INSTITUTO TECNOLOGICO Y DE ESTUDIOS
SUPERIORES DE MONTERREY
CAMPUS MONTERREY
PROGRAMA DE GRADUADOS EN COMPUTACION,
INFORMACION Y COMUNICACIONES

OPTIMACION DE LOS PARAMETROS DE EXCITACION PARA CODIFICACION DE VOZ EN TRANSMISION SOBRE UN MEDIO OPTICO

TESIS

POR:

CARLOS ABUD AZUARA

MONTERREY, N. L. MARZO 2005

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN COMPUTACION, INFORMACION Y COMUNICACIONES

OPTIMACION DE LOS PARAMETROS DE EXCITACION PARA CODIFICACION DE VOZ EN TRANSMISION SOBRE UN MEDIO OPTICO

TESIS

POR:

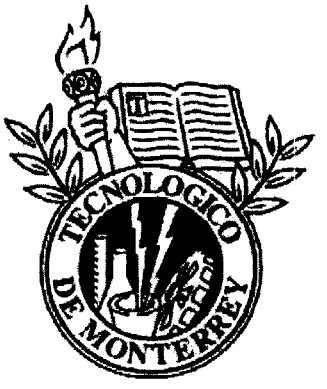
CARLOS ABUD AZUARA

MONTERREY, N. L. MARZO 2005

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES



OPTIMACIÓN DE LOS PARÁMETROS DE EXCITACIÓN PARA CODIFICACIÓN DE VOZ EN TRANSMISIÓN SOBRE UN MEDIO ÓPTICO

Tesis

por

Carlos Abud Azuara

Monterrey, N.L., Marzo del 2005

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

Los miembros del comité de tesis recomendamos que la presente tesis de Carlos Abud Azuara sea aceptada como requisito parcial para obtener el grado de **Maestría en Ciencias en Ingeniería Electrónica con especialidad en Telecomunicaciones**.

Comité de Tesis

Dr. José Ramón Rodríguez Cruz
Asesor Principal

Dr. Juan Arturo Nolazco F.
Sinodal

Dr. Jorge Carlos Mex P.
Sinodal

Dr. David A. Garza Salazar
Director del Programa de Posgrado en Electrónica, Computación, Información y Comunicaciones
Marzo del 2005

INDICE

Lista de Tablas

Lista de Figuras

Resumen

Modificando los parámetros de excitación del codificador híbrido CS-ACELP (Conjugate-Structure Algebraic-Code-Excited Linear-Prediction) y utilizando codificación conjunta de canal-fuente (JSCC) para la codificación de una señal de voz a ser transmitida sobre un canal de fibra óptica con ciertas características, se busca disminuir la tasa de bits o los errores inducidos por el canal.

El CS-ACELP utiliza un libro de códigos algebraico para representar la excitación junto con un libro de códigos adaptivo. Estos libros de códigos son modificados por sus respectivas ganancias. Reemplazando el libro de códigos algebraico del CS-ACELP por un libro de códigos fijo generado por medio de COVQ (Channel Optimized Vector Quantization), se buscará minimizar el efecto nocivo del canal de fibra óptica sobre los parámetros de excitación.

El codificador CS-ACELP cuenta ya con codificación conjunta de canal-fuente, sin embargo, la implementación de un libro de códigos fijo optimado para disminuir los errores del canal de fibra hace énfasis en este punto. La protección contra errores de canal que el CS-ACELP otorga a la señal sólo la ofrece para los parámetros LSP (Line Spectral Pairs) por contar estos con predicción MA (Moving Average Prediction) y no es específica a cierto canal. El CS-ACELP modificado por el libro de códigos vectorial generado con COVQ, creará un verdadero ambiente de codificación conjunta en el que los índices de los parámetros de voz a ser transmitidos son generados de forma que al ser decodificados, la señal obtenida es afectada lo menos posible por el ambiente ruidoso del canal de fibra óptica.

Se vislumbrarán los resultados por medio de simulación del sistema de comunicación y serán expuestos en valores numéricos de las formas de medición más utilizadas y en gráficas. El sistema será comparado con el mismo CS-ACELP en su forma original.

Introducción

La necesidad de unir grandes extensiones de terreno ha generado diversos tipos de comunicación, desde internet y mensajes escritos hasta voz sobre IP. Si bien es cierto que la tecnología digital se torna cada vez más fuerte, lo es también la necesidad de generar métodos de transmisión de señales de voz que cuenten con características de protección contra los distintos impedimentos (impairments) que el envío de información conlleva. La reducción de ancho de banda, comunicación en tiempo real, eliminación de efectos de ruido, son sólo algunos de los aspectos que retan a los investigadores en esta área.

Existen muchas técnicas para la transmisión de voz y la reducción de los impedimentos de comunicación en forma eficiente . Las señales digitales tienen características que facilitan su manipulación de tal forma que es posible disminuir el ancho de banda y la tasa de bits o proteger la señal contra errores. Estas ventajas se pueden obtener por medio de la codificación de la señal. La codificación es una representación de la señal de voz por medio de parámetros que son transmitidos de tal forma que el receptor regenera una señal que se parezca lo más posible a la de voz.

Al utilizar codificación se puede mantener cierto nivel de calidad en el proceso de recuperación de la voz, minimizando el número de bits de transmisión y aumentando la cantidad de información que se envía. También nos permite proteger la señal de voz de tal forma que sea irreconocible para terceros, como señal de voz en si, en caso de que al pasar por el canal de transmisión sea interceptada.

Algunos codificadores de voz buscan simular el tracto vocal extrayendo parámetros importantes de la voz representando tiempos de silencio, cantidad de aire que corre através del tracto (llamada señal de excitación) y su duración, intensidad de este, coeficientes de filtros de síntesis (que simulan el tracto) entre otros. Aunque la aproximación no es exacta, se han llegado a crear codificadores muy eficientes que reconstruyen la señal de voz de tal manera que el aparato auditivo humano alcanza a percibir tan solo pequeñas diferencias con respecto a las señales originales.

Existen tres tipos principales de codificadores de voz. Los codificadores de forma de onda, los vocoders y los híbridos. Los codificadores de forma de onda son aquellos que intentan reproducir la forma de onda de la

señal. Su codificación se puede llevar a cabo tanto en el dominio del tiempo como en el de la frecuencia. Entre los más importantes codificadores de onda se encuentran el PCM (Pulse Code Modulation), DPCM (Differential PCM) y ADPCM (Adaptive DPCM) que puede llegar a tener estándares de calidad muy altos.

En la codificación usando PCM cada muestra toma una valor de amplitud entre un rango finito de niveles específicos, esto es, se cuantifica. A cada nivel le corresponde un numero binario que será transmitido por el canal y será decodificado por el receptor.

Para DPCM se sigue un criterio parecido al de PCM pero en lugar de transmitir niveles de voltaje por muestra se transmiten las diferencias en amplitud entre una muestra y un estimado de la misma. Este tipo de codificación es muy útil para señales como la voz donde los niveles de correlación son altos. En el DPCM las muestras anteriores se utilizan para predecir la muestra actual. Esta muestra actual se resta de la señal original y de ahí se obtiene el llamado residuo R_s que es lo que codificamos y transmitimos.

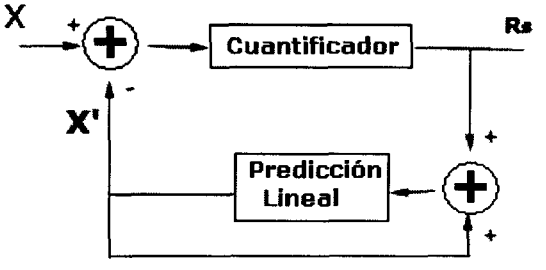


Figura 1: Codificador DPCM

El codificador ADPCM es parecido al DPCM a diferencia que el predictor y el cuantizador son adaptables, lo que minimiza el error cuadrático medio y nos proporciona una señal reconstruida más apegada a la señal original.

Los *vocoders* son codificadores de cuantización vectorial que eficientizan el uso de memoria y minimizan la tasa de bits gracias a la reducción en

la cantidad de parámetros enviados. Se caracterizan por tomar en cuenta la naturaleza de la señal a codificar y aprovechar las características inherentes de estas. Los vocoders tratan de reproducir la señal de audio sin importar que la señal reconstruida se parezca en forma a la señal original. Este tipo de codificadores ha logrado producir una señal de voz que sea intelegible, sin embargo no ha logrado generar una que suene natural. Las señales de voz reconstruidas por este tipos de codificadores suenan robóticas o producen ruidos extraños que el tracto vocal no generaría.

El LPC (Linear Predictive Coding) es un tipo de vocoder que supone que el tracto vocal puede ser modelado, típicamente, por un filtro de décimo orden.

MELP es un vocoder basado en el modelo tradicional paramétrico del LPC incluyendo cuatro características adicionales. Cuenta con con excitación mezclada, pulsos aperiodicos, dispersión de pulsos y un filtro adaptivo espectral mejorado. MELP codifica a una tasa de 2.4 kbps ubicándose como un codificador robusto y eficiente en cuanto a requerimientos computacionales se refiere.

Los codificadores híbridos son aquellos que explotan las características de los codificadores de forma de onda y de los vocoders codificando la señal a bajas tasas de transmisión y alta calidad. Estos codificadores llevan a cabo una representación de los parámetros de la señal de voz para tratar que la señal de síntesis se parezca lo más posible a la señal de voz original. El codificador híbrido trabaja con análisis por síntesis, es decir, en la etapa de codificación se analiza a la señal de voz donde se le extraen los parámetros más importantes para su codificación, tal que el receptor pueda decodificarlos y regenerar una señal aproximada de la señal de voz por medio de síntesis.

Entre los codificadores *híbridos* más importantes encontramos los de la familia CELP (Code Excited Linear Prediction). CS-ACELP y VSELP son parte de la familia CELP y tienen la característica de utilizar menos memoria y disminuir la tasa de bits al hacer uso de menos parámetros para representar la señal.

El CELP es un codificador que se basa en procedimientos de búsqueda de análisis por síntesis, cuantización vectorial y predicción lineal.

El VSELP es el estándar de las comunicaciones celulares digitales en Estados Unidos. Procesa la señal de voz a 7950 bps. La diferencia entre el VSELP y el CELP radica en la forma y en la estructura de su libro de códigos.

Existen otros tipos de codificadores de voz dentro de las tres rami-

ficaciones que hemos expuesto. Su trato teórico y, sobretodo, matemático va mucho más allá de lo aquí expuesto. El lector podrá, a lo largo de este escrito, conocer más a fondo el comportamiento del codificador híbrido llamado CS-ACELP sobre el cual estará basada la mayor parte de la investigación de la tesis.

La implementación de los codificadores de voz ha proveido a los sistemas de comunicación digital de la capacidad de transmitir más información con un ancho de banda más reducido. Por medio de la codificación de voz el emisor obtiene una mayor seguridad de que la información que transmita por un medio poco fiable o ruidoso se distorsione poco o no sea interceptada por un tercero durante su viaje hacia el receptor.

Debido a la importancia de los codificadores de voz en la era digital en la que vivimos y al uso de canales digitales como lo es la fibra óptica, el presente trabajo tiende a enforcarse en la optimación de un algoritmo de codificación basándose en las propiedades de un canal de fibra óptica. Por medio de técnicas iterativas de cuantización y el uso de modelos matemáticos, el lector se encontrará con un codificador híbrido modificado por cuantizadores vectoriales que busca mostrar un mejor desempeño comparativo al del conocido CS-ACELP.

Dentro de la optimación del algoritmo de codificación del CS-ACELP se encuentra la parte más importante y ardua de la investigación de esta tesis. En cuanto a codificación conjunta de canal-fuente (JSCC por Joint Source-Channel Coding) se refiere se llevarán a cabo los cambios más fuertes y se dará el enfoque más agudo. Dentro del algoritmo del CS-ACELP se llevarán a cabo cambios donde cambiará la forma de asignar índices por medio de cuantización vectorial optimada al canal usando COVQ.

Por medio del uso de JSCC se establece un compromiso entre la redundancia que se elimina por la codificación de fuente y aquella añadida por la de canal llegando a un punto intermedio en el que la compresión sea ideal sin perder protección contra errores del canal y sin añadir un exceso en número de bits de tasa de transmisión. El uso de codificación conjunta ha demostrado, en la mayoría de los casos, ser más eficiente. Originalmente, el uso de codificadores de fuente, que eliminaban la redundancia de esta, y codificadores de canal, que añadían redundancia de forma controlada, colocados en cascada no solo generaba bloques de código grandes sino que provocaba que el mensaje fuera propenso a distorsiones del canal. Es por lo anterior que se utilizará un codificador híbrido como el CS-ACELP que, se espera, modificado con COVQ, mejore las condiciones de distorsión de un canal de

fibra óptica ruidoso.

La implementación de los resultados aquí obtenidos podrían aplicarse a una línea de fibra de bajo precio como el utilizado por la tecnología FTTH (Fiber to the Home) en la que las compañías que ofrecen los servicios de videoconferencia, enlaces de voz en tiempo real e internet, puedan ofrecerlos através de un mismo medio de comunicación.

El uso de líneas ópticas baratas, como se verá en el ***CAPITULO 1***, ha sido utilizado con éxito en Japón y Estados Unidos [20][21]. La reducción de costos en la renta de servicios de esta índole por medio del uso de un medio de comunicación barato através de la integración de diversos servicios sobre la misma línea de comunicación y al mismo precio, ha hecho de la fibra óptica una solución rentable. Se espera que los medios ópticos baratos, aunados a la integración de servicios múltiples, logren gran impacto en el área de las comunicaciones y abaraquen un mercado de grandes dimensiones.

El primer capítulo de la tesis describe la problemática que impulsa el proyecto de investigación mismo, así como la justificación de la selección del canal de comunicación a utilizar, el codificador a modificar y el cuantizador a implementar para hacer posibles los cambios en el codificador de voz.

El ***CAPÍTULO 2*** da una breve explicación del surgimiento del codificador a utilizar, el CS-ACELP, describiendo algunas de sus características, da más detalles sobre el sistema de comunicación a implementar y habla sobre algunas de las investigaciones que anteceden a este trabajo.

En el ***CAPÍTULO 3*** se hace una descripción más profunda sobre el codificador CS-ACELP. Se desciben algunas de sus características primordiales, funcionamiento, su codificador y decodificador.

El trato matemático para lo obtención de un cuantizador vectorial óptimo enfocado a disminuir los errores de un canal de comunicación en base a sus características físicas y ambiente nocivo para la señal se muestra en el ***CAPÍTULO 4***.

El modelo del canal de transmisión se describe en el ***CAPÍTULO 5*** donde se hace incapié no sólo en las características generales de un canal de fibra si no, más específicamente, en las del canal de fibra utilizado en la simulación de esta investigación. En este capítulo se describe el ruido modal y su implementación matemática para efectos de simulación.

Es en el ***CAPÍTULO 6*** donde se describe la implementación de las fórmulas matématicas para el cálculo del codificador vectorial. En este capítulo se describe la función iterativa que el software debe seguir para obtener el cuantizador. En el capítulo se muestran algunos resultados de la

simulación que se llevó a cabo para obtener el cuantizador.

El ***CAPÍTULO 7*** describe el funcionamiento del software y consta de varias partes. En este capítulo se describe, inicialmente, la búsqueda iterativa del codificador vectorial optimado a un canal de fibra con ruido modal, en una segunda etapa se explican las modificaciones hechas a la norma original del CS-ACELP. La obtención del archivo llamado de excitación, que contiene los vectores codificados de cada señal utilizada para los experimentos, se describe en este capítulo. La simulación del canal de fibra, la generación de los pulsos de LED (Light Emission Diode) y la distorsión de bits por el ruido modal se describen aquí. Se especifica el uso de un decodificador realizado en Matlab que simula el decodificador del CS-ACELP y que fue de gran utilidad para la obtención de resultados.

Los resultados de la simulación y conclusiones de este trabajo se describen en el ***CAPÍTULO 8***, mientras que el ***CAPÍTULO 9*** despliega algunas opciones de trabajo a futuro sobre la misma línea de investigación que lleva esta tesis.

1. Definición del Problema

Una de las principales aplicaciones de transmisión de señales de voz es la de llevar a cabo comunicación en tiempo real. La integración de datos y voz en una misma red ha sido, durante los últimos años, una premisa en la innovación y desarrollo de nuevas tecnologías en sistemas de comunicación.

Para realizar una conversación de voz en tiempo real sobre una red digital, es necesario que esta cuente con una capacidad de transmisión muy alta, que no agregue retardos extras a los que la conversión, análogo-digital, de la señal de voz conlleva. La fibra óptica maneja anchos de banda que van desde cientos de MHz a decenas de GHz, cumpliendo con esta característica.

Otra de las causas por las que se escogió la fibra óptica por sobre otros canales de transmisión, el par trenzado por ejemplo, es que la fibra no presenta diafonía, no puede ser interferida por señales externas, es dieléctrica, tiene una capacidad de multiplexaje amplia, pesa poco, soporta grandes tensiones, es inmune a la corrosión y es flexible.

Aunque la fibra óptica pareciera el conductor por excelencia suele ser más cara que otros medios de transmisión, presenta pérdidas de acoplamiento y los emisores de luz distorsionan la señal dependiendo de la vida útil que hayan dado. Estas características serán expuestas de manera más extensa en la **CAPÍTULO 4**, donde se explican más a fondo las desventajas físicas de la fibra óptica.

Se mencionó que la fibra óptica provee a la señal de voz de la capacidad de viajar a una gran velocidad. Dos interlocutores pueden entablar una conversación en tiempo real sin retrasos perceptibles. Sin embargo, el canal de fibra, como cualquier otro canal, deformará la señal de voz al grado que el enlace de comunicación y la percepción auditiva de la señal recibida, pudieran verse seriamente afectados. Por tal razón, es conveniente tratar la señal de voz utilizando algún algoritmo de codificación que nos permita, no solo protegerla contra errores inherentes del canal de fibra si no que además, no agregue retardos extras exagerados debidos a la codificación.

Un canal de fibra de un kilómetro puede pesar 190 kilogramos comparado con los casi 8000 kilogramos que pesaría uno de coaxial, además observaríamos una atenuación de 0.4 decibeles en el primer kilómetro en un canal de fibra contra 40 decibeles para uno de coaxial. Si bien es cierto que el canal de fibra cuenta con ciertas ventajas en cuanto a otros canales basados en cobre con respecto a desempeño y trato físico se refiere, lo es también que la implementación y el mantenimiento de este son más caros. Es por eso

que el uso de un codificador podría compensar el costo del canal de fibra y de su mantenimiento al hacer una compresión de la señal, obteniendo un uso más eficiente del canal. La figura 2 muestra una tabla comparativa del CS-ACELP contra otros codificadores.

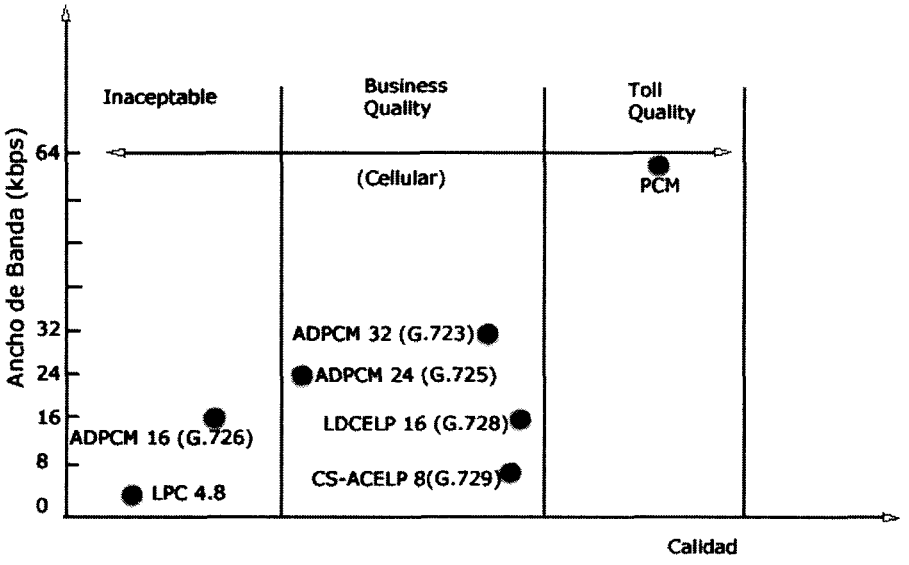


Figura 2: Comparación- CS-ACELP

En la tabla anterior se puede observar la alta calidad de codificación que ofrece el CS-ACELP aunada a la compresión ofrecida del sistema. Esta característica de compresión del CS-ACELP, guardando la calidad de la señal, nos permite hacer uso de su implementación para el sistema en cuestión de una forma favorable.

El CS-ACELP no sólo cuenta con un buen balance entre compresión y calidad sino que es adaptable a sistemas de comunicación basados en paquetes, es el estándar mayormente probado en sistemas de comunicación, incluyendo sistemas inalámbricos, y es el estándar de la ITU-T de más bajo nivel de tasa de bits de transmisión con calidad toll. Estas son algunas de las características que hicieron del CS-ACELP el candidato ideal para la implementación de este trabajo y su descripción se dará de forma más explicita en el ***CAPÍTULO 2***.

Para nuestro estudio cabe mencionar que este codificador protege a la señal contra errores inherentes del canal mostrando robustez. Es aquí donde los medios de esta tesis propondrán justificar el fin, cuando en lugar de utilizar la metodología de protección de errores típica del CS-ACELP se supla por un codificador vectorial (COVQ). Para la implementación de este se podría reemplazar alguno de los parámetros específicos del CS-ACELP por un libro de códigos vectorial que minimice los errores inherentes del canal. La implementación de COVQ se tratará más a fondo en el ***CAPÍTULO 3***.

Al implementar COVQ (Channel Optimized Vector Quantization), tipo conocido de cuantización vectorial, en el vocoder CS-ACELP (Conjugate-Structure Algebraic-Code-Excited linear-prediction) se espera obtener una disminución en el tasa de bits o en la cantidad de errores presentes en la señal de voz recibida. Esta señal será transmitida a través de un canal de fibra óptica que introduce ruido modal a la señal, el cual será representado por un modelo matemático a determinar.

El lector deberá tomar en cuenta a lo largo del escrito un sistema de comunicaciones donde una señal de voz modulada con PCM a 16 bits y muestreada a 8Kbps se codifica utilizando dos algoritmos distintos por cuestiones de comparación. Una vez codificada la señal se hace pasar por un canal de fibra óptica que distorsiona a la señal agregándole ruido modal. Ya que la señal codificada ha pasado a través del canal de fibra, se decodifica usando la técnica correspondiente a cada sistema de codificación para su comparación.

El objetivo principal de la investigación es el de desarrollar un sistema de comunicación óptimo que cuenta con ciertas características de robustez y compararlo con otro que no ha sido diseñado para un desempeño enfocado a la protección de los parámetros de codificación de voz al ser transmitidos sobre un canal con distorsiones debidas a ruido modal.

La señal de voz se codificará, en primer instancia, usando un sistema de codificación combinado con COVQ, que es un cuantizador vectorial que busca optimar la asignación de índices de un cierto parámetro para disminuir cierta medida de distorsión que está relacionada con el comportamiento de algún canal de comunicación, en este caso, la fibra óptica. Al tratar de preveer el comportamiento del canal de fibra y asignar los índices correspondientes a algún parámetro cuantizado de la voz de acuerdo a este, se espera que los resultados obtenidos para el sistema de codificación arrojen respuestas más favorables en cuestión de protección de errores o disminución de tasa de bits de transmisión.

El sistema descrito se muestra en la figura 3 en forma de diagrama de bloques.

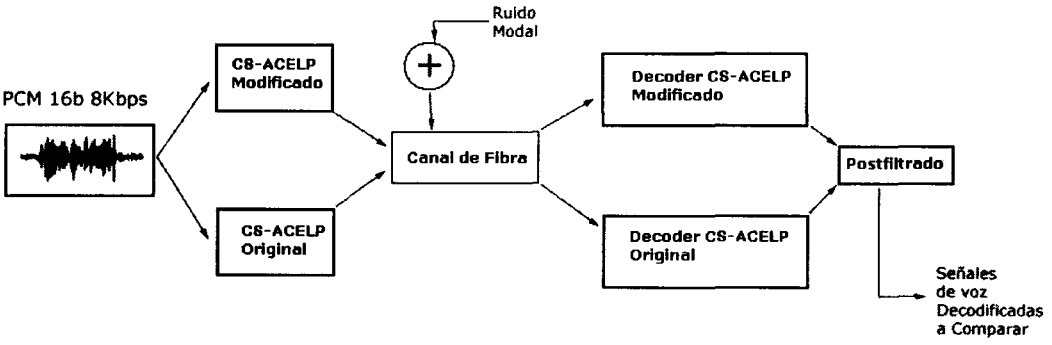


Figura 3: Sistema de Transmisión

El otro sistema de codificación a utilizar es el específicado por la norma del G729, el CS-ACELP en su forma original y descrito en el ***CAPÍTULO 3***.

El acrecentado avance de la tecnología provoca que sus usuarios clamen por soluciones múltiples de comunicación como redes de voz y datos integradas que no sólo les permita enviar información y video si no que a su vez les permita establecer una conversación de voz en tiempo real.

Existe una arquitectura de fibra óptica que permitiría en algún futuro crear un sistema de comunicación de múltiples servicios integrados donde el ancho de banda y la calidad no serían limitantes.

El consejo de la Asociación FTTH (Fiber to the Home) y la Asociación de la industria de las Telecomunicaciones de los Estados Unidos de Norteamérica anunció el 19 de Mayo del 2004 la implementación de FTTH en 192 comunidades en 32 de los estados del país en cuestión , lo que habla del importante crecimiento de esta tecnología en cuanto a enlaces de voz se refiere, con un aumento de un 40 a un 75 por ciento de aumento en la cantidad de suscriptores en dichas comunidades para los siguientes tres meses [20][21].

Una sola línea de fibra sería capaz de dar servicio hasta a 32 usuarios con todos los servicios que estos hayan contratado incluyendo conversaciones de voz, lo que representaría un gran ahorro de dinero y sobre todo de espacio.

Observamos sin dificultad la importancia que la fibra óptica tiene en nuestros días y que pudiera alcanzar en un futuro. La implementación de FT-TH podría tener tanto impacto que se habla de ella como uno de los futuros estándares de comunicación de la industria, es por eso que una investigación como la expuesta por esta tesis, resulta de vigencia, importante y pudiera resultar un punto de partida para alguna solución de posible implementación para una aplicación práctica como la mencionada.

2. Antecedentes

Creado para generar una codificación de la misma calidad que el ADPCM a 32kps, con una reducción en la tasa de bits de transmisión y con una protección contra errores menor al 1 % con respecto a la señal original, aparecen dos versiones de codificadores de voz que para 1995 conjuntarían sus mejores características para generar el CS-ACELP, que cumpliría así con las especificaciones requeridas por la ITU-T.

El codificador está basado en predicción lineal. La señal de voz codificada se compara con ella misma para generar parámetros que, seleccionados adecuadamente, arrojan el error cuadrático medio mínimo.

Son varios los usos que se le dan al CS-ACELP en la actualidad. En la literatura podemos encontrar algunas líneas de investigación que difieren a las que expone este trabajo y algunas otras que pretenden similares.

Los codificadores de voz, generalmente, se diseñan para trabajar bajo condiciones generales. Si una señal de voz se codifica utilizando un CS-ACELP, podríamos transmitirla sobre un canal inalámbrico o uno de cobre utilizando la modulación correcta. El hecho de que los parámetros de codificación de las señales de voz de estos codificadores estén diseñados de forma genérica, da a los investigadores la capacidad de modificar la forma de generar uno o más parámetros de manera indistinta con el propósito de mejorar el desempeño del codificador.

En esta tesis se expone un sistema óptimo de codificación que proteje a la señal de voz al pasar por un canal de fibra óptica ruidoso. Bajo este lineamiento, investigadores en otras partes del mundo han desarrollado trabajos de investigación de gran interés. En [11] el autor propone modificar el CS-ACELP de una forma similar a lo aquí expuesto. En el trabajo mencionado se propone el reemplazo del libro de códigos fijo por un método de codificación adaptable que disminuya la tasa de bits de transmisión. Observando los resultados el autor afirma obtener una compresión de hasta 6.4 kbps con una calidad muy similar al CS-ACELP a 8 kbps.

El uso del CS-ACELP no se limita únicamente a trabajos de investigación enfocados a la disminución de la tasa de bits de transmisión o de errores inducidos por el canal sobre la señal de voz. En [10] y en [12] aparecen dos aplicaciones interesantes del CS-ACELP a tecnologías actuales.

La primera referencia habla sobre la implementación de un codificador CS-ACELP de tasa de bits de transmisión variable basado en nuevos algoritmos robustos a ambientes ruidosos inalámbricos. El codificador presenta ocho modos de operación que van de 0 a 8Kbps con una tasa de bits de transmisión promedio de 4Kbps. Se menciona una disminución de un 20 por ciento en la tasa de bits de transmisión sin disminuir la calidad perceptiva y manteniendo el retraso algorítmico de codificación. En la siguiente figura, obtenida de [10], se puede observar el porcentaje del modo seleccionado en uso, la tasa de transmisión promedio y el CMOS (Comparison Mean Opinion Score) que es una medida de comparación realizada en forma de encuesta entre varias personas bajo distintas condiciones acústicas. Para el mejor entendimiento de los resultados consultar la referencia.

