

UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional del Neuquén

Sistema de monitoreo para heladas

Autor: Peroni Bruno Giovanni

Director: Ing. Gustavo Monte

NEUQUÉN, 2021

1 Resumen:

En el presente proyecto, se diseña un sistema que permite monitorear variables climáticas y de estado. La idea es proveer una forma de control mucho más cómoda y amigable para el trabajador rural, ante una condición de trabajo tan desfavorable como lo es el fenómeno ambiental conocido como helada.

De esta manera, el operador podrá visualizar los valores actuales de variables ambientales como temperatura, humedad y presión atmosférica del lugar donde se ubiquen los sensores dentro de la chacra, contando con la posibilidad acceder a datos históricos que se encuentran almacenados en una base de datos.

Como es un sistema pensado para una condición en la cual se presentará el fenómeno de helada, el diseño cuenta con la posibilidad observar variables que indiquen el estado el sistema de riego contra heladas.

1	Resumen:.....	1
2	Introducción al capítulo:	19
2.1	Helada.....	19
2.2	Efectos de las heladas sobre los cultivos:	20
2.3	El microambiente vegetal	21
2.4	Tipos de heladas	23
2.5	Factores que influyen sobre la intensidad de una helada	27
2.6	Factores climáticos importantes	29
2.6.1	La temperatura del índice actino térmico	31
2.7	Instrumental meteorológico de medición	31
2.7.1	Termohigrógrafo con faja diaria	34
2.7.2	Psicrómetro	35
2.7.3	Mástil termométrico	38
2.7.4	Anemógrafo	38
2.8	Estudio micro climático de la zona cultivada	39
2.9	Tablas de resistencia al frío	40
2.10	Factores de importancia en la resistencia	41
2.10.1	La velocidad del enfriamiento:	41
2.10.2	Las condiciones del tiempo:	41
2.10.3	El estado nutricional del árbol	42
2.10.4	El fenómeno de “sobrefusión”	42
2.11	Defensa activa contra heladas	43
2.11.1	Método de riego por aspersión	43
3	Introducción al capítulo	46
3.1	Desarrollo	46
3.2	Sensor de Temperatura y Humedad Modelo DHT 22	47
3.3	Conexión del sensor y especificaciones del fabricante	48
3.3.1	Especificaciones técnicas	48
3.3.2	Especificaciones eléctricas	50
3.3.3	Dimensiones:	50
3.3.4	Condiciones de trabajo y almacenamiento	51
3.3.5	Procedimientos de recalibración:	52

3.4	Introducción al modulo acelerómetro ADXL345 [15]:	52
3.4.1	Especificaciones Importantes:	53
3.5	Placa NodeMCU WiFi ESP8266 Versión 3 [16][17]	54
3.5.1	Características generales de NodeMCU:	56
3.5.2	Esquema de entradas y salidas de NodeMCU:	57
3.5.3	Alimentación:	59
3.6	Tarjeta NodeMCU WiFi ESP32 [28][29][30]	59
3.6.1	Comparación entre ESP32 y ESP8266	61
3.6.2	Especificaciones del ESP32	63
3.6.3	Esquema de entradas y salidas de NodeMCU ESP32:	65
3.6.4	Modo Deep Sleep	66
3.7	Fuente de alimentación Protoboard 5v 3.3v Mb102	68
3.7.1	Especificaciones:	69
3.8	Modulo relé opto acoplado [34] [35]	69
3.9	Panel solar:	74
3.10	Batería Ion Litio Modelo 18650	76
3.11	Módulo cargador de batería TP4056	77
3.11.1	Características del chip TP4056	79
3.11.2	Características eléctricas [31]	80
3.11.3	Ciclo de carga	80
3.11.4	Circuito integrado de protección [32]	81
3.12	Regulador de baja caída MCP1700-3302E [33]	81
3.12.1	Características	82
3.12.2	Aplicaciones:	83
3.13	Transistor 2N2222 [35]:	83
3.13.1	Especificaciones del 2N2222	84
3.14	Modulo DC-DC Step Up 0.9-5V a 5V [36]	84
3.15	Diodo 1N4001 [37]	85
3.16	Pantalla OLED	86
3.16.1	Especificaciones	87
4	Introducción al capítulo:	90
4.1	Manejo de la información empleando base de datos y servidor	90

4.1.1 Creación de la página web	90
4.1.2 Lenguaje PHP [20]	91
4.1.3 Lenguaje HTML [21] [22]	92
4.2 Software XAMPP [23]	93
4.2.1 Panel de control de XAMPP:	94
4.3 Base de datos MySQL [25]	102
4.3.1 PhpMyAdmin [26]	103
4.3.2 HeidiSQL [27]	104
4.3.3 Creación de base de datos y tablas:	104
4.4 Desarrollo de la página web:	108
5 Introducción al capítulo:	127
6 Configuración de tarjeta NodeMCU	127
6.1 IDE de Arduino	127
6.2 Configuración de NodeMCU:	130
6.3 Desarrollo de códigos para el manejo de sensores y envío de información:	136
6.3.1 Desarrollo de código para DHT22	136
6.3.2 Desarrollo de código para BMP280	139
6.3.3 Desarrollo de código para ADXL345	141
6.3.4 Desarrollo de código de prueba de pantalla OLED	146
6.3.5 Desarrollo de código prueba para trabajar con multitareas empleando los dos núcleos del ESP32.....	150
6.3.6 Desarrollo de código para enviar datos mediante método POST con NodeMCU	152
6.3.7 Desarrollo de código para activar modo Deep Sleep.	157
6.3.8 Desarrollo de código para activación o desactivación de sistema de riego:	160
6.3.9 Desarrollo de código principal:	171
7 Introducción al capítulo	190
7.1 Desarrollo	190
7.2 Circuito ON-Off para manejar alimentación del módulo:	204
8 Introducción.....	211
8.1 Montaje y prueba de placa controladora de sistema de riego	211
8.2 Montaje y prueba de placa controladora de sensores	218
9 Trabajo a futuro.....	226

10	Conclusión.....	228
11	Bibliografía.....	230
12	Apéndice A (códigos desarrollados):	235
12.1.1	Código index.php	235
12.1.2	Código auxiliar.php	238
12.1.3	Código historicos.php	240
12.1.4	Código recibe_datos.php	244
12.1.5	Código de prueba para conexión wifi con NodeMCU	246
12.1.6	Código de prueba para sensor bmp280	248
12.1.7	Código de prueba para sensor dht22	249
12.1.8	Código de prueba para sensor ds18B20	249
12.1.9	Código de prueba de ADXL345	250
12.1.10	Código de prueba para emplear Display OLED 128x64 I2C	252
12.1.11	Código de prueba para NodeMCU ESP32 empleando sus dos núcleos	253
12.1.12	Código de prueba para envío de datos mediante método POST desde NodeMCU.....	254
12.1.13	Desarrollo de código para enviar datos mediante método POST, empleando un núcleo para el trabajo con los sensores y el otro núcleo para enviar los datos	257
12.1.14	Desarrollo de código para activar el modo sueño profundo (Deep sleep) del ESP32.....	261
12.1.15	Desarrollo de código para enviar datos mediante método POST, empleando un núcleo para controlar los sensores y el otro núcleo para enviar los datos y activar el modo sueño profundo.....	262
12.1.16	Desarrollo de código final completo	267
12.1.17	Desarrollo de código final para controlar sistema de riego mediante wifi a través de una página web levantada desde la NodeMCU y empleando dos núcleos:	277
13	Apéndice B (protocolo I2C)	285

Índice de figuras

Figura N° 1	Esquema del balance energético durante una noche de helada [1]	20
Figura N° 2	Cambio de temperatura con la altura. [2]	21
Figura N° 3	Cuando no hay capa de inversión. [2]	22
Figura N° 4	El calor irradiado por el suelo se refleja en las nubes	24

Figura N° 5	El aire frío discurre por las laderas y se acumula en el fondo. [4]	25
Figura N° 6	Absorción de energía del vapor de agua	26
Figura N° 7	Saturación del vapor de agua	27
Figura N° 8	Termómetro de mínima en un monte de duraznero en floración	29
Figura N° 9	Imagen de un termohigrógrafo	31
Figura N° 10	Psicrómetro	32
Figura N° 11	cálculo de la humedad relativa	33
Figura N° 12	Psicrómetro digital	34
Figura N° 13	Anemómetro de cazoletas.	35
Figura N° 14	Estados fenológicos de los frutales y temperaturas críticas de daño	36
Figura N° 15	Proceso de sobrefusión	38
Figura N° 16	Sensor DHT22	43
Figura N° 17	Gráfica de los pines de conexión	43
Figura N° 18	Pines de conexión del sensor a la placa	44
Figura N° 19	Parámetros de humedad relativa del DHT22	44
Figura N° 20	Parámetros de temperatura del DHT22	44
Figura N° 21	Error cometido con la humedad relativa	45
Figura N° 22	Error cometido en la temperatura	45
Figura N° 23	Características eléctricas	46
Figura N° 24	Dimensiones del sensor DHT22 en milímetros	47
Figura N° 25	Error en la medición según la temperatura de trabajo [14]	49
Figura N° 26	Resoluciones posibles de configurar en el DS18B20[14]	50
Figura N° 27	Sensor de temperatura LMT87	50
Figura N° 28	Diagrama en bloques de la arquitectura interna	50
Figura N° 29	Esquema de conexión	51
Figura N° 30	Acelerómetro ADXL345	52
Figura N° 31	Diagrama en bloques de la arquitectura interna	52
Figura N° 32	Configuración de pines	53
Figura N° 33	Descripción de la función de cada pin	53
Figura N° 34	Esquema genérico	54
Figura N° 35	Tarjeta NodeMcu WiFi ESP8266 V.3	54

Figura N° 36	Características de NodeMCU WiFi ESP8266 V.3	55
Figura N° 37	Pinout NodeMCU V3 [18]	56
Figura N° 38	Pinout NodeMCU visto desde la tarjeta NodeMCU [19]	57
Figura N° 39	Imagen de tarjeta NodeMCU ESP32	60
Figura N° 40	Diagrama en bloques del ESP32	62
Figura N° 41	Pinout de la ESP32	64
Figura N° 42	Modos de energía y sus diferencias	65
Figura N° 43	Consumos de energía de los diferentes modos de energía	65
Figura N° 44	Consumos del modo activo	66
Figura N° 45	Pinout del ESP32 con los puertos RTC diferenciados	67
Figura N° 46	Fuente de alimentación MB-102	68
Figura N° 47	Módulo Relé de dos canales	69
Figura N° 48	Características del relay	69
Figura N° 49	Especificaciones del relay	70
Figura N° 50	Pines de entrada y salida del módulo relay de dos canales	70
Figura N° 51	Esquema eléctrico de módulo relay de un canal	71
Figura N° 52	Optoacoplador empleado en el módulo	71
Figura N° 53	Relé desactivado	72
Figura N° 54	Relé activado	72
Figura N° 55	Perspectiva de entrada del módulo	73
Figura N° 56	Panel solar 5volt 220mAmp	75
Figura N° 57	Batería recargable modelo 18650	76
Figura N° 58	Módulo cargador de batería de litio 18650	77
Figura N° 59	Especificaciones brindadas por el fabricante	79
Figura N° 60	Ciclo de carga obtenido de Datasheet	80
Figura N° 61	Regulador de baja caída MCP1700	81
Figura N° 62	Transistor 2N2222	82
Figura N° 63	Pinout de transistor 2N2222	83
Figura N° 64	Modulo DC-DC Step Up	84
Figura N° 65	Diodo 1N4001	85
Figura N° 66	Display Oled 128x64 Px I2C	86

Figura N° 67	Panel de control de XAMPP	93
Figura N° 68	Módulos activados se muestran en verde	93
Figura N° 69	Inicio del servidor	94
Figura N° 70	Inicio de phpMyAdmin	94
Figura N° 71	Ruta de ubicación de la carpeta principal del servidor	95
Figura N° 72	Carpeta contenedora de la página web	95
Figura N° 73	Vista principal de la página web	97
Figura N° 74	Pestaña que se abre al clicar en Datos Históricos	98
Figura N° 75	Pestaña que se abre al clicar en Buscar Datos	99
Figura N° 76	Pestaña que se abre al clicar en “Sistema de riego”	100
Figura N° 77	Administrador de sesiones de HeidiSQL	103
Figura N° 78	Base de datos creada	103
Figura N° 79	Imagen del formato de la tabla de datos “sensores”	104
Figura N° 80	Ejemplo de carga de datos en tabla sensores	104
Figura N° 81	Imagen del formato de la tabla de datos “históricos”	105
Figura N° 82	Ejemplo de carga de datos en tabla sensores	105
Figura N° 83	Primera parte del archivo index.php	107
Figura N° 84	Segunda parte del archivo index.php	109
Figura N° 85	Tercera parte del archivo index.php	112
Figura N° 86	Imagen de index.php	115
Figura N° 87	Primera parte del archivo auxiliar.html	116
Figura N° 88	Segunda parte del archivo auxiliar.html	116
Figura N° 89	Imagen de la ventana auxiliar.html	118
Figura N° 90	Primera parte del archivo historicos.php	119
Figura N° 91	Imagen de la ventana historicos.php	120
Figura N° 92	Primera parte del archivo recibe_datos.php	121
Figura N° 93	Segunda parte del archivo recibe_datos.php	122
Figura N° 94	IDE de Arduino	125
Figura N° 95	Zonas de trabajo de Arduino	125
Figura N° 96	Zona 2 IDE de Arduino	126
Figura N° 97	Cómo seleccionar la tarjeta de trabajo.	128

Figura N° 98	Esquemático de conexión empleado para el led indicador	128
Figura N° 99	Puerto serial mostrando la correcta conexión de la NodeMCU	132
Figura N° 100	Error de conexión debido a que se colocó mal la contraseña	133
Figura N° 101	Esquemático de conexión empleado con sensor DHT22	134
Figura N° 102	Puerto serial mostrando la respuesta del sensor DHT22	135
Figura N° 103	Esquemático de conexión empleado para BMP280	136
Figura N° 104	Puerto serial mostrando la respuesta del sensor BMP280	138
Figura N° 105	Esquemático de conexión empleado con ADXL345	140
Figura N° 106	No se puede establecer comunicación con el ADXL345	143
Figura N° 107	Se observa por los valores que no hay viento	143
Figura N° 108	Se observa por los valores en X que hay viento	144
Figura N° 109	Esquema de conexión entre pantalla OLED y NodeMCU	144
Figura N° 110	Prueba de código para comunicarse con pantalla OLED	147
Figura N° 111	NodeMCU ESP32 funcionando en multitarea	150
Figura N° 112	Monitor serial respondiendo al envío por método POST	155
Figura N° 113	Información que va llegando desde la NodeMCU	155
Figura N° 114	Respuesta del ESP32 despertando cada 60 segundos	158
Figura N° 115	Página para activar el sistema de riego	166
Figura N° 116	Página para desactivar el sistema de riego	166
Figura N° 117	Esquemático empleado para medir consumo de corriente	173
Figura N° 118	Consumo de energía cuando la tarjeta NodeMCU está activa	174
Figura N° 119	Consumo de tarjeta NodeMCU en modo Deep Sleep	174
Figura N° 120	Batería de litio modelo 18650	175
Figura N° 121	Curva de carga	176
Figura N° 122	Especificaciones del TP4056	177
Figura N° 123	Esquemático del circuito de carga	177
Figura N° 124	Esquema de conexión eléctrica para circuito de carga	178
Figura N° 125	Imagen inferior de la conexión entre diodo, panel solar y módulo de carga.....	179
Figura N° 126	Imagen superior de la conexión entre diodo, panel solar y módulo de carga.....	179

Figura N° 127	Tensión de batería 18650 en función a la descarga	180
Figura N° 128	Tensión de las baterías antes de ser cargadas	181
Figura N° 129	Soporte para dos baterías en paralelo conectado al módulo cargador	182
Figura N° 130	Pinout de regulador de tensión MCP1700	183
Figura N° 131	Circuito regulador a 3,3V montado con el circuito de carga	184
Figura N° 132	Esquema parcialmente completo	184
Figura N° 133	Circuito de energía alimentando conectado a NodeMCU	186
Figura N° 134	Divisor resistivo en conjunto con el resto del circuito de carga y circuito regulador de 3,3V.....	187
Figura N° 135	Pinout del transistor 2N2222	188
Figura N° 134	Esquema del circuito a realizar	205
Figura N° 135	Imagen del circuito montado en una protoboard	207
Figura N° 136	Esquemático del circuito controlador de sistema de riego	210
Figura N° 137	Circuito cargador de batería a partir de energía solar o eléctrica	211
Figura N° 138	Modulo Step Up 3,3V a 5V	211
Figura N° 139	Medición de tensión aplicada al módulo Step Up	212
Figura N° 140	Medición de tensión de salida del módulo Step Up	212
Figura N° 141	Imagen ampliada con circuito On-Off y led indicador	213
Figura N° 142	Conexión entre la placa y el módulo relé	214
Figura N° 143	Vista superior de placa controladora de sistema de riego finalizado	214
Figura N° 144	Vista lateral de placa controladora de sistema de riego finalizado	215
Figura N° 145	Circuito de control de sistema de riego completo en placa	215
Figura N° 146	Prueba de tiempo de carga de baterías para un máximo consumo	217
Figura N° 147	Esquemático del circuito controlador de sensores	218
Figura N° 148	Circuito de carga de batería en placa sin panel solar y sin soportes de baterías.....	218
Figura N° 149	Imagen ampliada de placa con BMP280 y regleta de pines para el ADXL345.....	219
Figura N° 150	Sensor DHT22 y Led en placa controladora de sensores	219
Figura N° 151	Display oled comunicado con la placa y funcionando	220
Figura N° 152	Conexión de ADXL345 a placa controladora	220

Figura N° 153	Sistema para detectar viento	221
Figura N° 154	Sistema para detectar viento completo	221
Figura N° 155	Circuito de control de sensores completo en placa	222
Figura N° 156	Esquema de varios esclavos comunicándose por un mismo bus a un maestro.....	284
Figura N° 157	Esquema genérico del bus I2C	285
Figura N° 158	Secuencia de inicio	286
Figura N° 159	Bits transmitidos luego de la condición de inicio	287
Figura N° 160	Secuencia de parada	287

Índice de tablas

Tabla N° 1 44

Tabla N° 2 44

Tabla N° 3 45

Tabla N° 4 46

Tabla N° 5 52

Tabla N° 6 58

Tabla N° 7 Horas acumuladas de uso con la respectiva tensión medida en ese momento.....219

Sistema de monitoreo para heladas

CAPÍTULO 1

HELADAS

2 Introducción al capítulo:

En el presente capítulo se abarcan conceptos que permiten entender que es el fenómeno denominado helada, las condiciones climáticas para las cuales tenemos su formación y presencia, los daños que produce en las distintas especies de plantas, las formas de predecir el fenómeno, los instrumentos de medición que se utilizan, los métodos que permiten realizar las mediciones y por último los factores influyen en la agresividad del fenómeno para con la fruticultura

○

2.1 Helada

La helada es uno de los factores climáticos que más daño causa en la producción agrícola. Se considera helada a la ocurrencia de temperatura igual o menor a 0 °C, en el abrigo meteorológico a 1,50 metros de altura sobre el suelo independientemente de su duración e intensidad.

El fenómeno de helada como contingencia agrícola, ocurre cuando la temperatura del aire ha disminuido lo suficiente al punto tal, de producir daños en los órganos vegetales, debido a la formación de hielo en los tejidos celulares. El efecto producido por el hielo altera o impide el normal desarrollo de los órganos y sus funciones.

La formación e intensidad de las heladas depende de factores macro-climáticos (como altitud, latitud, cercanía a cuerpos de agua y otros) y topo-climáticos (por ejemplo, el efecto del relieve, que favorece el estancamiento o movilidad de masas de aire frío, o la exposición sur o norte de una ladera) las que deben ser consideradas para tomar acciones de prevención del daño por heladas. Además, existen factores microclimáticos y de técnicas de cultivo que también inciden en el riesgo de sufrir una helada. De esta manera, las heladas no pueden ser consideradas solamente como un episodio de bajas temperaturas.

Cuanto más baja es la temperatura mayor cantidad de hielo se forma en el interior de los tejidos, por lo que podría pensarse en utilizar el descenso térmico para medir la helada, pero esta medida no proporciona siempre la información adecuada para evaluar los daños, ya que éstos dependen de otros muchos factores relacionados con las condiciones de la helada y con las características de la planta.

La protección contra las heladas es una preocupación constante en el productor en cuanto a cultivos de hortalizas, frutales y florales, y de una forma especial para aquellos cultivos que dan cosechas de valor. Las heladas son un riesgo y como todo riesgo, se puede expresar en términos de probabilidades o dicho de otra forma en términos de frecuencia. [1]

2.2 Efectos de las heladas sobre los cultivos:

El efecto de la helada sobre el cultivo dependerá, entre otros factores, de la especie y del estado de desarrollo en que se encuentre la planta, siendo más sensibles las etapas desde botón floral hasta fruto pequeño. Por lo que es necesario considerar en forma muy cuidadosa la ubicación geográfica de las variedades más tempranas, donde los árboles o parras florecen antes, quedando así más susceptibles a una helada. También son importantes las condiciones propias del predio, donde se presentan distintas temperaturas mínimas en diferentes sitios, con menores temperaturas en los bajos y en las partes inferiores de laderas.

El daño por helada no se debe a la formación de hielo en el exterior de la planta. El daño se produce por cambios en el agua existente en los espacios intercelulares de los tejidos de la planta. En una helada ocurre formación de hielo en esos espacios, el cual extrae agua de las células, deshidratadas o bien forma cristales internos que las rompen.

Se llama temperatura crítica, a aquélla que comienza a generar determinados niveles de daño, lo cual depende de factores tales como el estado de desarrollo de los tejidos, especie, variedad, edad de la planta, ubicación en el predio y tiempo de exposición a la condición de helada. [2]

Los brotes jóvenes primaverales y las flores son los más sensibles por su alto contenido en agua. El hielo puede ocasionar heridas en la planta por las cuales suelen ingresar agentes patógenos. Se pueden también destruir las yemas y las flores, impidiendo que se transformen en frutos. En el caso de una helada tardía (primavera), se dañan los frutos en formación y los que sobreviven resultan con malformaciones.

2.3 El microambiente vegetal:

Es importante entender como es la interferencia energética que se produce, para poder interpretar como y donde deberán efectuarse las mediciones.

La radiación solar es la primera fuente de calor contenida en los cuerpos ubicados sobre la superficie terrestre. En tanto que la primera causa de enfriamiento es la dispersión del calor por radiación hacia el espacio. Los intercambios energéticos se producen durante el día y en noches calmas, en el comienzo de la actividad vegetativa. Ambos casos son de fundamental importancia.

Durante el día, la **radiación solar directa** atraviesa el primer estrato de la atmósfera. En una pequeña parte se refleja en componentes como el polvo atmosférico y el vapor de agua, así como **radiación solar difusa**. Otra parte de energía es absorbida por estos componentes que se calientan y la emite, como **radiación atmosférica**. Continuando su trayectoria, la energía encuentra más obstáculos que la absorben y parte de esta es reenviada como **radiación terrestre** (radiación suplementaria).

Estos cuatro tipos de energía llegan al microambiente vegetal. Algunos son reflejados, otros transformados y otros absorbidos como calor. La superficie del suelo se calienta y el calor se transmite por conducción a las capas inferiores y por convección a los estratos de aire superiores. El porcentaje de energía que se usa para la fotosíntesis es escaso. En las plantas, la linfa descendente lleva calor al suelo frío. La evaporación del suelo y la transpiración de

las plantas extraen calor del sustrato. El balance energético vegetativo en las horas centrales de un día sereno es positivo. La máxima temperatura se alcanza a nivel del suelo, en las primeras horas de la tarde.

Durante una noche fría, límpida y calma, el aporte de radiación solar directa y difusa es nulo. La energía que entra en juego es la acumulada durante el día en el terreno, en la vegetación y en la atmósfera. La primera causa del intercambio energético en el ambiente vegetal es la radiación terrestre de toda la superficie sólida (vegetación y suelo) que se dirige al espacio. Este fenómeno provoca que se enfríe primero la superficie, posteriormente las plantas y el aire que está en contacto con estas, y por último muy lentamente el suelo, debido a su gran capacidad térmica.

El calor irradiado se dispersa rápido hacia el espacio, mientras que el calor transmitido por convección-conducción mantiene el microambiente templado. Si la atmósfera está turbida, algo de energía en forma de radiación atmosférica vuelve al microambiente.

Las plantas absorben energía de radiación terrestre y la reenvían hacia lo alto y hacia el suelo como energía de retorno, respiran de noche y aportan energía de este proceso. Además, cuando se produce el rocío con el posterior congelamiento y la formación de hielo sobre las plantas, se genera una energía positiva no despreciable. La temperatura del aire aumenta a medida que se asciende en la atmósfera hasta un cierto nivel, para luego decrecer nuevamente. Este fenómeno es conocido como inversión térmica. El punto de inflexión en el que esta comienza a descender es el llamado “techo de inversión”. El origen del intercambio energético es la radiación terrestre de onda larga. Primero se enfría la superficie radiante (suelo - plantas) y posteriormente el aire que la rodea, por convección. [1]

Sistema de monitoreo para heladas

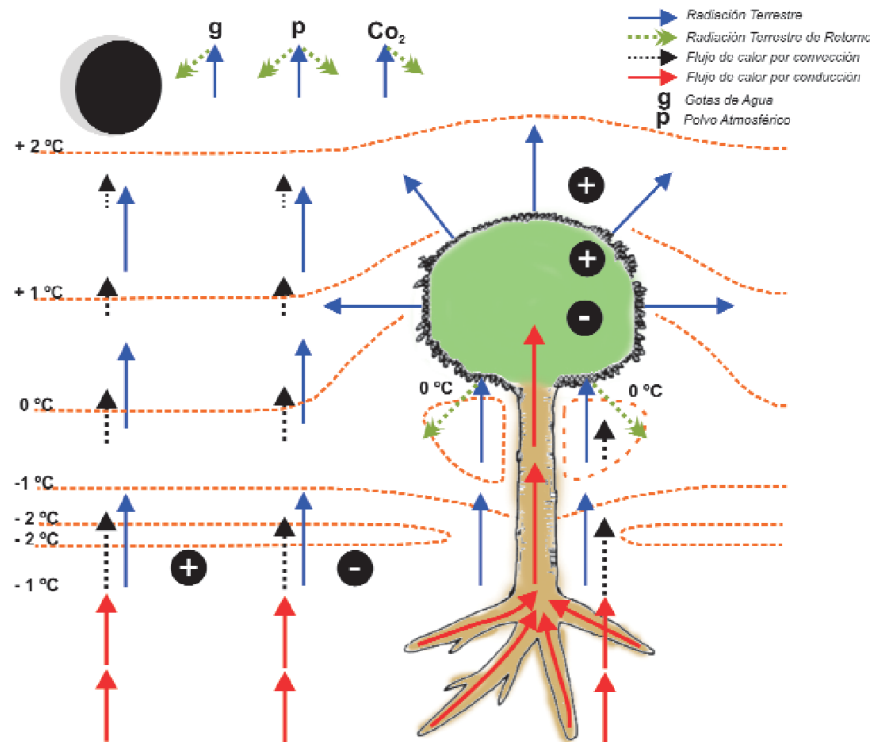


Figura N° 1 Esquema del balance energético durante una noche de helada [1]

2.4 Tipos de heladas [1][2]:

En primer lugar, podemos clasificar las heladas según el origen climatológico:

- Heladas de radiación o heladas radiactivas:

Son las más comunes. Ocurren en noches despejadas y sin viento. Bajo estas condiciones, el suelo pierde energía y se enfría rápidamente, disminuyendo la temperatura del aire que está sobre él. En ausencia de viento, el aire no se mezcla, y como consecuencia las capas de aire que están cerca del suelo serán más frías que las superiores, y esto favorece el desarrollo de las heladas. Además del viento, hay diversos factores que influyen en la aparición y conservación de las heladas, como el tipo de suelo, la cobertura vegetal y el relieve. Las largas noches de invierno también ayudan al descenso de temperatura, ya que el suelo pasa más tiempo perdiendo energía. En la noche, la temperatura de las yemas de las flores, los frutos y del árbol en general, es inferior a la del aire que los rodea. La temperatura del aire aumenta

hasta el llamado “techo de inversión”, a partir del que comienza a descender cuanto mayor altura se alcance.

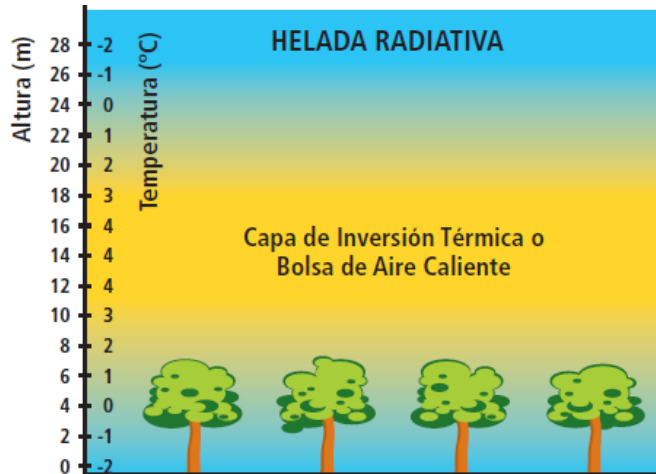


Figura N° 2 Cambio de temperatura con la altura. [2]

- **Heladas advectivas:**

Se producen cuando las temperaturas llegan a bajo cero °C, con el acompañamiento de brisas regulares a fuertes. Este movimiento de aire alcanza velocidades del orden de 1 a 2 m/seg. (3.6 a 7.2 km/h) medida a la altura de los árboles. Son condiciones más persistentes, pudiendo extenderse por varias horas en la noche y parte de la mañana o por varias noches seguidas. Algunas heladas del tipo radiactivas pueden, en determinado momento de la noche, transformarse en advectivas. Estas heladas se asocian con aire más seco y frío, por lo que son más dañinas para las plantas. Se presentan ocasionalmente, pero por sus características generan grandes daños.

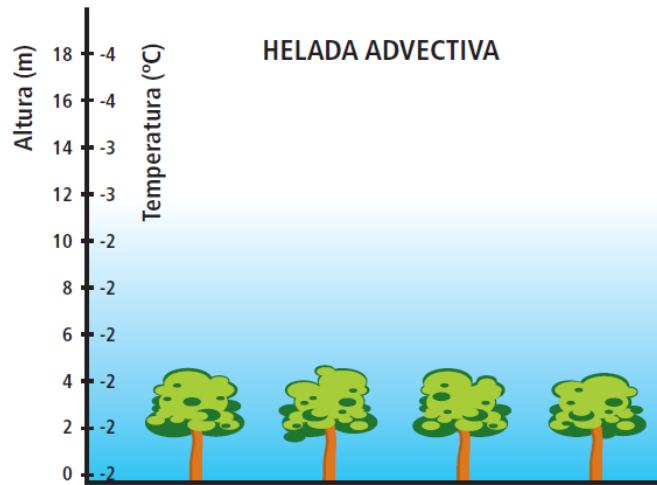


Figura N° 3 Cuando no hay capa de inversión. [2]

- **Heladas evaporativas:**

Ocurren luego de un evento de lluvias originadas por el pasaje de un frente frío. Se producen cuando se evapora el agua que está sobre la superficie de los órganos vegetales, produciendo un marcado descenso térmico con temperaturas por debajo de los 0 °C y en consecuencia daños por frío en los tejidos. La energía necesaria para provocar la evaporación del agua es de 600 cal./g, que es tomada o extraída del vegetal que sufre un rápido enfriamiento. Normalmente, es al amanecer cuando los rayos solares inciden sobre tejidos mojados o cubiertos de hielo, provocando este rápido fenómeno de enfriamiento evaporativo. Cuando lo que se evapora es el hielo, se produce el fenómeno de “sublimación, es decir el pasaje del estado de hielo a vapor de agua. Para ello son necesarias 660 cal/gr de energía que son provistas también por el vegetal. Podemos realizar otra clasificación en cuanto a la apariencia. Lo más común es diferenciarlas por el color que adquiere la vegetación. De esta manera, es posible encontrar:

- **Heladas blancas:** este tipo de helada se caracteriza debido a la escarcha blanca que se forma sobre las superficies y las hojas. Ocurren cuando la temperatura desciende de 0 °C con elevada humedad en el ambiente. Esto genera saturación del ambiente,

100% de humedad relativa y la formación de rocío sobre los vegetales. Este rocío indica que el agua sufrió un cambio de estado, al pasar de vapor en atmósfera a agua líquida, produciendo así una liberación de calor de 600 cal/gr al medio ambiente y formándose pequeñas estructuras de hielo en todas las superficies. Dicho proceso de enfriamiento hace que el descenso térmico sea lento. La escarcha formada, funciona como protección al frío extremo, ya que el hielo aísla a la planta de las bajas temperaturas y no se congela. [3]

- **Heladas negras:** son mucho más peligrosas para los cultivos y reciben este nombre por la coloración de las hojas luego de ser afectadas por este fenómeno. Se produce cuando el aire tiene muy poca humedad y no se observa la formación de hielo. Entonces el frío afecta directamente a las plantas congelando los líquidos que están dentro de ella, dañando a las células. Luego de unos días, los tejidos afectados de la planta adquieren una coloración negra debido a su destrucción. [3]

Por último, es posible clasificarlas según la época de ocurrencia:

- **Helada primaveral:** también conocidas como heladas tardías, son bastantes dañinas en el valle patagónico y afectan a las plantas durante periodos muy activos. Se presentan por descensos de temperatura ambiental.
- **Helada otoñal:** también conocida como heladas tempranas, se produce por la llegada de las primeras masas de frío. Provocan daños en el proceso de formación de los cultivos, pudiendo interrumpir bruscamente los procesos de crecimiento y habitualmente se les atribuye la disminución del rendimiento de cosechas en una región.
- **Helada invernal:** se forman durante el invierno cuando la temperatura ambiental disminuye notablemente. En invierno la mayoría de los cultivos se encuentran en reposo, con mejor disposición para soportar bajas temperaturas.

2.5 Factores que influyen sobre la intensidad de una helada [4][5]:

- **La nubosidad:** la irradiación terrestre es muy intensa cuando el cielo está despejado. Cuando el cielo está nublado, una parte del calor que irradia la tierra se refleja en las nubes y vuelve a la superficie terrestre, calentándola de nuevo.

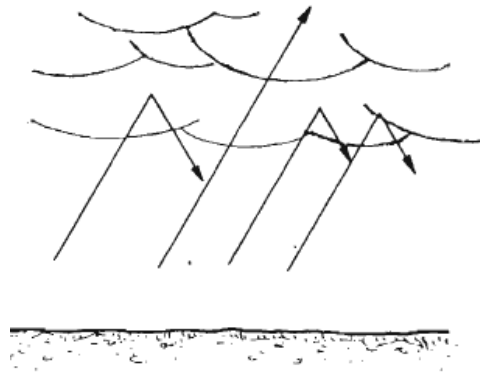


Figura N° 4 El calor irradiado por el suelo se refleja en las nubes

- **El viento:** la irradiación del calor de la tierra ocasiona un enfriamiento de las capas atmosféricas que están en contacto con el suelo. Este enfriamiento es muy intenso hasta una altura que suele variar de 10 a 100 metros. Por encima de esa altura el aire está más caliente y al tener menor densidad, no se mezcla con el aire frío situado a nivel inferior. Cuando hay viento moderado se mezclan ambas capas de aire y con ello aumenta la temperatura del aire que rodea a las plantas, disminuyendo el riesgo de helada.
- **El grado de humedad:** cuando la humedad del aire es muy elevada se producen condensaciones del vapor de agua. Este proceso implica un desprendimiento del calor de ese vapor de agua, con lo cual aumenta la temperatura del ambiente. Por eso, cuando la humedad relativa del aire es elevada, la irradiación del calor terrestre durante la noche provoca nieblas, pero no heladas. Además, la niebla forma al igual que las nubes, una pantalla protectora que evita una pérdida excesiva del calor terrestre, con lo cual disminuye el riesgo de helada. Las nieblas y las heladas de irradiación se producen con análogas situaciones: aire frío y denso, cielo despejado y viento en calma. En las mesetas y zonas altas, donde el aire es seco, se producen heladas; en los valles de los ríos donde abunda la humedad se producen nieblas. En

resumen, la ausencia de nubosidad, el aire seco y transparente y la ausencia de viento son los factores meteorológicos que favorecen las heladas de irradiación. Además de los elementos meteorológicos citados existen otros factores, ajenos a los meteoros, que también influyen en la intensidad de las heladas de irradiación: estos factores son, entre otros, la topografía del terreno y la constitución del suelo.

- **La topografía del terreno:** el aire frío es más denso que el aire caliente por lo que se sitúa junto al suelo. Cuando el terreno está en pendiente, el aire frío discurre hacia las partes más bajas, de modo semejante a como lo haría una corriente de agua. De esta forma, el aire frío discurre por las laderas y se acumula en el fondo de los valles, por lo que en dichas zonas aumenta el riesgo de heladas.



Figura N° 5 **El aire frío discurre por las laderas y se acumula en el fondo. [4]**

- **La constitución del suelo:** los suelos sueltos y pedregosos se enfrían con más rapidez que los suelos compactos, debido a que estos conducen mejor el calor y tienen una mayor exposición a la intemperie. Por esta causa, en igualdad de otras circunstancias, las heladas son más frecuentes en los suelos sueltos que en los suelos compactos.

2.6 Factores climáticos importantes [1]:

- **El vapor de agua:** es importante entender el papel que juega el vapor de agua en el descenso térmico nocturno, en el tipo de helada y en su duración. Cuando se produce el enfriamiento y se forma rocío, indica que en el cambio de estado del agua ha habido una importante liberación de calor al medio (600 kcal/l), fenómeno que atenúa el proceso de enfriamiento. Por otra parte, el vapor de agua absorbe energía de onda larga emitida por las plantas y los otros cuerpos del ambiente. En el siguiente gráfico, se puede observar que la mayor absorción de energía se produce en los seis micrones de longitud de onda y en menor medida en los 10 - 12 micrones, sector de la ventana atmosférica.

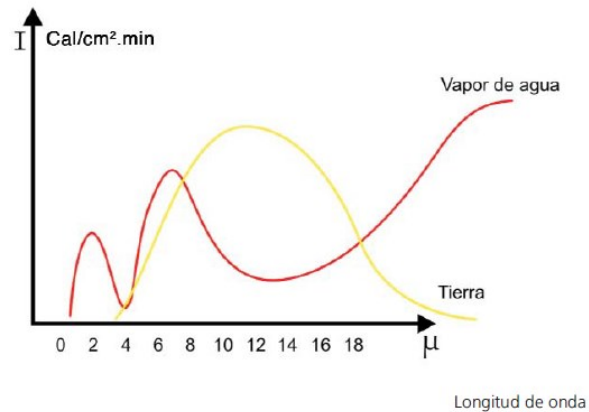


Figura N° 6 Absorción de energía del vapor de agua

Y en el siguiente gráfico se observa el contenido máximo de vapor de agua de la atmósfera para cada temperatura. Por ejemplo, para 0 °C es de 4.83 gr/m³.

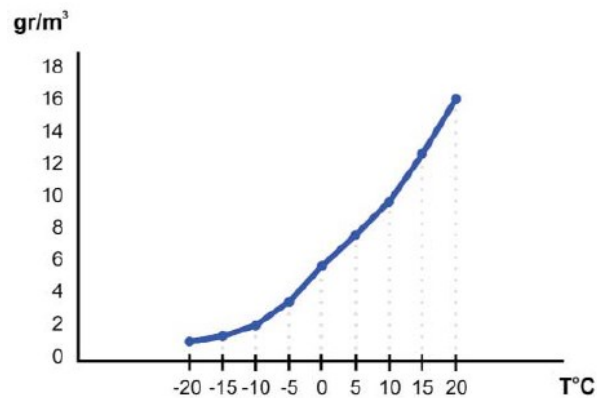


Figura N° 7 Saturación del vapor de agua

- **Punto de rocío:** es la temperatura en la cual el vapor de agua contenido en una masa de aire se condensa, lo cual generalmente sucede por disminución de la temperatura. En las horas previas a la helada, el punto de rocío es un dato de utilidad dado que permite conocer a qué temperatura se producirá la saturación de la masa de aire. Valores bajos indicarían que la humedad relativa (HR) es baja y que el descenso térmico será importante. Por el contrario, puntos de rocío cercanos a 0°C indicarían alta humedad ambiental, formación de rocío, descensos térmicos lentos y mínimas no muy extremas.

2.6.1 La temperatura del índice actino térmico (IA)

Es la temperatura medida en un termómetro de mínima, expuesto a todo tipo de radiación, ubicado horizontalmente y al aire libre. El método más preciso para conocer la temperatura del vegetal es hacer la medición con termocuplas de cobre-constantán que se insertan en los órganos, como las flores o frutos y se conectan por cable a un registrador. Sin embargo, es un método poco práctico para utilizar en las chacras.

Las temperaturas medidas con IA, son similares a las de un órgano vegetal próximo. La lectura de la temperatura en IA se realiza con termómetros de mínima o también con termorresistencias de platino (Pt) como sensores, con registro electrónico de datos. Es recomendable que los sensores tengan una superficie expuesta similar al bulbo de alcohol del termómetro de mínima. La altura de ubicación de los sensores dependerá de la altura de las plantas y de los órganos que interese evaluar.

Por otra parte, las temperaturas medidas en la casilla meteorológica o en abrigo, son las del aire y no reflejan las de los vegetales próximos. El aire es atérmico, es decir que sufre calentamiento o enfriamiento al tomar contacto con los cuerpos, como las plantas, el suelo, etc. Durante la noche, al producirse el enfriamiento, la temperatura del aire que rodea a las plantas es superior a las mismas.

2.7 Instrumental meteorológico de medición [1]

En el termómetro de mínima, el elemento sensible es el alcohol. Se emplea por su buen comportamiento y respuesta con temperaturas bajo 0 °C. Inmerso en el alcohol hay un cursor en forma de “T” que es arrastrado por el líquido, al descender la temperatura.

Este indicará la temperatura mínima registrada durante la helada. En el mercado nacional existen dos tipos de termómetros de mínima:

- El modelo más usado tiene el **bulbo en forma de “U”** y con la escala de temperaturas en el interior del vidrio protector.
- Otro modelo tiene **bulbo único** y la escala de temperatura está marcada sobre la superficie del vidrio.

Se recomienda utilizar una escala con resolución mínima de 0.2 °C. Quien observe el termómetro, al realizar la lectura debe apreciar la décima de °C.

El termómetro de mínima se instala en forma horizontal, apoyado en un soporte en dirección este-oeste geográfico. Dicha instalación debe realizarse a la altura de las primeras ramas fructíferas del cultivo que se protegerán de las heladas. En general, por norma de instalación del instrumental se ubica a 1.5 m de altura.

En la siguiente figura puede observarse como es la disposición que se emplea para un termómetro de mínima ubicado en un cuadro de cultivo:



Figura N° 8 **Termómetro de mínima en un monte de duraznero en floración**

Después de la lectura diaria de la temperatura mínima, se debe retirar el termómetro del campo ya que la radiación solar produce la evaporación del alcohol y por lo tanto posteriores errores en las lecturas. Este alcohol luego se condensa en gotitas en el extremo de la cámara de aire. Para unir estas gotas con el alcohol de la columna deberán efectuarse golpes bruscos empujando el termómetro desde la base. También es posible lograr ese efecto calentando lentamente el bulbo del termómetro en agua caliente.

Para el uso en defensa contra heladas, el error máximo admitido es de 0.2 °C. Para los termómetros nuevos, el Servicio Meteorológico Nacional realiza un control de calidad donde se indica el nivel del error.

Una vez al año, previo al periodo primaveral, es necesario realizar un control del error de los termómetros. Para esto se puede emplear el método del “hielo fundente”, que consiste en moler cubitos de hielos hechos con agua destilada y dentro de un trapo limpio. Luego se pasan a un termo de boca ancha y se agrega agua destilada para lograr una mezcla de agua-hielo que garantice una temperatura de 0 °C. Se introducen los bulbos de los termómetros en el termo y se realizan una serie de lecturas de la temperatura en cada instrumento. El error observado se indicará en una tela blanca engomada en el extremo de cada termómetro, debido a que este valor deberá sumarse o restarse a cada lectura. Este método de chequeo también es válido para el control de otros sensores como el de las termorresistencias de platino.

En cuanto a la posibilidad de reemplazar el instrumento por alguno de tipo electrónico, es importante que respondan de igual manera. Las termorresistencias de platino por ejemplo es un sensor adecuado para este uso debido a su buen comportamiento en temperaturas por debajo de los 0°C, con posibilidad de protegerlo además con algún otro material como vidrio o porcelana

También es posible obtener información mediante estaciones agrometeorológicas las cuales se instalan en las diferentes áreas de producción. Es un modo básico para obtener un conocimiento acabado del fenómeno de las heladas y sirve de referencia para los productores que están en el área de influencia.

El instrumental necesario para obtener información válida para el pronóstico y el control de heladas que se emplean en este tipo de estaciones son:

En la Casilla meteorología:

2.7.1 Termohigrógrafo con faja diaria [7]:

Es un instrumento de medición utilizado en meteorología para registrar tanto la temperatura como la humedad relativa.

Usualmente el sensor de temperatura es una placa bimetálica que por acción de la variación de la temperatura del aire genera una dilatación/contracción en las placas. Al ser de metales con diferente coeficiente de dilatación provocan un movimiento que es transmitido a un brazo. Este contiene en su extremo una pluma con tinta que traza en la banda de papel la temperatura.

El sensor de humedad relativa está formado por un atado de crines de caballo o similar que es muy sensible a la variación de la humedad atmosférica. De manera similar al caso de la temperatura se transmite el movimiento al brazo que con la pluma traza el papel.

Es un instrumento mecánico que posee en la base del tambor un mecanismo de relojería para que lo haga girar. Posee un engranaje doble calibrado para que dé un giro completo en un día o una semana.

La banda de papel lleva impresa una cuadrícula y unas escalas numeradas con los días o las horas del día y la humedad relativa en % y la temperatura en °C, o unidades según sistema de medida.

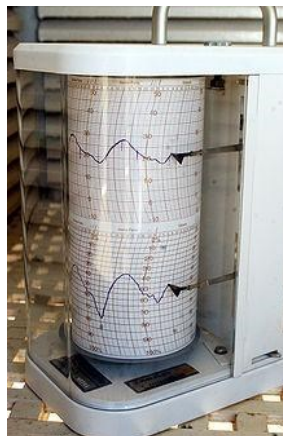


Figura N° 9 Imagen de un termohigrógrafo

2.7.2 Psicrómetro [6]:

Es un instrumento empleado en meteorología para determinar la humedad del aire atmosférico, se compone de dos termómetros ordinarios, uno de los cuales tiene la bola humedecida con agua, y por la comparación de las temperaturas indicadas en ellos se calcula el grado de humedad del aire. Estos aparatos constan de un termómetro de bulbo húmedo y un termómetro de bulbo seco. Su funcionamiento se basa en la comparación de las lecturas

de los dos termómetros. El húmedo marca menos temperatura que el otro debido a que el agua que empapa la muselina se evapora, pero para hacerlo necesita calor el cual obtiene del termómetro, por lo cual la temperatura baja. El agua evaporada es reemplazada por la que llega a través de la mecha, por lo que al termómetro le llega exactamente la misma cantidad de agua que se evapora. El descenso de temperatura que provoca la evaporación depende de la velocidad de evaporación, siendo que cuanto menos humedad relativa tenga el aire más rápida será la evaporación y en consecuencia será más baja la temperatura de este termómetro con respecto a la que marca el termómetro seco.

La relación entre la diferencia de temperatura que miden los dos termómetros y la humedad relativa no es directa ya que depende de la temperatura real del aire, y de la presión atmosférica. Con ayuda de unas tablas y conociendo las temperaturas de los dos se puede calcular fácilmente la humedad relativa.

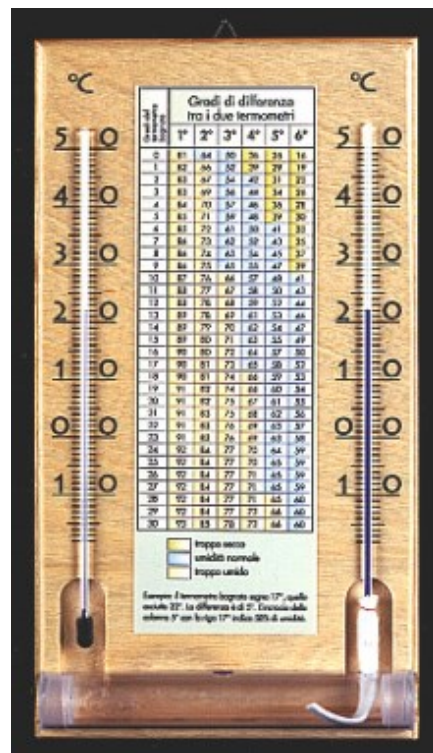


Figura N° 10 Psicrómetro

Empleando la siguiente tabla, es posible calcular la humedad relativa del aire con la información de los termómetros seco y húmedo. Por ejemplo, para el caso de una temperatura de 10 °C en el termómetro seco y de 7 °C en el termómetro húmedo, se encuentra una diferencia entre ellos de 3 °C. Entrando en ordenadas con 10 °C del termómetro seco y con 3 °C de diferencia de temperatura entre ellos se observa que la humedad relativa es del 66%.

HR del aire - % Temperatura termómetro seco y húmedo

Diferencia de temperatura termómetro seco y húmedo

	0,2	0,4	0,6	0,8	1,0	1,2	1,4	1,6	1,8	2,0	2,2	2,4	2,6	2,8	3,0	3,2	3,4	3,6	3,8	4,0	4,5	5,0	5,5	6,0
0	96	93	89	85	81	78	74	71	67	64	60	57	53	50	46	43	40	36	33	29	21	12	5	-
1	97	93	90	86	83	80	76	73	70	66	63	59	56	53	49	46	43	40	36	33	23	17	10	-
2	97	93	90	87	84	81	78	74	71	68	65	62	59	55	52	49	46	43	40	37	29	22	14	7
3	97	94	91	88	84	82	78	76	72	70	67	64	61	58	53	52	49	46	43	40	33	26	19	12
4	97	94	91	88	85	82	79	77	74	71	68	65	62	60	57	54	51	48	46	43	36	29	22	10
5	97	94	91	88	86	83	80	77	75	72	69	67	64	61	58	56	53	51	48	45	39	33	26	20
6	97	94	92	89	86	84	81	78	76	73	70	68	65	63	60	58	55	53	50	48	41	35	29	24
7	97	95	92	89	87	84	82	78	77	74	72	68	67	64	62	59	57	54	52	50	44	38	32	28
8	97	95	92	90	87	85	82	80	77	75	73	70	68	65	63	61	58	56	54	51	46	40	35	29
9	98	95	93	90	88	85	83	81	78	76	74	71	69	67	64	62	60	58	55	53	48	42	37	32
10	98	95	93	90	88	86	83	81	78	77	74	72	70	68	66	63	61	59	57	55	50	44	39	34
11	98	95	93	91	89	83	94	92	80	78	75	73	71	69	67	65	62	60	58	56	51	46	41	36
12	98	96	93	91	89	87	85	82	80	78	76	74	72	70	68	66	64	62	60	58	53	48	43	39
13	98	96	93	91	89	87	85	83	81	79	77	75	73	71	69	67	65	63	61	59	50	54	45	41
14	98	96	94	92	90	88	86	84	82	79	78	76	74	72	70	68	66	61	62	60	56	51	47	42
15	98	96	94	92	90	88	86	84	82	80	78	76	74	73	71	69	67	65	63	61	57	53	48	41

Figura N° 11 cálculo de la humedad relativa

También hay psicrómetros digitales los cuales detectan las condiciones ambientales como la temperatura y humedad relativa, valores de medición máxima y mínimo que se registran en una pantalla y se pueden pasar en tiempo real a la computadora para su evaluación posterior. Algunos también permiten medir la temperatura superficial de paredes u otros materiales a través de un sensor externo que poseen y que permite medir la temperatura a distancia; también detectar la temperatura del punto de rocío. La posibilidad de que la medición pueda ser detectada, registrada y archivada es de suma utilidad para todo tipo de uso.



Figura N° 12 **Psicrómetro digital**

En el campo de observación:

2.7.3 Mástil termométrico

Dicho mástil es quien permite sostener los termómetros de mínima a la altura necesaria para efectuar una correcta medición.

2.7.4 Anemógrafo:

El anemógrafo es un instrumento que mide la velocidad y la dirección del viento. Este aparato permite medir instantáneamente la velocidad del viento al tomar siempre un valor medio en intervalos de tiempo. Para realizar las mediciones debe ser colocado a 2 metros de altura.



Figura N° 13 Anemómetro de cazoletas.

2.8 Estudio microclimático de la zona cultivada:

Es necesario realizar un estudio del microclima para la determinación de las zonas o lugares fríos. Para entender el estudio, se tomará como ejemplo un cuadro frutal de alrededor de 2 hectáreas es decir 100 metros de ancho x 200 metros de largo, en el cual se instalarán tres termómetros de mínima a 1.5 metros de altura. Uno de ellos en medio del cuadro y los otros dos en los extremos de una diagonal. Durante la época primaveral se registran las temperaturas mínimas diarias durante al menos diez noches en condiciones de enfriamiento, es decir temperaturas cercanas a 0 °C, cielo despejado y sin viento.

Analizando los datos obtenidos, ocurrirá sistemáticamente que un sitio es más frío que el resto. Nuevamente, se realizará el mismo procedimiento en la segunda diagonal del mismo cuadro, detectando así un segundo sitio más frío que el resto. Cotejando ambos lugares, quedará definido el lugar más frío del cuadro.

En la época de defensa de heladas, generalmente los datos que se toman para una alarma o aviso son normalmente de un termómetro cercano a la casa u oficina. Considerando el estudio realizado, se deberán tener en cuenta las diferencias de temperatura entre este lugar de alarma con el lugar más frío detectado en el estudio microclimático.

2.9 Tablas de resistencia al frío [1][8]:

Tal como se puede observar en la siguiente tabla proporcionada por el instituto nacional de tecnología agropecuaria INTA, la sensibilidad al frío aumenta a medida que avanza el estado fenológico del cultivo. Esta sensibilidad tiene relación directa con el contenido de agua de los tejidos en los órganos del vegetal, debido a que el agua aumenta con la evolución del desarrollo vegetativo.

De esta manera, es posible encontrar distintos valores de temperaturas críticas para los distintos estados y especies de plantas, tal como se puede observar a continuación:

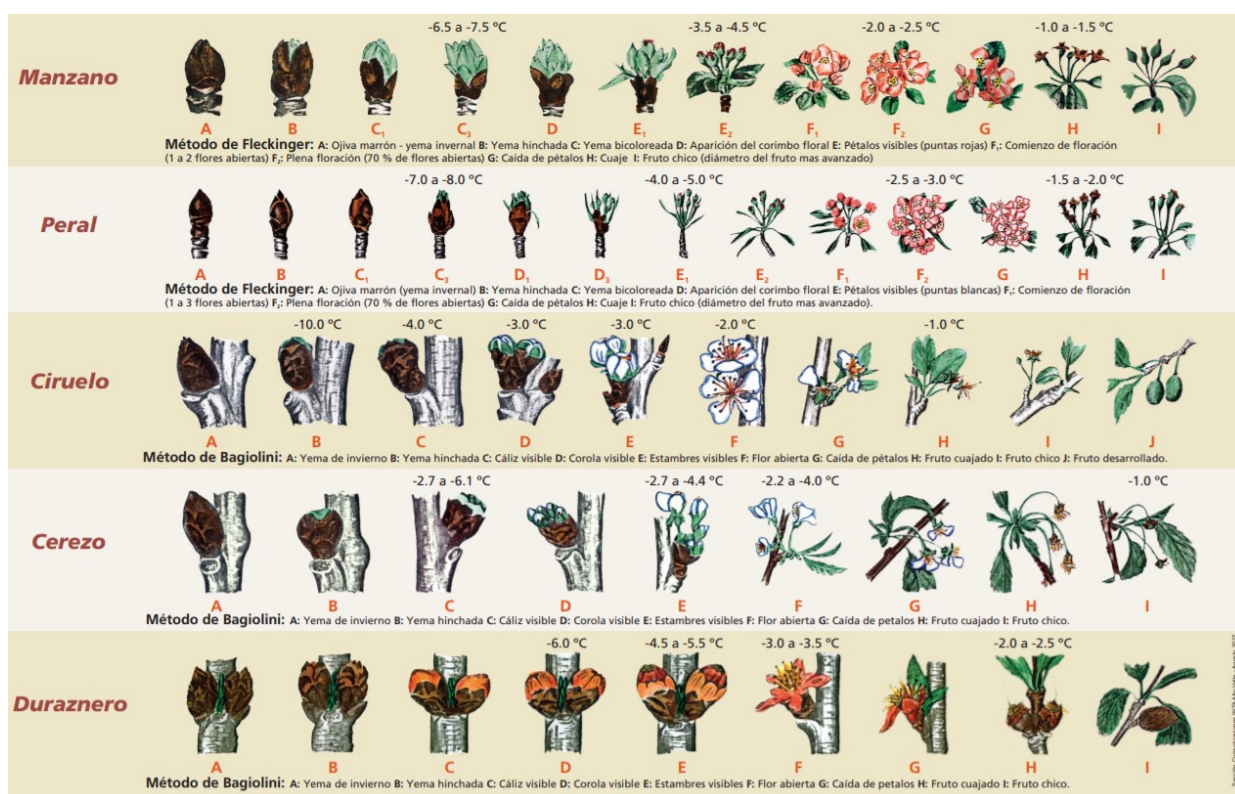


Figura N° 14 Estados fenológicos de los frutales y temperaturas críticas de daño

2.10 Factores de importancia en la resistencia

Los siguientes factores influyen en la resistencia al frío de las diferentes especies de plantas:

2.10.1 La velocidad del enfriamiento:

Cuando se produce un descenso térmico lento con caídas de temperatura en el orden de 0.5 °C a 1 °C por hora y cuando la temperatura llega a estar por debajo de 0 °C, en los tejidos vegetales se inicia la formación de hielo en los espacios intercelulares, luego se produce una difusión de agua desde la célula a estos espacios a través de la membrana celular por osmosis.

En la célula se produce concentración del jugo celular y en consecuencia un descenso del punto de congelamiento, además de un aumento de su resistencia al frío. Este proceso continuo en el tejido reduce la posibilidad de daño. Cuando el descenso térmico es brusco con temperaturas que descienden en el orden de 2 °C por hora o incluso más pronunciadas, se produce un congelamiento masivo del tejido vegetal. Generalmente, los tejidos y los órganos del árbol sufren daños de importancia que suelen ser irreversibles.

2.10.2 Las condiciones del tiempo:

Cuando los días previos a una helada son soleados, con altas temperaturas, sin viento y que son favorables las condiciones para estimular el crecimiento de las plantas, aumenta el tenor de agua en los tejidos y por ende la sensibilidad al daño por frío.

Cuando los días previos al fenómeno son de tiempo frío, el crecimiento vegetativo será escaso y permitirá una mejor adaptación de los tejidos a las condiciones ambientales. En la región del Alto Valle del Rio Negro, las típicas heladas primaverales están precedidas por uno o más días de viento frío y seco (con baja humedad), proveniente del sector suroeste. Además, al atardecer o durante la noche se reduce la velocidad del viento y con el arribo de la calma se inicia un enfriamiento rápido, de varios grados por hora de descenso térmico.

2.10.3 El estado nutricional del árbol:

Está comprobada la importancia del potasio y del fósforo (P) en los tejidos, para aumentar la resistencia al frío. El ion Potasio (K) produce un descenso del punto de congelamiento en el tejido. El Nitrógeno (N) suministrado al árbol a fin de temporada estimula el crecimiento y retarda el angostamiento de los tejidos haciéndolos más sensibles al daño por frío. Las modificaciones metabólicas de la célula y de la pared celular se hacen a través de enzimas y catalizadores. La síntesis de estos se produce gracias al Magnesio (Mg) y a los oligoelementos como el Boro (B), Cobre (Cu), Manganeso (Mn), Zinc (Zn) y el Molibdeno (Mb).

2.10.4 El fenómeno de “sobrefusión”:

La sobrefusión en el tejido vegetal se produce cuando la temperatura desciende por debajo del punto de congelamiento y no hay formación de hielo. El descenso térmico continúa hasta valores importantes por debajo de los 0 °C, pudiendo llegar incluso a -5 °C o más grados sin que se logre la formación de hielo. Luego de transcurrido un tiempo de este proceso de enfriamiento, se desencadena rápidamente la formación de hielo con el consecuente aumento de la temperatura debido a la liberación de “calor de fusión”.

En el siguiente gráfico es posible ver como la temperatura continúa descendiendo lentamente. La sobrefusión protege a las plantas del daño por frío. El fenómeno se produce y se estimula la formación cuando los descensos de la temperatura son lentos.

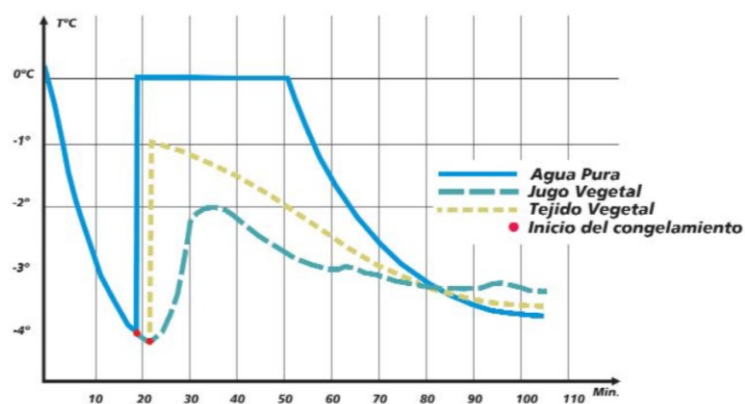


Figura N° 15 Proceso de sobrefusión

2.11 Defensa activa contra heladas

La defensa activa contra heladas es una práctica que se aplica a un cuadro frutal durante las horas nocturnas de helada, con el fin de aumentar la temperatura de las plantas e impedir que se alcance el umbral de resistencia al frío del vegetal.

2.11.1 Método de riego por aspersión [1][5]:

El uso de una aspersión con agua para luchar contra las heladas, aprovecha la liberación de calor que se produce al congelarse el agua. Una capa de agua sobre una hoja que se está enfriando, al congelarse libera energía que es aprovechada por la hoja y su temperatura no descenderá de 0 °C siempre y cuando se mantenga una aspersión constante durante el periodo de temperaturas bajas, hasta que el hielo se haya fundido por acción del sol.

Los motores del sistema de riego, pueden automatizar su accionamiento y detención, ajustando la duración de la defensa contra helada y reducir los costos

La condición básica para el correcto funcionamiento de este método es que, durante el tiempo de uso en defensa, haya permanentemente agua líquida congelándose sobre las plantas y órganos que se protegen. El calor liberado por el agua al congelarse, se transmite por conducción a través del hielo que recubre a las yemas, flores o frutos de la planta.

El operador para iniciar la defensa debe basarse, en un primer momento, en las temperaturas de resistencia al frío del estado fenológico predominante en el cultivo. Las temperaturas serán las obtenidas con termómetro húmedo.

Cuando el descenso térmico es brusco, con caídas del orden de 2 °C/hora o aún más pronunciadas, el equipo se deberá poner en marcha con 1 °C por encima de la temperatura crítica de daño. Los descensos térmicos bruscos se producen, generalmente, cuando calma el viento después de un período de vientos persistentes, fríos y secos, del sector suroeste, de uno o varios días.

Sistema de monitoreo para heladas

CAPÍTULO 2

Sensores y Módulos empleados

3 Introducción al capítulo:

En el presente capítulo se realizará una breve descripción de los módulos y sensores empleados para llevar a cabo el proyecto. Se detallarán sus especificaciones técnicas y funcionalidades, para tener un conocimiento previo antes de empezar a explicar el desarrollo del proyecto.

3.1 Desarrollo:

Tal como se mencionó en el capítulo 1, para determinar la presencia de helada es importante contar con información de temperatura, humedad relativa, la presencia o no de viento, la dirección del viento y la presión atmosférica. Esta información, le va a permitir al agricultor establecer en función al estado de su plantación, si se encuentra o no bajo la presencia de helada en la zona de cultivo.

En cuanto a la energización de los circuitos, se procederá a diseñar un sistema de alimentación a través de energía solar, pensando en la posible dificultad de no contar con una fuente de energía eléctrica cerca de la zona de sensado.

Para poder realizar este trabajo, se procedió a seleccionar los siguientes sensores y módulos:

- Sensor de Temperatura y Humedad Modelo DHT 22
- Sensor de Presión Atmosférica y Temperatura BMP280
- Modulo Acelerómetro ADXL345 3 Ejes Digital
- Placa NodeMCU WiFi ESP8266 Versión 3
- Placa NodeMCU WiFi ESP32
- Fuente de alimentación Protoboard 5v 3.3v Mb102
- Panel solar 5 Volt
- Batería de Ion Litio modelo 18650
- Modulo cargador de batería de Litio TP4056
- Regulador de baja caída (LDO) MCP1700 3302E
- Módulo relé de 5v opto acoplado de 10 Amp y 2 Canales

- Módulo Step Up
- Modulo pantalla OLED 128x64 monocroma
- Transistor 2N2222
- Diodo 1N4001

3.2 Sensor de Temperatura y Humedad Modelo DHT 22 [9]

El DHT22 es un sensor del tipo capacitivo que releva la humedad y la temperatura de manera digital. Posee una alta fidelidad y una excelente estabilidad a largo plazo gracias a la tecnología de recolección de datos que posee.

