UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ECATEPEC

TITULO DEL TRABAJO:

DESARROLLO DE UN ALGORITMO DE RECONOCIMIENTO DE COMANDOS DE VOZ PARA LA MANIPULACIÒN DE UN ACTUADOR ELÉCTRICO.

T E S I S

QUE PARA OBTENER EL TITULO DE INGENIERO EN COMPUTACION

P R E S E N T A

C. OMAR VÁZQUEZ CASTRO

ASESOR: Dr. En I. de S. TERESA IVONNE CONTRERAS TROYA

REVISOR: M. RAÚL SALINAS RODRÍGUEZ

REVISOR: M. OSDALI BOJÓRQUEZ ISLAS

ECATEPEC DE MORELOS ESTADO DE MÉXICO

OCTUBRE 2018

INDICE

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

Cada vez es más común para el ser humano encontrarse inmiscuido en situaciones en las que deba interactuar con algún tipo de máquina o herramienta para lograr la consecución de ciertas metas u objetivos. Desde la antigüedad el hombre ha buscado que la interacción hombre-máquina se lleve de manera natural, por lo cual se han desarrollado diversas investigaciones para que esa interacción se realice mediante los diversos métodos de comunicación utilizados por el ser humano, ya sean comandos de voz, estímulos dieléctricos, reconocimiento de gestos y señas, etc. (Fazzino & Sanchéz, 2008). Las industrias buscan constantemente automatizar procesos, con ayuda de maquinaria y tecnología reciente, lo cual demuestra una tendencia a mejorar procesos procurando que la menor cantidad de energía sea derrochada al momento de realizar las actividades planeadas.

El reconocimiento del habla proviene del deseo del ser humano por prescindir de accesorios (mouse, teclado) para interactuar con la máquina, la meta, tal y como la señalan Cancelo y Alonso (2007) en su libro *La tercera revolución,* es que sin duda en un futuro las personas tengan la capacidad de conversar con sus ordenadores y sistemas informáticos y, que a su vez, éstos sean capaces de asimilar dicha conversación. Esta comunicación se basa en la recepción de una señal emitida por un ser humano (acústica, fonética, fonológica, léxica, sintáctica, semántica y pragmática) por medio de una herramienta computacional, la cual, trata de obtener una interpretación aceptable de la información contenida dentro de dicho mensaje, con la finalidad de convertir dicha información en texto o acciones de cierto tipo. Tal sería el caso de desarrollar un algoritmo de reconocimiento de comandos de voz, para mejorar la manera en la que se manipulan los dispositivos electrónicos, como caso específico, un actuador eléctrico.

OBJETIVO

Objetivo general

Desarrollar un algoritmo capaz de reconocer comandos de voz para manipular un actuador eléctrico con ayuda de la plataforma NetBeans 8.1.

Objetivos particulares

1. Revisar las diversas herramientas para realizar reconocimiento de voz dentro de NetBeans 8.1.
2. Seleccionar las herramientas más adecuadas para realizar el algoritmo de reconocimiento de voz.
3. Desarrollar los módulos correspondientes para que el algoritmo de reconocimiento de comandos de voz desarrollado en NetBeans 8.1 funcione correctamente.

JUSTIFICACIÓN

El reconocimiento del habla acorde a lo que señala O´Shaughnessy (1987) se ha vuelto una herramienta muy poderosa e importante, y está presente en varios aspectos de la vida cotidiana del ser humano, puesto que tiene varias aplicaciones, por ejemplo, en el área de la seguridad la policía ocupa el reconocimiento del habla para comparar voces y localizar el o los lugares en los cuales fue emitido un mensaje, al mismo tiempo es posible hacer uso de esta tecnología para limpiar conversaciones, seleccionar palabras clave e incluso mensajes completos. De igual manera dentro de la medicina es posible habilitar los aparatos quirúrgicos (hablando de cirugías robotizadas) para que éstos sean capaces de responder ante comandos de voz por parte de un ordenador que ejecuta órdenes. Inclusive existe la posibilidad de generar síntesis de voz para las personas impedidas de alguna manera y que por dicho impedimento no tienen la capacidad de manipular un teclado o por algún motivo necesiten sintetizar su voz. Básicamente es posible adaptar cualquier tipo de interacción entre hombres y máquinas

para que los aparatos y herramientas respondan ante comandos de voz. (O´Shaughnessy, 1987)

Con base en lo anterior es que se ha considerado que desarrollar un algoritmo de reconocimiento de comandos de voz para la manipulación de un actuador eléctrico seria la base para posteriormente desarrollar algoritmos más complejos que ayuden a que la interacción hombre-máquina, y que dicho proceso se realice sin la necesidad de gadgets o accesorios.

El lenguaje para desarrollar el algoritmo es Java, si bien no es el único capaz de dicha tarea, es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible, y esto le da una gran ventaja sobre otros lenguajes.

Para realizar el reconocimiento de voz dentro de Java, se puede utilizar Java Speech API también conocida como JSAPI, la cual es una aplicación multiplataforma, que soporta comandos de reconocimiento, sistemas de dictado, inclusive los sintetizadores de voz.

JSAPI está basada en el manejo de eventos que son generados por el motor de voz y pueden ser identificados y manejados según sea requerido. Los eventos generados pueden ser manejados mediante la interfaz del motor de reconocimiento, inclusive a través del motor de síntesis.

El software de JSAPI fue seleccionado ya que éste realiza todo el reconocimiento de voz aplicando los modelos ocultos de Markov, lo cual lo hace bastante eficiente en esta tarea, por ser un software de uso libre para aplicaciones sin fines de lucro, y por la fácil integración que tiene con los proyectos de Java.

ANTECEDENTES

En 2005 García Galván desarrolló en la Universidad Autónoma Metropolitana un reconocedor de voz, el cual identifica fonemas y palabras pronunciadas en inglés por un locutor hispanoparlante, este reconocedor está basado en modelos ocultos de Markov entrenados independientemente del hablante. García Galván argumenta que la mayoría de los sistemas de reconocimiento de voz en inglés se realizaron con locutores cuya lengua nativa es este idioma, por lo cual la mayoría de los errores de reconocimiento se atribuyen a la variedad de acentos extranjeros, y que por esta razón es necesario adaptar los modelos existentes. Para este trabajo las técnicas de adaptación fueron implementadas a un conjunto de modelos ocultos de Markov entrenados independientemente del locutor, puesto que, según la autora, esto resulta mejor opción respecto al entrenamiento de modelos dependientes (García Galván, 2005).

Al mismo tiempo, Villamil Espinosa (2005) realizó un análisis de las aplicaciones en reconocimiento de voz al utilizar HTK (Hidden Markov Model Toolkit). En su análisis Villamil hace una comparación entre diversos métodos aplicables al reconocimiento de voz, pero se enfoca en la herramienta HTK puesto que ésta permite utilizar diversos métodos de entrenamiento de los Modelos ocultos de Markov. Al mismo tiempo Villamil ejemplifica el uso de HTK en dos aplicaciones: el reconocimiento de palabras aisladas y el reconocimiento de dígitos conectados. Según Villamil el objetivo de construir un reconocedor de voz es explotar el uso de las herramientas en el proceso (Villamil Espinoza, 2005).

Por otra parte, Llanque García desarrolló en 2009 para la Escuela Superior Politécnica de Litoral, un telemando electrónico activado por voz. Llanque García se inspira en las necesidades de personas con limitaciones físicas o motoras, obteniendo un dispositivo capaz de reemplazar el uso de las manos por la voz para hacer funcionar dispositivos electromecánicos. Se compararon comandos de voz emitidos por un locutor con palabras previamente almacenadas y con base en eso tomar un determinado curso de acción (Llanque García, 2009). Aunque este proyecto se pensó y enfocó en las

personas con algún tipo de limitante, también pretende ser de utilidad para personas sin limitaciones que busquen mayor comodidad dentro de su vida cotidiana.

Panta Martínez desarrolló en 2012 en la Universidad Politécnica de Valencia un sistema para el control domótico por voz. Su escrito documenta el proceso de análisis, diseño, implementación, y pruebas de un sistema para controlar un hogar mediante la voz. Dicho trabajo se basó en el supuesto de que el sistema domótico ya existía, el cual estaría formado por un servidor domótico y partes móviles distribuidas dentro de un hogar (Panta Martínez, 2012).

Mientras tanto Macas Macas y Padilla Pineda (2012) en su estudio de los modelos ocultos de Markov y el desarrollo de un prototipo para el reconocimiento del habla, hacen un análisis no sólo de los modelos ocultos de Markov, sino también de algunas herramientas útiles para aplicar dichos modelos al desarrollo antes planteado. Macas y Padilla profundizan en HTK, el cual es un conjunto de herramientas portátiles que sirven para manipular y construir modelos ocultos de Markov, aunque también pueden modelar cualquier tipo de serie temporal de datos. Este conjunto de herramientas fue creado por el departamento de Ingeniería de Cambridge para construir modelos basados en el procesamiento de señales del habla (Macas Macas & Padilla Pineda, 2012).

Por su lado, Pérez Badillo, Ponceros Martínez y Villalobos Ponce en 2013 realizaron para el Instituto Politécnico Nacional un sistema de seguridad que funciona mediante el reconocimiento de voz, en dicho sistema el locutor graba una palabra mediante un micrófono y ésta es comparada con una base de datos de palabras previamente almacenadas. Al mismo tiempo este proyecto se ayuda de una Interfaz gráfica que permite la interacción entre el usuario y el sistema de seguridad. El sistema de seguridad una vez ingresada la palabra la procesa por medio de algoritmos de predicción lineal. Este sistema de seguridad no sólo analiza los comandos emitidos por un locutor, también identifica al locutor según los parámetros de la onda que genera su voz (Pérez Badillo, Ponceros Martínez, & Villalobos Ponce, 2013).

Delgado Gallardo, Trujillo Castellanos y Valdez Simón desarrollaron en 2014 para el Instituto Politécnico Nacional un sistema de domótica controlado por voz para personas con deficiencias motrices. En el trabajo ellos abordaron la evolución de los hogares del ser humano y cómo es que éstos se adaptan a las tecnologías disponibles en la época. Al mismo tiempo explican el concepto de domótica y cómo es que ésta se integra con el reconocimiento de voz, puesto que la domótica integra muchas más funciones con el fin de introducir la tecnología y los elementos desarrollados dentro de los hogares para mejorar la calidad de vida de sus habitantes (Delgado Gallardo, Trujillo Castellanos, & Valdez Simón, 2014).

METODOLOGÍA DE LA INVESTIGACIÓN

1. Revisar la bibliografía de las diferentes herramientas capaces de realizar reconocimiento de voz dentro de NetBeans 8.1

Dentro de Java existen varias herramientas que se pueden utilizar, por mencionar algunas:

- JavaSpeech (JSAPI)
- CloudGarden JavaSpeech (CGJSAPI)
- Sphinx
- TalkingJava
- JavaTalk

2. Seleccionar la herramienta para realizar el algoritmo

El desarrollador debe considerar el API que se adapta de mejor manera al desarrollo del algoritmo de reconocimiento de comandos de voz.

3. Desarrollar el algoritmo seleccionado bajo la plataforma de NetBeans en su versión 8.1

En esta etapa se realiza la codificación y compilación del algoritmo desarrollado, así como la definición de las reglas gramaticales aplicables al mismo, es necesario tener instalado en un ordenador NetBeans 8.1, la versión de Java actual al momento de desarrollar la aplicación, y también la interfaz de programación de aplicaciones (API) seleccionada para desarrollar dicho software.

4. Analizar los diversos actuadores eléctricos

En esta etapa es necesario hacer una investigación de las características de los diversos actuadores eléctricos para generar una decisión fundamentada acerca de cuál es la opción más viable para el proyecto.

5. Seleccionar un actuador eléctrico

Una vez que se cuente con la información necesaria se deberá tomar la decisión acerca de qué actuador será el más conveniente para ser manipulado con el software desarrollado.

6. Adaptar el algoritmo para conectar aplicación y actuador eléctrico

Esta etapa incluye cierta codificación, ya que el actuador eléctrico se conecta a Java por medio de la placa Arduino, lo cual implica ciertas modificaciones al código Java desarrollado y la codificación en el lenguaje nativo de Arduino.

7. Realizar pruebas y analizar resultados

Al final del desarrollo es necesario hacer pruebas para corroborar que el software desarrollado es una opción viable para manipular de forma eficiente un actuador eléctrico, los resultados de esta prueba deberán ser cuantificados y analizados para su posterior exposición.

CAPÍTULO 1. MARCO TEÓRICO

En este capítulo se dan a conocer conceptos teóricos y fundamentos básicos para realizar el reconocimiento del habla con ayuda de un ordenador.

1. MARCO TEÓRICO

Dentro del reconocimiento del habla existen dos vertientes, la primera es el reconocimiento del locutor, el cual está enfocado en reconocer las características en la voz del emisor del mensaje, y cuyo objetivo es reconocer al locutor más que comprender dicho mensaje; por otra parte, se encuentra el reconocimiento de comandos de voz, el cual está diseñado para reconocer y diferenciar los mensajes emitidos sin tomar en cuenta a la persona que los produce.

El reconocimiento del habla proporciona a un ordenador la habilidad de escuchar las palabras emitidas por el locutor y determinar qué es lo que éste ha dicho.

Los principales pasos de un reconocedor de voz son los siguientes:

- Diseño de Gramática o Reglas gramaticales: Define las palabras que pueden ser reconocidas, inclusive reglas sintácticas para el lenguaje que podrá utilizar el locutor para interactuar con el ordenador.
- Procesamiento de señales: Analiza el espectro, es decir la frecuencia característica de la señal entrante de audio.
- Reconocimiento de fonemas: Compara los patrones del espectro con los patrones de los fonemas del idioma que se reconoce.
- Reconocimiento de palabras: Compara la secuencia de fonemas con las palabras y patrones de palabras especificados por las gramáticas activa.
- Emisión de resultados: Proporciona a la aplicación con información de las palabras que el reconocedor ha detectado en la señal de audio entrante.

1.1. Voz

Según Torres Gallardo y Gimeno Pérez (2008) en su libro *Anatomía de la voz* es el sonido producido por el aire espirado, que, después de una serie de modificaciones se convierte en palabras o canto. En términos más técnicos la voz es el sonido que se emite al pasar el aire entre las cuerdas vocales y hacerlas vibrar, este sonido puede ser agudo o grave dependiendo de la presión a la cual estén sometidas las cuerdas

vocales. Al mismo tiempo este sonido se amplía a su paso por las cavidades de resonancia, las cuales están conformadas por todas las estructuras situadas por encima de las cuerdas vocales. Los resonadores principales son la boca o cavidad bucal, en la cual el sonido se articula con ayuda de la lengua y los labios convirtiéndose en lenguaje, y la cavidad nasal en la cual resonarán los sonidos nasales. (Gimeno Perez & Torres Gallardo, 2008).

Para analizar la voz a través de una computadora, es necesario un proceso para digitalizar la voz de alguna manera, por lo regular este proceso se realiza con ayuda de un micrófono, ya sea externo o interno para que posteriormente la voz sea reconocida en la computadora como una señal, la cual se pueda manipular mediante algún tipo de proceso o plataforma de software.

Digitalización de la voz

Como se mencionó anteriormente la voz es un sonido, es decir, en pocas palabras, es un fenómeno analógico, una onda continúa en el tiempo surgida de las diferencias de presión del aire que se encuentra alrededor y que viaja a través de éste. Con un micrófono es posible generar una onda eléctrica análoga a estas diferencias de presión. Esta señal eléctrica analógica no puede ser almacenada directamente en un sistema digital; para ello, dicha señal debe digitalizarse, es decir, transformarla en una secuencia de números. (Perez, 2014)

En la digitalización de una señal analógica se requieren dos procesos. El primero es tomar muestras de la amplitud de dicha señal a intervalos regulares de tiempo, y el segundo es asignar a cada muestra un valor numérico proporcional. Se podría decir que la primera etapa se refiere al muestreo de la señal y la segunda a la cuantificación de la misma. (Perez, 2014)

La precisión en la digitalización depende de varios factores, entre ellos el nivel de ruido que se esté dispuesto a tolerar en la señal reconstruida, ya que la propia cuantificación

de la señal es fuente de ruido, un fenómeno conocido como ruido de cuantificación. (Perez, 2014)

1.2. Señales

Una señal es la magnitud eléctrica que transporta información útil, dentro de un sistema de comunicación (Sanchis, Torralba, Gonzalez, & Torres, 2005).

Dependiendo de la clasificación las señales pueden ser:

- **Señales analógicas y señales digitales.**

Éstas se refieren a toda magnitud eléctrica cuyas variaciones llevan información sobre un proceso o magnitud física. Las señales cuya amplitud varía de forma continua con respecto del tiempo se denominan analógicas (véase Figura #1), mientras que las que pueden tomar sólo una serie de valores concretos se denominan señales de amplitud discreta en el tiempo (véase Figura #2). Las señales que sólo pueden tomar valores de amplitud discreta en instantes concretos se denominan señales digitales, y su amplitud viene dada por un código que se representa mediante señales con sólo dos niveles de tensión. Por extensión suele denominarse como digitales a todas las señales de amplitud discreta, aunque sean continuas en el tiempo (Pallás Areny, 1993).

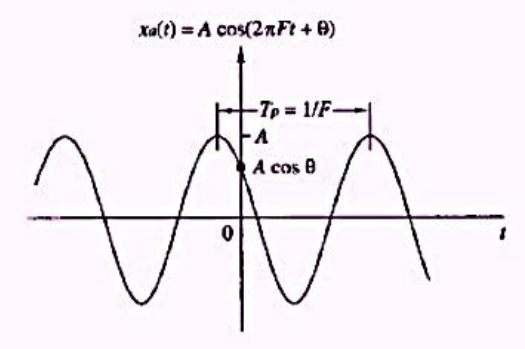

Figura # 1 Ejemplo de una señal sinusodal Analogica (Pallás Areny, 1993).

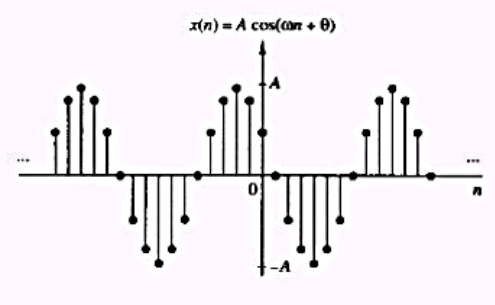

Figura # 2 Ejemplo de una señal sinusoidal discreta. (Pallás Areny, 1993).

- **Señales unipolares y señales diferenciales**

De acuerdo con la disposición física de los terminales en los que se presentan, las señales pueden ser unipolares o bipolares (véase Figura # 3).

Las señales unipolares se miden entre una terminal y otro de referencia. Se denominan señales unipolares puestas a tierra, a aquellas cuyo terminal de referencia está conectado a tierra (véase Figura # 4). Si entre el terminal de referencia existe una tensión, se dice de esta que es una tensión en modo común y no se puede conectar a tierra ninguno de los terminales de la señal (véase Figura # 4); la impedancia equivalente del generador puede tener valores muy dispares según el caso (Pallás Areny, 1993).

Las señales bipolares, diferenciales o polares, aparecen entre dos terminales que son independientes del terminal de referencia, que a su vez puede o no estar conectado a tierra (véase Figura # 4). La impedancia entre cada uno de los terminales de señal y el de tierra es similar. La polaridad con que se tome la señal es irrelevante: solo cambia el signo. Existen tres posibilidades: señal diferencial puesta a tierra, flotante o con tensión en modo común (véase Figura # 4). El punto de referencia para las señales flotantes, o con cualquiera de los dos tipos de terminales de señal puede conectarse a tierra; para las señales de tensión de modo común, no se puede conectar a tierra ningún terminal,

ni siquiera el de referencia. Se puede, sin embargo, invertir la polaridad de la salida (Pallás Areny, 1993).

Las señales diferenciales se distinguen porque las diferencias de potencial respectivas entre cada terminal y el de referencia varían simultáneamente en la misma magnitud, pero en sentido opuesto. Mientras una señal unipolar puede darse con dos terminales (alto A y bajo B), una señal diferencial necesita siempre de al menos tres terminales para su representación (alto A, bajo B y común C).

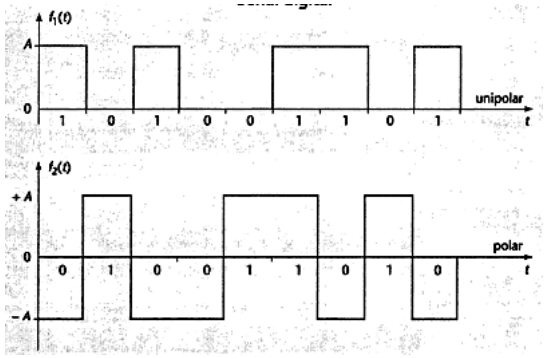

Figura # 3 Arriba ejemplo de una señal digital unipolar, y abajo ejemplo de una señal digital polar (Castro Lechtaler & Jorge Fusario, 1999).

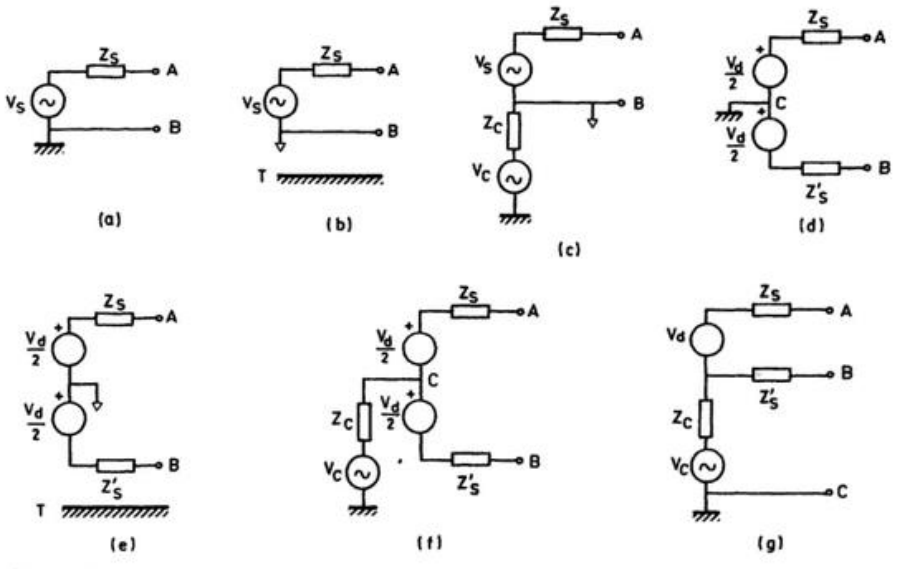

Figura # 4 Tipos de señales según sus terminales a) Unipolar puesta a tierra. b) Unipolar flotante. c) Unipolar con tensión en modo común d) Diferencial puesta a tierra. e) Diferencial flotante. f) Diferencial con tensión en modo común. g) Diferencial con tensión en modo común (Pallás Areny, 1993).

