Departamento de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
EL5202-2 Laboratorio de Sistemas Digitales

Informe Módulo 3

Conversión análogo digital y Extracción de características

Integrantes: Diego Campanini
Eduardo Hormazábal
Profesor: Helmuth Thiemer
Auxiliar: Carlos Navarro
Ayudante: Rodrigo Maureira
Fecha: 15 de noviembre de 2015

Índice

Índice de figuras

Índice de códigos

1. Introducción

La señal de los electrodos fue acondicionada en el módulo anterior, predispuesta para ser la entrada de este módulo. Con esto se procede a discretizar la señal con un conversor A/D mediante un Arduino, el que posibilita la salida muestreada para diferentes protocolos como SPI, RS232, etc. Esta facilidad hace que el muestreo y el procesamiento de la señal se pueda depurar, visualizándolo en un monitor Serial en un ordenador.

Una vez obtenida la conversión digital se procesa la señal digital en el Arduino Due **??**, que se encarga de extraer características útiles para el futuro entrenamiento del clasificador. Las características más usadas en los sistemas en base a electromiogramas son: Integral del Electromiograma (IEMG), Slope Sign Changes (SSC), Wafegorm length (WL), Amplitud de Willison (WAMP), Varianza (VAR), Skewness, Kurtosis, Desviación estándar, RMS, y Energía de las muestras.

También se obtiene la transformada de Fourier mediante un algoritmo alternativo que posee una librería para Arduino. El espectro de frecuencias se utilizará más adelante como otra característica, pues como es sabido entrega información relevante sobre el tipo de la señal.

2. Conversión A/D

Para muestrear la señal analógica proveniente del circuito anterior se utilizará el conversor A/D del Arduino. En particular el Arduino DUE posee un conversor análogo-digital de 12 bits, con una frecuencia de muestreo máxima de aproximadamente 25$[kHz]$. Esto hace posible muestrear la señal a una frecuencia mucho mayor que 500$[Hz]$, que es la la frecuencia utilizada en el *paper* en que está basado el proyecto [1].

Para visualizar la señal digitalizada por el Arduino se utilizará una comunicación serial entre Arduino y Ordenador, que permite obtener los valores medidos directamente en el espacio de trabajo del software, y así graficarlos rápidamente. El código a utilizar para depurar el proceso de envío de datos es el mostrado en el Código fuente 1, en donde el vector *times* corresponde al tiempo en microsegundos y el vector *val* corresponde a los valores obtenidos del ADC para N muestras.

Código fuente 1: Envío de datos por Serial

```
Serial.println("INICIO");
for (int j = 0; j<N; j++){
    Serial.print(times[j]);
    Serial.print(",");
    Serial.print(val[j]);
    Serial.println(";");
}
Serial.println("FIN");
```

Por otro lado, la conversión análoga-digital se realiza con la función *analogRead* de Arduino, a un periodo aproximado de 2 microsegundos, como se muestra en el Código fuente 2. Cabe destacar que se transforman los datos medidos por analogRead en torno a valores entre $-2{,}5$ y 2,5, que son formas de visualización ad'hoc a los voltajes análogos a la entrada del ADC. Además se observa que se calcula la media *mean* de los datos mientras éstos se obtienen, con el fin de disponer de ese parámetro para futuros cálculos.

Código fuente 2: Adquisición de datos

```
mean = 0;
while (i<N){
    time2 = micros();
    if (time2<time1+2200 && time2>time1+1800){
        time1 = micros();
        if (i<N){
            val[i] = analogRead(analogInPin)*5.0*0.25/1024 - 2.5;
            times[i] = time1;
            mean += val[i];
            i++;
        }
    }
}
mean = mean/N; // Media de los datos en [V]
```

3. Procesador de la señal

El procesador de la señal es el encargado de realizar las operaciones matemáticas necesarias para obtener una salida deseada. En particular se deben calcular distintas características de la señal en el dominio del tiempo, en donde la cantidad de muestras es igual al tiempo dedicado a muestrear por la frecuencia de muestreo, lo que significa una cantidad no menor de datos (500 datos por segundo para la frecuencia de muestreo teórica).

La alternativa a utilizar es un Arduino Due, pues posee un ciclo de reloj de $84[MHz]$ [3]. Además tiene una memora SRAM de $96[KB]$ (capaz de almacenar $96 \cdot 1000 \cdot 8/12 = 64000$ valores de 12 bits), que es lo suficientemente grande como para almacenar 6000 muestras en el caso en que se replique el tiempo de muestreo de [1].

