مقدمه

صنعت کامپیوتر که راهبری سیستمهای اطلاعاتی، مالی و امنیتی نوین را بر عهده دارد برای اعمال کنترل صحیح بر نحوه‌ی دسترسی به سیستمهای یاد شده توسط کاربران، سالهاست که از سیستم ساده‌ی نام کاربر و کلمه‌ی عبور استفاده می‌نماید. با وجود آن که سیستم مزبور از نظر پیاده‌سازی بسیار ساده و کم‌هزینه می‌باشد، به این دلیل که تضمین نمی‌کند شخص دارنده‌ی یک شناسه‌ی مجاز همان شخصی است که شناسه‌ی مزبور برای استفاده‌ی اختصاصی او در نظر گرفته شده، امنیت مورد انتظار برای سیستمهای مهم را تأمین نمی‌نماید. آمار رو به‌فزون تقلبهای کامپیوتری به صورت گذر غیر مجاز از سیستم یاد شده نشانگر این عدم توانایی‌ست که انگیزه‌ی جستجوی جایگزینی مطمئن برای آن را در ذهن متخصصان امر پرورده است.

از سوی دیگر تکنیک استفاده از ویژگیهای منحصر به فرد زیستی انسانها برای بازشناسی آنان دیرزمانیست که در حیطه‌ی جرم‌شناسی به لحاظ باور علمی این مطلب که این ویژگیها غیر قابل تقلید بوده، احتمال مشابهت آنها در افراد صفر یا عددی مشابه آن است به عنوان ابزاری مطمئن و کارآمد مطرح می‌باشد و متخصصان طراحی سیستمهای امنیتی الکترونیکی نیز در دهه‌های اخیر به این فن‌آوری به عنوان مهم‌ترین اساس برای طراحی سیستمهای امنیتی وابسته به فرد خاص نظر داشته‌اند.

اهمیت موضوع یاد شده از یک سو و تازگی آن از سوی دیگر تهیه‌کننده‌ی این نوشتار را بر آن داشت تا با راهنمایی استاد محترم جناب آقای دکتر نقش‌نیلچی موضوع پروژه‌ی پایانی خود را مطالعه بر روی این گونه سیستمها و طراحی یک سیستم امنیتی نمونه بر اساس ویژگی‌های منحصر به فرد صوتی افراد انتخاب نماید که دستاورد آن یک سیستم تشخیص گوینده‌ی وابسته به متن به عنوان نتیجه‌ی عملی و این نوشتار به عنوان فراهم آورنده‌ی پیش‌زمینه‌های علمی و توصیف‌کننده‌ی نحوه‌ی عملکرد آن می‌باشد.

پیش از هر چیز تأکید بر این مطلب ضروری به نظر می‌رسد که این نوشتار نه به عنوان یک مرجع برای موضوع طراحی سیستمهای تشخیص گوینده بلکه به عنوان راهنمایی برای افراد علاقمند به ادامه‌ی این کار فراهم آمده است و از این لحاظ سعی شده که به جای پوشش تمامی روشهای ممکن و ارائه‌ی الگوریتمهایی که در پروژه‌ی عملی کاربردی نداشته‌اند از آنها به صورت اشاره‌وار یاد شود و در عوض زمینه‌های تئوری و روشهای برنامه‌نویسی عملاً استفاده شده، به صورت مشروح‌تر بیان شده‌اند. ذکر فهرست منابع هر فصل در پایان آن، توضیح کد توابع استفاده شده و ارائه‌ی پیش‌زمینه‌های نظری و روشهای برنامه‌نویسی برای پردازش صوت به صورت ابزارهایی برای تحقق این هدف به کار گرفته شده‌اند که به نظر می‌رسد نمی‌توانند جایی در بین مطالب یک مرجع برای مطلب یاد شده داشته باشند.

بدنه‌ی اصلی نوشتار از سه بخش تشکیل شده است. بخش اول مقدمه‌ای است بر اهمیت و کاربردهای سیستمهای زیست‌سنجی و به طور خاص سیستمهای تشخیص گوینده. این بخش شامل دو فصل است و مطالب آن عمدتاً مقدمات لازم برای بخش سوم نوشتار را فراهم می‌آورد. در بخش دوم پردازش صوت را به صورت نظری مورد بررسی قرار خواهیم داد و سپس مطالب نظری یاد شده را در قالب یک کتابخانه برای پردازش صوت از دیدگاه برنامه‌نویسی پیاده‌سازی خواهیم نمود. این بخش تا حد زیادی مستقل از دو بخش دیگر نوشتار می‌باشد ولی به لحاظ نوع مخاطب و هدف این نوشتار پرداختن به آن ضروری به نظر می‌رسد. بخش یاد شده نیز شامل دو فصل می‌باشد. بخش آخر که مرتبطترین بخش نوشتار با سیستم عملی است و دربردارنده‌ی چهار فصل است پردازش صحبت را از دیدگاه تئوری و برنامه‌نویسی مورد بحث قرار می‌دهد. بحث با فصلی در مورد پردازش سیگنال صحبت که در واقع به نوعی می‌تواند ادامه‌ی بخش قبل تلقی گردد آغاز می‌گردد و در ادامه مدلسازی سیگنال به عنوان شیوه‌ای برای استخراج الگوهای قابل مقایسه از سیگنال صحبت مطرح می‌شود و در آخرین فصل نظری این نوشتار روشهای مختلف معمول برای طراحی سیستمهای تشخیص گوینده بررسی می‌شوند. نهایتاً در آخرین فصل این نوشتار یک کتابخانه‌ی نمونه‌ی تشخیص گوینده که در پیاده‌سازی قسمت عملی مورد استفاده قرار گرفته است ارائه می‌گردد.

در ضمیمه‌ی اول این نوشتار شیوه‌ی نصب و روش استفاده از سیستم پیاده‌سازی شده مطرح گردیده است و ضمیمه‌ی دوم آن ارائه‌دهنده‌ی فهرست کاملی از منابع متفرقه‌ی اینترنتی یافت شده که بسیاری از آنها به دلایل مختلف در این پروژه مورد استفاده قرار نگرفته‌اند برای خوانندگانی که قصد انجام کاری مشابه با این پروژه را دارند می‌باشند.

با توجه به آن که منابع عمده‌ی این نوشتار، عمدتاً انگلیسی زبان می‌باشند ارائه‌ی معادلهای مناسبِ حتی‌الامکان فارسی برای اصطلاحات علمی که در منابع فارسی معادلی برای آنها یافت نشده سرلوحه‌ی کار فراهم‌آورنده‌ی این نوشتار قرار داشته و به منظور انتقال هر چه بهتر مطلب سعی شده با ارائه‌ی پاورقیها یا توضیحات اضافی، مخاطب نوشتار با اصل اصطلاحات آشنا شود. علاوه بر آن در مواردی که ارائه‌ی واژه به صورت انگلیسی مناسب‌تر تشخیص داده شده واژه با حروف فارسی در متن اصلی به کار برده شده تا تناسب متن حفظ گردد. اغلبِ اصطلاحات مخفف شده به صورت اخیر آورده شده‌اند. البته در فصلهای مرتبط با برنامه‌نویسی به لحاظ نوع مطلب شیوه‌ی دیگری متناسب با محتوای آنها به کار گرفته شده است. در مجموع ضمن آن که تلاش شده مطلب به صورت واضح بیان گردد فراهم‌آورنده‌ی این نوشتار تلاش برای ارائه‌ی اصطلاحات درست فارسی و رعایت قوانین این زبان توانمند را همواره در نظر داشته است.

با وجود آن که طراحی سیستمهای تشخیص گوینده مدتهاست مد نظر کارشناسان قرار دارد و با وجود ارائه‌ی نتایج بسیاری از این تحقیقات به صورت سیستمهای تجاری، هنوز سیستمی که بتواند مستقل از محیط و نوع آموزش کاربران عملکرد مناسبی داشته باشد ارائه نشده و این زمینه هنوز هم به عنوان یک افق علمی باز و دارای زمینه‌ی تحقیقاتی فراوان مطرح می‌باشد. با وجود آن که محدودیتهای زمانی و ... مانع از ارائه‌ی یک کار دلخواه فراهم‌آورنده‌ی این نوشتار و استاد راهنمای محترم شد امیدواریم این نوشتار بتواند مقدمه‌ای برای انجام کارهای کامل‌تر فراهم آورد ضمن آن که علاوه بر موضوع طراحی سیستمهای تشخیص گوینده بسیاری از مطالب این نوشتار می‌توانند مقدمه‌ای کاملاً ارضا کننده و مناسب برای افراد علاقمند به طراحی سیستمهای تشخیص صحبت فراهم آورند.

امید است که تلاشهای فراهم‌آورنده‌ی این نوشتار و زحمات بی‌دریغ استاد راهنمای محترم بتواند پیش‌زمینه‌های لازم برای ارائه‌ی کارهای کامل‌تر در نوع خود را فراهم آورد.

فصل اول - زیست‌سنجی و کاربردهای آن در سیستمهای امنیتی

۱- تعریف، ضرورت و کاربردها

زیست‌سنجی[۱] عبارت است از دانش و فن‌آوری اندازه‌گیری و تحلیل آماری داده‌های زیستی.[۲] در فن‌آوری اطلاعات واژه‌ی زیست‌سنجی به مجموعه فن‌آوریهایی اطلاق می‌گردد که در آنها از اندازه‌گیری و تحلیل ویژگیهایی از بدن انسان همچون اثر انگشت، اثر کف دست، شبکیه و عنبیه‌ی چشم، الگوهای صوتی، الگوهای مربوط به رخسار ، دمانگاری صورت[۳]، شکل دست یا گوش، داده‌های به دست آمده از گام، الگوهای وریدی[۴]، دی.ان.ای[۵] و یا ویژگیهایی همچون دستخط (امضا) و دینامیک ضربه زدن به صفحه‌کلید[۶] برای تأیید هویت[۷] اشخاص استفاده می‌شود.[۸] این فن‌آوریها در تلاشند تا اندازه گیری و مقایسه‌ی ویژگیهای برشمرده شده را به منظور بازشناسی افراد به صورت خودکار درآورند.

فن‌آوریهای زیستی در ابتدا برای کاربردهای تخصصی نیازمند امنیت بالا پیشنهاد شدند اما اینک به عنوان عناصر کلیدی در توسعه‌ی تجارت الکترونیک و سیستمهای برخط[۹] و به همان صورت برای سیستمهای امنیتی نابرخط[۱۰] و سیستمهای امنیتی منفرد[۱۱] مطرح می‌باشند.

این فن‌آوریها اجزاء مهمی را برای تنظیم و نظارت بر نحوه‌ی دسترسی و حضور در سیستم فراهم می‌آورند. محدوده‌های عمده‌ی کاربرد این فن‌آوریها عبارتند از : تجارت الکترونیک، نظارت امنیتی، دسترسی به پایگاه داده‌ها، کنترل مرزها و مهاجرت، تحقیقات قضایی و پزشکی از راه دور[۱۲].

توسعه‌ی فن‌آوریهای زیست‌سنجی فراتر از کاربردهای سنتی نیازمند امنیت بالا، یک اجبار نشأت گرفته از انگیزه‌های مالی است. امنیت معاملات برای آینده‌ی توسعه‌ی تجارت الکترونیک یک مسأله‌ی حیاتی است و نگرانیهای فراوانی درباره‌ی راه حلهای فعلی وجود دارد. مشکل شماره‌های شناسایی شخصی[۱۳] و شناسه‌های هویتی[۱۴] - مانند کارتها- این است که آنها صحت هویت شخصی را که از آنها استفاده می‌کند تأیید نمی‌کنند. آمارها میزان زیان ناشی از تقلب را به طور سالیانه برای کارتهای اعتباری بالغ بر چهارصد و پنجاه میلیون

دلار و برای خودپردازها[۱۵] حدود سه‌میلیارد دلار برآورد می‌کنند. برتری سیستمهای مبتنی بر زیست‌سنجی آن است که به شدت به ویژگیهای فردی اشخاص وابسته‌اند و به راحتی نمی‌توانند مورد سوء استفاده قرار گیرند.

۲- بررسی عملکرد سیستمهای موجود

فعالیتهای انجام شده تا به حال منجر به ظهور ماشینهای گران قیمت زیست- سنجی شده است که علاوه بر قیمت زیاد معمولاً از لحاظ سرعت و عملکرد مناسب نیستند یا حداقل برای دستیابی به عملکرد مناسب باید محیط استفاده‌ی آنها شرایط خاصی را داشته باشد و یا کاربران آنها آموزشهای گسترده‌ای را گذرانده باشند.

در حالی که بعضی از فن‌آوریهای زیست‌سنجی در قالب تولیدات تجاری به بازار عرضه شده‌اند بسیاری از این دسته فن‌آوریها در مرحله‌ی تحقیق و آزمایش قرار دارند. فن‌آوریهای مزبور نیازمند کارهای مطالعاتی بیشتر برای افزایش پایداری و بهبود عملکردشان برای استفاده در کاربردهای ویژه هستند.

پایداری در برابر تقلب ،دقت عملکرد، سرعت و تجهیزات مورد نیاز، همخوانی با سخت‌افزار و نرم‌افزار موجود، هزینه ،سادگی استفاده و پذیرش از سوی کاربر از جمله عوامل تعیین‌کننده در موفقیت هر یک از فن‌آوریهای به کار گرفته شده می‌باشند.

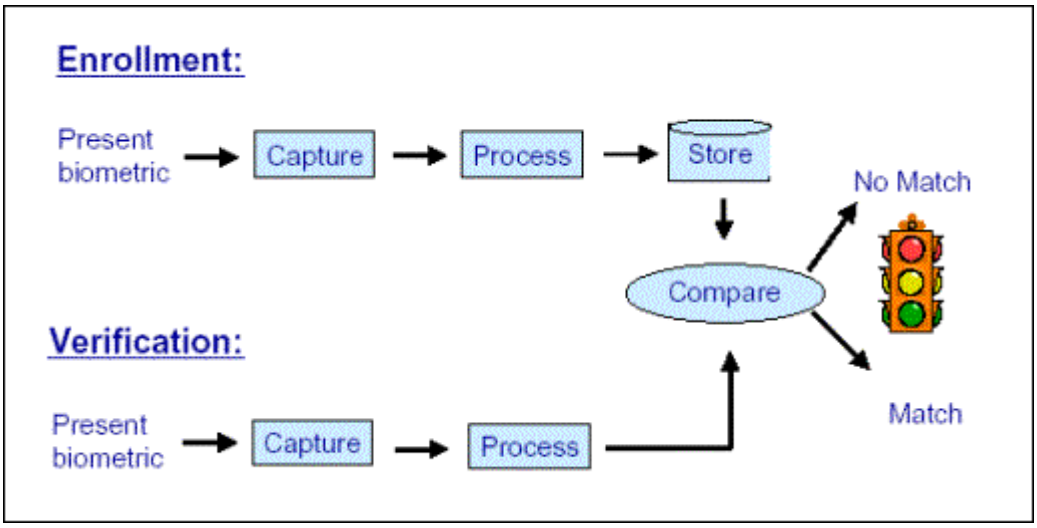
جدول شماره‌ی ۱ مقایسه‌ای از معمول‌ترین سیستمهای زیست‌سنجی موجود را ارائه می‌دهد

نوع سیستم	دقت عملکرد	سادگی استفاده	میزان پذیرش کاربر
اثر انگشت	بالا	متوسط	پایین
هندسه‌ی دست	متوسط	بالا	متوسط
صوت	متوسط	بالا	بالا
شبکیه‌ی چشم	بالا	پایین	پایین
عنبیه‌ی چشم	متوسط	متوسط	متوسط
امضا	متوسط	متوسط	بالا
رخسار	پایین	بالا	بالا

جدول شماره ۱ - مقایسه‌ی سیستمهای زیست‌سنجی معمول (منبع شماره ۱)

۳- اجزای سیستمهای زیست‌سنجی

عملیات سیستمهای زیست‌سنجی در بر دارنده‌ی دو مرحله‌ی مجزا می‌باشد: ثبت کاربر و بازشناسی کاربر. در مرحله‌ی اول اطلاعات مربوط به کاربر به سیستم وارد می‌شوند و در مرحله‌ی دوم اطلاعات ورودی حاضر با اطلاعات ذخیره شده مقایسه می‌گردند.



شکل شماره ۱ - مراحل لازم عملیاتی در یک سیستم امنیتی مبتنی بر زیست‌سنجی (منبع شماره ۳)

مرحله‌ی تأیید هویت[۱۶] عبارت است از تطبیق ویژگیهای مورد ادعای یک شخص بر ویژگیهای موجود او در پایگاه داده‌ها که یک فرایند یک به یک است.

سیستمهای امنیتی مبتنی بر زیست‌سنجی بنا به انتخاب به وجود آورنده، به جای مرحله‌ی تأیید هویت می‌توانند مرحله‌ی دیگری را که بازشناسی[۱۷] نامیده می‌شود جایگزین کنند. در این روش نیاز نیست که درخواست کننده ادعای هویت شخص خاصی را بنماید بلکه سیستم ویژگیهای او را با تمامی رکوردهای موجود مقایسه می‌کند و در صورت تطابق با یکی از آنها او را به عنوان شخص دارای ویژگیهای موجود در رکورد یافت شده بازشناسی می‌کند که این فرایند یک پردازش یک به چند را شکل می‌دهد.

سیستمهای تشخیص هویت زیستی معمول غالباً شامل اجزای زیر می‌باشند:

الف)گیرنده‌ی اطلاعات[۱۸]: زیرسیستمی است که گرفتن نمونه‌های زیست‌سنجی (صوتی، تصویری و...) را بر عهده دارد. ویژگیهای خاص استخراج شده از نمونه‌ها قالبهایی را برای مقایسه‌ی بعدی تشکیل می‌دهند. این فرایند باید سریع و ساده بوده در عین حال قالبهایی با کیفیت خوب را تولید کند.

ب) ذخیره کننده[۱۹]: قالبهای به دست آمده باید برای مقایسه‌ی بعدی ذخیره شوند. این زیر سیستم می‌تواند جزئی از وسیله‌ی گیرنده‌ی اطلاعات سیستم باشد و یا در یک سرور مرکزی قابل دستیابی توسط یک شبکه جای گیرد. جایگزین دیگر، یک شناسه‌ی قابل حمل نظیر یک کارت هوشمند[۲۰] است. هر کدام از انتخابهای فوق مزایا و مشکلات خاص خود را دارد.

ج) مقایسه گر[۲۱]: اگر سیستم زیست‌سنجی در مقام بازشناسی افراد به کار گرفته شود باید هویت شخص با قالب ذخیره شده‌ی مورد ادعای او مقایسه شود. در بعضی سیستمها ممکن است امکان بروزآوری خودکار قالب مورد مراجعه پس از هر تطبیق درست وجود داشته باشد. این امر به سیستم توانایی سازگاری با تغییرات تدریجی کوچک در ویژگیهای کاربر را می‌دهد.

د) اتصالات[۲۲]: غالباً برای ایجاد ارتباط بین گیرنده‌ی اطلاعات، ذخیره کننده و مقایسه‌گر نیاز به اتصالات لازم وجود دارد. غالباً سیستمهای زیست‌سنجی نیازمند شبکه و رابطهای برنامه‌نویسی مورد نیاز برای ایجاد اتصال بین اجزاء می‌باشند. امنیت و کارایی، عناصر کلیدی برای این جزء می‌باشند.

۴- ارزیابی کارایی سیستمهای امنیتی مبتنی بر زیست‌سنجی

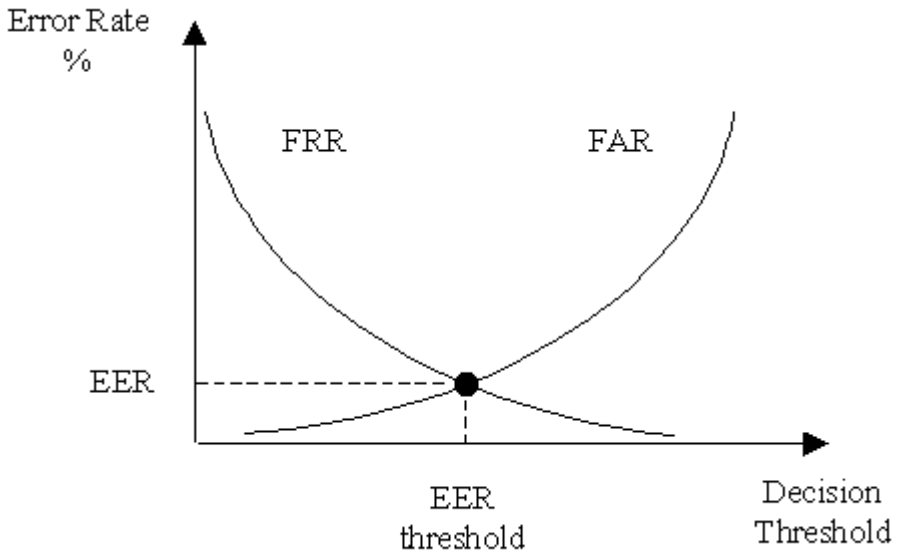
موضوع مهمی که در پذیرش سیستمهای زیست‌سنجی از اهمیت شایان توجهی برخوردار است تعیین کارایی هر یک از اجزاء و کل سیستم زیست‌سنجی به روشی قابل اعتماد و هدفمند است.

برای تعیین کارایی سیستمهای امنیتی مبتنی بر زیست‌سنجی معیارهای ویژه‌ای به کار گرفته می‌شوند. در این کاربردها تعدادی کاربر (سرویس‌گیرنده)[۲۳] به سیستم وارد می‌شوند و متقلب[۲۴] به عنوان شخصی تعریف می‌شود که مدعی هویت شخص دیگری است. متقلب ممکن است به عنوان کاربر در سیستم وجود داشته باشد و عمل وی ممکن است عمدی یا غیرعمدی باشد. عمل تأیید هویت باید کاربران را بپذیرد و متقلبان را رد کند.

نرخ پذیرش نادرست (اف. ای. آر)[۲۵] به عنوان نسبت تعداد متقلبانی که به اشتباه توسط سیستم پذیرفته شده‌اند به تعداد کل متقلبان آزمایش شده تعریف گردیده، به صورت درصد بیان می‌شود. این نرخ، احتمال پذیرش متقلبان را توسط سیستم بیان می‌کند و باید در سیستمهای نیازمند امنیت بالا کمینه شود.

نرخ عدم پذیرش نادرست (اف. آر. آر)[۲۶] به عنوان نسبت تعداد کاربران سیستم که به اشتباه توسط سیستم پذیرفته نشده‌اند به تعداد کل کاربران مورد آزمایش قرار گرفته تعریف گردیده، به صورت درصد بیان می‌شود. این نرخ، احتمال عدم پذیرش کاربران مجاز را توسط سیستم بیان می‌کند و باید به صورت ایده‌آل مخصوصاً در سیستمهایی که در آنها کاربر در صورت عدم پذیرش از دسترسی به سیستم محروم می‌شود کمینه گردد.

روند تشخیص هویت مبتنی بر زیست‌سنجی دربردارنده‌ی محاسبه‌ی فاصله‌ی قالب ذخیره شده و نمونه‌ی حاضر است. تصمیم برای پذیرش یا رد نمونه‌ی حاضر بر اساس یک آستانه‌ی[۲۷] از پیش تعریف شده اتخاذ می‌گردد. بنابراین واضح است که کارایی سیستم به شدت وابسته به انتخاب این آستانه است و این امر موجب ایجاد یک بده‌بستان بین نرخ پذیرش نادرست و نرخ عدم پذیرش نادرست می‌گردد. نرخ خطای برابر (ای.ای.آر)[۲۸] به صورت آستانه‌ی برابری این دو نرخ تعریف می‌شود و غالباً به عنوان یک ویژگی نشان دهنده‌ی کارایی سیستم مطرح می‌گردد. شکل شماره‌ی ۲ نشان دهنده‌ی رابطه‌ی سه پارامتر تعریف شده برای یک سیستم نمونه است.



شکل شماره ۲- FAR، FRR و ERR برای یك سیستم نمونه (منبع شماره ۱)

پارامتر مهم دیگر کارایی، زمان تشخیص هویت[۲۹] است که به صورت زمان متوسط صرف شده برای فرایند تشخیص هویت تعریف می‌شود. این زمان شامل زمان لازم برای گرفتن نمونه‌ی حاضر نیز می‌باشد.

در حالی که بعضی از عرضه‌کنندگان سیستمهای امنیتی مبتنی بر زیست‌سنجی برای محصولاتشان پارامترهای کارایی فوق را در شرایط آزمایشگاهی بیان می‌کنند پارامترهای کارایی قابل طرح در جهان واقعی برای سنجش کارایی واقعی این گونه سیستمها به ندرت وجود دارند. علت این امر این واقعیت است که به حساب آوردن همه‌ی پیچیدگیهای ممکن جهان واقعی تأثیر گذار بر سیستمهای زیست‌سنجی تقریباً غیر ممکن است. به عنوان نمونه زمان واقعی تشخیص هویت به شدت وابسته به میزان آموزش کاربر، محیط عملیاتی و

شرایط روانی کاربر همچون میزان فشار روحی اوست. مشخصات ارائه شده توسط عرضه‌کننده را باید به دید راهنماهای نه چندان متناسب با دنیای واقعی نگریست.

۱- تعریف و کاربردها

تشخیص گوینده[۱] عبارت است از فرایند تشخیص خودکار هویت شخص صحبت‌کننده بر اساس اطلاعات یکتای موجود در موج صوتی صحبت او.

این فن‌آوری امکان تشخیص هویت شخص گوینده و در نتیجه امکان کنترل دسترسی او در هنگام استفاده از خدماتی همانند شماره‌گیری صوتی، بانکداری تلفنی، خرید تلفنی، خدمات دسترسی به پایگاه داده‌ها، خدمات اطلاعاتی، پست الکترونیکی صوتی، کنترل امنیتی برای ورود به قلمروهای اطلاعاتی محرمانه و دسترسی از راه دور به کامپیوترها را فراهم می‌آورد. علاوه بر موارد فوق که عموماً با کامپیوتر و کاربران آن سروکار دارند این فن‌آوری در مسائل قضایی نیز کاربردهای خاص خود را دارد.

