Laboratorio 2 Procesamiento Digital de Señales

Alejandro Ayala Gil y David Contreras Franco

***Resumen*—En este documento se desarrolla un algoritmo que calcula la Transformada Discreta de Fourier, con el procedimiento de la transformada de una señal particular a mano, y variando el tiempo de muestreo para observar cómo este afecta la transformada al agregar picos de energía en la magnitud y nuevas ondas sierras periódicas en la fase. Además, se realiza un algoritmo para identificar las frecuencias fundamentales del audio de la voz humana por ventanas de tiempo y tres algoritmo para desplazar las frecuencias de dichas ventanas de tiempo para ajustar el "*Pitch*".**

I. Objetivos

1. Implementar un algoritmo que calcule la transformada discreta directa de Fourier.
2. Encontrar la transformada de Fourier de un pulso rectangular.
3. Analizar los efectos en la magnitud y la fase al variar el tiempo de muestreo.
4. Desarrollar un algoritmo que pueda corregir las notas "desafinadas" de la voz de una persona.

II. Materiales

1. Matlab Digital Signal Processing Toolbox

III. Introducción

En este documento se busca implementar un algoritmo de la transformada discreta directa de Fourier, buscando extraer los elementos de esta, como lo son su partes reales e imaginarias, magnitud y fase. Además se desea indagar en el procedimiento matemático que se lleva al usar esta transformada y el comportamiento de la magnitud y la fase a la hora de variar el tiempo de muestreo. Por otra parte, se busca realizar un o una serie de algoritmos que logren corregir el "*Pitch*" de la voz de una persona, ajustando la frecuencia de la voz en las ventanas de tiempo que estén "desafinadas".

IV. Procedimiento

IV-A. Transformada Discreta Directa de Fourier

La transformada discreta directa de Fourier, es un proceso que permite hacer un análisis de señales discretas finitas en su dominio de la frecuencia. Este procedimiento define una función que dependerá de una variable w (denominada como frecuencia) y estará constituida por una sumatoria que se define dentro del dominio de la señal discreta, multiplicada por la señal discreta y una exponencial compleja de la siguiente manera:

$$X(w) = \sum_{n=-\infty}^{\infty} x(n)e^{-jnw} \tag{1}$$

Al poseer una exponencial compleja en la ecuación, se puede ver que la respuesta estará definida en los dominios real e imaginario. Por lo cual, como se solicitó en el laboratorio, se puede replantear la ecuación anterior de tal forma de que exprese su parte real y su parte imaginaria. Esto se puede apreciar en las siguientes ecuaciones:

$$ReX(k) = \sum_{i=0}^{N-1} x(i)cos(\frac{2\pi ki}{N}) \tag{2}$$

$$ImX(k) = \sum_{i=0}^{N-1} x(i)sin(\frac{2\pi ki}{N}) \tag{3}$$

IV-A1. Desarrollo del algoritmo: Considerando las dos ecuaciones anteriores, se procede entonces a implementar un algoritmo, que reciba como parámetros la señal discreta y el número de muestras a considerar. Para la realización de este se hizo uso de un ciclo y de la rutina *sum* de matlab, permitiendo asi retornar estos valores en dos vectores. Además como es solicitado en el laboratorio, también se requiere extraer la magnitud y la fase de la transformada discreta directa de fourier. Por lo que para la magnitud se tiene en consideración la operación a continuación:

$$Mag = \sqrt{Re^2 + Im^2} \tag{4}$$

En cuanto a la fase se hace uso de la rutina de matlab *atan2*, de tal manera que esta permita representar la arcotangente del cociente entre la parte imaginaria de la transformada discreta directa de fourier y la parte real. La función *atan2* permite obtener los resultados entre -pi y pi, que a diferencia de *atan*, los representa entre -pi/2 y pi/2. El proceso llevado acabo se puede ver a continuación:

$$Fase = tan^{-1}(\frac{Im}{Re}) \tag{5}$$

IV-A2. Transformada Discreta Directa de Fourier de un pulso rectangular: Implementado ya el algoritmo de la transformada discreta directa de fourier, se requiere en el laboratorio hacer el procedimiento de la transformada discreta directa de fourier de un pulso rectangular, el cual estará definido desde 0 hasta una constante L. Matemáticamente se plantea de la siguiente manera:

$$x(n) = \begin{cases} 1 & 0 \leq n < L \\ \\ 0 & \text{en otro caso} \end{cases} \tag{6}$$

Ahora, para llevar acabo este procedimiento, se parte de la primera ecuación, tomando la sumatoria desde 0 hasta L - 1 y multiplicándola por la exponencial compleja de la siguiente forma:

$$X(w) = \sum_{n=0}^{L-1} e^{-jnw} \tag{7}$$

Se procede a simplificar la expresión haciendo uso de series geométricas, como se presenta en el siguiente desarrollo:

$$X(w) = 1 + e^{-jw} + e^{-2jw} + e^{-3jw} + ... + e^{-(L-1)jw} \tag{8}$$

$$X(w)e^{-jw} = e^{-jw} + e^{-2jw} + e^{-3jw} + ... + e^{-Ljw} \tag{9}$$

$$X(w) - X(w)e^{-jw} = 1 + e^{-jw} + e^{-2jw} + e^{-3jw} + ...+ e^{-(L-1)jw} - (e^{-jw} + e^{-2jw} + e^{-3jw} + ... + e^{-Ljw}) \tag{10}$$

$$X(w)(1 - e^{-jw}) = 1 - e^{-Ljw} \tag{11}$$

$$X(w) = \frac{1 - e^{-Ljw}}{1 - e^{-jw}} \tag{12}$$

Luego, haciendo uso de factorización e identidades de exponenciales, se obtiene el siguiente resultado:

$$X(w) = \frac{1 - e^{-Ljw}}{e^{\frac{-jw}{2}}(e^{\frac{jw}{2}} - e^{\frac{-jw}{2}})} \tag{13}$$

$$X(w) = \frac{1 - e^{-Ljw}}{e^{\frac{-jw}{2}} 2jsin(\frac{w}{2})} \tag{14}$$

$$X(w) = \frac{e^{\frac{-jwL}{2}}(e^{\frac{jwL}{2}} - e^{\frac{-jwL}{2}})}{e^{\frac{-jw}{2}} 2jsin(\frac{w}{2})} \tag{15}$$

$$X(w) = \frac{e^{\frac{-jwL}{2}} 2jsin(\frac{wL}{2})}{e^{\frac{-jw}{2}} 2jsin(\frac{w}{2})} \tag{16}$$

$$X(w) = \frac{e^{\frac{-jwL}{2}} sin(\frac{wL}{2})}{e^{\frac{-jw}{2}} sin(\frac{w}{2})} \tag{17}$$

De lo cual, se llega al resultado final:

$$X(w) = e^{\frac{-jw(L-1)}{2}} \frac{sin(\frac{wL}{2})}{sin(\frac{w}{2})} \tag{18}$$

IV-A3. Aplicación del algoritmo: En lo que respecta a probar el funcionamiento del código, se requiere en el laboratorio graficar la magnitud y la fase de un pulso rectangular de 41 muestras. Para esto, se realiza un script en el que se crea un vector de unos, haciendo uso de las rutinas de matlab *ones* y *zeros*. Posteriormente se extraen los componentes de magnitud y fase del algoritmo y, haciendo uso de la rutina *linspace*, se generan puntos entre menos pi y pi. Con todo esto se hace uso de la rutina *plot*, para graficar la magnitud y la fase. Como se puede ver en la figura 1, la energía alcanza sus picos máximos en valores cercanos a menos pi y pi. También, se puede apreciar los lóbulos diminutos que se generan en este rango intermedio.

