SISTEMA DE SEGUIMIENTO BOVINO POR MEDIO DE VARIABLES DE INTERÉS E INFORMACIÓN GEOGRÁFICA

SARA ROMÁN PALACIO
DANIELA ALEJANDRA SOLER ARAGÓN

Trabajo de grado para optar al título de
INGENIERA MECATRÓNICA

Camilo Alberto Buriticá Restrepo

UNIVERSIDAD EIA
INGENIERÍA MECATRÓNICA
ENVIGADO
2020
AGRADECIMIENTOS

A nuestros familiares y amigos que creyeron en nosotros y nos acompañaron en nuestra formación.

A nuestros profesores, que no enseñaron que no solo se aprende en la academia.

A nuestros directores de carrera que nos guiaron en nuestra formación e hicieron posible el desarrollo de este trabajo.

A nuestro director Camilo Alberto Buriticá, por la paciencia, acompañamiento y consejos dados en el desarrollo de este proyecto.

Agradecimientos especiales a nuestros compañeros y amigos Mateo Penagos Ramírez y Jonathan Veles Alonso por todo su apoyo y ayuda incondicional durante el desarrollo de este proyecto y a nuestras amigas Andrea Giovanna Esquiaqui y Valentina Osorio Peláez el apoyo en los peores momentos.

A todos, muchas gracias

Daniela Alejandra Soler Aragón
Sara Román Palacio
CONTENIDO

INTRODUCCIÓN

1. PRELIMINARES
   1.1 Planteamiento del problema
   1.2 Objetivos del proyecto
      1.2.1 Objetivo General
      1.2.2 Objetivos Específicos
   1.3 Marco de referencia
      1.3.1 Antecedentes
      2.3.2 Marco Teórico

METODOLOGÍA

2. DESARROLLO DEL PROYECTO
   2.1 IDENTIFICACIÓN DE REQUERIMIENTOS
   2.2 Diseño y selección de concepto
      1.2.1 Registro de soluciones
      2.2.2 Matriz morfológica
      3.2.3 Matriz de selección
   2.3 DISEÑO DE DETALLE
      1.3.1 Diseño electrónico
      2.3.2 Desarrollo del software de adquisición de datos
      3.3.3 Desarrollo de envío de datos a la nube
      4.3.4 Procesamiento de datos
      5.3.5 Interfaz gráfica
2.4 Pruebas ......................................................................................................................... 63
  2.4.1 Pruebas Hardware .................................................................................................. 63
  2.4.2 Pruebas Servidor .................................................................................................. 65
  2.4.3 Pruebas Software ................................................................................................. 66
  2.4.4 Pruebas de funcionamiento en un ambiente controlado .................................... 68
3 ANÁLISIS DE RESULTADOS .......................................................................................... 71
4 CONCLUSIONES Y CONSIDERACIONES FINALES .................................................... 72
REFERENCIAS .................................................................................................................. 74
ANEXOS .............................................................................................................................. 78
LISTA DE TABLAS

Tabla 1. Requerimientos de usuario. ................................................................. 22
Tabla 2. Características de ingeniería............................................................... 23
Tabla 3. Registro de soluciones – Toma de datos – Sensores............................ 27
Tabla 4. Registro de soluciones – Comunicación............................................. 30
Tabla 5. Registro de soluciones – Microprocesador........................................ 31
Tabla 6. Registro de soluciones – Servidores.................................................. 32
Tabla 7. Matriz morfológica.............................................................................. 33
Tabla 8. Matriz de selección ............................................................................ 35
Tabla 9. Arquitectura sentencia GPRMC (Quactel, 2016).................................. 49
Tabla 10. Configuraciones GPS....................................................................... 50
Tabla 11. Comparación entre formatos XML y JSON...................................... 55
Tabla 12. Comandos AT usados en el módulo M95.......................................... 56
Tabla 13. Comparación mediciones punto 1..................................................... 64
Tabla 14. Comparación mediciones punto 2..................................................... 64
Tabla 15. Comparación mediciones punto 3..................................................... 65
Tabla 16. Comparaciones mediciones GPS....................................................... 65
LISTA DE FIGURAS

Figura 1. Población de bovinos por departamento 2019 (Instituto Colombiano Agropecuario, 2019) ................................................................................................. 15

Figura 2. Predios Bovinos por departamento en 2019 (Instituto Colombiano Agropecuario, 2019) .............................................................. 16

Figura 3. House of Quality ................................................................. 24

Figura 4. Caja Negra basada en el análisis morfológico ................................... 26

Figura 5. Caja Transparente basada en el análisis morfológico ........................... 26

Figura 6. Circuito de pruebas unitarias y desarrollo ................................................ 36

Figura 7. Conexiones del circuito de pruebas unitarias y desarrollo ....................... 37

Figura 8. Conexiones PIC ................................................................ 38

Figura 9. Conexión sensor DHT22 y LM75A .................................................... 38

Figura 10. Conexión módulo L70 ............................................................... 39

Figura 11. Conexión módulo M95 ............................................................... 40

Figura 12. Circuito de carga ................................................................ 41

Figura 13. Conexión puertos UART y PICkit 3. .............................................. 41

Figura 14. PCB capa superior ................................................................. 43

Figura 15. PCB capa inferior ................................................................ 43

Figura 16. Modelado 3D de la capa superior ..................................................... 44

Figura 17. Modelado 3D de la capa inferior ..................................................... 44

Figura 18. Modelo 3D de la caja ............................................................... 45

Figura 19. Ensamble PCB ................................................................ 45

Figura 20. Dispositivo en su caja ............................................................... 46
Figura 21. Diagrama de flujo del sensor DHT22 ................................................................. 47
Figura 22. Datos del sensor DHT22 .................................................................................... 48
Figura 23. Diagrama de flujo para la toma de datos del módulo L70 ............................. 51
Figura 24. Tramas GPRMC enviadas por el módulo L70 .................................................. 51
Figura 25. Datos del sensor LM75A ................................................................................. 52
Figura 26. Diagrama de flujo del módulo M95 ................................................................. 54
Figura 27. Trama JSON mandadas generadas por el PIC ............................................... 56
Figura 28. Base de datos creada para cada bovino según su id .................................... 57
Figura 29. Trama guardada en la base de datos ............................................................... 58
Figura 30. Procesamiento fecha ...................................................................................... 58
Figura 31. Procesamiento temperaturas y humedad ....................................................... 59
Figura 32. Procesamiento coordenadas ......................................................................... 59
Figura 33. Mensaje inicial sección Mapa ...................................................................... 60
Figura 34. Estado actual ................................................................................................. 61
Figura 35. Mapa de calor ............................................................................................... 61
Figura 36. Mensaje inicial sección Datos ..................................................................... 62
Figura 37. Histórico de datos ......................................................................................... 63
Figura 38. Correspondencia de datos ............................................................................ 66
Figura 39. Verificación de alertas en Estado Actual ...................................................... 67
Figura 40. Verificación de alertas en Datos .................................................................. 67
Figura 41. Mapa de calor posición 1 ............................................................................ 69
Figura 42. Mapa de calor posición 2 ............................................................................ 69
Figura 43. Mapa de calor posición 3 ............................................................................ 70
Figura 44. Pines en estado actual .................................................................................. 70
LISTA DE ANEXOS

pág.
Anexo A. Librería "DHT22.h" ................................................................. 78
Anexo B. Librería "DHT22.c" ................................................................. 78
Anexo C. Librería "UART_18f46k22.h" .............................................. 80
Anexo D. Librería "UART_18f46k22.c" ................................................ 82
Anexo E. Librería "I2C_18f46k22.h" ..................................................... 83
Anexo F. Librería "I2C_18f46k22.c" ..................................................... 84
Anexo G. Librería "LM75A.h" .............................................................. 86
Anexo H. Librería "LM785A.c" .............................................................. 86
Anexo I. Código en C de la adquisición y envío de datos al servidor en PIC18F46K22 .... 88
Anexo J Servidor 1 - Conexión MongoDB y seguridad ......................... 88
Anexo K Servidor 1.1 – Endpoint POST medición ............................ 105
Anexo L Servidor 1.2 - Endpoint GET medición ................................ 105
Anexo M. APP datos recibidos y procesamiento de la información .......... 106
GLOSARIO

HATO: conjunto de animales de ganado mayor o menor.

ESTRO: se llama estro o época de celo al período durante el cual las hembras de la clase mamíferos están receptivas sexualmente.

COTRAL: es una especie de pendiente de plástico que se pone en las orejas del ganado para poder identificarlo.

PROTOCOLO I2C: Protocolo de comunicación desarrollado por Philips que permite la conexión entre múltiples dispositivos en un mismo bus.

PROTOCOLO UART: Protocolo de comunicación serial que permite la conexión de dos dispositivos.

DYNOS: Son los bloques de construcción que alimentan cualquier aplicación de Heroku, están diseñados para ejecutar código basado en un comando especificado por el usuario.

HOMEOTERMOS: Son aquellos animales que mantienen una temperatura corporal relativamente constante, independientemente de la temperatura ambiente.
RESUMEN

Los sistemas de monitoreo son indispensables para las industrias, ya que permiten optimizar tiempo, recursos y materiales en sus procesos. El monitoreo en la producción ganadera está relacionado con el agroecosistema y la agricultura, dos actividades humanas que dependen de variables del medio ambiente como lo son temperatura y humedad relativa, actualmente la recolección de estos valores se realiza de forma manual, utilizando instrumentos especializados definidos según el objetivo final (Noriega Loredo, 2014).

En el presente trabajo se desarrolló un dispositivo que permite monitorear el ganado de forma automatizada teniendo en cuenta posición, temperatura del bovino, humedad relativa y temperatura del ambiente; además, por medio de un sistema IoT (internet of things) se realizó la comunicación entre este dispositivo y una aplicación web, la cual permite visualizar organizadamente la información recibida. Este prototipo se probó en un ambiente controlado logrando satisfactoriamente los resultados esperados.

Palabras claves: IoT, Monitoreo, Bovinos, Alertas, Comunicación.
ABSTRACT

Monitoring systems are indispensable for industries, since they allow the optimization of time, resources, and materials within a process. Monitoring in livestock production is related to the agro-ecosystem and agriculture, two human activities that depend on environmental variables such as temperature and relative humidity. Currently, this process is carried out manually, using specialized instruments defined according to the final objective (Noriega Loredo, 2014).

In the present work, a device was developed that allows the monitoring of the cattle in an automated way, taking into account position, temperature of the cattle, relative humidity and temperature of the environment; in addition, by means of a system IoT (internet of things) the communication between this device and a web application was carried out, which allows to visualize the received information in an organized way. This prototype was tested in a controlled environment achieving satisfactory results.

Keywords: IoT, Monitoring, Bovine, Alerts, Communication
INTRODUCCIÓN

El sector ganadero a lo largo de los años se ha posicionado como uno de los sectores con mayor importancia en la economía nacional, tanto por su aporte en el sector alimenticio, generación de empleos e ingresos en familias rurales (Ruiz Somoyar, 2016); por lo cual a lo largo de los años se ha buscado la manera más eficiente de mejorar el monitoreo de los bovinos, para evitar pérdidas y optimizar la producción.

La implementación de sistemas de monitoreo electrónico se ha convertido en una herramienta de apoyo sumamente importante para la administración de los datos más relevantes sobre el cuidado de un hato de bovinos, logrando así llevar un proceso controlado desde la cría del animal hasta su venta; además, se han establecido normas y leyes para los procesos productivos con el fin de mejorar los procesos productivos y sanitarios (Ruiz Somoyar, 2016).

Los bovinos como todos los mamíferos son animales homeotermos, es decir, son capaces de mantener una temperatura corporal constante a través de un proceso termoregulatorio, el cual involucra el flujo de calor mediante cuatro vías básicas (conducción, convección, radiación y evaporación) (Oyhanart, Insaugarat, & Yumo, 2017), sin embargo, bajo circunstancias muy extremas el animal puede atravesar puntos muy críticos afectando su producción, por lo cual es de vital importancia revisar constantemente su estado.

Por otra parte, las empresas ganaderas también son golpeadas por el fenómeno de inseguridad del país como el robo de los animales, esto se presenta especialmente en las fincas con grandes extensiones de tierras que dificultan un sistema de vigilancia efectivo (Impacto, 2017): actualmente con ayuda del IoT (internet of things), se han logrado desarrollar dispositivos que facilitan a los ganaderos obtener la información que se necesite para identificar patrones en el comportamiento de los bovinos y así actuar rápidamente cuando sea necesario (CONtexto ganadero, 2018).

En el presente trabajo se desarrolló un sistema que permite almacenar en una base de datos la información recolectada por los sensores del dispositivo: temperatura del bovino, posición, humedad y temperatura del ambiente. Estas permiten determinar el estado del hato bovino y como se ve afectado su comportamiento al transcurrir el día; además, con ayuda de mapas de calor se pueden identificar los lugares del terreno más frecuentados por el ganado y así tomar decisiones respecto a zonas de alimentación o modificación de estas, todo esto se visualiza en una aplicación web lo que facilita el análisis de los datos obtenidos, permitiendo al usuario identificar zonas críticas y el estado de salud de sus bovinos.
1. PRELIMINARES

1.1 PLANTEAMIENTO DEL PROBLEMA

Según Heikell (2015), la demanda de los derivados de la ganadería creció con el pasar de los años; debido al aumento de vacas en las granjas no siempre hay tiempo de supervisar cuidadosamente los animales o crear una conexión directa con cada uno. Lo anterior se tradujo en pérdidas, tanto en tiempo como en dinero; por lo tanto, surgió la necesidad de mantener un monitoreo continuo para su salud, a través del seguimiento de los hábitos alimenticios y factores climáticos (Heikell, 2015).

La temperatura del ambiente y la humedad relativa hacen parte de los factores climáticos con mayor influencia para mantener al animal en una zona de confort térmico, lo que afecta directamente su ritmo alimenticio y plan de crecimiento (Oyhanart et al., 2017). Adicionalmente se conoció que las cinco principales causas de muerte en el sector ganadero fueron problemas respiratorios (26%), problemas digestivos (13%), complicaciones durante el parto (12%), clima (12%) y causas desconocidas (11%) (Wildearth Guardians, 2010); por lo anterior se considera de gran importancia reunir datos sobre estos factores y analizar su impacto en la producción ganadera.

1.2 OBJETIVOS DEL PROYECTO

1.2.1 Objetivo General

Desarrollar un sistema IoT para el seguimiento de variables fisiológicas y ambientales que permita generar alertas ante posibles anomalías en el sector agropecuario (caso bovino).

2.2.2 Objetivos Específicos

- Desarrollar un dispositivo para la adecuada medición de las variables de interés.

- Implementar un sistema de almacenamiento y gestión que permita el análisis de la información adquirida.

- Evaluar el desempeño del dispositivo en un ambiente controlado.
1.3 MARCO DE REFERENCIA

1.3.1 Antecedentes

Dentro de la búsqueda de antecedentes se encontraron algunos ejemplos de casos donde utilizaron dispositivos con el fin de proporcionar información esencial en tiempo real de diferentes animales, independiente de su ubicación geográfica. La idea principal fue generar información sin la necesidad de estar en contacto con el animal.

Para el año 2014 se diseñó un sistema de monitoreo ganadero electrónico, dada la necesidad existente de aumentar la competitividad en la producción de leche o carne; con el fin de detectar, transmitir y visualizar información del terreno referente a la ubicación de los animales en tiempo real; este sistema ubicado por medio de un dispositivo en la oreja del animal monitoreaba su posición, movimiento, desplazamiento y/o temperatura ambiental. (Muñoz G., 2014). Este dispositivo no realizaba mediciones sobre variables propias del bovino.

Como segundo acercamiento se presenta una aplicación para mascotas, el cual es un collar inteligente que transmite datos todo el día, a los cuales se puede acceder rápida y fácilmente desde un “smartphone” (Valenzuela, 2015), además de diferentes funciones que le permiten al usuario interactuar con su mascota.

En el año 2016 surgió la necesidad de recolectar datos sobre la frecuencia de alimentación de cada bovino directamente en su entorno productivo, por lo que se diseñó un sistema instalado sobre la oreja del animal; el cual enviaba señales cada que el bovino estaba en el lugar predestinado para su alimentación. (CONTEXTO Ganadero, 2016).

Así mismo, se propuso un sistema para detectar cuando una vaca se encuentra en celo, ya que en muchas ocasiones pasaba desapercibido para los granjeros, provocando pérdidas en diferentes sectores de la ganadería. Este sistema está basado en monitoreo en tiempo real de señales de temperatura y movimiento mediante redes de sensores inalámbricos para evitar pérdidas económicas y aumentar la producción con nuevas crías. (Muñoz Poblete, 2015).

Por otra parte, se implementó un sistema de monitoreo de vacas con el fin de recopilar y analizar datos con respecto a su actividad física y rumia. Estos valores permiten el análisis del estado de salud del animal, la detección de celo e identificación de cada animal. Cada dispositivo transmitía por medio de radiofrecuencia los datos tomados para ser presentados al usuario por medio de una aplicación. (SCR by Allflex, s. f.).

2.3.2 Marco Teórico

a) Bovinos en Colombia

Para el 2019, la ganadería representó el 6% del empleo nacional, el 19% del empleo agropecuario y 6.4% del PIB nacional (Federación Colombiana de Ganaderos, 2019) por eso es necesario hablar sobre la ganadería en Colombia en el desarrollo de este proyecto.
Según el Instituto Colombiano agropecuario (2019), en Colombia para el 2019, habían aproximadamente 27,234,027 de bovinos repartidos en 623,794 predios, ubicados principalmente en Antioquia, Casanare y Córdoba con 11,35%, 7,84% y 7,84% respectivamente, en la Figura 1 se puede observar el número de bovinos por departamento, mientras que en la Figura 2 se encuentra la distribución de los predios bovinos en Colombia.

**POBLACIÓN DE BOVINOS POR DEPARTAMENTO 2019**

<table>
<thead>
<tr>
<th>Departamento</th>
<th>Población de Bovinos 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTIOQUIA</td>
<td>1,796,737</td>
</tr>
<tr>
<td>CASANARE</td>
<td>2,134,723</td>
</tr>
<tr>
<td>CORDOBA</td>
<td>2,134,681</td>
</tr>
<tr>
<td>META</td>
<td>2,045,984</td>
</tr>
<tr>
<td>CAQUETÁ</td>
<td>1,897,555</td>
</tr>
<tr>
<td>SANTANDER</td>
<td>1,617,198</td>
</tr>
<tr>
<td>CUNO (NORTE)</td>
<td>1,484,812</td>
</tr>
<tr>
<td>Cesar</td>
<td>1,482,922</td>
</tr>
<tr>
<td>Magdalena</td>
<td>1,342,115</td>
</tr>
<tr>
<td>Bolívar</td>
<td>1,222,434</td>
</tr>
<tr>
<td>Arauca</td>
<td>1,209,520</td>
</tr>
<tr>
<td>Boyaca</td>
<td>1,181,505</td>
</tr>
<tr>
<td>Sucre</td>
<td>1,103,675</td>
</tr>
<tr>
<td>Tolima</td>
<td>733,944</td>
</tr>
<tr>
<td>Valle</td>
<td>558,101</td>
</tr>
<tr>
<td>Norde Santander</td>
<td>482,233</td>
</tr>
<tr>
<td>Cundinamarca</td>
<td>451,994</td>
</tr>
<tr>
<td>Guaviare</td>
<td>443,633</td>
</tr>
<tr>
<td>Huila</td>
<td>432,587</td>
</tr>
<tr>
<td>Nariño</td>
<td>407,143</td>
</tr>
</tbody>
</table>

*La gráfica muestra los 20 departamentos con el mayor número de bovinos, entre los 12 departamentos restantes suman 1,796,737 bovinos en un rango de 733 a 558.*

**Figura 1. Población de bovinos por departamento 2019 (Instituto Colombiano Agropecuario, 2019)**
Debido a la ubicación geográfica de Colombia y la variedad de sus pisos térmicos, desde el nivel del mar hasta los páramos, es permitido el trabajo con diversas razas de bovinos. Según la Federación Colombiana de Ganaderos (2019) en Colombia para el 2019, el 66.4% de las cabezas de ganado eran hembras, donde se dividían en tres modalidades de negocio, el 59% de las cabezas de ganado eran usadas en la ganadería de carne, el 35% representaba la ganadería lechera y el restante, 35%, eran usados en doble propósito.