PERCENTAGE OF MODE SELECTION, AVERAGE BIT RATE (ABR), AND COMPARISON MEAN OPINION SCORE (CMOS) IN VARYING ACOUSTIC CONDITIONS (C = CAR, T = TRAFFIC, B = BABBLE). A MINUS SIGN IN THE LAST COLUMN MEANS THAT THE PROPOSED VBR CODER IS WORSE THAN THE ITU-T G.729 WITH THE VAD ITU-T G.729 ANNEX B

Noise	SNR	Mode (%)								ABR (kbit/s)		CMOS
	(dB)	1	2	3	4	5	6	7	8	VBR	G.729/DTX	
None		60.3	3.10	0.0	0.0	0.0	8.2	4.2	24.2	2.24	2.52	- 0.18
	20	44.6	10.4	0.0	0.0	0.0	3.9	4.0	37.1	3.21	6.05	- 0.21
Car	10	24.0	6.20	0.0	0.0	0.0	2.9	4.9	62.0	5.18	7.36	- 0.12
	0	10.8	7.20	0.1	0.2	0.0	2.6	3.7	75.4	6.23	7.15	+ 0.10
	20	42.4	11.9	0.0	0.0	0.0	12.1	3.9	29.7	2.81	4.69	- 0.10
Traffic	10	25.6	17.3	3.4	2.7	5.1	7.7	4.5	33.7	3.49	5.12	- 0.12
	0	18.8	14.2	10.9	9.4	6.6	6.5	2.2	31.4	3.69	5.07	- 0.06
	20	18.3	12.3	0.9	0.9	1.4	13.1	3.4	49.7	4.54	5.18	- 0.08
Babble	10	4.5	3.70	12.9	10.9	10.8	5.3	3.6	48.3	5.30	4.78	+ 0.15
	0	2.5	2.10	15.5	12.9	12.6	3.6	2.4	48.4	5.45	4.69	+ 0.10
Average		25.18	8.84	4.37	3.70	3.65	6.59	3.68	43.99	4.21	5.26	- 0.05

Figura 4: Tabla IV-Referencia [10]

La segunda referencia hace incapié en el uso del CS-ACELP como un codificador multi-modo y de múltiples velocidades de transmisión que presenta calidad toll. El codificador es desarrollado específicamente para trabajar con voz sobre IP y ofrecer QoS (Quality of Service o calidad de servicio). El codificador usa 3 modos de transmisión estandar para la ITU-T y cuatro categorías de codificación. El codificador explota técnicas de reconocimiento de lógica difusa para clasificación de multi-modos y la codificación de nuevos modelos para las diferentes clases fonéticas existentes. De acuerdo a los resultados expuestos, utilizando un mecanismo de control para el codificador, el sistema sería capaz de otorgar calidad en el servicio para cualquier tipo de red. Los resultados numéricos expuestos por Beritelli, Ruggeri y Casale en la referencia se despliegan de forma similar a los de la referencia [10] y se recomienda al lector buscarlos en [12].

No sólo el CS-ACELP ha sido participe importante del área del las telecomunicaciones, el cuantizador vectorial COVQ es comunmente mencionado en la literatura y se utiliza ampliamente no sólo para el G.729 de la ITU-T si no para otros codificadores de voz [18][19].

El desarrollo de un algoritmo de codificación conjunta canal-fuente de tasa de bits de transmisión baja variable sobre canales ruidosos se expone en [13]. El algoritmo se desarrolla aplicando COVQ directamente sobre el ya mencionado codificador CELP y buscará robustez para canales AWGN (Additive White Gaussian Noise) y canales con desvanecimientos tipo Rayleigh. En la investigación de [13] buscan reintegrar al algoritmo la redundancia que la codificación de fuente ha removido de la señal de voz sin incrementar el ancho de banda o la complejidad del sistema. Esto llevando una codificación más intensa de canal por medio de Turbo códigos y aproximación iterativa al limite de Shannon. Los resultados son favorables para decodificación por decisiones soft y se hace uso de MOS para evaluar el desempeño comparativo del codificador. La figura siguiente, obtenida de [13], muestra las tablas de resultados 2 y 3 de la referencia en cuestión con el criterio de evaluación y los resultados basados en MOS y POP (Paired-Comparison Preference).

Table-2 Speech quality evaluation criterion

Score	*Speech*	*Levels of Distortion*
5	*Excellent*	*Imperceptible*
4	*Good*	*Just Perceptible, but not annoying*
3	*Fair*	*Perceptible and slightly annoying*
2	*Poor*	*Annoying, but not objectionable*
1	*Unsatisfactory*	*Very annoying and objectionable*

Table-3 Listening tests based on MOS and PCP

Soft Decoding	Hard Decoding	MOS Test (Scores)*		PCP Test (Chosen Times)	
Fig.3.(c)	Fig.3.(b)	3.2	2.1	30	0
Fig.3.(e)	Fig.3.(d)	3.0	1.0	30	0
Fig.3.(g)	Fig.3.(f)	3.3	2.6	30	0
Fig.3.(i)	Fig.3.(h)	3.1	1.0	30	0

* MOS score for the original female speech is set as 5.0.

Figura 5: Tablas 2 y 3-Referencia [13]

Uno de los ejemplos más palpables de investigaciones similares sobre el mismo curso es el expuesto en [15]. En la referencia la autora expone un sistema de comunicación conformado por un canal de fibra que cuenta con características de distorsión basadas en dispersión. En la tesis de investigación se trabaja sobre un CELP que modificado por medio de COVQ minimiza la medida de distorsión promedio presente en el receptor buscando una compresión de la señal. Si el lector busca un mejor entendimiento de esta u otra investigación similar a la mencionada, la referencia [15] es un muy buen apoyo y cuenta con resultados de gran interés.

Existen múltiples aplicaciones y líneas de investigación sobre el CS-ACELP, el COVQ y la fibra óptica en caso de que el lector quisiera encontrar mayor apoyo al querer seguir un estudio del estilo. La intención de este capítulo es informativa y no busca ser más explícita por lo que se continúa con el desarrollo de la tesis.

3. Codificador CS-ACELP

El CS-ACELP (Conjugate-Structure Algebraic-Code-Excited Linear-Prediction) es un codificador basado en el codificador de predicción lineal CELP. Opera con tramas de 10 milisegundos que correspondes a 80 muestras para una tasa de muestreo de 8000 muestras por segundo. Este codificador es equivalente en calidad a un codificador ADPCM de 32 Kbps para la mayoría de las condiciones de operación.

Entre las principales aplicaciones del codificador se encuentran sistemas de comunicación personal (PCS), sistemas satelitales digitales y otras aplicaciones como empaquetado de voz.

Por sus características este codificador debe ser capaz de soportar errores del canal y pérdida de tramas debido a que cuenta con algoritmos de predicción para robustez contra canales ruidosos y tiene capacidad de detectar y reconstruir tramas perdidas.

3.1. Codificador

El codificador se basa en el modelo de codificación del CELP. En este modelo, la señal decodificada se compara contra la original y los parámetros del codificador se seleccionan de tal manera que el error cuadrático medio entre la señal original y la reconstruida se minimice.

El CS-ACELP utiliza libros de código que son conformados por vectores. A cada vector se le deberá asignar un índice. La asignación de los índices deberá ser tal que al comparar la señal de voz original con la reconstruida la diferencia audible sea mínima o por lo menos, que la calidad de codificación de este codificador sea igual o mejor que la del ADPCM a 32Kbps (con la ventaja de una disminución en la tasa de bits de transmisión o el error de transmisión).

Para codificar la señal, se utilizan algoritmos de asignación y búsqueda de los índices de los libros de código que permitan una manejo rápido de estos. Dentro de estos procesos se utilizan predictores basados en filtros MA

(Moving Average). Esto proporciona protección contra errores de canal en varias de las etapas de codificación.

La principal finalidad en la búsqueda de parámetros y asignación de índices descritos por la norma del CS-ACELP no es la reducción de errores de canal sino la simplificación de los algoritmos para determinar los índices de los libros de código y el hacer más sencilla su búsqueda.

Para asignar lo índices correspondientes a cada vector representativo de los libros de código, podemos utilizar métodos de codificación conjunta. En el caso de esta tesis se implementará Channel-Optimized Vector Quantizers. Este cuantizador vectorial buscará asignar los índices a cada vector o centroide de los libros de código fijo.

La implementación del cuantizador está justificada en la correcta asignación de énfasis a distintos parámetros de la señal de voz. Así, intentando crear un mejor compromiso entre el número de bits usado para protección de la señal de voz y el usado para protección de errores del canal. Se deben exponer los resultados referentes a la mejora en disminución de tasa de bits y protección contra errores, si acaso existiera, como premisa fundamental en la elaboración de la investigación.

El codificador algebraico que usaremos está diseñado para operar con una señal digital muestreada a 8000 Hz que se modulará con PCM a 16 bits. El codificador opera con tramas de 10 ms que corresponden a 80 muestras si se ha muestreado a 8000 muestras por segundo. Cada 10 ms la señal de voz se analiza para extraerle los parámetros del modelo CELP, como son los coeficientes del filtro de predicción lineal, y los índices y ganancias de los libros de código fijo y adaptivo. Obtenido de [7] se muestra un diagrama del principio del codificador del CS-ACELP, ver la figura 6.

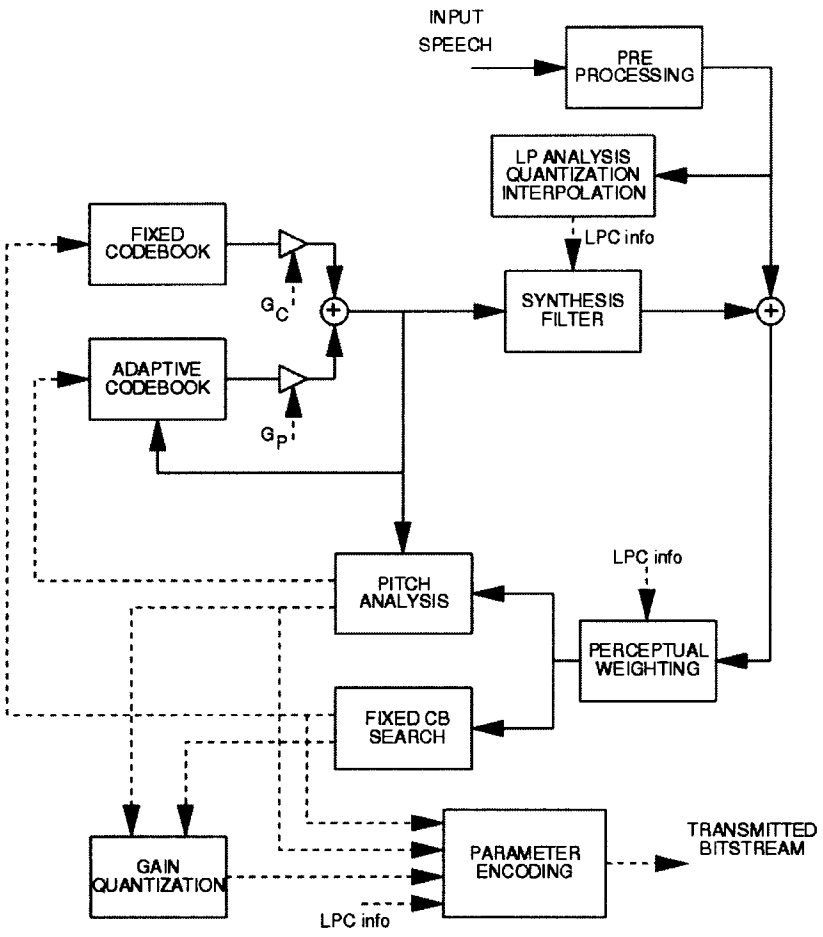


Figura 6: Codificador

3.1.1. Predicción Lineal

El CS-ACELP utiliza predicción lineal de tal forma que podamos utilizar la correlación entre los segmentos de la señal de voz para así lograr una mejor compresión. Se lleva a cabo un análisis de la señal de voz utilizando un filtro LP (Linear Prediction). El residual del filtro son algunos de los parámetros que representan las características importantes de la señal de

voz. Estos parámetros se cuantizan en el dominio de LSF (Line Spectral Frecuencies o LSP para Line Spectral Pair) dado que los coeficientes a_i del predictor lineal tienen una distribución pobre por lo que es difícil interpolarlos para su cuantización.

Lo primero es la obtención de los coeficientes de predicción lineal. El filtro LP es un filtro de décimo orden. El filtro de síntesis es de la forma,

$$\frac{1}{\hat{A}(z)} = \frac{1}{1 + \sum_{i=1}^{10} \hat{a} Z^{-i}} \tag{1}$$

donde $\hat{a}_i$ son los coeficientes cuantizados LP. Los coeficientes se calculan cada 10 ms usando 5 ms de voz pasada y 5 ms de futura. El cálculo de los coeficientes se lleva a cabo por medio del algoritmo de Levinson Durbin. Para su cuantización convertimos los coeficientes del filtro LP a LSP usando polinomios chebyshev, después representamos los coeficientes LSP como frecuencias normalizadas en radianes.

$$w_i = \arccos(q_i), i = 1, ..., 10 \tag{2}$$

donde q_i son los coeficientes LSP en el dominio del coseno y w_i son los coeficientes LSF en el dominio dela frecuencia. Una vez cuantizados los coeficientes LSF se pueden codificar usando el criterio de error cuadrático medio [7].

3.1.2. Análisis de Lazo Abierto del Pitch

Después de la cuantización, los coeficientes LSP se interpolan para después convertirlos de nuevo a coeficientes LP que son los que se utilizarán para el filtro de peso perceptual:

$$W(z) = \frac{1 + \sum_{i=1}^{10} \gamma_1^i a_i z^{-i}}{1 + \sum_{i=1}^{10} \gamma_2^i a_i z^{-i}} \tag{3}$$

Los valores de γ representan la respuesta a la frecuencia del filtro. A partir de este filtro se obtiene la señal perceptual de voz que se utilizará para realizar el análisis de lazo abierto para encontrar el pitch:

$$sw(n) = s(n) + \sum_{i=1}^{10} a_i \gamma_1^i s(n-i) + \sum_{i=1}^{10} a_i \gamma_2^i s(n-i) \tag{4}$$

donde $s(n)$ es la señal de voz preprocesada y a_i son los coeficientes LP.

Una vez realizado el análisis de lazo abierto para el pitch se calcula la respuesta al impulso,

$$h(n) = \frac{W(z)}{\hat{A}(z)} \tag{5}$$

que se realiza para cada subtrama.

3.1.3. Libro de Códigos Adaptable

La señal $x(n)$ (target signal) para la búsqueda del libro de códigos adaptable se calcula usando la ecuación 5, sin embargo, la norma recomienda utilizar el residual de la señal de voz obtenido a través de la combinación con el filtro de síntesis. La señal residual

$$r(n) = s(n) + \sum_{i=1}^{10} \hat{a}_i s(n-i), n = 0, ..., 39 \tag{6}$$

se calcula para cada subtrama y se usa en la búsqueda del libro de códigos adaptivo.

Los parámetros del libro de códigos adaptivo son el retraso y la ganancia que representan al pitch. Para la implementación del filtro de pitch, la excitación se repite para los retrasos menores que el largo de la subtrama. En la etapa de búsqueda, la excitación se extiende por el residuo de LP para simplificar la búsqueda de lazo cerrado. La búsqueda del libro adaptivo se realiza cada 5 ms. Se usan valores iniciales de retraso de pitch para calcular los posteriores. La búsqueda de lazo cerrado de pitch se realiza minimizando el error cuadrático medio entre la señal original y la reconstruida

$$R(k) = \frac{\sum_{n=0}^{39} x(n) y_k(n)}{\sqrt{\sum_{n=0}^{39} y_k(n) y_k(n)}} \tag{7}$$

donde $x(n)$ es la señal target y $y_k(n)$ es la excitación filtrada pasada en el retraso k, que es la excitación pasada convuelta con $h(n)$.

$$y_k(n) = y_k(n-1) + u(-k)h(n), n = 39, ..., 0 \tag{8}$$

donde $u(n)$ es el buffer de excitación para n=-143,...,39.

Después de llevar a cabo una interpolación de R_k se genera el vector del libro de códigos adaptivo interpolando la excitación pasada de la señal $u(n)$ en cierto valor entero del retraso k y la fracción t:

$$v(n) = \sum_{i=0}^{9} u(n-k+i)b_{30}(t+3i) + \sum_{i=0}^{9} u(n-k+1+i)b_{30}(3-t+3i) \tag{9}$$

para $n = 0, ..., 39$ y $t = 0, 1, 2$. Una vez calculado el retraso del libro adaptivo se obtiene la ganancia

$$g_p = \frac{\sum_{n=0}^{39} x(n)y(n)}{\sum_{n=0}^{39} y(n)y(n)} \tag{10}$$

donde $y(n)$ es el vector del libro adaptivo filtrado que se obtiene de la convolución entre $v(n)$ con $h(n)$.

Se utiliza una cuantización por medio de MA (Moving Average Prediction) la cual, basada en la correlación de la señal de voz en si, protege los datos contra errores de transmisión. Sin embargo, esta protección es inherente al tipo de predicción y no fue diseñada específicamente para proteger a la señal de voz contra errores del canal. Es por eso que la implementación sugerida de COVQ como un apoyo real a la codificación conjunta canal-fuente llevando a cabo una cuantización de los parámetros de la voz optimada al canal es propicia.

3.1.4. Libro de Códigos Fijo

El libro de códigos fijo está basado en la estructura de un libro de códigos algebraico. Cada vector del libro contiene cuatro pulsos con valor distinto a cero cuya amplitud toma valores de 1 o -1 y puede asumir las posiciones mostradas en la figura 7 obtenida de [7]

Pulse	*Sign*	*Positions*
i_0	s_0: ± 1	m_0: 0, 5, 10, 15, 20, 25, 30, 35
i_1	s_1: ± 1	m_1: 1, 6, 11, 16, 21, 26, 31, 36
i_2	s_2: ± 1	m_2: 2, 7, 12, 17, 22, 27, 32, 37
i_3	s_3: ± 1	m_3: 3, 8, 13, 18, 23, 28, 33, 38
		4, 9, 14, 19, 24, 29, 34, 39

Figura 7: Posición de Pulsos- Vector del Libro de Códigos Fijo

El vector del libro $c(n)$ se construye colocando cada pulso en un vector de ceros y de dimensión 40, respetando sus respectivas posiciones y multiplicándolos por su signo.

$$c(n) = s_0\delta(n - m_0) + s_1\delta(n - m_1) + s_2\delta(n - m_2) + s_3\delta(n - m_3) \qquad (11)$$

para $n = 0, ..., 39$ donde δ representa un pulso unitario.

El procedimiento de búsqueda del libro fijo se lleva a cabo minimizando el error cuadrático medio entre la señal $s_w(n)$ y $x(n)$. Si c_k es el vector k-ésimo del libro fijo, el libro se obtiene maximizando:

$$\frac{C_k^2}{E_k} = \frac{(\sum_{n=0}^{39} d(n)c_k(n))^2}{c_k^t \Phi c_k} \qquad (12)$$

donde $d(n)$ es una señal de correlación obtenida de la señal target y la respuesta al impulso $h(n)$. Si la matriz H se define como la matriz triangular de convolución Toeplitz con diagonal $h(0)$ y diagonal menor $h(1), ..., h(39)$:

$$\Phi(i,j) = \sum_{n=j}^{39} h(n-i)h(n-j), i = 0, ..., 39; j = i, ..., 39 \qquad (13)$$

El denominador de la ecuación 12 está dado por:

$$C = \sum_{i=0}^{3} s_i d(m_i) \tag{14}$$

donde m_i es la posición del pulso i-ésimo y s_i su amplitud. La energía en el denominador de la ecuación 12 esta dada por:

$$E = \sum_{i=0}^{3} \Phi(m_i, m_i) + 2 \sum_{i=0}^{3} \sum_{j=i+1}^{3} s_i s_j \Phi(m_i, m_j) \tag{15}$$

Para simplificar el procedimiento de búsqueda, las amplitudes de los pulsos son predeterminados cuantizando la señal de correlación $d(n)$. Después de una serie de ajustes, la ecuacion de correlación se reescribe como:

$$C = |d(m_0)| + |d(m_1)| + |d(m_2)| + |d(m_3)| \tag{16}$$

y la de energía dividida entre 2 se conforma por la suma de los siguientes términos:

$$\Phi'(m_0, m_0) \tag{17}$$

$$\Phi'(m_1, m_1) + \Phi'(m_0, m_1) \tag{18}$$

$$\Phi'(m_2, m_2) + \Phi'(m_0, m_2) + \Phi'(m_1, m_2) \tag{19}$$

$$\Phi'(m_3, m_3) + \Phi'(m_0, m_3) + \Phi'(m_1, m_3) + \Phi'(m_2, m_3) \tag{20}$$

Se hace más simple la búsqueda realizando una aproximación en la que se usa un umbral para determinar si se obtiene o no el cuarto término de la energía. Así, el último lazo se calcula únicamente si la correlación absoluta debida a los tres primeros pulsos es mayor al umbral calculado.

Las posiciones de los tres primeros pulsos se codifican con 3 bits cada una, mientras que la del cuarto pulso con 4 bits. Definimos $s = 1$ si el signo es positivo y $s = 0$ si es negativo, el signo de la palabra código se obtiene de:

$$S = s_0 + 2s_1 + 4s_2 + 8s_3 \tag{21}$$

mientras que la palabra código se obtiene de:

$$C = (\frac{m_0}{5}) + 8(\frac{m_1}{5}) + 64(\frac{m_2}{5}) + 512(\frac{2m_3}{5} + jx) \tag{22}$$

donde $j_x = 0$ si $m_3 = 3, 8, ..., 38$, y $j_x = 1$ si $m_3 = 4, 9, ..., 39$.

Las ganancias del libro de códigos fijo y adaptable se codifican usando 7 bits minimizando el error cuadrático medio entre la señal original y la reconstruida.

La intención de este tipo de codificadores es la de eliminar la redundancia que existe en señales como la de voz reteniendo las características importantes de esta para disminuir así la tasa de datos. Al remover esta redundancia la señal se vuelve muy sensible al ruido del canal. Es por eso que se asignan índices binarios a los vectores de código de los cuantizadores, que a su vez otorgan protección contra errores del canal.

La justificación de utilizar codificación conjunta en la cuantización vectorial radica en que los JSCC, utilizando información estadística de las propiedades del canal, intenta mejorar los cuantizadores vectoriales (VQ) mediante un juicioso diseño de fuente-canal del libro de códigos y de la asignación de índices. En forma más explícita, predicen como será modificada la señal al pasar por el canal de voz.

La matemática del CS-ACELP es mucho más extensa de lo aquí descrito. Se ha expuesto a grandes rasgos la forma de trabajar de este codificador haciendo énfasis en la obtención del libro de códigos fijo ya que en él se enfocará el trabajo en cuestión. Para vislumbrar de mejor manera la forma de trabajar de este codificador algebraico puede referirse a [7].

3.2. Decodificador

La función del decodificador consiste en decodificar los parámetros, tanto del filtro LP como de los vectores de los libros de código y sus ganancias, y el llevar a cabo un proceso de síntesis para obtener la señal de voz reconstruida. Además deberá realizarse una etapa de pos-procesamiento que consiste de un filtro adaptivo y de un filtro pasa altas fijo.

Para la decodificación se regeneran los coeficientes del filtro LP usando los parámetros transmitidos. Después, para cada sub-trama, los vectores escalados de los libros de código adaptivo y fijo se suman y se filtran con el filtro LP de síntesis para generar la señal reconstruida de voz.

El post-procesado consiste de un post-filtrado adaptivo y de un filtrado pasa altas. El post-filtro se actualiza cada 5ms (para cada sub-trama). La señal pasa por un control de ganancia para igualar la energía de esta señal reconstruida con la recibida antes de la decodificación. Esta nueva señal pasará por un filtro pasa altas con frecuencia de corte en 100 Hz. El post-filtrado adaptivo consiste de tres filtros: el primero a largo plazo, el segundo a corto plazo y un tercero para compensar el llamado tilt en el segundo. Dentro de la determinación de los parámetros del filtro adaptivo se busca determinar si es necesario este tipo de filtrado para cada sub-trama. Los tres filtros del post-filtrado se acomodan en cascada en su orden de aparición descrito. Posteriormente, se lleva a cabo un control de ganancia para compensar la diferencia de ganancias entre la señal reconstruida y la señal obtenida del post-filtrado. Obtenido de [7] se muestra un diagrama del decodificador del CS-ACELP.

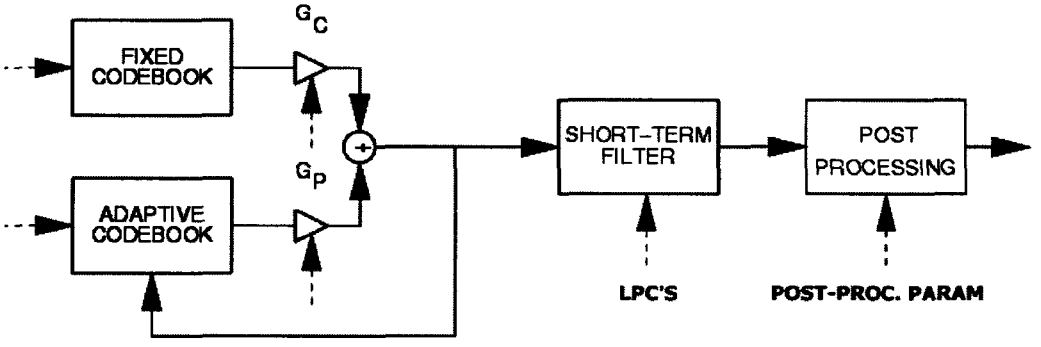


Figura 8: Decodificador

4. COVQ

Descrito en [6] y en [14] aparece un algoritmo para generar cuantizadores vectoriales basados en modelos probabilísticos o en una secuencias grandes de entrenamiento. El cuantizador puede utilizar medidas de distorsión generales y bloques de múltiple y gran tamaño. Aunque se describe el uso de ciertas medidas de distorsión para la convergencia hacia un cuantizador óptimo, en la implementación del cuantizador de este proyecto se utilizará una medida de distorsión descrita en [7] y expuesta matemáticamente más adelante en este escrito.

COVQ es útil para el diseño de cuantizadores de bloque y vectoriales para medidas de distorsión en un rango amplio. El algoritmo se basa en la aproximación de Lloyd y trabaja bien aún cuando la distribución tiene componentes discretos.

La finalidad de utilizar estos cuantizadores vectoriales radica en las propiedades de robustez que un parámetro dado adquiere cuando es cuantizado y transmitido a través de un medio en forma de índices digitales. Es por esta característica que se decidió optar por reemplazar alguno de los parámetros del CS-ACELP por un libro de códigos vectorial que, se espera, proveerá de una mayor protección contra errores de canal.

Para diseñar el COVQ se usa un algoritmo iterativo. De acuerdo a [14], para el cuantizador $\boldsymbol{q}$ y el índice de asignación $\boldsymbol{b}$. La distorsión promedio está dada por,

$$D(q;b) = \frac{1}{k}\sum_{i=1}^{M}\sum_{j=1}^{M} P(b(c_i)|b(c_i)) \int p(x)d(x,c_j) \tag{23}$$

donde el largo de las palabras de código es,

$$n = log_2 M \tag{24}$$

k es la dimensión del vector cuantizador x, c ($C = c_1, c_2, ..., c_M$) es el libro de códigos y $\boldsymbol{S}$ ($\boldsymbol{P}$=S_1,S_2,...,S_M) es el espacio de posibles valores que puede tomar $\boldsymbol{x}$. Esto para una $i = 1, 2, ..., M$. Este algoritmo itera actualizando el libro de códigos adaptivo C para una partición $\boldsymbol{P}$, seguida por una actualización de $\boldsymbol{P}$ para encontrar el libro de códigos fijo $\boldsymbol{C}$. Entonces la finalidad

será la de disminuir esta distorsión $\boldsymbol{D}$(q;b) obteniendo los valores óptimos de los libros de código $\boldsymbol{c}_i$ (adaptivo) y $\boldsymbol{c}_j$ (fijo) que usa el CS-ACELP.

La aportación del COVQ radica en que la distorsión D(q;b) representa tanto la distorsión promedio por muestra,

$$Ds(q) = \frac{1}{k}\sum_{i=1}^{M}\int_{S_i} p(x)d(x,c_i)dx \tag{25}$$

como la distorsión por muestra ocasionada por el canal,

$$Dc(b) = \frac{1}{k}\sum_{i=1}^{M}\sum_{j=1}^{M} P(c_i)P(b(c_j)|b(c_i))d(c_i,c_j) \tag{26}$$

donde $\boldsymbol{p(x)}$ es la k-ésima densidad de probabilidad de la fuente, $\boldsymbol{d(x,c_i)}$ es una medida no negativa de distorsión, $\boldsymbol{P(j/i)}$ es la probabilidad de que el índice j sea recibido dado que se transmitió el indice i, $\boldsymbol{b}$ es el vector de índices y $\boldsymbol{P(c_i)}$ es la probabilidad apriori de la palabra de código $\boldsymbol{c}_j$.

Es entendible que las formas recursivas de búsqueda en la selección de los vectores de los libros de código tendrán que cambiar.

El algoritmo de cuantización vectorial para el caso en el que se cuenta con una distribución probabilistica conocida es el que nos interesa. Son cuatro las partes principales del algoritmo:

- ***Inicio***: Dado un número $\boldsymbol{N}$ de niveles, tenemos un umbral de distorsión mucho mayor que cero y un alfabeto inicial $\boldsymbol{A}_0$ de $\boldsymbol{N}$ niveles. Contamos con una distribución que define al canal y con la constante m igual a 0 y $\boldsymbol{D}_{-1}$ que tiende a infinito.

- Dado un alfabeto $\boldsymbol{A}_m$ debemos encontrar la partición de distorsión mínima $\boldsymbol{P[A_m]}$

  $$P(\hat{A}_m) = S_i; i = 1, ..., N : x \in S_i \tag{27}$$

  si la medida de distorsión calculada es menor a las anteriores. Se calcula la medida de distorsión promedio resultante.

- Si,

  $$(D_{m-1} - D_m)/D_m \leq umbral \tag{28}$$

termina el proceso iterativo de búsqueda donde la partición $\boldsymbol{P[A_m]}$ definirá el cuantizador final. De otra manera, continuar.

- Calcular la reproducción óptima del alfabeto para $\boldsymbol{P[A_m]}$. Se establece,

$$\hat{A}_{m+1} = \hat{x}(P(\hat{A}_m)) \tag{29}$$

y reemplazamos m por $m + 1$ y volver al punto 2.

Si en algún punto la partición óptima contiene alguna célula cuya probabilidad sea cero, uno pudiera eliminar esa célula y continuar con un algoritmo que produjera un cuantizador de ***N-1*** vectores. Sin embargo, algo que soluciona el problema y que se llevó a cabo en esta tesis fue el cambiar la partición inicial de tal forma que todas las células tuvieran una participación o cambiando la célula inicial por una que sea distinta a las demás y que garantice una probabilidad.

La metodología iterativa de búsqueda mostrada se define en el ***CAPÍTULO 6*** haciendo más énfasis en la distribución probabilística del canal en la medida de distorsión utilizada para encontrar el mínimo.

