ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROTOTIPO DE MONITOREO DE TEMPERATURA Y POSICIÓN BASADO EN REDES DE SENSORES INALÁMBRICOS IEEE 802.15.4

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

FABIÁN VINICIO DURANGO PANTOJA

DIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA, MSc.

Quito, junio 2018

AVAL

Certifico que el presente trabajo fue desarrollado por Fabián Vinicio Durango Pantoja, bajo mi supervisión.

Ing. Carlos Roberto Egas Acosta, MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Fabián Vinicio Durango Pantoja, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Fabián Vinicio Durango Pantoja

DEDICATORIA

Dedico este trabajo a mis padres Lígia y Fabián, además de mi abuelita Fanny, quienes han sido mi apoyo a lo largo de este proceso; asimismo, a mi hermana Pamela que me ha incentivado a seguir adelante y cumplir este objetivo.

Fabián

AGRADECIMIENTO

Agradezco primeramente a Dios y a la vida, que me han permitido poder cumplir con este objetivo tan anhelado y por las bendiciones que me han dado a lo largo de mi vida.

Agradezco a mi abuelita Fanny, quien ha sido un apoyo incondicional y me ha formado como la persona que soy.

Agradezco a mis padres Lígia y Fabián, quienes siempre han estado para mí en los buenos y malos momentos, por el apoyo que siempre me han brindado y por el gran ejemplo que han sido para mí.

Agradezco a mi hermana Pamela, quien me ha guiado a ser mejor persona y seguir con mis objetivos.

Agradezco a mis tíos, Marquito, Patricio, Diego, Tula, Fernando, Conny y mi abuelito Delfín quienes siempre me han apoyado y confiado en mí.

Agradezco a todos mis amigos, Anita, Andrés, Javier, José, Carlita, amigos que conocí en esta etapa y que de una u otra manera me apoyaron a seguir adelante y ser mejor persona; agradezco a Ronald, José, Darío, David quienes han sido un apoyo tanto académico como de amistad incondicional, hemos compartido gratos momentos que llevaré por siempre en mi corazón.

Agradezco a Carol, una persona muy especial que me ha sabido ayudar y guiar en este proceso, siempre apoyándome para culminarlo.

Agradezco a mis compañeros, que conocí a lo largo de este tiempo.

Agradezco al Ing. Carlos Egas MSc, quien ha sido mi guía en la elaboración de este proyecto, gracias a su ayuda he podido culminarlo con las expectativas deseadas.

Agradezco la rama estudiantil IEEE EPN y todos los que formamos parte de ella, por los consejos y logros adquiridos en ella.

Agradezco a todos los profesores de la Escuela Politécnica Nacional, que con su sabiduría y conocimiento me han ayudado a tener criterio tanto profesional como personal, para así desempeñarme en el campo laboral.

Fabián

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE SEGMENTOS DE CÓDIGO

RESUMEN

El presente estudio técnico se estableció en el desarrollo de un prototipo de monitoreo de posición y temperatura basado en redes de sensores inalámbricas IEEE 802.15.4, específicamente para ser usado en zonas de difícil acceso en donde se dificulte utilizar un dispositivo cableado y que requiera recarga de energía constante.

Con la finalidad de resolver la necesidad de monitoreo de personas u objetos, se creó el presente prototipo el cual estuvo enmarcado en la creación de una red de sensores inalámbrica IEEE 802.15.4 que consta de varias motas que portan un sensor de temperatura, acelerómetro y GPS; asimismo, de un Waspmote que se conecta a un computador mediante un puerto USB y concentra las comunicaciones; conjuntamente de una aplicación de escritorio la cual permite desplegar sobre un mapa un marcador con la posición de cada mota, al igual que su temperatura y su estado (normal o caído); además, se emite una alarma tanto en la mota como en la aplicación en el caso de sobrepasar el nivel máximo de temperatura el cual puede ser definido por el usuario o encontrarse en estado caído.

De esta manera se pudo concluir que con la implementación de este prototipo se obtuvo un dispositivo autónomo sin necesidad de conectarlo constantemente dado su bajo consumo de energía; además, permite aprender a través del diseño y así contribuir a crear nuevos proyectos basados en este prototipo en un futuro.

PALABRAS CLAVE: IEEE 802.15.4, Prototipo, Waspmote, Red de sensores inalámbrica, Monitoreo, GPS.

ABSTRACT

The present study was established in the development of a prototype of monitoring and positioning based on wireless sensor networks IEEE 802.15.4, specifically to be used in areas where it is difficult to use a wired device that requires constant energy recharge. To solve the need to monitor people or objects, the present prototype was created on wireless sensors network IEEE 802.15.4 that consists of several motes that carry a temperature sensor, accelerometer and GPS; also, a Waspmote that connects to a computer through a USB port.

In addition, a desktop application which allows to display on a map a marker with the position of each mote, as well as it's temperature and its state (normal or fallen); in addition, an alarm is issued both in the mote and in the application in case of exceeding the maximum temperature level which can be defined by the user or be in a fallen state.

In this way it was possible to conclude that the implementation of this prototype, an autonomous device was obtained without the need to connect it constantly given its low energy consumption; In addition, it allows learning through design and thus contribute to create new prototypes in the future.

KEY WORDS: IEEE 802.15.4, Prototype, Waspmote, Wireless sensor network, Monitoring, GPS.

1 INTRODUCCIÓN

Hoy en día las redes de sensores inalámbricas se han convertido en una herramienta muy usada seguramente dado la facilidad que estas tienen para ser manipuladas y el bajo consumo de energía que disponen; estas redes ofrecen la posibilidad de implementar dispositivos con bajo costo y larga duración de batería sin necesidad de estarlo recargando constantemente; así mismo, son capaces de enviar y recibir información del entorno de manera inalámbrica a una tasa de transmisión alta de alrededor de 250 Kbps.

Por este motivo, se realiza un prototipo de monitoreo de temperatura y posición, el cual consta de varios sensores incluidos en las diferentes motas como se muestra en la Figura 1.1; primero, un sensor de temperatura el cual envía la temperatura del lugar donde se encuentra el nodo sensor; luego, un sensor acelerómetro el cual detecta si el estado de la mota es caído o normal; por último, un sensor GPS que recibe información de latitud y longitud de los distintos satélites.

Adicionalmente, cuenta con un *gateway* el cual recibe los datos que envían las motas y los direcciona al puerto USB, quien posteriormente los envía a un computador; por otra parte, se desarrolla una aplicación de escritorio, sobre la cual se despliega un mapa y sobre él un marcador que muestra la posición, estado y temperatura de cada nodo sensor; asimismo, al sobrepasar el nivel de temperatura establecido emite una alarma que alerta tanto en la mota como en la aplicación de escritorio desarrollada; además, de una alarma si el estado es caído.

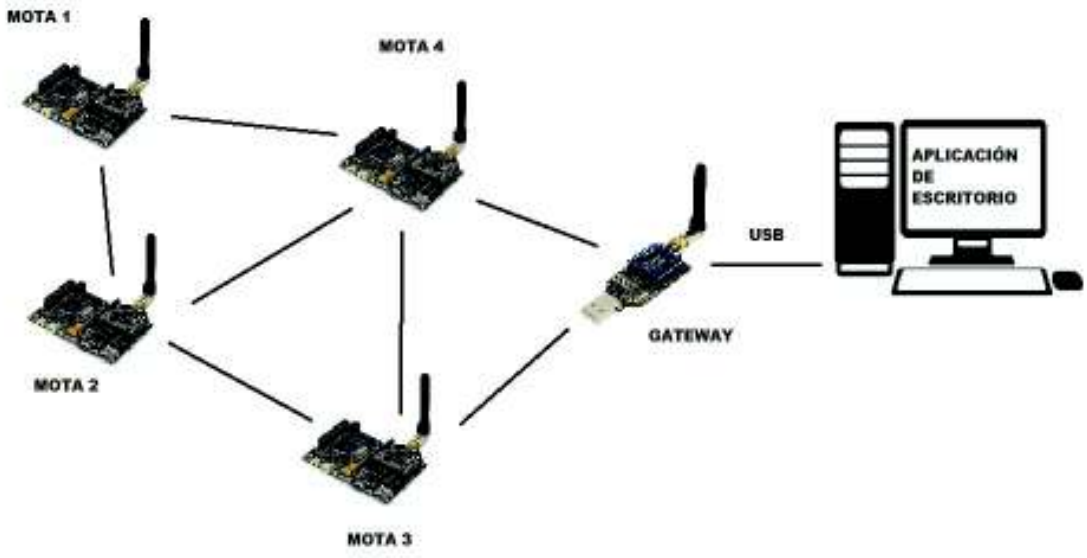


Figura 1.1 Red de sensores inalámbrica

1.1 Objetivos

1.1.1 Objetivo General

- Desarrollar un prototipo de sistema de monitoreo de temperatura y posición con redes de sensores inalámbricas basadas en IEEE 802.15.4.

1.1.2 Objetivos específicos

- Analizar el estándar IEEE 802.15.4 y los distintos lenguajes para programar las motas y los sensores disponibles.
- Diseñar los elementos que conformarán el prototipo que permita monitorear temperatura y posición de la mota considerando los requisitos que debe cumplir el sistema.
- Implementar el prototipo con base al diseño realizado.

1.2 Alcance

El prototipo de sistema de monitoreo de temperatura y posición consta de tres motas que utilizan el estándar IEEE 802.15.4, un y una computadora; cada mota envía la temperatura del lugar donde se encuentra la persona u objeto que porta la mota, así mismo, mediante el sensor acelerómetro se simula la función de estado caído, con la cual se detecta si la mota se encuentra en estado normal o caído; además, el sensor GPS envía la posición de la mota, en este caso latitud y longitud; adicionalmente, al sobrepasar el nivel máximo establecido de temperatura o detectar un estado de caído, emite una alarma tanto en la mota como en la aplicación de escritorio.

El *gateway* recibe los datos y los envía mediante el puerto USB a un computador, sobre el cual se instalará una aplicación de escritorio desarrollada para procesar los datos recibidos con el fin de desplegar en un mapa la posición, estado y temperatura de cada mota, al igual que emitir alarmas en el caso de estado caído o sobrepasar el límite máximo de temperatura establecido por el usuario.

1.3 Marco teórico

El uso de redes de sensores inalámbricas en la actualidad ha resuelto varios inconvenientes sobre el envío de información y la facilidad de transporte de dispositivos; en consecuencia, estas redes se han convertido en una herramienta importante dado que se encuentran en varios lugares como la industria, grandes y pequeñas empresas y el hogar.

1.3.1 Comunicación Inalámbrica

La comunicación inalámbrica, conocida en inglés como *Wireless Communication,* consta de uno o más dispositivos conectados entre sí, sin la necesidad del uso de cables para la transmisión de datos; una red inalámbrica permite a un usuario mantenerse comunicado con otro en un área específica llamada área de cobertura, permitiéndole movilidad dentro de dicha área [1].

Existen varias tecnologías que se basan en la comunicación inalámbrica de corto alcance, entre ellas las más conocidas son Wifi, Bluetooth, Zigbee. Una comunicación inalámbrica de bajo alcance se refiere a un área de cobertura que abarca unos pocos metros, las cuales utilizan ondas electromagnéticas (ondas de radio) que requieren baja potencia para su distribución, alrededor de 1 mW; además, trabajan en una banda de frecuencia específica para la transmisión de datos.

El mayor alcance de la comunicación entre dispositivos se consigue cuando entre los nodos no existen obstáculos, a esto se lo llama línea de vista; los obstáculos que se encuentran entre los dispositivos causan atenuación o perdida de señal, con lo cual requiere mayor potencia para concretar la comunicación. Este tipo de tecnologías se ha convertido en algo muy necesario en la actualidad, dado que no necesitan estar físicamente conectadas para lograr comunicación, permitiendo movilidad al usuario.

1.3.2 Red de sensores inalámbrica (WSN)

Una WSN (*Wireless Sensor Network*) es una red inalámbrica que se compone de varios nodos, los cuales pueden contener sensores capaces de interactuar con el entorno físico donde están ubicados e incluso interactuar con un sistema mediante actuadores[2]; la red recolecta información de los sensores que pueden ser de temperatura, posición, humedad, contaminación, radiación, entre otros.

Cada nodo en la red es independiente y entre ellos colaboran para distribuir la información recibida; así mismo, un módulo de procesamiento será el encargado de administrar los paquetes de datos recibidos y finalmente enviarlos a un nodo receptor conocido como *gateway*, el cual será el encargado de recopilar la información y procesarla para el envío a la aplicación o al host interesado en dicha información.

Al formar una WSN los nodos se dispersan en distintos lugares dentro del área de cobertura que se desea cubrir; cada nodo se alimenta mediante una batería o incluso un panel solar dependiendo de la aplicación que se requiera, permitiendo movilidad y fácil implementación de la red.

1.3.3 Redes inalámbricas

Las redes inalámbricas comunican nodos sin la necesidad de cables a través de ondas electromagnéticas para el envío de información, con esto se eliminan las conexiones físicas en la comunicación pero implica una desventaja la cual brinda menor seguridad en la comunicación dado que un intruso podría capturar los paquetes y descifrarlos[3]. En la Figura 1.2 se observa la clasificación de las redes inalámbricas, las cuales según el área de cobertura se pueden clasificar en WPAN, WLAN, WMAN y WWAN.

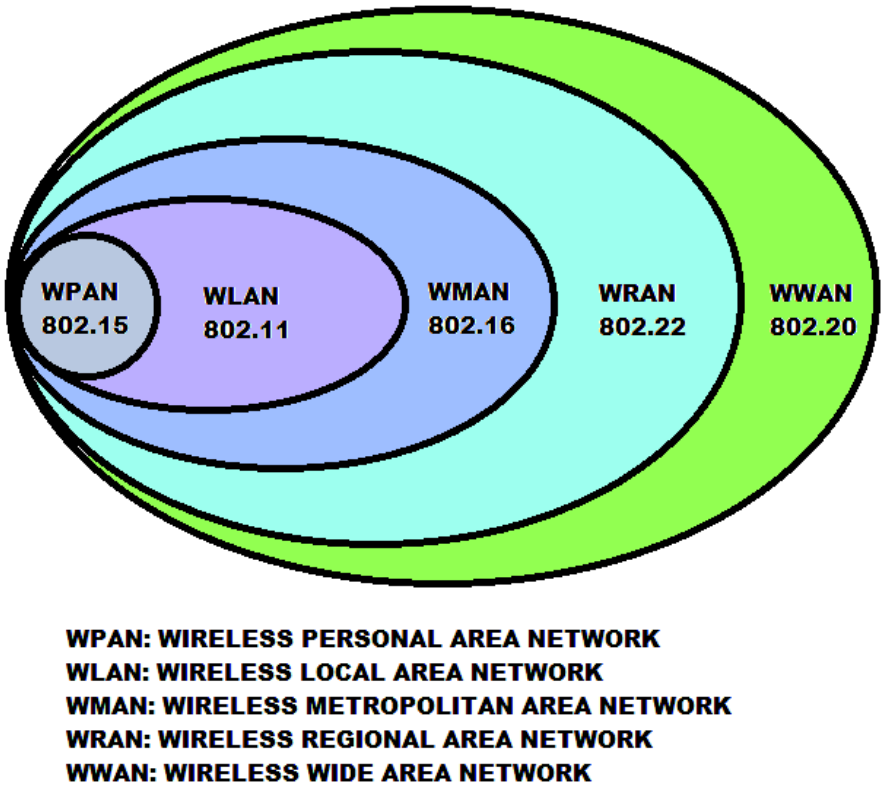


Figura 1.2 Clasificación de las redes inalámbricas

a) Redes inalámbricas de área personal (WPAN)

Las redes inalámbricas de área personal (*Wireless Personal Area Network)* son redes de cobertura muy pequeña, las cuales cubren solo unos cuantos metros; además, pueden conectar varios dispositivos como coordinadores o nodos finales. Sus aplicaciones principales son sensores a nivel del cuerpo humano que interactúan entre sí; comunican dispositivos pequeños a una tasa de velocidad de datos relativamente baja de aproximadamente 250 Kbps; además, permite movilidad y larga duración de batería, su infraestructura e implementación son simples lo cual provee eficiencia en los recursos ya que su protocolo no es tan complejo.

1.3.4 Análisis del estándar IEEE 802.15.4

IEEE 802.15 es un grupo de trabajo dentro de IEEE 802, especializado en redes inalámbricas de área personal, posteriormente se crea el estándar IEEE 802.15.4 definido como *Low-Rate Wireless Personal Area Network* (LR-WPANs) el cual precisa una red de área personal inalámbrica de baja velocidad alrededor de 250Kbps. El estándar inicial fue creado en el año 2003 especificando la capa física y la capa de enlace del modelo OSI[1], posteriormente existieron revisiones del estándar en el año 2009 y finalmente en el año 2011 se realizó la última revisión [4].

a) Generalidades

Una LR-WPAN es una red simple y de bajo costo, la cual permite comunicación inalámbrica con un limitado consumo de energía. La Tabla 1.1 muestra las características principales del protocolo IEEE 802.15.4, entre ellas el número de canales, la banda de frecuencia en la que trabaja, su alcance, su direccionamiento y el método de acceso al canal.

Tabla 1.1 Características del estándar IEEE 802.15.4 [4]

ESPECIFICACIÓN	VALOR
Velocidad de transmisión	2,4 GHz-250 Kbps
	915 MHz-40 Kbps
	868 MHz-20 Kbps
Latencia	Menor a 15 ms
Canales	868 MHz: 1 Canal
	915 MHz: 10 canales
	2,4 GHz: 16 Canales
Alcance	10 m a 90 m
Direccionamiento	Corto de 16 bits y extendido de 64 bits
Canal de acceso	CSMA-CA y CSMA-CA ranurado
Topología	Estrella
	Peer to peer
Consumo de energía	Bajo

[1] OSI es el modelo de interconexión de sistemas abiertos, es un modelo de referencia para los protocolos definiendo una arquitectura en capas.

b) Tipos de dispositivos

Una LR-WPAN se define como una red de bajo costo y consumo de energía la cual permite comunicación inalámbrica entre dispositivos, existen dos tipos de dispositivos que pueden participar de la red que son:

- **Dispositivos de función completa, *Full Function Device* (FFD). -** Es un dispositivo que forma parte de la red y actúa como coordinador, por lo tanto, requiere mayores capacidades computacionales; adicionalmente, es capaz de procesar información y enviarla dependiendo de su aplicación.
- **Dispositivos de función reducida, *Reduced Function Device* (RFD). -** Es un dispositivo que actúa como un módulo final con capacidades reducidas; por lo tanto, se lo utiliza para cumplir acciones simples como recibir información mediante sensores y el envío de datos a través de la red.

c) Topologías de red

Una red LR-WPAN puede trabajar en topología tipo estrella o tipo *point to point* dependiendo de la aplicación a utilizar, la red requiere al menos un coordinador o un dispositivo de función reducida.

c.1) Topología Estrella

Una topología en estrella se compone de uno o más dispositivos de función reducida que se comunican con uno o más dispositivos coordinadores, centralizando la red en el nodo coordinador.

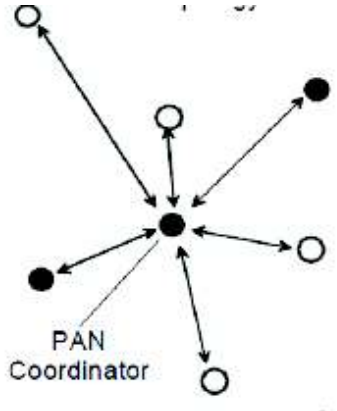


Figura 1.3 Topología tipo estrella [4]

En la Figura 1.3 se muestra la topología estrella de una red de sensores inalámbricos, en la cual, cada nodo que forma parte de la red se conecta al *gateway* y envían información

hacia el coordinador; además, el *gateway* es el encargado de centralizar las comunicaciones y coordinar la red.

c.2) Topología Point to point

Una topología *point to point* se compone de uno o más dispositivos de función reducida, los cuales, actúan como maestro/esclavo, es decir que cualquiera de estos dispositivos puede cumplir con estos roles; además, todos los nodos se comunican entre sí y envían información al nodo más cercano quien posteriormente lo reenvía hasta alcanzar un *gateway*. En la Figura 1.4 se muestra una topología tipo *point to point* la cual permite el envío de información entre nodos.

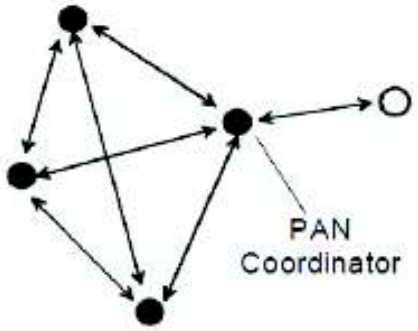


Figura 1.4 Topología point to point [4]

d) Arquitectura del estándar IEEE 802.15.4

Para simplificar el estándar, la arquitectura de IEEE 802.15.4 ha definido varios bloques llamados capas; cada capa es responsable de una parte del estándar y ofrece servicios a las capas superiores. Cada dispositivo que utiliza el estándar IEEE 802.15.4 comprende con al menos una capa física PHY la cual contiene un transceptor de radio frecuencia y la subcapa MAC que provee acceso al canal físico.

e) Capa Física PHY

La capa física provee una interfaz entre la subcapa MAC y el canal de radio, la cual es la responsable de las siguientes tareas:

- Activar o desactivar el transmisor de radio.
- Evaluación del canal mediante CSMA-CA.
- Selección del canal de frecuencia.
- Transmisión y recepción de datos.

La capa física especifica las bandas de frecuencia donde puede trabajar el dispositivo que son 868 MHz, 915 MHz y 2,4 GHz, la cual cuenta con 16 canales y es la más ampliamente usada.

Tabla 1.2 Bandas de frecuencia de la capa física y sus especificaciones [4]

Capa física PHY (MHz)	Banda de frecuencia	Modulación	Tasa de símbolo Kbps	KSÍmbolo/s	Símbolos
868/915	868-868,6	BPSK[2]	20	20	Binario
	902-928	BPSK	40	40	Binario
868/915 Opcional	868-868,6	ASK[3]	250	12,5	20 BIT-PSSS[4]
	902-928	ASK	250	50	5 BIT-PSSS
868/915 Opcional	868-868,6	O-QPSK[5]	100	25	16 *ARY-ORTHOGONAL*[6]
	902-928	O-QPSK	250	62,5	16 *ARY-ORTHOGONAL*
2450	2400-2483,5	O-QPSK	250	62,5	16 *ARY-ORTHOGONAL*

La Tabla 1.2 muestra las diferentes especificaciones de la capa física, el tipo de modulación, la tasa de bits, la tasa de símbolo; para garantizar que no exista interferencia entre las señales inalámbricas se utiliza un método llamado DSSS (*Direct Sequence Spread Spectrum*), el cual es una técnica de modulación de canal que se usa para transmitir señales digitales sobre el espectro radioeléctrico; para esto, utiliza un código de Pseudo ruido que se origina a partir de una secuencia pseudo aleatoria en base a pulsos binarios.

DSSS modula digitalmente una señal portadora, por lo tanto, aumenta el ancho de banda de transmisión y reduce el nivel de potencia de la señal en una frecuencia dada; en conclusión, genera una señal muy parecida al ruido que solo tiene significado en el receptor

[2] BPSK (*Binary phase shift keying*), Es un proceso de modulación digital, el cual modula la fase de una señal de referencia, permite modular un bit/símbolo a una alta velocidad de transmisión [27].

[3] ASK (*Amplitude shift keying*), La Modulación por desplazamiento de amplitud, representa datos digitales como variaciones de amplitud de la onda portadora en función de la información [28].

[4] PSSS (*Parallel secuence spread spectrum*), pertenece a la capa física de comunicación de radiofrecuencia, la cual integra codificación, modulación y ecualización [29].
[5] O-QPSK (*Orthogonal-Quadrature phase-shift keying*), conocida como modulación por desplazamiento de fase cuaternario ortogonal, permite codificar dos bits por cada símbolo [27].
[6] *ARY-Orthogona*l, La señal codificada fue creada para evitar la interferencia entre tramas e incrementar la tasa de información [30].

[5]; la capa física provee servicios de datos y administración de servicios que se explican a continuación:

- Los servicios de datos permiten la transmisión y recepción de unidades de datos de protocolo (PDUs) sobre el medio físico; además, define la banda de frecuencia sobre la cual trabajará el dispositivo, que pueden ser, 868 MHz, 915 MHz y 2,4 GHz [4].
- La administración de servicios es la responsable de mantener una base de datos de administración de objetos que pertenecen a la capa física.

f) Sincronización

Para que los receptores conozcan que se inicia una transmisión requiere que exista sincronización entre transmisor y receptor; la Figura 1.5 muestra la cabecera de sincronización IEEE 802.15.4, la cual consta de un preámbulo de 32 ceros, seguido de un delimitador de comienzo de trama SFD (*Start of Frame Delimiter*) que se compone de la secuencia 10100111, todo esto forma la cabecera de sincronización SHR (*Synchronization Header)* de 5 bytes; posteriormente, la cabecera de la capa física que consta de 7 bits e indican la longitud de la trama y R de tamaño 1 bit el cual es reservado para futuros usos; finalmente, se incluye la carga útil de longitud variable entre 0-127 bytes donde se incluye la información a ser transmitida.

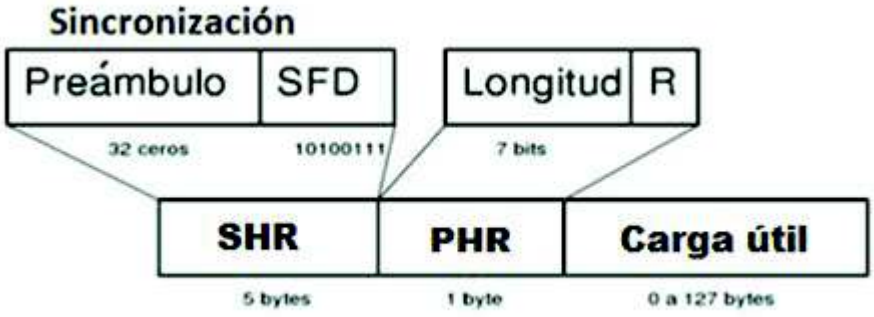


Figura 1.5 Sincronización de la trama IEEE 802.15.4 [9]

g) Trama IEEE 802.15.4

La trama IEEE 802.15.4 ha sido desarrollada para reducir su complejidad, pero obteniendo varias prestaciones y a su vez realizar una transmisión confiable y lo suficientemente robusta para operar en un canal que introduce ruido, el estándar IEEE 802.15.4 define cuatro estructuras de tramas MAC:

- La trama *beacon* usada por un coordinador para asociar dispositivos; además, contiene toda la información sobre la red inalámbrica y son retransmitidos periódicamente.
- Una trama de datos usada para todo tipo de transmisión de datos.
- Una trama ACK usada para garantizar la entrega de tramas, son tramas de confirmación que se envían en la comunicación al recibir una trama de manera correcta.
- Una trama de comandos MAC usada para el control de transferencia de unidades MAC.

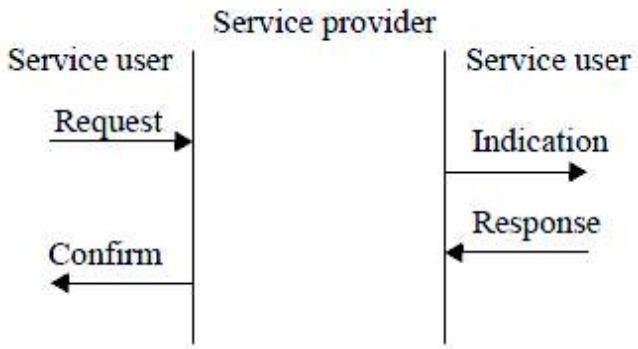


Figura 1.6 Servicios

Mediante las distintas tramas se presta servicios a las capas inferiores, estos servicios son ilustrados en la Figura 1.6 que muestra la jerarquía de servicios y la relación entre dos usuarios que intercambian información; estos servicios describen el flujo de información entre N usuarios utilizando las primitivas que pueden ser del tipo:

- **Requerimiento:** Se la utiliza para requerir que se inicie un servicio.
- **Indicación:** Permite indicar al usuario que ha ocurrido un evento interno.
- **Respuesta:** Se usa para completar un procedimiento invocado previamente por la primitiva de indicación.
- **Confirmación:** Se utiliza para comunicar el resultado de una o más asociaciones.

h) Subcapa MAC

Esta subcapa MAC provee un interfaz entre el SSCS (Subcapa de convergencia de servicios específicos) y la capa física; además, provee dos servicios, los servicios de datos MAC y la interfaz de administración de servicios MAC. Los servicios de datos de la subcapa

MAC realizan la transmisión y recepción de unidades de datos de protocolo (MPDUs) sobre el servicio de datos de la capa física.

Las características de la subcapa MAC son la administración de tramas, la gestión de acceso al canal, validación de tramas, tramas de confirmación de entrega, asociación y disociación; adicionalmente puede proveer seguridad en la entrega de los datos. La subcapa MAC es la responsable de las siguientes tareas:

- Generar *Beacons* de red si el dispositivo es un coordinador, quien puede determinar si trabajara en modo *beacon* habilitado o no.
- Permite asociación o disociación en una red de área personal.
- Emplear el mecanismo de acceso al canal CSMA-CA.
- Proporciona un enlace confiable entre dos entidades pares MAC.

La trama de la subcapa MAC a la cual se la denomina unidad de datos de protocolo MAC MPDU (*MAC Protocol Data Unit)*, cuenta con los elementos descritos en la Figura 1.7, la cual se compone del encabezado MAC MHR (*MAC Header*), la unidad de servicios de datos MAC MSDU (*Mac Service data Unit*), la carga útil (*Payload*) y finalmente el fin de la MAC MFR *(MAC Footer*).

