



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

ESCUELA PROFESIONAL DE
INGENIERÍA DE SISTEMAS

PROYECTO ARQUITECTURA DE COMPUTADORAS

Sistema de monitoreo ambiental

Estudiantes:

Dioses Bellota Angel (22200209)
Sernaqué Gutierrez Jose (22200063)
Guerrero Falla, Christian (22200211)
Martínez Fuentes, Kenny (22200206)
Motta Chumbe Jeanpierre (22200030)
Triveño Daza Felipe (22200051)

Profesor:

Dr. Fermin Perez

Arquitectura de Computadoras

Grupo 2

Lima,
28 de febrero de 2024

Índice

1. Introducción	5
2. Problema General	6
3. Objetivos	6
3.1. Objetivo general	6
3.2. Objetivos específicos	6
4. Materiales	7
4.1. DsPIC33Fj32mc204	8
4.2. Pickit 3	8
4.3. ESP32	8
4.4. LM35	8
4.5. BM280	8
4.6. Sensor PIR	9
4.7. Sensor Ultrasonico	9
4.8. Cables de Conexión	9
4.9. Oscilador de Cristal	9
4.10. Buzzer (Activo)	9
5. Funcionamiento	10
6. dsPIC	12
7. Comunicaciones	13
7.1. Conexión UART con el dsPIC - ESP32	13
7.1.1. dsPIC33FJ32MC204:	13
7.1.2. ESP32:	13
7.1.3. Implementación en el MPLAB	14
7.2. Conexión I2C ESP32 - BME280	15

8. Codificación del dsPIC y ESP32	16
8.1. Codigo de MPlabX	16
8.1.1. Librerías:	16
8.1.2. Configuraciones de bits	17
8.1.3. Definiciones	18
8.1.4. Prototipos de funciones	18
8.1.5. Main	19
8.1.6. Funcion de Reloj	20
8.1.7. Funciones de ingreso y recibo de caracteres	21
8.1.8. Funciones del ADC	21
8.2. Codigo de Arduino:	23
8.2.1. Librerías	23
8.2.2. Definiciones	23
8.2.3. Prototipos de Funciones	25
8.2.4. Funcion Loop	26
8.2.5. Funcion Configuracion de Sistema	27
8.2.6. Funcion de Pedir Estado	28
8.2.7. Funcion de la Estación Meteorologica	29
8.2.8. Funcion del Sensor Ultrasonico	30
8.2.9. Funcion del Sensor BME280	30
9. Filtrado adaptativo:	32
9.1. Filtrado LMS (Least Mean Squares):	32
9.1.1. Filtro RLS (Recursive Least Squares):	33
9.1.2. Comparativa filtro LMS vs. RLS:	34
9.1.3. Codigo de Matlab:	34
9.1.4. Graficos:	38
10. Simulación	40
10.1. Proteus	40
10.2. Wokwi	40

11. Lectura y Escritura del Archivo.HEX	41
12. Resultados de consola	42
13. Manejo de Datos	43
13.1. Conexión Wifi al ThingSpeak	43
13.2. Bot de Telegram	47
14. Conclusiones	50
15. Evidencias y Anexos	51

1. Introducción

La creciente urbanización plantea desafíos para la calidad de vida en las ciudades, como el ruido excesivo que afecta la salud y el bienestar de los habitantes. Para abordar este problema, este proyecto busca mejorar la calidad ambiental y promover un entorno más habitable en la comunidad limeña, justificandolo por la necesidad de capturar datos meteorológicos en tiempo real y realizar un análisis eficiente de los mismos. En un entorno urbano dinámico como lo es Lima, donde las condiciones climáticas pueden cambiar rápidamente y tienen un impacto significativo en la calidad de vida de sus habitantes, es crucial contar con dispositivos que puedan manejar grandes volúmenes de información de manera rápida y precisa. Los microcontroladores dspPIC ofrecen la capacidad de implementar algoritmos de procesamiento de datos meteorológicos en tiempo real, lo que permite detectar patrones climáticos, identificar tendencias específicas y generar pronósticos precisos de manera casi instantánea.

Además, la integración de filtros de señales en el diseño del sistema de monitoreo meteorológico es fundamental para garantizar la precisión y confiabilidad de los datos recopilados. Los filtros permiten eliminar el ruido no deseado y resaltar las señales relevantes, lo que mejora la calidad de la información obtenida y facilita la toma de decisiones informadas en relación con el clima. En este sentido, la utilización de tecnología dsPIC33FJ32MC204 con capacidades avanzadas de procesamiento y filtros de señales integrados representa una solución óptima para abordar la necesidad de mejorar el monitoreo meteorológico en Lima, contribuyendo así a una gestión más efectiva de los recursos y una mejor planificación urbana en respuesta a las condiciones climáticas cambiantes.

En este contexto, la presente investigación se enfocará en proponer soluciones innovadoras y específicas para abordar estas problemáticas críticas. A través de la implementación de sistemas de sensores y microprocesadores buscaremos llevar la tecnología a un nuevo nivel, aprovechando la capacidad de recopilación y análisis de datos en tiempo real abordando desafíos particulares de manera inteligente y sostenible.

2. Problema General

El medio ambiente influye directamente en la salud y el bienestar de las personas. Las condiciones ambientales pueden afectar el comportamiento y la salud, especialmente en casos de contaminación y estrés. Ante emergencias ambientales, se implementan planes de acción inmediata para abordar los problemas causados por daños ambientales, con medidas específicas para las áreas afectadas y vecinas. Es fundamental mantener condiciones ambientales adecuadas para evitar sobrepassar los límites establecidos y proteger la salud pública.

3. Objetivos

3.1. Objetivo general

Diseñar, desarrollar e implementar un sistema de medición de variables ambientales utilizando un procesador dsPIC33FJ32MC204 en zonas de desastre para controlar y alertar la calidad del aire y del suelo.

3.2. Objetivos específicos

- Optimizar el sistema a fin de que realice el procesamiento de las variables ambientales y su visualización de los resultados mediante plataformas como Thiksppeak y en la computadora personal.
- Definir, en rangos, los parámetros de alerta en cada una de las variables ambientales establecidas.
- Registrar en el sistema los valores registrados durante las 24 horas del día.

4. Materiales



Figura 1: *Materiales para el sistema*

4.1. DsPIC33Fj32mc204

Es un microcontrolador de 16 bits fabricado por Microchip Technology, diseñado para aplicaciones que requieren alto rendimiento en tiempo real y procesamiento de señales digitales (DSP). Con una amplia gama de periféricos integrados, como convertidores analógico-digitales (ADC), UART, SPI, I2C y capacidades de control de motores, es utilizado en una variedad de sectores industriales, médicos, automotrices y de consumo.

4.2. Pickit 3

Es un programador y depurador de microcontroladores fabricado por Microchip Technology. Permite cargar programas en microcontroladores PIC y depurar código mediante conexión USB a una computadora. Es esencial para desarrolladores de sistemas embebidos y diseñadores de circuitos electrónicos.

4.3. ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento ampliamente utilizado en proyectos de IoT. Con su conectividad WiFi y Bluetooth integrada, potencia de procesamiento de doble núcleo, bajo consumo de energía, variedad de interfaces para sensores, amplia gama de librerías y frameworks, y capacidad de actualización a través del aire (OTA).

4.4. LM35

Es un sensor de temperatura que proporciona una salida de voltaje proporcional a la temperatura medida en grados Celsius. Con un rango típico de -55°C a +150°C y una precisión de alrededor de ±0.5°C, es ampliamente utilizado en sistemas de control de temperatura y monitoreo ambiental.

4.5. BM280

El BME280 es un sensor digital de humedad, presión y temperatura basado en conexiones para microcontroladores a través de I2C o SPI. Posee un tamaño de solo 2.5 x 2.5 mm² y una altura de 0.93 mm. Esto, sumado al bajo costo de energía, permiten su implementación en dispositivos alimentados por batería como teléfonos móviles, módulos GPS, relojes, sistemas de Autopiloto para Drones, entre otros.

4.6. Sensor PIR

Son dispositivos que detectan variaciones de la radiación infrarroja en el área de cobertura, por lo que son especialmente útiles para detectar la presencia de personas o animales a través del calor que emiten sus cuerpos.

4.7. Sensor Ultrasonico

Es un sensor que detecta la distancia entre algún objeto y el mismo sensor. Su funcionamiento es muy sencillo. El sensor ultrasónico emite un sonido ultrasónico y recibe el sonido restando el tiempo que tardó entre emitirse y recibirse calculando la distancia.

4.8. Cables de Conexion

Estos cables son esenciales en la construcción de circuitos al permitir una conexión segura y temporal entre los diferentes elementos del proyecto, lo que facilita la creación y modificación de conexiones en un protoboard o en una placa de circuito impreso.

4.9. Oscilador de Cristal

Es un oscilador electrónico que utiliza la resonancia mecánica de un cristal vibratorio de material piezoeléctrico para crear una señal eléctrica con una frecuencia precisa (20 Mhz máximo en nuestro caso). Esta frecuencia se utiliza comúnmente para controlar el tiempo, como en los relojes de cuarzo, proporcionando una señal de reloj estable para circuitos integrados digitales.

4.10. Buzzer (Activo)

Transductor electroacústico que convierte la señal eléctrica en acústica. Se utiliza comúnmente para generar alarmas sonoras en tarjetas electrónicas, multímetros, luces navideñas y otros.

5. Funcionamiento

01-Sensores

- Sensor de Temperatura (LM35)
- Sensor BME - 280
- Sensor de Distancia (HC - SR04)
- Sensor PIR (HC - SR501)
- Buzzer

02-Procesamiento de señales

- Se utiliza el conversor ADC para codificar la señal analógica a intervalos regulares y asignar un valor digital a cada muestra del sensor LM35.
- Se emplea el protocolo de comunicación serial I2C para la transferencia de datos entre dispositivos en un bus de dos cables (BME280 - ESP32).
- Se implementó el protocolo de comunicación UART para transferir datos de forma bidireccional entre los microcontroladores.

03-Visualización de datos

- Se muestran los datos medidos por los diferentes sensores en el monitor serial de Arduino IDE a 9600 baudios (baud rate).

04-Conexión Inalámbrica

- Se establece una conexión inalámbrica para recopilar los datos obtenidos en ThingSpeak.
- Además, se implementó un Telegram bot para manejar la estación meteorológica de forma remota.

05-Resultados

- Los datos obtenidos son finalmente estudiados para realizar diferentes proyecciones y simulaciones en el ámbito de la investigación ambiental.

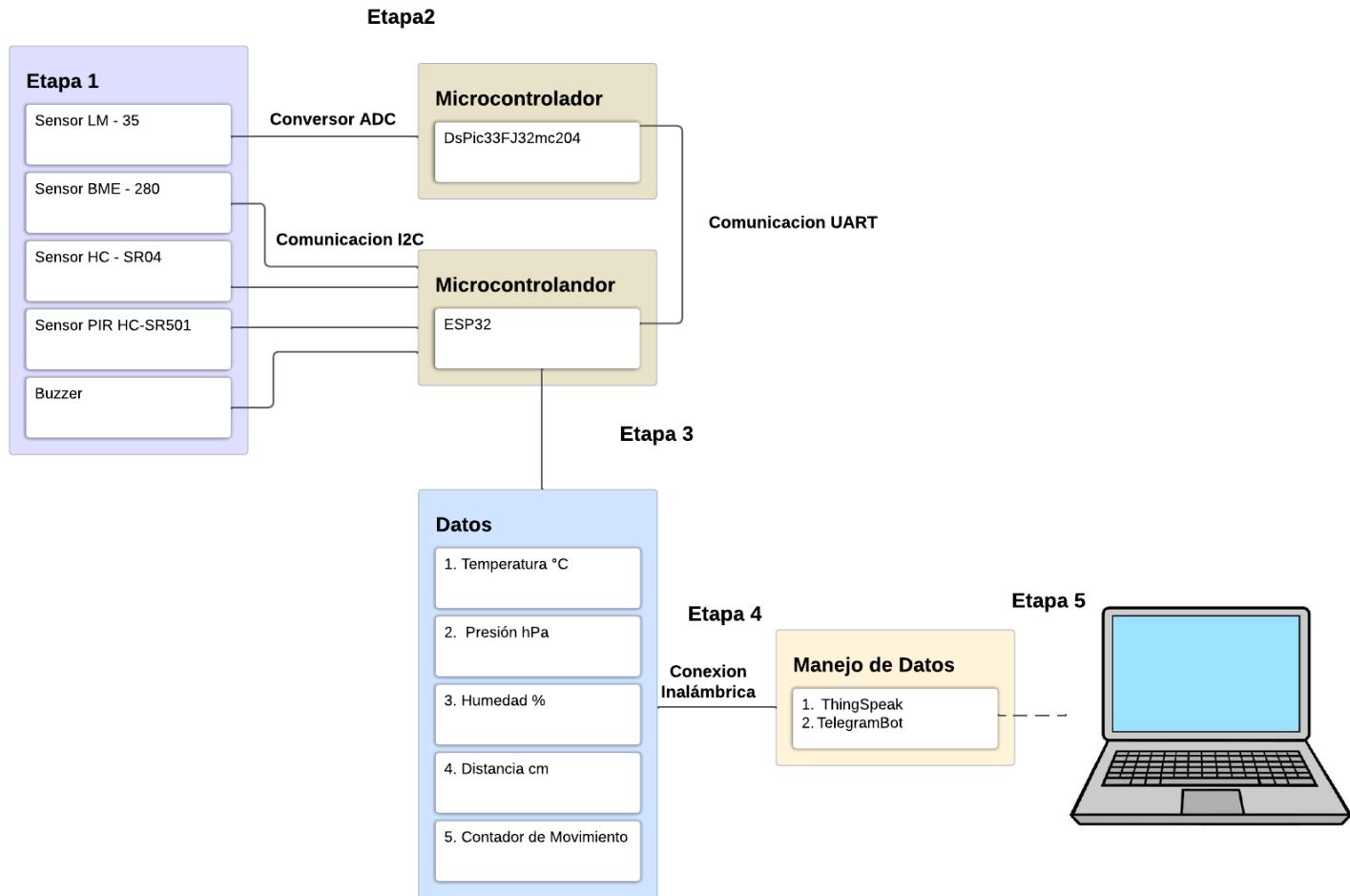


Figura 2: Esquema de interaccion sensores-PC

6. dsPIC

El dsPIC33FJ32MC204 es un microcontrolador de la familia dsPIC33 de Microchip y su arquitectura es de dsPIC, contando con 16 bits con núcleo dsPIC con características avanzadas que lo hacen adecuado para aplicaciones como aquellas que requieren procesamiento de señales digitales (DSP), con una memoria Flash de hasta 32 KB y RAM de hasta 4 KB.