La ganadería de carne está compuesta en cría, levante y ceba: la etapa de cría consiste en la producción temprana (hasta los seis (6) meses). La fase de levante radica en llevar a los bovinos a un peso aproximado de 230kg, va desde el séptimo (7) hasta el dieciochoavo (18) mes. Por último, la etapa de ceba busca llevar el ganado a un peso entre 450 a 470 kg, normalmente va desde el mes 19 hasta los 24 o 36 meses. (Fonseca, 2016)

La ganadería de leche se puede realizar por medio de dos sistemas: ordeño manual o lechería especializada, la última, se apoya en desarrollo tecnológico para la realización de la actividad; la ganadería doble propósito parte del principio de utilizar animales cruzados con el objetivo de tener un ejemplar que sirva tanto para carne como para leche. (Fonseca, 2016)
b) Estrés térmico

Hay ocasiones donde los animales sufren lo que se denomina estrés térmico, es un estado donde el animal no es capaz de disipar el calor que su cuerpo genera. Por lo general, en las épocas del año más calurosas es cuando la diferencia en temperatura entre el bovino y el medio ambiente disminuye haciendo que sea más difícil mantener una temperatura corporal cómoda. (Lima Cerqueira, Araújo, Cantalapiedra, Pedernera, & Blanco-Pened, 2014)

El estrés térmico no solo depende de la temperatura del ambiente, se debe tener en cuenta la interacción de varios factores como lo es la humedad, ventilación y radiación solar. Por lo general se usa el índice de temperatura y humedad (ITH) para verificar la condición de estrés que presente el animal; cuando es mayor a 72, el animal empieza a sufrir estrés por calor. (Temple, Bargo, Mainau, Ipharraguerre, & Manteca, 2015)

Según Lima Cerqueira et al. (2014) como consecuencia a el estrés térmico se manifiestan en alteraciones en el comportamiento como lo son las siguientes actitudes:

- Adopción de posturas en extensión, de pie o tumbadas
- Permanecer en zonas frescas, ventiladas y/o con sombra
- Dispersión entre animales
- Contacto con superficies frías
- Menos actividad corporal
- Mayor consumo de agua
- Cambios en los hábitos alimenticios

c) Internet de las cosas (IoT)

Como puede encontrarse en el libro “Internet of Things. Principles and paradigms”, el Internet de las cosas es “una red unificada de objetos inteligentes y seres humanos responsables de operarlos (si es necesario), los cuales son capaces de comunicarse entre ellos universal y omnipresentemente” (Rouse, 2019). A raíz de esto, muchas empresas a nivel mundial iniciaron procesos de implementación de grandes sistemas de comunicación por medio del uso de esta herramienta; una de ella fue la compañía IBM, quien desarrolló una línea de negocio basado en IoT (IBM, 2018).

Algunas de las aplicaciones del Internet de las cosas (IoT, por sus siglas en inglés) a nivel industrial son (Microsoft Azure, 2019):

- Ofrecer mantenimiento predictivo y monitoreo de desempeño para el equipamiento producido por cada empresa.
- Mantener el desempeño de los vehículos.
- Creación de edificios inteligentes.
- Capacitar a los empleados con información basada en datos.
- Monitorear activos médicos.
- Optimizar el uso de recursos, entre otras
d) **Bases de datos**

Es un sistema de cómputo compartido e integrado que guarda datos de usuarios finales y metadatos; los últimos presentan información más detallada y qué relación existente entre ellos (Coronel, Morris, & Rob, 2011). Para entender mejor el funcionamiento y uso de una base de datos es indispensable saber la diferencia entre “datos” e “información”, cuando se habla de “datos” se refiere a lo que se almacena, mientras que “información” es el conjunto organizado de “datos” que son procesados de acuerdo a lo solicitado por el usuario (Llanos Ferraris, 2010).

Según Date (2001) un sistema de bases de datos es un sistema de cómputo para guardar datos, que permite que el usuario pueda generar, recuperar y actualizar registros de datos de interés. Tiene cuatro componentes principales:

- Datos
- Hardware
- Software
- Usuarios.

Además, a finales de 1970 las bases de datos almacenaban una gran cantidad de datos en una estructura poco entendible, se debía saber cuál era el uso que esta iba a tener para su diseño y si se quería algún cambio era un proceso lento y costoso. Hoy en día gracias a los avances tecnológicos es más fácil modificaciones sin un costo elevado. (Coronel et al., 2011)

e) **Aplicaciones web**

Una aplicación web es una herramienta informática accesible desde cualquier navegador, ya sea a través de internet o una red local, no necesita ser instalada en el pc, ya que los datos o archivos utilizados están almacenados en una red o en la nube, de igual forma se encuentran estrechamente relacionadas con el almacenamiento de datos ya que toda la información requerida está en servidores web, que además de alojar la información, la envían a los dispositivos cuando es requerida. (Gibb, 2016)

Este tipo de aplicaciones tienen ventajas como (Flores, 2019):

- Son muy adaptables y fáciles de actualizar.
- Al estar en la nube es accesible para cualquier ordenador o dispositivo que tenga acceso a internet.
- Es multiplataforma y multidispositivo.
- El dispositivo no se ve afectado por el peso de memoria de la aplicación.

f) **Interfaz de usuario**

Como el usuario final del sistema puede estar por fuera del área de ingeniería, es necesario desarrollar una interfaz que sea intuitiva, llamativa y funcional para lograr una mayor difusión. Como enuncian Wigdor y Wixton en su capítulo sobre “The natural user interface” (2011), una interfaz de usuario debe:
- Crear una experiencia que, para usuarios expertos se sienta como una extensión de su cuerpo.
- Crear una experiencia que se sienta tan natural para un usuario novato como para uno experimentado.
- Crear una experiencia que sea auténtica a su medio.
- Construir una interfaz de usuario que considere el contexto, incluyendo las metáforas adecuadas, indicadores visuales, realimentación y métodos de ingreso/salida de datos para el contexto.
- Evitar caer en la trampa de copiar los paradigmas de una interfaz de usuario ya existente. (Wigdor & Wixon, 2011)

Otros aspectos a tener en cuenta a la hora de crear una interfaz de usuario adecuada pueden encontrarse en páginas y artículos como en Principles of User Interface Design. Allí Josua Porter (s. f.) menciona que el diseño cambiará dependiendo del tipo de usuario o medio, por lo que es necesario realizar una visita de campo en la cual se identifiquen cuáles son las necesidades del usuario y las interfaces a las que normalmente tiene acceso para poder realizar un montaje adecuado e intuitivo.

g) Sistema de Posicionamiento Global (GPS)

Es un sistema de navegación satelital, fue desarrollado por el Departamento de Defensa de Estados Unidos en la década de 1970, inicialmente fue destinado para uso militar. Tiempo después este sistema se puso a disposición de las personas del común. (El-Rabbany, 2002)

El Sistema de posicionamiento Global más conocido como GPS, por sus siglas en inglés, trabaja mediante el uso de satélites que permite ubicar en la tierra un receptor por medio de técnicas de triangulación (Toloza, 2013), en consecuencia se puede tener la posición continua e información aproximada en cualquier parte del globo terráqueo sin importar las condiciones del ambiente (El-Rabbany, 2002).

h) Sistema global para las comunicaciones móviles (GSM)

La red GSM (sistema global para las comunicaciones móviles) fue la más usada en Europa al principio del siglo XXI, también es conocida como el estándar de segunda generación 2G. Esta red implementó el cambio de comunicaciones análogas a comunicaciones totalmente digitales. (Villagómez, 2017)

GSM fue desarrollada para dar una mayor conectividad entre países. Al principio solo se trasmitía voz, a medida que fue desarrollando se le agregó un servicio de mensajes cortos e intercambio de datos con una velocidad máxima de 9.6kbps. (Pachón de la Cruz, 2004)

Una de las características principales de esta red fue el tener dos bandas de frecuencia para la transmisión de datos, en Europa funcionaba entre 900 y 1800 MHz, mientras en América a 1900MHz. Por esta razón no todos los móviles servían a nivel mundial, a menos que el dispositivo móvil reconociera las dos frecuencias, en este caso, tanto en Europa como en América funcionaba. (Universidad Internacional de Valencia, 2018)
i) Sistema global para las comunicaciones móviles (GPRS)

El GPRS o Servicio general de paquetes vía radio, es el paso intermedio entre redes 2G y 3G, fue introducida como una mejora en la conectividad de las redes existentes de GSM. Se desarrolló para poder tener acceso a protocolos IP y de esta manera integrar los servicios móviles con la infraestructura de internet. (Pachón de la Cruz, 2004)

Según Llumiquinga Bustos (2018), el sistema GPRS hace una modificación en la manera de transmisión y como consecuencia fue aumentar la velocidad entre 9,6 y 171 Kbps. El objetivo de esta red es adaptar la red GSM con banda ancha, se introdujeron la comunicación por paquetes y mantener una comunicación continua para disminuir el tiempo de conexión y aumentar la transmisión de datos.
METODOLOGÍA

Para el desarrollo metodológico de este Proyecto de grado se utilizará la metodología de Ulrich para desarrollo de producto (Ulrich & Eppinger, 2012).

Objetivo 1: **Desarrollar un dispositivo para la adecuada medición de las variables de interés.**

   Actividad 1: Búsqueda de requerimientos para el diseño del dispositivo.

   Actividad 2: Definición de componentes y herramientas de desarrollo.

   Actividad 3: Planteamiento inicial y pruebas unitarias de la estructura de hardware del sistema.

   Actividad 4: Construcción y pruebas del dispositivo.

Objetivo 2: **Implementar un sistema de almacenamiento y gestión que permita el análisis de la información adquirida.**

   Actividad 1: Búsqueda de requerimientos para el diseño del sistema de almacenamiento y gestión.

   Actividad 2: Definición del software y servidor por el cual se implementará el sistema de almacenamiento.

   Actividad 3: Planteamiento inicial, simulaciones y pruebas iniciales de la estructura de software del sistema.

   Actividad 4: Implementación del método de almacenamiento y gestión de información.

   Actividad 5: Construcción y prueba de la plataforma.

Objetivo 3: **Evaluar el desempeño del dispositivo en un ambiente controlado.**

   Actividad 1:Determinar protocolo de pruebas de funcionamiento del sistema completo en un ambiente controlado.

   Actividad 2: Implementar protocolo de prueba.

   Actividad 3: Desarrollo de las correcciones necesarias
2. DESARROLLO DEL PROYECTO

2.1 IDENTIFICACIÓN DE REQUERIMIENTOS

Para el desarrollo de este trabajo de grado se recopiló información a partir de diferentes entrevistas con expertos acerca del tema; ellos lograron orientarnos y nos proporcionaron la información que se plasmó a continuación. Con dicha información se definieron: los requerimientos para el funcionamiento del sistema y las variables a medir.

Para realizar un monitoreo adecuado en el ganado bovino fue indispensable conocer el comportamiento de las vacas dependiendo del entorno donde se encuentren. Dado esto, se definió que las variables para tener en cuenta para el sistema fueron:

- Temperatura del ambiente
- Humedad relativa del ambiente
- Temperatura corporal
- Ubicación geográfica
- Movimiento

Las mediciones de las variables debían ser almacenadas en la nube para posteriormente acceder a ellas y actualizar la interfaz gráfica del usuario.

Luego de obtener la medición de las variables se analizó cómo los cambios afectaron el comportamiento del bovino con respecto a los lugares mayormente frecuentados. De acuerdo con estos datos se arrojaron conclusiones que ayudaron a mejorar el proceso de cuidado en los bovinos, así como determinar las características de los lugares donde prefieren estar las reses.

Para lograr agrupar adecuadamente las necesidades del proyecto se utilizó la House of Quality (HoQ). Herramienta que se usó para definir y satisfacer las necesidades del usuario mediante el proceso de desarrollo de producto, nos ayudó a identificar las características de ingeniería (enumeradas de la “a” hasta la “o”) y los requerimientos de usuario (enumerados del 1 al 16) presentados en las Tabla 1 y Tabla 2 (Dieter & Schmidt, 2009). La relación que hay entre ellos se presenta en la Figura 3.

**Tabla 1.**
*Requerimientos de usuario.*

<table>
<thead>
<tr>
<th>Número</th>
<th>Requerimientos de usuario</th>
<th>Importancia</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Batería duradera</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Resistente al clima</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Ubicación</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Estrés térmico</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Ganancia de peso por día</td>
<td>5</td>
</tr>
<tr>
<td>Letra</td>
<td>Características de ingeniería</td>
<td>Unidad</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>a</td>
<td>Alarma</td>
<td>--</td>
</tr>
<tr>
<td>b</td>
<td>Tamaño</td>
<td>mm</td>
</tr>
<tr>
<td>c</td>
<td>Procesador</td>
<td>--</td>
</tr>
<tr>
<td>d</td>
<td>Alimentación</td>
<td>DCV</td>
</tr>
<tr>
<td>e</td>
<td>Comunicación</td>
<td>--</td>
</tr>
<tr>
<td>f</td>
<td>Movimiento</td>
<td>mm</td>
</tr>
<tr>
<td>g</td>
<td>Envío de datos</td>
<td>--</td>
</tr>
<tr>
<td>h</td>
<td>Ubicación</td>
<td>º</td>
</tr>
<tr>
<td>i</td>
<td>Temperatura</td>
<td>ºC</td>
</tr>
<tr>
<td>j</td>
<td>Humedad</td>
<td>%RH</td>
</tr>
<tr>
<td>k</td>
<td>Alcance</td>
<td>m</td>
</tr>
<tr>
<td>l</td>
<td>Bajo consumo</td>
<td>A</td>
</tr>
<tr>
<td>m</td>
<td>Velocidad de trasmisión</td>
<td>bps</td>
</tr>
<tr>
<td>n</td>
<td>Batería</td>
<td>Ah</td>
</tr>
<tr>
<td>o</td>
<td>Precisión</td>
<td>--</td>
</tr>
</tbody>
</table>

**Tabla 2.**

Características de ingeniería
Para obtener la información contenida en la Figura 3 se tuvieron en cuenta los siguientes pasos:

- **Asignar un valor de importancia** entre 1 y 5 a los requerimientos de usuario, siendo 1 poco importante y 5 muy importante.

- **Definir una dirección de mejora** a las características de ingeniería:
  - "^": más es mejor,
  - "v": menos es mejor
  - "N/A": no representa ninguna mejora en el proyecto.

Para ilustrar: entre menor el tamaño, mejor para el proyecto.

- **Dar una correlación** entre características de ingeniería, según la siguiente escala:
  - "--": relación fuerte negativa
  - "-": relación negativa

*Figura 3. House of Quality*
“-”: no hay relación
“+”: relación positiva
“++”: relación fuerte positiva.

Por ejemplo; entre el procesador y la velocidad de trasmisión existía una correlación “fuerte positiva”; esto significaba que, si una mejoraba, la otra también lo hacía.

- Establecer un nivel de relación entre requerimientos de usuario y características de ingeniería:
  - 1: relación débil
  - 3: relación media
  - 9: relación fuerte.

- Teniendo el nivel de relación y la importancia se calcularon las importancias absolutas y relativas para obtener un ranking de las características de ingeniería.

Como resultado de la House of Quality (HoQ - Figura 3) se concluyó:

Las 5 características de ingeniería más importantes fueron: precisión, batería, procesador, alimentación y ubicación; por lo cual, a lo largo del desarrollo del proyecto, se le dio una mayor prioridad. En conjunto, cada una de estas aportó para que el producto final fuera funcional y agrupara todas las necesidades requeridas. Por otra parte, se pudo concluir que las características que presentan menor peso para el proyecto son el alcance, tamaño y velocidad de transmisión.

2.2 DISEÑO Y SELECCIÓN DE CONCEPTO

Con la caja negra (Figura 4) se pretendió determinar la interacción del sistema con el medio que lo rodea, es decir, identificar qué grupo de señales componen las entradas y cuáles las salidas, sin importar qué proceso se lleve a cabo para su funcionamiento.
Figura 4. Caja Negra basada en el análisis morfológico

En la caja transparente (Figura 5) se detallaron los pasos a seguir y cómo estos se debían conectar entre sí para lograr un correcto funcionamiento del sistema.

Figura 5. Caja Transparente basada en el análisis morfológico
1.2.1 Registro de soluciones

A partir de las funciones descritas en la caja transparente (Figura 5) se realizó una búsqueda de posibles componentes del sistema, para cada uno se presentó una breve descripción, ventajas y desventajas para tener en cuenta en la decisión, es importante aclarar que estas ventajas y desventajas fueron seleccionadas teniendo en cuenta las prioridades y necesidades de nuestro proyecto más no por las especificaciones y cualidades de los componentes mencionados.

**Toma de datos de los sensores**

Los criterios para la selección de los componentes fueron: no comprometer la integridad física del aminal, tener un bajo costo, ser pequeños y usar protocolos de comunicación ampliamente difundidos en la industria.

Según la información encontrada en la *House of Quality, Caja negra y Caja trasparente*; se tuvieron en cuenta las siguientes variables para la selección de los sensores (Tabla 3): temperatura ambiental, humedad relativa, temperatura corporal y ubicación por medio de GPS.