- **Señales de alta y baja impedancia**

Cuando se conectan dos elementos de un sistema electrónico existen dos posibilidades, que se desee que la tensión o corriente de entrada de un elemento coincida con la correspondiente a la salida del elemento precedente, o bien se desea transferir la máxima potencia de un elemento al siguiente. En ambos casos hay que adaptar la impedancia como se muestra en la Figura # 5 (Pallás Areny, 1993).

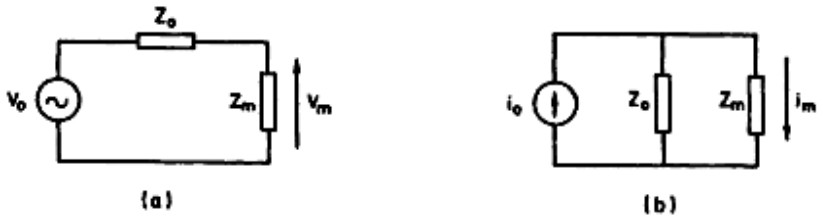

Figura # 5 5 Impedancia de salida y de entrada a) Cuando se mide tensión. b) Cuando se mide corriente (Pallás Areny, 1993).

Señal de audio

Con base en lo anterior se puede definir una señal de audio como: corriente eléctrica que representa al sonido dentro de un equipo electrónico ya sea por medio de una señal analógica o una digital (Gomez Gutierrez, 2009), en la Figura # 6 se muestra una señal de audio capturada en el dominio continuo.

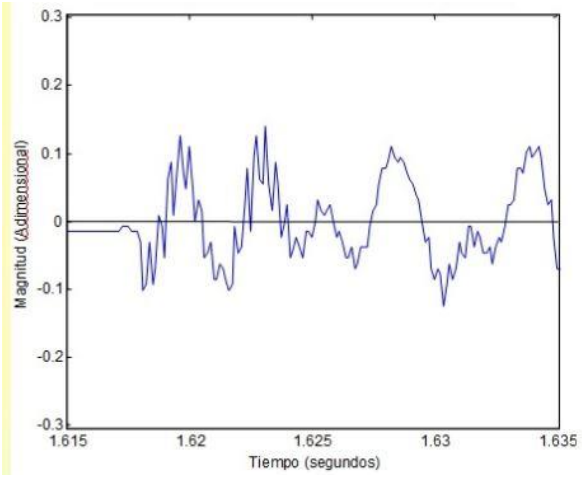

Figura # 6 Señal de audio capturada en el dominio continuo (Sanchez & Lemus, s.f.).

1.2.1. Procesamiento Digital de una Señal de Audio

Para que una señal pueda ser procesada digitalmente debe ser en tiempo discreto y tomar valores discretos tal y como se muestra en la figura # 7 (señal digital). Si la señal a procesar es analógica, se debe convertir a digital haciendo un muestreo en el tiempo y obteniendo por tanto una señal en tiempo discreto para posteriormente cuantificar sus valores en un conjunto discreto.

Existen tres enfoques diferentes para realizar el reconocimiento de voz con ayuda de un ordenador:

1. Enfoque Acústico - Fonético
2. Enfoque de reconocimiento de patrones
3. Enfoque de inteligencia artificial.

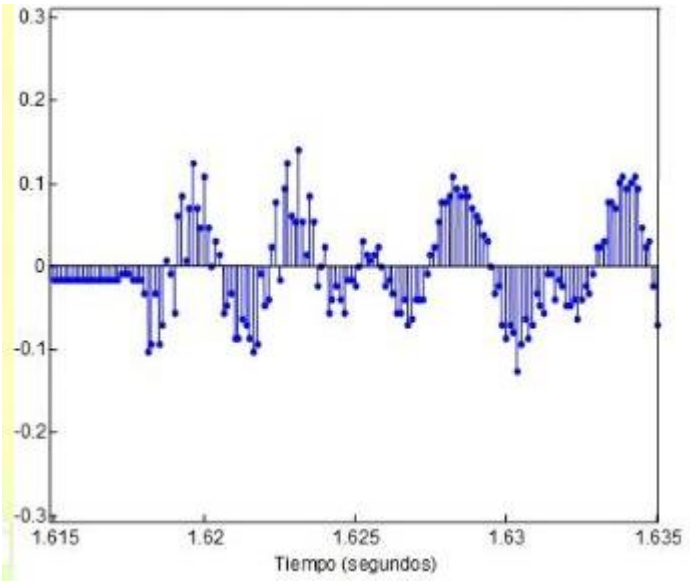

Figura # 7 Señal de audio capturada en el dominio discreto (Sanchez & Lemus, s.f.).

- **Enfoque acústico fonético**

Consiste en detectar sonidos elementales y asignarles determinados rótulos. La base de este enfoque es la hipótesis de que en el lenguaje hablado existe un número finito de unidades fonéticas distintas (fonemas) y que estas unidades pueden caracterizarse

por un conjunto de propiedades acústicas que se manifiestan en la señal hablada en función del tiempo. Si bien las propiedades acústicas de los fonemas son altamente variables con el locutor y con los fonemas vecinos (coarticulación de sonidos), se asume que las reglas que gobiernan la variabilidad son simples y pueden ser aprendidas fácilmente por el sistema de reconocimiento dicho sistema se muestra en la Figura # 8.

El reconocimiento consiste básicamente en dos pasos:

1. Segmentación y rotulado. La señal es dividida en regiones acústicas a las que son asignados uno o más fonemas, resultando en una caracterización de la señal de voz mediante un reticulado de fonemas.
2. Se trata de determinar una palabra (o conjunto de palabras) válida a partir de la secuencia de fonemas rotulados en el primer paso. Se introducen en esta etapa restricciones lingüísticas (vocabulario, sintaxis, y reglas semánticas).

La primera etapa en el procesamiento es la etapa de análisis de voz, que provee una representación (espectral) de las características de la señal de voz. Los métodos más comunes en esta etapa son análisis con banco de filtros y análisis LPC (Linear Predictive Coding). (Rabiner & Juang, 1993)

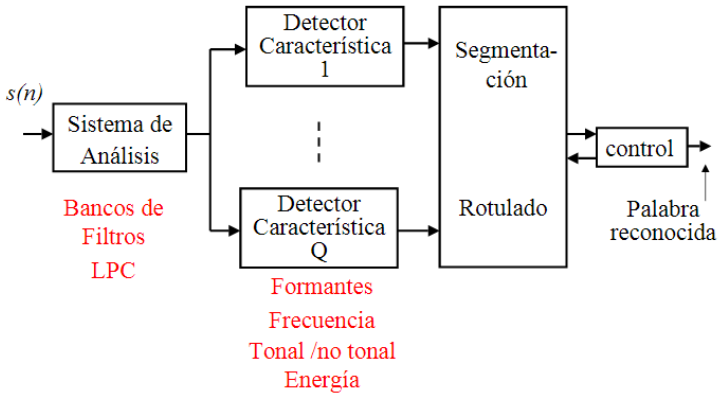

Figura # 8 Sistema de reconocimiento de voz basado en el enfoque acústico-fonético. (www.fceia.unr.edu.ar, s.f.)

La siguiente etapa es la extracción de características, en donde se convierten las medidas espectrales en un conjunto de parámetros que describen las propiedades acústicas de las unidades fonéticas. Estos parámetros pueden ser: nasalidad (presencia o ausencia de resonancia nasal), fricación (presencia o ausencia de excitación aleatoria en la voz), ubicación de los formantes (frecuencias de las tres primeras resonancias), clasificación entre sonidos tonales y no tonales, etc.

La tercera etapa del procesamiento es la etapa de segmentación y rotulado en donde el sistema trata de encontrar regiones estables donde las características cambian poco, que son rotuladas teniendo en cuenta cuan bien la característica en la región se ajusta a unidades fonéticas individuales. Ésta es usualmente la etapa más difícil de llevar a cabo en forma confiable.

El resultado de la etapa de segmentación y rotulado es un reticulado de fonemas a partir del cual se determina la palabra (o secuencia de palabras) que mejor se ajusta, teniendo en cuenta restricciones lingüísticas (de vocabulario, de sintaxis, y semánticas).

- **Enfoque de reconocimiento de patrones**

Este consiste básicamente en dos pasos:

1. Entrenamiento de patrones
2. Comparación de patrones

La característica principal de este enfoque es que usa un marco matemático bien definido y que establece representaciones consistentes de los patrones de voz que pueden usarse para comparaciones confiables a partir de un conjunto de muestras rotuladas, usando algoritmos de entrenamiento; dicho enfoque se muestra en la Figura # 9. La representación de los patrones de voz puede ser una plantilla, o un modelo estadístico, que puede aplicarse a un sonido, una palabra, o una frase.

En la etapa de comparación de patrones se realiza una comparación directa entre la señal de voz desconocida y todos los posibles patrones aprendidos en la etapa de

entrenamiento, de manera de determinar el mejor ajuste de acuerdo a algún criterio (Rabiner & Juang, 1993).

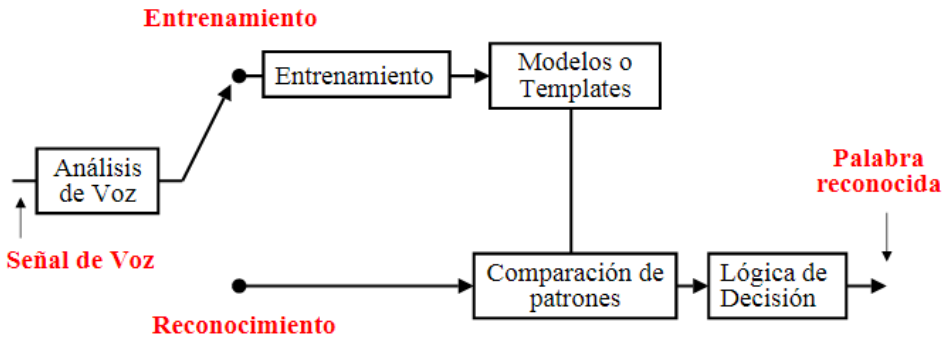

Figura # 9 Reconocimiento de voz basado en el reconocimiento de patrones. (www.fceia.unr.edu.ar, s.f.).

- **Enfoque de la inteligencia artificial (IA)**

En este enfoque se intenta automatizar el procedimiento de reconocimiento de acuerdo a la forma en que una persona aplica su inteligencia en la visualización, análisis y caracterización de la voz basada en un conjunto de características acústicas. Algunas técnicas que se emplean son: sistemas expertos (redes neuronales) que integran conocimientos prácticos fonéticos, sintácticos, semánticos para la segmentación y el rotulado, y usan herramientas tales como redes neuronales artificiales para aprender las relaciones entre eventos fonéticos. (Rabiner & Juang, 1993)

1.2.2. Representación de una señal de audio

- **Representación temporal**

Un método simple de representación de una señal de audio es realizar un dibujo de ésta en una gráfica dependiente del tiempo (véase Figura # 10). Esta representación se denomina representación en el dominio temporal. En este caso, se representa la evolución de la amplitud con respecto al tiempo. En el caso del sonido, la amplitud representa la variación de la presión atmosférica respecto al tiempo. En general, la

amplitud se representa a partir del valor cero (posición de equilibrio o valor medio de la presión) hasta el punto de máxima amplitud de la forma de onda (Roads, 1996).

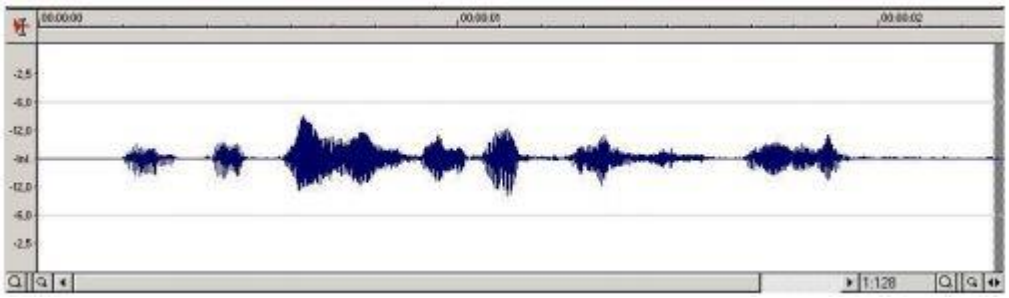

Figura # 10 Representación temporal de una señal de audio. (Gómez Gutierrez, 2009).

- **Representación frecuencial.**

La representación frecuencial captura las características espectrales de una señal de audio. Además de la frecuencia fundamental (la cual es la frecuencia más baja de una forma de onda periódica) existen muchas frecuencias presentes en una forma de onda. Una representación en el dominio frecuencial o representación espectral muestra el contenido frecuencial de un sonido. Las componentes de frecuencias individuales del espectro pueden denominarse harmónicos o parciales. Las frecuencias armónicas son enteros simples de la frecuencia fundamental. Cualquier frecuencia puede denominarse parcial, sea o no múltiplo de la frecuencia fundamental. De hecho, muchos sonidos no tienen una fundamental clara. (Gómez Gutierrez, 2009)

El contenido frecuencial de un sonido puede mostrarse de diversas maneras. Una forma estándar es la de dibujar cada parcial como una línea en el eje x. La altura de cada línea correspondería a la fuerza o amplitud de cada componente frecuencial. Una señal sinusoidal pura viene representada por una sola componente frecuencial. (Gómez Gutierrez, 2009)

❖ Espectro

El espectro de una señal es una representación en el dominio de la frecuencia que viene dada por la evolución de la amplitud y de la fase respecto a la frecuencia.

$$|A(f)|_{\psi(j)}$$

Otra medida adicional sería la densidad de potencia espectral (Power Spectrum Density) o PSD, que se aplica a ruidos de espectro continuo. Una definición simple de PSD es que la densidad de potencia espectral es igual ~~al~~ espectro de potencia dentro de un ancho de banda específico (Roads, 1996)

- Espectro de potencia

Del espectro de amplitud se puede derivar el espectro de potencia. Generalmente, se define la potencia de una señal como el cuadrado de la amplitud de dicha señal. Por tanto, el espectro de potencia sería el cuadrado del espectro de amplitud. La potencia espectral tiene relación con la percepción humana de la intensidad, y por ello es útil esta representación. (Roads, 1996).

$$P(f) = A(f)^2$$

El espectro cambia constantemente, por lo que las representaciones mencionadas anteriormente presentan sólo una porción de sonido que se ha analizado. Esta gráfica puede dibujarse de forma tridimensional, representando los distintos espectros a lo largo del tiempo.

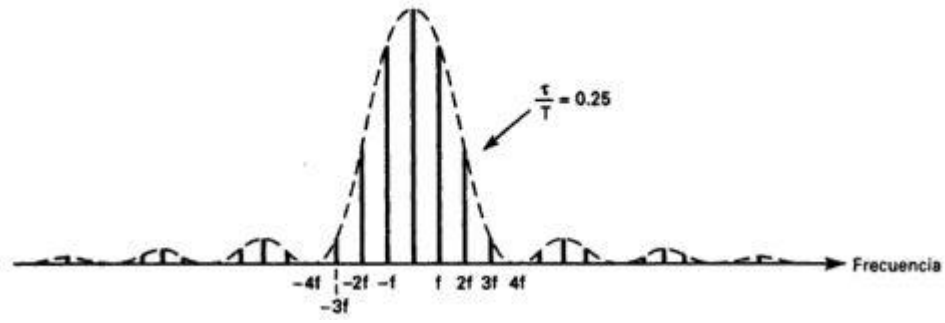

Figura # 11 Espectro de potencia de un pulso rectangular con ciclo de trabajo de 25% (Wayne, 2003).

Otra forma de representar esta variación es dibujando un sonograma o espectrograma. Este tipo de gráficas presentan la variación del contenido frecuencial respecto al

tiempo, donde el tiempo se presenta en el eje horizontal (o eje de las abscisas), la frecuencia en el eje vertical (o eje de las ordenadas) y la amplitud se dibuja a través de distintos colores del trazo. A continuación, se muestra una herramienta para representar tiempo-frecuencia de una señal. (Gómez Gutierrez, 2009)

- Short Time Fourier Transform

La transformada de Fourier es una herramienta matemática que permite pasar de una representación en el dominio temporal a una representación en el dominio frecuencial. De manera general, la Transformada de Fourier es un procedimiento matemático que mapea cualquier señal analógica estacionaria a una serie infinita de sinusoides, cada una de ellas con una determinada amplitud y fase (Gomez Gutierrez, 2009)

Para adaptar el análisis de Fourier al dominio de las señales muestreadas, de duración finita y variantes respecto al tiempo, se define la transformada de Fourier localizada a corto plazo (Short-Time Fourier Transformo STFT). En este caso, se divide la señal en pequeñas porciones o segmentos, que se denominan tramas de análisis. El contenido frecuencia o espectro se calcula en cada una de las tramas utilizando la transformada de Fourier (Roads, 1996)

STFT impone una secuencia de ventanas temporales (tramas de análisis) de la señal de entrada, es decir, divide la señal en fragmentos delimitados en el tiempo por una función ventana. Una ventana es un tipo específico de envolvente que se aplica para un análisis espectral. La duración de la ventana está normalmente comprendida entre $1\ ms$ y $1\ s$, y los segmentos a veces se transponen. A través del análisis espectral individual de cada segmento, se obtiene una secuencia de medidas (espectros) que constituyen al espectro variable a lo largo del tiempo o sonograma (Gómez Gutierrez, 2009). Desafortunadamente, este proceso tiene la desventaja de producir distorsiones en la medida del espectro, ya que el analizador de espectro no mide sólo la señal de entrada sino el producto de la misma por la ventana. El espectro que resulta es la convolución del espectro de la señal de entrada y el espectro de la ventana (Roads, 1996).

A continuación, la STFT aplica la transformada de Fourier discreta (DFT) a cada segmento. La DFT es un tipo de transformada de Fourier que puede trabajar con señales discretas en el tiempo, es decir, muestreadas.

1.2.3. Procesamiento de la Señal de Audio

Después de obtener la señal con éxito es necesario procesarla mediante un algoritmo que permita obtener sus características específicas y diferenciarla de las demás señales que se reciben. Por lo regular este paso lo lleva a cabo una aplicación especial, en el caso de las aplicaciones desarrolladas en Java, esto lo realiza JSAPI o alguna de sus versiones desarrolladas por terceros, ésta a su vez utiliza los modelos ocultos de Markov (HHM por sus siglas en ingles), los cuales a su vez son un modelo estocástico para ejercer el reconocimiento de voz.

1.2.3.1. Procesos estocásticos

Un proceso estocástico es una colección de variables aleatorias $\{X_t: t \in T\}$ parametrizada por un conjunto T, llamado espacio parámetral, en donde las variables toman valores en un conjunto S llamado espacio de estados. (Rincón, 2012)

En los casos más sencillos se toma como espacio parámetral el conjunto discreto $T = \{0,1,2, \dots, n\}$ y estos números se interpretan como tiempos. En este caso se dice que el proceso es a tiempo discreto, y en general, se denota por $\{X_n: n = 0,1, \dots\}$, o bien:

$$X_0, X_1, X_2, \dots$$

Así, para cada n, X_n es el valor del proceso o estado del sistema al tiempo n. Este modelo corresponde a un vector aleatorio de dimensión infinita, tal y como se muestra en la Figura # 12.

El espacio parámetral puede tomarse como conjunto continuo $T = [0, \infty)$. En este caso se dice que el proceso es a tiempo continuo y este se denota por:

$$\{X_t: t \geq 0\}$$

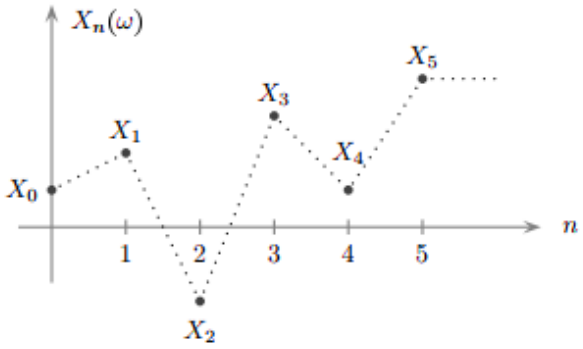

Figura # 12 Vector aleatorio de dimensión infinita (Rincón, 2012).

Por ende, si el subíndice es n, los tiempos son discretos, y si el subíndice es t, el tiempo se mide de manera continua. En particular, para poder hablar de variables aleatorias con valores en el espacio de estados S, es necesario asociar a éste conjunto una $\sigma - álgebra$.

Considerando que S es un subconjunto de $\mathbb{R}$**,** puede tomarse $\sigma - álgebra$ de Borel de $\mathbb{R}$ restringida a una S, es decir $S \cap \mathcal{B}(\mathbb{R})$. Un proceso estocástico, también llamado proceso aleatorio puede considerarse como una función de dos variables:

$$X: T \times \Omega \to S$$

Tal que a la pareja de (t, ω) se le asocia al valor o estado $X(t, \omega)$, lo cual también puede escribirse como $X_t(\omega)$. Para cada valor de t en T, el mapeo $\omega \to X_t(\omega)$ es una variable aleatoria, mientras que para cada ω en Ω fijo, la función $t \to X_t(\omega)$ es llamada una trayectoria o realización del proceso. Es decir, a cada ω del espacio muestral le corresponde una trayectoria del proceso, es por esta razón que los procesos estocásticos algunas veces se definen como una función aleatoria. (Rincón, 2012). Un ejemplo de dicha trayectoria sería un movimiento Browniano, el cual se puede ver en la Figura # 13.