La implementación del COVQ se dará en simulación sobre MATLAB. El exponer a grandes razgos los bloques principales del codificador no tiene más finalidad que el mayor entendimiento del escrito para el lector. La verdadera aportación al tema se da por la adecuación del JSCC a la norma del CS-ACELP, además de su implementación en software haciendo uso de un canal de fibra óptica a ser pepresentado por un modelo matemático de ciertas características a especificar. La características del modelo tendrán una relación estrecha con la creación de los libros de código y la asignación de sus índices usando codificación conjunta.

5. Modelo del Canal de Fibra

El modelo matemático de fibra óptica a utilizar deberá describir características del canal que son ya conocidas. Varios son los puntos a tomar en cuenta en un modelo de fibra óptica, refiriéndose, principalmente, a la forma en la que la fibra óptica degrada la señal:

- Respuesta a encendido y apagado descrita con una respuesta lineal
- Memoria; característica que puede ser representada con respuesta lineal si se supone que el efecto del bloque es dependiente de la frecuencia usando,

$$C_0 \frac{dT}{dt} = (\frac{T}{R_0}) + V_j i_L(t) \tag{30}$$

  donde $\boldsymbol{R_o}$ y $\boldsymbol{C_o}$ son la resistencia y capacitancia térmicas respectivamente, $\boldsymbol{V_j}$ es el voltaje del LED cuando está encendido (generalmente constante), $\boldsymbol{T}$ es la temperatura en el LED e $i_L(t)$es la corriente que pasa por el LED en el tiempo.

- Pulsación del Laser, que se refiere a las oscilaciones sostenidas del laser (mayores con la edad del dispositivo).
- Reflexiones, producidas por brechas de aire entre conductores de fibra.
- Espectro, generalmente modelado como una distribución Gaussiana Normalizada, aunque el modelo Gaussiano generalmente se desvia drásticamente del espectro del laser,

$$S(\lambda) = \frac{1}{\gamma}\sqrt{2\Pi}\frac{exp(-(\lambda - \lambda_s))}{2(\gamma - 2)} \tag{31}$$

  donde $\boldsymbol{s}$ es el ancho de onda.

- Pérdidas debidas a laser,

$$L_{LED}(\lambda) = -\frac{10}{l} log_{10} \int S(\lambda) 10^{lL(\lambda)/10} d(\lambda) \tag{32}$$

  donde $\boldsymbol{l}$ es el largo de la fibra

- Dispersión, dependiente del modo de la fibra (mono o multi-modo).
- **Ruido Modal**

Las características del canal y la forma en la que este modifica a la señal pueden ser representadas por bloques en casacada que definan cada uno de los impedimentos de transmisión ya mencionados.

La complejidad matemática de esta tesis radica grandemente en acoplar la función de tranferencia del canal de tal forma que pueda ser utilizada por el COVQ dentro del codificador CS-ACELP.

Tomando en cuenta los impedimentos de transmisión propios de un canal de fibra se utilizó un modelo que se apega a un canal de fibra óptica seccionado y con empalmes (splices) de una longitud determinada y que provee de un arreglo matemático que reproduce los efectos del **ruido modal**.

5.1. Características de un canal con Ruido Modal

Para el modelo del canal de fibra se introducirá ruido modal como una distribución estocástica de intensidad de campo situada alrededor del radio de la fibra. El modelo del canal es uno muy simple en el que contamos con un tranmisor, una sección de fibra unida por medio de un conector a una segunda sección de fibra que se une al receptor. El ruido modal aparece durante la conversión de la frecuencia de modulación del laser a modulación de intensidad, cuando un conector desacopla dos sectores de fibra óptica y en fibras multimodo.

El modelo utilizado para este trabajo es el propuesto por [4]. En la Figura 9 se puede observar el modelo de comunicación de fibra óptica y el lugar en donde el ruido modal se hace presente.

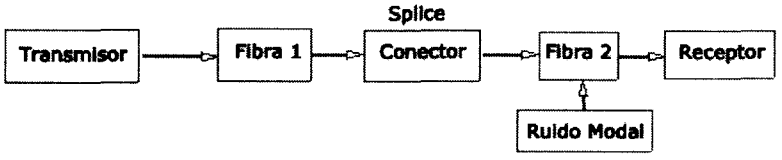


Figura 9: Modelo de comunicación de Fibra Optica

Como muestra la Figura 9, el conector tiene un *splice*, o empalme de fibra, de $\boldsymbol{L}$ metros de longitud. El modelo cuenta con un circuito tranmisor y un receptor, entre ellos se encuentra una resistencia típica de la fibra denominada $\boldsymbol{R}$.

Ruido termal es añadido a la señal del detector y se forma la señal usada para decodificación digital según [4], donde la señal observada después de $\boldsymbol{T}$ segundos de integración es

$$v = \frac{qR}{T}n + n_T \tag{33}$$

donde q es la carga del electrón, n el número de electrones en el tiempo T, y n_T es el ruido termal del circuito detector de voltaje de señal al final de la fibra.

Las literales, ecuaciones y valores expuestos serán utilizados y explicados más a fondo en el ***CAPÍTULO 5.1.1*** donde se llevará a cabo el cálculo de probabilidades de error y de transición.

5.1.1. Probabilidad de Error en presencia de Ruido Modal

Para hacer uso de la codificación conjunta es necesario contar con las probabilidades de transición de los índices a transmitir a través del canal. Estas probabilidades se obtienen a partir de la probabilidad de error de transmisión de un solo bit a lo largo de la fibra.

La probabilidad de error de transmisión de un solo bit se obtiene de la probabilidad de transmitir un 1 o un 0 y de las probabilidades condicionales de transmitir un 1 y recibir un 0 y viceversa.

$$Pe = P(0\backslash 1)P(1) + P(1\backslash 0)P(0) \tag{34}$$

Se le dio la misma probabilidad de transmisión a un valor de uno que a uno de cero. Por lo que procedemos a buscar las probabilidades de transición. Estas se definen como [4]:

$$p_1(v\backslash\lambda) = \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(v-\bar{v})^2}{2\sigma^2}} \frac{K_1(A+m)^{2n}}{n!} e^{-K(A+m)^2} \tag{35}$$

$$p_0(v\backslash\lambda) = \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(v-\bar{v})^2}{2\sigma^2}} \frac{K_1(m)^{2n}}{n!} e^{-K(m)^2} \tag{36}$$

Usando Matlab se puede obtener una probabilidad de error por bit y a partir de esta, generar una tabla de probabilidades de transición. El cálculo de estos parámetros se encuentran en el **APENDICE A** y, principalmente, en el ***CAPÍTULO 6*** en forma de simulación. La solución de las ecuaciones 35 y 36 se dan por medio de matlab tomando en cuenta varios puntos.

Algunas de las literales a tomar en cuenta tienen explicación en la siguiente tabla tomando valores típicos o se pueden definir por distribuciones conocidas. Los valores mostrados a continuación fueron obtenidos de la referencia [4].

Tabla 1 *Descripción de Literales*

A	Amplitud de la señal óptica recibida	1 volt
Lambda	Valor promedio de eléctrones en el tiempo T	1-0
K1	cte deacuerdo a la respuesta del detector	0.8
Rk	cte deacuerdo a la respuesta del detector	0.85
m	Ruido modal	Aleatorio
R	Resistencia de la fibra	50 Ohms
q	Carga del electrón	1.6021892e-19 C
nT	Ruido termal del circuito	Aleatorio
B	Ancho de Banda	1.668 Gbps
Fibra	FT-G-1.7	1.3 micro$m$
L	Longitud en Kilómetros sin repetidor	menor a 46
	Canales de voz	24192

De las ecuaciones 35 y 36 podemos observar la literal $\boldsymbol{m}$ representando al ruido modal como una variable formada por números aleatorios generado por una fuente de ruido blanco gaussiano con una varianza de 0.09. Se observa que para obtener las probabilidades condicionales se debe variar el número de electrones $\boldsymbol{n}$ pasando a través del canal. Otro valor a tomarse en cuenta son los momentos de la distribución gaussiana para la variablo $\boldsymbol{v}$.

Esta variable esta definida en la ecuación 33. Los momentos gaussianos están definidos como,

$$\bar{v} = \frac{qR}{T} \tag{37}$$

$$\sigma^2 = 4kBR\Theta \tag{38}$$

donde k es la constante de Boltzman y Θ se refiere a temperatura en grados kelvin. Se acotaron las sumatorias definidas en las ecuaciones 35 y 36 hasta valores de n dependiente de ciertos valores de amplitud, resistencia, entre otros. Se tomó la ecuación 33, se supuso un valor nulo de ruido termal, se igualo la resistencia a 50 Ohmios y la amplitud a 5 volts y se obtuvo un valor de n electrones de $3{,}9009 \times 10^{13}$. Este valor es que acota el límite superior de la sumatoria de las ecuaciones 35 y 36.

La probabilidad de error por bit obtenida de la simulación arrojó el valor mostrado en la tabla siguiente.

Tabla 2 *Probabilidad de Error por Bit*

BER	$1{,}6281 \times 10^{-6}$

Con este valor de BER se pudo trabajar y modelar el canal de fibra. El valor de probabilidad de error para un solo bit provocado por un canal de fibra con distorsiones por ruido modal y la forma en la que fue calculada está bien documentado y puede corroborarse en las referencias [4] y [8].

Las probabilidades de transición se obtuvieron tomando en cuenta que dado un vector con 3 bits, la probabilidad de correcto de este es la multiplicación de las probabilidades de correcto de cada uno de sus bits. Entonces, para generar la tabla de probabilidades de transición se hace lo siguiente,

Tabla 3 *Probabilidades de Transición*

Tx-Rx	000	001	010	.	.
000	(1-Pe)(1-Pe)(1-Pe)	(1-Pe)(1-Pe)Pe	(1-Pe)Pe (1-Pe)	.	.
001	(1-Pe)(1-Pe)Pe	(1-Pe)(1-Pe)(1-Pe)	(1-Pe)(Pe)(Pe)	.	.
010	(1-Pe)Pe(1-Pe)	(1-Pe)(Pe)(Pe)	(1-Pe)(1-Pe)(1-Pe)	.	.
.	.	.	.	.	.
.	.	.	.	.	.

La tabla anterior muestra la forma de generar la tabla de probabilidades de transición, que para el estudio, se refiere a una matriz cuadrada de 512 por 512, lo que representa vectores de 9 bits.

Aunque para nuestros fines el cálculo de la probabilidad de error por el medio mostrado es suficiente, existe otra forma de encontrarlo mencionada en [4]. Si no contamos con las probabilidad de error por bit, podemos calcularla usando la ecuación,

$$BER = P(H1)\int_{-\infty}^{V_T} p_1(v)dv + P(H0)\int_{V_T}^{\infty} p_0(v)dv \tag{39}$$

Para la que debemos determinar un valor umbral de detección de voltaje V_T, donde ***P(H1)*** y ***P(H0)*** son las probabilidades de transmitir uno o cero respectivamente y pueden ser diferentes o idénticas como en el caso que definimos anteriormente. La verdadera problemática radica en el trato matemático que se le debe de dar a las integrales que están definidas por las ecuaciones,

$$p_1(v) = \int_{-\infty}^{\infty} \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(v-\bar{v})^2}{2\sigma^2}} \frac{K_1(A+m)^{2n}}{n!} e^{-K(A+m)^2} \frac{1}{\sqrt{2\pi}\sigma_m} e^{-\frac{m^2}{2\sigma_m^2}} dm \tag{40}$$

$$p_0(v) = \int_{-\infty}^{\infty} \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(v-\bar{v})^2}{2\sigma^2}} \frac{K_1 m^{2n}}{n!} e^{-Km^2} \frac{1}{\sqrt{2\pi}\sigma_m} e^{-\frac{m^2}{2\sigma_m^2}} dm \tag{41}$$

Se formulan dos hipótesis asumiendo que se recibe 1 o 0 y están dadas por,

$$H1 : \lambda = K(A + m)^2 \tag{42}$$

$$H0 : \lambda = Km^2 \tag{43}$$

donde A esta definida como la amplitud de la señal optica recibida y m corresponde al ruido modal.

Para las demás ecuaciones, la varianza del ruido modal se define como,

$$\sigma^2 = 2P_0^2\eta^2(1-\eta)^2 10^{-\frac{L\alpha}{10}} \tag{44}$$

la eficiencia de transmisión es,

$$\eta = 10^{-\frac{\eta_0}{10}} \tag{45}$$

donde η_0 es la pérdida en el *empalme* (división de fibra) en dB, la potencia óptica ejercida promedio está dada por,

$$P_0 = \frac{b_0 + b_1}{2} \tag{46}$$

donde b_1 y b_0 representan la potencia óptica del bit uno y cero respectivamente.

Ya resuelta la ecuación 39 se pueden calcular las probabilidades de transición utilizando el programa *probtrans.m* descrito en el **CAPÍTULO 6**.

Si se cuenta con la probabilidad de error por bit del canal de fibra, que generalmente se encuentra alrededor de 1×10^{-9}, y no con el valor de voltaje umbral, se debería utilizar la ecuación 39 para despejar V_T y utilizar

ese valor umbral en la detección de errores que se describirá en el ***CAPÍTULO 6.4***.

Este arreglo matemático para la búsqueda del BER resultó muy exhaustivo y las ecuaciones a las que se llegaron resultaron de resoulución complicada. Algunas de las integrales que se generaron pudieron ser resueltas por partes, sin embargo, otras quedaron sin resolver y no se les dio seguimiento exhaustivo para encontrar un método de solución. Dado que el trato matemático de la investigación no se centraba en esta parte del trabajo, se optó por utilizar el primer método para encontrar la probabilidad de error por bit.

6. Creación de un libro de Códigos

Para el codificador CS-ACELP se utilizan dos libros de códigos algebraicos, uno fijo y uno adaptivo. Al utilizar predicción MA para generar los parámetros que conforman estos libros, obtenemos cierto nivel de codificación de canal y a su vez de codificación conjunta dado que el codificador también cuenta con codificación de fuente. Sin embargo, los impedimentos de transmisión para un canal en específico no se toman en cuenta, buscando tan solo minimizar el error cuadrático medio y no las características dañinas inherentes de un canal específico.

El objetivo de este trabajo es el de optimar el desempeño del codificador para un canal de fibra óptica con ciertas características que distorsionan la señal de voz transmitida. De forma más específica, utilizaremos un modelo del canal de fibra que degrada la señal debido a ruido modal.

El ruido modal aparece en el radio de la fibra como una distribución estocástica de intensidad de campo . Este fenómeno ocurre en la conversión de la modulación de la frecuencia del laser a la modulación de su intensidad y en fibras con divisiones que aparecen en canales que tienen una longitud considerable.

Son varios los parámetros del codificiador CS-ACELP que podrían optimarse de acuerdo a un modelo de canal de fibra. En este caso se propone el reemplazar el libro de códigos fijo, que se obtiene de forma algebraica a partir de índices, signos y ganancias, por una tabla de valores de determinado tamaño que, tentativamente, deberá disminuir el margen de error entre la señal original y la reconstruida. Esta tabla se obtendrá por medio de cuantización vectorial optimada de canal (COVQ).

6.1. Cálculo del Libro de Códigos Fijo

Utilizando COVQ [6] se puede obtener un libro de códigos óptimo que represente el residual obtenido de pasar la señal de voz a través del filtro A(z) definido en la norma del CS-ACELP [7]. Este libro de códigos estará formado por valores típicos codificados del residual en cuestión.

Para obtener el libro de códigos por medio de COVQ, necesitamos de una secuencia de entrenamiento de considerable tamaño. Se utilizará un archivo con muestras de voz provenientes de hombres y mujeres con diferentes timbres de voz, que articulan frases múltiples.

Para generar el libro de códigos debemos trabajar con una medida

de distorsión adecuada. Una medida comunmente utilizada es la del error cuadrático medio. Existen otras medidas de distorsión comunes, sin embargo, debemos escoger aquella que tome en cuenta los impedimentos de transmisión del canal de fibra óptica que se desean modelar.

La medida de distorsión a utilizar está dada en base a las transiciones de probabilidad de los índices que representan al libro de códigos. Esta se define como [5]:

$$D_i = \frac{1}{K}(P_i M_i + b_i - 2\mathbf{a}_i^b \mathbf{c}_i) \tag{47}$$

$$\mathbf{a}_i = \sum_{\hat{i}=0}^{N-1} \sum_{\hat{j}=0}^{N-1} P_j \mathbf{c}_j \frac{P_{\hat{I}|I}(\hat{i}|j)}{P_{\hat{I}}(\hat{i})} P_{\hat{I}|I}(\hat{i}|i) \tag{48}$$

$$\mathbf{b}_i = \sum_{\hat{i}=0}^{N-1} \sum_{\hat{m};n=0}^{N-1} P_m \mathbf{c}_m^T \mathbf{c}_n P_n \frac{P_{\hat{I}|I}(\hat{i}|m) P_{\hat{I}|I}(\hat{i}|n) P_{\hat{I}|I}(\hat{i}|i)}{(P_{\hat{I}}(\hat{i}))^2} \tag{49}$$

El vector $\boldsymbol{M_i}$ se conforma como el promedio de cada uno de los vectores codificados por la misma partición. $\boldsymbol{c_i}$ se refiere a los centroides de las particiones del libro de códigos que se estan generando.

6.1.1. Codificador Vectorial

Para la obtención de un codificador vectorial que reemplace a la exitación del libro de códigos fijo del CS-ACELP necesitamos una secuencia de entrenamiento. Se utilizó la base de datos YOHO. Para la secuencia de entrenamiento se usaron dos archivos de voz de cada persona. Veinte individuos para la secuencia formando un total de 60 archivos de voz, treinta de voz femenina y treinta de masculina. Cada archivo de voz tiene una duración promedio de tres segundos.

Para la obtención del codificador óptimo, que será una tabla de 512 vectores de tamaño 80, (40 muestras por cada subtrama) se siguen 5 pasos iterativos [5].

- **Inicio** Se toma una distorsión inicial D_0=constante grande. Se selecciona una partición inicial P_0. Se establece un final de algoritmo $\boldsymbol{d>0}$. Se iguala el índice de iteración $j = 0$. Es importante recalcar que la partición inicial se tomó como una matriz de 512 vectores de tamaño 80, los cuales se colocaron en orden ascendente.

En principio, la partición inicial fue obtenida de los promedios máximos y mínimos de los residuales de la secuencia completa de entrenamiento a través del filtro LP. Esta aproximación a la partición inicial necesaria para correr el proceso iterativo de la obtención de un libro de códigos vectorial óptimo dio como resultado una distribución de los vectores residuales cuantizados muy dispar. Los vectores residuales cuantizados utilizando esta matriz inicial tendían a agruparse sobre un mismo grupo de vectores iniciales y el sistema iterativo no tendía a una minimización de la distorsión expuesta en la ecuación 40. Fue por ello que se optó por un camino diferente.

Dado que el uso de un rango de promedios máximos y mínimos de residuales de la secuencia de entrenamiento como partición inicial en búsqueda del libro de códigos vectorial resultó poco adecuada, se buscó otro tipo de partición inicial. Se obtuvo un rango mínimo y máximo del error cuadrático medio y en lugar de utilizar vectores con valores promedio como parte de la partición se usaron 512 vectores residuales de la misma secuencia de entrenamiento acomodados de acuerdo a su medida de error cuadrático medio como partición inicial. Esta modificación a la partición inicial realzó las cualidades de optimación del COVQ y tras varias horas de iteración se llegó a un resultado numérico para la ecuación 47 que convergió en un valor aceptable.

- **Probabilidades** Se obtienen las probabilidades de la partición $\boldsymbol{P_i}$ y de los centroides.

- **Parámetros** Se calculan las ecuaciones 48 y 49

- **Cálculo de Distorsión** Se incrementa en uno el índice de iteración j y se calcula la distorsión de la ecuación 47. Si $\boldsymbol{(D_{j-1}\text{-}D_j)/D_j < d}$, se ha encontrado el codificador óptimo. En caso contrario, continuar al siguiente paso.

- **Partición Optima** Calcular la nueva partición óptima $\boldsymbol{P_j}$. Esta nueva partición se obtiene de la suma de los residuales que han sido codificados por vector de partición anterior y dividiendo entre el número de residuales codificados por este. Volver a **Probabilidades**.

La obtención de un resultado numérico para la ecuación 47 a partir de las ecuaciones 48, 49 y de $\boldsymbol{M_i}$ se lleva a cabo de una forma iterativa exhaustiva.

Las probabilidades condicionales encontradas en las ecuaciones 41 y 42 son obtenidas de la tabla de probabilidades de transición para el canal de fibra cuya descipción detallada se encuentra en el ***CAPÍTULO 5.1.1***. La matriz definida en estas ecuaciones es una cuadrada de 512 columnas por 512 renglones.

Las probabilidades de los índices se van ajustando para cada ciclo anidado del proceso para obtener el cuantizador vectorial. Esto es, si un índice cuantiza 50 vectores residuales de voz de entre 5000 de una secuencia de entrenamiento, por ejemplo, su probabilidad sería de 0.01.

La probabilidad de cada vector $\boldsymbol{P}_i$ que aparece en la ecuación 40 está definida por la dimensión del libro de códigos vectorial a generar. En este caso es un libro de 512 valores por 80. Esto nos da una probabilidad para toda i de 1 sobre 512 o 0.001953125.

Los centroides se obtienen del promedio de la suma de todos los vectores residuo que hayan sido cuantizados por cierto índice en particular.

Se puede observar la complejidad computacional de procesamiento para lograr un resultado numérico para la ecuación 47. Tan solo dentro de la ecuación 48 existen tres ciclos anidados de 512 iteraciones cada uno

Por medio de COVQ se minimiza la distorsión propuesta en la ecuación 47 y la tabla obtenida suplantará aquella definida en la norma, calculada de forma algebraica. La programación en Matlab para la obtención de las probabilidades de transición y del libro de códigos se encuentra en el **APENDICE A**.

Para la codificación vectorial se encontró el rango promedio de la media de todos los archivos de voz y su distribución se muestra en la gráfica siguiente.

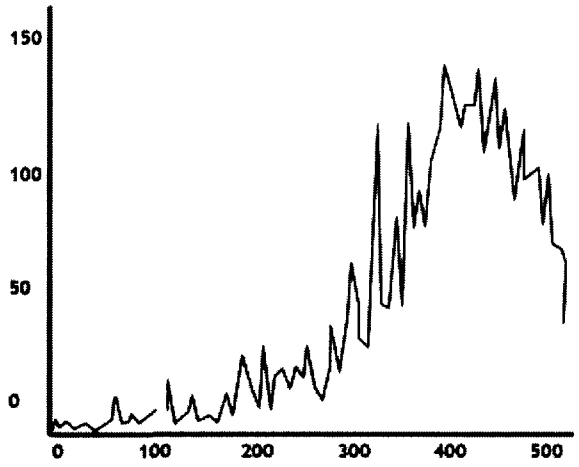


Figura 10: Distribución de Vectores de Voz

Ahí se pueden observar cuantos vectores de voz fueron codificados por cada vector de codificación. Para medir la distorsión promedio de cada vector de voz para su codificación se utilizó la medida definidas por las ecuaciones 47, 48 y 49.

Esta depende de la probabilidad de error y de las probabilidades de transición de los bits al pasar a través del canal de fibra.

La distorsión final fue de .349, la distorsión entre el ciclo anterior al final es del orden de 10^{-15} y bastaron 50 ciclos anidados para llegar al resultado. Un total de 75 horas seguidas de procesamiento fueron necesarias para llevar a cabo la síntesis del proceso iterativo. Se utilizó un procesador PENTIUM 4 a 2.2 GHZ con 256MB de memoria RAM. El equipo utilizado fue una DELL inspiron 8200.

Dado que pudiera resultar complicado para el lector vislumbrar la operación del COVQ en forma escrita, a continuación se muestra un diagrama de bloques donde se explica el curso que siguen los residuales para generar el libro de códigos fijo óptimo para un canal de fibra con ruido modal.

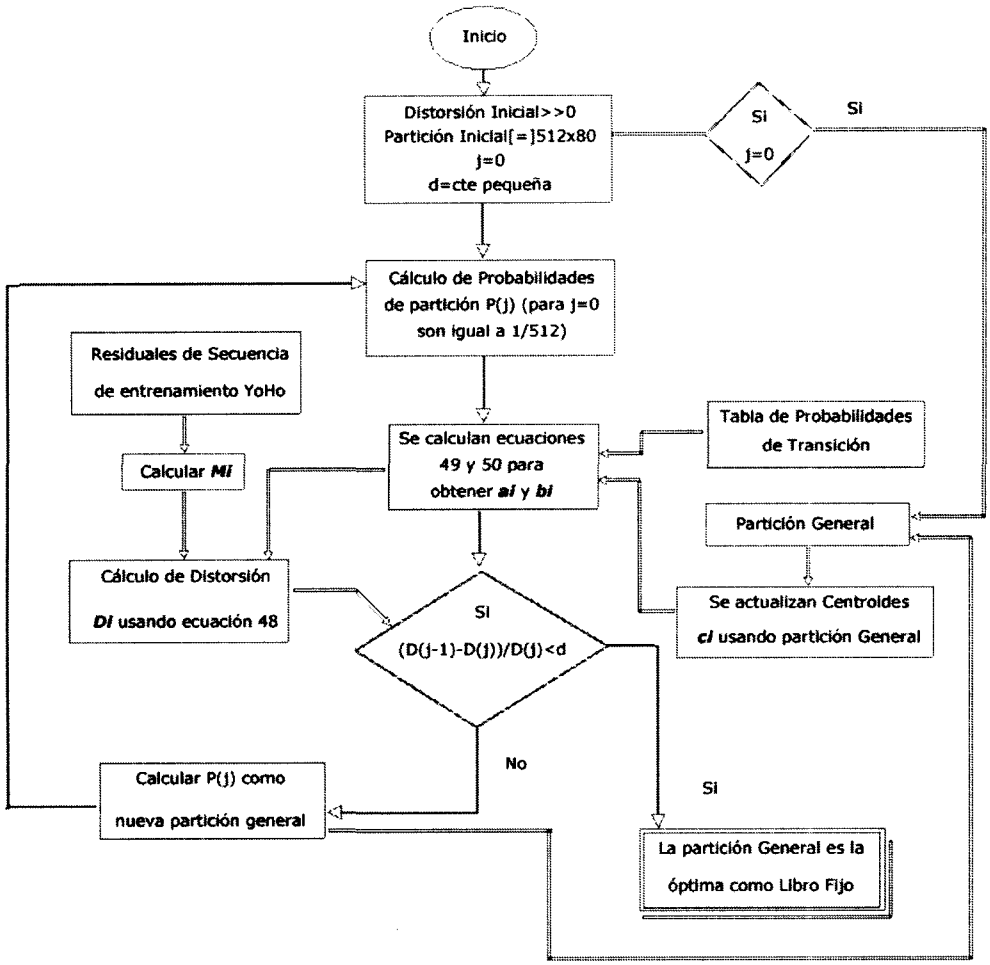


Figura 11: Búsqueda exhaustiva - Libro de Códigos

El diagrama a bloques representa gráficamente los pasos que se siguieron para encontrar la partición que representa el libro de códigos fijo que reemplaza al libro de códigos vectorial del CS-ACELP original. Las flechas más grandes indican el recorrido normal de la búsqueda exhaustiva mientras que las líneas pequeñas indican que existen momentos en que ciertas matrices deben ser actualizadas y operaciones se deben de llevar a cabo para realizar algun cálculo específico de la búsqueda.

Como se explicó, se da inicio especificando una partición inicial que actualiza a la partición general y que en algún momento de la búsqueda exhaustiva se convertirá en la partición óptima que representará al libro de códigos fijo de excitación. Se procede con el cálculo de las probabilidades P_j, que para lazos más avanzados de la búsqueda, dependerán de cuantos vectores residuo codifique cada vector de la partición general.

Se calculan las ecuaciones 48 y 49 haciendo uso de la tabla de probabilidades de transición, que debió ser previamente calculada, y de los centroides actualizados de la partición general presente. El paso siguiente es hacer un cálculo de D_i actualizada por medio de a_i, b_i y M_i que a su vez hace uso del promedio de los residuales de la secuencia de entrenamiento.

Si $(D_{j-1}\text{-}D_j)/(D_j)<d$ es menor que la distorsión mínima aceptable por la partición a obtener podemos decir que la partición general con la que contamos es la óptima, en caso contrario procedemos al siguiente paso.

Se calculan la nueva partición general usando como sus vectores la sumatoria de los residuales de la secuencia de entrenamiento codificados por la partición general anterior dividiendo cada vector partición entre el número de residuales que codificó el vector partición anterior. Esta nueva partición general actualiza la matriz de partición general y el ciclo comienza a partir del cálculo de las probabilidades de la partición general. Estas probabilidades dependerán también de cuantos vectores residuales hayan sido codificados por cada vector partición anterior. Se incrementa j en uno para este nuevo ciclo siguiendo un lazo exhaustivo hasta obtener una partición óptima.

Generalmente, el entrenamiento del cuantizador vectorial lleva un proceso en el que los residuales se van calculando trama a trama conforme se vayan necesitando. Para esta investigación, por cuestión de simplicidad y tiempo de procesamiento, se obtuvieron todos los residuales de excitación del libro de códigos y se acomodaron en un archivo de texto.

El cálculo de los residuales se llevó a cabo para las señales de voz ya mencionadas en un solo paso haciéndolas pasar por el filtro LPC tal y como lo índica el algoritmo de la norma para el G729. Los archivos de voz se unieron en un solo archivo de voz de tal forma que la memoria del filtro LPC nunca se llevó a cero por lo que el filtro percibió siempre el vector de voz como uno solo. Cabe mencionar que cada vector de voz perteneciente a cada uno de los hablantes contenía al principio y al final de ellos uno o más segundos de silencio en presencia de ruido. El único momento en el que el filtro LPC contaba con una memoria con valores de cero fue al calcular la primer trama de residuales.

Los valores de memorias para matrices y vectores correspondientes para el COVQ varian como se define en la descripción del algoritmo y en la figura 11. El cálculo en un solo bloque de los residuales evitó un mayor tiempo de procesado para cada uno de los entrenamientos del cuantizador realizado. Dado que la partición inicial óptima tardó en encontrarse este proceso tuvo llevarse a cabo en varias ocasiones por lo que el contar con los residuales agrupados en un solo archivo para disponer de ellos a placer simplificó la búsqueda del libro de códigos vectorial óptimo.