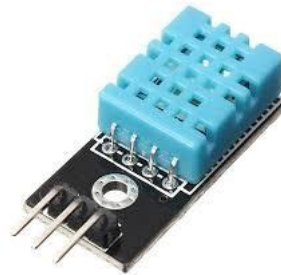


Figura N° 16 **Sensor DHT22**

El sensor incluye un componente capacitivo para la toma de datos de humedad y un dispositivo de gran calidad (el datasheet no cita el tipo de tecnología) para la medición de la temperatura. Estos se conectan a un microcontrolador de 8 bits (Memoria interna al módulo o E2PROM). Este sensor posee una excelente calidad, muy buena respuesta en tiempo, fuerte capacidad para evitar las interferencias electromagnéticas y un costo relativamente bajo.

3.3 Conexión del sensor y especificaciones del fabricante:

3.3.1 Especificaciones técnicas:

Posee un bajo consumo energético y una excelente estabilidad a largo plazo. Además, emplea un único bus para transmitir los datos recabados.

El diagrama de conexión de sus pines es el siguiente:

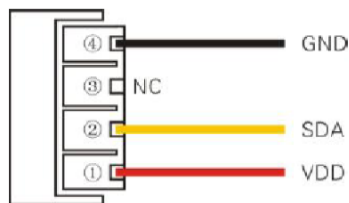


Figura N° 17 Grafica de los pines de conexión

Pin	Nombre	Descripción
1	VDD	Power (3.3V – 5.5V)
2	SDA	Serial data
3	NC	Empty
4	GND	Ground

Tabla N° 1 Pines de conexión del sensor a la placa

El sensor de temperatura y humedad debe ser alimentado con un voltaje de entre 3.3 Volt y 5.5 Volt, pero se recomienda alimentarlo con 5 Volt cuando el cable de conexión entre el sensor y el módulo de adquisición de datos es de una distancia considerable (30 metros como máximo).

Parámetro	Condición	Mínimo	Típico	Máximo	Unidad
Resolución			0.1		%HR
Rango		0		99.9	%HR
Exactitud	25°C		± 2		%HR
Repetitividad			± 0.3		%HR
Respuesta			<5		Seg
Variación	Típico		0.5		%HR/año

Tabla N° 2 Parámetros de humedad relativa del DHT22

Parámetro	Condición	Mínimo	Típico	Máximo	Unidad
Resolución			0.1		° C

Rango		-40		+80	° C
Exactitud			± 0.5	± 1	° C
Repetitividad			± 0.2		° C
Respuesta			< 10		Seg
Variación			± 0.3		° C /año

Tabla N° 3 Parámetros de temperatura del DHT22

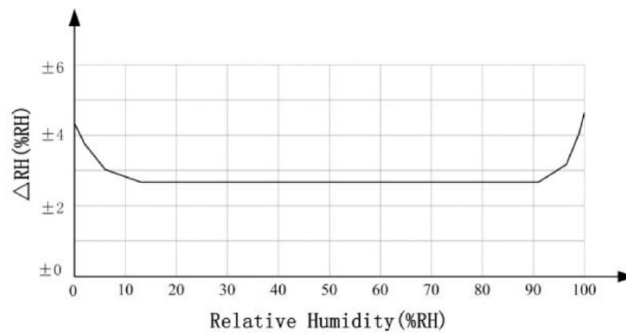


Figura N° 18 Error cometido con la humedad relativa

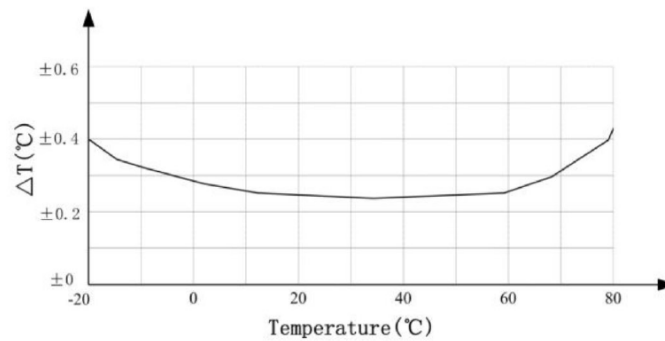


Figura N° 19 Error cometido en la temperatura

3.3.2 Especificaciones eléctricas

En la siguiente tabla se presentan las principales características del sensor:

Parameter	Condition	min	typ	max	Unit
Voltage		3.3	5	5.5	V
Power consumption ^[4]	Dormancy	10	15		μ A
	Measuring		500		μ A
	Average		300		μ A
Low level output voltage	I_{OL} ^[5]	0		300	mV
High output voltage	$R_p < 25 \text{ k}\Omega$	90%		100%	VDD
Low input voltage	Decline	0		30%	VDD
Input High Voltage	Rise	70%		100%	VDD
R_{pu} ^[6]	VDD = 5V VIN = VSS	30	45	60	$\text{k}\Omega$
Output current	turn on		8		mA
	turn off	10	20		μ A
Sampling period		2			S

Tabla N° 4 Características eléctricas

3.3.3 Dimensiones:

El dispositivo es de tamaño pequeño, bajo consumo energético y con una alta capacidad de transmisión de la señal debido a que permite alcanzar distancias de unos 30 metros. Estas cualidades hacen que el DHT22 sea la mejor opción para elegir en una gran cantidad de aplicaciones con elevadas exigencias.

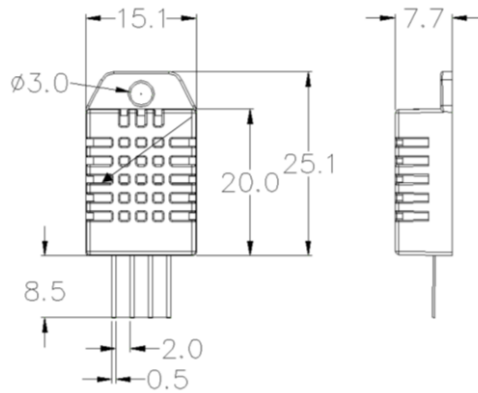


Figura N° 20 Dimensiones del sensor DHT22 en milímetros

3.3.4 Condiciones de trabajo y almacenamiento

Cuando el sensor trabaja fuera de las condiciones normales de trabajo puede suceder que su vida útil se vea reducida notablemente y que además de esto sea necesario seguir un proceso de recalibración del sensor. Los elementos deben estar almacenados en un lugar seco y fresco. Las siguientes condiciones deben evitarse:

- Ambiente Salino.
- Zonas en donde se almacene ácido clorhídrico o gases oxidantes tales como el dióxido de azufre.

También debe evitarse que el sensor sea expuesto a sustancias químicas debido a que puede producir una disminución de la sensibilidad de este, en cuyo caso los cambios serán irreversibles. En cuanto a la influencia de la temperatura y la luz solar, es importante que sea colocado en un lugar alejado de fuentes de calor que puedan alterar la lectura del elemento. La exposición prolongada a la luz solar o radiación ultravioleta puede degradar el funcionamiento del elemento. Por otro lado, se recomienda un entorno de almacenamiento en donde:

- La temperatura se encuentre entre 10 °C a 40 °C
- La Humedad Relativa sea menor a 60 % HR.

3.3.5 Procedimientos de calibración:

Cuando el sensor es sometido a condiciones de trabajo extremas, posiblemente puede desajustarse. Si esto ocurriera, debemos realizar un proceso de calibración para ajustar su funcionamiento. En primer lugar, se debe mantener el sensor durante 2 horas bajo las siguientes condiciones:

- Humedad < 10 % HR.
- Temperatura 45 °C

Luego del proceso anterior, se debe mantener el sensor durante 5 horas bajo las siguientes condiciones:

- Humedad > 70 % HR.
- Temperatura entre 20 °C y 30 °C

3.4 Introducción al módulo acelerómetro ADXL345 [15]:

El acelerómetro ADXL345 de Analog Devices es un dispositivo pensado para aplicaciones móviles. Se puede conectar fácilmente a través de su interfaz SPI (3 o 4 hilos) e I2C. El sensor es adecuado para medir aceleración estática y dinámica. Posee una resolución de 4 mg / LSB permitiendo detectar cambios de inclinación de hasta 1°.

El ADXL345 se suministra en un tamaño pequeño, delgado, de 3 mm × 5 mm × 1 mm.



Figura N° 21 **Acelerómetro ADXL345**

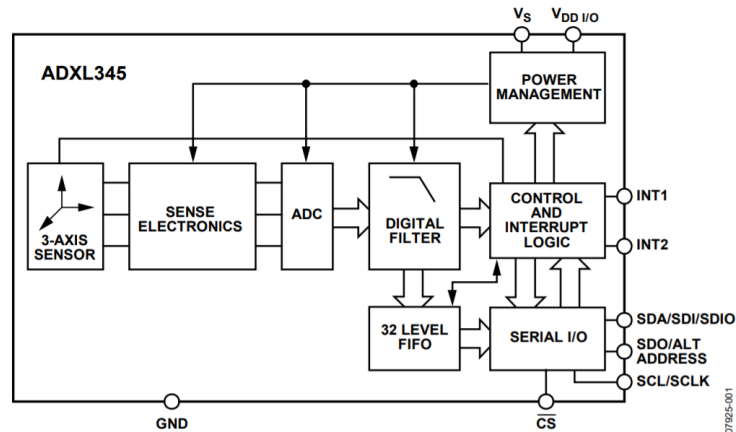


Figura N° 22 Diagrama en bloques de la arquitectura interna

3.4.1 Especificaciones Importantes:

- Acelerómetro con interfaz digital I2C y SPI
- Rango de medición seleccionable +/- 2, 4, 8 y 16 g
- El módulo o tarjeta de interconexión incluye circuito regulador de voltaje
- Pines configurables para generar interrupciones
- Voltaje de operación de a 3.6 V
- Detección de picos de aceleración dobles y sencillos (como interfaz de usuario)
- Detección de caída libre
- Puede conectarse con el bus I2C y SPI.

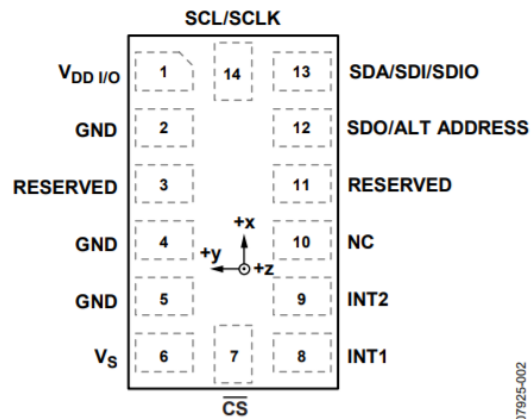


Figura N° 23 Configuración de pines

Pin N°	Asignación	Función
1	Vdd	Tensión de alimentación
2,4,5	GND	Pines de conexión a tierra
3	Reserved	Este pin debe conectarse a Vs
6	Vs	Tensión de suministro
7	CS	Chip Select
8	INT1	Interruptor 1 de salida
9	INT2	Interruptor 2 de salida
10	NC	No conectado internamente
11	Reserved	Este pin debe ser conectado a GND
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I2 C Address Select (I2 C).
13	SDA/SDI/SDIO	Serial Data (I2 C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire)
14	SCL/SCLK	Serial Communications Clock

Tabla N° 5 Descripción de la función de cada pin

3.5 Placa NodeMCU WiFi ESP8266 Versión 3 [16][17]

La placa NodeMCU es al igual que Arduino una placa de desarrollo libre a nivel de software y de hardware. La principal ventaja que tiene este microcontrolador es que consta del microcontrolador ESP-12 o ESP-12E que incorpora como se ha mencionado anteriormente el módulo ESP8266 que permite la conexión WiFi, para elaborar proyectos de IoT con sistemas inalámbricos.

Es importante destacar que NodeMCU no es un microcontrolador, sino que es una placa de desarrollo, cuyo objetivo es facilitar la programación de los componentes que la forman. El microcontrolador de esta unidad es el Tensilica L 106 y se encuentra integrado en el chip ESP8266. En la siguiente figura se muestra el esquema de la estructura de la placa de desarrollo:

Sistema de monitoreo para heladas

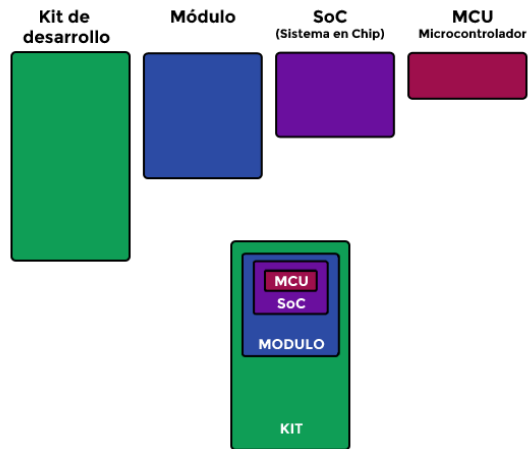


Figura N° 24 Esquema genérico

La placa de desarrollo NodeMCU será la que se encargue de las entradas, salidas y cálculos que requiera el programa en cada momento. Los principales fabricantes de este tipo de placas son Amica (distribuidor oficial), DOIT y Loli/Wemos:

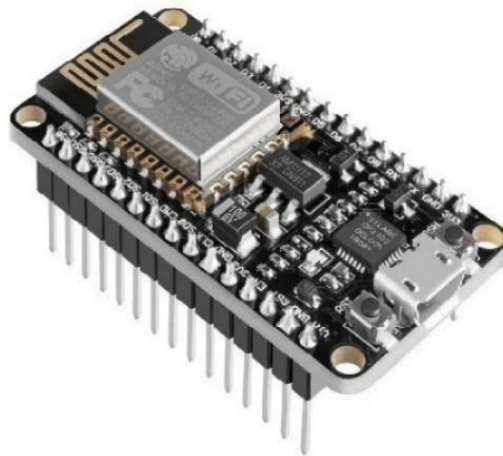


Figura N° 25 Tarjeta NodeMCU WiFi ESP8266 V.3

En la siguiente tabla se muestran las principales características del módulo NodeMCU:

Procesador	Tensilica L106 32-bit
Tensión de operación	3,3 V
Frecuencia	80 Mhz – 160 Mhz
Pines E/S Analógicos	1 con resolución de 10 bits
Memoria Flash	4 Mb
Memoria RAM	128 KB
Conexión	WiFi
Soporte / Norma	802.11 b/ g/ n
Longitud	57mm
Ancho	30 mm

Tabla N° 6 Características de NodeMCU WiFi ESP8266 V.3

Existen varias versiones de NodeMCU ya que es una placa de hardware libre y cualquier fabricante puede crear su modelo. Lo que hay que tener en cuenta es que todos los módulos se basan en el ESP8266 y en el ESP-12 o ESP-12E dependiendo de la versión con la que el usuario trabaje, este último componente es el que incorpora la memoria flash que el módulo ESP8266 no dispone. La principal diferencia entre las distintas versiones son los números de pines y las dimensiones que tienen.

3.5.1 Características generales de NodeMCU:

- Fácil acceso a los pines
- Conversor Serie-USB para poder programar y alimentar a través de este
- Pines de alimentación para sensores y componentes
- Leds para indicar el estado del dispositivo
- Botón de reset

Hay 3 versiones de placa NodeMCU y tienen ciertas diferencias entre sí:

- **Primera generación V 0.9/ V1:** emplea los módulos ESP-12 y ESP8266 con una memoria flash de 4MB. Tamaño de 47mmx31mm. El inconveniente de esta placa es el tamaño ya que ocupa todo el espacio de una protoboard y no deja hueco libre para conectar los pines.

- **Segunda generación V1.0/ V2:** en este caso emplea el módulo ESP-12E y como ventaja respecto a la generación anterior es que reduce el tamaño de la tarjeta, dejando una fila de pines disponible a cada lado, siendo posible usar la protoboard de forma más cómoda para realizar conexiones.
- **Tercera generación V1.0/ V3:** Esta versión no es una nueva especificación oficial de NodeMCU, sino una versión creada por el fabricante Lolin. Como mejora, incorpora un pin para la alimentación Vin a 5V, un GND adicional y un puerto USB más robusto. La desventaja está en el tamaño, que vuelve a ser más grande y no se puede acoplar a una protoboard.

3.5.2 Esquema de entradas y salidas de NodeMCU:

En la siguiente figura, se puede observar el esquema de los diferentes pines a los cuales tenemos acceso en la placa NodeMCU de la versión V3, la cual será empleada en este proyecto:

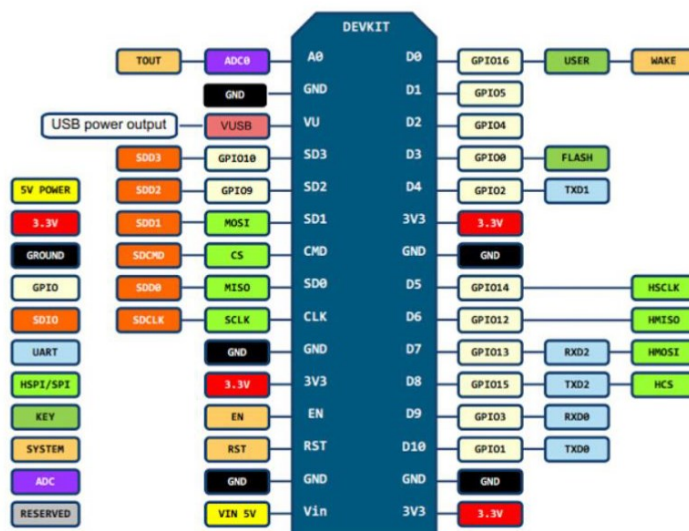


Figura N° 26 Pinout NodeMCU V3 [18]

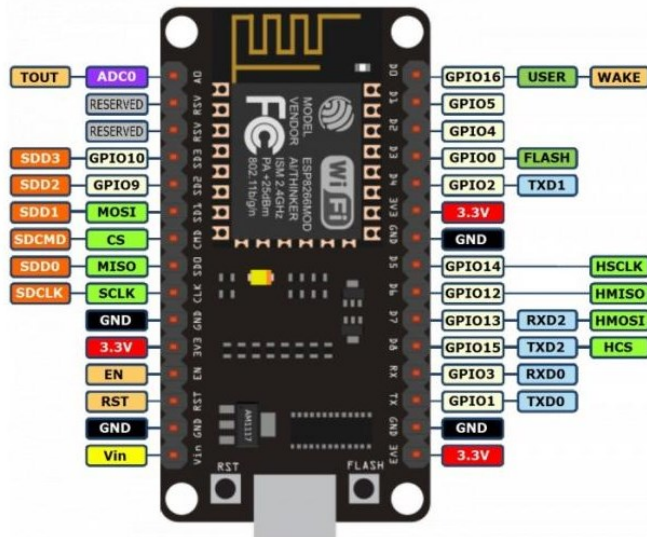


Figura N° 27 Pinout NodeMCU visto desde la tarjeta NodeMCU [19]

La placa NodeMCU dispone de trece pines digitales numerados desde D0 al D12. También se encuentra el nombre de los pines que conecta con el módulo ESP8266, apareciendo con la nomenclatura GPIO xx. El módulo ESP8266 no dispone de memoria flash, por lo que se deben dejar pines libres para esa conexión. Este es el motivo por el cual no es recomendable utilizar los pines GPIO del 6 al 1. El GPIO 09 y GPIO 10 corresponden respectivamente a los pines D11 y D12 hay que evitar el uso de estos pines, al igual que los pines D09 y D10 que corresponden a los pines Rx y Tx, usados para recibir y transmitir un programa o para la comunicación entre el NodeMCU y el ordenador a través del puerto serie. De este modo solo quedarían disponible nueve pines digitales de entrada y salida, es decir desde el D0 al D8.

Hay que tener en cuenta que, al ser entradas y salidas digitales, los sensores y actuadores que se conecten a estos deberían ser del tipo digital ya que estas E/S solo están diseñadas para recibir y enviar valores binarios, es decir sólo distinguen un nivel alto o bajo.

En cuanto a las entradas analógicas, la NodeMCU solo cuenta con un pin analógico denominado A0. Este tiene un rango de valores que va desde 0 a 3.3V. Como la placa solo puede trabajar con valores digitales, contiene un conversor analógico a digital con una resolución de 10 bits.

3.5.3 Alimentación:

La placa NodeMCU dispone de pines de alimentación, estos pines tienen principalmente dos funciones, la primera es alimentar la placa y la segunda alimentar sensores y componentes de salida. El voltaje de operación del dispositivo es de 3.3V y es por esto que solo se podrán alimentar componentes a 3V. Al alimentar el NodeMCU con el puerto USB con 5V, dispone de un regulador que saca 3,3V para alimentar la placa y sacar a los pines para alimentar los componentes externos. Hay dos posibles formas de suministrar energía y alimentar el microcontrolador:

- Mediante una fuente de alimentación como de 3,3V y 5V, usando la línea de potencia de 3,3V. Para estos se puede emplear la fuente de alimentación MB V2.
- Mediante una computadora, gracias a que el NodeMCU dispone de un conversor USB a serie.

3.6 Tarjeta NodeMCU WiFi ESP32 [28][29][30]

La placa de desarrollo NodeMCU-32 es una herramienta muy potente para el diseño de prototipos con IoT (Internet de las cosas). Integra en una placa el SoM ESP-WROOM-32 que tiene como base al SoC ESP32, el conversor USB-serial CP2102 necesario para programar por USB el ESP32, reguladores de voltaje y leds indicadores.

La ESP32 es la evolución del ESP8266 mejorando sus capacidades de comunicación y procesamiento computacional. A nivel de conectividad permite utilizar diversos protocolos de comunicación inalámbrica como WiFi, Bluetooth y BLE.

En cuanto a procesamiento, cuenta con una CPU 32-bit de dos núcleos de hasta 240 Mhz que se pueden controlar independientemente. Además, incluye internamente una gran cantidad de periféricos para la conexión con sensores táctiles capacitivos, sensor de efecto Hall, amplificadores de bajo ruido, interfaz para tarjeta SD, Ethernet, SPI de alta velocidad, UART, I2S e I2C. Aplicado en Mini Servidores Web, Procesamiento digital, Webcams, Cámara IP, Robótica móvil, Domótica y más.

NodeMCU-32 está diseñada especialmente para poder trabajar montada en protoboard. Puede alimentarse directamente por el puerto micro-USB o utilizando una fuente externa de 5V o 3V ya que posee regulador de voltaje en placa, recomendándose utilizar una fuente de 5VDC/1A y colocar un capacitor de 100uF en paralelo con la fuente de alimentación para filtrar los picos de corriente.

Los pines de entradas/salidas (GPIO) trabajan a 3.3V por lo que para conexión a sistemas de 5V es necesario utilizar convertidores de nivel.

La plataforma ESP32 permite el desarrollo de aplicaciones en diferentes lenguajes de programación, frameworks, librerías y recursos diversos. Los más comunes a elegir son:

- Arduino (en lenguaje C++)
- MicroPython, LUA
- Esp-idf(Espressif IoT Development Framework) desarrollado por el fabricante del chip
- Simba Embedded Programming Platform(en lenguaje Python)
- RTOS's (como Zephyr Project, Mongoose OS, NuttX RTOS)
- Javascript (Espruino, Duktape, Mongoose JS)
- Basic.

La familia incluye los chips:

- ESP32-D0WDQ6
- ESP32-D0WD
- ESP32-D2WD
- ESP32-S0WD
- Y el SiP (System in Package) ESP32-PICO-D4.

Los ESP32 poseen un alto nivel de integración. En su pequeño encapsulado se incluyen:

- interruptores de antena
- balun de RF
- amplificador de potencia
- amplificador de recepción de bajo ruido
- filtros y módulos de administración de energía

Además, logra un consumo de energía muy bajo a través de funciones de ahorro de energía que incluyen sincronización de reloj y múltiples modos de operación. Esto hace que sea una muy buena herramienta para proyectos energizados con baterías o aplicaciones IoT.

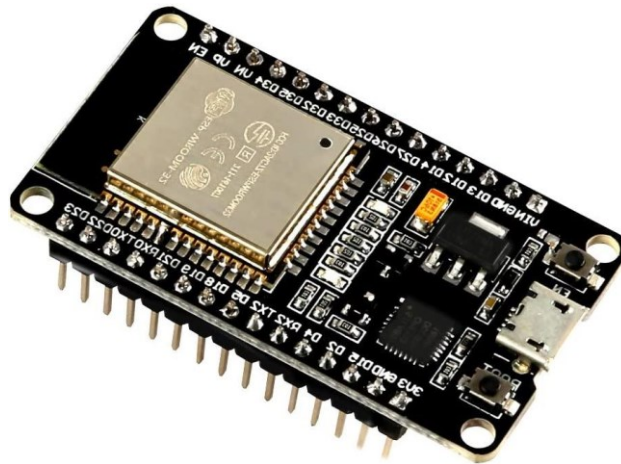


Figura N° 28 Imagen de tarjeta NodeMCU ESP32

3.6.1 Comparación entre ESP32 y ESP8266

Podemos ver al ESP32 cómo un ESP8266 con algunas funcionalidades más, ya que cuenta con:

- un núcleo adicional
- WiFi más rápida
- mayor número de pines de entrada/salida
- compatibilidad con Bluetooth 4.2 y Bluetooth de baja energía (low energy).

Además, ESP32 viene con pines sensibles al tacto que se pueden usar para “despertar” al ESP32 del modo deep sleep (sueño profundo) y un sensor de efecto Hall incorporado.

En la tabla siguiente podemos encontrar un resumen de las principales diferencias técnicas entre el ESP8266 y el ESP32.

Características	ESP8266	ESP32
Microprocesador	Xtensa single-core 32bit L106	Xtensa dual-core 32bit LX6 con 600 DMIPS
WI-FI (802. 11 b/g/n)	HT20	HT40
Bluetooth	No posee	Bluetooth 4.2 y BLE
Frecuencia de operación	80Mhz típico	160Mhz típico
SRAM	No posee	448Kb
Flash	No posee	520Kb
GPIO	17	34
PWM (hardware)	No posee	No posee
PWM (software)	8 canales	16
SPI	2	4
I2C	1	2
I2S	2	2
UART	2	2
ADC	10 bits de resolución	12 bits de resolución
CAN	No	Si
Interfaz MAC ethernet	No	Si
Sensro de tacto	No	Si
Sensor de temperatura	No	Si en algunas versiones
Sensor de efecto hall	No	Si
Temperatura de trabajo	-40 °C a 125 °C	-40 °C a 125 °C

Tabla N° 7 **Comparación entre especificaciones de NodeMCU ESP32 vs NodeMCU ESP12**

3.6.2 Especificaciones del ESP32

En la siguiente imagen podemos observar todos los bloques funcionales que conforman un SoC ESP32.

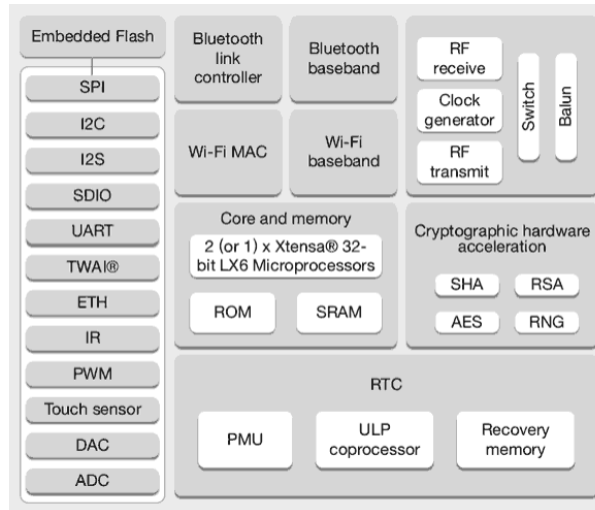


Figura N° 29 **Diagrama en bloques del ESP32**

Vemos que el chip cuenta con conectividad WiFi, siendo compatible con 802.11 b/g/n en la banda de los 2.4GHz, alcanzando velocidades de hasta 150 Mbits/s. También incluye comunicación Bluetooth compatible con Bluetooth v4.2 y Bluetooth Low Energy (BLE).

Luego vemos el bloque de radio, el cual está estrechamente ligado a los módulos de comunicación inalámbricos, siendo el que realmente transmite y recibe la información. Es decir que este bloque es el que se encarga de tomar los datos digitales provenientes de los módulos Wifi y Bluetooth, y los convierte en señales electromagnéticas que viajan por el aire para comunicarse con un teléfono móvil o router. También realiza la operación inversa, es decir que puede traducir las ondas electromagnéticas generadas por otros dispositivos en datos digitales que los módulos Wifi y Bluetooth son capaces de interpretar.

Como ya mencionamos, el ESP32 cuenta con dos microprocesadores de bajo consumo Tensilica Xtensa de 32 bits LX6, pero además cuenta con un co-procesador de ultra bajo consumo que es utilizado para realizar conversiones analógico-digital y otras operaciones mientras el dispositivo se encuentra funcionando en el modo de bajo consumo deep sleep. De esta forma, se consigue un consumo muy bajo por parte del SoC. Estos procesadores ofrecen grandes ventajas típicas de un procesador digital de señales:

- Frecuencia de operación: 240 MHz (ejecuta instrucciones 15 veces más rápido que una placa Arduino UNO)
- Permite realizar operaciones con números reales (números con coma) de forma muy eficiente.
- Permite realizar multiplicaciones de números grandes de forma instantánea.

En cuanto a las memorias del ESP32, podemos encontrar dos tipos de memorias que se clasifican como memorias internas y memorias externas. Las memorias internas son aquellas que se encuentran ya incluidas en el SoC, y las memorias externas son aquellas que se pueden adicionar para expandir la capacidad del sistema.

Muchas placas de desarrollo basadas en ESP32 añaden memorias externas para lograr un sistema con mejores prestaciones. Las memorias internas del ESP32 son:

- **Memoria ROM (448 KB):** esta memoria es de solo escritura, es decir que no se puede reprogramar. En esta memoria se almacenan los códigos que manejan la pila Bluetooth, el control de la capa física de la Wifi, algunas rutinas de propósito general y el cargador de arranque (bootloader) para iniciar el código de la memoria externa.
- **Memoria SRAM interna (520 KB):** esta memoria es utilizada por el procesador para almacenar tanto datos como instrucciones. Su ventaja es que, para el procesador, es mucho más fácil acceder a esta que a la SRAM externa.
- **RTC SRAM (16 KB):** esta memoria es utilizada por el co-procesador cuando el dispositivo opera en modo deep sleep.
- **Efuse (1 Kbit):** 256 bits de esta memoria son utilizados por el propio sistema y los 768 bits restantes están reservados para otras aplicaciones.
- **Flash empotrada (Embedded flash):** esta memoria es donde se almacena el código de nuestra aplicación. La cantidad de memoria varía en dependencia del chip utilizado:
 - 0 MiB (chips ESP32-D0WDQ6, ESP32-D0WD y ESP32-S0WD)
 - 2 MiB (chip ESP32-D2WD)

- 4 MiB (módulo SiP ESP32-PICO-D4)

Para los ESP32 que no poseen memoria empotrada o simplemente cuando la memoria es insuficiente para la aplicación que se desea diseñar, es posible adicionar más memoria de forma externa. Se pueden agregar hasta 16 MiB de memoria flash externa y también admite hasta 8 MiB de memoria SRAM externa.

3.6.3 Esquema de entradas y salidas de NodeMCU ESP32:

En la siguiente figura, se puede observar el esquema de los diferentes pines a los cuales tenemos acceso en la placa:

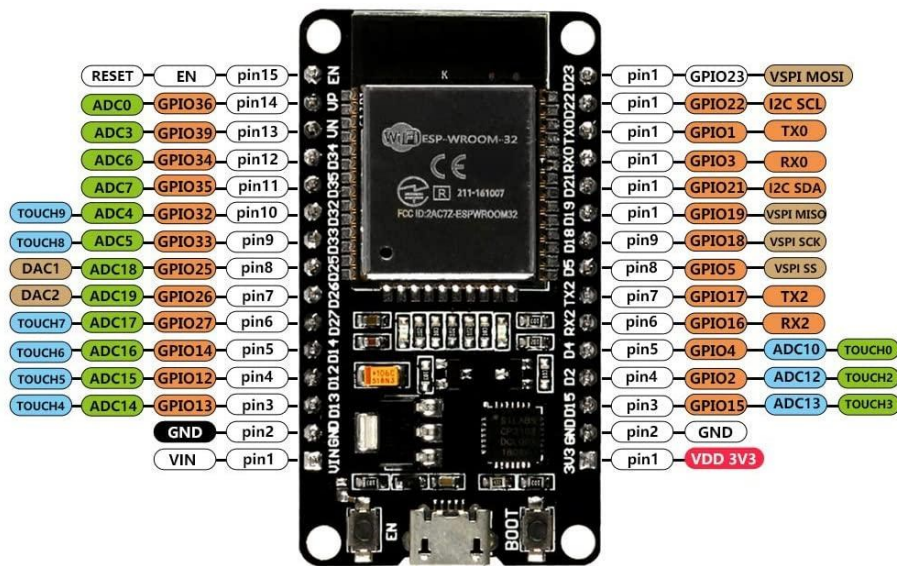


Figura N° 30 Pinout de la ESP32

3.6.4 Modo Deep Sleep

El ESP32 se puede configurar para diferentes modos de energía, clasificados en:

- Active mode
- Modem Sleep mode
- Light Sleep mode
- Deep Sleep mode

- Hibernation mode

Estos cinco modos difieren entre sí, en qué partes se dejan activas y cuales se desactivan. En la siguiente figura podemos observar las diferencias:

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Figura N° 31 Modos de energía y sus diferencias

También podemos apreciar las diferencias en cuanto al consumo de energía para los diferentes modos:

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA Normal speed 80 MHz: 20 mA ~ 25 mA Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μ A
	ULP sensor-monitored pattern	100 μ A @1% duty
	RTC timer + RTC memory	10 μ A
Hibernation	RTC timer only	5 μ A
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μ A

Figura N° 32 Consumos de energía de los diferentes modos de energía

En la siguiente figura podemos ver el consumo de energía en el modo activo:

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11b, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11g, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Figura N° 33 **Consumos del modo activo**

Comparando la información, vemos que al activar el modo sueño profundo se reduce el consumo de energía, permitiendo que las baterías que energizan el circuito, duren más.

Tener el ESP32 en modo de suspensión profunda, implica cortar con las actividades que consumen más energía mientras está en funcionamiento, pero dejando actividad suficiente para despertar el procesador cuando se cumpla alguna condición de interés.

Cuando el modo sueño profundo está activado, no se ejecutan actividades de CPU ni de WIFI, pero el coprocesador de energía ultra baja, es decir el ULP, aún puede activarse. Estando activo el modo, la memoria RTC también permanecerá encendida, permitiendo que pueda escribirse código para manejar el procesador ULP, permitiendo acceder a dispositivos periféricos, temporizadores internos y sensores internos.

Durante el sueño profundo, el coprocesador ULP puede utilizar los pines RTC GPIO y los pines táctiles TOUCH del ESP32. Estos pines podemos identificarlos en la siguiente imagen, estando los pines RTC GPIO encuadrados en naranja y los TOUCH en rosado:

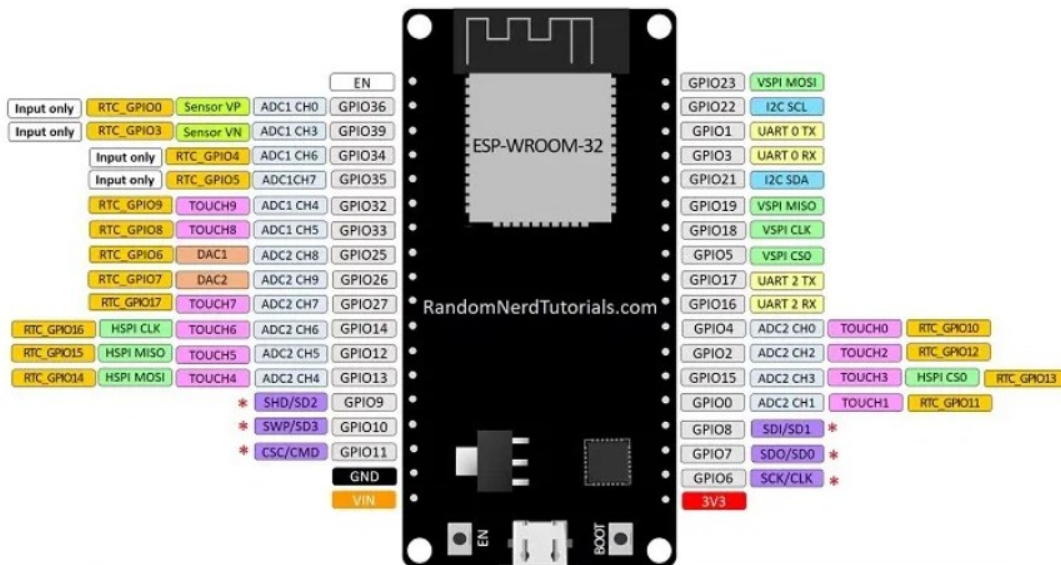


Figura N° 34 **Pinout del ESP32 con los puertos RTC diferenciados**

Al momento de querer activar el ESP 32 luego de estar en modo sueño profundo, tenemos varias formas de realizarlo:

- Usar un temporizador, fijando periodos de tiempos predefinidos
- Usar los pines táctiles
- Usar una interrupción externa
- Usar el coprocesador ULP para despertarse

3.7 Fuente de alimentación Protoboard 5v 3.3v Mb102

Es un módulo que permite alimentar una protoboard MB-102 mediante plug a un transformador de pared hasta 12V. Proporciona dos salidas independientes que son seleccionables mediante jumpers y permiten suministrar 5V o 3.3V. Cuenta además con salida de 5V por conector USB.

Es una fuente práctica, que simplifica la alimentación de la protoboard y circuitos electrónicos especialmente los de carácter digital, ideal con cualquier versión de Arduino y microcontroladores PIC. Cada lado de la fuente tiene un jumper de encendido / apagado y de selección de voltaje.



Figura N° 35 Fuente de alimentación MB-102

3.7.1 Especificaciones:

- Compatibilidad: MB-102 de 400 y 830 puntos
- Voltaje de entrada: 6.5V a 12V mediante el Jack
- Voltaje de salida 1: 3.3V o 5V (seleccionable)
- Voltaje de salida 2: 3.3V o 5V (seleccionable)

- Salida de tensión USB: 5V
- Corriente máxima de salida: 700 mA
- LED indicador de encendido
- Botón de encendido / apagado
- Dimensiones 5.6 cm × 3 cm × 2.5 cm

○

3.8 Módulo relé opto acoplado [34] [35]

Un relé es un interruptor mecánico operado eléctricamente que puede ser encendido o apagado mediante niveles de tensión bajos. En este proyecto se empleará un módulo relé que posee dos canales y que dispone de optoacopladores. De esta forma, podremos controlar el encendido y apagado de cualquier aparato que se conecte a una fuente de alimentación eléctrica externa siempre que se encuentre dentro de las especificaciones de corriente y tensión que manejan los relés.

Cada relé funciona como si fuera un interruptor, el cual será activado o desactivado mediante una entrada de datos. En la siguiente figura podemos ver como es el módulo:

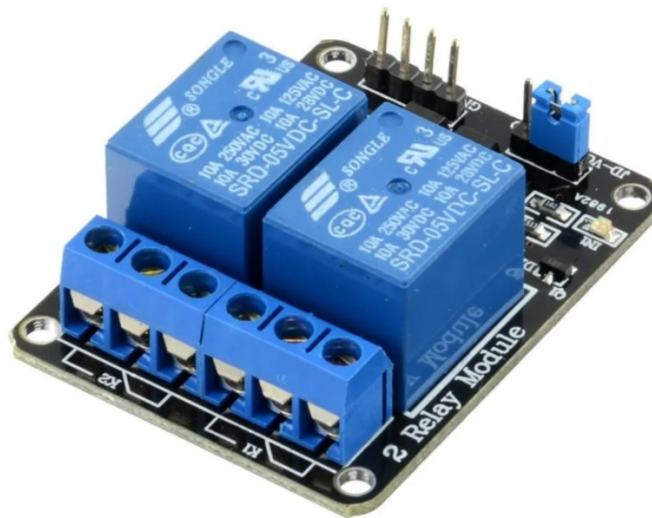


Figura N° 36 **Módulo Relé de dos canales**

De la siguiente figura, podemos ver en función al código de los relay que podemos activarlos con 5 Volts, generando una corriente nominal de aproximadamente 70 mAmp, generando un consumo de aproximadamente 360 mWatts.

Sistema de monitoreo para heladas

Bobina Sensibilidad	Bobina Voltaje Código	Nominal Voltaje (VDC)	Nominal Actual (mamá)	Bobina Resistencia ($\Omega \pm 10\%$)	Poder Consumo (W)	Pull-In Voltaje (VDC)	Abandonar Voltaje (VDC)	Máximo permitido Voltaje (VDC)
S R D (Elevado Sensibilidad)	03	03	1 2 0	25	abt. 0,36 W	75% máx.	10 minutos.	1 2 0%
	05	05	71,4	70				
	06	06	60	1 0 0				
	09	09	40	2 2 5				
	12	12	30	4 0 0				
	24	24	15	1 6 0 0				
S R D (Estándar)	48	48	7.5	6 4 0 0	abt. 0,45 W	75% máx.	10 minutos.	1 1 0%
	03	03	1 5 0	20				
	05	05	89,3	55				
	06	06	75	80				
	09	09	50	1 8 0				
	12	12	37,5	3 2 0				
	24	24	18,7	1 2 8 0				
	48	48	10	4 5 0 0	abt. 0,51 W			

Figura N° 37 Características del relay

Este modelo de relé nos permite manejar 10 Amper de corriente con 28 VDC o con 240 VAC según lo que especifica el fabricante en la siguiente figura.

Artículo	Escribe S R D	
	FORMULARIO C	FORMULARIO A
Capacidad de contacto	7A 28VDC	10A 28VDC
Carga resistiva ($\cos \Phi = 1$)	10A 125VAC 7A 240VAC	10A 240VAC
Carga inductiva (por $\Phi = 0,4 L / R = 7 \text{ ms}$)	3A 120VAC 3A 28VDC	5A 120VAC 5A 28VDC
Max. Voltaje permitido	250VAC / 110VDC	250VAC / 110VDC
Max. Fuerza de potencia permitida	800VAC / 240W	1200VA / 300W
Material de contacto	A g C d O	A g C d O

Figura N° 38 Especificaciones del relay

En la siguiente imagen podemos ver cuáles son los pines de entrada y salida que tiene este módulo. Podemos distinguir los pines de salida que manejan alto voltaje y corriente, encontrando NC (normal cerrado), NA (normal abierto) y COM (pin común) de cada relé. Por otro lado, los pines de la derecha manejan baja tensión, donde encontraremos los pines de alimentación de 5V y GND, como así también los pines de entrada lógica que controlan a cada relé.

Sistema de monitoreo para heladas

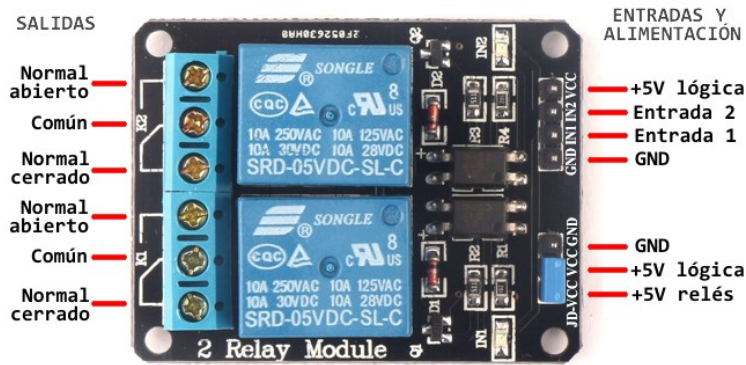


Figura N° 39 Pines de entrada y salida del módulo relay de dos canales

El fabricante nos proporciona un esquema eléctrico para un solo relé, el cual se va repitiendo a medida que se agregan más canales. En la siguiente imagen podemos apreciar el circuito, donde Vcc es el pin de alimentación positiva de 5V para la parte de la lógica de entrada conformada por la resistencia R1, la sección de emisión de luz de U1 (pines 1 y 2 del optoacoplador), el led indicador N1 y IN0 que sería la entrada de control. También podemos observar el pin JD-Vcc, por donde ingresan 5V para alimentar el circuito de accionamiento del relé, el cual se compone por el fototransistor de U1 (pines 3 y 4 del optoacoplador), la resistencia R2, el transistor Q1 que maneja la corriente de la bobina, el diodo D1 para eliminar la contracorriente del bobinado y la bobina propia del relé.

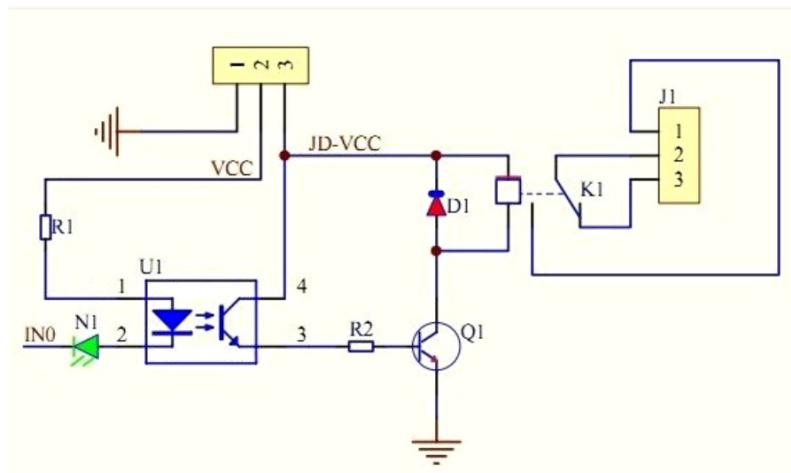


Figura N° 40 Esquema eléctrico de módulo relay de un canal

El optoacoplador que emplea cada uno de los canales es el PC817, cuyo integrado podemos observar a continuación:

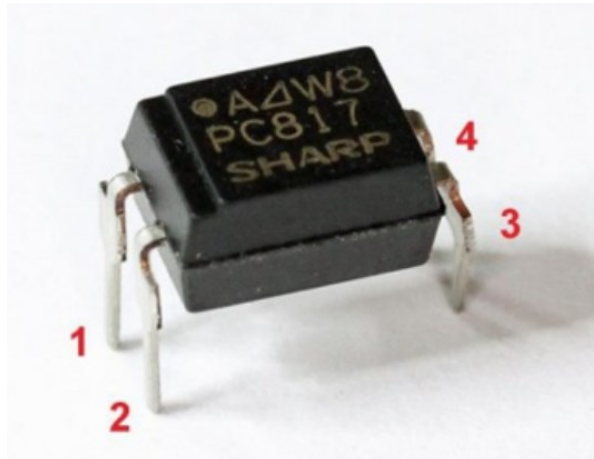


Figura N° 41 Optoacoplador empleado en el módulo

El optoacoplador posee una entrada de dos (pines 1 y 2) que conectan a un led y una salida de dos pines que conectan a un fototransistor (pines 3 y 4). El encendido del led pone en conducción el fototransistor, lo cual produce el cierre del circuito entre su colector (pin 4) y su emisor (pin 3). La característica más importante del funcionamiento es que no hay contacto eléctrico entre entrada y salida. El único contacto es la luz emitida por el led, lo que implica una aislación muy elevada entre entrada y salida que ronda en el orden de los 5000V, protegiendo las entradas ante cualquier riesgo de alto voltaje producido en el área de salida.

En las siguientes dos figuras podemos ver en el circuito eléctrico cómo funciona cuando el relé se encuentra desactivado o activado:

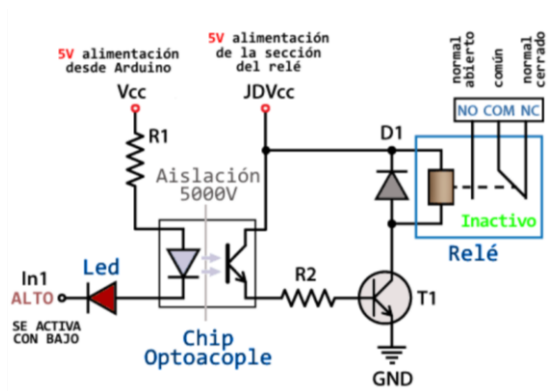


Figura N° 42 Relé desactivado

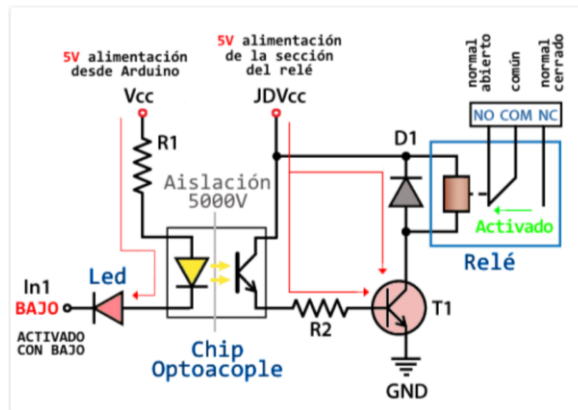


Figura N° 43 Relé activado

Si nos concentramos en las entradas del módulo, las cual se señala en la siguiente figura, podemos distinguir dos conjuntos de pines, uno de 4 y otro de 3:

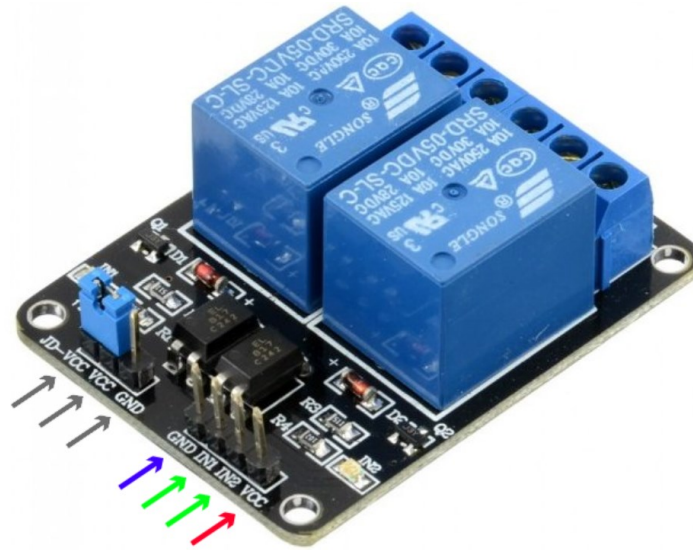


Figura N° 44 Perspectiva de entrada del módulo

El conjunto señalado con flechas de color se conforma por las conexiones de VCC y GND para la alimentación de la lógica digital del módulo, conformada esencialmente por los leds de los optoacopladores de entrada, la entrada 1 (IN1) y entrada 2 (IN2) para controlar los relés 1 y 2, respectivamente, aplicando un nivel lógico bajo.

Por otro lado, el conjunto de pines señalado con flechas de color gris se conforma por los pines JD-VCC, VCC y GND. Como mencionamos anteriormente, el pin JD-VCC es la

alimentación para los electroimanes de los relés, y se lo alimenta generalmente con una fuente de 5V separada debido al consumo de corriente que implica su funcionamiento. Para hacerlo se retira el puente (celeste en la imagen) que une la alimentación de los leds de los optoacopladores con la alimentación de las bobinas de los relés. Usando dos fuentes independientes se obtiene la máxima separación eléctrica entre las entradas de control y la línea controlada de 220V, u otro voltaje superior a 5V, en las salidas.

3.9 Panel solar:

Un panel solar es un dispositivo que aprovecha la energía del sol para generar electricidad. Los paneles fotovoltaicos, generan electricidad a partir de la radiación solar que incide sobre las células fotovoltaicas del panel.

Los paneles solares fotovoltaicos constan de celdas, llamadas células fotovoltaicas, que convierten la radiación solar en electricidad. Se genera electricidad debido al efecto fotovoltaico que provoca la energía solar (fotones), generando cargas positivas y negativas en dos semiconductores próximos de distinto tipo, lo que genera un campo eléctrico que producirá corriente eléctrica. Los materiales más utilizados para fabricar estas células son el arseniuro de galio (GaAs) y el silicio (Si), de menor costo económico.

Las células de silicio son las más comunes y más utilizadas. Su rendimiento depende de la estructura tridimensional interna que tengan estas láminas de silicio. Según esta estructura podemos clasificarlas del siguiente modo

- Células de silicio monocristalino: constituido por un solo cristal de grandes dimensiones que es cortado en finas láminas, generalmente de azul uniforme. Son las más avanzadas, su costo de fabricación es superior y proporcionan un mayor rendimiento bajo determinadas condiciones.
- Células de silicio policristalino: están constituidas por varios cristales, tienen un color azul no uniforme, aunque las últimas técnicas de fabricación ya otorgan de mayor uniformidad al aspecto de la célula.

- Células de silicio amorfo: no está formada por cristales. Es la más barata pero también la que menores rendimientos ofrece, y tienen la particularidad de que pueden producir electricidad (en poca cantidad) aunque no estén expuestas directamente a la radiación solar de manera perpendicular.

Para este proyecto, se procederá a utilizar paneles solares de 5 volt que proporcionan una máxima corriente de 220 mA ideales, lo cual implica que son de una potencia de 1 W. Sus dimensiones son 100x70x3.5 mm y pueden ser empleados en ambiente exterior.



Figura N° 45 **Panel solar 5volt 220mAmp**

El número de paneles a utilizar dependerá de la cantidad de baterías que necesitemos para cada circuito y del tiempo de carga que deseemos; podemos conectar varios paneles en paralelo para reducir el tiempo de carga, obteniendo el doble de corriente, pero manteniendo la tensión.

3.10 Batería Ion Litio Modelo 18650

La batería 18650, es una pila recargable muy utilizada cuya tensión es de 3,7 voltios. Hay varios tipos de pilas 18650 y se diferencian por el amperaje que pueden entregar y por sus capacidades máximas. Esta capacidad viene expresada en miliamperios hora, y nos indica la autonomía que tiene cada pila, siendo de mayor duración cuanto más mAh tengan. Las capacidades más comunes para estos modelos son entre 2000 mAh a 4000 mAh. Cabe destacar que son relativamente livianas, no exigen mucho mantenimiento y tienen una vida útil de 500 a 1000 ciclos.



Figura N° 46 **Batería recargable modelo 18650**

Las dimensiones de esta pila están cifradas en su nombre de modelo 18650. Los dos primeros números 18 indican su diámetro 18 mm, mientras que el número 650 indica que su longitud es 65 mm.

Es importante indicar que tienen algunas desventajas, como por ejemplo que son muy sensibles y con frecuencia se dañan en caso de sufrir una sobrecarga o calentamiento. Para evitar estos efectos indeseables, suelen equiparse con un circuito electrónico diseñado para proteger las baterías de una sobrecarga o calentamiento durante el proceso de carga. Es peligroso usar baterías sin este circuito de protección ya que el recalentamiento las puede hacer explotar. Si las baterías vienen protegidas, suele indicarse esto mediante una carátula que dice "Protected", "With protective PCB", "Protection Circuit", etc.

Para el presente proyecto usaremos baterías 18650 de 2600mAh y las cargaremos mediante energía solar. En cuanto a la cantidad de baterías a utilizar, dependerá del consumo final del circuito que deseamos alimentar y del tiempo de funcionamiento.

3.11 Módulo cargador de batería TP4056

El módulo cargador consta de un chip TP4056, el cual se encargará de gestionar la carga de una batería. Es decir, su trabajo consta en adecuar la entrada de energía para el estándar de 1A de la mayoría de baterías de litio que se usan en la industria electrónica, contando además con la capacidad de controlar la temperatura.

Su chip es del tipo SOP y se complementa con un bajo número de componentes externos haciendo que el módulo sea ideal para aplicaciones portátiles.

El TP4056 emplea retroalimentación térmica, regulando la corriente de carga para limitar la temperatura durante el funcionamiento a alta potencia o temperatura ambiente alta. El voltaje de carga se fija en 4,2 V y la corriente de carga se puede programar externamente con una sola resistencia. El ciclo de carga terminará automáticamente cuando la corriente de carga cae a una décima parte del valor programado después de que se alcanza el voltaje de flotación final.

Otras características que incluye, es el monitoreo de corriente, bloqueo de baja tensión, recarga automática y dos leds de estado para indicar que la carga concluyó carga y la presencia de un voltaje de entrada.



Figura N° 47 **Modulo cargador de batería de litio
18650**

De la figura anterior, podemos distinguir las siguientes partes fundamentales del módulo:

- Puerto mini USB, el cual permite alimentar el módulo a través de él.
- Bornes + y – de entrada, que permiten conectar por ejemplo las salidas de un panel solar y alimentar el módulo a través de él.

- Led rojo que indicará que se está cargando la batería y led azul que indicará cuando la batería está completamente cargada.
- Bornes B + y B - , que son los bornes de salida mediante los cuales se conectara la batería que deseamos cargar.
- Bornes OUT+ y OUT- que se emplean para alimentar el circuito que deseamos energizar, permitiendo entregarle energía mientras se carga la batería.

3.11.1 Características del chip TP4056

- Corriente de carga programable hasta 1000 mA
- No requiere diodo de bloqueo
- Cargador lineal completo para baterías de celda única
- Corriente constante / voltaje constante
- Carga baterías de iones de litio de celda única directamente desde el puerto USB
- Voltaje de carga preestablecido de 4.2V con 1.5% Precisión
- Recarga automática
- Dos pines de salida de estado de carga
- C / 10 Terminación de carga
- Umbral de carga lenta de 2,9 V
- Arranque suave para limitar la corriente de irrupción

3.11.2 Características eléctricas [31]

En la siguiente imagen podemos ver las características eléctricas que especifica el fabricante del chip.

The ● denotes specifications which apply over the full operating temperature range, otherwise specifications are at $T_A=25^{\circ}\text{C}$, $V_{CC}=5\text{V}$, unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
V_{CC}	Input Supply Voltage		● 4.0	5	8.0	V
I_{CC}	Input Supply Current	Charge Mode, $R_{PROG} = 1.2\text{k}$	●	150	500	μA
		Standby Mode (Charge Terminated)	●	55	100	μA
		Shutdown Mode (R_{PROG} Not Connected, $V_{CC} < V_{BAT}$, or $V_{CC} < V_{UV}$)	●	55	100	μA
V_{FLOAL}	Regulated Output (Float) Voltage	$0^{\circ}\text{C} \leq T_A \leq 85^{\circ}\text{C}$, $I_{BAT}=40\text{mA}$	4.137	4.2	4.263	V
I_{BAT}	BAT Pin Current Text condition: $V_{BAT}=4.0\text{V}$	$R_{PROG} = 2.4\text{k}$, Current Mode	● 450	500	550	mA
		$R_{PROG} = 1.2\text{k}$, Current Mode	● 950	1000	1050	mA
		Standby Mode, $V_{BAT} = 4.2\text{V}$	● 0	-2.5	-6	μA
I_{TRIKL}	Trickle Charge Current	$V_{BAT} < V_{TRIKL}$, $R_{PROG}=1.2\text{K}$	● 120	130	140	mA
V_{TRIKL}	Trickle Charge Threshold Voltage	$R_{PROG}=1.2\text{K}$, V_{BAT} Rising	2.8	2.9	3.0	V
V_{TRHYS}	Trickle Charge Hysteresis Voltage	$R_{PROG}=1.2\text{K}$	60	80	100	mV
T_{LIM}	Junction Temperature in Constant Temperature Mode			145		$^{\circ}\text{C}$

Figura N° 48 Especificaciones brindadas por el fabricante

De estas especificaciones, nos concentramos en la máxima tensión que tolera el módulo, la máxima corriente que tolera el módulo y la máxima corriente que entrega. En función a estas especificaciones y al consumo de nuestro sistema, determinaremos cuantos paneles, celdas, módulos y configuraciones se utilizarán.

3.11.3 Ciclo de carga

En la siguiente imagen podemos observar el ciclo completo de carga que aplica este módulo de carga para las baterías 18650.

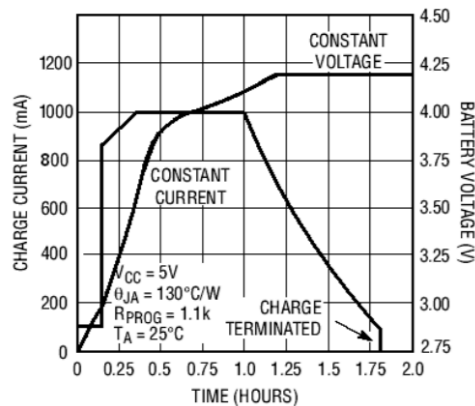


Figura N° 49 Ciclo de carga obtenido de Datasheet

3.11.4 Circuito integrado de protección [32]

El chip DW01-P que se encuentra en el módulo, es un circuito integrado diseñado para proteger la batería de Ion Litio de daños o degradación de su vida útil debido a sobrecarga, sobre descarga o sobre corriente.

Su pequeño tamaño y los pocos componentes necesarios lo hacen ideal para ser integrado en el espacio limitado del paquete de baterías. El voltaje preciso de detección de sobrecarga de ± 50 mV garantiza una carga segura. La corriente de reserva muy baja drena poca corriente de la celda mientras está almacenada.

3.12 Regulador de baja caída MCP1700-3302E [33]

Un regulador de baja caída, es un regulador de voltaje especial que mantiene una salida estable a un nivel de voltaje de entrada solo ligeramente superior al voltaje de salida. Los reguladores de voltaje estándar requieren un voltaje de entrada que sea mayor en más de aproximadamente 5 volts que el voltaje de salida. El regulador de baja caída puede mantener una regulación de voltaje estricta, incluso cuando la entrada sea más alta solo en 1 volt o menos.

El MCP1700 es una familia de reguladores de voltaje de baja caída (LDO) CMOS que pueden entregar hasta 250 mA de corriente mientras consumen sólo 1.6 μ A de corriente de reposo (típico). El rango operativo de entrada se especifica de 2,3 V a 6,0 V, por lo que es

una opción ideal para algunas aplicaciones alimentadas por batería de celda primaria, así como aplicaciones de iones de litio de celda única.

Es capaz de entregar 250 mA con solo 178 mV de diferencial de voltaje de entrada a salida. La tolerancia de voltaje de salida del MCP1700 es típicamente $\pm 0.4\%$ a $+25\text{ }^\circ\text{C}$ y $\pm 3\%$ máximo sobre el rango de temperatura de la unión de $-40\text{ }^\circ\text{C}$ a $+125\text{ }^\circ\text{C}$.

Los voltajes de salida disponibles para el MCP1700 varían de 1.2V a 5.0V. La salida LDO es estable cuando se usa solo una capacitancia de salida de $1\text{ }\mu\text{F}$. Los condensadores electrolíticos de cerámica, tantalito o aluminio se pueden utilizar para la entrada y la salida. El límite de sobre corriente y el apagado por sobre temperatura brindan una solución sólida para cualquier aplicación.



Figura N° 50 **Regulador de baja caída MCP1700**

3.12.1 Características

- Calificado AEC-Q100 y apto para PPAP
- Corriente de reposo típica de $1,6\text{ }\mu\text{A}$
- Rango de voltaje operativo de entrada: $2,3\text{ V}$ a $6,0\text{ V}$
- Rango de voltaje de salida: $1,2\text{ V}$ a $5,0\text{ V}$
- Corriente de salida de 250 mA para salida Voltajes mayores o iguales $2,5\text{ V}$
- Corriente de salida de 200 mA para salida Voltajes menores $2,5\text{ V}$
- Voltaje de baja caída (LDO) 78 mV típico @ 250 mA para $\text{VOUT} = 2,8\text{ V}$
- Tolerancia de voltaje de salida típica del $0,4\%$
- Opciones de voltaje de salida estándar: $1,2\text{ V}$, $1,8\text{ V}$, $2,5\text{ V}$, $2,8\text{ V}$, $2,9\text{ V}$, $3,0\text{ V}$, $3,3\text{ V}$, $5,0\text{ V}$
- Estable con condensador de salida de cerámica de $1,0\text{ }\mu\text{F}$

- Protección contra cortocircuitos
- Protección contra sobrecalentamiento

3.12.2 Aplicaciones:

- Dispositivos a batería
- Circuitos de alarma alimentados por batería
- Detectores de humo
- Detectores de CO₂
- Buscapersonas y teléfonos móviles
- Paquetes de baterías inteligentes
- Referencia de voltaje de corriente de reposo bajo
- PDA
- Cámaras digitales
- Energía del microcontrolador

3.13 Transistor 2N2222 [35]:

El 2N2222 es un transistor de silicio del tipo NPN de baja potencia, diseñado para aplicaciones de amplificación lineal y conmutación. Es de uso general debido a que es bueno amplificando pequeñas corrientes y tensiones pequeñas o medianas, además de poder trabajar con frecuencias medianamente altas. Este transistor en muchos casos puede ser reemplazado también por el BC548 cuando la corriente de salida necesaria es menor de 0,1A.

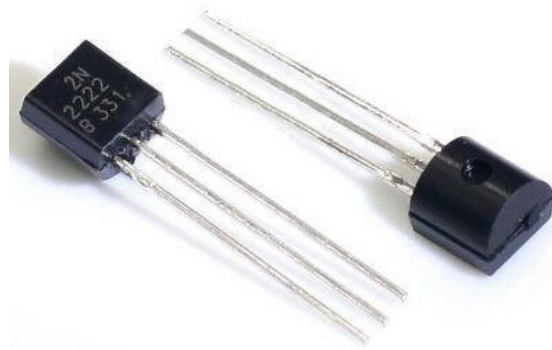


Figura N° 51 **Transistor 2N2222**

Los 2N2222 tienen una capacidad de corriente de colector significativa, que alcanza hasta 800 mA, característica que permite usarlos también en muchas aplicaciones de control de media potencia como por ejemplo drivers para pequeños motores, relés y también tiras de leds cortas.