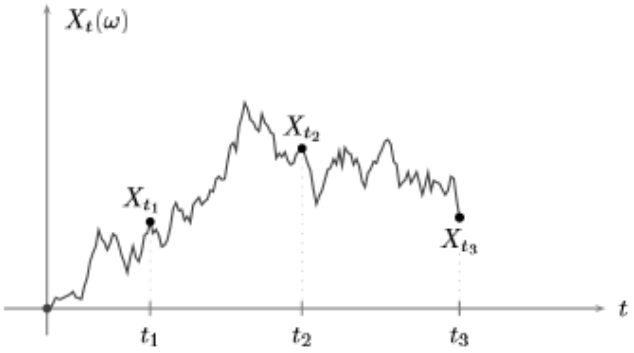

Figura # 13 Trayectoria de movimiento Browniano.

Los diferentes tipos de procesos estocásticos se obtienen al considerar las distintas posibilidades para el espacio parámetral, el espacio de estados, las características de las trayectorias, y principalmente las relaciones de dependencia entre las variables aleatorias que conforman el proceso.

Existen varios tipos de procesos estocásticos, entre ellos: es una clasificación,

- **Proceso de ensayos independientes**

El proceso a tiempo discreto $\{X_n: n = 0,1, \dots\}$ puede estar constituido por variables aleatorias independientes. Este modelo representa una sucesión de ensayos independientes de un mismo experimento aleatorio, como lanzar una moneda, o lanzar un dado en repetidas ocasiones. El resultado u observación del proceso en un momento cualquiera es, por lo tanto, independiente de cualquier otra observación pasada o futura del proceso (Rincón, 2012).

- **Procesos de Markov**

Estos tipos de procesos son modelos en donde, suponiendo conocido el estado presente del sistema, los estados anteriores no tienen influencia en los estados futuros del sistema. Esta condición se llama propiedad de Markov y puede expresarse de la siguiente forma: para cualquiera de los estados: $(x_0, x_1, \dots, x_{n-1}, x_n, x_{n+1})$

En donde:

x_{n-1} : Representa el pasado,

x_n : Representa el presente y

x_{n+1}: Representa el futuro

Y se cumple la igualdad:

$$P(X_{n+1} = x_{n+1} | X_0 = x_0, \dots, X_n = x_n) = P(X_{n+1} = x_{n+1} | X_n = x_n)$$

De esta forma la probabilidad del evento futuro $(X_{n+1} = x_{n+1})$ solo depende del evento $X_n = x_n)$, mientras que la información correspondiente al evento pasado $(X_0 = x_0, \dots, X_{n-1} = x_{n-1})$ es irrelevante. En particular, los sistemas dinámicos deterministas dados por una ecuación diferencial pueden considerarse procesos de Markov, pues su evolución futura queda determinada por la posición inicial del sistema y la ley de movimiento. (Rincón, 2012)

- **Procesos con incrementos independientes**

Se dice que un proceso estocástico a tiempo continuo $\{X_t : t \geq 0\}$ tiene incrementos independientes si para cualquier tiempo $0 \leq t_1 < t_2 < \cdots < t_n$, las variables $X_{t_1}, X_{t_2} - X_{t_1}, \dots, X_{t_n} - X_{t_{n-1}}$ son independientes. Esto quiere decir que los desplazamientos que tiene el proceso en estos intervalos disjuntos de tiempo son independientes unos de otros. (Rincón, 2012)

- **Procesos estacionarios**

Se dice que un proceso estocástico a tiempo continuo $\{X_t : t \geq 0\}$ es estacionario en el sentido estricto si para cualesquiera tiempos $t_1, \dots, t_n$, la distribución del vector $(X_{t_1}, \dots, X_{t_n})$ es la misma que la del vector $(X_{t_1+h}, \dots, X_{t_n+h})$ para cualquier valor de $h > 0$. En particular, la distribución de X_t es la misma que la de X_{t+h} para cualquier valor de $h > 0$. (Rincón, 2012)

- **Procesos con incrementos estacionarios**

Se dice que un proceso estocástico a tiempo continuo $\{X_t: t \geq 0\}$ tiene incrementos estacionarios si para cualesquiera tiempos $s < t$, y para cualquier $h > 0$, las variables $X_{t+h} - X_{s+h}$ y $X_t - X_s$ tienen la misma distribución de probabilidad. Es decir, el incremento que tiene el proceso entre los tiempos s y t sólo depende de estos tiempos a través de la diferencia $s - t$, y no de los valores específicos de s y t. (Rincón, 2012)

- **Martingalas**

Una martingala a tiempo discreto es, en términos generales, un proceso $\{X_n: n = 0,1, \dots\}$ que cumple la condición:

$$E(X_{n+1} \mid X_0 = x_0, \dots, X_n = x_n) = x_n$$

Esta igualdad significa que el valor promedio del proceso al tiempo futuro $n + 1$ es el valor del proceso en su último momento observado, es decir x_n. Esto es, se trata de una ley de movimiento aleatorio equilibrada o simétrica, pues en promedio el sistema no cambia del último momento observado. A estos procesos también se les conoce como procesos de juegos justos. (Rincón, 2012)

- **Procesos de Levy**

Se dice que un proceso estocástico a tiempo continuo $\{X_t: t \geq 0\}$ es un proceso de Levy si sus incrementos son independientes y estacionarios. Tanto el proceso de Poisson como el movimiento Browniano son ejemplos de este tipo de procesos (Rincón, 2012)

- **Procesos Gaussianos**

Se dice que un proceso estocástico a tiempo continuo $\{X_t: t \geq 0\}$ es un proceso Gausiano si para cualquier colección finita de tiempos $t_1, \dots, t_n$ tiene distribución normal o Gausiana multivariada. El movimiento Browniano también es un ejemplo de este tipo de procesos (Rincón, 2012).

- **Modelos ocultos de Markov**

Es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Markov de parámetros desconocidos. El objetivo es determinar los parámetros desconocidos (u ocultos, de ahí el nombre) de dicha cadena a partir de los parámetros observables. Los parámetros extraídos se pueden emplear para llevar a cabo sucesivos análisis. Un modelo oculto de Markov se puede considerar como la red bayesiana dinámica más simple (Macas Macas & Padilla Pineda, 2012).

Son modelos estocásticos, que toman a la incertidumbre para realizar cálculos, analizando los procesos que se presentan de forma aleatoria entre estados en los cuales tiene influencia el azar. En concreto un modelo oculto de Markov es un conjunto finito de estados de los cuales está asociado una distribución de la probabilidad. Las transacciones entre los estados son administradas por un conjunto de probabilidades de transición. Para un determinado estado, un resultado u observación se puede generar de acuerdo a la distribución de probabilidad asociada a éste.

Al mismo tiempo los modelos ocultos de Markov permiten modelar procesos o datos secuenciales, lo que permite una mejor representación de los eventos probabilísticos producidos de manera aleatoria entre estados.

La principal característica de los modelos ocultos de Markov, y de la cual adquieren su nombre es su doble proceso estocástico, uno oculto y otro observable, mientras que los estados del proceso oculto no se pueden ver directamente, las variables que son influenciadas por este si pueden ser observadas (Macas Macas & Padilla Pineda, 2012).

Los estados de un modelo oculto de Markov son:

- N: Ésta representa el número de estados que tiene el modelo. Estos se encuentran ocultos:
  - Estado: $S = \{S_1, S_2, S_3, \dots, S_n\}$

- Estados en el tiempo $t\ como\ q_t$

❖ M: Ésta representa el número de observaciones producidas. Si éstas son continuas, entonces M tiene a infito. Ésta se denota mediante:

- Símbolo de observación: $V = \{V_1.V_2, V_3, \dots, V_M\}$
- Observación en tiempo: $t, O_t \in V$

1.2.4. Lenguajes De Programación En El Reconocimiento De Voz

Son lenguajes formales diseñados con el propósito de realizar procesos que pueden ser llevados a cabo por máquinas. Estos lenguajes pueden ser utilizados para crear programas que controlen el comportamiento lógico o físico de una máquina, para expresar algoritmos con precisión, o para comunicar una máquina con un ser humano (Depto.CCIA Universidad Alicante, 2006).

Un lenguaje de programación está formado por una serie de símbolos y reglas sintácticas y semánticas las cuales definen su estructura, así como el significado de sus elementos y expresiones. Es importante conocer que al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se llama programación (Depto.CCIA Universidad Alicante, 2006).

Aunque también la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos.

- **JAVA**

Es un lenguaje de programación de propósito general concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares

en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados (ORACLE CORPORATION, s.f.).

- **IDE**

Aunque Java es un lenguaje de programación, y es posible escribirlo dentro de cualquier editor de texto, lo más cómodo es hacerlo dentro de un entorno de desarrollo integrado. Un entorno de desarrollo integrado es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. (Alonzo Velázquez, 2010)

Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador.

Los IDE están diseñados para maximizar la productividad del programador proporcionando componentes muy unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se lleva a cabo todo el desarrollo. Generalmente, este programa suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de software (Alonzo Velázquez, 2010).

- **NetBeans**

Es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java, aunque existe además un número importante de módulos para extendérselo a otros lenguajes. (NetBeans, 2018)

NetBeans es un producto libre y gratuito, sin restricciones de uso, permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial que lo identifica como módulo (NetBeans, 2018)

- **API**

Como se menciona en el párrafo anterior, NetBeans hace uso de módulos y APIs. Un API es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción (3scale, 2012).

Son usadas generalmente en las bibliotecas de programación. Estas clases son escritas utilizando el lenguaje de programación Java y ejecutada con ayuda de JVM.

El API desarrollada para el reconocimiento de voz en Java es JSAPI (Java Speech API), esta permite a la aplicación Java incorporar tecnología de dictado. Define una API entre plataformas que soporta comandos y reconocedores de control, así como sistemas de dictado y sintetizadores de voz. Esta API fue liberada desde el 26 de octubre de 1998 en su versión 1.0 (Fazzino & Sanchéz, 2008).

- **Java Speech API (JSAPI)**

El JSAPI, desarrollado por Sun Microsystems en cooperación con Apple Computer Inc., AT&T, Dragon Systems Inc., IBM Corporation, Novell Inc., Philips Speech Processing, Texas Instruments Incorporated, entre otras, define una interfaz del software que permite a los diseñadores aprovechar la tecnología del reconocimiento del habla. El JSAPI define un estándar de fácil uso, además soporta dos tecnologías: el reconocimiento del habla, y el sintetizador de la misma. (Fazzino & Sanchéz, 2008)

El primero proporciona la habilidad a las computadoras para escuchar el idioma hablado y determinar lo que se ha dicho. En otros términos, procesa la entrada de audio que contiene el mensaje hablado convirtiéndolo en texto. Mientras que la síntesis del habla o síntesis de voz proporciona el proceso inverso de producir un mensaje, es decir, sintetiza el texto generado por una aplicación, un applet o un usuario y lo convierte en sonido. (Fazzino & Sanchéz, 2008)

La manera principal en que una aplicación controla la actividad de un reconocedor del habla es por medio de las reglas gramaticales. Una regla gramatical es un objeto en el JSAPI que indica qué palabras se espera que diga un usuario y el orden en que éstas podrán ser aceptadas por el sistema. (Fazzino & Sanchéz, 2008)

1.2.4.1. Grammar

Pero no basta con agregar JSAPI al proyecto para realizar el reconocimiento de los comandaos de voz, pues el reconocimiento de patrones, requiere de un archivo el cual dicte las reglas gramaticales, es decir, cuáles serán las palabras que podrán ser reconocidas, y el orden de las mismas.

La gramática es un objeto de JSAPI en el cual se indican las palabras que se esperan sean emitidas por el locutor, y la secuencia en la que estas pueden ocurrir. Este archivo es vital, pues las restricciones que se especifican en este hacen que el reconocimiento de señales funcione de manera más rápida y efectiva ya que el reconocedor se enfoca sólo en las palabras que se encuentran en este archivo sin la necesidad de que busque palabras u oraciones extrañas.

Para esto es necesario crear un archivo JSGF (Java Speech Grammar Format), el cual es una plataforma independiente, que se desempeña como un proveedor de gramática para aplicaciones de reconocimiento del habla. Este tipo de archivos denominados Grammars son utilizados en el reconocimiento del habla para determinar qué es lo que el reconocedor deberá reconocer y para describir la pronunciación de palabras que un locutor podrá hacer. JSGF adopta el estilo y lineamientos del lenguaje de programación Java, además del uso de las notaciones gramaticales tradicionales. (Hunt, 2000)

Los sistemas de reconocimiento del habla proveen a la computadora de la habilidad para escuchar el dictado de un locutor, y determinar lo que este ha dicho. Pero la tecnología actual, todavía no soporta el tipo de reconocimiento del habla sin restricciones, es decir, la habilidad de escuchar cualquier dictado en cualquier contexto y comprenderlo acertadamente. Para alcanzar un razonable éxito tanto en el

reconocimiento del habla como en su velocidad de respuesta, los reconocedores actuales restringen lo que escuchan y pueden reconocer mediante el uso de grammars. (Hunt, 2000)

Los JSGF definen una manera de describir un tipo de gramática o regla gramatical. Ésta usa una representación textual la cual puede ser leída y editada tanto por desarrolladores como por equipos de cómputo, al mismo tiempo que puede ser incluida dentro del código fuente. (Hunt, 2000)

Una regla gramatical especifica los tipos de combinaciones de palabras que un locutor podrá utilizar, esto es bastante similar a la sintaxis de una oración escrita. Lo que un usuario dice depende del contexto, pero como se mencionó anteriormente, la computadora es incapaz de reconocer dicho contexto, es por esto que para que una aplicación de reconocimiento de voz realice un reconocimiento adecuado, es necesario proveer a dicha aplicación, con un gramar adecuado a las palabras con las que podrá encontrarse. (Hunt, 2000)

Nombre del Grammar y nombres de los paquetes

Cada gramar definido por JSGF tiene un nombre único, el cual es declarado en el cabecero del mismo, la estructura para esto es:

$$nombreDelPaquete.nombreSimpleDelGrammar$$

$$nombreDelGrammar$$

La primera a $(\text{nombreDelPaquete} + \text{nombreSimpleDelGrammar})$ es el nombre completo del gramar, la segunda forma es únicamente el nombre simple del gramar.

Los nombres tanto de paquetes como de grammars tienen el mismo formato que los paquetes y clases en el lenguaje de programación Java. El nombre completo de un gramar es una lista de identificadores separados por puntos. Esto minimiza la posibilidad de conflictos generados por la duplicidad de nombres (Hunt, 2000).

Nombres de reglas

Un gramar está compuesto por un conjunto de reglas, que definen lo que puede ser hablado y será reconocido. Las reglas son combinaciones de palabras escritas con posibilidades de ser dichas por un locutor, así como también referencias a otras reglas. Cada regla tiene un nombre único. La referencia a una regla se representa mediante el nombre de la regla escrito dentro de los caracteres (<>).

Un nombre válido para una regla es similar a un identificador en Java, con la diferencia de que éste, permite una serie de símbolos extra. Un nombre valido para una regla es una secuencia sin límite de caracteres Unicode, los cuales cumplen con las siguientes características:

- Puede ser cualquier carácter valido para un identificador en Java.
- Puede contener: + - : ; = | / \ ( ) [ ] @ # % ¡ ^ & ~

Es muy importante tener en consideración que los nombres de reglas son comparados con una coincidencia exacta de caracteres Unicode, esto quiere decir que tanto mayúsculas como minúsculas son consideradas caracteres diferentes, así también, es importante conocer que no se permiten espacios en blanco dentro de los nombres de reglas. (Hunt, 2000)

Los nombres de reglas $< null >$ y $< void >$ son nombres reservados. JSGF incluye la mayoría de los lenguajes hablados en la actualidad, así que las reglas pueden ser escritas en varios lenguajes, con sus respectivas limitantes. (Hunt, 2000)

- **Nombres calificados y totalmente calificados**

Aunque los nombres de las reglas son únicos dentro de un gramar, éstos pueden ser reutilizados en alguna otra, así que si se llega a hacer referencia de un gramar dentro de otro y éstos contienen alguna regla con el mismo nombre, generara ambigüedad. Los nombres calificados y totalmente calificados se utilizan para generar referencias entre grammars sin ambigüedad. (Hunt, 2000)

Un nombre totalmente calificado incluye tanto el nombre completo del gramar como el nombre simple de la regla, mientras que un nombre calificado incluye únicamente el nombre simple del gramar y el nombre de la regla.

- **Acuerdos sobre nombres de reglas**

❖ Las reglas locales tienen prioridad. Si una regla local y una o más reglas importadas tienen el mismo nombre, entonces la referencia a una regla mediante su nombre simple hará referencia a la regla local.

❖ Si dos o más reglas importadas tienen el mismo nombre, pero no existe regla local que lo comparta, entonces una referencia a esta regla mediante su nombre simple será ambigua y por ende generará un error. Para resolver este tipo de error es necesario hacer la referencia a la regla mediante su nombre calificado o totalmente calificado.

❖ Si dos o más reglas importadas tienen el mismo nombre y ambas provienen de grammars con el mismo nombre simple, entonces la referencia a ésta deberá ser mediante su nombre totalmente calificado.

❖ Una referencia mediante el nombre totalmente calificado nunca es ambigua.

Cuando la referencia a una regla no puede ser resuelta, pues no ésta definida localmente y no pertenece a un gramar importado), el manejo de dicha referencia es definida mediante la interfaz del reconocedor. (Hunt, 2000)

Tokens

Un token, también llamado símbolo de terminal es la parte del grammar que define que puede ser hablado por un usuario, por lo regular un token es equivalente a una palabra. Así mismo los tokens pueden aparecer aislados o como una secuencia de tokens separados por un espacio en blanco. (Hunt, 2000). Es una referencia a una entrada en el vocabulario del reconocedor, regularmente se hace referencia a este como lexicón. El vocabulario del reconocedor define la pronunciación de los tokens y con la pronunciación, el reconocedor es capaz de escuchar dicho token. (Hunt, 2000)

En JSGF un token es una secuencia de caracteres encerrados por espacios en blanco, por comillas, o delimitado por algún otro símbolo significativo dentro del grammar como:

; = | * + < > ( ) { } [ ] /* */ //

JSGF permite grammars multi-lenguajes, esto quiere decir que acepta tokens en más de un idioma.

La mayoría de los reconocedores tienen un lenguaje exhaustivo para cada lenguaje que manejan, pero sería imposible incluir el 100% de dicho lenguaje, así que para todas las palabras que el vocabulario no contempla existen tres posibilidades.

1. Una aplicación o usuario pueden agregar el token y la pronunciación al vocabulario para asegurar que el reconocimiento se realice de manera correcta.
2. Algunos reconocedores son capaces de suponer la pronunciación de varias palabras que no se encuentren dentro del vocabulario.
3. Si ninguna de las opciones pasadas funciona, el comportamiento dependerá de la interfaz del software.

Los tokens no tienen que ser palabras del lenguaje escritas de forma común, suponiendo que el token ha sido definido propiamente dentro del vocabulario del reconocedor. Por ejemplo, para manejar la pronunciación diferente de alguna palabra dependiendo ya sea por el contexto que la rodea, o por algún signo de puntuación, se pueden declarar tokens diferentes dentro del JSGF. (Hunt, 2000)

- **Símbolos y Puntuación**

La mayoría de los reconocedores proporciona la habilidad de manejar símbolos y signos de puntuación sencillos. Aun así, existen varias formas que son difíciles de manejar sin ambigüedad. En estos casos el desarrollador debe crear tokens que representen de manera apropiada la forma en la que las personas podrán hablar, por ejemplo:

- Números: “1 2 3”, estos deberán ser declarados mediante su nombre completo, “uno dos tres”
- Fechas “21/11/2017” estas deberán ser declaradas como las pronuncie el locutor, en este caso en particular “veintiuno de noviembre de dos mil diecisiete”
- Abreviaciones: “Dr.”, “Ing.”, al igual que con los nombres, éstas deberán ser escritas en la forma de la palabra completa, “doctor” “ingeniero”.
- Símbolos especiales: “&”, “+”, al igual que los números, estos deberán de declararse según sea el nombre del símbolo, o mejor dicho su pronunciación. (Hunt, 2000)

JSGF soporta varios lenguajes, pero originalmente se creó para el idioma inglés, así que presenta ciertas limitantes al momento de reconocer palabras locales. En el caso del idioma español, en específico, del español hablado en México, JSGF no reconoce la pronunciación de la letra “x” por esto es necesario analizar la pronunciación de esta letra dependiendo de la palabra, y sustituirla por la letra adecuada según sea el caso (j, cs, cc, …), así mismo, no permite declarar palabras con acentos, por lo que se recomienda utilizar una letra mayúscula en el lugar donde se sitúa una vocal acentuada.

Encabezado

La definición del gramar contiene dos partes:

1. Encabezado: Incluye un encabezado de auto identificación, declara el nombre del grammar, e importa reglas de otros grammars.
2. Cuerpo: Define las reglas del grammar, algunas de éstas pueden ser públicas.

- **Encabezado de Auto Identificación**

Un archivo JSGF comienza con un encabezado de auto identificación. Éste identifica el contenido del documento así como la versión del JSGF que se utiliza. Algunas veces y de manera opcional, se especifica el tipo de cifrado que se utiliza en el documento. También opcionalmente se puede especificar el lugar del grammar, esto se refiere a la

lengua y opcionalmente el país o variante regional que el grammar soportará. Las declaraciones del cabecero terminan con ";" . (Hunt, 2000)

El formato del encabezado de auto identificación es:

$$\#JSGF\ version\ char-encoding\ locale;$$

En este ejemplo, no se proporciona el tipo de cifrado, ni el lugar, así que éstos se asumen por default.

El carácter de hashtag $(\#)$ debe ser el primero del documento, y todos los caracteres en el encabezado de auto identificación deben estar en el subconjunto ASCII de la codificación que se utiliza.

