TÌM TẦN SỐ CƠ BẢN CỦA TÍN HIỆU TRÊN MIỀN THỜI GIAN DÙNG HÀM TỰ TƯƠNG QUAN, HÀM VI SAI BIÊN ĐỘ TRUNG BÌNH VÀ TRÊN MIỀN TẦN SỐ DÙNG PHÉP BIẾN ĐỔI FOURIER NHANH.

Đàm Quang Tiến, Nguyễn Nghĩa Thịnh, Trần Giang Phong

Nhóm 10, lớp HP: 1022103.2010.18.10

Điểm	**Bảng phân công nhiệm vụ**		**Chữ ký của SV**
	Đàm Quang Tiến (nhóm trưởng)	Phân công nhiệm vụ và đôn đốc tiến độ của mỗi thành viên. Đọc tài liệu, cài đặt và viết báo cáo về **thuật toán tự động tính F0 trên miền tần số** (áp dụng thuật toán FFT), làm slide và thuyết trình PowerPoint. Duyệt lại báo cáo, slide của nhóm.	
	Nguyễn Nghĩa Thịnh	Đọc tài liệu, cài đặt và viết báo cáo về **thuật toán tự động tính F0 trên miền thời gian dùng hàm vi sai biên độ trung bình (AMDF function),** làm slide và thuyết trình PowerPoint.	
	Trần Giang Phong	Đọc tài liệu và viết báo cáo về **thuật toán tự động tính F0 trên miền thời gian dùng hàm tự tương quan (autocorrelation function)**, làm slide và thuyết trình PowerPoint.	

Lời cam đoan: Chúng tôi, gồm các sinh viên có chữ ký ở trên, cam đoan rằng báo cáo này là do chúng tôi tự viết dựa trên các tài liệu tham khảo liệt kê ở cuối báo cáo. Các số liệu thực nghiệm và mã nguồn chương trình nếu không chỉ dẫn nguồn tham khảo đều do chúng tôi tự làm. Nếu vi phạm thì chúng tôi xin chịu trách nhiệm và tuân theo xử lý của giáo viên hướng dẫn.

TÓM TẮT— Tìm tần số cơ bản của tín hiệu để phát hiện các đặc tính cơ bản của các thanh âm người nói. Ở đây chúng tôi xét các âm thanh trong hai môi trường, hai người nói với giới tính khác nhau để kiểm tra tính chính xác của thuật toán khác nhau khi dùng để phát hiện cao độ của âm thanh. Kết quả thực nghiệm cũng cho thấy rằng chúng ta có thể sử dụng các thuật toán trên miền thời gian và miền tần số để phát hiện tần số cơ bản với sai số tùy vào điều kiện đầu vào và tùy chỉnh các thông số của các thuật toán để tinh chỉnh.

Từ khóa— Tính tần số cơ bản (Fundamental Frequency), Hàm tự tương quan trên miền thời gian (Autocorrelation Function), Hàm vi sai biên độ trung bình (Average Magnitude Different Function), Giảm bậc hài của phổ trên miền tần số (Harmonic Product Spectrum on frequency domain).

Mục lục bài báo cáo

I. ĐẶT VẤN ĐỀ

Một trong những công việc quan trọng của việc xử lý ngôn ngữ tự nhiên chính là xử lý tiếng nói, công cụ trao đổi trực tiếp chính của con người. Có hai cách để xử lý tiếng nói là thông qua việc xử lý tín hiệu hoặc thông qua hệ thống mạng neuron máy tính, tuy vậy hiệu suất xử lý của hệ thống xử lí tín hiệu là tốt hơn rất nhiều – không yêu cầu nhiều sức mạnh tính toán. Đặc tính phát âm của con người có thể được phát hiện thông qua việc phát hiện tần số cơ bản của âm thanh đó (fundamental frequency). Sự biến đổi của tần số cơ bản có thể giúp máy tính phán đoán con người đang phát âm thanh điệu nào của ngôn ngữ, ở đây chúng tôi xét tiếng việt, qua đó giúp tăng chất lượng giọng nói, tổng hợp tiếng nói. Có nhiều phương pháp khác nhau để xác định tần số cơ bản như AMDF, tự tương quan, giảm bậc hài phổ trên miền thời gian, fourier nhanh,…

Tần số cơ bản - cao độ - f0 của một tín hiệu tuần hoàn chính là nghịch đảo của chu kỳ cơ bản của tín hiệu. Trong đó, chu kỳ cơ bản là khoảng thời gian nhỏ nhất mà tín hiệu tuần hoàn trên miền thời gian. Tần số cơ bản mang thông tin có ý nghĩa vật lý đặc trưng cho tín hiệu như đặc tính của nguồn phát. Có hai phía tiếp cận để tính toán tần số cơ bản là tiếp cận theo miền thời gian và theo miền tần số. Trong báo cáo này, hàm tự tương quan và hàm vi sai biên độ trung bình được dùng để tìm tần số cơ bản trên miền thời gian, dùng phép biến đổi Fourier nhanh (FFT) để phân tích phổ và tính tần số cơ bản trên miền tần số.

Bài báo cáo có bố cục như sau. Phần II trình bày về cơ sở lý thuyết của các thuật toán và vấn đề liên quan đến việc tính toán tần số cơ bản trên miền thời gian. miền tần số. Phần III trình bày mã nguồn cài đặt các thuật toán. Phần IV trình bày kết quả thực nghiệm mô tả dữ liệu dùng để đánh giá độ chính xác, đưa ra các đánh giá. Phần V trình bày kết luận. Phần VI trình bày phần tài liệu tham khảo đã dùng.

II. LÝ THUYẾT XỬ LÝ TÍN HIỆU TIẾNG NÓI VÀ CÁC THUẬT TOÁN TÍNH TẦN SỐ CƠ BẢN TRÊN MIỀN THỜI GIAN VÀ TRÊN MIỀN TẦN SỐ

A. Vấn đề cần giải quyết

Tần số cơ bản của tín hiệu có thể được tìm trên miền thời gian hoặc trên miền tần số.

Trên miền thời gian, chúng tôi sử dụng hai thuật toán sử dụng "Hàm tự tương quan" và "Hàm vi sai biên độ trung bình" để từ đó phát hiện được các đặc tính tuần hoàn của tín hiệu và từ đó xác định tần số cơ bản của tín hiệu.

Trên miền tần số, chúng tôi sử dụng phép biến đổi Fourier nhanh để chuyển tín hiệu từ miền thời gian sang miền tần số, sau đó giảm bậc của kết quả để tính tần số cơ bản của tín hiệu.

B. Hàm tự tương quan để tính tần số cơ bản trên miền thời gian

1. Cơ sở lý thuyết

Trong xử lý tín hiệu số nói chung và xử lý tín hiệu tiếng nói nói riêng, hàm tự tương quan dùng để biến đổi tín hiệu tuần hoàn thành một tín hiệu tuần hoàn khác có các điểm cực đại có thể xác định được dễ dàng, nhờ đó ứng dụng để xác định chu kỳ cơ bản T0 và tần số cơ bản F0.

Hàm tự tương quan của tín hiệu được xác định bởi công thức **:**

$$r_{xx}(l) = \lim_{N\to\infty} \frac{1}{(2N+1)} \sum_{n=-N}^{N} x(n)x(n+l)$$

trong đó: $r_{xx}(l)$ là giá trị hàm tự tương quan theo độ trễ l, $(2N+1)$ là độ dài khung tín hiệu, $x(n)$ là biên độ tín hiệu tại thời điểm n.

Hàm tự tương quan có các tính chất sau:

- Là một hàm chẵn: $r_{xx}(l) = r_{xx}(-l)$

- Đạt giá trị cực đại tại $l = 0$: $|r_{xx}(l)| \leq r_{xx}(0)$ với mọi l;

- Đại lượng $r_{xx}(0)$ bằng năng lượng của tín hiệu tiếng nói.

Khi xử lý tín hiệu dùng kỹ thuật xử lý ngắn hạn (phần 1.4), ta chia tín hiệu tiếng nói thành các khung tín hiệu có độ dài hữu hạn và công thức tự tương quan trở thành :

$$r_t(\tau) = \sum_{j=t+1}^{t+W} X_j X_{j+\tau}$$

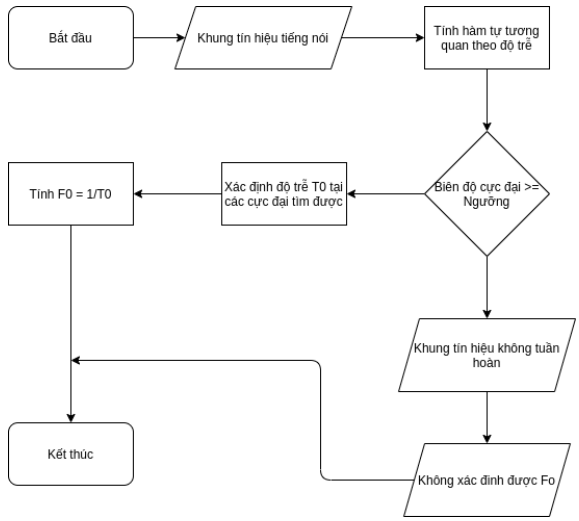
trong đó X_j là biên độ tín hiệu tại thời điểm j, $r_t(\tau)$ là giá trị của hàm tự tương quan theo độ trễ τ tại khung tín hiệu t, và W là độ dài của khung tín hiệu.

Nếu $T0$ là chu kỳ cơ bản của tín hiệu tuần hoàn, khi đó các giá trị độ trễ: 0, $\pm T0$, $\pm 2T0$,... sẽ là các điểm mà hàm tự tương quan đạt cực đại cục bộ. Đây là ý tưởng chính để xác định F0 của tín hiệu tiếng nói bằng hàm tự tương quan.

Âm hữu thanh (voiced speech) là âm phát ra có thanh, ví dụ như các nguyên âm /a/, /e/, /i/, /o/, /u/ hoặc các phụ âm như /m/, /n/, /l/. Trong xử lý tín hiệu tiếng nói, âm hữu thanh gồm các khung tín hiệu tuần hoàn nên có thể tính được tần số cơ bản F0.

Âm vô thanh (non-voiced speech) là âm khi tạo ra tiếng thì dây thanh không rung hoặc rung đôi chút tạo ra giọng như giọng thở, ví dụ như /t/, /p/ hay /k/. Trong xử lý tín hiệu tiếng nói, âm vô thanh không có ích khi tính tần số cơ bản. Vì âm vô thanh không có khung tín hiệu tuần hoàn. Tần số cơ bản ở âm vô thanh là không xác định.

