Informe del desafío 1

# Hugo Esteban Barrero García – Jose Gabriel Giraldo

# Universidad de Antioquia

# Informática 2

# Aníbal José Guerra Soler

# Medellín, Colombia

Septiembre, 2024

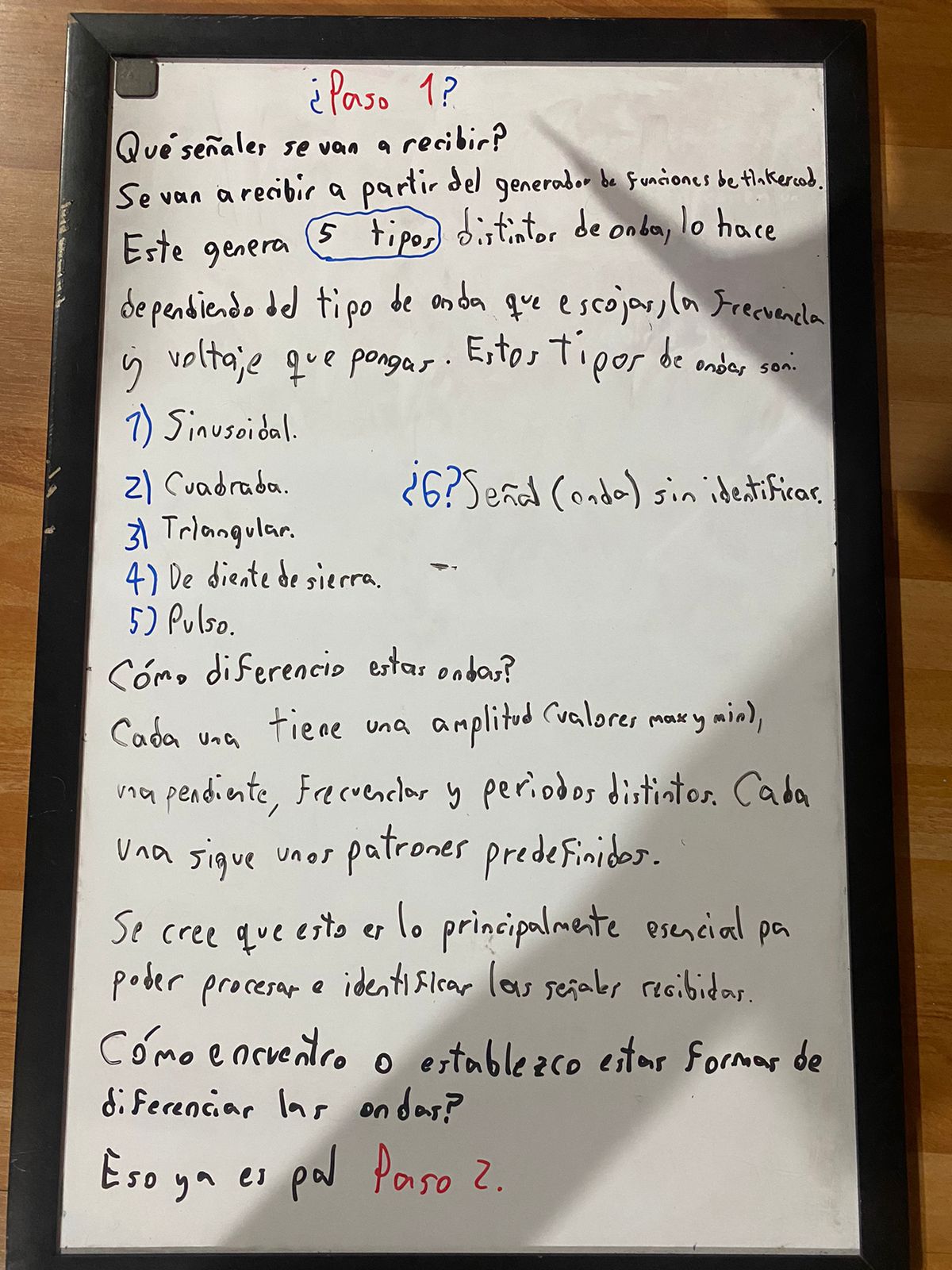
Índice

1. Introducción y análisis del problema
   * Breve descripción del proyecto o problema abordado.
   * Descripción detallada del problema a resolver.
   * Consideraciones iniciales y restricciones.
   * Entendimiento del programa a desarrollar.
   * Esquemas y diagramas que ilustren el diseño inicial.
2. Desarrollo del Programa
   * 2.1) Versioneo del Programa en Arduino.
   * 2.2) Funciones y Código.
3. Resultados y Análisis
   * Análisis de los resultados y comparación con los objetivos iniciales.
   * Conclusiones
   * Resumen de los logros alcanzados.
   * Limitaciones encontradas.

Introducción y Análisis

Análisis realizado por Jose y por Hugo en el intento por ver los caminos a seguir para entender y lograr resolver este desafío:A paper with writing on it

Description automatically generated.



Continuación Paso 1:

Para este generador de funciones del tinkercad solo se tienen en cuenta las señales sinusoidales, cuadradas y triangulares.

Para diferenciarlas se pueden hacer estos dos pasos.

Mirar si tiene cambios muy abruptos en su amplitud, no tienen valores medios, algo como una señal digital, que cuenta con solo valores altos y bajos, si cumple estas características es una onda cuadrada, y si no la cumple entonces:

La onda es senoidal o triangular

Paso 2.