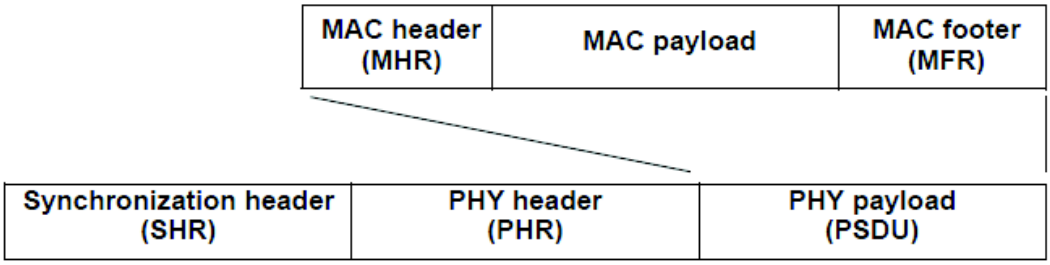


Figura 1.7 Trama MAC IEEE 802.15.4

Además, en la Figura 1.7, se indica el *MAC Header* (MHR), la cabecera de la trama MAC se compone de un control de flujo de trama de 2 bytes, el número de secuencia de 1 byte, la dirección de destino que varía desde 0 a 10 bytes, la dirección de origen que varía desde 0 a 10 bytes y una cabecera auxiliar de información de hasta 14 bytes.

Posteriormente se encuentra el MAC *Payload*, en donde se ubica la carga útil la cual dispone de longitud variable y no debe exceder 127 bytes de información; finalmente, se encuentra el MFR (*MAC Footer)*, sobre el cual se escribe una secuencia de números para comprobar la validación de la trama, además, enlazan las tramas de confirmación con la secuencia anteriormente transmitida. Para garantizar el envío de la trama, IEEE 802.15.4

utiliza varios mecanismos entre ellos la forma de acceso al canal mediante CSMA-CA que se explica a continuación.

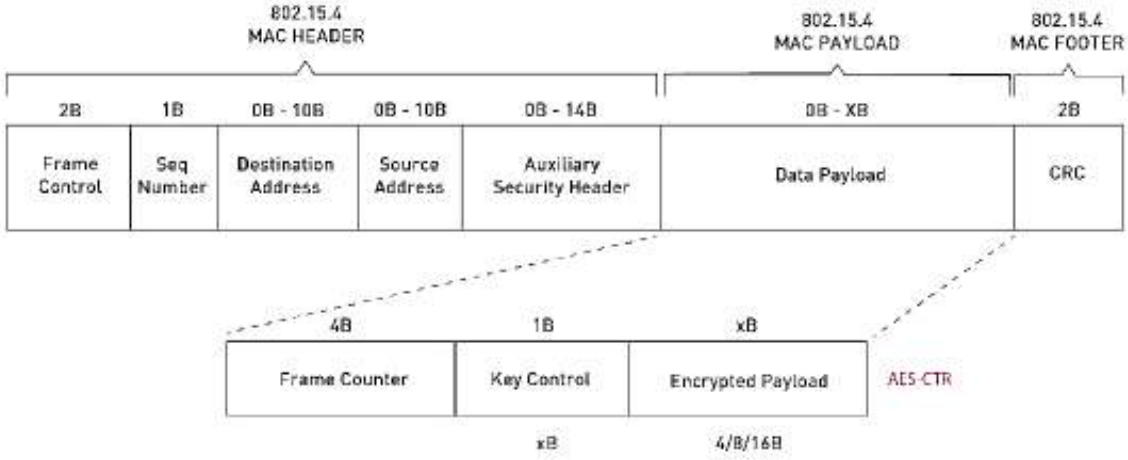


Figura 1.8 Características de la trama MAC [4]

El manejo de la carga útil permite el uso de encripción de tipo AES (*Advanced Encryption Standard*), de la cual se puede usar AES-CCM (*Counter Mode Encryption and Cipher Block Chaining Message Authentication Code*), AES-CTR (*Counter Mode*) y AES-CBC-MAC (*Cipher Block Chaining Message Authentication Code*); la Figura 1.8 muestra la trama con encripción del tipo AES-CTR. El análisis de esta sección no se la realiza dado que la carga útil usada en el prototipo no requiere ningún tipo de encripción.

i) CSMA-CA

El método de acceso al canal CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*) conocido como acceso múltiple por detección de portadora y prevención de colisiones, es un protocolo de acceso al medio que permite a múltiples hosts acceder a la red mediante un medio de comunicación común.

Cada host antes de transmitir información sobre la red anuncia el envío de datos, con lo cual, se puede evitar colisiones dado que cada estación sabe que un host se encuentra transmitiendo o no información; en consecuencia, una vez que se libera el canal se espera un tiempo corto y aleatorio para el uso de este, si después de este tiempo de guarda aleatorio el canal sigue libre se procede a utilizar el canal; por consiguiente, esta técnica reduce la probabilidad de colisión.

1.3.5 Internet de las Cosas

El Internet comúnmente conocido (*Interconnected Networks)* se entiende como redes interconectadas que se forman a partir de la unión de todas las redes y hosts distribuidos

por todo el mundo; asimismo, es una red que constantemente se encuentra evolucionando y en los últimos años ha crecido a pasos grandes.

El Internet de las cosas IoT (*Internet of Things*) se ha considerado como el futuro del Internet, con el cual podremos realizar una interacción maquina a máquina dado que provee conectividad para todos en cualquier momento e incluso en cualquier lugar [6]**.** El internet de las cosas permite conectar varios objetos para realizar una actividad, he incluso tomar decisiones, proveer servicios e invocar acciones; además, provee una interacción del mundo real hacia un enlace virtual que puede ser una aplicación.

Actualmente ha crecido el número de dispositivos que realizan estas interacciones, entre ellos se encuentran computadores, *tablets*, teléfonos inteligentes y sensores. En el año 2005 la Unión Internacional de Telecomunicaciones (ITU) propuso que el "Internet de las Cosas" conectará los objetos con el mundo real de una manera inteligente y a base de sensores [6].

a) Aplicaciones de IoT

Dadas las facilidades que brinda IoT, se pueden implementar un sin número de aplicaciones que pueden ayudar a las personas a minimizar el tiempo en los trabajos u optimizar procesos; esta tecnología busca la comunicación de todo con todo, lo cual es un concepto en el que todos los equipos puedan disponer de una dirección IP y conectarse a la red para recibir y enviar información dependiendo de la aplicación que realice; por consiguiente, las aplicaciones de IoT pueden ser:

- Parqueaderos inteligentes.
- Monitoreo de pacientes.
- Mapas urbanos de ruido o contaminación.
- Detección inteligente de gases tóxicos y no tóxicos.
- Automatización de hogares.
- Detección de incendios forestales.
- Calidad de agua, seguridad, salud, industria, etc.

1.3.6 Libelium

Libelium es una empresa española que se encarga de fabricar sensores y módulos que se adaptan al estándar IEEE 802.15.4, permitiendo comunicación inalámbrica de datos;

además, dispone de varios productos entre software y hardware que satisfacen las diversas aplicaciones para Internet de las Cosas.

a) Nodos sensores

Un dispositivo modular y programable capaz de ser adaptado a varias aplicaciones, se lo denomina “Mota”, el cual permite un ambiente de desarrollo ya que cuenta con varias estructuras cambiantes y adaptables [7]. Las motas internamente disponen de un microcontrolador Atmel ATmega 1281, el cual carga la primera instrucción en 62,5 ms después de ser encendido; asimismo, las motas libelium se las denomina como Waspmote, lo cual especifica el modelo de mota utilizada.

b) Especificaciones técnicas

Según la guía técnica de Libelium Waspmote se han recopilado las siguientes características técnicas [8]:

Tabla 1.3 Características técnicas Waspmote PRO

Waspmote PRO	Característica
Microcontrolador	ATmega1281
Frecuencia	14.7456 MHz
EEPROM	4 KB
SRAM	8 KB
FLASH	128 KB
Tarjeta SD	2 GB
Peso	20 gramos
Dimensiones	73.5 x 51 x 13 mm
Rango de operación de temperatura	-10 °C a +65 °C

La Tabla 1.3 muestra las características técnicas de la placa Waspmote, la cual posee un microcontrolador ATmega 1281, el cual internamente combina una memoria flash de 128 KB, una SRAM de 8 KB, EEPROM de 4 KB.

La Figura 1.9 presenta los componentes de la vista superior de la placa Waspmote, la cual se compone de puertos de entrada y salida donde se ubican los sensores, los puertos de radio donde se ubica el módulo XBee, *socket* de batería, USB y panel solar; asimismo, el microcontrolador, un acelerómetro integrado, leds y un interruptor de encendido y apagado.

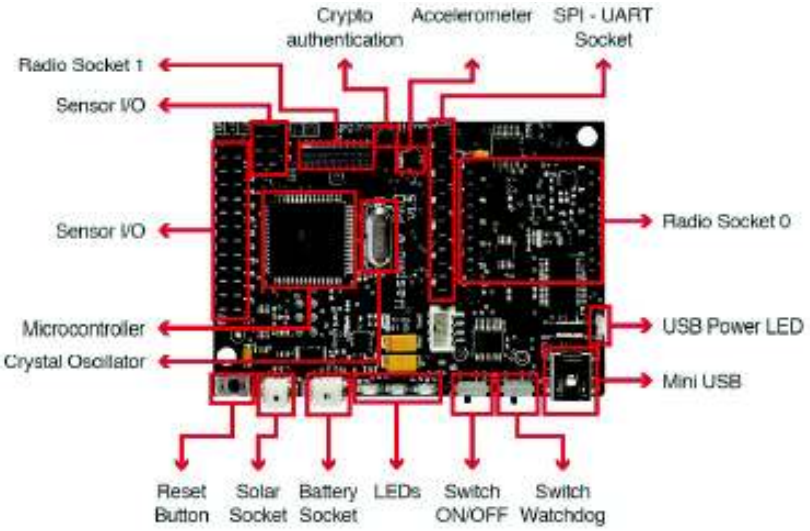


Figura 1.9 Vista superior de la placa Waspmote [8]

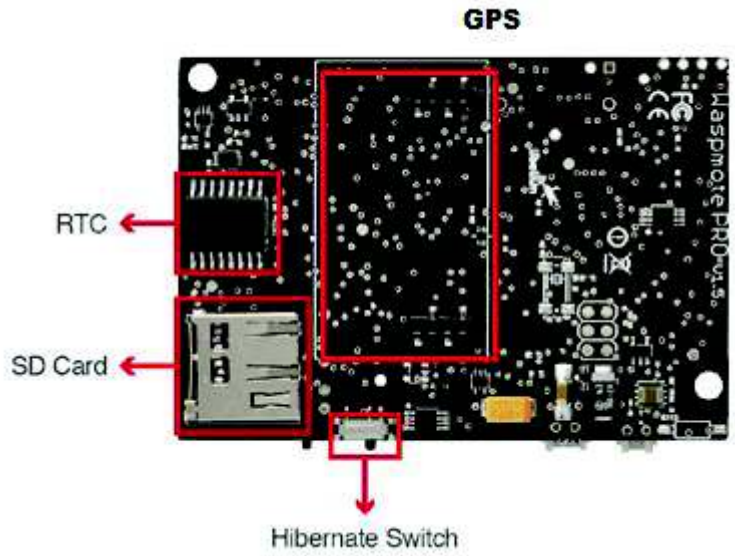


Figura 1.10 Vista inferior de la placa Waspmote [8]

En la Figura 1.10 se presentan los componentes de la vista inferior de la placa Waspmote, la cual se compone del *socket* de la tarjeta SD, los puertos para la conexión del sensor GPS y un interruptor de encendido y apagado; además, cuenta con el RTC (*real time clock*) un reloj en tiempo real el cual se lo utiliza para la sincronización, lo que permite programarlo para realizar acciones en función del tiempo.

b.1) Puertos de entrada y salida

Waspmote dispone de varios puertos de entrada/salida que se utilizan para conectar los distintos sensores o placas que requiera el usuario; la Figura 1.11 muestra los puertos de entrada y salida con su respectiva etiqueta, existen dos tipos de puertos que pueden ser

analógicos o digitales, los cuales se los activa o desactiva programando la mota; además, permite alimentar los sensores con 3V o 5V, con su respectiva conexión a tierra.

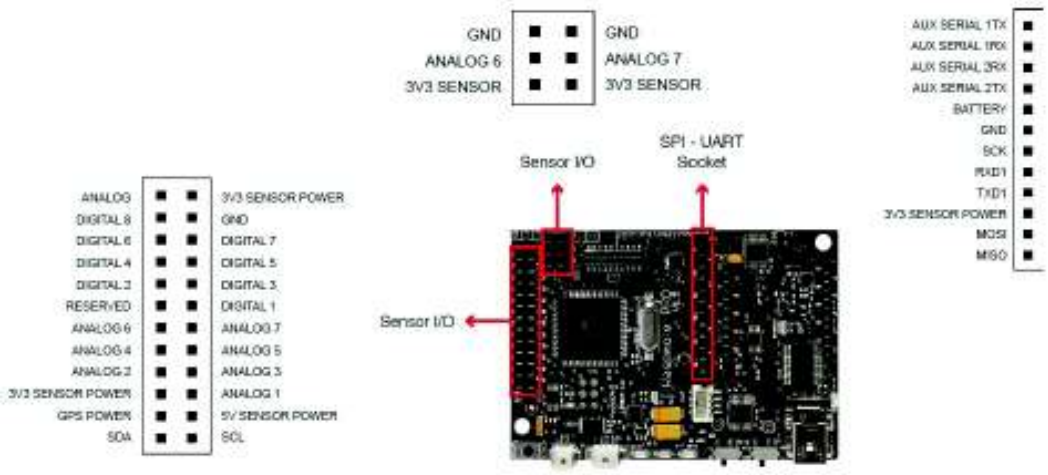


Figura 1.11 Puertos de entrada y salida de la placa Waspmote [8]

b.2) LEDs

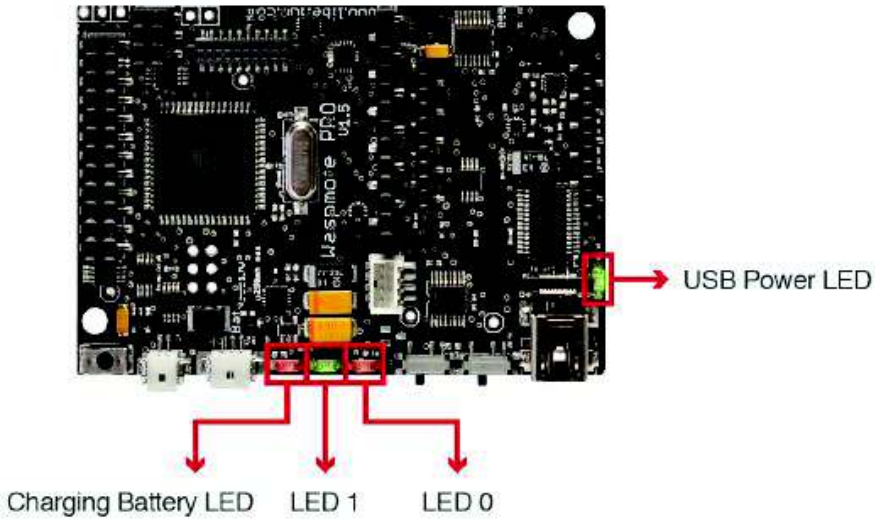


Figura 1.12 Leds indicadores en la placa Waspmote [9]

La Figura 1.12 muestra la placa Waspmote, específicamente se señalan los LEDs que dispone, los cuales son:

- **LED indicador de carga de batería (*Charging battery LED indicator*):** Es un LED de color rojo, el cual indica que la batería está conectada y se encuentra cargando; la carga se realiza mediante el puerto USB o mediante un panel solar; asimismo, una vez que la batería se encuentre cargada completamente el led se apagará automáticamente.

- **LED0 y LED1 (*Programmable LED*):** Es un LED que se encuentra conectado directamente al microcontrolador, el cual es totalmente programable por el usuario mediante código; adicionalmente, el LED0 puede indicar cuando la mota se resetea.
- **LED de carga USB (*USB Power LED indicator*):** Es un LED de color verde el cual indica que la mota está conectada a un puerto USB, lo cual permite cargarlo o programarlo.

c) Especificaciones de energía

La Tabla 1.4 muestra los valores operacionales de energía con los que trabaja la mota, los cuales son voltaje mínimo y máximo de operación de la batería, voltaje de carga USB, voltaje de carga del panel solar; además, del voltaje y corriente en cualquier pin de entrada o salida.

Tabla 1.4 Valores operacionales de Waspmote

Valores operacionales	
Voltaje mínimo de operación de batería	3.3 V
Voltaje máximo de operación de batería	4.2 V
Voltaje de carga USB	5 V
Voltaje de carga panel solar	6-12 V
Voltaje en cualquier pin	-0.5 V, +3.8 V
Máxima corriente en cualquier entrada o salida digital	40 mA

d) Consumo de energía

Para optimizar el ahorro de energía, Waspmote cuenta con cuatro distintos modos:

- **Encendido (*On*):** Modo de operación normal, consume 17 mA.

- **Dormido (*Sleep*):** El programa principal se pausa, por lo tanto, el microcontrolador pasa a un estado de reposo que finaliza al recibir una llamada asincrónica; la duración de hibernación varía de 32 ms a 8 segundos con un consumo de 30 uA.
- **Profundamente dormido (*Deep Sleep*):** El programa principal pausa el microcontrolador y lo cambia a un estado de reposo, pero mediante una llamada asincrónica o sincrónica lo despierta; la duración de este estado puede ser de minutos, horas e incluso días con un consumo de 33 uA.
- **Hibernación:** El programa principal está detenido, el microcontrolador y todos los módulos están completamente desconectados, pero la única forma de reactivar el dispositivo es mediante una alarma previamente configurada en el RTC; su consumo es de 7 uA.

Tabla 1.5 Modos de operación y su consumo de energía [9]

	Consumption	Microcontroller	Cycle	Accepted interruptions
On	17 mA	On	-	All interruption sources
Sleep	30 µA	On	Depends on INT source	All interruption sources
Deep Sleep	33 µA	On	1 s – 31 days	All interruption sources (RTC always used)
Hibernate	7 µA	Off	1 s – 31 days	Only RTC

La Tabla 1.5 muestra el consumo de energía de la mota en los distintos modos de operación; además, el consumo de energía en el modo sobre el cual trabaja que puede ser encendido, dormido, profundamente dormido o hibernación.

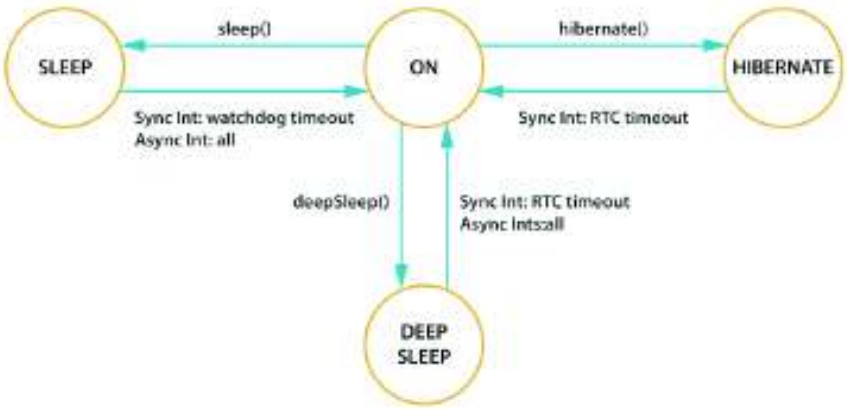


Figura 1.13 Diagrama de modos de operación de Waspmote [9]

La Figura 1.13 describe los distintos modos de operación que utiliza Waspmote, cada modo dispone de diferentes interrupciones que puede recibir el microcontrolador, entre ellos encendido, dormido, profundamente dormido o hibernación; el control de interrupción

permite activar al microcontrolador en el modo deseado dependiendo de las aplicaciones y los sensores que se usen.

e) Interrupciones

Waspmote se diseñó para trabajar con dos tipos de interrupciones que son sincrónicas y asincrónicas, cualquier interrupción puede despertar a Waspmote si se encuentra en un estado dormido o profundamente dormido; pero, solo una interrupción sincrónica puede despertar a Waspmote si se encuentra en estado de hibernación.

Interrupción sincrónica: Están programadas mediante temporizadores, es decir se puede indicar cuando se quiere que se activen; existen dos tipos de interrupciones sincrónicas.

- **Alarma periódica**: Es aquella en la que se especifica un momento de tiempo particular en el futuro controlado mediante el RTC; por ejemplo, una alarma programada cada tercer día de cada mes a las 18h00 y durante 10 segundos.
- **Alarma relativa:** Es aquella que considera el momento actual mediante el RTC y se la programa en un lapso a partir de dicho momento, registrado mediante el microcontrolador; por ejemplo, una alarma programada durante 30 minutos.

Interrupción asincrónica: Es aquella que no se encuentra programada, por lo tanto, no se conoce cuando va a ejecutarse ya que depende de la acción que realicen los sensores; por ejemplo, en el caso de sobrepasar un límite establecido en el sensor de temperatura, este dispara una alarma.

f) Transmisión de señal de datos

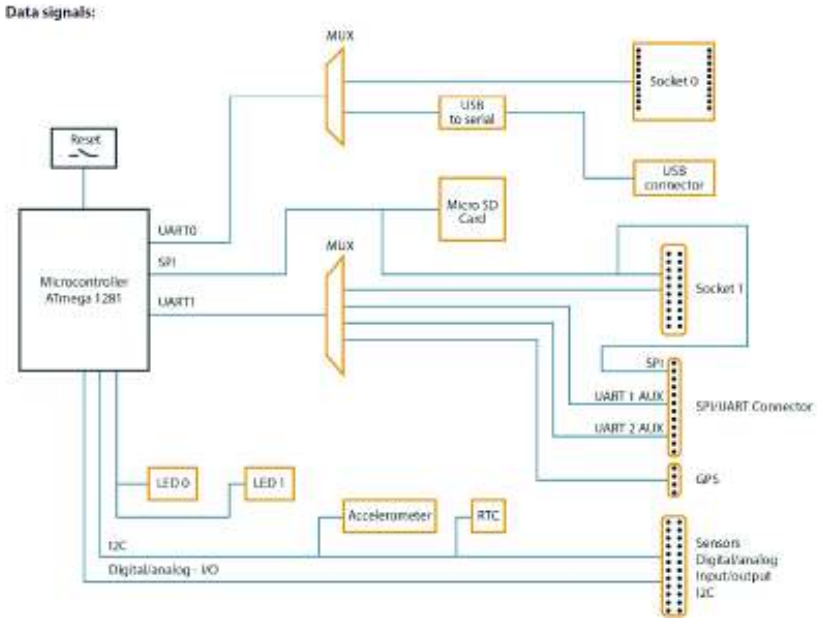


Figura 1.14 Diagrama de bloques de la señal de datos [9]

La Figura 1.14 muestra cómo se envía la señal de datos a través de los puertos y el microcontrolador mediante el UART[7]; cada puerto de entrada y salida envía y recibe información del microcontrolador, el cual procesa la información y decide las acciones a tomar.

1.3.7 Waspmote PRO-IDE

Waspmote PRO-IDE es el entorno de desarrollo integrado para Waspmote que permite escribir el código, depurarlo y posteriormente cargarlo a la mota; además, permite administrar la salida de datos en serie. El IDE contiene la API (*Application Programming Interface*), la cual es un conjunto de todas las bibliotecas que Waspmote necesita para compilar el programa; el software de Waspmote PRO-IDE puede ser instalado sobre múltiples sistemas operativos entre ellos Windows, Mac OS y Linux [10].

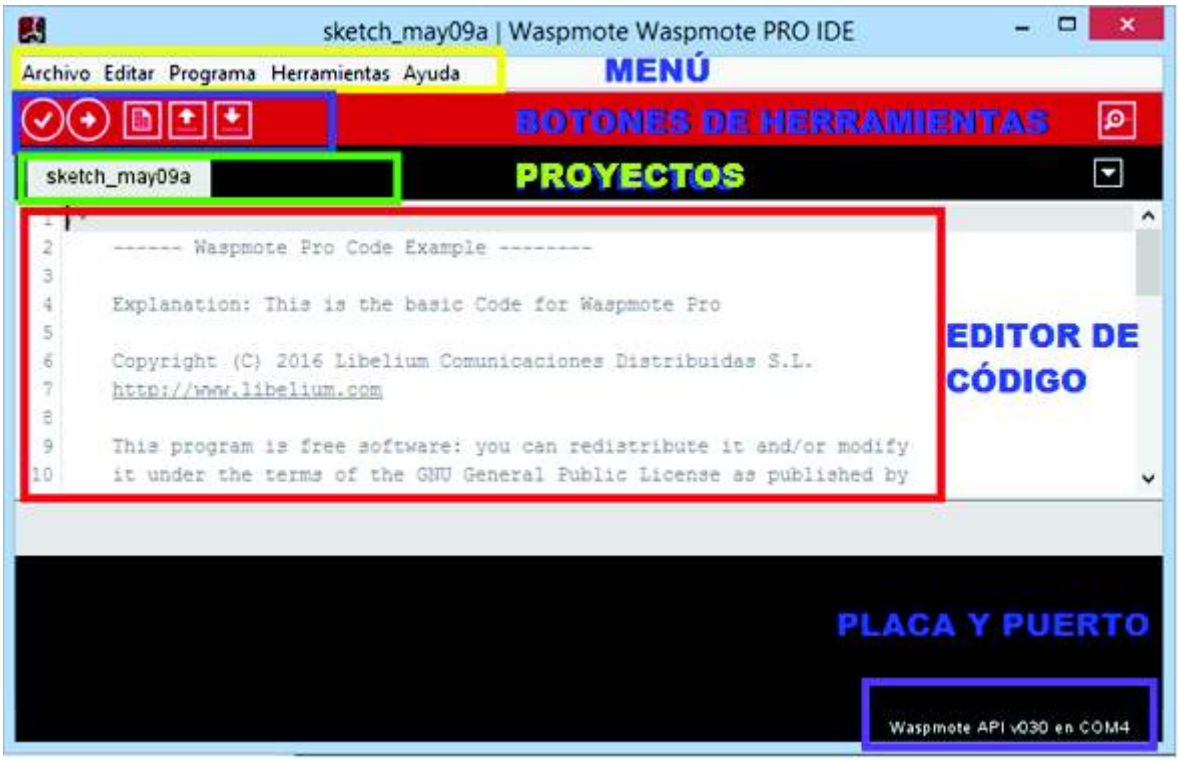


Figura 1.15 Componentes de Waspmote PRO-IDE

La Figura 1.15 indica los distintos componentes de Waspmote PRO-IDE; además, en la esquina inferior derecha se muestra la placa que se encuentra en uso y el puerto por el que se comunica al computador. Los códigos escritos usando el IDE se crean con la

[7] UART (*Universal Asynchronous Receiver Transmitter)*, es un módulo que controla los puertos y dispositivos serie; además, maneja las interrupciones de los dispositivos conectados al puerto serie y los convierte en formato paralelo [31].

extensión “. pde”, los cuales admiten archivos en C con la extensión “.c” y archivos de c++ con la extensión “.cpp” o archivos de cabecera “.h”. Los componentes de Waspmote PRO-IDE son los siguientes:

a) Menú

Permite desplegar las distintas herramientas del programa que son archivo, editar, programa, herramientas y ayuda.

b) Botones de herramientas

Figura 1.16 Botones de herramientas

La Figura 1.16 describe las funciones de los diferentes botones de herramientas que realizan acciones en el código como compilar, subir, guardar, abrir y crear un nuevo programa.

- **Compilar:** El botón de compilar permite traducir el código de programación a un código ejecutable por la maquina; además, en este proceso indica si existen errores en el código.
- **Subir:** Compila el código y lo sube a la placa Waspmote, siempre y cuando no existan errores en el código.
- **Nuevo:** Crea un nuevo proyecto.
- **Abrir:** Abre un proyecto ya creado.
- **Guardar:** Guarda los cambios realizados en el proyecto actual.

c) Editor de código

Permite agregar, editar o eliminar código; asimismo, la estructura del código se divide en dos componentes básicos que son *setup* y *Loop;* la Figura 1.17 muestra la estructura del código, dividida en dos secciones que son:

- ***Setup*:** Es la primera parte del código, la cual se ejecuta una sola vez al inicializarlo; por consiguiente, en esta sección se inicializan los sensores que se van a utilizar, al igual que la parte del código que es importante cuando la mota inicialice.

- ***Loop*:** Esta parte del código se ejecuta continuamente formando un lazo infinito, es recomendable el uso de interrupciones para mejorar el código lo cual permitirá ahorro de energía en la mota.

