UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

APLICACIÓN DEL MODELO DE CANAL DE COMUNICACIONES EN LÍNEA DE VISTA PARA POSICIONAMIENTO RELATIVO ENTRE DISPOSITIVOS EN MOVIMIENTO

TESIS
PARA OBTENER EL GRADO DE
INGENIERO EN REDES

PRESENTA
ROYER FRANCISCO SANTOS CHIMAL

DIRECTOR DE TESIS
DR. JAVIER VÁZQUEZ CASTILLO

ASESORES
MTI. VLADIMIR VENIAMIN CABAÑAS VICTORIA DR.
JAIME SILVERIO ORTEGÓN AGUILAR
MSI. RUBÉN ENRIQUE GONZÁLEZ ELIXAVIDE
MTI. MELISSA BLANQUETO ESTRADA

CHETUMAL QUINTANA ROO, MÉXICO, MARZO DE 2017

UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TRABAJO DE TESIS ELABORADO BAJO SUPERVISIÓN DEL COMITÉ DE ASESORÍA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERO EN REDES

COMITÉ DE TESIS

DIRECTOR:

DR. JAVIER VÁZQUEZ CASTILLO

ASESOR:

MTI. VLADIMIR VENIAMIN CABAÑAS VICTORIA

ASESOR:

DR. JAIME SILVERIO ORTEGÓN AGUILAR

UNIVERSIDAD DE QUINTANA ROO
SERVICIOS ESCOLARES
TITULACIONES

CHETUMAL QUINTANA ROO, MÉXICO, [illegible] DE 20[illegible]

Universidad de Quintana Roo
DCI DIVISIÓN DE CIENCIAS E INGENIERÍA

Dedicatoria

Dedico este trabajo, a todas aquellas personas cercanas que confiaron en mí, que me apoyaron incondicionalmente a lograr el objetivo de poder terminar una carrera y que cada día me impulsan a ser mejor en todos los aspectos de mi vida.

A mis padres, abuelos y hermanas, que me brindaron el apoyo siempre para seguir adelante, a pesar de todas las adversidades que se nos presentaron y la fortaleza para no rendirme ante nada.

A aquellos tíos que fueron parte importante en mi superación académica y personal, muchas gracias, siempre estaré agradecido con ustedes.

A mis profesores, que con sus enseñanzas del día a día, me brindaron las herramientas para superar las pruebas que se presentaban en la carrera.

También dedico este trabajo a Kari, pilar fundamental durante casi 6 años en mi formación profesional y personal.

Agradecimientos

Este trabajo de tesis tuvo apoyo del “Proyecto Ciencia Básica 2014/241272” y de SEP-PRODEP 2014.

Agradezco a mi madre, por ser una madre ejemplar, por estar siempre ahí para nosotros, por todo el apoyo incondicional que me ha brindado durante toda la vida.

A mi padre, agradezco todo el esfuerzo que ha hecho por nosotros, por ser un padre trabajador, que siempre ha buscado la manera de sacarnos adelante a pesar de todas las complicaciones que la vida nos presentó.

A mis hermanas: Yudi, Ney, Yas y Ximena, las tengo presente todo el tiempo, gracias por el apoyo de toda la vida.

A mis tíos: Alex, Amalia, Selina y Gerardo, parte fundamental en mi desarrollo académico y personal durante mucho tiempo, gracias por toda la ayuda que me brindaron, siempre estaré agradecido.

Al Dr. César Cristóbal Escalante, gracias a usted, pude en un momento crítico de la carrera, poder seguir adelante, le agradezco las oportunidades y amistad que me ha brindado.

Al Dr. Javier Vázquez Castillo, le agradezco todo el apoyo que me dio durante la elaboración de este proyecto, así como del apoyo académico y personal durante varios ciclos en la carrera.

A mis profesores: Jaime, Rubén, Vladimir y Melissa, gracias por toda la enseñanza, consejos y apoyo brindado.

A Kari: mi novia, amiga, compañera y apoyo incondicional en el transcurso de mi carrera, te agradezco infinitamente todo lo que has hecho por mi durante estos casi 6 años, por todos los consejos, por alentarme a seguir adelante, por todos los momentos en los que estuviste conmigo y para mí.

RESUMEN

Este trabajo de tesis brinda una solución a las dificultades que se presentan en la medición de la distancia entre dispositivos que se encuentran en movimiento. Para ello, se implementó un esquema de transmisión/recepción de datos utilizando el sistema de comunicación inalámbrica a nivel de operación de módulos Xbee, así como, la recepción y representación de la intensidad de la señal recibida (RSSI), en una placa Arduino Uno.

Se estableció un algoritmo para la estimación de la distancia entre módulos Xbee haciendo uso de los valores RSSI obtenidos. De esta manera, en los resultados se reflejan los valores RSSI y la distancia estimada en conjunto, de los dispositivos involucrados.

Se realizaron pruebas de envío/recepción de datos y estimación de distancia bajo entornos reales de propagación con la metodología propuesta. Al finalizar las pruebas, se compararon las curvas estimadas con las de referencia real, y los resultados fueron mejores a los esperados.

Finalmente, de manera adicional este proyecto de tesis brinda información y material importante y de gran utilidad a estudiantes, profesores y público en general que desee conocer o reforzar sus conocimientos sobre la tecnología Xbee y el microcontrolador Arduino Uno.

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 CAPITULO 1.

1.1 ANTECEDENTES

Actualmente se han realizado trabajos de investigación para calcular el posicionamiento relativo entre dispositivos a través de la potencia de la señal recibida (RSSI). Los resultados de estos estudios arrojan mediciones inexactas, es necesario utilizar una gran cantidad de muestras para lograr una estimación de una medición lo más aproximada posible. En muchos casos, los factores externos influyen de manera importante en la calidad de las mediciones realizadas, es decir, el medio en los cuales se realizan (geometría física del ambiente de propagación), los dispositivos empleados, además de que los valores de la potencia de la señal (RSSI) son inestables, y aun teniendo un dispositivo quieto, éste tiende a no tener valores fijos.

Algunos métodos usados para el cálculo de distancia entre dispositivos más relevantes son:

- ***Sistema de Posicionamiento Basado en Bluetooth con Calibrado Dinámico*** [1, 2]. Consiste en el uso de técnicas de triangulación usando la potencia de la señal recibida (RSSI). Consta de varios dispositivos a ser localizados que usan la información de otros dispositivos que actúan como balizas. Las balizas tienen una posición fija y conocida. En la Figura 1 podemos ver como se estructura el sistema de localización.

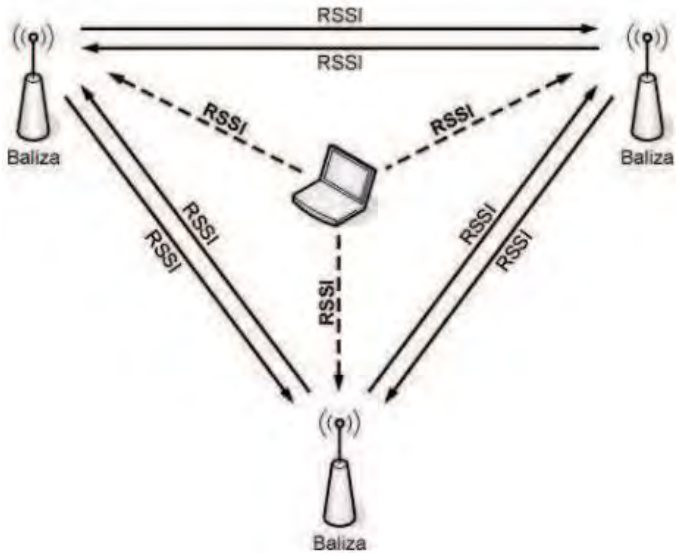

Figura 1. Sistema de localización por bluetooth usando RSSI (Javier Rodas, 2009).

- ***Evaluación de la Fiabilidad del RSSI para la Localización en Interiores*** [3]: En redes donde se usan sensores inalámbricos, los nodos involucrados, pueden ser estáticos o móviles, dependiendo del tipo de sistema o aplicación donde se emplean y las funciones que buscan realizar. Lidiar con elementos en un espacio cerrado y la movilidad de los nodos, hace que el diseño, la conectividad y la localización de los dispositivos sea más complicado. Se usan varios métodos para la calibración y mapeo de la potencia de la señal para conocer la fiabilidad en el uso del RSSI para la localización en interiores entre los que incluyen el Método de Promedio Ponderado, Método de ajuste de la Curva. Entre otros métodos encontramos algunos algoritmos basados en modelos de pérdida de ruta, así como el uso del algoritmo ROCRSSI.

- ***Localización de Nodos usando RSSI en una Red de Sensores Inalámbricos (WSN) por medio de Trilateración***. [4, 5]: En una red WSN se requiere conocer la ubicación de ciertos nodos para que la detección y recolección de información sea más precisa y

significativa, pero el esquema de distribución de este tipo de redes hace difícil estimar la ubicación de los dispositivos.

La Trilateración, es uno de los métodos más comunes para este tipo de problemas; usando la intensidad de la señal recibida (RSSI) en la estimación de distancia entre nodos; aunque es una técnica sencilla y muy común, arroja un margen de error significativo.

1.2 PROBLEMÁTICA

Hoy en día, son muchas las aplicaciones que necesitan estimar la posición relativa entre dispositivos como son un transmisor y un receptor de datos. La propagación de una señal a través de un medio inalámbrico, tiende a estar en conflicto con gran parte de los elementos a su paso; en este sentido, varios factores hacen que la calidad de la trasmisión se vea afectada; por ejemplo, la propagación multitrayecto, la temperatura, los muebles, puertas, así como la presencia de personas.

Es importante mencionar que la potencia de la señal en los dispositivos en reposo no es constante, aun en condiciones óptimas, la variación de este valor es notable. En comunicaciones de vehículos aéreos no tripulados (UAV, Unmanned Aerial Vehicle), cuando se necesita posicionar en escala de centímetros, una unidad GPS no brinda la resolución esperada. Otra aplicación importante se da en la comunicación de vehículo a vehículo (estándar IEEE 802.11p) [6], donde es necesario calcular el tiempo estimado durante el cual los vehículos podrán establecer comunicación para el intercambio de información, por lo que es crucial el poder estimar la posición o distancia entre vehículos involucrados.

1.3 JUSTIFICACION

La creciente demanda de crear dispositivos que cada día mejoren la calidad de envío y recepción de datos, así como evitar fenómenos negativos como la pérdida por propagación de la señal transmitida, aunado a la aparición de nuevos estándares y aplicaciones en el campo de las comunicaciones, ha traído la necesidad de proponer una metodología de estimación de la distancia entre dispositivos móviles.

Existe la posibilidad de conocer la distancia relativa entre dispositivos mediante la medición de la potencia de la señal que se transmite (RSSI) y el conocimiento del canal de comunicaciones.

Este trabajo de tesis plantea la problemática establecida y buscará estimar la distancia entre dispositivos móviles, así como la intensidad de la señal trasmitida en un medio con línea de vista.

1.4 OBJETIVO GENERAL

El objetivo general de este proyecto, consiste en la estimación de la distancia entre un dispositivo transmisor y un dispositivo receptor con línea de vista, haciendo uso de la intensidad de la señal recibida (RSSI) en módulos de radiofrecuencia Xbee.

1.5 OBJETIVOS ESPECIFICOS

- Determinar la pérdida por propagación y potencia de señal (RSSI).
- Definir las características relevantes del estándar Xbee para esta propuesta.
- Implementar los algoritmos para el funcionamiento de los radios Xbee.
- Proponer una técnica de estimación de distancia usando la información del RSSI.
- Implementar e integrar el algoritmo y los dispositivos necesarios.
- Realizar las pruebas finales.

2 CAPITULO 2: ARQUITECTURA DEL SISTEMA

2.1 MICROCONTROLADORES:

A lo largo del tiempo, la tecnología ha ido evolucionando de tal manera que hoy en día, es parte de nuestra vida cotidiana, está presente en casi todo lo que nos rodea, desde las tareas más simples, hasta aquellas que conllevan una mayor dificultad. Existen muchos dispositivos electrónicos que cumplen con funciones importantes, los microcontroladores son una parte importante en este mundo en constante evolución tecnológica. Desde su aparición en el año 1971, fecha en que Intel presentó el primer microcontrolador de 4 bits con velocidad de 6000 operaciones por segundo, llamado 4004, y un año después, presentarían un microcontrolador de 8 bits, llamado 8008, el cual sería el predecesor a los microcontroladores que conocemos hoy en día; el avance y evolución de estos dispositivos se ha disparado hasta convertirse en poderosas herramientas indispensables para un gran número de sectores académicos e industriales, surgiendo constantemente nuevas mejoras y versiones con más funcionalidades y capacidades.

Un microcontrolador es un circuito integrado, también llamado “chip”, que tiene la capacidad de ser programable. Es decir, es capaz de ejecutar de manera autónoma una serie de instrucciones que el usuario le defina previamente. Éste se compone de bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador contiene 3 unidades funcionales principales: la unidad central de procesamiento, la memoria y los periféricos de entrada/salida.

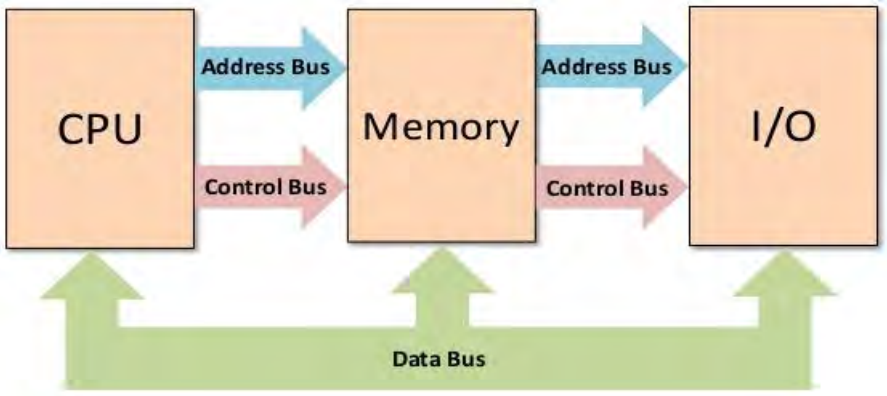

Figura 2. Arquitectura de un microcontrolador.

En la Figura 2, podemos observar un esquema de los componentes que integran un microcontrolador, los componentes se unen por medio de buses, los cuales pueden ser de direcciones, de control o de datos.

Los microcontroladores están perfectamente diseñados para hacer económicos tanto en cuestión de energía, tamaño y precio a un sistema en particular, dependiendo de la aplicación en la que se utilice, el tamaño de la unidad central de procesamiento, memoria y periféricos deberán tener las características necesarias.

Los microcontroladores se encuentran en gran cantidad de cosas que usamos todos los días (automóviles, computadoras, celulares, etc.), algunos mientras más microcontroladores tengan, más capacidades tienen y son más robustos. Por eso no es lo mismo el microprocesador usado en una licuadora al usado en un teléfono o una computadora.

Un microcontrolador tiene un reloj integrado y una cantidad de memoria, el microcontrolador pone en ejecución todas las instrucciones almacenadas en memoria que han sido asignadas por el usuario, para que después, interactúen con el exterior a través de las líneas de entrada/salida por medio de sensores y actuadores. Hoy en día es posible que los microcontroladores puedan ser manipulados mediante un lenguaje de programación integrado como BASIC, C++, entre otros.

Los microcontroladores se usan en gran variedad de aplicaciones, y el uso de ellos trae ventajas considerables como:

- Reducción del producto en tamaño: El uso de microcontroladores disminuye el tamaño de los productos donde se aplican, así como la reducción de mano de obra en el ensamblaje de chips.
- Flexibilidad: La modificación solo necesita cambios en el programa de instrucciones.
- Fiabilidad: Más probabilidad de que el sistema o aplicación cumpla sus funciones aún con cambios en los elementos que la conforman.

Como se mencionó anteriormente, los microcontroladores se encuentran en más del 50% de los aparatos que usamos día con día. Cuando se requiere hacer uso de un microcontrolador, se deben

tomar ciertos requisitos y características para que se adapten al sistema, aparato o aplicación a desarrollar. Algunas de esas características y requisitos a analizar son:

- Procesamiento de datos: Algunas veces será necesario que el microcontrolador realice grandes cálculos en determinado tiempo, se debe asegurar seleccionar un dispositivo que cumpla la rapidez necesaria. Otro punto importante es la precisión de los datos que se manejarán. Si no es suficiente con un microcontrolador de determinado número de bits (8 bits por ejemplo), se debe buscar la alternativa de usar uno de 16 bits o 32 bits, pero se debe considerar también el costo de ello.
- El consumo: Algunos sistemas que usan microcontroladores usan energía a base de baterías. Para optimizar el bajo consumo, se pueden activar a través de un interruptor solo cuando se necesite, y el resto del tiempo permanecer en reposo, pero de igual manera hay que considerar aquellos aparatos o aplicaciones que necesitan estar activas todo el tiempo.
- Dispositivos de Entrada/Salida: Para conocer las necesidades de Entrada/Salida del sistema y/o producto, se debe dibujar un diagrama de bloques del mismo, para poder identificar que se necesita controlar y el número de periféricos.
- La memoria: Para conocer las necesidades de memoria de nuestro producto o aplicación, debemos analizar por separado la memoria volátil (RAM), la no volátil (ROM, EPROM) y memoria volátil modificable (EEPROM). Tomando a consideración las capacidades de nuestra aplicación, de la robustez y fiabilidad, dependerá el tipo de memoria que emplearemos, así como del uso y mercado al cual será útil.
- Diseño de la placa: Dependiendo del tipo de microcontrolador que usemos, determinará el diseño de la placa de circuitos. Mientras mejor sea el microcontrolador, mejor será el resto del producto o sistema, usar un microcontrolador barato, limitará las capacidades del producto o aplicación.

Un microcontrolador normalmente dispone de los siguientes componentes:

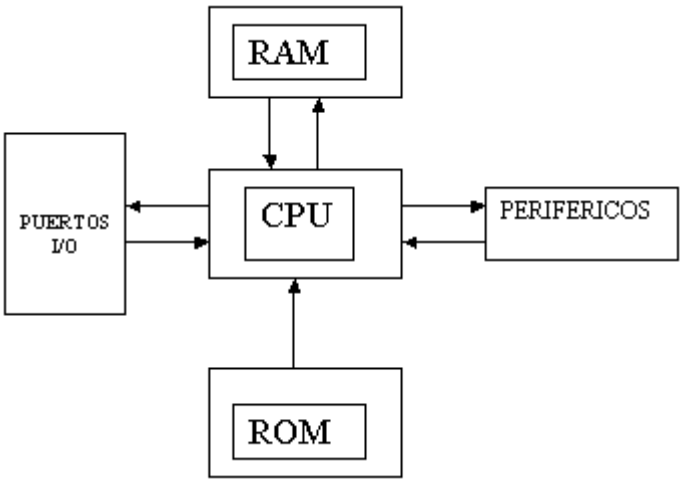

Figura 3. Componentes de un microcontrolador.

- CPU (Unidad Central de Procesamiento): Una de las partes más importantes del microcontrolador. La unidad central de procesamiento es la encargada de leer, interpretar y ejecutar las instrucciones almacenadas en la memoria; es encargado del control de registros y de la unidad aritmético-lógica (ALU). Es fundamental para el funcionamiento del procesador, determina parámetros como la velocidad de ejecución, tiempo del ciclo de máquina, los tipos de buses, etc.
- Memorias RAM, ROM/PROM/EPROM: En los microcontroladores encontramos una cantidad de memoria limitada. La memoria RAM es la encargada de almacenar información temporal que es usada por el procesador para realizar cálculos y operaciones lógicas. El tipo de memoria RAM que utilizan los microcontroladores es llamada SRAM; el cual evita problemas de calentamiento.