3.13.1 Especificaciones del 2N2222

Las especificaciones más relevantes brindadas en la hoja de datos por el fabricante, son las siguientes:

- Voltaje colector emisor en corte 60 V (V_{ce0})
- Corriente de colector constante 800mA (I_c)
- Potencia total disipada 500mW (P_d)
- Ganancia o hfe 35 mínima
- Frecuencia de trabajo 250 MHz (F_t)
- Estructura NPN

A continuación, podemos ver el pinout del transistor:

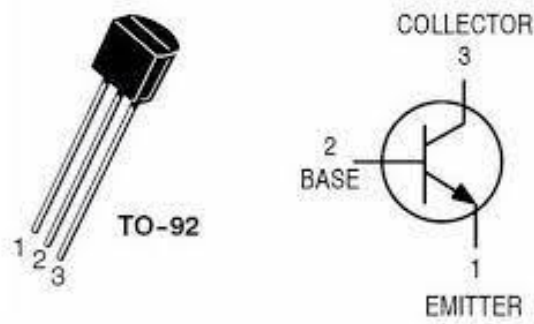


Figura N° 52 Pinout de transistor 2N2222

3.14 Modulo DC-DC Step Up 0.9-5V a 5V [36]

Un convertidor de tensión DC-DC Step-Up tiene como función entregar en su salida una tensión DC superior a la tensión de entrada, frente a variaciones en la tensión entrante o frente a variaciones de carga. En este caso particular, el módulo puede elevar tensiones que

irán desde los 0,9V hasta 4,5V, entregando siempre en su salida una tensión constante de 5V y una corriente máxima que dependerá de la tensión entrante.



Figura N° 53 **Modulo DC-DC Step Up**

Si bien no se encuentra una hoja de datos clara respecto a este módulo, los vendedores especifican las siguientes características de corriente máxima de salida en función a la tensión de entrada aplicada:

- Tensión de entrada: 0.9V - Corriente máxima de salida: 7mA
- Tensión de entrada: 1-1.5V - Corriente máxima de salida: 40-100mA
- Tensión de entrada: 1.5-2V - Corriente máxima de salida: 100-150 mA
- Tensión de entrada: 2-3V - Corriente máxima de salida: 150-380 mA
- Tensión de entrada: >3V - Corriente máxima de salida: 380-480 mA

3.15 Diodo 1N4001 [37]

El 1N4001 es un diodo rectificador de propósito general, que tiene como función principal conducir corriente en una sola dirección. Está fabricado de un material semiconductor con una unión N-P. Dentro de sus especificaciones más importantes, el fabricante brinda las siguientes:

- Voltaje inverso pico repetitivo: 50 Volts
- Voltaje de funcionamiento a 1 A: 1.1 Volts
- Corriente de rectificación promedio: 1 Amp
- Corriente pico no repetitiva en medio ciclo de onda senoidal (8.3 ms): 30 Amp
- Disipación de potencia a 25 °C : 3 Watts

- Rango de temperatura de la unión: -65 a 150 °C
- Capacitancia total a 4 Volts, 1 MHz: 15 pF
- Tipo de encapsulado: DO-41
- Peso: 0.30 gramos (Aproximadamente)



Figura N° 54 **Diodo 1N4001**

3.16 Pantalla OLED

El display Oled 1.3" I2C SH1106 posee una resolución de 128x64 píxeles, permitiendo controlar cada píxel individualmente y mostrar tanto texto como gráficos.

Tienen un consumo muy bajo (cerca a los 20 mA) debido a que solo se enciende el píxel necesario y no requieren de backlight (retroiluminación) como los displays LCD.

El módulo posee interfaz de comunicación de tipo I2C (en otros modelos también vienen con SPI), lo cual facilita la comunicación y las conexiones necesarias.

Su diseño permite emplearlas para trabajar a 5V directamente ya que posee un regulador de voltaje en placa, pero también puede trabajar con sistemas de 3.3V o 5V sin necesidad de convertidores. Estas pantallas, se destacan por su gran contraste, bajo consumo de energía y buena calidad de imagen



Figura N° 55 **Display Oled 128x64 Px I2C**

3.16.1 Especificaciones

- Voltaje de Operación: 3V – 5.5V DC
- Driver: SH1106
- Interfaz: I2C (dirección I2C: 0x3C)
- Resolución: 128*64 píxeles
- Monocromo: píxeles blancos (fondo negro)
- Ángulo de visión: 160°
- Área visible (display): 30*15 mm
- Consumo de energía ultra bajo: 0.04W (cuando están encendidos todos los píxeles)
- Temperatura de trabajo: -30°C ~ 70°C
- Dimensiones: 35*33*4 mm
- Peso: 7 gramos

Sistema de monitoreo para heladas

CAPÍTULO 3

Página Web

4 Introducción al capítulo:

En el presente capítulo, nos encontramos con el desarrollo desde cero de la página web empleada para la telemetría del sistema. Observaremos el paso a paso de la creación, como también así el diseño de la base de datos que almacenará la información obtenida por los sensores.

4.1 Manejo de la información empleando base de datos y servidor

Haciendo uso de la conexión WiFi los datos obtenidos por los sensores, serán enviados a través de la tarjeta NodeMCU hacia un servidor. Esa información, será almacenada en una base de datos creada en MySQL, quedando disponibles para ser consultados y visualizados en pantalla a través de una página web. Recordemos que el fin de esto, es poder crear un entorno que permita consultar los datos en tiempo real y los datos históricos, ya sea desde un dispositivo móvil o desde una computadora, los cuales podrán o no encontrarse conectados a la misma red o no.

Esto último dependerá de si se usa un servidor local o un servidor web. En nuestro caso, debido a los costos que implica alquilar un dominio, se optó por realizar el desarrollo en un servidor local, ya que el proceso de creación será el mismo que si empleamos servidor web.

Para los datos obtenidos, se decidió emplear una página web creada en código HTML, PHP y CSS. Por otro lado, el servidor local empleado será el servidor virtual XAMPP.

4.1.1 Creación de la página web

Para llevar a cabo la creación de una página web, se procedió a utilizar el editor de código multiplataforma Sublime Text. Este programa, es una herramienta que permite tener varios documentos abiertos mediante pestañas e incluso emplear varios paneles para aquellos que utilicen más de un monitor. Dispone de modo de pantalla completa, para aprovechar al máximo el espacio visual disponible de la pantalla, y además soporta un gran número de lenguajes (C, C++, C#, CSS, D, Erlang, HTML, Groovy, Haskell, HTML, Java, JavaScript, LaTeX, Lisp, Lua, Markdown, Matlab, OCaml, Perl, PHP, Python, R, Ruby, SQL, TCL, Textile and XML).

Si bien el desarrollo de la página puede hacerse en un simple archivo “.TXT”, a diferencia de este, sublime posee un buen número de herramientas para la edición del código y automatización de tareas, soporta macros, snippets y auto completar, incluso siendo posible agregar algunas características mediante plugins. Todo esto hace que sea preferible emplear Sublime Text (u otro editor de texto potente) para desarrollar la página web.

4.1.2 Lenguaje PHP [20]

PHP es un lenguaje de código abierto muy popular, adecuado para desarrollo web y que puede ser incrustado en HTML. Es popular porque un gran número de páginas y portales web están creadas con PHP. Código abierto significa que es de uso libre y gratuito para todos los programadores que quieran usarlo. Incrustado en HTML significa que en un mismo archivo vamos a poder combinar código PHP con código HTML, siguiendo unas reglas.

PHP se utiliza para generar páginas web dinámicas, siendo que llamamos página estática a aquella cuyos contenidos permanecen siempre igual, mientras que llamamos páginas dinámicas a aquellas cuyo contenido no es el mismo siempre.

El lenguaje PHP se procesa en servidores, los cuales son potentes ordenadores con un software y hardware especial. Cuando se escribe una dirección tipo `http://www.***.com/index.php` en un navegador web como Internet Explorer, Firefox o Chrome, se envían los datos de la solicitud al servidor que los procesa, reúne los datos (por eso decimos que es un proceso dinámico) y el servidor lo que devuelve es una página HTML como si fuera estática. El esquema se inicia con una petición de página web al servidor. El servidor recibe la petición, reúne la información necesaria consultando a bases de datos o a otras páginas webs, otros servidores, etc. Luego el servidor responde enviando una página web “normal” (estática) pero cuya creación ha sido dinámica realizando procesos de modo que la página web devuelta no siempre es igual. Por regla general este tipo de lenguaje suele ser utilizado para crear contenido dinámico y poder interactuar con el usuario.

4.1.3 Lenguaje HTML [21] [22]

El lenguaje HTML, hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del World Wide Web Consortium (W3C) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.

HTML se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (intérprete del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

HTML es un lenguaje de marcado que nos permite indicar la estructura de nuestro documento mediante etiquetas. Este lenguaje nos ofrece una gran adaptabilidad, una estructuración lógica y es fácil de interpretar tanto por humanos como por máquinas.

4.2 Software XAMPP [23]

Se procedió a realizar la instalación del servidor virtual XAMPP en la PC para de esta manera luego poder levantar desde el servidor la página creada.

XAMPP es una distribución de Apache que incluye varios softwares libres. El nombre es un acrónimo compuesto por las iniciales de los programas que lo constituyen: el servidor web Apache, los sistemas relacionales de administración de bases de datos MySQL y MariaDB, así como los lenguajes de programación Perl y PHP. La inicial X se usa para representar a los sistemas operativos Linux, Windows y Mac OS X.

- **Apache:** el servidor web de código abierto es la aplicación más usada globalmente para la entrega de contenidos web. Las aplicaciones del servidor son ofrecidas como software libre por la Apache Software Foundation.
- **MySQL/MariaDB:** con MySQL, XAMPP cuenta con uno de los sistemas relacionales de gestión de bases de datos más populares del mundo. En combinación con el servidor web Apache y el lenguaje PHP, MySQL sirve para el almacenamiento de datos para servicios web. En las versiones actuales de XAMPP esta base de datos se ha reemplazado por MariaDB, una ramificación (“Fork”) del proyecto MySQL.
- **PHP:** es un lenguaje de programación de código de lado del servidor que permite crear páginas web o aplicaciones dinámicas. Es independiente de plataforma y soporta varios sistemas de bases de datos.
- **Perl:** este lenguaje de programación se usa en la administración del sistema, en el desarrollo web y en la programación de red. También permite programar aplicaciones web dinámicas.

Además de estos componentes principales, esta distribución gratuita también incluye, según el sistema operativo, otras herramientas como el servidor de correo Mercury, el programa de administración de bases de datos phpMyAdmin, el software de analítica web Webalizer, OpenSSL, Apache Tomcat y los servidores FTP FileZilla o ProFTPD.

Un servidor XAMPP se puede instalar rápido y fácilmente como sistema de test local bajo Linux, Windows y Mac OS X con un único archivo ejecutable. El paquete del software contiene los mismos componentes que se utilizan en cualquier servidor web, de forma que permite a los desarrolladores testar proyectos localmente y transferirlos cómodamente a sistemas reales. Sin embargo, XAMPP no se recomienda como servidor público, debido a que como busca mantener la facilidad de uso, existen ciertas limitaciones en cuanto a seguridad.

4.2.1 Panel de control de XAMPP:

En la interfaz de usuario del panel de control se encuentran todas las acciones, siendo posible activar o desactivar los módulos por separado con un simple clic. Además, se dispone de diversas utilidades como:

- **Config:** para configurar XAMPP, así como otros componentes aislados.
- **Netstat:** muestra todos los procesos en funcionamiento en el ordenador local
- **Shell:** lanza una ventana de comandos UNIX
- **Explorer:** abre la carpeta XAMPP en el explorador de Windows
- **Services:** muestra todos los servicios en funcionamiento
- **Help:** incluye enlaces a foros de usuarios
- **Quit:** se usar para salir del panel de control

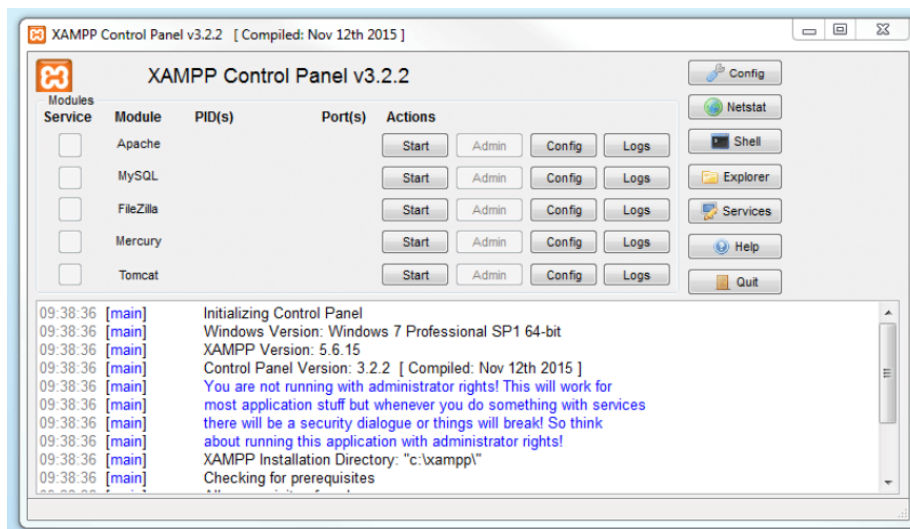


Figura N° 56 **Panel de control de XAMPP**

En la parte superior se pueden iniciar o interrumpir los módulos de XAMPP por separado mediante los comandos “Start” y “Stop” bajo “Actions”. Los módulos que se activaron aparecen marcados en verde.

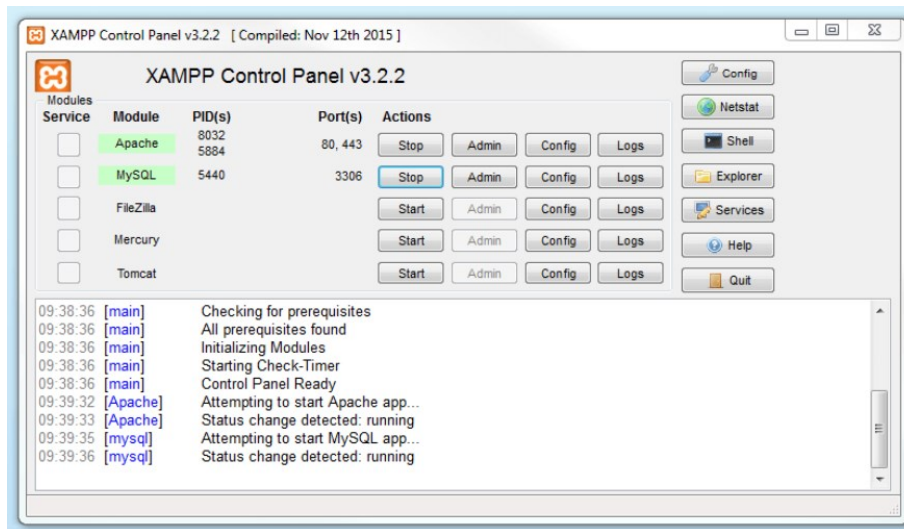


Figura N° 57 **Módulos activados se muestran en verde**

Una vez que el servidor esté encendido, una forma de verificar que esté funcionando correctamente es yendo a un navegador y escribiendo la dirección “localhost”. Si el servidor está funcionando correctamente, la ventana del navegador debería mostrar lo siguiente:

Sistema de monitoreo para heladas



Figura N° 58 Inicio del servidor

Para verificar ahora que la base de datos también está correctamente activa, podemos hacer click en phpMyAdmin o bien citar “localhost/phpmyadmin/” en el navegador. Si todo está bien, debería mostrar lo siguiente en pantalla:

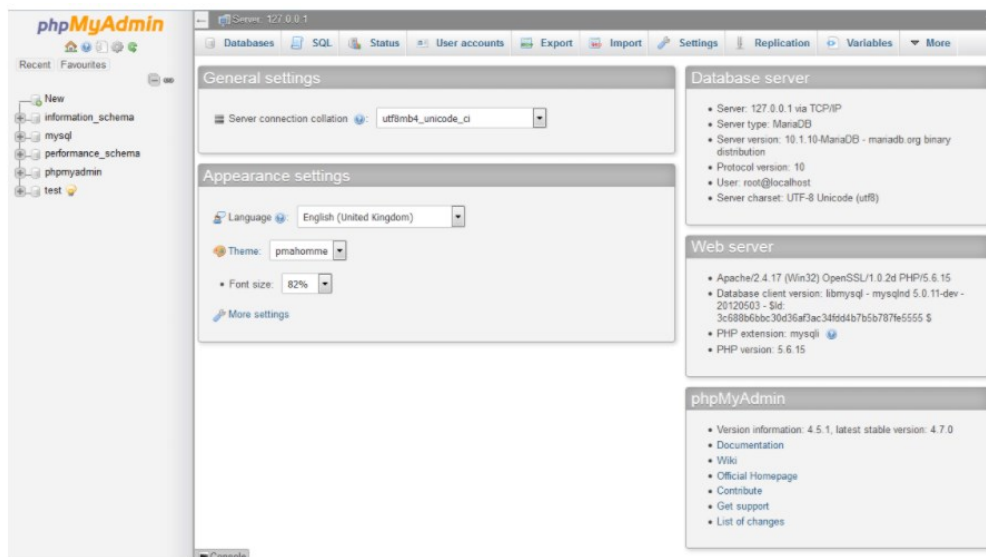


Figura N° 59 Inicio de phpMyAdmin

Para comenzar con el desarrollo del código, primero deberemos crear una carpeta dentro del servidor virtual, la cual contendrá todo aquello que forme parte de la página creada. Para esto, dicha carpeta deberá ser creada en la siguiente dirección:



Figura N° 60 Ruta de ubicación de la carpeta principal del servidor

Esto es necesario ya que el servidor sólo podrá levantar una página si se encuentra ubicada dentro de la carpeta htdocs. Para no mezclar información con otras páginas, se recomienda crear una carpeta aparte dentro de htdocs.

Entonces procedemos a crear una carpeta con el nombre que deseemos, en nuestro caso quedó con “proyectomodificado” y dentro de ella guardaremos todos los archivos creados, imágenes, tablas, etc.

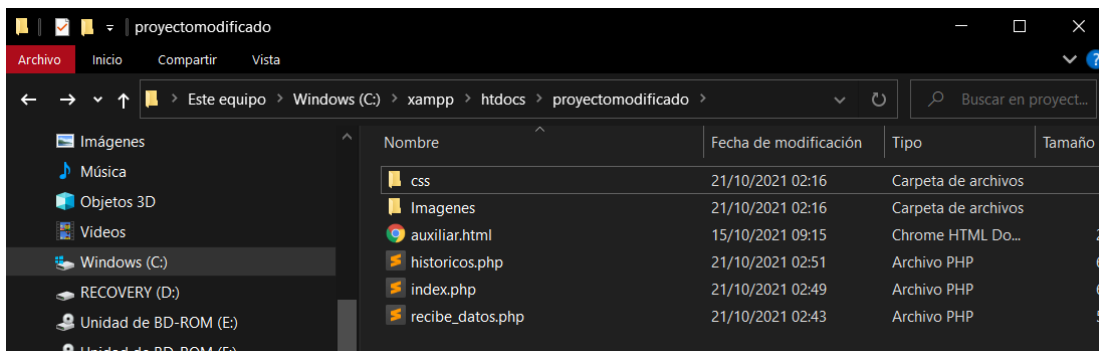


Figura N° 61 Carpeta contenedora de la página web

Puede observarse que la mencionada carpeta contiene archivos en su interior, donde cada uno cumple una función. La carpeta “Imágenes” solamente contiene las imágenes que son levantadas por la página web, y fue creada simplemente para mantener un orden. Por otro lado, la carpeta “CSS” contiene los archivos “.css” en los cuales se desarrolla todo lo relacionado con la edición gráfica de las páginas web.

El formato CSS representa hojas de estilo en cascada (Cascading Style Sheets). Es un lenguaje utilizado para especificar la presentación de documentos en una página web y se almacena en archivos nombrados con la extensión CSS. La codificación dentro del documento (php y otros lenguajes de marcas) se organiza y formatea utilizando el lenguaje CSS para su presentación. La mayoría de los navegadores web utiliza CSS para organizar datos e información. Los elementos dentro del documento CSS afectan a la apariencia, al color, a la fuente, a las imágenes y a otros detalles. Este formato permite la gestión mejorada del contenido, la flexibilidad y la accesibilidad a la información de la página web.

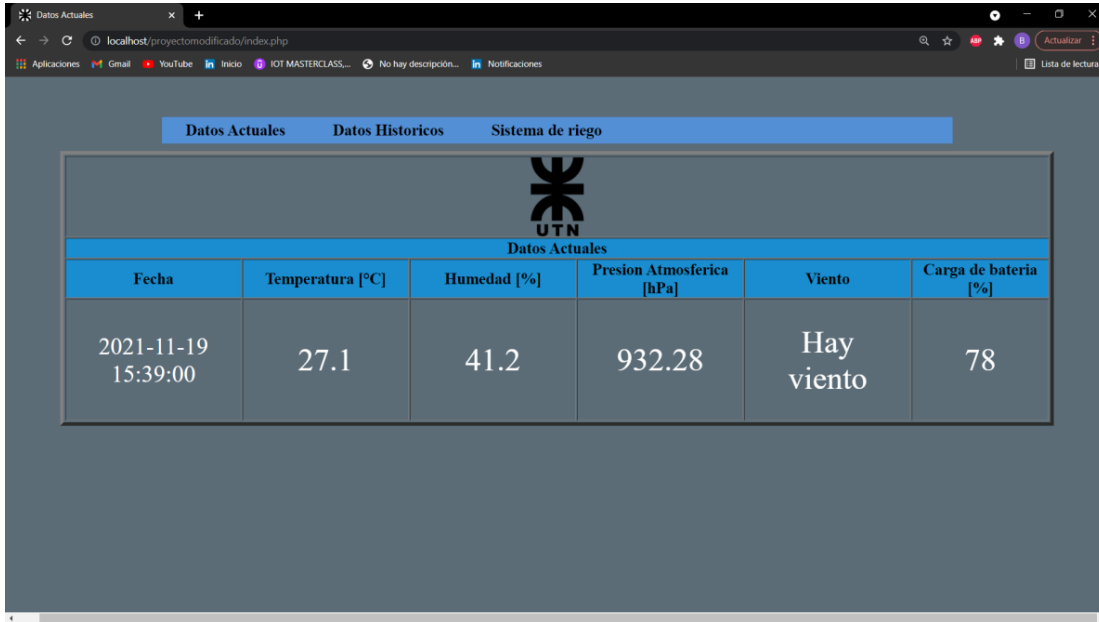
El uso de CSS posibilita que varias páginas cuenten con el mismo formato y la misma organización fácilmente, así como utilizar métodos de renderización [24].

El cuerpo de la página web se compone por 3 archivos php y un archivo html, tal como se observa en la imagen anterior. Estos fueron nombrados según la función que cumplen como:

- Index.php
- Históricos.php
- recibe_datos.php
- auxiliar.html

A continuación, se ilustrará una captura del diseño final de las páginas, añadiendo una breve explicación de la función que cumplen.

- **Enlace de Inicio:**



Datos Actuales					
Fecha	Temperatura [°C]	Humedad [%]	Presion Atmosferica [hPa]	Viento	Carga de bateria [%]
2021-11-19 15:39:00	27.1	41.2	932.28	Hay viento	78

Figura N° 62 Vista principal de la página web

El archivo index.php, contiene la estructura principal de la página web. En ella, nos encontraremos con la posibilidad de seleccionar a través de botones, si deseamos visualizar datos históricos, datos actuales o si deseamos activar o desactivar el sistema de riego. En el centro de la pantalla vamos a observar información de temperatura en grados centígrados, humedad relativa en porcentaje, presión atmosférica en pascales, información de fecha y hora a la cual pertenecen los datos observados, un indicador de presencia de viento o no en el sector frutal y por último información del porcentaje de carga que tiene la batería que alimenta el circuito de telemetría.

Siendo localhost la raíz del servidor, para poder ejecutar la página debemos introducir como enlace la ubicación de la página creada, por ejemplo, para este caso:

- localhost/proyectomodificado/index.php

Que será tomada al apretar la tecla “Enter” como:

- <http://localhost/proyectomodificado/index.php>

Si hacemos click en “Datos Actuales” lo que sucederá es que se actualizará la página web, ya que nos encontramos presentes en datos actuales. Si hacemos click en “Datos Históricos” veremos que se abrirá una nueva pestaña, donde nos permitirá seleccionar una fecha y hora de inicio y una fecha y hora de fin, las cuales nos permitirán seleccionar el lapso de tiempo de datos históricos que deseamos ver en pantalla. La ventana que se abre es la siguiente, y vendría a ser una página auxiliar creada en HTML, cuyo enlace es el siguiente:

- <http://localhost/proyectomodificado/auxiliar.html>

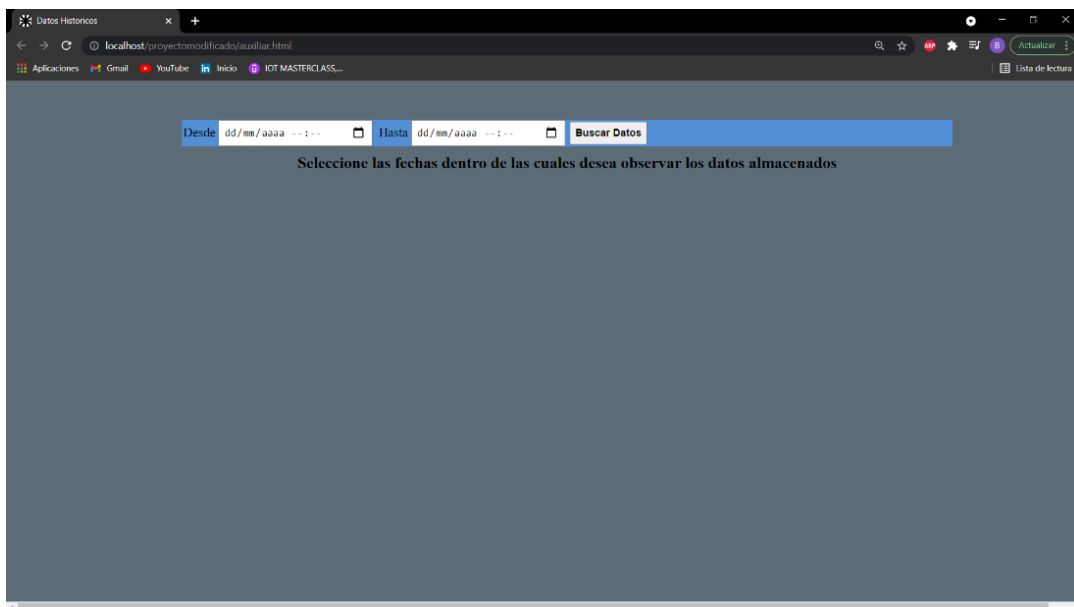


Figura N° 63 Pestaña que se abre al seleccionar en Datos Históricos

Una vez que seleccione las fechas, debemos seleccionar la opción buscar datos para que esa información sea pasada a una página php llamada históricos.php la cual se encargará de realizar las consultas necesarias a la base de datos, para luego mostrar en una tabla los datos correspondientes a ese intervalo de tiempo:

- <http://localhost/proyectomodificado/historicos.php>

Sistema de monitoreo para heladas



UTN					
Datos Actuales					
Fecha	Temperatura [°C]	Humedad [%]	Presion Atmosferica [hPa]	Viento	Carga de bateria [%]
2021-11-19 15:39:00	27.1	41.2	932.28	Hay viento	78

Figura N° 64 Pestaña que se abre al clickear en **Buscar Datos**

De esta forma, le proporcionamos al usuario una forma cómoda y rápida de visualizar los datos en pantalla, como también un acceso al historial de datos. Es importante aclarar que el diseño está pensado para que los valores en tiempo real sean actualizados cada 1 minuto.

Si seleccionamos la “opción sistema de riego”, se abrirá una página web que nos permitirá comunicarnos con otra tarjeta NodeMCU que estará funcionando como una estación. Esta contiene en su código una página web que permitirá al usuario activar o desactivar el sistema de riego a partir de un par de botones:

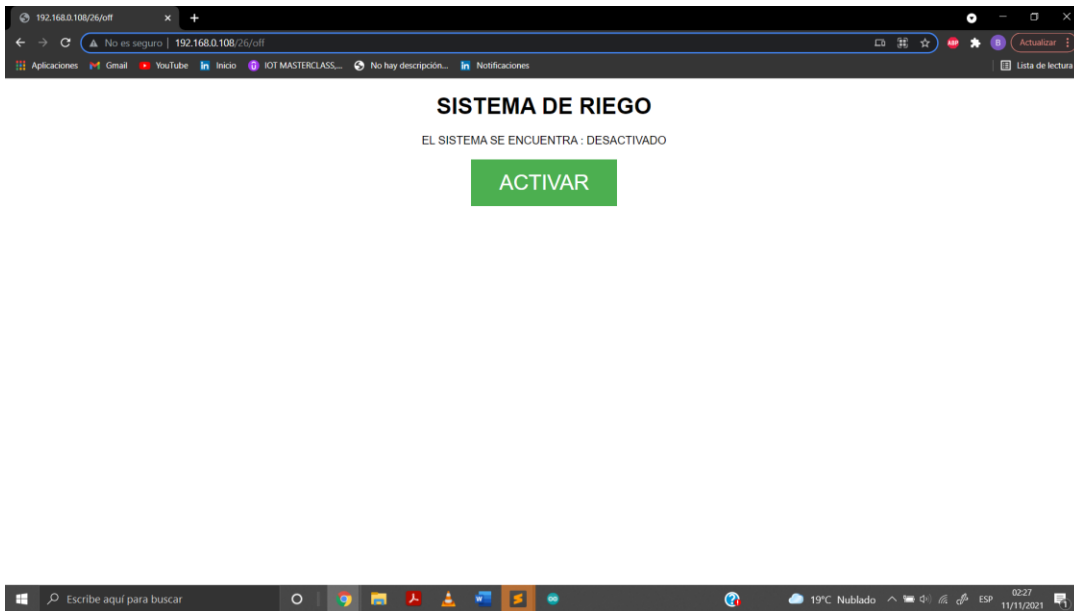


Figura N° 65 **Pestaña que se abre al clickear en “Sistema de riego”**

4.3 Base de datos MySQL [25]

MySQL es un sistema de gestión de base de datos (SGBD) de código abierto. El SGBD MySQL pertenece actualmente a Oracle y funciona con un modelo cliente-servidor. Esto quiere decir que los ordenadores que instalan y ejecutan el software de gestión de base de datos se denominan clientes. Cada vez que necesitan acceder a los datos, los clientes se conectan al servidor del sistema de gestión de base de datos y le solicitan la información que necesitan. El servidor brindara la información siempre y cuando tenga los derechos de acceso.

Además de ser usado como sistema de gestión de base de datos, también es bastante frecuente encontrarse MySQL funcionando con los sistemas operativos, servidores y lenguajes de programación de Linux, Apache y PHP/Per/Python para desarrollar aplicaciones web, por ejemplo, webs dinámicas. Es por esto que suele encontrarse el acrónimo LAMP (las iniciales de Linux, Apache, MySQL y PHP/Per/Python) cuando se habla de MySQL.

4.3.1 PhpMyAdmin [26]

PhpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web. Con esta herramienta puedes crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos. Algunas características de PhpMyAdmin son:

- Interfaz web intuitiva
- Soporta la mayoría de las características de MySQL:
- Examinar y eliminar bases de datos, tablas, vistas, campos e índices
- Crear, copiar, eliminar, cambiar el nombre y alterar bases de datos, tablas, campos e índices
- Mantenimiento del servidor, bases de datos y tablas, con propuestas sobre la configuración del servidor
- Ejecutar, editar y marcar cualquier sentencia SQL, incluso batch-queries
- Administrar cuentas de usuario de MySQL y privilegios
- Gestionar procedimientos almacenados y disparadores
- Importar datos desde CSV y SQL
- Exportar datos a varios formatos: CSV, SQL, XML, PDF, ISO/IEC 26300 - OpenDocument de Texto y Hoja de Cálculo, Word, LATEX y otros
- Administración de varios servidores
- Creación de gráficos del layout de la base de datos en varios formatos
- Creación de consultas complejas mediante Query-by-example (QBE)
- Búsqueda global en una base de datos o en un subconjunto de la misma
- Transformar datos almacenados en cualquier formato utilizando un conjunto de funciones predefinidas, como mostrar datos BLOB como imagen o enlace de descarga

4.3.2 HeidiSQL [27]

HeidiSQL, inicialmente conocido como MySQL-Front, es un software libre y de código abierto que permite conectarse a servidores MySQL y sus derivaciones como MariaDB y Percona Server, así como Microsoft SQL Server y PostgreSQL.

MySQL-Front comenzó a ser desarrollado en Delphi por el programador alemán Ansgar Becker, quién por motivos personales dejó el proyecto sin terminar. Más tarde el desarrollador alemán Nile Hoyer contactó a Ansgar y adquirió los derechos para utilizar el nombre "MySQL-Front" en su propio proyecto, sin embargo, tuvo que cancelarlo porque surgió una infracción de derechos de autor con MySQL Labs sobre el uso del nombre "MySQL". Finalmente, Ansgar y otros colaboradores retomaron el proyecto MySQL-Front renombrándolo HeidiSQL.

Para administrar las bases de datos con HeidiSQL, los usuarios deben iniciar una sesión en un servidor MySQL local o remoto. Sus características permiten realizar las operaciones de base de datos más comunes y avanzadas, sin embargo, aún sigue en desarrollo a fin de integrar la máxima funcionalidad que se espera en una interfaz de base de datos de SQL.

4.3.3 Creación de base de datos y tablas:

Tal como se mencionó, haremos uso del software HeidiSQL para poder tener una forma más amigable y sencilla de crear la base de datos.

En primer lugar, debemos iniciar MySQL en XAMPP y luego abrir HeidiSQL. Configuraremos el inicio tal como se observa en la siguiente figura y le daremos a abrir:

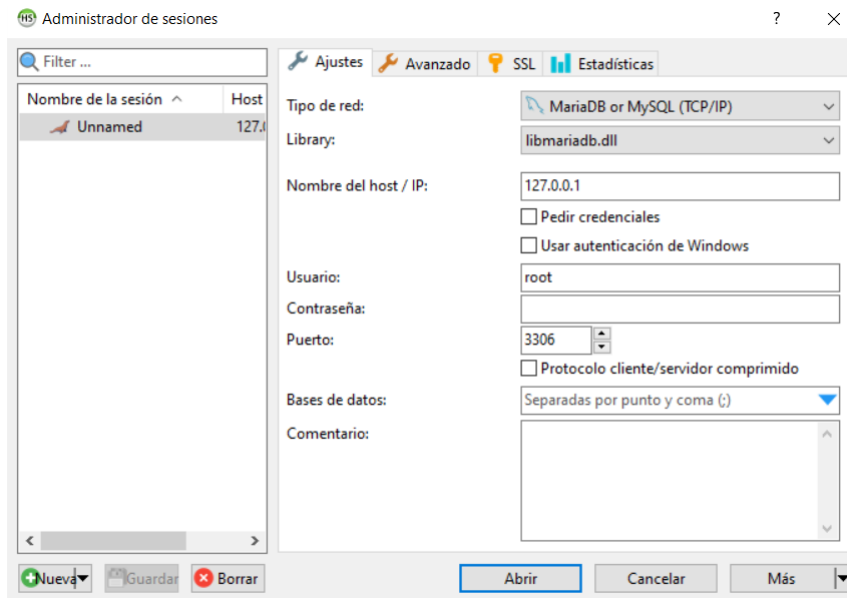


Figura N° 66 Administrador de sesiones de HeidiSQL

Ahora procederemos a crear la base de datos llamada “proyectofinal” y dentro de ella crearemos dos tablas de datos:

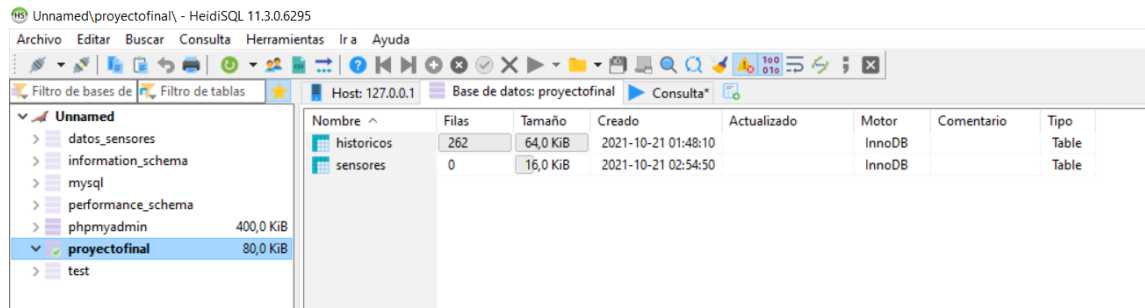
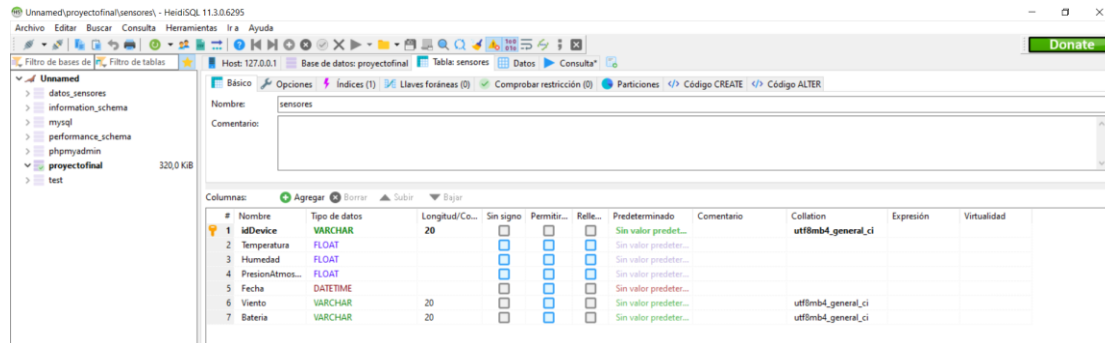


Figura N° 67 Base de datos creada

Una de las tablas se llama “sensores” y es la que va a almacenar siempre los últimos datos que llegan de la placa NodeMCU. A partir de esta tabla es que nosotros actualizaremos los valores actuales que mostraremos en el inicio de nuestra página web. Dicha tabla esta creada con el siguiente formato:

Sistema de monitoreo para heladas



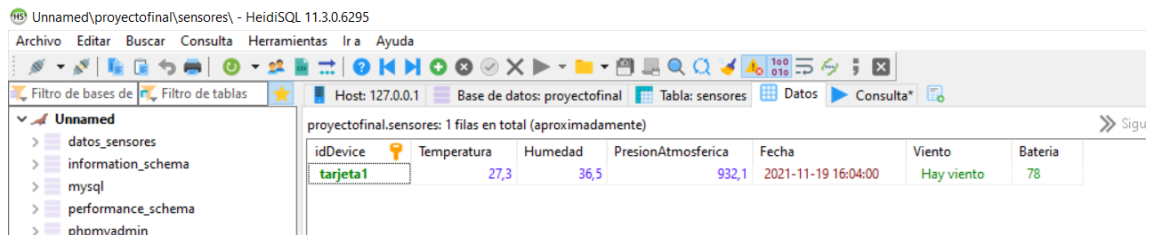
The screenshot shows the HeidiSQL interface for a table named 'sensores'. The table structure is as follows:

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir...	Rell...	Predeterminado	Comentario	Collation	Expresión	Virtualidad
1	idDevice	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
2	Temperatura	FLOAT		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...				
3	Humedad	FLOAT		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...				
4	PresionAtmos...	FLOAT		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...				
5	Fecha	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...				
6	Viento	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
7	Bateria	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		

Figura N° 68 Imagen del formato de la tabla de datos “sensores”

Dentro de la tabla vemos que se han creado 5 columnas donde cada una alojara 1 dato que se ira actualizando cada 1 minuto por el dato más reciente.

La primera columna tiene el nombre idDevice, donde se pretende almacenar la identificación del dispositivo. Esta columna se coloca pensando en la posibilidad de que en algún momento haya más de un dispositivo que esté accediendo a guardar datos en la tabla. Entonces para poder diferenciarlos, se asignará una etiqueta a cada dispositivo. En el caso de nuestro proyecto, como solo es una NodeMCU quien accede a esta base de datos, el idDevice se llama “tarjeta1”. Luego tenemos 3 columnas del tipo FLOAT, donde pretendemos guardar los valores de humedad relativa, presión y temperatura. Por último, tenemos una columna del tipo DATETIME donde almacenaremos la información de fecha y hora a la cual pertenecen los datos almacenados, y una columna del tipo Varchar para almacenar información de viento. La tabla siempre estará completa como se muestra en el ejemplo de la siguiente imagen:



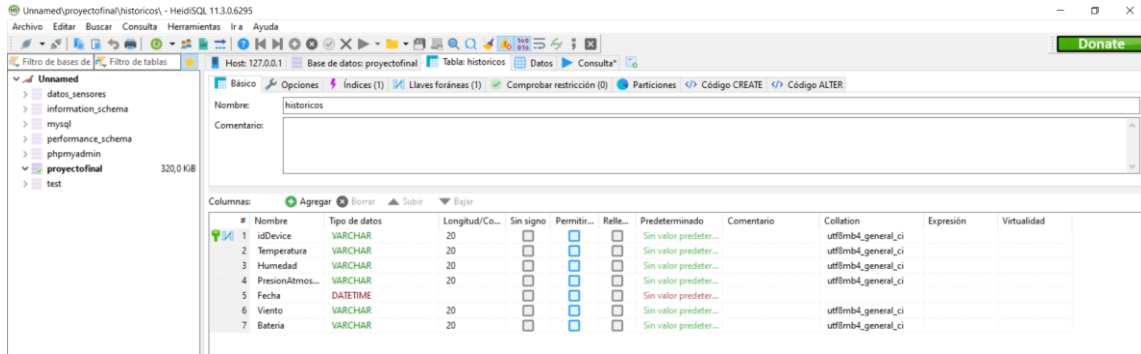
The screenshot shows the HeidiSQL interface displaying a single row of data in the 'sensores' table:

idDevice	Temperatura	Humedad	PresionAtmosferica	Fecha	Viento	Bateria
tarjeta1	27,3	36,5	932,1	2021-11-19 16:04:00	Hay viento	78

Figura N° 69 Ejemplo de carga de datos en tabla sensores

Sistema de monitoreo para heladas

Luego tenemos la tabla llamada históricos, donde pretendemos ir almacenando el historial de datos, de forma tal que puedan ser consultados en algún momento. Esta tabla se creó de la siguiente manera:

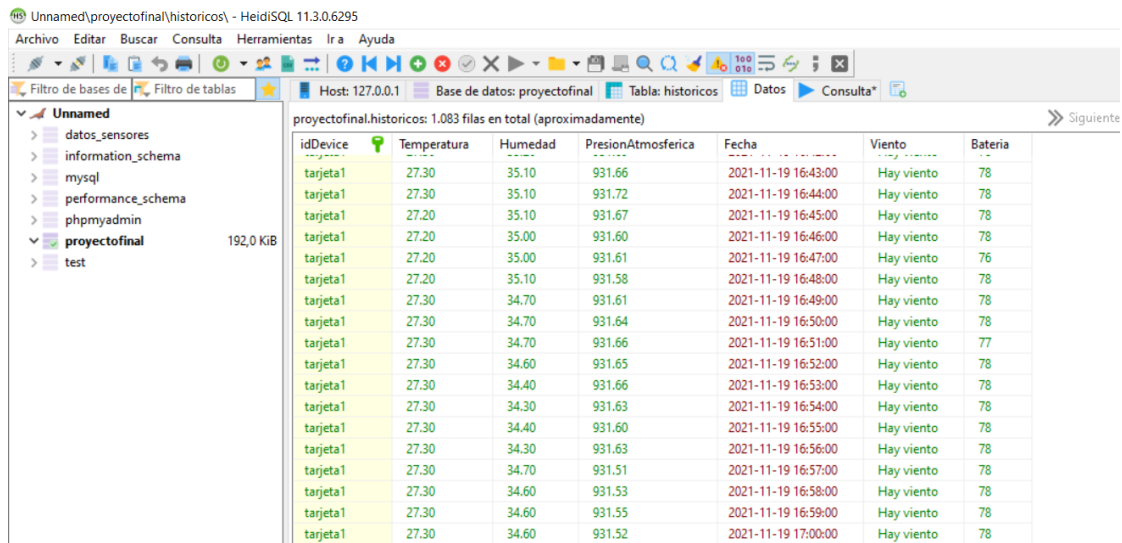


The screenshot shows the HeidiSQL interface for a MySQL database. The table 'historicos' is selected, and its structure is displayed in the 'Columnas' section. The table has 7 columns: idDevice (VARCHAR, 20), Temperatura (VARCHAR, 20), Humedad (VARCHAR, 20), PresionAtmos... (VARCHAR, 20), Fecha (DATETIME), Viento (VARCHAR, 20), and Bateria (VARCHAR, 20). All columns are of type VARCHAR with a length of 20, except for 'Fecha' which is DATETIME. The 'Collation' for all VARCHAR columns is utf8mb4_general_ci.

#	Nombre	Tipo de datos	Longitud/Ce...	Sin signo	Permitir...	Relle...	Predeterminado	Comentario	Collation	Expresión	Virtualidad
1	idDevice	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
2	Temperatura	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
3	Humedad	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
4	PresionAtmos...	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
5	Fecha	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...				
6	Viento	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		
7	Bateria	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Sin valor predeter...		utf8mb4_general_ci		

Figura N° 70 Imagen del formato de la tabla de datos “históricos”

Vemos que el formato de la tabla es igual al de la tabla sensores, lo cual era de esperarse ya que son los mismos datos los que se van a guardar. A diferencia de la tabla sensores, acá los datos temperatura, humedad y presión se guardan en variables varchar para reducir el espacio de memoria utilizada. En la siguiente imagen podemos ver cómo se irán almacenando los datos:



The screenshot shows the HeidiSQL interface displaying data loaded into the 'historicos' table. The table contains 1,083 rows of data. The columns are: idDevice, Temperatura, Humedad, PresionAtmosferica, Fecha, Viento, and Bateria. The data shows temperature, humidity, atmospheric pressure, date, wind, and battery level for multiple sensor cards.

idDevice	Temperatura	Humedad	PresionAtmosferica	Fecha	Viento	Bateria
tarjeta1	27.30	35.10	931.66	2021-11-19 16:43:00	Hay viento	78
tarjeta1	27.30	35.10	931.72	2021-11-19 16:44:00	Hay viento	78
tarjeta1	27.20	35.10	931.67	2021-11-19 16:45:00	Hay viento	78
tarjeta1	27.20	35.00	931.60	2021-11-19 16:46:00	Hay viento	78
tarjeta1	27.20	35.00	931.61	2021-11-19 16:47:00	Hay viento	76
tarjeta1	27.20	35.10	931.58	2021-11-19 16:48:00	Hay viento	78
tarjeta1	27.30	34.70	931.61	2021-11-19 16:49:00	Hay viento	78
tarjeta1	27.30	34.70	931.64	2021-11-19 16:50:00	Hay viento	78
tarjeta1	27.30	34.70	931.66	2021-11-19 16:51:00	Hay viento	77
tarjeta1	27.30	34.60	931.65	2021-11-19 16:52:00	Hay viento	78
tarjeta1	27.30	34.40	931.66	2021-11-19 16:53:00	Hay viento	78
tarjeta1	27.30	34.30	931.63	2021-11-19 16:54:00	Hay viento	78
tarjeta1	27.30	34.40	931.60	2021-11-19 16:55:00	Hay viento	78
tarjeta1	27.30	34.30	931.63	2021-11-19 16:56:00	Hay viento	78
tarjeta1	27.30	34.70	931.51	2021-11-19 16:57:00	Hay viento	78
tarjeta1	27.30	34.60	931.53	2021-11-19 16:58:00	Hay viento	78
tarjeta1	27.30	34.60	931.55	2021-11-19 16:59:00	Hay viento	78
tarjeta1	27.30	34.60	931.52	2021-11-19 17:00:00	Hay viento	78

Figura N° 71 Ejemplo de carga de datos en tabla sensores

4.4 Desarrollo de la página web:

En primer lugar, vamos a comenzar creando el inicio de nuestra página web, tal como ya observamos anteriormente en la imagen 4.1.2.7, necesitamos darle color y forma a lo que queremos observar en pantalla. Parte de lo explicado para este archivo php de inicio será aplicado a los demás. Iremos viendo el código por partes para poder entenderlo y explicarlo de una mejor forma, pero solamente haciendo hincapié en las partes más importantes del código. En primer lugar, veremos algunos elementos principales de la estructura básica del código html:

- **<!DOCTYPE html>**: El tipo de documento. Es un preámbulo requerido. Anteriormente, los tipos de documento actuaban como vínculos a un conjunto de reglas que el código HTML de la página debía seguir para ser considerado bueno, lo que podía significar la verificación automática de errores y algunas otras cosas de utilidad. Sin embargo, hoy día es simplemente un artefacto antiguo que a nadie le importa, pero que debe ser incluido para que todo funcione correctamente.
- **<html></html>**: Este elemento encierra todo el contenido de la página entera y a veces se lo conoce como el elemento raíz (root element).
- **<head></head>**: Este elemento actúa como un contenedor de todo aquello que quieres incluir en la página HTML que no es contenido visible por los visitantes de la página. Incluye cosas como palabras clave (keywords), una descripción de la página que quieres que aparezca en resultados de búsquedas, código CSS para dar estilo al contenido, declaraciones del juego de caracteres, etc.
- **<meta charset="utf-8"> — <meta>**: Este elemento establece el juego de caracteres que el documento usará en utf-8, que incluye casi todos los caracteres de todos los idiomas humanos. Básicamente, puede manejar cualquier contenido de texto que puedas incluir. No hay razón para no establecerlo, y puede evitar problemas en el futuro.

- **<title></title>**: El elemento <title> establece el título de tu página, que es el título que aparece en la pestaña o en la barra de título del navegador cuando la página es cargada, y se usa para describir la página cuando es añadida a los marcadores o como favorita.
- **<body></body>**: Encierra todo el contenido que deseas mostrar a los usuarios web que visiten tu página, ya sea texto, imágenes, videos, juegos, pistas de audio reproducibles, y demás.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <link rel="shortcut icon" type="image/x-icon" href="Imagenes/Logo.jpg">
5     <title>Datos Actuales</title>
6     <!-- a continuación le digo que las modificaciones las tome del siguiente CCS en cuanto al formato visual-->
7     <link rel="stylesheet" href="css/estilos de tablas.css" type="text/css">
8     <style type="text/css">
9       td{
10         padding: 30px
11       }
12     </style>
13   </head>
14   <!------->
15   <!-- a continuación lo que hago es agregarle el menu superior a la pagina, es decir lo que contiene Datos actuales y Datos Historicos, como para dar
16   opcion a que retorne a donde quiera-->
17   <body>
18     <header>
19       <nav>
20         <ul>
21           <li><a href="http://localhost/proyectomodificado/index.php"target=> <b>Datos Actuales </b></a> </li>
22           <li><a href="http://localhost/proyectomodificado/auxiliar.html"target="_blank"> <b>Datos Historicos</b></a> </li>
23           <li ><a href="http://192.168.0.108/" target="_blank"> <b>Sistema de riego </b></a> </li>
24         </ul>
25       </header>
26       <!-- Al pulsar en datos historicos le indico que viaje al archivo auxiliar.html, Este es un archivo que uso de intermediario para poder
27       emplear
28       el metodo POST, ya que no encuentre forma de que funcione si no es desde un html a un php.
29       Lo que hago en ese html es seleccionar el rango de fechas desde donde y hasta donde deseo ver los datos historicos. Despues con un boton
30       envio esos datos a la pagina historicos.php, donde los uso para buscar en la base de datos Mysql -->
31       <!-- <li><a href="http://localhost/proyectomodificado/historicos.php"target=> <b>Datos Historicos</b></a> </li>
32       Target_blank me permite hacer que al clicar en TEMPERATURA HUMEDAD O VIENTO, se abra el link en una nueva pestaña-->
33     </body>
34   </html>

```

Figura N° 72 Primera parte del archivo index.php

Con las siguientes sentencias, voy a crear las tres opciones disponibles en la línea superior de la pantalla llamadas “Datos Actuales”, “Datos Historicos” y “Sistema de riego”. Vale mencionar que todo código original de los archivos se irá colocando en fuente courier new para que pueda ser distinguido rápidamente.

```

<li><a href="http://localhost/proyectomodificado/index.php"target=> <b>Datos Actuales
</b></a> </li>
  <li><a href="http://localhost/proyectomodificado/auxiliar.html"target=_blank>
<b>Datos Historicos</b></a> </li>
  <li ><a href="http://192.168.0.108/" target="_blank"> <b>Sistema de riego </b></a>
</li>

```

En estas líneas tenemos dos atributos importantes:

- **Href** =: es un atributo que se utiliza para hacer referencia a otro documento. Se utiliza en etiquetas de anclaje <a> para crear hipervínculos en sitios web. El valor del atributo contiene la URL a la que apunta el hipervínculo.
- **Target** =: el atributo target indica dónde se abre el enlace. En este caso la página que se cargue se actualizará en la misma pestaña. En el caso de **target=_blank** se abrirá una nueva pestaña al hacer click en la opción

En el caso de hacer click en “Datos Actuales”, vemos que simplemente se actualizará la página ya que le está indicando que cargue index.php. Por otro lado al hacer click en la opción “Datos Historicos”, se abrirá una nueva pestaña y se cargará la página auxiliar.html.

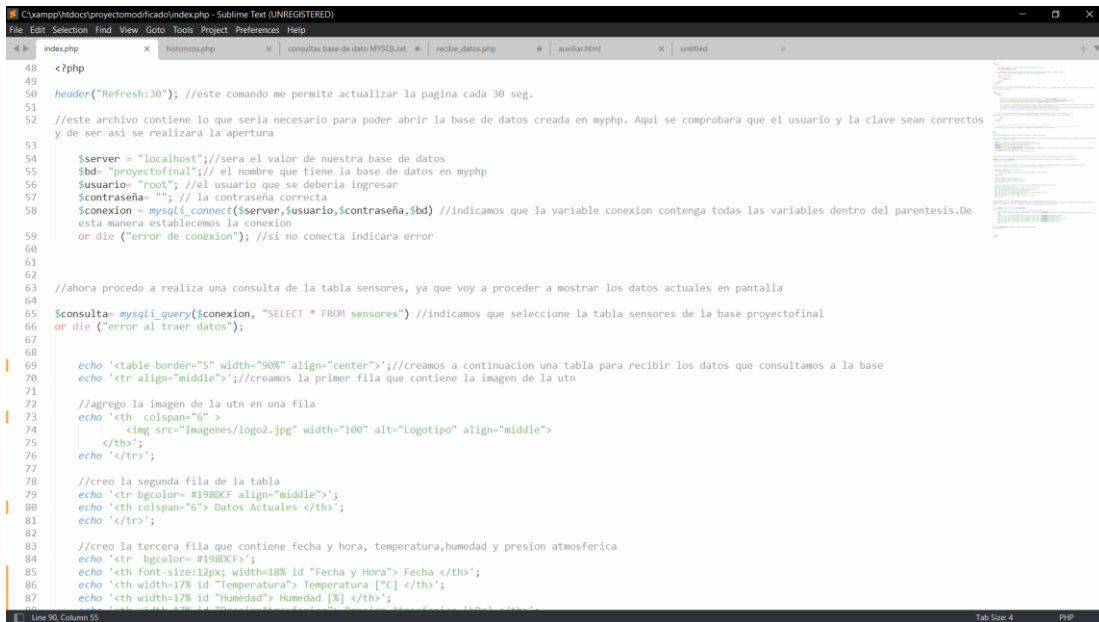
Este archivo html fue creado para uso intermediario, ya que solo así fue posible emplear el método POST que nos permitirá enviar a un archivo php valores seleccionados de fecha y hora que deseamos buscar en la base de datos para mostrar los datos históricos.

Tal como se verá más adelante, en auxiliar.html lo que realizo es una selección del rango de fechas desde donde y hasta donde deseo ver los datos históricos. Después con un botón, enviar esos datos a la página historicos.php, donde serán empleados para consultar la base de datos.

Por último la opción “Sistema de riego”, llamará a la Ip estática de otra tarjeta NodeMCU, la cual estará funcionando como un servidor, es decir que será configurada en modo estación. Al ejecutar esta opción, se abrirá una nueva pestaña que me dará la posibilidad de activar o desactivar el sistema de riego, con tan solo seleccionar el botón correspondiente.

En la siguiente imagen podremos observar la segunda parte del código, donde ya establecemos conexión con la base de datos, tablas y una posterior representación en pantalla de los datos:

Sistema de monitoreo para heladas



```
48 <?php
49
50 header("Refresh:30"); //este comando me permite actualizar la pagina cada 30 seg.
51
52 //este archivo contiene lo que sería necesario para poder abrir la base de datos creada en myphp. Aquí se comprobara que el usuario y la clave sean correctos
53 y de ser así se realizara la apertura
54
55 $server = "localhost"; //sera el valor de nuestra base de datos
56 $bd= "proyectofinal"; // el nombre que tiene la base de datos en myphp
57 $usuario= "root"; //el usuario que se deberia ingresar
58 $contraseña= ""; // la contraseña correcta
59 $conexion = mysqli_connect($server,$usuario,$contraseña,$bd) //indicamos que la variable conexion contenga todas las variables dentro del parentesis.De
60 esta manera establecemos la conexion
61 or die ("error de conexion"); //si no conecta indicara error
62
63
64 //ahora procedo a realiza una consulta de la tabla sensores, ya que voy a proceder a mostrar los datos actuales en pantalla
65
66 $consulta= mysqli_query($conexion, "SELECT * FROM sensores") //indicamos que seleccione la tabla sensores de la base proyectofinal
67 or die ("error al traer datos");
68
69
70 echo '<table border="5" width="90%" align="center">;//creamos a continuacion una tabla para recibir los datos que consultamos a la base
71 echo '<tr align="middle">;//creamos la primer fila que contiene la imagen de la utn
72
73 //agrego la imagen de la utn en una fila
74 echo '<th colspan="6">
75 
76 </th>;
77 echo '</tr>;
78 //creo la segunda fila de la tabla
79 echo '<tr bgcolor= #1980CF align="middle">;
80 echo '<th colspan="6"> Datos Actuales </th>;
81 echo '</tr>;
82
83 //creo la tercera fila que contiene fecha y hora, temperatura,humedad y presion atmosferica
84 echo '<tr bgcolor= #1980CF>;
85 echo '<th font-size:12px; width:10% id "Fecha y Hora"> Fecha </th>;
86 echo '<th width:17% id "Temperatura"> Temperatura [°C] </th>;
87 echo '<th width:17% id "Humedad"> Humedad [%] </th>;
```

Figura N° 73 Segunda parte del archivo index.php

El siguiente comando, nos permite realizar una actualización de la página cada 30 segundos. Esto es necesario ya que cada 1 minuto estará llegando nueva información a la base de datos y por lo tanto necesito refrescar la tabla de datos actuales observada en pantalla. De esta manera, logré que se ejecute nuevamente la consulta a la base de datos de forma automática, sin que el usuario deba actualizar la página.

```
header("Refresh:30"); //este comando me permite actualizar la pagina cada 30 seg.
```

Luego procedemos a conectarnos con la base de datos. Para esto, creamos las variables server, bd, usuario, contraseña y conexión, a las cuales les asignaremos valores. Vemos que para definir que son variables, debemos agregar el signo \$ en el inicio de cada una. Haciendo uso de la función `mysqli_connect()`, pasamos información del servidor, usuario, contraseña y la base de datos a la cual queremos conectarnos. Si por algún motivo no puede conectarse, devolverá false y si todo está bien devolverá true.

```
$server = "localhost"; //sera el valor de nuestra base de datos
$bd= "proyectofinal"; // el nombre que tiene la base de datos en myphp
$usuario= "root"; //el usuario que se deberia ingresar
```

Sistema de monitoreo para heladas

```
$contraseña= ""; // la contraseña correcta
$conexion = mysqli_connect($server,$usuario,$contraseña,$bd) //indicamos que la
variable conexion contenga todas las variables dentro del parentesis.De esta
manera establecemos la conexion
or die ("error de conexion"); //si no conecta indicara error
```

Una vez establecida la conexión, podemos proceder a realizar una consulta a la base de datos. Para esto, emplearemos la función `mysqli_query()` a la cual debemos pasarle la variable conexión y la tabla que queremos consultar dentro de la base de datos. También devuelve true o false según la consulta haya sido exitosa o no.

```
$consulta= mysqli_query($conexion, "SELECT * FROM sensores") //indicamos que seleccione la
tabla sensores de la base proyectofinal
or die ("error al traer datos");
```

Realizada la consulta, debemos crear la tabla donde vamos a almacenar los datos que llegan de la consulta. Con la sentencia hecha, usada para imprimir una cadena de caracteres entre otras opciones, podemos ejecutar código html.

En primer lugar iniciamos la tabla indicando la cantidad de columnas con border, el ancho de la tabla con width, e indicamos que la queremos centrada en la pantalla con align="center". Luego procedemos a crear la primera fila, donde solamente colocaremos la imagen de la facultad. Vemos que indicamos dónde buscar la imagen dentro de la carpeta cabecera de todos los archivos, y luego se le da un ajuste en cuanto a tamaño y ubicación:

```
echo '<table border="5" width="80%" align="center">'; //creamos a continuacion una tabla
para recibir los datos que consultamos a la base

echo '<tr align="middle">'; //creamos la primer fila que contiene la imagen de la
utn

//agrego la imagen de la utn en una fila
echo '<th colspan="5" >
    
    </th>';
echo '</tr>';
```

Ahora procedemos a crear una nueva fila de la tabla, la cual contendrá la etiqueta

“Datos Actuales”:

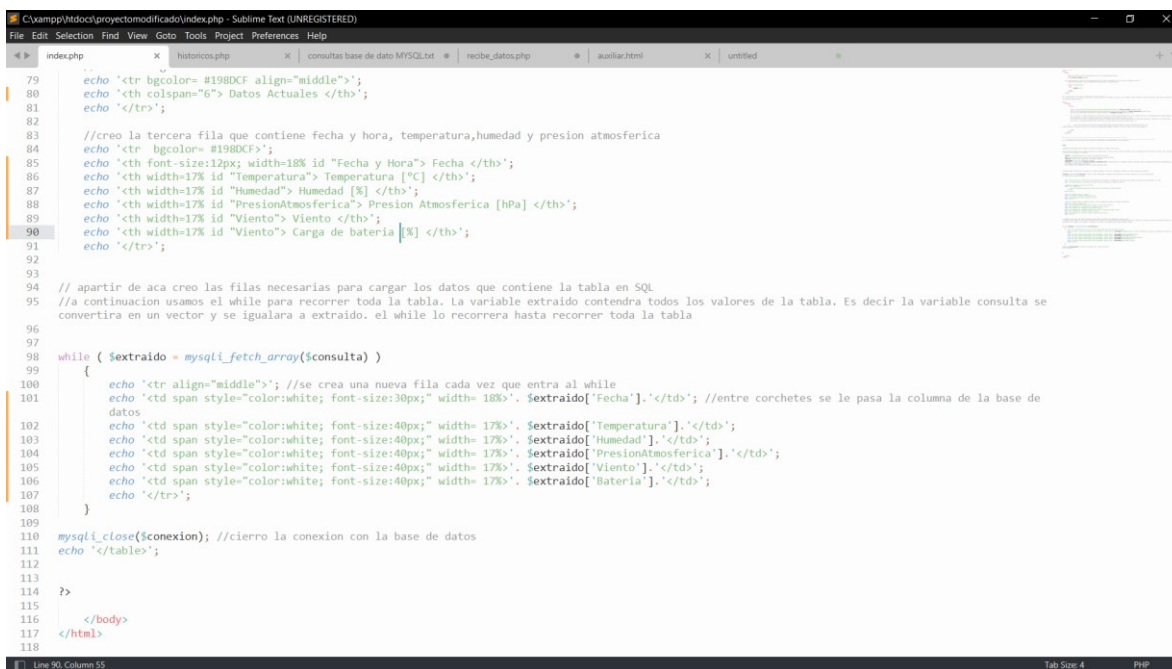
```
//creo la segunda fila de la tabla
echo '<tr bgcolor= #198DCF align="middle">';
echo '<th colspan="5"> Datos Actuales </th>';
echo '</tr>';
```

Continuamos con la siguiente fila, la cual contendrá 5 columnas empleadas para indicar el nombre de la variable que se va a mostrar:

```
//creo la tercera fila que contiene fecha y hora, temperatura, humedad y presion atmosferica
echo '<tr bgcolor= #198DCF>';
    echo '<th font-size:12px; width=20% id "Fecha y Hora"> Fecha </th>';
    echo '<th width=20% id "Temperatura"> Temperatura [°C] </th>';
    echo '<th width=20% id "Humedad"> Humedad [%] </th>';
    echo '<th width=20% id "PresionAtmosferica"> Presion Atmosferica [hPa] </th>';
    echo '<th width=20% id "Viento"> Viento </th>';
    echo '<th width=20% id "Bateria"> Bateria </th>';
echo '</tr>';
```

Con las siguientes líneas de código empezaremos a traer los datos de la tabla seleccionada en la consulta.

Sistema de monitoreo para heladas



```
index.php
historicos.php
consultas base de dato MySQL.txt
recibe_datos.php
auxiliar.html
untitled

79 echo '<tr bgcolor=#198DCF align="middle">';
80 echo '<th colspan="6"> Datos Actuales </th>';
81 echo '</tr>';
82
83 //creo la tercera fila que contiene fecha y hora, temperatura, humedad y presion atmosferica
84 echo '<tr bgcolor=#198DCF>';
85 echo '<th font-size:12px; width=18% id "Fecha y Hora"> Fecha </th>';
86 echo '<th width=17% id "Temperatura"> Temperatura [°C] </th>';
87 echo '<th width=17% id "Humedad"> Humedad [%] </th>';
88 echo '<th width=17% id "PresionAtmosferica"> Presion Atmosferica [hPa] </th>';
89 echo '<th width=17% id "Viento"> Viento </th>';
90 echo '<th width=17% id "Viento"> Carga de bateria [%] </th>';
91 echo '</tr>';
92
93
94 // apartir de aca creo las filas necesarias para cargar los datos que contiene la tabla en SQL
95 //a continuacion usamos el while para recorrer toda la tabla. La variable extraido contendra todos los valores de la tabla. Es decir la variable consulta se
convertira en un vector y se igualara a extraido. el while lo recorrera hasta recorrer toda la tabla
96
97
98 while ( $extraido = mysqli_fetch_array($consulta) )
99 {
100     echo '<tr align="middle">'; //se crea una nueva fila cada vez que entra al while
101     echo '<td span style="color:white; font-size:30px;" width= 18%>'. $extraido['Fecha']. '</td>'; //entre corchetes se le pasa la columna de la base de
datos
102     echo '<td span style="color:white; font-size:40px;" width= 17%>'. $extraido['Temperatura']. '</td>';
103     echo '<td span style="color:white; font-size:40px;" width= 17%>'. $extraido['Humedad']. '</td>';
104     echo '<td span style="color:white; font-size:40px;" width= 17%>'. $extraido['PresionAtmosferica']. '</td>';
105     echo '<td span style="color:white; font-size:40px;" width= 17%>'. $extraido['Viento']. '</td>';
106     echo '<td span style="color:white; font-size:40px;" width= 17%>'. $extraido['Bateria']. '</td>';
107     echo '</tr>';
108 }
109
110 mysqli_close($conexion); //cierro la conexion con la base de datos
111 echo '</table>';
112
113
114 ?>
115
116 </body>
117 </html>
118
```

Figura N° 74 Tercera parte del archivo index.php

La variable `extraído` irá recibiendo todos los datos de la consulta, es decir que la variable `consulta` se convertirá en un vector y se igualará a la variable `extraído`.

La invocación de `mysqli_fetch_array()` devuelve un registro (una fila) de los resultados obtenidos en una consulta a la tabla de la base de datos. Si la fila ha sido establecida usando `mysqli_data_seek`, nos devolverá la fila que hayamos especificado. Si no usamos `mysqli_data_seek` nos devuelve la primera fila de los resultados.

Si `mysqli_fetch_array` vuelve a ser invocada sin especificar la fila a extraer, devuelve el siguiente registro o fila, siguiendo el orden que tienen los resultados de la consulta.

Decimos que `mysqli_fetch_array` tiene vinculado un puntero o referencia a la fila que debe devolver. Este puntero o referencia puede definirse usando `data seek`, o en caso de no definirse, es inicialmente 0, lo que significa que nos devolverá inicialmente la fila 0. Una vez invocada `mysqli_fetch_array` y devueltos los resultados de la fila 0, el puntero queda apuntando al siguiente registro, es decir, a la fila 1. Este proceso puede repetirse tantas veces como se desee y por cada llamada `mysqli_fetch_array` va avanzando una fila. Cuando ya no

existen más filas en los resultados de la consulta la función devuelve NULL (no hay resultados).

La sentencia `while($extraido=mysqli_fetch_array($consulta))` se emplea para que mientras existan resultados de la consulta, se extraigan los resultados de una fila en forma de array y se almacenen en la variable `extraído`. Usamos el bucle para recorrer toda la tabla de la base de datos, suponiendo que podría llegar a tener más de una fila de datos. Se recorrerá el vector hasta finalizar.

La sentencia `$extraido= mysqli_fetch_array($consulta)` hace que los valores existentes en la fila se introduzcan en un vector cuyos índices en principio pueden ser tanto asociativos (el nombre de la columna) como numéricos (empezando por cero). Por ejemplo, si la primera columna en la tabla de la base de datos es `<<temperatura>>` podemos usar `$extraido['temperatura']` para acceder al valor existente en la fila con la que estemos trabajando para la columna temperatura. Igualmente podríamos usar `$extraido[1]` para referirnos a la primera columna, `$extraido[1]` para referirnos a la segunda columna, `$extraido[2]` para referirnos a la tercera columna y así sucesivamente.

