Sistema de monitoreo de Temperatura, Humedad y PH en un cultivo basado en Internet Of Things (IoT), mediante red WiFi y 4G LTE.

Jhon Alexander Camacho Herrera, Dayner Fabianni Alvarado Ojeda, Jonathan Javier Manrique Londoño

Trabajo de Grado para optar por el título de Ingeniero Electrónico

Director:

Mag. Jaime Guillermo Barrero Pérez

Magister en Potencia Eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2021

Agradecimientos.

Yo Dayner fabianni Alvarado Ojeda, quiero agradecer en primera medida al creador por la oportunidad de realizar el proceso de crecimiento personal y profesional en esta importante institución de educación superior del país y la región; a mi alma mater y mi escuela E3T UIS, quienes me acogieron desde una edad temprana cómo mi segundo hogar, me formaron, me hicieron una persona con criterio y con sentido social principalmente al darme la oportunidad de conocer un interesante y plural mundo cómo es el movimiento social y estudiantil así cómo de un universo completo de conocimiento.

A cada uno de mis docentes en la E3T, que día a día con su conocimiento, enseñanzas y exigencias nos formaron a muchos amigos y compañeros cómo profesionales, que en medio de lágrimas, alegrías, esfuerzos, trabajos y muchas experiencias de vida forman el carácter del profesional UIS, entre ellos a los profesores Jaime Barrero y David Forero quienes me ayudaron a madurar la idea de tesis y en el caso del profesor Jaime nos acompañó en el proceso cómo director de nuestro proyecto de implementación de estas nuevas tecnologías cómo las aplicaciones IOT.

Finalmente y de manera amplia agradecer a mis dos motores de vida y formadoras, mi madre Zoraida Alvarado y mi abuela Ana Clovis Ojeda, que quién junto con mis dos hermanos Angie y Yuber son mi familia y por quienes día a día lucho por superarme y ser mejor persona.

Agradecimientos.

Yo, Jhon Alexander Camacho Herrera, quiero agradecer a Dios por brindarme salud y licencia para llegar a buen término este Trabajo de Grado. El Jesucristo que siempre estuvo en pie de lucha y nunca de rodillas al régimen siempre ha sido la bandera para mi vida.

A mis padres, el Ing. Faustino Camacho Mendoza y la Ing. Hiomary Herrera Osses (hijos de esta *Alma Mater*), por el apoyo brindado y por hacerme ver la UIS como un gran referente para mi preparación profesional. También agradezco a mi nona Socorro Osses Martínez por ser una compañía fundamental en mi vida y no solo ser un apoyo para mí sino también para mis padres en momentos díficiles. Y para terminar, agradecer a mi querido hermano Santiago Camacho Herrera por ser mi luz todos los días desde el 22 de Octubre del 2002, día que jamás se me olvidará en mi vida.

A la Universidad Industrial de Santander en general, y en especial a la Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, por brindarme una formación integral tanto a nivel técnico como a nivel personal. A todos los compañeros que compartí en la universidad (tanto a los que siguen vivos como a los que han partido al cielo).

Al profesor Jaime Barrero, que con su apoyo y dedicación desde el inicio de este Trabajo de Grado fue fundamental para su culminación en buen término. También agradecimiento para mis compañeros de tesis Dayner Alvarado y Jonathan Manrique, que desde el inicio hubo un *feeling* interesante para la realización de este Trabajo de Grado.

Por último agradecer a estas personas que han sido fundamentales en mi vida universitaria: Mario Alonso, Kamela Berrocal, Aldry Mancilla, Frank Salazar, Isail Salazar, Carlos Rueda y Juan Carlos Celis. Gracias por sus consejos y en darme ánimos en los momentos más difíciles.

Agradecimientos.

Yo, Jonathan Javier Manrique Londoño, quiero agradecer a mis padres Sandra Londoño Herrera y Edwin Romero por estar presente cada día en mi desarrollo como persona y profesional, brindándome su apoyo incondicional, paciencia, cariño y las mejores energías para ayudarme a superar los momentos más difíciles en mi formación universitaria. Sin su apoyo este trabajo nunca se habría realizado.

A la Universidad Industrial de Santander y principalmente a la escuela de Ingeniería Electrónica E3T, por los conocimientos aportados en mi formación académica, que fueron bases fundamentales para el desarrollo del proyecto; A mis compañeros de la universidad por los grandes momentos compartidos en las clases y prácticas de laboratorio.

Al profesor Jaime Barrero, nuestro director de tesis, por adoptarnos, apoyarnos, impulsarnos y guiarnos con todo su conocimiento durante el desarrollo del proyecto; agradezco sus comentarios y sugerencias porque demostraba la dedicación y el interés de asesorarnos para culminar con éxito el trabajo de grado.

Gracias a mis amigos, que siempre me han prestado un gran soporte moral y humano durante estos años fuera de mi casa, a mis compañeros Jhon Camacho y Dayner Alvarado por su sincera amistad, su comprensión y por su gran apoyo en los momentos difíciles de este trabajo.

Pero, sobre todo, gracias a mi novia Laura Melissa quién ha estado a mi lado durante este proceso, por su paciencia, amor, solidaridad, apoyo incondicional, sin el cual no tendría la fuerza y energía para seguir creciendo y cumplir mi objetivo, es por eso que este título también es el suyo; Finalmente a mi abuelita Cleotilde, mis hermanas Andrea y Laura, mi hermano Cesar, mis sobrinos Sebastian y Julieta que todas las semanas me hacen videollamada.

Tabla de Contenido.

Lista de Figuras.

Pág

Lista de Tablas.

Lista de Apéndices.

Glosario

Humedad: cantidad de vapor de agua que se encuentra en el aire, suelo u otro medio. Entre mas caliente este el ambiente menor humedad podremos encontrar, pero entre mas baja o fría sea la temperatura mayor será.

Microcontrolador: es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.

Monitoreo: proceso sistemático de recolectar, analizar y utilizar información para hacer seguimiento al progreso de un programa en pos de la consecución de sus objetivos.

Módulo: elemento con función propia concebido para poder ser agrupado de distintas maneras con otros elementos constituyendo una unidad mayor.

Suelo: es la porción más superficial de la corteza terrestre, constituida en su mayoría por residuos de roca provenientes de procesos erosivos y otras alteraciones físicas y químicas, así como de materia orgánica fruto de la actividad biológica que se desarrolla en la superficie.

Temperatura: es una propiedad física que se refiere a las nociones comunes de calor o ausencia de calor, sin embargo, su significado formal en termodinámica es más complejo, a menudo el calor o el frío percibido por las personas tiene más que ver con la sensación térmica, que con la temperatura real.

Variable: que está sujeto a cambios frecuentes o probables. Característica que puede fluctuar y cuya variación es susceptible a adoptar diferentes valores.

pH: medida del grado de acidez o alcalinidad de una sustancia o una solución. El pH se mide en una escala de 0 a 14. Un valor pH de menos de 7 significa que es más ácida, y un valor pH de más de 7 significa que es más alcalina

Resumen

Título: Sistema de monitoreo de Temperatura, Humedad y PH en un cultivo basado en Internet Of Things (IoT), mediante red WiFi y 4G LTE.*

Autor: Jhon Alexander Camacho Herrera, Dayner Fabianni Alvarado Ojeda, Jonathan Javier Manrique Londoño. **

Director: Jaime Guillermo Barrero Pérez

Palabras clave: Humedad, temperatura, PH, suelo, 4G LTE, WiFi, GSM, IoT, módulo, variable, monitoreo, microcontrolador, transmisión, instrumentación, alojamiento web, comandos AT.

Descripción:

En este trabajo de grado se evidencia el proceso, desarrollo y consolidación de un módulo basado en Internet of Things (IoT), que tiene por objetivo medir en tiempos programados variables medioambientales (tales como temperatura ambiente, temperatura del suelo, humedad del suelo y PH del suelo). Las variables seleccionadas son las mínimas a tener en cuenta en las mediciones agrícolas, por lo tanto el módulo implementado funciona a nivel general con cualquier tipo de cultivo.

Generalmente, en este tipo de Trabajos de Grado en donde se aplican tecnologías IoT, se utilizan redes GSM para la comunicación móvil en caso de que no exista conexión WiFi; como diferenciador, el módulo final a desarrollar, implementar y presentar en este documento tendrá conexión a redes 4G LTE y WiFi.

A modo de presentación, los datos tomados en la medición que envía el dispositivo, se almacenan en la nube mediante una base de datos. De esa forma, el usuario del módulo podrá acceder a la plataforma mediante una página web. La información se muestra de manera gráfica sobre las muestras tomadas por los sensores, y a su vez se le enviará un mensaje de texto al celular del interesado sobre novedades en la actualización de los datos en la plataforma.

---

*Trabajo de Grado.

**Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Jaime Guillermo Barrero Pérez.

Abstract

Title: Temperature, Humidity and PH monitoring system in a crop based on Internet Of Things (IoT), through WiFi and 4G LTE network.*

Author: Jhon Alexander Camacho Herrera, Dayner Fabianni Alvarado Ojeda, Jonathan Javier Manrique Londoño. **

Director: Jaime Guillermo Barrero Pérez

Key words: Humidity, temperature, PH, soil, 4G LTE, WiFi, GSM, IoT, module, variable, monitoring, microcontroller, transmission, instrumentation, webhosting, AT commands.

Description:

This degree work shows the process, development and consolidation of a module based on Internet of Things (IoT), which aims to measure at scheduled times environmental variables (such as ambient temperature, soil temperature, soil moisture and soil PH). The selected variables are the minimum to be taken into account in agricultural measurements, therefore the implemented module works at a general level with any type of crop.

Generally, in this type of Degree Works where IoT technologies are applied, GSM networks are used for mobile communication in case there is no WiFi connection. As a differentiator, the final module to be developed, implemented and presented in this document will have WiFi connection and 4G LTE networks.

By way of presentation, the measurements sent by the device are stored in the cloud through a database. In this way, the user of the module can access the platform through a web page. The information is displayed graphically on the samples taken by the sensors, and in turn a text message will be sent to the cell phone of the person concerned about news on the update of the data on the platform.

---

*Degree work.

**Faculty of Physicomechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Director: Jaime Guillermo Barrero Pérez.

Introducción.

La población mundial alcanzará los 9.600 millones en 2050 (Morales, 2018). Por lo tanto, el futuro alimentario del mundo debe ser un tema sobre la mesa para Gobiernos, ONGs, entidades gubernamentales y no gubernamentales a nivel regional y mundial.

El desarrollo tecnológico y en especial la ingeniería no deben ser indiferentes y/o inferiores al reto que supone hacer frente a dicha tendencia mundial. Internet of Things (IoT) tiene la capacidad de transformar el mundo que vivimos. Esto llevará a que la industria sea más eficiente y que todo lo que esté a nuestro alrededor sea conectado a una red.

En la industria agrícola, el IoT tiene un impacto significativo en pro de superar las dificultades presentes en la actualidad y en el futuro del sector. Cabe resaltar los desafíos que implican el cambio climático, y el impacto ambiental de las prácticas agrícolas intensivas para satisfacer la demanda de más alimentos. Es por esto que la agricultura de precisión (ModernAg, 2017), basada en las tecnologías IoT permite a los productores y agricultores disminuir el desperdicio, explorar nuevas alternativas, prevenir plagas y/o prever variaciones que afecten la productividad de los cultivos.

En el desarrollo de este proyecto se enlazan conocimientos respectivos al área de las telecomunicaciones, dominio de los múltiples tipos de redes y/o antenas, dispositivos de transmisión-recepción, módulos y protocolos de comunicación, entre otras herramientas, que permiten la comunicación de emisor-receptor para el intercambio de información de interés. De acuerdo a las condiciones del campo colombiano y de acuerdo al apagón de las redes redes 2G y 3G proyectado para el año 2022 por parte de MinTIC (MinTIC, 2020), es preciso ajustarse en términos de las posibilidades de interconexión en el sector rural del país. Por esta razón, se decide estructurar un módulo dual, con el fin de brindarle al usuario final la opción de transmitir los datos

conectado a WiFi o conectado a un operador con redes 4G LTE en el sector.

Teniendo en cuenta que el proyecto corresponde a la construcción e implementación de un módulo IoT que permita el monitoreo de variables medioambientales de cultivos, se hace uso de la instrumentación y/o sensórica adecuada con el fin de obtener el menor porcentaje de error en la toma de decisiones referentes al proceso del cultivo. El dispositivo al ser de bajo costo se trabaja con sensores de fácil adquisición en detrimento de la precisión que podría entregar sensores de mayor rango de precio. Aún así la información suministrada por estos sensores son muy útiles al momento de hacer mediciones *in situ* y con respecto a estas, tomar decisiones que logren mantener la productividad al máximo del cultivo.

Estas condiciones aprueban tener un módulo accesible al público y/o población interesada; puesto que al poder contar con una herramienta práctica, de poco tamaño y que se ajuste a la realidad del campo colombiano. El módulo permite el acople a otras tecnologías o la implementación de mejoras en él, con el objetivo de masificarla en las UPA (unidades productivas agropecuarias)(DANE, 2016a) del país y producción a gran escala de alimentos; de igual manera el monitoreo a proyectos de recuperación ambiental mediante reforestación.

En este Trabajo de Grado se hace uso de los Comandos AT, el cual la información general pertinente a estos se encuenta en el **Apéndice A**.

1. Objetivos.

1.1. Objetivo General.

Implementar un sistema de monitoreo en un cultivo, por medio de una red de sensores articulada a una red WiFI y/o 4G LTE, que permita establecer un servicio IoT y al mismo tiempo, dar posibilidad al usuario de verificar variables medio-ambientales en tiempos programados.

1.2. Objetivos Específicos.

Diseñar y construir la unidad sensor- transmisor para monitorear el pH, humedad y temperatura de un cultivo.

Implementar el envío de la información adquirida de las variables medio- ambientales medidas, usando red WiFi y/o 4G LTE.

Diseñar las interfaces para computador o celular que permita el acceso y la consulta de información por parte de la población interesada.

2. Marco conceptual.

2.1. Intenet Of Things (IoT).

El Internet Of Thigs (IoT) se refiere a una red de “cosas” conectada a internet, en donde hay presencia de comunicación e intercambio de datos para un fin en específico(Oracle, 2021).

2.1.1. Proyección del IoT en el mundo.

Según proyecciones de expertos, se estima que para el 2025 los dispositivos IoT lleguen a una cifra considerable de 21 billones de dispositivos con aplicación IoT, con respecto a la medición del 2020 del estimado de 10 billones(Oracle, 2021). Haciendo cálculos a partir de esta proyección se tiene que los dispositivos IoT aumentarían hasta un 210%, lo cual representa un aumento signicativo de aplicaciones IoT en menos de 5 años.

2.1.2. Potencialidad del Internet of Things (IoT) en el campo colombiano.

El auge de las Tecnologías de la Información y la Comunicación (TIC) ha representado un cambio de paradigma en cómo se venían realizando procesos en todos los ámbitos de la sociedad (desde aspectos sencillos hasta trabajos complejos). El Internet of Things (IoT) es una de las ramas en donde más se aplicará las TICs en el mediano y largo plazo. Aunque en términos generales, latinoamérica se encuentra un poco por debajo de la implementación de tecnologías IoT en el campo (Soto et al., 2019), sigue siendo una oportunidad de oro con la finalidad de buscar una mayor optimización de los cultivos, puesto que aplicaciones IoT en específico como mediciones en tiempos programados de variables medioambientales, pueden ser soporte fundamental al momento de tomar medidas, pensando en mitigar riesgos provocados por sequías, heladas u otros factores que puedan incidir en el rendimiento del cultivo.

2.2. Variables más comunes para la medición de parámetros medioambientales en la agricultura.

De acuerdo al Tercer Censo Nacional Agropecuario(DANE, 2016a), se estima que hay al rededor de 11,5 millones de hectáreas de área rural, de las cuales el 38,6% tienen uso agropecuario; es por eso que para este proyecto se decide usar las variables medioambientales de los cultivos más generales y transversales a las condiciones de páramo, trópico y planicies que generan los distintos pisos térmicos característicos del país en su gran variedad de cultivos.

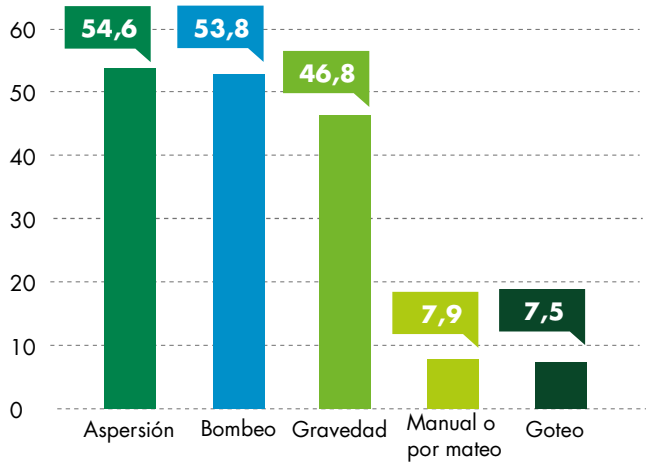
2.2.1. Humedad.

Un sistema de riego adecuado suministra la cantidad necesaria de agua en los momentos precisos, humedeciendo el suelo hasta la profundidad que requiera el cultivo. Regar en exceso puede suponer un daño perjudicial para el cultivo pues facilita el ataque de hongos y enfermedades, aparte de que conlleva a una pérdida de recursos hídricos para la finca.

Si la humedad es demasiado baja, el crecimiento de las plantas se verá comprometido, ya que los cultivos ante una falta de agua o alimento reducen su crecimiento, además para evitar la pérdida de agua por evapotranspiración la primera respuesta ante la falta de riego es la perdida de hojas (Martínez C, 2019).

Figura 1

Utilización de riego según su tipo en UPA.



Nota: Tomado de (DANE, 2016a, p.100).

Al analizar el tipo de riego, el 3er CNA (censo nacional agropecuario) halla que en el 54,6% de las UPA (unidades productivas agropecuarias) con cultivos, que utilizan riego, emplea el sístema de aspersión, seguido por el bombeo con el 53,8% (DANE, 2016a).

La humedad ambiente es importante para que la fotosíntesis sea posible; una buena humedad alrededor de la planta es todavía más importante que en otros cultivos, porque la planta únicamente puede absorber una pequeña cantidad de humedad y, por tanto, evapora menos agua que el resto de plantas. Si la planta pierde demasiada agua, los estomas se cerrarán, lo cual provocará que la fotosíntesis se frene. Si esto sucede, no podrá absorber más dióxido de carbono, y el CO_2 es necesario para mantener en marcha la fotosíntesis.

2.2.2. Temperatura.

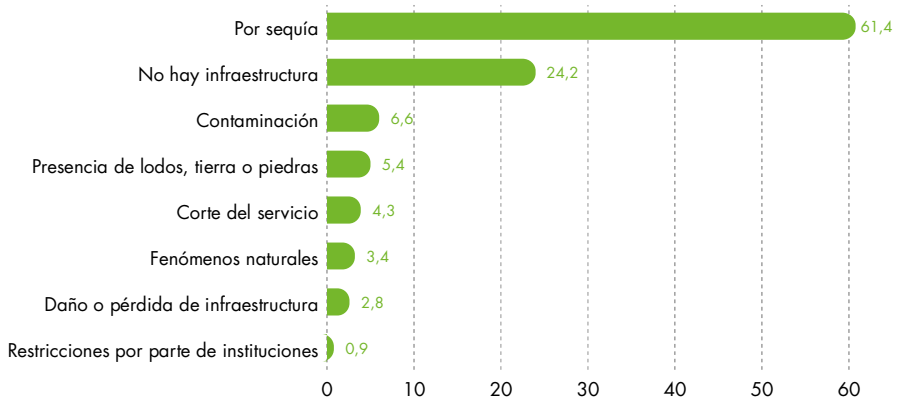
La temperatura del suelo tiene un impacto directo sobre el crecimiento y desarrollo de las plantas. Está delegada a cambios tanto estacionales como diurnos siendo más insignificantes hacia los horizontes más profundos, esta dinámica conforma el perfil térmico del suelo.

La temperatura del suelo puede ser un factor limitante en la germinación de la semilla, desarrollo y crecimiento de las raíces, así como el ritmo de degradación de la materia orgánica. Por tanto, la actividad microbiana desciende con el frente de humectación hacia horizontes más profundos en los periodos más desfavorables del año, ya sea por frío intenso o calor. Seguidamente, cuando el clima de la superficie incrementa su temperatura éstos tienden a ascender a horizontes superficiales (Ioland, 2020).

Según el CNA (DANE, 2016a), la principal dificultad de las UPA en torno al acceso al agua tiene que ver con la sequía y falta de acceso al agua en los terrenos, siendo ella la principal causa de las dificultades de las UPA que asciende al 61,4% del total de las dificultades.

Figura 2

Dificultades en cultivos en torno al uso del Agua.



Nota: Tomado de (DANE, 2016a, p.169).

La temperatura ambiente es un elemento esencial en el cultivo y desarrollo de las plantas, junto con los niveles de luz, dióxido de carbono, humedad del aire, agua y nutrientes. Esta influye en el crecimiento de la planta y la productividad de las cosechas. Estos factores deberían estar equilibrados ya que afectan a la planta tanto a corto como a largo plazo.

La mayoría de los procesos biológicos se acelerarán con temperaturas altas, lo cual puede ser tanto positivo como negativo. Un rápido crecimiento o producción de frutos es un beneficio en la mayoría de los casos. Sin embargo, la excesiva respiración que se produce es desfavorable porque implica que quedará menos energía disponible para el desarrollo de los frutos, resultando en unos frutos más pequeños. Algunos efectos se manifiestan a corto plazo mientras que otros lo harán a largo plazo. El equilibrio de asimilación de la planta, por ejemplo, se ve influenciado rápidamente por la temperatura, sin embargo, la inducción floral requerirá más tiempo (Canna Research, 2021).

2.2.3. pH.

La acidez del suelo determinará las reacciones químicas y solubilidad de los nutrientes, consiguiendo que estos puedan ser absorbidos sin problemas por las raíces. Un rango de pH inadecuado afecta directamente a la capacidad del sistema radicular, pues si los valores del pH son extremos, puede derivar en una precipitación de ciertos nutrientes haciendo que dejen de estar disponibles.

La acidez se mide en una escala de valores de pH que oscila entre 0 y 14, siendo el pH entre 5,5 y 6,5 el rango óptimo establecido para que exista una mayor disponibilidad de los nutrientes para la mayoría de los cultivos (con excepciones).

Los suelos alcalinos (con un rango superior a 7) provocan precipitados de abonos y obturación en los goteros, mientras que los suelos ácidos (con un rango menor a 5) dañan las raíces impidiendo que puedan absorber los nutrientes necesarios. Al preparar la solución de riego habrá que asegurar que el pH se encuentra en un rango entre 5,5 y 6,5 para garantizar que todos los elementos nutritivos se encuentren disponibles para la planta.

El pH de la solución de riego es esencial para una correcta gestión de la fertirrigación. Un sistema que dosifique los fertilizantes y el ácido de manera estable, sin oscilaciones en el pH y

conectividad, dará lugar a un buen desarrollo del cultivo y un aumento de la calidad y cantidad de la producción. Los equipos de fertirrigación permiten el control del pH y la CE mediante la aplicación exacta de fertilizante y ácido (Nutricontrol, 2020).

2.3. WiFi.

WiFi es una tecnología de comunicación inalámbrica que permite conectar a internet equipos electrónicos, como computadoras, tablets, smartphones o celulares, etc; mediante el uso de radiofrecuencias o infrarrojos para la trasmisión de la información.

En este sentido, la tecnología WiFi es una solución informática que comprende un conjunto de estándares para redes inalámbricas basados en las especificaciones IEEE 802.11, lo cual asegura la compatibilidad e interoperabilidad en los equipos certificados bajo esta denominación.

La comunicación inalámbrica, como tal, es aquella que prescinde de cables o medios físicos visibles de propagación, y que, por el contrario, emplea ondas electromagnéticas para su trasmisión, siendo que esta, no obstante, estará limitada a un radio específico de cobertura (Significados, 2019).

2.4. Long-Term Evolution (LTE).

Es el estándar más utilizado a nivel mundial para la transmisión de datos en redes de cuarta generación (4G). Según el informe de Marzo de 2021 de la GSA (*Global Mobile Suppliers Association*)(GSA, 2021), hay en la actualidad aproximadamente 5.8 billones de suscriptores conectados a redes LTE, representando el 62 % de los usuarios de telefonía móvil alrededor del mundo, siendo esta la red móvil mayormente usada.

Con respecto a las redes de tercera generación, la red LTE representa un incremento de hasta 15 veces en rata de transmisión de datos con respecto a redes 3G y hasta 2.5 veces con respecto a redes HSPA+ (el estándar más avanzado en redes 3G)(Thales, 2019).

La red LTE permite mayor eficiencia en el uso del espectro con respecto a redes 2G y 3G.

El VoLTE (*Voice Over LTE*) puede realizar llamadas telefónicas en alta calidad (HD) sin realizar consumo excesivo de ancho de banda puesto que la comunicación se realiza mediante paquetes de datos mediante el protocolo IP (VoLTE es el VoIP de la red LTE)(Jones & Beaver, 2021)(Huawei, 2019) en lugar de las llamadas telefónicas comunes en redes 2G y 3G.

Mediante el estándar más avanzado de las redes LTE, LTE-A (*Long-Term Evolution Advanced*), permite obtener velocidades mayores alcanzadas en el estándar LTE, principalmente por la utilización de la técnica de *Carrier Agregation* (Agregación de Portadoras), la cual permite que un dispositivo compatible con red LTE-A se pueda interconectar hasta con 5 portadoras de 20 MHz cada una, llegando a velocidades de descarga cercanas a 300 Mbps(Fitchard, 2015)(Alonso, 2017).

2.5. Transición de redes móviles en Colombia (programa Plan TIC 2018-2022).

Es un programa gubernamental del MinTIC, que tiene la finalidad de proveer mayor acceso a las herramientas TIC en el grueso de la población colombiana (principalmente en las zonas rurales, en donde la gran mayoría no tiene redes móviles 4G o conectividad fija a internet). Uno de los principales puntos de este programa es la transición de las redes 2G a redes 4G (MinTIC, 2020).

La última subasta realizada por MinTIC en el año 2019, se subastó espectro en las bandas 2(1900 MHz), 7(2600 MHz) y 28(700 MHz). A diferencia de las subastas anteriores, los oferentes tienen de compromiso cubrir un cierto número de territorios apartados del país con conectividad 4G usando la o las bandas asignadas(MinTIC, 2019).

