



# **Análisis y planteamiento de la solución del desafío I**

Faith Alvarado Silva

Carla Zapata Valencia

Profesor: Anibal Guerra

Informática II

Septiembre 14, 2024

## **ÍNDICE**

1. Introducción
2. Marco teórico
3. Análisis del problema y alternativa propuesta
4. Variables y tipo de datos
5. Estructura de datos y uso de memoria dinámica
6. Funciones implementadas para determinar las características de la onda
7. Tabla de margen de error
8. Montaje del Arduino en Tinkercad
9. Diagrama de flujo del funcionamiento del programa

## **Introducción**

El presente informe detalla el desarrollo de un sistema de adquisición y análisis de señales utilizando Arduino, como parte del Desafío I de la asignatura Informática II. En este proyecto se busca implementar un algoritmo capaz de capturar y procesar señales, identificando características clave como la forma de onda, amplitud y frecuencia.

La señal es producida a través de un generador de señales conectado al Arduino. La lectura de los datos inicia una vez el pulsador de adquisición es presionado, el usuario debe detener la lectura de datos con el segundo pulsador e inmediatamente se mostrará por medio de una pantalla LCD el resultado de la lectura.

Para realizar el procesamiento de la señal, se implementarán funciones que determinarán la forma de la onda entre tres opciones establecidas, una onda senoidal, triangular o cuadrada, así como su frecuencia y amplitud. En caso de presentarse una forma de onda desconocida se mostrará un mensaje en la pantalla LCD.

A lo largo de este informe se realiza una conceptualización teórica, se describe el análisis realizado y la solución propuesta para el desarrollo del proyecto.

## **Marco teórico**

Una señal es una función de una o más variables que describen un fenómeno físico. Transportan información entre sistemas de comunicación, caracterizándose por tener una frecuencia, amplitud y forma.

La frecuencia indica la cantidad de ciclos que completa por segundo y se mide en Hertz (Hz). Por otro lado, la amplitud de onda es la distancia que existe entre el valor máximo que registra una variable, midiéndose desde el punto medio o de equilibrio. En algunos casos la onda puede estar desfasada (offset) de la línea de equilibrio.

Existen diversas formas que puede tomar una señal, entre ellas, senoidales, triangulares y cuadradas, las cuales se evaluarán y analizarán en este caso. Por ejemplo, en la figura 1 se muestran las características de una onda senoidal, donde su frecuencia se mide en Hertz (Hz), su amplitud en voltios (V) y el desfase u offset en voltios (V).

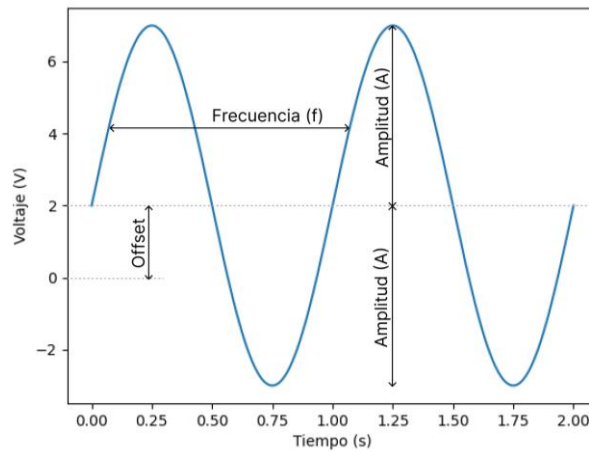


Figura 1. Características de una onda senoidal

Por otra parte, las ondas triangulares son un tipo de señal periódica, habitualmente son simétricas y las transiciones entre los niveles de voltaje de estas ondas cambian a una tasa constante. Estas transiciones se denominan rampas. En la figura 2 se observa una onda triangular.