۲- انواع سیستمهای تشخیص گوینده

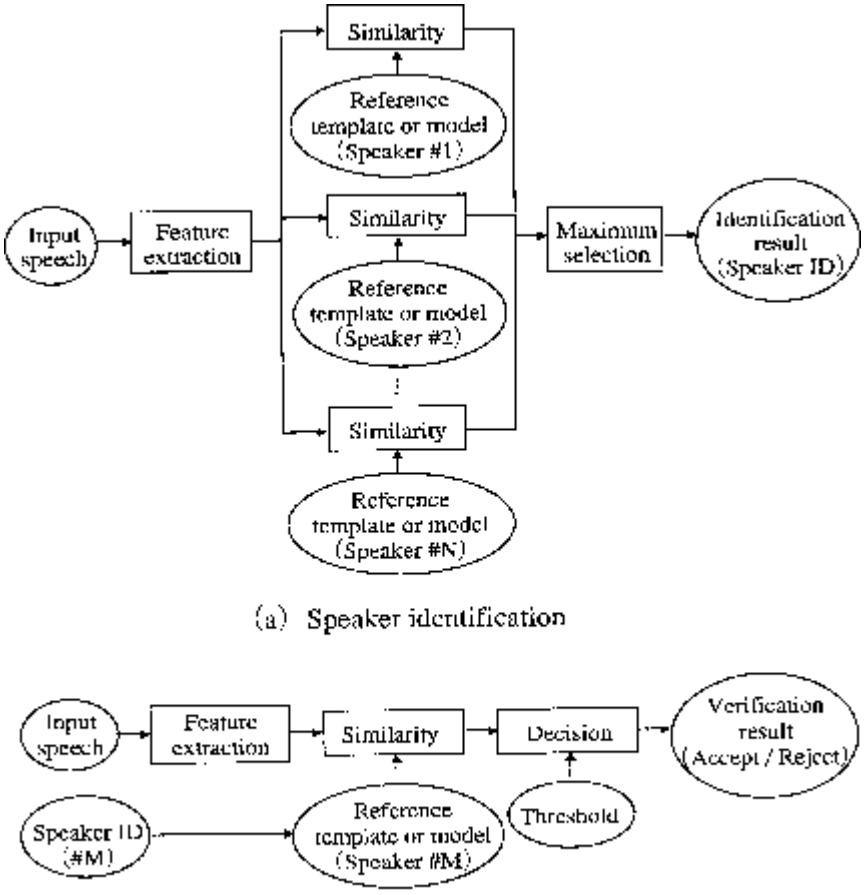
سیستمهای تشخیص گوینده از لحاظ روش استفاده، همانند آنچه برای کلیه‌ی سیستمهای امنیتی مبتنی بر زیست‌سنجی در فصل پیش بیان شد، عموماً در دو دسته‌ی سیستمهای تأیید هویت گوینده[۲] و سیستمهای بازشناسی هویت گوینده[۳] قرار می‌گیرند.

در یک سیستم تأیید هویت گوینده، شخص عموماً با انتخاب یا وارد کردن نام یکی از کاربران خاص سیستم ادعا می‌کند که او همان کاربر ثبت‌شده‌ی سیستم است. در این حالت سیستم وظیفه دارد ویژگیهای صوتی شخص مدعی را با ویژگیهای صوتی ذخیره شده‌ی کاربر ثبت شده‌ی مورد ادعا مقایسه نموده و با استفاده از نتیجه‌ی به دست آمده ادعای شخص را بپذیرد یا رد کند.

در یک سیستم بازشناسی هویت گوینده، شخص صحبت کننده ادعای هویت یک کاربر خاص ثبت شده را نمی‌نماید و این سیستم است که وظیفه دارد که او را در میان کاربران ثبت شده‌ی سیستم بازشناسی نماید و یا تشخیص دهد که ویژگیهای صوتی او با هیچ یک از کاربران ثبت شده همخوانی ندارد.

به نظر می‌رسد در آینده کاربردهای سیستمهای نوع دوم در سیستمهای بزرگ چند کاربره چشمگیرتر از کاربردهای سیستم نوع اول باشد،[۴] هر چند که در اساس این دو سیستم تفاوتهای چشمگیری مشاهده نمی‌شود.

شکل شماره‌ی ۱ ساختار اساسی این دو نوع سیستم تشخیص گوینده را به تصویر می‌کشد.



شکل شماره‌ی ۱- ساختار اساسی سیستمهای بازشناسی هویت و تأیید هویت گوینده (منبع شماره‌ی ۱)

سیستمهای تشخیص گوینده از دیدگاه دیگری به دو دسته‌ی سیستمهای تشخیص گوینده‌ی وابسته به متن[۵] و سیستمهای تشخیص گوینده‌ی مستقل از متن[۶] تقسیم می‌شوند. روش اول نیازمند آن است که گوینده کلمات کلیدی یا جمله‌های ثابتی را چه در مرحله‌ی یادگیری و چه در آزمونهای تشخیصی بیان کند، در حالی که دومی وابسته به جمله یا کلمه‌ی خاصی نیست.

هر دو روش دارای یک مشکل هستند و آن این است که می‌توان از صدای ضبط شده‌ی کاربران ثبت‌شده برای ورود به سیستم استفاده نمود و به آسانی سیستم را فریب داد. برای غلبه بر این مشکل روشهایی وجود دارند مثلاً می‌توان از یک مجموعه‌ی کوچک از کلمات مانند ارقام به عنوان کلمات کلیدی استفاده نمود و در هر زمان به صورت تصادفی از کاربر خواست که یک دنباله از آنها را بیان کند. حتی این روش هم کاملاً قابل اطمینان نیست چرا که می‌تواند با استفاده از تجهیزات پیشرفته‌ی الکترونیکی که توانایی تولید دنباله‌های عبارات را دارند فریب داده شود. سیستمهای دارای ساختار اخیر به سیستمهای تشخیص گوینده‌ی اعلان متن[۷] (متن تولید شده توسط ماشین) معروفند.

۳- روشهای پیاده‌سازی

تقریباً در تمامی سیستمهای تشخیص هویت با استفاده از فرایندی که به تشخیص الگو[۸] شهرت دارد شباهت هر زوج نمونه نمره‌گذاری می‌شود. استفاده از این روش نیازمند وجود دسته‌ای از خصایص منحصر به فرد و قابل مقایسه که از ویژگی انتخاب شده به عنوان ورودی سیستم استخراج شده می‌باشد.

ویژگیهای فیزیکی افراد نظیر ساختار اندامهای صوتی، اندازه‌ی چاله‌ی بینی و ویژگیهای تارهای صوتی منحصر به فرد بوده و از طریق الگوریتمهای پردازش سیگنال به صورت پارامترهای خصیصه‌ای[۹] یا

مجموعه‌ی خصایص[۱۰] قابل استخراج می‌باشند. این حقیقت پایه‌ی روشهای پیاده‌سازی سیستمهای تشخیص صحبت می‌باشند.

مهمترین گلوگاه سیستمهای تشخیص گوینده (و به تبع هم خانواده بودن مهمترین گلوگاه سیستمهای تشخیص صحبت) نحوه‌ی عملکرد آنها در مکانهای دارای شرایط متفاوت با شرایط آزمایشگاهی که از ویژگیهای عمده‌ی آنها می‌توان به حضور نویز در سیستم اشاره کرد می‌باشد. برای غلبه بر این مشکل از روشهای هنجارسازی[۱۱] استفاده می‌گردد که این روشها نیز انواع مختلفی دارند و در سیستمهای تجاری موجود، اغلب نمود پیدا می‌کنند.

۱ـ دستگاه شنوایی انسان

پردازش صوت محدوده‌های گوناگونی را در بر می‌گیرد که همه به منظور ارائه‌ی صدا به شنوندگان انسانی ابداع شده‌اند. سه محدوده‌ی تکثیر موسیقی با کیفیتی به خوبی اصل همانند آنچه در سی‌دی‌های صوتی وجود دارد ،ارتباط صوتی از راه دور که نام دیگر شبکه‌ی تلفنی است و ،ترکیب صحبت که در آن کامپیوترها الگوهای صوتی انسان را تولید کرده یا تشخیص می‌دهند از دیگر قلمروهای دانش پردازش صوت مهم‌ترند. با وجود این که اهداف و مسائل این کاربردها متفاوتند همگی در یک نقطه‌ی مشترک به هم می‌رسند و آن گوش انسان است.

گوش انسان یک عضو به گونه‌ای فزاینده پیچیده است. قضیه وقتی پیچیده‌تر می‌شود که اطلاعات ارسالی از دو گوش در یک شبکه‌ی پیچیده‌ی گیج کننده که همانا مغز انسان باشد با هم ترکیب می‌شوند. به یاد داشته باشیم که بیان فوق یک گذر کلی بر قضیه است و تعداد زیادی از پدیده‌ها و آثار دقیق مرتبط با گوش انسان هنوز به درستی درک نشده‌اند.

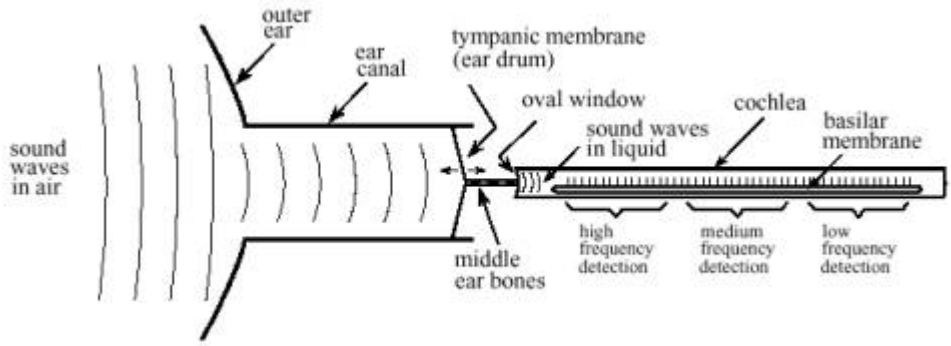
شکل ۱ قسمت اعظم ساختارها و پردازشهایی را که گوش انسان را در بر دارند به تصویر می‌کشد. گوش خارجی از دو بخش تشکیل شده است: نرمی پوست قابل مشاهده و غضروف متصل به کنار سر و کانال گوش که لوله‌ایست به قطر تقریبی ۰,۵ سانتیمتر و تا حدود ۳ سانتیمتر در داخل سر فرو می‌رود. این ساختارها صداهای محیط را به بخشهای حساس گوش میانی و گوش داخلی که در درون استخوانهای جمجمه محافظت می‌شود راهبری می‌کنند. در انتهای کانال گوش یک ورقه‌ی نازک از نسوج که پرده‌ی صماخ[۱] یا طبل گوش نامیده می‌شود کشیده شده است. امواج صدا با برخورد به پرده‌ی صماخ باعث لرزش آن می‌شوند. گوش میانی مجموعه‌ای از استخوانهای کوچک است که لرزش مزبور را به حلزون گوش[۲] (گوش داخلی) انتقال می‌دهند و در آنجا این لرزشها تبدیل به ضربه‌های عصبی می‌گردند. حلزون گوش یک لوله‌ی پر از مایع است که به زحمت قطر آن به ۲ میلیمتر و طول آن به ۳ سانتیمتر می‌رسد. اگر چه حلزون گوش در شکل شماره ۱ به صورت یک لوله‌ی مستقیم نشان داده شده اما در واقع به دور خودش همانند صدف حلزون پیچ خورده است و وجه تسمیه‌ی آن که ریشه در کلمه‌ای یونانی به معنای حلزون دارد نیز این واقعیت است.

وقتی یک موج صوتی سعی دارد از هوا وارد مایع شود تنها کسر کوچکی از آن از بین دو محیط عبور می‌کند و باقیمانده‌ی انرژی آن بازتابیده می‌شود. دلیل این امر مقاومت مکانیکی پایین هوا (ناشی از پایین بودن میزان فشار صوتی و سرعت بالای ذرات هوا که به نوبه‌ی خود از چگالی پایین و تراکم‌پذیری بالای آنها نشأت می‌گیرد) در برابر مقاومت مکانیکی بالای مایع است. به عبارت ساده‌تر دلیل این امر مشابه دلیل این موضوع است که برای ایجاد موج با دست در درون آب به تلاش بیشتری به نسبت انجام این کار در هوا نیازمندیم. تفاوت موجود باعث بازتابش قسمت اعظم صوت در مرز هوا/مایع می‌گردد.

گوش میانی یک شبکه‌ی تطبیق مقاومت[۳] است که کسر انرژی صوتی وارد شده به مایع گوش داخلی را زیاد می‌کند. برای نمونه ماهی پرده‌ی صماخ یا گوش میانی ندارد چرا که نیازی به شنیدن در هوا ندارد. تغییر شدت، بیشتر ناشی از تفاوت مساحت پرده‌ی صماخ (که صدا را از هوا دریافت می‌کند) و دریچه بیضوی[۴] (که مطابق شکل ۱صدا را به داخل مایع انتقال می‌دهد) می‌باشد. مساحت پرده‌ی صماخ حدوداً ۶۰ میلیمتر مربع است حال آن که دریچه‌ی بیضوی حدوداً ۴ میلیمتر مربع مساحت دارد. از آنجا که فشار برابر است با نسبت نیرو به مساحت، این تفاوت مساحت فشار موج صدا را حدوداً ۱۵ برابر افزایش می‌دهد.

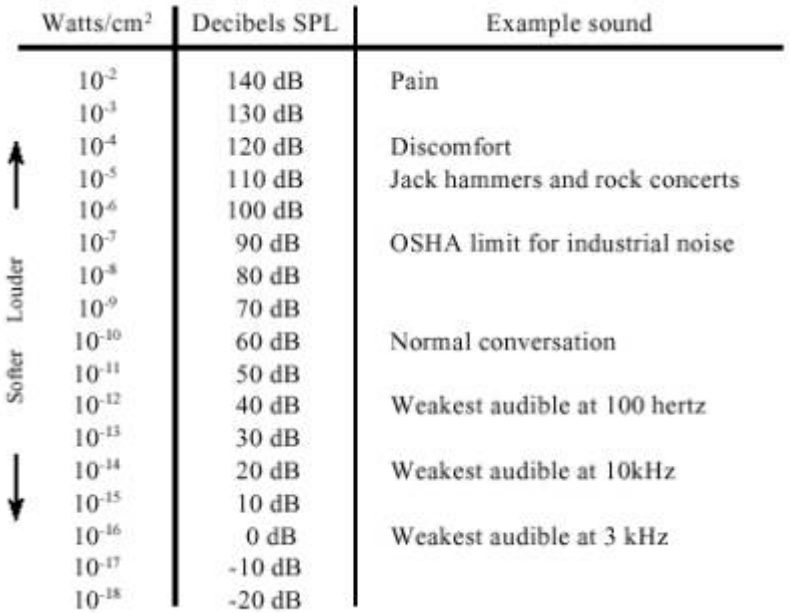
در داخل حلزون گوش پرده‌ی اصلی[۵] قرار دارد که ساختاری را برای ۱۲۰۰۰ سلول حسی که شکل‌دهنده‌ی عصب حلزونی است ایجاد می‌کند. پرده‌ی اصلی در نزدیکی دریچه‌ی بیضوی بسیار سفت است و در انتهای دیگر انعطاف‌پذیرتر است که این امر به این عضو کمک می‌کند تا به عنوان تحلیلگر طیف فرکانسی عمل کند. وقتی پرده‌ی اصلی در معرض یک سیگنال با فرکانس بالا قرار می‌گیرد در قسمت سفت‌تر طنین می‌اندازد که سبب تحریک سلولهای عصبی نزدیک به دریچه‌ی بیضوی می‌گردد. به همین ترتیب فرکانسهای پایین موجب تحریک انتهای دورتر پرده‌ی اصلی می‌شوند. این امر موجب پاسخگویی رشته‌های خاص عصب حلزونی در برابر فرکانسهای خاص می‌گردد. این سازوکار اصل مکان[۶] نامیده می‌شود و در سراسر مسیر به سمت مغز حفظ می‌شود.

طرح کدگذاری اطلاعات دیگری نیز در شنوایی انسان به کار می‌رود که اصل رگبار[۷] نامیده می‌شود. سلولهای عصبی اطلاعات را با تولید پالسهای الکتریکی کوچکی که پتانسیل کنش[۸] نامیده می‌شوند انتقال می‌دهد. یک سلول عصبی واقع بر پرده‌ی پایینی می‌تواند اطلاعات صوتی را با تولید یک پتانسیل کنش در پاسخ هر سیکل لرزش کدگذاری کند. برای نمونه یک موج صدای ۲۰۰ هرتزی می‌تواند توسط یک نورون ایجاد کننده‌ی ۲۰۰ پتانسیل کنش در ثانیه نشان داده شود. در هر صورت این روش تنها در فرکانسهای زیر حدوداً ۵۰۰ هرتز – بالاترین سرعت ممکن تولید پتانسیل کنش در نورونها – به کار می‌آید. گوش انسان برای غلبه بر این مشکل به نورونها اجازه می‌دهد که برای انجام این کار دسته‌جمعی عمل کنند. برای نمونه یک صدای ۳۰۰۰ هرتزی می‌تواند توسط ده سلول عصبی که هر کدام ۳۰۰ ضربه در ثانیه علامت می‌دهند نشان داده شود. این پدیده بازه‌ی کارایی اصل رگبار را تا ۴ کیلوهرتز گسترش می‌دهد که بالاتر از بازه‌ی عملیاتی اصل مکان می‌باشد.



شکل شماره ۱- توضیحات مربوط به شکل: نمودار کارکردی گوش انسان. گوش خارجی امواج صوتی را از محیط می‌گیرد و آنها را به سوی پرده‌ی صماخ (طبل گوش) که ورقه‌ی نازکی از بافت است و هماهنگ با شکل موج هوا می‌لرزد راهبری می‌کند. استخوانهای گوش میانی (استخوانهای چکشی، سندانی و رکابی) این لرزشها را به دریچه‌ی بیضوی که پرده‌ای منعطف واقع در حلزون گوش پر از مایع است انتقال می‌دهند. در داخل حلزون گوش پرده‌ی اصلی قرار دارد که ایجاد کننده‌ی ساختاری برای ۱۲۰۰۰ سلول عصبی شکل‌دهنده‌ی عصب حلزون گوش است. بسته به سفتی متغیر پرده‌ی پایینی، هر سلول فقط به بازه‌ی کوچکی از فرکانسهای صدا پاسخ می‌دهد که این پدیده گوش را تبدیل به یک تحلیلگر طیف فرکانسی می‌نماید.

شکل شماره ۲ رابطه‌ی بین شدت صدا و بلندی مشاهده شده را نشان می‌دهد. غالباً شدت صدا را با یک اندازه‌ی لگاریتمی که دسی‌بل اس.پی.ال.[۹] (سطح توان صدا) نامیده می‌شود نشان می‌دهند. در این معیار ۰ دسی‌بل اس.پی.ال موج صدایی با قدرت ده به توان منفی شانزده وات بر سانتیمتر مربع است که حدوداً ضعیف‌ترین صدای قابل تشخیص توسط گوش انسان است. صحبت معمولی حدوداً ۶۰ دسی‌بل اس.پی.ال است و صدایی با شدت ۱۴۰ دسی‌بل اس.پی.ای برای گوش دردناک و زیان‌آور است.



شکل شماره ۲ - واحدهای شدت صدا. شدت صدا به صورت توان بر واحد مساحت تعریف می‌شود (مثلاً وات بر سانتیمتر مربع) یا به صورت معمول‌تر با استفاده از یک اندازه‌ی لگاریتمی که دسی‌بل اس.پی.ال خوانده می‌شود. همچنان که این جدول نشان می‌دهد قوه‌ی شنوایی انسان بیشتر به صداهای بین ۱کیلوهرتز تا ۴ کیلوهرتز حساس است.

اختلاف بلندترین و ضعیف‌ترین صداهایی که انسان می‌تواند بشنود ۱۲۰ دسی‌بل است که از لحاظ دامنه معادل بازه‌ای حدود یک میلیون است. شنونده تغییر بلندی صدا را وقتی صدا حدود ۱ دسی‌بل (٪۱۲ در دامنه) تغییر کند تشخیص می‌دهد به عبارت دیگر تنها ۱۲۰ سطح بلندی صدا از ملایم‌ترین نجوا تا بلندترین تندر قابل تشخیص است. حساسیت گوش آنقدر جالب توجه است که هنگام شنیدن به ضعیف‌ترین صداها پرده‌ی صماخ به اندازه‌ای کمتر از قطر یک ملکول به لرزش درمی‌آید!

احساس بلندی صدا با توان صدا رابطه‌ی توانی با نمای ۱/۳ دارد. به عنوان نمونه اگر شما توان صدا را ده برابر کنید شنوندگان آن صدا دو برابر شدن بلندی صدا را احساس و گزارش می‌کنند.

این مسأله یک مشکل بزرگ برای حذف صداهای محیطی ناخواسته به وجود می‌آورد. برای نمونه فرض کنید که شما ٪۹۹ دیوار را با عایق صوتی پوشانده‌اید و تنها ٪۱ که مربوط به درها، گوشه‌ها، منافذ و... هستند باقی مانده‌اند. با وجود آن که توان صدا تا اندازه‌ی ٪۱ مقدار اولیه‌ی آن کاسته شده بلندی صدا تنها به اندازه‌ی ٪۲۰ کاهش پیدا کرده‌است.

بازه‌ی شنیداری انسان بین ۲۰ هرتز تا ۲۰ کیلوهرتز در نظر گرفته می‌شود، حال آن که بیشتر صداهای قابل حس در بازه‌ی ۱ کیلوهرتز تا ۴ کیلوهرتز قرار دارند. برای نمونه شنوندگان می‌توانند صدایی به میزان صفر دسی‌بل را در فرکانس ۳ کیلوهرتز بشنوند حال آن که برای شنیدن یک صدای ۱۰۰ هرتزی حداقل مقدار آن باید ۴۰ دسی‌بل باشد. شنوندگان می‌توانند بگویند که دو صدا متفاوتند اگر فرکانس آنها بیش از حدود ٪۰,۳ در ۳ کیلوهرتز متفاوت باشد. به عنوان نمونه کلیدهای کنار هم در پیانو به اندازه‌ی حدود ٪۶ تفاوت فرکانس دارند.

مهم‌ترین مزیت داشتن دو گوش تشخیص جهت صداست. شنوندگان انسانی می‌توانند تفاوت بین دو منبع صدا را که فاصله‌ای به کمی ۳ درجه دارند (حدوداً برابر با عرض یک انسان در فاصله‌ی ده متری) تشخیص دهند. این اطلاعات جهتی به دو روش جداگانه به دست می‌آیند. اولاً فرکانسهای حدوداً بالای ۱ کیلوهرتز به شدت زیر سایه‌ی سر قرار می‌گیرند. به بیان دیگر گوشی که به منبع نزدیک‌تر است سیگنال قوی تری را به نسبت گوشی که در جهت مخالف دارد دریافت می‌کند. روش دیگر تشخیص جهت آن است که گوش دورتر به خاطر فاصله‌ی بیشترش از منبع صدا را کمی دیرتر از گوش نزدیک‌تر دریافت می‌کند. به واسطه‌ی اندازه‌ی معمول

سر (حدوداً ۲۲ سانتیمتر) و سرعت صوت (حدود ۳۴۰ متر در ثانیه) تفاوت‌گذاری زاویه‌ای سه درجه دقت زمانی حدود ۳۰ میکروثانیه نیاز دارد. چون این فاصله‌ی زمانی نیازمند اصل رگبار است این روش جهت‌یابی برای صداهای دارای فرکانس کمتر از حدود ۱ کیلوهرتز به کار می‌رود.

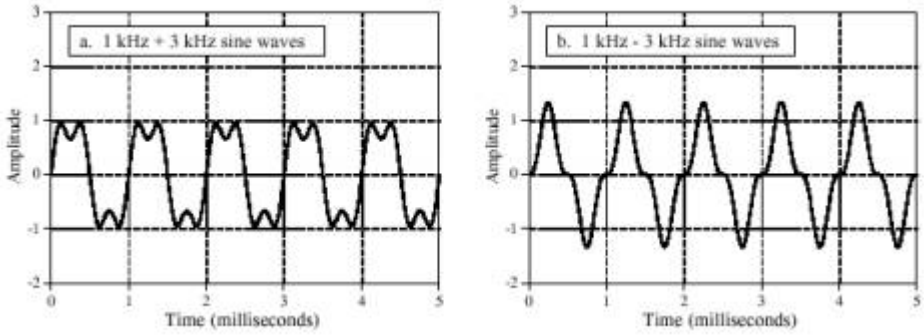
در حالی که قوه‌ی شنوایی انسان می‌تواند جهت صدا را تشخیص دهد در نشخیص فاصله‌ی منبع صدا مشکل دارد. این امر بدان علت است که چیزهای کمی در موج صدا وجود دارد که اطلاعات این گونه را در اختیار بگذارد. شنوایی انسان به صورت ضعیفی در می‌یابد که منابع صداهای با فرکانس بالا نزدیکند و صداهای با فرکانس پایین از فاصله‌ی دورتری پخش می‌شوند. این به آن دلیل است که صداها در فاصله‌های دور از میزان فرکانسشان کاسته می‌شود. پژواک روش ضعیف دیگری برای تشخیص فاصله است و با استفاده از آن مثلاً می‌توان ابعاد یک اتاق را حدس زد. برای نمونه صداهای موجود در یک تالار بزرگ پژواکهایی با وقفه‌ی ۱۰۰ میلی ثانیه دارند، حال آن که برای یک دفتر کار کوچک این مقدار ۱۰ میلی ثانیه است. بعضی از موجودات با استفاده از دستگاه طبیعی تشخیص فاصله‌ی صوتی[۱۰] مسأله‌ی فاصله‌یابی را حل کرده‌اند. مثلاً خفاشها و دلفینها صداهایی مثل تیک و جیغ تولید می‌کنند که از سوی اشیاء نزدیک بازتابیده می‌شوند. با اندازه‌گیری میزان وقفه‌ی بازتاب این صداها این جانوران می‌توانند با دقت ۱سانتیمتر اشیاء را مکانیابی کنند. تجربیات نشان داده‌اند که بعضی انسانها به خصوص نابینایان تا حد کمی از روش مکانیابی با استفاده از پژواک استفاده می‌کنند.

۲- ویژگیهای امواج صوتی

غالباً برای درک یک صوت پیوسته مثل نت یک ابزار موسیقیایی سه بخش مجزا را باید تشخیص داد: بلندی صدا، زیری یا بمی صدا (پیچ[۱۱]) و طنین صدا[۱۲]. بلندی همانگونه که قبلاً توضیح داده شد معیاری برای شدت موج صوتی است. پیچ، فرکانس جزء اصلی صدا – فرکانسی تکرار موج صوتی توسط خودش – می‌باشد.

