Informe Final Desafio 1 Informatica 2

2024-2

Diego Alejandro Alegria Cano

Felipe Rodas Vasquez

Facultad de Ingeniería, Universidad de Antioquia

Informática II

Dr. Aníbal José Guerra Soler

17 de Septiembre

Análisis del problema

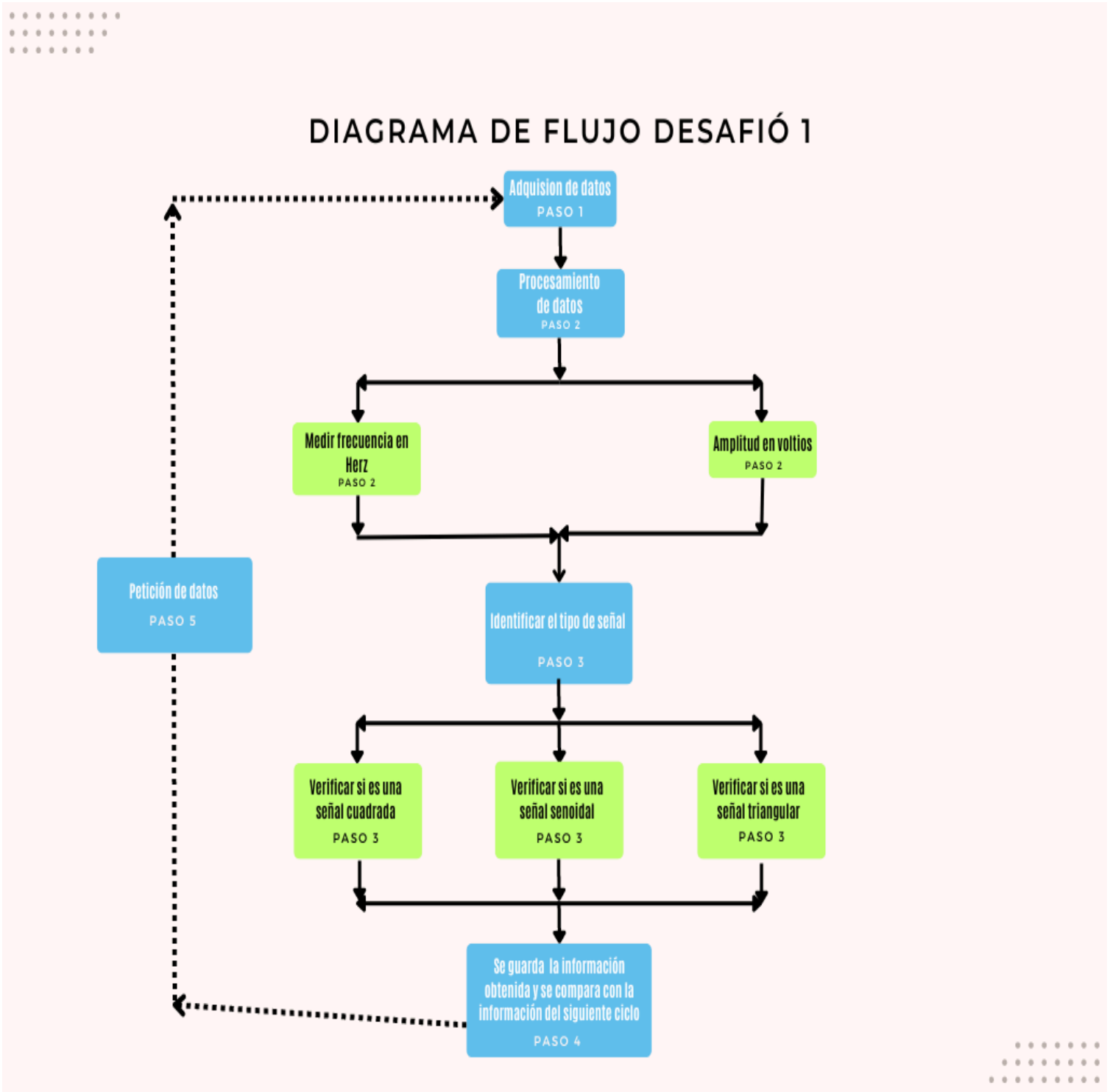
En un circuito simulado en Tinkercad, se diseña un sistema que a través de dos pulsadores permita leer los valores de una señal analógica generada por un generador de señales censar los valores obtenidos y mostrar los resultados generados cuando el usuario lo pida pulsado uno de los pulsadores. Se debe imprimir en una pantalla LCD la frecuencia en hertz y amplitud en voltios de la onda sensada, y mostrar el tipo de señal (cuadrada,triangular o senoidal). Si la señal no pertenece a ninguno de los tres tipos de señal se mostrará que la onda no es conocida.