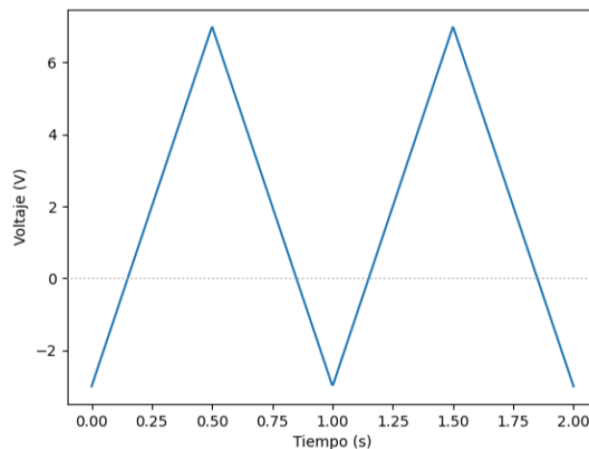
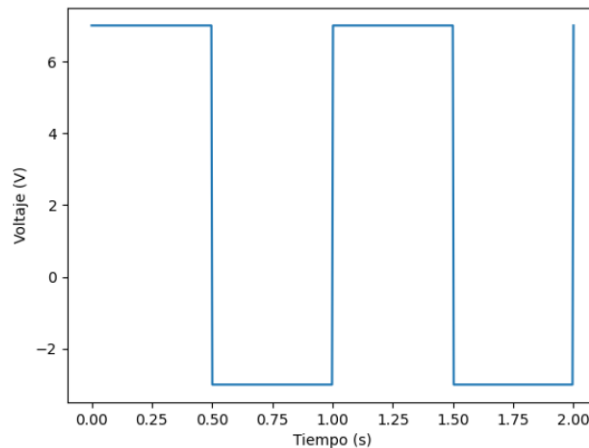


Figura 2. Características de una onda triangular

Las ondas cuadradas son aquellas que solo alternan su valor entre dos puntos extremos, a intervalos regulares, en un tiempo muy reducido. Son comúnmente usadas para la generación de pulsos eléctricos, utilizados como señales (1 y 0) que permiten ser manipuladas fácilmente. Un circuito electrónico que genera ondas cuadradas se conoce como generador de pulsos y son la base de la electrónica digital. En la figura 3 se observa una onda cuadrada.



*Figura 3. Onda cuadrada.*

## **Análisis del problema y alternativa propuesta**

Para resolver el problema se analizaron distintas alternativas; descripción a partir de las pendientes, de los máximos y mínimos, ajustes de curvas o de series de Fourier. Con las dos primeras opciones se observó que las respuestas eran muy sensibles a los puntos considerados para su determinación. Por ejemplo, en el caso de las ondas triangulares, el cálculo de la pendiente presenta pequeñas variaciones que podían confundirse con una onda senoidal o la determinación de los máximos y mínimos se ve afectada por la discontinuidad de la pendiente. De igual manera estas dificultades se observaron con las ondas cuadradas. Para la opción de series de Fourier, un análisis espectral para filtrar el ruido de la onda principal se escapa del objetivo de este trabajo.

De esta forma, dado que los valores de voltaje suministrados por el generador de señales en el Arduino son valores discretos obtenidos para un incremento de tiempo establecido con la función `delay()`, se decidió usar lo que hemos denominado “Criterio del percentil superior e inferior”. El percentil es una medida estadística la cual divide una serie de datos en 100 partes iguales. Este criterio se trata de un indicador que busca la proporción de la serie de datos que corresponden al 5% de las medidas de voltaje más altas y bajas registradas desde el generador de señales. Teóricamente se puede demostrar que cada una de estas señales cuando se tratan como puntos discretos, la proporción dada por el percentil superior e inferior se encuentra acotada dentro de ciertos valores característicos. Ahora procederemos a explicar cómo se obtuvieron estas proporciones para cada una de las ondas senoidal, triangular y cuadrada.

En la figura 4, se muestra una onda senoidal a partir de puntos discretos. En ésta, se observan las dos regiones de los criterios de los percentiles. Para nuestro caso, se ha dispuesto que los percentiles corresponden al 5% de los valores máximos y mínimos de amplitud. En el primer caso que analizaremos consideramos lo siguiente (1) una onda senoidal normalizada varía entre -1 y 1, (2) el 5% superior o inferior de los valores estaría entre 0.95 y 1.0, y (3) la onda sigue la función  $y = \sin(x)$ .

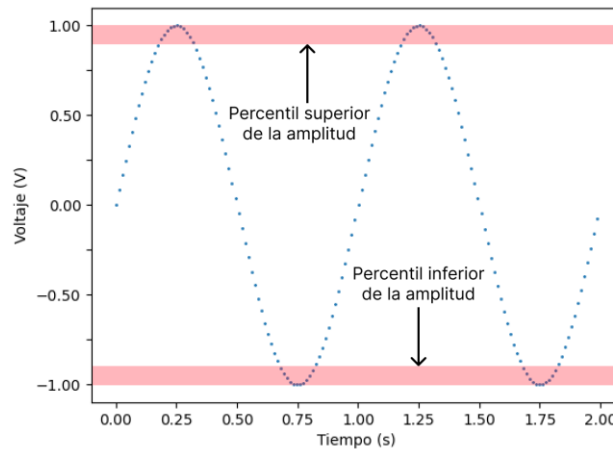


Figura 4. Criterio de percentiles para una onda senoidal discreta

Para encontrar qué porcentaje de los puntos discretos resaltados dentro de las franjas en la figura 4, necesitamos determinar qué fracción del ciclo de la onda está por encima de 0.95. Usando la función arcsen para encontrar el ángulo correspondiente, se tiene:

$$\arcsen(0.95) = 1.25 \text{ rad} = 71.6^\circ$$

Dado que la función senoidal es una función par, este ángulo ocurre cuatro veces en un ciclo, dos veces en la parte superior y dos veces en la parte inferior, como se muestra en la figura 5. Un ciclo completo corresponde a  $360^\circ$ , por lo que el porcentaje de puntos discretos en los percentiles superior e inferior sería

$$(90^\circ - 71.6^\circ) * 4 / 360^\circ * 100\% = \mathbf{20.5\%}$$

Así, para una onda senoidal normalizada respecto a la línea de equilibrio, aproximadamente el 10.25% de los puntos estaría dentro de la región extrema superior y el otro 10.25% de los puntos en la extrema inferior. Si la onda senoidal no está normalizada el resultado en términos de porcentaje de puntos discretos sería el mismo, como una consecuencia de la naturaleza periódica y simétrica de la función seno. La amplitud y el desplazamiento solo trasladan o escalan la onda, pero no cambian la proporción de tiempo que pasan en sus valores extremos.

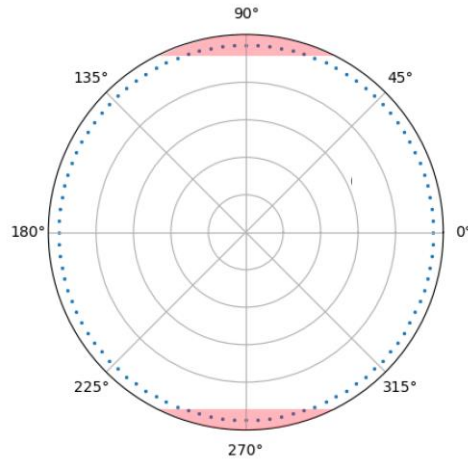


Figura 5. Fases de una onda senoidal normalizada.

Para abordar el caso de una onda triangular discreta como la mostrada en la figura 6, consideramos (1) una onda normalizada oscila entre -1 y 1, (2) que el 5% de los valores se encuentran entre 0.95 y 1, y (3) que la onda tiene una pendiente constante en sus segmentos lineales. Debido a esta relación lineal, podemos observar que el tiempo durante el cual la onda triangular supera el 95% de su amplitud es equivalente a que un 5% de los puntos discretos se encuentren dentro del 5% superior de la amplitud máxima de la onda. Una conclusión similar se puede aplicar a los puntos discretos que corresponden a la amplitud mínima de la onda. De este modo, aproximadamente el **10%** de los puntos discretos estarán dentro de los percentiles superior e inferior.

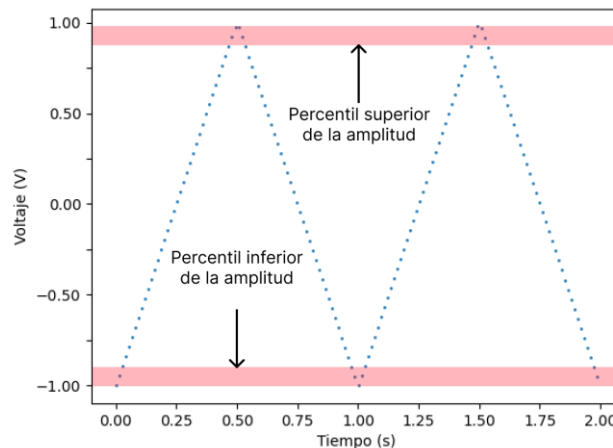


Figura 6. Criterio de percentiles para una onda triangular discreta

Para las ondas cuadradas como la mostrada en la figura 7, éstas alternan instantáneamente entre dos valores fijos que representan como sabemos el valor máximo y mínimo de la amplitud. En una onda cuadrada perfecta, todos los puntos están ya sea en el valor máximo o en el mínimo, no hay valores intermedios. Aplicando nuestro criterio del percentil superior e inferior, el 50% de los puntos están en el valor máximo y el 50% en el valor mínimo.