طنین صدا از دو جزء قبلی پیچیده‌تر است و با تعیین محتوای همساز[۱۳] صدا تعیین می‌گردد. شکل شماره ۳ دو موج را که هر دو از جمع یک موج سینوسی یک کیلوهرتزی با دامنه‌ی یک و یک موج سینوسی سه کیلوهرتزی با دامنه‌ی یک دوم به وجود آمده‌اند نشان می‌دهد. تفاوت آنها در آن است که در شکل b جزء با فرکانس بالاتر ابتدا معکوس شده و سپس با موج دوم جمع شده است. علی‌رغم موجهای در دامنه‌ی زمان بسیار متفاوت این دو صوت یکسان به نظر می‌رسند. این به خاطر آن است که شنوایی انسان بر اساس دامنه‌ی فرکانسهاست و نسبت به فاز آنها بسیار غیر حساس است. شکل موج صوتی در دامنه‌ی زمان فقط به صورت غیر مستقیم با شنوایی رابطه دارد و معمولاً در سیستمهای صوتی در نظر گرفته نمی‌شود.

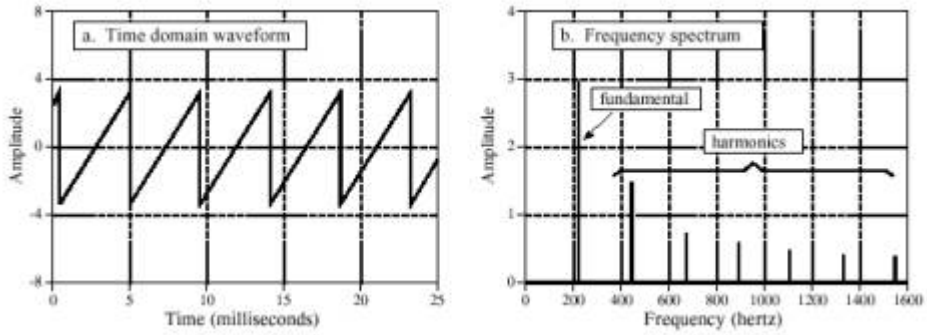
عدم حساسیت گوش به فاز صدا با توجه به روش پخش شدن آن در محیط قابل درک است. فرض کنید که شما در یک اتاق به صحبتهای فردی گوش می‌دهید. بیشتر صداهایی که گوش شما دریافت می‌کند حاصل بازتاب صدای اصلی از دیوارها، سقف و کف اتاق است. از آنجا که انتشار صدا بستگی به فرکانس آن دارد و میرایی ،بازتاب و مقاومت در برابر صدا بر روی آن تأثیرگذار است فرکانسهای متفاوتی از مسیرهای متفاوت به گوش می‌رسد. این به این معنی است که وقتی شما جای خود را در اتاق عوض می‌کنید فاز هر یک از فرکانسها تغییر می‌کند. چون گوش این تغییر فازها را نادیده می‌انگارد با وجود تغییر مکان شما تغییری در صدای شخص صحبت کننده احساس نمی‌کنید. از دیدگاه فیزیکی فاز یک سیگنال صدا در هنگام پخش در یک محیط پیچیده به صورت تصادفی تغییر می‌کند. از طرف دیگر گوش به فاز صدا غیر حساس است زیرا این جزء دارای اطلاعات قابل استفاده‌ی بسیار کمی می‌باشد.



شکل شماره ۳ – تشخیص فاز توسط گوش انسان. گوش انسان نسبت به فاز نسبی سینوسیهای مرکب بسیار غیر حساس است. برای نمونه این دو موج یکسان به نظر خواهند رسید، زیر دامنه‌ی اجزاء آنها یکسان است اگر چه فاز نسبی آنها متفاوت است.

در حالت کلی نمی‌توان گفت که گوش نسبت به فاز کاملاً ناشنواست. چرا که تغییر فاز می‌تواند باعث تغییر آرایش زمانی یک سیگنال صوتی شود. اما چنین امری یک پدیده‌ی نادر است که در محیطهای شنیداری طبیعی اتفاق نمی‌افتد.

فرض کنید از یک نوازنده‌ی ویولون خواسته‌ایم نتی را بنوازد. وقتی که موج صوتی ایجاد شده بر روی اسیلوسکوپ نشان داده شود یک موج دندانه‌اره‌ای مانند شکل شماره ۴ (a) مشاهده می‌شود. شکل شماره ۴ (b) نشان می‌دهد که این صوت چگونه توسط گوش دریافت می‌شود. گوش یک فرکانس اساسی (در مثال شکل ۲۲۰ هرتز) را و همسازهایی را در ۴۴۰، ۶۶۰، ۸۸۰ و... هرتز دریافت می‌کند. اگر این نت بر روی ابزار دیگری نواخته شود گوش هنوز هم همان ۲۲۰ هرتز (همان فرکانس اساسی) را دریافت می‌کند. و از این لحاظ دو صوت مشابهند که گفته می‌شود این دو صوت پیچ یکسانی دارند ولی چون دامنه‌ی همسازها متفاوت است دو صوت یکسان نیستند و گفته می‌شود که طنین دو صوت متفاوت است.



شکل شماره ۴ – موج صوتی ویولن. ویولن موج دندانه‌اره‌ای ایجاد می‌کند (شکل a)، صدای دریافت شده شامل فرکانس اساسی و همسازهای آن است (شکل b)

اغلب گفته می‌شود که طنین صدا از روی شکل موج صوتی تعیین می‌گردد. این مسأله درست است ولی کمی گمراه کننده است. احساس طنین صدا از روی میزان هارمونیکهای تشخیص داده شده توسط گوش تعیین می‌گردد. در حالی که هارمونیکها از روی شکل موج صوتی تعیین می‌گردد عدم حساسیت گوش به فاز رابطه را بسیار یک طرفه می‌کند. به همین دلیل هر موج صوتی فقط یک طنین دارد حال آن که یک زنگ خاص متعلق به تعداد بی‌نهایتی از موجهای صوتی است.

گوش بیشتر برای شنیدن هارمونیکهای اساسی تنظیم شده است. اگر یک شنونده به صدایی که حاصل ترکیب دو موج صوتی سینوسی ۱ کیلوهرتز و ۳ کیلوهرتز است گوش دهد آن را مطلوب و طبیعی توصیف خواهد کرد حال آن که اگر از موجهای ۱ کیلوهرتزی و ۳,۱ کیلوهرتزی استفاده شود برای شنونده شکایت برانگیز خواهد بود. این مسأله اساسی برای اندازه‌ها و اختلافهای استاندارد ابزارهای موسیقیایی فراهم می‌آورد.

۳- روشهای دیجیتالی ذخیره‌ی صدا

در طراحی یک سیستم صوتی دیجیتال دو پرسش وجود دارند که باید پاسخ داده شوند: ۱- چقدر لازم است صوت خوب به نظر برسد؟ ۲- چه نرخ داده‌ای قابل تحمل است؟ جواب به این پرسشها غالباً به یکی از این سه انتخاب منجر می‌شود: اول موسیقی با وفاداری بالا[۱۴] که در آن کیفیت صدا مهم‌ترین چیز است و تقریباً هر نرخ داده‌ای قابل قبول است. دوم ارتباط تلفنی[۱۵] که نیازمند طبیعی به نظر رسیدن صحبت و یک نرخ داده‌ی پایین برای کاهش هزینه‌ی سیستم است. سوم صحبت فشرده شده[۱۶] که در آن کاهش نرخ داده بسیار مهم است و مقداری غیر طبیعی به نظر رسیدن کیفیت صدا قابل تحمل است. این مورد در بر دارنده‌ی ارتباطات نظامی، تلفنهای سلولی و صحبت ذخیره شده به صورت دیجیتال برای پست الکترونیکی صوتی یا کاربردهای چند رسانه‌ای است.

شکل شماره ۵ بده بستانهای موجود در انتخاب هر یک از این سه روش را نشان می‌دهد.

در حالی که موسیقی نیازمند پهنای باند ۲۰ کیلوهرتز است صحبتی که طبیعی به نظر برسد فقط به پهنای باندی در حدود ۳٫۲ کیلوهرتز نیازمند است. در این حال هر چند پهنای باند به اندازه‌ی ٪۱۶ مقدار اولیه محدود می‌شود ولی فقط ٪۲۰ اطلاعات اولیه از دست می‌رود.

سیستمهای ارتباط راه‌دور اغلب از نرخ نمونه‌برداری در حدود ۸ کیلوهرتز استفاده می‌کنند که اجازه‌ی انتقال صحبت را با کیفیتی در حد طبیعی می‌دهد ولی اگر از آن برای انتقال موسیقی استفاده شود تا میزان بالایی از کیفیت آن از دست می‌رود. شما احتمالاً با تفاوت این دو میزان آشنایی دارید: ایستگاههای رادیویی اف.ام با پهنای باندی در حدود ۲۰ کیلوهرتز اقدام به پخش می‌کنند حال آن که ایستگاههای ای.ام محدود به ۳٫۲ کیلوهرتز هستند. صحبت و صداهای معمول روی ایستگاههای نوع دوم طبیعی به نظر می‌رسد حال آن که موسیقی این گونه نیست.

Sound Quality Required	Bandwidth	Sampling rate	Number of bits	Data rate (bits/sec)	Comments
High fidelity music (compact disc)	5 Hz to 20 kHz	44.1 kHz	16 bit	706k	Satisfies even the most picky audiophile. Better than human hearing.
Telephone quality speech	200 Hz to 3.2 kHz	8 kHz	12 bit	96k	Good speech quality, but very poor for music.
(with companding)	200 Hz to 3.2 kHz	8 kHz	8 bit	64k	Nonlinear ADC reduces the data rate by 50%. A very common technique.
Speech encoded by Linear Predictive Coding	200 Hz to 3.2 kHz	8 kHz	12 bit	4k	DSP speech compression technique. Very low data rates, poor voice quality.

شکل شماره ۵ - نرخ داده‌ی صوتی در برابر کیفیت صدا. کیفیت صدای یک سیگنال صوتی دیجیتال به نرخ داده‌ی آن که برابر با حاصلضرب نرخ نمونه‌برداری آن در تعداد بیتهای آن در هر نمونه بستگی دارد که به سه بخش تقسیم می‌شود: موسیقی باوفاداری بالا (۷۰۶کیلوبیت بر ثانیه)، صحبت با کیفیت تلفن (۶۴کیلوبیت بر ثانیه) وصحبت فشرده شده (۴ کیلوبیت بر ثانیه)

سیستمهایی که فقط با صدا (و نه موسیقی) سر و کار دارند می‌توانند مقدار دقت را از ۱۶ بیت به ۱۲ بیت بدون از دست رفتن دقتی قابل توجه کاهش دهند. این میزان می‌تواند با انتخاب اندازه‌ی نامتساوی برای گام مقدارگزینی[۱۷] می‌تواند به ۸ بیت در هر نمونه نیز کاهش یابد. یک نرخ نمونه‌برداری ۸ کیلوهرتز با دقت ای.دی.سی ۸ بیت در هر نمونه به نرخ داده‌ی ۶۴کیلوبیت بر ثانیه می‌انجامد. این یک حد نهایی برای طبیعی به نظر رسیدن صحبت است. دقت کنید که صحبت نیازمند نرخ داده‌ای معادل ٪۱۰ نرخ داده‌ی موسیقی با وفاداری بالاست.

نرخ داده‌ی ۶۴ کیلو بیت بر ثانیه نمایانگر کاربرد نهایی نظریه‌ی نمونه‌برداری و مقدارگزینی برای سیگنالهای صوتی است. روشهای کاهش نرخ داده به اندازه‌ای بیشتر از این مبتنی بر فشرده‌سازی جریان داده با حذف تکرارهای ذاتی سیگنال صحبت است. یکی از کاراترین روشهای موجود ال.پی.سی[۱۸] است که انواع و زیرگروههای متعدد دارد. بر اساس کیفیت سیگنال صحبت مورد نیاز این روش می‌تواند نرخ داده را تا اندازه‌ای بین ۲ تا ۶ کیلو بیت بر ثانیه کاهش دهد.

۱- ساختار مورد نیاز برای نگهداری ویژگیهای صدا

همچنان که در فصل پیش اشاره شد برای ذخیره یا بازخوانی یک نمونه صدا به صورت دیجیتال نیازمند آنیم که برخی ویژگیهای خاص صدای دیجیتالی از قبیل نرخ نمونه‌برداری، تعداد بیت هر نمونه و یک‌کاناله یا دوکاناله بودن صدا را مشخص کنیم.

برای این منظور در محیط برنامه‌نویسی مورد نظر ما (ویندوز) از ساختاری به نام WAVEFORMATEX استفاده می‌گردد که به صورت زیر تعریف می‌گردد:

```
typedef struct {
   WORD  wFormatTag;
   WORD  nChannels;
   DWORD nSamplesPerSec;
   DWORD nAvgBytesPerSec;
   WORD  nBlockAlign;
   WORD  wBitsPerSample;
   WORD  cbSize;
} WAVEFORMATEX;
```

در این ساختار فیلد wFormatTag فرمت فایل را که نشان دهنده‌ی نوع الگوریتمهای به کار گرفته شده برای فشرده‌سازی صدا و... است را مشخص می‌کند. برای استفاده‌ی مورد نظر ما فرمت خاصی که با ثابت WAVE_FORMAT_PCM مشخص می‌گردد و فرمت پی.سی.ام[۱] نامیده می‌شود مناسب است. علاوه بر آن فیلد cbSize برای فرمتهای غیر پی.سی.ام استفاده می‌شود و ما همواره مقدار آن را صفر در نظر خواهیم گرفت.

از آنجا که پردازش این ساختار در برنامه‌نویسی صدا برای پروژه‌ی مورد نظر بارها صورت می‌گیرد و از آنجا که یک شیوه‌ی طراحی شیءگرا (شیوه‌ی ام.اف.سی[۲])برای پیاده‌سازی پروژه در نظر گرفته شده بود و از آنجا که پردازش این ساختار نیاز به برخی محاسبات تکراری (تعیین nBlockAlign و nAvgBytesPerSec) دارد و به چند دلیل دیگر تصمیم گرفته شد که این ساختار و پردازش آن به صورت یک کلاس با نام HSound پیاده سازی گردد که ضمن خودکار نمودن پردازش این ساختار کلاسهایی که به اعمال پخش و ضبط را بر عهده دارند از این کلاس ارثبری نموده برنامه نویسی را آسانتر و کد به دست آمده را خواناتر نمایند.

تعریف این کلاس به صورت زیر است:

```
class HSound
{
public:
        //constructor and destructor:
        HSound();
        virtual ~HSound();
        //setting wave data:
        void SetBitsPerSample(int bps);
        void SetSamplesPerSecond(int sps);
        void SetNumberOfChannels(int nchan);
       //retrieving wave data:
       WAVEFORMATEX* GetFormat();
       int GetSamplesPerSecond();
       int GetBitsPerSample();
       int GetNumberOfChannels();
protected:
    WAVEFORMATEX m_wfData;
private:
    void Update();
};
```

قبل از هر چیز باید به این نکته اشاره شود که این کلاس برای پردازش فرمت پی.سی.ام در نظر گرفته شده، لذا ضمن تعریف مقادیر پیش‌فرض لازم برای این فرمت و استفاده از روشهای خاص این فرمت برای محاسبه‌ی فیلدهای مختلف امکان تغییر فرمت و استفاده از سایر فرمتها را به برنامه‌نویس نمی‌دهد و به منظور پردازش سایر فرمتها ساختار کلاس می‌بایست تغییر کند.

فیلد wBitsPerSample تعداد بیتِ هر نمونه را مشخص میکند که برای فرمت پی.سی.ام فقط می‌تواند یکی از دو مقدار ۸ و ۱۶ را داشته باشد و برای سایر فرمتها مقادیر ممکن بستگی به مشخصات منتشر شده توسط شرکتهای به وجود آورنده و پشتیبانی‌کننده‌ی آنها دارد.

متدی که در پی می‌آیند آن را مقدارگذاری می‌کند (در مورد متد Update و علت فراخوانی آن در ادامه توضیح داده خواهد شد) :

```
void HSound::SetBitsPerSample(int bps)
{
    m_wfData.wBitsPerSample = bps;
    Update();
}
```

و متد زیر مقدار انتخاب شده را برمی‌گرداند:

```
int HSound::GetBitsPerSample()
{
    return m_wfData.wBitsPerSample;
}
```

فیلد nSamplesPerSec تعداد نمونه‌ها در هر ثانیه (نرخ نمونه‌برداری) را مشخص می‌کند. برای فرمت پی.سی.ام مقادیر معمول ۸کیلوهرتز (۸۰۰۰)، ۱۱,۰۲۵کیلوهرتز (۱۱۰۲۵)، ۲۲,۰۵کیلوهرتز (۲۲۰۵۰) و ۴۴,۱کیلوهرتز (۴۴۱۰۰) می‌باشد و برای سایر فرمتها مقادیر ممکن بستگی به مشخصات منتشر شده توسط شرکتهای به وجود آورنده و پشتیبانی‌کننده‌ی آنها دارد.

متد مقدارگذاری این فیلد:

```
void HSound::SetSamplesPerSecond(int sps)
{
    m_wfData.nSamplesPerSec = sps;
    Update();
}
```

و متد دریافت مقدار آن:

```
int HSound::GetSamplesPerSecond()
{
    return m_wfData.nSamplesPerSec;
}
```

فیلد nChannels تعداد کانالهای موج صوتی را مشخص می‌کنند. صداهای تک کانال (مقدار فیلد برابر با ۱) مونو و صداهای دوکاناله (مقدار فیلد برابر با ۲) استریو خواهند بود.

متد مقدارگذاری این فیلد:

```
void HSound::SetNumberOfChannels(int nchan)
{
    m_wfData.nChannels = nchan;
    Update();
}
```

و متد دریافت مقدار آن:

```
int HSound::GetNumberOfChannels()
{
    return m_wfData.nChannels;
}
```

هر چند تعداد کانالهای صدا در این کلاس قابل تغییراست اما در کلاسهای مشتق شده همواره الگوریتمها برای موج صوتی تک‌کاناله نوشته شده‌اند و استفاده از آنها برای پردازش موج صوتی دو کاناله نیازمند دستکاری کد این کلاسهاست که به لحاظ استفاده‌ای که ما از این کلاسها نموده‌ایم یک کار اضافی و غیرضروری به نظر می‌رسد.

فیلد nBlockAlign کمینه‌ی تعداد واحد داده را برای فرمت انتخاب شده تعیید می‌کند که اگر فرمت انتخاب شده پی.سی.ام باشد برابر با حاصل ضرب تعداد کانالها (nChannels) در تعداد بیتِ هر نمونه (nBitsPerSample) تقسیم بر تعداد بیتهای موجود در هر بایت (۸) خواهد بود و برای سایر فرمتها بستگی به مشخصات منتشر شده توسط شرکتهای به وجود آورنده و پشتیبانی‌کننده‌ی آنها دارد. فیلد nAvgBytesPerSec نیز تعداد متوسط بایتهای موجود در هر ثانیه‌ی صدا را مشخص می‌کند و برای فرمت پی.سی.ام برابر با تعداد نمونه‌های موجود در هر ثانیه (nSamplesPerSec) در کمینه‌ی تعداد واحد داده (nBlockAlign) خواهد بود و برای سایر فرمتها بستگی به مشخصات منتشر شده توسط شرکتهای به وجود آورنده و پشتیبانی‌کننده‌ی آنها دارد.

متد Update که در کد مقدارگذاری سایر فیلدها محاسبات توضیح داده شده‌ی بالا را انجام می‌دهد:

```
void HSound::Update()
{
  m_wfData.nBlockAlign = m_wfData.nChannels*(m_wfData.wBitsPerSample/8);
  m_wfData.nAvgBytesPerSec =
m_wfData.nSamplesPerSec*m_wfData.nBlockAlign;
}
```

در صورتی که نیاز باشد با ساختار اصلی WAVEFORMATEX کار شود متد زیر مقدار عضوی از کلاس را که از این نوع است باز می‌گرداند:

```
WAVEFORMATEX* HSound::GetFormat()
{
    return &m_wfData;
}
```

در متد سازنده‌ی این کلاس به طور پیش‌فرض برای نمونه‌ی صوتی مورد نظر نرخ نمونه‌برداری ۴۴٫۱کیلوهرتز با ۱۶ بیت در هر نمونه در نظر گرفته شده و فرض بر آن است که نمونه‌ی صوتی یک کاناله است:

```
HSound::HSound()
{
    m_wfData.wFormatTag     = WAVE_FORMAT_PCM;
    m_wfData.cbSize         = 0;
    SetBitsPerSample(16);
    SetSamplesPerSecond(44100);
    SetNumberOfChannels(1);
}
```

همچنانکه از روی تعریف کلاس قابل فهم است این کلاس در واقع تمامی اعمال را روی عضو داده‌ی محافظت شده‌ی m_wfData اِعمال می‌نماید و با غیر مستقیم نمودن دسترسی به این عضو داده برای برنامه‌ی استفاده کننده ضمن رعایت اصل پنهانسازی اطلاعات به فراخوانی رویه‌ی Update در متدهای تغییر دهنده‌ی اعضای مرتبط با nBlockAlign و nAvgBytesPerSec تغییرات لازم را به آنها اعمال می‌کند.

۲- انجام پردازش صدا به صورت یک رشته‌ی[۳] مستقل

می‌توان با استفاده از توابع کار با صدای ویندوز به گونه‌ای برنامه‌نویسی نمود که نیازی به ایجاد رشته‌های مستقل برای پردازنده‌های صدا نباشد، اما وجود دلایلی از قبیل عدم انعطاف‌پذیری این روش و تک‌وظیفه‌ای شدن برنامه در حین انجام عملیات پردازش صدا باعث می‌شود که روش استفاده از رشته‌های مستقل مورد توجه ما قرار گیرد.

از آغاز در نظر داشتیم که رابط برنامه به گونه طراحی شود که کاربر در هنگام کار با برنامه و انجام عملیاتی نظیر ضبط صدا از عملکرد برنامه مطمئن باشد. به این معنی که مثلاً در حین هنگام صدا با استفاده از یک رابط گرافیکی مانند یک نمایشگر اسیلوسکوپی از این که برنامه واقعاً و به درستی در حال ضبط صدای اوست و یا به لحاظ فاصله‌ی نامتناسب با میکروفن یا عدم اتصال درست آن به کارت صوتی یا خرابی آن بیشتر آنچه ضبط می‌شود سکوت و یا نویز است مطلع گردد. یک روش مناسب برای ایجاد چنین رابطی استفاده از پیام فرستاده شده برای پردازش صدا توسط یک رشته برای فعال شدن یک تابع رسم‌کننده‌ی نمودار اسیلوسکوپی است که نیاز به آن دارد که بدون قطع شدن جریان ضبط پردازش دیگری صورت گیرد. به این منظور و با استفاده از کد اولیه‌ای که در منبع شماره‌ی ۲ به آن اشاره شده کلاسی به نام HSoundRunner را از کلاس HSound اعضای داده و متدهای مرتبط با پردازش صوت و از کلاس ام.اف.سی CwinThread اعضای داده و متدهای لازم برای یک رشته را ارثبری می‌کند به صورت زیر تعریف نمودیم:

```
class HSoundRunner:
	public CWinThread,
	public HSound
{
public:
	DECLARE_DYNCREATE(HSoundRunner)
	HSoundRunner();
	~HSoundRunner();
	void	SetBufferSize(int nSamples);
	int	GetBufferSize();
	//this metheods should be overriden:
	void AddBuffer();
	BOOL Start(WAVEFORMATEX* pwfex=NULL);
	BOOL Stop();
	//for graphical display:
	void SetOwner(CWnd* pWnd);
	void ClearOwner(COLORREF crBkColor=0×000000);
public:
	//{{AFX_VIRTUAL(HSoundRecorder)
public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL
protected:
	DWORD m_dwThreadID;
	int m_iBufferSize;  // number of samples per each period
	int m_nBuffers;		//number of buffers remained to be run
	int m_nSamples;		//number of samples stored
	short* m_pSamples;	//samples stored
	BOOL m_bRunning;	//indicated running or not
//if graphical display is intended set this value
	CWnd* m_pOwner;
	void DrawBuffer(int nSamples, short* pSamples, COLORREF
crBkColor=0×000000, COLORREF crLineColor=0×00FF00);
};
```

اعضای داده‌ی این کلاس در روند انجام عملیات توسط کلاسهای مشتق شده از آنها نقش خود را نشان خواهند داد و به عنوان نمونه عضو داده‌ی m_iBufferSize که نشان دهنده‌ی آن است که بعد از ضبط یا پخش چند نمونه تابع پزدازنده‌ی پیام در کلاس پنجره‌ی کنترل کننده باید فراخوانی شود در این هیچکدام از متدهای این کلاس نقش عملی پیدا نمی‌کند و فقط مقدارگذاری آن از طریق متد SetBufferSize و دیافت مقدار فعلی آن از طریق GetBufferSize صورت می‌گیرد:

```
void HSoundRunner::SetBufferSize(int nSamples)
{
	m_iBufferSize=nSamples;
}
int HSoundRunner::GetBufferSize()
{
	return m_iBufferSize;
}
```

عضو داده‌ی m_dwThreadID مقدار شناسه‌ی رشته‌ی ایجاد شده را که در کلاس سازنده با فراخوانی CreateThead ایجاد می‌شود در بر می‌گیرد که در کلاسهای مشتق شده برای کار با فراخوانیهای ای.پی.آی پردازش صدا کاربرد پیدا می‌کند. مقدار این عضو داده در متد بازنویسی[۴] شده‌ی InitInstance و به صورت زیر تعیین می‌گردد:

```
BOOL HSoundRunner::InitInstance()
{
    m_dwThreadID = ::GetCurrentThreadId();
    return TRUE;
}
```

اعضای داده‌ی m_nSamples و m_pSamples به ترتیب تعداد نمونه‌های ضبط شده یا آماده برای پخش و آرایه‌ی حاوی آنها -که به لحاظ آسان‌تر شدن کار با کتابخانه‌ای که برای پردازش سیگنال صحبت استفاده شده و همسان با آن از نوع short در نظر گرفته شده- را نشان می‌دهند. این دو عضو داده به صورت پویا پس از انجام عملیات ضبط مقدارگذاری می‌شوند و برای عملیات پخش باید قبلاً مقدارگذاری شده باشند.

آنچه این کلاس انجام می‌دهد غیر از ایجاد یک رشته برای انجام پردازش صدا فراهم آوردن روشی برای نمایش اسیلوسکوپی صداست که از طریق متد DrawBuffer انجام می‌شود.

