar varias instrucciones en un solo ciclo de reloj estas deben ser separadas con una coma ",".

El ensamblador soporta varios tipos de comentarios, " // " el doble slash se coloca al inicio de la la línea indicando que dicha línea es un comentario; cuanto se desee colocar comentarios que abarquen más de una línea se utiliza " /* " " */ " encerrando todo el comentario[22].

.6.1. Cargar

Carga un inmediato en un registro: Esta instrucción puede cargar una contante o inmediato de 16-bits tanto en la parte baja como en la parte alta de un registro ("R7-0, P5-0, SP, FP, I3-0, M3-0, B3-0, L3-0"). Los registros-A ("A0, A1") 40-bits pueden ser cargados únicamente con ceros. Si se desea cargar un registro con inmediatos de 32-bits se deben realizar dos operaciones separadas.

$Reg = Constante;$

```
/* Cargar un inmediato en la parte baja de un registro-D y
   se aplica cero extendido a la parte alta del registro-D */
R7 = 63 (z) ;
/* Cargar un inmediato en la parte baja de un registro-P y
   se aplica cero extendido a la parte alta del registro-P */
P3 = 12 (z) ;
/* Cargar -344 en R0.L y aplica signo extendido en R0.H */
R0 = -344 (x) ;
/* Cargar 436 en R7.L y aplica cero extendido en R7.H */
R7 = 436 (z) ;
/* Cargar 0x89AB en M2.L y aplica cero extendido en M2.H */
M2 = 0x89ab (z) ;
/* Cargar 0x1234 en P1.L y aplica cero extendido en P1.H */
P1 = 0x1234 (z) ;
/* Cargar 0x3456 en M3.L y aplica signo extendido en M3.H */
M3 = 0x3456 (x) ;
/* Cargar 0xBCDE en L3.H */
L3.h = 0xbcde ;
// Cargar un inmediato en un acumulador de 40 bits
A1 = A0 = 0;
```

Cargar un registro puntero: Esta instrucción carga una palabra de 32-bits en un registro-P ("P5-0"), la palabra proviene de una posición de memoria a la que apunta un Preg ("P5-0, SP, FP"). El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$registro - P = [Preg];$

```
// Carga en un registro-P el valor al que apunta un registro-P
P3 = [P2];
// Carga en P5 el dato al que apunta P0 y post-incrementa en 4 bytes P0
P5 = [P0++];
// Carga en P2 el dato al que apunta SP y post-decrementa en 4 bytes SP
P2 = [SP--];
// Carga en P3 el dato al que apunta P2 con un offset de 8 bytes
P3 = [P2+8];
// Carga en P0 el dato al que apunta P2 con un offset de 0x4008 bytes
P0 = [P2+0x4008];
// Carga en P1 el dato al que apunta FP con un offset de menos 16 bytes
P1 = [FP-16];
```

Cargar un registro de datos: Esta instrucción carga una palabra de 32-bits en un registro-D ("R7-0"), la palabra proviene de una posición de memoria a la que apunta un registro-P, un registro-I ("I3-0"), o el puntero de marco ("FP", por sus siglas en ingles). El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$Dreg = [Direccion\_Indirecta];$

```
// Carga en un registro-D el valor al que apunta un registro-P
R3 = [P0];
// Carga en R7 el dato al que apunta P1 y post-inc en 4 bytes P1
R7 = [P1++];
// Carga en R2 el dato al que apunta SP y post-dec en 4 bytes SP
R2 = [SP--];
// Carga en R6 el dato al que apunta P2 con un offset de 12 bytes
R6 = [P2+12];
// Carga en R0 el dato al que apunta P4 con un offset de 0x800C bytes
R0 = [P4+0x800C];
// Carga en R1 el dato al que apunta P0 y post-inc en 4*P1 bytes a P0
R1 = [P0++P1];
// Carga en R5 el dato al que apunta FP con un offset de menos 12 bytes
R5 = [FP-12];
// Carga en un registro-D el dato al que apunta un registro-I
R2 = [I2];
// Carga en R0 el dato al que apunta I0 y post-inc I0 en 4 bytes
R0 = [I0++];
// Carga en R0 el dato al que apunta I0 y post-dec I0 en 4 bytes
R0 = [I0--];
```

Carga 16-bits con cero extendido en un registro-D: Esta instrucción carga un dato de 16-bits en la parte baja de un registro-D y extiende a ceros la parte alta de dicho registro, el dato proviene de una posición de memoria a la que apunta un registro-P. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$Dreg = W[Preg](Z);$

```
/* Carga en la parte baja de R3, 16 bits a los que apunta P0 y
   extiende a ceros la parte alta del registro */
R3 = W[P0](Z);
/* Carga en R7.L, 16 bits a los que apunta P1 y extiende a ceros
   la parte alta del registro y post-inc P1 en 2 bytes */
R7 = W[P1++](Z);
/* Carga en R2.L, 16 bits a los que apunta SP y extiende a ceros
   la parte alta del registro y post-dec SP en 2 bytes */
R2 = W[SP--](Z);
/* Carga en R6.L, 16 bits a los que apunta P2 con un offset de
   12 bytes y extiende a ceros la parte alta del registro */
R6 = W[P2+12](Z);
```

```
/* Carga en R0.L, 16 bits a los que apunta P4 con un offset de
   0x8004 bytes y extiende a ceros la parte alta del registro */
R0 = W[P4+0x8004](Z);
/* Carga en R1.L, 16 bits a los que apunta P0 y extiende a ceros
   la parte alta del registro y post-inc P0 en 2*P1 bytes */
R1 = W[P0++P1](Z);
```

Carga 16-bits con signo extendido en un registro-D: Esta instrucción carga un dato de 16-bits en la parte baja de un registro-D y aplica signo extendido a la parte alta de dicho registro, el dato proviene de una posición de memoria a la que apunta un registro-P. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$Dreg = W[Preg](X);$

```
/* Carga en R3.L, 16 bits a los que apunta P0 y aplica
   signo extendido a la parte alta del registro */
R3 = W[P0](X);
/* Carga en R7.L, 16 bits a los que apunta P1, aplica signo extendido
   a la parte alta del registro y post-inc P1 en 2 bytes */
R7 = W[P1++](X);
/* Carga en R2.L, 16 bits a los que apunta SP, aplica signo extendido
   a la parte alta del registro y post-dec SP en 2 bytes */
R2 = W[SP--](X);
/* Carga en R6.L, 16 bits a los que apunta P2 con un offset de 12
   bytes y aplica signo extendido a la parte alta del registro */
R6 = W[P2+12](X);
/* Carga en R0.L, 16 bits a los que apunta P4 con un offset de 0x800E
   bytes y aplica signo extendido a la parte alta del registro */
R0 = W[P4+0x800E](X);
/* Carga en R1.L, 16 bits a los que apunta P0 y aplica signo extendido
   a la parte alta del registro y post-inc P0 en 2*P1 bytes */
R1 = W[P0++P1](X);
```

Carga 16-bits en la parte alta de un registro-D: Esta instrucción carga un dato de 16-bits en la parte alta de un registro-D, el dato proviene de una posición de memoria a la que apunta un registro-I o un registro-P. La operación no afecta a la mitad menos significativa. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$Dreg\_H = W[Direccion\_Indirecta];$

```
/* Carga en la parte alta de un registro-D, 16 bits a los que
   apunta un registro-I */
R3.H = W[I1];
/* Carga en la parte alta de R7, 16 bits a los que apunta I3 y
   post-inc en 2 bytes I3 */
R7.H = W[I3++];
/* Carga en la parte alta de R1, 16 bits a los que apunta I0 y
   post-dec en 2 bytes I0 */
```

```
R1.H = W[I0--];
/* Carga en la parte alta de un registro-D, 16 bits a los que
   apunta un registro-P */
R2.H = W[P4];
/* Carga en la parte alta de R5, 16 bits a los que apunta P2 y
   post-inc en 2*P0 a P2 */
R5.H = W[P2++P0];
```

Carga 16-bits en la parte baja de un registro-D: Esta instrucción carga un dato de 16-bits en la parte baja de un registro-D, el dato proviene de una posición de memoria a la que apunta un registro-P o un registro-I. La operación no afecta a la mitad más significativa. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$Dreg\_L = W[Ireg];$

```
/* Carga en la parte baja de R3, 16 Bits a los que apunta I1 */
R3.L = W[I1];
/* Carga en R7.L, 16 bits a los que apunta I3 y post-inc I3 */
R7.L = W[I3++];
/* Carga en R1.L, 16 bits a los que apunta I0 y post-dec I0 */
R1.L = W[I0--];
/* Carga en R2.L, 16 bits a los que apunta P4 */
R2.L = W[P4];
/* Carga en R5.L, 16 bits a los que apunta P2 y post-inc P2 en 2*P0 */
R5.L = W[P2++P0];
```

Carga un byte con cero extendido en un registro-D: Esta instrucción carga un dato de 8-bits en la parte baja de un registro-D. El dato proviene de una posición de memoria indicada por un registro-I o un registro-P. En los restantes 3-bytes se aplica la operación de cero extendido. El direccionamiento indirecto y offset no tienen restricciones por lo que no se producirá una excepción de acceso a la memoria.

$Dreg = B[Preg](Z);$

```
/* Carga en los 8 LSB de un registro-D, 8 bits a los que apunta un
   registro-P y extiende a ceros el resto del registro-D */
R3 = B[P0](Z);
/* Carga en los 8 LSB de R7, 8 bits a los que apunta P1, extiende
   a ceros el resto del registro y post-inc P1 en 1 byte */
R7 = B[P1++](Z);
/* Carga en los 8 LSB de R2, 8 bits a los que apunta SP, extiende
   a ceros el resto del registro y post-dec SP en 1 byte */
R2 = B[SP--](Z);
/* Carga en los 8 LSB de R0, 8 bits a los que apunta P4 con un offset
   de 0xFFFF800F y extiende a ceros el resto del registro */
R0 = B[P4+0xFFFF800F](Z);
```

Carga un byte con signo extendido: Esta instrucción carga un dato de 8-bits en la parte baja de un registro-D. El dato proviene de una posición de memoria indicada por un registro-I o un registro-P. En los restantes 3-bytes se aplica la operación

de signo extendido. El direccionamiento indirecto y offset no tienen restricciones por lo que no se producirá una excepción de acceso a la memoria.

$Dreg = B[Preg](X);$

```
/* Carga en los 8 LSB de un registro-D, 8 bits a los que apunta un
   registro-P y aplica signo extendido al resto del registro-D */
R3 = B[P0](X);
/* Carga en los 8 LSB de R7, 8 bits a los que apunta P1, aplica signo
   extendido al resto del registro y post-inc P1 en 1 byte */
R7 = B[P1++](X);
/* Carga en los 8 LSB de R2, 8 bits a los que apunta SP, aplica signo
   extendido al resto del registro y post-dec SP en 1 byte */
R2 = B[SP--](X);
/* Carga en los 8 LSB de R0, 8 bits a los que apunta P4 con un offset
   de 0xFFFF800F, aplica signo extendido al resto del registro */
R0 = B[P4+0xFFFF800F](X);
```

.6.2. Guardar

Guardar el contenido de un registro-P: Esta instrucción guarda una palabra de 32-bits en la dirección a la que apunta un Preg. El dato proviene de un registro-P. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$[Preg] = registro - P;$

```
// Guarda el valor de un registro-P en la direccion que
   apunta otro registro-P
[P2] = P3;
/* Guarda el valor de P5 en la direccion que apunta SP y
   post-inc el SP en 4 bytes*/
[SP++] = P5;
/* Guarda el valor de P2 en la direccion que apunta P0 y
   post-dec P0 en 4 bytes */
[P0--] = P2;
/* Guarda el valor de P3 en la direccion que apunta P2 con
   un offset de 8 bytes */
[P2+8] = P3;
/* Guarda el valor de P0 en la direccion que apunta P2 con
   un offset de 0x4444 byte  */
[P2+0x4444] = P0;
/* Guarda el valor de P1 en la direccion que apunta FP con
   un offset de menos 12 bytes */
[FP-12] = P1;
```

Guardar el contenido de un registro-D: Esta instrucción guarda una palabra de 32-bits en la dirección a la que apunta un Preg o un registro-I. El dato proviene de un registro-D. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada

se produce una excepción de acceso a la memoria.

$[Preg] = Dreg;$

```
/* Guarda el valor de un registro-D en la direccion que
   apunta un registro-P */
[P0] = R3;
/* Guarda el valor de R7 en la direccion que apunta P1 y
   post-inc P1 en 4 bytes */
[P1++] = R7;
/* Guarda el valor de R2 en la direccion que apunta SP y
   post-dec el SP en 4 bytes*/
[SP--] = R2;
/* Guarda el valor de R6 en la direccion que apunta P2 con
   un offset de 12 bytes */
[P2+12] = R6;
/* Guarda el valor de R0 en la direccion que apunta P4 con
   un offset de menos 0x1004 byte */
[P4-0x1004] = R0;
/* Guarda el valor de R1, en la direccion que apunta P0 y
   post-inc P0 en 4*P1 */
[P0++P1] = R1;
/* Guarda el valor de R5, en la direccion que apunta FP con
   un offset de menos 28 bytes */
[FP-28] = R5;
/* Guarda el valor de R2, en la direccion que apunta I2 */
[I2] = R2;
/* Guarda el valor de R0, en la direccion que apunta I0 y
   post-inc I0 en 4 bytes */
[I0++] = R0;
/* Guarda el valor de R0, en la direccion que apunta I0 y
   post-dec I0 en 4 bytes */
[I0--] = R0;
/* Guarda el valor de R7, en la direccion que apunta I3 y
   post-inc I3 en 4*M0 bytes */
[I3++M0] = R7;
```

Guardar el contenido de la parte alta de un registro-D: Esta instrucción guarda un dato de 16-bits en la porción de memoria de 16-bits a la que apunta un registro-P o un registro-I. El dato proviene de la parte alta de un registro-D. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$W[Direccion\_Indirecta] = Dreg\_hi;$

```
/* Guarda los 16 MSB de un registro-D, en los 16 bits a los
   que apunta un registro-I */
W[I1] = R3.H;
/* Guarda los 16 MSB de R7, en los 16 bits a los que apunta
   I3 post-inc I3 en 2 bytes */
W[I3++] = R7.H;
```

```
/* Guarda los 16 MSB de R1, en los 16 bits a los que apunta
 I0 y post-dec I0 en 2 bytes */
W[I0--] = R1.H;
/* Guarda los 16 MSB de R2, en los 16 bits a los que apunta
   P4 */
W[P4] = R2.H;
/* Guarda los 16 MSB de R5, en los 16 bits a los que apunta
   P2 y post-inc P2 en 2*P0 bytes */
W[P2++P0] = R5.H;
```

Guardar el contenido de la parte baja de un registro-D: Esta instrucción guarda un dato de 16-bits en la porción de memoria de 16-bits a la que apunta un registro-P o un registro-I. El dato proviene de la parte baja de un registro-D. El direccionamiento indirecto y offset deben ser múltiplos de 2 para mantener la alineación a 2-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$W[Direccion\_Indirecta] = Dreg\_lo;$

```
/* Guarda los 16 LSB de R3, en los 16 bits a los apunta I1 */
W[I1] = R3.L;
/* Guarda los 16 LSB de R3, en los 16 bits a los apunta P0 */
W[P0] = R3;
/* Guarda los 16 LSB de R7, en los 16 bits a los apunta I3 y
   post-inc I3 en 2 bytes */
W[I3++] = R7.L;
/* Guarda los 16 LSB de R1, en los 16 bits a los apunta I0 y
   post-dec I0 en 2 bytes */
W[I0--] = R1.L;
/* Guarda los 16 LSB de R2, en los 16 bits a los apunta P4 */
W[P4] = R2.L;
/* Guarda los 16 LSB de R7, en los 16 bits a los apunta P1 y
   post-inc P1 en 2 bytes */
W[P1++] = R7;
/* Guarda los 16 LSB de R2, en los 16 bits a los apunta SP y
   post-dec SP en 2 bytes */
W[SP--] = R2;
/* Guarda los 16 LSB de R6, en los 16 bits a los apunta P2 con
   un offset de 12 bytes */
W[P2+12] = R6;
/* Guarda los 16 LSB de R0, en los 16 bits a los apunta P4 con
   un offset de menos 0x200C bytes */
W[P4-0x200C ] = R0;
/* Guarda los 16 LSB de R5, en los 16 bits a los apunta P2 y
   post-inc P2 en 2*P0 bytes */
W[P2++P0] = R5.L;
```

Guardar un byte de un registro-D: Esta instrucción guarda un dato de 8-bits en la porción de memoria de 8-bits a la que apunta un registro-P. El dato proviene de los 8-bits menos significativos de un registro-D. El direccionamiento indirecto y offset no tienen restricciones por lo que no se producirá una excepción de acceso a la memoria.

$B[Direccion\_Indirecta] = Dreg;$

```
/* Guarda los 8 LSB de R3, en los 8 bits a los que apunta
   P0 */
B[P0] = R3;
/* Guarda los 8 LSB de R7, en los 8 bits a los que apunta
   P1 y post-inc P1 en 1 byte */
B[P1++] = R7;
/* Guarda los 8 LSB de R2, en los 8 bits a los que apunta
   SP y post-dec SP en 1 byte */
B[SP--] = R2;
/* Guarda los 8 LSB de R0, en los 8 bits a los que apunta
   P4 con un offset de 0x100F bytes */
B[P4+0x100F] = R0;
/* Guarda los 8 LSB de R0, en los 8 bits a los que apunta
   P4 con un offset de menos 0x53F bytes */
B[P4-0x53F] = R0;
```

.6.3. Mover

Mover contenido de un registro: Esta instrucción mueve o copia el contenido de un registro fuente en un registro destino, esta función no afecta el contenido del registro fuente. Todo movimiento de un registro más pequeño a un registro más grande es signo extendido, todo movimiento desde un registro-A de 40-bits a un registro-D de 32-bits soporta la operación de saturación.