- **Nombre del Grammar**

El nombre del grammar debe ser declarado como la primera sentencia del grammar la declaración debe utilizar el nombre completo del grammar. En consecuencia, el formato de declaración es:

$$grammar\ nombreDelPaquete.nombreSimpleDelGrammar$$

$$grammar\ nombreDelGrammar$$

Cuerpo del Grammar

- **Definición de Reglas**

En el cuerpo del grammar se definen las reglas. Cada regla es definida dentro de una definición de regla, y es definida solo una vez en el grammar. El orden, o la definición de la regla no son importantes. (Hunt, 2000)

Existen dos patrones para definir una regla:

1. $< nombreDeRegla >= expansionDeRegla;$
2. $public < nombreDeRegla >= expansionDeRegla;$

Los componentes para definir una regla son:

1. Una declaración $public$ (opcional).
2. Nombre de la regla.
3. Un signo de igual (=).
4. La expansión de la regla
5. Un signo de ; para cerrar la definición

Los espacios en blanco son ignorados en la definición, éstos sólo tienen valor dentro de la expansión de la regla.

La expansión de la regla define como podrá ser hablada la regla. Una expansión de regla es una combinación de tokens y referencias a otras reglas. El termino expansión se utiliza ya que hace referencia a la manera en la que una regla se expande al ser hablada, ya que una regla puede expandirse en n combinaciones de palabras habladas, asi como en la expansión de otras reglas. (Hunt, 2000)

- **Reglas Públicas**

Cualquier regla dentro de un grammar puede ser declarada como pública con el simple uso de la palabra $public$. Una regla pública tiene tres posibles usos:

1. Puede ser referenciada fuera de alguna definición de regla, o de algún otro grammar únicamente mediante su nombre completamente calificado o mediante una declaración $import$.
2. Puede ser usada una regla activa para el reconocimiento. Esto significa que la regla puede ser utilizada por el reconocedor para determinar lo que puede ser hablado.
3. Se puede referenciar de manera local por cualquier regla pública o no que haya sido definida dentro del mismo grammar.

Sin la palabra $public$ la regla es implícitamente privada y sólo puede ser referenciada dentro de la definición de reglas en el grammar local.

- **Expansión de reglas**

Las expansiones de reglas más simples son una referencia a un token y una referencia a una regla. Por ejemplo:

$$< a >= palabra;$$

$$< b >=< x >;$$

$$< c >=< com.acme.grammar.y >;$$

Lo anterior se puede explicar cómo:

- Que para hablar $< a >$ el usuario deberá decir “palabra”
- La regla $< b >$ se expande como $< x >$ lo cual significa que el usuario deberá decir algún token que este definido dentro de la regla $< x >$ para hablar $< b >$.
- Para hablar $< c >$ el usuario debera decir algún token definido dentro de la regla $< com.acme.grammar.y >$

En términos formales lo anterior se traduce como el hecho de que las siguientes expansiones son válidas.

- Cualquier token o serie de tokens.
- Alguna referencia a alguna regla no publica definida dentro del mismo grammar.
- Alguna referencia a alguna regla publica definida en algún otro grammar y que haya sido importada dentro del grammar.
- Alguna referencia de alguna regla publica definida dentro de algún otro grammar, siempre y cuando se referencie esta mediante su nombre completamente calificado (con o sin la sentencia $import$).

La definición de una regla puede referenciar una regla privada de algún otro grammar a través de una regla pública de algún otro grammar.

Una definición vacía, no es valida

$$< d >= ;$$ // esta definición no es valida

Pero la definición de alguna regla ya sea $< null >$ o $< void >$ si es valida

$$< e >=< null >;$$ //valida

$$< f >=< void >;$$ //valida

Composición

- **Secuencias**

Una regla puede ser definida por una secuencia de expansiones validas, cada una separada mediante un espacio en blanco.

Para que el usuario hable alguna de estas reglas, es necesario que cada componente (token, regla o alguna secuencia) sea dicho en el orden exacto en el que fue definido en la regla.

- **Alternativas**

Una regla puede ser definida como un conjunto de tokens o alternativas, separadas mediante pipes "|" y opcionalmente por espacios en blanco.

$$< nombreRegla >= alternativa1 |< alternativa2 > | alternativa3 | \dots | alternativaN;$$

Este caso es diferente a las secuencias, ya que el usuario para hablar la regla en cuestión solo debe decir alguna de las alternativas, y no la secuencia exacta. (Hunt, 2000). Al igual que en las reglas, no se permite declarar una alternativa vacía.

$$< nombreRegla >=< alternativa1 > | | alternativa3;$$ //no valida

$$< nombreRegla >=< alternativa1 > \mid alternativa2 \mid;$$ //no valida

Agrupamiento

- **Paréntesis**

Cualquier expansión valida puede ser agrupada mediante el uso de paréntesis. El agrupamiento tiene una prioridad alta, por lo que se utiliza para asegurar la correcta interpretación de las reglas. (Hunt, 2000)

$$< nombreRegla >= alternativa1\ (alternativaX \mid alternativaY \mid alterntivaZ);$$

Lo anterior significa que para hablar la regla anterior el usuario deberá pronunciar alguna de las secuencias siguientes:

- $alternativa1\ alternativaX$
- $alternativa1\ alternativaY$
- $alternativa1\ alternativaZ$

La declaración vacía de paréntesis no es válida.

- **Agrupamiento opcional**

Corchetes pueden ser ubicados alrededor de cualquier definición de regla, lo que indica que el contenido de la misma es opcional, este tipo de agrupamiento tiene el mismo nivel de prioridad que los paréntesis. (Hunt, 2000)

$$< nombreRegla1 >= (alternativaX \mid alternativaY \mid alterntivaZ);$$

$$< nombreRegla2 >= [< nombreRegla1 >]\ palabra;$$

Esto permite al usuario decir alguna alternativa de $< nombreRegla1 >$ más $palabra$ o simplemente decir $palabra$ para pronunciar la regla $< nombreRegla2 >$.

Al igual que los paréntesis, la definición de corchetes vacíos no es válida. (Hunt, 2000)

1.2.5. Actuadores

Un actuador es un dispositivo con la capacidad de generar una fuerza que ejecute un cambio de posición, velocidad o estado de algún tipo sobre un elemento mecánico, a partir de la transformación de energía (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014)

Los actuadores se clasifican en dos grupos:

1. Por el tipo de energía utilizada como se muestra en la Figura # 14 (neumático, hidráulico, eléctrico).
2. Por el tipo de movimiento que generan (lineales y rotatorios).

- **Actuadores eléctricos**

Los actuadores eléctricos transforman de energía eléctrica en energía mecánica ya sea rotacional o lineal. Son los actuadores más utilizados, ya que su fuente de alimentación en la energía eléctrica, mientras que los actuadores neumáticos o hidráulicos necesitan compresores para generar dicha energía.

Los actuadores eléctricos se basan en el principio de funcionamiento en el cual se establece que si en un filamento conductor por el cual circula una corriente eléctrica se ubica dentro de la acción de un campo magnético, dicho filamento experimenta una fuerza electromagnética que induce un desplazamiento perpendicular a las líneas de acción del campo magnético. (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

Con la finalidad de aumentar la magnitud de la fuerza de desplazamiento, un actuador eléctrico ésta constituido por un gran número de filamentos conductores, conocidos como espiras. Debido a que la corriente eléctrica que circula a través de un conjunto de espiras adquiere propiedades magnéticas, éste provoca el movimiento circular en el eje (rotor) del actuador, gracias a la interacción con los polos (imanes o electroimanes), con lo que se produce la energía mecánica. (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

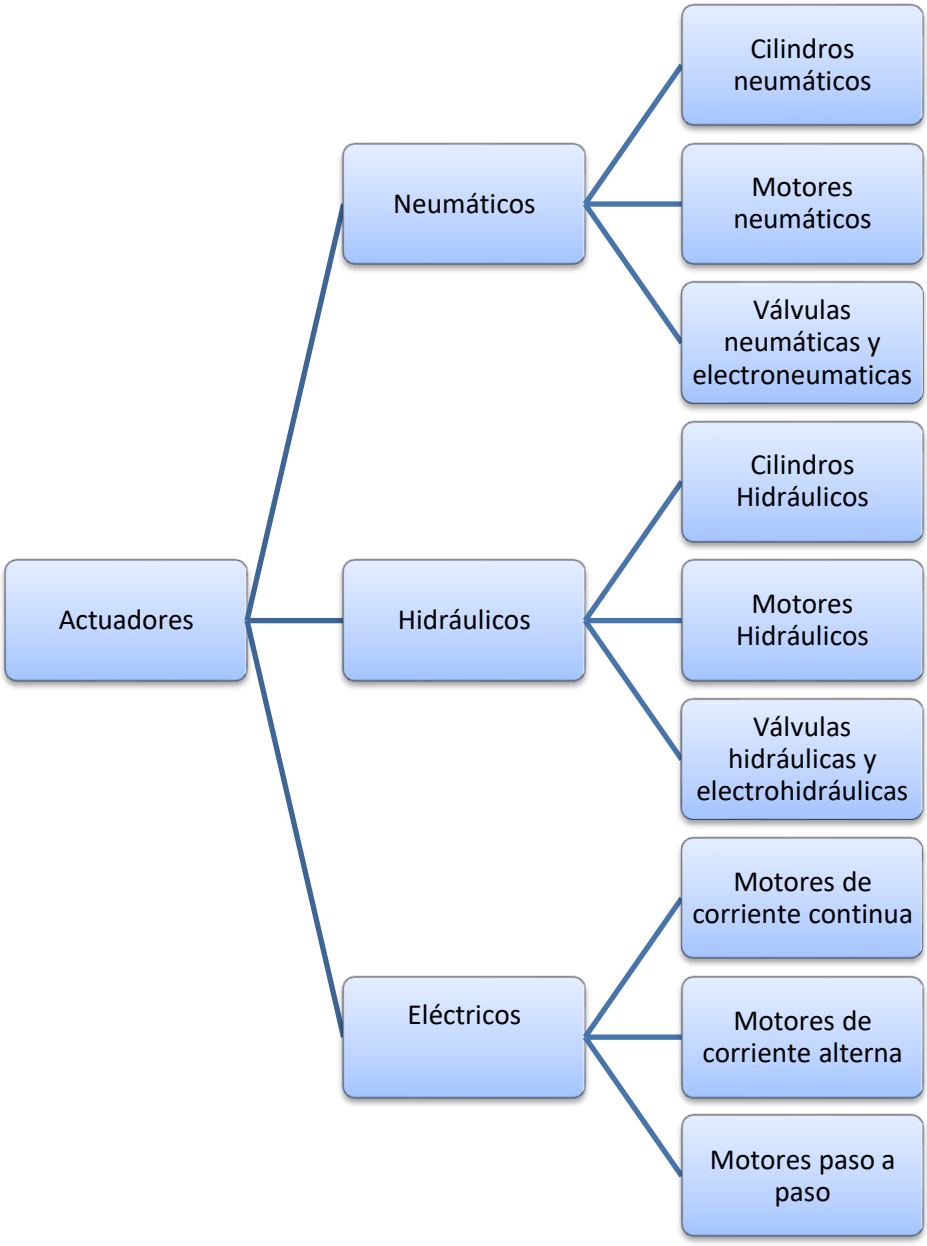

Figura # 14 Clasificación de los actuadores según el tipo de energía utilizada (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014)

Por lo general los actuadores eléctricos se clasifican de acuerdo con el tipo de energía eléctrica con la que son alimentados, por el tipo de movimiento que generan y por la forma de excitación entre otros aspectos (véase Figura # 15).

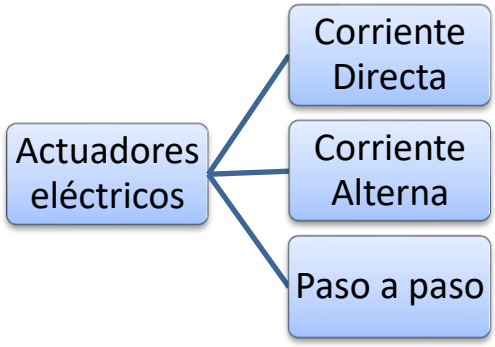

Figura # 15 Clasificación de los actuadores eléctricos (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

- **Actuadores de corriente directa**

Para su funcionamiento, los actuadores de corriente directa demandan un flujo electrolítico de corriente que circula en un solo sentido. Este tipo de actuadores se compone de dos partes fundamentales conocidas comúnmente como rotor y estator. El rotor constituye la parte móvil del actuador, además de que es la parte que proporciona la fuerza que actúa sobre el elemento mecánico. Por su parte, el estator constituye la parte fija del actuador y es aquella que provoca el magnetismo necesario para inducir la fuerza electromotriz. Una de las principales características de los actuadores de corriente directa radica en que al variar el voltaje de alimentación se puede modificar la velocidad del eje del actuador ya que la velocidad de rotación en un motor de corriente directa es proporcional al voltaje, además de que el par es proporcional a la corriente que circula por el devanado. (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014)

- **Actuadores de corriente alterna**

Se sirven de un flujo eléctrico en el cual la intensidad cambia de dirección periódicamente, esto como consecuencia del cambio periódico de polaridad de la tensión aplicado en los bornes de alimentación del motor.

En aplicaciones de velocidad variable, los motores de corriente alterna dependen del funcionamiento de la frecuencia de operación de voltaje aplicado para modificar los rangos de velocidad. (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014)

- **Motores paso a paso**

Este tipo de motores funciona con el mismo principio físico fundamental de los actuadores de corriente directa y de corriente alterna, solo que este tipo de actuador electromecánico convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo cual implica que éste es capaz de avanzar un determinado valor de grados (pasos) del eje motriz dependiendo de las entradas de control. En el mercado existen tres tipos de motores paso a paso:

1. De imanes permanentes
2. De reluctancia variable
3. Híbridos

- **Motores paso a paso de imán permanente**

Este tipo de motor se conoce como motor paso a paso de imán permanente debido a que un imán cerámico con forma de cilindro dentado constituye el rotor de ese y porque su estator está fabricado de material ferromagnético dispuesto en forma de láminas, esta característica del rotor representa una de las principales ventajas de este motor, ya que en ausencia de excitación eléctrica el eje del motor permanece en la misma posición, este se muestra en la Figura # 16 (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

- **Motores paso a paso de reluctancia variable**

Este tipo de actuadores paso a paso está construido por un rotor dentado construido a base de láminas ferromagnéticas y un estator donde se disponen bobinas que forman los polos, los cuales se alojan en ranuras, de manera longitudinal, de modo que hacen más efectiva la acción del campo magnético, gracias a que el material con que están

construidas ofrece baja resistencia a la circulación del flujo magnético (véase Figura # 17). Su funcionamiento es similar a los actuadores de imán permanente, aunque en condiciones de reposo no existe par en el eje del motor, lo que significa que el rotor gira con libertad. (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014)

La característica principal de este tipo de actuador es que si se requiere se puede construir para que funcione con pasos más pequeños que los de un motor de imán permanente.

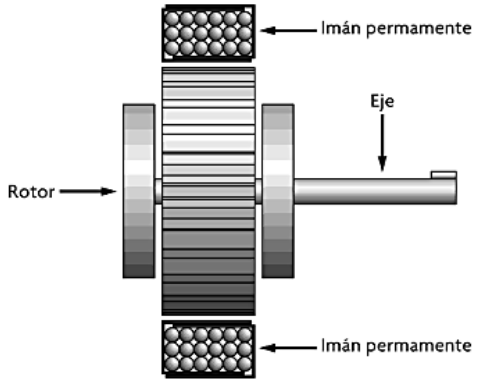

Figura # 16 Vista en sección de un motor paso a paso de imán permanente (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

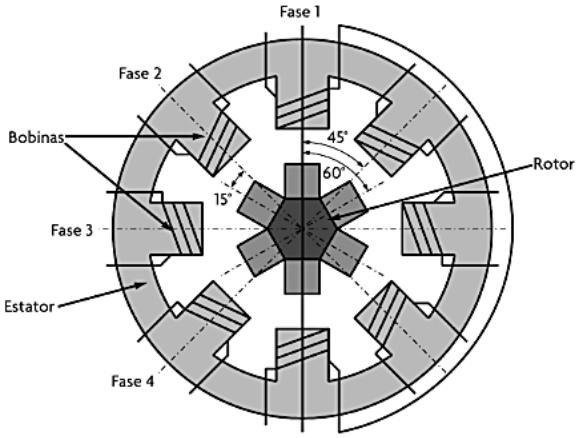

Figura # 17 Vista seccional de un motor paso a paso de reluctancia variable (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

- **Motores paso a paso híbridos**

Se conocen como motores híbridos todos aquellos que combinan las características de los motores de imán permanente y de reluctancia variable como se ejemplifica en la Figura # 18. Una cualidad palpable de esta característica de hibridad es que este tipo de motores heredan las ventajas de cada uno de estos tipos de motores, los cuales se conjugan en un solo motor, lo que se ve reflejado en la obtención de ángulos pequeños de paso con un alto par. El estator en los motores híbridos es similar al de los motores tratados con anterioridad; sin embargo, difieren en que el rotor de los motores híbridos está conformado por un imán o material imantado en forma de disco cilíndrico en posición longitudinal al eje, el cual produce un flujo magnético (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

Después se realiza el reconocimiento de comandos de voz con ayuda de NetBeans y JSAPI, es necesario realizar una conexión para que por medio de estos comandos se pueda controlar el actuador eléctrico.

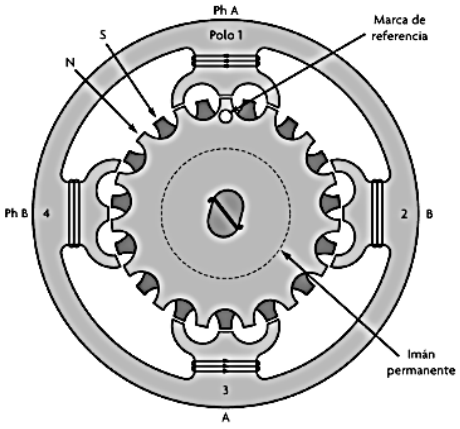

Figura # 18 Motor paso a paso híbrido (Corona Ramírez, Abarca Jiménez, & Mares Carreño, 2014).

1.2.6. Tarjeta Arduino Mega 2560

La conexión mencionada anteriormente se realiza mediante Arduino Mega. Esta es una placa de microcontrolador basada en ATmega1280. Tiene 54 pines digitales de entrada y salida (de los cuales 14 se pueden usar como salidas PWM), 16 entradas analógicas,

4 UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, un encabezado ICSP, y un botón de reinicio. Contiene todo lo necesario para soportar el microcontrolador; basta con conectarlo a una computadora con un cable USB o con un adaptador de CA a CC o batería para comenzar. El Mega es compatible con la mayoría de los escudos diseñados para el Arduino Duemilanove o Diecimila (Arduino, 2018).

Figura # 19 Tarjeta Arduino Mega (Arduino, 2018)

- **Composición**

Componente	Descripción
Microcontrolador	ATmega1280
Tensión de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines de E / S digitales	54 (de los cuales 15 proporcionan salida de PWM)
Clavijas de entrada analógica	Dieciséis
Corriente DC por Pin E / S	40 mA
Corriente DC para 3.3V Pin	50 mA

Memoria flash	128 KB de los cuales 4 KB utilizados por el gestor de arranque
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz

- **Alimentación**

El Arduino Mega puede alimentarse a través de la conexión USB o con una fuente de alimentación externa. La fuente de poder se selecciona automáticamente.

La alimentación externa (no USB) puede provenir de un adaptador de CA a CC (wall-wart) o batería. El adaptador se puede conectar al enchufar un conector positivo de 2.1 mm en el conector de alimentación de la placa. Los cables de una batería se pueden insertar en los conectores Gnd y Vin pin del conector POWER. (Arduino, 2018)

La placa puede operar con un suministro externo de 6 a 20 voltios. Sin embargo, si se suministra con menos de 7 voltios, el pin de 5 voltios puede suministrar menos de cinco voltios y la placa puede ser inestable. Si usa más de 12 voltios, el regulador de voltaje puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- VIN. El voltaje de entrada a la placa Arduino cuando usa una fuente de alimentación externa (a diferencia de 5 voltios de la conexión USB u otra fuente de alimentación regulada). Puede suministrar voltaje a través de este pin o, si suministra voltaje a través del conector de alimentación, acceda a través de este pin. (Arduino, 2018)
- 5V. La fuente de alimentación regulada utilizada para alimentar el microcontrolador y otros componentes en el tablero. Esto puede provenir de VIN a través de un regulador de a bordo, o puede ser suministrado por USB u otro suministro regulado de 5V. (Arduino, 2018)

- 3V3. Un suministro de 3.3 voltios generado por el chip FTDI a bordo. El consumo máximo de corriente es de 50 mA.
- GND. Pines de tierra

- **Memoria**

El ATmega1280 tiene 128 Kb de memoria flash para almacenar el código (de los cuales 4 Kb se utilizan para el gestor de arranque), 8 Kb de SRAM y 4 Kb de EEPROM (que se pueden leer y escribir con la biblioteca EEPROM).

- **Entrada y salida**

Cada uno de los cincuenta y cuatro pines digitales del Mega se puede usar como entrada o salida, usando las funciones pinMode () digitalWrite () y digitalRead (). Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50 kOhms. Además, algunos pines tienen funciones especializadas:

- Serie: 0 (RX) y 1 (TX); Serial 1: 19 (RX) y 18 (TX); Serial 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX). Se usa para recibir (RX) y transmitir (TX) datos en serie TTL. Los pines 0 y 1 también están conectados a los pines correspondientes del chip serie FTDI USB a TTL.
- Interrupciones externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3) y 21 (interrupción 2). Estos pines se pueden configurar para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor. Vea la función attachInterrupt () para más detalles.
- PWM: 2 a 13 y 44 a 46. Proporcionan salida PWM de 8 bits con la función analogWrite ().
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estos pines admiten la comunicación SPI, que, aunque proporcionada por el hardware subyacente, no

se incluye actualmente en el lenguaje Arduino. Los pines SPI también se dividen en el encabezado ICSP, que es físicamente compatible con Duemilanove y Diecimila.

- LED: 13. Hay un LED integrado conectado al pin digital 13. Cuando el pin tiene un valor ALTO, el LED está encendido, cuando el pin está BAJO, está apagado.
- I2C: 20 (SDA) y 21 (SCL). Admite la comunicación I 2 C (TWI) utilizando la biblioteca Wire (documentación en el sitio web de Wiring). Tenga en cuenta que estos pines no se encuentran en la misma ubicación que los pines I 2 C en Duemilanove o Diecimila.