2. Sơ đồ khối



Hình 1. Sơ đồ khối thuật toán tìm tần số cơ bản trên miền thời gian sử dụng hàm tự tương quan

3. Các tham số quan trọng của thuật toán

Các tham số quan trọng ảnh hưởng nhiều đến độ chính xác của thuật toán tự tương quan, đó là độ dài khung tín hiệu và ngưỡng xác định hữu thanh/vô thanh

a) Độ dài khung tín hiệu

Thuật toán tính F0 dựa trên hàm tự tương quan có sử dụng kỹ thuật xử lý ngắn hạn (phân tín hiệu thành nhiều khung nhỏ) nên việc chọn loại cửa sổ và độ dài cửa sổ thích hợp là quan trọng. Có rất nhiều hàm cửa sổ có thể được sử dụng trong kỹ thuật xử lý tín hiệu ngắn hạn: Hamming, Hanning, Blackman, tam giác, chữ nhật .

b) Ngưỡng xác định hữu thanh/vô thanh

Trong thuật toán tìm F0 dùng hàm tự tương quan, có một tham số quan trọng nữa đó là ngưỡng để xác định một khung tín hiệu là của âm hữu thanh hay của âm vô thanh. Việc tăng hoặc giảm ngưỡng này ảnh hưởng đến việc xác định âm hữu thanh hoặc âm vô thanh của đoạn tín hiệu tiếng nói. Nếu ngưỡng này đặt ra là quá thấp, khi tính F0, các khung tín hiệu vô thanh sẽ bị nhầm thành các khung tín hiệu hữu thanh. Nếu ngưỡng này đặt ra là quá cao, khi tính F0, các khung tín hiệu hữu thanh sẽ bị nhầm thành các khung tín hiệu vô thanh.

4. Các vấn đề và giải pháp khắc phục

Thuật toán tự tương quan tương đối ít bị ảnh hưởng bởi nhiễu, nhưng lại rất nhạy cảm với tần số lấy mẫu. Bởi vì nó tính tần số cơ bản trực tiếp từ việc trượt mẫu, và sự nhạy cảm này có ảnh hưởng như sau, nếu chúng ta có tần số lấy mẫu thấp thì sẽ thu được tần số cơ bản thấp hơn.

Bên cạnh đó, hàm tự tương quan yêu cầu một lượng lớn các phép tính. Tuy nhiên, việc này có thể áp dụng kĩ thuật thích ứng thì số lượng các phép tính có thể suy biến và chạy trong thời gian gần-thời-gian-thực.

C. Hàm vi sai biên độ trung bình để tính tần số cơ bản trên miền thời gian

1. Cơ sở lý thuyết

Phát hiện chu kỳ tín hiệu trong nền nhiễu có một vị trí rất quan trọng trong nhiều ứng dụng kỹ thuật, chẳng hạn như phát hiện cao độ trong xử lý giọng nói, chu kỳ rung của trục lăn khai thác lỗi va đập trong kỹ thuật cơ khí, điều kiện rung của stato động cơ và phát hiện lỗi rôto hệ thống điện, v.v. Hàm vi sai biên độ trung bình (AMDF) là một phương pháp được sử dụng rộng rãi để trích xuất chu kỳ của tín hiệu tuần hoàn vì độ phức tạp tính toán thấp và độ chính xác cao. AMDF là một biến thể của hàm tự tương quan, tuy nhiên thay vì tương quan tín hiệu đầu vào ở các độ trễ khác nhau như

đã làm với tương quan, nơi chúng ta phải thực hiện phép nhân và tổng ở mỗi giá trị của độ trễ, thì AMDF được dùng để lấy giá trị tuyệt đối của sự khác biệt giữa tín hiệu gốc và tín hiệu trễ để làm giảm độ phức tạp của thuật toán, điều này làm cho AMDF phù hợp hơn với các ứng dụng thời gian thực.

Hàm vi sai biên độ trung bình (Average Magnitude Difference Function) được định nghĩa như sau:

$$D(\tau) = \frac{1}{N-1-\tau} \sum_{n=0}^{N-1-\tau} |x(n) - x(n+\tau)|$$

Trong đó:

- $x\,(n)$ là tín hiệu giọng nói được xử lý với độ dài N
- τ là độ trễ và nằm trong khoảng từ 0 đến $N-1$

Nếu $x\,(n)$ là tín hiệu tuần hoàn của chu kỳ P, hàm sai lệch $x\,(n) - x\,(n+\tau)$ sẽ tiến về 0 đối với độ trễ trung bình của biên độ tín hiệu với các khoảng thời gian là $0, \pm P, \pm 2P$ ... Do đó tần số cơ bản f0 được xác định chính là tần số ứng với độ trễ mà tại đó giá trị của $D(\tau)$ là nhỏ nhất và đồng thời phù hợp với khoảng tần số đã được xác định trước.

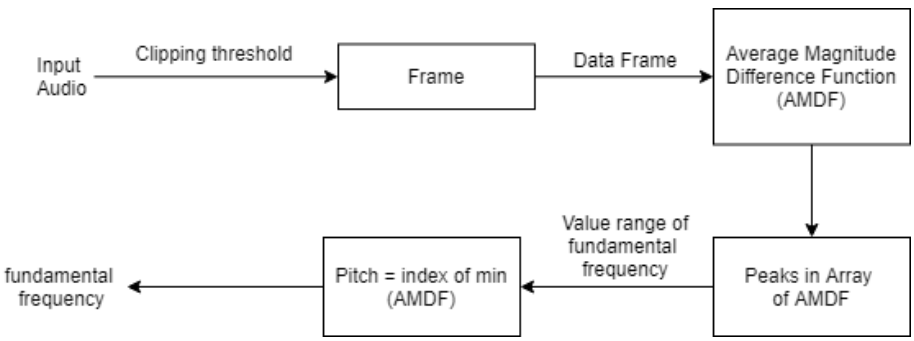
Công thức xác định tần số cơ bản:

$$T_p = \arg\underset{\tau}{\text{MIN}}\{D(\tau)\}$$

Trong đó:

- τ được giới hạn từ τ min đến τ max đã được cho trước.

2. Sơ đồ khối



Hình 2. Sơ đồ khối thuật toán tìm tần số cơ bản trên miền thời gian dùng hàm vi sai biên độ trung bình và dùng tìm đỉnh.

3. Các tham số quan trọng của thuật toán

FrameLength: Độ dài của một frame trong tín hiệu đầu vào được xác định theo yêu cầu đề bài chu kỳ lấy mẫu là T = 20ms tức là 320 mẫu.

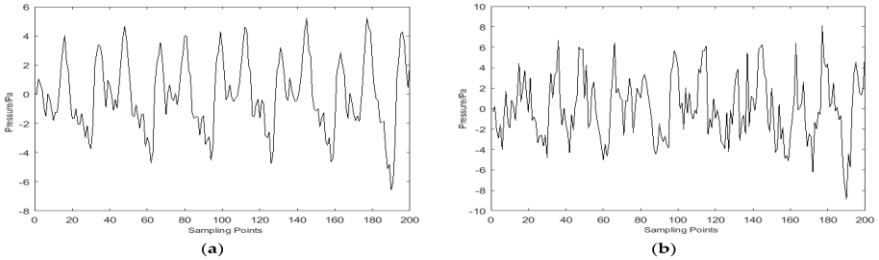
HopLenght: Chiều dài bước nhảy của frame sẽ cho biết số lượng frame xét trong tín hiệu. Theo yêu cầu thì chiều dài vước nhảy của frame bằng một nửa chiều dài của frame.

f_min: tần số thấp nhất mà con nam hoặc nữ hoặc một người nào đó có thể nói được dùng để xác định tần số cơ bản của người nói.

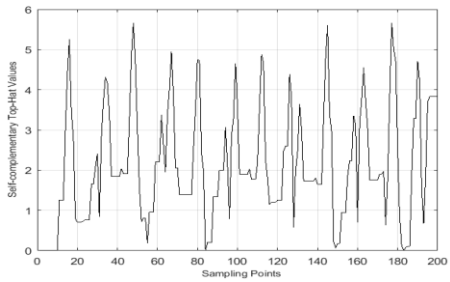
f_max: tần số cao nhất mà con nam hoặc nữ hoặc một người nào đó có thể nói được dùng để xác định tần số cơ bản của người nói.

4. Các vấn đề và giải pháp khắc phục

Hàm vi sai biên độ trung bình (AMDF) có ưu điểm là độ phức tạp tính toán thấp và độ chính xác cao. Tuy nhiên, phương pháp này có độ chính xác phát hiện thấp khi tiếng ồn xung quanh mạnh, khi đó phương pháp AMDF dễ bị nhiễu ngẫu nhiên dẫn đến vết khía không đủ rõ ràng dẫn đến tính toán tần số cơ bản không chính xác, vì vậy nó cần một phương pháp lọc để khử tín hiệu trước. Giải pháp ở đây là sử dụng biến đổi hình thái Top-Hat (STH) kết hợp với AMDF, Biến đổi Top-Hat tự bổ sung (STH) là một phương pháp làm giảm nhiễu dựa trên một số hoạt động của nó, sử dụng tốt trong việc khử nhiễu nền ở mức độ lớn và giữ lại các chi tiết của tín hiệu gốc.



Hình 3. (a) Đoạn tín hiệu không bị nhiễu (b) Đoạn tín hiệu bị nhiễu



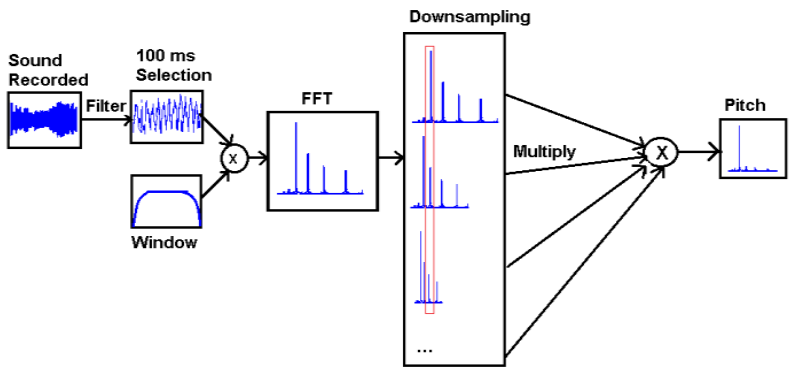
Hình 4. Đoạn tín hiệu sau khi sử dụng Top-Hat
Credit: Zhao Han & Xiaoli Wang. [4]

D. Phép biến đổi Fourier nhanh và giảm bậc hài để tìm tần số cơ bản trên miền tần số

1. Cơ sở lý thuyết

Như ở kĩ thuật phát hiện tần số cơ bản trên miền thời gian, chúng ta sẽ chỉ làm việc với một cửa sổ của tín hiệu một lúc (với độ dài và bước nhảy cho trước bởi đầu vào của thuật toán). Với mỗi cửa sổ như vậy, chúng ta nhân nó với một cửa sổ Hanning (Hanning Window), sau thực hiện phép biến đổi Fourier nhanh (FFT). Do FFT sinh ra khá dư dả (tính đối xứng) về dữ liệu cho một tín hiệu thực, chúng ta chỉ làm việc với một nửa kết quả của FFT. Nếu cần có thể tái tạo lại về sau.