```
while ( $extraido = mysqli_fetch_array($consulta) )
{
echo '<tr align="middle">'; //se crea una nueva fila cada vez que entra al while
echo '<td span style="color:white; font-size:30px;" width= 20%>'.
$extraido['Fecha'].'</td>'; //entre corchetes se le pasa la columna de la base de datos
echo '<td span style="color:white; font-size:40px;" width= 20%>'.
$extraido['Temperatura'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 20%>'.
$extraido['Humedad'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 20%>'.
$extraido['PresionAtmosferica'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 20%>'.
$extraido['Viento'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['Bateria'].'</td>';
echo '</tr>';
}

mysqli_close($conexion); //cierro la conexion con la base de datos
echo '</table>';
?>
```

```
</body>  
</html>
```

Con la sentencia `echo '<tr align="middle">';` cada vez que se inicia nuevamente el bucle `while`, va a crear una nueva fila en la tabla y la va a centrar en el medio. Luego vamos a ir creando las columnas dentro de esa misma fila, e iremos seleccionando qué valores queremos guardar de la fila que tiene la variable `$extraido`, tal como lo hacen las siguientes 6 sentencias que se observan en el código anterior. También aprovechamos y configuramos color de la fuente, tamaño y ancho de la columna:

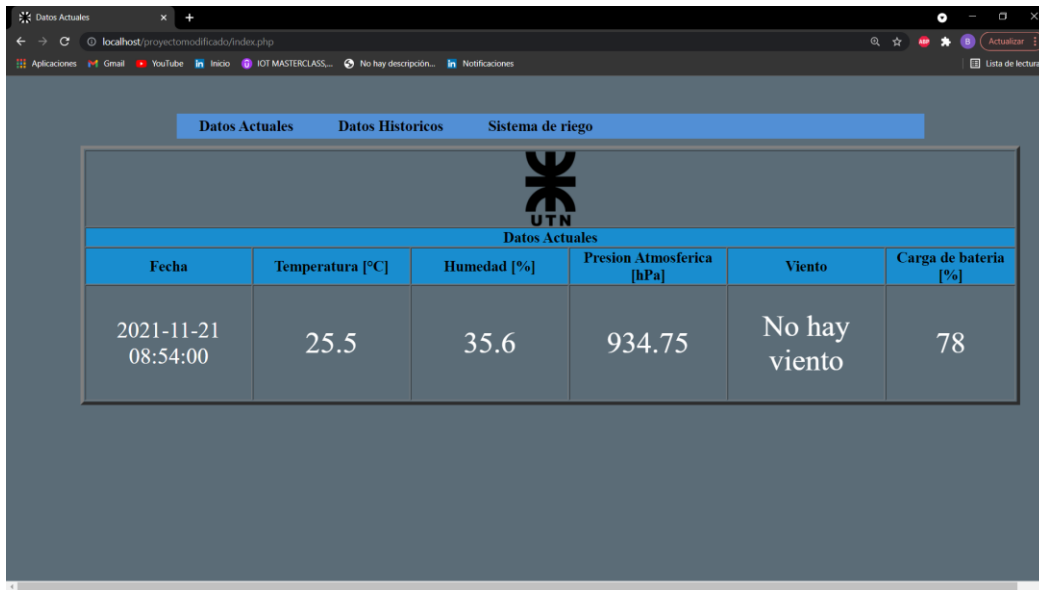
```
echo '<td span style="color:white; font-size:30px;" width= 20%>'.  
$extraido['Fecha'].'</td>'; //entre corchetes se le pasa la columna de la base de datos  
echo '<td span style="color:white; font-size:40px;" width= 20%>'.  
$extraido['Temperatura'].'</td>';  
echo '<td span style="color:white; font-size:40px;" width= 20%>'.  
$extraido['Humedad'].'</td>';  
echo '<td span style="color:white; font-size:40px;" width= 20%>'.  
$extraido['PresionAtmosferica'].'</td>';  
echo '<td span style="color:white; font-size:40px;" width= 20%>'.  
$extraido['Viento'].'</td>';  
echo '<td span style="color:white; font-size:40px;" width= 17%>'.  
$extraido['Bateria'].'</td>';
```

Por último, debemos finalizar la conexión a la base de datos, llamando a la sentencia siguiente:

```
mysqli_close($conexion); //cierro la conexion con la base de datos
```

De esta forma ya quedaría creada la página de inicio cuya imagen podemos ver en la figura que volvemos a mostrar a continuación:

Sistema de monitoreo para heladas



Datos Actuales					
Fecha	Temperatura [°C]	Humedad [%]	Presion Atmosferica [hPa]	Viento	Carga de bateria [%]
2021-11-21 08:54:00	25.5	35.6	934.75	No hay viento	78

Figura N° 75 **Imagen de index.php**

Si hacemos click en la opción “Datos Historicos”, se va a abrir una nueva ventana que resulta ser el archivo auxiliar.html. Este archivo tal como ya se explicó, tiene la función de permitir seleccionar una fecha-hora de inicio y una fecha-hora de finalización, intervalo dentro del cual deseamos ver los datos almacenados. Esa información de fecha-hora, será enviada mediante el método POST al archivo históricos.php el cual procederá a realizar acciones con esa información.

Sistema de monitoreo para heladas

```
C:\xampp\htdocs\proyectormodificado\auxiliar.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
auxiliar.html x
1 <!DOCTYPE html>
2 <html>
3 <head>
4
5 <link rel="shortcut icon" type="image/x-icon" href="Imagenes/Logo.jpg">
6 <title>Datos Historicos</title>
7
8 <!-- a continuacion le digo que las modificaciones las tome del siguiente CCS en cuanto al formato visual-->
9 <link rel="stylesheet" href="css/estilos de tablas.css" type="text/css">
10
11
12 <style type="text/css">
13 {
14     padding: 5px
15 }
16
17 button,input
18 {
19     padding: 5px
20 }
21 label
22 {
23     padding: 20px;
24     padding-right: 20px;
25     padding-left: 20px;
26 }
27
28
29 </style>
30
31
32 </head>
33
34
35 <|-----
36 -----
37 -----
38 -----
39 <|-----
40
```

Figura N° 76 Primera parte del archivo auxiliar.html

No hay mucho que decir de esta primera parte, ya que la función que cumple es la misma que ya fue explicada para index.php, solamente que tiene algunas configuraciones distintas en cuanto a los formatos de diseño que usa.

```
C:\xampp\htdocs\proyectormodificado\auxiliar.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
auxiliar.html x
34
35 <|-----
36 -----
37 -----
38 -----
39 <|-----
40
41 <body>
42 <header>
43 <nav>
44 <ul>
45
46 <form method="POST" action="historicos.php" > <!-- genero un form y luego indico el metodo que usare para enviar los datos, el cual puede
47 ser POST o GET, y le indico a que pagina php voy a enviar esos datos. La envio a historicos.php porque ahí tengo armado el codigo de
48 busqueda en mysql-->
49
50 <small><!-- esto es para establecer la fuente en tamaño small-->
51
52 <!-- A continuacion creo unos input del tipo datetime-local que me permiten crear cada uno un selector de fecha y hora-->
53
54 <label > Desde <input type="datetime-local" name="fecha_min" value= ></label>
55 <label > Hasta <input type="datetime-local" name="fecha_max" value= ></label>
56
57 <small>
58 <button type="submit" width:5px ><b>Buscar Datos</b></button> <!-- el boton tiene que ser del tipo submit para que estos datos
59 seleccionaos sean enviados a historicos.php, si no fuera submit, los datos no se enviarian a ningun lado-->
60
61 </form>
62 </ul>
63 </nav>
64 </header>
65 </body>
66
67 <h2 align="center">Seleccione las fechas dentro de las cuales desea observar los datos almacenados</h2>
68 </html>
```

Figura N° 77 Segunda parte del archivo auxiliar.html

En esta segunda parte del código, se crea un formulario haciendo uso de la siguiente línea:

```
<form method="POST" action="historicos.php">
```

En el proceso de enviar y recibir datos de un formulario existen dos páginas. Tenemos la página que tiene el formulario, que en este caso sería auxiliar.html y la página que lo recibe vendría a ser la página históricos.php.

El formulario tiene un atributo llamado "action" que indica la dirección de la página PHP que recibirá el formulario. Además, si queremos que los datos se envíen por POST, debemos incluir el atributo "method" con el valor "post".

En el formulario tenemos que incluir los campos que necesitamos. El nombre de los campos, definido por el atributo name de la etiqueta, definirá el nombre de la variable que vamos a recibir más tarde en PHP. En las siguientes líneas de código, vemos que se definieron las variables "fecha_min" y "fecha_max" que vendrían a ser los datos enviados mediante POST. En el archivo php deberemos recibir estos datos exactamente con el mismo nombre que se empleó en el archivo html.

Como veremos más adelante en la página históricos.php, el campo de formulario lo vamos a recibir con el array super global \$_POST. Este es un array asociativo que se crea automáticamente por PHP y que está disponible en cualquier lugar del código de la página donde recibimos el formulario. El nombre del campo de formulario es el que usaremos como índice del array \$_POST.

```
<small><!-- esto es para establecer la fuente en tamaño small-->
  <!-- A continuacion creo unos input del tipo datetime-local que me permiten crear
  cada uno un selector de fecha y hora-->

  <label > Desde <input type="datetime-local" name="fecha_min" value= ></label>
  <label > Hasta <input type="datetime-local" name="fecha_max" value= ></label>
  <small>

  <button type="submit" width:5px ><b>Buscar Datos<b></button> <!-- el boton tiene que
  ser del tipo submit para que estos datos seleccionaos sean enviados a historicos.php, si no
  fuera submit, los datos no se enviarian a ningun lado-->

</form>
```

El elemento button, teniendo el valor "submit" en su atributo type, representa un botón que, cuando es presionado envía el formulario al que pertenece. La etiqueta de un botón button es representada por el contenido del elemento. En este caso, el botón es el llamado “buscar datos”, y al ser seleccionado enviará la información al archivo históricos.php, actualizándose la pantalla y apareciendo una tabla con los datos históricos comprendidos entre esas fechas.

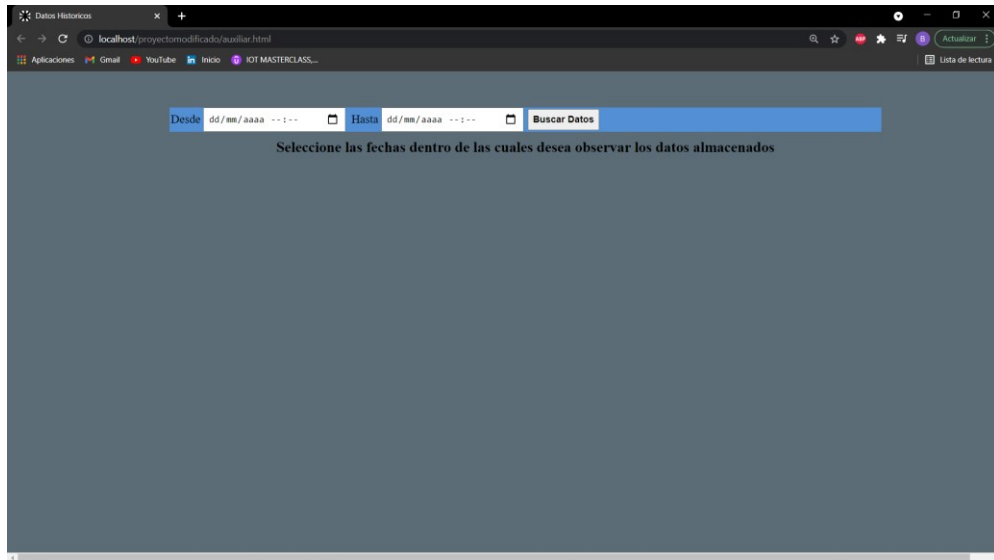
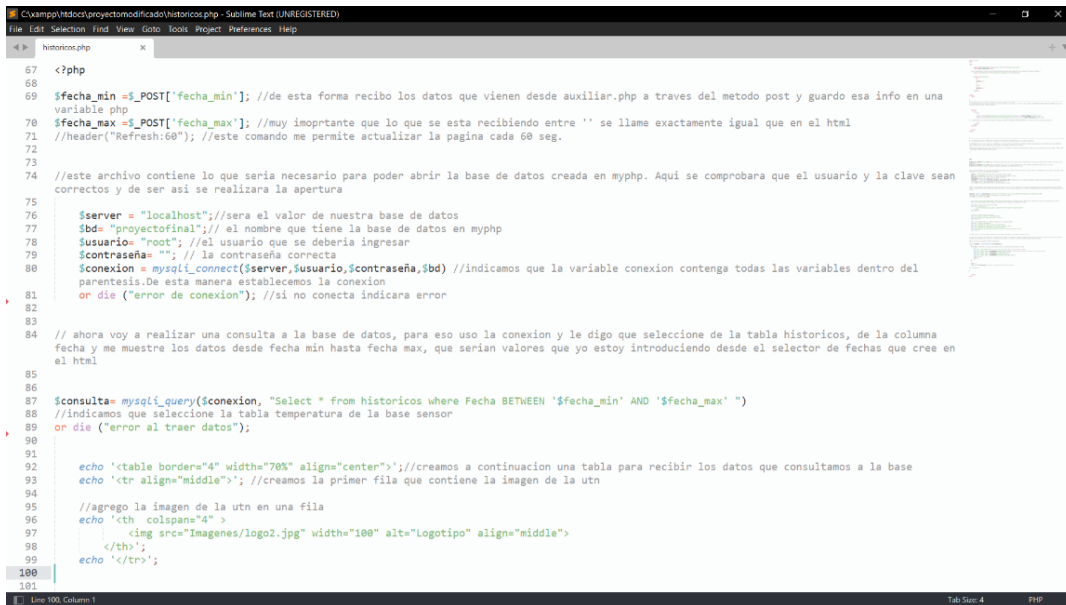


Figura N° 78 Imagen de la ventana auxiliar.html

En el archivo histórico.php, nos encontramos con muchas sentencias similares a las empleadas en index.php, con la salvedad de que acá no voy a mostrar una sola fila, sino que voy a mostrar todo lo que esté comprendido entre dos valores de fecha-hora. Vamos a explicar la parte más importante que es la de código php:

Sistema de monitoreo para heladas



```
67 <?php
68
69 $fecha_min=$_POST['fecha_min']; //de esta forma recibo los datos que vienen desde auxiliar.php a traves del metodo post y guardo esa info en una
variable php
70 $fecha_max=$_POST['fecha_max']; //muy importante que lo que se esta recibiendo entre '' se llame exactamente igual que en el html
71 //header("Refresh:60"); //este comando me permite actualizar la pagina cada 60 seg.
72
73
74 //este archivo contiene lo que seria necesario para poder abrir la base de datos creada en myphp. Aqui se comprobara que el usuario y la clave sean
correctos y de ser asi se realizara la apertura
75
76 $server = "localhost";//sera el valor de nuestra base de datos
77 $bd= "proyectofinal";// el nombre que tiene la base de datos en myphp
78 $usuario= "root"; //el usuario que se deberia ingresar
79 $contraseña= ""; // la contraseña correcta
80 $conexion = mysqli_connect($server,$usuario,$contraseña,$bd) //indicamos que la variable conexion contenga todas las variables dentro del
parentesis.De esta manera establecemos la conexion
or die ("error de conexion"); //si no conecta indicara error
81
82
83
84 // ahora voy a realizar una consulta a la base de datos, para eso uso la conexion y le digo que seleccione de la tabla historicos, de la columna
fecha y me muestra los datos desde fecha min hasta fecha max, que serian valores que yo estoy introduciendo desde el selector de fechas que cree en
el html
85
86
87 $consulta=mysqli_query($conexion, "Select * from historicos where Fecha BETWEEN '$fecha_min' AND '$fecha_max' ")
88 //indicamos que seleccione la tabla temperatura de la base sensor
89 or die ("error al traer datos");
90
91
92 echo '<table border="4" width="70%" align="center">'; //creamos a continuacion una tabla para recibir los datos que consultamos a la base
93 echo '<tr align="middle">'; //creamos la primer fila que contiene la imagen de la utn
94
95 //agrego la imagen de la utn en una fila
96 echo '<th colspan="4" >
97     
98 </th>';
99 echo '</tr>';
100
101
```

Figura N° 79 Primera parte del archivo historicos.php

Lo primero que hacemos es recibir los datos que están llegando desde el formulario de auxiliar.html. Para esto empleamos las siguientes sentencias, las cuales como ya mencionamos deben respetar el formato y sobre todo el nombre de la variable que fue enviada:

```
$fecha_min=$_POST['fecha_min'];
$fecha_max=$_POST['fecha_max'];
```

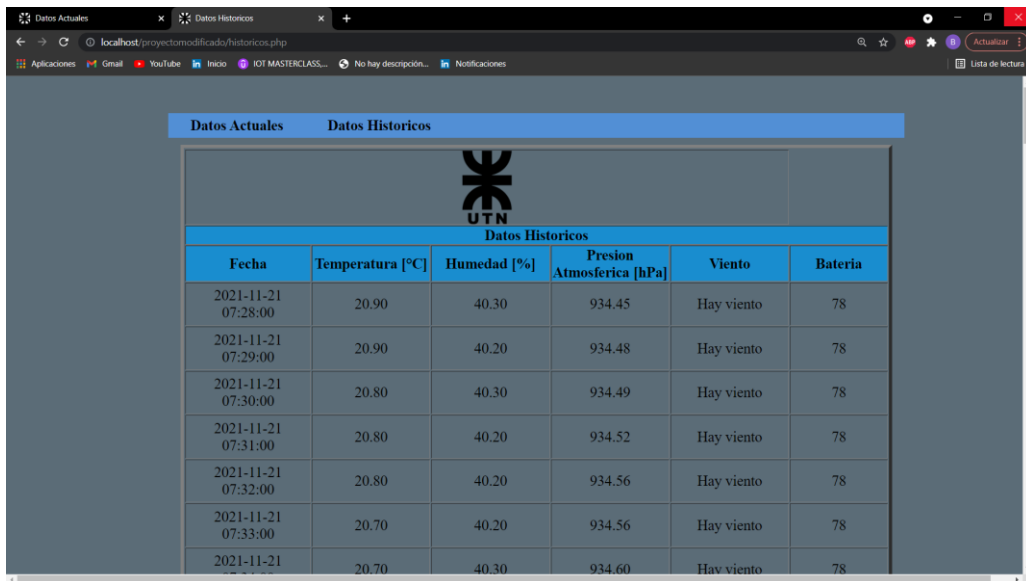
En la variable \$fecha_min vamos a guardar la información que trae 'fecha_min' y en \$fecha_max la información que trae 'fecha_max'. Es importante aclarar que el nombre de la variable no tiene que ser igual al del elemento que recibe. Iniciamos la conexión con la base de datos:

```
$server = "localhost";
$bd= "proyectofinal";
$usuario= "root";
$contraseña= "";
$conexion = mysqli_connect($server,$usuario,$contraseña,$bd)
or die ("error de conexion");
```

Procedemos a realizar una consulta a la base de datos “proyectorfinal”, indicando que seleccione la tabla “historicos”, de la columna fecha y me muestre los datos desde \$fecha_min hasta \$fecha_max, que serían valores que yo estoy introduciendo desde el selector de fechas de auxiliar.html.

```
$consulta= mysqli_query($conexion, "Select * from historicos where Fecha BETWEEN '$fecha_min' AND '$fecha_max' ")  
  
or die ("error al traer datos");
```

El código restante emplea las mismas sentencias ya explicadas cuando tratamos index.php, es decir ya realizando la consulta solo debemos proceder a crear la tabla y empezar a completar filas de datos.

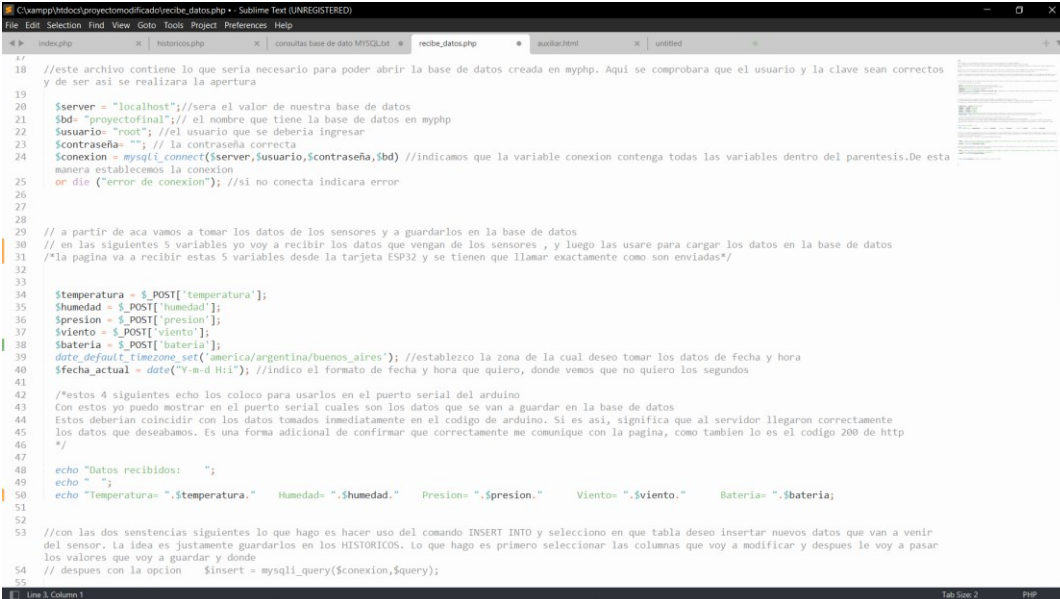


Datos Actuales		Datos Historicos			
UTN					
Datos Historicos					
Fecha	Temperatura [°C]	Humedad [%]	Presion Atmosferica [hPa]	Viento	Bateria
2021-11-21 07:28:00	20.90	40.30	934.45	Hay viento	78
2021-11-21 07:29:00	20.90	40.20	934.48	Hay viento	78
2021-11-21 07:30:00	20.80	40.30	934.49	Hay viento	78
2021-11-21 07:31:00	20.80	40.20	934.52	Hay viento	78
2021-11-21 07:32:00	20.80	40.20	934.56	Hay viento	78
2021-11-21 07:33:00	20.70	40.20	934.56	Hay viento	78
2021-11-21 07:34:00	20.70	40.30	934.60	Hay viento	78

Figura N° 80 Imagen de la ventana historicos.php

Un archivo que cumple una función muy importante es el llamado recibe_datos.php, el cual es quien estará recibiendo los datos que serán enviados desde la tarjeta NodeMCU y además actualizará las tablas alocadas en la base de datos cada 1 minuto. Debemos tener presente que es un archivo que se ejecuta en segundo plano, es decir que no vamos a visualizar en pantalla algún cambio, sino que la tarjeta NodeMCU hará uso de él como si fuera un programa fantasma.

Sistema de monitoreo para heladas



```
18 //este archivo contiene lo que sería necesario para poder abrir la base de datos creada en myphp. Aqui se compraba que el usuario y la clave sean correctos
19 y de ser así se realizara la apertura
20 $server = "localhost";//sera el valor de nuestra base de datos
21 $db = "proyectofinal";// el nombre que tiene la base de datos en myphp
22 $usuario = "root";//el usuario que se debería ingresar
23 $contraseña = ""; // la contraseña correcta
24 $conexion = mysqli_connect($server,$usuario,$contraseña,$db) //indicamos que la variable conexion contenga todas las variables dentro del parentesis.De esta
manera establecemos la conexion
25 or die ("error de conexion"); //si no conecta indicara error
26
27
28
29 // a partir de aca vamos a tomar los datos de los sensores y a guardarlos en la base de datos
30 // en las siguientes 5 variables yo voy a recibir los datos que vengan de los sensores , y luego las usare para cargar los datos en la base de datos
31 /*la pagina va a recibir estas 5 variables desde la tarjeta ESP32 y se tienen que llamar exactamente como son enviadas*/
32
33
34 $temperatura = $_POST['temperatura'];
35 $humedad = $_POST['humedad'];
36 $presion = $_POST['presion'];
37 $viento = $_POST['viento'];
38 $bateria = $_POST['bateria'];
39 date_default_timezone_set('america/argentina/buenos_aires');//establezco la zona de la cual deseo tomar los datos de fecha y hora
40 $fecha_actual = date("Y-m-d H:i");//indico el formato de fecha y hora que quiero, donde vemos que no quiero los segundos
41
42 //estos 4 siguientes echo los coloco para usarlos en el puerto serial del arduino
43 Con estos yo puedo mostrar en el puerto serial cuales son los datos que se van a guardar en la base de datos
44 Estos deberian coincidir con los datos tomados inmediatamente en el codigo de arduino. Si es así, significa que al servidor llegaron correctamente
45 los datos que deseabamos. Es una forma adicional de confirmar que correctamente me comuniqué con la pagina, como tambien lo es el codigo 200 de http
46 */
47
48 echo "Datos recibidos: ";
49 echo " ";
50 echo "Temperatura= ".$temperatura." Humedad= ".$humedad." Presion= ".$presion." Viento= ".$viento." Bateria= ".$bateria;
51
52
53 //con las dos sentencias siguientes lo que hago es hacer uso del comando INSERT INTO y selecciono en que tabla deseo insertar nuevos datos que van a venir
del sensor. La idea es justamente guardarlos en los HISTORICOS. lo que hago es primero selecciono las columnas que voy a modificar y despues le voy a pasar
los valores que voy a guardar y donde
54 // despues con la opcion $insert = mysqli_query($conexion,$query);
55
```

Figura N° 81 Primera parte del archivo recibe_datos.php

En primer lugar, veremos que se realiza la conexión a la base de datos ya explicada anteriormente. Luego se procede a recibir las variables que estarán llegando desde la tarjeta NodeMCU, los cuales están siendo enviados mediante el método POST.

Veremos más adelante en el código que emplea la tarjeta NodeMCU que los datos no son enviados exactamente como se hace en HTML, pero sí debemos tener presente que se reciben de la misma forma y respetando las mismas reglas:

```
$temperatura = $_POST['temperatura'];
$humedad = $_POST['humedad'];
$presion = $_POST['presion'];
$viento = $_POST['viento'];
$bateria = $_POST['bateria'];
date_default_timezone_set('america/argentina/buenos_aires');
$fecha_actual = date("Y-m-d H:i");
```

Además de las primeras 4 variables que están viniendo desde los sensores, también guardamos la fecha y hora actual en la variable \$fecha_actual. La idea es guardar también esa información junto con los datos, para tener un registro de tiempo de ellos. Con la sentencia date_default_timezone_set() vamos a indicar la región que queremos tomar como

referencia para el tiempo. Luego con la sentencia `date()` vamos a indicarle el formato que deseamos, en el cual estamos indicando que vamos a guardar año/mes/día- hora/minutos.

Ahora que ya tenemos la información, podemos proceder a guardarla en las tablas que deseamos. Para esto, procedemos a hacer uso del comando `INSERT INTO`, al cual debemos indicarle la tabla en la cual deseamos insertar los datos, en que columnas y que valores vamos a guardar:

```
$query = "INSERT INTO historicos(idDevice, Temperatura, Humedad, PresionAtmosferica, Fecha,
Viento,Bateria)VALUES ('tarjeta1','$temperatura','$humedad','$presion','$fecha_actual','$viento',
'$bateria')";
$insert = mysqli_query($conexion,$query);
```

Ahora procedo a con la sentencia `UPDATE` lo que hago es actualizar los valores que están en la tabla temperatura. Debo indicarle que tabla deseo actualizar y los valores de qué columna. Por último, finalizamos la conexión con la base de datos.

```
$query = "UPDATE sensores SET Temperatura = '$temperatura', Humedad = '$humedad',
PresionAtmosferica= '$presion', Fecha = '$fecha_actual' ,Viento = '$viento', Bateria =
'$bateria' WHERE idDevice ='tarjeta1'";
$insert = mysqli_query($conexion,$query);
mysqli_close($conexion); //cierro la conexion con la base de datos
?>
```

```

37 $viento = $_POST['viento'];
38 $bateria = $_POST['bateria'];
39 date_default_timezone_set('america/argentina/buenos_aires'); //establezco la zona de la cual deseo tomar los datos de fecha y hora
40 $fecha_actual = date("Y-m-d H:i"); //indico el formato de fecha y hora que quiero, donde vemos que no quiero los segundos
41
42 /*estos 4 siguientes echo los coloco para usarlos en el puerto serial del arduino
43 Con estos yo puedo mostrar en el puerto serial cuales son los datos que se van a guardar en la base de datos
44 Estos deberian coincidir con los datos tomados inmediatamente en el codigo de arduino. Si es asi, significa que al servidor llegaron correctamente
45 los datos que deseabamos. Es una forma adicional de confirmar que correctamente me comuniqué con la pagina, como tambien lo es el codigo 200 de http
46 */
47
48 echo "Datos recibidos: ";
49 echo " ";
50 echo "Temperatura=" ."$temperatura." Humedad=" ."$humedad." Presion=" ."$presion." Viento=" ."$viento." Bateria=" ."$bateria";
51
52
53 //con las dos sentencias siguientes lo que hago es hacer uso del comando INSERT INTO y selecciono en que tabla deseo insertar nuevos datos que van a venir
del sensor. La idea es justamente guardarlos en los HISTORICOS. Lo que hago es primero seleccionar las columnas que voy a modificar y despues le voy a pasar
los valores que voy a guardar y donde
54 // despues con la opcion $insert = mysqli_query($conexion,$query);
55
56
57 $query = "INSERT INTO historicos(idDevice, Temperatura, Humedad, PresionAtmosferica, Fecha, Viento, Bateria) VALUES('tarjeta1','$temperatura','$humedad','$
presion','$fecha_actual','$viento','$bateria')";
58 $insert = mysqli_query($conexion,$query);
59
60 //con la siguiente sentencia lo que hago es actualizar la tabla sensores, la cual resulta ser lo que se va a mostrar como datos actuales y que es lo que yo
voy a estar monitoreando como valores en tiempo real
61
62 $query = "UPDATE sensores SET Temperatura = '$temperatura', Humedad = '$humedad', PresionAtmosferica= '$presion', Fecha = '$fecha_actual' ,Viento = '$viento'
, Bateria = '$bateria' WHERE idDevice ='tarjeta1'";
63 $insert = mysqli_query($conexion,$query);
64
65 mysqli_close($conexion); //cierro la conexion con la base de datos
66
67
68 ?>
```

Figura N° 82

Segunda parte del archivo
recibe_datos.php

Sistema de monitoreo para heladas

CAPÍTULO 4

OBTENCIÓN Y TRANSMISIÓN DE INFORMACIÓN

5 Introducción al capítulo:

En el presente capítulo, nos encontramos con el desarrollo de código empleado en las tarjetas NodeMCU. A partir de ellos controlaremos los sensores y módulos empleados en este proyecto. Se verá paso a paso cómo fueron siendo probados los sensores y como se fue logrando la comunicación con el servidor

6 Configuración de tarjeta NodeMCU

Para poder establecer comunicación con la tarjeta, necesitamos realizar una serie de configuraciones. De esta manera, al conectar la tarjeta al puerto USB de una computadora, esta será reconocida por el IDE de Arduino. Si no se realizarán estos pasos, no será posible realizar la carga de código, debido a que nunca será encontrada en los puertos de comunicación.

6.1 IDE de Arduino

En este apartado se realiza una breve introducción de las diferentes partes del software libre Arduino.

Para poder comenzar debemos descargar el software de Arduino, este se puede obtener la página oficial de Arduino. Una vez descargado y ejecutado, al abrirlo aparecerá en nuestro ordenador la siguiente IDE (Integrated Development Environment), lo que se conoce como entorno de desarrollo integrado. Es la herramienta que permite escribir y editar el programa que ejecuta el microcontrolador.

Sistema de monitoreo para heladas



Figura N° 83 IDE de Arduino

En la IDE de Arduino se pueden distinguir cinco zonas distintas que podemos ver en la siguiente imagen. La zona 1 es la barra de menús, zona 2 es la barra de botones, zona 3 es el editor de código, zona 4 es la barra de mensajes y zona 5 es la barra de estado.

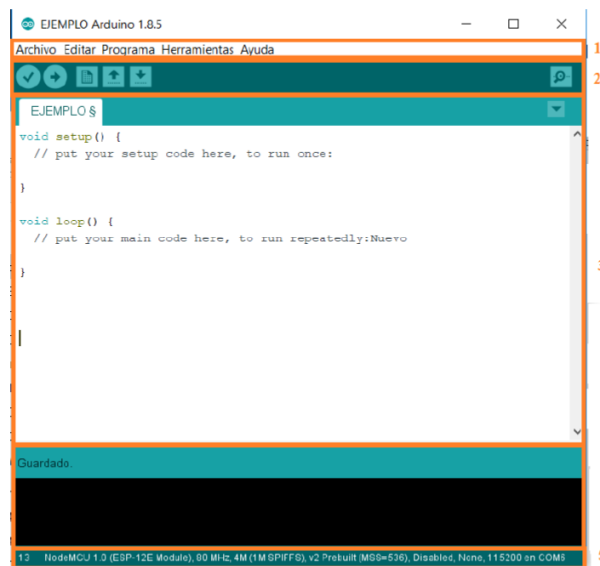


Figura N° 84 Zonas de trabajo de Arduino

- La barra de menús cuenta con 5 desplegables distintos que son archivo, editar, programa, herramientas y ayuda. La pestaña de archivo tiene funciones como las de nuevo archivo, abrir proyectos, ejemplos que vienen definidos de serie, preferencias donde podremos modificar según interese como por ejemplo el idioma, la fuente, el tamaño de la fuente. El menú editar permite acciones como cortar/pegar, hacer / deshacer, comentar / descomentar. El apartado de programa permite las acciones de compilar, subir, incluir librerías y añadir ficheros. El apartado de herramientas permite dar autoformato al programa abrir el monitor y puerto serie y nos permite seleccionar la placa con la cual deseamos trabajar. Por último, el menú de ayuda permite el acceso a la página oficial de Arduino, donde se pueden consultar tutoriales, artículos y ejemplos de ayuda.
- La zona 2 o zona de botones, consta de seis botones distintos:



Figura N° 85 **Zona 2 IDE de Arduino**

El botón con la etiqueta 1 permite compilar el programa, es decir, comprueba que no haya errores en el código. El botón con la etiqueta 2 es para cargar el programa al microcontrolador (antes de subir el programa al microcontrolador vuelve a compilar). El botón 3 crea un nuevo programa en blanco. El botón 4 permite abrir cualquier programa o ejemplo que tengamos guardado previamente. El botón 5 es para guardar el archivo y por último el Monitor serie que nos permite ver los datos e información cuando indiquemos en el programa.

- La zona 3 o editor de código es la zona donde se escriben los programas.
- La zona 4 es la zona de barra de mensajes, donde aparece el estado del programa si ha sido guardado, si está compilando o si se ha subido al microcontrolador.

- La zona 5 barra de estado muestra la línea en la que se encuentra el cursor escribiendo, la placa que se está utilizando y el puerto serie que en el que se está mostrando la información.

6.2 Configuración de NodeMCU:

Para preparar el entorno de Arduino para el uso del módulo NodeMCU debemos seguir los siguientes pasos:

- **Paso 1:** Debemos añadir la placa NodeMCU. Al instalar el IDE de Arduino, si buscamos la placa NodeMCU no aparecerá entre las disponibles, es por esto que debemos instalar las placas adicionales que cuentan con el módulo ESP32. Para esto debemos ir a la pestaña de Archivo, buscar en Preferencias, y en gestor de Urls adicionales de Tarjetas añadimos la siguiente dirección y le damos a aceptar:
 - https://dl.espressif.com/dl/package_esp32_index.json
- **Paso 2:** Debemos añadir los drivers del ESP32. El fabricante nos indica el nombre del driver que debemos instalar, que en este caso vemos que solicita instalar el driver CP2102. Este driver lo podemos encontrar en el siguiente enlace:
 - <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>
- **Paso 3:** Seleccionar la placa con la que se vaya a trabajar:

Sistema de monitoreo para heladas

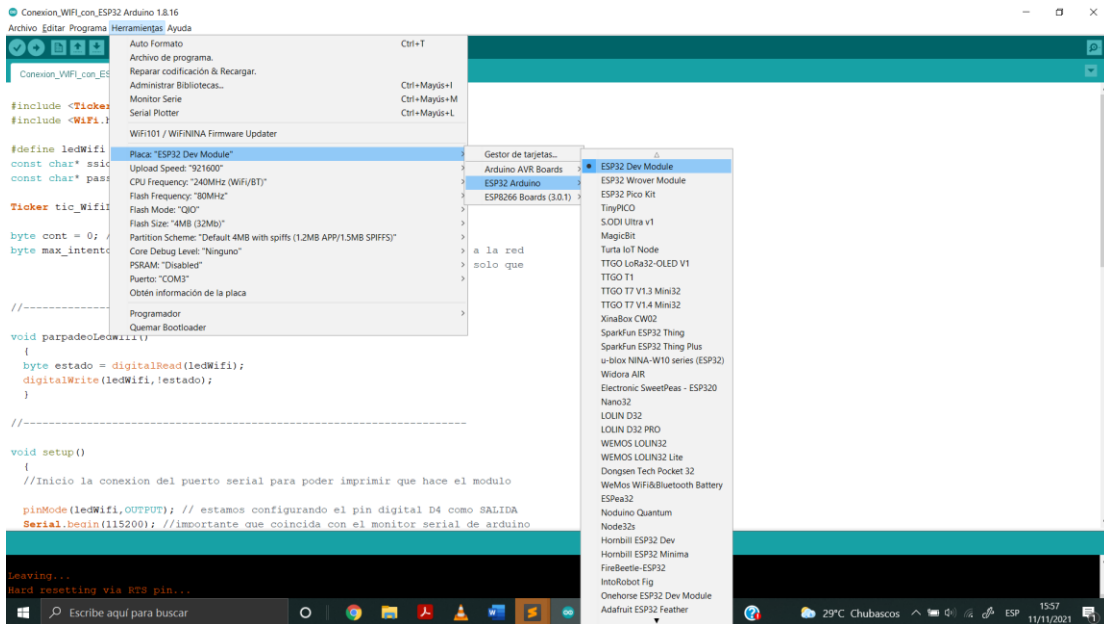


Figura N° 86 Cómo seleccionar la tarjeta de trabajo.

Para verificar que esté todo bien, procedimos a primero crear el código que me va a permitir conectarme a una red wifi con la NodeMCU. Comenzaremos analizando las partes más importantes de este código que luego adjuntamos al final del informe.

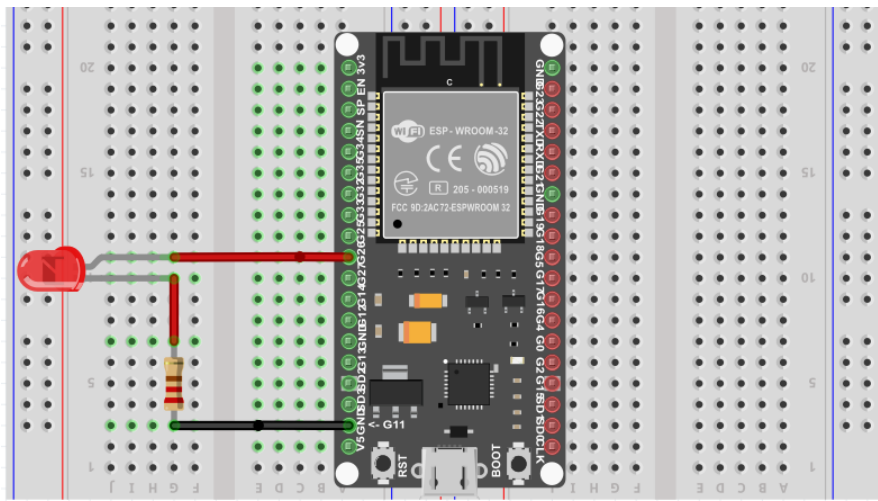


Figura N° 87 Esquemático de conexión empleado para el led indicador

En primer lugar, estamos incluyendo la librería WiFi. Esta librería proporciona las rutinas específicas WiFi de ESP32 a las que llamamos para conectarse a la red. También agregamos la librería Ticker la cual nos permite llamar a funciones repetidas cada cierto periodo de tiempo.

Luego definimos variables globales que serán empleadas en el desarrollo del programa. Definimos que el pin digital 26 este asignado a la etiqueta ledWifi, ya que la tarjeta NodeMCU tiene un led conectado a ese pin. Vamos a declarar dos variables globales del tipo String llamadas ssid y password, las cuales contendrán el nombre de la red a la cual deseo conectarme y su contraseña.

Luego declaramos un objeto de la clase Ticker llamado tic_wifiLed, con el cual llamaremos a funciones de esa librería. Por último, declaramos dos variables del tipo byte ya que ocupan menos memoria que una variable del tipo Int y cumplen la misma función.

```
#include <Ticker.h>
#include <WiFi.h>

#define ledWifi 26 //La tarjeta NodeMCU tiene un led conectado a la salida digital 26
const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red
Ticker tic_WifiLed;
    byte cont = 0; //ire contando los intentos con este contador
byte max_intentos = 20; //establezco un maximo de intentos de conectarse a la red
    //las declaro de tipo byte porque es como un INT solo que
    //en vez de ocupar 4 byte va a ocupar 1 byte
```

Luego tenemos una función llamada parpadeoLedWifi() la cual simplemente enciende o apaga un led en función a su estado anterior, tal que este parpadee.

```
void parpadeoLedWifi()
{
    byte estado = digitalRead(ledWifi);
    digitalWrite(ledWifi,!estado);
}
```


Ahora procedemos a realizar la conexión del puerto serial para poder imprimir qué acciones o respuestas entrega la NodeMCU. Asignamos que la velocidad de transmisión sea de 115200 baudios, e iniciamos el led apagado. Luego haciendo uso del objeto ticker creado, llamamos a la función attach() y le indicamos que queremos que cada 200mseg se llame a la función parpadeLedWifi()

```
void setup()
{
  //Inicio la conexion del puerto serial para poder imprimir que hace el modulo

  pinMode(ledWifi,OUTPUT); // estamos configurando el pin digital D4 como SALIDA
  Serial.begin(115200); //importante que coincida con el monitor serial de arduino
  Serial.println("\n"); // Es un salto de linea "ENTER"
  tic_WifiLed.attach(0.2,parpadeoLedWifi); //desde aca va a empezar a parpadear el led
  mientras este intentando conectarse a la red
```

A partir de acá comenzamos con la conexión a la red. En primer lugar, abrimos la conexión wifi llamando a la función WiFi.begin() y le pasamos como variables el usuario y contraseña de la red

```
WiFi.begin(ssid, password); //abrimos la conexion wifi pasandole el usuario y
contraseña
de la red
```

El proceso de conexión puede demorar unos segundos, así que comprobamos que esto se complete en el siguiente ciclo:

```
while(WiFi.status() != WL_CONNECTED and cont < max_intentos) //mientras no se conecte ira
aumentando un contador hasta 20 intentos

{
  cont++;
  delay(500);
  Serial.print(".");
}
```

El bucle while() seguirá en bucle mientras WiFi.status() sea distinto a WL_CONNECTED y se siga cumpliendo que cont sea menor a max_intentos. Vemos que para no crear un bucle infinito en el caso de que no podamos concretar la conexión, iremos aumentando un contador cada 500mSeg y fijamos un máximo de intentos de conexión con la variable max_intentos. De esta manera permitimos que intente conectarse solo hasta que se deje de cumplir la condición cont < max_intentos.

Luego necesitamos saber si pudimos lograr o no la conexión, por lo cual haremos uso del monitor serial. Si cont no llegó a ser mayor que max_intentos, eso significa que se conectó la placa a la red wifi, por lo cual podemos imprimir información de la ip que le asignó el router a la tarjeta, la dirección mac y el nombre de la red. Por otro lado, si cont llegó a superar a max_intentos, eso significa que no llegó a conectarse dentro del tiempo límite de espera que fijamos, por lo cual imprimimos en pantalla que la conexión no pudo concretarse.

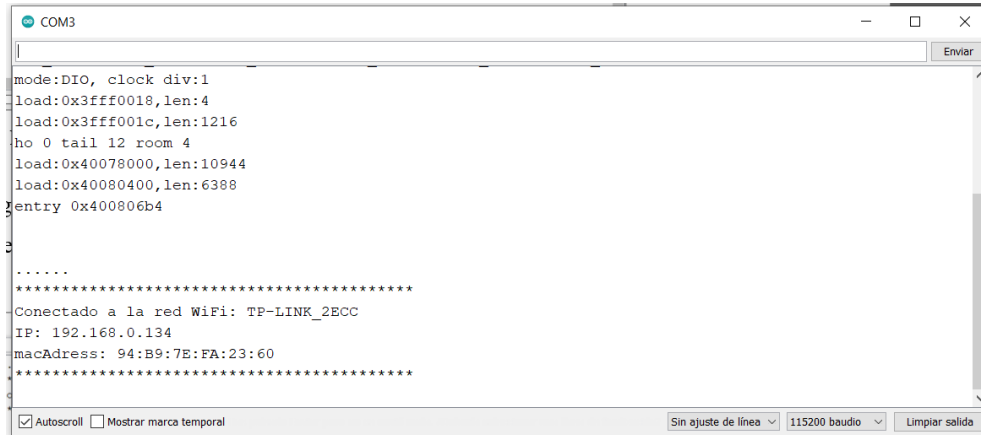
```
Serial.println(""); //salto de linea

if(cont < max_intentos) //si entra quiere decir que se conecto
{
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
  Serial.println("*****");
  Serial.print("Error de conexion");
  Serial.println("*****");
}
```

Una vez que termino de intentar conectarse a la red wifi, debemos apagar el led, ya que la idea de hacerlo parpadear es solamente para mostrar que está en esa etapa de intento de conexión. Debemos tener presente que los leds de la tarjeta están negados, por lo cual un cero encendería el led y un nivel alto lo apagaría.

```
tic_WifiLed.detach();//una vez que termina de intentar conectarse, terminamos el proceso
del led
digitalWrite(ledWifi,HIGH); // apagamos el led por si llego a quedar prendido
//recordar que los led de la tarjeta estan NEGADOS por lo que
se prenden con un CERO y se apagan con UNO
}
void loop()
{
```

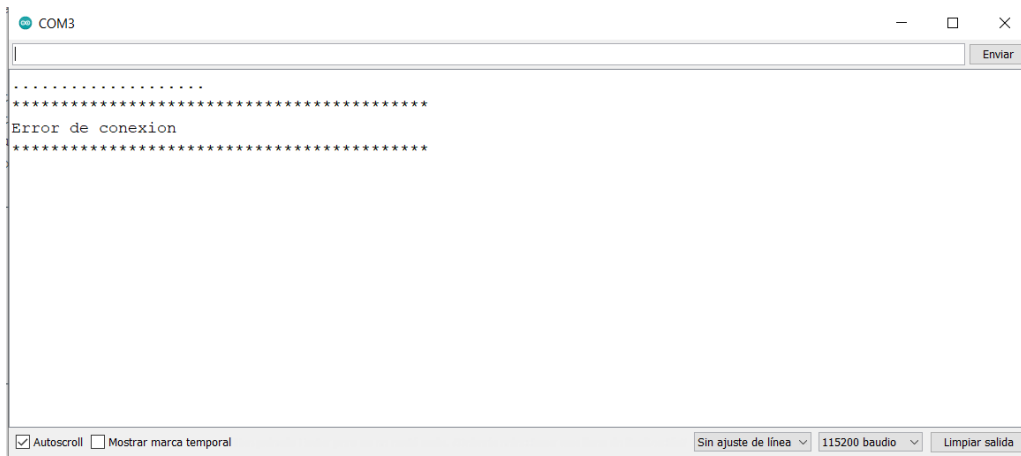
Con el código terminado, procedemos a subirlo a la placa para proceder a probarla, obteniendo un resultado positivo en la conexión a una red wifi, tal como podemos ver en la siguiente imagen:



```
COM3
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
.....
*****
Conectado a la red WiFi: TP-LINK_2ECC
IP: 192.168.0.134
macAddress: 94:B9:7E:FA:23:60
*****
 Autoscroll  Mostrar marca temporal
Sin ajuste de línea 115200 baudio Limpiar salida
```

Figura N° 88 Puerto serial mostrando la correcta conexión de la NodeMCU

En el siguiente caso vemos la respuesta que entrega el puerto serial al no poder concretar la conexión dentro del tiempo permitido en el código.



```
COM3
.....
*****
Error de conexión
*****
 Autoscroll  Mostrar marca temporal
Sin ajuste de línea 115200 baudio Limpiar salida
```

Figura N° 89 Error de conexión debido a que se colocó mal la contraseña

6.3 Desarrollo de códigos para el manejo de sensores y envío de información:

Ahora que confirmamos que la tarjeta puede programarse correctamente, podemos empezar establecer la comunicación con los sensores. Iremos viendo un sensor o módulo a la vez para poder explicarlo mejor y luego simplemente se trasladará todo a un solo código que será el finalmente empleado.

6.3.1 Desarrollo de código para DHT22

Primero comenzaremos con el sensor dht22, del cual conectaremos su pin de datos al pin D1 de la tarjeta NodeMCU. Para poder obtener los datos del sensor de forma sencilla, utilizaremos la librería DHT, la cual se encargará de enviar la señal de consulta y proporcionar la información solicitada en un determinado momento.

Vemos que la alimentación la tomaremos desde la tarjeta NodeMCU y el pin de datos lo colocaremos en GPIO 32.

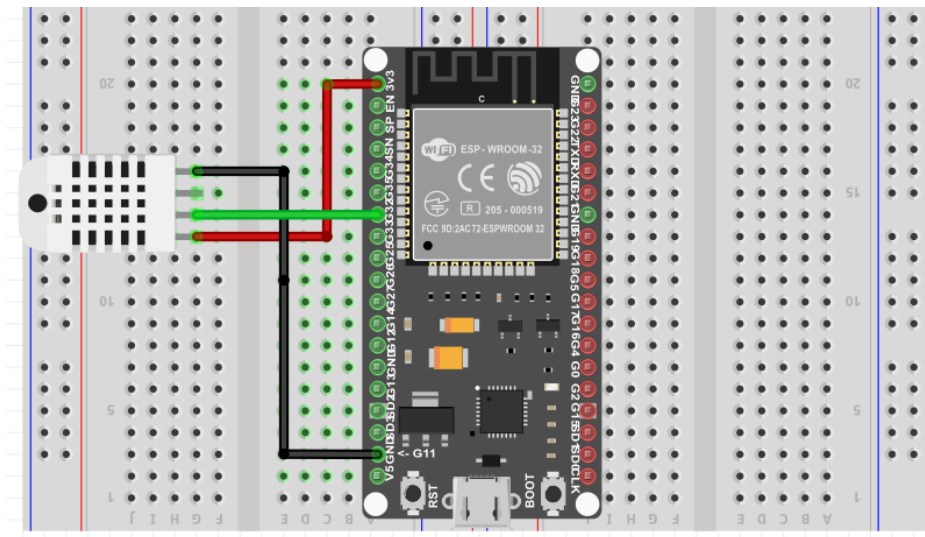


Figura N° 90 Esquemático de conexión empleado con sensor DHT22

Una vez incluidas las librerías, procedemos a crear un objeto que llamaremos dht de la clase DHT, y le indicaremos que se usará el pin D1 para recibir los datos del sensor y además se le indica que el sensor a emplear será el DHT22, debido a que esta librería también funciona con el sensor DHT11.

Luego creamos dos variables globales del tipo float para guardar los datos de temperatura y humedad, los cuales tendrán parte entera y decimal.

```
#include <DHT.h>
#include <DHT_U.h>
DHT dht(32 ,DHT22); //indico que los datos del sensor se conectan al 32 de la placa
                    // y que usare el sensor DHT11
float temperatura,humedad; //declaro dos variables globales
```

Inicializamos el puerto serial para poder visualizar si logramos una correcta comunicación con el sensor y también inicializamos la comunicación con el sensor.

```
void setup()
{
  Serial.begin(115200);
  dht.begin();
}
```

Ahora procedemos a obtener el valor de temperatura al llamar con el objeto dht a la función readTemperature(), la cual devolverá el valor de temperatura que entregue el sensor en ese momento, y lo guardará en la variable temperatura. Lo mismo ocurre para la humedad pero haciendo uso de la función readHumidity() y guardará el valor que devuelve en la variable humedad. Obtenidos estos datos, vamos a imprimir la información obtenida, haciendo uso del puerto serie.

Por último, vamos a crear un delay de 5 seg para que el bucle se repita cada ese tiempo. Es decir que cada 5 segundos se le van a pedir datos al sensor de temperatura y humedad y se van a mostrar en pantalla.

```
void loop()
{
  temperatura = dht.readTemperature();
  humedad = dht.readHumidity();
  Serial.println("Temperatura: " + String(temperatura) + "Humedad: " + String(humedad));
```

```
delay(5000);  
}
```

Se procede a cargar el código en la placa NodeMCU y a abrir el puerto serial para observar qué información me proporciona el sensor y si funciona correctamente el código:

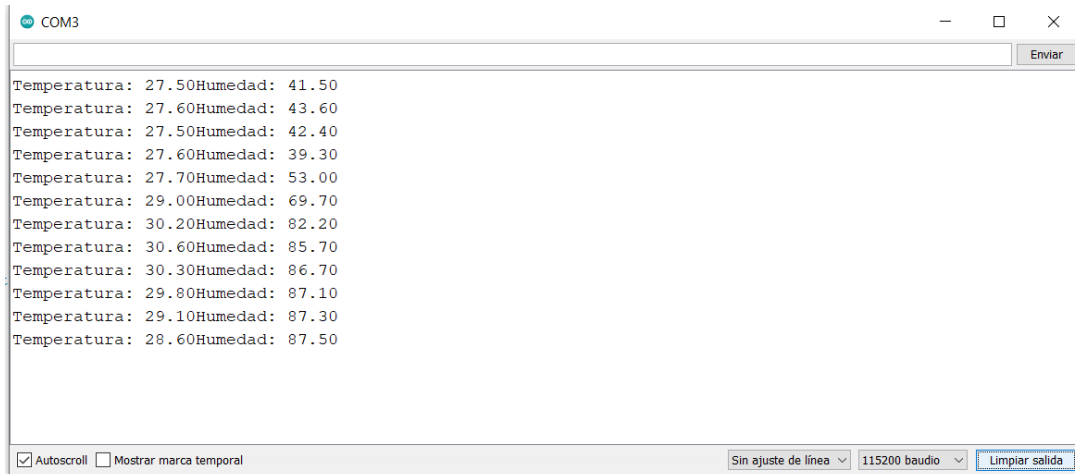


Figura N° 91 Puerto serial mostrando la respuesta del sensor DHT22

6.3.2 Desarrollo de código para BMP280

Ahora desarrollaremos el código para poder manejar el sensor BMP280, del cual obtendremos datos de presión atmosférica y temperatura.

En la siguiente imagen podemos ver cuál fue la conexión empleada para la prueba de este sensor. Vemos que se alimenta el sensor a partir de los pines 3,3V y GND de la placa, y se emplean los pines GPIO 22 para SCL y GPIO 21 para SDA.

Sistema de monitoreo para heladas

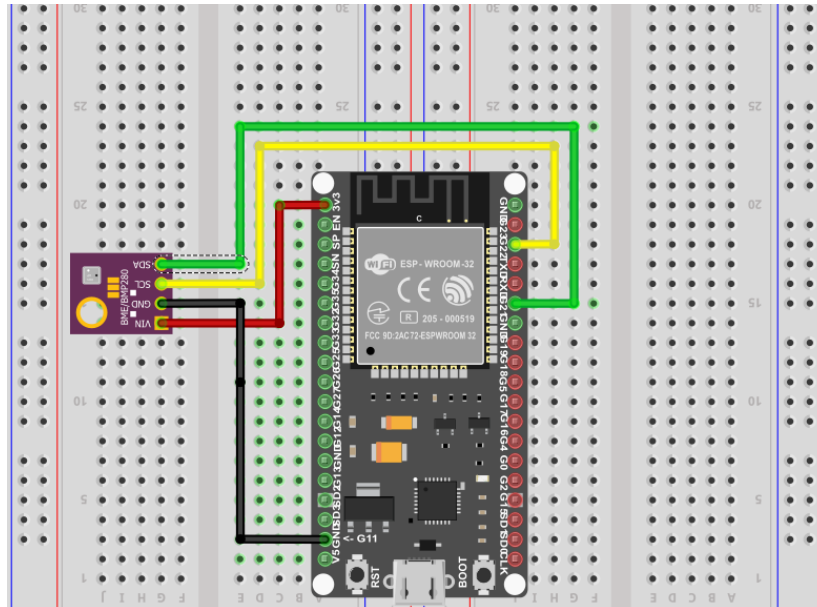


Figura N° 92 Esquemático de conexión empleado para BMP280

En primer lugar, debemos incluir las librerías que nos permitirán comunicarnos con el sensor. La librería Wire.h permite comunicarse a la NodeMCU a través de I2C, ya sea como maestro a otros dispositivos o como esclavo recibiendo peticiones y respondiendo datos. También se agregan 2 librerías de adafruit que me permitirán emplear este sensor.

```
#include <Wire.h> // se incorpora para poder trabajar con I2C
#include <Adafruit_Sensor.h> // estas dos librerías son para emplear el sensor
#include <Adafruit_BMP280.h>
```

Procedemos a crear un objeto de la clase Adafruit_BMP280 que llamaremos bmp. Luego creamos dos variables globales del tipo float que llamaremos presión y temperatura, donde almacenaremos la información que nos enviará el sensor.

```
Adafruit_BMP280 bmp; // creamos un objeto de esa clase
float temperatura; // variables donde guardaremos los datos
float presion;
```

Ahora inicializamos el puerto serial, configurando la velocidad de transmisión en 115200 baudios. Luego inicializamos la comunicación con el sensor a través de la sentencia `bmp.begin()` y le pasamos la dirección por la cual el bus I2C se comunicara con el sensor, que para el `bmp280` es `0x76`. Si no se puede establecer la conexión con el sensor, la sentencia anterior devolverá un `false` y entraremos en el `if` declarado, mostrando en pantalla que no está siendo posible conectarse con el sensor. Luego con una sentencia `while(1)` hacemos que permanezca dentro del `if` hasta que si llegara a concretar comunicación salga de él.

```
void setup()
{
  Serial.begin(115200);
  Serial.println("Iniciando:");
  if(!bmp.begin(0x76))
  {
    //vamos a comprobar si podemos comunicarnos o no con el sensor
    Serial.println("sensor de presion BMP280 no encontrado");
    while(1); //permanecera aca
  }
}
```

Procedemos a leer la temperatura llamando invocando a la sentencia `readTemperature()` y la presión invocando a `readPressure()`, guardando los valores obtenidos en las variables globales `temperatura` y `presión` respectivamente. En el caso de la presión, dividimos el valor obtenido por 100 para pasar de Pa a hPa. Luego simplemente mostramos estos valores en pantalla y por último colocamos un `delay(5000)` para que el bucle se repita cada 5 Seg, es decir que cada 5 segundos se obtendrán nuevos valores de presión y temperatura.

```
void loop()
{
  temperatura = bmp.readTemperature();
  presion = bmp.readPressure()/100; // asi como esta obenemos el dato en hPa
  Serial.println(" Temperatura: " + String(temperatura) + "°C ");
  Serial.println(" Presion: " + String(presion) + "Pa ");
  Serial.println("*****");
  delay(5000);
}
```


Procedemos a cargar el código en la tarjeta NodeMCU y vemos que logramos una correcta conexión y respuesta del sensor BMP280:

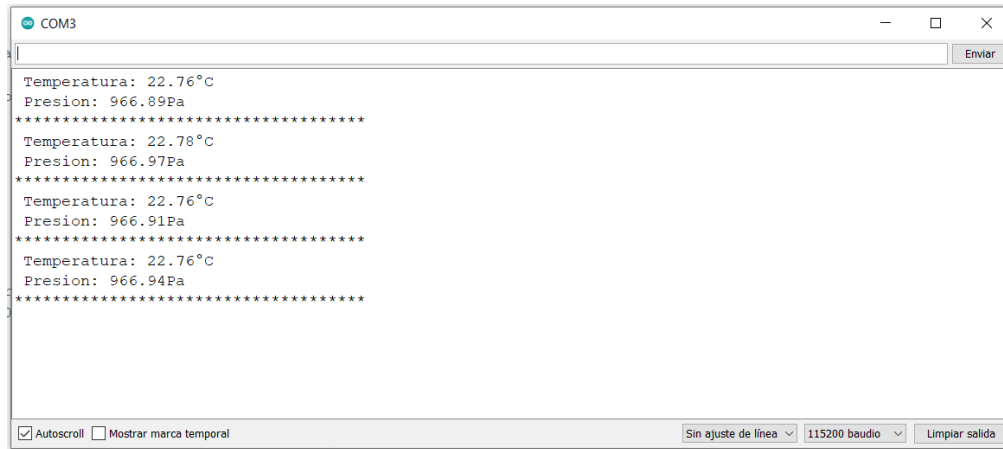


Figura N° 93 Puerto serial mostrando la respuesta del sensor BMP280

6.3.3 Desarrollo de código para ADXL345

Ahora se procederá a realizar el código que nos permitirá comunicarnos con el módulo acelerómetro ADXL345. En la siguiente imagen podemos ver cuál fue la conexión empleada para la prueba de este módulo. Vemos que se alimenta el sensor a partir de los pines 3,3V y GND de la placa, y se emplean los pines GPIO 22 para SCL y GPIO 21 para SDA.

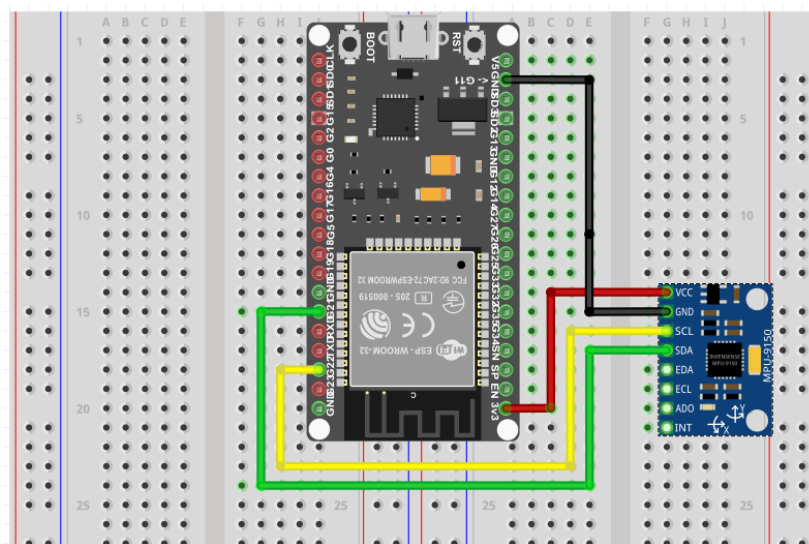


Figura N° 94 **Esquemático de conexión empleado con ADXL345**

En primer lugar, como el módulo se comunica mediante protocolo I2C, necesitaremos incluir la librería Wire.h. Además, incluimos las librerías de adafruit para poder leer de forma sencilla y rápida la información que nos entrega el acelerómetro.