2.1) Almacenar los datos recibidos de amplitud (en voltaje).

2.2) Calcular la frecuencia.

(Puede servir calcular el número de veces que los valores de la amplitud coinciden con el eje x o calcular el número de ciclos.).

(También debo calcular el tiempo entre pulsaciones).

2.3) Calcular la amplitud (comparar los valores máximos y mínimos).

2.4) Establecer los patrones predefinidos de cada tipo de señal e identificar la forma de la onda por medio de compararla con los patrones predefinidos de cada onda.

2.5) Imprimir lo procesado en pantalla. (LCD).

Paso 3.

Ya con el código funcional y listo, probarlo y verificarlo en tinkercad, probar en qué valores llegue a fallar y tener todo bien documentado.

Hacer el circuito funcional con dos pulsadores, A y B, cuando se presiona el pulsador A, se empiece la recolección de datos del generador de funciones y cuando se presione el pulsador B se dejen de recibir datos, pero que el programa siga funcionando y cuando se presione nuevamente el pulsador A se finalice la ejecución del programa.

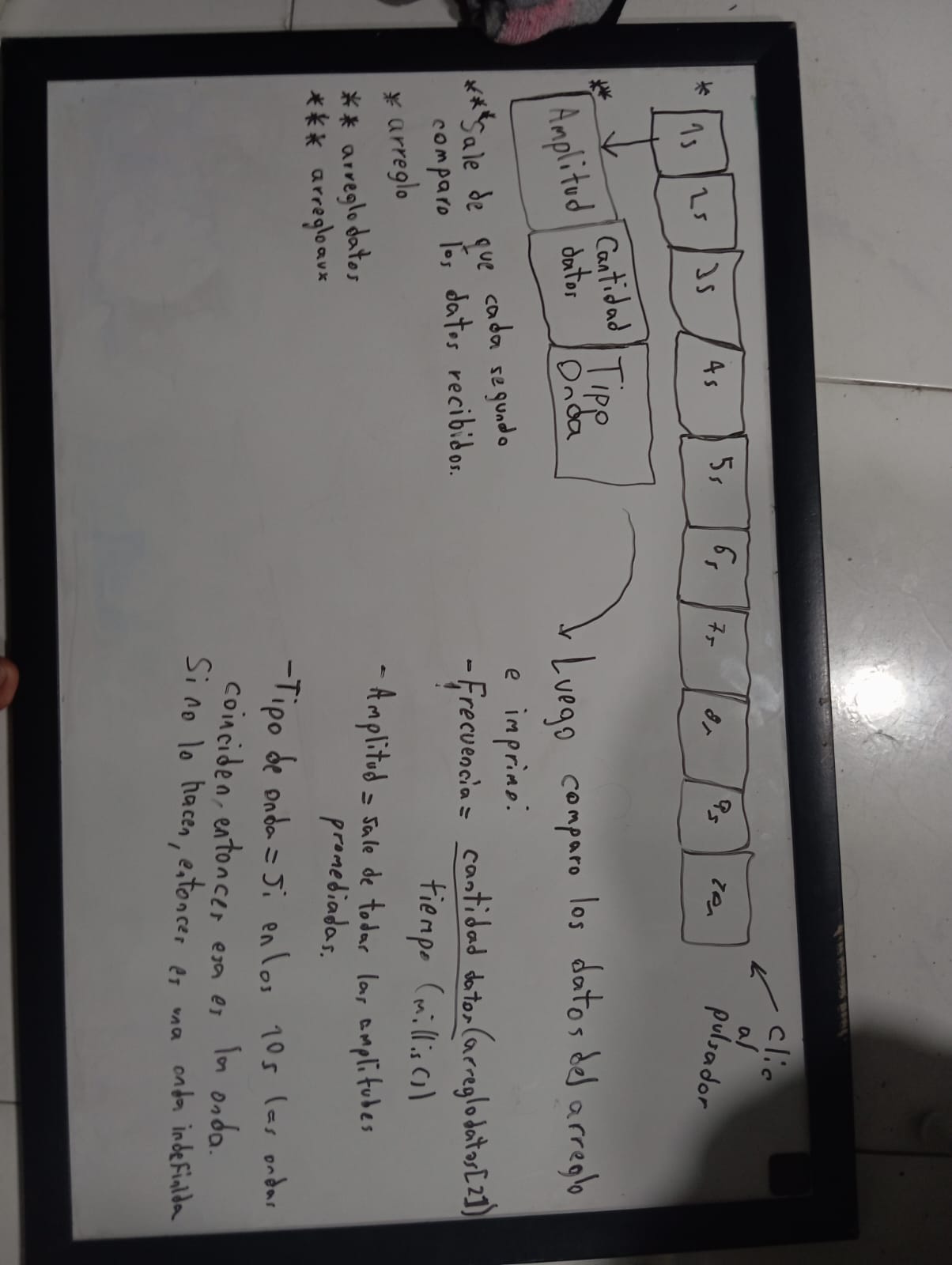
Paso 4.

A lo largo del desarrollo del programa, tener bien documentado cada paso, desarrollar un documento en Word con Normas Apa con todo bien escrito y detallado, en cada documento de código y funciones se tendrá sus pequeñas especificaciones y docstrings. Y para la realización del vídeo una explicación general de los pasos y funcionamiento de todo, no tanto decir lo del documento sino más bien explicar lo que entendemos.

