ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería Eléctrica y Computación

INFORME DE MATERIA DE GRADUACIÓN

"LOCALIZACIÓN DE OBJETOS EN RANGOS CORTOS EN REAL-TIME USANDO SEÑALES DE RADIOFRECUENCIA"

Previo a la obtención del Título de:

INGENIERO EN TELEMÁTICA

Presentado por:

Johnny Francisco Bravo Moreno

José Andrés Gómez Graciano

GUAYAQUIL – ECUADOR

2013

AGRADECIMIENTO

A mis amigos los cuales también son parte de esta alegría, pues han estado presentes en las distintas etapas de mi vida, por ayudarme a crecer como persona, gracias a ustedes estoy en este punto, ustedes son lo máximo.

Al Phd. Daniel Ochoa, por su guía a través del desarrollo de nuestro proyecto.

TRIBUNAL DE SUSTENTACIÓN

_____

Phd. Daniel Ochoa

PROFESOR DE LA MATERIA DE GRADUACIÓN

_____

Ing. Patricia Chávez MSEE

PROFESOR DELEGADO POR LA UNIDAD ACADÉMICA

DEDICATORIA

A DIOS por todas las bendiciones recibidas. A mis padres y hermanos por brindarme siempre su amor y motivación durante toda mi vida.

Johnny Francisco Bravo Moreno.

A mis padres ya que ellos han sido mi principal motivación para seguir adelante y alcanzar una de mis metas.

José Andrés Gómez Graciano.

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este informe, nos corresponde exclusivamente, y el patrimonio intelectual del mismo a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

Johnny Francisco Bravo Moreno

_____

José Andrés Gómez Graciano

RESUMEN

En el capítulo 1 de este documento se definen las razones por las que se estudia e implementan sistemas de localización de objetos con dispositivos de transmisión y recepción de señales y computadores que procesen estas señales, para luego detallar los objetivos de nuestro proyecto.

En el capítulo 2 se detallan los fundamentos teóricos en los que se basa el funcionamiento de los dispositivos y computadoras a utilizar, los fenómenos físicos que los afectan, la programación utilizada en estas y el método matemático para estimar la posición de un objeto en tiempo real.

En el capítulo 3 se describe la tecnología que se utilizó para implementar el sistema, se establecen los posibles modelos de funcionamiento y el diseño del sistema.

En el capítulo 4 se detallan las pruebas necesarias para evaluar el sistema en un lugar cerrado como lo es el laboratorio del Centro de Visión y Robótica, para verificar su viabilidad respecto a las expectativas propuestas en este proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

CPU: Central Process Unit ó Unidad Central de Procesamiento.

CSMA-CA: Carrier Sense Multiple Access with Collision Avoided ó
Acceso por detección de portadora con evasión de colisiones.

CVR: Centro de Visión y Robótica.

DSSS: Direct Sequence Spread Spectrum ó
Espectro Ensanchado por Secuencia Directa.

FIFO: First In, First Out ó Primer entrante, primer saliente

IEEE: Institute of Electrical and Electronics Engineers ó
Instituto de Ingenieros Eléctricos y Electrónicos.

IP: Internet Protocol ó Protocolo de Internet.

MAC: Medium Access Control ó Control de Acceso al Medio.

O-QPSK: Offset Quadrature Phase Shift Keying ó
Desplazamiento de fase en cuadratura escalonada.

OSI: Open System Interconnection ó Sistema Abierto de Interconexión.

PCB: Process Control Block ó Bloque de Control de Proceso.

RSSI: Received Signal Strength Indicator ó Fuerza de señal recibida.

RR: Round Robin ó Tajada Robin

RT: Real Time ó Tiempo Real.

INTRODUCCIÓN

Los sistemas que permiten estimar la ubicación de un objeto en movimiento de manera automática han sido objeto de un intenso estudio e investigación durante varios años. El Sistema de Localización Global (GPS) creado en los Estados Unidos es el más utilizado debido a que presenta un margen de error promedio de 20 metros de la posición correcta en la que se encuentra el objeto. Esta precisión del GPS la brindan 24 satélites, que reciben señales de un transmisor incorporado en el objeto, cuyos relojes atómicos miden los tiempos de transmisión con los que se hallan las distancias del objeto a los satélites, a aplicarse en el método de localización por trilateración para estimar la posición.

Por esta razón se han querido crear sistemas similares que mejoren la precisión del GPS entre los que se resaltan: GLONASS de Rusia, Galileo de la Unión Europea y Beidou de la República Popular China, y una tendencia al estudio de la localización de objetos para lugares abiertos o cerrados. [1]

Este proyecto tiene como fin estudiar los métodos de localización existentes y la tecnología disponible en el mercado para implementar un sistema que estime la ubicación aproximada de un objeto en tiempo real, que pueda ser utilizado como un complemento al GPS y posiblemente que mejore su precisión.

CAPÍTULO 1

PLANTEAMIENTO

El objetivo de este capítulo es describir los problemas que existen en el GPS y en los sistemas de localización de objetos que se complementan a estos, y definir los objetivos a alcanzar con la realización de este proyecto.

1.1 DEFINICIÓN DEL PROBLEMA.-

El GPS es un sistema de localización de un objeto en lugares abiertos que brinda un margen de error de 20 [metros], sin embargo en lugares cerrados se obtienen errores superiores debido a fenómenos físicos, estructura del lugar, que atenúan la señal de los dispositivos utilizados para implementar el sistema.

Para solucionar esto se han implementado diversos sistemas con dispositivos que abarquen la suficiente distancia y computadores con la suficiente capacidad de procesamiento, que se complementen con un GPS para estimar la posición de un objeto en un lugar cerrado de manera precisa, sin embargo si el objeto se mueve rápidamente estos sistemas suelen dar lecturas imprecisas a causa de que el tiempo de respuesta del sistema no es lo suficientemente rápido para estimar la posición en tiempo real, es decir apenas el objeto se mueva. [2]

1.2 OBJETIVOS DEL PROYECTO.-

1.2.1 OBJETIVOS GENERALES

Estudiar los métodos de localización existentes y tecnología disponible en el mercado para implementar un sistema que estime la ubicación aproximada de un objeto en tiempo real, que pueda ser utilizado como un complemento al GPS.

1.2.2 OBJETIVOS ESPECÍFICOS

- Crear programas que permitan transmitir o recibir señales entre dispositivos con antenas de radio.
- Implementar una red de computadores que procese las señales provenientes de los dispositivos con antenas de radio.
- Crear un programa que estime la posición de un objeto en movimiento por medio de técnicas de programación soft real time.

1.3 METODOLOGÍA.-

Los pasos para la realización de este proyecto son:

1. Instalación de distribución de Linux en los computadores utilizados en el proyecto. Para el proyecto se utilizó la distribución Ubuntu 12.10.
2. Instalación de programa escrito en POSIX C en el computador conectado al xbee sobre el objeto a localizar, para que este radio xbee pueda transmitir datos con potencias específicas.
3. Instalación de programa en los computadores conectados a los xbee en posiciones de referencia conocidas alrededor del objeto a localizar, para que los estos radios xbee puedan recibir datos transmitidos por el xbee sobre el objeto, y extraer la potencia específica de los datos transmitidos.
4. Instalación de programa en computador que aplica el método de localización por trilateración con las potencias obtenidas y que con funciones de soft real time permite localizar al objeto en tiempo real.
5. Ejecución de programas en conjunto en los computadores para localizar al objeto en tiempo real.

CAPÍTULO 2

MARCO TEÓRICO

El objetivo de estudio de este capítulo es explicar los fundamentos teóricos que se utilizan en la implementación de un sistema de localización de objetos. Se empieza con una descripción general acerca de los dispositivos de transmisión y recepción de señales así como los fenómenos físicos que afectan su funcionamiento, luego se describe el funcionamiento de un computador en base a la administración de las tareas requeridas por un usuario y finalmente se analiza el método matemático de trilateración para localizar un objeto por medio de los dispositivos de transmisión y recepción de señales de radiofrecuencia.

2.1 EXACTITUD EN MEDIDAS DE POTENCIA

Un sistema de comunicación inalámbrico es un conjunto de varios dispositivos con antenas que permiten transmitir datos específicos de un lugar a otro, de forma que la comunicación se origina desde un dispositivo con una antena capaz de transmitir los datos hacia uno o más dispositivos con antenas capaces de recibir los datos, como se muestra en la figura 2.1. [3]

Una señal electromagnética es la representación energética en el tiempo de un tipo de información que se quiera transmitir, y está definida por:

- **Frecuencia:** Es la repetición “f” expresada en [Hertz] de una señal en un tiempo determinado.
- **Potencia:** Es la energía mínima “Pt” o “P_R” expresada en [Watts] requerida por una antena para transmitir o recibir una señal respectivamente, sin que se pierdan los datos que contiene.

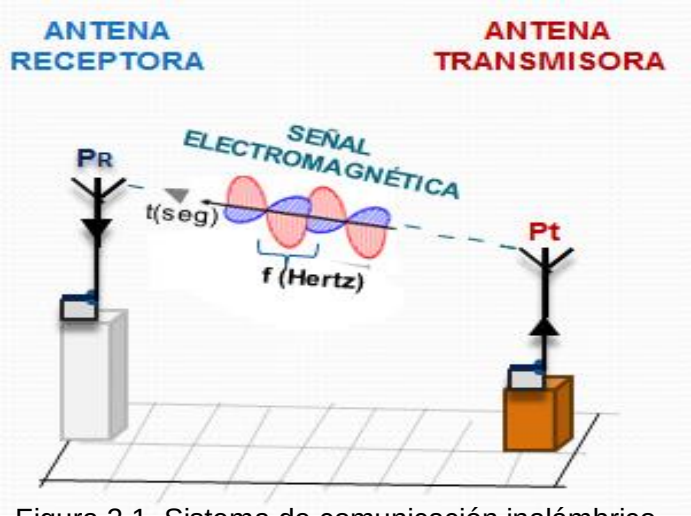


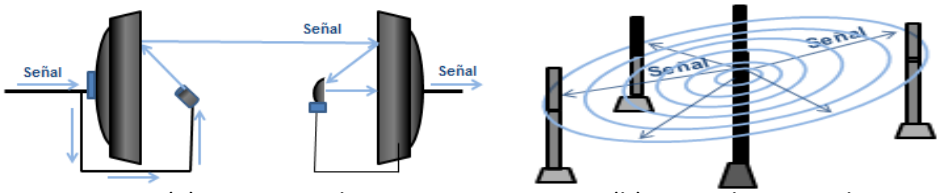
Figura 2.1 Sistema de comunicación inalámbrico

El objetivo en un sistema de comunicación es mantener constante la potencia de la señal durante toda su propagación, sin embargo el material de fabricación de las antenas y ciertos fenómenos físicos presentes en el ambiente provocan que parte de esta potencia se pierda en el entorno, lo que implica una pérdida de los datos que contiene la señal y altera la información transmitida.

2.1.1 Pérdidas de Potencia de la señal en las antenas.

La señal puede transmitirse por medio de 2 tipos de antenas:

- **Direccional:** Concentra en su punto central la señal y la transmite en una sola dirección. [Ver figura 2.2a]
- **Omnidireccional:** Concentra radialmente la señal a su alrededor y la transmite en varias direcciones. [Ver figura 2.2b]



(a) Direccional (b) Omnidireccional

Figura 2.2 Tipos de antenas de un dispositivo

El material de fabricación de las antenas es el metal, el cual absorbe mucha potencia. Debido a esto, la potencia P_{tIN} de la señal que llega a la antena transmisora se reduce a una potencia P_{tOUT} en su salida, y la potencia P_{RIN} que llega a la antena receptora se reduce a una potencia P_{ROUT} en la salida. Estas pérdidas se cuantifican como ganancias: $G_t = 10.\log [P_{tOUT}/P_{tIN}]$ en la transmisora y $G_R = 10.\log [P_{ROUT}/ P_{RIN}]$ en la receptora, expresadas en [Decibelios] como se ve en la figura 2.3. [4]

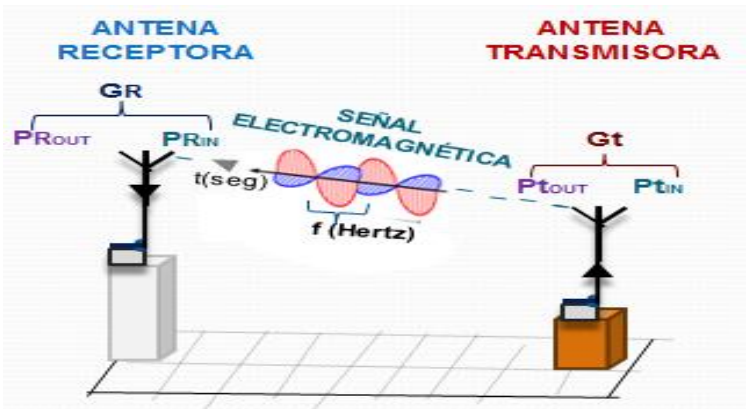


Figura 2.3 Pérdidas de potencia de la señal en las antenas

2.1.2 Pérdidas de Potencia de la señal durante su propagación.

Durante la propagación de la señal entre antenas existen 2 fenómenos físicos que pueden provocar una disminución de potencia:

- **Reflexión:** Una señal que no se desea recibir en una antena se encuentra con una superficie que absorbe parte de la señal, pero otra parte llega a la antena.
- **Difracción:** Una señal que no se desea recibir en una antena se encuentra con la abertura u obstáculo de una superficie, la cual desvía la señal hacia la antena. [4]

La combinación de señales no deseadas que llegan a la antena en un tiempo determinado se la conoce como **multitrayectoria.** [Ver figura 2.4]

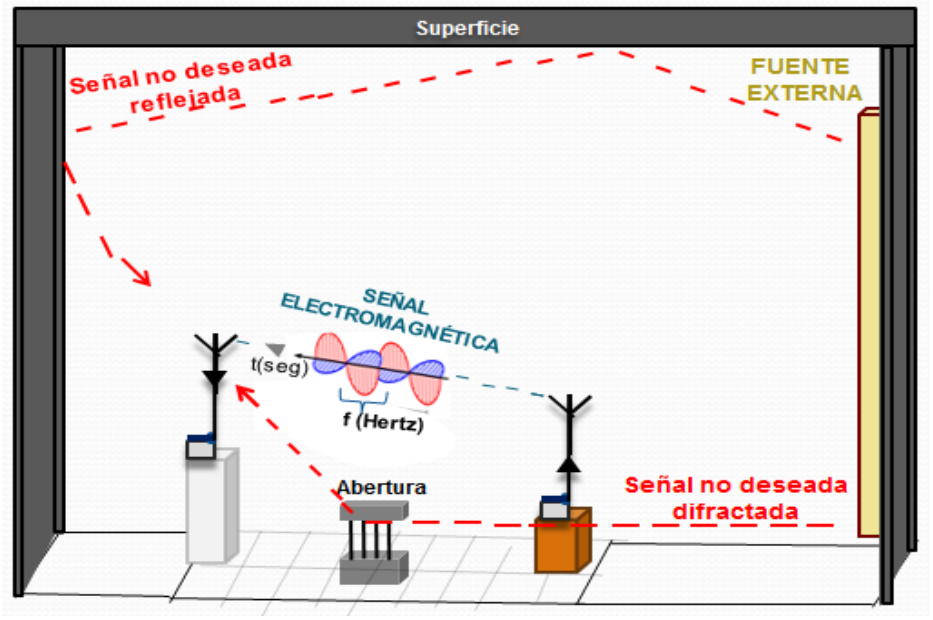


Figura 2.4 Pérdidas de potencia de la señal en el entorno

Para reducir los efectos de estos fenómenos se ubica la receptora a una altura h_r en una dirección de línea de vista (LOS) en la que no se presenten superficies metálicas, de agua o aberturas que pueden reflejar o difractar la señal y generar pérdidas de potencia. Esta altura h_r se halla mediante el trazo de un triángulo como se puede observar en el esquema de la figura 2.4. [Ver figura 2.5]

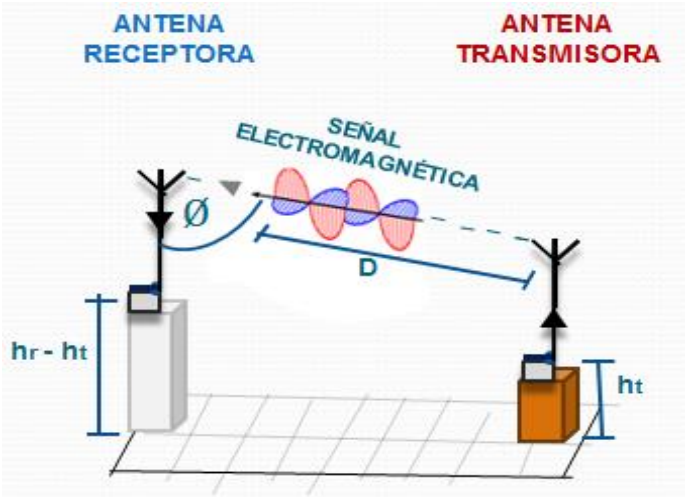


Figura 2.5 Representación Geométrica de propagación sin pérdidas

Donde la altura ideal hr se expresa como:

$$h_r = h_t + D.\cos \emptyset \quad (1)$$

Sin embargo como D y $\emptyset$ pueden presentar una infinita cantidad de valores, la ecuación (1) no permite determinar la altura h_r. En la práctica esta altura se limita a los extremos más altos del lugar donde se instale la antena, en la que no se generen mayoritariamente los fenómenos de reflexión y difracción que produzcan pérdidas de potencia considerables.