Analicemos a detalles algunos perifericos del dsPIC33FJ32MC204:

- UARTs : Puede tener hasta 2 módulos UART.
- ADCs : Dispone de hasta 10 canales de ADC de 10 bits para la conversión de señales analógicas a digitales.
- I2C :Cuenta con hasta 2 módulos I2C que permiten la comunicación serial entre dispositivos.
- SPI : Este microcontrolador también está equipado con módulos SPI, que facilitan la comunicación serial síncrona con otros dispositivos.
- Temporizadores: Dispone de varios temporizadores que pueden ser configurados para generar interrupciones en intervalos de tiempo específicos o para controlar eventos en el sistema.

¿Por qué se le dice dsPIC33FJ32MC204?

- dsPIC: Indica que es un microcontrolador que pertenece a la familia de dispositivos dsPIC de Microchip, ya que esta familia contiene un núcleo DSP (Digital Signal Processor) integrado, lo que los hace ideales para aplicaciones que requieren procesamiento digital de señales.
- 33: Hace referencia a la arquitectura de 16 bits del microcontrolador. Específicamente, la familia dsPIC33 utiliza una arquitectura de 16 bits.
- FJ: Es una designación específica del modelo dentro de la familia dsPIC33.
- 32: Indica la cantidad de memoria del microcontrolador, en este caso, el modelo tiene 32 KB de memoria Flash.
- MC204: Indica que posee 44 pines.

7. Comunicaciones

7.1. Conexión UART con el dsPIC - ESP32

Esta conexión implica comunicación serial que es posible cuando ambos dispositivos tienen niveles de voltaje compatibles en sus puertos UART para garantizar una comunicación segura. Los puertos UART son los siguientes:

7.1.1. dsPIC33FJ32MC204:

- TX (Transmitir) del dsPIC33FJ32MC204 a RX (Recibir) del ESP32.
- RX (Recibir) del dsPIC33FJ32MC204 a TX (Transmitir) del ESP32.
- Conecta las tierras (GND) de ambos dispositivos.

7.1.2. ESP32:

- Conecta el pin TX del ESP32 al pin RX del dsPIC33FJ32MC204.
- Conecta el pin RX del ESP32 al pin TX del dsPIC33FJ32MC204.
- Conecta las tierras (GND) de ambos dispositivos.

En el código realizado en arduino, la conexión UART se establece en la función setup()

¹ `Serial.begin(9600, SERIAL_8N1, RXD2, TxD2);`

Esta línea inicializa la comunicación serial UART con una velocidad de baudios de 9600, configuración de 8 bits de datos, sin paridad y 1 bit de stop; asimismo, los pines RxD2 y TxD2 son los pines específicos del ESP32 que se utilizan para la comunicación UART. RxD2 es el pin de recepción (RX) y TxD2 es el pin de transmisión (TX).

En el ESP32, los pines RX y TX están asignados a los pines 16 y 17 respectivamente.

7.1.3. Implementación en el MPLAB

```

1     void Configurar_UART() {
2         RPINR18 = 0x0000; // RP0 como U1RX
3         RPOR0 = 0x0300; // RP1 como U1TX
4         U1BRG = BRGVAL;
5         U1MODE = 0; // 8 bits, sin paridad, 1 bit de parada
6         U1STA = 0; // Reset estados
7         U1MODEbits.UARTEN = 1; // Habilita UART
8         U1STAbits.UTXEN = 1; // Habilita transmisión
9         IFS0bits.U1RXIF = 0; // Limpia flag RX
10        IEC0bits.U1RXIE = 1; // Habilita interrupción RX
11    }

```

La función configura y habilita el módulo UART en el microcontrolador, define los pines de entrada y salida para la comunicación UART y establece la velocidad de baudios, el número de bits de datos, la paridad y los bits de parada. Cada parte hace lo siguiente:

1. `RPINR18 = 0x0000;`: Se establece que el pin de entrada RP0 se asigne a la función de recepción U1RX (entrada del módulo UART1).
2. `RPOR0 = 0x0300;`: Se asigna que el pin de salida RP1 se utilice como salida de transmisión U1TX (salida del módulo UART1).
3. `U1BRG = BRGVAL;`: Configura el Baud Rate Generator con el valor calculado para lograr la velocidad de baudios deseada.
4. `U1MODE = 0;`: Configura el modo del módulo UART, en este caso, se ha configurado para usar 8 bits de datos, sin paridad y 1 bit de parada.
5. `U1STA = 0;`: Reinicia los estados del módulo UART.
6. `U1MODEbits.UARTEN = 1;`: Habilita el módulo UART.
7. `U1STAbits.UTXEN = 1;`: Habilita la transmisión UART.
8. `IFS0bits.U1RXIF = 0;`: Limpia la bandera de interrupción de recepción UART.
9. `IEC0bits.U1RXIE = 1;`: Habilita la interrupción de recepción UART.

La función configura y habilita el módulo UART en el microcontrolador, define los pines de entrada y salida para la comunicación UART y establece la velocidad de baudios, el número de bits de datos, la paridad y los bits de parada. Cada parte hace lo siguiente: RPINR18 = 0x0000; Se establece que el pin de entrada RP0 se asigna a la función de recepción U1RX (entrada del módulo UART1).

7.2. Conexión I2C ESP32 - BME280

Es un tipo de comunicación serial de dos hilos que permite la transferencia de datos entre dispositivos. El I2C es usado para interconectar microcontroladores, sensores, periféricos y otros dispositivos electrónicos. En este caso, se conectará con el BME 280 y para la transferencia de datos, el maestro controla el reloj del bus I2C, mientras que los dispositivos esclavos (como el sensor BME280) responden a las solicitudes del maestro.

Los puertos I2C están disponibles en múltiples pines GPIO para mayor flexibilidad. Por ejemplo, en la placa de desarrollo ESP32 DevKit, los puertos I2C están disponibles en los pines GPIO 21 (SDA) y GPIO 22 (SCL), y en los pines GPIO 4 (SDA) y GPIO 15 (SCL).

```
1  unsigned status;  
2  status = bme.begin(0x76);
```

Aquí inicializa el sensor BME280 utilizando la comunicación I2C. El método `begin(0x76)` intenta iniciar la comunicación con el sensor BME280 en la dirección I2C 0x76. Si el sensor responde correctamente, `status` será igual a 0, lo que indica que la inicialización fue exitosa.

Entonces, esta línea establece la conexión I2C entre el ESP32 y el sensor BME280 en la dirección 0x76. La librería Adafruit_BME280 utiliza la comunicación I2C para interactuar con el sensor y leer los datos de temperatura, presión y humedad.

8. Codificación del dsPIC y ESP32

8.1. Código de MP labX

8.1.1. Librerías:

Bibliotecas que proporcionan funciones para el desarrollo de software abarcando desde operaciones básicas de entrada/salida hasta operaciones matemáticas y manipulación de cadenas. Además que está dirigido a un entorno específico de un microcontrolador (dsPIC).

- xc.h: Para el compilador XC16.
- libpic30.h: Para operaciones de temporización, configuración de puertos y manipulación de interrupciones.
- stdlib.h y stdio.h: Bibliotecas estandares de C.
- string.h: Para manipulación de cadenas de caracteres,
- math.h: Para funciones matemáticas, trigonométricas, exponenciales, etc.

```
1 #include "xc.h"  
2 #include <libpic30.h>  
3 #include <stdlib.h>  
4 #include <stdio.h>  
5 #include <string.h>  
6 #include <math.h>
```

8.1.2. Configuraciones de bits

Estas configuraciones son críticas para asegurar un funcionamiento adecuado y estable del microcontrolador. El oscilador se establece como cristal de alta velocidad (HS), lo que significa que se espera que el microcontrolador utilice un cristal de cuarzo externo para generar la frecuencia de reloj del sistema. Por último, la fuente del reloj principal del microcontrolador como PRIPLL, lo que significa que el oscilador principal se utilizará como fuente de reloj, y se activará el PLL (Phase Locked Loop) para generar una frecuencia de reloj más alta si es necesario.

```
1 // Para el FPOR
2 #pragma config PWMPIN = ON
3 #pragma config HPOL = ON
4 #pragma config LPOL = ON
5 #pragma config ALTI2C = OFF
6 #pragma config FPWRT = PWR128
7
8 // Para el FWDT
9 #pragma config WDTPOST = PS32768
10 #pragma config WDTPRE = PR128
11 #pragma config WINDIS = OFF
12 #pragma config FWDTEN = OFF
13
14 // Para el FOSC
15 #pragma config POSCMD = HS
16 #pragma config OSCIOFNC = OFF
17 #pragma config IOL1WAY = ON
18 #pragma config FCKSM = CSDCMD
19
20 // Para el FOSCSEL
21 #pragma config FNOSC = PRIPLL
22 #pragma config IESO = OFF
```

8.1.3. Definiciones

Estas definiciones permiten establecer parámetros importantes del sistema, como la velocidad del microcontrolador, la frecuencia del cristal externo, los valores para los retardos, la velocidad de transmisión serial y el voltaje de entrada, facilitando así el desarrollo y la configuración del sistema.

```

1 #define Speed 40 // MIPS deseados (maximo 40 para el dsPIC33FJ32MC204)
2 #define Crystal_Externo 20 //Valor del crystal en MHZ
3 #define Freq Speed*1000000
4 #define delay_ms(x) __delay32((Freq/1000)*x) // Delay en milisegundos
5 #define delay_us(x) __delay32(Speed*x) // Delay en microsegundos
6 #define BAUDRATE 9600
7 #define BRGVAL ((Freq/BAUDRATE)/16)-1
8 #define VOLTAJE 5 //Valor del voltaje de entrada
9 float mV1;
10 float Cels1;
```

8.1.4. Prototipos de funciones

Aquí únicamente se declaran como prototipos las funciones principales de las comunicaciones que realizarán nuestro dsPIC y ESP32, aclarar que todas son del tipo void (vacío) dado que no devuelven un valor como tal.

```

1 void adc(void);
2 void Reloj_PLL(void);
3 void Configurar_UART();
```

8.1.5. Main

Se inicia configurando todos los pines como entradas digitales, lo que asegura que no haya entradas analógicas interferiendo. Luego, se llama a las funciones Reloj PLL y Configurar UART, las cuales se encargan de configurar el PLL para generar la frecuencia de reloj adecuada y establecer la comunicación UART, respectivamente.

Después, se llama a la función ADC para inicializar la conversión analógico-digital. Se inicia la toma de muestras del ADC, se espera el tiempo de muestreo, se realiza la conversión y se calcula la temperatura en grados Celsius a partir de los datos del ADC. Posteriormente, se formatea esta temperatura como una cadena de caracteres y se envía a través de UART para su visualización.

```

1 int main(void) {
2     AD1PCFGL = 0xFFFF; //Todos son digitales
3     Reloj_PLL();
4     Configurar_UART(); // Llama funcion UART
5     adc(); // Llama funcion ADC
6     while (1) {
7         AD1CON1bits.SAMP = 1; // Start sampling
8         delay_us(10); // Wait for sampling time (10us)
9         AD1CON1bits.SAMP = 0; // Start the conversion
10        while (!AD1CON1bits.DONE);
11        mV1 = (float) ((ADC1BUFO * VOLTAJE) / (1023.0)) * 1000;
12        Cels1 = mV1 / 10;
13        char buffer[32];
14
15        // Convertir el valor de Cels1 a string y almacenarlo en buffer
16        sprintf(buffer, "% .3f C \r\n", (double) Cels1);
17
18        // Enviar la cadena a través de UART
19        Serial_SendString(buffer);
20        delay_ms(1);
21    }
22    return 0;
23 }
```

8.1.6. Funcion de Reloj

La función Reloj se encarga de configurar el reloj mediante un oscilador externo y un circuito de fase de bucle en modo de bloqueo. La configuración específica se realiza ajustando los registros PLLFBD, CLKDIVbits.PLLPRE y CLKDIVbits.PLLPOST. Luego, el bucle while espera hasta que el bloqueo del PLL se complete antes de continuar con la ejecución del programa.