این متد در توابع پیامهای مربوط به پردازش صوت خود به خود فراخوانی می‌گردد و در صورتی که مقدار m_pOwner اشاره‌گر به یک پنجره یا کنترل انتخاب شود (توسط متد SetOwner) آرایه‌ی ورودی که معمولاً یک تکه‌ی تازه ضبط یا پخش شده از کل صداست متناسب با طول و عرض پنجره‌ی مورد نظر بر روی آن کشیده می‌شود. این کار با ایجاد یک ابزار متن[۵] و یک بیتمپ متناسب با ابزار متن پنجره‌ی مورد نظر، کشیدن طرح لازم با استفاده از این دو و در نهایت نمایش تصویر ایجاد شده بر روی پنجره‌ی مقصد و مطابق با کد زیر انجام می‌شود:

```
void HSoundRunner::DrawBuffer(int nSamples, short* pSamples, COLORREF
crBkColor, COLORREF crLineColor)
{
    if(m_pOwner==NULL)
        return;
    CRect rc;
    m_pOwner->GetClientRect(&rc);
    int iWidth=rc.Width();
    int iHeight=rc.Height();
    CDC* pDC=m_pOwner->GetDC();
    CBitmap Bitmap;
    Bitmap.CreateCompatibleBitmap(pDC, iWidth, iHeight);
    CDC dc;
    dc.CreateCompatibleDC(pDC);
    dc.SelectObject(&Bitmap);
    CBrush Brush(crBkColor);
    dc.FillRect(&rc,&Brush);
    CPen Pen(PS_SOLID,1,crLineColor);
    dc.SelectObject(&Pen);
    dc.SetBkColor(crBkColor);
    if(GetBitsPerSample()==16)
    {
        float fx=iWidth/float(nSamples);
        float fy=float(iHeight/32767.0);
```

```
            dc.MoveTo(0, iHeight/2);
            int i=0;
            for(float f=0; f<iWidth&&i<nSamples; f+=fx, i++)
                  dc.LineTo(int(f), int(iHeight/2+fy*pSamples[i]));
            pDC->BitBlt(0, 0, iWidth, iHeight, &dc, 0, 0, SRCCOPY);
      }
}
```

متد ClearBuffer یک روش قابل دسترسی توسط برنامه برای پاک کردن پنجره‌ی مورد استفاده به وجود می‌آورد و شامل یک فراخوانی متد محافظت شده‌ی DrawBuffer با یک آرایه‌ی به طول صفر است:

```
void HSoundRunner::ClearOwner(COLORREF crBkColor)
{
      DrawBuffer(0,NULL,crBkColor);
}
```

عضو داده‌ی m_nBuffers تعداد بافرهای اختصاص داده شده و استفاده نشده را نشان می‌دهد که در متد AddBuffer این کلاس و بازنویسی شده‌ی آن برای کلاسهاس مشتق شده مقدار آن به ازای هر بار فراخوانی یک واحد افزوده می‌شود و در پیامهای پردازش صدا که از طرف سیستم عامل فعال می‌شوند و نشانگر استفاده شدن بافر مورد نظر است (در کلاسهای مشتق شده) یک واحد کاهش می‌یابد. در نهایت صفر نبودن این عضو داده نشانگر استفاده‌ی ناکامل از بافرهای اختصاص داده شده (معادل با کامل انجام نشدن فرایند ضبط یا پخش) است که می‌تواند پردازش مناسب برای آن صورت گیرد:

```
void HSoundRunner::AddBuffer()
{
      m_nBuffers++;
}
```

عضو داده‌ی m_bRunning به منظور تشخیص این که برنامه در حال اجرای عملیات پردازش صداست و یا نه به منظور جلوگیری از ایجاد اشکال با فراخوانیهای تکراری در نظر گرفته شده که در هنگام آغاز عملیات مقدار آن برابر با TRUE و در پایان آن برابر با FALSE انتخاب می‌گردد. همچنان که در ادامه توضیح داده خواهد شد کلاسهای مشتق شده متدهایی از لحاظ نام متناسب با عملی که برای آن در نظر گرفته شده‌اند (ضبط یا پخش) برای بازگرداندن این مقدار به برنامه‌نویس کاربر این کلاسها دارند:

```
BOOL HSoundRunner::Start(WAVEFORMATEX* pwfex)
{
      if(m_bRunning)
            return FALSE;
      if(pwfex != NULL)
            m_wfData = *pwfex;
      return TRUE;
}
BOOL HSoundRunner::Stop()
{
      if(m_bRunning)
      {
            m_bRunning=FALSE;
            Sleep(500);
            return TRUE;
      }
```

```
    return FALSE;
}
```

فراخوانی استاندارد Sleep در متد Stop برای مصرف کامل بافر ایجاد شده انجام می‌گردد. در ضمن متد Start روشی برای جایگزینی مقدار پیش‌فرض m_wfData (عضو کلاس Hsound) با مقدار جدید در اختیار می‌گذارد.

در متد سازنده اعضای داده با مقادیر پیش‌فرض مقدارگذاری شده و رشته‌ی مورد نظر با فراخوانی CreateThead ایجاد می‌گردد:

```
HSoundRunner::HSoundRunner()
{
    m_iBufferSize= 2048;
    m_nBuffers = 0;
    m_bRunning = FALSE;
    m_nSamples=0;
    m_pSamples=NULL;
    m_pOwner=NULL;
    CreateThread();
}
```

در متد ویرانگر[۶] نیز در صورتی که شیء از نوع این کلاس در حال انجام عملیات پردازش صوت باشد متوقف خواهد شد:

```
HSoundRunner::~HSoundRunner()
{
    if(m_bRunning)
            Stop();
}
```

به لحاظ آن که آرایه‌ی داده‌ها (m_pSamples) در این کلاس ایجاد نمی‌گردد در متد ویرانگر آزاد شدن آن پیشبینی نشده است.

۳ـ ضبط صدا

برای ضبط صدا و انجام پردازشهای مرتبط با آن کلاسی به نام HSoundRecorder به صورت زیر از کلاس HSoundRunner مشتق گردید:

```
class HSoundRecorder :  public HSoundRunner
{
    DECLARE_DYNCREATE(HSoundRecorder)
public:
    HSoundRecorder();
    virtual ~HSoundRecorder();
protected:
    void AddBuffer();
    //Message Map For WM_WIM_DATA:
    afx_msg void OnDataReady(UINT uParm, LONG lWaveHdr);
private:
    HWAVEIN m_hWaveIn;
```

```
    HShortQueue* m_pQueue;
public:
    BOOL Start(WAVEFORMATEX* pwfex=NULL);
    BOOL Stop();
    short* GetSamples(int& nSamples);
    BOOL IsRecording();
    DECLARE_MESSAGE_MAP()
};
```

برای استفاده از این کلاس کافی است متدهای Start و Stop آن فراخوانی گردند ولی درک کامل نحوه‌ی عملکرد آن نیاز به برخی مقدمات دارد.

برای شروع کار ضبط از فراخوانی ای.پی.آی زیر با پارامترهای مناسب برای در اختیار گرفتن یک ابزار ورودی صدا استفاده می‌کنیم:

```
MMRESULT waveInOpen(LPHWAVEIN phwi, UINT uDeviceID,
LPWAVEFORMATEX pwfx, DWORD dwCallback, DWORD dwCallbackInstance,
DWORD fdwOpen);
```

که در آن phwi اشاره‌گر به بافری است که یک handle به ابزار باز شده برای ورودی صدا را در اختیار می‌گذارد. این پارامتر ابزاری را برای دسترسی به وسیله‌ی ورودی صدا در اختیار می‌گذارد که ما در سایر فراخوانیهای مرتبط به آن نیاز داریم لذا در کلاس تعریف شده متغیر m_hWaveIn را برای ذخیره‌ی این پارامتر پس از این فراخوانی و دسترسی به آن در سایر متدها در نظر گرفته‌ایم.

پارامتر uDeviceID شناسه‌ی ابزار ورودی را به تابع می‌دهد. می‌توان از شناسه‌ی WAVE_MAPPER استفاده کرد که با استفاده از آن برنامه سخت‌افزار پیش‌فرض موجود را که دارای قابلیت پردازش فرمت انتخاب شده که توسط پارمتر pwfx به تابع داده می‌شود و ما از عضو داده‌ی m_wfData از کلاس HSound برای انتخاب مقدار آن استفاده می‌کنیم به طور خودکار انتخاب می‌کند.

پارامتر dwCallback شناسه‌ی پنجره، پردازه یا رشته‌ای را که پیامهای چندرسانه‌ای به آن ارسال خواهد شد به تابع می‌دهد که ما از شناسه‌ی رشته (عضو داده‌ی m_dwThreadID) برای تعیین این پارامتر استفاده خواهیم نمود. پارامتر بعدی dwCallbackInstance داده‌ی سطح کاربری را که به ساز و کار فرتخوانی callback ارسال می‌شود تعیین می‌نماید و ما از این پارامتر استفاده نخواهیم نمود.

پارامتر آخر یعنی fdwOpen پرچمی برای ابزار ورودی است که سازوکار تفسیر پارامترها را مشخص می‌کند و چون ما از سازوکار فراخوانی رشته‌ای استفاده می‌کنیم مقدار آن را برابر با CALLBACK_THREAD انتخاب می‌نماییم.

مقدار بازگشتی در صورت بروز اشکال غیرصفر خواهد بود مسائلی از قبیل اختصاص یافتن وسیله‌ی ورودی به یک پردازه‌ی دیگر به صورتی که سیستم عامل به صورت اشتراکی آن را در اختیار نگذارد، کمبود حافظه و ... ممکن است باعث بروز اشکال شوند که نوع اشکال با توجه به مقدار بازگشتی مشخص می‌گردد و میتواند به کاربر اعلام گردد.

بعد از در اختیار گرفتن یک وسیله‌ی ورودی لازم است که برای عملیات حافظه اختصاص یابد و این عمل می‌تواند با فراخوانی AddBuffer به تعداد کافی صورت گیرد.

بعد از تخصیص حافظه توسط فراخوانی ای.پی.آی زیر عمل ضبط شروع می‌گردد:

```
MMRESULT waveInStart(HWAVEIN hwi);
```

پارامتر وروی همان پارامتر بازگشت با مقدار فراخوانی waveInOpen یعنی phwi است که همچنانکه اشاره شد ما آن را در عضو داده‌ی m_hWaveIn نگهداری می‌کنیم. این فراخوانی نیز مانند قبلی در صورت موفقیت‌آمیز بودن مقدار صفر باز می‌گرداند.

توضیحات بالا مقدمات کافی را برای درک کد متد Start که در زیر می‌آید فراهم می‌آورد:

```
BOOL HSoundRecorder::Start(WAVEFORMATEX* pwfex)
{
    if(!HSoundRunner::Start(pwfex))
     return FALSE;
    m_pQueue=new HShortQueue;
    //Open the wave device:
    if(::waveInOpen(&m_hWaveIn, WAVE_MAPPER, &m_wfData,
m_dwThreadID, 0L, CALLBACK_THREAD))
     return FALSE;
    //Add several buffers to queue:
    for(int i=0;i<3;i++)
     AddBuffer();
    if(::waveInStart(m_hWaveIn))
     return FALSE;
    m_bRunning=TRUE;
    return TRUE;
}
```

اما توابع چندرسانه‌ای ویندوز سازوکار خاصی برای اضافه کردن بافر دارند که می‌بایست آنها را در نسخه‌ی بازنویسی شده‌ی AddBuffer این کلاس لحاظ کنیم. برای اضافه کردن یک بافر ابتدا باید آن را توسط فراخوانی زیر برای استفاده آماده کنیم:

```
MMRESULT waveInPrepareHeader(HWAVEIN hwi, LPWAVEHDR pwh, UINT
cbwh);
```

به جای پارامتر hwi عضو داده‌ی m_hWaveIn را که قبلاً در فراخوانی waveInOpen مقدارگذاری شده قرار می‌دهیم. پارامتر دوم یک اشاره‌گر به متغیری با ساختار زیر است:

```
typedef struct {
   LPSTR  lpData;
   DWORD  dwBufferLength;
   DWORD  dwBytesRecorded;
   DWORD  dwUser;
   DWORD  dwFlags;
   DWORD  dwLoops;
   struct wavehdr_tag * lpNext;
   DWORD  reserved;
} WAVEHDR;
```

که لازم است اشاره‌گر lpData به یک حافظه حاوی تعداد مورد نیاز عضو اشاره کند. از آنجا که ما هر بار بافری با اندازه‌ی m_iBufferSize در نظر می‌گیریم، تعداد خانه‌های این آرایه بر حسب بایت برابر با اندازه‌ی بافر ضرب در حداقل تعداد بلوک برای فرمت انتخاب شده (فیلد nBlockAlign ساختار WAVEFORMATEX) می‌باشد و لازم است که به این تعداد حافظه اختصاص داده اشاره‌گر lpData را برابر با آدرس آن انتخاب کنیم، در ضمن اندازه‌ی بافر اختصاص داده شده را از طریق فیلد

dwBufferLength به اطلاع تابع استفاده کننده می‌رسانیم. پارامتر آخر فراخوانی مورد بحث باید برابر با اندازه‌ی پارامتر دوم بر حسب بایت قرار داده شود.

بعد از آماده شدن بافر آن را توسط فراخوانی زیر به بافرهای آماده برای اعمال چندرسانه‌ای اضافه می‌کنیم:

```
MMRESULT waveInAddBuffer(HWAVEIN hwi, LPWAVEHDR pwh, UINT cbwh);
```

مانند فراخوانیهای قبل مقدار خروجی دو فراخوانی اخیر در صورت عدم بروز خطا صفر خواهد بود. کد زیر پیاده‌سازی کامل متد بازنویسی شده‌ی AddBuffer را برای کلاس HSoundRecorder به نمایش می‌گذارد:

```
void HSoundRecorder::AddBuffer()
{
    //new a buffer:
    char      *sBuf=new char[m_wfData.nBlockAlign*m_iBufferSize];
    //new a header:
    LPWAVEHDR       pHdr=new WAVEHDR;
    if(!pHdr) return;
    ZeroMemory(pHdr,sizeof(WAVEHDR));
    pHdr->lpData=sBuf;
    pHdr->dwBufferLength=m_wfData.nBlockAlign*m_iBufferSize;
    //prepare it:
    ::waveInPrepareHeader(m_hWaveIn, pHdr, sizeof(WAVEHDR));
    //add it:
    ::waveInAddBuffer(m_hWaveIn, pHdr, sizeof(WAVEHDR));
    HSoundRunner::AddBuffer();
}
```

بعد از آن که عمل ضبط آغاز شد و پس از پر شدن هر بافر پیغامی به پنجره یا رشته‌ای که شناسه‌ی آن به فراخوانی waveInOpen داده شده ارسال می‌گردد که با شناسه‌ی MM_WIM_DATA می‌توان به آن مراجعه نمود. در هنگام فعال شدن این پیغام لازم است بافر استفاده شده برای استفاده‌ی مجدد آماده گردد و در ضمن مکان مناسب برای ذخیره‌ی داده‌های ضبط شده و همچنین نمایش آنها پس از فعال شدن این پیغام است.

از آنجا که طول زمان ضبط صدا مشخص نیست طول بافری که نهایتاً داده‌ها باید در آن قرار گیرند قابل پیشبینی نمی‌باشد. از این رو ما از یک ساختار که نوع آن را HShortPocket نامگذاری کرده‌ایم برای ذخیره‌ی داده‌های ارسال شده استفاده می‌نماییم و حاصل را در صفی که در قالب کلاس HShortQueue پیاده‌سازی شده درج می‌نماییم.(این دو ساختار ربطی به برنامه‌نویسی پردازش صدا ندارند و درک عملکرد آنها نیاز به توضیح اضافی ندارد لذا در اینجا توضیح داده نمی‌شوند.) عضو داده‌ی m_pQueue از کلاس HSoundRecorder صفی است که در سطور قبل در مورد آن بحث شد.

ما به شیوه‌ی ام.اف.سی برای پیغام MM_WIM_DATA تابعی به نام OnDataReady می‌سازیم که پارامترهای آن پارامترهای ارسالی از طرف پیغام هستند که دومین آنها که حاوی ساختار بافر استفاده شده است برای ما اهمیت دارد. با توجه به توضیحات داده شده درک کد این تابع میسر است:

```
void HSoundRecorder::OnDataReady(UINT uParm, LONG lWaveHdr)
{
    LPWAVEHDR       pHdr=(LPWAVEHDR)lWaveHdr;
    ::waveInUnprepareHeader(m_hWaveIn, pHdr, sizeof(WAVEHDR));
    if(m_bRunning)
    {
    //Save Input Data:
```

```
        m_pQueue->InsertItem(pHdr->dwBytesRecorded/2, (short*)pHdr->lpData);
        //Draw Buffer:
        DrawBuffer(pHdr->dwBytesRecorded/2, (short*)pHdr->lpData);
        //reuse the header:
        ::waveInPrepareHeader(m_hWaveIn, pHdr, sizeof(WAVEHDR));
        ::waveInAddBuffer(m_hWaveIn, pHdr, sizeof(WAVEHDR));
        return;
        }
        //we are stopping:
        delete pHdr->lpData;
        delete pHdr;
        m_nBuffers–;
}
```

در صورتی که متد Stop احضار شده باشد مقدار m_bRunning برابر با FALSE است در این هنگام نه تنها نیازی به اضافه کردن بافر نداریم بلکه می‌توانیم بافرهای اختصاص داده شده را آزاد کنیم. قسمت آخر کد چنین عملی را انجام می‌دهد.

پس از انجام عمل و احضار متدStop توسط برنامه لازم است با فراخوانی waveInStop عمل ضبط را متوقف کنیم و با فراخوانی waveInClose ابزار ضبط آن را برای استفاده‌ی سایر برنامه‌ها آزاد کنیم. علاوه بر این در این هنگام تمامی بافرها ارسال شده‌اند و می‌توانیم آن را به صورت یک آرایه‌ی معمولی ذخیره کنیم و متغیر m_pQueue را آزاد نماییم:

```
BOOL HSoundRecorder::Stop()
{
        if(!HSoundRunner::Stop())
        return FALSE;
        ::waveInStop(m_hWaveIn);
        ::waveInClose(m_hWaveIn);
        m_pSamples=m_pQueue->ConvertToArray(m_nSamples);
        delete m_pQueue;
        return TRUE;
}
```

پس از انجام این عمل برنامه می‌تواند با فراخوانی GetSamples به آرایه‌ی حاوی صدای ضبط شده دسترسی پیدا کند:

```
short* HSoundRecorder::GetSamples(int& nSamples)
{
        nSamples=m_nSamples;
        return m_pSamples;
}
```

این آرایه در هنگام آزاد شدن متغیر از نوع HSoundRecorder در متد ویرانگر آزاد می‌شود:

```
HSoundRecorder::~HSoundRecorder()
{
        if(m_pSamples)
         delete []m_pSamples;
}
```

در متد سازنده‌ی کلاس پدر مقدار m_pOwner برابر با NULL در نظر گرفته می‌شود. این به این معنی است که در حالت پیش‌فرض عملیات به صورت گرافیکی نشان داده نمی‌شود و برای انجام این عمل لازم است که ابتدا مقدار m_pOwner به اشاره‌گر به یک پنجره یا کنترل توسط فراخوانی SetOwner مقدارگذاری شود. در متد سازنده‌ی این کلاس به منظور جلوگیری از مقدارگزینیهای ناخواسته مقدار m_hWaveIn برابر با NULL انتخاب می‌گردد:

```
HSoundRecorder::HSoundRecorder()
{
    m_hWaveIn       =NULL;
}
```

۴- پخش صدا

پردازشهای مربوط به پخش صدا مشابهت زیادی با پردازشهای مربوط به ضبط دارد در اینجا نیز باید ابتدا یک ابزار صدا را برای خروجی باز کرد، تعدادی بافر اضافه نمود، در تابع پیام بافرهای جدید آماده نمود و سرانجام در متد Stop ابزار خروجی را بست:

```
class HSoundPlayer:
    public HSoundRunner
{
public:
    HSoundPlayer();
    ~HSoundPlayer();
    void SetData(int nSamples, short* pSamples);
    BOOL Start(WAVEFORMATEX* format=NULL);
    BOOL Start(int iSize, short* pData, WAVEFORMATEX* pwfex=NULL);
    BOOL Stop();
    BOOL IsPlaying();
    void AddBuffer();
    DECLARE_MESSAGE_MAP()
    afx_msg void OnMM_WOM_DONE(UINT parm1, LONG parm2);
private:
    HWAVEOUT m_hWaveOut;
    int m_nSamplesPlayed;
};
```

تفاوت تنها در این مورد است که در اینجا ما باید داده‌های آماده را به برنامه بدهیم. به دلیل آن که در برنامه‌های ما خروجی همیشه پس از ورودی مطرح می‌گردد و ما پس از پایان ورودی به داده‌های آن دسترسی داریم ساده‌ترین راه دادن داده‌ها مقداردهی اشاره‌گر m_pSamples با بافر حاوی داده‌های ورودی است که معمولاً ما آن را از شیء ضبط کننده می‌گیریم:

```
void HSoundPlayer::SetData(int iSize, short* pData)
{
    m_nSamples=iSize;
    m_pSamples=pData;
}
```

به منظور افزایش انعطاف‌پذیری می‌توان این داده‌ها را در متد Start نیز دریافت نمود:

```
BOOL HSoundPlayer::Start(int iSize, short* pData, WAVEFORMATEX* format)
{
    SetData(iSize, pData);
    return Start(format);
}
```

فراخوانی waveOutOpen عملی مشابه waveInOpen را برای خروجی صدا انجام می‌دهد و پارامترهای آن مشابه با آن است:

```
BOOL HSoundPlayer::Start(WAVEFORMATEX* format)
{
    if(m_pSamples==NULL)
     return FALSE;
    if(!HSoundRunner::Start(format))
    return FALSE;
    else
    {
    // open wavein device
    MMRESULT mmReturn = 0;
    mmReturn = ::waveOutOpen(&m_hWaveOut, WAVE_MAPPER, &m_wfData,
m_dwThreadID, NULL, CALLBACK_THREAD);
    if(mmReturn)
     return FALSE;
    else
    {
     m_bRunning = TRUE;
     m_nSamplesPlayed=0;
     for(int i=0; i<3; i++)
       AddBuffer();
    }
    }
    return TRUE;
}
```

در اینجا تفاوتی نیز وجود دارد. در فرایند ضبط این برنامه‌ی کاربر بود که پیغام Stop را ارسال می‌کرد حال آن که در اینجا علاوه بر کاربر، تمام شدن بافر حاوی داده‌ها نیز باید باعث فعال شدن آن پیغام شود. برای این منظور از متغیر شمارشگری به نام m_pSamplesPlayed استفاده کرده‌ایم که تعداد نمونه‌های پخش شده بر حسب تعداد حافظه‌ی short (که نصف تعداد نمونه در حافظه‌ی معادل بر حسب بایت است) را ذخیره می‌کند.

تفاوت دیگری نیز وجود دارد و آن این است که ما باید توسط فراخوانی waveOutWrite داده‌ها را روی بافر ارسالی بنویسیم:

```
void HSoundPlayer::AddBuffer()
{
    MMRESULT mmReturn = 0;
    // create the header
    LPWAVEHDR pHdr = new WAVEHDR;
    if(pHdr == NULL) return;
    // new a buffer
    pHdr->lpData=(char*)(m_pSamples+m_nSamplesPlayed);//buffer;
```

```
    pHdr->dwBufferLength = m_iBufferSize;
    pHdr->dwFlags = 0;
    // prepare it
    mmReturn=::waveOutPrepareHeader(m_hWaveOut, pHdr, sizeof(WAVEHDR));
    // write the buffer to output queue
    mmReturn =::waveOutWrite(m_hWaveOut, pHdr,sizeof(WAVEHDR));
    if(mmReturn) return;
    // increment the number of waiting buffers
    m_nSamplesPlayed+=m_iBufferSize/2;
    HSoundRunner::AddBuffer();
}
```

در انجام عمل پخش نیز پیغامی پس از پایان پخش هر بافر با شناسه‌ی MM_WON_DONE به پنجره یا رشته‌ی کنترل کننده ارسال می‌شود که در آن می‌توانیم داده‌ی پخش شده را به صورت گرافیکی نشان دهیم، بافر بعدی را بفرستیم و در صورت رسیدن به پایان داده‌ها پیغام Stop را ارسال کنیم:

```
void HSoundPlayer::OnMM_WOM_DONE(UINT parm1, LONG parm2)
{
    LPWAVEHDR pHdr = (LPWAVEHDR) parm2;
    //Draw Buffer:
    DrawBuffer(pHdr->dwBufferLength/2, (short*)pHdr->lpData);
    if(::waveOutUnprepareHeader(m_hWaveOut, pHdr, sizeof(WAVEHDR)))
     return;
    m_nBuffers–;
    if(m_bRunning)
    {
    if(!(m_nSamplesPlayed+m_iBufferSize/2>=m_nSamples))
    {
   AddBuffer();
   // delete old header
   delete pHdr;
   return;
    }
    else
    {
   Stop();
    }
    }
    // we are closing the waveOut handle,
    // all data must be deleted
    // this buffer was allocated in Start()
    delete pHdr;
    if(m_nBuffers == 0 && m_bRunning == false)
    {
    if (::waveOutClose(m_hWaveOut))
     return;
    }
}
```

اگر دقت کرده باشید در هنگام رسیدن به پایان داده‌ها در همین تابع اخیر ما ابزار خروجی را می‌بندیم. اگر زودتر پیغام Stop توسط برنامه ارسال شود نیز به لحاظ FALSE شدن مقدار m_bRunning این عمل انجام می‌پذیرد لذا کافی است که در متد Stop ابزار برای استفاده‌ی بعدی به طور کامل آزاد گردد:

```
BOOL HSoundPlayer::Stop()
{
    if(HSoundRunner::Stop())
     return (::waveOutReset(m_hWaveOut)!=0);
    return FALSE;
}
```

از آنجا که در این کلاس حافظه‌ای اختصاص داده نمی‌شود در متد ویرانگری نیز آزاد شدن حافظه انجام نمی‌گیرد. مشابه متد سازنده‌ی کلاس HSoundPlayer مقدار m_hWaveOut در ابتدا برابر با NULL در نظر گرفته می‌شود:

```
HSoundPlayer::HSoundPlayer()
{
    m_hWaveOut = NULL;
}
```

۵- کتابخانه‌ی پردازش صوت

از آنجا که کلاسهای پیاده‌سازی شده که در این فصل توضیح داده شدند مورد استفاده‌ی بیش از یک برنامه قرار گرفته‌اند آنها را به صورت یک کتابخانه‌ی ایستا و با نام HSoundLib گردآوری نمودیم تا تغییر کد این کتابخانه راحت‌تر در تمامی برنامه‌ها اعمال گردد و نیاز به تغییر کد تک تک آنها نباشد.