Por otro lado Arduino Due cuenta con una librería de transformada de Fourier basada en el algoritmo *Split Radix FFT* [4]. Esta se puede utilizar para tamaños de datos de 2^k, con $k = 3, .., 11$. La librería se denomina SplitRadixReal FFT [5], en donde la operación para un arreglo de tamaño 2048 tarda aproximadamente 1,3$[ms]$, lo que es un tiempo razonable para el clasificador de movimientos.

4. Preprocesamiento de la señal

4.1. Descomposición de la señal

Una posible descomposición de la señal es la IMFs usando la descomposición de modo empírico para obtener una representación distinta de la señal original y de esta forma tener más datos, para esto, las filas de las matrices de datos obtenidos (ie las muestras recolectadas) se descompondrán en IMFs (Intrinsic Mode Functions) utilizando la Descomposición de Modo Empírico (EMD). El EMD es un método que permite descomponer cualquier conjunto de datos en un finito número de componentes (IMFs). Esta técnica puede ser aplicada en el dominio del tiempo a procesos no lineales y no estacionario, como sucede en este caso.

Una IMF es definida como una función que se caracteriza porque el número de mínimos y máximos, es igual o difiere en uno, con el número de veces que la IMF cruza el cero, además el promedio de la envolvente definida por el máximo y la definida por el mínimo debe ser cero.

No obstante, por simplicidad se propone aplicar la transformada de Fourier de la señal, con la que se obtendrá una descomposición ortogonal de la señal en el dominio de frecuencia. Cabe destacar que la IMF es un propuesto para futuros trabajos, por lo que no se descarta la efectividad de dicha descomposición.

4.2. Extracción de características

La extracción de características tiene un rol fundamental en los resultados del clasificador, incluso puede influir más en el resultado de la clasificación la selección de características que el tipo de clasificador que se utilice.

El objetivo de esta etapa es extraer información desde los datos recolectados, tal que, se pueda discriminar con el mínimo de error los movimientos a clasificar. Las características deben ser seleccionadas de tal forma que se maximice la separabilidad entre las clases y que se minimice la redundancia entre características.

En este trabajo se proponen una serie de características que serán extraídas desde la señal original y su primer IMF, en el domino del tiempo y en el de frecuencia. Para determinar la calidad de las características se medirá la correlación entre clases por cada característica y la correlación entre características. A continuación se mencionan las características más usadas en la implementación de un sistema en base a electromiogramas, junto al código Arduino que implementa la respectiva extracción,

- **Integral del Electromiograma (IEMG):** Es el valor promedio del valor absoluto del EMG, matemáticamente se expresa de la siguiente forma:

$$IEMG = \frac{1}{N} \cdot \sum_{k=1}^{N} |x_k| \tag{1}$$

  En la ecuación anterior x_k representa la k-ésima muestra de los datos obtenidos desde el EMG. En el Código fuente 3 se observa la implementación en Arduino.

Código fuente 3: Extracción de IEMG

```
double iemg = 0; // IEMG
for (int k = 0; k <N; k++)
{
    iemg += abs(val[k]);
}
iemg = iemg/N;
```

- **Slope Sign Changes (SSC):** Cuenta el número de veces que la pendiente de la señal cambia de signo. Para esto se toman tres muestras desde el EMG, dadas por x_{k-1}, x_k y x_{k+1}, luego el npumero de slope sign changes biene dado por $\sum f(x)$, con f(x) calculado como sigue:

$$f(x) = \begin{cases} 1 \text{ si } (x_k < x_{k+a} \ y \ x_k < x_{k-1}) \ o \ (x_k > x_{k+a} \ y \ x_k > x_{k-1}) \\ 0 \qquad\qquad\qquad\qquad \text{otro caso} \end{cases} \tag{2}$$

En el Código fuente 4 se observa la implementación en Arduino. Para este procedimiento se deben definir dos valores anteriores *val_prev* y *val_prev_prev* con los que se comparará la señal actual. También se considera la posibilidad de que el cambio de signo de la pendiente no sea entre solamente tres puntos, lo que implica la presencia de un *if* en el código.

Código fuente 4: Extracción de SSC

```
double val_prev = -2.5;
double val_prev_prev = -2.5;
int ssc = 0;  // Slope Sign Changes
for (int k = 0; k <N; k++)
{
    ssc += (val_prev<val[k] && val_prev<val_prev_prev) ||
    (val_prev>val[k] && val_prev>val_prev_prev);
    if (val_prev != val[k) // Caso en que la pendiente es nula
        val_prev_prev = val_prev;
    val_prev = val[k];
}
```

- **Waveform length (WL):** Es una variación acumulativa del EMG que puede indicar el grado de variación de la señal del EMG, esta se calcula como sigue:

$$WL \ = \ \sum_{k=1}^{N-1} (|x_{k+1} - x_k|) \tag{3}$$

En el Código fuente 5 se observa la implementación en Arduino. Se observa que se debe usar un valor anterior *val_prev* de la misma forma que para la extracción del SSC, lo que significa que estos valores se podrían compartir al momento de integrar todas las características.