$reg\_dest = reg\_fuente;$

```
/* Mueve el dato guardado en R0 a R3 */
R3 = R0;
/* Mueve el dato guardado en P2 a R7 */
R7 = P2;
/* Mueve el dato guardado en los 32 LSB a A0 en el registro
   par R2 */
R2 = A0;
/* Mueve el dato guardado en A1 al acumulador A0 */
A0 = A1;
/* Mueve el dato guardado en A0 al acumulador A1 */
A1 = A0;
/* Mueve el dato guardado en R7 a los 32 LSB de A0 */
A0 = R7;
/* Mueve el dato guardado en R3 a los 32 LSB de A1 */
A1 = R3;
/* Mueve el dato guardado en P0 al registro del sistema
   RETN (se debe estar en modo supervisor) */
RETN = P0;
/* Mueve el dato guardado en los 32 LSB a A1 al registro
   impar R7 */
R7 = A1;
/* Mueve el dato guardado en los 32 LSB a A0 al registro par R0
   y realiza  operacion de escalamiento truncacion y saturacion */
R0 = A0 (ISS2);
```

Mover con condicional: Esta instrucción mueve el contenido de un registro fuente a un registro destino si y solo si el bit CC es verdadero. Los registros fuente y destino son solo registros-D o registros-P.

$IF\ CC\ reg\_dest = reg\_fuente;$

```
/* Se mueve R0 a R3 si CC==1 */
if CC R3 = R0;
/* Se mueve P4 a R2 si CC==1 */
if CC R2 = P4;
/* Se mueve R7 a P0 si CC==1 */
if CC P0 = R7;
/* Se mueve P5 a P2 si CC==1 */
if CC P2 = P5;
/* Se mueve R0 a R3 si CC==0 */
if ! CC R3 = R0;
/* Se mueve P4 a R2 si CC==0 */
if ! CC R2 = P4;
/* Se mueve R7 a P0 si CC==0 */
if ! CC P0 = R7;
/* Se mueve P5 a P2 si CC==0 */
if ! CC P2 = P5;
```

Mover un dato de 16-bits a un registro de 32-bits con cero extendido: Esta instrucción mueve los 16-bits menos significativos de un registro fuente a un registro destino de 32-bits y realiza la operación de cero extendido. Esta instrucción solo soporta registros-D. La operación de cero extendido es apropiada solo para valores sin signo, si se utiliza con valores con signo, un dato pequeño de 16-bits negativo se convertirá en un valor positivo muy grande.

$dest\_reg = reg\_fuente\_L(Z);$

```
/* Mueve el dato guardado en los 16 LSB de R0 a R4 y  extiende
   a ceros el resto del registro */
R4 = R0.L (Z);
```

Mover un dato de 16-bits a un registro de 32-bits con signo extendido: Esta instrucción mueve los 16-bits menos significativos de un registro fuente a un registro destino de 32-bits y realiza la operación de signo extendido. Esta instrucción solo soporta registros-D.

$dest\_reg = reg\_fuente\_L(X);$

```
/* Mueve el dato guardado en los 16 LSB de R0 a R4 y aplica signo
   extendido al resto del registro */
R4 = R0.L(X);
/* Mueve el dato guardado en los 16 LSB de R0 a R4 y aplica signo
   extendido por defecto */
R4 = R0.l ;
```

Mover un dato de 16-bits a un registro de 32-bits: Esta instrucción mueve los 16-bits más significativos ó los 16-bits menos significativos de un registro destino, a la parte alta o baja respectivamente de un registro-D o un Acumulador. Esta instrucción no afecta a los otros 16-bits no especificados del registro de destino.

$reg\_dest\_L\_H = reg\_fuente\_L\_H;$
$reg\_dest\_L\_H = Acumulador(opc);$

```
/* Mueve el dato guardado en los 8 LSB de R1 a los 8 MSB de A0 */
A0.X = R1.L;
/* Mueve el dato guardado en los 8 LSB de R4 a los 8 MSB de A1 */
A1.X = R4.L;
/* Mueve el dato guardado en los 8 MSB de A0 a los 8 LSB de R7 */
R7.L = A0.X;
/* Mueve el dato guardado en los 8 MSB de A1 a los 8 LSB de R0 */
R0.L = A1.X;
/*Mueve el dato guardado en los 16 LSB de R2 a los LSB A0 */
A0.L = R2.L;
/*Mueve el dato guardado en los 16 LSB de R1 a los LSB A1 */
A1.L = R1.L;
/*Mueve el dato guardado en los 16 LSB de R5 a los LSB A0 */
A0.L = R5.L;
/*Mueve el dato guardado en los 16 LSB de R3 a los LSB A1 */
A1.L = R3.L;
/*Mueve el dato guardado en los 16 MSB de R7 a los MSB A0 */
A0.H = R7.H;
/*Mueve el dato guardado en los 16 MSB de R0 a los MSB A1 */
A1.H = R0.H;
/* Copia los 16 LSB de A0.W en R7.L con saturacion por defecto */
R7.L = A0 ;
/* Copia los 16 MSB de A1.W en R2.H con saturacion por defecto */
R2.H = A1 ;
/* Copia A0.L en R3.L y A1.H en R3.H. Ambas mitades deben ir
   a el mismo registro de destino. */
R3.L = A0, R3.H = A1;
/* Copia A1.H en R1.H y A0.L en R1.L. Ambas mitades deben ir
   a el mismo registro de destino. */
R1.H = A1, Rl.L = A0;
/* Copia A1.L en R0.H con Saturacion. Por Defecto. */
R0.H = A1 (IS);
/* Copia A0.H en R5.L; trunca A0.L; no satura.*/
R5.L = A0 (T);
/* Copia A0.H en R1.L con escalamiento, redondeo y saturacion. */
R1.L = A0 (S2RND);
/* Copia A1.L en R2.H con escalamiento y saturacion.*/
R2.H = A1 (ISS2);
/* Copia A0.H en R6.L con saturacion y despues redondeo. */
R6.L = A0 (IH);
```

Mover un dato de 8-bits a un registro de 32-bits con cero extendido: Esta instrucción mueve los 8-bits menos significativos de un registro fuente a un registro destino de 32-bits y realiza la operación de cero extendido. Esta instrucción solo soporta registros-D.

$reg\_dest = reg\_fuente\_Byte(Z);$

```
/* Copia los 8 LSB de R2 en R7 y extiende a ceros el resultado. */
R7 = R2.B (Z) ;
```

Mover un dato de 8-bits a un registro de 32-bits con signo extendido: Esta instrucción mueve los 8-bits menos significativos de un registro fuente a un registro destino de 32-bits y realiza la operación de signo extendido. Esta instrucción solo soporta registros-D.

$reg\_dest = reg\_fuente\_Byte(X);$

```
/* Mueve el dato guardado en los 8 LSB de R2 a R7 y aplica signo
   extendido por defecto */
R7 = R2.B;
/* Mueve el dato guardado en los 8 LSB de R2 a R7 y aplica signo
   extendido al resto del registro */
R7 = R2.B(X);
```

.6.4. Control del flujo del programa

Salto: La instrucción $JUMP$ sirve para forzar el contador del programa (PC, por sus siglas en ingles) para que tome un nuevo valor y de esta forma cambiar el flujo del programa. El valor en el Preg debe ser un número par (bit0=0) para mantener el alineamiento a 16-bits, un offset impar causara que el procesador invoque una excepción de acceso a la memoria

$JUMP$

```
/* Salta a la direccion absoluta contenida en un registro-P */
JUMP (P5);
/* Salto con offset a la direccion relativa indicada por P2,
   asi la direccion objetivo es PC + P2 */
JUMP (PC + P2) ;
/* Salto con offset positivo de 13-bits, asi la direccion
   objetivo es PC + 0x224 */
JUMP 0x224 ;
/* Salto con offset positivo de 13-bits, asi la direccion
   objetivo es PC + 0x224  (.S, Short) = 13-bits */
JUMP.S 0x224 ;
/* Salto con offset negativo de 25-bits, asi la direccion
   objetivo es PC + 0x1FA CE86 (.L, Long) = 25-bits*/
JUMP.L 0xFFFACE86 ;
/* Salto a etiqueta, El ensamblador resuelve el objetivo
   cuando es un offset abstracto */
JUMP etiqueta_usuario ;
```

Salto condicional: La instrucción de salto condicional obliga a un nuevo valor entrar en el (PC) para cambiar el flujo del programa si y solo si el valor del bit CC

es igual que 1..

IF CC JUMP

```
ï»¿/* Salto si y solo si CC = 1, el offset es negativo de 11-bits
     con salto predictivo (BP, por sus siglas en ingles) */
IF CC JUMP 0xFFFFFE08 (BP) ;
/* Salto si y solo si CC = 1, el offset es positivo de 11-bits,
   asi el salto es hacia adelante, sin salto predictivo */
IF CC JUMP 0x0B4 ;
/* Salto si y solo si CC = 1, el offset es negativo de 11-bits
     asi el salto es hacia atrÃ¡s, con salto predictivo */
IF CC JUMP 0xFFFFFE08 (BP) ;
/* Salto si y solo si CC = 0, el offset es negativo de 11-bits
     asi el salto es hacia atrÃ¡s, con salto predictivo */
IF !CC JUMP 0xFFFFFC22 (BP) ;
/* Salto si y solo si CC = 0, el offset es positivo de 11-bits
     asi el salto es hacia adelante, sin salto predictivo */
IF !CC JUMP 0x120 ;
/* Salto si y solo si CC = 0, el ensamblador resuelve el objetivo
   cuando es un offset abstracto */
IF !CC JUMP etiqueta_usuario ;
```

Llamar subrutina: La instrucción *CALL* llama a una subrutina desde una dirección indicada por un registro-P o mediante el uso de un offset de un PC relativo. Después de que la instrucción *CALL* se ejecuta, el registro *RETS* contiene la dirección de la siguiente instrucción. El valor en el registro-P debe ser un valor par para mantener la alineación de 16-bits.

CALL

```
/* Llamado a subrutina en la direccion absoluta
     indicada por un registro-P */
CALL ( P5 ) ;
/* Llamado a subrutina en la direccion relativa
     indicada por P2 */
CALL ( PC + P2 ) ;
/* Llamado a subrutina en la direccion relativa
     indicada por un inmediato */
CALL 0x123456 ;
/* Llamado a subrutina con etiqueta definida por el usuario, el
   ensamblador resuelve el objetivo cuando es un offset abstracto */
CALL etiqueta_usuario ;
```

Retorno: Las instrucciones de retorno tienen como función forzar el retorno de una subrutina, una interrupción NMI, una excepción, o la rutina de emulación.

RTS: Esta instrucción sirve para forzar el procesador a retornar desde una subrutina, esto lo hace cargando el valor que se encuentra en el registro RETS en el PC, haciendo que el procesador vaya a la siguiente instrucción que indica el registro RETS. Para anidar varias subrutinas, se debe guardar el valor del

registro RETS, ya que con el siguiente llamado de subrutina el registro RETS toma un valor diferente.

RTI: Esta instrucción forzá al procesador a retornar desde una sub-rutina de interrupción, esto lo hace cargando el valor que se encuentra en el registro RETI en el PC. Cuando se genera una interrupción, el procesador entra en un estado de no-interrupción, guarda el registro RETI en la pila y vuelve a habilitar la detección de interrupciones, así cuando se esta en la rutina de servicio a las interrupciones, el procesador atenderá de acuerdo a la prioridad de cada interrupción generada.

RTX: Esta instrucción sirve para forzar el procesador a retornar desde una excepción, esto lo hace cargando el valor que se encuentra en el registro RETX en el PC.

RTN: Esta instrucción sirve para forzar el procesador a retornar desde una interrupción no-enmascarable (NMI, por sus siglas en ingles), esto lo hace cargando el valor que se encuentra en el registro RETN en el PC.

RTE: Esta instrucción sirve para forzar el procesador a retornar desde una subrutina de emulación y el modo emulación, esto lo hace cargando el valor que se encuentra en el registro RETE en el PC.

RTS, *RTI*, *RTX*, *RTN*, *RTE*

```
// Retorno de una subrutina
RTS ;
// Retorno de una interrupcion
RTI ;
// Retorno de una excepcion
RTX ;
// Retorno de una NMI
RTN ;
// Retorno de una Emulacion
RTE ;
```

Bucle: Las instrucciones de configuración del Bucle proveen un contador flexible por medio de un mecanismo de hardware. La arquitectura incluye dos juegos de tres registros, Los registros son Loop_Top (LTn), Loop_Bottom (LBn) y Loop_Count (LCn). En consecuencia, LT0, LB0, y LC0 son para describir el loop0 y LT1, LB1 y LC1 son para describir el Loop1.

LSETUP, *LOOP*

```
/* Forma 1 */
/* Configura el inicio y fin del bucle ademas de el numero
   de veces que se repetira el ciclo, indicado por el LCn */
LSETUP(Etiqueta_Inicial,Etiqueta_Final) LC1=P4;
    Etiqueta_Inicial:R7 = B[P5](Z);   // Alguna Instruccion
        P5=P5+P3;         // Alguna Instruccion
    Etiqueta_Final:A0+=R7.L*R1.L(FU); // Alguna Instruccion

/* Forma 2 */
/* Configura el inicio y fin del bucle ademas de el numero
   de veces que se repetira el ciclo, indicado por el LCn */
LOOP Etiqueta_Usuario LC1=P4;
LOOP_BEGIN Etiqueta_Usuario
    R7 = B[P5](Z);                    // Alguna Instruccion
LOOP_END Etiqueta_Usuario
```

.6.5. Control de la pila

Empujar un registro en la pila: La instrucción *Push* almacena el contenido de un registro especificado en la pila. La instrucción pre-disminuye el puntero de pila a la siguiente posición disponible como primera posición en la pila. Empujar un valor y empujar múltiples valores son las únicas instrucciones que realizan funciones de pre-modificación. La pila crece hacia abajo, desde la parte alta de la memoria hasta la parte baja de la memoria. El puntero de pila apunta siempre a la última ubicación utilizada. Por lo tanto, la próxima dirección efectiva es SP-4. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$[--SP] = fuente;$

```
/* Disminuye la direccion a la que apunta SP
   en 4-bytes y guarda algun registro */
[--SP] = R0;
/* Disminuye la direccion a la que apunta SP
   en 4-bytes y guarda R1 */
[--SP] = R1;
/* Disminuye la direccion a la que apunta SP
   en 4-bytes y guarda P0 */
[--SP] = P0;
/* Disminuye la direccion a la que apunta SP
   en 4-bytes y guarda I0 */
[--SP] = I0;
```

Empujar múltiples registros en la pila: La instrucción *PushMultiple* almacena el contenido de múltiples registro-D o múltiples registros-P en la pila. La instrucción siempre incluye el registro de índice más alto (R7 o P5), además de un índice inferior contiguo de registros especificados por el usuario de forma descendente hasta incluir R0 y P0, a pesar de esto en la pila se guardan primero los registros con indices más bajos. Esta instrucción pre-disminuye el puntero de pila a la siguiente posición disponible como primera posición en la pila. Empujar un valor

y empujar múltiples valores son las únicas instrucciones que realizan funciones de pre-modificación. La pila crece hacia abajo, desde la parte alta de la memoria hasta la parte baja de la memoria. El puntero de pila apunta siempre a la última ubicación utilizada. Por lo tanto, la próxima dirección efectiva es SP-4. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$[--SP] = (fuentes);$

```
/* Empuja multiples registros en la pila */
[--SP] = (R7:5,P5:0);
/* Empuja multiples registros en la pila */
[--SP] = (R7:2);
/* Empuja multiples registros en la pila */
[--SP] = (P5:4);
```

Recuperar un dato de la pila y cargalo en un registro: La instrucción *Pop* carga el contenido de la pila en un registro especificado. La instrucción post-incrementa el puntero de pila a la siguiente posición efectiva. La pila crece hacia abajo, desde la parte alta de la memoria hasta la parte baja de la memoria, por supuesto, la intención habitual de Pop es de cargar o recuperar los valores de registros que fueron empujados previamente en la pila, por lo que el puntero de pila apunta a la próxima dirección efectiva, ya que cuando la instrucción se uso inicialmente el SP cambio a SP+4. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$reg\_dest = [SP++];$

```
/* Carga en un registro el dato de la pila al que
   apunta SP y post-incrementa en 4-bytes el SP */
R0 = [SP++];
/* Carga en P4 el dato de la pila al que apunta
     SP y post-incrementa en 4-bytes el SP */
P4 = [SP++];
/* Carga en I1 el dato de la pila al que apunta
     SP y post-incrementa en 4-bytes el SP */
I1 = [SP++];
/* Carga en RETI el dato de la pila al que apunta
     SP y post-incrementa en 4-bytes el SP */
RETI = [SP++];
```

Recuperar varios datos de la pila y cargalos en varios registros: La instrucción *PopMultiple* carga el contenido de la pila en múltiples registros. La instrucción siempre incluye el registro de índice más alto (R7 o P5), además de un índice inferior contiguo de registros especificados por el usuario de forma descendente hasta incluir R0 y P0, conforme a esto se cargan primero los registros con indices más altos hasta cargar los más bajos. Los registros-P se extraen antes que los registros-D, si se especifican ambas en la misma instrucción. La pila crece hacia abajo, desde la parte alta de la memoria hasta la parte baja de la memoria, por su-

puesto, la intención habitual de Pop es de cargar o recuperar los valores de registros que fueron empujados previamente en la pila, por lo que el puntero de pila apunta a la próxima dirección efectiva, ya que cuando la instrucción se uso inicialmente el SP cambio a SP+4. El direccionamiento indirecto y offset deben ser múltiplos de 4 para mantener la alineación a 4-byte, si no se mantiene la alineación adecuada se produce una excepción de acceso a la memoria.

$(regs_dest) = [SP++];$

```
/* Recupera multiples datos de la pila */
(P5:4) = [SP++];
/* Recupera multiples datos de la pila */
(R7:2) = [SP++];
/* Recupera multiples datos de la pila */
(R7:5,P5:0) = [SP++];
```

.6.6. Gestión de control de código de bit

Comparación de registros de datos: La instrucción de comparación de registros D establece CC a 1 si y solo si la base de comparación es cierta. Los operandos de entrada son registros-D. La operación de comparación es no invasiva, lo que quiere decir que los operandos de entrada no se alteran y solo se afecta el CC y las banderas. El valor del bit de CC determina todas las posteriores bifurcaciones condicionales.