فصل پنجم - پردازش صحبت

۱- ترکیب و تشخیص صحبت

کاربردهای نیازمند پردازش صحبت اغلب در دو دسته‌ی ترکیب صحبت[۱] و تشخیص صحبت[۲] مورد بررسی قرار می‌گیرند.

ترکیب صحبت عبارت است از فن‌آوری تولید مصنوعی صحبت به وسیله‌ی ماشین و به طور عمده از پرونده‌های متنی به عنوان ورودی آن استفاده می‌گردد. در اینجا باید به یک نکته‌ی مهم اشاره شود که بسیاری از تولیدات تجاری که صدای شبیه به صحبت انسان ایجاد می‌کنند در واقع ترکیب صحبت انجام نمی‌دهند بلکه تنها یک تکه‌ی ضبط شده به صورت دیجیتال از صدای انسان را پخش می‌کنند. این روش کیفیت صدای بالایی ایجاد می‌کند اما به واژه‌ها و عبارات از پیش ضبط شده محدود است. از کاربردهای عمده‌ی ترکیب صحبت می‌توان به ایجاد ابزارهایی برای افراد دارای ناتوانی بینایی برای مطلع شدن از آنچه بر روی صفحه‌ی کامپیوتر می‌گذرد اشاره کرد.

تشخیص صحبت عبارت است از تشخیص کامپیوتری صحبت تولید شده توسط انسان و تبدیل آن به یک سری فرامین یا پرونده‌های متنی. کاربردهای عمده‌ی موجود برای این گونه سیستمها دربرگیرنده‌ی بازه‌ی گسترده‌ای از سیستمها وکاربردها از سیستمهای دیکته‌ی کامپیوتری که در سیستمهای آموزشی و همچنین سیستمهای

پردازش واژه کاربرد دارد گرفته تا سیستمهای کنترل کامپیوترها به وسیله‌ی صحبت و به طور خاص سیستمهای فراهم آورنده‌ی امکان کنترل کامپیوترها برای افراد ناتوان از لحاظ بینایی یا حرکتی می‌باشد.

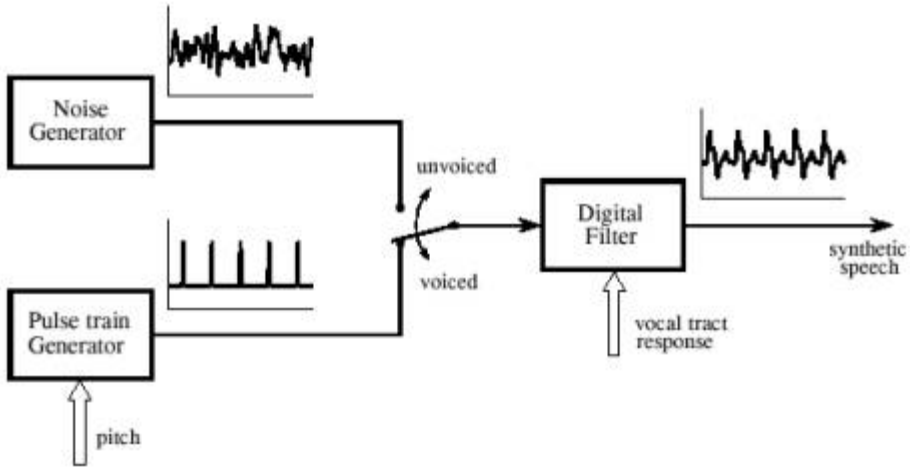
کاربرد مورد نظر ما یعنی تشخیص گوینده از لحاظ نحوه‌ی پیاده‌سازی و استفاده تناسب فراوانی با خانواده‌ی دوم یعنی تشخیص کامپیوتری صحبت دارد، ولی از لحاظ اهداف و کاربردها می‌تواند در خانواده‌ای جداگانه از کاربردهای نیازمند پردازش صحبت قرار گیرد.

ترکیب و تشخیص کامپیوتری صحبت مسائل دشواری هستند. روشهای مختلفی مورد آزمایش قرار گرفته‌اند که موفقیت کمی داشته‌اند. این زمینه از زمینه‌های فعال در تحقیقات پردازش سیگنال دیجیتال (دی.اس.پی) بوده و بدون شک سالها این گونه خواهد ماند. در حال حاضر از ابزارهای برنامه‌نویسی جاافتاده در زمینه‌های برشمرده شده می‌توان به ای.پی.آی صحبت شرکت مایکروسافت [۳] اشاره نمود که دارای تواناییهای عمده‌ای در زمینه‌های تشخیص و ترکیب صحبت است و توانایی آن تا حدی گسترده است که در محصول بزرگ و توانمند MS Office XP از آن استفاده‌ی عملی شده است. ابزار عمده‌ی دیگر تولید شرکت آی.بی.ام است و ViaVoice نام دارد که به لحاظ پشتیبانی آن برای سیستم‌عاملهای متعدد و زبانهای گوناگون از اهمیت و کاربرد خاصی برخوردار است.

۲- مدلی برای توصیف روش تولید صحبت

تقریباً تمام تکنیکهای ترکیب و تشخیص صحبت بر اساس مدل تولید صحبت انسان که در شکل شماره ۳ نشان داده شده است ایجاد شده‌اند. بیشتر صداهای مربوط به صحبت انسان به دو دسته‌ی صدادار [۴] و سایشی [۵] تقسیم می‌شوند. اصوات صدادار وقتی که هوا از ریه‌ها و از مسیر تارهای صوتی به بیرون دهان یا بینی رانده می‌شوند ایجاد میگردند. تارهای صوتی دو رشته‌ی آویخته از بافت هستند که در مسیر جریان هوا کشیده شده‌اند. در پاسخ به کشش ماهیچه‌ای متفاوت تارهای صوتی با فرکانسی بین ۵۰ تا ۱۰۰۰ هرتز ارتعاش می‌کنند که باعث انتقال حرکتهای متناوب هوا به نای می‌شود. در شکل شماره ۳ اصوات صدادار با یک مولد پالس ترین [۶] با پارامتر قابل تنظیم پیچ (فرکانس پایه‌ی موج صوتی) نشان داده شده است.

در مقایسه، اصوات سایشی به صورت نویز تصادفی و نه حاصل از ارتعاش تارهای صوتی به وجود می‌آیند. این حادثه زمانی رخ می‌دهد که تقریباً جریان هوا به وسیله‌ی زبان و لبها یا دندانها حبس می‌شود که این امر باعث ایجاد اغتشاش هوا در نزدیکی محل فشردگی می‌گردد

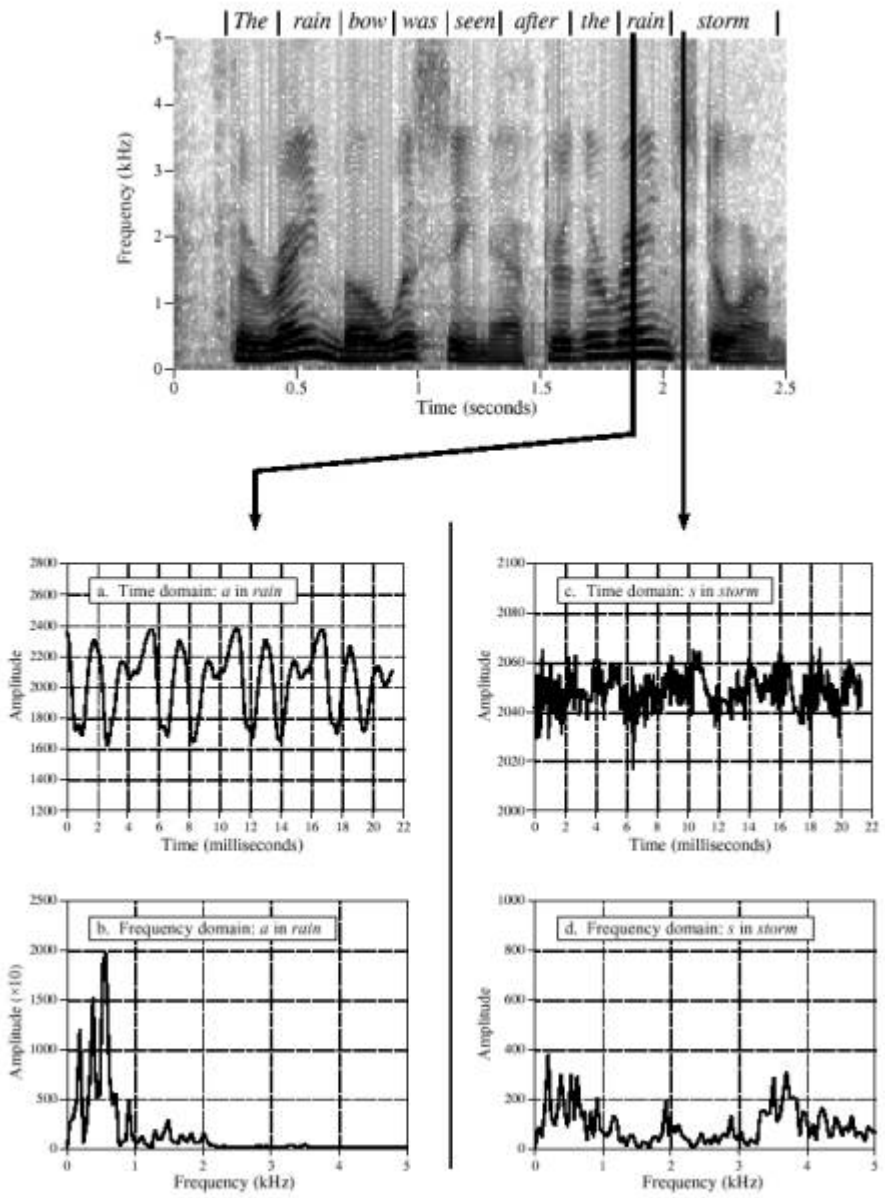


شکل شماره ۳ - مدل صحبت انسان. در یک تکه زمان کوتاه، حدود ۲ تا ۴۰ میلی‌ثانیه صحبت می‌تواند با استفاده از سه پارامتر مدلسازی شود: ۱- انتخاب یک آشفتگی متناوب یا نویزوار. ۲- پیچ آشفتگی متناوب ۳- ضرایب یک فیلتر خطی بازگشتی که پاسخ اثر صوتی را تقلید می‌کند.

اصوات سایشی زبان انگلیسی عبارتند از s، f، sh، z، v و th. در مدل شکل شماره ۳ اصوات سایشی با استفاده از یک مولد نویز نشان داده شده‌اند.

هر دو نوع این اصوات، توسط چاله‌های صوتی که از زبان، لبها، دهان، گلو و گذرگاههای بینی تشکیل شده‌اند دچار تغییر می‌شوند. چون انتشار صدا در این ساختارها یک فرایند خطی است می‌تواند با استفاده از یک فیلتر خطی با یک پاسخ ضربه‌ی مناسب نمایش داده شود. در بیشتر موارد از یک فیلتر بازگشتی که ضرایب بازگشتی آن ویژگیهای فیلتر را مشخص می‌کند استفاده می‌شود. به خاطر این که چاله‌های صوتی ابعادی به اندازه‌ی چند سانتیمتر دارند پاسخ فرکانسی یک دنباله از تشدیدها با اندازه‌های کیلوهرتزی است. در اصطلاح پردازش صوت این قله‌های تشدید فرکانسهای فرمانت[۷] خوانده می‌شوند. با تغییر جایگاه نسبی زبان و لبها فرکانسهای فرمانت هم از لحاظ دامنه و هم از لحاظ فرکانس ممکن است تغییر کنند.

شکل شماره ۴ یک روش معمول برای نمایش سیگنالهای صحبت را نشان می‌دهد که طیف‌نگاره[۸] یا اثر صوت[۹] خوانده می‌شود. سیگنال صوتی به تکه‌های کوچک به اندازه‌ی ۲ تا ۴۰ میلی‌ثانیه تقسیم می‌شوند و از الگوریتم اف.اف.تی برای یافتن طیف فرکانسی هر تکه استفاده می‌شود. این طیفها در کنار هم قرار داده شده تبدیل به یک تصویر سیاه و سفید[۱۰] می‌شود (دامنه‌های پایین روشن و دامنه‌های بالا تیره می‌شوند). این کار یک روش گرافیکی برای مشاهده‌ی این که چگونه محتویات فرکانسی صحبت با زمان تغییر می‌کند به وجود می‌آورد. اندازه‌ی هر تکه بر اساس اعمال یک بده‌بستان بین دقت فرکانسی (که با تکه‌های بزرگ‌تر بهتر می‌شود) و دقت زمانی (که با تکه‌های کوچک‌تر بهتر می‌شود) انتخاب می‌گردد.



شکل شماره ۴- طیف صوت. شکلهای a و b ویژگیهای عمومی اصوات صدادار و شکلهای c و d ویژگیهای عمومی اصوات سایشی را نمایش می‌دهند.

همچنانکه در شکل ۴ دیده می‌شود اصوات صدا دار مثل a در rain دارای موج صوتی متناوبی مانند آنچه در شکل a نشان داده شده و طیف فرکانسی آنها که عبارت است از یک دنباله از همسازهای با اندازه‌ی منظم مانند شکل b می‌باشد در مقابل، اصوات سایشی مانند s در storm دارای یک سیگنال نویزی در دامنه‌ی زمان مانند شکل c و یک طیف نویزی مانند شکل d هستند.این طیفها همچنین شکل فرکانسهای فرمانت برای هر دو نوع صوت نشان می‌دهند. همچنین به این نکته توجه کنید که نمایش زمان-فرکانس کلمه‌ی rain در هر دو باری که ادا شده شبیه به هم است.

در یک دوره‌ی کوتاه برای نمونه ۲۵ میلی‌ثانیه یک سیگنال صحبت می‌تواند با مشخص کردن سه پارامتر تقریب زده شود:

۱) انتخاب یک اغتشاش متناوب یا نویزوار

۲)فرکانس موج متناوب (اگر مورد استفاده قرار گرفته باشد)

۳)ضرایب فیلتر دیجیتالی که برای تقلید پاسخ تارهای صوتی استفاده شده است.

صحبت پیوسته با بروزآوری این سه پارامتر به صورت پیوسته به اندازه‌ی ۴۰ بار در ثانیه ترکیب شود. این راهکار برای یکی از کاربردهای تجاری دی.اس.پی که «صحبت و املا» نامیده می‌شود و یک وسیله‌ی الکترونیکی پرفروش برای بچه‌هاست مناسب است. کیفیت صدای این نوع ترکیب کننده‌ی صحبت پایین است و بسیار مکانیکی و متفاوت با صدای انسان به نظر می‌رسد. ولی در هر صورت نرخ داده‌ی خیلی پایینی در حدود چند کیلوبیت بر ثانیه نیاز دارد.

همچنین این راهکار پایه‌ای برای روش کدگذاری پیشگویانه‌ی خطی (ال.پی.سی)[۱۱] در فشرده‌سازی صحبت فراهم می‌آورد. صحبت ضبط شده‌ی دیجیتالی انسان به تکه‌های کوچک تقسیم می‌شود و هر کدام با توجه به سه پارامتر مدل توصیف می‌شود. این عمل به طور معمول نیاز به یک دوجین بایت برای هر تکه دارد که نرخ داده‌ای برابر با ۲ تا ۶ کیلوبایت بر ثانیه را طلب می‌کند. این تکه‌ی اطلاعاتی ارسال می‌شود و در صورت لزوم ذخیره می‌گردد و سپس توسط ترکیب کننده‌ی صحبت بازسازی می‌شود.

الگوریتمهای تشخیص صحبت با تلاش برای شناسایی الگوهای پارامترهای استخراج شده از این روش نیز پیش‌تر می‌روند. این روشها معمولاً شامل مقایسه‌ی تکه‌های اطلاعاتی با قالبهای صدای از پیش ذخیره شده در تلاش برای تشخیص کلمات گفته شده می‌باشند. مشکلی که در اینجا وجود دارد این است که این روش همیشه به درستی کار نمی‌کند. این روش برای بعضی کاربردها قابل استفاده است اما با تواناییهای شنوندگان انسانی خیلی فاصله دارد.

۳- آینده‌ی فناوریهای پردازش صحبت

ارزش ایجاد فن‌آوریهای ترکیب و تشخیص صحبت بسیار زیاد است. صحبت سریع‌ترین و کاراترین روش ارتباط انسانهاست. تشخیص صحبت پتانسیل جایگزینی نوشتن، تایپ، ورود صفحه‌کلید و کنترل الکترونیکی را که توسط کلیدها و دکمه‌ها اعمال می‌شود را داراست و فقط نیاز به آن دارد که کمی برای پذیرش توسط بازار تجاری بهتر کار کند.

ترکیب صحبت علاوه بر آن که همانند تشخیص صحبت می‌تواند استفاده از کامپیوتر را برای کلیه‌ی افراد ناتوان بدنی که دارای تواناییهای شنوایی و گفتاری مناسب هستند آسان‌تر سازد به عنوان یک وسیله‌ی خروجی کاربرپسند در محیطهای مختلف می‌تواند با جایگزین کردن بسیاری از علائم دیداری (انواع چراغها و...) و شنوایی (انواع زنگهای اخطار و ...) با گفتارهای بیان کننده‌ی کامل پیامها استفاده از و رسیدگی به سیستمهای نیازمند این گونه پیامها را بهینه کند.

در اینجا لازم است به این نکته اشاره شود که پیشرفت در فن‌آوری تشخیص صحبت (و همچنین تشخیص گوینده) همان قدر که محدوده‌ی دی.اس.پی را در بر می‌گیرد نیازمند دانش به دست آمده از محدوده‌های هوش مصنوعی و شبکه‌های عصبی است. شاید این تنوع دانشهای مورد نیاز به عنوان عامل دشواری مطالعه‌ی مبحث پردازش صحبت در نظر گرفته شود حال آن که این گونه نیست و این تنوع راهکارها بخت رسیدن به سیستم با کارایی مطلوب را افزایش می‌دهد.

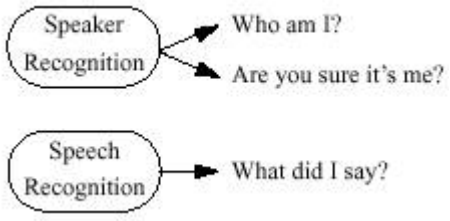
تواناییهای ابزارهایی که در بخش اول این فصل به آنها اشاره شد امیدواریهای فراوانی را در زمینه‌ی موفقیت ابزارهای موجود فراهم می‌آورد و دامنه‌ی وسیع شرکتها و مراکز دانشگاهی که در این زمینه فعالیت می‌کنند بر تنوع در قابلیتها و کاربردهای پیاده‌سازی شده‌ی این ابزارها می‌افزاید.

۱- اهمیت مدلسازی سیگنال

تشخیص کامپیوتری صحبت در واقع بر دارنده‌ی دو نوع عمل اصلی تشخیص است: تشخیص صحبت و تشخیص گوینده. با تحلیل یک موج صوتی می‌توان خصیصه‌های[۱] اندامهای گفتاری گوینده را تخمین زد که این خصیصه‌ها راهکاری برای تشخیص هویت و تصدیق آن به روش زیست‌سنجی فراهم می‌آورند. در مقابل، سیستمهای تشخیص صحبت برای درک مفهوم موج صوتی گفته شده تلاش می‌کنند. جهت بیشتر تحقیقات فعلی در فن‌آوری تشخیص صحبت به سمت ایجاد سیستمهای مستقل از گوینده است که توانایی تبدیل صحبت همه‌ی گویندگان را داشته باشد. در حالی که اهداف این دو نوع سیستم کاملاً متفاوت به نظر می‌رسند هر دو عمیقاً از آبشخوری به نام الگوریتمهای پردازش سیگنال برای استخراج خصیصه‌ها[۲] تغذیه می‌شوند. در هر دو زمینه تلاش برای پیدا کردن دسته‌ای از خصیصه‌ها که در مقابل تغییرات محیطی پایدار باشند ادامه دارد. این قسمت مروری خواهد داشت بر الگوریتمهای استخراج خصیصه‌ها که در هر دو زمینه استفاده شده‌اند و شامل ارزیابی کوتاهی از الگوریتمهای گوناگون مدلسازی سیگنال با آزمایشهای تشخیصی کوچک می‌باشد.

۲- آشنایی با مدلسازی سیگنال

هدف سیستمهای تشخیص گوینده بازشناسی خصیصه‌های اندامهای گفتاری و حالت صحبت کردن با استفاده از صدای گوینده به منظور اهداف تشخیص هویتی می‌باشد. ساختار اندامهای صوتی، اندازه‌ی چاله‌ی بینی و ویژگیهای تارهای صوتی همگی با استفاده از تحلیل سیگنال قابل تخمین هستند. تشخیص گوینده اصطلاحی کلی است که به اعمال تشخیص هویت گوینده و تأیید هویت گوینده اطلاق می‌گردد. برای تشخیص، خصیصه‌های تخمینی گوینده با خصیصه‌های موجود در یک پایگاه داده‌ها از کاربران ثبت شده برای یافتن نزدیک‌ترین خصیصه‌های قابل تطبیق مقایسه می‌شوند. برای تأیید هویت، ادعای هویتی گوینده بر اساس امضای زیست‌سنجی وی پذیرفته می‌شود و یا رد می‌گردد.



شکل شماره ۱ – وظایف مختلف

تشخیص صحبت تلاش دارد تا یک سیگنال صوتی صحبت را به واژه‌ها تبدیل کند. انسانها واژه‌ها را با حرکت دادن اندامهای صوتی به یک سری از مکانهای قابل پیشبینی ادا می‌کنند. اگر این دنباله‌ها از سیگنال استخراج گردند واژه‌های گفته شده می‌توانند تشخیص داده شوند. بسیاری از کاربردهای تشخیص صحبت نیازمند سیستمهای مستقل از گوینده می‌باشند این تولیدات می‌توانند صحبت هر گوینده‌ای را تشخیص دهند.

اگر چه این دو هدف کاملاً متفاوت به نظر می‌رسند هر دوی آنها بر روی داده‌های صحبت تشخیص الگو را اعمال می‌کنند. بعضی از سیستمهای موجود مانند server ۶Nuance هم تشخیص صحبت و هم تأیید هویت گوینده را به صورت همزمان اعمال می‌کنند. به خاطر همین شباهت رویه هر دوی این فن‌آوریها از یک نقطه ضربه می‌خورند: یک تنزل کارایی شدید در اثر تفاوتهای محیطهای آموزشی و آزمایشی به وجود می‌آید. به طور خلاصه کارایی این فن‌آوریها شدیداً به محیطی که در آن توسعه می‌یابند وابسته است و بنابراین حالات پر از نویز جهان واقعی آنها را به کارایی زیر کارایی بهینه راهبری می‌کند.

الگوریتمهایی مورد استفاده‌ی محصولات پردازش کننده‌ی صحبت بر اساس مدل صوتی ناحیه‌ی صوتی و کانال گوش استوارند. بخش بعدی اهمیت استخراج خصیصه‌ها را با یک مرور کلی از تشخیص الگو روشن می‌کند و سپس با توصیف الگوریتمهای رایج در محصولات پراستفاده ادامه پیدا می‌کند.

۳- تشخیص الگو

یک سیستم تشخیص الگو شامل دو جزء است: یک استخراج کننده‌ی خصیصه‌ها و یک طبقه‌بندی کننده. ایده‌آل آن است که وقتی داده‌ها به فضای داده‌های خصیصه‌ها انتقال پیدا کرد به سمت طبقه‌ای کشیده شود که از همه به آن نزدیک‌تر است و از طرف طبقه‌های[۳] متفاوت دیگر بازپس زده شود. وقتی که به طبقه‌بندی کننده[۴] آموزش داده شد که بین طبقه‌ها در این فضای انتقال داده شده از خصیصه‌ها تمایز قائل شود یک سیستم تشخیص نیازمند آن است که تنها داده‌های ورودی را از طریق همان سیستم استخراج خصیصه‌ها انتقال دهد و مشخص کند که در کدام طبقه یک مشاهده‌ی جدید رخ می‌دهد.

دو مشکل مهم در اعمال این راهکار به پردازش صحبت وجود دارد. اولی آن است که هیچ التزامی وجود ندارد که محیط آموزش و محیط آزمایش قابل مقایسه باشند. استفاده از یک میکروفون متفاوت، نویز پس‌زمینه و کانالهای انتقال می‌تواند باعث کاهش کارایی جدی شود (یک معیار اساسی برای قضاوت در مورد یک مجموعه از خصیصه‌ها پایداری آن در مقابل چنین تغییرات کانالی می‌باشد) . دومین مشکل آن است که که برهمنهی زیادی بین طبقه‌های موجود در فضای خصیصه‌ها وجود دارد. ژائو[۵] نمودارهایی برای نشان دادن این برهمنهی در دو دسته داده‌های صحبت جمع‌آوری شده از طریق شبکه‌ی تلفن ارائه می‌کند. موتورهای تشخیص صحبت برای غلبه بر این مشکل برهمنهی از پردازشهای آماری توانمند برای یکسان‌سازی مدل زبان استفاده می‌کنند که فراتر از حد این نوشتار است.

۴- الگوریتمهای مدلسازی سیگنال

هدف مدلسازی سیگنال (که اغلب از آن با عنوان استخراج خصیصه‌ها یاد می‌شود) انتقال داده‌های صوتی به فضایی است که مشاهدات مربوط به یک طبقه با هم در یک گروه قرار گیرند و مشاهدات مربوط به طبقات متفاوت از هم جدا شوند. این انتقالها بر اساس مطالعات زیست‌شناختی سیستمهای صوتی و اندامهای گفتاری انسان انتخاب می‌شوند. برای مثال اندامهای گفتاری نمی‌توانند از یک مکان به مکان دیگر در کمتر از حدود پنج میلی‌ثانیه جابه‌جا شوند لذا سیستمهای عملی می‌توانند از طیف ۱۰۰ بار در ثانیه نمونه‌برداری کنند در حالی که از دقت عملیات فقط مقدار بسیار کمی کاسته شود.

