Proyecto Fin de Carrera

Desarrollo de un sistema de seguimiento de pitch y detección de caras en secuencias de vídeo de bajo coste computacional para su aplicación en herramientas de asistencia a la logopedia

Autor

Gabriel García García

Director

Antonio Miguel Artiaga

Escuela de Ingeniería y Arquitectura

2014

RESUMEN

Desarrollo de un sistema de seguimiento de pitch y detección de caras en secuencias de vídeo de bajo coste computacional para su aplicación en herramientas de asistencia a la logopedia

Con el avance tecnológico de las últimas décadas, los procesadores se han hecho cada vez más rápidos y baratos, permitiendo incorporar una mayor capacidad de cálculo tanto en ordenadores como, durante los últimos años, en dispositivos móviles. Gracias a ello, las "tecnologías del habla" han dado un gran salto, permitiendo no solo disponer de herramientas más precisas para ayudar a las personas con patologías del lenguaje, sino poder hacer uso de ellas desde cualquier dispositivo. Una de las aplicaciones de ayuda a la logopedia con mayor relevancia es 'PreLingua', desarrollada en el Grupo de Tecnología de las Comunicaciones en el Departamento de Ingeniería Electrónica y Comunicaciones de la Universidad de Zaragoza.

En este proyecto de fin de carrera se proponen dos objetivos: la mejora de la estimación de pitch para su aplicación en un conjunto de herramientas logopédicas, y la detección de caras a partir de la secuencia de vídeo capturada por un medio de bajo coste como una webcam, todo ello con un coste computacional mínimo, de manera que estas tareas se puedan ejecutar sin esfuerzo en cualquier equipo.

El primer objetivo consiste en realizar una mejora del sistema de detección de pitch, añadiéndole la capacidad de realizar el seguimiento de la frecuencia de pitch mediante un algoritmo de programación dinámica, proporcionando así robustez frente a transiciones bruscas (principalmente a los armónicos, frecuencias con las que se confunde con mayor probabilidad por poseer mayor energía); el segundo objetivo consiste en añadir una funcionalidad de detección de caras. Para lograr un sistema de bajo coste computacional y baja latencia, el seguimiento de pitch se realizará con una cantidad de información limitada en memoria, mientras que la detección de caras hará uso de un modelo simple de mezcla de Gaussianas entrenada para responder a los píxeles de piel, reduciendo de esta forma el coste computacional frente a otros algoritmos convencionales de detección de caras (como Viola-Jones).

El objetivo de mejorar y dotar a las herramientas de estos sistemas de análisis es favorecer y mejorar la realimentación que obtiene el usuario, que es fundamental para un correcto aprendizaje y progreso. Además, gracias a la fiabilidad mejorada, el logopeda es capaz de mejorar sus decisiones y adaptar mejor las actividades y el esfuerzo del usuario.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

Las “Tecnologías del Habla” son un conjunto de herramientas diseñadas para tratar los datos relativos a la señal de voz que pueden ser de gran ayuda para las personas que presenten alguna patología del lenguaje.

Desde su aparición, estas tecnologías han ido evolucionando lentamente, debido principalmente a que, a pesar de que existiese la tecnología para poder utilizarlas, el usuario no solía tener acceso a unas herramientas adecuadas que le permitieran emplearlas, como un micrófono para capturar la señal de voz digitalizada con precisión o un ordenador que pudiera tratar esos datos de manera ágil.

Gracias al gran avance de la tecnología durante los últimos años, las “Tecnologías del Habla” han dado un gran salto, al surgir tanto una gran variedad de dispositivos para capturar la señal de voz como gran cantidad de procesadores más rápidos y baratos que permiten incorporar una mayor capacidad de cálculo.

El uso de las “Tecnologías del Habla” en el campo de la logopedia se basa en el desarrollo de herramientas que permitan a los logopedas desempeñar mejor sus tareas, sean aplicaciones que ayuden al usuario a comprender mejor las tareas que debe llevar a cabo (por ejemplo, mostrándole como debe de situar la lengua para pronunciar una cierta vocal), o herramientas que permitan obtener unos resultados más fiables.

Sin embargo, los profesionales de la logopedia se han encontrado tradicionalmente con serios problemas, dado el coste de estas herramientas, además de estar diseñadas para idiomas distintos al español.

Este proyecto está enmarcado dentro del Grupo ViVoLab siguiendo la línea de proyectos realizados con anterioridad en este mismo grupo[1].

1.1 Estado del arte

Ante la elevada demanda por parte de los logopedas de un software específico que les permitiese ayudar a personas con patologías del habla, el Grupo ViVoLab, en colaboración con el Colegio Público de Educación Especial Alborada, creó el proyecto "Comunica"[2] enfocado al desarrollo de herramientas libres de ayuda a la logopedia.

Dentro de este proyecto, la aplicación "PreLingua"[2] se encarga del tratamiento del pre-lenguaje, es decir, de las habilidades anteriores al habla en sí, tales como la entonación, el volumen, etc. Esta aplicación se basa en una interfaz gráfica que permite al usuario realizar una serie de ejercicios que transforman uno o varios parámetros de la señal de voz en alguna clase de realimentación, obteniendo así una forma más amena (y, sobre todo, eficaz, al presentar al usuario la realimentación de cómo se están llevando a cabo esos ejercicios) de poder trabajar esta clase de problemas, así como proporcionando al logopeda una estimación precisa de estos parámetros.

Una de las herramientas de las que hace uso la aplicación "PreLingua" es un estimador de la frecuencia de pitch de la señal de voz. En este proyecto se pretende mejorar dicha herramienta, a la par que proporcionar otra herramienta que permita realizar detección de caras.

1.2 Objetivos

El objetivo de este proyecto es realizar una mejora de la aplicación "PreLingua" [3], [8]. Para ello, se van a desarrollar dos herramientas:

- Por un lado, una mejora del algoritmo de estimación de la frecuencia de pitch que proporcione robustez frente a transiciones bruscas, principalmente a los armónicos, ya que, por ser las frecuencias con mayor energía, será con las que mayor probabilidad se confunda.
- Por otro lado, una nueva herramienta de detección facial, que permita localizar la posición en la que se encuentra la cara del usuario.

1. http://vivolab.es
2. http:/dihana.cps.unizar.es/~alborada/index.html

Para poder aprovechar todavía más las posibilidades de esta aplicación, estas herramientas van a ser desarrolladas de manera que tengan un coste computacional reducido, de forma que puedan ser utilizadas sin necesidad de disponer de un equipo potente. Gracias a ello conseguiremos no solo que los profesionales de la logopedia obtengan una realimentación mucho mejor de la actividad del usuario (al poder disponer de mejores herramientas con las que obtener los resultados), sino que los usuarios puedan ser capaces de utilizar estas herramientas desde su propio equipo (ordenador o incluso móvil) y puedan practicar más cómodamente los ejercicios propuestos por el logopeda en las sesiones.

1.3 Tareas Realizadas

En este apartado se recogen las tareas que han sido necesarias para llevar a cabo este proyecto.

1.3.1 Documentación

En primer lugar, se llevó a cabo una labor de documentación dedicada a conocer el estado del arte correspondiente a las dos herramientas a desarrollar ("estimación robusta de la frecuencia de pitch" y "detección facial"). Para ello, fue necesario consultar y comprender los trabajos realizados en Proyectos Fin de Carrera, Tesis Doctorales y artículos de investigación que permitiesen comprender como enfocar el trabajo a realizar de la manera más eficaz posible.

1.3.2 Tratamiento de la señal de voz

Con la información recopilada, la primera tarea del proyecto fue llevar a cabo la adquisición y el tratamiento previo de la señal de voz. Para ello, se realizó la captura de la señal de voz mediante un micrófono, de forma que se grabasen ficheros de audio a partir de los que obtener la señal de voz en MATLAB, donde se realizaría el tratamiento previo. Esta etapa consistió principalmente en la familiarización con las técnicas de procesado de voz utilizadas, así como su implementación en MATLAB.

1.3.3 Tratamiento de la imagen

Análogamente a la etapa anterior, se llevó a cabo el mismo proceso para la adquisición y el tratamiento previo de la imagen: se realizó la captura de una secuencia de vídeo

mediante una Webcam de la que poder obtener los datos de imagen en MATLAB; esta etapa consistió en la familiarización con las técnicas de tratamiento de imagen utilizadas y su implementación en MATLAB.

1.3.4 Desarrollo del algoritmo dinámico de seguimiento de pitch

Una vez realizado el tratamiento de la señal de voz, se desarrolló el algoritmo dinámico de seguimiento de pitch, que permitiría realizar un seguimiento del pitch reduciendo la posibilidad de confundir la frecuencia de pitch con los armónicos.

1.3.5 Detección de caras

Del mismo modo, una vez realizado el tratamiento de la imagen, se desarrolló el algoritmo de detección facial, para lo que fue necesario la adquisición de imágenes correspondientes a caras (en concreto, de píxeles de piel) para poder realizar el entrenamiento de un sistema que respondiese a estos píxeles.

1.4 Tecnologías empleadas

Para este proyecto se ha utilizado MATLAB para el desarrollo de las herramientas debido a su excelente rendimiento con operaciones vectoriales y matriciales, de manera que se puedan tratar cómodamente los datos tanto de la señal de voz como la imagen (o secuencia de vídeo).

2. Estimación robusta del pitch

En este apartado se va a detallar el trabajo realizado para el desarrollo de un algoritmo de estimación robusta de la frecuencia de pitch de la señal de voz.

2.1 Tratamiento previo de la señal de voz

2.1.1 La señal de voz

La voz es la onda de presión generada por el tracto vocal. Esta onda sonora es el resultado de la vibración del aire que produce la excitación (el pulso glotal, que es donde se origina el sonido) que atraviesa todo el sistema fonador.

El tracto vocal puede aproximarse, desde la glotis (el origen de la excitación de la voz) hasta la boca y la nariz (la salida al exterior de la señal), como una serie de tubos. De esta forma, puede ser modelado de una forma sencilla como un filtro todo polos que dependerá de la longitud de cada uno de esos tubos (es decir, dependiente de la fisiología de cada persona).

En general, en cualquier palabra o frase, la señal de voz no es estacionaria (periódica), ya que contiene toda la información que se quiere transmitir, por lo que tiene que ir variando sus características (si no variase, no se podría transmitir esa información). Para poder obtener la información útil que contiene, deberá realizarse un análisis localizado de la señal (es decir, por ventanas, donde la señal si será estacionaria).

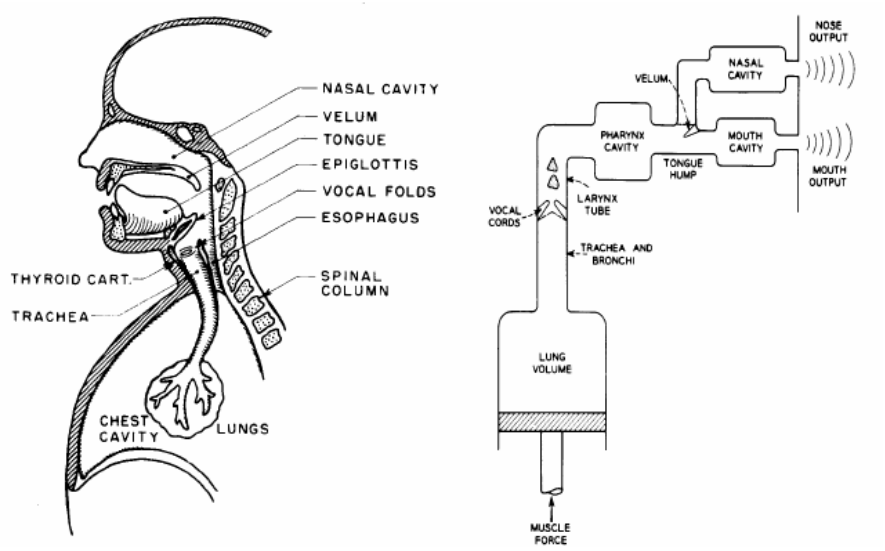

Figura 2.1. Tracto vocal y modelo de tubos

2.1.2 Normalización

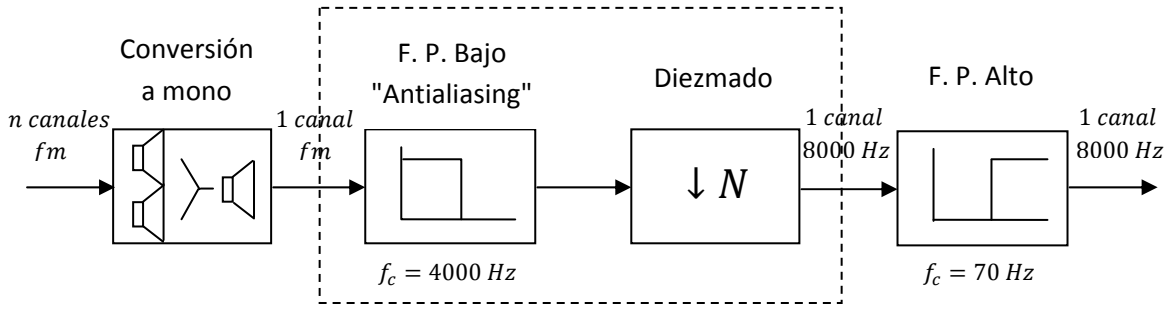

Figura 2.2. Diagrama de bloques de la normalización de la señal de voz para una frecuencia de muestreo de 8000Hz

Para empezar, la señal de voz grabada deberá ser normalizada, de modo que consigamos obtener unas condiciones similares (y más favorables) para todas las señales de voz, independientemente del lugar, dispositivo de grabación, etc.