Y Configurar UART(): Esta función configura el UART (Universal Asynchronous Receiver/Transmitter), que es un componente utilizado para la comunicación serial. Esta función asigna los pines de E/S relevantes para la comunicación UART, configura la velocidad de transmisión mediante el registro U1BRG, establece el modo de operación del UART (U1MODE), habilita la UART y la transmisión (U1MODEbits.UARTEN y U1STAbits.UTXEN respectivamente), y habilita las interrupciones relacionadas con la recepción (IEC0bits.U1RXIE).

```

1 void Reloj_PLL(void) {
2     // Con oscilador externo
3     int M = Speed * 8 / Crystal_Externo;
4     PLLFBD = M - 2; // M = 28;
5     CLKDIVbits.PLLPRE = 0; // N1 = 2;
6     CLKDIVbits.PLLPOST = 0; // N2 = 2;
7     while (_LOCK == 0);
8 }
9
10 void Configurar_UART() {
11     RPINR18 = 0x0000; // RP0 como U1RX
12     RPOR0 = 0x0300; // RP1 como U1TX
13
14     U1BRG = BRGVAL;
15     U1MODE = 0; // 8 bits, sin paridad, 1 bit de parada
16     U1STA = 0; // Reset estados
17     U1MODEbits.UARTEN = 1; // Habilita UART
18     U1STAbits.UTXEN = 1; // Habilita transmisión
19     IFS0bits.U1RXIF = 0; // Limpia flag RX
20     IEC0bits.U1RXIE = 1; // Habilita interrupción RX
21 }
22

```

8.1.7. Funciones de ingreso y recibo de caracteres

Estas funciones se utilizan para enviar una cadena de caracteres a través del UART, carácter por carácter, asegurando que la transmisión sea realizada correctamente sin saturar el buffer de transmisión.

```

1 void Serial_SendString(char *str) {
2     char *p;
3     p = str;
4     while (*p) Serial_PutChar(*p++);
5 }
6
7 void Serial_PutChar(char Ch) {
8     while (U1STAbits.UTXBF); // Espera buffer vacío
9     U1TXREG = Ch;
10 }
```

8.1.8. Funciones del ADC

Esta función establece la resolución del ADC a 10 bits y deshabilita el inicio automático del muestreo, mientras configura el método de inicio de conversión manual. Define el formato de salida como un entero de 16 bits y la entrada negativa del ADC como VSS (tierra), al mismo tiempo que habilita el modo de muestreo secuencial (scan). Configura el buffer para almacenar las muestras en formato de 16 palabras, y el ADC para realizar conversiones en un solo canal. Al final configura el tiempo de conversión y habilita la entrada de canal AN0 para el escaneo.

```

1 void adc() {
2
3     AD1CON1bits.AD12B = 0;
4
5     // El muestreo inicia cuando bit SAMP = 1
6     AD1CON1bits.ASAM = 0;
7
8     // Termina el muestreo e inicia la conversión
9     AD1CON1bits.SSRC = 0;
10
11    // Selecciona como se presentan los resultados de la conversión
```

```
12 // En el buffer AD1CON1 <9:8>, LA SALIDA SERA DE TIPO INT, 3 ES FRACCION
13 AD1CON1bits.FORM = 0;
14
15 // La entrada negativa sera VSS
16 AD1CHS0bits.CHONA = 0;
17 //AD1CHS0bits.CHOSA = 0 //Entrada positiva sera VSS
18
19 // Muestreo secuencia (scan) habilitado
20 AD1CON2bits.CSCNA = 1;
21
22 AD1CON2bits.VCFG = 0;
23
24 // Configurado como buffer de 16 palabras
25 AD1CON2bits.BUFM = 0;
26
27 // Solo muestra un canal
28 AD1CON2bits.ALTS = 0;
29
30 // Reloj del ADC es derivdo del sistema
31 AD1CON3bits.ADRC = 0;
32
33 // Tiempo de conversi n
34 AD1CON3bits.ADCS = 21;
35
36 AD1CON2bits.SMPI = 0;
37
38 // AD1CSSH/AD1CSSL: A/D Imput Scan Selection Register
39 AD1CSSL = 0x0000;
40 AD1CSSLbits.CSS0 = 1; // Enable AN0 for channel scan
41 AD1PCFGLbits.PCFG0 = 0; // AN0 as Analog Input
42 AD1CON1bits.ADON = 1; // Habilita ADC
43 }
```

8.2. Código de Arduino:

8.2.1. Librerías

Permiten la comunicación con sensores, la conexión a Internet, la interacción con servicios en la nube y la creación de bots de Telegram. En conjunto, estas librerías facilitan la implementación de sistemas complejos y conectados, desde la adquisición de datos hasta la interacción con el usuario.

- Wire.h: para la comunicación I2C.
- Adafruit-Sensor.h y Adafruit-BME280.h: para trabajar con el sensor BME280.
- CTBot.h: para la creación y manejo de bots de Telegram.
- Utilities.h: contiene utilidades adicionales necesarias para el proyecto.
- EEPROM.h: para acceder y manipular la memoria EEPROM.
- WiFi.h: para la conexión a redes Wi-Fi.
- ThingSpeak.h: para enviar datos a la plataforma ThingSpeak.

```
1      #include <Wire.h>
2
3      #include <Adafruit_Sensor.h>
4
5      #include <Adafruit_BME280.h>
6
7      #include "CTBot.h"
8
9      #include "Utilities.h"
10
11     #include <EEPROM.h>
12
13     #include <WiFi.h>
14
15     #include <ThingSpeak.h>
```

8.2.2. Definiciones

Las definiciones proporcionadas establecen parámetros y funcionalidades esenciales para un dispositivo con microcontrolador. Esto incluye la configuración de la presión atmosférica, el tamaño de la memoria EEPROM, la comunicación con sensores como el BME280, la interacción con la API de Telegram y la conexión WiFi. Además, se definen pines para controlar dispositivos externos y se establecen variables para el control del tiempo y la activación del dispositivo.

```
1 #define SEALEVELPRESSURE_HPA (1013.25)
2
3 #define EEPROM_SIZE 12
4
5 CTBot miBot;
6
7 CTBotInlineKeyboard miTeclado;
8
9 Adafruit_BME280 bme;
10
11 WiFiClient client;
12
13 #define RXD2 16
14
15 #define TXD2 17.
16
17 int Led = 2;
18
19 int Buzzer = 19;
20
21 int PIR = 14;
22
23 int UltrasonicTrigger = 13;
24
25 int UltrasonicEcho = 12;
26
27 int distancia = 0;
28
29 float tiempo = 0;
30
31 float espera = 60;
32
33 int movimientoContador = 0;
34
35 const int DireccionEstacion = 0;
36
37 boolean Estacion = true;
38
39 const int DireccionActivo = 1;
40
41 boolean Activo = true;
42
43 const char* ssid = "ProyectoFisica";
44
45 const char* password = "NEPTOR_bot0711";
46
47 const String token = "6760974551:AAFkFz0xHJQoe_hfij1peG7IzokqDyLiEqg";
48
49 int64_t IDchats[] = {6742384195};
50
51 int numIDchats = sizeof(IDchats) / sizeof(IDchats[0]);
52
53 const String nombre = "NEPTOR_bot Activado.";
54
55
56 // ThingSpeak configuration
57
58 char thingSpeakAddress[] = "api.thingspeak.com";
59
60 unsigned long channelID = 2341025;
61
62 const char* writeAPIKey = "18DGITXF2JLL7F62";
```

8.2.3. Prototipos de Funciones

Configura un dispositivo para actuar como un bot de Telegram llamado "NEPTORbot". Realiza varias tareas de inicialización, incluyendo la configuración de la comunicación serial, la memoria EEPROM, un sensor BME y los pines de hardware. Además, establece la conexión a una red Wi-Fi y configura los botones y teclado para interactuar con el bot de Telegram.

```
1   Serial.begin(9600, SERIAL_8N1, RXD2, TXD2 );
2   Serial.println();
3   Serial.println("Iniciando NEPTOR_bot de Telegram.");
4
5   EEPROM.begin(EEPROM_SIZE);
6   Serial.println("EEPROM Configurada (Datos de memoria restaurados.)");
7
8   Activo = EEPROM.read(DireccionActivo);
9   Serial.print("Sensor: ");
10  Serial.println(Activo ? "Activada" : "Desactivada");
11
12  Estacion = EEPROM.read(DireccionEstacion);
13  Serial.print("Estacion Meteorologica: ");
14  Serial.println(Estacion ? "Activada" : "Desactivada");
15
16  unsigned status;
17  status = bme.begin(0x76);
18  pinMode(Buzzer, OUTPUT);
19  pinMode(Led, OUTPUT);
20  pinMode(PIR, INPUT_PULLUP);
21
22  ledcSetup(0, 5000, 10);
23  ledcAttachPin(Led, 0);
24
25  miBot.wifiConnect(ssid, password);
26
27  miBot.setTelegramToken(token);
```

```

28
29     if (miBot.testConnection()) {
30         Serial.println("\n NEPTOR_bot no pudo conectarse,
31                     solicitando reinicio manual.");
32     }
33     miTeclado.addButton("Sensor", "sensor", CTBotKeyboardButtonQuery);
34     miTeclado.addButton("Estacion", "estacion", CTBotKeyboardButtonQuery);
35     miTeclado.addButton("Estado", "estado", CTBotKeyboardButtonQuery);
36     miTeclado.addRow();
37     miTeclado.addButton("Medir", "medir", CTBotKeyboardButtonQuery);
38     miTeclado.addButton("Documentacion", "https://docs.google.com
39 /document/d/1LSTlQ107GTg0WFdCJ3tccveje7qE2hPbAw19HBoGtY8/
40 edit?usp=drivesdk", CTBotKeyboardButtonURL);
41
42     for (int64_t chatID : IDchats) {
43         miBot.sendMessage(chatID, "En Lnea, Estacion Meteorologica:
44                     " + nombre);
45     }
46     tiempo = -espera * 1000;
47
48     // Inicializar ThingSpeak
49     ThingSpeak.begin(client);
50 }
```

8.2.4. Funcion Loop

La función loop() controla el flujo principal del programa, ejecutando funciones para recopilar datos meteorológicos y manejar la configuración del sistema en cada iteración del bucle.

```

1
2     void loop() {
3         EstacionMeteorologica();
4         SistemaConfiguracion();
5     }
```

8.2.5. Funcion Configuracion de Sistema

Esta posibilita la configuración remota del dispositivo mediante la mensajería de Telegram. Al recibir mensajes, verifica su origen y contenido para ejecutar acciones específicas, como cambiar el estado del sensor, activar/desactivar la estación meteorológica o solicitar mediciones. Además, actualiza los ajustes en la memoria EEPROM para mantener la configuración persistente.

```

1 void SistemaConfiguracion() {
2
3     TBMessage msg;
4
5     if (miBot.getNewMessage(msg)) {
6
7         // Itera sobre los ID de chat
8
9         for (int i = 0; i < numIDchats; i++) {
10
11             if (msg.sender.id == IDchats[i]) {
12
13                 if (msg.messageType == CTBotMessageText) {
14
15                     if (msg.text.equalsIgnoreCase("opciones")) {
16
17                         PedirEstado();
18
19                     } else {
20
21                         Serial.println("Enviar 'opciones'");
22
23                         miBot.sendMessage(msg.sender.id, "prueba 'opciones'");
24
25                     }
26
27                 } else if (msg.messageType == CTBotMessageQuery) {
28
29                     Serial << "Mensaje: " << msg.sender.firstName << "\n";
30
31                     if (msg.callbackQueryData.equals("sensor")) {
32
33                         Activo = !Activo;
34
35                         String Mensaje = "Sensor: ";
36
37                         Mensaje += (Activo ? "Activo" : "Apagado");
38
39                         Serial.println(Mensaje);
40
41                         miBot.endQuery(msg.callbackQueryID, Mensaje);
42
43                         EEPROM.put(DireccionActivo, Activo);
44
45                         EEPROM.commit();
46
47                     } else if (msg.callbackQueryData.equals("estacion")) {
48
49                         Estacion = !Estacion;
50
51                         String Mensaje = "Estacion: ";
52
53                     }
54
55                 }
56
57             }
58
59         }
60
61     }
62
63 }
```

```

28         Mensaje += (Estacion ? "Activo" : "Apagado");
29         Serial.println(Mensaje);
30         miBot.endQuery(msg.callbackQueryID, Mensaje);
31         EEPROM.put(DireccionEstacion, Estacion);
32         EEPROM.commit();
33     } else if (msg.callbackQueryData.equals("estado")) {
34         PedirEstado();
35     }else if (msg.callbackQueryData.equals("medir")) {
36         MedirBME280();
37     }
38 }
39 }
40 }
41 }
42 }
```