Análisis para la toma y almacenamiento de datos:

Inicialmente hicimos una prueba con un arreglo que almacenara absolutamente todos valores analógicos que íbamos recibiendo, pero claramente no tenemos memoria infinita y en poco tiempo llegaba a colapsar el programa, por lo que decidimos hacerlo por tramos. Entonces lo que hacemos es un arreglo dinámico con cierta capacidad que reciba los datos del pin analógico asociado al generador de funciones, y por cada segundo que pasa, los datos recibidos se van a procesar (hallar amplitudes, tipos de onda, ciclos, etc.) y estos datos importantes encontrados se van a almacenar en distintas variables y posteriormente se va a reiniciar este arreglo para continuar nuevamente su toma de datos hasta que pase el siguiente ciclo (ósea que pase un segundo) o que se presione el pulsador B para dejar de recibir datos. (También está implementado que en caso tal que el arreglo se llene antes que pase el segundo, en ese momento también se procesen los datos y se reinicie el arreglo, pero es algo que nunca llegó a pasar).

Inicialmente tuvimos esta idea para la toma de datos. Pero tenia muchos arreglos y nos dimos cuenta de unos errores como por ejemplo en la frecuencia de la onda que no se halla con la cantidad de datos sino con la cantidad de ciclos que hubo en el tiempo de ejecución entre pulsadores.



Por lo que decidimos implementar la toma de datos así:

A white board with writing on it

Description automatically generated

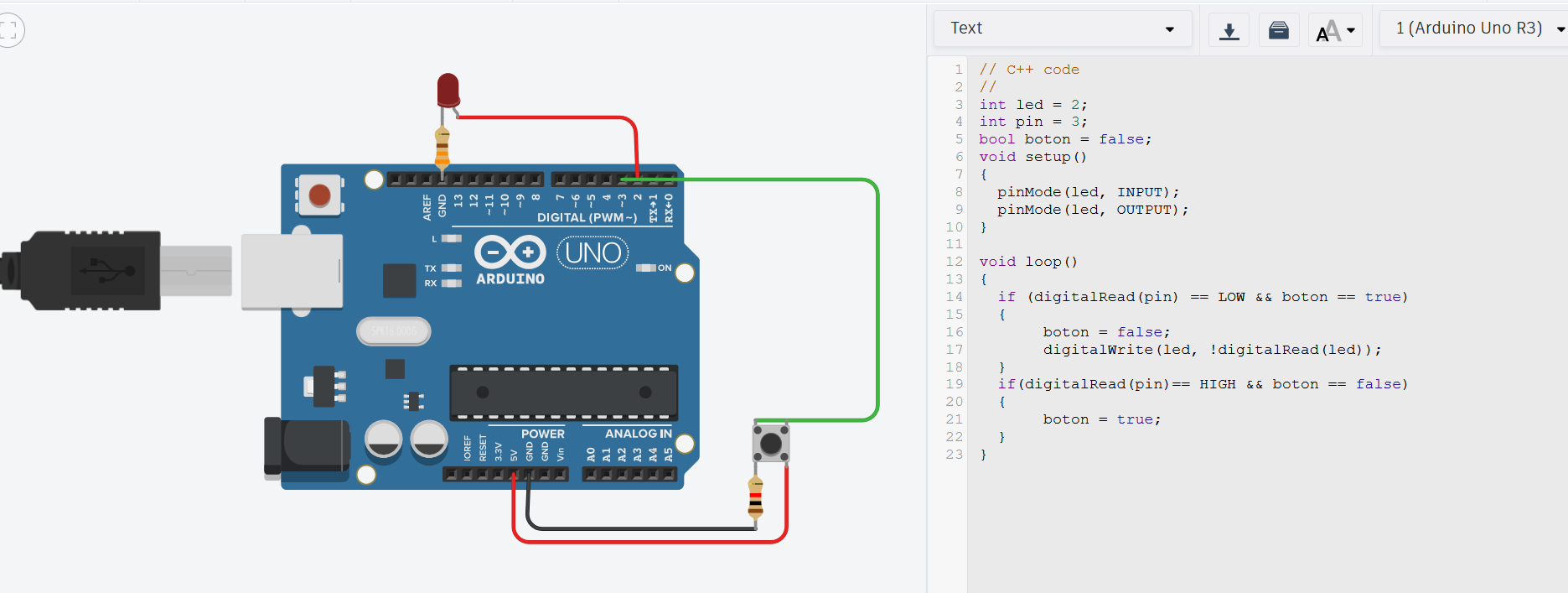
Desarrollo del Programa

Versiones del Programa en arduino: pensamos en esto como si se estuviera creando diferentes versiones de un programa, cada una enfocada en resolver un problema específico. Inicialmente, para entender bien cómo funcionaban los pulsadores y el LCD en Arduino. Por ejemplo, lograr que el pulsador A iniciara la recolección de datos y que el pulsador B la detuviera.