En primer lugar, la señal será transformada a "mono" (un solo canal, eliminando la información existente que pudiese haber en otro canal en el caso de "estéreo") y disminuiremos su frecuencia de muestreo, consiguiendo así reducir todo lo posible el coste computacional al reducir el número de muestras de que se dispone. Para hacerlo sin pérdida de información, se muestreará, según el teorema de *Nyquist*, al doble de la frecuencia máxima que se quiera obtener. Para el caso de la voz, la frecuencia máxima que consideraremos en nuestra aplicación será de 4000 Hz (es decir, con calidad de canal telefónico), con lo que se fijará la frecuencia de muestreo (mínima) en 8000 Hz.

A continuación, la señal de voz será tratada para eliminar la componente paso bajo, ya que no contendrá información de la voz en sí misma, sino únicamente ruido (sobre todo en torno a los 50 Hz debido a la influencia de la red eléctrica). Para ello, pasaremos la señal por un filtro FIR paso alto de orden N; para eliminar la mayor cantidad de ruido de bajas frecuencias, elegiremos una frecuencia de corte del filtro de 70 Hz y un orden elevado, por ejemplo, $N = 100$, de modo que la banda de transición del filtro sea lo más reducida posible.

2.1.3 Obtención de la información de pitch

Una vez disponemos de la señal normalizada y filtrada, podemos proceder a obtener la información que necesitamos de ella, es decir, la frecuencia de pitch o frecuencia del pulso glotal.

Para ello podemos intentar obtenerla directamente a partir de la autocorrelación localizada de la señal de voz, es decir, la autocorrelación de cada una de las ventanas de N muestras de la señal tomadas con un desplazamiento entre ellas de M muestras.

Para minimizar el coste computacional del cálculo de la autocorrelación localizada, el valor de N debe ser pequeño (manteniendo así la estacionariedad en la señal enventanada y pudiendo obtener la información de la frecuencia de pitch con la suficiente precisión), pero siendo lo suficientemente grande como para poder tomar varios periodos de la señal, mientras que el valor de M ha de ser elevado, de modo que el número de ventanas sea lo más reducido posible. Los valores óptimos elegidos serán $N = 500$ y $M = 180$ (que para la frecuencia de muestreo de $8000\ Hz$ serán de $62{,}5\ ms$ y $22{,}5\ ms$ respectivamente, suficiente para dar una representación fiable de la evolución temporal de la autocorrelación sin la necesidad de realizar un número excesivo de operaciones).

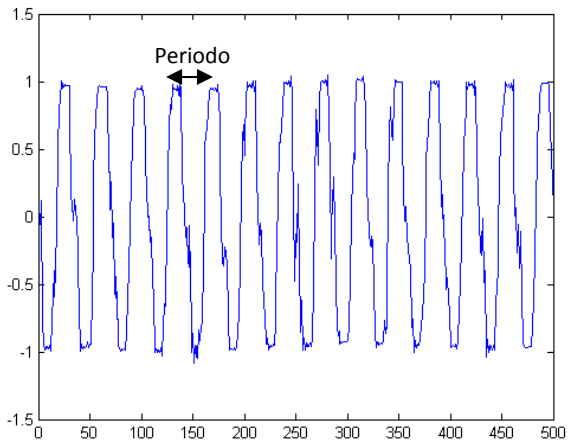

Figura 2.3. Señal enventanada correspondiente a una ventana de la señal de voz

En la Figura 2.3 podemos observar una de las señales enventanadas obtenidas para el cálculo de la autocorrelación localizada; en ella, observamos el carácter periódico de esta señal, cuyo periodo corresponderá al periodo del tren de impulsos que conforma el pulso glotal. Podemos comprobar que, para el valor de $N = 500$ elegido, y considerando valores normales de la frecuencia de pitch (como el de la Figura 2.3), la señal enventanada va a contener varios periodos del pulso glotal.

Calculando, a continuación, su autocorrelación, podemos obtener el valor del periodo de pitch, que corresponderá al número de muestras para las que se obtiene el primer máximo de la autocorrelación (el resto de máximos que aparecen se deben a la periodicidad de la señal).

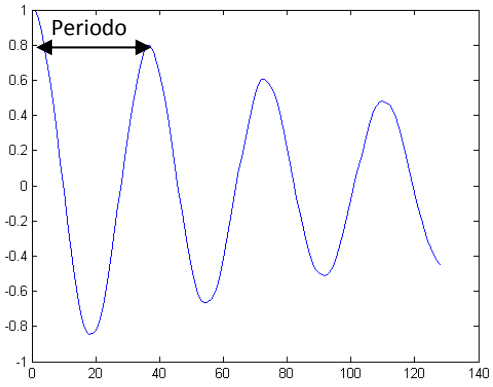

Figura 2.4. Autocorrelación correspondiente a la señal enventanada

De esta forma, efectuaremos este cálculo para todas las ventanas temporales y representaremos el resultado de forma que cada columna representará la autocorrelación $r(\tau)$ de esa ventana (como se puede ver en la Figura 2.5).

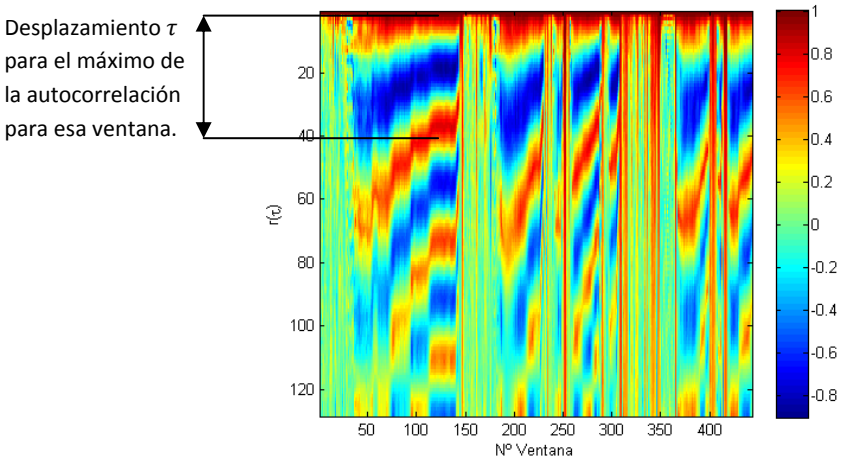

Figura 2.5. Autocorrelación localizada de la señal de voz. La ubicación de los máximos en cada ventana indica periodicidad con ese valor de desplazamiento como periodo.

En los máximos de la autocorrelación encontramos, como se ha comentado anteriormente, indicios de la periodicidad de la señal, debido al carácter periódico del tren de impulsos que conforma el pulso glotal. Sin embargo, también encontramos información correspondiente al tracto vocal que, al colorear la señal periódica de los pulsos glotales, hace que aparezcan armónicos que generan confusión (puesto que su energía puede llegar a ser comparable o incluso mayor a la del armónico fundamental que estamos buscando), pudiendo generar errores en el proceso de detección de pitch.

Por ello, para conseguir obtener esta información de manera más precisa, se puede proceder de dos maneras distintas que serán detalladas a continuación.

2.1.3.1 Análisis LPC

Como se ha explicado en el apartado 2.1.1, la señal de voz $s(n)$ puede ser aproximada por la convolución de la excitación provocada por el pulso glotal, $e(n)$, con el filtro todo-polos que simula el tracto vocal $h(n)$. Así, realizando el filtrado de la señal de voz con un filtro $a(n)$ inverso a $h(n)$ podemos obtener la señal de excitación estimada $\hat{e}(n)$.

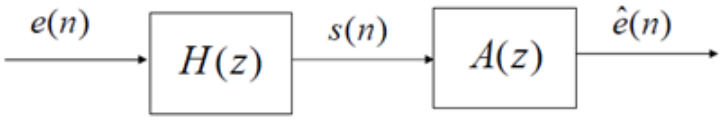

Figura 2.6. Diagrama de bloques del filtro inverso

Para realizar el cálculo del filtro inverso $a(n)$ se hace uso del análisis LPC, mediante el que se estima, para cada ventana temporal de la señal de voz $s(n)$, el filtro inverso. Para poder diseñar el filtro inverso de forma que $H(z) = \frac{G}{A(z)}$, el orden P del análisis LPC debe ser lo suficientemente grande como para poder modelar las resonancias principales del tracto vocal, pero no tanto como para tomar también la información de la propia excitación.

El valor óptimo de P se encontrará entre 8, a partir del que aparecen los 4 formantes de mayor energía de la señal de voz, y aproximadamente 20, a partir del cual comenzará a introducirse la información del pulso glotal. Para estimar el filtro del tracto vocal de la manera más fiable posible, elegiremos $P = 16$, consiguiendo obtener así los 4 formantes dejando cierto margen para no introducir la información de la excitación.

Convolucionando la señal de voz con este filtro inverso obtenemos el error de predicción, $\hat{e}(n)$, blanqueando de esta forma la señal y eliminando en gran parte el problema de los armónicos. A partir de este error de predicción se calcula su autocorrelación localizada, tomando como parámetros N, el tamaño en muestras de las ventanas temporales, y M, el desplazamiento entre ventanas sucesivas (en muestras). Como hemos comentado en el apartado anterior, para minimizar el coste computacional elegiremos $N = 500$ y $M = 180$ (62,5 ms y 22,5 ms).

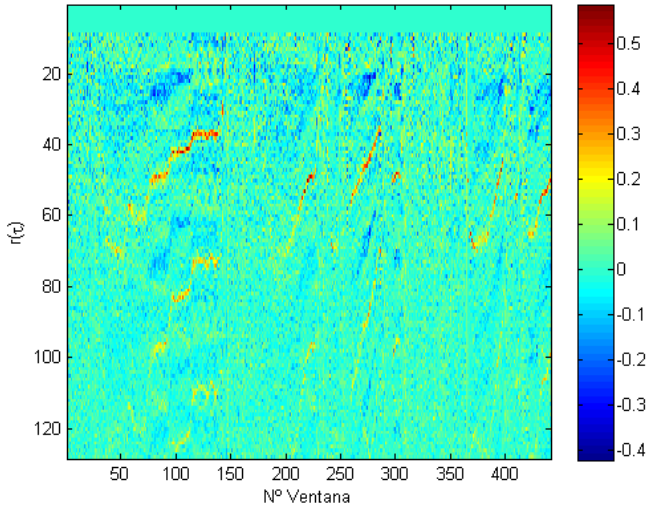

Figura 2.7. Autocorrelación localizada del error de predicción

La Figura 2.7 muestra la energía de la autocorrelación localizada del error de predicción; comparando con la autocorrelación de la señal original, vemos que en este caso los máximos se encuentran mucho más localizados dentro de cada ventana, de manera que conseguiremos obtener una menor confusión en la obtención del máximo.