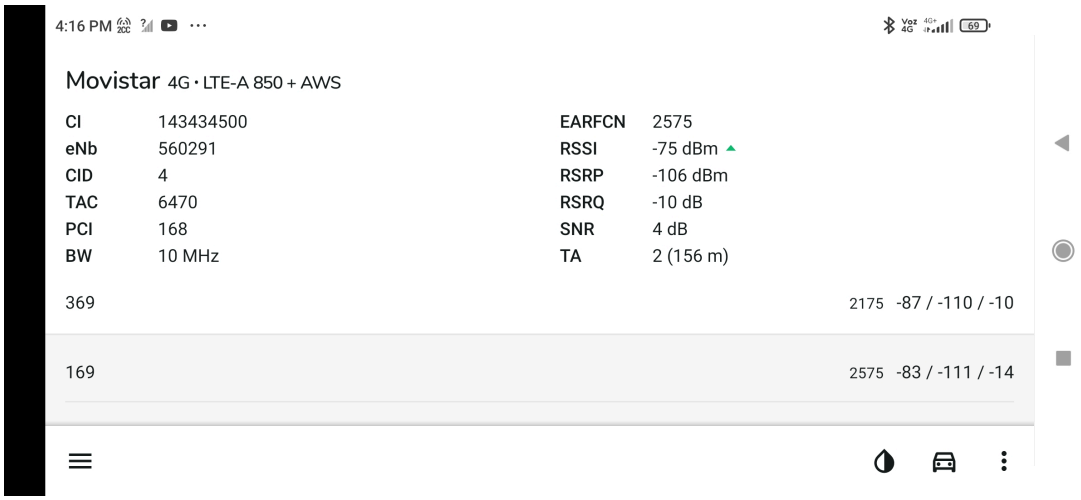
A partir del apagón programado de las redes 2G para el 2021, ya los operadores están haciendo reuso de las bandas asignadas en 2G ahora en redes 4G, y a su vez haciendo combinación de bandas mediante la técnica de *Carrier Aggregation* (o agregación de portadoras) y en los celulares se conoce con el famoso 4.5G o 4G+ (Wannstrom, 2013). A modo de ejemplo práctico,

mediante la aplicación de la Play Store de Google™ llamada Netmoster™, permite observar en el celular sobre qué tecnología móvil y a qué o a cuales bandas se está conectando el dispositivo móvil utilizado. Un dispositivo móvil puede o no hacer Carrier Aggregation dependiendo de la marca, software o hardware.

La prueba que se muestrea en la Figura 3 realiza la inspección de conexión de red en una SIM de Movistar con un celular de referencia Xiaomi Poco X3 Pro (el cual realiza Agregación de Portadoras con todos los operadores móviles en Colombia). Para este caso, se realizó la prueba mediante una SIM Card del operador Movistar, arrojando una combinación de portadoras con las bandas 4 y 5. En la figura 3 se evidencia la captura de pantalla con dicha combinación.

Figura 3

Captura de imagen de la aplicación Netmonster™ mostrando Carrier Agregation con el operador móvil Movistar, usando un Poco X3 Pro.



Nota: Pantallazo tomado por Autores.

3. Módulo IoT planteado.

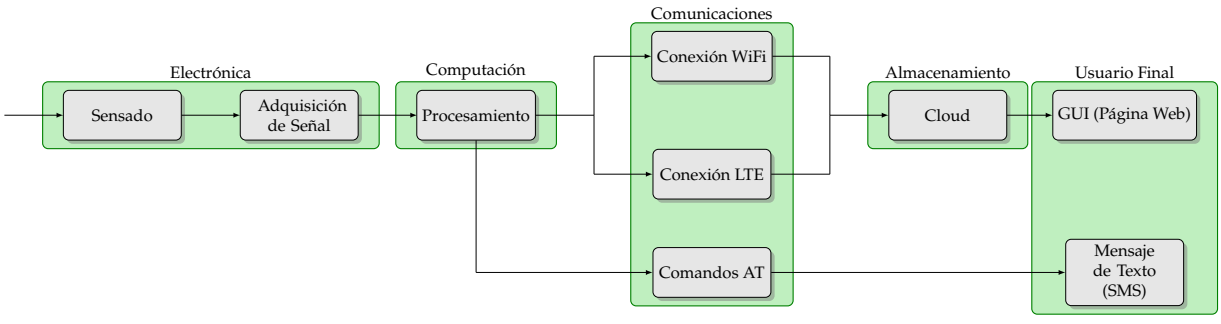
3.1. Bosquejo del módulo.

El enfoque central del Trabajo de Grado es plantear un módulo IoT, que cumpla los objetivos trazados dentro del plan inicial, ejecutando el sistema de monitoreo, teniendo en cuenta las variables medioambientales escogidas. El problema se aborda desde distintas áreas, como lo es, la Instrumentación (sensores), la selección del microcontrolador a usar, la programación y establecer la comunicación del dispositivo a la nube, el envío de los datos y del mensaje de texto (SMS) con valores promedio de las variables medidas en los tiempos programados.

Antes de entrar a hablar de temas netamente técnicos, es importante mencionar el concepto de sistema. El sistema es un objeto en el que variables de distintos tipos interactúan y producen señales observables. Estas señales que interactúan son entradas, perturbaciones y salidas (Brunete et al., 2020). A partir del entendimiento del concepto de sistema, se puede mostrar con mayor claridad el sentido del módulo mediante el siguiente diagrama de bloques:

Figura 4

Diagrama de bloques del concepto del módulo planteado.



Nota. Diagrama creado por los autores.

Este sistema comprende de una fase de sensado y adquisición de señal, una fase de

procesamiento y/o adecuación de las señales analógicas y digitales recibidas de la sensórica usada (cálculo del promedio de las mediciones, generación de los tiempos de dormir y despertar del procesador del ESP32 en pro del ahorro de energía), para continuar con la adecuación de la comunicación y transmisión de datos (red WiFi y 4G LTE para el envío de los datos a la nube, y comandos AT para el envío del valor promedio de la medida a la hora de tomar la muestra mediante mensaje de texto). Para que el usuario pueda visualizar los datos, se realiza una página web, dónde invoca los valores almacenados en la nube mediante una base de datos, el cual es lo que conforma en este caso la interfaz de usuario (GUI).

3.2. Especificaciones de diseño.

La especificación del módulo final, es que este pueda funcionar para cualquier cultivo. Las variables de medición en la agricultura, son la Temperatura del suelo, Humedad del suelo y PH del suelo (León Merchán et al., 2020). Para no solo centrar las mediciones en el suelo, el módulo también incluye medición de la temperatura y humedad ambiente, que también son de gran importancia en las condiciones medioambientales en las que se desarrollan los cultivos.

Como propuesta de diseño, se requiere especificar que el mismo módulo no sea de gran tamaño, que sea de fácil acople para otro tipo de sensórica que se desee incluir y adecuarle a futuro sistemas de alimentación. Se plantea que el mismo módulo procure un bajo consumo de energía o en su defecto se pueda programar para que realice mediciones en tiempos definidos y en otros periodos de tiempo, establecer el modo ahorro de energía o modo sleep.

3.2.1. Sensórica.

La selección de los sensores utilizados en este Trabajo de Grado deben satisfacer los objetivos trazados desde el inicio. A su vez deben ser sensores que no representen algún trabajo adicional a la hora de configurarlos en el módulo, ya que la finalidad principal, es el establecimiento

del protocolo de comunicaciones para el envío de los datos y la programación del envío de estos en tiempos programados.

Los criterios a tener en cuenta en la selección de la sensórica para este Trabajo de Grado fueron:

- Dispositivo de fácil acceso en el mercado colombiano.
- Costo medio o bajo.
- Fácil acceso a repositorios y/o documentación referente a los sensores.
- Adecuación correcta y puesta de funcionamiento en el campo colombiano.

3.2.1.1. Sensor de temperatura DS18B20. El DS18B20 es un sensor digital que tiene por finalidad medir la temperatura del suelo, utiliza el protocolo serial digital 1-Wire para la comunicación, este protocolo necesita solo un pin de datos para comunicarse. La presentación comercial más utilizada por conveniencia y robustez, es la del sensor dentro de un tubo de acero inoxidable resistente al agua, se puede observar en la siguiente figura.

Figura 5

Sensor DS18B20.

Nota: Tomado de (Murky Robot, 2021).

Tabla 1

Parámetros del sensor DS18B20.

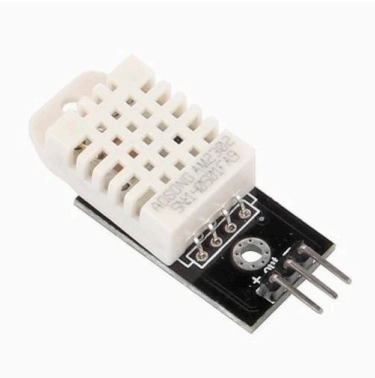
Parámetros	**Rango de Operación**
Voltaje de Alimentación	3.0 V a 5.5 V
Tiempo de respuesta	$<$ 750 ms
Rango de Medida	-55 °C a $+125$ °C
Precisión	± 0.5 °C, de -10 °C a $+85$ °C
Corriente de suministro	1.5 mA

Nota: Tomado de (Maxim Integrated, 2019).

3.2.1.2. Sensor de temperatura y humedad relativa DHT22. El DHT22 es un sensor digital de temperatura y humedad relativa de buen rendimiento y bajo costo. Integra un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos. Tambien índica el valor de la sensación térmica.

Figura 6

Sensor DHT22.



Nota: Tomado de (Arca Electrónica, 2021).

Tabla 2

Parámetros del sensor DHT22.

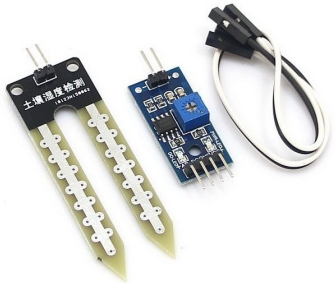
Parámetros	**Rango de Operación**
Voltaje de Alimentación	3.3 V a 6 V
Rango de Medida	Humedad: 0 % a 100 % HR
	Temperatura: −40 °C a 80 °C
Precisión	Humedad: ±0.1 % HR
	Temperatura: 0.1 °C
Corriente de suministro	1.5 mA

Nota: tomado de (Aosong Electronics Co.,Ltd, 2021).

3.2.1.3. Sensor de humedad del suelo YL-69 y YL-38. Este sensor tiene por finalidad medir la cantidad de humedad presente en el suelo que lo rodea, empleando dos electrodos que pasan corriente a través del suelo, y lee la resistencia. Consiste en una sonda con dos terminales separados adecuadamente y un módulo que contiene un integrado comparador LM393 muy estable.

Figura 7

Sensor YL-69 y YL-38.



Nota: Tomado de (Moviltronics SAS, 2021).

Tabla 3

Parámetros del sensor YL-69.

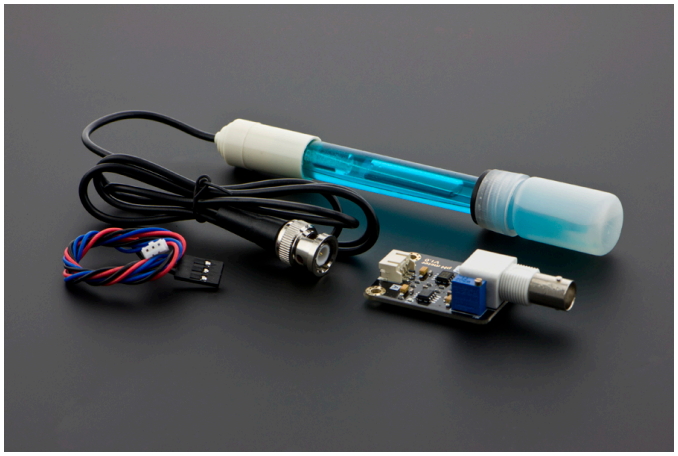
Parámetros	**Rango de Operación**
Voltaje de Alimentación	3.3 V a 5 V
Rango de Medida	0 % a 100 %
Tiempo de respuesta	10 Seg
Corriente de suministro	35 mA

Nota: Tomado de (Cubides Tórres & Manrique Suarez, 2020).

3.2.1.4. Sensor de pH SEN0161. Es un dispositivo medidor de pH analógico, el cual se compone por un electrodo que cuenta con una solución de referencia de pH conocido en su interior, y un módulo que adecua o adapta la señal para ser interpretada por el microcontrolador.

Figura 8

Sensor pH SEN0161.



Nota: Tomado de (DFROBOT, 2021).

Tabla 4

Parámetros de la sonda de pH SEN0161 y del módulo de adecuación.

Parámetros	**Rango de Operación**
Voltaje de alimentación	5 V
Rango de temperatura	5 °C a 60 °C
Punto cero	7 ± 0.5
Tiempo de respuesta	< 1 min
Rango de detección	$0 - 14$ PH
Voltaje de salida	0 V a 3 V
Presición de medición	± 0.1 a 25 °C

Nota: Tomado de (DFROBOT, 2021).

3.3. Procesamiento.

En esta etapa del proyecto, lo primero que se realiza es la adecuación de la lectura de los sensores (proporcionadas en la fase de adquisición de señal); señalando particularmente para cada caso las funciones, comandos y/o condicionales adecuados y usados de los repositorios de información de cada uno de los sensores.

En esta sección, se menciona el módulo central de procesamiento, luego se aborda el cómputo de los sensores con el módulo central de procesamiento, y su alistamiento para el envío de información en la etapa de Comunicaciones. De igual manera, se indica al final, la práctica piloto realizada con un módulo GSM antes de proceder a relatar los resultados finales de este Trabajo de Grado.

El procesamiento de datos consiste en almacenar y manipular de forma matemática una

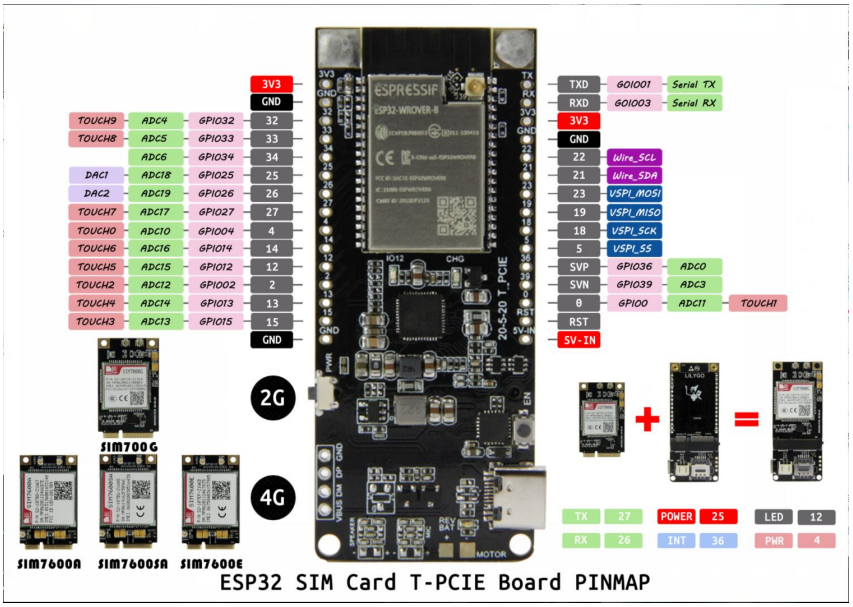
señal de información para modificarla o mejorarla en algún sentido de interés. El microcontrolador utilizado es el ESP32 (el cuál se menciona más adelante), el software empleado para el procesamiento y observación preliminar de las señales fue IDE Arduino.

3.3.1. Espressif ESP32.

Este microcontrolador es famoso en aplicaciones IoT por su buena relación costo-rendimiento, el cual permite realizar varias operaciones y mediciones sin necesidad de invertir una cantidad considerable de dinero. La ESP32 tiene por ventaja contar con dos protocolos de comunicación inalámbricas integrados en el módulo (WiFi y Bluetooth). Otra gran ventaja es la viabilidad al momento de trabajar con sensores gracias a sus puertos GPIO (que funcionan tanto como entradas analógicas como entradas digitales)(Carmenate, 2021). Para el desarrollo de este Trabajo de Grado, se hace uso de un módulo que incorpora una ESP32 llamado *LilyGO T-PCIE*.

Figura 9

Módulo LilyGO T-PCIE junto con su pinout (mapa de pines).



Nota: Tomado de (LilyGO, 2021b).

Este módulo en particular cuenta con un puerto PCIe que sirve para interconectar diferentes módulos de comunicación en telefonía celular y a su vez para el envío de datos de aplicaciones IoT mediante la red móvil.

3.3.2. Lectura sensor de humedad del suelo YL-69 y YL-38.

La señal de entrada es una señal analógica. Es por ello que se hace la lectura de la misma desde el GPIO de entrada mediante la función `analogRead`. Luego se realiza un mapeo de ese valor analógico a un rango de medición mediante la función `map`; realizando la equivalencia de valores del rango de los niveles de cuantización de 4095 a 1200 a un valor de 0 a 100, que corresponde al porcentaje de humedad.

Figura 10

Código lectura sensor YL-69.

```
1 //Lectura sensor YL 69
2
3 void loop(){
4 int const readYL69value = analogRead(YL69Pin);
5 // map valores de 0 .. 100%
6       int const convertedPercentage = map(readYL69value,
        4095, 1200, 0, 100);
7 }
```

Nota: Basado en (Petrick, 2019).

La función `map()` de Arduino™ permite transformar un valor entero de un rango de entrada al valor correspondiente a otro rango de salida. Los 5 parámetros de entrada son valores enteros:

- Valor de entrada.
- Inicio rango de entrada.
- Final rango de entrada.
- Inicio rango de salida.

- Final rango de salida.

Y la función devuelve el valor entero de salida una vez realizado el «mapeo».

3.3.3. Lectura sensor de temperatura DS18B20.

Para el procesamiento de la señal digital de entrada de este sensor, se usa las funciones de la librería `DallasTemperature`, que permite específicamente obtener la lectura del mismo al invocar las funciones `sensors.getTempCByIndex` y `sensors.getTempFByIndex` para la lectura en grados Celsius(°C) y Fahrenheit(°F), todo ello mediante el bus de comunicación 1-wire al buscar los dispositivos conectados.

Figura 11

Código lectura sensor DS18B20.

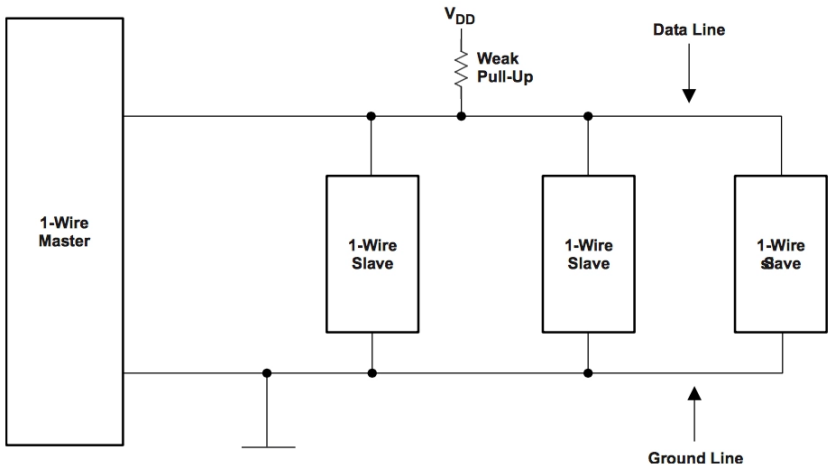
```
// Lectura sensor DSB18B20.
#include <OneWire.h>
#include <DallasTemperature.h>
void loop(){
// DSB
sensors.requestTemperatures();
float temperatureC = sensors.getTempCByIndex(0);
float temperatureF = sensors.getTempFByIndex(0);
}

```

Nota: Basado en (R. Santos, 2019a).

El bus 1-Wire consiste en un esquema de señalización, direccionamiento y arbitraje que permite comunicaciones bidireccionales entre un dispositivo maestro y uno o varios periféricos (esclavos) sobre un solo hilo de cobre o pista de circuito impreso. La siguiente imagen muestra la conexión típica de un microcontrolador con dispositivos 1-wire:

Figura 12

Topología OneWire.



Nota: Tomado de (Geek Factory, 2021).

3.3.4. Lectura sensor DHT22.

Para la lectura de la entrada digital de este sensor se usa la librería `Dht.h`, el cual define el tipo o referencia de sensor DHT y el pin o GPIO asociado a la entrada de señal. La obtención de la lectura de cada una de las variables se hace mediante el llamado de las funciones `dht.readHumidity`, `dht.readTemperature` y `dht.readTemperature(true)`. Para las sensaciones térmicas se calculan con otra función `computeHeatIndex`, que usa cómo argumento los valores de humedad y temperatura de entrada. El código también comprende una función de verificación de existencia de las lecturas tomadas y procesadas por la librería.

Figura 13

Código lectura DHT22.

```
1 //Lectura sensor DHT22
2
3 #include "DHT.h"
```

```
#define DHTPIN 32     //  pin Digital lectura de datos sensor DHT
//#define DHTTYPE DHT
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
// Inicializa sensor  DHT y sus librerías.
DHT dht(DHTPIN, DHTTYPE);
dht.begin();
void loop(){
float h = dht.readHumidity();
float t = dht.readTemperature();
float f = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
}

// Compute heat index in Fahrenheit (the default)
float hif = dht.computeHeatIndex(f, h);
// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);
}
```

Nota: Tomado de (S. Santos, 2019).

Para inicializar la lectura del sensor se usa la siguiente función: `dht.begin()` y para las lecturas las funciones `dht.read`. Se debe tener en cuenta que solo se puede captar una lectura cada 2 segundos; es por eso que se debe poner una pausa de al menos 2 segundos antes de tomar las lecturas, para estar seguro que ya han transcurrido 2 segundos después de la lectura anterior.

3.3.5. Lectura sensor de PH SEN0161.

Para la lectura de la entrada analógica de este sensor, se define una variable que aloje esta información y posteriormente mediante la función `analogRead` se hace la lectura del GPIO y se calcula el valor correspondiente mediante la fórmula $P_o = (1023 - \texttt{analogRead(pHpin)})/73.07)$. La anterior ecuación vincula los niveles de cuantización menos el valor correspondiente a la lectura sobre 73.07 para dar un valor de 0 a 14 (que es la escala de la medición del PH).

Figura 14

Código lectura sensor SEN0161.

```
//Sensor de PH SEN0161

float po;
void loop(){
Po = (1023 - analogRead(pHpin)) / 73.07; // Read and
reverse the analogue input value from the pH sensor then
scale 0-14.
}
```

Nota: Basado en (Montoya, 2018).

3.3.6. Cálculo del promedio de las variables.

Para este cálculo primero se define una serie de variables tipo `float`, variables que serán acumuladoras y que después de cumplir el ciclo repetitivo del número de muestras se les hace el llamado a las mismas, para después dividir por el número de muestras.

$$\text{Promedio} = \frac{\text{Acumpromedio}}{10} \tag{1}$$

Figura 15

Código para cálculo y almacenamiento del valor promedio de las variables medidas.

```
void loop(){
temp_amb_celcius_prom = acum_temp_amb_celcius/10;
hum_Ambiente_Array_prom  = acum_hum_Ambiente_Array/10;
temp_suelo_celcius_Array_prom =
acum_temp_suelo_celcius_Array/10;
temp_suelo_celcius_Farenheit_prom =
acum_temp_suelo_celcius_Farenheit/10;
temp_amb_farenheit_prom = acum_temp_amb_farenheit/10;
sensa_termi_celcius_Array_prom =
acum_sensa_termi_celcius_Array/10;
sensa_termi_faren_Array_prom  = acum_termi_faren_Array/10;
humedad_suelo_prom  = acum_humedad_suelo/10;
ph_liqui_prom  = acum_ph_liqui/10;
}
```

Nota: Creación propia de los Autores.

3.3.7. Modo sleep del procesador.

Para llevar a cabo la programación del modo ahorro de energía de la aplicación IOT, es decir poner en modo hibernar o ahorro de energía el procesador, se usa la función `esp_deep_sleep_start` y en el caso de desactivar el modo ahorro de energía la función `esp_sleep_enable_timer_wakeup`; esta función recibe cómo argumento el tiempo deseado en el que se requiere duerma el procesador multiplicado por la variable que almacena el número equivalente de pasar 1 μs a 1 s.

Figura 16

Código de establecimiento del modo Sleep.

```
//Modo sleep procesador y revertirlo (despertar procesador)

#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  50       /* Time ESP32 will go to sleep (in seconds) */
void loop(){
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
esp_deep_sleep_start();
}
```

Nota: Tomado de (educ8s.tv, 2018).

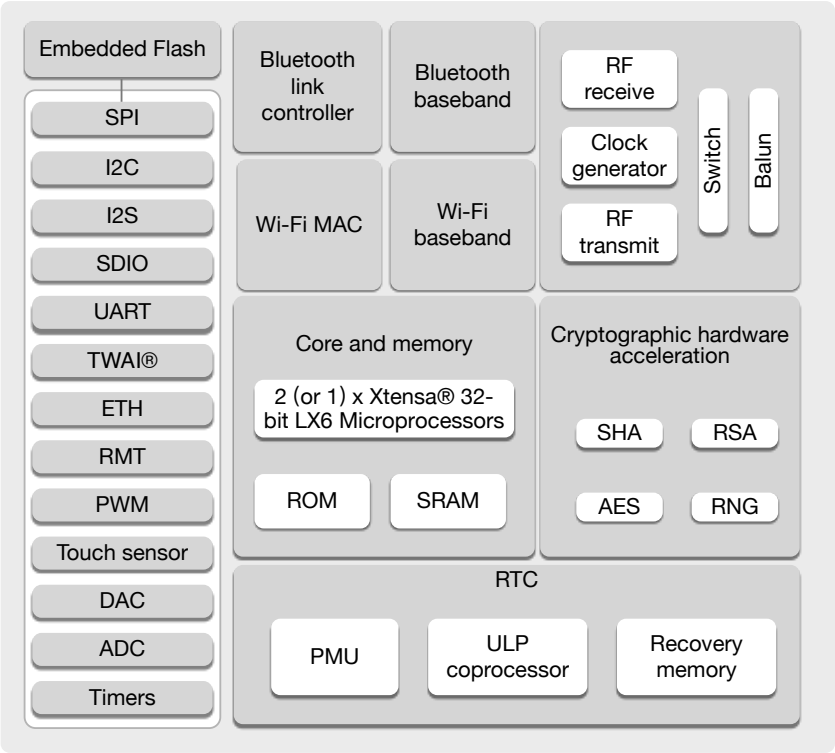
El ESP32 tiene un bloque llamado RTC. En él, se encuentra un ULP (Ultra Low Power), un PMU (Phasor Measurement Unit) y una memoria (8 KB). Todo esto conforma una etapa procesadora que funciona con muy poca energía.