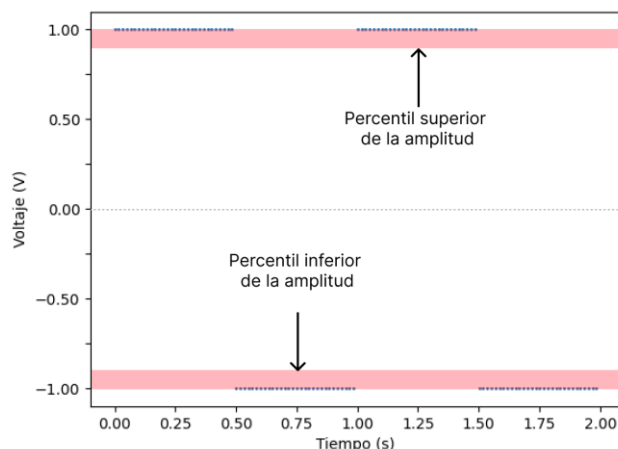


Figura 7. Criterio de percentil de una onda cuadrada discreta

Resumiendo nuestro análisis, podemos caracterizar las ondas senoidales, triangular y cuadrada a partir de la comparación del porcentaje de puntos discretos en las regiones de los percentiles superior o inferior, como se muestra en la tabla 1.

Tabla 1. Comparación de porcentajes de puntos discretos según la forma de onda

Onda	Puntos discretos (%)
Senoidal	10.25
Triangular	5.0
Cuadrada	50.0

### Cálculo de las amplitudes y frecuencias de las ondas

La amplitud de una onda representa el valor máximo de ésta respecto a la línea de equilibrio. Teniendo una señal simulada como la obtenida en Tinkercad, la amplitud de la onda se puede medir directamente desde los valores de voltaje máximos y mínimos. De este modo, la amplitud se obtiene a partir de la expresión

$$A = 0.5 * (V_{\text{max}} - V_{\text{min}})$$

donde  $V_{\text{max}}$  y  $V_{\text{min}}$  son respectivamente los voltajes máximos y mínimos de la señal.



Una onda senoidal perfecta se modela mediante la siguiente expresión general

$$y(t) = A * \text{sen}(2\pi * f * t + \Phi) + B$$

donde:

A es la amplitud

f es la frecuencia

t el tiempo

$\Phi$  es la fase

B es el desfase

Al despejar la frecuencia de la onda senoidal, se obtiene la siguiente expresión

$$f = \frac{\arcsen\left(\frac{y(t) - B}{A}\right) - \Phi}{2\pi t}$$

La frecuencia de una onda triangular o cuadrada es similar a la de una onda senoidal, aunque las formas características de estas últimas son diferentes a la de una onda senoidal.

Para estimar la frecuencia de una onda a partir de puntos discretos hemos dispuesto seguir los siguientes pasos suponiendo que tenemos una serie de puntos que representan la onda

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

En general la onda puede tener un desfase del voltaje, el cual, de acuerdo con nuestro análisis procederemos a eliminarlo, para ello, calculamos promedio de los voltajes de los puntos discretos para aproximar el desfase,

$$\text{desfase} = (y_1 + y_2 + \dots + y_n) / n$$

Luego, restamos el desfase a cada punto discreto para centrar la onda en la línea de equilibrio

$$y_i = y_i - \text{desfase}; \text{ para } i = 1, 2, \dots, n$$

Para estimar la frecuencia, primero calculamos la diferencia de fase entre cada par de puntos discretos donde ocurre cambio de signo del voltaje. Ahora calculamos la frecuencia usando la expresión

$$f = \frac{\text{cruces por cero}}{2 * \text{tiempo de captura}}$$

## Variables y tipo de datos

- **Variables primitivas**

int: Se utiliza para la mayoría de las variables que almacenan valores enteros, como pines digitales, estados de pulsadores y valores analógicos. Este tipo de dato ocupa 2 bytes en Arduino, lo que es suficiente para los valores que se manejan y ayuda a optimizar la memoria.

bool: Se usa para adquisiciónActiva, ocupando solo 1 byte.

unsigned int: Se emplea para variables que nunca serán negativas, como índices y tamaños. Esto permite un rango mayor de valores positivos en el mismo espacio de memoria (2 bytes).

float: Se utiliza para offset, factorConversion y frecuencia. Aunque ocupan más memoria (4 bytes), son necesarios para cálculos que requieren precisión decimal.