En el año de 1945 el Ingeniero Danés H. Friss cuantificó la ganancia G que se genera durante la propagación de la señal entre las antenas de un sistema de comunicación, en términos de las ganancias G_t y G_R, velocidad de propagación c, frecuencia f y distancia entre antenas D:

$$G = P_{RIN} / P_{tOUT} = 10. \log [G_t.G_{R.}(c / 4\pi f.D)^2] \ [dB] \quad (2)$$

Donde la distancia D a la que una señal puede ser transmitida o recibida por una antena sin pérdida de datos se denomina: alcance de transmisión o recepción respectivamente.

La ecuación (2) describe la potencia mínima P_{RIN} que requiere una antena para recibir una señal con un alcance D, sin embargo la potencia varía durante la propagación debido a fenómenos físicos de reflexión y difracción, por lo que P_{RIN} puede representarse como una variable aleatoria gaussiana de media $\overline{P_{RIN}}$ y desviación σ_{PRIN} con un intervalo: $(\overline{P_{RIN}} - \sigma_{PRIN} , \overline{P_{RIN}} + \sigma_{PRIN})$ en el que el valor de P_{ROUT} es correcto con una certeza del 68%. [5]

En el año 1960 se creó la teoría de propagación de errores que calcula la incertidumbre de una variable dependiente de otra de incertidumbre conocida. Como D depende de P_{ROUT} se puede determinar la incertidumbre de D como su desviación σ_D en términos de la derivada de P_{RIN} como P_{RIN}' y la desviación σ_{PROUT}.

$$\sigma_D = \sigma_{PRIN} / P_{RIN}' \quad (3)$$

2.2 SISTEMA COMPUTACIONAL

Un computador es un sistema que organiza y ejecuta secuencias de instrucciones llamadas programas por medio de 4 partes o recursos:

- **Memoria Principal y Secundaria:** Espacios direccionados en los que se guardan de forma temporal y permanente respectivamente: programas, datos y resultados de procesamiento.
- **Dispositivo de Entrada y Salida:** Dispositivo conectado al computador que ejecuta instrucciones de Entrada y Salida (I/O) como obtener datos o comunicar resultados respectivamente.
- **Procesador (CPU):** Ejecuta secuencialmente procesos que son las instrucciones de cálculo de datos en memoria combinadas con los recursos necesarios para ejecutarlas.[Ver figura 2.6]

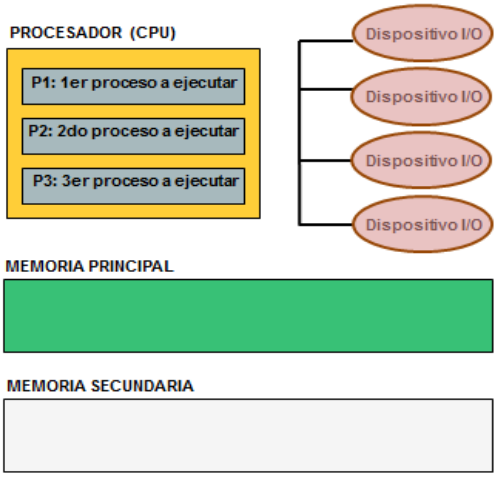


Figura 2.6 Estructura de un Sistema Computacional

En esta sección se detalla como ha evolucionado el funcionamiento en conjunto de los recursos de un computador para ejecutar las instrucciones de uno o varios procesos, y los métodos para comunicar varios procesos alojados en computadores diferentes. [6]

2.2.1 Planificación de Procesos en un computador

El sistema operativo es el programa almacenado en la memoria secundaria del computador, que brinda el entorno para la administración eficiente de los recursos del computador y está compuesto de:

- **Kernel:** Codificación principal del sistema operativo que es la que se encarga de gestionar los recursos y comunicar los procesos del computador con su hardware.
- **Sistema de Archivos:** Codificación que organiza jerárquicamente la información almacenada en la memoria secundaria, para facilitar el acceso a esta por parte del kernel.
- **Interfaz de usuario:** Interfaz que facilita la interacción entre los usuarios y los recursos del computador por medio de comandos (CLI) o gráficos (GUI). [Ver figura 2.7]

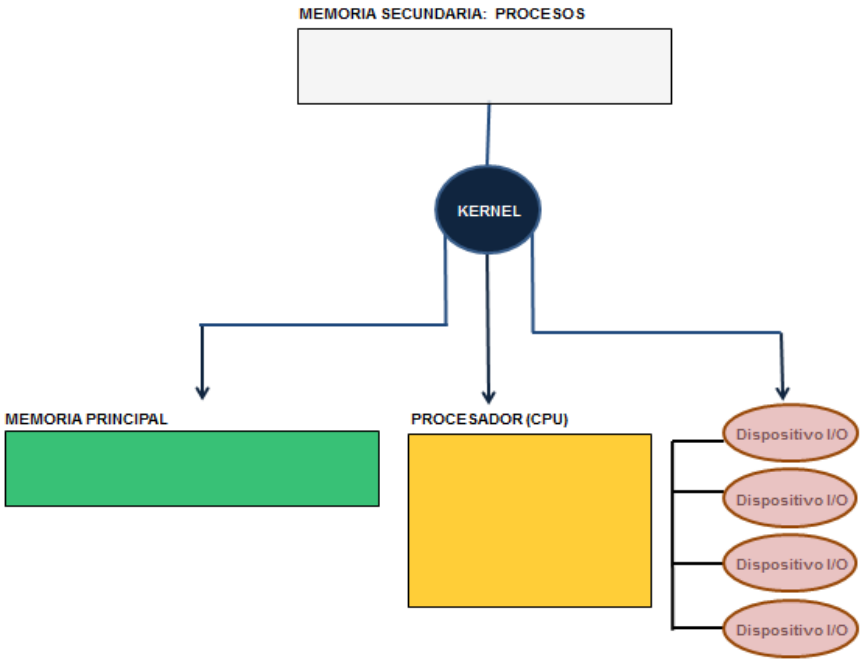


Figura 2.7 Sistema Operativo

El kernel gestiona como los procesos utilizan el CPU para ejecutar sus instrucciones de cálculo eficientemente, por medio de las siguientes estructuras de datos en su codificación:

- **Bloque de Control de procesos (PCB):** Estructura representativa de cada proceso del sistema.
- **Cola de procesos listos:** Estructura en la que se cargan los PCB de los procesos listos para ejecutar sus instrucciones.
- **Cola de procesos bloqueados:** Estructura en la que se cargan los PCB de los procesos mientras ejecuten una instrucción de I/O. [Ver figura 2.8]

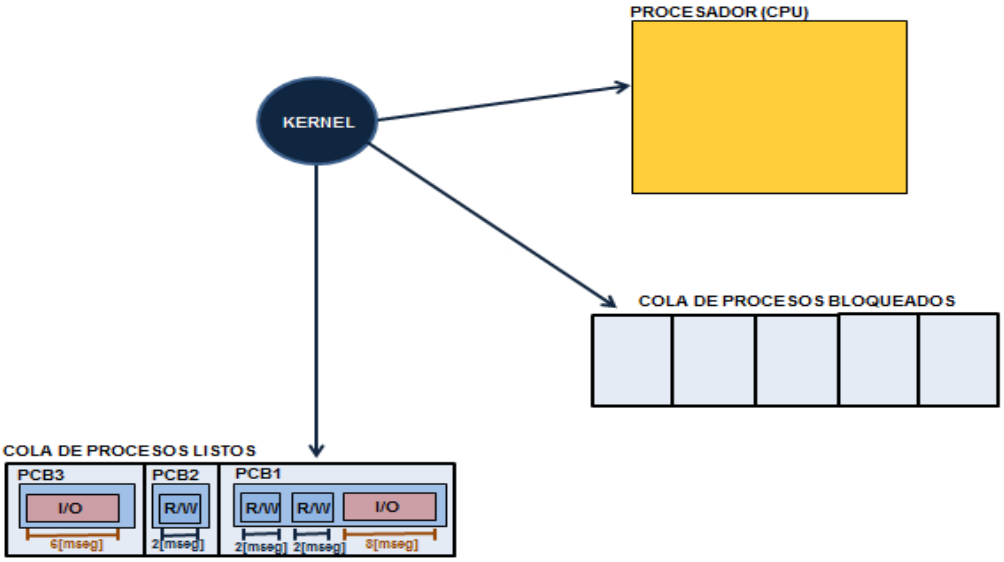


Figura 2.8 Planificación del Kernel

Cuando un proceso requiere ejecutar una instrucción de cálculo, el kernel le da acceso al CPU para que ejecute la instrucción. De la misma manera si un proceso requiere ejecutar una instrucción de I/O, el kernel le da acceso al dispositivo de I/O requerido para ejecutar la instrucción y el PCB del proceso se carga en la cola de procesos bloqueados.

Los primeros CPUs creados en los años 50 eran controlados por un sistema monoprogramado en el que se ejecutan todas las instrucciones de un proceso listo antes de ejecutarse las del siguiente proceso listo.

Si se consideran los procesos P1, P2 y P3 de la figura 2.8, un sistema monoprogramado ejecuta sus instrucciones así:

- Se carga el proceso P1 de la cola de procesos listos al CPU:
  - Se ejecuta su instrucción de cálculo en 2[mseg].
  - Se ejecuta su instrucción de I/O y el PCB1 se carga en la cola de procesos bloqueados durante 8[mseg]
  - Se ejecuta su instrucción de cálculo en 2[mseg].
- Se carga el proceso P2 de la cola de procesos listos al CPU:
  - Se ejecuta su instrucción de cálculo en 2[mseg].
- Se carga el proceso P3 de la cola de procesos listos al CPU:
  - Se ejecuta su instrucción de cálculo en 6[mseg].

En total un sistema monoprogramado se tomó 20[mseg] para ejecutar las instrucciones de estos procesos.

La desventaja de este tipo de sistemas es que las instrucciones de I/O normalmente tardan mucho tiempo en ejecutarse, durante el cual el CPU no ejecuta instrucción de cálculo alguna, lo que implica que la capacidad de procesamiento del CPU es desperdiciada durante ese tiempo.

En los años 60 se introdujo la multiprogramación en la que se aprovecha el tiempo del CPU mientras espera que una instrucción de I/O se ejecute por un dispositivo específico. Durante este tiempo, el kernel le cede el CPU al siguiente proceso listo para ejecutar sus instrucciones.

Si se consideran los procesos P1, P2 y P3 de la figura 2.8, un sistema multiprogramado ejecuta las instrucciones así:

- Se carga el proceso P1 de la cola de procesos listos al CPU:
    - Se ejecuta su instrucción de cálculo en 2[mseg].
    - Se ejecuta su instrucción de I/O y el PCB1 se carga en la cola de procesos bloqueados durante 8[mseg], y mientras esto ocurre:
        - Se carga el proceso P2 en el CPU y se ejecuta su instrucción de cálculo en 2[mseg].
        - Se carga el proceso P3 en el CPU y se ejecuta la instrucción de cálculo en 6[mseg].

En total el sistema multiprogramado se tomó 10[mseg] para ejecutar las instrucciones, que es menor que los 20[mseg] que tomaba con un sistema monoprogramado.

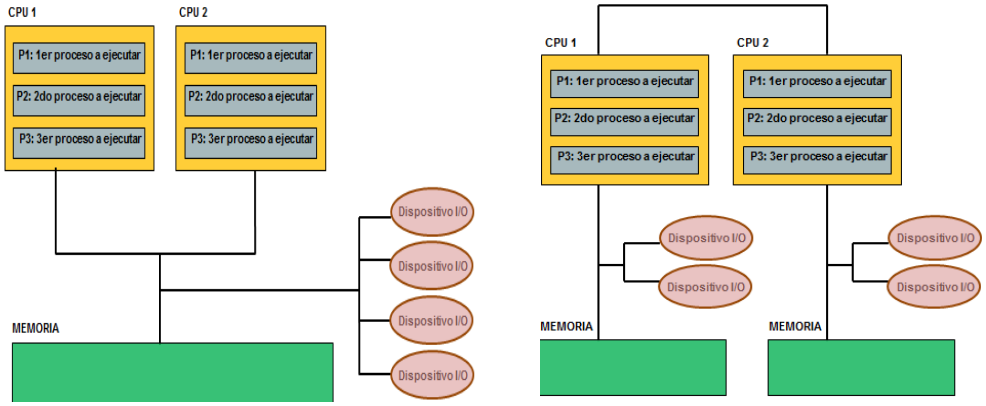
Se mejoró la multiprogramación por medio del concepto de tiempo compartido en el que las instrucciones se ejecutan como en un sistema con multiprogramación pero además limitados por un tiempo denominado: quantum que es definido por el kernel del sistema, para así repartir equitativamente el CPU entre procesos.

En los años 70 se crearon los sistemas de multiprocesamiento formados por varios CPUs en el que cada uno ejecuta uno o varios procesos con multiprogramación, lo que generaba una gran capacidad de procesamiento en el sistema.

Los CPUs eran administrados por el sistema según 2 modelos:

- **UMA:** Los CPUs comparten la memoria y los dispositivos de I/O, lo que causa problemas cuando varios procesos quieren modificar un dato al mismo tiempo y por ello el sistema operativo solo acepta la acción de uno de los procesos y anula las demás. [Ver figura 2.9a]

- **NUMA:** Cada CPU tiene su propio bloque de memoria al que puede acceder lo que reduce el problema del modelo UMA, sin embargo genera retardos de tiempo a un proceso cuando este busca un dato en un bloque de memoria que no le pertenece.



(a) UMA (b) NUMA

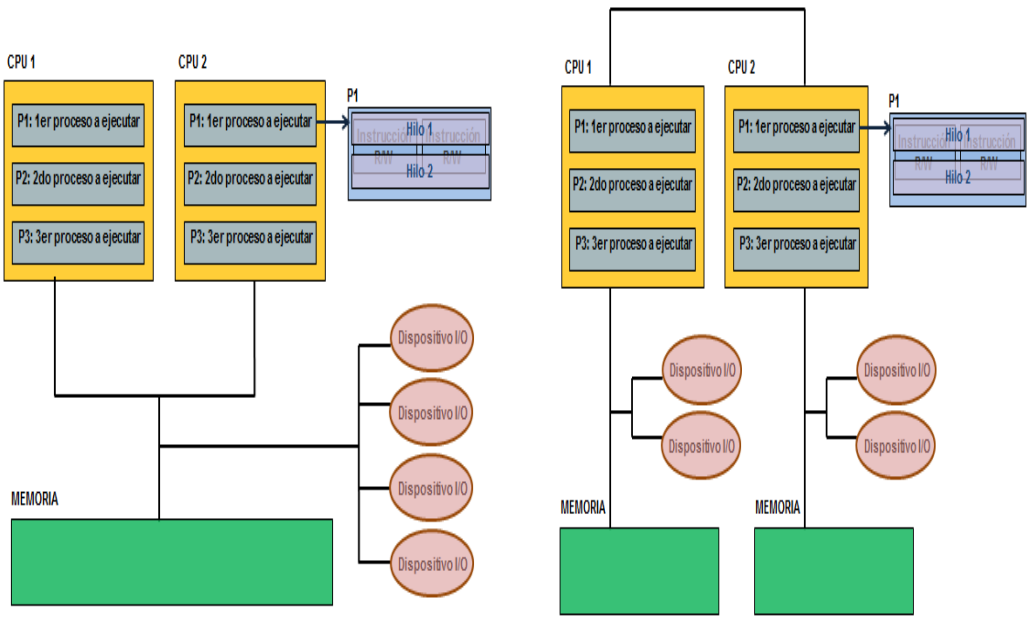
Figura 2.9 Modelos de multiprocesamiento

Para solucionar el problema del modelo UMA se aplican semáforos que son variables contadoras que permiten sincronizar los procesos de un CPU, de manera que mientras un proceso acceda a un espacio de memoria los demás no pueden accederlo.