Este máximo se encontrará en torno al periodo (en muestras) correspondiente a la frecuencia de pitch según la expresión:

$$F_p = \frac{F_m}{T_p}$$

con F_m la frecuencia de muestreo de la señal, T_p el periodo correspondiente a la frecuencia de pitch en muestras, y F_p la frecuencia de pitch. Según esta relación, podemos eliminar la información de la autocorrelación de las 10 primeras muestras sin perder información relevante de la frecuencia de pitch, puesto que corresponderán a frecuencias superiores a $800\,Hz$ (frecuencias de pitch que el ser humano no puede alcanzar).

Además de los máximos de energía en torno a T_p, observamos que también aparecen máximos en torno a $2T_p, 3T_p$, etc., debido al carácter periódico de la señal enventanada. Estos máximos pueden crear confusión en nuestro sistema, de manera que, si bien una buena primera aproximación para estimar la frecuencia de pitch es obtener directamente el máximo de la autocorrelación para cada ventana, el resultado no será lo suficientemente preciso.

2.1.3.2 Análisis homomórfico (Cepstrum)

Otra manera de obtener la información de pitch a partir de la señal de voz filtrada es mediante el Cepstrum.

El Cepstrum se basa en realizar la transformación homomórfica sobre la señal de voz, que consiste en convertir la convolución en una suma, no siendo así necesario el cálculo del filtro inverso para estimar la excitación.

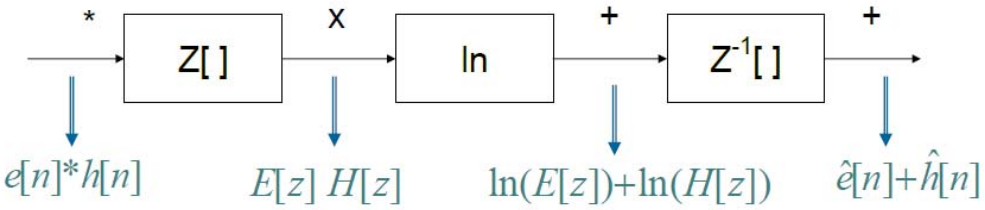

Figura 2.8. Diagrama de bloques del Cepstrum

Para ello, partimos de la señal de voz:

$$s[n] = e[n] * h[n]$$

Aplicando la transformada Z, obtenemos:

$$S[z] = E[z]H[z]$$

de forma que transformamos la convolución en producto. Para transformarlo a suma, aplicamos logaritmos a ambos lados de la igualdad, de forma que:

$$\ln S[z] = \ln(E[z]H[z]) = \ln(E[z]) + \ln(H[z])$$

Definiendo la señal cepstral:

$$\hat{s}[n] = Z^{-1}\left[\ln Z\left[s[n]\right]\right]$$

y trasladándonos al dominio cepstral, obtenemos:

$$\hat{s}[n] = \hat{e}[n] + \hat{h}[n]$$

donde n representa la *quefrencia*, que podremos asociar con el índice temporal.

La ventaja de utilizar el Cepstrum para analizar la señal de voz es que, en el dominio cepstral, el filtro que conforma el tracto vocal está concentrado en los valores bajos de n, mientras que la excitación será un tren de impulsos cuya amplitud decae de forma exponencial, y cuyo periodo se corresponderá con el período del pulso glotal.

Puesto que, en general, la frecuencia de pitch de la voz humana no puede alcanzar valores superiores a aproximadamente $600\ Hz$, que, trabajando a una frecuencia de muestreo de $8000\ Hz$, correspondería a un periodo de pitch de 13 muestras (es decir, en general el periodo de pitch será superior a 13 muestras), podremos separar la información del filtro de la de la excitación.

En la Figura 2.9 podemos observar el Cepstrum de una ventana concreta y cómo la información más relevante del filtro queda contenida en las primeras muestras, mientras que el máximo del tren de impulsos que representa la parte del pulso glotal aparece en torno a las 40 muestras.

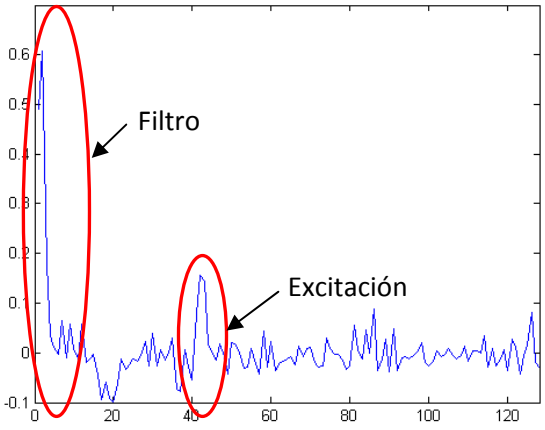

Figura 2.9. Separación de la información del filtro y la excitación en la señal cepstral

Así, de la misma forma que con el análisis LPC, calcularemos el Cepstrum de la señal enventanada, eligiendo, con el mismo criterio del análisis LPC que nos permite obtener la información de la frecuencia de pitch con el menor coste computacional posible, el tamaño en muestras de las ventanas temporales $N = 500$ y el desplazamiento de la ventana en muestras $M = 180$ ($62{,}5\ ms$ y $22{,}5\ ms$).

Representando el resultado de forma que cada columna corresponda al Cepstrum de esa ventana, y habiendo eliminado previamente las 13 primeras muestras del mismo (que como se ha comentado previamente, corresponderán a los valores más significativos del tracto vocal y no contendrán información de la frecuencia de pitch) dispondremos de la evolución de la información del pulso glotal.

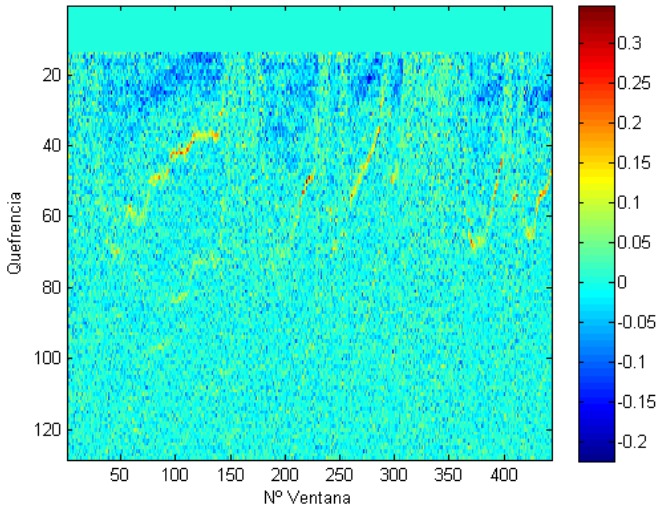

Figura 2.10. Cepstrum de la señal de voz

Del mismo modo que mediante el análisis LPC, podemos ver que con el uso del Cepstrum los máximos se encuentran mucho más localizados dentro de cada ventana, reduciéndose así la confusión en la obtención del máximo, pero que siguen apareciendo máximos secundarios en los múltiplos del periodo del pulso glotal, con lo que obtener directamente el máximo de cada ventana tampoco será un resultado suficientemente preciso.

2.2 Algoritmo de Viterbi

En el punto anterior hemos obtenido la información de pitch de la señal de voz; sin embargo, el cálculo que se realiza limitaría la dependencia de la frecuencia de pitch a la longitud de la ventana temporal, que ha de ser pequeña para poder mantener las condiciones de estacionariedad de la señal.

En realidad, la frecuencia de pitch de la señal de voz depende también de la información anterior (y posterior) a la ventana que se esté analizando, con lo que obtenerla directamente de este modo puede generar errores (debido principalmente a la influencia del ruido y los armónicos).

Para solucionar este problema, realizaremos un seguimiento robusto del pitch basado en el uso del algoritmo de Viterbi [1], [2], de forma que se puedan evitar estos errores y dar un resultado más fiable.

El algoritmo de Viterbi hará uso de la información del pitch obtenida por medio del análisis LPC o del análisis homomórfico. Sus estados se corresponderán con las

muestras de la autocorrelación; así, tomaremos el número de estados $Q = 256$ con el que conseguiremos un buen compromiso entre el coste computacional del cálculo y su precisión (equiespaciando cada estado 1 muestra y centrándolos en $1,2,\dots 256$, y utilizando la relación $F = \frac{F_m}{n}$, obtendremos unos márgenes de frecuencia de pitch más que suficientes entre $30\ Hz$ y $8000\ Hz$). Además, puesto que es una potencia de 2, conseguiremos optimizar el cálculo de la autocorrelación, dado que requiere realizar la transformada rápida de Fourier (FFT) y su cálculo se realiza de manera eficiente para un número de puntos que sea una potencia de 2.

2.2.1 Matriz de transiciones

Para obtener el mejor camino posible, el algoritmo de Viterbi calcula, mediante el diagrama de Trellis, el camino que, para cada instante $t \in 1,2,\dots T$ (siendo T el número de muestras de nuestra observación) y para cada estado $j \in 1,2,\dots Q$ maximice la expresión:

$$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_t) \qquad 2 \le t \le T$$

, siendo i el estado en el instante $t-1$, a_{ij} la probabilidad de transición del estado i al estado j y $b_j(O_t)$ la probabilidad a priori del estado j en el instante t.

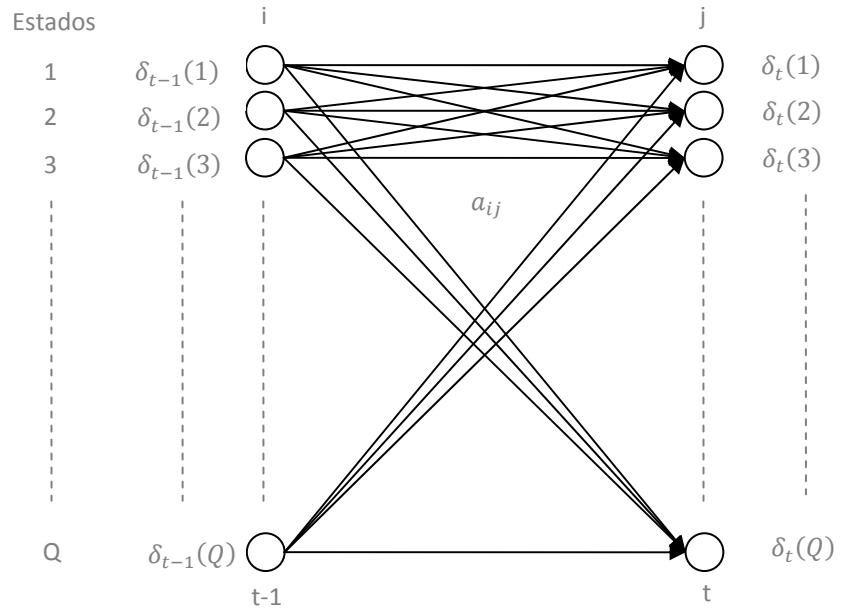

Figura 2.11. Diagrama de Trellis

A partir de esta expresión podemos obtener también el argumento que hace máxima dicha función, de modo que:

$$\psi_t(j) = \arg\max_{1\leq i\leq N}\left[\delta_{t-1}(i)a_{ij}\right]$$

Así, el estado óptimo para el instante T será:

$$q_T{}^* = \arg\max_{1\leq i\leq N}\left[\delta_T(i)\right]$$

y, mediante *Backtraking*, podremos obtener los estados óptimos para el resto de instantes:

$$q_t{}^* = \psi_t(q_{t+1}{}^*) \qquad\qquad t = T-1, T-2, \dots, 1$$

obteniendo de esta forma el camino óptimo.

Como valor de la probabilidad a priori de cada estado tomaremos el valor de la autocorrelación o el Cepstrum en cada ventana dividido por la suma de los valores de cada ventana (de manera que la suma de las probabilidades en cada ventana sea 1).

