Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Módulo de reducción de dimensiones espectrales en un sistema de reconocimiento de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Mario Sequeira Zúñiga

Cartago, 22 de junio de 2011

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Mario Sequeira Zúñiga

Cartago, 22 de junio de 2011

Céd: 1-1160-0365

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Dr. Alfonso Chacón Rodríguez
Profesor Lector

M.Sc. Néstor Hernández Hostaller
Profesor Lector

Dr. Pablo Alvarado Moya
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 17 de junio de 2011

Resumen

Como parte de una iniciativa de protección ambiental se persigue implementar una red inalámbrica de sensores con capacidad de detectar sonidos de disparos y motosierras y alertar en tiempo real a las autoridades correspondientes la presencia de actividades ilegales. Dicha red de sensores está compuesta por un conjunto de nodos sensoriales, que son sistemas empotrados diseñados a la medida con cuatro componentes principales: una unidad de recolección de energía del entorno, una unidad de comunicación encargada de transmitir y recibir paquetes de datos hacia y desde nodos vecinos, una unidad computacional que controla todo el nodo y finalmente sensores que vigilan el entorno. Los requisitos de bajo consumo energético en el nodo, impuestos por las limitaciones de consecución de energía en el bosque, obligan a que parte del procesamiento sea delegado a circuitos integrados de aplicación específica, cuyo diseño y tecnología se orientan precisamente al bajo consumo.

En este trabajo se describe el desarrollo del hardware para una sección de reducción de dimensiones basada en discriminantes de Fisher, que es parte de la cadena de procesamiento para el reconocimiento de patrones acústicos. Se propone un sistema que consta de dos fases de operación: una de entrenamiento y otra de ejecución. Para llevar a cabo el entrenamiento se utilizan en conjunto herramientas de software y hardware. Un generador escrito en C++ permite generar automáticamente el código VHDL del hardware optimizado para la reducción energética. Durante la etapa de ejecución, se utilizan multiplicadores CSD (canonical sign-digit) de bajo consumo energético, que permiten reducir el área y la complejidad del diseño. Los recursos utilizados por la implementación propuesta indican un ahorro en área y energía en comparación con otros multiplicadores.

Palabras clave: Reconocimiento de patrones acústicos, LDA, PCA, CSD, FPGA, VHDL.

Abstract

As an iniciative for enviromental protection, it is pursued the implementation of a wireless sensor network capable of detecting gunshots and chainsaws to alert in real-time relevant authorities about the presence of illegal activities. This sensor network consists of a set of sensor nodes which are customized embedded systems with four principal components: a unit for environmental energy harvesting, a communications unit which receives and transmits data packets from and toward neighbor nodes, a computational unit who controls the node, and finally the sensors that monitor the environment. Node's low energy consumption requirements, imposed by limitations of finding energy sources in the forest, force that part of the required processing is delegated to application-specific integrated circuits, whose design and technology are oriented precisely to low power consumption.

This work describes the hardware development of a dimension reduction module based on Fisher's discriminant, which is part of a processing chain for acoustic pattern recognition. A system with two operation stages: training and execution, is presented. For training, software and hardware tools are used altogether. A generator written in C++ is used to generate VHDL code of the optimized hardware for power reduction. During the execution stage, low power consumption CSD (canonical sign-digit) multipliers reduces area and complexity. The resources used in the proposed implementation indicate less area requirements and energy savings when compared to other multipliers.

Keywords: Acoustic pattern recognition, LDA, PCA, CSD, FPGA, VHDL.

a mis queridos padres

Agradecimientos

Este trabajo es la culminación del esfuerzo realizado durante los últimos años, no sólo por mi persona, sino por todos quienes han estado a mi alrededor y de una u otra forma me han brindado su ayuda, y no hubiera sido posible sin la colaboración de todos ustedes.

Quiero agradecer principalmente a mis padres, que me han apoyado, ayudado, comprendido, cuidado y acompañado durante toda mi vida. Todo lo que soy es gracias a ustedes. También a mi hermano, quien ha sido mi único compañero y amigo verdadero a través de todos estos años.

Gracias a todos las personas que me ayudaron durante este camino: a mis amigos, a mis compañeros, que también terminaron siendo grandes amigos después de todas las luchas que tuvimos que dar en esta carrera, tanto estudiando, haciendo proyectos, o simplemente estando ahí. A todos los profesores del Tec que me ayudaron, gracias porque su trabajo es muy importante. A todas las personas que hicieron más fácil mi estancia en este lugar. Al Tec por brindarnos una oportunidad de superación no sólo para mi, sino para todos los que tuvimos la dicha de estudiar aquí para poder servir al país. A Erick Salas por su buena disposición y ayuda en este proyecto. Y por su puesto al doc, que siempre me brindó su ayuda de principio a fin, su buena disposición, tiempo, colaboración, explicaciones, por la invitación a la conferencia y el proceso que eso significó, por todo lo que me enseñó. Gracias por ayudar a muchas personas como yo. Gracias de veras, profe Pablo Alvarado.

A todos gracias de corazón. Espero poder retribuirles de alguna manera a ustedes todo lo que han hecho por mí.

Gracias de verdad

Mario Sequeira Zúñiga

Cartago, 22 de junio de 2011

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

ASIC	Circuito Integrado de Aplicación Específica
CSD	Canonical Signed-Digit
FPGA	Field Programmable Gate Arrays
LDA	Análisis de discriminantes lineales
LUT	Look-Up Table
PCA	Análisis de componentes principales
RIS	Réd inalámbrica de sensores
SiRPA	Sistema de Reconocimiento de Patrones Acústicos
VHDL	VHSIC (Very High Speed Integrated) Hardware Description Language

Notación general

A Matriz.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

$\underline{\mathbf{x}}$ Vector.

$$\underline{\mathbf{x}} = [x_1 \; x_2 \; \ldots \; x_n]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Capítulo 1

Introducción

El exterminio de bosques impacta con la destrucción de ecosistemas, el detrimento de la calidad del aire, el incremento en desastres naturales a causa del aumento en la erosión y la modificación de los sistemas naturales de retención de agua, por mencionar algunos efectos. En la actualidad ya se cuantifican las implicaciones socioeconómicas de la deforestación por medio de Certificados Transferibles de Emisiones de Carbono (CTO) [15] o los Pagos por Servicios Ambientales [4]. Con consecuencias del calentamiento global cada vez mayores, el interés por la protección de los bosques existentes crece y se plantean iniciativas para recuperar zonas boscosas perdidas.

Los bosques y sus ecosistemas son amenazados principalmente por incendios forestales causados por mano humana o efectos naturales, así como por actividades ilegales de caza y tala. En los bosques tropicales unos pocos guarda-parques son responsables de vigilar miles de hectáreas, con recursos fuertemente limitados, lo que evita una cobertura real de protección.

La toma de decisiones para la protección de las zonas boscosas debe ser ágil y para ello es necesario contar con información fiable sobre eventos anormales, en el preciso instante y lugar donde éstos ocurren, y es aquí donde la tecnología de redes de sensores entra en juego. Una red inalámbrica de sensores (RIS) está conformada por un conjunto de nodos con poder computacional restringido, equipados con unidades de comunicación inalámbrica y de sensado [18]. Estos nodos sensoriales se diseminan sobre la región a vigilar, donde cada nodo sensorial es responsable de extraer y procesar datos del entorno y enviarlos a través de uno o más nodos hacia un *sumidero* que transfiere la información a un usuario final. En 2003 la revista *Technology Review* del Instituto Tecnológico de Massachusetts seleccionó las RIS como una de las diez tecnologías emergentes que cambiarán el mundo [11].

El presente trabajo forma parte del desarrollo de un módulo para nodos sensoriales de una RIS dedicada a la detección de sonidos de disparos y motosierras en bosques y áreas protegidas. La arquitectura del Sistema de Reconocimiento de Patrones Acústicos (SiRPA) [1] se ilustra en la figura 1.1, y está conformada por etapas para el preprocesamiento

de la señal de entrada, la extracción de características y la clasificación de los patrones.

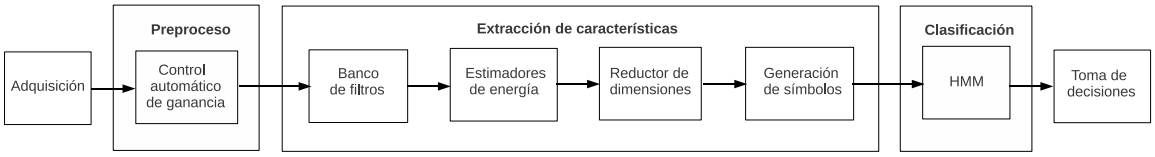


Figura 1.1: Diagrama de bloques del Sistema de Reconocimiento de Patrones Acústicos (SiRPA)

En concreto, este trabajo presenta resultados de la implementación de un reductor de ocho a tres dimensiones espectrales ubicado en la etapa de extracción de características, que por el contexto de aplicación debe optimizarse en consumo de energía, y complementa al sistema parcialmente implementado en [12]. Se ha utilizado el lenguaje de descripción de hardware VHDL y como plataforma de prueba se utiliza una tarjeta de desarrollo con FPGA Spartan 3E de Xilinx. El módulo de reducción de dimensiones cuenta con dos fases de operación: una de entrenamiento y otra de ejecución. Éstas se llevan a cabo utilizando en conjunto herramientas de hardware y software, propuestas en el marco de este trabajo.

El entrenamiento se basa en el Análisis de Discriminantes de Fisher, también conocido como LDA (Linear Discriminant Analysis) [3], el cual se utiliza para encontrar una transformación lineal que proyecta el espacio octodimensional de entrada en un subespacio tridimensional de tal modo que se maximice la varianza inter-clase y se minimice la varianza intra-clase.

Con el propósito de comparar los resultados de la separación de clases obtenida con LDA, se realizó el mismo proceso utilizando Análisis de Componentes Principales (PCA). Ésta es una técnica de reducción orientada a la representación de señales. Su objetivo es mapear las muestras con precisión en el subespacio dimensional, por lo que hace una proyección preservando la mayor cantidad de información posible.

El hardware es equipado con un módulo que permite transmitir a un computador personal los datos a las salida de los módulos de estimación de energía de las bandas espectrales calculadas por el banco de filtros (ver figura 1.1). Con los datos recopilados y agrupados por clases, fuera de línea se realiza el entrenamiento del módulo para encontrar los coeficientes de la matriz de transformación con que se realiza la proyección.

La implementación de los multiplicadores necesarios para la multiplicación de una matriz de tamaño 3×8 con el vector octodimensional utiliza codificación CSD (Canonical Signed-Digit) para reducir la complejidad de multiplicadores sin acarreo, lo que aporta al objetivo de bajo consumo energético.

Para facilitar la implementación en hardware reconfigurable del reductor y los multiplicadores que lo componen, o su eventual rediseño, se desarrolló una aplicación de software para generar código VHDL automáticamente, el cual puede ser sintetizado posteriormente para diferentes tecnologías de FPGA.

1.1 Objetivos y estructura del documento

El objetivo principal de éste trabajo ha sido la implementación en hardware reconfigurable de un módulo reductor de ocho a tres dimensiones espectrales, el cual debe acoplarse a la etapa de extracción de características de la arquitectura del SiRPA propuesta en [12]. Con el propósito de optimizar la reducción se debe utilizar una técnica enfocada a la separación de clases que ayude a facilitar el problema de clasificación. Además, la implementación en hardware del reductor debe minimizar el uso de área y el consumo energético.

Este documento se ha estructurado de la siguiente manera: En el capítulo 2 se describe la teoría y fundamentos matemáticos utilizados en el desarrollo del proyecto: la transformación vectorial que permite la reducción de dimensiones y las técnicas aplicadas para éste propósito, conocidas como LDA y PCA. También se explica la técnica CSD utilizada para la implementación de multiplicadores de bajo consumo y la distancia de Bhattacharyya, que permite comparar la similitud de dos distribuciones probabilísticas.

En el capítulo tres se justifica la reducción de dimensiones, y se describe cómo se utiliza LDA y PCA para realizar dicho proceso. También explica cómo mejorar el rendimiento del sistema utilizando codificación CSD para realizar las multiplicaciones requeridas. Se presenta además la estructura de un programa de C++ para generar automáticamente código VHDL para la implementación de los multiplicadores en hardware reconfigurable.

En el capítulo 4 se presentan los resultados de la separación de clases obtenida en un espacio tridimensional utilizando las técnicas descritas anteriormente. También se demuestra la eficiencia y la mejora en rendimiento con la utilización de los multiplicadores diseñados, en comparación con el uso de multiplicadores binarios normales. Además se resumen los resultados de la síntesis y uso de recursos de hardware del reductor de dimensiones.

Finalmente en el capítulo 5, se presentan las conclusiones obtenidas y algunas recomendaciones.

Capítulo 2

Marco Teórico

2.1 Antecedentes

El análisis de discriminantes de Fisher [3] encuentra amplia aplicación en el reconocimiento de patrones (por ejemplo [16, 8, 17]). Durante los procesos de reconocimiento, el análisis de discriminantes se reduce a una proyección lineal implementada como producto matriz-vector. El uso de hardware reconfigurable (FPGA) es utilizado en implementaciones que aprovechan alto paralelismo para obtener soluciones rápidas al producto de matriz-vector (entre otras [19, 7, 10]). Para la red de sensores en el bosque se persiguen sin embargo soluciones de bajo consumo energético, por sobre aquellas que buscan velocidad.

En el caso particular de LDA, el producto matriz-vector se caracteriza porque los elementos de la matriz son constantes, determinados previamente en una etapa de entrenamiento. En [9] se comparan diferentes implementaciones de multiplicadores para el caso particular donde uno de los multiplicandos es constante. Se concluye que los multiplicadores CSD (Canonic-Signed Digit) son la mejor opción para diseños de bajo consumo, ya que son de menor tamaño y más eficientes en el uso de energía que los multiplicadores de propósito general. En ese mismo trabajo se desarrolló un programa escrito en C++ para la generación de código VHDL de un multiplicador por un término constante utilizando ésta técnica. Dichos resultados serán extendidos en este trabajo para el caso de productos matriz-vector.

En [12] se demuestra que con multiplicadores CSD de 16 bits, sólo se produce una desviación máxima del 0,01% respecto a los multiplicadores binarios estándar. Adicionalmente, la etapa de extracción de características del SiRPA utiliza también codificación CSD para las multiplicaciones requeridas por el banco de filtros.

2.2 Reducción de dimensiones

El cuerpo de datos necesario para entrenar un sistema de reconocimiento de patrones crece exponencialmente con la cantidad de dimensiones del espacio de entrada; este principio se conoce como la *maldición de la dimensionalidad* [3]. En el caso concreto de este proyecto, la captura de sonidos en el bosque, incluyendo a disparos y motosierras, es un proceso costoso, y por tanto el sistema de reconocimiento de patrones debe poder generalizar utilizando los datos de entrenamiento con que se disponga.

Desde un punto de vista energético, se sacrifica área y consumo energético en el circuito de reducción de dimensiones para evitar un mayor consumo y área en las etapas posteriores, que de otra forma deberían tratar con datos vectoriales en ocho dimensiones.

Para el caso de generación de símbolos en SiRPA se determinó empíricamente que con 32 símbolos se obtienen resultados satisfactorios de clasificación cuando dichos símbolos se ubican en un espacio tridimensional [14]. El módulo de generación de símbolos tiene como tarea buscar la posición del símbolo más cercano al patrón que se encuentre a su entrada. Asumiendo la tendencia exponencial que predice la *maldición de la dimensionalidad*, se requerirían más de 32000 símbolos para lograr poblar de forma equivalente el espacio en ocho dimensiones, lo que implica mayor tiempo, área y costo energético para encontrar cuál de estos símbolos es el más cercano a cada patrón octodimensional que aparezca en la entrada. En vez de cinco cálculos de distancia con vectores de 3 dimensiones, se requerirían 16 cálculos de distancia con vectores en 8 dimensiones, esto sin contar que los recursos de memoria y cálculo utilizados por los Modelos Ocultos de Markov (HMM) crecen polinomialmente con el número de símbolos utilizado. Lo anterior justifica la reducción del número de dimensiones de los datos antes de la generación de símbolos y las etapas de clasificación.