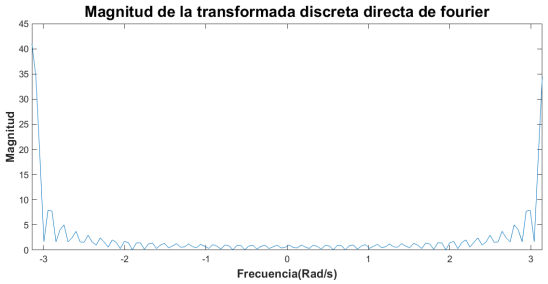

Figura 1. Magnitud de la transformada discreta directa de fourier de un pulso rectangular de 41 muestras.

Por otro lado, para la fase se puede observar en la figura 2 que la energía se encuentra creciendo a medida que se va acercando a pi y esta oscila en forma de onda de sierra.

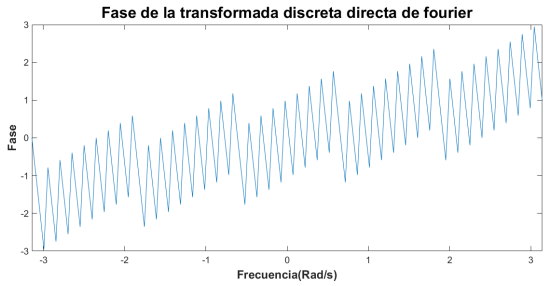

Figura 2. Fase de la transformada discreta directa de fourier de un pulso rectangular de 41 muestras.

Para comprobar que el algoritmo quedó correctamente implementado, se hace uso de la rutina de matlab *fft*. Sin embargo, es necesario utilizar dos rutinas más para lograr extraer del vector resultante la magnitud y la fase, las cuales son *abs* y *angle* respectivamente. El resultado que se obtuvo para el procedimiento se puede observar en la figura 3 y la figura 4, donde se gráfica en el mismo dominio de -pi a pi.

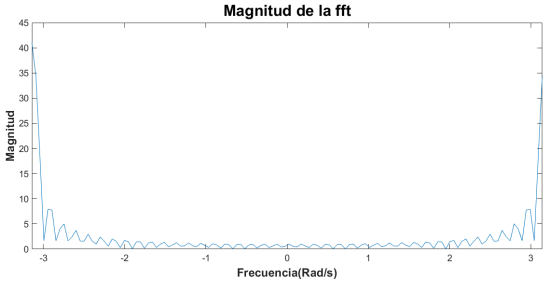

Figura 3. Magnitud de la fft de un pulso rectangular de 41 muestras.

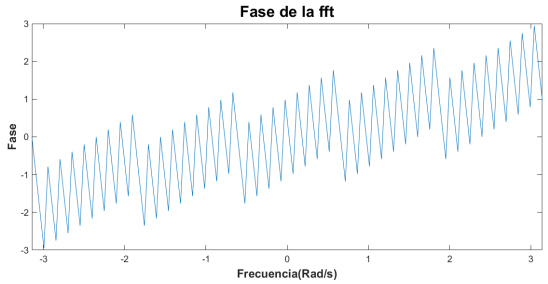

Figura 4. Fase de la fft de un pulso rectangular de 41 muestras.

En cuanto a la parte final del primer punto, se requiere hacer una variación en el tiempo de muestreo de la señal discreta, para asi analizar su comportamiento en magnitud y fase. Por lo tanto, se procede implementando variaciones en el tiempo de muestreo del pulso rectangular de 41 muestras. La primera prueba consiste en aumentar el tiempo de muestreo, donde se establecerán una cantidad de ceros intermedios entre cada muestra, determinados por el tiempo muestreo. Se observa en las figuras 5, 6 y 7, que al variar el tiempo de muestreo aparecen nuevos picos de energía.

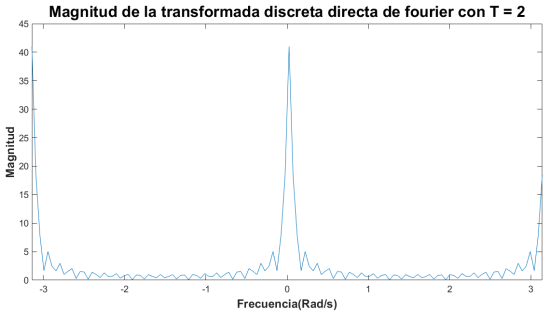

Figura 5. Magnitud de la transformada discreta directa de fourier con tiempo de muestreo 2

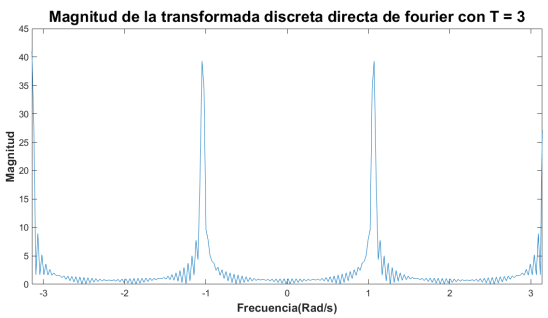

Figura 6. Magnitud de la transformada discreta directa de fourier con tiempo de muestreo 3

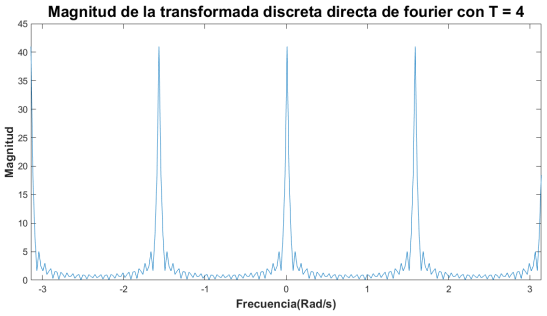

Figura 7. Magnitud de la transformada discreta directa de fourier con tiempo de muestreo 4

En cuanto a la fase, si se aumenta el tiempo de muestreo, se logra observar en las figuras 8, 9 y 10 que esta se comporta de manera periódica a medida que se aumenta el tiempo de muestreo, se ve como la función se repite dentro de todo el intervalo en forma de sierra.