A finales de los años 70 se introdujo el concepto de multihilo para aumentar más la capacidad de procesamiento de un sistema de multiprocesamiento, en el que se divide un proceso en varias instancias de este llamadas hilos que:

- Comparten el bloque de memoria del proceso al que pertenecen.
- Cooperan en la realización de sus instrucciones.

Sin embargo al igual que los retardos de tiempo o bloqueo entre procesos presentes en los modelos de multiprocesamiento, un hilo también puede presentar los mismos problemas con respecto a otro hilo, que al igual que los procesos puede solucionarse por medio de semáforos que sincronizan los hilos de un proceso. [Ver figura 2.10]



(a) Hilos en UMA (b) Hilos en NUMA

Figura 2.10 Hilos de Ejecución de instrucciones simultaneas

Cuando un sistema ejecuta sus instrucciones, se puede obtener el tiempo de respuesta límite que requiere para ejecutarlas correctamente, sin embargo los componentes del sistema pueden presentar retardos de tiempo que provoquen que este tiempo no se cumpla. Por esta razón desde el año 2000 se implementó el concepto de sistema en tiempo real (RT), que define conceptos de planificación por multiprogramación, multiprocesos o multihilos, con funciones integradas en el kernel, que permitan que el sistema ejecute sus instrucciones efectivamente en su tiempo límite especificado. Puede ser de 2 tipos:

- **Tiempo Real Crítico (Hard Real time):**

  Estos sistemas no toleran que sus instrucciones terminen fuera del tiempo límite de respuesta ni una sola vez, ya que pueden tener consecuencias catastróficas, un ejemplo de esto es el marcapasos, que si no respondiera a tiempo una vida humana correría peligro.

  Para obtener este tipo de sistema se debe instalar un sistema operativo por naturaleza de tiempo real o un sistema al que se le instale una extensión específica que contenga las funciones para convertirlo en un sistema en tiempo real. [Ver figura 2.11]

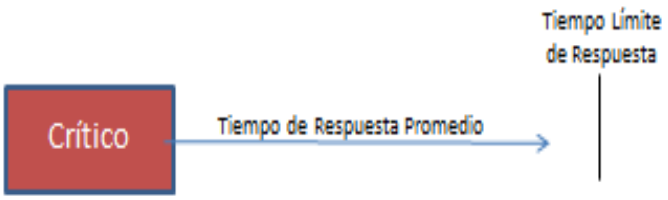


Figura 2.11 Tiempo Real Crítico

- **Tiempo Real Acrítico (Soft Real Time):**

  Estos sistemas pueden tolerar pequeños retardos ocasionales de tiempo fuera del tiempo límite de respuesta que no crearán fallas catastróficas en el sistema, como es en el caso de la transmisión de datos entre dispositivos de radio como los utilizados en este proyecto, pues si se pierde una cierta cantidad de datos durante la transmisión, la precisión de esta comunicación no será tan exacta, pero aceptable y no provoca fallas catastróficas en la respuesta final del sistema. [7]

  Para obtener este tipo de sistema, se debe instalar un sistema operativo con un kernel que sea compatible con las funciones POSIX de soft real time de acuerdo al lenguaje de programación que se utilice, y se manipula la planificación y prioridades en la ejecución de los procesos o hilos por parte del kernel por medio de algoritmos de planificación específicamente creados para administrar los recursos de un sistema de forma más eficiente en términos de tiempo y de priorización de ciertos procesos o hilos sobre otros, que con un sistema que no es de tiempo real. [Ver figura 2.12]

Figura 2.12 Tiempo Real Acrítico

2.2.2 Comunicación entre procesos en computadores diferentes

Socket es una estructura lógica de programación que le permite a un proceso cliente en un computador compartir datos a un proceso servidor en otro computador. [8]

Para implementar esta comunicación, en cada computador se crea un socket que va a tener asociado los siguientes parámetros:

- **IP:** Dirección que identifica a cada computador en la red.
- **Puerto:** Número que identifica al proceso en cada computador.
- **Protocolo de transporte de datos:** Normas de cómo establece el cliente la conexión con el servidor, y puede ser:

    - **TCP o Stream:** El cliente establece una conexión con el servidor que garantiza que se transmitan todos los datos y en orden al servidor por medio de un mensaje de confirmación ACK, que implica un retardo de tiempo.

    - **UDP o Datagram:** El cliente no establece una conexión con el servidor, ni garantiza que lleguen todos los datos ni en orden al servidor, pero no implica retardos de tiempo por ACK.

Sin importar el lenguaje de programación que se haya utilizado para codificar los procesos, la comunicación por sockets TCP se establece mediante los siguientes pasos:

- **Creación del socket:** En cada computador se crea un socket con dirección IP local, número de puerto y protocolo de transporte TCP.

- **Apertura del socket:** El servidor se queda en un estado en el que escucha peticiones de clientes que deseen conectarse, y en el cual puede ocurrir los siguientes casos :

    - Un cliente envía una petición de conexión hacia el servidor que este acepta, con lo que el cliente está habilitado para compartirle datos al servidor hasta que este último le niegue la conexión.

    - Un cliente envía una petición de conexión hacia el servidor que este deniega, con lo que el servidor sigue en estado de escucha de peticiones de conexión.

- **Cierre del socket:** El servidor deja de escuchar peticiones de clientes que quieran conectarse al mismo.

De igual manera la comunicación por sockets UDP se establece mediante los siguientes pasos:

- **Creación del socket:** En cada computador se crea un socket con dirección IP local, número de puerto y protocolo de transporte UDP.

- **Apertura del socket:** El servidor se queda en un estado de disponibilidad en el que cualquier cliente de la red con el mismo número de puerto está habilitado para compartirle datos al servidor.

- **Cierre del socket:** El servidor deja de estar en el estado de disponibilidad para los clientes. [Ver figura 2.13]

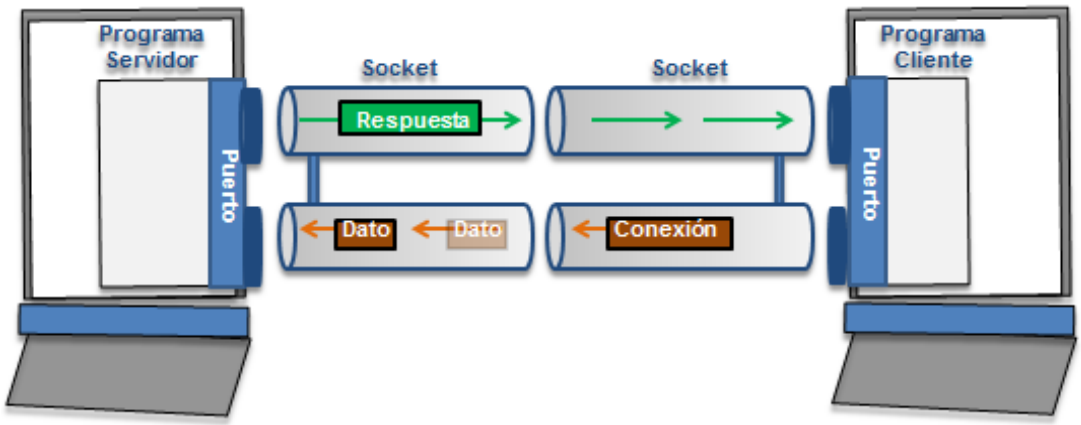


Figura 2.13 Modelos de Comunicación por Sockets

2.3 LOCALIZACIÓN DE OBJETOS CON TRILATERACIÓN.-

Trilateración es un método matemático que permite estimar la ubicación P:(x,y,z) de un objeto en un plano tridimensional con la utilización de:

- **Puntos de referencia conocidos:** $P_1(x_1,y_1,z_1)$,$P_2(x_2,y_2,z_2)$ y $P_3(x_3,y_3,z_3)$
- **Distancias entre el objeto y cada uno de los puntos:** d_{10}, d_{20}, d_{30}.

Con estos se grafican los centros y radios respectivamente de 3 esferas que se intersectan en el punto P:(x,y,z) a determinar. [Ver figura 2.14]

En la práctica en los puntos de referencia conocidos se ubican dispositivos de transmisión, para que emitan señales que presentan potencias o tiempos de propagación con los que se pueden estimar las distancias entre el objeto y cada uno de los puntos; si los dispositivos de transmisión tienen integrados relojes que permiten estimar el tiempo de propagación exacto de sus señales, se utiliza este tiempo de propagación multiplicado por su velocidad de propagación para estimar estas distancias, sino se utilizan las potencias de las señales. [9]

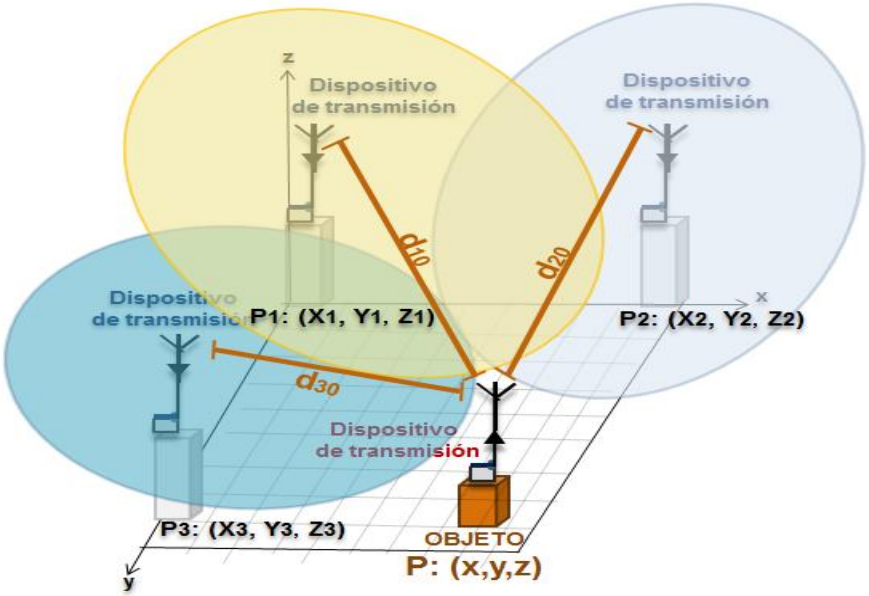


Figura 2.14 Esquema de Localización por Trilateración

Con las ecuaciones de estas esferas se genera un sistema de 3 ecuaciones con 3 incógnitas de esta forma:

$$d_{10}{}^2 = (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 \qquad (4)$$

$$d_{20}{}^2 = (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 \qquad (5)$$

$$d_{30}{}^2 = (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 \qquad (6)$$

Por facilidad en la resolución de este sistema se asume que:

- El objeto se desplaza solo en forma horizontal P: (x,y) por lo que la altura z a la que se encuentra siempre será la misma y no interesa, con lo que se asume la coordenadas z_1, z_2 y z_3 de los puntos de referencia como ceros.
- Las esferas se encuentran en puntos de referencia que faciliten el análisis matemático del sistema: P_1:(0,0), P_2:(x_2,0) y P_3:(0,y_3). [Ver figura 2.15]

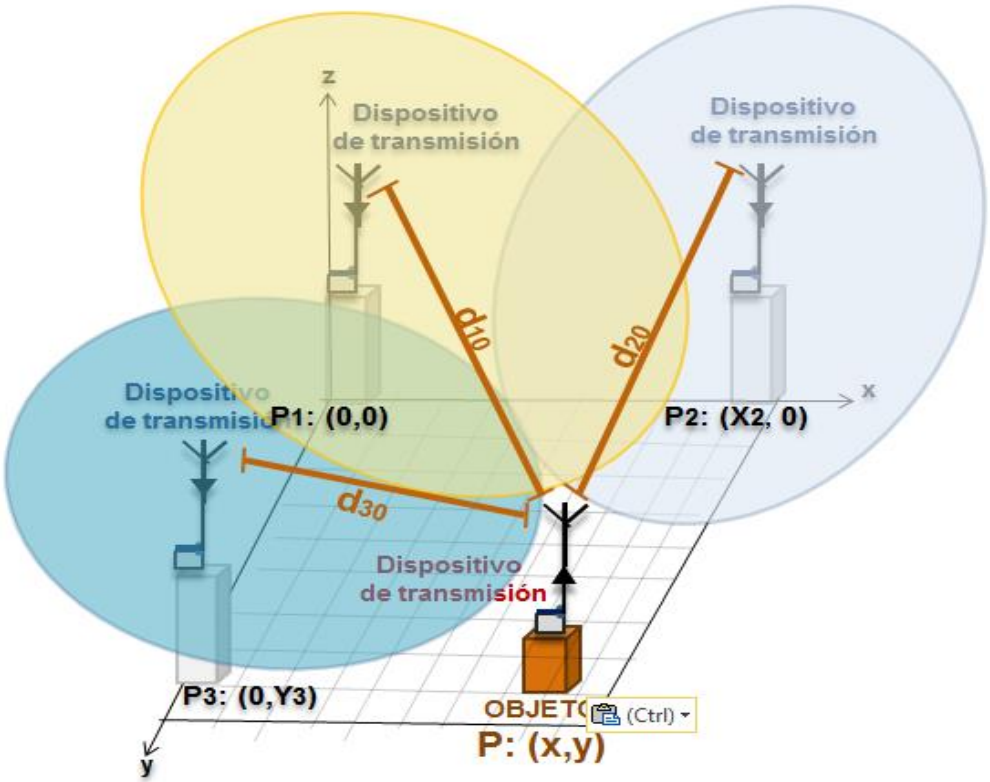


Figura 2.15 Esquema de Localización por Trilateración Reducido

Donde el sistema de ecuaciones queda expresado de esta forma:

$$\mathrm{d_{10}}^2 = \mathrm{x}^2 + y^2 + \mathrm{z}^2 \qquad (7)$$

$$\mathrm{d_{20}}^2 = (\mathrm{x} - \mathrm{x_2})^2 + \mathrm{y}^2 + \mathrm{z}^2 \qquad (8)$$

$$d_{30}{}^2 = x^2 + (y - y_3)^2 + z^2 \qquad (9)$$

Para resolver este sistema se resta la ecuación (8) de la ecuación (7) con lo que obtenemos la coordenada x del objeto.

$$x = \frac{d_{10}{}^2 - d_{20}{}^2 + x_2{}^2}{2.x_2} \qquad (10)$$

Al sustituir la ecuación (10) en la ecuación (7) se obtiene:

$$y^2 + z^2 = d_{10}{}^2 - \frac{\left(d_{10}{}^2 - d_{20}{}^2 + x_2{}^2\right)^2}{4.(x_2)^2} \qquad (11)$$

Al igualar la ecuación (11) con la ecuación (9) se obtiene la coordenada y del objeto.

$$\mathrm{y} = \frac{\mathrm{d_{10}}^2 - \mathrm{d_{30}}^2 + \mathrm{y_3}^2}{2.\mathrm{y_3}} - \frac{1}{\mathrm{y_3}}\,\mathrm{x} \qquad (12)$$

El objeto se encuentra en la posición P:(x,y) con las coordenadas determinadas en las ecuaciones (10) y (12).

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN

El objetivo de este capítulo es describir los dispositivos, modelos y programas para implementar el sistema de localización de objetos. Primero se detallan las características de los computadores utilizados en la implementación del sistema, luego se muestran las características de los dispositivos de transmisión que se utilizan, su configuración por computador y el funcionamiento del estándar que les permite comunicarse. Más adelante se describe el funcionamiento del sistema por medio de diagramas de flujo y finalmente se presentan los posibles modelos para diseñar el sistema de localización y se define el modelo más adecuado para implementarlo.

3.1 COMPUTADORES DEL SISTEMA

Los computadores que se utilizan para implementar el sistema de localización de objetos en tiempo real no presentan requisitos de hardware específicos que afecten de alguna manera el comportamiento del sistema, mientras que la tecnología que se utilice en el sistema esta definida de la siguiente forma:

- **Sistema Operativo Linux:** Se escogió Linux porque facilita la interacción entre procesos y recursos del sistema de esta forma:
  - El kernel de Linux presenta funcionalidades para programación de planificación en soft real time desde la versión 2.4.
  - Se representan los procesos y recursos del computador como archivos de texto simples ubicados jerárquicamente en el sistema de archivos.
  - Cada archivo que representa los procesos y recursos del computador presenta un descriptor que es un número clave entero por medio del que un proceso le pide al kernel acceso a un archivo específico del sistema.