**Tabla 3.**

Registro de soluciones – Toma de datos – Sensores

<table>
<thead>
<tr>
<th>Variable</th>
<th>Opciones de solución</th>
<th>Descripción</th>
<th>Ventajas</th>
<th>Desventajas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperatura y humedad del ambiente</td>
<td>Sensor de temperatura y humedad HDC1008</td>
<td>• Voltaje de funcionamiento: 3V ~ 5V. &lt;br&gt; • Rango de temperatura: -20 ºC ~ 85 ºC. &lt;br&gt; • Resolución de la medición en temperatura: ±0.2 ºC. &lt;br&gt; • Interfaz: I2C. &lt;br&gt; • Rango de medición de humedad: 10% ~ 80%. &lt;br&gt; • Resolución humedad: ±4%.</td>
<td>• Tamaño pequeño. &lt;br&gt; • Alto grado de precisión. &lt;br&gt; • Poco procesamiento de datos. &lt;br&gt; • Precio bajo.</td>
<td>• Susceptible a ruido térmico. &lt;br&gt; • Alto rango de temperatura.</td>
</tr>
<tr>
<td>Temperatura y humedad del ambiente</td>
<td>Módulo Sensor de Temperatura y Humedad</td>
<td>• Voltaje de funcionamiento 3.3V ~ 5V.</td>
<td>• Tamaño pequeño. &lt;br&gt; • Bajo consumo.</td>
<td>• Mayor procesamiento de datos.</td>
</tr>
</tbody>
</table>
AM2302/DHT22
- Corriente de medición: 500 µA
- Rango de temperatura: -40 ºC ~ 80 ºC.
- Error de medición de temperatura: ± 2 ºC.
- Resolución de la medición de temperatura: 0.1 ºC.
- Resolución de la medición de humedad: 0.1%RH.
- Rango de humedad: 0-99.9% RH.
- Error de medición de Humedad: ± 5% RH.
- Fácil de usar
- Precio bajo
- Alto rango de temperatura

Temperatura y humedad del ambiente
- Módulo sensor de temperatura y humedad DHT11
- Voltaje de funcionamiento: 3.3V ~ 5V.
- Rango de medición de humedad: 20% ~ 90%.
- Rango de temperatura: 0 ºC ~ 60 ºC.
- Comunicación: One-wire
- Bajo rango de temperatura
- Tamaño pequeño
- Precio bajo
- Fácil de usar
- Alto procesoamiento de datos

Ubicación
- Tarjeta GPS L70
- Voltaje de alimentación: 3V ~ 5V
- Consumo en adquisición: 18 mA
- Consumo en
- Tamaño pequeño
- Bajo consumo
- Precio alto
- Alto procesoamiento de datos
- Tamaño pequeño
seguimiento: 12 mA
- Modo sleep para reducir el consumo de energía

Ubicación
- Ubicación GPS NEO-6M
- Voltaje de alimentación 3V ~ 5V
- Comunicación UART
- Consumo: 67mA
- Alta documentación
- Bajo Precio
- Alto procesamiento de datos
- Sensible al ruido

Temperatura
- Temperatura Sensor de temperatura DS18B20
- Rango de temperatura -55 ºC ~ 120ºC
- Voltaje de alimentación: 3V ~ 5.5V bits
- Precisión: ±0.5 ºC
- Protocolo de comunicación one-wire
- Resolución de 9 y 12 bits
- Tamaño pequeño
- Bajo Costo
- Bajo procesamiento de datos
- Alto rango de medición
- Sensible al ruido electromagnético

Temperatura
- Temperatura Sensor de temperatura DS18B20
- Rango de temperatura -55 ºC ~ 120ºC.
- Voltaje de alimentación: 3V ~ 5.5V
- Precisión: ±0.5 ºC
- Protocolo de comunicación one-wire
- Resolución de 9 y 12 bits
- Bajo costo
- Tamaño pequeño
- Bajo grado de error
- Altos
- Alto rango de temperatura
- Sensible al ruido
- Alto procesamiento de datos.
### Temperatura

<table>
<thead>
<tr>
<th>Sensor Analógico de temperatura LM35</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Voltaje de funcionamiento 5V</td>
</tr>
<tr>
<td>• Rango de temperatura 0 ºC ~ 60 ºC</td>
</tr>
<tr>
<td>• Consumo de corriente 60μA</td>
</tr>
<tr>
<td>• Lectura análoga</td>
</tr>
<tr>
<td>• Bajo costo</td>
</tr>
<tr>
<td>• Sensor con medición lineal</td>
</tr>
<tr>
<td>• Tamaño pequeño</td>
</tr>
<tr>
<td>• Alto procesamiento de datos</td>
</tr>
<tr>
<td>• Sensible al ruido</td>
</tr>
<tr>
<td>• Bajo grado de precisión</td>
</tr>
</tbody>
</table>

### Comunicación entre dispositivos

Para la comunicación entre dispositivos se tuvo en cuenta: la capacidad para recibir y enviar información, el consumo de energía y su facilidad de crear o conectarse a una red para establecer comunicación entre ellos. La Tabla 4 expone las posibles soluciones para lo anterior.

**Tabla 4. Registro de soluciones – Comunicación**

<table>
<thead>
<tr>
<th>Alternativa de solución</th>
<th>Descripción</th>
<th>Ventajas</th>
<th>Desventajas</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIFI ESP8266</td>
<td>• Voltaje de alimentación 5V ~ 9V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Conversor ADC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Comunicación: one-wire, I2C y SPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Modo sleep</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Bajo precio</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Tamaño pequeño</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Baja protección de datos</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Alto consumo de energía</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Módulo Bluetooth HC05</td>
<td>• Voltaje de alimentación 3.3V ~ 6V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Longitud de alcance 10m</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Protocolo UART RS 232 serial</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Modo sleep para reducir el consumo de energía</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Tamaño pequeño</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Precio bajo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Bajo consumo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• El rango de alcance es bajo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GSM/GPS M95</td>
<td>• Comunicación MSM, voz y datos</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Voltaje de alimentación 2.7V a 5.5V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Protocolo TCP/IP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Protocolo HTTP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Comunicación por datos</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Tamaño pequeño</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Alto consumo de batería</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Precio alto</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Microcontrolador**

El microcontrolador (Tabla 5) estaba encargado del manejo de; sensores, módulos, protocolos de comunicación, además, tiempos de funcionamiento. Se tuvo en cuenta la capacidad de almacenamiento y procesamiento, número de periféricos necesarios (puertos UART, I2C, y pines digitales), tamaño reducido, bajo costo, así mismo, su fácil manejo.

**Tabla 5.**
*Registro de soluciones – Microprocesador*

<table>
<thead>
<tr>
<th>Alternativa de solución</th>
<th>Descripción</th>
<th>Ventajas</th>
<th>Desventajas</th>
</tr>
</thead>
</table>
| PIC 18F2550             | • Voltaje de alimentación 2V ~ 5.5V  
• Memoria del programa 32KB  
• SRAM: 2048 bytes  
• Protocolo de comunicación: 1 USART, 1 MSSP | • Lenguaje de programación C  
• Alto rendimiento  
• Amplias funcionalidades  
• Amplia documentación  
• Precio bajo | • Baja capacidad de memoria  
• Pocos USART por hardware |
| FRDM-KL25Z              | • Voltaje de alimentación 2V ~ 5.5V  
• Memoria del programa 128KB  
• SRAM: 16KB  
• Protocolo de comunicación: 3 USART, 2 SPI, 2 I2C | • Alto rendimiento  
• Alta capacidad de procesamiento  
• Amplias funcionalidades  
• Soluciones amplias de manejo de datos | • Alto costo  
• Alto consumo de tiempo  
• Tamaño grande |
| PIC 18F46K22            | • Voltaje de alimentación 1.8V ~ 5.5V  
• Memoria del programa 64KB  
• SRAM: 3896 bytes  
• Protocolo de comunicación: 2 USART, 2 MSSP | • Lenguaje de programación C  
• Alto rendimiento  
• 2 módulos UART  
• Amplia documentación  
• Precio bajo | • Alto procesamiento de datos |
**Almacenamiento de datos en la nube**

Para el almacenamiento de datos fue importante tener en cuenta la opción con mayor capacidad de memoria, facilidad de uso y alta seguridad. En la tabla 6 se pueden ver las opciones con sus definiciones, ventajas y desventajas.

**Tabla 6. Registro de soluciones – Servidores**

<table>
<thead>
<tr>
<th>Alternativa de solución</th>
<th>Definición</th>
<th>Ventajas</th>
<th>Desventajas</th>
</tr>
</thead>
</table>
| MongoDB                 | Base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado. | • Fácil de usar  
• Alto rendimiento  
• Amplias funcionalidades  
• Soporte para web, Android y iOS. | • Plan gratuito limitado  
• Diferentes enfoques de las herramientas |
| Amazon web services (AWS) | Plataforma que proporciona una variedad de servicios de infraestructura como almacenamiento, redes, bases de datos, servicios módulos, entre otros, los cuales permiten proyectos grandes. (Amazon, s. f.) | • Flexible  
• Seguro  
• Escalabilidad  
• Alta capacidad de procesamiento  
• Soluciones amplias de manejo de datos | • Alto costo  
• Ofrece herramientas que no se van a usar |
| Servidor local          | Creación de la infraestructura del servidor (creación del hardware) que se encargue de correr un programa controlando conexiones y operaciones que se requieran según la necesidad. | • A largo plazo no genera costos adicionales | • Alto procesamiento de datos  
• Alto consumo de tiempo |
2.2.2 Matriz morfológica

Con las posibles soluciones mostradas en las Tabla 3, Tabla 4, Tabla 5 y Tabla 6, se realizó la matriz morfológica (Tabla 7), la cual contiene tres conceptos mostrados en las columnas A, B y C.

Tabla 7. 
*Matriz morfológica*

<table>
<thead>
<tr>
<th>CONCEPTOS SOLUCIÓN</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOMA DE DATOS – Temperatura y humedad del ambiente</td>
<td><img src="image1.png" alt="Imagen A" /></td>
<td><img src="image2.png" alt="Imagen B" /></td>
<td><img src="image3.png" alt="Imagen C" /></td>
</tr>
<tr>
<td>TOMA DE DATOS - Ubicación</td>
<td><img src="image4.png" alt="Imagen A" /></td>
<td><img src="image5.png" alt="Imagen B" /></td>
<td><img src="image6.png" alt="Imagen C" /></td>
</tr>
<tr>
<td>TOMA DE DATOS - Temperatura</td>
<td><img src="image7.png" alt="Imagen A" /></td>
<td><img src="image8.png" alt="Imagen B" /></td>
<td><img src="image9.png" alt="Imagen C" /></td>
</tr>
<tr>
<td>Comunicación</td>
<td><img src="image10.png" alt="Imagen A" /></td>
<td><img src="image11.png" alt="Imagen B" /></td>
<td><img src="image12.png" alt="Imagen C" /></td>
</tr>
<tr>
<td>Microprocesador</td>
<td><img src="image13.png" alt="Imagen A" /></td>
<td><img src="image14.png" alt="Imagen B" /></td>
<td><img src="image15.png" alt="Imagen C" /></td>
</tr>
</tbody>
</table>
A partir de los mostrado en la matriz morfológica (Tabla 7) se procedió a explicar cada concepto.

**Concepto A**

Para la toma de la temperatura y humedad del ambiente se usaría el sensor HDC1008, el cual es pequeño, tiene una alta precisión y un precio bajo; el GPS NEO-6M de tamaño pequeño, con el modo de *sleep*, para disminuir el consumo energético; para la medición de temperatura del ganado se implementaría el sensor de temperatura LM35. Todos estarían conectados al microprocesador FRDM-KL25Z.

Para la comunicación entre dispositivos se usaría el módulo de Bluetooth HC05, el cual es pequeño, tiene un bajo consumo y una cantidad finita de dispositivos que se pueden conectar en una misma red, además de necesitar una conexión a internet externa.

Para el procesamiento de datos se definiría un servidor local desarrollado exclusivamente para este proyecto, en consecuencia, esta opción poseería un alto procesamiento de datos, además se planearía enviar los datos a la nube y procesarlos en el mismo servidor para así mostrar los datos en la interfaz de usuario.

**Concepto B**

En esta posible solución se utilizaría el microprocesador PIC18F46K22 con un sensor DHT22, sensor de temperatura en el ambiente y humedad relativa, con un tamaño pequeño, bajo precio y con un rango de temperatura bajo, para la ubicación se usaría el Tarjeta GPS L70, este requiere un bajo procesamiento de datos; para la toma de datos de la temperatura del animal se definiría el sensor DS18B20, usa protocolo de comunicación *one-wire*.

Para la comunicación con la nube se usaría un módulo GSM/GPRS M95, este es pequeño, con precio bajo, además de no necesitar un alto procesamiento de datos, funciona por medio de protocolo UART para enviar los datos tomados al servidor para ser procesados y mostrados al usuario por medio de la plataforma MongoDB.

**Concepto C**

Por último, este concepto trabajaría con el sensor DHT11 para medir la temperatura y la humedad del ambiente, tiene un tamaño reducido y un nivel de resolución bajo. La temperatura del ganado se tomaría con el sensor analógico LM35, este posee un alto
rango de temperatura y un tamaño pequeño. Los anteriores estarían controlados al PIC18F2550.

Para la comunicación entre dispositivos se usaría el módulo WIFI ESP8266, tiene la característica de tener bajo consumo, con este mismo se tomarían los datos de la variable de la ubicación, este módulo tiene un tamaño mediano y un precio bajo, para el procesamiento de datos se definiría usar el servidor Amazon web services (AWS), el cual tiene un amplio catálogo de soluciones por un precio alto.

3.2.3 Matriz de selección

Para tomar la decisión de cuál de los conceptos explicados anteriormente sería la mejor opción para nuestro proyecto, se realizó la matriz de selección presentada en la Tabla 8 (Calderón, 2017), en la cual se evaluaron cada uno de los conceptos por medio de un criterio de ponderación, el cual nos permitió juzgar cada uno de estos decidiendo variables primordiales y peso en el proyecto, para poder comparar las soluciones y concluir cual es la más adecuada.

Decidimos implementar la matriz de selección ya que esta nos permite escoger la mejor opción para nuestro proyecto, reduciendo y descartando por medio de valores entre todo el conjunto de soluciones agrupadas.

Tabla 8. Matriz de selección

<table>
<thead>
<tr>
<th>Criterio de decisión</th>
<th>Concepto A</th>
<th>Concepto B</th>
<th>Concepto C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Costo</td>
<td>15%</td>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>Precisión</td>
<td>10%</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>Batería duradera</td>
<td>15%</td>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>Estrés térmico</td>
<td>40%</td>
<td>4</td>
<td>1.6</td>
</tr>
<tr>
<td>Ubicación</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
</tr>
<tr>
<td>Puntaje</td>
<td></td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>Posición</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¿Desarrollar?</td>
<td>No</td>
<td>Si</td>
<td>No</td>
</tr>
</tbody>
</table>

Teniendo en cuenta la suma de las ponderaciones, el concepto seleccionado para nuestro proyecto fue el B, el cual incluyó un sensor para la temperatura y humedad del ambiente el DHT22, la tarjeta GPS L70, sensor de temperatura corporal LM75A y el módulo GSM/GPRS M95.

2.3 DISEÑO DE DETALLE

Una vez definido y seleccionado el concepto B, se prosiguió con el desarrollo electrónico para la toma y envío de datos al servidor. En las siguientes líneas se presentará el
desarrollo del software que permite el manejo de los sensores y el módulo de comunicación GPRS.

1.3.1 Diseño electrónico

Cabe resaltar que cada sensor tiene protocolos diferentes, por lo tanto, conexiones distintas, el DHT22 usa protocolo one-wire, el LM75A usa protocolo I2C, y los módulos M95 y L70 se comunican por medio de protocolo UART.

En la Figura 8 se muestran las conexiones mínimas para el funcionamiento del PIC, con un cristal de 16 MHz, un botón para el reinicio y un led que ayuda a identificar si este se encuentra energizado. Para el sensor DHT22 se definió el pin RC0, fue necesaria una resistencia pull-up de 10kΩ. Debido a la emergencia sanitaria generada por la COVID-19, fue necesario el cambio del sensor DS18B20 por la tarjeta basada en el sensor LM75A, en el momento de la realización del proyecto, el sensor seleccionado anteriormente no se encontraba en el mercado. El sensor de temperatura LM75A tiene un interfaz I2C, con una resolución de 9 a 12 bits y cuenta con modo sleep. Dicha tarjeta incluye las resistencias pull-up necesarias en el protocolo I2C. Se conectó a los puertos RC3 y RC4 correspondientes a SCL1 y SDA1. La Conexión de estos dos sensores se muestra en la Figura 9. El PIC usado posee dos puertos UART, el módulo GSM/GPRS M95 conectado al UART_1, mientras que el L70 al UART_2.

A partir de lo anteriormente mencionado, se diseñó un circuito para las pruebas de concepto y el desarrollo del proyecto (Figura 6 y Figura 7), se usó un PIC con empaquetado E/P, un sensor DHT22, un módulo LM75A, un módulo L70 para XBEE, una tarjeta M95, y un regulador de voltaje TLV2217 de 3.3V.

Figura 6. Circuito de pruebas unitarias y desarrollo
Una vez terminada la programación de los componentes, explicada en la siguiente sección, se procedió al diseño de las conexiones de los sensores y módulos a partir de las hojas de datos de cada uno. Para disminuir el tamaño del dispositivo, se usaron componentes de montaje superficial (SMD - Surface Mounting Device). Para el diseño de la PCB fueron utilizadas resistencias, capacitores e inductores con encapsulados 0805 y leds 1206.

En la Figura 8 es mostrado el circuito básico de funcionamiento para el PIC18f46k22-I/PT, como oscilador externo un cristal de 16MHz serie CSM-7 y sus respectivos capacitores de 22pF. En el pin de alimentación se adicionó un capacitor de desacople para evitar picos en la señal de entrada. En el pin 18 se ubicó un botón para el reinicio del programa. Por último, un led para indicar si hay energía en la tarjeta.
El pin Data del sensor DHT22 estaba conectado a una resistencia pull-up de 5.1kΩ (AOSONG, 2010) y al pin RC0 del PIC. Mientras tanto el sensor LM75A fueron conectados directamente a los puertos SCL1 y SDA1 correspondientes a los pines RC3 y RC4, como se mencionó anteriormente esta tarjeta ya tenía integradas las dos resistencias pull-up necesarias en el protocolo I2C. Conexiones mostradas en la Figura 9.

Para comenzar con la conexión del módulo L70 (Figura 10), se tenía un capacitor de desacople ubicado en el pin de alimentación VDD. Los pines correspondientes a la comunicación UART (RXD1 y TXD1) conectados a los puertos RD6 y RD7 del PIC, correspondientes a TX y RX del UART_2. Adicionalmente se agregaron 2 leds para indicar la trasmisión y la recepción de datos controlados desde los pines RB4 y RB3 del PIC.

El módulo L70 (Figura 10) cuenta con una referencia alimentación (VCC_RF) para una antena activa, esta puede variar entre 2.8 y 4.3 V, usualmente es un valor de 3.3 V; la
resistencia de 10Ω fue usada para proteger el circuito de cualquier corto que pueda generar entre la antena y la tierra del sistema y por último el inductor de 47nH fue usado para evitar la filtración de la señal de la antena (pin RF_IN) en VCC_RF además de enrutar el suministro de polarización a la antena. (Quectel, 2013a)

En la Figura 11 se muestran las conexiones finales del módulo M95. En los pines “PWRKEY” y “EMER_OFF” se conectaron dos botones, uno para el inicio y otro para el paro de emergencia. Los pines correspondientes al UART (RXD y TXD), se unieron a los pines RC6 y RC7, correspondientes al UART_1 en el PIC. Al igual que el L70 hay dos leds que permitieron identificar si se está trasmitiendo información entre el módulo m95 y el PIC, estos estaban ubicados en los pines RB1 y RB2.

El módulo tiene incorporados dos señales usualmente usadas para indicar si el módulo este encendido, pin “STATUS”, y el estado de la conexión con la red móvil, pin “NETLIGHT”, la frecuencia del esta última señal indicaba la conexión con la red celular. Las conexiones mostradas en la Figura 11 de estas dos señales son dadas por el fabricante en su hoja de datos (Quectel, 2013b).