Así, para variar el resultado del algoritmo de Viterbi, modificaremos las probabilidades de transición entre los estados, es decir, la matriz de transiciones, que definiremos de forma exponencial, de manera que la mayor probabilidad de transición será de un estado a él mismo, e irá decayendo de manera exponencial en función de su separación frecuencial:

$$a_{ij} = \frac{e^{-k(|i-j|+1)}}{\sum_n e^{-k(|n-j|+1)}}$$

De esta forma, para permitir las transiciones entre los estados más próximos con mayor probabilidad sin impedir las transiciones a los estados más lejanos (aunque dándole una probabilidad más baja de transición), elegiremos un valor de $K = 0.5$.

Para minimizar el coste computacional, podemos reducir más el número de estados ajustando los límites en torno a los que los situamos; así, con $Q = 128$ (manteniendo que sea una potencia de 2), y situando los estados a partir de 10 muestras equiespaciados cada 1 muestra, conseguiremos unos límites en frecuencia entre $63\ Hz$ y 800 Hz, que seguirán siendo suficientes.

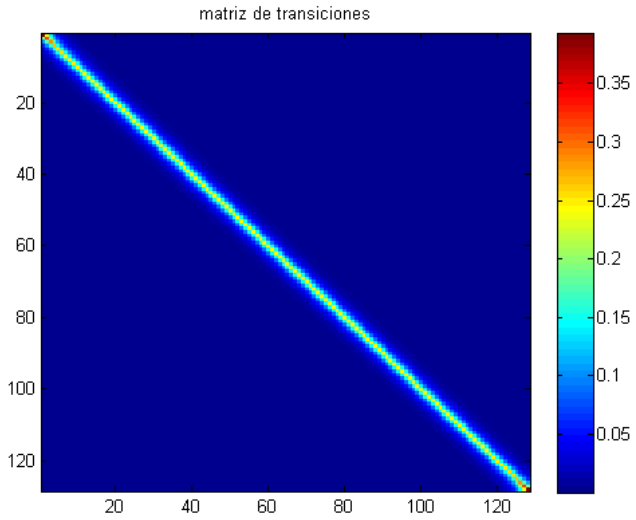

Figura 2.12. Matriz de transiciones del algoritmo de Viterbi para $Q = 128$

El número de estados es un factor clave, ya que el coste computacional de la estimación del pitch dependerá directamente del número de estados puesto que para calcular el camino óptimo en cada instante temporal se ha de calcular el mejor camino posible desde y hacia cada uno de los estados existentes. Por ello, para disminuir aún más el coste computacional podríamos considerar:

- Dependiendo de la resolución en frecuencia que se desease obtener, variar el número de muestras entre cada uno de los estados (por ejemplo, conseguiríamos los mismos límites con $Q = 64$ situando los estados cada 2 muestras, aunque perdiendo resolución en frecuencia).
- Perder la optimización de la transformada de Fourier que se obtiene al realizarla de 2^n puntos, consiguiendo así tener menos estados, y ajustando más los limites en frecuencia (por ejemplo, con $Q = 96$ y situando los estados a partir de 18 muestras, obtendríamos unos límites entre $70\ Hz$ y $440\ Hz$).

2.2.2 Algoritmo de Viterbi enventanado

Como la estimación de la frecuencia de pitch ha de hacerse en tiempo real, la solución anterior no es viable: si, por ejemplo, estimásemos la frecuencia de pitch durante varios minutos (lo que no sería improbable para nuestra aplicación), se debería aplicar el algoritmo de Viterbi a un número de ventanas demasiado grande (que seguiría aumentando conforme pasase el tiempo), lo que provocaría que la latencia aumentase demasiado como para proporcionar una realimentación para el usuario.

La solución a este problema consistirá en realizar el algoritmo de Viterbi con memoria finita: en lugar de realizarlo para toda la señal, se realizará para un número K de

ventanas. De esta manera conseguiremos que el coste computacional para un instante determinado sea independiente de la duración de la señal de voz.

El valor de K será el que fijará las prestaciones del sistema. Por un lado, K debe ser bajo, de forma que se reduzca el coste computacional, mientras que por el otro K debe ser alto, de forma que se disponga de la información suficiente como para poder realizar un seguimiento robusto del pitch.

Además, hay que tener en cuenta también que el sistema tendrá un estado transitorio al empezar durante $K-1$ ventanas hasta poder devolver una estimación de la frecuencia de pitch correcta, es decir, hasta que haya K ventanas con las que realizar este algoritmo de Viterbi enventanado.

2.2.2.1 Directo

La primera aproximación para conseguirlo sería utilizar "fuerza bruta", es decir, tomar directamente, para cada instante, la información de la ventana actual y las $K-1$ ventanas temporales anteriores para realizar el algoritmo de Viterbi y obtener así la frecuencia de pitch de ese instante.

Para seleccionar el valor óptimo de K comprobaremos, para diferentes señales de voz (con saltos en la frecuencia de pitch, de forma que podamos comprobar si la estimación está funcionando correctamente), tanto el resultado obtenido por el algoritmo de Viterbi enventanado como el tiempo que le ha llevado obtenerlo, de forma que podamos calcular ese valor K que optimice el tiempo de cálculo minimizando el error de estimación del pitch.

En la Figura 2.13 se puede ver la comparación entre la estimación del pitch de la señal completa y la estimación con la señal enventanada con diferentes números de muestras por ventana K.

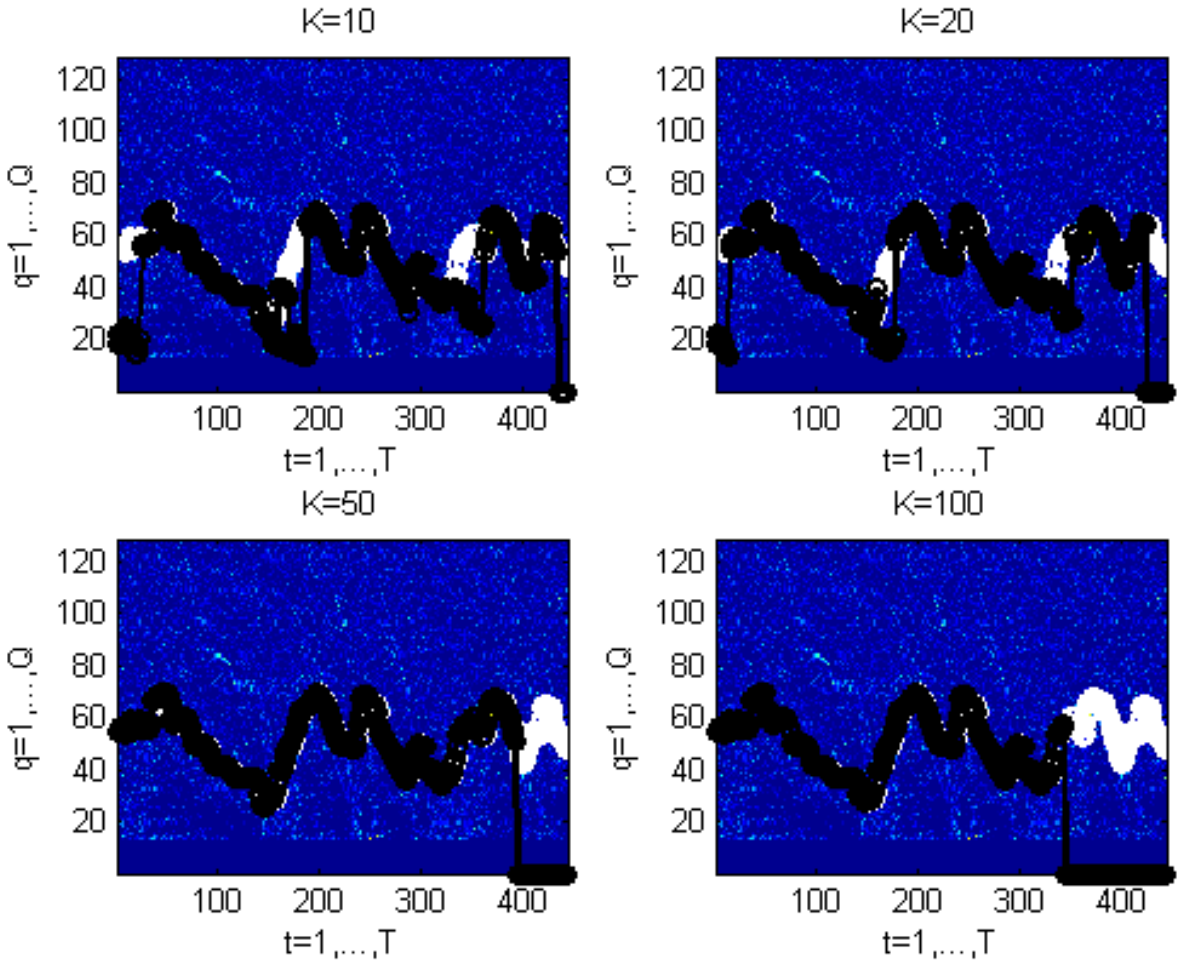

Figura 2.13. Comparación entre la estimación de la frecuencia de pitch de la señal completa (en blanco) y la señal enventanada (en negro) de K ventanas.

En ella observamos cómo, con $K = 10$ y $K = 20$, aparecen desajustes entre la estimación del pitch con la señal completa y la señal enventanada, haciéndose prácticamente inapreciables para $K = 50$ y desapareciendo para $K = 100$.

Teniendo en cuenta este resultado, que el tiempo de ejecución no aumenta de forma significativa (si bien si ligeramente) en función de K y que para obtener la salida de la ventana actual deberemos disponer de la información de las $K - 1$ ventanas anteriores almacenadas (es decir, que conforme K aumente necesitaremos guardar cada vez más datos para poder obtener la respuesta a la ventana actual), podremos elegir $K = 100$, con el que vemos que obtendremos una salida prácticamente idéntica a la de la señal completa (la caída que aparece en los últimos instantes de cada señal se debe a que nuestra señal es finita, y no se dispone aún de la información necesaria para proporcionar la salida).

Si bien en MatLAB este método (es decir, tener que tomar una matriz de $Q * K$ elementos y volver a calcular, para las K ventanas, los estados óptimos) no presentaría demasiado problema, a la hora de realizar este algoritmo en otro lenguaje (como C o Java, lenguajes en los que se encuentra implementada actualmente la aplicación "PreLingua") supondría tener que realizar $Q * K$ asignaciones y obtener el máximo de K vectores de Q elementos para cada instante.

Además de costoso, este proceso es innecesario, puesto que, partiendo de que disponemos de la información de la ventana N, para la $N + 1$ únicamente necesitaríamos la información de ese instante (ya que ya disponemos de la de las $K - 2$ ventanas anteriores), pasando así a tener que realizar únicamente Q asignaciones y obtener el máximo de un vector de Q elementos.

2.2.2.2 Circular

Para conseguir realizar la mejora que comentábamos en el apartado anterior mediante la que consigamos reaprovechar la información de una ventana a la siguiente, realizaremos el siguiente proceso:

1. En primer lugar, declararemos una matriz de Q filas por K columnas.
2. Para el instante K, cargaremos en esa matriz la información de la ventana actual y las $K - 1$ anteriores, y realizaremos el algoritmo de Viterbi sobre esta matriz.
3. Para el instante $K + 1$, sobrescribiremos la información en la primera columna, y realizaremos el algoritmo de Viterbi con la información desde la segunda columna hasta la última y a continuación esta primera columna.
4. Para los sucesivos instantes $K + 2, K + 3 \ldots 2K$ realizaremos la misma operación que en el caso anterior, ubicando la información en las columnas $2, 3, \ldots K$ y realizando el algoritmo de Viterbi de forma que la columna correspondiente al último instante sea la $2,3, \ldots K$.
5. Para el instante $2K + 1$, volveremos a realizar el paso 3, y continuaremos el proceso hasta haberlo realizado para todos los instantes de la señal de voz.

Comparando los algoritmos de Viterbi enventanados directo y cíclico, observamos que obtenemos los mismos resultados, si bien con una importante mejora en el número de operaciones a realizar (reflejada en una disminución sustancial del tiempo de ejecución).