Se puede establecer que, mientras el ESP32 no esté realizando operaciones de WiFi y/o Bluetooth, se deshabiliten los procesadores principales del Dual Core y actúe el RTC. También se desactiva la memoria principal, por lo cual la información que se tiene se perderá. Para que los datos de la memoria no se pierdan, se guardan en la memoria del RTC “Recovery memory”. De esa

forma, cuando se vuelve al estado Despierto, se recuperan esos datos.

Figura 17

Diagrama de arquitectura de la ESP32



Nota: Tomado de (Espressif Systems, 2021, p.12).

3.4. Comunicaciones.

El eje central de este Trabajo de Grado, es establecer la forma en cómo se van a enviar los valores ya organizados en la etapa de procesamiento hacia los dos enfoques que va a tener el usuario final: interfaz de usuario mediante una página web e información preliminar a la mano mediante mensaje de texto (SMS).

3.4.1. Comunicación vía WiFi.

En esta sección se describe la comunicación realizada mediante WiFi, la cuál consta de un llamado de variables tanto en el `void setup` cómo en el `void loop`. Primero se invoca la

librería `WiFi.h` y `http.h`. En el `void setup` se hace el llamado al SSID y el PASSWORD correspondientes a las credenciales (nombre, contraseña) de red inalámbrica a la cual se pretende acceder. Se inicia a buscar la conexión, verificación de la conexión y una vez conectada se verifica la misma al visualizar la IP local de conexión.

Una vez iniciado el `void loop`, se puede verificar que existe un cliente `http`. Luego mediante `http.begin` se hace el llamado al respectivo servidor configurado para el almacenamiento de la información. El comando `http.addHeader` permite conocer el formato y tipo de conexión. Después del proceso de tratamiento de información, se procede a organizar el empaquetado de datos para el envío de los mismos; conformado por un `api key` que funciona cómo un identificador entre el código compilado y la programación `php` del servidor web, operando así como una "contraseña" o "token" entre las partes.

La estructura del envío del paquete de información al servidor web se compone del `api key + nombre de la variable + valor de la variable`, así sucesivamente. De esta manera se empaqueta y se imprime la cadena de datos que la programación del servidor `php` logra interpretar mediante el código `post-data.php` y de esa forma extraer la información enviada para almacenarla en la base de datos del servidor utilizado. Una vez realizado este proceso, se desconecta del modo de comunicación WiFi y se reinicia de nuevo la conexión para proceder con una nueva toma de datos.

Figura 18

Código para la gestión de la conexión WiFi y envío de datos al servidor.

```
#ifdef ESP32
  #include <WiFi.h>
  #include <HTTPClient.h>
#else
  #include <ESP8266WiFi.h>
```

```
  #include <ESP8266HTTPClient.h>
  #include <WiFiClient.h>
#endif
#include <Wire.h>
// Credenciales de conexión
const char* ssid     = ""; //Usuario
const char* password = ""; //Contraseña
//Definición servidor envío datos
const char* serverName = ""; //Dirección del servidor del código post-data.php
String apiKeyValue = "tPmAT5Ab3j7F9"; //Credencial api key

void setup(){
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("Connecting");

  while(WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
      }
    Serial.println("");
    Serial.print("Connected to WiFi network with IP Address: ");
    Serial.println(WiFi.localIP());
}

void loop() {
     while(bootCount1 <= 10)
    {
      if(WiFi.status()== WL_CONNECTED){
        HTTPClient http;

      // Your Domain name with URL path or IP address with path
      http.begin(serverName);

      // Specify content-type header
      http.addHeader("Content-Type", "application/x-www-form-urlencoded");

      // Prepare your HTTP POST request data
      String httpRequestData = "api_key=" + apiKeyValue + "&value1=" + String(temperatureC) + "&value2=" + String(temperatureF) + "&value3=" + String(h) + "&value4=" + String(t) + "&value5=" + String(f) + "&value6=" + String(hic) + "&value7=" + String(hif) + "&value8=" + String(convertedPercentage) + "&value9=" + String(Po);
      Serial.print("httpRequestData: ");
      Serial.println(httpRequestData);
```

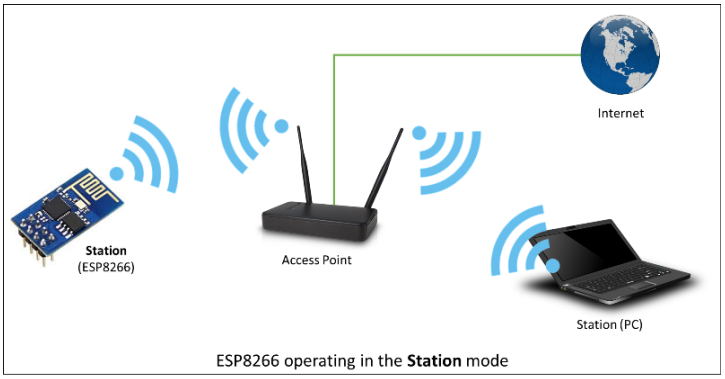
```
47
48        // Send HTTP POST request
49        int httpResponseCode = http.POST(httpRequestData);
50
51        if (httpResponseCode>0) {
52          Serial.print("HTTP Response code: ");
53          Serial.println(httpResponseCode);
54        }
55        else {
56          Serial.print("Error code: ");
57          Serial.println(httpResponseCode);
58        }
59        // Free resources
60        http.end();
61        }
62        else {
63        Serial.println("WiFi Disconnected");
64        }
65      }
66  }
```

Nota: Basado en (R. . Santos, 2019c).

Los dispositivos que se conectan a la red WiFi se llaman estaciones (STA). La conexión a WiFi es proporcionada por un punto de acceso (AP), que actúa como un centro para una o más estaciones. El punto de acceso en el otro extremo está conectado a una red cableada. Un punto de acceso generalmente se integra con un router para proporcionar acceso desde la red WiFi a Internet. Cada punto de acceso es reconocido por un SSID (Service Set IDentifier), que esencialmente es el nombre de la red que el usuario selecciona cuando conecta un dispositivo (estación) al WiFi. El módulo ESP puede funcionar como una estación, por lo que se puede conectar a la red WiFi. Y también puede funcionar como un punto de acceso wireless (SoftAP), para establecer su propia red WiFi. Por lo tanto, se puede conectar otras estaciones a dicho módulo ESP. ESP también puede operar tanto en modo estación como en modo punto de acceso (ESP8266 Arduino Core, 2017).

Figura 19

ESP8266 Conectada en modo Estación (la misma analogía se aplica con la ESP32).



Nota: Tomado de (ESP8266 Arduino Core, 2017).

3.4.2. Comunicación vía 4G LTE.

Para esta comunicación, se utiliza un módulo LTE de referencia SIM7600SA, de la marca SIMCOM, el cual se conecta con la ESP32 utilizada en el desarrollo de este Trabajo de Grado mediante una ranura PCIe.

Figura 20

Módulo SIM7600SA conectado al puerto PCIe de la ESP32 del módulo LilyGo-T-PCIE.

Fuente: (Aliexpress, 2021).

Tabla 5

Parámetros del módulo SIM7600SA.

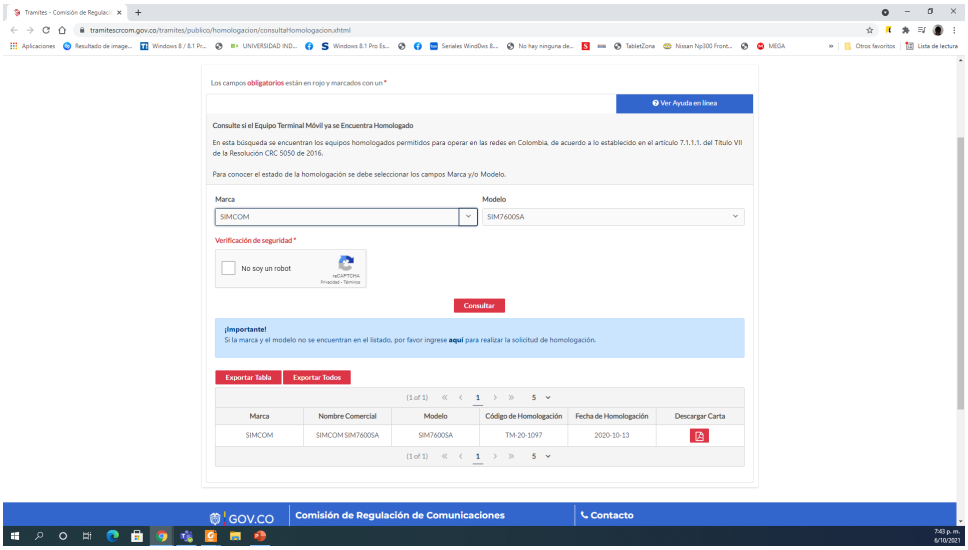
Parámetros		**Rango de Operación**
Voltaje de Alimentación		3.4 V a 4.2 V
Bandas de Frecuencia	LTE-FDD	B1/B2/B3/B4/B5/B7/B8/B28/B66
	LTE-TDD	B40
	WCDMA	B1/B2/B5/B8
	GSM	B2/B3/B5/B8
Transferencia de Datos	LTE	10 Mbps(DL)/5 Mbps(UL)
	HSPA+	42 Mbps(DL)/5.76 Mbps(UL)
	WCDMA	384 kbps(DL)/384 kbps(UL)
	GSM	236.8 kbps(DL)/236.8 kbps(UL)

Nota: Tomado de (SIMCOM, 2021).

La elección del módulo se hizo con base, tanto en temas de compatibilidad con las redes LTE existentes en Colombia como de si el módulo se encuentra homologado en el país. Para el tema de la compatibilidad en redes LTE, se debe tener en cuenta que el módulo sea compatible con todas las bandas de frecuencia usadas por los operadores móviles para dicha red. En el caso de la homologación, el módulo debe contar con la certificación de la Comisión Federal de Comunicaciones de los Estados Unidos (FCC), el cual exige la Comisión Reguladora de Comunicaciones (CRC) en Colombia para la respectiva homologación.

Figura 21

Captura de pantalla de consulta de homologación del módulo SIM7600SA en la CRC.



Nota: Captura de pantalla tomada por Autores.

A partir de estas consideraciones de elección del módulo, se procede a realizar la conexión teniendo en cuenta que las condiciones de conexión cambian con respecto a las establecidas para la conexión WiFi. En primera medida, en 4G LTE se trabaja un módulo distinto a la ESP32 y en este caso para la tarjeta utilizada viene anexo a la misma. Se inicia realizando el llamado de variables, en este caso la de configuración del APN, requisito para el uso de la sim (sus credenciales se asignan manualmente). Dichas credenciales dependen de cada operador y es obligatorio consultar en los medios de los mismos operadores esta información.

Luego de definir las credenciales, se debe especificar el acceso web en el cuál se encuentra el servidor de almacenamiento de datos y con el cuál se va a comunicar. Cómo se trabaja con un módulo independiente a la ESP32, se procede a realizar el llamado de las variables del mismo módem. Teniendo el llamado de las variables del módem, se incializa en el código al serial del módulo cómo los comandos AT. La librería que administra el módem se llama `TINY_GSM_MODEM`,

el cual es compatible para una amplia gama de módems. Como en este caso se trabaja con el módem SIM7600SA, entonces la librería queda `TINY_GSM_MODEM_7600` ya que la sigla `7600` hace referencia al módem, y este se acompaña de buffer de 1024 o un Kilobyte. Al final se hace llamado a las librerías `Wire.h` y `TinyGsmClient.h`. Luego se hace el primer llamado a los comandos AT mediante `SerialMon Serial` y `SerialAT Seriall`.

En el `void setup` se hace el llamado a los pines definidos y a los comandos AT. Se inicializa el módem, el cuál comienza haciendo la lectura si la simcard tiene o no habilitado código PIN. El código dentro del ciclo `do` es parte esencial del proceso, que es la refrendación de un comando AT en donde permite programar el módem de acuerdo a la red que se desee (en este caso está programado con la opción 2, que significa modo automático y por ende el dispositivo se conecta automáticamente a cualquier tipo de red disponible).

En el `void loop` se inicia a trabajar con el módulo conectándose a la apn especificada. Mediante un comparador se verifica la conexión exitosa o de lo contrario se recibirá un mensaje notificando el fallo. El siguiente paso es realizar la conexión al servidor en el cuál serán enviados y almacenados los datos (si es exitosa la conexión muestra ok de lo contrario un mensaje de fallo).

Punto seguido, se ejecuta un comando AT que permite comprobar a qué red y a qué banda de frecuencia de red se realizó la conexión mediante el comando `SerialAT.println("AT+CPSI?")`, este, es la comprobación de que la opción seleccionada de conexión el `void setup` es la correcta (si está automático deberá arrojar cualquiera de las 3 redes: 2G, 3G o 4G). Adicionalmente se obtiene el dato de la frecuencia a la cuál se conecta, en el caso del 4G no muestra este valor de frecuencia sino que indica el número de la respectiva banda (4 o banda AWS, el cual separa el canal de subida y bajada en dos frecuencias 1700 y 2100 Mhz, 5 o banda de 850 Mhz, 2 o banda de 1900 Mhz, 7 o banda de 2600 Mhz y 28 o banda de los 700 Mhz), todo esto de acuerdo al operador con que trabaje

el módulo.

Finalmente el empaquetado de datos es muy similar al usado en la conexión WiFi (el paquete de información generado con las funciones `client`). Por último se hace una consulta si se está conectándose y al final si después de un tiempo existe una respuesta del servidor desconecta el módulo por cada ciclo.

El comando AT que está relacionado con el parámetro establecido dentro del ciclo `do` es `AT+CNMP`. Los diferentes modos de configuración se definen en la siguiente tabla:

Tabla 6

Modos de conexión del módulo SIM7600SA, basado en el comando `AT+CNMP`.

Configuración	Tipo de Red
2	Automático
13	GSM
14	WCDMA
38	LTE

Nota: Tomado de (SIMCOM, 2021).

Figura 22

Código de ejecución 4G LTE del módulo del Trabajo de Grado.

```
1
2  const char apn[]      = "web.colombiamovil.com.co"; //
   Configuración APN
3  const char gprsUser[] = ""; // GPRS User
4  const char gprsPass[] = ""; // GPRS Password
5
6  // SIM card PIN
7  const char simPIN[]   = "";
8
9  // Server details
10
11 const char server[] = "pruebas.testiot.co"; // Dominio
12 const char resource[] = "/post-data.php";         // acceso
   al PHP
13 const int  port = 80; // Puerto
14
15 //Identificador api key
```

```
String apiKeyValue = "tPmAT5Ab3j7F9";

// Pines ESP32 referentes al SIM7600SA
#define MODEM_RST            5
#define MODEM_PWKEY          4
#define MODEM_POWER_ON       25
#define MODEM_TX             27
#define MODEM_RX             26
#define I2C_SDA              21
#define I2C_SCL              22

#define SerialMon Serial
#define SerialAT Serial1

// Configuración librería Tiny GSM
#define TINY_GSM_MODEM_SIM7600      // Modem is SIM800
#define TINY_GSM_RX_BUFFER   1024  // Set RX buffer to 1Kb

#include <Wire.h>
#include <TinyGsmClient.h>

#ifdef DUMP_AT_COMMANDS
  #include <StreamDebugger.h>
  StreamDebugger debugger(SerialAT, SerialMon);
  TinyGsm modem(debugger);
#else
  TinyGsm modem(SerialAT);
#endif

// I2C del SIM7600SA
TwoWire I2CPower = TwoWire(0);

// Cliente TinyGSM para conexión a Internet
TinyGsmClient client(modem);

#define IP5306_ADDR          0x75
#define IP5306_REG_SYS_CTL0  0x00

bool setPowerBoostKeepOn(int en){
  I2CPower.beginTransmission(IP5306_ADDR);
  I2CPower.write(IP5306_REG_SYS_CTL0);
  if (en) {
    I2CPower.write(0x37); // Set bit1: 1 enable 0 disable
    boost keep on
  } else {
    I2CPower.write(0x35); // 0x37 is default reg value
  }
  return I2CPower.endTransmission() == 0;
}

void setup() {
```

```
66
67    SerialMon.begin(115200);
68    I2CPower.begin(I2C_SDA, I2C_SCL, 400000);
69
70    // Keep power when running from battery
71    bool isOk = setPowerBoostKeepOn(1);
72    SerialMon.println(String("IP5306 KeepOn ") + (isOk ? "OK"
      : "FAIL"));
73
74    Serial.println(F("DHTxx test!"));
75
76    dht.begin();
77
78    // Set modem reset, enable, power pins
79    pinMode(MODEM_PWKEY, OUTPUT);
80    pinMode(MODEM_RST, OUTPUT);
81    pinMode(MODEM_POWER_ON, OUTPUT);
82    digitalWrite(MODEM_PWKEY, LOW);
83    digitalWrite(MODEM_RST, HIGH);
84    digitalWrite(MODEM_POWER_ON, HIGH);
85
86    SerialAT.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
87    delay(3000);
88
89    SerialMon.println("Initializing modem...");
90    modem.restart();
91
92    if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
93      modem.simUnlock(simPIN);
94    }
95
96    String result;
97
98 /*
99 Parte Importante del Trabajo de Grado, selección tipo de red
100 */
101
102 do {
103           result = modem.setNetworkMode(2);
104           delay(500);
105      } while (result != "OK");
106 }
107
108 /*
109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
110 */
111
112 void loop() {
113 SerialMon.print("Connecting to APN: ");
114       SerialMon.print(apn);
115       if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
```

```
116           SerialMon.println(" fail");
117         }
118         else {
119           SerialMon.println(" OK");
120           SerialMon.print("Connecting to ");
121           SerialMon.print(server);
122           if (!client.connect(server, port)) {
123             SerialMon.println(" fail");
124           }
125           else {
126             SerialMon.println(" OK");
127             SerialAT.println("AT+CPSI?");     //Get
                connection type and band
128         delay(500);
129         if (SerialAT.available()) {
130           String r = SerialAT.readString();
131           Serial.println(r);
132         }
133
134 // Making an HTTP POST request
135         SerialMon.println("Performing HTTP POST request...");
136         // Prepare your HTTP POST request data (Temperature
            in Celsius degrees)
137         String httpRequestData = "api_key=" + apiKeyValue +
            "&value1=" + String(temperatureC) + "&value2=" +
            String(temperatureF) + "&value3=" + String(h) +
            "&value4=" + String(t) + "&value5=" + String(f) +
            "&value6=" + String(hic) + "&value7=" + String(hif) +
            "&value8=" + String(convertedPercentage) + "&value9="
            + String(Po) ;
138
139         client.print(String("POST ") + resource + "
            HTTP/1.1\r\n");
140         client.print(String("Host: ") + server + "\r\n");
141         client.println("Connection: close");
142         client.println("Content-Type:
            application/x-www-form-urlencoded");
143         client.print("Content-Length: ");
144         client.println(httpRequestData.length());
145         client.println();
146         client.println(httpRequestData);
147
148         unsigned long timeout = millis();
149         while (client.connected() && millis() - timeout <
            10000L) {
150           // Print available data (HTTP response from server)
151           while (client.available()) {
152             char c = client.read();
153             SerialMon.print(c);
154             timeout = millis();
155           }
```

```
156         }
157         SerialMon.println();
158       //delay(250);
159         // Close client and disconnect
160         client.stop();
161         SerialMon.println(F("Server disconnected"));
162         modem.gprsDisconnect();
163         SerialMon.println(F("GPRS disconnected"));
164
165       }
166 }
```

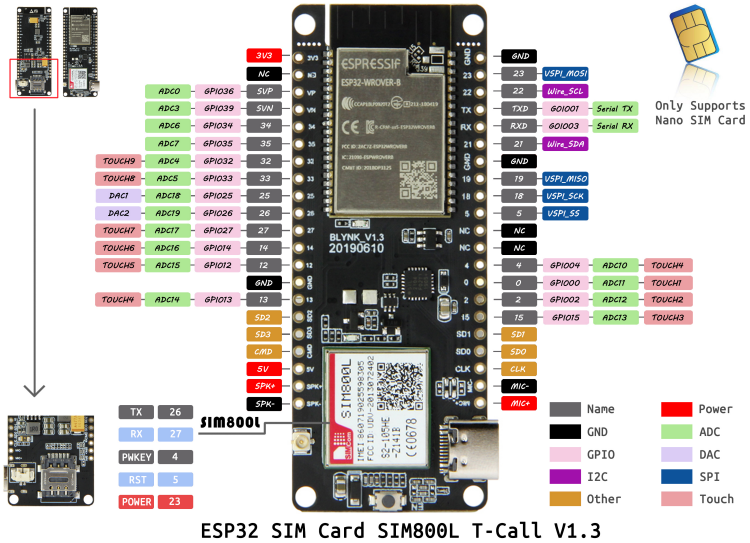
Nota: Basado en (R. . Santos, 2019b) y (LilyGO, 2021b).

3.5. Práctica piloto GSM.

A modo de hacer una prueba preliminar de funcionamiento, se implementó una práctica piloto usando un módulo con una ESP32 junto con un módulo SIM800L. En este caso se hace uso de un módulo *LilyGo T-Call SIM800* que trae integrado el SIM800L soldada a la placa base.

Figura 23

Módulo LilyGo T-Call SIM800 junto con su pinout (mapa de pines).



Nota: Tomado de (LilyGO, 2021a).

En esta prueba, se utiliza uno de los sensores a implementar en el Trabajo de Grado

(en este caso el DHT22) y la prueba consiste en enviar los datos arrojados por el sensor a un servidor web para que este los almacene. Con el resultado de esta prueba, se comprende cómo es el funcionamiento del envío de los datos al servidor web y a su vez, a la muestra de los datos almacenados en la base de datos al usuario final. Para este caso, se hace uso de tres códigos: uno en la plataforma Arduino™ y dos en `php`.

El código en Arduino™ tiene la particularidad de realizar tanto el proceso de conexión del módulo SIM800L a la red GSM como de la lectura de los sensores propiamente hablando. El código comienza haciendo llamado a las variables y a las librerías necesarias para la ejecución del código (en este caso se destacan librerías como `TinyGsmClient.h` y `Adafruit_Sensor.h`, también se hace llamado a los pines del módulo SIM800L soldado a la placa y a la información referente para la conexión a internet (APN y dirección del servidor web). Ya entrando de lleno en la programación, en el `void setup()` se inicializa el funcionamiento tanto del módulo SIM800L como del sensor DHT22, y en el `void loop()` se realiza la conexión a internet mediante el APN y el servidor web. Al final dentro del mismo `void loop()` se obtienen los valores arrojados por el sensor y se empaquetan para ser enviados al servidor a un código en `php` llamado `post-data-gsm.php` que se encuentra almacenado en el servidor web y sirve de puente entre la información entregada por la ESP32 y la base de datos. El código usado para esta prueba se encuentra en el Apéndice E.

4. Resultados.

Se implementó de manera física un módulo que logra articular la unidad sensórica con la ESP32, y a su vez se logra establecer la comunicación que permite el envío de datos a un servidor que los almacenará cumpliendo la función *Cloud* del proceso, para finalmente hacer entrega de la información mediante una interfaz de usuario. Con esto se articula una solución IoT que satisface los objetivos propuestos en el Trabajo de Grado.

4.1. Pruebas con el SIM7600SA.

Para conocer más acerca del funcionamiento del módulo, se realizan pruebas de este en base a los códigos suministrados por *LilyGO* en su repositorio en Github (LilyGO, 2021b). En él hay dos códigos de testeo (uno referente a la conexión a la red móvil y otro directamente con los comandos AT).

El código de testeo del módulo SIM7600, permite probar funcionalidades como conexión a red móvil, GPS y también el modo Sleep del ESP32 funcionando también con el módulo. El segundo código es propiamente de pruebas con comandos AT (en donde se introduce la información en la Shell de Arduino™.

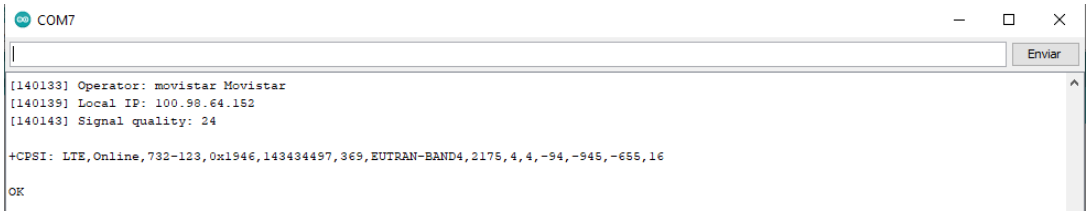
4.1.1. Prueba del código `SIM7600.ino`.

En este código realizamos la prueba de conexión y enlace a la red móvil usando el módulo SIM7600SA conectado mediante una línea PCIe a la ESP32. Acá no se realizó ningún tipo de envío sino que sólamente se conectó el dispositivo al a red mediante la APN del operador móvil.

A modo de identificación de la red en el cual el módulo se conecta, se invoca en este caso el comando `AT+CPSI?`, el cual nos arroja información referente a la red al cual se está conectando (tecnología de la red, banda de conexión y nombre del operador). Al realizar la prueba de conexión, mediante la Shell de Arduino™ podemos ver la información arrojada por dicho comando AT.

Figura 24

Captura de la Shell de Arduino™ de la prueba realizada con el código `SIM7600.ino`.



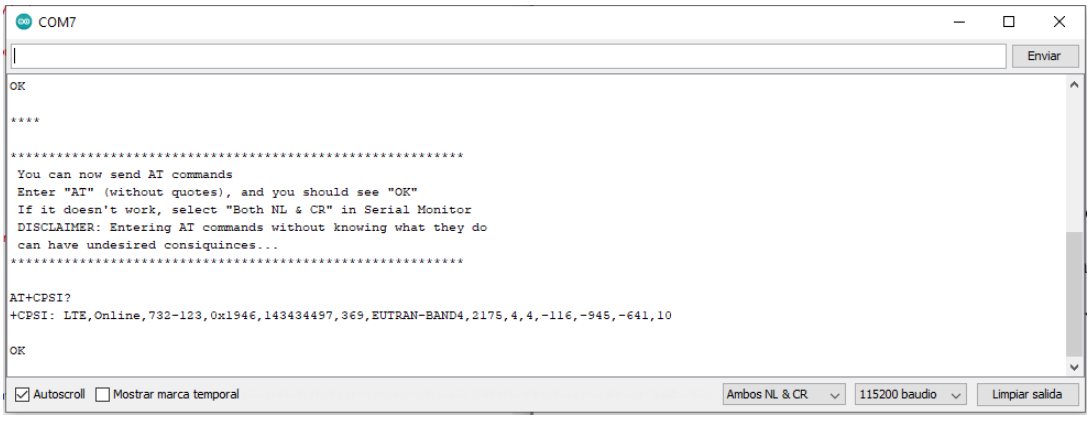
Nota: Captura tomada por Autores.

