*UNIVERSIDAD DE LOS LLANOS*
*Facultad de Ciencias básicas e ingenierías*
*Departamento de Matemáticas y Física*

*INFORME DE*
*LABORATORIO III*
*ANÁLISIS DE SEÑALES*

# PROYECTO-ANALISIS DE SEÑALES

**M. Paula Correa [1], K. Santiago Pachón [2], L. Sebastián Beltrán [3], Jefferson Pérez [4], C. Andrés Villa [5]**

1. Cod: 161004710, Ing. Electrónica,
2. Cod: 161004724, Ing. Electrónica,
3. Cod: 161004704, Ing. Electrónica,
4. Cod: 161004727, Ing. Electrónica,
5. Cod: 161004635, Ing. Electrónica.

*Facultad de Ciencias Básicas e Ingenierías.*
*Ing. Electrónica.*

## RESUMEN

*El presente informe tiene como objetivo demostrar el paso a paso del proyecto final de Análisis de señales aplicando temas profundizados en clase, tales como la aplicación de la "Transformación rápida de Fourier" (FFT) a las imágenes obtenidas del mapa venoso y al espectro de voz de cada integrante del grupo, así como también se trabaja el tema de reconocimiento de figuras geométricas en mediante vértices. Para el desarrollo del proyecto se llevó a cabo el uso de una tarjeta Rasberry, en donde se programó mediante el lenguaje Python los códigos utilizados para lograr un correcto análisis de datos aplicando la FFT. El proyecto consta de 3 etapas: en la primera etapa se enfatiza el reconocimiento de voz; en esta etapa los componentes utilizados fueron una cámara, un brazo robótico, un micrófono y 2 figuras geométricas (cubo y esfera), lo que se busca en esta etapa es identificar la palabra que diga el usuario; la segunda etapa viene después de haber realizado el reconocimiento de voz, en el cual dependiendo de la palabra "cubo" o "esfera" el brazo robótico mediante una cámara identificara en que posición se encuentra la figura geometría y la tomará. Para el caso de la palabra "cubo" se activará la tercera etapa del proyecto que consta de reconocer el mapa venoso de cualquiera de los integrantes y mostrar el nombre en una pantalla LCD, por el contrario, si la palabra identificada es "esfera" se encenderá un motor DC.*

## INTRODUCCIÓN

Se implemento la transformada rápida de Fourier (FFT), la cual nos permite trabajar en el proyecto con mayor facilidad ya que se va a manejar una cantidad de datos muy grandes y lo que se quiere es tener una manera rápida y ligera para la máquina. Ecuación transformada discreta de Fourier (DFT):

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W_n^{nk} \qquad (1)$$

También usamos la fórmula que va a permitir calcular las energías y potencias. la ecuación de energía es la siguiente:

$$E = \frac{1}{N} \sum_{n=0}^{N-1} \|X(k)\|^2 \qquad (2)$$

Ecuación para hallar las potencias:

$$P = \frac{1}{N^2} \sum_{n=0}^{N-1} \|X(k)\|^2 \qquad (3)$$

Estas fórmulas se usaron con el propósito de hacer el reconocimiento de voz y el reconocimiento de imagen.
Se le añade que también se tienen en cuenta los espectros ya que estos permitieron ver cuáles son los niveles de intensidad que muestra cada energía tomada en el proyecto.

Espectro de amplitudes:

$$\|X(k)\| \; VS \; K\Omega_0 \qquad (4)$$

Espectro de la amplitud:

$$A_k = [\frac{1}{N} \|X(k)\| \qquad con\, K = 0,1,2, \ldots., N-1] \qquad (5)$$

## MARCO TEORICO

**1.** Transformada rápida de Fourier (FFT).

En el tratamiento digital de señales, el algoritmo FFT impone algunas limitaciones en la señal y en el espectro resultante ya que la señal muestreada y que se va a transformar debe consistir de un número de muestras igual a una potencia de dos1. La mayoría de los analizadores de FFT permiten la transformación de 512, 1024, 2048 o 4096 muestras [1] .
Las aplicaciones de la FFT son vastas y se encuentran en una variedad de campos. En el procesamiento de señales digitales, la FFT se utiliza para filtrar ruido, analizar frecuencias y codificar señales de audio y video. En la física, se utiliza para resolver ecuaciones diferenciales parciales. En la ingeniería eléctrica, se utiliza para el diseño y análisis de sistemas de filtrado.

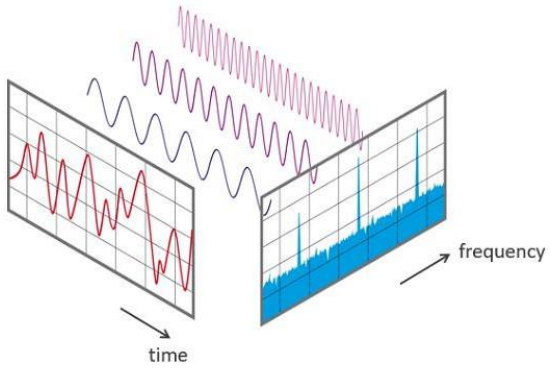



***Figura 1.*** *Aplicación de la FFT a una señal en el dominio del tiempo.*

**2.** Rapsberry Pi 4.

[2] La Raspberry Pi 4 es el último producto de la familia Raspberry Pi, con un procesador quad-core de 64 bits con 1,5 GHz, Red inalámbrica de banda dual 2.4/5 GHz, Bluetooth 5 / BLE, Ethernet más rápido (Gigabit y capacidad PoE a través de un PoE HAT separado.
La LAN inalámbrica de doble banda viene con certificación de cumplimiento modular, lo que permite que la placa se diseñe en productos finales con pruebas de cumplimiento de LAN inalámbrica significativamente reducidas, lo que mejora tanto el costo como el tiempo de comercialización.