Nếu tín hiệu đầu vào là một tín hiệu tuần hoàn thì phổ của nó sau khi qua phép biến đổi FFT sẽ bao gồm một chuỗi các đỉnh, tương ứng với tần số cơ bản với những đỉnh hài (harmonic) thành phần gấp nguyên lần tần cơ bản. Vì vậy khi chúng ta nén phổ của tín hiệu đó một số lần (downsampling), và so sánh nó với phổ nguyên gốc, ta sẽ thấy các đỉnh hài lớn nhất thẳng hàng với nhau trên một tọa độ tần số. Đỉnh thứ nhất của phổ nguyên gốc trùng với đỉnh thứ hai của phổ đã nén với tỉ lệ hai lần, trùng với đỉnh thứ ba của phổ đã nén với tỉ lệ ba lần. Vậy nên, khi những phổ này nhân với nhau sẽ sinh ra kết quả là một phổ với đỉnh cao nhất tại tần số cơ bản. Từ đó ta chỉ cần duyệt phổ kết quả này để tính tần số cơ bản của khung tín hiệu đó. [1]



Hình 5. Các bước tìm kiếm tần số cơ bản trên miền tần số thông qua phương pháp giảm bậc hài của phổ biên độ.
Credit: Vo, Thanh & Sawada, Hideyuki. (2017). [2]

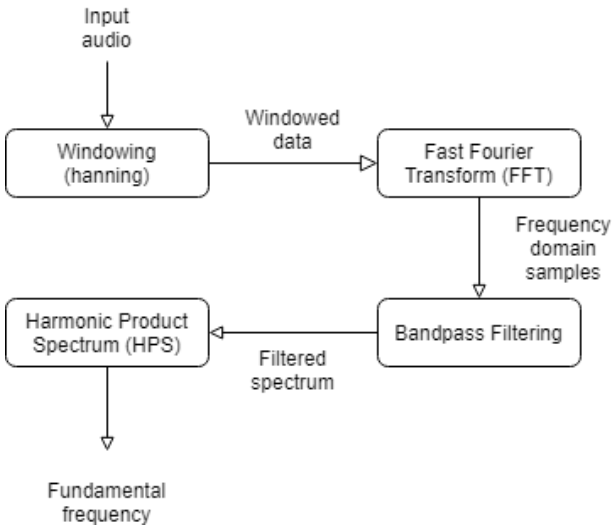
Phương pháp này có thể được biểu diễn bằng công thức sau:

$$Y(e^{j\omega}) = \prod_{r=1}^{R} \left|X(e^{j\omega r})\right|^2$$

$$p = \underset{\omega}{\operatorname{argmax}}\{Y(\omega)\}$$

Trong đó, R là số hài (harmonics) được xét. Sau đó thực hiện tìm kiếm giá trị p lớn nhất của dữ liệu sau tích năng lượng trên miền tần số và trả lại được giá trị tần số cơ bản.

2. Sơ đồ khối



Hình 6. Sơ đồ khối thuật toán tìm tần số cơ bản trên miền tần số bằng phép biến đổi Fourier nhanh và phương pháp giảm bậc hài của phổ.

3. Các tham số quan trọng của thuật toán

WINDOW_SIZE: Điều chỉnh kích thước của các frame dữ liệu đầu vào cho thuật toán, frame càng nhỏ hay càng lớn sẽ ảnh hưởng đến độ chính xác của thuật toán.

FFT_POINT: Điều chỉnh độ dài của phép biến đổi Fourier nhanh, qua đó điều chỉnh độ phân giải của kết quả.

PRODUCTS: Điều chỉnh số lần hạ bậc tối đa của khung dữ liệu. Đối với những tần số cơ bản thấp, khi sử dụng số lần hạ bậc cao sẽ đem lại độ chính xác nhất định.

LOW_PASS: Điều chỉnh tần số tối thiểu mà thuật toán có thể sinh để lọc những trường hợp nhiễu, ảo.

HIGH_PASS: Điều chỉnh tần số tối đa mà thuật toán có thể sinh để lọc những trường hợp nhiễu, ảo.

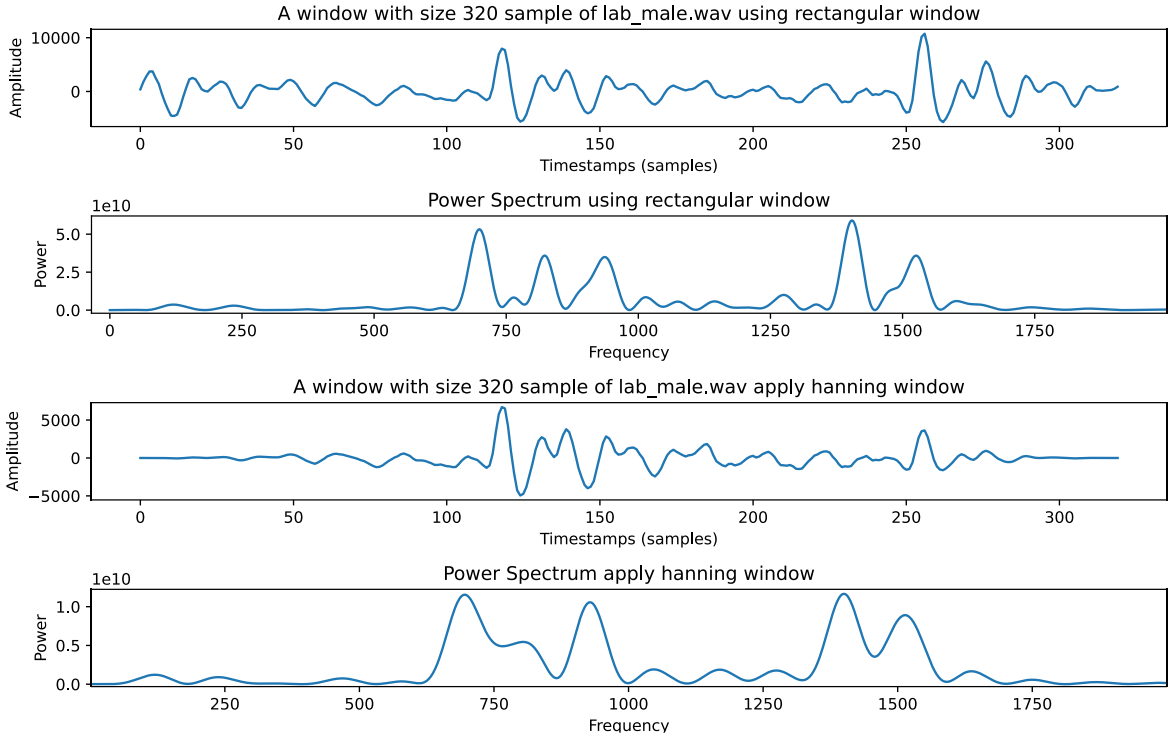
4. Các vấn đề và giải pháp khắc phục

Tuy vậy một điểm yếu nghiêm trọng của phương pháp giảm bậc hài phổ đó là độ phân giải của kết quả sẽ tốt nhất chỉ bằng với độ phân giải của phép biến đổi Fourier nhanh. Nếu chúng ta chỉ thực hiện một phép biến đổi Fourier nhanh và ngắn, chúng ta đã giới hạn số lượng giá trị tần số rời rạc mà kết quả của thuật toán có thể sinh ra được. Để có được một kết quả có độ phân giải cao hơn (nhờ đó nhìn thấy biểu đồ cao độ mượt hơn), chúng ta phải thực hiện một phép Fourier dài hơn, tức là tốn nhiều thời gian hơn.

Ở thuật toán nén hài phổ (Harmonic Product Spectrum) còn bộc lộ một điểm yếu lớn là khi nén phổ lại thông tin sẽ dần bị mất dần, nên đối với những tần số cơ bản quá thấp, thuật toán này sẽ phát hiện sai. Cách khác là tăng kích thước cửa sổ để mức năng lượng của các tần số thấp sau khi thực hiện phép biến đổi Fourier tăng cao hơn để khi thuật toán thực hiện không bỏ lỡ mức năng lượng này. Xem khảo sát ở phần kết quả định lượng để thấy rõ sự thay đổi này.

Khi xét phân khung, có một khó khăn rất phổ biến đó là phân biệt giữa các khung có tín hiệu và giữa các khung không có tín hiệu. Giải pháp đề ra là dựa trên biên độ năng lượng sau khi biến đổi FFT để tạo nên một ngưỡng. Biên độ của tín hiệu trong khung vô thanh thường nhỏ hơn rất nhiều so với biên độ tín hiệu trong khung hữu thanh, dựa vào đặc tính này chúng ta sinh ra một ngưỡng để lọc các khung không có tín hiệu hữu thanh về không có tần số cơ bản.

Bên cạnh đó một vấn đề khác của việc phân khung đó là vấn đề rò phổ, do việc xét trong một cửa sổ tín hiệu có kích thước xác định gây nên. Để khắc phục hiện tượng này, chúng ta có thể áp dụng các cửa sổ Hanning (hoặc các loại cửa sổ khác) hoặc tăng tần số lấy mẫu của tín hiệu. Tuy nhiên việc tăng tần số lấy mẫu của tín hiệu là không khả thi, nên trong phạm vi bài báo cáo này, chúng tôi áp dụng cửa sổ Hanning để giải quyết vấn đề rò phổ một phần. Do cửa sổ Hanning có đặc tính là giảm dần biên độ về hai phía, giữ biên độ cao nhất ở phần trung tâm, nhờ đó khi áp dụng cửa sổ Hanning vào cửa sổ đang xét phần nằm ở biên giới cửa sổ có thể là một phần của chu kì trước hoặc sau sẽ bị giảm năng lượng. Phép biến đổi Fourier nhanh sau đó có thể cho ra một kết quả giảm bớt hiện tượng rò phổ.



Hình 7. Hiện tượng rò phổ sinh ra các đỉnh con ảo làm kém chính xác thuật toán. Khi áp dụng cửa sổ chữ nhật, thuật toán trả lại kết quả f0 là 232.4219Hz. Còn khi áp dụng cửa sổ Hanning, thuật toán trả lại kết quả 117.1875Hz, và con số này chính xác hơn trong trường hợp tần số cơ bản thấp này.

