UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

G.711

Apéndice I

(09/99)

SERIE G: SISTEMAS Y MEDIOS DE TRANSMISIÓN, SISTEMAS Y REDES DIGITALES

Sistemas de transmisión digital – Equipos terminales – Codificación de señales analógicas mediante modulación por impulsos codificados (MIC)

Modulación por impulsos codificados de frecuencias vocales

Apéndice I: Algoritmo de baja complejidad y alta calidad para el ocultamiento de pérdida de paquetes con la Recomendación G.711

Recomendación UIT-T G.711 – Apéndice I

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE G

SISTEMAS Y MEDIOS DE TRANSMISIÓN, SISTEMAS Y REDES DIGITALES

CONEXIONES Y CIRCUITOS TELEFÓNICOS INTERNACIONALES	G.100–G.199
SISTEMAS INTERNACIONALES ANALÓGICOS DE PORTADORAS	
CARACTERÍSTICAS GENERALES COMUNES A TODOS LOS SISTEMAS ANALÓGICOS DE PORTADORAS	G.200–G.299
CARACTERÍSTICAS INDIVIDUALES DE LOS SISTEMAS TELEFÓNICOS INTERNACIONALES DE PORTADORAS EN LÍNEAS METÁLICAS	G.300–G.399
CARACTERÍSTICAS GENERALES DE LOS SISTEMAS TELEFÓNICOS INTERNACIONALES EN RADIOENLACES O POR SATÉLITE E INTERCONEXIÓN CON LOS SISTEMAS EN LÍNEAS METÁLICAS	G.400–G.449
COORDINACIÓN DE LA RADIOTELEFONÍA Y LA TELEFONÍA EN LÍNEA	G.450–G.499
EQUIPOS DE PRUEBAS	
CARACTERÍSTICAS DE LOS MEDIOS DE TRANSMISIÓN	
SISTEMAS DE TRANSMISIÓN DIGITAL	
EQUIPOS TERMINALES	G.700–G.799
Generalidades	G.700–G.709
Codificación de señales analógicas mediante modulación por impulsos codificados (MIC)	**G.710–G.719**
Codificación de señales analógicas mediante métodos diferentes de la MIC	G.720–G.729
Características principales de los equipos múltiplex primarios	G.730–G.739
Características principales de los equipos múltiplex de segundo orden	G.740–G.749
Características principales de los equipos múltiplex de orden superior	G.750–G.759
Características principales de los transcodificadores y de los equipos de multiplicación de circuitos digitales	G.760–G.769
Características de operación, administración y mantenimiento de los equipos de transmisión	G.770–G.779
Características principales de los equipos múltiplex de la jerarquía digital síncrona	G.780–G.789
Otros equipos terminales	G.790–G.799
REDES DIGITALES	G.800–G.899
SECCIONES DIGITALES Y SISTEMAS DIGITALES DE LÍNEA	G.900–G.999

Para más información, véase la Lista de Recomendaciones del UIT-T.

RECOMENDACIÓN UIT-T G.711

MODULACIÓN POR IMPULSOS CODIFICADOS DE FRECUENCIAS VOCALES

APÉNDICE I

Algoritmo de baja complejidad y alta calidad para el ocultamiento de pérdida de paquetes con la Recomendación G.711

Resumen

Los algoritmos de ocultamiento de pérdida de paquetes (PLC), conocidos también como algoritmos de ocultamiento de borraduras de tramas, ocultan pérdidas de transmisión en un sistema audio cuando la señal de entrada es codificada y empaquetada en el transmisor, enviada por una red y recibida en el receptor que decodifica el paquete y reproduce la salida. Muchos de los codificadores vocales basados en la predicción lineal con excitación por código (CELP), tienen algoritmos PLC incorporados en sus normas. El algoritmo descrito en este apéndice proporciona un método para la Recomendación G.711.

Orígenes

El apéndice I a la Recomendación UIT-T G.711 ha sido preparado por la Comisión de Estudio 16 (1997-2000) del UIT-T y fue aprobado por el procedimiento de la Resolución N.° 1 de la CMNT el 30 de septiembre de 1999.

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución N.° 1 de la CMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión *empresa de explotación reconocida (EER)* designa a toda persona, compañía, empresa u organización gubernamental que explote un servicio de correspondencia pública. Los términos *Administración, EER* y *correspondencia pública* están definidos en la *Constitución de la UIT (Ginebra, 1992)*.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Página

Recomendación G.711

MODULACIÓN POR IMPULSOS CODIFICADOS DE FRECUENCIAS VOCALES

APÉNDICE I

Algoritmo de baja complejidad y alta calidad para el ocultamiento de pérdida de paquetes con la Recomendación G.711

(Ginebra, 1999)

I.1 Introducción

Los algoritmos de ocultamiento de pérdida de paquetes (PLC, *packet loss concealment*), conocidos también como algoritmos de ocultamiento de borraduras de tramas, ocultan pérdidas de transmisión en un sistema audio cuando la señal de entrada es codificada y empaquetada en el transmisor, enviada por una red y recibida en el receptor que decodifica el paquete y reproduce la salida. Muchos de los codificadores vocales basados en la predicción lineal con excitación por código (CELP, *code-excited linear-prediction*), tales como los de las Recomendaciones G.723.1 [1], G.728 [2], y G.729 [3], tienen algoritmos PLC incorporados en sus normas. El algoritmo descrito en este apéndice proporciona un método para la Recomendación G.711.

El objetivo de PLC es generar una señal vocal sintética para cubrir los datos omitidos (borraduras) en un tren de bits recibido. Idealmente, la señal sintetizada tendrá las mismas características de timbre y espectrales que la señal omitida, y no creará perturbaciones artificiales. Como a menudo las señales vocales son estacionarias localmente, es posible utilizar los datos históricos de las señales para generar una aproximación razonable del segmento que falta. Si las borraduras no son demasiado grandes, y no están en una región donde la señal cambia rápidamente, pueden ser inaudibles después del ocultamiento.

I.2 Descripción del algoritmo

Para añadir el algoritmo PLC a un sistema G.711 que normalmente no oculta las pérdidas, sólo hay que efectuar cambios en el receptor. Los datos de audio codificados según la Recomendación G.711 son muestreados a 8 kHz. En este apéndice se supone que los datos están divididos en tramas de 10 ms (80 muestras). Mediante el ajuste de algunos parámetros, es posible acomodar otras tamaños de paquetes o velocidades de muestreo.

I.2.1 Tramas intactas

Durante el funcionamiento normal (paquetes o tramas intactos), el receptor decodifica el paquete recibido y envía su salida al puerto de audio. Para soportar el algoritmo PCL, sólo hay que efectuar dos pequeños cambios en el receptor:

1) Se guarda una copia de la salida decodificada en una memoria tampón circular de datos históricos, cuya capacidad es de 48,75 ms (390 muestras). Esta memoria de datos históricos se utiliza para calcular el periodo de tono vigente y extraer formas de onda durante una borradura. Este almacenamiento en memoria tampón no introduce retardo en la señal de salida.

2) La salida es retardada 3,75 ms (30 muestras) antes de ser enviada al puerto de audio. Este retardo del algoritmo, utilizado para una adición con superposición (OLA, *overlap add*) al comienzo de la borradura, permite que el código PLC efectúe una transición uniforme entre la señal real y la sintetizada.

I.2.2 Primera trama no intacta

Al comenzar la borradura, la memoria tampón circular de datos históricos es copiada a una memoria, denominada la memoria tampón de tono, con la cual es más fácil trabajar. El contenido de la memoria tampón de tono se utiliza durante la borradura. Si ésta dura más de 10 ms, se hace una copia adicional de 1/4 del periodo de tono más reciente, que se denomina memoria tampón `lastq`.