Para la conexión con la red celular se necesitaba una antena y una SIM. Se ubicó una antena celular 4 bandas conectada directamente al pin RF_ANT del módulo. Por otro lado, las conexiones entre la SIM y el M95 fueron tomadas de la hoja de datos dada por el fabricante (Quectel, 2013b). Actualmente en el mercado existen dos tipos de SIM, dependiendo el tipo de tecnología que se use, por esta razón el módulo M95 tiene integrado un regulador para la alimentación de la SIM capaz de reconocer el tipo de tecnología y seleccionar entre 1.8V y 3.0V. Usa capacitores de desacoplo, con un valor de 33pF, en las líneas de alimentación (SIM_VDD), señal de reset (SIM_RST), señal de reloj (SIM_CLK) y transmisión de datos (SIM_DATA), adicionalmente se usó un capacitor de desacoplo con un valor de 0.1µF entre las líneas de alimentación y tierra de la SIM. Para evitar interferencias electromagnéticas y posibles cortos se agregaron resistencias de 22Ω en las líneas de reloj, datos y reset.
Figura 11. Conexión módulo M95

Debido a la naturaleza del dispositivo fue obligatorio el tener una batería para la alimentación del circuito, se estableció el uso de una batería de litio recargable de 3.7v y 1000mA.

En la Figura 12 se muestra el circuito de carga de la batería. Como punto de partida se tomó el circuito de SparkFun “LiPo Charger Basic - Micro-USB” (SparkFun, 2011), este consiste en un conector USB tipo B, una celda de carga para baterías LiPo y un conector para la batería. Se adicionó un switch para prender y apagar la tarjeta.

Para proteger todo el circuito de picos altos de voltaje y prolongar la duración de la batería fue necesario un regulador de voltaje; gracias a que todos los componentes usados funcionaban a 3.3V, el regulador de voltaje usado fue de 3.3V con una corriente de máxima de 1 A, la conexión de este regulador fue tomada de la hoja de datos del mismo (Texas Instruments, 2015).
Para poder verificar los mensajes trasmitidos en los dos puertos seriales se agregaron 3 pines por cada uno para poder leerlos en cualquier momento desde un puerto en el computador. Para finalizar con el diseño de los esquemáticos, se agregaron otros 5 pines para la conexión con el depurador y programador PICkit 3 (Figura 13).

El diseño de la PCB se realizó por medio del software Altium Designer versión 20.2.5. Antes de iniciar se debe tener claro las referencias comerciales de los componentes para a partir de las dimensiones crear o buscar los footprint. Las huellas o footprint son la representación de los componentes físicos en la superficie de PCB, eventualmente serán soldados directamente a estas marcas sobre la placa del circuito impreso. Actualmente existen varias bases de datos para la busca de estas huellas, incluso los mismos fabricantes ofrecen esta información, para este desarrollo se usaron las bases de datos de Altium, SnapEDA y Microship.

Después de tener los esquemáticos y los footprint de los componentes, se continuó con la ubicación de los componentes dependiendo de sus especificaciones, por ejemplo, el módulo L70 en sus conexiones (Figura 10) tenía un capacitor de desacople, este debe estar lo más cercano posible al pin donde estaba conectado para cumplir de manera adecuada su función.

Los módulos L70 y M95 usaron antenas, para evitar que estas señales generen interferencia a los demás componentes deben estar encapsuladas con GND (tierra), es decir caminos desde el conector de la antena (J4 y J2) hasta el pin 11 del L70 y pin 39 del M95 deben estar rodeados de la señal de tierra, además de estar alejados de otros componentes ya sea en la capa superior o en la inferior de la tarjeta.

Para tener un mejor orden en la PCB, se ubicaron los componentes por zonas, en la capa superior (Figura 14) están ubicados el PIC, el GPS L70, el módulo GSM/GPRS M95, los sensores DHT22 y LM75A, además de los pines para la lectura de los puertos seriales y
el PICkit 3. En la capa inferior (Figura 15) se encuentra el circuito de carga junto con la base para la SIM card.

Pensando en la ubicación de los sensores para la toma de datos se decidió que estos fueran conectados por medio de cables desde el circuito impreso hasta el lugar donde fueron requeridos. Los componentes DHT22 y LM75A tomaron medidas del ambiente o del cuerpo bovino por tales razones no pudieron ir en la caja del dispositivo.

Los componentes pertenecientes al circuito de carga, capa inferior, fueron ubicados para no generar interferencia con las señales de la capa superior, si no se tenía en cuenta se podría generar problemas por la disipación de calor del regulador. Para terminar, el circuito de carga tuvo integrados dos elementos que van al borde de la tarjeta para facilitar su acceso, estos eran el Conector USB y el switch de encendido y apagado.

Quectel (2013b) da algunas recomendaciones para que el módulo M95 tenga un correcto funcionamiento:

- Tener la SIM card lo más cercano posible a el módulo M95, la distancia no debe ser mayor a 20 centímetros.
- Tener la SIM card lo más alejado posible de la señal de la antena.
- El capacitor de desacople entre los pines SIM_VDD y SIM_GND debe ser menor a 1µF.
- Tener las señales de SIM_DATA y SIM_CLK lo más alejadas posible y rodearlas por tierra.

Teniendo los componentes ubicados en el circuito impreso se procede a rutear de la siguiente manera:

- Tratar de que las rutas no tengan ángulos cerrados esto genera deterioro de las señales.
- Las rutas deben salir por el centro de los pads para así evitar calentamiento en estos
- Usar la menor cantidad de vías (caminos que comunican la capa superior e inferior) posibles.
- Evitar que a partir de las vías salgan muchas rutas, esto genera calentamiento en la vía generando daños en la señal.
- Generar las rutas de forma ordenada, primero los ruteos entre componentes de la misma zona, seguido por las rutas a otra zona. Una vez terminadas las rutas entre todos los componentes, generar el camino de la alimentación y por último generar un polígono de tierra.

Al usar Altium se facilitó la integración entre los esquemáticos, el circuito impreso y el modelado 3D. En las Figura 14 y Figura 15 se presenta el diseño final de la PCB en las dos capas. En las Figura 16 y Figura 17 se presenta el modelo 3D de dicha tarjeta.
Figura 14. PCB capa superior

Figura 15. PCB capa inferior
Con el fin de acomodar y proteger al dispositivo y al bobino, se diseñó una caja para almacenar la tarjeta y los componentes. Para su diseño se tuvo en cuenta las siguientes restricciones mecánicas:

- El sensor de la temperatura ambiental y humedad relativa, DHT22, debe ir por fuera de la carcasa para poder leer datos reales.
- El sensor de temperatura corporal, LM75A, debe estar en contacto con la piel del bobino.
- Acceso al puerto USB para la carga del dispositivo.
- El switch para el encendido y apagado de la tarjeta debe tener fácil acceso.

La caja (Figura 18) fue diseñada en ABS con un grosor de 2 mm, en los laterales de la caja se agregaron 3 orificios para conectar un cable de USB tipo B para la carga, prender o apagar el dispositivo y conectar la antena de red móvil. En la cara superior se adicionaron 3 agujeros para: la conexión de los sensores por medio de cables, el botón de...
reinicio, y la conexión de la antena GPS. En la Figura 18 se puede observar el diseño de la caja con algunos de los componentes.

Figura 18. Modelo 3D de la caja

Teniendo el diseño de la PCB. Se generaron los archivos Gerber y la máscara de componentes para la fabricación de la tarjeta y posteriormente soldar los componentes sobre esta. En la Figura 19 se muestra la tarjeta con sus componentes. Una vez se tenía la tarjeta ensamblada y la caja impresa en ABS se procedió a armar el dispositivo (Figura 20), este tiene un tamaño de 92 mm de ancho, 78 mm de lago y 40 mm de alto.

Figura 19. Ensamble PCB
2.3.2 Desarrollo del software de adquisición de datos

En este apartado se explicará el proceso seguido para la programación y manejo de los sensores DHT22, temperatura y humedad relativa ambiental; LM75A, temperatura corporal; y L70, ubicación.

Para implementar el manejo de los sensores, se codificó la lectura y escritura de cada uno en el PIC. El ambiente de desarrollo fue MPLAB X IDE en su versión 5.30. La herramienta de depuración y programación que se usó fue PICkit 3 de Microchip debido a su fácil manejo y compatibilidad con MPLAB, por último, se estableció el uso del compilador XC8 v2.10 para generar el código máquina del microcontrolador.

Una vez definidas estas herramientas, se continuó con la creación del proyecto, configuración de los registros de inicio (FUSES) y los registros básicos para el funcionamiento mínimo del PIC con un cristal externo de 16 MHz.

Se comenzó con el sensor DHT22. Después de su estudio, se encontró que usa un protocolo de comunicación simplificado one-wire, es decir, una línea de comunicación para el intercambio de datos entre maestro-esclavo; en consecuencia, el Host es el único que inicia la comunicación con una secuencia predeterminada. Si hay un error en la señal de inicio no habrá una respuesta de parte del esclavo. Además, cuenta con un byte de paridad para la detección de errores en la comunicación. (AOSONG, 2010)

El diagrama de flujo de la Figura 21 expone el manejo del sensor DHT22 por parte del PIC. A partir del mismo se generaron las librerías “DHT_22.h” con su par “DHT_22.c”, mostradas en los Anexo A y B.
Las librerías anteriormente nombradas contienen dos funciones: la inicialización del pin RC0, “DHT_inicio”, y el protocolo establecido por el fabricante en la hoja de datos del componente (“Leer_DHT”); esta última recibe como parámetros de entrada dos direcciones de memoria con el fin de almacenar de los datos de humedad relativa y temperatura ambiental. En consecuencia, retorna un entero con 3 posibles resultados:

- 1: Valores de temperatura y humedad correctos
- 2: El sensor no ha respondido en el tiempo estimado
- 3: Los datos leídos no corresponden a la paridad leída.

En la Figura 22 se presentan los datos almacenados en la memoria del PIC dados por el sensor DHT22.

Para el monitoreo y corrección del código se vio necesario el envío de datos a una terminal serial del computador, para esto se usó el software Hercules SETUP utility y un módulo FTDI permitiendo una comunicación por protocolo UART.

El PIC 18F46K22 posee dos puertos USART, definidos como UART_1 y UART_2. Los parámetros usados para su configuración fueron: una comunicación asincrona, velocidad de 9600 baudios y trasmisión de 8 bits, estas funciones están mostradas en la librería mostrada en los Anexo C y D. Es importante destacar que cada función en UART_1 tiene su homólogo en UART_2. Se decidió usar interrupciones para la recepción de datos, esto
permite que el sistema atienda dichos datos en cualquier momento y no tener la probabilidad de la perdida de estos esperando su ejecución en el programa principal (Reyes Cortes & Cid Monjaraz, 2015).

Una vez validado y verificado el funcionamiento del sensor DHT22, se continuó con el módulo L70. Dicho módulo GPS es fabricado por la empresa Quectel, usa protocolo de comunicación UART y maneja comandos MTK NMEA para su manejo (Quectel, 2016), la estructura de esos mensajes se muestra a continuación:

- Comienza con el carácter “$”.
- Prioridad del mensaje representada con el carácter “PMTK”.
- Los 3 números predefinidos para la identificación del comando a ejecutar.
- Variables, si son necesarias para la ejecución del comando.
- Carácter “*” como finalización de la data.
- Número hexadecimal de dos dígitos como Checksum
- Final de la sentencia NMEA como “\r\n”

Para leer de la información se utilizan sentencias NMEA 0183, dichas fueron creadas por National Marine Electronics Association, trasmiten datos como la posición, la velocidad, el estado de los satélites, entre otros; usa buses de 4800 baudios, los cuales solo están diseñados para ser escuchados desde un dispositivo que maneje este protocolo (National Marine Electronics Association, s. f.). Lo que quiere decir que el módulo L70 solo es capaz de leer estas tramas mas no enviarlas a través de este bus.

El L70 maneja las sentencias NMEA: RMC, VTG, GGA, GSA, GSV y GLL, cada una da un dato como la ubicación, velocidad, orientación, entre otros. Para este caso solo fue de
interés la sentencia RMC, contiene fecha, hora, latitud y longitud. La estructura mostrada en la Tabla 9. explica cómo estaba ordenada la información, cada uno estaba separado por comas (,). Para mejor ilustración tenemos el ejemplo: “$GPRMC,053550.000,A, 0613.6294,N,07533.9707,W,1.66,292.48,300920,,,A*70”, de esta trama se pudo extraer la fecha “30 de septiembre del 2020”, hora “05:35:50”, y las coordenadas “06.136294 N 075.339707 W”.

Tabla 9. 
Arquitectura sentencia GPRMC (Qnectel, 2016)

<table>
<thead>
<tr>
<th>Parte</th>
<th>Descripción</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Inicio de mensaje</td>
</tr>
<tr>
<td>GPRM</td>
<td>Identificador sentencia</td>
</tr>
<tr>
<td>Hora UTC</td>
<td>Hora en formato “HHMMSS.SS”</td>
</tr>
<tr>
<td>Validez del dato</td>
<td>A = válido</td>
</tr>
<tr>
<td></td>
<td>V = Invalído</td>
</tr>
<tr>
<td>Latitud</td>
<td>En formato de grados y minutos</td>
</tr>
<tr>
<td>Hemisferio</td>
<td>N = Norte</td>
</tr>
<tr>
<td></td>
<td>S = Sur</td>
</tr>
<tr>
<td>Longitud</td>
<td>En formato de grados y minutos</td>
</tr>
<tr>
<td>Hemisferio</td>
<td>E = Este</td>
</tr>
<tr>
<td></td>
<td>W = Oeste</td>
</tr>
<tr>
<td>Velocidad</td>
<td>Velocidad en el suelo dado en nudos</td>
</tr>
<tr>
<td>Curso</td>
<td>Curso del movimiento en el suelo dado en nudos</td>
</tr>
<tr>
<td>Fecha</td>
<td>Fecha en formato “DDMMAA”</td>
</tr>
<tr>
<td>Velocidad magnética</td>
<td>Velocidad magnética parte numérica dado en grados</td>
</tr>
<tr>
<td>Velocidad magnética</td>
<td>Velocidad magnética indicador “E” o “W”</td>
</tr>
<tr>
<td></td>
<td>N = No configurado</td>
</tr>
<tr>
<td>Modo de posicionamiento</td>
<td>A = Automático</td>
</tr>
<tr>
<td></td>
<td>D = Diferencial</td>
</tr>
<tr>
<td>*</td>
<td>Final de datos</td>
</tr>
<tr>
<td>Checksum</td>
<td>Checksum entre los caracteres “$” y “*”</td>
</tr>
<tr>
<td>“\r\n”</td>
<td>Final del mensaje</td>
</tr>
</tbody>
</table>

Las funciones relacionadas con el manejo del sensor L70 se muestran en el Anexo I, se decidió usar el UART_2 para su comunicación; para la inicialización se usa la función “GPS_unit”, esta no recibe ni retorna ningún valor. El módulo por defecto envía las 6 sentencias NMEA mencionadas anteriormente cada 200 milisegundos, en primer lugar se realizó un cambio en ese tiempo de muestreo, se solicitó el envío de solo la sentencia RMC y por último se seleccionó el método “Hot Start” para el inicio del módulo, según Qnectel (2016), la última configuración mencionada guarda los parámetros de los satélites de la última lectura válida para el inicio del módulo, si el módulo no dura más de 3 horas en reposo es recomendado usarlo, lo que causa una disminución del tiempo de respuesta. Una vez configurado estos parámetros se prosiguió a cambiar el tiempo de transmisión al mínimo de 200 milisegundos hasta que el módulo se sincronice con los satélites y trasmita datos válidos de latitud y longitud, una vez se termine de guardar los parámetros para el cálculo de la posición se procede a cambiar la velocidad de transmisión a 1 segundo y de
esta manera se facilita la comunicación con el PIC y evitando pérdida de datos en el proceso; por último, se trasmite el comando para entrar en modo “Standby”. Los comandos para realizar este proceso se muestran en la Tabla 10 con su respectiva función.

**Tabla 10.**
*Configuraciones GPS*

<table>
<thead>
<tr>
<th>Comando</th>
<th>Función</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PMTK220,10000*2F\n</td>
<td>Envío de datos cada 1 segundo</td>
</tr>
<tr>
<td>$PMTK314,0,1,0,0,0,0,0,0,0,0,0\n</td>
<td>Envío de solo la sentencia RMC</td>
</tr>
<tr>
<td>$PMTK101*32\n</td>
<td>Configuración “Hot Start”</td>
</tr>
<tr>
<td>$PMTK220,200*2C\n</td>
<td>Envío de datos cada 200 milisegundo</td>
</tr>
<tr>
<td>$PMTK161,0*28\n</td>
<td>Modo “Standby”</td>
</tr>
</tbody>
</table>

La lectura y almacenamiento de datos GPS se realizó con 4 funciones: “GPS_lectura”, “checksum”, “GPS_buscar” y “GPS_guardar”, mostradas en el Anexo I. La primera está basada en la Figura 23, en donde se aclara el proceso para la lectura y almacenamiento de un dato. Consiste en prender el módulo, esperar que se sincronice con los satélites y envíe datos válidos, volver a modo “Standby” y almacenar lo leído. Para asegurarse que el dato recibido sea correcto, se verifica los dígitos correspondientes al Checksum, con la función “checksum”. Cuando el resultado de lo anterior sea válido y se tenga la trama GPRMC (ejemplos en la Figura 24), se usa la función “GPS_buscar” se buscan las posiciones donde se encuentren los separadores (“,”), de esta manera ubicar los datos y poder guardar los. La fecha está ubicada en la novena coma; hora en la primera coma; latitud cuarta coma; longitud, sexta coma y hemisferios, quinta y séptima coma. Estos datos se almacenan en cadenas o matrices de char y facilitar su manejo.
El sensor LM75A, elegido para tomar la temperatura corporal, es un sensor digital con interfaz de comunicación I2C, por tal razón se adaptó las librerías I2C existentes al PIC 18f46k22 generando la librería “I2C_18f46k22.h” con su equivalente “I2C_18f46k22.c” mostradas en los Anexos Anexo E y Anexo F.

El LM75A puede usar entre 8 a 11 bits de resolución lo que traduce en variaciones entre 1°C, 0.5°C, 0.25°C y 0.125°C. Para propósito de este desarrollo se escogió trabajar con 11 bits, esta configuración se encontraba en los bits 4 y 3 del registro 0x01, encargado de la configuración del todo el sensor, el bit 0 de este mismo registro fue el responsable del
encendido y apagado del sensor. Por otra parte, se eligieron los últimos 3 bits de la dirección en el bus I2C como cero quedando esta como 0x48 en hexadecimal.

Para la utilización del LM75A se generó la librería “LM75A.h” (Anexo G) con su equivalente "LM75A.c" (Anexo H), cabe aclarar que fue basada en el protocolo I2C. Se desarrollaron 3 funciones para su manejo: “LM75A_Start”, “LM75A_Stop” y “LM75A_read”. Las dos primeras fueron las encargadas de prender y pagar el módulo sin afectar la resolución de 11 bits. La última estaba en encargada de leer los 2 bytes del registro de temperatura (0x00) para luego hacer la conversión y tener el valor de la temperatura. La Figura 25 muestra algunas de las medidas de este sensor.

![Figura 25. Datos del sensor LM75A](image)

Después del estudio de cada sensor y módulo se decidió guardar los datos relacionados con el GPS L70 como cadenas de char para los hemisferios, y matrices de char para las variables: Fecha, longitud y latitud. El tamaño de las matrices dependió de la longitud de caracteres un solo dato, como número de columnas y la cantidad de datos guardados, como número de filas, es decir, un dato de latitud estaba compuesto como “MM-SS.SSSS” donde hay 10 caracteres, y se hacía la toma de datos 12 veces, por ende, la matriz donde se almacena la latitud tiene 10 columnas y 12 filas. Las variables de hemisferio fueron cadenas de char de tamaño igual a la cantidad de datos guardados. Si se tomaron 12 datos, la cadena de caracteres tiene una longitud de 12 posiciones.