Una reducción de la dimensionalidad conduce inevitablemente a pérdidas de información. En la mayoría de los casos, la información descartada se compensa al lograrse un mapeo más preciso en un espacio de menos dimensiones. Sin embargo, las pérdidas deben minimizarse, ya que puede pasar que dicha información sea clave en las siguientes etapas del sistema de reconocimiento de patrones. Se deben elegir métodos que permitan entrenar el reductor de dimensiones de forma supervisada, utilizando la información de pertenencia de cada patrón a alguna de las clases de interés ('*motosierra*', '*bosque*' o '*disparo*'). El uso de métodos de reducción de dimensiones "sin supervision" que no toman en cuenta información de las clases objetivo, puede llevar a resultados menos eficientes desde el punto de vista del problema de la clasificación [5]. También, debido a los objetivos de este proyecto, se deben utilizar técnicas de reducción de dimensiones que permitan una implementación en hardware sencilla y de bajo consumo energético.

2.2.1 Transformación lineal

Cuando un vector n-dimensional $\underline{\mathbf{x}}$ es transformado linealmente a otro vector $\underline{\mathbf{y}}$ de m-dimensiones, $\underline{\mathbf{y}}$ se expresa como una función de $\underline{\mathbf{x}}$ de la forma

$$\underline{\mathbf{y}} = \mathbf{W}\underline{\mathbf{x}}$$

donde $\mathbf{W}$ es una matriz de $m \times n$. Los elementos del vector $\underline{\mathbf{x}}$ del espacio de entrada se multiplican por una matriz $\mathbf{W}$ con parámetros ajustables, con lo que se obtiene el vector $\underline{\mathbf{y}}$ con la reducción dimensional deseada.

En el caso de SiRPA $\underline{\mathbf{x}}$ tiene ocho dimensiones, $\underline{\mathbf{y}}$ tres, y $\mathbf{W}$ es una matriz de 3×8. La figura 2.1 ilustra este proceso de transformación.

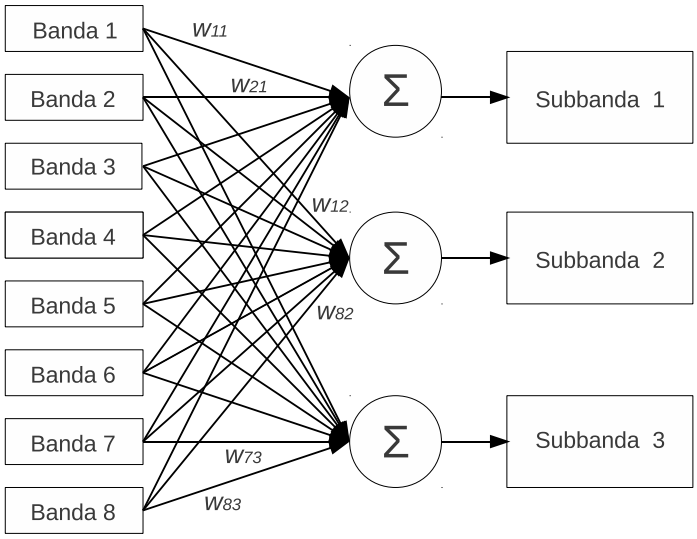


Figura 2.1: Transformación lineal.

Existen diversas técnicas de reducción de dimensiones que permiten obtener la matriz $\mathbf{W}$. En éste trabajo se basa en los métodos conocidos como LDA y PCA, los cuales se detallan a continuación.

2.2.2 Análisis de discriminantes lineales (LDA)

El Análisis de Discriminantes Lineales (LDA), también conocido como Discriminante Lineal de Fisher, permite obtener una reducción óptima de la dimensionalidad. El LDA proyecta el espacio vectorial de entrada $\underline{\mathbf{x}}$ (en el caso de SiRPA, ocho dimensiones) a un subespacio de menor dimensión $\underline{\mathbf{y}}$ (en SiRPA, tridimensional) con una orientación tal que maximiza la varianza inter-clase (separación entre las distintas clases) y a la vez minimiza

la varianza intra-clase (separación entre los miembros de una misma clase) [3], realizando la transformación lineal

$$\underline{\mathbf{y}} = \mathbf{W}\underline{\mathbf{x}}$$

La matriz $\mathbf{W}$ que separa mejor las clases cumple con la siguiente función de criterio:

$$J(\mathbf{W}) = \frac{|\hat{\mathbf{S}}_B|}{|\hat{\mathbf{S}}_W|} = \frac{|\mathbf{W}^T\mathbf{S}_B\mathbf{W}|}{|\mathbf{W}^T\mathbf{S}_W\mathbf{W}|}$$

donde $\mathbf{S}_B$ es la matriz de covarianza inter-clases y $\mathbf{S}_W$ es la matriz de covarianza intra-clase.

Para el caso de múltiples clases, se asume que la dimensionalidad del espacio de entrada d es mayor que el número de clases c, esto es $d > c$.

La matriz de covarianza intra-clase $\mathbf{S}_W$ es

$$\mathbf{S}_W = \sum_{k=1}^{c} \mathbf{S}_k$$

donde

$$\mathbf{S}_k = \sum_{n \epsilon C_k} (\underline{\mathbf{x}}^n - \underline{\mathbf{m}}_k)(\underline{\mathbf{x}}^n - \underline{\mathbf{m}}_k)^T$$

y

$$\underline{\mathbf{m}}_k = \frac{1}{N_k} \sum_{n \epsilon C_k} \underline{\mathbf{x}}^n$$

N_k es el número de patrones en la clase C_k y $\underline{\mathbf{x}}^n$ la n-ésima muestra.

La matriz de covarianza inter-clases $\mathbf{S}_B$ se define como

$$\mathbf{S}_B = \sum_{k=1}^{c} N_k(\underline{\mathbf{m}}_k - \underline{\mathbf{m}})(\underline{\mathbf{m}}_k - \underline{\mathbf{m}})^T$$

donde $\underline{\mathbf{m}}$ es el vector de media del conjunto total de datos

$$\underline{\mathbf{m}} = \frac{1}{N} \sum_{n=1}^{N} \underline{\mathbf{x}}^n = \frac{1}{N} \sum_{k=1}^{c} N_k \underline{\mathbf{m}}_k$$

y N es la cantidad total de puntos del conjunto de datos.

La forma de encontrar el $\mathbf{W}$ óptimo es en un problema de eigenvalores y eigenvectores, los que deben cumplir con

$$\mathbf{S}_B \underline{\mathbf{w}}_i = \lambda_i \mathbf{S}_W \underline{\mathbf{w}}_i$$

donde estos eigenvalores λ pueden encontrarse como las raíces del polinomio

$$\mathbf{S}_B - \lambda_i \mathbf{S}_W = 0$$

Resolviendo

$$(\mathbf{S}_B - \lambda_i \mathbf{S}_W)\underline{\mathbf{w}}_i = 0$$

se obtienen los vectores $\underline{\mathbf{w}}_i$ que constituyen las filas de la matriz $\mathbf{W}$.

2.2.3 Análisis de componentes principales (PCA)

El Análisis de componentes principales (PCA) es una técnica que puede ser utilizada para obtener una reducción de dimensiones. A diferencia de LDA, que se basa en la *clasificación*, PCA es una proyección orientada a la *representación* de señales. Su objetivo es mapear con precisión un espacio N-dimensional en un subespacio M-dimensional, donde $M < N$, haciendo una proyección preservando la mayor cantidad de información posible. Este procedimiento no utiliza información de las clases para obtener la matriz de transformación $\mathbf{W}$, por lo que se puede ver como entrenamiento sin supervisión [3].

La transformación realizada con PCA analiza un conjunto de muestras de datos del espacio de entrada, y define como el primer componente en el espacio resultante al componente del espacio de entrada en que se presenta mayor *varianza*. El siguiente componente se define como el que tenga la mayor varianza posible y sea ortogonal al componente anterior, y así sucesivamente. Debido a este proceso, la información con la mayor varianza se concentra en los primeros componentes, de ahi que se puede utilizar PCA como una técnica de reducción de dimensiones con pérdidas mínimas de información [13].

Los *componentes principales* corresponden a los eigenvectores de la matriz de covarianza $\mathbf{\Sigma}$ de los datos en el espacio de entradas $\underline{\mathbf{x}}^n$, cuyo vector de media se define como

$$\underline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \underline{\mathbf{x}}^n$$

La matriz de covarianza $\mathbf{\Sigma}$ está dada por

$$\mathbf{\Sigma} = \sum_{n} (\underline{\mathbf{x}}^n - \underline{\mathbf{x}})(\underline{\mathbf{x}}^n - \underline{\mathbf{n}})^T$$

A partir de ésta se obtienen sus eigenvectores y sus eigenvalores. Los eigenvectores correspondientes a los M-eigenvalores más grandes, proyectan los vectores del espacio de entrada en el subespacio de salida reducido [3].

2.3 Canonic Signed-Digit (CSD)

Es una representación numérica la cual consiste en una secuencia de símbolos (+, 0 ó -) según sea suma o resta, en una posición que denota una potencia de 2, con una cantidad mínima de elementos distintos de cero. Por definición, la representación *canonical sign-digit* es un sistema numérico que representa números sin dígitos distintos de cero

adyacentes entre sí, y cada número tiene una única representación CSD posible . Algunos ejemplos de codificación CSD se muestran en la figura 2.2 Un algoritmo para convertir un número en complemento a dos a codificación CSD es descrito en el apéndice C.1.

La representación CSD reduce el número de bits distintos de cero con respecto a la representación binaria. Al no haber dígitos adyacentes distintos de cero, cualquier número de n-bits puede representarse con un máximo de $n/2$ bits distintos de cero [9]. Un ejemplo de esto se demuestra gráficamente en la figura 2.3 Para un número binario de 16 bits, el peor caso de su correspondiente representación CSD (esto es el máximo número de bits distintos de cero) es ocho.

$$1_{10} = 1_2 = 00+.00000_{CSD} = 2^0$$

$$-2_{10} = -10_2 = 0-0.00000_{CSD} = -2^1$$

$$1.5_{10} = 1.1_2 = 0+0.-00000_{CSD} = 2^1 - 2^{-1}$$

$$0.99997_{10} = 0.111111111111111_2 = +.00000000000000-_{CSD} = 2^0 - 2^{-15}$$

Figura 2.2: Algunos ejemplos de codificación CSD

$$0.66666_{10} = 0.+0+0+0+0+0+0+0+_{CSD} = 2^{-1} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-11} + 2^{-13} + 2^{-15}$$

Figura 2.3: Peor caso CSD: ocho dígitos distintos de cero.

2.4 Multiplicadores CSD

Dados los requerimientos de bajo consumo energético del sistema de reconocimiento de patrones acústicos al que pertenece el reductor de dimensiones, debe seleccionarse un diseño para los multiplicadores que reduzca el hardware necesario y el consumo de potencia. Para el caso donde uno de los multiplicandos es constante, los multiplicadores CSD (Canonic-Signed Digit) son la mejor opción para diseños de bajo consumo, ya que son de menor tamaño y más eficientes en el uso de energía que los multiplicadores de propósito general [9]. Para realizar una multiplicación con CSD se sustituye el término constante por una serie de desplazamientos y sumas del multiplicando variable. Los desplazamientos necesarios para obtener el producto son definidos por la codificación CSD del operando constante, como se observa en la figura 2.4.

Un número en CSD tiene menos dígitos distintos de cero que su equivalente en codificación binaria, lo que reduce la cantidad de sumas necesarias para realizar una multiplicación [2].

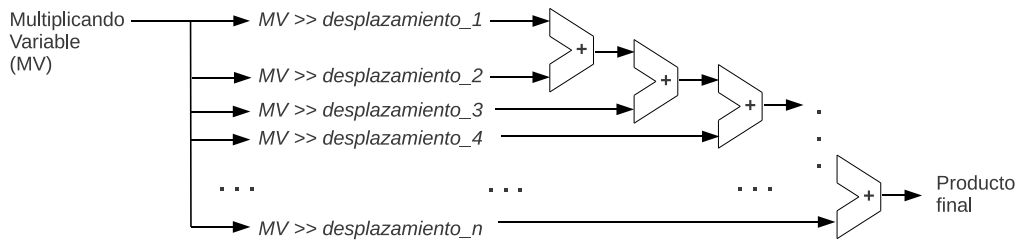


Figura 2.4: Ejemplo de una multiplicación usando la técnica CSD.

Al ser una representación canónica, no hay dígitos distintos de cero adjacentes entre sí, por lo que dicha reducción puede alcanzar en el peor de los casos el 50% en comparación con un multiplicador de propósito general. Esto implica que el sistema debe hacer menos cálculos, tendrá menos conmutación, ocupa menos área y tiene menor consumo de potencia [9].

2.5 Distancia de Bhattacharyya

La distancia de Bhattacharyya μ mide la similitud entre dos distribuciones discretas o contínuas. Para distribuciones guassianas,

$$\mu = \frac{1}{8}(\underline{\mathbf{m}}_1 - \underline{\mathbf{m}}_2)^T \mathbf{P}^{-1}(\underline{\mathbf{m}}_1 - \underline{\mathbf{m}}_2) + \frac{1}{2}\ln\left(\frac{|\mathbf{P}|}{\sqrt{|\mathbf{P_1}||\mathbf{P_2}|}}\right) = \mu_1 + \mu_2$$

donde

$$0 \leq \mu \leq \infty$$

$$\mu_1 = \frac{1}{8}(\underline{\mathbf{m}}_1 - \underline{\mathbf{m}}_2)^T \mathbf{P}^{-1}(\underline{\mathbf{m}}_1 - \underline{\mathbf{m}}_2)$$

$$\mu_2 = \frac{1}{2}\ln\left(\frac{|\mathbf{P}|}{\sqrt{|\mathbf{P_1}||\mathbf{P_2}|}}\right)$$

$\underline{\mathbf{m_i}}$ son los vectores de medias, $\mathbf{P_i}$ son las matrices de covarianza de las distribuciones, $|\mathbf{P_i}|$ sus determinantes y

$$\mathbf{P} = \frac{\mathbf{P_1} + \mathbf{P_2}}{2}$$

Los términos μ_1 y μ_2, miden la diferencia entre las distribuciones debido a la variación de las medias o de las covarianzas respectivamente. Esto significa que el primer y el segundo término de la distancia de Bhattacharyya indican de dónde proviene la separación de clases, de la diferencia entre medias o de la diferencia entre covarianzas [5].

Puesto que los métodos de PCA y LDA parten de un modelo gaussiano de los datos, y utilizan por tanto las matrices de covarianza y medias como únicos parámetros descriptivos de la distribución real, se puede emplear para la medición de la separabilidad de clases ésta distancia, que parte de la misma suposición.

Capítulo 3

Reducción de dimensiones

3.1 Reducción de dimensiones

Para obtener la reducción de dimensiones se realiza una transformación lineal del espacio de entrada de ocho dimensiones mediante un producto matriz-vector a partir del cual se obtiene un espacio tridimensional. El espacio de entrada corresponde a ocho bandas de frecuencia obtenidas a través de las etapa de extracción de características del SiRPA previas al reductor de dimensiones. La figura 3.1 ilustra este proceso. Los rangos de frecuencias correspondientes a cada banda se anotan en la tabla 3.1.

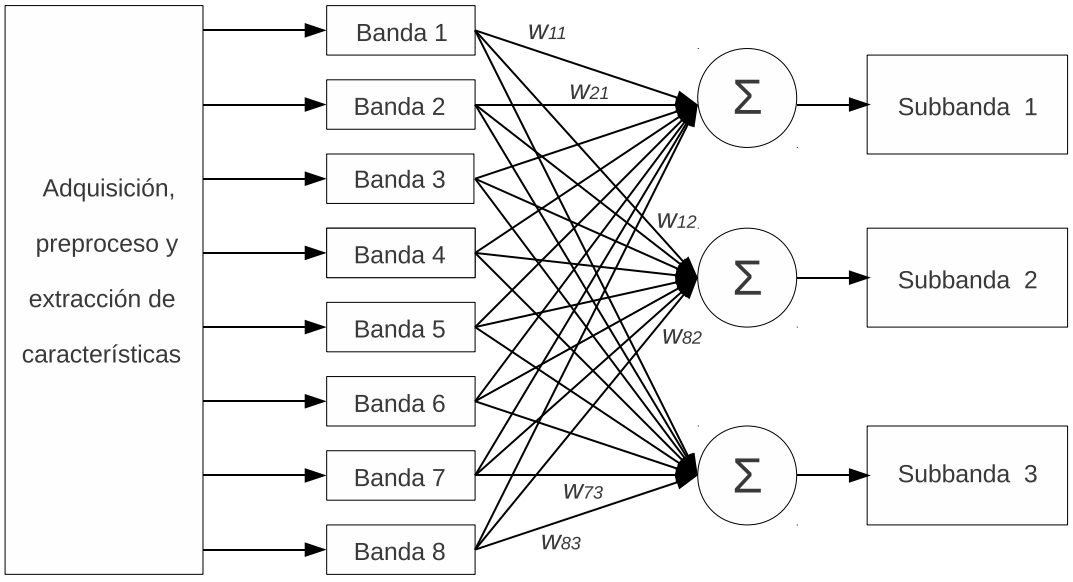


Figura 3.1: Reductor de Dimensiones. Etapas previas descomponen la señal de entrada en ocho bandas mediante filtros digitales. Durante una fase de entrenamiento se obtienen los 24 pesos w_{ij} que conforman la matriz de transformación.