- **Arreglo dinámico**

const unsigned int tamanoArreglo = 300: Se define como una constante, lo que garantiza que el tamaño del arreglo no cambiará durante la ejecución.

- **Variables de gran capacidad**

long int sumaTotal: Se usa para acumular la suma de todos los valores. El tipo long int (4 bytes) previene desbordamientos en sumas grandes.

- **Constantes**

const float factorConversion: Al declararlo como constante, se asegura que no cambiará y potencialmente permite al compilador optimizar su uso.

- **Caracteres**

char formaOnda: Usa solo 1 byte para identificar el tipo de onda, siendo una opción eficiente en memoria.

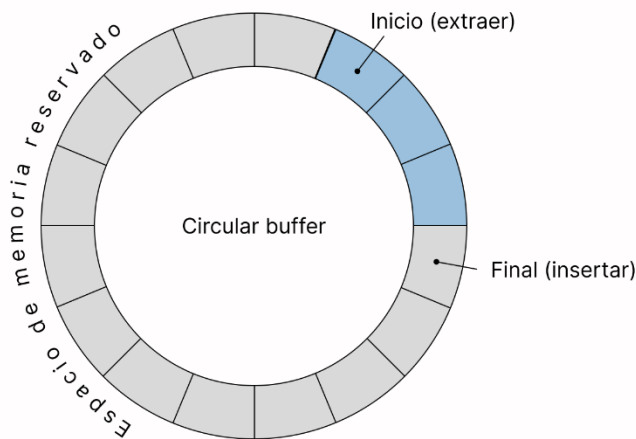
## Estructura de datos y uso de memoria dinámica

Para almacenar los datos, se implementó un arreglo dinámico unidimensional con espacio de memoria reservada para un total de 300 enteros, el cual, durante la ejecución del programa no modifica su tamaño.

### Índices y manejo circular del arreglo

Para optimizar la memoria y evitar el crecimiento descontrolado del arreglo, se usa el método de inserción conocido como buffer circular, el cual se explica a continuación.

Un buffer circular es una estructura de datos que consta de tres partes: un arreglo con tamaño definido y dos índices. El primer índice, el cual llamaremos 'inicio', hace referencia al primer elemento del arreglo y el segundo índice llamado 'fin' corresponde al último elemento, como se esquematiza en la figura 8.



*Figura 8. Partes de un buffer circular.*

El buffer circular gestiona los datos de manera eficiente, sobrescribiendo los elementos más antiguos cuando se alcanza la capacidad máxima del arreglo. Esta técnica resulta particularmente útil cuando se recibe un flujo constante de datos y solo se requiere almacenar una cantidad finita de éstos en la memoria. Esto resulta importante en dispositivos como Arduino, que tienen una RAM de apenas 2kB.

### Implementación del buffer circular

1. A medida que se agregan datos al arreglo y éste no está lleno, los datos se almacenan secuencialmente en las posiciones del arreglo.
2. Una vez que el buffer alcanza su capacidad, los nuevos datos reemplazan a los datos más antiguos. El índice "inicio" se va moviendo hacia adelante, y el "fin" se coloca en

la siguiente posición disponible, formando un ciclo continuo, como se muestra en la figura 8.

3. El buffer circular tiene la ventaja de que siempre podemos acceder al primer y al último elemento de manera eficiente, sin tener que desplazar los elementos de posición dentro del arreglo.

A continuación se muestra la función que implementa el buffer circular.

```
void agregarDatos(int dato){  
  
    sumaTotal += dato;  
  
    if(cantidadDatos == tamanoArreglo){  
        sumaTotal -= datosSenal[inicio];  
        datosSenal[inicio] = dato;  
        inicio = (inicio + 1) % tamanoArreglo;  
    }  
    else{  
        datosSenal[fin] = dato;  
        fin = (fin + 1) % tamanoArreglo;  
        cantidadDatos++;  
    }  
}
```

## **Liberación de memoria**

Cuando se reinicia la adquisición de datos, la memoria del arreglo que había sido reservada mediante el operador `new[]`, se libera usando la función `delete[]` y puede volver a utilizarse para la adquisición de datos de una nueva onda. Esto es necesario para evitar fugas de memoria y asegurarse de que el Arduino no se quede sin memoria luego de múltiples ejecuciones del programa.