Las variables de temperatura corporal, temperatura ambiental y humedad relativa son guardadas directamente como una cadena de caracteres, para facilitar su almacenamiento.
Finalmente decidimos monitorear a los bovinos tomando datos cada diez minutos y que estos se envíen al servidor cada dos horas, esto con el fin de agrupar la mayor cantidad de datos y evitar conexiones con el api (MongoDB) con poco flujo de información. (Anexo I)

3.3.3 Desarrollo de envío de datos a la nube

Para la comunicación con MongoDB se usó el módulo GSM/GPRS M95, el manejo de este se hacía por medio de comandos AT.

Los comandos Hayes fueron desarrollados por Dennis Hayes en 1977, su objetivo, en un principio fue ser un método de comunicación entre hombre-máquina para la configuración y la trasmisión de directrices por medio de instrucciones codificadas. Con el tiempo y el desarrollo de estos, la tecnología los adapto como medio de intercambio de información hasta llegar a lo que se conoce hoy, su nombre cambio a comandos AT por la abreviación de “attention”. (Gómez Cubides & Herrera Moreno, 2019)

El M95 usa protocolo UART, anteriormente se explicó que la recepción de datos se hizo por medio de interrupciones. Su programación se encuentra en el programa principal del PIC (Anexo I).

Después del estudio del módulo se generó el diagrama de flujo mostrado en la Figura 26, para la programación fue necesario definir: el operador con su respectivo APN, el enlace del servidor y la arquitectura de la trama a enviar. Con respecto a la primera incógnita se decidió usar una SIM del operador Claro con una APN, “Internet.comcel.com.co”.

Se desplegó el servidor en Heroku, este es un contenedor en la nube que permite desplegar y administrar aplicaciones modernas, que cuenta con las siguientes ventajas (Heroku, 2017):

- Cuenta con la infraestructura de AWS (Amazon Web Services) lo que garantiza un excelente rendimiento, disponibilidad y estabilidad.
- Es seguro ya que los dynos se encuentran aislados entre sí, además el firewall y usuarios se encuentran administrados y configurados.
- Los dynos cuentan con alta capacidad de procesamiento por lo que el funcionamiento es rápido
- Integra varios servicios dentro de su estructura
- Permite el uso de diferentes lenguajes de programación.

Luego de analizar la estructura y necesidades de nuestra aplicación se decidió usar este servicio para desplegar nuestro servidor por lo cual el enlace se definió de la siguiente manera: https://deploytesis.herokuapp.com/medicion
Para la estructura de la trama de datos se tuvo en cuenta la información de la Tabla 11, para seleccionar la que mejor se adaptara a las necesidades de nuestro proyecto.

**Tabla 11. Comparación entre formatos XML y JSON**

<table>
<thead>
<tr>
<th>Alternativa de solución</th>
<th>XML <em>(extensible markup language)</em></th>
<th>JSON <em>(java script object notation)</em></th>
</tr>
</thead>
</table>
| **Ventajas**            | • Tiene un formato estructurado y fácil de comprender.  
                          • Se pueden definir estructuras complejas y reutilizables.  
                          • Tiene soporte a cualquier tipo de datos.  
                          • Fácil lectura  
                          • Es más ligero *(bytes)* en las transmisiones.  
                          • Velocidad de procesamiento alta.  
                          • Soporte para web, Android y IOS. |
| **Desventajas**         | • El formato es muy estricto.  
                          • Lleva más tiempo procesarlo.  
                          • Un error en cualquier parte del formato puede hacer que todo el documento sea inválido.  
                          • Tiene una estructura compleja y difícil de interpretar a simple vista. |

Finalmente se seleccionó el formato JSON, este nos permite estandarizar los datos en la sintaxis de objetos *JavaScript*, lenguaje de programación en el cual de desarrolló la aplicación, y así por medio de métodos nativos del lenguaje la manipulación de la información es más sencilla y rápida. Para definir la trama se tienes las variables:

- Fecha
- Hora
- ID
- Temperatura corporal
- Temperatura ambiental
- Humedad relativa
- Latitud (parte numérica)
- Hemisferio Latitud
- Longitud (Parte numérica)
- Hemisferio Longitud

Definidos los datos a enviar junto con la estructura básica de una trama JSON se estructura la trama enviar: `{"f":"MM-DD-AAAA-H:MM:SS","id":###,"tc":###,"ta":###,"h":###,"la":"MM-SS.SSSS","lae":"H","lo":"MM-SS.SSSS","loe":"H"}`.
Con la trama, se procedió a generar la función “M95_trama”, como parámetro de entrada estaba el índice del vector para generar la trama. Inicia convirtiendo los números en cadena de char, se procede a enviar carácter a carácter la trama a través de UART_1. En la Figura 27 se muestran algunas tramas generadas por esta función.

La función “M95_Comunicacion” (Anexo I) se basó en la Figura 26, aceptaba como parámetro el número de tramas a enviar al servidor. Para llevar a cabo el envío de datos a MongoDB se debía hacer una conexión a la VPN del operador, ingresar la URL del servidor, enviar las tramas según el índice por medio de métodos POST, verificar la respuesta del servidor, “dato guardado”, y enviar la siguiente trama, así hasta terminar. Terminado el envío de datos se desconectó el módulo de la VPN, por ende, también del servidor. Estos comandos AT están especificados en la Tabla 12. Cabe resaltar que se verifica la respuesta de cada comando AT para poder continuar con el proceso.

<table>
<thead>
<tr>
<th>Comando</th>
<th>Función</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+COPS?\n</td>
<td>Mostramos a que operador está conectado</td>
</tr>
<tr>
<td>AT+QICSGP=1,&quot;Internet.comcel.com.co&quot; \n</td>
<td>Conexión a APN</td>
</tr>
<tr>
<td>AT+QHTTPURL=42\n</td>
<td>Conexión al servidor, parámetro</td>
</tr>
<tr>
<td></td>
<td>cantidad de caracteres del enlace</td>
</tr>
<tr>
<td>AT+QHTTPPOST=118\n</td>
<td>POST en el servidor, parámetro</td>
</tr>
<tr>
<td></td>
<td>cantidad de caracteres de la trama</td>
</tr>
<tr>
<td>AT+QHTTPREAD\n</td>
<td>Leer mensaje de respuesta mandado</td>
</tr>
<tr>
<td></td>
<td>por el servidor</td>
</tr>
<tr>
<td>AT+QIDEACT\n</td>
<td>Desconexión de la APN</td>
</tr>
</tbody>
</table>
Para el servidor como se visualiza en el Anexo J se inició creando la conexión con MongoDB, se verifica que este paso se cumpla correctamente, si es así se crea la base de datos nombrada “datos” donde más adelante se guardarán las tramas enviadas por el M95 dentro de diferentes colecciones, en nuestro caso cada colección se divide por cada Id, es decir, cada bovino cuenta con su propia base de datos como se muestra en la Figura 28, por el contrario si se presenta algún error se retorna en este caso “Error occurred while connecting to MongoDB Atlas”.

<table>
<thead>
<tr>
<th>Collection Name</th>
<th>Documents</th>
<th>Documents Size</th>
<th>Documents Avg</th>
<th>Indexes</th>
<th>Index Size</th>
<th>Index Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>7</td>
<td>1.53KB</td>
<td>224B</td>
<td>1</td>
<td>36KB</td>
<td>36KB</td>
</tr>
<tr>
<td>102</td>
<td>11</td>
<td>2.4KB</td>
<td>223B</td>
<td>1</td>
<td>36KB</td>
<td>36KB</td>
</tr>
<tr>
<td>103</td>
<td>32</td>
<td>7KB</td>
<td>224B</td>
<td>1</td>
<td>30KB</td>
<td>36KB</td>
</tr>
</tbody>
</table>

Figura 28. Base de datos creada para cada bovino según su id

Luego de esto por medio de los protocolos HTTP y el método POST se creó el endpoint que usaría el módulo M95 para enviar los datos recolectados, como se aprecia en el Anexo K la ruta de este fue “/medición”, recibe los paquetes de datos, los cuales por medio del “JSON.parse()” se transforman en un objeto de JavaScript; luego de esto, por medio de la propiedad “id” que se encuentra dentro de la trama recibida se validan las colecciones existentes dentro de la base de datos y si ya existe una nombrada con este valor, la trama se guarda en esta, pero, si no se encuentra se agrega una nueva con el id como nombre.

Para validar, se envía al módulo M95 el mensaje “dato guardado” cuando todo el proceso se cumple correctamente, en caso de ocurrir alguna anomalía con la trama o el proceso se retorna un mensaje de advertencia.

4.3.4 Procesamiento de datos

Luego de recibir las tramas y guardarlas en la base de datos satisfactoriamente se debía enviar esta información a la aplicación para la visualización de estas. Como se muestra en el Anexo L con la ayuda del método GET del mismo endpoint “/mediciones” se envía por medio de la línea de código “res.send(JSON.stringify(result))” toda la información recolectada.

El siguiente paso se realiza directamente en el código de la aplicación, como se aprecia en el Anexo M por medio del comando “fetch()”, este apunta al endpoint “/mediciones” el cual proporciona una forma más fácil y lógica de obtener la información de forma asíncrona a través de la red, es decir, proporciona una interfaz para JavaScript para acceder y manipular las solicitudes y respuestas hechas por HTTP. (MDN Web Docs, 2020)
Posteriormente se indica que la respuesta obtenida por el “fetch()” es un archivo JSON y así poder iniciar con el procesamiento de los datos, cabe resaltar que este procedimiento lo realizó directamente en la aplicación para evitar sobreprocesamientos en el PIC y afectar de una u otra forma la capacidad de memoria de este. En la base de datos las tramas se guardan como se observa en la Figura 29, donde MongoDB asigna un “_id” para identificarlos dentro de la base de datos, pero no interfere con los valores enviados por el módulo M95.

Los datos llegan tal cual se muestran en la Figura 29, por lo cual era necesario modificar la mayoría para lograr que fueran entendibles para los usuarios, el proceso general se muestra en el Anexo M, pero a continuación se explicará cada uno:

- **Fecha**: este valor llega a la base de datos separando con guiones los valores, es decir día, mes, año y hora; por lo cual en el código como se muestra en la Figura 30 solo fue necesario asignar estos valores a variables independientes, para luego con la función “Date()” establecer el formato adecuado.

```javascript
// procesamiento fecha
const [dia, mes, año, hora] = element.fecha.split("-\")
const fecha = new Date("${mes}-${dia}-${año}-${hora}"")
fecha.setHours(fecha.getHours() - 5)
```

- **Temperatura ambiente, corporal y humedad**: estos valores tuvieron el mismo procesamiento dado que llegaban con la misma estructura, por lo cual solo fue necesario indicarle a JavaScript que eran números y debía dividirlos por diez como se presenta en la Figura 31, finalmente los valores son los siguientes:

  - **Temperatura ambiente**: 24.9
La temperatura corporal es 38.6°C y la humedad es 59.3%.

```javascript
// procesamiento temperaturas y humedad
const temperaturaAmbiente = Number(element.temperaturaAmbiente) / 10
const humedad = Number(element.humedad) / 10
const temperaturaCorporal = Number(element.temperaturaCorporal) / 10
```

La Figura 31 muestra el código para procesar temperaturas y humedad.

**Latitud y longitud:** Con estos valores era necesario convertir las coordenadas que se encontraban en formato de grados y minutos, al formato de grados decimales que recibe GOOGLE MAPS; para esto se divide por 60 los minutos y se les suma este resultado a los grados.

Según los datos presentados en la Figura 29 las operaciones que se deben realizar son las siguientes:

\[
\text{Latitud} = 61 + \frac{50.573}{60} \quad \text{Longitud} = 75 + \frac{34.2286}{60}
\]

Para lograr esto primero se debían asignar los valores en variables separadas ya que en la trama de datos llega un único valor y poder manipularlos y llevar a cabo la operación como se muestra en la Figura 32 y así obtener los valores “Latitud: 61.84288333333” y “Longitud: -75.57047666666”.

```javascript
// procesamiento coordenadas
const latitud1 = Number(element.latitud.split('-')[0])
const latitud2 = Number(element.latitud.split('-')[1])
const lat = latitud1 + latitud2 / 60
const longitud1 = Number(element.longitud.split('-')[0])
const longitud2 = Number(element.longitud.split('-')[1])
const lng = (longitud1 + longitud2 / 60) * -1
```

La Figura 32 muestra el código para procesar coordenadas.

### 5.3.5 Interfaz gráfica

Para el diseño inicial de la aplicación se decidió seccionarla de tal forma que toda la información se encontrara agrupada estratégicamente y para poder navegar entre ella, se implementó una barra estática en la parte superior de la pantalla. Partiendo de esto, se decidieron las siguientes pantallas.

En la pantalla principal, se implementó un carrusel de información, con el cual se tenía la posibilidad de explicar en diferentes secciones todo lo relacionado con nuestro trabajo, empezando con una idea general, diferentes datos de interés y tema principal.
En el diseño final se generaron 3 secciones del carrusel, Moo World, Datos de interés y Estrés térmico. Con la finalidad de abarcar y dejar claridad sobre todos los temas tratados a lo largo del trabajo y que opciones se encuentran dentro de la aplicación web.

En la sección Mapa aparece inicialmente un mensaje describiendo más a profundidad las funcionalidades existentes en este apartado y así evitar confusiones en el usuario mientras maneja la aplicación (Figura 33). Luego de este mensaje lo primero que se visualiza en el mapa son diferentes pines, los cuales hacen referencia a los bovinos que se encuentran dentro de la base de datos hasta ese momento, al poner el cursor sobre cada uno aparece la información del bovino seleccionado, además, este pin aparece de diferentes colores (verde, naranja, rojo) teniendo en cuenta los datos de cada animal para indicar el estado actual del mismo, de igual forma cuando se agrega un nuevo bovino inmediatamente se puede observar en el mapa con su respectiva información (Figura 34).

Mapa de calor tiene la posibilidad de seleccionar el bovino por medio del “id” y mostrar el mapa de calor de acuerdo con las posiciones de este pasado el tiempo, es decir, visualmente muestra los lugares del terreno más frecuentados por el bovino y con base en esto es posible tomar decisiones respecto a las zonas más importantes del sector, además si hay un bovino nuevo en la base de datos inmediatamente permite seleccionarlo para crear su mapa de calor.

![Figura 33. Mensaje inicial sección Mapa](image-url)
Figura 34. Estado actual

Figura 35. Mapa de calor
En la sección de *Datos* se muestra inicialmente una breve descripción de los valores de la tabla y la explicación detallada para generar y guardar gráficas según las columnas deseadas (Figura 36), luego dentro de esta sección se despliega toda la información enviada por el dispositivo a la base de datos; por cada bovino se visualizan los valores actuales de cada variable dando la opción de desplegar el histórico de estos; para el campo de temperatura corporal se agregó un led indicando el estado de este a partir de diferentes colores (verde, naranja, rojo) y poder sacar alertas dependiendo de esto, además en el campo de estrés térmico se calculó el índice de temperatura y humedad (ITH) con los valores de humedad relativa y temperatura ambiente para definir el estado de estrés térmico en el bovino (Figura 37). Toda la configuración de la aplicación web se implementó en el siguiente link: [https://monitoreo-bovino-eia.netlify.app](https://monitoreo-bovino-eia.netlify.app)

Figura 36. Mensaje inicial sección Datos
2.4 PRUEBAS

A continuación, se explica el proceso de evaluación que se llevó a cabo en el hardware (tarjeta), api (servidor) y página web, con el fin de verificar su comportamiento individual bajo diferentes condiciones. Para después, poner a prueba el sistema integrado en un ambiente controlado, de tal forma que fuera posible analizar la información, identificar falencias y posibles mejoras para obtener resultados óptimos.

2.4.1 Pruebas Hardware

Para probar el funcionamiento del dispositivo físico se tomaron 3 coordenadas geográficas diferentes, en cada una se hicieron 5 mediciones por sensor, permitiendo calcular un porcentaje de error con respecto a los valores teóricos.

La temperatura corporal dada por el LM75A se comparó con un termómetro, mientras que los datos medidos por el DHT22 se verificaron con un termohigrómetro. En la Tabla 13 Tabla 14 y Tabla 15 se muestran dichos datos dependiendo del punto de medición. Cabe aclarar que tanto los valores de los sensores y los instrumentos están tomados en el mismo momento.
Tabla 13.
Comparación mediciones punto 1

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Medición dispositivo</th>
<th>Medición referencia</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperatura ambiente</td>
<td>23.5 °C</td>
<td>24.4 °C</td>
<td>3.83%</td>
</tr>
<tr>
<td>(DHT22)</td>
<td>19.7 °C</td>
<td>19.7 °C</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>19.9 °C</td>
<td>19.7 °C</td>
<td>1.01%</td>
</tr>
<tr>
<td></td>
<td>19.6 °C</td>
<td>19.2 °C</td>
<td>2.04%</td>
</tr>
<tr>
<td></td>
<td>19.6 °C</td>
<td>18.6°C</td>
<td>5.10%</td>
</tr>
<tr>
<td>Temperatura corporal</td>
<td>36.2 °C</td>
<td>36.4 °C</td>
<td>0.55%</td>
</tr>
<tr>
<td>(LM75A)</td>
<td>36.4 °C</td>
<td>36.5 °C</td>
<td>0.27%</td>
</tr>
<tr>
<td></td>
<td>36.6 °C</td>
<td>36.6 °C</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>36.8 °C</td>
<td>36.6 °C</td>
<td>0.54%</td>
</tr>
<tr>
<td></td>
<td>36.5 °C</td>
<td>36.6 °C</td>
<td>0.27%</td>
</tr>
<tr>
<td>Humedad relativa</td>
<td>75.4%</td>
<td>74%</td>
<td>1.85%</td>
</tr>
<tr>
<td>(DHT22)</td>
<td>92.1%</td>
<td>87%</td>
<td>5.54%</td>
</tr>
<tr>
<td></td>
<td>92.0%</td>
<td>87%</td>
<td>5.43%</td>
</tr>
<tr>
<td></td>
<td>92.1%</td>
<td>89%</td>
<td>3.37%</td>
</tr>
<tr>
<td></td>
<td>91.9%</td>
<td>90%</td>
<td>2.07%</td>
</tr>
</tbody>
</table>