En lo que al filtro de síntesis se refiere, la memoria para la primer trama se mantiene en cero. Los valores de memoria acumulados en para la trama en cuestión serán los utilizados por la trama siguiente y a su vez la trama subsecuente a esta usará la memoria de la anterior. La memoria del filtro de síntesis no vuelve a borrase, es decir, no vuelve a tomar valores de ceros al menos de forma arbitraria. Esto para lograr que la resconstrucción de una trama a otra sea lo más suave y fiable posible. Esto sucede de igual manera para la generación de los residuos durante el proceso de búsqueda del libro de códigos vectorial

7. Manejo del Software

Tenemos un canal de fibra afectado por ruido modal y enviamos a través de este una señal de voz modulada con PCM a 16 bits. Codificamos la señal con el algoritmo del CS-ACELP y se recupera la señal usando el algoritmo de Matlab que ya se ha mencionado con anterioridad.

La señal de voz modulada con PCM debe ser codificada con CS-ACELP para protegerla contra errores del canal y para comprimirla de tal forma que transmitamos más información utilizando menos recursos. Se usan dos sistemas de codificación, el ya conocido G729 de la ITU-T y una versión modificada que reemplaza el libro de códigos fijo del primero por uno vectorial que servirá como reemplazo a la excitación.

Los parámetros de codificación de ambos sistemas pasarán a través de un canal de fibra que distorsiona a la señal con ruido modal debido a un empalme defectuoso que se encuentra en un conector situado en una de las uniones de un segmento de fibra con otro.

La señal codificada llega a su destino modificada, se decodifica usando dos métodos. El primer método dispone del decodificador de la norma del G729 mientras que el segundo hace uso de un decodificador que asemeja al primero bajo una plataforma de simulación llamada Matlab.

El proceso de comunicación se pudo observar en la figura 3, sin embargo, a continuación se muestra el trato que se le dio a las señales, parámetros, matrices y vectores para la simulación del sistema.

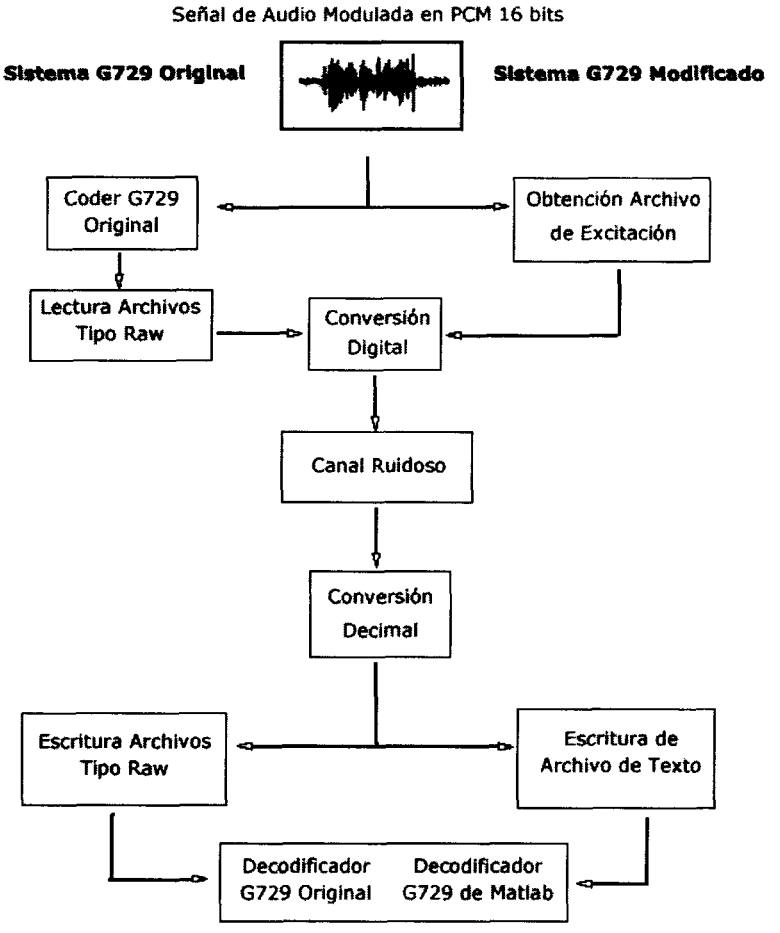


Figura 12: Diagrama de Bloques- Sistema de Comunicación

La figura 12 explica de forma muy resumida la forma en la que se tuvo que trabajar con el software para simular el sistema de comunicaciones. Como ya se ha explicado y se definirá de forma más explícita, se utilizaron dos sistemas de codificación. El primero basado en la norma del CS-ACELP, el segundo modificado por un archivo de excitación que se lee en formato de texto.

El diagrama de bloques muestra a grandes rasgos lo que se presentará en este capítulo. La señal de voz modulada con PCM a 16 bits se codifica

con CS-ACELP, los parámetros obtenidos se utilizan para generar un archivo de excitación que cuantiza la excitación de la señal de voz codificada además de sus funciones regulares. Esta obtención del archivo de excitación se lleva a cabo en Matlab y no por medio de la extracción de los parámetros del codificador original de la norma del G729 pero se sigue la misma fórmula para generarlos.

Los parámetros obtenidos del G729 original se leen por medio de Matlab se transforman en pulsos binarios, después en pulsos de LED y se les agrega ruido modal con cierta varianza. Después se transforman nuevamente en valores decimales para convertirlos en un archivo tipo raw para su decodificación posterior.

Como se mencionó los parámetros obtenidos de la señal de voz, específicamente los LPC, se utilizan para generar residuales que por medio de Matlab son comparados con el codificador fijo vectorial obtenido ya con anterioridad en un proceso de selección. Los vectores del libro de códigos vectorial seleccionados por su similitud con los residuales de la señal de voz son codificados y cuantizados. Aunque en el diagrama se utiliza un solo bloque para la conversión digital de todos los parámetros el trato es muy distinto para los dos. Los parámetros convertidos a pulsos binarios se transforman en pulsos de LED y son distorsionados por ruido en la misma forma que los demás parámetros.

Los vectores de excitación se generan convirtiendo estos residuales en decimales nuevamente para posteriormente guardar un archivo en formato de texto y utilizarlo en la decodificación.

Existen, como se explicará a detalle, dos programas de decodificación usados en este trabajo. El archivo de excitación y los demás parámetros de codificación serán decodificados por ambos programas pero los resultados serán distintos como se expondrá.

Inicialmente, para este proyecto se esperaba modificar la norma original del G729 de tal forma que la obtención de resultados fuera más transparente y no se tuviera que llegar a desarrollar una nueva replica del codificador CS-ACELP en algún otro lenguaje de programación para un trato más sencillo de gráficas o valores.

Dentro de este capítulo se presentan dos ramas de software diferentes: una más enfocada a lenguaje C y otra a Matlab. Sin embargo, para la obtención de los resultados se debieron utilizar ambos lenguajes computacionales pero en diferente escala.

En primera instancia se intentó utilizar la norma original del CS-

ACELP para obtener una señal de voz decodificada introduciendo el libro de códigos vectorial calculado reemplazando en C la matriz de excitación. Los resultados, como se verá, no fueron los esperados debido a que la forma en la que se lleva a cabo la correlación en la norma original está hecha para que tome menos tiempo que una correlación común y corriente. En otras palabras, haciendo uso de la energía de la señal y de los lugares donde se concentra la mayor cantidad de ésta en los vectores de las matrices, se lleva a cabo una correlación más eficiente que toma en cuenta únicamente aquellas posiciones de los vectores que la norma ubica como las de mayor importancia. Estas son aquellas posiciones que los pulsos del libro de códigos fijo del G729 pueden tomar por definición como se ha mostrado en la figura 7.

Debido a que no se pudo trabajar con la norma original del CS-ACELP, se optó por utilizar [9] donde el autor ha desarrollado en Matlab el decodificador del CS-ACELP y obtuvo resultados similares a los de la norma llevando a cabo la correlacion en el filtro de síntesis de forma completa. Fue por medio de la readaptación de este software que se pudieron obtener los resultados que se muestran en el ***CAPÍTULO 7.5***.

7.1. Obtención del Libro de Códigos

Para la obtención de nuestro codificador vectorial se siguió el siguiente diagrama de flujo.

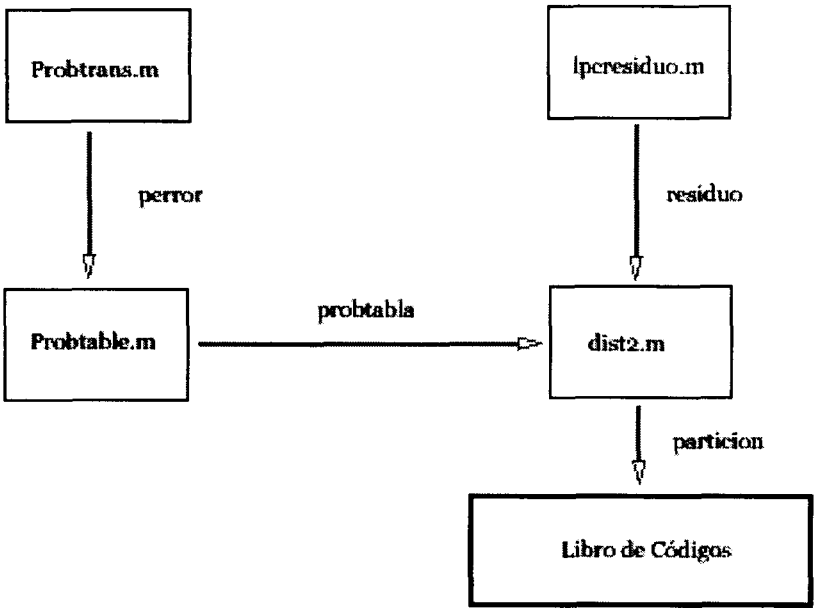


Figura 13: Diagrama de Bloques - Obtención Libro de Códigos

Los bloques representan los programas utilizados en Matlab, estos se pueden encontrar en el **APENDICE A**. El archivo *probtrans.m* utiliza las ecuaciones 34, 35 y 36 para calcular la probabilidad de error por bit de la fibra. Una vez obtenida la probabilidad de error por bit *probtable.m* genera la tabla de probabilidades de transición de dimensión 512x512 que representa las probabilidades de error para vectores representados por índices de nueve bits.

Esta tabla de probabilidades de transición y los residuos de la secuencia de entrenamiento producidos por *lpcresiduo.m* serán utilizados por *dist2.m* para obtener el libro de códigos óptimo, esto haciendo uso de las medidas de distorsión mencionadas en las ecuaciones 47, 48 y 49. Este libro de códigos será representado por índices de 9 bits usando el comando *dec2bin* de matlab.

7.2. Modificacion de la Norma

La norma del G729 de la ITU se conforma por diversos archivos programados en lenguaje C que desempeñan funciones importantes bien definidas. Para su compilación se utilizan los archivos coder.mak y decoder.mak. Una vez compilados se usan los archivos *coder.c* y *decoder.c* para simular el codificador y el decodificador. El codificador toma una señal de voz modulada con PCM a 16 bits y genera un archivo que contiene los parámetros de voz necesarios para la decodificación de la voz. Esto es **coder inputfile bitstreamfile**. El archivo *coder.c* queda intacto para nuestros fines, es durante la decodificación que el libro de códigos vectorial será intróducido y utlizado como excitación.

El archivo *decoder.c* toma el archivo **bitstreamfile** y genera un archivo de voz de 16 bits PCM. El archivo *decoder.c* fue modificado a modo que reemplace la excitación del libro de códigos algebraico por un vector de excitación generado previamente y que contenga la excitación de la señal de voz a decodificar, así, **decoder bistreamfile outputfile** se descarta y usamos **decoder bitstream outputfile f-fixcode** en su lugar. Donde **f-fixcode** representa un archivo de texto conteniendo los residuos de la señal de voz a decodificar y acomodados en un vector de dimensiones 1xsize(**f-fixcode**).

En la sección *Main Routine* del archivo *decoder.c* se puede observar que se necesitan tres archivos para realizar la decodificación. La sección *Loop for each "L-FRAME"speech data* nos muestra la generación de la matriz ***c*** que contendrá los vectores de excitación obtenidos del libro de códigos vectorial para la señal de voz a decodificar, esto leyendo el archivo que se le ha indicado al *decoder.c*. El archivo *acelp-co.c* busca el libro de códigos algebraico, sin embargo, este no fue modificado dado que en el *decoder.c* ya indicamos que matriz contiene los vectores de excitación. El archivo *decoder.c* sí calcula los signos y las posiciones de los pulsos para la excitación pero los utiliza en el filtro de síntesis.

La rutina de decodifiación se lleva a cabo en *dec-ld8k.c* en la que se llama a la matriz ***c*** que contiene los vectores de excitación, esto en la sección *Static Memory Allocation*. Este archivo utiliza la matriz ***cod*** para guardar los vectores de excitación para la señal de voz a codificar. En la rutina principal de decodificación se llama a la variable *frame*, ya especificada en *decoder.c* para saber que en que trama nos encontramos cuando se suman las ganancias y la contribución del libro de códigos adaptable a los vectores de excitación.

Del resultado de esta suma se obtiene la señal que entra al filtro de síntesis.

La modificación más importante hecha a la norma se lleva a cabo en el archivo *de-acelp.c*. En el archivo original se decodifican las posiciones de los cuatro pulsos que conforman la excitación contenida por el libro algebraico del CS-ACELP. Así mismo, se decodifican los signos de los pulsos y se construye el libro de códigos. Toda esta sección de la norma original se elimina. En el archivo *de-acelp.c* modificado se llama a la variable *frame* y *subfr* para determinar en qué trama y subtrama nos encontramos. Se hace un ciclo anidado en el que se guardan los vectores de excitación del libro de códigos que contiene ***c*** en la matriz ***cod***. Dado que el decodificador utiliza ***cod*** directamente en el filtro de síntesis basta con reemplazar los vectores originales de excitación que contienen los cuatro pulsos por nuestros vectores de excitación. Si se observa el archivo *de-acelp.c* de la norma original encontramos que los pulsos del vector de excitación toman valores de 8191 para uno positivo y de -8192 para uno negativo, por lo que es muy importante que el archivo que contiene los vectores de excitación del libro de códigos vectorial utilizado en la norma modificada esten acoplados a estos valores máximo y mínimo antes de correr el *decode.c*. Este acoplamiento de la amplitud de la excitación se lleva a cabo desde Matlab usando el programa *acoplesvector.m* que se puede ver en el **APENDICE A**. Otro punto a tomar en cuenta es que los valores utilizados por el decodificador son números enteros por lo que el archivo que contiene los vectores de excitación del libro de códigos vectorial que se usará como la nueva excitación deben ser sólo valores enteros.

Otra modificación importante hecha a la norma se da en el archivo *ld8k.c*. En este archivo se encuentran parámetros importantes para la decodificación. Entre estos encontramos el tamaño de la trama y la subtrama, el tamaño de la ventana del filtro de síntesis y el orden del filtro LP. Sin embargo, el más importante para el fin de la modificación de la norma original, es el tamaño del archivo que contiene los vectores de excitación que se van a usar. En la sección *Codec constant parameters* se agregó una línea que define a *MAX-SAMPLES*, que es la constante que indica el tamaño del archivo de texto que utiliza *decode.c* como entrada para generar ***cod***. Esta constante se debe modificar dependiendo del tamaño del archivo de texto que contiene los vectores de excitación de la señal de voz a decodificar. Estos se encuentran acomodados uno tras otro y el vector es de tamaño MAX-SAMPLESx1.

Por último se hace una modificación que de no realizarse no permitiría una justa comparación con otro trabajo de investigación del mismo tipo a futuro. El chequeo del bit de paridad deberá desactivarse o forzarce de

tal forma que si el decodificador detecta que alguna trama esta dañada no trate de reconstruirla con información anterior. Esto se modificó dentro de la norma forzando a que el vector que guarda los valores de error o de acierto del bit de paridad siempre tome el valor correcto.

7.3. Archivo de Excitación

El archivo de excitación, como llamaremos a aquel que contiene los vectores de excitación a usar por *decoder.c*, se genera como se muestra en el diagrama siguiente.

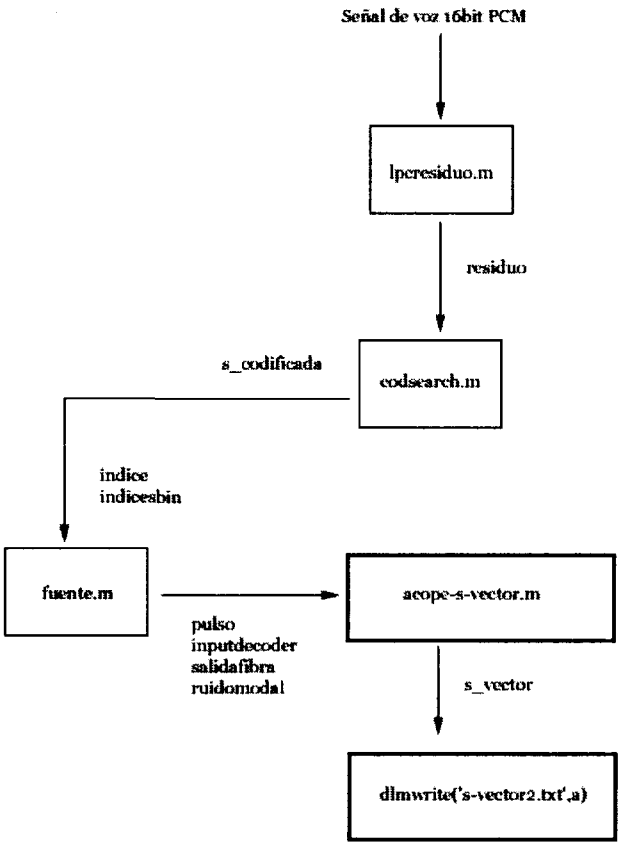


Figura 14: Diagrama de Bloques - Archivo de Excitación con ruido

Se obtienen los residuos de la señal a codificar usando el mismo programa *lpcresiduo.m* que se usó con anterioridad. Los residuos obtenidos se usan para generar *s-codificada* que es la matriz que contiene los vectores de excitación sin acotar de 41 a 120 , esto usando el programa *codsearch.m.* Una vez generada esta señal *s-codificada* y usando el programa *codsearch.m*, obtenemos las matrices ***indice*** e ***indicesbin*** que contienen los vectores de excitación acotados como se van a usar en el filtro de síntesis y los índices en código binario respectivamente. La matriz ***indicesbin*** se usa en el archivo *fuente.m* para representar la señal con pulsos de LED y agregarles ruido modal para después convertir esta señal en índices y posteriormente en un nueva matriz de excitación. Se utiliza la función de matlab *dlmwrite('speechfile',a)* para grabar en un archivo de texto nuestro vector de excitación. Es muy importante que antes de grabar el archivo de voz se acople la señal usando *acople-s-vector.m* de tal forma que los valores de los vectores no excedan 8191, no sean menores a -8192 y que además, sean valores enteros.

7.4. Complejidad Computacional

El proceso de codificación de un solo vector residual de excitación implica un tiempo de procesamiento extra al codificador. Utilizando el comando *cputime* de matlab pudimos obtener el tiempo de procesamiento en segundos que toma al codificador CS-ACELP modificado codificar un solo vector de excitación. Esto es, el tiempo que le toma codificar el vector del libro de códigos vectorial de una sola trama que nos arroja un valor de

Tabla 4 - *Tiempo de Procesamiento Extra*

Procesamiento Extra	17.925 ms

El retardo total de este codificador CS-ACELP modificado es de poco menos que ***48 ms*** procesado con una máquina PENTIUM 4 a 2 GHz y 256 MB de RAM. Con las nuevas tecnologías en procesadores el retardo debe ser mucho menor. Se pudieran utilizar máquinas con procesadores en paralelo para llevar a cabo la codificación con velocidades mayores y una más alta capacidad de memoria RAM.

Se entiende que la complejidad de procesamiento del CS-ACELP modificado por el libro de códigos vectorial es diferente a la del G729 de la

norma. Se midió el tiempo de procesamiento que toma al CS-ACELP modificado llevar a cabo la codificación y decodificación de los vectores residuales así como lo que le toma al decoder de Matlab decodificar los parámetros del codificador fijo algebraico del CS-ACELP original.

Tomando en cuenta una procesador Pentium 4 de 2 Ghz de velocidad de procesamiento con una unidad aritmética lógica Dhrystone con una capacidad de 3150 MIPS y de acuerdo al tiempo de procesamiento de cpu para llevar a cabo las operaciones de codificación y decodificación pertinentes observamos los siguientes resultados.

Tabla 5 - *MIPS*

	Procesamiento (seg)	**MIPS**
Codificador CS-ACELP modificado	17.925 ms	56.46375
Decodificador Libro Vectorial	10 ms	31.5
Decodificador Libro Algebraico	25 ms	78.75

Los valores calculados son para una simulación llevada a cabo en Matlab. La diferencia en la cantidad de millones de instrucciones por segundo entre un codificador y otro equivale a tan solo el **0.2925** por ciento de la capacidad de procesamiento de la ALU (Unidad aritmética lógica) mencionada. Aunque la parte de codificación del CS-ACELP modificado con el libro de códigos vectorial aumenta la complejidad de procesamiento del sistema, esta se compensa en la etapa de decodificación ya que la búsqueda del vector a la que apuntan los índices de excitación es muy sencilla lo que en el CS-ACELP requiere de lazos más complicados y de un mayor número de operaciones.

Observando la diferencia en MIPS del estándar del CS-ACELP con el CS-ACELP modificado y evaluado en Matlab observamos que este último agrega una complejidad extra debido al trato que le da a la simulación el software mismo. Es por eso que se calcularon la cantidad de operaciones necesarias para la síntesis del vector de excitación obtenido del CS-ACELP modificado.

Se puede llevar a cabo una comparación del codificador híbrido modificado por el libro de códigos vectorial con un codificador de forma de onda con una calidad toll en cuestión de complejidad de procesamiento computacional.

Observando la tabla 6 podemos comparar la cantidad de instrucciones por segundo llevadas a cabo para codificar una señal de voz usando ADPCM a 16kbps. Este codificador no cuenta con calidad Toll y aun así lleva a cabo la mitad de las operaciones realizadas por el codificador CS-ACELP original sin siquiera contar con una buena calidad de codificación.

Tabla 6 - *MIPS- ADPCM 16kbps*

Etapa	***MIPS***
Codificación PCM 64kbps	1.8
Compresión a ADPCM 16kbps	7.8
Total	9.6

En la tabla 7 observamos la cantidad de instrucciones por segundo de múltiples codificadores:

Tabla 7 - *MIPS- Codificadores*

Codificador	***MIPS***
LDCELP a 16kbps	50
LPC-10	15
ADPCM 32kbps	13.3
Codificador CS-ACELP Estándar	55
Decodificador CS-ACELP Estándar	30.8

Los datos obtenidos se pueden corroborar en la referencia [16]. Ahí podemos observar que para una PC común tomará más tiempo llevar a cabo las operaciones necesarias para la codificación de una señal de voz. Existen procesadores DSP como el TMS320C54xx de Texas Instruments que eficientiza la codificación y disiminuye la cantidad de instrucciones por segundo necesarias para codificar la señal.

Podemos llevar a cabo el cálculo de las instrucciones por segundo necesarias de forma analítica contando las operaciones de forma manual. El resultado de llevar a cabo la convolución entre el vector de excitación usado por el CS-ACELP modificado por el libro de códigos vectorial y el filtro de síntesis da como resultado

Tabla 8 - *Operaciones Calculadas*

Cantidad de Multiplicaciones	510
Cantidad de Sumatorias	540
Número Total de operaciones por trama	1050

Cabe mencionar que durante la convolución con el filtro de síntesis el CS-ACELP original realiza menos operaciones lo que lo hace aún menos complejo que el CS-ACELP modificado por el libro de códigos vectorial.

Las operaciones llevadas a cabo por codificadores de forma de onda son mucho menores a aquellas realizadas por los codificadores híbridos. La comparación real que puede hacerse en cuestión de procesamiento de máquina entre uno y otro se da a través de la compensación que otorga el contar con protección contra errores del canal y disminución de tasa de bits contra un aumento en el número de operaciones realizadas a cabo por trama.

7.5. Canal de Fibra

De acuerdo a [8] la señal modificada por un canal de fibra con ruido modal puede representarse como:

$$p(t) = \hat{p}(t) + n(t) \tag{50}$$

donde ***p(t)*** testada es la potencia de entrada con distribución Poisson y ***n(t)*** es cualquier otro tipo de ruido que se asume como un proceso aleatorio de distribución gaussiana con media cero. Esta aseveración es de suma importancia por la forma en la que se simula en esta tesis el ruido modal agregándolo a los pulsos de LED como una reproducción de voltajes aleatorios con media cero y varianza específica.

El ruido modal de acuerdo a [4] aparece como una distribución estocástica de intensidad de campo en el radio de la fibra y la influencia de este sobre la señal se calcula a través de su varianza que en este caso será de 0.09 como valor típico obtenido de la misma referencia.

El ruido modal aparece en el detector de voltaje montado sobre los bits que arriban al destino y que se introdujeron al canal en forma de pulsos de LED. El pulso producido por el LED para un valor de 1 y el pulso modificado por el ruido modal se puede ver en la figura siguiente.

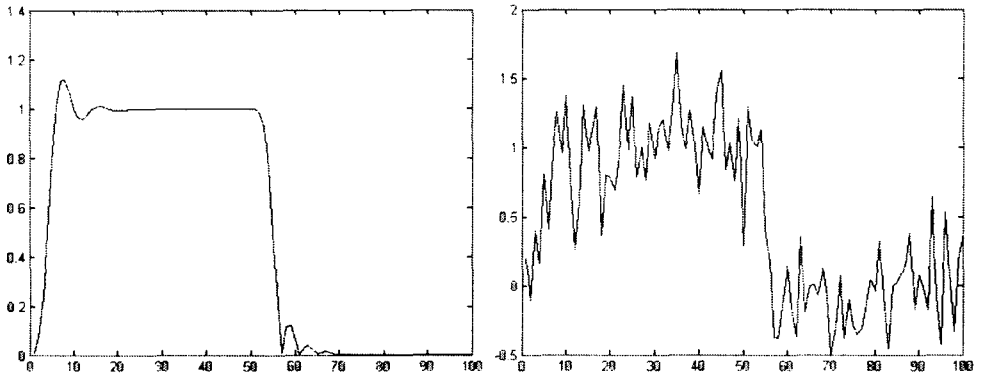


Figura 15: Pulso de LED- Con y sin ruido

El programa *fuente.m* hace una comparación para determinar los unos y ceros de la nueva señal. Una vez obtenido este nuevo vector de excitación podemos usar *dlmwrite('s-vector.txt',a)* para generar un archivo de texto que se utilizará en la norma modificada.

Dado que los parámetros producidos por el archivo *decode.c* se encuentran en código bruto se usa un programa en Matlab para leer archivos tipo raw. Una vez hecho esto podemos usar *ruido.m* para agregar ruido modal a los demás parámetros de voz que se van a utilizar en la codificación para despúes convertirlos de nuevo a un archivo raw. La señal de salida del decodificador es una señal a 16bit PCM pero en el mismo formato que los demás parámetros, de tal forma que para comparar esta señal con la de voz original y obtener los resultados se debe seguir el mismo proceso de conversión.

Para nuestros vectores de excitación se usó un proceso idéntico para simular el efecto del ruido producido por el canal que el utilizado para los demás parámetros de voz. Sin embargo, debido a que el formato en el que estos se encuentran es diferente, se tuvieron que usar archivos distintos para su trato.

7.6. Uso de un Decodificador en Matlab

El decodificador descrito en [9] se asemeja bastante en desempeño al descrito por la norma del G729 y para cuestión de comparación es más que

suficiente. Sin embargo, varias modificaciones tuvieron que realizarse para que la simulación sirviera a nuestros fines.

El archivo *decod-ideal.m* simula la rutina de decodificación del CS-ACAELP. Al programa se le indicó que leyera el archivo con los vectores de excitación codificados del libro de códigos vectorial y que los colocara en una matriz. Este archivo no debió ser reescalado previamente como se hacía con la norma original del CS-ACELP. Los vectores se utilizan tal y como se obtuvieron originalmente. Esto es, no utilizamos el programa *acople-s-vector.m* que es el que acota la señal y acopla los valores de amplitud a los valores que requería la norma. El diagrama de bloques muestra la secuencia de l

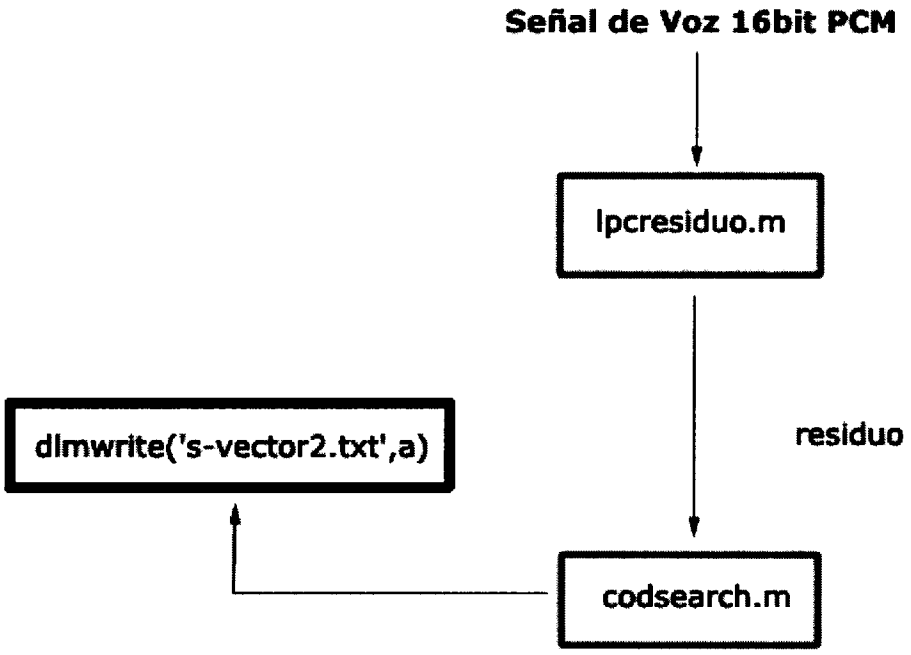


Figura 16: Diagrama de Bloques- Archivo de Excitación sin ruido

Se modificaron partes muy importantes como fueron la decodificación del vector del libro de códigos fijo que fue sustituido por la matriz mencionada donde se hizo caso omiso del cálculo de signos y posicionamiento de los pulsos de este.