## Implementación de las funciones

### 1. Cálculo del promedio de los puntos discretos de la onda para estimar el offset

```
float calcularPromedioTotal() {  
    if (cantidadDatos == 0) {  
        return 0; // Evitar división por cero  
    }  
    return (float)sumaTotal/cantidadDatos;  
}
```

Primero, se verifica si la cantidad de datos es cero para evitar una división indeterminada, lo cual retornaría un error. Si no hay datos (`cantidadDatos == 0`), la función devuelve 0. En caso contrario, calcula el promedio dividiendo la suma total de los datos (`sumaTotal`) entre la cantidad de éstos, y lo convierte a un valor de tipo `float` para asegurar precisión decimal.

### 2. Cálculo de los valores extremos de la onda

Esta función identifica el valor máximo y mínimo dentro de los datos de señal almacenados.

```
void calcularExtremos(int* max, int* min){  
    *max = datosSenal[0];  
    *min = datosSenal[0];  
    unsigned int i = 0;  
  
    while(i < cantidadDatos){  
        if (datosSenal[i] > *max){  
            *max = datosSenal[i];  
        }  
        if(datosSenal[i] < *min){  
            *min = datosSenal[i];  
        }  
  
        i++;  
    }  
}
```

Los punteros `max` y `min` son utilizados para almacenar los valores extremos de la señal, y se inicializan con el primer registro almacenado en el arreglo `datosSenal[]`. Luego, un bucle `while` compara cada registro en el arreglo `datosSenal[]`, con el valor máximo actual. El mismo tratamiento se aplica para el valor mínimo. De esta manera, al final del recorrido, los punteros corresponderán a los valores extremos de la señal.

### 3. Cálculo del percentil de la señal discreta

El siguiente código determina el porcentaje de datos que se encuentran en el 5% superior e inferior de la señal discreta, utilizando para ello los valores máximos y mínimos determinados en el apartado anterior.

```
float percentilSuperior(int *max, int *min) {
    unsigned int i = 0;
    unsigned int percentil = 0;
    float umbral = *max - (*max - *min) * 0.05; //

    for (i = 0; i < cantidadDatos; i++) {
        if (datosSenal[i] >= umbral) {
            percentil++;
        }
    }
    return (float)percentil / cantidadDatos * 100;
}
```

Primero, se calcula un umbral basado en los valores máximo y mínimo de la señal, correspondiente al 5% superior e inferior de la amplitud. Para esto, se resta el 5% de la diferencia entre el valor máximo y mínimo al valor máximo. A continuación, se recorre el arreglo `datosSenal[]` para contar cuántos de los datos superan o son iguales a este umbral. El número de datos que exceden el umbral se divide entre el número total de datos (`cantidadDatos`) y el resultado se multiplica por 100 para obtener el porcentaje correspondiente al percentil superior.

### 4. Definición de la forma de la onda términos del percentil

```
void defineForma(int percentilSup) {
    if(percentilSup > 12 && percentilSup < 25){
        formaOnda = 'S';
    }
    else if(percentilSup >= 8 && percentilSup <= 12){
        formaOnda = 'T';
    }
    else if(percentilSup >= 45 && percentilSup <= 55){
        formaOnda = 'C';
    }
    else{
        formaOnda = 'D';
    }
    lcd.setCursor(14, 0);
    lcd.print(formaOnda);
}
```

Se utiliza el valor del percentilSup (el porcentaje de puntos en el 5% superior) para identificar el tipo de onda:

- Si el percentil está entre 12% y 25%, la forma de la onda se define como senoidal ('S').
- Si está entre 8% y 12%, es una onda triangular ('T').
- Si está entre 45% y 55%, es una onda cuadrada ('C').
- Si no cumple ninguna de estas condiciones, se asume una forma de onda desconocida ('D').

## 5. Cálculo de la frecuencia

Esta función calcula la frecuencia de la señal, en términos del número de veces que ésta cruza la línea de equilibrio (offset), es decir, los puntos donde la señal cambia de signo.

```
float calcularFrecuencia() {
    unsigned int ceros = 0;

    for (int i = 1; i < cantidadDatos; i++) {
        if ((datosSenal[i-1] - offset) * (datosSenal[i] - offset) < 0) {
            ceros++;
        }
    }

    float tiempoCaptura = cantidadDatos * valorDelay;
    return (float) ceros * 1000 / (2 * tiempoCaptura);
}
```

Se declara una variable `ceros` que se utiliza para contar cuántas veces la señal cruza el valor de `offset` (que representa la línea de equilibrio o el centro de la onda).

A partir del segundo dato ( $i = 1$ ), se recorre el arreglo `datosSenal[]`.