Código fuente 5: Extracción de WL

```
1 double val_prev = -2.5;
2 double wl = 0; // Waleform Length
3 for (int k = 0; k <N; k++)
4 {
5     wl += abs(val[k]-val_prev);
6     val_prev = val[k];
7 }
```

- **Amplitud de Willison (WAMP):** Es el número de veces que la diferencia de amplitud de la señal EMG entre dos muestras, excede un determinado umbral. Lo anterior se puede escribir como $WAMP(x) = \sum_{k=1}^{N-1} f(|x_{k+1} - x_k|)$, con f dado por:

$$f(x) = \begin{cases} 1 \text{ si } x > \text{threshold} \\ 0 \qquad \text{otro caso} \end{cases} \tag{4}$$

En el Código fuente 6 se observa la implementación en Arduino. De la misma forma que WL, también es posible compartir el valor anterior *val_prev*. En este caso se debe definir un umbral, el que por defecto es $thr = 0{,}1$.

Código fuente 6: Extracción de WAMP

```
1 double val_prev = -2.5;
2 double thr_wamp = 0.1; // umbral de Willison
3 int wamp = 0; // Amplitud de Willison
4 for (int k = 0; k <N; k++)
5 {
6     wamp += abs(val[k]-val_prev)>thr_wamp;
7     val_pref = val[k];
8 }
```

- **Varianza:** Es una medida de la densidad de potencia del EMG, destacar que la varianza es muy sensible a los *outliers*. La varianza se puede calcular como sigue:

$$VAR \quad = \quad \frac{1}{N-1} \sum_{k=1}^{N} (x_k)^2 \tag{5}$$

En el Código fuente 7 se muestra la implementación en Arduino. Se observa que para este estadístico es necesario utilizar la media calculada previamente en el muestreo de datos del conversor análogo-digital.

Código fuente 7: Extracción de Varianza

```
1 double var = 0;  // Varianza
2 for (int k = 0; k <N; k++)
3 {
4     var += (val[k]-mean)*(val[k]-mean);
5 }
6 var = var/(N-1);
```

- **Skewness:** Es una medida de la simetría de la función de densidad de probabilidad en torno a un promedio. La asimetría puede tomar valores positivos, negativos, cero e incluso o indefinido. Una distribución simétrica tiene skewness igual a cero, si tiene una cola hacia la izquierda el skewness es negativo y si tiene una cola hacia la derecha el skewness es positivo, lo anterior se muestra en la figura 1. El cálculo de esta característica se puede realizar de la siguiente forma:

$$S = \frac{E(x-\mu)^3}{\sigma^3} \tag{6}$$

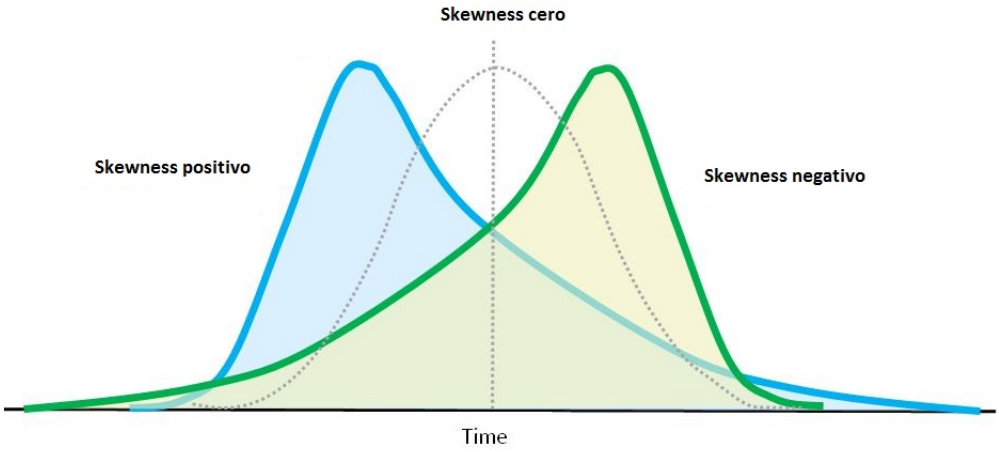


Figura 1: Representación del skewness de tres distribuciones

En el Código fuente 8 se muestra la implementación en Arduino. Se observa que para este estadístico es necesario utilizar la media y posteriormente se debe normalizar el Skewness por la varianza. Esto es posible realizar en conjunto con el cálculo de la varianza, pues esta última solamente se necesita al finalizar el ciclo *for*, en donde ya se obtuvo la varianza.