Tabla 14.
Comparación mediciones punto 2

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Medición dispositivo</th>
<th>Medición referencia</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperatura ambiente</td>
<td>21.1°C</td>
<td>21.0°C</td>
<td>0.47%</td>
</tr>
<tr>
<td>(DHT22)</td>
<td>21.0°C</td>
<td>20.0°C</td>
<td>4.76%</td>
</tr>
<tr>
<td></td>
<td>22.4°C</td>
<td>20.5°C</td>
<td>8.48%</td>
</tr>
<tr>
<td></td>
<td>23.4°C</td>
<td>21.3°C</td>
<td>8.97%</td>
</tr>
<tr>
<td></td>
<td>22.8°C</td>
<td>22.0°C</td>
<td>3.51%</td>
</tr>
<tr>
<td>Temperatura corporal</td>
<td>36.5°C</td>
<td>36.6°C</td>
<td>0.27%</td>
</tr>
<tr>
<td>(LM75A)</td>
<td>36.6°C</td>
<td>36.6°C</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>36.6°C</td>
<td>36.6°C</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>37.0°C</td>
<td>36.5°C</td>
<td>1.35%</td>
</tr>
<tr>
<td></td>
<td>36.9°C</td>
<td>36.6°C</td>
<td>0.81%</td>
</tr>
<tr>
<td>Humedad relativa</td>
<td>90.0%</td>
<td>85%</td>
<td>5.56%</td>
</tr>
<tr>
<td>(DHT22)</td>
<td>87.0%</td>
<td>83%</td>
<td>4.60%</td>
</tr>
<tr>
<td></td>
<td>88.2%</td>
<td>88%</td>
<td>0.23%</td>
</tr>
<tr>
<td></td>
<td>88.2%</td>
<td>82%</td>
<td>7.03%</td>
</tr>
<tr>
<td></td>
<td>74.4%</td>
<td>81%</td>
<td>8.87%</td>
</tr>
</tbody>
</table>
### Tabla 15.
*Comparación mediciones punto 3*

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Medición dispositivo</th>
<th>Medición referencia</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperatura ambiente</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(DHT22)</td>
<td>26.1°C</td>
<td>25.3°C</td>
<td>3.07%</td>
</tr>
<tr>
<td></td>
<td>26.3°C</td>
<td>25.2°C</td>
<td>4.18%</td>
</tr>
<tr>
<td></td>
<td>26.4°C</td>
<td>26.0°C</td>
<td>1.52%</td>
</tr>
<tr>
<td></td>
<td>26.2°C</td>
<td>25.7°C</td>
<td>1.91%</td>
</tr>
<tr>
<td></td>
<td>25.9°C</td>
<td>26.0°C</td>
<td>0.39%</td>
</tr>
<tr>
<td>Temperatura corporal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(LM75A)</td>
<td>36.3°C</td>
<td>36.6°C</td>
<td>0.83%</td>
</tr>
<tr>
<td></td>
<td>36.6°C</td>
<td>36.6°C</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>36.5°C</td>
<td>36.5°C</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>36.8°C</td>
<td>36.6°C</td>
<td>0.54%</td>
</tr>
<tr>
<td></td>
<td>36.6°C</td>
<td>36.7°C</td>
<td>0.27%</td>
</tr>
<tr>
<td>Humedad relativa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(DHT22)</td>
<td>71.8%</td>
<td>73%</td>
<td>1.67%</td>
</tr>
<tr>
<td></td>
<td>69.9%</td>
<td>72%</td>
<td>3.00%</td>
</tr>
<tr>
<td></td>
<td>72.8%</td>
<td>70%</td>
<td>3.85%</td>
</tr>
<tr>
<td></td>
<td>70.4%</td>
<td>75%</td>
<td>6.53%</td>
</tr>
<tr>
<td></td>
<td>69.3%</td>
<td>69%</td>
<td>0.43%</td>
</tr>
</tbody>
</table>

La verificación de dato del L70, se hizo de la misma forma que los sensores. Se tomaron 5 diferentes coordenadas y se compararon con datos tomados por un GPS. En la Tabla 16 se muestran los datos tomados con su respectivo error.

### Tabla 16.
*Comparaciones mediciones GPS*

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Medición dispositivo</th>
<th>Medición referencia</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitud</td>
<td>6.194017</td>
<td>6.193947</td>
<td>0.0011%</td>
</tr>
<tr>
<td></td>
<td>6.193860</td>
<td>6.193841</td>
<td>0.0003%</td>
</tr>
<tr>
<td></td>
<td>6.193618</td>
<td>6.193893</td>
<td>0.0044%</td>
</tr>
<tr>
<td></td>
<td>6.225423</td>
<td>6.225512</td>
<td>0.0014%</td>
</tr>
<tr>
<td></td>
<td>6.227279</td>
<td>6.226947</td>
<td>0.0053%</td>
</tr>
<tr>
<td>Longitud</td>
<td>-75.595020</td>
<td>-75.595012</td>
<td>0.0001%</td>
</tr>
<tr>
<td></td>
<td>-75.592126</td>
<td>-75.592315</td>
<td>0.0002%</td>
</tr>
<tr>
<td></td>
<td>-75.593326</td>
<td>-75.593254</td>
<td>0.0001%</td>
</tr>
<tr>
<td></td>
<td>-75.566135</td>
<td>-75.566336</td>
<td>0.0002%</td>
</tr>
<tr>
<td></td>
<td>-75.565802</td>
<td>-75.566521</td>
<td>0.0009%</td>
</tr>
</tbody>
</table>
2.4.2 Pruebas Servidor

Debido a que el sistema envía datos a la nube mediante protocolo HTTP, fue necesario verificar el correcto manejo de estos por parte del *hardware* y el *software*. Para esto se generaron tres paquetes de datos, cada uno con 90 tramas, donde la humedad aumentaba de 10 a 99, por último, se modificó la velocidad de envío de datos a 5 segundos donde el sistema funciona adecuadamente y a máxima velocidad. Como resultado, todos los paquetes enviados llegaron al servidor. En la Figura 38. Correspondencia de datos Figura 38 se muestran los datos recibidos por el servidor.

![Correspondencia de datos en el servidor](attachment:fig38.png)

Figura 38. Correspondencia de datos

2.4.3 Pruebas Software

Para las pruebas del Software se escogieron 4 puntos estratégicos alrededor de América, con la finalidad de enviar tramas con datos específicos para verificar el funcionamiento de las alertas en la sección Mapa en la página web. En cuanto a la sección Datos, se enviaron tramas de prueba con ids existentes, para evaluar el comportamiento de los indicadores en las columnas de temperatura corporal y estrés térmico.

En la Figura 39 se muestran los 4 puntos con su respectivo color en el estado actual, así mismo, en la Figura 40 se muestra el histórico de datos con los indicadores de temperatura corporal y las alertas de estrés térmico.
Figura 39. Verificación de alertas en Estado Actual

Figura 40. Verificación de alertas en Datos
2.4.4 Pruebas de funcionamiento en un ambiente controlado

Los siguientes pasos fueron diseñados para la preparación de las pruebas en un ambiente controlado y verificar el funcionamiento del prototipo y la obtención de la información:

1. ESTIMACIONES PREVIAS

   Se decidió cambiar el tiempo de muestreo de 20 minutos a 10 segundos, con la finalidad de verificar el comportamiento del dispositivo y la calidad de los valores guardados en la base de datos.

2. LUGARES DE TOMA DE DATOS

   Para tener comportamientos diferentes y poder verificar la precisión de los sensores se decidió tomar datos con diferentes ids en tres lugares diferentes teniendo entre ellos una distancia aproximada de 130 metros. En cada lugar se tomaron 100 datos.

3. VERIFICACIONES PERIÓDICAS

   Mediante periodos de aproximadamente 15 minutos, se verificó el correcto funcionamiento del dispositivo, teniendo en cuenta lo siguiente:

   a. Dispositivo encendido.

   b. El tiempo de muestreo sea el establecido.

   c. Los sensores estén conectados y tomando datos.

   d. Los datos tomados sean enviados correctamente al servidor.

   e. Una visualización de los datos de manera correcta en la página web. Verificando que los valores presentados en las tablas y pines ubicados en los mapas correspondan con la información almacenada en el servidor.

   En cuanto a los datos de posición enviados por el L70 es importante recalcar que los mapas de calor generados a partir de estos valores serán más representativos en la medida que lleguen más datos, dado que es un sistema que grafica el acumulado de la información. Esto se puede verificar con los mapas generados en cada punto de medición, ya que el de mayor cantidad de datos (Figura 43) tiene mayor concentración y ofrece información más exacta que los otros puntos (Figura 41 y Figura 42).
Figura 41. Mapa de calor posición 1

Figura 42. Mapa de calor posición 2
En cuanto a la funcionalidad esperada en estado actual, se pueden observar en la Figura 44 los pines ubicados en los lugares de medición cumpliendo satisfactoriamente con los rangos definidos anteriormente.
3 ANÁLISIS DE RESULTADOS

Los esquemáticos presentados en la sección de diseño electrónico (Figura 8, Figura 9, Figura 10, Figura 11 y Figura 12) fueron diseñados acordes a las recomendaciones de las hojas de datos de cada componente, facilitando la estructuración y funcionamiento de la PCB (Figura 14 y Figura 15), ya que con ayuda de estos se realizaron las conexiones de todo el dispositivo.

Los porcentajes de error presentados en las Tabla 13, Tabla 14, Tabla 15 y Tabla 16 indicaron que el dispositivo tiene un error promedio de 2.09% con respecto a los datos teóricos (termómetro, termohigrómetro y GPS) por lo cual se puede decir que los resultados obtenidos son válidos. Sin embargo, el sensor DHT22 presenta errores hasta el 9% debido a su configuración.

De igual forma se considera que el protocolo de comunicación HTTP entre el hardware y el software cumplió a cabalidad con los requisitos inicialmente estipulados; teniendo en cuenta las pruebas realizadas anteriormente (Figura 38) podemos concluir que no hubo pérdida de datos durante la comunicación, debido a que todos los paquetes enviados por el dispositivo se pudieron visualizar en el servidor.

El índice de temperatura y humedad, ITH, nos permitió identificar el estado de estrés térmico por medio de los datos previamente recolectados, validados y enviados por el dispositivo, además, se pudo verificar el estado del animal. Con lo anterior es posible formular planes de mejoramiento en los ambientes para la comodidad del ganado, es decir, adecuar el terreno agregando nuevos bebederos, sales, alimentos y sombras en lugares donde menos se frecuenta partir de la información dada por los mapas de calor, de esta forma es posible rotar el ganado y aumentar el aprovechamiento del terreno.

Adicionalmente, dada la escalabilidad de este proyecto es viable agregar un nuevo bovino, reconocer las variables y mantener un monitoreo individual tanto de su estado de salud como su posición dentro del terreno, lo cual, permite identificar rápidamente comportamientos irregulares como el aislamiento y falta de movimiento en cada individuo, para actuar acorde a la situación.
4 CONCLUSIONES Y CONSIDERACIONES FINALES

Es importante destacar que la selección de los componentes fue fundamental para la adquisición, comunicación y visualización de la información de este proyecto, ya que nos permitió construir un dispositivo para tomar los datos de forma confiable y adecuada de las variables de interés.

Para almacenar, analizar y visualizar toda la información se creó un servidor con base en los requerimientos de diseño definidos durante el proyecto, el cual facilitó la integración entre los componentes del *hardware* y *software*. La aplicación se desarrolló teniendo en cuenta la perspectiva de un usuario, para poder mostrar los datos de forma clara, concisa y dejar una experiencia agradable en cada persona como lo son los mapas de calor.

El protocolo de pruebas permitió comprobar el funcionamiento de las partes del sistema de una manera detallada, de esta manera asegurar el correcto comportamiento del sistema en su totalidad.

Teniendo en cuenta lo anterior, se desarrolló satisfactoriamente un sistema que toma, envía, procesa y muestra los datos de interés para el proyecto. Al mismo tiempo, se generaron alertas e información con respecto al estado del bovino de una manera fácil de entender, con lo cual el ganadero puede tomar decisiones respecto al manejo del ganado.

Adicionalmente, la interfaz presenta un ambiente intuitivo, de fácil manejo y comprensión, cumpliendo con el objetivo de mostrar datos de diferentes bovinos al mismo tiempo; de igual forma, la página web dentro de cada pestaña explica detalladamente su funcionamiento teniendo en cuenta el público objetivo, igualmente ofrece la opción de generar varios tipos de gráficas según las necesidades del usuario facilitando al ganadero el análisis y lectura de las variables recolectadas por el prototipo con ayuda de indicadores.

Para futuras aplicaciones se considera importante agregar líneas de tendencia, para lograr identificar patrones de comportamientos a lo largo de los años y generar planes de acción para ahorrar tiempo y dinero aumentando ganancias y productividad.

Igualmente se puede implementar herramientas estadísticas para eliminar datos atípicos dados por los sensores y aumentar la precisión de estos, lo cual se vería reflejado en la visualización y análisis.

Para mejorar la experiencia de usuario se propone que la página web soporte una mayor cantidad de resoluciones de pantalla, es decir, poder abrir la aplicación desde cualquier dispositivo sin afectar la composición de los elementos.
Para finalizar, se plantea la posibilidad de agregar a la página web y al servidor un sistema de seguridad, el cual cuente con usuario y contraseña, para aumentar la escalabilidad y privacidad respecto a los demás ganaderos. Igualmente se puede pensar en encriptar los datos pensando en posibles robos de la información durante la comunicación.
REFERENCIAS


National Marine Electronics Association. (s. f.). NMEA 0183 STANDARD. Recuperado de https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard


https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

Ruiz Somoyar, J. A. (2016). *Monitoreo en la ganadería bovina a través de redes GSM/GPRS*. Medellín. Recuperado de https://repository.usta.edu.co/bitstream/handle/11634/1126/Monitoreo en la ganadería bovina a través de redes gsm gprs.pdf?sequence=1


ANEXOS

Anexo A. Librería “DHT22.h”

/*
* File:    DHT22.h
* Author:  Daniela Soler y Sara Roman
* Comments: -
* Revision history: October 24, 2020
*/

// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef DHT22_H
#define DHT22_H

#include <xc.h>

#define OneWire_DHT_TRIS TRISCbits.RC0
#define OneWire_DHT_PORT PORTCbits.RC0
#define OneWire_DHT_LAT LATCbits.LC0

//************************** MACROS **************************

//Macro Pin C2 para lectura de sensor dht22
#define OneWire_DHT_TRIS TRISCbits.RC0 //Definir entrada salida
#define OneWire_DHT_PORT PORTCbits.RC0 //Lectura del puerto
#define OneWire_DHT_LAT LATCbits.LC0 //Escritura del puerto

//************************** VARIABLES **************************

unsigned short int check_DHT = 0; // Respuesta sensor DHT -- VALIDEZ DATO
int data_temp = 0;       // Variable de almacenamiento Tem. Amb.
int data_humedad = 0;   // Variable de almacenamiento Humedad

//************************** FUNCIONES **************************

void DHT_inicio(void);
int Leer_DHT(int *dht_tem, int *dht_hum);

#endif /* DHT22_H */

Anexo B. Librería "DHT22.c"

/*
* File:    DHT22.c
* Author:  Daniela Soler y Sara Roman
* Comments: -
* Revision history: October 24, 2020
*/

#include <xc.h>
#include "DHT22.h"
#define _XTAL_FREQ 16000000

//************************** Configuración registros **************************

void DHT_inicio(void){
    OneWire_DHT_TRIS = 0; //Pin como salida
    OneWire_DHT_LAT = 1; //Salida en 1

//******************* Funciones *****************************

int Leer_DHT(int *dht_tem, int *dht_hum) {
//Función para manejo del sensor DHT22
//Se envía la posición de memoria donde se va a guardar los datos:
// - temperatura -> XXX
// - humedad  -> XXX
//Error 2: El sensor no respondió en el tiempo
//Error 3: Los datos leídos tienen error de paridad

    //Variables DHT22
    unsigned short int Data_DHT[5];
    unsigned short int Lectura_DHT = 0;
    int Temperatura_16b = 0;

    //1. Señal de inicio -> 18 ms en 0 y 40us en 1
    OneWire_DHT_LAT = 0; //Nivel bajo
    __delay_ms(18);
    OneWire_DHT_LAT = 1; //Nivel alto
    __delay_us(40);

    //2. Pin como entrada
    OneWire_DHT_TRIS = 1; //Pin como entrada

    //3. Señal de control, respuesta en 80us en 0 y 80 us en 1
    //Error 2: El sensor no respondió en el tiempo
    if (OneWire_DHT_PORT == 1) {
        OneWire_DHT_TRIS = 0; //Pin como salida
        OneWire_DHT_PORT = 1; //Nivel alto
        return 2;
    }
    __delay_us(80);

    if (OneWire_DHT_PORT == 0) {
        OneWire_DHT_TRIS = 0; //Pin como salida
        OneWire_DHT_PORT = 1; //Nivel alto
        return 2;
    }
    __delay_us(80);

    //4. Lectura de los 40 datos: 2 byte de humedad, 2 de temperatura, 1 de control de paridad
    for (int j = 0; j < 5; j++) {
        Lectura_DHT = 0;
        for (int i = 0; i < 8; i++) {
            while (OneWire_DHT_PORT == 0); //Espera Flanco de subida
            __delay_us(45); //Tiempo mayor a 0 y menor a 1 en el protocolo
        }
    }

    return 0;
}

// ******************** Funciones ********************
if (OneWire_DHT_PORT == 1) { //Almacena en l en la posición 7-1 del array
    Lectura_DHT |= (l << (7 - i));
} while (OneWire_DHT_PORT == 1); //Espera flanco de bajada
Data_DHT[j] = Lectura_DHT;

//5. Final del protocolo, pin como salida y en valor 1
__delay_us(50);
OneWire_DHT_TRIS = 0; //Pin como salida
OneWire_DHT_PORT = 1; //Nivel alto

//6. verificación de byte de paridad
//Error 3: Los datos leídos tienen error de paridad
    Temperatura_16b = (Data_DHT[2] << 8) | Data_DHT[3];
    if (Temperatura_16b >= 0x8000) {
        Temperatura_16b = 0x8000 - Temperatura_16b;
        *dht_tem = Temperatura_16b;
    } else {
        *dht_tem = Temperatura_16b;
    }
    *dht_hum = (Data_DHT[0] << 8) | Data_DHT[1];
    return 1;
}
return 3;

Anexo C. Librería "UART_18f46k22.h"
/*
 * File: UART_18f46k22.h
 * Author: Daniela Soler y Sara Roman
 * Comments: -
 * Revision history: October 24, 2020
 */

/* This is a guard condition so that contents of this file are not included
 // more than once.
 ifndef UART_18F46K22_H
 define UART_18F46K22_H

 #include <xc.h>

 //************************************************** MACROS **************************************************

 //Macro Pin B2 y B3 indicador de comunicacion serial 1
 define UART_RX1_Tris TRISBbits.RB1
 define UART_RX1_LAT LATBbits.LB1
#define UART_TX1_Tris TRISBbits.RB2
#define UART_TX1_LAT LATBbits.LB2

//Macro Pin B2 y B3 indicador de comunicacion serial 2
#define UART_RX2_Tris TRISBbits.RB3
#define UART_RX2_LAT LATBbits.LB3
#define UART_TX2_Tris TRISBbits.RB4
#define UART_TX2_LAT LATBbits.LB4

//Macro Deshabilitar puertos UART
#define UART_1_OFF PMD0bits.UART1MD
#define UART_2_OFF PMD0bits.UART2MD

//**************************** VARIABLES *******************************

//UART 1
char leer_1 = '\0';  // Variable para leer serial 1
char buff_1[100];    // Buffer para guardar la info que llega por serial 1
char mensaje_1[100]; // Mensaje recibido por serial 1
unsigned short int index_1 = 0; // Posicion vector buffer 1

//UART 2
char leer_2 = '\0';  // Variable para leer serial 2
char buff_2[100];    // Buffer para guardar la info que llega por serial 2
char mensaje_2[100]; // Mensaje recibido por serial 2
unsigned short int index_2 = 0; // Posicion vector buffer 2