8.2.6. Función de Pedir Estado

Envía mensajes a través de Telegram para proporcionar el estado actual del sistema a los usuarios autorizados, incluyendo el estado del sensor y de la estación meteorológica.

```

1     void PedirEstado() {
2
3         Serial.println("Enviando 'opciones'");
4
5         String Mensaje = "Estado Actual\n";
6
7         Mensaje += "Sensor: ";
8
9         Mensaje += (Activo ? "Activo" : "Apagado");
10
11        Mensaje += " - Estacion: ";
12
13        Mensaje += (Estacion ? "Activo" : "Apagado");
14
15        Serial.println(Mensaje);
16
17        // Envía mensajes a cada ID de chat
18
19        for (int i = 0; i < numIDchats; i++) {
20
21            miBot.sendMessage(IDchats[i], Mensaje);
22
23            miBot.sendMessage(IDchats[i], "Cambiar", miTeclado);
24        }
25    }
```

8.2.7. Funcion de la Estación Meteorologica

Monitorea una estación meteorológica. Verifica si está activa y luego utiliza un sensor de movimiento PIR y un sensor ultrasónico para detectar objetos cercanos. Si se detecta movimiento o un objeto a menos de 20 cm de distancia y la estación está activa, registra la distancia detectada y realiza una medición con el sensor BME280. Esta función asegura un funcionamiento eficiente y preciso de la estación meteorológica en diversas condiciones.

```
1 void EstacionMeteorologica() {
2     if (Activo) {
3         int pirValue = digitalRead(PIR);
4         long distancia = 0.01723 * readUltrasonicDistance
5             (UltrasonicTrigger, UltrasonicEcho);
6         if (pirValue == HIGH || (distancia <= 20 and distancia > 0)) {
7             if (distancia <= 20 and distancia > 0) {
8                 Serial.println("Movimiento detectado por el sensor PIR.");
9                 Serial.println("Objeto detectado por el sensor ultras nico.");
10                Serial.print("Distancia: ");
11                Serial.println(distancia);
12                if (Estacion)
13                {
14                    ThingSpeak.setField(1, distancia);
15                    MedirBME280();
16                }
17            } else {
18                digitalWrite(Led, LOW);}
19        } else {
20            digitalWrite(Led, LOW);
21            Serial.println("Sin movimiento");
22            delay(3000);}
23    }
24 }
```

8.2.8. Funcion del Sensor Ultrasonico

Se encarga de medir la distancia utilizando un sensor ultrasónico. Envía una señal ultrasónica a través de un pin de salida, luego mide el tiempo que tarda en recibir el eco de esa señal a través de un pin de entrada. Devuelve este tiempo de vuelo como una medida de la distancia al objeto más cercano en microsegundos.

```

1   long readUltrasonicDistance(int UltrasonicTrigger,
2                                 int UltrasonicEcho) {
3
4     pinMode(UltrasonicTrigger, OUTPUT);
5     digitalWrite(UltrasonicTrigger, LOW);
6     delayMicroseconds(2);
7
8     digitalWrite(UltrasonicTrigger, HIGH);
9     delayMicroseconds(10);
10    digitalWrite(UltrasonicTrigger, LOW);
11    pinMode(UltrasonicEcho, INPUT);
12
13    return pulseIn(UltrasonicEcho, HIGH);
14 }
```

8.2.9. Funcion del Sensor BME280

Recopila mediciones meteorológicas utilizando el sensor BME280. Primero verifica si la estación meteorológica está activa y luego realiza mediciones de temperatura, presión, altitud y humedad. Los datos medidos se envían a ThingSpeak para su registro y análisis, mientras se emite un tono a través de un buzzer para indicar el resultado del envío de datos.

```

1   void MedirBME280(){
2
3     if (Estacion){
4
5       Serial.print("Temperatura = ");
6
7       String recibido = Serial.readStringUntil('\n');
8
9       Serial.println(recibido);
10      Serial.println(" C ");
11
12      Serial.print("Presi n = ");
13
14      Serial.print(bme.readPressure() / 100.0F);
15
16      Serial.println(" hPa");
17
18      Serial.print("Altitud Aproximada = ");
```

```
11     Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
12     Serial.println(" m");
13     Serial.print("Humedad = ");
14     Serial.print(bme.readHumidity());
15     Serial.println(" %");
16     Serial.println();
17     digitalWrite(Led, HIGH);
18     movimientoContador++;
19     ThingSpeak.setField(2, movimientoContador);
20     ThingSpeak.setField(3, recibido);
21     ThingSpeak.setField(4, bme.readPressure());
22     ThingSpeak.setField(5, bme.readAltitude(SEALEVELPRESSURE_HPA));
23     ThingSpeak.setField(6, bme.readHumidity());
24     int x = ThingSpeak.writeFields(channelID, writeAPIKey);
25     if(x == 200){
26         tone(Buzzer, 2700);
27         Serial.println("Datos ingresados de manera exitosa.");
28     }else{
29         noTone(Buzzer);
30         Serial.println("Datos no ingresados.");
31         Serial.println("-----");
32         digitalWrite(Led, LOW);
33         delay(3000);
34     }
35 }
36 }
```

9. Filtrado adaptativo:

9.1. Filtrado LMS (Least Mean Squares):

Es un tipo de filtro adaptativo que ajusta sus coeficientes en función de la diferencia entre la salida deseada y la salida actual del filtro, este utiliza el algoritmo de gradiente descendente para minimizar el error cuadrático medio entre la señal deseada y la salida de filtro.

La taza de aprendizaje:

- La tasa de aprendizaje, denotada como μ , controla la velocidad a la que el filtro LMS se adapta a los cambios en la señal.
- Un valor más alto de μ resulta en una adaptación más rápida, pero puede causar inestabilidad si es demasiado grande.
- Un valor más bajo de μ puede conducir a una adaptación más lenta pero más estable.

Convergencia:

- El filtro LMS converge hacia una solución óptima en el sentido de los mínimos cuadrados medios cuando la señal de entrada es estacionaria y cumple ciertas condiciones de ergodicidad y estacionalidad.
- Sin embargo, puede tener dificultades para converger en entornos no estacionarios o en presencia de ruido excesivo.

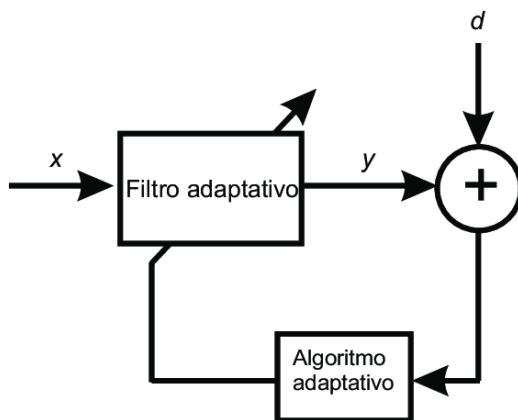


Figura 3: *Filtro LMS*

9.1.1. Filtro RLS (Recursive Least Squares):

Es un tipo de filtro adaptativo que usa el método de mínimo cuadrados recursivos para ajustar sus coeficientes, a diferencia del filtro LMS, el filtro RLS utiliza información pasada y presente para estimar los coeficientes del filtro de forma recursiva. Factor de olvido

- El factor de olvido, denotado como α , controla la importancia relativa de los datos pasados y presentes en el cálculo de los coeficientes del filtro.
- Un valor cercano a 1 da más peso a los datos más recientes, mientras que un valor cercano a 0 da más peso a los datos pasados.

Estabilidad y precision:

- El filtro RLS es conocido por su rápida convergencia y su capacidad para adaptarse a cambios en la señal de entrada. Es más adecuado para aplicaciones donde se requiere alta precisión y estabilidad incluso en presencia de ruido y cambios en la señal.

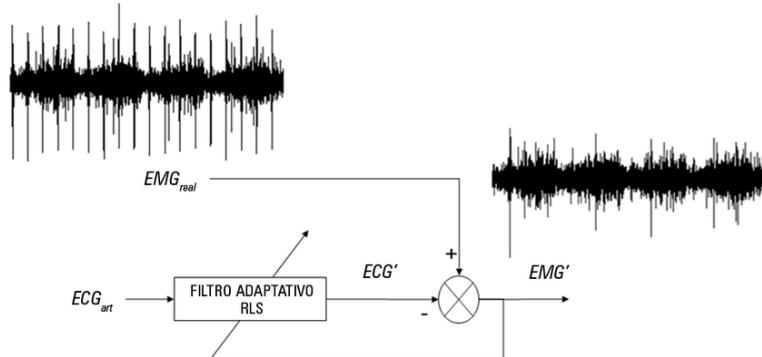


Figura 4: *Filtro RLS*

9.1.2. Comparativa filtro LMS vs. RLS:

Comparativa	LMS	RLS
Rendimiento en diferentes condiciones de señal	El algoritmo LMS es estable y converge rápidamente, lo que lo hace adecuado para adaptaciones rápidas.	El algoritmo RLS proporciona una mejor precisión en la estimación de los coeficientes del filtro en señales lentas.
Requisitos de memoria y computacionales	El algoritmo LMS es más eficiente en términos de memoria y computación, adecuado para sistemas con recursos limitados.	El algoritmo RLS, aunque más complejo computacionalmente, puede proporcionar una mejor precisión con un menor número de coeficientes del filtro, beneficioso en aplicaciones con alta precisión y recursos limitados.

Figura 5: Tabla Comparativa Filtro LMS vs. RLS

9.1.3. Código de Matlab:

Se borran todas las variables y se cierran todas las figuras abiertas para comenzar con un espacio de trabajo limpio.

```

1 clc; % Limpia la ventana de comandos
2 clear; % Borra todas las variables en el espacio de trabajo
3 close all; % Cierra todas las figuras abiertas

```

Se simulan datos de temperatura ficticios para un año, utilizando una función sinusoidal con ruido gaussiano añadido.

```

1 % Generar datos de temperatura ficticios
2 (por ejemplo, temperaturas diarias para un año)
3 dias = 365; % Número de días en el año
4 temperaturas = 20 + 5 * sin(2*pi*(1:dias)/365) + randn(1,dias);
5 % Temperaturas simuladas con una tendencia sinusoidal y ruido

```

Los datos de temperatura generados se guardan en un archivo .mat y luego se cargan de nuevo en el programa.

```
1 % Guardar los datos en un archivo .mat
```

```

2 save('datos_temperatura.mat', 'temperaturas');
3 % Guarda las temperaturas en un archivo .mat
4
5 % Cargar datos de temperatura
6 load datos_temperatura.mat; % Carga las temperaturas desde el archivo .mat

```

Se definen los parámetros para los algoritmos de filtro adaptativo. Estos incluyen la longitud de la señal de entrada (N), la longitud del filtro FIR (M), la tasa de aprendizaje para el algoritmo LMS (mu), el factor de olvido para el algoritmo RLS (lambda), y el número de iteraciones (*num_iteraciones*).

```

1 % Par metros del filtro adaptativo
2 N = length(temperaturas); % Longitud de la señal de entrada
3 M = 11; % Longitud del filtro FIR
4 mu = 0.0001; % Tasa de aprendizaje para el algoritmo LMS (reducida a n m s)
5 lambda = 0.95; % Factor de olvido para el algoritmo RLS
6 num_iteraciones = 1000;
7 % Número de iteraciones para el algoritmo adaptativo (aumentado a n m s)

```

Se inicializan las variables necesarias para almacenar resultados y realizar cálculos dentro del bucle principal.

```

1 % Inicialización de variables
2 error_medio_LMS = zeros(1,N);
3 error_medio_RLS = zeros(1,N);
4 x_i = zeros(1,M); % Inicializar el vector de entrada para el algoritmo LMS
5 x_r = zeros(1,M); % Inicializar el vector de entrada para el algoritmo RLS

```

Se realiza un bucle para cada iteración del algoritmo adaptativo. En cada iteración, se generan señales de entrada con ruido, se inicializan los coeficientes del filtro, y se aplican los algoritmos LMS y RLS para adaptar los coeficientes del filtro.

```

1 for i = 1:num_iteraciones
2 % Bucle para cada iteración del algoritmo adaptativo (incrementado)
3 % Generar señal de entrada (datos de temperatura con ruido)
4 temperatura_con_ruido = temperaturas + 0.1 * randn(size(temperaturas));
5