Código fuente 8: Extracción de Skewness

```
1 double ske = 0;  // Skewness
2 for (int k = 0; k <N; k++)
3 {
4     ske += (val[k]-mean)*(val[k]-mean)*(val[k]-mean);
5 }
6 ske = ske/(N*pow(var,1.5));
```

- **Kurtosis:** Es una medida de la forma de la función de distribución, indica que tan achatada o puntiaguda es una distribución (figura 2), una forma de calcular esto es la siguiente:

$$S = \frac{E(x-\mu)^4}{\sigma^4} \tag{7}$$

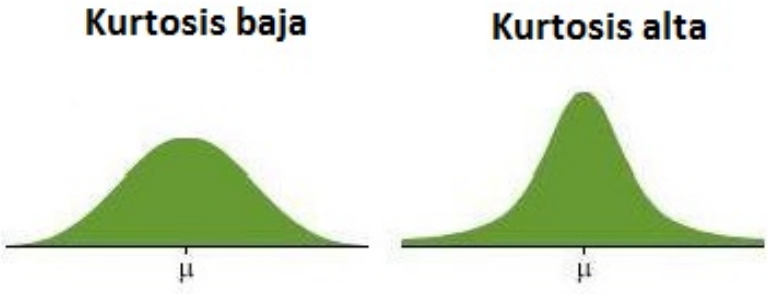


Figura 2: Kurtosis baja y alta

En el Código fuente 9 se muestra la implementación en Arduino. Se observa que para este estadístico se realiza un procedimiento similar al del cálculo del Skewness, por lo que es posible integrarlo en un mismo ciclo *for* junto a las demás características.

Código fuente 9: Extracción de Kurtosis

```
1 double kur = 0;  // Kurtosis
2 for (int k = 0; k <N; k++)
3 {
4     kur += (val[k]-mean)*(val[k]-mean)*(val[k]-mean)*(val[k]-mean);
5 }
6 kur = kur/(N*(var*var));
```

- **Desviación estándar:** La desviación estándar es la raíz cuadrada de la varianza. Puede ser interpretada como una medida de incertidumbre. Matemáticamente se puede expresar de la siguiente forma:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(x_k - \overline{x})^2} \tag{8}$$

La desviación estándar es una característica dependiente de la varianza, por lo que no se realiza un cálculo de ésta. Se debe recordar que el objetivo de la extracción de características es obtener valores con la mayor independencia posible para lograr un clasificador robusto.

- **RMS:** *Root Mean Square* es una medida estadística definida como la raíz cuadrada de la media aritmética de los cuadrados de los valores. Lo anterior matemáticamente se define como:

$$x_{rms} = \sqrt{\frac{1}{N}\cdot\sum_{k=1}^{N}(x_k)^2} \tag{9}$$

El RMS es una característica correlacionada con la energía, por lo que es necesario calcular sólo una de ellas para efectos del clasificador a construir más adelante. En particular se calculará la energía.

- **Energía para todas las muestras:** Se calcula la energía de todos los datos de la muestra, originalmente se pretende contar con 6000 muestras, ya que, se muestreará por 6 s a 500 Hz. La energía se determina como sigue:

$$E = \sum_{k=1}^{N} |x_k|^2 \tag{10}$$

En el Código fuente 10 se muestra la implementación en Arduino.

Código fuente 10: Energía de todas las muestras

```
double ene_muestra = 0; // Energia de una muestra
double ene = 0; // Energia total
for (int k = 0; k <N; k++)
{
    ene_muestra = val[k]*val[k];
    ene += ene_muestra;
}
```

- **Energía acumulada por intervalos:** Se usará análisis por ventana deslizante sin solapamiento, esto quiere decir, que se calculará la energía por intervalos de muestras, por ejemplo, si se tienen 6000 muestras, se considerará el cálculo de energía cada 300 muestras.

  La energía acumulada se puede definir como una fracción de la energía de todas las muestras, es decir para un intervalo de tiempo menor. Es por esto que se puede definir en conjunto con la energía total, como se muestra en el Código fuente 11.