- **Lenguaje de programación C:** Se utiliza lenguaje C puesto que:
  - Presenta funciones de programación en bajo nivel fáciles de utilizar para manipular a los dispositivos de transmisión y recepción utilizados en el proyecto.
  - Presenta la Librería POSIX la cual permite a Linux trabajar con procesos, hilos y la sincronizar estos, para obtener la respuesta del sistema eficiente en un tiempo adecuado y conocido.

3.2 DISPOSITIVOS DE RADIO XBEE

Los dispositivos de transmisión y recepción de señales que se utilizaron son los radios xbee, creados por la Empresa Estadounidense de Soluciones Tecnológicas Inalámbricas Digi, los cuales presentan una antena omnidireccional que opera en el rango entre 2.400 y 2.480 [GHz], como se muestra en la tabla 3.1. [10]

Tabla 3.1 Datos Teóricos de los xbee S1

Fuente: sparkfun.com/product

S1 ESTÁNDAR	S1 PRO
Alcance: - Con Línea de Vista: 90 m - Sin Línea de Vista: 30 m.	**Alcance:** - Con Línea de Vista: 1.6 km - Sin Línea de Vista: 90 m.
Antena tipo Chip: - Potencia de Recepción (P_{RIN}): 1[mW] - Radiación de señales: Irregular 	**Antena tipo Cable:** - Potencia de Recepción (P_{RIN}): 10[mW] - Radiación de señales: Regular 
Símbolo General de S1: 	**Símbolo General de S1-PRO:** 

3.2.1 IEEE 802.15.4.

IEEE 802.15.4 es un estándar de comunicación inalámbrica de red de área pequeña creado en el año 2005 por el grupo Zigbee y el IEEE 802 que permite a los radios xbee transmitir y recibir datos entre ellos. Surgió como una alternativa de bajo costo a otros estándares como: IEEE 802.11 y Bluetooth, con una baja tasa de transmisión de datos. [11]

Este estándar esta basado en la capa física y de enlace de datos del modelo de comunicación por capas OSI en el que en cada capa se verifica o asigna información a los datos para controlar eficientemente su transmisión entre los xbee. [Ver tabla 3.2]

Tabla 3.2 Modelo de Comunicación del estándar IEEE 802.15.4.
Fuente: http://docente.ucol.mx/al950441/public_html/osi1hec_B.htm

APLICACIÓN:	Programas que brindan servicios a los dispositivos que comparten datos.
PRESENTACIÓN:	Traduce los datos al lenguaje de los dispositivos.
SESIÓN:	Establece el enlace de comunicación entre dispositivos.
TRANSPORTE:	Verifica la fiabilidad de los datos transmitidos.
RED:	Paquete con dirección lógica IP orígen y destino de datos.
ENLACE DE DATOS:	Dirección física de orígen y destino de la señal transmitida.
FÍSICA:	Medio físico de transmisión y adaptación de la señal a este medio.

3.2.2 Capa Física en el estándar IEEE 802.15.4.

En la Capa Física se describe la estructura física del radio xbee, el mismo que presenta una distribución de pines así:

- **Pin 1 (Vcc):** Se conecta a la fuente que polariza al xbee.
- **Pin 2 (Buffer de Datos):** Presenta un buffer para ingresar 1 dato de 100 bytes y transmitirlo a otro u otros xbee.
- **Pin 3 (Buffer de Comandos):** Presenta un buffer para ingresar 1 comando de 100 bytes enviado por un computador conectado al xbee y que ejecute la acción del comando.
- **Pin 6 (RSSI):** Reporta la RSSI del dato en el buffer de salida de datos de un xbee, que se define como la relación entre la potencia recibida P_{ROUT} real y teórica en su antena:

$$RSSI = 10.\log(P_{RIN} / P_{RIN\,(Teórica)})\ [dB] \quad (13)$$

  Como los xbee S1 y S1 PRO presentan $P_{RIN(Teórica)}$ diferentes, las RSSI que reportan también son diferentes. [Ver tabla 3.1]

- **Pin 9 (DTR):** Indicador de xbee listo para recibir comandos.
- **Pin 10 (GND):** Se conecta a la tierra que polariza al xbee.
- **Pin 16 (RTS):** Indicador de xbee listo para enviar datos.

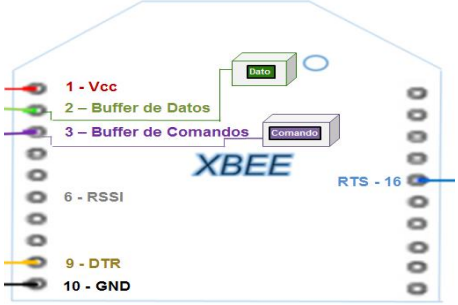


Figura 3.1 Estructura de los radios xbee S1

El dato o comando emitido por estos radios xbee es representado por una señal de bits 0 y 1 de periodo de tiempo de repetición T que se prepara para transmitirse por medio de 2 métodos:

- **Codificación de canal:** Técnica de Espectro Ensanchado por Secuencia Directa (DSSS) que aplica una secuencia de Barker, en la que los bits 0 se representan como una combinación 101010, mientras que los bits 1 se representan como una combinación 010101, con el objetivo de que los bits transmitidos no sean tan susceptibles a interferencias que cambien el contenido de la información significativamente.

- **Modulación/Codificación de senal:** Técnica de modulación de Desplazamiento de fase por Cuadratura compensada (O-QPSK) en la que se separa la senal en señales de bits pares (I) e impares (Q), se codifican mediante un código NRZ en el que la senal no se vuelve a 0 entre bits consecutivos de valor 1. Luego I y Q se combinan con señales seno y coseno, y se suman para formar una señal sin desfases mayores de 90 grados puesto que estos requieren mucha potencia para transmitirse. [Ver figura 3.2]

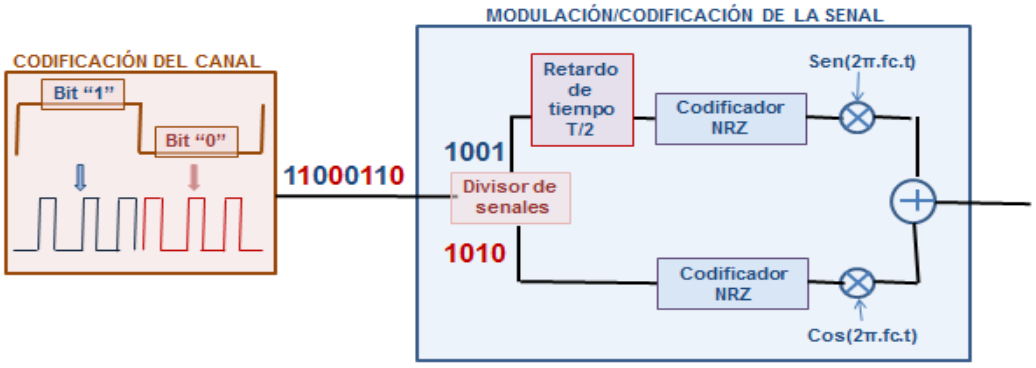


Figura 3.2 Capa Física en el estándar IEEE 802.15.4.

3.2.3 Capa de Enlace de datos en el estándar IEEE 802.15.4.

En la capa de enlace de datos se establece que cada xbee presenta las siguientes direcciones expresadas en 4 bits hexadecimales:

- **MY:** Dirección de cada xbee.
- **DL-DH:** Dirección de destino del dato almacenado en el xbee.
- **ID:** Dirección de red que integra un grupo de radios xbee.

Estas direcciones permiten dirigir un dato a 250 kbps de un xbee a otro por medio de un canal gracias al protocolo de Acceso Múltiple con Evasión de Portadora (CSMA-CA) que reparte los 80 MHz entre 16 canales de 5 MHz cada que van desde el número 11 al 26. [Ver tabla 3.3]

Tabla 3.3 Rango de Canales de Transmisión en estándar 802.15.4

Fuente: http://blogs.heraldo.es/ciencia/?p=1417

Canal	Hexadecimal	Frecuencia (GHz)
11	0x0B	2,4050
12	0x0C	2,4100
13	0x0D	2,4150
14	0x0E	2,4200
15	0x0F	2,4250
15	0x10	2,4250
17	0x11	2,4350
18	0x12	2,4400
19	0x13	2,4450
20	0x14	2,4500
21	0x15	2,4550
22	0x16	2,4600
23	0x17	2,4650
24	0x18	2,4700
25	0x19	2,4750
26	0x1A	2,4800

Esta cantidad de canales y el ancho de los mismos logra que no se generen interferencias entre los datos transmitidos en estos, a diferencia del IEEE 802.11 que crea 12 canales de 22 MHz que transmiten mucha información pero con interferencia, o el Bluetooth que crea 79 canales de 1 MHz cada uno lo que no genera interferencia pero no permite transmitir una cantidad significativa de información, como se ve en la figura 3.3. [12]

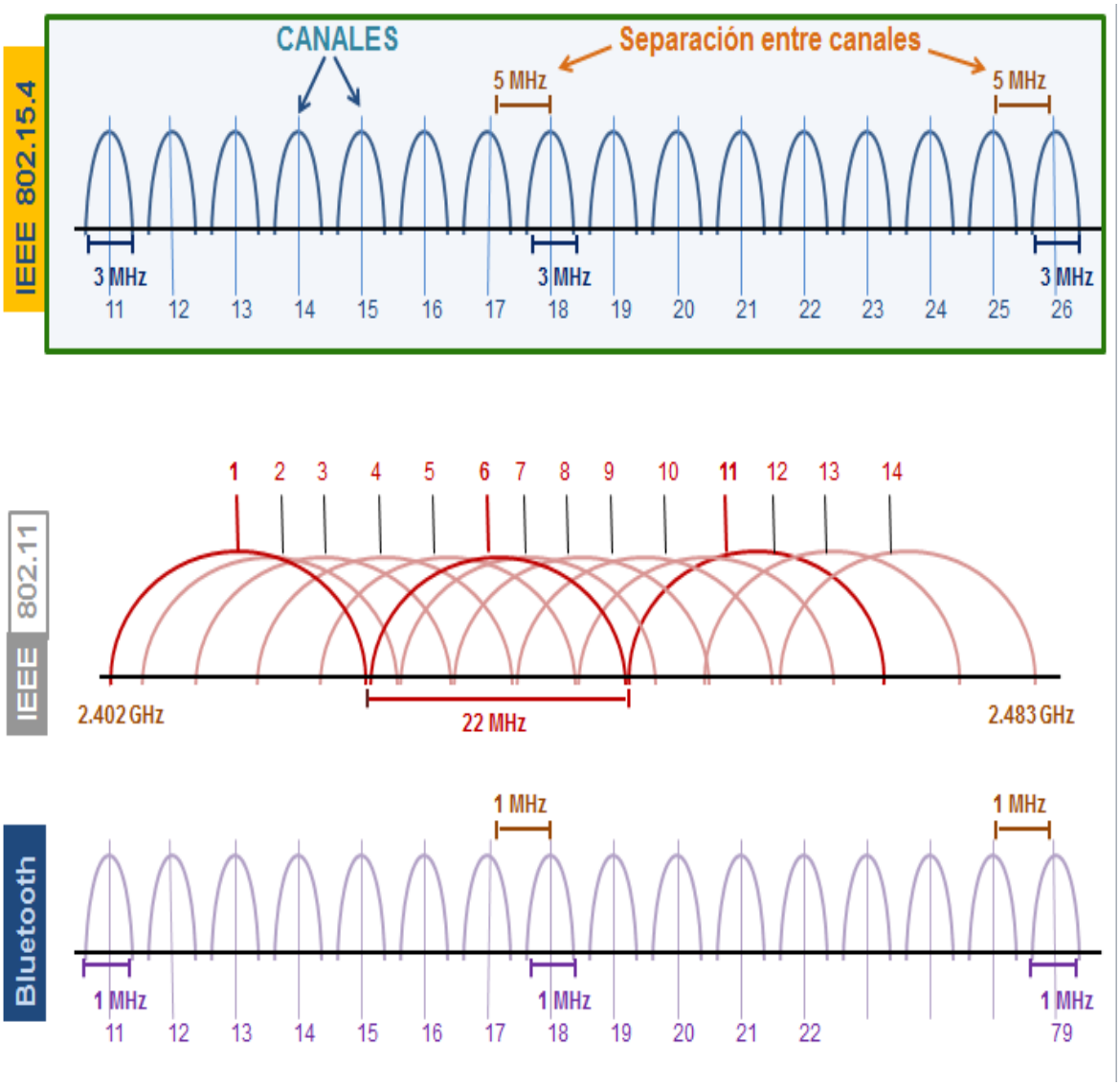


Figura 3.3 Distribución de canales en estándar 802.15.4

3.2.4 XBEE EXPLORER.

Los xbee no pueden polarizarse o realizar operaciones como transmitir datos o recibir comandos mediante sus buffers por sí mismos, por lo que la empresa Digi diseño una tarjeta llamada Xbee Explorer, que incorpora un microcontrolador programado que polariza al xbee al conectarse a un computador y le permite comunicarse vía USB a este. [Ver figura 3.4]

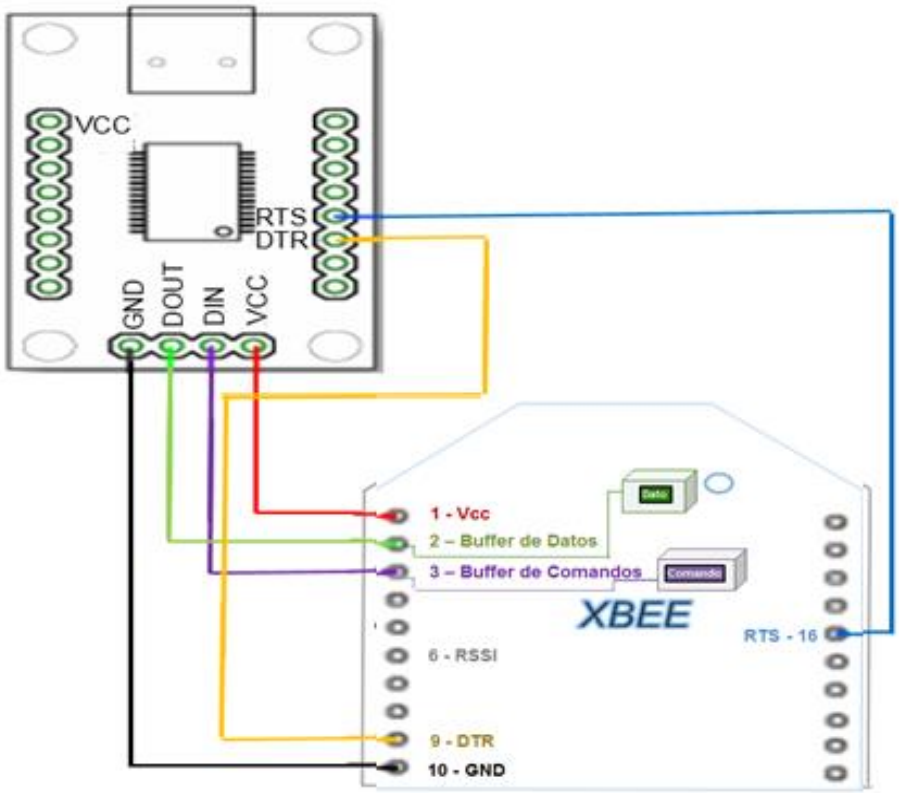


Figura 3.4 Tarjeta de Conexión a Computador Xbee Explorer.

El Linux instalado en el computador conectado al xbee identifica al xbee como un archivo de nombre USBi, donde i = 0, 1, 2, 3, etc, de acuerdo al puerto USB en el que este conectado este radio.

Este computador ejecuta un programa escrito en C que le permite al xbee recibir comandos o transmitir datos a otro xbee, para lo cual se debe manipular sus buffers por medio de las siguientes funciones de bajo nivel disponibles en la librería <fcntl.h>:

- **Acceso a buffers del xbee:** int open (char* path, int oflag) que retorna su descriptor, en el que path es la ruta del puerto conectado al xbee y oflag una constante que indica si quiere leer (O_RDONLY), ingresar (O_WRONLY) datos o ambas. (O_RDWR)

- **Ingresar dato/comando en el buffer del xbee:**
  int write (int fd, void* buf, int nbytes) que toma los primeros nbytes de la cadena buf, los almacena en el buffer respectivo del xbee cuyo descriptor es fd y retorna el número de bytes ingresados.