  La memoria ROM (Memoria de solo lectura), en el cual se almacenan los programas
- Líneas de Entrada/Salida: Parte fundamental de los microcontroladores, estos elementos permiten leer datos del exterior o bien escribir en ellos desde el interior del microcontrolador. La función final de las líneas de entrada/salida es trabajar con dispositivos como LED's, relés, y cualquier otro dispositivo que se requiera para el sistema o aplicación.

2.2 ARQUITECTURAS HARVARD Y VON NEUMANN

La arquitectura de un microcontrolador permite definir la estructura de su funcionamiento. Las instrucciones que se almacenan en memoria pasan por la Unidad Central de Almacenamiento (CPU) donde son decodificados y ejecutados, la conexión que existe entre CPU y memoria es un aspecto fundamental a considerar. Existen dos arquitecturas distintas relacionadas con el uso y distribución de memoria: Arquitectura Von Neumann y Arquitectura Harvard.

Los modelos Von Neumann y Harvard, la primera creada en 1945 por el físico-matemático John Von Neumann, el cual describe un modelo computacional que dispone de una única memoria principal donde se almacenan datos e instrucciones, a los cuales es posible acceder por medio de buses, ya sea de datos, de direcciones o de control.

La arquitectura Harvard cuenta con dos memorias independientes, una contiene instrucciones, y la otra, datos, ambas con sus respectivos sistemas de buses de acceso, y es posible realizar operaciones de lectura o escritura simultáneamente en ambas memorias.

En la Figura 4, podemos observar la estructura de la arquitectura de Harvard, la cual, como se menciona anteriormente, posee dos memorias por separado conectada a la CPU con sus respectivos buses, los cuales son totalmente independientes. Una de las ventajas de este modelo es que, al tener las memorias por separado, el tamaño de las instrucciones no está directamente relacionado con el de los datos, por lo que puede se puede optimizar para que cualquier instrucción ocupe una posición en la memoria del programa, logrando con ello, mayor velocidad en las consultas y menor longitud del programa.

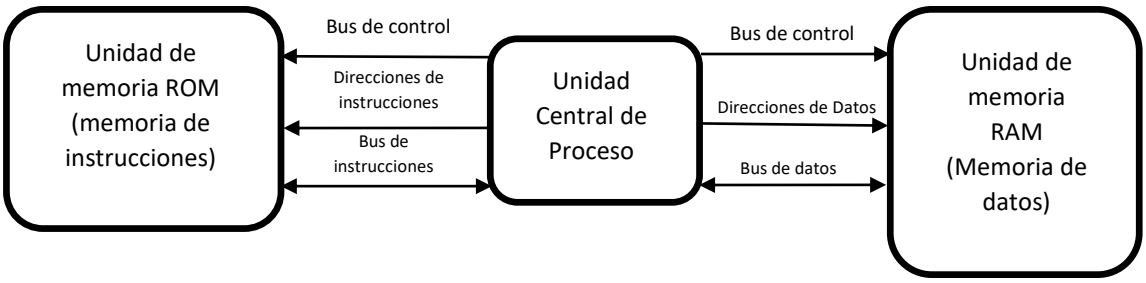

Figura 4. Esquema Arquitectura Harvard.

En la Figura 5 tenemos la arquitectura de Von Neumann, donde podemos ver que utiliza únicamente una memoria central para almacenar datos e instrucciones. Una desventaja de este modelo, es por la longitud de las instrucciones que pueden pasar por el bus de datos, requiere que se realicen varios accesos a memoria para buscar instrucciones más complejas. Otra limitación, es la velocidad de operación a causa del único bus de datos e instrucciones, el cual no permite acceder simultáneamente a uno u otro, lo cual implica dividir el tiempo entre procesos, ya que no es posible superponer ambos tiempos de acceso.

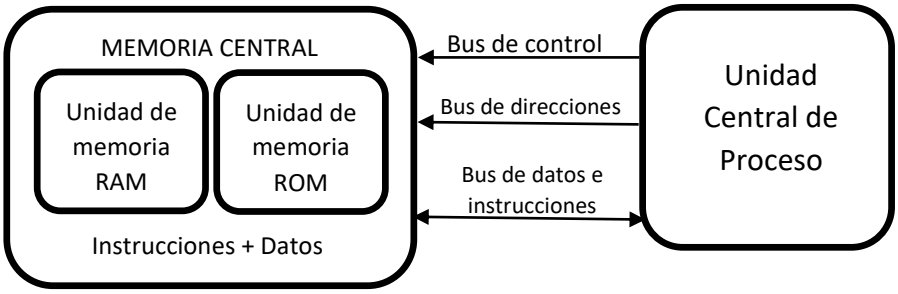

Figura 5. Esquema Arquitectura Von Neumann

2.3 ARDUINO:

Arduino [7], es el nombre que recibe un conjunto de placas con software libre más usados en todo el mundo para la creación y configuración de proyectos electrónicos, desde un sistema de control de agua, hasta un servidor web, etc. El hardware de Arduino consiste es un circuito impreso con un microcontrolador basado en la tecnología Atmel AVR; la familia de microcontroladores AVR es un CPU de arquitectura Harvard. Es como tener una pequeña computadora a la cual le podemos conectar una gran cantidad de dispositivos, sensores, actuadores y otros módulos (shields) entre otros circuitos integrados, con lo cual podemos convertir el Arduino en un cerebro programable para cualquier sistema o aplicación, lo único que nos limita es la imaginación. Arduino es muy fácil

de usar, no se requieren grandes conocimientos en electrónica o programación, podemos hacer de Arduino convertirse en lo que queramos, el lenguaje usado por Arduino, está pensado para ser muy intuitivo y de fácil aprendizaje.

Gracias a la extensa comunidad que existe detrás de esta plataforma y de la ventaja de ser código libre, existen muchos proyectos que podemos encontrar y mejorar, así como gran cantidad de hardware compatible. Debido a la popularidad y demanda que obtuvo, muchos fabricantes han ido mejorando el entorno de Arduino, sacando nuevos productos (hardware) que se pueden usar en conjunto con la placa Arduino, los cuales dan una mejora considerable a los proyectos.

2.3.1 VENTAJAS DE ARDUINO

- Entorno de programación muy simple: El entorno de programación de Arduino es muy sencillo, está pensado para aquellas personas sin amplios conocimientos en programación, además de ser muy flexible para aquellos con experiencia. Basado en el lenguaje de programación Processing, los usuarios con conocimientos en este entorno, podrá adaptarse muy rápido al entorno de desarrollo de Arduino. Arduino cuenta con su propia página web, así como existen cientos de foros y sitios web, donde se pueden compartir información, aclarar dudas y hacer colaboraciones.
- De código abierto, y software ampliable: El software que se utiliza para la programación de Arduino es libre, se puede conseguir de manera gratuita. Cualquier usuario con amplios conocimientos en programación puede ampliar el software, mediante la creación de nuevas librerías.
- Es de bajo costo: Existe una gran variedad de placas Arduino, dependiendo de las necesidades del usuario, las cuales resultan más accesibles comparadas con otras plataformas de microcontroladores similares. En México podemos conseguirlas a un precio bastante aceptable.
- Usa hardware ampliable: Arduino está basado en la familia de microcontroladores de ATMEL, en específico de megaAVR. Las arquitecturas de los módulos se pueden

encontrar en línea bajo licencia, por lo tanto, diseñadores de circuitos pueden crear su propia versión del módulo, pueden recrear su propia placa, y así poder agregarle nuevas características y optimizarlo.

- Es multiplataforma: El software que utiliza Arduino cuenta con la gran ventaja de poder funcionar tanto en Windows, Macintosh OSX o Linux, una característica que no todos los entornos de microcontroladores pueden hacer, muchos de ellos limitados únicamente a ser usado en Windows.

Arduino, simplifica el proceso de trabajar con microcontroladores, gracias a todas las características mencionadas con anterioridad, lo pone por encima de otras plataformas, la preferencia de los usuarios es mayor debido a todas las posibilidades que ofrece.

2.3.2 MÓDULOS ARDUINO DISPONIBLES:

Existe una variada cantidad de modelos de placas Arduino disponibles, los cuales se eligen según las necesidades del usuario o las exigencias del programa o sistema. Todos cumplen con tareas específicas, y juntos aumentan sus capacidades y posibilidades para crear nuevos proyectos. Algunos de los más usados se enlistan a continuación:

- Arduino UNO (éste es uno de los más usados)
- Arduino Leonardo
- Arduino 101
- Arduino Robot (contiene dos microcontroladores)
- Arduino Esplora (basado en el Leonardo)
- Arduino Micro
- Arduino Nano (compacto, similar al UNO)
- Arduino Mini
- Arduino Mega
- Arduino Zero
- Arduino Due

- Arduino Pro
- Arduino Yún
- Arduino Gemma
- Arduino Lilypad
- Entre muchos otros modelos…

Ahora, podemos describir algunos de estos modelos, ya que son muchos para describirlos todos por completo, tenemos que resaltar y describir más a fondo el Arduino UNO, ya que es el modelo que usamos en este proyecto:

Arduino Leonardo: El Arduino Leonardo es una placa integrada con un microcontrolador ATmega32u4. Cuenta con 20 pines digitales de entrada/salida, y trabaja a 16 MHz. Entre otras características generales, tiene integrada una conexión micro USB y un conector de alimentación. En memoria encontramos 32 KB, de los cuales 4KB son utilizados en el arranque, cuenta con 2.5 de SRAM y 1 KB de EEPROM. En la Figura 6 podemos apreciar el modelo descrito.

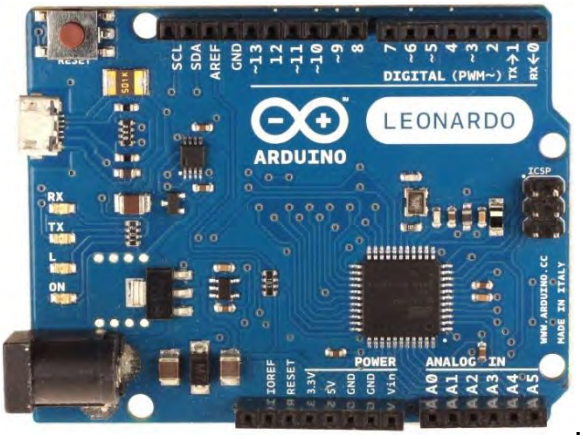

Figura 6. Arduino Leonardo.

Arduino Micro: El Arduino Micro está basado en su hermano mayor, el Arduino Leonardo, utilizando el mismo microcontrolador ATmega32u4, con las mismas características en memoria y velocidad de trabajo de 16 MHz, así como el mismo número de pines, pero en tamaño mucho más compacto. Dispone de un puerto USB nativo, el cual evita tener que usar un convertidor de serie a USB. En la Figura 7 podemos ver el modelo mencionado:

Figura 7. Arduino Micro.

Arduino MEGA: El Arduino Mega es uno de los productos más poderosos que existen en la familia Arduino. Con 54 pines funcionando como entrada/salida y entradas análogas, trabajando a una velocidad de 16 MHz. Esta placa, debido a sus capacidades, es utilizada en grandes proyectos que requieren de sus características, usado por ejemplo en domótica, así como en impresoras 3D. Es compatible con la mayoría de las tarjetas de ampliación, así como de las shields disponibles para lograr obtener mayores capacidades y ser más robusto. En la Figura 8 apreciamos el modelo:

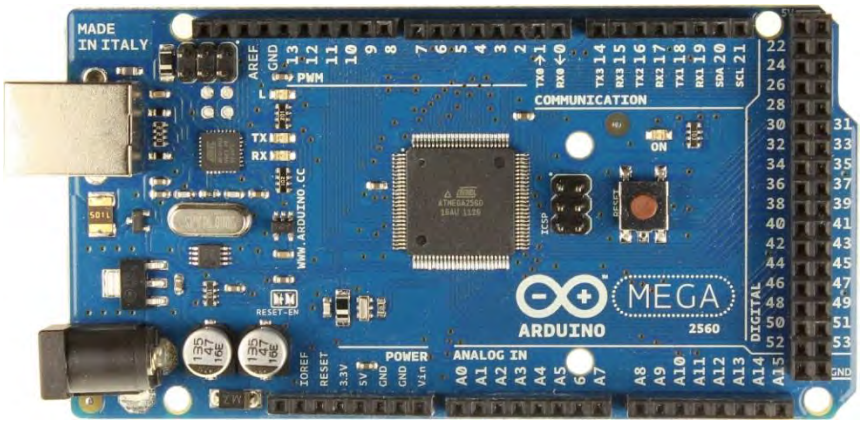

Figura 8. Arduino Mega.

Arduino Gemma: Un proyecto de Arduino en colaboración con la empresa Adafruit, dieron como resultado Arduino Gemma; consiste en una placa circular del tamaño de una moneda. Una placa compatible con todos los demás productos Arduino, es decir hardware y software. Cuenta con 8K de flash, 512 bytes de SRAM, 512 bytes de EEPROM y trabaja a una velocidad de 8 MHz. Unas ventajas considerables de este modelo, son su tamaño, su bajo costo y su bajo consumo de energía a pesar de su pequeña capacidad de memoria. Ideal para proyectos donde se requiera poco espacio en tamaño. En la figura 9 podemos ver el modelo:

Figura 9. Arduino Gemma.

Arduino ESPLORA: El Arduino Esplora, es un controlador listo para ser usado, permite una gran cantidad de proyectos, viene integrado con sensores y actuadores, sin tener que soldarlos ni utilizar cables o una protoboard. Algunos de los sensores y actuadores son los siguientes: un sensor de luz, un sensor de temperatura, un acelerómetro, un potenciómetro, entre otros. Basado en el Arduino Leonardo, cuenta con un microcontrolador ATmega32u4, con 2.5KB de SRAM, 1KB de EEPROM y 32 KB de memoria flash. Su peculiar diseño y el conjunto de elementos integrados, hace al Arduino Esplora, un modelo muy diferente a los demás modelos presentados. En la Figura 10 apreciamos mejor el diseño de este modelo:

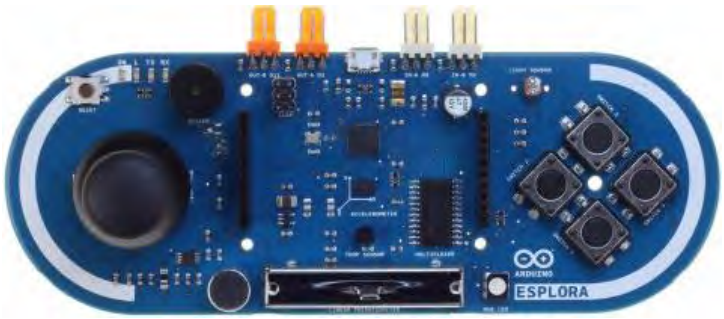

Figura 10. Arduino Esplora.

Arduino Due: La placa Arduino Due, es el primer modelo basado en la arquitectura ARM. Esta poderosa placa contiene un microcontrolador ARM CortexM3 de 32 bits trabajando a una velocidad de 84 MHz, aumentando de esta manera la potencia de cálculo disponible para los usuarios. Este modelo cuenta con dos puertos USB, uno para la programación y comunicación, y el otro actúa como cliente o host, o para utilizar un mouse o teclado, lo cual aumente las posibilidades de eso para esta placa. Además, es el primer Arduino en tener integrado dos conversores de digital a analógico. Sin duda una excelente opción para grandes proyectos, en la Figura 11 podemos apreciarlo mejor:

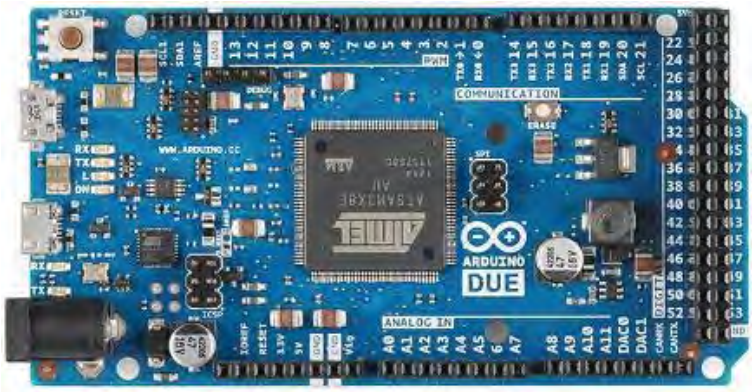

Figura 11. Arduino DUE.

A continuación, se presenta la placa utilizada en el presente proyecto, el Arduino Uno, este se describirá con más detalle, en la Figura 12 podemos ver el modelo del cual hablaremos. Arduino UNO es la uno de los productos más usado de la familia Arduino y une excelente opción empezar a realizar proyectos basado en esta plataforma. Esta placa viene integrada con el microcontrolador Atmega328, una conexión USB por la cual se puede programar mediante una PC, usando el software de Arduino, cuenta con fuente de alimentación, la cual se puede conectar por medio de corriente eléctrica o usando un adaptador AC-DC para usar una batería.

Cuenta con 14 pines de entrada/salida, de las cuales 6 se pueden usar como salidas PWM (señales moduladas por ancho de pulso), 6 entradas analógicas, donde podemos conectar una gran cantidad de sensores y actuadores que son compatibles, trabaja a una velocidad de 16 MHz. El microcontrolador cuenta con 0.5 KB para el gestor de arranque, viene con 2 KB de memoria SRAM y 1 KB de memoria EEPROM.

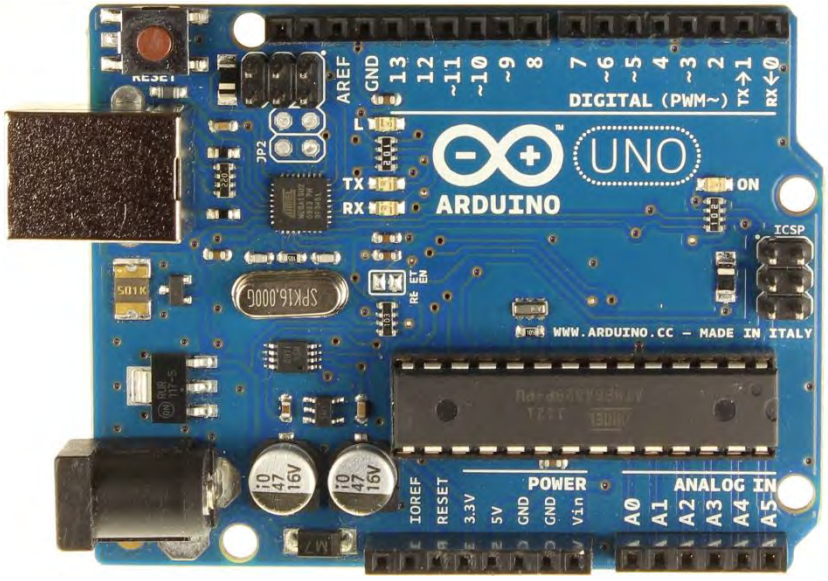

Figura 12. Arduino UNO.