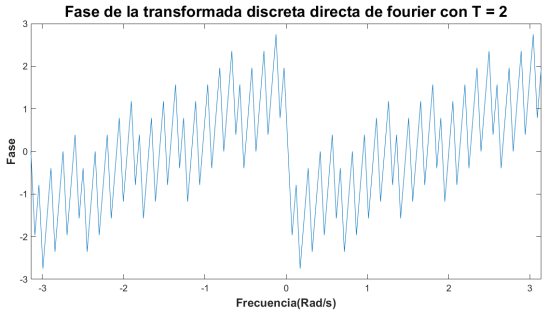

Figura 8. Fase de la transformada discreta directa de fourier con tiempo de muestreo 2

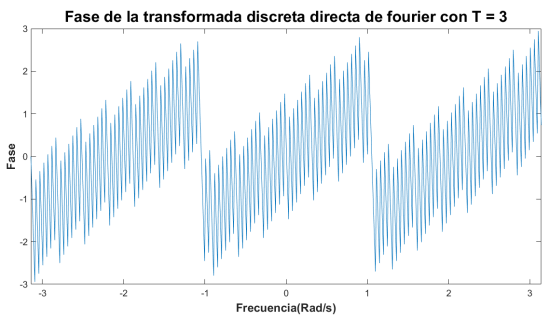

Figura 9. Fase de la transformada discreta directa de fourier con tiempo de muestreo 3

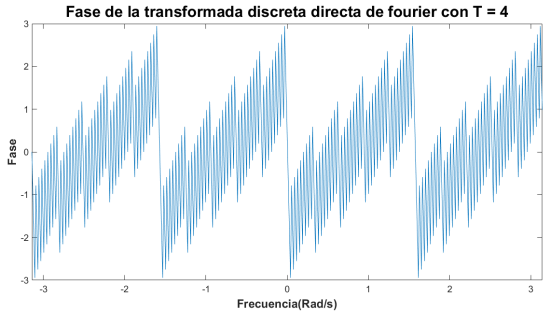

Figura 10. Fase de la transformada discreta directa de fourier con tiempo de muestreo 4

En cuanto a la fase, si se aumenta el tiempo de muestreo, se logra observar en las figuras 8, 9 y 10 que esta se comporta de manera periódica a medida que se aumenta el tiempo de muestreo, se ve como la función se repite en forma de sierra dentro de todo el intervalo.

IV-B. Auto-Tune

Auto-Tune es un software de Antares Audio Technologies que puede alterar el "*Pitch*" de un sonido para "ajustar" el sonido a una nota musical específica. Esto debido a que es difícil reproducir sonido a una nota específica, por ejemplo al cantar. En este orden de ideas, el objetivo es diseñar un algoritmo que pueda tomar fragmentos cortos de audio, alrededor de los 20 segundos, y corregir el "*Pitch*" a lo largo del audio.

Considerando que las diferentes notas musicales son frecuencias fundamentales definidas, con sus respectivos armónicos, la cuestión de alterar o corregir el "*Pitch*" de una persona al cantar se resume en dos aspectos:

- Identificar o estimar el "*Pitch*" durante el audio en cuestión al tiempo. Esto debido a que la nota en la cual canta una persona varía en el tiempo y la capacidad de sostener una nota por un tiempo no es fácil, lo que resulta en pequeñas variaciones de la frecuencia al a cual se desea cantar y que son muy evidentes para el oído humano.
- Desplazar cada "*Pitch*" durante el audio a la frecuencia apropiada. Esto implica que en cada segmento analizado se debe identificar el "*Pitch*" al cual se quiere llegar, asumiendo cierta cercanía él (no se va a cantar en una nota G6 queriendo hacer una B2); además de desplazar todas las frecuencias durante ese segmento a la nota deseada.

Con lo anterior definido es importante aclarar que se trabajará sólo con la voz y los audios en los cuales la voz esté aislada (sin otros instrumentos y buscando la menor cantidad de ruido externo posible) debido a las complicaciones de identificar la frecuencia de la voz rodeada de otras frecuencias dadas.

IV-B1. Identificación del Pitch: El primer componente implica identificar la frecuencia fundamental durante un segmento de tiempo. Para esto existen muchos algoritmos alrededor del concepto de estimación de las frecuencias fundamentales en un tramas de tiempo como el agoritmo YIN [1], Estimador de Máxima Verosimilitud [2][3][4], entre otros ya incluidos en la función Pitch de MATLAB [6] o algoritmos similares ya implementados [5]. A partir de estos, se desarrolló un algoritmo que usa la Short-Time Fourier Transform (STFT), sobre la cual se selecciona el valor mayor y se considera la ubicación de este en las frecuencias como la frecuencia fundamental de esa trama. Adicional se intenta corregir algunos errores que se pueden presentar por sonido de fondo al decidir ignorar todos lo valores de frecuencia identificados que estén fuera del rango de la voz humana, se por silencio u otro factor externo que afecte esa trama de tiempo, esto detallado en el algoritmo 1. El resultado de esta primera parte es un vector que al relacionarlo con el vector de tiempo resultante de la STFT se tienen todas las frecuencias fundamentales en el tiempo.

Algorithm 1: Identificador de Frecuencias Fundamentales en el tiempo

Input: Audio: Vector 1 x N
[S, F, T] = STFT(Audio);
[Value, Index] = Max(S);
Frecuencias Identificadas = F(Index);
Frecuencias Fundamentales = Frecuencias Identificadas($27 < X < 1800$);
Result: Vector de Frecuencias Fundamentales

Se tomó 27 y 1800 ya que este rango incluye los rangos de frecuencias posibles para la voz humana y permite cierto nivel de error en el "*Overlapping*" (5 %). Como ventana de tiempo se tomó 80 ms debido a que es la recomendada por MATLAB en la documentación y la que presentaba un mejor desempeño durante pruebas al igual que un "*Overlapping*" del 50 %. Además, se consideraron todas las frecuencias en este rango que se consideren una nota musical exceptuando los semitonos, con lo cual se puede visualizar respecto a las frecuencias identificadas, qué tan alejadas están de la nota correcta más cercana. El listado de las frecuencias respecto a las notas consideradas se puede observar en la tabla I.

A partir de esto se comparó el algoritmo construido con otros ya implementados; aunque por cuestiones de tiempo y facilidad esta comparación se realizó con los algoritmos propios de MATLAB:

- NCF: Normalized Correlation Function
- PEF: Pitch Estimation Filter
- CEP: Cepstrum Pitch Determination
- LHS: Log-Harmonic Summation
- SRH: Summation of Residual Harmonics

Los cuales se probaron utilizando el archivo FurElise.mat que viene con MATLAB, el cual es un audio de 7 segundos que con tonos simples recreando los primeros segundos de la bagatela "Für Elise" de Ludwig van Beethoven. Este audio se usó para pruebas debido a la simplicidad de sus sonidos y consistencia. Las gráficas de la señal de audio usada, el espectrograma de este audio y los diferentes estimadores de "*Pitch*" se pueden ver a continuación.

Cuadro I
FRECUENCIA DE CADA NOTA

Nota	Frecuencia (Hz)
A6	1760
G6	1568
F6	1397
E6	1319
D6	1175
C6	1047
B5	988
A5	880
G5	784
F5	698
E5	659
D5	587
C5	523
B4	494
A4	440
G4	392
F4	349
E4	330
D4	294
C4	262
B3	247
A3	220
G3	196
F3	175
E3	165
D3	147
C3	131
B2	123
A2	110
G2	98
F2	87
E2	82
D2	73
C2	65
B1	61
A1	55
G1	49
F1	44
E1	41
D1	37
C1	33
B0	31
A0	28

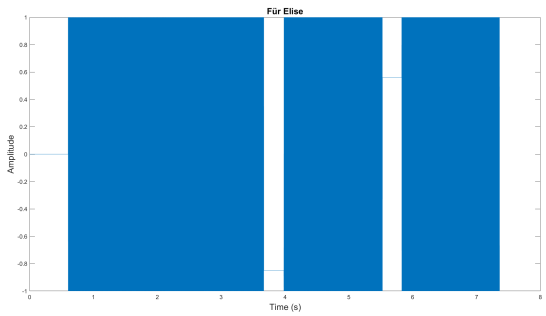

Figura 11. Gráfica del sonido usado de amplitud en cuestión del tiempo.