$CC = Dreg\ (Oper)\ Dreg; Oper\ (<=, ==, <)$

```
/* Si R3 es igual que R2 entonces CC=1 si no CC=0 */
CC = R3 == R2;
/* Si R7 es igual que 1 entonces CC=1 si no CC=0 */
CC = R7 == 1;
/* Si R0 es menor que R3 entonces CC=1 si no CC=0 */
CC = R0 < R3;
/* Si R2 es menor que -4 entonces CC=1 si no CC=0 */
CC = R2 < -4;
/* Si R6 es menor o igual que R1 entonces CC=1 si no CC=0 */
CC = R6 <= R1;
/* Si R4 es menor o igual que 3 entonces CC=1 si no CC=0 */
CC = R4 <= 3;
/* Si R0 es menor que R3 entonces CC=0 si no CC=1, los
   valores que se comparan son sin signo */
CC = R0 < R3 (IU);
/* Si R1 es menor que 0x7 entonces CC=0 si no CC=1, los
   valores que se comparan son sin signo */
CC = R1 < 0x7 (IU);
/* Si R2 es menor o igual que R0 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = R2 <= R0 (IU);
/* Si R3 es menor o igual que 2 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = R3 <= 2 (IU);
```

Comparación de registros punteros: La instrucción de comparación de registros P establece CC a 1 si y solo si la base de comparación es cierta. Los operandos de entrada son registros-P. La operación de comparación es no invasiva, lo que quiere decir que los operandos de entrada no se alteran y solo se afecta el CC y las banderas. El valor del bit de CC determina todas las posteriores bifurcaciones condicionales.

$CC = Preg\ (Oper)\ Preg; Oper\ (<=, ==, <)$

```
/* Si P3 es igual que P2 entonces CC=1 si no CC=0 */
CC = P3 == P2;
/* Si P0 es igual que 1 entonces CC=1 si no CC=0 */
CC = P0 == 1;
/* Si P0 es menor que P3 entonces CC=1 si no CC=0 */
CC = P0 < P3;
/* Si P2 es menor que -4 entonces CC=1 si no CC=0 */
CC = P2 < -4;
/* Si P1 es menor o igual que P0 entonces CC=1 si no CC=0 */
CC = P1 <= P0;
/* Si P4 es menor o igual que 3 entonces CC=1 si no CC=0 */
CC = P4 <= 3;
/* Si P5 es menor que P3 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = P5 < P3 (IU);
/* Si P1 es menor que 0x7 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = P1 < 0x7 (IU);
/* Si P2 es menor o igual que P0 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = P2 <= P0 (IU);
/* Si P3 es menor o igual que 2 entonces CC=0 si no CC=1,
   los valores que se comparan son sin signo */
CC = P3 <= 2 (IU);
```

Comparación de registros Acumuladores: La instrucción de comparación de registros A establece CC a 1 si y solo si la base de comparación es cierta. Los operandos de entrada son registros-A. La operación de comparación es no invasiva, lo que quiere decir que los operandos de entrada no se alteran y solo se afecta el CC y las banderas. El valor del bit de CC determina todas las posteriores bifurcaciones condicionales.

$CC = Areg\ (Oper)\ Areg; Oper\ (<=, ==, <)$

```
/* Si A0 ==A1 entonces CC=1 si no CC=0 */
CC = A0 == A1;
/* Si A0 <A1 entonces CC=1 si no CC=0 */
CC = A0 < A1;
/* Si A0 <=A1 entonces CC=1 si no CC=0 */
CC = A0 <= A1;
```

Mover CC: Esta instrucción carga desde o hacia cc, un dato desde o hacia las banderas o los registros-D. El valor de CC cuando se carga a un registro de 32 bits se posiciona en el bit menos significativo y aplica cero extendido al resto de dicho

registro. El registro de datos cuando se carga en cc, si el registro de datos es igual a 0, entonces cc es igual a cero, con cualquier otro dato que se encuentre en el registro-D cc sera igual a 1.

destino (*Oper*) *fuente*; *Oper* (<=, =, |=, & =, ^ =)

```
/* Carga en el primer bit de un registro-D el valor
   de CC, y extiende a ceros el resto del registro-D */
R0 = CC ;
/* Carga en la bandera AZ el valor de CC */
AZ = CC ;
/* Carga en la bandera AN el valor de la
   operacion OR entre AN y CC */
AN |= CC ;
/* Carga en la bandera AC0 el valor de la
   operacion AND entre AC0 y CC */
AC0 &= CC ;
/* Carga en la bandera AV0 el valor de la
   operacion XOR entre AV0 y CC */
AV0 ^= CC ;
/* Si el registro-D es igual a cero entonces CC toma el
   valor de cero, cualquier otro valor en el registro-D
     configurara a CC como 1*/
CC = R4 ;
/* Carga en CC el valor de la bandera AV1 */
CC = AV1 ;
/* Carga en la bandera CC el valor de la
   operacion OR entre CC y AQ */
CC |= AQ ;
/* Carga en la bandera CC el valor de la
   operacion AND entre CC y AN */
CC &= AN ;
/* Carga en la bandera CC el valor de la
   operacion XOR entre CC y AC1 */
CC ^= AC1 ;
```

Negar CC: La instrucción negar CC, invierte el estado lógico de CC.

$CC = !CC$;

```
/* CC es igual a CC negado*/
cc =! cc;
```

.6.7. Operaciones lógicas

Operación lógica (AND): Esta instrucción realiza la operación lógica *AND* bit a bit de dos registros origen de 32-bits y almacena el resultado en un registro destino. Los registros fuente y destino son registros-D, El fuente1 y fuente2 pueden ser los mismos registro-D.

$Dreg = Dreg \ \& \ Dreg$;

```
/* En R4 se guarda el resultado obtenido de la operación
lógica AND entre R4 y R3  */
R4 = R4 & R3;
```

Operación lógica de complemento a uno: Esta instrucción realiza la operación lógica NOT bit a bit de un registro origen de 32-bits y almacena el resultado en un registro destino de 32-bits. El registro fuente y destino son registros-D, El fuente y el destino pueden ser los mismos registro-D.

$Dreg = \sim Dreg$

```
/* En R3 se guarda el valor negado de R4, resultado
obtenido con la operación lógica NOT*/
R3 = ~ R4;
```

Operación lógica (OR) : Esta instrucción realiza la operación lógica OR bit a bit de dos registros origen de 32-bits y almacena el resultado en un registro destino. Los registros fuente y destino son registros-D, El fuente1 y fuente2 pueden ser los mismos registro-D.

$Dreg = Dreg \mid Dreg;$

```
/* En R4 se guarda el resultado obtenido de la operación
lógica OR entre R4 y R3  */
R4 = R4 | R3;
```

Operación lógica XOR: Esta instrucción realiza la operación lógica XOR bit a bit de dos registros origen de 32-bits y almacena el resultado en un registro destino. Los registros fuente y destino son registros-D, El fuente1 y fuente2 pueden ser los mismos registro-D.

$Dreg = Dreg \wedge Dreg;$

```
/* En R4 se guarda el resultado obtenido de la operación
lógica XOR entre R4 y R3 */
R4 = R4 ^ R3;
```

.6.8. Operaciones de bits

BITCLR La instrucción de limpiar un bit, lleva a cero un bit especifico en un registro especifico, esta instrucción no afecta a ningún otro bit de dicho registro. El rango de valores de la posición del bit a limpiar va desde el 0 hasta el 31, donde 0 indica el LSB, y 31 indica el MSB del registro-D de 32 bits.

$BITCLR(registro, posicion\_de\_bit);$

```
/* Limpia el tercer bit de R2 */
BITCLR(R2,3);
```

BITSET La instrucción de setear un bit, lleva a uno un bit especifico en un registro especifico, esta instrucción no afecta a ningún otro bit de dicho registro. El rango de valores de la posición del bit setear va desde el 0 hasta el 31, donde 0 indica el LSB, y 31 indica el MSB del registro-D de 32 bits.

$BITSET(registro, posicion\_de\_bit)$;

```
/* Configura el bit 7 de R2 a 1 */
BITSET (R2,7);
```

BITTGL La instrucción de conmutar un bit, invierte el valor de un bit especifico en un registro especifico, esta instrucción no afecta a ningún otro bit de dicho registro. El rango de valores de la posición del bit a invertir va desde el 0 hasta el 31, donde 0 indica el LSB, y 31 indica el MSB del registro-D de 32 bits.

$BITTGL(registro, posicion\_de\_bit)$;

```
/* Invierte el valor del bit 24 del registro R2 */
BITTGL (R2, 24) ;
```

BITTST: La instrucción de testear un bit, Lleva a cero o a uno el CC dependiendo del valor de un bit especifico en un registro especifico, esta instrucción no afecta a ningún bit de dicho registro. El rango de valores de la posición del bit a invertir va desde el 0 hasta el 31, donde 0 indica el LSB, y 31 indica el MSB del registro-D de 32 bits.

$CC\ (Oper)\ BITTST(registro, posicion\_de\_bit)$; $Oper\ (=, =!)$

```
/* Testea el bit 15 de R5 si es igual a 1 entonces CC=1 */
CC = BITTST (R7, 15) ;
/* Testea el bit 0 de R3 si es igual a 0 entonces CC=1 */
CC = !BITTST (R3, 0) ;
```

ONES La instrucción de conteo de unos, carga el número de unos contenido en un registro fuente en la mitad menos significativa de un registro destino. Los registros fuente y destino son registros-D, El fuente1 y fuente2 pueden ser los mismos registro-D.

$Dreg\_l = ONES\ Dreg$;

```
/* Carga en los 16 LSB de R3 la cantidad
   de 1 que contiene el registro R7 */
R3.L = ONES R7 ;
```

.6.9. Operaciones de rotación y corrimiento

Adición con corrimiento: Esta instrucción combina la operación de adición con corrimiento lógico con uno o dos desplazamientos a la izquierda. Si se reali-

za un desplazamiento a la izquierda se obtiene una multiplicación por dos y si se realizan dos desplazamientos se obtiene una multiplicación por cuatro. La instrucción soporta tanto registros-D como registros-P, pero se debe realizar con un solo tipo de registros esta operación, la versión de registros-P no afecta ninguna bandera.

$Reg = (Reg + Reg) << 1;$
$Reg = (Reg + Reg) << 2;$

```
/* En P3 se guarda dos veces la suma de P3 y P2;
     esta operacion es equivalente a: P3 = (P3 + P2) * 2 */
P3 = (P3+P2)<<1 ;
/* En P3 se guarda cuatro veces la suma de P3 y P2;
     esta operacion es equivalente a: P3 = (P3 + P2) * 4 */
P3 = (P3+P2)<<2 ; /* P3 = (P3 + P2) * 4 */
/* En R3 se guarda dos veces la suma de R3 y R2;
     esta operacion es equivalente a: R3 = (R3 + R2) * 2 */
R3 = (R3+R2)<<1 ; /* R3 = (R3 + R2) * 2 */
/* En R3 se guarda cuatro veces la suma de R3 y R2;
     esta operacion es equivalente a: R3 = (R3 + R2) * 4 */
R3 = (R3+R2)<<2 ; /* R3 = (R3 + R2) * 4 */
```

Corrimiento con Suma: Esta instrucción combina la operación de corrimiento lógico con uno o dos desplazamientos a la izquierda con la suma. Si se realiza un desplazamiento a la izquierda se obtiene una multiplicación por dos y si se realizan dos desplazamientos se obtiene una multiplicación por cuatro. La instrucción soporta solo registros-P.

$Preg = Preg + (Preg << 1);$
$Preg = Preg + (Preg << 2);$

```
/* En P3 se guarda la suma de P0 con dos veces el valor de P3;
     esta operacion es equivalente a: P3 = P0 + (P3 * 2) */
P3 = P0+(P3<<1) ;
/* En P3 se guarda la suma de P0 con cuatro veces el valor de P3;
     esta operacion es equivalente a: P3 = P0 + (P3 * 4) */
P3 = P0+(P3<<2) ; /* P3 = (P3 * 4) + P0 */
```

Corrimiento aritmético: Esta instrucción realiza corrimiento de un registro en una dirección y distancia especificada, mientras preserva el signo del registro original. Un corrimiento a la izquierda satura el resultado si el dato se aleja demasiado y no garantiza la preservación del signo. La sintaxis ">>>=" solo se soporta a la derecha la instrucción es de una longitud máxima de 16-bits, La sintaxis ">>>" y "<<", y "ASHIFT" soporta corrimiento a la derecha e izquierda, son de 32-bits, los registros fuente y destino pueden ser diferentes y se pueden realizar operaciones en paralelo como las de cargar y guardar. Ambas sintaxis soportan inmediatos y registros para indicar la magnitud del corrimiento, para la versión "ASHIFT" la magnitud del corrimiento indica también la dirección, lo que quiere decir que una magnitud positiva sera un corrimiento a la izquierda y una magnitud negativa sera un corrimiento a la derecha.

$reg\_dest >>>= magnitud;$
$reg\_dest = reg\_fuente >>> magnitud(opc);$
$reg\_dest = reg\_fuente << magnitud(opc);$
$Acumulador = Acumulador >>> magnitud;$
$reg\_dest = ASHIFT\ reg\_fuente\ BY\ magnitud(opc);$
$Acumulador = ASHIFT\ Acumulador\ BY\ magnitud;$

```
/* Realiza 19 corrimientos a la derecha en R0
   y se guarda en R0 */
R0 >>>= 19;
/* Realiza 7 corrimientos a la derecha en R0.L
   y se guarda en R3.L */
R3.L = R0.H >>> 7 ;
/* Realiza 5 corrimientos a la derecha en R0.H
   y se guarda en R3.H */
R3.H = R0.H >>> 5 ;
/* Realiza 7 corrimientos a la derecha en R0.H
   y se guarda en R3.L y satura el resultado */
R3.L = R0.H >>> 7(S) ;
/* Realiza 20 corrimientos a la derecha en R2
   y se guarda en R4 */
R4 = R2 >>> 20 ;
/* Realiza 1 corrimiento a la derecha en A0
   y se guarda en A0 */
A0 = A0 >>> 1 ;
/* Realiza R2 corrimientos a la derecha en R0
   y se guarda en R0 */
R0 >>>= R2 ;
/* Realiza 12 corrimientos a la izquierda en R0.H
   y se guarda en R3.L */
R3.L = R0.H << 12 (S) ;
/* Realiza 24 corrimientos a la izquierda en R2
   y se guarda en R3.L y satura el resultado */
R5 = R2 << 24(S) ;
/* R7.L indica la direccion y numero de corrimientos
   de R0.H para ser guardados en R3.L */
R3.L = ASHIFT R0.H BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.H */
R3.H = ASHIFT R0.L BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de R0.H para ser guardados en R3.H */
R3.H = ASHIFT R0.H BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.L */
R3.L = ASHIFT R0.L BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de R0.H para ser guardados en R3.L y satura el resultado */
R3.L = ASHIFT R0.H BY R7.L(S);
/* R7.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.H y satura el resultado */
R3.H = ASHIFT R0.L BY R7.L(S);
/* R7.L indica la direccion y numero de corrimientos
   de R0.H para ser guardados en R3.H y satura el resultado */
```

```
R3.H = ASHIFT R0.H BY R7.L(S);
/* R7.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.L y satura el resultado */
R3.L = ASHIFT R0.L BY R7.L (S);
/* R7.L indica la direccion y numero de corrimientos
   de R2 para ser guardados en R4 */
R4   = ASHIFT R2 BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de R2 para ser guardados en R4 y satura el resultado */
R4   = ASHIFT R2 BY R7.L (S);
/* R7.L indica la direccion y numero de corrimientos
   de A0 para ser guardados en A0 */
A0   = ASHIFT A0 BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de A1 para ser guardados en A1 */
A1   = ASHIFT A1 BY R7.L;
```

Corrimiento Logico: Esta instrucción realiza corrimiento de bits en un registro en una dirección y distancia especificada. La sintaxis ">>=" y "<<=" soporta a la derecha y a la izquierda una instrucción de una longitud máxima de 16-bits, La sintaxis ">>" y "<<", y "LSHIFT" soporta corrimiento a la derecha e izquierda, son de 32-bits, los registros fuente y destino pueden ser diferentes y se pueden realizar operaciones en paralelo como las de cargar y guardar. Ambas sintaxis soportan inmediatos y registros para indicar la magnitud del corrimiento, para la versión "ASHIFT" la magnitud del corrimiento indica también la dirección, lo que quiere decir que una magnitud positiva sera un corrimiento a la izquierda y una magnitud negativa sera un corrimiento a la derecha.