***Figura 2.*** *Rasberry Pi 4.*

**3.** Librería OpenCV.

OpenCV, que significa Open Source Computer Vision (Visión Artificial Abierta), es una biblioteca de programación informática de código abierto desarrollada originalmente por Intel. Proporciona cientos de funciones para la captura, análisis y manipulación de datos visuales.
[3] OpenCV se ha utilizado en una gran cantidad de aplicaciones, desde sistemas de seguridad con detección de movimiento hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos3. Algunas de las áreas de aplicación de OpenCV incluyen características 2D y 3D, estimación de pose de cámara, reconocimiento facial, reconocimiento de gestos, interacción persona-computadora, robótica móvil, comprensión de movimientos, reconocimiento de objetos, segmentación y estereoscopía, entre otros. Esta gran biblioteca tiene interfaces para múltiples lenguajes, incluidos Python, Java y C++.

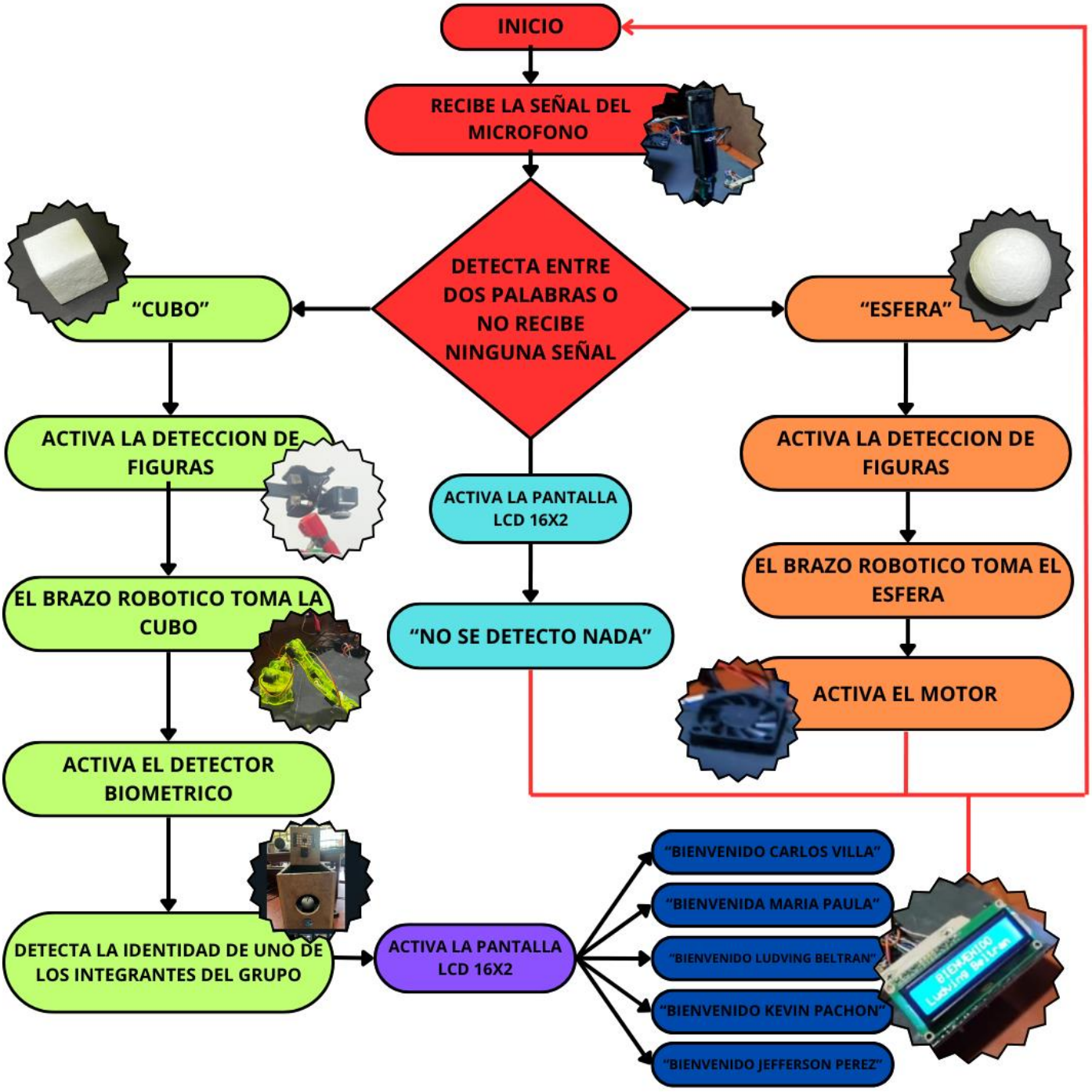
**4.** Pantalla LCD 16x2

[4] Este display tiene un tamaño de 16×2 que hace referencia a que la pantalla cuenta con 2 filas y cada fila tiene la capacidad de mostrar 16 caracteres o símbolos, por lo general alfanuméricos, los cuales se pueden definir desde programación utilizando un microcontrolador o tarjeta de desarrollo.


