Instituto Politécnico Nacional

Escuela Superior de Cómputo

Reconocimiento de palabras mediante Modelado HMM

Alumnos:
Escamilla Reséndiz Aldo
Sierra Fierro Samuel Isaac
Vásquez Morales Haniel Ulises

Asignatura: Reconocimiento de Voz

Profesor: Enrique Alfonso Carmona García

Grupo: 7BM1

Fecha de entrega: 27 de junio del 2025

Índice

Índice de figuras

1. Introducción

El reconocimiento automático del habla es una disciplina consolidada en el procesamiento de señales y el aprendizaje automático. Uno de los enfoques clásicos para el modelado de señales acústicas es el uso de Modelos Ocultos de Markov (HMM), capaces de capturar dependencias temporales en secuencias de características fonéticas [1]. Este trabajo implementa un sistema completo que inicia desde la segmentación acústica de la señal, extrae características relevantes como los coeficientes cepstrales en la frecuencia de Mel (MFCC), y entrena un HMM por clase fonética. Posteriormente, se integra esta arquitectura con una interfaz gráfica que responde a comandos de voz para controlar el movimiento de un vehículo en una pista simulada. Esta metodología se inspira en prácticas consolidadas en el reconocimiento de habla [2], y extiende su aplicación a sistemas interactivos de control por voz.

2. Objetivo

Objetivo general: Codificar un sistema de reconocimiento automático de voz sencillo, el sistema reconocerá las palabras: izquierda, centro y derecha, para mover un carro en una interfaz gráfica.

Objetivos específicos:

- Generar y entrenar un modelo por medio de Gaussian HMM.
- Evaluar la precisión del modelo entrenado.
- Crear DataSet de palabras: izquierda, centro y derecha.
- Detectar comando en interfaz gráfica.

3. Aplicación Interactiva Basada en Reconocimiento de Voz

Con el objetivo de validar el sistema en un entorno funcional, se desarrolló una interfaz gráfica con `Tkinter` que simula el desplazamiento de un vehículo en tres carriles, controlado mediante comandos de voz. Este módulo traduce entradas acústicas (como "izquierda", "centro" o "derecha") en acciones de movimiento sobre un lienzo gráfico.

3.1. Interfaz Gráfica

El entorno gráfico está compuesto por una pista de fondo y una imagen de un vehículo que se puede desplazar entre tres posiciones horizontales predefinidas. La interfaz es creada mediante la biblioteca `Tkinter`, mientras que las imágenes utilizadas se cargan y ajustan mediante la librería `Pillow` (`PIL`), permitiendo una manipulación eficiente de recursos visuales en formato rasterizado.

El vehículo responde a comandos de voz previamente clasificados por el sistema acústico-fonético, lo que permite emular una interacción básica basada en lenguaje natural. El movimiento está restringido a tres carriles: izquierdo, central y derecho, simulando una dinámica de conducción asistida controlada por voz.

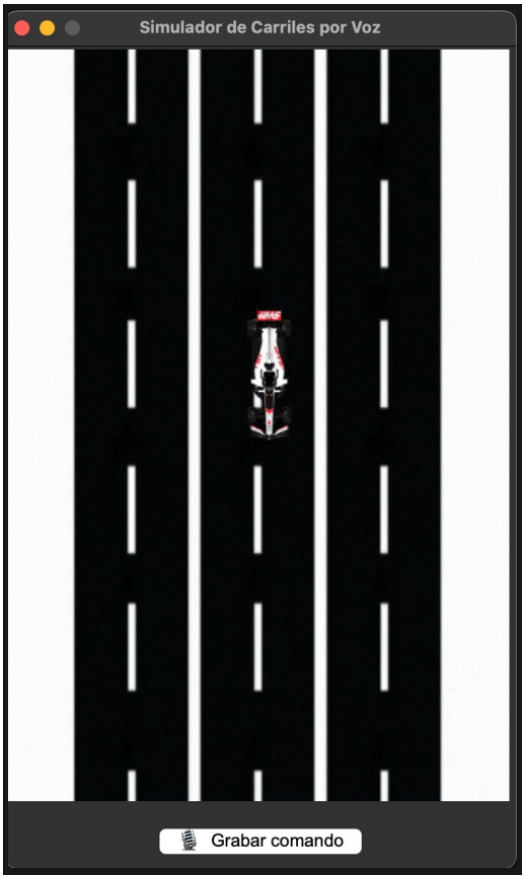


Figura 1: Interfaz base: pista con vehículo móvil controlado por comandos de voz

3.2. Captura y Clasificación de Comandos de Voz

Cada vez que el usuario presiona el botón "Grabar comando", el sistema activa el micrófono y graba un fragmento de audio de dos segundos a una frecuencia de muestreo de 22 050 Hz. La grabación se guarda en formato `.wav` y luego se procesa para su clasificación.

```
def grabar_audio(nombre_archivo='prueba.wav', duracion=2, sr
    =22050):
    audio = sd.rec(int(duracion * sr), samplerate=sr,
    channels=1, dtype='int16')
    sd.wait()
```

```
    write(nombre_archivo, sr, audio)
```