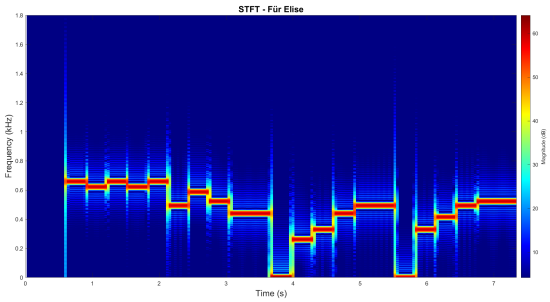

Figura 12. Espectrograma del sonido usado ajustado al rango de frecuencia definido en la tabla I para apreciar las frecuencias fundamentales.

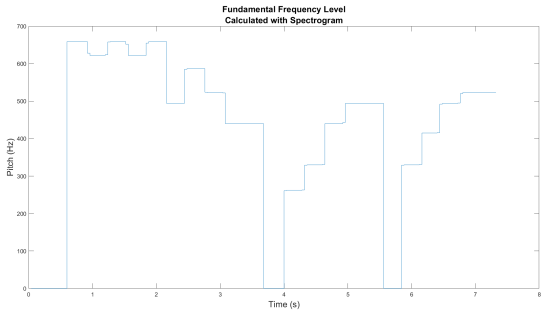

Figura 13. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo diseñado.

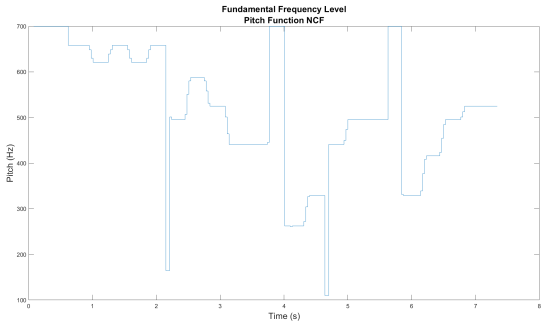

Figura 14. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo NCF de MATLAB.

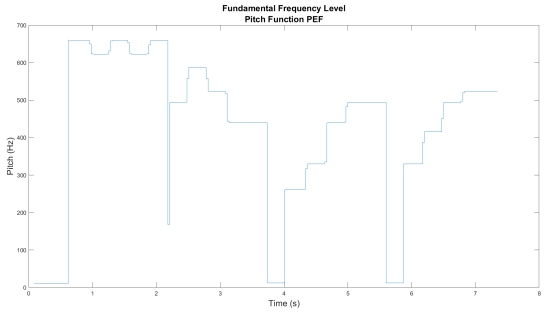

Figura 15. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo PEF de MATLAB.

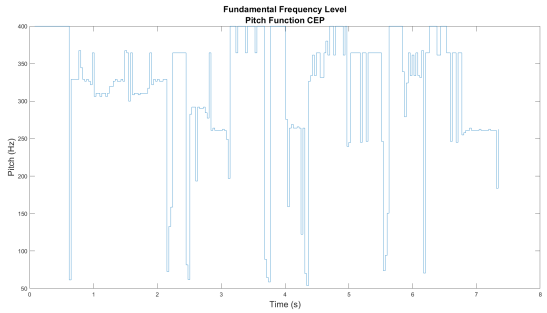

Figura 16. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo CEP de MATLAB.

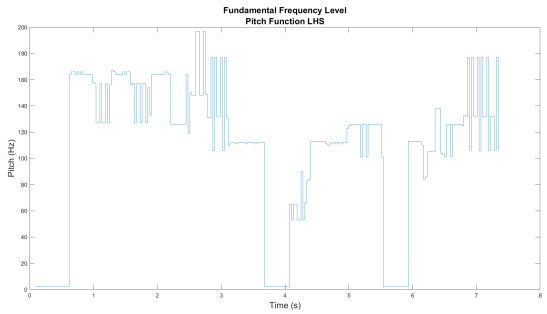

Figura 17. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo LHS de MATLAB.

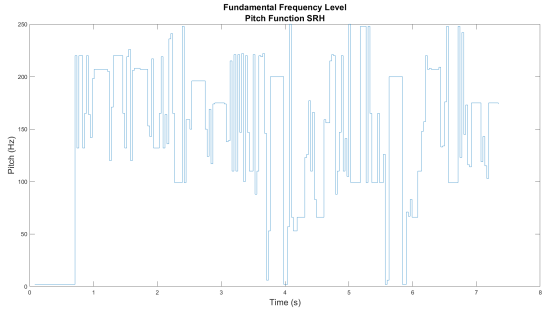

Figura 18. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo SRH de MATLAB.

Analizando las diferentes gráficas se observa que hay alguna similitud entre todas (exceptuando el algoritmo SRH que sin importar los diferentes cambios de configuración mantuvo su comportamiento errático) para reconocer los momentos en los que hay sonido en el audio usado y las variaciones de frecuencia; aunque hay una marcada diferencia en la sensibilidad de algunos algoritmos y el cómo interpretan los cambios de frecuencia como pasar de sonido a silencio. Se evidencia también como el algoritmo diseñado tiene cierta validez al ser bastante símil las frecuencias con mayor cantidad de energía observado en el espectrograma de la figura 12, considerando que el algoritmo diseñado ignora las frecuencias fuera del rango de la voz humana se considera que puede ser usado para el Auto-Tune.

Como comparativo se sigue usando los datos del algoritmo PEF debido a que identifica con mayor estabilidad (menos variación de una ventana de tiempo a otra) las frecuencias y a su alta similitud con el algoritmo diseñado, considerando que este algoritmo no está limitado al rango de frecuencias planteado. Además dentro de la documentación se recomienda el algoritmo PEF como algoritmo con menor error a cambio de mayor costo computacional al compararlo con otros algoritmos; pero debido a que esta no es una aplicación de tiempo real y los audios que se van a trabajar son bastante cortos, es preferible disminuir el error a costa del costo computacional.

Con el algoritmo definido, se pasa a graficar los "*Pitch*" deseados que estén dentro del rango funcional de frecuencias detectado por el algoritmo. Estos "*Pitch*" se pueden graficar como líneas superpuestas a la gráfica de frecuencias fundamentales de la figura 13 y se puede observar en la figura 19, en la cual es evidente que hay cierta diferencia entre la frecuencia identificada por el algoritmo y las frecuencias en las que se debería encontrar según la tabla I. Es importante considerar que los niveles de frecuencia deberían coincidir con los "*Pitch*" graficados. Ya con este se puede proceder al desplazamiento de las frecuencias para ajustar el "*Pitch*", el resultado debería ser la misma gráfica de la figura 19 al aplicarle el algoritmo 1 al nuevo audio pero en la cual las frecuencias fundamentales que antes no coincidían ahora ubicadas en el "*Pitch*" más cercano.