***Figura 3.*** *Pantalla LCD 16x2.*

## SECCION EXPERIMENTAL

**Diagrama de flujo:**




Para identificar y reconocer las figuras geométricas utilizadas se utilizó un brazo robótico impreso en 3D con 6 servomotores SG90 de 360° ensamblados a la estructura del brazo para garantizar un movimiento seguro, esos servomotores están ubicados desde la base hasta la garra del brazo y están controlados por el driver PCA9685, dicho driver va alimentado a 5V, tiene 16 puertos para conectar los servomotores y cuenta con pines SDL y SDA los cuales van conectados a la Raspberry en los pines GPIO 3 y 5, además de ello cuanta con un puerto GND que va conectado al GND de la Raspberry. Se utilizó una cámara webcam ubicada por encima del brazo robótico que abarcara una zona en donde se ubicaran las figuras geométricas para que la misma cámara reconozca la posición y el tipo de figura que esta

enfocando. Para este caso las figuras geométricas utilizadas fueron un cubo y una esfera de icopor, una vez reconocida la figura geométrica solicitada por voz, el brazo debe dirigirse hacia donde se encuentre y agarrarla.

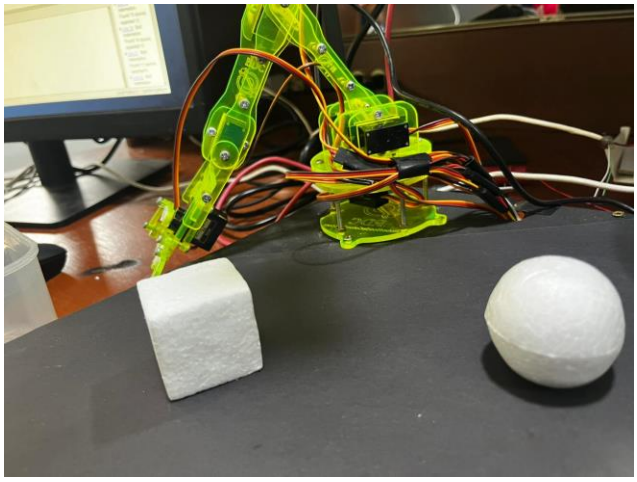


***Figura 4.*** *Brazo robótico, controlador de servomotores y figuras geométricas utilizadas.*

Para el caso de que el comando de voz sea “cubo” el brazo robótico recibe la información de la posición en donde se encuentra el cubo mediante la cámara y el controlador PCA9685, luego de ello se dirige a la posición recibida, la recoge y se activa el reconocimiento del mapa venoso.

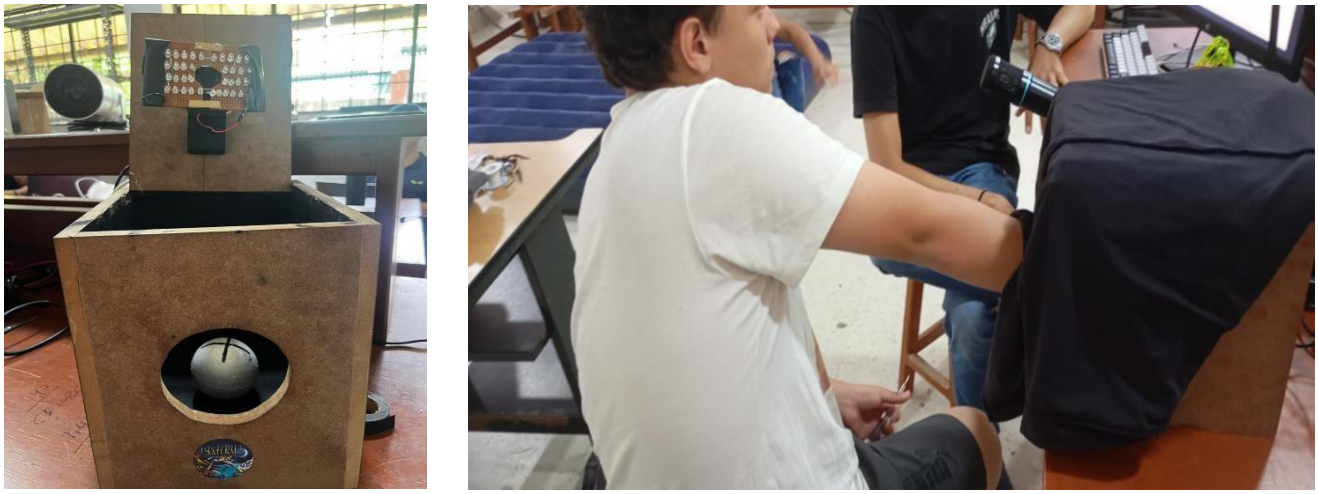


***Figura 5.*** *Caja de reconocimiento del mapa venoso.*

Para el reconocimiento del mapa venoso se tiene una caja que en el interior es totalmente oscura, tiene una cámara ubicada en la parte superior de la caja en donde también está rodeada por una matriz de leds infrarrojos, esto con el fin de que los leds iluminen la parte inferior y las venas de la mano se observen en la cámara, esa matriz de leds esta alimentada con 5V.

Luego de que el usuario haya ingresado la mano en la caja, el código implementado hace que la cámara registre 100 fotos para luego realizarles un proceso de filtro para que la imagen quede en blanco y negro, después de ello las imágenes pasan ahora por un proceso de suavizado de imagen para que las venas se observen con mayor claridad y así poder identificar el mapa venoso del usuario; se le aplica también un ecualizador de contraste para obtener que las partes oscuras de la imagen permanezcan oscuras y las partes grises logren obtener un color más claro.