4.1.2. Prueba del código `ATDebug.ino`.

En este código se trabaja directamente con Comandos AT, los cuales se introducen en la Shell de Arduino™. Para que la Shell detecte bien el comando, se debe poner en modo `NL y CR`. El comando AT a usar en esta prueba va a ser el mismo que nos arroja a qué tipo de red se está conectando el módulo (`AT+CPSI?`). En esta prueba no hay que especificar el APN del operador.

Figura 25

Captura de la Shell de Arduino™ de la prueba realizada con el código `ATDebug.ino`.



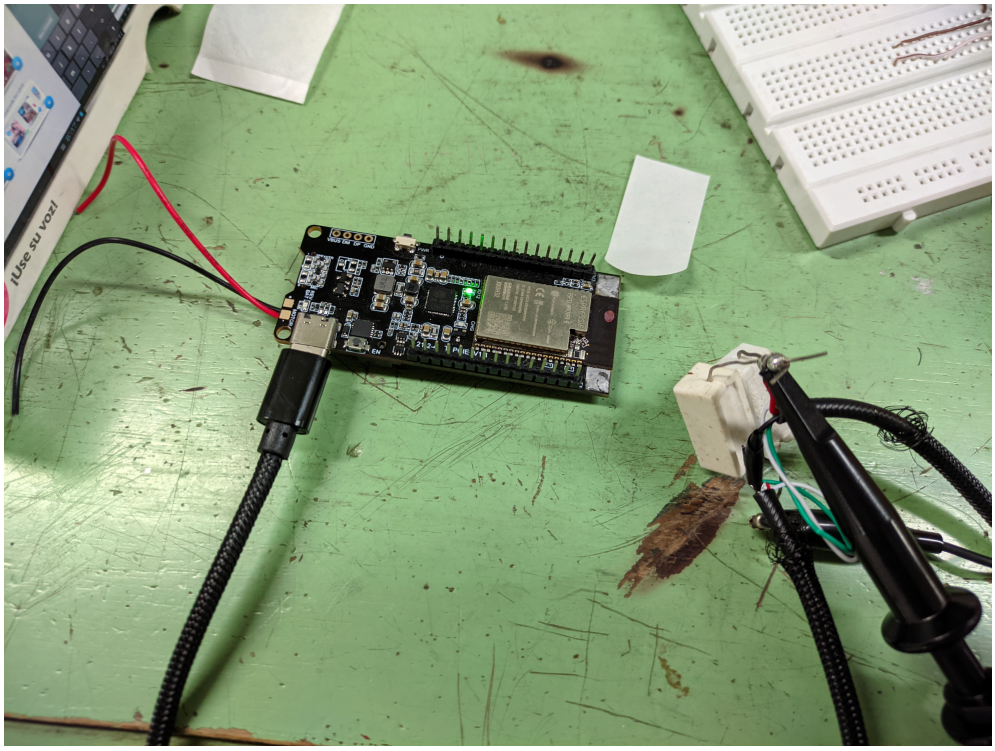
Nota: Captura tomada por Autores.

4.2. Mediciones de consumo.

Para tener una mayor idea del consumo del módulo, se realizó una medición de corriente consumida por el módulo en su ejecución. Para la realización de esta prueba, se conectó en serie a la línea VCC del cable USB una resistencia en serie de 1 Ω, y en los terminales de la misma se conectó un osciloscopio. Con esta medición, se observó los picos de la caída de tensión de la resistencia, y mediante la Ley de Ohm ($V = IR$), se calculó la corriente de manera indirecta usando de valor de referencia el voltaje pico máximo mostrado en la prueba.

Figura 26

Prueba de medición de corriente, usando una configuración de resistencias en paralelo, con valor final de 1 Ω*, conectadas en serie.*



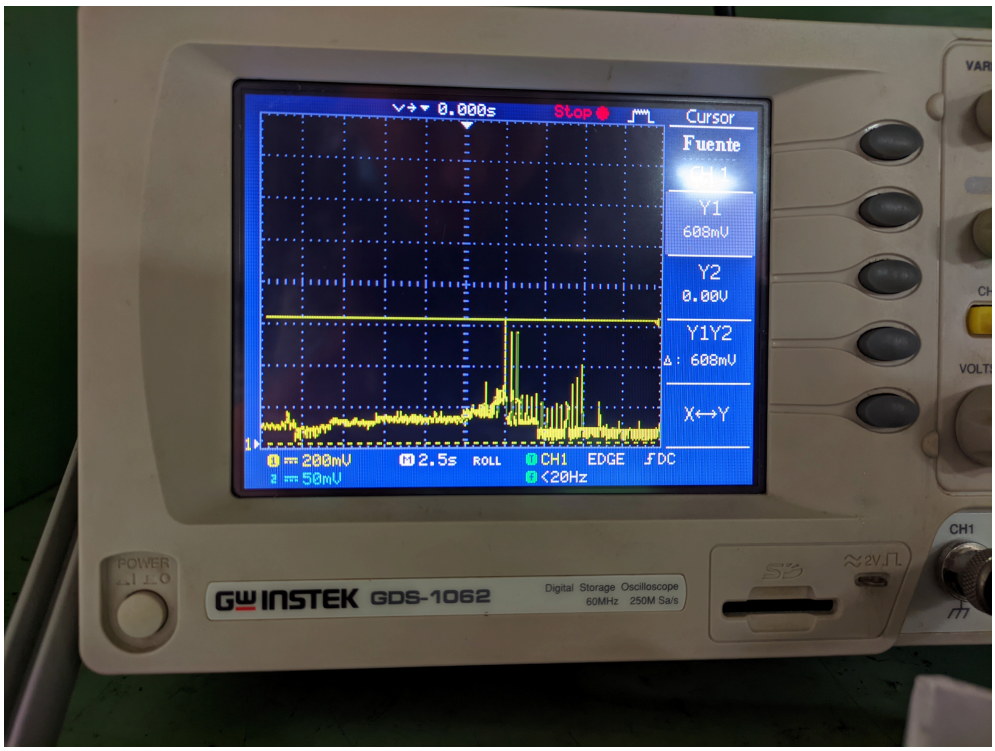
Nota: Prueba realizada por Autores.

4.2.1. Medida de corriente en ejecución del código `SIM7600.ino`.

Esta prueba tiene por finalidad medir el consumo máximo del módulo cuando este realiza la conexión a la red móvil. Se hace uso de una ESP32 *LilyGO T-PCIE* y conectada a su línea PCIe el módulo SIM7600SA. Se programa esta ESP32 con el código `SIM7600.ino`. La medición de pico máximo de corriente nos indica el consumo total de la ESP32 más el SIM7600SA. Al realizar diferentes medidas, se lograron valores de voltaje pico en la caída de resistencia dentro del rango de 400 mV y 610 mV.

Figura 27

Captura de pantalla del osciloscopio con la medición del valor pico máximo de tensión de la ESP32 junto con el SIM7600SA, en el modo de conexión a red móvil.



Nota: Prueba realizada por Autores.

En la anterior figura, se puede evidenciar que el pico máximo de la tensión es de 608 mV, el cual teniendo en cuenta que la resistencia en donde se mide la caída de tensión es de 1 Ω, teniendo

en cuenta la Ley de Ohm, se obtiene un valor de corriente de 608 mA.

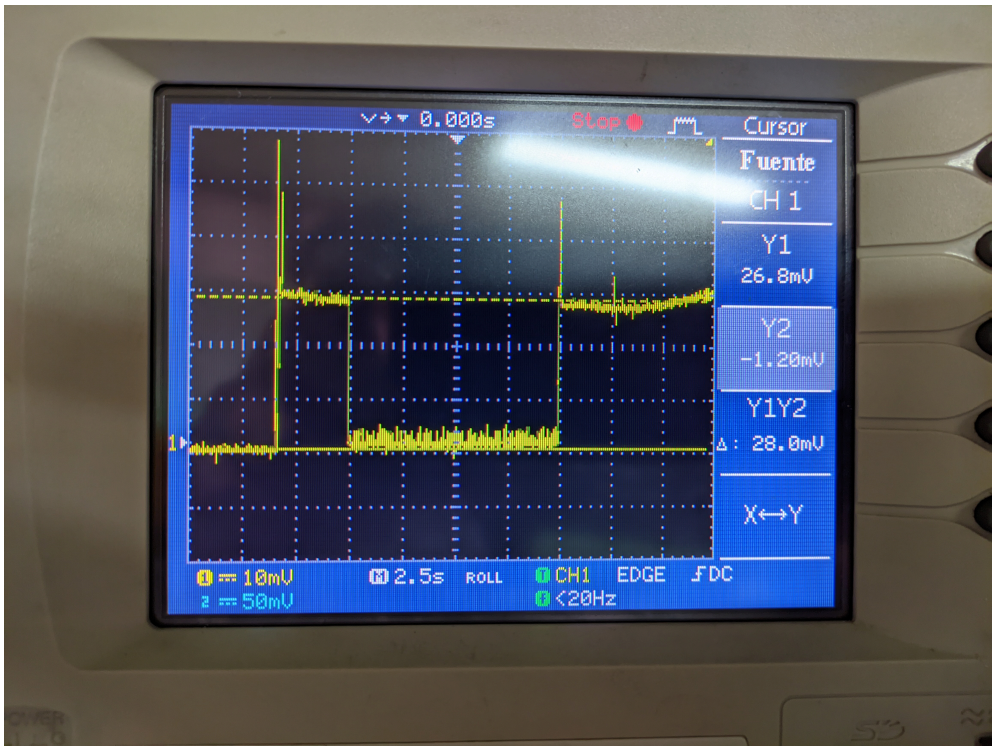
4.2.2. Medida de corriente del modo Sleep.

Para la medición del modo *Sleep*, se utilizó una ESP32 *LilyGO T-PCIE*, pero esta vez sin ningún modulo móvil conectado en su puerto PCIe. En cuanto a la programación, se realiza mediante un código en Arduino™ ejecutando la función `esp_deep_sleep_start()`.

Para llevar a cabo la prueba, se programa el módulo que duerma por 10 segundos. En el osciloscopio se observó la tensión en la resistencia de 1 Ω tanto en el modo Despierto como en el modo *Sleep*, los cuales se observan en la figura 28 y 29.

Figura 28

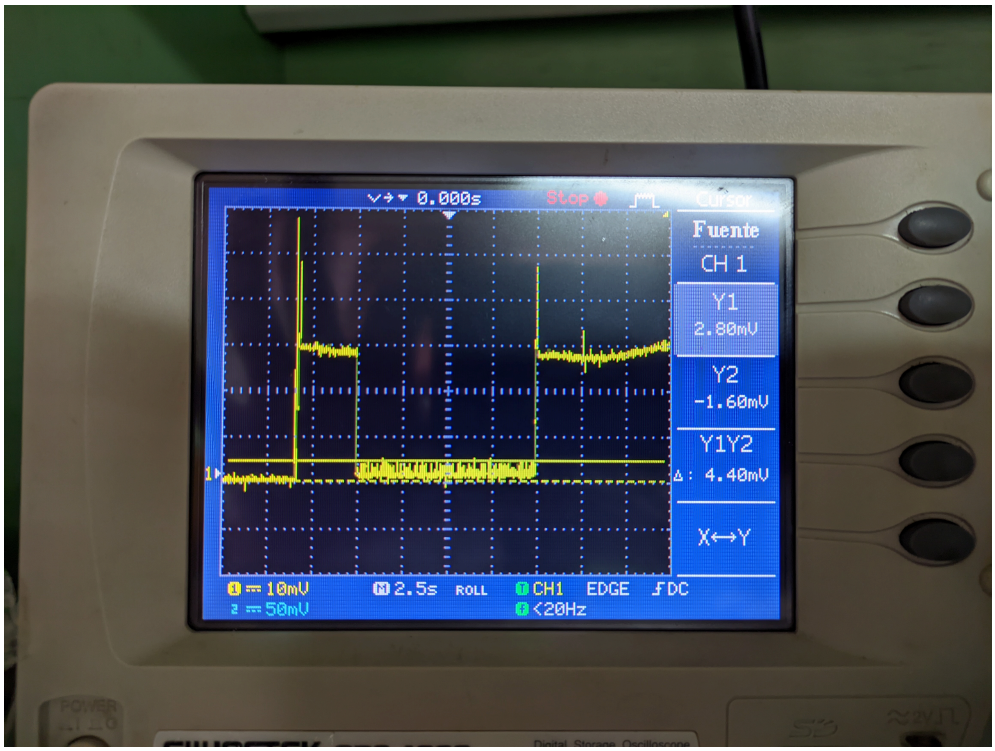
Captura de pantalla del osciloscopio con la medición de los valores picos en modo Despierto y en modo Sleep. Medición de voltaje en el modo Despierto.



Nota: Prueba realizada por Autores.

Figura 29

Captura de pantalla del osciloscopio con la medición de los valores picos en modo Despierto y en modo Sleep. Medición de voltaje en el modo Sleep.



Nota: Prueba realizada por Autores.

La anterior medición deja como resultado, el valor pico de voltaje de 28 mV para el caso del modo Despierto y 4.4 mV para el modo *Sleep*. Teniendo en cuenta la resistencia de 1 Ω en donde se realizó la medida, los valores de corriente en modo Despierto y *Sleep* son, 28 mA y 4.4 mA, respectivamente. Según esto, el modo *Sleep* representa un consumo de aproximadamente 84.29 % menos con respecto al modo Despierto, lo cual es una muestra que programar el módulo a dormir luego de tomar la muestra representa un ahorro significativo de energía.

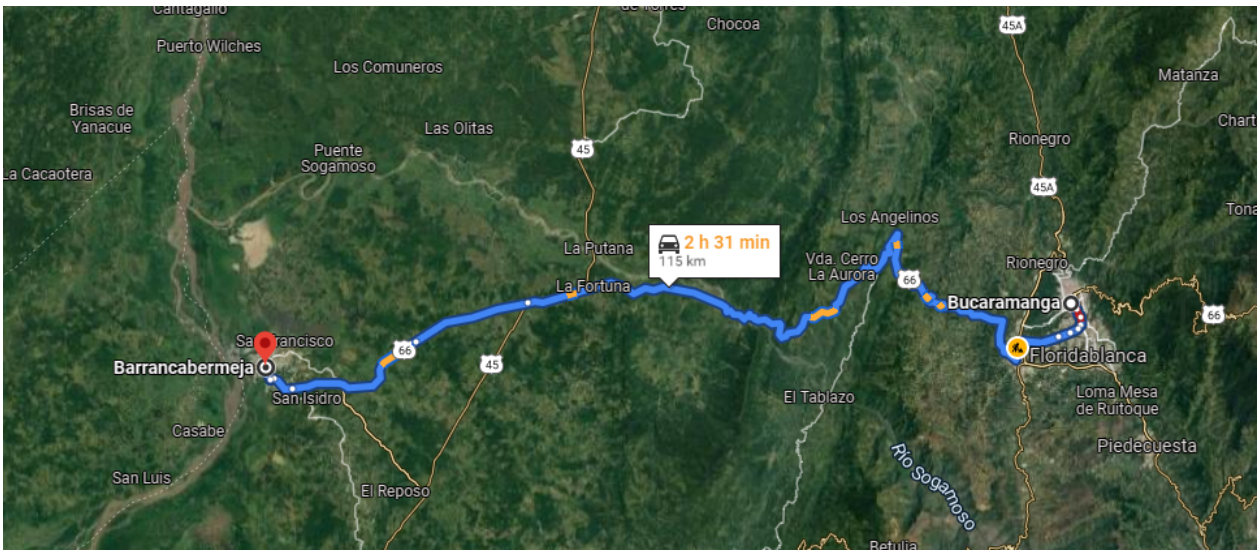
4.3. Pruebas de conexión *in situ*.

Para el desarrollo de esta prueba, se recorrió parte del trayecto entre la ciudad de Bucaramanga y Barrancabermeja en un carro, en búsqueda de zonas en dónde un celular Poco™ X3

NFC con una SIM del operador Tigo se conectara exclusivamente en red 2G o 3G, para luego hacer una prueba *in situ* de conexión del módulo SIM7600SA usando otra SIM de Tigo para corroborar que el modo automático del comando `AT+CNMP` se conecta a la red presente en la zona de medición. Para esta prueba se realiza mediante el código `SIM7600.ino`.

Figura 30

Vía Bucaramanga-Barrancabermeja.



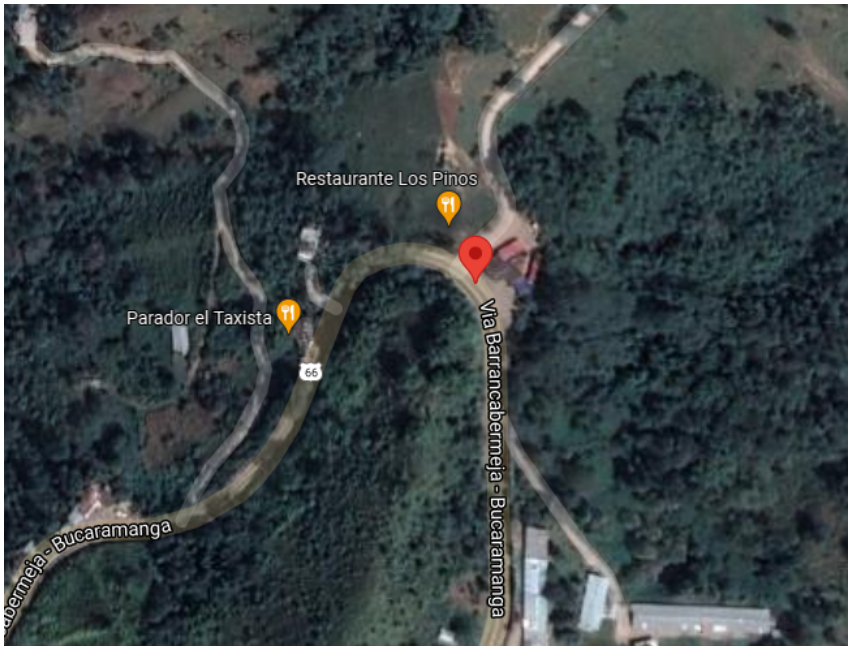
Nota: Captura tomada por Autores mediante Google Maps™.

4.3.1. Conexión in situ a red 2G con el SIM7600SA.

En la prueba para red 2G, se encontró en el trayecto de la vía Bucaramanga-Barrancabermeja en el sector del "Restaurante Brisas", ubicado aproximadamente en las coordenadas `7°10'51.2"N 73°16'45.5"W`. En esta zona, el celular se conectó únicamente a red GSM y de acuerdo a esto, se procede a realizar la prueba de conexión con el SIM7600SA configurado en modo automático.

Figura 31

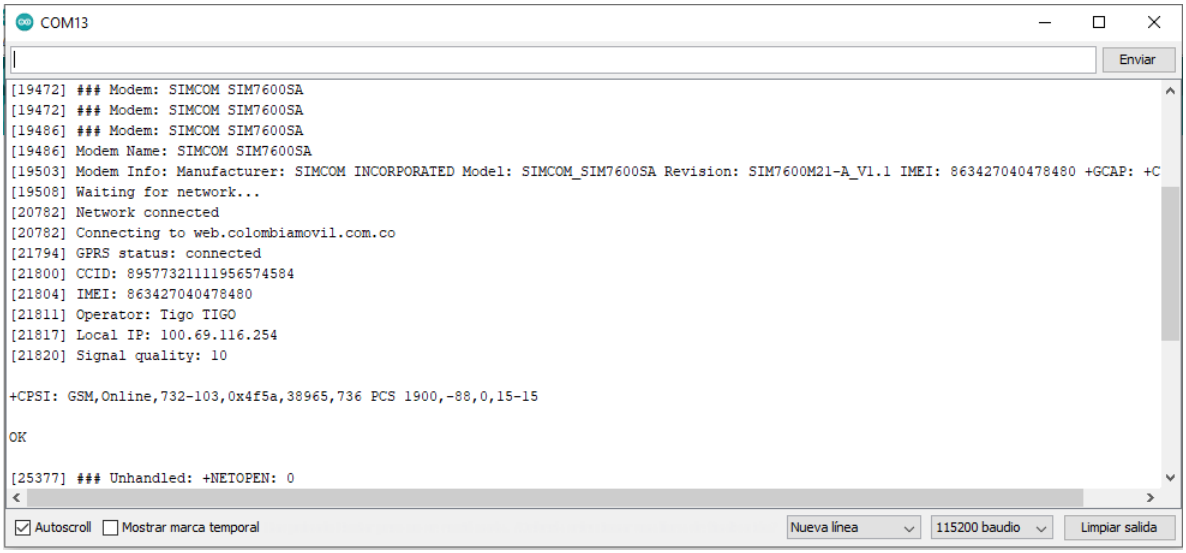
Punto de medición red GSM con el SIM7600SA.



Nota: Captura tomada por Autores mediante Google Maps™.

Figura 32

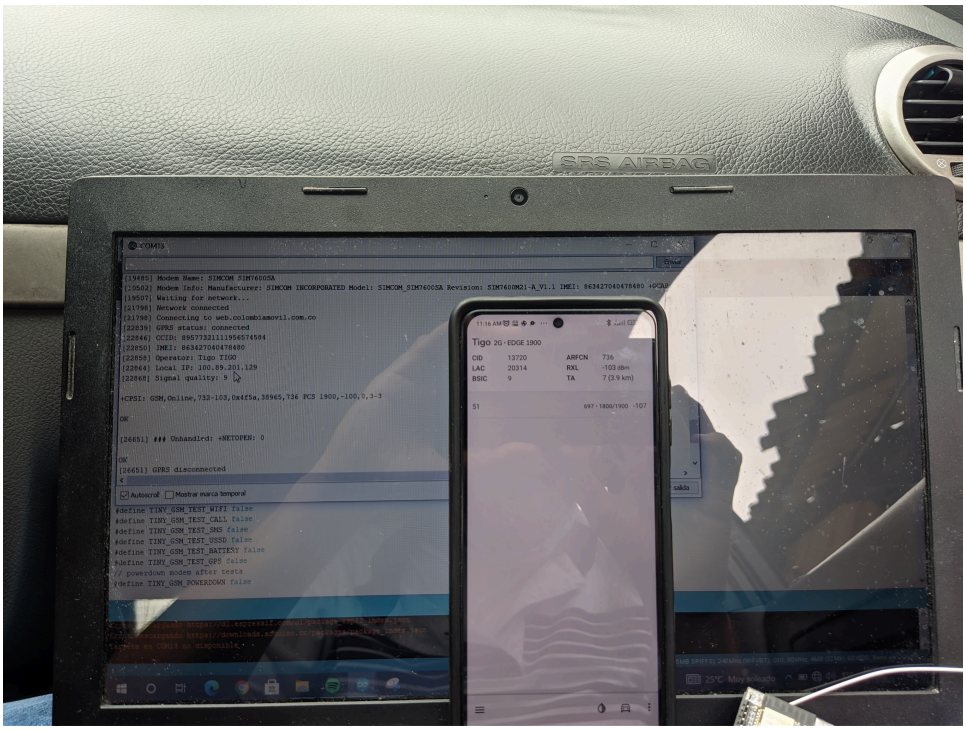
Captura de la Shell de Arduino™ con la prueba de conexión a red GSM in situ.



Nota: Captura tomada por Autores.

Figura 33

Fotografía de prueba in situ de la conexión GSM con el módulo SIM7600SA, junto con la información arrojada por la aplicación Netmonster™ en un Poco™ X3 NFC.



Nota: Foto tomada por Autores.

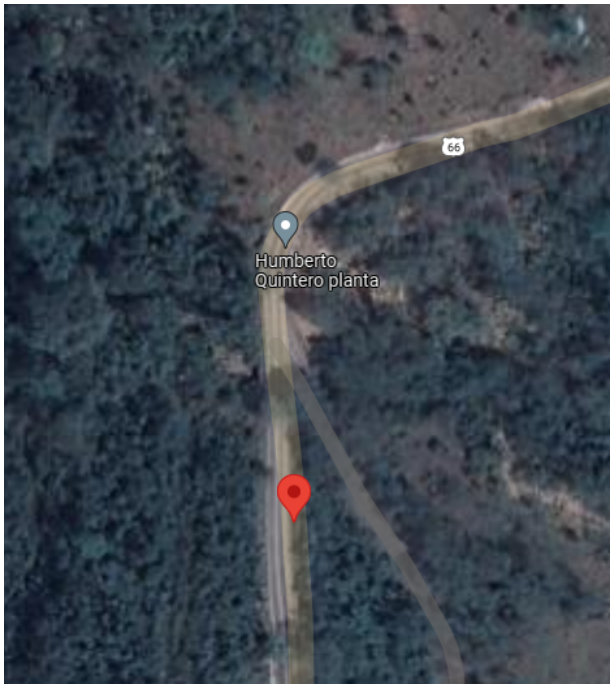
En la anterior imagen queda certificada la prueba en red GSM en un punto en específico en donde solo hay presencia de dicha red en la zona, lo cual confirma que el modo automático del comando `AT+CNMP` funciona correctamente en el SIM7600SA.

4.3.2. Conexión in situ a red 3G con el SIM7600SA.

Para esta prueba, se avanzó más por la carretera hasta llegar a la vía sustitutiva del sector de Hidrosogamoso. Aproximadamente en la coordenada `7°07'00.8"N 73°20'07.2"W`. En este caso, el celular se conectó a la red HSPA+ (el cual se considera red 3G) y se procedió a realizar la prueba de conexión con el SIM7600SA configurado en modo automático.

Figura 34

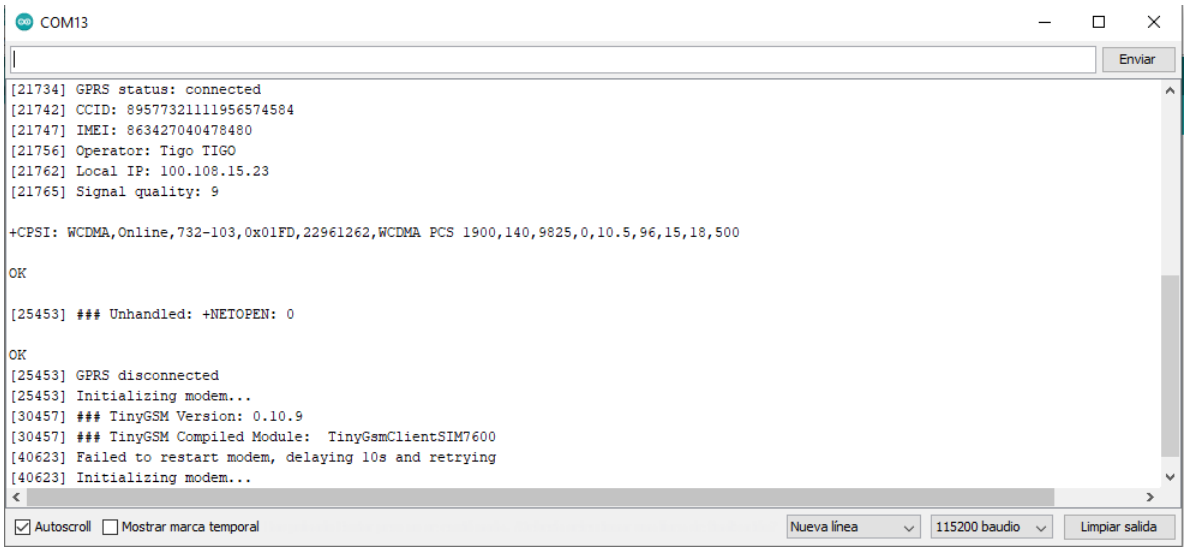
Punto de medición red WCDMA con el SIM7600SA.