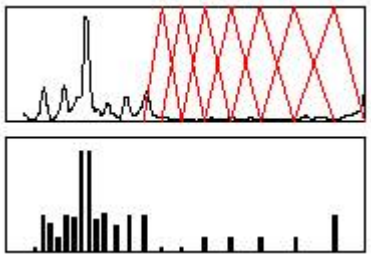
صحبت یک سیگنال پویاست لذا ما علاقمند به آزمون طیف بازه‌ی کوچک هستیم. زمان استمرار یک قاب به صورت طول زمانی که یک مجموعه از پارامترها معتبر هستند تعریف می‌شوند. با وجود این که قابها همپوشانی ندارند ما معمولاً از پنجره‌ی تحلیل دارای همپوشانی برای در نظر داشتن تعداد بیشتری از نمونه‌های سیگنال برای هر اندازه‌گیری طیف استفاده می‌کنیم. اعمال مستقیم تحلیل طیفی بر روی چنین مقدار کمی از داده‌ها معادل با اعمال یک پنجره‌ی مستطیلی تیز به سیگنال است که باعث ایجاد اعوجاج طیفی می‌شود. پاسخ فرکانسی پالس مستطیلی یک تابع sinc می‌باشد($\text{sinc } x=\sin x/x$) که دارای یک باند عبور منحنی شکل و مقدار زیادی ناهمواری در باند توقف می‌باشد. شکلهای مختلف برای پنجره‌ها از طریق اعمال یک تابع وزن به دست می‌آیند. پنجره‌ی همینگ[۶] با رابطه‌ی

$$w(n)= (a-(۱-a)\cos(۲p/[N-۱])/ b$$

یک نمونه‌ی ویژه از پنجره‌ی هنینگ[۷] با a =۰,۵۴ می‌باشد (p عدد پی (... ۳,۱۴۱۵) است). پارامتر b برای هنجارسازی به گونه‌ای انتخاب می‌شود که انرژی سیگنال در خلال آزمایش بدون تغییر باقی بماند. شکل پنجره‌ی همینگ یک تحلیل طیفی با باند عبور هموارتر و باند توقف به طور قابل ملاحظه‌ای بدون اعوجاج به دست می‌دهد که هر دوی این خصوصیات برای به دست آوردن تخمینهای پارامتری متغیر مهم هستند. بیشتر سیستمهای امروزی از یک از یک فریم با اندازه‌ی زمانی ۱۰ میلی‌ثانیه و یک پنجره با اندازه‌ی زمانی ۲۵ میلی‌ثانیه استفاده میکنند.

یک خصیصه‌ی استخراج شده از سیگنال انرژی مطلق سیگنال است. دسته‌ی دیگر، اندازه‌گیری طیفی انرژی فرکانسهای خاص است. این اندازه‌ها مشابه حالات اولیه‌ی حرکات دستگاه صوتی انسان هستند (سلولهای مو در حلزون گوش برای دستیابی به هدف مشابهی استفاده می‌شوند). سه راه برای دستیابی به این اندازه‌های صوتی وجود دارد: اعمال مستقیم یک بانک فیلتر دیجیتال در دامنه‌ی زمان، استفاده از تبدیل فوریه و تحلیل پیشگویانه‌ی خطی. دو روش اخیر به لحاظ کارایی محاسباتی در سیستمهای امروزی رایج‌ترند.

از آنجا که شنوایی انسان در طول یک اندازه‌ی خطی به صورت مساوی حساس نیست، ما طیف را به یک اندازه‌ی فرکانسی قابل درک[۸] نقش می‌کنیم. تجربیات در مورد ادراک انسان نشان داده‌اند که فرکانسهایی با یک پهنای باند معین یک فرکانس اسمی که به پهنای باند بحرانی معروف است نمی‌توانند به صورت جداگانه از هم تشخیص داده شوند. اندازه‌ی مل[۹] یک تقریب ساده‌تر است که پیچ قابل مشاهده‌ی یک صدا را به اندازه‌ی خطی نقش می‌کند. استیونز[۱۰] و فولکمن[۱۱] در سال ۱۹۴۰ به صورت تجربی نگاشتی بین اندازه‌ی مل و فرکانسهای واقعی تعیین کردند. تفاوت اندازه به سختی به صورت خطی زیر ۱۰۰۰هرتز و به صورت لگاریتمی بالای ۱۰۰۰هرتز می‌باشد.



شکل شماره ۲- بانکهای فیلتر با فضای مثلثی مل

بانکهای فیلتر مبتنی بر تبدیل فوریه‌ی ساده که برای خصیصه‌های نهایی طراحی شده‌اند دقت فرکانسی دلخواه را بر اساس مقیاس مل[۱۲] به دست می‌دهند. برای پیاده‌سازی این بانک فیلتر پنجره‌ی داده‌های صحبت با استفاده از تبدیل فوریه به دامنه‌ی فرکانس انتقال می‌یابد. در دامنه‌ی فرکانس ضرایب دامنه‌ی هر بانک فیلتر با اعمال یک ترکیب خطی از طیف و پاسخ فرکانسی فیلتر دلخواه پیدا می‌شوند. در عمل بانکهای فیلتر مثلثی دارای برهم‌نهی استفاده می‌شوند که در آن از فرکانس مرکزی یک فیلتر به عنوان نقاط انتهایی دو فیلتر مجاور استفاده می‌شود. بنابراین ضرایب دامنه‌ی هر بانک فیلتر مقدار متوسط طیف در کانال فیلتر را نشان می‌دهند:

$$S_{avg}(f) = \frac{1}{N_s}\sum_{n=0}^{N_s} w_{FB}(n)\,S(f)$$

که در آن N(s تعداد نمونه‌های استفاده شده برای دستیابی به مقدار متوسط و W(n تابع وزنیابی (مشابه تابع مثلثی که قبلاً توضیح داده شد) می‌باشد و S(f مقدار پاسخ فرکانسی است که با تبدیل فوریه محاسبه می‌شود.

تحلیل پیشگویانه خطی[۱۳] وسیله‌ای برای به دست آوردن پوشش طیفی هموار P(w از یک مدل تمام- قطب طیف توان است. ضرایب خطی پیشگو همبستگی مستقیمی با نسبتهای ناحیه‌ی لگاریتمی که پارامترهای هندسی مدل لوله‌ای نقصان برای تولید صحبت هستند دارد. دامنه‌های بانک فیلتر با نمونه‌برداری از مدل طیفی پیشگویانه‌ی خطی در فرکانسهای بانک فیلتر مناسب به دست می‌آیند. این کار می‌تواند با ارزیابی مستقیم مدل ال.پی.سی انجام شود ولی در عمل تبدیل فوریه بر روی ضرایب پیشگو اعمال می‌شود. چون تعداد ضرایب ال.پی.سی کمتر از نمونه‌های صوت است این روش از لحاظ محاسباتی کاراست. ضرایب دامنه‌ی بانک فیلتر همان گونه که از طیف حاصل از تبدیل فوریه[۱۴] به دست می‌آمدند از طیف حاصل از پیشگویانه‌ی خطی[۱۵] به دست می‌آیند.

یک سیستم همریخت[۱۶] برای پردازش صحبت قابل استفاده است زیرا روشی برای جدا کردن سیگنال آشفتگی از شکل ناحیه‌ی صوتی فراهم می‌آورد. یک فضای دارای این ویژگی سپستروم[۱۷] است که با محاسبه‌ی عکس تبدیل فوریه‌ی گسسته‌ی لگاریتم انرژی به دست می‌آید.ضرایب سپسترال با محاسبه‌ی دامنه‌های بانک فیلتر با استفاده از معادله‌ی زیر به دست میآیند:

$$c(n) = \frac{1}{N_s}\sum_{k=0}^{N_s} \log\left|S_{avg}(k)\right| e^{j\frac{2\pi}{N_s}kn}, \quad 0 \le n \le N_s - 1$$

که S(avg) مقدار متوسط سیگنال در کانال kام فیلتر است. در عمل تبدیل کسینوسی گسسته به خاطر کارایی محاسباتی استفاده می‌شود. ضرایب سپسترال اغلب برای کمینه کردن تغییراتی که منجر به ایجاد اطلاعات نمی‌شوند وزنیابی می‌گردند که این پردازه لیفترینگ[۱۸] نامیده می‌شود. جالب است بدانیم که در ادبیات تشخیص صحبت خصیصه‌های مربوط به گوینده به عنوان تغییرات غیر دادهزا حذف می‌گردند ولی سیستمهای تشخیص گوینده نیز از لیفترینگ استفاده می‌کنند.

هر دو نوع سیستم تشخیص صحبت و تشخیص گوینده اطلاعات موضعی زمان کوتاه را با گرفتن مشتق خصوصیات اولیه نسبت به زمان به دست می‌دهند. به عنوان مثال یک صوت صدادار می‌تواند با پیدا شدن فرمانتهای[۱۹] آن در طیف تشخیص داده شود، حال آن که یک صوت بی‌صدا (سایشی) با استفاده از انتقال طیف مدل می‌شود. مقادیر مشتق مرتبه‌ی اول خصائص ضرایب دلتا[۲۰] و مقادیر مشتق مرتبه‌ی دوم آن شتاب[۲۱] یا ضرایب دلتا-دلتا[۲۲] نامیده می‌شوند.مشتق زمانی با استفاده از یک رابطه‌ی رگرسیون که یک مجموعه فریم را پیش و پس از فریم کنونی می‌کشد تقریب زده می‌شود.

سیستمهای تشخیص گوینده از یک پیمانه‌ی انتخاب خصیصه نیز در چارچوب تشخیص الگو استفاده می‌کنند. برای تشخیص صحبت تمامی سیگنال باید به یک نمایش متنی نگاشته شود حال آن که سیستم تشخیص گوینده نیازی به کار تحت این اجبار ندارد. بنابراین پیمانه‌ی انتخاب خصیصه فقط خصیصه‌ها مربوط به اصوات صدادار را ذخیره می‌کند. اصوات صدادار مستقیماً فرضیات مدلسازی پیشگویانه‌ی خطی را برآورده می‌سازند و کمتر تحت تأثیر نویز صوتی قرار می‌گیرند.

۱ـ مقدمه

همچنان که پیش از این گفته شد سیستمهای تشخیص گوینده[۱] در حالت کلی به دو نوع سیستمهای تأیید هویت گوینده و سیستمهای بازشناسی گوینده[۲] تقسیم می‌شوند. تفاوت این دو سیستم در نحوه‌ی پذیرش ورودی است: در سیستمهای نوع اول گوینده با ارائه‌ی یک شناسه ادعای هویت یک کاربر خاص را می‌نماید حال آن که در سیستمهای نوع دوم گوینده فقط عبارت عبور خود را بیان می‌کند و سیستم او را از بین تمامی کاربران خود تشخیص می‌دهد.

در فصل قبل در مورد ساختار الگوهای مورد بحث صحبت کردیم و متوجه شدیم که عمل مدلسازی سیگنال یا استخراج خصیصه‌ها[۳] با حذف ویژگیهای بدون استفاده‌ی سیگنال صحبت و حفظ ویژگیهای قابل استفاده برای بازشناسی عبارات خاص الگوهایی را با ویژگیهای انتخاب شده در اختیار ما قرار می‌دهد.

ساختارهایی که برای هر دو نوع سیستم ارائه شد هر دو دارای یک مرحله برای تشخیص میزان شباهت الگوهای متعلق به گوینده‌ی حاضر با گوینده‌ی مورد ادعا (نوع اول) یا همه‌ی گویندگان است که با استفاده از آن معیاری برای تصمیم گیری در اختیار ما قرار داده می‌شود.

همچنان که برای تشخیص الگو الگوریتمهای متعدد و روشهای گوناگون وجود دارد الگوریتمهای گوناگونی نیز برای یافتن میزان شباهت میان الگوها وجود دارد که انتخاب هر کدام از آنها بستگی به ساختار سیستم مقصد دارد.

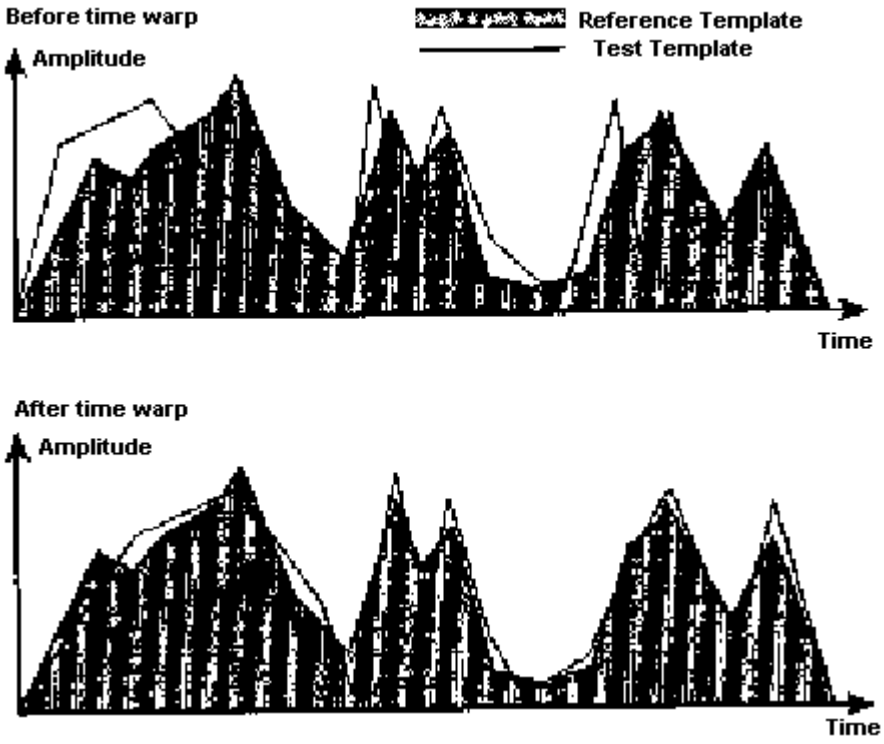
انتخاب یک روش به ویژگیهای سیستم هدف بستگی دارد. بعضی از روشهای موجود تنها می‌توانند فقط برای سیستمهای وابسته به متن[۴] یا فقط برای سیستمهای مستقل از متن[۵] مورد استفاده قرار گیرند و بعضی می‌توانند برای هر دو نوع مورد استفاده قرار گیرند.

بحث این فصل که سه روش عمده‌ی یافتن میزان شباهت الگوها را به صورت کلی مورد بحث قرار خواهد داد عملاً پیش‌زمینه‌های نظری لازم برای طراحی سیستم هدف را کامل می‌کند.

۲- روشهای مبتنی بر چشمپوشی زمانی پویا[۶]

این روش کلاسیک برای تشخیص خودکار گوینده در حالت وابسته به متن بر اساس یکسان‌سازی الگوها با استفاده از الگوهای طیفی[۷] یا روش طیف‌نگاره[۸] استوار است. در حالت کلی سیگنال صحبت به صورت یک دنباله از بردارهای خصیصه[۹] که رفتار سیگنال صحبت را برای یک گوینده‌ی خاص مشخص می‌کند نمایش داده می‌شود. یک الگو می‌تواند نمایشگر یک عبارت چند کلمه‌ای، یک کلمه‌ی منفرد، یک هجا یا یک صدای ساده باشد.

در روشهای یکسان‌سازی الگوها مقایسه‌ای بین الگوی عبارت ورودی و الگوی مرجع برای تشخیص هویت گوینده انجام می‌گیرد. یک جزء مهم در این روشها بهنجارسازی تغییرات زمانی هر آزمون تا آزمون بعدی می‌باشد. بهنجارسازی می‌تواند با روش چشمپوشی زمانی پویا صورت گیرد. این روش یک تابع بهینه‌ی توسیع/ فشرده‌سازی زمانی را برای ایجاد صف‌بندی زمانی غیرخطی به کار می‌گیرد. شکل ۱ الگوها را پیش و پس از اعمال این روش نشان می‌دهد. به این نکته توجه شود که چگونه چشمپوشی الگوهای نمونه‌ی آزمون میزان نزدیکی دو الگو را افزایش داده است:



شکل شماره ۱ – نمونه‌ی یک الگو پیش و پس از اعمال روش چشمپوشی زمانی پویا

در شکل شماره‌ی ۱ فریمهای صحبت که الگوهای آزمون و مرجع را به وجود می‌آورند به صورت مقادیر دامنه‌ای اسکالر بر روی نموداری که محور افقی آن نشانگر زمان است نشان داده شده‌اند. بنابراین یک تابع تصمیم‌گیری با جمع‌آوری اندازه‌گیریها بر حسب زمان می‌تواند محاسبه شود. در عمل الگوها بردارهای چند بعدی هستند و فاصله بین آنها به صورت فاصله‌ی اقلیدسی[۱۰] مورد محاسبه قرار می‌گیرد. نوع دیگر فاصله که برای مقایسه‌ی دو مجموعه از ضرایب پیشگویانه‌ی خطی مورد استفاده قرار می‌گیرد فاصله‌ی ایتاکورا[۱۱] می‌باشد.

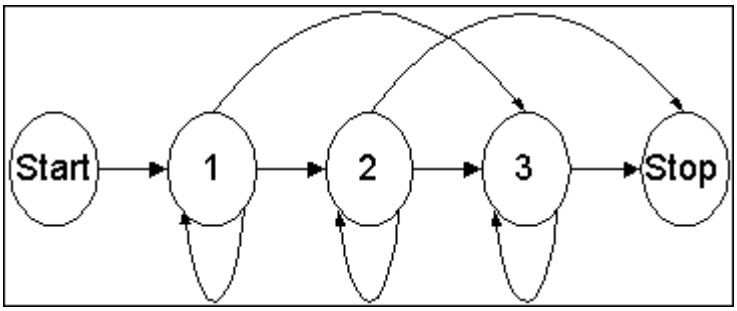
۳- روشهای مبتنی بر مدلهای نهان مارکف[۱۲]

روشهای مبتنی برمدل نهان مارکف جایگزینهایی برای روش یکسان‌سازی الگوها که توسط روشهای چشمپوشی زمانی پویا ارائه شد می‌باشند که مدلهای احتمالی از سیگنال صحبت به وجود می‌آورند که ویژگیهای متغیر با زمان آن را توصیف می‌کند. یک مدل نهان مارکف یک فرایند اتفاقی دوگانه[۱۳] برای ایجاد یک دنباله از نشانه‌های مشاهده شده است. معنای دوگانه بودن این فرایند اتفاقی آن است که این فرایند دارای یک زیرفرایند اتفاقی دیگر است که قابل مشاهده نمی‌باشد (از اینجا مفهوم عبارت نهان مشخص می‌گردد) ولی می‌تواند توسط فرایند اتفاقی دیگری که یک دنباله از مشاهدات را ایجاد می‌کند مشاهده گردد. در سیستمهای نشخیص صحبت یا تشخیص گوینده دنباله‌ی موقتی طیف صوتی می‌تواند به صورت یک زنجیره‌ی مارکف[۱۴] مدلسازی شود تا روشی را که یک صدا به صدای دیگری تبدیل می‌شود توصیف کند. این عمل سیستم را تا اندازه‌ی یک مدل که قادر است فقط در یکی از یک تعداد متناهی از حالات متفاوت باشد (به عنوان نمونه یک ماشین حالت متناهی[۱۵] کوچک می‌کند. روشهای مبتنی بر مدل نهان مارکف می‌توانند هم در سیستمهای وابسته به متن و هم در سیستمهای مستقل از متن مورد استفاده قرار گیرند.

وقتی که بعد از یک انتقال حالت وارد یک حالت دیگر در ماشین حالت متناهی می‌شویم یک نشانه از مجموعه نشانه‌های آن حالت به عنوان خروجی برگزیده می‌شود. خروجی می‌تواند یک تعداد متناهی (روش مدل نهان مارکف گسسته) و یا یک مقدار پیوسته از خروجیها (روش توزیع پیوسته) باشد. هر دو مدل به صورت مؤثر

اطلاعات موقتی را مدلسازی می‌کنند. سیستم در بازه‌های منظم زمانی تغییر حالت می‌دهد. حالتی که مدل در هر آغاز هر بازه‌ی زمانی به آن می‌رود به احتمالات بستگی دارد.

تعدادی توپولوژی مدل که برای نمایش ماشین حالت متناهی استفاده می‌شوند وجود دارند. یک ساختار معمول ساختار چپ به راست است که به آن مدل بکیس[۱۶] هم گفته می‌شود و مثال آن نمونه‌ای است که در شکل ۲ نشان داده شده است. هر حالت یک انتقال توقف[۱۷] ، یک انتقال پیش‌رونده[۱۸] و یک انتقال جهشی[۱۹] دارد. با وجود آن که دز شکل نشان داده نشده است احتمالهای مختلفی به انتقالهای حالت متناهی وابسته‌اند و همچنین خروجی هر حالت را کنترل می‌کنند. نوع دیگر توپولوژی مدل نهان مارکف که در اینجا نشان داده نشده ساختار ارگودیک[۲۰] می‌باشد که در آن همانند یک شبکه‌ی کاملاً متصل به هم هر حالت به همه‌ی دیگر حالات دارای انتقال است.

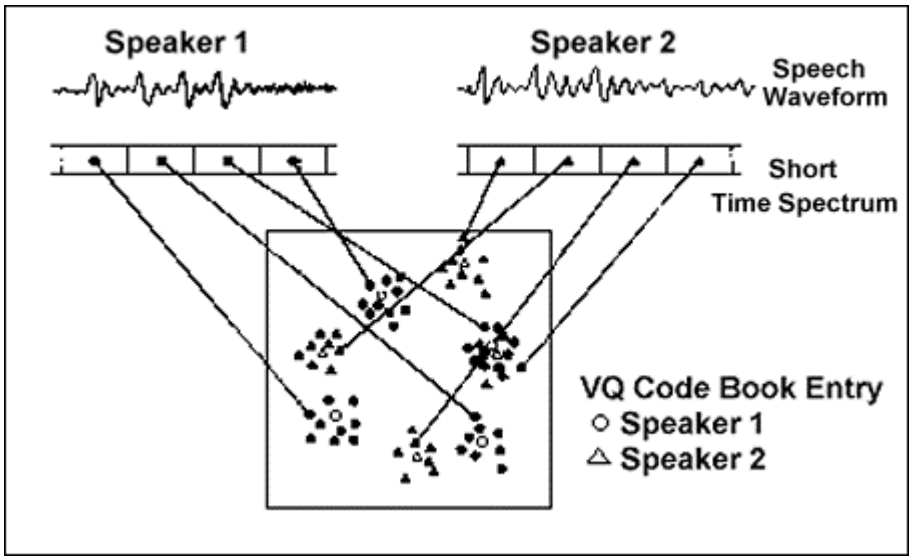


شکل شماره ۲ – مثالی از ساختار مدل نهان مارکف چپ به راست

۴- روشهای مبتنی بر مقدارگزینی برداری[۲۱]

یک مجموعه از بردارهای خصیصه‌ی بازه‌ی کوتاه زمانی یک گوینده که برای آموزش سیستم به سیستم داده می‌شوند می‌توانند مستقیماً برای نمایش ویژگیهای مهم عبارت ایراد شده توسط وی به کار گرفته شوند. در هر صورت نتیجه‌ی کار آن است که نیازمندیهای حافظه برای ذخیره‌ی داده‌ها و پیچیدگی محاسباتی به سرعت با افزایش تعداد بردارهای آموزش دهنده‌ی سیستم افزایش می‌یابد. بنابراین یک نمایش مستقیم عملی نخواهد بود.

مقدارگزینی برداری اساساً روشی برای فشرده‌سازی داده‌های آموزش دهنده‌ی سیستم تا اندازه‌ای قابل مدیریت و کارا می‌باشد. با استفاده از یک دفتر کد[۲۲] مقدارگزینی برداری که شامل تعداد کمی بردارهای خصیصه با نمایانگری بالاست می‌توان داده‌های اولیه را به مجموعه‌ی کوچکی از نقاط نمایانگر کاهش داد. مقدارگزینی برداری هم در سیستمهای وابسته به متن و هم در سیستمهای مستقل از متن قابل استفاده است.



شکل شماره ۳ – نمودار مفهومی که شکلگیری یک دفتر کد مقدارگزینی برداری را به تصویر می‌کشد

شکل ۳ یک نمودار مفهومی را که مثالی از شکلگیری یک دفتر کد مقدارگزینی برداری را به تصویر می‌کشد نشان می‌دهد. یک گوینده می‌تواند بر اساس مکان مرکز ثقل بردارها از دیگری تشخیص داده شود. در شکل ۳ خصیصه‌های طیفی زمان کوتاه با یک فضای اقلیدسی دوبعدی نشان داده شده‌اند. برای ایجاد یک مجموعه از نقاط گامهای زیر اجرا شده‌اند:

- از دو گوینده خواسته شده تا چند دنباله عبارت برای آموزش سیستم بیان کنند.

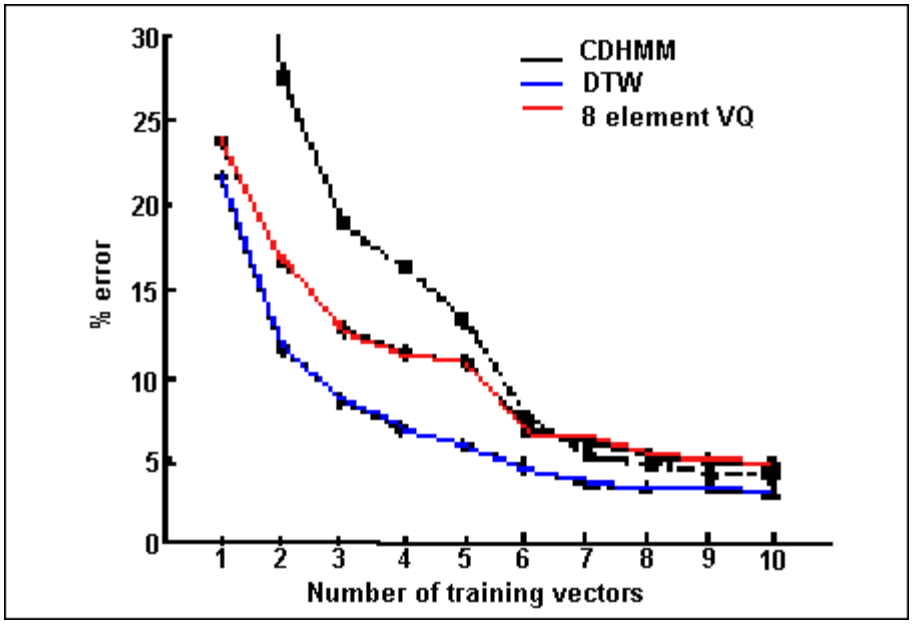
- دنباله‌های آموزش دهنده‌ی سیستم تحلیل می‌شوند و برای آموزش دفتر کد مقدارگزینی برداری استفاده می‌گردند.