III. MÃ CHƯƠNG TRÌNH CÀI ĐẶT CÁC THUẬT TOÁN

A. Hàm tự tương quan để tính tần số cơ bản trên miền thời gian

1. Các thư viện cần dùng

```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.io.wavfile import read
```

2. Chia tín hiệu thành các frames

```
def divide_signal_into_frames(y, frame_size, frame_stride, fs):
        frame_len = int(fs*frame_size) # number of samples in a single frame
        frame_step = int(fs*frame_stride) # number of overlapping samples
        total_frames = int(np.ceil(float(np.abs(len(y)-frame_len))/frame_step)) # total
        frames of signal
        #push new array to end of matrix array
        padded_y = np.append(np.array(y), np.zeros(frame_len * total_frames - len(y)))
        framed_y = np.zeros((total_frames, frame_len)) #frame y (shape)
        for i in range(total_frames): #loop through over of total frames
                # Apply ag Xj * Xj+t
                framed_y[i] = padded_y[i*frame_step : i*frame_step + frame_len]
        return framed_y
```

3. Tính hàm tự tương quan của mỗi frame

```
#Compute Autocorr Of each frames
def calculate_difference(frame) :
        half_len_signal = len(frame)//2
        j = 0
        # Init new Empty array to contain autocorrs of frame
        autocorr = np.zeros(half_len_signal)
        for j in range(half_len_signal):
                for i in range(half_len_signal):
                diff = frame[i] - frame[i+j] # Do lech bien do
                        autocorr[j] += diff**2
        return autocorr
```

4. Chuẩn hóa

```
def normalize_with_cumulative_mean(autocorr, halflen):
        new_autocorr = autocorr
        new_autocorr[0] = 1
        running_sum = 0.0
        for tau in range(1,halflen):
                running_sum += autocorr[tau]
                new_autocorr[tau] = autocorr[tau]/((1/tau)*running_sum)
        return new_autocorr
```

5. Ngưỡng tuyệt đối

```
def absolute_threshold(new_autocorr, halflen, threshold):
        #create new array with condition
        temp = np.array(np.where(new_autocorr < threshold))
        if (temp.shape == (1,0)):
                tau = -1
        else :
                tau = temp[:,0][0]
        return tau
```

6. Nội suy Parabol

```
def parabolic_interpolation(new_autocorr, tau, frame_len):
        if tau > 1 and tau < (frame_len//2-1):
                alpha = new_autocorr[tau-1]
                beta = new_autocorr[tau]
                gamma = new_autocorr[tau+1]
                improv = 0.5*(alpha - gamma)/(alpha - 2*beta + gamma)
        else :
                improv = 0
        new_tau = tau + improv
        #return new_tau
        return new_tau
```

7. Pitch Tracking(Tìm đỉnh)

```
def pitch_tracker(y, frame_size, frame_step, sr):
        #call to divide_signal and push args into it
        framed_y = divide_signal_into_frames(y, frame_size, frame_step, sr)
        pitches = []
        for i in range(len(framed_y)):
        #Find autocorrelation with i in rnage len of framed_y
                autocorr = calculate_difference(framed_y[i])
                #Find new autocorrelation with i in rnage len of framed_y
                new_autocorr = normalize_with_cumulative_mean(autocorr, frame_len//2)
        #find threshold
                tau = absolute_threshold(new_autocorr, frame_len//2, 0.16)
                new_tau = parabolic_interpolation(new_autocorr, tau, frame_len)
                if (new_tau == -1):
                        pitch = 0
                else :
                        pitch = sr/new_tau
                #push pitch into pitches array
                pitches.append(pitch)
        #get timestamps
        list_times = [int(i * frame_len / 2) for i in range(len(pitches))]
        #return a tuple type with pitches and times
        return (pitches, list_times)
```

8. Hàm main

```
if __name__ =='__main__':
        sr, y = read('lab_female.wav')
        frame_size = 0.03
        frame_step = 0.01 #hop length
        frame_len = int(frame_size * sr) # #block length

        pitches = pitch_tracker(y, frame_size, frame_step, sr)
        #0 ->500
        plt.ylim([0, 500])
        #title
        plt.title('Pitch Tracking Of Signal lab_female.wav Using AutoCorrelation')
        #xlable
        plt.xlabel('Time Stamps')
        plt.ylabel('F0')
        # plot
        plt.plot(pitches[1], pitches[0], 'o')
```

```
#show
plt.show()
```

B. Hàm vi sai biên độ trung bình để tính tần số cơ bản trên miền thời gian

1. Cài đặt thư viện và hàm con

a) Thư viện.

```
import numpy as np
from scipy.io.wavfile import read
import matplotlib.pyplot as plt
```

b) Hàm con.

```
# find min of array x
# input: x
# output: value min 'm'
def minx(x):
    m = x[0]
    for n in x:
        if(n < m): m = n
    return m

# find max of array x
# input: x
# output: value max 'm'
def maxx(x):
    m = x[0]
    for n in x:
        if(n > m): m = n
    return m
```

```
# find sum of array x
# input: x
# output: value sum 's'
def sumx(x):
    s = 0
    for n in x:
        s += n
    return s

# find mean of array x
# input: x
# output: value mean of ‘s’ with ‘s’ is sum of array x
def meanx(x):
    s = 0
    if(len(x)==0): return 0
    for n in x:
        s +=n
    return s/len(x)
```

2. Hàm tính vi sai biên độ trung bình.

```
# find the mean amplitude deviation of each frame in signal
# input: a frame of signal
# output: an array of mean amplitude deviation of each frame in signal after using ADMF
def computeAmdf(frame):
    k = len(frame)
    if k <= 0:
        return 0
    # create a matrix with one row and k colum
    afAmdf = np.ones(k)
    # find mean amplitude deviation in frames and add that value into array afAmdf
    for i in range(0,k):
        afAmdf[i] = meanx(np.abs(frame[np.arange(0, k - 1 - i)] - frame[np.arange(i + 1, k)]))
    return (afAmdf)
```

3. Tìm đỉnh.

```
# find the mean of signal to create an 'baseline' for 'multi_peak_finding' function
# input: signal x
# output: the mean of signal x
def baseline(x):
    return meanx(x)

# Find the peak of a 'frame' of signal base on 'baseline'
# input: frame of signal
# output: a array contain low peaks of a 'frame' in signal called  'peakIndices'
def multi_peak_finding(frame, baseline):
    peakIndices = []
    peakIndex = None
    peakValue = None
    for index, value in enumerate(frame):
        if value < baseline:
            if peakValue == None or value < peakValue:
                peakIndex = index
                peakValue = value
        elif value > baseline and peakIndex != None:
            peakIndices.append(peakIndex)
            peakIndex = None
            peakValue = None
    if peakIndex != None:
        peakIndices.append(peakIndex)
    return peakIndices
```

4. Tìm tần số cơ bản trên miền thời gian.

```
# Pitch Tracking of signal using AMDF
# input:
#       x: the signal x read from file,
#       FrameLength: the length of a frame in signal
#       Hoplenght: Hop length of a frame in signal
#       f_s: sample rate of signal read from file and f_s = 16000Hz
# output:
#       f : an array sampling frequency (f0) in each frame of signal
#       t: time stamps
def PitchTimeAmdf(x, FrameLength=320, HopLength=160, f_s):
    # from 80 Hz to 250 Hz: fequency of male can speak
    # from 120 Hz to 400 Hz: fequency of female can speak
    # from 75 Hz to 350 Hz: the average fequency of adults can speak
    f_min = 80
    f_max = 250

    # the number of frames in signal
    NumOfFrames = x.size // HopLength

    # compute time stamps
    t = (np.arange(0, NumOfFrames) * HopLength + (FrameLength / 2)) / f_s

    # create a matrix 'f' contain sampling frequency (f0) in each frame
    f = np.zeros(NumOfFrames)

    # the limited samples of male or female can speak in signal
    sample_max = f_s / f_min
    sample_min = f_s / f_max

    # find threshold of signal x to determine which frame is voice sound or unvoice sound
    cout = 0
    thres = 0
    for n in x:
        if(n > 0):
            thres += n
            cout +=1
    thres = thres / cout

    # find f0 in each frame
    for n in range(0, NumOfFrames):
        # the first position of frame
        i_start = n * HopLength
        # the last position of frame
        i_stop = minx([len(x) - 1, i_start + FrameLength - 1])

        # return true i_start < i_stop and false if  i_start >= i_stop
        if(i_start >= i_stop):
            continue
        else:
            # create a frame
            x_tmp = x[np.arange(i_start, i_stop + 1)]

            # if the max pitch in frame < threshold of signal then f0 = 0
            thres_frame = maxx(x_tmp)
            if(thres_frame < thres):
                f[n] = 0
                continue

            # compute AMDF (average magnitude difference function) to create an array of AMDF
            afAmdf = computeAmdf(x_tmp)

        # create an array contain peaks in afAmdf
        peak_array = multi_peak_finding(afAmdf, baseline(afAmdf))

        # create an array contain the value of peaks in afAmdf
        k = [afAmdf[peak] for peak in peak_array]

        # return f0 = 0 in frame n if 'multi_peak_finding' function can not find peaks in this fame
        if(len(k)<=2):
            f[n]=0
            continue

        # find f0 in frame n
        index = 1
        while(index < len(k)):
```

```
                # find min value of peak in frame
                if(k[index] == minx(k[1:len(k)-1])):
                    # if min value of peak in frame > sample_max,
                    # i will plus that min value to 1000 and then it is not a min value.
                    if(peak_array[index]>sample_max):
                        k[index] += 1000
                        index=1
                    elif(peak_array[index]<sample_min):
                        f[n]=0
                    else:
                        f[n] = f_s / peak_array[index]
                        break
                index+=1

        return (f, t)
```

C. Phép biến đổi Fourier nhanh và giảm bậc hài kết quả của nó để tính tần số cơ bản trên miền tần số

1. Cài đặt thư viện và hàm con

Trong chương trình có sử dụng các hàm built-in của Python và các hàm nguyên thủy của thư viện `numpy` để xử lý mảng. Sử dụng thư viện `matplotlib.pyplot` để xuất đồ thị, và thư viện `scipy.io` để đọc file audio.

a) Thư viện

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

b) Hàm tìm ngưỡng tối thiểu

```
def find_baseline(data, sampling_rate, PRODUCTS):

'''
    To tell the difference of the non-voice and voice parts of the signal, we have to form a baseline based
on the power spectrum.

    This function assume the first 2000 samples is in the non-voice part of the signal.
    It'll use the Harmonic Product Spectrum function to generate the corresponding baseline. For further
details about the function, please find it in the main function, because this is just a simplify version of
it.

    Input:
        data : array-like
           Full-length signal.
        sampling_rate : int
           Sampling rate of the signal.
        PRODUCTS : int
           Number of products to evaluate the harmonic product spectrum.
    Output:
        maxv : int
           Base energy level that the algorithm based on to filter out the non-voice part's f0.
'''

    WINDOW_SIZE = 2000
    window = np.ones(WINDOW_SIZE)
    i=1
    get_data_slice = data[int((0.5*i-0.5)*WINDOW_SIZE) : int((0.5*i+0.5)*WINDOW_SIZE)]
    fourier_transform = np.fft.rfft(get_data_slice*window, 2**13)

    abs_fourier_transform = np.abs(fourier_transform)
    power_spectrum = np.square(abs_fourier_transform)

    down = []
    for i in range(1, PRODUCTS+1):
        down.append(power_spectrum[::i])

    res = []
    for i in range(len(down[-1])):
        mul = 1
        for j in range(0, PRODUCTS):
            mul *= down[j][i]
        res.append(mul)

    maxv = max(res)

    return maxv
```