I.2.3 Detección de tono

El periodo de tono se estima primero hallando la cresta de la correlación cruzada normalizada de los 20 ms más recientes de señales vocales en la memoria tampón de datos históricos con las señales vocales previas en derivaciones de 5 (40 muestras) a 15 ms (120 muestras). Esto corresponde a frecuencias de 200 a 66 Hz. La gama de tonos se ha elegido basada en una gama utilizadas en el posfiltro de la Recomendación G.728. Aunque la Recomendación G.728 utiliza un límite inferior de 2,5 ms (20 muestras), en este caso se aumenta a 40 muestras, de modo que el mismo periodo de tono no se repita más de dos veces en una trama borrada de 10 ms. Para reducir la complejidad, la estimación del tono se calcula en dos fases. En primer lugar, se efectúa una búsqueda en una señal diezmada de 2:1, y después se efectúa una búsqueda más afinada en la vecindad de la cresta de la búsqueda inicial. Es posible reducir la complejidad con una ligera degradación de la calidad omitiendo la búsqueda más afinada. En lo que sigue, el término *longitud de onda* se utiliza también para hacer referencia al valor de salida de este cálculo, pues la señal que falta puede contener voz o no.

A partir de la adición con superposición de desplazamiento de forma de onda (WSOLA, *waveform shift overlap add*), se sabe que la función de correlación cruzada normalizada puede ser sustituida con una correlación cruzada no normalizada, o una función de diferencia de magnitud media cruzada (AMDF, *average magnitude difference function*) y se obtendrán resultados globales similares.

I.2.4 Generación de señal sintética durante los primeros 10 ms

Durante los primeros 10 ms de la borradura, los mejores resultados se obtienen generando la señal sintetizada a partir del último periodo de tono sin atenuación. Sólo se utilizan los periodos de tono de 1,25 más recientes de la memoria tampón de tono durante los primeros 10 ms. Con el fin de asegurar una transición uniforme entre la señal real y la sintética, y se considera que la transición es uniforme si el periodo de tono es repetido múltiples veces, se aplica una OLA utilizando una ventana triangular en 1/4 del periodo de tono entre los periodos de tono último y penúltimo. Para 1/4 de longitud de onda, la señal que comienza en periodos de tono de 1,25 a partir del final de la memoria tampón de tono es multiplicada por una rampa de pendiente ascendente y se añade al último periodo de tono de 0,25 en la memoria tampón `lastq` multiplicada por una rampa de pendiente descendente. Si la complejidad no es un problema, las ventanas triangulares puede ser sustituidas con ventanas de Hamming en todas las operaciones de OLA.

El resultado de OLA sustituye la cola de la memoria tampón de tono y la cola de la memoria tampón de datos históricos. Es también la salida del receptor durante la cola de la última trama intacta, en sustitución de la señal original. Esto introduce el retardo del algoritmo, no se puede dar salida a la cola de la última trama hasta saber si la siguiente trama está borrada. Si se produce una borradura, la señal en la cola de la última trama intacta es modificada por OLA para asegurar una transición uniforme a la señal sintetizada.

La señal sintetizada durante los 10 ms que dura la borradura es generada colocando un puntero en un periodo de tono desde el final de la memoria tampón de tono y copiando las muestras a la salida. Si el periodo de tono es inferior a 10 ms, cuando el puntero imprime el final de la memoria tampón de tono, el punto es fijado de nuevo exactamente un periodo de ton antes de continuar. Si el periodo de tono es corto (la frecuencia es alta), el último periodo de tono en la memoria tampón de tono se repite múltiples veces durante la borradura de 10 ms.

Mientras progresa la borradura, la memoria tampón de datos históricos es actualizada con la salida sintetizada. De esta manera, la memoria tampón de datos históricos tiene siempre una señal continua y uniforme. Esta continuidad es importante si se produce la secuencia "una trama no intacta, una trama intacta, una trama no intacta".

I.2.5 Generación de señal sintética después de 10 ms

Si la trama siguiente es borrada también, la borradura durará por lo menos 20 ms, y se requiere ejecutar otra acción. Aunque la repetición de un periodo de tono funciona bien para borraduras cortas (por ejemplo, 10 ms), en borraduras largas introduce perturbaciones armónicas artificiales (sonidos agudos cortos). Esto es especialmente perceptible si la borradura se produce en una región sin voz de la señal vocal, o en una región de transición rápida, como una parada. Se ha descubierto gracias a la experimentación que estas perturbaciones se reducen considerablemente aumentando el número de periodos de tono utilizados para sintetizar la señal a medida que progresa la borradura. La reproducción de más periodos de tono aumenta la variación de la señal. Aunque los periodos de tono no son reproducidos en el orden en que se produjeron en la señal original, la salida resultante suena aún natural. En 10 ms en la borradura el número de periodos de tono utilizados para sintetizar la señal vocal es aumentado a dos, y en 20 ms se añade un tercer periodo de tono. Para borraduras superiores a 20 ms, no se efectúan modificaciones adicionales de la memoria tampón de tono.

Cuando aumenta el número de periodos de tono utilizados en la memoria tampón de tono, es importante que la transición en la señal sintetizada sea uniforme. Esto se logra continuando la salida de la memoria tampón de tono existente durante 1/4 de un periodo de tono en el comienzo de la segunda y tercera trama borradas, actualizando la memoria tampón de tono, manteniendo el puntero de la memoria tampón sincronizado con la fase intacta y aplicando OLA con la salida de la nueva la memoria tampón de tono.

La memoria tampón de tono es actualizada exactamente como durante la primera trama borrada, salvo que se aumenta el número de periodos de tono. Por ejemplo, en el comienzo de la segunda trama borrada, durante un 1/4 de longitud de onda, la señal que comienza en periodos de tono de 2,25 a partir del final de la memoria tampón de tono es multiplicada por una rampa de pendiente ascendiente y es añadida a la 1/4 de longitud de onda en la memoria `lastq` multiplicada por una rampa de pendiente descendente. El resultado de OLA sustituye el último 1/4 de longitud de onda en la memoria tampón de tono. Para mantener la fase del puntero de salida vigente, los periodos de tono son restados del puntero hasta que esté en el primer periodo de tono utilizado.

I.2.6 Atenuación

Al igual que otros algoritmos PLC, tales como los de las Recomendaciones G.729 y G.728 anexo I, cuando se producen borraduras largas es necesario atenuar la señal mientras que la borradura progresa. A medida que se alarga la borradura, es más probable que la señal sintetizada difiera de la señal real. Sin atenuación, la retención demasiado larga de ciertos tipos de sonidos crea extrañas perturbaciones, incluso si el segmento de señal sintetizada aislada suena natural. Durante los 10 primeros ms de una borradura, la señal no es atenuada. Al comenzar los segundos 10 ms, la señal sintetizada es atenuada linealmente con una rampa a razón de 20% por cada 10 ms. Después de 60 ms, la señal sintetizada es cero.

I.2.7 Primera trama intacta después de una borradura

En la primera trama intacta después de una borradura, se necesita una transición uniforme entre la señal vocal borrada sintetizada y la señal real. Para ello, la señal vocal sintetizada de la memoria tampón de tono es continuada más allá del fin de la borradura, y es mezclada con la señal real mediante OLA. La longitud de OLA depende del periodo de tono y de la longitud de la borradura. Para borraduras cortas, de 10 ms, se utiliza una ventana de 1/4 de longitud de onda. Para borraduras más largas, se aumenta la ventana en 4 ms por cada 10 ms de borradura, hasta un máximo del tamaño de trama, 10 ms.