Listing 1: Grabación de audio desde micrófono

El archivo resultante es procesado con las mismas funciones de segmentación y extracción de características definidas previamente. Posteriormente, cada modelo HMM evalúa la secuencia observada y retorna una probabilidad logarítmica, eligiendo la clase fonética más probable.

```
def clasificar_audio(modelos, archivo):
    data, sr = librosa.load(archivo)
    segmentos = segmentAudio(data, sr)
    features = extractFeatures(segmentos, sr)
    scores = {fonema: modelo.score(features) for fonema,
    modelo in modelos.items()}
    return max(scores, key=scores.get)
```

Listing 2: Clasificación de comandos de voz

3.3. Movimiento Basado en Clasificación

La clase predicha se asocia a una posición del vehículo en la pista. La lógica de movimiento se actualiza dinámicamente en la interfaz:

- Si la clase es ïzquierda", el vehículo se mueve al carril izquierdo.
- Si la clase es çentro", se mantiene en el carril central.
- Si la clase es "derecha", se traslada al carril derecho.

El siguiente fragmento de código muestra la lógica del botón de grabación y actualización de posición:

```
def actualizar_carro():
    grabar_audio('prueba.wav')
    clase = clasificar_audio(modelos, 'prueba.wav')
    if clase == 'izquierda':
        current_position = 0
    elif clase == 'centro':
        current_position = 1
    elif clase == 'derecha':
        current_position = 2
    canvas.coords(carro_id, car_position_x[current_position],
    car_position_y)
```

Listing 3: Actualización gráfica según comando de voz

4. Fundamentación

4.1. Preprocesamiento y Segmentación Acústica

El preprocesamiento de señales de voz constituye una etapa crítica en cualquier sistema de reconocimiento acústico. En este trabajo, el análisis comienza con la segmentación de la señal continua en intervalos cortos mediante una ventana deslizante de duración fija. Específicamente, se emplean ventanas de 50 ms con saltos de 25 ms, lo que permite capturar la cuasiestacionariedad de la señal de habla, característica inherente a escalas temporales reducidas [1].

Para garantizar la relevancia de los segmentos analizados, se establece un umbral mínimo de energía ($E > 10^{-5}$) por ventana. Este filtrado busca descartar silencios o ruido de fondo, preservando solo regiones con información fonética útil. La energía de cada segmento se calcula como la media de la potencia instantánea:

$$E = \frac{1}{N} \sum_{n=1}^{N} x(n)^2$$

```
def segmentAudio(data, sr, seg_duration=0.05, hop_duration=0.025):
    seg_length = int(seg_duration * sr)
    hop_length = int(hop_duration * sr)
    segments = []
    for i in range(0, len(data) - seg_length + 1, hop_length):
        segment = data[i : i + seg_length]
        energy = np.sum(segment**2) / len(segment)
        if energy > 0.00001:
            segments.append(segment)
    return segments
```

Listing 4: Segmentación de señal con umbral de energía

4.2. Extracción de Características Acústicas

Cada segmento de audio se describe mediante un vector de características acústicas compuesto por:

- **Energía**: permite diferenciar entre segmentos sonoros y silenciosos.
- **Tasa de cruce por cero (ZCR)**: mide la cantidad de veces que la señal cambia de signo, lo cual aporta información sobre la sonoridad y la textura espectral del fonema.
- **Coeficientes MFCC (Mel-Frequency Cepstral Coefficients)**: representan la envolvente espectral de la señal en una escala perceptual. Se utilizan 12 coeficientes cepstrales extraídos con una longitud de ventana efectiva de 1024 puntos FFT.
- **Derivadas temporales de primer y segundo orden**: también conocidas como coeficientes *delta* y *delta-delta*, proporcionan información sobre la dinámica de cambio de los MFCC a lo largo del tiempo.