Podemos describir algunos elementos, así como de los pines con los que cuenta el Arduino UNO, así como de las funciones y tareas que llevan a cabo:

- Pines RX, TX: Se utilizan para la recepción y envío de datos serie TTL.
- Pines 2 y 3: Estos pines se usan para activar una interrupción en el microcontrolador. Se activan cuando existe un valor bajo, cambio de un valor o un flanco ascendente o descendente.
- Pines 0 – 13: Cuenta con 14 pines digitales de entrada/salida dependiendo de las necesidades. Proporcionan un máximo de 40 mA por pin, y funcionan a 5V.
- Pines GND: Pines con función de tierra.

- Pines SPI: Pines que pueden utilizarse para llevar a cabo comunicaciones SPI, los cuales permiten trasladar información full dúplex en un entorno maestro/esclavo. El bus de interfaz periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico-digital que acepte flujo de bits serie regulado por un reloj.
- Pin IOREF: Pin que proporciona la referencia de voltaje con la que el microcontrolador trabaja.
- Pines 3.3V y 5V: Pines de referencia de voltaje para el uso de la placa, no deben usarse con demanda de alto voltaje, las cuales no deben pasar de 50 mA ya que podría dañarse. La intensidad viene dada de la fuente de alimentación que se utilice.
- Pin VIN: Pin para la tensión de entrada a la placa cuando se usa una fuente de alimentación externa. Se puede proporcionar voltaje a través de este pin, o, si se alimenta a través de una conexión de 2.1mm, acceder a ella a través de este pin.
- Pin RESET: Pin con el cual podemos reiniciar la placa Arduino UNO, con él, podemos borrar un programa cargado en el microcontrolador y escribir uno nuevo.
- Pin AREF: Con el cual se establece la tensión de referencia para las entradas analógicas.
- Pin ICSP: Conector para la programación serial en circuito, es un sistema utilizado en los PIC para poder programarlos sin tener que retirar el chip del circuito.

En la Tabla 1 podemos ver los datos técnicos del Arduino UNO:

Microcontrolador	ATmega328
Voltage de funcionamiento	5V
Alimentación (recomendada)	7-12V
Voltage máximo de entrada(no recomendado)	20V
Pines digitales I/O	14 (de los cuales 6 dan salida PWM)
Pines de entrada analógica	6
Corriente DC por I/O Pin	40 mA
Corriente DC para el pin 3.3V	50 mA
Memoria Flash	32 KB (ATmega328) 0.5 KB usados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de reloj	16 MHz

Tabla 1. Características generales Arduino UNO.

2.4 SOFTWARE DE DESARROLLO ARDUINO:

Arduino nos proporciona un software liviano y consistente en un entorno de desarrollo integrado (IDE), el cual podemos descargar de manera gratuita en el sitio web oficial de Arduino, este software nos da las herramientas básicas que necesitamos para programar, subir y depurar nuestro código a la placa.

El IDE de Arduino consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Esta plataforma de desarrollo cuenta con su propio lenguaje de programación, el cual está basado en los lenguajes C/C++, por lo que soporta todas las funciones de C, incluidas algunas de C++.

En la Figura 13 se muestra la interfaz principal del software de Arduino con las funciones que llevan a cabo algunos de sus componentes.

Describiendo alguno de los componentes principales tenemos:

- Verificar Archivo: Con el cual podremos compilar el programa que hayamos creado dentro del editor de código, antes de ser subido a nuestra placa Arduino. Si existe un error en nuestro código nos aparecerá en la parte de abajo, donde se encuentra la Consola de errores.
- Cargar programa: De este modo, podremos subir nuestro programa a la placa Arduino, una vez verificado y depurado de errores.
- Nuevo archivo y Abrir archivo: Ambos con el mismo objetivo, comenzar un nuevo proyecto, o abrir uno existente.
- Monitor Serie: Nos permite establecer comunicación con nuestra placa, enviar y recibir datos, a través del puerto serial.

Dentro del editor de código, encontramos, de manera predeterminada, dos funciones del programa al abrir:

- Void Setup () { }: Esta función se ejecuta solo una vez al inicio de cualquier programa, aquí es donde iniciamos nuestras variables globales, también podemos abrir canales de comunicación (puerto serie, Ethernet, etc.), donde podemos cargar la configuración y el estado inicial de nuestro proyecto.
- Void Loop () { }: Aquí tenemos el cuerpo principal de nuestro programa, en el cual se ejecuta un bucle infinito de nuestro código.

Independientemente de estas funciones predeterminadas, nosotros podemos ir insertando muchas más, haciendo uso de librerías, ciclos, clases, los cuales se explicaran en el apartado de Desarrollo del proyecto.

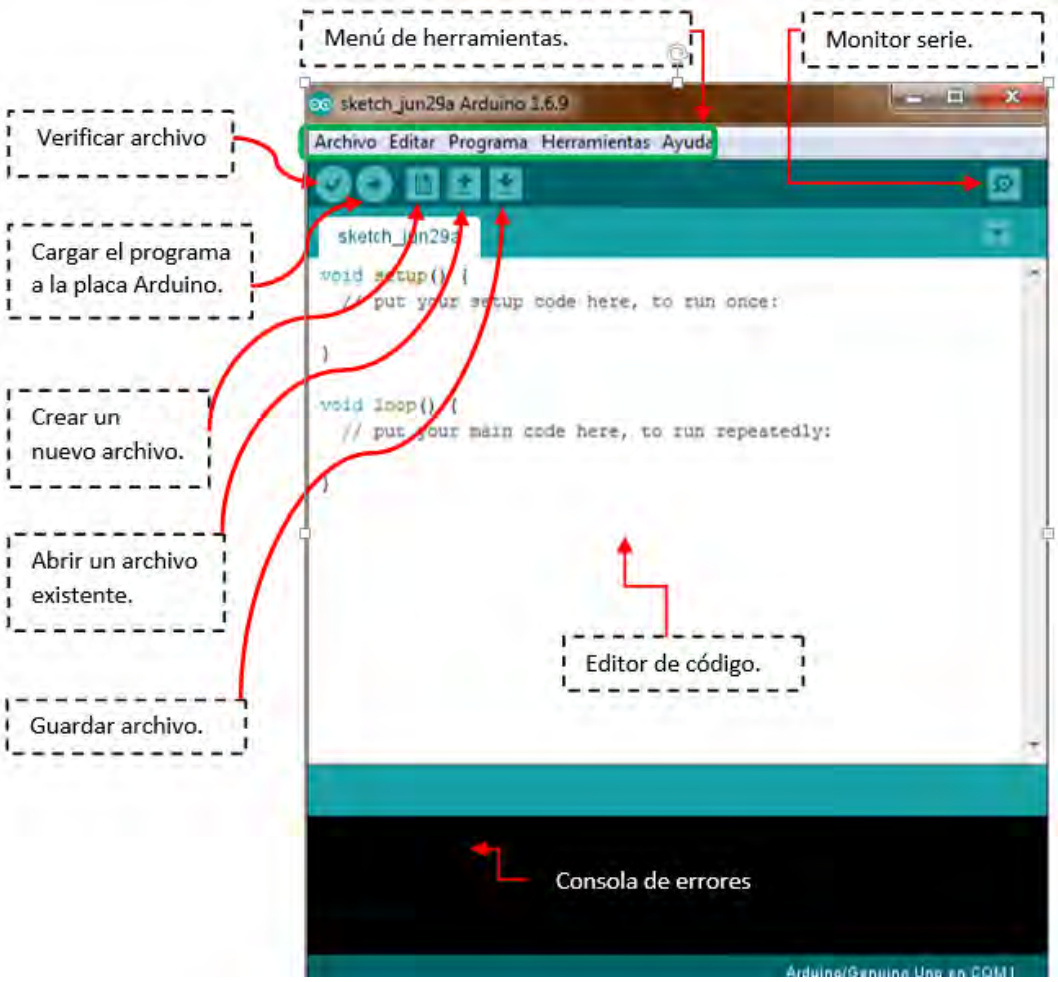

Figura 13. Entorno de desarrollo Arduino

2.5 Zigbee [8]:

Es un estándar que define un conjunto de protocolos de alto nivel, creado para utilizarse en radiodifusión digital de bajo consumo, basado en el estándar IEEE 802.15.4 de redes inalámbricas de área personal.

Diseñado para aplicaciones domóticas, así como aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y la optimización en el uso de energía. Opera en las bandas de 868 MHz, 915 MHz y 2.4 MHz, pudiendo trasmitir datos hasta de 250Kbps.

Una de las desventajas de este protocolo es que no permite un gran ancho de banda, pero sus módulos requieren un mínimo uso de energía, lo cual es una gran ventaja en aquellos nodos remotos que son alimentados por medio de baterías. Fue diseñado bajo las siguientes características:

- Bajo consumo de energía, ideal en el uso con baterías.
- Bajo costo de instalación y mantenimiento.
- Alcance de 50 metros aproximadamente.
- Velocidad de transmisión menor a 250 Kbps.

Zigbee, es un protocolo abierto con fines no comerciales, los principales módulos certificados por Zigbee Alliance (compañías que mantienen el estándar público) que engloban las funcionalidades del protocolo, son los módulos Xbee, los cuales se explicaran más adelante.

Una red Zigbee, básicamente se forma de 3 tipos de elementos, un dispositivo Coordinador, uno o varios dispositivos Routers y dispositivos finales (end points).

Coordinador: Es el elemento de la red, el cual tiene la función de formar una red, responsable de establecer el canal de comunicaciones y del identificador de red para toda la red. Una vez que se han establecido estos parámetros, el coordinador puede ahora si formar la red, permitiendo que los otros elementos, los routers y dispositivos finales puedan unirse a él. Al final, el coordinador hace función de router, participando en el enrutado de paquetes y ser origen o destinatario de información.

Routers: Estos dispositivos crean y mantienen información en la red para determinar la mejor ruta para la trasmisión de paquetes.

Dispositivos Finales: Estos dispositivos no tienen la capacidad de enrutar paquetes, interactúan a través de un nodo padre, el cual puede ser el Coordinador o un Router, lo que quiere decir que no puede enviar información directamente a otro dispositivo final.

En la Figura 14 podemos ver el esquema tradicional donde interactúan los dispositivos en el modo programado, coordinador, router o dispositivo final:

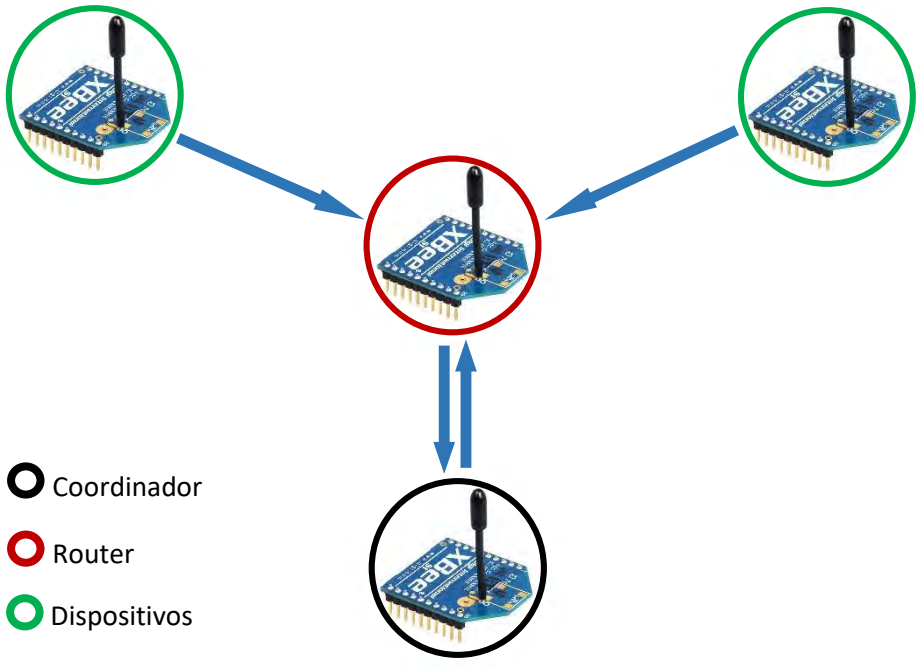

Figura 14. Esquema tradicional dispositivos Xbee.

2.6 XBEE:

Otro de los elementos que usaremos durante este proyecto son los módulos Xbee [9]. Son pequeños chips propiedad de la empresa Digi, basado en protocolos Zigbee, capaces de establecer comunicación inalámbrica entre ellos, y otros dispositivos compatibles. Estos módulos trabajan bajo el protocolo de red IEEE 802.15.4, para crear redes Punto a Punto (Peer –to-Peer) o redes Punto a Multipunto (Point-to-Multipoint). Estos dispositivos fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Las áreas donde son más usados son:

- Entretenimiento en casa y control: Iluminación inteligente, control avanzado de temperatura, seguridad, música, etc.
- Hogar: Sensores de agua, electrodomésticos inteligentes, etc.
- Servicios Móviles: Monitoreo y control móvil, tele-asistencia, seguridad y control de acceso móvil, etc.
- Plantas Industriales: Control de procesos, gestión de ventajas, gestión ambiental, gestión de energía, control de dispositivos industriales, etc.

2.6.1 Series Xbee:

Los módulos Xbee se clasifican en un conjunto de familias o series con características y capacidades diferentes dependiendo del tipo de proyecto donde son empleados:

- Series 1: La serie de Xbee más fácil de usar y configurar. Son los más recomendables para comenzar a trabajar con esos módulos. Son ideales para comunicación punto a punto. Estos módulos no son compatibles con las Series 2, solo pueden usarse con otros dispositivos similares Serie 1.
- Xbee Znet 2.5: Son un poco más complejos que los módulos de Serie 1. Tiene mayores capacidades, y la forma de configuración es un poco más laborioso. Pueden funcionar en

modo transparente o a través de comandos API. No son compatibles con Series 1, únicamente con módulos de la misma serie.

- ZB: Es el módulo Xbee Znet 2.5 actualizado, con nuevo firmware.
- 2B: Módulos más recientes, con mejoras en hardware, con potencia aumentada. Funcionan con el firmware del módulo ZB.
- 900 MHz: Más que una Serie, es una familia de módulos. Estos módulos pueden funcionar con dos firmwares distintos, el firmware DigiMesh, y el firmware Pont-to-Point. Son módulos tipo plug and play.
- XSC: Básicamente módulos de 900MHz, pero sin alta velocidad de datos por el alcance que logran (los módulos 900MHz tienen una velocidad de datos de aproximadamente 156 Kbps, mientras que los módulos XSC de 10 Kbps

En la Figura 15 podemos ver algunos de los modelos mencionados:

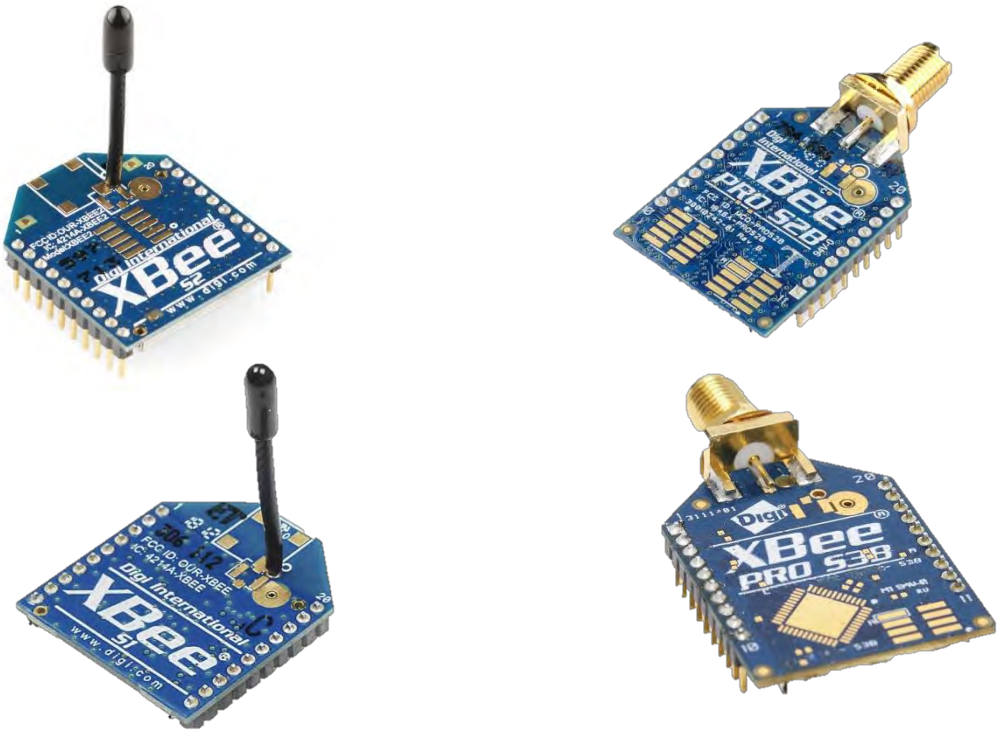

Figura 15. Ejemplos módulos Xbee.

2.6.2 Tipos de Antenas:

En los módulos Xbee, dependiendo del modelo usado, podemos encontrar diferentes tipos de antenas integradas:

- Chip: Antena en forma de chip pequeño.
- Wire: Pequeño cable que sobresale.
- u.FL: Conector pequeño para incrustar una antena propia.
- RPSMA: Conector de mayor tamaño para conectar una antena propia.

El modelo que usaremos en el presente proyecto es el Xbee Series 1 de MaxStream, el cual describiremos con más detalle:

2.6.3 Modos de operación:

Los módulos Xbee, pueden funcionar en cinco diferentes modos, según se requiera:

- **Modo Transparente:** En el modo transparente, todo lo que ingresa por el pin 3 (data in), se guarda en el buffer de entrada y luego es transmitido, y todo lo que ingresa como paquete RF, es guardado en el buffer de salida y luego enviado por el pin 2 (data out). Es el modo por defecto en los módulos Xbee.
- **Modo AT (de comandos):** En este modo, es posible ingresar comando AT al módulo Xbee para configurar parámetros cono la dirección de destino, entre otras cosas. Para entrar a este modo, se utiliza el Hyperterminal o el software X-CTU.
- **Modo Recibir/Transmitir:** En este modo, el módulo recibe un paquete RF a través de la antena (modo Recibir), o cuando se envía información serial al buffer del pin 3 (UART Data in), que luego se transmite (modo Transmitir. Es posible enviar información por Unicast y Broadcast.
- **Sleep Mode (Modo de bajo consumo):** En este modo de operación, el módulo entra en un estado de bajo consumo de energía, dependiendo de la configuración en la que se encuentre y del voltaje de alimentación utilizado.
- **Modo API:** El modo más complejo, permite el uso de frames con cabeceras que aseguran la transmisión de datos, similar al estilo TCP. Este modo extiende el

nivel en el cual la aplicación del cliente puede interactuar con las capacidades de red del módulo.

- Existe un modo en el cual, el módulo no se encuentra en ninguno de los modos anteriores, a este estado se le llama IDLE.

2.7 XCTU:

XCTU [10]; es una aplicación multiplataforma y gratuita que provee una interfaz gráfica que nos permitirá la administración, configuración, e interacción con los módulos de radiofrecuencia con los que estaremos trabajando, los Xbee.

Este software nos permite descubrir los módulos conectados a nuestra PC, así como configurarlos por medio de las interfaces que ofrece, usando comandos API y AT. Nos ofrece herramientas integradas para realizar tareas, como la recuperación de módulos dañados, reseteo y actualización de firmware de los dispositivos.