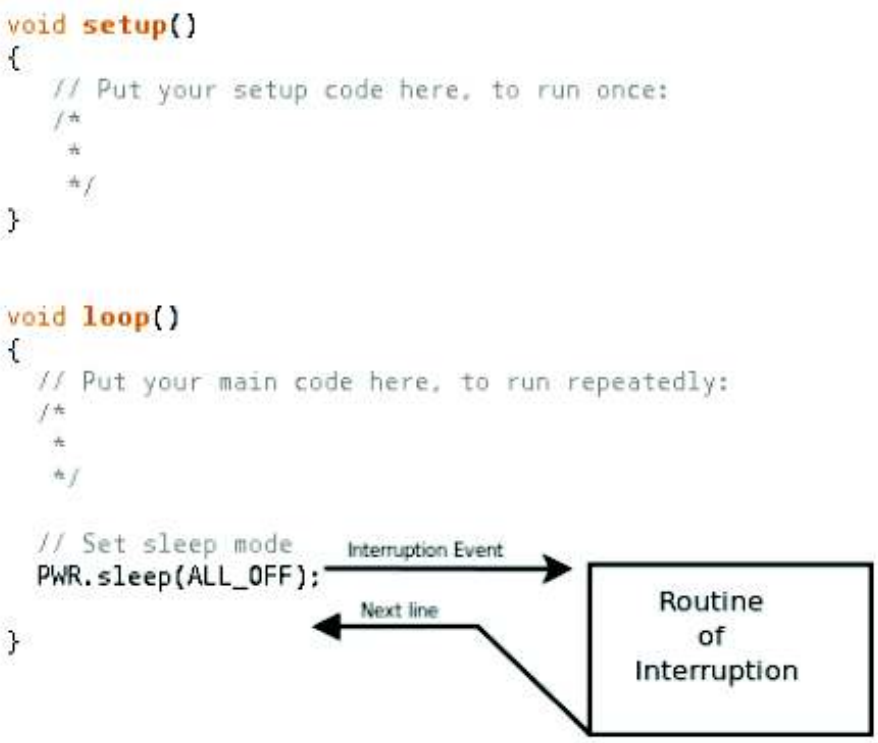


Figura 1.17 Estructura del código [9]

1.3.8 Comparación de los distintos lenguajes de programación para la aplicación de escritorio

Para la elaboración del prototipo se analizan varios lenguajes de programación y así se puede determinar cuál se adapta mejor a las necesidades del prototipo; por consiguiente, se busca un lenguaje capaz de desplegar un mapa y sobre él, los datos relacionados a su posición (latitud, longitud), su estado (caído, normal) y su temperatura (grados centígrados); para esto se analizan los lenguajes C, C++, C# y Java.

a) C

C es un lenguaje basado en el paradigma de programación estructurada, con el fin de permitir al programador dividir al programa en módulos; además, es un lenguaje de propósito general, el cual brinda al desarrollador un control total sobre sus programas, permitiendo la programación a bajo nivel como acceso a memoria y varios recursos, además, de programación de alto nivel. Es un lenguaje creado en la década de los 70 pero actualmente aún sigue en uso [11].

b) C++

Fue creado para extender al lenguaje C con el objetivo de manipular objetos, permitiendo a C++ ser un lenguaje multiparadigma que emplea una programación estructurada y además orientada a objetos; por lo tanto, define clases y brinda la posibilidad de capturar excepciones [12].

c) C#

Es una versión creada a partir de su antecesor C++, la cual dispone de una programación de alto nivel; además, utiliza el modelo de objetos de .NET similar a Java pero incluyendo mejoras derivadas de otro lenguaje [13].

C# es un lenguaje de programación independiente, el cual fue diseñado para generar programas sobre la plataforma .NET.

d) Java

Fue desarrollado puramente orientado a objetos, es decir que cualquier función debe pertenecer a una clase; por otra parte, se ejecuta sobre un procesador virtual JVM (*Java virtual machine*), el cual abstrae al programador de la plataforma concreta; de esta manera el compilador genera *bytecode* y lo envía al procesador de JVM [14].

En la Tabla 1.6 se muestra la comparación de los distintos lenguajes de programación analizados según el control de la plataforma, la liberación de memoria, el uso de punteros, el nivel de programación y la experiencia del autor en el uso del programa.

Se opta por usar el lenguaje de programación C# dado que C y C++ son lenguajes para programación de bajo nivel es decir sistemas que no corren en un computador sino más bien en otro tipo de dispositivos; además, C# es un lenguaje de alto nivel que mejor se adapta a las necesidades del prototipo dado su facilidad para el uso de complementos de mapa y comunicación serial.

C# es muy similar en varias características a JAVA, pero C# posee más funcionalidades presentes en C++ que se las ha adaptado; además, de la experiencia que se posee en el uso del programa.

Tabla 1.6 Comparación de lenguajes de programación

LENGUAJE	C	C++	C#	JAVA
Orientación a objetos	Programación estructurada, no orientada a objetos.	Programación orientada a objetos y no orientada	Orientado a objetos	Puramente orientado a objetos
Control de la plataforma	Programador tiene control del procesador	Programador tiene control del procesador	Compilación a un lenguaje intermedio MSIL (*microsoft intermediate language*)	Programa se ejecuta sobre un procesador virtual JVM
Liberación de memoria	Depende del programador	Depende del programador	Depende del programador	Los objetos creados son destruidos automáticamente
Punteros	Permite el uso de punteros	Permite el uso de punteros	Permite el uso de punteros	No existen punteros, los objetos son accedidos por referencia
Nivel de programación	Programación de bajo nivel	Programación de bajo nivel	Programación de alto nivel	Programación de alto nivel
Experiencia del desarrollador	Intermedia	Intermedia	Alta	Baja

e) Microsoft Visual Studio C#

C# es un lenguaje de programación de propósito general orientado a objetos y creado por Microsoft para su plataforma .NET, con el propósito de ser sencillo para el desarrollador de software [13].

C# permite crear aplicaciones cliente de Windows, servicios web XML, componentes distribuidos y aplicaciones cliente-servidor; asimismo, proporciona un editor de código avanzado, diseñadores de interfaz de usuario, un depurador integrado y varias herramientas adicionales, además, utiliza una sintaxis expresiva y sencilla para el desarrollador y proporciona características eficientes como tipos de valor, enumeraciones, delegados, entre otras; asimismo, admite métodos y tipos genéricos que proporcionan mayor seguridad y rendimiento [13].

2 METODOLOGÍA

Se realiza un prototipo de monitoreo de posición y temperatura basado en el estándar IEEE 802.15.4, el cual se compone de hardware y software; además, permite monitorear a la persona que porte la mota, tanto su posición mediante un GPS (latitud, longitud), como su estado (caído, normal) mediante un acelerómetro el cual detecta las posiciones x, y, z en el espacio, además de saber la temperatura a la cual está expuesta mediante un sensor de temperatura acoplado a la placa de la mota.

Para realizar el diseño del prototipo; primero, se plantea definir los requerimientos necesarios para su desarrollo; posteriormente, se realizará una descripción general del prototipo y la forma de dividirlo para su fácil entendimiento (red de sensores inalámbrica y aplicación de escritorio); luego, se definen los módulos necesarios que conforman cada mota y la forma de comunicación de la red inalámbrica; finalmente, se define el diseño de la aplicación de escritorio y sus diferentes métodos y clases necesarios para el funcionamiento del prototipo.

2.1 Requerimientos del prototipo

Para un mejor entendimiento del prototipo, se lo ha dividido en una red de sensores inalámbrica y la aplicación de escritorio.

2.1.1 Requerimientos de la red de sensores inalámbrica

La red de sensores inalámbrica compuesta por el *gateway* y las motas, pretende cumplir con los requisitos descritos a continuación:

- La red de sensores inalámbrica debe permitir comunicación entre el *gateway* y las distintas motas.
- La comunicación de las motas y el *gateway* se debe llevar a cabo mediante radiofrecuencia, cumpliendo con el estándar IEEE 802.15.4.
- Cada mota debe obtener la temperatura a la cual está expuesta; además, su posición mediante latitud y longitud y su estado mediante el valor del ángulo del eje z recibido del sensor acelerómetro.
- Cada mota debe emitir un sonido de alarma en el caso de sobrepasar el límite de temperatura establecido o a su vez, si se encuentra en estado caído.

- Cada mota debe crear una trama IEEE 802.15.4 y dentro de esta trama se debe adjuntar la información obtenida de los sensores y su identificador, el cual es único y lo diferencia de una u otra mota.
- Cada nodo sensor debe enviar una trama al *gateway* cada intervalo de tiempo definido por el programador.
- El *gateway* debe recibir la información de cada nodo sensor y procesarlo, para posteriormente enviar dicha información de manera digital al puerto USB.

2.1.2 Requerimientos de la aplicación de escritorio

La aplicación de escritorio pretende cumplir los requerimientos descritos a continuación:

- La aplicación debe recibir los datos del *gateway* enviados mediante el puerto USB y recibidos en la aplicación mediante un puerto Com, este puerto Com se lo debe seleccionar en la aplicación.
- La aplicación debe mostrar en un mapa a cada mota, diferenciándola con un marcador de color distinto.
- Sobre cada mota desplegada en el mapa debe ser posible determinar sus datos como temperatura, posición, estado, identificador y nombre.
- El mapa debe actualizar y mostrar los datos cada vez que reciba información de las motas.
- Una vez que la mota detecte que ha sobrepasado el umbral de temperatura definido, la aplicación debe emitir una alarma y alertar a la persona que monitorea la aplicación que dicha mota se encuentra expuesta a una alta temperatura.
- Una vez que la mota detecte que su estado es caído, debe emitir una alarma y alertar a la persona que monitorea la aplicación que dicha mota se encuentra en estado caído.
- En el caso de producirse una alarma por cualquier evento, ya sea estado caído o sobrepasar el límite de temperatura, la aplicación debe permitir grabar dichas alarmas para poder ser analizadas en un futuro, las cuales se almacenan en un archivo de texto con su fecha, hora y los datos relacionados a cada alarma.

2.2 Descripción general del prototipo

La Figura 2.1 muestra la descripción general del prototipo, el cual se ha divido en la red de sensores inalámbrica y la aplicación de escritorio, detallados a continuación.

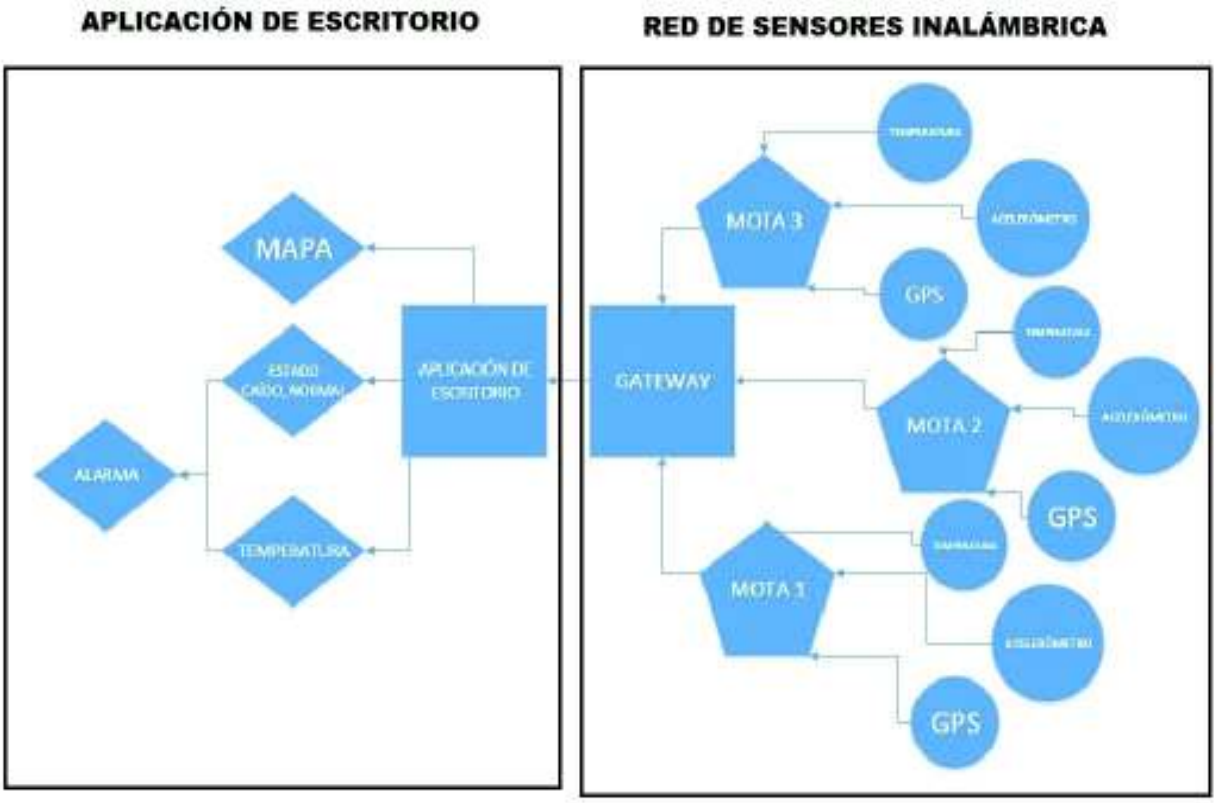


Figura 2.1 Descripción general del prototipo

2.2.1 Red de sensores inalámbrica IEEE 802.15.4

La red de sensores inalámbrica se compone de un *gateway* y una o más motas conectadas entre sí hasta alcanzar al *gateway*; asimismo, cada mota cuenta con un sensor acelerómetro integrado en su placa, un sensor de temperatura acoplado a un puerto digital y un sensor GPS acoplado al *socket* GPS. El estándar define que se pueden conectar hasta 256 nodos a cada *gateway* y permite una red de hasta 256 *gateways*, por consiguiente, la red puede crecer a medida que aumenten los nodos sensores y los *gateway*.

a) Nodo sensor

El nodo sensor o llamado también mota, es un módulo con capacidades reducidas capaz de procesar los datos de los sensores y enviarlos a través de la red inalámbrica, para el prototipo a implementar se selecciona el nodo sensor de la marca libelium llamado Waspmote PRO V1.2, dado que posee puertos de entrada/salida necesarios para cumplir con los requerimientos del prototipo; asimismo se utiliza el sensor acelerómetro incluido y se agregan, un sensor de temperatura y un sensor GPS.

b) Sensor de temperatura

Es un sensor que permite tomar datos de la temperatura a la cual está expuesto en un lugar determinado, estos datos los envía de manera digital al nodo sensor para que este lo procese y envíe en forma de grados centígrados al *gateway*.

c) Sensor acelerómetro

El sensor acelerómetro permite sensar los movimientos en los ejes x, y, z mediante un giroscopio integrado; un giroscopio percibe la rotación del dispositivo, mientras que el acelerómetro tiene la capacidad de medir la orientación del dispositivo, en conjunto logran describir el movimiento mediante los valores de los ejes x, y, z.

d) Sensor GPS

El sensor GPS permite obtener latitud y longitud mediante la conexión a los distintos satélites que encuentra el módulo, para esto requiere tener línea de vista con ellos y así obtener la información, se requiere la conexión con al menos tres satélites para recibir latitud y longitud; esta información la envía directamente al *gateway* para su procesamiento.

e) Gateway

Es el dispositivo que concentra las comunicaciones, recibe la información de todas las motas que pertenecen a una misma red; además, el *gateway* permite procesar dicha información recibida mediante radiofrecuencia y convertirla en información digital siguiendo el estándar IEEE 802.15.4 y así capturar su trama y usar los datos de su carga útil para procesarlos.

2.2.2 Aplicación de escritorio

La aplicación de escritorio permite al usuario final visualizar los datos y ubicar las distintas motas sobre un mapa; por consiguiente, monitorearlas y obtener su temperatura, posición y estado; además, en el caso de sobrepasar el límite de temperatura o encontrarse en estado caído se emite una alarma tanto en la aplicación como en la mota alertando a quien los esté monitoreando. La aplicación recibe la información del *gateway* quien mediante el puerto USB envía los datos de todas las motas y las diferencia con un identificador único.

2.3 Diseño de la red de sensores inalámbrica

Los módulos utilizados en el diseño del prototipo son:

- Módulo GPS.
- Módulo acelerómetro.
- Módulo de temperatura.

- Módulo de radio IEEE 802.15.4 (2.4 GHz) baja potencia.

2.3.1 Módulo GPS

El sensor GPS permite obtener datos de latitud, longitud, fecha, hora, velocidad, dirección; todos estos datos ayudan a ubicar de manera precisa a la mota y usar esta información para complementar la aplicación a implementar.

a) Características técnicas

Para el desarrollo del prototipo se concluye que el sensor GPS JH3, posee mejores características respecto al sensor A1084, dado que dispone de una mayor sensibilidad y su generación es más actual; en la Figura 2.2 se indican los dos componentes del sensor GPS que son la placa con su respectivo microprocesador y la antena, la cual dispone de un conector tipo UFL. El sensor GPS seleccionado para la implementación del prototipo cuenta con las siguientes características [15]:

Figura 2.2 Sensor GPS Libelium JH3

- Marca: Libelium.
- Modelo: JH3 (Telit).
- Sensibilidad:
    - Adquisición: -147 dBm.
    - Navegación: -160 dBm.
    - Seguimiento: -163 dBm.
- Tiempo de inicio:
    - Caliente: <1s.
    - Frio: < 35s.
- Conector de antena: UFL.

- Antena externa: 26 dBi.
- Error de posición < 2.5m.
- Compatibilidad de sistemas: EGNOS[8], WAAS[9], GAGAN[10] y MSAS[11].

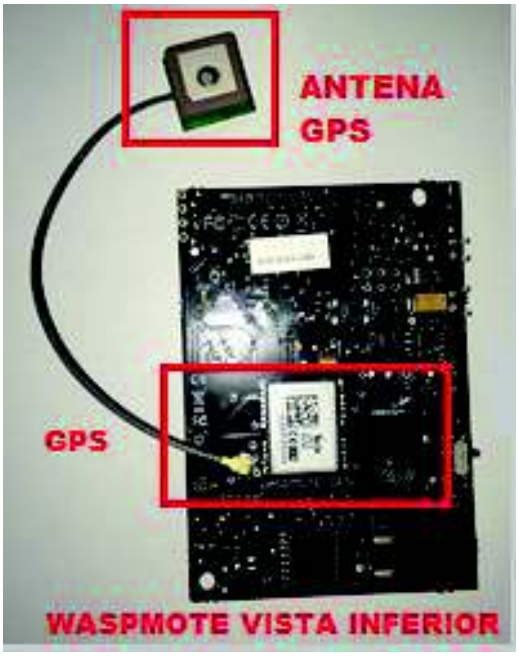


Figura 2.3 Ubicación de sensor GPS sobre la mota

En la Figura 2.3 se puede observar al sensor GPS con su respectiva antena y la vista inferior de la Placa Waspmote, sobre la cual va instalado el sensor; asimismo la placa dispone de un puerto especial para la conexión del sensor GPS. El sensor GPS dispone de un procesador el cual se encarga de interpretar la información que recibe del satélite.

2.3.2 Módulo acelerómetro

La mota dispone de un sensor acelerómetro LIS3331LDH integrado sobre su placa, el cual puede indicar la ubicación exacta en el espacio, mediante los valores de los ejes x, y, z; asimismo, el módulo posee un giroscopio y un acelerómetro de 3 ejes. El acelerómetro permite conocer si el dispositivo se encuentra en forma horizontal o vertical, con lo cual luego de una interpretación se podrá saber el estado es normal o caído. Dado estas

[8] EGNOS (*European Geostationary Navigation Overlay Service*), es una red de satélites geoestacionarios encargados de monitorizar los errores en la señal de GPS.
[9] WAAS (*Wide Area Augmentation System*), es un sistema de aumentación basado en satélites desarrollado por Estados Unidos.
[10] GAGAN (*GPS Aided Geo Augmented Navigation*) similar a WAAS, es un sistema de aumentación basado en satélites desarrollado por la agencia india de investigación espacial.
[11] MSAS (*Multi-functional Satellite Augmentation System*) es un sistema de aumentación basado en satélite multifuncional, diseñado para mejorar l precisión del sistema.

características y la facilidad del acelerómetro incluido en la placa Waspmote, se lo utiliza para la implementación.

a) Características técnicas

En la Figura 2.4 se puede observar al acelerómetro, el cual forma parte de la placa Waspmote; el sensor acelerómetro cuenta con las siguientes características [16]:

Figura 2.4 Módulo acelerómetro integrado a la placa Waspmote[9]

- Marca: Libelium, STMicroelectronics.
- Modelo: LIS3331LDH.
- Acelerómetro de amplio rango:
  - ± 2g.
  - ± 4g.
  - ± 8g.
- Modo de ahorro de energía:
  - 0.5 Hz.
  - 1 Hz.
  - 2 Hz.
  - 5 Hz.
  - 10 Hz.
- Modo normal:
  - 50 Hz.
  - 100 Hz.
  - 400 Hz.
  - 1000 Hz.

Se puede ajustar la sensibilidad, los valores más bajos obtienen una mejor resolución en movimientos lentos, mientras que los valores más altos obtienen una mejor resolución en

movimientos rápidos, el valor por defecto es de FS_2G; el acelerómetro cuenta con los parámetros de sensibilidad descritos en la Tabla 2.1.

Tabla 2.1 Sensibilidad del acelerómetro[16]

FS_2G	$\pm 2g$ (valor por defecto)
FS_4G	$\pm 4g$
FS_8G	$\pm 8g$

2.3.3 Módulo de temperatura

La implementación del prototipo requiere el uso de un sensor de temperatura capaz de enviar el valor de temperatura al cual se encuentra expuesto, para esto se seleccionaron dos sensores que son: LM 35 y DS18B20. Luego de un análisis se concluyó que el sensor que más se adapta a las características del prototipo es el DS18B20, dado que envía los valores de manera digital, contrario a LM 35 que envía valores análogos.

El sensor de temperatura DS18B20 funciona a manera de un termómetro digital que provee temperatura en grados centígrados, adicionalmente provee un único cable para transmisión de datos y dos para alimentación (positivo y negativo); asimismo, puede recibir energía del mismo cable de envío de datos en modo parásito.

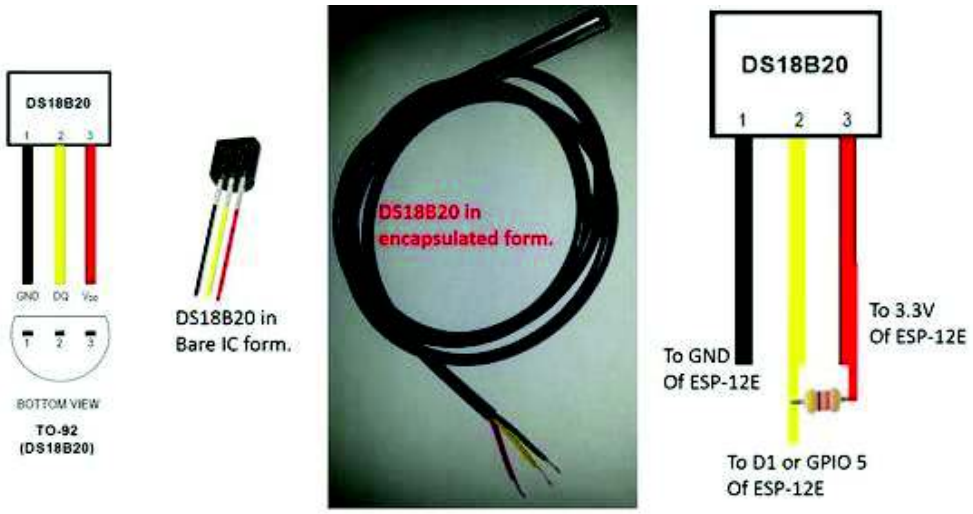


Figura 2.5 Módulo de temperatura [17]

El sensor permite trabajar en conjunto con varios sensores sobre el mismo bus, que se diferencian uno de otro mediante un código serial único de 64 bits. La Figura 2.5 indica los pines del sensor de temperatura; primero Vin que permite polarización de 3 a 5.5 V, luego, GND que indica la conexión a tierra; y, por último, VQ que provee el puerto digital de salida

de datos, adicionalmente indica el uso de una resistencia de 4.7 KΩ entre los pines Vin y VQ para su correcto funcionamiento.

a) Características técnicas

Tabla 2.2 Características técnicas del sensor de temperatura DS18B20 [17]

Característica	Valor
Rango de medición	-55°C - 125°C
Tolerancia	±0.5 °C
Alimentación	3 V - 5.5 V
Resolución de temperatura	9, 10, 11, 12 Bits
	0.5, 0.25, 0.125, 0.0625°C
Resolución por defecto	12 bits
Valor máximo de temperatura	85°C

La Tabla 2.2 muestra las características técnicas que brinda el sensor de temperatura DS18B20, las cuales son: rango de medición, tolerancia, alimentación, resolución y valor máximo de temperatura.

b) Formato de registro de temperatura

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	**BIT 14**	**BIT 13**	**BIT 12**	**BIT 11**	**BIT 10**	**BIT 9**	**BIT 8**
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Figura 2.6 Formato de registro de temperatura [17]

En la Figura 2.6 se observa el uso de los 16 bits que requiere el sensor de temperatura para el envío de información.

c) Relación de datos hexadecimal y binario

En la Tabla 2.3 se describe la interpretación de temperatura mediante una comparación de datos binarios y hexadecimales. La principal función del DS18B20 es su sensor directo de temperatura digital; la resolución del sensor es configurable por el usuario entre 9, 10, 11 y 12 bits que corresponden a incrementos de 0.5, 0.25, 0.125 y 0.0625 °C respectivamente, por defecto se encuentra configurado con sensibilidad de 12 bits.

Tabla 2.3 Datos hexadecimal y binario [17]

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

d) Encendido del equipo mediante una fuente externa

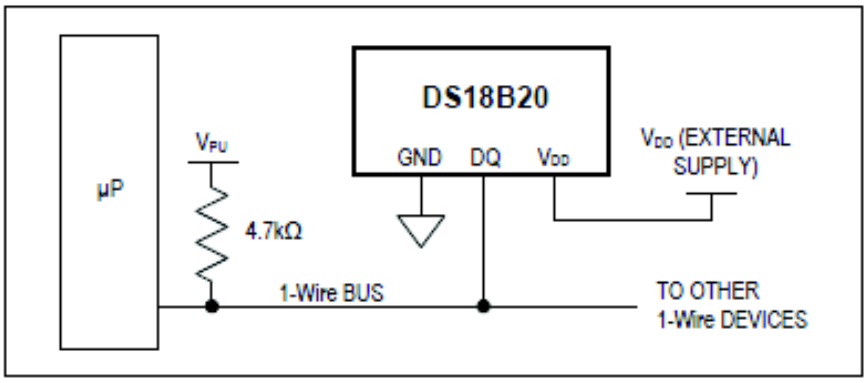


Figura 2.7 Conexión de sensor de temperatura [17]

La Figura 2.7 indica la manera correcta de conectar el sensor de temperatura, el cual requiere del uso de una resistencia externa para su funcionamiento. El equipo utiliza las salidas de voltaje de la placa para su alimentación, utiliza la salida de 3V incluida en la placa, además de la tierra y un puerto digital configurado como puerto de entrada.

2.3.4 Módulo de radio XBee IEEE 802.15.4

Existen dos versiones de radio XBee 802.15.4, una diseñada para Europa y otra con distribución mundial, las dos trabajando en la banda de 2.4 GHz; para la implementación del prototipo se utiliza el módulo de radio XBee S1 802.15.4, dado que se adapta de mejor manera a la placa Waspmote la cual dispone un puerto compatible para esta versión; las características del módulo de radio se detallan en la Tabla 2.4. El alcance varía de acuerdo con el medio donde se utiliza el módulo de radio, en la Tabla 2.4 se define un alcance de 30 m para interiores y 100 m para exteriores; pero, en el capítulo de implementación se definirá el alcance real de la red.

Tabla 2.4 Características del módulo de radio XBee S1 802.15.4 [18]

Característica	Valor
Versión de radio	XBee S1 802.15.4
Número de canales	16
Protocolo	802.15.4
Velocidad de transmisión	250 kbps
Banda de frecuencia	2.4 GHz
Potencia de transmisión	0 dBm (1 mW)
Sensibilidad	-92 dBm
Alcance en interiores	30 m
Alcance en exteriores	100 m

Figura 2.8 Módulo de radio XBee S1 802.15.4

En la Figura 2.8 se muestra el módulo de radio XBee S1 802.15.4 y su antena, la cual es del tipo PCB (*Printed Circuit Board*), es decir, que está integrada en el circuito impreso, por este motivo el alcance del módulo de radio es corto y se limita en distancia. En la Figura 2.9 se muestra el número de canales sobre la banda de frecuencia de 2.4 GHz, se divide en 16 canales con un ancho de banda de 5 MHz cada canal. El módulo de radio XBee añade dos características respecto al protocolo IEEE 802.15.4, las cuales son descubrimiento de módulos y detección de paquetes duplicados.

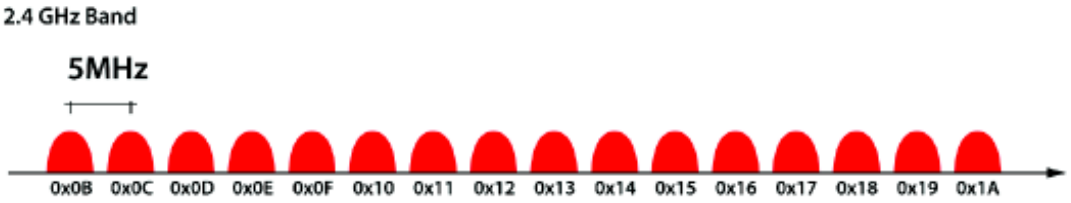


Figura 2.9 Banda de frecuencia y número de canales [18]

- **Descubrimiento de módulos:** Se añade información en la cabecera de la trama, la cual permite enviar un mensaje de descubrimiento de nodos y el resto de los nodos responden indicando su información con un formato especifico que es el identificador de nodo o MAC de 16 bits.
- **Detección de paquetes duplicados**: Añade información del número de trama lo que permite detectar si se reciben tramas duplicadas; las enumera en secuencia desde 1 hasta 255, al sobrepasar este valor se reinicia a 1 dado que para esta función utiliza 8 bits.

a) Distribución de pines

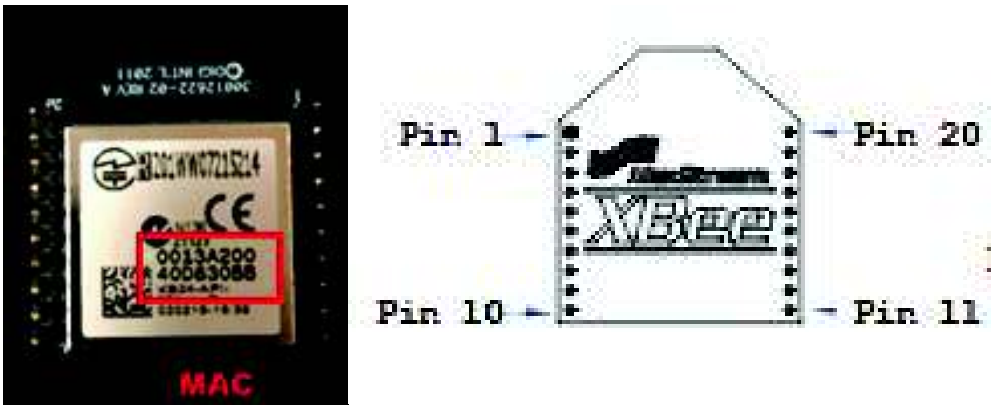


Figura 2.10 Distribución de pines del módulo XBee[19]

En la Figura 2.10 se muestra la distribución de pines del módulo de radiofrecuencia XBee, además, en la vista posterior del módulo se aprecia su dirección MAC. La Tabla 2.5 muestra la función de cada pin del módulo de radio XBee.