- **Leer dato/comando del buffer del xbee:**
  int read (int fd, void *buf, int nbytes) que lee los nbytes del dato almacenado en el buffer del xbee cuyo descriptor es fd y retorna el número de bytes leídos del buffer.

Estas funciones se las utilizan para permitir al xbee operar en un modo en el que se puedan configurar sus direcciones y canales, o en otro modo en el que un xbee puede transmitir o recibir datos como se ve a continuación. [13]

3.2.5 Modo de configuración por comandos de un xbee.

En este modo se configuran las direcciones y canales del xbee por medio del ingreso de comandos en su buffer de comandos mediante la función write(). [Ver figura 3.5]

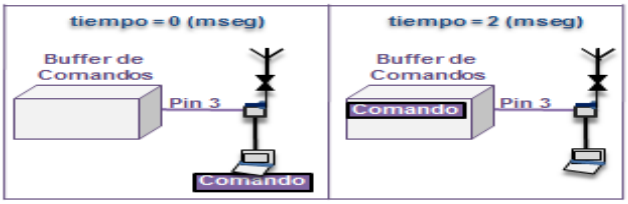


Figura 3.5 Ingreso de comando en un xbee

Primero para acceder a este modo se ingresa el comando "**+++**", y ya en este modo se pueden utilizar los siguientes comandos:

- **ATMY 3BF1:** Asigna 3BF1 como dirección del xbee.
- **ATDL 5FCC:** Asigna 5FCC como destino del dato del xbee.
- **ATID 1234:** Establece 1234 como dirección grupal de xbee.
- **ATCH 10:** Asigna el canal 15 a un xbee. [Ver tabla 3.3]

El xbee se tarda un tiempo en ingresar a este modo que por defecto es 1000 [mseg] (0x3e8 en hexadecimal). Este tiempo se puede configurar mediante el comando **ATGT 3E8**.

Para comprobar la realización correcta de las acciones de estos comandos, el xbee responde con un **"OK"** en su buffer de comandos.

También se obtiene la RSSI del dato del xbee con el comando **ATDB,** al que responde con la RSSI solicitada en su buffer de comandos.

Finalmente para salir del modo comando se ingresa **ATCN**.

3.2.6 Modo de transmisión de un xbee.

Es el modo habilitado por defecto de un xbee cualquiera en el que puede transmitir un dato almacenado en su buffer de datos a:

- **Unicast:** Otro xbee; En modo de configuración por comandos se asigna la dirección MY del xbee como destino DL del otro y luego el computador conectado al xbee ejecuta un programa que con la función write() ingresa un dato en su buffer y automáticamente lo transmite al otro xbee. [Ver figura 3.6]

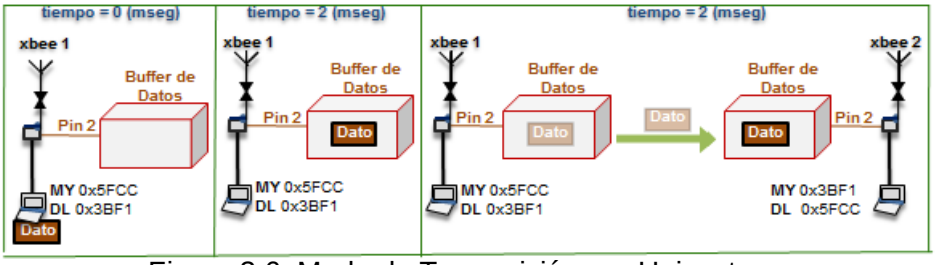


Figura 3.6 Modo de Transmisión por Unicast

- **Broadcast:** Varios xbee con el mismo ID y canal CH; el computador conectado al xbee ejecuta un programa que con la función write() ingresa un dato en su buffer y automáticamente lo transmite a los buffers de los xbees que tienen el mismo ID y canal CH. En los paquetes comerciales de xbee vienen por defecto con el mismo ID y CH. [Ver figura 3.7]

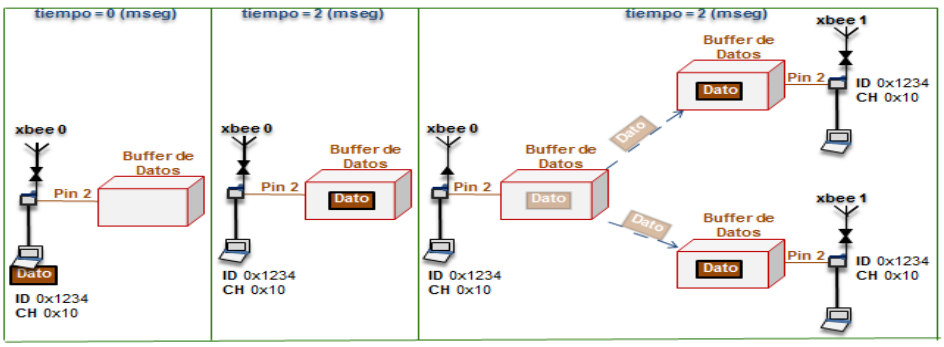


Figura 3.7 Modo de Transmisión por Broadcast

3.3 COMUNICACIÓN ENTRE PROCESOS POR SOCKETS TCP.

Para transmitir un dato entre un proceso en un computador y un proceso en otro computador se utiliza el método de comunicación por sockets TCP, por medio de las siguientes funciones disponibles en la librerías <sys/socket.h>, <sys/types.h>, <sys/time.h> y <unistd.h>:

- **Creación de un Socket en el computador cliente y en el servidor:**
  int socket (int dominio, int tipo, int protocolo) que retorna el descriptor de cada socket, en el que:
  - **dominio:** constante que define el protocolo de red utilizado, ya sea: IP (AF_INET) o comunicación interna (AF_UNIX).
  - **tipo:** constante que define el protocolo de transporte del socket ya sea: TCP(SOCK_STREAM) o UDP(SOCK_DGRAM)
  - **protocolo:** valor constante de cero.

- **Asignación de una dirección local al socket servidor**
  int bind (int sockfd, sockaddr_in *addr, int addrlen) que asigna al socket del servidor representado por el descriptor sockfd, la dirección addr de tamaño addrlen definida por la estructura sockaddr_in definida por los siguientes parámetros:
  - **sin_family:** Constante que indica el protocolo de red utilizado ya sea: IP (AF_INET) o comunicación interna (AF_UNIX).
  - **sin_port:** Entero Corto del puerto asignado al socket.
  - **sin_addr:** Entero largo de la dirección IP asignada al socket.

- **Apertura del socket del servidor para escuchar peticiones de conexión de clientes:**

  int listen (int sockfd, int nc) que hace que el socket del servidor definido por el descriptor sockfd espere por un número máximo de "nc" peticiones de clientes de conexión al servidor.

- **Petición de cliente para conexión con el servidor:**

  int connect (int sockfd, sockaddr_in* serv_addr, int addrlen) que le permite al socket del cliente definido por el descriptor sockfd enviar una petición de conexión al servidor cuyo socket está definido por la dirección serv_addr de tamaño addrlen.

- **Aceptación de la petición del cliente de conexión al servidor:**

  int accept (int sockfd, sockaddr* addr, int addrlen) que le permite al socket del servidor definido por el descriptor sockfd aceptar la petición de conexión del cliente cuyo socket está definido por la dirección addr de tamaño addrlen, para que este último quede habilitado para compartirle datos al servidor.

- **Limite de tiempo en recepción de datos en el servidor:**

  int select (int fd, fd_set* rwd, timeval* t) que comprueba si el cliente identificado por el descriptor fd, el cual se lee mediante la estructura rwd, presenta algún dato almacenado a compartir con el servidor en un tiempo t.

- **Cierre del socket del servidor para escuchar peticiones de conexión de clientes:**

  int close (int fd) que le permite al servidor, identificado por el descriptor fd, dejar de escuchar peticiones de conexión al mismo. si el cliente identificado por el descriptor fd. [14]

Estas funciones se utilizan para que uno o varios procesos clientes puedan compartirle datos a un proceso servidor de acuerdo al esquema que se observa en la figura 3.8.

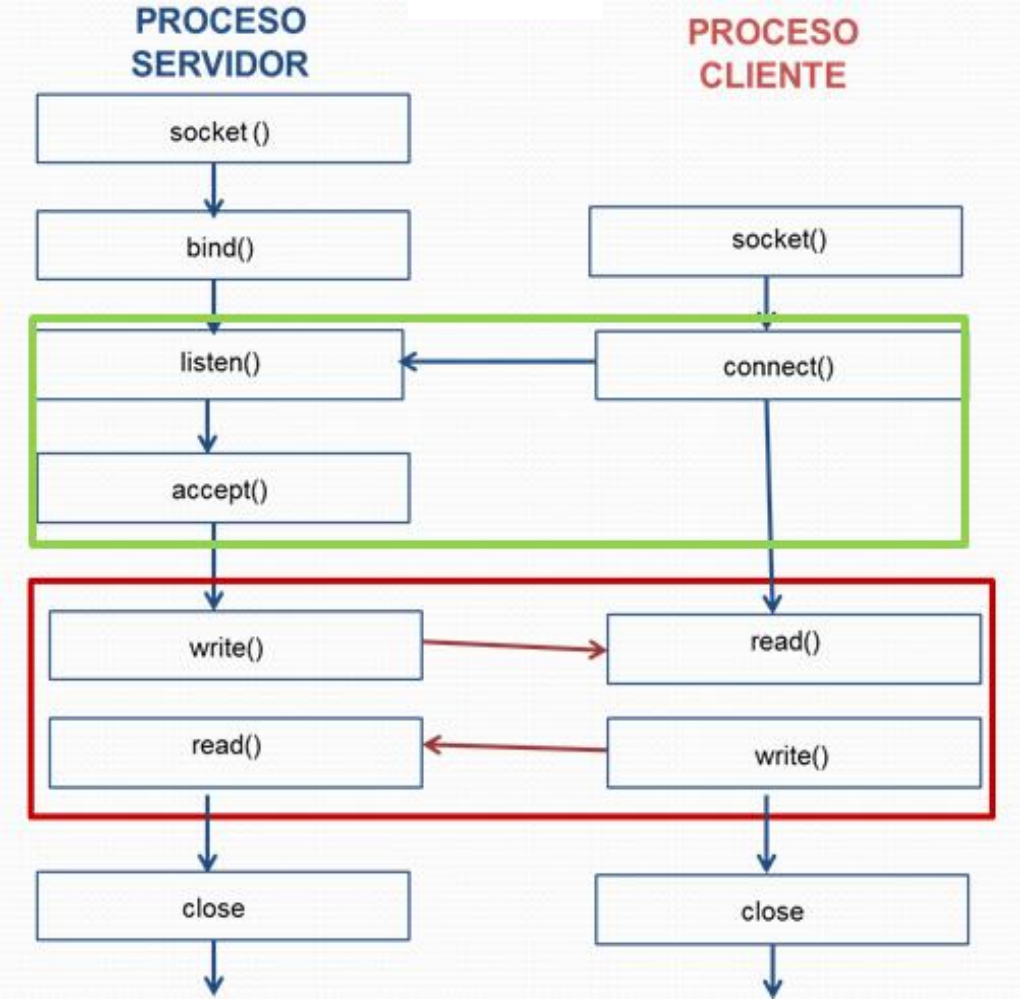


Figura 3.8 Comunicación entre procesos por sockets TCP

3.4 PLANIFICACIÓN EN LINUX EN TIEMPO REAL ACRÍTICO.

En los sistemas Linux hay dos tipos de procesos e hilos puntuales:

- **Normales:** Son los que presentan instrucciones de I/O o de cálculos que se vieron en la sección 2.2, y se ejecutan según sus prioridades.
- **En tiempo real:** Son los que presentan prioridades de ejecución mayores que las de los procesos normales.

El kernel Linux permite establecer dos políticas diferentes para procesos e hilos en tiempo real con igual prioridad, las cuales son:

- **Primero que entra, primero sale (FIFO):** Los procesos se ejecutan de acuerdo con su tiempo de llegada a la cola de procesos listos. Una vez que el proceso obtiene el CPU se apropia de este último, es decir se ejecuta hasta terminar. Puede ocasionar que procesos largos hagan esperar a procesos cortos y que procesos no importantes hagan esperar a procesos importantes, por lo que no garantizan buenos tiempos de respuesta interactivos.

- **Round Robin (RR):** Se asigna a cada proceso una porción de tiempo equitativa y ordenada, tratando a todos los procesos con la misma prioridad. El kernel le da un tiempo máximo de uso de CPU a cada proceso, pasado el cual es desalojado y retornado a la cola de procesos listos. [15]

Para implementar, sincronizar y planificar la ejecución de los hilos en tiempo real de un proceso en un computador se utilizan las siguientes funciones disponibles en la librería <pthread.h> y <sched.h>:

- **Crear hilo de ejecución:**

  int pthread_create (pthread_t *restrict thread, pthread_attr_t *restrict_attr, void * (*start_routine) (void*), void *restrict_arg), que devuelve el descriptor de un nuevo hilo de ejecución de un proceso con atributos restrict_attr, y con banderas de inicialización restrict_arg y (*start_routine) (void*) definidas normalmente como NULL.

- **Asignar esquema de planificación a hilos de ejecución:**

  void pthread_setschedparam (pthread_t thread, int policy, sched_param *param) que asigna la planificación definida por la constante “param” que puede ser: SCHED_FIFO para definir una política de planificación FIFO o SCHED_RR para definir una política Round Robin al hilo thread, y policy definido por defecto como cero.

- **Crear semáforo:** pthread_mutex_t PTHREAD_MUTEX_INITIALIZER es una función que permite inicializar un semáforo creado.

- **Destruir semáforo:** int pthread_mutex_destroy (pthread_mutex_t* mut) que destruye el semáforo mut.

- **Bloquear semáforo:** int pthread_mutex_lock (pthread_mutex_t* mut) que bloquea temporalmente las acciones del semáforo mut de negar el acceso de una de dos variables que quieran acceder a un mismo espacio de memoria. [16]

3.5 DISEÑO DEL SISTEMA

Para diseñar un sistema de localización de objetos en tiempo real se ubica un xbee sobre el objeto a localizar y tres fijos en los extremos más altos del lugar de localización, donde cada xbee tiene conectada una computadora. Estas computadoras ejecutan programas de transmisión y recepción de datos, que permiten obtener las RSSI de los datos transmitidos entre el xbee sobre el objeto y cada xbee fijo, y enviarlos por socket TCP a un computador central.

El computador central ejecuta un programa que por socket TCP recibe las tres RSSI y genera un hilo que basado en la RSSI halla la distancia del xbee sobre el objeto a cada fijo, para estimar la posición del objeto por trilateración.