En la Figura 16 podemos ver la interfaz gráfica principal del programa XCTU:

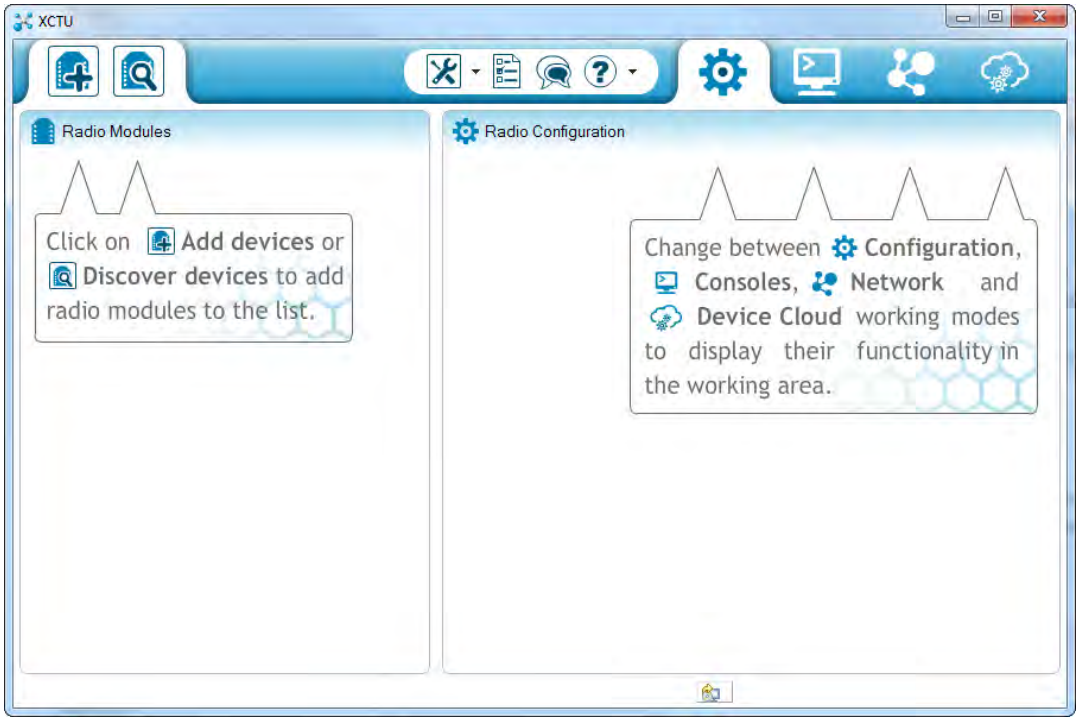

Figura 16. Interfaz gráfica software XCTU.

2.8 Xbee Shield y Adaptador Xbee USB:

Dos herramientas importantes que usaremos en el presente proyecto son los siguientes:

- Adaptador Xbee USB: Consiste en una tarjeta equipada con pines para el módulo Xbee compatible con Arduino, el cual nos ayudará para conectar a nuestra PC el módulo y poder programarlo vía USB, o de igual forma la comunicación con cualquier microcontrolador que no tenga USB a través de puerto serial.

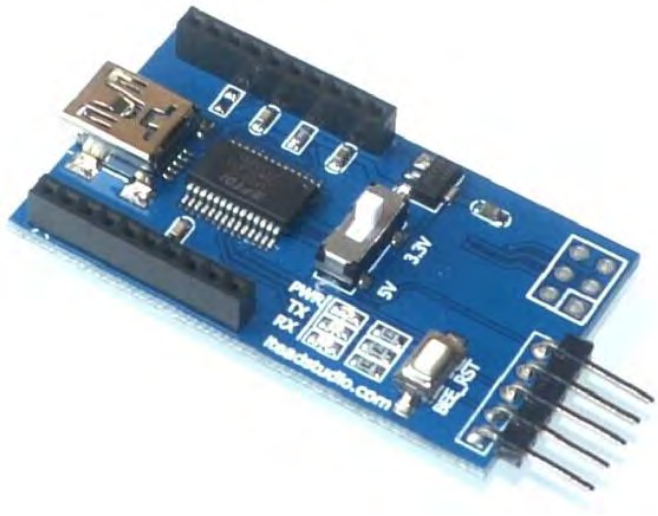

Figura 17. Adaptador Xbee USB.

A este dispositivo no le configuraremos nada en específico, solo debemos asegurar que el interruptor de voltaje esté en 3.3V antes de conectar el módulo Xbee, de tenerlo en 5V, podríamos dañar nuestros radios al conectarlo a nuestra PC para su configuración.

- Xbee Shield: El Xbee Shield es un módulo que simplifica la tarea de interconectar los módulos Xbee con las placas Arduino. De esta manera podemos usar los pines disponibles para conectar el módulo Xbee y establecer comunicación. Esta herramienta integra un indicador de RSSI. Una configuración importante en el hardware de este dispositivo se mostrará más adelante.

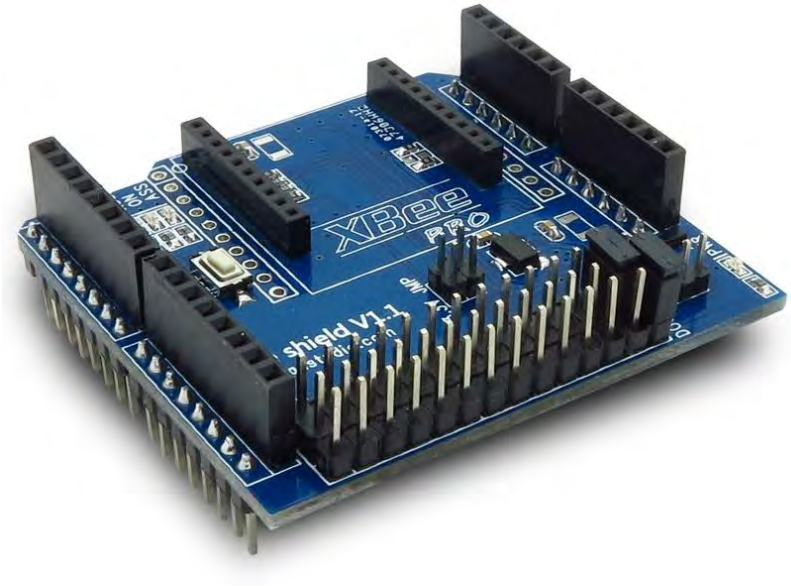

Figura 18. Xbee Shield.

2.9 Lenguaje de Programación en Arduino

La plataforma Arduino UNO nos será de gran ayuda en la realización de este proyecto. Ya explicamos sus características así como el entorno de desarrollo propio que utiliza para su programación.

Ahora, necesitamos conocer a fondo el lenguaje de programación con el cual trabajamos. Conoceremos las funciones, ciclos, variables, etc., etc., así como la manera de usarlos durante nuestro proyecto.

2.9.1 Estructura de programa:

Como indicamos con anterioridad, la estructura básica de un programa en el software Arduino se compone de dos bloques. Estás dos partes son fundamentales, los cuales contienen declaraciones e instrucciones:

```
void setup() {

        // put your setup code here, to run once:

        }

void loop() {

        // put your main code here, to run repeatedly:

}
```

2.9.2 Funciones:

Una función es un bloque de código identificado por un nombre, el cual es ejecutado cuando la función es llamada y puede retornar un valor. Las funciones pueden tomar parámetros que pueden modificar la manera en que funcionan. Las funciones que podemos crear pueden ser escritas para realizar tareas repetitivas y ser llamadas cuando se requiera, de esta manera reducir el tamaño del programa. Cuando se declara una función incluye el tipo de datos que devuelve, por ejemplo ‘int’ nos devolverá un dato numérico de tipo entero. Cuando la función se programa para no devolver ningún valor, se le coloca la palabra ‘void’, lo que significa ‘función vacía’. Una vez que hemos declarado el tipo de dato, escribimos el nombre de nuestra función, y de ser preciso, los parámetros que devolverá al ser llamada.

2.9.3 Variables:

Las variables forman parte importante en la estructura del código. Una variable es una manera de almacenar un valor para su uso posterior en el programa. A las variables se les pueden aplicar valores que pueden cambiar el tiempo que se requiera, contrario a las constantes, que tienen un valor fijo. Las variables deben ser declaradas antes de poder ser usadas, pueden ser declaradas al inicio del programa, convirtiéndose así en variables globales, las cuales pueden ser usadas en cualquier bloque del programa, o dentro de las funciones, siendo así, variables locales, pudiendo ser utilizadas únicamente en el bloque declarado, es posible declarar variables dentro de un bloque con bucles como ‘if’, ‘for’, etc.,; se les asigna un nombre, se define el tipo, por ejemplo ‘int’, aplicado a un valor numérico entero o ‘float’, un valor de coma flotante, etc. De ser necesario se les aplica un valor inicial.

Algunas de los tipos que podemos emplear en variables más comunes son:

- Byte: El cual puede almacenar valores numéricos de 8 bits sin decimales, con un rango entre 0 a 255.
- Int: Puede almacenar valores numéricos de 16 bits sin decimales, con un rango entre 32767 a -32767.
- Long: Puede almacenar valores numéricos enteros de 32 bits sin decimales dentro del rango de 2147483647 a -2147483647.
- Float: Conocido como de punto flotante o coma flotante, aplicable a números con decimales, en un rango entre 3.4028235E+38 y -3.4028235E+38.
- Array: Un array es un conjunto de valores, a los cuales podemos tener acceso por medio de números índices. Conociendo el nombre del arreglo y el número de índice, podemos tener acceso a cualquier valor dentro del conjunto.
- Bolean:
- Char:
- Word:
- Double:
- String:
- Entre otros…

2.9.4 Operadores Aritméticos:

Los operadores aritméticos que podemos utilizar a la hora de programar son los ya conocidos: suma, resta, multiplicación y división. Haciendo uso de las variables, previamente presentadas, y los operadores aritméticos, podremos realizar cálculos necesarios dentro de nuestro programa.

2.9.5 Asignaciones Compuestas:

Este tipo de asignación combina un operador aritmético con una variable asignada y son utilizadas en bucles:

- X ++ (igual que: X=X+1)
- X - - (igual que: X=X-1)
- X += Y (igual que: X=X+Y)
- X -= Y (igual que: X=X-Y)
- X *= Y (igual que: X=X*Y)
- X /= Y (igual que X=X/Y)

2.9.6 Operadores de Comparación:

Este tipo de operaciones se usan regularmente en las estructuras condicionales como ‘if’ para poder comparar valores y así poder tomar decisiones:

- X == Y (X es igual a Y)
- X != Y (X es diferente de Y)
- X < Y (X es menor que Y)
- X > Y (X es mayor que Y)
- X <= Y (X es menor o igual que Y)
- X >= Y (X es mayor o igual que Y)

2.9.7 Operadores Lógicos:

Este tipo de operadores se usan para comparar expresiones y devolver un valor TRUE o FALSE, dependiendo del tipo de operador. Se usan, al igual que los operadores de comparación, con estructuras condicionales 'if':

- AND → && (devuelve TRUE cuando ambas condiciones son verdaderas)
- OR → || (devuelve TRUE cuando al menos una de las dos condiciones es verdadera)
- NOT → ! (Niega el valor de la expresión)

2.9.8 Estructuras de Control:

Los ciclos sirven para realizar acciones repetitivas hasta llegar a una condición para detenerse, o de lo contrario se convierte en un bucle reiterativo. Algunos de los más usados son los siguientes:

- IF: Se usa para probar si una determinada condición se ha alcanzado, y ejecutar una serie de operaciones si es verdad, si es falso, el programa salta las operaciones de la estructura.
- IF…ELSE: Se usa cuando analizamos la idea de "si algo no se cumple, entonces se hace esto".
- FOR: Usamos 'for' para repetir un bloque de sentencias un determinado número de veces. Cada vez que se ejecutan las instrucciones del bucle, se vuelve a evaluar la condición. Si la condición no se cumple, el bucle termina.
- WHILE: El bucle 'while', es de ejecución continua mientras se cumpla una expresión que se coloca en la cabecera del bucle. La variable de prueba, tiene que cambiar para que el bucle pueda detenerse o salir de él.

3 CAPITULO: DESARROLLO DEL PROYECTO:

3.1 METODOLOGÍA:

En el siguiente capítulo, determinaremos el proceso llevado a cabo para la estimación de la distancia entre nuestros módulos de radiofrecuencia Xbee S1 usando la información de la intensidad de la señal (RSSI); así mismo se definirá el proceso de investigación de los dispositivos que integran el proyecto. Por último, se presenta el sistema final, y las pruebas correspondientes para demostrar su funcionamiento usando una metodología de muestreo del trabajo; es necesario comparar medidas reales con medidas estimadas para corroborar los resultados obtenidos.

3.1.1 Intensidad de Señal (RSSI):

El indicador de fuerza de la señal recibida (RSSI) [11], es una medida de la potencia de una señal recibida en un dispositivo empleado en redes inalámbricas como puede ser un radio o un móvil; podría decirse que es medida que indica que tan bien un dispositivo puede “escuchar” una señal de un emisor, el cual puede ser un Access Point, un Router, un radio, etc.

Se tiende a confundir el RSSI con dBm, ambos son unidades de medida que representan la intensidad de la señal; la diferencia es que el RSSI es un índice relativo, mientras que el valor dBm es un número absoluto que representa los niveles de potencia en mili watts (mW).

En resumen, el valor RSSI, mientras más se acerque a un valor de 0, es mejor o ideal, pero prácticamente imposible de alcanzar en un entorno real. En la Tabla 2 podemos ver la relación de valores RSSI:

Intensidad de Señal	Valoración
0	Señal ideal, muy difícil de conseguir en la práctica en un entorno real.

-40 a -60	Señal idónea, con tasas de transferencia estables.
-60	Enlace estable.
-70	Enlace normal, medianamente bueno, muy inestable en un clima de lluvia y viento.
-80	Señal mínima aceptable para poder establecer conexión, se producen cortes de comunicación, perdidas de datos, etc.

Tabla 2. Rango de Valores RSSI.

3.1.2 Técnica de estimación de distancia usando RSSI:

En la elaboración de la técnica para la estimación de la distancia entre los módulos Xbee usando la intensidad de la señal (RSSI), se analizaron trabajos de investigación relacionados al tema. Las técnicas más comunes, como se mencionó anteriormente, incluyen la Trilateración, y posicionamiento por Bluetooth. Algunas de estas técnicas, obtienen valores de medición con un margen de error considerable. Esto es normal, ya que al usar el RSSI, éste toma valores inestables dependiendo del espacio y medio ambiente donde se emplea. Entonces, un valor inestable de intensidad de la señal, da como resultado, mediciones poco aproximadas a los valores reales.

Entre los fenómenos que influyen en la propagación de datos [12], y que afectan a que la transmisión y recepción de información se vean aún más afectada, están:

- Atenuación por espacio libre: Diferencia que existe en la intensidad de la señal cuando existe entre dispositivos transmisor y receptor, una línea de vista, en función de la longitud de onda (λ), y la distancia (d), que los separa.
- Atenuación por multitrayecto: Aquí influye mucho el medio donde se realizan las mediciones, los objetos, así como personas, es donde se dan los fenómenos de propagación como reflexión, refracción y dispersión.
- Perdida por transmisión: Representa la pérdida de la señal causada por los obstáculos que atraviesan los rayos que se propagan.

Mediante el estudio de los trabajos de investigación mencionados anteriormente, se comprueba que la información utilizada para la localización y posicionamiento de un dispositivo que permanece inmóvil, varía constantemente. En los dispositivos móviles, se agrava un poco la situación, ya que, además de influir el medio donde se desarrolla la dinámica, los valores inestables del RSSI y el movimiento de los dispositivos, hacen que la tarea de realizar estimaciones de distancia sea más complicada.

El sistema propuesto, para el posicionamiento relativo entre dispositivos en movimiento con línea de vista, busca una alternativa más viable en el uso de la intensidad de la señal (RSSI) para la estimación de la distancia en la que se encuentra un módulo Xbee transmisor de uno receptor. Aun teniendo la ventaja de línea de vista, existen otros factores que afectan la transmisión y recepción de datos, y la potencia de señal sufre los fenómenos de pérdida por propagación, ruido y otras interferencias.

Existen ecuaciones que establecen que la pérdida por propagación de la señal recibida por un dispositivo receptor, a una distancia d_i del dispositivo transmisor, como vemos en la ecuación siguiente:

$$P_L(d_i)[dB] = P_L(d_0)[dB] + 10nlog_{10}\frac{d_i}{d_0} \qquad (1)$$

Donde $P_L(d_0)$ es la pérdida de propagación a una distancia de referencia conocida d_0 (generalmente = 1 metro). Donde n es una constante que depende del medio que existe entre dispositivos. Nosotros trabajaremos en espacio libre, con línea de vista, aunque, en un entorno de pruebas real, la propagación de la señal se verá afectada por los fenómenos mencionados con anterioridad (reflexión, difracción, dispersión). Entonces, de la ecuación anterior, tenemos que:

$$n = \frac{P_L(d_i) - P_L(d_0)}{10log_{10}\frac{d_i}{d_0}} \qquad (2)$$

Esta ecuación nos permite la estimación de la constante n a partir de la medición de la potencia entre los nodos transmisor y receptor, así como la distancia de separación.

Los valores que n podría tomar, están ligados a un concepto denominado Path Loss (pérdida por trayecto o atenuación por trayectoria), el cual consiste en la reducción de la densidad de potencia (atenuación) de las ondas electromagnéticas mientras se propaga a través del espacio.

Empíricamente, la relación entre la potencia promedio recibida y la distancia, está determinada por la expresión:

$$P_r \propto d^{-\gamma} \qquad (3)$$

De la ecuación anterior, γ es llamada exponente de perdida por trayectoria (Path Loss Exponent). En la Tabla 3 podemos ver los valores que puede adquirir, dependiendo del medio o entorno donde se aplique el sistema o donde se lleven a cabo las pruebas y estimaciones.

Medio Ambiente, Entorno	Path Loss Exponent
Espacio Libre (Free Space)	2
Área Urbana (Urban Area)	2.7 a 3.5
Área Suburbana (Suburban Area)	3 a 5
Interiores	1.6 a 1.8

Tabla 3. Valores del exponente de pérdida por trayectoria (Path Loss Exponent).

Obtenemos un modelo de estimación de distancia, propuesto por la compañía Texas Instruments. La ecuación es la siguiente:

$$RSSI = -(10 * n) log_{10}(d) - A \qquad (4)$$

Donde tenemos que, RSSI es el indicador de la intensidad de señal de un disposivo en dBm (decibelio-milivatios). La n representa la constante de pérdida de la señal mencionada previamente; en nuestro caso, trabajaremos con línea de vista y espacio libre, entonces este valor será una constante igual a 2.

En la ecuación, A es una referencia de la intensidad de la señal recibida; el valor del RSSI obtenido, cuando la separación entre el transmisor y el recetor es de un metro de distancia.

Finalmente, de la ecuación (4), despejamos la variable de distancia d, la cual nos servirá más adelante en la programación de nuestra placa Arduino, para la estimación de la distancia entre nuestros dispositivos. Obtenemos la siguiente ecuación:

$$d = 10^{\frac{RSSI-A}{10*n}} \qquad (5)$$

3.1.3 Configuración Módulos Xbee:

Para la configuración de los módulos Xbee, usaremos el software XCTU, los módulos Xbee Series 1, un Adaptador Xbee USB, un adaptador mini USB y una PC. Al acoplar el módulo Xbee al Adaptador Xbee, quedaría de la siguiente manera, listo para conectar y configurar en una PC:

Figura 19. Conexión módulo Xbee y Adaptador Xbee.