```
//en la NodeMCU el pin d1 es SCL y el pin d2 es SDA
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
```

Dentro de las variables globales, incluimos una variable boolean que emplearemos como indicador, para poder detectar si hay presencia de viento o no. Este tipo de variable solo puede tomar dos estados (False o true), por lo que de esa forma podremos diferenciar luego si se detecta circulación de viento o no, en función de la posición que tome el acelerómetro.

Procedemos a crear un objeto de la clase Adafruit_ADXL345 el cual llamaremos ADXL345 y que emplearemos para comunicarnos con el módulo y hacer llamadas a las funciones de lectura.

Por último, procedemos a crear 3 variables del tipo Int para almacenar los valores que entregará el acelerómetro en cada eje.

```
boolean viento=false; // con esta variable bandera voy a ver si hay o no viento
Adafruit_ADXL345_Unified ADXL345 = Adafruit_ADXL345_Unified();
int x=0,y=0 ,z=0; // guardare los valores de cada eje para ver si hay presencia de viento o
no, tomando un valor estatico de x=0 y=0 y Z=-10 como referencia de que no hay viento
```

Inicializamos la comunicación con el puerto serial para poder ver cómo responde el módulo, indicando que emplearemos una velocidad de transmisión de 115200 baudios.

Luego inicializamos la comunicación con el módulo acelerómetro a través de la sentencia ADXL345.begin() y la incluimos dentro de un if. Si la conexión con el módulo no se concreta, entonces mostraremos en pantalla que la comunicación no pudo ser iniciada y nos mantendremos en esa parte del bucle haciendo uso de la sentencia while(1) hasta que pueda comunicarse con el módulo y salga.

```
void setup()
{
  Serial.begin(115200);

  if(!ADXL345.begin())
  {
    Serial.println("El sensor no inicio");
    while(1);
  }
}
```

Ahora con la sentencia `sensors_event_t` crearemos una variable llamada evento, con la cual haremos los llamados a función para obtener el valor de aceleración de cada eje y luego lo mostraremos en pantalla cada un segundo.

```
void loop()
{
  sensors_event_t evento;
  ADXL345.getEvent(&evento);
  x = evento.acceleration.x;
  y = evento.acceleration.y;
  z = evento.acceleration.z;
  Serial.print("Eje x = " + String(evento.acceleration.x));
  Serial.print("\t\t\tEje y = " + String(evento.acceleration.y));
  Serial.println("\t\t\tEje z = " + String(evento.acceleration.z));
  delay(1000);
}
```

Tal como se observa en la imagen anterior, el acelerómetro estando en la posición que será empleado, entregará en reposo los siguientes valores $X=0$, $Y=0$ y $Z=0$ que serán tomados como referencia para establecer que no hay viento. De esta forma, estableceremos condiciones para las cuales dependiendo de los valores que entreguen los ejes X, Y y Z en función a cómo se esté posicionando el módulo debido al desplazamiento que cause el viento, determinemos si hay presencia de viento o no. Si decidimos que hay presencia de viento, entonces a la variable booleana llamada viento le asignamos el valor True. En caso de establecer que no hay viento, le asignamos el valor False.

```
//comparamos valores para ver si hay o no presencia de viento
// la referencia estatica es x=0 y=0 z= -10
if( -1 < x && x < 1)
```

Sistema de monitoreo para heladas

```
{
  if( -1 < y && y < 1)
  {
    viento = false;
  }
}

if( x > 5 || x < -5)
{
  viento= true;
}
if( y > 5 || y < -5)
{
  viento= true;
}
if(z > -9 && z < -4)
{
  viento= true;
}

if(viento == true)Serial.println("Hay viento");
if(viento == false)Serial.println("No Hay viento");
}
```

En la siguiente figura podemos ver que no se puede establecer comunicación con el módulo:

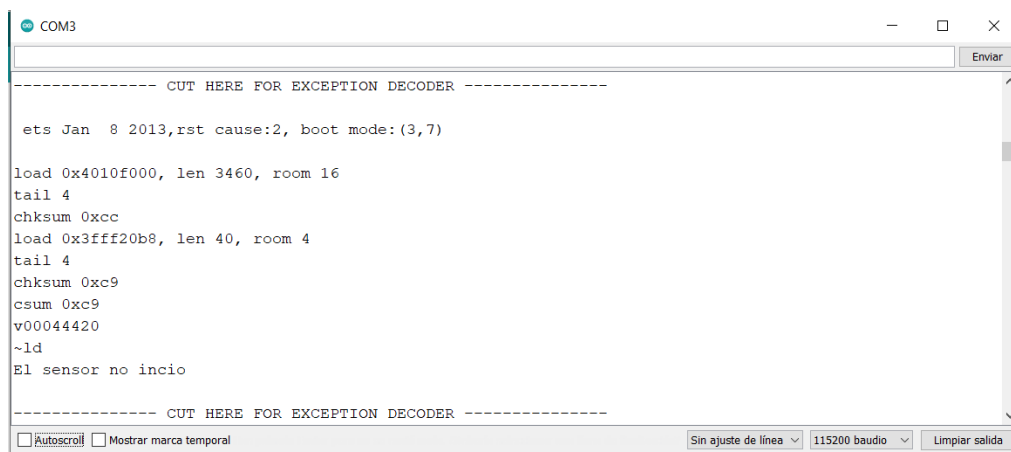


Figura N° 95 No se puede establecer comunicación con el ADXL345

En las siguientes dos figuras podemos ver cómo en función a los valores en los ejes X, Y y Z determinamos si hay o no viento:

```

COM3
Eje x = 0.82      Eje y = 0.39      Eje z = -9.65
No Hay viento
Eje x = 0.71      Eje y = 0.39      Eje z = -9.73
No Hay viento
Eje x = 0.75      Eje y = 0.47      Eje z = -9.77
No Hay viento
Eje x = 0.67      Eje y = 0.39      Eje z = -9.77
No Hay viento
Eje x = 0.67      Eje y = 0.35      Eje z = -9.73
No Hay viento
Eje x = 0.71      Eje y = 0.31      Eje z = -9.73
No Hay viento
Eje x = 0.51      Eje y = 0.35      Eje z = -9.81
No Hay viento
Eje x = 0.51      Eje y = 0.20      Eje z = -9.77
    
```

Figura N° 96 Se observa por los valores que no hay viento

```

COM3
Eje x = 0.39      Eje y = 0.16      Eje z = -9.89
No Hay viento
Eje x = -3.96     Eje y = -0.78     Eje z = -9.41
No Hay viento
Eje x = -6.51     Eje y = -1.45     Eje z = -6.90
Hay viento
Eje x = -6.59     Eje y = -0.51     Eje z = -6.94
Hay viento
Eje x = -6.71     Eje y = -0.63     Eje z = -6.90
Hay viento
Eje x = -6.63     Eje y = -0.43     Eje z = -6.79
Hay viento
Eje x = -7.34     Eje y = 0.08      Eje z = -6.12
Hay viento
Eje x = -7.22     Eje y = 0.27      Eje z = -6.35
    
```

Figura N° 97 Se observa por los valores en X que hay viento

6.3.4 Desarrollo de código de prueba de pantalla OLED

Se procede a desarrollar un código de prueba para establecer comunicación entre la NodeMCU y la pantalla OLED. La conexión entre el módulo y la NodeMCU es la observada en el siguiente esquema, donde el cable amarillo representa la conexión del pin SDA de la pantalla con el GPIO 21 y el cable gris representa la conexión del pin SCL con el GPIO 22:

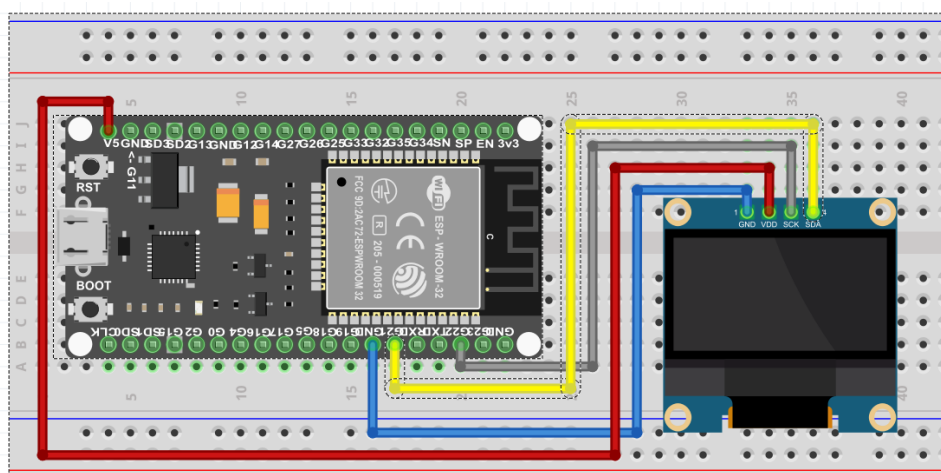


Figura N° 98 Esquema de conexión entre pantalla OLED y NodeMCU

En primer lugar, debemos incluir las librerías necesarias para manejar el módulo. Teniendo presente que el módulo se comunica a través de protocolo I2C, incluimos la librería `wire.h` para poder establecer comunicación. Por otro lado, incluimos la librería `Adafruit_GFX.h` y `Adafruit_SH110X.h` que nos permitirán manejar la representación gráfica en la pantalla y el controlador propio del módulo que usamos en este proyecto.

```
#include <Wire.h>
#include <Adafruit_GFX.h> // estas dos librerias son empleadas para manejar el modulo oled
#include <Adafruit_SH110X.h> // esta especialmente solo funciona con el modulo 128x64 o
128x32 o 128x128 pero que figura 1,3" ya que usan el microcontrolador SH1106
```

Comenzamos definiendo la dirección I2C con la cual nos comunicaremos con el módulo. El modelo empleado en este proyecto tiene la dirección `0X3C`, por lo que procederemos a guardar dicho valor en una variable definida como `Direccion_I2C`.

Luego definiremos una etiqueta llamada `Ancho_display` donde guardaremos el ancho en píxeles de la pantalla, una etiqueta llamada `Altura_display` donde guardaremos la altura en píxeles de la pantalla y por último una etiqueta llamada `OLED_RESET` donde guardaremos un `-1`, valor que indica que el módulo oled empleado no posee botón de reset.

```
#define Direccion_I2C 0x3c //defino la direccion I2C del modulo que para este caso es 0x3c
pero puede variar
```

Sistema de monitoreo para heladas

```
#define Ancho_display 128 // definimos el ancho en pixeles del display
#define Altura_display 64 // definimos el altura en pixeles del display
#define OLED_RESET -1 // algunos modelos de pantallas incluyen el pin de reset pero en
este caso no lo trae, por lo tanto indicaremos esto con un -1
```

Realizada la primera parte de la configuración, se procede a crear una instancia de la librería Adafruit_SH110x, la cual llamaremos display. Para crear dicha instancia deberemos invocar a la siguiente sentencia, pasando como parámetros el ancho de la pantalla, altura de la pantalla, un puntero a la comunicación I2C y por último la etiqueta de reset.

```
//Ahora creamos una instancia de la libreria y la llamamos display y le pasamos el ancho de
la pantalla, alto, un puntero a la comunicacion I2C y por ultimo el parametro de pin de
reset
Adafruit_SH1106G display = Adafruit_SH1106G(Ancho_display, Altura_display, &Wire,
OLED_RESET);
```

Inicializamos el monitor serial y configuramos la velocidad en 115200 baudios. Luego vamos a iniciar la comunicación con el módulo display invocando a la sentencia `display.begin()` y pasando como parámetros la dirección I2C del módulo y el valor booleano `true`. Si no puede inicializarse correctamente, se imprimirá en el monitor serial que hubo un error de conexión con la pantalla OLED.

```
void setup()
{
  Serial.begin(115200);
  if(!display.begin(Direccion_I2C, true)) //inicializamos la pantalla pasandole la direccion
del modulo dentro de un if. Si no se cumple la condicion de TRUE, imprimiremos que hay un
error de conexion a traves del puerto serial
  {
    Serial.println("Error de conexion con pantalla OLED");
  }
}
```

Ahora procederemos a mostrar una línea de texto en la pantalla. En primer lugar vamos a limpiar la pantalla haciendo uso de la sentencia `clearDisplay()`. Luego vamos a configurar el tamaño de la fuente haciendo uso de `setTextSize()` y pasando como parámetro un valor entero, con el cual podrá modificar el tamaño. Continuamos con la sentencia

setTextColor(), la cual permite establecer el color que queremos darle al texto que vamos a escribir. Esto es posible ya que algunos módulos de display OLED traen 2 colores posibles de utilizar. En el caso del modelo empleado aquí, es un display monocromático, por lo cual cada pixel puede tomar el color de encendido o apagado.

Luego vamos a indicar a partir de qué ubicación queremos mostrar el texto. Debemos tener presente que el cero de referencia para el display es el extremo superior izquierdo tanto en el eje X como en el Y. invocando a la sentencia setCursor(x,y) vamos a pasarle la ubicación del pixel a partir del cual queremos mostrar la línea de texto.

```
void loop()
{
  display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
  display.setTextSize(1); //ajusto el tamaño de la fuente
  display.setTextColor(SH110X_WHITE); // dependiendo del modelo de pantalla puedo ajustar el
color
  display.setCursor(0, 10); // ubico el texto tomando como referencia de cero la esquina
superior izquierda de la pantalla. Primero se pasa la posicion en X y luego en Y
```

Una vez configurados estos parámetros referidos a la línea de texto que deseamos imprimir en el display, haciendo uso de la sentencia display.println("") indicaremos entre comillas la línea de texto que deseamos ver en la pantalla. Luego ejecutamos haciendo uso de la sentencia display.display() con la cual vamos a mostrar en pantalla todo lo que se haya escrito en código anterior.

```
display.println("Probando codigo"); // escribo el texto que deseo mostrar
display.println("Peroni Bruno");
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
}
```

Al compilar el código, veremos en pantalla los resultados de la siguiente imagen:

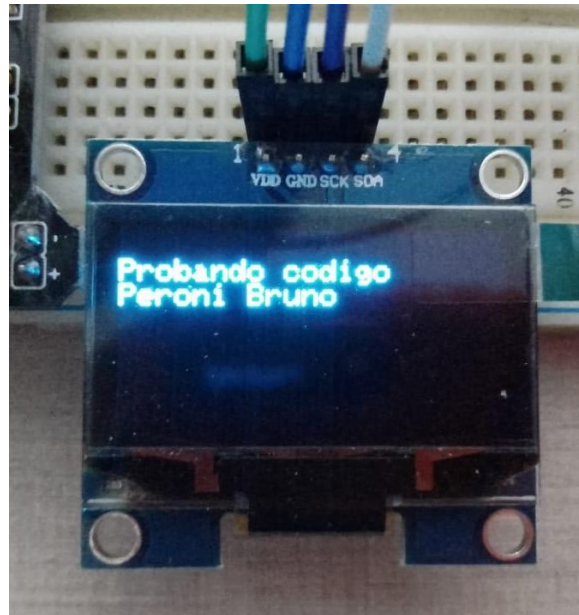


Figura N° 99 Prueba de código para comunicarse con pantalla OLED

6.3.5 Desarrollo de código prueba para trabajar con multitareas empleando los dos núcleos del ESP32

Se procede a desarrollar un código de prueba que me permita usar los dos núcleos que tiene el módulo ESP32, para luego implementar la metodología multitarea en el código principal, de forma tal que un núcleo se encargue de la obtención de la información proveniente de los sensores y el otro núcleo se encargue de la conexión wifi y manejo de información.

En primer lugar, creamos una tarea haciendo uso de la sentencia `TaskHandle` y la llamaremos `tarea1`. Esta tarea luego se la asignaremos a una función y a un núcleo en particular, y de esa forma lograremos emplear el otro núcleo. Además, vamos a inicializar una variable global llamada `cont`, la cual simplemente llevará un conteo. La idea de esta variable es emplearla como contador y luego mostrarla en distintas funciones para poder demostrar que ambos núcleos pueden acceder a cualquier información sin importar donde se encuentre.

Sistema de monitoreo para heladas

```
TaskHandle_t tarea1; //para poder usar mas de 1 nucleo voy a necesitar definir tareas
int cont=0; // declaramos una variable global que sera aumentada desde los dos nucleos, para
mostrar que pueden compartir informacion
```

Ahora debemos indicarle al ESP32 que debe ejecutar la tarea creada y configuraremos donde y como deseamos que se ejecute. Haciendo uso de la sentencia `xTaskCreatePinnedToCore()` vamos a realizar la configuración mencionada, pasándole como argumento la función donde quiero que se ejecute la tarea, el nombre de la tarea, el tamaño de la pila, algún valor que desee pasar como entrada, la prioridad que deseo darle a esta tarea, el identificador y por último el núcleo donde deseo que se ejecute. Luego simplemente inicializo el puerto serial para poder imprimir información.

```
void setup()
{
xTaskCreatePinnedToCore(loop2, "tarea_1", 10000, NULL, 1, &tarea1, 0);
Serial.begin(115200);
}
```

Ahora creamos una función denominada `loop2()` que contendrá el código de la tarea 1. Debemos tener presente que es una función creada de forma artesanal, por lo que no se va a repetir infinitamente por sí sola como si lo hace por defecto la función `loop()`. Debido a esto, debemos generar de alguna forma ese bucle infinito y es por eso que dentro de la función, el código a ejecutar irá dentro del `for(; ;)`. Dentro del `for`, simplemente voy a imprimir el valor que registra la variable `cont` y aumenta en 1 su valor. Además, haciendo uso de la función `xPortGetCoreID()` sabré en qué núcleo se está ejecutando el código en ese momento, por lo que haré uso de esta información y la imprimiré en pantalla.

```
void loop2(void *parameter)
{
  for(;;)
  {
    Serial.println("*****");
    Serial.println("\t\t\tEl nucleo que esta corriendo es el N° "
+String(xPortGetCoreID()));
    delay(100);
    cont++;
    Serial.println("La cuenta es: " + String(cont));
  }
}
```

Sistema de monitoreo para heladas

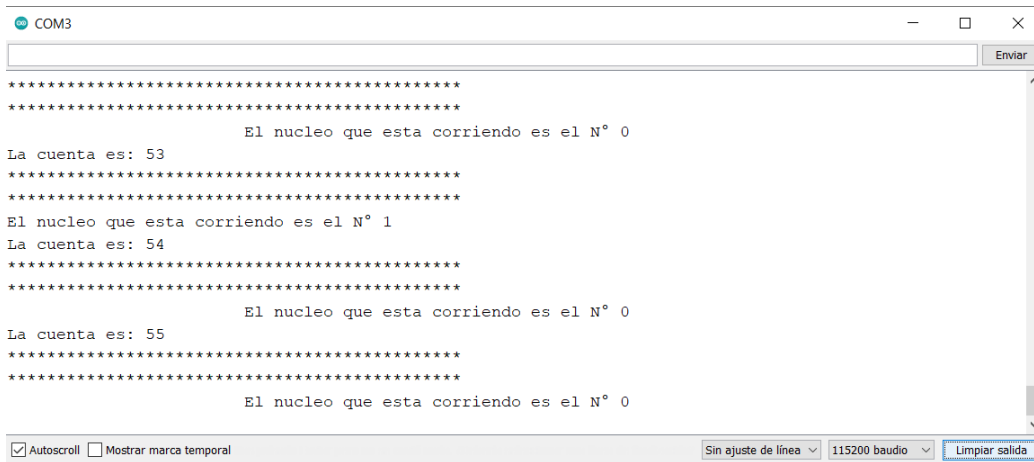
```
Serial.println("*****");
}
vTaskDelay(10);0
}
```

Lo mismo que realizó en la función `loop2()` lo hacemos con `loop()`. De esta manera, en el puerto serial iremos viendo cómo se va ejecutando código en el núcleo 1 o en el núcleo 0 según corresponda. Vemos que la función `loop2()` tiene un delay de 1 segundo y `loop()` un delay de 5 segundos. Si tenemos presente que las dos funciones se están ejecutando al mismo tiempo sin interferir una en la otra, verificaremos en pantalla que cada 1 segundo la función `loop2()` me estará mostrando el valor de la cuenta y en qué núcleo se está ejecutando el código. Por otro lado la función `loop()` cada 5 segundos me va a mostrar el valor almacenado en la variable `cont` y el núcleo donde se está ejecutando el código. De esta manera podemos confirmar que está logrado el funcionamiento de ambos procesadores al mismo tiempo y de manera independiente

```
void loop()
{
Serial.println("*****");
Serial.println("El nucleo que esta corriendo es el N° " + String(xPortGetCoreID())); // con
la funcion xPortGetCoreID() yo voy a obtener en que nucleo se estara ejecutando este void
loop()
delay(1000);
cont++;
Serial.println("La cuenta es: " + String(cont));
Serial.println("*****");
}
```

En la siguiente imagen, podemos observar cómo se va mostrando en pantalla el valor de la cuenta que lleva la variable `cont` y el número del núcleo que se está ejecutando en ese momento.

Sistema de monitoreo para heladas



```
COM3
*****
*****
El nucleo que esta corriendo es el N° 0
La cuenta es: 53
*****
*****
El nucleo que esta corriendo es el N° 1
La cuenta es: 54
*****
*****
El nucleo que esta corriendo es el N° 0
La cuenta es: 55
*****
*****
El nucleo que esta corriendo es el N° 0

 Autoscroll  Mostrar marca temporal
Sin ajuste de línea 115200 baudio Limpiar salida
```

Figura N° 100 NodeMCU ESP32 funcionando en multitarea

6.3.6 Desarrollo de código para enviar datos mediante método POST con NodeMCU

En el siguiente apartado, se desarrollará un código que permitirá enviar información a través de la comunicación wifi, desde la tarjeta NodeMCU hacia un servidor web mediante el método POST. De esta forma los datos se irán almacenando en una base de datos MySQL, a partir de la cual luego podrán ser consultados desde la página web.

En primer lugar, agregaremos la librería WiFi con la cual conectaremos la NodeMCU a una red wifi y la librería HTTPClient para poder realizar las solicitudes HTTP POST. En las dos variables `const char*` vamos a guardar el usuario y contraseña de la red wifi a la cual deseamos conectarnos. Luego crearemos variables globales.

```
//librerias para la conexion WIFI - Cliente ESP32

#include <HTTPClient.h>
#include <WiFi.h>

//creo dos variables constantes donde guardare el nombre de la red y la contraseña

const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red

float temperatura_bmp280 = 0; // la temperatura que detecta el bmp290
float presion=0; // la presion que detecta el bmp290
int temperatura=0,humedad=0; //la temperatura y humedad del sensor dht22
```

Sistema de monitoreo para heladas

```
string viento= "Hay viento"; //variable donde guardaremos información del viento
```

En la siguiente variable llamada url, vamos a almacenar la dirección del archivo php a la cual queremos enviarle los datos. Es importante ver que figura una dirección ip como raíz de la url y eso se debe a que como se está empleando un servidor local, la NodeMCU solo podrá comunicarse con la página si la busca a través de la ip de la computadora que está corriendo el servidor web local. Teniendo esto presente, es importante establecer una ip estática para la computadora que tenga el servidor web local, y mucho más importante es hacerla coincidir con la ip guardada en la url, que para este proyecto es 192.168.0.123.

```
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";
```

Procedemos a realizar la conexión con la red wifi; las siguientes líneas de código ya han sido explicadas anteriormente.

```
void setup()
{
  Serial.begin(115200); //definimos la velocidad del puerto serial
  WiFi.begin(ssid, password); //Iniciamos la conexión con la red, recibiendo como parametros
  el nombre y la contraseña de la red
  Serial.print("Conectando..."); //imprimimos que la conexión se esta realizando

  while(WiFi.status() != WL_CONNECTED and cont < 50) //mientras no se conecte ira aumentando
  un contador //hasta 20 intentos
  {
    cont++;
    delay(500);
    Serial.print(".");
  }

  Serial.println(""); //salto de linea

  if(cont < 50) //si entra quiere decir que se conecto
  {
    Serial.println("*****");
    Serial.print("Conectado a la red WiFi: ");
    Serial.println(WiFi.SSID());
    Serial.print("IP: ");
    Serial.println(WiFi.localIP());
    Serial.print("macAdress: ");
```

Sistema de monitoreo para heladas

```
Serial.println(WiFi.macAddress());
Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
Serial.println("*****");
Serial.println("Error de conexion");
Serial.println("*****");
}
cont=0;
}
```

En el void loop () simplemente llamaremos a una función EnvioDatos() que será quien se encargue del proceso principal. Esto se realiza de esta forma, ya que está pensado para luego poder ser empleado en multitarea, usando los dos núcleos disponibles de la tarjeta NodeMCU ESP32.

```
void loop()
{
EnvioDatos(); //implementamos el envio de los datos al servidor
delay(60000); //espera 60s
}
```

Las sentencias que se encuentran dentro del if, se van a ejecutar solo si la placa se encuentra conectada a la red wifi, es por eso que se coloca la sentencia WiFi.status() == WL_CONNECTED. Vamos a crear un objeto de HTTPClient, lo nombraremos http y luego con la sentencia http.begin() inicializamos la comunicación, pasando como parámetro la url con la cual deseo comunicarme. La url dirige a un archivo del tipo php que se encargará de recibir los datos que se le envían.

Luego haremos uso de la sentencia http.addHeader(), para poder establecer cómo será el encabezado. Los parámetros pasados, establecerán que los datos viajen de forma estructurada y no de forma amontonada. Esto permite que puedan ser recuperados de forma simple en el archivo php.

```
void EnvioDatos()
{
if (WiFi.status() == WL_CONNECTED)
{
HTTPClient http;
```

```
http.begin(url);  
http.addHeader("Content-Type", "application/x-www-form-urlencoded");
```

Se procede a preparar la información que se enviará. Para esto, se crea una variable de tipo string llamada `datos_a_enviar`, y se le asigna el mensaje de forma estructurada tal como se puede ver a continuación:

```
String datos_a_enviar = "temperatura= " + String(temperatura) + "&humedad= " +  
String(humedad) + "&presion= " + String(presion) ;
```

Vemos que se compone por la etiqueta correspondiente, seguido por la variable que contiene el valor. Es importante que las etiquetas utilizadas tengan exactamente el mismo nombre que las etiquetas empleadas en el archivo php. De no ser así, la información no podrá ser recibida en el archivo, entregando como respuesta un error.

Creamos una variable del tipo int llamada `código_respuesta` y la usaremos para almacenar la respuesta que devuelve la función POST. Invocando la sentencia `http.POST()` se está ejecutando el envío de la información que se pase como parámetro.

Tal como se mencionó, la respuesta resulta ser un código numérico. Dependiendo de su valor y signo, podremos saber si los datos fueron enviados correctamente o si está habiendo un problema para concretar el envío. Es por esto que se imprime en el monitor serial cual es la respuesta que devuelve la sentencia `http.POST()`. Cuando los datos son enviados correctamente, la respuesta es el código 200. Por último finalizamos la petición llamando a la sentencia `http.end()`.

```
int codigo_respuesta = http.POST(datos_a_enviar);  
if (codigo_respuesta>0)  
{  
  Serial.println("Código HTTP: "+ String(codigo_respuesta));  
  if (codigo_respuesta == 200)  
  {  
    String cuerpo_respuesta = http.getString();  
    Serial.println("El servidor respondió: ");  
    Serial.println(cuerpo_respuesta);  
  }  
}  
else  
{
```

Sistema de monitoreo para heladas

```
Serial.print("Error enviado POST, código: ");  
Serial.println(codigo_respuesta);  
}  
  
http.end(); // finalizo la peticion  
}  
}
```

En la siguiente imagen podemos ver la respuesta del servidor cuando los datos son enviados correctamente. Básicamente buscamos observar el código de la respuesta y los datos que fueron recibidos por parte del archivo php.

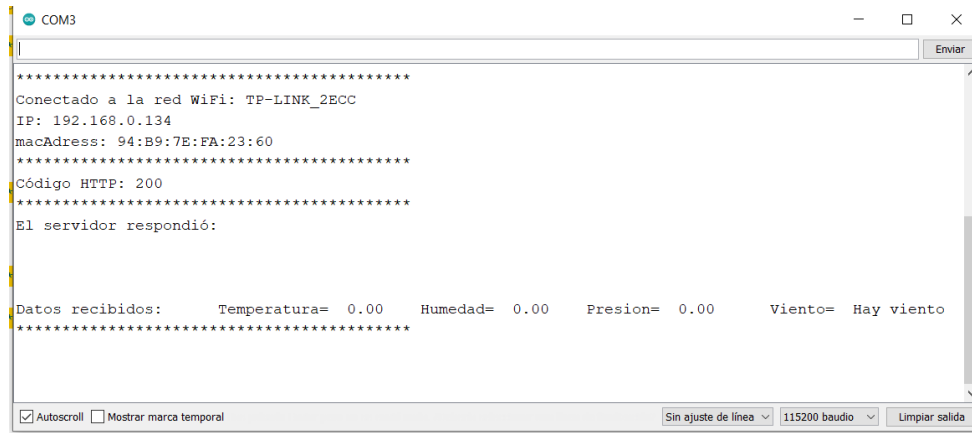


Figura N° 101 **Monitor serial respondiendo al envío por método POST**

Una forma de verificar que los datos se estén recibiendo correctamente es observando la base de datos:

Sistema de monitoreo para heladas

idDevice	Temperatura	Humedad	PresionAtmosferica	Fecha	Viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:43:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:44:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:45:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:46:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:47:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:48:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:49:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:50:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:51:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:52:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:53:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:54:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:55:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:56:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:57:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:58:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 01:59:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:00:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:01:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:02:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:03:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:04:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:05:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:06:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:07:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:08:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:09:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:10:00	Hay viento
tarjeta1	0.00	0.00	0.00	2021-11-12 02:11:00	Hay viento

Figura N° 102 Información que va llegando desde la NodeMCU

6.3.7 Desarrollo de código para activar el modo Deep Sleep.

Tal como ya hemos explicado, el modo sueño profundo nos permitirá reducir el consumo de energía del sistema, de forma tal que pueda extenderse la vida útil de las baterías que alimentan al circuito. En el presente apartado, se desarrolla un código de prueba para poner en funcionamiento este modo, mandando a dormir el microprocesador por 60 segundos para luego despertarlo y así repetir el ciclo.

Comenzamos definiendo dos etiquetas que usaremos para la configuración del tiempo que pondremos a dormir el ESP32. La etiqueta `factor_conversion` simplemente contiene el factor que permitirá pasar de segundos a microsegundos, debido a que la función `esp_sleep_enable_timer_wakeup()` acepta como parámetro el tiempo en microsegundos. Por otro lado, la etiqueta `segundos` contendrá la cantidad de segundos que deseamos esté activo el modo sleep. Haciendo uso de estas dos etiquetas, guardaremos en la variable `tiempo` la multiplicación de ambas, dando como resultado la conversión del valor que se encuentra en la variable `segundos` a microsegundos, listo para ser pasado como parámetro de configuración.

```
#define factor_conversion 1000000
#define segundos 60
```

```
int tiempo = segundos * factor_conversion ;
```

Tal como ya mencionamos anteriormente en este modo los núcleos, la mayor parte de la RAM y todos los periféricos digitales que están sincronizados, están apagados. Las únicas partes de los chips que aún se pueden encender son el controlador RTC, periféricos RTC y memorias RTC. Con el ESP32, podemos guardar datos en las memorias RTC, ya que cuenta con 8 Kbyte de SRAM en la parte RTC, la cual se la conoce como memoria rápida RTC. Los datos guardados en esta parte de la memoria no se borran durante el sueño profundo. Para poder guardar datos en la memoria RTC, tenemos que incluir la sentencia RTC_DATA_ATTR antes de una definición de variable. Usaremos la variable cont para contar cuántas veces se ha despertado el ESP32 del sueño profundo.

```
RTC_DATA_ATTR int cont= 0;
void setup()
{
  Serial.begin(115200);
}
```

Cada vez que sea despertado el ESP32, vuelve a ejecutar el código desde el principio, pasando por el void setup () y luego continuando con el resto de las funciones. Es por esto que cada vez que se ingrese al void loop () se aumentará el contador ya que eso significa que el ESP32 despertó.

A continuación, se imprime en pantalla el valor del contador, para tener una idea de cuantas veces ya ha despertado. Continuamos con la asignación del tiempo que deseamos que dure el modo sueño profundo, incluyendo la sentencia ya mencionada y pasando como parámetro la variable tiempo.

```
void loop()
{
  cont++;//incremento cada vez que se despierta
  Serial.println("*****");
  Serial.println("Despierta ESP32, el numero de veces que durmio es: " +
String(cont));
  Serial.println("");
}
```

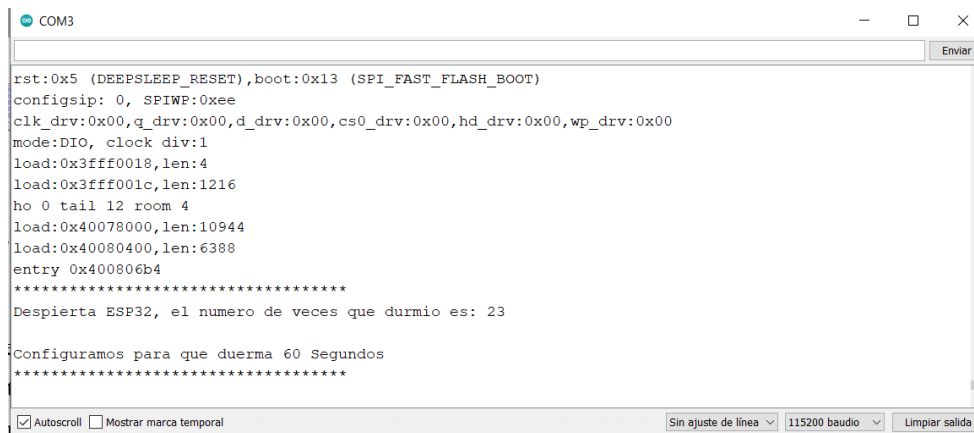
Sistema de monitoreo para heladas

```
esp_sleep_enable_timer_wakeup(tiempo); //configuramos el tiempo que deseamos que
duerma
Serial.println("Configuramos para que duerma " + String(segundos) +
" Segundos");
Serial.println("*****");
```

Para poder mandar a dormir el ESP32, se invoca a la sentencia `esp_deep_sleep_start()` y a partir de ese momento se inicia el temporizador. Es importante tener presente que no se continuará con la ejecución del código que pueda llegar a encontrarse debajo de la sentencia. Esto es importante tenerlo presente ya que me limita a tener que hacer todas las tareas pendientes si o si antes de mandar a dormir el ESP32. Para verificar esto último, se busca imprimir en el monitor serial una línea de texto.

```
esp_deep_sleep_start(); //con esta sentencia iniciamos el modo deep sleep
Serial.println("Esto nunca deberia imprimirse porque como mande a dormir el micro,
cuando despierta es como si se reseteara la placa, volviendo al principio del setup ");
}
```

En la siguiente imagen podemos apreciar las respuestas de la tarjeta en el monitor serial



```
COM3
rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10944
load:0x40080400,len:6388
entry 0x400806b4
*****
Despierta ESP32, el numero de veces que durmio es: 23
Configuramos para que duerma 60 Segundos
*****
```

Figura N° 103 Respuesta del ESP32 despertando cada 60 segundos

6.3.8 Desarrollo de código para activación o desactivación de sistema de riego:

En el presente apartado, se explicará el desarrollo del código que contendrá la segunda tarjeta NodeMCU, con la cual activaremos o desactivaremos el sistema de riego. Para

llevar a cabo el diseño, se emplearon los dos núcleos que tiene disponible la tarjeta NodeMCU. Con el núcleo N°0 se va a monitorear la correcta conexión de la tarjeta NodeMCU a la red wifi deseada, indicando esto con un led parpadeando a una velocidad característica. Por otro lado, si se detectara una pérdida de conexión, se reiniciará la tarjeta para que pueda volver a intentar establecer conexión con la red. El núcleo N°1 se encargará de ejecutar el servidor web, levantar la página y atender las peticiones realizadas por el usuario, que serán encender o apagar el sistema de riego.

Se tendrá presente que ya fue explicado en profundidad como emplear multitareas con la NodeMCU ESP32 en el apartado 6.3.5, por lo que no se abordará aquí.

En la siguiente figura podemos ver el circuito esquemático que vamos a utilizar para probar el código:

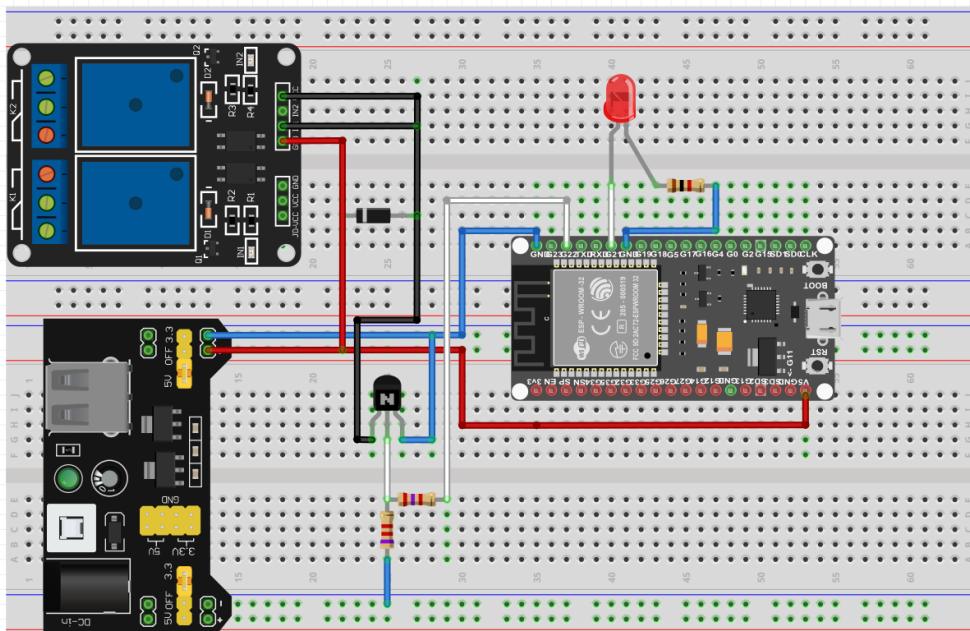


Figura N° 104 **Circuito esquemático para activación de sistema de riego**

En primer lugar, incluimos la librería que contiene todas las funciones que nos permitirá establecer la conexión y el modo de funcionamiento. Luego definimos al pin GPIO 22 con la etiqueta pinLed y el pin GPIO 21 con la etiqueta parpadeo. Haciendo uso de GPIO 22 entregaremos el estado alto o bajo seleccionado a través de la página web, con el cual accionaremos el sistema de riego o lo desactivaremos, ya que actuaremos sobre un transistor trabajando en corte o saturación para hacer el efecto de llave on-off. Por otro lado, con GPIO 21 lo que haremos es emplear un led como indicador del estado de conexión wifi entre la placa NodeMCU y la red wifi, determinando la correcta conexión o no en función a la rapidez con la cual se enciende y apaga el led. También definiremos dos variables del tipo String, donde almacenaremos el nombre y la contraseña de la red a la cual queremos que la placa se conecte. En las siguientes imágenes podemos ver el circuito real implementado en protoboard:

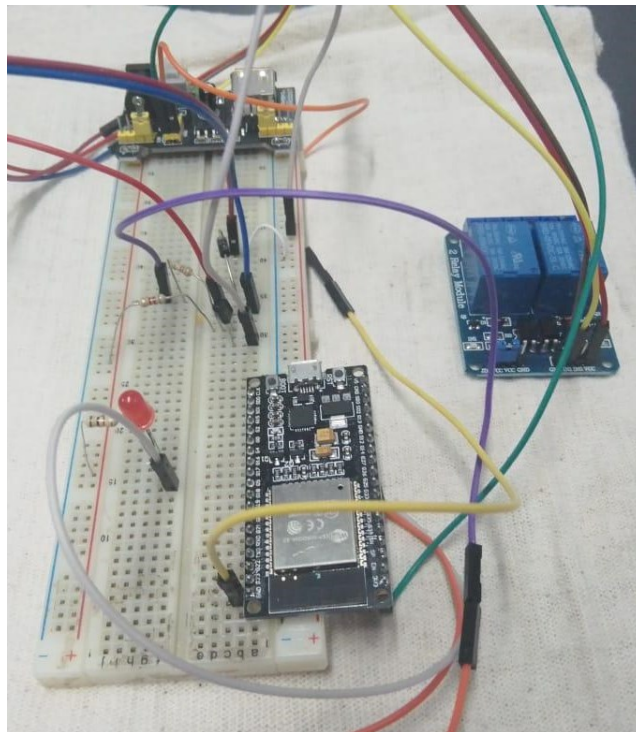


Figura N° 105 **Circuito de prueba montado en protoboard**

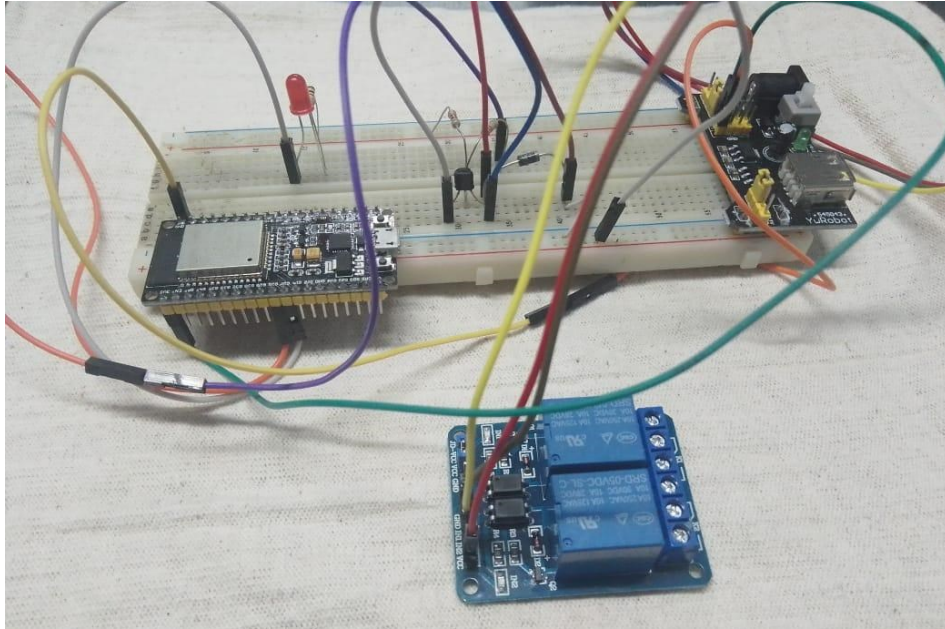


Figura N° 106 Circuito de prueba montado en protoboard desde otra perspectiva

```
#include <WiFi.h>

TaskHandle_t tarea1;//para poder usar mas de 1 nucleo voy a necesitar definir tareas

#define parpadeo 21
#define pinLed 22
const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red
```

Ahora vamos a realizar una pequeña configuración para asignar una ip estática a la NodeMCU, ya que para poder levantar la página web cargada en su código, deberemos llamar a la ip de la tarjeta. Esto es necesario debido a que el router asigna una ip distinta cada vez que se conecta un nuevo dispositivo, lo cual no sería útil.

```
//a continuacion asigno una ip estatica para la tarjeta, para que siempre sea la 108
IPAddress ip(192,168,0,108);
IPAddress gateway(192,168,0,1);
IPAddress submit(255,255,255,0);
```

Se procede a crear un objeto de la clase WiFiServer y se indica emplear el puerto 80 para establecer la comunicación. Luego se inicializa una variable global llamada cont, la cual será empleada para determinar si se superó o no el límite de tiempo fijado para conectarse a la red. También es creada una variable string llamada header, en la cual se guardará el HTTP request y una variable string denominada estado, la cual se emplea en la interacción con la página web.

```
WiFiServer server(80); //objeto de la clase WiFiServer
                        //asigno el puerto 80 que es el tcp, ya que es el puerto que se
emplea para la navegacion web http
int cont1 = 0;
int cont = 0;
String header; //Usare esta variable para guardar el HTTP request
String estado = "DESACTIVADO" ;
```

Iniciamos la comunicación por el puerto serial, establecemos una velocidad de transmisión de 115200 baudios y configuramos pinLed y parpadeo como salidas. Luego vamos a inicializar la configuración de la ip estática haciendo uso de la sentencia WiFi.config y pasándole las variables globales definidas anteriormente. Se continua con el inicio de la conexión Wifi empleando la sentencia WiFi.begin(), pasándose el usuario y contraseña de la red como parámetros.

```
void setup()
{
  // Inicia Serial
  Serial.begin(115200);
  Serial.println("\n");
  pinMode(parpadeo,OUTPUT);
  pinMode(pinLed,OUTPUT);
  digitalWrite(parpadeo, LOW);
  digitalWrite(pinLed, LOW);
  WiFi.config(ip,gateway,subnet); //configuro la ip estatica
  WiFi.begin(ssid, password); //inicio la conexion wifi
```

A continuación, buscamos determinar si hay o no una correcta conexión a la red wifi. Haciendo uso del monitor serial y del pin GPIO 21 definido como parpadeo, podremos representar que está sucediendo con la conexión.

En primer lugar, si detectamos que no se está conectando la placa a la red deseada, haremos parpadear el led conectado en el pin GPIO 21, respetando los tiempos de 100 mseg en alto y 500 mseg en bajo, lo cual se visualiza como un parpadeo rápido. Esto se ejecutará solo si se cumple la condición de no existir una conexión satisfactoria y que, además, la variable cont sea menor a 20. Este valor de 20 es simplemente un valor seleccionado como tolerancia de intentos de conexión, para que no se cree un bucle infinito en el caso de que nunca pueda lograrse la conexión.

```
while (WiFi.status() != WL_CONNECTED && cont<20 )
{
  cont++;
  Serial.print(".");
  digitalWrite(parpadeo, HIGH);
  delay(100);
  digitalWrite(parpadeo, LOW);
  delay(500);
}
```

Si la conexión se concreta, la variable cont no debería superar el valor 20, por lo que, si esto se cumple, se procede a mostrar información representativa de una comunicación satisfactoria con la red, continuando con la inicialización del servidor.

```
if(cont < 20)
{
  Serial.println("");
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
  server.begin(); //begin() levantamos el servidor
}
```

Por otro lado, si la variable cont supera el número 20, significa que superó el número máximo de intentos de conexión, por lo cual la tarjeta no logró comunicarse con la red. Si esto se cumple, se mostrará en pantalla que hubo un error de conexión y se ejecutará la

sentencia `ESP.restart()`. Esta función permite reiniciar la tarjeta NodeMCU, teniendo el mismo efecto que si se apagara y volviera a encender. De esta forma, el código volverá a ejecutarse desde el principio, iniciando nuevamente el intento de conexión. Esto debe realizarse de esta forma, ya que necesitamos volver a ejecutar las configuraciones wifi definidas en el `setup()`, las cuales se ejecutan solamente una vez.

```
// si no llega a conectarse, indicare que hubo error y voy a reiniciar la placa
mediante el comando ESP.restart() para que vuelva al inicio del setup() e intente conectarse
nuevamente
else
{
  Serial.println("");
  Serial.println("Error de conexion");
  ESP.restart();
}
xTaskCreatePinnedToCore(loop2, "tarea_1", 10000, NULL, 1, &tareal, 0); // vamos a crear una tarea
que apunte a un nucleo en particular, en este caso el nucleo N°0
```

El loop principal será ejecutado por el núcleo N°1, el cual se va a encargar de correr todo lo referido a la página web.

Vamos a crear un objeto de la clase `WiFiClient` llamado `client`, es decir básicamente que estamos creando un cliente wifi. Este cliente lo igualamos a `server.available()`, ya que es una sentencia que se va a encargar de escuchar a los clientes entrantes, es decir a los navegadores web que se están queriendo conectar con la NodeMCU y si hay un cliente disponible el resultado se lo va asignar al objeto `client`.

Luego creó una variable `String` llamada `auxiliar`, donde espero almacenar la información proveniente desde el cliente. Continuamos creando un bucle el cual se va a ejecutar mientras el cliente esté conectado, haciendo uso de la sentencia `client.connected()`.

Llamando a la función `client.available()` veremos si hay bytes para leer desde el cliente y si es así, empezaremos a leer byte a byte haciendo uso de la sentencia `client.read()` y lo imprimimos en el monitor serial. Cada nuevo byte que irá llegando lo iremos concatenando en la variable `header`, de forma tal de almacenar toda la cabecera del HTTP request.

```
void loop()
{
```

Sistema de monitoreo para heladas

```
delay(1000);
WiFiClient client = server.available(); //objeto de la clase WiFiClient

if (client)
{
  Serial.println("Nuevo cliente..."); //Si se conecta un nuevo cliente entonces lo indicamos
  en pantalla
  String auxiliar = "";
  while (client.connected())
  { //mientras el cliente este conectado
    if (client.available())
    { //si hay bytes para leer desde el cliente
      char c = client.read(); // entonces procedemos a leer un byte y luego imprimimos ese
      byte en pantalla
      Serial.write(c);
      auxiliar += c;
    }
  }
}
```

Necesitamos determinar cuando llegamos al final del HTTP request, por lo cual debemos buscar un salto de línea seguido de un espacio en blanco. Una vez encontrado esto, procedemos a responderle al cliente.

```
if (c == '\n')
{ /* vamos a buscar el byte que sea un caracter salto de linea
ya que eso significaria que llegamos al fin del HTTP request del cliente, por lo cual
procederemos a responder : */

  if (auxiliar.length() == 0) //con esto buscamos la línea en blanco
  {
    // Este es el formato de respuesta que se respeta ante una HTTP request, donde se va
    a responder con el tipo d encabezado "HTTP/1.1 200 OK", el tipo de contenido que va a
    recibir y una línea en blanco
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close");
    client.println();
  }
}
```

Una vez que almacenamos todos los bytes en la variable header, lo que hacemos es analizar que petición está realizando el cliente. Si la petición es “GET /22/on” lo que debemos hacer es asignarle a la variable estado la palabra “ACTIVADO” y luego poner en alto la salida GPIO. Por otro lado, si la petición es “GET /22/off”, debemos asignarle a la variable estado la palabra “DESACTIVADO” y debemos poner en bajo la salida GPIO. Como

condición vemos que establecemos que sea mayor o igual a cero y eso se debe a que si el resultado fuera igual a -1, significa que no encontró ninguna petición.

```

if (header.indexOf("GET /22/on") >= 0)
    {
        Serial.println("GPIO 22 ACTIVO");
        estado = "ACTIVADO";
        digitalWrite(pinLed, HIGH);
    }
else if (header.indexOf("GET /22/off") >= 0)
    {
        Serial.println("GPIO 22 DESACTIVADO");
        estado = "DESACTIVADO";
        digitalWrite(pinLed, LOW);
    }

```

Procedemos a imprimir en el cliente la página html que deseamos levantar, teniendo presente que debemos reemplazar las comillas dobles del código original, por comillas simples. Básicamente la página consiste en un solo botón mediante el cual podremos activar o desactivar el riego.

```

client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">
```

Sistema de monitoreo para heladas

```
client.println("</body></html>");
```

Para finalizar la respuesta HTTP request lo que debemos hacer es enviar una línea en blanco al cliente. Luego procedemos a limpiar las variables String empleadas y finalizamos la conexión con el cliente

```
    client.println();
    break; //con esto salimos del bucle
}
else
{ // limpiamos la variable auxiliar por si volvemos a utilizarla
  auxiliar = "";
}
}
else if (c != '\r') // si c es distinto al carácter de retorno de carro, ire agregando
carácter a carácter a la variable auxiliar. La idea es ir agregando todo lo que llega
{
  auxiliar += c;
}
}
}
Header = ""; //limpiamos la variable header

client.stop(); //cerramos la conexión
Serial.println("Cliente desconectado");
Serial.println("");
}
}
```

En las siguientes imágenes podemos ver la página web que se cargará cuando me comunique con la NodeMCU y desee activar el sistema de riego o desactivarlo.

Sistema de monitoreo para heladas

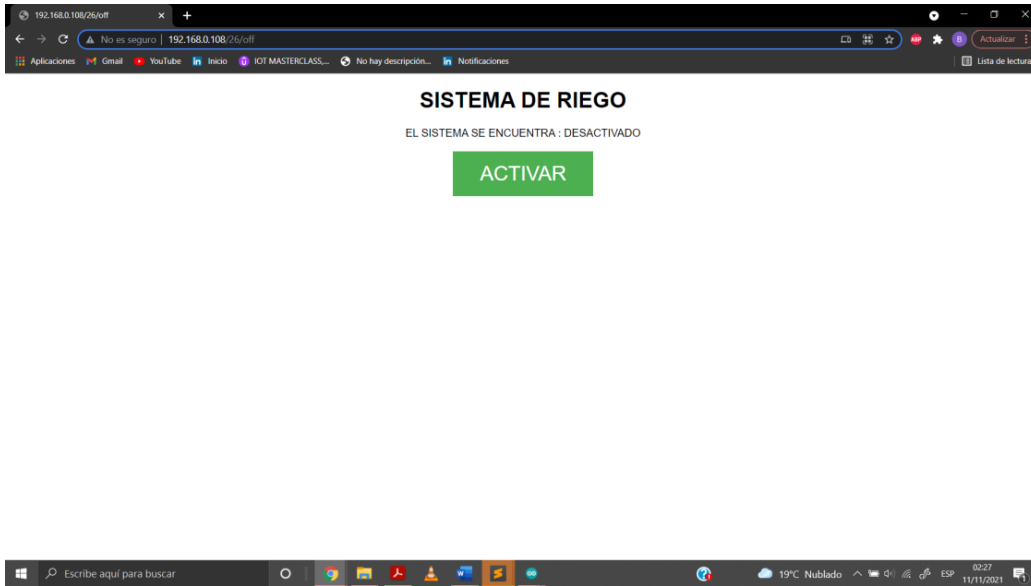


Figura N° 107 **Página para activar el sistema de riego**



Figura N° 108 **Página para desactivar el sistema de riego**

Por otro lado, tenemos código que se estará ejecutando en el núcleo N°0 al mismo tiempo, a través de la función denominada `loop2()`. Dentro de esta función, simplemente se busca controlar un led para poder tener información visual sobre el estado de conexión entre

la placa y la red wifi. Además, se monitorea el posible evento de pérdida de conexión con la red wifi, de forma tal que al detectarse esto se vuelva a realizar el intento de comunicación.

Tal como se puede observar en primer lugar, se colocan algunas sentencias explicadas anteriormente, que permitirán observar en el monitor serial en qué núcleo se está ejecutando esta parte del código. Esto es solamente para tener una verificación visual de que el código está funcionando en multitarea correctamente. Además, se incrementa un contador para tener un registro de funcionamiento sin interrupción.

```
void loop2(void *parameter)
{
  for(;;) //de esta manera logro que se ejecute infinitamente
  {
    Serial.println("*****");
    Serial.println("\t\t\tEl nucleo que esta corriendo es el N° " + String(xPortGetCoreID()));
    // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo se estara ejecutando este
void loop()
    cont1++;
  }
}
```

Haciendo uso de la sentencia `if (WiFi.status() != WL_CONNECTED)`, se busca estar constantemente monitoreando una posible desconexión de la red. Si esta condición se cumple, significa que la NodeMCU ya no está conectada a la red wifi por algún motivo. Para poder restablecer la conexión, necesitamos volver a iniciar la conexión wifi, realizando nuevamente las configuraciones necesarias, las cuales fueron ejecutadas en el `setup()`. Para concretarlo, se procede a invocar a `ESP.restart()` para que se vuelva a ejecutar el código desde el inicio.

```
if(WiFi.status() != WL_CONNECTED )
{
  ESP.restart();
}
```

Por otro lado, mientras se concrete la conexión correctamente, se procederá a indicar ese estado de forma visual, haciendo parpadear el led conectado en GPIO 27 a razón de 100 mseg en alto y 2 seg en bajo, lo cual da un parpadeo lento, muy distinguible al parpadeo rápido que se ejecuta cuando se pierde conexión con la red.