Gracias a esta mejora, el tiempo de computación con los diferentes valores de K será prácticamente el mismo, con lo que conseguimos así tanto optimizar la salida del sistema como minimizar su coste computacional.

2.3 Obtención de los parámetros estadísticos de la frecuencia de pitch: el electroglotógrafo

Con la introducción del algoritmo de Viterbi hemos conseguido proporcionar a nuestro sistema robustez ante las transiciones bruscas, obteniendo una estimación de la frecuencia de pitch mucho más precisa. En general, esta solución devolverá la frecuencia de pitch correcta; no obstante, debido a la influencia del ruido y a los máximos que aparecen en los múltiplos del periodo de pitch, pueden existir casos en los que el resultado sea erróneo.

Para intentar evitar este problema, vamos a modificar los valores de las probabilidades a priori de los estados, de forma que se tengan en cuenta los datos estadísticos de la frecuencia de pitch.

Para obtener estos datos, necesitaremos hacer uso de una base de datos de señales de voz lo suficientemente representativa de la población, y, dado que van a ser utilizados como base para el funcionamiento del sistema, necesitamos obtener estos valores de la frecuencia de pitch con la mayor exactitud posible. Por este motivo, las señales van a ser obtenidas mediante un electroglotógrafo.

El electroglotógrafo es un dispositivo que permite grabar únicamente la señal correspondiente al pulso glotal gracias a un par de electrodos que se sitúan a ambos lados del cuello. Gracias a él, obtendremos las señales con el menor ruido posible y, de esta forma, conseguiremos ser capaces de asegurar casi con total certeza que el máximo de la autocorrelación localizada en cada ventana corresponderá con el periodo correspondiente al pulso glotal.

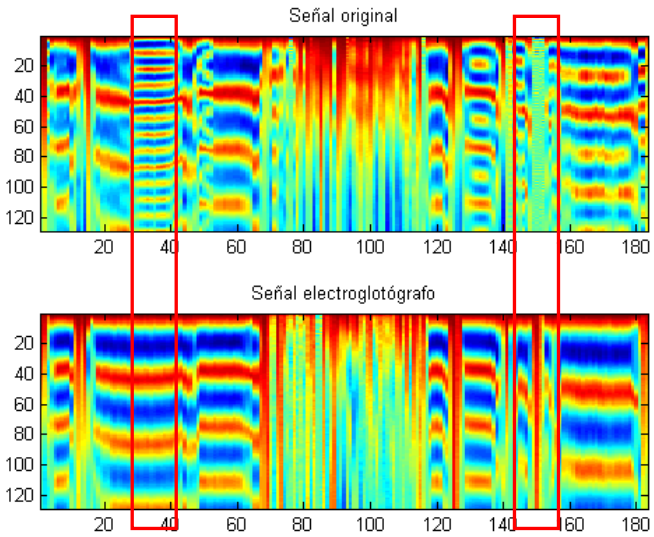

Figura 2.14. Autocorrelación obtenida para una señal de voz; arriba, grabada con un micrófono; abajo, grabada con el electroglotógrafo.

En la Figura 2.14 podemos ver la comparación de la autocorrelación obtenida para una señal grabada con un micrófono ordinario y la obtenida para esa misma señal grabada mediante el electroglotógrafo. En ella comprobamos como la autocorrelación obtenida con la señal del electroglotógrafo es mucho más limpia que la obtenida mediante un micrófono (como vemos sobre todo en las zonas marcadas, donde gracias al electroglotógrafo reducimos el "ruido" al eliminar la información innecesaria).

Como demostración para este proyecto, utilizaremos una muestra de 400 señales, cada una de ellas grabadas por una persona diferente[3]. No obstante, para la realización de este sistema, la muestra debería ser mayor, de manera que se consiguiese representar de la forma más fiable posible la distribución de la frecuencia de pitch.

A partir de estas señales, obtendremos los valores de la frecuencia de pitch de aquellas ventanas cuya energía supere un umbral, puesto que serán aquellas cuyos sonidos sean sonoros, y por tanto aquellas que podamos considerar que tienen una frecuencia de pulso glotal.

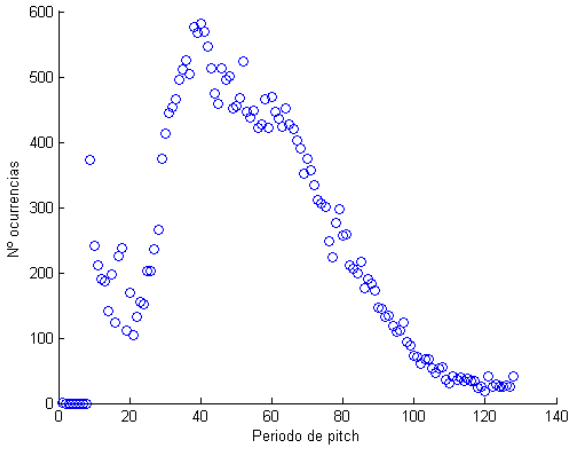

Figura 2.15. Distribución del periodo del pulso glotal para la base de datos de señales obtenidas mediante el electroglotógrafo.

Gracias a estos datos seremos capaces de mejorar los valores de las probabilidades a priori de cada estado al multiplicar los valores tomados previamente (el valor de la autocorrelación del error de predicción) por la probabilidad de ocurrencia de cada periodo de pitch (normalizado).

Además de obtener estos valores del periodo de pitch, obtendremos también el valor absoluto de la variación entre el anterior periodo de pitch estimado y el actual. De esta forma, seremos capaces de proporcionar a la matriz de transiciones del algoritmo de Viterbi unos valores más ajustados a la realidad.

3. http://stimmdb.coli.uni-saarland.de/

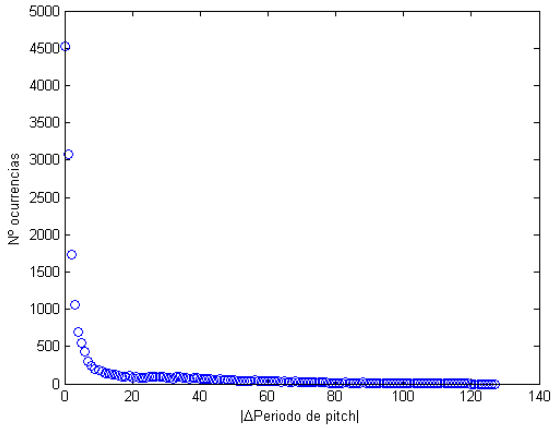

Figura 2.16. Distribución de los cambios en el periodo del pulso glotal para la base de datos de señales obtenidas mediante el electroglotógrafo.

Utilizando los resultados que podemos ver en la Figura 2.16 rediseñaremos la matriz de transiciones; así, en lugar de ser una matriz exponencial, los valores seguirán la expresión:

$$a_{ij} = \frac{p(|i-j|)}{\sum_n p(|n-j|)}$$

donde $p(x)$ será la probabilidad de que el valor absoluto del incremento del periodo sea igual a x, que podremos definir según los datos obtenidos como:

$$p(x) = \frac{\{Nº\ ocurrencias\ \Delta Periodo = x\}}{\sum_{i=0}^{127}\{Nº\ ocurrencias\ \Delta Periodo = i\}}$$

obteniendo así la matriz de transiciones de la Figura 2.17.

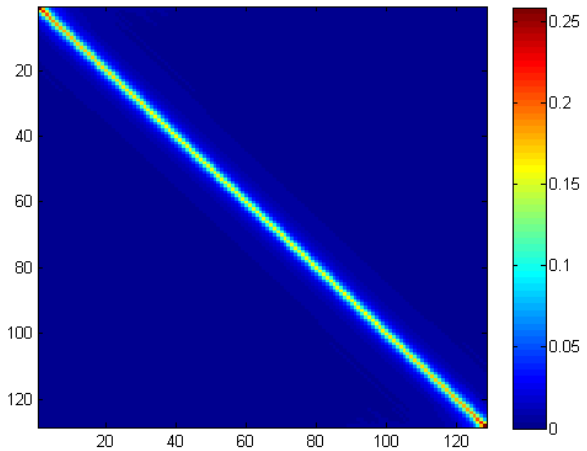

Figura 2.17. Matriz de transiciones del algoritmo de Viterbi para $Q = 128$ con los valores obtenidos a partir de la base de datos.

Así, en la Figura 2.18 podemos ver cómo, gracias a la información de la distribución de la frecuencia de pitch introducida en el sistema, se reduce tanto la influencia del ruido como la de la de los máximos que aparecen en los múltiplos del periodo del pulso glotal.

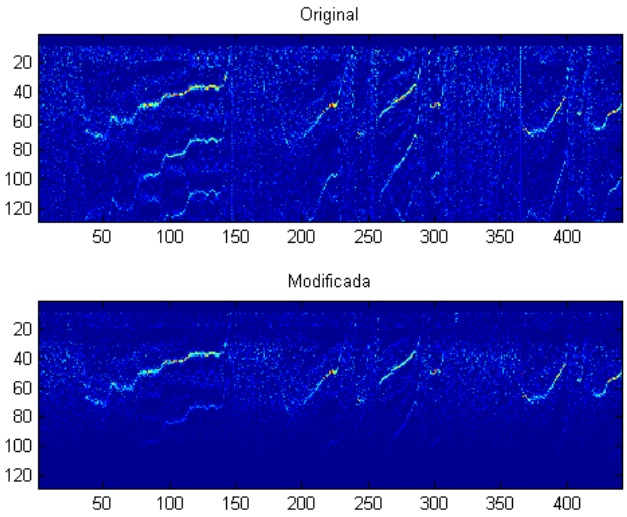

Figura 2.18. Matrices de probabilidades a priori obtenidas para el algoritmo de Viterbi; arriba, la obtenida directamente a partir de la autocorrelación del error de predicción; abajo, la misma multiplicada por la probabilidad del periodo del pulso glotal correspondiente.

De esta manera, al aplicar el diagrama de Trellis sobre esta señal más limpia, el resultado obtenido será más fiable (como se puede observar en la Figura 2.19).

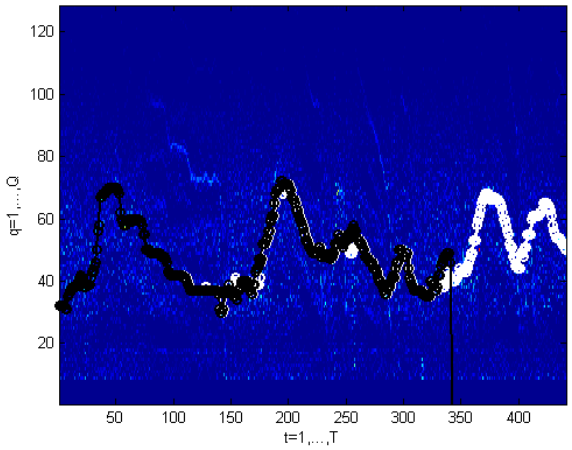

Figura 2.19. Comparación entre la estimación de la frecuencia de pitch de la señal completa (en blanco) y la señal enventanada con 100 ventanas (en negro) con la información de la distribución del pitch.

Además, gracias a que la base de datos de la que disponemos también contiene las señales de voz grabadas con un micrófono normal (además de con el electroglotógrafo), podremos evaluar el error cuadrático medio obtenido entre la frecuencia de pitch de la señal grabada con el electroglotógrafo (que, como se ha comentado anteriormente, podemos considerar como la correcta) y la estimación obtenida con nuestro algoritmo sobre la señal grabada con el micrófono en función del tamaño del número de ventanas K.

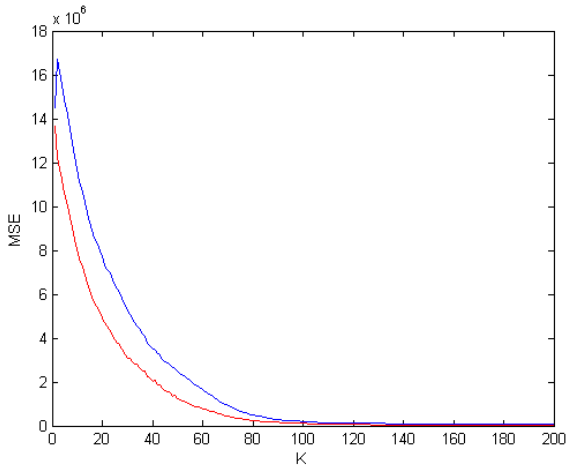

Figura 2.20. Error cuadrático medio entre la frecuencia de pitch de la señal del electroglotógrafo y la estimación realizada en la señal del micrófono en función del número de ventanas K. En azul, mediante el análisis LPC; en rojo, mediante el Cepstrum.