I.2.8 Ejemplos

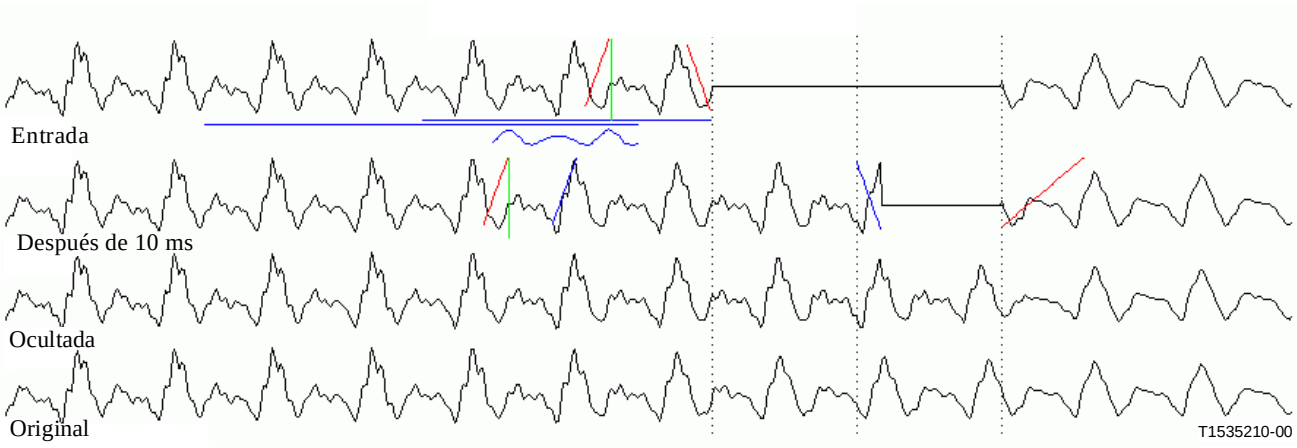


Figura I.1/G.711 – Algoritmo de ocultamiento de borraduras de tramas G.711

La figura I.1 muestra un ejemplo gráfico de cómo funciona el algoritmo con una borradura de 20 ms (es decir, dos tramas) en un segmento con voz de una señal vocal de un hablante masculino. La forma de onda de la parte de arriba muestra la entrada. La posición de la borradura está delimitada por las tres líneas verticales de puntos a intervalos de 10 ms que atraviesan todas las formas de onda. La señal vocal antes de la borradura en el gráfico "Entrada" es el contenido de la memoria tampón de datos históricos cuando comienza la borradura. La señal vocal después de la borradura es 20 ms de señal vocal no corrompida que llega después que ha terminado la borradura.

Por debajo de la señal "Entrada" hay dos líneas horizontales que representan la ventana utilizada para detección de tono con la correlación cruzada normalizada. La línea superior corta muestra los últimos 20 ms de señal vocal antes de la borradura y es la señal de referencia. La línea más baja representa la ventana de 20 ms que aparece en derivaciones desde 40 a 120 muestras. El pequeño gráfico debajo de las líneas horizontales es la salida de la correlación cruzada normalizada. La cresta de este gráfico mostrada por la línea vertical es la estimación de tono. Esta línea vertical está colocada un periodo de tono antes del comienzo de la borradura.

El contenido de 1/4 de longitud de onda antes de la borradura es guardado en la memoria tampón `lastq` para uso ulterior. Con el fin de crear la memoria tampón de tono, se aplica OLA con una ventana triangular al 1/4 de longitud de onda desde antes de la borradura hasta el 1/4 de longitud de onda del periodo de tono anterior. Las ponderaciones y las posiciones de las ventanas son mostradas por las dos rampas en la señal en "Entrada". Los resultados de esta OLA sustituyen el 1/4 de longitud de onda de la señal antes de la borradura. Durante los primeros diez minutos de la borradura, la señal sintética es generada repitiendo la memoria tampón de tono de un periodo, por ejemplo, la región que comienza en la línea vertical en la cresta de la correlación cruzada y termina en el comienzo de la borradura, tantas veces como sea necesario.

Los resultados de esto se muestran en el gráfico de la segunda forma de onda, "Después de 10 ms". En esta forma de onda, la señal sintética es ampliada 1/4 de longitud de onda más allá del límite de los primeros 10 ms de borradura, pues esto es necesario para la siguiente OLA. El desplazamiento en la memoria tampón de tono de un periodo al final de 10 ms se guarda en una variable denominada `poffset`. Si la borradura termina después de 10 ms, se aplicaría OLA a este 1/4 de longitud de onda con la señal de entrada y se termina el algoritmo de ocultamiento. En este caso la borradura es superior a 10 ms, de modo que la memoria tampón de tono es ampliada para aumentar la variación de la señal.

La variación de la señal es aumentada añadiendo otro periodo de tono a la memoria tampón de tono. Se ha dibujado una línea vertical en los dos periodos de tono de forma de onda antes del comienzo de la borradura. Se aplica OLA a este 1/4 de longitud de onda antes de esta línea, mostrada por la rampa de pendiente ascendente, con la memoria tampón `lastq` (la región debajo de la rampa de pendiente descendente en la forma de onda de "Entrada") y es colocada en la memoria tampón de tono. Durante los segundos 10 ms, la memoria tampón de tono es así la región entre el final de la primera rampa de pendiente ascendente y el comienzo de la borradura en la forma de onda en "Después de 10 ms". El 1/4 de longitud de onda a la derecha antes de la borradura en la memoria tampón de tono es diferente de la mostrada en la forma de onda, porque es el resultado de una OLA.

Con el fin de asegurar una transición uniforme entre las memorias tampón de tono de un periodo y de dos periodos se aplica OLA a 1/4 de longitud de onda en el comienzo de los segundos 10 ms en la borradura. La región por debajo de la rampa de pendiente descendente en los 10 ms en la borradura se combina con la región debajo de la rampa de pendiente ascendente cerca de la primera cresta de señal en la memoria tampón de tono de dos periodos con una OLA. El resultado coloca la región por debajo de la rampa de pendiente descendente. La posición de la rampa de pendiente ascendente se calcula a partir de puntero `poffset` guardado sustrayendo periodos de tono hasta que el puntero de tono esté en la primera longitud de onda de la porción de la memoria tampón de tono utilizada en ese momento. Esto asegura que se mantiene la fase de forma de onda apropiada a medida que aumenta el número de periodos en la memoria tampón de tono. Durante los segundos 10 ms de la trama borrada, la forma de onda sintética es generada copiando simplemente la señal de la memoria tampón de tono.

Al igual que durante los primeros 10 ms, la forma de onda sintética es ampliada más allá de final del límite de los siguientes 10 ms para una OLA con el segmento siguiente. Si la borradura continúa, basta un 1/4 de longitud de onda, y se aumenta de nuevo el número de periodos en la memoria tampón de tono. Si la borradura termina, se aumenta la longitud de ventana de OLA en 4 ms por cada 10 ms adicional de borradura (hasta un máximo de 10 ms) pues son muy probables discordancias de fase entre las señales sintética y la real. La ventana de OLA durante el fin de la borradura de 20 ms (1/4 de longitud de onda + 4 ms) se muestra como la rampa de pendiente ascendente al final de la borradura en la forma de onda de "Después de 10 ms". Durante los segundos 10 ms de una borradura, la forma de onda es atenuada también con una rampa línea. En el caso de borraduras largas, esta atenuación activa la señal sintética a 0 después de 60 ms.

La forma de onda "Ocultada" muestra la salida final de algoritmo. A efectos de comparación, se muestra también debajo la forma de onda sin borraduras. La señal vocal sintética se asemeja mucho a la señal vocal antes de la borradura, y es una aproximación adecuada de la forma de onda original. Una inspección detallada de la forma de onda sintética revela que el primer periodo de tono en la borradura viene del último periodo antes de la borradura, el segundo periodo viene de dos periodos antes de la borradura, y el tercer periodo repite el primer periodo antes de la borradura. Asimismo, debido a un cambio de tono durante la borradura, la cresta del último periodo en la señal sintética no está alineada exactamente con el último tono de la señal original. Es por esto que la ventana de OLA en la cola de la borradura debe ser ampliada a medida que la borradura se alarga.