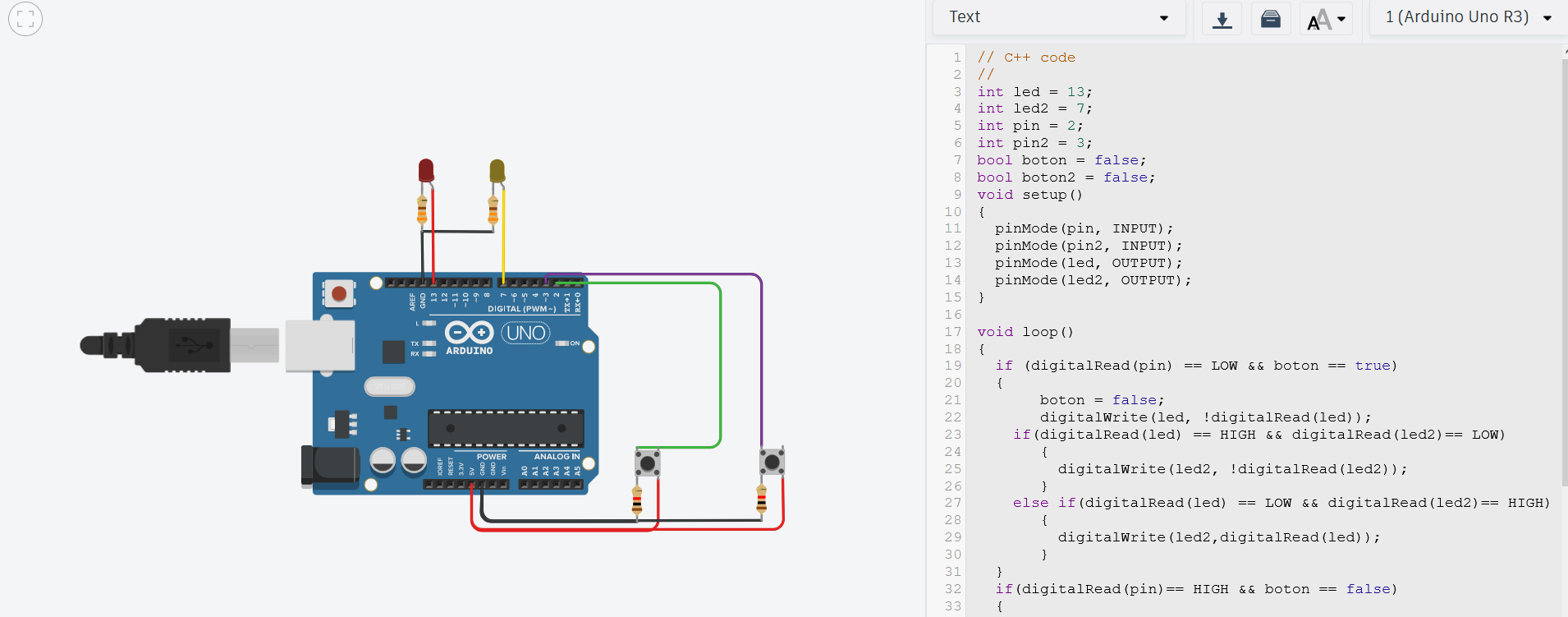
Para esto, se empezó a experimentar con el código. Realizamos que era importante usar una resistencia pull-down con los pulsadores. Esta resistencia hace que el pin del Arduino al que está conectado el pulsador esté en un estado bajo (cero) cuando no se presiona el botón. Al presionar el botón, el estado cambia a alto (uno).

También se usaron leds para visualizar lo que estaba sucediendo. Cada vez que se presione un pulsador, cambiaba el estado de un LED para confirmar que la señal estaba siendo detectada. Luego aplicar esta lógica con dos pulsadores y posteriormente ya al circuito final con el generador de funciones y todo.

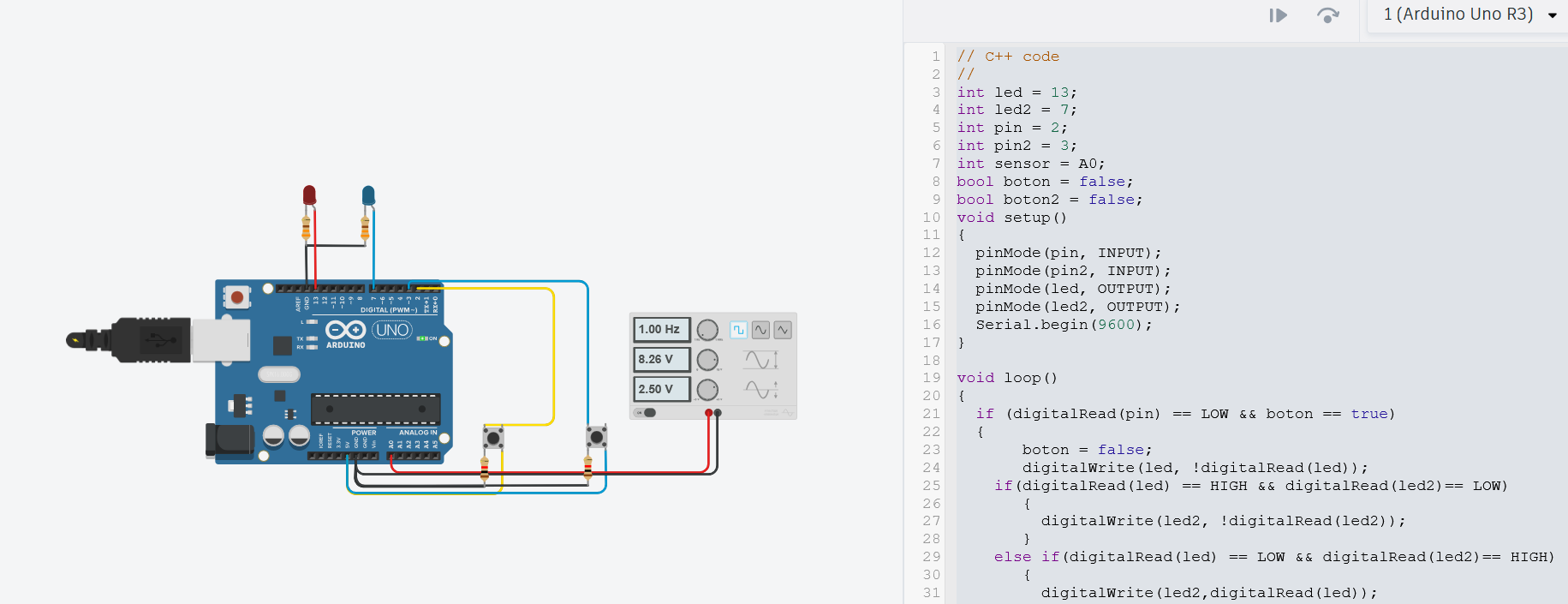
Primer programa pulsadores. (Versión 1)