Tabla 3.1: Ancho de banda del banco de filtros de la etapa de extracción de características. Tomado de [12].

Banda	Frecuencias en kHz
Banda 1	20 - 11,025
Banda 2	11,025 - 5,512
Banda 3	5,512 - 2,756
Banda 4	2,756 - 1,378
Banda 5	1,378 - 0,689
Banda 6	0,689 - 0,344
Banda 7	0,344 - 0,172
Banda 8	0,172 - 0,086

La implementación del reductor de dimensiones consta de dos etapas: una de entrenamiento y otra de ejecución.

Durante la fase de entrenamiento se obtiene la matriz de transformación **W** mediante un entrenamiento realizado con herramientas de software en un computador externo al sistema empotrado, a partir de muestras de datos de cada una de las clases que se quieren detectar (en este caso: '*disparo*', '*motosierra*' y '*bosque*'). Para obtener dicha matriz se utilizaron dos técnicas diferentes, LDA y PCA, para poder comparar la capacidad de separación de clases que realiza cada una durante la reducción dimensional.

Durante la fase de ejecución, se utilizan los pesos w_{ij} en la combinación lineal de las salidas octo-dimensionales del banco de filtros del SiRPA, para obtener vectores en el subespacio de tres dimensiones. Esto se realiza multiplicando los 24 coeficientes de la matriz **W** por las bandas de entrada y combinando los productos obtenidos en tres sumadores, como se muestra en la figura 3.1. Los resultados de estas sumas corresponden a las subbandas de salida de tres dimensiones.

3.1.1 Entrenamiento

El entrenamiento utiliza conjuntamente herramientas de software y hardware, y consiste en obtener la matriz de transformación **W** a partir de muestras de datos en ocho dimensiones. Dichos datos son capturados en el FPGA directamente de las salidas de cada una de las bandas espectrales del circuito de extracción de características, y enviados por puerto serial a un computador, donde éstos datos son recibidos y almacenados. Un programa en dicho computador se encarga de reproducir una serie de archivos de audio para los que se conoce su clase ('*motosierra*', '*bosque*' o '*disparo*') y almacenar los datos pertenecientes cada a uno de ellos. Con esto se cuenta con todos los datos necesarios para calcular la matriz **W** según el procedimiento indicado en la sección 2.2.2 para LDA y en la sección 2.2.3 para PCA. Una vez calculada la matriz **W**, otra aplicación de software se encarga de generar el código VHDL que posteriormente se sintetiza para obtener la

implementación de hardware del reductor de dimensiones en FPGA. El diagrama de la figura 3.2 muestra este proceso de interacción entre software y hardware y en las siguientes secciones se explica en detalle cada subproceso.

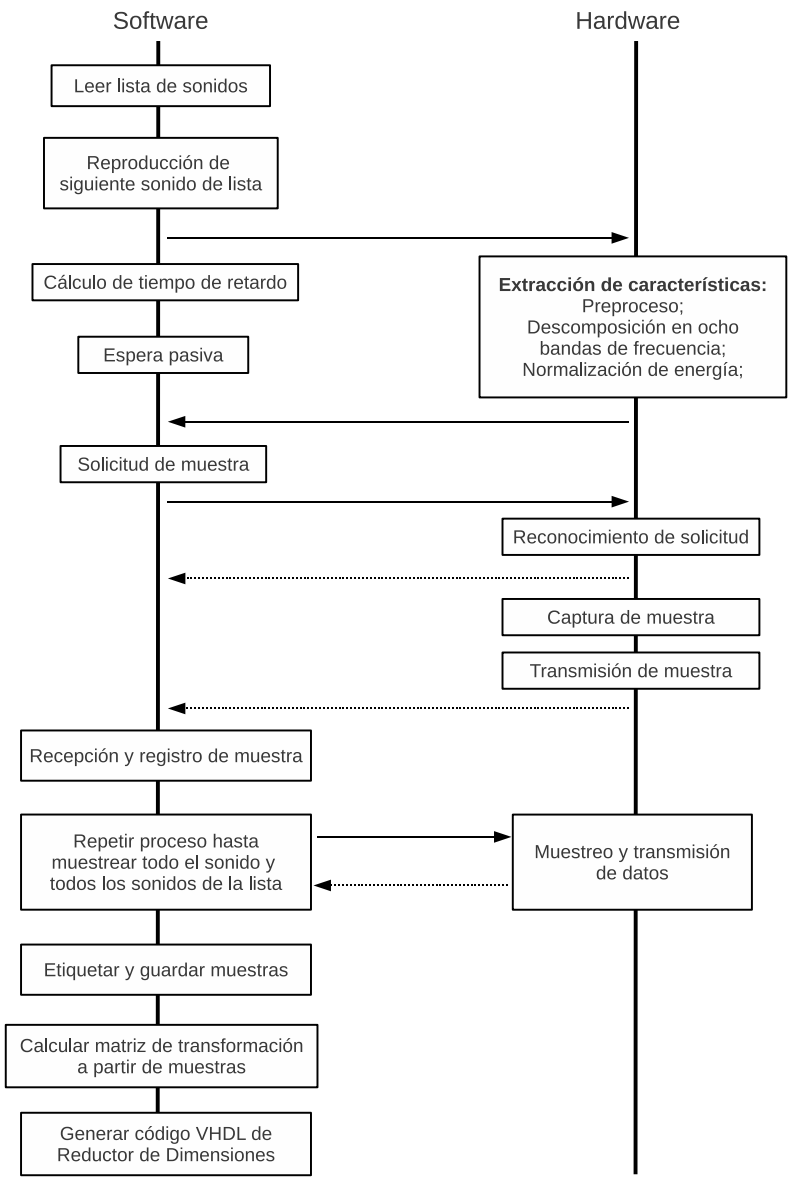


Figura 3.2: Interacción software-hardware durante entrenamiento.

Los sonidos a partir de los cuales se obtienen las muestras de '*disparos*', '*motosierras*' y '*bosque*', están almacenados en un computador y corresponden a grabaciones realizadas en el Parque Nacional Braulio Carrillo el día 14 de agosto del 2006. Los '*disparos*' y '*motosierras*' fueron grabados a distancias que oscilan entre los 30 y los 600 metros y desde ángulos de 0, 90 y 180 grados entre el micrófono y la fuente del sonido. Para los primeros se utilizaron diferentes tipos de armas de fuego, utilizadas comúnmente por los cazadores. Los sonidos de '*bosque*' corresponden a todo lo demás que se esperaría encontrar en una zona boscosa y que no pertencen a '*disparos*' ni '*motosierras*'; por ejemplo: ríos, pájaros,

viento, lluvia, voces humanas, etc.

En la tabla 3.2 se detallan la cantidad de sonidos de cada clase utilizados para obtener las muestras, la duración promedio por clase, su correspondiente desviación estándar y la cantidad de muestras obtenidas.

Tabla 3.2: Sonidos utilizados para entrenamiento.

Clase	Archivos	Duración promedio (s)	Desviación estándar (s)	Muestras
Disparos	44	0.837	0.505	651
Motosierras	11	22.396	6.857	2449
Bosque	81	27.213	28.644	25021

Ya que las muestras necesarias para el entrenamiento deben incluir datos que representen específicamente cada clase, las grabaciones de '*disparos*' y '*motosierras*' fueron editadas de manera que no incluyeran silencios que se puedan confundir con sonidos normales de '*bosque*'. Además, por ésta misma razón, el programa encargado de la reproducción de los sonidos incluye una función para calcular la duración del silencio (o sonido de '*bosque*') presente al inicio de cada grabación de '*disparos*' y '*motosierras*', y a partir de éste, retardar el inicio del proceso de toma de muestras.

Captura de muestras

Para obtener las muestras representativas de '*disparos*', '*motosierras*' y del '*bosque*', se reproducen sonidos de cada una de estas clases en una PC; dicha señal de audio se introduce al circuito de extracción de características implementado en [12], el cual descompone la señal analógica en ocho bandas de frecuencia. El circuito muestreador captura y transmite los datos mediante puerto serial a una PC, en la cual una aplicación de software sincroniza la reproducción de los sonidos, la solicitud de muestras y la recepción de estas desde el hardware. En dicha aplicación, las muestras son registradas y etiquetadas en grupos correspondientes a las clases '*disparo*', '*motosierra*' y '*bosque*', con las que posteriormente se realizará el entrenamiento con LDA (y con PCA) para obtener las correspondientes matrices de transformación. Este proceso se ilustra de manera simplificada en la figura 3.3. En la figura 3.4 se muestra el diagrama de bloques del circuito diseñado para capturar las muestras de datos en ocho dimensiones y transmitirlas a la PC para su almacenamiento. Un diagrama de flujo en la figura 3.5 explica el funcionamiento de este circuito.

El banco de filtro digitales implementado en [12], actualiza los datos de cada banda a diferentes frecuencias, las cuales se resumen en la tabla 3.3. Por este motivo, la frecuencia de muestreo en las salidas de los estimadores de energía en las bandas del banco de filtros debe ser menor que 339 Hz, puesto que de otro modo no se capturaría la variación de todas las bandas, lo que implicaría un despilfarro energético. Por otro lado, esta frecuencia debe ser lo suficientemente alta para capturar la variación de las muestras,

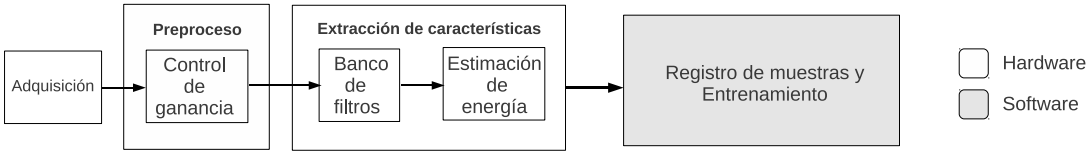


Figura 3.3: Muestreo de las ocho bandas de salida de las etapas de extracción de características

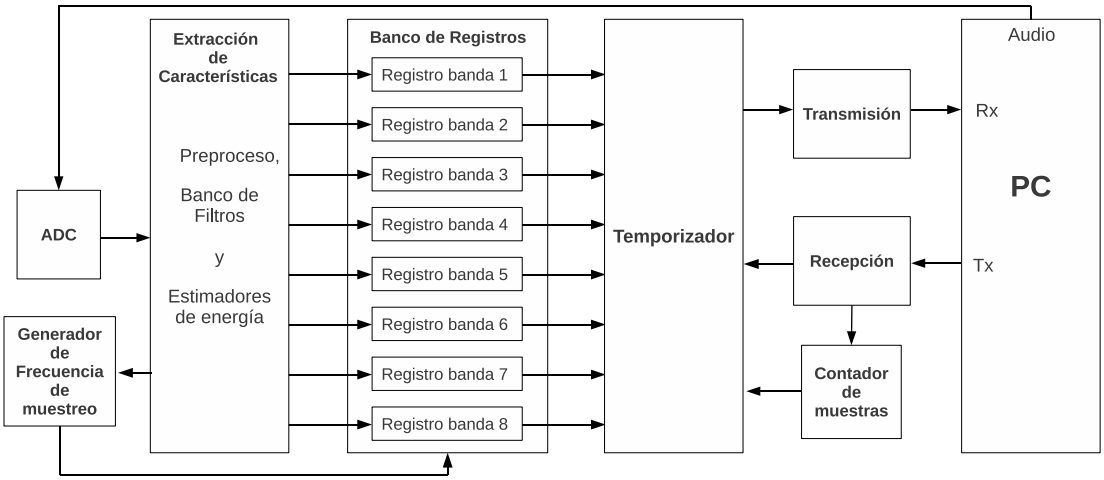


Figura 3.4: Diagrama de bloques del circuito muestreador.

sin perder información, pero lo suficientemente baja para no saturar ni la transmisión de datos, ni su almacenamiento, ni producir exceso de cálculos en el entrenamiento debido a la gran cantidad de muestras obtenidas. Se seleccionó una frecuencia de muestreo de 56 Hz, que cumple con las condiciones establecidas anteriormente.

Tabla 3.3: Frecuencias de refrescamiento de datos por banda.

Banda	Frecuencias de muestreo en kHz
Banda 1	43,4
Banda 2	21,7
Banda 3	10,8
Banda 4	5,4
Banda 5	2,72
Banda 6	1,36
Banda 7	0,675
Banda 8	0,339

El funcionamiento de la aplicación de software diseñada para controlar el proceso de muestreo se muestra en la figura 3.6. Este programa lee una lista donde se indican los sonidos a reproducir y su ubicación en el sistema de archivos de la PC. Carga un sonido a la vez y antes de iniciar su reproducción, una función recorre la señal de audio para

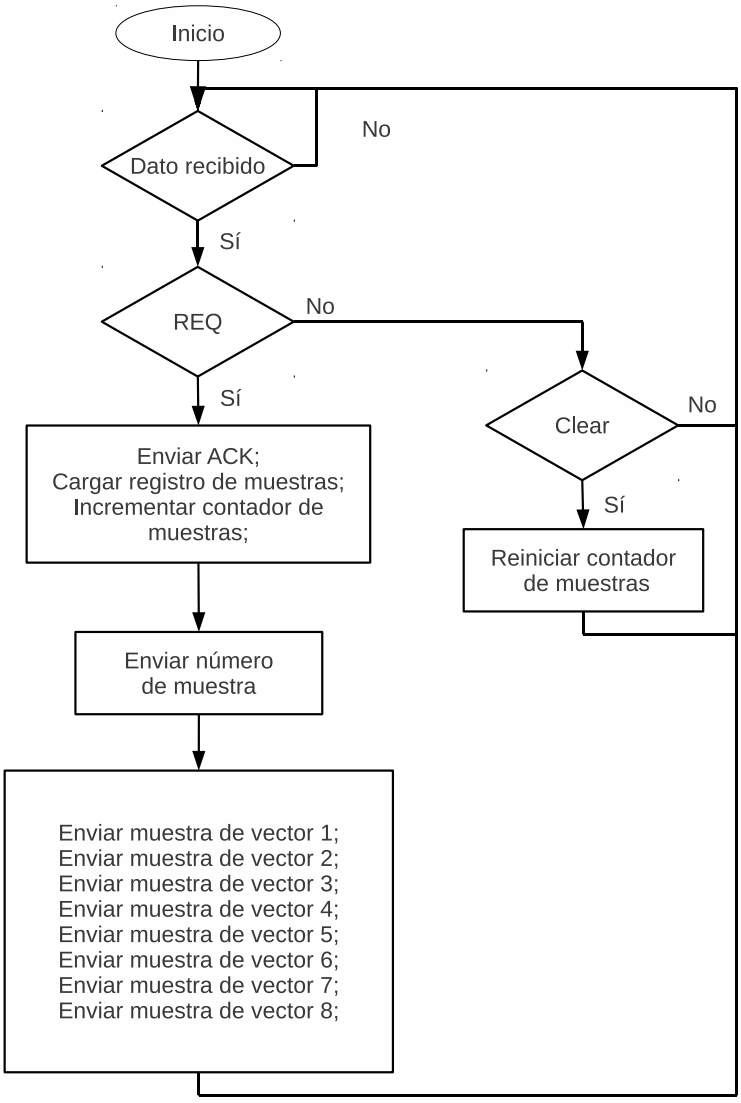


Figura 3.5: Algoritmo de circuito muestreador.

localizar el instante en que se inicia propiamente el '*disparo*' o el sonido '*motosierra*'. Esto lo hace detectando dónde se localiza el primer 'pico' en la señal de audio, y lo considera como el instante donde inicia realmente el sonido deseado. Este proceso previo se realiza únicamente para las grabaciones de estas dos clases, ya que se hace con el propósito de tomar muestras reales de '*disparos*' y '*motosierras*' sin que se filtren muestras de '*bosque*' como parte del conjunto de datos respectivo a cada clase. Una vez calculado este tiempo, se inicia la reproducción del sonido, pero el proceso de muestreo no inicia hasta que haya pasado el tiempo de retardo.