El vector final de características por segmento se forma concatenando los promedios de los coeficientes estáticos y dinámicos:

$$\vec{f} = [E, ZCR, \mu(\text{MFCC}), \mu(\Delta), \mu(\Delta^2)]$$

```
def extractFeatures(segments, sr):
    features = []
    for segment in segments:
        energy = np.sum(segment**2) / len(segment)
        zcr_val = zero_crossing_rate(segment)
        mfcc_feat = mfcc(segment, samplerate=sr, numcep=12,
   nfft=1024)
        delta_feat = delta(mfcc_feat, 2)
        delta_delta_feat = delta(delta_feat, 2)
        mfcc_combined = np.hstack([
            np.mean(mfcc_feat, axis=0),
            np.mean(delta_feat, axis=0),
            np.mean(delta_delta_feat, axis=0)
        ])
        feature_vector = [energy, zcr_val] + mfcc_combined.
   tolist()
        features.append(feature_vector)
    return np.array(features)
```

Listing 5: Extracción de características acústicas

4.3. Modelado Estocástico mediante HMM

Una vez extraídas las características acústicas de cada segmento, se procede al modelado estocástico de las secuencias utilizando Modelos Ocultos de Markov (HMM, por sus siglas en inglés). Estos modelos son ampliamente utilizados en el reconocimiento de voz debido a su capacidad para modelar fenómenos temporales donde las observaciones son generadas por una secuencia de estados no observables directamente [1].

Cada clase fonética c_k es representada mediante un HMM M_k independiente, entrenado exclusivamente con ejemplos etiquetados de esa clase. La estructura formal de un HMM está definida por una tripleta (A, B, π):

- $A = \{a_{ij}\}$: matriz de transición de estados, donde $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$ es la probabilidad de transición del estado s_i al estado s_j.
- $B = \{b_j(\vec{f})\}$: conjunto de funciones de densidad de probabilidad de las observaciones, donde $b_j(\vec{f}) = P(\vec{f_t} \mid q_t = s_j)$ indica la probabilidad de observar un vector de características $\vec{f_t}$ dado que el sistema está en el estado s_j.
- $\pi = \{\pi_i\}$: distribución de probabilidades iniciales, donde $\pi_i = P(q_1 = s_i)$ representa la probabilidad de que la secuencia inicie en el estado s_i.

En este trabajo, se implementan HMMs con $n = 5$ estados ocultos por clase fonética. Esta elección no es arbitraria, sino que busca reflejar la estructura interna de los fonemas, los cuales pueden conceptualizarse como unidades acústicas con una evolución temporal diferenciada. Aunque un fonema puede durar apenas entre 40 y 150 milisegundos, su producción articula distintas fases que afectan sus propiedades espectrales y energéticas. Utilizar cinco estados permite capturar esta evolución interna con suficiente granularidad, manteniendo al mismo tiempo una complejidad computacional manejable.

Los cinco estados ocultos pueden interpretarse como una descomposición funcional de la dinámica fonética, aproximadamente distribuida de la siguiente manera:

1. **Inicio articulatorio:** fase en la que se inicia la producción del fonema, caracterizada por un cambio abrupto en las propiedades espectrales.
2. **Transición creciente:** fase de incremento energético y estabilización del tracto vocal.
3. **Meseta estable:** fase central donde se mantiene una configuración articulatoria sostenida; aquí se encuentra el núcleo acústico del fonema.
4. **Transición decreciente:** etapa de relajación en la producción, con cambios en las frecuencias formantes.
5. **Liberación o cierre:** fase terminal, frecuentemente asociada a la preparación para el siguiente fonema o una caída energética.

Esta segmentación funcional no pretende ser fonéticamente exacta, pero sí proporciona una base estructural útil para capturar patrones relevantes en la evolución temporal de cada clase fonética.

Las distribuciones de emisión B en cada estado se modelan como gaussianas multivariadas con matriz de covarianza diagonal. Esta decisión implica asumir independencia entre las distintas dimensiones del vector de características (MFCCs, energía, ZCR, y sus derivadas), lo cual reduce significativamente la cantidad de parámetros a estimar. Esta simplificación permite un entrenamiento más eficiente, especialmente cuando el conjunto de datos disponible es limitado, y ha sido validada empíricamente en múltiples aplicaciones de reconocimiento de voz [2].