Se verifica si el valor actual `datosSenal[i]` y el anterior `datosSenal[i-1]` tienen signos diferentes respecto al `offset` (línea de equilibrio). Esto se hace multiplicando los valores desplazados por el `offset` y revisando si el resultado es negativo. Si lo es, significa que ha ocurrido un cruce por cero, lo cual es un indicativo de un ciclo de la señal. Cada vez que ocurre un cruce por el valor de `offset`, el contador `ceros` se incrementa.

### **Cálculo del tiempo**

El tiempo de captura es el tiempo total durante el cual se ha registrado la señal. Se calcula multiplicando el número total de datos `cantidadDatos` por el valor del `delay` entre lecturas `valorDelay`, que es el tiempo que pasa entre la adquisición de cada dato.

### **Cálculo de la frecuencia**

La frecuencia de una onda es el número de ciclos completos por unidad de tiempo. Como cada ciclo completo de una señal tiene dos cruces por el valor de offset (uno hacia arriba y otro hacia abajo), el número total de cruces por cero se divide entre 2.

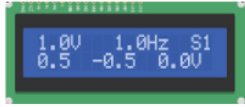
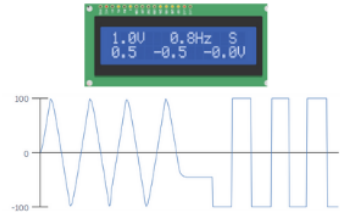
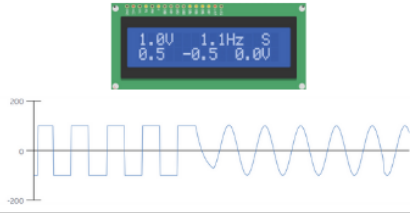
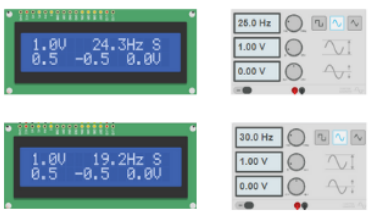
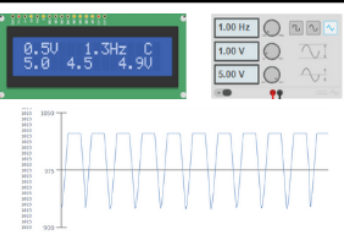
Luego, la fórmula para la frecuencia es:

$$f = \frac{\text{cruces por cero} * 1000}{2 * \text{tiempo de captura}}$$

Multiplicamos por 1000 para convertir el tiempo de captura a milisegundos, ya que `valorDelay` usualmente está en milisegundos.



## Tabla de margen de error

ERROR	HIPOTESIS	EJEMPLO
Codigo 1	Si el tiempo entre iniciar la adquisición y detenerla fue muy corto (toma menos de 200 datos), el arreglo no tiene suficientes datos para determinar la forma.	
Salida esperada desconocida 1	Si se cambia el tipo de onda mientras se adquieren los datos, a veces el programa lo toma como uno de los tipos de onda definidos (hay poca probabilidad de que salga cuadrada en estos casos). Esto se debe a que los percentiles (al ser de un promedio de toda la adquisición) pueden llegar al rango de valores que se usa para estimar el tipo de onda.	
Salida esperada desconocida 2	Si se cambian los tipos de onda al momento de adquirirlos debería ser una onda desconocida, pero como solo tomamos los últimos 300 datos, si estos son de un solo tipo de onda va a salir por la LCD que pertenece a ese tipo de onda.	
Frecuencia después de los 24Hz	El valor Delay lo tenemos como 20, lo que establece un retraso de 20 milisegundos entre cada lectura. Esto significa que la tasa de muestreo es de 50 Hz (1 / 0.02 s). Según el teorema de Nyquist, la frecuencia máxima que puedes medir de forma precisa debe ser la mitad de esta tasa de muestreo, es decir, 25 Hz. Por esto y porque el cálculo de la frecuencia lo hacemos con aproximaciones, después de los 24Hz muestra una frecuencia cada vez más diferente a la que muestra el generador	
Valor máximo 1023	Esto se debe a que la función analogRead() lee valores de 0 a 1023 entonces al intentar generar una onda en la cual haya puntos que deberían exceder este valor, el programa no es capaz de graficarlo. Por esto la onda se ve alterada y los datos en la pantalla LCD serían erróneos.	