En la Figura 2.20 podemos ver como este error disminuye de manera exponencial tanto con el uso del análisis LPC como con el del Cepstrum, siendo estos errores de un orden de magnitud similar (si bien el obtenido mediante el Cepstrum parece ser un poco menor, pero necesitaríamos calcular el error obtenido con un número suficientemente elevado y variado de señales de voz antes de poder asegurar que siempre vaya a ser así).

Podemos comprobar también como el valor elegido previamente de $K = 100$ es, como habíamos visto sobre la Figura 2.13, un buen compromiso entre el error obtenido en la estimación del pitch (valor aproximado a partir del cual el error se estabiliza y decae más lentamente) y la cantidad de memoria necesaria para almacenar la información de las ventanas anteriores.

3. Detección facial mediante clustering

En este apartado se va a detallar el trabajo realizado para el desarrollo de un algoritmo de detección de caras basado en la agrupación (o *clustering*) del color sobre una secuencia de vídeo.

3.1 Tratamiento previo de la señal de vídeo

3.1.1 La señal de vídeo

La señal de vídeo es una sucesión de imágenes cuyo número y tamaño vendrán definidos por el número de fotogramas por segundo o *FPS* (*Frames Per Second*) y por la resolución de la señal de vídeo, respectivamente.

Así, el tratamiento de la señal de vídeo se reducirá al tratamiento de las imágenes que la conforman.

3.1.2 Redimensionado de la señal de vídeo

El primer paso a realizar será normalizar la señal de vídeo, de manera que el punto de partida del sistema sea siempre una señal con unas características similares y que favorezcan el bajo coste computacional que queremos proporcionar a nuestro sistema.

En primer lugar, deberemos tener en cuenta el número de imágenes por segundo: este valor deberá de ser bajo de forma que se reduzca el número de imágenes que se deben tratar cada segundo, pero alto para mantener la respuesta en tiempo real. Por

ello inicialmente elegiremos un valor de 5 *FPS*, que nos proporcionará una precisión suficiente y reducirá el coste computacional.

Del mismo modo, deberemos considerar la resolución de la señal de vídeo, es decir, el tamaño de las imágenes a tratar, que será uno de los factores más críticos, puesto que necesitaremos evaluar cada píxel (como se explicará en apartados posteriores): para poder minimizar el número de operaciones a realizar, deberemos reducir este tamaño, de manera que el número de píxeles en cada fotograma se reduzca. Sin embargo, para hacerlo sin perder la información disponible en dicha imagen, fijaremos un límite de $80x60$ (es decir, 80 píxeles de anchura por 60 de altura).

Las reducciones llevadas a cabo en imágenes por segundo y en resolución son muy importantes de cara a la implementación de este sistema en un lenguaje de programación como *C* o *Java*, ya que, si bien en MatLAB podemos tratar fácilmente los píxeles de las imágenes como matrices (de forma que el coste computacional no se incrementaría demasiado aunque la imagen no fuese pequeña), en la mayoría de lenguajes deberemos tratar los elementos de la imagen uno a uno.

Una vez realizado este redimensionado de la señal de vídeo, pasaremos a normalizar los valores de cada píxel.

3.1.3 Teoría del color

Una imagen digital en color se representa normalmente mediante su descomposición en componentes RGB (*Red Green Blue*), de manera que el valor de cada píxel quede representado por el valor de cada una de estas tres componentes. Este valor dependerá del formato de vídeo de la señal, concretamente del parámetro conocido como profundidad de color, es decir, el número de bits utilizados para representar el valor del píxel.

En la actualidad es muy común el uso de 24 bits (8 bits por componente), lo que se conoce como *true color* o color verdadero, puesto que el ojo humano se vuelve incapaz de diferenciar dos colores que difieran en un bit (y añadir más bits para la representación del color no nos permitiría obtener más colores perceptibles al ojo humano, puesto que seríamos incapaces de diferenciarlos). Así, el valor de estas componentes sería un entero entre 0 y 255 (al utilizar 8 bits); sin embargo, en caso de utilizar un número de bits cualquiera n, el valor de ese entero estaría comprendido entre 0 y $2^n - 1$, por lo que deberemos normalizar su valor.

Para ello, transformaremos el valor de cada componente en un número real cuyos límites irán entre 0 y 1, de manera que consigamos que el valor del píxel no dependa de la profundidad de color.

Aunque el espacio de color RGB es el más conocido y usado, existen otros espacios de color que presentan determinadas ventajas: en concreto, el espacio de color $YCbCr$ (donde Y representa la *luminancia* o cantidad de luminosidad de la imagen, mientras que Cb y Cr representan la *crominancia* o información del color diferencia de azul y diferencia de rojo, respectivamente) permite separar, casi totalmente, la información de color de la de la luminosidad de la imagen.

3.2 Detección facial

La detección de caras es un problema típicamente resuelto mediante el algoritmo de Viola-Jones [5], [7]. Este método básicamente consiste en comparar máscaras de diferentes valores y tamaños con cada bloque de la imagen del mismo tamaño que dicha máscara.

Este método tiene el problema de ser dependiente de la inclinación de la cara en la imagen; en general, las máscaras empleadas para la detección de caras se plantean para caras sin inclinación, de forma que los resultados obtenidos empeoran conforme aumenta la inclinación de la cara respecto a esa posición.

Una solución a este problema es incluir dos nuevos conjuntos de máscaras que nos permitan detectar caras inclinadas 45º hacia cualquiera de los dos lados [4], de manera que, además de detectar las caras con dicha inclinación, se conseguirá también una mejor detección de las caras con otras inclinaciones (si bien no con la misma precisión). Sin embargo esta solución presenta dos inconvenientes: por un lado, el incremento del coste computacional, puesto que se deberá comparar la imagen con el triple de máscaras, y por otro el aumento de los "falsos positivos" detectados, puesto que se añadirán las detecciones erróneas de dichas posiciones.

Por este motivo, vamos a implementar otro método que permita realizar una detección facial independiente de la inclinación de la cara y con el menor coste computacional posible.

3.2.1 Clustering de Piel mediante mezcla de Gaussianas (MoG)

Para conseguir obtener un sistema con las características comentadas en los párrafos anteriores nos basaremos en el color [6]: diseñaremos un sistema que responda ante los píxeles que se considere que, por su color, pertenecen a una cara.

Para realizar este sistema utilizaremos reconocimiento de patrones, que puede ser desarrollado mediante varias aproximaciones:

- *Redes neuronales*: se define la respuesta como la salida de una red de unidades (neuronas) a una determinada entrada. No necesita mucho conocimiento del problema que se quiere resolver, pero una vez entrenado es demasiado complejo tratar de obtener una función que sea capaz de representarlo.
- *Modelos estadístico*s: se define un modelo estadístico que define la distribución de los parámetros. Una vez entrenado queda definido mediante una función de probabilidad condicionada.

En el caso de la detección de caras, utilizaremos un modelo estadístico debido a que, al ser definido mediante una función de probabilidad, podremos controlar mejor la respuesta del sistema.

Este modelo estadístico será generado mediante una mezcla de gaussianas, es decir, un conjunto de C gaussianas de D dimensiones con sus correspondientes medias, matrices de covarianzas y pesos dentro de la mezcla, de manera que, con el número suficiente de gaussianas, podemos aproximar con suficiente precisión una gran cantidad de problemas, con la ventaja de poder adaptar fácilmente los parámetros.

El entrenamiento del modelo será la parte crítica de nuestro sistema de detección facial, puesto que su respuesta dependerá de los datos proporcionados en esta fase.

3.2.2 Entrenamiento de la MoG mediante el algoritmo Viola-Jones

Para determinar si un píxel es piel, el sistema se basará en la comparación de su respuesta a dos mezclas de Gaussianas: una entrenada con los píxeles correspondientes a la cara y otra entrenada con el resto de píxeles.

Para obtener estas dos mezclas, necesitaremos disponer de píxeles de piel y de "no piel" con los que entrenarlas; para ello, emplearemos el algoritmo de Viola-Jones (con un umbral elevado que nos permita reducir la probabilidad de detectar como cara algo que no lo sea) en cada fotograma de la secuencia de vídeo hasta que sea detectada alguna cara (para lo que, al inicio, el usuario tendrá que mantener la cabeza erguida). De esta forma, tomaremos como datos para realizar el entrenamiento de la mezcla de gaussianas de piel los píxeles (es decir, sus componentes *RGB,* que representan el color en cada píxel) correspondientes al área definida por el algoritmo de Viola-Jones en la imagen normalizada, y como datos para realizar el entrenamiento de la mezcla de gaussianas de "no piel" el resto de píxeles de la imagen.

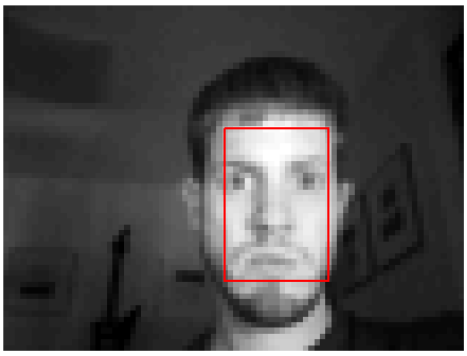

Figura 3.1. Área delimitada por el algoritmo Viola-Jones para el entrenamiento de las mezclas de Gaussianas.

Para minimizar el coste computacional, aprovecharemos la teoría del color transformando nuestra imagen al espacio *YCbCr*, de forma que, como hemos comentado en el punto 3.1, conseguiremos separar casi totalmente la información del color de la luminosidad; de esta manera, podremos reducir la dimensionalidad del problema disminuyendo el número de componentes a tratar de 3 (*RGB*) a 2 (*Cb* y *Cr*).

Una vez disponemos de los datos a emplear, procederemos a crear las dos mezclas de gaussianas necesarias para nuestro sistema.

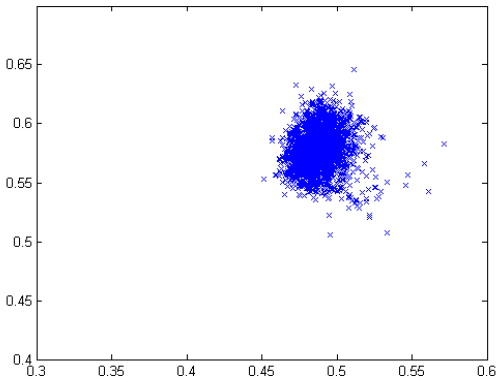

Figura 3.2. Componentes *Cb* y *Cr* de los píxeles correspondientes a una cara.

En primer lugar, necesitaremos definir el número de gaussianas que componen ambas mezclas. En la Figura 3.2 podemos ver una representación de las componentes *Cb* y *Cr* de los píxeles correspondientes a una cara de un fotograma de una señal de vídeo, y como estos puntos se distribuyen de manera aproximada según una distribución normal bidimensional. De esta forma, podemos elegir un número de gaussianas para

esta mezcla $C = 1$ (es decir, la mezcla de Gaussianas será una única gaussiana), consiguiendo obtener una buena aproximación del problema a la par que minimizando el coste computacional (dado que tanto el entrenamiento como la evaluación posterior se realizarán con una sola gaussiana).

En el caso de la mezcla de Gaussianas correspondiente a "no piel", hay que tener en cuenta que va a estar compuesta por cualquier objeto de la imagen que no pertenezca a una cara, de forma que va a depender de cada imagen en concreto. Por ello, para intentar modelar este conjunto de la forma más general posible, elegiremos un número mayor de Gaussianas para la mezcla (por ejemplo, $C = 5$).