```
if(WiFi.status() == WL_CONNECTED )
```

```
{  
  digitalWrite(parpadeo, HIGH);  
  delay(100);  
  digitalWrite(parpadeo, LOW);  
  delay(2000);  
}  
Serial.println("La cuenta es: " + String(cont1));  
Serial.println("*****");  
}  
}
```

6.3.9 Desarrollo de código principal:

En el presente apartado se explicará el desarrollo del código final que correrá la tarjeta NodeMCU que se encargará de manejar la información sensada. Debido a que el código es muy extenso, se lo adjuntará directamente al final del informe en el Apéndice A, encontrándose en el apartado 11.1.17 con todos sus comentarios para poder ser interpretado rápidamente. Sin embargo, procederemos a explicar cómo se realizan en código las funciones principales, que serían las siguientes:

- Funcionar en modo multitarea, empleando sus dos núcleos
- Establecer la conexión inalámbrica con una red wifi
- Comunicarse con los sensores para leer su información
- Enviar la información a una base de datos
- Mostrar la información de los sensores en un display
- Activar el modo sueño profundo por 49 segundos para luego despertar.

Para realizar la prueba de este código, se procedió a realizar el armado del circuito en una protoboard, respetando el esquema que se puede observar a continuación:

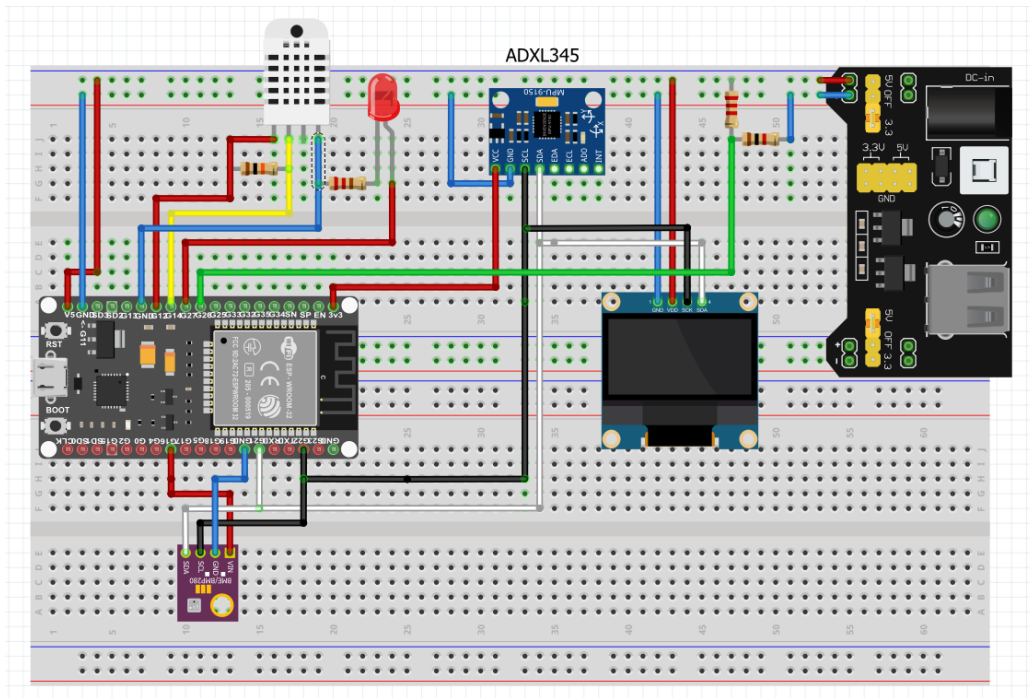


Figura N° 109 **Circuito esquemático empleado para probar el código**

En primer lugar, podemos ver que tenemos conectado un led al pin GPIO 27, el cual estará configurado como salida. Este led, se emplea como un indicador del estado de conexión wifi. Mientras la tarjeta esté intentando concretar la conexión, el led estará parpadeando. Una vez que confirme una conexión correcta con la red, este dejará de parpadear y quedará apagado. Si el led parpadea indefinidamente, eso indicará que la conexión no se está logrando, por lo que al cabo de un tiempo se reiniciará la placa para volver a intentar la conexión.

Luego podemos ver el sensor DHT22, el cual se esta alimentando con 3,3V y GND, empleando el pin GPIO 12 para comunicarse con el pin de datos del sensor, a través del cable de color amarillo.

También podemos ver el sensor BMP280 alimentado con 3,3V y GND, que al emplear protocolo I2C, comunicara su pin SCL al GPIO 21 (cable de color blanco) y el pin SDA al GPIO 22 (cable de color negro) de la tarjeta, y se alimentara a través del pin GPIO 16. Lo mismo sucederá con el acelerómetro ADXL345 y con la pantalla Oled, ya que ambos se alimentan con 3,3V y emplean protocolo I2C, lo cual implica que comparten el mismo bus de datos

Por último, vemos el cable verde que comunica el pin GPIO 26 con un divisor resistivo. Esto es para simular la lectura analógica que realiza este pin, ya que luego será empleado para medir la carga que tendrán las baterías que energizan el circuito.

En las siguientes imágenes, podemos ver la implementación real montada en protoboard:

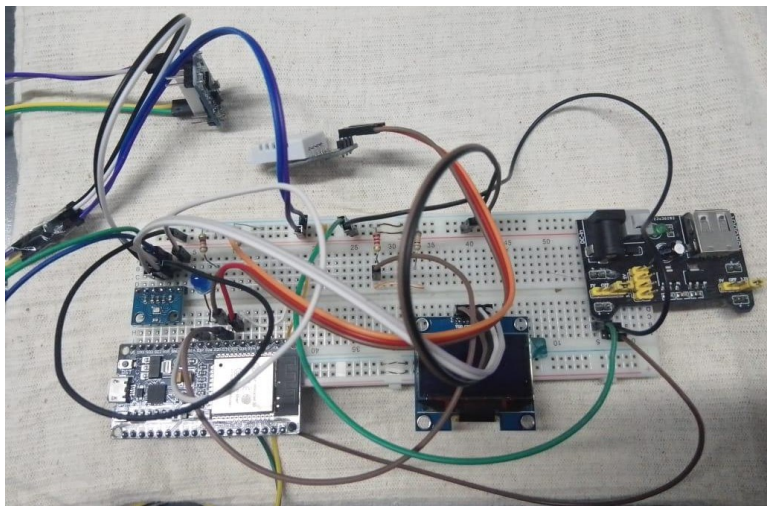


Figura N° 110 **Circuito empleado en la prueba de código**

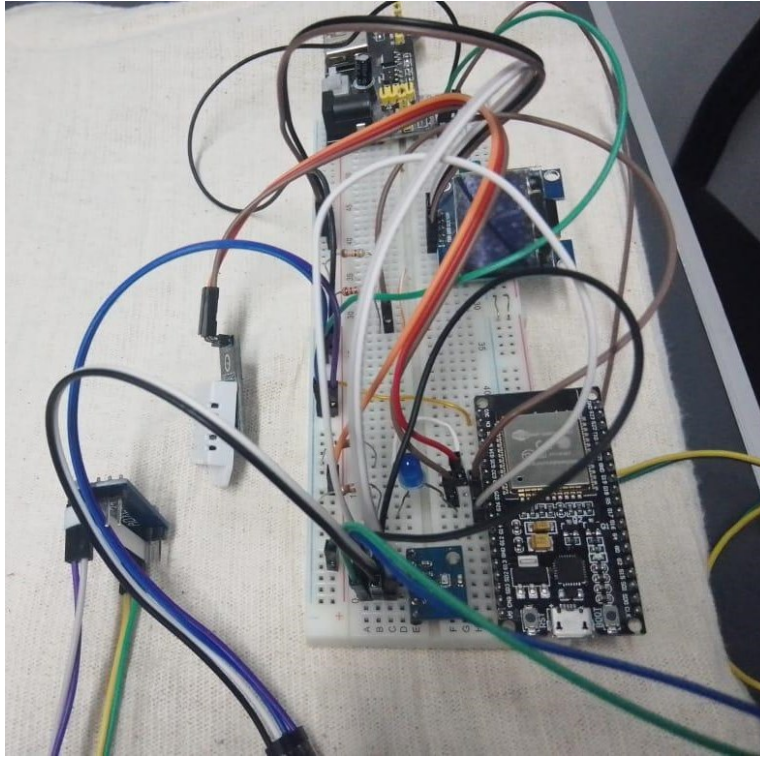
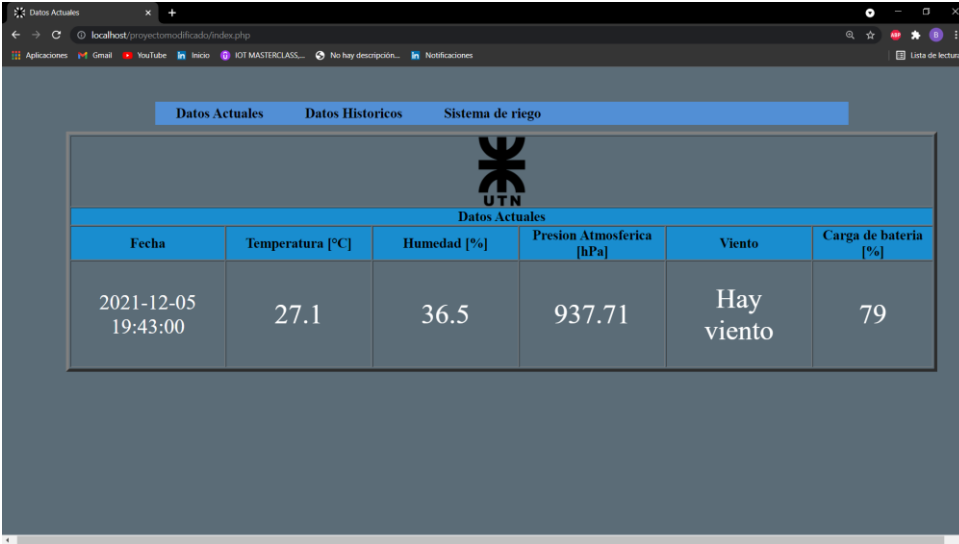


Figura N° 111 **Circuito empleado en la prueba de código desde otra perspectiva**

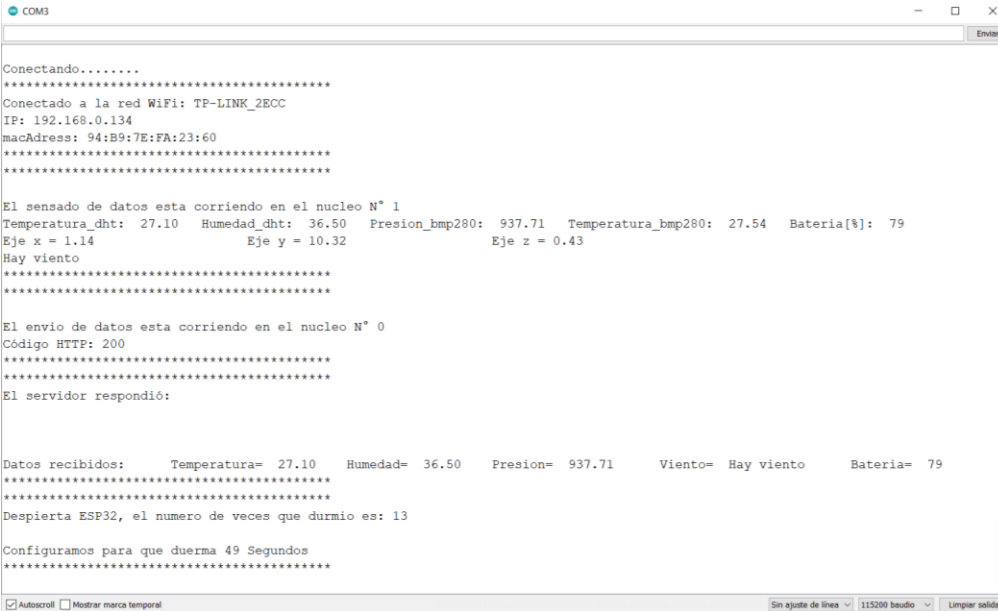
En primer lugar, se procedió a energizar el circuito a través del puerto USB de la NodeMCU, para poder corroborar que los datos visualizados en la telemetría, coincidan con la información de control que se imprime por el monitor serial del IDE de Arduino. Los resultados obtenidos fueron correctos y se pueden apreciar en las siguientes dos imágenes:

Sistema de monitoreo para heladas



Fecha	Temperatura [°C]	Humedad [%]	Presion Atmosferica [hPa]	Viento	Carga de bateria [%]
2021-12-05 19:43:00	27.1	36.5	937.71	Hay viento	79

Figura N° 112 Captura de pantalla de la telemetría



```
COM3
Conectando.....
*****
Conectado a la red WiFi: TP-LINK_2ECC
IP: 192.168.0.134
macAddress: 94:B9:7E:FA:23:60
*****
El sensado de datos esta corriendo en el nucleo N° 1
Temperatura_dht: 27.10 Humedad_dht: 36.50 Presion_bmp280: 937.71 Temperatura_bmp280: 27.54 Bateria[%]: 79
Eje x = 1.14 Eje y = 10.32 Eje z = 0.43
Hay viento
*****
El envio de datos esta corriendo en el nucleo N° 0
Codigo HTTP: 200
*****
El servidor respondi6:

Datos recibidos: Temperatura= 27.10 Humedad= 36.50 Presion= 937.71 Viento= Hay viento Bateria= 79
*****
Despierta ESP32, el numero de veces que durmio es: 13

Configuramos para que duerma 49 Segundos
*****
Autoscroll [x] Mostrar marca temporal [ ] Sin ajuste de linea [v] 115200 baudio [v] Limpiar salida [b]
```

Figura N° 113 Captura de pantalla del monitor serial del IDE de Arduino

Comenzando con el código, en primer lugar, se cargan todas las librerías que se van a utilizar, se definen etiquetas y se inicializan variables globales. Tal como vemos a continuación, estas primeras líneas de código se han separado para diferenciar rápidamente que se emplea con un respectivo modulo o sensor, para tratar de mantener un orden o estructura:

Sistema de monitoreo para heladas

```
//***** librerias para la conexion WIFI - Cliente ESP32 *****
#include <WiFi.h> //la libreria WIFI.h me permite acceder a las funciones que van a *
dejarme conectar a la red wifi
#include <Ticker.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <HTTPClient.h>

//***** Librerias para el sensor DHT22*****
#include <DHT.h> //libreria para usar el DHT22
#include <DHT_U.h> //libreria para usar el DHT22

//***** librerias para manejar BMP280 y ADXL345*****
#include <Wire.h> // se incorpora para poder trabajar con I2C
#include <Adafruit_Sensor.h> //estas dos librerias son para emplear el sensor
#include <Adafruit_BMP280.h>
#include <Adafruit_ADXL345_U.h>

//***** lineas definidas para la conexion WIFI *****
#define ledWifi 26 // definimos el pin 26 como el pin de led wifi
Ticker tic_WifiLed; //creamos un objeto de la clase ticker
const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red
int cont=0; //contador usado para limitar a 50 intentos de conexion wifi

void parpadeoLedWifi()
{
    //esta funcion me permite cambiar el estado actual del led al contrario
    byte estado = digitalRead(ledWifi); //digitalRead lee el estado en cual se encuentra el
led
    digitalWrite(ledWifi, !estado); // setea el led en el estado contrario
}

//*****
//***** variables empleadas para almacenar la informacion de los sensores*****

DHT dht(32,DHT22); //indico que los datos del sensor se conectan al 32 de la placa y que
usare el sensor DHT22. El sensor solo funciona en el pin 32 o 36 de la ESP32
Adafruit_BMP280 bmp; //creamos un objeto de esa clase
String device="tarjeta1"; //es el idDevice que estamos usando en la base de datos para la
ESP8622
float temperatura_bmp280 = 0; // la temperatura que detecta el bmp290
```

Sistema de monitoreo para heladas

```
float presion=0; // la presion que detecta el bmp290
float temperatura=0,humedad=0; //datos provenientes desde el DHT22

Adafruit_ADXL345_Unified ADXL345 = Adafruit_ADXL345_Unified();
int x=0,y=0 ,z=0; // guardare los valores de cada eje para ver si hay presencia de viento o
no, tomando un valor estatico de x=0 y=0 y Z=-10 como referencia de que no hay viento
#define alimentacion_controlada 16
#define alimentacion_controlada2 25

boolean Viento = true;
String viento="";
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";

/* /hola/historicos.php aca asignamos la url a la cual queremos enviarle los datos. Es
importante ver que tengo que reemplazar LOCALHOST por la ip de mi maquina ya que solamente
asi podra comunicarse la tarjeta ESP32.Si fuera un servidor de internet, o haria falta.
*/

//***** tarea creada para poder emplear el otro nucleo del ESP32 *****
TaskHandle_t tarea1; //para poder usar mas de 1 nucleo voy a necesitar definir tareas

//***** variables empleadas para el sueño profundo *****
#define factor_conversion 1000000 /* creamos un factor de conversion para poder pasar de
microsegundos a segundos, debido a que la funcion que me permite configurar el tiempo que
deseo mandar a dormir el ESP32 toma los valores en microsegundos*/
#define segundos 49 /* asignamos el numero de segundos que deseamos mandar a dormir
la placa*/
int tiempo = segundos * factor_conversion ; //los segundos pasados a microsegundos
RTC_DATA_ATTR int contador= 0; //de esta forma declaramos una variable que se almacenara en
la memoria flash del micro procesador RTC el cual no se apaga cuando mandamos a dormir

int bateria=0; //variable donde voy a guardar el porcentaje de carga de la bateria

//*****
// librerias y variables para el uso de pantalla OLED 128x64 monocromatica

#include <Adafruit_GFX.h> // estas dos librerias son empleadas para manejar el modulo oled
#include <Adafruit_SH110X.h> // esta especialmente solo funciona con el modulo 128x64 o
128x32 o 128x128 pero que figura 1,3" ya que usan el microcontrolador SH1106

#define Direccion_I2C 0x3c //defino la direccion I2C del modulo que para este caso es 0x3c
pero puede variar
#define Ancho_display 128 // definimos el ancho en pixeles del display
```

Sistema de monitoreo para heladas

```
#define Altura_display 64 // definimos el altura en pixeles del display
#define OLED_RESET -1 // algunos modelos de pantallas incluyen el pin de reset pero en
este caso no lo trae, por lo tanto indicaremos esto con un -1

/* Ahora creamos una instancia de la libreria y la llamamos display y le pasamos el ancho de
la pantalla, alto, un puntero a la comunicacion I2C y por ultimo el parametro de pin de
reset */
Adafruit_SH1106G display = Adafruit_SH1106G(Ancho_display, Altura_display, &Wire,
OLED_RESET);
```

Luego continuamos con el `setup()`, función que se ejecuta una sola vez y en la cual debo encargarme de configurar e inicializar las comunicaciones necesarias. Fundamentalmente vamos a configurar una tarea para que pueda ser ejecutada por el segundo núcleo que tiene disponible la ESP32. Luego vamos a inicializar las comunicaciones con los sensores, módulos y monitor serial, pasándole todo parámetro de configuración que sea necesario para la comunicación. Por último se intenta establecer la comunicación con la red wifi que le indicamos a la placa. En caso de no poder conectarse luego de un determinado número de intentos, el código llevará a que se reinicie la placa. Este proceso de restablecimiento, se ejecutará las veces que sean necesarias hasta que se concrete la comunicación, por lo que no se tomarán datos de los sensores hasta que la NodeMCU se conecte a la red, ya que la ejecución del código nunca saldrá del `void setup()`. A continuación podemos ver el código almacenado en la función `setup()`.

```
void setup()
{
//necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a
tomar
xTaskCreatePinnedToCore(EnvioDatos,"tarea_1",10000,NULL,1,&tarea1,0);// vamos a crear una
tarea que apunte a un nucleo en particular, en este caso el nucleo N°0
/* (le paso la funcion que quiero que se ejecute ,le paso un nombre a esta funcion ,le paso
el tamaño de la pila ,indico algun valor que desee pasarle a la tarea que pare este caso
sera NULL,indico la prioridad , indico el nombre que le dimos a la tarea,indico el nucleo
donde quiero que se ejecute la tarea) */
```

Sistema de monitoreo para heladas

```
Serial.begin(115200); //definimos la velocidad del puerto serial
pinMode(ledWifi,OUTPUT);
pinMode(alimentacion_controlada,OUTPUT);
pinMode(alimentacion_controlada2,OUTPUT);
digitalWrite(alimentacion_controlada,HIGH);
digitalWrite(alimentacion_controlada2,HIGH);
ADXL345.begin();//inicializo la comunicacion con el acelerometro ADXL345
bmp.begin(0x76);//inicializo la comunicacion con el sensor bmp280
dht.begin(); //inicializo la comunicacion con el sensor dht22
if(!display.begin(Direccion_I2C, true)) //inicializamos la pantalla pasandole la direccion
del modulo dentro de un if. Si no se cumple la condicion de TRUE, imprimiremos que hay un
error de conexion a traves del puerto serial
{
    Serial.println("Error de conexion con pantalla OLED");
}

Serial.println("\n");
tic_WifiLed.attach(0.2,parpadeoLedWifi);// hacemos parpadear un led cada 0,2 segundos

WiFi.begin(ssid, password); //Iniciamos la conexion con la red, recibiendo como parametros
el nombre y la contraseña de la red
Serial.print("Conectando..."); //imprimimos que la conexion se esta realizando

while(WiFi.status() != WL_CONNECTED and cont < 50) //mientras no se conecte ira aumentando
un contador
{
    cont++;
    delay(500);
    Serial.print(".");
}

    Serial.println(""); //salto de linea

if(cont < 50) //si entra quiere decir que se conecto
{
    Serial.println("*****");
    Serial.print("Conectado a la red WiFi: ");
    Serial.println(WiFi.SSID());
    Serial.print("IP: ");
    Serial.println(WiFi.localIP());
    Serial.print("macAdress: ");
    Serial.println(WiFi.macAddress());
    Serial.println("*****");
}
else //si entra quiere decir que no se conecto
```

Sistema de monitoreo para heladas

```
{  
  Serial.println("*****");  
  Serial.println("Error de conexion");  
  Serial.println("*****");  
  ESP.restart();  
}  
cont=0;  
tic_WifiLed.detach(); //detenemos el proceso de parpadeo de led con este comando  
digitalWrite(ledWifi,LOW); //para apagar el led  
}
```

Luego nos encontramos con dos funciones principales denominadas `loop()` y `EnvioDatos()`. Cada función ejecuta sus propias tareas, con la particularidad de que no comparten el mismo núcleo, es decir que cada función se ejecuta en el respectivo núcleo designado. De esta forma vamos a reducir el tiempo de procesamiento total, en comparación al tiempo que llevaría ejecutar el código por un solo procesador.

Es importante tener presente que las dos funciones estarán ejecutándose al mismo tiempo de forma independiente, por lo cual deberemos coordinarlas para poder concretar todas las tareas correctamente y esto se debe a que principalmente `loop()` se encargará de leer los datos de los sensores y `EnvioDatos()` se encargará de enviarlos a la base de datos, motivo por el cual si o si primero necesitamos tener la información y luego realizar el envío. Se intentó emplear banderas para poder darle una habilitación a la función `EnvioDatos()`, de forma que espere a que se recaude la información y luego ejecute el envío a la base de datos, pero ocurre un reset indeseado de la placa. Para evitar esto, directamente se coloca un `delay()` para que `EnvioDatos()` espere a que `loop()` recaude la información necesaria.

Vamos a comenzar analizando la función `loop()`. Para poder tener un control de que está haciendo el código y si su ejecución es correcta, imprimimos a través del monitor serial información. Por ejemplo, imprimimos el número del núcleo en cual se está ejecutando esa parte del código, para verificar que correctamente cada núcleo ejecute las tareas que le corresponden. Luego se realiza la comunicación con cada sensor y se guardan los valores que devuelve en variables globales. Vemos que en primer lugar se obtiene información de temperatura y humedad provenientes del sensor DHT22, luego temperatura y presión atmosférica del sensor BMP280 y luego se lee el valor analógico que se está aplicando al GPIO 33. Se procede a imprimir esta información en el monitor serial y luego se establece la

comunicación con el acelerómetro ADXL345. Una vez que se obtiene la información del acelerómetro, se comparan los valores obtenidos con los almacenados como referencia, para determinar si hay presencia de viento o no. Esta información también será mostrada en el monitor serial.

```
void loop()
{
  Serial.println("*****");
  Serial.println("");
  Serial.println("El sensado de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
  temperatura = dht.readTemperature();
  humedad = dht.readHumidity();
  temperatura_bmp280 = bmp.readTemperature();
  presion = bmp.readPressure()/100; // asi como esta obtenemos el dato en hPa
  // analogRead(33); //para leer el valor analogico por el puerto GPIO33 de la ESP32
  bateria = map(analogRead(33), 2730.0f, 4095.0f, 0, 100); //paso el valor medido en volts
a porcentaje, considerando el 100% a los 3,3v
  Serial.println("Temperatura_dht: " + String(temperatura) + " Humedad_dht: " +
String(humedad)+ " Presion_bmp280: " + String(presion)+ " Temperatura_bmp280: " +
String(temperatura_bmp280)+ " Bateria[%]: " + String(bateria));

  sensors_event_t evento;
  ADXL345.getEvent(&evento);
  x = evento.acceleration.x;
  y = evento.acceleration.y;
  z = evento.acceleration.z;
  Serial.print("Eje x = " + String(evento.acceleration.x));
  Serial.print("\t\t\tEje y = " + String(evento.acceleration.y));
  Serial.println("\t\t\tEje z = " + String(evento.acceleration.z));

  //comparamos valores para ver si hay o no presencia de viento
  // la referencia estatica es x=0 y=0 z= -10

  if( -1 < x && x < 1)
  {
    if( -1 < y && y < 1)
    {
      Viento = false;
    }
  }
}
```

Sistema de monitoreo para heladas

```
    }

    if( x > 5 || x < -5)
    {
        Viento= true;
    }
    if( y > 5 || y < -5)
    {
        Viento= true;
    }
    if(z > -9 && z < -4)
    {
        Viento= true;
    }

    if(Viento == true)
    {
        viento = "Hay viento";
        Serial.println("Hay viento");
    }
    if(Viento == false)
    {
        viento = "No hay viento";
        Serial.println("No Hay viento");
    }

    Serial.println("*****");
```

Ya obtenida toda la información proveniente de los sensores, la función EnvioDatos () puede comenzar a trabajar con ella. La función loop () seguirá ejecutando código, ya que ahora debe mostrar la información en el display. Por lo que mientras EnvioDatos () estará procesando la información, en paralelo la función loop() estará mostrando en pantalla el valor de temperatura del DHT22, el valor de humedad del DHT22, la presión atmosférica del BMP280 y el valor en porcentaje de la carga de la batería, durante 2 segundos cada uno. La idea de sumar este modo de visualización en el sistema, ayuda a poder verificar si funciona bien el circuito de sensado, si en algún momento se desconfiara de los datos observados en la telemetría.

Sistema de monitoreo para heladas

```
//*****  
// empiezo a mostrar informacion sensada en la pantalla oled  
  
    display.clearDisplay(); //con este comando borramos cualquier contenido que se este  
mostrando en la pantalla  
    display.setTextColor(SH110X_WHITE);// dependiendo del modelo de pantalla puedo ajustar el  
color  
  
// muestro valor de temperatura  
    display.setTextSize(1);//ajusto el tamaño de la fuente  
    display.setCursor(0,0);  
    display.print("TEMPERATURA: ");  
    display.setTextSize(2);  
    display.setCursor(20,25);  
    display.print(temperatura);// cargamos el valor sensado de temperatura  
    display.print(" ");  
    display.setTextSize(1);  
    display.cp437(true);// esta linea y la siguiente la utilizamos para imprimir el simbolo °  
ya que de otra forma no se podra  
    display.write(167); // el simbolo ° corresponde al caracter 167  
    display.setTextSize(2);  
    display.print("C");  
    display.display(); // con esta sentencia muestro todo lo que coloque anteriormente  
    delay(2000);  
  
// muestro valor de humedad  
    display.clearDisplay(); //con este comando borramos cualquier contenido que se este  
mostrando en la pantalla  
    display.setTextSize(1);//ajusto el tamaño de la fuente  
    display.setCursor(0,0);  
    display.print("HUMEDAD: ");  
    display.setTextSize(2);  
    display.setCursor(20,25);  
    display.print(humedad);// cargamos el valor sensado de humedad  
    display.print(" ");  
    display.setTextSize(2);  
    display.print("%");  
    display.display(); // con esta sentencia muestro todo lo que coloque anteriormente  
    delay(2000);  
  
// muestro presion atmosferica  
    display.clearDisplay(); //con este comando borramos cualquier contenido que se este  
mostrando en la pantalla
```

Sistema de monitoreo para heladas

```
display.setTextSize(1);//ajusto el tamaño de la fuente
display.setCursor(0,0);
display.print("PRESION ATMOSFERICA: ");
display.setTextSize(2);
display.setCursor(0,25);
display.print(presion);// cargamos el valor sensado de presion atmosferica
display.print(" ");
display.setTextSize(2);
display.print("hPa");
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);

// muestro carga de bateria
display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
display.setTextSize(1);//ajusto el tamaño de la fuente
display.setCursor(0,0);
display.print("CARGA DE BATERIAS: ");
display.setTextSize(2);
display.setCursor(35,25);
display.print(bateria);// cargamos el valor sensado de carga
display.print(" ");
display.setTextSize(2);
display.print("%");
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);
// borro la ultima informacion de la pantalla para que no queden pixeles encendidos al
momento de irse a dormir el ESP32
display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
```

Una vez que se muestran los valores deseados en el display, continua la ejecución de las líneas de código que configuran y activan el modo sueño profundo del ESP32. Cuando se activa este modo, el sistema queda en bajo consumo hasta que pasado el tiempo configurado (49 segundos) vuelve a su modo activo y ejecuta el código desde el principio.

Es importante asegurarse que antes de poner a dormir el ESP32, los datos hayan sido enviados efectivamente a la base de datos.

```
//*****
//***** activacion del sueño profundo *****
```

Sistema de monitoreo para heladas

```
contador++; //incremento cada vez que se despierta
Serial.println("*****");
Serial.println("Despierta ESP32, el numero de veces que durmio es: " + String(contador));
Serial.println("");
esp_sleep_enable_timer_wakeup(tiempo); //configuramos el tiempo que deseamos que duerma
Serial.println("Configuramos para que duerma " + String(segundos) +
" Segundos");
Serial.println("*****");

esp_deep_sleep_start(); //con esta sentencia iniciamos el modo deep sleep
Serial.println("Esto nunca deberia imprimirse porque como mande a dormir el micro, cuando
despierta es como si se reseteara la placa, volviendo al principio del setup ");
}
```

Ahora analizaremos la función `EnvioDatos()` la cual se encarga principalmente de comunicarse con el servidor web y ordenar la información para que este pueda recibirla y almacenarla en las tablas correspondientes. Tal como mencionamos anteriormente, primero espera a que la función `loop()` obtenga toda la información necesaria. El código desarrollado dentro de la función `EnviarDatos()` ya fue explicado cuando se desarrolló el código para enviar datos por el método POST, con la salvedad que en este caso se envían algunos datos más pero respetando la misma lógica.

```
// rutina de envio de datos por POST
void EnvioDatos(void *parameter)
{
for( ; ; )
{
    delay(4000); //espera 4seg
    Serial.println("*****");
    Serial.println("");
    Serial.println("El envio de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
    vTaskDelay(10);

    //las sentencias dentro del siguiente if se van a ejecutar solo si la placa se encuentra
conectada a la red wifi
if (WiFi.status() == WL_CONNECTED)
{
HTTPClient http; // creamos un objeto de la clase HTTPClient que se encuentra en HTTPClient.h
```

Sistema de monitoreo para heladas

```
http.begin(url);//con el metodo begin inicializamos el objeto. Le pasamos como parametro una
URL
http.addHeader("Content-Type","application/x-www-form-urlencoded");

/*con esta sentencia, podemos especificar el tipo de contenido de la cabecera
Yo con esta declaracion, puedo enviar los datos de forma estructurada.
*/

String datos_a_enviar ="temperatura= " + String(temperatura) + "&humedad= " +
String(humedad) + "&presion= " + String(presion)+ "&viento= " + String(viento)+ "&bateria= "
+ String(bateria) ;

/* con postData indicamos la informacion que vamos a enviar, y la informacion se envia en
PARES(Variable = Valor)El nombre de la variable se pone entre comillas, cuando se envia mas
de una variable, la siguiente variable se une a la anterior usando el signo ampersand & como
ocurre con "&huedad". La primera va si & Despues de la variable va el valor que se va a
enviar y debe convertirse a variable STRING, porque todo lo que se envia es string Entre
comillas se coloca la etiqueta que voy a enviar, es decir luego en el php debo respetar ese
nombre o no voy a poder entregarle el dato. Con el signo + yo CONCATENO. A continuacion del
+ le paso el valor convertido a string */

int codigo_respuesta = http.POST(datos_a_enviar);
/*llamamos al metodo POST haciendo uso de http.POST y le pasamos una variable STRING La
variable postData son los datos que vamos a enviar Cuando llamamos al metodo POST, el metodo
nos regresa un INT, el cual sera un codigo de respuesta que me servira para saber si los
datos se enviaron correctamente (si regresa 200) o si no pudieron enviarse (si me devuelve un
-1)*/

if (codigo_respuesta>0) //si el codigo de respuesta es mayor a cero lo imprimo por el
puerto serial
{
Serial.println("Código HTTP: "+ String(codigo_respuesta));
if (codigo_respuesta == 200) // si da 200 es una respuesta satisfactoria
{
String cuerpo_respuesta = http.getString();
Serial.println("*****");
Serial.println("*****");
Serial.println("El servidor respondió: ");
Serial.println(cuerpo_respuesta);
Serial.println("*****");
}
}
else // caso contrario imprimo el codigo que entrego en servidor
{
Serial.print("Error enviado POST, código: ");
```

Sistema de monitoreo para heladas