I.3 Descripción del algoritmo con código C++ anotado

El algoritmo PLC ha sido realizado en coma flotante con una clase C++[1]. En esta subcláusula se presenta el código, junto con una explicación de cómo funciona. La implementación completa es aproximadamente 360 líneas de código C++, incluidos los comentarios. El código C++ contenido en este anexo se incluye sólo con fines ilustrativos.

---

[1] Alternativamente, se dispone de un código ANSI-C en el módulo G.711 de la Biblioteca de Herramientas de Soporte Lógico del UIT-T (Rec. UIT-T G.191).

I.3.1 Definiciones de tipo y constantes

Para poder conmutar entre las aritméticas de coma flotante de doble precisión y de una precisión, se define el siguiente tipo:

```
typedef float Float;
```

Para conmutar al modo doble precisión, cambiar el "float" (flotante) a un "double" (doble). El código define las siguientes constantes de preprocesador.

```
#define PITCH_MIN       40              /* tono permitido mínimo, 200 Hz */
#define PITCH_MAX       120             /* tono permitido máximo, 66 Hz */
#define PITCHDIFF       (PITCH_MAX - PITCH_MIN)
#define POVERLAPMAX     (PITCH_MAX >> 2)/* ventana OLA de tono máximo */
#define HISTORYLEN      (PITCH_MAX * 3 + POVERLAPMAX) /* memoria tampón de datos
                                                         históricos */
#define NDEC            2               /* 2:1 diezmo */
#define CORRLEN         160             /* 20 ms longitud de correlación */
#define CORRBUFLEN      (CORRLEN + PITCH_MAX) /* longitud de memoria tampón
                                                 de correlación */
#define CORRMINPOWER    ((Float)250.)   /* potencia mínima */
#define EOVERLAPINCR    32              /* fin OLA incremento por trama, 4 ms */
#define FRAMESZ         80              /* 10 ms en 8 KHz */
#define ATTENFAC((Float).2)     /* factor de atenuación por trama, 10 ms */
#define ATTENINCR       (ATTENFAC/FRAMESZ) /* atenuación por muestra */
```

I.3.2 Declaración de clase

```
 1 class LowcFE {
 2 public:
 3              LowcFE();
 4      void    dofe(short *s);         /* sintetiza señal vocal para
                                           borradura */
 5      void    addtohistory(short *s); /* añade una trama intacta a la
                                           memoria tampón de datos
                                           históricos */
 6 protected:
 7      int     erasecnt;               /* tramas borradas consecutiva */
 8      int     poverlap;               /* superposición basada en tono */
 9      int     poffset;                /* desplazamiento en periodo de
                                           tono */
10      int     pitch;                  /* estimación de tono */
11      int     pitchblen;              /* longitud de memoria tampón de
                                           tono vigente */
12      Float   *pitchbufend;           /* fin de memoria tampón de tono */
13      Float   *pitchbufstart;         /* comienzo de memoria tampón de
                                           tono */
14      Float   pitchbuf[HISTORYLEN];   /* memoria tampón  para ciclos de
                                           señal vocal */
15      Float   lastq[POVERLAPMAX];     /* último cuarto de longitud de
                                           onda guardado */
```

```
16      short   history[HISTORYLEN];    /* memoria tampón de datos históricos */
17
18      void    scalespeech(short *out);
19      void    getfespeech(short *out, int sz);
20      void    savespeech(short *s);
21      int     findpitch();
22      void    overlapadd(Float *l, Float *r, Float *o, int cnt);
23      void    overlapadd(short *l, short *r, short *o, int cnt);
24      void    overlapaddatend(short *s, short *f, int cnt);
25      void    convertsf(short *f, Float *t, int cnt);
26      void    convertfs(Float *f, short *t, int cnt);
27      void    copyf(Float *f, Float *t, int cnt);
28      void    copys(short *f, short *t, int cnt);
29      void    zeros(short *s, int cnt);
30 };
```

Esta clase se denomina `LowcFE`, para la ocultación de borraduras de trama de baja complejidad. Su interfaz es definida por tres funciones públicas. El constructor en la línea 3 inicializa las variables internas. La función `addtohistory()` en la línea 5 trata las tramas intactas. Su argumento, un puntero a un ordenamiento de abreviaturas de longitud FRAMESZ debe contener una trama de datos que ha sido decodificada. El código supone que una abreviatura contiene datos firmados de 16 bits y la salida del decodificador son datos MIC de 16 bits uniformes. La función `dofe()` genera la señal vocal sintética durante una borradura.

El resto de la clase está protegido, lo que significa que el usuario de la clase no puede acceder a sus miembros y funciones. La variable `erasecnt` sigue el número de tramas borradas consecutivas. Su valor comienza en 0 y es incrementada en 1 cada 10 ms durante una borradura y es reiniciada a 0 en la primera trama intacta después de una borradura.

La variable `poverlap` es la longitud de la ventana de OLA, y corresponde con 1/4 del tono vigente. La variable `poffset`, en la línea 9, es el desplazamiento en la memoria tampón de tono vigente y se utiliza para generar la forma de onda sintética. La variable `pitch`, en la línea 10, es la estimación de tono vigente. La variable `pitchblen`, es la longitud de la porción de la memoria tampón de tono que se está utilizando en ese momento. Esta longitud cambia si la borradura dura más de una trama. La variable `pitchbufend` indica el fin de la memoria tampón de tono. La variable `pitchbufstart` es un puntero al comienzo de la porción de la memoria tampón de tono utilizada en ese momento. La variable `pitchbuf`, en la línea 14, contiene la memoria tampón de datos históricos en el comienzo de una borradura. Además del último 1/4 de longitud de onda, esta memoria tampón permanece estática mientras dura la borradura. La memoria tampón `lastq` contiene una copia escondida del último 1/4 de longitud de onda de señal antes del comienzo de la borradura y se utiliza en las operaciones de OLA. La memoria tampón `history` , en la línea 16, contiene los datos históricos recientes de la señal. Es actualizada durante las tramas intactas y borradas.

Las funciones protegidas en las líneas 18 a 29 implementan el núcleo del algoritmo y son descritas ulteriormente.

I.3.3 Bucle principal

A continuación se muestra el principal bucle de procesamiento de un programa que utiliza la clase `LowcFE` con el sistema G.711. Se supone que la función `receiveframe()` devuelve verdadero si se ha recibido una trama de trenes de bits G.711 de 10 ms sin errores y falso si se produce una borradura. La función `g711dec()` convierte el tren de bits G.711 en salidas MIC uniformes de 16 bits y `output()` produce la salida de audio con las borraduras de trama ocultadas. La implementación de estas funciones no se describe porque está fuera del ámbito de este apéndice.

```
 1 void process()
 2 {
 3      char    bitstream[FRAMESZ];
 4      short   speech[FRAMESZ];
 5      LowcFE  fec;
 6      bool    frameisgood;
 7
 8      for(;;) {
 9              frameisgood = receiveframe(bitstream);
10              if (frameisgood) {
11                      g711dec(bitstream, speech);
12                      fec.addtohistory(speech);
13              } else
14                      fec.dofe(speech);
15              output(speech);
16      }
17 }
```

En la línea 10, se hace una prueba para ver si se ha borrado la trama de entrada. Si la trama está intacta, es enviada al decodificador en la línea 11, y la salida del decodificador se entrega al algoritmo PLC con la función del miembro `addtohistory()`. Si la trama está borrada, se invoca la función `dofe()` para genera la señal vocal sintética.