```

```

6    % Inicializaci n de coeficientes del filtro
7    W_LMS = zeros(1,M);
8    W_RLS = zeros(1,M);
9    P = eye(M)/lambda;
10
11    % Aplicar algoritmo LMS
12    for k = 1:N
13        x_i = [temperatura_con_ruido(k) x_i(1:M-1)];
14        y_LMS(k) = W_LMS * x_i.';
15        error_LMS(k) = temperaturas(k) - y_LMS(k);
16        W_LMS = W_LMS + mu * error_LMS(k) * x_i;
17    end
18
19    % Aplicar algoritmo RLS
20    for k = 1:N
21        x_r = [temperatura_con_ruido(k), x_r(1:M-1)];
22        Pk = P * x_r.';
23        K = Pk / (lambda + x_r*Pk);
24        s = temperaturas(k) - W_RLS * x_r.';
25        W_RLS = W_RLS + K.' * conj(s);
26        P = (1/lambda) * P - (1/lambda) * K * x_r * P;
27        error_RLS(k) = s;
28        y_RLS(k) = W_RLS * x_r.'; % Calcular la salida filtrada por RLS
29    end

```

Se calcula el MSE para cada iteración de los algoritmos LMS y RLS.

```

1 % Calcular el error medio cuadr tico para cada iteraci n
2     error_medio_LMS = error_medio_LMS + error_LMS.^2;
3     error_medio_RLS = error_medio_RLS + error_RLS.^2;
4 end
5
6 % Calcular el promedio de los errores cuadr ticos medios
7 error_medio_LMS = error_medio_LMS / num_iteraciones;

```

```
8 % Dividido por el n mero de iteraciones  
9 error_medio_RLS = error_medio_RLS / num_iteraciones;  
10 % Dividido por el n mero de iteraciones
```

Se generan gráficos para visualizar los datos originales de temperatura, los datos de temperatura con ruido, las predicciones del filtro LMS, las predicciones del filtro RLS, y el MSE a lo largo de las iteraciones.

```
1 % Graficar resultados  
2 figure;  
3 subplot(3,1,1);  
4 plot(temperaturas, 'LineWidth', 1.1);  
5 hold on;  
6 plot(temperatura_con_ruido, 'LineWidth', 1.1);  
7 title('Datos de temperatura originales vs. con ruido');  
8 legend('Original', 'Con ruido');  
9 grid on;  
10  
11 subplot(3,1,2);  
12 plot(y_LMS, 'LineWidth', 1.1);  
13 hold on;  
14 plot(temperaturas, 'LineWidth', 1.1);  
15 title('Predicci n LMS vs. temperatura original');  
16 legend('Predicci n LMS', 'Original');  
17 grid on;  
18  
19 subplot(3,1,3);  
20 plot(y_RLS, 'LineWidth', 1.1);  
21 hold on;  
22 plot(temperaturas, 'LineWidth', 1.1);  
23 title('Predicci n RLS vs. temperatura original');  
24 legend('Predicci n RLS', 'Original');  
25 grid on;  
26
```

```
27 % Graficar MSE
28 figure;
29 semilogy(error_medio_LMS , 'LineWidth' , 1.5);
30 hold on;
31 semilogy(error_medio_RLS , 'LineWidth' , 1.5);
32 legend('LMS' , 'RLS');
33 xlabel('Iteración');
34 ylabel('MSE');
35 title('Error cuadrático medio (MSE)');
36 grid on;
```

9.1.4. Graficos:

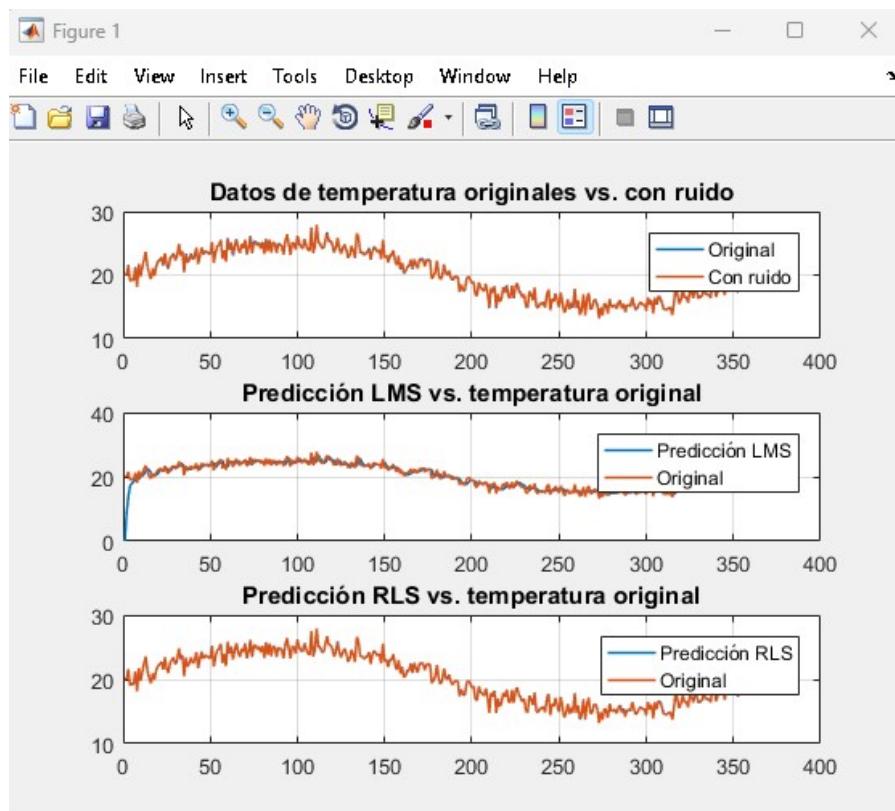


Figura 6: *Grafica de la precision respecto a la temperatura*

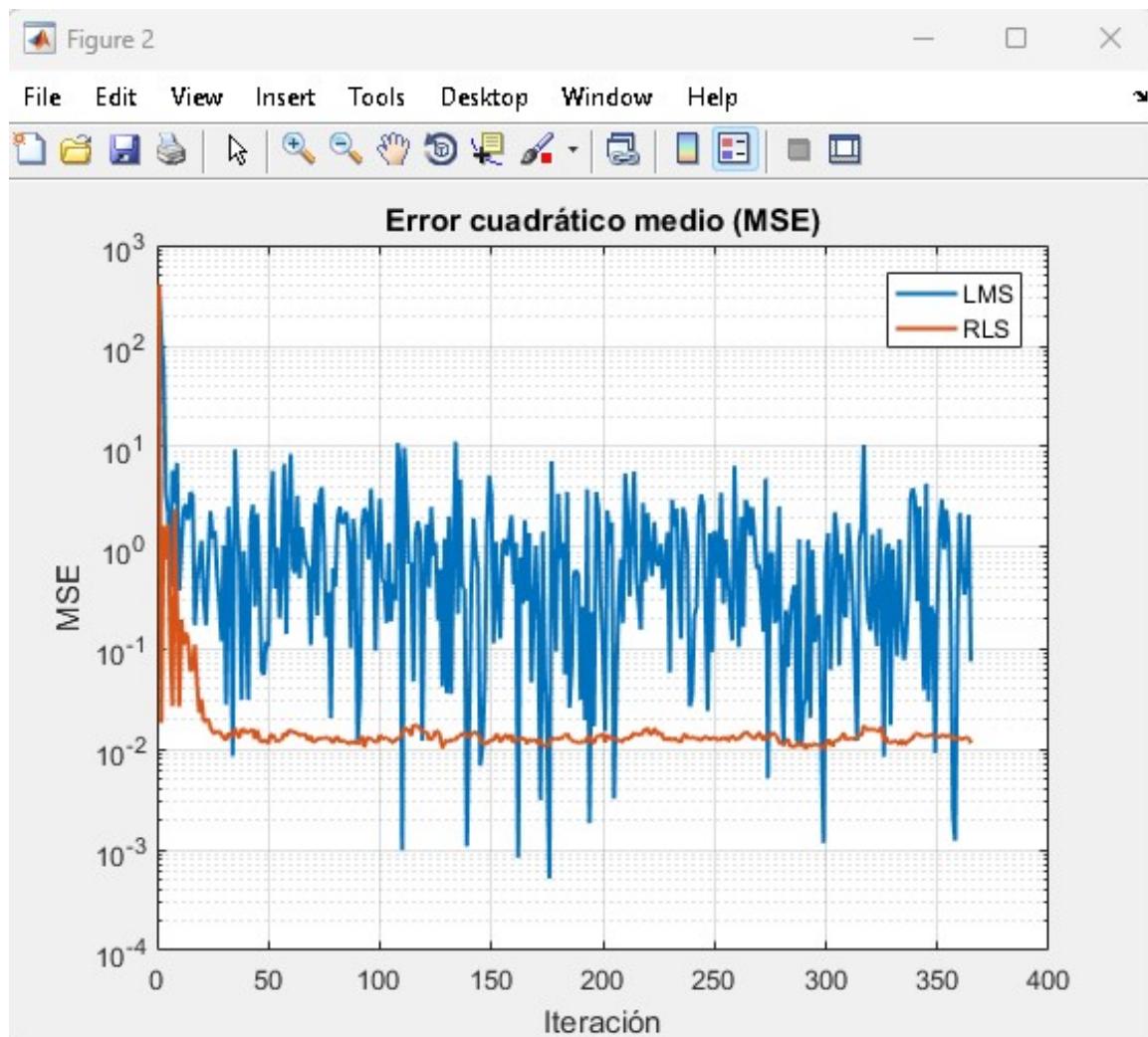


Figura 7: Grafica sobre el Error Cuadratico Medio

10. Simulación

10.1. Proteus

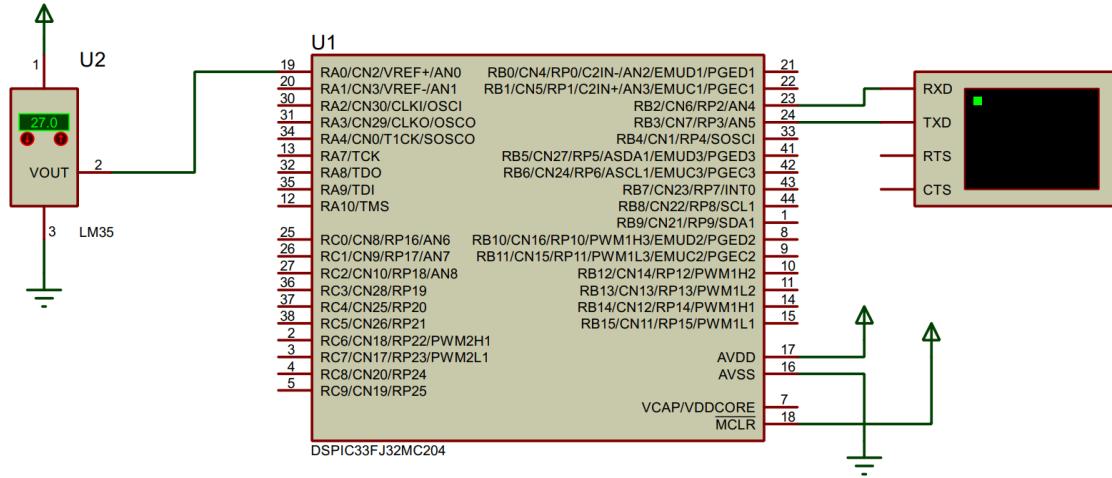


Figura 8: Simulacion del /dsPIC - LM35/ en Proteus

10.2. Wokwi

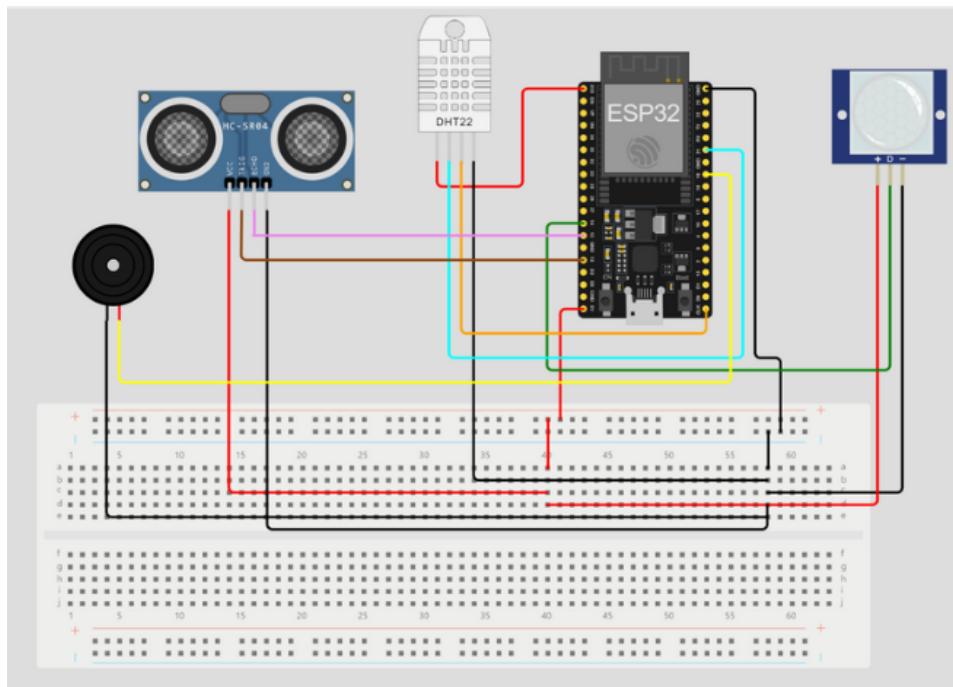


Figura 9: Simulacion del /ESP32 y sus Sensores/ en Wokwi

11. Lectura y Escritura del Archivo.HEX

Una vez programado nuestro software para el funcionamiento de nuestros microcontroladores y sensores, el siguiente paso es poder llevarlo a nuestro dsPIC para cargar el código compilado en el dsPIC desde un entorno de desarrollo en una computadora hacia el microcontrolador, facilitando su depuración del software.