El Mega tiene 16 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto, miden desde tierra a 5 voltios, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la función analogReference (). (Arduino, 2018)

Hay un par de otros pines en el tablero:

- AREF. Voltaje de referencia para las entradas analógicas. Usado con analogReference ().
- Reiniciar. Normalmente se usa para agregar un botón de reinicio a los escudos que bloquean el que está en el tablero.

- **Comunicación**

El Arduino Mega tiene una serie de instalaciones para comunicarse con una computadora, otro Arduino u otros microcontroladores. El ATmega1280 proporciona cuatro UART de hardware para comunicación serial TTL (5V). Un FTDI FT232RL en la placa canaliza uno de estos sobre USB y los controladores FTDI (incluidos con el software Arduino) proporcionan un puerto virtual para el software en la computadora. El software Arduino incluye un monitor serie que permite el envío de datos textuales simples hacia y desde la placa Arduino. Los LED RX y TX de la placa parpadearán

cuando los datos se transmitan a través del chip FTDI y la conexión USB a la computadora (pero no para la comunicación serial en los pines cero y uno). (Arduino, 2018)

Una biblioteca de SoftwareSerial permite la comunicación serial en cualquiera de los pines digitales de Mega.

El ATmega1280 también soporta I2C (TWI) y la comunicación SPI. El software Arduino incluye una biblioteca Wire para simplificar el uso del bus I2C

- **Programación**

Arduino Mega se puede programar con el software Arduino

El ATmega1280 en el Arduino Mega viene pregrabado con un gestor de arranque que le permite cargar un nuevo código sin el uso de un programador de hardware externo. Se comunica utilizando el protocolo original STK500. (Arduino, 2018). También puede omitir el gestor de arranque y programar el microcontrolador a través del encabezado ICSP (Programación serial en circuito).

- **Restablecimiento automático (software)**

En lugar de requerir una pulsación física del botón de reinicio antes de una carga, el Arduino Mega está diseñado de manera que permite su reinicio mediante un software que se ejecuta en una computadora conectada. Una de las líneas de control de flujo de hardware (DTR) del FT232RL está conectada a la línea de restablecimiento del ATmega1280 a través de un condensador de 100 nano faradios. Cuando esta línea se afirma (se toma bajo), la línea de reinicio cae lo suficiente como para restablecer el chip. El software Arduino utiliza esta capacidad para permitirle cargar código simplemente presionando el botón de carga en el entorno Arduino. Esto significa que el gestor de arranque puede tener un tiempo de espera más corto, ya que la disminución de DTR puede coordinarse bien con el inicio de la carga. (Arduino, 2018)

Esta configuración tiene otras implicaciones. Cuando Mega está conectada a una computadora con Mac OS X o Linux, se restablece cada vez que se realiza una conexión desde el software (a través de USB). Durante el siguiente medio segundo más o menos, el gestor de arranque se está ejecutando en el Mega. Mientras está programado para ignorar datos mal formados (es decir, cualquier cosa además de una carga de código nuevo), interceptará los primeros bytes de datos enviados a la placa después de que se abra una conexión. Si un boceto que se ejecuta en la placa recibe una configuración de una sola vez u otros datos cuando se inicia por primera vez, asegúrese de que el software con el que se comunica espera un segundo después de abrir la conexión y antes de enviar esta información. (Arduino, 2018)

El Mega contiene un rastro que se puede cortar para desactivar el reinicio automático. Las almohadillas a cada lado de la traza pueden soldarse juntas para volver a habilitarlo. Está etiquetado como "RESET-EN". También puede desactivar el restablecimiento automático conectando una resistencia de 110 ohmios desde 5 V a la línea de reinicio; mira este hilo del foro para más detalles.

- **Protección de sobre intensidad USB**

El Arduino Mega tiene un polifuse reseteable que protege los puertos USB de tu computadora contra cortos y sobrecorrientes. Aunque la mayoría de las computadoras proporcionan su propia protección interna, el fusible proporciona una capa adicional de protección. Si se aplica más de 500 mA al puerto USB, el fusible romperá automáticamente la conexión hasta que se elimine el cortocircuito o la sobrecarga. (Arduino, 2018)

- **Características físicas y compatibilidad del escudo**

La longitud y el ancho máximos de la Mega PCB son de 4 y 2.1 pulgadas respectivamente, con el conector USB y el conector de alimentación extendiéndose

más allá de la dimensión anterior. Tres orificios para tornillos permiten que la placa se una a una superficie o caja. Se debe tener en cuenta que la distancia entre los pines digitales 7 y 8 es de 160 mil (0.16 "), no es un múltiplo par del espaciado de 100 mil de los otros pines.

El Mega está diseñado para ser compatible con la mayoría de los escudos diseñados para Diecimila o Duemilanove. Los pines digitales 0 a 13 (y los pines adyacentes AREF y GND), las entradas analógicas 0 a 5, el encabezado de alimentación y el encabezado ICSP están todos en ubicaciones equivalentes. Además, el UART principal (puerto serie) está ubicado en los mismos pines (0 y 1), al igual que las interrupciones externas 0 y 1 (pines 2 y 3, respectivamente). SPI está disponible a través del encabezado de ICSP en Mega y Duemilanove / Diecimila. Tenga en cuenta que I 2 C no se encuentra en los mismos pines en el Mega (20 y 21) como el Duemilanove / Diecimila (entradas analógicas 4 y 5) (Arduino, 2018).

CAPÍTULO 2. DESARROLLO DEL TRABAJO

En este capítulo se da a conocer el procedimiento necesario para la creación de un reconocedor de voz en el IDE NetBeans 8.1

2. DESARROLLO DEL TRABAJO

El reconocimiento digital de comandos de voz se lleva a cabo por medio de diferentes pasos, los cuales se ejemplifican en el siguiente diagrama según Peinado (1994) en su tesis doctoral "*Seleccion y estimacion de parametros en sistemas de reconocimiento de voz basados en modelos ocultos de Markov*".

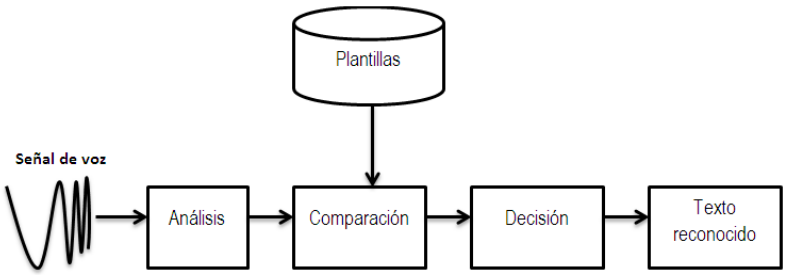

Figura # 20 Esquema básico de reconocimiento según Peinado (Peinado, 1994).

Con base en lo desarrollado por Peinado (1994), se desarrolló el siguiente diagrama (véase Figura # 21), el cual ejemplifica el funcionamiento particular del reconocedor de comandos de voz del que se habla en este proyecto.

2.1. Desarrollo en NetBeans

Para desarrollar una aplicación de reconocimiento de comandos de voz es necesario primero contar con el IDE especial para desarrollarla, en este caso el IDE es NetBeans en su versión 8.1, así mismo es necesario contar con una versión de Java actual, ya que es el lenguaje en el que se desarrolla esta aplicación, en este proyecto la versión de Java aplicada es 1.8.0_111.

Tanto el IDE como la versión de Java requerida se encuentra disponible de manera gratuita desde la página oficial de cada uno de éstos https://www.Java.com/es/download/ para Java y https://NetBeans.org/downloads/ para NetBeans, al descargar la versión de NetBeans ofrece la opción de descargar la versión aplicable de diferentes lenguajes de programación compatibles con dicho IDE como se puede observar en la Figura # 22.

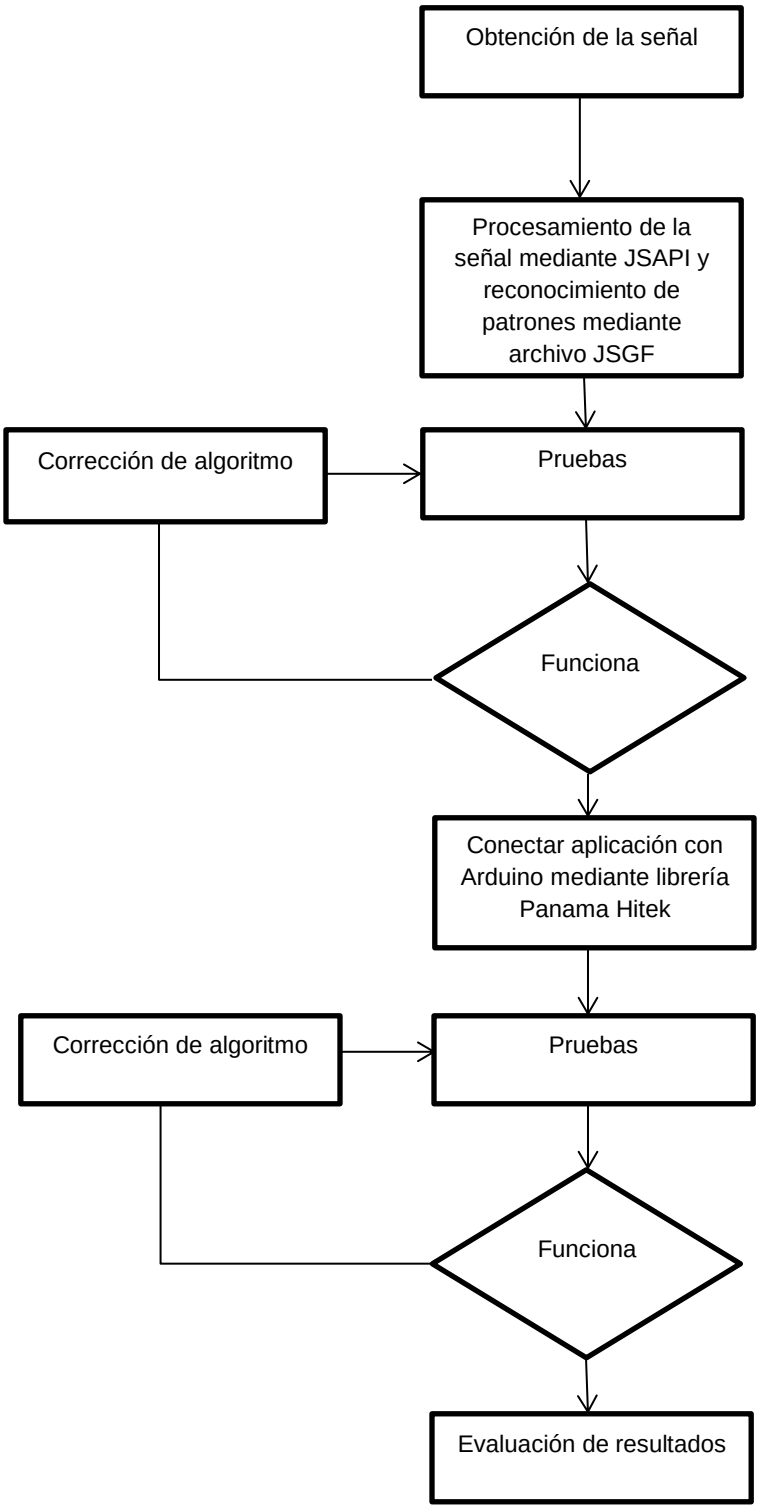

Figura # 21 Esquema de reconocimiento de comandos de voz. Elaboración Propia

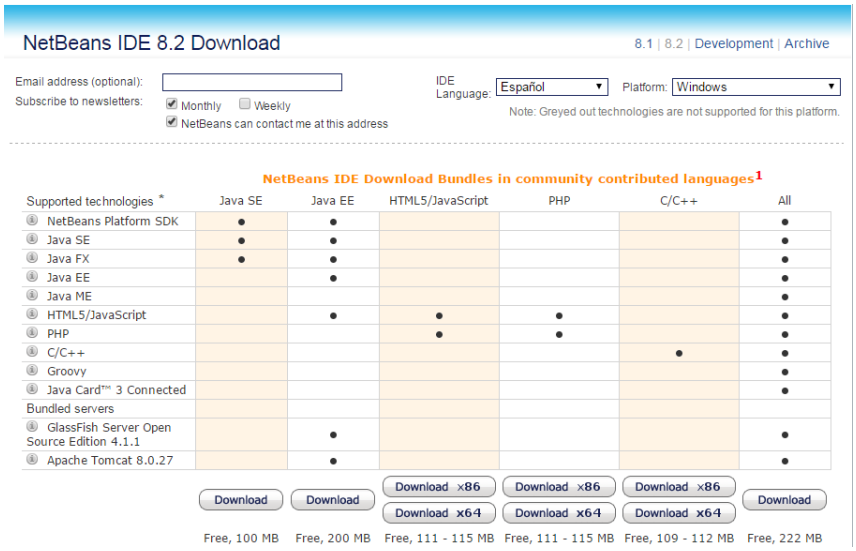

Figura # 22 Página de descarga de NetBeans (2017). (NetBeans)

Ya que se cuenta con el IDE y con la versión de Java necesaria se utiliza JSAPI, ya que por sí solo Java no realiza el reconocimiento de comandos de voz. Y aunque en principio es una API desarrollada por Java, existen versiones mejoradas de la misma por terceros, como el caso en concreto de este trabajo de tesis, en el cual se usa una versión de Java Speech desarrollada por Cloud Garden (CGJSAPI).

En esta etapa del proyecto es necesario hacer una aclaración, ya que la compañía que dio vida a la versión de Java Speech utilizada para este trabajo la ofrecía de manera gratuita dentro de su sitio web, éste por motivos desconocidos ya no existe, por lo cual dicha API debe ser descargada de alguna otra dirección web. Aun así, este software creado por Cloud Garden se encuentra sólo disponible de manera gratuita si el proyecto a desarrollar se realiza con fines no lucrativos, ya que, si este genera ganancias, es necesario pagar una licencia a los creadores de dicha API.

Ya que se cuenta con dicha API es necesario incorporarla a NetBeans, esto se realiza de la siguiente manera, en la pantalla principal de NetBeans>tools>libraries, se muestra de manera gráfica en la Figura # 23.

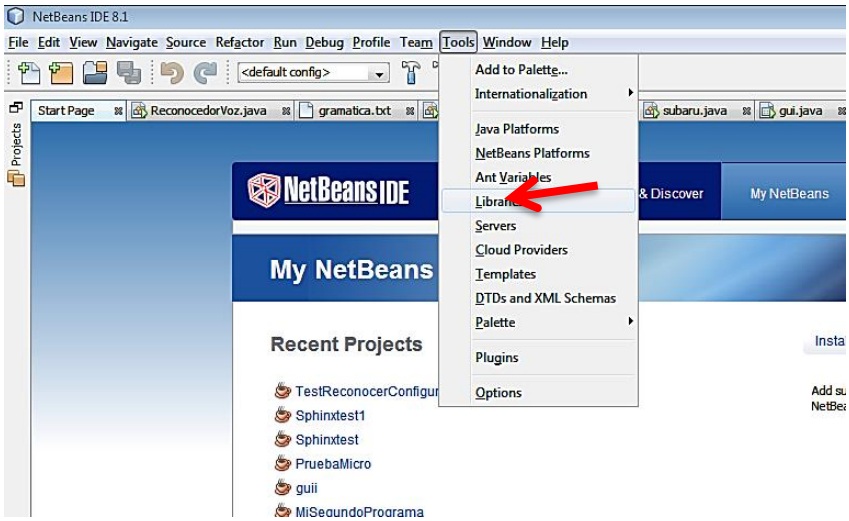

Figura # 23 Menú desplegable del menú Tools de NetBeans 8.1. Figura Propia

Una vez dentro del menú se selecciona la opción “New library” y es ahí donde se nombra la librería como se considere conveniente, para el proyecto (véase Figura # 24).

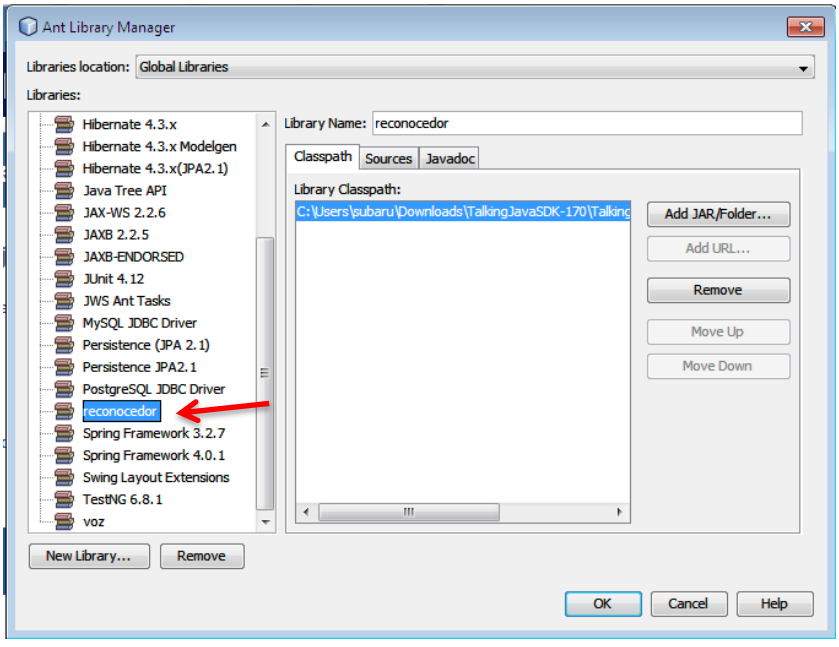

Figura # 24 Menú desplegable de la opción New Library del IDE NetBeans 8.1. Figura Propia

Para este proyecto en específico se consideró adecuado nombrar dicha librería como “reconocedor” como se demuestra en la Figura # 24 ya que ésta es la función principal que realiza la aplicación.

Inmediatamente se debe dar clic sobre la opción “Add JAR/Folder” para seleccionar la ruta del archivo .jar que integraremos a esta nueva librería (véase Figura # 24). En el caso específico de este proyecto se encuentra en la ruta “C:\Users\subaru\Downloads\TalkingJavaSDK-170\TalkingJavaSDK-170\packet\cgjsapi.jar”.

Esta ruta corresponde a:

- “C:” es la partición o disco principal
- “Users” es el tipo de usuario (usuario o administrador) el cual se encuentra seguido inmediatamente del nombre del usuario en sesión.
- “subaru” nombre del usuario con sesión activa
- “Downloads” nombre de la carpeta o repositorio donde se ubican por default archivos provenientes de descargas de internet, a excepción de que al momento de realizar la descarga se especifique lo contrario
- “TalkingJavaSDK-170” el primero hace referencia al nombre de una carpeta contenedora ubicada dentro de “Downloads”
- “TalkingJavaSDK-170” el segundo hace referencia a una carpeta contenedora ubicada dentro de la primer carpeta de nombre “TalkingJavaSDK-170”
- “packet” es el nombre de otra carpeta contenedora ubicada dentro de la segunda carpeta de nombre “TalkingJavaSDK-170”
- “cgjsAPI.jar” es el nombre del archivo .jar que agregaremos a nuestra librería.

Ver de manera gráfica en la Figura # 25.

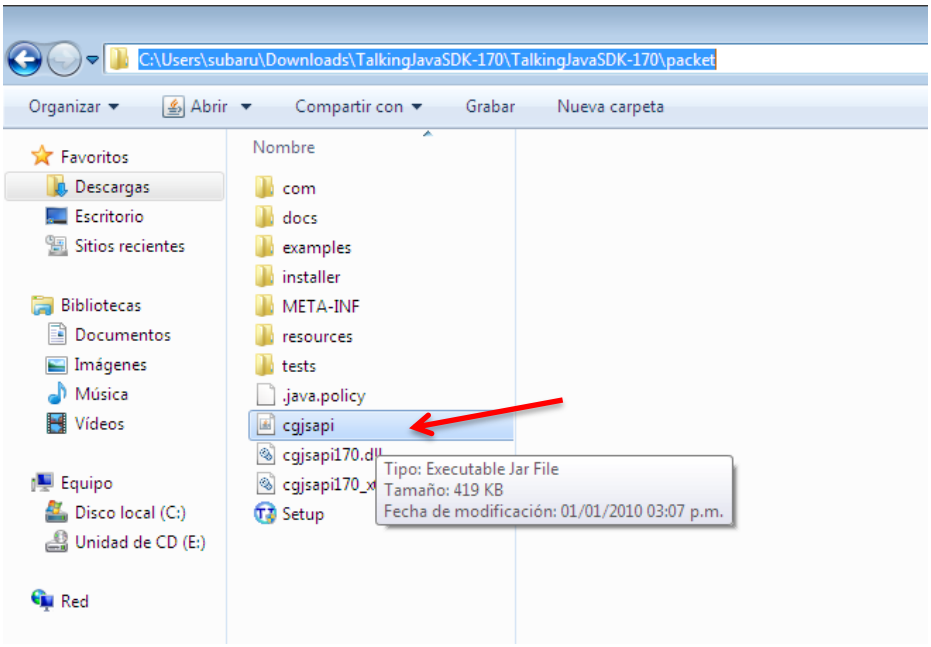

Figura # 25 Ruta del archivo cgjsapi.jar. Figura Propia

Esto sirve solo para agregar el archivo cgjsapi.jar a NetBeans, ya que como se mencionó previamente, dentro del repositorio general de este IDE no se cuenta con una librería para realizar la tarea de reconocimiento de voz, pero todavía es imprescindible agregar dicha librería al proyecto qué hará uso de esta, y con el cual se manipulara el actuador eléctrico.

Ahora bien, para agregar la librería al proyecto es preciso ubicar el mouse sobre el nombre de nuestro proyecto en la ventana “projects”, la cual se encuentra ubicada en la parte izquierda de la pantalla de NetBeans 8.1; en dado caso de que esta ventana no aparezca, también se puede desplegar presionando las teclas “ctrl”+”1” o bien hacer clic directamente sobre el menú “Window” y buscar la opción “projects” la cual en la version 8.1 de NetBeans se encuentra al principio de la lista dentro de este menú (véase Figura # 26).