//**************************** FUNCIONES *******************************

void UART_1_inicio(void);  // Configuraciones serial 1
void UART_2_inicio(void);  // Configuraciones serial 2
void UART_1_write(char dato);  // envió de un carácter por el serial 1
void UART_2_write(char dato);  // envió de un carácter por el serial 2
void UART_1_enviar(char *cadena);  // envió de una cadena por el serial 1
void UART_2_enviar(char *cadena);  // envió de una cadena por el serial 2

#endif /* UART_18F46K22_H */
Anexo D. Librería "UART_18f46k22.c"

/**
 * File:   UART_18f46k22.c
 * Author: Daniela Soler y Sara Roman
 * Comments: -
 * Revision history: October 24, 2020
 */

#include <xc.h>
#include "UART_18f46k22.h"

//******************** configuración registros *******************
void UART_1_inicio(void){
    //Registros para UART 1
    BAUDCON1bits.BRG16 = 0; // configuración de 8 bit
    SPBRGH1 = 0;
    SPBRC1 = 103; // Registros para definir velocidad:
    9600
    TXSTA1bits.SYNC = 0; // Modo asíncrono
    TXSTA1bits.BRGH = 1; // High speed de transmisión
    TXSTA1bits.TXEN = 1; // Habilitar bit de transmisión de datos
    RCSTA1bits.SPEN = 1; // Habilitar Puerto serial
    RCSTA1bits.CREN = 1; // Habilitar transmisión de datos
    TRISCbits.TRISC6 = 0; // Puerto tx1 como salida
    TRISCbits.TRISC7 = 1; // Puerto rx1 como entrada
    //led de control UART 1
    UART_TX1_Tris = 0; // definición salida - Led de Escritura
    UART_TX1_LAT = 0; // Salida en 0
    UART_RX1_Tris = 0; // definición salida - Led de lectura
    UART_RX1_LAT = 0; // Salida en 0
}

void UART_2_inicio(void){
    //Registros para UART 2
    BAUDCON2bits.BRG16 = 0; // configuración de 8 bits
    SPBRGH2 = 0;
    SPBRC2 = 103; // Registros para definir velocidad:
    9600
    TXSTA2bits.SYNC = 0; // Modo asíncrono
    TXSTA2bits.BRGH = 1; // High speed de transmisión
    TXSTA2bits.TXEN = 1; // Habilitar bit de transmisión de datos
    RCSTA2bits.SPEN = 1; // Habilitar Puerto serial
    RCSTA2bits.CREN = 1; // Habilitar transmisión de datos
    TRISDbits.TRISD6 = 0; // Puerto tx2 como salida
    TRISDbits.TRISD7 = 1; // Puerto rx2 como entrada
//led de control UART 2
UART_TX2_Trisk = 0;  // definición salida - Led de Escritura
UART_TX2_LAT = 0;    // Salida en 0
UART_RX2_Trisk = 0;  // definición salida - Led de lectura
UART_RX2_LAT = 0;    // Salida en 0
}

/****************** Funciones **************************

void UART_1_write(char dato) {
    //FUNCION PARA ENVIAR UN CARACTER (8BITS) POR EL SERIAL 1
    //Recibe el caracter a enviar
    while (!PIR1bits.TX1IF);  //Espera mientras el buffer este ocupado
    TXREG1 = dato;          //envia el dato
}

void UART_2_write(char dato) {
    //FUNCION PARA ENVIAR UN CARACTER (8BITS) POR SERIAL
    //Recibe el caracter a enviar
    while (!PIR3bits.TX2IF);  //Espera mientras el buffer este ocupado
    TXREG2 = dato;          //envia el dato
}

void UART_1_enviar(char *cadena){
    //FUNCION PARA ENVIAR UNA CADENA DE CARACTERES POR EL SERIAL 1
    //Recibe la posición de memoria inicial de la cadena a enviar

    UART_TX1_LAT = 1;
    while(*cadena){
        UART_1_write(*cadena);
        cadena++;
    }
    UART_TX1_LAT = 0;
}

void UART_2_enviar(char *cadena){
    //FUNCION PARA ENVIAR UNA CADENA DE CARACTERES POR EL SERIAL 2
    //Recibe la posición de memoria inicial de la cadena a enviar

    UART_TX2_LAT = 1;
    while(*cadena){
        UART_2_write(*cadena);
        cadena++;
    }
    UART_TX2_LAT = 0;
}

Anexo E. Librería "I2C_18f46k22.h"
/*
* File:  I2C_18f46k22.h
// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef I2C_18F46K22_H
#define I2C_18F46K22_H

#include <xc.h>

//**************************** MACROS **********************************
//**************************** VARIABLES *******************************
//**************************** FUNCIONES *******************************
void I2C_inicio(void);
void I2C_Start(void);
void I2C_Stop(void);
void I2C_Restart(void);
int I2C_write(char dato);
char I2C_read(void);
void I2C_ACK(void);
void I2C_NACK(void);
#endif /* I2C_18F46K22_H */

Anexo F. Librería "I2C_18f46k22.c"
/*
 * File:   I2C_18f46k22.c
 * Author: Daniela Soler y Sara Roman
 * Comments: -
 * Revision history: October 24, 2020
 */

#include <xc.h>
#include "I2C_18f46k22.h"

//******************** configuración registros ***********************

void I2C_inicio(void){
    // Registros I2C
    PORTCbits.RC3 = 1;       // Pin SCL1 como entrada
    PORTCbits.RC4 = 1;       // Pin SDA1 como entrada
    SSP1STATbits.SMP = 1;    // Deshabilitar control de velocidad
    SSP1CON1bits.SSPEN1 = 1; // Habilitar puerto I2C
    SSP1CON1bits.SSPM = 0b1000; // Habilitar velocidad de transmisión
}
SSP1CON2 = 0x00; // Registros de funcionamiento
I2C
SSP1ADD = 0x27; // Velocidad de transmisión I2C

//*************************** Funciones *******************************

void I2C_Start(void){
    SSP1CON2bits.SEN = 1; // señal de inicio de comunicación
    while(SSP1CON2bits.SEN);
}

void I2C_Stop(void){
    SSP1CON2bits.PEN = 1; // Señal de stop
    while(SSP1CON2bits.PEN);
}

void I2C_Restart(void){
    SSP1CON2bits.RSEN = 1; // Señal de re-estar
    while(SSP1CON2bits.RSEN);
}

int I2C_write(char dato){
    PIR1bits.SSP1IF = 0; // Limpiar bit de interrupción
    SSP1BUF = dato; // envió de Byte
    while(!PIR1bits.SSP1IF); // Esperar a fin de la transmisión
    if(SSP1CON2bits.ACKSTAT == 1){ // Respuesta -ACK del esclavo
        return 1; // Respondio
    }
    else{
        return 0; // No respondió
    }
}

char I2C_read(void){
    PIR1bits.SSP1IF = 0; // Limpiar bit de interrupcion
    SSP1CON2bits.RCEN = 1; // Habilitar lectura de datos
    while(!PIR1bits.SSP1IF);
    return SSP1BUF; // Dato recibido
}

void I2C_ACK(void){
    PIR1bits.SSP1IF = 0; // Limpiar bit de interrupcion
    SSP1CON2bits.ACKDT = 0; // Bit Acknowledge
    SSP1CON2bits.ACKEN = 1; // Trasmision bit ACKDT
    while(!PIR1bits.SSP1IF); // Esperar a fin de trasmision
}

void I2C_NACK(void){
    PIR1bits.SSP1IF = 0; // Limpiar bit de interrupcion
    SSP1CON2bits.ACKDT = 1; // Bit no Acknowledge
    SSP1CON2bits.ACKEN = 1; // Trasmision bit ACKDT
    while(!PIR1bits.SSP1IF); // Esperar a fin de trasmision
Anexo G. Librería "LM75A.h"

/ * File:   LM75A.h
* Author: Daniela Soler y Sara Roman
* Comments: -
* Revision history: October 24, 2020
*/

// This is a guard condition so that contents of this file are not included
// more than once.
#ifndef LM75A_H
#define LM75A_H
#include <xc.h>

//******************************* MACROS *******************************
#define Direccion_LM75A 0b1001000
#define Comando_W_LM75A (Direccion_LM75A << 1 | 0x00)
#define Comando_R_LM75A (Direccion_LM75A << 1 | 0x01)

//******************************* VARIABLES *******************************

//******************************* FUNCIONES *******************************

void LM75A_Start(void); // configuración del sensor
float LM75A_read(void);   // Lectura del sensor
void LM75A_Stop(void);    // apagado modulo

#endif /* LM75A_H */

Anexo H. Librería "LM785A.c"

/*
 * File:   LM75A.c
 * Author: Daniela Soler y Sara Roman
 * Comments: -
 * Revision history: October 24, 2020
 */
#include <xc.h>
#include "LM75A.h"
#include "I2C_18f46k22.h"

/******************************* Funciones *******************************

void LM75A_Start(void){
    I2C_Start();
}
I2C_write(Comando_W_LM75A);
I2C_write(0x01);
I2C_write(0x18);
I2C_Stop();
return;
}

float LM75A_read(void){
    unsigned short int MSB = 0;
    unsigned short int LSB = 0;
    int temp = 0;
    float temp_real = 0;

    // Paso 1: Start y escritura de registro Temp
    I2C_Start();
    I2C_write(Comando_W_LM75A);
    I2C_write(0x00);
    // Paso 2: Re-start y lectura del registro
    I2C_Restart();
    I2C_write(Comando_R_LM75A);
    // Paso 3: Lectura MSB
    MSB = I2C_read();
    // Paso 4: Master acknowledge
    I2C_ACK();
    // Paso 5: Lectura LSB
    LSB = I2C_read();
    // Paso 6: Master not acknowledge
    I2C_NACK();
    // Paso 7: Stop
    I2C_Stop();

    // Paso 8: Calculo temperatura

    temp = (MSB << 3) | (LSB>>5);
    if(temp & 0x400){
        temp = temp & 0x3FF;
        temp = (~temp) + 0x01;
        temp_real = temp * 0.125 * (-1);
    } else {
        temp_real = temp * 0.125;
    }
    return temp_real;
}

void LM75A_Stop(void){
    I2C_Start();
    I2C_write(Comando_W_LM75A);
    I2C_write(0x01);
    I2C_write(0x19);
    I2C_Stop();
    return;
}
Anexo I. Código en C de la adquisición y envío de datos al servidor en PIC18F46K22

/*
 * File: t_main.c
 * Author: Daniela Soler y Sara Román
 * Created on October 2, 2020, 1:17 PM
 */

// PIC18F46K22 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1H
#pragma config FOSC = HSMP // Oscillator Selection bits (HS oscillator (medium power 4-16 MHz))
#pragma config PLLCFG = OFF // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON // Primary clock enable bit (Primary clock is always enabled)
#pragma config FCEN = OFF // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config PWRTEN = OFF // Power-up Timer Enable bit (Power up timer disabled)
#pragma config BOREN = SBORDI // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 190 // Brown Out Reset Voltage bits (VBOR set to 1.90 V nominal)

// CONFIG2H
#pragma config WDTE = OFF // Watchdog Timer Enable bits (Watch dog timer is always disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768 // Watchdog Timer Post Scale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC1 // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = ON // PORTB A/D Enable bit (PORTB<5:0> pins are configured as analog input channels on Reset)
#pragma config CCP3MX = PORTB5 // P3A/CCP3 Mux bit (P3A/CCP3 input/output is multiplexed with RB5)
#pragma config HFOFST = ON // HFINTOSC Fast Start-up (HFINTOSC output and ready status are not delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0 // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTD2 // ECCP2 B output mux bit (P2B is on RD2)
#pragma config MCRL = EXTMCLR // MCLR Pin Enable bit (MCLR pin enabled, RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON  // Stack Full/Underflow Reset Enable bit
(Stack full/underflow will cause Reset)
#pragma config LVP = ON  // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config XINST = OFF  // Extended Instruction Set Enable bit
(Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF  // Code Protection Block 0 (Block 0 (000800-003FFFh) not code-protected)
#pragma config CP1 = OFF  // Code Protection Block 1 (Block 1 (004000-007FFFh) not code-protected)
#pragma config CP2 = OFF  // Code Protection Block 2 (Block 2 (008000-00BFFFh) not code-protected)
#pragma config CP3 = OFF  // Code Protection Block 3 (Block 3 (00C000-00FFFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF  // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF  // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF  // Write Protection Block 0 (Block 0 (000800-003FFFh) not write-protected)
#pragma config WRT1 = OFF  // Write Protection Block 1 (Block 1 (004000-007FFFh) not write-protected)
#pragma config WRT2 = OFF  // Write Protection Block 2 (Block 2 (008000-00BFFFh) not write-protected)
#pragma config WRT3 = OFF  // Write Protection Block 3 (Block 3 (00C000-00FFFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF  // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF  // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF  // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF  // Table Read Protection Block 0 (Block 0 (000800-003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF  // Table Read Protection Block 1 (Block 1 (004000-007FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF  // Table Read Protection Block 2 (Block 2 (008000-00BFFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF  // Table Read Protection Block 3 (Block 3 (00C000-00FFFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF  // Boot Block Table Read Protection bit
(Boot Block (000000-0007FFh) not protected from table reads executed in
other blocks)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

// ************************** LIBRERIAS *******************************
#include <xc.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define _XTAL_FREQ 16000000

// Librerías periféricos
#include "UART_18f46k22.h"    // Librería UART
#include "I2C_18f46k22.h"     // Librería I2C
#include "DHT22.h"            // Librería sensor DHT22
#include "LM75A.h"            // Librería sensor LM75A

//************************** Macros *******************************
//************************** VARIABLES *******************************

// Almacenamiento
int id = 107;  // ID dispositivo
char Fecha[20][19];    // Matriz para almacenar fecha
int TemAmb[20];  // Vector para almacenar Temperatura Ambiente
int Humedad[20];  // vector para almacenar Humedad Ambiente
char TempCor[62];  // Vector para almacenar Temperatura Corporal
char Latitud[20][10]; // Vector para almacenar Latitud
char LatEmi[20];  // Vector para almacenar hemisferio de la latitud
char Longitud[20][10]; // Vector para almacenar longitude
char LonEmi[20];  // Vector para almacenar hemisferio de la longitud
int div_GPRMC[12];  // Vector Posición de ',' en la sentencia NMEA
int mensaje_div[12]; // Vector de divisiones en la sentencia NMEA

// Almacenamiento de conversión números a cadenas
char cId[4];  // Id del dispositivo
char cTC[4];  // Temperatura corporal
char cTA[4];  // Temperatura Ambiente
char cHum[3]; // Humedad

// Principal
int cont = 0;
//**************************** FUNCIONES *******************************

// General
void inicio(void); // Inicialización de las variables del sistema
void borrar(void); // Borrar array
void guardar(int dato, int *posicion);

// GPS - L70
void GPS_init(void); // Configuración y conexión inicial
texto
int checksum(int modo, char *mensaje, char checksum_mensaje[2]); // Función para verificación y cálculo de checksum.
int GPS_buscar(char *cadena, int *posicion, char simbolo); // Buscar posición de x simbolo en la sentencia NMEA
void GPS_guardar(int indicador, char *Data); // Guardar en la posición indicada los datos
int GPS_lectura(int nSec); // Función para lectura y almacenamiento del GPS
void GPS_apagado(void); // Modo de sleep

// M95
void M95_Comunicacion(int numTramas); //Comunicación con el servidor
void M95_trama(int contTrama); //Generación y envío de Yeison

//**************************** PROGRAMA *******************************

int main(void) {
    OSCCONbits.SCS = 0b00; // Uso de cristal

    //CONTROL PINES ANALOGOS
    ANSELA = 0;
    ANSELB = 0;
    ANSELC = 0;
    ANSELD = 0;
    ANSELE = 0;

    // Inicio de periféricos
    UART_1_inicio();
    UART_2_inicio();
    I2C_inicio();

    //Habilitar interrupción
    PIE1bits.RC1IE = 1; // Habilitar interrupción por RX1
    PIE3bits.RC2IE = 1; // Habilitar interrupción por RX2
    INTCONbits.GIE = 1; //Habilitar interrupciones globales
    INTCONbits.PEIE = 1; //Habilitar interrupciones de periféricos

    //Inicio macros
/inicilizaciones variables
inicio();
borrar();

// Inicio sensores
GPS_init();
DHT_inicio();

while(1){
    UART_1_enviar(" Inicio ");
    sprintf(cId, "%d ", cont);
    UART_1_enviar(&cId[0]);

    // L70
    GPS_lectura(cont);
    UART_1_enviar("L70 ");

    // dht22
    Leer_DHT(&data_temp, &data_humedad);
    guardar(data_temp, &TemAmb[cont]);
    guardar(data_humedad, &Humedad[cont]);
    UART_1_enviar("DHT ");

    // lm75a
    LM75A_Start();
    float valor = LM75A_read() * 10;
    char cValor[4];
    int aux_tc = cont * 3;
    sprintf(cValor, "%.0f ", valor);
    for(int c = 0; c < 3; c++ ){
        TempCor[aux_tc] = cValor[c];
        aux_tc = aux_tc + 1;
    }
    LM75A_Stop();
    UART_1_enviar("LM75A ");

    // enviado de datos al servidor cada 2 horas
    if(cont >= 12){
        for(int d = 0; d < 12; d++){
            M95_trama(d);
            UART_1_enviar("\r\n");
        }
        cont = -1;
    }
}

// Esperar 8 minutos para tomar datos
for(int a = 0; a < 60; a++){
    __delay_ms(10000);
}
__delay_ms(10000);
cont = cont+1;
void inicio(void){
    //FUNCION PARA INICIALIZAR ALMACENAMIENTO DATOS

    // Inicio variables numéricas
    for (int n = 0; n < sizeof(TemAmb); n++) {
        TemAmb[n] = 0;
        Humedad[n] = 0;
    }

    for (int n = 0; n < sizeof(TempCor); n++) {
        TempCor[n] = '0';
    }

    // Inicio variables no numéricas
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++){
            Latitud[i][j] = '\0';
            Longitud[i][j] = '\0';
        }
        LonEmi[i] = '\0';
        LatEmi[i] = '\0';
    }

    //Variables DHT22
    check_DHT = 0;
    data_temp = 0;
    data_humedad = 0;