2. Hàm tìm tần số cơ bản trên miền tần số sử dụng Harmonic Product Spectrum

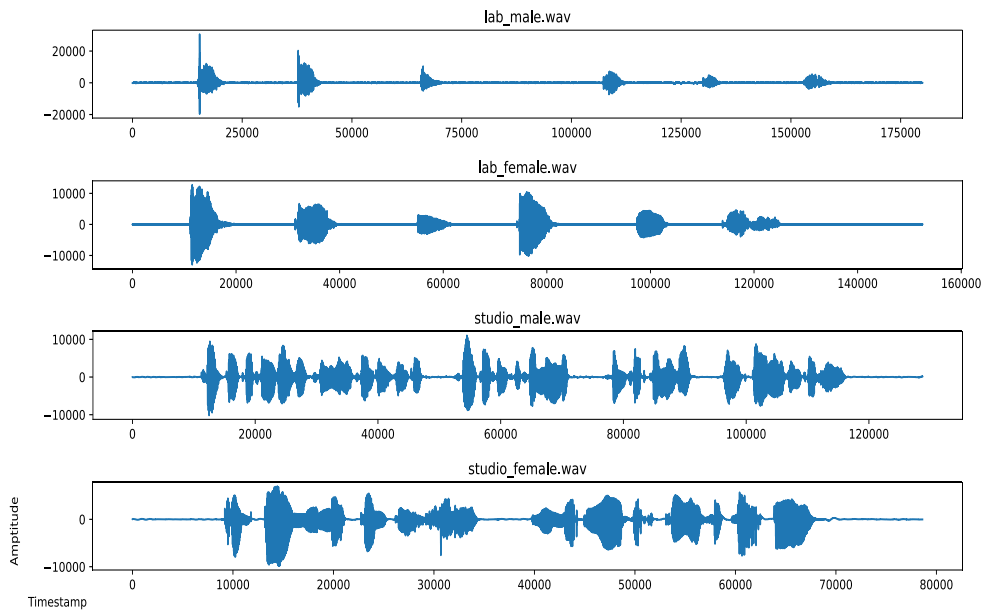
```
'''
    Harmonic product spectrum.

    This algorithm is used for fundamental frequency detection. It evaluates
    the magnitude of the FFT (fast Fourier transform) of the input signal, keeping
    only the positive frequencies. It then element-wise-multiplies this spectrum
    by the same spectrum downsampled by 2, then 3, ..., finally ending after
    numProd downsample-multiply steps.

    Parameters
    ----------
    FILE_PATH : String
      Specify where to locate original signal

    PRODUCTS : int
      Number of products to evaluate the harmonic product spectrum.

    FFT_POINT : int
      The length of the FFT. FFT lengths with low prime factors (i.e., products of 2,
      3, 5, 7) are usually (much) faster than those with high prime factors.

    LOW_PASS : int
      Lowest frequency that will be in the output

    HIGH_PASS : int
      Highest frequency that will be in the output

    WINDOW_SIZE : int
      The size of window of signal

    Return
    ----------
    Result : Array-like
      List of fundamental frequency corresponding with each window
'''

def find_f0_using_HPS(WINDOW_SIZE, LOW_PASS, HIGH_PASS, PRODUCTS, FILE_PATH, FFT_POINT):

    # Read data from original file
    sampling_rate, data = wavfile.read(FILE_PATH)
    # Innitiallize hanning window with the same size ofthe window
    window = np.hanning(WINDOW_SIZE)
    # Calculation of basic baseline to filter noise later
    base = find_baseline(data, sampling_rate, PRODUCTS)
    # Result list variable
    result = []

    # Slide through the signal with windows that overlaps 50% of each other
    for i in range(1, int(len(data)/(WINDOW_SIZE))*2-1):

        # EXTRACT DATA FROM SIGNAL AND COMPUTE FFT
        # Slice data with the windowsize that overlaps 50%
        get_data_slice = data[int((0.5*i-0.5)*WINDOW_SIZE) : int((0.5*i+0.5)*WINDOW_SIZE)]
        # Generate the fourier power and its corresponding frequency
        fourier_transform = np.fft.rfft(get_data_slice*window, FFT_POINT)
        frequency = np.linspace(0, sampling_rate/2, num=len(fourier_transform))

        # BANDPASS FILTERING
        # Create filter the value of too high or too low frequencies in the fourier result
        pas = []
        for i in range(len(frequency)):
            # Cut off the too low and too high frequencies
            if frequency[i]<LOW_PASS or frequency[i]>HIGH_PASS*PRODUCTS:
                pas.append(0)
            else:
                pas.append(1)
        # Apply the filter to the result of the FFT
        for i in range(len(frequency)):
            fourier_transform[i] *= pas[i]
        # Compute the Power Spectrum
        abs_fourier_transform = np.abs(fourier_transform)
        power_spectrum = np.square(abs_fourier_transform)

        # DOWNSAMPLING
        # Downsampling by the numnber of products
```

```
        down = [] # This is for saving the downsampled sequence
        for i in range(1, PRODUCTS+1):
            # Here, "downsampling a vector by N" means keeping only every N samples: downsample(v, N) = v[::N].
            down.append(power_spectrum[::i])

        # MULTIPLICATION
        # Element-wise-multiplies by the downsampled spectrum versions
        res = [] # this is for saving the result after the multiplication
        for i in range(len(down[-1])):
            mul = 1
            for j in range(0, PRODUCTS):
                mul *= down[j][i]
            res.append(mul)

        # FIND THE PEAK AND EXTRACT F0
        # Find the index of the max peak of the multiplied result of the downsample versions
        maxv = max(res)
        indv = res.index(maxv)
        # Compare with the base TO FILTER OUT non_sound parts
        if maxv < base:
            indv = 0
        # Final compare with the condition then extract f0 for this frame
        if frequency[indv] < LOW_PASS or frequency[indv] > HIGH_PASS:
            result.append(0)
            continue
        else:
            result.append(frequency[indv])

    return result
```

IV. KẾT QUẢ THỰC NGHIỆM

A. Dữ liệu mẫu

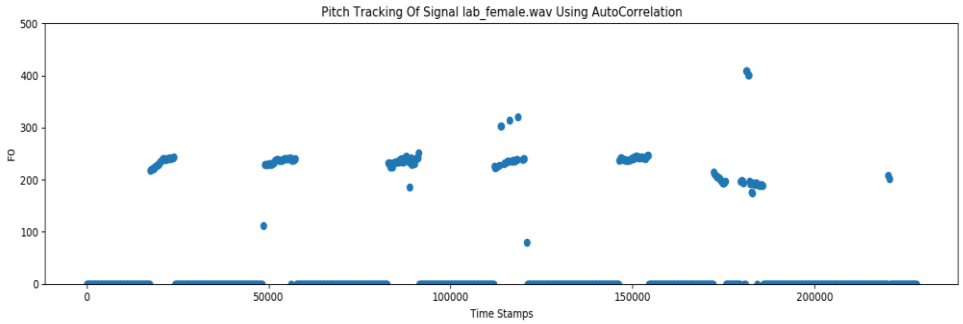
Dữ liệu dùng để đánh giá thuật toán là 4 tín hiệu người nam, nữ nói trong 2 môi trường lab và studio với tần số lấy mẫu là 16000Hz có biểu diễn đồ thị như hình bên dưới.



Hình 8. Tín hiệu đầu vào gồm 4 tín hiệu tiếng nói nam, nữ như trên.

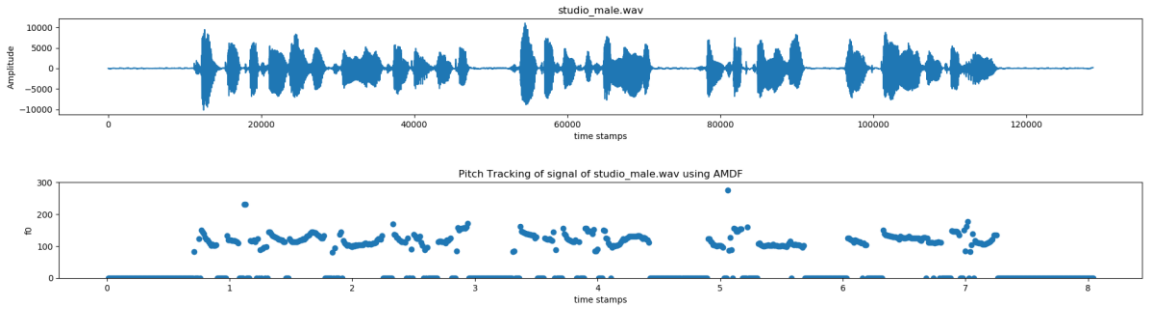
B. Kết quả định tính

1. Hàm tự tương quan

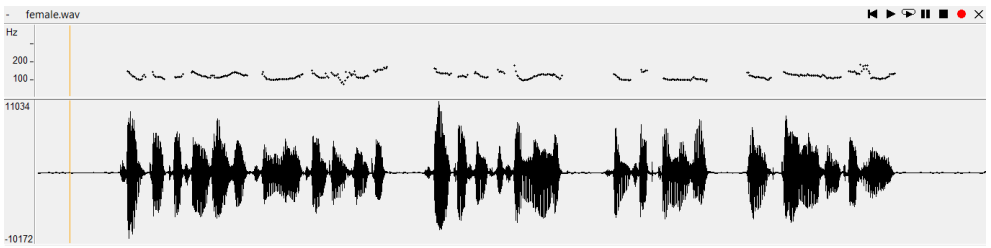


Hình 9. Biểu đồ kết quả tính tần số cơ bản của giọng nói nữ trong môi trường lab sử dụng hàm tự tương quan

2. Hàm vi sai biên độ trung bình



Hình 10. Biểu đồ kết quả tính tần số cơ bản của giọng nói nam trong môi trường studio qua thuật toán AMDF

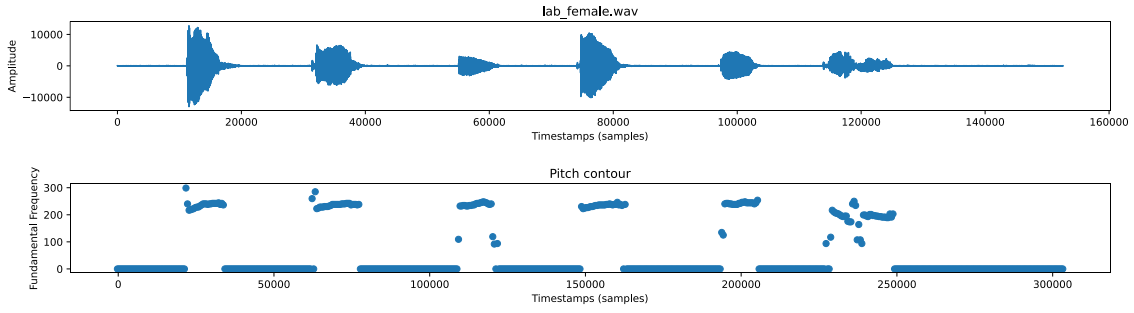


Hình 11. Biểu đồ kết quả tính tần số cơ bản của giọng nói nam trong môi trường studio qua phần mền Wavesurfer

So sánh giữa 2 biểu đồ ta có thể nhận thấy kết quả tính toán tần số cơ bản qua thuật toán AMDF là gần như chính xác, tuy nhiên có sự khác nhau một số điểm do sự khó khăn trong việc phân biệt tín hiệu tuần hoàn và không tuần hoàn nên dẫn đến sai số nhất định

3. Phép biến đổi Fourier nhanh và giảm bậc hài của nó

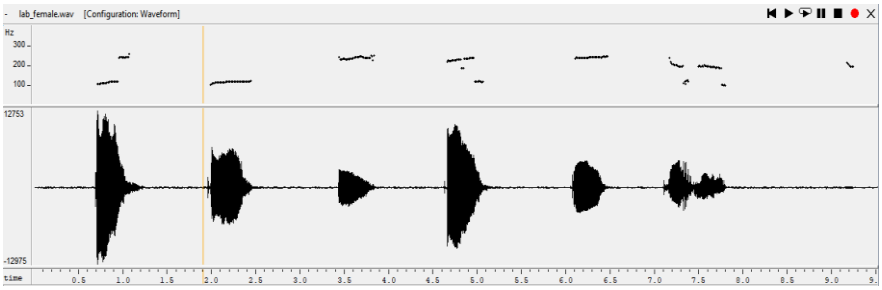
Sử dụng tín hiệu thu âm của người nữ trong môi trường lab để thử kết quả thực hiện của thuật toán.