- سپس نقاط به بخشهای جداگانه افراز می‌گردند و دو دفتر کد تولید می‌گردد که هر کدام چهار عنصر دارند. عناصر دفتر کد مقدارگزینی برداری به صورت دایره و مثلث نمایش داده می‌شوند و مرکز ثقل بخشهای مرتبط با فضای خصیصه‌ی هر گوینده را نشان می دهند.

همچنان که در شکل ۳ قابل مشاهده است با وجود کمی روی‌هم‌افتادگی دو دفتر کد هنوز کاملاً مجزا هستند و بنابراین هر گوینده می‌تواند از دیگری تشخیص داده شود. هدف آموزش یک دفتر کد مقدارگزینی برداری یافتن افرازهای مناسب از یک فضای برداری به صورت تعدادی ناحیه‌ی بدون روی‌هم‌افتادگی می‌باشد. هر افراز با یک بردار مرکز ثقل مرتبط نشان داده می‌شود. روشی معمول برای یافتن یک افرازبندی مناسب استفاده از یک رویه‌ی بهینه‌سازی مانند الگوریتم تعمیم‌یافته‌ی لوید[۲۳] که آشفتگی متوسط در بین بردارهای آموزش سیستم و مرکز ثقلها را کمینه می‌کند می‌باشد. سایر روشها عبارتند از معیار کمترین بیشینه[۲۴] (کمینه کردن بیشترین آشفتگی) که الگوریتم پوشش[۲۵] نیز نامیده می‌شود و استفاده از قانون Kامین همسایه‌ی نزدیک[۲۶] به جای قانون نزدیک‌ترین همسایه در محاسبه‌ی آشفتگی.

۵- مقایسه‌ی کارایی

آزمایشهای گوناگونی برای تعیین این که کدام روش برای تشخیص گوینده بهترین روش است صورت گرفته است و مهم است که به این نکته توجه شود که چگونه محققان مختلف در وضعیتهای گوناگون به نتایج متفاوتی دست پیدا نموده‌اند. به عنوان نمونه اروین[۲۷] در نوشتار خود در ارتباط با آزمایشهایی که وی در زمینه‌ی سیستمهای وابسته به متن برای مقایسه‌ی سه روش برشمرده شده انجام داده است به این نتیجه رسیده است که روش مقدارگزینی برداری بهترین کارایی را ارائه می‌کند. حال آن که یو[۲۸] ، میسن[۲۹] و اگلبی[۳۰] در مقاله‌ی خود اشاره به اجرای آزمایشهایی مشابه نموده‌اند که نتایج متفاوتی را احراز نموده‌اند. نتیجه‌ی تجربه‌ی آنان که در بردارنده‌ی آزمایشهایی برای سه روش توضیح داده شده برای سیستمهای وابسته به متن و دو روش متأخر برای سیستمهای مستقل از متن است نمودار شکل ۴ برای سیستمهای مستقل از متن و شکل ۵ برای سیستمهای مستقل از متن است. همچنان که در شکل ۴ مشاهده می‌شود بر اساس تجربیات این گروه روش چشمپوشی زمانی پویا دارای بهترین کارایی است و همچنین روشهای مدل نهان مارکف با چگالی پیوسته[۳۱] و مقدارگزینی برداری هشت‌عنصری استفاده شده به ازای تعداد بردارهای آموزش سیستم متفاوت کاراییهای متفاوت دارند:

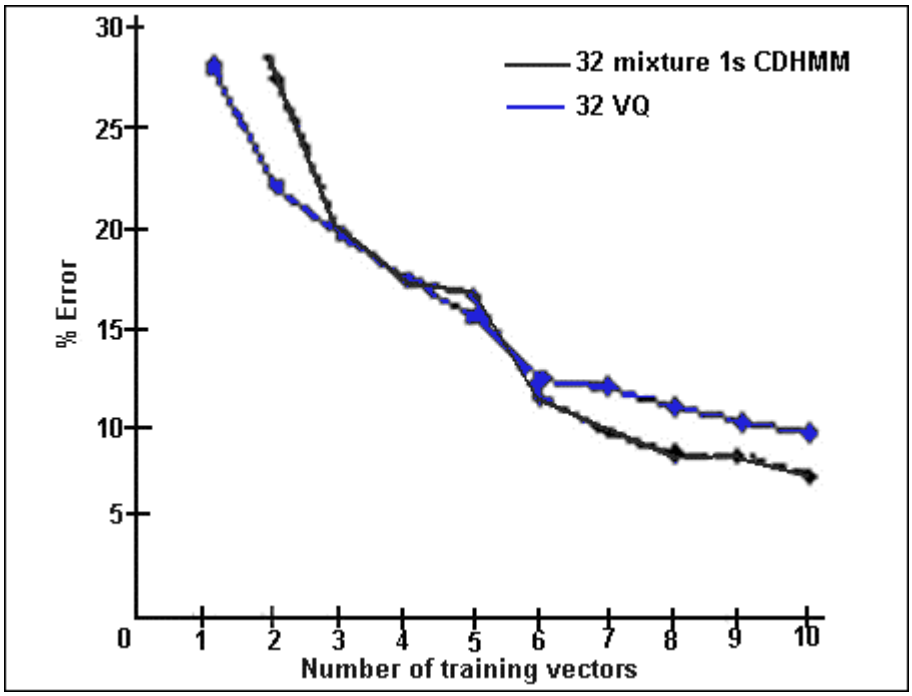


شکل شماره ۴ – درصد خطا بر اساس تعداد بردارهای آموزش سیستم برای روشهای وابسته به متن چشمپوشی زمانی پویا، مقدارگزینی برداری ۸عنصری و مدل نهان مارکف با چگالی پیوستهی ۸ حالتهی ۱ ترکیبه

همچنین از روی نمودار میتوان نتیجه گرفت که با وجود آن که برای تعداد بردارهای آموزش کم روش چشمپوشی زمانی پویا عملکرد بهتری دارد با افزایش تعداد بردارها این تفاوت عملکرد دیگر به صورت واضح مشاهده نمیشود.

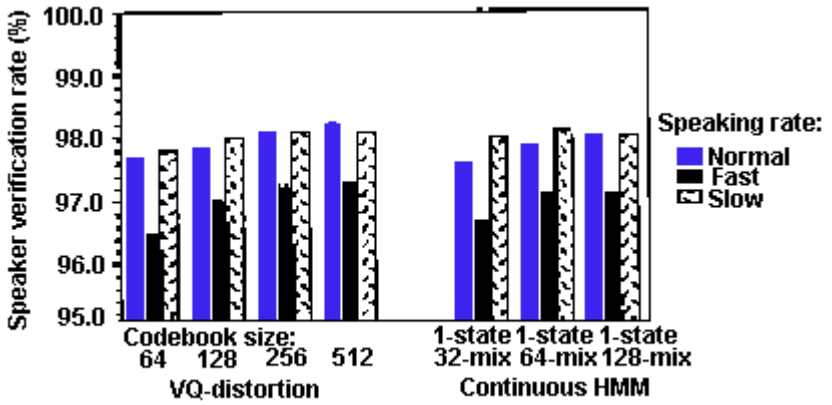
شکل شمارهی ۵ نتیجهی تجربیات این گروه را برای سیستمهای مستقل از متن نشان میدهد:

از این شکل این گونه بر میآید که روش مدل نهان مارکف با چگالی پیوسته نیازمند تعداد بردارهای آموزش سیستم بیشتری میباشد.



شکل شماره ۵ - درصد خطا بر اساس تعداد بردارهای آموزش سیستم برای روشهای مستقل از متن مقدارگزینی برداری ۳۲ عنصری و مدل نهان مارکف با چگالی پیوسته‌ی تک حالته‌ی ۳۲ ترکیبه

ماتسوی[۳۲] و فروی[۳۳] نیز سیستمهای مستقل از متن پیاده‌سازی شده با دو روش متأخر را مقایسه نمودند و اشاره نموده‌اند که روش مدل نهان مارکف ارگودیک پیوسته در مقابل تغییرات عبارت پایداری همسانی با روش مقدارگزینی برداری دارد و عملکرد بسیار بهتری نسبت به روش مدل نهان مارکف ارگودیک گسسته دارد. آنها همچنین به نتیجه‌ای مشابه با گروه قبلی دست یافته‌اند و آن این است که سیستمهای مبتنی بر روش مقدارگزینی برداری برای مقادیر کم داده پایدارتر از سیستمهای مبتنی بر روش مدل نهان مارکف پیوسته می‌باشند. شکل ۶ نتیجه‌ی تجربیات آنان را به تصویر می‌کشد:



شکل شماره ۶ – مقایسه‌ی سیستمهای مستقل از متن (ماتسوی و فوروی ۱۹۹۲)

فصل هشتم - کتابخانه‌ی تشخیص گوینده

۱- کتابخانه‌ی تشخیص صحبت – گوینده‌ی جیالانگ هی[۱]

کتابخانه‌ی مزبور که از طریق مرجع شماره‌ی ۱ قابل دستیابی است به زبان C++ نوشته شده و شامل حدود ۲۰ کلاس است که الگوریتمهای رایج استخراج خصیصه‌ها و تکنیکهای رایج برای تشخیص صحبت و تأیید هویت گوینده را پیاده‌سازی می‌نماید.

محتویات این کتابخانه امکان پردازش صحبت به صورت بدون سرباره[۲] و با سرباره در فرمتی خاص را به کاربر می‌دهد و امکانات مختلفی را برای کار با فایلهای حاصل که حاصل اعمال یک الگوریتم به صورت یک الگو و یا یک مدل بر داده‌های ورودی است فراهم می‌آورد.

در بحث مدلسازی سیگنال استخراج خصیصه‌ها خصیصه‌های مبتنی بر الگوریتم فوریه‌ی سریع با اندازه‌ی مل[۳]، خصیصه‌های به دست آمده از روش پیشگویانه‌ی خطی[۴]، خصایص پویای مبتنی (دلتاها که در فصل ششم توضیح داده شد) را به همراه زیری و بمی[۵] صدا را با استفاده از این کتابخانه به دست آورد.

روشهای متداول مدلسازی صحبت شامل چندین روش مبتنی بر مدل نهان مارکف[۶]، مدل مقدارگزینی برداری[۷]، مدل گاوسی[۸] و مدلسازی به شیوه‌ی چشمپوشی زمانی پویا[۹] نیز در این کتابخانه وجود دارند.

این موارد در کنار امکانات اولیه‌ی پیاده‌سازی شده در این کتابخانه شامل تبدیل سریع فوریه و عکس آن، تبدیل کسینوسی گسسته‌ی فوریه، تحلیل پیشگویانه‌ی خطی، امکانات طراحی فیلتر، توابع پنجره‌ای (هنینگ، همینگ و ...) و چندین تابع ابتدایی دیگر این کتابخانه را یک نقطه‌ی آغاز مناسب برای طراحی برنامه‌های کامپیوتری تشخیص صحبت و یا گوینده نموده است.

با وجود آن که این کتابخانه به صورت رایگان قابل دستیابی است کد آن در دسترس قرار ندارد و این نکته باعث بروز مشکلاتی در تطبیق آن با انتخابهای برنامه‌نویسی کاربر می‌گردد. در ضمن با وجود یک سری مستندات ناقص و ارجاعهای متعدد منابع مرتبط با پردازش کامپیوتری صحبت به این کتابخانه مسأله‌ی درک چگونگی کارکرد و استفاده‌ی درست از این کتابخانه به طور کامل نیازمند پیش‌زمینه‌ی قوی در زمینه‌ی پردازش صحبت است و مستندات آن برای درک روش عملکرد آن کافی نیست.

در هر صورت ما از این کتابخانه برای مدلسازی سیگنال به شیوه‌ی استخراج خصیصه‌های مبتنی بر الگوریتم فوریه‌ی سریع با اندازه‌ی مل از این کتابخانه بهره جسته‌ایم و برای مقایسه‌ی الگوهای به دست آمده نیز از شیوه‌ی چشمپوشی زمانی پویای پیاده‌سازی شده در این کتابخانه استفاده کرده‌ایم که کارایی سیستم وابسته به متن به دست آمده می‌تواند گویای تواناییهای این کتابخانه باشد.

۲ـ پایگاه داده‌های عبارتهای عبور ـ شمای کلی[۱۰]

ما برای پردازش عبارتهای عبور ایجاد شده از یک ساختار پایگاه داده‌ها استفاده می‌کنیم که در آن الگوهای به دست آمده از عبارتهای عبور به همراه نام کاربران را ذخیره می‌نماییم. برای ساده‌تر شدن نحوه‌ی پردازش عبارتهای مختلف برای یک کاربر آنها را به صورت تکراری با نام یکسان ذخیره می‌کنیم و کار طبقه‌بندی آنها تحت نام یک کاربر را در هنگام بازیابی آنها انجام می‌دهیم.

برای نیل به هدف یاد شده ما کلاسی به نام HSpeaker برای نمایش هر عبارت گفته شده توسط کاربر خاص تعریف کرده‌ایم و وظیفه‌ی پردازش دسته‌ای از اشیاء از نوع این کلاس را به کلاسی به نام HSpeakersDB واگذار نموده‌ایم که البته این کلاس از یک کلاس و دو ساختار میانی برای پردازش اعمال مورد نظر استفاده می‌نماید. برنامه‌ی نهایی از کلاس HSpeakersDB برای پردازش نهایی استفاده می‌کند هر چند در بعضی موقعیتها ارجاعهای مستقیمی نیز به کلاس HSpeaker صورت پذیرفته است.

بخشهای بعدی ساختار درونی کلاسهای یاد شده را توضیح می‌دهد.

۳ـ عبارت عبور و پردازشهای مربوط به آن

هر عبارت عبور به همراه نام گوینده‌ی آن توسط شیئی از نوع کلاس HSpeaker قابل نمایش است:

```
class HSpeaker
{
public:
    HSpeaker();
    HSpeaker(SV_Model_DTW* pModel);
    ~HSpeaker();
    BOOL LoadFrom(CFile* pFile);
    BOOL SaveTo(CFile* pFile);
    void SetUserName(CString strName);
    CString GetUserName();
    void SetPassphrase(int nSamples, short* pSamples);
    double Verify(int nSamples, short* pSamples);
    double Verify(HSpeaker* pSpeaker);
    void PrepareFeature(SV_Feature_MFCC &Feature);
    void PrepareFeature(SV_Feature_Pitch &Feature);
```

```
    void PrepareFeature(SV_Feature_LPCC &Feature);
private:
    CString m_strName;
    SV_Data* m_pPassphrase;
    SV_Model_DTW* m_pModel;
};
```

کلاس SV_Data یکی از کلاسهای کتابخانه‌ی SVLib[۱۱] است که نمایشگر یک ماتریس می‌باشد. نتیجه‌ی پردازشها و الگوهای به دست آمده از آنها در نهایت به داده‌هایی از این نوع تبدیل می‌شوند. عضو داده‌ی m_pPassphrase الگوی به دست آمده یا ذخیره شده را که اعمال مقایسه‌ای روی آن صورت می‌پذیرد نشان می‌دهد. علت تعریف این عضو به صورت اشاره‌گر آن است که مقدار خروجی متدهای استخراج خصیصه اشاره‌گر به این کلاس است و در ضمن در خود این متدها فضای حافظه‌ی لازم به داده‌ی مورد نظر تخصیص داده می‌شود.

کلاس SV_Model_DTW عمل مقایسه‌ی دو الگو را که به صورت اشیائی از نوع SV_Data نمایش داده می‌شوند انجام می‌دهد و عضو داده‌ی مورد نظر برای چنین امری در نظر گرفته شده است. علت تعریف این عضو داده به صورت اشاره‌گر و اختصاص فضای حافظه بدون آزاد کردن آن مشکلی بود که در استفاده از این عضو داده در محیط برنامه‌نویسی غیر متنی به واسطه‌ی خطای مراجعه به فضای حافظه‌ی غیر مجاز به وجود می‌آمد که به نظر می‌رسد ناشی از تلاش برای آزاد کردن حافظه‌ای که قبلاً آزاد شده در متد ویرانگر این کلاس می‌باشد و به لحاظ عدم وجود کد کتابخانه به این صورت رفع شده است.[۱۲]

عضو داده‌ی m_strName نام گوینده‌ی عبارت را ذخیره می‌کند که توسط متدهای SetUserName و GetUserName اعمال مقدارگذاری و دریافت مقدار آن صورت می‌پذیرد:

```
void HSpeaker::SetUserName(CString strName)
{
    m_strName=strName;
}
CString HSpeaker::GetUserName()
{
    return m_strName;
}
```

کلاس سازنده دارای دوشکل متفاوت است. علت آن است که در عمل مقایسه لازم نیست هر دو کلاس دارای عضو m_pModel باشند و تخصیص حافظه به هر دو نوعی هدر دادن حافظه است لذا اگر برای یک شیء برای این عضو داده حافظه اختصاص داده شود دیگری می‌تواند از حافظه‌ی تخصیص داده شده‌ی دیگری استفده نماید:

```
HSpeaker::HSpeaker()
{
    m_pModel=new SV_Model_DTW;
    m_pPassphrase=NULL;
    m_strName="";
}
HSpeaker::HSpeaker(SV_Model_DTW* pModel)
{
    m_pModel=pModel;
    m_pPassphrase=NULL;
    m_strName="";
}
```

در کلاس ویرانگر اگر حافظه‌ای به m_pPassphrase اختصاص داده شده باشد آزاد می‌گردد:

```
HSpeaker::~HSpeaker()
{
    if(m_pPassphrase)
        delete m_pPassphrase;
}
```

قبل از هر گونه عمل مقایسه بایستی عبارت عبور الگویابی شود این عمل با دریافت داده‌های صوتی که به صورت یک آرایه از نوع short دریافت می‌شود و تبدیل آن به الگوی مورد نظر در متد SetPassphrase صورت می‌پذیرد:

```
void HSpeaker::SetPassphrase(int nSamples, short* pSamples)
{
    SV_Feature_MFCC Feature;
    PrepareFeature(Feature);
    Feature.CopySignal(pSamples, nSamples);
    m_pPassphrase=Feature.ExtractFeature();
}
```

کلاس SV_Feature_MFCC نیز مربوط به کتابخانه‌ی SVLib است و پردازشهای لازم برای استخراج الگوهای MFCC از سیگنال صوت در آن قرار داده شده است. قبل از فراخوانی متدهای آن می‌بایست انتخابهای لازم انجام شود و این عمل با استفاده از متد PrepareFeature صورت می‌پذیرد:

```
void HSpeaker::PrepareFeature(SV_Feature_MFCC &Feature)
{
    Feature.Para.MFCC_Order = 12;
    Feature.Para.NFilter    = 60;
    Feature.Para.FFTSz      = 512;
    Feature.Para.DEnergy    = 1;
    Feature.Para.WinSz = 512;
    Feature.Para.StpSz = 256;
    Feature.Para.Alpha = 0.97;
    Feature.Para.HammingWin = 1;
    Feature.Para.RmvSilence = 1;
}
```

در این متد انتخابهای مربوط به نوع پنجره‌بندی (همینگ یا پنجره‌بندی مستطیلی)، اندازه‌ی پنجره، اندازه‌ی بازه‌ی fft ، انتخاب این که آیا انرژی سیگنال هم به الگو ضمیمه شود یا نه، انتخاب این که آیا سکوت از سیگنال حذف شود یا نه و... با مقدارگذاری اعضای داده‌ی عمومی کلاس SV_Feature انجام می‌پذیرد.

پس از مقدارگذاری موارد یاد شده متد CopySignal برای ورود سیگنال به شیء استخراج کننده‌ی الگو فراخوانی می‌شود. از آنجا که این متد و سایر متدهای مشابه کلاس SV_Featue_MFCC (وکلاسهای مشابه برای استخراج خصیصه‌ها از روشهای دیگر) یک آرایه از نوع short را به عنوان سیگنال ورودی می‌پذیرند همچنان که در فصل چهارم اشاره شد اعضای داده‌ی کلاسهای پردازش صوت را از این گونه انتخاب نمودیم. لذا در این مرحله نیازی به هیچ گونه تغییر نداریم.

بعد از انجام تمامی مقدار گذاریها متد ExtractFeature الگوهای خواسته شده را استخراج نموده حاصل را به صورت اشاره‌گر بازمی‌گرداند. همچنان که قبلاً اشاره شد اختصاص حافظه به صورت درونی صورت می‌پذیرد و نیازی به انجام این کار پیش از فراخوانی این متد وجود ندارد.

بعد از استخراج الگوها یا فراخوانی یک متد ساده از کلاس SV_Model_DTW می‌توان عمل مقایسه را انجام داد و حاصل را که یک عدد اعشاری نشان دهنده‌ی میزان فاصله‌ی دو الگو از یک دیگر است به محل فراخوانی بازگرداند:

```
double HSpeaker::Verify(HSpeaker* pSpeaker)
{
    return m_pModel->DTW_Comp(*m_pPassphrase, *pSpeaker->m_pPassphrase);
}
```

شکل دیگر این متد برای انعطاف‌پذیری بیشتر نوشته شده است:

```
double HSpeaker::Verify(int nSamples, short* pSamples)
{
    HSpeaker Temp(m_pModel);
    Temp.SetPassphrase(nSamples, pSamples);
    return Verify(&Temp);
}
```

برای ذخیره‌ی شیء از نوع HSpeaker ابتدا نام کاربر ذخیره می‌گردد و سپس از یک کلاس دیگر به نام SV_DataIO از کتابخانه‌ی SVLib برای ذخیره‌ی موقت الگو در یک فایل و به دست آوردن اندازه‌ی آن بر حسب بایت و سپس ذخیره‌ی این اندازه و الگوی به دست آمده در فایل مقصد استفاده می‌گردد و در نهایت این فایل موقتی نیز حذف می‌گردد:

```
BOOL HSpeaker::SaveTo(CFile* pFile)
{
    BYTE bNameLength=(BYTE)m_strName.GetLength();
    pFile->Write(&bNameLength,sizeof(BYTE));
    for(int bc=0;bc<bNameLength;bc++)
    {
        char c=m_strName[bc];
        pFile->Write(&c,sizeof(char));
    }
    SV_DataIO Temp;
    Temp.OpenFile("temp.sv",WRITE_REC);
    Temp.PutDataRec(*m_pPassphrase);
    Temp.CloseFile();
    CFile TempFile("temp.sv", CFile::modeRead);
    BYTE byte;
    int iSize=0;
    while(TempFile.Read(&byte, sizeof(byte))==sizeof(byte))
            iSize++;
    TempFile.Close();
    pFile->Write(&iSize, sizeof(int));
    TempFile.Open("temp.sv", CFile::modeRead);
    int i=0;
    while(i<iSize)
    {
        TempFile.Read(&byte, sizeof(byte));
        pFile->Write(&byte, sizeof(byte));
        i++;
```

```
    }
    TempFile.Close();
    DeleteFile("temp.sv");
    return TRUE;
}
```

برای بازیابی نیز ابتدا نام کاربر، سپس اندازه‌ی الگو و در نهایت خود الگو با استفاده از یک فایل موقتی بازیابی می‌گردد:

```
BOOL HSpeaker::LoadFrom(CFile* pFile)
{
    BYTE bNameLength;
    if(pFile->Read(&bNameLength, sizeof(BYTE)) != sizeof(BYTE))
            return FALSE;
    char* name=new char[bNameLength+1];
    pFile->Read(name,bNameLength*sizeof(char));
    name[bNameLength]='\0';
    m_strName=name;
    delete []name;
    int iSize;
    pFile->Read(&iSize, sizeof(int));
    Cfile TempFile("temp.sv", CFile::modeWrite|CFile::modeCreate);
    BYTE byte;
    for(int i=0; i<iSize; i++)
    {
            pFile->Read(&byte,sizeof(byte));
            TempFile.Write(&byte, sizeof(byte));
    }
    TempFile.Close();
    SV_DataIO Temp;
    Temp.OpenFile("temp.sv", READ_REC);
    m_pPassphrase=Temp.GetDataRec();
    Temp.CloseFile();
    DeleteFile("temp.sv");
    return TRUE;
}
```

برای فایلی که در آن ذخیره با بازیابی صورت می‌گیرد اعمال باز کردن و بستن انجام نمی‌شود زیرا نحوه‌ی استفاده از این متدها معمولاً شامل چندین فراخوانی متوالی است که یک بار بازکردن فایل و در نهایت یک بار بستن آن به ازای کلیه‌ی فراخوانیها کافی است.

۴- چند عبارت عبور برای یک کاربر و پردازشهای مربوط به آن

هر کاربر می‌تواند چندین کلمه‌ی عبور داشته باشد، برای پردازش این حالت ساختار زیر یک لیست پیوندی از کلمه‌ی عبورهای یک کاربر تشکیل می‌دهد:

```
struct SpeakerPasses
{
        SpeakerPasses(){pNext=NULL;}
        HSpeaker* pSpeaker;
```

```
        SpeakerPasses* pNext;
        double Verify(HSpeaker* pSpeaker);
};
```

متد Verify متد هم نام خود را متعلق به عضو داده‌ی pSpeaker فراخوانی می‌کند و مقدار بازگشتی آن را می‌گرداند و با فراخوانی مستقیم این متد چندان تفاوتی ندارد.

با استفاده از ساختار فوق و با استفاده از زنجیره‌ای از داده‌های از این نوع می‌توان لیست عبارات عبور مربوط به یک کاربر را نگهداری نمود.

ایجاد زنجیره‌ای از کاربران با استفاده از ساختار زیر صورت می‌گیرد:

```
struct SpeakersPasses
{
        CString strName;
        SpeakersPasses(){pNext=NULL;}
        SpeakerPasses* pFirst;
        SpeakersPasses* pNext;
        double Identify(HSpeaker* pSpeaker);
        double Find(HSpeaker* pSpeaker,int &Index);
};
```

هر عضو این زنجیره دارای نامی منحصر به فرد است که در عضو داده‌ی strName نگهداری می‌شود. یک اشاره‌گر به آغاز زنجیره‌ی عبارات عبور که با pFirst نشان داده می‌شود و یک اشاره‌گر به عضو بعدی در لیست کاربران که با pNext نشان داده می‌شود این ساختار را تشکیل می‌دهند.