Cabe señalar que la función `addtohistory` en la línea 12 no hace más que copiar la trama de la señal vocal para la memoria tampón de datos históricos, pues modifica también el contenido de la matriz de señales vocales. La señal de salida es retardada por muestras de `POVERLAPMAX.` Además, en la primera trama intacta después de una borradura, se aplica OLA a la señal de entrada con la señal vocal sintetizada antes del retorno.

I.3.4 Funciones de miembros de servicio

Las funciones de servicio en las líneas 25 a 29 de la declaración de clase simplemente convierten abreviaturas en números reales `(convertsf)` y viceversa `(convertfs()),` copian matrices de números reales `(copyf())` y de abreviaturas `(copys())`, y anulan matrices de abreviaturas `(zeros())`. Se presentan primero porque son utilizadas por otros subprogramas (routines).

```
1 void LowcFE::convertsf(short *f, Float *t, int cnt)
2 {
3      for (int i = 0; i < cnt; i++)
4              t[i] = (Float)f[i];
5 }
```

```
 6
 7 void LowcFE::convertfs(Float *f, short *t, int cnt)
 8 {
 9      for (int i = 0; i < cnt; i++)
10              t[i] = (short)f[i];
11 }
12
13 void LowcFE::copyf(Float *f, Float *t, int cnt)
14 {
15      for (int i = 0; i < cnt; i++)
16              t[i] = f[i];
17 }
18
19 void LowcFE::copys(short *f, short *t, int cnt)
20 {
21      for (int i = 0; i < cnt; i++)
22              t[i] = f[i];
23 }
24
25 void LowcFE::zeros(short *s, int cnt)
26 {
27      for (int i = 0; i < cnt; i++)
28              s[i] = 0;
29 }
```

No hay saturación ni redondeo en el subprograma `convertfs.`

I.3.5 Constructor

El constructor inicializada los miembros internos de la clase.

```
1 LowcFE::LowcFE()
2 {
3      erasecnt = 0;
4      pitchbufend = &pitchbuf[HISTORYLEN];
5      zeros(history, HISTORYLEN);
6 }
```

En la línea 3, `erasecnt` se pone a 0, de modo que el código sabe que no está en una borradura en ese momento. Después, se fija `pitchbufend` para indicar el final de la memoria tampón de tono. A continuación, la memoria tampón de tono es vaciada, para que no se produzcan si hay una borradura en el comienzo de la señal.

I.3.6 Funciones addtohistory y savespeech

A continuación se presenta la función de interfaz pública `addtohistory()`, junto con una función protegida, `savespeech`, denominadas internamente por `addtohistory()`. `addtohistory()` es denominada por la aplicación después de la decodificación de una trama intacta, pero antes de la señal de salida.

```
/*
 * Guardar valor de tramas de la nueva señal vocal en la memoria tampón de
   datos históricos.
 * Devolver la señal vocal de salida retardada por POVERLAPMAX.
 */
void LowcFE::savespeech(short *s)
{
     /* acomodar la nueva señal */
     copys(&history[FRAMESZ], history, HISTORYLEN - FRAMESZ);
     /* copiar en la nueva trama */
     copys(s, &history[HISTORYLEN - FRAMESZ], FRAMESZ);
     /* copiar la trama retardada */
     copys(&history[HISTORYLEN - FRAMESZ - POVERLAPMAX], s, FRAMESZ);
}

/*
 *   Se recibió y decodificó una trama intacta.
 *   Si corresponde después de una borradura, aplicar una OLA con la señal
     sintética.
 *   Añadir la trama a la memoria tampón de datos históricos
 */
void LowcFE::addtohistory(short *s)
{
     if (erasecnt) {
             short overlapbuf[FRAMESZ];
             /*
              * borraduras más largas requieren solapes más largos
              * facilitar la transición entre las señales sintética
              * y real.
              */
             int olen = poverlap + (erasecnt - 1) * EOVERLAPINCR;
             if (olen > FRAMESZ)
                     olen = FRAMESZ;
             getfespeech(overlapbuf, olen);
             overlapaddatend(s, overlapbuf, olen);
             erasecnt = 0;
     }
```

```
36      savespeech(s);
37 }
```

En la línea 22, `addtohistory()`comprueba para ver si ésta es la primera trama intacta después de una borradura. Si la borradura se produjo en la trama anterior, `erasecnt` contendrá el número de tramas de 10 ms en la borradura. Si la trama anterior no fue borrada, `erasecnt` es 0.

Si la trama anterior fue borrada, las líneas 23-34 continúan la generación de la señal sintética, se aplica OLA a la señal sintética con la señal de entrada, y se libera `erasecnt`. La línea 29 determina la longitud de la ventana de OLA. La variable `poverlap` contiene el número de muestras en 1/4 del periodo de tono estimado vigente. Si la borradura es sólo 10 ms (`erasecnt == 1`), la longitud de la ventana de OLA se pone a `poverlap`. Si múltiples tramas fueron borradas, la longitud de la ventana de OLA es aumentada durante 4 ms por cada 10 ms de borradura. Las líneas 30-31 aseguran que la ventana de OLA no excede de 10 ms durante borraduras largas. La línea 32 genera la señal vocal sintética en la memoria tampón temporal, `overlapbuf`. En la línea 33 se aplica OLA a la señal sintética con la señal de entrada. La salida se coloca detrás en `s`, en sustitución de la señal de entrada original.

La línea 36 invoca `savespeech()` para guardar la señal en la memoria tampón de datos históricos y añadir el retardo del algoritmo. Si ésta no es la primera trama después de una borradura, se guarda la señal vocal. En los demás casos, es el resultado de OLA en la línea 33.

La función `savespeech()`, en las líneas 5 a 13, guarda la señal vocal en la memoria tampón de datos históricos. En el caso de procesamiento de señales digitales, ésta podría ser una memoria tampón circular, pero para simulación es más sencillo desplazar el contenido, La línea 8 efectúa un desplazamiento en la memoria tampón para acomodar la nueva trama, que es copiada en la cola de la memoria tampón en la línea 10, y la línea 12 sustituye el contenido de la matriz de entrada con la señal retardada por muestras `POVERLAPMAX` (30). Esto introduce el retardo del algoritmo de 3,75 ms.

I.3.7 Dofe

La función de miembro público `dofe` genera la señal sintética durante una borradura y contiene el grueso del algoritmo PLC.

```
 1 /*
 2  * Generar la señal sintética.
 3  * En el comienzo de una borradura, determinar el tono y extraer
 4  * un periodo de tono de la cola de la señal. Aplicar OLA durante 1/4
 5  * del tono para alisar la señal. Repetir después la señal extraída
 6  * mientras dura la borradura. Si la borradura continúa más de
 7  * 10 ms, aumentar el número de periodos en la pitchbuffer. Al final de
 8  * una borradura, aplicar OLA con el comienzo de la primera trama intacta.
 9  * La ganancia disminuye a medida que se alarga la borradura.
10  */
11 void LowcFE::dofe(short *out)
12 {
13      if (erasecnt == 0) {
14              convertsf(history, pitchbuf, HISTORYLEN); /* obtener datos
                históricos*/
15              pitch = findpitch();            /* hallar tono */
```

```
16              poverlap = pitch >> 2;     /* OLA 1/4 longitud de onda */
17              /* guardar últimas muestras poverlap originales */
18              copyf(pitchbufend - poverlap, lastq, poverlap);
19              poffset = 0;            /* crear memoria tampón  de tono
                                           con 1 periodo */
20              pitchblen = pitch;
21              pitchbufstart = pitchbufend - pitchblen;
22              overlapadd(lastq, pitchbufstart - poverlap,
23                      pitchbufend - poverlap, poverlap);
24              /* actualizar último 1/4 longitud de onda en memoria tampón
                   de datos históricos */
25              convertfs(pitchbufend - poverlap, &history[HISTORYLEN-poverlap],
26                      poverlap);
27              getfespeech(out, FRAMESZ);      /*  obtener señal sintetizada */
28      } else if (erasecnt == 1 || erasecnt == 2) {
29              /* estimación de cola de tono previo*/
30              short tmp[POVERLAPMAX];
31              int saveoffset = poffset;       /*  guardar desplazamiento
                                                    para OLA */
32              getfespeech(tmp, poverlap);     /*  continuar con antigua
                                                    pitchbuf */
34              poffset = saveoffset;
35              while (poffset > pitch)
36                      poffset -= pitch;
37              pitchblen += pitch;             /* añadir un periodo */
38              pitchbufstart = pitchbufend - pitchblen;
39              overlapadd(lastq, pitchbufstart - poverlap,
40                      pitchbufend - poverlap, poverlap);
41              /* OLA antigua pitchbuffer con nueva */
42              getfespeech(out, FRAMESZ);
43              overlapadd(tmp, out, out, poverlap);
44              scalespeech(out);
45      } else if (erasecnt > 5) {
46              zeros(out, FRAMESZ);
47      } else {
48              getfespeech(out, FRAMESZ);
49              scalespeech(out);
50      }
51      erasecnt++;
52      savespeech(out);
53 }
```