Hình 12. Biểu đồ kết quả phân tính cao độ của tín hiệu giọng nói nữ trong môi trường lab qua thuật toán HPS.

So sánh với kết quả của chương trình WaveSurfer, nhận thấy rằng kết quả của thuật toán có phần chính xác hơn, khi tính được tần số cơ bản nằm trong khoảng 230hz. Trong khi đó ở kết quả của chương trình WaveSurfer, chúng ta có thể nhìn thấy có đoạn chương trình tính toán được tần số cơ bản nằm trong khoảng 120 hz, tức là chỉ bằng một nửa so với thực tế.

Tuy vậy chúng ta vẫn còn thấy ngưỡng lọc hữu thanh và vô thanh vẫn chưa hoạt động hiệu quả, thông qua việc xuất hiện các giá trị đột biến ở đầu và cuối đoạn có âm, từ kết quả của output này chúng ta có thể thực hiện một phép lọc trung vị để làm mượt, loại bỏ các giá trị đột biến.



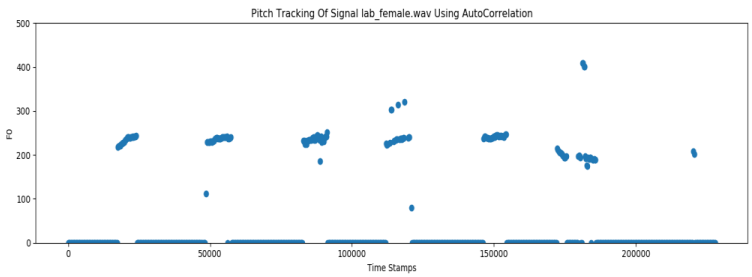
Hình 13. Biểu đồ kết quả của phần mềm WaveSurfer đối với file lab_female.wav

C. Kết quả định lượng

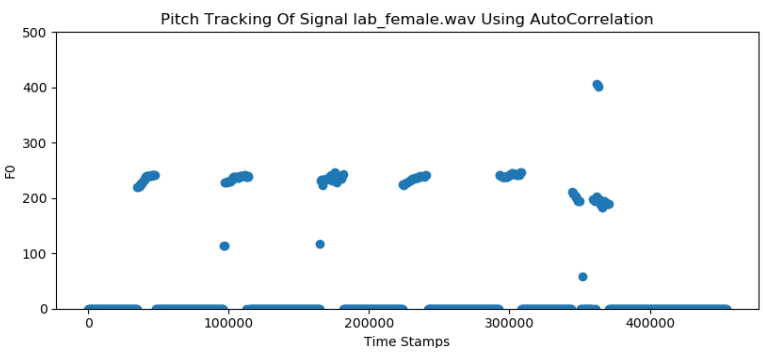
1. Hàm tự tương quan

a) Khảo sát thay đổi kích thước Frame Length

Tăng kích thước này từ 0.03s lên 0.06s ta được đồ thị kết quả như phía dưới.



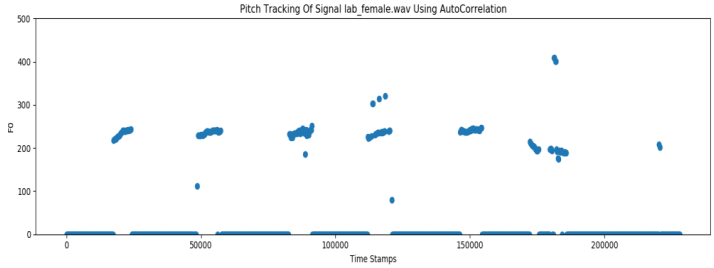
Hình 14. Đồ thị tìm cao độ giọng nói nữ sử dụng hàm tự tương quan với kích thước khung là 0.03s



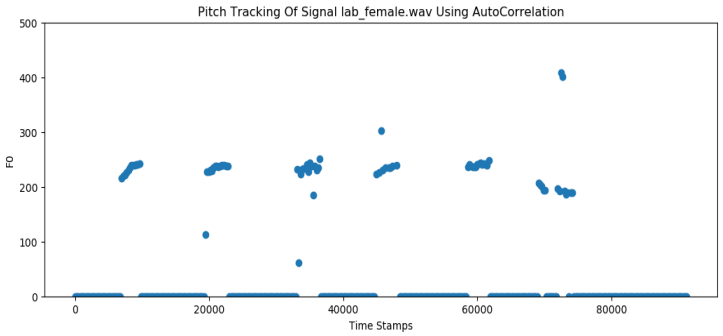
Hình 15. Đồ thị tìm cao độ giọng nói nữ sử dụng hàm tự tương quan với kích thước khung là 0.06s

Nhận thấy số lượng giá trị trả về sẽ ít hơn so với trước khi tăng, nhưng nhờ vậy mà giảm bớt các giá trị đột biến trong kết quả trả về.

b) Khảo sát thay đổi kích thước Hop Length



Hình 16. Đồ thị tìm cao độ giọng nói nữ sử dụng hàm tự tương quan với Hop Length 0.01

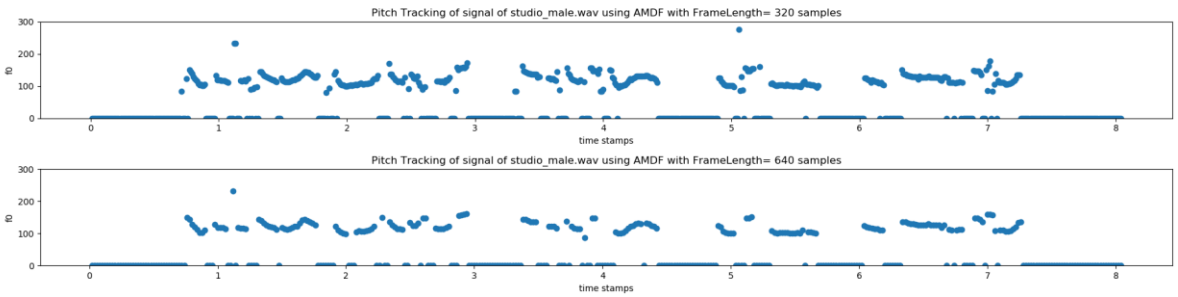


Hình 17. Đồ thị tìm cao độ giọng nói nữ sử dụng hàm tự tương quan với Hop Length 0.025

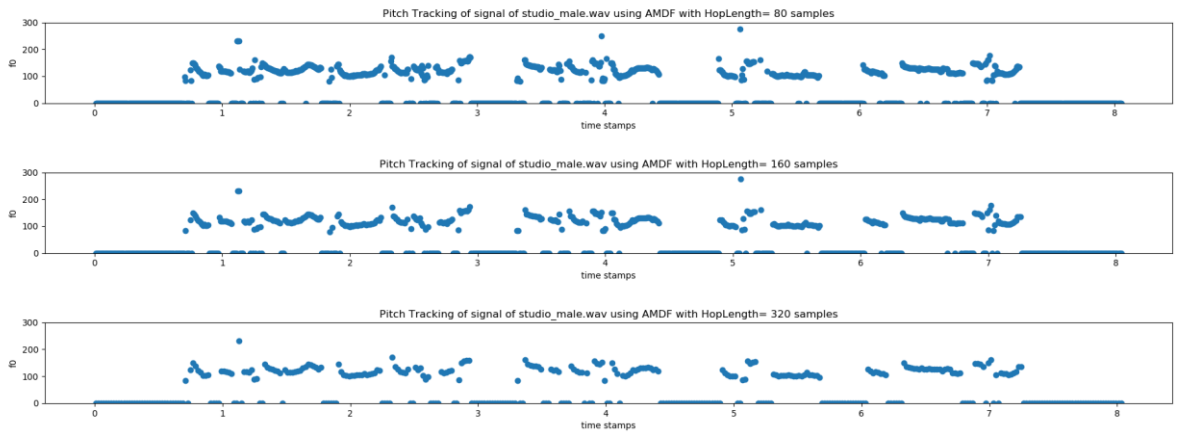
Khi tăng Hop Length lên cao gần với độ dài của khung, ta nhận thấy dường như không thấy sự thay đổi nhiều.

2. Hàm vi sai biên độ trung bình

Các tham số ảnh hưởng đến độ chính xác thuật toán như đã đề cập là FrameLength, HopLength, f_min, f_max được mô tả như hình vẽ.