Este diseño de sistema puede ser implementado con 2 modelos:

- **Transmisor sobre el objeto:** El computador conectado al xbee sobre el objeto ejecuta un programa de transmisión de un dato en modo broadcast hacia cada xbee fijo. El computador conectado a cada xbee fijo ejecuta un programa de recepción de este dato, extrae su RSSI, y la envía por socket TCP a un computador.[Ver figura 3.8a]

- **Receptor sobre el objeto:** El computador conectado a cada xbee fijo ejecuta un programa de transmisión de un dato en modo unicast hacia el xbee sobre el objeto. El computador conectado al xbee sobre el objeto ejecuta un programa de recepción de los datos, extrae sus RSSI y las envía por socket TCP a un computador.[Ver figura 3.8b]

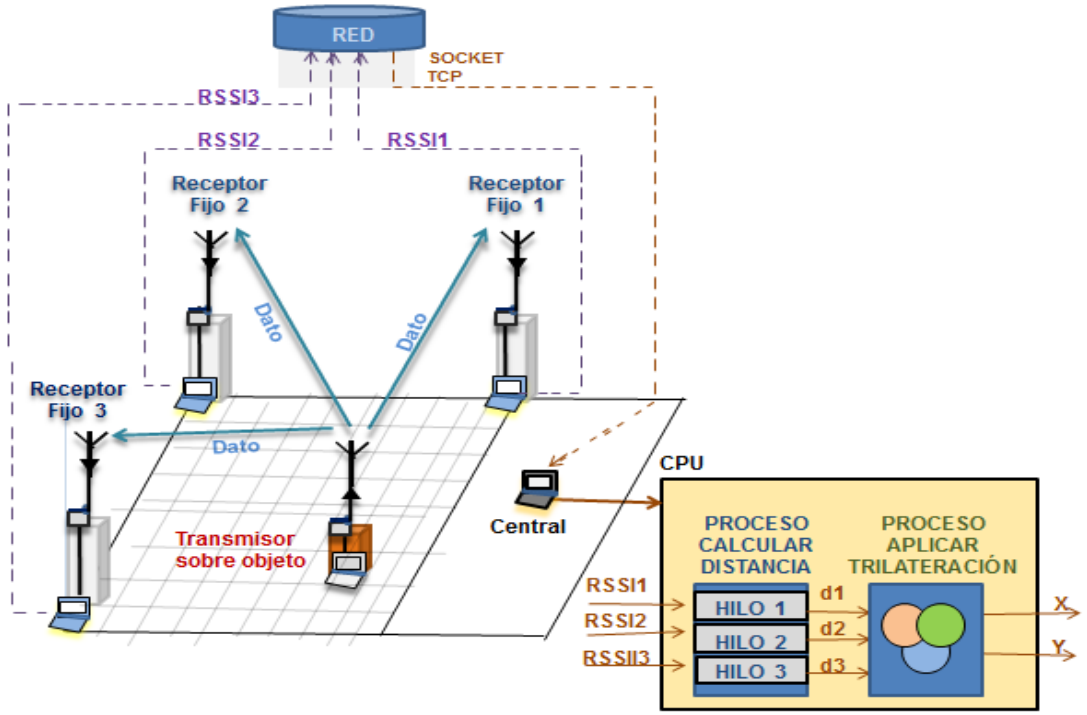


Figura 3.9a Modelo de xbee transmisor en el objeto

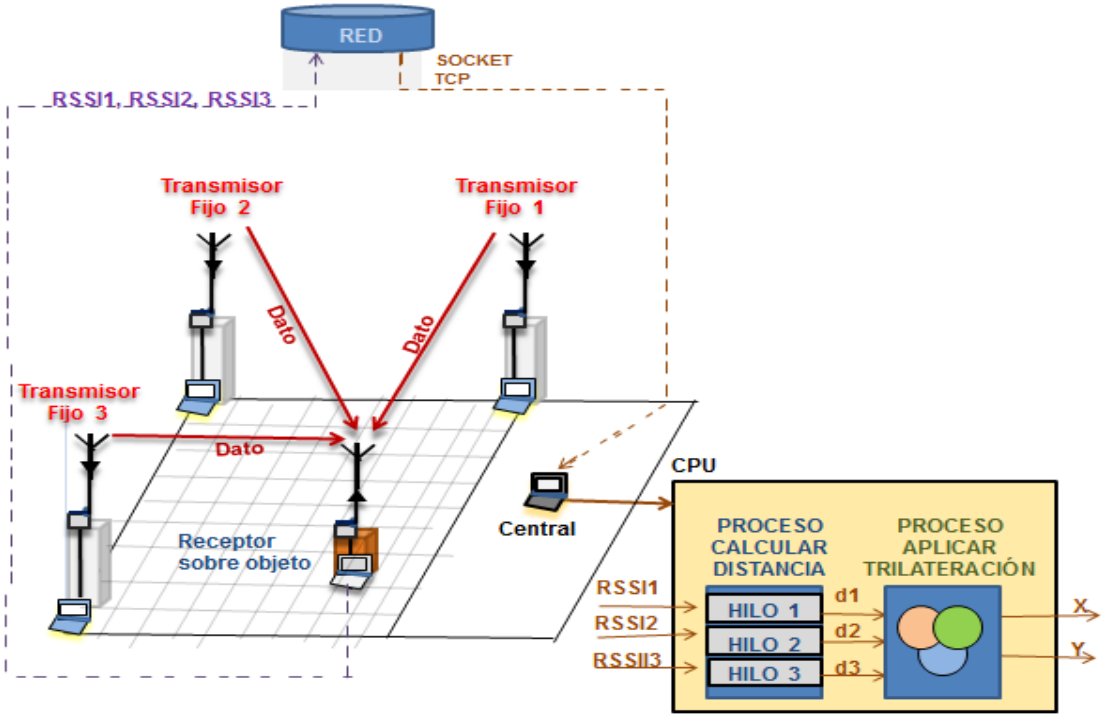


Figura 3.9b Modelo de xbee receptor en el objeto

Los programas que se ejecutan en los computadores que se utilizan para implementar el sistema son los siguientes:

Programa Transmisor.

Programa que permite a un xbee almacenar un dato “Xi” cada 50 [mseg] en su buffer de datos y transmitirlo automáticamente en modo broadcast a todos los xbees configurados en este modo. La cantidad máxima de datos que se transmiten se define por la constante NUM_DATOS del programa. [Ver figura 3.10 y Anexo 3.1]

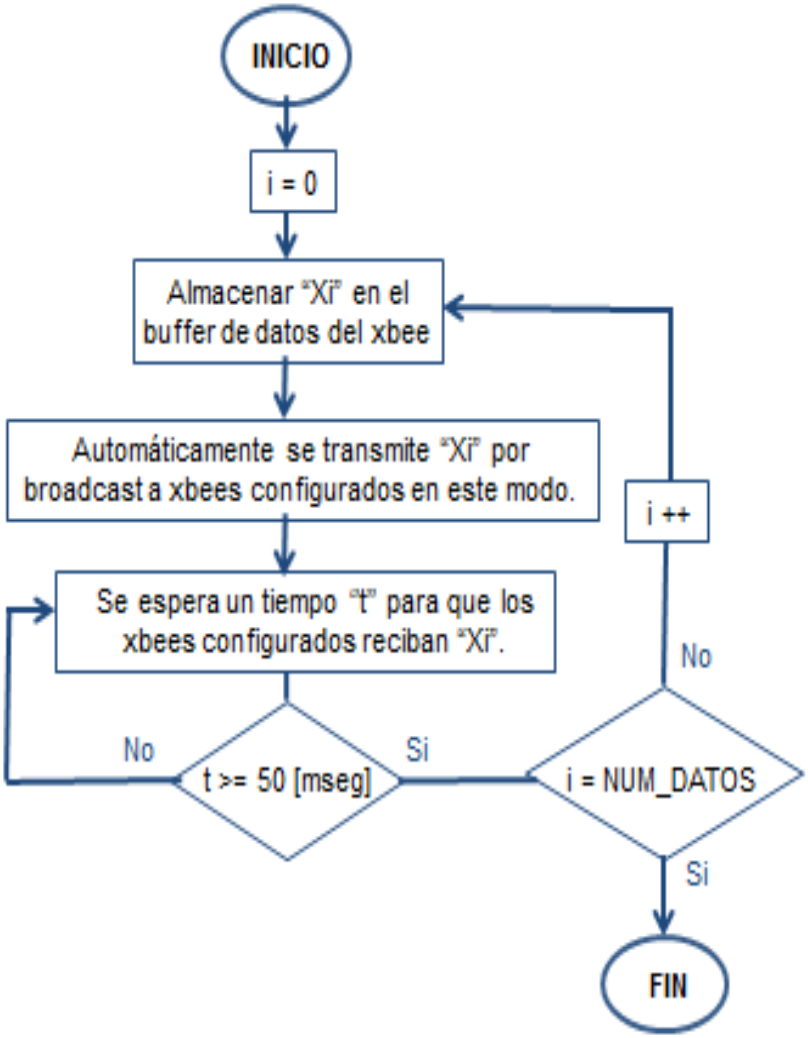


Figura 3.10 Diagrama de flujo del programa transmisor

Programa Receptor.

Programa que modifica el tiempo de guarda del xbee por defecto de 1[seg], para cambiarse de modo de transmisión a configuración o viceversa, a un tiempo de 50[mseg], y luego realiza una de estas dos acciones:

- Si no hay un dato en el buffer del xbee espera por un dato.
- Si hay un dato en el buffer del xbee, se lee el dato, se extrae su RSSI y la envía por socket TCP a un computador, todo esto en un tiempo de 50 [mseg]. [Ver figura 3.11 y Anexo 3.2]

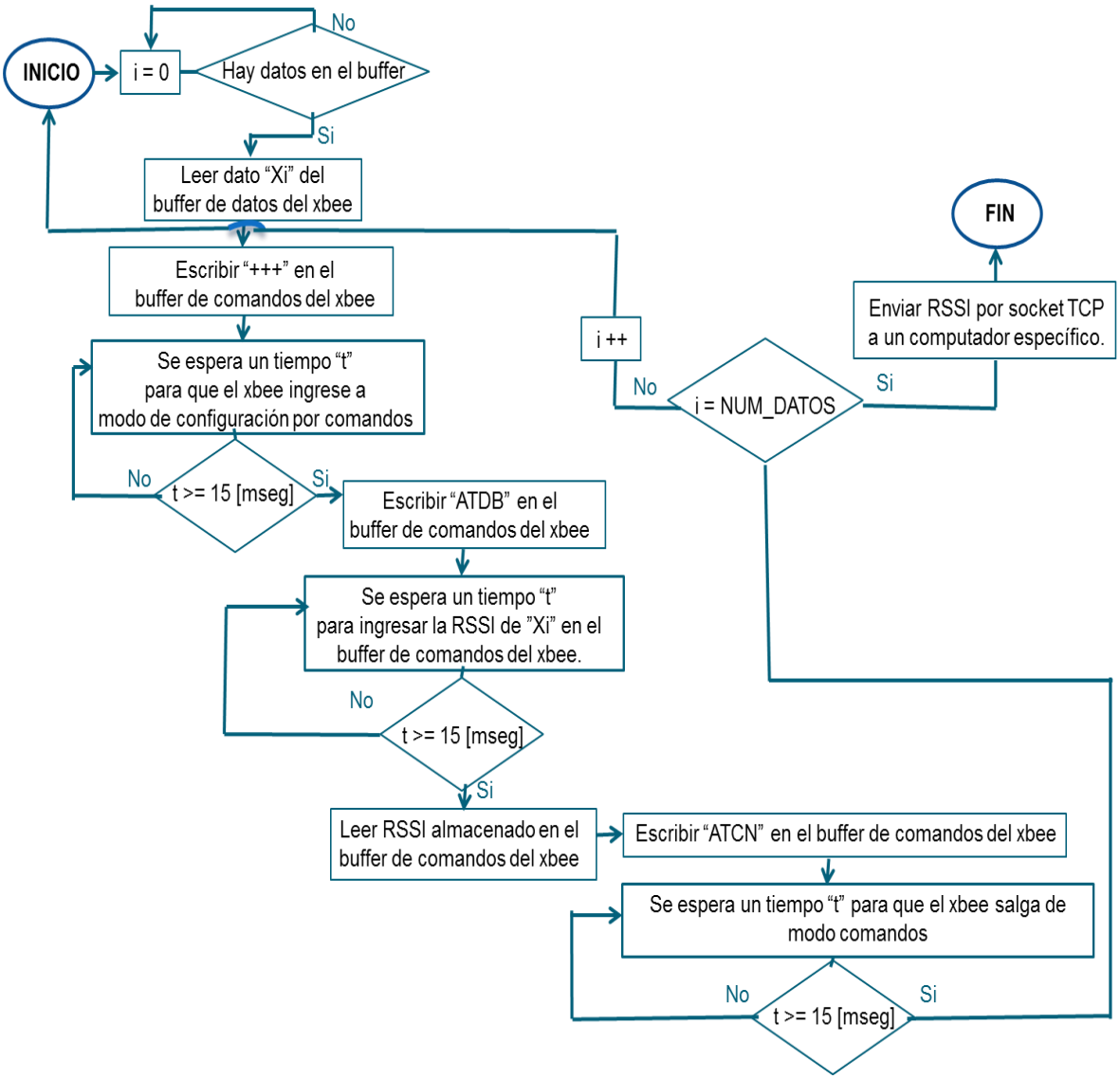


Figura 3.11 Diagrama de flujo del programa receptor

Programa Central.

Programa ejecutado en el computador central que espera por recibir las 3 RSSI provenientes de los xbee fijos por medio de sockets TCP, para luego generar y sincronizar por medio de una variable semáforo a 3 hilos, bajo planificación en tiempo real round robin, que calculen las distancias de cada xbee fijo al objeto, y aplica trilateración para estimar la posición (x,y) del objeto. [Ver figura 3.12 y Anexo 3.3]

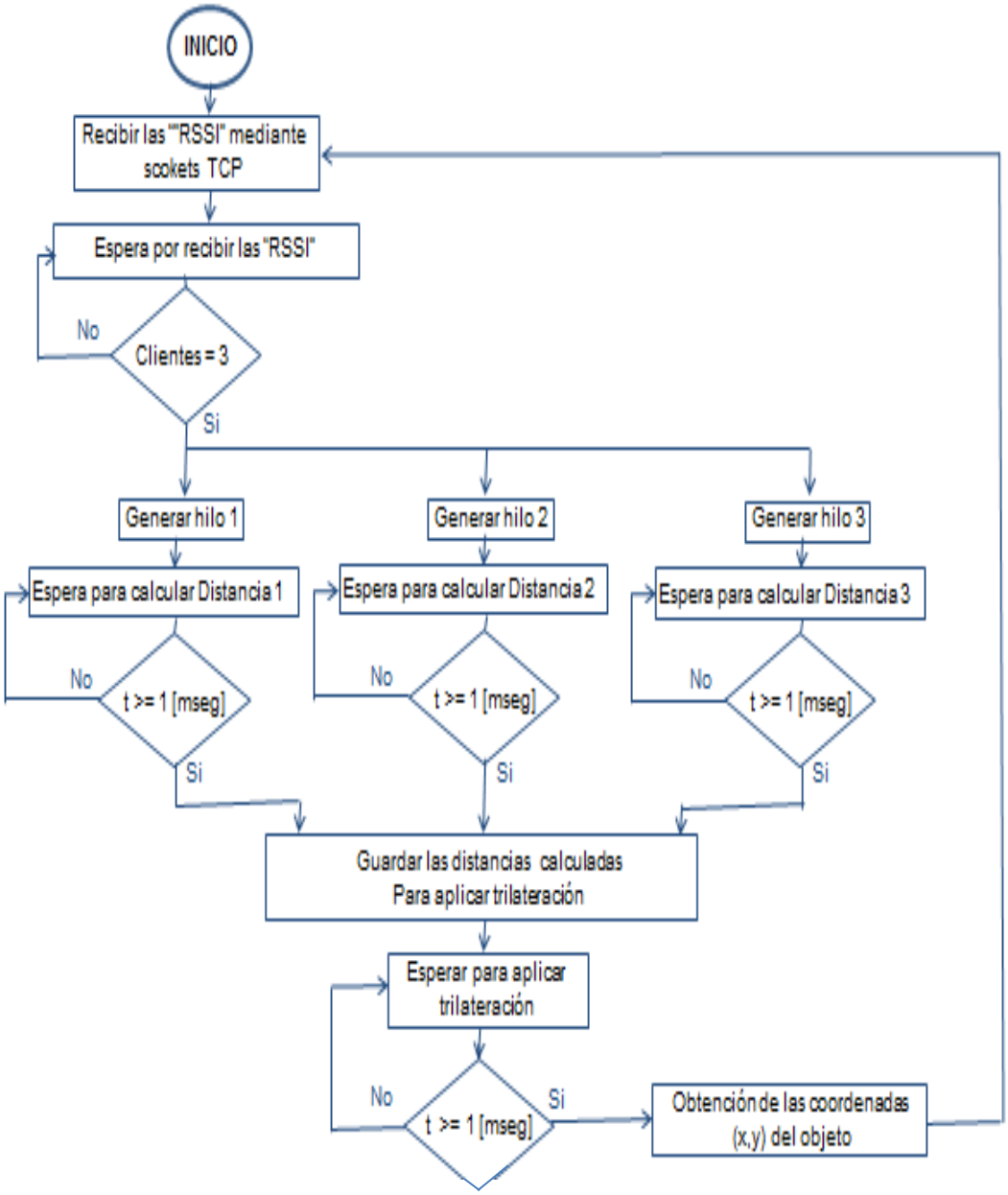


Figura 3.12 Diagrama de flujo del programa central.

Los modelos de transmisor y receptor sobre el objeto utilizan un computador que ejecuta un programa central que genera hilos que calculan las distancias basandose en las RSSI recibidas y con estas aplica el método matemático de trilateración; con el modelo del receptor sobre el objeto estas acciones pueden ser realizadas por el mismo programa receptor en el computador conectado al xbee sobre el objeto, lo que quita la necesidad de un computador central y ya no se necesita comunicación por sockets TCP, sin embargo las RSSI transmitidas por cada xbee fijo pueden interferirse entre ellas y para solucionar este problema el programa receptor sería más complejo en su implementación.