Se encontró que al no tomar en cuenta las ganancias del libro de códigos fijo para la síntesis se mejoraba evidentemente el desempeño del codificador evitando picos de voltaje. Esto debido a que cuando se calculó el libro de códigos vectorial la energía se conservó en su totalidad. Esta modificación es la de mayor peso en cuanto a la medida de los resultados se refiere como se mostrará en el ***CAPITULO 8***, ya que de no llevarse a cabo el CS-ACELP modificado por el libro de códigos vectorial tendrá un desempeño menor al del CS-ACELP original.

Uno de los cambios más importantes a este simulador del CS-ACELP fue el programa que simula el cálculo del bit de paridad. En si, este no fue modificado, pero cuando se trata de calcular el retardo de pitch se indica al programa que la paridad calculada es la misma que la que indica el bit de paridad, por lo que la trama llegó correctamente y el programa no detectó error alguno. Este, como ya se mencionó es uno de los cambios más importantes si es que se desea contar con una comparación equitativa entre ambos sistemas.

Para la recuperación de la señal de audio simulando el CS-ACELP original se utilizó el mismo decodificador en Matlab haciendo uso de la modificación del bit de paridad únicamente, lo que nos da la ventaja de una igualdad comparativa. El libro de códigos fijo del CS-ACELP usa vectores de excitación que, como se explicó con anterioridad, son conformados por cuatro pulsos positivos o negativos, por lo que no toda la energía de la excitación de la señal original se guarda. Para la excitación del codidicador vectorial se utilizan vectores que cuentan con mayor energía que aquella contenida en los vectores del CS-ACELP original debido a que este libro de códigos vectorial fue generado por promedios de residuales de excitación de la señal original de voz y no por aproximaciones por medio de pulsos y ganancias, lo que nos debería otorgar una cierta ventaja.

8. Resultados de la Simulación

Para la obtención de los resultados se utilizó el archivo *reconvoz.m*, se acopló la amplitud de la señal, se guardó en un archivo de tipo *archivo.mat*. Este procedimiento para correr el programa *PS.m* con el que podremos obtener varios parámetros de importancia.

A continuación se presentan las señales relevantes del estudio.

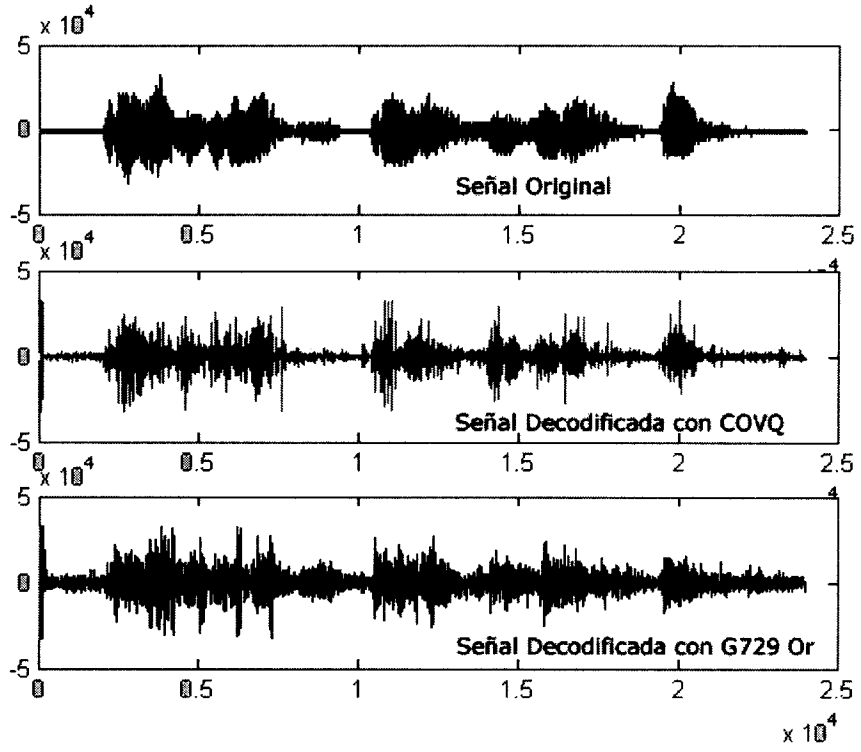


Figura 17: Comparación de señales de Voz

La primer señal es la señal original de audio, la segunda es la señal de voz decodificada haciendo uso del CS-ACELP modificado con el libro de códigos vectorial después de que todos los parámetros pasaron por el canal de fibra y la tercer señal es la señal decodificada usando el CS-ACELP original después de que los parámetros fueron afectados por el ambiente ruidoso del canal de fibra óptica.

Para obtener el SSNR (Segmental Signal to Noise Ratio) primero escalamos la señal de audio original de tal forma que alcance valores máximos de 65536 y mínimos de -65536, esto multiplicando la señal por un factor de 2^{16}. La señal obtenida se eleva al cuadrado para eliminar los negativos de lo que obtenemos la potencia de la señal de voz original. Se hace lo mismo para la señal decodificada por el CS-ACELP originaly por el CS-ACELP modificado con el libro de códigos vectorial. Para eliminar los picos en infinito negativo del valor SSNR se agregó a estas tres señales un valor promedio muy pequeño del orden 10^{-9} de tal manera que cuando se calcule el logaritmo correspondiente a la operación para obtener el SSNR no nos encontremos con valores infinitos generados por logaritmos de cero.

Una vez obtenidos estos valores se puede obtener el SSNR con la siguiente ecuación

$$\mathbf{SSNR} = 10\log(\frac{Ps}{Pr}) \tag{51}$$

Donde P_r se calcula restando la señal resconstruida por cada uno de los decodificadores a la señal original de voz y elevando este vector al cuadrado obteniendo así la potencia del ruido. Para obtener valores confiables de SSNR se deben eliminar picos máximos y mínimos de este cálculo. De no hacerlo los valores de SSNR serán muy bajos y la comparación entre los sistemas no será fiable. Este fue uno de los problemas principales con que se dieron en el cálculo de resultados, obteniendo valores de SSNR menores a 40 decibeles lo que habla de señales irreconocibles. El SSNR se cálcula de la misma forma que el SNR (Signal to Noise Ratio) con la diferencia de el uso de unas cuantas tramas de la señal de voz en lugar de utilizar la señal completa de para realizar la operación.

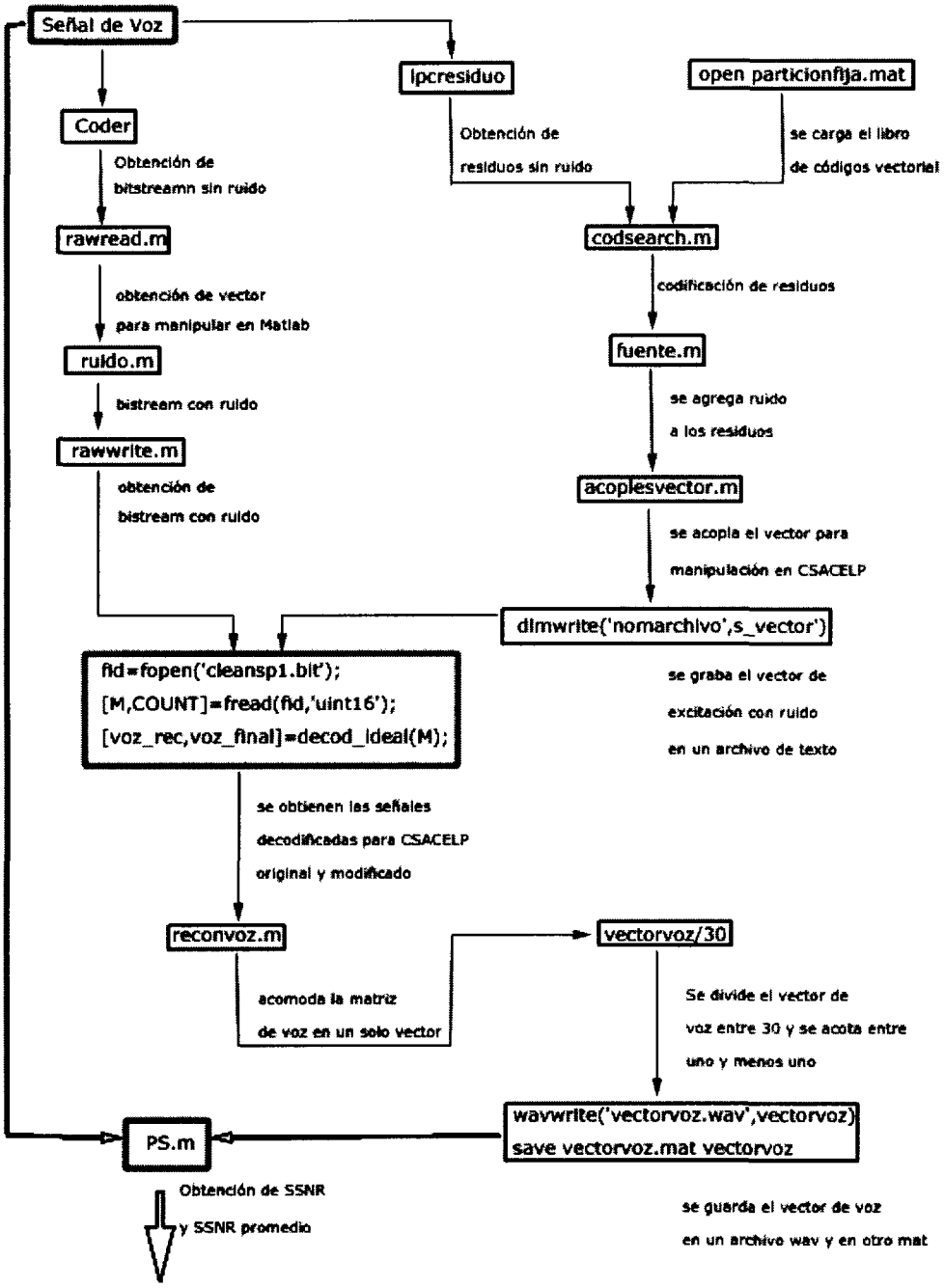


Figura 18: Simulación de Ruido y Obtención de SSNR

La figura 18 muestra el diagrama de bloques de la simulación para la obtención del SSNR y del cálculo de la media de esta medida de comparación tanto para el CS-ACELP original como el modificado para un canal de fibra en presencia de ruido modal. Otorga al lector una guía para la reproducción del experimento llevado a cabo para la obtención del SSNR y de la simulación del canal ruidoso.

La comparación del SSNR para la señal decodificada por el CS-ACELP original y el modificado con el libro de códigos vectorial se puede observar en la siguiente figura.

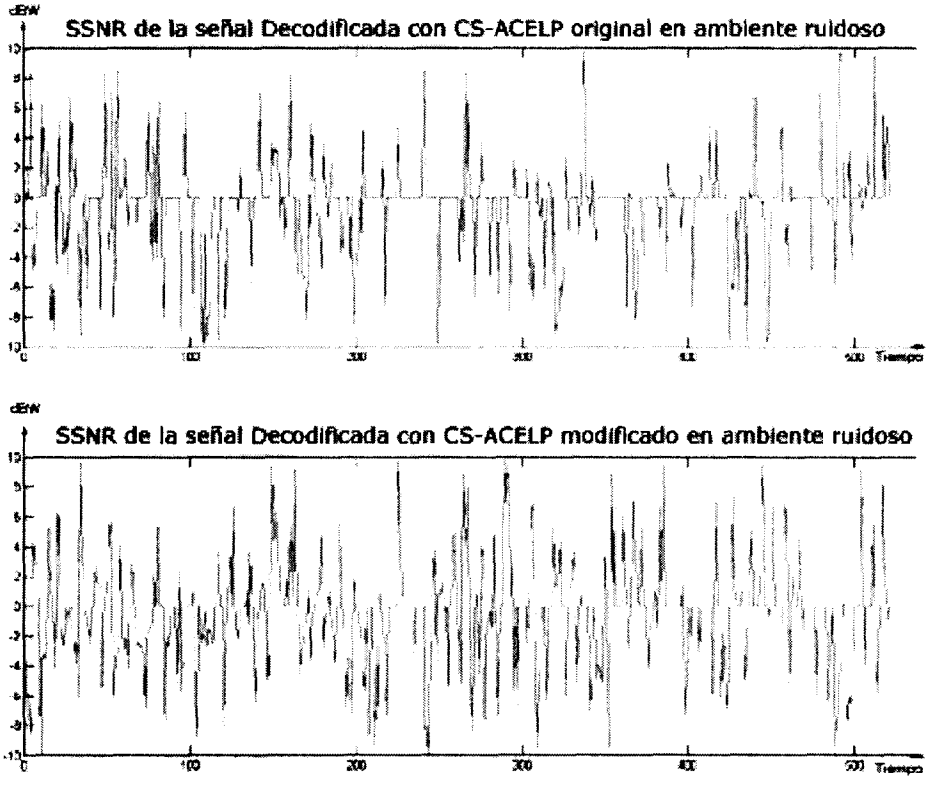


Figura 19: Medida-SSNR

Dado que esta gráfica no muestra claramente si existe una mejora en el desempeño entre uno de los sistemas se tomó una media sobre la medida SSNR que muestra una diferencia clara entre uno y otro proceso.

Tabla 9 - SSNR

dBW/CS-CACELP ->	*Original*	*Modificado*	*Con Ganancia*
SSNR Prom Total	0.0358	**0.2083**	0.1725
SSNR Prom 4 tramas	-0.0828	**0.0807**	0.1635
SSNR calculado sin silencios	0.1511	**0.2189**	0.1292

La diferencia de prácticamente 0.3029 decibeles de mejora para la señal recuperada utilizando el CS-ACELP modificado con el libro de códigos vectorial sobre la recuperada con el CS-ACELP original, apoya la mejora auditiva en cuestión de inteligibilidad de una señal con respecto a la otra. Cabe mencionar que en cuestión auditiva el oyente encuentra a la señal modificada por el libro de códigos vectorial más inteligible que la del CS-ACELP original, sin embargo, hace incapié en que a pesar de la característica robótica de la segunda se pueden detectar las palabras con menor dificultad que para el primer caso. El lector debe observar que al tomar en cuenta la ganancia del libro de códigos vectorial de la norma original durante el proceso de síntesis, el desempeño del CS-ACELP modificado por el libro de códigos vectorial es evidentemente menor. Esto debido a que el libro de códigos vectorial no discrimina las posiciones de máxima energía de los vectores de energía separándolas de sus ganancias que son cuantizadas separadamente. Al llevar a cabo la síntesis tomando en cuenta la ganancia del libro de códigos vectorial en el CS-ACELP modificado, se está añadiendo mayor en energía a los vectores de excitación, lo que provoca una mayor distorsión a al señal decodificada. Es por ello que es de suma importancia excluir del proceso de síntesis las ganacias del codificador vectorial calculadas en el proceso de codificación debido a que el codificador vectorial calculado por medio de COVQ ya las toma en cuenta.

La tercer línea de la tabla 9 nos muestra el cálculo del SNR para los segmentos de las señales de voz sin tomar en cuenta las áreas de silencio. Los valores de SNR mejoran un poco calculando el SNR de esta forma y se puede observar una pequeña mejora del decodificador vectorial contra el decodificador del CS-ACELP original.

El SSNR es una medida objetiva del desempeño de los codificadores y como se muestra en la tabla 9, al ser calculado para un número de tramas pequeño este puede variar.

A continuación se repite el experimento para un canal de fibra ruidoso donde la presencia de ruido modal se modela, como se mencionó antes, através de la varianza. Sin embargo, esta vez, disminuiremos el valor de la varianza para encontrarnos con un canal más amigable y observar como responden los sistemas de codificación en cuestión bajo estas condiciones.

Utilizando una varianza de 0.01 para el ruido modal se obtuvieron los resultados siguientes.

Tabla 10 - SSNR

CS-CACELP ->	***Original***	***Modificado***	***BER*** para $\sigma = 0{,}1$
SSNR Promedio dBW	0.0443	**0.2517**	5.8335e-006

Los resultados desplegados nos muestran una mejora pequeña en desempeño del CS-ACELP modificado con el libro de códigos vectorial sobre el original en 0.2084 dB al tomar en cuenta el valor SSNR sobre un canal ruidoso debido a que el libro de códigos fue calculado sobre las probabilidades de transición para preveer errores de transmisión. Dado que la varianza del ruido es muy pequeña, el canal ruidoso distorsiona poco la señal de voz, por lo que el desempeño de ambos codificadores llega a ser muy similar.

Durante el cálculo del SSNR al elevar al cuadrado P_r si el efecto de ruido es muy grande para determinados puntos y siguiendo la fórmula,

$$SSNR = 10(log\frac{P_s}{P_r}) = 10(log(P_s) - log(P_r)) \tag{52}$$

se observa que el término del lado derecho de la segunda igualdad nos daría un valor de SSNR muy bajo. Caso que se dio durante el cálculo de valores de SSNR donde encontrabamos picos de hasta -100 dbW cuando la mayor parte de los puntos del vector de SSNR se encontraban alrededor de 1 dbW lo que ocasionaba una media muy baja cerca de -50 dbW. Por ello es importante acotar estos picos que de no ser ignorados no nos permitirían una comparación justa. De acuerdo a [2] el SSNR nos otorga una medida objetiva de la energía media de la señal de voz con respecto al ruido. El cálculo se lleva a cabo de forma segmental usando espacios de 20 ms pero eliminando los picos mencionados se puede evaluar el SSNR total.

La ventaja en inteligibilidad de la señal codificada con CS-ACELP modificado podría deberse, como se mencionó antes, a la evidente ganancia en cuestión de energía perdida del CS-ACELP original con respecto a nuestro sistema. Mientras que el libro de códigos vectorial de nuestro sistema se generó en base a promedios de los residuales de excitación de la señal original de voz, la excitación del codificador original fue generada por una aproximación de cuatro pulsos modificados por ciertas ganancias, lo cual hace que se pierda la energía de las 76 posiciones restantes del vector de excitación.

Como punto final se debe hacer incapié en un tema en el que no se había profundizado en el trabajo con anterioridad. La compresión de la señal

de voz para un codificador CS-ACELP como lo describe la norma original indica un valor de 8Kbps. Si analizamos la figura 20 obtenida directamente de la norma de la ITU-T para el CS-ACELP nos encontramos con

Parameter	*Codeword*	*Subframe 1*	*Subframe 2*	*Total per frame*
Line Spectrum Pairs	$L0$, $L1$, $L2$, $L3$			18
Adaptive-codebook delay	$P1$, $P2$	8	5	13
Pitch-delay parity	$P0$	1		1
Fixed-codebook index	$C1$, $C2$	13	13	26
Fixed-codebook sign	$S1$, $S2$	4	4	8
Codebook gains (stage 1)	$GA1$, $GA2$	3	3	6
Codebook gains (stage 2)	$GB1$, $GB2$	4	4	8
Total				80

Figura 20: Acomodo de Bits por trama

La figura describe la ubicación de los bits por trama para el CS-ACELP original. Los signos del libro de códigos fijos ocupan 8 bits por trama y los índices para el proceso de búsqueda del mismo libro ocupan 26 bits por trama. Un total de 34 bits por trama se utilizan para reconstruir la contribución de la excitación hecha por los parámetros del libro de códigos fijo, y un total de 40 al tomar en cuenta los seis bits que se utilizan para cuantizar las ganancias de los libros de códigos algebraico y adaptable.

Para el codificador CS-ACELP modificado por el libro de códigos vectorial podemos reemplazar los 34 bits por trama mencionados por los 9 bits que conforman nuestro vector de excitación. De esta forma se obtiene un ahorro de 22 bits por trama o por cada 10 ms lo que nos daría como resultado un CS-ACELP modificado con una tasa de transmisión de 5.8Kbps.

Aunque el resultado suena muy impresionante no hay que despegar los pies del piso. El codificador obtenido tiene ciertas ventajas cualitativas auditivamente hablando y ofrece una mayor compresión para un canal de fibra simulado bajo la suposición de que el único impedimento con que la señal digital transmitida por el emisor se topa son la distorsión generada por un LED y el ruido modal generado por la guía de fibra óptica. Para

la obtención de un codificador más robusto con la compresión mencionada obtenida se debe llevar a cabo una exploración matemática más profunda del canal de fibra óptica y de su ambiente ruidoso. Así mismo, es importante el poder modificar el decodificador de la norma original del CS-ACELP para utilizarlo como base de los resultados de los experimentos que establecerán el desempeño de sistemas generados bajo lineamientos similares a los propuestos en este trabajo de investigación.

Se han mencionado ya las propiedades robóticas adquiridas por el audio codificado con el codificador CS-ACELP original comparado con la mejora en inteligibilidad provocada por el reemplazo del libro de códigos fijo por uno vectorial. Sin embargo, al realizar una prueba de opinión promedio (MOS) entre 15 individuos estos daban más importancia a la claridad encontrada en la señal codificada por el CS-ACELP original que a la inteligibilidad obtenida gracias al CS-ACELP modificado con el libro de códigos vectorial. La tabla siguiente muestra los resultados obtenidos del MOS siguiendo la misma metodología mostrada en la tabla .

Tabla 11 - MOS

	Inteligibilidad	**Claridad**	**Preferencia**
CS-ACELP modificado	2.6	1.6	1.8
CS-ACELP original	1.4	3.2	2.8

La prueba sobre la inteligibilidad y claridad fueron evaluadas con valores que fueron de 1 a 5 con cinco como la calificación más alta y uno la más baja.

Un punto de soberbia importancia y sobre el que no se hizo se enfatizó de forma exhaustiva es en el retraso mínimo de procesamiento de codificación y decodificación que requiere el CS-ACELP. Los 15 mili segundos de retraso por trama que agrega el proceso de codificación y decodificación al enlace de comunicación, hace de este codificador uno óptimo para el enlace de voz en tiempo real. Debido a que la forma en la que realiza las convoluciones para sintetizar la señal de voz es muy eficiente, el CS-ACELP provoca un retraso imperceptible para los interlocutores. El sistema propuesto en esta tesis cuenta con un mayor tiempo de procesamiento de codificación y decodificación. En primera instancia por llevar a cabo la convolución en el filtro

de síntesis de forma regular y en segundo porque la búsqueda del libro de códigos vectorial no fue diseñada para minimizar el retardo de la búsqueda del vector codificado por tal o cual índice. Sin embargo, con el surgimiento de nuevos procesadores y máquinas más potentes se espera que este retardo sea cada vez menor hasta llegado el punto en que las ventajas otorgadas por un sistema de esta índole superen sus contrapartes adversas.

Es esta tesis un pequeño paso y una invitación hacia la exploración profundizada del tema de la codificación conjunta enfocada a canales de fibra óptica y espera ser de utilidad para quien la lea.

9. Trabajo Futuro

La transmisión de señales de voz a través de fibra óptica es una aplicación de uso actual que, a su vez, es objeto de investigación constante. Existen, como se ha mencionado, diversas características propias de un canal de fibra óptica. Para su modelación y correcta aplicación es difícil tomar en cuenta todas y cada una de estas características. Es por eso que, muchas veces, se modelan sólo aquellas que son de importancia para el usuario.

Algunas de las líneas de trabajo futuro podrían estar enfocadas hacía la búsqueda de libros de códigos diseñados de acuerdo a las probabilidades de transición de un canal de fibra con características diferentes a las expuestas en este trabajo. Pudieramos utilizar un canal de fibra con probabilidades de transición que dependieran, por ejemplo, de las pérdidas debido al laser. Por otro lado, se pudiera realizar investigación utilizando un modelo de canal basado en múltiples características de canal, esto, haciendo incapié en más de un impedimento de transmisión, como podría ser un canal de fibra con reflexiones y ruido modal.

Para desarrollar un trabajo como el mencionado se deben contar con los modelos de canal propios y las probabilidades de transición adecuadas para llevar los arreglos matemáticos óptimos para el caso.

En [5] se especifica como buscar un libro de códigos de tipo matricial. Según el autor, al hacer uso de este método, se puede obtener una disminución en la medida de distorsión con respecto al uso del algoritmo iterativo para encontrar el libro de códigos vectorial que se utilizó en este trabajo. Es decir, al generar un libro de códigos con este método algebraico matricial la señal decodificada, en teoría, debería estar más apegada a la señal de voz original.

Esta combinación de estudios puede generarse para distintos medios de comunicación en los que la codificación de voz y su recuperación sea el punto imperante.

Si el investigador no busca una mayor compresión de la señal de voz pudiera utilizar un método iterativo de búsqueda para generar un libro de códigos vectorial para cuantizar cierto parámetro de codificación con un mayor número de bits, buscando así perder menos información de la señal original. El completar este proceso conlleva un mayor costo en cuanto a tiempo máquina se refiere.

Una línea evidente de trabajo sería realizar una documentación exhaustiva sobre la forma en la que trabaja la norma en lenguaje C. Debido a que la obtención de resultados se complica dado que algunos bloques de

programación requieren una inspección más a fondo. Sería un buen esfuerzo el tratar de poner en orden la norma de forma que se pudiera usar para trabajos a futuro sobre esta misma línea. De hacerse lo anterior un investigador podría utilizar la norma de forma transparente y obtener una investigación más profunda y de mayor enfoque a las telecomunicaciones sin tener que encasquillarse en la programación de una norma que está bien establecida.

10. Conclusiones

El trabajo de investigación siguió dos líneas principales definidas en su totalidad de acuerdo a la plataforma de software sobre la cual se implementaron los decodificadores.

El decodificador original de la norma del CS-ACELP no pudo ser modificado al grado que los dos sistemas de comunicación expuestos a lo largo de la tesis pudieran ser comparados de forma equitativa. Razón por la que se optó otro recorrido hacia la búsqueda de resultados comparativos.

De acuerdo a las tablas 9 y 10 de la tesis, una por cada experimento (o por archivo de voz), la implementación de un codificador vectorial para sustituir la contribución de excitación dada por el libro de códigos fijo del CS-ACELP en su forma original otorga una mejora de hasta 0.3 dB sobre la señal decodificada.

La medida de SSNR podría ser suficiente, matemáticamente, para definir que un sistema es mejor que otro. Sin embargo, se utilizaron 15 sujetos para llevar a cabo una prueba auditiva en la que se les pidió que describieran la señal que escuchaban en términos de inteligibilidad, trasparencia de audio y ausencia de ruido. No se utilizó ninguna medida cualitativa para medir el desempeño, tan solo se pidió a los escuchas que definieran las características de cada señal decodificada por los sistemas.

Para la señal de audio decodificada por el CS-ACELP original se obtuvo una descripción general en la que se perdía total inteligibilidad. El escucha no podía identificar al parlante debido a la característica robótica de su timbre de voz debido a la distorsión provocada por el ruido modal.

La señal de audio decodificada con CS-ACELP modificado con el libro de códigos vectorial contaba de forma general con una inteligibilidad, prácticamente, perfecta pero con una distorsión sobre la articulación de las palabras debido al mismo ruido modal. El escucha podía identificar perfectamente al parlante pero no contaba con la misma capacidad para identificar las palabras que el parlante articulaba.

Para que este sistema tenga validez se debe de buscar un sistema computacional que haga a un lado los retardos ocasionados por este nuevo sistema de codificación. Gracias a la capacidad convolutiva eficiente del algoritmo del CS-ACELP el retardo ocasionado por el codificador original es imperceptible para los interlocutores al momento de hablar de un enlace de comunicación en tiempo real. Para implementar el sistema en cuestión se deben de contar con máquinas muy poderosas enfocadas únicamente a la

tarea asignada.

Otra observación que debe hacerse es sobre la compresión que se hizo sobre la señal al reducir el libro de códigos de 34 bits a 9 bits generando así un codificador de 5.8 kbps en lugar de uno de 8 kbps. Una vez el resultado parece impresionante pero deben tomarse en cuenta cada una de las ventajas y desventajas de los sistemas para hacer una comparación juiciosa equitativa. Es decir, las ventajas de uno son las desventajas de otro y un sistema con ciertas características aventajará a otro en medida de lo que uno requiere de él. Mientras que la inteligibilidad otorgada por el CS-ACELP modificado por el libro de códigos vectorial pudiera ser útil para una persona que desea escuchar a un ser querido, esta pudiera ser no tan necesario para un enlace de voz entre dos personas que necesitan entablar una conversación de negocios.

Para poder implementar un sistema fiable como el mencionado se debe tomar en cuenta que el canal de fibra tiene muchas más características que por cuestiones de complejidad se dejaron pasar por alto en esta como en muchas investigaciones, sin embargo, y como se mencionó antes, este es sólo un pequeño paso para futuras investigaciones y espera ser de ayuda para quien busque el desarrollo de las telecomunicaciones.

11. Bibliografía:

1. Kondoz, A. M. (Ahmet M.); Digital speech : coding for low bit rate communication systems; Chichester ; New York : J. Wiley, c1994

2. O'Shaughnessy, Douglas; Speech communications : human and machine; New York : Institute of Electrical and Electronics Engineers, c2000

3. Transferencia: Posgrado, Investigación y Extensión en el Campus Monterrey; Año 15, Número 58, Abril de 2002

4. Natasa A. Trivunac-Vukovic, Mihajlo C. Stefanovic; Performances Degradation of Digital Fiber Optic Systems Due to Modal Noise; Faculty of Electronic Engineering, Beogradska 14, 18000, Nis, Yugoslavia.