$Preg = Preg >> 1$
$Preg = Preg >> 2$
$Preg = Preg << 1$
$Preg = Preg << 2$
$reg\_dest >>= magnitud$;
$reg\_dest <<= magnitud$;
$reg\_dest = reg\_fuente >> magnitud$;
$reg\_dest = reg\_fuente << magnitud$;
$reg\_dest = LSHIFT\ reg\_fuente\ BY\ magnitud$;

```
/* Se guarda en P3 la mitad del dato que hay en P2 */
P3 = P2 >> 1;
/* Se guarda en P3 un cuarto del dato que hay en P3 */
P3 = P3 >> 2;
/* Se guarda en P4 dos veces el dato que hay en P5 */
P4 = P5 << 1;
/* Se guarda en P0 cuatro veces el dato que hay en P1 */
P0 = P1 << 2;
/* Se realizan 17 corrimientos a la derecha en R3
   y se guarda el resultado en R3 */
R3 >>= 17;
/* Se realizan 17 corrimientos a la izquierda en R3
   y se guarda el resultado en R3 */
```

```
R3 <<= 17;
/* Se guarda en R3.L el dato que hay en R0.L con 4
   corrimientos a la derecha */
R3.L = R0.L >> 4;
/* Se guarda en R3.L el dato que hay en R0.H con 4
   corrimientos a la derecha */
R3.L = R0.H >> 4;
/* Se guarda en R3.H el dato que hay en R0.L con 12
   corrimientos a la izquierda */
R3.H = R0.L << 12;
/* Se guarda en R3.H el dato que hay en R0.H con 14
   corrimientos a la izquierda */
R3.H = R0.H << 14;
/* Se guarda en R3 el dato que hay en R6 con 4
   corrimientos a la derecha */
R3 = R6 >> 4;
/* Se guarda en R3 el dato que hay en R6 con 4
   corrimientos a la izquierda */
R3 = R6 << 4;
/* Se guarda en A0 el dato que hay en A0 con 7
   corrimientos a la derecha */
A0 = A0 >> 7;
/* Se guarda en A1 el dato que hay en A1 con 25
  corrimientos a la derecha */
A1 = A1 >> 25;
/* Se guarda en A0 el dato que hay en A0 con 7
   corrimientos a la izquierda */
A0 = A0 << 7;
/* Se guarda en A1 el dato que hay en A1 con 14
   corrimientos a la izquierda */
A1 = A1 << 14;
/* Se guarda en R3 el dato que hay en R3 con R0
  corrimientos a la derecha */
R3 >>= R0;
/* Se guarda en R3 el dato que hay en R3 con R1
   corrimientos a la izquierda */
R3 <<= R1;
/* R2.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.L */
R3.L = LSHIFT R0.L BY R2.L;
/* R2.L indica la direccion y numero de corrimientos
   de R0.L para ser guardados en R3.H */
R3.H = LSHIFT R0.L BY R2.L;
/* R7.L indica la direccion y numero de corrimientos
   de A0 para ser guardados en A0 */
A0 = LSHIFT A0 BY R7.L;
/* R7.L indica la direccion y numero de corrimientos
   de A1 para ser guardados en A1 */
A1 = LSHIFT A1 BY R7.L;
```

Rotación: Esta instrucción gira un registro a través del bit cc, es decir se forma un registro de 33 bits, y a medida que se realizan las rotaciones cada bit que gira fuera del registro (el LSB para rotación a la derecha o la MSB para rotación izquierda) se almacena en el bit de cc, y el bit que se encuentra en cc quien tiene un valor n inicial desconocido entra a formar parte del registro ocupando el bit que se

encuentra vacante debido a la rotación en el extremo opuesto del registro.

reg_dest = ROT reg_fuente BY magnitud;
Acumulador = ROT Acumulador BY magnitud;

```
/* Rota 8 veces a la izquierda el registro R1 y guarda en R4 */
R4 = ROT R1 BY 8 ;
/* Rota 5 veces a la derecha el registro R1 y guarda en R4 */
R4 = ROT R1 BY  -5 ;
/* Rota 22 veces a la izquierda el acumulador A0 y guarda en A0 */
A0 = ROT A0 BY  22 ;
/* Rota 31 veces a la derecha el registro A1 y guarda en A1 */
A1 = ROT A1 BY  -31 ;
/* R2.L contiene la direccion y la cantidad de rotaciones que
   realizara el registro R1 y guarda en R4 */
R4 = ROT R1 BY  R2.L ;
/* R3.L contiene la direccion y la cantidad de rotaciones que
   realizara el acumulador A0 y guarda en A0 */
A0 = ROT A0 BY  R3.L ;
/* R7.L contiene la direccion y la cantidad de rotaciones que
   realizara el acumulador A1 y guarda en A1 */
A1 = ROT A1 BY  R7.L ;
```

.6.10. Operaciones aritméticas

ABS: La instrucción de valor absoluto, calcula el valor absoluto de un registro de 32-bits y guarda este dentro de un registro destino de 32 bits acorde a las siguientes reglas: si el valor de entrada is positivo o cero este se copia en el registro destino sin modificarse, pero si el valor de entrada es negativo entonces el resultado es la resta entre cero y el valor. La operación ABS puede también realizarse en ambos acumuladores con una simple instrucción.

reg_dest = ABS reg_fuente;

```
/* Carga en A0 el valor absoluto de A0 */
A0 = ABS A0 ;
/* Carga en A0 el valor absoluto de A1 */
A0 = ABS A1 ;
/* Carga en A1 el valor absoluto de A0 */
A1 = ABS A0 ;
/* Carga en A1 el valor absoluto de A1 */
A1 = ABS A1 ;
/* Carga en A1 el valor absoluto de A1 en paralelo
   carga en A0 el valor absoluto de A0 */
A1 = ABS A1, A0= ABS A0 ;
/* Carga en R3 el valor absoluto de R1 */
R3 = ABS R1 ;
```

ADD: La instrucción de suma realiza la suma entre dos registros fuente y almacena el resultado en un registro destino. existen dos caminos para realizar una suma de datos de 32-bits en registros-D. la primera es sumar dos medios registros es

decir 16-bits y no soporta saturación, la otra forma es sumar dos registros de 32-bits la cual soporta saturación. la instrucción también permite realizar sumas entre dos registros-P.

$reg\_dest = reg\_fuente + reg\_fuente;$

```
/* En R5 se guarda el resultado de la suma entre
   R2 y R1 */
R5 = R2 + R1 ;
/* En R5 se guarda el resultado de la suma entre
   R2 y R1 con saturacion*/
R5 = R2 + R1(s) ;
/* En P5 se guarda el resultado de la suma entre
   P2 y P0 */
P5 = P3 + P0 ;
/* En R4.L se guarda el resultado de la suma entre
   R0.L y R7.L sin saturacion */
R4.L = R0.L + R7.L (ns) ;
/* En R4.L se guarda el resultado de la suma entre
   R0.L y R7.H con saturacion */
R4.L = R0.L + R7.H (s) ;
/* En R0.L se guarda el resultado de la suma entre
   R2.H y R4.L sin saturacion */
R0.L = R2.H + R4.L (NS) ;
/* En R1.L se guarda el resultado de la suma entre
   R3.H y R7.H sin saturacion */
R1.L = R3.H + R7.H (NS) ;
/* En R4.H se guarda el resultado de la suma entre
   R0.L y R7.L sin saturacion */
R4.H = R0.L + R7.L (NS) ;
/* En R4.H se guarda el resultado de la suma entre
   R0.L y R7.H sin saturacion */
R4.H = R0.L + R7.H (NS) ;
/* En R0.H se guarda el resultado de la suma entre
   R2.H y R4.L con saturacion */
R0.H = R2.H + R4.L (N) ;
/* En R1.H se guarda el resultado de la suma entre
   R3.H y R7.H sin saturacion */
R1.H = R3.H + R7.H (NS) ;
```

Suma y resta con prescale hacia abajo: Esta instrucción permite combinar dos valores de 32-bits para producir un resultado de 16-bits. Esta instrucción realiza un corrimiento de 4-bits a la derecha de ambos operandos, realiza la operación deseada (suma o resta), y por último redondea el resultado a 16-bits

$reg\_dest = reg\_fuente + reg\_fuente\ (RND20);$
$reg\_dest = reg\_fuente + reg\_fuente\ (RND20);$

```
/* Se suman R6 y R7 despues de realizarles 4 corrimientos a la
   derecha y guarda el resultado redondeando a 16 bits en R1.L */
R1.L = R6+R7(RND20) ;
/* Se restan R6 y R7 despues de realizarles 4 corrimientos a la
```

```
   derecha y guarda el resultado redondeando a 16 bits en R1.L */
R1.L = R6-R7(RND20) ;
/* Se suman R6 y R7 despues de realizarles 4 corrimientos a la
   derecha y guarda el resultado redondeando a 16 bits en R1.H */
R1.H = R6+R7(RND20) ;
/* Se restan R6 y R7 despues de realizarles 4 corrimientos a la
   derecha y guarda el resultado redondeando a 16 bits en R1.H */
R1.H = R6-R7(RND20) ;
```

Suma y resta con prescale hacia arriba: Esta instrucción permite combinar dos valores de 32-bits para producir un resultado de 16-bits. Esta instrucción realiza un corrimiento de 4-bits a la izquierda de ambos operandos, realiza la operación deseada (suma o resta), y por último redondea el resultado a 16-bits

$reg\_dest = reg\_fuente + reg\_fuente\ (RND12);$
$reg\_dest = reg\_fuente + reg\_fuente\ (RND12);$

```
/* Se suman R6 y R7 despues de realizarles 4 corrimientos a la
   izquiera y guarda el resultado redondeando a 16 bits en R1.L */
R1.L = R6+R7(RND12) ;
/* Se restan R6 y R7 despues de realizarles 4 corrimientos a la
   izquiera y guarda el resultado redondeando a 16 bits en R1.L */
R1.L = R6-R7(RND12) ;
/* Se suman R6 y R7 despues de realizarles 4 corrimientos a la
   izquiera y guarda el resultado redondeando a 16 bits en R1.H */
R1.H = R6+R7(RND12) ;
/* Se restan R6 y R7 despues de realizarles 4 corrimientos a la
   izquiera y guarda el resultado redondeando a 16 bits en R1.H */
R1.H = R6-R7(RND12) ;
```

Sumar un inmediato: Esta instrucción permite adicionar una constante sin saturación, la instrucción soporta registros-D, registros-I y registros-P.

$registro += constante$

```
/* Se le suma a R0 la constante 40 y se guarda el resultado en R0 */
R0 += 40 ;
/* Se le suma a P5 la constante -4 y se guarda el resultado enP5 */
P5 += -4 ;
/* Se le suma a I0 la constante 2 y se guarda el resultado enI0 */
I0 += 2 ;
/* Se le suma a I1 la constante 4 y se guarda el resultado enI0 */
I1 += 4 ;
```

División primitiva: Esta instrucción realiza la operación de la división por medio del método de restas sucesivas. El dividendo o numerador es un valor de 32-bits y es quien determina el signo de la operación, y el divisor o denominador es un valor de 16-bits siempre positivo, este se encuentra ubicado en la parte baja de un registro, la parte alta de dicho registro es ignorada totalmente.

$DIVS\ (reg\_dividendo, reg\_divisor)$
$DIVQ\ (reg\_dividendo, reg\_divisor)$

```
/* Se carga en P0 el valor 15 para indicar que la
     division se realizara con enteros de 16 bits */
P0 = 15 ;
/* Se carga en R0 el valor del dividendo o numerador */
R0 = 70 ;
/* Se carga en R1 el valor del divisor o denominador */
R1 = 5 ;
/* Corrimiento a la izquierda por uno
   necesario para una division en enteros */
R0 <<= 1 ;
/* Se evalua el cociente MSB, se inicializa la
   bandera AQ para el bucle DIVQ */
DIVS (R0, R1) ;
/* Se evalua  DIVQ P0=15 veces */
LOOP .DIV_PRIM LC0=P0 ;
LOOP_BEGIN .DIV_PRIM;
DIVQ (R0, R1) ;
LOOP_END .DIV_PRIM ;
/* Se aplica signo extendido al resto del registro que
   contiene el cociente el cual es de 32 bits*/
R0 = R0.L (X) ;
/* R0 contiene el cociente  (70/5 = 14). */
```

MAX: Esta instrucción retorna al registro destino el mayor valor positivo que se encuentra entre los registros fuente. La instrucción realiza una resta entre los dos registros fuente y selecciona la salida basándose en los signos de los datos de entrada y las banderas aritméticas. Esta operación se realiza entre registros-D

$$reg\_dest = MAX\ (reg\_fuente\ ,\ reg\_fuente)$$

```
/* Guarda en R5 el dato mayor entre R2 y R3 */
R5 = MAX (R2, R3) ;
```

MIN: Esta instrucción retorna al registro destino el menor valor negativo que se encuentra entre los registros fuente. La instrucción realiza una resta entre los dos registros fuente y selecciona la salida basándose en los signos de los valores de entrada y las banderas aritméticas. Esta operación se realiza entre registros-D

$$reg\_dest = MIN\ (reg\_fuente\ ,\ reg\_fuente)$$

```
/* Guarda en R5 el valor menor entre R2 y R3 */
R5 = MIN (R2, R3) ;
```

Decrementa y modifica: La instrucción decrementa el registro destino por una cantidad definida por el usuario en el registro fuente.

$$reg\_dest - = reg\_fuente$$

```
/* Decrementa el valor en A0 por el valor indicado en A1 */
A0 -= A1 ;
/* Decrementa el valor en A0 por el valor indicado en A1
   y satura el resultado a 32-bits signo extendido*/
```

```
A0 -= A1 (W32) ;
/* Decrementa el valor en P3 por el valor indicado en P0 */
P3 -= P0 ;
/* Decrementa el valor en I1 por el valor indicado en M2 */
I1 -= M2 ;
```

Incrementa y modifica: La instrucción incrementa el registro destino por la cantidad definida por el usuario en el registro fuente, en algunas ocasiones la instrucción copia el resultado en un tercer registro.

$reg\_dest += reg\_fuente\ reg\_dest\_1 = (reg\_dest\_0 += reg\_fuente\_1)$

```
/* Incrementa el valor en A0 por el valor indicado en A1 */
A0 += A1 ;
/* Incrementa el valor en A0 por el valor indicado en A1
   y satura el resultado a 32-bits signo extendido*/
A0 += A1 (W32) ;
/* Incrementa el valor en P3 por el valor indicado en P0
   y bit de acarreo*/
P3 += P0 (BREV);
/* Incrementa el valor en I1 por el valor indicado en M1 */
I1 += M1 ;
/* Incrementa el valor en I0 por el valor indicado en M0
   y bit de acarreo*/
I0 += M0 (BREV) ;
/* Incrementa el valor en A0 por el valor indicado en A1
   y  guarda en R5 el resultado */
R5 = (A0 += A1) ;
/* Incrementa el valor en A0 por el valor indicado en A1
   y  guarda en R2.L el resultado */
R2.L = (A0 += A1) ;
/* Incrementa el valor en A0 por el valor indicado en A1
   y  guarda en R5.H el resultado */
R5.H = (A0 += A1) ;
```

Multiplicación de dos operandos de 16-bits: Esta instrucción multiplica dos operandos de 16-bits y guarda el resultado directamente en el registro destino con saturación. La instrucción soporta solo registros-D.

$reg\_dest = reg\_fuente\_0 * reg\_fuente\_1\ (opc)$

```
/* Multiplica R3.H por R2.H y guarda en R3.L */
R3.L=R3.H*R2.H ;
/* Multiplica R6.H por R4.L y guarda en R3.H */
R3.H=R6.H*R4.L (FU) ;
/* Multiplica R3.H por R4.H y guarda en R6 a 32-bits */
R6=R3.H*R4.H ;
```

Multiplicación de dos operandos de 32-bits: Esta instrucción multiplica dos registros de datos de 32-bits y guarda el producto directamente en el registro destino. Esta instrucción imita la multiplicación que se realiza en lenguaje c realizando la operación exactamente de la siguiente forma Dreg1 = (Dreg1 * Dreg2). Los desbordamiento son posibles pero no son detectables. La instrucción soporta solo

registros-D.

$reg\_dest *= reg\_multiplicador$

```
/* Multiplico R0 por R3 y guardo el resultado en R3 */
R3 *= R0 ;
```

Multiplicar dos registros de 16-bits con acumulación: Esta instrucción multiplica dos operandos de 16-bits, y guarda, suma o resta el producto en un acumulador. La instrucción soporta solo registros-D y registros-A.

$Acumulador = reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$Acumulador += reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$Acumulador -= reg\_fuente\_0 * reg\_fuente\_1\ (opc)$

```
/* Multiplicar la parte baja de R2 por la parte baja de R3
   y guardar en el acumulador A0 */
A0=R3.H*R2.H ;
/* Multiplicar la parte baja de R2 por la parte baja de R3
   y  acumula en A0 sin signo */
A1+=R6.H*R4.L (FU) ;
```

Multiplicar dos registros de 16-bits y acumula en un reg_dest_L_H Esta instrucción multiplica dos operandos de 16-bits. Guarda la suma o resta del producto en un acumulador designado con saturación a 16-bits para posteriormente ser copiado en un registro destino de 16-bits. La instrucción soporta solo registros-D y registros-A.

$reg\_dest\_L\_H = Acumulador = reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$reg\_dest\_L\_H = Acumulador += reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$reg\_dest\_L\_H = Acumulador -= reg\_fuente\_0 * reg\_fuente\_1\ (opc)$

```
/* Multiplica dos medios registros-D  guarda en un acumulador y
   copia el resultado en un tercer medio registro-D */
R3.L=(A0=R3.H*R2.H) ;
/* Multiplica dos medios registros-D  guarda en un acumulador y copia
   el resultado en un tercer medio registro-D con saturacion */
R3.H=(A1+=R6.H*R4.L) (FU) ;
```

Multiplicar dos registros de 16-bits y acumula en un registro-D Esta instrucción multiplica dos operandos de 16-bits. Guarda la suma o resta el producto en un acumulador designado con saturación a 32-bits para posteriormente ser copiado en un registro destino de 32-bits. La instrucción soporta solo registros-D y registros-A.

$reg\_dest = Acumulador = reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$reg\_dest = Acumulador += reg\_fuente\_0 * reg\_fuente\_1\ (opc)$
$reg\_dest = Acumulador -= reg\_fuente\_0 * reg\_fuente\_1\ (opc)$

```
/* Multiplica dos registros-D guarda en un acumulador y copia
   el resultado en un tercer registro-D */
R4=(A0=R3.H*R2.H);
 /* Multiplica dos registros-D guarda en un acumulador y copia
   el resultado en un tercer registro-D con saturacion*/
R3=(A1+=R6.H*R4.L) (fu)
```

Negar, complemento a dos Esta instrucción devuelve la misma magnitud del dato que contenga el registro fuente con el signo contrario. La versión para los acumuladores satura el resultado a 40 bits. La instrucción hace el cálculo restando el valor que contiene el registro fuente de cero. La instrucción soporta solo registros-D y registros-A.

$reg\_dest = -\ reg\_fuente$
$reg\_dest = -\ Acumulador\_fuente$

```
/* Se guarda en R5 el valor de R0 con el signo contrario */
R5 =-R0 ;
/* Se guarda en A0 el valor de A0 con el signo contrario */
A0 =-A0 ;
/* Se guarda en A0 el valor de A1 con el signo contrario */
A0 =-A1 ;
/* Se guarda en A1 el valor de A0 con el signo contrario */
A1 =-A0 ;
/* Se guarda en A1 el valor de A1 con el signo contrario */
A1 =-A1 ;
/* Se guarda en R0 el valor de R1 con el signo contrario
   con saturacion */
R0 =-R1(S) ;
/* Se guarda en R5 el valor de R0 con el signo contrario
   sin saturacion*/
R5 =-R0 (NS) ;
```

Redondeo para 16-bits Esta instrucción realiza un redondeo a 32-bits y normaliza el resultado en 16-bits y satura los bits del 31 - 16.