En resumen, el proyecto implica la integración de hardware y software para crear un sistema de adquisición y análisis de señales analógicas en un entorno simulado con Tinkercad. Los puntos críticos son el desarrollo de un algoritmo robusto para medir la frecuencia y amplitud, así como la identificación del tipo de señal.

La solución planteada al Desafío es cuando se presione el pulsador para iniciar la captura de datos, almacenar los valores tomados de la señal y almacenarlos en un arreglo dinámico unidimensional. Mientras se vaya censando la señal se irán adquiriendo datos de la señal para ser comparados con un siguiente tramo de la señal.Una vez los datos del tramo se hayan obtenido, el espacio de memoria se liberara para ser ocupado por un siguiente segmento de la señal, esto se hace con el objetivo de no generar un aborto en la ejecucion por un excesivo uso de memoria. Al momento de comparar datos obtenidos se valora primero que la frecuencia y amplitud sean congruentes en un rango de 51%, si esto sucede, se procede a obtener el tipo de onda y compararla con la naturaleza del segmento inmediatamente anterior. Las coincidencias o diferencias se almacenan en un arreglo estático de dimensión cuatro en el que se van almacenando el número de coincidencias para cada tipo de señal y el número de diferencias, cada una de estas en un índice diferente del arreglo. Una vez se presione el pulsador para enseñar la información obtenida se obtendrá índice del mayor elemento del arreglo que será correspondiente al elemento tipo de onda con mayor coincidencia, se imprime el tipo de onda,

su frecuencia y amplitud. En caso de que el mayor elemento del arreglo sea el contador de diferencias se imprimirá que la Señal no es conocida y no se enseñara ningún dato más. La recolección de datos se reanudará de forma automática una vez finalizado este proceso.

Esquema.



Pasos del desarrollo	Información
Paso 1. Adquisición de datos.	Se recibe la información del generador de señales y ésta es almacenada en un arreglo dinámico unidimensional de capacidad fija.
Paso 2. Procesamiento de datos.	El procesamiento de datos se basa en calcular mediante teoría de ondas y fórmulas matemáticas la frecuencia de la onda y su amplitud en las medidas correspondientes para cada una Hertz y Voltios.
Paso 3. Identificar el tipo de señal.	Después del procesamiento de datos y de obtener la frecuencia y la amplitud, se identificara el tipo de señal utilizando funciones y arreglos para comparar la señal que tenemos con los tres tipos de señales establecidas o en caso tal que no ser ninguna de las tres decir que es una señal desconocida.
Paso 4: Guardar información obtenida y compararla.	Una vez la información de la frecuencia, amplitud y el tipo de señal sean procesados se guardarán para ser analizados posteriormente. El valor de la frecuencia y la amplitud serán guardados en variables primitivas y el tipo de señal será guardado en un arreglo unidimensional.
Paso 5. Petición de datos	La petición de los datos se hará mediante un pulsador, en el momento que se oprima el pulsador se dejará de guardar datos en el arreglo y los resultados de la comparación de la información obtenida anteriormente será mostrada.

Algoritmos implementados

A continuación se enseñan las funciones implementadas en el desarrollo del desafío:

```
// Función para detectar si la señal es cuadrada
bool senalCuadrada(int* arr, int size) {
  int numerosUnicos = 0;
  int unicos[4];

  for (int i = 0; i < size; i++) {
    int j = 0;
    for (j = 0; j < numerosUnicos; j++) {
      if (arr[i] == unicos[j]) {
        j = numerosUnicos;
      }
    }
    if (j == numerosUnicos) {
      unicos[numerosUnicos++] = arr[i];
      if (numerosUnicos > 4) {
        return false;
      }
    }
  }
  return true;
}
```

- La función **senalCuadrada** está diseñada para detectar si una señal representada como un arreglo de enteros puede considerarse "cuadrada" basándose en la información del arreglo. Los parámetros que recibe la función son: Un **puntero** a entero que representa un arreglo de enteros, es decir, la secuencia de valores que forman la señal.