Tabla 2.5 Función de pines de XBee [19]

Número de pin	Nombre	Tipo	Descripción
1	VCC	-	Fuente de poder
2	DOUT	Salida	Salida de datos UART

Número de pin	Nombre	Tipo	Descripción
3	DIN	Entrada	Entrada de datos UART
4	DO8	Salida	Salida digital 8
5	RESET	Entrada	Módulo Reset
6	RSSI	Salida	*Reception Signal Strength Indicator*
7	PWM1	Salida	Salida PWM 1
8	RESERVADO	-	No Conectado
9	DI8	Entrada	Entrada digital 8
10	GND	-	Tierra
11	AD4/DIO4	Entrada / Salida	Entrada Analógica 4/ Entrada Salida Digital 4
12	DIO7	Entrada / Salida	Entrada, Salida Digital 7
13	SLEEP	Salida	Indicador de estado de módulo
14	VREF	Entrada	Voltaje de referencia para entrada Analógica o Digital
15	AD5/DIO5	Entrada / Salida	Entrada Analógica 5/ Entrada, Salida Digital 5
16	AD6/DIO6	Entrada / Salida	Entrada Analógica 6/ Entrada, Salida Digital 6
17	AD3/DIO3	Entrada / Salida	Entrada Analógica 3/ Entrada, Salida Digital 3
18	AD2/DIO2	Entrada / Salida	Entrada Analógica 2/ Entrada, Salida Digital 2
19	AD1/DIO1	Entrada / Salida	Entrada Analógica 1/ Entrada, Salida Digital 1
20	AD0/DIO0	Entrada / Salida	Entrada Analógica 0/ Entrada, Salida Digital 0

2.3.5 Transmisión de datos

Los datos que recibe la mota pasan a través de un proceso de modulación de señal para ser enviados mediante el módulo de radiofrecuencia y posteriormente ser recibidos en el *gateway*.

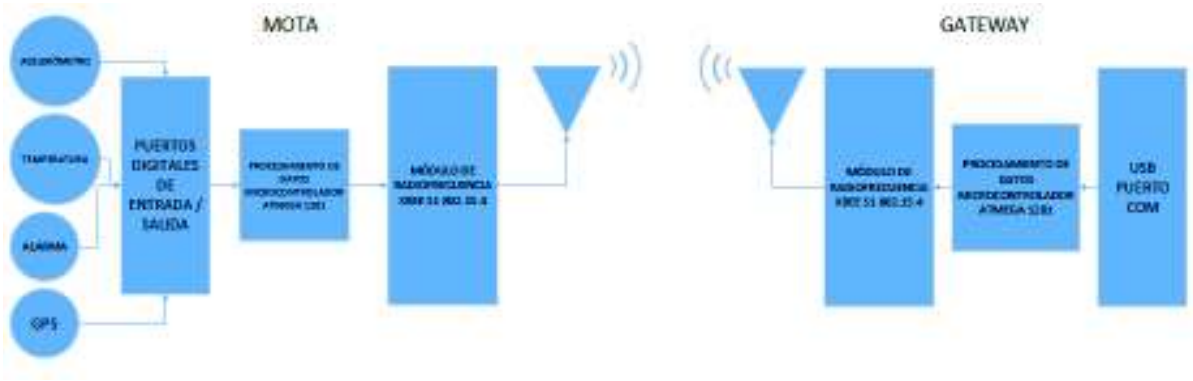


Figura 2.11 Comunicación entre mota y gateway

La Figura 2.11 muestra la comunicación entre el *gateway* y la mota; la mota dispone de un sensor de temperatura, acelerómetro y GPS; los sensores obtienen datos y los envían mediante los puertos de entrada y salida digitales, posteriormente pasan a través del microprocesador quien los procesa y envía al módulo de radiofrecuencia quien modula la señal y la envía al *gateway*, el cual, recibe la información a manera de ondas electromagnéticas y la convierte en una trama que posteriormente será enviada al microcontrolador, quien es el encargado de interpretar dicha información y finalmente enviarla al puerto USB.

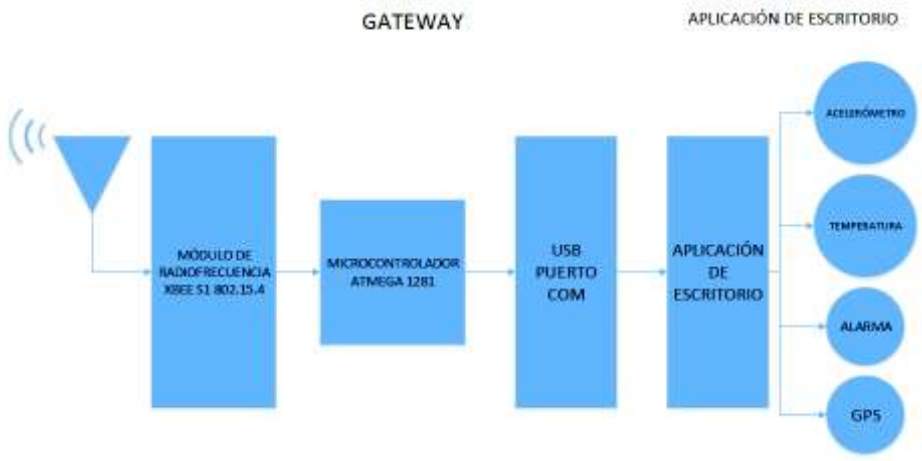


Figura 2.12 Comunicación entre gateway y aplicación de escritorio

La Figura 2.12 muestra el envío de datos entre el *gateway* y la aplicación de escritorio; el *gateway* recibe los datos de las distintas motas a través del módulo de radiofrecuencia, quien posteriormente lo envía al microcontrolador y finalmente lo envía a través del puerto

USB a la aplicación. El *gateway* se encuentra conectado a un computador mediante el puerto USB, asimismo, la aplicación de escritorio se encuentra instalado en el computador, el cual asigna un puerto Com asociado a dicha comunicación; posteriormente la aplicación de escritorio recibe los datos del *gateway* y los procesa de tal manera que le permite desplegar la información de las distintas motas sobre un mapa.

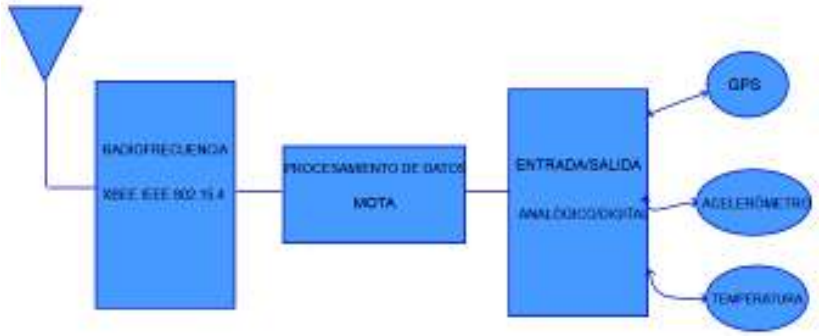


Figura 2.13 Transmisión de datos Mota-Sensores

En la Figura 2.13 se muestra cómo interactúan los distintos sensores en la mota y como se envía información hasta alcanzar el módulo de radiofrecuencia.

- El sensor acelerómetro se encuentra integrado en la placa Waspmote, el cual se conecta a través del microcontrolador quien procesa la información y posteriormente la almacena en la trama.
- El sensor de temperatura se encuentra conectado en un puerto de entrada/salida digital, a través del pin VQ del sensor de temperatura, el puerto digital de la mota recibe la señal digital del sensor y la envía al microcontrolador, quien la procesa y la almacena en la trama.
- El sensor GPS se conecta a través del puerto GPS que posee la mota, esta información viaja de manera digital al microcontrolador, quien posteriormente la almacena en la trama para ser enviada.

Luego el microcontrolador almacena y procesa la información recibida de los distintos sensores; una vez almacenada la información, la envía con un formato de trama al módulo de radiofrecuencia, quien convierte la señal eléctrica en radiofrecuencia para ser enviada en base al protocolo IEEE 802.15.4.

2.3.6 Diseño de la carga útil de la trama en el nodo sensor

Una vez concluida la forma en la que se comunican los distintos componentes del prototipo, se procede a discernir qué información debe enviarse en la trama IEEE 802.15.4, para de

esta manera ser leída desde el *gateway* e interpretada en la aplicación de escritorio. Al programar las distintas motas se asocia la dirección MAC destino que en este caso es la del *gateway*, este parámetro es muy importante dado que, al no incluirlo, las motas no se podrán conectar con el *gateway*. El *gateway* y las motas deben operar en la misma banda de frecuencia y en el mismo canal de comunicación para que puedan intercambiar información en la red.

Para el diseño de la carga útil de la trama, se analiza qué información es importante recibir en el *gateway* y posteriormente en la aplicación de escritorio. El módulo de radiofrecuencia XBee permite el envío de tramas de información; cada trama por defecto necesita de un delimitador de inicio y fin, un identificador y un número de secuencia, adicionalmente permite añadir a la trama información que en este caso es Nombre, Latitud, Longitud, Ángulo y Temperatura; la Figura 2.14 muestra la estructura de la trama.

INICIO DE DATOS	IDENTIFICADOR	NOMBRE	NÚMERO DE TRAMA	GPS (LATITUD, LONGITUD)	ACELERÓMETRO (ÁNGULO)	TEMPERATURA	FIN DE DATOS

Figura 2.14 Estructura de información de la trama

- **Inicio y fin de datos:** Muestra una serie de caracteres que indica el inicio o fin de la carga útil de la trama, principalmente sirve para diferenciar una trama de otra dado que la comunicación que recibe el *gateway* es serial y envía una cadena de tramas.
- **Identificador:** Cada mota posee un identificador único, el cual permite diferenciarlo en una misma red y utilizarlo de mejor manera.
- **Nombre:** Es un valor asignado a cada mota al momento de la programación, es configurable por el administrador, permite el uso de datos tipo *string*.
- **Número de trama:** Indica el número de cada trama enviada de manera secuencial, se lo utiliza para conocer cuantas tramas se han enviado desde el momento en que empezó la comunicación, lo cual permite conocer si se encuentran tramas repetidas o perdidas.
- **Datos de GPS:** Muestra los datos obtenidos por el GPS, separados como Latitud y Longitud.
- **Datos de acelerómetro:** Muestra los datos recibidos por el acelerómetro, presentado como un valor de ángulo que varía entre -90° y 90°.

- **Temperatura:** Muestra los datos obtenidos por el sensor de temperatura, el sensor usado envía la información directamente en grados centígrados.

2.3.7 Diseño de programación de nodos sensores

Para la correcta programación de las distintas motas se utiliza un diagrama de flujo como guía para que cumpla las funcionalidades necesarias.

Figura 2.15 Diagrama de flujo de la programación de las motas

La Figura 2.15 muestra el diagrama de flujo de la programación de las motas; primero, la mota debe encender el módulo de radio, el acelerómetro, el GPS y el puerto digital 7 donde se conecta el sensor de temperatura, luego, entra en el lazo que se va a ejecutar cada 3 segundos; dentro del lazo se crea la trama IEEE 802.15.4, sobre la cual va contenida la información a ser enviada.

Después se intenta conectar al satélite mediante el GPS, al no lograr conectarse durante 20 segundos añade información de latitud y longitud por defecto (0.000000, 0.000000), si recibe datos del GPS añade la información recibida a la trama; continua al sensor de temperatura, si el sensor de temperatura recibe datos los envía a la trama, caso contrario añade un valor por defecto que es -1000, si el valor de temperatura es superior al definido por el usuario emitirá una alarma.

Posteriormente se reciben datos del acelerómetro, al no recibir datos se escribe un ángulo por defecto de 0°; asimismo, si detecta que el ángulo se encuentra entre -45° y 45° no emitirá alarma, caso contrario sonará una alerta; finalmente, se añade esta información a la trama y se la envía; si el envío de la trama falla producirá un error, caso contrario la comunicación será exitosa y regresará al lazo para realizar el mismo proceso cada 3 segundos.

a) Proceso del módulo GPS

Figura 2.16 Diagrama de flujo de la programación del GPS

La Figura 2.16 muestra el proceso que lleva a cabo el GPS para garantizar su conexión a los distintos satélites y recibir información; primero, se enciende el GPS y se espera conexión al satélite, posteriormente se define un estado el cual espera un tiempo de 20 segundos para la conexión a los satélites, transcurrido este tiempo y al no obtener conexión se establece un estado falso, contrario al existir conexión el "estado = *true*", al retornar un estado *true*, es posible obtener la latitud y longitud del dispositivo, esta información es añadida a la trama posteriormente.

b) Proceso del módulo acelerómetro

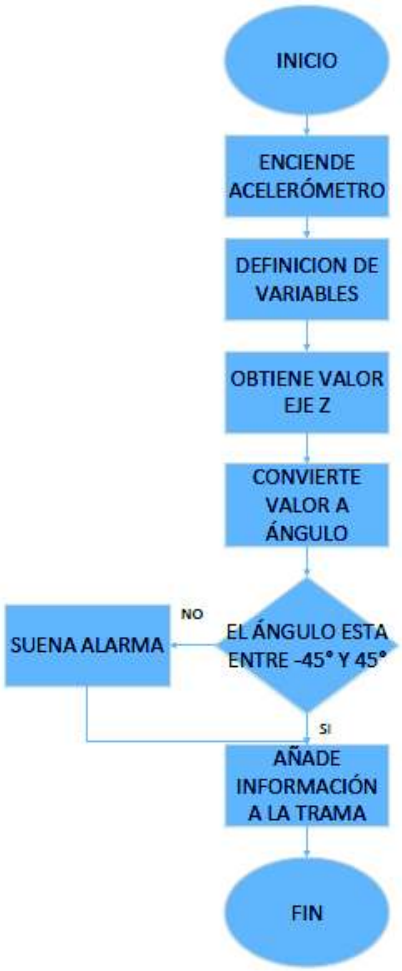


Figura 2.17 Diagrama de flujo de la programación del acelerómetro

La Figura 2.17 muestra el proceso que se lleva a cabo para obtener datos del sensor acelerómetro, con lo cual primero se lo enciende, posteriormente, se define una variable tipo flotante que asocia el valor del eje z; luego, se lo convierte a un valor de ángulo para facilitar la interpretación, asimismo si detecta que el ángulo es mayor a 45° o menor a -45°, emite una alarma en la mota; finalmente, añade la información del ángulo a la trama para posteriormente ser enviada al *gateway*.

En la parte de diseño del prototipo se buscó la mejor manera de recibir los datos del acelerómetro, con lo cual se implementaron dos métodos, el primero fue ubicar al sensor en posición horizontal y vertical e ir tabulando los datos para determinar la posición de caído o normal, el segundo método fue el convertir los valores del eje z en un ángulo, con lo cual se obtiene un ángulo de 0° para la posición vertical y 90° en posición horizontal; los métodos se detallan a continuación:

b.1) Tabulación de datos del acelerómetro

Mediante pruebas realizadas con el sensor acelerómetro se concluye las Tabla 2.6 y 2.7, las cuales muestran los valores obtenidos al ubicar a la mota en posición horizontal con su parte superior hacia arriba o hacia abajo.

Estado caído (Vista superior e inferior de la placa)

Tabla 2.6 Datos del estado de la mota obtenidos del acelerómetro caído 1

Valor x	Valor y	Valor z
-44	-86	1051
-39	-83	1050
-47	-84	1046
-37	-83	1049
-41	-87	1049
-37	-83	1038
-43	-83	1047

Tabla 2.7 Datos del estado de la mota obtenidos del acelerómetro caído 2

Valor x	Valor y	Valor z
13	-38	-988
2	-32	-983
14	-38	-981
5	-30	-981
10	-30	-985
13	-31	-984
6	-32	-976

La Figura 2.18 y Figura 2.19 indican los datos recibidos a través del *gateway* y desplegados mediante una aplicación llamada PUTTY [20]; el cual es un cliente SSH, Telnet y RLOGIN, con licencia libre, el cual permite leer un puerto Com dentro de un computador, mediante esta aplicación se logra leer los datos que el acelerómetro envía para así determinar los parámetros x, y, z que indican la posición horizontal o vertical de a mota.

Figura 2.18 Datos del acelerómetro de la vista superior horizontal

Figura 2.19 Datos del acelerómetro de la vista inferior horizontal

Al tabular los datos se concluye que los estados caídos serían:

Caído 1: Eje x debe estar entre -200 y 200.

Eje y debe estar entre -200 y 500.

Eje z debe estar entre -800 y -1200.

Caído 2: Eje x debe estar entre -220 y 200.

Eje y debe estar entre -200 y 500.

Eje z debe estar entre 800 y 1200.

Normal: El estado normal se aplica cuando ninguna condición de caído se cumple.

b.2) Conversión de datos del acelerómetro a ángulo

El segundo método se diseñó a partir del valor del eje z con el cual se trabaja para convertirlo en un ángulo, una vez realizada la tabulación de los valores x, y, z se aprecia que los valores x e y no son necesarios de considerar dado que no varían y solo el valor de z varía en función de la posición horizontal o vertical de la mota. Para convertir el valor del eje z en un ángulo se debe considerar que el acelerómetro envía valores del eje z entre -1200 y 1200 mg (masa gravedad), a este valor se le aplica la siguiente fórmula:

$$angulo\ z = \frac{180}{\pi} * \operatorname{asin}\frac{z}{1000}$$

Considerando que la conversión resulta ser efectiva se opta por usar este método para la implementación del prototipo.

c) Proceso del módulo de temperatura

Figura 2.20 Diagrama de flujo de la programación del sensor de temperatura

La Figura 2.20 muestra el proceso que se lleva a cabo para obtener el valor de temperatura, dado que el sensor utiliza 3V para su polarización; primero se debe encender el puerto 3V3, posteriormente se enciende el puerto digital 7 por el cual se reciben los datos del sensor de temperatura; además, en la aplicación se define el valor máximo de temperatura, una vez superado dicho valor sonará una alarma, finalmente se añade la información a la trama en grados centígrados. Si el sensor de temperatura no envía un valor, se muestra el valor -1000.

2.4 Implementación de la red de sensores inalámbrica

Para la implementación de la red de sensores inalámbrica se utilizan recursos tanto de hardware como de software y se la realiza mediante el siguiente proceso descritos a continuación:

- Configuración de los módulos de radiofrecuencia XBee 802.15.4.
- Configuración del *gateway*.
- Conexión de los módulos sensores sobre la mota.
- Programación de la mota.

2.4.1 Configuración de los módulos de radiofrecuencia XBee 802.15.4

Para la implementación del módulo de radiofrecuencia se utiliza el módem XBee S1 802.15.4, el cual pertenece a la compañía Digi; esta compañía ofrece una plataforma de configuración para los módulos XBee llamada XCTU, el cual es una aplicación multiplataforma de uso libre compatible con varios sistemas operativos entre ellos Windows, Linux y MacOS. Esta plataforma permite administrar y configurar los dispositivos XBee; además, permite actualizar el *firmware* de los módulos y generar tramas de prueba sin necesidad de tener un módulo de comunicación.

Se utiliza el programa XCTU dado que el fabricante de los módulos XBee lo recomienda, para esto, se procede a descarga el programa de la página de Digi [55] y posteriormente se lo instala en un computador sobre el sistema operativo Windows. La Figura 2.21 muestra la interfaz gráfica de XCTU, la cual permite acceder a los módulos de radiofrecuencia conectados en el puerto USB y posteriormente configurarlos.

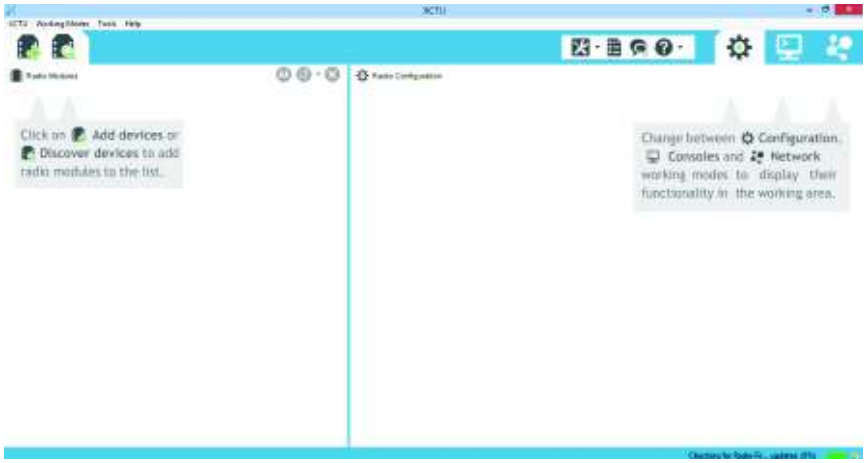


Figura 2.21 Software de configuración XCTU para XBee

Figura 2.22 Conexión de XBee sobre el gateway

Para que el programa logre reconocer y posteriormente configurar los módulos, se lo conecta en el *gateway* como se muestra en la Figura 2.22 de manera que permita conectarlo a un puerto USB y así permitir que el programa lo lea y posteriormente escriba la nueva configuración.

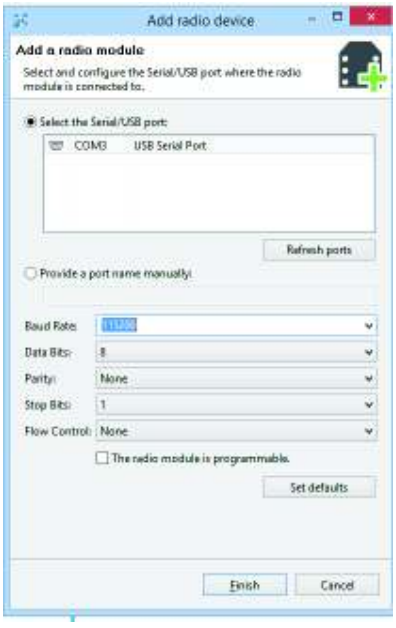


Figura 2.23 Parámetros de configuración de puerto Com

La Figura 2.23 muestra los parámetros a configurar al momento de añadir un nuevo módulo de radio, el fabricante especifica los parámetros que son, velocidad de transmisión de 115200 baudios, tasa de 8 bits, sin paridad, un bit de parada y sin control de flujo; una vez configurados estos parámetros el programa empieza a conectarse con el *gateway*.

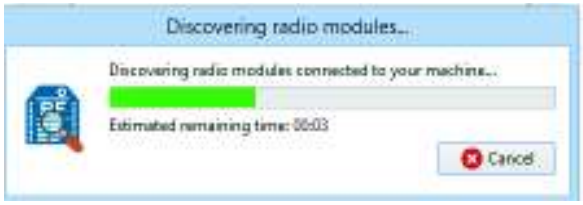


Figura 2.24 Conexión con el gateway mediante el puerto Com

La Figura 2.24 muestra el momento en el que el programa XCTU se conecta al *gateway*, si algún parámetro está mal configurado no existe conexión; un error habitual se da cuando no coincide la velocidad de transmisión, en este caso no existiría conexión.

Figura 2.25 Asociación de un nuevo módulo XBee

La Figura 2.25 muestra al módulo XBee asociado a la aplicación a través del *gateway*, una vez asociado permite observar y configurar sus parámetros que son:

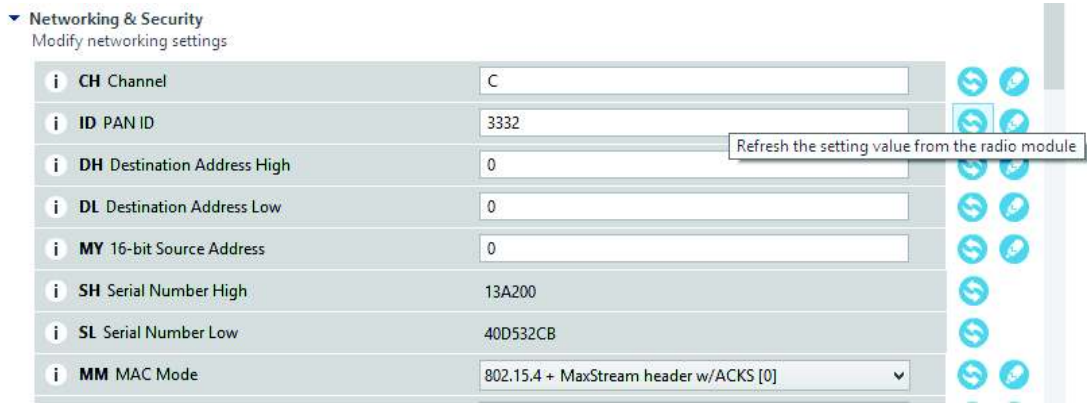


Figura 2.26 Parámetros de configuración de red del módulo XBee

En la Figura 2.26 se identifican los principales parámetros del módulo de radio XBee.

CH canal: Permite seleccionar el canal de frecuencia sobre el cual trabaja IEEE 802.15.4, en este caso se escoge el canal C que pertenece a la banda de 2.405 GHz hasta 2.410 GHz con un ancho de banda de 5 MHz por canal.

PAN ID: Identifica a la red de área personal, todas las motas de la misma red deben pertenecer al mismo PAN ID, en este caso 3332; este parámetro es muy importante dado que solo las motas y *gateway* con el mismo PAN ID se pueden ver y comunicar. Para comunicarse con todas las PAN se debe indicar FFFF.

Dirección de destino: Permite indicar la dirección de destino, que debe ser la del *gateway*; para enviar tramas a todos los módulos se indica una dirección de *broadcast* FFFFFFFF.

Número de serie: El número de serie alto y bajo, son parámetros únicos de cada módulo y estos no pueden ser modificados.

MAC *mode*: Permite definir el valor del modo MAC que va de 0 a 2, se puede habilitar o deshabilitar la detección de paquetes duplicados, además, de incluir o no ACK. Se escoge la opción 0 *MaxStreamHeader* con lo cual envía ACK y número de trama para el manejo de tramas duplicadas.

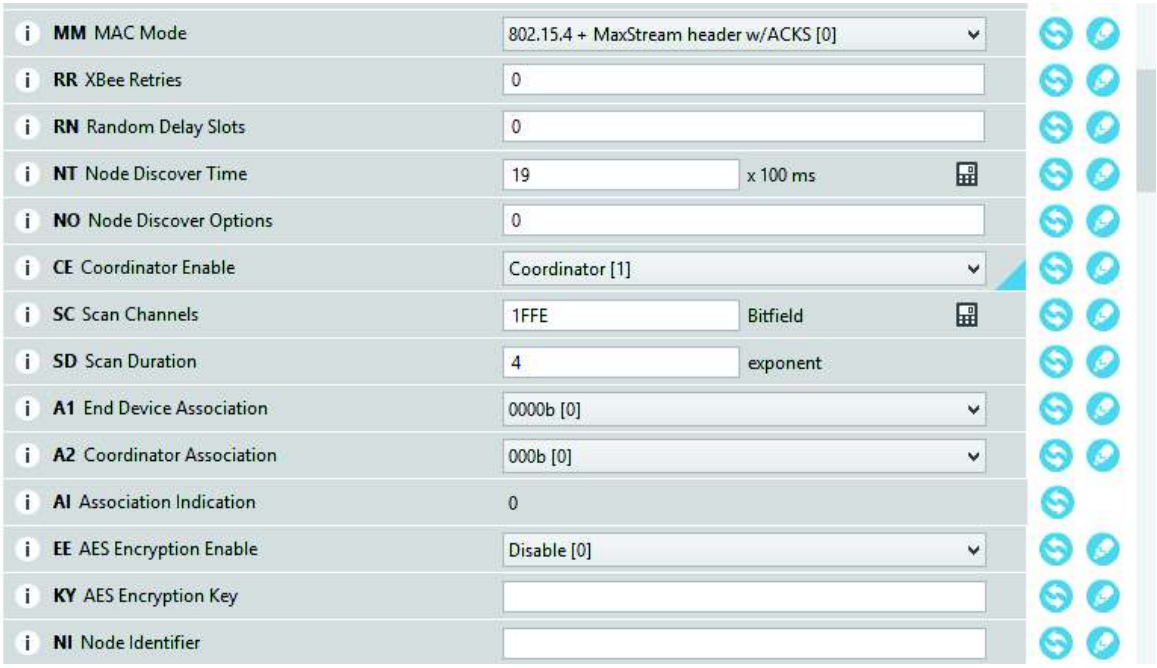


Figura 2.27 Parámetros de configuración de asociación del módulo XBee

En la Figura 2.27 se indica la continuación de los parámetros principales que se configura en el módulo XBee.

Tiempo de descubrimiento de nodos: Define el tiempo máximo para poder descubrir nodos en la red, se establece en 1.0 segundos.

Activar coordinador: Permite seleccionar al dispositivo como coordinador o dispositivo final. La red requiere al menos un coordinador conectado al computador para monitorear a las motas y al menos un dispositivo final que se conectará al *gateway*.

Asociación de coordinador y dispositivo final: Permite opciones de asociación entre el coordinador y el dispositivo final, deben estar en la misma configuración en este caso 000b.

Encriptación AES: Permite enviar la trama de manera encriptada, para el prototipo no se define ningún tipo de encriptación dado que l información a enviar no es crítica en el caso que alguien la capture, además para realizar pruebas del envío de los datos es más rápido si se encuentra en texto plano la información.