Código fuente 11: Energía por intervalos

```
const int N_ENE = 5;    // Numero de intervalos
double ene_int[N_ENE];  // Energia por intervalos
double ene_muestra = 0; // energia de una muestra
for (int k = 0; k <N; k++)
{
    ene_muestra = val[k]*val[k];
    ene_int[(int)k*N_ENE/N] = ene_muestra;
}
```

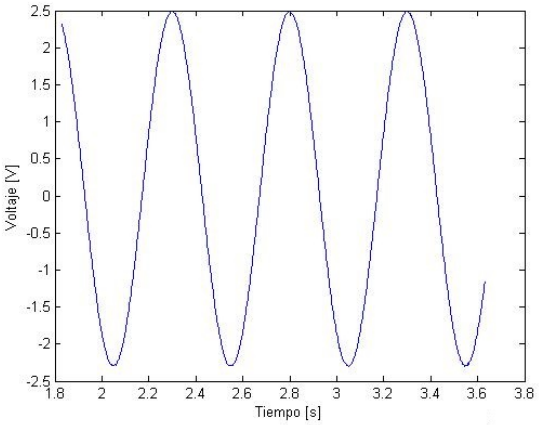
5. Resultados

Para comprobar la correcta conversión análogo-digital del Arduino y las extracciones de características, se generaron un conjunto de señales de baja frecuencia mediante el generador de señales. Sin embargo todas las señales deben tener características en común por las restricciones del ADC.

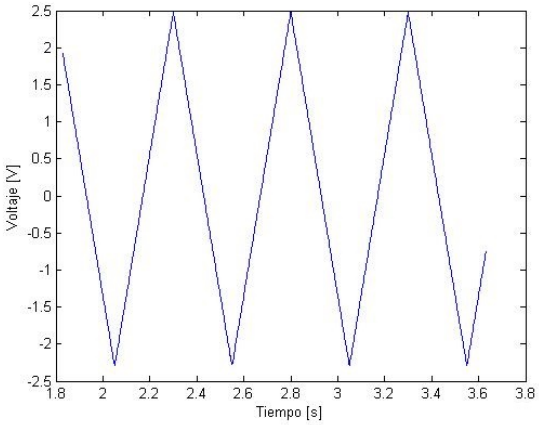
Como la frecuencia de muestreo utilizada es de $500[Hz]$, se debe tener en consideración señales de frecuencia menor o igual a $250[Hz]$ por consecuencia del teorema de muestreo de Nyquist-Shannon. La cantidad de puntos por señal es asignada a $N = 1000$. En particular para una buena observación de los datos graficados posteriormente, las señales generadas tienen una frecuencia de no más de $2[Hz]$

Por otro lado se debe tener en consideración que el ADC del Arduino DUE soporta voltajes entre 0 y 3,3$[V]$, por lo que las señales se deben escalar a 3,3$[Vpp]$, con un offset de 1,7$[V]$. Una vez realizado este acondicionamiento el Arduino puede transformar los bits entregados por el ADC a la escala que se desee. En particular como se observa en la Sección 2, Código fuente 2, el voltaje se escala a $5[V]$.

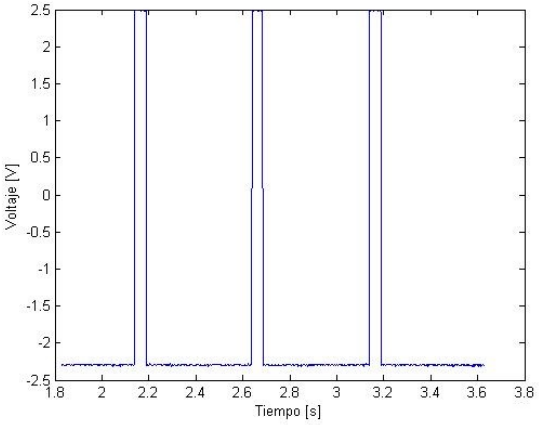
En la Figura 3 se muestran 4 señales diferentes a las que se sometió el Arduino. Se observa que la escala de voltaje es la esperada con respecto a la normalización de la señal efectuada por el código Arduino. Además la resolución del ADC basta como para identificar señales típicas.