$reg\_dest = reg\_fuente\ (RND)$

```
/* Redondea a 16 bits el contenido de R6 y lo guarda en la
   parte baja de R1 */
R1.L = R6 (RND) ;
/* Redondea a 16 bits el contenido de R6 y lo guarda en la
   parte alta de R1 */
R1.H = R7 (RND) ;
```

Saturación Esta instrucción satura a 32 bits los registros-A y aplica signo extendido a los demás bits del registro-A.

$reg\_dest = reg\_fuente\ (S)$

```
/* Satura a 32-bits el contenido del acumulador A0 */
A0 = A0 (S) ;
```

```
/* Satura a 32-bits el contenido del acumulador A1 */
A1 = A1 (S) ;
```

Resta La instrucción resta al registro fuente-2 el registro fuente-1 y guarda el resultado en el registro destino
$reg\_dest = reg\_fuente\_1 - reg\_fuente\_2$

```
/* Se le resta a R2 el valor contenido en R1 y guarda
   el resultado en R5 sin saturacion */
R5 = R2 - R1 ;
/* Se le resta a R2 el valor contenido en R1 y guarda
   el resultado en R5 sin saturacion */
R5 = R2 - R1(NS) ;
/* Se le resta a R2 el valor contenido en R1 y guarda
   el resultado en R5 con saturacion */
R5 = R2 - R1(S) ;
/* Se le resta a R0.L el valor contenido en R7.L y guarda
   el resultado en R4.L sin saturacion */
R4.L = R0.L - R7.L (NS) ;
/* Se le resta a R0.L el valor contenido en R7.L y guarda
   el resultado en R4.H con saturacion */
R4.L = R0.L - R7.H (S) ;
/* Se le resta a R2.H el valor contenido en R4.L y guarda
   el resultado en R0.L sin saturacion */
R0.L = R2.H - R4.L(NS) ;
/* Se le resta a R3.H el valor contenido en R7.H y guarda
   el resultado en R1.L sin saturacion */
R1.L = R3.H - R7.H(NS) ;
/* Se le resta a R0.L el valor contenido en R7.L y guarda
   el resultado en R4.H sin saturacion */
R4.H = R0.L - R7.L (NS) ;
/* Se le resta a R0.L el valor contenido en R7.H y guarda
   el resultado en R4.H sin saturacion */
R4.H = R0.L - R7.H (NS) ;
/* Se le resta a R2.H el valor contenido en R4.L y guarda
   el resultado en R0.H con saturacion */
R0.H = R2.H - R4.L(S) ;
/* Se le resta a R3.H el valor contenido en R7.H y guarda
   el resultado en R1.H sin saturacion */
R1.H = R3.H - R7.H(NS)
```

Restar una constante La instrucción resta del registro fuente una constante y lo guarda en un registro destino sin saturación. El registro fuente y destino son los mismos.
$reg\_dest = reg\_fuente\_1 - reg\_fuente\_2$

```
/* Se le resta una constante a un registro-I */
I0 -= 4 ;
/* Se le resta la constante 2 al registro I2 */
I2 -= 2 ;
```

Algoritmo principal

.7. Código Matlab

```
clear all;
close all;
clc;
% Recibe imagen RGB %
Ima_Ent = imread('ImagenesTesis\OImaInC.jpg');

% Transforma imagen RGB a escala de grises %
Ima_Gris = rgb2gray(Ima_Ent);

tic;
% Filtrado de la imagen de entrada en escala de grises %
Ima_Sua = Suavizado(Ima_Gris,3,3);
eTime = toc;
fprintf('El tiempo Sua es: %5.5f Seg', eTime);

tic;
% Aplica primera fase de la Segmentacion %
Kbajo = 20;
Kalto = 0;
[Ima_Seg_Ini,Prom] = Segmentacion_Inicial(Ima_Sua,Kbajo,Kalto);
eTime = toc;
fprintf('El tiempo Seg_Ini es: %5.5f Seg', eTime);

tic;
% Aplica segunda fase de la Segmentacion a través de Sauvola %
Ima_Seg_Fin = Segmentacion_Final(Ima_Seg_Ini,5,5);
eTime = toc;
fprintf('El tiempo Seg_Final es: %5.5f Seg', eTime);

tic;
% Aplicar Conectividad %
[Vec] = Conectividad(Ima_Seg_Fin,Prom);
[Coordenadas,ConNumEst] = size(Vec);
eTime = toc;
fprintf('El tiempo Con es: %5.5f Seg', eTime);

tic;
% Aplicar Centroide %
[Filas,Col,Acu] = Centroide(Ima_Sua,Vec);
eTime = toc;
```

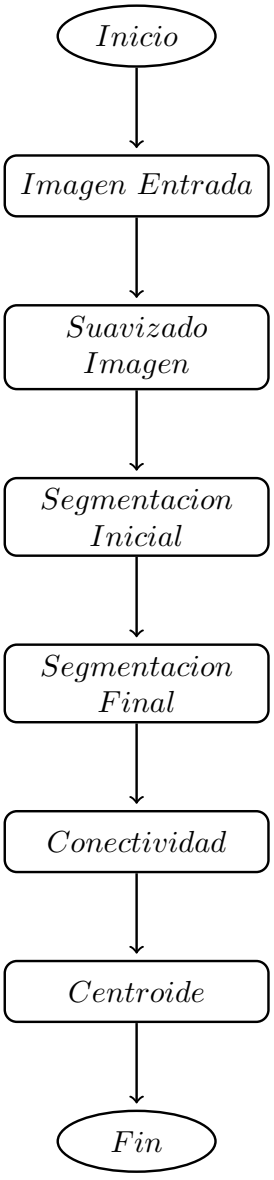
```
fprintf('El tiempo Cen es: %5.5f Seg', eTime);

% Calibracion del Centro %
W = EliminaFalsosPositivos(Ima_Sua,Filas,Col,ConNumEst,Vec);

for i=1:ConNumEst
    if W(1,i)==1
        W(9,i)=Prom(W(10,i),W(11,i));
    end
end

tic;
Ima_Cen = my_showcruz(Ima_Sua,Filas,Col,ConNumEst);
eTime = toc;
fprintf('El tiempo Mostrar Cruces es: %5.5f Seg', eTime);

tic;
Ima_Con = my_showcuadros(Ima_Sua,Vec,ConNumEst);
eTime = toc;
fprintf('El tiempo Mostrar Cuadros es: %5.5f Seg', eTime);

[I9,Conf,Desc,NivS1] = my_showestrellas(Ima_Sua,Vec,ConNumEst,W);

% figure(1)
% imshow(Ima_Gris);
% figure(2)
% imshow(Ima_Sua);
% figure(3)
% imshow(Ima_Seg_Ini);
% figure(4)
% imshow(Ima_Seg_Fin);
% figure(5)
% imshow(Ima_Con);
% figure(6)
% imshow(Ima_Cen);
% figure(7)
% imshow(Imagen9);
```

.8. Código en VisualDSP++

```
/*****************************************************************
 * DetEstrellas_VS_101.c
 * AUTORES: LEONARDO FABIO GARCIA RENDON
 *          ADOLFO ALEXANDER CARDONA CORREA
 * UNIVERSIDAD TECNOLOGICA DE PEREIRA
 * COLOMBIA
 *****************************************************************/

// Directivas del Procesador
#include<cdefBF533.h>
#include "variables.h"
#include <time.h>
#include <stdio.h>

// Funciones Externas
extern void Suavizado(void);
extern void Segmentacion_Inicial(void);
extern void Segmentacion_Final(void);
extern void Duplicado_Segmentacion_Final(void);
extern void Conectividad(void);
extern void Centroide(void);
extern void Im_Cruz(void);
extern void Im_Cuadro(void);

int main( void )
{
    volatile clock_t c_start;
    volatile clock_t c_stop;
    double secs;

    c_start = clock();
    Suavizado();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo del Suavizado es: %e s\n",secs);

    //c_start = clock();
    Segmentacion_Inicial();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de la SegIni es: %e s\n",secs);

    //c_start = clock();
    Segmentacion_Final();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de la SegFin es: %e s\n",secs);

    //c_start = clock();
```

```
    Duplicado_Segmentacion_Final();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de la DupSegFin es: %e s\n",secs);


    //c_start = clock();
    Conectividad();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de la Conectividad es: %e s\n",secs);

    //c_start = clock();
    Centroide();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo del Centroide es: %e s\n",secs);

    //c_start = clock();
    Im_Cruz();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de Im Cruz es: %e s\n",secs);

    //c_start = clock();
    Im_Cuadro();
    //c_stop = clock();

    //secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    //printf("El Tiempo de Im Cuadro es: %e s\n",secs);

    c_stop = clock();
    secs = ((double) (c_stop - c_start))/ CLOCKS_PER_SEC;
    printf("El Tiempo Total es: %e s\n",secs);

}
```

.9. Diagrama de flujo algoritmo principal



Algoritmo Suavizado

.10. Código Matlab

```
function ImaOut = Suavizado(ImaIn,Fil,Col)
Fil2 = (Fil-1)/2;
Col2 = (Col-1)/2;
ImagenW = ReplicaImagen(ImaIn,Fil2,Col2);
[m,n] = size(ImagenW);
W = zeros(m,n);

Var = Fil * Col;
W1 = zeros(1,Var);
D_I = double(ImagenW);

for r = 1+Fil2:m-Fil2
    for c = 1+Col2:n-Col2
        g = 1;
        for j = r-Fil2:r+Fil2
            for k = c-Col2:c+Col2
                W1(1,g) = D_I(j,k);
                g = g+1;
            end
        end
        W(r,c) = mean(W1);
    end
end

ImagenGris = InvReplicaImagen(W,Fil2,Col2);
ImaOut = uint8(ImagenGris);
end
```

.11. Código en VisualDSP++

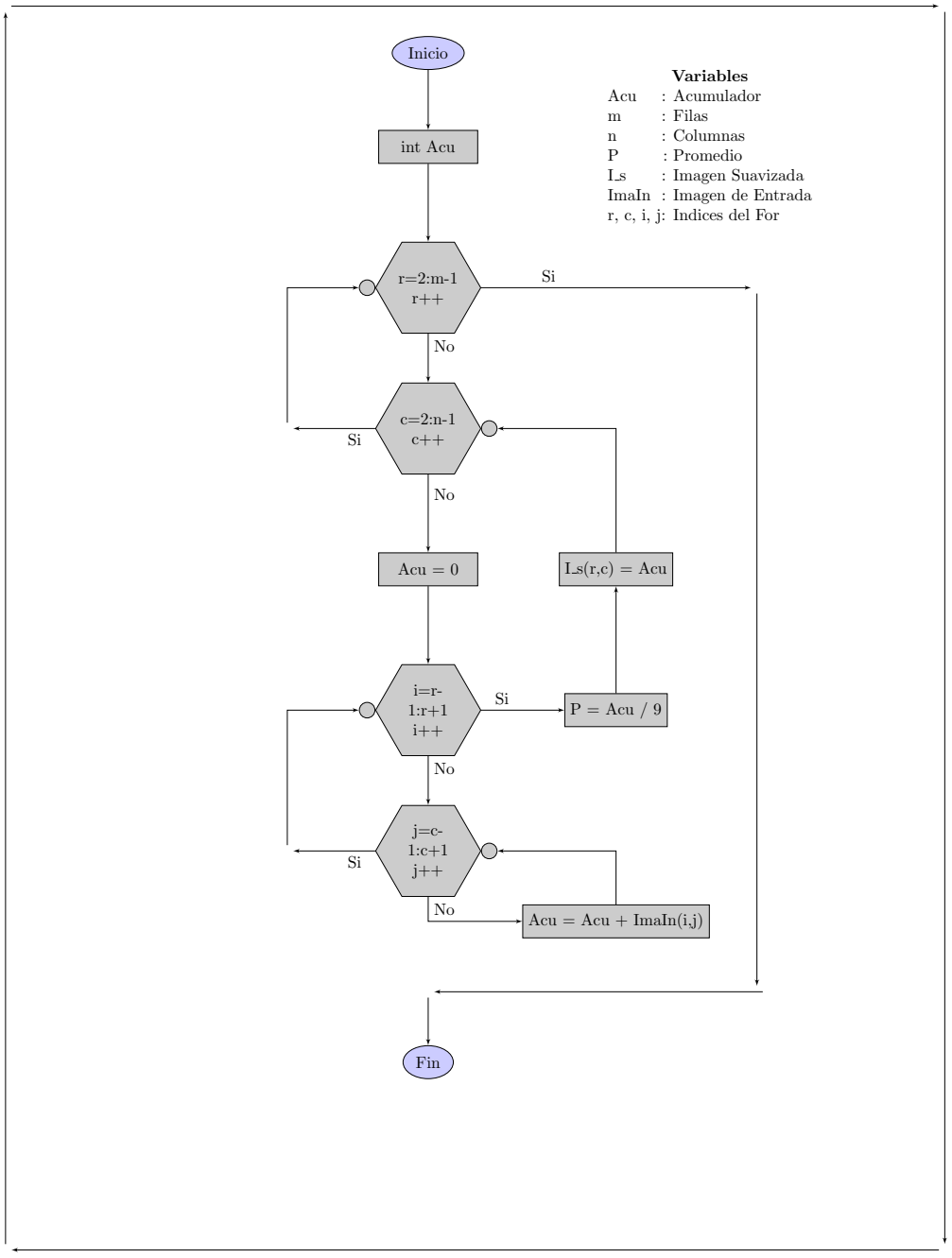
```
////////////////////////////////////////////////////////////////////////
/*                  ADSP-BF533 Filtro del Promedio                    */
////////////////////////////////////////////////////////////////////////

#include "variables.h"

// Comandos del preprocesador

//Seccion programa

.section L1_code;
.global _Suavizado;
.align 8;

// Etiqueta del assembler
_Suavizado:

////////////////////////////////////////////////////////////////////////
/*              Inicio: Inicialización de variables                   */
////////////////////////////////////////////////////////////////////////

    [--SP]=(R7:0,P5:0);
    SP+=-24;

    // Guarda dirección de inicio de la imagen de entrada en P1
    P1.L=LO(gray_direcc_inicio);
    P1.H=HI(gray_direcc_inicio);

    // Guarda dirección de inicio de la imagen de salida en P0
    P0.L=LO(Ima_filtro_direcc_inicio);
    P0.H=HI(Ima_filtro_direcc_inicio);

    // Guarda número de píxeles de la imagen en P2
    P2.L=LO(N);
    P2.H=HI(N);

    // Guarda componente del filtro: (0.111 * 32768)
    R7.L=0XE38;
    R7.H=0XE38;

    // R0 = 0
    R0=R7-R7(NS);
    // R4 = 0
    R4=R7-R7(NS);

    // Recupera el valor de filas
    P3.L=LO(filas);
    P3.H=HI(filas);

    // Recupera el valor de columnas
    P4.L=LO(columnas);
    P4.H=HI(columnas);
```

```
////////////////////////////////////////////////////////////////////////
/*                Fin: Inicialización de variables                    */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*        Primera Fila igual a la imagen de entrada                   */
////////////////////////////////////////////////////////////////////////

    // Ciclo igual al número de columnas
    LSETUP(Primera_Fila_Ini,Primera_Fila_Fin)LC0=P4;

        // Toma el Byte al que apunta P1, lo estiende a ceros
        // y lo carga en R6 y aumenta P1;
        Primera_Fila_Ini:R6=B[P1++](Z);

        // Toma el primer Byte de R6 y lo carga en la posición que
        // apunta P0 y lo aumenta
        Primera_Fila_Fin:B[P0++]=R6;

////////////////////////////////////////////////////////////////////////
/*                          Fin Primera Fila                          */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*                 Inicio Segunda fila en adelante                    */
////////////////////////////////////////////////////////////////////////

    // filas - 2
    P3+=-2;
    // columnas - 2
    P4+=-2;

    // Ciclo igual número de filas
    LSETUP(Fil_Ini,Fil_Fin)LC0=P3;

        // 1er Pixel de las filas = Ima Entrada
        Fil_Ini:R6=B[P1++](Z);
                B[P0++]=R6;

        // Ciclo igual número de columnas
        LSETUP(Col_Ini,Col_Fin)LC1=P4;

            Col_Ini:
            // Carga I11 en R5
            R5=B[P1-f_mas](Z);
            // Acumulado + I11 * M11(1/9)
            A0=R5.L*R7.L;
            // Carga I12 en R6
            R6=B[P1-filas](Z);
            // Acumulado + I12 * M12(1/9)
            A0+=R6.L*R7.L;
            // Carga I13 en R5
            R5=B[P1-f_men](Z);
            // Acumulado + I13 * M13(1/9)
```

```
            A0+=R5.L*R7.L;

            // Carga I21 en R6
            R6=B[P1-uno](Z);
            // Acumulado + I21 * M21(1/9)
            A0+=R6.L*R7.L;
            // Carga I22 en R5
            R5=B[P1](Z);
            // Acumulado + I22 * M22(1/9)
            A0+=R5.L*R7.L;
            // Carga I23 en R6
            R6=B[P1+uno](Z);
            // Acumulado + I23 * M23(1/9)
            A0+=R6.L*R7.L;

            // Carga I31 en R5
            R5=B[P1+f_men](Z);
            // Acumulado + I31 * M31(1/9)
            A0+=R5.L*R7.L;
            // Carga I32 en R6
            R6=B[P1+filas](Z);
            // Acumulado + I32 * M32(1/9)
            A0+=R6.L*R7.L;
            // Carga I33 en R5
            R5=B[P1+f_mas](Z);
            // Acumulado + I33 * M33(1/9)
            R4.L=(A0+=R5.L*R7.L);
            // Píxel de salida igual a acumulado
            B[P0++]=R4;
            // Aumenta P1 para continuar con el siguiente pixel
            Col_Fin:P1+=1;

            // Ultimo píxel de la fila = Ima entrada
            R6=B[P1++](Z);
        Fil_Fin:B[P0++]=R6;


////////////////////////////////////////////////////////////////////////
/*            Ultima Fila igual a imagen de entrada                   */
////////////////////////////////////////////////////////////////////////

        // Recupera el valor de columnas del Stack Pointer
        P4.L=LO(columnas);
        P4.H=HI(columnas);
        // P4=[SP+68];

        // Ciclo igual al numero de columnas
        LSETUP(Ultima_Fila_Ini,Ultima_Fila_Fin)LC0=P4;

        // Toma el Byte al que apunta P1, lo estiende a ceros
        // y lo carga en R6 y aumenta P1;
        Ultima_Fila_Ini:R6=B[P1++](Z);

        // Toma el primer Byte de R6 y lo carga en la posición que
        // apunta P0 y lo aumenta
```

```
        Ultima_Fila_Fin:B[P0++]=R6;

////////////////////////////////////////////////////////////////////////
/*                          Fin ultima fila                           */
////////////////////////////////////////////////////////////////////////

            SP+=24;
            (R7:0,P5:0)=[SP++];
            RTS;
            NOP;

_Suavizado.end:
```