```
    Serial.println(codigo_respuesta);  
  }  
  http.end(); // finalizo la peticion  
}  
delay(5000); //espera 5 segundos  
}  
}
```

CAPÍTULO 5

Energización de circuitos

7 Introducción al capítulo

En el presente capítulo, se abordará el desarrollo del circuito de alimentación que energiza los sistemas conformados por módulos y sensores. La energía será obtenida mediante paneles solares, empleando baterías de litio para almacenarla.

7.1 Desarrollo

En primer lugar, mediremos el consumo energético del sistema. Para esto, emplearemos el siguiente conexionado:

Sistema de monitoreo para heladas

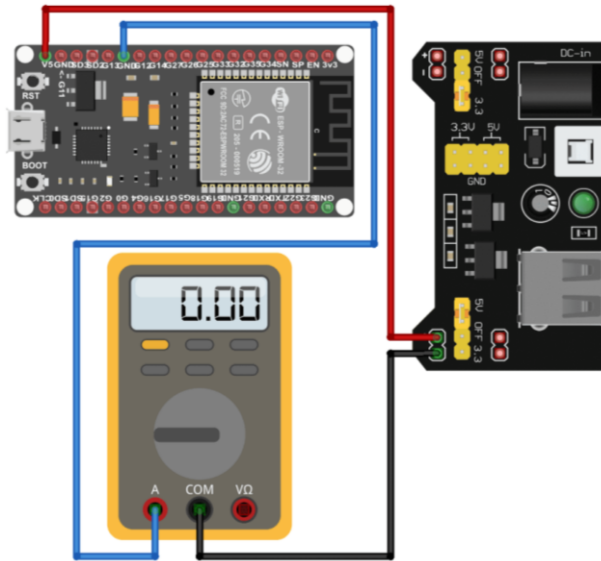


Figura N° 114 **Esquemático empleado para medir consumo de corriente**

En función al consumo determinado, podremos establecer el número y las características de los paneles solares y baterías que emplearemos. En las siguientes figuras, podemos ver el consumo que registra el multímetro cuando está la placa en modo activa, tomando datos y enviándolos a la base de datos, y por otro lado tenemos el consumo del sistema cuando la tarjeta está en modo Deep sleep. De las especificaciones obtuvimos que, funcionando en modo activo, el consumo promedio de la placa NodeMCU es de 80 mAmp estableciendo la conexión wifi, con un límite de consumo máximo de 250 mAmp. Vemos que el consumo más alto es de 125 mAmp, estando en modo activo por no más de 10 segundos. Por otro lado, el fabricante especifica que estando en modo sueño profundo, el consumo es de 7 μ Amp. De la medición realizada, vemos que el consumo mínimo que marca el multímetro es de 0.2 μ Amp durante aproximadamente 50 segundos mientras está durmiendo.

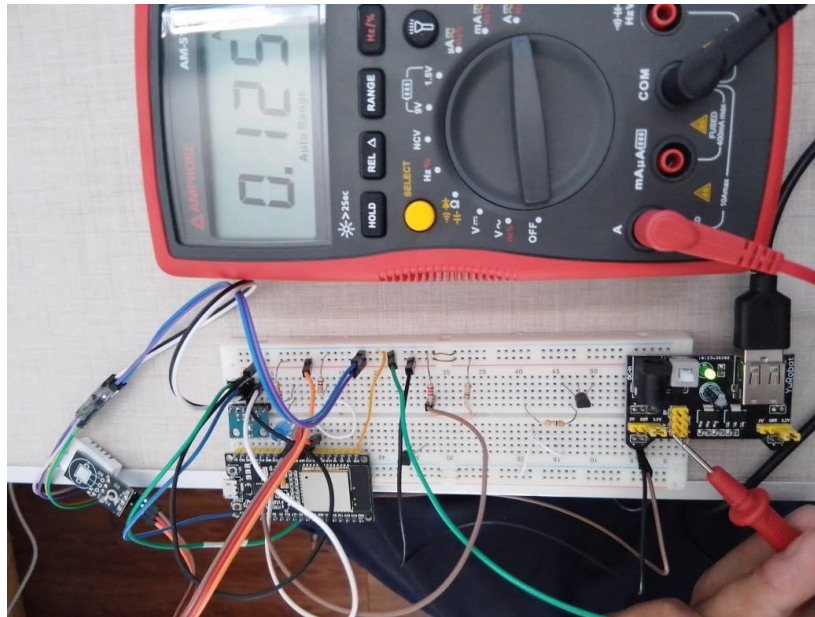


Figura N° 115 Consumo de energía cuando la tarjeta NodeMCU está activa

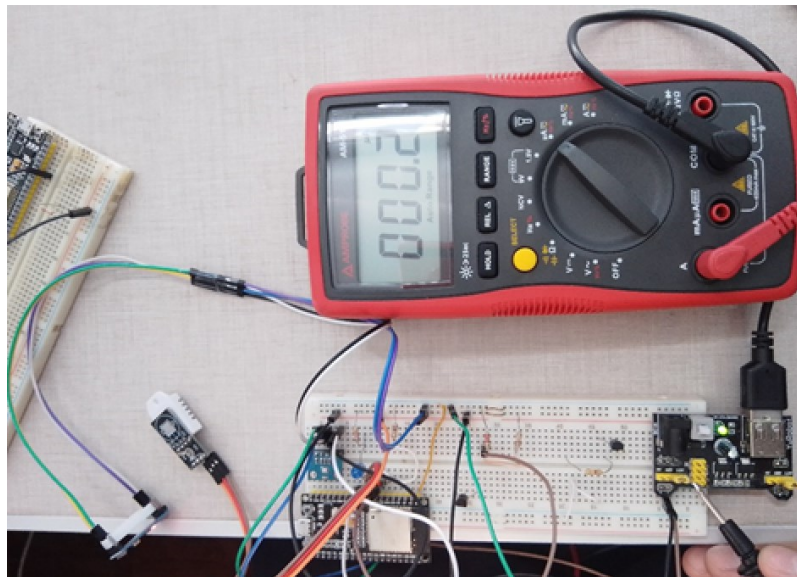


Figura N° 116 Consumo de tarjeta NodeMCU en modo Deep Sleep

Para el diseño, vamos a suponer que el consumo será de 40 mAmp constantes, sin considerar que se activa el modo Deep Sleep, alimentando el sistema con una tensión de 3,3 volt. Si bien vamos a sobredimensionar el cálculo, lo haremos teniendo en cuenta que las

baterías más comunes que se consiguen rápidamente en cualquier comercio son las de litio modelo 18650, las cuales son de 3,7 volts y rondan entre 2000 a 3000 mAh. Para este proyecto se usarán las siguientes, las cuales son de 2600mAh:



Figura N° 117 **Batería de litio modelo 18650**

Teniendo el consumo del circuito y conociendo los mAh de la batería, podemos calcular las horas de energía que vamos a poder tener en funcionamiento el sistema partiendo de la batería cargada completamente:

$$\text{Horas} = \frac{2600 \text{ mAh}}{40 \text{ mAmp}} = 65 \text{ Horas} \equiv 2,7 \text{ días}$$

Esta batería permitirá mantener el circuito funcionando por dos días y medio aproximadamente, considerando condiciones ideales. Debemos tener presente que el uso de estos circuitos será por la noche, en horarios en los cuales no hay luz solar. Por lo tanto, supondremos que la batería deberá cargarse durante el día, considerando que el panel solar recibirá luz durante medio día, es decir 12 horas.

Para cargar este tipo de baterías, emplearemos el módulo de carga TP4056, el cual está diseñado para las baterías de litio modelo 18650. Como podemos observar en la siguiente figura, la corriente que entregará el módulo será como máximo 1 Amp a una tensión de 4 V. Vemos también que la batería debería cargarse en no más de 2 horas, quedando con una tensión en 4,2V aproximadamente.

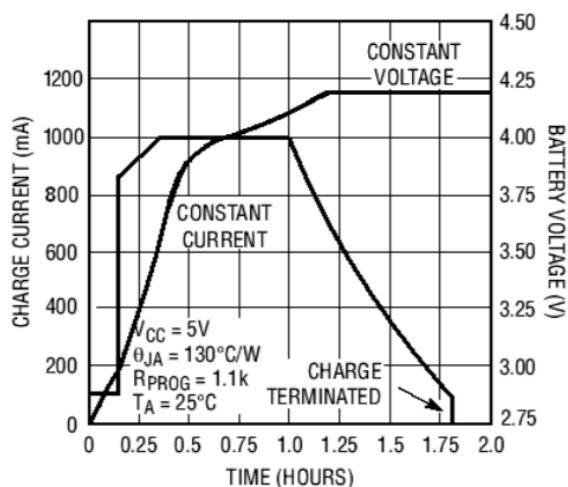


Figura N° 118 Curva de carga

En cuanto a la alimentación del módulo, vemos en la siguiente figura que como máximo tolera una corriente de 500 μ Amp de entrada durante el proceso de carga y una alimentación máxima de 8 voltios.

Para alimentar el módulo, se procederá a utilizar un panel solar de 5 volt que proporciona una corriente máxima de 220 mA ideales, lo cual implica que son de una potencia de 1 W. Según la curva de carga, con los 5 volts de este panel solar, en condiciones ideales debería respetar los tiempos de carga definidos por el fabricante.

The ● denotes specifications which apply over the full operating temperature range, otherwise specifications are at $T_A=25^{\circ}C$, $V_{CC}=5V$, unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
V_{CC}	Input Supply Voltage		● 4.0	5	8.0	V
I_{CC}	Input Supply Current	Charge Mode, $R_{PROG} = 1.2k$ Standby Mode (Charge Terminated) Shutdown Mode (R_{PROG} Not Connected, $V_{CC} < V_{BAT}$, or $V_{CC} < V_{UV}$)	●	150 55 55	500 100 100	μ A μ A μ A
V_{FLOAL}	Regulated Output (Float) Voltage	$0^{\circ}C \leq T_A \leq 85^{\circ}C$, $I_{BAT}=40mA$	4.137	4.2	4.263	V
I_{BAT}	BAT Pin Current Text condition: $V_{BAT}=4.0V$	$R_{PROG} = 2.4k$, Current Mode $R_{PROG} = 1.2k$, Current Mode Standby Mode, $V_{BAT} = 4.2V$	● 450 ● 950 ● 0	500 1000 -2.5	550 1050 -6	mA mA μ A
I_{TRIKL}	Trickle Charge Current	$V_{BAT} < V_{TRIKL}$, $R_{PROG}=1.2K$	● 120	130	140	mA
V_{TRIKL}	Trickle Charge Threshold Voltage	$R_{PROG}=1.2K$, V_{BAT} Rising	2.8	2.9	3.0	V
V_{TRHYS}	Trickle Charge Hysteresis Voltage	$R_{PROG}=1.2K$	60	80	100	mV
T_{LIM}	Junction Temperature in Constant Temperature Mode			145		$^{\circ}C$

Figura N° 119 Especificaciones del TP4056

En resumen, el circuito que se realizará, se representa en el siguiente esquema y se conformará por los siguientes elementos:

- Panel solar 5v
- Módulo de carga de batería TP4056
- Baterías de litio 18650 de 3,7v y 2600 mAh
- Capacitor de 100 uF
- Capacitor de 100 nF
- Resistor de 100Kohm
- Resistor de 27Kohm
- Soporte para baterías
- Regulador de baja caiga MCP1700
- Cables

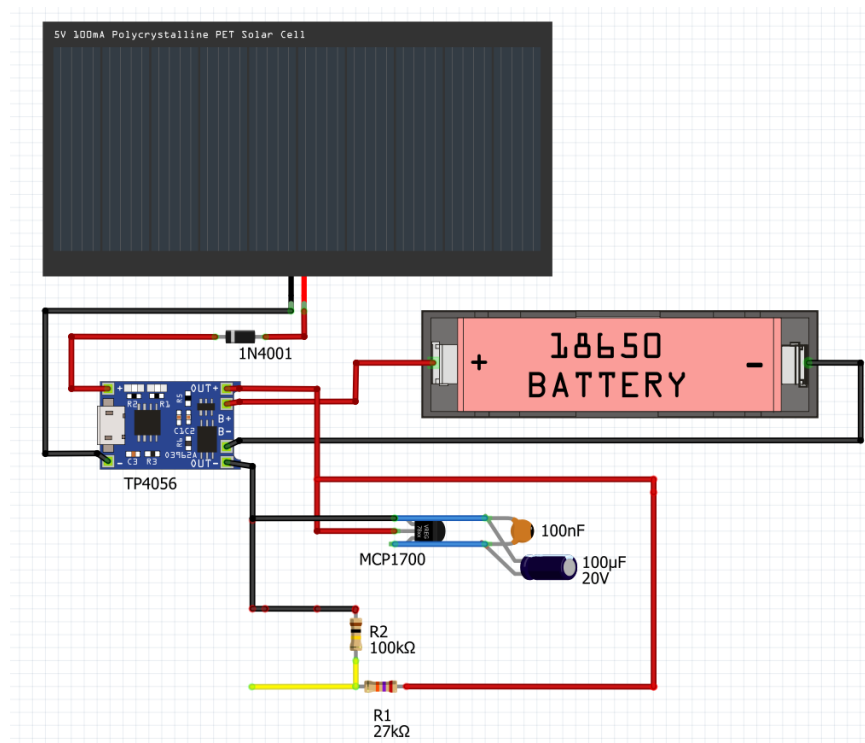


Figura N° 120 Esquemático del circuito de carga

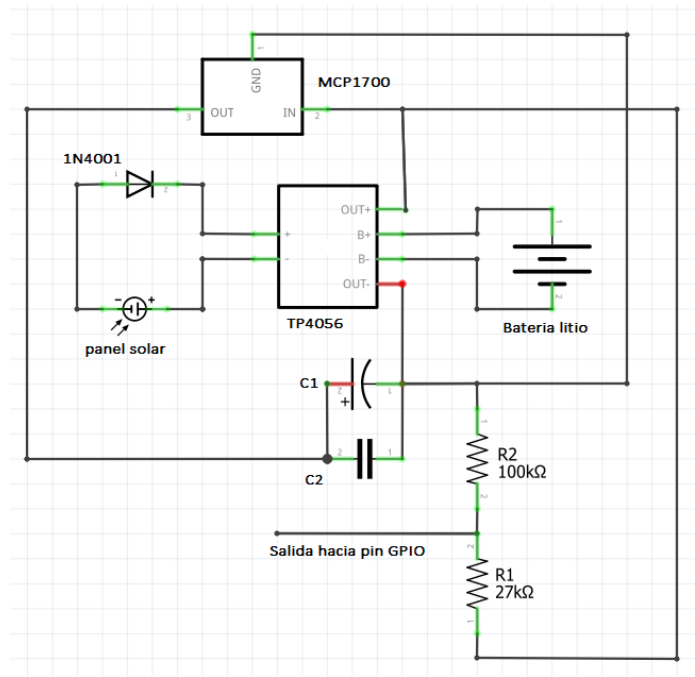


Figura N° 121 **Esquema de conexión eléctrica para
circuito de carga**

En primer lugar, se procedió a energizar el módulo TP4056 haciendo uso de un cable USB conectado a una computadora. Se detecta que cuando no tiene una carga conectada, se enciende un led azul, el cual se emplea para indicar que la batería está cargada. Se obtuvo la tensión entre los bornes B+ y B- en este estado, midiendo un valor de 4,11V, al igual que entre los bornes OUT+ y OUT-.

Se procederá a conectar el panel solar y el diodo, a la entrada del módulo TP4056. Soldamos el pin+ del panel al borne IN+ del módulo, colocando entre ellos el diodo, y luego el pin - del panel al borne IN- del módulo. El diodo rectificador 1N4001, se coloca en el camino que une los terminales positivos de entrada, con el objetivo de solo permitir el paso de la corriente en un sentido. Si bien el fabricante del módulo TP4056 especifica que no es necesario colocarlo, es una buena práctica asegurar el panel solar. La idea es que siempre fluya energía desde el panel hacia el módulo y no desde el módulo hacia el panel. De esta forma, podremos incluso cargar las baterías empleando energía eléctrica, sin necesidad de desconectar el panel solar. En las siguientes imágenes se pueden observar cómo quedó la conexión:

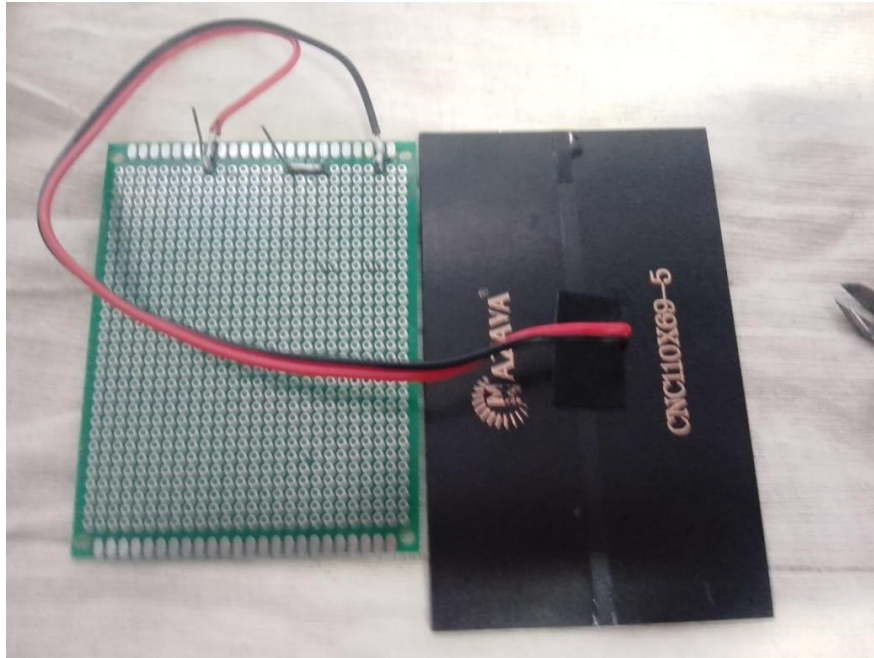


Figura N° 122 Imagen inferior de la conexión entre diodo, panel solar y módulo de carga

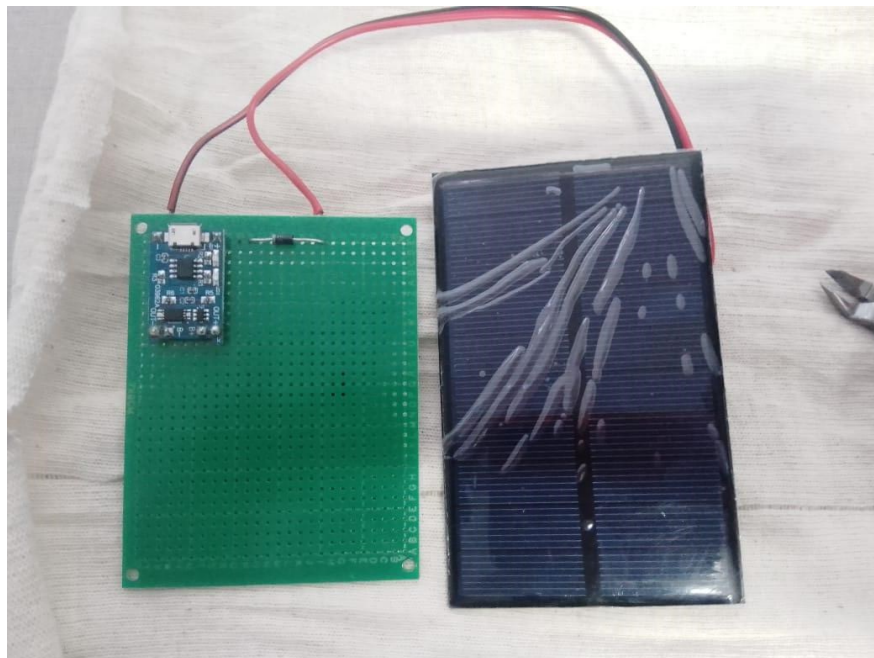


Figura N° 123 Imagen superior de la conexión entre diodo, panel solar y módulo de carga

Luego medimos la tensión de la batería para ver en qué estado de carga se encuentra. Este tipo de baterías de litio suelen producir hasta 4,2 volts cuando están completamente cargadas (aunque tienen 3,7 V marcados en la etiqueta). Pero a medida que la batería se va descargando, el voltaje comienza a caer tal como podemos observar en la siguiente imagen:

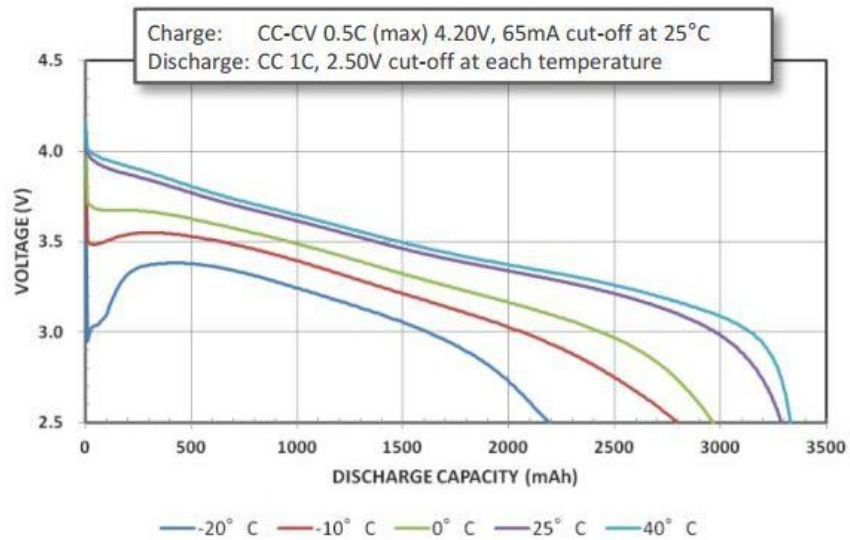


Figura N° 124 Tensión de batería 18650 en función a la descarga

Es interesante establecer el estado de la batería para poder determinar cuánto tiempo tardará en cargarse. Al medir su tensión, podemos observar un valor de 3,81V, lo cual implica que están bastante cargadas, pero no al 100%.



Figura N° 125 Tensión de las baterías antes de ser cargadas

Procedemos a colocar la batería dentro de su soporte, y soldamos este a los bornes B+ y B- respectivamente. Tener presente que, en la imagen anterior el soporte es para dos baterías y esto se debe a que se van a cargar dos al mismo tiempo con un solo módulo TP4056. Si bien el módulo fue diseñado para cargar una sola batería a la vez, las baterías se conectarán en paralelo manteniendo la tensión, pero aumentando la capacidad miliamperios que puede entregar por hora, por lo que solamente debería ocurrir un aumento en el tiempo de carga. Esta implementación se realiza pensando en la alimentación del circuito encargado de controlar el sistema de riego, es decir de aquel que levanta su propio servidor web y controla la activación o no del sistema. Su consumo de energía es mayor que el consumo del circuito que controla los sensores, ya que debe mantener la conexión wifi y suministrar energía para activar el módulo relé sin posibilidad de activar el modo sueño profundo. Para el circuito que controla los sensores, como ya se estableció anteriormente, se emplea una sola batería para alimentar el circuito. El esquemático para una o dos baterías es indistinto, motivo por el cual no se observa una imagen con dos baterías, solo se debe tener presente que para dos baterías la conexión de ellas es en paralelo.

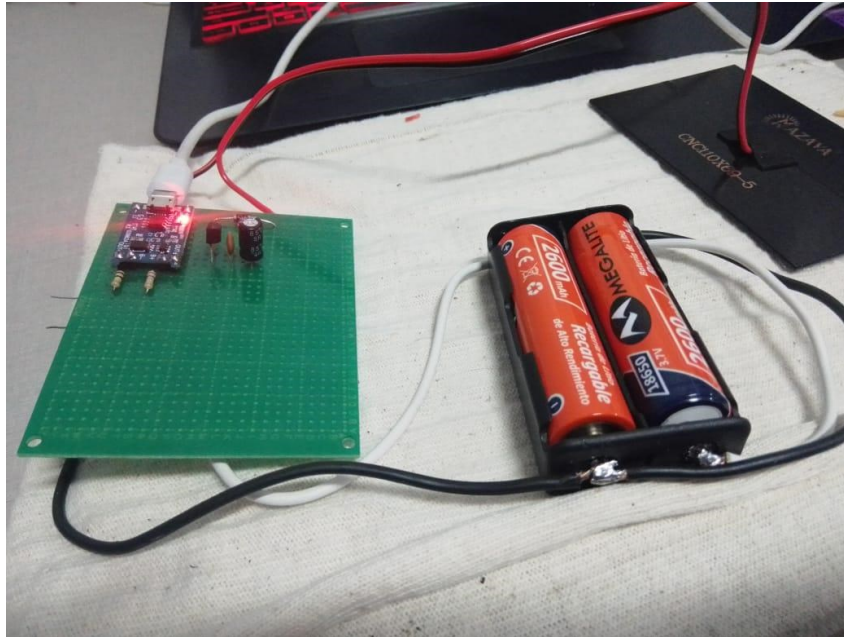


Figura N° 126 **Soporte para dos baterías en paralelo conectado al módulo cargador**

El soporte que se observa en la imagen, originalmente de fábrica viene preparado para cargar baterías en serie, por lo que debió modificarse su cableado para que puedan ser cargadas en paralelo.

Se procedió a cargar la batería haciendo uso de energía eléctrica, para verificar el correcto funcionamiento del circuito y observar cuanto tiempo se tardará en cargar las baterías. Se tardó aproximadamente 2 horas en llevar la tensión de las dos baterías desde 3,81V a 4,2V (tensión de flote) en los bornes B+ y B-, indicando la finalización de carga al apagar el led rojo del módulo TP4056 y encender el led azul. Es importante tener presente que el tiempo que se tarda en cargar las baterías, dependerá tanto de la carga que tengan en el momento que se inicie la carga, como también de la corriente que suministre la fuente. Para esta prueba en particular, se empleó un cargador de celular que entrega 5V y 1,5 Amp.

Continuando con el circuito, debemos tener presente que la tensión de funcionamiento recomendada para la tarjeta NodeMCU es de 3,3 voltios, por lo que no podrá alimentarse directamente desde la batería, ya que recordemos es de 3,7 voltios e incluso completamente cargada llega a los 4,2 volts. Para solucionar este problema, necesitamos emplear un regulador de tensión de baja caída, debido a que un regulador de tensión común necesita una

tensión mayor a la que entrega la batería cargada para poder entregar 3,3 volt. Si usáramos un regulador típico en este circuito, sucederá que cuando la batería comience a descargarse, el regulador dejará de funcionar. Por otro lado, un regulador de tensión de baja caída si puede funcionar correctamente con valores de tensión muy cercanos a los que desea entregar en su salida, eliminado el problema que podría ocurrir cuando la batería comience a descargarse.

Tal como se mencionó anteriormente, el regulador de baja caída que emplearemos será el MCP1700 cuyo pinout es el siguiente:

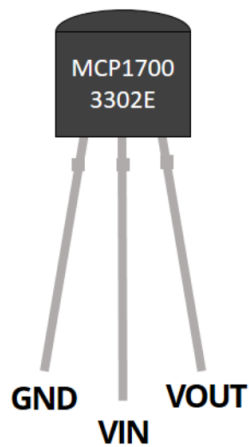


Figura N° 127 Pinout de regulador de tensión MCP1700

Según las especificaciones del fabricante, puede entregar hasta 250 miliamperios para tensiones de salida mayores a 2,5 volts, lo cual es aceptable para funcionar correctamente con una carga como el circuito diseñado. Para suavizar los picos de tensión, colocaremos un capacitor cerámico 100nF y un capacitor electrolítico de 100uF conectados en paralelo a GND y Vout. En la siguiente imagen podemos observar el circuito regulador a 3,3 voltios montado en la placa junto con el circuito cargador de batería:

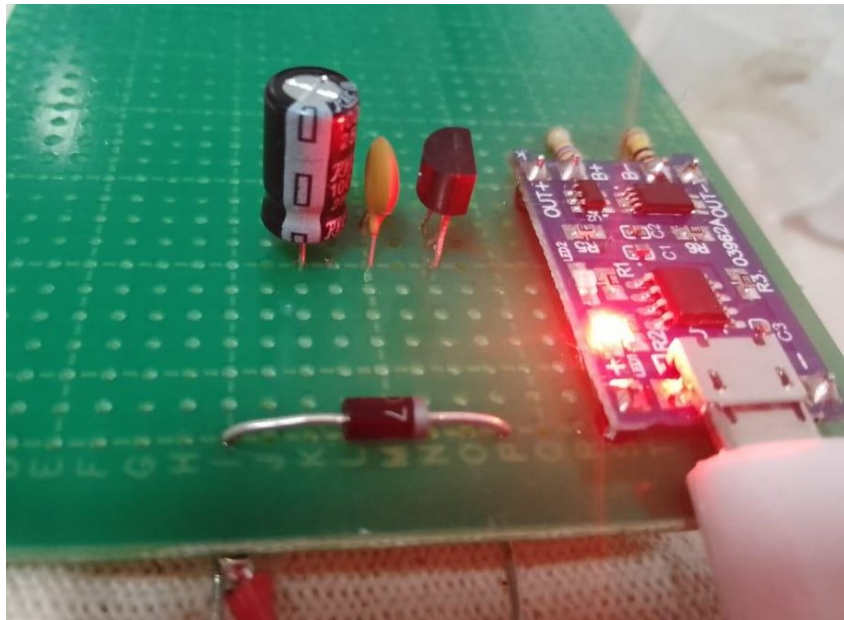


Figura N° 128 **Circuito regulador a 3,3V montado con el circuito de carga**

Se procedió a verificar el correcto funcionamiento del regulador, midiendo la tensión de salida del regulador MCP1700 estando las baterías en su tensión de flote, obteniéndose un valor de 3,308V.

Hasta el momento, el circuito montado es el observado en la siguiente imagen:

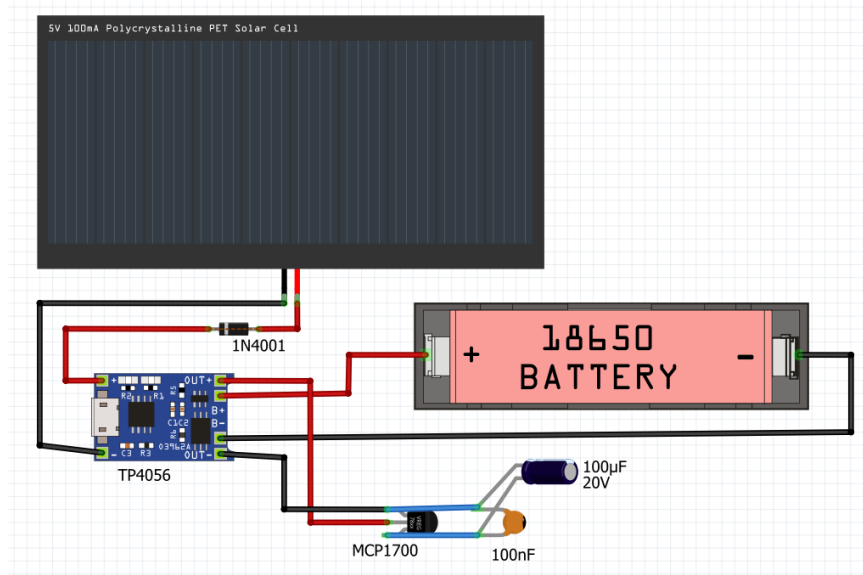


Figura N° 129 **Esquema parcialmente completo**

Para poder tener información sobre cuál es la carga que tiene la batería, se procederá a medirla empleando un divisor resistivo. Para el cálculo de los valores del divisor, debemos partir sabiendo que el máximo valor de tensión que puede tomar este modelo de batería es de 4,2V y que el máximo valor posible de aplicar a un pin analógico de la NodeMCU es de 3,3V. También debemos tener presente que el menor valor que podrá tomar la batería será de 2,8V. Esta información es importante de tener en cuenta al momento de asociarla con la resolución del ADC empleado, ya que nos permitirá establecer el rango equivalente al 0 y 100% de carga.

Para el cálculo, podemos establecer un valor de resistencia fijo y despejar el otro valor para poder realizar la conversión necesaria. Fijando R1 como 100Kohm, el valor de R2 lo podemos encontrar a partir de la siguiente ecuación:

$$R2 = R1 \left(\frac{V_{out}}{V_{in} - V_{out}} \right)$$

$$R2 = 100 \text{ Kohm} \left(\frac{3.3 \text{ V}}{4.2 \text{ V} - 3.3 \text{ V}} \right)$$

$$R2 = 27,272 \text{ Kohm}$$

Por lo tanto, el divisor resistivo será implementado por los valores comerciales de 100Kohm y 27Kohm para R1 y R2 respectivamente. Haciendo uso del divisor, logramos obtener a la salida del mismo, como máximo 3,3V para una entrada de 4,2V y una salida de 2,2V para una entrada de 2,8V asegurándonos de no dañar la placa. Si bien podría conseguirse el mismo efecto con valores resistivos más chicos, se emplean valores altos con el objetivo de tomar bajos niveles de corriente tanto para proteger la placa como para no generar un consumo significativo de la energía de las baterías. Tener presente además que las resistencias comerciales tienen un determinado porcentaje de tolerancia, siendo apropiado emplear para esta medición resistores con la mejor tolerancia posible.

En la siguiente figura podemos observar cómo se realizará la alimentación del circuito de energía a través de los pines 3V3 y GND, usando el pin de entrada analógico GPIO 33 para obtener la tensión que entrega el divisor resistivo.

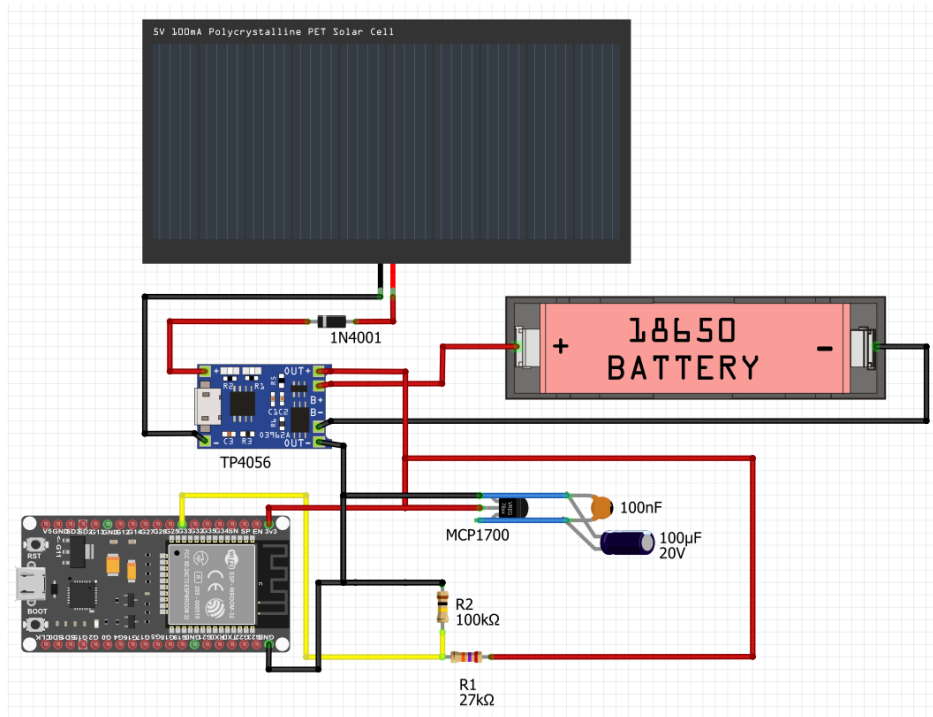


Figura N° 130 **Circuito de energía alimentando conectado a NodeMCU**

Cuando la batería esté completamente cargada se medirán 3,3V, los cuales serán tomados como referencia para indicar que se encuentra al 100% de su carga. Esta información será visualizada en tiempo real en la telemetría, para que el operador pueda conocer el estado de carga del sistema. En la práctica, el valor medido con el multímetro sobre R2 es de 3,264V si se quiere ser más preciso en la representación del 100% de la carga. En la siguiente imagen, podemos ver el divisor resistivo montado en conjunto con el circuito de carga de baterías y el circuito regulador a 3,3V:

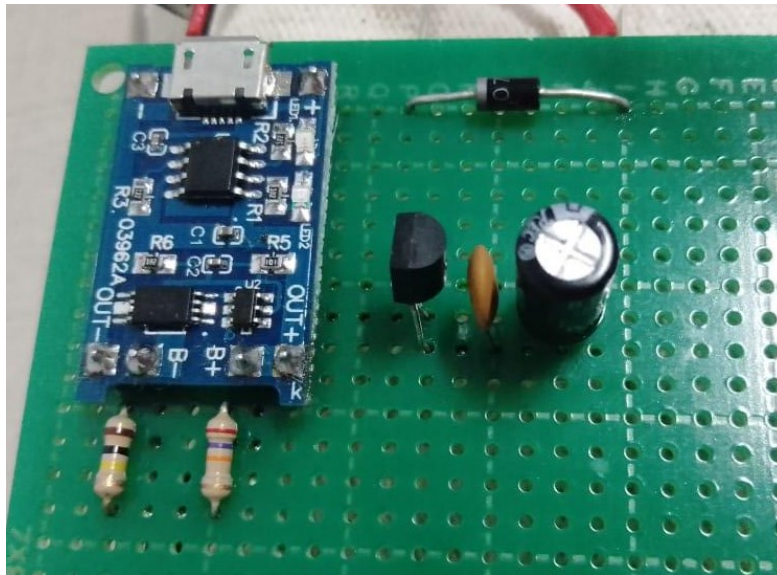


Figura N° 131 **Divisor resistivo en conjunto con el resto del circuito de carga y circuito regulador de 3,3V**

Para poder leer tensión a través del pin GPIO 33, debemos invocar a la sentencia `analogRead(33)` y para convertir los valores a porcentaje usaremos la siguiente sentencia `map(analogRead(33) , 2730.0f , 4095.0f , 0, 100)` y la igualamos a una variable float para almacenar el porcentaje. La función `map` nos permitirá convertir una lectura a porcentaje, pasándole el valor de la conversión que se tomará como referencia de 0% y cual como 100%. Para este caso, el 100% lo tendremos cuando tengamos 3,3V aplicados al pin GPIO 33, valor que el ADC codifica como 4095. Por otro lado, el 0% lo tendremos cuando tengamos 2,2V aplicados al pin GPIO 33, valor que el ADC codifica como 2730. Estableciendo los límites mencionados, podremos obtener la carga de la batería en porcentaje.

7.2 Circuito ON-Off para manejar alimentación del módulo:

Se procederá a desarrollar un circuito switch, es decir del tipo ON-OFF, para poder controlar la alimentación del módulo relé. De esta manera podremos decidir a través de una de las salidas digitales de la placa, cuando energizar el módulo, evitando así cualquier tipo de consumo extra de las baterías. Para llevar a ejecutar el circuito usaremos un transistor, el cual trabajara en corte y saturación, siendo controlado por la NodeMCU. Con esto también evitaremos tomar altos niveles de corriente de la placa, ya que con bajos niveles de corriente es posible saturar el transistor.

De las especificaciones del módulo relé que vamos a utilizar, vemos que para alimentarlo necesitaremos una tensión de 5V, una corriente de aproximadamente 80 mAmp y que la impedancia de la bobina es de aproximadamente 63 ohm. Partiendo de los datos de tensión y corriente es que seleccionamos un transistor apropiado, que pueda trabajar con esos niveles sin ser destruido.

El transistor 2N2222 es uno de los transistores más económicos y que se acopla perfectamente a las especificaciones de corriente y tensión mencionadas, siendo posible de tolerar una corriente de colector de 800 mAmp y una tensión de 60 V como valores máximos.

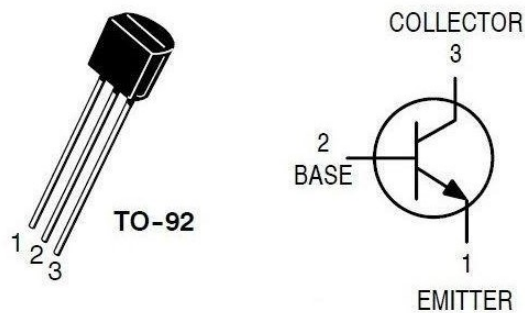


Figura N° 132 Pinout del transistor 2N2222

En la siguiente figura podemos ver el esquema que vamos a diseñar, donde vemos al transistor empleado en configuración emisor común ya que es una de las más básicas. También emplearemos dos resistencias R1 y R2, donde R1 es la resistencia limitadora de base, con la cual evitaremos tomar elevados niveles de corriente desde la placa y R2 es la resistencia de pull down que se encargará de mantener el cero cuando no se esté aplicando tensión desde la salida de la NodeMCU. Podemos ver en el colector un diodo freewheeling,

con el cual protegeremos al transistor contra una sobretensión. Este diodo se colocará en paralelo a la carga inductiva, es decir a la bobina, evitando que se produzca una sobretensión cuando el transistor pase desde la saturación al corte. De esta forma, la corriente que venía circulando por la bobina va a continuar circulando por el diodo, con lo cual la bobina se descarga a través de él. En cuanto al diodo, se procedió a seleccionar un 1N4001 ya que soporta 50V de tensión y 1 Amp de corriente. Vemos que los cables de color negro de GND y IN1 del relé se conectan ambos al colector del transistor. Esto se debe a que la conexión a tierra recién será concretada cuando el transistor pase a saturación, es decir cuando le lleguen 3,3V a la base. De esta forma me aseguro que el módulo relé sea energizado y accionado solo cuando el transistor entre en conducción. Los 3,3V serán controlados por el pin GPIO 26 de la placa NodeMCU, que tal como se puede ver es el que comunica el cable blanco con la resistencia de base. También se puede observar el pin GPIO 27 conectado a un led rojo, el cual será utilizado como indicador del estado de conexión entre la placa y la red wifi, tal como se explicó anteriormente. Por último los cables rojo y azul representan la conexión a + y – de la fuente de tensión utilizada para la prueba.

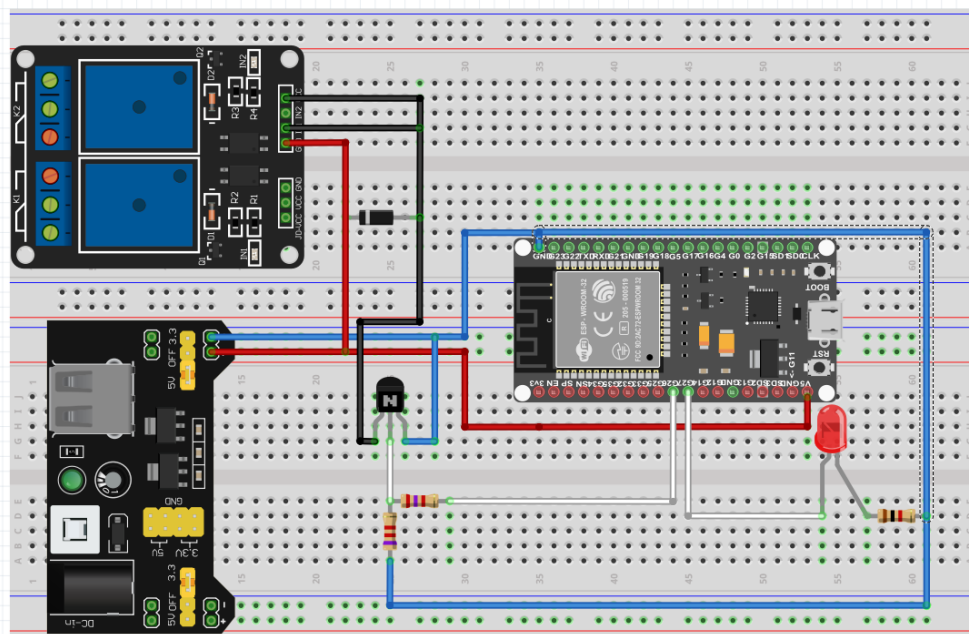


Figura N° 133 Esquema del circuito a realizar

Para seleccionar los valores de R1 y R2 debemos realizar los cálculos de polarización del circuito. En primer lugar, vamos a calcular la corriente de base que necesitamos, partiendo de cuál es la corriente de colector que deseamos. Siendo que en el colector tendremos los 80 mA que consume el relé y conociendo la ganancia de corriente del transistor que es de 100, podemos calcular la corriente de base:

$$I_b = \frac{I_c}{H_{fe}} = \frac{80mA}{75} = 1.066 mA = 1mA$$

Para calcular R2 partimos de que se estima que la corriente que circula por ella es de aproximadamente 10 veces menos que la corriente Ib:

$$I_{R2} = \frac{I_B}{10} = \frac{1 mA}{10} = 100 \mu A$$

Conociendo la corriente en R2 y obteniendo de la hoja de datos la tensión de la juntura base emisor Vbe, podemos calcular R2 como:

$$R2 = \frac{V_{be}}{I_{R2}} = \frac{0,7 V}{100 \mu A} = 7K\Omega$$

La corriente en R1 será la suma entre la corriente de base y la corriente en R2:

$$I_{R1} = I_{R2} + I_b = 0,1 mA + 1mA = 1,1 mA$$

Conociendo IR1 podemos determinar el valor de R1, el cual es muy importante debido a que, si seleccionamos un valor muy grande, nunca lograremos llevar a saturación el transistor. Por otro lado, si seleccionamos un valor muy chico podríamos tomar demasiada corriente de la placa NodeMCU y quemar el integrado. Llamaremos Vmc a la tensión que entregará el microcontrolador en su pin de salida:

$$R1 = \frac{V_{mc} - V_{be}}{I_{R1}} = \frac{3,3 V - 0,7 V}{1,1mA} = 2363 K\Omega$$

Dentro de los valores comerciales que se pudieron conseguir, se reemplazó el valor de 2363Kohm para R1 por la resistencia de valor comercial 2,7Kohm y el valor de 7kohm para R2 se reemplazó por el valor comercial de 7,2Kohm.

En la siguiente imagen, se puede ver el circuito montado en una protoboard, donde se procedieron a realizar pruebas y mediciones para verificar el óptimo funcionamiento.

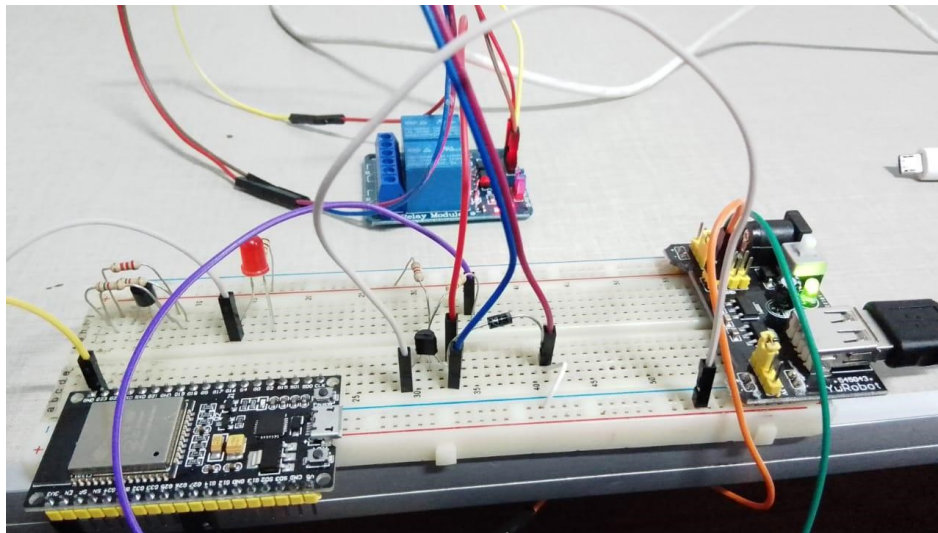


Figura N° 134 **Imagen del circuito montado en una protoboard**

Una vez armado el circuito, se procedió a realizar mediciones, en búsqueda de verificar si el circuito funciona correctamente. Para realizar la medición de una forma cómoda, se reemplazó el relé por una resistencia de 63 ohm, valor similar a la impedancia que presenta la bobina del relé.

Parámetro	Valor
Corriente por R1	1 mAmp
Corriente de colector	63 mAmp
Tensión colector-emisor	86 mV
Consumo de corriente	105 mAmp

Tabla N° 8 **Mediciones realizadas en circuito On-Off controlador de módulo relé**

Sistema de monitoreo para heladas

CAPÍTULO 6

Montaje en placa y pruebas

8 Introducción:

En el presente capítulo, se procederá a realizar el montaje de todos los componentes en sus respectivas placas, realizando pruebas para ir verificando que todo funcione correctamente. Se procederán a realizar 2 placas, una para la parte de sensado y otra para la parte de accionamiento, dejando ambas energizadas por baterías y energía solar.

8.1 Montaje y prueba de placa controladora de sistema de riego

Procederemos a realizar el montaje sobre la misma placa que contiene el circuito de carga para las baterías y regulación de tensión. Como se puede observar en la siguiente imagen, el circuito se conforma por el módulo cargador de batería TP4056, soportes de baterías, panel solar, divisor resistivo para la medición de la carga de batería y por último un regulador de baja caída para entregar una tensión fija de 3,3V, que será utilizada para energizar la NodeMCU. Tener presente que el diseño y montaje de este circuito ya fue explicado y diseñado en el capítulo 5, por lo que se puede consultar a él para un análisis más detallado.

El circuito esquemático que se observa en la siguiente imagen es el que se pasará a placa, teniendo presente que el módulo de energía no se incluirá:

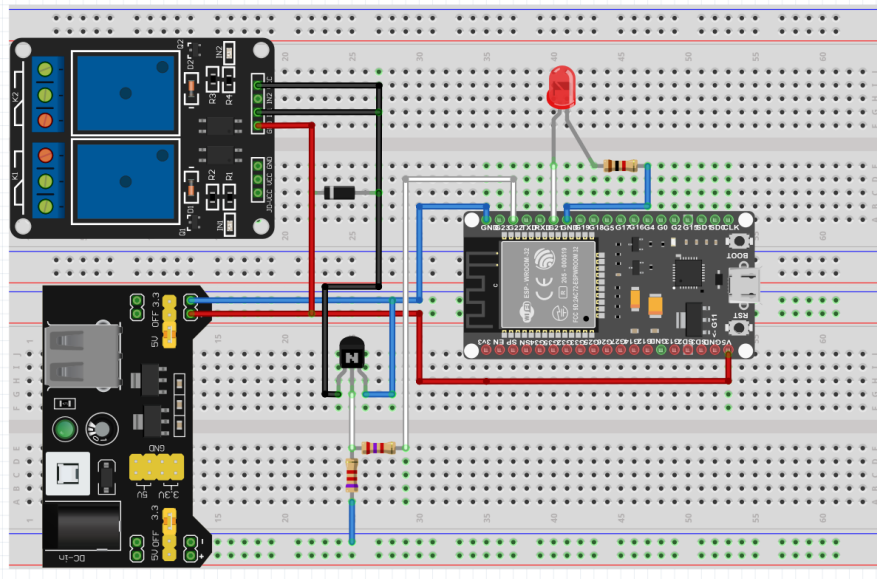


Figura N° 135 Esquemático del circuito controlador de sistema de riego

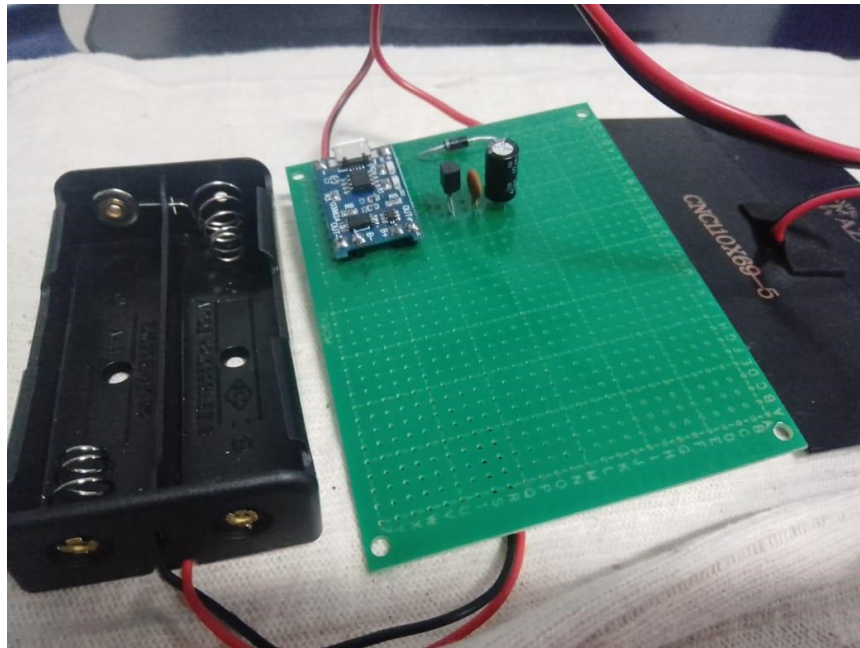


Figura N° 136 **Circuito cargador de batería a partir de energía solar o eléctrica**

Partiendo del circuito anterior, primero se procedió a montar el convertidor DC-DC Step Up, con el cual elevaremos la tensión que entrega el regulador de baja caída a 5V, ya que el módulo relé no funciona con 3,3V. El módulo Step UP es el siguiente:



Figura N° 137 **Modulo Step Up 3,3V a 5V**

Una vez añadido a la placa, se realizó una medición de tensión entre Vin-GND y Vout-GND. En las siguientes imágenes se puede observar el resultado de medición, donde se obtiene que la tensión de salida de este módulo es de 5,7V para una entrada de 3,309V.

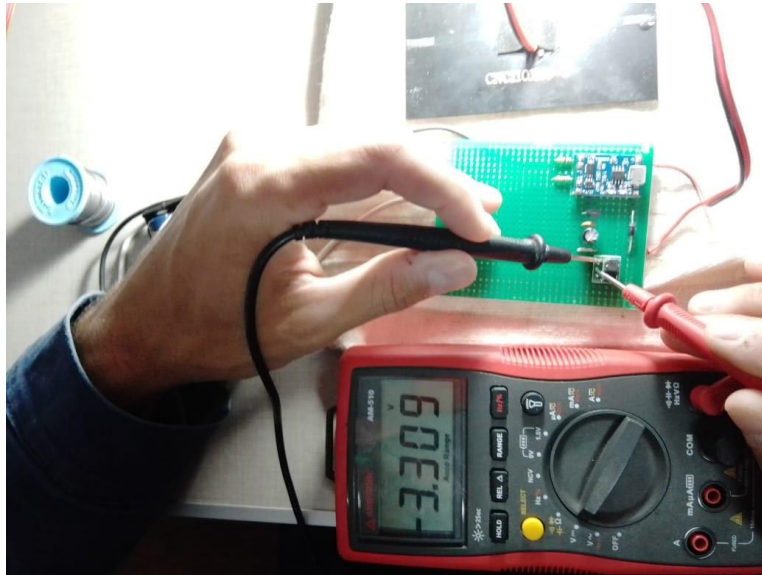


Figura N° 138 Medición de tensión aplicada al módulo Step Up

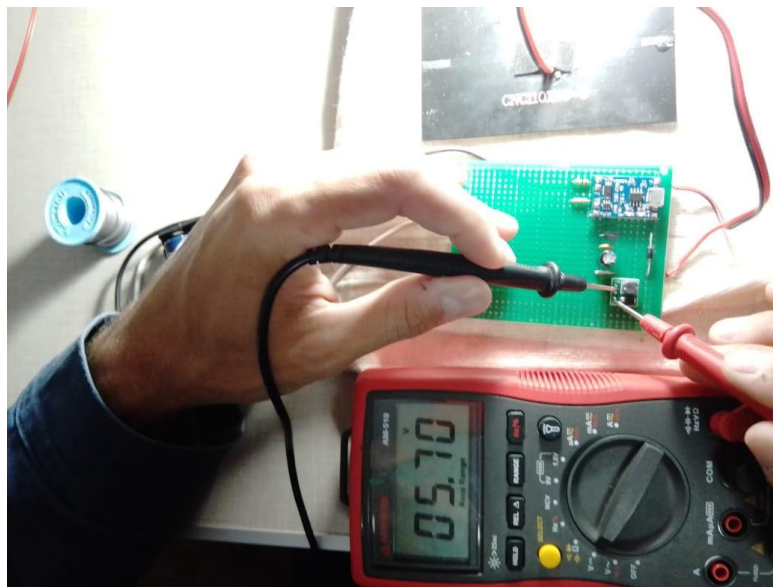


Figura N° 139 Medición de tensión de salida del módulo Step Up

Si bien la tensión de salida de 5,7V supera a los 5V ideales con los cuales funciona el módulo, se corroboró en la práctica que puede funcionar correctamente con ese nivel de tensión, evitando tener que emplear un regulador de tensión.

Continuando con el montaje, se añadió el circuito on-off que controla el encendido o apagado del módulo relé, compuesto por un transistor 2N2222, diodo 1N4001 y dos resistencias de base (2,7Kohm y 7,2Kohm). En la siguiente imagen se pueden observar los componentes mencionados dentro del sector marcado con línea continua roja, como también así el led indicador de conexión con su resistencia a tierra dentro del sector marcado con línea continua azul.

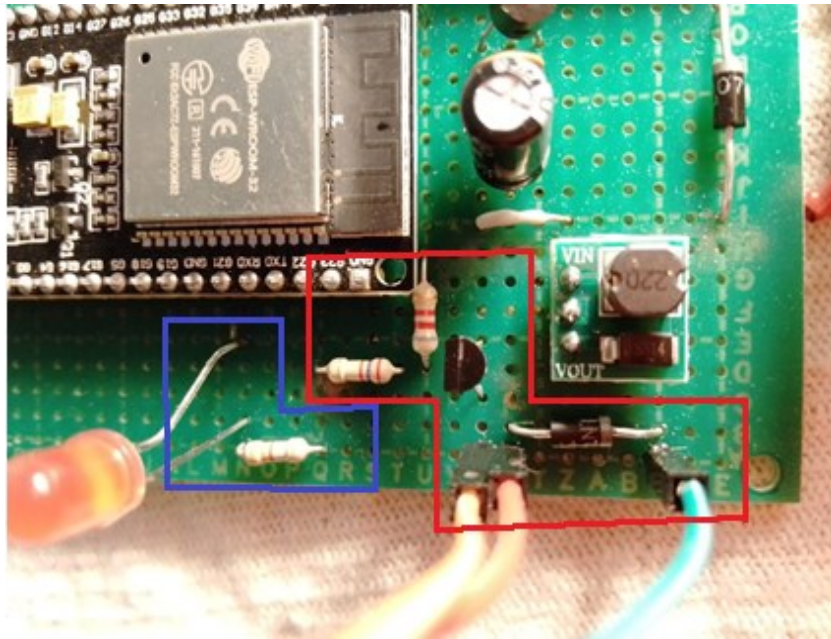


Figura N° 140 **Imágen ampliada con circuito On-Off y led indicador**

Los cables que se aprecian en la figura anterior, comunican la placa con el módulo relé. Tal como se puede observar a continuación, el cable rojo y naranja se conectan al GND y IN1 del módulo relé, ya que son aquellos que se encuentran en contacto con el colector del 2N2222, quedando a GND cuando este entre en saturación. Por otro lado, el cable de color verde está conectado a la salida del módulo Step Up, llevando los 5,7V al pin Vcc del relé.

Sistema de monitoreo para heladas

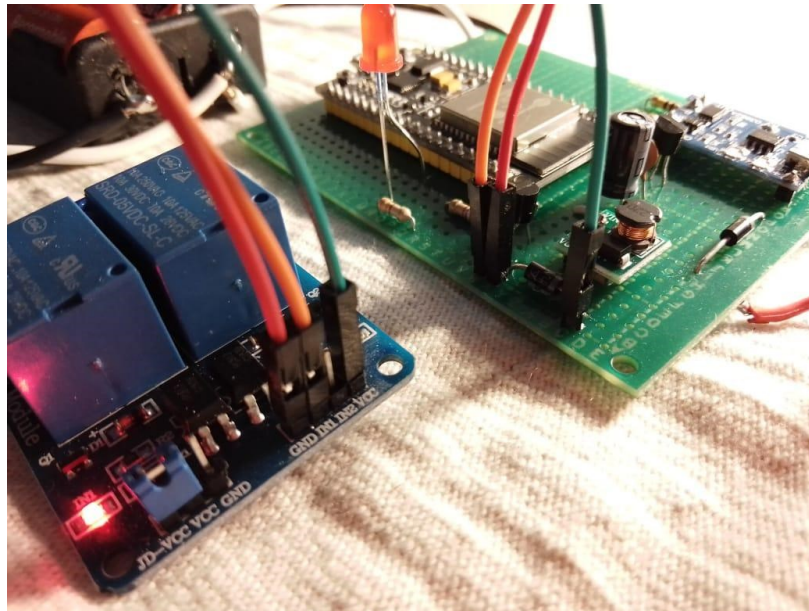


Figura N° 141 Conexión entre la placa y el módulo relé

Por último, se añadió la tarjeta NodeMCU ESP32, la cual se alimentó con los 3,3V que entrega el regulador de baja caída MCP1700.

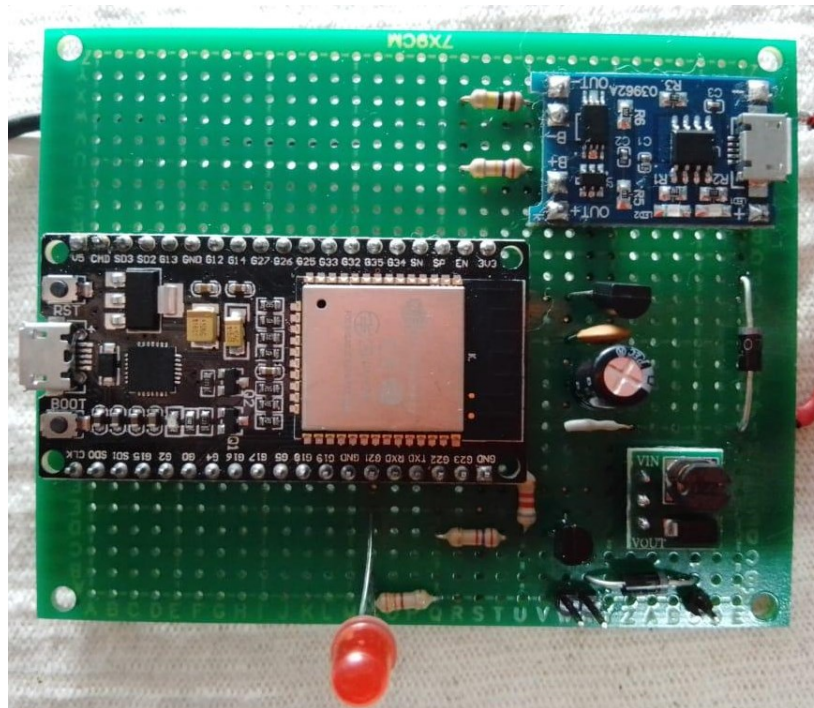


Figura N° 142 Vista superior de placa controladora de sistema de riego finalizada

Sistema de monitoreo para heladas

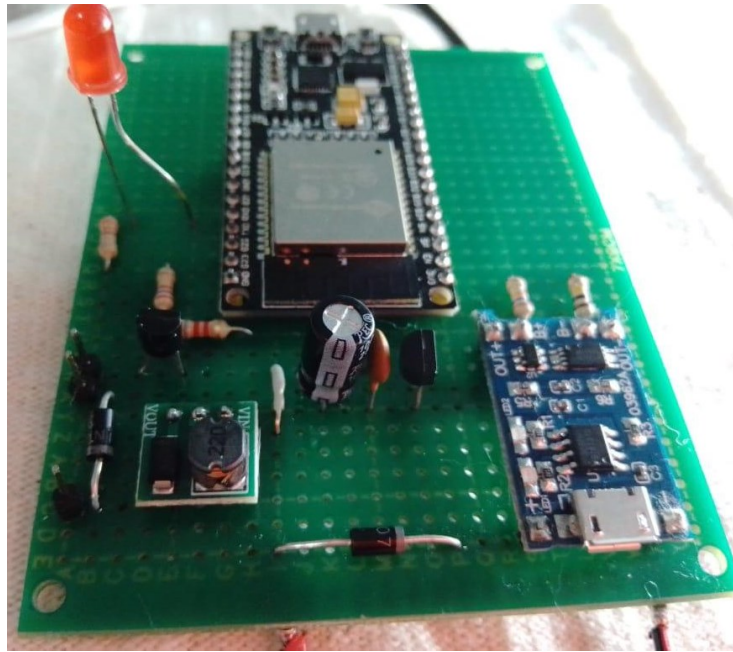


Figura N° 143 Vista lateral de placa controladora de sistema de riego finalizada

En la siguiente imagen podemos ver el sistema completo conformado por la placa, el panel solar, las baterías y el módulo relé:

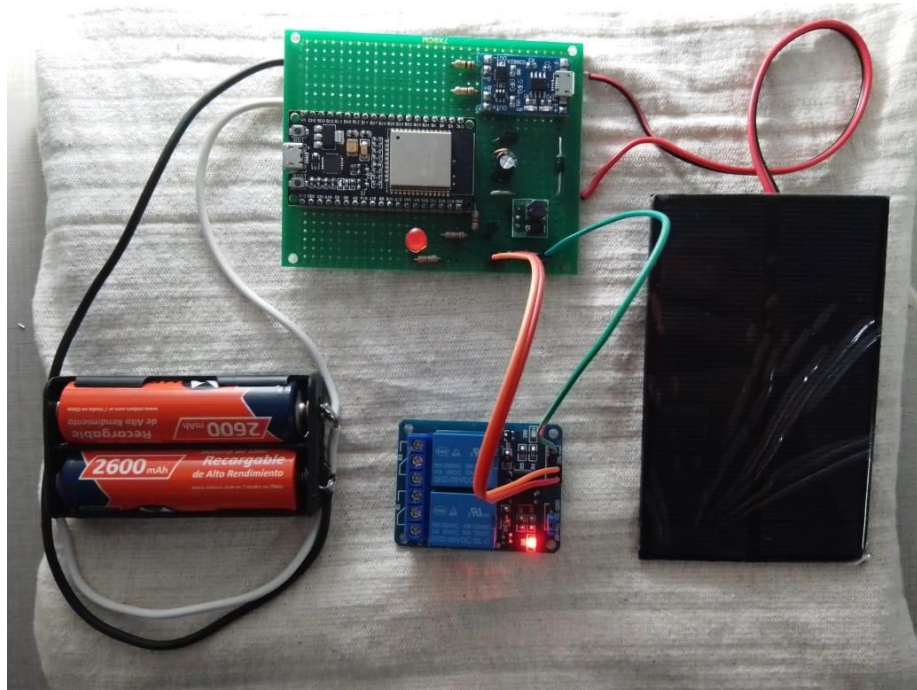


Figura N° 144 Circuito de control de sistema de riego completo en placa

Partiendo con ambas baterías completamente cargadas, indicando una tensión de 4,2V y manteniendo el relé todo el tiempo energizado, se verificó un tiempo de funcionamiento de aproximadamente 18 horas, verificando una tensión de 3V pasado dicho tiempo. Hasta ese valor de tensión, se verificó un correcto funcionamiento del circuito regulador de tensión, el cual, si bien entregó 2,8V en su salida, permitió mantener en funcionamiento la NodeMCU. El problema apareció en el módulo relé, el cual se desactivo y no volvió a responder ante las indicaciones realizadas por medio de la página. Las baterías aún pueden seguir entregando energía, pero debemos tener presente que el regulador de tensión de baja caída ya no podrá mantenerse en funcionamiento. Debido a esto, si pasadas las 18hs de uso, las baterías no reciben carga, posiblemente la NodeMCU se apague o bien el circuito interno de protección propio de las baterías, evitará que se sigan descargando y por lo tanto dejará de suministrar energía a la carga.

Por otro lado, se procedió a realizar una prueba de tiempo de carga, la cual consistió en partir de las baterías en un valor de carga que sólo permitiera mantener la conexión wifi, sin poder activar el módulo relé. En la siguiente tabla, podemos encontrar los valores de tensión que se fueron midiendo en las baterías, partiendo desde el valor sin energizar el módulo de carga. Luego comenzamos registrando el tiempo que llevo alcanzar a un determinado valor de tensión, teniendo presente que el circuito se inició con su máximo consumo de energía, es decir, conexión wifi establecida y relé activo. En la siguiente tabla tenemos información de los tiempos acumulados de carga y las tensiones que se fueron relevando en cada momento:

Tiempo acumulado de carga	Tensión
Sin conectar alimentación al módulo de carga	2,85V
00:00 Hs	3,16V
00:30 Hs	3,58V
01:00 Hs	3,74V
01:45 Hs	3,76V

02:15 Hs	3,80V
02:45 Hs	3,90V
03:15 Hs	4.00V
03:45 Hs	4,05V
04:15 Hs	4,10V
05:15 Hs	4,13V
06:00 Hs	4,20V

Figura N° 145 **Prueba de tiempo de carga de baterías para un máximo consumo**

De los datos observados, vemos que en la condición establecida llevo 6 horas cargar las baterías, lo cual es un resultado aceptable si pensamos en que un solo módulo está pensado para cargar una sola batería en poco más de dos horas y dependiendo de la corriente que pueda suministrar la fuente de energía. En este caso tenemos dos baterías en paralelo, por lo cual debemos suministrar el doble de energía a la carga.

8.2 Montaje y prueba de placa controladora de sensores

Procederemos a realizar el montaje sobre la misma placa que contiene el circuito de carga para las baterías y regulación de tensión. Como se puede observar en la siguiente imagen, el circuito se conforma por el módulo cargador de batería TP4056, soportes de baterías, panel solar, divisor resistivo para la medición de la carga de batería y por último un regulador de baja caída para entregar una tensión fija de 3,3V, que será utilizada para energizar la NodeMCU. Tener presente que el diseño y montaje de este circuito ya fue explicado y diseñado en el capítulo 5, por lo que se puede consultar a él para un análisis más detallado.

El circuito esquemático que se observa en la siguiente imagen es el que se pasará a placa, teniendo presente que el circuito de energía no se incluirá:

Sistema de monitoreo para heladas

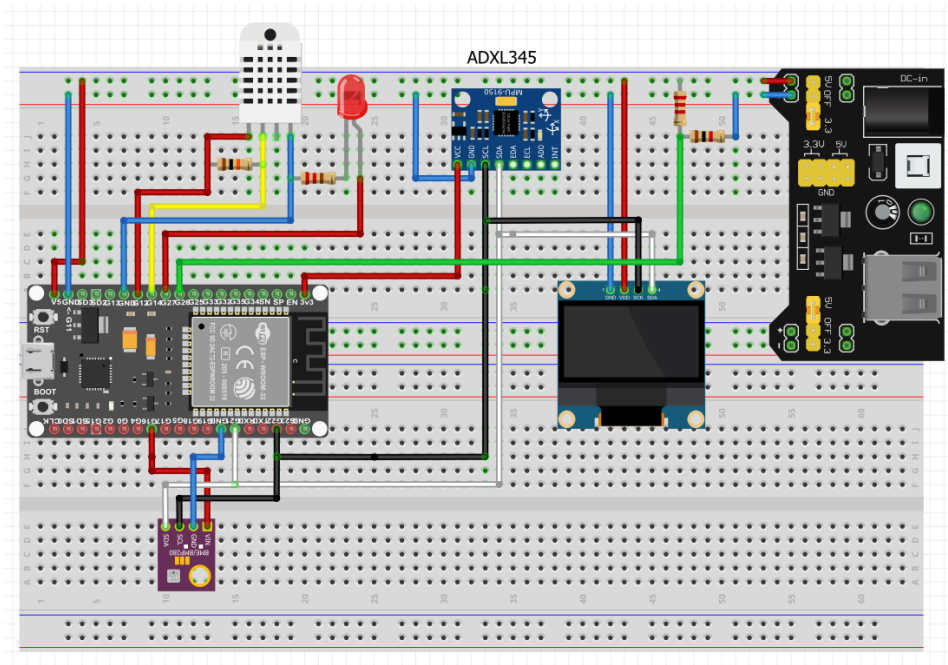


Figura N° 146 Esquemático del circuito controlador de sensores

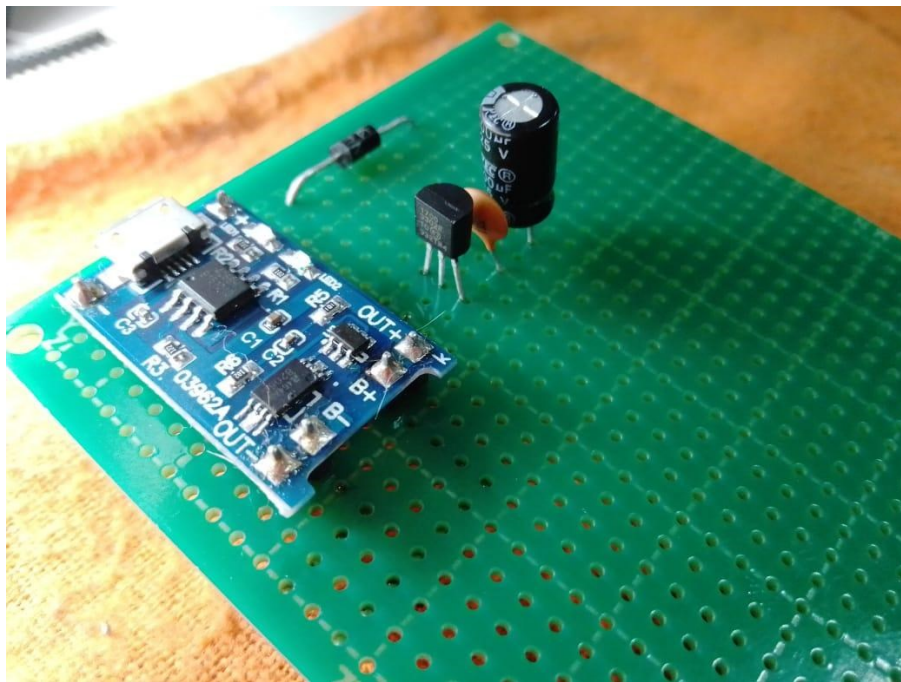


Figura N° 147 Circuito de carga de batería en placa sin panel solar y sin soportes de baterías

En primer lugar, se procedió a montar el módulo BMP280, y se colocó una regleta de 4 pines para poder realizar la conexión con el módulo acelerómetro ADXL345:

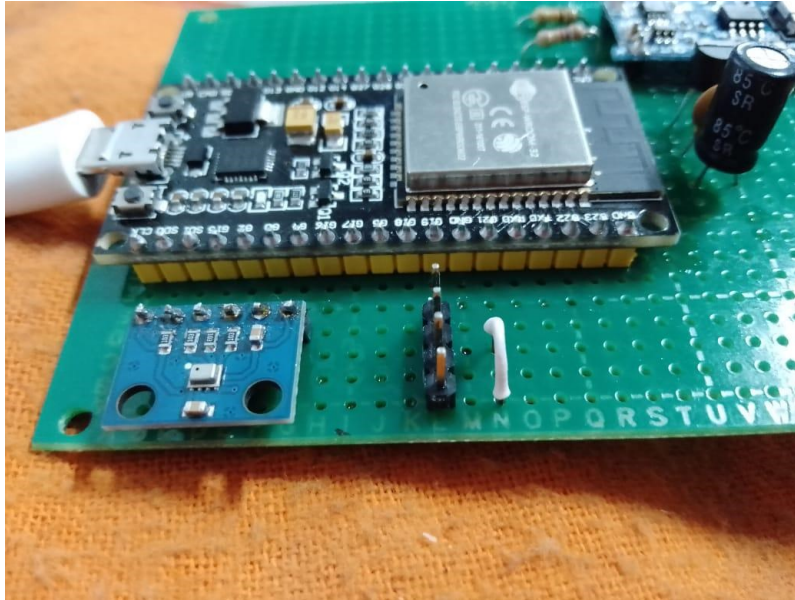


Figura N° 148 Imagen ampliada de placa con BMP280 y regleta de pines para el ADXL345

Continuaremos agregando a la placa el sensor DHT22 y el led indicador del estado de conexión wifi:

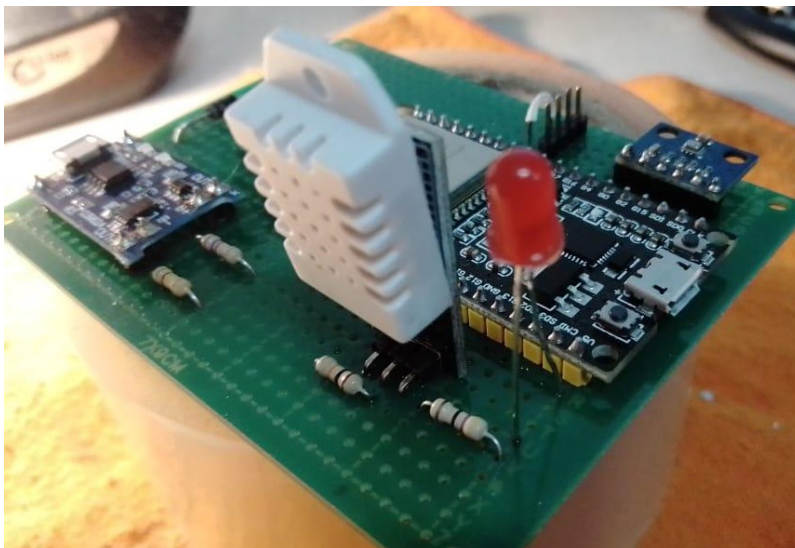


Figura N° 149 Sensor DHT22 y Led en placa controladora de sensores

Luego se añadieron cables para poder comunicar el display oled con la placa, de forma tal que se más cómoda su manipulación para ser empleada:

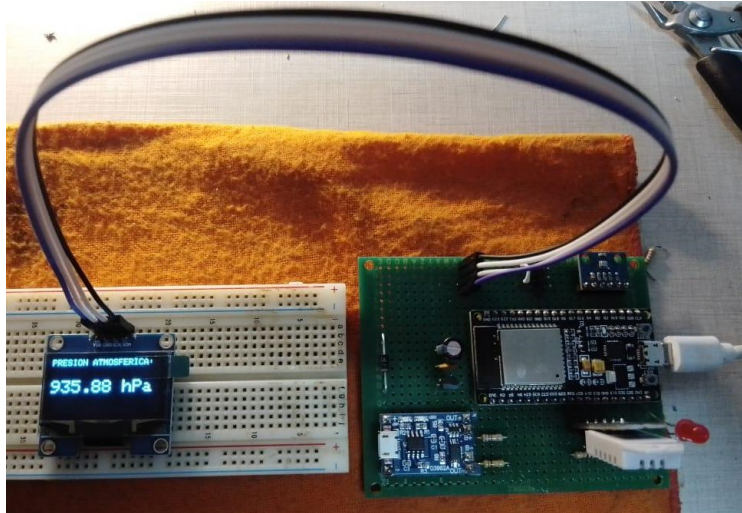


Figura N° 150 **Display oled comunicado con la placa y funcionando**

Por último, se procederá a añadir los cables que conectan el módulo acelerómetro con la placa, panel solar y soporte para baterías 18650:

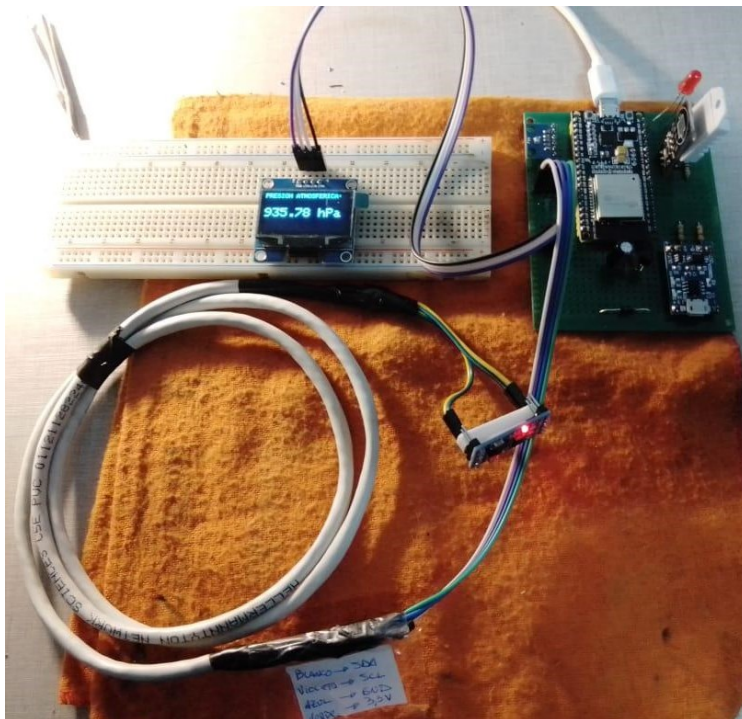


Figura N° 151 Conexión de ADXL345 a placa controladora

El módulo se introdujo en una esfera de polietileno expandido, de modo tal que el viento tenga un mayor contacto superficial y pueda variar la posición del módulo si hay presencia de este.



Figura N° 152 Sistema para detectar viento

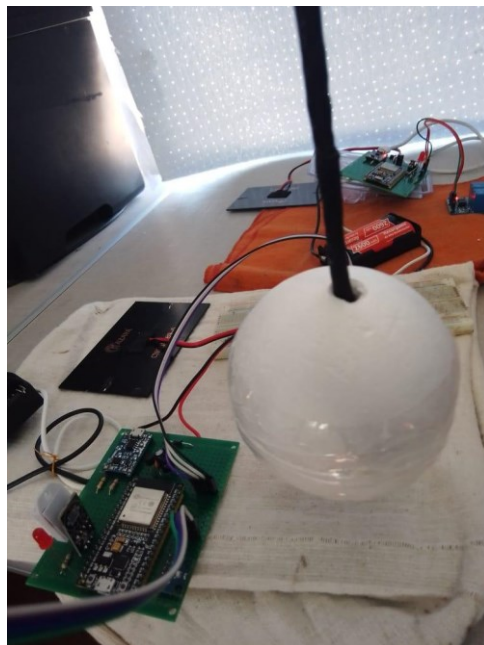


Figura N° 153 **Sistema para detectar viento completo**

A continuación, podemos ver una imagen del circuito de control de sensores completo, con todas sus conexiones necesarias para funcionar:

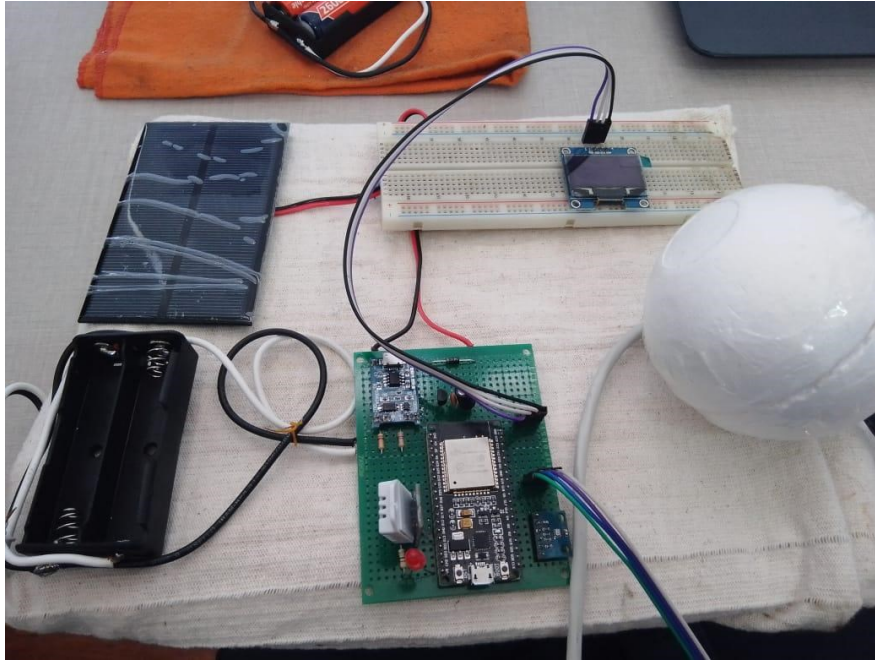


Figura N° 154 **Circuito de control de sensores completo en placa**

Se procederá a medir durante un lapso de tiempo, los valores de tensión que irán tomando las baterías en función al consumo del circuito en su estado normal. En la siguiente tabla se adjunta los valores de tensión medidos y los lapsos de tiempo de funcionamiento al cual corresponde:

Tiempo que lleva funcionando el circuito	Tensión Medida
00:00 Hs	4,10V
01:00 Hs	4,09V
02:00 Hs	4,08V
03:00 Hs	4,07V
04:00 Hs	4,06V
05:00 Hs	4,05V

Sistema de monitoreo para heladas

06:00 Hs	4,04V
07:00 Hs	4,03V
08:00 Hs	4,02V
09:00 Hs	4,01V
10:00 Hs	4,00V
11:00 Hs	3,99V
12:00 Hs	3,98V

Tabla N° 9 **Horas acumuladas de uso con la respectiva tensión medida en ese momento**

Tal como podemos observar, el consumo que genera este sistema es mínimo, debido a que funciona la mayor parte del tiempo en modo sueño profundo, pasando al modo activo solamente 11 segundos una vez por minuto. Esta prueba fue realizada con una sola batería cargada a casi el 100% (recordemos que 4,2V indica el 100%), estimando que es suficiente para mantener energizado el sistema por poco más de 5 días sin recibir ningún tipo de energía.

Sistema de monitoreo para heladas

9 Trabajo a futuro:

- 1- Mejorar la etapa de activación del sistema de riego. Si bien se concreta el control del sistema, no se toma ninguna información que permita confirmar el correcto funcionamiento del circuito a controlar. Podrían emplearse sensores de corriente, rpm, caudal, vibración, etc, cuya información nos permita comparar con valores de referencia, si el funcionamiento es correcto o no.

Todos estos valores ayudarán a tener un mejor control de nuestro sistema, pudiendo incluso establecer estados de alarma. Contar con esa información, podría permitir tener un mejor monitoreo en cuanto a la telemetría, ya que podría ser mostrada en pantalla.

- 2- Mejorar la activación o desactivación del sistema de riego. Esto es más que nada porque si yo cerrara la página, cuando la desee abrir, por defecto siempre iniciará en activar y esa no siempre será la condición real, ya que, si el sistema fue activado anteriormente, la página debería cargarse directamente con la opción de desactivar. Si bien se concreta la activación o no del sistema, no se almacena en ninguna variable el estado en el cual queda. Teniendo registro del estado, a través del algoritmo se podría mejorar estas condiciones particulares, brindando un sistema mucho más eficiente

- 3- Mejorar el sistema de control del riego, para que se mantenga activo si se pierde conexión wifi o se desenergiza el circuito de control. Esto apunta a que el sistema se activa a partir de energizar o no un relé. La NodeMCU que se encarga de ejecutar esta tarea, está programada para iniciarse ante una pérdida de conexión WIFI. Esto lo hace en búsqueda de volver a establecer comunicación, ya que es la única forma de controlar el sistema de riego a través de la telemetría. Si se reinicia por pérdida de conexión wifi, dejará de energizar el relé debido a que se vuelve a ejecutar el código desde el inicio, volviendo a su condición normal y teniendo que encenderlo nuevamente por medio de la página una vez que se normaliza la conexión wifi. Teniendo información del sensado mencionado, podría realizar un sistema que me permita no desactivar el sistema de riego por más que se me apague la NodeMCU. Además, podría reiniciarse la tarjeta y al ver determinados valores de rpm, corriente,

caudal, etc, podría darse cuenta cual es el estado del sistema, como si fuera una especie de memoria para saber en qué condiciones ubicar sus variables.

- 4- Mejorar el restablecimiento de conexión wifi entre la NodeMCU y la red. Al perderse conexión, el programa lo detecta y reinicia la tarjeta para poder ejecutar nuevamente el void setup y así proceder a intentar establecer nuevamente la conexión. Sería una gran mejora encontrar una alternativa que no implique el reinicio.
- 5- Mejorar el sensado del viento, empleando algún anemómetro que permita relevar información de la velocidad del viento.
- 6- Ampliar el uso a un servidor web que no sea solamente local, es decir buscar la forma de poder acceder a la telemetría, a través de internet y desde cualquier parte del mundo.
- 7- Mejorar el sistema de conexión, de forma tal que permita ampliar la distancia máxima que puede separarse la NodeMCU del router que entrega la red wifi.
- 8- Mejorar el circuito empleado para medir la carga de batería, ya que se emplean resistencias comerciales con tolerancia del 5%.

10 Conclusión:

Se lograron los resultados esperados, con un satisfactorio rendimiento en cuanto al diseño planteado. Aparenta ser un sistema eficiente, el cual siendo empleado con algunos sensores que puedan relevar información del estado de funcionamiento de una bomba de riego o de un motor a combustión, mejoraría completamente su aplicación a un caso real de sistema de riego por aspersión, brindando mayor tranquilidad a su usuario.

Si bien no permite tener un acceso a la telemetría a través de internet, el hecho de ser un sistema local asegura siempre tener datos presentes y en tiempo real. Esto nos lleva a lograr un sistema mucho más seguro, incluso brindando mayor tranquilidad al usuario que no puede prescindir de este tipo de información ante una futura presencia de una condición climática de helada. En un sistema de telemetría que use internet, si la conexión cae, dejaríamos de ver información actualizada en la telemetría, desviándose del objetivo principal de este sistema que es relevar datos de forma segura y cómoda para el usuario.

Efectivamente es un sistema que aportaría una gran comodidad al trabajador rural en este tipo de condiciones ambientales, siendo logrado gracias a la introducción de tecnología a un sector que ofrece resistencia a estos tipos de avances.

Además, es un sistema que puede ser empleado en otras operaciones rurales. La telemetría brinda información importantísima para el usuario en cuanto a las condiciones del cuadro frutal, la cual podría ser utilizada para el proceso de cura de la plantación que, sin entrar en mucho detalle, es llevada a cabo ante ciertas condiciones de temperatura y humedad.

11 Bibliografía

- [1] Heladas primaverales, protección en frutales de clima templado-frio – INTA
https://inta.gob.ar/sites/default/files/inta_las-heladas-primaverales.pdf

- [2] Heladas -Tipos, medidas de prevención y manejos posteriores al daño guía de uso del sitio <http://www.fdf.cl/biblioteca/publicaciones/2016/HELADAS.pdf>
- [3] Heladas: orígenes, tipos y métodos para combatirlas
<https://www.smn.gob.ar/noticias/heladas-or%C3%ADgenes-tipos-y-m%C3%A9todos-para-combatirlas>
- [4] Protección contra heladas
http://www.controlheladas.com/downloads/Proteccion_Contra_las_Heladas-Hojas_Divulgadoras.pdf
- [5] El origen de las heladas en Mendoza
<http://www.contingencias.mendoza.gov.ar/pdf/heladas.pdf>
- [6] Psicrómetro <https://www.ecured.cu/Psigr%C3%B3metro>
- [7] Termohigrógrafo <https://es.wikipedia.org/wiki/Termohigr%C3%B3grafo>
- [8] Temperaturas críticas y estados fenológicos
https://inta.gob.ar/sites/default/files/inta_estados-fenologicos-pepita-y-carozo.pdf
- [9] Datasheet del sensor DHT22
<https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>
- [10] Instrumentos de medición y control del viento
<https://instrumentosdemedicion.wordpress.com/>
- [11] Hoja de especificaciones del módulo acelerómetro MMA7361
<https://www.nxp.com/docs/en/data-sheet/MMA7361LC.pdf>
- [12] Datasheet nómada del módulo transceptor
<http://nomada-e.com/descargas/datasheet/60-Mo%CC%81dulo%20de%20RF%202.4GHz%20%5BnRF24L01%5D.pdf>
- [13] Datasheet del sensor DS18B20
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>
- [14] Tablas del sensor DS18B20
<https://programarfacil.com/blog/arduino-blog/ds18b20-sensor-temperatura-arduino>
- [15] Datasheet del sensor ADXL345
<https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf>

- [16] Datasheet del módulo ESP8266
https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
- [17] Información de la tarjeta NodeMCU
https://earchivo.uc3m.es/bitstream/handle/10016/29308/TFG_Teresa_Castro_Blanco.pdf?sequence=1
- [18] Pinout de tarjeta NodeMCU en grafico
<https://www.luisllamas.es/esp8266-nodemcu/>
- [19] Pinout de tarjeta NodeMCU
https://naylorlampmechatronics.com/blog/56_usando-esp8266-con-el-ide-de-arduino.html
- [20] Lenguaje php
https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=492:ique-es-php-y-ipara-que-sirve-un-potente-lenguaje-de-programacion-para-crear-paginas-web-cu00803b&catid=70&Itemid=193
- [21] Lenguaje html
<https://es.wikipedia.org/wiki/HTML>
- [22] Lenguaje html
<https://www.hostinger.com.ar/tutoriales/que-es-html>
- [23] Software XAMPP
<https://www.ionos.es/digitalguide/servidores/herramientas/instala-tu-servidor-local-xampp-en-unos-pocos-pasos/>
- [24] Lenguaje CCS
https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=546:que-es-y-para-que-sirve-el-lenguaje-css-cascading-style-sheets-hojas-de-estilo&catid=46&Itemid=163
- [25] Base de datos MySQL
<https://www.ticportal.es/glosario-tic/mysql>
- [26] información de phpMyAdmin

<https://blog.toadworld.com/2017/04/04/que-es-phpmyadmin-y-como-podemos-gestionar-la-base-de-datos-mysql-con-esta-herramienta>

- [27] Información de Heidi SQL
<https://es.wikipedia.org/wiki/HeidiSQL>
- [28] Información de NodeMCU ESP32
<https://programarfacil.com/esp8266/esp32/>
- [29] Información de NodeMCU ESP32
<http://esp32.net/>
- [30] Datasheet de NodeMCU ESP32
<https://www.alldatasheet.com/datasheet-pdf/pdf/1148023/ESPRESSIF/ESP32.html>
- [31] Datasheet de chip cargador de baterías TP4056
<https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf>
- [32] Datasheet de circuito de protección DW01-P
https://cdn.electronilab.co/wp-content/uploads/2016/05/DW01-P_DataSheet_V10.pdf
- [33] Datasheet de regulador de baja caída MCP1700
<https://ww1.microchip.com/downloads/en/DeviceDoc/MCP1700-Low-Quiescent-Current-LDO-20001826E.pdf>
- [34] Datasheet de relay SRD05VDCSLC
<https://html.alldatasheet.com/htmlpdf/1131944/SONGLERELAY/SRD05VDCSLC/850/1/SRD05VDCSLC.html>
- [35] Información sobre modulo rele de dos canales
<http://robots-argentina.com.ar/didactica/modulos-de-rele-y-arduino-domotica-1/>
- [35] Datasheet de transistor 2N2222
<http://web.mit.edu/6.101/www/reference/2N2222A.pdf>
- [36] Modulo DC-DC Step UP
<https://www.nubbeo.com.ar/productos/mini-modulo-step-up-eleva-a-5v-desde-0-9v-nubbeo/>
- [37] Datasheet de diodo 1N4001

<https://www.diodes.com/assets/Datasheets/ds28002.pdf>

- [38] Display Oled 128x64 I2C

<https://naylorlmechatronics.com/oled/638-display-oled-i2c-130-12864-sh1106.html>

Apéndice A

Códigos desarrollados

12 Apéndice A (códigos desarrollados):

En el presente anexo podemos encontrar todos los códigos originales empleados en el desarrollo de este proyecto.

12.1.1 Código index.php

Sistema de monitoreo para heladas

```
<!DOCTYPE html>
<html>
  <head>

    <link rel="shortcut icon" type="image/x-icon" href="Imagenes/Logo.jpg">
    <title>Datos Actuales</title>

    <!--a continuacion le digo que las modificaciones las tome del siguiente CCS en cuanto
al formato visual-->
    <link rel="stylesheet" href="css/estilos de tablas.css" type="text/css">

    <style type="text/css">
      td{
        padding: 30px
      }
    </style>
  </head>

  <!------->
  <!--a continuacion lo que hago es agregarle el menu superior a la pagina, es decir lo que
contiene Datos actuales y Datos Historicos, como para dar opcion a que retorne a donde quiera-
->
  <body>
    <header>
      <nav>
        <ul>

          <li><a href="http://localhost/proyectomodificado/index.php"target=>
<b>Datos Actuales </b></a> </li>
          <li><a
href="http://localhost/proyectomodificado/auxiliar.html"target="_blanck"> <b>Datos
Historicos</b></a> </li>
          <li ><a href="http://192.168.0.108/" target="_blanck"> <b>Sistema de
riego </b></a> </li>

          <!-- Al pulsar en datos historicos le indico que viaje al archivo
auxiliar.html, Este es un archivo que uso de intermediario para poder emplear
el metodo POST, ya que no encuentre forma de que funcione si no es desde
un html a un php.