En la línea se prueba 13 `erasecnt` para ver si es 0. Si es verdadero, ésta es la primera trama de una borradura y se ejecuta el código en las líneas 14 a 27. La línea 14 copia el contenido de la memoria tampón de datos históricos en la memoria tampón de tono, convirtiéndolo a números reales en el proceso. Salvo para el último 1/4 de longitud de onda, el contenido de la memoria tampón de tono no cambia durante la borradura. La línea 15 invoca `findpitch` para ejecutar una correlación cruzada normalizada que estima el tono. La función `findpitch()` devuelve un valor entre `MIN_PITCH`(40) y `MAX_PITCH`(120). `findpitch()` domina la complejidad del algoritmo y sólo es invocada en la primera trama de una borradura. La línea 16 fija la longitud de ventana de OLA a 1/4 del periodo de tono. En la línea 18 el último 1/4 de longitud de onda de la memoria tampón de tono es guardado en `lastq`, en el caso de que la borradura dure más de una trama. Las líneas 19-21 establecen después la memoria tampón de tono de modo que sólo se utiliza el último periodo en la memoria tampón para generar la señal vocal sintética. En la línea 22, se aplica OLA al contenido de `lastq` con el 1/4 de periodo que comienza 1,25 periodos antes en la memoria tampón de tono. Esto asegura una transición uniforme entre la señal vocal original y la sintética en el comienzo de una borradura, y también una transición uniforme si la memoria tampón de tono de un solo periodo es repetida durante la primera trama, por ejemplo, el periodo es inferior a 80 muestras (10 ms).

El resultado de esta OLA se coloca en la cola de la memoria tampón de tono, y sustituye el último 1/4 de periodo en la memoria tampón de datos históricos en la línea 25. La actualización de la memoria tampón de datos históricos asegura la salida de la señal vocal con OLA cuando se invoca `savespeech` en la línea 52 y que la memoria tampón de datos históricos no contenga discontinuidades. La línea 27 genera después la señal vocal sintética para la trama llamando `getfespeech()`. La función `getfespeech()` copia simplemente el último periodo en la memoria tampón de tono en la matriz de salida, repitiendo el periodo cuantas veces sea necesario para rellenar las 80 muestras. Después de la línea 27, el código salta a la línea 51, que incrementa `erasecnt.` La línea 52 actualiza la memoria tampón de datos históricos con la señal vocal sintética. La función `savespeech()` retarda también la señal, de modo que cuando `savespeech()` devuelve el resultado de OLA en la línea 22 estará presente en los primeros 3,75 ms de salida.

En las tramas segunda y tercera de borradura se toma la derivación en las líneas 29-44. Esta rama aumenta el número de periodos en la memoria tampón de tono utilizados para sintetizar el tono. El aumento del número de periodos de tono aumenta la variación de la señal, que a su vez disminuye el número de perturbaciones armónicas artificiales (sonidos agudos cortos) que se producen si sólo se utiliza un periodo de tono. En las líneas 30-32 se continúa la salida de la memoria tampón de tono utilizada en la trama borrada anterior durante 1/4 periodo y se coloca en una memoria tampón temporal. Se aplica OLA a esta señal con la nueva memoria tampón de periodo de tono para asegurar una transición uniforme en la señal de salida a medida que aumenta el número de periodos de tono en la memoria tampón de tono.

En la línea 31 el desplazamiento en la memoria tampón de periodo de tono se guarda en `saveoffset.` Después la antigua memoria tampón de tono genera la forma de onda en la línea 32, se restablece `poffset` a `saveoffset` en la línea 34. Esto asegura que se mantiene la fase de la señal sintética. Las líneas 35-36 substraen periodos de tono de `poffset` hasta que indica el primer periodo.

La línea 37 añade un periodo a la memoria tampón de tono. La línea 38 actualiza `pitchbufstart,` el puntero al comienzo de la porción utilizada de la memoria tampón de tono. Después la línea 39 aplica OLA entre los datos guardados en `lastq` y el 1/4 de periodo de 1/4 antes de `pitchbufstart`, colocando los resultados en el último 1/4 de periodo en la cola de la memoria tampón de tono. Como en la primera trama, esto asegura que la memoria tampón de tono es uniforme si se repite múltiples veces en la señal de salida.

En la línea 42 se extrae la señal sintética de la nueva memoria tampón de tono mientras dura la trama, y se aplica OLA al 1/4 de periodo al comienzo de estos datos, creando la memoria tampón temporal en la línea 32. En la línea 44, la señal sintética es atenuada con una rampa lineal mediante una invocación a `scalespeech`. Esta atenuación se efectúa a razón de 20% por cada 10 ms y comienza en el principio de la segunda trama borrada. Obsérvese que la señal sintética no es atenuada durante la primera trama de una borradura.

Durante las tramas cuarta, quinta y sexta de una borradura, el procesamiento es más sencillo, porque la memoria tampón de tono permanece estática, como se muestra en las líneas 48-49. La señal sintética es generada invocando `getfespeech()`, y después atenuada con `scalespeech()`. Después de 60 ms, la señal sintética se pone a 0 en la línea 46.