.12. Diagrama de flujo filtro Suavizado



Algoritmo Segmentación Inicial

.13. Código Matlab

```
function [ImaOut,Promedios] = Segmentacion_Inicial(ImGray,Kbajo,Kalto)
[m,n] = size(ImGray);
D_I = double(ImGray);
Promedios = zeros(m,n);
ImaOut = ImGray;
TamVen = 16;
CantidadCiclosM = floor(m/TamVen)-1;
CantidadCiclosN = floor(n/TamVen)-1;
W = uint8(zeros(TamVen,TamVen));
for i = 0:CantidadCiclosM
    for j = 0:CantidadCiclosN
        % Ciclo para adquirir cada ventana de 64*64
        Prom = 0;
        for r = 1:TamVen
            for c = 1:TamVen
                W(r,c) = ImGray(r+(i*TamVen),c+(j*TamVen));
                Prom = Prom+D_I(r+(i*TamVen),c+(j*TamVen));
            end
        end

        % Se determina el umbral de dicha ventana
        Prom = Prom/TamVen/TamVen;
        if Prom >= 150
            K = Kalto;
        else
            K = Kbajo;
        end

        % Se elimina el fondo apartir del umbral determinado por metodo
        % del promedio %
        for r = 1:TamVen
            for c = 1:TamVen
                Promedios(r+(i*TamVen),c+(j*TamVen)) = Prom;
                if W(r,c) < Prom+K
                    ImaOut(r+(i*TamVen),c+(j*TamVen)) = 0;
                end
            end
        end
    end
end
```

```
% Donde las imagenes no son multiplos del tamaño de la ventana se
% realiza el siguiente procedimiento

m1 = floor(m/TamVen)*TamVen;
n1 = floor(n/TamVen)*TamVen;
m2 = m-m1;
n2 = n-n1;
WM2 = uint8(zeros(m2,TamVen));
WN2 = uint8(zeros(TamVen,n2));
WMN2 = uint8(zeros(m2,n2));
CantidadCiclosM1 = ceil(m/TamVen)-1;
CantidadCiclosN1 = ceil(n/TamVen)-1;
for i = CantidadCiclosM1:CantidadCiclosM1
    for j = 0:CantidadCiclosN
        % Ciclo para adquirir cada ventana de 64*64
        Prom = 0;
        for r = 1:m2
            for c = 1:TamVen
                WM2(r,c) = ImGray(r+(i*TamVen),c+(j*TamVen));
                Prom = Prom+D_I(r+(i*TamVen),c+(j*TamVen));
            end
        end

        % Se determina el umbral de dicha ventana
        Prom = Prom/TamVen/m2;

        % eliminar fondo con el umbral determinado por el metodo Otsu
        for r = 1:m2
            for c = 1:TamVen
                Promedios(r+(i*TamVen),c+(j*TamVen)) = Prom;
                if WM2(r,c)<Prom+K
                    ImaOut(r+(i*TamVen),c+(j*TamVen)) = 0;
                end
            end
        end
    end
end

for i = 0:CantidadCiclosM
    for j = CantidadCiclosN1:CantidadCiclosN1
        % Ciclo para adquirir cada ventana de 64*64
        Prom = 0;
        for r = 1:TamVen
            for c = 1:n2
                WN2(r,c) = ImGray(r+(i*TamVen),c+(j*TamVen));
                Prom = Prom+D_I(r+(i*TamVen),c+(j*TamVen));
            end
        end

        % Se determina el umbral de dicha ventana
        Prom = Prom/TamVen/n2;

        % eliminar fondo con el umbral determinado por el metodo Otsu
        for r = 1:TamVen
```

```
            for c = 1:n2
                Promedios(r+(i*TamVen),c+(j*TamVen)) = Prom;
                if WN2(r,c) < Prom+K
                    ImaOut(r+(i*TamVen),c+(j*TamVen)) = 0;
                end
            end
        end
    end
end

 for i = CantidadCiclosM1:CantidadCiclosM1
    for j = CantidadCiclosN1:CantidadCiclosN1
        % Ciclo para adquirir cada ventana de 64*64
        Prom = 0;
        for r = 1:m2
            for c = 1:n2
                WMN2(r,c) = ImGray(r+(i*TamVen),c+(j*TamVen));
                Prom = Prom+D_I(r+(i*TamVen),c+(j*TamVen));
            end
        end

        % Se determina el umbral de dicha ventana
        Prom = Prom/m2/n2;

        % eliminar fondo con el umbral determinado por el metodo Otsu
        for r = 1:m2
            for c = 1:n2
                Promedios(r+(i*TamVen),c+(j*TamVen)) = Prom;
                if WMN2(r,c) < Prom+K
                    ImaOut(r+(i*TamVen),c+(j*TamVen)) = 0;
                end
            end
        end
    end
end

end
```

.14. Código en VisualDSP++

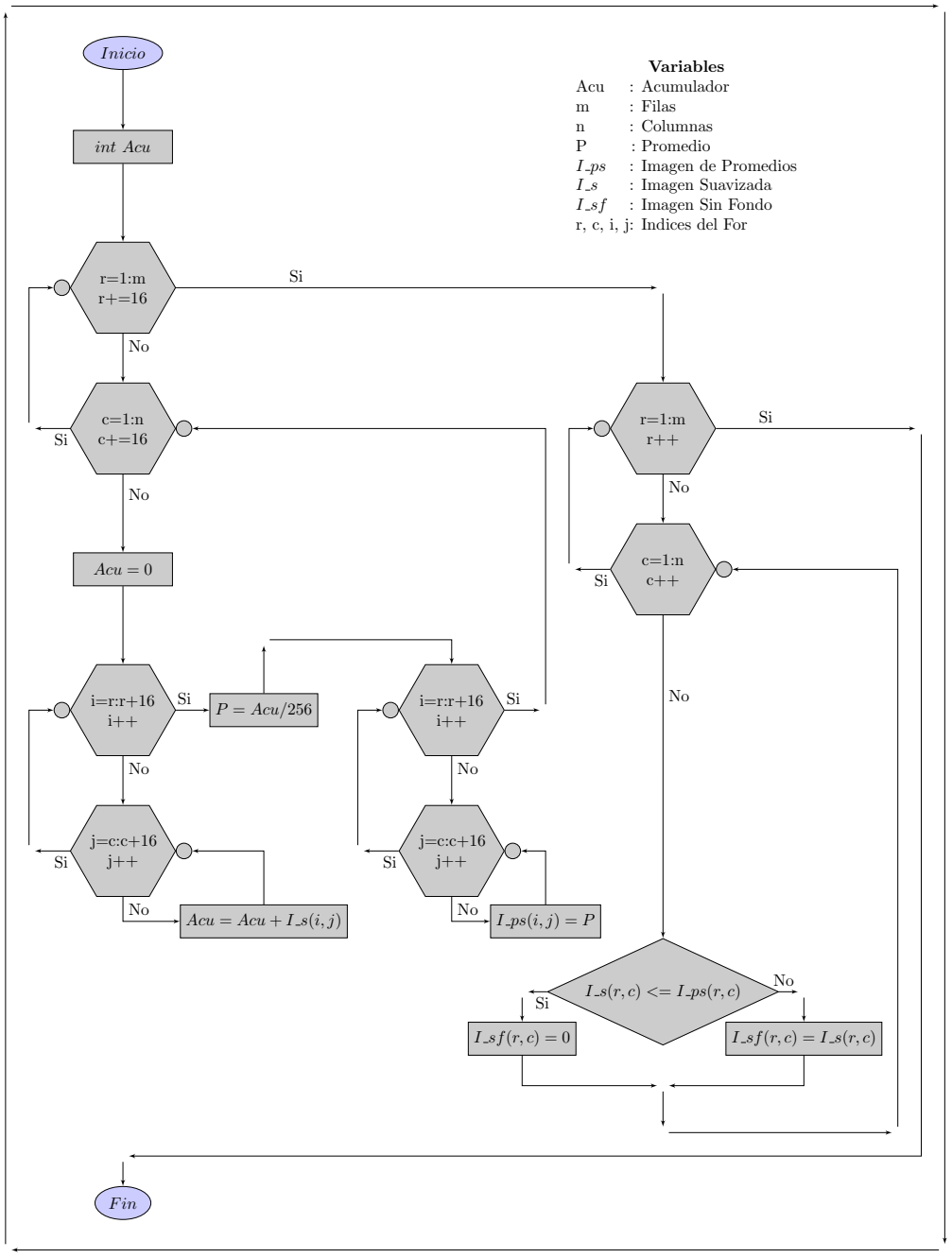
```
////////////////////////////////////////////////////////////////////////
/*                      ADSP-BF533 Elimina Fondo                      */
////////////////////////////////////////////////////////////////////////

#include "variables.h"

// Comandos del Preprocesador

//Seccion programa

.section L1_code;
.global _Segmentacion_Inicial;
.align 8;

// Etiqueta del Assembler
_Segmentacion_Inicial:

////////////////////////////////////////////////////////////////////////
/*              Inicio: Inicialización de variables                   */
////////////////////////////////////////////////////////////////////////

    [--SP]=(R7:0,P5:0);
    SP+=-24;

    P1.L=LO(Ima_filtro_direcc_inicio);
    P1.H=HI(Ima_filtro_direcc_inicio);
    P0.L=LO(elimina_fondo_direcc_inicio);
    P0.H=HI(elimina_fondo_direcc_inicio);
    P2.L=LO(promedios_direcc_inicio);
    P2.H=HI(promedios_direcc_inicio);
    P3.L=LO(Tamano_Ventana);
    P3.H=HI(Tamano_Ventana);
    R1.L=LO(filas);
    R1.H=HI(filas);
    R2.L=LO(columnas);
    R2.H=HI(columnas);
    // Guarda componente de promedio /256: (0.00390625 * 32768)
    R7.L=0X80;
    R7.H=0X80;
    // Almacena R2 (columnas) en R3 para no modificarlo
    R3=R2;
    // Numero de columnas - 16 (Para saltar de fila a fila)
    R3+=-16;
    // Guarda en el SP la cantidad de salto1
    [SP+24]=R3;
    // Almacena R2 (columnas) en R3 para no modificarlo
    R3=R2;
    // Numero de columnas*16
    R4=R3<<4;
    // ((16 * columnas) -16) (para volver atras despues de 16 salto1)
    R4+=-16;
    // Guarda en el SP la cantidad de salto2 o de retorno
```

```
    [SP+20]=R4;
    // Almacena R1 (filas) en R3 para no modificarlo
    R3=R1;
    // filas / 16
    R4=R3>>4;
    // Guarda en el SP la cantidad de iteraciones de filas
    [SP+16]=R4;
    // Almacena R2 en R3 para no modificarlo
    R3=R2;
    // columnas / 16
    R4=R3>>4;
    // Guardo en el SP la cantidad de iteraciones de columnas
    [SP+12]=R4;
    // Variable para comparar los saltos entre ventanas
    R1=[SP+16];
    // Variable para determinar el fin de los ciclos
    R2=[SP+12];
    // R0 = R3 = R4 = R5 = 0
    R0=R7-R7(NS);
    R3=R7-R7(NS);
    R4=R7-R7(NS);
    R5=R7-R7(NS);

////////////////////////////////////////////////////////////////////////
/*                Fin: Inicialización de variables                    */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*                 Inicio: Promedio de la ventana                     */
////////////////////////////////////////////////////////////////////////
Iteracion:

    // P5 Recupera del SP el salto de fila a fila
    P5=[SP+24];
    // Reinicia A0 para una nueva acumulación
    A0=R0;
    // Ciclo del Tamaño de la ventana 16*16
    LSETUP(Fil_Ini,Fil_Fin)LC0=P3;
        Fil_Ini:
            LSETUP(Col_Ini,Col_Fin)LC1=P3;
                Col_Ini:R6=B[P1++](Z);
                Col_Fin:A0+=R6.L*R7.L;
        Fil_Fin:P1=P1+P5;
        // R5 toma el valor del acumulado
        R5.L=A0;

////////////////////////////////////////////////////////////////////////
/*                   Fin: Promedio de la ventana                      */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*       Inicio: Genera una nueva ventana con el promedio             */
////////////////////////////////////////////////////////////////////////

        LSETUP(FilPro_Ini,FilPro_Fin)LC0=P3;
```

```
            FilPro_Ini:
                LSETUP(ColPro_Ini,ColPro_Fin)LC1=P3;
                    ColPro_Ini:
                    ColPro_Fin:B[P2++]=R5;
            FilPro_Fin:P2=P2+P5;

////////////////////////////////////////////////////////////////////////
/*         Fin: Genera una nueva ventana con el promedio              */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*         Inicio: Determina el tipo de salto a efectuar              */
////////////////////////////////////////////////////////////////////////

        R4+=1;
        cc=R4==R2;
        if cc JUMP SaltoVentana;
        SaltoFila:
            P5=[SP+20];
            P1-=P5;
            P2-=P5;
            JUMP Iteracion;
        SaltoVentana:
            R3+=1;
            // R4 = 0
            R4=R7-R7(NS);
            cc=R3==R1;
            if cc JUMP Comparacion;
            P5=[SP+24];
            P1-=P5;
            P2-=P5;
            JUMP Iteracion;

////////////////////////////////////////////////////////////////////////
/*          Fin: Determina el tipo de salto a efectuar                */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/* Inicio:Compara píxel vs promedio si es menor lo hace igual a cero*/
////////////////////////////////////////////////////////////////////////

        Comparacion:
            // Recupera N del SP
            P4.L=LO(N);
            P4.H=HI(N);
            // Recupera DirImaIn
            P1.L=LO(Ima_filtro_direcc_inicio);
            P1.H=HI(Ima_filtro_direcc_inicio);
            // Recupera DirImaOut
            P0.L=LO(elimina_fondo_direcc_inicio);
            P0.H=HI(elimina_fondo_direcc_inicio);
            // Recupera DirImaPromedios
            P2.L=LO(promedios_direcc_inicio);
            P2.H=HI(promedios_direcc_inicio);
            // Variable para determinar zona de alta luz
```

```
            R3.L=0X96;
            R3.H=0X0;
            LSETUP(Comparacion_Ini,Comparacion_Fin)LC1=P4;
                R6=B[P1++](Z);
                Comparacion_Ini:
                    R5=B[P2++](Z);
                    cc=R3<R5;
                    if cc JUMP VariableK;
                        R5+=20;
                    VariableK:
                            cc=R6<R5;
                            if cc JUMP zeros;
                                B[P0++]=R6;
                                JUMP Comparacion_Fin;
                                zeros:B[P0++]=R0;
                Comparacion_Fin:R6=B[P1++](Z);

////////////////////////////////////////////////////////////////////////
/*  Fin:Compara píxel vs promedio si es menor lo hace igual a cero  */
////////////////////////////////////////////////////////////////////////

            SP+=24;
            (R7:0,P5:0)=[SP++];
            RTS;
            NOP;

_Segmentacion_Inicial.end:
```

.15. Diagrama de flujo Segmentación Inicial



Algoritmo Segmentación Final

.16. Código Matlab

```
function ImaOut = Segmentacion_Final(ImGray,Fil,Col)
[m,n] = size(ImGray);
Im1 = double(ImGray);
ImaOut = ImGray;
W = zeros(1,Fil*Col);
fil = (Fil-1)/2;
col = (Col-1)/2;

Y = 0.5;
for r = 1+fil:m-fil
    for c = 1+col:n-col
        if Im1(r,c) ~= 0
            g = 1;
            for j = r-fil:r+fil
                for k = c-col:c+col
                    W(1,g) = Im1(j,k);
                    g = g+1;
                end
            end

            W2 = mean(W)*(1+Y*((std(W,1)-1)/255));
            if Im1(r,c) <= W2
                ImaOut(r,c) = 0;
            end
        end
    end
end
end
```