En el modelo del transmisor sobre el objeto cada xbee fijo recibe una RSSI que se almacena en su respectivo computador conectado, por lo que estas RSSI no se interfieren entre sí, a pesar de que en este modelo si se necesita un computador central y comunicación con este por sockets TCP, sin embargo se hace más simple el desarrollo del programa receptor en cada xbee fijo, por lo que se escoge el modelo del transmisor sobre el objeto para implementar el sistema de localización de objetos de este proyecto.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

El objetivo de este capítulo es mostrar los resultados de las pruebas realizadas para verificar la viabilidad de implementar el sistema de localización de objetos en el laboratorio del CVR de la ESPOL. Primero se muestran los resultados de la prueba de Distancia versus RSSI en la que se transmite una cantidad de datos de un radio xbee hacia otro, para por medio de la potencia promedio de las señales transmitidas hallar una relación Distancia = f (RSSI), que aplique el programa central del sistema para estimar la posición del objeto a localizar, y luego se muestran los resultados de la prueba de localización de un objeto en movimiento en la que se verifica la precisión y tiempo de respuesta del sistema, luego de aplicar técnicas de soft real time en el programa central del sistema.

4.1 CONDICIONES EXPERIMENTALES.

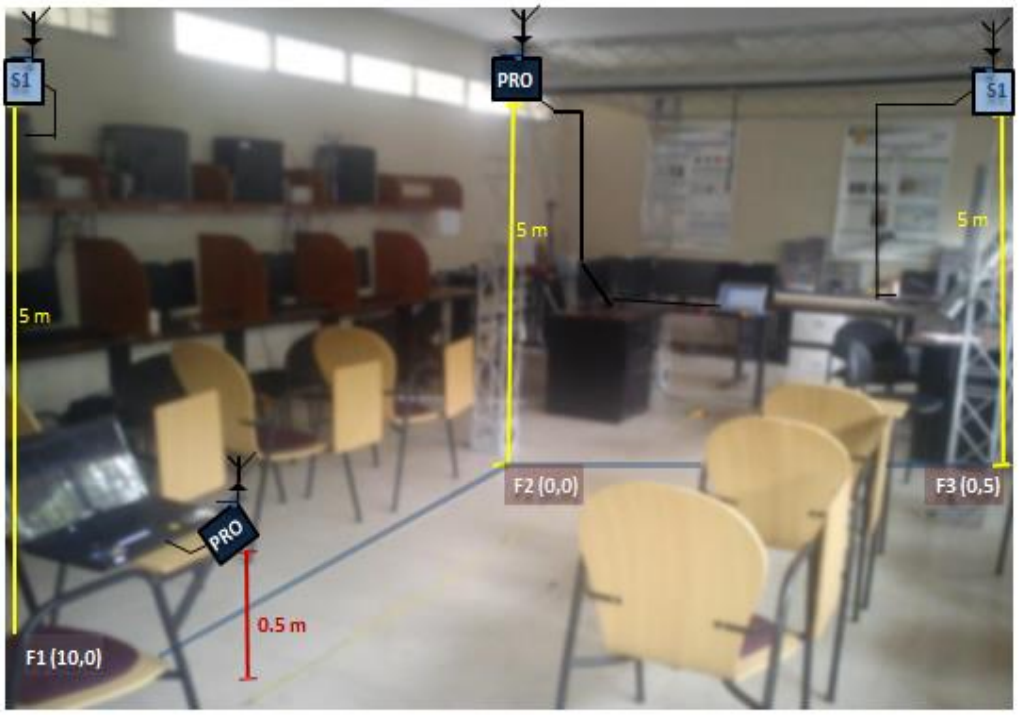
El sistema se implementó en el laboratorio del Centro de Visión y Robótica de la ESPOL que tiene un área de 5 x 10 [m^2] y 5 [m] de altura, para localizar una silla de 0.5 [m] de altura,

Una portátil conectada a un xbee ubicado sobre la silla ejecuta el programa transmisor para que el xbee transmita un dato a tres xbee fijados en las esquinas altas del laboratorio en posiciones: F1:(0,0), F2:(0,10) y F3:(5,0), como se puede observar en la figura 4.1a.

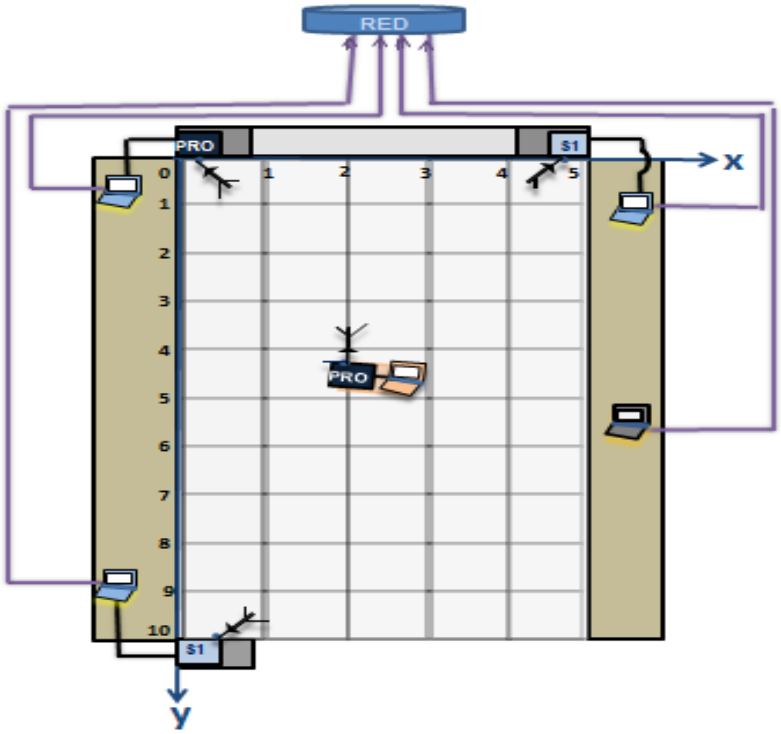
Cada uno de los xbee fijos está conectado a un computador que ejecuta el programa receptor que extrae la RSSI del dato recibido y por medio de un socket TCP envía la RSSI al programa central alojado en otra portátil.

Esta portátil ejecuta el programa central que recibe las tres RSSI y ejecuta 3 hilos al mismo tiempo que calculan las distancias de cada xbee fijo al objeto en función de estas RSSI y con estas distancias por medio de trilateración estima la posición del objeto en el plano x versus y, como se puede observar en la figura 4.1b, en la que se divide al laboratorio del CVR en cuadros de 1 metro de ancho y largo, para representar las posibles posiciones (x,y) del objeto a localizar.

No se espera que el sistema sea completamente preciso al estimar la posición del objeto, debido a que el laboratorio tiene varios objetos metálicos en su interior que producen muchas pérdidas de potencia que afectan a las distancias utilizadas en el método de trilateración para ubicar al objeto. Se espera manipular variables y funciones de programación en soft real time en el programa central para permitir que el sistema funcione en un tiempo lo suficientemente rápido, que permita realizar un análisis de corrección de errores en un futuro proyecto que mejore la precisión del sistema.



(a) Vista Panorámica



(b) Vista plano x versus y

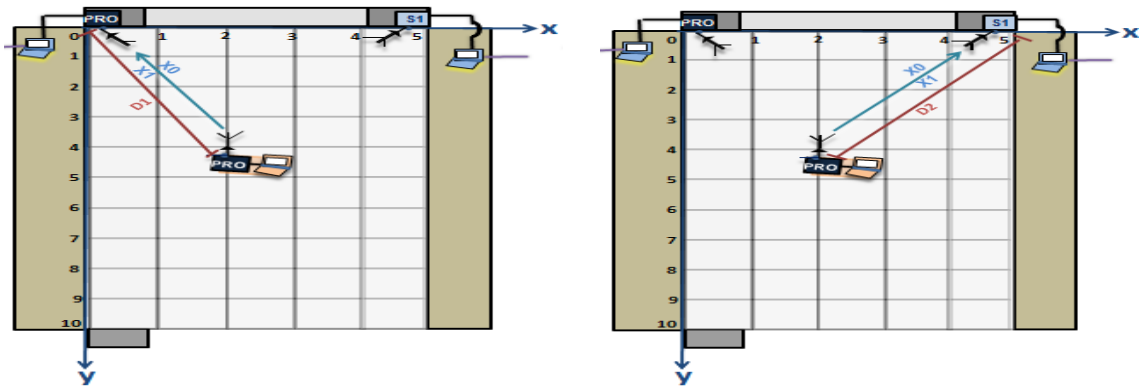
Figura 4.1 Modelo de transmisor en el laboratorio del CVR

4.2 PRUEBA DE RSSI VERSUS DISTANCIA

El xbee S1-PRO sobre la silla transmite una trama de datos X_0,X_1 hacia:

- Un xbee fijo S1-PRO ubicado a una distancia D1 que varía desde 20[cm] hasta 5[m]. Esto procedimiento se repite 10 veces y el programa receptor en este xbee fijo obtiene la RSSI promedio de los datos para cada distancia D1, con los que se puede generar una gráfica de RSSI versus Distancia: $RSSI1 = f(D1)$. [Ver figura 4.2a]
- Un xbee fijo S1 ubicado a una distancia D2 que varía desde 20[cm] hasta 5[m]. Este procedimiento se repite 10 veces y el programa receptor en este xbee fijo obtiene la RSSI promedio de los datos para cada distancia D1, con los que se puede generar una gráfica de RSSI versus Distancia: $RSSI2 = f(D2)$. [Ver figura 4.2b]

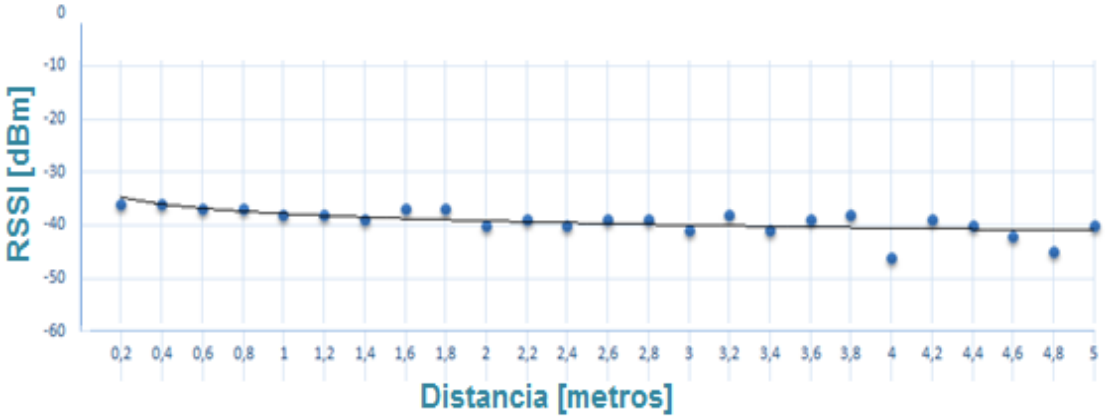
Además con la teoría de propagación de errores mencionada en la sección 2, se obtuvo la incertidumbre en los valores de RSSI y distancias generados en el laboratorio representados por las desviaciones σ_{RSSI} y σ_{D}. Esta información es útil para un análisis de corrección de errores a realizar en un futuro proyecto.



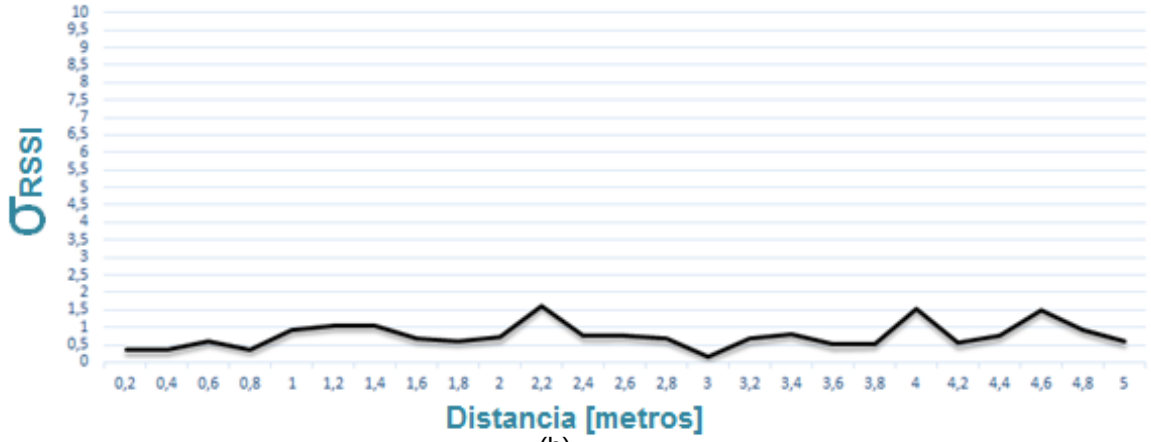
(a) Transmisión S1-PRO a S1-PRO. (b) Transmisión S1-PRO a S1.

Figura 4.2 Esquema de Prueba de RSSI versus Distancia

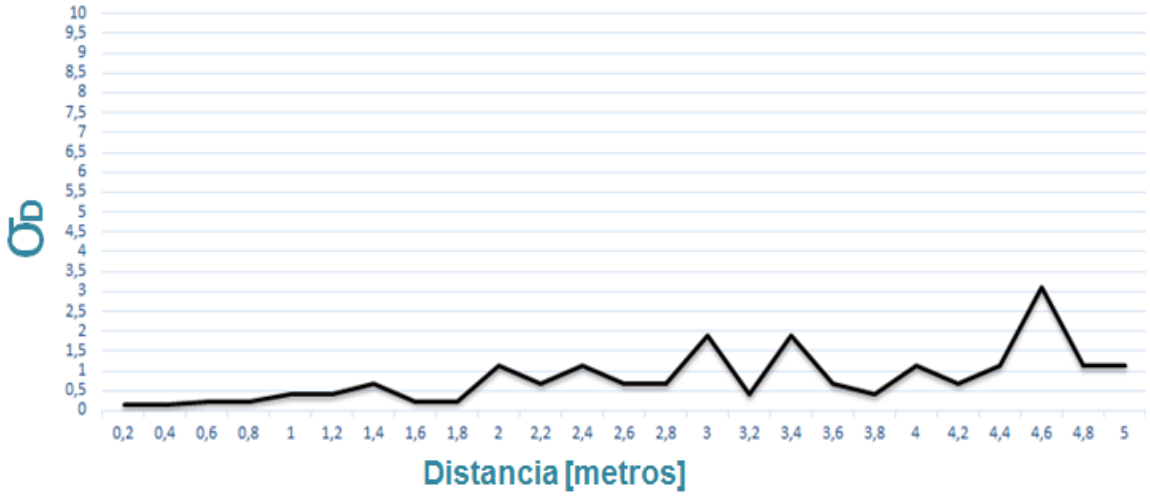
Primero se obtuvo la figura 4.3a: RSSI1 = f(D1) = -1.975.ln(D1) - 37.837. Las figuras 4.3b: σ_{RSSI1} = f(D1) y 4.3c: σ_{D1} = f(D1) muestran la incertidumbre de la RSSI1 y D1 respectivamente.