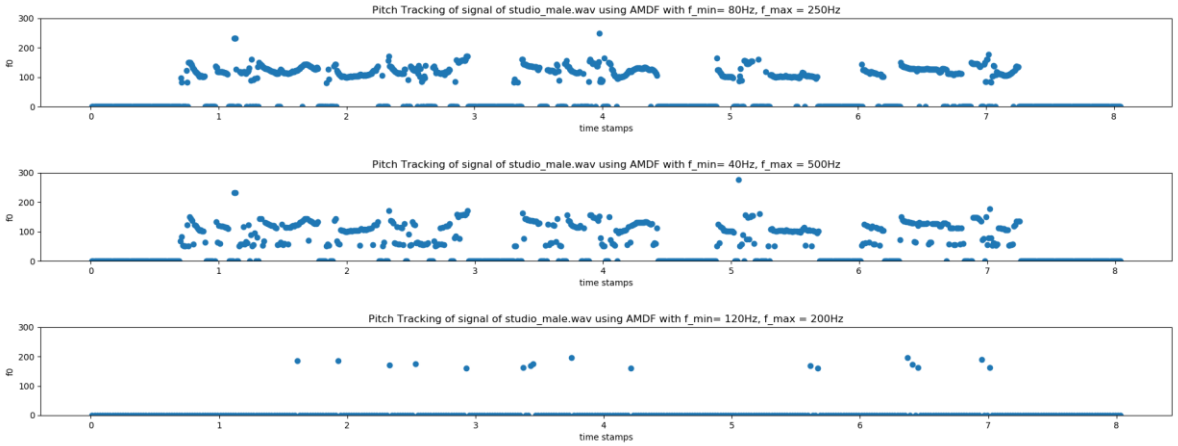


Hình 18. Sự khác nhau của tần số cơ bản khi tăng kích thước của FrameLength lên 2 lần



Hình 19. Sự khác nhau của tần số cơ bản khi tăng kích thước của HopLength lên 2 lần

Hình 18 và Hình 19 ta có thể nhận thấy rằng khi tăng FrameLength, HopLength tức là càng tăng số lượng mẫu thì càng ít mẫu hơn để xét dẫn đến làm giảm sự chính xác của thuật toán khi tồn tại nhiều chu kỳ tuần hoàn trong một mẫu nhưng không được xét đến. Nên cần phải chọn kích cỡ frame và chiều dài bước nhảy cho phù hợp để xác định chính xác tần số cơ bản.



Hình 20. Sự thay đổi f_min và f_max của giọng nói nam môi trường studio với tần số cơ bản chuẩn nằm trong khoảng 80Hz đến 250 Hz

f_min và f_max là giá trị đã cho trước tuy nhiên rất quan trọng việc xác định vị trí để tìm giá trị của tần số cơ bản. Nếu f_max quá lớn và f_min quá nhỏ thì dễ nhầm lẫn với vị trí ảo của tần số cơ bản, còn nếu f_max quá nhỏ và f_min quá lớn thì dễ xác định sai vị trí hoặc không thể tìm thấy tần số cơ bản.

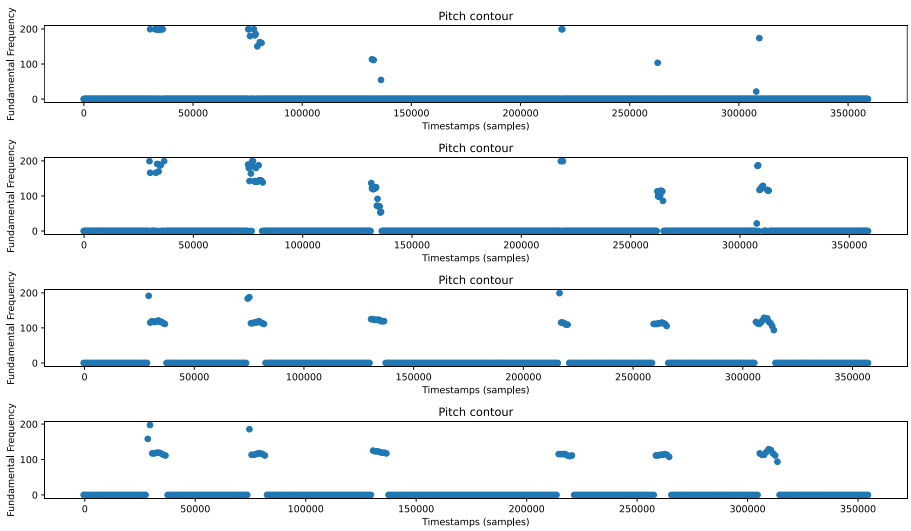
3. Phép biến đổi Fourier nhanh và giảm bậc hài của nó

Các tham số tham gia vào độ chính xác của thuật toán có thể kể đến là WINDOW_SIZE, FFT_POINT, PRODUCTS, LOW_PASS và HIGH_PASS. Trong phần này chúng ta khảo sát sự thay đổi của các tham số này, qua đó tính toán các ảnh hưởng của nó đến kết quả tính toán của thuật toán.

a) Khảo sát sự ảnh hưởng của tham số WINDOW_SIZE:

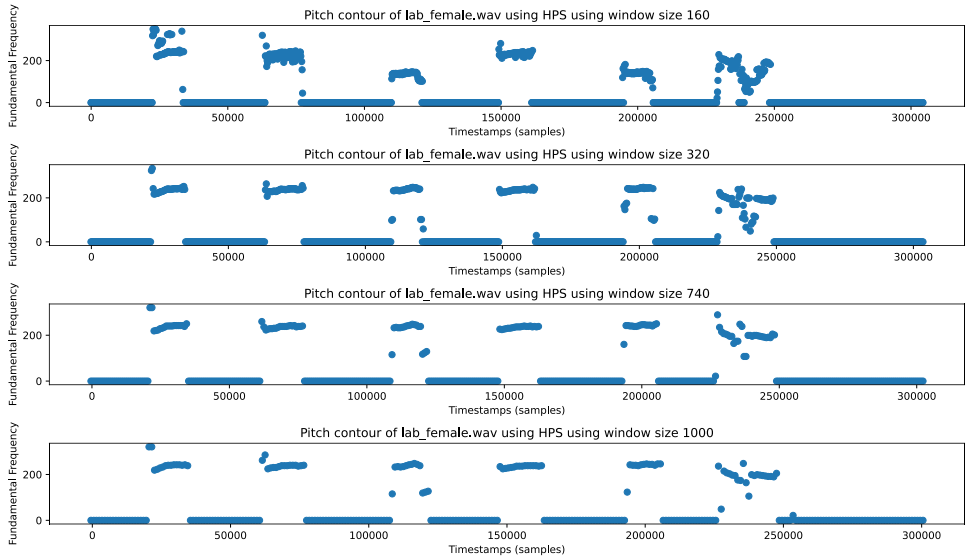
Phương pháp HPS có một nhược điểm, đó là khi dùng để phát hiện những âm thanh có tần số cơ bản thấp nó thường có xu hướng phát hiện tần số cao gấp đôi so với tần số gốc. Vì trong quá trình nén đã làm mất bớt thông tin dần dần khiến độ phân giải phát hiện tần số thấp ngày càng khó khăn hơn. Một trong những giải pháp đó chính là tăng kích thước cửa sổ để làm tăng mức năng lượng của tần số thấp, qua đó không chọn nhầm đỉnh của các đỉnh có tần số lớn hơn.

Để khảo sát tham số WINDOW_SIZE, xét tệp giọng nói nam "lab_male.wav" có tần số cơ bản khoảng 112Hz ở trong môi trường lab. Khảo sát kích thước cửa sổ giao động lần lượt từ 160, 320, 740 và 1000 mẫu với mức nén trong thuật toán PRODUCTS=5, độ phân giải phép Fourier nhanh FFT_POINT=2^{13}, tần số thấp nhất cho phép LOW_PASS=20, tần số cao nhất cho phép HIGH_PASS=200. Cho thấy khi tăng dần kích thước cửa sổ từ 160 đến 1000 mẫu kết quả thể hiện tốt nhất ở vào khoảng 740 - 1000 mẫu khi dùng để phát hiện cao độ của giọng trầm.



Hình 21. Sự thay đổi của kết quả đầu ra khi thay đổi kích thước cửa sổ, trong trường hợp tìm tần số cơ bản thấp trong tệp tín hiệu "lab_male.wav".

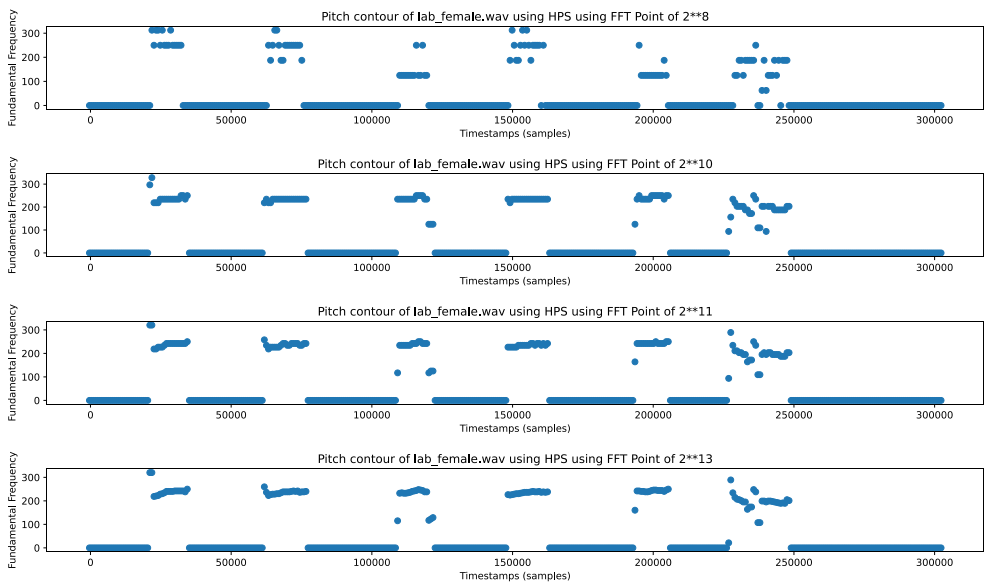
Trong trường hợp áp dụng cho tệp tín hiệu giọng cao như của nữ, ta nhận thấy từ kích thước cửa sổ khoảng 320 mẫu là đã đủ giúp thuật toán phát hiện tần số cơ bản ở mức chấp nhận được, tức là có khá ít sai số. Tuy vẫn có một số đột biến ở đầu và cuối đoạn hữu âm.