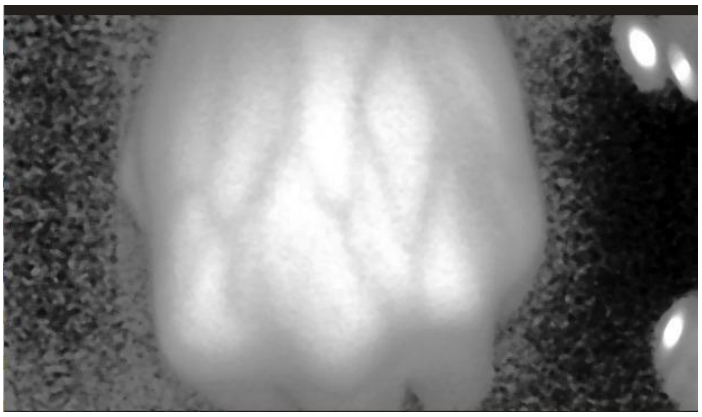


***Figura 6.*** *Resultado aplicación de filtros, suavizado y ecualizador de contraste.*

Las líneas de código utilizadas para obtener la imagen de esta manera son las siguientes:

Lo primero es realizar el llamado de la camara conectada a la Raspberry y tomar 100 capturas de la mano.

```
def pachon():
    cv2.namedWindow("captura")
    capture = cv2.VideoCapture(0 )
    photos=[]
    while True:
        ret,frame= capture.read()
        #time.sleep(2)
        if not ret:
            print("No se reconoce la camara")
            break
        cv2.imshow("captura",frame)
        if (cv2.waitKey(10)&0xFF==ord('c')):
            for i in range(50):
                cv2.imwrite("captura_venas.jpg",frame)
                photos.append(frame)
                cv2.destroyAllWindows()
                print("Picture taken_{}".format(i))
                tratamiento(frame)

            print("Energia total:\n", energias_totales)
            h=reorganization()
            break
```

Luego de reconocer la cámara y tomar 100 fotos para realizar el procesamiento evidenciado en la figura 6, se utilizan las siguientes líneas de código para realizar la escala de binarios y hacer el procesamiento de suavizado luego de aplicar el tratamiento morfológico de la imagen.

A continuación, se pueden ver que dividimos la imagen del mapa venoso en 9 cuadrantes cada uno de ellos tiene un valor de energía el cual se ve a continuación.

INTEGRANTE 1= Santiago Pachón

| | | |
|---|---|---|
| E1=11765.001953125 | E2=19818.896484375 | E3=15051.521484375 |
| E4=18395.921875 | E5=18775.16015625 | E6=7014.4775390625 |
| E7=111115.4775390625 | E8=16191.4833984375 | E9=21923.056640625 |

INTEGRANTE 2= Ludving

| | | |
|---|---|---|
| E1=13078.220703125 | E2= 20439.09375 | E3=10610.3447265625 |
| E4=20447.4609375 | E5=18672.416015625 | E6=5807.9990234375 |

| | | |
|---|---|---|
| E7=5363.58447265625 | E8=16022.8330078125 | E9=24098.271484375 |

INTEGRANTE 3= Jefferson

| | | |
|---|---|---|
| E1=11404.7724609375 | E2= 18808.29296875 | E3=16177.5185546875 |
| E4=23511.9916015625 | E5=18192.056640625 | E6=5941.51513671875 |
| E7=2872.433349609375 | E8=8 118.24755859375 | E9=20528.267578125 |

INTEGRANTE 4= Maria

| | | |
|---|---|---|
| E1=10444.515625 | E2= 20435.81640625 | E3=12889.08203125 |
| E4= 18919.77734375 | E5=18665.48828125 | E6= 4946.70361328125 |
| E7=2015.4368896484375 | E8=8 22542.62109375 | E9=20672.794921875 |

INTEGRANTE 5= Carlos

| | | |
|---|---|---|
| E1=9722.839484375 | E2= 19784.8457703125 | E3=13268.5537109375 |
| E4= 18907.759765625 | E5=17597.318359375 | E6= 5589.6337890625 |
| E7=7237.80322265625 | E8=10569.9091796875 | E9=20452.794921875 |

```python
def tratamiento(fotico):
    img= cv2.cvtColor(fotico,cv2.COLOR_BGR2GRAY) #Transformación de RGB a escalde grises.
    Filtromediana = cv2.medianBlur(img,5) #(imagen, grado de filtrado)
    cv2.imwrite("grises.jpg",img)
    cv2.imwrite("filtromediana.jpg",Filtromediana)
    suavizado=cv2.GaussianBlur(Filtromediana, (5, 5), 0.5)
    cv2.imwrite("suavizado.jpg",suavizado)
    equ = cv2.equalizeHist(Filtromediana)
    cv2.imwrite("nnnn.jpg",equ)

    sobel_image = cv2.Sobel(Filtromediana, cv2.CV_64F, 1, 0, ksize=5)  # Adjust ksize for detail control
    cv2.imwrite("sobel_image.jpg",sobel_image)
    thresh, binary_image = cv2.threshold(Filtromediana, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    cv2.imwrite("binary_image.jpg",binary_image)

    # Convert binary_image to a data type compatible with Sobel image (e.g., cv2.CV_32F)
    binary_image = cv2.convertScaleAbs(binary_image, alpha=1.0, beta=0.0)  # Convert to float32
    cv2.imwrite("binary_image22.jpg",binary_image)

    # Combine results with explicitly specified output data type (optional)
    combined_image = cv2.addWeighted(sobel_image, 0.5, binary_image, 0.5, 0, dtype=cv2.CV_32F)
    cv2.imwrite("combined_image.jpg",combined_image)

    kernel = np.ones((3, 3), np.uint8)
    imagen_morfologica = cv2.morphologyEx(combined_image, cv2.MORPH_CLOSE, kernel)
    cv2.imwrite("morfo.jpg",imagen_morfologica)
    normalizada=cv2.normalize(imagen_morfologica,None,0,1,cv2.NORM_MINMAX) #Normalizacion de contraste
    cv2.imwrite("normalizada.jpg",normalizada)
```

```python
    cutoff = 0.2
    fs=imagen_morfologica.shape[0] #o filtromediana
    filtered_image = apply_butterworth_filter(imagen_morfologica, cutoff, fs)
    cv2.imwrite("filtered_image.jpg",filtered_image)
```

El código expresado da como resultado las imágenes observadas en Figura 7 y 8.