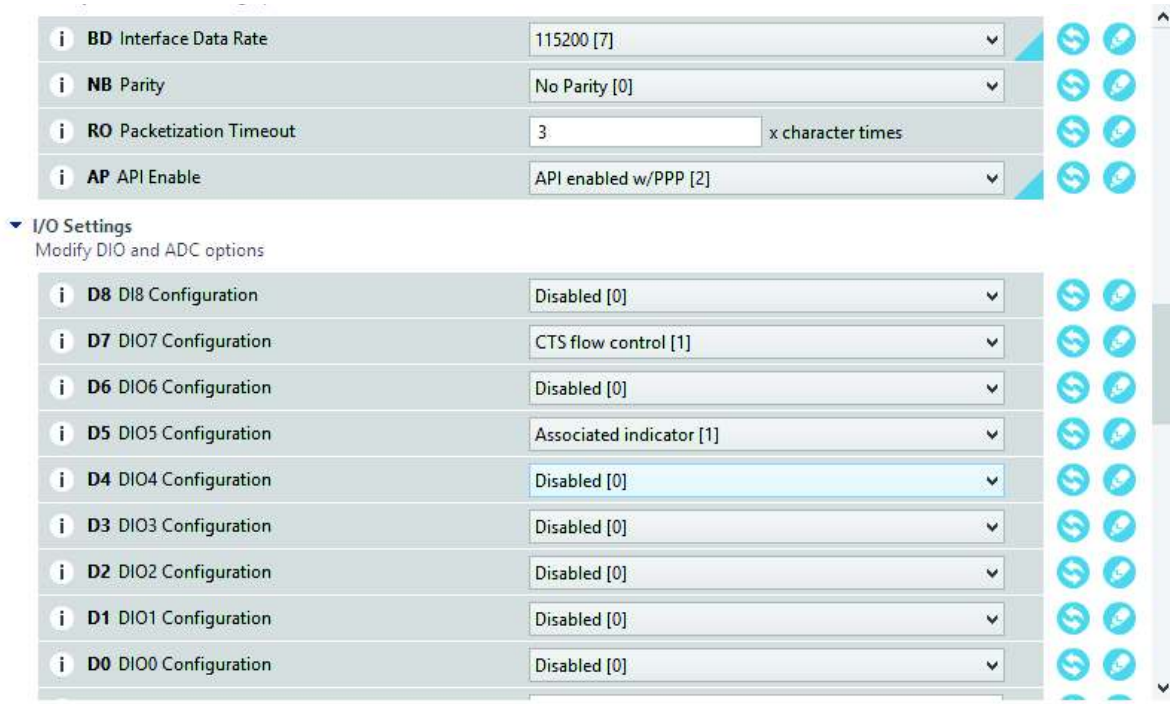


Figura 2.28 Parámetros de configuración entrada y salida del módulo XBee

La Figura 2.28 muestra los parámetros de conexión serial, además, del uso de los puertos digitales que posee el XBee [10].

Velocidad de transmisión: Define la velocidad a la cual se transmiten los datos mediante el puerto serial que es de 115200 baudios.

Paridad: No utiliza ningún bit de paridad, es decir no permite detectar ni corregir errores en la comunicación.

API disponible: Habilita o deshabilita el API.

D0-8: Permite habilitar los puertos de entrada/salida del módulo XBee o configurarlos para una acción específica.

Una vez establecidos los parámetros principales del módulo XBee se procede a su configuración; a continuación, se muestra un ejemplo de cómo se deben configurar los dos módulos descritos en la Tabla 2.8, con la cual todos los dispositivos que pertenecen a la red se conectan al *gateway*. Los nodos deben pertenecer al mismo PAN ID, canal y el dispositivo final siempre debe apuntar hacia el *gateway*, además, se puede realizar la transmisión a varios nodos, con lo que se indica la dirección destino 0XFFFF; con lo cual se consigue que las motas alcancen al *gateway*.

Tabla 2.8 Ejemplo de configuración del módulo XBee

XBee A Valores	**XBee B Valores**
CANAL C	CANAL C
DH 0000FFFF	DH 0000FFFF
DL 00000000	DL 00000000
COORDINADOR	END-DEVICE
PAN ID 234	PAN ID 234
SH 13A200 (viene por defecto)	SH 13A200 (viene por defecto)
SL 4076E26E (viene por defecto)	SL 4076E267 (viene por defecto)

2.4.2 Configuración del gateway

El *gateway* requiere ciertos parámetros de configuración para poder ser usado en la red; por una parte, debe estar configurado como un nodo coordinador con lo cual centrará las comunicaciones en él; por otra parte, se deben definir los parámetros de comunicación mediante el puerto USB, que son velocidad de transmisión, paridad, bits de parada y control de flujo; todos estos parámetros se los modifica mediante la herramienta XCTU. Para poder configurar los módulos que no son coordinadores, se debe conectar el módulo XBee de cada mota en el *gateway* y configurarlo separadamente mediante este dispositivo y posteriormente conectarlo a cada mota.

2.4.3 Programación de las motas

Una vez establecida la comunicación entre el *gateway* y las motas mediante los módulos XBee, se procede a la programación de las motas, con el propósito de recibir los datos de los distintos sensores y realizar acciones dependiendo del prototipo.

Primero se debe descargar la interfaz de desarrollo Waspmote PRO-IDE desde la página oficial de libelium [21]. La plataforma acepta la descarga del entorno en diferentes sistemas operativos entre ellos Windows, Linux y MacOS [10]; para la implementación del prototipo, se instala el programa Waspmote PRO-IDE sobre Windows; una vez instalados y configurados los parámetros necesarios para la comunicación de las motas y el *gateway*, se realiza la implementación del software sobre cada mota y sensor; es decir, se carga el código a las motas que permite el manejo de los distintos sensores a través de la mota.

```
9  //libreria de transmisión inalámbrica 802.15.4
10 #include <WaspXBee802.h>
11 //libreria para el manejo de tramas
12 #include <WaspFrame.h>
13 //libreria para el manejo del GPS
14 #include <WaspGPS.h>
```

Código 2.1 Librerías

El Código 2.1 muestra las librerías usadas en el código implementado en las motas, el módulo de comunicación usado es un XBee S1 802.15.4. el cual permite configurarse para asociar al *gateway* y las motas; asimismo, al usar el módulo de comunicación se debe usar la librería descrita en la línea 10, la cual crea un objeto de la clase WaspXBee802, este objeto es llamado XBee802, al momento de crear el constructor algunas variables se crean por defecto y se inicializan; la línea 12 muestra la librería *WaspFrame* que permite la creación y uso de tramas 802.15.4; la línea 14 contiene la librería para el manejo del sensor GPS.

```
16 //define ángulo en función del acelerómetro
17 float angulo_z;
18 //define temperatura
19 float temp = 0;
20 //variable para el envío y recepción de datos
21 uint8_t error;
22 //define la temperatura máxima a la cual produce una alarma
23 float valor_maximo = 30;
24 //define tiempo para conectar con satélite
25 #define TIMEOUT 20
26 //define estado de conexión GPS
27 bool estado;
28 //MAC address del gateway
29 char mac_gateway[] = "0000000000000FFFF";
30 //identificador de mota
31 char id_mota[] = "NodoFabian";
```

Código 2.2 Variables

El Código 2.2 muestra las diferentes variables que se crean para el manejo de la mota; la línea 17 define una variable tipo flotante para asignarla al ángulo z; la línea 19 define una variable tipo flotante para asignar el valor de temperatura y se la inicializa en 0; la línea 21 define la variable error que se usa para el envío y recepción de información; la línea 23

asigna un valor flotante para asignar el valor máximo de temperatura; la línea 25 define un tiempo de *Timeout* el cual espera 20 segundos y trabaja en conjunto con el sensor GPS.

La línea 27 define el estado que posteriormente permite saber si es caído o normal, finalmente las líneas 29 definen la Mac de broadcast que es 000000000000FFFF y la línea 31 asignan un nombre a la mota, cada módulo de radio frecuencia posee un identificador único llamado MAC, el cual está compuesto por 64 bits y se los divide en 2 grupos de 32 bits (alto y bajo).

```
33 void setup()
34 {
35   //enciende puerto de 3 voltios
36   PWR.setSensorPower(SENS_3V3,HIGH);
37   //enciende acelerómetro
38   ACC.ON();
39   //enciende puerto USB
40   USB.ON();
41   //almacena identificador de mota en la memoria
42   frame.setID( id_mota );
43   //enciende módulo XBee
44   xbee802.ON();
45   //enciende GPS
46   GPS.ON();
47
48   //comprueba conexión a satélite
49   if( estado == true )
50   {
51     // configura la hora para uso de GPS
52     GPS.setTimeFromGPS();
53     //enciende leds, parpadea
54     Utils.blinkLEDs(1000);
55   }
56 }
```

Código 2.3 Código Setup

El Código 2.3 define los parámetros *Setup* que se ejecutan una sola vez, cuando se enciende la mota; la línea 36 enciende el puerto de energía de 3V, usado para energizar el sensor de temperatura; la línea 38 enciende el sensor acelerómetro. La línea 40 enciende el puerto USB que se lo usa para comprobación de comunicación de la mota, mediante la herramienta Putty se puede leer el puerto Com asociado a la mota y saber cuál información está transmitiendo; la línea 42 añade el nombre a la trama; la línea 44 enciende el módulo de radiofrecuencia XBee; la línea 46 enciende el sensor GPS; finalmente, desde la línea 49 a 55, se comprueba el estado del GPS, luego de una espera de 20 segundos, define la hora del GPS y enciende los leds.

```
58 void loop()
59 {
60   //Manejo del acelerómetro
61     //obtiene datos de acelerometro
62     byte check =ACC.check();
63     //toma los datos del eje z del acelerómetro
64     float z_acc = ACC.getZ();
65     if (z_acc > 1000) z_acc = 1000;
66     if (z_acc < -1000) z_acc = -1000;
67     //trasforma el dato del eje z a ángulo
68     angulo_z = (180 / PI) * asin(z_acc/1000);
```

Código 2.4 Código Loop

El Código 2.4 define los parámetros *Loop* que se ejecutan varias veces formando un lazo el cual está definido en 3 segundos; la línea 62 comprueba la información del acelerómetro y la línea 64 la almacena en una variable tipo flotante asignada como z_acc; desde la línea 65 hasta la 68 se convierte la información del eje z en un ángulo, el cual varía desde -90° hasta 90° con lo cual es más fácil determinar el estado de la mota si es caído o normal.

```
70     //manejo de alarmas de la mota
71     if (angulo_z > 45)//si el angulo es mayor a 45 grados activa la alarma
72     {
73       //suena la alarma conectada al puerto digital1
74       analogWrite(DIGITAL1, 190);
75       //enciende led rojo
76       Utils.blinkRedLED();
77     }
78     //si el ángulo es menor a -45°, se activa la alarma
79     else if ( angulo_z < -45)
80     {
81       //suena la alarma conectada al puerto digital1
82       analogWrite(DIGITAL1, 190);
83       //enciende led rojo
84       Utils.blinkRedLED();
85     }
86     //si la temperatura es mayor al valor maximo establecido, enciende alarma
87     else if ( temp > valor_maximo)
88     {
89       //suena la alarma conectada al puerto digital1
90       analogWrite(DIGITAL1, 10);
91       //enciende led rojo
92       Utils.blinkRedLED();
93     }
```

Código 2.5 Manejo de alarmas

El Código 2.5 permite el manejo de alarmas; de la línea 71 a la 77 es un condicional, si el ángulo es mayor a 45°, enciende el puerto digital1 donde se encuentra conectada la alarma, en ese momento emite un sonido; de la línea 79 a la 85 es un condicional, si el ángulo es menor a -45°, enciende el puerto digital1 y emite un sonido; de la línea 87 a la 93 es un condicional, si la temperatura es mayor al valor máximo emite un sonido en base a la alarma conectada al puerto digital1; caso contrario no emitirá ningún sonido.

```
100     //Creación de trama
101     //se crea la trama ASCII
102     frame.createFrame(ASCII);
103     //se usa el puerto digital 7 para conectar el sensor de temperatura
104     pinMode (DIGITAL7, INPUT);
105     //lee el sensor de temperatura del puerto digital 7
106     temp = Utils.readTempDS1820 (DIGITAL7);
107
108     //manejo del GPS
109     //al no existir comunicacion al satélite, añade información inicial a la trama latitud 0.0000
110     //añade valor de latitud y longitud a la trama
111     frame.addSensor(SENSOR_GPS,
112                     GPS.convert2Degrees(GPS.latitude, GPS.NS_indicator),
113                     GPS.convert2Degrees(GPS.longitude, GPS.EW_indicator));
114   //añade el ángulo a la trama
115     frame.addSensor(SENSOR_STR, angulo_z);
116     //añade el valor de temperatura a la trama
117     frame.addSensor(SENSOR_TCA, temp);
118     //muestra la trama
119     frame.showFrame();
```

Código 2.6 Creación de tramas

El Código 2.6 permite crear las tramas a ser enviadas mediante el módulo de radiofrecuencia hacia el *gateway*; la línea 102 crea la trama en formato ASCII; las líneas 104 y 106 usan el valor del puerto digital 7 y lo almacenan en la variable temperatura; la línea 111 recibe la información del GPS y la añade a la trama, se convierte el valor de latitud y longitud de valores GMS (Grados, Minutos, Segundos) a grados decimales dado que requiere menor envío de caracteres desde la mota al *gateway*, lo cual produce ahorro de energía en la mota; la línea 115 añade el valor del ángulo z en la trama; la línea 117 almacena el valor de temperatura en la trama, sobre cada tipo de dato de cada sensor es necesario añadir la línea *frame.addSensor*; finalmente, la línea 119 muestra la trama creada para posteriormente ser enviada.

```
121     //envia la trama al gateway
122     error = xbee802.send( mac_gateway, frame.buffer, frame.length );
123
124     //comprueba bandera de tx
125     if ( error == 0)
126     {
127     USB.println(F("enviado correcto"));
128     //parpadea led verde al enviar correctamente
129     Utils.blinkGreenLED();
130     }
131     else
132     {
133     USB.println(F("envia error"));
134     //parpadea led rojo
135     Utils.blinkRedLED();
```

Código 2.7 Envío de trama

El Código 2.7 muestra cómo se envía la trama creada hasta el módulo de radiofrecuencia que posteriormente lo transmitirá hacia el *gateway*; la línea 122 envía la trama con dirección a la dirección MAC del *gateway*; las líneas 125 hasta 136 toman el valor de error y si se envía correctamente muestra el mensaje de envío correcto sobre el puerto USB, este mensaje se puede observar al conectar la mota con un cable al puerto USB y abrir una consola que lea el puerto Com, caso contrario envía error de comunicación.

```
138     //recepción de información
139     // Recibe el paquete xbee (escucha cada 500ms)
140     error = xbee802.receivePacketTimeout(500);
141
142     // revisa si existe respuesta
143     if( error == 0 )
144     {
145     // muestra los datos almacenados en el payload
146     USB.print(F("Data: "));
147     USB.println( xbee802._payload, xbee802._length);
148     }
149     else
150     {
151     // Imprime mensaje de error en el caso de no recibir nada
152     /*
153      * '7' : Buffer full. Not enough memory space
154      * '6' : Error escaping character within payload bytes
155      * '5' : Error escaping character in checksum byte
156      * '4' : Checksum is not correct
157      * '3' : Checksum byte is not available
158      * '2' : Frame Type is not valid
159      * '1' : Timeout when receiving answer
160     */
161     USB.print(F("Error receiving a packet:"));
162     USB.println(error,DEC);
163   }
```

Código 2.8 Recepción de tramas

El Código 2.8 muestra el proceso de recepción de las tramas que envía el *gateway*, para esto en la línea 140 escucha la trama cada medio segundo; posteriormente, al no existir errores muestra el mensaje, caso contrario indica un mensaje de error que ocurre por diversas situaciones detalladas a continuación.

1.- Se acaba el tiempo de espera de recepción del mensaje.

2.- El tipo de trama no es válido.

3.- El byte de *checksum* no está disponible.

4.- El *checksum* no es correcto.

5.- Error de caracter de escape en el *checksum*.

6.- Error de caracter de escape en la carga útil.

7.- El *buffer* está lleno, no hay suficiente memoria.

```
165   //espera tiempo especifico para obtener senal gps
166   //el timeout esta definido en 20 segundos
167     estado = GPS.waitForSignal(TIMEOUT);
168   //Se repite el lazo cada 3 segundos
169     delay (3000);
170 }
```

Código 2.9 Espera de conexión a satélite

El Código 2.9 muestra el estado de conexión del satélite con un tiempo de espera máximo de 20 segundos, en este caso busca los satélites y en el caso de no lograr conectarse luego de 20 segundos, regresa al lazo y vuelve a iniciar; en la línea 169 se muestra el retardo de 3 segundos que establece el periodo del lazo, es decir cada 3 segundos se inicia.

2.4.4 Carga del código en el nodo sensor

Concluida la programación de las diferentes motas, se procede a cargar el código en cada una de ellas, solamente cambiando el nombre de la mota cada vez que se cargue en una distinta; se debe conectar un cable Mini-USB a la mota y el otro extremo USB al computador, como se observa en la Figura 2.29.

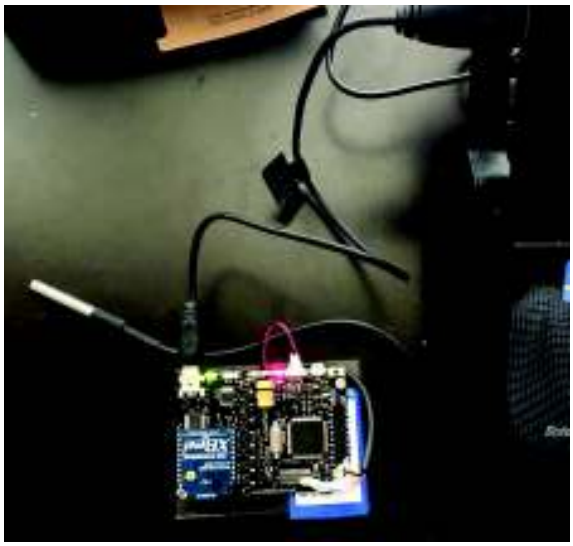


Figura 2.29 Conexión Mota-computador

Antes de cargar la información a la placa se debe comprobar que el código no contenga errores, para esto primero se debe compilar para generar el código de máquina y adicionalmente nos muestra si existen errores; dentro de la interfaz de desarrollo se debe seleccionar el puerto Com asociado a la mota y además la placa que se está usando.

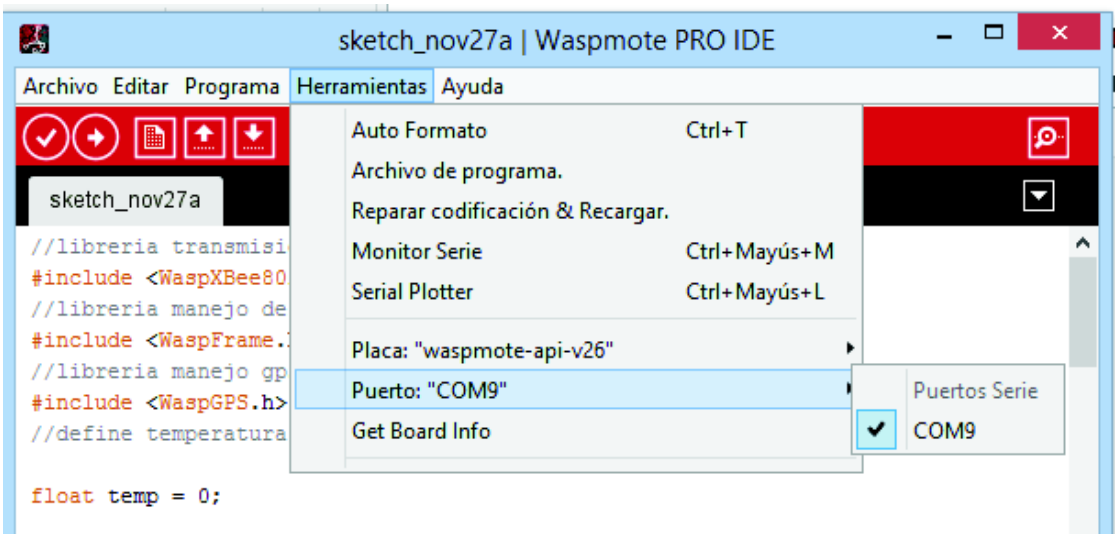


Figura 2.30 Selección del puerto Com

La Figura 2.30 describe la selección del puerto conectado a través del conector USB; esta información se despliega en la opción “Herramientas, Puerto” e indica los dispositivos conectados al computador; realizada esta acción se procede a cargar el código con lo cual el programa primero compila el código y si este no posee errores lo carga; una vez finalizado indica un aviso de subido, en ese momento se sabe que se cargó el código y se puede desconectar la mota y usarla. El código completo de las motas se presenta en el Anexo G del documento.

2.5 Diseño de la aplicación de escritorio

La aplicación de escritorio se encarga de interpretar la información recibida por las motas a través del *gateway*, al momento de diseñar la aplicación de escritorio se crea un esquema general de la aplicación el cual servirá de base para el diseño; en este esquema se definen la ubicación de los diferentes controles, las clases y los métodos que se van a usar.

2.5.1 Diagrama de clases

Para la correcta comprensión del código, en la Figura 2.31 se muestra como interactúan las distintas clases en la aplicación. La Figura 2.32 muestra el diagrama de casos de uso que se realizó en el prototipo.

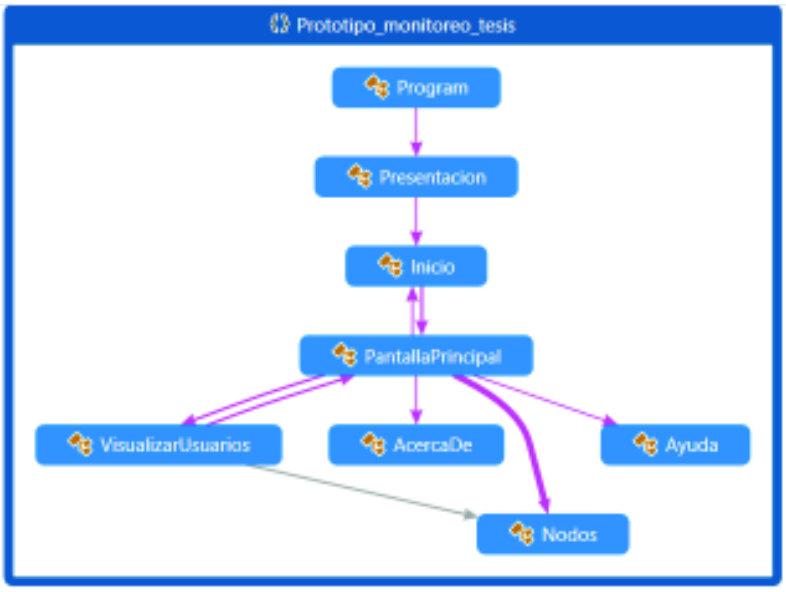


Figura 2.31 Interacción de clases

Finalmente, se muestra en la Figura 2.33 el diagrama de clases para la aplicación de escritorio.

2.5.2 Diagrama de casos de uso

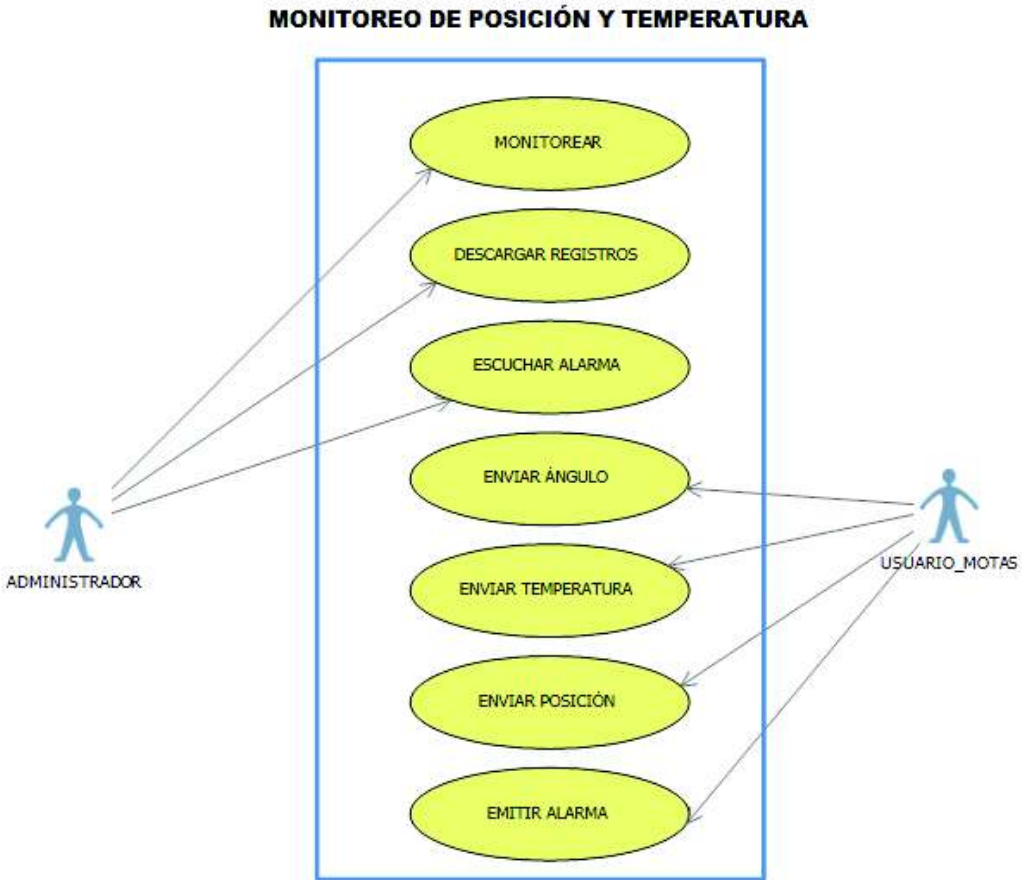


Figura 2.32 Diagrama de casos de uso

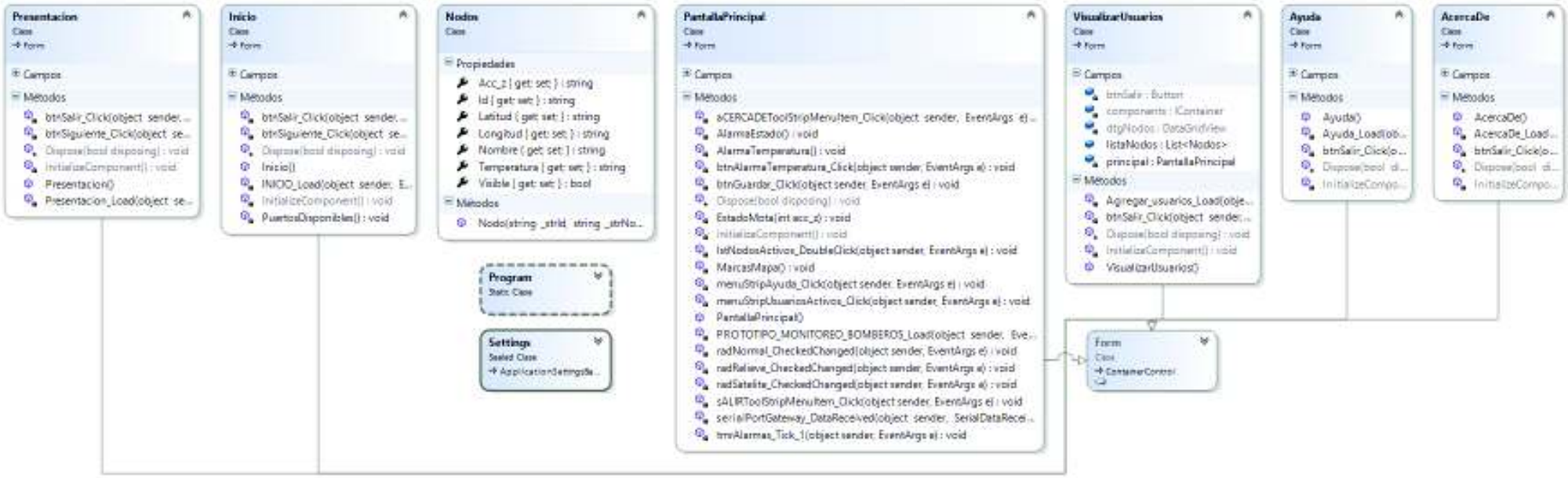


Figura 2.33 Diagrama de clases

Tabla 2.9 Caso de uso monitorear

CU-001	Monitorear
Actor	Administrador
Descripción	El sistema deberá permitir a la persona que monitoree las motas, en cualquier momento poder observar que sucede con cada mota que pertenece a la red y está asociada mediante el *gateway*.
Secuencia Normal	1+ El administrador inserta el *gateway* en el puerto USB del computador. 2+ Se debe correr la aplicación de escritorio.
	2+ Se despliega la pantalla de presentación de la aplicación. 3+ El administrador pulsa el botón siguiente y se muestra la pantalla de inicio.
	4+ En la pantalla de inicio el administrador escoge el puerto Com asociado al *gateway*. 5+ El administrador pulsa el botón siguiente y se despliega la pantalla inicial.
	6+ La pantalla inicial permite observar en un mapa las distintas motas asociadas a la red inalámbrica. 7+ El administrador puede observar la información asociada a cada usuario (id, nombre, latitud, longitud, estado y temperatura). 8+ En el caso de ocurrir una alerta, sobre el mapa se desplegará un aviso indicando la información del usuario.
Excepciones	4’ No existen registros de puertos Com. + Comprobar que el *gateway* esté conectado correctamente. 5’ El botón siguiente despliega un aviso. +Al no seleccionar un puerto o seleccionar un puerto que se encuentra en uso, el sistema envía un aviso.