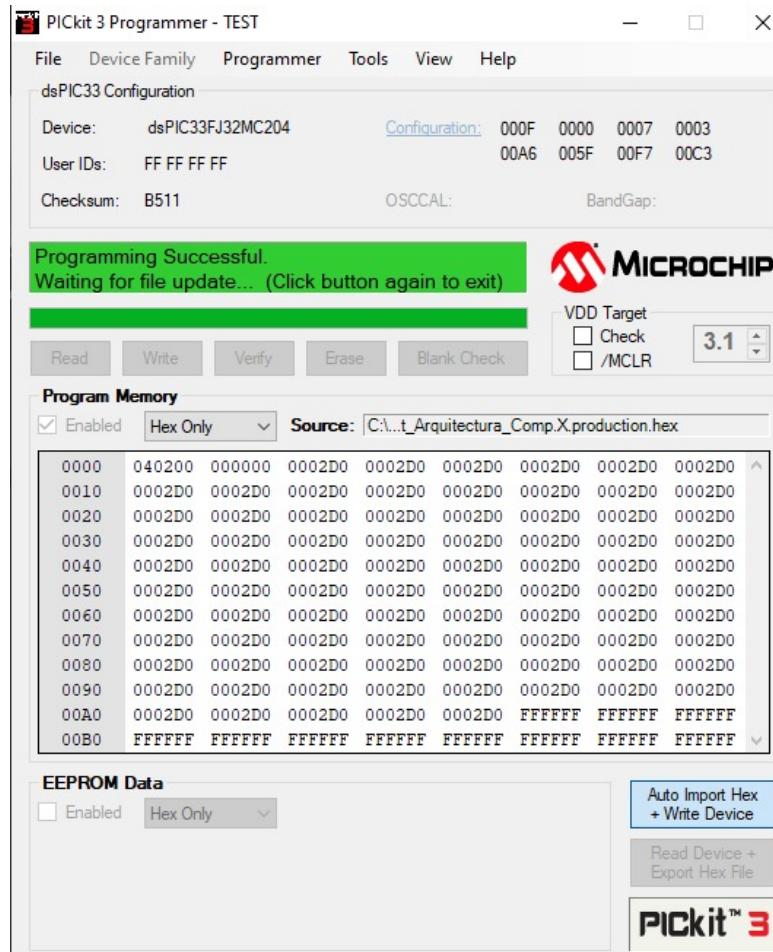


Figura 10: Materiales para el sistema

Procedemos a compilar el MPLAB X IDE y al generar el archivo .hex vamos a cargarlo dentro de la aplicación PICkit3 para su posterior escritura (write) lectura (read), esto a su vez permitirá configurar el voltaje del dsPIC ya sea automática como manualmente.

12. Resultados de consola

```
22:22:02.617 -> Iniciando NEPTOR_bot de Telegram.  
22:22:02.753 -> EEPROM Configurada (Datos de memoria restaurados.)  
22:22:02.753 -> Sensor: Activada  
22:22:02.753 -> Estacion Meteorológica: Activada  
22:22:05.496 ->  
22:22:05.496 -> Neptor_bot conectado a la red Wi-Fi.
```

Figura 11: *Inicio de Neptor*

```
21:53:08.899 -> Movimiento detectado por el sensor PIR.  
21:53:08.899 -> Objeto detectado por el sensor ultrasónico.  
21:53:08.899 -> Distancia: 13  
21:53:08.899 -> Temperature = 31.63 °C  
21:53:08.899 -> Pressure = 993.74 hPa  
21:53:08.899 -> Humidity = 50.61 %
```

Figura 12: *Impresión de Resultados*

```
22:28:26.854 -> Mensaje: Kenny Danny  
22:28:26.854 -> Enviando 'opciones'  
22:28:26.854 -> Estado Actual  
22:28:26.854 -> Sensor: Activo - Estacion: Activo
```

Figura 13: *Pedir Estado*

```
21:53:26.807 -> Sin movimiento  
21:53:29.968 -> Datos no ingresados.  
21:53:29.968 -> ---  
21:53:34.453 -> Sin movimiento  
21:53:37.614 -> Datos no ingresados.  
21:53:37.614 -> ---  
21:53:42.131 -> Sin movimiento  
21:53:45.331 -> Datos no ingresados.  
21:53:45.331 -> ---
```

Figura 14: *No detecta movimiento*

13. Manejo de Datos

13.1. Conexión Wifi al ThingSpeak

En el código que se ha realizado en Arduino, esta conexión funciona de la siguiente manera:

1. En primer lugar se importan las librerías:

```
1 #include <WiFi.h>
2 #include <ThingSpeak.h>
```

2. Configuración para enviar datos de la estación meteorológica:

```
1 // ThingSpeak configuration
2 char thingSpeakAddress[] = "api.thingspeak.com";
3 unsigned long channelID = 2341025;
4 const char* writeAPIKey = "18DGITXF2JLL7F62";
```

Aquí se configuran los parámetros necesarios para la comunicación con el servicio de ThingSpeak.

Esto incluye la dirección del servidor de ThingSpeak (`thingSpeakAddress`), el ID del canal (`channelID`), que se utiliza para identificar el canal al que se enviarán los datos, y la clave API de escritura (`writeAPIKey`), que actúa como una contraseña para permitir la escritura de datos en un canal específico de ThingSpeak.



Figura 15: Plataforma principal en ThingSpeak

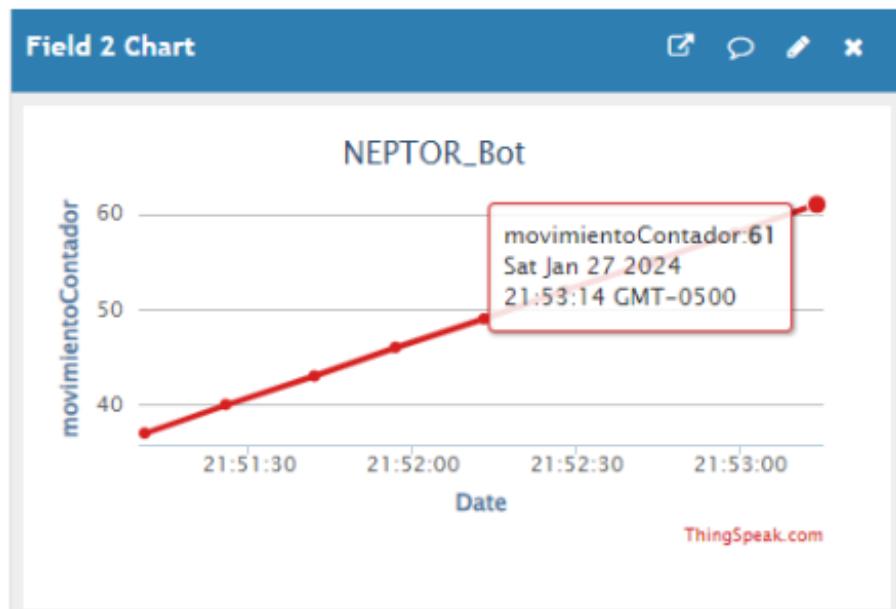


Figura 16: Gráfico en ThingSpeak respecto al contador de movimientos

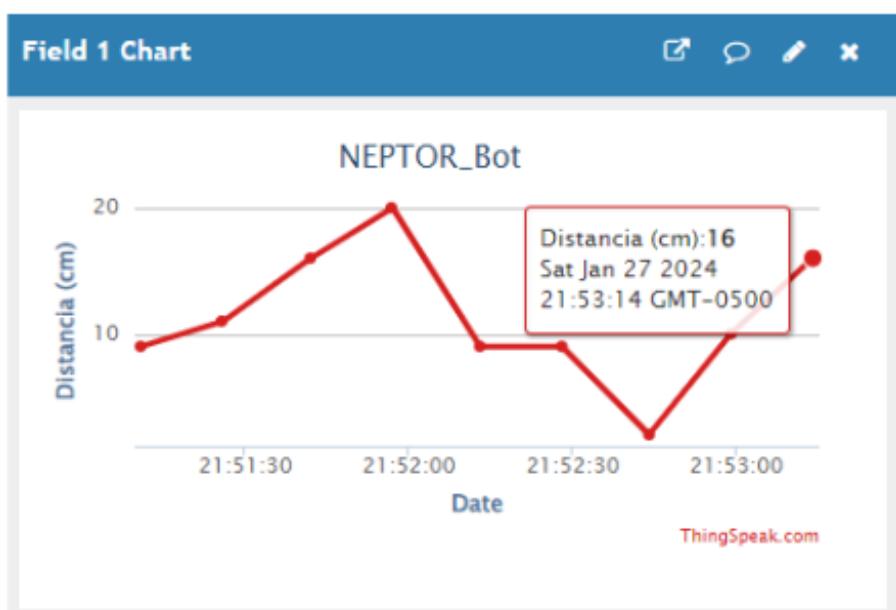


Figura 17: Gráfico en ThingSpeak respecto a la distancia (cm)

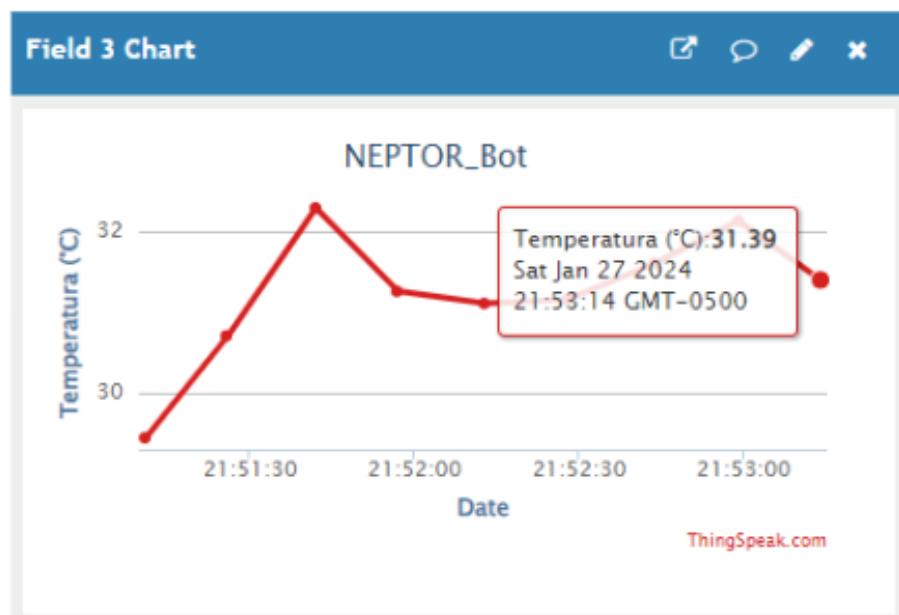


Figura 18: Gráfico en ThingSpeak respecto a la temperatura ($^{\circ}\text{C}$)

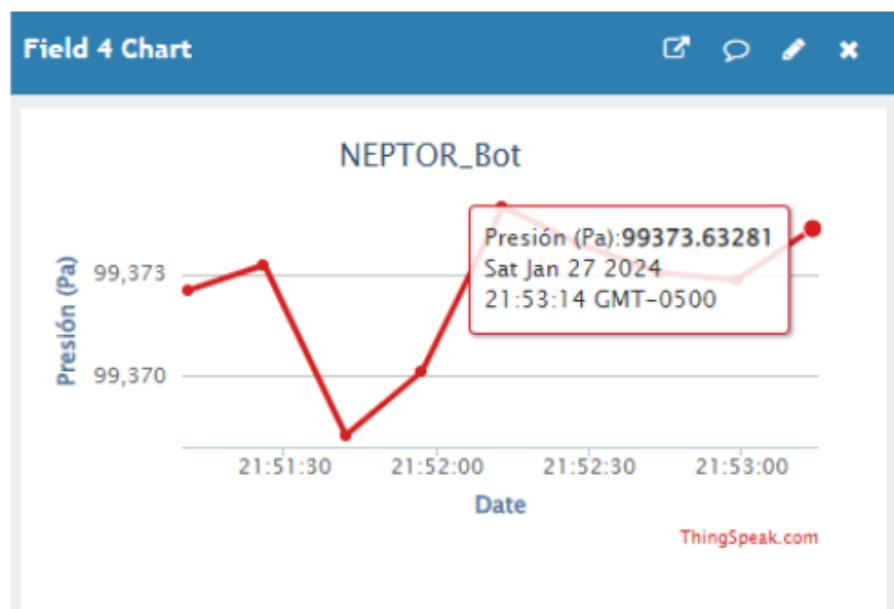


Figura 19: Gráfico en ThingSpeak respecto a presión (Pa)