Luego de haber usado el ecualizador de contraste, convertimos la imagen a escala de binarios y usamos un procesamiento de imágenes para detectar los bordes de la mano

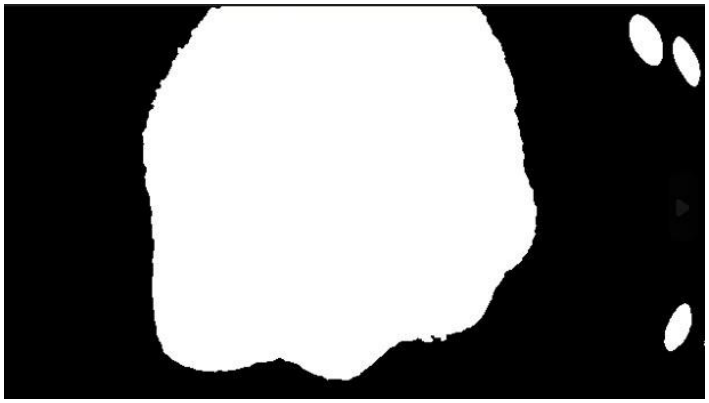


***Figura 7.*** *Detección de bordes de la mano.*

Luego de pasar las imágenes por los 2 procesamientos anteriores se hace un proceso final el cual consta de realizar un procesamiento de imágenes morfológicas. Las operaciones básicas en el procesamiento de imágenes morfológicas incluyen la erosión y la dilatación, así como la apertura y el cierre. La erosión reduce el tamaño de los objetos en una imagen al eliminar los píxeles en los bordes del objeto, mientras que la dilatación hace lo contrario, expandiendo los objetos al agregar píxeles a los bordes. La apertura y el cierre son combinaciones de erosión y dilatación. La apertura suaviza la imagen y elimina los píxeles aislados y los filamentos del objeto, mientras que el cierre puede eliminar agujeros y píxeles que ya se sabe que no están en el lugar correcto. Luego de obtener el resultado del ultimo procesamiento se combina con el resultado de las imágenes en escala de binarios para obtener el siguiente resultado.

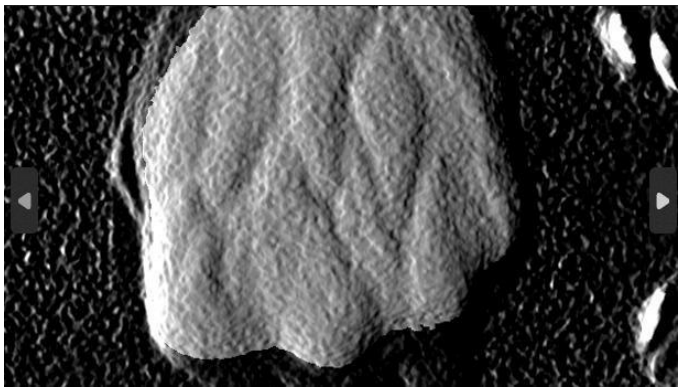


***Figura 8.*** Procesamiento de las imágenes morfológicas a normalizadas.

Una vez obtenido los resultados del procesamiento de imágenes, se aplica un filtro Butterworth pasa altas, para luego dividir cada una de las 100 imágenes en 4 sub-bandas. A cada sub-banda de la imagen se le haya la FFT y se le calculan las energías, una vez se obtengan las energías de cada sub-banda se promedian para que se obtenga un solo valor por cada sub-banda y a su vez se calculan las desviaciones estándar para así configurar los rangos dentro de los cuales las energías serán aceptadas para así identificar cada integrante. El código que realiza la acción descrita es el siguiente:

```python
def img_partition(img):
    energias=[]
    #Se obtienen las dimensiones de la imagen
    height, width = img.shape
    channels = 1

    #Se calcula la dimension de cada parte
    part_height = height // 2
    part_width = width // 2

    #Recorrer cada parte y guardarla
    for i in range(2):
        for j in range(2):
            # Calcular las coordenadas de la parte actual
            x = j * part_width
            y = i * part_height
            # Extraer la parte actual de la imagen
            part = img[y:y + part_height, x:x + part_width]
            # Guardar la parte en un archivo de imagen
            cv2.imwrite('part_{}_{}.jpg'.format(i, j), part)
            #Obtencion de energias de cada una de las partes de la imagen
            energias.append(calculate_energy(part))
    energias_totales.append(energias)
```

```python
def calculate_energy(img):
    suma = 0
    n = len(img)
    for i in range(0, n):
        suma = pow(abs(img[i]), 2) + suma
    energy = (1 / n) * suma
    return np.mean(energy)

def butter_highpass(cutoff, fs, order=5):
    nyq = 0.5 * fs  # Nyquist frequency
    normal_cutoff = cutoff / nyq  # Normalize cutoff frequency
    b, a = butter(order, normal_cutoff, btype='highpass')
    return b, a
```

Por último, se toma una sola foto cuando se vuelva a ingresar la mano a la caja y se le aplica el mismo procedimiento de filtro y procesamiento de imagen, dividiendo la imagen también en 4 sub-bandas y aplicando FFT a cada una de ellas para obtener sus energías, dichas energías serán comparadas con los promedios creados a base de los promedios y desviaciones estándar obtenidos de las primeras 100 fotos y así lograr identificar el mapa venoso que se está observando en ese instante.