La inicialización de cada mezcla de Gaussianas se realizará asignando, para cada una de las gaussianas que la componen, los valores de la media, matriz de covarianzas y peso:

- La media de cada gaussiana tendrá un valor aleatorio entre 0 y 1 para cada componente del color (ya que previamente hemos normalizado las componentes para estar también entre 0 y 1).
- La matriz de covarianza de cada gaussiana será una matriz identidad multiplicada por una constante (en nuestro caso, 0.1)
- Los pesos asignados serán iguales para cada gaussiana (en el caso de *n* gaussianas, $\frac{1}{n}$).

Una vez inicializada la gaussiana, se procederá a su entrenamiento mediante el algoritmo EM (como se describe en el Anexo I).

Puesto que la mezcla correspondiente a la piel únicamente consta de una gaussiana, el resultado del paso E será que el valor esperado de la probabilidad de pertenencia a esa gaussiana $\langle \delta_{z_{n,1}} \rangle$ (que, como se describe en el Anexo I, se obtiene como la probabilidad a posteriori de la componente dada la observación) de cada uno de los N píxeles pertenecientes a la matriz de datos X será igual a 1. De este modo, el cálculo en el paso M de los nuevos parámetros de la gaussiana (el peso p_1, la media μ_1 y la matriz de covarianzas Σ_1) será:

$$p_1 = \frac{\sum_n \langle \delta_{z_{n,1}} \rangle}{N} = \frac{N}{N} = 1$$

$$\mu_1 = \frac{\sum_n \langle \delta_{z_{n,1}} \rangle x_n}{\sum_n \langle \delta_{z_{n,1}} \rangle} = \frac{\sum_n x_n}{N}$$

$$\Sigma_1 = \frac{\sum_n \langle \delta_{z_{n,1}} \rangle (x_n - \mu_1)(x_n - \mu_1)^t}{\sum_n \langle \delta_{z_{n,1}} \rangle} = \frac{\sum_n (x_n - \mu_1)(x_n - \mu_1)^t}{N}$$

es decir, la media y la covarianza de la gaussiana serán la estimación de máxima verosimilitud de la media y la covarianza de la matriz de datos X. Puesto que estos parámetros son constantes para unos datos de entrada fijos (es decir, para los N píxeles con los que realizamos el entrenamiento), las sucesivas iteraciones del algoritmo EM no modificarán su valor, con lo que podremos realizar una única iteración sin perder precisión en el cálculo y minimizando el coste computacional.

Una vez entrenadas ambas mezclas, podremos proceder a la evaluación, con cada una de las mezclas de Gaussianas, de los siguientes fotogramas de la señal de vídeo; para ello, calcularemos la verosimilitud de cada píxel con cada mezcla como:

$$\mathcal{N}(x) = \sum_{c=1}^{C} p_c \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_c|^{1/2}} e^{-\frac{1}{2}(x-\mu_c)^t \Sigma_c^{-1}(x-\mu_c)}$$

siendo x el píxel a evaluar (es decir, sus componentes C_b y C_r), D las dimensiones de la gaussiana (en nuestro caso 2) y μ_c, Σ_c y p_c la media, matriz de covarianzas y peso de cada gaussiana que compone la mezcla.

De esta forma, determinaremos si un píxel se considera piel cuando el cociente entre las verosimilitudes obtenidas para dicho píxel con la mezclas entrenadas con los píxeles de piel y con los píxeles de "no piel" supera el umbral establecido.

El valor de este umbral será crítico en nuestro sistema: por un lado, debe ser elevado, de manera que la probabilidad de detección de los píxeles que correspondan a piel P_d sea elevada; sin embargo, conforme el umbral aumenta no solo aumenta la probabilidad de detección de los píxeles de piel, sino también la probabilidad de detectar píxeles que no sean piel P_{fa}. Por ello, necesitaremos encontrar un valor para el umbral que maximice P_d (idealmente 1) minimizando P_{fa} (idealmente 0).

Inicialmente, estableceremos este umbral igual a 1, que podremos considerar como el valor por defecto, dado que se establecerá que un píxel es piel cuando la verosimilitud obtenida con la mezcla de Gaussianas de piel sea superior a la verosimilitud obtenida con la mezcla de "no piel".

Para comprobar los resultados obtenidos de una forma más visual, representaremos la imagen original frente al resultado obtenido utilizando el sistema (donde cada píxel será blanco si ha sido detectado como piel utilizando el umbral establecido anteriormente, o negro en caso contrario).

Figura 3.3. Respuesta del sistema de detección facial a los fotogramas de una secuencia de vídeo; en blanco los píxeles detectados como cara; en negro, los detectados como "no cara".

En la Figura 3.3 podemos ver que, para el umbral establecido (igual a 1), la respuesta obtenida se comporta según lo esperado: prácticamente la totalidad de píxeles de la cara son detectados como piel, mientras que el resto son detectados como "no piel", consiguiendo obtener la separación entre "cara" y "no cara". Así, obteniendo la media de las posiciones de los píxeles correspondientes a la cara dispondríamos de la posición de la cara en la imagen (e incluso calculando su varianza el tamaño estimado de la misma, pudiendo deducir si se encuentra cerca o lejos de la cámara).

Además, podemos comprobar cómo la inclinación de la cara en la imagen no influye en el resultado obtenido, puesto que este resultado no depende de la cara en conjunto, sino de cada píxel que la compone por separado.

3.2.3 Adaptación de la MoG a las condiciones de luminosidad actuales

La solución planteada funciona correctamente mientras las condiciones de luminosidad de la señal de vídeo permanezcan invariables; sin embargo, cuando estas condiciones cambian, la respuesta empeora.

Figura 3.4. Respuesta del sistema de detección facial a los fotogramas de una secuencia de vídeo con variaciones en la luminosidad; en blanco los píxeles detectados como cara; en negro, los detectados como "no cara".

En la Figura 3.4 podemos ver cómo, ante una variación en la luminosidad de la señal de vídeo con respecto a las condiciones iniciales (es decir, las condiciones presentes en el fotograma con el que se entrena el sistema), perdemos prestaciones en cuanto a la probabilidad de detección de los píxeles correspondientes a la cara.

Para conseguir que el sistema sea capaz de adaptarse a estas variaciones, aprovecharemos el resultado obtenido con el sistema en un fotograma determinado para actualizar los parámetros de las mezclas de Gaussianas que lo componen usando los datos de los píxeles detectados como "cara" y como "no cara".

El cálculo adaptativo de la media de cada gaussiana en el instante $K+1$ se realizará como:

$$\mu_{k+1}^{ADAP} = \alpha\, \mu_k^{ADAP} + (1-\alpha)\, \mu_k^{ML}$$

donde α será el peso utilizado para el cálculo adaptativo, y μ_k^{ML} la media obtenida mediante el algoritmo EM en el instante K, que, como se ha explicado, responderá a la expresión:

$$\mu_k^{ML} = \frac{\sum_n \langle \delta_{z_{n,c}} \rangle x_n}{\sum_n \langle \delta_{z_{n,c}} \rangle}$$

Dado que, en general, los cambios en la luminosidad de la imagen serán graduales y no demasiado frecuentes, no será necesario realizar este cálculo adaptativo para cada fotograma, evitando así realizar un excesivo número de operaciones que no proporcionarían información adicional. Así, actualizaremos estas medias cada segundo (puesto que hemos elegido como parámetro para la señal de vídeo 5 *FPS*, las actualizaremos cada 5 fotogramas) para que el tiempo de respuesta sea bajo, y daremos a α un valor elevado (0.9) que permitirá que se actualice sin perder la información de que disponíamos en fotogramas anteriores.

Figura 3.5. Respuesta del sistema de detección facial con actualización de la media a los fotogramas de una secuencia de vídeo con variaciones en la luminosidad; en blanco los píxeles detectados como cara; en negro, los detectados como "no cara".

Si comparamos las Figuras 3.4 y 3.5 (donde vemos, para una misma señal de vídeo, la respuesta del sistema sin y con el cálculo adaptativo de las medias, respectivamente) podemos comprobar cómo, al realimentar el sistema con la información obtenida en fotogramas anteriores, conseguimos una mejora de las prestaciones ante una variación en la luminosidad de la señal (vemos que, al desplazarse la mezcla de gaussianas para adaptarse a la nueva información de que disponemos, conseguimos detectar con mayor probabilidad los píxeles pertenecientes a la cara).

No obstante, puesto que el parámetro que queremos calcular es la posición de la cara en la imagen, tendremos que comprobar en qué medida mejorar la detección de los píxeles de la cara mejora la estimación de dicha posición.

Para ello calcularemos, para diversas señales (tanto con variación en la luminosidad como sin ella), el error cuadrático entre la posición calculada mediante el algoritmo de Viola-Jones y la calculada por nuestro sistema. Si bien para el desarrollo de la memoria hemos fijado los parámetros de nuestro sistema con los valores que hemos considerado más adecuados, realizaremos el cálculo del error en función de los mismos, de modo que obtengamos los mejores valores para ellos empíricamente.

Hay que tener en cuenta que, puesto que el sistema está basado en dos mezclas de Gaussianas, el resultado que obtendremos será una estimación, puesto que para cada realización variará (puesto que las gaussianas se distribuirán de forma diferente cada vez que ejecutemos el algoritmo EM).

Los parámetros principales del sistema serán el número de Gaussianas de cada una de las dos mezclas de Gaussianas y el parámetro α para el cálculo adaptativo de las medias de las Gaussianas.

Al analizar el número de Gaussianas necesarias para la mezcla de Gaussianas entrenada con los píxeles de la cara, veíamos como la distribución de los mismos se aproximaba a una normal bidimensional. Por ello, vamos a analizar en primer lugar el error obtenido en función de este parámetro.

Nº Gaussianas / Señal Vídeo	1	2	3	4	5
Vídeo 1: Sin cambios en la luminosidad	7.95	7.91	7.81	7.81	7.79
Vídeo 2: Sin cambios en la luminosidad	8.63	8.78	8.58	8.88	9.04
Vídeo 3: Con cambios en la luminosidad	2.70	2.68	2.68	2.66	2.72

Tabla 3.1. Error cuadrático medio entre la posición de la cara obtenida mediante el algoritmo de Viola-Jones y el sistema desarrollado para varias señales de vídeo en función del número de gaussianas de la mezcla entrenada con los píxeles de piel.

Como podemos ver en la Tabla 3.1, el error cuadrático medio obtenido prácticamente es el mismo independientemente del número de Gaussianas, con lo que podemos deducir que nuestra anterior elección de este valor es la más adecuada, puesto que reducirá el coste del sistema manteniendo sus prestaciones.

A continuación realizaremos el mismo proceso con el número de Gaussianas necesarias para la mezcla de "no cara".

Nº Gaussianas / Señal Vídeo	1	2	3	4	5
Vídeo 1: Sin cambios en la luminosidad	9.32	8.95	8.94	9.06	9.01
Vídeo 2: Sin cambios en la luminosidad	8.30	7.67	7.70	7.70	7.86
Vídeo 3: Con cambios en la luminosidad	2.79	2.66	2.60	2.72	2.66

Tabla 3.2. Error cuadrático medio entre la posición de la cara obtenida mediante el algoritmo de Viola-Jones y el sistema desarrollado para varias señales de vídeo en función del número de gaussianas de la mezcla entrenada con los píxeles de "no piel".

Como vemos en la Tabla 3.2, en general conforme aumentamos el número de Gaussianas de la mezcla de "no piel" se consigue disminuir el error cuadrático medio ligeramente, independientemente de si hay o no variaciones en el nivel de luminosidad de la señal de vídeo. Como se ha comentado anteriormente, esto se debe a que, al ejecutar el algoritmo EM con un número mayor de Gaussianas, la mezcla es capaz de ajustarse con más exactitud a esos datos.

Una vez comprobado que los números de Gaussianas para los que obtenemos los mejores resultados son aquellos que hemos seleccionado anteriormente (es decir, $C_1 = 1$ y $C_2 = 5$), los utilizaremos para calcular el error en función de α.