  Un **entero** que indica el tamaño del arreglo, es decir, la cantidad de elementos en el arreglo. Una vez recibidos los parámetros la función Inicializa una variable llamada **numerosUnicos** a 0, que llevará la cuenta de la cantidad de números únicos encontrados en la señal, declara un arreglo **unicos[4]** que podrá almacenar hasta 4 números únicos, luego, la función recorre cada valor en el arreglo **arr** usando un bucle **for**. Si después de recorrer todo el arreglo **arr[]** se han encontrado 4 o menos números únicos, la función retorna **true**, lo que indica que la señal es **cuadrada**.

```
//Funcion que verifica si es una señal senoidal
bool senalSenoidal(int *arr,int size,float amplitud,float frecuencia){
  float cambio=0,promCambios=0,totalCambios=0;
  float valorMax = 0;

  valorMax = amplitud*100.0;

  for(int i=0;i<size;i++){
    if(arr[i]>=valorMax-2 and arr[i]<=valorMax+2 and i>=2){
        for(int k=i-2;k<i+2;k++){
            cambio += abs(arr[k+1]-arr[k]);
            totalCambios++;
        }
      i+=2;
    }
  }
  promCambios = cambio/totalCambios;
  if(promCambios<2*frecuencia*amplitud){
    return true;
  }
  return false;
}
```

- La función **senalSenoidal** está hecha para detectar si un patrón de valores guardados en un arreglo corresponde a una señal senoidal.Tiene como argumentos un puntero al arreglo que guarda los valores de la señal y su tamaño, además de la frecuencia y amplitud. La estrategia de esta función es buscar los máximos almacenados en el arreglo y obtener la diferencia entre los dos valores anteriores y los dos siguientes al valor máximo con el fin de únicamente analizar los picos de la señal, la resta se hace en orden entre cada par de elementos contiguos para al final obtener el promedio de los tamaños de paso y poder dar un veredicto. Dentro de la función se declaran 4 variables locales: **cambio** es un acumulador que tendrá el resultado de las diferencias evaluadas, **totalCambios** es un contador que tendrá la cantidad de restas realizadas, **promCambios** almacenará el valor promedio del resultado de las diferencias de los valores analizados y finalmente **valorMax** es la estimación del número máximo que se debe buscar durante el recorrido del arreglo. Mediante un ciclo for se recorre el arreglo, dentro del for va un if que tiene como condición que el valor actual esté dentro de un

rango de ± 2 al valor máximo, una vez encontrado se entra otro ciclo for en el que se itera entre los dos valores anteriores al maximo y los dos siguientes, se obtiene la resta entre cada pareja de elementos seguidos y se van sumando todos los resultados y contando cada resta realizada, terminado el proceso, se busca otro pico para repetir el proceso. Una vez recorrido todo el arreglo se obtiene el promedio de las diferencias hechas y se compara con el doble del producto entre amplitud y frecuencia, si el promedio de los cambios es menor a este producto, se considera la señal como senoidal. Esta función es viable en frecuencias entre 1 Hz y 3 Hz, con frecuencias más altas se confunde la señal dado a un cambio más brusco en los valores de los picos de la señal, haciendo la señal más similar a una triangular.

```
//Funcion que verifica si es una señal triangular
bool senalTriangular(int *arr,int size,float amplitud,float frecuencia){
  float cambio=0,promCambios=0,totalCambios=0;
  float valorMax = 0;

  valorMax = amplitud*100.0;

  for(int i=0;i<size;i++){
    if(arr[i]>=valorMax-2 and arr[i]<=valorMax+2 and i>=2){
        for(int k=i-2;k<i+2;k++){
            cambio += abs(arr[k+1]-arr[k]);
            totalCambios++;
        }
      i+=2;
    }
  }
  promCambios = cambio/totalCambios;
  if(promCambios>2*frecuencia*amplitud){
    return true;
  }
  return false;
}
```