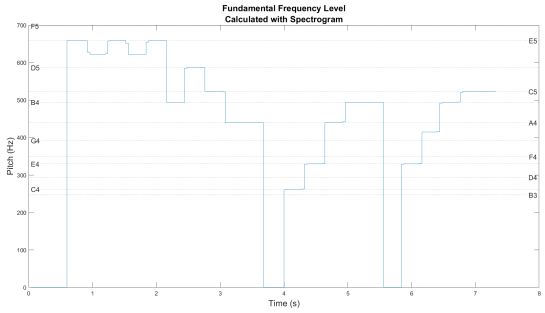

Figura 19. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo usando el algoritmo diseñado con los *Pitch* de la tabla I correspondientes al rango de frecuencias para el audio.

IV-B2. Diferencia de Frecuencia: Para el desplazamiento en frecuencia se realizaron tres algoritmos, cada uno con mejor o peor comportamiento que los otros. Aún así, los tres tienen una fase inicial que calcula la diferencia en frecuencia entre las frecuencias fundamentales por ventana de tiempo identificadas en la primera parte y el "*Pitch*" más cercano de los valores de "*Pitch*" considerados en la tabla I. Este algoritmo consiste en evaluar la diferencia absoluta entre cada de las frecuencias fundamentales con cada uno de los valores de "*Pitch*" y seleccionar el menor valor para cada ventana de tiempo; esto se puede detallar en el algoritmo 2.

Con las diferencias definidas se puede pasar a desplazar acorde a los tres algoritmos definidos.

IV-B3. Desplazamiento en Frecuencia usando la Exponencial Compleja en el Dominio del Tiempo: Como algoritmo inicial se realizó un algoritmo que utiliza la exponencial compleja para desplazar en frecuencia el audio. Esto es permitido debido a que la frecuencia de una señal puede ser desplazada por f si se multiplica la señal en el tiempo por

Algorithm 2: Cálculo de la diferencia en frecuencia entre las frecuencias fundamentales y el *Pitch* más cercano.

Input: Frecuencias Fundamentales: Vector 1 x N
Valores de Pitch: Vector M x 1
Convertir Frecuencias Fundamentales repitiendo el vector M veces: Matriz M x N;
Convertir Valores de *Pitch* repitiendo el vector N veces: Matriz M x N;
Diferencias = Matriz Frecuencias Fundamentales - Matriz Valores de *Pitch*;
[Value, Index] = Min(|Diferencias|);
Diferencia de Frecuencias = Vector de Ceros 1 x N;
for *k = 1:N* **do**
  Diferencia de Frecuencias(k) = Diferencias(Index(k), k)
end
Result: Vector de diferencia en frecuencia por cada ventana de tiempo.

una exponencial compleja. El detalle se puede observar en la siguientes ecuaciones, considerando que ω equivale a $2\pi f$.

$$\frac{1}{2\pi}e^{j\omega_0 t} \xrightarrow{\mathscr{F}} \delta(\omega - \omega_0) \tag{19}$$

$$x(t) \xrightarrow{\mathscr{F}} X(\omega) \tag{20}$$

$$x(t) * \frac{1}{2\pi}e^{j\omega_0 t} \xrightarrow{\mathscr{F}} X(\omega - \omega_0) \tag{21}$$

Lo que significa que conociendo el vector de diferencia de frecuencias para cada ventana de tiempo, se puede construir un vector de exponenciales elevadas a 2π multiplicado por el resultando del producto del vector de diferencia de frecuencias por el instante de tiempo correspondiente a cada diferencia de frecuencia. Es importante aclarar que debido a la STFT cada diferencia de tiempo corresponde a una variedad de instantes de tiempo ya que estas diferencias se tienen por ventanas de tiempo no instantes; considérese instantes en este escenario como las muestras de tiempo correspondientes a la frecuencia de muestreo de la grabación. Para solucionar el problema, se extiende el tamaño del vector de diferencias de frecuencia para que coincida con el vector de tiempo y poder resolver la exponencial usando producto elemento a elemento. El algoritmo resultante se puede detallar en el algoritmo 3, en el cual .* representa la multiplicación por elementos.

Con este algoritmo se realiza el desplazamiento en frecuencia y se vuelve a realiza el análisis del espectrograma con la STFT y el estimador de frecuencias fundamentales para verificar el comportamiento del algoritmo.

Se puede observar tanto en la figura 20 como en la 21 que efectivamente se desplazó la frecuencia que antes no ajustaba a los niveles de "*Pitch*" considerados. Aún así, se evidencia que el desplazamiento no es elegante y se encuentran perturbaciones de dos tipos: en el espectrograma es claro que las transiciones de frecuencia para los cambios de tono se vio afectado y esto se nota escuchando el audio; no es

Algorithm 3: Desplazamiento en frecuencia usando la exponencial compleja en el dominio del tiempo.

Input: Diferencia de Frecuencias: Vector 1 x N
Audio: Vector 1 x M
Frecuencia de Muestreo: Fs
Longitud del Audio = M / Fs;
Vector de Tiempo = M muestras de 0 - Longitud del Audio;
Extender muestras del vector Diferencia de Frecuencias: Vector 1 x M;
Exponencial Compleja = exp(j * 2 * π * Vector Diferencia de Frecuencias Extendido .* Vector de Tiempo);
Audio Desplazado = Audio .* Exponencial Compleja;
Result: Audio desplazado en el tiempo acorde a los desplazamientos calculados por cada ventana de tiempo.

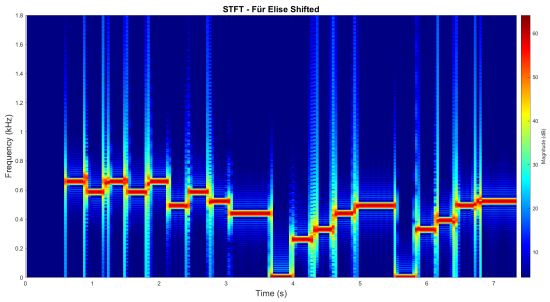

Figura 20. Espectrograma del sonido desplazado en la frecuencia usando el algoritmo 3 ajustado al rango de frecuencia definido en la tabla I para apreciar las frecuencias fundamentales.

consistente el ajuste de frecuencia evidenciado en el estimado de frecuencias fundamentales, esto debido a que el algoritmo 2 no considera contexto para calcular la diferencia y en puntos donde la frecuencia fundamental no tiene definido una más cercana (como en los semitonos no considerados en la tabla I) lo que ocasione que en momentos el "*Pitch*" se pase al tono superior o inferior, ocasionando variaciones fuertes no agradables al momento de escuchar el audio. Un problema adicional no observado en las gráficas ya que estas pueden

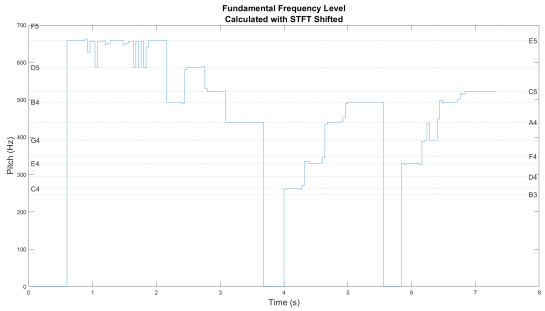

Figura 21. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia usando el algoritmo 3 con los *Pitch* de la tabla I correspondientes al rango de frecuencias para el audio.

procesar los datos reales o complejos sin problema, es el hecho de que al momento de reproducir el audio este sólo recibe valores reales. Esto implica que es necesario considerar sólo la parte real del audio con frecuencias desplazadas y esto equivale a no haber multiplicado por una exponencial compleja sino por un coseno, por lo que el audio no refleja el desplazamiento calculado.