Segundo programa pulsadores. (Version 2)



Tercer programa pulsadores ya con el generador de funciones. (Versión 3)

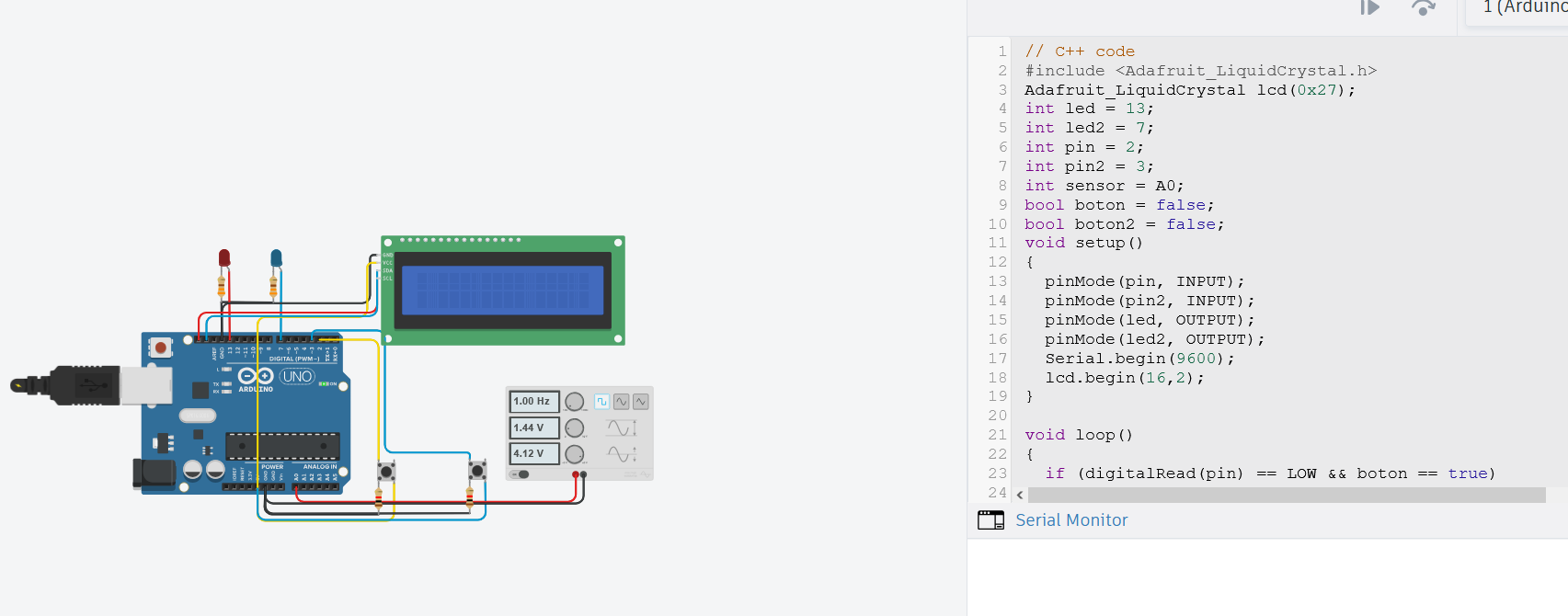


Versión 1 (LCD)

A computer screen shot of a computer

Description automatically generated

Cuarto programa pulsadores ya con el generador de funciones y el LDC. (Versión 4)



Quinto programa, toda la versión 4 pero hecha modularmente y con la obtención del tiempo en los intervalos necesarios (Entre pulsaciones). (Versión 5)

A computer screen shot of a computer

Description automatically generated

A partir de esta versión no hay casi cambios en el circuito principal, ya la lógica del funcionamiento del circuito esta podemos decirlo que completa para la finalidad del programa y a partir de aquí es que ocurren mayores cambios con el código, como por ejemplo la inclusión de lógica y funciones para tomar datos y procesarlos cada segundo o para calcular amplitudes, diferenciar el tipo de onda y calcular ciclos y frecuencias. Todos estos cambios se les hicieron sus respectivos commits en el repositorio y se explica cada cambio hecho.

Codigo final:

La funcion para calcular ciclos la dejamos como por dejar el mejor intento que hicimos pero la funcion no es correcta y da resultados similares a una frecuencia pero de bajos Hz.