I.3.8 Detección de tono

El detector de tono es la única parte del algoritmo que tiene una complejidad considerable. Para reducirla, se efectúa primero una búsqueda ordinaria, en una señal diezmada. Después se efectúa una búsqueda más afinada cerca de la cresta de la búsqueda inicial. Se aplica la correlación cruzada a los últimos 20 ms de la señal retardando la señal anterior de PITCH_MIN a PITCH_MAX.

```
 1 /*
 2  * Estimar el tono.
 3  * l - puntero a la primera muestra en los últimos 20 ms de la señal.
 4  * r - indica la muestra PITCH_MAX antes de l
 5  */
 6 int LowcFE::findpitch()
 7 {
 8      int     i, j, k;
 9      int     bestmatch;
10      Float   bestcorr;
11      Float   corr;           /* correlación */
12      Float   energy;         /* energía continua */
13      Float   scale;          /* correlación de escala por potencia media */
14      Float   *rp;            /* segmento para concordar */
15      Float   *l = pitchbufend - CORRLEN;
16      Float   *r = pitchbufend - CORRBUFLEN;
17
18      /* búsqueda inicial*/
19      rp = r;
20      energy = 0.f;
21      corr = 0.f;
22      for (i = 0; i < CORRLEN; i += NDEC) {
23              energy += rp[i] * rp[i];
24              corr += rp[i] * l[i];
25      }
26      scale = energy;
27      if (scale < CORRMINPOWER)
28              scale = CORRMINPOWER;
```

```
29      corr = corr / (Float)sqrt(scale);
30      bestcorr = corr;
31      bestmatch = 0;
32      for (j = NDEC; j <= PITCHDIFF; j += NDEC) {
33              energy -= rp[0] * rp[0];
34              energy += rp[CORRLEN] * rp[CORRLEN];
35              rp += NDEC;
36              corr = 0.f;
37              for (i = 0; i < CORRLEN; i += NDEC)
38                      corr += rp[i] * l[i];
39              scale = energy;
40              if (scale < CORRMINPOWER)
41                      scale = CORRMINPOWER;
42              corr /= (Float)sqrt(scale);
43              if (corr >= bestcorr) {
44                      bestcorr = corr;
45                      bestmatch = j;
46              }
47      }
48      /* búsqueda afinada */
49      j = bestmatch - (NDEC - 1);
50      if (j < 0)
51              j = 0;
52      k = bestmatch + (NDEC - 1);
53      if (k > PITCHDIFF)
54              k = PITCHDIFF;
55      rp = &r[j];
56      energy = 0.f;
57      corr = 0.f;
58      for (i = 0; i < CORRLEN; i++) {
59              energy += rp[i] * rp[i];
60              corr += rp[i] * l[i];
61      }
62      scale = energy;
63      if (scale < CORRMINPOWER)
64              scale = CORRMINPOWER;
65      corr = corr / (Float)sqrt(scale);
66      bestcorr = corr;
67      bestmatch = j;
68      for (j++; j <= k; j++) {
69              energy -= rp[0] * rp[0];
70              energy += rp[CORRLEN] * rp[CORRLEN];
```

```
71              rp++;
72              corr = 0.f;
73              for (i = 0; i < CORRLEN; i++)
74                      corr += rp[i] * l[i];
75              scale = energy;
76              if (scale < CORRMINPOWER)
77                      scale = CORRMINPOWER;
78              corr = corr / (Float)sqrt(scale);
79              if (corr > bestcorr) {
80                      bestcorr = corr;
81                      bestmatch = j;
82              }
83      }
84      return PITCH_MAX - bestmatch;
85 }
```

Cuando se invoca `findpitch()` el contenido de la memoria tampón de tono concuerda con el contenido de la memoria tampón de datos históricos. La línea 15 fija la señal de referencia, l, a los 20 ms de muestra antes del comienzo de la borradura. La línea 16 fija la señal de retardo a muestras de `MAX_PITCH` antes de eso. Las líneas 19-29 calculan la correlación cruzada normalizada en `MAX_PITCH` en una señal diezmada de 2:1. Las líneas 26-28 fijan la energía a un nivel mínimo, para evitar una división por cero y desacentuar regiones con energía muy baja.

Las líneas 32-47 repiten el cálculo de correlación cruzada normalizada para cada otro retardo. Sólo los retardos pares son examinados. Se mantiene una suma continua de la energía, de modo que sólo se requieren dos acciones multiplicar-acumular (MAC, *multiply accumulator*) para actualizarla por iteración. La cresta de la correlación se mantiene en `bestcorr,` y el retardo correspondiente se mantiene en `bestmatch.`

Las líneas 48-54 calculan la región utilizada para la búsqueda afinada. Si la cresta bruta no está en el retardo mínimo o máximo, se buscan 3 retardos en la búsqueda afinada. Las líneas 55-83 repiten los cálculos efectuados en la búsqueda inicial, pero sin diezmo. La cresta de la búsqueda afinada es devuelta como la estimación de tono en la línea 84.

I.3.9 Generación y atenuación de la señal sintética

La función `getfespeech()` extrae la forma de onda sintetizada de la memoria tampón de tono, mientras que `scalespeech` aplica la pendiente de atenuación a la señal vocal sintetizada.

```
 1 /*
 2  * Obtener muestras de la memoria tampón de tono circular. Actualizar
 3  * poffset para que cuando se borren las tramas subsiguientes la señal
     continúe.
 4  */
 5 void LowcFE::getfespeech(short *out, int sz)
 6 {
 7      while (sz) {
 8              int cnt = pitchblen - poffset;
 9              if (cnt > sz)
```

```
10                      cnt = sz;
11              convertfs(&pitchbufstart[poffset], out, cnt);
12              poffset += cnt;
13              if (poffset == pitchblen)
14                      poffset = 0;
15              out += cnt;
16              sz -= cnt;
17      }
18 }
19
20 void LowcFE::scalespeech(short *out)
21 {
22      Float g = (Float)1. - (erasecnt - 1) * ATTENFAC;
23      for (int i = 0; i < FRAMESZ; i++) {
24              out[i] = (short)(out[i] * g);
25              g -= ATTENINCR;
26      }
27 }
```

`getfespeech()` en las líneas 5-18 no hace más que copiar la forma de onda de la memoria tampón de tono en la matriz de salida. Si el tamaño solicitado es mayor que la memoria tampón de tono, el puntero de salida `poffset` vuelve al comienzo de la memoria tampón de tono y continúa a partir de ahí. La memoria tampón de tono comienza en `pitchbufstart` y las muestras tienen la longitud de `pitchblen`. La variable `poffset` es la posición vigente en la memoria. `poffset` es actualizada en cada iteración a través del bucle y en los puntos de retorno a la siguiente muestra que debe salir. Esta generación de señal sintética puede continuar si se invoca de nuevo `getfespeech()`.

La función `scalespeech()` en las líneas 20-27 aplica la pendiente de atenuación a un valor de trama de señal vocal sintética. No es invocada durante la primera trama de una borradura. En la segunda trama, `erasecnt` será 1 porque no ha sido incrementada aún, de modo que la ganancia comenzará en 1 y disminuirá hasta 0,8. La atenuación continúa a razón del 20% por cada trama para las siguientes tramas borradas.

I.3.10 Operadores de adición con superposición

Los operadores de adición con superposición completan el código fuente. Se proporcionan tres subprogramas. Dos de ellos son idénticos, salvo sus tipos de argumento y tienen el mismo nombre de función de miembro, `overlapadd.` Se invoca la versión Float para las OLA en la memoria tampón de tono, mientras que versión abreviada se invoca para las OLA en la señal de salida.

```
1 /*
2  * lados izquierdo y derecho de OLA
3  */
4 void LowcFE::overlapadd(Float *l, Float *r, Float *o, int cnt)
5 {
6      Float incr = (Float)1. / cnt;
7      Float lw = (Float)1. - incr;
```

```
 8      Float rw = incr;
 9      for (int i = 0; i < cnt; i++) {
10              Float t = lw * l[i] + rw * r[i];
11              if (t > 32767.)
12                      t = 32767.;
13              else if (t < -32768.)
14                      t = -32768.;
15              o[i] = t;
16              lw -= incr;
17              rw += incr;
18      }
19 }
20
21 void LowcFE::overlapadd(short *l, short *r, short *o, int cnt)
22 {
23      Float incr = (Float)1. / cnt;
24      Float lw = (Float)1. - incr;
25      Float rw = incr;
26      for (int i = 0; i < cnt; i++) {
27              Float t = lw * l[i] + rw * r[i];
28              if (t > 32767.)
29                      t = 32767.;
30              else if (t < -32768.)
31                      t = -32768.;
32              o[i] = (short)t;
33              lw -= incr;
34              rw += incr;
35      }
36 }
```

OLA utiliza ventanas triangulares que son calculadas en el subprograma sobre la base de la longitud del argumento `cnt`. La línea 6 calcula el incremento de ventana por muestra, mientras que las líneas 7 y 8 inicializan las ponderaciones de ventana de los lados izquierdo y derecho. Las líneas 10-17 aplican las ponderaciones a cada muestra, saturan el valor a un entero de 16 bits, proporcionan la salida del resultado y actualizan después las ponderaciones para la siguiente iteración.