Cumplido el tiempo de espera, el programa envía una solicitud de muestra al circuito, y espera una respuesta de reconocimiento por parte del circuito muestreador. Si se vence el plazo de espera, envía una nueva solicitud. Una vez que se recibe el reconocimiento de parte del circuito, inmediatamente se recibe el número de muestra correspondiente a

ese sonido y los ocho valores enteros muestreados, correspondientes a cada banda. La muestra octo-dimensional se registra en una tabla y se verifica si se ha reproducido todo el sonido. De no ser así, se solicitan, reciben y registran nuevas muestras hasta que se complete la reproducción. Cuando el sonido llega a su fin, se reinicia todo el proceso: se carga un nuevo sonido, se calcula el tiempo a esperar antes de iniciar el muestreo, se solicitan muestras, etc. Esto se repite hasta que se hayan reproducido y muestreado la totalidad de los sonidos indicados en la lista. Cuando se ha recorrido toda la lista de sonidos, las muestras registradas se guardan en un archivo de texto y el programa finaliza su ejecución.

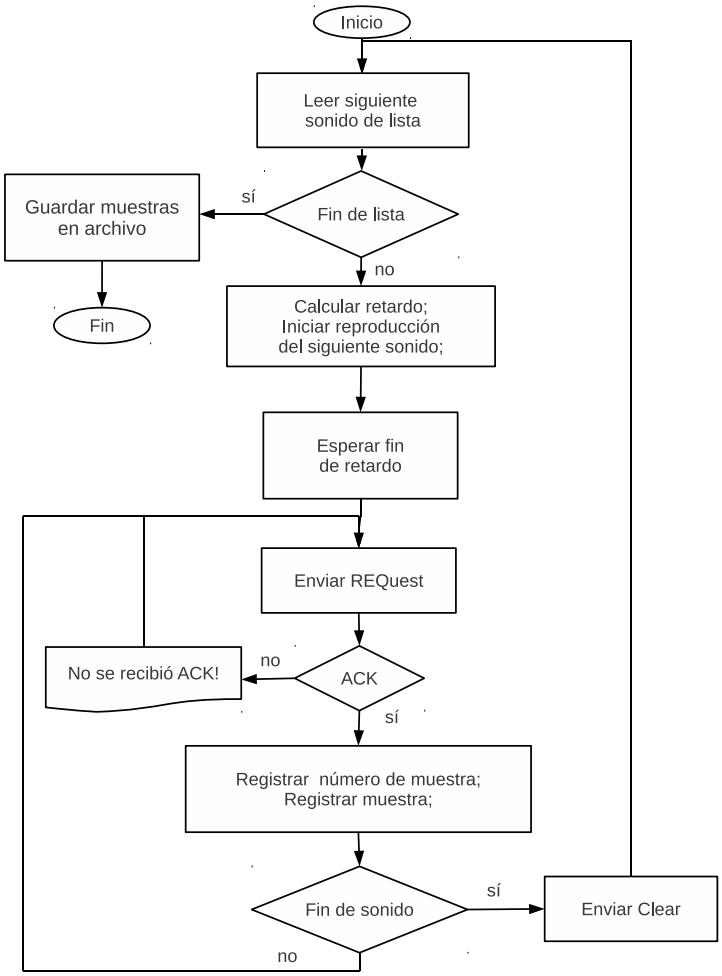


Figura 3.6: Algoritmo de programa muestreador en PC.

Entrenamiento LDA

La transformación realizada con LDA busca proyectar el espacio de entrada a uno de tres dimensiones, maximizando la separación entre las diferentes clases y minimizándola entre miembros de una misma clase. Para lograr esto, las muestras de cada una de las clases

obtenidas mediante el proceso descrito en la sección anterior, son agrupadas en una sola matriz total, en la cual los datos octodimensionales se etiquetan de acuerdo a su clase correspondiente.

A partir de ésta matriz de muestras etiquetadas por clase, se realiza el proceso descrito en la sección 2.2.2 para obtener la matriz de transformación **W**. En la figura 3.7 se muestra éste proceso.

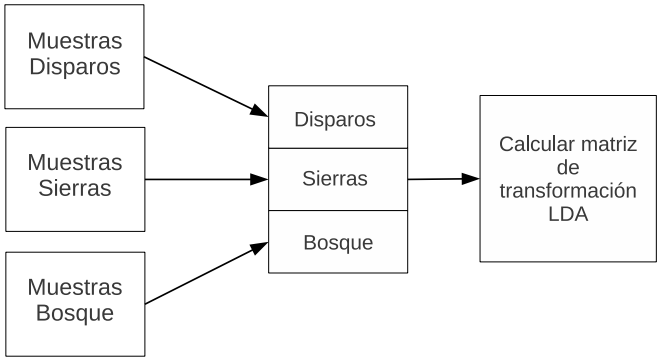


Figura 3.7: Entrenamiento LDA. Las muestras de datos en ocho dimensiones son etiquetadas con su nombre de clase correspondiente y se agrupan en una nueva matriz, a partir de la cual se aplica el proceso descrito en la sección 2.2.2 para obtener la matriz **W**

Entrenamiento PCA

Para obtener la matriz de transformación **W** con PCA, se realiza un proceso de entrenamiento similar al realizado con LDA, a partir de una matriz que contiene las muestras en ocho dimensiones de las tres clases de datos. A diferencia del entrenamiento LDA, en PCA estas muestras no son etiquetadas y agrupadas de acuerdo a su clase, y la matriz **W** se obtiene con el procedimiento descrito en la sección 2.2.3, a partir de una matriz que incluye muestras de todas las clases. Éste proceso se grafica en la figura 3.8.

3.1.2 Ejecución

Restador de media

Las muestras en ocho dimensiones de disparos, motosierras y bosques, utilizadas para el entrenamiento, brindan una referencia del comportamiento esperado de los datos a la salida de la etapa de extracción de características del SiRPA. Durante la etapa de ejecución, el reductor de dimensiones resta a las señales de entrada la media (u offset) de los datos en ocho dimensiones, con el propósito de trasladar los vectores octo-dimensionales del espacio de entrada al origen del sistema de coordenadas. Es necesario hacer esto para contribuir a la estabilidad numérica.

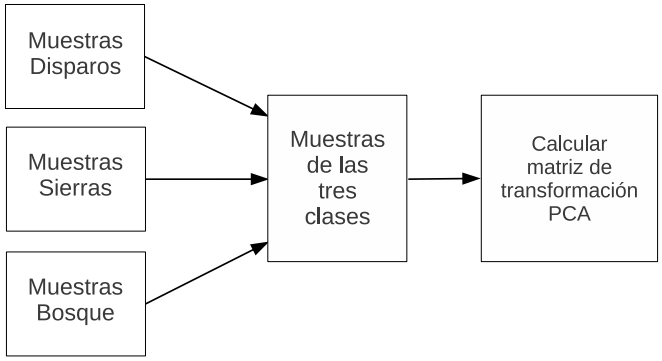


Figura 3.8: Entrenamiento PCA. Las muestras en ocho dimensiones pertenecientes a las tres clases son agrupadas en una sola matriz a partir de la cual se obtienen los coeficientes de la matriz de transformación **W**.

El diagrama del reductor de dimensiones, incluyendo los bloques restadores del valor medio correspondiente a cada banda se muestra en la figura 3.9.

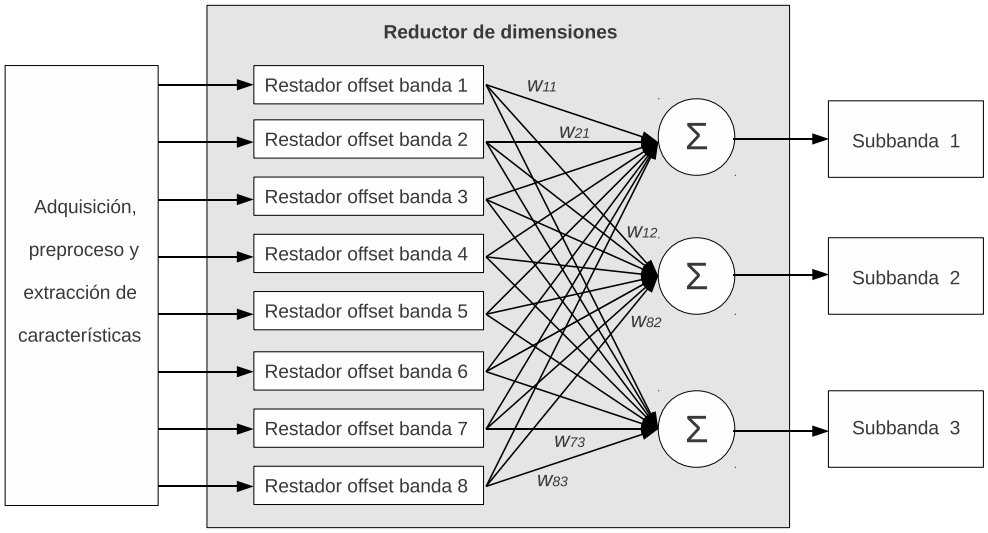


Figura 3.9: Reductor de dimensiones, etapa de ejecución. Incluye un bloque que resta el offset correspondiente a cada banda.

Multiplicadores CSD

Durante la etapa de ejecución del reductor de dimensiones, el espacio de entrada de ocho dimensiones es mapeado por la matriz de transformación para obtener la proyección a un espacio tri-dimensional. Esto implica realizar 24 multiplicaciones entre las bandas de entrada y los coeficientes de la matriz **W**. Las entradas del reductor corresponden a las ocho bandas de frecuencia en que se descomponen las señales de audio a través de las

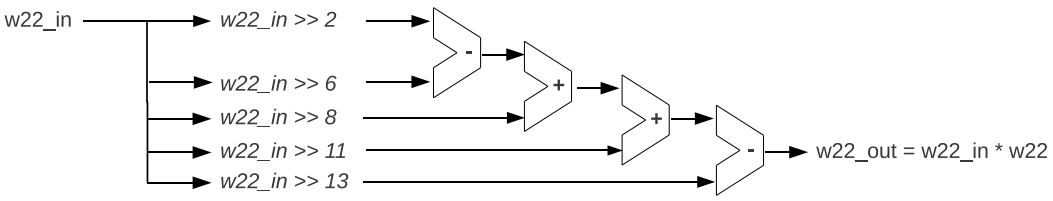
etapas de extracción de características.

Las señales de cada banda tienen representación de 16 bits en complemento a dos y por tanto pueden tomar valores enteros variables en el rango de -2^{15} a $2^{15}-1$. Los coeficientes de $\mathbf{W}$ obtenidos tanto con LDA como con PCA tienen valores reales constantes que oscilan entre $[-1, 1]$.

Para realizar los productos de las señales de entrada con los coeficientes w_{ij}, la implementación de multiplicadores CSD en lugar de multiplicadores binarios normales permite un ahorro de hardware al reducir al menos a la mitad la cantidad de sumadores necesarios para los multiplicadores.

$$w_{22} = 0.2386699307646236_{10} = 0.0+000-0+00+0-00_{CSD} = 2^{-2} - 2^{-6} + 2^{-8} + 2^{-11} - 2^{-13}$$

(a)



(b)

Figura 3.10: Multiplicador CSD. En (a) se obtiene a la codificación CSD del multiplicando constante correspondiente al coeficiente w_{22}. En (b) se observa la cantidad de desplazamientos a la derecha que se deben hacer a la señal de entrada w_{22}_in y las sumas y restas necesarias para obtener el producto de dicha señal por la constante w_{22}.

3.2 Generación automática de código

El reductor de dimensiones se implementó utilizando hardware reconfigurable. Éste debe acoplarse con los demás módulos del sistema de reconocimiento de patrones diseñados en [12], los cuales fueron implementados utilizando el lenguaje de descripción de hardware VHDL.

Para facilitar la implementación del reductor y los 24 multiplicadores que lo componen, o su eventual rediseño, se desarrolló un programa de C++ para generar código VHDL automáticamente, el cual puede ser sintetizado posteriormente para diferentes tecnologías de FPGA. El diagrama de flujo del programa generador de código se muestra en la figura 3.11. Éste crea un archivo VHDL correspondiente al módulo principal del reductor de dimensiones, otro correspondiente al restador de offset y un módulo para cada uno

de los 24 módulos multiplicadores correspondientes a cada coeficiente de la matriz de transformación **W** obtenida durante la fase de entrenamiento.

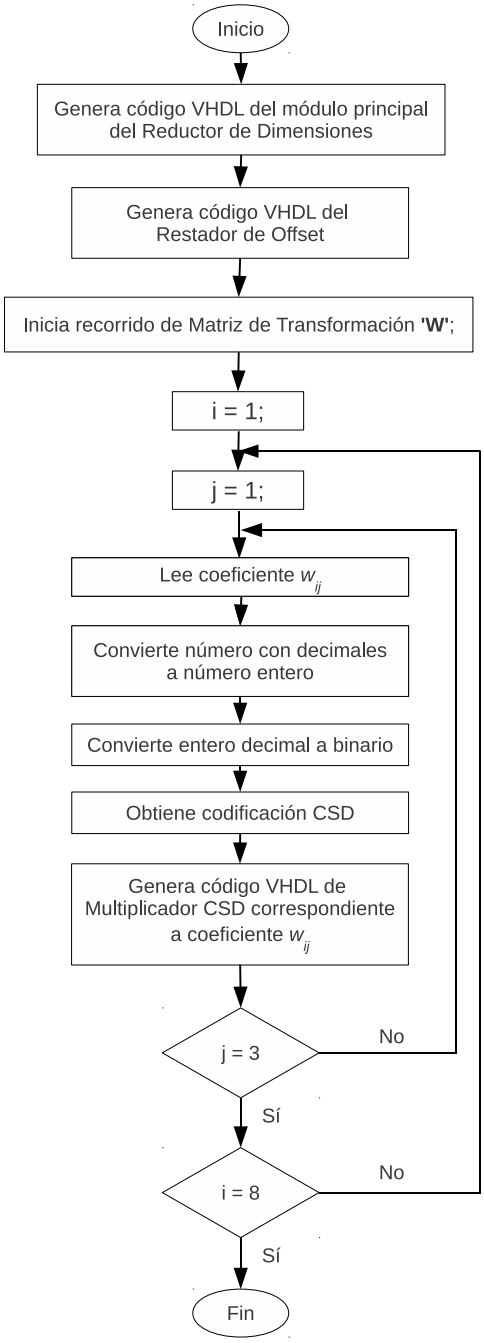


Figura 3.11: Generación automática de código

El código VHDL correspondiente al reductor de dimensiones es fijo, es decir, no depende de ningún parámetro variable, como sí lo hacen los módulos del restador de offset y los multiplicadores. El código generado completo de este módulo se muestra en el apéndice D.1. Una sección de este código se muestra en las figuras 3.12 y 3.13, donde se aprecian la definición de las ocho entradas octo-dimensionales, las salidas de tres di-

mensiones, el bloque restador de offset, los dos primeros multiplicadores y la combinación lineal de los productos obtenidos, con la que se obtiene la reducción dimensional.

Después de haber generado el módulo principal del reductor de dimensiones, el programa generador de código crea el módulo VHDL del restador de offset. El diagrama de flujo de este proceso se muestra graficamente en la figura 3.14a. En primera instancia, en el archivo VHDL se imprime la parte fija del código, lo que corresponde a las primeras 36 líneas del código mostrado en la figura 3.15. Luego se lee el vector del offset obtenido durante el entrenamiento a partir de las muestras de datos en ocho dimensiones, y a cada entrada correspondiente a una dimensión, se le resta el valor correspondiente del vector de offset, como se observa en las líneas 37-44.

Posteriormente el generador de código procede a crear los 24 multiplicadores CSD que conforman el reductor de dimensiones. El programa recorre la matriz $\mathbf{W}$ y para cada coeficiente genera un multiplicador CSD correspondiente. Para hacer esto, primero convierte cada coeficiente, que es un número real (con decimales), a un número entero en representación decimal. Éste se codifica en binario y luego en CSD. A partir del número en codificación CSD, se calculan los desplazamientos y sumas que se deben hacer a la señal de entrada para obtener el producto por el multiplicando constante.