Una vez ubicada dicha opción se hace clic derecho sobre el nombre de nuestro proyecto y luego clic sobre propiedades, lo cual desplegará un menú emergente en el cual se selecciona la opción “Libraries” (véase Figura # 27).

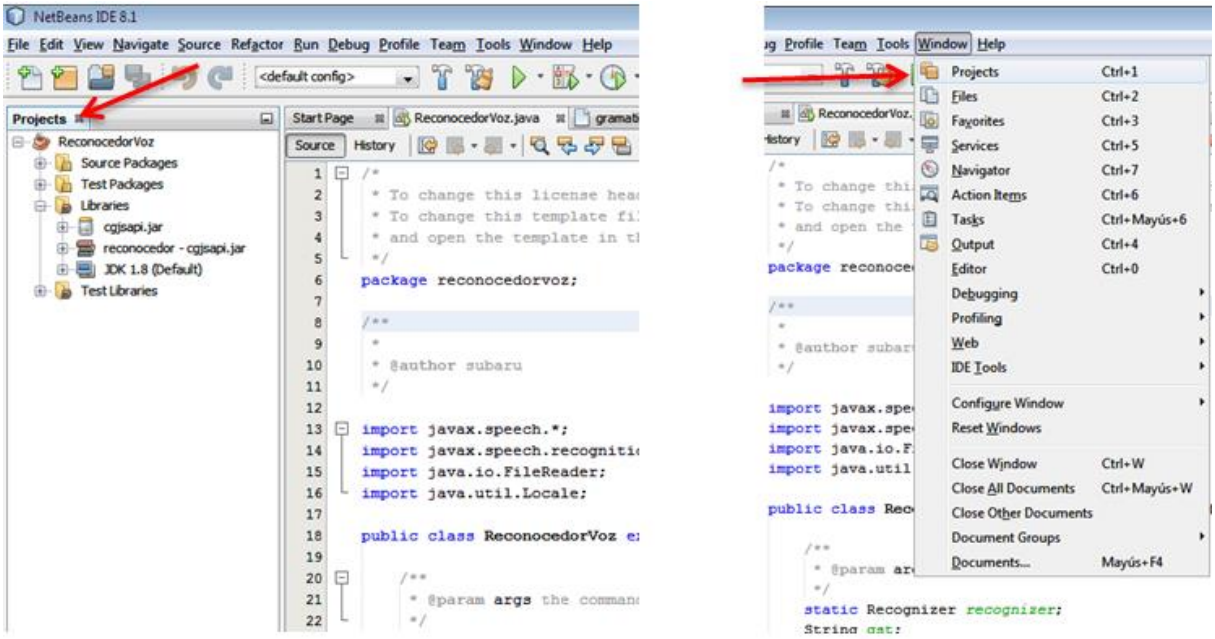

Figura # 26 Maneras de acceder a "Projects" desde la pantalla de NetBeans 8.1. Figura Propia

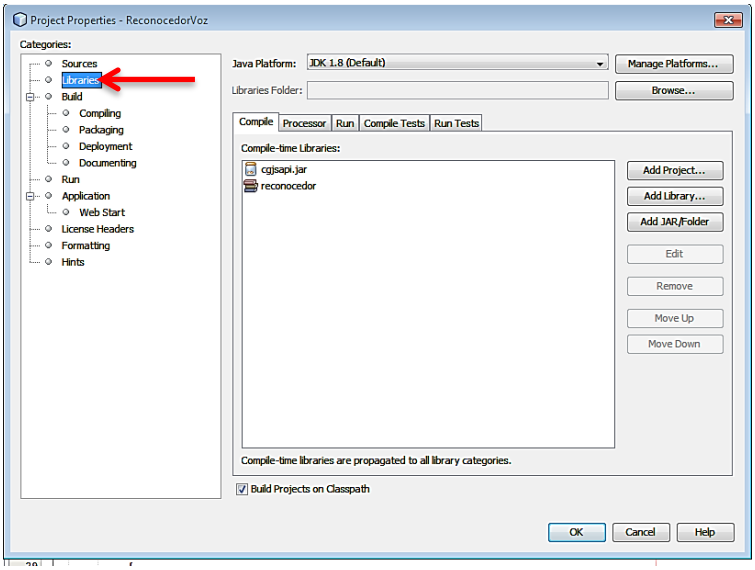

Figura # 27 Menú emergente de las propiedades de nuestro proyecto. Figura Propia

Una vez dentro del menú que acaba de aparecer se debe hacer clic sobre la opción “Add Library” la cual hará aparecer un nuevo menú donde se encuentra una lista de librerías, incluida la librería que se creó con antelación especialmente para reconocer los comandos de voz (véase Figura # 28).

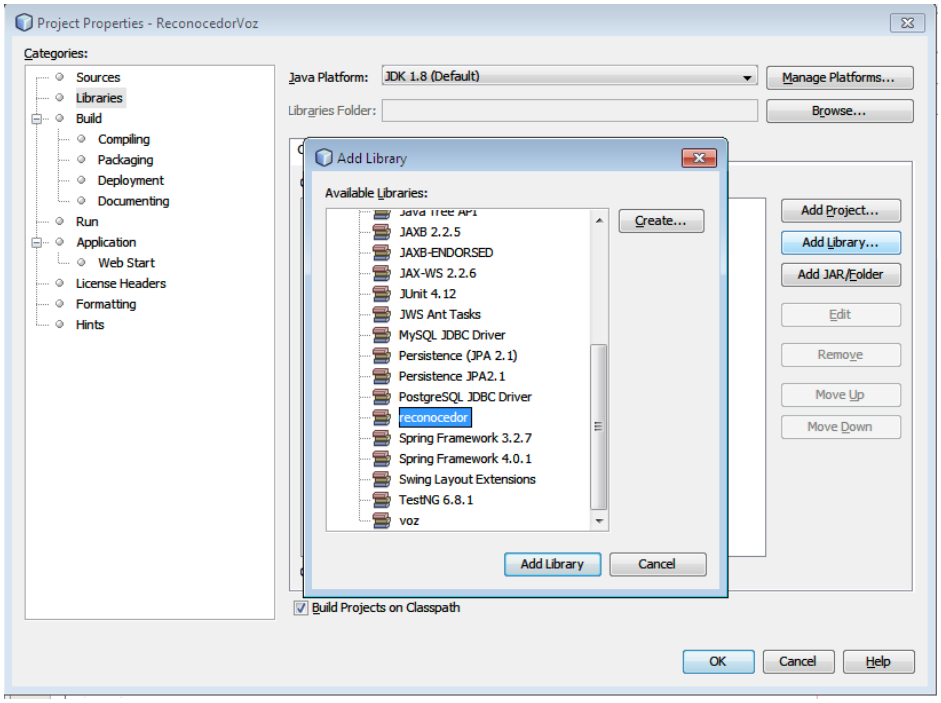

Figura # 28 Menú emergente de la opción "Add Library" en NetBeans 8.1. Figura Propia

Una vez encontrada la opción deseada, se selecciona y se da clic sobre el botón “Add Library” y posteriormente sobre “ok” y con esto la librería creada a partir de cgjsapi.jar estaría incluida dentro del proyecto de manera exitosa. Lo cual puede corroborarse haciendo clic sobre la ventana “projects” y desplegando la opción con el proyecto sobre el que se ésta trabajando e inmediatamente la opción de “Libraries” en la cual deberá mostrarse la librería que se acaba de agregar (véase Figura # 29).

Una vez configurada la librería que se utilizará es posible comenzar con la creación del código de este proyecto, lo primero es importar la librería Javax.speech.

Javax.speech es un paquete que define el comportamiento de todos los motores de voz, es decir de aquellos capaces de reconocer comandos de voz como de los que están enfocados en crear síntesis de voz.

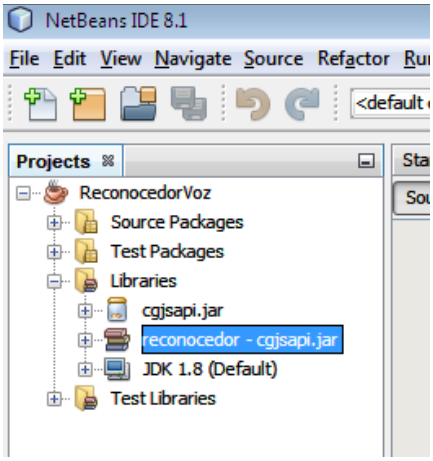

Figura # 29 Librerías contenidas dentro del proyecto. Figura Propia

Javax.speech es un paquete que define el comportamiento de todos los motores de voz, es decir de aquellos capaces de reconocer comandos de voz como de los que están enfocados en crear síntesis de voz.

El paquete de Javax.speech del API Java Speech define un software de representación abstracta de un motor de voz. Ahora bien, un motor de voz, es un término genérico utilizado para hacer referencia tanto a sistemas diseñados para manejar la entrada como la salida de voz. El caso de los sistemas de identificación del locutor, así como los encargados de reconocer comandos de voz son ejemplos de motores de voz.

En este caso en específico y como se pretende crear una aplicación capaz de reconocer comandos de voz, se hará especial énfasis en la parte de Javax.speech capaz de realizar dicha tarea y se dejará de lado la encargada de realizar la síntesis de voz.

Para poder realizar el reconocimiento de voz, Javax.speech hace uso del paquete Javax.speech.recognition el cual define capacidades específicas para realizar dicha tarea.

2.2. Archivo JSGF

Para que el reconocimiento de voz funcione de manera correcta es necesario contar con un formato de gramática, un archivo JSGF que dicte las reglas gramaticales, es decir las palabras o frases que podrán ser reconocidas.

Como el archivo JSGF es un archivo de texto plano, éste puede ser creado dentro de cualquier editor de texto. En este caso se creó dentro de la aplicación de “block de notas” de Windows 7 (véase Figura # 30).

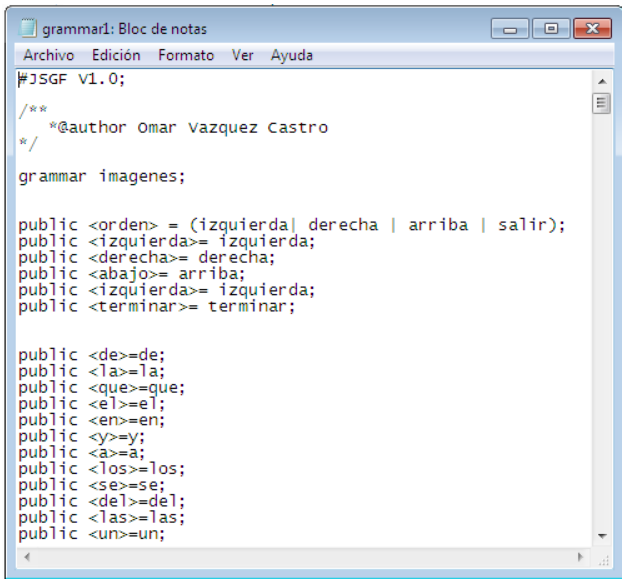

Figura # 30 Archivo JSGF. Figura Propia

Como este archivo se crea con fines de manipular un actuador por medio de comandos de voz, se aprecia cómo es que no existen combinaciones, agrupamientos, sino solo reglas cuya expansión es una sola palabra, o para los fines de este proyecto, un simple comando.

La ruta en la que se guardó el archivo es en el disco C:// como se aprecia en la Figura # 31.

Es importante saber, que tanto la ruta como el nombre del archivo son utilizados dentro del algoritmo desarrollado en NetBeans. También es importante conocer que, tanto la ruta como el nombre del archivo pueden variar según el desarrollador, y la aplicación funcionará sin complicaciones. Este archivo JSGF debe estar conectado con el código en Java, y esto se logra por medio de las librerías antes mencionadas.

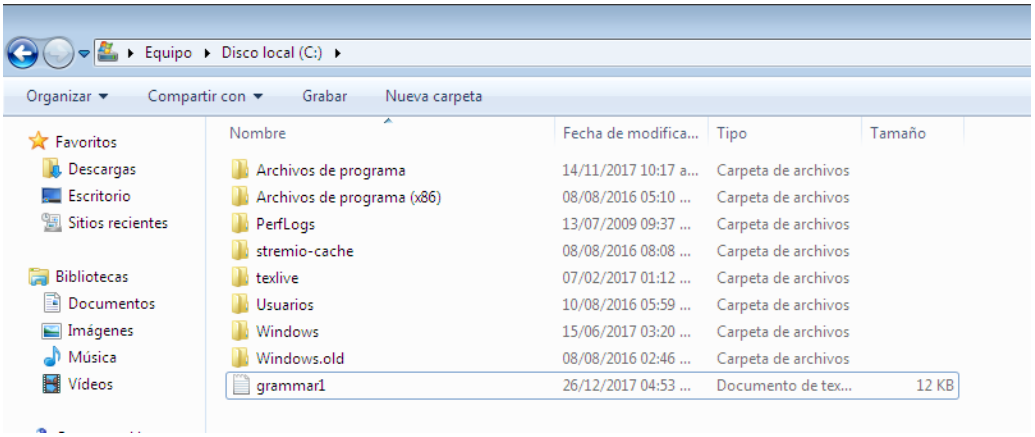

Figura # 31 Ruta del archivo JSGF. Figura Propia.

2.3. Programación de la placa Arduino.

También es imprescindible configurar una conexión entre la placa Arduino y la aplicación en Java, para poder manipular el actuador eléctrico, lo cual se logra incluyendo la librería Panama Hitek dentro del proyecto en NetBeans.

Para programar la placa Arduino Mega se puede descargar el IDE directamente desde la página oficial de Arduino, en la sección de software de manera gratuita (véase Figura # 32).

Una vez instalado se puede comenzar a escribir el código, sin la necesidad de añadir librerías, o paquetes como en NetBeans. El código es bastante sencillo, se trata

únicamente de especificar que Pin de la placa Arduino se utilizará como salida, el puerto serial de la computadora al cual se conectará dicha placa, y cuál será la velocidad de comunicación.

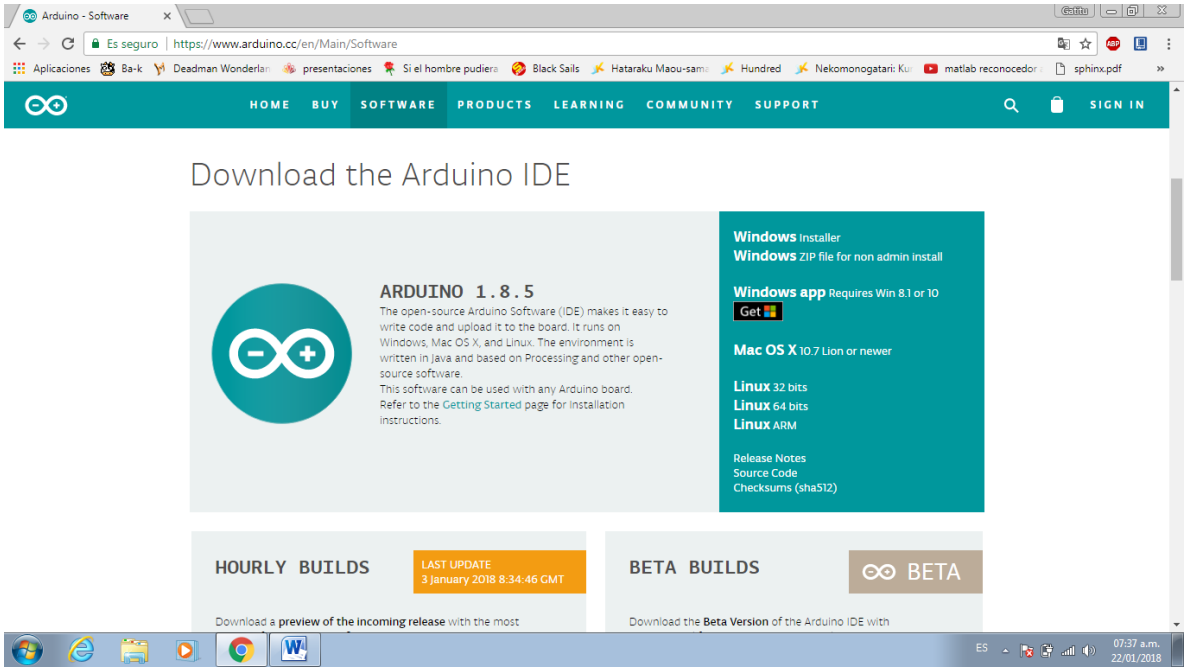

Figura # 32 Página de descarga del IDE de la Plataforma Arduino

Para poder visualizar en que puerto se conectó la placa Arduino se debe consultar la ruta como lo muestra la Figura # 33:

$$inicio > Panel\ de\ control > Administrador\ de\ dispositivos > Puertos\ (COM\ y\ LPT)$$

Al mismo tiempo, y como el código desarrollado para Arduino se conecta con el desarrollado en NetBeans es importante recoger el valor que la aplicación de código Java recibió mediante el reconocimiento de voz, y si es alguno de los comandos programados para encender el motor enviar un “1” como señal de salida, si es un comando para apagar el motor enviar un “0” como señal de salida.

Una vez realizado el código, es hora de grabarlo en la placa Arduino, para esto basta con guardarlo, y posteriormente hacer clic sobre la opción “subir” (véase Figura # 35),

para esta opción es necesario que la placa Arduino haya sido previamente conectada a la computadora (véase Figura # 34).

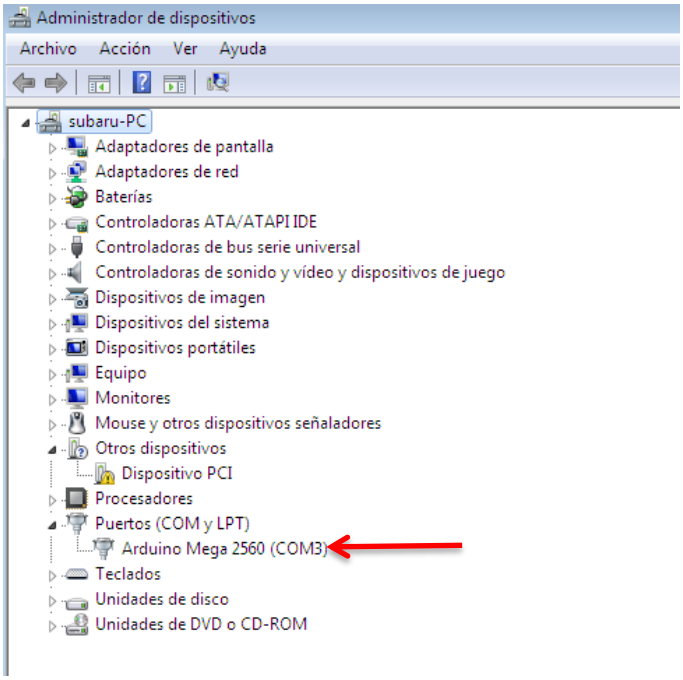

Figura # 33 Administrador de Dispositivos. Figura Propia

Figura # 34 Placa Arduino conectada a una computadora. Figura Propia

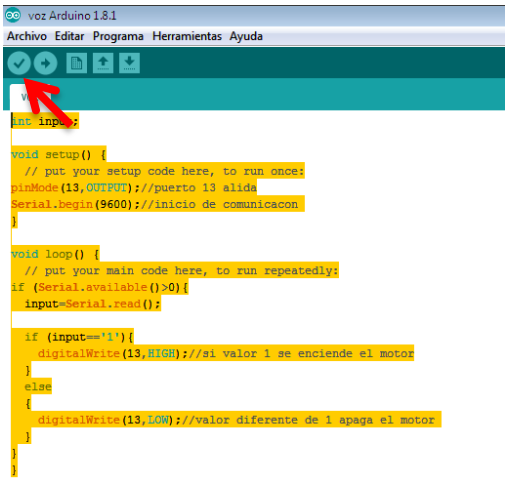

Figura # 35 Grabar código en placa Arduino Mega. Figura Propia.

2.4. Conexión Física de componentes

Al momento de conectar la placa Arduino mega con la PC para efectuar las pruebas de efectividad de la aplicación ya en conjunto con el motor surge la necesidad de crear un pequeño circuito eléctrico, ya que la placa Arduino no genera el voltaje suficiente para activar el motor por sí sola, el circuito contiene:

- 1 transistor bjt el cual servirá para regular el voltaje que llega al motor
- 1 resistencia de 330 ohms
- 1 pila de 9 v la cual servirá para alimentar el moto-reductor

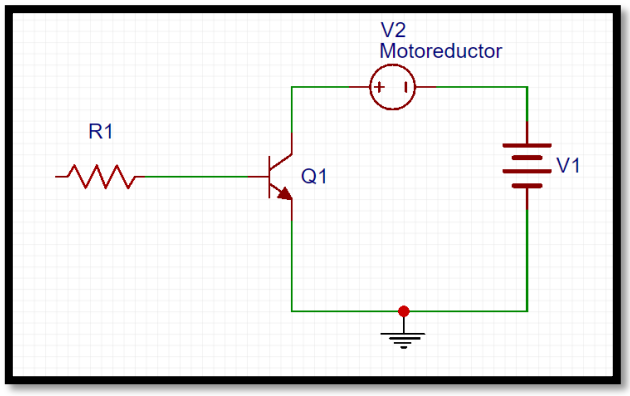

Figura # 36 Circuito eléctrico que regula el voltaje saliente de Arduino para poder encender un moto-reductor. Figura Propia

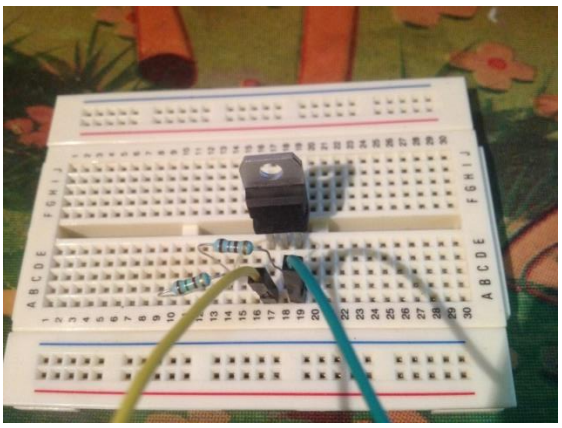

Figura # 37 Imagen del circuito eléctrico ya ensamblado. Figura Propia

Del otro lado de la resistencia estará conectada la salida 13 de la placa Arduino mega, la cual fue previamente configurada para funcionar en conjunto con la aplicación Java.