Imagen mapa venoso (Procesamiento imágenes morfológicas) de los integrantes

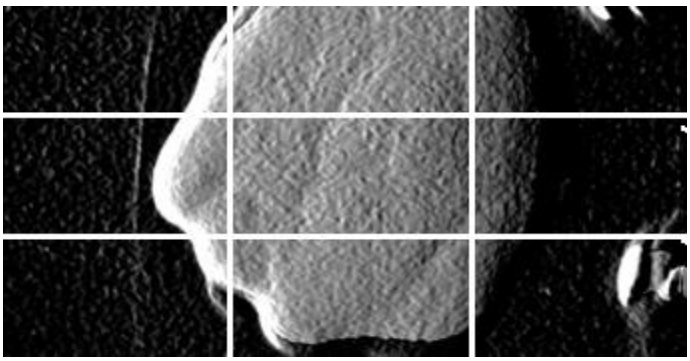


***Figura 9.*** Jefferson Pérez

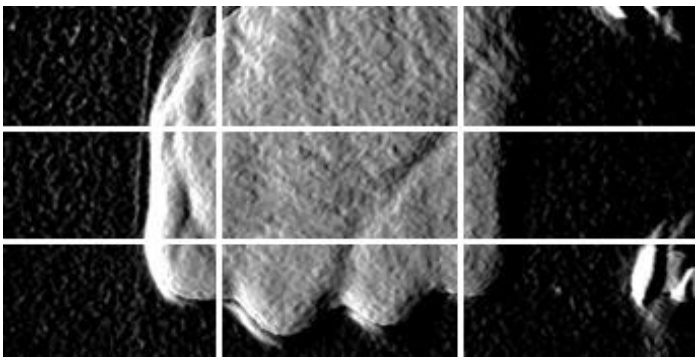


***Figura 10.*** Ludving Beltran

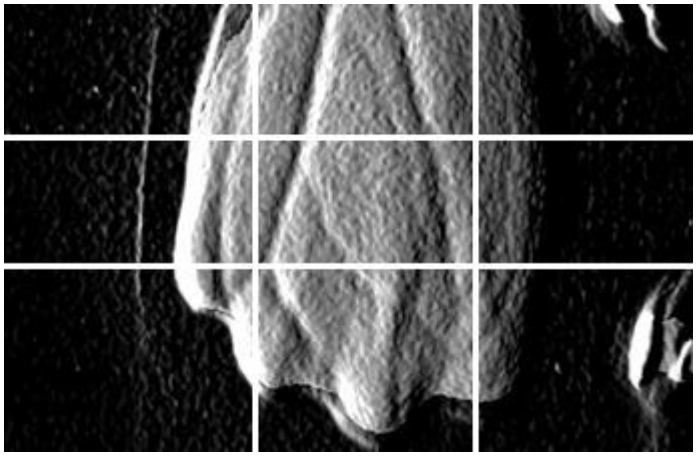


***Figura 11.*** Paula Correa.

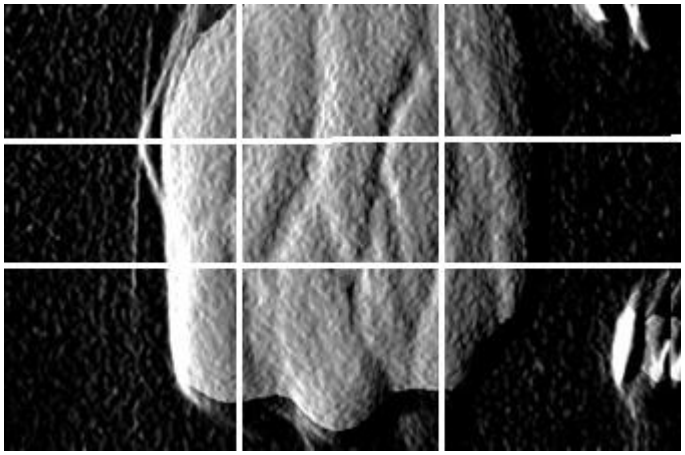


***Figura 12*** Santiago Pachon.

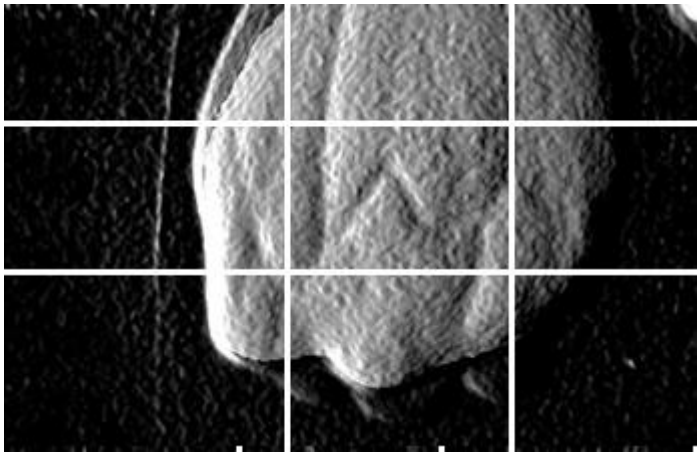


***Figura 13.*** Carlos Villa.

## RECONCOMIENTO DE VOZ

Para poder realizar el reconocimiento de voz se divide el proceso en 2 tareas diferentes, en la primera parte realizamos 100 grabaciones de dos segundos diciendo "cubo" y "esfera", cada grabación se almacena en arreglos de 44100 posiciones, a cada uno de los arreglos le calculamos la transformada rápida de Fourier y los dividimos en 10 partes iguales para a cada una de esas partes calcularle las energías.

```
# Configuración de la grabación
duration = 2  # Duración de la grabación en segundos
sample_rate = 44100  # Frecuencia de muestreo en Hz
channels = 1  # Número de canales (1 para mono, 2 para estéreo)
```