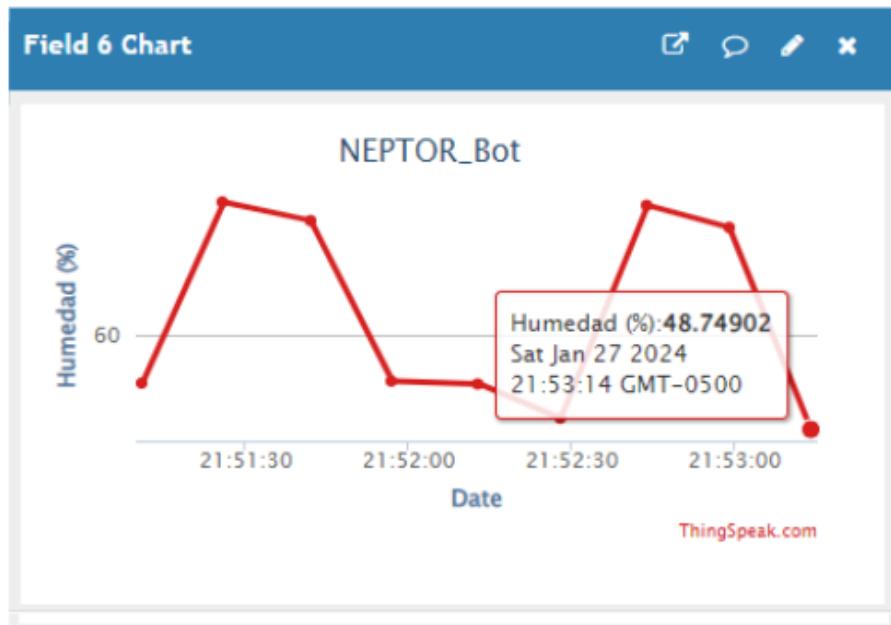


Figura 20: Gráfico en ThingSpeak respecto a humedad

13.2. Bot de Telegram

Nuestro bot de Telegram para sistemas meteorológicos basados en DSPIC33 proporciona una solución completa y conveniente para la gestión del clima. Con una interfaz intuitiva a través de Telegram, los usuarios pueden acceder fácilmente a datos meteorológicos en tiempo real y funciones de control remoto.

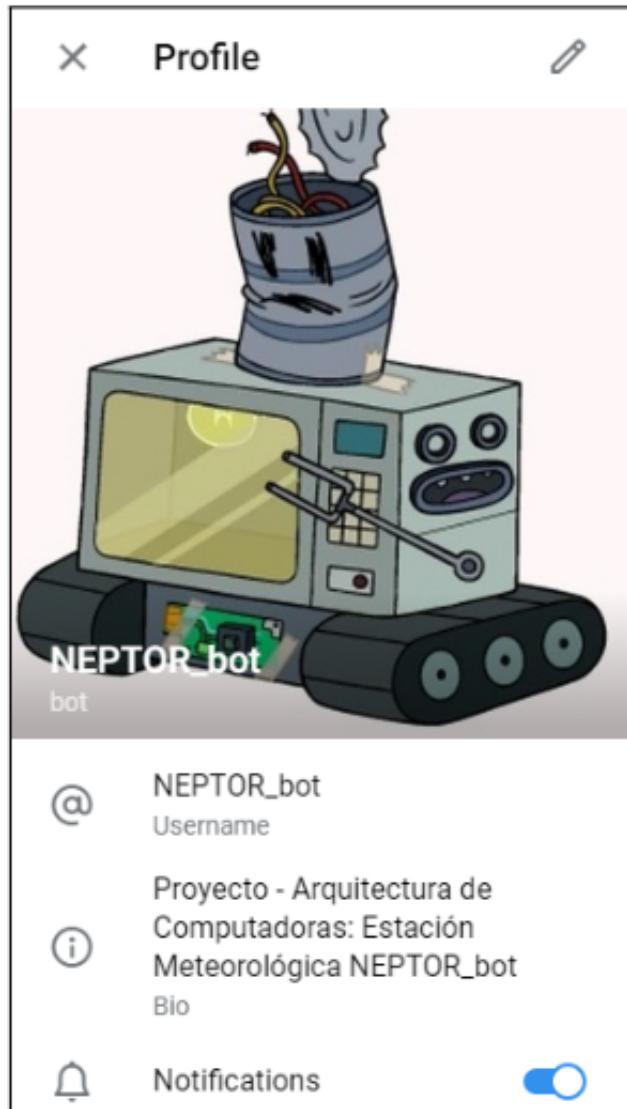


Figura 21: Perfil detallado de NEPTOR-bot en Telegram



Figura 22: *Información de NEPTOR-bot en Telegram*

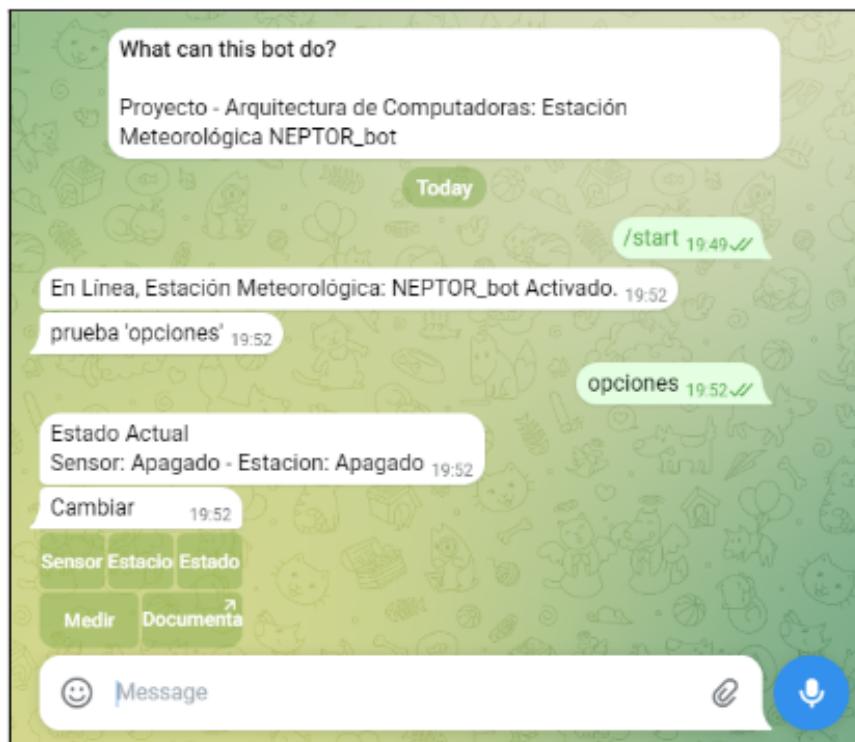
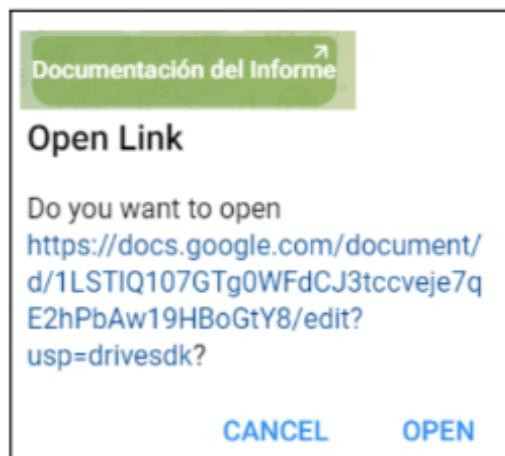


Figura 23: *Chat de NEPTOR-bot en Telegram*

Figura 24: *Interacción de Sensor*Figura 25: *Interacción de Estación*Figura 26: *Estado Actual*Figura 27: *Documentación del Informe*

14. Conclusiones

- Hemos logrado con éxito implementar un sistema de medición meteorológica utilizando sensores y el dspPIC, cumpliendo así nuestro objetivo principal. Esta implementación nos permitió monitorear en tiempo real las condiciones climáticas en la comunidad limeña, contribuyendo a mejorar la calidad ambiental y el bienestar de sus habitantes al proporcionar información precisa y oportuna sobre el clima.
- Nuestra investigación ha llevado nuestras ideas más allá, al considerar la combinación de la velocidad y precisión del dspPIC con la versatilidad de programas y sensores del ESP32. Este enfoque ha resultado en un éxito rotundo para la funcionalidad integral del sistema. La capacidad de integrar diferentes tecnologías de manera efectiva nos abre nuevas oportunidades para abordar desafíos futuros de manera más eficiente y versátil.
- La implementación de la transmisión de datos a la nube a través de plataformas como ThingSpeak ha ampliado nuestra perspectiva sobre el trabajo con circuitos, al proporcionarnos una visión más completa de cómo los datos pueden ser utilizados y compartidos de manera efectiva. Esperamos que esta experiencia nos sirva para futuras exploraciones en áreas como redes, transmisión de datos e IoT, ofreciendo una base sólida para seguir avanzando en el campo de la tecnología y la ingeniería.