El entrenamiento se realiza mediante el algoritmo de *Baum-Welch*, una instancia del método de Expectation-Maximization (EM), que ajusta iterativamente los parámetros (A, B, π) para maximizar la probabilidad total de las observaciones dadas las secuencias de entrenamiento.

```
def entrenar_modelos_hmm(archivos_dict, seg_duration=0.05, hop_duration=0.025):
    modelos = {}
    for fonema, lista_archivos in archivos_dict.items():
        print(f"Entrenando HMM para fonema: {fonema}")
        features = Dataset(lista_archivos, seg_duration, hop_duration)
        if len(features) > 0:
            modelo = hmm.GaussianHMM(n_components=5, covariance_type='diag', n_iter=200)
            modelo.fit(features)
            modelos[fonema] = modelo
    return modelos
```

Listing 6: Entrenamiento de modelos HMM por clase

Durante la fase de prueba, cada archivo es segmentado y transformado en una secuencia de vectores de características. Esta secuencia es evaluada con cada modelo M_k, y se calcula la probabilidad logarítmica de que haya sido generada por dicho modelo. La clase asignada corresponde al modelo con mayor verosimilitud:

$$\hat{c} = \arg\max_k \log P(\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_T \mid M_k)$$

donde T es el número de segmentos extraídos y $\vec{f}_t$ representa el vector de características del segmento t-ésimo.

```python
def clasificar_audio(modelos, archivo, seg_duration=0.05, hop_duration=0.025):
    data, sr = librosa.load(archivo)
    segmentos = segmentAudio(data, sr, seg_duration, hop_duration)
    features = extractFeatures(segmentos, sr)

    scores = {}
    for fonema, modelo in modelos.items():
        try:
            score = modelo.score(features)
            scores[fonema] = score
        except:
            scores[fonema] = float('-inf')
    predicho = max(scores, key=scores.get)
    return predicho, scores
```

Listing 7: Clasificación de un archivo de audio

4.4. Evaluación de Desempeño

Con el fin de validar cuantitativamente la efectividad del sistema propuesto, se implementa una evaluación basada en la precisión (*accuracy*) obtenida sobre un conjunto de prueba. Para ello, el corpus se divide en dos subconjuntos mutuamente excluyentes:

- **Entrenamiento**: se utiliza para ajustar los parámetros de cada modelo HMM.
- **Prueba**: contiene ejemplos no vistos durante el entrenamiento, utilizados para estimar el desempeño general del sistema.

Cada archivo del conjunto de prueba se clasifica mediante el procedimiento descrito anteriormente, y se compara la clase predicha con la clase real. A partir de estas comparaciones, se computa la métrica de precisión como:

$$\text{Accuracy} = \frac{N_{\text{correctas}}}{N_{\text{total}}} \times 100$$

donde $N_{\text{correctas}}$ es el número total de predicciones coincidentes con la clase esperada, y N_{total} es el total de muestras evaluadas.

Este enfoque proporciona una medida global del rendimiento del sistema, permitiendo identificar posibles sesgos o deficiencias en el entrenamiento de ciertos fonemas.

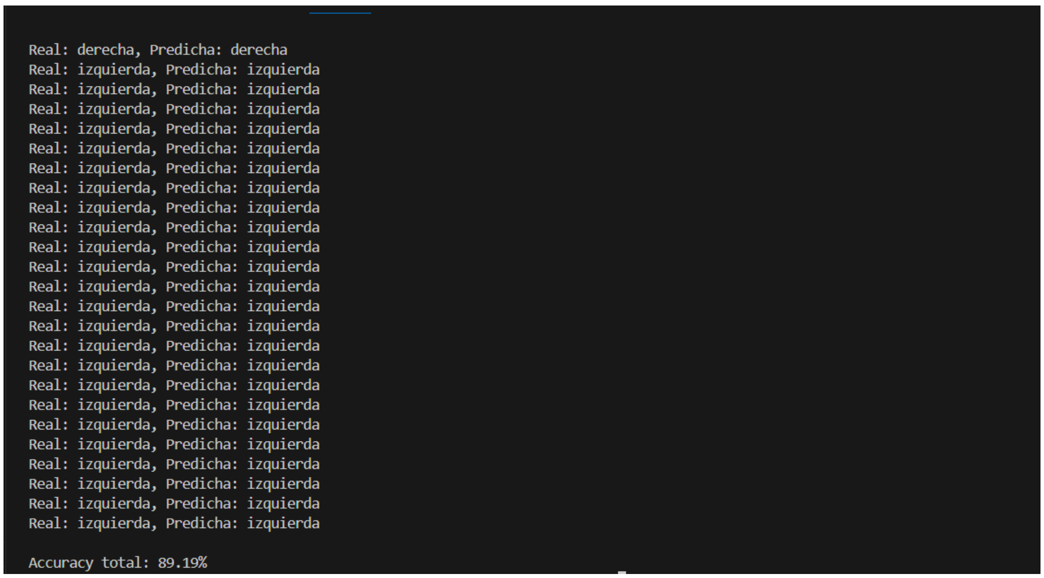
```
def evaluar_modelos(modelos, test_dict):
    total = 0
    aciertos = 0
    for clase_real, archivos in test_dict.items():
        for archivo in archivos:
            clase_predicha, _ = clasificar_audio(modelos,
   archivo)
            total += 1
            if clase_predicha == clase_real:
                aciertos += 1
            print(f"Real: {clase_real}, Predicha: {
   clase_predicha}")
    accuracy = aciertos / total if total > 0 else 0
    print(f"\nAccuracy total: {accuracy*100:.2f}%")
```