5. Pérez Córdoba,Jose Luis; Estudio Sobre la Codificación Conjunta Canal-Fuente en Canales Continuos; Universidad de Granada, Departamento de Electrónica y Tecnología de Computadores; Granada, Noviembre del 2000

6. Linde-Buzo-Gray; An algorithm for Vector Quantizer Design; IEEE Transactions on Communications, Vol. COM-28, No. 1, January 1980

7. ITU-T; Coding of Speech at 8bit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP); ITU-T Recommendation G.729; 03/96

8. Donald G. Duff; Computer-Aided-Design of Digital Systems; IEEE Journal on Communications, Vol SAC-2, No 1, Enero 1984

9. Arteaga Sainz, Iván; Implementación y Desarrollo del Algoritmo para Decodificación de Voz del Codificador CS-ACELP de 8 Kbps; ITESM, Campus Monterrey; Monterrey, NL, MEXICO; Mayo 2003

10. Beritelli, Francesco; A Modified CS-ACELP Algorithm for Variable-Rate Speech Coding Robust in Noisy Environments; IEEE Signal Processing Letters, Vol 6, No 2, Febrero 1999

11. Hong Kook Kim; Adaptive Enconding of Fixed Codebook in Celp Coders; Proceedings 1998, ICASSP 1998, IEEE International Conference, Vol 1, pag. 149-152, Mayo 1998

12. Ruggeri-Beritelli-Casale; Hybrid Multi-Mode/Multi-Rate CS-ACELP Speech Coding for Adaptive Voice Over IP; Proceedings 2001, ICASSP 2001, IEEE International Conference Vol 2, Pag 733-736, Mayo 2001

13. Xiao-Vucetic; Combined Variable Low-Bit-Rate Speech-Channel Coding Over Noisy Channels; Universal Personal Communications, ICUPC 1998, IEEE 1998 International Conference Vol 2, Pag 1005-1009, Octubre 1998 vol.2

14. Farvardin, Nariman; A Study of Vector Quantization for Noisy Channels; IEEE Transactions on Information Theory, Vol. 36, No. 4, July 1990

15. Garde Martínez, Ainara; Codificación Conjunta de Fuente y Canal Aplicado a un CELP sobre un Canal de Fibra Óptica Limitado por Dispersión; ITESM, Campus Monterrey;Monterrey, NL, MEXICO; Mayo 2004

16. *http://www.spiritcorp.com/g729.html*

17. Cruz Rodríguez, José Ramón; CODIFICACIÓN CONJUNTA DE FUENTE Y CANAL APLICADA A CODECS HÍBRIDOS DE VOZ; UNIDAD ZACATENCO DEPARTAMENTO DE INGENIERIA ELECTRICA SECCION COMUNICACIONES; Mexico, D.F.; Agosto 2000

18. Goldsmith-Effros; Iterative Joint Design of Source Codes and Multiresolution Channel Codes; California Institure of Thechnology IEEE, 1997

19. Xiao-Vucetic; Combined Variable Low-Bit-Rate Speech-Channel Coding Over Noisy Channels; IEEE, 1998

20. *http://www.ftthcouncil.org*

21. *http://www.convergedigest.com/DSL/ftth.sp*

GLOSARIO

ADPCM: Adaptive Differential Pulse Code Modulation. Algoritmo basado DPCM con la capacidad de adecuarse a la magnitud de la muestra de la señal a codificar.

ALU: Siglas referentes a la unidad aritmética lógica de un ordenador.

AWGN: Additive White Gaussian Noise. Referente a ruido gausiano blanco aditivo caracterizado por tener una densidad de potencia espectral plana y densidad de probabilidad gausiana.

BER: Bit Error Rate. Referente a la probabilidad de error por cada bit.

CELP: Code Excited Linear Predictor. Codificador híbrido basado en una técnica de análisis por síntesis, cuantización vectorial y predicción lineal.

COVQ: Channel Optimized Vector Quantization. Cuantizador vectorial que busca optimar la asignación de índices de un cierto parámetro para disminuir cierta medida de distorsión que está relacionada con el comportamiento de algún canal de comunicación. El algoritmo genera cuantizadores vectoriales basados en modelos probabilísticos o en secuencias grandes de entrenamiento.

CS-ACELP: Conjugate-Structure Algebraic-Code-Excited linear-prediction. Codificador Híbrido basado en una técnica de análisis por síntesis, cuantización vectorial y predicción lineal de la familia del CELP generado para contar con una calidad similar a la de un ADPCM de 32 kbps pero con una mayor compresión.

DPCM: Differential Pulse Code Modulation. Codificador que transmite la diferencia en amplitud entre la muestra de la señal original y un estimado de la misma.

FTTH: Fiber to The Home. Siglas referentes a la tecnología recientemente implementada en Ciudades de Japón y Estados Unidos que provee de servicios de videoconferencia, enlaces de voz en tiempo real e Internet.

G729: Se refiere al estándar del CS-ACELP descrito por la ITU-T.

JSCC: Joint Source-Channel Coding. Siglas en ingles para codificación conjunta de canal-fuente para una codificación que no se lleva a cabo en

cascada si no de forma conjunta lo que, para casos diversos, logra muy buenos resultados.

LPC: Linear Prediction Coding. Tipo de vocoder que supone que el tracto vocal puede ser modelado, típicamente, por un filtro de décimo orden.

LSP: Line Spectral Pairs. Representación de los parámetros obtenidos del filtro LPC. Los LPC's se usan dado que los coeficientes LPC's cuentan con una distribución pobre por lo que resulta difícil interpolarlos para su cuantización.

MA: Moving Average. Tipo de predicción basada en la correlación de la señal lo que otorga al codificador cierta protección contra errores del canal.

MIPS: Millions of Instructions per Second. Medida de complejidad de un algoritmo determinado para su comparación con otros.

MOS: Mean Opinion Store. Evalúa el desempeño comparativo del codificador de una forma subjetiva.

PCM: Pulse Codee Modulación. Codificador en el que cada muestra toma una valor de amplitud entre un rango finito de niveles específicos, esto es, se cuantifica. A cada nivel le corresponde un número binario que será transmitido por el canal y será decodificado por el receptor.

SNR: Signal to Noise Ratio. Relación señal a ruido. Medida objetiva de distorsión de la señal comparativa al ruido.

SSNR: Segmental Signal to Noise Ratio. SNR segmental que permite realizar una comparación objetiva de la señal con respecto al ruido en pequeños segmentos o tramas de la señal para posteriormente promediar este valor que sería de mayor utilidad que el SNR.

VQ: Vector Quantizer. Siglas referentes a Cuantizador Vectorial.

YOHO: Base de datos de Sonido que cuenta con cientos de archivos de voz masculina y femenina de duración de tres o más segundos.

ANEXO A

Contenido:

Archivos Creados para la tesis

Uso de Paquetería
Cálculo de la Probabilidad de Error por Bit
Cálculo de la tabla de probabilidades de transición
Obtención de los vectores residuales
Obtención del Libro de Códigos Vectorial
Codificación de los vectores residuales
Generación de Pulsos de LED y Ruido Modal 1
Generación de Pulsos de LED y Ruido Modal 2
Acople del archivo de excitación
Reconstrucción de señal final DECODER MATLAB
Cálculo de SSNR

Archivos de ayuda

Decoder Matlab Modificado
rawread.m
rawwrite.m
Archivos modificados en C de la norma del CS-ACELP

Archivos Creados para la tesis

Uso de Paquetería

Para el correcto funcionamiento del software presentado en este ANEXO el lector puede guiarse por la siguiente descripción. Es importante guardar en archivos mat los vectores y matrices indicados para su uso en las etapas posteriores de la simulación.

1. Obtención de Archivo de voz en formato wav PCM 16bits
    - Guardar archivo mat con información de vector wav escalada para uso posterior
2. Cálculo de parámetros LPC y residuos: lpcresiduo.m
    - Guardar residuos en archivo mat
3. Cálculo de probabilidad de error: probtrans.m
4. Cálculo de tabla de probabilidades de transición: probtable.m
    - Guardar matriz de probabilidades de transición en archivo mat
5. Cálculo de Partición fija óptima: dist2.m
    - Guardar partición óptima en archivo mat
6. Cálculo de archivo de excitación: codsearch.m
7. Acople se la señal para uso en el decodificador: acoplesvector.m
8. El archivo de excitación pasa por el canal ruidoso de fibra: fuente.m
    - Se guarda un archivo de texto utilizando el comando dlmwrite de Matlab
9. Se obtiene el archivo raw con los parámetros de codificación de la señal de voz usando los archivos correspondientes de la norme G729
10. Se lee el archivo raw en matlab para modificación: rawread.m
11. Los parámetros de codificación pasan por el canal ruidoso de fibra: ruido.m

12. Se convierte el vector de parámetros modificados por el ruido nuevamente a formato raw: rawwrite.m

13. Se utiliza el decodificador señalado en [9]

- Cabe señalar que por cuestión de comparación se hicieron modificaciones en este decodificador. Se hacen dos experimentos de los que se obtienen dos señales. La primera usando el decoder con un cambio en la detección de tramas con error que se elimina. De ahí se obtiene un archivo y se graba con format mat para posterior uso. El segundo experimento usa un decodificador más modificado que a su vez utiliza el archivo de excitación. En ambos experimentos se utiliza el archivo raw modificado por el ruido del canal de fibra.

14. Reacomodo de vector de voz para comparación de experimentos: reconvoz.m

- Guardar vectores de voz en archivo mat

15. Obtención de SSNR o SNR según se requiera: PS.m

- Después de acoplar la señal de voz a un rango de amplitud similar a la de las señales de voz decodificadas se utiliza esta señal en conjunto con los archivos mat previamente guardados de las señales de voz decodificadas

probtrans.m

Cálculo de la probabilidad de error por bit a partir de las probabilidades de transición. Se toman valores obtenidos de la referencia [4].

```
%%%%%% obtencion de la probabilidad de error %%%%%%%

%Amplitud del voltaje
A=1;

Pw=.000063095e-3;
vv=0.0017761615917477777;

%carga del electron
q=1.6021892e-19;

%resistencia del splice
R=50;

%tiempo de un bit
T=0.0625e-3;

%Voltaje umbral
Vt=vv/2;

%Otros parametros
niuo=0.5;
niu=10^(-niuo/10);
Pav=-55;
alphaL=20;
Po=Pav/(niu^2);
v_v=q*R/T;
var=2*(Po^2)*(niu^2)*((1-niu)^2)*(10^(alphaL/10));

% Desviacion Estandar:  Este valor se modifica para obtener las tablas de transicion
de probabilidades
% adecuadas a una varianza dad

sigma=.3;
k=.8;
k1=.8;

%Maximo  de electronces considerando A=5 R=50 T=.0625 nt=0
nmax=3.9009e13;


%Calculo de P(1\0) y P(0\1)

P0_1=0;
P1_0=0
```

```
for n=1:150
   nt(n)=0.000177616159174777777*randn(1);
   m(n)=0.000177616159174777777*randn(1);
   v(n)=(((q*R)/T)*n)+nt(n);
   termino1(n)=(exp(-((v(n)-v_v)^2)/(2*(sigma^2))))/(sqrt(2*pi)*sigma);
   termino2(n)=(k1*((vv+m(n))^(2*n)))/(factorial(n));
   termino3(n)=exp(-k*((vv+m(n))^2));
   P01(n)=termino1(n).*termino2(n).*termino3(n);
   P0_1=P0_1+P01(n);
   n


   i=n+6.9287e+009;

   termino1n(n)=(exp(-((v(n)-v_v)^2)/(2*(sigma^2))))/(sqrt(2*pi)*sigma);
   termino2n(n)=(k1*((m(n))^(2*n)))/(factorial(n));
   termino3n(n)=exp(-k*((m(n))^2));
   P10(n)=termino1n(n).*termino2n(n).*termino3n(n);
   P1_0=P1_0+P10(n);


   if P01(n)<=10e-9
      n=200;
   end

end


P0_1
P1_0

% Se toma la probabilidad de obtener uno o cero como 0.5 respectivamente
suponiendo que
% la probabilidad de enviar UNO o un CERO es la misma

perror=P0_1*(.5)+P1_0*(.5);
```

probtable.m

Cálculo de la tabla de probabilidades de transición a partir de la probabilidad de error obtenida de ***probtrans.m.***

```
%Probabilidad de error calculada con probtrans.m
Pe=perror;

column(1)=0;
for i=2:512
   column(i)=i-1;
end

%conversion de un vector que va de 0 a 512 a binario
colbin=de2bi(column);

%inicializacion de la tabla de probabilidades de trasicion
probtabla=zeros(512);
j=1;
for j=1:512
   for i=j:512
      compare=colbin(j,:)==colbin(i,:);
      suma=0;
      for r=1:9
         if compare(1,r)==1
            suma=suma+1;
         end
      end
      %Tabla de probabilidades de transicion
      probtabla(i,j)=((1-Pe)^suma)*(Pe^(9-suma));
   end
end



tranprobtabla=probtabla';
prob=probtabla+tranprobtabla;

for o=1:512
   prob(o,o)=prob(o,o)./2;
end
probtabla2=prob;

% Tabla de probabilidades de Transicion

probtabla=probtabla2;
```

lpcresiduo.m

Obtención de los vectores residuales a partir de los parámetros LPC. En el programa se pueden obtener los vectores residuales cambiando el nombre del archivo en la segunda línea del programa por aquel que contiene el de interés para el usuario.

```
% Obtencion de los vectores residuales

[x,fs]= wavread('prueba2');  %Digitalizamos el archivo .wav



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%Preenfasis%%%%%%%%%
%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%Le aplicamos el preenfasis a la señal x
% para enfatizar las altas frecuencias
y=filter([1 0.97],1,x);

y=y';

disp('etapa1')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%SEGMENTACION%%%%%%%%
%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

%Ventaneo de la señal de entrada
%Declaramos variables

Segundos_por_ventana = 0.02;
NumMuestras=fix(fs*Segundos_por_ventana);
NumVentanas=fix(length(y)/NumMuestras);
Traslape=fix(NumMuestras/2);
Avance = NumMuestras-Traslape;

%Declaramos matrices inicializadas a ceros

zw= zeros(NumVentanas,NumMuestras);
z2= zeros(NumVentanas,NumMuestras);
z= zeros(NumVentanas,NumMuestras);
ham= zeros(NumVentanas,2*NumMuestras);
```

```
%segmentacion

for i =1:2*NumVentanas-1
	li=(i-1)*Avance+1;
	ls=(i-1)*Avance+160;
	z(i,:)=y(li:ls) ;
	zw(i,:)= z(i,:).* hamming(160)';

end

disp('etapa2')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%FILTRO
PASABAJAS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%

% Pasamos por un filtro pasa bajas la señal muestreada
[B,A]=butter(4,1000/4000);   %Obtenemos la funcion de transferencia

%Inicializamos vectores de memoria a cero

Zilength=max(length(A),length(B))-1;   %define la longitud del vextor Zi
Zi=zeros(1,Zilength);  %Inicializa el vector Zi a ceros
Zf=zeros(1,Zilength);  %Inicializa el vector Zf a ceros

for j=1:2*NumVentanas-1
Zi=Zf;
[yw(j,:),Zf]=filter(B,A,zw(j,:),Zi);
end

disp('etapa3')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%OBTENCION DE LOS
LPC%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%A la señal yw le obtenemos los parametros lpc a cada trama y le aplicamos el filtro
inverso


[lp,gain]=lpc(zw',10);  %Obtenemos los parametros lpc.
lp=real(lp);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%FILTRO
INVERSO%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Zinilength=max(11,1)-1;   %define la longitud del vextor Zi
Zini=zeros(1,Zinilength);  %Inicializa el vector Zi a ceros
Zfin=zeros(1,Zinilength);  %Inicializa el vector Zf a ceros

for k= 1:2*NumVentanas-1
Zini=Zfin;
% Vector con los residuos correspondientes
[residuo(k,:),Zfin]= filter(lp(k,:),1,yw(k,:),Zini);
end
```

dist2.m

Obtención del libro de códigos vectorial usando COVQ, tratando de minimizar la medida de distorsión indicada por la ecuación ***48*** y definida por las ecuaciones ***49*** y ***50.***

```
%%%%%%%%%%%%%%%%%%%%%%%% MATRIZ DE INICIALIZACION PARA
EL COVQ  %%%%%%%%%%%%%%%%%%%%%%%%%%
for distancia=1:1

%Seleccion de un valor de Distancia grande
D1=10;

umbral=4.7343e-014;
Y = randn(512,160);
Y=Y./100;
original=Y;
distanciatotal=10;
residuo=residuo.*100;
inicio=1;


%%%%%%%%%%%%%%%%%%%%%%%% OBTENCION DE LA DISTANCIA
%%%%%%%%%%%%%%%%%%%%%%%%%%
particion=zeros(512,160);
aumenta=9.5703e-006;

for indpart=1:512
   particion(indpart,:)=particion(indpart,:)+(indpart.*aumenta);
end


particion=particion-0.0028;
c=sum(particion')./160;
c_i=particion;
inicio=1;
r=1;
mmed=zeros(1,60);

for r=1:50

   Dprom=zeros(512,160);
   D(1)=20;
   a=zeros(2*NumVentanas-1,1);
   veces=zeros(512,1);
   particiondecontrol=zeros(512,160);



   %% Paso: Obtencion de la matriz de particion 'particion', del numero de veces
que un vector de particion codifica a un vector residuo 'veces' y de los centroides 'c'
   %% Para la primera obtencion de la distancia usamos el error cuadratico medio
```

```
	for k=1:2*NumVentanas-1

	   for rr=1:512
	      %Calculo de la distorsion promedio para ver que vector particion codifica
mejor al vector residuo
	      Dprom(rr,:)=((residuo(k,:)-particion(rr,:)).^2);

	   end


	   Dpromedio=sum(Dprom')./160;
	   % Ubicacion del vector particion que codifica al vector residuo en cuestion y
valor de la distorsion promedio
	   [Dmin(k), a(k)]=min(Dpromedio);
	   veces(a(k,1))=veces(a(k,1))+1;
	   particiondecontrol(a(k,1),:)=particiondecontrol(a(k,1),:)+residuo(k,:);
	end

	for j=1:512
	   if veces(j)==0

	   else
	      particion(j,:)=particiondecontrol(j,:)./veces(j);
	      c(j)=sum((particion(j,:)')./160);
	   end

	end

	% Paso: Ontencion de los parametros ai, bi y Mi

	K=160; %tamaño del vector a la salida del decodificador


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CALCULO DE M_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
	M_i=zeros(1,512);

	for w=1:512
	   if veces(w)==0
	      pj(w)=0.00001;
	      PI_i=1/512;
	      M_i(1,w)=0;
	   else
	      pj(w)=veces(w)/512;
	      M_i(1,w)=(sum(particiondecontrol(w,:)'))./veces(w);
	      PI_i=1/512;

	   end
	end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CALCULO DE M_i
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    sum_ai=0;
    sum_bi=0;


    %obtencion de a_i y b_i
    a_ii=zeros(512,512);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CALCULO DE ai
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    a1=zeros(512,160);

    a2=zeros(512,160);

    for itestada=1:512
        % Calculo de a siguiendo al pie de l aletra la tesis del espa;ol

        %% Primero obtenemos las sumatorias sobre los indices ^i y las j's
despues solo multiplicamos por
        %% todos los vectores de la tabla de probabilidades de transicion para
obtener a_i

        for jota=1:512

                    a1(jota,:)=pj(jota).*c_i(jota,:).*probtabla(itestada,jota);

        end

        a2(itestada,:)=sum(a1);

        a_ii(itestada,:)=pj.*c.*probtabla(itestada,:)./PI_i;

    end

    %Calculo de ai con c usado como matriz
    a2=a2./PI_i;
    a3=(sum(sum(a2)))./160;
    a_icompleja=probtabla(1,:).*a3;

    %Calculo de ai con c usado como un vector de 512 valores
    a_iant=sum(sum(a_ii'));
    a_i=probtabla(1,:).*a_iant;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HASTA AQUI YA OBTUVIMOS ai
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CALCULO DE bi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    b_ii=zeros(512,512);
    b_1=zeros(512,160);

    for bb=1:512

      for n=1:512
        b_1(n,:)=pj(n).*c_i(n,:).*probtabla(bb,n);
      end
      b_1square=b_1.^2;
      cTc(bb,:)=(sum(b_1square'))./PI_i;
      b1=pj.*c.*probtabla(bb,:);
    end

    %Calvulo de bi con c usado com o matriz
    bb_iant=sum(sum(cTc'));
    bb_i=probtabla(1,:).*bb_iant;

    %Calculo de bi con c como un vector de 512 valores
    b2=b1;
    b3=(b2.*b1)./(PI_i.^2);
    b_iant=sum(b3);
    b_i=probtabla(1,:).*b_iant;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HASTA AQUI YA OBTUVIMOS
bi%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CALCULO DE Di
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    D2ant=pj.*(M_i+b_i-(2.*(a_i.*c)));

    D2=sum(D2ant)/512;
```

```
        medida=abs((abs(D1)-abs(D2))./abs(D2))
        mmed(1,r)=medida;

        D1
        D2
        D1=D2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HASTA AQUI YA OBTUVIMOS Di
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Fin de
OBTENCION DE D(i)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

%    for kk=1:512
%        if veces(kk)~=0
```

```
%           particion(kk)=particion(kk)./veces(kk);
%           Y(kk,:)=particion(kk,:);
%       end
%   end

%   distanciatotal=abs(sum(distancias)/(2*NumVentanas-1))


% distancias guarda la distancia min entre los vectores de voz y la matriz de
particion
% a(k) guarda la posicion del vector de la matriz de particion que codifico de mejor
manera al vector de voz
% veces guarda las veces que un vector de la matriz de particiones codifica a un
vector de voz
% particion guarda la sumatoria de  los vectores que conformaran la nueva matriz
de particion


% PROCEDIMIENTO
% 1- DAR UN VALOR DE DISTANCIA INICIAL GRANDE
% 2- CALCULAR LAS PROBABILIDADES DE PARTICION QUE SE OBTIENEN POR EL
NUMERO DE VECTORES DE VOZ QUE CODIFICA CADA PARTICION
% 3- CALCULAR LOS CENTROIDES QUE ES LA SUMA DE LOS VALORES DE LOS
VECTORES DE PARTICION DIVIDIDO ENTRE 160
% 4- CALCULAR ai, bi y Mi
% 5- SI [D(ANTERIOR)-D(PRESENTE)]/D(PRESENTE) ES MENOR QUE EL UMBRAL SE
TERMINA EL PROCESO DE BUSQUEDA DEL CODIFICADOR SI NO CONTINUAR
% 6- CALCULAR LA PARTICION OPTIMA  E IR AL PASO 2
```

codsearch.m

La codificación de los vectores residuales se lleva a cabo con la siguiente rutina tomando en cuenta que para ello se debieron haber llamado a los programas anteriores.

```
%% Este programa se debe correr despes de haber de cargar las variables del
archivo codificadorfijo.mat y resi-s-prueba.mat o despues de
%% correr lpcresiduo.m y cargar la particion fija

Dp=zeros(512,160);

%% resprueba es el nombre de la matriz que contiene los vectores residuo de la
senhal de voz y se debe igualar a
%% la matriz que tenemos con los residuos
ressprueba=residuo;

filas=max(size(ressprueba));
D_min=zeros(filas,1);
indice=zeros(filas,1);
cod_optimo=zeros(filas,160);
s_codificada=zeros(filas,80);
s_vector=zeros(1,filas*80);

      for k=1:max(size(ressprueba))

         for rr=1:512

            %Calculo de la distorsion promedio para ver que vector particion codifica
mejor al vector residuo
            Dp(rr,:)=((ressprueba(k,:)-particion(rr,:)).^2);

         end


         D_promedio=sum(Dp')./160;

         % Ubicacion del vector particion que codifica al vector residuo en cuestion y
valor de la distorsion promedio
         [D_min(k), indice(k)]=min(D_promedio);

         s_codificada(k,:)=particion(indice(k,1),41:120);
         final=k*80+1;
         inicio=final-80;
         s_vector(1,inicio:final-1)=s_codificada(k,:);
      end
      %%Aqui SE RESTA uno a indice bin porque las matrices solo van de 1 en
adelante pero en binario cuenta desde cero
      indicesbin=dec2bin(indice-1);
```

fuente.m

Por medio de este programa se simula la generación de los pulsos de LED, se genera ruido modal para el archivo de excitación y se lleva a cabo la detección de voltaje después de modificar los bits con ruido para determinar si ha llegado 1 o 0 al destino.

```
%% antes de que funione este programa mandar llamar a s-codificada.mat


delta=zeros(1,300);
delta(1,100:150)=1;

[B,A]=butter(4,1000/4000);   %Obtenemos la funcion de transferencia

%Inicializamos vectores de memoria a cero
Zilength=max(length(A),length(B))-1;   %define la longitud del vextor Zi
Zi=zeros(1,Zilength);  %Inicializa el vector Zi a ceros
Zf=zeros(1,Zilength);  %Inicializa el vector Zf a ceros
%salida=zeros(1,3000);
[salida(1,:),Zf]=filter(B,A,delta(1,:),Zi);

pulso=abs(salida(1,100:199));

maximo=max(size(indicesbin));
doble=double(indicesbin);
salidafibra=zeros(maximo,900);
for i=1:maximo
   intputfibra=zeros(maximo,900);

   inputfibra(1,1:100)=pulso.*(doble(i,1)-48);
   inputfibra(1,101:200)=pulso.*(doble(i,2)-48);
   inputfibra(1,201:300)=pulso.*(doble(i,3)-48);
   inputfibra(1,301:400)=pulso.*(doble(i,4)-48);
   inputfibra(1,401:500)=pulso.*(doble(i,5)-48);
   inputfibra(1,501:600)=pulso.*(doble(i,6)-48);
   inputfibra(1,601:700)=pulso.*(doble(i,7)-48);
   inputfibra(1,701:800)=pulso.*(doble(i,8)-48);
   inputfibra(1,801:900)=pulso.*(doble(i,9)-48);

   %GENERACION DE RUIDO MODAL
   mn=((i/150)-(floor(i/150)));
   %if mn==0

      ruidomodal=randn(1,900);
      ruidomodal=ruidomodal./(max(ruidomodal));
```

```
        % Aqui puedes modificar el ruido modal variando la varianza a tu conveniencia
multiplicando "ruidomodal" por el valor de varianza
        % que desees
        ruidomodal=ruidomodal.*(.1);

        salidafibra(i,:)=inputfibra(1,:)+ruidomodal;
        i;
    end


%DETECCION DE voltaje

inputdecoder=zeros(min(size(salidafibra)),9);
inputdec_dec=zeros(min(size(salidafibra)),1);
siz=min(size(salidafibra));
s_vectorruido=zeros(1,siz*80);

%r=100;
for r=1:(min(size(salidafibra)))

    j=5;
    bit=1;
    bitelevado=9;
    while j<=900
        if salidafibra(r,j)>=0.5
            inputdecoder(r,bit)=1;
            inputdec_dec(r,1)=inputdec_dec(r,1)+(2^(bitelevado-bit));
        else
            inputdecoder(r,bit)=0;
        end
        bit=bit+1;
        j=j+100;  % aumenta la suma de j en cien para checarlo en el siguiente loop
en el mismo punto
    end
    final=r*80+1;
    inicio=final-80;
    s_vectorruido(1,inicio:final-1)=particion((inputdec_dec(r,1)+1),41:120);

end

subplot(3,1,1)
plot(inputdec_dec,'r')
subplot(3,1,2)
plot(indice)
subplot(3,1,3)
plot(indice-inputdec_dec)
```

ruido.m

Por medio de este programa se simula la generación de los pulsos de LED, se genera ruido modal para los parámetros del CS-ACELP original y se lleva a cabo la detección de voltaje después de modificar los bits con ruido para determinar si ha llegado 1 o 0 al destino.

```
%% antes de que funione este programa mandar llamar a s-codificada.mat


delta=zeros(1,300);
delta(1,100:150)=1;

[B,A]=butter(4,1000/4000);   %Obtenemos la funcion de transferencia

%Inicializamos vectores de memoria a cero
Zilength=max(length(A),length(B))-1;   %define la longitud del vextor Zi
Zi=zeros(1,Zilength);  %Inicializa el vector Zi a ceros
Zf=zeros(1,Zilength);  %Inicializa el vector Zf a ceros
%salida=zeros(1,3000);
[salida(1,:),Zf]=filter(B,A,delta(1,:),Zi);

pulso=abs(salida(1,100:199));

%% La senal output es la salida del coder de la norma ya leida en matlab
maximo=max(size(output));
intputfibra=zeros(maximo,1);
salidafibra=output;

ss=1;
for i=1:maximo
   measure=(output(i,1)-127)./2;
   if abs(measure)>1
      inputfibra(i,1)=output(i,1);
      salidafibra(i,1)=output(i,1);
   else
         inputfibra(i,1)=output(i,1);
         ruidomodal=randn(1,100);
         ruidomodal=ruidomodal./(max(ruidomodal));

         % Aqui puedes cambiar el valor de varianza de ruido y acoplarlo al
multiplicar "ruidomodal"
         % por el valor de varianza que consideres conveniente
         ruidomodal=ruidomodal.*(.1);

         salidapulso=ruidomodal+(pulso.*measure);

            if salidapulso(1,5)>=0.5
               salidafibra(i,1)=1;
            else
```

```
                salidafibra(i,1)=0;

            end
    end

    if abs(salidafibra(i,1))>1
    else
        salidafibra(i,1)=(salidafibra(i,1).*2)+127 ;
    end
    i
        ss=ss+1;
end
```

acoplesvector.m

Para la utilización del archivo de excitación en la norma del CS-ACELP modificado se utiliza la siguiente rutina.

```
s_vector=s_vectorruido;
for yo=1:max(size(s_vector))
   if s_vector(1,yo)>=0
      if s_vector(1,yo)>1
         s_vector(1,yo)=1;
      end
      s_vector(1,yo)=s_vector(1,yo).*81910;
   else
      if s_vector(1,yo)<(-1)
         s_vector(1,yo)=-1;
      end

      s_vector(1,yo)=s_vector(1,yo).*81920;
   end
end
s_vector=floor(s_vector);
```

reconvoz.m

Para el uso del Decodificador en MATLAB se utilizó una pequeña rutina de reconstrucción de la señal final en un solo vector para el manejo de este en MATLAB de tal forma que el usuario pueda acotar su amplitud y guardarla en formato wav.

```
%% RECONFORMACION DE LA VOZ
fin=40*max(size(voz_final));
ff=1;
ii=1;
jj=1;
senialdevoz=zeros(1,40*max(size(voz_final)));
vozfinal=voz_rec';
while ii<=fin
   senialdevoz(1,jj:(jj+39))=vozfinal(ff,:);
   ff=ff+1;
   jj=jj+40;
   ii=ii+40
end
```