El archivo de texto con el código VHDL de cada multiplicador CSD correspondiente a un coeficiente w_{ij}, se crea siguiendo el procedimiento mostrado en la figura 3.14b. De manera similar que con los códigos generados anteriormente, el generador imprime la primera parte fija del código mostrado en la figura 3.16, lo que correspondería a las líneas 1-17, incluyendo el nombre del coeficiente w_{ij} correspondiente. Luego calcula las variables que se utilizarán dependiendo de la cantidad de desplazamientos y sumas que se deben hacer a la señal de entrada para obtener el producto deseado. Se imprimen las variables (línea 17), los desplazamientos (líneas 28-33) y la suma de desplazamientos (línea 35). Este proceso se realiza para cada uno de los 24 coeficientes de la matriz $\mathbf{W}$, como se muestra en la figura 3.11. El hardware de cada multiplicador generado recibe una entrada de 16 bits en complemento a dos y realiza los desplazamientos y sumas necesarios para obtener el producto del valor de entrada por el coeficiente w_{ij} correspondiente.

Finalmente, los módulos generados pueden ser sintetizados para diferentes tecnologías de FPGA y de esta forma obtener la implementación en hardware del reductor de dimensiones. Así, en caso de ser necesario un rediseño debido a la obtención de una nueva matriz de transformación $\mathbf{W}$, a partir de un conjunto de muestras distinto al utilizado para la realización de este proyecto, la nueva implementación del reductor de dimensiones se puede realizar en gran parte de manera automática gracias al generador automático de código VHDL diseñado.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
LIBRARY work;
USE work.ALL;
use work.def_type.all;


entity Reductor_de_dimensiones is
    generic (n_bits : integer := 16);
    port (
      dimension_in_1:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_2:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_3:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_4:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_5:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_6:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_7:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_in_8:     IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_out_1:    OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_out_2:    OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
      dimension_out_3:    OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1
    );
end Reductor_de_dimensiones;

[...]

begin
--sub modulos--

--Restador de offset
  Restador_de_offset: entity work.Restador_de_offset
    generic map(n_bits => n_bits)
    port map(
      dimension_in_1 => dimension_in_1,
      dimension_in_2 => dimension_in_2,
      dimension_in_3 => dimension_in_3,
      dimension_in_4 => dimension_in_4,
      dimension_in_5 => dimension_in_5,
      dimension_in_6 => dimension_in_6,
      dimension_in_7 => dimension_in_7,
      dimension_in_8 => dimension_in_8,
      dimension_1 => dimension_1,
      dimension_2 => dimension_2,
      dimension_3 => dimension_3,
      dimension_4 => dimension_4,
      dimension_5 => dimension_5,
      dimension_6 => dimension_6,
      dimension_7 => dimension_7,
      dimension_8 => dimension_8
    );

[...]
```

Figura 3.12: Algunas secciones del código VHDL del Reductor de Dimensiones generado automáticamente (primera parte)

```
[...]

--Implementaciones de multiplicadores
-------------------------------------------------
  coeficiente_w11: entity work.coeficiente_w11(Behavioral)
  generic map(n_bits => n_bits_multiplicadores)
  port map(w11_in => dimension_1,
           w11_out => w11_out);

  coeficiente_w12: entity work.coeficiente_w12(Behavioral)
  generic map(n_bits => n_bits_multiplicadores)
  port map(w12_in => dimension_1,
           w12_out => w12_out);

[...]

-------------------------------------------------
--  Suma de productos
-------------------------------------------------

  dimension_out_1 <= ( w11_out + w21_out + w31_out  + w41_out
                     + w51_out + w61_out  + w71_out + w81_out ) ;

  dimension_out_2 <= ( w12_out + w22_out + w32_out  + w42_out
                     + w52_out + w62_out + w72_out  + w82_out ) ;

  dimension_out_3 <= (  w13_out + w23_out + w33_out  + w43_out
                     + w53_out + w63_out + w73_out + w83_out ) ;

end Behavioral;
```

Figura 3.13: Algunas secciones del código VHDL del Reductor de Dimensiones generado automáticamente (segunda parte)

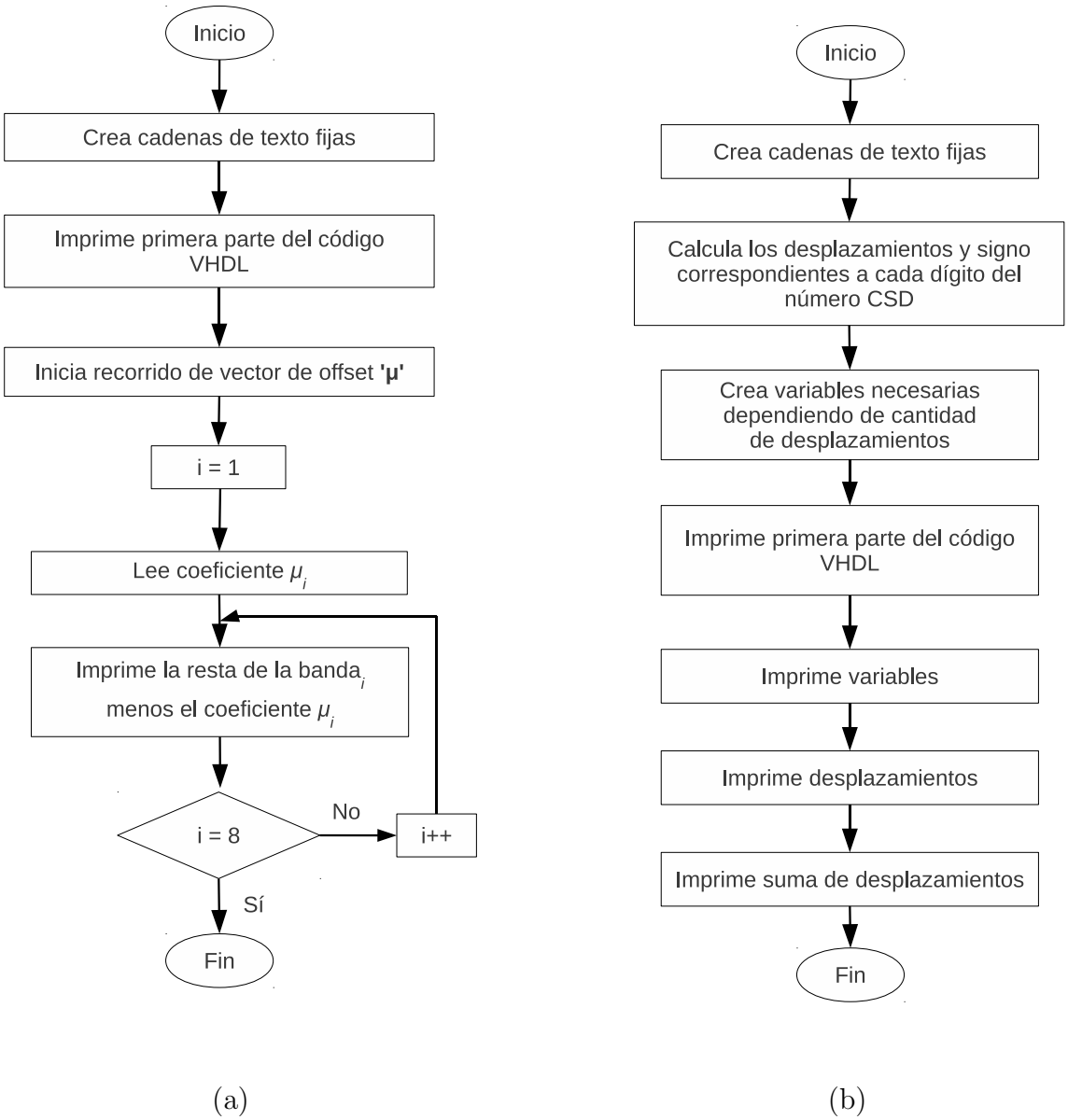


Figura 3.14: Generación automática de código (a) Restador de offset. (b) Multiplicador CSD.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  LIBRARY work;
8  USE work.ALL;
9  use work.def_type.all;
10
11
12 entity Restador_de_offset is
13     generic (n_bits : integer := 16);
14     port (
15       dimension_in_1:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
16       dimension_in_2:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
17       dimension_in_3:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
18       dimension_in_4:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
19       dimension_in_5:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
20       dimension_in_6:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
21       dimension_in_7:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
22       dimension_in_8:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
23       dimension_1:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
24       dimension_2:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
25       dimension_3:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
26       dimension_4:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
27       dimension_5:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
28       dimension_6:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
29       dimension_7:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
30       dimension_8:        OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1
31       );
32 end Restador_de_offset;
33
34 architecture Behavioral of Restador_de_offset is
35 begin
36
37         dimension_1  <=   dimension_in_1 - 3973;
38         dimension_2  <=   dimension_in_2 - 3285;
39         dimension_3  <=   dimension_in_3 - 3027;
40         dimension_4  <=   dimension_in_4 - 2767;
41         dimension_5  <=   dimension_in_5 - 2756;
42         dimension_6  <=   dimension_in_6 - 2571;
43         dimension_7  <=   dimension_in_7 - 1881;
44         dimension_8  <=   dimension_in_8 - 1612;
45
46 end Behavioral;
```

Figura 3.15: Código VHDL del Restador de Offset generado automáticamente

```
1
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  USE ieee.std_logic_arith.ALL;
5  USE ieee.std_logic_signed.ALL;
6  use IEEE.numeric_std.all;
7
8  entity coeficiente_w11 is
9   generic (n_bits : integer := 19);
10   PORT (
11      w11_in:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
12      w11_out:     OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1
13    );
14  end coeficiente_w11 ;
15
16  architecture Behavioral of coeficiente_w11 is
17    signal w11_v_in:  bit_vector(n_bits-1 downto 0);
18          signal a_v, b_v, c_v, d_v, e_v, f_v:   std_logic_vector (n_bits-1 downto 0);
19    signal partial :          std_logic_vector (n_bits downto 0);
20
21 --      w11 = 0.448921      Codificacion CSD: 00.+00-0+0-000-0-0
22
23
24  begin
25
26    w11_v_in <= To_bitvector(CONV_STD_LOGIC_VECTOR(w11_in, n_bits));
27
28          a_v <= To_StdLogicVector(w11_v_in sra 1);
29          b_v <= To_StdLogicVector(w11_v_in sra 4);
30          c_v <= To_StdLogicVector(w11_v_in sra 6);
31          d_v <= To_StdLogicVector(w11_v_in sra 8);
32          e_v <= To_StdLogicVector(w11_v_in sra 12);
33          f_v <= To_StdLogicVector(w11_v_in sra 14);
34
35          partial <=  +(a_v(n_bits-1) & a_v) -b_v +c_v -d_v -e_v -f_v ;
36
37      w11_out <= CONV_INTEGER(partial(n_bits-1 downto 0));
38
39 end Behavioral;
```

Figura 3.16: Código VHDL de un multiplicador CSD generado automáticamente

Capítulo 4

Resultados y Análisis

4.1 Reducción de dimensiones

El vector de media (u offset) de los datos octodimensionales obtenido a partir de todos las muestras de las tres clases obtenidas durante la etapa de entrenamiento es

$$\underline{\mu}^T = [3973\ 3285\ 3027\ 2767\ 2756\ 2571\ 1881\ 1612]$$

y cada uno de estos valores se resta a su banda correspondiente durante la ejecución del reductor de dimensiones. Las componentes del vector están ordenadas de mayor a menor frecuencia. Una mayor variación de las bandas de alta frecuencia produce que el valor promedio sea superior. Se observa un comportamiento monótonamente creciente en dicho valor en función de la frecuencia.

4.1.1 LDA

La matriz de transformación LDA obtenida con el entrenamiento es

$$\mathbf{W}^T = \begin{bmatrix} -0.2464741953056455 & -0.3555502486411402 & -0.5306721935663081 \\ -0.1518257925170513 & 0.2386699307646236 & 0.06560739882374836 \\ -0.1928240011636543 & 0.09450822295660272 & 0.1414011819189337 \\ -0.185215429819623 & -0.212719791093454 & 0.2203161467779507 \\ 0.6898374160150471 & -0.556174540152289 & -0.4451596912224341 \\ -0.3794124749407183 & -0.2304713597493061 & 0.3941648276280388 \\ 0.1925034214615343 & -0.290196581935068 & -0.08664246573236879 \\ 0.4333895418184657 & 0.5619412319501699 & -0.5333940518223561 \end{bmatrix}$$

De ésta se deduce que para obtener la primera banda del espacio tridimensional resultante, las bandas cinco, seis y ocho del espacio de entrada son las más importantes. Para la

segunda banda del espacio tridimensional, son las bandas uno, cinco y ocho; y para la tercera la uno, cinco, seis y ocho las que tienen más peso. Esto se resume en la tabla 4.1, y se puede concluir que la quinta dimensión del espacio de entrada es la más relevante para la trasformación con LDA. De acuerdo a la tabla 3.1, donde se enlista el rango de frecuencias correspondiente a cada banda, ésta corresponde a las frecuencias de 689 a 344 Hz.

Tabla 4.1: Bandas relevantes para la trasformación con LDA.

Banda del espacio tridimensional	Bandas del espacio octodimensional
Primera	5, 6 y 8
Segunda	1, 5 y 8
Tercera	1, 5, 6 y 8

En la figura 4.1 se muestra una gráfica en tres dimensiones con los resultados de la proyección de las clases, a partir de la reducción del espacio vectorial de entrada de ocho dimensiones utilizando LDA. Se observa cómo las muestras correspondientes a cada una de las clases: '*disparos*', '*motosierras*' y '*bosque*', ocupan distintas regiones del espacio tri-dimensional con poco o ningún traslape. Este conjunto de muestras visto desde diferentes perspectivas se muestra en el apéndice A.1, y permite apreciar los resultados de la separación de clases obtenida en el espacio de tres dimensiones. En el apéndice A.2 se muestran gráficas con los resultados de la reducción de dimensiones para tres conjuntos de muestras diferentes, donde se puede observar la similitud de los patrones tridimensionales obtenidos después de la reducción.

4.1.2 PCA

La matriz de transformación PCA obtenida es

$$\mathbf{W}^T = \begin{bmatrix} 0.4489206740373123 & 0.4665375609133601 & 0.6223651402401669 \\ 0.5787229673912969 & -0.03992915339565018 & 0.1589659807376512 \\ 0.5002682970906582 & -0.3613921009670321 & -0.2504716681166103 \\ 0.2999747313752874 & -0.4218201664974374 & -0.140792011579691 \\ 0.07578326618235576 & -0.4631298129721954 & 0.1883124344573798 \\ -0.2274396467805563 & -0.4527198665450621 & 0.5956275890400133 \\ -0.1781229652573134 & -0.2153266441247524 & 0.2985026428769103 \\ -0.1846538918762513 & -0.08001124022636517 & 0.1596727169143085 \end{bmatrix}$$

En la tabla 4.2 se indican cuáles bandas del espacio de entrada tienen más peso para obtener la trasformación con PCA.

Para la primera banda del espacio resultante, las primeras tres bandas tienen mayores pesos. La segunda banda se ve influenciada principalmente por las bandas uno, cuatro,

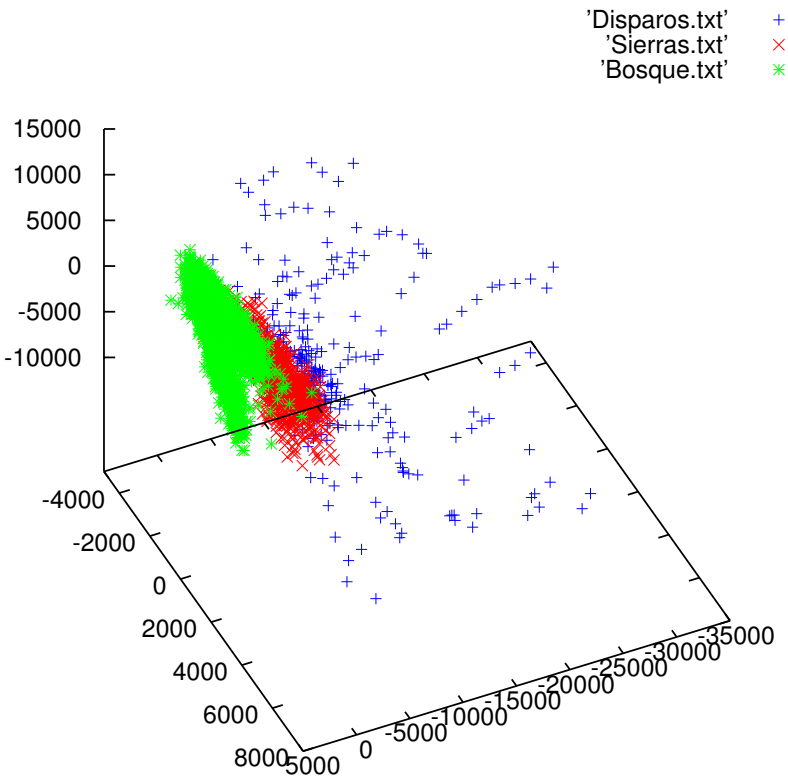


Figura 4.1: Muestras de datos en tres dimensiones luego de la reducción utilizando LDA

cinco y seis. Y para la tercera banda de la proyección tridimensional, las bandas uno, seis y siete son las determinantes.