Y tampoco logramos finalizar la distinción entre las ondas triangulares de las senoidales.

float calcularciclos(float\* arreglo, int capacidad, float amplitudmax, float amplitudmin) {

int num\_ciclos = 0;

bool subiendo = true;

float umbral\_superior = 0.9 \* amplitudmax;

float umbral\_inferior = 0.1 \* amplitudmin;

for (int i = 1; i < capacidad - 1; i++) {

if (subiendo && arreglo[i] > umbral\_superior) {

num\_ciclos++;

subiendo = false;

} else if (!subiendo && arreglo[i] < umbral\_inferior) {

num\_ciclos++;

subiendo = true;

}

}

return num\_ciclos/2;

}

Funciones importantes:

//Función para la amplitud.

void amplitud(float\*arreglo,int capacidad)

{

for (int i = 0; i < capacidad; i++)

{

if (arreglo[i] != 0 && arreglo[i]!= -1)

{

if (arreglo[i] >= amplitudmaxactual)

{

amplitudmaxactual = arreglo[i];

}

else if (arreglo[i] < amplitudminactual)

{

amplitudminactual = arreglo[i];

}

if (arreglo[i] >= amplitudmax)

{

amplitudmax = arreglo[i];

}

else if (arreglo[i] < amplitudmin)

{

amplitudmin = arreglo[i];

}

}

}

}

//Intento funcion para calcular ciclos.

float calcularciclos(float\* arreglo, int capacidad, float amplitudmax, float amplitudmin) {

int num\_ciclos = 0;

bool subiendo = true;

float umbral\_superior = 0.9 \* amplitudmax;

float umbral\_inferior = 0.1 \* amplitudmin;

for (int i = 1; i < capacidad - 1; i++) {

if (subiendo && arreglo[i] > umbral\_superior) {

num\_ciclos++;

subiendo = false;

} else if (!subiendo && arreglo[i] < umbral\_inferior) {

num\_ciclos++;

subiendo = true;

}

}

return num\_ciclos/2;

}

//Intento funcion para calcularonda.   
void identificaronda(float\* arreglo, int capacidad, float amplitudmaxactual, float amplitudminactual) {

bool cuadrada = false;

bool triangular = true;

int indicemax = 0;

int indicemin = 0;

for(int i = 0; i < 3; i++) {

if (arreglo[i] == amplitudmaxactual || arreglo[i] == amplitudminactual || arreglo[i] == 10000 || arreglo[i] == -10000 || arreglo[i] == 0) {

cuadrada = true;

} else {

cuadrada = false;

break;

}

}

if (cuadrada == true)

{

cantcuadrada++;

Serial.println("La onda es cuadrada");

}

else

{

for (int i = 0; i < capacidad; i++)

{

if (arreglo[i] == amplitudmaxactual) {

indicemax = i;

} else if (arreglo[i] == amplitudminactual) {

indicemin = i;

}

}

if (triangular == true) {

Serial.println("La onda es triangular");

canttriangular++;

} else

{

Serial.println("La onda es sinusoidal");

cantsenoidal++;

}

}

}

Los siguientes códigos son puras versiones o intentos de la funcion para identificar el tipo de onda.

1)void identificaronda(float\* arreglo, int capacidad, float amplitudmax, float amplitudmin)

{

bool cuadrada = false;

bool triangular = false;

int indicemax = 0;

int indicemin = 0;

// Verificamos primero si es cuadrada (tal y como lo tienes)

for(int i = 0; i < 3; i++) {

if (arreglo[i] == amplitudmax || arreglo[i] == amplitudmin) {

cuadrada = true;

} else {

cuadrada = false;

break;

}

}

if (cuadrada == true) {

Serial.println("La onda es cuadrada");

}

// Si no es cuadrada, entonces verificamos si es triangular

else {

// Encontramos los índices del máximo y mínimo

for (int i = 0; i < capacidad; i++) {

if (arreglo[i] == amplitudmax) {

indicemax = i;

}

if (arreglo[i] == amplitudmin) {

indicemin = i;

}

}

// Ahora analizamos las pendientes para identificar si es triangular

// Comparamos las pendientes a la izquierda y derecha del máximo

float pendiente\_izquierda = (arreglo[indicemax] - arreglo[0]) / (indicemax - 0);

float pendiente\_derecha = (arreglo[indicemax] - arreglo[capacidad-1]) / (capacidad - 1 - indicemax);

if (abs(pendiente\_izquierda - pendiente\_derecha) < 0.1) { // Si las pendientes son aproximadamente iguales

triangular = true;

} else {

triangular = false;

}

if (triangular == true) {

Serial.println("La onda es triangular");

} else {

Serial.println("La onda es senoidal"); // Si no es cuadrada ni triangular, entonces es senoidal

}

}

}

2) char\* identificarOnda(float\* arreglo, int capacidad, float amplitudmax, float amplitudmin) {

// Inicializar variables para la detección

int cambiosPendiente = 0;

int crucesEje = 0;

float pendienteAnterior = 0;

bool pendientePositiva = true;

float pendiente;

// Analizar los datos del arreglo

for (int i = 1; i < capacidad; i++) {

// Calcular la pendiente actual

pendiente = arreglo[i] - arreglo[i - 1];