          Lo que hago en ese html es seleccionar el rango de fechas desde donde
y hasta donde deseo ver los datos historicos. Despues con un boton envio esos datos a la
pagina historicos.php, donde los uso para buscar en la base de datos Mysql -->
```


Sistema de monitoreo para heladas

```
                <!-- <li><a
href="http://localhost/proyectomodificado/historicos.php"target=>                <b>Datos
Historicos</b></a> </li>
    Target=_blank me permite hacer que al clicar en TEMPERATURA HUMEDAD O VIENTO, se abra el
    link en una nueva pestaña-->

                </ul>
            </nav>
        </header>

<!-------
----->
<!-- a continuacion creo la tabla que contendra la informacion correspondiente a los sensores-
->

<?php

header("Refresh:30"); //este comando me permite actualizar la pagina cada 30 seg.
//este archivo contiene lo que seria necesario para poder abrir la base de datos creada en
myphp. Aqui se comprobara que el usuario y la clave sean correctos y de ser asi se realizara
la apertura

    $server = "localhost";//sera el valor de nuestra base de datos
    $bd= "proyectofinal";// el nombre que tiene la base de datos en myphp
    $usuario= "root"; //el usuario que se deberia ingresar
    $contraseña= ""; // la contraseña correcta
    $conexion = mysqli_connect($server,$usuario,$contraseña,$bd) //indicamos que la
variable conexion contenga todas las variables dentro del parentesis.De esta manera
establecemos la conexion
        or die ("error de conexion"); //si no conecta indicara error

//ahora procedo a realiza una consulta de la tabla sensores, ya que voy a proceder a mostrar
los datos actuales en pantalla

$consulta= mysqli_query($conexion, "SELECT * FROM sensores") //indicamos que seleccione la
tabla sensores de la base proyectofinal
or die ("error al traer datos");

    echo '<table border="5" width="90%" align="center">';//creamos a continuacion una
tabla para recibir los datos que consultamos a la base
    echo '<tr align="middle">';//creamos la primer fila que contiene la imagen de la utn

//agrego la imagen de la utn en una fila
```

Sistema de monitoreo para heladas

```
echo '<th colspan="6" >
    
    </th>';
echo '</tr>';

//creo la segunda fila de la tabla
echo '<tr bgcolor= #198DCF align="middle">';
echo '<th colspan="6"> Datos Actuales </th>';
echo '</tr>';

//creo la tercera fila que contiene fecha y hora, temperatura,humedad y presion
atmosferica
echo '<tr bgcolor= #198DCF>';
echo '<th font-size:12px; width=18% id "Fecha y Hora"> Fecha </th>';
echo '<th width=17% id "Temperatura"> Temperatura [°C] </th>';
echo '<th width=17% id "Humedad"> Humedad [%] </th>';
echo '<th width=17% id "PresionAtmosferica"> Presion Atmosferica [hPa] </th>';
echo '<th width=17% id "Viento"> Viento </th>';
echo '<th width=17% id "Viento"> Carga de bateria [%] </th>';
echo '</tr>';

// apartir de aca creo las filas necesarias para cargar los datos que contiene la tabla en
SQL
//a continuacion usamos el while para recorrer toda la tabla. La variable extraido contendra
todos los valores de la tabla. Es decir la variable consulta se convertira en un vector y se
igualara a extraido. el while lo recorrera hasta recorrer toda la tabla

while ( $extraido = mysqli_fetch_array($consulta) )
{
echo '<tr align="middle">'; //se crea una nueva fila cada vez que entra al while
echo '<td span style="color:white; font-size:30px;" width= 18%>'. $extraido['Fecha'].'</td>';
//entre corchetes se le pasa la columna de la base de datos
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['Temperatura'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['Humedad'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['PresionAtmosferica'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['Viento'].'</td>';
echo '<td span style="color:white; font-size:40px;" width= 17%>'.
$extraido['Bateria'].'</td>';
echo '</tr>';
```

Sistema de monitoreo para heladas

```
    }

    mysqli_close($conexion); //cierro la conexion con la base de datos
    echo '</table>';

?>

    </body>
</html>
```

12.1.2 Código auxiliar.php

```
<!DOCTYPE html>
<html>
<head>

    <link rel="shortcut icon" type="image/x-icon" href="Imagenes/Logo.jpg">
    <title>Datos Historicos</title>

    <!--a continuacion le digo que las modificaciones las tome del siguiente CCS en
cuanto al formato visual-->
    <link rel="stylesheet" href="css/estilos de tablas.css" type="text/css">

    <style type="text/css">
        td
        {
            padding: 5px
        }

        button,input
        {
            padding: 5px
        }

        label
        {
            padding: 20px;
            padding-right:2px;
            padding-left: 2px;
        }

    </style>
```

Sistema de monitoreo para heladas

```
</head>

<!-------
-----
-----
----->
<!-->

<body>
  <header>
    <nav>
      <ul>

        <form method="POST" action="historicos.php" > <!-- genero un form y
luego indico el metodo que usare para enviar los datos, el cual puede ser POST o GET, y le
indico a que pagina php voy a enviar esos datos. La envio a historicos.php porque ahi tengo
armado el codigo de busqueda en mysql-->

          <small><!-- esto es para establecer la fuente en tamaño small-
->

              <!-- A continuacion creo unos input del tipo datetime-
local que me permiten crear cada uno un selector de fecha y hora-->
              <label > Desde <input type="datetime-local"
name="fecha_min" value= ></label>
              <label > Hasta <input type="datetime-local"
name="fecha_max" value= ></label>

              <small>
                <button type="submit" widh:5px ><b>Buscar Datos<b></button>
<!-- el boton tiene que ser del tipo submit para que estos datos seleccionaos sean enviados
a historicos.php, si no fuera submit, los datos no se enviarian a ningun lado-->

            </form>

          </ul>
        </nav>
      </header>
    </body>

    <h2 align="center">Seleccione las fechas dentro de las cuales desea observar los datos
almacenados</h2>

</html>
```

12.1.3 Código historicos.php

```
<!DOCTYPE html>
<html>

<head>

    <link rel="shortcut icon" type="image/x-icon" href="Imagenes/Logo.jpg">
    <title>Datos Historicos</title>

    <!--a continuacion le digo que las modificaciones las tome del siguiente CCS en
    cuanto al formato visual-->
    <link rel="stylesheet" href="css/estilos de tablas.css" type="text/css">
    <style type="text/css">
        td
        {
            padding: 5px
        }

        input
        {
            padding: 5px
        }
    </style>

</head>
    <body>

    <!-------
    -----
    ----->

    <!--a continuacion lo que hago es agregarle el menu superior a la pagina, es decir lo que
    contiene TEMPERATURA INICIO VIENTO Y HUMEDAD, como para dar opcion a que retorne a donde
    quiera-->

    <header>
        <nav>
            <ul>

                <li><a href="http://localhost/proyectomodificado/index.php"target=>
<b>Datos Actuales </b></a> </li>
```

Sistema de monitoreo para heladas

```
        <li><a
href="http://localhost/proyectomodificado/auxiliar.html"target=> <b>Datos Historicos</b></a>
</li>
```

```
<!-- Target=_blanck me permite hacer que al cliquear en DATOS ACTUALES O DATOS HISTORICOS,
se abra el link en una nueva pestaña-->
```

```
        </ul>
    </nav>
</header>
</body>
</html>
```

```
<!-------
----->
```

```
<!-- a continuacion creo la tabla que contendra la informacion correspondiente a los DATOS
HISTORICOS
```

Aca simplemente lo que voy a hacer es conectarme a la base de datos, pasando los limites de fecha establecidos por los datos que estan viniendo desde auxiliar.html. Luego simplemente los muestro en pantalla a traves de una tabla

A diferencia del archivo index.php, aca no quiero que se actualice la pagina cada 60segundos, debido a que es necesario que los campos 'fecha_min' y 'fecha_max' siempre tengan un dato valido

```
-->
```

```
<?php
```

```
$fecha_min=$_POST['fecha_min']; //de esta forma recibo los datos que vienen desde
auxiliar.php a traves del metodo post y guardo esa info en una variable php
$fecha_max=$_POST['fecha_max']; //muy importante que lo que se esta recibiendo entre '' se
llame exactamente igual que en el html
//header("Refresh:60"); //este comando me permite actualizar la pagina cada 60 seg.
```

```
//este archivo contiene lo que seria necesario para poder abrir la base de datos creada en
myphp. Aqui se comprobara que el usuario y la clave sean correctos y de ser asi se realizara
la apertura
```

```
    $server = "localhost";//sera el valor de nuestra base de datos
    $bd= "proyectofinal";// el nombre que tiene la base de datos en myphp
    $usuario= "root"; //el usuario que se deberia ingresar
```

Sistema de monitoreo para heladas

```
$contraseña= ""; // la contraseña correcta
$conexion = mysqli_connect($server,$usuario,$contraseña,$bd) //indicamos que la
variable conexion contenga todas las variables dentro del parentesis.De esta manera
establecemos la conexion
    or die ("error de conexion"); //si no conecta indicara error

// ahora voy a realizar una consulta a la base de datos, para eso uso la conexion y le digo
que seleccione de la tabla historicos, de la columna fecha y me muestre los datos desde
fecha min hasta fecha max, que serian valores que yo estoy introduciendo desde el selector
de fechas que cree en el html

$consulta= mysqli_query($conexion, "Select * from historicos where Fecha BETWEEN
'$fecha_min' AND '$fecha_max' ")
//indicamos que seleccione la tabla historicos de la base sensor
or die ("error al traer datos");

    echo '<table border="5" width="70%" align="center">';//creamos a continuacion una
tabla para recibir los datos que consultamos a la base
    echo '<tr align="middle">'; //creamos la primer fila que contiene la imagen de la
utn

//agrego la imagen de la utn en una fila
echo '<th colspan="5" >
        
        </th>';
echo '</tr>';

//creo la segunda fila de la tabla
echo '<tr bgcolor= #198DCF align="middle">';
echo '<th colspan="6"> Datos Historicos </th>';
echo '</tr>';

//creo la segunda fila que contiene fecha hora y el valor medido
echo '<tr bgcolor= #198DCF>';
echo '<th width=18% id "Fecha y Hora"> Fecha </th>';
echo '<th width=17% id "Temperatura"> Temperatura [°C] </th>';
echo '<th width=17% id "Humedad"> Humedad [%] </th>';
echo '<th width=17% id "PresionAtmosferica"> Presion Atmosferica [hPa] </th>';
echo '<th width=17% id "Viento"> Viento </th>';
echo '<th width=17% id "Bateria"> Bateria </th>';
```

Sistema de monitoreo para heladas

```
echo '</tr>';

// apartir de aca creo las filas necesarias para cargar los datos que contiene la tabla en
SQL

//a continuacion usamos el while para recorrer toda la tabla. La variable extraido contendra
todos los valores de la tabla. Es decir la variable consulta se convertira en un vector y se
igualara a extraido. el while lo recorrera hasta recorrer toda la tabla

$aux=0; //creo una variable auxiliar cualquiera

while ( $extraido = mysqli_fetch_array($consulta) )
{
    //if ( $aux < 48) //lo que hago es que como mucho me muestre hasta 48 filas de
datos.
    //    {
        echo '<tr align="middle">'; //se crea una nueva fila cada vez que entra al
while
        echo '<td width= 18%>'. $extraido['Fecha'].'</td>'; //entre corchetes se le
pasa la columna de la base de datos
        echo '<td width= 17%>'. $extraido['Temperatura'].'</td>';
        echo '<td width= 17%>'. $extraido['Humedad'].'</td>';
        echo '<td width= 17%>'. $extraido['PresionAtmosferica'].'</td>';
        echo '<td width= 17%>'. $extraido['Viento'].'</td>';
        echo '<td width= 17%>'. $extraido['Bateria'].'</td>';
        echo '</tr>';
        $aux++;
    //    }
}

$aux= 0;
mysqli_close($conexion); //cierro la conexion con la base de datos

echo '</table>';

?>

</body>
</html>

12.1.4 Código recibe_datos.php

<?php
```


Sistema de monitoreo para heladas

/*Este archivo se va a encargar de recibir los datos que estan viniendo desde la tarjeta ESP8266.

Se va a tomar el trabajo de primero conectarse a la Base de datos ubicada en el servidor local creado con XAMPP.

Una vez conectado a la base de datos, procede a guardar en las variables \$temperatura, \$humedad, \$presion y \$bateria los datos que estan viniendo desde la tarjeta.

Despues una vez que tengo todos los datos que deseo, lo primero que hago es cargarlos en la tabla HISTORICOS de la base de datos proyectofinal.

Despues de esto voy y actualizo la tabla de datos actuales, que son los que uno se supone que esta mirando en tiempo real

La idea de crear este archivo recibe_datos.php es que sera un programa que se ejecutara en segundo plano, sin interferir con las paginas principales con las cuales va a interactuar el usuario. Es decir, es una pagina que no deseamos ver en una pestaña, sino que solo queremos que se ejecute sin que nos moleste

*/

//este archivo contiene lo que seria necesario para poder abrir la base de datos creada en myphp. Aqui se comprobara que el usuario y la clave sean correctos y de ser asi se realizara la apertura

```
$server = "localhost";//sera el valor de nuestra base de datos
$dbd= "proyectofinal";// el nombre que tiene la base de datos en myphp
$usuario= "root"; //el usuario que se deberia ingresar
$contraseña= ""; // la contraseña correcta
$conexion = mysqli_connect($server,$usuario,$contraseña,$dbd) //indicamos que la
variable conexion contenga todas las variables dentro del parentesis.De esta manera
establecemos la conexion
or die ("error de conexion"); //si no conecta indicara error
```

// a partir de aca vamos a tomar los datos de los sensores y a guardarlos en la base de datos

// en las siguientes 5 variables yo voy a recibir los datos que vengan de los sensores , y luego las usare para cargar los datos en la base de datos

/*la pagina va a recibir estas 5 variables desde la tarjeta ESP32 y se tienen que llamar exactamente como son enviadas*/

```
$temperatura = $_POST['temperatura'];
```

Sistema de monitoreo para heladas

```
$humedad = $_POST['humedad'];
$presion = $_POST['presion'];
$viento = $_POST['viento'];
$bateria = $_POST['bateria'];
date_default_timezone_set('america/argentina/buenos_aires'); //establezco la zona
de la cual deseo tomar los datos de fecha y hora
$fecha_actual = date("Y-m-d H:i"); //indico el formato de fecha y hora que quiero,
donde vemos que no quiero los segundos

/*estos 4 siguientes echo los coloco para usarlos en el puerto serial del arduino
Con estos yo puedo mostrar en el puerto serial cuales son los datos que se van a
guardar en la base de datos
Estos deberian coincidir con los datos tomados inmediatamente en el codigo de
arduino. Si es asi, significa que al servidor llegaron correctamente
los datos que deseabamos. Es una forma adicional de confirmar que correctamente me
comunique con la pagina, como tambien lo es el codigo 200 de http
*/

echo "Datos recibidos: ";
echo " ";
echo "Temperatura= ".$temperatura." Humedad= ".$humedad." Presion=
".$presion." Viento= ".$viento." Bateria= ".$bateria;

//con las dos sentencias siguientes lo que hago es hacer uso del comando INSERT
INTO y selecciono en que tabla deseo insertar nuevos datos que van a venir del sensor. La
idea es justamente guardarlos en los HISTORICOS. Lo que hago es primero seleccionar las
columnas que voy a modificar y despues le voy a pasar los valores que voy a guardar y donde
// despues con la opcion $insert = mysqli_query($conexion,$query);

$query = "INSERT INTO historicos(idDevice, Temperatura, Humedad, PresionAtmosferica,
Fecha, Viento, Bateria)
VALUES('tarjeta1', '$temperatura', '$humedad', '$presion', '$fecha_actual', '$viento', '$bateria')
";
$insert = mysqli_query($conexion,$query);

//con la siguiente sentencia lo que hago es actualizar la tabla sensores, la cual
resulta ser lo que se va a mostrar como datos actuales y que es lo que yo voy a estar
monitoreando como valores en tiempo real

$query = "UPDATE sensores SET Temperatura = '$temperatura', Humedad = '$humedad',
PresionAtmosferica= '$presion', Fecha = '$fecha_actual' ,Viento = '$viento', Bateria =
'$bateria' WHERE idDevice ='tarjeta1'";
$insert = mysqli_query($conexion,$query);
```

Sistema de monitoreo para heladas

```
mysqli_close($conexion); //cierro la conexion con la base de datos
```

```
?>
```

12.1.5 Código de prueba para conexión wifi con NodeMCU

```
#include <Ticker.h>
#include <WiFi.h>

#define ledWifi 26
String ssid = "TP-LINK_A370"; //coloco el nombre de la red a la que quiero conectarme
String password = "70932911"; //coloco la contraseña de la red

Ticker tic_WifiLed;

byte cont = 0; //ire contando los intentos con este contador
byte max_intentos = 20; //establezco un maximo de intentos de conectarse a la red
                        //las declaro de tipo byte porque es como un INT solo que
                        //en vez de ocupar 4 byte va a ocupar 1 byte

//-----

void parpadeoLedWifi()
{
  byte estado = digitalRead(ledWifi);
  digitalWrite(ledWifi,!estado);
}

//-----

void setup()
{
  //Inicio la conexion del puerto serial para poder imprimir que hace el modulo

  pinMode(ledWifi,OUTPUT); // estamos configurando el pin digital D4 como SALIDA
  Serial.begin(115200); //importante que coincida con el monitor serial de arduino
  Serial.println("\n"); // Es un salto de linea "ENTER"

  tic_WifiLed.attach(0.2,parpadeoLedWifi); //desde aca va a empezar a parpadear el led
  mientras este intentando conectarse a la red

  //Conexion wifi, esta es la parte importante donde me conecta o no con la red wifi
```

Sistema de monitoreo para heladas

```
WiFi.begin(ssid, password); //abrimos la conexion wifi pasandole el usuario y contraseña
de la red

while(WiFi.status() != WL_CONNECTED and cont < max_intentos) //mientras no se conecte ira
aumentando un contador

                                                                    //hasta 20 intentos
{
  cont++;
  delay(500);
  Serial.print(".");
}

Serial.println(""); //salto de linea

if(cont < max_intentos) //si entra quiere decir que se conecto
{
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
  Serial.println("*****");
  Serial.println("Error de conexion");
  Serial.println("*****");
}

tic_WifiLed.detach();//una vez que termina de intentar conectarse, terminamos el proceso
del led
digitalWrite(ledWifi,HIGH); // apagamos el led por si llego a quedar prendido
                                                                    //recordar que los led de la tarjeta estan NEGADOS por lo que
se prenden con un CERO y se apagan con UNO
}

void loop()
{
}
```

12.1.6 Código de prueba para sensor bmp280

Sistema de monitoreo para heladas

```
#include <Wire.h>// se incorpora para poder trabajar con I2C
#include <Adafruit_Sensor.h>//estas dos librerias son para emplear el sensor
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bmp;//creamos un objeto de esa clase

float temperatura;//variables donde guardaremos los datos
float presion;

void setup()
{
  Serial.begin(115200);
  Serial.println("Iniciando:");
  if(!bmp.begin(0x76))
  {
    //vamos a comprobar si podemos comunicarnos o no con el sensor
    Serial.println("sensor de presion BMP280 no encontrado");
    while(1);//permanecera aca
  }
}

void loop()
{
  temperatura = bmp.readTemperature();
  presion = bmp.readPressure()/100;// asi como esta obtenemos el dato en hPa

  Serial.println(" Temperatura: " + String(temperatura) + "°C ");
  Serial.println(" Presion: " + String(presion) + "Pa ");
  Serial.println("*****");
  delay(5000);
}
```

12.1.7 Código de prueba para sensor dht22

```
#include <DHT.h>
#include <DHT_U.h>
DHT dht(D7,DHT22); //indico que los datos del sensor se conectan al D1 de la placa
                // y que usare el sensor DHT11

float temperatura,humedad; //declaro dos variables globales

void setup()
{
  Serial.begin(115200);
  dht.begin();
}
```

Sistema de monitoreo para heladas

```
    }

void loop()
{
    temperatura = dht.readTemperature();
    humedad = dht.readHumidity();
    Serial.println("Temperatura: " + String(temperatura) + "Humedad: " + String(humedad));
    delay(5000);
}
```

12.1.8 Código de prueba para sensor ds18B20

```
#include <Wire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 0 //conecto el pun de datos del sensor al pin digital 0 de la placa

OneWire ourWire(ONE_WIRE_BUS); // configuro una instancia oneWire para comunicar el sensor
DallasTemperature ds18B20(&ourWire); //creo un objeto de la clase DallasTemperature para
poder llamar a las funciones y le paso la referencia oneWire

void setup()
{
    Serial.begin(115200); //inicializo el puerto serial
    ds18B20.begin(); //inicializo la libreria del sensor
}

void loop()
{
    ds18B20.requestTemperatures(); //se envia el comando que permitira que el sensor devuelva la
temperatura
    Serial.println("Temperatura sensor ds18B20: " + String(ds18B20.getTempCByIndex (0)) + " °C
"); // imprimo la temperatura en grados centigrados
    Serial.println("*****");
    delay(5000);
}
```

12.1.9 Código de prueba de ADXL345

```
// conectar alimentacion, GND y los pines SDA y SCL
// son solamente 4 pines los que se conectan

//*****
//*****
```

Sistema de monitoreo para heladas

```
/*La hoja de datos de ADXL345 dice en la página 10 que nos dice que el ADXL puede tener
dos direcciones, dependiendo de cómo lo conecte. Estas direcciones son 0x53 y 0x1D.
La alteración de direcciones se realiza cambiando el estado ALTO / BAJO del pin ADXL SDO.
En mi Publicación anterior de ADXL Lo conecté para que responda a la dirección 0x53,
conectando a tierra el pin SDO (configurándolo en LOW)*/

//en la NodeMCU el pin d1 es SCL y el pin d2 es SDA
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

boolean viento=false; // con esta variable bandera voy a ver si hay o no viento

Adafruit_ADXL345_Unified ADXL345 = Adafruit_ADXL345_Unified();
int x=0,y=0 ,z=0; // guardare los valores de cada eje para ver si hay presencia de viento o
no, tomando un valor estatico de x=0 y=0 y Z=-10 como referencia de que no hay viento
void setup()
{
  Serial.begin(115200);

  if(!ADXL345.begin())
  {
    Serial.println("El sensor no incio");
    while(1);
  }
}

void loop()
{
  sensors_event_t evento;
  ADXL345.getEvent(&evento);
  x = evento.acceleration.x;
  y = evento.acceleration.y;
  z = evento.acceleration.z;
  Serial.print("Eje x = " + String(evento.acceleration.x));
  Serial.print("\t\t\tEje y = " + String(evento.acceleration.y));
  Serial.println("\t\t\tEje z = " + String(evento.acceleration.z));
  delay(1000);
  //comparamos valores para ver si hay o no presencia de viento
  // la referencia estatica es x=0 y=0 z= -10
  if( -1 < x && x < 1)
  {
    if( -1 < y && y < 1)
    {
```

Sistema de monitoreo para heladas

```
        viento = false;
    }
}

if( x > 5 || x < -5)
{
    viento= true;
}

if( y > 5 || y < -5)
{
    viento= true;
}

if(z > -9 && z < -4)
{
    viento= true;
}

if(viento == true)Serial.println("Hay viento");
if(viento == false)Serial.println("No Hay viento");
}
```

12.1.10Código de prueba para emplear Display OLED 128x64 I2C

```
#include <Wire.h>
#include <Adafruit_GFX.h> // estas dos librerias son empleadas para manejar el modulo oled
#include <Adafruit_SH110X.h> // esta especialmente solo funciona con el modulo 128x64 o
128x32 o 128x128 pero que figura 1,3" ya que usan el microcontrolador SH1106

#define Direccion_I2C 0x3c //defino la direccion I2C del modulo que para este caso es 0x3c
pero puede variar
#define Ancho_display 128 // definimos el ancho en pixeles del display
#define Altura_display 64 // definimos el altura en pixeles del display
#define OLED_RESET -1 // algunos modelos de pantallas incluyen el pin de reset pero en
este caso no lo trae, por lo tanto indicaremos esto con un -1

//Ahora creamos una instancia de la libreria y la llamamos display y le pasamos el ancho de
la pantalla, alto, un puntero a la comunicacion I2C y por ultimo el parametro de pin de
reset
Adafruit_SH1106G display = Adafruit_SH1106G(Ancho_display, Altura_display, &Wire,
OLED_RESET);

void setup()
{
    Serial.begin(115200);
```


Sistema de monitoreo para heladas

```
    if(!display.begin(Direccion_I2C, true)) //inicializamos la pantalla pasandole la direccion
del modulo dentro de un if. Si no se cumple la condicion de TRUE, imprimiremos que hay un
error de conexion a traves del puerto serial
    {
        Serial.println("Error de conexion con pantalla OLED");
    }
}

void loop()
{

    display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
    display.setTextSize(1);//ajusto el tamaño de la fuente
    display.setTextColor(SH110X_WHITE);// dependiendo del modelo de pantalla puedo ajustar el
color
    display.setCursor(0, 10); // ubico el texto tomando como referencia de cero la esquina
superior izquierda de la pantalla. Primero se pasa la posicion en X y luego en Y

    display.println("Probando codigo"); // escribo el texto que deseo mostrar
    display.println("Peroni Bruno"); // escribo el texto que deseo mostrar
    display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
}
```

12.1.11 Código de prueba para NodeMCU ESP32 empleando sus dos núcleos

```
TaskHandle_t tareal; //para poder usar mas de 1 nucleo voy a necesitar definir tareas
int cont=0; // declaramos una variable global que sera aumentada desde los dos nucleos, para
mostrar que pueden compartir informacion

//-----
//-----

void setup()
{
    //necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a
tomar
    xTaskCreatePinnedToCore(loop2,"tarea_1",10000,NULL,1,&tareal,0);// vamos a crear una tarea
que apunte a un nucleo en particular, en este caso el nucleo N°0
    /* (le paso la funcion que quiero que se ejecute ,
    le paso un nombre a esta funcion ,
    le paso el tamaño de la pila ,
    indico algun valor que desee pasarle a la tarea que pare este caso sera NULL,
```

Sistema de monitoreo para heladas

```
        indico la prioridad ,
        indico el nombre que le dimos a la tarea ,
        indico el nucleo donde quiero que se ejecute la tarea)
    */
Serial.begin(115200);

}

//-----
/*el void loop() tradicional por defecto se ejecutara infinitamente debido a que arduino asi
lo definio pero este void loop2() no lo va a hacer por si solo, sino que yo debere hacer que
suceda*/

// este void loop2() deberia correr en el nucleo N°0

void loop2(void *parameter)
{
    for(;;) //de esta manera logro que se ejecute infinitamente
    {
        Serial.println("*****");
        Serial.println("\t\t\tEl nucleo que esta corriendo es el N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
        delay(1000);
        cont++;
        Serial.println("La cuenta es: " + String(cont));
        Serial.println("*****");
    }
    vTaskDelay(10);
}

//-----
//-----

void loop()
{
    Serial.println("*****");
    Serial.println("El nucleo que esta corriendo es el N° " + String(xPortGetCoreID())); // con
la funcion xPortGetCoreID() yo voy a obtener en que nucleo se estara ejecutando este void
loop()
    delay(5000);
    cont++;
    Serial.println("La cuenta es: " + String(cont));
    Serial.println("*****");
}
}
```

12.1.12 Código de prueba para envío de datos mediante método POST desde NodeMCU

```
//*****
//*****
/*
recordar desactivar el FIREWALL de windows y de usar una ip estatica 192.168.0.123 para la
notebook que tenga el servidor local ya que sino no funcionara

*/
//*****
//*****

#include <HTTPClient.h>
#include <WiFi.h>

//creo dos variables constantes donde guardare el nombre de la red y la contraseña

const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red

int cont=0;
float temperatura_bmp280 = 0; // la temperatura que detecta el bmp290
float presion=0; // la presion que detecta el bmp290
float temperatura=0,humedad=0; //datos provenientes desde el DHT22
String viento="Hay viento";
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";/*
/hola/historicos.php aca asignamos la url a la cual queremos enviarle los datos. Es
importante ver que tengo que reemplazar LOCALHOST por la ip de mi maquina ya que solamente
asi podra comunicarse la tarjeta ESP8622. si fuera un servidor de internet, o haria falta.*/

void setup()
{
  Serial.begin(115200); //definimos la velocidad del puerto serial
  WiFi.begin(ssid, password); //Iniciamos la conexion con la red, recibiendo como parametros
el nombre y la contraseña de la red
  Serial.print("Conectando..."); //imprimimos que la conexion se esta realizando

  while(WiFi.status() != WL_CONNECTED and cont < 50)
  {
    cont++;
    delay(500);
    Serial.print(".");
  }
}
```

Sistema de monitoreo para heladas

```
}

Serial.println(""); //salto de linea

if(cont < 50) //si entra quiere decir que se conecto
{
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
  Serial.println("*****");
  Serial.println("Error de conexion");
  Serial.println("*****");
}
cont=0;
}

void loop()
{
  EnvioDatos(); //implementamos el envio de los datos al servidor
  delay(60000); //espera 60s
}

// rutina de envio de datos por POST

void EnvioDatos()
{
  //las sentencias dentro del siguiente if se van a ejecutar solo si la placa se encuentra
  conectada a la red wifi
  if (WiFi.status() == WL_CONNECTED)
  {
    HTTPClient http;// creamos un objeto de la clase HTTPClient que se encuentra en
    ESP8266HTTPClient.h
    http.begin(url);//con el metodo begin inicializamos el objeto. Le pasamos como parametro una
    URL
    http.addHeader("Content-Type","application/x-www-form-urlencoded");
```

Sistema de monitoreo para heladas

```
String datos_a_enviar ="temperatura= " + String(temperatura) + "&humedad= " +
String(humedad) + "&presion= " + String(presion)+ "&viento= " + String(viento) ;

int codigo_respuesta = http.POST(datos_a_enviar);//vamos a almacenar la respuesta del
servidor en una variable entera
//estos codigos numericos nos van a indicar si las solicitudes al servidor se han completado

    if (codigo_respuesta>0) //si el codigo de respuesta es mayor a cero lo imprimo por el
puerto serial
    {
        Serial.println("Código HTTP: "+ String(codigo_respuesta));
        if (codigo_respuesta == 200) // si da 200 es una respuesta satisfactoria
        {
            String cuerpo_respuesta = http.getString();
            Serial.println("*****");
            Serial.println("El servidor respondió: ");
            Serial.println(cuerpo_respuesta);
            Serial.println("*****");
            Serial.println("");
            Serial.println("");
        }
    }
else // caso contrario imprimo el codigo que entrego eñ servidor
{
    Serial.print("Error enviado POST, código: ");
    Serial.println(codigo_respuesta);
}
http.end(); // finalizo la peticion
}
}
```

12.1.13 Desarrollo de código para enviar datos mediante método POST, empleando un núcleo para el trabajo con los sensores y el otro núcleo para enviar los datos

```
/*
*****
*****
*/
andando para la ESP32, recordar desactivar el FIREWALL de windows y de usar una ip estatica
192.168.0.123 para la notebook que tenga el servidor local ya que sino no funcionara
*/
/*
*****
*****
*/
```

Sistema de monitoreo para heladas

```
//librerias para la conexion WIFI - Cliente ESP32

#include <HTTPClient.h>
#include <WiFi.h>

//creo dos variables constantes donde guardare el nombre de la red y la contraseña

const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red

int cont=0; //contador usado para limitar a 50 intentos de conexion wifi
float temperatura_bmp280 = 0; // la temperatura que detecta el bmp290
float presion=0; // la presion que detecta el bmp290
float temperatura=0,humedad=0; //datos provenientes desde el DHT22
String viento="Hay viento";
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";/*
/hola/historicos.php aca asignamos la url a la cual queremos enviarle los datos. Es
importante ver que tengo que reemplazar LOCALHOST por la ip de mi maquina ya que solamente
asi podra comunicarse la tarjeta ESP32. si fuera un servidor de internet, o haria falta. */

TaskHandle_t tarea1; //para poder usar mas de 1 nucleo voy a necesitar definir tareas

//*****
//*****
//*****

void setup()

{

//necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a
tomar
xTaskCreatePinnedToCore(EnvioDatos,"tarea_1",10000,NULL,1,&tarea1,0); // vamos a crear una
tarea que apunte a un nucleo en particular, en este caso el nucleo N°0
/* (le paso la funcion que quiero que se ejecute ,
le paso un nombre a esta funcion ,
le paso el tamaño de la pila ,
indico algun valor que desee pasarle a la tarea que pare este caso sera NULL,
indico la prioridad ,
indico el nombre que le dimos a la tarea ,
indico el nucleo donde quiero que se ejecute la tarea)
*/
*/
```

Sistema de monitoreo para heladas

```
Serial.begin(115200); //definimos la velocidad del puerto serial
WiFi.begin(ssid, password); //Iniciamos la conexion con la red, recibiendo como parametros
el nombre y la contraseña de la red
Serial.print("Conectando..."); //imprimimos que la conexion se esta realizando

while(WiFi.status() != WL_CONNECTED and cont < 50) //mientras no se conecte ira aumentando
un contador
{
  cont++;
  delay(500);
  Serial.print(".");
}

  Serial.println(""); //salto de linea

if(cont < 50) //si entra quiere decir que se conecto
{
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdres: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
  Serial.println("*****");
  Serial.println("Error de conexion");
  Serial.println("*****");
}
cont=0;
}

//*****
//*****
//*****

void loop()
{
  Serial.println("*****");
  Serial.println("");
```

Sistema de monitoreo para heladas

```
Serial.println("El sensado de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
    delay(10000); //espera 10seg
}

//*****
//*****
//*****
// rutina de envio de datos por POST

void EnvioDatos(void *parameter)
{

for( ; ; )
    {
        delay(5000); //espera 5seg
        Serial.println("*****");
        Serial.println("");
        Serial.println("El envio de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
        vTaskDelay(10);

        //las sentencias dentro del siguiente if se van a ejecutar solo si la placa se encuentra
conectada a la red wifi
        if (WiFi.status() == WL_CONNECTED)
            {
                HTTPClient http;// creamos un objeto de la clase HTTPClient que se encuentra en
ESP8266HTTPClient.h
                http.begin(url);//con el metodo begin inicializamos el objeto. Le pasamos como
parametro una URL
                http.addHeader("Content-Type","application/x-www-form-urlencoded");
                String datos_a_enviar ="temperatura= " + String(temperatura) + "&humedad= " +
String(humedad) + "&presion= " + String(presion)+ "&viento= " + String(viento) ;

                int codigo_respuesta = http.POST(datos_a_enviar);//vamos a almacenar la respuesta del
servidor en una variable entera
                //estos codigos numericos nos van a indicar si las solicitudes al servidor se han completado

                if (codigo_respuesta>0) //si el codigo de respuesta es mayor a cero lo imprimo por el
puerto serial
                    {
                        Serial.println("Código HTTP: "+ String(codigo_respuesta));
                        if (codigo_respuesta == 200) // si da 200 es una respuesta satisfactoria
```


Sistema de monitoreo para heladas

```
Serial.begin(115200);
delay(100);
}

//-----
//-----
//-----

void loop()

{
  cont++; //incremento cada vez que se despierta
  Serial.println("*****");
  Serial.println("Despierta ESP32, el numero de veces que durmio es: " +
String(cont));
  Serial.println("");
  esp_sleep_enable_timer_wakeup(tiempo); //configuramos el tiempo que deseamos que
duerma
  Serial.println("Configuramos para que duerma " + String(segundos) +
" Segundos");
  Serial.println("*****");
  esp_deep_sleep_start(); //con esta sentencia iniciamos el modo deep sleep
  Serial.println("Esto nunca deberia imprimirse porque como mande a dormir el micro,
cuando despierta es como si se reseteara la placa, volviendo al principio del setup ");
}
}
```

12.1.15 Desarrollo de código para enviar datos mediante método POST, empleando un núcleo para controlar los sensores y el otro núcleo para enviar los datos y activar el modo sueño profundo

```
/**
Se procede a incluir en este codigo el envio de datos por metodo POST hacia un archivo php
el cual se comunicara con la base de datos MySQL. Lo haremos empleando los dos nucleos
principales y ademas activando el modo sueño profundo cada 1 minuto
*/
//librerias para la conexion WIFI - Cliente ESP32

#include <HTTPClient.h>
#include <WiFi.h>

// variables empleadas para la conexion wifi
```

Sistema de monitoreo para heladas

```
const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
////creo dos variables constantes donde guardare el nombre de la red y la contraseña
const char* password = "brunela21"; //coloco la contraseña de la red
int cont=0; //contador usado para limitar a 50 intentos de conexion wifi

//*****
// variables empleadas para almacenar la informacion de los sensores

float temperatura_bmp280 = 0;// la temperatura que detecta el bmp290
float presion=0; // la presion que detecta el bmp290
float temperatura=0,humedad=0; //datos provenientes desde el DHT22
String viento="Hay viento";
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";/*
/hola/historicos.php aca asignamos la url a la cual queremos enviarle los datos. Es
importante ver que tengo que reemplazar LOCALHOST por la ip de mi maquina ya que solamente
asi podra comunicarse la tarjeta ESP32. si fuera un servidor de internet, o haria falta. */

//*****
// tarea creada para poder emplear el otro nucleo del ESP32

TaskHandle_t tarea1; //para poder usar mas de 1 nucleo voy a necesitar definir tareas

//*****
// variables empleadas para el sueño profundo

#define factor_conversion 1000000 /* creamos un factor de conversion para poder pasar de
microsegundos a segundos, debido a que la funcion que me permite configurar el tiempo que
deseo mandar a dormir el ESP32 toma los valores en microsegundos*/
#define segundos 55 /* asignamos el numero de segundos que deseamos mandar a dormir la
placa*/

int tiempo = segundos * factor_conversion ; //los segundos pasados a microsegundos
RTC_DATA_ATTR int contador= 0; //de esta forma declaramos una variable que se almacenara en
la memoria flash del micro procesador RTC el cual no se apaga cuando mandamos a dormir

//*****
//*****
//*****

void setup()
{

//necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a
tomar
```

Sistema de monitoreo para heladas

```
xTaskCreatePinnedToCore(EnvioDatos,"tarea_1",10000,NULL,1,&tareal,0); // vamos a crear una
tarea que apunte a un nucleo en particular, en este caso el nucleo N°0
    /* (le paso la funcion que quiero que se ejecute ,
    le paso un nombre a esta funcion ,
    le paso el tamaño de la pila ,
    indico algun valor que desee pasarle a la tarea que pare este caso sera NULL,
    indico la prioridad ,
    indico el nombre que le dimos a la tarea ,
    indico el nucleo donde quiero que se ejecute la tarea)
    */

    Serial.begin(115200); //definimos la velocidad del puerto serial
    WiFi.begin(ssid, password); //Iniciamos la conexion con la red, recibiendo como parametros
el nombre y la contraseña de la red
    Serial.print("Conectando..."); //imprimimos que la conexion se esta realizando

    while(WiFi.status() != WL_CONNECTED and cont < 50) //mientras no se conecte ira aumentando
un contador
    {
        cont++;
        delay(500);
        Serial.print(".");
    }

    Serial.println(""); //salto de linea

    if(cont < 50) //si entra quiere decir que se conecto
    {
        Serial.println("*****");
        Serial.print("Conectado a la red WiFi: ");
        Serial.println(WiFi.SSID());
        Serial.print("IP: ");
        Serial.println(WiFi.localIP());
        Serial.print("macAdress: ");
        Serial.println(WiFi.macAddress());
        Serial.println("*****");
    }
    else //si entra quiere decir que no se conecto
    {
        Serial.println("*****");
        Serial.println("Error de conexion");
        Serial.println("*****");
    }
    cont=0;
}
```

Sistema de monitoreo para heladas

```
//*****  
//*****  
//*****  
  
void loop()  
{  
    Serial.println("*****");  
    Serial.println("");  
    Serial.println("El sensado de datos esta corriendo en el nucleo N° " +  
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo  
se estara ejecutando este void loop()  
    delay(1000); //espera 10seg  
}  
  
//*****  
//*****  
//*****  
  
// rutina de envio de datos por POST  
  
void EnvioDatos(void *parameter)  
{  
  
for( ; ; )  
    {  
        delay(3000); //espera 5seg  
        Serial.println("*****");  
        Serial.println("");  
        Serial.println("El envio de datos esta corriendo en el nucleo N° " +  
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo  
se estara ejecutando este void loop()  
        vTaskDelay(10);  
  
        //las sentencias dentro del siguiente if se van a ejecutar solo si la placa se encuentra  
conectada a la red wifi  
        if (WiFi.status() == WL_CONNECTED)  
            {  
                HTTPClient http;// creamos un objeto de la clase HTTPClient que se encuentra en  
ESP8266HTTPClient.h  
                http.begin(url);//con el metodo begin inicializamos el objeto. Le pasamos como  
parametro una URL  
                http.addHeader("Content-Type","application/x-www-form-urlencoded");  
                String datos_a_enviar ="temperatura= " + String(temperatura) + "&humedad= " +  
String(humedad) + "&presion= " + String(presion)+ "&viento= " + String(viento) ;
```

Sistema de monitoreo para heladas

```
int codigo_respuesta = http.POST(datos_a_enviar);//vamos a almacenar la respuesta del
servidor en una variable entera
//estos codigos numericos nos van a indicar si las solicitudes al servidor se han completado

if (codigo_respuesta>0) //si el codigo de respuesta es mayor a cero lo imprimo por el
puerto serial
{
  Serial.println("Código HTTP: "+ String(codigo_respuesta));
  if (codigo_respuesta == 200) // si da 200 es una respuesta satisfactoria
  {
    String cuerpo_respuesta = http.getString();
    Serial.println("*****");
    Serial.println("El servidor respondió: ");
    Serial.println(cuerpo_respuesta);
    Serial.println("*****");
  }
}
else // caso contrario imprimo el codigo que entrego eñ servidor
{
  Serial.print("Error enviado POST, código: ");
  Serial.println(codigo_respuesta);
}
http.end(); // finalizo la peticion
}

//*****
//***** activacion del sueño profundo *****
//*****

contador++;//incremento cada vez que se despierta
Serial.println("*****");
Serial.println("Despierta ESP32, el numero de veces que durmio es: " + String(contador));
Serial.println("");
esp_sleep_enable_timer_wakeup(tiempo); //configuramos el tiempo que deseamos que duerma
Serial.println("Configuramos para que duerma " + String(segundos) +
" Segundos");
Serial.println("*****");
//Serial.flush();
esp_deep_sleep_start(); //con esta sentencia iniciamos el modo deep sleep
Serial.println("Esto nunca deberia imprimirse porque como mande a dormir el micro, cuando
despierta es como si se reseteara la placa, volviendo al principio del setup ");

}
}
```

12.1.16 Desarrollo de código final completo

```
//*****  
//*****CONFIRMO QUE ES EL MAS COMPLETO 4/12/21 *****  
  
/*  
Se procede a incluir en este codigo el envio de datos por metodo POST hacia un archivo php  
el cual se comunicara  
con la base de datos MySQL. Lo haremos empleando los dos nucleos principales y ademas  
activando el modo sueño  
profundo cada 1 minuto  
  
*/  
//*****  
//*****  
//librerias para la conexion WIFI - Cliente ESP32  
  
#include <WiFi.h> //la libreria WIFI.h me permite acceder a las funciones que van a *  
dejarme conectar a la red wifi  
#include <Ticker.h>  
#include <WiFiClient.h> //  
#include <WebServer.h> //  
#include <HTTPClient.h> //  
  
//*****  
//*****  
//Librerias para el sensor DHT22  
  
#include <DHT.h> //libreria para usar el DHT22  
#include <DHT_U.h> //libreria para usar el DHT22  
  
//*****  
//*****  
//librerias para manejar BMP280 y ADXL345  
  
#include <Wire.h> // se incorpora para poder trabajar con I2C  
#include <Adafruit_Sensor.h> //estas dos librerias son para emplear el sensor  
#include <Adafruit_BMP280.h>  
#include <Adafruit_ADXL345_U.h>  
  
//*****  
//*****  
//lineas definidas para la conexion WIFI
```

Sistema de monitoreo para heladas

```
#define ledWifi 13 // definimos el pin 26 como el pin de led wifi
Ticker tic_WifiLed;//creamos un objeto de la clase ticker
const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red
int cont=0; //contador usado para limitar a 50 intentos de conexion wifi

void parpadeoLedWifi()
{
    //esta funcion me permite cambiar el estado actual del led al contrario
    byte estado = digitalRead(ledWifi); //digitalRead lee el estado en cual se encuentra el
led
    digitalWrite(ledWifi, !estado);// setea el led en el estado contrario
}

//*****
//*****
// variables empleadas para almacenar la informacion de los sensores

DHT dht(12,DHT22); //indico que los datos del sensor se conectan al 32 de la placa y que
usare el sensor DHT22. El sensor solo funciona en el pin 32 o 36 de la ESP32
Adafruit_BMP280 bmp;//creamos un objeto de esa clase

String device="tarjeta1"; //es el idDevice que estamos usando en la base de datos para la
ESP8622
float temperatura_bmp280 = 0;// la temperatura que detecta el bmp290
float presion=0; // la presion que detecta el bmp290
float temperatura=0,humedad=0; //datos provenientes desde el DHT22

Adafruit_ADXL345_Unified ADXL345 = Adafruit_ADXL345_Unified();
int x=0,y=0 ,z=0; // guardare los valores de cada eje para ver si hay presencia de viento o
no, tomando un valor estatico de x=0 y=0 y Z=-10 como referencia de que no hay viento
#define alimentacion_controlada 16 //controla el VCC aplicado al BMP280
#define alimentacion_controlada2 14 //controla el VCC aplicado al DHT22
boolean Viento = true;
String viento="";
String url="http://192.168.0.123/proyectomodificado/recibe_datos.php";/*
/hola/historicos.php aca asignamos la url a la cual queremos enviarle los datos. Es
importante ver que tengo que reemplazar LOCALHOST por la ip de mi maquina ya que solamente
asi podra comunicarse la tarjeta ESP32. si fuera un servidor de internet, o haria falta.
*/

//*****
//*****
// tarea creada para poder emplear el otro nucleo del ESP32

TaskHandle_t tarea1; //para poder usar mas de 1 nucleo voy a necesitar definir tareas
```


Sistema de monitoreo para heladas

```
//*****  
//*****  
// variables empleadas para el sueño profundo  
  
#define factor_conversion 1000000 /* creamos un factor de conversion para poder pasar de  
microsegundos a segundos, debido a que la funcion que me permite configurar el tiempo que  
deseo mandar a dormir el ESP32 toma los valores en microsegundos*/  
#define segundos 49 /* asignamos el numero de segundos que deseamos mandar a dormir la  
placa*/  
int tiempo = segundos * factor_conversion ; //los segundos pasados a microsegundos  
RTC_DATA_ATTR int contador= 0; //de esta forma declaramos una variable que se almacenara en  
la memoria flash del micro procesador RTC el cual no se apaga cuando mandamos a dormir  
  
int bateria=0; //variable donde voy a guardar el porcentaje de carga de la bateria  
  
//*****  
//*****  
// librerias y variables para el uso de pantalla OLED 128x64 monocromatica  
  
//#include <Wire.h>      esta libreria se esta invocando mas arriba  
#include <Adafruit_GFX.h> // estas dos librerias son empleadas para manejar el modulo oled  
#include <Adafruit_SH110X.h> // esta especialmente solo funciona con el modulo 128x64 o  
128x32 o 128x128 pero que figura 1,3" ya que usan el microcontrolador SH1106  
#define Direccion_I2C 0x3c //defino la direccion I2C del modulo que para este caso es 0x3c  
pero puede variar  
#define Ancho_display 128 // definimos el ancho en pixeles del display  
#define Altura_display 64 // definimos el altura en pixeles del display  
#define OLED_RESET -1 // algunos modelos de pantallas incluyen el pin de reset pero en  
este caso no lo trae, por lo tanto indicaremos esto con un -1  
  
//Ahora creamos una instancia de la libreria y la llamamos display y le pasamos el ancho de  
la pantalla, alto, un puntero a la comunicacion I2C y por ultimo el parametro de pin de  
reset  
Adafruit_SH1106G display = Adafruit_SH1106G(Ancho_display, Altura_display, &Wire,  
OLED_RESET);  
  
//*****  
//*****  
  
void setup()  
{  
  
//necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a  
tomar
```

Sistema de monitoreo para heladas

```
xTaskCreatePinnedToCore(EnvioDatos,"tarea_1",10000,NULL,1,&tarea1,0); // vamos a crear una
tarea que apunte a un nucleo en particular, en este caso el nucleo N°0
    /* (le paso la funcion que quiero que se ejecute ,
    le paso un nombre a esta funcion ,
    le paso el tamaño de la pila ,
    indico algun valor que desee pasarle a la tarea que pare este caso sera NULL,
    indico la prioridad ,
    indico el nombre que le dimos a la tarea ,
    indico el nucleo donde quiero que se ejecute la tarea)
    */

Serial.begin(115200); //definimos la velocidad del puerto serial
pinMode(ledWifi,OUTPUT);
pinMode(alimentacion_controlada,OUTPUT);
pinMode(alimentacion_controlada2,OUTPUT);
digitalWrite(alimentacion_controlada,HIGH);
digitalWrite(alimentacion_controlada2,HIGH);
pinMode(27,INPUT);
ADXL345.begin();//inicializo la comunicacion con el acelerometro ADXL345
bmp.begin(0x76);//inicializo la comunicacion con el sensor bmp280
dht.begin(); //inicializo la comunicacion con el sensor dht22
if(!display.begin(Direccion_I2C, true)) //inicializamos la pantalla pasandole la direccion
del modulo dentro de un if. Si no se cumple la condicion de TRUE, imprimiremos que hay un
error de conexion a traves del puerto serial
    {
        Serial.println("Error de conexion con pantalla OLED");
    }

Serial.println("\n");
tic_WifiLed.attach(0.2,parpadeoLedWifi);// hacemos parpadear un led cada 0,2 segundos

WiFi.begin(ssid, password); //Iniciamos la conexion con la red, recibiendo como parametros
el nombre y la contraseña de la red
Serial.print("Conectando..."); //imprimimos que la conexion se esta realizando

while(WiFi.status() != WL_CONNECTED and cont < 50) //mientras no se conecte ira aumentando
un contador
    {
        cont++;
        delay(500);
        Serial.print(".");
    }

Serial.println(""); //salto de linea
```

Sistema de monitoreo para heladas

```
if(cont < 50) //si entra quiere decir que se conecto
{
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
}
else //si entra quiere decir que no se conecto
{
  Serial.println("*****");
  Serial.println("Error de conexion");
  Serial.println("*****");
}
cont=0;
tic_WifiLed.detach(); //detenemos el proceso de parpadeo de led con este comando
digitalWrite(ledWifi,LOW);//para apagar el led
}

//*****
//*****

void loop()
{
  Serial.println("*****");
  Serial.println("");
  Serial.println("El sensado de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
  temperatura = dht.readTemperature();
  humedad = dht.readHumidity();
  temperatura_bmp280 = bmp.readTemperature();
  presion = bmp.readPressure()/100;// asi como esta obenemos el dato en hPa
  analogRead(33); //para leer el valor analogico por el puerto GPIO26 de la ESP32
  bateria = map(analogRead(33), 0.0f, 4095.0f, 0, 100); //paso el valor medido en volts a
procentaje, considerando el 100% a los 3,3v
  Serial.println("Temperatura_dht: " + String(temperatura) + " Humedad_dht: " +
String(humedad)+ " Presion_bmp280: " + String(presion)+ " Temperatura_bmp280: " +
String(temperatura_bmp280)+ " Bateria[%]: " + String(bateria));

  sensors_event_t evento;
  ADXL345.getEvent(&evento);
```

Sistema de monitoreo para heladas

```
x = evento.acceleration.x;
y = evento.acceleration.y;
z = evento.acceleration.z;
Serial.print("Eje x = " + String(evento.acceleration.x));
Serial.print("\t\t\tEje y = " + String(evento.acceleration.y));
Serial.println("\t\t\tEje z = " + String(evento.acceleration.z));

//comparamos valores para ver si hay o no presencia de viento
// la referencia estatica es x=0 y=0 z= -10

if( -1 < x && x < 1)
{
    if( -1 < y && y < 1)
    {
        Viento = false;
    }
}

if( x > 5 || x < -5)
{
    Viento= true;
}

if( y > 5 || y < -5)
{
    Viento= true;
}

if(z > -9 && z < -4)
{
    Viento= true;
}

if(Viento == true)
{
    viento = "Hay viento";
    Serial.println("Hay viento");
}

if(Viento == false)
{
    viento = "No hay viento";
    Serial.println("No Hay viento");
}

Serial.println("*****");
```

Sistema de monitoreo para heladas

```
//*****
//*****
// empiezo a mostrar informacion sensada en la pantalla oled

    display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
    display.setTextColor(SH110X_WHITE);// dependiendo del modelo de pantalla puedo ajustar el
color

// muestro valor de temperatura
    display.setTextSize(1);//ajusto el tamaño de la fuente
    display.setCursor(0,0);
    display.print("TEMPERATURA: ");
    display.setTextSize(2);
    display.setCursor(20,25);
    display.print(temperatura);// cargamos el valor sensado de temperatura
    display.print(" ");
    display.setTextSize(1);
    display.cp437(true);// esta linea y la siguiente la utilizamos para imprimir el simbolo °
ya que de otra forma no se podra
    display.write(167); // el simbolo ° corresponde al caracter 167
    display.setTextSize(2);
    display.print("C");
    display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);

// muestro valor de humedad
    display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
    display.setTextSize(1);//ajusto el tamaño de la fuente
    display.setCursor(0,0);
    display.print("HUMEDAD: ");
    display.setTextSize(2);
    display.setCursor(20,25);
    display.print(humedad);// cargamos el valor sensado de humedad
    display.print(" ");
    display.setTextSize(2);
    display.print("%");
    display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);

// muestro presion atmosferica
    display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
    display.setTextSize(1);//ajusto el tamaño de la fuente
```

Sistema de monitoreo para heladas

```
display.setCursor(0,0);
display.print("PRESION ATMOSFERICA: ");
display.setTextSize(2);
display.setCursor(0,25);
display.print(presion);// cargamos el valor sentido de presion atmosferica
display.print(" ");
display.setTextSize(2);
display.print("hPa");
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);

// muestro carga de bateria
display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
display.setTextSize(1);//ajusto el tamaño de la fuente
display.setCursor(0,0);
display.print("CARGA DE BATERIAS: ");
display.setTextSize(2);
display.setCursor(35,25);
display.print(bateria);// cargamos el valor sentido de carga
display.print(" ");
display.setTextSize(2);
display.print("%");
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente
delay(2000);

// borro la ultima informacion de la pantalla para que no queden pixeles encendidos al
momento de irse a dormir el ESP32
display.clearDisplay(); //con este comando borramos cualquier contenido que se este
mostrando en la pantalla
display.display(); // con esta sentencia muestro todo lo que coloque anteriormente

//*****
//***** activacion del sueño profundo *****
//*****

contador++;//incremento cada vez que se despierta
Serial.println("*****");
Serial.println("Despierta ESP32, el numero de veces que durmio es: " + String(contador));
Serial.println("");
esp_sleep_enable_timer_wakeup(tiempo); //configuramos el tiempo que deseamos que duerma
Serial.println("Configuramos para que duerma " + String(segundos) +
" Segundos");
Serial.println("*****");
```

Sistema de monitoreo para heladas

```
//Serial.flush();

esp_deep_sleep_start(); //con esta sentencia iniciamos el modo deep sleep
Serial.println("Esto nunca deberia imprimirse porque como mande a dormir el micro, cuando
despierta es como si se reseteara la placa, volviendo al principio del setup ");

}

//*****
//*****
// rutina de envio de datos por POST

void EnvioDatos(void *parameter)
{
for( ; ; )
{
    delay(4000); //espera 4seg
    Serial.println("*****");
    Serial.println("");
    Serial.println("El envio de datos esta corriendo en el nucleo N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
    vTaskDelay(10);

    //las sentencias dentro del siguiente if se van a ejecutar solo si la placa se encuentra
conectada a la red wifi
    if (WiFi.status() == WL_CONNECTED)
    {
        HTTPClient http;// creamos un objeto de la clase HTTPClient que se encuentra en
ESP8266HTTPClient.h
        http.begin(url);//con el metodo begin inicializamos el objeto. Le pasamos como
parametro una URL
        http.addHeader("Content-Type","application/x-www-form-urlencoded");

        /*con esta sentencia, podemos especificar el tipo de contenido de la cabecera
Yo con esta declaracion, puedo enviar los datos de forma estructurada.
*/

        String datos_a_enviar ="temperatura= " + String(temperatura) + "&humedad= " +
String(humedad) + "&presion= " + String(presion)+ "&viento= " + String(viento)+ "&bateria= "
+ String(bateria) ;
```

Sistema de monitoreo para heladas

```
/* con postData indicamos la informacion que vamos a enviar, y la informacion se
envia en PARES(Variable = Valor) El nombre de la variable se pone entre comillas, cuando se
envia mas de una variable, la siguiente variable se une a la anterior usando el signo
amperson & como ocurre con "&huedad". La primera va si & Despues de la variable va el valor
que se va a enviar y debe convertirse a variable STRING, porque todo lo que se envia es
string Entre comillas se coloca la etiqueta que voy a enviar, es decir luego en el php debo
respetar ese nombre o no voy a poder entregarle el dato. Con el signo + yo CONCATENO. A
continuacion del + le paso el valor convertido a string */

int codigo_respuesta = http.POST(datos_a_enviar);/*llamamos al metodo POST haciendo
uso de http.POST y le pasamos una variable STRING La variable postData son los datos que
vamos a enviar Cuando llamamos al metodo POST, el metodo nos regresa un INT, el cual sera un
codigo de respuesta que me servira para saber si los datos se enviaron correctamente (si
regresa 200)o si no pudieron enviarse(si me devuelve un -1)*/

if (codigo_respuesta>0) //si el codigo de respuesta es mayor a cero lo imprimo por el
puerto serial
{
  Serial.println("Código HTTP: "+ String(codigo_respuesta));
  if (codigo_respuesta == 200) // si da 200 es una respuesta satisfactoria
  {
    String cuerpo_respuesta = http.getString();
    Serial.println("*****");
    Serial.println("*****");
    Serial.println("El servidor respondió: ");
    Serial.println(cuerpo_respuesta);
    Serial.println("*****");
  }
}
else // caso contrario imprimo el codigo que entrego eñ servidor
{
  Serial.print("Error enviado POST, código: ");
  Serial.println(codigo_respuesta);
}
http.end(); // finalizo la peticion
}

delay(5000); //espera 5 segundos

}
}
```

12.1.17Desarrollo de código final para controlar sistema de riego mediante wifi a través de una página web levantada desde la NodeMCU y empleando dos núcleos:

Sistema de monitoreo para heladas

```
//voy a identificar los 2 nucleos del ESP32 a traves del puerto serial

#include <WiFi.h>

TaskHandle_t tareal; //para poder usar mas de 1 nucleo voy a necesitar definir tareas

#define parpadeo 21 // segun como sea la rapidez con la cual parpadea el led, podre
identificar si esta la placa conectada o no a wifi
#define pinLed 22 //por este pin voy a entregar un 1 o un 0 desde la pagina web, ya sea para
prender un led o para realizar una accion
//const char* ssid = "BRUNO"; //coloco el nombre de la red a la que quiero conectarme
//const char* password = "eros1234"; //coloco la contraseña de la red

const char* ssid = "TP-LINK_2ECC"; //coloco el nombre de la red a la que quiero conectarme
const char* password = "brunela21"; //coloco la contraseña de la red

/*a continuacion asigno una ip estatica para la tarjeta, para que siempre sea la 108*/
IPAddress ip(192,168,0,108);
IPAddress gateway(192,168,0,1);
IPAddress submit(255,255,255,0);
WiFiServer server(80); //objeto de la clase WiFiServer
//asigno el puerto 80 que es el tcp, ya que es el puert que se
emplea para la navegacion web http

int cont1 = 0;
int cont = 0;
String header; //Usare esta variable para guardar el HTTP request
String estado = "DESACTIVADO" ;

//-----

void setup()
{
    // Inicia Serial
    Serial.begin(115200);
    Serial.println("\n");

    pinMode(parpadeo,OUTPUT);
    pinMode(pinLed,OUTPUT);
    digitalWrite(parpadeo, LOW);
}
```

Sistema de monitoreo para heladas

```
digitalWrite(pinLed, LOW);
WiFi.config(ip,gateway,subnet); //configuro la ip estatica
WiFi.begin(ssid, password); //inicio la conexion wifi

//intentare conectarme hasta que la variable cont supere las 20 veces. Mientras tanto ire
parpadeando rapidamente un led

while (WiFi.status() != WL_CONNECTED && cont<20 )
{
  cont++;
  Serial.print(".");
  digitalWrite(parpadeo, HIGH);
  delay(100);
  digitalWrite(parpadeo, LOW);
  delay(500);
}

// Si la conexion se concreto, mostrare informacion respecto a la conexion

if(cont < 20)
{
  Serial.println("");
  Serial.println("*****");
  Serial.print("Conectado a la red WiFi: ");
  Serial.println(WiFi.SSID());
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("macAdress: ");
  Serial.println(WiFi.macAddress());
  Serial.println("*****");
  server.begin(); //begin() levantamos el servidor
}

// si no llega a conectarse, indicare que hubo error y voy a reiniciar la
placa mediante el comando ESP.restart() para que vuelva al inicio del setup() e intente
conectarse nuevamente
else
{
  Serial.println("");
  Serial.println("Error de conexion");
  ESP.restart();
}

//necesito avisarle a la ESP32 que se va a ejecutar una nueva tarea ya que sino no la va a
tomar
```

Sistema de monitoreo para heladas

```
xTaskCreatePinnedToCore(loop2,"tarea_1",10000,NULL,1,&tareal,0);// vamos a crear una tarea
que apunte a un nucleo en particular, en este caso el nucleo N°0
    /* (le paso la funcion que quiero que se ejecute ,
       le paso un nombre a esta funcion ,
       le paso el tamaño de la pila ,
       indico algun valor que desee pasarle a la tarea que pare este caso sera
NULL,
       indico la prioridad ,
       indico el nombre que le dimos a la tarea ,
       indico el nucleo donde quiero que se ejecute la tarea)
    */
}

//-----
/*el void loop() tradicional por defecto se ejecutara infinitamente debido a que arduino asi
lo definio
pero este void loop2() no lo va a hacer por si solo, sino que yo debere hacer que suceda*/

// este void loop2() deberia correr en el nucleo N°0

void loop2(void *parameter)
{
    for(;;) //de esta manera logro que se ejecute infinitamente
    {
        Serial.println("*****");
        Serial.println("\t\t\tEl nucleo que esta corriendo es el N° " +
String(xPortGetCoreID())); // con la funcion xPortGetCoreID() yo voy a obtener en que nucleo
se estara ejecutando este void loop()
        cont1++;

        if(WiFi.status() != WL_CONNECTED ) // si en algun momento del bucle for infinito se
llega a perder la conexion wifi, lo que hare sera reiniciar la placa
        {
            // para que ejecute el codigo desde el inicio,
pasando por el setup() e intentando reestablecer la conexion.
            ESP.restart();
        }

        if(WiFi.status() == WL_CONNECTED )// la idea es parpadear un led cada 2 segundos como
indicador de que esta conectada la placa a la red wifi
        {
            // diferenciandose del parpadeo rapido usado para
cuando no hay una correcta conexion wifi
            digitalWrite(parpadeo, HIGH);
            delay(100);
            digitalWrite(parpadeo, LOW);
        }
    }
}
```

Sistema de monitoreo para heladas

```
    delay(2000);
  }

  Serial.println("La cuenta es: " + String(cont1));
  Serial.println("*****");
}
vTaskDelay(10);
}

//-----

void loop()

{
  delay(1000);
  WiFiClient client = server.available(); //objeto de la clase WiFiClient

  if (client)
  {
    Serial.println("Nuevo cliente..."); //Si se conecta un nuevo cliente entonces lo indicamos
    en pantalla
    String auxiliar = "";
    while (client.connected())
    { //mientras el cliente este conectado
      if (client.available())
      { //si hay bytes para leer desde el cliente
        char c = client.read(); // entonces procedemos a leer un byte y luego imprimimos ese
        byte en pantalla
        Serial.write(c);
        header += c;
        if (c == '\n')
        { /* vamos a buscar el byte que sea un caracter salto de linea
          ya que eso significaria que llegamos al fin del HTTP request del cliente, por lo cual
          procederemos a responder : */

          if (auxiliar.length() == 0) //con esto buscamos la línea en blanco
          {
            /* Este es el formato de respuesta que se respeta ante una HTTP request,
            donde se va a responder con el tipo d encabezado "HTTP/1.1 200 OK", el tipo de
            contenido que va a recibir y una línea en blanco*/
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
          }
        }
      }
    }
  }
}
```

Sistema de monitoreo para heladas

```
client.println();

//INICIA LA PAGINA

if (header.indexOf("GET /22/on") >= 0)
{
  Serial.println("GPIO 22 ACTIVO");
  estado = "ACTIVADO";
  digitalWrite(pinLed, HIGH);
}
else if (header.indexOf("GET /22/off") >= 0)
{
  Serial.println("GPIO 22 DESACTIVADO");
  estado = "DESACTIVADO";
  digitalWrite(pinLed, LOW);
}

client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");
client.println("<style>html { font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;});");
client.println(".button { background-color: #4CAF50; border: none; color: white;
padding: 16px 40px;});");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;});");
client.println(".button2 {background-color: #555555;}</style></head>");

client.println("<body><h1>SISTEMA DE RIEGO</h1>");

client.println("<p>EL SISTEMA SE ENCUENTRA : " + estado + "</p>");
if (estado=="DESACTIVADO")
{
  client.println("<p><a href=\"/22/on\"><button
class=\"button\">ACTIVAR</button></a></p>");
}
else
{
  client.println("<p><a href=\"/22/off\"><button class=\"button
button2\">DESACTIVAR</button></a></p>");
}
```

Sistema de monitoreo para heladas

```
client.println("</body></html>");

//FIN DE LA PAGINA

client.println(); // Para finalizar la respuesta HTTP request lo que debemos hacer
es enviar una línea en blanco al cliente
break; //con esta sentencia salgo del bucle
}
else
{ // limpiamos la variable auxiliar por si volvemos a utilizarla
auxiliar = "";
}
}
else if (c != '\r') //iremos agregando los bytes que llegan siempre y cuando no haya
llegado el caracter de retorno de carro \r
{
auxiliar += c;
}
}
}

header = ""; //limpiamos la variable header

client.stop(); //cerramos la conexion
Serial.println("Cliente desconectado");
Serial.println("");
}

}
```

Apéndice B

Protocolo I2C

13 Apéndice B (protocolo I2C)

I2C es la abreviatura de Inter-IC (inter integrated circuits), y es un tipo de bus diseñado por Philips Semiconductors a principios de los 80s, que se utiliza para conectar circuitos integrados (ICs). El I2C es un bus con múltiples maestros, lo que significa que se pueden conectar varios chips al mismo bus y que todos ellos pueden actuar como maestro, sólo con iniciar la transferencia de datos. Este bus se utiliza dentro de una misma placa de un dispositivo.

El bus I2C, es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de inteligencia, que sólo requiere de dos líneas

de señal. Permite el intercambio de información entre muchos dispositivos a una velocidad aceptable de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz.

La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos.

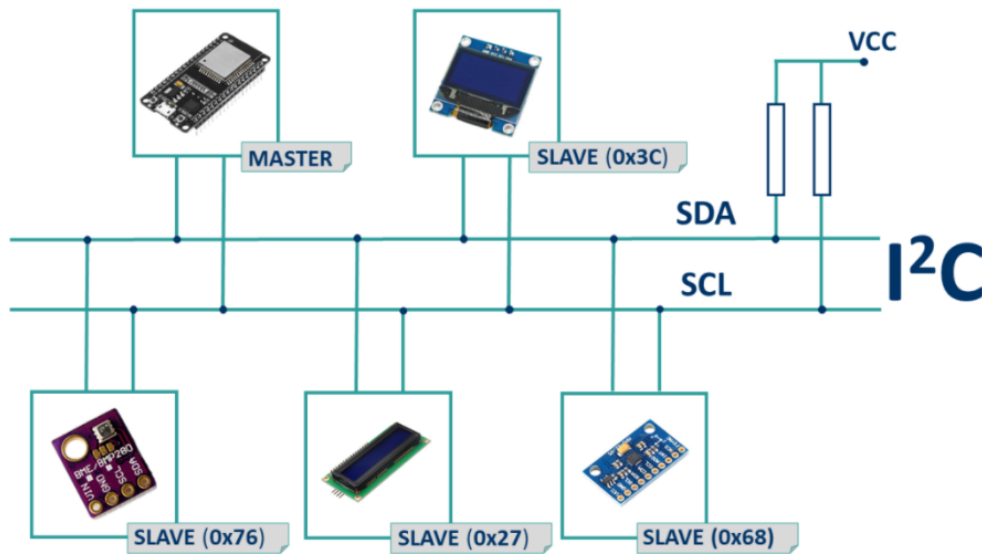


Figura N° 155 Esquema de varios esclavos comunicándose por un mismo bus a un maestro

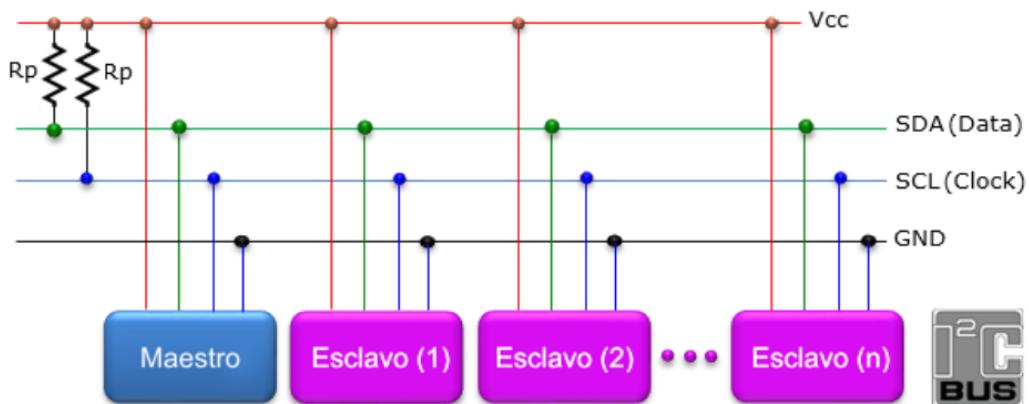


Figura N° 156 Esquema genérico del bus I2C

Las líneas SDA y SCL son del tipo drenaje abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (o FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores pull-up) lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas.

- SCL (System Clock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos enganchados al bus.
- Maestro (Master): es el dispositivo que determina los tiempos y la dirección del tráfico en el bus. Es el único que aplica los pulsos de reloj en la línea SCL. Cuando se conectan varios dispositivos maestros a un mismo bus la configuración obtenida se denomina multi-maestro.
- Esclavo (Slave): Es todo dispositivo conectado al bus que no tiene la capacidad de generar pulsos de reloj. Los dispositivos esclavos reciben señales de comando y de reloj generados desde el maestro.

Del esquema anterior, podemos ver que las dos líneas del bus están en un nivel lógico alto cuando están inactivas. En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque hay que observar que la capacidad máxima sumada de todos los dispositivos no supere los 400 pF. El valor de los resistores de polarización no es muy crítico, y puede ir desde 1,8Kohm a 47Kohms. Un valor menor de resistencia incrementa el consumo de los integrados, pero disminuye la sensibilidad al ruido y mejora el tiempo de los flancos de subida y bajada de las señales. Los valores más comunes en uso son entre 1.8Kohm y 10Kohm.

Habiendo varios dispositivos conectados sobre el bus, para establecer una comunicación a través de él se debe respetar un protocolo. En primer lugar, debemos tener presente que habrá dispositivos maestros y dispositivos esclavos. Sólo los dispositivos maestros pueden iniciar una comunicación.

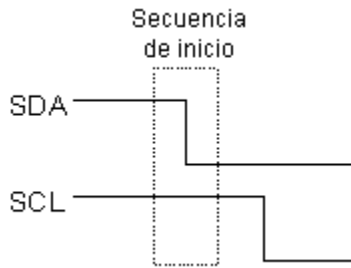


Figura N° 157 **Secuencia de inicio**

La condición inicial, de bus libre, es cuando ambas señales están en estado lógico alto. En este estado cualquier dispositivo maestro puede ocuparlo, estableciendo la condición de inicio (start). Esta condición se presenta cuando un dispositivo maestro pone en estado bajo la línea de datos (SDA), pero dejando en alto la línea de reloj (SCL).

El primer byte que se transmite luego de la condición de inicio contiene siete bits que componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura). Si el dispositivo cuya dirección corresponde a la que se indica en los siete bits (A0-A6) está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.

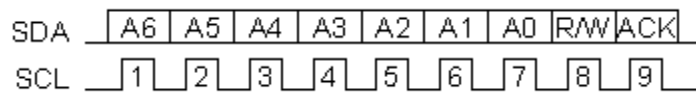


Figura N° 158 **Bits transmitidos luego de la condición de inicio**

Si el bit de lectura/escritura (R/W) fue puesto en esta comunicación a nivel lógico bajo (escritura), el dispositivo maestro envía datos al dispositivo esclavo. Esto se mantiene mientras continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se hayan transmitido todos los datos.

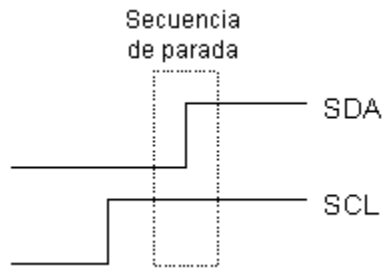


Figura N° 159 **Secuencia de parada**

En el caso contrario, cuando el bit de lectura/escritura estaba a nivel lógico alto (lectura), el dispositivo maestro genera pulsos de reloj para que el dispositivo esclavo pueda enviar los datos. Luego de cada byte recibido el dispositivo maestro (quien está recibiendo los datos) genera un pulso de reconocimiento.

El dispositivo maestro puede dejar libre el bus generando una condición de parada. Si se desea seguir transmitiendo, el dispositivo maestro puede generar otra condición de inicio en lugar de una condición de parada. Esta nueva condición de inicio se denomina inicio reiterado y se puede emplear para direccionar un dispositivo esclavo diferente o para alterar el estado del bit de lectura/escritura.