Excepciones	6’ No se carga el mapa. + Comprobar la conexión a internet. 6’ No se despliegan usuarios en el mapa. +No se han asociado usuarios a la red, encender las motas y esperar.
Excepciones	8’ No se despliega automáticamente la información de las motas. + Solo en el caso de ocurrir una alerta se muestra automáticamente la información, caso contrario se debe poner el cursor sobre la marca.

Tabla 2.10 Caso de uso descargar registros

CU-002	Descargar registros
Actor	Administrador
Descripción	El administrador puede descargar los registros de las alarmas ocurridas durante el proceso de monitoreo.
Secuencia Normal	1+ El administrador puede descargar los registros de las alarmas en el momento que desee. 2+ Sobre un textbox se escriben las alarmas generadas con su fecha y hora. 3+ Se debe presionar el botón descarga de registros y seleccionar la ubicación del archivo que se desea guardar y su nombre.
Excepciones	3’ El registro guardado no contiene información. + Solo si existen alarmas de temperatura o posición se graban datos en el registro.

Tabla 2.11 Caso de uso escuchar alarma

CU-003	Escuchar alarma
Actor	Administrador
Descripción	El administrador puede escuchar un sonido originado por las alarmas de temperatura y posición.

Secuencia Normal	1+ Al sobrepasar el límite establecido de temperatura, la aplicación emitirá una alarma y sobre la marca se desplegará la información del usuario asociado a dicha alarma. 2+ Al determinar que el estado es caído, la aplicación emitirá una alarma y sobre la marca se desplegará la información del usuario asociado a dicha alarma.
Excepciones	1' No suena alarma. + Solo si existen alarmas de temperatura o posición emite un sonido.

Tabla 2.12 Caso de uso Enviar ángulo

CU-004	Enviar ángulo
Actor	Usuario
Descripción	La mota recibe del acelerómetro la posición z y la convierte a ángulo, para posteriormente enviarla al *gateway*.
Secuencia Normal	1+ La mota a través del acelerómetro recibe el dato del eje z, el cual luego de procesarlo lo convierte a ángulo.
Excepciones	1' Llega un dato erróneo. + Al tomar un dato erróneo envía un mensaje de aviso de conversión de acelerómetro.

Tabla 2.13 Caso de uso enviar temperatura

CU-005	Enviar temperatura
Actor	Usuario
Descripción	La mota recibe del sensor de temperatura el valor en grados centígrados, para posteriormente enviarla al *gateway*.
Secuencia Normal	1+ La mota a través del sensor de temperatura envía el valor al *gateway*.
Excepciones	1' Llega un dato erróneo. + Al tomar un dato erróneo envía un valor de -1000 que significa que no está trabajando el sensor.

Tabla 2.14 Caso de uso Enviar posición

CU-006	Enviar posición
Actor	Usuario
Descripción	La mota recibe del sensor GPS los valores de latitud y longitud, para posteriormente enviarla al *gateway*.
Secuencia Normal	1+ La mota a través del sensor GPS recibe los valores de latitud y longitud, posteriormente los envía al *gateway*.
Excepciones	1' Llega un dato erróneo. + Al tomar un dato erróneo envía un valor de 0.00000 que significa que no está trabajando el sensor.

Tabla 2.15 Caso de uso Emitir alarma

CU-007	Emitir alarma
Actor	Usuario
Descripción	El usuario puede escuchar un sonido originado por las alarmas de temperatura y posición.
Secuencia Normal	1+ Al sobrepasar el límite establecido de temperatura, la mota emitirá un sonido de alarma. 2+ Al determinar que el estado es caído, la mota emitirá un sonido de alarma.
Excepciones	1' No suena alarma. + Solo si existen alarmas de temperatura o posición emite un sonido.

2.5.3 Diseño de pantalla de bienvenida

La pantalla inicial permitirá desplegar y seleccionar el puerto Com disponible donde se encuentra conectado el *gateway*, esto se consigue con la ayuda de la librería *System.IO* y la función *System.IO.Ports.SerialPort.GetPortNames()*, la cual permite desplegar los puertos disponibles; el botón salir permitirá cerrar la aplicación y el botón siguiente recogerá los datos seleccionados de la lista de puertos y los enviará al siguiente formulario, además

de abrir el formulario principal. La Figura 2.34 indica el primer boceto de la pantalla de inicio de la aplicación de escritorio.

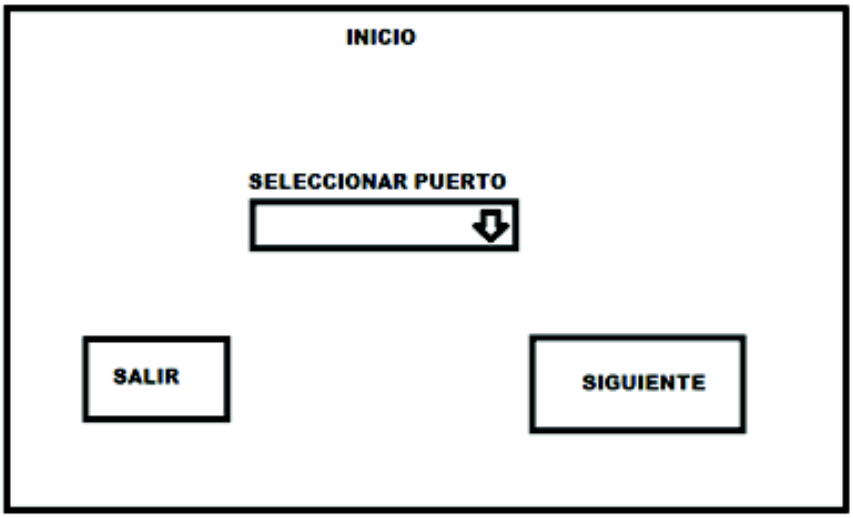


Figura 2.34 Esquema de pantalla de inicio

Con el fin de crear un ambiente gráfico más agradable y sencillo, se usa la herramienta llamada “Pencil” [24], la cual permite de manera fácil y rápida crear bocetos de aplicaciones; “Pencil” permite crear varios tipos de interfaces de usuario dado que posee la mayoría de las figuras que usa un desarrollador; la Figura 2.35 muestra la forma base de la pantalla inicial a ser desarrollada.

Figura 2.35 Segundo esquema de la pantalla de inicio

2.5.4 Diseño de pantalla principal

En la pantalla principal se encontrará un menú que permitirá seleccionar la opción de “Archivo” y desplegar la opción de “Salir” de la aplicación, además la opción “Ayuda” que

cuenta con una pestaña “Acerca de” la cual permite obtener información sobre el prototipo y su desarrollador.

La parte principal de la aplicación es el mapa sobre el cual se desplegará un marcador que indicará la posición de la mota, además del nombre, identificador, temperatura, estado, latitud y longitud de cada mota que forma la red; asimismo, cuenta con el bloque de controles de mapa en el que se puede seleccionar el tipo de mapa, entre ellos “Satélite”, “Normal” y “Relieve”; además, de un control de zoom que mediante la rueda del cursor permite alejar o acercar el mapa de acuerdo a las necesidades del usuario.

En la sección de alarmas, en el caso de ocurrir una alarma ya sea por sobrepasar el nivel de temperatura máximo o encontrarse caído, emitirá un sonido alertando dicha situación; además, permitirá grabar dichas alarmas sobre un archivo de texto en el que se escribirán los datos relacionados a cada mota involucrada en la alarma; la Figura 2.36 indica el primer boceto de la pantalla principal de la aplicación de escritorio.

Para poder manejar todos los controles que presenta la pantalla de inicio y principal, se crean diferentes métodos que realizan acciones específicas desarrolladas para su interpretación. Cada método se encarga de desarrollar una tarea que en conjunto recrea el mapa y despliega la información recibida por todas las motas.

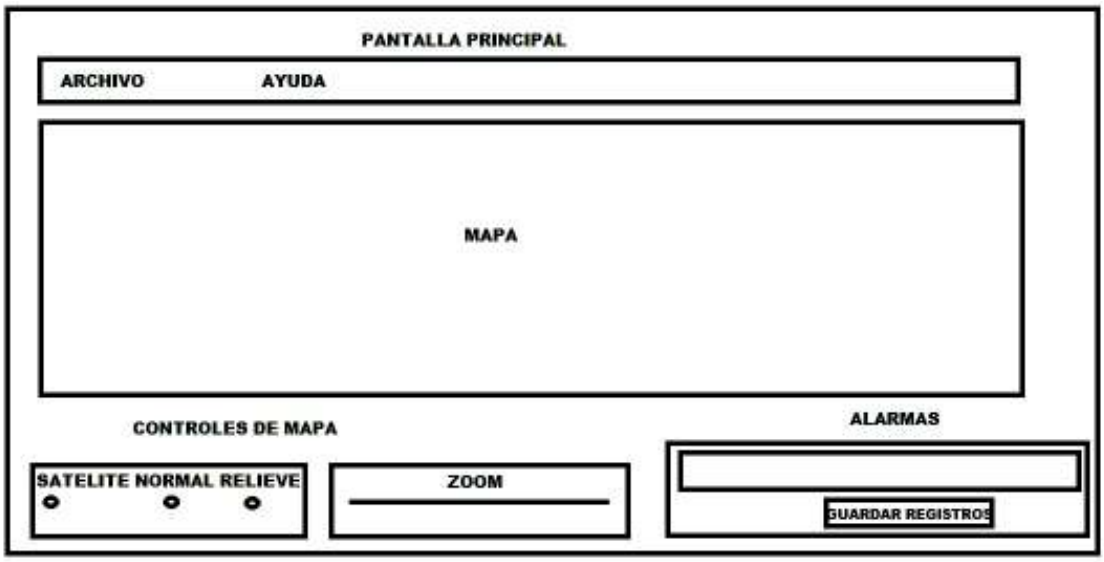


Figura 2.36 Esquema inicial de la pantalla principal

La Figura 2.37 indica el esquema de la pantalla principal de la aplicación que se lo realizará en la implementación.

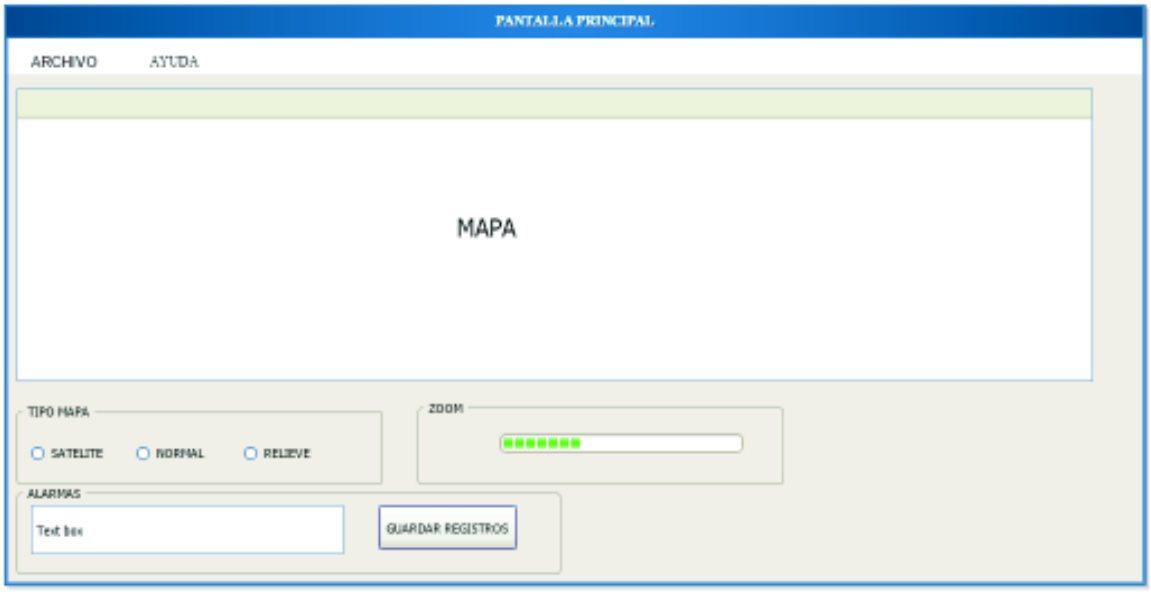


Figura 2.37 Segundo esquema de la pantalla principal

2.5.5 Diseño de los métodos

La aplicación de escritorio utilizará varios métodos que realizarán acciones específicas dentro del programa, esto permitirá interpretar la información recibida y desplegarla en el mapa; los distintos métodos realizados en el diseño se detallan a continuación:

a) Método puertos disponibles

El método permite reconocer que puertos Com se encuentran activos en el computador para posteriormente seleccionarlos; en el caso que el puerto se encuentre en uso emitirá un aviso indicando que se encuentra en uso y se debe seleccionar otro puerto o a su vez se debe cerrar la aplicación que usa el puerto para poder recibir los datos en la aplicación.

b) Método recepción de datos

El método permitirá recibir los datos del puerto serial y los almacenará en una variable, la trama que se recibe del puerto serial se la separa mediante la función *Split* la cual permite diferenciar una sección de información de otra en la carga útil, al momento de programar las motas se establece en la carga útil de la trama el separador “#”, con el cual se diferencia uno de otro dato. Cada mota posee un identificador único, el cual permite diferenciar una de otra mota en la aplicación y así mostrar la información de cada mota de manera separada y ordenada.

c) Método estado

El método estado permitirá obtener el ángulo enviado por el acelerómetro a través del *gateway*; se toma el valor del ángulo y se lo compara, si es mayor a 45° o menor a -45° lo define como caído, caso contrario lo define como estado normal.

d) Método alarma de temperatura

El método alarma de temperatura permitirá establecer el máximo valor de temperatura que soportará la persona u objeto que porte la mota, se establece que la máxima temperatura soportada antes de emitir la alarma será de 30°C, dado que al exceder la temperatura de 43°C el cuerpo humano alcanza un nivel en el que las proteínas se cocinan y los músculos se atrofian, la sangre no llega de manera adecuada al cerebro y este deja de funcionar; una vez que sobrepase el valor de 30°C definido por defecto, emitirá una alarma y a su vez indicará en pantalla los datos relacionados a dicha alarma; la aplicación permite establecer de manera manual cual es el valor máximo con el cual sonará la alarma.

e) Método alarma de estado

El método alarma de estado utilizará los datos del método estado, el cual define si la persona se encuentra en estado “Normal” o “Caído”, una vez que detecta el estado “Caído” emite una alarma y a su vez indica en pantalla los valores relacionados a dicha alarma.

f) Método temporizador

El temporizador permitirá actualizar los métodos de alarma de temperatura y alarma de estado cada intervalo de tiempo definido en el programa cada segundo, los cuales emitirán un sonido cada vez que se disparen las alarmas. Cada segundo el método de alarma revisará los estados y temperaturas de cada mota y emitirá una alarma o no de ser el caso.

g) Método mapa

El método mapa permitirá desplegar en un mapa las diferentes motas conectadas al *gateway*, dentro del mapa dibujará cada mota que se encuentra diferenciada por un identificador único; en este método se definen las propiedades del mapa, entre ellas se permitirá que el mapa se mueva o no con el cursor, además se definirá el *zoom* mínimo y máximo permitido, asimismo se dibujará una marca que identificará a cada mota.

Dentro del mapa se desplegará la información que recibe el *gateway* de cada mota, además se utilizarán los niveles máximos de temperatura y si sobrepasa el nivel indicará dentro del mapa los datos relacionados a dicha alarma, al igual si detecta la posición como “Caído” emitirá una alarma.

2.5.6 Uso del complemento GMap

Es una herramienta de código abierto desarrollada sobre .NET que dispone de enrutamiento y geo código; además, de mapas de Google, Bing, Yahoo!, OpenStreetMap y varias plataformas para desplegar mapas [25]. La idea principal de GMap es simplemente obtener el dato requerido de Google o alguna plataforma que use mapas y usarla de una manera simple.

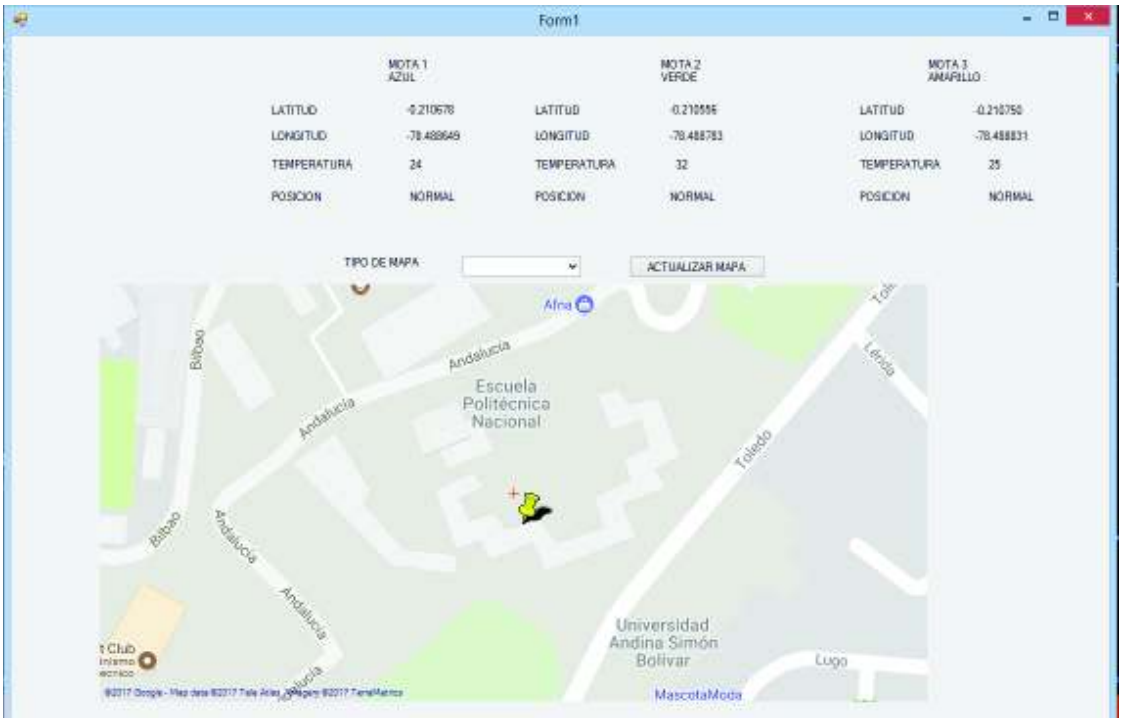


Figura 2.38 Uso de GMap sobre C#

La Figura 2.38 muestra un ejemplo básico desarrollado sobre C# y haciendo uso de GMap, la herramienta despliega un mapa y sobre él un marcador que muestra la ubicación almacenada en la aplicación, se desarrollará una aplicación de escritorio sobre el lenguaje de programación C# haciendo uso de GMap y adaptado a la red de sensores inalámbrica.

2.6 Implementación de la aplicación de escritorio

La implementación de la aplicación de escritorio se realizó mediante el IDE Microsoft Visual Studio 2015 [26], una vez establecida la carga útil que recibe el *gateway* y sus diferentes campos el siguiente paso es el separar e interpretar la información recibida, para esto se desarrollan los diferentes métodos analizados en la parte de diseño.

2.6.1 Pantalla de inicio

Figura 2.39 Pantalla de inicio

La Figura 2.39 muestra la pantalla de inicio desarrollada a partir de C#, basada en los esquemas presentados anteriormente; la pantalla de inicio muestra información sobre puertos disponibles, se debe seleccionar un puerto y si se encuentra disponible permite pasar a la pantalla principal mediante el botón siguiente.

a) Método puertos disponibles

El Código 2.10 permite desplegar los puertos Com activos en el computador mediante un *comboBox*, se lo utiliza para capturar la información del puerto Com y así pasarlo a la aplicación principal mediante la variable “Puertos”.

```
23          //metodo puertosDisponibles
24          //permite desplegar en un combobox los puertos seriales COM que estan en uso en el computador
            1 referencia
25          private void PuertosDisponibles()
26          {
27              foreach (string puertosDisponibles in System.IO.Ports.SerialPort.GetPortNames())
28              {
29                  cboPuertos.Items.Add(puertosDisponibles);
30              }
31          }
```

Código 2.10 Despliega los puertos disponibles

b) Botón salir

El botón salir permite cerrar la ventana y salir de la aplicación como se muestra en el Código 2.11.

```
1 referencia
48      private void btnSalir_Click(object sender, EventArgs e)
49      {
50          //PERMITE CERRAR PANTALLA ACTUAL
51          this.Close();
52
53      }
```

Código 2.11 Botón salir

c) Botón siguiente

Una vez capturado el valor del puerto Com en la variable “Puerto”, mediante el Código 2.12 se envía hacia la pantalla principal y posteriormente se la despliega.

```
1 referencia
35      private void btnSiguiente_Click(object sender, EventArgs e)
36      {
37          PantallaPrincipal princ = new PantallaPrincipal();
38          //envia la informacion seleccionada de combobox hacia la variable puertos en la pantalla principal
39          princ.Puertos = cboPuertos.Text;
40          //despliega pantalla principal
41          princ.Show();
42          if (cboPuertos.Text != "")
43          {
44              this.Hide();
45          }
46      }
```

Código 2.12 Botón siguiente

2.6.2 Pantalla principal

La Figura 2.40 muestra la pantalla principal creada a partir del esquema desarrollado en la parte de diseño; la pantalla principal consta de varios bloques que realizan funciones específicas que son:

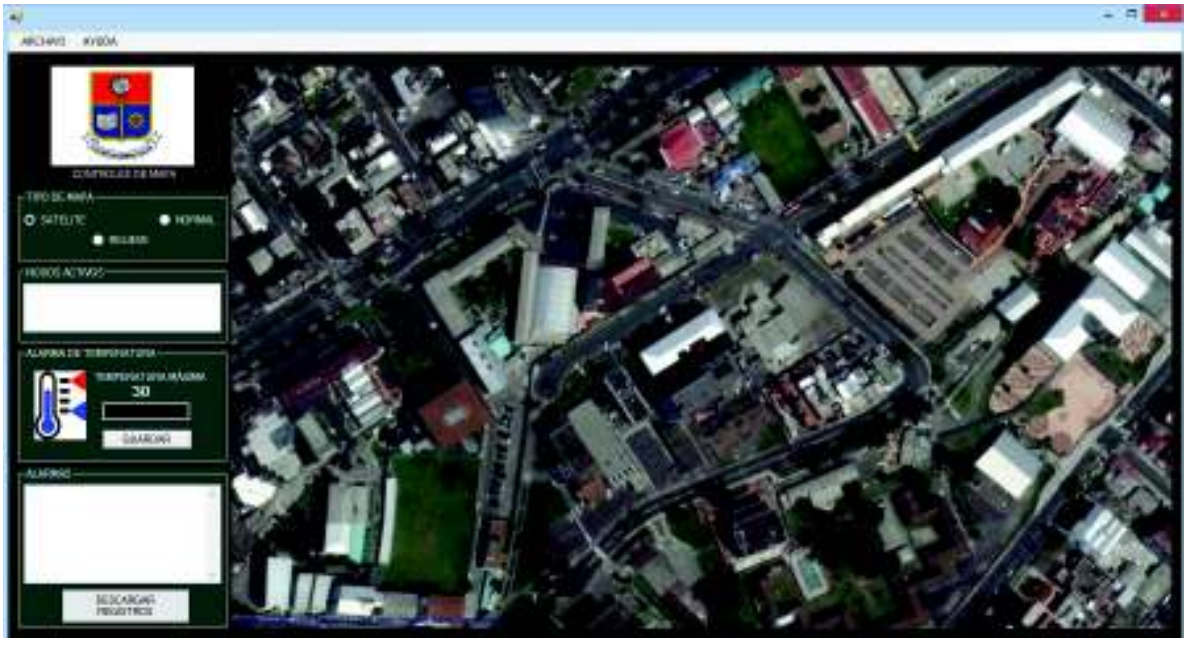


Figura 2.40 Pantalla principal

a) Menú principal

Se lo crea a partir de la herramienta *menuStrip*, la cual permite añadir controles a cada función asignada, se compone de dos pestañas que son “Archivo” y “Ayuda”; dentro de la pestaña de archivo se encuentra la pestaña de “UsuariosActivos” y “Salir”; en la pestaña de ayuda se encuentra “Acerca de” y “Ayuda”.

a.1) Usuarios activos

Figura 2.41 Pantalla de usuarios activos

Al seleccionar en la pestaña de usuarios activos, se abre la ventana visualizar usuarios mostrada en la Figura 2.41.

```
13      5 referencias
        public partial class VisualizarUsuarios : Form
14      {
15          //llama a la listaNodos desde el form principal
16          PantallaPrincipal principal = new PantallaPrincipal();
17          public List<Nodos> listaNodos = new List<Nodos>();
            1 referencia
18          public VisualizarUsuarios()
19          {
20              InitializeComponent();
21          }
22
            1 referencia
23          private void Agregar_usuarios_Load(object sender, EventArgs e)
24          {
25              //muestra los usuarios de la listaNodos sobre datagridview
26              dtgNodos.DataSource = listaNodos;
27          }
28
            1 referencia
29          private void btnSalir_Click(object sender, EventArgs e)
30          {//boton de salir
31              this.Close();
32          }
33
```

Código 2.13 Visualizar usuarios

Dentro de la pantalla de visualizar usuarios se llama a la pantalla principal y se obtiene la lista de nodos que almacena todos los nodos de la red y su información como se muestra en el Código 2.13; posteriormente se despliega la lista de nodos sobre un *dataGridView* el cual mostrará los nodos activos en ese momento.

a.2) Acerca de

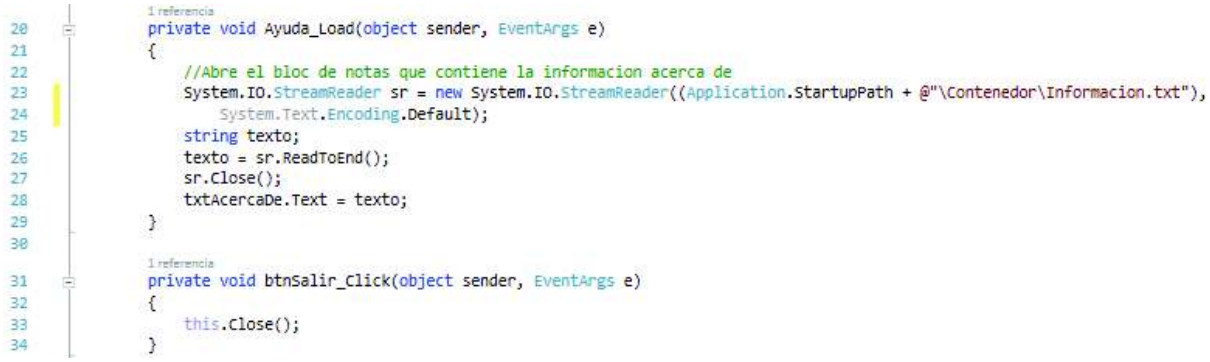
```
private void AcercaDe_Load(object sender, EventArgs e)
{
    //Abre el bloc de notas que contiene la información acerca de
    System.IO.StreamReader sr = new System.IO.StreamReader((Application.StartupPath + @"\contenedor\Acercade.txt"), System.Text.Encoding.Default);
    string texto;
    texto = sr.ReadToEnd();
    sr.Close();
    txtacercade.Text = texto;
}

private void btnsalir_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Código 2.14 Pantalla “Acerca de”

El Código 2.14 permite abrir un bloc de notas llamado “AcercaDe” sobre un *textbox*, dentro del bloc de notas se escribe la información necesaria que muestra información acerca del desarrollador.

a.3) Ayuda



```
1 referencia
private void Ayuda_Load(object sender, EventArgs e)
{
    //Abre el bloc de notas que contiene la informacion acerca de
    System.IO.StreamReader sr = new System.IO.StreamReader((Application.StartupPath + @"\Contenedor\Informacion.txt"),
        System.Text.Encoding.Default);
    string texto;
    texto = sr.ReadToEnd();
    sr.Close();
    txtAcercaDe.Text = texto;
}

1 referencia
private void btnSalir_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Código 2.15 Pantalla de “Ayuda”

El Código 2.15 permite abrir un bloc de notas llamado “Ayuda” sobre un *textbox*, dentro del bloc de notas se escribe la información necesaria para ayudar al manejo de la aplicación.

b) Controles de mapa

Los controles del mapa poseen las siguientes funciones:

b.1) Tipo de mapa

Permite seleccionar mediante un control tipo *radioButton* la opción de cada mapa, en este caso se crean tres opciones para los mapas que son Satélite, Normal y Relieve

```
1 referencia
private void radSatelite_CheckedChanged(object sender, EventArgs e)
{//control de radioboton para desplegar mapa tipo satelite
    //carga el mapa de googlechinasatelite
    gMapControlMapa.MapProvider = GMapProviders.GoogleChinaSatelliteMap;
    gMapControlMapa.Zoom = 18;
}
1 referencia
private void radNormal_CheckedChanged(object sender, EventArgs e)
{//control de radioboton para desplegar mapa tipo normal
    //carga el mapa de googlemap normal
    gMapControlMapa.MapProvider = GMapProviders.GoogleMap;
    gMapControlMapa.Zoom = 18;
}
1 referencia
private void radRelieve_CheckedChanged(object sender, EventArgs e)
{//control de radioboton para desplegar mapa tipo relieve
    //carga el mapa de googleterrainmap
    gMapControlMapa.MapProvider = GMapProviders.GoogleTerrainMap;
    gMapControlMapa.Zoom = 18;
}
```

Código 2.16 Tipo de mapa

Cada control llama a un tipo de mapa de la librería GMap y lo asocia con el proveedor de acuerdo con la característica solicitada; el tipo de mapa satélite llama a "*GoogleChinaSatellite"*, el tipo de mapa normal a "*GoogleMaps"* y el tipo de mapa relieve a "*GoogleTerrainMap"* como se muestra en el Código 2.16; gracias a esto permite el cambio de mapa de acuerdo con la selección escogida; además, permite el control del zoom inicial con el cual el mapa se carga, en este caso se lo define con el valor de 18.