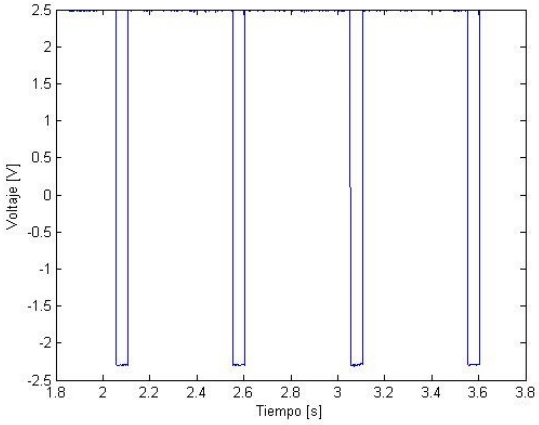
(a) Sinusoide



(b) Rampa



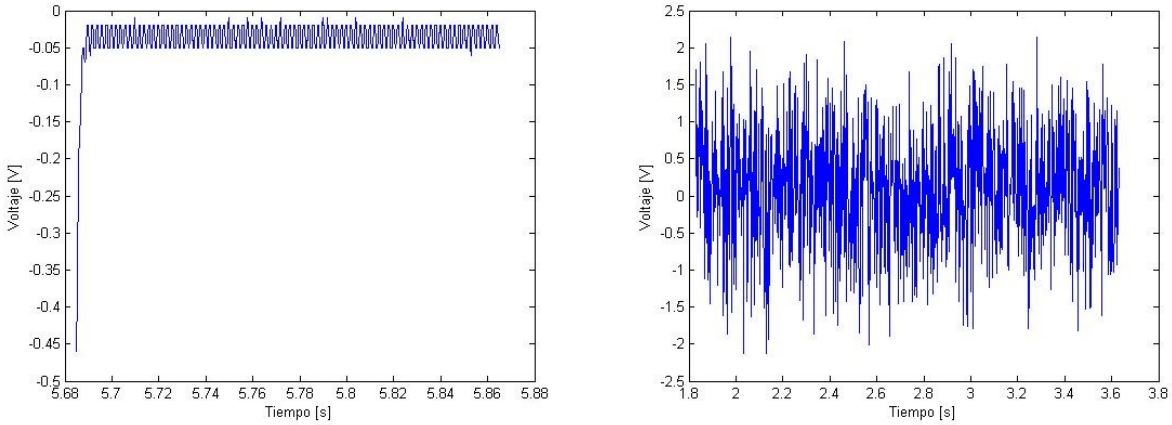
(c) Pulso de duración del 10 %



(d) Pulso de duración del 90 %

Figura 3: Entradas de señales de 2 [Hz] al ADC

Por otro lado, en la Figura 4 se muestran señales ruidosas. En primera instancia la Figura 4(a) corresponde al ADC con entradas en circuito abierto. En ella se observa que el voltaje medido varía entre los 50$[mV]$ y 0$[mV]$, lo que significa una energía total relativamente pequeña. Por otro lado la Figura 4(b) corresponde a una entrada ruidosa de amplitud pico-pico de 3,3$[V]$ y un offset de 1,7$[V]$.



(a) Ausencia de entrada al ADC

(b) Ruido Blanco

Figura 4: Entradas de Ruido al ADC

En el Cuadro 1 se muestran las principales características de cada una de las señales anteriores. A simple vista se observa que cada señal tiene características particulares que las diferencian entre ellas, lo que significa que hacer un clasificador para este tipo de señales es factible. En la etapa de entrenamiento, del siguiente módulo, se realizará la extracción de características para entradas obtenidas con los electrodos, ante distintos movimientos de la mano.

Cuadro 1: Características de distintas señales de frecuencia 2 [Hz]

	IEMG	**SSC**	**WL**	**WAMP**	**Media**	**VAR**	**Skewness**	**Kurtosis**	**Energía**
Sinusoide	1.53	28	39.47	0	-0.03	2.86	0.10	1.50	2857.57
Rampa	1.20	8	38.78	0	-0.01	1.90	0.08	1.79	1895.21
Pulso 10 %	2.31	529	33.75	6	-1.90	1.74	3.02	10.12	5350.05
Pulso 90 %	2.47	511	47.83	8	1.95	2.28	-2.46	7.04	6079.83
Ruido CA	0.04	178	8.60	0	-0.04	0.00	-8.38	107.25	2.21
Ruido blanco	0.67	668	946.33	936	0.10	0.67	-0.10	2.59	682.80

Para comprobar que el cálculo de características implementado en Arduino estuviera correcto, se procedió a obtener las características del cuadro 1, para la misma entrada sinusoidal, obteniendo valores similares, por ejemplo para IEMG se obtuvo 1.53 con Arduino y 1.5256 con Matlab, difiriendo en los decimales, debido a que Arduino aproxima sus resultados numéricos, con menor representación decimal que Matlab. Análogamente, cuando se le aplicó transformada de Fourier a la sinusoide de entrada y luego se extrajeron las mencionadas características, se obtuvieron similares resultados en Matlab y Arduino. El código con el cuál se calcularon las características en Matlab se encuentra en anexos.