- La función **senalTriangular** está hecha para detectar si un patrón de valores guardados en un arreglo corresponde a una señal senoidal.Tiene como argumentos un puntero al arreglo que guarda los valores de la señal y su tamaño, además de la frecuencia y amplitud. Su funcionamiento es prácticamente igual al de la función **senalSenoidal.** La

diferencia radica en la condición de su veredicto, el cual es si el promedio de los cambios en los picos de la señal es mayor al doble del producto por la frecuencia, se considera la señal como triangular. Esta función es viable para frecuencias entre 1 Hz y 7 Hz, debido a que en frecuencias más altas se presentan cambios muy radicales en la señal y es más difícil obtener la condición de señal Triangular.

```
// Función para medir la amplitud
float medirAmplitud(int* arr, int size) {
  int valorMaximo = 0;

  for (int i = 0; i < size; i++) {
    if (arr[i] > valorMaximo) {
      valorMaximo = arr[i];
      if(arr[i+1]<valorMaximo){
        break;
      }
    }

  }

  float amplitud = valorMaximo/100.0;  // Convertir a voltios
  return amplitud;
}
```

- La función **medirAmplitud** está diseñada para medir la amplitud de una señal representada por un arreglo de enteros.Se tienen como argumentos de la función un puntero al arreglo donde están almacenados los valores analógicos de la señal y el tamaño del arreglo. La amplitud corresponde al valor máximo que toma la señal. Con un ciclo for se recorre el arreglo almacenando en **ValorMaximo** todos los valores más grandes que vaya encontrando, se sabrá que encuentra el valor máximo o un pico cuando el dato inmediatamente siguiente a él es menor. Cuando esto pase se interrumpe el ciclo(Con el fin de no recorrer todo el arreglo si ya se encontró un valor confiable para medir la amplitud de la señal), el valor máximo se divide entre 100 para pasarlo a Voltios y se retorna la amplitud registrada.

```
int indiceMax(short unsigned int *arr, int size) {

    int valorMax = arr[0];
    int indiceMax = 0;
    int conteoMax = 0;

    for (int i = 1; i < size; i++) {
        if (*(arr+i) > valorMax) {
            valorMax = *(arr+i);
            indiceMax = i;
            conteoMax = 1;
        } else if (*(arr+i) == valorMax) {
            conteoMax++;
        }
    }

    if (conteoMax > 1) {
        return 3;
    }

    return indiceMax;
}
```

- La función **indiceMax** busca el **índice** del **valor máximo** en un arreglo de enteros sin signo.Tiene como argumentos a un arreglo unidimensional estático y su tamaño. Este arreglo corresponde a la cantidad de veces que se encontró una onda de cada tipo y en último lugar las desconocidas.El índice de mayor valor me dirá cuál fue el tipo de onda que más recurrencia tuvo durante el muestreo. Sin embargo, si el valor máximo no es único (es decir, si hay más de un elemento con el valor máximo en el arreglo), la función está modificada para retornar 3, que es el caso de onda desconocida.

```
void Reseteo(int*& arregloVal, int capacidad, int& cantidadVal,unsigned int& tiempo_actual) {
    // Limpiar el arreglo
    delete[] arregloVal;
    // Crear un nuevo arreglo vacío con la capacidad especificada
    arregloVal = new int[capacidad];
    // Reiniciar el índice para la nueva captura de datos
    cantidadVal = 0;
    // Reiniciar el tiempo actual
    tiempo_actual = millis();
}
```

- La función **Reseteo** tiene como propósito reiniciar y limpiar los valores de varias estructuras de datos clave en un sistema de adquisición de señales, como un arreglo que

almacena las lecturas de la señal, un arreglo de contadores que guarda el número de coincidencias en la detección de la señal, y una variable que mide el tiempo. Se utiliza cuando se completa una captura de datos y se necesita reiniciar el sistema para la siguiente adquisición.

```
void indetificarCoincidencias(float frecuenciaActual,float frecuenciaAnterior,float amplitudActual,float amplitudAnterior,short unsigned int *contadores,bool Cuadrada,bool Senoidal,bool Triangulo){

    // Comparar con el arreglo anterior
    if (amplitudAnterior != 0) {
      if(frecuenciaActual<=frecuenciaAnterior+frecuenciaAnterior*0.51 and frecuenciaActual>=frecuenciaAnterior-frecuenciaAnterior*0.51){
            if(amplitudActual<=amplitudAnterior+amplitudAnterior*0.51 and amplitudActual>=amplitudAnterior-amplitudAnterior*0.51){
            if (Cuadrada){
                              contadores[0]++;
                     }
                  else if (Senoidal){
                              contadores[1]++;
                  }
                  else if (Triangular){
                              contadores[2]++;
                  }
                  else{
                        contadores[3]++;
                  }
         }
            else{
                  contadores[3]++;
            }

      }
      else{
            contadores[3]++;
      }

    }
}
```