α / Señal Vídeo	0	0.2	0.4	0.6	0.8	1
Vídeo 1: Sin cambios en la luminosidad	7.79	7.51	7.75	7.68	7.92	7.85
Vídeo 2: Sin cambios en la luminosidad	7.91	7.68	7.44	7.54	7.69	7.81
Vídeo 3: Con cambios en la luminosidad	2.61	2.65	2.54	2.66	2.61	2.74

Tabla 3.3. Error cuadrático medio entre la posición de la cara obtenida mediante el algoritmo de Viola-Jones y el sistema desarrollado para varias señales de vídeo en función del valor de α.

Como podemos ver en la Tabla 3.3, los valores del error se comportan según lo esperado:

- Para señales sin cambios de luminosidad, el error cuadrático medio obtenido permanece prácticamente constante con el valor de α, si bien puede darse el caso (como ocurre en el Vídeo 2) de que, aunque no haya habido un cambio en la luminosidad, al actualizar las medias de las gaussianas obtengamos una mejor representación de la los píxeles pertenecientes a cara y/o de los pertenecientes a "no cara", con lo que al establecer $\alpha \neq 1$ (es decir, al actualizar las medias) conseguimos disminuir el error.
- Para señales con cambios de luminosidad, el error cuadrático medio obtenido disminuye cuando $\alpha \neq 1$ (es decir, al actualizar las medias) puesto que conseguimos que las mezclas se adecúen mejor a los píxeles con la nueva iluminación.

De esta forma, vemos que el error depende de α, pero no hay un valor para el que se alcance un mínimo para todos los casos, puesto que va a depender de cada señal de vídeo en concreto. Por ello, a la hora de utilizar este algoritmo, podremos dar un valor por defecto con el que sepamos que se va a obtener, en general, un buen resultado (como, por ejemplo, 0.8), si bien cuando en un futuro desarrollo se traslade este

algoritmo a una aplicación final, lo lógico sería permitir que el usuario pueda configurarlo, de forma que pueda intentar mejorar la respuesta obtenida.

4. Conclusiones

En este apartado vamos a analizar las características de los sistemas diseñados (comprobando que se adecúan a los objetivos definidos en el punto 1 de la memoria), así como los resultados obtenidos con ellos y las posibles mejoras que podrían llevarse a cabo para mejorarlos.

4.1 Seguimiento robusto del pitch

Uno de los objetivos del proyecto era realizar una mejora del algoritmo de estimación de la frecuencia de pitch que proporcionase robustez frente a transiciones bruscas.

Con el uso del algoritmo de Viterbi en la estimación del pitch hemos conseguido evitar los posibles errores de detección del pitch que podrían aparecer en caso de tomar directamente la frecuencia de pitch estimada para cada ventana (independientemente de la información del resto de la señal).

Además, gracias a la información del pitch obtenida a partir de la base de datos de señales de voz grabadas con un electroglotógrafo (que, dadas las características de este instrumento, podemos considerar como la referencia más exacta a la que podemos tener acceso), hemos sido capaces de modificar las probabilidades a priori de los estados del algoritmo de Viterbi al tener en cuenta la probabilidad a priori de la frecuencia de pitch.

De esta manera, y gracias a estas mejoras, el sistema final de detección de la frecuencia de pitch va a ser capaz de ajustarse con gran exactitud a la realidad y, gracias a su implementación con un bajo coste computacional, será capaz de hacerlo

con baja latencia (dependiendo del número de ventanas K fijado para realizar el algoritmo de Viterbi, que hemos fijado en 100).

4.2 Detección facial mediante clustering

El otro objetivo del proyecto era desarrollar una nueva herramienta de detección facial que permita localizar la posición en la que se encuentra la cara del usuario sin dependencia de la inclinación de la cara.

Para evitar esta dependencia con la inclinación de la cara, en lugar de utilizar directamente el algoritmo de Viola-Jones para cada fotograma (que sí es dependiente de la inclinación), hemos utilizado la información proporcionada por el algoritmo de Viola-Jones (las componentes de color de los píxeles detectados como cara por el) para entrenar dos mezclas de Gaussianas: una que responderá a los píxeles de piel (es decir, a la cara) y otra que responderá a los píxeles de "no piel" (es decir, a "no cara").

Una vez entrenadas, obtenemos si cada píxel pertenece (o no) a una cara sí el cociente entre las verosimilitudes obtenidas para él con la mezcla de piel y la mezcla de "no piel" es mayor (o menor) que un umbral (que hemos establecido igual a 1).

Gracias a esta implementación dependiente únicamente del color de cada píxel, conseguimos que la respuesta de nuestro sistema no dependa de la inclinación de la cara (a excepción de en el instante inicial, en el que necesitaremos que la cara no esté inclinada para ser capaces de detectarla mediante el algoritmo de Viola-Jones para poder entrenar las mezclas).

Además, para conseguir evitar la degradación del sistema ante variaciones de luminosidad, hemos actualizado las mezclas de Gaussianas (la media de cada gaussiana) adaptativamente a lo largo del tiempo con la información de los píxeles detectados como cara y "no cara" por el sistema.

Así, dado la señal de vídeo ha sido redimensionada a un tamaño menor (es decir, dado que tenemos menos píxeles que evaluar), este sistema va a conseguir ser capaz de detectar la posición de la cara con un bajo coste computacional y sin importar la inclinación de la misma.

4.3 Líneas futuras

Si bien los dos sistemas creados para este proyecto (tanto el de seguimiento robusto del pitch como el de detección facial) son funcionales y pueden ser directamente

implementados, existen varios mejoras que podrían llevarse a cabo para una mejor respuesta y/o una mejor ejecución de ellos.

Para el sistema de *detección de la frecuencia de pitch*:

- Para conseguir que las probabilidades a priori de la frecuencia de pitch se ajusten aún mejor a la realidad, será necesario obtener mediante el electroglotógrafo una base de datos de señales de voz (de las que podremos asegurar que la frecuencia de pitch estimada es correcta) lo más amplia y variada posible.
  Idealmente, se deberán obtener señales de voz de hombres y mujeres de todos los rangos de edad cuyo número dependerá de la distribución de la población para cada rango de edad.
  Otra posible solución sería obtener esas mismas señales de voz de hombres y mujeres de todos los rangos de edad, pero obtener la probabilidad a priori de la frecuencia de pitch para cada sexo y rango de edad por separado; así, al iniciar el sistema, el logopeda seleccionará los valores de esos parámetros (sexo y rango de edad), lo que permitirá cargar las probabilidades correspondientes.
- Al introducir en el sistema las probabilidades calculadas gracias a la base de datos, hemos sido capaces de limitar el rango en torno al cual la frecuencia de pitch puede ser detectada con mayor probabilidad. Sin embargo, aún es posible que, dentro de ese rango, pudiesen aparecer errores debido a la energía de los armónicos.
  Para evitarlo, sería necesario entrenar un sistema (para el que podría utilizarse la base de datos obtenida mediante el electroglotógrafo, puesto que podemos considerar sus datos correctos) que devolviese el rango en el que debería encontrarse la frecuencia de pitch a detectar, evitando así totalmente la influencia de los armónicos.

Para el sistema de *detección facial*:

- Para agilizar en gran medida el cálculo de la verosimilitud para cada una de las mezclas de Gaussianas de cada píxel, deberá implementarse mediante el uso de la *GPU*, la unidad de procesamiento de gráficos. Mientras que mediante el cálculo típico (en el que se usa la *CPU*, unidad central de procesamiento) las instrucciones se ejecutan secuencialmente (con lo que se evaluará píxel a píxel), la *GPU* tiene la característica de tener un alto grado de paralelismo, con lo que, dado que el tratamiento de cada píxel no depende del resto de píxeles, puede utilizarse para ejecutar en paralelo la evaluación de un conjunto de píxeles, disminuyendo en gran medida el tiempo de ejecución del algoritmo.

Bibliografía

[1] J. E. García Laínez, D. Ribas González, A. Miguel Artiaga, E. Lleida Solano y J. R. Calvo de Lara, "Beam-Search Formant Tracking Algorithm Based on Trajectory Functions for Continuous Speech", Communications Technology Group (GTC), Aragon Institute for Engineering Research (I3A), Advanced Technologies Application Center (CENATAV), 2012.

[2] D. Ribas González, J. E. García Laínez, A. Miguel, A. Ortega Giménez, E. Lleida y J. R. Calvo de Lara, "Evaluation of a New Beam-Search Formant Tracking Algorithm in Noisy Environments", Communications Technology Group (GTC), Aragon Institute for Engineering Research (I3A), Advanced Technologies Application Center (CENATAV), 2012.

[3] W. R. Rodríguez Dueñas, "Aplicación de las Tecnologías del Habla en la Educación de la Voz Infantil Alterada", Tesis Doctoral, Departamento de Ingeniería Electrónica y Comunicaciones, Universidad de Zaragoza, 2010.

[4] R. Lienhart y J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection", Intel Labs, Intel Corporation, 2002.

[5] D.-S. Chen y Z.-K. Liu, "Generalized Haar-Like Features For Fast Face Detection", Dept. of Computer Science Huaqiao University, Dept. of Electrical Eng. and Information Sci. Univ. of Sci. & Tech. of China, 2007.

[6] J. Kovac, P. Peer y F. Solina, "Human Skin Colour Clustering for Face Detection", Faculty of Computer and Information Science University of Ljubljana, 2003.

[7] P. Viola y M. J. Jones, "Robust Real-Time Face Detection", International Journal of Computer Vision 57(2), 137–154, 2004.

[8] Y. Galve, "Diseño de herramientas de asistencia a la logopedia en una plataforma distribuida", Proyecto Fin de Carrera, Departamento Ingeniería Electrónica y Comunicaciones, Universidad de Zaragoza, 2012.

ANEXO I. Entrenamiento de la mezcla de Gaussianas (MoG). Algoritmo EM

Una mezcla de Gaussianas es una superposición lineal de varios modelos de Gaussianas gracias a la que se puede conseguir caracterizar modelos más complejos.

Dada una mezcla de C Gaussianas de D dimensiones, cada una de ellas con media μ_c y covarianza Σ_c y un peso asociado p_c, la verosimilitud de un dato x con esa mezcla será:

$$p(x) = \sum_{c=1}^{C} p_c \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_c|^{1/2}} e^{-\frac{1}{2}(x-\mu_c)^t {\Sigma_c}^{-1}(x-\mu_c)}$$

En general, no dispondremos directamente de esta mezcla o de qué datos corresponden a cada una de las gaussianas; por el contrario, únicamente dispondremos de un conjunto de datos sin etiquetar a los que querremos que la mezcla se adapte lo mejor posible.

Por ello, haremos uso del algoritmo EM para entrenar la mezcla de Gaussianas. Este entrenamiento consiste en un proceso iterativo de dos pasos:

- Primero en el paso E se estima la probabilidad de que los datos pertenezcan a cada una de las gaussianas que componen la mezcla como:

$$\langle \delta_{z_{n,c}} \rangle = \frac{p(x_n, z_n = c)}{p(x_n)}$$

- En el paso M, gracias a las probabilidades calculadas en el paso anterior, disponemos de a qué gaussiana está asociada con mayor probabilidad cada

dato; con esta información, y para cada una de las gaussianas, actualizamos las medias, matrices de covarianzas y pesos con los datos que pertenecen a cada una de ellas como:

$$p_c = \frac{\sum_n \langle \delta_{z_{n,c}} \rangle}{N}$$

$$\mu_c = \frac{\sum_n \langle \delta_{z_{n,c}} \rangle x_n}{\sum_n \langle \delta_{z_{n,c}} \rangle}$$

$$\Sigma_c = \frac{\sum_n \langle \delta_{z_{n,c}} \rangle (x_n - \mu_c)(x_n - \mu_c)^t}{\sum_n \langle \delta_{z_{n,c}} \rangle}$$

Estos dos pasos se repetirán recursivamente un número determinado de veces. Durante las primeras iteraciones nos encontraremos en un estado transitorio en el que la mezcla de gaussianas se actualiza según los datos del entrenamiento, hasta que eventualmente llegué al estado estacionario en el que las sucesivas iteraciones no la modifiquen; en general, este número de iteraciones será a partir del que la mezcla de gaussianas se ajuste lo mejor posible a los datos de entrenamiento.