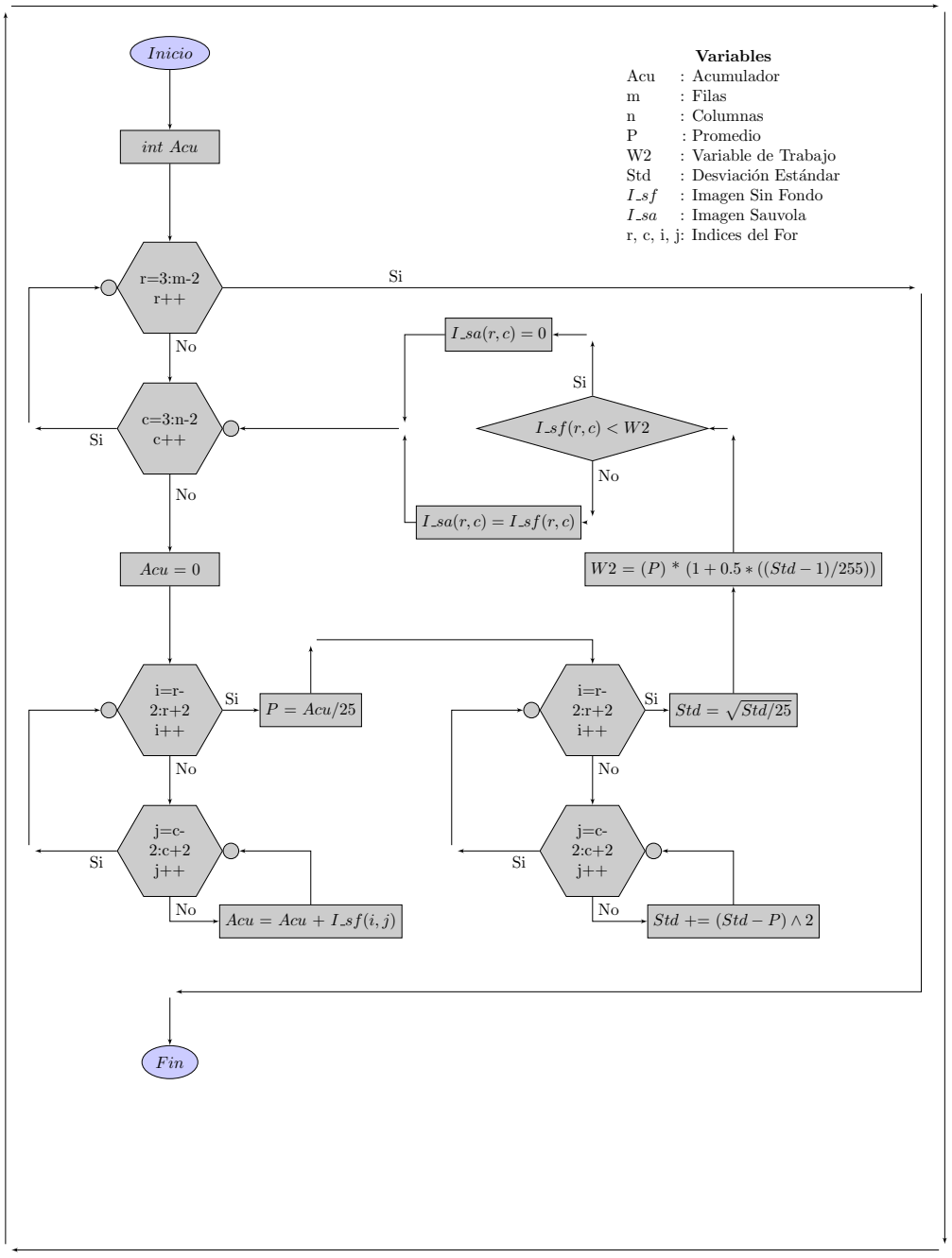
.17. Código en VisualDSP++

```
////////////////////////////////////////////////////////////////////////
/*                          ADSP-BF533 Sauvola                        */
////////////////////////////////////////////////////////////////////////

// Seleccion de datos
#include <math.h>
#include <stdio.h>

#include "variables.h"


// Comandos del preprocesador

unsigned char *a,*b;

//Seccion programa

void Segmentacion_Final(void )
{
    int i,j,k,l,m,n;
    float promedio,std,std2,W,W2;


    a=elimina_fondo_direcc_inicio;
    b=sauvola_direcc_inicio;

////////////////////////////////////////////////////////////////////////
/*         Inicio: Imagen de salida igual imagen entrada              */
////////////////////////////////////////////////////////////////////////

    for (i=0; i<=filas; i++)
    {
        for (j=0; j<=columnas; j++)
        {
            k=j+(i*columnas);
            b[k]=a[k];
        }
    }

////////////////////////////////////////////////////////////////////////
/*          Fin: Imagen de salida igual imagen entrada                */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*                      Inicio: Programa Sauvola                      */
////////////////////////////////////////////////////////////////////////

    for (i=2; i<=filas-2; i++)
    {
        for (j=2; j<=columnas-2; j++)
        {
            k=j+(i*columnas);
```

```
if(a[k])
{

promedio = 0;

////////////////////////////////////////////////////
/*         Inicio: Promedio ventana 5*5           */
////////////////////////////////////////////////////
for (m=i-2; m<=i+2; m++)
    {
        for (n=j-2; n<=j+2; n++)
            {
                l=n+(m*columnas);
                promedio = promedio + a[l];
            }

    }

promedio=promedio/25;

////////////////////////////////////////////////////
/*            Fin: Promedio ventana 5*5           */
////////////////////////////////////////////////////


////////////////////////////////////////////////////
/*          Inicio: Desviacion estándar 5*5       */
////////////////////////////////////////////////////

std2 = 0;

for (m=i-2; m<=i+2; m++)
    {
        for (n=j-2; n<=j+2; n++)
            {
                l=n+(m*columnas);
                W=a[l]-promedio;
                W=W*W;
                std2+=W;
            }

    }

std2=std2/25;
std=sqrt(std2);

////////////////////////////////////////////////////
/*          Fin: Desviacion estádar 5*5           */
////////////////////////////////////////////////////

////////////////////////////////////////////////////
/*        Inicio: Comparacion con Sauvola         */
////////////////////////////////////////////////////
```

```
            W2 = (promedio) * (1 + 0.5 * ((std-1)/255));

            if ( a[k] <= W2)
                {
                    b[k]=0;
                }

            ////////////////////////////////////////////////////
            /*          Fin: Comparacion con Sauvola          */
            ////////////////////////////////////////////////////

            }

        }
    }

}
////////////////////////////////////////////////////////////////////////
/*                      Fin: Programa Sauvola                         */
////////////////////////////////////////////////////////////////////////
```

.18. Diagrama de flujo Segmentación Final



Algoritmo Conectividad

.19. Código Matlab

```
function VecOut = Conectividad(ImGray,Umbrales)
[m,n] = size(ImGray);
D_I = double(ImGray);
Estrellas = 0;
VecInf  =zeros(4,1024);
Porcentaje = 2;

for r = 2:m-1
    for c = 2:n-1
        if (D_I(r-1,c)~=0)&&(D_I(r,c-1)~=0)&&(D_I(r,c)~=0)&& ...
                (D_I(r,c+1)~=0)&&(D_I(r+1,c)~=0)

            c_right = 1;
            stop = 0;
            while stop == 0
                Acum = 0;
                c_right = c_right+1;
                if (c+c_right <= n)&&(r+c_right <= m)
                    for i = 1:c_right+2
                        Acum = Acum+D_I(r-2+i,c+c_right);
                    end
                end
                if Acum <= (Umbrales(r,c)*(c_right+2)/Porcentaje)
                    c_right = c_right-1;
                    stop = 1;
                else
                    stop = 0;
                end
            end

            c_left = 1;
            stop = 0;
            while stop == 0
                Acum = 0;
                c_left = c_left+1;
                if (c-c_left >= 1)&&(r+c_left <= m)
                    for i = 1:c_left+2
                        Acum = Acum+D_I(r-2+i,c-c_left);
                    end
                end
```

```
                if Acum < (Umbrales(r,c)*(c_left+2)/Porcentaje)
                    c_left = c_left-1;
                    stop = 1;
                else
                    stop = 0;
                end
            end

            r_bottom = 1;

            w = c_left+c_right+1;
            r_top = 1;
            Acum = Umbrales(r,c)*w;
            while Acum > (Umbrales(r,c)*(w)/Porcentaje)
                Acum = 0;
                r_top = r_top+1;
                if r+r_top <= m
                    for i = 1:w
                        Acum = Acum+D_I(r+r_top,c-c_left+i-1);
                    end
                end
            end
            r_top = r_top-1;

            if (c_right>1)||(c_left>1)||(r_top>1)
                Estrellas = Estrellas+1;
                VecInf(1,Estrellas) = r-r_bottom;
                VecInf(2,Estrellas) = c+c_right;
                VecInf(3,Estrellas) = r+r_top;
                VecInf(4,Estrellas) = c-c_left;

                % Borrar contenido de la estrella
                for j = VecInf(1,Estrellas):VecInf(3,Estrellas)
                    for k = VecInf(4,Estrellas):VecInf(2,Estrellas)
                        D_I(j,k) = 0;
                    end
                end
            end
        end
    end
end

VecOut = zeros(4,Estrellas);
for j = 1:Estrellas
    VecOut(1,j) = VecInf(1,j);
    VecOut(2,j) = VecInf(2,j);
    VecOut(3,j) = VecInf(3,j);
    VecOut(4,j) = VecInf(4,j);
end

end
```

.20. Código en VisualDSP++

```
////////////////////////////////////////////////////////////////////////
/*                    ADSP-BF533 Conectividad                         */
////////////////////////////////////////////////////////////////////////

// Seleccion de datos
#include "variables.h"

//Seccion programa
.section L1_code;
.global _Conectividad;
.align 8;

// Etiqueta del Assembler
_Conectividad:

////////////////////////////////////////////////////////////////////////
/*               Inicio: Inicialización de variables                  */
////////////////////////////////////////////////////////////////////////

    [--SP]=(R7:0,P5:0);
    SP+=-24;

    P2.L=LO(promedios_direcc_inicio);
    P2.H=HI(promedios_direcc_inicio);
    P1.L=LO(W_sauvola_direcc_inicio);
    P1.H=HI(W_sauvola_direcc_inicio);
    P0.L=LO(conectividad_direcc_inicio);
    P0.H=HI(conectividad_direcc_inicio);
    P5.L=LO(N);
    P5.H=HI(N);
    R3=filas;
    P3=R3;
    R4=columnas;
    //  Puntero de ImaIn en la segunda fila
    P1=P3+P1;
    P1+=1;
    //  Puntero de Promedios en la segunda fila
    P2=P3+P2;
    P2+=1;
    //  N menos la primera fila
    P5-=P3;
    //  N menos la ultima fila
    P5-=P3;
    R0=0X2;
    P4=R0;
    //  N menos el primer píxel segunda fila y el ultimo píxel de
    //  penultima fila
    P5-=P4;
    R5=dos;          //  R5 = 2
    R6=dos;          //  R6 = 2

////////////////////////////////////////////////////////////////////////
```

```
/*                 Fin: Inicialización de variables                 */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*          Inicio: Imagen de salida igual imagen entrada           */
////////////////////////////////////////////////////////////////////////

    LSETUP(Fil_Ini,Fil_Fin)LC0=P5;
        Fil_Ini:cc = R5 == R3;
            if !cc JUMP Proceso;
            R6+=1;
            R5=zero;
    Proceso:R0 = B[P1 - filas](Z);
            cc = R0 == zero;
            if cc JUMP Final;
            R0 = B[P1 - uno](Z);
            cc = R0 == zero;
            if cc JUMP Final;
            R0 = B[P1](Z);
            cc = R0 == zero;
            if cc JUMP Final;
            R0 = B[P1 + uno](Z);
            cc = R0 == zero;
            if cc JUMP Final;
            R0 = B[P1 + filas](Z);
            cc = R0 == zero;
            if cc JUMP Final;

////////////////////////////////////////////////////////////////////////
/*                  Inicio: Proceso hacia la derecha                */
////////////////////////////////////////////////////////////////////////

            ProcesoDerecha:R2=uno;
            ProcesoDerecha2:R2+=1;
                R7=R3-R2;
                cc = R5 <= R7;
                if !cc JUMP FinProcesoD;
                R7=R4-R2;
                cc = R6 <= R7;
                if !cc JUMP FinProcesoD;
                R1=P1;
                R7=R1-R3;
                R0=R7+R2;
                P5=R0;
                A0 = 0;
                P4=R2;
                P4+=2;
                P3=R3;
                R1.L=0X1;
                LSETUP(AcuD_Ini,AcuD_Fin)LC1=P4;
                    AcuD_Ini:R7 = B[P5](Z);
                        P5=P5+P3;
                    AcuD_Fin:A0+=R7.L*R1.L(FU);
                    R0=A0(IS);
                    R1=B[P2](Z);
```

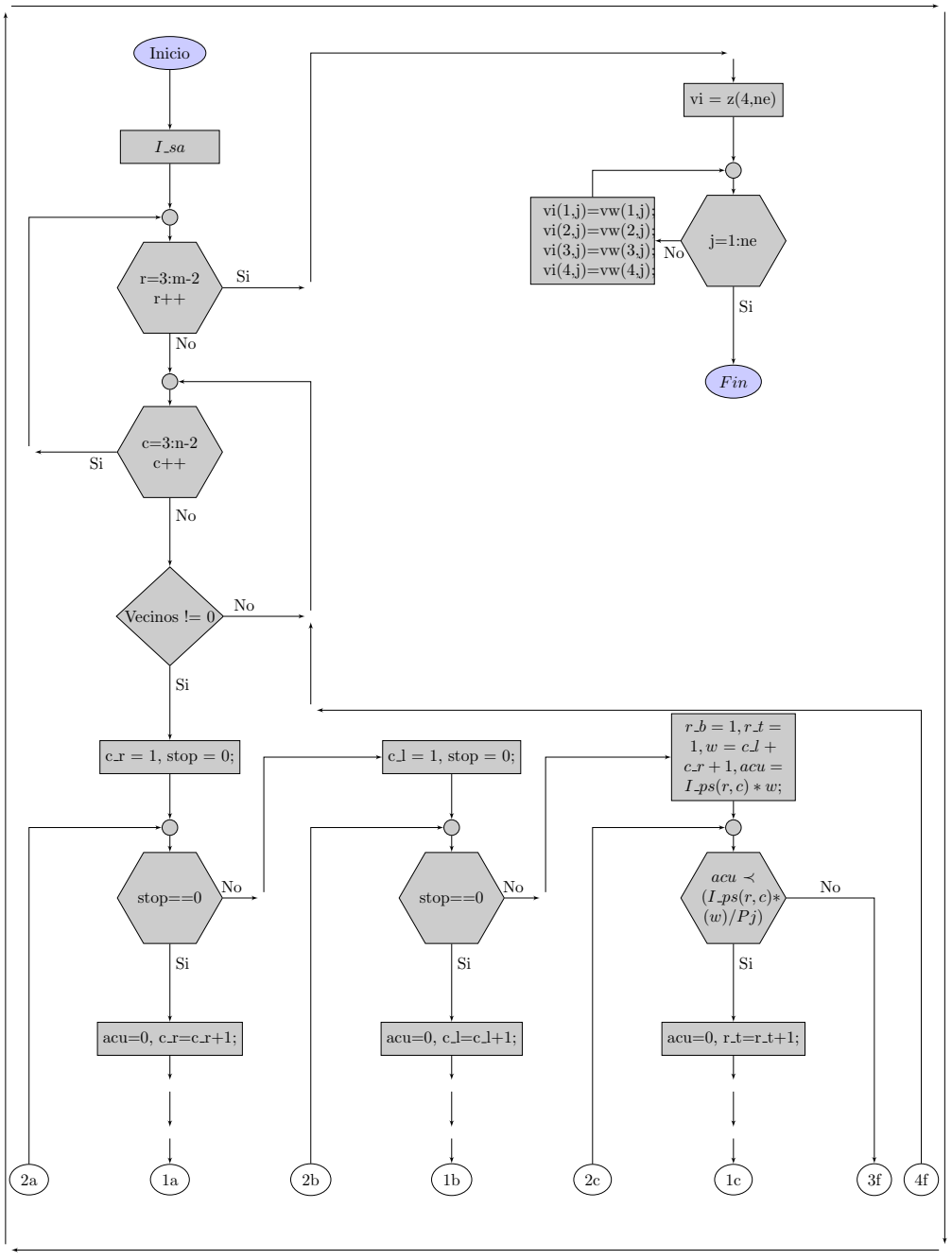
```
                    R7=R2;
                    R7+=2;
                    A1=R1.l*R7.L(FU);
                    R7=A1(IS);
                    R1=R7>>1;
                    cc = R1 <= R0;
                    if cc JUMP ProcesoDerecha2;
        FinProcesoD:R7=uno;
                    R2=R2-R7;
                    [SP+24]=R2;

////////////////////////////////////////////////////////////////////////
/*                   Fin: Proceso hacia la derecha                    */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*                Inicio: Proceso hacia la izquierda                  */
////////////////////////////////////////////////////////////////////////

            ProcesoIzquierda:R2=uno;
            ProcesoIzquierda2:R2+=1;
                R7=uno;
                R0=R2+R7;
                cc = R0 <= R5;
                if !cc JUMP FinProcesoI;
                R7=R4-R2;
                cc = R6 <= R7;
                if !cc JUMP FinProcesoI;
                R1=P1;
                R7=R1-R3;
                R0=R7-R2;
                P5=R0;
                A0 = 0;
                P4=R2;
                P4+=2;
                R1.L=0X1;
                LSETUP(AcuI_Ini,AcuI_Fin)LC1=P4;
                    AcuI_Ini:R7 = B[P5](Z);
                        P5=P5+P3;
                    AcuI_Fin:A0+=R7.L*R1.L(FU);
                    R0=A0(IS);
                    R1=B[P2](Z);
                    R7=R2;
                    R7+=2;
                    A1=R1.l*R7.L(FU);
                    R7=A1(IS);
                    R1=R7>>1;
                    cc = R1 <= R0;
                    if cc JUMP ProcesoIzquierda2;
        FinProcesoI:R7=uno;
                    R2=R2-R7;
                    [SP+20]=R2;

////////////////////////////////////////////////////////////////////////
/*                   Fin: Proceso hacia la izquierda                  */
```