Nota: Captura tomada por Autores mediante Google Maps™.

Figura 35

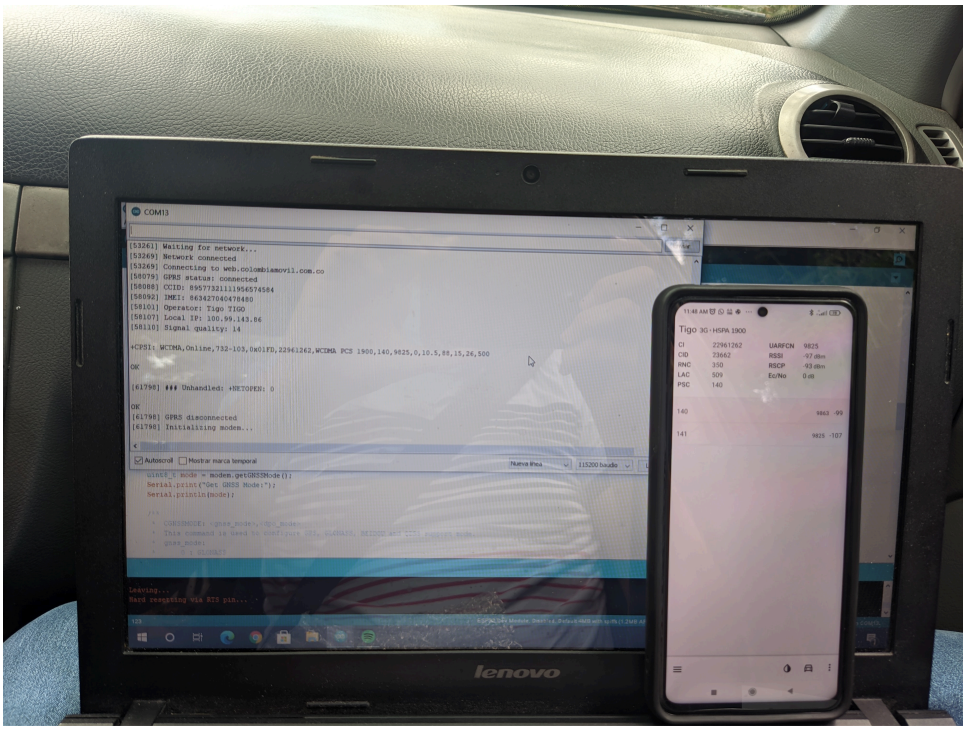
Captura de la Shell de Arduino™ con la prueba de conexión a red WCDMA in situ.



Nota: Captura tomada por Autores.

Figura 36

Fotografía de prueba in situ de la conexión WCDMA con el módulo SIM7600SA, junto con la información arrojada por la aplicación Netmonster™ en un Poco™ X3 NFC.



Nota: Foto tomada por Autores.

En conclusión, el módulo SIM7600SA logra conectarse automáticamente a la red presente en el sector, lo cual muestra una enorme potencialidad a la hora de realizar aplicaciones IoT en zonas apartadas ofreciendo mayor flexibilidad en conexión a redes móviles.

4.4. Conexiones físicas en la protoboard.

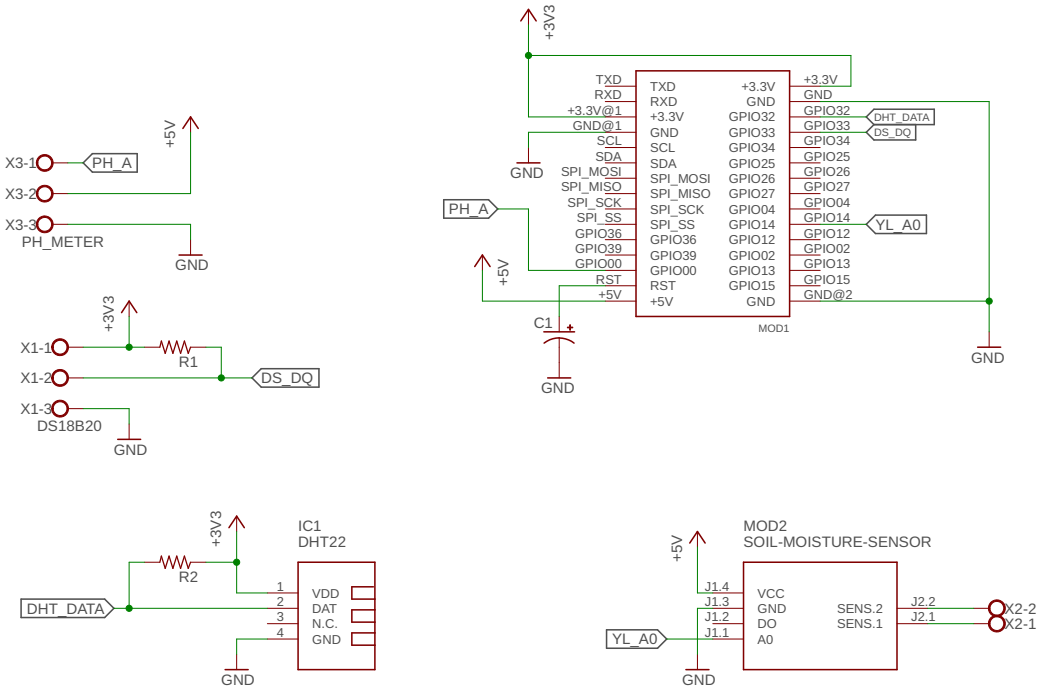
La construcción del módulo final comprende de manera física la interconexión entre los distintos sensores, los planos de alimentación 3.3 V y 5 V, plano de tierra y los respectivos GPIO del ESP32 usando los siguientes pines y generando el siguiente esquemático de conexiones:

- Sensor YL69 al pin 14.
- Sensor de PH al pin 0.
- Sensor DHT al pin 32.
- Sensor DSB al pin 18.

- MODEM RST al pin 5.
- MODEM PWKEY al pin 4
- MODEM POWER ON al pin 25
- MODEM TX al pin 27
- MODEM RX al pin 26
- I2C SDA al pin 21
- I2C SCL al pin 22.

Figura 37

Plano esquemático del módulo implementado.



Nota: Diagrama creado por Autores.

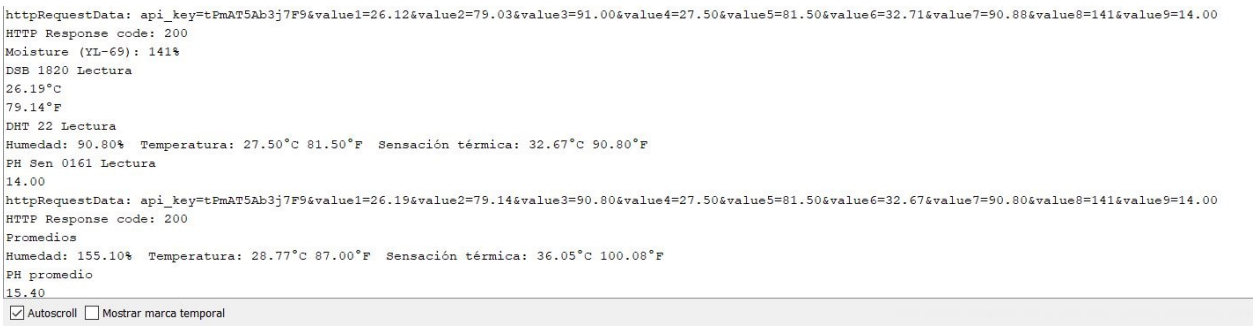
4.5. Envío de datos de los sensores vía WiFi.

Una vez implementadas las conexiones del módulo y de realizar las primeras pruebas de lectura de los datos referenciadas en el capítulo anterior, se procede a la adecuación, análisis y apropiación de los comandos, líneas de código y comandos que intervienen en el proceso de conexión vía WiFi; obteniendo un proceso exitoso en torno a la programación para dormir el procesador del ESP32 y despertarlo en tiempos programados, la conexión y desconexión de red

WiFi, el ciclo repetitivo que permite la toma de 10 muestras por variable, cálculo de su promedio e interconexión con el enlace web que aloja el servidor y permite la visualización de resultados.

Figura 38

Captura de la Shell de Arduino™ con la prueba del envío de los datos vía WiFi.



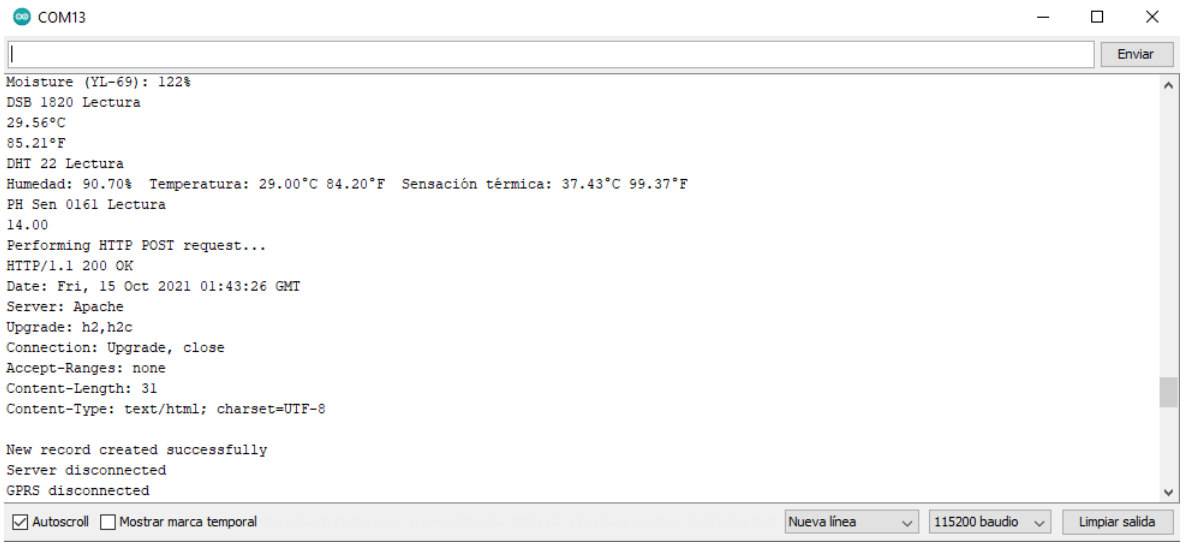
Nota: Captura tomada por Autores.

4.6. Envío de datos de los sensores vía LTE.

Al realizar las primeras pruebas de lectura de los datos y primera interconexión en modo automático, se logra acceder a la conexión 4G LTE, dando como resultado la adecuación de lectura de datos de los sensores, análisis y apropiación de los comandos AT e interconexión con el módem.

Figura 39

Captura de la Shell de Arduino™ con la prueba del envío de los datos vía LTE.



Nota: Captura tomada por Autores

Se adquieren resultados en torno a la toma de datos en tiempos programados: conexión y desconexión de la red 4G LTE por cada ciclo, toma de 10 muestras por variable y cálculo de promedios e interconexión con el enlace web que aloja el servidor.

4.6.1. Envío de mensaje de texto (SMS).

Tanto para la tecnología WiFi y 4G LTE, se implementa las configuraciones y comandos AT que permiten organizar el paquete de envío de datos correspondientes a los promedios y notificación a un número de teléfono de usuario mediante SMS. Tan pronto se cumple el ciclo de toma de 10 muestras por tecnología y cálculo de su promedio, se procede a iniciar el modo sleep del procesador del ESP32.

Figura 40

Captura de pantalla con la recepción de los mensajes de texto (SMS).

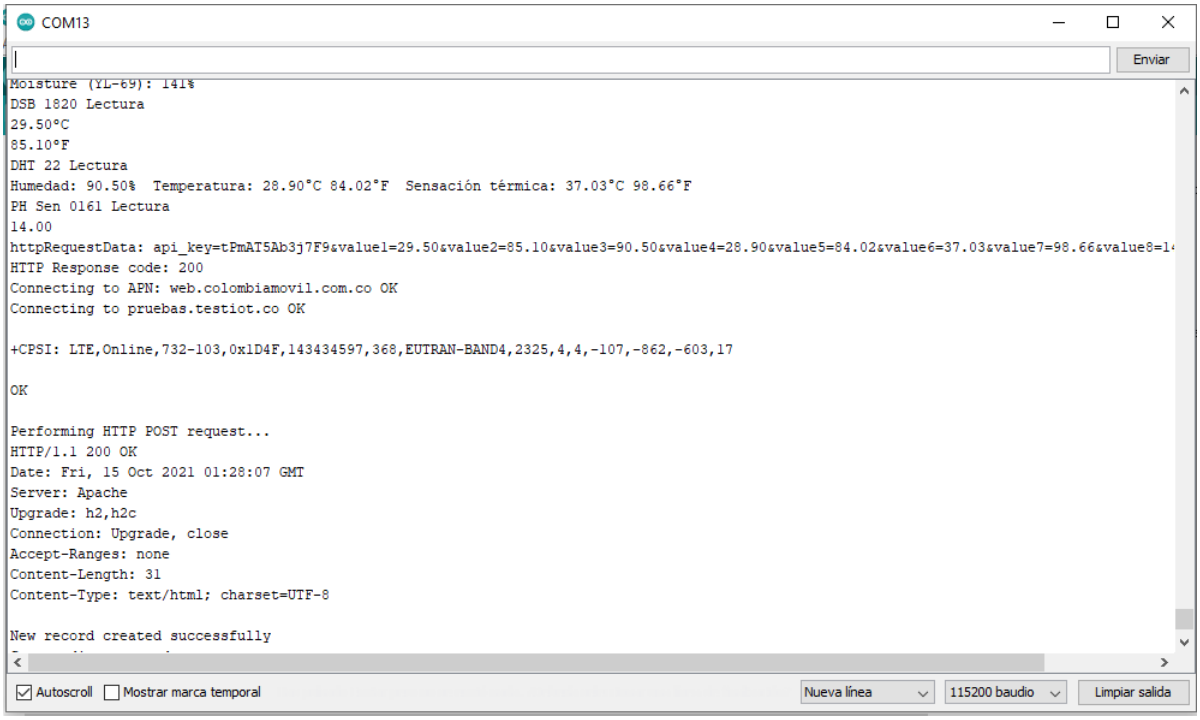
Nota: Captura tomada por Autores.

4.6.2. Módulo final

Este módulo comprende la adecuación de las dos tecnologías, en la cuál en un mismo ciclo toma 10 muestras enviadas por tecnología WiFi y 10 muestras enviadas por tecnología 4G LTE (ambos con sus respectivos promedios), notificación por mensaje SMS e interfaz de usuario.

Figura 41

Captura de la Shell de Arduino™ con la prueba del envío de los datos vía WiFi y LTE



Nota: Captura tomada por Autores.

Luego de tener las respectivas pruebas funcionando, se procede a realizar la PCB y a implementar el módulo junto con su sensórica. Se buscó que la PCB fuera compacta pero a su vez pudiera contener en ella todas las conexiones de los sensores al módulo central ESP32 *LilyGO T-PCIE*. Para mayor información acerca de la PCB del módulo final, ver la **Sección 4.8**. El código final se encuentra en el **Apéndice B** y también en el repositorio de GitHub™ en la dirección web https://github.com/1995Dayner/M-dulo-IOT-Wifi-4GLTE.

Figura 42

Fotografía del módulo final implementado.

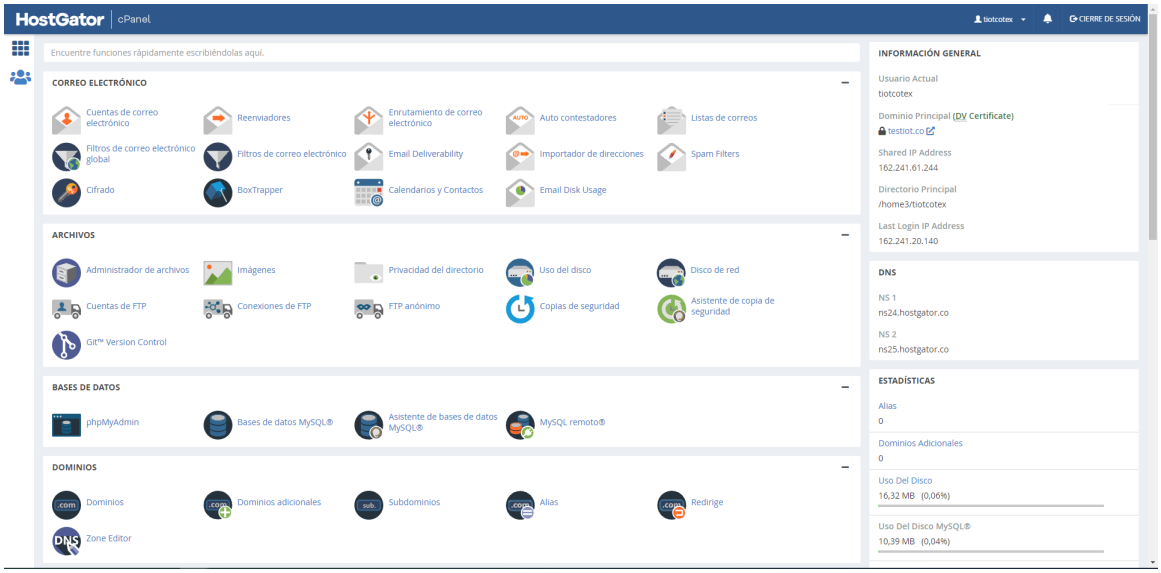
Nota: Foto tomada por Autores.

4.7. Interfaz gráfica.

Para la muestra de los datos arrojados por el módulo, se requiere una interfaz que permita ver los valores almacenados en tiempo real, con gráficas sencillas pero a la vez completas mostrando el comportamiento de cada variable medida. Para este caso en particular, los datos se suben a una base de datos en MySQL™ en un servidor web. En este caso en particular para este Trabajo de Grado, se está usando un alojamiento web u Hosting llamado Hostgator™, el cual ofrece una interfaz intuitiva y completa para la configuración de los parámetros de la página web mediante la plataforma cPanel™.

Figura 43

Pantallazo de la interfaz cPanel™ del hosting Hostgator™.



Nota: Pantallazo tomado por Autores.

4.7.1. Base de datos.

Como se mencionó con anterioridad, se hace uso de base de datos en MySQL™. Una base de datos debe estar compuesta por lo siguiente:

- Host de conexión a Base de Datos.
- Nombre de Base de Datos.
- Usuario asociado a la Base de Datos
- Contraseña

Lo anterior se realiza mediante la interfaz cPanel™ en la opción llamada “Asistente de bases de datos MySQL™”. Ahí se introduce el nombre de la base de datos, el nombre de usuario, contraseña y por último se definen los permisos de administración. La administración total de las bases de datos se realiza mediante la interfaz phpMyAdmin™ (R. . Santos, 2019c).

4.7.1.1. Establecimiento de las variables en la Base de Datos.

Al crear la base de datos, mediante phpMyAdmin™ se crea la tabla de datos mediante código SQL. Teniendo en cuenta que este módulo entrega 9 variables, debe existir en la base de datos 9 casillas para almacenar los datos de cada variable. Mediante la opción SQL en phpMyAdmin™, se introduce la siguiente línea de código:

Figura 44

Código de definición de las variables para la tabla en SQL.

```
CREATE TABLE Sensor (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    value1 VARCHAR(10),
    value2 VARCHAR(10),
    value3 VARCHAR(10),
    value4 VARCHAR(10),
    value5 VARCHAR(10),
    value6 VARCHAR(10),
    value7 VARCHAR(10),
    value8 VARCHAR(10),
    value9 VARCHAR(10),
    reading_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
    UPDATE CURRENT_TIMESTAMP
)
```

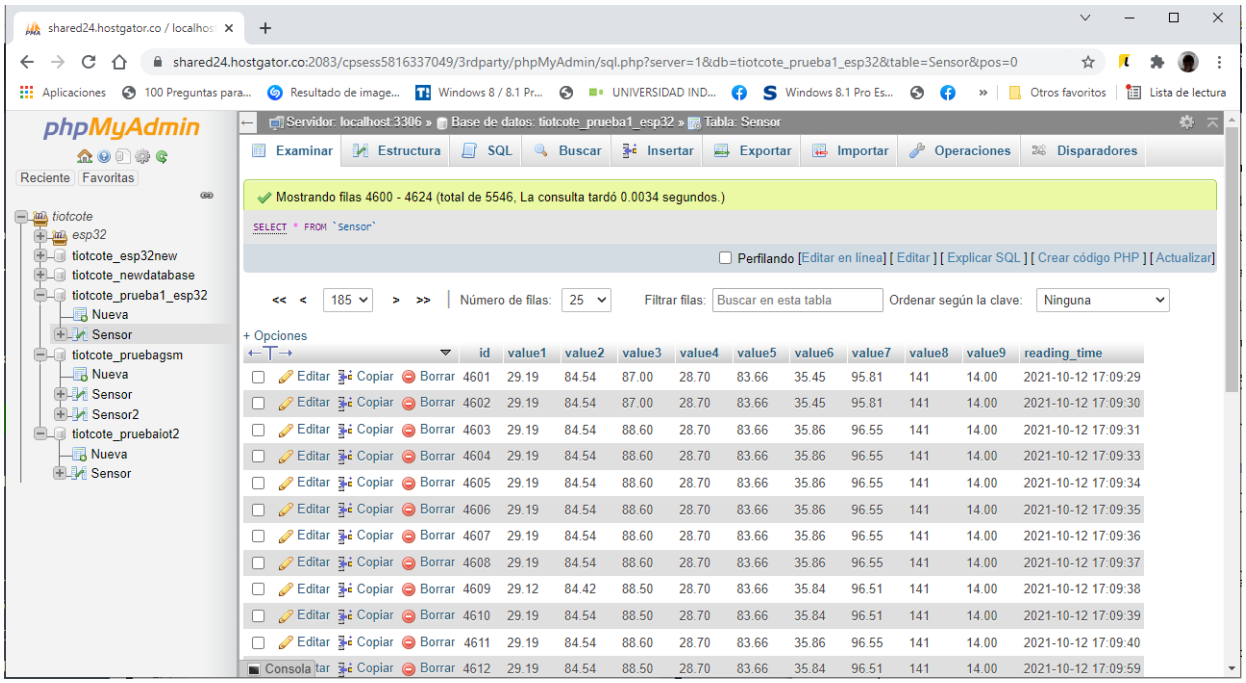
Nota: Basado en (R. . Santos, 2019c).

4.7.1.2. Muestra de los datos almacenados en la Base de Datos.

Mediante phpMyAdmin™ se pueden observar todos los datos almacenados en cada una de las bases de datos creadas para tal fin en la plataforma. En este caso en el Trabajo de Grado, la base de datos en MySQL™ hacen la función de “Cloud”; y a diferencia de plataformas como Sigfox u similares, acá se tiene mayor control del proceso puesto que al preparar la interfaz de usuario se parte de los datos almacenados en la base de datos y no a estar amarrado a una credencial o conexión a una red en específico. Los datos se almacenan y se acceden a ellos mediante redes comunes WiFi y/o 4G LTE sin ningún problema, representando una mayor flexibilidad y cobertura.

Figura 45

Pantallazo de la interfaz phpMyAdmin™ mostrando los datos almacenados en la base de datos, en la tabla `Sensor`.



Nota: Pantallazo tomado por Autores.

4.7.2. Enlace entre Arduino™ y la base de datos.

El puente entre la interfaz de Arduino™ y la base de datos se realiza mediante un código en php llamado `post-data.php`. Este código se encarga de obtener los datos entregados por la ESP32 mediante el modo `POST`, para luego almacenarlos en la base de datos.

Para comenzar, se inicializa primero a la base de datos y se obtienen los valores que entrega la ESP32 empaquetados. Uno de los detalles que debe tener en común es el `api key`, el cual sirve como identificador y sólo funcionará el código si la ESP32 está programada con la misma `api key` introducida en el `post-data.php`. Luego del identificador `api key`, se obtienen los valores de las 9 variables que entrega la ESP32.

Figura 46

Código de obtención de variables entregadas por la ESP32 al código `post-data.php`.

```
$servername = "Nombre_servidor";
$dbname = "Nombre_base_de_datos";
$username = "Usuario_base_de_datos";
$password = "Contraseña_base_de_datos";

$api_key_value = "tPmAT5Ab3j7F9";

$api_key = $value1 = $value2 = $value3 = $value4 = $value5
= $value6 = $value7 = $value8 = $value9 = "";

)
```

Nota: Basado en (R. . Santos, 2019c).

La otra parte importante de este código es la comparación del `api key` y de la comparación de las credenciales de acceso brindadas al inicio del código `post-data.php`. Si todo es correcto, va a generar un nuevo valor en la base de datos. Si hay error en la conexión en la base de datos por credenciales erradas, `api key` no coincidente con el que está empaquetado en los datos entregados por la ESP32 y/o falla en la conexión al servidor; el código no va a almacenar ningún dato.

Figura 47

Código de conexión y envío de datos a la base de datos establecido en `post-data.php`.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = test_input($_POST["api_key"]);
    if($api_key == $api_key_value) {
        $value1 = test_input($_POST["value1"]);
        $value2 = test_input($_POST["value2"]);

        $conn = new mysqli($servername, $username,
        $password, $dbname);

        if ($conn->connect_error) {
            die("Connection failed: " .
            $conn->connect_error);
        }

        $sql = "INSERT INTO Sensor (value1, value2)
        VALUES ('" . $value1 . "', '" . $value2 . "')";

```

```
        if ($conn->query($sql) === TRUE) {
            echo "New record created successfully";
        }
        else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }
        $conn->close();
    }
    else {
        echo "Wrong API Key provided.";
    }
}
else {
    echo "No data posted with HTTP POST.";
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

Nota: Basado en (R. . Santos, 2019c), y se ejemplifica para sólamente 2 variables (ya que el código del módulo final usa 9 variables en total).

4.7.3. Llamado a la base de datos y gráficas.

A partir de los datos almacenados en la base de datos usando el código `post-data.php`, para hacer visible la información al usuario final del comportamiento de los sensores se realiza mediante otro código en `php` llamado `esp-chart.php`.