A partir de ahora, comenzaremos a la asignación de parámetros a los radios, dependiendo de las funciones llevaran a cabo, primero comenzaremos con el radio transmisor, el cual tendrá la función de Coordinador, y el segundo módulo será configurado como radio receptor, o Dispositivo Final (End Device).

Módulo Transmisor: Conectamos nuestro primer radio (transmisor) al Adaptador Xbee USB (indispensable para poder configurar el módulo), y lo conectamos mientras el cable mini USB es conectado a nuestra PC. Posteriormente, abrir el programa XCTU, el cual hace que la interfaz del programa esté disponible como se muestra en la Figura 20.

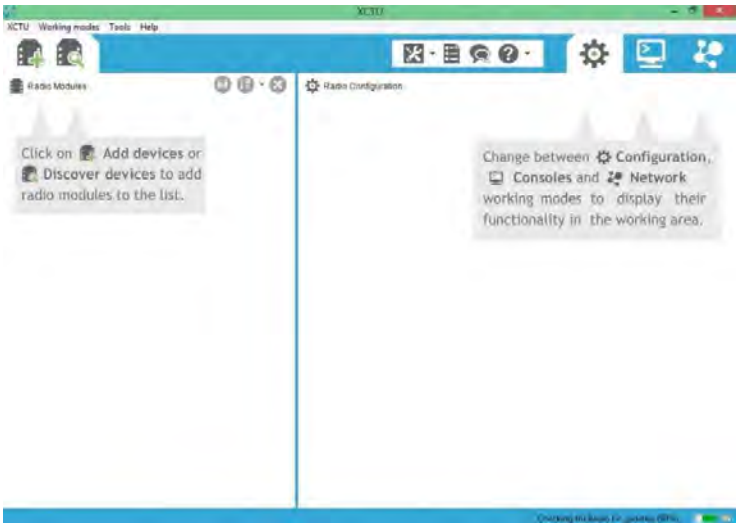

Figura 20. Interfaz principal Software XCTU

Después de que la pantalla principal está disponible, se procede a realizar la búsqueda de modulo(s) conectado(s) a nuestra PC. En la Figura 21, podemos ver el icono de "Búsqueda de dispositivos".

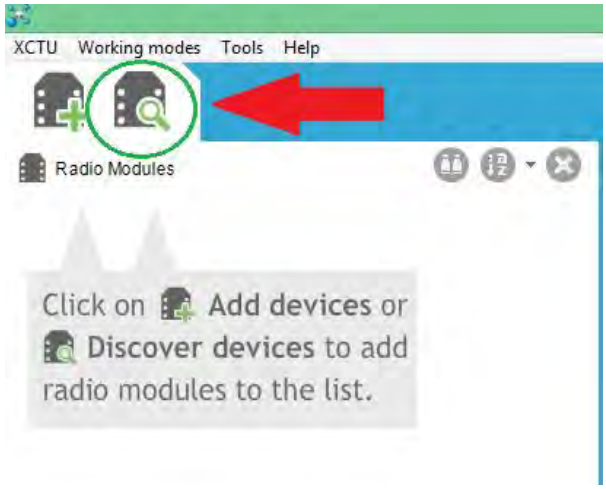

Figura 21. Botón "Búsqueda de Dispositivos".

Una vez que hayamos pulsado el botón de Búsqueda de Dispositivos, nos aparecerá la ventana de la Figura 22. Aquí, nos indicará que puertos COM podemos escanear, de los cuales, alguno estará asignado a nuestro módulo Xbee de manera automática.

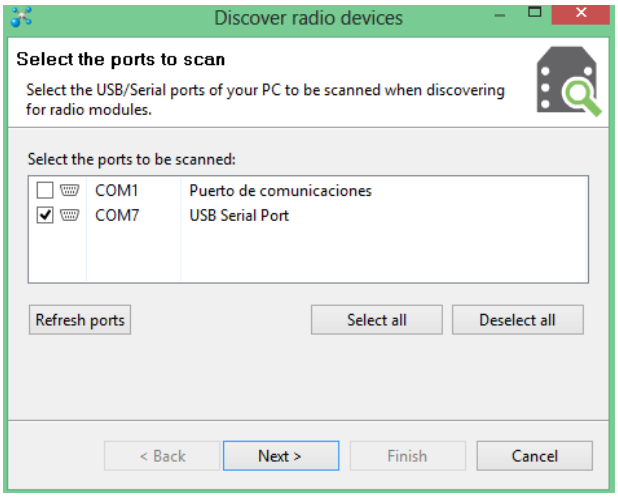

Figura 22. Seleccionar Puertos de para Escanear Módulos.

Después de que hayamos seleccionado el puerto asignado a nuestro módulo y darle siguiente, nos aparecerá la ventana de parámetros por default. Cuando conectamos por primera vez los módulos Xbee, estos parámetros vienen configurados de manera predeterminada, y de esta forma podremos encontrar los radios. Se recomienda no cambiar estos parámetros a menos de que se necesite realizar una búsqueda con parámetros específicos.

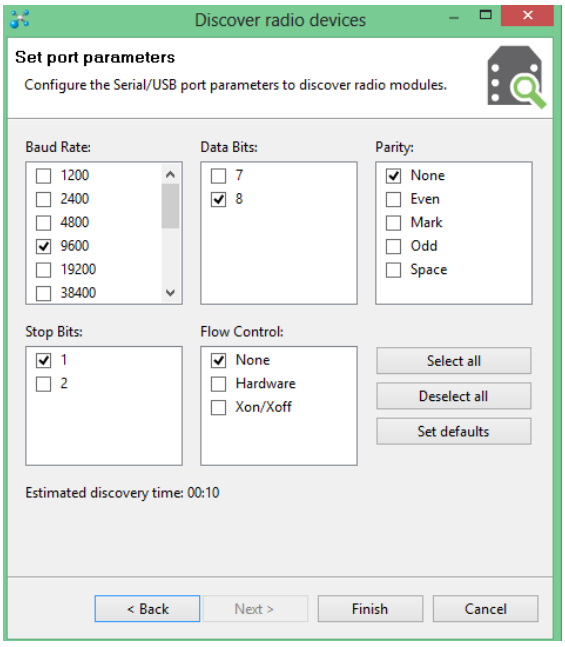

Figura 23. Parámetros predeterminados para la búsqueda de módulos Xbee.

La siguiente ventana en aparecer, nos muestra como el software realiza la búsqueda de los módulos una vez realizado los pasos anteriores.

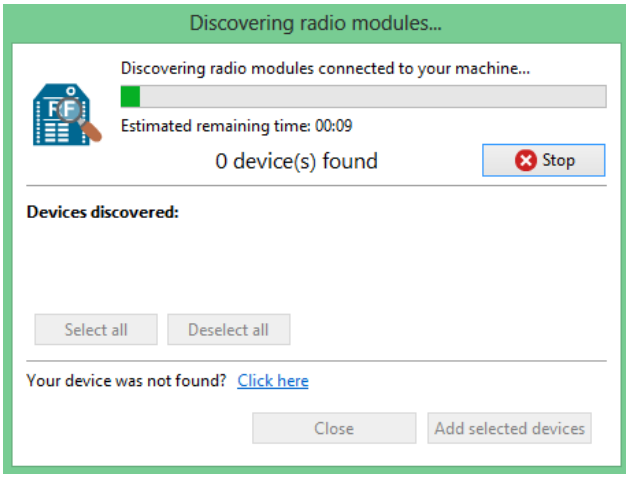

Figura 24. Búsqueda de módulos Xbee.

Una vez finalizada la búsqueda de dispositivos, el software nos enlistará los módulos encontrados. En nuestro caso aparece el único conectado para su configuración. Añadimos el dispositivo y seguimos.

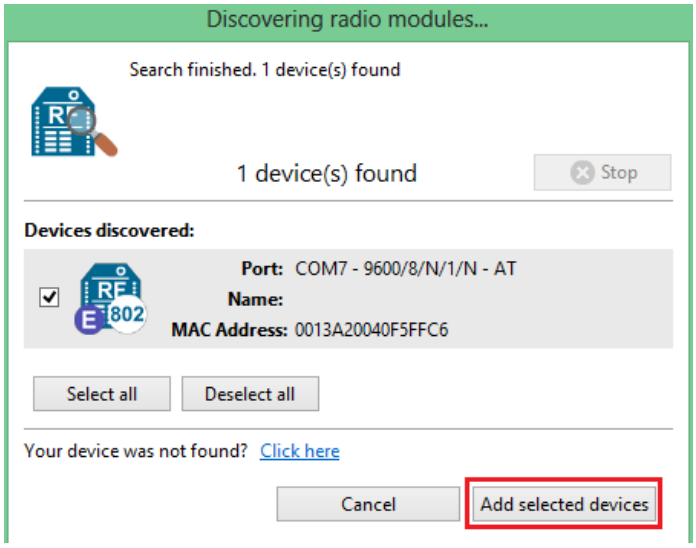

Figura 25. Módulos Xbee encontrados.

Después de que añadimos el módulo encontrado, se tendrá una nueva ventana como la mostrada en la Figura 26. Una vez realizado todo lo anterior, estará el primer módulo listo para ser configurado con los parámetros necesarios.

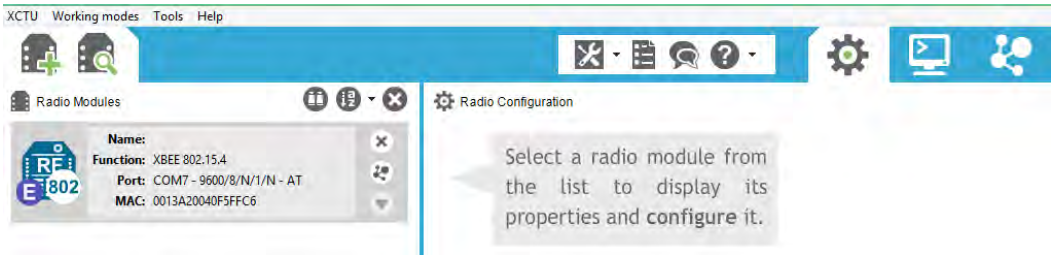

Figura 26. Módulo listo para ser configurado.

Seleccionamos el módulo Xbee a configurar y aparecerá la ventana mostrada en la Figura 27, donde se pueden apreciar los parámetros configurables del radio. Se puede apreciar una lista extensa, de los cuales, solo configuraremos las necesarias para el desarrollo del proyecto.

Como se mencionó anteriormente, el primer módulo será configurado como radio transmisor y/o tendrá funciones de Coordinador.

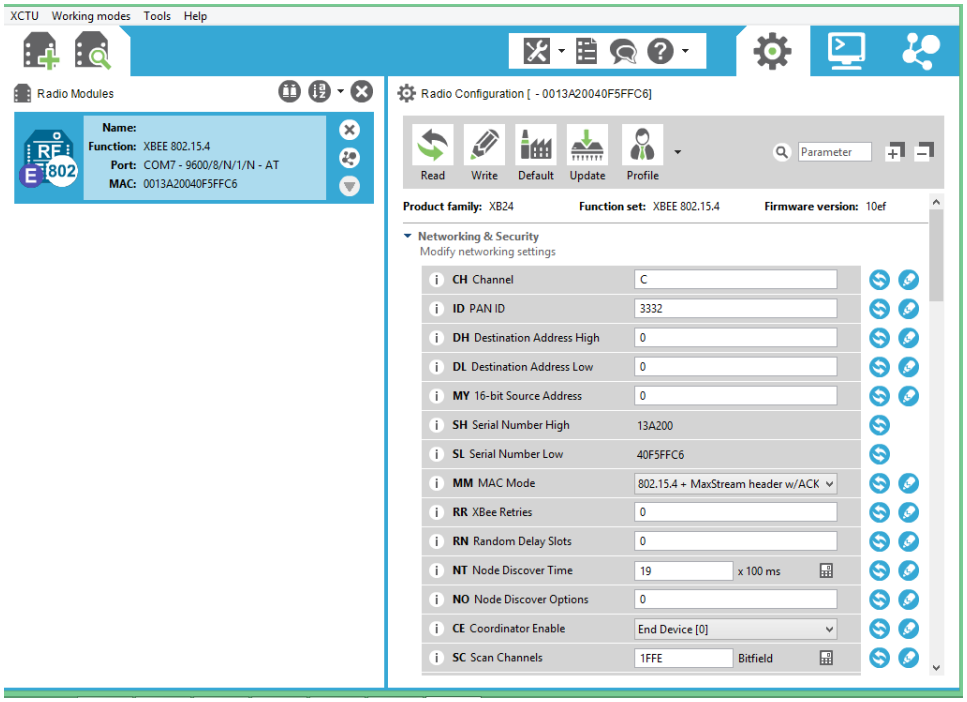

Figura 27. Parámetros disponibles en la configuración del módulo Xbee.

En la Tabla 4, podemos ver los parámetros que necesitamos configurar en el radio Coordinador, una breve explicación de cada uno y los valores que asignamos para que funcione como se necesita.

Parámetro	Descripción	Valores Asignados.
Canal (Channel)	Nos permite asignar el canal que será utilizado para transmitir o recibir datos entre los módulos.	**E** (podemos asignar el valor que queramos en el canal, mientras sea el mismo en todos los módulos, si no, no se establecerá comunicación).
PAN ID	Permite identificar a una red de Área Personal en particular	**B000** (los valores de este parámetro también pueden ser como lo decidamos, pero de igual manera, no debe ser distinto en los módulos).
MY	Este parámetro define una dirección de origen de 16 bits. Para deshabilitar la dirección de 16 bits y habilitar la dirección de 64 bits, ponemos como valor 0xFFFF.	**AAA0**
A2, Coordinator Association	En este parámetro definimos a nuestro módulo como Coordinador.	**1** (con este valor, el coordinador examinará el canal para operar en el que está disponible).
AP, API Enable	Establece el modo de operación del módulo, cuando AP = 0, opera en modo transparente, cuando AP = 1, opera en	**AP** = 2, esta es otra opción de modo API, ya que el modo API 1, no permite la conexión entre módulos.

	moto API (Application Programming Interface)	

Tabla 4. Parámetros configurados en el módulo Coordinador.

Ahora que tenemos los parámetros necesarios para configurar el primer módulo, los seleccionamos en los campos disponibles del programa XCTU como aparece en la Figura 28 y después le damos Write (escribir) para guardar la configuración en el dispositivo.

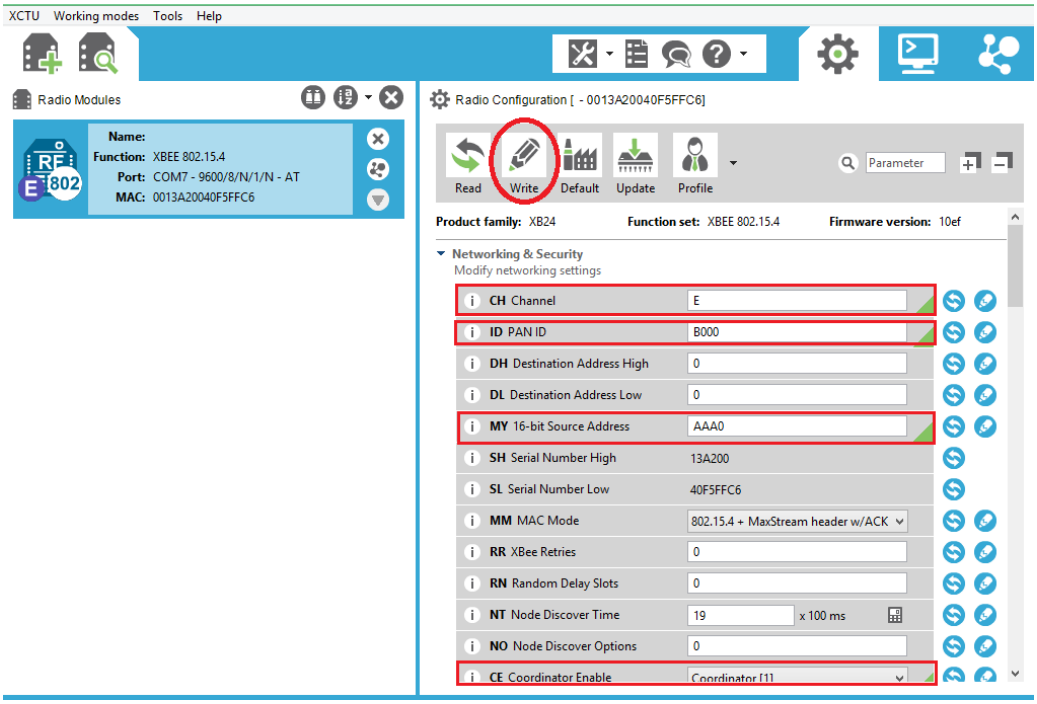

Figura 28. Parámetros configurados.

Al finalizar la configuración del módulo, el programa XCTU nos muestra que ahora el radio está configurado en modo Coordinador:

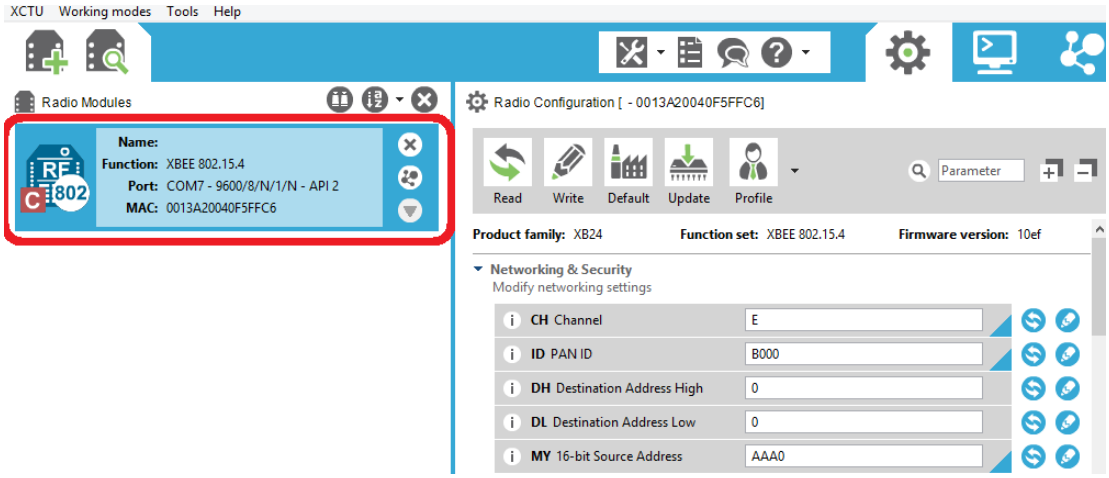

Figura 29. Módulo configurado como Coordinador.

Ahora podemos configurar el segundo módulo Xbee, que como se mencionó anteriormente, tendrá la función de Dispositivo Final (End Device). Los parámetros que se configuraran en este dispositivo se enlistan en la Tabla 5.

Parámetro	Valores Asignados.
Canal (Channel)	E
PAN ID	B000
MY	**AAA1**
AP, API Enable	**AP** = 2

Tabla 5. Parámetros configurados en módulo Dispositivo Final.

Una vez ingresados y escritos todos los parámetros correspondientes, el programa nos indicará que el módulo ahora es un Dispositivo Final.