    //variables GPS
    for (int n = 0; n < sizeof(mensaje_div); n++) {
        mensaje_div[n] = 0;
    }
}

void borrar(void) {
    //FUNCION PARA BORRAR LAS VARIABLES DE LECTURA DE SERIAL
    for (int n = 0; n < sizeof(buff_1); n++) {
        buff_1[n] = '\0';
        mensaje_1[n] = '\0';
        buff_2[n] = '\0';
        mensaje_2[n] = '\0';
    }
    index_1 = 0;
    index_2 = 0;
}

void guardar(int dato, int *posicion){
    *posicion = dato;
}
void GPS_init(void){
    //inicialización modulo GPS
    unsigned short int com_gps = 1;       //Duración de la comunicación GPS
    unsigned short int cont6 = 0;        //Contador switch inicio GPS
    unsigned short int aux_men = 0;      //auxiliar mensaje
    int pos_sep[12];                     //Arrey para guardar posiciones de separadores
    int cantidad = 0;                     //resultado función buscar GPS

    while(com_gps){
        switch (cont6){
            case 1:
                //envió de datos cada 10 segundos para no generar errores en la configuración
                UART_2_enviar("$PMTK220,10000*2F\r\n");
                if((strstr(mensaje_2,"PMTK001,220,3*30") != NULL)){
                    borrar();
                    cont6 = 2;
                }
                __delay_ms(2000);
                break;

            case 2:
                // Solo enviar la sentencia NMEA tipo GPRMC
                UART_2_enviar("$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n");
                if((strstr(mensaje_2,"PMTK001,314,3*36") != NULL)){
                    cont6 = 3;
                    borrar();
                }
                __delay_ms(1000);
                break;

            case 3:
                //Configuración Hot Start
                UART_2_enviar("$PMTK101*32\r\n");
                if (aux_men == 2) {
                    cont6 = 4;
                    borrar();
                    aux_men = 0;
                }
                __delay_ms(2000);
                aux_men++;
                break;

            case 4:
                // envió de datos cada 0.2 segundos
                cont6 = 4;
                UART_2_enviar("$PMTK220,200*2C\r\n");
                borrar();
                cont6 = 5;
                __delay_ms(1000);
        }
    }
}
break;

case 5:
    //Comprobar Dato valido
    cantidad = GPS_buscar(&mensaje_2[0],&pos_sep[0],',','');
    if( mensaje_2[pos_sep[1]+1] == 'A'){
        cont6 = 6;
    }else{
        cont6 = 5;
    }
    __delay_ms(199);
    break;

case 6:
    //Al menos 10 veces se ingrese un dato valido
    aux_men++;
    if(aux_men == 10){
        cont6 = 7;
        aux_men = 0;
    } else {
        cont6 = 5;
    }
    break;

case 7:
    //Envía comando para pausar GPS
    UART_2_enviar("$PMTK161,0*28\r\n");
    cont6 = 8;
    __delay_ms(500);
    break;

    case 8:
    com_gps = 0;
    borrar();

default:
    cont6 = 1;

}
return;

int checksum(int modo, char *mensaje, char checksum_mensaje[2]){
    // Función para cálculo de checksum y validar mensaje
    // Parámetros de entrada:
    // Variable modo = 0 cálculo del Checksum, modo = 1 validez del mensaje
    // Posición de memoria del mensaje
    // checksum_mensaje recibido desde el sensor
    // Retorna el valor del Checksum del mensaje si el modo está en 0, de otra
    // manera retorna 1 si es válido el mensaje y 0 si no lo es.

    int valor = 0;
while(*mensaje){
    valor ^= *mensaje;
    mensaje++;
}
if(modo == 0){ //cálculo del Checksum
    return valor;
} else { //Validez del mensaje
    char svalor[2];
    for(int i = 2; i <100; i++){
        checksum_mensaje[i] = '\0';
    }
    sprintf(svalor, "%X", valor);
    if(strcmp(checksum_mensaje, svalor) == 0){
        return 1;
    } else {
        return 0;
    }
}
return 0;
}

int GPS_buscar(char *cadena, int *posicion, char simbolo){
    //Función Buscar las posiciones donde se encuentre la variable simbolo
    //Devuelve la cantidad de símbolos en la sentencia
    //Como parámetro se entrega:
    // - *cadena -> Posición de memoria del primer carácter del mensaje
    // - *posicion -> Posición de memoria donde almacenar las posiciones
    // - Simbolo -> Carácter a buscar en cadena
    int cont_pos = 0;
    int cantidad = 0;
    while(*cadena){
        if(*cadena == simbolo){
            *posicion = cont_pos;
            posicion++;
            cantidad++;
        }
        cadena++;
        cont_pos++;
    }
    return (cantidad);
}

void GPS_guardar(int indicador, char Data[100]){ // Función para separar y guardar los datos de la sentencia MNEA
    // Recibe como parámetro:
    // - indicador: index del dato a guardar
    // - Data[100]: Cadena de 100 caracteres donde se encuentra la sentencia MNEA
    // No retorna ningún dato

    //FECHA
GPS_buscar(&Data[0],&mensaje_div[0],',');  //Encontrar separaciones
  int ind_copia1 = mensaje_div[8]+1;  //Separación 8 para fecha
  int ind_copia2 = mensaje_div[0]+1;  //separación 0 para hora

  //Almacenamiento de Fecha y hora
  for(int j = 0; j < 19; j++){
    if(j == 2 || j == 5 || j == 10 ){
      Fecha[indicador][j]= '—';
    } else if( j == 13 || j == 16){
      Fecha[indicador][j]= ':';
    }else if(j < 10){
      Fecha[indicador][j]= Data[ind_copia1];
      ind_copia1++;
    } else {
      Fecha[indicador][j]= Data[ind_copia2];
      ind_copia2++;
    }
  }

  //LATITUD - LONGITUD
  ind_copia1 = mensaje_div[2]+1;  //Separación latitud
  ind_copia2 = mensaje_div[4]+2;  //Separación longitud
  for(int j = 0; j < 10; j++){
    if(j == 2){
      Latitud[indicador][j] = '—';
      Longitud[indicador][j] = '—';
    } else {
      Latitud[indicador][j]= Data[ind_copia1];
      Longitud[indicador][j]= Data[ind_copia2];
      ind_copia1++;
      ind_copia2++;
    }
  }

  //HEMISFERIOS
  LatEmi[indicador] = mensaje_div[3]+1;  //Separación hemisferio N-S
}

int GPS_lectura(int nSec){
  // Función para lectura de las variables GPS
  // Parámetros de entrada:
  // - nSec: Secuencia de entrada para el almacenamiento de datos
  // Retorna:
// 1: Si la cadena leída del L70 es válida
// 0: Si la cadena leída del L70 no es válida

// Paso 0: variables locales
char mensaje_2_copy[100]; // Almacenamiento de
Sentencia NMEA
char mensaje_2_CH[2] = {'\0','\0'}; // Almacenamiento de
Checksum de entrada
int posicion_final[4]; // Almacenamiento de *

// Paso 1: envió de carácter para comenzar el módulo L-70
UART_2_enviar("start");
__delay_ms(300);
// Paso 2: Esperar que el módulo me entregue un valor válido
while(mensaje_2[mensaje_div[1]+1] != 'A'){
    GPS_buscar(&mensaje_2[0],&mensaje_div[0],',');
    __delay_ms(199);
}
// Paso 3: Copiar Mensaje de entrada en la variable 'mensaje_2_copy'
strcpy(mensaje_2_copy,mensaje_2);
__delay_ms(200);
// Paso 4: envió de sentencia para sleep mode
UART_2_enviar("$PMTK161,0*28r
");
__delay_ms(300);

// Paso 5: verificación de Checksum y almacenamiento de datos
// Paso 5.1: verificación de "GPRMC"
if strstr(mensaje_2_copy,"GPRMC") != NULL){
    // Paso 5.2: preparación de datos para verificación de CH
    mensaje_2_copy[0] = ' ';
    GPS_buscar(&mensaje_2_copy[0], &posicion_final[0], '*');
    mensaje_2_CH[0]= mensaje_2_copy[posicion_final[0]+1];
    mensaje_2_CH[1]= mensaje_2_copy[posicion_final[0]+2];
    for(int a = posicion_final[0]; a < sizeof(mensaje_2_copy) ; a++){
        mensaje_2_copy[a] = '\0';
    }
    // Paso 5.3: Validación del CH de entrada
    if(checksum(1,&mensaje_2_copy[1], mensaje_2_CH) == 1){
        GPS_guardar(nSec, mensaje_2_copy);
        return 1;
    }else{
        return 0;
    }
} else {
    return 0;
}
return 0;
void M95_Comunicacion(int numTramas) {
    //FUNCIÓN PARA COMUNICACIÓN CON EL SERVIDOR
    //SE ENVIA EL NUMERO DE TRAMAS PARA ENVIAR

    unsigned short int aux_M95_caso = 1; // Contador Switch
    unsigned short int aux_M95_on = 1; // comunicación con servidor
    unsigned short int cont_trama = 0; // Contador comando tramas
    unsigned short int cont_M95 = 0; // Indicador comparación de mensaje

    //Comprobación

    while (aux_M95_on) {
        switch (aux_M95_caso) {
        case 1: // AT
            borrar();
            UART_1_enviar("AT\r\n");
            __delay_ms(2000);
            if (strstr(mensaje_1,"OK") != NULL) {
                aux_M95_caso = 2;
            }
            borrar();
            break;

        case 2: // At+cops? Conexión a red móvil
            borrar();
            UART_1_enviar("AT+COPS?\r\n");
            __delay_ms(2000);
            if (strcmp(mensaje_1,"AT+COPS?\r\n\n+COPS: 0,0,"Claro"\r\n\nOK\r\n\n") == 0) {
                aux_M95_caso = 3;
            }
            borrar();
            break;

        case 3: // at+QICSGP conexión a APN
            borrar();
            UART_1_enviar("AT+QICSGP=1,\"Internet.comcel.com.co\"\r\n");
            __delay_ms(2000);
            if (strstr(mensaje_1,"OK") != NULL) {
                aux_M95_caso = 4;
            } else {
                aux_M95_caso = 2;
            }
            borrar();
            break;

        case 4: // at+QHTTPURL conexión a servidor
            if (cont_M95 == 0) {
                borrar();
            }
        }
    }
UART_1_enviar("AT+QHTTPURL=42\r\n");
}
cont_M95 = 1;
__delay_us(100);

if (strcmp(buff_1, "AT+QHTTPURL=42\r\r\nCONNECT\r\n") == 0) {
    aux_M95_caso = 5;
    cont_M95 = 0;
}
break;

case 5: // Enlace del servidor
    __delay_us(50);
UART_1_enviar("https://deploytesis.herokuapp.com/medicion");
    __delay_ms(3000);
if (strstr(mensaje_1, "OK") != NULL) {
    aux_M95_caso = 6;
} else {
    aux_M95_caso = 4;
}
borrar();
break;

case 6:// at+QHTTPPOST post en servidor
if (cont_M95 == 0) {
    borrar();
    UART_1_enviar("AT+QHTTPPOST=118\r\n");
}
cont_M95++;
__delay_ms(1);

if (cont_M95 == 60000) {
    //Se el servidor se demora o si hay una llamada al
    //tiempo de espera es de 1 min.
    cont_M95 = 0;
    borrar();
}

if (strcmp(buff_1, "AT+QHTTPPOST=118\r\r\nCONNECT\r\n") == 0) {
    aux_M95_caso = 7;
    cont_M95 = 0;
}
break;

case 7:// enviado de trama
if (cont_trama > numTramas - 1) {
    cont_trama = 0;
}
M95_trama(cont_trama);
__delay_ms(3000);
if (strstr(mensaje_1,"OK") != NULL) {
    aux_M95_caso = 8;
} else {
    aux_M95_caso = 6;
}
borrar();
delay_ms(2000);
break;

case 8: //AT+QHTTPREAD leer respuesta servidor
    UART_1_enviar("AT+QHTTPREAD\r\n");
    delay_ms(3500);
    if (strstr(mensaje_1, "dato guardado") != NULL) {
        aux_M95_caso = 9;
    } else {
        aux_M95_caso = 6;
    }
borrar();
break;

case 9: // cambiar trama
    if (cont_trama < numTramas - 1) {
        cont_trama++;
        aux_M95_caso = 6;
    } else {
        aux_M95_caso = 10;
        cont_trama = 0;
    }
borrar();
break;

case 10: // at+QIDEACT desconexión del servidor
    borrar();
    UART_1_enviar("AT+QIDEACT\r\n");
    delay_ms(3500);
    cont_M95++;  
    if (strcmp(mensaje_1, "AT+QIDEACT\r\r\nDEACT OK\r\n\n") == 0) {
        delay_ms(10000);
        aux_M95_on = 0;
        cont_M95 = 0;
    }
    if (cont_M95 == 2) {
        delay_ms(10000);
        aux_M95_on = 0;
        cont_M95 = 0;
    }
borrar();
break;
default:
    aux_M95_caso = 1;
borrar();
break;
void M95_trama(int contTrama){
    //FUNCION PARA GENERACION DE TRAMAS
    //SE ENVIA EL INDICE DE CUAL TRAMA GENERAR, SE MANDA LA POSICION DEL ARREY DE DATOS

    // CONVERSION DE NUMEROS A CADENA DE CARACTERES
    sprintf(cId, "%d", id);
    sprintf(cTA, "%d", TemAmb[contTrama]);
    sprintf(cHum, "%d", Humedad[contTrama]);

    //ENVIO DE TRAMA
    // "f":"
    UART_1_enviar("{"f":""}"");
    //fecha: DD-MM-AAAA-HH:MM:SS
    for(int i = 0; i < 19; i++){
        UART_1_write(Fecha[contTrama][i]);
    }
    // ",id":
    UART_1_enviar("","id"":""};
    // id:XXX
    UART_1_enviar(&cId[0]);
    //UART_1_enviar("103");
    // ",tc":
    UART_1_enviar("","tc"":""};
    //tem corp: XXX
    UART_TX1_LAT = 1;
    for(int i = 0; i < 3; i++){
        int aux2 = (contTrama * 3)+i;
        UART_1_write(TempCor[aux2]);
    }
    UART_TX1_LAT = 0;
    // ",ta":
    UART_1_enviar("","ta"":""};
    //temp amb: XXX
    UART_1_enviar(&cTA[0]);
    // ",h":
    UART_1_enviar("","h"":""};
    //humedad: XXX
    UART_1_enviar(&cHum[0]);
    // ",la":"
    UART_1_enviar("","la"":""};
    // latitud: XX-XX.XXXX
    UART_TX1_LAT = 1;
    for(int i = 0; i < 10; i++){
        UART_1_write(Latitud[contTrama][i]);
    }
    UART_TX1_LAT = 0;

    return;
}
UART_1_enviar("\n","lae":"\n");

UART_1_write(LatEmi[contTrama]);
UART_1_enviar("N");
UART_1_enviar("\n","loe":"\n");

UART_TX1_LAT = 1;
for (int i = 0; i < 10; i++){
    UART_1_write(Longitud[contTrama][i]);
}
UART_TX1_LAT = 0;
UART_1_enviar("\n","loe":"\n");
UART_1_write(LonEmi[contTrama]);
UART_1_enviar("E");
UART_1_enviar("\n");
return;
}

void __interrupt() isr_routine(void){

//INTERRUPCIÓN EXTERNA COMUNICACIÓN SERIAL 1 - SERVIDOR
if (PIR1bits.RC1IF) {
    UART_RX1_LAT = 1;
    leer_1 = RCREG1; //LEER INFO DEL SERIAL
    if (index_1 < 100) {
        if (leer_1 == 'n') {
            if (buff_1[index_1 - 3] == 'O' && buff_1[index_1 - 2] == 'K' && buff_1[index_1 - 1] == '\r') {
                buff_1[index_1] = '\n';
                index_1 = 0;

                for (int n = 0; n < 100; n++) {
                    mensaje_1[n] = buff_1[n];
                    buff_1[n] = '\0';
                }
            } else {
                buff_1[index_1] = '\n';
                index_1++;
            }
        } else {
            buff_1[index_1] = leer_1; //Llenar buffer
            index_1++;
        }
    } else {
        index_1 = 0; //Reinicializar index
    }

}
UART_RX1_LAT = 0;

//INTERRUPTIÓN EXTERNA COMUNICACIÓN SERIAL 2 - GPS
if (PIR3bits.RC2IF) {
    UART_RX2_LAT = 1;
    leer_2 = RCREG2;  //LEER INFO DEL SERIAL
    if (Index_2 < 100) {
        if (leer_2 == '\n'){
            if (buff_2[index_2 - 1] == '\r') {
                buff_2[index_2] = '\n';
                index_2 = 0;
                for (int n = 0; n < 100; n++) {
                    mensaje_2[n] = buff_2[n];
                    buff_2[n] = '\0';
                }
            } else {  
                buff_2[index_2] = '\n';
                index_2++;
            }
        } else { 
            buff_2[index_2] = leer_2; //Llenar buffer
            index_2++;
        } else {
            index_2 = 0; //Reinicializar index
        }
    }
    UART_RX2_LAT = 0;
}

Anexo J Servidor 1 - Conexión MongoDB y seguridad
var express = require('express');
const bodyParser = require('body-parser');
const { restart } = require('nodemon');
var app = express();
const MongoClient = require('mongodb').MongoClient
const uri ='mongodb+srv://sara-tesis:sar40rom45@tesis-qfte4.mongodb.net/test?retryWrites=true&w=majority'
const PORT = process.env.PORT || 3000

let db;
MongoClient.connect(uri, function (err, client) {
    if (err) {
        console.log('Error occurred while connecting to MongoDB Atlas...\n', err);
    }else{
        console.log('Connected...');
Anexo K Servidor 1.1 - Endpoint POST medición
app.post('/medicín', (req, res) => {
  console.log(req.body);
  const obj1 = JSON.parse(Object.keys(req.body)[0]);
  const { f: fecha, id, tc: temperaturaCorporal, ta: temperaturaAmbiente, h: humedad, la: latitud, lae: hemisferioLatitud, lo: longitud, loe: hemisferioLongitud } = obj1;
  const final = { fecha, id, temperaturaCorporal, temperaturaAmbiente, humedad, latitud, hemisferioLatitud, longitud, hemisferioLongitud };
  db.createCollection(`${id}`).then(() => {
    db.collection(`${id}`).insertOne(final, (err, result) => {
      if (err) return console.log(err);
      res.send('dato guardado')
    })
  });
});

Anexo L Servidor 1.2 - Endpoint GET medición
app.get('/medicín', (req, res) => {
  db.listCollections().toArray().then(collectionsInfo => {
    const collections = collectionsInfo.map(collection => collection.name)
    const result = {};
    let count = 0;
    collections.forEach(async (collection, index, array) => {
      const data = await db.collection(collection).find({}).toArray()
      result[collection] = data;
      count++
      if (array.length === count)
        res.send(JSON.stringify(result));
    });
  });
});
Anexo M. APP datos recibidos y procesamiento de la información

```javascript
useEffect(() => {
  fetch("https://deploytesis.herokuapp.com/medicion").then(response =>
  response.json()).then(response => {
    Object.keys(response).forEach(key => {
      response[key] = {
        heatmapData: {
          positions: response[key].map(element => {
            // procesamiento coordenadas
            const latitud1 = Number(element.latitud.split('-')[0])
            const latitud2 = Number(element.latitud.split('-')[1])
            const lat = latitud1 + latitud2 / 60
            const longitud1 = Number(element.longitud.split('-')[0])
            const longitud2 = Number(element.longitud.split('-')[1])
            const lng = (longitud1 + longitud2 / 60) * -1
            // procesamiento fecha
            const [dia, mes, año, hora] = element.fecha.split('-')
            const fecha = new Date(dia + '-' + mes + '-' + año + '-' + hora)
            fecha.setHours(fecha.getHours() - 5)
            // procesamiento temperatura y humedad
            const temperaturaAmbiente = Number(element.temperaturaAmbiente) / 10
            const humedad = Number(element.humedad) / 10
            const temperaturaCorporal = Number(element.temperaturaCorporal) / 10
            return {
              lat,
              lng,
              fecha,
              temperaturaAmbiente,
              temperaturaCorporal,
              humedad,
              id: element.id
            }
          }),
          options: {
            radius: 20,
            opacity: 20
          }
        }
      })
    })
  });
  Object.keys(response).forEach(key => {
    response[key].pinData = response[key].heatmapData.positions[response[key].heatmapData.positions.length - 1]
  })
  setData(response)
}, [])
```