15. Evidencias y Anexos

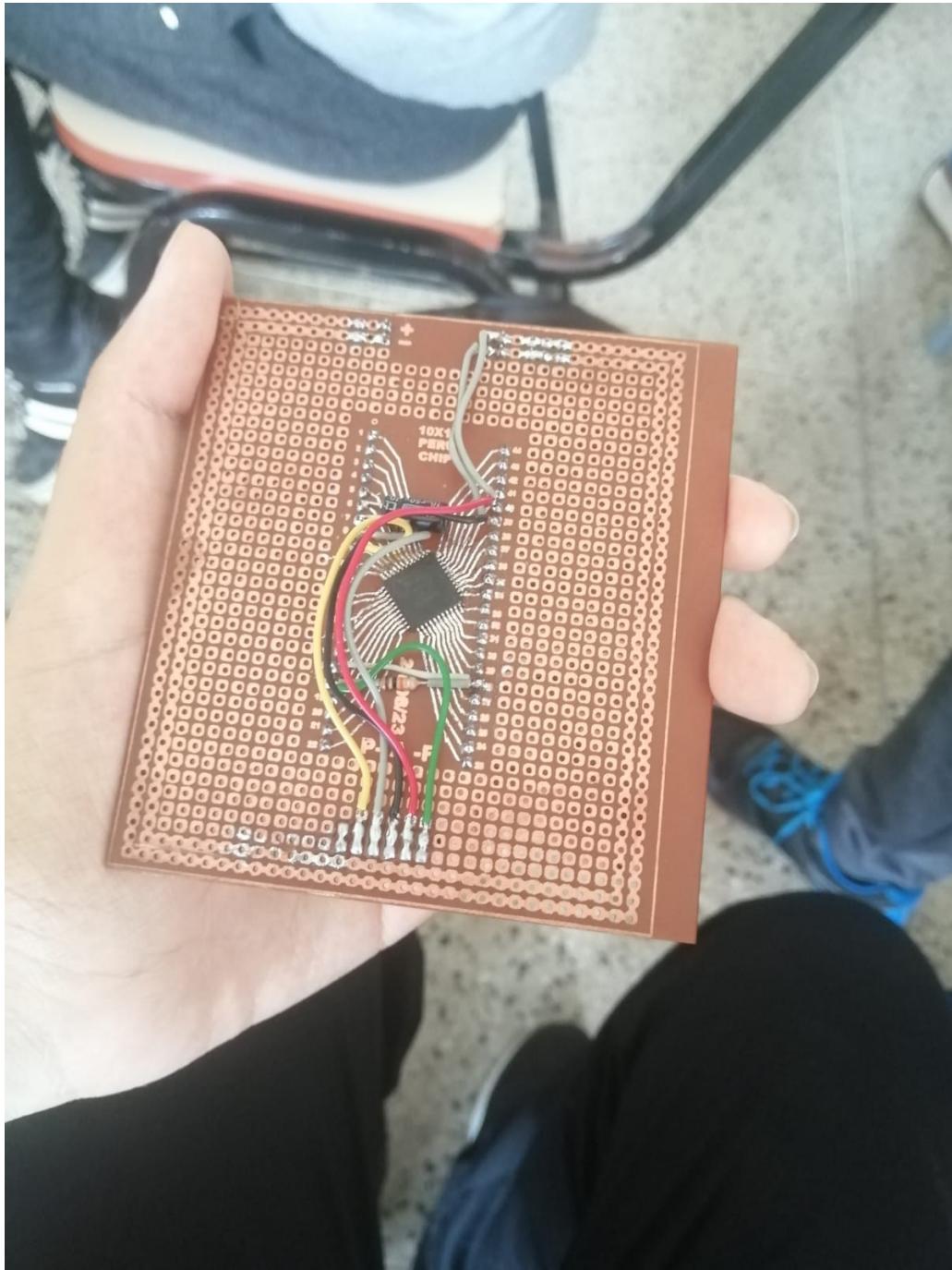


Figura 28: Vista delantera de nuestro dsPIC

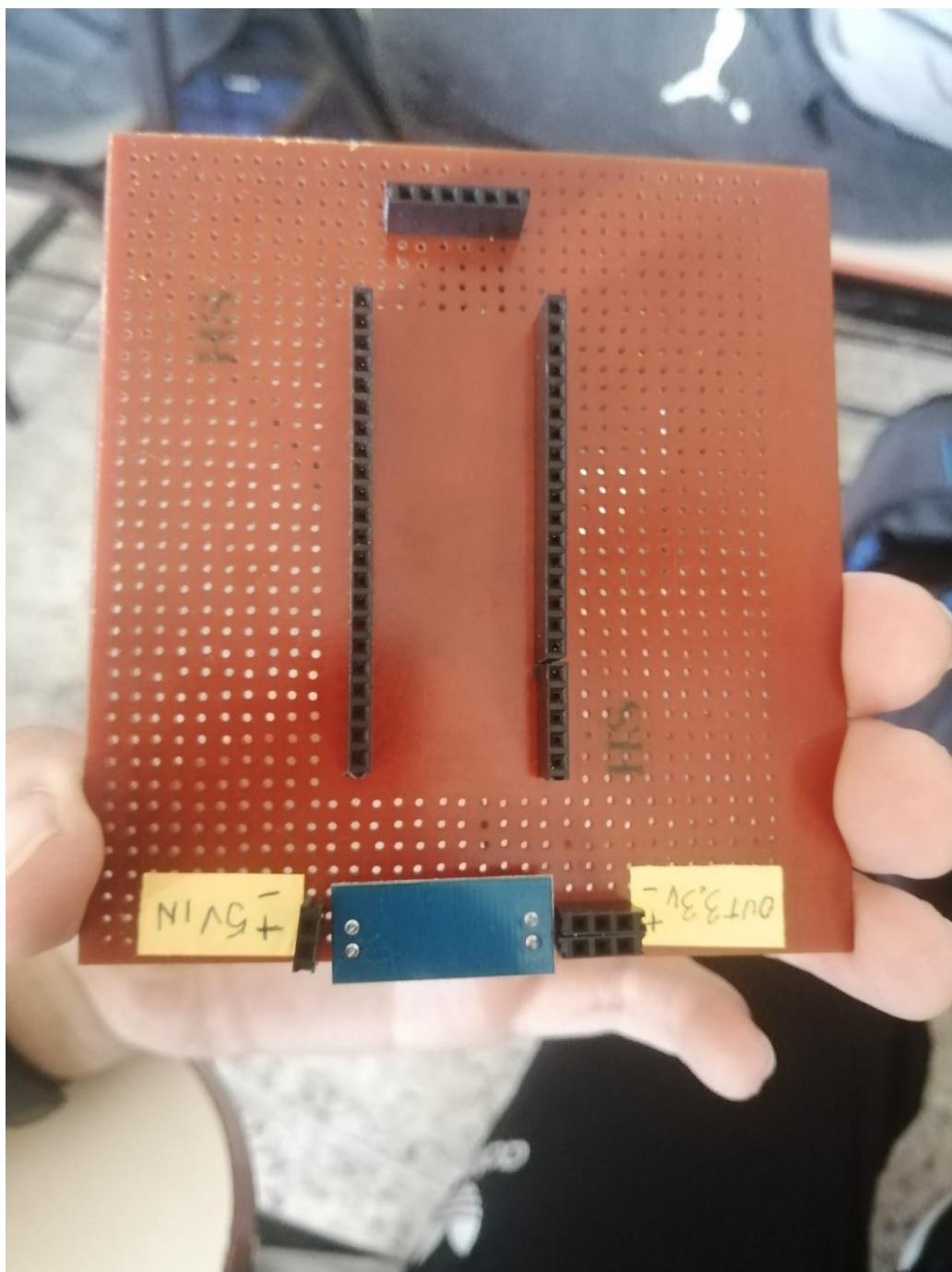


Figura 29: Vista trasera de nuestro dsPIC

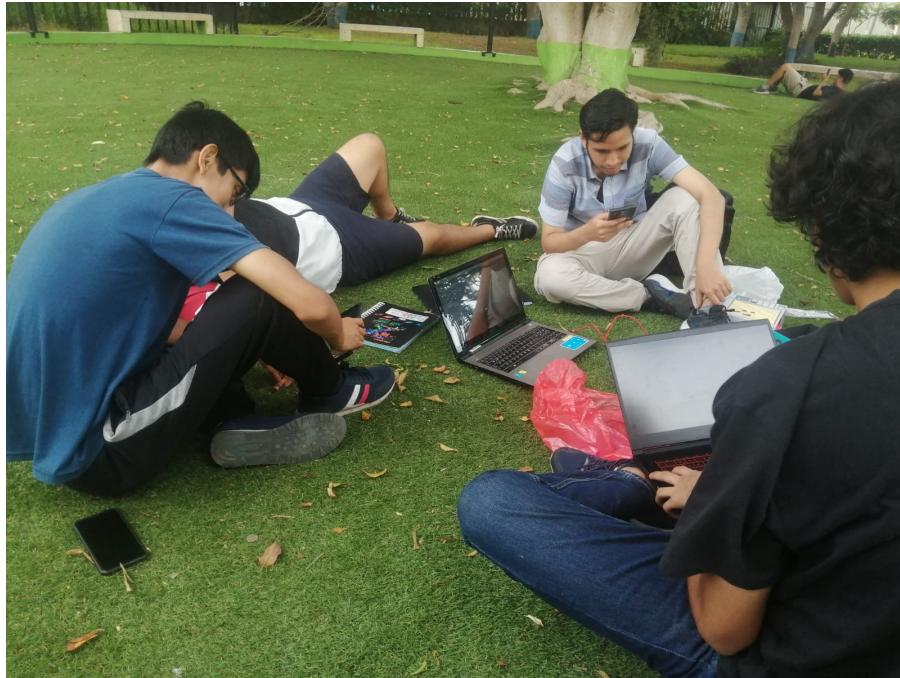


Figura 30: Reunion de trabajo presencial

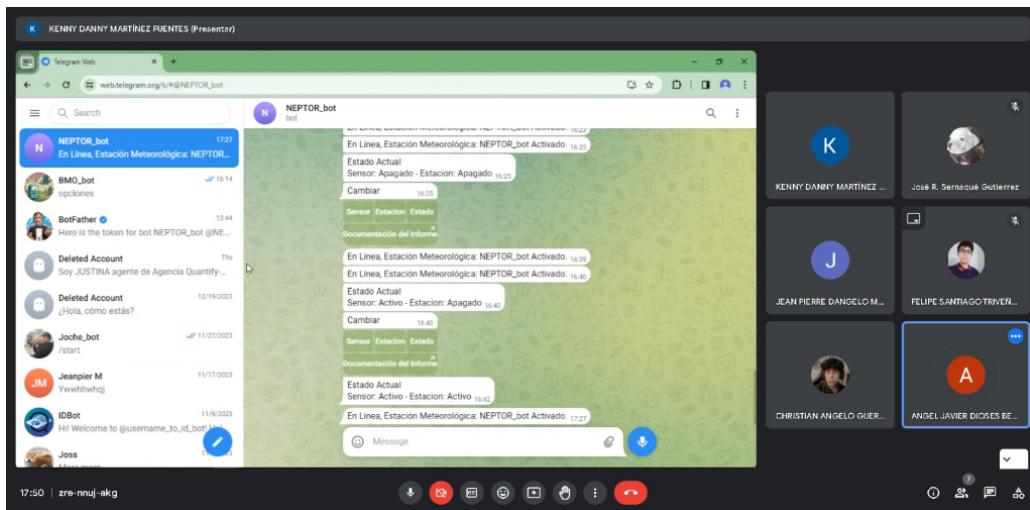


Figura 31: Reunion de trabajo virtual



Figura 32: *Proyecto construido*

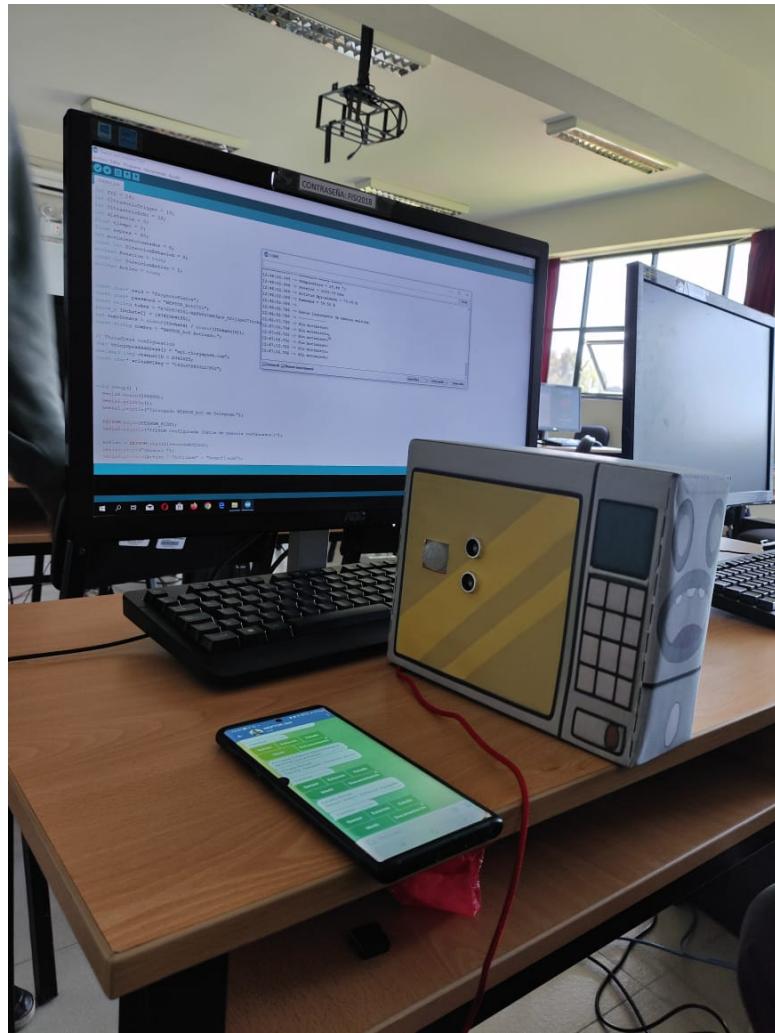


Figura 33: *Prueba de conexión con Telegram*

Referencias

- [1] V. B. A. Alexander. Sistema de medición de variables ambientales y control mediante dspic en zona de desastre. url<https://repositorio.unac.edu.pe/handle/20.500.12952/5756>, 2020.
 - [2] DSPIC33FJ32MC204. 16-bit dsc for precision motor control. url<https://www.microchip.com/en-us/product/dspic33fj32mc204>, 2020.
 - [3] Ing. Jhon Ore. Curso de DSPIC33FJ-Uso del ADC -Vídeo 9 de 14. url = <https://www.youtube.com/watch?v=D37qWEf3z3I>, November 2021.
 - [4] Ing. Jhon Ore. Curso de DSPIC33FJ-Uso del ADC y UART -Vídeo 13 de 14. url = <https://www.youtube.com/watch?v=ldLPrrP3NmE>, November 2021.
 - [5] Ing. Jhon Ore. Curso de DSPIC33FJ-Uso del UART RX -Vídeo 12 de 14. url = <https://www.youtube.com/watch?v=Prvv3UraKI>, November 2021.
 - [6] Ing. Jhon Ore. Curso de DSPIC33FJ-Uso del UART TX -Vídeo 11 de 14. url = <https://www.youtube.com/watch?v=AgZTrtiZ0z4>, November 2021.
 - [7] M. Pajek. Proyecto ESP32: Visualización del pronóstico del tiempo. url = <https://www.az-delivery.de/es/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/esp32-projekt-anzeige-fur-wettervorhersage>, 2018.
 - [8] M. Palacios and F. Ignacio. DISEÑO DE UNA ESTACION METEOROLOGICA WIFI CON ESP32 Y THINGSPEAK. url<https://repositorio.usm.cl/bitstream/handle/11673/52686/3560900267028UTFSM.pdf>, 2017.
 - [9] Z. Ramos. CURSO DE MPLAB XC16 CLASE 5 (MODULO INPUT CAPTURE). url = <https://www.youtube.com/watch?v=ZzpgmWsSg0I>, September 2020.
 - [10] P. Turnero. El algoritmo adaptativo mínimo cuadrado recursivo rls, 2015.
- [1] [6] [5] [4] [3] [7] [8] [9] [2] [10]