La primera dimensión del espacio de entrada, correspondiente a las frecuencias entre 20 y 11,025 kHz de acuerdo a la tabla 3.1, es la más influyente para la proyección con PCA.

Tabla 4.2: Bandas relevantes para la trasformación con PCA.

Banda del espacio tridimensional	Bandas del espacio octodimensional
Primera	1, 2 y 3
Segunda	1, 4, 5 y 6
Tercera	1, 6 y 7

En la figura 4.2 se muestran los resultados de la reducción del espacio vectorial de entrada utilizando PCA. Estas distribuciones vistas desde diferentes perspectivas se muestran en el apéndice B.1, y en B.2 se muestran gráficas con los resultados de la reducción de dimensiones para diferentes conjuntos de muestras; nuevamente se puede observar la similitud de los patrones tridimensionales obtenidos con la reducción para distintos conjuntos de datos. En dicha proyección no se presenta una separación marcada entre las

muestras de las distintas clases. Particularmente, se puede apreciar el traslape existente entre las muestras correspondientes a '*motosierras*' y '*bosque*'. Este comportamiento se debe a la similitud de los patrones característicos de estas clases, y demuestra que PCA no es una técnica de reducción enfocada a la *clasificación*, sino a la *representación*, por lo que no se realiza una separación visualmente tan clara como la obtenida con LDA. En la siguiente sección se muestran resultados que complementan esta afirmación.

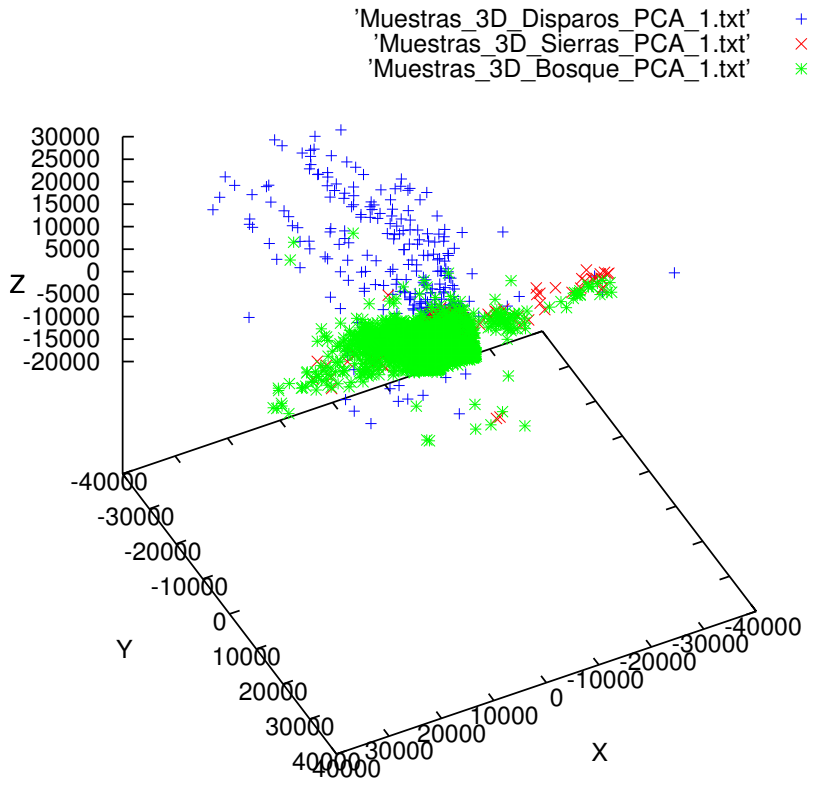


Figura 4.2: Muestras de datos en tres dimensiones luego de la reducción utilizando PCA

4.2 LDA contra PCA

Para comparar los resultados de la reducción realizada mediante ambas técnicas, se calculó la distancia de Bhattacharyya entre las distribuciones tridimensionales obtenidas tanto con LDA como con PCA. Esta distancia se utiliza para cuantificar la separación entre dos distribuciones obtenidas en tres dimensiones luego de la proyección. En la tabla 4.3 se resumen los resultados obtenidos utilizando las ecuaciones descritas en la seccion 2.5 y los vectores de media y matrices de covarianza de las distribuciones tridimensionales, detallados en la sección 4.2.1. Las distancias de Bhattacharyya obtenidas demuestran una mayor separación entre clases con LDA en comparación con PCA.

En ambos casos, la mayor separación se presenta en la comparación entre las distribuciones de '*disparos*'-'*motosierras*' y entre '*disparos*'-'*bosque*'. Esto se puede comprobar gráficamente en las figuras 4.3 y 4.4, donde se observa que dichas distribuciones ocupan distintas regiones del espacio tridimensional, y existe poco o ningún traslape.

De acuerdo a la tabla 4.3, la menor separación entre clases se presentó entre '*motosierras*' y '*bosque*', tanto para LDA y PCA. En la figura 4.5 se aprecia que los mejores resultados en el proceso de separación se obtienen con LDA. Particularmente en la figura 4.5b, correspondiente a las distribuciones obtenidas con PCA, se observa un alto grado de traslape entre ambas distribuciones y no se distingue ninguna separación entre ellas en el espacio de tres dimensiones.

En las tablas 4.4 y 4.5 se deglosan los resultados del primer y segundo término de la distancia de Bhattacharyya para LDA y PCA respectivamente. Para las distribuciones obtenidas con PCA, la separación entre clases se debe principalmente a la diferencia entre sus matrices de covarianza. De la misma manera con LDA, para las comparaciones entre '*disparos*'-'*motosierras*' y '*disparos*'-'*bosque*'. Sin embargo, para el caso '*motosierras*'-'*bosque*', la separación entre clases es debida a la diferencia entre medias de dichas distribuciones.

Tabla 4.3: Separación de clases de LDA contra PCA

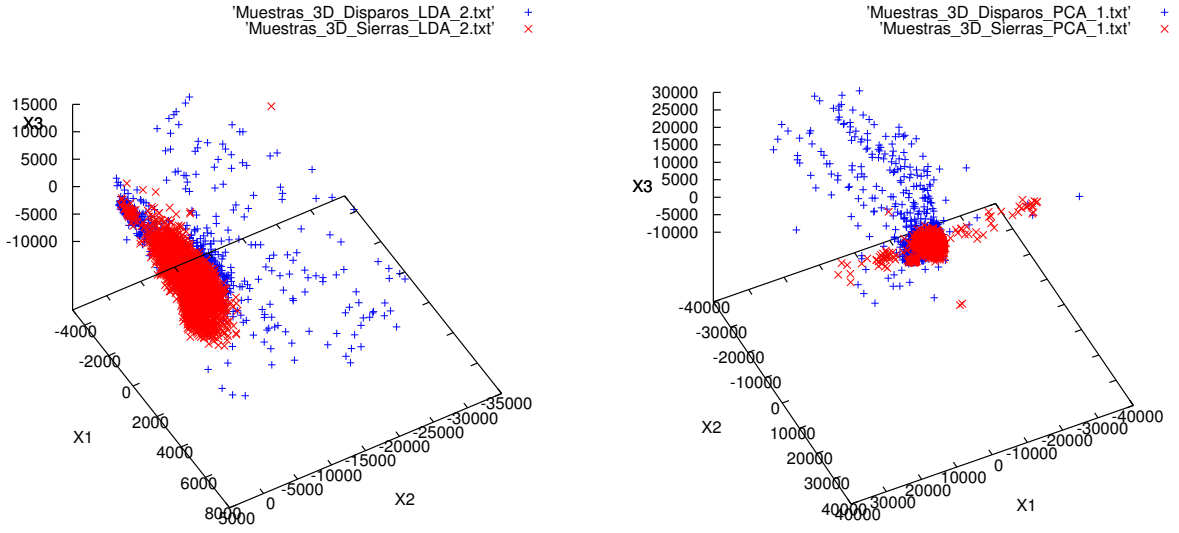
Distribuciones	LDA	PCA
Disparos-Sierras	0,7430	0,6455
Disparos-Bosque	0,5053	0,3684
Sierras-Bosque	0,3826	0,1996

Tabla 4.4: Distancia de Bhattacharyya para distribuciones LDA,

Distribuciones	Primera parte	Segunda parte	Total
Disparos-Sierras	0,2499	0,4931	0,7430
Disparos-Bosque	0,0946	0,4106	0,5053
Sierras-Bosque	0,3195	0,0631	0,3826

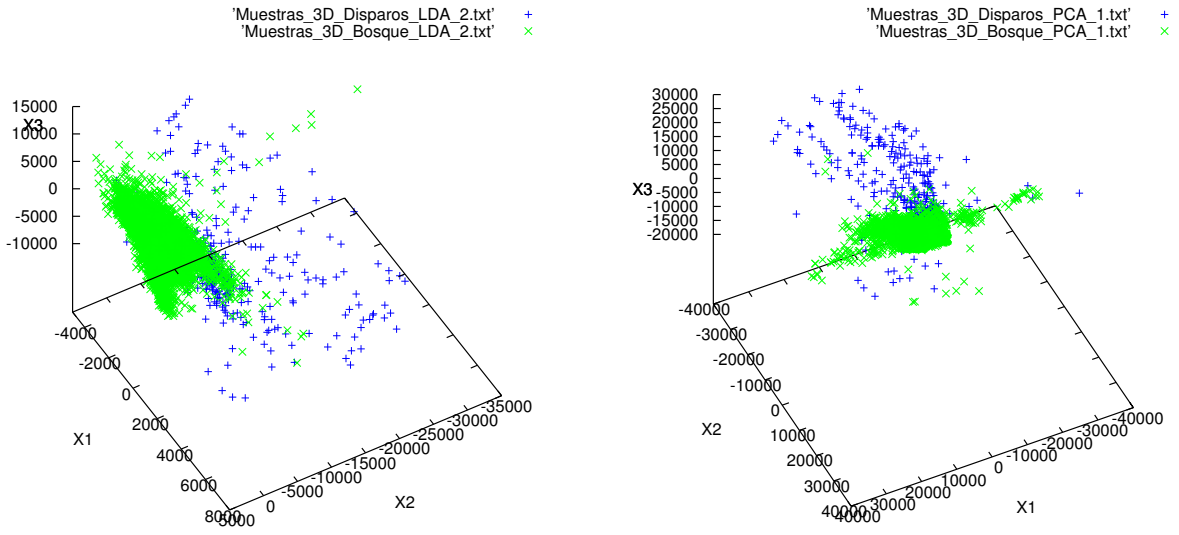
Tabla 4.5: Distancia de Bhattacharyya para distribuciones PCA.

Distribuciones	Primera parte	Segunda parte	Total
Disparos-Sierras	0,2218	0,4236	0,6455
Disparos-Bosque	0,1386	0,2297	0,3684
Sierras-Bosque	0,0870	0,1126	0,1996



(a) (b)

Figura 4.3: Distribuciones de Disparos y Sierras. (a) LDA. (b) PCA.

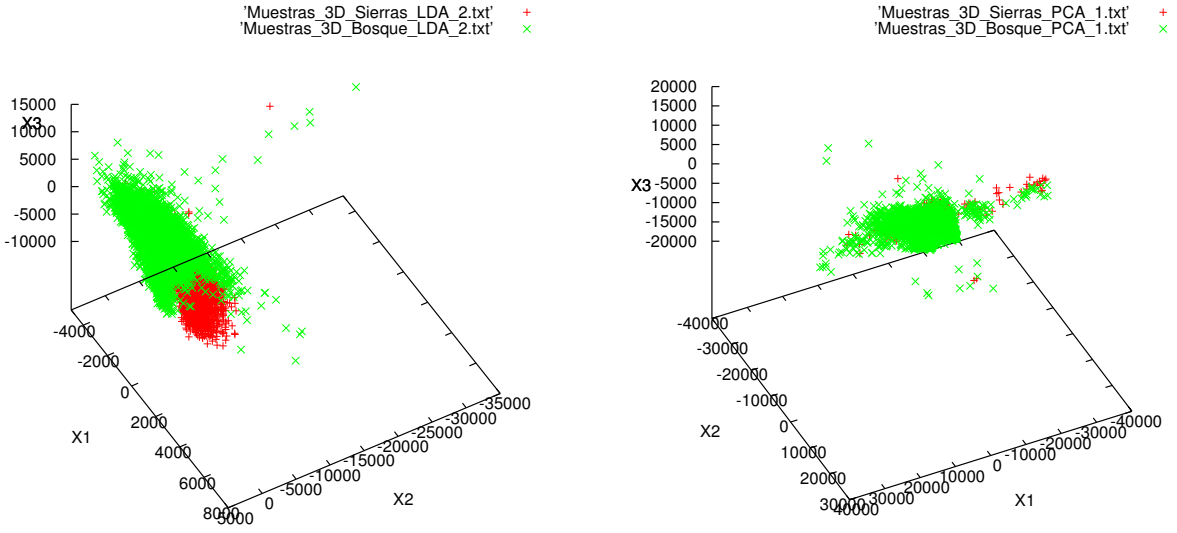


(a) (b)

Figura 4.4: Distribuciones de Disparos y Bosque. (a) LDA. (b) PCA.

4.2.1 Matrices de covarianza y vectores de media de las distribuciones tridimensionales.

Los resultados de las distancias de Bhattacharyya resumidos en las Tablas 4.4 y 4.5 fueron obtenidos de acuerdo a las ecuaciones descritas en la seccion 2.5, utilizando las correspondientes matrices de covarianza y vectores de media detallados a continuación. Aunque



(a) (b)

Figura 4.5: Distribuciones de Sierras y Bosque. (a) LDA. (b) PCA.

para llevar a cabo la transformación PCA no se utiliza información relevante a las clases, para comparar los resultados de la proyección realizada se analizan las distribuciones obtenidas luego de la reducción dimensional.

Distribuciones LDA

$$\underline{\textbf{mediaDisparos}} = [-845.028 \ -2849.03 \ -1230.32]$$

$$\textbf{covarianzaDisparos} = \begin{bmatrix} 9.97378e+06 & -5.11432e+06 & -865379 \\ -5.11432e+06 & 1.52483e+07 & -2.69753e+06 \\ -865379 & -2.69753e+06 & 4.69605e+06 \end{bmatrix}$$

$$\underline{\textbf{mediaSierras}} = [796.333 \ -87.1015 \ -1203.11]$$

$$\textbf{covarianzaSierras} = \begin{bmatrix} 3.3286e+06 & 892480 & 435247 \\ 892480 & 1.5958e+06 & -343183 \\ 435247 & -343183 & 1.51606e+06 \end{bmatrix}$$

$$\underline{\textbf{mediaBosque}} = [-1766.52 \ -439.769 \ -1094.51]$$

$$\textbf{covarianzaBosque} = \begin{bmatrix} 2.19631e+06 & 82811.6 & 514180 \\ 82811.6 & 1.83659e+06 & 132808 \\ 514180 & 132808 & 2.3378e+06 \end{bmatrix}$$

Distribuciones PCA

$$\underline{\textbf{mediaDisparos}} = [3944.15 \ -1087.73 \ 3421.52]$$

$$\textbf{covarianzaDisparos} = \begin{bmatrix} 1.77733e+07 & 4.67355e+06 & 1.57981e+06 \\ 4.67355e+06 & 3.75469e+07 & -9.85375e+06 \\ 1.57981e+06 & -9.85375e+06 & 9.02126e+06 \end{bmatrix}$$

$$\underline{\textbf{mediaSierras}} = [-219.097 \ 387.838 \ 938.38]$$

$$\textbf{covarianzaSierras} = \begin{bmatrix} 1.87021e+07 & 3.49614e+06 & 2.3975e+06 \\ 3.49614e+06 & 5.50028e+06 & 1.17472e+06 \\ 2.3975e+06 & 1.17472e+06 & 1.59018e+06 \end{bmatrix}$$

$$\underline{\textbf{mediaBosque}} = [3063.55 \ 1296.21 \ 828.711]$$

$$\textbf{covarianzaBosque} = \begin{bmatrix} 2.09519e+07 & -697076 & 220682 \\ -697076 & 9.13536e+06 & 1.63082e+06 \\ 220682 & 1.63082e+06 & 3.57677e+06 \end{bmatrix}$$

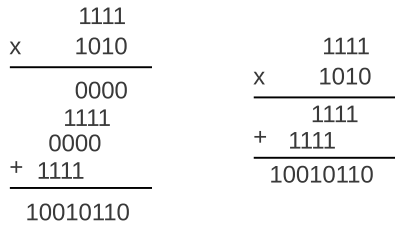
4.3 Uso de recursos de hardware.