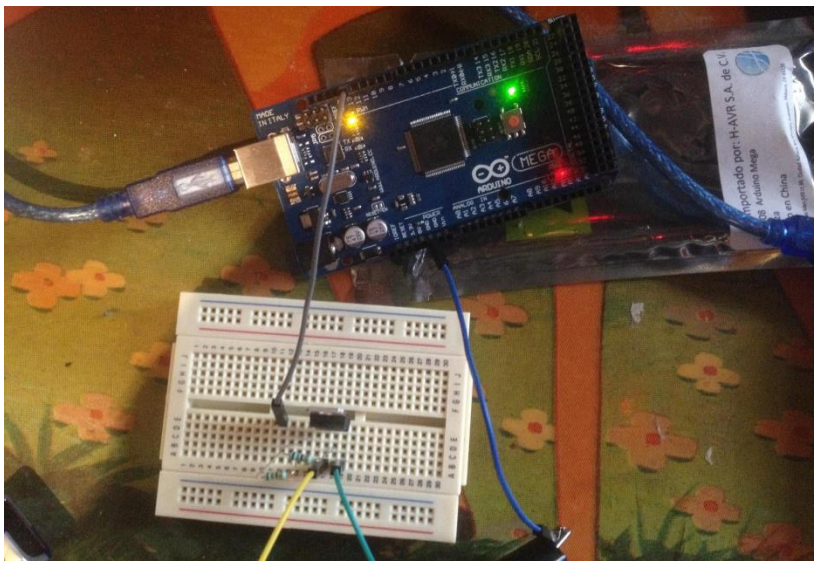

Figura # 38 Placa Arduino Mega conectada al circuito regulador de voltaje. Figura Propia

La función de este circuito es suministrar y regular el voltaje que llega al motor cada que la aplicación reconoce el comando para encender o apagar dicho motor

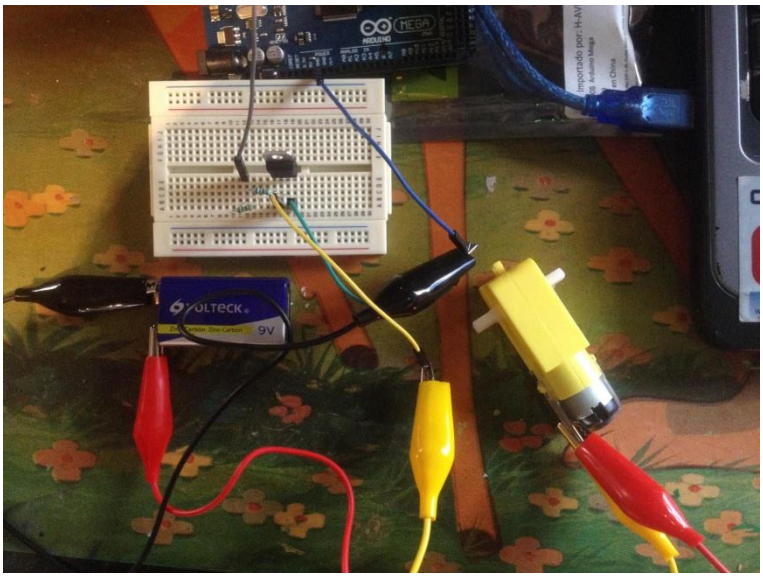

Figura # 39 **Conexión de Placa Arduino Mega, Circuito regulador de voltaje, Motoreductor y Computadora. Figura Propia**

Cabe señalar que tanto la aplicación Java, el archivo de gramática (JSGF), la placa Arduino Mega, el circuito regulador de voltaje, y el motoreductor, funcionan como un solo sistema, y no como módulos por separado, es decir que, sí falta alguno, o alguno llega a tener un error de funcionamiento, todo fallará.

2.5. Interpretación de resultados

Para este proyecto, los resultados se interpretaron en tres fases diferentes, cada una correspondiente a una fase del desarrollo de la aplicación diferente.

En la fase de creación de la aplicación se analizó la efectividad del algoritmo mediante la aparición de diferentes imágenes en la pantalla del ordenador, acorde a diferentes comandos proporcionados por un locutor, así como la aparición de las palabras reconocidas por la aplicación (véase Figura # 41).

Como se aprecia en la Figura # 40, al inicio de la aplicación aparecen mensajes de las dos diferentes aplicaciones que se utilizan, tanto Panama Hitek, como CGJSAPI.

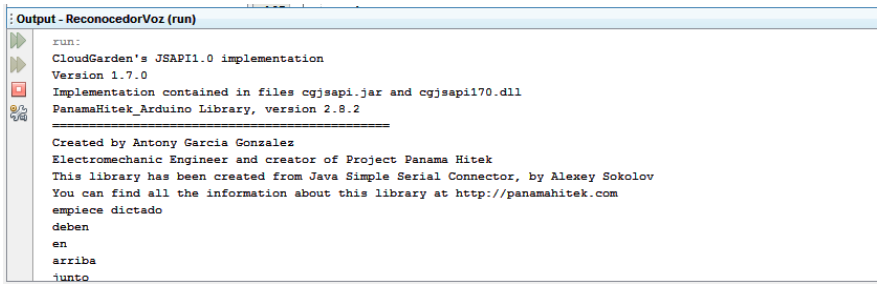

Figura # 40 Consola de salida de Aplicación de reconocimiento de comandos de voz. Figura Propia

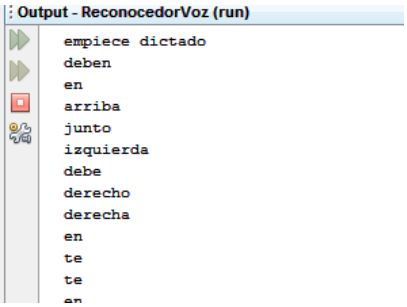

Figura # 41 Consola de salida de Aplicación de reconocimiento de comandos de voz después de haber reconocido algunas palabras. Figura Propia

En la segunda fase, la conexión entre la aplicación y la placa Arduino, se interpretaron los resultados por medio de un led, el cual se encendía o apagaba según el comando proporcionado por un locutor.

Por último, en la tercera fase, después de conectar la placa Arduino al circuito en la figura anterior se interpretó el funcionamiento de la aplicación con ayuda de un motoreductor, el cual se encendía o apagaba según el comando proporcionado por un locutor.

CAPITULO 3. RESULTADOS Y CONCLUSIONES.

En este capítulo se exponen y se hace un análisis de los resultados obtenidos mediante las pruebas efectuadas al algoritmo de reconocimiento de comandos de voz.

3. RESULTADOS Y CONCLUSIONES

3.1 Resultados

Las pruebas se dividen en dos etapas:

1. Pruebas del reconocimiento de palabras emitidas por un locutor de habla hispana

2. Pruebas del reconocimiento de comandos de voz para manipular un actuador eléctrico

En ambas pruebas el locutor interactuara de manera directa con la aplicación

Para la primera etapa, se hicieron pruebas con cinco personas de diferentes edades y género, en estas pruebas se pidió a los locutores que leyeran una serie de 50 palabras, y se registró el desempeño del reconocedor en la Tabla # 1.

Palabra	Persona 1	Persona 2	Persona 3	Persona 4	Persona 5
Hemos	1	1	1	1	1
Trata	1	1	1	1	1
Mal	1	1	1	1	1
Algún	0	0	0	0	0
Tuvo	1	1	1	1	1
Respecto	1	1	1	1	1
Semana	0	1	1	1	1
Varios	1	1	1	1	1
Real	1	1	1	1	1
Proyecto	1	1	1	1	1
Mercado	1	1	1	1	1
Mayoría	0	0	0	0	0
Orden	1	1	1	1	1
Español	0	0	0	0	0
Programa	1	1	1	1	1
Palabras	1	1	1	1	1
Internacional	1	1	1	1	1
Segunda	1	1	1	1	1

Empresa	1	1	1	1	1
Libro	1	1	1	1	1
Igual	1	1	1	1	1
Persona	0	1	0	1	1
Últimos	0	0	0	0	0
Condiciones	1	1	1	1	1
Acción	0	0	0	0	0
Policía	0	0	0	0	0
Puerta	1	0	0	1	1
Fuerza	1	1	0	1	1
Interior	1	1	0	1	1
Tampoco	1	1	1	1	1
Música	0	0	0	0	0
Ningún	0	0	0	0	0
Hubiera	1	1	1	1	1
Saber	1	1	0	1	0
Presencia	1	1	1	1	1
Comisión	0	0	0	0	0
Servicio	1	1	1	1	1
Última	0	0	0	0	0
Minutos	1	1	1	1	1
Producción	0	0	0	0	0
Camino	1	1	1	1	1
Dirección	0	0	0	0	0
Papel	1	1	0	1	1
Diferentes	1	1	1	1	1
Libertad	1	1	0	1	1
Relaciones	1	1	0	1	1
Espacio	1	1	0	1	1
Medios	1	1	0	1	1
Terminar	1	1	1	1	1

Tabla # 1 Desempeño de reconocedor en primera etapa. Creación Propia.

Los datos de la Tabla # 1 representan los aciertos y los errores del reconocedor, siendo que los “1” representan un acierto, y los “0” representan un error.

Como resultado del análisis se obtiene que el reconocedor es efectivo, reconociendo un setenta por ciento de las palabras analizadas.

Figura # 42 Representación en porcentajes de la efectividad de la aplicación de reconocimiento de comandos de voz. Figura Propia

Los errores en el reconocimiento de palabras se deben a las diferentes maneras de pronunciar que tiene cada persona, ya que como el este reconoce fonemas, es posible que la pronunciación de algunas palabras resulte confusa para el reconocedor, al mismo tiempo que CGJSAPI tiene conflictos con la letras “ñ”, “x” (por sus diferentes variaciones de pronunciación en el Español hablado en México), así como el hecho de que el reconocedor al momento de declarar las reglas gramaticales, y las palabras que podrán ser reconocidas no permite el ingreso de acentos, los cuales son necesarios dentro del habla hispana.

3.2 Conclusiones

Con base en lo anterior es posible concluir que el reconocimiento de comandos de voz es una herramienta muy poderosa, con un sinfín de aplicaciones ya que todo lo que se manipula con las manos es posible manipularlo mediante comandos de voz, con sus debidas modificaciones.

Gracias al avance de la tecnología cada vez es posible encontrar el reconocimiento de voz en diversos aspectos de la vida cotidiana, debido a esto no cabe duda de que la explotación de esta tecnología siga en crecimiento con el paso del tiempo.

Ahora bien, hablando en concreto del desarrollo de esta aplicación cabe destacar:

- Encontrar la herramienta (API) para crear la aplicación se vuelve una tarea compleja, ya que la documentación acerca del tema es muy poca, y cuando se encuentra dicha herramienta, utilizarla es igual de complejo, por la misma razón, la falta de documentación.
- El archivo JSGF encargado de establecer las reglas sintácticas para el trabajo de la aplicación, aunque se dice que es de uso universal, carece de funcionalidad para el uso del mismo dentro de una lengua como la hablada dentro de México, esto debido a:
    - Los diferentes sonidos que puede tener la letra “x” dependiendo de la palabra,
    - No soporta las palabras que contienen la letra “ñ” puesto que su sonido no existe dentro del habla anglosajona.
    - Así como sucede con las palabras que contienen la letra “ñ” lo mismo sucede con las palabras acentuadas, puesto que el idioma ingles no utiliza acentos, el archivo JSGF no reconoce dichos caracteres.
- Hablando del archivo JSGF, con base en las pruebas realizadas durante el desarrollo de la aplicación cabe señalar que entre mayor sea el número de palabras contenidas dentro de las reglas gramaticales, mejor será el desempeño

de la aplicación, puesto que tendrá una mayor base de datos con la cual realizar la comparación de lo emitido por el locutor.

- Es importante mencionar que la aplicación tiene un alto índice de error con palabras de una sola silaba, ya que al basarse en reconocimiento de fonemas, palabras de una sola silaba que contengan la misma vocal emiten una señal muy similar.
- Para conectar la aplicación con Arduino basta con añadir la librería “PanamaHitek” y hacer unas modificaciones sencillas al código, la parte importante de esta conexión ésta en el aparato que se desee manipular mediante los comandos de voz, puesto que como en este caso en particular, Arduino mega no suministra suficiente voltaje para activar el motor por lo cual es necesario incluir un pequeño circuito a la salida del mismo.
- Por último, es de suma importancia conocer que para el funcionamiento de esta aplicación se crearon:
    - Código en el IDE NetBeans mediante código Java,
    - Un archivo de texto plano (JSGF) para establecer las reglas sintácticas con las que trabajaría la aplicación, así como para establecer todas las palabras que el algoritmo es capaz de reconocer,
    - Código en el IDE específico para Arduino Mega, así como su respectivo almacenamiento en dicha placa
    - Y un circuito para regular el voltaje saliente de la placa Arduino mega.

  Sin embargo, no funcionan de manera independiente, sino que cada uno de los puntos anteriormente mencionados funciona como módulos dentro de un sistema que reconoce comandos de voz para manipular un actuador eléctrico. Por sí solos son inútiles.

TRABAJO A FUTURO

Este trabajo posee ciertas limitaciones, es decir, ciertos aspectos que, aunque no contravienen a la ejecución del algoritmo de reconocimiento de comandos de voz, si pudiesen ser eliminadas, se podría tener un algoritmo con una mayor eficiencia.

Como se mencionó con antelación en este trabajo se hace uso de una API de Java, específicamente JSAPI la cual sirve tanto para el reconocimiento de comandos de voz, como para la síntesis de la misma, pero esta aplicación solo hace uso de la primera parte, lo más fácil a desarrollar para un trabajo a futuro seria incluir la síntesis de voz, para que así la aplicación pueda interactuar con el usuario de una manera más natural, o humana por decirlo de alguna manera.

Ahora bien, el poder manipular un motor por medio de comandos de voz es solo la base para desarrollar mecanismos de control mucho más complejos, es decir, se podría adaptar el algoritmo para que controle diferentes componentes electrónicos, e incluso diferentes aparatos por medio de la voz.

Otra posibilidad que se ha explorado, y que posiblemente representaría la opción más compleja, es a partir del algoritmo de comandos de voz crear una aplicación que sea capaz de entablar una conversación con un ser humano, esto requeriría de:

- Modificar el algoritmo, para que sea capaz de sintetizar voz
- De igual manera se requeriría ampliar el archivo JSGF para que la base de palabras que el algoritmo puede reconocer sea suficiente para comprender cualquier conversación.
- Si bien lo anterior es importante para el desarrollo de esta propuesta, me parece imprescindible crear una red neuronal, y conectarla con la aplicación, puesto que la mayoría de conversaciones dependen de cierto contexto, el cual es imposible de comprender para una máquina, por ende, es necesario dotarla de esta capacidad

Es importante decir que esta aplicación tendría un sinfín de usos, puesto que no solo se trata de la creación de tecnología, sino que podría ayudar a personas que sufran de trastornos del habla, podría ser usado por personas que deseen aprender un idioma nuevo al tener estos con quien entablar una conversación, incluso podría ser usado como un tipo de compañía para personas que se encuentran en soledad.

ANEXOS

A. Código Java

A continuación, se encuentra el código perteneciente a la clase principal de la aplicación de un reconocedor de comandos de voz para la manipulación de un actuador eléctrico.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package reconocedorvoz;

import com.panamahitek.ArduinoException;
import com.panamahitek.PanamaHitek_Arduino;
import Javax.speech.*;
import Javax.speech.recognition.*;
import Java.io.FileReader;
import Java.util.Locale;
import Java.util.logging.Level;
import Java.util.logging.Logger;

public class ReconocedorVoz extends ResultAdapter{
public ReconocedorVoz(){

try {
   arduino.arduinoTX("COM3",9600);} catch (ArduinoException ex) {
   Logger.getLogger(ReconocedorVoz.class.getName()).log(Level.SEVERE, null, ex);
  }
}

/**
```

```
* @param args the command line arguments
*/
  PanamaHitek_Arduino arduino = new PanamaHitek_Arduino();
  static Recognizer recognizer;
  String gst;
  @Override
  public void resultAccepted(ResultEvent re)
  {
    try
    {
      Result res = (Result)(re.getSource());
      ResultToken tokens[]=res.getBestTokens();

      String args[]=new String[1];
      args[0]="";
      for(int i=0;i<tokens.length;i++)
      {
        gst=tokens[i].getSpokenText();
        args[0]+=gst+"";//no se ve bien si es ++ o +=
        System.out.print(gst+" ");
      }
      System.out.println();
      if (gst.equals("terminar"))
      {
        recognizer.deallocate();
        args[0]="may we meet again";
        System.out.println(args[0]);
        //lee main.args
        System.exit(0);
      }
      //muestra la imagen de emilia
      else if(gst.equals("izquierda"))
```

```
    {

        interfaz i= new interfaz();
        i.imagen();
        try {
 arduino.sendData("0");
} catch (ArduinoException ex) {
  Logger.getLogger(ReconocedorVoz.class.getName()).log(Level.SEVERE, null, ex);
}
      }
      //muestra la imagen de rem
    else if(gst.equals("derecha"))
    {
      rem r= new rem();
      r.imagen();
      try {
 arduino.sendData("1");
} catch (ArduinoException ex) {
  Logger.getLogger(ReconocedorVoz.class.getName()).log(Level.SEVERE, null, ex);
}
    }
    else if(gst.equals("arriba"))
    {
      subaru s= new subaru();
      s.imagen();
      try {
 arduino.sendData("1");
} catch (ArduinoException ex) {
  Logger.getLogger(ReconocedorVoz.class.getName()).log(Level.SEVERE, null, ex);
}
    }
    else
```

```
    {
      recognizer.suspend();
      //lee main(args)
      recognizer.resume();
    }
   }catch(Exception ex){
     System.out.println("ha ocurrido algo inesperado"+ex);
   }
 }

 public static void main(String[] args) {
   try
   {
     recognizer =Central.createRecognizer(new EngineModeDesc(Locale.ROOT));
     recognizer.allocate();

     FileReader grammar1;
     grammar1 = new FileReader("C:/grammar1.txt");

     RuleGrammar rg= recognizer.loadJSGF(grammar1);
     rg.setEnabled(true);

     recognizer.addResultListener(new ReconocedorVoz());

     System.out.println("empiece dictado");
     recognizer.commitChanges();

     recognizer.requestFocus();
     recognizer.resume();
   }
   catch(Exception e){
     System.out.println("error "+e);
```

```
      e.printStackTrace();
      System.exit(0);
    }
  }
}//*****************************************************************************************************
```

El código anterior hace huso de ciertas clases desarrolladas para comprobar la efectividad del mismo al momento de realizar el reconocimiento de diferentes comandos, el código de estas clases, aparece a continuación.

```
/*clase uno*/
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package reconocedorvoz;

/**
 *
 * @author subaru
 */

import Java.awt.*;
import Javax.swing.*;
import Java.awt.event.*;

public class interfaz extends JFrame {
  ImageIcon imagen = new ImageIcon("C:\\Users\\subaru\\Pictures\\emiliatan.png");
  JLabel etiqueta = new JLabel(imagen);
```

```
//Icon icono = new ImageIcon(imagen.getImage().getScaledInstance(etiqueta.getWidth(),etiqueta.getHeight(),Image.SCALE_DEFAULT));



public interfaz(){
  super("Emilia...");
  getContentPane().add(etiqueta);
  this.setSize(500,500);
}
public int imagen(){
   interfaz p = new interfaz();
  p.show();

  //codigo para permitir cerrar ventana
  p.addWindowListener(new WindowAdapter(){
  public void windowClosing(WindowEvent evento){
    System.exit(0);
  }
});
  return 0;
}
public static void main(String args[]){
  interfaz p = new interfaz();
  p.show();

  //código para permitir cerrar ventana
  p.addWindowListener(new WindowAdapter(){
  public void windowClosing(WindowEvent evento){
    System.exit(0);
  }
```

```
  });
  }//fin de main
}//fin de la clase*****************************************************************************

/*clase dos*/
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package reconocedorvoz;

import Java.awt.event.*;
import Javax.swing.*;
import Java.awt.*;

/**
 *
 * @author subaru
 */
public class rem extends JFrame{
 ImageIcon imagen = new ImageIcon("C:\\Users\\subaru\\Pictures\\rem3.jpg");
  JLabel etiqueta = new JLabel(imagen);

  public rem(){
    super("rem...");
    getContentPane().add(etiqueta);
    this.setSize(500,500);
  }
  public int imagen(){
     rem p = new rem();
    p.show();
```

```
    //código para permitir cerrar ventana
    p.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent evento){
      System.exit(0);
    }
  });
    return 0;
  }
  public static void main(String args[]){
    rem p = new rem();
    p.show();

    //codigo para permitir cerrar ventana
    p.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent evento){
      System.exit(0);
    }
  });
  }//fin de main
}//*************************************************************************************************


/*clase tres*/
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package reconocedorvoz;

import Java.awt.event.*;
```

```
import Java.awt.*;
import Javax.swing.*;

/**
 *
 * @author subaru
 */
public class subaru extends JFrame{
  ImageIcon imagen = new ImageIcon("C:\\Users\\subaru\\Pictures\\4720625_640px.jpg");
  JLabel etiqueta = new JLabel(imagen);

  public subaru(){
    super("Subaru...");
    getContentPane().add(etiqueta);
    this.setSize(500,500);
  }
  public int imagen(){
     subaru p = new subaru();
    p.show();

    //codigo para permitir cerrar ventana
    p.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent evento){
      System.exit(0);
    }
  });
    return 0;
  }
  public static void main(String args[]){
    subaru p = new subaru();
    p.show();
```

```
    //codigo para permitir cerrar ventana
    p.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent evento){
      System.exit(0);
    }
  });
  }//fin de main
}
//*************************************************************************************************************
```

En realidad, los códigos son bastante similares, ya que cumplen con la función de mostrar una imagen en pantalla

B. Archivo JSGF

Como se mencionó anteriormente, el archivo JSGF es aquel que dicta las reglas gramaticales que deberá seguir el reconocedor de voz, sus partes y la manera de crear un archivo de este tipo también se enuncia en capítulos anteriores.

Cabe destacar que el archivo podría funcionar con tan solo las primeras cuatro reglas gramaticales, pero entre mayor cantidad de palabras tenga este archivo, mejor será el desempeño de la aplicación al reconocer comandos de voz.

Este archivo fue creado según las quinientas palabras más pronunciadas dentro del lenguaje español según el artículo publicado en el sitio web www.clarin.com con el título "¿Cuáles son las 500 palabras más usadas en español?"

Como se mencionó con anterioridad, el archivo JSGF es un archivo en texto plano, el cual se puede crear en cualquier editor de texto plano. A continuación, aparece su contenido, dentro de dos columnas, por motivos de espacio.