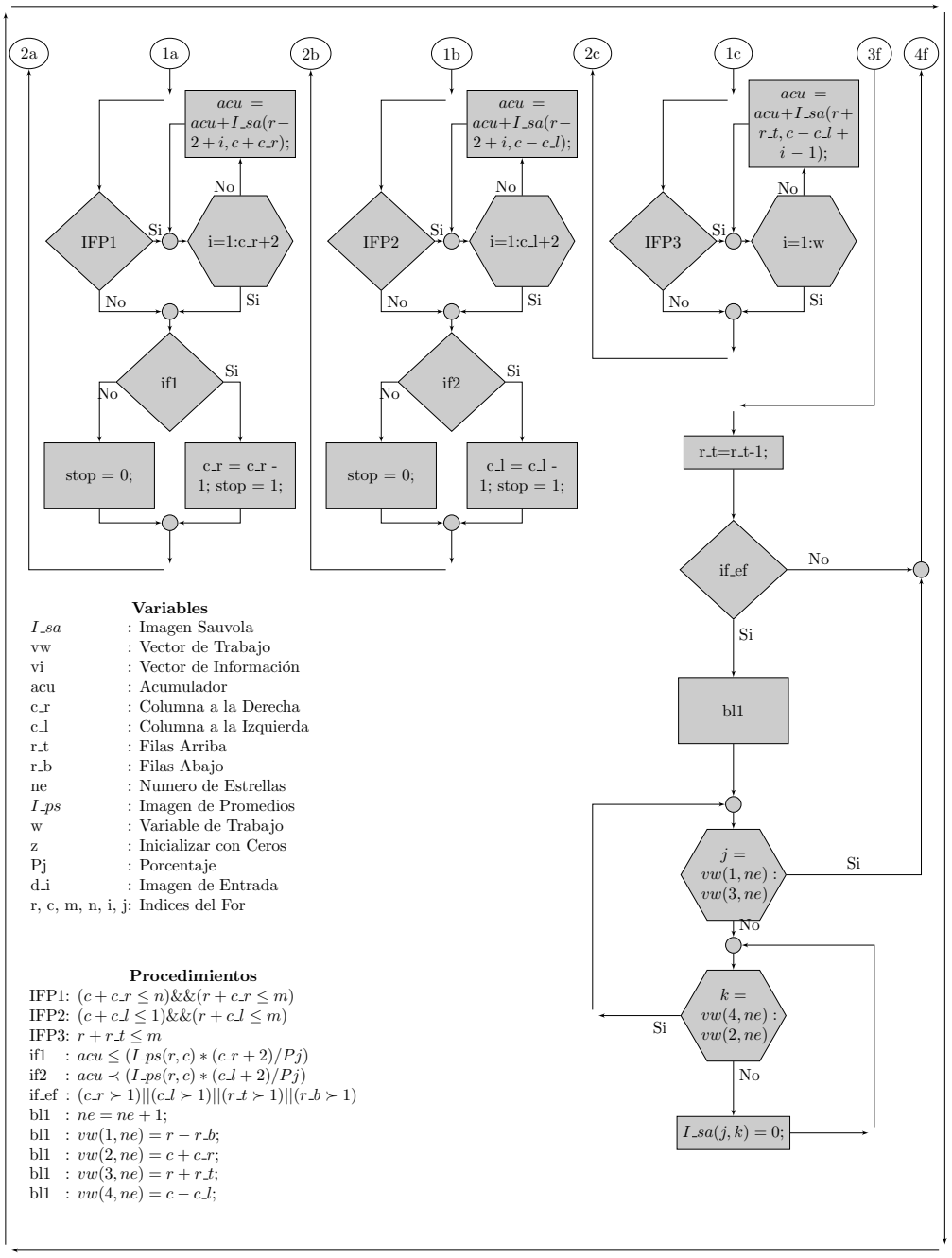
```
////////////////////////////////////////////////////////////////////////

                R2=uno;
                [SP+16]=R2;

////////////////////////////////////////////////////////////////////////
/*                    Inicio: Proceso hacia abajo                     */
////////////////////////////////////////////////////////////////////////

    ProcesoAbajo:R2=uno;
                R0=[SP+24];
                R1=[SP+20];
                R7=R0+R1;
                R0=R7+R2;
                P4=R0;
                R1=P1;
                R7=R1+R3;
                R1=[SP+24];
                R0=R7+R1;
                P5=R0;
                P5+=1;
            ProcesoAbajo2:R2+=1;
                P5-=P4;
                P5=P5+P3;
                R7=R4-R2;
                cc = R6 <= R7;
                if !cc JUMP FinProcesoA;
                A0 = 0;
                R1.L=0X1;
                LSETUP(AcuA_Ini,AcuA_Fin)LC1=P4;
                    AcuA_Ini:R7 = B[P5++](Z);
                    AcuA_Fin:A0+=R7.L*R1.L(FU);
                    R0=A0(IS);
                    R1=B[P2](Z);
                    R7=P4;
                    A1=R1.l*R7.L(FU);
                    R7=A1(IS);
                    R1=R7>>1;
                    cc = R1 <= R0;
                    if cc JUMP ProcesoAbajo2;
        FinProcesoA:R7=uno;
                    R2=R2-R7;
                    [SP+12]=R2;

////////////////////////////////////////////////////////////////////////
/*                      Fin: Proceso hacia abajo                      */
////////////////////////////////////////////////////////////////////////

                    R1=uno;
                    R0=[SP+24];
                    cc = R1 < R0;
                    if cc JUMP Estrella;
                    R0=[SP+20];
                    cc = R1 < R0;
                    if cc JUMP Estrella;
```

```
                R0=[SP+12];
                cc = R1 < R0;
                if !cc JUMP Final;

        Estrella:R0=P1;
                R1=R0-R3;
                R0=[SP+20];
                R7=R1-R0;
                [P0++]=R7;
                R2=uno;
                R0=[SP+24];
                R1=[SP+20];
                R7=R0+R1;
                R0=R7+R2;
                [P0++]=R0;
                R0=[SP+16];
                R1=[SP+12];
                R7=R0+R1;
                R0=R7+R2;
                [P0++]=R0;
                R0=zero;
                [P0++]=R0;
                R0=LC0;
                R1=LT0;
                R7=LB0;
                M0=R0;
                M1=R1;
                M2=R7;
                P5=[P0-16];
                P4=[P0-8];
                P3=R3;
                R0=zero;
                LSETUP(BorrarC_Ini,BorrarC_Fin)LC0=P4;
                    P4=[P0-12];
                    BorrarC_Ini:
                        LSETUP(BorrarF_Ini,BorrarF_Ini)LC1=P4;
                            BorrarF_Ini:B[P5++]=R0;
                            P5-=P4;
                    BorrarC_Fin:P5=P5+P3;
                R0=M0;
                R1=M1;
                R7=M2;
                LC0=R0;
                LT0=R1;
                LB0=R7;
     Final: P2+=1;
            P1+=1;
    Fil_Fin:R5+=1;          // Ultimo píxel de la fila = 0

    R1.L=LO(conectividad_direcc_inicio);
    R1.H=HI(conectividad_direcc_inicio);
    R0=P0;
    R2=R0-R1;
    R7=R2>>4;
```

```
        [SP+28]=R7;             // Total Estrellas

            SP+=24;
            (R7:0,P5:0)=[SP++];
            RTS;
            NOP;

_Conectividad.end:
```

.21. Diagrama de flujo Conectividad





Variables

I_sa	: Imagen Sauvola
vw	: Vector de Trabajo
vi	: Vector de Información
acu	: Acumulador
c_r	: Columna a la Derecha
c_l	: Columna a la Izquierda
r_t	: Filas Arriba
r_b	: Filas Abajo
ne	: Numero de Estrellas
I_ps	: Imagen de Promedios
w	: Variable de Trabajo
z	: Inicializar con Ceros
Pj	: Porcentaje
d_i	: Imagen de Entrada
r, c, m, n, i, j:	Indices del For

Procedimientos

IFP1: $(c + c\_r \leq n)\&\&(r + c\_r \leq m)$
IFP2: $(c + c\_l \leq 1)\&\&(r + c\_l \leq m)$
IFP3: $r + r\_t \leq m$
if1 : $acu \leq (I\_ps(r,c) * (c\_r + 2)/Pj)$
if2 : $acu \prec (I\_ps(r,c) * (c\_l + 2)/Pj)$
if_ef : $(c\_r \succ 1)||(c\_l \succ 1)||(r\_t \succ 1)||(r\_b \succ 1)$
bl1 : $ne = ne + 1;$
bl1 : $vw(1,ne) = r - r\_b;$
bl1 : $vw(2,ne) = c + c\_r;$
bl1 : $vw(3,ne) = r + r\_t;$
bl1 : $vw(4,ne) = c - c\_l;$

Algoritmo Centroide

.22. Código Matlab

```
function [Fil,Col,Acu] = Centroide(ImaIn,VecInfo)
D_Ima = double(ImaIn);
[~,NumEstrellas] = size(VecInfo);
Fil = zeros(1,NumEstrellas);
AcuFil = zeros(1,NumEstrellas);
AcuCol = zeros(1,NumEstrellas);
Col = zeros(1,NumEstrellas);
Acu = zeros(1,NumEstrellas);
for i = 1:NumEstrellas
    for j = VecInfo(1,i):VecInfo(3,i)
        for k = VecInfo(4,i):VecInfo(2,i)
            Acu(1,i) = Acu(1,i)+D_Ima(j,k);
        end
    end
end
% R = round((1:size(I,1))*sum(I,2)/sum(I(:)));

for i = 1:NumEstrellas
    Fil(1,i) = 0;
    for j = VecInfo(1,i):VecInfo(3,i)
        AcuCol(1,i) = 0;
        for k = VecInfo(4,i):VecInfo(2,i)
            AcuCol(1,i) = AcuCol(1,i)+D_Ima(j,k);
        end
        Fil(1,i) = j*AcuCol(1,i)+Fil(1,i);
    end
    Fil(1,i) = Fil(1,i)/Acu(1,i);
end

for i = 1:NumEstrellas
    Col(1,i) = 0;
    for k = VecInfo(4,i):VecInfo(2,i)
        AcuFil(1,i) = 0;
        for j = VecInfo(1,i):VecInfo(3,i)
            AcuFil(1,i) = AcuFil(1,i)+D_Ima(j,k);
        end
        Col(1,i) = k*AcuFil(1,i)+Col(1,i);
    end
    Col(1,i) = Col(1,i)/Acu(1,i);
end
```

```
end
```

.23. Código en VisualDSP++

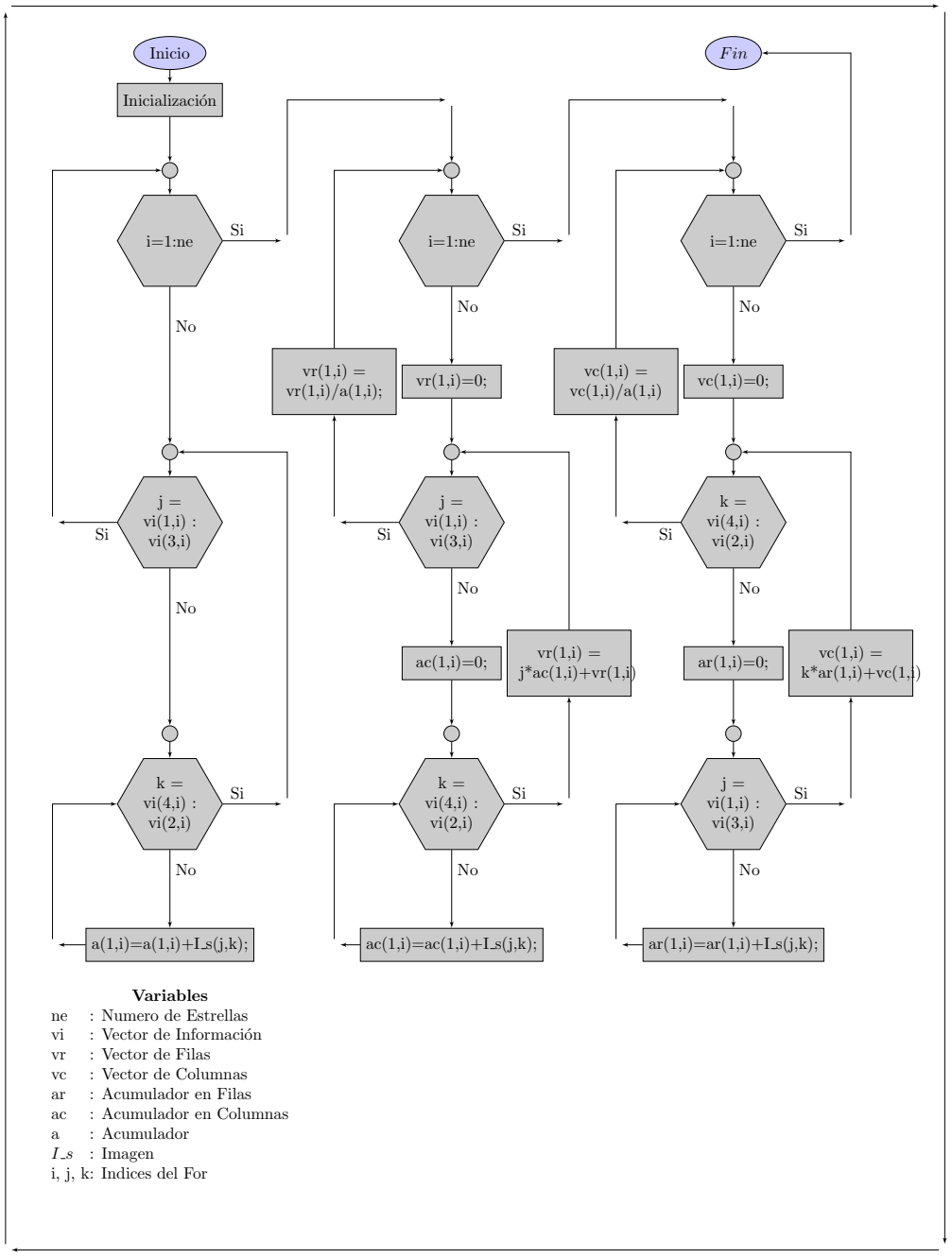
```
////////////////////////////////////////////////////////////////////////
/*                      ADSP-BF533 Centroide                          */
////////////////////////////////////////////////////////////////////////

// Seleccion de datos
#include "variables.h"

//Seccion programa
.section L1_code;
.global _Centroide;
.align 8;

// Etiqueta del Assembler
_Centroide:

////////////////////////////////////////////////////////////////////////
/*              Inicio: Inicialización de variables                   */
////////////////////////////////////////////////////////////////////////

    [--SP]=(R7:0,P5:0);
    SP+=-24;

    P2.L=LO(filas);
    P2.H=HI(filas);
    P1.L=LO(conectividad_direcc_inicio);
    P1.H=HI(conectividad_direcc_inicio);
    P0.L=LO(centroide_direcc_inicio);
    P0.H=HI(centroide_direcc_inicio);
    R5.L=0X0;
    R5.H=0X100;
    R7=[SP+28];
    R6=zero;

////////////////////////////////////////////////////////////////////////
/*                 Fin: Inicialización de variables                   */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*        Inicio: Acumulador píxeles de toda la estrella              */
////////////////////////////////////////////////////////////////////////

    Proceso:R6+=1;
            cc = R6 == R7;
            if cc JUMP Final;
            R3=[P1++];
            R3=R3-R5;
            P5=R3;
            P3=[P1++];
            M3=P3;
            P4=[P1++];
            P1+=4;
            R1=uno;
```

```
            A0=zero;
            LSETUP(AcuTC_Ini,AcuTC_Fin)LC0=P4;
                AcuTC_Ini:
                LSETUP(AcuTF_Ini,AcuTF_Fin)LC1=P3;
                    AcuTF_Ini:R0 = B[P5++](Z);
                    AcuTF_Fin:A0+=R0.L*R1.L(FU);
                    P5-=P3;
                AcuTC_Fin:P5=P5+P2;
            R4=A0(IS);

////////////////////////////////////////////////////////////////////////
/*          Fin: Acumulador píxeles de toda la estrella               */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/*      Inicio: Acumulador píxeles de las columnas para hallar        */
/*                    la coordenada de las fila                       */
////////////////////////////////////////////////////////////////////////

            A1=0;
            R2=0;
            P5=R3;
            LSETUP(AcuF1_Ini,AcuF1_Fin)LC0=P3;
                AcuF1_Ini:R2+=1;
                            A0=0;
                            M0=P5;
                LSETUP(AcuF2_Ini,AcuF2_Fin)LC1=P4;
                    AcuF2_Ini:R0 = B[P5](Z);
                                P5=P5+P2;
                    AcuF2_Fin:A0+=R0.L*R1.L(FU);
                    P5=M0;
                    R0=A0(IS);
                    A1+=R0.L*R2.L(FU);
                AcuF1_Fin:P5+=1;
            R1=A1(IS);
            R0=R4>>1;
            R2=zero;
  Divicion1:R2+=1;
            R1=R1-R4;
            CC=AN;
            if !cc JUMP Divicion1;
            R1=R1+R4;
            cc = R0 <= R1;
            if cc JUMP siguiente1;
            R2+=-1;
  siguiente1:[SP+24]=R2;

////////////////////////////////////////////////////////////////////////
/* Fin:Acumulador píxeles de las columnas para hallar la coordenada */
/*                            de las fila                            */
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
/* Inicio:Acumulador píxeles de las filas para hallar la coordenada */
/*                           de las columnas                         */
```

```
////////////////////////////////////////////////////////////////////

            P3=M3;
            R1=uno;
            A1=0;
            R2=0;
            P5=R3;
            LSETUP(AcuF3_Ini,AcuF3_Fin)LC0=P4;
                AcuF3_Ini:R2+=1;
                            A0=0;
                            M0=P5;
                LSETUP(AcuF4_Ini,AcuF4_Fin)LC1=P3;
                    AcuF4_Ini:R0 = B[P5++](Z);
                    AcuF4_Fin:A0+=R0.L*R1.L(FU);
                    P5=M0;
                    R0=A0(IS);
                    A1+=R0.L*R2.L(FU);
                AcuF3_Fin:P5=P5+P2;
            R1=A1(IS);
            R0=R4>>1;
            R2=zero;
  Divicion2:R2+=1;
            R1=R1-R4;
            CC=AN;
            if !cc JUMP Divicion2;
            R1=R1+R4;
            cc = R0 <= R1;
            if cc JUMP siguiente2;
            R2+=-1;
  siguiente2:[SP+20]=R2;

////////////////////////////////////////////////////////////////////
/*  Fin Acumulador píxeles de las filas para hallar la coordenada   */
/*                            de las columnas                       */
////////////////////////////////////////////////////////////////////

            R0=[SP+24];
            R0+=-1;
            R1=[SP+20];
            R1+=-1;
            R2=filas;
            A0=R1.L*R2.L(FU);
            R4=A0(IS);
            R2=R3+R0;
            R4=R2+R4;
            [P0++]=R4;
            [P0++]=R0;
            [P0++]=R1;
            P0+=4;
            JUMP Proceso;
    Final:  SP+=24;
            (R7:0,P5:0)=[SP++];
            RTS;
            NOP;
```

```
_Centroide.end:
```

.24. Diagrama de flujo Centroide



Variables

ne : Numero de Estrellas
vi : Vector de Información
vr : Vector de Filas
vc : Vector de Columnas
ar : Acumulador en Filas
ac : Acumulador en Columnas
a : Acumulador
I_s : Imagen
i, j, k: Indices del For