En el código `esp-chart.php` comenzamos en primer lugar incializando la base de datos con las credenciales y también se hace llamado a las variables almacenadas. Al igual que el código `post-data.php`, se pregunta primero si las credenciales asignadas son las correctas o no para el acceso a la base de datos en específico. La diferencia con respecto al código `esp-chart.php` es que en este caso no se hace llamado al `api key` puesto que se está trabajando con información ya almacenada en la base de datos. La variable `readings_time` tiene por finalidad almacenar desde

la base de datos el tiempo en el cual se almacenó la muestra (fecha y hora).

Figura 48

Código de lectura de variables almacenadas en la base de datos mediante `esp-chart.php`.

```php
$servername = "localhost";

$dbname = "tiotcote_prueba1_esp32";
$username = "tiotcote_prueba1_user";
$password = "Jhonc-1994";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, value1, value2, reading_time FROM Sensor order by reading_time desc limit 40";

$result = $conn->query($sql);

while ($data = $result->fetch_assoc()){
    $sensor_data[] = $data;
}

$readings_time = array_column($sensor_data, 'reading_time');

```

Nota: Basado en (R. . Santos, 2019c).

Como php nos permite trabajar con diferentes lenguajes referentes a páginas web, en este caso se trabaja dentro del `esp-chart.php` tanto Javascript (para el asunto de animaciones dentro de la página web), HTML (para la estructura de la página web a nivel de información) y CSS (para temas de forma y color).

En este caso del `esp-chart.php`, se comienza inicializando las variables obtenidas en la base de datos codificando su valor en variable JSON (Javascript Object Notation). El proceso se realiza porque las gráficas se basarán en una plataforma llamada Highcharts™.

Figura 49

Código de para la codificación de los valores obtenidos en la base de datos a JSON.

```
$value1 =
json_encode(array_reverse(array_column($sensor_data,
'value1')), JSON_NUMERIC_CHECK);
$value2 =
json_encode(array_reverse(array_column($sensor_data,
'value2')), JSON_NUMERIC_CHECK);

$reading_time = json_encode(array_reverse($readings_time),
JSON_NUMERIC_CHECK);

```

Nota: Basado en (R. . Santos, 2019c).

Ya teniendo cuadrada las nuevas variables a trabajar, se realiza la estructura de la página web mediante HTML dentro del código `esp-chart.php`.

Figura 50

Código de la estructura de la página web.

```
<!DOCTYPE html>
<html>
<meta name="viewport" content="width=device-width,
initial-scale=1">
  <script
  src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      min-width: 310px;
      max-width: 1280px;
      height: 500px;
      margin: 0 auto;
    }
    h2 {
      font-family: Arial;
      font-size: 2.5rem;
      text-align: center;
    }
  </style>
  <body>
    <h2>ESP Weather Station</h2>
    <div id="chart-temperature-dsb1820-C"
    class="container"></div>
    <div id="chart-temperature-dsb1820-F"
    class="container"></div>
```

```
22 <script>
```

Nota: Basado en (R. . Santos, 2019c).

Para la realización de la gráfica, primeramente se convierten las variables a Javascript y luego se envocan en otro código que representa la gráfica como tal.

Figura 51

Código de la gráfica de los valores almacenados en la base de datos.

```
var value1 = <?php echo $value1; ?>;
var value2 = <?php echo $value2; ?>;

var reading_time = <?php echo $reading_time; ?>;

var chartTC_DSB1820 = new Highcharts.Chart({
  chart:{ renderTo : 'chart-temperature-dsb1820-C' },
  title: { text: 'DSB1820 Temperatura (°C)' },
  series: [{
    showInLegend: false,
    data: value1
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#059e8a' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Temperatura (Celsius)' }
  },
  credits: { enabled: false }
});
```

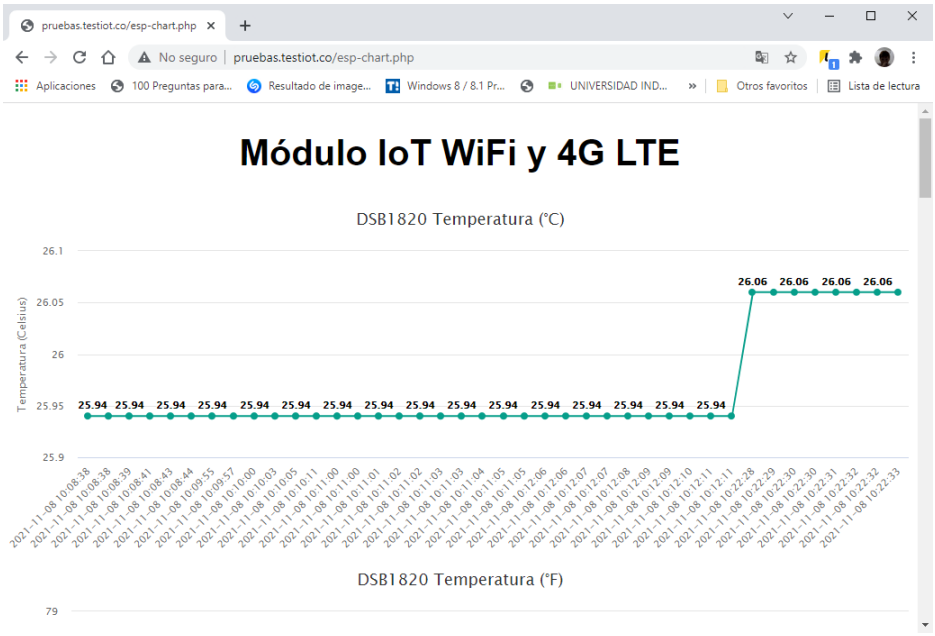
Nota: Basado en (R. . Santos, 2019c).

A partir de los códigos anteriomente mencionados, se logra graficar los valores arrojados por los sensores en tiempo real, con una interfaz amigable para el usuario final (mostrando no solo el valor medido sino con la fecha y hora de la medición). La dirección web para la visualización de

los sensores es https://pruebas.testiot.co/esp-chart.php.

Figura 52

Captura de pantalla de la interfaz de usuario final.



Nota: Captura tomada por Autores.

4.8. Diseño de PCB.

El diseño de circuitos impresos convierte en realidad el diseño en protoboard de los circuitos electrónicos, mediante un software de diseño que combina la colocación y el enrutamiento de componentes para definir la conectividad eléctrica en un circuito impreso manufacturado.

Para llevar a cabo el proceso del diseño del PCB, primero se realiza el montaje en protoboard, con el objetivo de probar el funcionamiento del circuito conectando el microcontrolador ESP32 con los sensores de humedad, temperatura y ph. Luego mediante el software se crea un esquemático y se agregan los componentes o dispositivos a utilizar, que en este caso son el sensor DHT22, el DS18B20, el YL69, el PH SEN6101, el ESP32, dos resistencias de 4.7 kΩ y un capacitor de 10 μF.

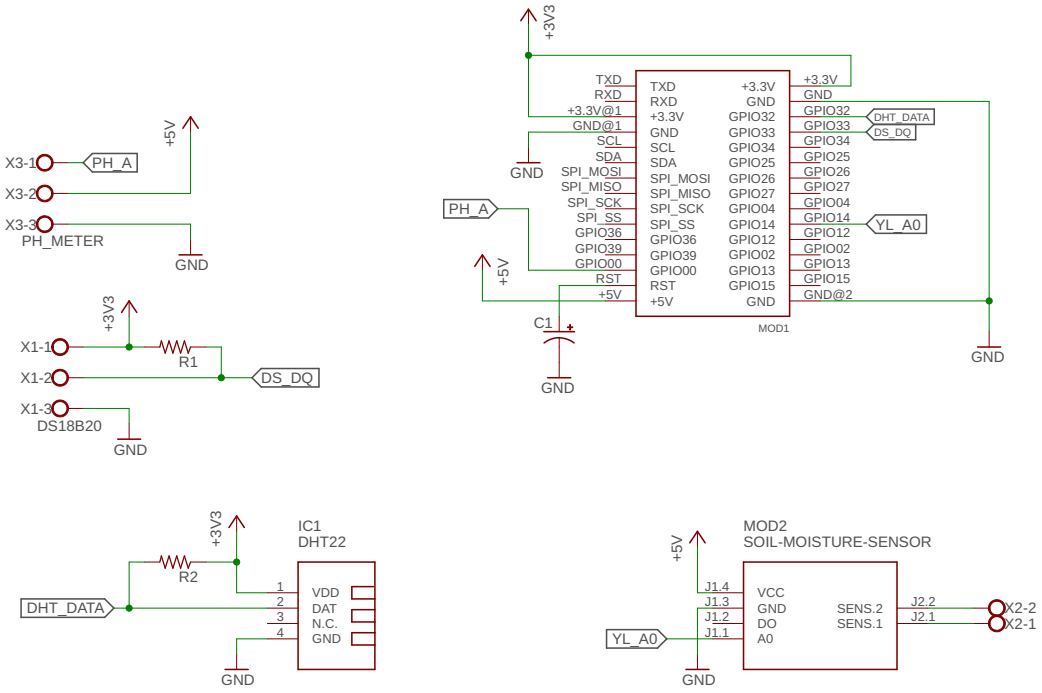
En un esquemático los componentes deben estar bien organizados, alineados e identificados de manera que sea legible y en la medida de lo posible, colocar los componentes de manera que facilite la conexión y enrutado en el diseño final del PCB.

El sensor DHT22 posee tres pines de conexión, positivo, negativo y salida de la señal. El pin positivo va conectado a la tensión de alimentación 3.3 V, la señal de salida se conecta al GPIO del microcontrolador ESP32 y el negativo al tierra GND. El sensor de temperatura DS18B20 también opera con una tensión de alimentación de 3.3 V y su señal de salida se conecta al GPIO de la ESP32 para poder tomar lectura de los datos. Se usa una resistencia de 4.7 kΩ a la señal de salida de los sensores, a modo de protección eléctrica.

El sensor de PH SEN0161 es de tres pines, positivo que va conectado a la tensión de alimentación 5 V, pin de tierra o GND y el pin de señal de salida; que entrega los datos tomados al microcontrolador por medio de la conexión con el GPIO. El sensor YL-69, tiene la capacidad de medir la humedad del suelo aplicando una pequeña tensión entre los terminales del módulo y hace pasar una corriente que depende básicamente de la resistencia que se genera en el suelo, también se le conoce como sensor FC-28 y posee tres pines de conexión, positivo conectado a la alimentación de 5 V, negativo a la tierra del circuito y el pin de señal de salida A_O, que se conecta al GPIO del microcontrolador con el fin de entregar los datos tomados en la medición.

Figura 53

Plano esquemático del módulo implementado.



Nota: Diagrama creado por Autores.

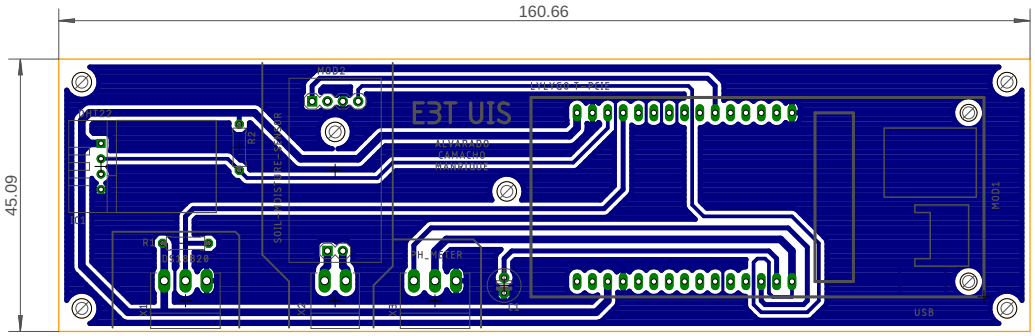
La imagen anterior, corresponde al esquemático del módulo final. Se puede observar las conexiones pertinentes entre cada dispositivo y el microcontrolador. Es importante tener en cuenta que las conexiones no se entrecrucen, orientar los planos de alimentación y tierra, definir ancho de las pistas, acomodar los componentes de la mejor forma posible y revisar que los pines estén de forma correcta de acuerdo al diseño.

El software de diseño facilita el desarrollo de la PCB, ya que indica con lineas punteadas la forma en que se deben realizar las conexiones entre los pines. Una vez que se unen todos los caminos, teniendo en cuenta que no se crucen dos líneas, se verifica el ancho de los pines, ancho de las pistas, que no hayan componentes desconectados y que los puertos de tensión o alimentación estén independientes. No obstante, el programa realiza un chequeo para confirmar que no haya

errores en el diseño antes de generar el archivo svg o pdf, el cuál es enviado a una empresa capacitada en impresión de placas para proceder al montaje físico y final del módulo IoT.

Figura 54

Plano con la información de la PCB y sus respectivas medidas.



Nota: Diagrama creado por Autores.

La PCB final mide 160.66 mm × 45.09 mm.

5. Observaciones y Conclusiones.

A partir del trabajo anteriomente expuesto, se deja a modo de observaciones y conclusiones aspectos fundamentales a tener en cuenta para futuros Trabajos de Grado en áreas afines a las tratadas en este documento.

5.1. Observaciones.

- En el módulo se usaron sensores de bajo costo, pero esto trae en contra que la sensórica no presente la precisión deseada. Para tener un módulo más robusto en cuanto a la medición, se sugiere usar sensores de mayor precisión. Esto representaría un costo adicional al módulo final pero a favor se tendría mediciones más fiables.
- El módulo SIM7600SA, según la librería TinyGSM(TinyGSM, 2021), no es compatible con certificados SSL (fundamental para protocolos de seguridad https). Para futuras implementaciones se recomienda aumentar la seguridad de conexión usando otras técnicas para así dar mayor integralidad a la transmisión y almacenamiento de los datos en la base de datos.
- Se propone cómo una acción de mejora del proyecto para futuras iniciativas, mejorar el diseño del servidor, configuraciones de seguridad web y login de usuario para el acceso y manejo de datos en torno a la articulación de nuestro servidor y/o diseño web dónde mostramos los resultados. Así como la articulación de un mayor número de datos, histórico de valores y/o sensores; lo cuál requeriría un servidor y diseño web más robusto.
- Se sugiere realizar para futuros Trabajos de Grado, un análisis previo de consumo de corriente del módulo final implementado, para así tener mayor información del comportamiento energético del dispositivo y así tomar decisiones con respecto a la fuente de alimentación. Dadas las condiciones adversas del campo colombiano, una opción

interesante podría ser la alimentación mediante panel solar.

5.2. Conclusiones.

- Este modulo tiene por ventaja lograr una comunicación a nivel de red móvil a cualquiera de las tecnologías presentes hasta la fecha con sólo cambiar un parámetro en el código de programación de Arduino™. La ventaja de poder funcionar en red 4G LTE es una mejora sustancial en la respuesta de la conexión entre la ESP32 y el servidor web con respecto a módulos IoT con conexión a red GSM.
- El módulo diseñado y sus formas de comunicación para el envío de datos es totalmente transversal y posibilita la implementación de mejoras a futuro o mediano plazo en torno a la instrumentación usada para la adquisición de datos de las variables y la respectiva calibración de su sensórica.
- El módulo IoT planteado permite una mayor flexibilidad a la hora de configurar los parámetros directamente involucrados en la transmisión y muestra de los datos, puesto que estos se almacenan en un servidor web configurado especialmente para este módulo. En el mercado hay soluciones IoT con redes propietarias de diferentes empresas (por ejemplo Sigfox y LoRa), pero estas no tienen la misma flexibilidad que usar un servidor web configurado de manera específica para todos los requerimientos del módulo. Además, al enviar los datos a un servidor web permite usar redes WiFi y/o móviles comunes, dando mayor flexibilidad para la transmisión de los datos.

Referencias Bibliográficas.

Agnihotri, N. . (2021, 14 de septiembre). *AT Commands, GSM AT command set*. Consultado el 6 de octubre de 2020, desde https://www.engineersgarage.com/at-commands-gsm-at-command-set/

Aliexpress. (2021). *17.92€ 15% de DESCUENTO|Chip de ESP32 WROVER B NANO tarjeta SIM serie hardware configurable SIMCOM SIM7000G SIM7600A SIM7600E SIM7070G SIM868 LET CAT4 módulo| | - AliExpress*. Consultado el 6 de octubre de 2021, desde https://es.aliexpress.com/item/10000334880659.html?af=3866332&aff_fcid=789758fb1e354e9b9323f40589bc79ed-1633565539635-03936-_pJ11Pyl&aff_fsk=_pJ11Pyl&aff_platform=api-new-link-generate&sk=_pJ11Pyl&aff_trace_key=789758fb1e354e9b9323f40589bc79ed-1633565539635-03936-_pJ11Pyl&terminal_id=ed9cdc1819f44db698651a3062da673c

Alonso, P. . (2017, 29 de noviembre). *¿Qué es LTE Advanced?* Consultado el 6 de octubre de 2020, desde https://www.teldat.com/blog/es/lte-advanced-3gpp-release-911-norm-comp-8x8-mimo-antennas/

Aosong Electronics Co.,Ltd. (2021). *DHT22 Datasheet*. Consultado el 12 de octubre de 2021, desde https://pdf1.alldatasheet.com/datasheet-pdf/view/1132459/ETC2/DHT22.html

Arca Electrónica. (2021). *Sensor De Temperatura Y Humedad Dht22 Am2302 + Pcb Con Cable*. Consultado el 8 de octubre de 2021, desde https://www.arcaelectronica.com/products/sensor-de-temperatura-y-humedad-dht22-am2302-pcb-con-cable

Brunete, San Segundo & Herrero. (2020, 28 de julio). *1.2 Concepto de sistema | Introducción a la Automatización Industrial*. Consultado el 13 de octubre de 2021, desde https://bookdown.org/alberto_brunete/intro_automatica/concepto-de-sistema.html

Canna Research. (2021). *Influencia de la temperatura ambiental en las plantas | CANNA España*. Consultado el 6 de octubre de 2020, desde https://www.canna.es/influencia_temperatura_ambiental_en_las_plantas

Carmenate, J. . G. . (2021, 18 de febrero). *ESP32 Wifi + Bluetooth en un solo lugar*. Consultado el 12 de octubre de 2021, desde https://programarfacil.com/esp8266/esp32/

Cubides Tórres, D. Y. & Manrique Suarez, S. F. (2020). *Diseño y construcción de un módulo electrónico de bajo costo para el sensado de variables asociadas con la calidad del agua y del suelo* (Trabajo de Grado (Ingeniero Electrónico)). Universidad Industrial de Santander. Bucaramanga - Colombia.

DANE. (2016a). 3er Censo Nacional Agropecuario. Hay campo para todos.

DANE. (2016b, 1 de junio). *Encuesta Nacional Agropecuaria (ENA) 2015*. Consultado el 10 de diciembre de 2020, desde https://www.dane.gov.co/files/investigaciones/agropecuario/enda/ena/2015/boletin_ena_2015.pdf

DFROBOT. (2021). *Gravity: Analog pH Sensor / Meter Kit For Arduino SEN0161*. Consultado el 15 de octubre de 2021, desde https://www.dfrobot.com/product-1025.html

educ8s.tv. (2018, 22 de febrero). *ESP32 Deep Sleep Tutorial*. Consultado el 12 de octubre de 2021, desde http://educ8s.tv/esp32-deep-sleep-tutorial/

ESP8266 Arduino Core. (2017). *Librería ESP8266WiFi — documentación de ESP8266 Arduino Core - 2.4.0*. Consultado el 13 de octubre de 2021, desde https://esp8266-arduino-spanish.readthedocs.io/es/latest/esp8266wifi/readme.html

Espressif Systems. (2021). *ESP 32 Datasheet*. Consultado el 9 de octubre de 2021, desde https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Fitchard, K. . (2015, 17 de agosto). *¿Qué tan "avanzada" es la red LTE-Advanced?* Consultado el 6 de octubre de 2020, desde https://www.telesemana.com/blog/2015/07/01/que-tan-avanzada-es-la-red-lte-advanced/

Geek Factory. (2021, 9 de octubre). *DS18B20 con Arduino: Sensor de temperatura digital*. Consultado el 9 de octubre de 2021, desde https://www.geekfactory.mx/tutoriales-arduino/ds18b20-con-arduino-sensor-de-temperatura-digital/

GSA. (2021, marzo). *LTE to 5G: March 2021 - Global update*. Consultado el 6 de octubre de 2020, desde https://gsacom.com/paper/lte-to-5g-march-2021-global-update/?utm=reports4g

Huawei. (2019). *¿Sabes acerca del concepto VoLTE y lo que significa? | HUAWEI Soporte México*. Consultado el 6 de octubre de 2020, desde https://consumer.huawei.com/mx/support/article-list/article-detail/es-us00823385/

Ioland. (2020, 29 de diciembre). *LA IMPORTANCIA DEL CONTROL DE LA TEMPERATURA DEL SUELO*. Consultado el 6 de octubre de 2020, desde https://ioland.es/6346-2/

IoT Applications in Agriculture. (2018, 30 de junio). Consultado el 10 de diciembre de 2020, desde https://www.iotforall.com/iot-applications-in-agriculture

Jones, D. . & Beaver, K. . (2021, 28 de abril). *LTE (Long-Term Evolution)*. Consultado el 6 de octubre de 2020, desde https://searchmobilecomputing.techtarget.com/definition/Long-Term-Evolution-LTE

León Merchán, K. L., Calderón, S. & Andrés, O. (2020). *Sistema de monitoreo de variables ambientales en cultivos de papa mediante IOT y energía solar fotovoltaica* (Trabajo

de Grado (Ingeniero Electrónico y de Telecomunicaciones)). Universidad Católica de Colombia. Bogotá - Colombia.

LilyGO. (2021a, 28 de septiembre). *GitHub - Xinyuan-LilyGO/LilyGo-T-Call-SIM800*. https://github.com/Xinyuan-LilyGO/LilyGo-T-Call-SIM800

LilyGO. (2021b, 19 de agosto). *GitHub - Xinyuan-LilyGO/LilyGo-T-PCIE*. Consultado el 13 de octubre de 2021, desde https://github.com/Xinyuan-LilyGO/LilyGo-T-PCIE

Lozano, R. . (2020, 7 de febrero). *¿Como mandar SMS desde arduino y SIM800L?* Consultado el 12 de octubre de 2021, desde https://www.taloselectronics.com/blogs/tutoriales/como-mandar-sms-desde-arduino-y-sim800l

Martínez C, S. (2019, 10 de septiembre). *Importancia de la humedad en los cultivos: MOISTURE (NDMI)*. Consultado el 6 de octubre de 2020, desde https://www.mapsens.com/importancia-de-la-humedad-en-los-cultivos-moisture-ndmi/

Maxim Integrated. (2019). *DSB18B20 Datasheet*. Consultado el 12 de octubre de 2021, desde https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

MinTIC. (2019, 20 de diciembre). *Comunicado/ Resultados de la Subasta del Espectro Radioeléctrico*. Consultado el 6 de octubre de 2020, desde https://mintic.gov.co/portal/inicio/Sala-de-prensa/Noticias/124713:Comunicado-Resultados-de-la-Subasta-del-Espectro-Radioelectrico

MinTIC. (2020, 24 de junio). *Colombia inicia transición de redes 2G y 3G hacia 4G*. Consultado el 6 de octubre de 2020, desde https://mintic.gov.co/portal/inicio/Sala-de-Prensa/Noticias/145550:Colombia-inicia-transicion-de-redes-2G-y-3G-hacia-4G

ModernAg. (2017, 1 de noviembre). *Agricultura de Precisión, ¿de qué se trata?* Consultado el 6 de octubre de 2021, desde https://www.agmoderna.com.ar/tecnologia-en-el-campo/agricultura-de-precision-de-que-se-trata/

Montoya, S. . (2018, 9 de marzo). *Cómo medir pH desde tu laptop en tiempo real con Arduino?* Consultado el 12 de octubre de 2021, desde https://gidahatari.com/ih-es/como-medir-ph-desde-tu-laptop-en-tiempo-real-con-arduino

Morales, E. . P. . N. . (2018, 4 de enero). *Aplicaciones de IoT en agricultura*. Consultado el 6 de octubre de 2021, desde https://agriculturers.com/aplicaciones-de-iot-en-agricultura/#:%7E:text=En%20la%20agricultura%20inteligente%20basada,del%20campo%20desde%20cualquier%20lugar.

Moviltronics SAS. (2021, 30 de junio). *Sensor humedad de Suelo YL69*. Consultado el 8 de octubre de 2021, desde https://moviltronics.com/tienda/sensor-yl69/

Murky Robot. (2021). *Sonoff Sensor - DS18B20 - Temperatura*. Consultado el 8 de octubre de 2021, desde https://www.murkyrobot.com/catalogo/ds18b20-sensor-de-temperatura-sumergible

Nutricontrol. (2020, 21 de enero). *La importancia del control de pH en los cultivos*. Consultado el 6 de octubre de 2020, desde https://nutricontrol.com/es/la-importancia-del-control-de-ph-en-los-cultivos/

Ojeda, D. (2018, 17 de febrero). Internet de las cosas avanza en el agro colombiano [El Espectador]. Consultado el 10 de diciembre de 2020, desde https://www.elespectador.com/noticias/economia/internet-de-las-cosas-avanza-en-el-agro-colombiano/

Oracle. (2021). *What is the Internet of Things (IoT)?* https://www.oracle.com/internet-of-things/what-is-iot/

Petrick, M. . (2019, 12 de marzo). *ESP32: integrated the YL-69 for moisture-metering, also fried my first BME280 | solutions; not code*. Consultado el 12 de octubre de 2021, desde https: //marcelpetrick.bplaced.net/wp_solutionsnotcode/?p=1229

Rodríguez Rodríguez, R. (2005). *Diseño de un sistema multimedia público utilizando Telefonía Móvil GSM* (Trabajo de Grado (Ingeniero Electricista)). Universidad Central de Venezuela. Caracas - Venezuela.