Hình 22. Sự thay đổi của kết quả đầu ra khi thay đổi kích thước cửa sổ, trong trường hợp tìm tần số cơ bản thấp trong tệp tín hiệu "lab_female.wav".

b) Khảo sát sự ảnh hưởng của tham số FFT_POINT

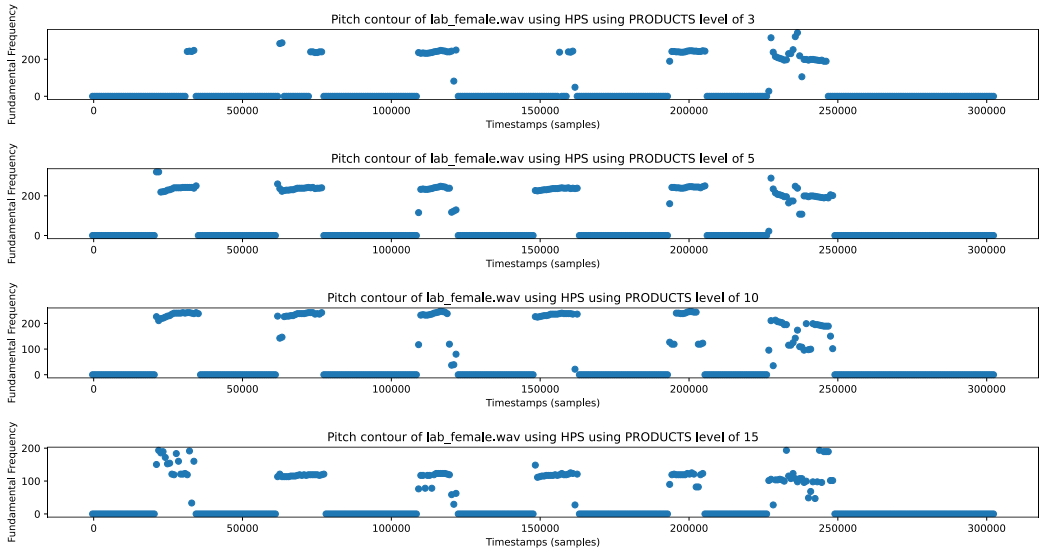
Tham số FFT_POINT xác định độ phân giải của phép biến đổi Fourier nhanh, tham số này càng lớn thì kết quả ra càng chính xác, độ phân giải của tần số cơ bản trả lại ngày càng chi tiết hơn, hay độ chia nhỏ nhất về tần số của phép Fourier nhanh càng giảm. Tuy vậy khi tăng tham số này, thuật toán sẽ tốn nhiều thời gian chạy hơn. Cụ thể thời gian chạy phép tìm tần số cơ bản với các FFT_POINT 2^8, 2^{10}, 2^{11}, 2^{13} lần lượt là 0.103925s, 0.350857s, 0.688157s, 2.650529s.



Hình 23. Kết quả tìm tần số cơ bản của thuật toán HPS khi tăng dần độ phân giải của phép biến đổi Fourier nhanh trên tệp âm thanh lab_female.wav

c) Khảo sát sự ảnh hưởng của tham số PRODUCTS:

Tham số này quyết định xem thuật toán sẽ nén bao nhiêu lần để cuối cùng sinh ra đỉnh cao nhất tại vị trí của tần số cơ bản của khung tín hiệu. Như kết quả khảo sát ở dưới, mức nén nên nằm trong khoảng từ 5 đến 10 để đạt độ chính xác của kết quả cao nhất.

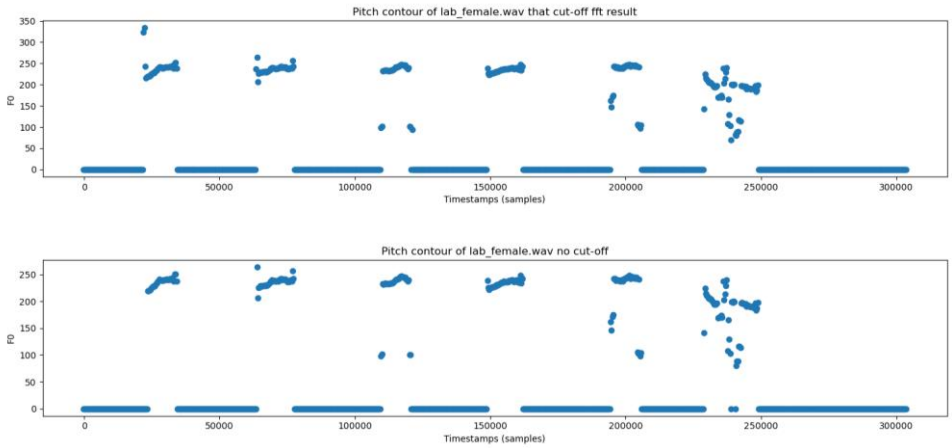


Hình 24. Sự thay đổi của kết quả tìm tần số cơ bản sử dụng HPS khi tăng dần mức độ nén phổ cuối từ 3, 5, 10, 15

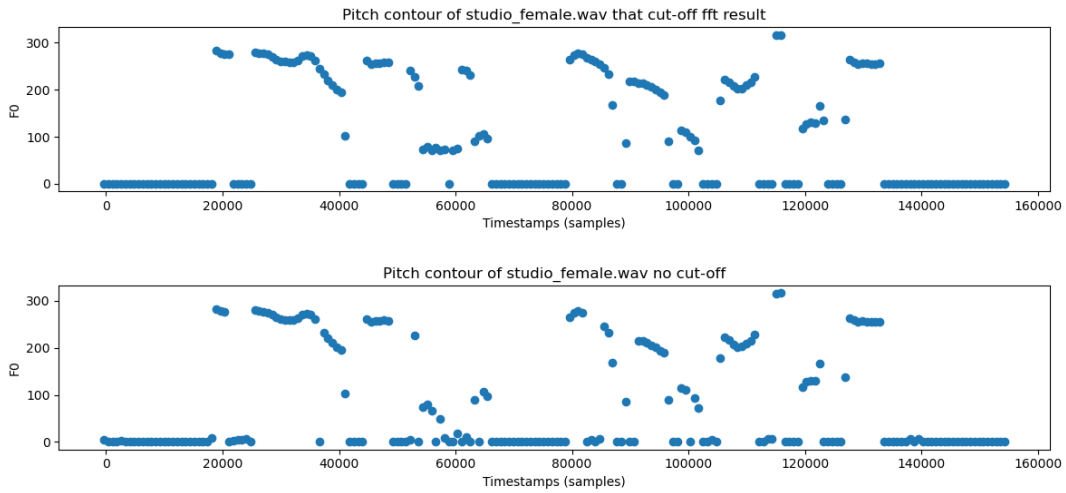
d) Khảo sát sự ảnh hưởng của tham số LOW_PASS và HIGH_PASS

Về các tham số này có kết quả khảo sát tương tự việc khảo sát hai tham số f_max và f_min ở hàm vi sai biên độ trung bình, do độ dài của báo cáo có hạn nên không lặp lại khảo sát của hai tham số này.

e) So sánh đánh giá tính chính xác của hệ thống HPS đã cải thiện cắt tần số và HPS ở bài báo gốc tại [1]



Hình 25. Trong môi trường lab, người nói là nữ nói những nguyên âm đơn giản, thì kết quả của việc có cắt đi hay không cắt cũng không thay đổi nhiều.



Hình 26. Tuy nhiên khi đánh giá file studio female, khi người nữ nói một câu dài và nhiều âm hơn, thì chương trình có phần cắt đi phổ của các tần số cao và thấp đã cho ra một kết quả đáng tin cậy hơn, khi cho ra các giá trị liền mạch và chính xác hơn.

Từ hai hình ảnh khảo sát trên, ta có thể nhận thấy rằng, trong trường hợp những âm thanh người nói đơn giản như các nguyên âm, thì việc cắt bớt phổ sẽ không ảnh hưởng nhiều đến kết quả cuối cùng của thuật toán. Nhưng trong trường hợp phân tích câu dài và liền mạch hơn, đi cùng với nhiều âm khác nhau, việc cắt bớt phổ đã giúp cải thiện độ chính xác của kết quả sinh ra cuối cùng.

4. Đánh giá các thuật toán trên sai số và thời gian thực thi

Bảng 1. Bảng đánh giá sai số của các thuật toán trên trên cùng một khung có tiếng nói của hai file âm thanh lab male và lab female.

Khung của file	Tính thủ công	Hàm tự tương quan	Hàm vi sai biên độ trung bình	Hàm HPS trên miền tần số
lab_male.wav	116.2791 Hz	117.6471 Hz	117.6988 Hz	116.3321 Hz
lab_female.wav	226.0185 Hz	227.5992 Hz	231.8848 Hz	227.9035 Hz

Từ đây ta có thể thấy các hàm tự tương quan có độ chính xác cao nhất nếu so sánh với kết quả quan sát thủ công, hàm vi sai biên độ trung bình thì nhạy hơn với trường hợp lab male, có tần số cơ bản thấp hơn. Hàm HPS trên miền tần số có kết quả tính toán phụ thuộc vào độ phân giải của phép biến đổi fourier nhanh, trong trường hợp này sử dụng độ phân giải là 2^{17}, cũng cho ra kết quả có độ chính xác cao.

Bảng 2. Bảng đánh giá thời gian thực thi của các thuật toán trên cùng một hệ thống và phân tích file lab_female.wav

Khung của file	Thời gian thực thi
Hàm tự tương quan	28.5358 s
Hàm vi sai biên độ trung bình	5.9144 s
Hàm HPS trên miền tần số	0.8811 s

Từ đây, ta có thể kết luận rằng thuật toán của hàm tự tương quan có độ phức tạp tính toán cao, thuật toán của hàm vi sai biên độ trung bình đã cải thiện làm giảm đi thời gian chạy của thuật toán đi đáng kể giúp nó có thể ứng dụng để phát hiện tần số cơ bản trong thời gian thực. Phương pháp HPS trên miền tần số đã là một cải tiến đáng kể so với hai thuật toán còn lại.

V. KẾT LUẬN

Hàm tự tương quan là một thuật toán tương đối phức tạp về mặt cài đặt cũng như độ phức tạp tính toán. Hàm vi sai biên độ trung bình là một thuật toán được cải tiến từ hàm tự tương quan nên ưu điểm của nó là có độ phức tạp thấp và độ chính xác cao trong việc tìm tần số cơ bản của tín hiệu. Tuy nhiên điểm yếu nhất vẫn là khó xác định được tần số trong môi trường nhiễu cao. Do đó cần phải có những biện pháp lọc nhiễu để tăng sự chính xác của thuật toán.

Khác với hai hàm còn lại, phương pháp sử dụng phép biến đổi Fourier nhanh và giảm bậc hài kết quả của nó để tìm tần số cơ bản của tín hiệu có độ phức tạp để khi cài đặt cũng như tính toán khá thấp, có thể ứng dụng để chạy thời gian thực để tìm kiếm tần số cơ bản của giọng nói. Tuy vậy phép tính này còn dựa nhiều vào điều kiện môi trường đầu vào để cho ra kết quả chính xác nhất.

Về hướng phát triển của báo cáo này, có thể thực hiện cài đặt thêm các hàm làm mượt để chính xác hóa và loại bỏ các giá trị đột biến. Và sử dụng kết quả của các thuật toán này giúp phân biệt các âm khác nhau trong câu nói Tiếng Việt.

VI. TÀI LIỆU THAM KHẢO

[1] Au Gareth Middleton, Pitch Detection Algorithms. OpenStax CNX. Dec 18, 2003 http://cnx.org/contents/8b900091-908f-42ad-b93d-806415434b46@2

[2] Vo, Thanh & Sawada, Hideyuki. (2017). Singing performance of the talking robot with newly redesigned artificial vocal cords. 665-670. 10.1109/ICMA.2017.8015895.

[3] Saad Ahmad. FFT Spectral Leakage and Windowing http://saadahmad.ca/fft-spectral-leakage-and-windowing/

[4] A Signal Period Detection Algorithm Based on Morphological Self-Complementary Top-Hat Transform and AMDF https://www.mdpi.com/2078-2489/10/1/24/htm

[5] Trần Văn Tam, xác định tần số cơ bản của tín hiệu tiếng nói dùng hàm tự tương quan. Đà Nẵng, 2019.