IV-B4. Desplazamiento en Frecuencia usando la Exponencial Compleja en el Dominio de la Frecuencia: Como consideración adicional y para evitar perder datos de frecuencia al sólo considerar la exponencial compleja en las ventanas de tiempo de forma completa y sin el *Overlapping* considerado al momento de realizar la STFT, se decide desplazar las frecuencias con la exponencial compleja pero en el dominio de la frecuencia a diferencia del dominio de tiempo, eso implica que al realizar la STFT Inversa, se considera el *Overlapping* y se pierde menos información de frecuencia evidenciado con el algortimo 3 en la figura 20. El algoritmo 4 detalla las variaciones al algoritmo 3 para incluir las consideraciones explicadas.

Algorithm 4: Desplazamiento en frecuencia usando la exponencial compleja en el dominio de la frecuencia.

Input: Diferencia de Frecuencias: Vector 1 x N
Audio: Vector 1 x M
Frecuencia de Muestreo: Fs
Longitud del Audio = M / Fs;
Vector de Tiempo = M muestras de 0 - Longitud del Audio;
Extender muestras del vector Diferencia de Frecuencias: Vector 1 x M;
Exponencial Compleja = exp(j * 2 * π * Vector Diferencia de Frecuencias Extendido .* Vector de Tiempo);
STFT Audio = STFT(Audio);
STFT Exponencial Compleja = STFT(Exponencial Compleja);
Frecuencias Desplazadas = Convolución(STFT Audio, STFT Exponencial Compleja);
Audio Desplazado = ISTFT(Frecuencias Desplazadas);
Result: Audio desplazado en el tiempo acorde a los desplazamientos calculados por cada ventana de tiempo.

El problema evidente de este algoritmo es la complejidad computacional requerida, debido a que es una convolución en dos dimensiones. Lo que implica que cualquier audio de longitud de más de uno o dos segundos a una frecuencia de muestreo que intente incluir el rango de la voz humana ya representa una cantidad masiva de datos que requiere mucho tiempo para ser procesada. Por lo tanto, se plantea este algoritmo en búsqueda de resolver uno de los problemas encontrados en el algoritmo 3 pero no se puede evidenciar su funcionamiento debido a la complejidad computacional del audio y la frecuencia de muestreo con la que se había probado el algoritmo anterior; en este orden de ideas se podría ajustar la longitud o frecuencia de muestreo, pero esto sólo evidencia la no viabilidad de este algoritmo.

IV-B5. Desplazamiento forzoso: Se le da el nombre del algoritmo debido a que no se desarrolla un análisis matemático, sólo un desplazamiento directo de las columnas (frecuencia) de la STFT. Debido a esto no se utilizan los algoritmo 1 y 2, debido a que el proceso de desplazamiento es a la par del cálculo de la diferencia en frecuencia y directamente con los datos obtenidos en la STFT; por eso, se considera que este algoritmo es la unión del algoritmo 1 y 2 agregando el cálculo de la STFT Inversa al finalizar el corrimiento.

Algorithm 5: Desplazamiento de frecuencias por ventanas de tiempo de un audio usando corrimiento forzoso

Input: Audio
Valores de *Pitch*: Vector M x 1;
[S, F, T] = STFT(Audio);
[Value, Index] = Max(S);
Frecuencias Identificadas = F(Index);
Frecuencias Fundamentales = Frecuencias Identificadas($27 < X < 1800$);
Frecuencias Fundamentales: Vector 1 x N Convertir Frecuencias Fundamentales repitiendo el vector M veces: Matriz M x N;
Convertir Valores de *Pitch* repitiendo el vector N veces: Matriz M x N;
Diferencias = Matriz Frecuencias Fundamentales - Matriz Valores de *Pitch*;
[Value, Index] = Min(|Diferencias|);
Frecuencias Desplazadas = S;
for *k = 1:N* **do**
  Frecuencias Desplazadas(:, k) = circshift(S(:, k), Diferencias(Index(k), k))
end
Sonido Desplazado = ISTFT(Frecuencias Desplazadas);
Result: Audio con las frecuencias desplazadas por ventanas de tiempo.

Al utilizar este algoritmo en el mismo audio probado hasta, se detalla en la figura 22 que no se presenta el error observado con el algoritmo 3 puesto que los cambios de frecuencia ya no son tan bruscos aunque la banda es menos definida al espectrograma original. Adicionalmente, en la figura 23 se observa que es más definido el desplazamiento hacia las notas definidas en la tabla I. En general es mucho más acertado este algoritmo a los dos anteriores, por su resultado en frecuencia respecto al algoritmo 3 y por su viabilidad computacional respecto al algoritmo 4. Aún así, este algoritmo entrega valores complejos que deben ser limitados a valores reales al momento de reproducirlos, preferiblemente considerando la parte real únicamente no la magnitud; esto indica que el algoritmo no es perfecto y hay otras alternativas que pueden tener mejor resultado.

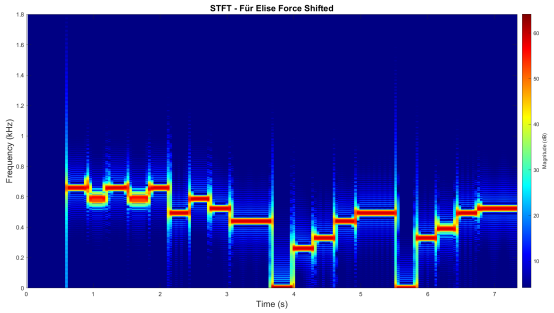

Figura 22. Espectrograma del sonido desplazado en la frecuencia usando el algoritmo 5 ajustado al rango de frecuencia definido en la tabla I para apreciar las frecuencias fundamentales.

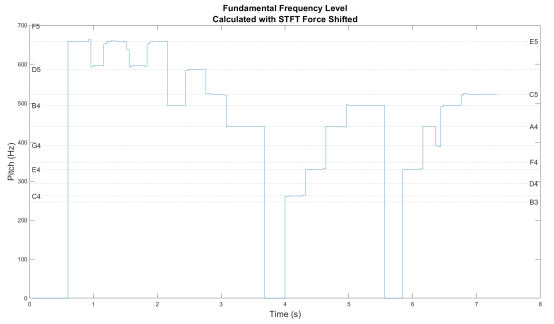

Figura 23. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia usando el algoritmo 5 con los *Pitch* de la tabla I correspondientes al rango de frecuencias para el audio.