Santos, R. (2019a, 16 de julio). *ESP32 DS18B20 Temperature Sensor with Arduino IDE (Single, Multiple, Web Server)*. Consultado el 12 de octubre de 2021, desde https : / / randomnerdtutorials.com/esp32-ds18b20-temperature-arduino-ide/

Santos, R. . (2019b, 9 de septiembre). *ESP32 SIM800L: Publish Data to Cloud without Wi-Fi*. Consultado el 13 de octubre de 2021, desde https : / / randomnerdtutorials . com / esp32 - sim800l-publish-data-to-cloud/

Santos, R. . (2019c, 26 de diciembre). *Visualize ESP32/ESP8266 Sensor Readings from Anywhere in the World*. Consultado el 12 de octubre de 2021, desde https://randomnerdtutorials.com/ visualize-esp32-esp8266-sensor-readings-from-anywhere/

Santos, S. (2019, 4 de mayo). *ESP32 with DHT11/DHT22 Temperature and Humidity Sensor using Arduino IDE*. Consultado el 12 de octubre de 2021, desde https://randomnerdtutorials.com/ esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/

Significados. (2019, 7 de agosto). *Significado de Wifi*. Consultado el 13 de octubre de 2021, desde https://www.significados.com/wifi/

SIMCOM. (2021). *SIM7600X Module 4G Wireless Solutions | SIMCom Wireless Solutions Co.,Ltd.* Consultado el 13 de octubre de 2021, desde https://www.simcom.com/product/SIM7600X. html

SMS Tutorial: Introduction to AT Commands, Basic Commands and Extended Commands. (2021). Consultado el 6 de octubre de 2020, desde https://www.developershome.com/sms/atCommandsIntro.asp

Soto, J. P. T., Suárez, J. d. l. S. S., Rodríguez, A. B. & Cainaba, G. O. R. (2019). Internet de las cosas aplicado a la agricultura: estado actual. *Lámpsakos*, (22), 86-105.

Thales. (2019). *Long Term Evolution (LTE) - 4G LTE explained.* Consultado el 6 de octubre de 2020, desde https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/lte

TinyGSM. (2021, 19 de mayo). *GitHub - vshymanskyy/TinyGSM: A small Arduino library for GSM modules, that just works*. Consultado el 19 de mayo de 2021, desde https://github.com/vshymanskyy/TinyGSM

Triana Useche, J. C., Rodriguez Leguizamo, R. E. et al., (2018). Prototipo de solución Iot con tecnología Lora en monitoreo de cultivos agrícolas.

Villalpando. (2021). *234.- Wemos D1 R32 ESP32. Deep Sleep. Sueño profundo. Suspensión.* Consultado el 10 de octubre de 2021, desde http://kio4.com/arduino/9234f.htm

Wannstrom. (2013). *Carrier Aggregation explained*. Consultado el 6 de octubre de 2020, desde https://www.3gpp.org/technologies/keywords-acronyms/101-carrier-aggregation-explained

Apéndices.

Apéndice A. Comandos AT.

Los comandos AT tienen por finalidad controlar el modem de comunicaciones. Mediante un alfabeto establecido, se puede dictaminar instrucciones específicas con solo unas palabras. Es usado para controlar módems Dial-UP, modems GSM/GPRS y teléfonos móviles. («SMS Tutorial: Introduction to AT Commands, Basic Commands and Extended Commands», 2021). Las tareas básicas en las cuales se usan los comandos AT son las siguientes:

- Obtener información básica del teléfono móvil o módem móvil. Por ejemplo, nombre del fabricante (AT+CGMI), número del modelo (AT+CGMM), número de IMEI (AT+CGSN) y versión del software (AT+CGMR).
- Obtener información básica acerca del suscriptor. Por ejemplo el IMSI (International Mobile Suscriber Identity)(AT+CIMI).
- Conocer el estado actual del teléfono móvil o del módem, como por ejemplo la intensidad de señal de radio (AT+CSQ).
- Saber acerca del tipo de red al cual está conectado el teléfono móvil o el módem (AT+CPSI).

Los comandos AT vienen en 4 tipos según la finalidad del uso (Agnihotri, 2021).

1. **Comandos de Test**, usados para saber si un comando es soportado por el módem o no.

   Sintaxis: AT+<Nombre de Comando>=?

   Ejemplo: ATD=?

2. **Comandos de Lectura**, usados para obtener ajustes de un módem o teléfono celular para una operación.

Sintaxis: AT+<Nombre de comando>?.

Ejemplo: AT+CBC?.

3. **Comandos de ajuste**, usados para modificar ajustes de un módem o teléfono celular para una operación.

   Sintaxis: AT+<Nombre del comando>=valor1, valor2, ... , valorN

   Algunos valores en los comandos de ajuste pueden ser opcionales.

   Ejemplo: AT+CMSS=1,"+ 9876543210", 120

4. **Comandos de Ejecución:** usados para realizar una operación

   Sintaxis: AT+<Nombre del Comando>=parametro1, parametro2,..., parametroN

   Los comandos de lectura no sirven para obtener el valor del último parámetro asignado porque los parámetros asignados en los comandos de ejecución no se almacenan.

   Ejemplo: AT+CMSS=1,"+ 9876543210", 120

Apéndice A.1. Modos de operación de los Comandos AT.

Los módems tienen cuatro modos de operación y se comportan en forma diferente en cada uno de esos modos:

Modo Espera: Es el modo de operación por defecto cuando el módem se enciende por primera vez. El módem está listo y habilitado para aceptar comandos y continúa en este estado hasta que es configurado para pasar a otro.

Modo Comando: Luego que un comando AT válido es detectado, el módem entra en este estado. Los carácteres del comando son almacenados hasta que la tecla Enter es presionada, en este momento, el módem carga el comando. Estos comandos pueden tanto configurar el módem como hacer que el módem realice una acción determinada.

Modo Data: En este modo, el módem ha sido llamado por otro módem y retornará una señal de acarreo e intentará establecer la conexión. Una vez conectado, el módem está configurado para intercambiar data entre los módems. El dispositivo sale del Modo Data sólo si una de las siguientes cosas ocurre:

(a) La secuencia de salida "+++" es recibida.

(b) La señal de acarreo del otro módem se pierde.

(c) La señal DTR (Data Terminal Ready) se pierde, lo cual indica que el computador conectado a este módem perdió la conexión.

(d) Ha pasado un largo tiempo desde que la última vez que se recibió o envió data.

Modo Interactivo: El módem entra en este estado cuando, una vez conectado, recibe la señal "+". El módem puede recibir comandos AT mientras continúa aún conectado con otro módem; es utilizado usualmente para ajustar parámetros de la conexión (Rodríguez Rodríguez, 2005).

Tabla 7

Comandos AT más comunes.

Comando	Descripción
AT	Código de atención que procede a todos los comandos
ATA	Responde una llamada entrante
ATD	Marca un número telefónico
ATH	Desconexión de línea
ATL	Volumen de altavoz
ATZ	Reset del módem
AT & F	Inicializa parámetros por defecto del módem
AT + CSCS	Establece la configuración de carácteres
AT + CMGF	Selecciona el formato de los mensajes
AT + CMGL	Muestra todos los mensajes de texto
AT + CMGS	Envía un mensaje de texto
AT + CMGR	Lee un mensaje de texto de la SIM
AT + CPSI	Consulta tipo de red conectada
AT + CNMP	Selección red predeterminada
AT + CLIP	Activa la identificación de llamada
AT + CPBR	Lee los contactos del directorio telefónico
AT + CGDCONT	Define el protocolo PDP de GPRS
AT + CGREG	Registra GPRS en la red

Nota: Tomado de (Rodríguez Rodríguez, 2005).

Apéndice B. Código del módulo final (WiFi y LTE).

```
// Modulo lectura Sensores
// Código combinado sensores DHT22, YL 69, DSB 1820 y PH Sen 0161.

// Definiciones DHT
#include "DHT.h"

#define DHTPIN 32

// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

// Definiciones YL-69
#ifdef __cplusplus
extern "C" {
#endif
uint8_t temprature_sens_read();
#ifdef __cplusplus
}
#endif
uint8_t temprature_sens_read();

// use G23 of ESP32 to read data
const int YL69Pin = 14;

// Definiciones DSB 18B20
#include <OneWire.h>
#include <DallasTemperature.h>

// GPIO where the DS18B20 is connected to
const int oneWireBus = 33;

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

// Definiciones SEN 0161
const byte pHpin = 0;// Connect the sensor's Po output to analogue pin 0.
float Po;

```

```
46 // Variables acumuladoras y promedios
47 float temperatureC;
48 float temperatureF;
49 float h;
50 float t;
51 float f;
52 float hic;
53 float hif;
54 int convertedPercentage;
55
56 float acum_temp_amb_celcius = 0;
57 float acum_hum_Ambiente_Array = 0;
58 float acum_temp_suelo_celcius_Array = 0;
59 float acum_temp_suelo_celcius_Farenheit = 0;
60 float acum_temp_amb_farenheit = 0;
61 float acum_sensa_termi_celcius_Array = 0;
62 float acum_termi_faren_Array= 0;
63 float acum_humedad_suelo = 0;
64 float acum_ph_liqui = 0;
65
66 float acum_temp_amb_celcius_lte = 0;
67 float acum_hum_Ambiente_Array_lte = 0;
68 float acum_temp_suelo_celcius_Array_lte = 0;
69 float acum_temp_suelo_celcius_Farenheit_lte = 0;
70 float acum_temp_amb_farenheit_lte = 0;
71 float acum_sensa_termi_celcius_Array_lte = 0;
72 float acum_termi_faren_Array_lte = 0;
73 float acum_humedad_suelo_lte = 0;
74 float acum_ph_liqui_lte = 0;
75
76 float temp_amb_celcius_prom = 0;
77 float hum_Ambiente_Array_prom  = 0;
78 float temp_suelo_celcius_Array_prom = 0;
79 float temp_suelo_celcius_Farenheit_prom = 0;
80 float temp_amb_farenheit_prom = 0;
81 float sensa_termi_celcius_Array_prom = 0;
82 float sensa_termi_faren_Array_prom  = 0;
83 float humedad_suelo_prom  = 0;
84 float ph_liqui_prom  = 0;
85
86 float temp_amb_celcius_prom_lte = 0;
87 float hum_Ambiente_Array_prom_lte  = 0;
88 float temp_suelo_celcius_Array_prom_lte = 0;
89 float temp_suelo_celcius_Farenheit_prom_lte = 0;
90 float temp_amb_farenheit_prom_lte = 0;
91 float sensa_termi_celcius_Array_prom_lte = 0;
92 float sensa_termi_faren_Array_prom_lte  = 0;
93 float humedad_suelo_prom_lte  = 0;
94 float ph_liqui_prom_lte  = 0;
95
96 // Variables modo sleep
```

```
97 #define uS_TO_S_FACTOR 1000000  /* Conversion factor for
   micro seconds to seconds */
98
99 //Configuración de parámetro de tiempo de dormida del
   procesador 60= 1 segundo
100
101 #define TIME_TO_SLEEP  180        /* Time ESP32 will go to
   sleep (in seconds) */
102
103 RTC_DATA_ATTR int bootCount = 0;
104 int bootCount1 = 0;
105 int bootCount2 = 0;
106
107 //int bootCount1 = 0;
108
109 //Conexión Wiffi
110 #ifdef ESP32
111   #include <WiFi.h>
112   #include <HTTPClient.h>
113 #else
114   #include <ESP8266WiFi.h>
115   #include <ESP8266HTTPClient.h>
116   #include <WiFiClient.h>
117 #endif
118
119 //Credenciales de red wifi a la cuál se conectará el módulo
120 #include <Wire.h>
121 // Replace with your network credentials (Credenciales SSID
   y Password)
122 const char* ssid     = " ";// Nombre de red o ssid
123 const char* password = "} ";// Contraseña o credencial de
   acceso a la red
124
125 // REPLACE with your Domain name and URL path or IP address
   with path
126 // const char* serverName =
   "http://testiot.atwebpages.com/post-data.php";
127 const char* serverName =
   "http://pruebas.testiot.co/post-data.php";
128
129 // Keep this API Key value to be compatible with the PHP
   code provided in the project page.
130 // If you change the apiKeyValue value, the PHP file
   /post-data.php also needs to have the same key
131 String apiKeyValue = "tPmAT5Ab3j7F9";
132
133 // Configure TinyGSM library
134 // SIM card PIN (leave empty, if not defined)
135 const char simPIN[]   = "";
136
```

```
137 // Your phone number to send SMS: + (plus sign) and country
    code, for Portugal +351, followed by phone number
138 // SMS_TARGET Example for Portugal +351XXXXXXXXX
139 #define SMS_TARGET  " " // Información de la SIM del
    usuario al cuál se el enviarán el sms con los resultados
140
141
142 // Your GPRS credentials (leave empty, if not needed)
143 const char apn[]      = "web.colombiamovil.com.co"; // APN
    (example: internet.vodafone.pt) use
    https://wiki.apnchanger.org
144 const char gprsUser[] = ""; // GPRS User
145 const char gprsPass[] = ""; // GPRS Password
146
147
148 // Servidor de almacenamiento de datos
149 // The server variable can be just a domain name or it can
    have a subdomain. It depends on the service you are using
150 const char server[] = "pruebas.testiot.co"; // domain name:
    example.com, maker.ifttt.com, etc
151 const char resource[] = "/post-data.php";         //
    resource path, for example: /post-data.php
152 const int  port = 80; // server port number
153
154 // Librerias y definiciones GSM- Comandos AT-4GLTE
155 // Configure TinyGSM library
156 #define TINY_GSM_MODEM_SIM7600      // Modem is SIM800
157 #define TINY_GSM_RX_BUFFER   1024  // Set RX buffer to 1Kb
158
159 #include <Wire.h>
160 #include <TinyGsmClient.h>
161
162 // TTGO T-Call pins... pines de la sp32 usados por módulo
    sim 7600 SA
163 #define MODEM_RST            5
164 #define MODEM_PWKEY          4
165 #define MODEM_POWER_ON       25
166 #define MODEM_TX             27
167 #define MODEM_RX             26
168 #define I2C_SDA              21
169 #define I2C_SCL              22
170
171 // Set serial for debug console (to Serial Monitor, default
    speed 115200)
172 #define SerialMon Serial
173 // Set serial for AT commands (to SIM800 module)
174 #define SerialAT Serial1
175
176 // Configure TinyGSM library
177 #define TINY_GSM_MODEM_SIM7600      // Modem is SIM800
178 #define TINY_GSM_RX_BUFFER   1024  // Set RX buffer to 1Kb
```

```
179
180 // Define the serial console for debug prints, if needed
181 //#define DUMP_AT_COMMANDS
182
183 #include <Wire.h>
184 #include <TinyGsmClient.h>
185
186 #ifdef DUMP_AT_COMMANDS
187   #include <StreamDebugger.h>
188   StreamDebugger debugger(SerialAT, SerialMon);
189   TinyGsm modem(debugger);
190 #else
191   TinyGsm modem(SerialAT);
192 #endif
193
194 // I2C for SIM800 (to keep it running when powered from
    battery)
195 TwoWire I2CPower = TwoWire(0);
196
197 // TinyGSM Client for Internet connection
198 TinyGsmClient client(modem);
199
200
201 #define TIME_TO_SLEEP  3600        /* Time ESP32 will go to
    sleep (in seconds) 3600 seconds = 1 hour */
202
203 #define IP5306_ADDR          0x75
204 #define IP5306_REG_SYS_CTL0  0x00
205
206 bool setPowerBoostKeepOn(int en){
207   I2CPower.beginTransmission(IP5306_ADDR);
208   I2CPower.write(IP5306_REG_SYS_CTL0);
209   if (en) {
210     I2CPower.write(0x37); // Set bit1: 1 enable 0 disable
      boost keep on
211   } else {
212     I2CPower.write(0x35); // 0x37 is default reg value
213   }
214   return I2CPower.endTransmission() == 0;
215 }
216
217 // cadenas de texto de entrega de resultados vía SMS
218 String test = "Wiffi: temperatura ambiente en farenheit ";
219 String test1 = "humedad ambiente  ";
220 String test2 = "temperatura suelo celcius  ";
221 String test3 = "temperatura suelo   farenheit";
222 String test4 = "temperatura ambiente celcius  ";
223 String test5 = "sensacion termica celcius   ";
224 String test6 = "sensacion termica farenheit  ";
225 String test7 = "humedad suelo  ";
226 String test8 = "PH  ";
```

```
String test9 = "4G LTE: temperatura ambiente en farenheit
";
String test10 = "4G LTE:humedad ambiente  ";
String test11 = "4G LTE:temperatura suelo celcuis  ";
String test12 = "4G LTE:temperatura suelo farenheit  ";
String test13 = "4G LTE:temperatura ambiente farenheit  ";
String test14 = "4G LTE:sensacion termica celcius  ";
String test15 = "4G LTE:sensacion termica farenheit  ";
String test16 = "4G LTE:humedad suelo  ";
String test17 = "4G LTE:PH  ";


void setup() {
  Serial.begin(115200);
  // Inicialización librerías
  sensors.begin();
  dht.begin();

  }



void loop(){
  if(bootCount == 0) //Conteo de inicio del módulo
  {

      bootCount = bootCount+1;
  }else
  {

        // código wifi
        Serial.println("Código wifi");

        // Set modem reset, enable, power pins
        pinMode(MODEM_PWKEY, OUTPUT);
        pinMode(MODEM_RST, OUTPUT);
        pinMode(MODEM_POWER_ON, OUTPUT);
        digitalWrite(MODEM_PWKEY, LOW);
        digitalWrite(MODEM_RST, LOW);
        digitalWrite(MODEM_POWER_ON, LOW);

        // Lectura sensor DHT
      float h = dht.readHumidity();
      // Read temperature as Celsius (the default)
      float t = dht.readTemperature();
      // Read temperature as Fahrenheit (isFahrenheit =
      true)
      float f = dht.readTemperature(true);

      // Check if any reads failed and exit early (to try
      again).
```

```
275       if (isnan(h) || isnan(t) || isnan(f)) {
276         Serial.println(F("Failed to read from DHT
            sensor!"));
277         return;
278       }
279
280       // Compute heat index in Fahrenheit (the default)
281       float hif = dht.computeHeatIndex(f, h);
282       // Compute heat index in Celsius (isFahreheit = false)
283       float hic = dht.computeHeatIndex(t, h, false);
284
285
286       // Lectura YL-69
287       int const readYL69value = analogRead(YL69Pin);
288       // map inversely to 0..10%
289       int const convertedPercentage = map(readYL69value,
          4095, 1200, 0, 100);
290
291
292       // Lectura DSB 18B20
293       sensors.requestTemperatures();
294       float temperatureC = sensors.getTempCByIndex(0);
295       float temperatureF = sensors.getTempFByIndex(0);
296
297       // ciclo repetitivo primeras 10 muestras con
          comunicación wifi
298
299       while(bootCount1 <= 9)
300     {
301       if (bootCount1 == 0){
302
303         // Conexión red wifi
304         WiFi.begin(ssid, password);
305         Serial.println("Connecting");
306
307
308         while(WiFi.status() != WL_CONNECTED) {
309               delay(500);
310               Serial.print(".");
311             }
312           Serial.println("");
313           Serial.print("Connected to WiFi network with IP
              Address: ");
314           Serial.println(WiFi.localIP());
315         }
316
317           if(WiFi.status()== WL_CONNECTED){
318             // Conexión al servidor web diseñado
319             HTTPClient http;
320
```

```
321             // Your Domain name with URL path or IP address
                with path
322             http.begin(serverName);
323
324             // Specify content-type header
325             http.addHeader("Content-Type",
                "application/x-www-form-urlencoded");
326
327             // acumuladores de las muestras tomadas para
                promedio
328
329             Po = (1023 - analogRead(pHpin)) / 73.07; //
                Read and reverse the analogue input value from
                the pH sensor then scale 0-14.
330
331             acum_humedad_suelo = acum_humedad_suelo +
                convertedPercentage;
332             acum_temp_suelo_celcius_Array =
                acum_temp_suelo_celcius_Array + temperatureC;
333             acum_temp_suelo_celcius_Farenheit =
                acum_temp_suelo_celcius_Farenheit +
                temperatureF;
334             acum_hum_Ambiente_Array =
                acum_hum_Ambiente_Array + h;
335             acum_temp_amb_celcius = acum_temp_amb_celcius +
                f;
336             acum_temp_amb_farenheit =
                acum_temp_amb_farenheit + t;
337             acum_sensa_termi_celcius_Array =
                acum_sensa_termi_celcius_Array + hic;
338             acum_termi_faren_Array = acum_termi_faren_Array
                + hif;
339             acum_ph_liqui = acum_ph_liqui + Po;
340
341             // Muestra de las lecturas en el shell
342
343             Serial.println("PH Sen 0161 Lectura");
344             Serial.println(Po, 2);// Print the result in
                the serial monitor.
345
346             Serial.print(F("DSB 18B20 lectura: "));
347             Serial.print(temperatureC);
348             Serial.println("ºC");
349             Serial.print(temperatureF);
350             Serial.println("ºF");
351
352             Serial.print("Lectura (YL-69): ");
353             Serial.print(convertedPercentage);
354             Serial.print("%\n");
355
356             Serial.print(F("DHT lectura: "));
```

```
357             Serial.print(F("Humedad: "));
358             Serial.print(h);
359             Serial.print(F("%  Temperatura: "));
360             Serial.print(t);
361             Serial.print(F("°C "));
362             Serial.print(f);
363             Serial.print(F("°F  sensación térmica: "));
364             Serial.print(hic);
365             Serial.print(F("°C "));
366             Serial.print(hif);
367             Serial.println(F("°F"));
368
369             // Enpaquetamiento de datos a enviar
370             // Prepare your HTTP POST request data
371             String httpRequestData = "api_key=" +
                apiKeyValue + "&value1=" + String(temperatureC)
                + "&value2=" + String(temperatureF) +
                "&value3=" + String(h) + "&value4=" + String(t)
                + "&value5=" + String(f) + "&value6=" +
                String(hic) + "&value7=" + String(hif) +
                "&value8=" + String(convertedPercentage) +
                "&value9=" + String(Po);
372             Serial.print("httpRequestData: ");
373             Serial.println(httpRequestData);
374
375               //    Cálculos de los promedios
376               temp_amb_celcius_prom =
                  acum_temp_amb_celcius/10;
377               hum_Ambiente_Array_prom  =
                  acum_hum_Ambiente_Array/10;
378               temp_suelo_celcius_Array_prom =
                  acum_temp_suelo_celcius_Array/10;
379               temp_suelo_celcius_Farenheit_prom =
                  acum_temp_suelo_celcius_Farenheit/10;
380               temp_amb_farenheit_prom =
                  acum_temp_amb_farenheit/10;
381               sensa_termi_celcius_Array_prom =
                  acum_sensa_termi_celcius_Array/10;
382               sensa_termi_faren_Array_prom  =
                  acum_termi_faren_Array/10;
383               humedad_suelo_prom  = acum_humedad_suelo/10;
384               ph_liqui_prom  = acum_ph_liqui/10;
385
386
387             // Envío de datos al servidor
388             // Send HTTP POST request
389             int httpResponseCode =
                http.POST(httpRequestData);
390
391             if (httpResponseCode>0) {
392               Serial.print("HTTP Response code: ");
```

```
393               Serial.println(httpResponseCode);
394             }
395             else {
396               Serial.print("Error code: ");
397               Serial.println(httpResponseCode);
398             }
399
400             if(bootCount1 == 9){
401
402               // Muestra de los promedios en el shell
403              Serial.println("Promedios wiffi");
404              Serial.print(F("Humedad suelo: "));
405              Serial.print(humedad_suelo_prom);
406              Serial.print(F("%  Temperatura: "));
407              Serial.print(temp_suelo_celcius_Array_prom);
408              Serial.print(F("°C "));
409              Serial.print(temp_suelo_celcius_Farenheit_prom);
410              Serial.print(F("°F  Sensación térmica: "));
411              Serial.print(sensa_termi_celcius_Array_prom);
412              Serial.print(F("°C "));
413              Serial.print(sensa_termi_faren_Array_prom);
414              Serial.println(F("°F"));
415              Serial.println("PH promedio");
416              Serial.println(ph_liqui_prom);
417              http.end();
418              Serial.println("WiFi Disconnected");
419
420               }
421
422       }// se cierra while de conexión
423
424         bootCount1++;
425         bootCount2++;
426     }// se cierra el while
427
428
429
430         //bootCount1++;
431         //bootCount2++;
432
433     }
434
435
436
437     // Ciclo repetitivo para envío de 10 muestras con
        tecnología 4G LTE
438     while(bootCount1 >= 10 && bootCount1 < 20)
439     {
440         // código 4GLTE
441         Serial.println("Conexión 4GLTE");
442
```

```
443         pinMode(MODEM_PWKEY, OUTPUT);
444         pinMode(MODEM_RST, OUTPUT);
445         pinMode(MODEM_POWER_ON, OUTPUT);
446         digitalWrite(MODEM_PWKEY, LOW);
447         digitalWrite(MODEM_RST, LOW);
448         digitalWrite(MODEM_POWER_ON, LOW);
449
450          // Lectura sensor DHT
451         float h = dht.readHumidity();
452         // Read temperature as Celsius (the default)
453         float t = dht.readTemperature();
454         // Read temperature as Fahrenheit (isFahrenheit =
            true)
455         float f = dht.readTemperature(true);
456
457         // Check if any reads failed and exit early (to try
            again).
458         if (isnan(h) || isnan(t) || isnan(f)) {
459           Serial.println(F("Failed to read from DHT
              sensor!"));
460           return;
461         }
462
463         // Compute heat index in Fahrenheit (the default)
464         float hif = dht.computeHeatIndex(f, h);
465         // Compute heat index in Celsius (isFahreheit = false)
466         float hic = dht.computeHeatIndex(t, h, false);
467
468
469         // Lectura YL-69
470         int const readYL69value = analogRead(YL69Pin);
471         // map inversely to 0..10%
472         int const convertedPercentage = map(readYL69value,
            4095, 1200, 0, 100);
473
474
475         // DSB
476         sensors.requestTemperatures();
477         float temperatureC = sensors.getTempCByIndex(0);
478         float temperatureF = sensors.getTempFByIndex(0);
479
480
481         Po = (1023 - analogRead(pHpin)) / 73.07; // Read and
            reverse the analogue input value from the pH sensor
            then scale 0-14.
482
483         // Pines del modo reset, enable y pin de potencia
            para la transmisión
484           // Set modem reset, enable, power pins
485         pinMode(MODEM_PWKEY, OUTPUT);
486         pinMode(MODEM_RST, OUTPUT);
```

```
487       pinMode(MODEM_POWER_ON, OUTPUT);
488       digitalWrite(MODEM_PWKEY, LOW);
489       digitalWrite(MODEM_RST, HIGH);
490       digitalWrite(MODEM_POWER_ON, HIGH);
491
492         // Set GSM module baud rate and UART pins
493       SerialAT.begin(115200, SERIAL_8N1, MODEM_RX,
          MODEM_TX);
494       delay(3000);
495
496       //modem.restart();
497       // use modem.init() if you don't need the complete
          restart
498
499       // Unlock your SIM card with a PIN if needed
500       if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
501         modem.simUnlock(simPIN);
502       }
503
504       String result;
505
506     do {
507             result = modem.setNetworkMode(2);
508             delay(500);
509         } while (result != "OK");
510
511     // ciclo repetitivo para empaquetamiento y envío de
      datos
512     while(bootCount1 >= 10 && bootCount1 < 20)
513     {
514       SerialMon.print("Connecting to APN: ");
515       SerialMon.print(apn);
516       if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
517         SerialMon.println(" fail");
518       }
519       else {
520         SerialMon.println(" OK");
521         SerialMon.print("Connecting to ");
522         SerialMon.print(server);
523         if (!client.connect(server, port)) {
524           SerialMon.println(" fail");
525         }
526         else {
527           SerialMon.println(" OK");
528           SerialAT.println("AT+CPSI?");     //Get
              connection type and band
529       delay(500);
530       if (SerialAT.available()) {
531         String r = SerialAT.readString();
532         Serial.println(r);
533       }
```

```
534         //
535               // Variables acumuladoras valores de las
                  muestras
536               acum_humedad_suelo_lte = acum_humedad_suelo_lte
                  + convertedPercentage;
537               acum_temp_suelo_celcius_Array_lte =
                  acum_temp_suelo_celcius_Array_lte +
                  temperatureC;
538               acum_temp_suelo_celcius_Farenheit_lte =
                  acum_temp_suelo_celcius_Farenheit_lte +
                  temperatureF;
539               acum_hum_Ambiente_Array_lte =
                  acum_hum_Ambiente_Array_lte + h;
540               acum_temp_amb_celcius_lte =
                  acum_temp_amb_celcius_lte + f;
541               acum_temp_amb_farenheit_lte =
                  acum_temp_amb_farenheit_lte + t;
542               acum_sensa_termi_celcius_Array_lte =
                  acum_sensa_termi_celcius_Array_lte + hic;
543               acum_termi_faren_Array_lte =
                  acum_termi_faren_Array_lte + hif;
544               acum_ph_liqui_lte = acum_ph_liqui_lte + Po;
545
546         // Muesta de resultados en el shell
547         Serial.println("PH Sen 0161 Lectura");
548               Serial.println(Po, 2);// Print the result in
                  the serial monitor.
549
550               Serial.print(F("DSB 18B20 lectura: "));
551               Serial.print(temperatureC);
552               Serial.println("ºC");
553               Serial.print(temperatureF);
554               Serial.println("ºF");
555
556               Serial.print("Lectura (YL-69): ");
557               Serial.print(convertedPercentage);
558               Serial.print("%\n");
559
560               Serial.print(F("DHT lectura: "));
561               Serial.print(F("Humedad: "));
562               Serial.print(h);
563               Serial.print(F("%  Temperatura: "));
564               Serial.print(t);
565               Serial.print(F("°C "));
566               Serial.print(f);
567               Serial.print(F("°F  sensación térmica: "));
568               Serial.print(hic);
569               Serial.print(F("°C "));
570               Serial.print(hif);
571               Serial.println(F("°F"));
572
```

```
573             // Empaquetamiento de datos para la transmisión
574
575       // Making an HTTP POST request
576       SerialMon.println("Performing HTTP POST request...");
577       // Prepare your HTTP POST request data (Temperature
          in Celsius degrees)
578       String httpRequestData = "api_key=" + apiKeyValue +
          "&value1=" + String(temperatureC) + "&value2=" +
          String(temperatureF) + "&value3=" + String(h) +
          "&value4=" + String(t) + "&value5=" + String(f) +
          "&value6=" + String(hic) + "&value7=" + String(hif) +
          "&value8=" + String(convertedPercentage) + "&value9="
          + String(Po) ;
579
580       client.print(String("POST ") + resource + "
          HTTP/1.1\r\n");
581       client.print(String("Host: ") + server + "\r\n");
582       client.println("Connection: close");
583       client.println("Content-Type:
          application/x-www-form-urlencoded");
584       client.print("Content-Length: ");
585       client.println(httpRequestData.length());
586       client.println();
587       client.println(httpRequestData);
588
589       unsigned long timeout = millis();
590       while (client.connected() && millis() - timeout <
          10000L) {
591         // Print available data (HTTP response from server)
592         while (client.available()) {
593           char c = client.read();
594           SerialMon.print(c);
595           timeout = millis();
596         }
597       }
598       SerialMon.println();
599
600       // Cálculos de los promedios
601       temp_amb_celcius_prom_lte =
          acum_temp_amb_celcius_lte/10;
602       hum_Ambiente_Array_prom_lte  =
          acum_hum_Ambiente_Array_lte/10;
603       temp_suelo_celcius_Array_prom_lte =
          acum_temp_suelo_celcius_Array_lte/10;
604       temp_suelo_celcius_Farenheit_prom_lte =
          acum_temp_suelo_celcius_Farenheit_lte/10;
605       temp_amb_farenheit_prom_lte =
          acum_temp_amb_farenheit_lte/10;
606       sensa_termi_celcius_Array_prom_lte =
          acum_sensa_termi_celcius_Array_lte/10;
```

```
607       sensa_termi_faren_Array_prom_lte  =
          acum_termi_faren_Array_lte/10;
608       humedad_suelo_prom_lte  = acum_humedad_suelo_lte/10;
609       ph_liqui_prom_lte  = acum_ph_liqui_lte/10;
610
611
612
613     //delay(250);
614     while (bootCount1 == 19) {
615       // Close client and disconnect
616       client.stop();
617       SerialMon.println(F("Server disconnected"));
618       modem.gprsDisconnect();
619       SerialMon.println(F("GPRS disconnected"));
620       bootCount1++;
621
622     }
623
624
625       }
626       }
627
628         bootCount1++;
629         bootCount2++;
630
631     }
632
633       // Muestra de los promedios en la shell
634       Serial.println("Promedios 4GLTE");
635       Serial.print(F("Humedad: "));
636       Serial.print(humedad_suelo_prom_lte);
637       Serial.print(F("%  Temperatura: "));
638       Serial.print(temp_suelo_celcius_Array_prom_lte);
639       Serial.print(F("°C "));
640       Serial.print(temp_suelo_celcius_Farenheit_prom_lte);
641       Serial.print(F("°F  Sensación térmica: "));
642       Serial.print(sensa_termi_celcius_Array_prom_lte);
643       Serial.print(F("°C "));
644       Serial.print(sensa_termi_faren_Array_prom_lte);
645       Serial.println(F("°F"));
646
647        // Envío de datos promedios al usuario mediante
          mensaje de texto SMS
648
649        if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
650           modem.simUnlock(simPIN);
651         }
652       //Organización datos sms
653         String smsMessage = test + temp_amb_celcius_prom +
          "\n" + test1 + hum_Ambiente_Array_prom + "\n" +
          test2 + temp_suelo_celcius_Array_prom + "\n" +
```

```
        test3 + temp_suelo_celcius_Farenheit_prom;
654     if(modem.sendSMS(SMS_TARGET, smsMessage)){
655     SerialMon.println(smsMessage);
656     }
657     else{
658       SerialMon.println("SMS failed to send");
659     }
660     String smsMessage1 = test4 +
        temp_amb_farenheit_prom + "\n" + test5 +
        sensa_termi_celcius_Array_prom  + "\n" + test6 +
        sensa_termi_faren_Array_prom + "\n" + test7+
        humedad_suelo_prom;
661     if(modem.sendSMS(SMS_TARGET, smsMessage1)){
662     SerialMon.println(smsMessage1);
663     }
664     else{
665       SerialMon.println("SMS failed to send");
666     }
667     String smsMessage2 = test8 + ph_liqui_prom;
668     if(modem.sendSMS(SMS_TARGET, smsMessage2)){
669     SerialMon.println(smsMessage2);
670     }
671     else{
672       SerialMon.println("SMS failed to send");
673     }
674
675     String smsMessage3 = test9 +
        temp_amb_celcius_prom_lte + "\n" + test10 +
        hum_Ambiente_Array_prom_lte + "\n" + test11 +
        temp_suelo_celcius_Array_prom_lte + "\n" + test12 +
        temp_suelo_celcius_Farenheit_prom_lte;
676     if(modem.sendSMS(SMS_TARGET, smsMessage3)){
677     SerialMon.println(smsMessage3);
678     }
679     else{
680       SerialMon.println("SMS failed to send");
681     }
682     String smsMessage4 = test13 +
        temp_amb_farenheit_prom_lte + "\n" + test14 +
        sensa_termi_celcius_Array_prom_lte  + "\n" + test15
        + sensa_termi_faren_Array_prom_lte + "\n" + test16+
        humedad_suelo_prom_lte;
683     if(modem.sendSMS(SMS_TARGET, smsMessage4)){
684     SerialMon.println(smsMessage4);
685     }
686     else{
687       SerialMon.println("SMS failed to send");
688     }
689     String smsMessage5 = test17 + ph_liqui_prom_lte;
690     if(modem.sendSMS(SMS_TARGET, smsMessage5)){
691     SerialMon.println(smsMessage5);
```

```
692           }
693           else{
694             SerialMon.println("SMS failed to send");
695           }
696
697         delay(120000); //acá se modifica el tiempo de periodo
            despierto  del procesador 300000 = 5 min
698
699     }
700
701     delay(3000);
702
703     // Funciones para dormir y despertar el procesador
704     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP *
        uS_TO_S_FACTOR);
705     esp_deep_sleep_start();
706 }
```