// Detectar cruces por el eje (de negativo a positivo o viceversa)

if ((arreglo[i - 1] <= 0 && arreglo[i] > 0) || (arreglo[i - 1] >= 0 && arreglo[i] < 0)) {

crucesEje++;

}

// Detectar cambios de pendiente

if ((pendiente > 0 && !pendientePositiva) || (pendiente < 0 && pendientePositiva)) {

cambiosPendiente++;

pendientePositiva = (pendiente > 0);

}

pendienteAnterior = pendiente;

}

// Clasificar la onda según los patrones detectados

char\* tipos[] = {"Desconocida", "Triangular", "Cuadrada", "Senoidal"};

int indiceTipo = 0; // Índice del tipo de onda

// Clasificación basada en cruces de eje y cambios de pendiente

if (cambiosPendiente == 2 && crucesEje == 2) {

indiceTipo = 3; // Onda sinusoidal

} else if (cambiosPendiente == 2 && crucesEje == 0) {

indiceTipo = 2; // Onda cuadrada

} else if (cambiosPendiente == 4) {

indiceTipo = 1; // Onda triangular

}

// Imprimir y retornar el tipo de onda detectado

Serial.print("Tipo de onda: ");

Serial.println(tipos[indiceTipo]);

return tipos[indiceTipo];

}

3) int identificarOnda() {

bool esCuadrada = false;

bool esSenoidal = true;

bool esTriangular = true;

for (int i = 1; i < numMuestras - 1; i++) {

float diferencia = valores[i+1] - valores[i];

// Detectar onda cuadrada: cambios abruptos entre alto y bajo

if (valorAbsoluto(diferencia) > TOLERANCIA\_SALTO) {

esCuadrada = true;

esSenoidal = false;

esTriangular = false;

}

// Detectar onda triangular: cambios lineales constantes

if (i > 1) {

float diferenciaAnterior = valores[i] - valores[i-1];

if (valorAbsoluto(diferencia - diferenciaAnterior) > TOLERANCIA\_LINEAL) {

esTriangular = false;

}

}

// Detectar onda senoidal: cambios suaves (no abruptos ni lineales)

if (esSenoidal && esTriangular) {

// Confirmar que los cambios son suaves y no abruptos ni lineales

esSenoidal = true;

} else {

esSenoidal = false;

}

}

// Priorizar los tipos de onda basados en los patrones detectados

if (esCuadrada) return 0; // Onda cuadrada

if (esSenoidal) return 1; // Onda senoidal

if (esTriangular) return 2; // Onda triangular

return -1; // No se pudo identificar

}

4) void identificarOnda(float\* arreglo, int capacidad, float amplitudmaxactual, float amplitudminactual) {

bool cuadrada = false;

bool triangular = false;

int indicemax = 0;

int indicemin = 0;

// Detección de onda cuadrada (sin cambios)

for(int i = 0; i < 3; i++) {

if (arreglo[i] == amplitudmaxactual ||

arreglo[i] == amplitudminactual ||

arreglo[i] == 10000 || arreglo[i] == -10000 ||

arreglo[i] == 0) {

cuadrada = true;

} else {

cuadrada = false;

break;

}

}

if (cuadrada == true) {

Serial.println("La onda es cuadrada");

} else {

// Encontrar el índice del máximo y mínimo

for (int i = 0; i < capacidad; i++) {

if (arreglo[i] == amplitudmaxactual) {

indicemax = i;

} else if (arreglo[i] == amplitudminactual) {

indicemin = i;

}

}

// Calcular pendientes antes y después de los extremos

int ventana = 10; // Ajustar según sea necesario

float pendiente\_antes\_max = (arreglo[indicemax] - arreglo[indicemax - ventana]) / ventana;

float pendiente\_despues\_min = (arreglo[indicemin + ventana] - arreglo[indicemin]) / ventana;

// Calcular la curvatura aproximada

int ventana\_curvatura = 5; // Ajustar según sea necesario

float curvatura\_total = 0;

for (int i = ventana\_curvatura; i < capacidad - ventana\_curvatura; i++) {

float pendiente\_izq = (arreglo[i] - arreglo[i - ventana\_curvatura]) / ventana\_curvatura;

float pendiente\_der = (arreglo[i + ventana\_curvatura] - arreglo[i]) / ventana\_curvatura;

curvatura\_total += abs(pendiente\_der - pendiente\_izq);

}

// Umbrales

float umbral\_pendiente = 0.1; // Ajustar según sea necesario

float umbral\_curvatura = 0.2; // Ajustar según sea necesario

// Clasificación

if (abs(pendiente\_antes\_max - pendiente\_despues\_min) < umbral\_pendiente) {

triangular = true;

} else if (curvatura\_total > umbral\_curvatura) {

Serial.println("La onda es sinusoidal");

} else {

Serial.println("La onda no es cuadrada, triangular ni sinusoidal");

}

if (triangular) {

Serial.println("La onda es triangular");

}

}

}

5)void identificaronda(float\* arreglo, int capacidad, float amplitudmax, float amplitudmin) {

bool cuadrada = false;

bool triangular = false;

// Detección de onda cuadrada (sin cambios)

for(int i = 0; i < 3; i++) {

if (arreglo[i] == amplitudmaxactual || arreglo[i] == amplitudminactual || arreglo[i] == 10000 || arreglo[i] == -10000 || arreglo[i] == 0) {

cuadrada = true;

} else {

cuadrada = false;

break;

}

}

if (cuadrada == true) {

Serial.println("La onda es cuadrada");