c) Nodos activos

Esta ventana permite desplegar los nodos activos dentro de la red, todos los nodos que se han conectado al *gateway* se desplegarán con su nombre, además de saber si se encuentran activos o inactivos sobre la pantalla.

```
//captura el registro del último nodo que llega al puerto serial
var RegistroUltimoNodo = Nodo.Nodo(Identificador, Nombre, Latitud, Longitud, Acc_z, Temperatura);
//añade a la lista de nodos los datos separandolos mediante el id
if (ListaNodos.Exists(x => x.Id == RegistroUltimoNodo.Id))
{
    //se crea el indice de la lista separandolo con el id
    int indice = ListaNodos.FindIndex(x => x.Id == RegistroUltimoNodo.Id);
    ListaNodos[indice].Latitud = RegistroUltimoNodo.Latitud;
    ListaNodos[indice].Longitud = RegistroUltimoNodo.Longitud;
    ListaNodos[indice].Acc_z = RegistroUltimoNodo.Acc_z;
    ListaNodos[indice].Temperatura = RegistroUltimoNodo.Temperatura;
}
else
//si no existe el id se añade a la lista
ListaNodos.Add(RegistroUltimoNodo);
LstNodosActivos.Items.Clear();
//recorre la lista de nodos y detecta si la condición es activa o inactiva
foreach (var Nodo in ListaNodos)
{
    //añade la condicion de activo si el nodo es visible
    if (Nodo.Visible==true)
     LstNodosActivos.Items.Add("Nombre: " + Nodo.Nombre + "  Condicion:  " + "Activo");
    else
        LstNodosActivos.Items.Add("Nombre: " + Nodo.Nombre + "  Condicion:  " + "Inactivo");
    //añade la condicion de inactivo si el nodo no es visible
}
//llama al método de mapa()
MarcasMapa();
```

Código 2.17 Nodos activos

El Código 2.17 muestra cómo obtener los datos de la lista de nodos, posteriormente se toma el registro del último nodo que registro el *gateway* y se crea un índice, con lo cual se obtiene la lista basada en el último nodo y registrando todos los nodos activos; además, se los diferencia mediante el Id; finalmente, se muestra sobre un *listBox* el nombre de los nodos y además permite activar o desactivar la opción de visible o no en el mapa.

```
1 referencia
388         private void lstNodosActivos_DoubleClick(object sender, EventArgs e)
389         {//al hacer doble clic sobre el elemento de la lista, lo vuelve visible o no
390             int posicion = LstNodosActivos.SelectedIndex;
391             if (ListaNodos[posicion].Visible == true)
392                 ListaNodos[posicion].Visible = false;
393             else
394                 ListaNodos[posicion].Visible = true;
395         }
```

Código 2.18 Nodos activos

El Código 2.18 muestra el evento doble clic del *listbox* usado, con lo cual al indicar los nodos activos se puede realizar con el cursor un doble clic y así activar o desactivar al nodo visible en el mapa.

d) Alarmas

La propiedad de alarmas posee las funciones de registro y descarga de registros, detalladas a continuación:

```
290         private void AlarmaTemperatura()
291         {//método para desplegar la alarma de temperatura
292          //compara el límite de temperatura con la temperatura recibida
293
294             if (Convert.ToDouble(Temperatura) >= LimiteTemperatura)
295             {
296                 var UltimoNodo = ListaNodos.Last();
297                 //en el caso de alarma de temperatura, carga el sonido en la aplicacion y emite sonido de alerta
298                 SoundPlayer SonidoTemperatura = new SoundPlayer(Application.StartupPath + @"\Contenedor\temp.wav");
299                 SonidoTemperatura.Play();
300                 //escribe los datos relacionados a la alarma sobre un textbox
301                 txtAlarma.Text += Fecha + "  " + UltimoNodo.Nombre + "  " + Estado + " TEMPERATURA:" + UltimoNodo.Temperatura +
302                     " LATITUD:  " + UltimoNodo.Latitud + " LONGITUD:" + UltimoNodo.Longitud+ " ";
303             }
304             else
305             {           }
306         }
            3 referencias
307         private void AlarmaEstado()
308         {
309             if (Estado == "CAIDO")
310             {
311                 var UltimoNodo = ListaNodos.Last();
312                 //en el caso de detectar al nodo sensor caido emite alarma
313                 SoundPlayer SonidoEstado = new SoundPlayer(Application.StartupPath + @"\Contenedor\caida.wav");
314                 SonidoEstado.Play();
315                 //escribe datos de la alarma sobre un textbox
316                 txtAlarma.Text += Fecha + "  " + UltimoNodo.Nombre + "  " + Estado + " TEMPERATURA:" + UltimoNodo.Temperatura +
317                     " LATITUD:  " + UltimoNodo.Latitud + " LONGITUD:" + UltimoNodo.Longitud + " ";
318              }
319             else
320             { }
321          }
```

Código 2.19 Alarma de estado y temperatura

El Código 2.19 muestra las condiciones que se llevan a cabo para emitir un sonido de alarma, en el caso de alarma de temperatura compara el valor del límite de temperatura con la temperatura recibida de la mota, si es mayor emite un sonido y escribe sobre el textbox los valores, caso contrario no realiza ninguna acción; por otro lado, la alarma de estado compara si se encuentra caído o normal, en el caso de caído emite un sonido y escribe los valores sobre un *textbox*, caso contrario en estado normal no realiza ninguna acción.

d.1) Límite de temperatura

```
1 referencia
private void btnAlarmaTemperatura_Click(object sender, EventArgs e)
{//permite definir el límite de temperatura, solo permite un numero entero
    try
    {
            for (int i = 0; i < 20; i++)
            {
            //envía el valor de temperatura a traves del puerto serial hacia el gateway y las motas
            SerialPortGateway.Write(txtLímiteTemperatura.Text);
            }
            LimiteTemperatura = Convert.ToInt16(txtLímiteTemperatura.Text);
            //muestra el límite de tempertura definido
            lblTemperatura.Text = Convert.ToString (LimiteTemperatura);
            //limpia el textbox
            txtLímiteTemperatura.Text = "";
    }
    catch (Exception ex)
    {
        //captura excepción si el numero escrito no es un entero
        MessageBox.Show("**USTED DEBE ESCRIBIR LA TEMPERATURA EN GRADOS CENTIGRADOS \n **EXPRESADA CON UN NUMERO ENTERO", "ERROR",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Código 2.20 Definir límite de temperatura

El Código 2.20 muestra cómo se define el límite de temperatura máximo antes de sonar la alarma; mediante un botón se captura el valor introducido sobre un textbox y se lo graba en la variable “LímiteTemperatura”; además, permite enviar el nuevo valor al *gateway* definido en reenviar 20 veces dicho valor para lograr llegar a todos los nodos de manera correcta, el *gateway* posteriormente lo enviará a todas las motas dentro de la red; la condición establecida es que el valor de temperatura sea un numero entero.

d.2) Registros

El Código 2.21 muestra la pantalla de registros la cual se utiliza para desplegar información del usuario, en el caso de provocar una alarma ya sea por temperatura o posición, escribe en un *textbox* la fecha, hora, temperatura, nombre, latitud, longitud y posición; la fecha y hora se la captura del mismo computador con la función *DateTime.Now*.

```
//escribe los datos relacionados a la alarma sobre un textbox
txtAlarma.Text += Fecha + "  " + UltimoNodo.Nombre + "  " + Estado + " TEMPERATURA:" + UltimoNodo.Temperatura +
    " LATITUD:  " + UltimoNodo.Latitud + " LONGITUD:" + UltimoNodo.Longitud+ " ";
```

Código 2.21 Ventana de registro de alarma

d.3) Descarga de registros

Mediante el botón de descargar registros mostrado en el Código 2.22 se guarda en un archivo de texto los registros de las alarmas desplegadas por estado caído o por sobrepasar el límite de temperatura; los datos generados de las alarmas se escriben sobre un *textbox*, al presionar el botón de guardar recoge dichos datos y los almacena en un archivo de texto.

```
1 referencia
private void btnGuardar_Click(object sender, EventArgs e)
{//boton para guardar las alarmas sobre un archivo de texto
    //permite guardar los datos referentes a las alarmas sobre un bloc de notas
    try
    {
       SaveFileDialog ArchivoAlarma = new SaveFileDialog();
        ArchivoAlarma.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
        ArchivoAlarma.FilterIndex = 2;
        ArchivoAlarma.RestoreDirectory = true;
        //permite seleccionar una carpeta para guardar el archivo y su nombre
        if (ArchivoAlarma.ShowDialog() == DialogResult.OK)
        {
            if (ArchivoAlarma.ShowDialog() == DialogResult.OK)
            {
                using (StreamWriter TextoAlarma = new StreamWriter(ArchivoAlarma.OpenFile()))
                {
                    TextoAlarma.Write(txtAlarma.Text);
                }
            }
        }
        //si se graba el archivo sin errores envia mensaje de guardado correctamente
        MessageBox.Show("Documento guardado correctamente");
    }
    catch (Exception ex)
    {
        //captura excepcion si no se guarda correctamente
        System.Console.WriteLine("{0} Excepcion.", ex.GetType().ToString());
        System.Console.WriteLine(ex.Message);
        MessageBox.Show("Documento no se guardo correctamente");
    }
```

Código 2.22 Guarda información de alarmas en un archivo de texto

e) Mapa

Se usa el control GMap para desplegar un mapa de acuerdo con la característica seleccionada en el tipo de mapa y sobre él mostrar los diferentes usuarios que se encuentran conectados al *gateway*; el control GMap cuenta con funciones que permiten manipular el mapa, la posición inicial, el *zoom*, etc.

```
        //permite mover el mapa con el cursor
        gMapControlMapa.DragButton = MouseButtons.Left;
        gMapControlMapa.CanDragMap = true;
        //Muestra la posición inicial centrando el mapa en dichas coordenadas
        gMapControlMapa.Position = new PointLatLng(-0.209243, -78.489356);
        //controles del mapa, zoom y scroll
        gMapControlMapa.MinZoom = 3;
        gMapControlMapa.MaxZoom = 23;
        gMapControlMapa.Zoom = 18;
        gMapControlMapa.AutoScroll = true;
```

Código 2.23 Despliegue del mapa

El Código 2.23 permite definir las características del mapa, entre ellas el mover el mapa con el botón del cursor, la posición inicial del mapa, en este caso se plantea una posición inicial dado que al no incluirla el mapa se inicia en la posición 0.000000 y muestra una imagen azul, para evitar esto se inicializa con coordenadas de la ciudad de Quito; además permite definir el *zoom* mínimo y máximo del mapa y la manipulación mediante la rueda del cursor.

```
1 referencia
private void MarcasMapa()
{//método marcas_sobre_mapa, dibuja las marcas en el mapa
    //instancia marcas
    GMapOverlay Marcas = new GMapOverlay("markers");
    //limpia las marcas
    gMapControlMapa.Overlays.Clear();
    int index = 1;
    //usa el ultimo nodo que entra en la lista_nodos
    var UltimoNodo = ListaNodos.Last();
    //recorre la lista de nodos
    foreach (var Nodo in ListaNodos)
    {
        EstadoMota(Convert.ToInt16(Nodo.Acc_z));
        AlarmaTemperatura();
        AlarmaEstado();
        //usa la fecha y hora del computador
        Fecha = DateTime.Now;
        //crea una marca de google
        GMarkerGoogle Marca = new GMarkerGoogle(new PointLatLng(Convert.ToDouble(Nodo.Latitud),
            Convert.ToDouble(Nodo.Longitud)), GMarkerGoogleType.red);
```

Código 2.24 Marcas en el mapa

Para el manejo de marcas que reflejan la posición del usuario sobre el mapa se usa la función "*GmapOverlay"* como se muestra en el Código 2.24, la cual utiliza marcas de Google y mediante información de latitud y longitud obtenidas de las motas dibuja un objeto en el lugar exacto donde se encuentra la mota. Para evitar que las marcas se superpongan se utiliza la función "Clear", con lo cual limpia las marcas antes de agregar una nueva, las marcas se generan a partir de la lista de nodos que contiene a todos los nodos activos con su información.

```
//recorre el indice y crea una marca con colores distintos, azul, verde y amarillo,
//en el caso de mas de 3 motas, reusa los colores
switch (index)
{
   case 1:
         Marca = new GMarkerGoogle(new PointLatLng(Convert.ToDouble(Nodo.Latitud),
             Convert.ToDouble(Nodo.Longitud)), GMarkerGoogleType.blue);
        index++;
         break;
     case 2:
        Marca = new GMarkerGoogle(new PointLatLng(Convert.ToDouble(Nodo.Latitud),
            Convert.ToDouble(Nodo.Longitud)), GMarkerGoogleType.green);
         index++;
         break;
     case 3:
          Marca = new GMarkerGoogle(new PointLatLng(Convert.ToDouble(Nodo.Latitud),
              Convert.ToDouble(Nodo.Longitud)), GMarkerGoogleType.yellow);
         index=1;
         break;
     default:
         break;
 }
 //muestra al nodo como visible
 Marca.IsVisible = Nodo.Visible;
```

Código 2.25 Dibuja marca de cada usuario en el mapa

El Código 2.25 muestra el uso de los casos para asignar un color a cada mota, los colores asignados son azul, verde y amarillo; estos colores se los asigna en orden de llegada al *gateway*; además, si existen más de 3 motas conectadas se reúsan los colores. El código de la aplicación de escritorio completo se muestra en el Anexo H del documento.

3 RESULTADOS Y DISCUSIÓN

En la sección de resultados y discusión se presentan los resultados más relevantes obtenidos en la implementación del prototipo de monitoreo de posición y temperatura basado en redes de sensores inalámbricas IEEE 802.15.4, todo esto demostrando la funcionalidad del prototipo y las características que este dispone.

3.1 Prueba del envío de datos

Se realizaron varias pruebas del envío de datos entre el *gateway* y las motas; cada prueba realizada se detalla con los resultados obtenidos. Para probar el funcionamiento de los diferentes sensores y la creación de la trama en cada mota, se envía la información de la trama mediante el puerto USB para ser leído e interpretado antes de ser enviada mediante radiofrecuencia.

3.1.1 Prueba del GPS

A manera de prueba se conecta un cable USB a la mota para así determinar cual información genera antes de transmitirla al *gateway*; mediante la herramienta "Putty" se recibe la información en el puerto Com al cual se le conecta la mota, dicha información se detalla a continuación:

La mota recibe información del sensor GPS conectado, este sensor trabaja de manera independiente de la mota, cada vez que el lazo se repita lo activara para que pueda conectarse a los satélites y obtener latitud y longitud, para esto dispone de 20 segundos; transcurrido este tiempo y al no conectarse enviará el valor 0.000000.

3.1.2 Prueba de envío de información de los sensores a cada mota

Se realizan pruebas de la creación de la trama en cada mota, la trama debe incluir la información de identificador, nombre, número de trama, latitud, longitud, ángulo y temperatura; para comprobar dicha información se conecta cada mota al computador y se lee mediante el puerto USB el valor que genera cada una.

a) Generación de tramas

```
==============================
Current ASCII Frame:
Length: 70
Frame Type:  128
frame (HEX): 3C3D3E8003233430333337343130302 34E6F646F4361726F6C2330234750533A302E3030
303030303B302E303030303030235354523A34235443413A32352E393420
frame (STR): <=>#403473440#NodoFabian#30#GPS:-0.211821;-78.488726#STR:90#TCA:20.45#
==============================
enviado correcto
Error receiving a packet:1
```

Figura 3.1 Trama generada en la mota NodoFabián

La Figura 3.1 muestra la trama que genera la mota NodoFabian, en la cual se observa que se encuentra conectada al módulo de radiofrecuencia, posteriormente tiene una longitud de 70 caracteres; además, el tipo de trama 128 que quiere decir que es una trama de datos, luego muestra la trama en formato hexadecimal y finalmente la muestra como caracteres; todos estos valores leídos en el puerto USB los escribe por defecto la mota y no es necesario programarlo.

```
==============================
Current ASCII Frame:
Length: 71
Frame Type:  128
frame (HEX): 3C3D3E800323343033343735373736234E6F646F50616D656C612330234750533A302E
3030303030303B302E3030303030302353545 23A3930235443413A33312E313633
frame (STR): <=>#403475776#NodoPamela#41#GPS:-0.211581;-78.488727#STR:0#TCA:21.56#
==============================
enviado correcto
Error receiving a packet:1
```

Figura 3.2 Trama generada en la mota NodoPamela

La Figura 3.2 muestra la trama que genera la mota NodoPamela, en la cual se observa que se encuentra conectada al módulo de radiofrecuencia, posteriormente tiene una longitud de 71 caracteres; además, el tipo de trama 128 que implica una trama de datos, luego muestra la trama en formato hexadecimal y finalmente la muestra como caracteres; todos estos valores leídos en el puerto USB los escribe por defecto la mota y no es necesario programarlo.