```
#JSGF V1.0;
/**
 *@author Omar Vazquez Castro
 */
```

```
grammar imagenes;

public <izquierda>= izquierda;
public <derecha>= derecha;
public <abajo>= arriba;
public <terminar>= terminar;

public <de>=de;
public <la>=la;
public <que>=que;
public <el>=el;
public <en>=en;
public <y>=y;
public <a>=a;
public <los>=los;
public <se>=se;
public <del>=del;
public <las>=las;
public <un>=un;
public <por>=por;
public <con>=con;
public <no>=no;
public <una>=una;
public <su>=su;
public <para>=para;
public <es>=es;
public <al>=al;
public <lo>=lo;
public <como>=como;
public <mas>=más;
public <o>=o;
public <pero>=pero;
public <sus>=sus;
public <le>=le;
public <ha>=ha;
public <me>=me;
public <si>=si;
public <sin>=sin;
public <sobre>=sobre;
public <este>=este;
public <ya>=ya;
public <entre>=entre;
public <cuando>=cuando;
public <todo>=todo;
public <esta>=esta;
public <ser>=ser;
public <son>=son;
public <dos>=dos;
public <tambien>=tambien;
public <fue>=fue;
public <habia>=había;
public <era>=era;
public <muy>=muy;
public <anios>=años;
public <hasta>=hasta;
public <desde>=desde;
public <esta>=está;
```

public <mi>=mi;
public <porque>=porque;
public <que>=qué;
public <solo>=sólo;
public <han>=han;
public <yo>=yo;
public <hay>=hay;
public <vez>=vez;
public <puede>=puede;
public <todos>=todos;
public <asi>=así;
public <nos>=nos;
public <ni>=ni;
public <parte>=parte;
public <tiene>=tiene;
public <el>=él;
public <uno>=uno;
public <donde>=donde;
public <bien>=bien;
public <tiempo>=tiempo;
public <mismo>=mismo;
public <ese>=ese;
public <ahora>=ahora;
public <cada>=cada;
public <e>=e;
public <vida>=vida;
public <otro>=otro;
public <despues>=despúes;
public <te>=te;
public <otros>=otros;
public <aunque>=aunque;
public <esa>=esa;
public <eso>=eso;
public <hace>=hace;
public <otra>=otra;
public <gobierno>=gobierno;
public <tan>=;
public <durante>=durante;
public <siempre>=siempre;
public <dia>=día;
public <tanto>=tanto;
public <ella>=ella;
public <tres>=tres;
public <si>=sí;
public <dijo>=dijo;
public <sido>=sido;
public <gran>=gran;
public <pais>=país;
public <segun>=según;
public <menos>=menos;
public <anio>=año;
public <antes>=antes;
public <estado>=estado;
public <contra>=contra;
public <sino>=sino;
public <forma>=forma;
public <caso>=caso;
public <nada>=nada;

public <hacer>=hacer;
public <general>=general;
public <estaba>=estaba;
public <poco>=poco;
public <estos>=estos;
public <presidente>=presidente;
public <mayor>=mayor;
public <ante>=ante;
public <unos>=unos;
public <les>=les;
public <algo>=algo;
public <hacia>=hacia;
public <casa>=casa;
public <ellos>=ellos;
public <ayer>=ayer;
public <hecho>=hecho;
public <primera>=primera;
public <mucho>=mucho;
public <mientras>=mientras;
public <ademas>=ademas;
public <quien>=quien;
public <momento>=momento;
public <millones>=millones;
public <esto>=esto;
public <espania>=españa;
public <hombre>=hombre;
public <estan>=están;
public <pues>=pues;
public <hoy>=hoy;
public <lugar>=lugar;
public <madrid>=madrid;
public <nacional>=nacional;
public <trabajo>=trabajo;
public <otras>=otras;
public <mejor>=mejor;
public <nuevo>=nuevo;
public <decir>=decir;
public <algunos>=algunos;
public <entonces>=entonces;
public <todas>=todas;
public <dias>=días;
public <debe>=debe;
public <politica>=política;
public <como>=cómo;
public <casi>=casi;
public <toda>=toda;
public <tal>=tal;
public <luego>=luego;
public <pasado>=pasado;
public <primer>=primer;
public <medio>=medio;
public <va>=va;
public <estas>=estas;
public <sea>=sea;
public <tenia>=tenía;
public <nunca>=nunca;
public <poder>=poder;
public <aqui>=aqui;

public <ver>=ver;
public <veces>=veces;
public <embargo>=embargo;
public <partido>=partido;
public <personas>=personas;
public <grupo>=grupo;
public <cuenta>=cuenta;
public <pueden>=pueden;
public <tienen>=tienen;
public <misma>=misma;
public <nueva>=nueva;
public <cual>=cual;
public <fueron>=fueron;
public <mujer>=mujer;
public <frente>=frente;
public <jose>=josé;
public <tras>=tras;
public <cosas>=cosas;
public <fin>=fin;
public <ciudad>=ciudad;
public <he>=he;
public <social>=social;
public <manera>=manera;
public <tener>=tener;
public <sistema>=sistema;
public <sera>=será;
public <historia>=historia;
public <muchos>=muchos;
public <juan>=juan;
public <tipo>=tipo;
public <cuatro>=cuatro;
public <dentro>=dentro;
public <nuestro>=nuestro;
public <punto>=punto;
public <dice>=dice;
public <ello>=ello;
public <cualquier>=cualquier;
public <noche>=noche;
public <aun>=aún;
public <agua>=agua;
public <parece>=parece;
public <haber>=haber;
public <situacion>=situacion;
public <fuera>=fuera;
public <bajo>=bajo;
public <grandes>=grandes;
public <nuestra>=nuestra;
public <ejemplo>=ejemplo;
public <acuerdo>=acuerdo;
public <habian>=habían;
public <usted>=usted;
public <estados>=estados;
public <hizo>=hizo;
public <nadie>=nadie;
public <paises>=países;
public <horas>=horas;
public <posible>=posible;
public <tarde>=tarde;

public <ley>=ley;
public <importante>=importante;
public <guerra>=guerra;
public <desarrollo>=desarrollo;
public <proceso>=proceso;
public <realidad>=realidad;
public <sentido>=sentido;
public <lado>=lado;
public <mi>=mí;
public <tu>=tu;
public <cambio>=cambio;
public <alli>=allí;
public <mano>=mano;
public <eran>=eran;
public <estar>=estar;
public <san>=san;
public <numero>=número;
public <sociedad>=sociedad;
public <unas>=unas;
public <centro>=centro;
public <padre>=padre;
public <gente>=gente;
public <final>=final;
public <relacion>=relación;
public <cuerpo>=cuerpo;
public <obra>=obra;
public <incluso>=incluso;
public <traves>=través;
public <ultimo>=último;
public <madre>=madre;
public <mis>=mis;
public <modo>=modo;
public <problemas>=problemas;
public <cinco>=cinco;
public <carlos>=carlos;
public <hombres>=hombres;
public <informacion>=información;
public <ojos>=ojos;
public <muerte>=muerte;
public <nombre>=nombre;
public <algunas>=algunas;
public <publico>=público;
public <mujeres>=mujeres;
public <siglo>=siglo;
public <todavia>=todavía;
public <meses>=meses;
public <maniana>=mañana;
public <esos>=esos;
public <nosotros>=nosotros;
public <hora>=hora;
public <muchas>=muchas;
public <pueblo>=pueblo;
public <alguna>=alguna;
public <dar>=dar;
public <problema>=problema;
public <don>=don;
public <da>=da;
public <tu>=tú;

public <derecho>=derecho;
public <verdad>=verdad;
public <maria>=maría;
public <unidos>=unidos;
public <podria>=podría;
public <seria>=sería;
public <junto>=junto;
public <cabeza>=cabeza;
public <aquel>=aquel;
public <luis>=luis;
public <cuanto>=cuanto;
public <tierra>=tierra;
public <equipo>=equipo;
public <segundo>=segundo;
public <director>=director;
public <dicho>=dicho;
public <cierto>=cierto;
public <casos>=casos;
public <manos>=manos;
public <nivel>=nivel;
public <podia>=podía;
public <familia>=familia;
public <largo>=largo;
public <partir>=partir;
public <falta>=falta;
public <llegar>=llegar;
public <propio>=propio;
public <ministro>=ministro;
public <cosa>=cosa;
public <primero>=primero;
public <seguridad>=seguridad;
public <hemos>=hemos;
public <mal>=mal;
public <trata>=trata;
public <algun>=algún;
public <tuvo>=tuvo;
public <respecto>=respecto;
public <semana>=semana;
public <varios>=varios;
public <real>=real;
public <se>=sé;
public <voz>=voz;
public <paso>=paso;
public <senior>=señor;
public <mil>=mil;
public <quienes>=quienes;
public <proyecto>=proyecto;
public <mercado>=mercado;
public <mayoria>=mayoría;
public <luz>=luz;
public <claro>=claro;
public <iba>=iba;
public <este>=éste;
public <pesos>=pesos;
public <orden>=orden;
public <espaniol>=español;
public <buena>=buena;
public <quiere>=quiere;

public <aquella>=aquella;
public <programa>=programa;
public <palabras>=palabras;
public <internacional>=internacional;
public <van>=van;
public <esas>=esas;
public <segunda>=segunda;
public <empresa>=empresa;
public <puesto>=puesto;
public <ahi>=ahí;
public <propia>=propia;
public <m>=m;
public <libro>=libro;
public <igual>=igual;
public <politico>=político;
public <persona>=persona;
public <ultimos>=últimos;
public <ellas>=ellas;
public <total>=total;
public <creo>=creo;
public <tengo>=tengo;
public <dios>=dios;
public <c>=c;
public <espaniola>=española;
public <condiciones>=condiciones;
public <mexico>=méxico;
public <fuerza>=fuerza;
public <solo>=solo;
public <unico>=único;
public <accion>=acción;
public <amor>=amor;
public <policia>=policía;
public <puerta>=puerta;
public <pesar>=pesar;
public <zona>=zona;
public <sabe>=sabe;
public <calle>=calle;
public <interior>=interior;
public <tampoco>=tampoco;
public <musica>=música;
public <ningun>=ningún;
public <vista>=vista;
public <campo>=campo;
public <buen>=buen;
public <hubiera>=hubiera;
public <saber>=saber;
public <obras>=obras;
public <razon>=razón;
public <ex>=ex;
public <ninios>=niños;
public <presencia>=presencia;
public <tema>=tema;
public <dinero>=dinero;
public <comision>=comisión;
public <antonio>=antonio;
public <servicio>=servicio;
public <hijo>=hijo;
public <ultima>=última;

public <ciento>=ciento;
public <estoy>=estoy;
public <hablar>=hablar;
public <dio>=dio;
public <minutos>=minutos;
public <produccion>=producción;
public <camino>=camino;
public <seis>=seis;
public <quien>=quién;
public <fondo>=fondo;
public <direccion>=dirección;
public <papel>=papel;
public <demas>=demás;
public <barcelona>=barcelona;
public <idea>=idea;
public <especial>=especial;
public <diferentes>=diferentes;
public <dado>=dado;
public <base>=base;
public <cAPItal>=cAPItal;
public <ambos>=ambos;
public <europa>=europa;
public <libertad>=libertad;
public <relaciones>=relaciones;
public <espacio>=espacio;
public <medios>=medios;
public <ir>=ir;
public <actual>=actual;
public <poblacion>=población;
public <empresas>=empresas;
public <estudio>=estudio;
public <salud>=salud;
public <servicios>=servicios;
public <haya>=haya;
public <principio>=principio;
public <siendo>=siendo;
public <cultura>=cultura;
public <anterior>=anterior;
public <alto>=alto;
public <media>=media;
public <mediante>=mediante;
public <primeros>=primeros;
public <arte>=arte;
public <paz>=paz;
public <sector>=sector;
public <imagen>=imagen;
public <medida>=medida;
public <deben>=deben;
public <datos>=datos;
public <consejo>=consejo;
public <personal>=personal;
public <interes>=interés;
public <julio>=julio;
public <grupos>=grupos;
public <miembros>=miembros;
public <ninguna>=ninguna;
public <existe>=existe;
public <cara>=cara;

public <edad>=edad;
public <etcetera>=etcetera;
public <movimiento>=movimiento;
public <visto>=visto;
public <llego>=llegó;
public <puntos>=puntos;
public <actividad>=actividad;
public <bueno>=bueno;
public <uso>=uso;
//public <niño>=niño;
public <dificil>=difícil;
public <joven>=joven;
public <futuro>=futuro;
public <aquellos>=aquellos;
public <mes>=mes;
public <pronto>=pronto;
public <soy>=soy;
public <hacia>=hacía;
public <nuevos>=nuevos;
public <nuestros>=nuestros;
public <estaban>=estaban;
public <posibilidad>=posibilidad;
public <sigue>=sigue;
public <cerca>=cerca;
public <resultados>=resultados;
public <educacion>=educación;
public <atencion>=atención;
public <gonzalez>=gonzalez;
public <capacidad>=capacidad;
public <efecto>=efecto;
public <necesario>=necesario;
public <valor>=valor;
public <aire>=aire;
public <investigacion>=investigación;
public <siguiente>=siguiente;
public <figura>=figura;
public <central>=central;
public <comunidad>=comunidad;
public <necesidad>=necesidad;
public <serie>=serie;
public <organizacion>=organizacion;
public <nuevas>=nuevas;
public <calidad>=calidad;

C. Código Arduino

El código Arduino es importante para encender el motor, pero no sirve de nada, si no se realizó antes la conexión entre la placa Arduino y la aplicación Java.

```
int input;


void setup() {
```

```
 // put your setup code here, to run once:
pinMode(13,OUTPUT);//puerto 13 alida
Serial.begin(9600);//inicio de comunicacon
}

void loop() {
 // put your main code here, to run repeatedly:
if (Serial.available()>0){
 input=Serial.read();

 if (input=='1'){
  digitalWrite(13,HIGH);//si valor 1 se enciende el motor
 }
 else
 {
  digitalWrite(13,LOW);//valor diferente de 1 apaga el motor
 }
}
}
```

GLOSARIO

Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.

Actuador: Dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o “actuar” otro dispositivo mecánico.

Amplitud: Es el valor máximo de un movimiento o señal. Se trata de la distancia que hay desde el punto de equilibrio (cero), hasta uno de los extremos del movimiento, puede ser el punto positivo o el negativo.

Convolución del espectro: Equivale a la multiplicación del espectro de dos señales. A grosso modo es como filtrar un sonido a través de un ecualizador, cuyos niveles de bandas han sido obtenidos a partir del análisis espectral de otro sonido.

Domótica: Conjunto de técnicas orientadas a automatizar una vivienda, que integran la tecnología en los sistemas de seguridad, gestión energética, bienestar o comunicaciones.

HTK: Kit de herramientas de software utilizado para el manejo de HMM. Se destina principalmente al reconocimiento de voz, pero también se utiliza en aplicaciones de reconocimiento de patrones que emplean modelos ocultos de Markov, incluida la síntesis de voz, el reconocimiento de caracteres y la secuenciación de ADN.

Impedancia: Oposición combinada de elementos del circuito al paso de corriente. Cuando en un mismo circuito se tienen resistencias, condensadores y bobinas, y por ellas circula corriente alterna, la oposición de este conjunto de elementos al paso de la corriente alterna se llama impedancia.

Modelo dependiente: Tipo de modelo en el cual la decisión final depende de otras variables que permiten explicar sus determinantes.

Señal flotante: Es aquella que no tiene conexión a tierra, pero tiene un punto fijo aislado eléctricamente del chasis o plaqueta donde se diseñó el circuito y en el cual convergen todos los retornos negativos.

Servidor domótico: Procesador en el que se encuentra albergado un sistema domótico, el cual puede ser instalado en una vivienda para controlar distintas

variables ambientales y conseguir así la máxima comodidad de los habitantes de manera automática o manual según los gustos y necesidades de los usuarios

BIBLIOGRAFÍA

3scale. (2012). *What is an API.* 3scale.

Alonzo Velázquez, J. L. (2010, 08). *www.cimat.mx*. Retrieved 01 05, 2018, from http://www.cimat.mx/~pepe/cursos/lenguaje_2010/slides/slide_17.pdf

Arduino. (2018). *https://www.arduino.cc*. Retrieved 06 09, 2018, from https://www.arduino.cc: https://www.arduino.cc/en/Main/arduinoBoardMega

Cancelo, P., & Alonso, J. (2007). *La tercera revolucion. Comunicación,tecnologia y su nomenclatura en inglés.* España: Netbio.

Castro Lechtaler, A. R., & Jorge Fusario, R. (1999). *Teleinformatica para ingenieros en sistemas de Informacion.* Buenos Aires: Reverté S. A.

Corona Ramírez, L., Abarca Jiménez, G., & Mares Carreño, J. (2014). *Sensores y Actudores: Aplicaciones con Arduino.* Ciudad de México: Grupo Editorial Patria.

Delgado Gallardo, V. A., Trujillo Castellanos, A., & Valdez Simón, D. (2014). *Sistema de domótica controlado por voz para personas con deficiencias motrices.* México, DF: IPN.

Department of General Linguistics at the University of Helsinki . (n.d.). *www.ling.helsinki.fi*. Retrieved 03 26, 2017, from http://www.ling.helsinki.fi/~gwilcock/Tartu-2003/L7-Speech/JSAPI/SpeechEngine.html#14106

Depto.CCIA Universidad Alicante. (2006). *Lenguajes de programacion .* Universidad Alicante.

Fazzino, M., & Sanchéz, O. (2008). Mouse para minusvalidos. *Tekhne Revista de la Facultad de Ingenieria* , 46-54.

García Galván, M. A. (2005). *Reconocedor de Voz Adaptado.* México, DF: UAM Azcapotzalco.

Gimeno Perez, F., & Torres Gallardo, B. (2008). *Anatomia de la voz.* Paidotribbo.

Gomez Gutierrez, E. (2009, 09 28). *www.dtic.upf.edu*. Retrieved 01 19, 2018, from http://www.dtic.upf.edu/~egomez/teaching/sintesi/SPS1/Tema3-Representacion.pdf

Gómez Gutierrez, E. (2009, 09 28). *www.dtic.upf.edu.* Retrieved 02 04, 2018, from http://www.dtic.upf.edu/~egomez/teaching/sintesi/SPS1/Tema3-Representacion.pdf

Hunt, A. (2000). *www.w3.org*. Retrieved 11 14, 2017, from www.w3.org: https://www.w3.org/TR/2000/NOTE-jsgf-20000605/

Macas Macas, D. X., & Padilla Pineda, W. A. (2012). *Estudio de los modelos ocultos de Markov y desarrollo de un prototipo para el reconocimiento automatico del habla.* Universidad Politécnica Salesiana.

NetBeans. (2018). *netbeans.org*. Retrieved 06 9, 2018, from https://netbeans.org/features/index.html

NetBeans. (2018). *netbeans.org*. Retrieved 06 09, 2018, from https://netbeans.org/downloads/

O´Shaughnessy. (1987). *Speech communication: human and machine.* Universities Press.

ORACLE CORPORATION. (n.d.). *www.java.com*. Retrieved 01 20, 2018, from https://www.java.com/es/download/faq/whatis_java.xml

Pallás Areny, R. (1993). *Adquisición y distribución de señales.* Barcelona: Marcombo, Boixareu Editores.

Panta Martínez, J. (2012). *Control Domotico por Voz.* Valencia, España: Universidad Politecnica de Válencia.

Peinado, A. (1994). *Seleccion y estimacion de parametros en sistemas de reconocimiento de voz basados en modelos ocultos de Markov .* Granada: Tesis Doctoral.

Pérez Badillo, E. O., Ponceros Martínez, F., & Villalobos Ponce, J. A. (2013). *Sistema de seguridad por reconocimiento de voz.* México, DF: IPN.

Perez, B. (2014). *Aprende a crear y diseñar soluciones en telefonia IP desde cero .* Asterisk PBX. Retrieved 01 19, 2018, from

http://ocw.innova.uned.es/mmm3/audio_digital/contenidos/pdf/La_senal_de_audio.pdf

Rabiner, L. R., & Juang, B. H. (1993). *Fundamental of speech recognition .* New Jersey: Prentice Hall.

Rincón, L. (2012). *Introducción a los procesos estocásticos.* UNAM.

Roads, C. (1996). *The Computer Music Tutorial.* MIT Press.

Sanchez, O., & Lemus, E. (n.d.). *prezi.com*. Retrieved 02 04, 2018, from https://prezi.com/c8srkwceictp/procesamiento-de-senales-de-audio/

Sanchis, E., Torralba, G., Gonzalez, V., & Torres, J. (2005). *Fundamentos y Electronica de las Comunicaciones.* Valencia: GUADA Impresores SL.

Varona Férnandez, A. (1997). *Antecedentes y desarrollo de los sistemas actuales del reconocimiento automatico del habla .* Bilbao: Universidad del país Vasco.

Villamil Espinoza, I. H. (2005). *Aplicaciones de reconocimientos de voz utilizando HTK.* Santa Fe de Bogota DC: Pontificia Universidad Javeriana.

Wayne, T. (2003). *Sistema de comunicaciones electronicas.* Prentice Hall.

www.fceia.unr.edu.ar. (n.d.). Retrieved 02 04, 2018, from http://www.fceia.unr.edu.ar/prodivoz/Reconocimiento_Automatico_Voz_bw.pdf