En la Tabla 4.6 se presentan los recursos de hardware requeridos en la implementación del reductor de dimensiones utilizando LDA, para el que se utilizó el 20% de LUTs (Look-Up Tables) disponibles en un FPGA Spartan 3E de Xilinx.

Tabla 4.6: Recursos del FPGA Spartan 3E utilizados por el Reductor de Dimensiones

Recursos	Utilizados	Disponibles	Utilización
Slices	987	4656	21%
LUTs	1917	9312	20%

En la Tabla 4.7 se presenta el desglose de los recursos de hardware utilizados en la implementación de los 24 multiplicadores CSD de 16 bits del reductor de dimensiones, y la cantidad de sumas o restas necesarias para cada uno de estos. Además se hace una comparación entre el número de sumas que implicaría el uso de multiplicadores binarios y multiplicadores CSD. Para el caso CSD, de los 24 multiplicadores implementados, los que más sumas requieren utilizan solamente 6 sumadores. La sumas requeridas para realizar las multiplicaciones CSD es en promedio 5,08 y para los multiplicadores binarios es 6,92.



(a) (b)

Figura 4.6: Multiplicadores binarios estándares. (a) Completos (de uso general). (b) Con coeficientes constantes. En el primer caso se realizan sumas para todos los dígitos del multiplicando 1010. En el segundo caso se eliminan los bloques de sumas correspondientes a los dígitos iguales a cero del multiplicando 1010.

En comparación con un multiplicador binario de 16 bits de uso general, el cual requiere hacer al menos 16 sumas para una multiplicación (truncando los resultados parciales y el producto final a 16 bits), un multiplicador CSD representa una reducción de aproximadamente 60% en la cantidad de hardware necesario.

Utilizando multiplicadores binarios se podría lograr una reducción de la cantidad de sumadores necesarios, considerando que los coeficientes a multiplicar representan términos constantes, por lo que se podrían eliminar los bloques sumadores correspondientes a los bits iguales a cero en la correspondiente codificación binaria del multiplicando constante. Una comparación entre los multiplicadores estándares completos de uso general y multiplicadores estándares con coeficientes constantes se muestra en la figura 4.6. Bajo este principio, la cantidad de las sumas necesarias en promedio se reduciría aproximadamente a siete (6.92), como se observa en la comparación de la Tabla 4.7. Los multiplicadores CSD producen una reducción del 26.5% en promedio con respecto a estos multiplicadores estándares con coeficientes constantes.

Además, para los 24 coeficientes w_{ij} obtenidos para la matriz de transformación LDA, los multiplicadores CSD implementados utilizan aún menos recursos que el peor caso posible de un multiplicador CSD, en el que se podrían presentar hasta ocho sumas (o restas) para datos de 16 bits, como se muestra en la sección 2.3. Estos porcentajes de disminución en la cantidad de bloques sumadores son significativos, y se reflejan directamente en el área que ocupa la implementación del circuito reductor de dimensiones en un ASIC (Circuito Integrado de Aplicación Específica).

Tabla 4.7: Recursos del FPGA Spartan 3E utilizados por multiplicadores

Coeficiente	Binario	Sumas	CSD	Sumas	Slices	LUTs
w_{11} = -0.2464741953056455	-0.001111110001100	8	0.0-00000+00-0+00	4	29	54
w_{12} = -0.3555502486411402	-0.010110110000010	6	0.-0+00+0+00000-0	5	40	74
w_{13} = -0.5306721935663081	-0.100001111101101	9	0.-000-00000+0+0-	5	50	95
w_{21} = -0.1518257925170513	-0.001001101101111	9	0.00-0-00+00+000+	5	48	92
w_{22} = 0.2386699307646236	0.001111010001100	7	0.0+000-0+00+0-00	5	39	73
w_{23} = 0.06560739882374836	0.000100001100101	5	0.000+000+0-00+0+	5	36	68
w_{31} = -0.1928240011636543	-0.001100010101110	7	0.0-0+00-0+0+00+0	6	49	92
w_{32} = 0.09450822295660272	0.000110000011000	4	0.00+0-0000+0-000	4	28	52
w_{33} = 0.1414011819189337	0.001001000011001	5	0.00+00+000+0-00+	5	38	71
w_{41} = -0.185215429819623	-0.001011110110101	9	0.0-0+0000+0+0-0-	6	49	92
w_{42} = -0.212719791093454	-0.001101100111010	8	0.0-00+0+0-00+0-0	6	49	92
w_{43} = 0.2203161467779507	0.001110000110011	7	0.0+00-000+0-0+0-	6	49	92
w_{51} = 0.6898374160150471	0.101100001001100	6	+.0-0-0000+0+0-00	6	50	94
w_{52} = -0.556174540152289	-0.100011100110000	6	0.-00-00+0-0+0000	5	50	95
w_{53} = -0.4451596912224341	-0.011100011111010	9	0.-00+00-0000+0-0	5	40	74
w_{61} = -0.3794124749407183	-0.011000010010000	4	0.-0+0000-00-0000	4	30	55
w_{62} = -0.2304713597493061	-0.001110110000000	5	0.0-000+0+0000000	3	19	35
w_{63} = 0.3941648276280388	0.011001001110011	8	0.+0-00+0+00-0+0-	6	50	93
w_{71} = 0.1925034214615343	0.001100010100011	6	0.0+0-000+0+00+0-	5	39	73
w_{72} = -0.290196581935068	-0.010010100100101	6	0.0-00-0-00-00-0-	6	59	113
w_{73} = -0.08664246573236879	-0.000101100010111	7	0.00-0+0+00-0+00+	6	48	90
w_{81} = 0.4333895418184657	0.011011101111001	10	0.+00-000-000-00+	5	40	74
w_{82} = 0.5619412319501699	0.100011111101101	10	0.+00+000000-0-0+	4	30	55
w_{83} = -0.5333940518223561	-0.100010001000110	5	0.-000-000-00-0+0	5	50	95
Total	-	166	-	122	1009	1893
Recursos del FPGA (%)	-	-	-	-	21.67%	19.75%
Promedio de sumas	-	6.92	-	5.08	-	-

Capítulo 5

Conclusiones y recomendaciones

El Análisis de Discriminantes Lineales (LDA) permite obtener una reducción del espacio vectorial de ocho a tres dimensiones espectrales, maximizando la separación entre las distintas clases de sonidos a detectar y minimizando la distancia entre patrones de una misma clase. Se logró una reducción dimensional en la que las tres clases a clasificar ocupan distintas regiones del espacio resultante. La discriminación entre patrones aislados traslapados se realiza en el análisis temporal con los Modelos Ocultos de Markov (HMM) de la etapa de clasificación del SiRPA. Realizar esta reducción dimensional contribuye a simplificar el diseño y reducir el consumo de energía de las etapas posteriores de clasificación del sistema de reconocimiento de patrones, a la hora de implementar el diseño final del circuito integrado de aplicación específica para la detección de disparos y motosierras.

Se comprobó que realizar la transformación vectorial con LDA como técnica de reducción, ofrece ventajas en la separación de clases con respecto al Análisis de Componentes Principales (PCA). Sin embargo, debido a los resultados obtenidos en el proceso de reducción, es recomendable tener en consideración el reductor de dimensiones implementado con PCA durante el entrenamiento de la etapa de "Generación de Símbolos" y durante el entrenamiento de los HMMs, pues allí el comportamiento dinámico podría conducir a otro resultado distinto al análisis estático realizado en este proyecto.

El diseño propuesto, tanto con LDA como con PCA, reduce significativamente la cantidad de hardware necesario para la implementación del reductor utilizando multiplicadores CSD. En comparación con multiplicadores binarios ordinarios, la disminución resultante es de más del 60%. También, en comparación con multiplicadores binarios donde uno de los multiplicandos es constante y se podrían eliminar los bloques sumadores correspondientes a los bits iguales a cero en la correspondiente codificación binaria, los multiplicadores CSD producen una reducción en promedio del 26.5%. Esto contribuye a optimizar no sólo el uso área, sino que, al tener menos bloques de sumadores, se requieren menos cálculos para realizar las multiplicaciones necesarias. Con esto se reduce el consumo de potencia, cumpliendo los objetivos de éste proyecto.

Una recomendación a tomar en cuenta para optimizar el reductor de dimensiones es

utilizar un algoritmo optimizado para obtener la codificación CSD a partir de un número en representación binaria, el cual reduce al máximo la cantidad de restas en la codificación CSD. Mediante el uso de éste, se podría reducir un poco más el hardware necesario en para la implementación de los multiplicadores, ya que se sustituyen algunos de los bloques restadores presentes por sumadores.

Otra consideración a probar una vez que se complete el diseño del SiRPA, sería reducir la precisión con que se codifican los números en CSD. De comprobarse que no se presentan pérdidas importantes de información al utilizar menos dígitos para la representación de los coeficientes del reductor, podrían suprimirse algunos bloques sumadores y minimizar con esto el uso de área y su consecuente consumo de potencia.

Bibliografía

[1] Pablo Alvarado, Néstor Hernández, and Marvin Hernandez. D2ARS: Design and development of applications based on sensor networks. Final Report of Research Project, ITCR, July 2010.

[2] Mahmud Benhamid and Masuri Bin Othman. Hardware implementation of a genetic algorithm based canonical singed digit multiplierless fast fourier transform processor for multiband orthogonal frequency division multiplexing ultra wideband applications. *Journal of Mathematics and Statistics*, pages 241–250, 2009.

[3] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[4] FAO and REDLACH. Foro electrónico latinoamericano de sistemas de pago por servicios ambientales en cuencas hidrográficas. Technical Report Informe Final, FAO and REDLACH, Santiago, Chile, 2004.

[5] Keinosuke Fukunaga. *Introduction to statical pattern recognition*. Academic Press, 1990.

[6] Ernest Jamro. *Parameterised automated generation of convolvers implemented in FPGAs*. PhD thesis, University of Mining and Metallurgy, Kraków, Poland, June 2001.

[7] Srinidhi Kestur, John D. Davis, and Oliver Williams. BLAS comparison on FPGA, CPU and GPU. *IEEE Annual Symposium on VLSI*, 2010.

[8] Hyunsoo Kim, Barry L. Drake, and Haesun Park. Multiclass classifiers based on dimension reduction with generalized LDA. 2006.

[9] Cheng-Yu Pai, A.J. Al-Khalili, and W.E. Lynch. Low-power constant-coefficient multiplier generator. *Journal of VLSI Signal Processing*, pages 187–194, 2003.

[10] Syed M. Qasim, Ahmed A. Telba, and Abdulhameed Y. AlMazroo. FPGA design and implementation of dense matrix-vector multiplication for image processing application. In *World Congress on Engineering and Computer Science 2010 Vol I*, San Francisco, USA, October 2010.

[11] W. Roush. Wireless sensor networks. 10 emerging technologies that will change the world. *Technology Review*, 2003.

[12] Erick Salas. Reconocimiento en tiempo real de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA de Modelos Ocultos de Markov. Tesis de licenciatura, Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Abril 2010.

[13] Jonathon Shlens. A tutorial on principal component analysis. Tutorial, December 2005. URL http://www.snl.salk.edu/~shlens/.

[14] Esteban Smith. Reconocimiento digital en línea de patrones acústicos para la protección del ambiente por medio de HMM. Tesis de licenciatura, Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Enero 2008.

[15] Nelson Suárez. Se arriendan bosques tropicales santander. *Vanguardia Liberal*, Febrero 2002.

[16] M.A. Tahir, A. Bouridane, and F. Kurugollu. An FPGA based coprocessor for the classification of tissue patterns in prostatic cancer. 2004.

[17] Hima Deepthi Vankayalapati. Evaluation of wavelet based linear subspace techniques for face recognition. Master thesis, Fakultät für Technische Wissenschaften, Alpen-Adria-Universität Klagenfurt, Klagenfurt, October 2008.

[18] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52:2292–2330, 2008.

[19] Yan Zhang, Yasser H. Shalabi, Rishabh Jain, Krishna K. Nagar, and Jason D. Bakos. FPGA vs. GPU for sparse matrix vector multiply. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 255–262, Sydney, December 2009.

Apéndice A

LDA

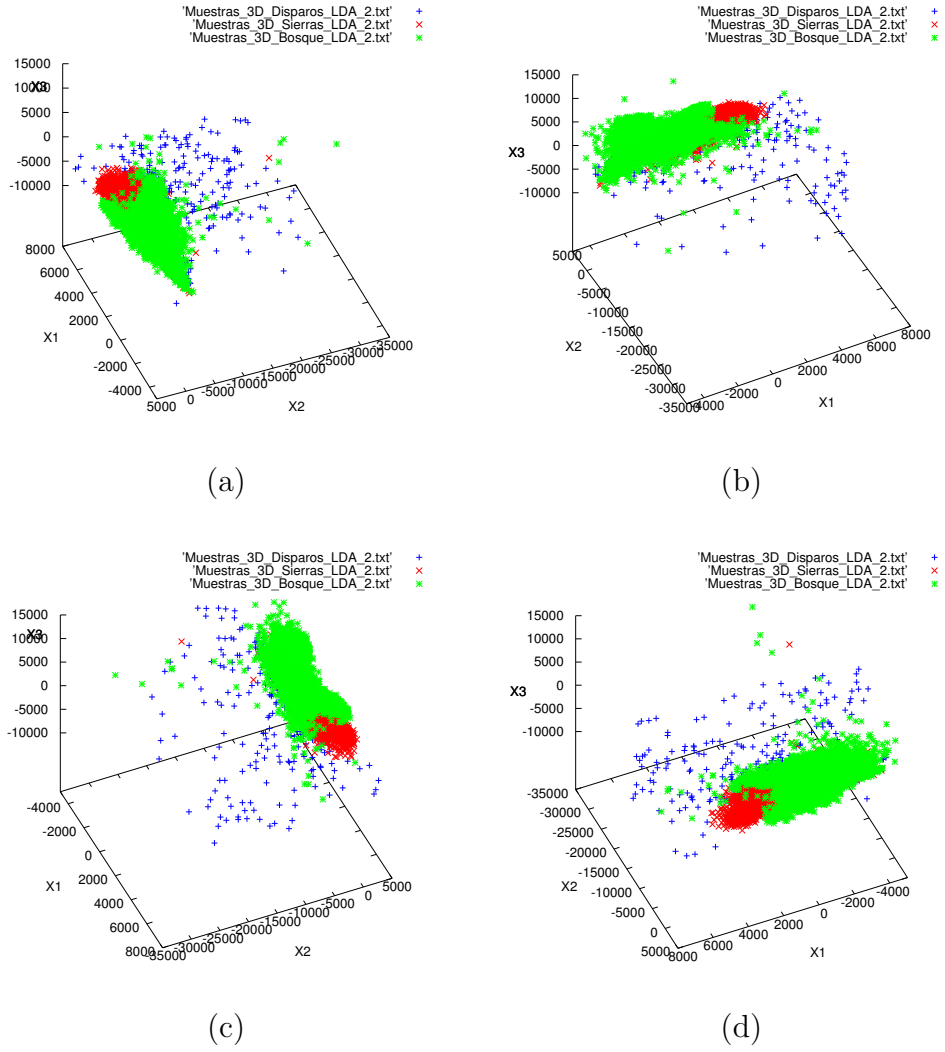
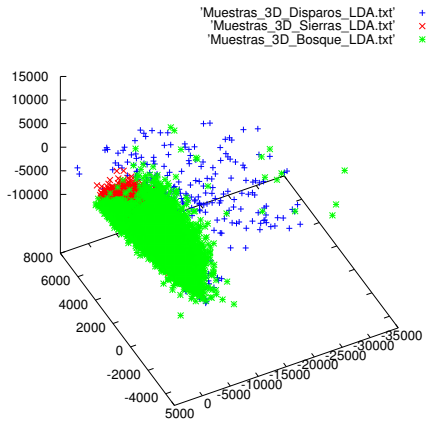
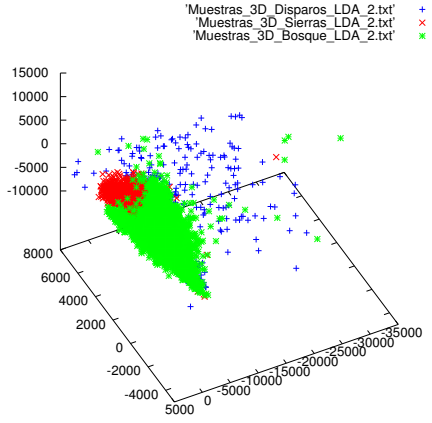


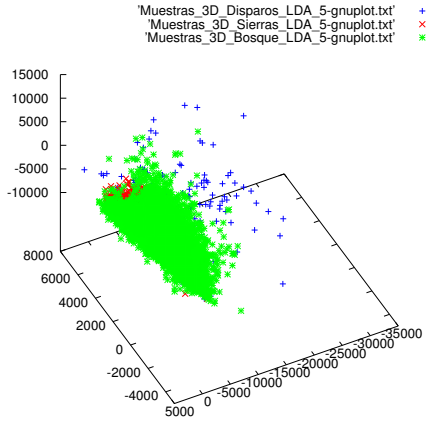
Figura A.1: Vista desde diferentes ángulos para un mismo conjunto de muestras de datos de '*disparos*', '*motosierras*' y '*bosque*' obtenidos a partir de la reducción realizada con LDA.