- La función **identificarCoincidencias** está diseñada para comparar las características de una señal (frecuencia y amplitud) con una señal previa, clasificando la señal como cuadrada, senoidal, triangular o no identificada según las coincidencias. La función recibe como parámetros cuatro variables de tipo **float,** un puntero a un arreglo de tipo **short unsigned int** y tres variables de tipo **bool,** las variables float tienen como nombre **frecuenciaActual, frecuenciaAnterior, amplitudActual, amplitudAnterior**; el puntero tiene como nombre **contadores** y las variables bool se llaman **Cuadrada, Senoidal y Triangular**, esta función lo que hace es verificar si la **amplitudAnterior** no es cero para proceder con las comparaciones. Compara la **frecuenciaActual** con la **frecuenciaAnterior** dentro de un margen del 51% (arriba o abajo). Si las frecuencias están dentro del rango, realiza una comparación similar para las amplitudes (51% de

margen).Si ambas comparaciones son exitosas, la función verifica si la señal es **Cuadrada**, **Senoidal** o **Triangular**:

1. Si es **Cuadrada**, incrementa el contador en **contadores[0].**
2. Si es **Senoidal**, incrementa el contador en **contadores[1].**
3. Si es **Triangular**, incrementa el contador en **contadores[2].**
4. Si no coincide con ninguna de estas formas, incrementa el contador en **contadores[3].**
5. Si las frecuencias y amplitudes no están dentro de los rangos, incrementa el contador para "no identificado" **(contadores[3]).**

Problemas en el desarrollo del desafío:

Nos enfrentamos a numerosos problemas en el transcurso de este desafío. Al principio empezamos de buena manera y avanzamos a buen ritmo. El problema más grande que encontramos fue la identificación del tipo de señal entre triangular y senoidal, atacamos este problema desde diferentes puntos de vista cómo calcular pendientes, encontrar la diferencia entre puntos seguidos de la señal, buscar patrones en los valores generados de cada tipo, etc. Todos estos sin un resultado significativo, lo que generó un estancamiento en nuestro desarrollo. Otro problema fue el manejo de la memoria del Arduino. Al estar acostumbrados a manejar memorias mucho más amplias no nos fijamos en el uso más eficiente de la memoria del Arduino en Tinkercad. Nuestro primer método para almacenar los datos de la señal fue un arreglo unidimensional dinámico que aumentaba su capacidad de almacenamiento según la cantidad de datos que se fuera obteniendo, al prolongar el tiempo de la muestra se generaba una interrupción de la simulación por falta de memoria. Este problema se agravó cuando decidimos guardar los datos en una matriz de dos filas, en la primera fila se almacenaban los valores de la señal y en la segunda fila los tiempos en que esos datos fueron tomados (con el fin de calcular

pendientes y frecuencias), esta estrategia comprometió aún más el correcto flujo del código pues era un arreglo que requería el doble de memoria que el anterior, generando una interrupción del código bastante prematura. Estos fueron los dos problemas más grandes a los que nos enfrentamos durante el desarrollo del Desafío pues a veces un proceso lógico puede fallar y el desconocimiento del uso de una capacidad de memoria no tan extensa puede jugar en contra.

Evolución de la Solución y consideraciones a tener en cuenta.