Se le calculan las energías a cada una de las 10 posiciones

```
fft_aquitoy = fft.calculate_fft(file_name)
p1, p2, p3, p4, p5, p6, p7, p8, p9, p10 = fft.split(fft_aquitoy)
em1 = fft.calculate_energy(p1)
```

De esta manera tenemos 100 arreglos de 10 posiciones, donde cada posición contiene las energías de una sección en específico del espectro grabado, posteriormente se promedian todos los 100 arreglos, de manera que se promedian todas las posiciones con índice 0, y así mismo todas las posiciones con el mismo índice, y a su vez calculamos las desviaciones estándar, teniendo así un arreglo con 10 promedios de energías y un arreglo con sus desviaciones estándar.

Una vez hecho eso, proseguimos con la segunda sección del proceso, para ello hacemos una sola grabación y la sometemos al mismo tratamiento hasta tener un arreglo de energías, dicho arreglo lo comparamos con el arreglo de los promedios de energías hallado previamente, si 6 de las 10 posiciones coinciden con el arreglo de promedios, decimos que las palabras coinciden.

- Repartición en Sub-bandas
  A cada comando se le calcula el espectro, se le toma el ancho de banda y se divide en 3 partes iguales, a cada parte se le hace la transformada rápida de Fourier (FFT) y se ingresa a un filtro pasa bandas. Posteriormente a

cada sub-banda se le calcula la energía para obtener una secuencia de 4 valores de energía por cada comando, se repite este proceso por cada una de las 100 grabaciones. Se gestiona una base de datos donde se promedian las energías de cada filtro de cada sub-banda y se obtienen las deviaciones de cada uno de los 10 promedios que nos servirán para hallar los umbrales que se toman como constantes para hacer la comparación en tiempo real con los comandos de entrada y así hacer el reconocimiento de voz.

Promedio de energía de cada comando
C1=esfera C2=cubo

C1= [13.275083059028 , 0.000393653707447738 , 0.0002272072458668789 , 0.0001854670085665495 , 0.00034089097793741006 ,0.00034089097793741006, 0.0001854545752634422 , 0.000227204500439741 , 0.00039637042565068 , 34.2461726215250 ]

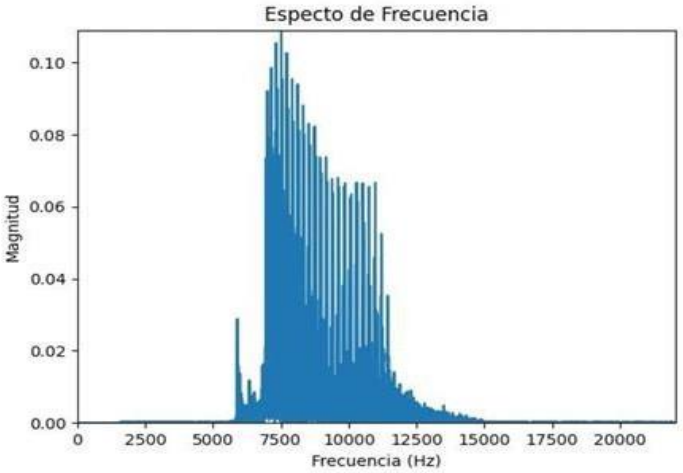
C2= [34.24805082446755 , 0.000436436130403396 , 0.0002411580611274326 , 0.0001934040702795968 ,0.000398808724012038 , 0.0000398672187062994 , 0.000193397172978084 , 0.0002411495852199998 , 0.000436413520856388 , 34.24617262152502 ]

Hay 20 muestras ya que tomamos tanto limite superior como limite inferior de cada uno de los promedios de los comandos.

L1= Esfera L2= Cubo

L1= [6.209850182725701, 20.3403159353311, 0.00024600843423416325, 0.0005412989806613134, 0.0001739856235585474, 0.0002804288681752105, 0.0001543077780358036, 0.0002166623909729555, -1.9132524514590533e-05, 8.731072010207254e-05, 2.920569063688884e-06, 6.523903012518084e-05, 0.0001739856235585474, 0.0002804288681752105, 0.0001543077780358036, 0.0002166623909729555, 0.0001739856235585474, 0.0002804288681752105, 0.0001543077780358036, 0.0002166623909729555]

L2=[17.66624718551186, 50.829854463423246, 0.00030907854572990794, 0.0005637937150768854, 0.0001937262339580003 2, 0.000288589888296865, 0.0001660046529968 2938, 0.0002208034875623 6434, -7.550954768228495e-06, 8.731269570636 18e-05, 1.2467801423531966e-05, 6.726663598906694e-05, 0.0001937262339580003 2, 0.000288589888296865, 0.0001660046529968 2938, 0.0002208034875623 6434, 0.0001937262339580003 2, 0.000288589888296865, 0.0001660046529968 2938, 0.0002208034875623 6434]