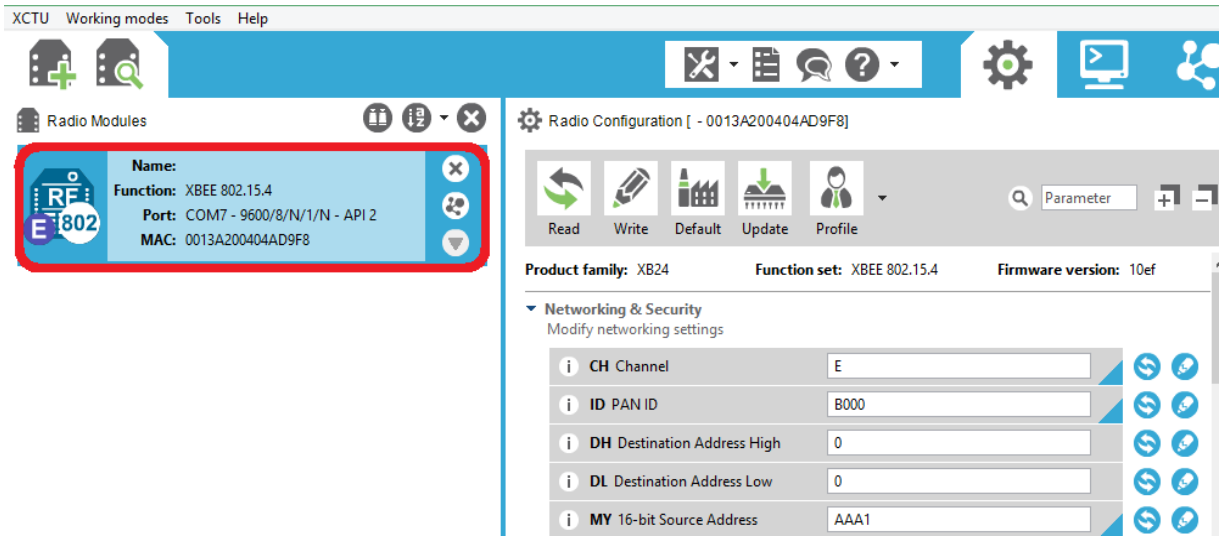

Figura 30. Módulo Xbee modo Dispositivo Final configurado.

Ahora que ya tenemos ambos módulos que usaremos en el proyecto, correctamente configurados, seguimos con la configuración y programación de nuestra placa Arduino, para luego acoplar los dispositivos y comenzar con la realización de pruebas que veremos más adelante.

3.1.4 Configuración y Código Arduino

Al inicio del presente proyecto, mencionamos el dispositivo que nos ayudará a la realización de este trabajo, la placa Arduino UNO. Ahora veremos cómo configurar y programar nuestra placa para después integrar los dispositivos restantes.

Figura 31. Arduino Uno y conector USB.

Se comenta que, se buscó optimizar por completo el programa realizado, incluyendo lo necesario para realizar a cabo las pruebas correspondientes, de esta manera, se consiguió crear un pequeño programa que describiremos a continuación.

En el primer bloque de código tenemos algunos parámetros iniciales. Ingresamos dos librerías necesarias, la librería Xbee, y la librería SoftwareSerial. Después creamos una variable que asignaremos al parámetro SoftwareSerial, el cual llamaremos **rxtx**, y asignamos los pines (2, 3) necesarios para la comunicación. En la Figura 32 podemos ver el bloque de código descrito.

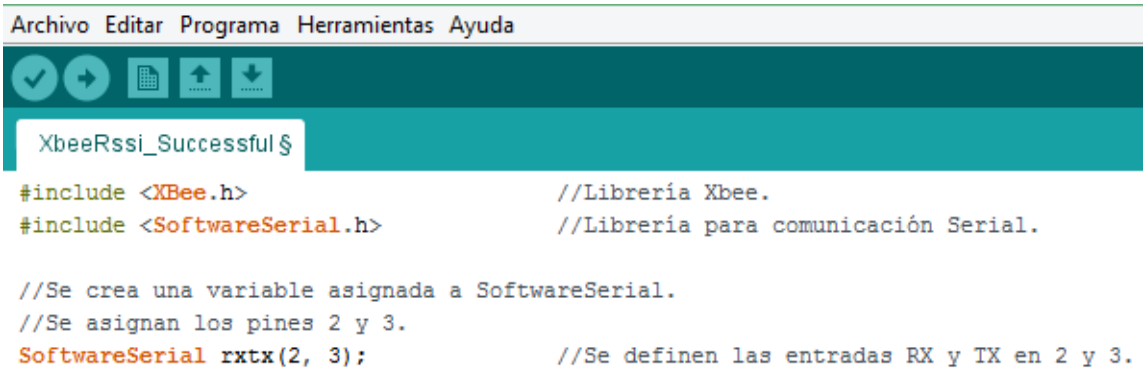

Figura 32. Bloque de código 1, comunicación Xbee-Arduino.

En el segundo bloque de código, comenzamos creando un objeto de tipo Xbee, el cual nos provee de funciones necesarias para el envío y recepción de paquetes. La API de Xbee ofrece una clase para los paquetes de recepción; se hereda de un objeto llamada XbeeResponse, el cual usaremos en nuestro código para la recepción de mensajes a nuestro Arduino vía Xbee. Después inicializamos el objeto Response para las respuestas que pensamos manejar. Para terminar este bloque, creamos 2 variables y una constante de tipo Entero (int); la primera (rssi) almacenara los valores del RSSI, la segunda (n), es la constante de propagación de señal, la cual tendrá el valor por defecto igual a 2, la última variable, de tipo entero, almacenara los valores de distancia recibidos. En la Figura 33 podemos ver el bloque descrito.

```
//Creamos un objeto Xbee. Este objeto provee
//funciones para enviar y recibir paquetes.
XBee xbee=XBee();

//Objeto XbeeResponse para recepción de paquetes.
XBeeResponse response = XBeeResponse();

//Inicializamos el objeto Response
Rx16Response rx16 = Rx16Response();


int rssi = 0;  //Variable que almacena los valores recibidos RSSI
int n = 2;     //Constante de propagación de la señal.
int d = 0;     //Varianble que almacena el valor de la distancia.
```

Figura 33. Bloque de código 2, objetos y variables.

El tercer bloque contiene la función Setup; aquí iniciamos la comunicación serial y asignamos la velocidad en 9600 bps, después asignamos a nuestra variable “rxtx”, creada anteriormente, la velocidad de transmisión, para después asignarla a la comunicación serial del Xbee, la cual también tendrá una velocidad de 9600 bps. En la Figura 34 podemos ver el bloque.

```
void setup()
{
  Serial.begin(9600);    // Iniciar comunicación serial.

  rxtx.begin(9600);      // Asignamos a variable SoftwareSerial velocidad de transmisión.

  xbee.setSerial(rxtx);  // Comunicación serial Xbee.
}
```

Figura 34. Bloque de código 3, Void Setup.

Por último, tenemos el bloque de código que contiene la función Loop; aquí queremos ver constantemente los paquetes que vienen a nuestro dispositivo, entonces usamos la función readPacket() justo comenzando el bucle de nuestro programa. Después incluimos un condicional IF, en el cual revisamos que el objeto Response esté disponible, dentro de este condicional, aplicamos otro condicional, el cual una vez cumplido el primer IF, revisará que el paquete sea del tipo especificado (16 bits).

```
void loop()
{
  xbee.readPacket(100);                          // Lectura de paquetes.
  if (xbee.getResponse().isAvailable())          // Condicional que verifique que el Response este disponible.
  {
    if(xbee.getResponse().getApiId() == RX_16_RESPONSE)     // Revisar que el paquete sea del tipo correcto RX16.
    {
```

Figura 35. Bloque de código 4. Void Loop.

Ahora que se ha verificado la entrada del paquete y tipo, procedemos a obtener y leer el paquete; para ello utilizamos el xbee.getResponse. Una vez que hayamos obtenido y leído el paquete, podemos obtener el valor de la intensidad de la señal (RSSI) usando rx16.getRssi, valores que asignaremos a nuestra variable creada con anterioridad (int rssi). En la Figura 36 podemos ver el código descrito.

```
if (xbee.getResponse().getApiId() == RX_16_RESPONSE)
{
  xbee.getResponse().getRx16Response(rx16);          // Se obtiene y lee el paquete.
  rssi = rx16.getRssi();                             // Se ontiene el valor de la intensidad de la señal.
```

Figura 36. Bloque de código 5, Void Loop, obtención valor RSSI.

En este punto ya tenemos el valor de la intensidad de la señal (RSSI). Ahora incluimos la fórmula (5) obtenida en la sección previa para realizar la estimación de la distancia entre nuestros

dispositivos. Después de estimar la distancia, imprimimos en pantalla del monitor serie el valor del RSSI, así como la distancia en milímetros. Obtener el valor en milímetros, nos ayuda a observar que tan precisa es la estimación de la distancia. Agregamos al final un separador, para poder identificar como los valores varían, agregando un segundo entre cada mensaje.

```
      d = pow (10, ((rssi-0.1)/ (10 * n)));          // Fórmula cálculo distancia entre dispositivos.
      Serial.print("Valor RSSI: ");                  // Mostramos en monitor serie el texto "Valor RSSI: ".
      Serial.println(rssi);                          // Se imprime el valor obtenido del RSSI.
      Serial.print("Distancia : ");                  // Imprimimos en monitor serie la palabra "Distancia: ".

      Serial.print(d);                               // Imprimimos el valor obtenido de la distancia.
      Serial.println(" mm");                         // Mostramos que la distancia se mide en milímetros.
      Serial.println("-------------------------");   // Indicamos una separación entre mensajes.
      delay(1000);                                   // Se retrasa la lectura del siguiente mensaje a 1 segundo.
    }

    }
  }
}
```

Figura 37. Bloque de código 6. Void Loop, Estimación de distancia.

3.1.5 Integración de dispositivos

En este proyecto, empleamos dos módulos Xbee (configuración mostrada con anterioridad), de los cuales, uno actuará como Coordinador para el envío de datos, y el otro como dispositivo final (End Device) para la recepción de datos. Un módulo Xbee se montará a una placa Arduino Uno (código mostrado y detallado anteriormente) mediante un Xbee Shield.

Antes de montar nuestro Xbee Shield a la placa Arduino Uno, debemos realizar una configuración simple, pero muy importante; cabe destacar, que éste mínimo detalle, es por el cual, muchos proyectos similares fallan al momento de querer adquirir los valores RSSI e imprimirlos por monitor serie.

En la Figura 38, podemos ver el hardware del Xbee Shield, y en la Tabla 6, las partes que lo conforman:

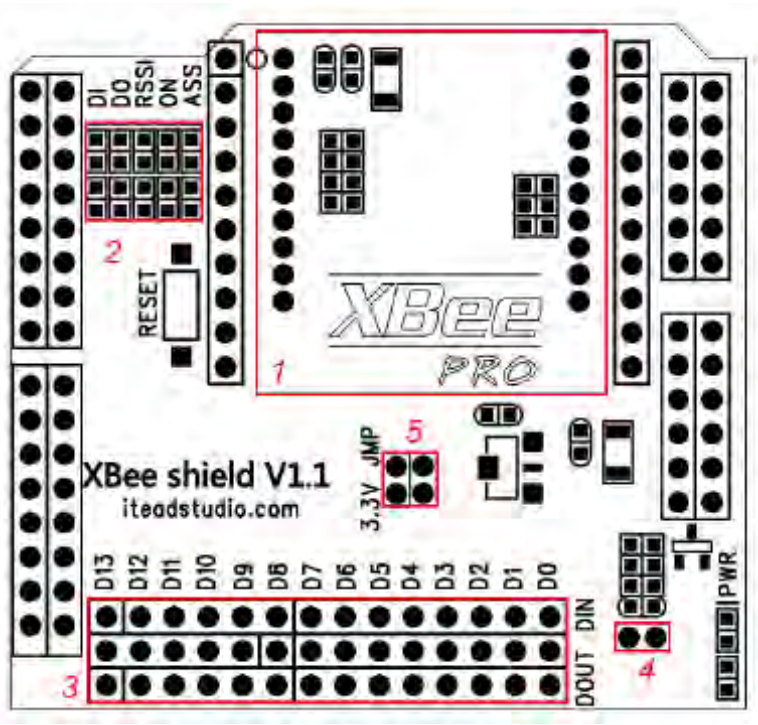

Figura 38. Hardware Xbee Shield.

Zona	Descripción
1	Socket para Xbee.
2	LED Indicadores.
3	Pines de comunicación Serial.
4	Jumper Wireless Arduino.
5	Jumper de voltaje.

Tabla 6. Partes del hardware Xbee Shield.

En la sección 3, donde encontramos los pines de comunicación serial, debemos configurar los dos jumpers que establecen conexión Xbee con los pines digitales de Arduino, estos se encuentran en DIN D1 y DOUT D0 respectivamente.

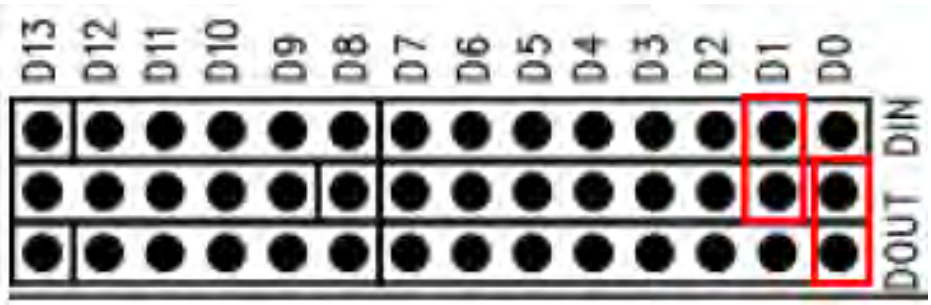

Figura 39. Configuración predeterminada de jumpers en Zona 3.

En nuestro código Arduino, ya cargado en nuestra placa Arduino UNO, establecimos que estos jumpers deben estar asignados a DIN D3, y DOUT D2 respectivamente, por lo que debe quedar de la siguiente manera:

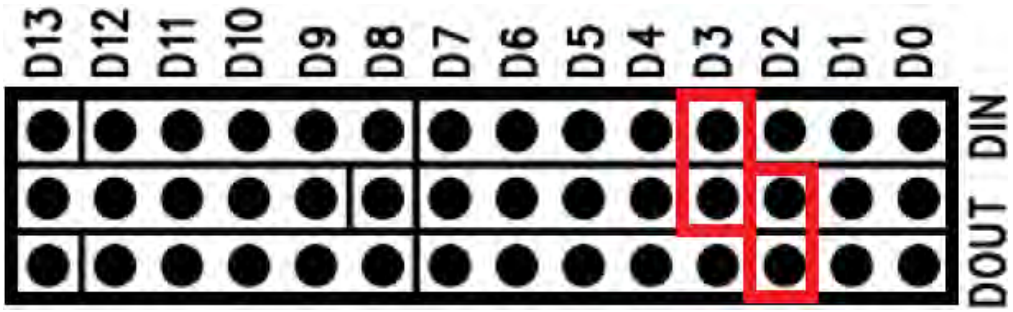

Figura 40. Configuración de jumpers correspondiente a nuestro programa en Arduino UNO.

El procedimiento anterior es de suma importancia realizar y no olvidar. Ahora, podemos integrar a nuestra placa Arduino UNO, con el código de nuestro programa ya cargado, el Xbee Shield, y por último, nuestro módulo Xbee configurado anteriormente como Dispositivo Final, quedando todo como se puede ver a continuación en la Figura 41 (líneas de conexión representativas), y Figura 42:

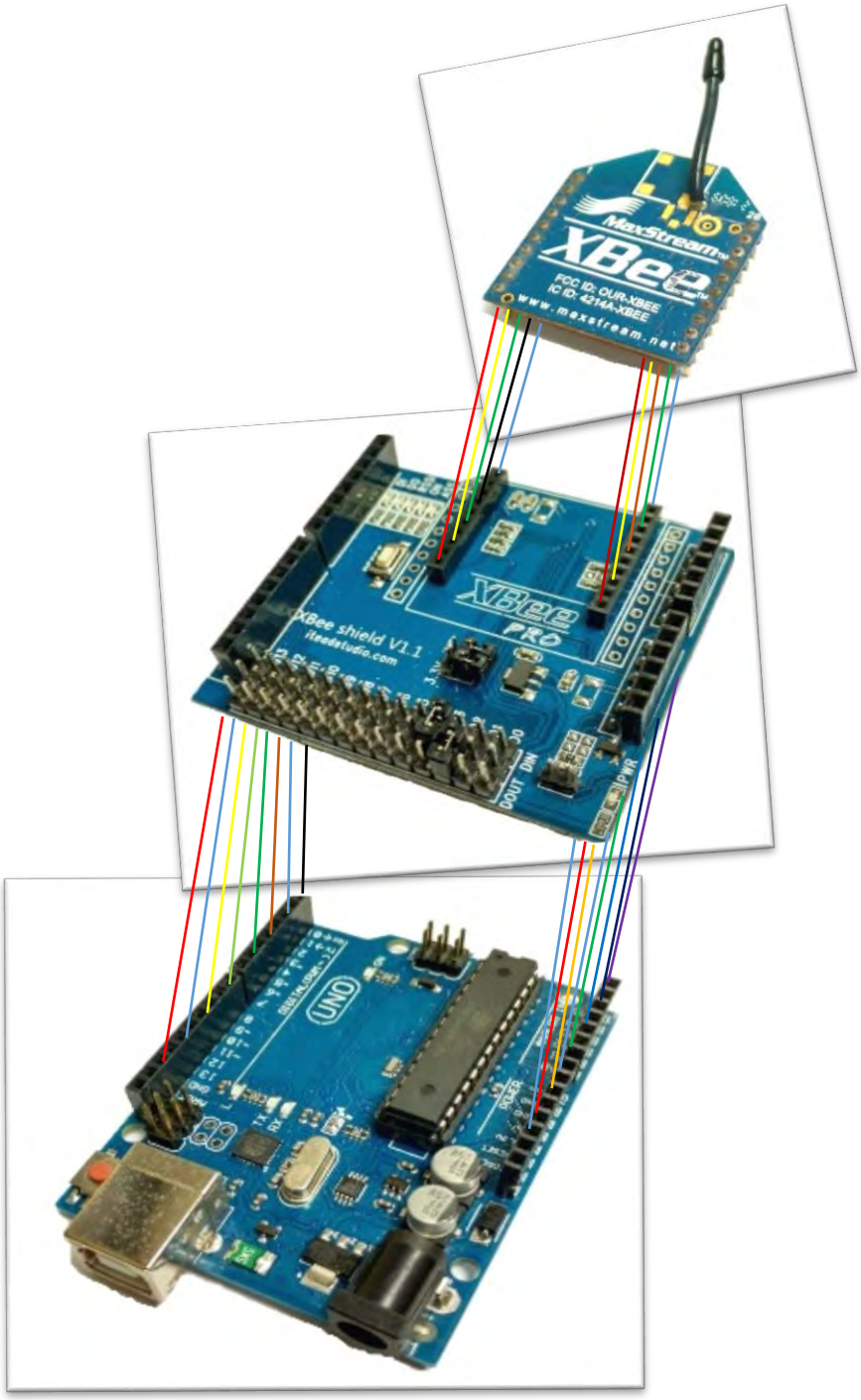

Figura 41. Forma de conexión de componentes.