} else {

// Calcular la curvatura aproximada

int ventana\_curvatura = 5; // Ajustar según sea necesario

float curvatura\_total = 0;

for (int i = ventana\_curvatura; i < capacidad - ventana\_curvatura; i++) {

float pendiente\_izq = (arreglo[i] - arreglo[i - ventana\_curvatura]) / ventana\_curvatura;

float pendiente\_der = (arreglo[i + ventana\_curvatura] - arreglo[i]) / ventana\_curvatura;

curvatura\_total += abs(pendiente\_der - pendiente\_izq);

}

// Umbral de curvatura (ajustar según sea necesario)

float umbral\_curvatura = 0.2;

// Clasificación basada en la curvatura

if (curvatura\_total > umbral\_curvatura) {

Serial.println("La onda es sinusoidal");

cantsenoidal++;

} else {

Serial.println("La onda es triangular");

canttriangular++;

}

}

}

6) void identificaronda(float\* arreglo, int capacidad, float amplitudmaxactual, float amplitudminactual) {

bool cuadrada = false;

bool triangular = false;

int indicemax = 0;

int indicemin = 0;

// Detección de onda cuadrada (sin cambios)

for(int i = 0; i < 3; i++) {

if (arreglo[i] == amplitudmaxactual || arreglo[i] == amplitudminactual || arreglo[i] == 10000 || arreglo[i] == -10000 || arreglo[i] == 0) {

cuadrada = true;

} else {

cuadrada = false;

break;

}

}

if (cuadrada == true) {

Serial.println("La onda es cuadrada");

} else {

// Encontrar el índice del máximo y mínimo

for (int i = 0; i < capacidad; i++) {

if (arreglo[i] == amplitudmaxactual) {

indicemax = i;

} else if (arreglo[i] == amplitudminactual) {

indicemin = i;

}

}

// Calcular pendientes antes y después de los extremos

int ventana = 10; // Ajustar según sea necesario

float pendiente\_antes\_max = (arreglo[indicemax] - arreglo[indicemax - ventana]) / ventana;

float pendiente\_despues\_min = (arreglo[indicemin + ventana] - arreglo[indicemin]) / ventana;

// Calcular la curvatura aproximada

int ventana\_curvatura = 5; // Ajustar según sea necesario

float curvatura\_total = 0;

for (int i = ventana\_curvatura; i < capacidad - ventana\_curvatura; i++) {

float pendiente\_izq = (arreglo[i] - arreglo[i - ventana\_curvatura]) / ventana\_curvatura;

float pendiente\_der = (arreglo[i + ventana\_curvatura] - arreglo[i]) / ventana\_curvatura;

curvatura\_total += abs(pendiente\_der - pendiente\_izq);

}

// Umbrales

float umbral\_pendiente = 0.1; // Ajustar según sea necesario

float umbral\_curvatura = 0.2; // Ajustar según sea necesario

// Clasificación

if (abs(pendiente\_antes\_max - pendiente\_despues\_min) < umbral\_pendiente) {

triangular = true;

} else if (curvatura\_total > umbral\_curvatura) {

// Si no es triangular, pero la curvatura es alta, es sinusoidal

cantsenoidal++;

} else {

// Si no cumple ninguna condición, podría ser otro tipo de onda

Serial.println("La onda no es cuadrada, triangular ni sinusoidal");

}

if (triangular) {

Serial.println("La onda es triangular");

canttriangular++;

} else if (cantsenoidal > 0) {

Serial.println("La onda es sinusoidal");

}

}

}

Resultados y Análisis Final

Durante el desarrollo de este proyecto, se buscaron implementar diversas estrategias para la adquisición de datos y su procesamiento. El sistema tenía la intención de identificar tres tipos de funciones (cuadrada, senoidal y triangular), usando un sensor analógico que mide la amplitud de la señal.

El sistema se basa en 2 pulsadores: el primero activa la recolección de datos del generador de señales, mientras que el segundo detiene el proceso. Se implementó un método que utiliza un arreglo dinámico para almacenar los valores de la señal y se reinicia periódicamente para evitar el colapso del sistema por exceso de datos. A lo largo de este proceso, se tomó en cuenta el análisis de las amplitudes máximas y mínimas de la señal, así como el número de ciclos y la pendiente para diferenciar entre los tipos de ondas.

El análisis mostro que el sistema es capaz de diferenciar con éxito las ondas cuadradas debido a sus cambios abruptos de amplitud, aunque presenta problemas en el análisis de funciones triangulares y senoidales, tratando de implementar estrategias como el cálculo de pendiente antes y después del punto máximo, además de pretender realizar el análisis de la curvatura para las funciones senoidales, aunque no se tuvo mucho éxito.

En conclusión, el proyecto cumple con algunas de las especificaciones propuestas en el desafío para el desarrollo de dicha actividad como la implementación de memoria dinámica, apuntadores y arreglos, sin embargo, se presentan algunas limitaciones relacionada con la cantidad de datos que el sistema puede procesar al mismo tiempo, lo que sugiere posibles mejoras en el manejo de la memoria y la optimización del código para un procesamiento más eficiente, por su parte se ve una apropiación de Arduino para su correcto funcionamiento con respecto al circuito y sus conexiones y el uso del tiempo de ejecución para apoyo en el desarrollo, aunque presenta dificultades no se descarta la posibilidad de mejoramiento y precisión.