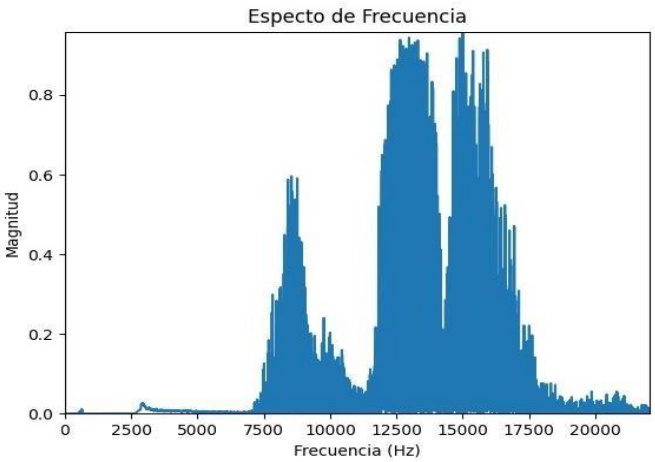




***Figura 14.*** Espectro de la frecuencia palabra cubo. ***Figura 15.*** Espectro de la frecuencia palabra esfera.

**ANEXOS**

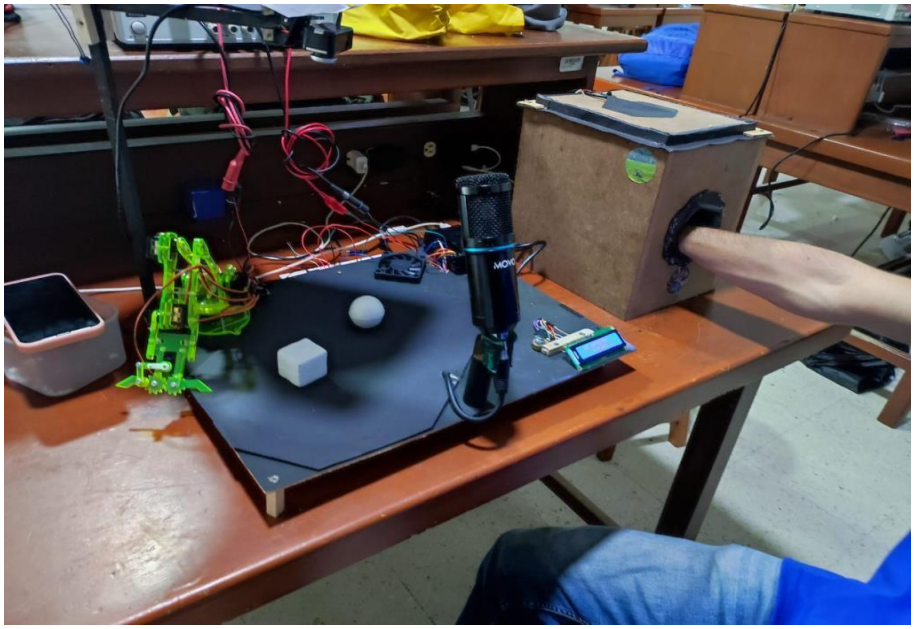


***Figura 16.*** *proyecto final.*

**CONCLUSIONES**

- La transformada rápida de Fourier es una herramienta matemática muy importante para este proyecto puesto que sin ella el análisis en frecuencia para el reconocimiento de voz sería mucho más lento y pesado para la Raspberry.
- Al hacer el reconocimiento de voz se observó que entre más grabaciones se tengan de cada comando el reconocimiento presenta un margen de error más pequeño y entre más divisiones tenga el ancho de banda el reconocimiento será más eficiente.
- Al realizar el proceso de reconocimiento del mapa venoso se evidencio que si las imágenes tomadas a la mano de los usuarios son en diferente posición o en un ambiente diferente podría generar una variación significativa, lo que quiere decir que el reconocimiento será erróneo y no reconocerá como debe.
- La aplicación de filtros digitales en este proyecto es una herramienta potencial que puede ayudar en el procesamiento de imágenes digitales.

**BIBLIOGRAFÍAS**


- [1] Svantek. (2023, 20 diciembre). Transformada rápida de Fourier FFT | Academia Svantek. SVANTEK - Sound And Vibration. https://svantek.com/es/academia/transformada-rapida-de-fourier-fft/#:~:text=Aplicaciones%3A%20Los%20analizadores%20FFT%20se,analizar%20el%20contenido%20de%20frecuencia
- [2] C Paguayo. (2019, 28 septiembre). Raspberry Pi 4 - Raspberry Pi. Raspberry Pi. https://raspberrypi.cl/raspberry-pi-4/#%3A~%3Atext%3DLa%20Raspberry%20Pi%204%20es%20el%20último%20producto%2CPoE%20a%20través%20de%20un%20PoE%20HAT%20separado
- [3] Darkcrizt. (2020, 7 enero). OpenCV una biblioteca para el reconocimiento de objetos en imágenes y cámaras. Desde Linux. https://blog.desdelinux.net/opencv-una-biblioteca-para-el-reconocimiento-de-objetos-en-imagenes-y-camaras/
- [4] LCD 16x2. (s. f.). https://hetpro-store.com/lcd-16x2-blog/#:~:text=Entonces%2C%20el%20t%C3%A9rmino%20LCD%2016x2,informaci%C3%B3n%2C%20por%20lo%20general%20alfanum%C3%A9rica.
- Fabian Velásquez. (2024). SecciónReconocimiento de voz. Análisis oProcesamiento de señales