En resumen, se desarrollaron tres algoritmos con diferentes niveles de certitud respecto al objetivo planteado. Aunque los algoritmos no son perfectos cumplen parcialmente su función en desplazar las frecuencias "desafinadas" de un audio. Adicional a los algoritmos desarrollados, se plantea un algoritmo adicional a partir de lo propuesto en la documentación de MATLAB [18].

IV-B6. Algoritmo alternativo propuesto: El objetivo en estos algoritmos era poder generar variaciones de frecuencia sin afectar el tiempo y en todos los algoritmos planteados se logró no afectar el tiempo del audio original, aún así no son perfectos los algoritmos planteados y es aquí donde se propone un algoritmo más simple: en la documentación de MATLAB se expresa cómo se puede variar el "*Pitch*" de la voz de un audio al realizar la STFT y la ISTFT en anchos de ventana de tiempo distinto generando una variación en la frecuencia; adicionalmente, si se modifica la frecuencia de muestreo considerada para el audio a partir de la diferencia de de los anchos de ventanas entre la transformada y la transformada inversa se puede variar el "*Pitch*" sin variar la duración del audio. Implementar esto considerando cada diferencia de ancho de ventana con lo requerido de diferencia de frecuencia para ajustar el "*Pitch*" puede ser una solución más completa a este problema. Lastimosamente por cuestiones de tiempo y complejidad del problema este algoritmo no se alcanzó a terminar de desarrollar e implementar/probar.

IV-B7. Pruebas con Voces Autores: Finalmente, se prueban los algoritmos con las voces de los autores:

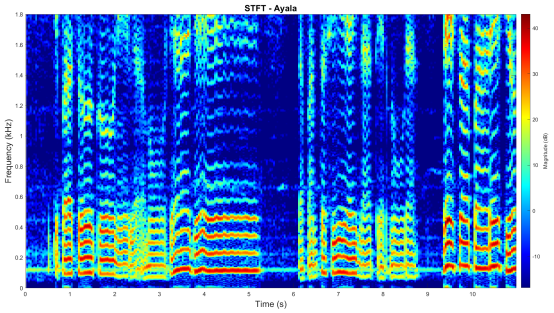

Figura 24. Espectrograma de la voz de Alejandro Ayala

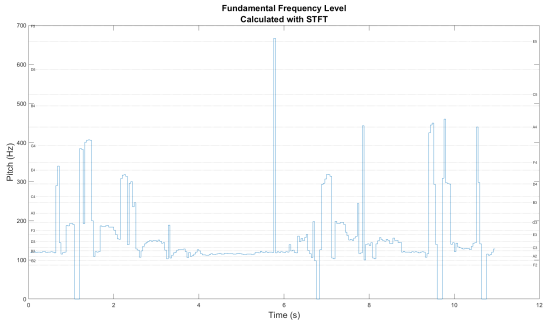

Figura 25. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia de la voz de Alejandro Ayala

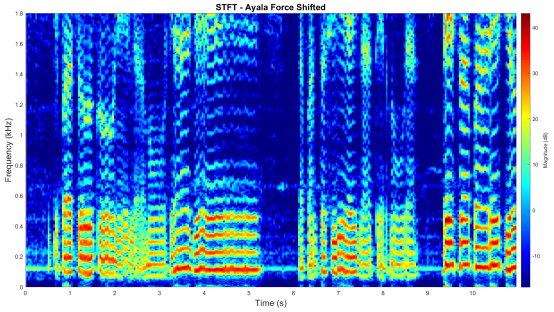

Figura 26. Espectrograma de la voz de Alejandro Ayala desplazado usando el algoritmo 5.

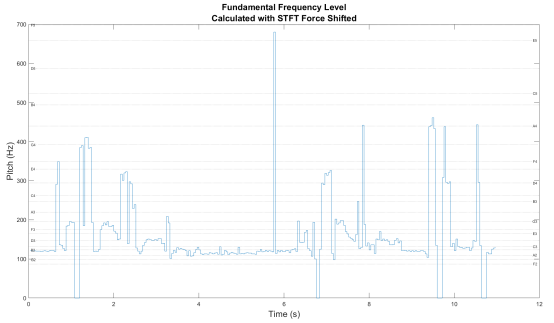

Figura 27. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia de la voz de Alejandro Ayala desplazado usando el algoritmo 5.

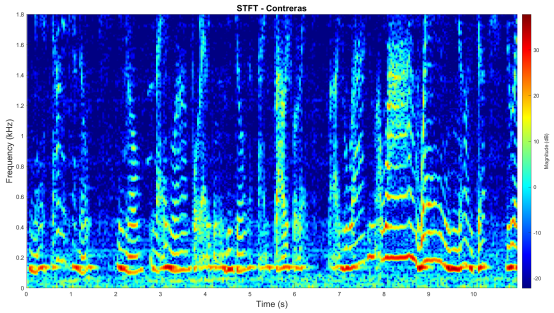

Figura 28. Espectrograma de la voz de David Contreras

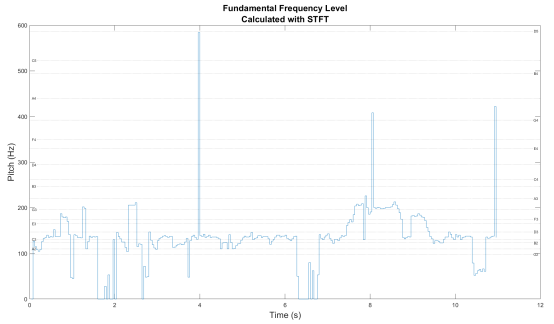

Figura 29. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia de la voz de David Contreras

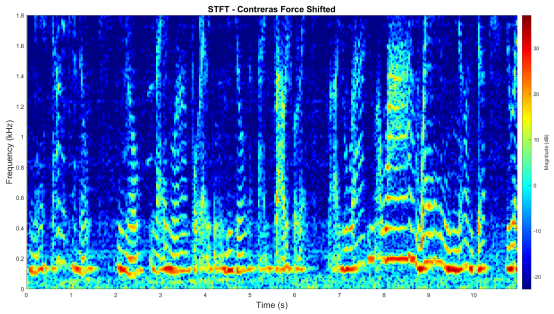

Figura 30. Espectrograma de la voz de David Contreras desplazado usando el algoritmo 5.

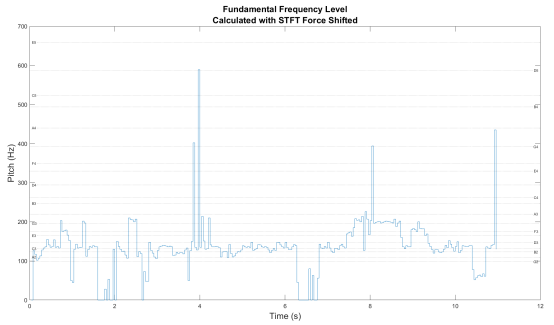

Figura 31. Gráfica de las frecuencias fundamentales identificadas en cuestión del tiempo del sonido desplazado en la frecuencia de la voz de David Contreras desplazado usando el algoritmo 5.