PS.m

```
%% Para correr este programa tienes que haber mandado llamar a las señales que
contienen las señales decodificadas tanto por el CS-ACELP original como por el
vectorial previamente grabadas en un archivo ".mat". Los parámetros de estas
señales debieron haber pasado por el canal ruidoso para compararlas.

prueba1=wavread('prueba2.wav');   %%Aqui lees el archivo original de voz original

ytrm=(salidaoriginal.*(2^16))./2;  %% y generada con tramas con ruido usando
csacelp original para decodificacion
%ytrm=tramasrvoz;

nogain=(salidacsmod.*(2^16))./2;  %% y generada con tramas con ruido usando
csacelp original para decodificacion



%x=prueba1(159:121278,1);  %% Aqui igualas la sl;al a la que le quieras sacar el
PS a senial
x=prueba1(1:max(size(salidaoriginal)));
x=x./(max(abs(x)));
x=(x.*(2^16))./2;

Ps=(x.*x)+0.0000000001;

Pr_ytrm=(x-ytrm).*(x-ytrm)+0.0000000001;

Pr_nogain=(x-nogain).*(x-nogain)+0.0000000001;

SSNRCS=10.*log(Ps./(Pr_ytrm));

SSNRnogain=10.*log(Ps./(Pr_nogain));



for kk=1:24000
   if SSNRCS(kk,1)>10
      SSNRCS(kk,1)=10;
   end
   if SSNRCS(kk,1)<-10
      SSNRCS(kk,1)=-10;
   end
   if SSNRnogain(kk,1)>10
      SSNRnogain(kk,1)=10;
   end
   if SSNRnogain(kk,1)<-10
      SSNRnogain(kk,1)=-10;
   end

end
```

```
% Aqui se eliminan los picos excesivos provocados por la diferencia en decibeles
entre la señal de voz original y las decodificadas en las áreas de silencio y que
provocan un SNR y SSNR muy bajo.


for kk=1:121120
   if SSNRCS(kk,1)>10
      SSNRCS(kk,1)=0;
   end
   if SSNRCS(kk,1)<-10
      SSNRCS(kk,1)=-0;
   end

   if SSNRMIO(kk,1)>10
      SSNRMIO(kk,1)=0;
   end
   if SSNRMIO(kk,1)<-10
      SSNRMIO(kk,1)=-0;
   end

   if SSNRMIOOR(kk,1)>10
      SSNRMIOOR(kk,1)=0;
   end
   if SSNRMIOOR(kk,1)<-10
      SSNRMIOOR(kk,1)=-0;
   end
      if SSNRnogain(kk,1)>10
      SSNRnogain(kk,1)=0;
   end
   if SSNRnogain(kk,1)<-10
      SSNRnogain(kk,1)=-0;
   end

end


subplot(2,2,1)
plot(x)
subplot(2,2,2)
plot(yprcsmod,'r')
subplot(2,2,3)
plot(ytrm)
subplot(2,2,4)
plot(yprsor,'b')
```

Archivos de ayuda

Aquí se pueden observar los cambios hechos al decoder de la referencia [9]. Este es el decoder utilizado para generar los resultados desplegados en las tablas de la ***SECCIÓN 8***.

decod_ideal.m

```
function [voz_rec,voz_final]=decod_ideal(M)
% La funcion [voz_rec,voz_TL,voz_TC,voz_out,voz_final]=DECODER(M) lleva a cabo
el
%       proceso completo de decodificacion de las tramas recibidas del canal,
%       dando como resultado la reconstruccion de la señal de voz original.
% M es un vector columna compuesto por el conjunto total de tramas recibidas
%       del canal.

[ntramas,L0,L1,L2,L3,P1,P0,C1,S1,GA1,GB1,P2,C2,S2,GA2,GB2]=contador(M);

% AQUI SE LEEE EL ARCHIVO DE RESIDUOS YA CODIFICADOS POR EL
LIBRO DE CODIGOS YA SEA CON O SIN RUIDO

librocodigos=dlmread('probcsmod380.txt');

% Calculo de los parametros LPC a partir de los indices L0, L1, L2 y L3.
LPC_est=decodLPC(L0,L1,L2,L3);

% Calculo del bit de paridad a partir del campo P1 para chequear P0.
[p]=paridad(P1);

for n=1:ntramas



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Se le le indica al programa que no existen tramas con error para que no
                              intente reconstruirlas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   p(1,n)=P0(1,n);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Calculo del Pitch Delay asociado a cada subtrama, T1 para la primera, T2 para
la segunda.
   if P0(1,n)==p(1,n)
      [entT1,fracT1]=obtieneT1(P1(1,n));
      T1(1,n)=entT1;
      T1(2,n)=fracT1;
      [entT2,fracT2]=obtieneT2(entT1,P2(1,n));
      T2(1,n)=entT2;
      T2(2,n)=fracT2;
   else
      if n==1
         T1(1,n)=0;
```

```
      else
         T1(1,n)=T2(1,n-1);
      end
      T1(2,n)=0;
      %P1(1,n)=pitch1;
      [entT2,fracT2]=obtieneT2(T1(1,n),P2(1,n));
      T2(1,n)=entT2;
      T2(2,n)=fracT2;
   end
end

u(1:183,1)=0;       % Definicion del vector u(n) n=-143,..,39 (1,..183 muestras en
Matlab) que usaremos como excitacion anterior.
                    % Las muestras 144,..,183 de u(n) corresponden a la excitacion
anterior, las 1,..,143 son de tramas anteriores.
memx(1:9,1)=0;
memy(1:9,1)=0;
h(1,1:4)=-14;       % Inicializacion de las diferencias entre energias para el calculo
de ganancias.
sig(1,1:10)=0;      % PARA LA FUNCION FILTER.
r_ant(1,1:144)=0;   % Para el filtrado de termino largo.
mem_TC(1,1:10)=0;   % Para el filtrado de termino corto.

for n=1:(2*ntramas)
   impar=mod(n,2);
   % Decodificacion del vector del "Adaptive-Codebook" para cada subtrama.
   exc_prev(1:183,1)=u(1:183,1);
   if n==1
      v(1:40,1)=0;
      [v,memx,memy]=decod_AdaptiveCB(exc_prev,T1(1:2,1),memx,memy);
   else
      if impar==0
         v(1:40,1)=0;
         [v,memx,memy]=decod_AdaptiveCB(exc_prev,T2(1:2,(n/2)),memx,memy);
      else
         v(1:40,1)=0;

[v,memx,memy]=decod_AdaptiveCB(exc_prev,T1(1:2,(n+1)/2),memx,memy);
      end
   end
   V(1:40,n)=v;

   % Decodificacion del vector del "Fixed-Codebook" para cada subtrama.

   % Aqui se USA ELLIBRO DE CODIGOS QUE GENERE DIRECTAMENTE PARA
   CADA SUBTRAMA

   if n==1
      codevector(1:40,n)=librocodigos(1:40,1);
   else
      ctrm=40*(n-1);
      codevector(1:40,n)=librocodigos((ctrm+1):(ctrm+40),1);
      n
```

```
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
La sección siguiente se comenta para que el decoder no calcule el libro de
códigos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%
%   if impar==1
%      [m0,m1,m2,m3,s0,s1,s2,s3]=decod_fixedCB(C1(:,(n+1)/2),S1(:,(n+1)/2));
%
%
%
%
% %        codevector(1:40,n)=0;
% %        codevector(m0+1,n)=s0;
% %        codevector(m1+1,n)=s1;
% %        codevector(m2+1,n)=s2;
% %        codevector(m3+1,n)=s3;
%
%
%
%      aux=T1(1,(n+1)/2);
%      if aux<40
%         if n==1
%            beta=0.8;
%         else
%            beta=Gp(1,n-1);
%         end
%         a(1:(40-aux),1)=codevector((aux+1):40,n);
%         b(1:(40-aux),1)=beta*codevector(1:(40-aux),n);
%
%
%
% %          codevector((aux+1):40,n)=a(1:(40-aux),1)+b(1:(40-aux),1);
%
%
%
%      end
%   else
%      [m0,m1,m2,m3,s0,s1,s2,s3]=decod_fixedCB(C2(:,n/2),S2(:,n/2));
%
%
% %        codevector(1:40,n)=0;
% %        codevector(m0+1,n)=s0;
% %        codevector(m1+1,n)=s1;
% %        codevector(m2+1,n)=s2;
% %        codevector(m3+1,n)=s3;
%
%
%      aux=T2(1,n/2);
%      if aux<40
```

```
%           if n==1
%              beta=0.8;
%           else
%              beta=Gp(1,n-1);
%           end
%           a(1:(40-aux),1)=codevector((aux+1):40,n);
%           b(1:(40-aux),1)=beta*codevector(1:(40-aux),n);
%
%
% %            codevector((aux+1):40,n)=a(1:(40-aux),1)+b(1:(40-aux),1);
%
%
%        end
%     end
%
   % Calculo de las ganancias del Fixed-Codebook y del Adaptive-Codebook, 1°
subtrama.
   if impar==1

[Gp(1,n),Gc(1,n),GAMA(1,n),h]=deco_gain(codevector(1:40,n),GA1(1,(n+1)/2),GB1
(1,(n+1)/2),h);
   else

[Gp(1,n),Gc(1,n),GAMA(1,n),h]=deco_gain(codevector(1:40,n),GA2(1,n/2),GB2(1,n/
2),h);
   end

   % Obtencion de la señal de excitacion exc(n).
   exc(1:40,n)=(Gp(1,n)*V(1:40,n))+(Gc(1,n)*codevector(1:40,n));
   temp(1:183,1)=0;
   temp(1:143,1)=u(41:183,1);
   u(1:143,1)=temp(1:143,1);
   u(144:183,1)=exc(1:40,n);

   % Computo de la la subtrama reconstruida de señal de voz, "voz_rec".
   if impar==1
      if n==1
         voz_rec(1:40,n)=exc(1:40,n);
      else
         [voz,sig]=filter(1,[1; LPC_est(1:10,(n+1)/2)]',exc(1:40,n)',sig);
         voz_rec(1:40,n)=voz';
      end
   else
      [voz,sig]=filter(1,[1; LPC_est(1:10,n/2)]',exc(1:40,n)',sig);
      voz_rec(1:40,n)=voz';
   end

   % Pasamos ya al proceso de POSTPROCESADO de la voz.
   if impar==1
      if n==1
         % Hacemos primero el filtrado de termino largo que nos devolvera la señal
"voz_TL".
         mem_rk1(1,1:27)=0;
```

```
        mem_rk2(1,1:111)=0;

[voz_TL(1:40,n),gan_l,r_sig,mem_resid,mem_rk1,mem_rk2,condicion,R(1,n)]=filtro
TL(voz_rec(1:40,n),LPC_est(1:10,(n+1)/2),T1(1,(n+1)/2),r_ant,[0 0 0 0 0 0 0 0 0
0],mem_rk1,mem_rk2);
        GL(1,n)=gan_l;
        COND(1,n)=condicion;

        % Hacemos ahora el filtrado de termino corto que nos devolvera la señal
"voz_TC".

[voz_TC(1:40,n),mem_TC,gan_f,gan_t,mem_tilt]=filtroTC(voz_TL(1:40,n),mem_TC,
LPC_est(1:10,(n+1)/2),[0]);

        % Hacemos el control adaptable de ganancia.

[voz_out(1:40,n),g_sig]=AdapGainControl(voz_rec(1:40,n),voz_TC(1:40,n),1);

        % Por ultimo se hace el postfiltrado paso alto y el escalado.
        [voz_final(1:40,n),memoria]=postHPF(voz_out(1:40,n),[0 0]);
    else
        % Hacemos primero el filtrado de termino largo que nos devolvera la señal
"voz_TL".

[voz_TL(1:40,n),gan_l,r_sig,mem_resid,mem_rk1,mem_rk2,condicion,R(1,n)]=filtro
TL(voz_rec(1:40,n),LPC_est(1:10,(n+1)/2),T1(1,(n+1)/2),r_sig,mem_resid,mem_rk
1,mem_rk2);
        GL(1,n)=gan_l;
        COND(1,n)=condicion;

        % Hacemos ahora el filtrado de termino corto que nos devolvera la señal
"voz_TC".

[voz_TC(1:40,n),mem_TC,gan_f,gan_t,mem_tilt]=filtroTC(voz_TL(1:40,n),mem_TC,
LPC_est(1:10,(n+1)/2),mem_tilt);

        % Hacemos el control adaptable de ganancia.

[voz_out(1:40,n),g_sig]=AdapGainControl(voz_rec(1:40,n),voz_TC(1:40,n),g_sig);

        % Por ultimo se hace el postfiltrado paso alto y el escalado.
        [voz_final(1:40,n),memoria]=postHPF(voz_out(1:40,n),memoria);
    end
  else
    % Hacemos primero el filtrado de termino largo que nos devolvera la señal
"voz_TL".

[voz_TL(1:40,n),gan_l,r_sig,mem_resid,mem_rk1,mem_rk2,condicion,R(1,n)]=filtro
TL(voz_rec(1:40,n),LPC_est(1:10,n/2),T1(1,n/2),r_sig,mem_resid,mem_rk1,mem_r
k2);
    GL(1,n)=gan_l;
    COND(1,n)=condicion;
```

```
        % Hacemos ahora el filtrado de termino corto que nos devolvera la señal
"voz_TC".

[voz_TC(1:40,n),mem_TC,gan_f,gan_t,mem_tilt]=filtroTC(voz_TL(1:40,n),mem_TC,
LPC_est(1:10,n/2),mem_tilt);

        % Hacemos el control adaptable de ganancia.

[voz_out(1:40,n),g_sig]=AdapGainControl(voz_rec(1:40,n),voz_TC(1:40,n),g_sig);

        % Por ultimo se hace el postfiltrado paso alto y el escalado.
        [voz_final(1:40,n),memoria]=postHPF(voz_out(1:40,n),memoria);
    end

end
```

Las siguientes rutinas no fueron generados por mi pero fueron de gran utilidad y son indispensables para leer y escribir archivos raw que son el estándar que utiliza el CS-ACELP original para escribir los parámetros de codificación en un archivo.

rawread.m

```
function [salida]=rawread(file,size)
% Funcion que lee archivos en formato raw
% Uso: salida=rawread(arhcivo, tipo de dato)
% tipo de dato puede ser
% 'char'=8 bits por muestra
% 'short'=16 bits por muestra
% 'int'=32 bits por muestra
% 'float'=punto flotante
% Se puede omitir el tipo de dato y el programa asume tipo char
% Salida es un vector con los valores leidos tal cual del archivo raw
if(nargin<2)
  size='char';
end

fid=fopen(file,'r');
if(fid==-1)
  error('Cannot Open File');
else
  salida=fread(fid,inf,size);
end
fclose(fid);
```

rawwrite.m

```
function rawwrite(archivo,vector,tipo)
%Funcion que escribe archivos en formato raw
%USO: rawwrite(archivo,vector,tipo de dato)
%Para tipos de datos ver ayuda de rawread
%Si se omite el parametro tipo el programa
%asume un tipo char=8bits por simbolo
if(nargin<3)
  tipo='char';
end

fid=fopen(archivo,'w');

if(fid==-1)
  error('Error al abrir el archivo para escribir');
end

fwrite(fid,vector,tipo);
fclose(fid);
```

Archivos modificados en C de la norma del CS-ACELP

Los archivos que a continuación se despliegan fueron obtenidos directamente de la norma definida por el estandar de la ITU-T para el G-729. Existen diversos cambios hechos a estos archivos definidos en su mayoría en la ***SECCIÓN 7.2*** de la tesis. Los archivos faltantes no fueron modificados y no serán desplegados aquí.

de_acelp.c

```
/* Version 3.3    Last modified: December 26, 1995 */

/*-----------------------------------------------------------*
 *  Function  Decod_ACELP()                                  *
 *  ~~~~~~~~~~~~~~~~~~~~~~~                                  *
 *   Algebraic codebook decoder.                             *
 *-----------------------------------------------------------*/

#include "typedef.h"
#include "basic_op.h"
#include "ld8k.h"

extern Word16 c[L_FRAME][MAX_SAMPLES];

void Decod_ACELP(
  Word16 sign,     /* (i)     : signs of 4 pulses.                    */
  Word16 index,    /* (i)     : Positions of the 4 pulses.            */
  Word16 cod[],    /* (o) Q13 : algebraic (fixed) codebook excitation */
  Word16 frame,
  Word16 subfr
)
{

  Word16 i;
  /* Decode the positions */
  for (i=0; i<L_SUBFR; i++) {


    printf(" \n");
    printf("%d",i+subfr);
    printf(" \n");
    printf("%d",frame);
    printf(" \n");
   /* printf("%d",c[frame][i+subfr]);*/
    printf(" \n");

    printf("Aqui esta el error al leer c en de_acelp.c\n");
    printf(" \n");
```

```
    cod[i] = c[i+subfr][frame];
  }

  return;
}
```

dec_ld8k.c

```
/* Version 3.3   Last modified: December 26, 1995 */

/*---------------------------------------------------------------*
 *   Functions Init_Decod_ld8k  and Decod_ld8k                   *
 *---------------------------------------------------------------*/

#include "typedef.h"
#include "basic_op.h"
#include "ld8k.h"

/*---------------------------------------------------------------*
 *   Decoder constant parameters (defined in "ld8k.h")           *
 *---------------------------------------------------------------*
 *   L_FRAME     : Frame size.                                   *
 *   L_SUBFR     : Sub-frame size.                               *
 *   M           : LPC order.                                    *
 *   MP1         : LPC order+1                                   *
 *   PIT_MIN     : Minimum pitch lag.                            *
 *   PIT_MAX     : Maximum pitch lag.                            *
 *   L_INTERPOL  : Length of filter for interpolation            *
 *   PRM_SIZE    : Size of vector containing analysis parameters *
 *---------------------------------------------------------------*/

/*-----------------------------------------------------*
 *       Static memory allocation.                     *
 *-----------------------------------------------------*/

      /* Excitation vector */

 static Word16 old_exc[L_FRAME+PIT_MAX+L_INTERPOL];
 static Word16 *exc;

      /* Lsp (Line spectral pairs) */

 static Word16 lsp_old[M]={
          30000, 26000, 21000, 15000, 8000, 0, -8000,-15000,-21000,-26000};

      /* Filter's memory */

 static Word16 mem_syn[M];
```

```
 static Word16 sharp;          /* pitch sharpening of previous frame */
 static Word16 old_T0;         /* integer delay of previous frame    */
 static Word16 gain_code;      /* Code gain                          */
 static Word16 gain_pitch;     /* Pitch gain                         */

/*-----------------------------------------------------------------*
 *  Function Init_Decod_ld8k                                       *
 *          ~~~~~~~~~~~~~~~~                                       *
 *                                                                 *
 *  ->Initialization of variables for the decoder section.         *
 *                                                                 *
 *-----------------------------------------------------------------*/

void Init_Decod_ld8k(void)
{

  /* Initialize static pointer */

  exc = old_exc + PIT_MAX + L_INTERPOL;

  /* Static vectors to zero */

  Set_zero(old_exc, PIT_MAX+L_INTERPOL);
  Set_zero(mem_syn, M);

  sharp  = SHARPMIN;
  old_T0 = 60;
  gain_code = 0;
  gain_pitch = 0;

  Lsp_decw_reset();
  return;
}

/*-----------------------------------------------------------------*
 *  Function Decod_ld8k                                            *
 *          ~~~~~~~~~~~                                            *
 *  ->Main decoder routine.                                        *
 *                                                                 *
 *-----------------------------------------------------------------*/

void Decod_ld8k(
  Word16  parm[],     /* (i)   : vector of synthesis parameters
                           parm[0] = bad frame indicator (bfi)  */
  Word16  voicing,    /* (i)   : voicing decision from previous frame */
  Word16  synth[],    /* (o)   : synthesis speech                     */
  Word16  A_t[],      /* (o)   : decoded LP filter in 2 subframes     */
  Word16  *T0_first,  /* (o)   : decoded pitch lag in first subframe  */
  Word16 frame
)
{
  Word16  *Az;              /* Pointer on A_t   */
  Word16  lsp_new[M];       /* LSPs             */
```

```
  Word16  code[L_SUBFR];       /* ACELP codevector */

  /* Scalars */

  Word16  i, j, i_subfr;
  Word16  T0, T0_frac, index;
  Word16  bfi;
  Word32  L_temp;
  Word16 g_p, g_c;            /* fixed and adaptive codebook gain */

  Word16 bad_pitch;           /* bad pitch indicator */
  extern Flag Overflow;

printf("entrar\n");

  /* Test bad frame indicator (bfi) */

  bfi = *parm++;

  /* Decode the LSPs */

printf("1\n");
  D_lsp(parm, lsp_new, bfi);
  parm += 2;

  /* Interpolation of LPC for the 2 subframes */

printf("2\n");
  Int_qlpc(lsp_old, lsp_new, A_t);

  /* update the LSFs for the next frame */

printf("3\n");
  Copy(lsp_new, lsp_old, M);

/*----------------------------------------------------------------------*
 *         Loop for every subframe in the analysis frame                *
 *----------------------------------------------------------------------*
 * The subframe size is L_SUBFR and the loop is repeated L_FRAME/L_SUBFR  *
 *  times                                                               *
 *     - decode the pitch delay                                         *
 *     - decode algebraic code                                          *
 *     - decode pitch and codebook gains                                *
 *     - find the excitation and compute synthesis speech               *
 *----------------------------------------------------------------------*/

  Az = A_t;           /* pointer to interpolated LPC parameters */

printf("4\n");
  for (i_subfr = 0; i_subfr < L_FRAME; i_subfr += L_SUBFR)
  {

    index = *parm++;          /* pitch index */
```

```
if(i_subfr == 0)
{

  i = *parm++;            /* get parity check result */
  bad_pitch = add(bfi, i);
  if( bad_pitch == 0)
  {
    Dec_lag3(index, PIT_MIN, PIT_MAX, i_subfr, &T0, &T0_frac);
    old_T0 = T0;
  }
  else                    /* Bad frame, or parity error */
  {
    T0  =  old_T0;
    T0_frac = 0;
    old_T0 = add( old_T0, 1);
    if( sub(old_T0, PIT_MAX) > 0) {
       old_T0 = PIT_MAX;
    }
  }
   *T0_first = T0;        /* If first frame */
}
else                      /* second subframe */
{
  if( bfi == 0)
  {
    Dec_lag3(index, PIT_MIN, PIT_MAX, i_subfr, &T0, &T0_frac);
    old_T0 = T0;
  }
  else
  {
    T0  =  old_T0;
    T0_frac = 0;
    old_T0 = add( old_T0, 1);
    if( sub(old_T0, PIT_MAX) > 0) {
       old_T0 = PIT_MAX;
    }
  }
}

/*-------------------------------------------------*
 * - Find the adaptive codebook vector.            *
 *-------------------------------------------------*/

 Pred_lt_3(&exc[i_subfr], T0, T0_frac, L_SUBFR);

/*-------------------------------------------------------*
 * - Decode innovative codebook.                         *
 * - Add the fixed-gain pitch contribution to code[].    *
 *-------------------------------------------------------*/

 if(bfi != 0)       /* Bad frame */
 {
```

```
  parm[0] = Random() & (Word16)0x1fff;     /* 13 bits random */
  parm[1] = Random() & (Word16)0x000f;     /*  4 bits random */
 }
 // a partir de aqui, revisar muy bien el comportamiento del code

 printf("Aqui esta el erro Decod_ACELP (parm code frame i_subfr)\n");

 Decod_ACELP(parm[1], parm[0], code, frame, i_subfr);
 parm +=2;

 j = shl(sharp, 1);         /* From Q14 to Q15 */
 if(sub(T0, L_SUBFR) <0 ) {
    for (i = T0; i < L_SUBFR; i++) {
      code[i] = add(code[i], mult(code[i-T0], j));
    }
 }

/*-----------------------------------------------*
 * - Decode pitch and codebook gains.            *
 *-----------------------------------------------*/

 index = *parm++;      /* index of energy VQ */

 Dec_gain(index, code, L_SUBFR, bfi, &gain_pitch, &gain_code);

/*-------------------------------------------------------------*
 * - Update pitch sharpening "sharp" with quantized gain_pitch *
 *-------------------------------------------------------------*/

 sharp = gain_pitch;
 if (sub(sharp, SHARPMAX) > 0) { sharp = SHARPMAX;  }
 if (sub(sharp, SHARPMIN) < 0) { sharp = SHARPMIN;  }

/*-----------------------------------------------------*
 * - Find the total excitation.                        *
 * - Find synthesis speech corresponding to exc[].     *
 *-----------------------------------------------------*/

 if(bfi != 0)       /* Bad frame */
 {
   if (voicing == 0 ) {
      g_p = 0;
      g_c = gain_code;
   } else {
      g_p = gain_pitch;
      g_c = 0;
   }
 } else {
   g_p = gain_pitch;
   g_c = gain_code;
 }
 for (i = 0; i < L_SUBFR;  i++)
```

```
    {
      /* exc[i] = g_p*exc[i] + g_c*code[i]; */
      /* exc[i]  in Q0   g_p in Q14         */
      /* code[i] in Q13  g_code in Q1       */

      L_temp = L_mult(exc[i+i_subfr], g_p);
      L_temp = L_mac(L_temp, code[i], g_c);
      L_temp = L_shl(L_temp, 1);
      exc[i+i_subfr] = round(L_temp);
    }

    Overflow = 0;
    Syn_filt(Az, &exc[i_subfr], &synth[i_subfr], L_SUBFR, mem_syn, 0);
    if(Overflow != 0)
    {
      /* In case of overflow in the synthesis         */
      /* -> Scale down vector exc[] and redo synthesis */

      for(i=0; i<PIT_MAX+L_INTERPOL+L_FRAME; i++)
        old_exc[i] = shr(old_exc[i], 2);

      Syn_filt(Az, &exc[i_subfr], &synth[i_subfr], L_SUBFR, mem_syn, 1);
    }
    else
      Copy(&synth[i_subfr+L_SUBFR-M], mem_syn, M);

    Az += MP1;    /* interpolated LPC parameters for next subframe */
  }

 /*-------------------------------------------------*
  * Update signal for next frame.                   *
  * -> shift to the left by L_FRAME  exc[]          *
  *-------------------------------------------------*/

printf("5\n");
  Copy(&old_exc[L_FRAME], &old_exc[0], PIT_MAX+L_INTERPOL);

printf("salir\n");
  return;
}
```

decoder.c

```
/* Version 3.3    Last modified: December 26, 1995 */

/*
   ITU-T G.729 Speech Coder     ANSI-C Source Code
   Copyright (c) 1995, AT&T, France Telecom, NTT, Universite de Sherbrooke.
   All rights reserved.
*/
```

```
/*-----------------------------------------------------------------*
 * Main program of the ITU-T G.729  8 kbit/s decoder.              *
 *                                                                 *
 *    Usage : decoder  bitstream_file  synth_file                  *
 *                                                                 *
 *-----------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include "typedef.h"
#include "basic_op.h"
#include "ld8k.h"

/*-----------------------------------------------------------------*
 *          Main decoder routine                                   *
 *-----------------------------------------------------------------*/

Word16 c[L_FRAME][MAX_SAMPLES];

int main(int argc, char *argv[] )
{
  Word16  synth_buf[L_FRAME+M], *synth; /* Synthesis                 */
  Word16  parm[PRM_SIZE+1];           /* Synthesis parameters        */
  Word16  serial[SERIAL_SIZE];        /* Serial stream               */
  Word16  Az_dec[MP1*2], *ptr_Az;     /* Decoded Az for post-filter  */
  Word16  T0_first;                   /* Pitch lag in 1st subframe   */
  Word16  pst_out[L_FRAME];           /* Postfilter output           */

  Word16  voicing;                    /* voicing from previous frame */
  Word16  sf_voic;                    /* voicing for subframe        */

  Word16  i,j, frame;
  FILE   *f_syn, *f_serial, *f_fixcode;

  char *tmp_cod = NULL;

  int len = 0;
  int continued = 1;

  printf("\n");
  printf("************    ITU G.729 8 KBIT/S SPEECH CODER    ************\n");
  printf("\n");
  printf("------------------- Fixed point C simulation ----------------\n");
  printf("\n");
  printf("----------------         Version 3.3       ----------------\n");
  printf("\n");

  /* Passed arguments */

  if ( argc != 4 )
    {
```

```
        printf("Usage :%s bitstream_file  outputspeech_file fixcode_file\n",argv[0]);
        printf("\n");
        printf("Format for bitstream_file:\n");
        printf("  One (2-byte) synchronization word,\n");
        printf("  One (2-byte) size word,\n");
        printf("  80 words (2-byte) containing 80 bits.\n");
        printf("\n");
        printf("Format for outputspeech_file:\n");
        printf("  Output is written to a binary file of 16 bits data.\n");
        exit( 1 );
      }

    /* Open file for synthesis and packed serial stream */

    if( (f_serial = fopen(argv[1],"rb") ) == NULL )
      {
        printf("%s - Error opening file  %s !!\n", argv[0], argv[1]);
        exit(0);
      }

    if( (f_syn = fopen(argv[2], "wb") ) == NULL )
      {
        printf("%s - Error opening file  %s !!\n", argv[0], argv[2]);
        exit(0);
      }
    if( (f_fixcode = fopen(argv[3], "rb") ) == NULL )
      {
        printf("%s - Error opening file  %s !!\n", argv[0], argv[3]);
        exit(0);
      }
    printf("a-\n");
    printf("Input bitstream file  :   %s\n",argv[1]);
    printf("b-\n");
    printf("Synthesis speech file :   %s\n",argv[2]);
    printf("c-\n");
    printf("Fixcode file :   %s\n",argv[3]);
printf("d-\n");

printf("1-\n");

  for (i=0; i<M; i++) synth_buf[i] = 0;
  synth = synth_buf + M;

  Init_Decod_ld8k();
  Init_Post_Filter();
  Init_Post_Process();
  voicing = 60;

  printf("2-\n");
/*-----------------------------------------------------------------*
 *          Loop for each "L_FRAME" speech data                    *
 *-----------------------------------------------------------------*/
```

```
// lees tu archivo de 80,000 muestras y la guardas en un arreglo
//

printf("3-\n");
/* for (i = 0; i < (MAX_SAMPLES/L_FRAME); i++) {*/
 for (i = 0; i < (2341); i++) {

  for (j = 0; j < L_FRAME; j++) {
   if ((getline(&tmp_cod, &len, f_fixcode)) == -1) {
    continued = 0;
      break;
   }
   /*ojo cambie la jota poor la i y la i por la jota*/
   c[j][i] = (Word16) atoi(tmp_cod);
  }
  if (!continued) {
   break;
  }
 }

printf("4-\n");
 if (tmp_cod) {
  free(tmp_cod);
 }
printf("5-\n");
 ////////////////////////
 frame = 0;
 while( fread(serial, sizeof(Word16), SERIAL_SIZE, f_serial) == SERIAL_SIZE)
 {
  bits2prm_ld8k( &serial[2], &parm[1]);