Se invoca un subprograma OLA adicional, `overlapaddatend()`en la primera trama intacta después de una borradura. Este subprograma difiere de los anteriores en la escala aplicada a la señal vocal sintética por el factor de atenuación antes de combinarla con la señal de entrada.

```
 1  */
 2  * OLA al final de la borradura con el comienzo de la primera trama intacta
 3  * Escalar la señal vocal sintética mediante el factor de ganancia antes de
      OLA.
```

```
 4  */
 5 void LowcFE::overlapaddatend(short *s, short *f, int cnt)
 6 {
 7      Float incr = (Float)1. / cnt;
 8      Float gain = (Float)1. - (erasecnt - 1) * ATTENFAC;
 9      if (gain < 0.)
10              gain = (Float)0.;
11      Float incrg = incr * gain;
12      Float lw = ((Float)1. - incr) * gain;
13      Float rw = incr;
14      for (int i = 0; i < cnt; i++) {
15              Float t = lw * f[i] + rw * s[i];
16              if (t > 32767.)
17                      t = 32767.;
18              else if (t < -32768.)
19                      t = -32768.;
20              s[i] = (short)t;
21              lw -= incrg;
22              rw += incr;
23      }
24 }
```

I.4 Complejidad y retardo

Se considera que la complejidad del algoritmo tiene una cresta de aproximadamente 1/2 de MIP del procesamiento de la señal digital. El promedio es mucho más bajo. La complejidad está dominada por el cálculo del término de correlación cruzada en el subprograma de detección de tono. Este cálculo sólo se efectúa durante la primera trama de una borradura. Para todas las demás tramas la complejidad es muy baja, del orden de algunas instrucciones MAC por muestra.

La complejidad se estima como sigue. En la primera trama borrada, el algoritmo debe estimar el tono y aplicar una OLA durante 1/4 de un periodo de tono. El valor máximo del tono es 120 muestras, de modo que 1/4 es 30. Los subprogramas `findpitch()` y `overlapadd()` tienen los siguientes cómputos de operadores:

Subprograma	MAC	comparar	div	sqrt
`findpitch`	3764	86	44	44
`overlapadd`	121	0	1	0
Total	3885	86	45	44

Suponiendo 2 ciclos para una comparación y 10 para una división o sqrt, esto da 3885 + 86 * 2 + (45 + 44) * 10 = 4947 ciclos. Como esto se produce en una trama de 10 ms, se multiplica por 100 para obtener 0,5 MIPS.

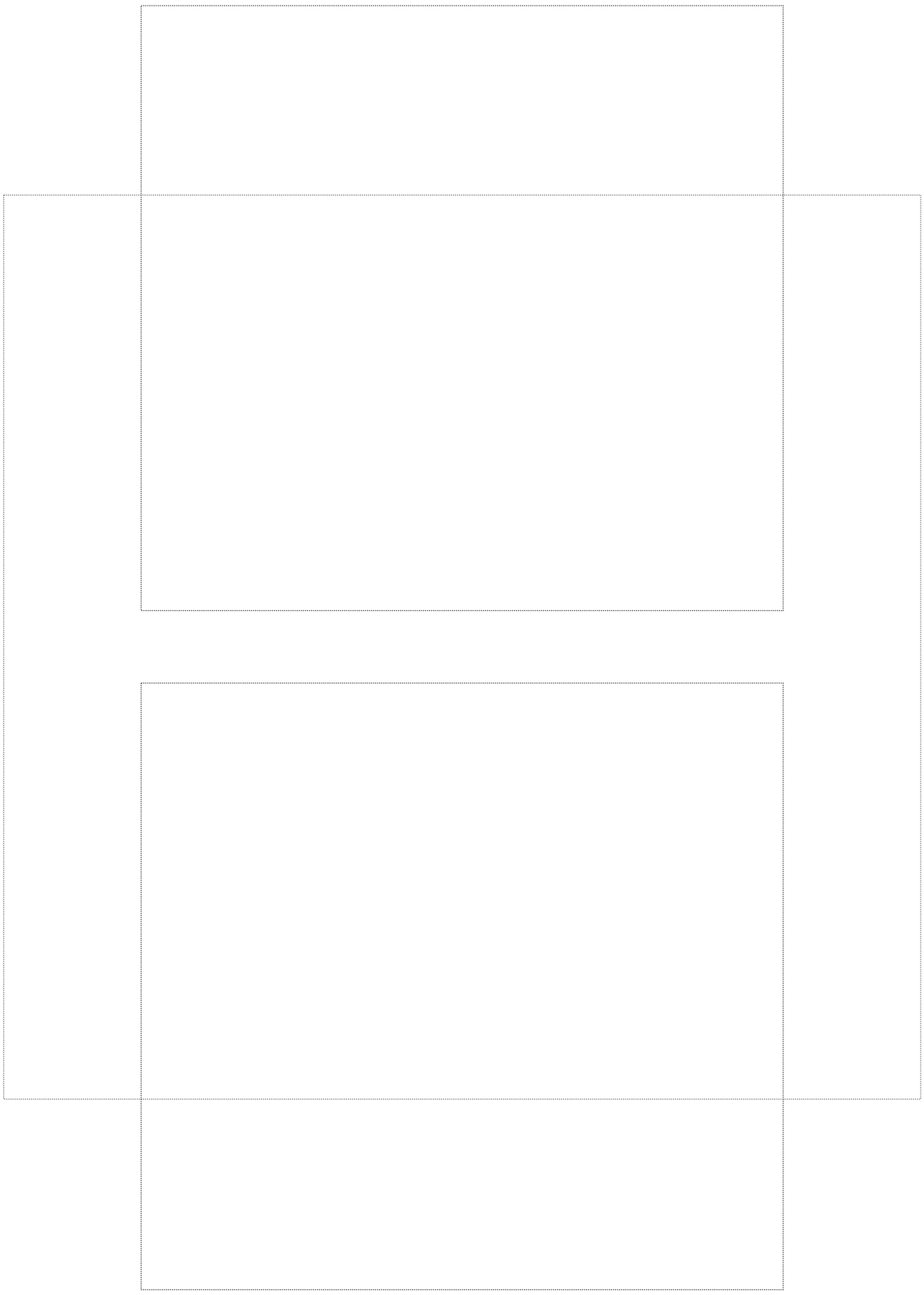
Como se ha indicado anteriormente, el algoritmo tiene un retardo de 3,75 ms. Con el fin de minimizar el retardo del cálculo, el algoritmo PLC puede ser ejecutado, con un pequeño costo de memoria, después de cada trama intacta, antes de saber si la próxima trama será borrada. Si la siguiente trama es borrada, la señal sintética está disponible inmediatamente, y si no es borrada, basta con descartar la señal sintética.

Bibliografía

[1] Recomendación UIT-T G.723.1 (1996), *Codificadores vocales: Codificador de voz de doble velocidad para la transmisión en comunicaciones multimedios a 5,3 y 6,3 kbit/s*.

[2] Recomendación CCITT G.728 (1992), *Codificación de señales vocales a 16 kbits/s utilizando predicción lineal con excitación por código de bajo retardo*.

[3] Recomendación UIT-T G.729 (1996), *Codificación de la voz a 8 kbit/s mediante predicción lineal con excitación por código algebraico de estructura conjugada (CS-ACELP).*



SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	**Sistemas y medios de transmisión, sistemas y redes digitales**
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos y comunicación entre sistemas abiertos
Serie Y	Infraestructura mundial de la información y aspectos protocolo Internet
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Impreso en Suiza
Ginebra, 2000