Aunque los cambios son sutiles ya que en ambos casos se cubre un rango más grande de frecuencias, se puede observar al igual que en el audio de prueba el funcionamiento del algoritmo.

V. Conclusiones

1. Debido a que el índice de los vectores de matlab comienzan en uno, se tuvo cierta complicación a la hora de implementar las ecuaciones que describen la parte real e imaginaria de la transformada discreta directa de fourier.
2. Al hacerse el desarrollo matemático de la transformada discreta directa de fourier, se pudo apreciar que la explicación de la serie geométrica usada no es compleja. Sin embargo, deducir el procedimiento detrás de esta serie geométrica se convierte en algo no tan visible.
3. Cuando se estuvo desarrollando el algoritmo de la transformada discreta directa de fourier, se llegaba a resultados similares en cuanto a la magnitud al compararlo con la rutina *fft*. Para la fase no se tuvo el resultado esperado, sin embargo, se encontró una solución al investigar el funcionamiento de las rutinas *atan* y *atan2* de matlab.
4. Uno de los criterios que se tuvo en cuenta para la creación del vector que representa al pulso rectangular, fue hacerlo en potencias de 2. Esto con el fin de hacerlo compatible con la rutina *fft* y, como se menciono en clase, entre más puntos tenga la transformada mejor es su habilidad para separar componentes de frecuencia cercanos.
5. Desarrollar un algoritmo “Autotunner” es una tarea tan compleja como el objetivo que se desee plantear y entre más automático y acertado intente ser la solución más complejo es el algoritmo.
6. Existen multiples acercamientos para la solución del problema del ajuste de “*Pitch*”, cada uno con sus ventajas y desventajas.

VI. Observaciones

1. Se puede observar para la magnitud de la transformada discreta directa de fourier, que se generan unos lóbulos dentro de todo el intervalo. Esto se debe al fenómeno de Gibbs, ya que el espectro utilizado posee discontinuidades fuertes.
2. En cuanto a la magnitud, se puede ver que los picos de energía se van haciendo cada vez más delgados al aumentar el tiempo de muestreo de la señal.
3. Al hacer uso del script que permite aumentar el tiempo de muestreo, se pudo visualizar que los picos de energía que aparecen centrados en el intervalo son, en cantidad, una unidad menos al tiempo de muestreo.
4. Visualizando la fase y la magnitud de manera simultanea, se puede deducir que el comportamiento donde empieza un ciclo de la señal sierra en la fase, se da en el mismo punto donde se origina un pico de energía en la magnitud.
5. Una de las dificultades encontradas para desarrollar el algoritmo de AutoTune es el alcance a definir, qué tan

acertado y automático ha de ser este algoritmo. Por ejemplo, sería interesante agregar más semitonos y sobretodo diseñar algoritmos que tengan más "contexto" al momento de desarrollar el desplazamiento; esto significa saber la base musical sobre la cual se está cantando e identificar mejor si una nota está en un punto para mantenerla o cambiar en vez de buscar ciegamente el "*Pitch*" más acercado.

REFERENCIAS

[1] A. de Cheveigne and H. Kawahara: YIN, an F0 estimator, Paris, France, 2002. Available: http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf. [Accessed: 11- April- 2020].

[2] J. K. Nielsen, T. L. Jensen, J. R. Jensen, M. G. Christensen and S. H. Jensen, .A FAST ALGORITHM FOR MAXIMUM LIKELIHOOD-BASED FUNDAMENTAL FREQUENCY ESTIMATION ", 23rd European Signal Processing Conference (EUSIPCO). Available: https://www.eurasip.org/Proceedings/Eusipco/Eusipco2015/papers/1570101959.pdf. [Accessed: 11- April- 2020].

[3] Decision and Control including the 16th Symposium on Adaptive Processes and A Special Symposium on Fuzzy Set Theory and Applications, 1977 IEEE Conference onVolume: 16. Available: https://www.researchgate.net/publication/224680938_Maximum_likelihood_pitch_estimation. [Accessed: 11- April- 2020].

[4] V. Mahadevan and C. Y.Espy-Wilson, "Maximum Likelihood Pitch Estimation Using Sinusoidal Modeling ", conference mahadevan ICCSP, 2011. Available: https://cpb-us-e1.wpmucdn.com/blog.umd.edu/dist/c/619/files/2019/11/conference_mahadevan_ICCSP_2011.pdf. [Accessed: 11- April- 2020].

[5] .orchidas/Pitch-Tracking", GitHub, 2020. [Online]. Available: https://github.com/orchidas/Pitch-Tracking. [Accessed: 11- Apr- 2020].

[6] .Estimate fundamental frequency of audio signal - MATLAB pitch", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/audio/ref/pitch.html. [Accessed: 11- Apr- 2020].

[7] B. Shafran, J. Miller and B. Ikei "PITCH CORRECTION TOOL IN MATLAB", University of Rochester, 2019. Available: http://www2.ece.rochester.edu/ zduan/teaching/ece472/projects/2018/BenjaminShafran_JoshuaMiller_BrentIkei_PitchCorrection_FinalReport.pdf [Accessed: 11- Apr- 2020].

[8] S. G. Miller, "How Did an Opera Singer Hit the Highest Note Ever Sung at the Met?", LiveScience, Nov 2017. Available: https://www.livescience.com/60993-how-to-sing-the-highest-notes.html [Accessed: 11- Apr- 2020].

[9] E. Sengpiel, "Note names of musical notes keyboard piano frequencies = octave piano keys number tone tones 88 notes frequency names of all keys on a grand piano standard concert pitch tuning German English system MIDI 88 - sengpielaudio Sengpiel Berlin", Sengpielaudio.com, 2020. [Online]. Available: http://www.sengpielaudio.com/calculator-notenames.htm. [Accessed: 11- Apr- 2020].

[10] "Inverse tangent in radians - MATLAB atan", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/atan.html. [Accessed: 11- Apr- 2020].

[11] "Four-quadrant inverse tangent - MATLAB atan2", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/atan2.html. [Accessed: 11- Apr- 2020].

[12] "Fast Fourier transform - MATLAB fft", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/fft.html. [Accessed: 11- Apr- 2020].

[13] "Create array of all ones - MATLAB ones", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/ones.html. [Accessed: 11- Apr- 2020].

[14] "Create array of all zeros - MATLAB zeros", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/zeros.html [Accessed: 17- Feb- 2020].

[15] "2-D line plot - MATLAB plot", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/plot.html. [Accessed: 11- Apr- 2020].

[16] "Absolute value and complex magnitude - MATLAB abs", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/abs.html. [Accessed: 11- Apr- 2020].

[17] "Phase angle - MATLAB angle", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/angle.html. [Accessed: 11- Apr- 2020].

[18] "Inverse short-time Fourier transform - MATLAB istft", Mathworks.com, 2020. [Online]. Available: https://www.mathworks.com/help/signal/ref/istft.html. [Accessed: 11- Apr- 2020].

[19] Apuntes de clase de procesamiento digital de señales. Ingeniería Electrónica y ciencias de la computación. Pontificia Universidad Javeriana Cali. Marzo 2020.