  /* the hardware detects frame erasures by checking if all bits
     are set to zero
   */
  parm[0] = 0;          /* No frame erasure */
  for (i=2; i < SERIAL_SIZE; i++)
   if (serial[i] == 0 ) parm[0] = 1; /* frame erased     */

  /* check parity and put 1 in parm[4] if parity error */

  parm[4] = Check_Parity_Pitch(parm[3], parm[4]);

  Decod_ld8k(parm, voicing, synth, Az_dec, &T0_first, frame);

  /* Postfilter */

  voicing = 0;
  ptr_Az = Az_dec;
  for(i=0; i<L_FRAME; i+=L_SUBFR) {
    Post(T0_first, &synth[i], ptr_Az, &pst_out[i], &sf_voic);
    if (sf_voic != 0) { voicing = sf_voic;}
    ptr_Az += MP1;
  }
```

```
   Copy(&synth_buf[L_FRAME], &synth_buf[0], M);

   Post_Process(pst_out, L_FRAME);

#ifdef HARDW
   {
     Word16 *my_pt;
     Word16 my_temp;
     int my_i;
     my_pt = pst_out;
     for(my_i=0; my_i < L_FRAME; my_i++) {
       my_temp = *my_pt;
       my_temp = add( my_temp, (Word16) 4); /* rounding on 13 bit */
       my_temp = my_temp & 0xFFF8; /* mask on 13 bit */
       *my_pt++ = my_temp;
     }
   }
#endif

   fwrite(pst_out, sizeof(Word16), L_FRAME, f_syn);
   frame++;
   printf("Frame =%d\r", frame);
  }
  return(0);
}
```

ld8k.c

```
/* Version 3.3    Last modified: December 26, 1995 */

/*-----------------------------------------------------------*
 * LD8K.H                                                    *
 * ~~~~~~~                                                   *
 * Function prototypes and constants use in G.729            *
 *                                                           *
 *-----------------------------------------------------------*/

/*---------------------------------------------------------------------*
 *       Codec constant parameters (coder, decoder, and postfilter)    *
 *---------------------------------------------------------------------*/

#define L_TOTAL     240       /* Total size of speech buffer.          */
#define L_WINDOW    240       /* Window size in LP analysis.           */
#define L_NEXT      40        /* Lookahead in LP analysis.             */
#define L_FRAME     80        /* Frame size.                           */
#define L_SUBFR     40        /* Subframe size.                        */
#define M           10        /* Order of LP filter.                   */
#define MP1         (M+1)     /* Order of LP filter + 1                */
#define MM1         (M-1)     /* Order of LP filter - 1                */
#define PIT_MIN     20        /* Minimum pitch lag.                    */
```

```
#define PIT_MAX     143       /* Maximum pitch lag.                     */
#define L_INTERPOL  (10+1)    /* Length of filter for interpolation.    */

#define PRM_SIZE    11        /* Size of vector of analysis parameters.  */
#define SERIAL_SIZE (80+2)    /* Bits/frame + bfi+ number of speech bits */

#define SHARPMAX    13017     /* Maximum value of pitch sharpening 0.8 Q14 */
#define SHARPMIN    3277      /* Minimum value of pitch sharpening 0.2 Q14 */

#define L_SUBFRP1 (L_SUBFR + 1)

#define GPCLIP      15564     /* Maximum pitch gain if taming is needed Q14*/
#define GPCLIP2     481       /* Maximum pitch gain if taming is needed Q9 */
#define GP0999      16383     /* Maximum pitch gain if taming is needed    */
#define L_THRESH_ERR 983040000L /* Error threshold taming 16384. * 60000.  */

#define MAX_SAMPLES 187280
/*-----------------------------------------------------------------------*
 * Mathematic functions.                                                 *
 *-----------------------------------------------------------------------*/

Word32 Inv_sqrt(    /* (o) Q30 : output value   (range: 0<=val<1)         */
  Word32 L_x        /* (i) Q0  : input value    (range: 0<=val<=7fffffff) */
);

void Log2(
  Word32 L_x,       /* (i) Q0  : input value                              */
  Word16 *exponent, /* (o) Q0  : integer part of Log2.   (range: 0<=val<=30)*/
  Word16 *fraction  /* (o) Q15 : fractional part of Log2. (range: 0<=val<1) */
);

Word32 Pow2(        /* (o) Q0  : result       (range: 0<=val<=0x7fffffff) */
  Word16 exponent,  /* (i) Q0  : integer part.      (range: 0<=val<=30)   */
  Word16 fraction   /* (i) Q15 : fractional part.  (range: 0.0<=val<1.0)   */
);

/*-----------------------------------------------------------------------*
 * Pre and post-process.                                                 *
 *-----------------------------------------------------------------------*/

void Init_Pre_Process(void);
void Init_Post_Process(void);

void Pre_Process(
  Word16 signal[],  /* (i/o)   : input/output signal                     */
  Word16 lg         /* (i)     : length of signal                        */
);

void Post_Process(
  Word16 signal[],  /* (i/o)   : input/output signal                     */
  Word16 lg         /* (i)     : length of signal                        */
);
```

```
/*-----------------------------------------------------------------------*
 * Main coder and decoder functions                                      *
 *-----------------------------------------------------------------------*/

void Init_Coder_ld8k(void);

void Coder_ld8k(
  Word16 ana[],    /* (o)    : analysis parameters                      */
  Word16 synth[]   /* (o)    : local synthesis                          */
);

void Init_Decod_ld8k(void);

void Decod_ld8k(
  Word16 parm[],   /* (i)    : vector of synthesis parameters
                               parm[0] = bad frame indicator (bfi)      */
  Word16 voicing,  /* (i)    : voicing decision from previous frame     */
  Word16 synth[],  /* (o)    : synthesized speech                       */
  Word16 A_t[],    /* (o)    : decoded LP filter for 2 subframes        */
  Word16 *T0_first, /* (o)   : decoded pitch lag in first subframe      */
  Word16 frame
);

/*-----------------------------------------------------------------------*
 * LPC analysis and filtering                                            *
 *-----------------------------------------------------------------------*/

void Autocorr(
  Word16 x[],      /* (i)    : input signal                             */
  Word16 m,        /* (i)    : LPC order                               */
  Word16 r_h[],    /* (o)    : autocorrelations  (msb)                  */
  Word16 r_l[]     /* (o)    : autocorrelations  (lsb)                  */
);

void Lag_window(
  Word16 m,        /* (i)    : LPC order                               */
  Word16 r_h[],    /* (i/o)  : autocorrelations  (msb)                  */
  Word16 r_l[]     /* (i/o)  : autocorrelations  (lsb)                  */
);

void Levinson(
  Word16 Rh[],     /* (i)    : Rh[m+1] autocorrelation coefficients (msb) */
  Word16 Rl[],     /* (i)    : Rl[m+1] autocorrelation coefficients (lsb) */
  Word16 A[],      /* (o) Q12 : A[m]   LPC coefficients  (m = 10)       */
  Word16 rc[]      /* (o) Q15 : rc[M]  Reflection coefficients.         */
);

void Az_lsp(
  Word16 a[],      /* (i) Q12 : predictor coefficients                  */
  Word16 lsp[],    /* (o) Q15 : line spectral pairs                     */
  Word16 old_lsp[] /* (i)    : old lsp[] (in case not found 10 roots)   */
);
```

```
void Lsp_Az(
  Word16 lsp[],    /* (i) Q15 : line spectral frequencies            */
  Word16 a[]       /* (o) Q12 : predictor coefficients (order = 10)  */
);

void Lsf_lsp(
  Word16 lsf[],    /* (i) Q15 : lsf[m] normalized (range: 0.0<=val<=0.5)  */
  Word16 lsp[],    /* (o) Q15 : lsp[m] (range: -1<=val<1)                 */
  Word16 m         /* (i)     : LPC order                                 */
);

void Lsp_lsf(
  Word16 lsp[],    /* (i) Q15 : lsp[m] (range: -1<=val<1)                 */
  Word16 lsf[],    /* (o) Q15 : lsf[m] normalized (range: 0.0<=val<=0.5)  */
  Word16 m         /* (i)     : LPC order                                 */
);

void Int_lpc(
  Word16 lsp_old[],  /* (i)    : LSP vector of past frame                */
  Word16 lsp_new[],  /* (i)    : LSP vector of present frame             */
  Word16 lsf_int[],  /* (o)    : interpolated lsf coefficients           */
  Word16 lsf_new[],  /* (o)    : new lsf coefficients                    */
  Word16 Az[]        /* (o)    : interpolated Az() for the 2 subframes   */
);

void Int_qlpc(
  Word16 lsp_old[],  /* (i)    : LSP vector of past frame                */
  Word16 lsp_new[],  /* (i)    : LSP vector of present frame             */
  Word16 Az[]        /* (o)    : interpolated Az() for the 2 subframes   */
);

/*-----------------------------------------------------------------------*
 *      PWF constant parameters                                          *
 *-----------------------------------------------------------------------*/

#define A1         4567      /*  2.23   in Q11                          */
#define L_B1       3271557L  /*  0.78   in Q22                          */
#define A2         11776     /*  5.75   in Q11                          */
#define L_B2       16357786L /*  3.90   in Q22                          */
#define A3         27443     /* 13.40   in Q11                          */
#define L_B3       46808433L /* 11.16   in Q22                          */
#define SEG1       1299      /*  0.6341 in Q11                          */
#define SEG2       1815      /*  0.8864 in Q11                          */
#define SEG3       1944      /*  0.9490 in Q11                          */

#define THRESH_L1  -3562     /* -1.74   in Q11                          */
#define THRESH_L2  -3116     /* -1.52   in Q11                          */
#define THRESH_H1  1336      /*  0.65   in Q11                          */
#define THRESH_H2  890       /*  0.43   in Q11                          */

#define GAMMA1_0   32113     /*  0.98   in Q15                          */
#define GAMMA1_1   30802     /*  0.94   in Q15                          */
#define GAMMA2_0_L 13107     /*  0.40   in Q15                          */
```

```
#define GAMMA2_0_H  22938     /*  0.70   in Q15                    */
#define GAMMA2_1    19661     /*  0.60   in Q15                    */

#define ALPHA       19302     /*  6*pi   in Q10                    */
#define BETA        1024      /*    1    in Q10                    */

void perc_var (
  Word16 *gamma1,  /* bandwidth expansion parameter                */
  Word16 *gamma2,  /* bandwidth expansion parameter                */
  Word16 *lsfint,  /* Interpolated LSP vector : 1st subframe       */
  Word16 *lsfnew,  /* New LSP vector : 2nd subframe                */
  Word16 *r_c      /* Reflection coefficients                      */
);

void Weight_Az(
  Word16 a[],      /* (i) Q12 : a[m+1]  LPC coefficients           */
  Word16 gamma,    /* (i) Q15 : Spectral expansion factor.         */
  Word16 m,        /* (i)     : LPC order.                         */
  Word16 ap[]      /* (o) Q12 : Spectral expanded LPC coefficients */
);

void Residu(
  Word16 a[],      /* (i) Q12 : prediction coefficients            */
  Word16 x[],      /* (i)     : speech (values x[-m..-1] are needed (m=10) */
  Word16 y[],      /* (o)     : residual signal                    */
  Word16 lg        /* (i)     : frame size                         */
);

void Syn_filt(
  Word16 a[],      /* (i) Q12 : a[m+1] prediction coefficients   (m=10)  */
  Word16 x[],      /* (i)     : input signal                       */
  Word16 y[],      /* (o)     : output signal                      */
  Word16 lg,       /* (i)     : size of filtering                  */
  Word16 mem[],    /* (i/o)   : memory associated with this filtering.   */
  Word16 update    /* (i)     : 0=no update, 1=update of memory.   */
);

void Convolve(
  Word16 x[],      /* (i)     : input vector                       */
  Word16 h[],      /* (i) Q12 : impulse response                   */
  Word16 y[],      /* (o)     : output vector                      */
  Word16 L         /* (i)     : vector size                        */
);

/*-----------------------------------------------------------------*
 *      LTP constant parameters                                    *
 *-----------------------------------------------------------------*/

#define THRESHPIT   27853     /* Threshold to favor small pitch 0.85  Q15  */
#define UP_SAMP     3         /* upsampling factor for fractional pitch    */
#define L_INTER4    4         /* length/2 for interpolation filter         */
#define FIR_SIZE_ANA (UP_SAMP*L_INTER4+1)
#define L_INTER10   10        /* length/2 for interpolation filter         */
```

```
#define FIR_SIZE_SYN (UP_SAMP*L_INTER10+1)

/*-----------------------------------------------------------------------*
 * Pitch functions.                                                      *
 *-----------------------------------------------------------------------*/

  Word16 Pitch_ol(  /* (o)    : open loop pitch lag                      */
  Word16 signal[],  /* (i)    : signal used to compute the open loop pitch */
                    /*      signal[-pit_max] to signal[-1] should be known */
  Word16 pit_min,   /* (i)    : minimum pitch lag                        */
  Word16 pit_max,   /* (i)    : maximum pitch lag                        */
  Word16 L_frame    /* (i)    : length of frame to compute pitch         */
);

  Word16 Pitch_fr3( /* (o)    : pitch period.                            */
  Word16 exc[],     /* (i)    : excitation buffer                        */
  Word16 xn[],      /* (i)    : target vector                            */
  Word16 h[],       /* (i) Q12 : impulse response of filters.            */
  Word16 L_subfr,   /* (i)    : length of subframe                       */
  Word16 t0_min,    /* (i)    : minimum value in the searched range.     */
  Word16 t0_max,    /* (i)    : maximum value in the searched range.     */
  Word16 i_subfr,   /* (i)    : indicator for first subframe.            */
  Word16 *pit_frac  /* (o)    : chosen fraction.                         */
);

Word16 G_pitch(     /* (o) Q14 : Gain of pitch lag saturated to 1.2      */
  Word16 xn[],      /* (i)    : Pitch target.                            */
  Word16 y1[],      /* (i)    : Filtered adaptive codebook.              */
  Word16 g_coeff[], /* (i)    : Correlations need for gain quantization. */
  Word16 L_subfr    /* (i)    : Length of subframe.                      */
);

Word16 Enc_lag3(    /* (o)    : Return index of encoding                 */
  Word16 T0,        /* (i)    : Pitch delay                              */
  Word16 T0_frac,   /* (i)    : Fractional pitch delay                   */
  Word16 *T0_min,   /* (i/o)  : Minimum search delay                     */
  Word16 *T0_max,   /* (i/o)  : Maximum search delay                     */
  Word16 pit_min,   /* (i)    : Minimum pitch delay                      */
  Word16 pit_max,   /* (i)    : Maximum pitch delay                      */
  Word16 pit_flag   /* (i)    : Flag for 1st subframe                    */
);

void Dec_lag3(      /* (o)    : return integer pitch lag                 */
  Word16 index,     /* (i)    : received pitch index                     */
  Word16 pit_min,   /* (i)    : minimum pitch lag                        */
  Word16 pit_max,   /* (i)    : maximum pitch lag                        */
  Word16 i_subfr,   /* (i)    : subframe flag                            */
  Word16 *T0,       /* (o)    : integer part of pitch lag                */
  Word16 *T0_frac   /* (o)    : fractional part of pitch lag             */
);

Word16 Interpol_3(  /* (o)    : interpolated value                       */
  Word16 *x,        /* (i)    : input vector                             */
```

```
  Word16 frac      /* (i)     : fraction                          */
);

void Pred_lt_3(
  Word16 exc[],    /* (i/o)   : excitation buffer                 */
  Word16 T0,       /* (i)     : integer pitch lag                 */
  Word16 frac,     /* (i)     : fraction of lag                   */
  Word16 L_subfr   /* (i)     : subframe size                     */
);

Word16 Parity_Pitch( /* (o)    : parity bit (XOR of 6 MSB bits)    */
  Word16 pitch_index /* (i)    : index for which parity to compute */
);

Word16 Check_Parity_Pitch( /* (o) : 0 = no error, 1= error        */
  Word16 pitch_index, /* (i)   : index of parameter               */
  Word16 parity      /* (i)   : parity bit                        */
);

/*-------------------------------------------------------------------*
 * fixed codebook excitation.                                        *
 *-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*
 *      FCB constant parameters                                      *
 *-------------------------------------------------------------------*/

#define DIM_RR      616       /* size of correlation matrix         */
#define NB_POS      8         /*                                    */
#define STEP        5         /* spacing for individual pulse       */
#define MSIZE       64
#define THRESHFCB   13107     /* 0.4 in Q15                         */
#define MAX_TIME    75        /* maximum number of iterations       */

/*-------------------------------------------------------------------*
 * FCB functions.                                                    *
 *-------------------------------------------------------------------*/

Word16 ACELP_Codebook( /* (o)  : index of pulses positions         */
  Word16 x[],      /* (i)     : Target vector                     */
  Word16 h[],      /* (i) Q12 : Impulse response of filters       */
  Word16 T0,       /* (i)     : Pitch lag                         */
  Word16 pitch_sharp,/* (i) Q14: Last quantized pitch gain        */
  Word16 i_subfr,  /* (i)     : Indicator of 1st subframe,        */
  Word16 code[],   /* (o) Q13 : Innovative codebook               */
  Word16 y[],      /* (o) Q12 : Filtered innovative codebook      */
  Word16 *sign     /* (o)     : Signs of 4 pulses                 */
);

void Decod_ACELP(
  Word16 sign,     /* (i)     : signs of 4 pulses.                */
  Word16 index,    /* (i)     : Positions of the 4 pulses.        */
  Word16 cod[],    /* (o) Q13 : algebraic (fixed) codebook excitation */
```

```
  Word16 frame,
  Word16 subfr
);

/*-----------------------------------------------------------------------*
 *      LSP quantizer constant parameters                                *
 *-----------------------------------------------------------------------*/

#define NC         5         /*  NC = M/2                                */
#define MA_NP      4         /* MA prediction order for LSP              */
#define MODE       2         /* number of modes for MA prediction        */
#define NC0_B      7         /* number of first stage bits               */
#define NC1_B      5         /* number of second stage bits              */
#define NC0        (1<<NC0_B) /* number of entries in first stage        */
#define NC1        (1<<NC1_B) /* number of entries in second stage       */

#define L_LIMIT    40        /* minimum lsf value Q13:0.005              */
#define M_LIMIT    25681     /* maximum lsf value Q13:3.135              */

#define GAP1       10        /* bandwidth expansion factor Q13           */
#define GAP2       5         /* bandwidth expansion factor Q13           */
#define GAP3       321       /* bandwidth expansion factor Q13           */
#define GRID_POINTS 60       /* search grid                              */

#define PI04       ((Word16)1029)      /* Q13  pi*0.04                   */
#define PI92       ((Word16)23677)     /* Q13  pi*0.92                   */
#define CONST10    ((Word16)10*(1<<11)) /* Q11  10.0                     */
#define CONST12    ((Word16)19661)     /* Q14  1.2                       */

/*-----------------------------------------------------------------------*
 * LSP VQ functions.                                                     *
 *-----------------------------------------------------------------------*/

void Lsf_lsp2(
  Word16 lsf[],     /* (i) Q13 : lsf[m] (range: 0.0<=val<PI)             */
  Word16 lsp[],     /* (o) Q15 : lsp[m] (range: -1<=val<1)               */
  Word16 m          /* (i)     : LPC order                               */
);

void Lsp_lsf2(
  Word16 lsp[],     /* (i) Q15 : lsp[m] (range: -1<=val<1)               */
  Word16 lsf[],     /* (o) Q13 : lsf[m] (range: 0.0<=val<PI)             */
  Word16 m          /* (i)     : LPC order                               */
);

void Qua_lsp(
  Word16 lsp[],     /* (i) Q15 : Unquantized LSP                         */
  Word16 lsp_q[],   /* (o) Q15 : Quantized LSP                           */
  Word16 ana[]      /* (o)     : indexes                                 */
);

void Get_wegt(
  Word16 flsp[],    /* (i) Q13 :                                         */
```

```
  Word16 wegt[]    /* (o) Q11 :  normalized                    */
);

void Lsp_encw_reset( void);

void Lsp_qua_cs(
  Word16 flsp_in[M],  /* Q13 */
  Word16 lspq_out[M], /* Q13 */
  Word16 *code
);

void Lsp_expand_1(
  Word16 buf[],    /* Q13 */
  Word16 gap       /* Q13 */
);

void Lsp_expand_2(
  Word16 buf[],    /* Q13 */
  Word16 gap       /* Q13 */
);

void Lsp_expand_1_2(
  Word16 buf[],    /* Q13 */
  Word16 gap       /* Q13 */
);

void Lsp_get_quant(
  Word16 lspcb1[][M],    /* Q13 */
  Word16 lspcb2[][M],    /* Q13 */
  Word16 code0,
  Word16 code1,
  Word16 code2,
  Word16 fg[][M],        /* Q15 */
  Word16 freq_prev[][M], /* Q13 */
  Word16 lspq[],         /* Q13 */
  Word16 fg_sum[]          /* Q15 */
);

void Lsp_get_tdist(
  Word16 wegt[],   /* normalized */
  Word16 buf[],    /* Q13 */
  Word32 *L_tdist, /* Q27 */
  Word16 rbuf[],   /* Q13 */
  Word16 fg_sum[]  /* Q15 */
);

void Lsp_last_select(
  Word32 L_tdist[], /* Q27 */
  Word16 *mode_index
);

void Lsp_pre_select(
  Word16 rbuf[],       /* Q13 */
```

```
  Word16 lspcb1[][M], /* Q13 */
  Word16 *cand
);

void Lsp_select_1(
  Word16 rbuf[],       /* Q13 */
  Word16 lspcb1[],     /* Q13 */
  Word16 wegt[],       /* normalized */
  Word16 lspcb2[][M], /* Q13 */
  Word16 *index
);

void Lsp_select_2(
  Word16 rbuf[],       /* Q13 */
  Word16 lspcb1[],     /* Q13 */
  Word16 wegt[],       /* normalized */
  Word16 lspcb2[][M], /* Q13 */
  Word16 *index
);

void Lsp_stability(
  Word16 buf[]    /* Q13 */
);

void Relspwed(
  Word16 lsp[],                    /* Q13 */
  Word16 wegt[],                   /* normalized */
  Word16 lspq[],                   /* Q13 */
  Word16 lspcb1[][M],              /* Q13 */
  Word16 lspcb2[][M],              /* Q13 */
  Word16 fg[MODE][MA_NP][M],       /* Q15 */
  Word16 freq_prev[MA_NP][M],      /* Q13 */
  Word16 fg_sum[MODE][M],          /* Q15 */
  Word16 fg_sum_inv[MODE][M],      /* Q12 */
  Word16 code_ana[]
);

void D_lsp(
  Word16 prm[],    /* (i)    : indexes of the selected LSP          */
  Word16 lsp_q[],  /* (o) Q15 : Quantized LSP parameters             */
  Word16 erase     /* (i)    : frame erase information              */
);

void Lsp_decw_reset( void);

void Lsp_iqua_cs(
  Word16 prm[],    /* (i)    : codes of the selected LSP            */
  Word16 lsp_q[],  /* (o)    : Quantized LSP parameters             */
  Word16 erase     /* (i)    : frame erase information              */
);

void Lsp_prev_compose(
  Word16 lsp_ele[], /* Q13 */
```

```
  Word16 lsp[],     /* Q13 */
  Word16 fg[][M],/* Q15 */
  Word16 freq_prev[][M],     /* Q13 */
  Word16 fg_sum[]   /* Q15 */
);

void Lsp_prev_extract(
  Word16 lsp[M],              /* Q13 */
  Word16 lsp_ele[M],           /* Q13 */
  Word16 fg[MA_NP][M],         /* Q15 */
  Word16 freq_prev[MA_NP][M],    /* Q13 */
  Word16 fg_sum_inv[M]         /* Q12 */
);

void Lsp_prev_update(
  Word16 lsp_ele[M],           /* Q13 */
  Word16 freq_prev[MA_NP][M]     /* Q13 */
);

/*-----------------------------------------------------------------------*
 * gain VQ constants.                                                    *
 *-----------------------------------------------------------------------*/

#define NCODE1_B  3              /* number of Codebook-bit              */
#define NCODE2_B  4              /* number of Codebook-bit              */
#define NCODE1    (1<<NCODE1_B)    /* Codebook 1 size                     */
#define NCODE2    (1<<NCODE2_B)    /* Codebook 2 size                     */
#define NCAN1     4              /* Pre-selecting order for #1          */
#define NCAN2     8              /* Pre-selecting order for #2          */
#define INV_COEF  -17103         /* Q19                                 */

/*-----------------------------------------------------------------------*
 * gain VQ functions.                                                    *
 *-----------------------------------------------------------------------*/

Word16 Qua_gain(
  Word16 code[],    /* (i) Q13 : Innovative vector.                     */
  Word16 g_coeff[], /* (i)     : Correlations <xn y1> -2<y1 y1>         */
                    /*           <y2,y2>, -2<xn,y2>, 2<y1,y2>           */
  Word16 exp_coeff[],/* (i)    : Q-Format g_coeff[]                     */
  Word16 L_subfr,   /* (i)     : Subframe length.                       */
  Word16 *gain_pit, /* (o) Q14 : Pitch gain.                            */
  Word16 *gain_cod, /* (o) Q1  : Code gain.                             */
  Word16 tameflag   /* (i)     : flag set to 1 if taming is needed      */
);

void Dec_gain(
  Word16 index,     /* (i)     : Index of quantization.                 */
  Word16 code[],    /* (i) Q13 : Innovative vector.                     */
  Word16 L_subfr,   /* (i)     : Subframe length.                       */
  Word16 bfi,       /* (i)     : Bad frame indicator                    */
  Word16 *gain_pit, /* (o) Q14 : Pitch gain.                            */
  Word16 *gain_cod  /* (o) Q1  : Code gain.                             */
```

```
);

void Gain_predict(
  Word16 past_qua_en[],/* (i) Q10 :Past quantized energies              */
  Word16 code[],    /* (i) Q13 : Innovative vector.                    */
  Word16 L_subfr,   /* (i)     : Subframe length.                      */
  Word16 *gcode0,   /* (o) Qxx : Predicted codebook gain               */
  Word16 *exp_gcode0 /* (o)    : Q-Format(gcode0)                      */
);

void Gain_update(
  Word16 past_qua_en[],/* (i) Q10 :Past quantized energies             */
  Word32 L_gbk12    /* (i) Q13 : gbk1[indice1][1]+gbk2[indice2][1]     */
);

void Gain_update_erasure(
  Word16 past_qua_en[]/* (i) Q10 :Past quantized energies              */
);

void Corr_xy2(
  Word16 xn[],      /* (i) Q0  :Target vector.                         */
  Word16 y1[],      /* (i) Q0  :Adaptive codebook.                     */
  Word16 y2[],      /* (i) Q12 :Filtered innovative vector.            */
  Word16 g_coeff[], /* (o) Q[exp]:Correlations between xn,y1,y2        */
  Word16 exp_g_coeff[]  /* (o)  :Q-format of g_coeff[]                 */
);

/*----------------------------------------------------------------------*
 * Postfilter constants and functions                                   *
 *----------------------------------------------------------------------*/

/* short term pst parameters :                                         */
#define GAMMA1_PST  22938     /* denominator weighting factor   (Q15)  */
#define GAMMA2_PST  18022     /* numerator  weighting factor (Q15)     */
#define LONG_H_ST   20        /* impulse response length               */
#define GAMMA3_PLUS 6554      /* tilt weighting factor when k1>0 (Q15) */
#define GAMMA3_MINUS 29491    /* tilt weighting factor when k1<0 (Q15) */

/* long term pst parameters :                                          */
#define F_UP_PST    8         /* resolution for fractional delay       */
#define LH2_S       4         /* length of short interp. subfilters    */
#define L2_LH2_L    4         /* log2(LH2_L)                           */
#define LH2_L       (1 << L2_LH2_L)
#define MIN_GPLT    21845     /* LT gain minimum (Q15)                 */

#define LH_UP_S     (LH2_S/2)
#define LH_UP_SM1   (LH_UP_S-1)
#define LH_UP_L     (LH2_L/2)
#define LH2_L_P1    (LH2_L + 1)

/* gain adjustment parameters                                          */
#define AGC_FAC     32358     /* gain adjustment factor  0.9875 (Q15)  */
#define AGC_FAC1    (Word16)(32768L - AGC_FAC)
```

```
/* Array sizes */
#define MEM_RES2    (PIT_MAX + 1 + LH_UP_L)
#define SIZ_RES2    (MEM_RES2 + L_SUBFR)
#define SIZ_Y_UP    ((F_UP_PST-1) * L_SUBFRP1)
#define SIZ_TAB_HUP_L ((F_UP_PST-1) * LH2_L)
#define SIZ_TAB_HUP_S ((F_UP_PST-1) * LH2_S)

void  Init_Post_Filter( void);

void Post(
  Word16 t0,       /* (i) : 1st subframe delay given by coder            */
  Word16 *signal_ptr, /* (i) : input signal (pointer to current subframe   */
  Word16 *coeff,    /* (i) : LPC coefficients for current subframe        */
  Word16 *sig_out,  /* (o) : postfiltered output                          */
  Word16 *vo        /* (o) : voicing decision 0 = uv,  > 0 delay          */
);

/*-----------------------------------------------------------------------*
 * Bitstream constants and functions                                     *
 *-----------------------------------------------------------------------*/

#define BIT_0    (short)0x007f /* definition of zero-bit in bit-stream      */
#define BIT_1    (short)0x0081 /* definition of one-bit in bit-stream       */
#define SYNC_WORD (short)0x6b21 /* definition of frame erasure flag         */
#define SIZE_WORD (short)80     /* number of speech bits                    */

void  prm2bits_ld8k(
  Word16 prm[],     /* (i)     : coder parameters                          */
  Word16 bits[]     /* (o)     : bit stream                                */
);
void  bits2prm_ld8k(
  Word16 bits[],    /* (i)     : bit stream                                */
  Word16 prm[]      /* (o)     : coder parameters                          */
);

/*-----------------------------------------------------------------------*
 * Prototypes for auxiliary functions.                                   *
 *-----------------------------------------------------------------------*/

void Copy(
  Word16 x[],       /* (i)     : input vector                              */
  Word16 y[],       /* (o)     : output vector                             */
  Word16 L          /* (i)     : vector length                             */
);

void Set_zero(
  Word16 x[],       /* (o)     : vector to clear                           */
  Word16 L          /* (i)     : length of vector                          */
);

Word16 Random(void);
```