(a)



(b)

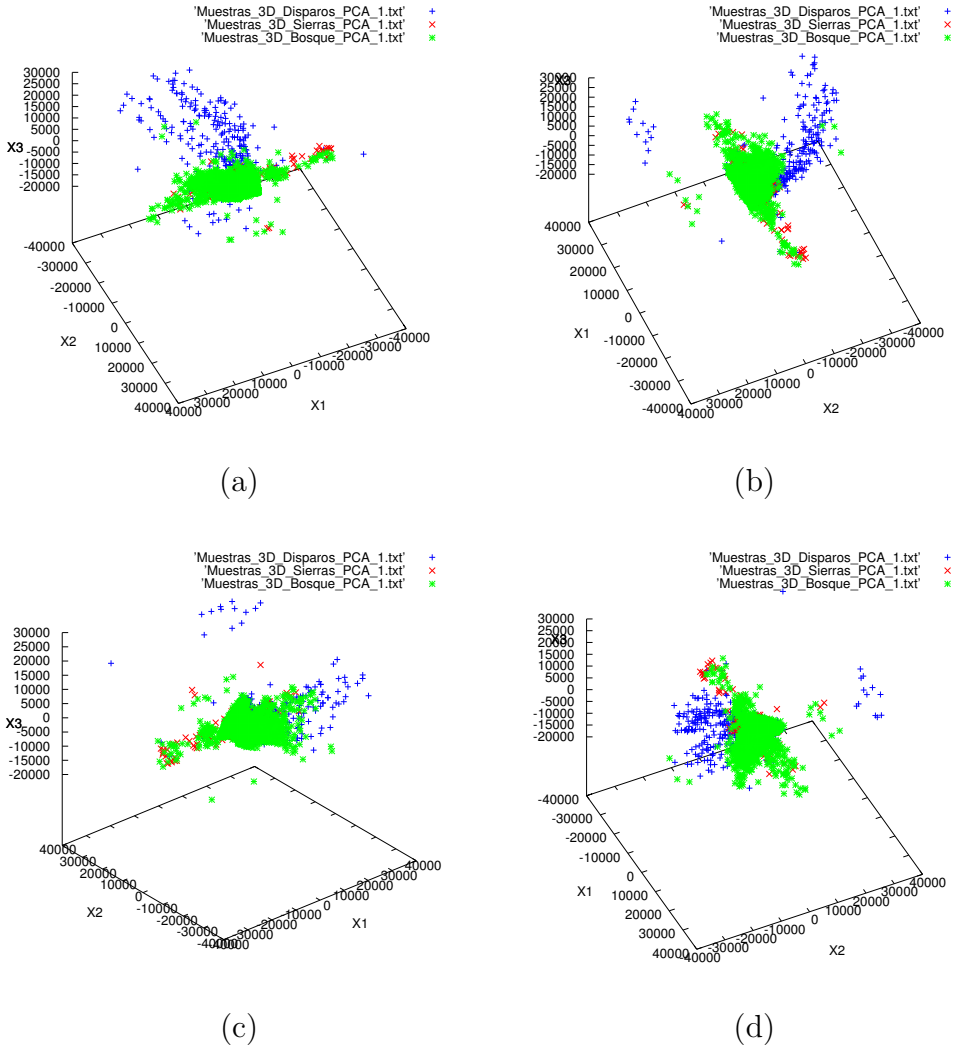


(c)

Figura A.2: Diferentes conjuntos de muestras de datos de '*disparos*', '*motosierras*' y '*bosque*' obtenidos a partir de la reducción realizada con LDA.

Apéndice B

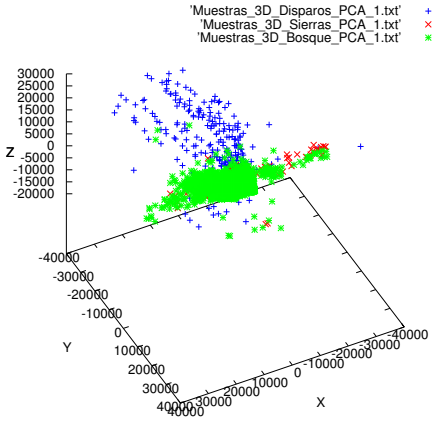
PCA



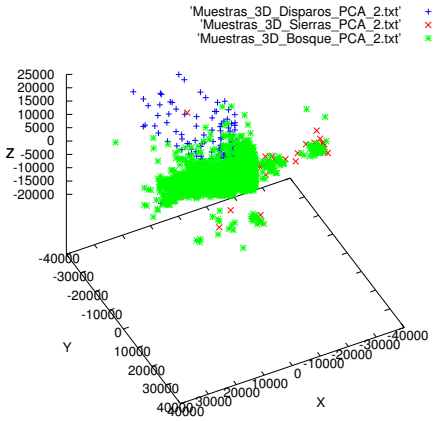
(a) (b)

(c) (d)

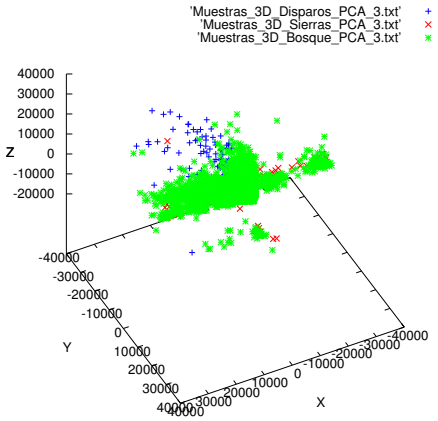
Figura B.1: Vista desde diferentes ángulos para un mismo conjunto de muestras de datos de '*disparos*', '*motosierras*' y '*bosque*' obtenidos a partir de la reducción realizada con PCA.



(a)



(b)



(c)

Figura B.2: Diferentes conjuntos de muestras de datos de '*disparos*', '*motosierras*' y '*bosque*' obtenidos a partir de la reducción realizada con PCA.

Apéndice C

Algoritmo de codificación de complemento a dos a CSD utilizado

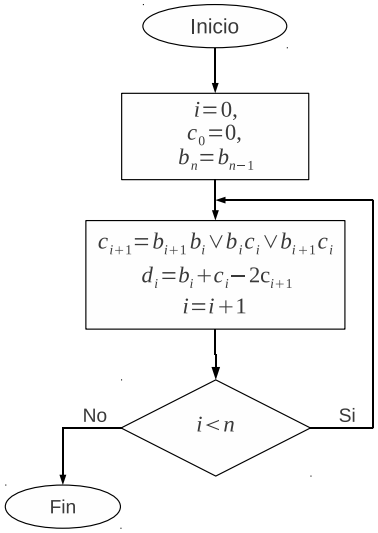


Figura C.1: Algoritmo utilizado para codificar números en Complemento a Dos a Canonic Sign-Digit. Tomado de [6].

Apéndice D

Código generado automáticamente

D.1 Reductor de dimensiones

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  library work;
7  USE work.ALL;
8  use work.def_type.all;
9
10
11 entity Reductor_de_dimensiones is
12     generic (n_bits : integer := 16);
13     port (
14       dimension_in_1:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
15       dimension_in_2:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
16       dimension_in_3:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
17       dimension_in_4:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
18       dimension_in_5:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
19       dimension_in_6:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
20       dimension_in_7:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
21       dimension_in_8:      IN INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
22       dimension_out_1:     OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
23       dimension_out_2:     OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
24       dimension_out_3:     OUT INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1
25     );
26 end Reductor_de_dimensiones;
27
28
29 architecture Behavioral of Reductor_de_dimensiones is
30
31   constant n_bits_multiplicadores: integer := 19;
32
33   signal dimension_1 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
34   signal dimension_2 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
35   signal dimension_3 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
36   signal dimension_4 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
37   signal dimension_5 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
38   signal dimension_6 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
39   signal dimension_7 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
40   signal dimension_8 :INTEGER RANGE -2**(n_bits-1) TO 2**(n_bits-1)-1;
41
```

```
42    signal w11_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
43    signal w12_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
44    signal w13_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
45    signal w21_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
46    signal w22_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
47    signal w23_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
48    signal w31_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
49    signal w32_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
50    signal w33_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
51    signal w41_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
52    signal w42_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
53    signal w43_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
54    signal w51_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
55    signal w52_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
56    signal w53_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
57    signal w61_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
58    signal w62_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
59    signal w63_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
60    signal w71_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
61    signal w72_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
62    signal w73_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
63    signal w81_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
64    signal w82_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
65    signal w83_out :INTEGER RANGE -2**(n_bits_multiplicadores-1) TO 2**(
          n_bits_multiplicadores-1)-1;
66
67
68  begin
69
70    Restador_de_offset: entity work.Restador_de_offset
71      generic map(n_bits => n_bits)
72      port map(
73        dimension_in_1 => dimension_in_1,
74        dimension_in_2 => dimension_in_2,
75        dimension_in_3 => dimension_in_3,
76        dimension_in_4 => dimension_in_4,
77        dimension_in_5 => dimension_in_5,
78        dimension_in_6 => dimension_in_6,
79        dimension_in_7 => dimension_in_7,
80        dimension_in_8 => dimension_in_8,
81        dimension_1 => dimension_1,
82        dimension_2 => dimension_2,
```

```
83        dimension_3 => dimension_3,
84        dimension_4 => dimension_4,
85        dimension_5 => dimension_5,
86        dimension_6 => dimension_6,
87        dimension_7 => dimension_7,
88        dimension_8 => dimension_8
89      );
90
91   coeficiente_w11: entity work.coeficiente_w11(Behavioral)
92    generic map(n_bits => n_bits_multiplicadores)
93    port map(w11_in => dimension_1,
94             w11_out => w11_out);
95
96    coeficiente_w12: entity work.coeficiente_w12(Behavioral)
97    generic map(n_bits => n_bits_multiplicadores)
98    port map(w12_in => dimension_1,
99             w12_out => w12_out);
100
101   coeficiente_w13: entity work.coeficiente_w13(Behavioral)
102   generic map(n_bits => n_bits_multiplicadores)
103   port map(w13_in => dimension_1,
104            w13_out => w13_out);
105
106   coeficiente_w21: entity work.coeficiente_w21(Behavioral)
107   generic map(n_bits => n_bits_multiplicadores)
108   port map(w21_in => dimension_2,
109            w21_out => w21_out);
110
111   coeficiente_w22: entity work.coeficiente_w22(Behavioral)
112   generic map(n_bits => n_bits_multiplicadores)
113   port map(w22_in => dimension_2,
114            w22_out => w22_out);
115
116   coeficiente_w23: entity work.coeficiente_w23(Behavioral)
117   generic map(n_bits => n_bits_multiplicadores)
118   port map(w23_in => dimension_2,
119            w23_out => w23_out);
120
121   coeficiente_w31: entity work.coeficiente_w31(Behavioral)
122   generic map(n_bits => n_bits_multiplicadores)
123   port map(w31_in => dimension_3,
124            w31_out => w31_out);
125
126   coeficiente_w32: entity work.coeficiente_w32(Behavioral)
127   generic map(n_bits => n_bits_multiplicadores)
128   port map(w32_in => dimension_3,
129            w32_out => w32_out);
130
131   coeficiente_w33: entity work.coeficiente_w33(Behavioral)
132   generic map(n_bits => n_bits_multiplicadores)
133   port map(w33_in => dimension_3,
134            w33_out => w33_out);
135
136   coeficiente_w41: entity work.coeficiente_w41(Behavioral)
137   generic map(n_bits => n_bits_multiplicadores)
138   port map(w41_in => dimension_4,
139            w41_out => w41_out);
140
141   coeficiente_w42: entity work.coeficiente_w42(Behavioral)
142   generic map(n_bits => n_bits_multiplicadores)
143   port map(w42_in => dimension_4,
144            w42_out => w42_out);
145   coeficiente_w43: entity work.coeficiente_w43(Behavioral)
146   generic map(n_bits => n_bits_multiplicadores)
147   port map(w43_in => dimension_4,
```

```
148             w43_out => w43_out);
149
150   coeficiente_w51: entity work.coeficiente_w51(Behavioral)
151   generic map(n_bits => n_bits_multiplicadores)
152   port map(w51_in => dimension_5,
153             w51_out => w51_out);
154
155   coeficiente_w52: entity work.coeficiente_w52(Behavioral)
156   generic map(n_bits => n_bits_multiplicadores)
157   port map(w52_in => dimension_5,
158             w52_out => w52_out);
159
160   coeficiente_w53: entity work.coeficiente_w53(Behavioral)
161   generic map(n_bits => n_bits_multiplicadores)
162   port map(w53_in => dimension_5,
163             w53_out => w53_out);
164
165   coeficiente_w61: entity work.coeficiente_w61(Behavioral)
166   generic map(n_bits => n_bits_multiplicadores)
167   port map(w61_in => dimension_6,
168             w61_out => w61_out);
169
170   coeficiente_w62: entity work.coeficiente_w62(Behavioral)
171   generic map(n_bits => n_bits_multiplicadores)
172   port map(w62_in => dimension_6,
173             w62_out => w62_out);
174
175   coeficiente_w63: entity work.coeficiente_w63(Behavioral)
176   generic map(n_bits => n_bits_multiplicadores)
177   port map(w63_in => dimension_6,
178             w63_out => w63_out);
179
180   coeficiente_w71: entity work.coeficiente_w71(Behavioral)
181   generic map(n_bits => n_bits_multiplicadores)
182   port map(w71_in => dimension_7,
183             w71_out => w71_out);
184
185   coeficiente_w72: entity work.coeficiente_w72(Behavioral)
186   generic map(n_bits => n_bits_multiplicadores)
187   port map(w72_in => dimension_7,
188             w72_out => w72_out);
189
190   coeficiente_w73: entity work.coeficiente_w73(Behavioral)
191   generic map(n_bits => n_bits_multiplicadores)
192   port map(w73_in => dimension_7,
193             w73_out => w73_out);
194
195   coeficiente_w81: entity work.coeficiente_w81(Behavioral)
196   generic map(n_bits => n_bits_multiplicadores)
197   port map(w81_in => dimension_8,
198             w81_out => w81_out);
199
200   coeficiente_w82: entity work.coeficiente_w82(Behavioral)
201   generic map(n_bits => n_bits_multiplicadores)
202   port map(w82_in => dimension_8,
203             w82_out => w82_out);
204
205   coeficiente_w83: entity work.coeficiente_w83(Behavioral)
206   generic map(n_bits => n_bits_multiplicadores)
207   port map(w83_in => dimension_8,
208             w83_out => w83_out);
209
210
211   dimension_out_1 <= ( w11_out + w21_out + w31_out  + w41_out
212                       + w51_out + w61_out  + w71_out + w81_out ) ;
```

```
213
214   dimension_out_2 <= ( w12_out + w22_out + w32_out  + w42_out
215                      + w52_out + w62_out + w72_out  + w82_out ) ;
216
217   dimension_out_3 <= (  w13_out + w23_out + w33_out  + w43_out
218                      + w53_out + w63_out + w73_out + w83_out ) ;
219
220 end Behavioral;
```

Código D.1: Código de Reductor de Dimensiones generado automáticamente