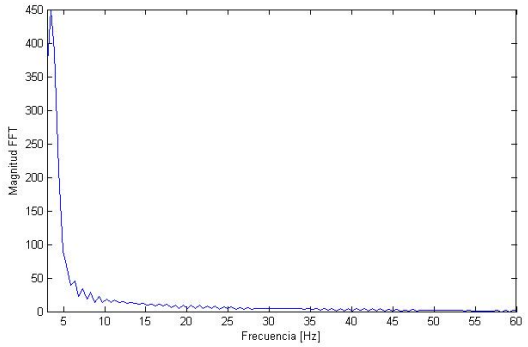
Por otro lado se calculó la transformada de Fourier en Arduino con la librería SplitRadixReal FFT. En la Figura 5 se muestran tres espectros de Fourier obtenidos a partir de los valores entregados por Arduino.

La Figura 5(a) corresponde a la FFT de la sinusoide de la Figura 3(a), en donde se observa un *peak*

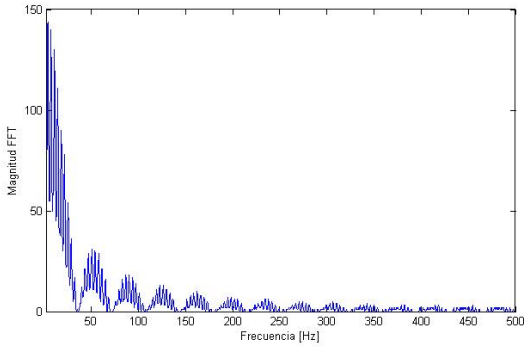
de 3,4$[Hz]$ cercano a la frecuencia original de 2$[Hz]$. Además la presencia de frecuencias altas es despreciable, lo que se espera de una señal sinusoidal limpia.

La Figura 5(b) corresponde a la FFT del pulso de la Figura 3(c), en donde se observa un peak cercano a la frecuencia principal de 2$[Hz]$. También aparece la función *sinc* típica de los pulsos, lo que demuestra una vez más la efectividad de la FFT.

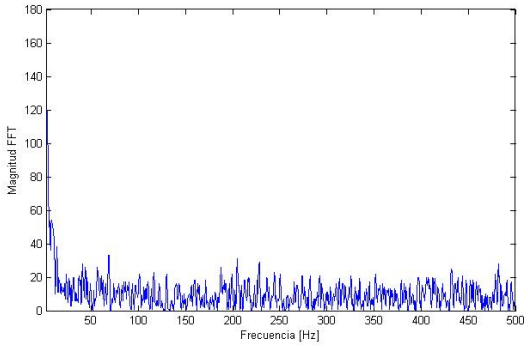
Por último la Figura 5(c) corresponde a la FFT del ruido blanco de la Figura 4(b). Se observa que la señal posee una gran cantidad de frecuencias a lo largo de todo el ancho de banda, lo que es característico del espectro de Fourier del ruido blanco.



(a) Espectro de Fourier sinusoide de 2 Hz



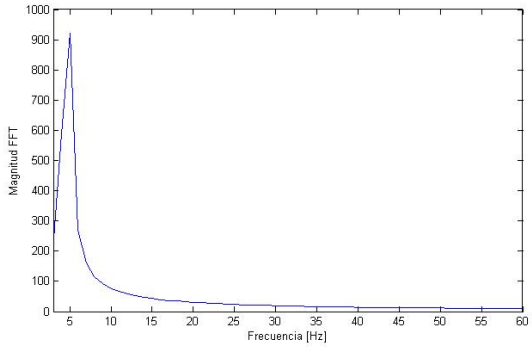
(b) Espesctro de Fourier pulso de 2 Hz de duración del 10 %



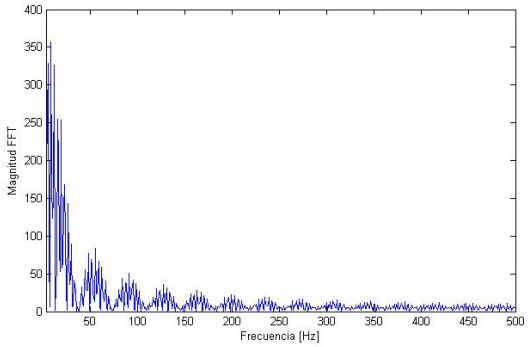
(c) Espectro de Fourier de ruido blanco

Figura 5: Espectros de Fourier de difererentes señales de entrada al Arduino

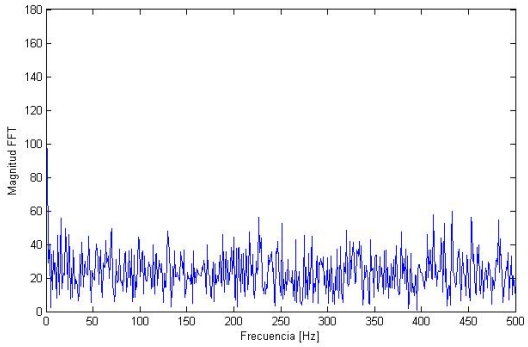
Con el fin de verificar de otra forma la efectividad de la librería de la transformada de Fourier, se calculó en Matlab la Fast Fourier Transform (FFT) para las tres señales anteriores (Figura 6). Claramente se observa que los espectros de Fourier tanto de Arduino como de Matlab son similares para las tres señales de muestra, lo que comprueba la efectividad de la transformada de Fourier implementada en Arduino.