متدهای این ساختار روشهایی برای مقایسه‌ی اجزای داخلی فراهم می‌آورد. متد Identify به سادگی مینیمم فاصله‌ی اجزای لیست با یک عبارت داده شده را به دست می‌دهد:

```
double SpeakersPasses::Identify(HSpeaker* pSpeaker)
{
        double fMin=10;
        double d;
        int i=0;
        for(SpeakerPasses* p=pFirst; p!=NULL; p=p->pNext,i++)
        if((d=p->Verify(pSpeaker))<fMin)
                fMin=d;
        return fMin;
}
```

این متد امکان مقایسه‌ی کلیه‌ی عبارات عبور تعریف شده به نام یک کاربر را با یک عبارت عبور فراهم می‌آورد.

در بخشهایی از کار ما می‌دانیم که یک عبارت متعلق به یک کاربر است و نیاز به این داریم که بدانیم دقیقاً کدام عبارت متعلق به کاربر است. مثلاً زمانی که بخواهیم یک عبارت پذیرفته شده از یک کاربر را از لیست عبارتهای مورد نظر او حذف کنیم به چنین امکانی نیاز داریم. متد زیر چنین امکانی را فراهم می‌آورد:

```
double SpeakersPasses::Find(HSpeaker* pSpeaker, int &Index)
{
        double fMin=10;
```

```
        double d;
        int i=0;
        for(SpeakerPasses* p=pFirst; p!=NULL; p=p->pNext,i++)
                if((d=p->Verify(pSpeaker))<fMin)
                {
                        Index=i;
                        fMin=d;
                }
        return fMin;
}
```

کد متد مزبور کاملاً شبیه به متد Identify است و تنها یک پردازش اضافی در آن انجام می‌شود.

اگر به مقدار بازگشتی متدهایی که به نحوی با تأیید هویت یا بازشناسی هویت یک کاربر ربط دارند توجه شود مشخص می‌گردد که همواره یک مقدار اعشاری بازگردانده می‌شود و هیچگاه یک نتیجه‌ی منطقی (که یکسان هستند یا نه) بر نمی‌گردد. این به دلیل آن است که برنامه قابلیت آن را دارد که طبق خواسته‌ی کاربر آستانه‌ی مورد استفاده برای یکسان بودن عبارات عبور را تغییر می‌دهد (که ممکن است باعث کاهش درصد خطای برنامه و در نتیجه‌ی افزایش امنیت یا کاهش میزان سختگیری برنامه و در نتیجه کاهش تعداد تلاشهای لازم برای گرفتن جواب شود). از این رو عمل مقایسه با آستانه در کلاسهای کنترل کننده‌ی تشخیص کاربر انجام می‌گیرد.

۵- ایجاد یک جدول از کاربران

در مرحله‌ی بعد نیاز به آن داریم که کلیه‌ی پردازشهای اشاره شده برای کاربران منفرد را در کنار هم قرار دهیم و به مجموعه‌ی کاربران به عنوان یک کل نگاه کنیم.

به این منظور کلاسی به نام HSpeakersTable با ساختار زیر طراحی شد:

```
class HSpeakersTable
{
public:
        HSpeakersTable(){m_pFirst=NULL;}
        void Insert(HSpeaker* pSpeaker);
        HSpeaker* Get(int iMain, int iSub=0);
        int GetSubsNum(int iMain);
        int GetIndex(CString str);
        int GetNum();
        void RemoveUser(int nIndex);
        BOOL RemPass(int nIndex, HSpeaker* pSpeaker, double fThreshold);
        int Identify(HSpeaker* pSpeaker, double fThreshold);
private:
        SpeakersPasses* m_pFirst;
};
```

تنها عضو داده‌ی این کلاس اشاره‌گر به آغاز لیست کاربران است که در حالتی که لیست خالی است مقدار آن برابر با NULL خواهد بود.

متدهای بعدی اعمال کار با لیست کاربران را پیاده‌سازی می‌نمایند. عملی که متد Insert انجام می‌دهد شامل مقایسه‌ی نام کاربران با نام وارد شده برای عبارت مورد نظر است. در صورتی که لیست تهی باشد مقایسه‌ای انجام نمی‌شود و تنها نود ابتدایی ایجاد می‌گردد:

```
void HSpeakersTable::Insert(HSpeaker* pSpeaker)
{
if(m_pFirst==NULL)
{
    m_pFirst=new SpeakersPasses;
    m_pFirst->strName=pSpeaker->GetUserName();
    m_pFirst->pFirst=new SpeakerPasses;
    m_pFirst->pFirst->pSpeaker=pSpeaker;
    return;
}
else
{
    int index;
        if( (index=GetIndex(pSpeaker->GetUserName())) !=  -1)
     {
        int i=0;
        for(SpeakersPasses* p=m_pFirst;  i<index; p=p->pNext,i++);
        for(SpeakerPasses* q=p->pFirst; q->pNext!=NULL; q=q->pNext);
        q->pNext=new SpeakerPasses;
        q->pNext->pSpeaker=pSpeaker;
    }
    else
        {
        for(SpeakersPasses* p=m_pFirst; p->pNext!=NULL; p=p->pNext);
        p->pNext=new SpeakersPasses;
        p->pNext->strName=pSpeaker->GetUserName();
        p->pNext->pFirst=new SpeakerPasses;
        p->pNext->pFirst->pSpeaker=pSpeaker;
    }
}
}
```

متد GetIndex که در متد قبلی فراخوانی شده است یک عمل مقایسه‌ی ساده را انجام می‌دهد و در صورت یافتن نام داده شده در لیست نام کاربران اندیس کاربر مورد نظر را برمی‌گرداند و در غیر این صورت مقدار ۱ - در خروجی این تابع نشانگر آن است که کاربر مورد نظر پیدا نشده و باید یک نود جدید ایجاد گردد:

```
int HSpeakersTable::GetIndex(CString str)
{
  int index=0;
  for(SpeakersPasses* p=m_pFirst; p!=NULL; p=p->pNext,index++)
      {
       if(str==p->strName)
            return index;
      }
       return -1;
}
```

تعداد کاربران را متد زیر با یک سری پردازش ساده روی لیست به دست می‌آید (متد GetNum) . اگر لازم داشته باشیم تعداد عبارات عبور یک کاربر را بدانیم از متد GetSubsNum کمک می‌گیریم و اگر بخواهیم

با یک عبارت به طور مستقیم کار کنیم از متد Get کمک می‌گیریم.کد این متدها که یک سری پردازش ساده روی لیست انجام می‌دهند در ادامه آمده است.

```
int HSpeakersTable::GetNum()
{
	if(m_pFirst==NULL)
		return 0;
	int Num=1;
	for(SpeakersPasses* p=m_pFirst; p->pNext!=NULL; p=p->pNext,Num++);
	return Num;
}
int HSpeakersTable::GetSubsNum(int iMain)
{
	if(m_pFirst==NULL)
		return 0;
	int index=0;
	for(SpeakersPasses* p=m_pFirst; index<iMain; p=p->pNext,index++);
	index=1;
	for(SpeakerPasses* q=p->pFirst; q->pNext!=NULL; q=q->pNext,index++);
	return index;
}
HSpeaker* HSpeakersTable::Get(int iMain, int iSub)
{
	int index=0;
	for(SpeakersPasses* p=m_pFirst; index<iMain; p=p->pNext,index++);
	index=0;
	for(SpeakerPasses* q=p->pFirst; index<iSub; q=q->pNext,index++);
	return q->pSpeaker;
}
```

عمل حذف یک کاربر که اندیس آن را در آرایه‌ی لیست کاربران می‌دانیم نیز یک عمل ساده‌ی پردازش لیست است:

```
void HSpeakersTable::RemoveUser(int nIndex)
{
	if(nIndex==0)
	{
		SpeakersPasses *p=m_pFirst;
		m_pFirst=m_pFirst->pNext;
		SpeakerPasses* q=p->pFirst;
		for(SpeakerPasses* r=q->pNext; r!=NULL; r=r->pNext)
		{
			delete q;
			q=r;
		}
		delete p;
		return;
	}
	int i=1;
	for(SpeakersPasses* p=m_pFirst; i<nIndex; p=p->pNext,i++);
```

```
        SpeakersPasses *pp=p->pNext;
        p->pNext=pp->pNext;
        SpeakerPasses* q=pp->pFirst;
        for(SpeakerPasses* r=q->pNext; r!=NULL; r=r->pNext)
        {
                delete q;
                q=r;
        }
        delete pp;
}
```

متد Identify عمل بازشناسی کاربر را انجام می‌دهد. برای بازشناسی یک کاربر عبارت عبور داده شده به کمک متد Verify ساختار SpeakersPasses با تمامی لیست کاربران مقایسه می‌شود و مقدار تفاوت نزدیک‌ترین کاربر با مقدار آستانه‌ی داده شده مقایسه می‌گردد و در صورتی که از آن میزان کمتر باشد اندیس کاربر و در غیر این صورت مقدار ۱- به محل فراخوانی بازگردانده می‌شود:

```
int HSpeakersTable::Identify(HSpeaker* pSpeaker, double fThreshold)
{
        double fMin=10,d;
        int index;
        int i=0;
        for(SpeakersPasses* p=m_pFirst; p!=NULL; p=p->pNext,i++)
        {
                if((d=p->Identify(pSpeaker))<fMin)
                {
                        fMin=d;
                        index=i;
                }
        }
        if(fMin<=fThreshold)
                return index;
        return -1;
}
```

متد RemPass متدی است که به وسیله‌ی آن برنامه‌نویس می‌تواند کاربری را که دارای عبارت عبور داده شده است حذف کند. در صورتی که عمل موفقیت‌آمیز باشد یعنی کاربری با عبارت داده شده یافته شود و حذف گردد مقدار منطقی TRUE و در غیر این صورت مقدار FALSE به محل فراخوانی بازگردانده می‌شود. ملاحظه‌ی این که عبارت مزبور از ابتدای لیست حذف می‌شود یا نه از پردازشهای این متد است. همچنین فرض بر این است که هیچگاه تقاضای حذف عبارت عبور کاربری که تنها یک عبارت عبور دارد نخواهد شد. این متد برای یافتن عبارت مزبور از متد Find ساختار SpeakersPasses سود می‌جوید:

```
BOOL HSpeakersTable::RemPass(int nIndex, HSpeaker* pSpeaker, double
fThreshold)
{
        int i=0;
        for(SpeakersPasses* p=m_pFirst; i<nIndex; p=p->pNext,i++);
        int Index=-1;
        if(p->Find(pSpeaker,Index)<=fThreshold)
        {
                if(Index==0)
```

```
            {
                SpeakerPasses *t=p->pFirst;
                p->pFirst=t->pNext;
                delete t;
            }
            else
            {
                int i=0;
                for(SpeakerPasses *q=p->pFirst; i<Index-1; q=q->pNext,i++);
                SpeakerPasses *t=q->pNext;
                q->pNext=t->pNext;
                delete t;
            }
            return TRUE;
        }
        return FALSE;
}
```

۶ـ تشکیل و کار با پایگاه داده‌ها

پایگاه داده‌های کاربران معمولاً تعداد عنصرهای محدودی دارد و پردازشهای آن نیز مختص به خود است. لذا استفاده از یک پایگاه داده‌های تخت مبتنی بر یک فایل مناسب به نظر می‌رسد. مزید بر این انتخابهای کاربر را نیز می‌توان در این ساختار ذخیره نمود.

اعمال کار با فایل که نهایت استفاده از این کتابخانه است در کلاس HSpeakersDB انجام می‌شود:

```
class HSpeakersDB
{
public:
        int GetUsersNum();
        void AddUser(HSpeaker* pSpeaker);
        CString GetUserName(int index);
        int GetPassesNum(int index);
        void RemoveUser(int nIndex);
        int Identify(HSpeaker* pSpeaker);
        BOOL RemPass(int index, HSpeaker* pSpeaker);
        HSpeakersDB(BOOL bAutoLoad=FALSE);
        void LoadUsers();
        void StoreUsers();
private:
        void CreateUsersDB();
private:
        HSpeakersTable* m_pTable;
        CString m_strDBFileName;
private:        //user identification options:
        int m_nRepetitions;
        float m_fMinPassLength;
        double m_fThreshold;
public:
        void SetRepsNum(int iNum);
```

```
        int GetRepsNum();
        void SetMinPassLen(float fMin);
        float GetMinPassLen();
        void SetSecurityLevel(int nLevel);
        int GetSecurityLevel();
};
```

هفت متد ابتدایی تنها متدهای متناظر خود در عضو داده‌ی m_pTable را فراخوانی می‌کنند و دو متد آخر مقدار پارامتر fThreshold در متدهای متناظر را با عضو داده‌ی m_fThreshold که توسط متد SetSecurityLevel مقدار گذاری می‌شود جایگزین می‌کنند:

```
int HSpeakersDB::GetUsersNum()
{
        return m_pTable->GetNum();
}
void HSpeakersDB::AddUser(HSpeaker *pSpeaker)
{
        m_pTable->Insert(pSpeaker);
}
CString HSpeakersDB::GetUserName(int index)
{
        return m_pTable->Get(index)->GetUserName();
}
int HSpeakersDB::GetPassesNum(int index)
{
        return m_pTable->GetSubsNum(index);
}
void HSpeakersDB::RemoveUser(int nIndex)
{
        m_pTable->RemoveUser(nIndex);
}
int HSpeakersDB::Identify(HSpeaker* pSpeaker)
{
        return m_pTable->Identify(pSpeaker, m_fThreshold);
}
BOOL HSpeakersDB::RemPass(int index, HSpeaker* pSpeaker)
{
        return m_pTable->RemPass(index, Speaker,m_fThreshold);
}
```

متد SetSecurityLevel مقدار m_fThreshold را با انتخاب آن از یک لیست از مقادیر به دست آمده از تجربه مقدارگزینی می‌نماید و متد GetSecurityLevel عکس این عمل را انجام می‌دهد:

```
void HSpeakersDB::SetSecurityLevel(int nLevel)
{
        switch(nLevel)
        {
        case VERYHIGH:
                m_fThreshold=0.3;
                break;
```

```
        case HIGH:
                m_fThreshold=0.5;
                break;
        case MEDIUM:
                m_fThreshold=0.7;
                break;
        case LOW:
                m_fThreshold=0.9;
                break;
        case VERYLOW:
                m_fThreshold=1.1;
                break;
        }
}
int HSpeakersDB::GetSecurityLevel()
{
        int nLevel;
        switch(int(m_fThreshold*10))
        {
        case 3:
                nLevel=VERYHIGH;
                break;
        case 5:
                nLevel=HIGH;
                break;
        case 7:
                nLevel=MEDIUM;
                break;
        case 9:
                nLevel=LOW;
                break;
        case 11:
                nLevel=VERYLOW;
                break;
        }
        return nLevel;
}
```

اعضای داده‌ی m_nRepetitions و m_fMinPassLength در هیچکدام از متدهای کلاس (به غیر از متدهای مقدارگذار و بازگرداننده‌ی مقدار آنها) کاربرد عملی ندارند و تنها برای آن که در متدهای ذخیره و بازیابی در پایگاه داده‌های عبارات رمز ذخیره و یا از آن بازیابی شوند عضو این کلاس هستند.

متد StoreUsers کاربران و انتخابهای مربوط به آنها را درپایگاه داده‌ها ثبت می‌نماید:

```
void HSpeakersDB::StoreUsers()
{
  CFile UsersDB;
  if(UsersDB.Open("USERS.DB", CFile::modeWrite|CFile::modeCreate))
        {
                UsersDB.Write(&m_nRepetitions, sizeof(m_nRepetitions));
```

```
        UsersDB.Write(&m_fMinPassLength, sizeof(m_fMinPassLength));
        UsersDB.Write(&m_fThreshold, sizeof(m_fThreshold));
                int iNum=m_pTable->GetNum();
                for(int i=0; i<iNum; i++)
                {
                        int iSubs=m_pTable->GetSubsNum(i);
                        for(int j=0; j<iSubs; j++)
                                m_pTable->Get(i,j)->SaveTo(&UsersDB);
                }
                UsersDB.Close();
        }
}
```

متد LoadUsrers کاربران ذخیره شده را بارگذاری می‌کند و در صورت عدم وجود، آن را به وجود می‌آورد:

```
void HSpeakersDB::LoadUsers()
{
        CFile UsersDB;
        m_nRepetitions=2;
        m_fMinPassLength=2.0;
        m_fThreshold=0.7;
        if(UsersDB.Open("USERS.DB",CFile::modeRead))
        {
                if(UsersDB.Read(&m_nRepetitions,
sizeof(m_nRepetitions))==sizeof(m_nRepetitions))
        if(UsersDB.Read(&m_fMinPassLength,
sizeof(m_fMinPassLength))==sizeof(m_fMinPassLength))
                UsersDB.Read(&m_fThreshold, sizeof(m_fThreshold));
                BOOL Finished=FALSE;
                while(!Finished)
                {
                        HSpeaker* pSpeaker=new HSpeaker;
                        if(pSpeaker->LoadFrom(&UsersDB))
                                m_pTable->Insert(pSpeaker);
                        else
                        {
                                delete pSpeaker;
                                Finished=TRUE;
                        }
                }
        }
        else
                CreateUsersDB();
}
```

متد CreateUsersDB فقط یک فایل خالی ایجاد می‌کند. انتخابها بعداً در این فایل نوشته خواهند شد.

مجموعه کلاسهای فوق یک کتابخانه‌ی ایستا به نام HSpeakersDBLib را تشکیل می‌دهند که می‌توان با استفاده از آن یک سیستم تشخیص گوینده‌ی وابسته به متن ساده را پیاده‌سازی نمود. همچنانکه اشاره شد

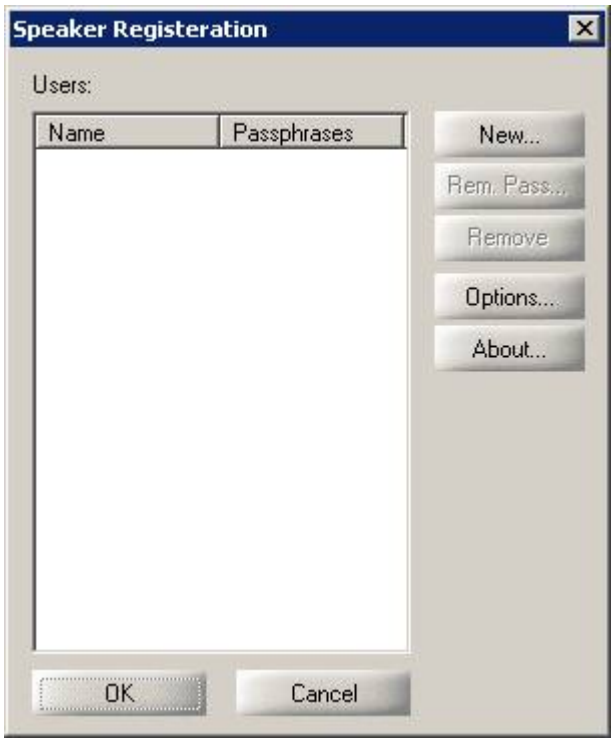
کتابخانه‌ی اصلی مورد استفاده تواناییهای گسترده‌ای برای کار با انواع روشها و الگوریتمها دارد که می‌توان با کار روی آنها سیستمهای تشخیص گوینده با تشخیص صحبت با کارایی عملی ایجاد نمود.

فرضیات ما در نحوه‌ی استفاده از این کلاس نحوه‌ی پیاده‌سازی این کلاس مؤثر بوده و برای به وجود آوردن یک مجموعه‌ی دارای کاربرد کلی‌تر باید بیشتر روی این کتابخانه کار شود اما به نظر می‌رسد همچنان که از عملکرد برنامه‌ی پیاده‌سازی شده قابل مشاهده است حتی این استفاده‌ی ساده از این کتابخانه نیز می‌تواند برطرف کننده‌ی نیازهای یک برنامه‌نویس در زمینه‌ی طراحی یک سیستم تشخیص گوینده می‌باشد.

ضمیمه شماره ۱ - نحوه‌ی استفاده از برنامه‌ی SpeakerID

۱- پروژه‌ی SpeakerID از دو فایل اجرایی اصلی به نامهای UsrMngr.exe و SpeakerID.exe تشکیل شده است که در صورت اجرای هر کدام برای اولین بار فایل پایگاه داده‌های پروژه که USERS.DB نام دارد در دایرکتوری جاری ایجاد می‌گردد. توجه گردد که دو فایل اجرایی مزبور باید در یک دایرکتوری قرار داشته باشند تا فایل پایگاه داده‌های آنها یکسان گردد.

۲- برنامه‌ی UsrMngr.exe کاربران جدید را ثبت می‌کند. با اولین اجرای آن جعبه‌ی گفتگویی مانند شکل زیر در صفحه ظاهر خواهد شد:



شکل شماره ۱ - جعبه‌ی گفتگوی آغازین برنامه‌ی ثبت کاربر

با کلیک بر روی دکمه‌ی New می‌توان نام کاربر جدیدی را وارد نمود و یا یک نام کاربر را برای اضافه کردن یک عبارت عبور جدید انتخاب نمود. همچنان که در متن توضیح داده شده است به نظر می‌رسد تن

صدای افراد در ساعات مختلف روز متفاوت است. از این لحاظ امکان ضبط عبارات عبور متفاوت برای یک کاربر می‌تواند مشکلاتی که ممکن است از این لحاظ برای کاربران به وجود آید از بین ببرد:

شکل شماره ۲ - ورود نام کاربر

در مرحله‌ی بعد با استفاده از دکمه‌های موجود می‌توان یک عبارت عبور جدید ضبط نمود. یک دکمه نیز برای انتخاب ورودی و تنظیم صدای میکروفن گذاشته شده است:

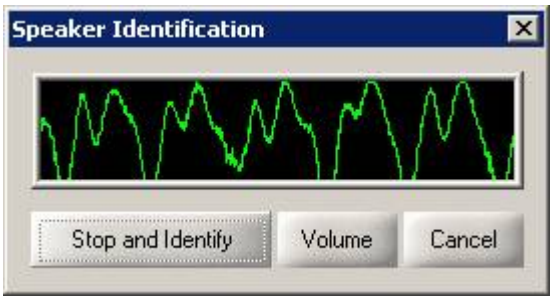
شکل شماره ۳ - ضبط یک عبارت عبور جدید

در صورتی که طول عبارت عبور از مقداری که در انتخابهای برنامه انتخاب شده کمتر نباشد با فعال شدن دکمه‌ی Next می‌توانید به مرحله‌ی بعد بروید. در مرحله یا مراحل بعد (که تعداد آنها توسط کاربر قابل انتخاب است) از کاربر خواسته می‌شود تا عبارت عبور خود را تکرار کند. در صورت تطابق عبارت عبور با اولین عبارت عبور می‌توانید مراحل بعد را که مشابه مرحله‌ی اول است پشت سر بگذارید و نهایتاً کاربر مورد نظر خود را ثبت نمایید:

شکل شماره ۴ - پس از طی مراحل توضیح داده شده کاربر مورد نظر ثبت می‌شود.

با کلیک بر روی دکمه‌ی OK کاربر مزبور در فایل پایگاه‌داده‌های کاربران ذخیره می‌گردد.

۳- با اجرای برنامه‌ی SpeakerID.exe می‌تواند کاربر ثبت شده را بازشناسی نمود:

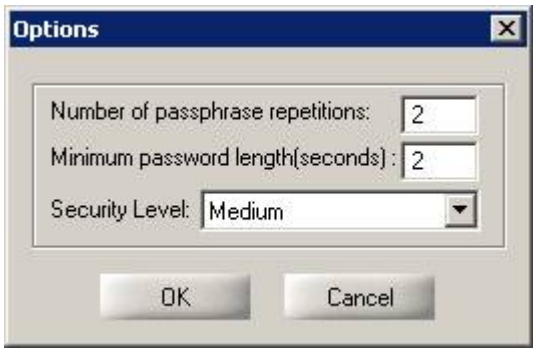


شکل شماره ۵ – برنامه‌ی تشخیص گوینده‌ی ثبت‌شده

در صورت بازشناسی کاربر برنامه‌ی یاد شده نام او را نشان خواهد داد و از برنامه خارج خواهد شد و گرنه لازم است بار دیگر عبارت مورد نظر گفته شود.

۴- برای حذف یک کاربر کافی است در برنامه‌ی UsrMngr.exe نام آن کاربر را از لیست کاربران انتخاب نموده بر روی دکمه‌ی Remove کلیک کنید. امکان حذف یک عبارت عبور خاص نیز با انتخاب کاربر مورد نظر و کلیک روی دکمه‌ی Rem.Pass... و سپس تکرار عبارت عبور مورد نظر وجود دارد. توجه شود که هر کاربر باید حداقل یک عبارت عبور داشته باشد و به همین دلیل برای کاربرانی که فقط یک عبارت عبور دارند این دکمه فعال نمی‌گردد.

۵- برنامه‌ی UsrMngr.exe به وسیله‌ی دکمه‌ی Options امکان تغییر بعضی پیش‌فرض‌های سیستم را در اختیار می‌گذارد که این تغییرات در فایل پایگاه‌داده‌های سیستم ذخیره می‌شوند و با حذف آن فایل از دست خواهند رفت:



شکل شماره ۶ – تغییر پیش‌فرض‌های سیستم

همچنان که در شکل نشان داده شده می‌توان تعداد تکرارهای لازم برای پذیرش عبارت عبور را تعیین نمود. حداقل این تعداد دو بار می‌باشد (یکی برای اولین بار و یکی تکرار آن). حداقل طول عبارت عبور نیز قابل تغییر است. در مستندات مختلف مربوط به سیستمهای تشخیص گوینده اشاره شده است که هر چه طول عبارت عبور بیشتر باشد میزان خطای سیستم کاهش می‌یابد. همچنان که قبلاً نیز اشاره شد اگر طول عبارت عبور گفته شده از حداقل تعیین شده کمتر باشد سیستم آن را نمی‌پذیرد. عبارات عبور قبلاً ثبت‌شده که ممکن است طولشان کمتر از میزان تعیین شده‌ی جدید باشد باز هم مجاز خواهند بود و آنها را باید به صورت دستی حذف نمود. در قسمت آخر می‌توان میزان سختگیری سیستم را تغییر داد. ممکن است در شرایط متفاوت (میکروفن و شرایط محیطی مختلف) نیاز به چنین تغییری پیدا شود. در محیط آزمایشی ما میزان تعیین شده برای Medium مناسب به نظر می‌رسید.

ضمیمه شماره ۲ - فهرستی از منابع برای مطالعه‌ی بیشتر