Listing 8: Evaluación de modelos HMM

Además del valor agregado de cuantificar la precisión, esta evaluación facilita futuras extensiones del sistema, como el análisis de matrices de confusión o la incorporación de métricas adicionales como *Recall* o *F1-Score*, especialmente útiles cuando se trata de clases desbalanceadas o tareas de reconocimiento multiclase más complejas.

5. Resultados

5.1. Resultados de evaluación


```
Real: derecha, Predicha: derecha
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda
Real: izquierda, Predicha: izquierda

Accuracy total: 89.19%
```

Figura 2: Accuracy

Se puede observar en la Figura2 un accuracy del 89.19 %. Este porcentaje refleja que funciona dentro del marco de nuestro conjunto de audios, es decir, los audios son de las voces de los miembros del equipo, por lo qué, no es un DataSet uniforme.

5.2. Detección de comandos

Se muestran los comandos ”derecha”, ”centro” e ”izquierda” funcionando correctamente, respectivamente.

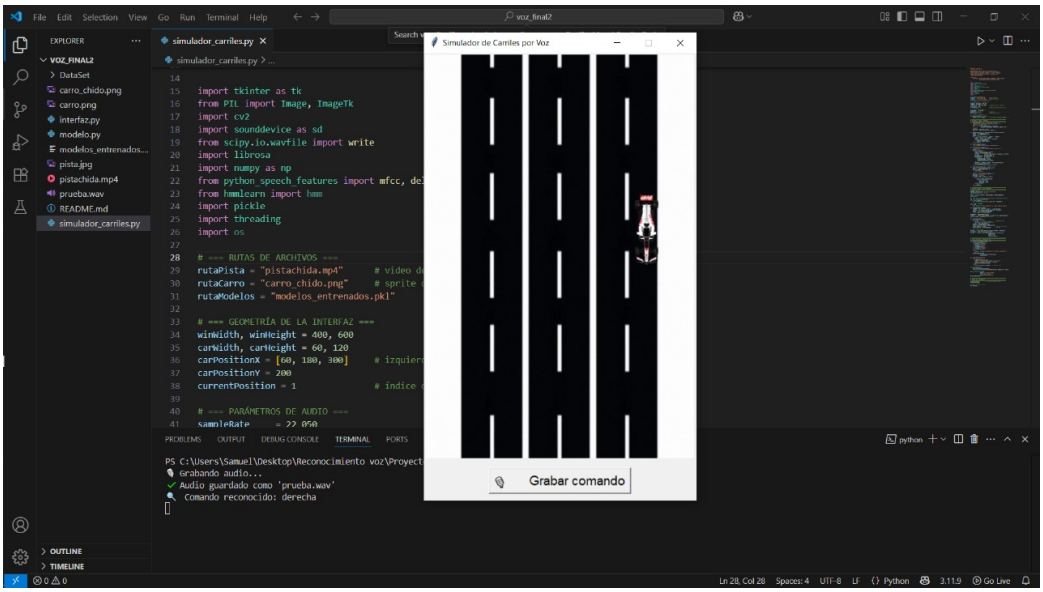


Figura 3: Derecha

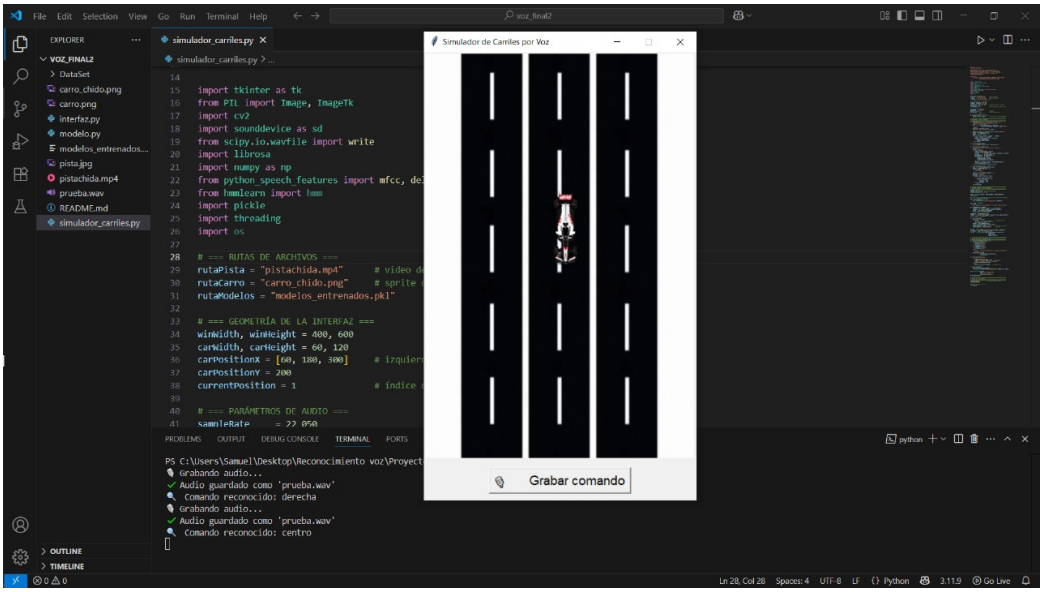


Figura 4: Centro

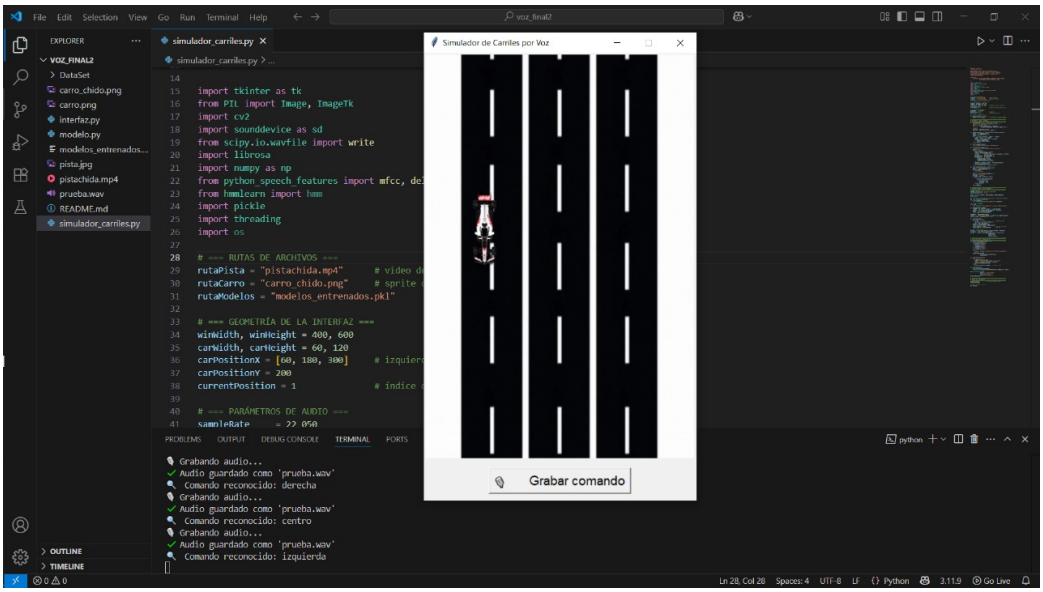


Figura 5: Izquierda

6. Conclusiones

Se cumplieron los objetivos específicos, así como el objetivo general. Se creo, entreno y evaluo un modelo de HMM para tener, en su totalidad, un sistema de reconocimiento automático de voz. Este sistema funciona correctamente al detectar los comandos con los que el modelo fue entrenado, sin embargo, tiene problemas para identificar otras voces, además de las del DataSet. Por ello, un área de oportunidad del proyecto sería ampliar el conjunto de audios para que reconozca de todo tipo de voces.

Referencias

[1] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989. doi: 10.1109/5.18626.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Prentice Hall, 2023. Versión en línea disponible en: https://web.stanford.edu/~jurafsky/slp3/