## Montaje del Arduino en Tinkercad

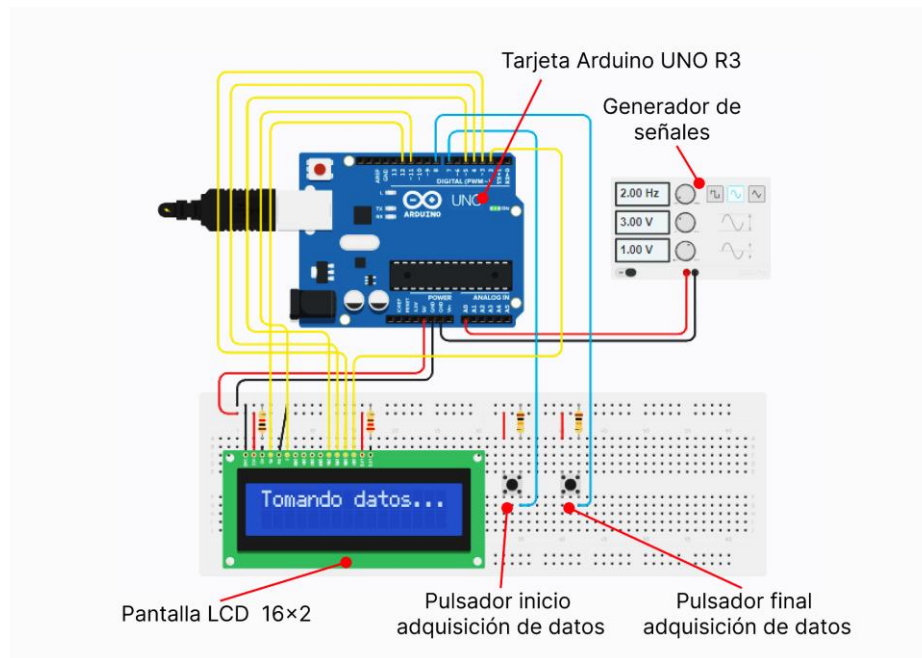


Figura 9. Componentes del montaje en Arduino

### Conexiones del Circuito

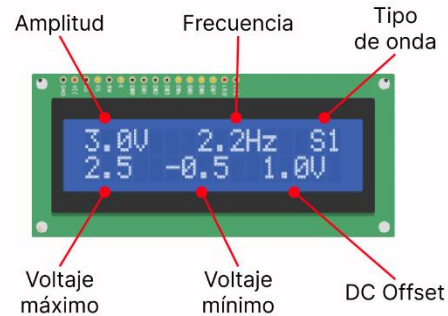
El circuito se construyó utilizando una placa Arduino UNO, un display LCD (16x2), dos pulsadores con sus resistencias y una breadboard como elemento de conexión entre los diferentes componentes. A continuación, se describen las conexiones y el propósito de cada componente en el sistema de generador de señales.

#### Placa Arduino UNO

La placa Arduino UNO se utiliza como el controlador principal del sistema. Es responsable de recibir los datos de la señal generada y procesarlos para calcular parámetros como el percentil superior, la forma de onda y la frecuencia. Posteriormente, los resultados se muestran en el display LCD.

- Alimentación: El Arduino está alimentado mediante una entrada de 9V, proporcionando los voltajes necesarios a los demás componentes a través de los pines 5V y GND.
- Control de señales: El Arduino también se encarga de analizar los valores de la señal discreta recibida desde el generador de señales, que está conectado a los pines analógicos para su procesamiento.

## Display LCD (16x2)



*Figura 10. Distribución de las lecturas de amplitud, frecuencia, tipo de onda, valores extremos y offset*

El display LCD se utiliza para mostrar los valores calculados, como la frecuencia de la señal, el tipo de onda y otros resultados obtenidos del análisis, como se muestra en la figura 10.

## Resistencias

- Resistencia de retroiluminación: Como se mencionó anteriormente, la resistencia de  $220\Omega$  protege el LED de retroiluminación del LCD.

## Generador de señales

El generador de señales conectado al circuito produce ondas discretas senoidales, triangulares y cuadradas. La señal generada es capturada por el Arduino a través de los pines analógicos, donde se realiza el análisis de sus características. Las conexiones relevantes son:

- Salida de señal (cables amarillos y azules): Están conectadas a los pines analógicos del Arduino, permitiendo la captura de la señal discreta para su posterior análisis.
- Alimentación (cables rojos y negros): Estas líneas proveen la energía necesaria para el funcionamiento del generador de señales, conectándose a los pines Vcc y GND.

## Diagrama de flujo del funcionamiento del program