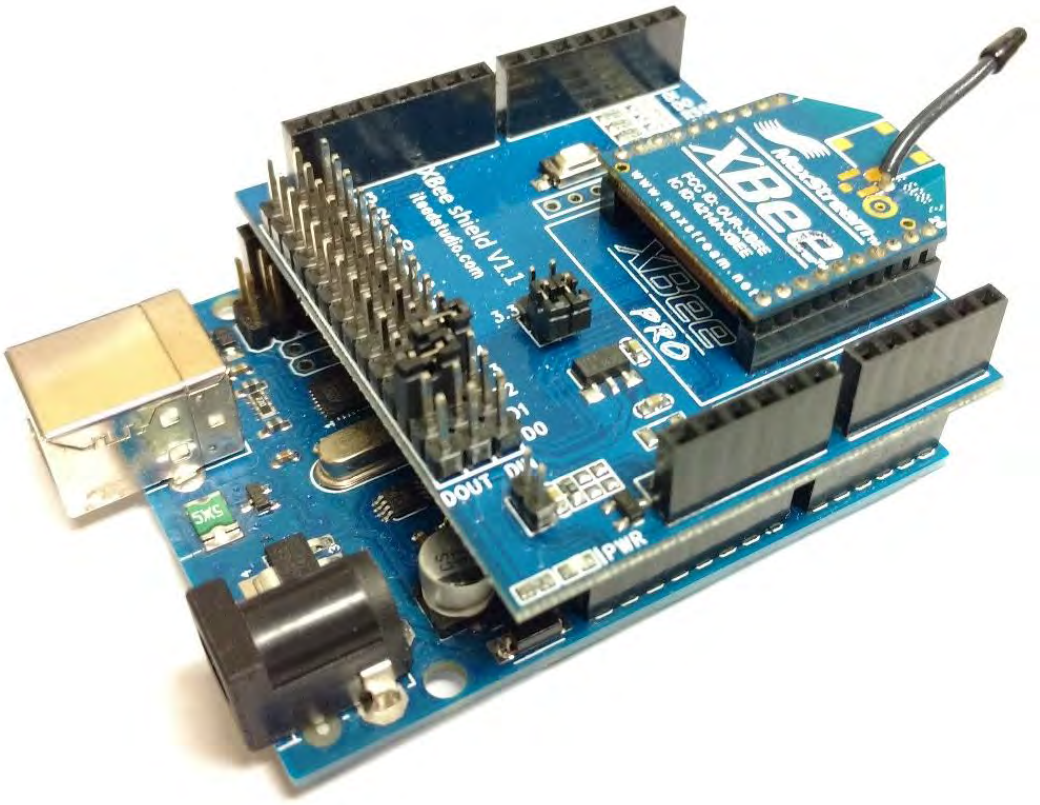

Figura 42. Dispositivos ensamblados.

3.1.6 Pruebas de estimación de distancia:

A partir de ahora, ya tenemos todos los componentes listos y acoplados; podemos realizar las pruebas correspondientes para la estimación de distancia entre nuestros dispositivos transmisor (módulo Xbee Coordinador) y receptor (módulo Xbee Dispositivo Final).

Primero crearemos un paquete en el programa XCTU, y lo cargaremos a nuestro módulo Xbee Coordinador. Los pasos para realizar este proceso, se muestra a continuación en una serie de figuras.

Aquí mostramos nuestro módulo ya conectado a nuestra PC, y configurado en el software XCTU.

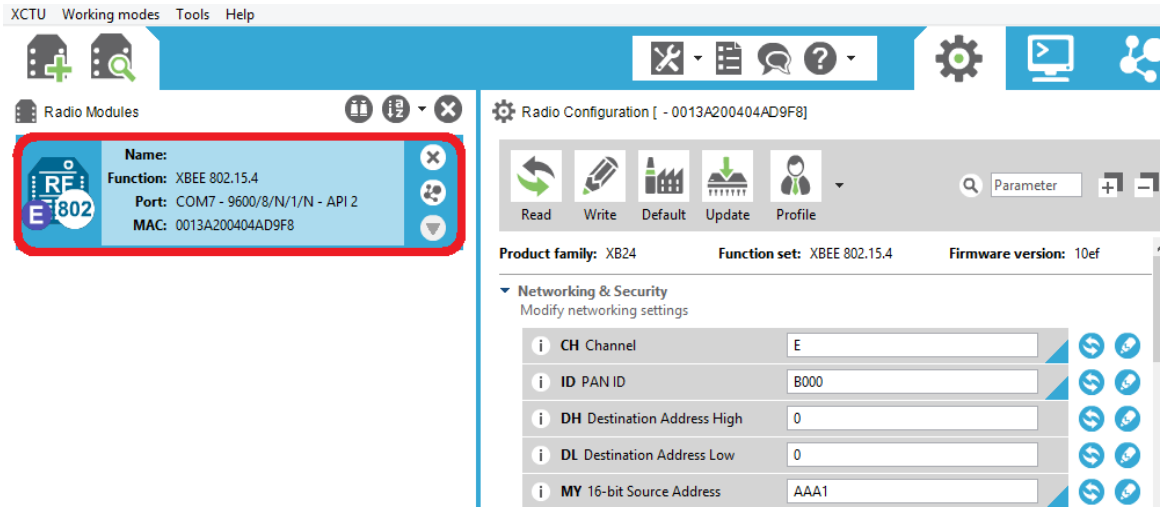

Figura 43. Módulo Xbee Coordinador en software XCTU.

Estando en esta ventana, seleccionamos el botón de Consola, y nos aparece otra ventana con otras características, damos clic en Cerrar Conexión Serial con nuestro módulo Xbee, en la Figura 44 mostramos encerrado en rojo estas instrucciones, y encerrado en verde, la opción para crear un nuevo paquete:

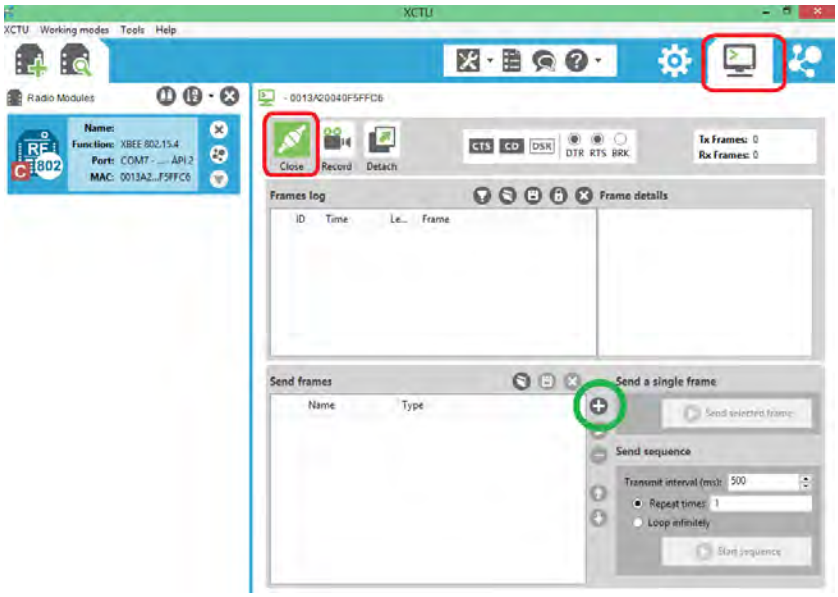

Figura 44. Modo Consola, Cerrar Conexión Serial con Módulo Xbee y añadir nuevo paquete.

Después de dar seleccionar “crear un nuevo paquete”, nos aparece otra ventana, donde nombraremos el paquete, y damos clic a la opción “crear paquete usando la herramienta Generador de Paquetes”:

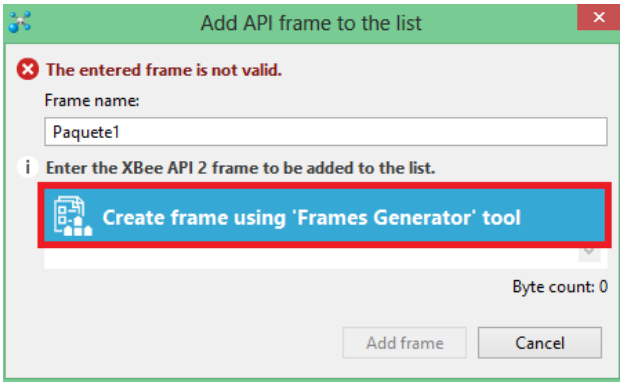

Figura 45. Añadir Paquete 1. Nombrar paquete.

Cuando seleccionamos la opción de crear paquete usando la herramienta “generador de paquetes” obtenemos otra ventana, donde configuraremos algunos parámetros del paquete que estamos creando:

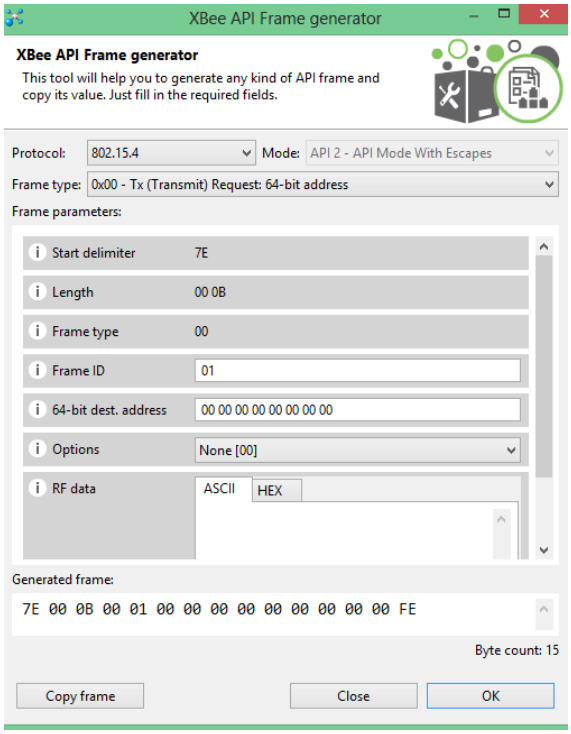

Figura 46. Añadir Paquete 2, Herramienta “generador de paquetes”.

Estando en esa ventana, comenzamos cambiando la opción de “Tipo de Paquete”, desplegamos las opciones y seleccionamos “Tx (Transmit) Request: 16 bits address”, ya que estamos usando direccionamiento de 16 bits en nuestros módulos Xbee. Después configuramos la opción “16-bit dest. address”, asignamos el valor a “FF FF” para que nuestro módulo transmisor pueda enviar datos hacia nuestro módulo receptor que se encuentra en el mismo canal (si tenemos más Dispositivos Finales, esta opción permite enviar broadcast a todos los módulos que se encuentren en el mismo canal). Por último, ingresamos los datos en la opción “RF data”, donde ingresaremos la información que enviaremos, el cual nos permite ver los datos en formato ASCII y Hexadecimal. En la Figura 47 podemos ver los parámetros ingresados. Después de configurar el paquete, lo añadimos como aparece en la Figura 48.

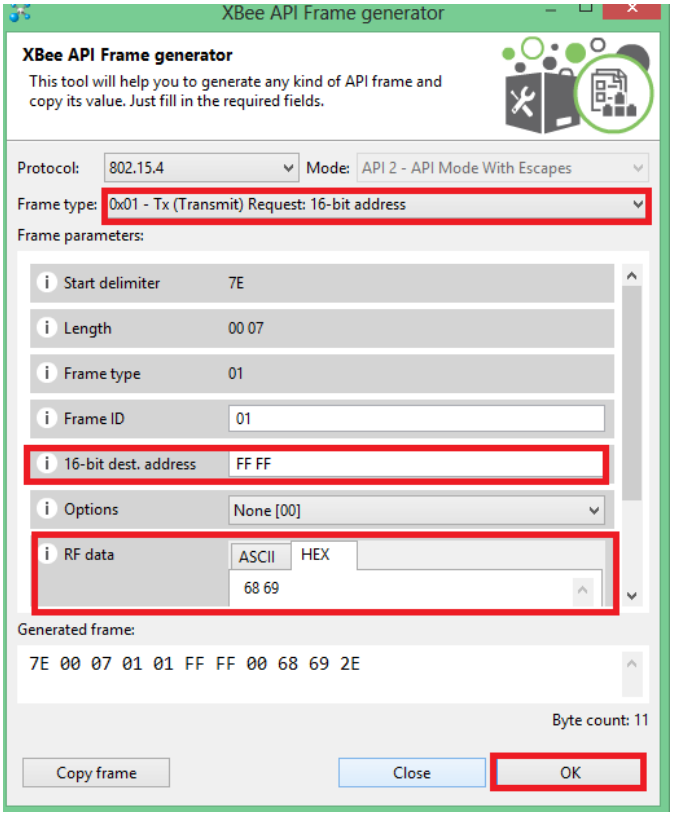

Figura 47. Parámetros configurados en el paquete.

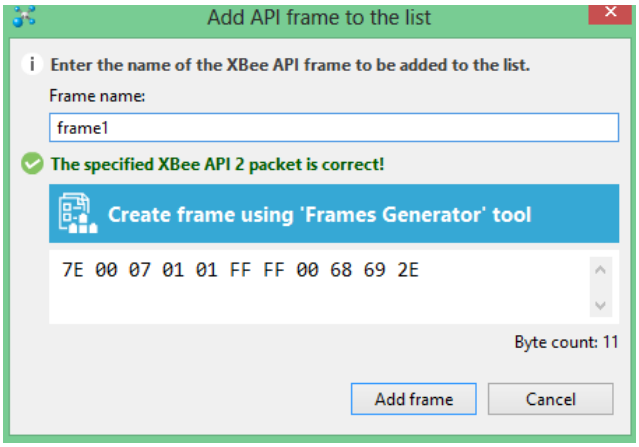

Figura 48. Añadir Frame creado.

En la Tabla 7, podemos ver la estructura del paquete creado.

Iniciar Delimitador	Longitud	Tipo de paquete	ID de paquete	Dirección Dest. 16-bit	Opciones	Datos de RF	Checksum
7E	00 07 (7)	01	01	FF FF	00	68 69	2E

Tabla 7. Campos que componen el paquete creado.

Ahora que tenemos el paquete listo en nuestro módulo Xbee Coordinador, comenzaremos a enviarlo a nuestro módulo Xbee Dispositivo Final, el cual ha sido ensamblado con nuestro Xbee Shield y nuestra Placa Arduino, con el código de nuestro programa cargado, presentado previamente. En el software XCTU, crearemos un loop en el cual estaremos enviando nuestro paquete a un intervalo de 500 ms, para que del otro lado (Dispositivo Final) obtengamos el valor RSSI del paquete recibido. En el cuadro de Registro de Paquetes, podemos ver el mensaje enviado, y seguido obtenemos la respuesta de envió, el cual, al darle clic, podemos ver que ha sido recibido con éxito por el módulo Dispositivo Final.

Figura 49. Módulo Xbee Coordinador transmitiendo.

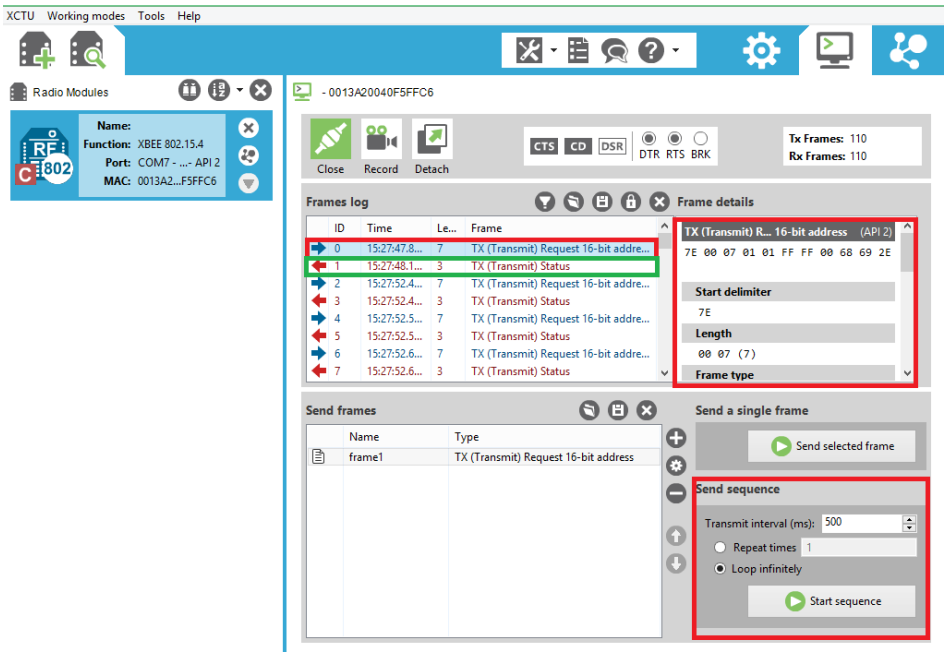

Figura 50. Envío del paquete creado desde módulo Xbee Coordinador hacia Xbee Dispositivo Final.

En el lado de nuestro Dispositivo Final, con nuestro programa cargado con anterioridad, podemos ver en Monitor Serie, los datos recibidos, nuestro código, recibe el paquete y muestra el valor de la intensidad de la señal (RSSI) y después estima la distancia mediante el algoritmo explicado anteriormente, en la Figura 51 vemos nuestro Dispositivo Final recibiendo los datos de nuestro módulo Coordinador.

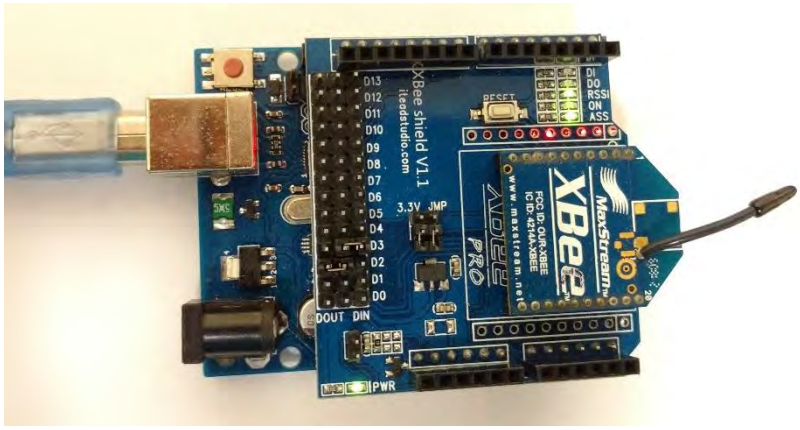

Figura 51. Dispositivo Final recibiendo los datos del módulo Coordinador.

Las pruebas de estimación de la distancia entre nuestros dispositivos, se realizaron en un espacio libre de objetos, comenzando con una distancia exacta de 30 centímetros y finalizando con una distancia de 10 metros de separación. Los resultados obtenidos son los siguientes:

- A una distancia de 30 centímetros: El valor del RSSI se mantiene un poco estable a ésta distancia, proporcionando una distancia con una margen de error pequeño como podemos ver en la Figura 52. En la Figura 53 vemos algunas muestras de las distancias estimadas comparadas con la distancia real.