(a) Espectro de Fourier de Matlab sinusoide de 2 Hz



(b) Espesctro de Fourier de Matlab pulso de 2 Hz de duración del 10 %



(c) Espectro de Fourier de Matlab de ruido blanco

Figura 6: Espectros de Fourier calculados con FFT de Matlab para difererentes señales de entrada al Arduino

6. Conclusión

Para ingresar la señal analógica acondicionada al conversor A/D debemos asegurarnos de que el rango de valores posibles de la señal esté comprendido entre los valores de voltaje que acepta el conversor $(0 - 3{,}3[V])$, pues de lo contrario se podrían presenciar conversiones saturadas que inducirían

Otro detalle a considerar es el correcto funcionamiento del entorno de programación de Arduino, para evitar la presencia de errores en tiempo de ejecución. Es por esto que siempre se debe tener en consideración los límites del código diseñado.

Una de las desventajas del método empleado de extracción de características es que es completamente procedural. Es decir que primero se dedica a obtener la cantidad de datos necesarios, y una vez obtenidos los N valores se procede a extraer características. Es por esto que se propone como trabajo futuro mejorar el código del Arduino, pues claramente es posible disminuir el tiempo total que requiere para generar características, como por ejemplo utilizando una metodología *pipeline* entre lectura de datos y cálculo de características.

Bibliografía

[1] Sapsanis, C.; Georgoulas, G.; Tzes, A., EMG based classification of basic hand movements based on time-frequency features,in Control & Automation (MED), 2013 21st Mediterranean Conference on , vol., no., pp.716-722, 25-28 June 2013

[2] Rahman K K, M.; Nasor, M.,Multipurpose Low Cost Bio-Daq System for Real Time Biomedical Applications, 2015 International Conference on Information and Communication Technology Research (ICTRC2015)

[3] `https://www.arduino.cc/en/Main/ArduinoBoardDue`

[4] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm,.[El]ectron. Lett. 20 (1), 14–16 (1984).

[5] `https://coolarduino.wordpress.com/2014/09/25/splitradixreal-fft-library`

7. Anexos

Código fuente 12: Cálculo Amplitud de Willison

```
function [ nwa ] =wa( simfs )
%Calcula Willison Amplitude (WAMP)
nwilla=0;
fc=size(simfs);
nwa=zeros(fc(1),1);
for i=1:fc(1)
    m=abs(diff(simfs(i,:)));
    for p=1:length(m);
        if m(p)>0.1
            nwilla=nwilla+1;
        end
    end
    nwa(i,:)=nwilla;
    nwilla=0;
end
end
```

Código fuente 13: Cáculo Waferom Length

```
function [ nwlen ] = wlen( simfs )
% Calcula el Waveform Length
fc=size(simfs);
nwlen=zeros(fc(1),1);
for i=1:fc(1)
    m=diff(simfs(i,:));
    nwlen(i,:) =sum(abs(m)) ;
end
end
```

Código fuente 14: Cálculo slope sign change

```
function [ nssc ] = ssc(simfs)
count=0;
fc=size(simfs);
nssc=zeros(fc(1),1);
for i=1:fc(1)
    m=diff(simfs(i,:));
    for p=1:length(m)-1
        if (m(p+1)>0 && m(p)<0) || (m(p+1)<0 && m(p)>0)
            count=count+1;
        end
    end
    nssc(i,:)=count;
    count=0;
end
end
```

Código fuente 15: Cálculo energía

```
function [ tenergy ] =energy(simfs)
fc=size(simfs);
tenergy=zeros(fc(1),1);
for i=1:fc(1)
    mm=abs(simfs(i,:));
    tenergy(i,:)=sum(mm.^2);
end
end
```

ón estandar

Código fuente 16: Cálculo media

```
charac(:,1)= mean(abs(simfs),2); % Integrated Electromyogram (IEMG)
charac(:,6)= var(simfs,0,2); % Variance (VAR)
charac(:,7)= skewness(simfs')'; % Skewness
charac(:,8)=kurtosis(simfs')'; % Kurtosis
charac(:,9)=median(simfs,2); % median
charac(:,10)= std(simfs,0,2); % Standard Deviation
```