Apéndice C. Código usado para la práctica piloto en GSM.

```
// Your GPRS credentials (leave empty, if not needed)
const char apn[]      = "web.colombiamovil.com.co"; // APN (example: internet.vodafone.pt) use https://wiki.apnchanger.org
const char gprsUser[] = ""; // GPRS User
const char gprsPass[] = ""; // GPRS Password

// SIM card PIN (leave empty, if not defined)
const char simPIN[]   = "";

// Server details
// The server variable can be just a domain name or it can have a subdomain. It depends on the service you are using
const char server[] = ""; // domain name: example.com, maker.ifttt.com, etc
const char resource[] = "";         // resource path, for example: /post-data.php
const int  port = 80;                              // server port number

// Keep this API Key value to be compatible with the PHP code provided in the project page.
// If you change the apiKeyValue value, the PHP file /post-data.php also needs to have the same key
String apiKeyValue = "tPmAT5Ab3j7F9";

// TTGO T-Call pins
#define MODEM_RST            5
#define MODEM_PWKEY          4
#define MODEM_POWER_ON       23
#define MODEM_TX             27
#define MODEM_RX             26
#define I2C_SDA              21
#define I2C_SCL              22
// BME280 pins
#define I2C_SDA_2            18
#define I2C_SCL_2            19

// Set serial for debug console (to Serial Monitor, default speed 115200)
#define SerialMon Serial
// Set serial for AT commands (to SIM800 module)
#define SerialAT Serial1

// Configure TinyGSM library
#define TINY_GSM_MODEM_SIM800      // Modem is SIM800
#define TINY_GSM_RX_BUFFER   1024  // Set RX buffer to 1Kb
\
```

```
40 // Define the serial console for debug prints, if needed
41 //#define DUMP_AT_COMMANDS
42
43 #include <Wire.h>
44 #include <TinyGsmClient.h>
45
46 #ifdef DUMP_AT_COMMANDS
47   #include <StreamDebugger.h>
48   StreamDebugger debugger(SerialAT, SerialMon);
49   TinyGsm modem(debugger);
50 #else
51   TinyGsm modem(SerialAT);
52 #endif
53
54 #include <Adafruit_Sensor.h>
55 #include <Adafruit_BMP280.h>
56
57 #include "DHT.h"
58
59 #define DHTPIN 0
60 #define DHTTYPE DHT22
61
62 // I2C for SIM800 (to keep it running when powered from
   battery)
63 TwoWire I2CPower = TwoWire(0);
64
65 // I2C for BME280 sensor
66 //TwoWire I2CBME = TwoWire(1);
67 //Adafruit_BMP280 bme;
68
69 // TinyGSM Client for Internet connection
70 TinyGsmClient client(modem);
71
72 #define uS_TO_S_FACTOR 1000000UL   /* Conversion factor for
   micro seconds to seconds */
73 #define TIME_TO_SLEEP  3600        /* Time ESP32 will go to
   sleep (in seconds) 3600 seconds = 1 hour */
74
75 #define IP5306_ADDR          0x75
76 #define IP5306_REG_SYS_CTL0  0x00
77
78
79 bool setPowerBoostKeepOn(int en){
80   I2CPower.beginTransmission(IP5306_ADDR);
81   I2CPower.write(IP5306_REG_SYS_CTL0);
82   if (en) {
83     I2CPower.write(0x37); // Set bit1: 1 enable 0 disable
     boost keep on
84   } else {
85     I2CPower.write(0x35); // 0x37 is default reg value
86   }
```

```
87    return I2CPower.endTransmission() == 0;
88  }
89
90  DHT dht(DHTPIN, DHTTYPE);
91
92
93  void setup() {
94    // Set serial monitor debugging window baud rate to 115200
95    SerialMon.begin(115200);
96
97    // Start I2C communication
98    I2CPower.begin(I2C_SDA, I2C_SCL, 400000);
99  //  I2CBME.begin(I2C_SDA_2, I2C_SCL_2, 400000);
100
101   // Keep power when running from battery
102   bool isOk = setPowerBoostKeepOn(1);
103   SerialMon.println(String("IP5306 KeepOn ") + (isOk ? "OK"
      : "FAIL"));
104
105   // Set modem reset, enable, power pins
106   pinMode(MODEM_PWKEY, OUTPUT);
107   pinMode(MODEM_RST, OUTPUT);
108   pinMode(MODEM_POWER_ON, OUTPUT);
109   digitalWrite(MODEM_PWKEY, LOW);
110   digitalWrite(MODEM_RST, HIGH);
111   digitalWrite(MODEM_POWER_ON, HIGH);
112
113   // Set GSM module baud rate and UART pins
114   SerialAT.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
115   delay(3000);
116
117   // Restart SIM800 module, it takes quite some time
118   // To skip it, call init() instead of restart()
119   SerialMon.println("Initializing modem...");
120   modem.restart();
121   // use modem.init() if you don't need the complete restart
122
123   // Unlock your SIM card with a PIN if needed
124   if (strlen(simPIN) && modem.getSimStatus() != 3 ) {
125     modem.simUnlock(simPIN);
126   }
127
128   // You might need to change the BME280 I2C address, in
      our case it's 0x76
129
130   /*
131   if (!bme.begin(0x76)) {
132     Serial.println("Could not find a valid BME280 sensor,
      check wiring!");
133     while (1);
134   }
```

```
135   */
136    dht.begin();
137   // Configure the wake up source as timer wake up
138   //esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP *
      uS_TO_S_FACTOR);
139 }
140
141 void loop() {
142   SerialMon.print("Connecting to APN: ");
143   SerialMon.print(apn);
144   if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
145     SerialMon.println(" fail");
146   }
147   else {
148     SerialMon.println(" OK");
149
150     SerialMon.print("Connecting to ");
151     SerialMon.print(server);
152     if (!client.connect(server, port)) {
153       SerialMon.println(" fail");
154     }
155     else {
156       SerialMon.println(" OK");
157
158
159       // Reading temperature or humidity takes about 250
          milliseconds!
160   // Sensor readings may also be up to 2 seconds 'old' (its
      a very slow sensor)
161   float h = dht.readHumidity();
162   // Read temperature as Celsius (the default)
163   float t = dht.readTemperature();
164   // Read temperature as Fahrenheit (isFahrenheit = true)
165   float f = dht.readTemperature(true);
166
167   // Check if any reads failed and exit early (to try
      again).
168   if (isnan(h) || isnan(t) || isnan(f)) {
169     Serial.println(F("Failed to read from DHT sensor!"));
170     return;
171   }
172
173   // Compute heat index in Fahrenheit (the default)
174   float hif = dht.computeHeatIndex(f, h);
175   // Compute heat index in Celsius (isFahreheit = false)
176   float hic = dht.computeHeatIndex(t, h, false);
177
178
179       // Making an HTTP POST request
180       SerialMon.println("Performing HTTP POST request...");
```

```
181       // Prepare your HTTP POST request data (Temperature
          in Celsius degrees)
182       String httpRequestData = "api_key=" + apiKeyValue +
          "&value1=" + String(dht.readHumidity())
183                        + "&value2=" +
                           String(dht.readTemperature()) +
                           "&value3=" +
                           String(dht.readTemperature(true)) + "";
184
185
186       client.print(String("POST ") + resource + "
          HTTP/1.1\r\n");
187       client.print(String("Host: ") + server + "\r\n");
188       client.println("Connection: close");
189       client.println("Content-Type:
          application/x-www-form-urlencoded");
190       client.print("Content-Length: ");
191       client.println(httpRequestData.length());
192       client.println();
193       client.println(httpRequestData);
194
195       unsigned long timeout = millis();
196       while (client.connected() && millis() - timeout <
          10000L) {
197         // Print available data (HTTP response from server)
198         while (client.available()) {
199           char c = client.read();
200           SerialMon.print(c);
201           timeout = millis();
202         }
203       }
204       SerialMon.println();
205
206       // Close client and disconnect
207       client.stop();
208       SerialMon.println(F("Server disconnected"));
209       modem.gprsDisconnect();
210       SerialMon.println(F("GPRS disconnected"));
211     }
212   }
213   // Put ESP32 into deep sleep mode (with timer wake up)
214   //esp_deep_sleep_start();
215 }
```

Apéndice D. Código `post-data.php` usado para la comunicación entre la ESP32 y la base de datos.

```php
<?php

$servername = "localhost";

// Reemplazar con el nombre de la base de datos
$dbname = "";
// Reemplazar con el nombre de usuario
$username = "";
// Reemplazar con la contraseña
$password = "";

// Keep this API Key value to be compatible with the ESP32 code provided in the project page. If you change this value, the ESP32 sketch needs to match
$api_key_value = "tPmAT5Ab3j7F9";

$api_key = $value1 = $value2 = $value3 = $value4 = $value5 = $value6 = $value7 = $value8 = $value9 = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = test_input($_POST["api_key"]);
    if($api_key == $api_key_value) {
        $value1 = test_input($_POST["value1"]);
        $value2 = test_input($_POST["value2"]);
        $value3 = test_input($_POST["value3"]);
        $value4 = test_input($_POST["value4"]);
        $value5 = test_input($_POST["value5"]);
        $value6 = test_input($_POST["value6"]);
        $value7 = test_input($_POST["value7"]);
        $value8 = test_input($_POST["value8"]);
        $value9 = test_input($_POST["value9"]);

        // Create connection
        $conn = new mysqli($servername, $username, $password, $dbname);
        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }

        $sql = "INSERT INTO Sensor (value1, value2, value3, value4, value5, value6, value7, value8, value9)
        VALUES ('" . $value1 . "', '" . $value2 . "', '" . $value3 . "', '" . $value4 . "', '" . $value5 . "',
```

```
        '" . $value6 . "', '" . $value7 . "', '" . $value8
        . "', '" . $value9 . "')";

        if ($conn->query($sql) === TRUE) {
            echo "New record created successfully";
        }
        else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }

        $conn->close();
    }
    else {
        echo "Wrong API Key provided.";
    }

}
else {
    echo "No data posted with HTTP POST.";
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

Apéndice E. Código `esp-chart.php` usado para la interfaz web.

```php
<?php

$servername = "localhost";

// REPLACE with your Database name
$dbname = "tiotcote_prueba1_esp32";
// REPLACE with Database user
$username = "tiotcote_prueba1_user";
// REPLACE with Database user password
$password = "Jhonc-1994";


// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, value1, value2, value3, value4, value5, value6, value7, value8, value9, reading_time FROM Sensor order by reading_time desc limit 40";

$result = $conn->query($sql);

while ($data = $result->fetch_assoc()){
    $sensor_data[] = $data;
}

$readings_time = array_column($sensor_data, 'reading_time');

// ******* Uncomment to convert readings time array to your timezone ********
/*$i = 0;
foreach ($readings_time as $reading){
    // Uncomment to set timezone to - 1 hour (you can change 1 to any number)
    $readings_time[$i] = date("Y-m-d H:i:s", strtotime("$reading - 1 hours"));
    // Uncomment to set timezone to + 4 hours (you can change 4 to any number)
    //$readings_time[$i] = date("Y-m-d H:i:s", strtotime("$reading + 4 hours"));
    $i += 1;
}*/

```

```
40 $value1 =
   json_encode(array_reverse(array_column($sensor_data,
   'value1')), JSON_NUMERIC_CHECK);
41 $value2 =
   json_encode(array_reverse(array_column($sensor_data,
   'value2')), JSON_NUMERIC_CHECK);
42 $value3 =
   json_encode(array_reverse(array_column($sensor_data,
   'value3')), JSON_NUMERIC_CHECK);
43 $value4 =
   json_encode(array_reverse(array_column($sensor_data,
   'value4')), JSON_NUMERIC_CHECK);
44 $value5 =
   json_encode(array_reverse(array_column($sensor_data,
   'value5')), JSON_NUMERIC_CHECK);
45 $value6 =
   json_encode(array_reverse(array_column($sensor_data,
   'value6')), JSON_NUMERIC_CHECK);
46 $value7 =
   json_encode(array_reverse(array_column($sensor_data,
   'value7')), JSON_NUMERIC_CHECK);
47 $value8 =
   json_encode(array_reverse(array_column($sensor_data,
   'value8')), JSON_NUMERIC_CHECK);
48 $value9 =
   json_encode(array_reverse(array_column($sensor_data,
   'value9')), JSON_NUMERIC_CHECK);
49
50 $reading_time = json_encode(array_reverse($readings_time),
   JSON_NUMERIC_CHECK);
51
52 /*echo $value1;
53 echo $value2;
54 echo $value3;
55 echo $reading_time;*/
56
57 $result->free();
58 $conn->close();
59 ?>
60
61 <!DOCTYPE html>
62 <html>
63 <meta name="viewport" content="width=device-width,
   initial-scale=1">
64   <script
     src="https://code.highcharts.com/highcharts.js"></script>
65   <style>
66     body {
67       min-width: 310px;
68       max-width: 1280px;
69       height: 500px;
```

```
      margin: 0 auto;
    }
    h2 {
      font-family: Arial;
      font-size: 2.5rem;
      text-align: center;
    }
  </style>
  <body>
    <h2>Módulo IoT WiFi y 4G LTE</h2>
    <div id="chart-temperature-dsb1820-C" class="container"></div>
    <div id="chart-temperature-dsb1820-F" class="container"></div>
    <div id="chart-humidity-dht22" class="container"></div>
    <div id="chart-temperature-dht22-C" class="container"></div>
    <div id="chart-temperature-dht22-F" class="container"></div>
    <div id="chart-heat-index-dht22-C" class="container"></div>
    <div id="chart-heat-index-dht22-F" class="container"></div>
    <div id="chart-ph" class="container"></div>
    <div id="chart-humidity-yl69" class="container"></div>
<script>

var value1 = <?php echo $value1; ?>;
var value2 = <?php echo $value2; ?>;
var value3 = <?php echo $value3; ?>;
var value4 = <?php echo $value4; ?>;
var value5 = <?php echo $value5; ?>;
var value6 = <?php echo $value6; ?>;
var value7 = <?php echo $value7; ?>;
var value8 = <?php echo $value8; ?>;
var value9 = <?php echo $value9; ?>;
var reading_time = <?php echo $reading_time; ?>;

/*
Acá va la interfaz gráfica para cada variable
*/

var chartTC_DSB1820 = new Highcharts.Chart({
  chart:{ renderTo : 'chart-temperature-dsb1820-C' },
  title: { text: 'DSB1820 Temperatura (°C)' },
  series: [{
    showInLegend: false,
    data: value1
  }],
  plotOptions: {
    line: { animation: false,
```

```
115       dataLabels: { enabled: true }
116     },
117     series: { color: '#059e8a' }
118   },
119   xAxis: {
120     type: 'datetime',
121     categories: reading_time
122   },
123   yAxis: {
124     title: { text: 'Temperatura (Celsius)' }
125     //title: { text: 'Temperature (Fahrenheit)' }
126   },
127   credits: { enabled: false }
128 });
129
130
131
132 var chartTF_DSB1820 = new Highcharts.Chart({
133   chart:{ renderTo : 'chart-temperature-dsb1820-F' },
134   title: { text: 'DSB1820 Temperatura (°F)' },
135   series: [{
136     showInLegend: false,
137     data: value2
138   }],
139   plotOptions: {
140     line: { animation: false,
141       dataLabels: { enabled: true }
142     }
143   },
144   xAxis: {
145     type: 'datetime',
146     //dateTimeLabelFormats: { second: '%H:%M:%S' },
147     categories: reading_time
148   },
149   yAxis: {
150     title: { text: 'Temperatura (Fahrenheit)' }
151   },
152   credits: { enabled: false }
153 });
154
155 var chartH_DHT22 = new Highcharts.Chart({
156   chart:{ renderTo:'chart-humidity-dht22' },
157   title: { text: 'DHT22 Humedad' },
158   series: [{
159     showInLegend: false,
160     data: value3
161   }],
162   plotOptions: {
163     line: { animation: false,
164       dataLabels: { enabled: true }
165     },
```

```
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Humedad (%)' }
  },
  credits: { enabled: false }
});

var chartTC_DHT22 = new Highcharts.Chart({
  chart:{ renderTo:'chart-temperature-dht22-C' },
  title: { text: 'DHT22 Temperatura (°C)' },
  series: [{
    showInLegend: false,
    data: value4
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Temperatura (Celsius)' }
  },
  credits: { enabled: false }
});

var chartTF_DHT22 = new Highcharts.Chart({
  chart:{ renderTo:'chart-temperature-dht22-F' },
  title: { text: 'DHT22 Temperatura (°F)' },
  series: [{
    showInLegend: false,
    data: value5
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
```

```
  },
  yAxis: {
    title: { text: 'Temperatura (Fahrenheit)' }
  },
  credits: { enabled: false }
});

var chartHIC_DHT22 = new Highcharts.Chart({
  chart:{ renderTo:'chart-heat-index-dht22-C' },
  title: { text: 'DHT22 Sensación Térmica (°C)' },
  series: [{
    showInLegend: false,
    data: value6
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Temperatura (Celsius)' }
  },
  credits: { enabled: false }
});


var chartHIF_DHT22 = new Highcharts.Chart({
  chart:{ renderTo:'chart-heat-index-dht22-F' },
  title: { text: 'DHT22 Sensación Térmica (°F)' },
  series: [{
    showInLegend: false,
    data: value7
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Temperatura (Fahrenheit)' }
  },
```

```
  credits: { enabled: false }
});

var chartH_YL69 = new Highcharts.Chart({
  chart:{ renderTo:'chart-humidity-yl69' },
  title: { text: 'YL69 Humedad' },
  series: [{
    showInLegend: false,
    data: value8
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'Humedad (%)' }
  },
  credits: { enabled: false }
});

var chart_PH = new Highcharts.Chart({
  chart:{ renderTo:'chart-ph' },
  title: { text: 'PH' },
  series: [{
    showInLegend: false,
    data: value9
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: true }
    },
    series: { color: '#18009c' }
  },
  xAxis: {
    type: 'datetime',
    categories: reading_time
  },
  yAxis: {
    title: { text: 'PH' }
  },
  credits: { enabled: false }
});
</script>
</body>
</html>
```

319