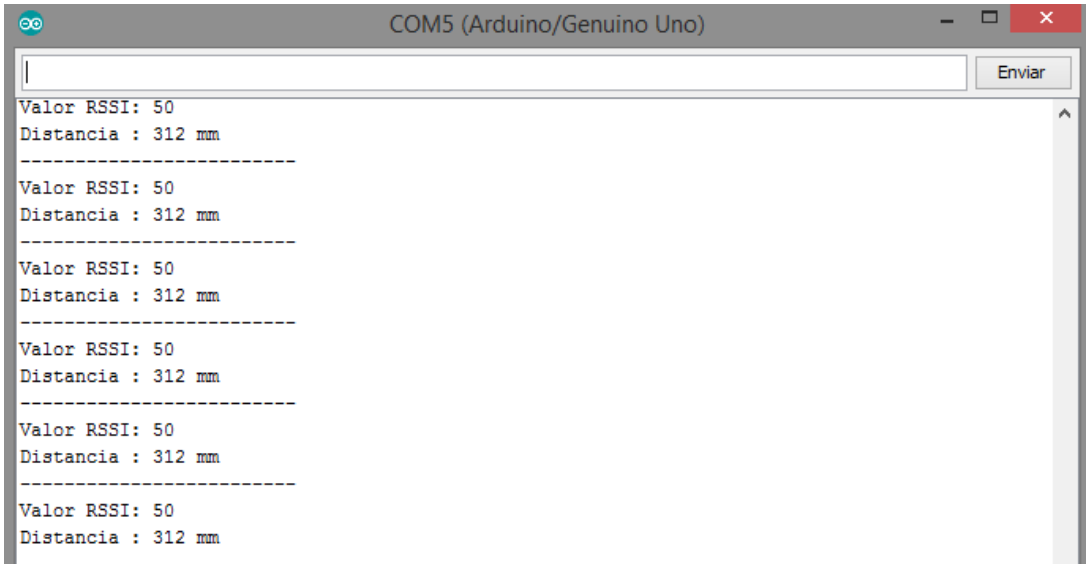

Figura 52. Estimación de distancia 1, 30 centímetros.

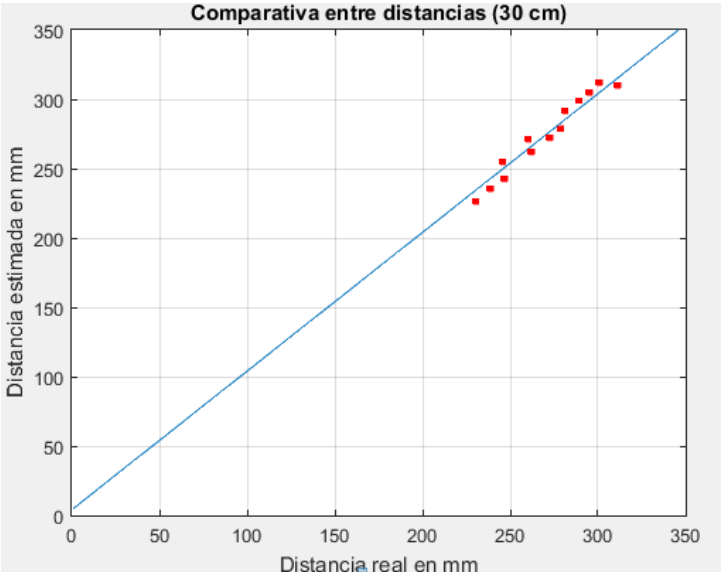

Figura 53. Comparación entre distancia real y estimada 1.

- A una distancia de un metro: A una distancia de 100 centímetros, el valor del RSSI se va poniendo un poco inestable, la estimación de la distancia, presenta un margen de error, la distancia disminuye y aumenta, dependiendo del valor del RSSI.

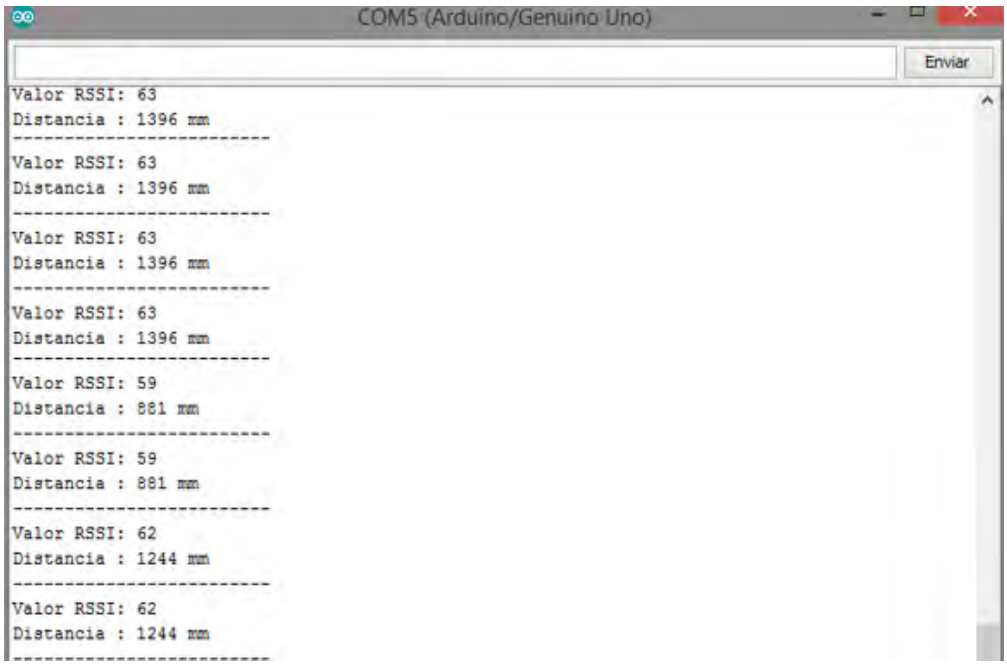

Figura 54. Estimación de distancia 2, 100 centímetros.

- A una distancia de 4 metros: Teniendo una separación entre dispositivos de 4 metros, la intensidad de la señal se vuelve inestable a medida que los módulos se distancia uno del otro, el valor del RSSI se va manteniendo estable a medida que los dispositivos permanecen inmóviles, obteniendo así los valores de la Figura 55; al igual que a un metro, existe margen de error en la distancia estimada y la distancia real.

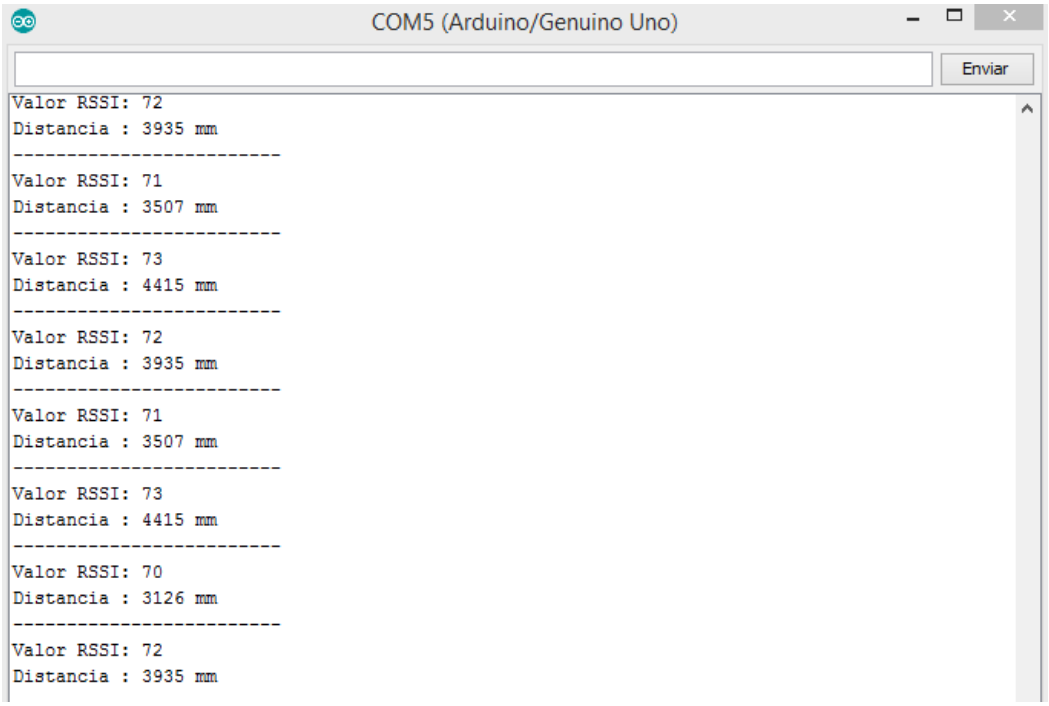

Figura 55. Estimación de distancia 3, Cuatro metros de separación.

- A una distancia de 10 metros: El valor del RSSI se pone inestable mientras los dispositivos se mueven a la posición de 10 metros de separación. A medida que toman la posición y permanecen inmóviles, estos valores se van poniendo cada vez más estables, y la estimación de la distancia es cada vez mejor, teniendo un margen de error como se puede ver en la Figura 56.

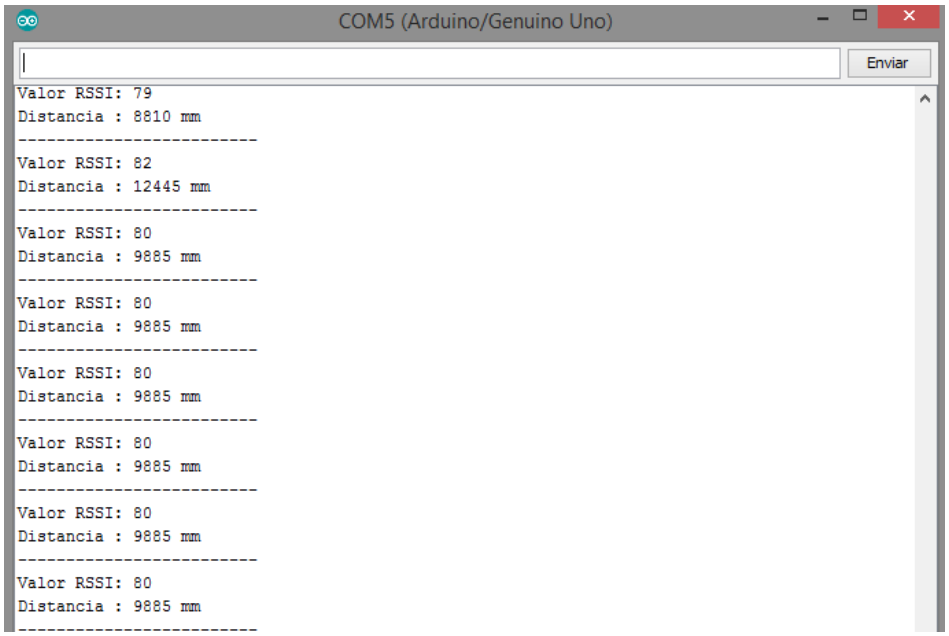

Figura 56. Estimación de distancia 4, separación de 10 metros.

En la Figura 57 podemos ver la comparación entre una distancia mayor a las tomadas anteriormente. A una distancia mayor, el valor del RSSI se pone muy inestable mientras el dispositivo se mueve, al quedar en un punto fijo, estos valores se van acercando cada vez más a la distancia real.

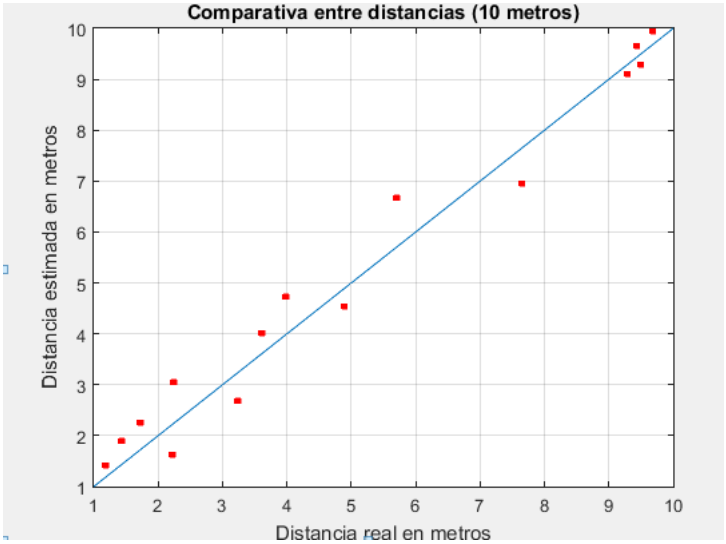

Figura 57. Comparación entre distancia real y estimada 2.

4 CAPITULO 4: CONCLUSIONES

Actualmente, son muchas las aplicaciones que necesitan estimar la posición relativa entre dispositivos; como ejemplos podemos citar a los sistemas de comunicaciones de vehículo a vehículo para hacer ajustes entre un transmisor y receptor de datos, la determinación correcta de la posición de un robot dentro de una habitación cerrada, o para saber la posición exacta de un dispositivo autónomo no tripulado (UAV por sus siglas en inglés) al momento de su aterrizaje.

Actualmente no existe una solución única y de precisión para la medición de distancia entre dispositivos. Estimar la distancia entre un dispositivo transmisor y un dispositivo receptor con línea de vista mediante la intensidad de la señal recibida (RSSI) demuestra que:

a) Los valores para determinar la distancia son inestables conforme va aumentando la distancia entre los dispositivos.
b) Interponer un objeto en la línea de vista aumenta/disminuye los valores de RSSI de manera considerable, aunque, se vuelven estables, al paso de unos segundos.
c) Cuando ambos dispositivos se mantienen a una distancia fija, los valores RSSI apenas muestran ligera alteración, por lo tanto, la distancia estimada, arroja valores cada vez más cercanos a los reales.
d) Al agregar movimiento a uno de los dispositivos, los valores obtenidos se disparan y alteran, y alcanzan valores estables a medida que alcanzan una posición fija.

La arquitectura propuesta permitirá en un futuro implementar nuevas técnicas de estimación y medición de la distancia (otros algoritmos de mayor complejidad) sin realizar cambios significativos en el hardware diseñado. El framework de transmisión y recepción de datos propuesto es fácilmente adaptable debido a su naturaleza de hardware abierto, lo que significa que bloques estandarizados de hardware pueden ser incorporados a la arquitectura actual de acuerdo a las interfaces estandarizadas.

5 Bibliografía

[1] Fernandes., T. (24 de Enero de 2011). *Indoor Localization Using Bluetooth.* Recuperado el 28 de Noviembre de 2016, de Faculdade de Engenharia da Universidade do Porto: http://paginas.fe.up.pt/~prodei/dsie11/images/pdfs/s5-4.pdf

[2] Javier Rodas, T. M. (22 de Abril de 2009). *ursi_rodas_2007. Sistema de Posicionamiento Basado en Bluetooth con Calibrado Dinámico.* Recuperado el 18 de Diciembre de 2016, de GTEC, Grupo de Tecnología Electrónica y Comunicaciones.: http://gtec.des.udc.es/web/images/pdfConferences/2007/ursi_rodas_2007.pdf

[3] Qian Dong, W. D. (14 de Mayo de 2012). *icwcuca. Evaluation of the Reliability of RSSI for Indoor Localization.* Recuperado el 02 de Diciembre de 2016, de Technische Universität Dresden: http://www.rn.inf.tu-dresden.de/dargie/papers/icwcuca.pdf

[4] Rukaiya Javaid, R. Q. (25 de Febrero de 2016). *RSSI based Node Localization using Trilateration in Wireless Sensor Network.* Recuperado el 14 de Diciembre de 2016, de Bahria University Journal of Information and Communication Technologies.: http://bujict.bimcs.edu.pk/wp-content/uploads/2016/02/27.-Finalized.pdf

[5] Xiaowei Luo, W. J. (27 de Octubre de 2010). *Comparative Evaluation of Received Signal-Strength Localization Techniques for construction jobsites.* Recuperado el 02 de Enero de 2017, de Mobile & Pervasive Computing Group.: http://mpc.ece.utexas.edu/Papers/ComparitiveEvaluationOfRSSI.pdf

[6] Juan Antonio Guerrero Ibáñez, C. F. (07 de Febrero de 2011). *Análisis de desempeño de estándar 802.11p en situaciones de handoff dentro de un entorno de redes vehiculares.* Recuperado el 12 de Diciembre de 2016, de UNIVERSITAT POLITÉCNICA DE CATALUNYA BARCELONA TECH: http://upcommons.upc.edu/bitstream/handle/2117/11304/CNCIIC-2010_angelica.pdf?sequence=1

[7] "Arduino". (s.f.). *Arduino*. Recuperado el 01 de Diciembre de 2016, de https://www.arduino.cc

[8] Marla Glen M, J. M. (2012 de Mayo de 23). *Wikispaces*. Recuperado el 03 de Diciembre de 2016, de https://sx-de-tx.wikispaces.com/ZIGBEE

[9] "Xbee". (s.f.). *Xbee.cl*. Recuperado el 03 de Diciembre de 2016, de http://xbee.cl/que-es-xbee/

[10] "DIGI". (s.f.). *DIGI*. Recuperado el 04 de Diciembre de 2016, de https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

[11] Ozerdo. (02 de Marzo de 2012). *AIRHEADS COMMUNITY*. Recuperado el 05 de Diciembre de 2016, de https://community.arubanetworks.com/t5/Community-Tribal-Knowledge-Base/Making-sense-of-RSSI/ta-p/25376

[12] *Propagación de RF.* (23 de Febrero de 2005). Recuperado el 14 de Diciembre de 2016, de UDLAP Bibliotecas: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/peredo_a_s/capitulo1.pdf

Anexos

```
#include <XBee.h>                    //Librería Xbee.
#include <SoftwareSerial.h>          //Librería para comunicación Serial.

//Se crea una variable asignada a SoftwareSerial.
//Se asignan los pines 2 y 3.
SoftwareSerial rxtx(2, 3);           //Se definen las entradas RX y TX en 2 y 3.


//Creamos un objeto Xbee. Este objeto provee
//funciones para enviar y recibir paquetes.
XBee xbee=XBee();

//Objeto XbeeResponse para recepción de paquetes.
XBeeResponse response = XBeeResponse();

//Inicializamos el objeto Response.
Rx16Response rx16 = Rx16Response();

int rssi = 0;  //Variable que almacena los valores recibidos RSSI
int n = 2.5;     //Constante de propagación de la señal.
int d = 0;     //Varianble que almacena el valor de la distancia.

void setup()
```

```
{
  Serial.begin(9600);    // Iniciar comunicación serial.

  rxtx.begin(9600);      // Asignamos a variable SoftwareSerial velocidad de transmisión.

  xbee.setSerial(rxtx);  // Comunicacíón serial Xbee.
}

void loop()
{
  xbee.readPacket(100);                          // Lectura de paquetes.
  if (xbee.getResponse().isAvailable())                  // Condicional que verifique que el Response este disponible.
  {
    if(xbee.getResponse().getApiId() == RX_16_RESPONSE)     // Revisar que el paquete sea del tipo correcto RX16.
    {

      if (xbee.getResponse().getApiId() == RX_16_RESPONSE)
      {
        xbee.getResponse().getRx16Response(rx16);          // Se obtiene y lee el paquete.
        rssi = rx16.getRssi();                          // Se ontiene el valor de la intensidad de la señal.

        d = pow (10, ((rssi-0.1)/ (10 * n)));             // Fórmula cálculo distancia entre dispositivos.
        Serial.print("Valor RSSI: ");                  // Mostramos en monitor serie el texto "Valor RSSI: ".
        Serial.println(rssi);                          // Se imprime el valor obtenido del RSSI.
        Serial.print("Distancia : ");                  // Imprimimos en monitor serie la palabra "Distancia: ".
```

```
      Serial.print(d);                          // Imprimimos el valor obtenido de la distancia.
      Serial.println(" mm");                    // Mostramos que la distancia se mide en milímetros.
      Serial.println("------------------------");       // Indicamos una separación entre mensajes.
      delay(1000);                              // Se retrasa la lectura del siguiente mensaje a 1 segundo.
    }


    }
  }
 }
```