```
==============================
Current ASCII Frame:
Length: 69
Frame Type:  128
frame (HEX): 3C3D3E800323343033333734313030234E6F646F4361726F6C2330234750533A302E303
03030303030B302E3030303030302353545 23A34235443413A32352E393423
frame (STR): <=>#403374100#NodoCarol#49#GPS:-0.211751;-78.489046#STR:78#TCA:23.48#
==============================
enviado correcto
Error receiving a packet:1
```

Figura 3.3 Trama generada en la mota NodoCarol

La Figura 3.3 muestra la trama que genera la mota NodoCarol, en la cual se observa que se encuentra conectada al módulo de radiofrecuencia, posteriormente tiene una longitud de 69 caracteres; además, el tipo de trama 128 que quiere decir que es una trama de datos, luego muestra la trama en formato hexadecimal y finalmente la muestra como caracteres; todos estos valores leídos en el puerto USB los escribe por defecto la mota y no es necesario programarlo. Las tramas creadas cuentan con un carácter de inicio y fin de trama, un identificador de trama, un nombre de la mota, un número de trama, GPS (latitud, longitud), acelerómetro (ángulo z) y finalmente temperatura(C°).; en la parte final se

muestra un error al recibir el paquete, esto se genera dado que no se ha enviado ninguna información desde el *gateway* a la mota.

3.1.3 Prueba del envío de datos entre las motas y el gateway

Para probar el correcto envío de datos entre las motas y el *gateway*, se utiliza la aplicación "putty"; se realiza una prueba simultánea entre la información que envía la mota y la que recibe el *gateway*.

Figura 3.4 Comparación de los datos del gateway y la mota

La Figura 3.4 muestra en el Com 5 la trama que genera la mota, mientras que en el Com 7 los datos que recibe el *gateway*; se compara la trama STR en la mota con la trama que recibe el *gateway* y posteriormente se aprecia que las dos tramas son iguales, con esto se prueba que el envío de datos mota *gateway* se realiza sin errores y no ha generado problemas dado que la configuración en los dispositivos se realizó de manera adecuada.

3.2 Interpretación de información

El *gateway* recibe los datos de múltiples motas que se encuentren dentro de la misma red, en este caso se simula el prototipo con 3 motas, las cuales arrojan los siguientes datos.

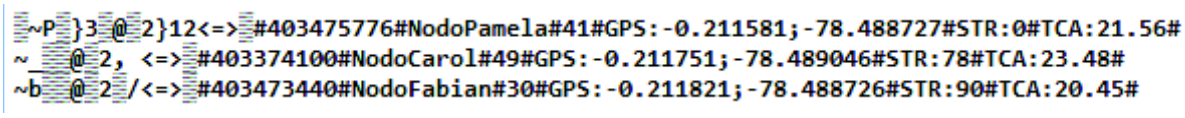


Figura 3.5 Datos recibidos por el gateway

La Figura 3.5 muestra un fragmento de los datos que recibe el *gateway*, dado que es una comunicación serial el *gateway* recibe de cada mota la información y la envía mediante el puerto USB de manera serial en el orden que recibe la información para que, posteriormente, la aplicación interprete dicha información y la despliegue en el mapa; cada mota posee un identificador único y un nombre único que se lo asigna en la programación,

esto para diferenciarla una de otra; a continuación, se detalla la interpretación de la trama recibida, desglosando cada campo y separando cada usuario.

3.2.1 Interpretación de las tramas

Tabla 3.1 Trama NodoFabián

Trama NodoFabian		
Inicio de trama		~b▒▒@ 2 /<=>▒
Identificador		403473440
Nombre		NodoFabian
Número de trama		30
Datos de GPS	Latitud	-0.211821
	Longitud	-78.488726
Datos del acelerómetro	Ángulo z	90
Temperatura		20.45
Fin de trama		▒

Tabla 3.2 Trama NodoPamela

Trama NodoPamela		
Inicio de trama		▒~P }3 @ 2}12<=>▒
Identificador		403475776
Nombre		NodoPamela
Número de trama		41
Datos de GPS	Latitud	-0.211581
	Longitud	-78.488727
Datos del acelerómetro	Ángulo z	0
Temperatura		21.56
Fin de trama		▒

Tabla 3.3 Trama NodoCarol

Trama nodo 3 Carol		
Inicio de trama		~_▒▒@ 2, <=>▒
Identificador		403374100
Nombre		NodoCarol
Número de trama		49
Datos de GPS	Latitud	-0.211751
	Longitud	-78.489046
Datos del acelerómetro	Ángulo z	78
Temperatura		23.48
Fin de trama		▒

La Tabla 3.1, Tabla 3.2 y Tabla 3.3 indican los datos separados de la trama recibida del NodoFabián, NodoPamela y NodoCarol, cada trama envía un carácter de inicio y fin de trama que ayudan a identificar cuando empieza y termina la trama, una vez separada la trama se la asocia con el identificador y nombre del nodo para posteriormente separar la información de temperatura, posición y estado.

3.3 Prueba de las funcionalidades de la aplicación

La aplicación de escritorio cuenta con varias funcionalidades entre ellas la lectura de la temperatura del ambiente donde se encuentra la mota, el estado de la persona que porta la mota (caído o normal), la ubicación de la persona mediante un GPS (latitud y longitud), alarmas en el caso de caído o sobrepasar el nivel máximo de temperatura establecido en 30°C pero configurable por el usuario; todas estas funciones se despliegan en un mapa para una fácil interpretación de la información de cada mota. El propósito de la aplicación de escritorio es que la información sea leída e interpretada de manera fácil y ordenada por el usuario, por esta razón se la realiza en su mayoría con un despliegue de la información sencillo y ambiente gráfico.

3.3.1 Pantalla de bienvenida

Figura 3.6 Pantalla de bienvenida de la aplicación

Al iniciar la aplicación se muestra la pantalla de presentación de la Figura 3.6, la cual indica los datos referentes al prototipo a manera de presentación; esta pantalla cuenta con un botón salir que permite cerrar la aplicación; además, de un botón siguiente permite cerrar esta pantalla y desplegar la siguiente pantalla que corresponde a la inicial.

3.3.2 Pantalla de inicio

Figura 3.7 Pantalla de inicio de la aplicación

La Figura 3.7 muestra la pantalla inicial de la aplicación, al correr el programa se despliega la pantalla de presentación y posteriormente la pantalla de inicio. En la parte central se encuentra la opción de seleccionar un puerto, mediante un *comboBox* se indican los puertos que el computador reconoce como activos y los despliega como Com seguido de un número; en esta opción se debe seleccionar el puerto donde se conectó el *gateway*, si el puerto está en uso por otra aplicación se despliega un aviso indicando que se encuentra en uso y se escoja otro o se deje de usar dicho puerto.

En la parte inferior izquierda se encuentra el botón salir que permite cerrar el programa, en la parte inferior derecha se encuentra el botón siguiente que permite desplegar la pantalla principal y se cierra la pantalla inicial, la aplicación pide que como requisito se seleccione un puerto Com para poder avanzar a la pantalla principal.

3.3.3 Pantalla principal

Una vez comprobada la correcta comunicación de las distintas motas y el *gateway*, asimismo, el correcto funcionamiento de los sensores que componen la mota se procede a comprobar la comunicación entre el *gateway* y la aplicación. El *gateway* se comunica con la aplicación mediante el puerto Com asociado a dicha conexión, para esto se programa en la aplicación que se pueda leer el puerto y recibir los datos de manera serial.

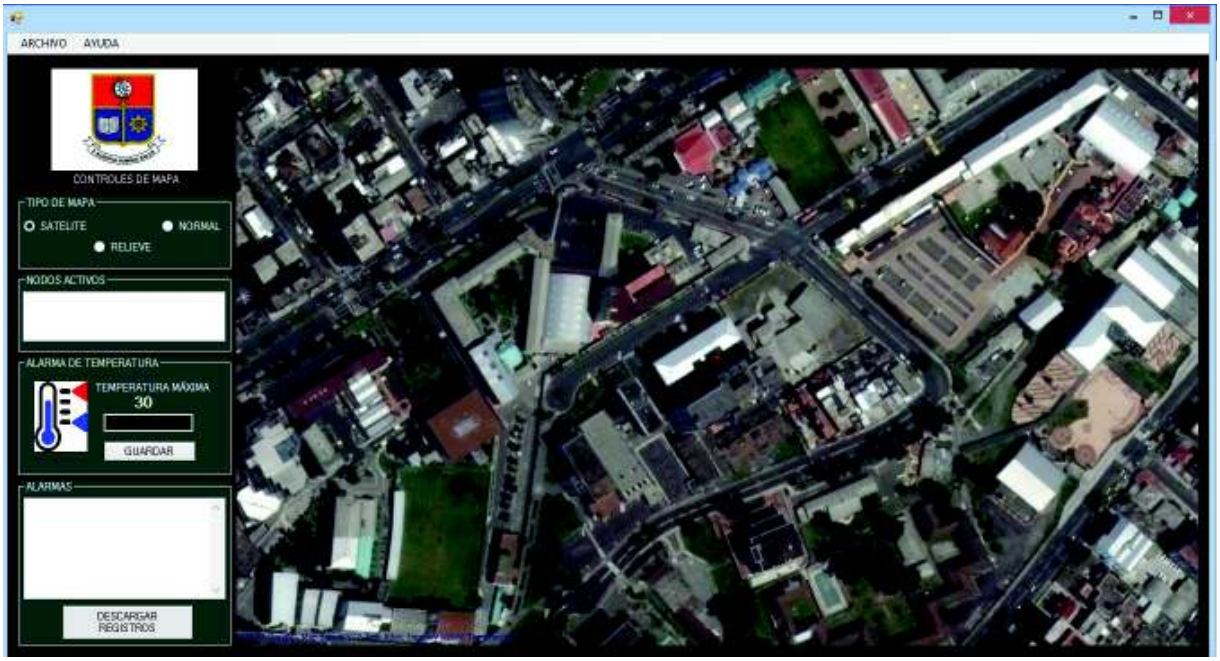


Figura 3.8 Pantalla principal de la aplicación

En la Figura 3.8 se muestra la pantalla principal, la cual consta de un *menuStrip* en la parte superior con el cual se puede seleccionar la función de archivo y posteriormente la opción usuarios activos y salir; el botón de ayuda permite seleccionar "Acerca de" con lo cual indica información sobre el desarrollo de la aplicación y la opción de ayuda. La pestaña "Acerca de" permite desplegar la siguiente pantalla mostrada en la Figura 3.9, la cual muestra información sobre el desarrollador del prototipo.

Figura 3.9 Pantalla "Acerca de"

La pestaña ayuda permite desplegar la siguiente pantalla mostrada en la Figura 3.10, la cual muestra información sobre el manejo de la aplicación.

Figura 3.10 Pantalla de ayuda

a) Nodos activos

En la pantalla principal, sobre el menú es posible seleccionar la función de visualizar usuarios que permite desplegar una ventana sobre la cual se muestran los usuarios dentro de la red como se muestra en la Figura 3.11; además, en la característica de visible se puede añadir o quitar el visto, permitiendo quitar del mapa a dicha mota seleccionada.

Figura 3.11 Pantalla de visualizar nodos

La Figura 3.12 muestra la ventana de nodos activos, la cual se encuentra en el menú principal; sobre ella se pueden observar todos los nodos que se encuentran activos en la red, permitiendo al usuario conocer de una manera más rápida los usuarios activos; además, con la función del doble clic del cursor permite activar o desactivar dichas motas en el mapa.

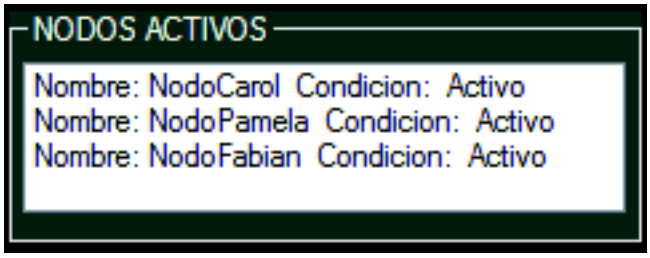


Figura 3.12 Pantalla de nodos activos

b) Tipo de mapa

La Figura 3.13 muestra cómo se puede seleccionar el tipo de mapa mediante el control *radioButton*; en el cuadro de tipo de mapa se puede escoger el tipo de mapa que se requiera entre ellos “Satélite”, “Normal” y “Relieve”.

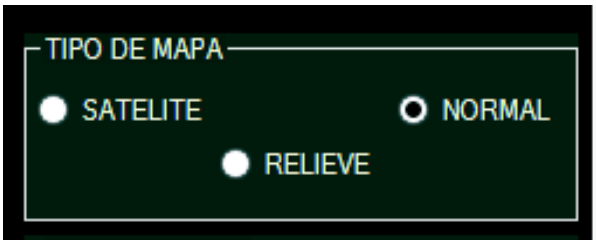


Figura 3.13 Tipo de mapa

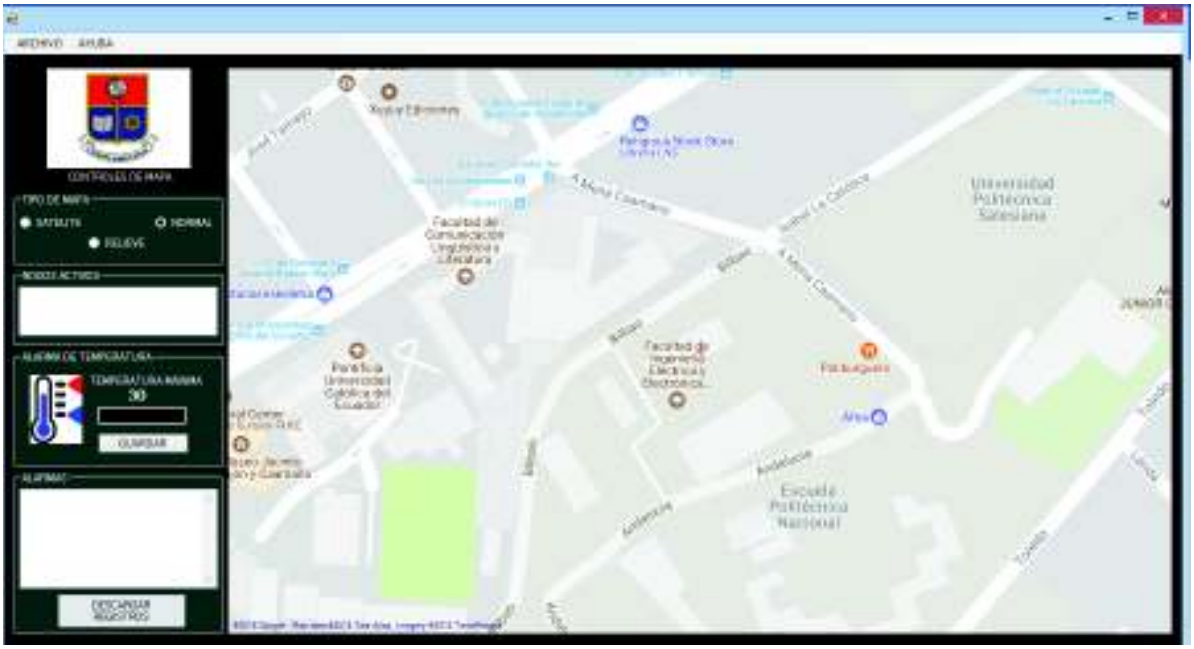


Figura 3.14 Mapa tipo normal

La Figura 3.14 muestra el tipo de mapa normal que utiliza “*GoogleMap”*, que muestra una vista general de las calles y sus nombres.

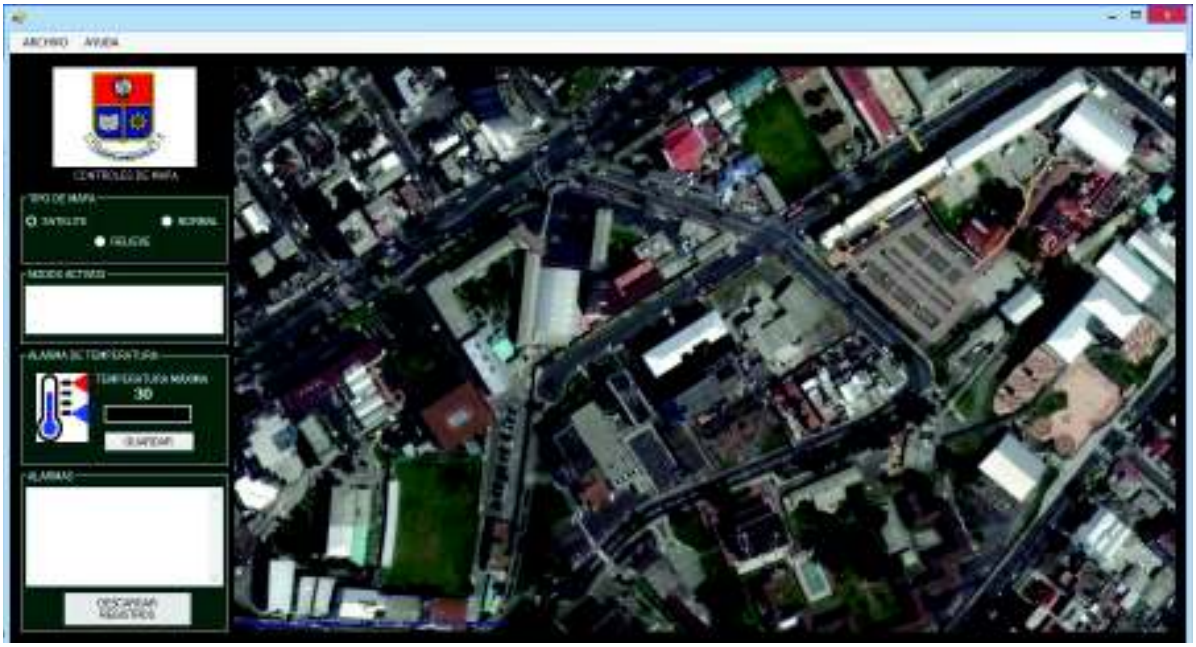


Figura 3.15 Mapa tipo satélite

La Figura 3.15 muestra el tipo de mapa satélite que utiliza "*GoogleChinaSatelliteMap"*, muestra una vista general con imágenes reales de la superficie donde se ubica.

Figura 3.16 Mapa tipo relieve

La Figura 3.16 muestra el tipo de mapa relieve que utiliza "*GoogleTerrainMap"*, muestra una vista general de calles y nombres además de líneas de nivel que denotan altitud.

c) Zoom

El mapa permite manipular el *zoom*, mediante el scroll del cursor permite manipular el acercamiento o alejamiento del mapa dependiendo de la necesidad del usuario.

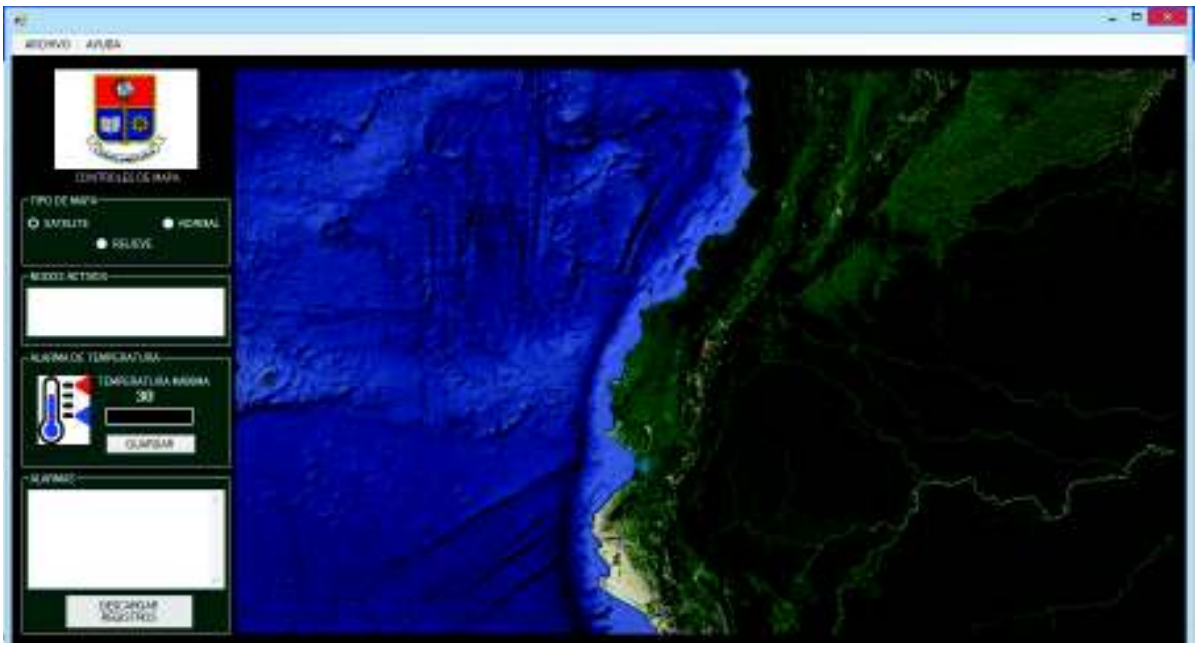


Figura 3.17 Pantalla *zoom out*, alejar mapa

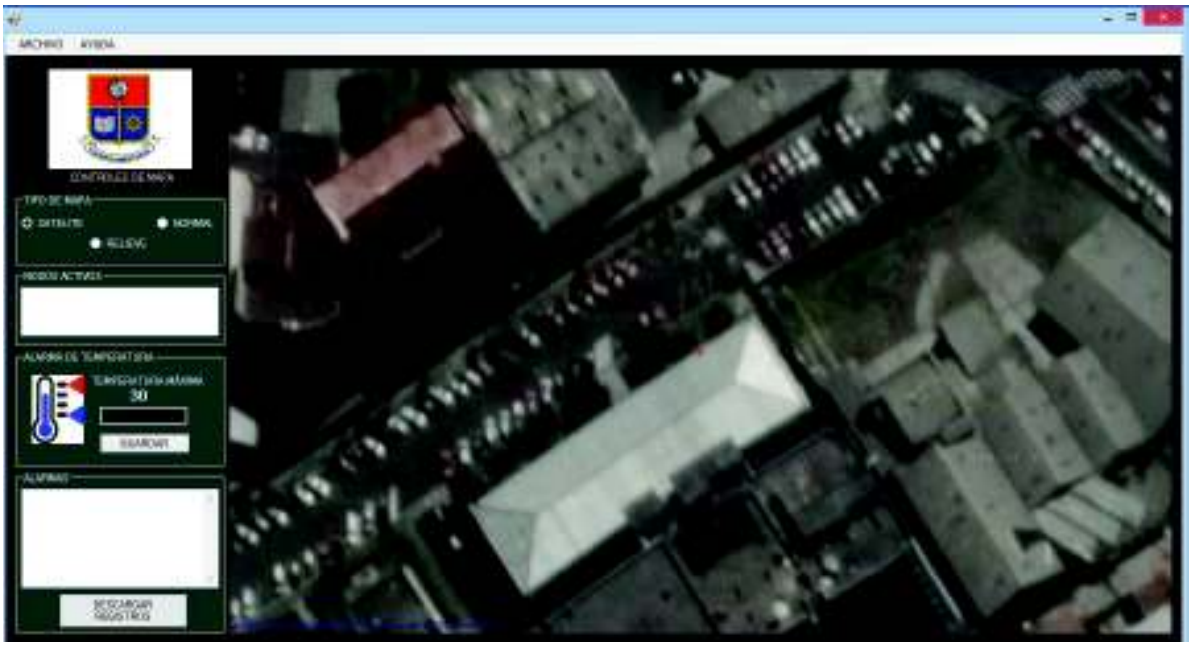


Figura 3.18 Pantalla *zoom in*, acercar mapa

La Figura 3.17 y Figura 3.18 muestran la comparación de un mapa con acercamiento, frente a uno con alejamiento (*zoom in, zoom out*).

3.3.4 Recepción de temperatura

Figura 3.19 Recepción de temperatura en la aplicación

La Figura 3.19 muestra sobre el mapa la marca del nodo sensor asociado a la red; además, sobre la marca se puede apreciar el valor de temperatura a la cual está expuesto, en este caso es de 18.81°C.

Figura 3.20 Definir límite de temperatura

Adicionalmente en la Figura 3.20 se observa un *textbox* sobre el cual se puede definir el nuevo límite máximo de temperatura, en este caso solo acepta números enteros; una vez introducido el nuevo valor se debe pulsar el botón guardar, el nuevo valor aparecerá en la etiqueta y de ser el caso que la temperatura sea mayor a la definida, sonará una alarma de alerta.

3.3.5 Recepción de los datos del acelerómetro y estado de la mota

El acelerómetro interno de cada mota envía la información de coordenadas x, y, z, a continuación, dentro de la mota se usa el valor del eje z y se lo convierte en un ángulo que posteriormente se envía a la aplicación; la aplicación interpreta este ángulo e indica si se

encuentra caído o normal la persona que porta la mota, esta información se despliega en la aplicación de la siguiente manera.

Figura 3.21 Estado de mota caído

La Figura 3.21 muestra cuando una mota se encuentra en estado caído, lo cual produce una alarma de posición; sobre el mapa se muestra su información y en el *textbox* de alarmas se escriben los datos relacionados a dicha alarma.

Figura 3.22 Estado de mota normal

La Figura 3.22 muestra el marcador de la mota en estado normal, solo se visualiza en el mapa su marcador de un color y no despliega más información, de ser requerida por el usuario debe ubicar el cursor sobre la marca para obtener la información. Este mecanismo se implementó dado que al existir varias motas sobre el mapa la información de cada una se superponía y no se visualizaba correctamente.

3.3.6 Recepción de GPS y ubicación

Una vez establecida la conexión al satélite, mediante el GPS que posee cada mota se reciben los datos de latitud y longitud; a través de la función GMap se lee, interpreta y despliega en un mapa la posición de cada mota. En el caso que la aplicación no encuentre ninguna mota mostrará un mapa centrado en la ciudad de Quito ya que se lo inicializa en ese lugar; en el momento en el que se descubren motas indica la posición cuando reciba la información de cada mota y actualiza el mapa de acuerdo con la información recibida.

En el caso del prototipo se simula con 3 motas y un *gateway*, cada mota envía su información cada 3 segundos al *gateway*; cada una independiente de otra. El *gateway* recibe la información de todas las motas dentro de su red y la envía a la aplicación mediante el puerto serial, en ese momento se visualiza en el mapa mediante un marcador y se la diferencia con un color distinto a cada mota, el color del marcador de cada mota se asigna de acuerdo con el orden de llegada de los datos.

Figura 3.23 Recepción del GPS sobre el mapa

La Figura 3.23 muestra las motas desplegadas en el mapa, cada mota se diferencia con un marcador de color distinto, nombre, temperatura, estado, latitud y longitud; esta información es visible al ubicar el cursor sobre el marcador.

3.3.7 Función de alarmas

Existen dos eventos de alarmas dentro de la aplicación, uno si sobrepasa el nivel máximo de temperatura establecido y otro si cae la persona u objeto que porta el sensor, lo cual indicaría el estado de caído. Siempre que una alarma se active la información asociada a cada mota se desplegará de manera automática al igual que se visualizará sobre el *textBox* de alarmas.

a) Alarma de estado caído

El evento de “alarma de estado caído” actúa al momento de detectar que la persona que porta la mota se encuentra en estado caído, automáticamente emite un sonido de alarma en la aplicación y en la mota indicando que se encuentra caído; además, los datos de esta persona se despliegan sobre el mapa, como se indica en la Figura 3.24. Sobre la marca de la persona en estado caído se puede evidenciar su información como nombre, temperatura, estado, latitud y longitud.

Figura 3.24 Alarma de estado caído

b) Alarma de temperatura

El evento de “alarma de temperatura” actúa al momento de detectar que sobrepaso el límite máximo establecido (inicialmente es de 30°C) o configurado de temperatura (en la Figura 3.25 se muestra el límite configurado de 15°C), al momento de sobrepasar el límite emite una alarma en la aplicación y en la mota, que permite alertar a la persona que lo monitorea.

Figura 3.25 Alarma de temperatura

3.3.8 Envío de registros de alarmas

Al momento de generar un evento de alarma, ya sea por estado caído o por sobrepasar la temperatura máxima establecida; la información asociada a cada mota que produce una alarma se registra en un *textbox* de alarma dentro de la aplicación, como se muestra en la Figura 3.26 la alarma se produjo por el estado caído de la mota.

Figura 3.26 Registro de alarmas

Adicionalmente el botón guardar permite crear un documento de texto en la ubicación deseada (ubicación del archivo dentro del computador) y sobre dicho archivo permite grabar los registros de las alarmas que incluyen la fecha, hora, identificador, nombre, temperatura, estado, latitud y longitud, como se muestra en la Figura 3.27.

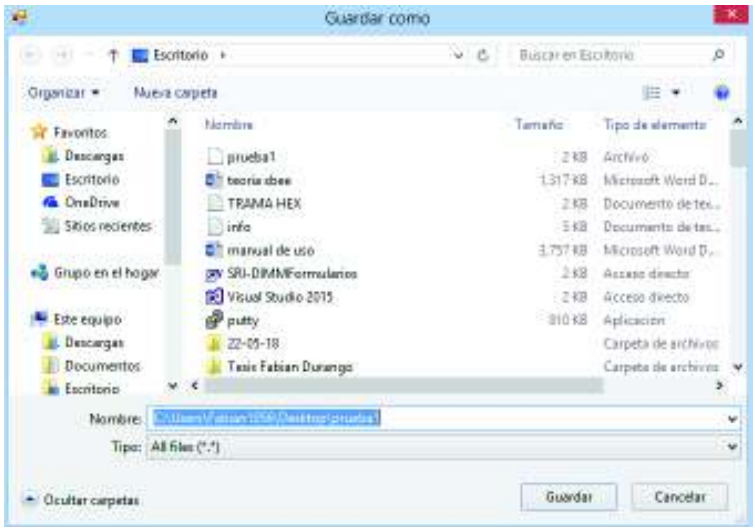


Figura 3.27 Grabar registro de alarmas

La Figura 3.28 muestra cómo se graba la información de las alarmas sobre el archivo de texto creado.

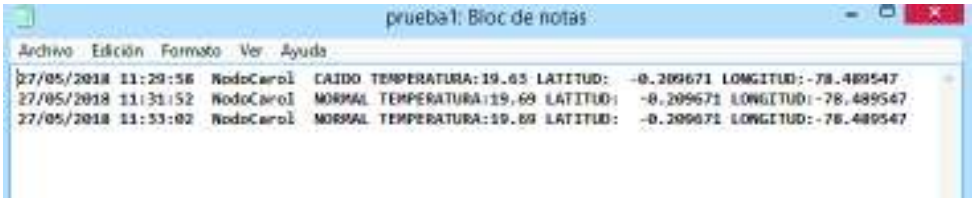


Figura 3.28 Registros de alarma guardados

3.4 Resultados

Una vez finalizado el prototipo se han generado los siguientes resultados.

- Se comprueba que es posible diseñar e implementar un prototipo de monitoreo de posición y temperatura basado en redes de sensores inalámbricas IEEE 802.15.4, utilizando motas Waspmote en la red de sensores y Visual Studio C# 2015 para la aplicación de escritorio, ya que cumplió con su objetivo que era el diseño e implementación del prototipo; además, dibuja sobre un mapa un marcador que muestra la posición actual de las distintas motas asociadas a la red; igualmente, se puede observar su temperatura y su estado; además de generar alarmas en el caso de sobrepasar la temperatura establecida y reconocer si la mota se encuentra en estado normal o caído.

- Como resultado del prototipo se obtiene la posición de la persona u objeto que porta el dispositivo con la ayuda del sensor GPS, el cual requiere un tiempo de conexión que varía dependiendo del lugar donde se ubica; para ambientes cerrados no existe conexión al satélite y este envía la posición 0.000000; al contrario, en ambientes abiertos toma entre 3 y 20 segundos el conectarse con los diferentes satélites que como mínimo deben ser tres, una vez conectado al satélite posee una exactitud de ± 3 metros.

- El sensor acelerómetro en todos los casos funcionó de la manera esperada ya que se encuentra integrado sobre la placa Waspmote; al interactuar con la aplicación de escritorio generó problemas en la función de detección del estado, este problema se corrigió convirtiendo el valor del eje z en un ángulo, esta acción se realizó en la mota y no en la aplicación dado que se lo utiliza para emitir el sonido de alarma en la mota, dejando el siguiente paso que fue interpretar el ángulo a estado caído o normal a la aplicación de escritorio.

- El sensor de temperatura fue el que más problemas generó dado que al usar una placa Waspmote que pertenece a la marca Libelium, posee sensores propios para su funcionamiento, esto limitó al prototipo dado que en nuestro mercado local no es comercial Libelium; se optó por usar el sensor DS18B20 el cual es mayormente usado para Arduino; pese a que Libelium dispone de una librería para este sensor existen casos en los que envía error en la lectura de temperatura.

- Se probaron tres tipos de sensores de la misma denominación DS18B20, pero con diferente encapsulado, uno diseñado para ambientes húmedos, otro integrado a una placa y otro con un sensor integrado; finalmente, se decidió implementar una placa propia con el sensor y una resistencia dado que la mayoría del tiempo generaban valores confiables en los datos.

- Se comprueba la funcionalidad de las alarmas de temperatura que al superar el valor máximo definido en 30°C o configurado un valor distinto por el usuario, emite un sonido tanto en la aplicación como en la mota; asimismo, al detectar que el estado es caído emite una alarma en la aplicación y en la mota, cada sonido es distinto entre alarma de temperatura y posición.

4 CONCLUSIONES Y RECOMENDACIONES

En base a los resultados obtenidos se muestran las conclusiones y recomendaciones del prototipo.

4.1 Conclusiones

- Se concluye que es posible crear una red de sensores inalámbrica basada en la tecnología Waspmote que interactúa con una aplicación de escritorio desarrollada en C#, la cual muestra sobre un mapa la temperatura, posición y estado de cada mota asociada a la red.

- El prototipo mostró un comportamiento dentro de lo esperado considerando el uso de los diferentes sensores en diferentes ambientes tanto abiertos como cerrados, en ambientes cerrados no utiliza la función de GPS, pero envía datos de temperatura y estado.

- El prototipo requiere su uso solo en ambientes externos para su funcionamiento adecuado, contando con un tiempo de espera que varía entre 3 y 20 segundos, transcurrido este tiempo el sensor reintenta la conexión hasta lograr establecer comunicación con los satélites y obtener su información.

- Se realiza la conversión de latitud y longitud de datos GMS (Grados, Minutos, Segundos) a grados decimales dentro de la mota dado que ahorra bytes en la transmisión, evitando que se envíen más caracteres y a su vez produciendo ahorro de energía en la mota.

- El acelerómetro integrado en cada mota es el elemento que menos problemas produjo dado que no necesita manipulación de hardware sino solo de software, lo cual permite obtener la posición en los ejes x, y, z.

- Se constató que la mejor manera de utilizar la función del acelerómetro fue usar el valor del eje z y convertirlo en ángulo (valores que varían entre -90 y 90°), para posteriormente enviar este dato al *gateway* y subsiguientemente enviar a la aplicación la cual determina si ese ángulo muestra un estado normal o caído.

- Los sensores de temperatura DS18B20 probados en el prototipo demostraron ser inestables ya que no enviaban el valor correcto de temperatura, por este motivo se probó con 3 tipos de sensores que son uno encapsulado para humedad, uno sobre una placa integrada y otro solo el sensor con una resistencia adaptado a una placa creada en el desarrollo de este prototipo concluyendo que es el más estable.

- Finalmente se puede concluir que Visual Basic es una herramienta muy útil para el desarrollo de aplicaciones que impliquen el manejo de puertos y el desarrollo de interfaces graficas; por esta razón se desarrolló la aplicación de escritorio basada en este IDE y se comprobó que funcionó de manera adecuada.

4.2 Recomendaciones

- Se recomienda el uso de los sensores GPS en ambientes abiertos, dado que el sensor debe detectar como mínimo 3 satélites para su conexión, generalmente con una exactitud de ± 3 metros.

- Al utilizar sensores de temperatura DS18B20, es indispensable la conexión de una resistencia de 4.7 KΩ entre los pines del sensor Vcc y Vout, esto ayuda a su funcionamiento caso contrario emitirá errores al momento de su comprobación.

- Los sensores DS18B20 son diseñados para la recopilación de datos digitales, esto ayuda en el momento de la programación dado que no es necesario interpretar dicha información, sino que se envía directamente como grados centígrados.

- Se recomienda actualizar el software de la placa Waspmote y el programa Waspmote PRO-IDE, ya que, al encontrarse en versiones no actualizadas entre ellas, el código puede dar problemas al intentar cargarlo.

- Se recomienda que los módulos de comunicación XBee 802.15.4 conectados a las motas y al *gateway* se encuentren en la misma red, canal y configuración, si alguno de estos módulos no se encuentra configurado correctamente no se podrá comunica dentro de la red.

- Es importante que al utilizar el programa Waspmote PRO-IDE, se seleccione en la opción de herramientas la ventana de placa y la de puertos ya que al estar en una placa o puerto distinto no se podrá cargar el código a la mota.

- Finalmente, para cargar el código a la placa Waspmote, se la debe conectar mediante un cable USB al computador y posteriormente encender los dos interruptores, sin esta acción no se podrá cargar el código.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] UNIVERSIDAD DE SEVILLA, “INTRODUCCIÓN A LAS REDES INALÁMBRICAS,” 2011.

[2] Diedrichs, “introduccion a las redes de sensores inalámbricas TELEINFORMATICA,” *Wireless Sensor Network*, vol. 1. pp. 1–20, 100AD.

[3] J. Salazar, “REDES INALÁMBRICAS,” *TechPedia*, p. 40.

[4] IEEE, “IEEE 802.15.4-2011 - IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).” [Online]. Available: https://standards.ieee.org/findstds/standard/802.15.4-2011.html. [Accessed: 02-Dec-2017].

[5] L. Villegas, “DSSS.” [Online]. Available: http://investigaciondsss.blogspot.com/. [Accessed: 28-Dec-2017].

[6] I. J. Cobos, Garcia, I. Jose, and A. Cruzatty, “Estudio técnico del comportamiento de trafico de internet,” 2017.

[7] D. Gascón, “TECNOLOGÍA Y SOCIEDAD. Redes de Sensores Inalámbricos, la tecnología invisible,” *Tecnol. y Soc.*, p. 54, 2010.

[8] Libelium Comunicaciones Distribuidas S.L., “Waspmote Datasheet v15,” p. 33, 2016.

[9] T. Guide, “Waspmote,” p. 133, 2017.

[10] U. Guide, “Waspmote IDE,” p. 16, 2014.

[11] S. Operativos, “El Lenguaje de programacion C,” p. 218.

[12] J. Barton and L. Nackman, “Scientific and Engineering,” p. 671, 1994.

[13] A. Hejlsberg, S. Wiltamuth, and P. Golde, *The C# programming language*. Addison-Wesley, 2004.

[14] S. Microsystems, “¿Qué es Java y para qué es necesario?” [Online]. Available: https://www.java.com/es/download/faq/whatis_java.xml. [Accessed: 03-Feb-2018].

[15] C. Nvidia, “Programming guide,” no. Revision 1, pp. 1–86, 2011.

[16] Libelium Comunicaciones Distribuidas S.L., “Waspmote Accelerometer,” p. 14, 2010.

[17] M. Capability *et al.*, “DS18B20 Programmable Resolution 1-Wire Digital Thermometer,” vol. 92, pp. 1–22.

[18] Libelium, “Waspmote 802.15.4 Networking Guide,” p. 6.

[19] D. International, “XBee/XBee-PRO 868 RF Module User Guide.”

[20] Putty, “Download PuTTY - a free SSH and telnet client for Windows.” 2017.

[21] Libelium, “SDK and Applications | Libelium.” [Online]. Available: http://www.libelium.com/development/waspmote/sdk_applications. [Accessed: 03-Feb-2018].

[22] “XBee X-CTU tutorial | Libelium.” [Online]. Available: http://www.libelium.com/development/waspmote/documentation/x-ctu-tutorial/. [Accessed: 10-Feb-2018].

[23] Libelium, “Waspmote Data Frame,” p. 18, 2013.

[24] E. Péter Cosma, “Home - Pencil Project,” 2013. [Online]. Available: https://pencil.evolus.vn/. [Accessed: 20-Feb-2018].

[25] “GMap.NET - Great Maps for Windows Forms & Presentation.” [Online]. Available: https://archive.codeplex.com/?p=greatmaps. [Accessed: 20-Feb-2018].

[26] Visual Studio, “IDE de Visual Studio, editor de código, Team Services y Mobile Center,” 2017. [Online]. Available: https://www.visualstudio.com/es/. [Accessed: 27-Feb-2018].

[27] U. de Valencia, “Modulación digital.,” *Univ. Val.*, vol. 5, pp. 1–18, 2013.

[28] J. Rodríguez and S. Suero, “Modulación de Señales Digitales,” *Univ. Politécnica Sevilla, …*, 1995.

[29] H. Schwedick and A. Wolf, “PSSS - parallel sequence spread spectrum a physical layer for RF communication,” in *IEEE International Symposium on Consumer Electronics, 2004*, pp. 262–265.

[30] D. I. Kim and T. Jia, “M-ary orthogonal coded/balanced ultra-wideband transmitted-reference systems in multipath,” *IEEE Trans. Commun.*, vol. 56, no. 1, pp. 102–111, Jan. 2008.

[31] P. Borensztejn, “Diseño de Sistemas con FPGA 1er cuatrimestre 2013 Patricia Borensztejn,” 2013.

ANEXOS

ANEXO A: DATA FRAME GUIDE, GUÍA DE MANEJO DE TRAMAS EN LIBELIUM

El documento se encuentra adjunto como anexo digital.

ANEXO B: QUICK START GUIDE, GUÍA INICIO RÁPIDO LIBELIUM

El documento se encuentra adjunto como anexo digital.

ANEXO C: WASPMOTE IDE USER GUIDE, GUÍA MANEJO DE PLATAFORMA WASPMOTE

El documento se encuentra adjunto como anexo digital.

ANEXO D: WASPMOTE TECHNICAL GUIDE, GUÍA TÉCNICA WASPMOTE, PUERTOS.

El documento se encuentra adjunto como anexo digital.

ANEXO E: WASPMOTE 802.15.4 NETWORKING GUIDE, GUÍA MANEJO COMUNICACIÓN.

El documento se encuentra adjunto como anexo digital.

ANEXO F: WASPMOTE GPS PROGRAMMING GUIDE, GUÍA MANEJO GPS

El documento se encuentra adjunto como anexo digital.

ANEXO G: CÓDIGO DE LOS NODOS SENSORES

El documento se encuentra adjunto como anexo digital.

ANEXO H: CÓDIGO DE LA APLICACIÓN DE ESCRITORIO

El documento se encuentra adjunto como anexo digital.