La evolución de la solución implementada empezó a buen ritmo. Primero empezamos por hacer el montaje inicial en Tinkercad, definir la función de cada pulsador , ver que se registraron en el serial los valores de las señales, y primeras impresiones en la LCD. El siguiente paso fue guardar los datos de la señal en un arreglo, en primera instancia se guardaban en un arreglo unidimensional que se redimensiona cuando se sobrepasa su capacidad inicial. El siguiente paso fue la implementación de una función que distinguiera una señal cuadrada con los datos del arreglo anterior. Se implementó una función que media la amplitud de la señal y un método para saber la frecuencia a través del cambio de signo de negativo a positivo y variables que almacenarán los tiempos en los que suceden estos cambios. Por consiguiente faltaban funciones que definieran si la señal registrada era triangular y senoidal. En este punto nos encontramos estancados un buen tiempo por lo que la evolución de la solución se tornó muy lenta y como se dijo anteriormente analizamos estas funciones de diferentes maneras e implementamos soluciones distintas ninguna con un resultado satisfactorio. Decidimos hacer un arreglo como matriz de dos fila y n columnas que guardaba cada valor de la señal y el tiempo en que este valor fue tomado con el propósito de calcular pendientes en los picos de las señales triangulares y senoidales para poder dar un veredicto de la naturaleza de la señal procesada. Esta alternativa hizo muy evidente el alto consumo de memoria que requería esta solucion causando abortos de

ejecucion de forma muy rapida. Para dar solución a este problema se decidió que sin importar si el usuario desea ver la información de la señal, cada cierta cantidad de datos la información sería procesada y guardada para compararla con un siguiente tramo de la señal. Una vez esta información era procesada se liberaba el arreglo y se reiniciaba para seguir registrando datos ahora sin interrupciones no deseadas. Después se decidió cambiar este criterio de un valor fijo a la ocurrencia de un evento, este evento es el periodo de la señal, de esta manera analizamos la señal periodo a periodo y se almacenan las coincidencias y diferencias en un arreglo de capacidad 4 cuyo primer índice serán las coincidencias en señales cuadradas, el segundo de señales senoidales, tercero triangulares y por ultimo señales desconocidas. Cuando el usuario oprima el pulsador para visualizar la información de la señal se verá que índice del arreglo tiene un mayor valor (mayor cantidad de períodos coincidentes en tipo, amplitud y frecuencia) para imprimir el tipo de onda que tenga más períodos que coincidan entre sí y la última amplitud y frecuencia registrada. Esta estrategia es más favorable para frecuencias bajas debido a que cuando hicimos la revisión por tamaño fijo a veces la señal no alcanzaba a realizar ni un periodo y se debía dejar bastante tiempo la captura de la señal para poder dar un buen veredicto. En frecuencias altas se realizaban muchos periodos en arreglos de 180 valores, esta implementación logró tener un mayor control sobre las alteraciones de la señal. Por último faltaban funciones más confiables para definir la forma senoidal y triangular que después de un largo proceso de análisis y depuración se lograron hacer funciones que funcionan bien en bajas frecuencias, basándose en el promedio de las restas de elementos cercanos al pico de la señal. En bajas frecuencias se tiene que este promedio para funciones senoidales es menor al doble del producto de la frecuencia por la amplitud y para señales triangulares el promedio es mayor a este mismo umbral. Con esta lógica se hicieron funciones que analicen los picos de las señales para dar con buen nivel de confiabilidad la naturaleza de la señal analizada. El resto del

proceso se basa en refinación del código como organización, variables más dicientes, etc. Este fue un resumen del recorrido de la solución del desafío 1.

Consideraciones: Nuestro código es más confiable en bajas frecuencias puesto que el patrón de cambio entre los valores de la señal es más repetitivo que en altas frecuencias. Como se dijo antes, la función para determinar señales senoidales tiene una buena confiabilidad en frecuencia entre 1 y 3 Hz por encima de estas frecuencias la señal se comienza a parecer bastante a una triangular y el promedio de las diferencias en sus picos son muy similares. La función de señales Triangulares es buena en frecuencias de 1 Hz hasta 7 Hz, por encima de estos rangos los cambios en los picos empiezan ser demasiado abruptos y difíciles de analizar.

Otra consideración a tener en cuenta es que a la hora de mostrar resultados se imprimirá el tipo de onda que tenga más periodos coincidentes.

Link de la simulacion en Tinkercad:

https://www.tinkercad.com/things/hLUEz3BsQe3-desafio-1-info2?sharecode=zFcRhMlUE7vxQAjAXnmZBzXwpVCLA2Hio8FLJ7Vzh5Y