(a)



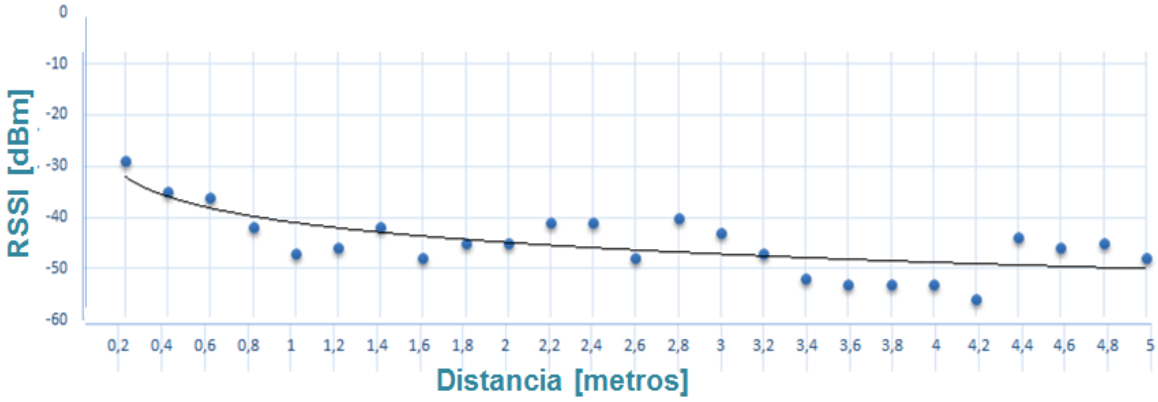
(b)



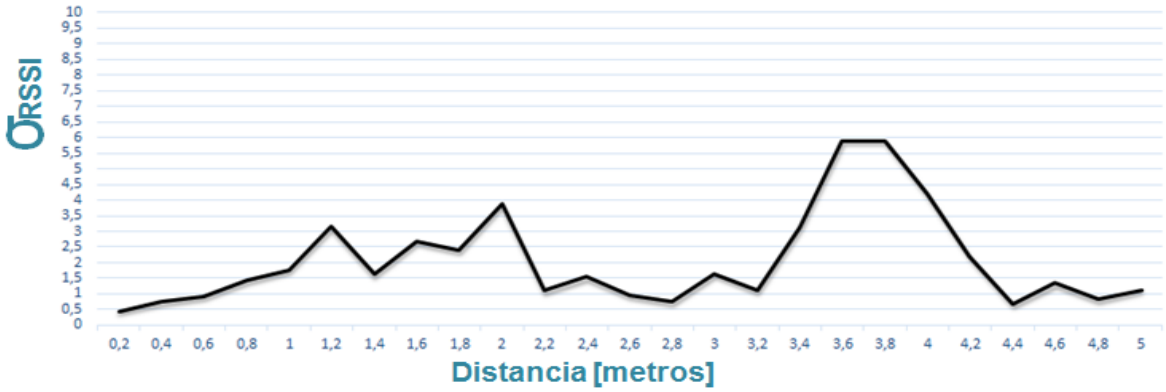
(c)

Figura 4.3 Resultados de prueba RSSI versus Distancia con fijo S1-PRO

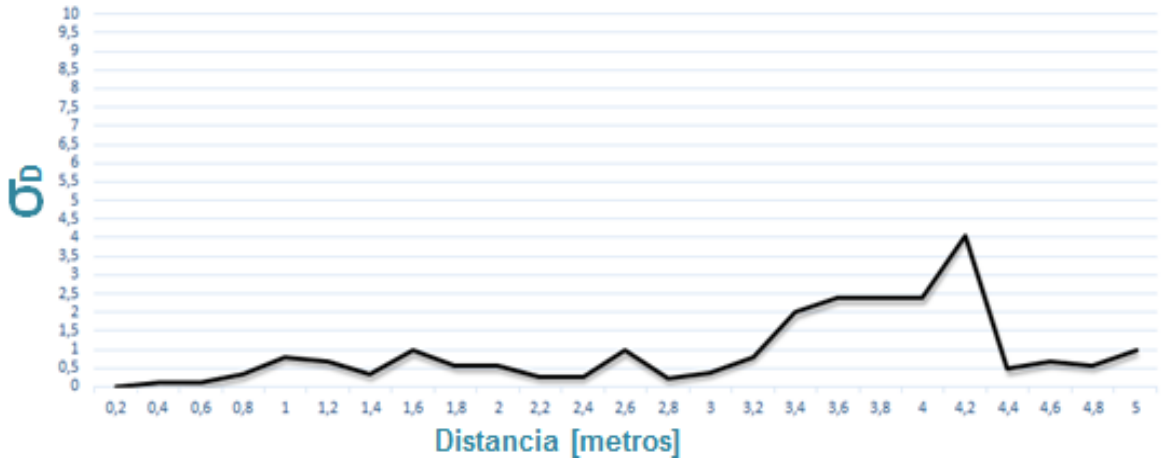
Luego se obtuvo la figura 4.4a: RSSI2 = f(D2) = -5.579.ln(D2) – 41.035. Las figuras 4.4a: σ_{RSSI2} = f(D2) y 4.4b: σ_{D2} = f(D2) muestran la incertidumbre de la RSSI2 y D2 respectivamente.



(a)



(b)



(c)

Figura 4.4 Resultados de prueba RSSI versus Distancia con fijo S1

En las gráficas 4.3b y 4.4b se puede observar que al utilizar un radio xbee S1 como receptor de datos, la RSSI presenta un margen de error mayor que al utilizar un radio xbee S1-PRO.

El xbee S1-PRO transmite datos hacia los receptores y como el lugar de localización solo mide 10 x 5[metros], los datos que transmita llegan correctamente y con la misma RSSI a los receptores S1-PRO y S1 que presentan alcances de recepción de 100[metros] y 30[metros] respectivamente.

Como la prueba se realizó en un entorno cerrado rodeado de muchos objetos metálicos, como es el laboratorio del CVR, esto provoca una gran disminución de potencia de las señales de datos por efecto de fenómenos de difracción, reflexión y multitrayectoria, lo que disminuye los alcances de recepción de los radios xbee considerablemente y provoca que algunos de los datos transmitidos hacia estos no sean recibidos y se presenten valores de RSSI incorrectos y menores en el caso del radio xbee S1 por presentar un alcance de recepción menor que el que presenta un radio xbee S1-PRO, lo cual se puede ver reflejado en la 4.3c y 4.4c en las que el error en la distancia representada por σ_{D1} y σ_{D2} se ve afectado notablemente por las RSSI recibidas.

4.3 PRUEBA DE LOCALIZACIÓN DE OBJETO

Se implementa el sistema de localización completo con el modelo de transmisor sobre el objeto y funciona automáticamente de esta forma:

- El xbee sobre la silla transmite “NUM_DATOS” datos Xi hacia los 3 xbee receptores fijos que extraen la RSSI promedio de los datos Xi recibidos y la envía por sockets TCP a la computadora central en la misma red.

- La Central ejecuta 3 hilos al mismo tiempo que aplican las relaciones: D1 = f(RSSI) y D2 = f(RSSI) para calcular las distancias de cada xbee fijo al objeto y con estas aplica trilateración para estimar la posición (x,y) del objeto. [Ver figura 4.1]

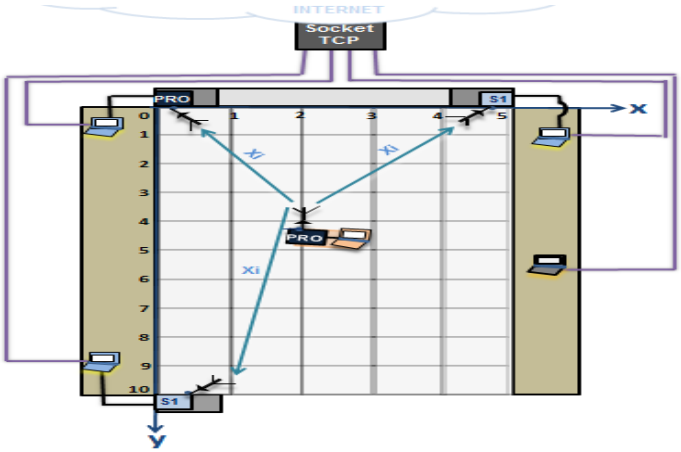


Figura 4.5 Esquema de prueba de localización del objeto

Esta prueba tiene como objetivo variar NUM_DATOS a valores de 1 y 10, y con estos valores ubicar el objeto en una posición específica para:

- Determinar que tan precisa es la medida que brinda el sistema.
- Obtener los tiempos de respuesta del sistema para estimar la velocidad máxima a la que un objeto puede moverse.

El programa receptor en cada xbee fijo permite conocer los tiempos que tardan en realizarse las operaciones del proceso para obtener las RSSI de los datos transmitidos y el programa central permite conocer el tiempo que se tardan en realizar las operaciones del proceso para estimar la posición del objeto.

Primero se ubica la silla en la posición (2,4) y se define en los programas transmisor y receptor la constante NUM_DATOS como 1, para que el xbee sobre el objeto transmita el dato X0 hacia los xbee receptores fijos.

Tabla 4.1 Resultados de prueba de localización con 1 dato transmitido

Tiempos de Respuesta Promedio del Programa Transmisor	
Ingreso de dato	Envío del dato por broadcast
2 [mseg]	1 [mseg]

Tiempos de Respuesta Promedio del Programa Receptor			
Lectura del dato	Ingresar a modo comando	Obtener la RSSI	Salir del modo comando
2 [mseg]	15 [mseg]	15 [mseg]	15 [mseg]

Tiempo de Respuesta Promedio del Programa Central		
Comunicación por Socket TCP	Cálculo de distancia por hilo	Trilateración
10 [mseg]	1 [mseg]	2 [mseg]

El tiempo de respuesta del sistema con la transmisión de 1 dato es de: 70[mseg], y como el lugar tiene una distancia de 10 [metros], la velocidad máxima con la que la silla puede moverse es 10/0.070[m/seg].

El programa central del sistema determinó que la silla está ubicada en la posición (5,5) lo que implica una precisión con un error de 3[metros] en la coordenada en x, y de 1 [metro] en la coordenada en y.

Luego se ubica la silla en la misma posición (2,4) y se define en los programas transmisor y receptor la constante NUM_DATOS como 10, para que el xbee sobre el objeto transmita los datos: X0, X1, X2, X3, X4, X5, X6, X7, X8, X9 hacia los 3 xbee receptores fijos.

Tabla 4.2 Resultados de prueba de localización con 10 datos transmitidos

Tiempos de Respuesta Promedio del Programa Transmisor	
Ingreso de datos	**Envío de datos por broadcast**
20 [mseg]	10 [mseg]

Tiempos de Respuesta Promedio del Programa Receptor			
Lectura de datos	**Ingreso a modo comando**	**Obtener las RSSI**	**Salir del modo comando**
20 [mseg]	150 [mseg]	150 [mseg]	150 [mseg]

Tiempo de Respuesta Promedio del Programa Central		
Comunicación por Socket TCP	**Cálculo de distancia por hilo**	**Trilateración**
10 [mseg]	1 [mseg]	2 [mseg]

Se concluyó que el tiempo de respuesta del sistema con la transmisión de 10 datos es: 513 [mseg], y la velocidad máxima con la que la silla puede moverse es 10/0.513[m/seg].

El programa central del sistema determinó que la silla está ubicada en la posición (3,4) lo que implica una precisión con un error de 1[metro] en la coordenada en x, y de 0 [metros] en la coordenada en y.

Esta pruebas de localización con 1 y 10 datos permiten concluir que a mayor cantidad de datos transmitidos, mayor precisión, pero menor velocidad a la que se puede mover un objeto. Además como el laboratorio presenta un área pequeña, las velocidades son muy altas.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- Es posible utilizar a los radios xbee para implementar un sistema de localización de objetos, pues presentan tiempos de respuesta cortos de [mseg] como se observó en los tiempos promedios de respuesta calculados en la prueba 4.3 de localización del objeto, y se concluyó que la precisión del sistema es directamente proporcional a la cantidad de datos transmitidos entre los xbee, pues una gran cantidad de datos disminuye la probabilidad de que la pérdida de un dato afecte a la RSSI promedio y a la ubicación, y es inversamente proporcional a la velocidad máxima a la que puede moverse el objeto a localizar, pues esta gran cantidad de datos requiere un gran intervalo de tiempo para procesarlos y la velocidad a la que los objetos pueden moverse disminuye.

- Con la optimización de la planificación de los procesos round robin por parte del kernel por medio de programación en soft real time y sincronización de hilos se puede obtener una precisión regular, sin necesidad de alterar del tiempo de respuesta del sistema. Esta precisión regular es reflejada en la precisión obtenida en la prueba 4.3 de localización de un objeto con la transmisión de 1 y 10 datos, y se debe a la presencia de varios objetos metálicos que bloquean la línea de vista de las señales emitidas.

RECOMENDACIONES

- Nuestro sistema es híbrido respecto a los tipos de radios xbee, ya sea S1 ó S1 PRO que se utilizan, lo cual genera dos curvas Distancia = f (RSSI) diferentes como se vio en la prueba de RSSI versus distancia de la sección 4.2, por lo que para facilidad de realización de estas pruebas se recomienda usar un solo tipo de xbee y específicamente los xbee S1-PRO por el gran alcance que brindan.

- Se puede mejorar la precisión en la posición (x,y) del sistema, por medio de un análisis más extenso de las incertidumbres $\sigma_{RSSI} = f(D)$, $\sigma_D = f(D)$, obtenidas en la prueba de RSSI versus Distancia de la sección 4.2, y con la combinación del sistema con un dispositivos de GPS en las posiciones de referencia conocidas de los xbee receptores.

- Es recomendable realizar estas pruebas en un lugar en el que no se presenten varios objetos metálicos pues como se analizó en la sección 2.1 sobre exactitud en medidas de potencia, el metal provoca que se presenten fenómenos de difracción y reflexión que afectan considerablemente los valores de potencia obtenidos en el sistema, lo que implica que afecten a las RSSI y a las distancias obtenidas en base a estas, y como se pudo reflejar al observar las incertidumbres $\sigma_{RSSI1} = f(D1)$, $\sigma_{D1} = f(D1)$ y $\sigma_{RSSI2} = f(D2)$, $\sigma_{D2} = f(D2)$ obtenidas en la prueba de RSSI versus Distancia de la sección 4.2.

BIBLIOGRAFÍA

[1] Blauden Electronics - España, (2008, Febrero), “¿Qué es el GPS?”, Soluciones de conectividad y movilidad [Online], Disponible en: http://www.zonagps.com/que-es-el-gps/

[2] Blauden Electronics - España, (2008, Febrero), “Sistemas de Localización GPS”, Soluciones de conectividad y movilidad [Online], Disponible en: http://www.zonagps.com/aplicaciones/localizacion-gps/ sistemas-de-localizacion-gps/

[3] Solred Radiodifusión – Argentina, (2008, Octubre), “Propagación de Ondas de Radio”, Propagación por Línea Visual [Online], Disponible en: http://www.solred.com.ar/lu6etj/tecnicos/handbook/ propagacion/propagacion.htm

[4] Universidad de Córdoba – Argentina, (2013, Mayo), “Antenas de Comunicación”, Tipos de Antenas [Online], Disponible en: http://www.cordobawireless.net/portal/descargas/Conceptos_basicos_sobre_antenas.pdf

[5] Wikipedia.org – Estados Unidos, (2013, Junio), “Propagación de Errores”, Error en combinaciones No Lineales [Online], Disponible en: http://es.wikipedia.org/wiki/Propagación_de_errores

[6] Escuela Superior de Ingenieros de Bilbao: Departamento de Sistemas – España, (2012, Mayo), “Sistemas Operativos”, Historia de los Sistemas Operativos [Online], Disponible en: http://www.disa.bi.ehu.es/spanish/asignaturas/ii/3_SO.pdf

[7] Universidad de Zaragoza: J. L. Villaroel – España, (2012, Mayo), “Sistemas de tiempo real”, Definicion y Clases de sistemas de tiempo real [Online], Disponible en: http://webdiis.unizar.es/joseluis/STR.pdf

[8] HispaLinux – España, (2007, Junio), “Sockets”, Programación de sockets [Online], Disponible en: http://es.tldp.org/Tutoriales/prog-sockets

[9] Wikipedia.org – Estados Unidos, (2013, Marzo), “Trilateración”, Localización de Objetos [Online], Disponible en: http://es.wikipedia.org/wiki/Trilateración

[10] Spark Fun Electronics Inc. – Estados Unidos, (2012, Marzo), “Modos de operación de un xbee”, Funcionamiento de xbee [Online], Disponible en: https://www.sparkfun.com/products/8664

[11] Universidad de Colima – México, (2001, Mayo), “Modelo OSI”, Las 7 capas del modelo OSI y sus funciones principales [Online], Disponible en: http://docente.ucol.mx/al950441/public_html/osi1hec_B.htm

[12] Libelium: David Gascón – España, (2010, Septiembre), “Dispositivos Inalámbricos, ¿Como conviven? ”, Banda de 2.4 Ghz [Online], Disponible en: http://blogs.heraldo.es/ciencia/?p=1417

[13] Blogspot Ingeniería Aplicada al Área de la Mecatrónica: Alvaro unal – Colombia, (2012, Julio), “Modulos xbee”, Configuración de módulos XBEE [Online], Disponible en: http://alvarounal.blogspot.com/2011/12/modulos-xbee-parte-6-configuraciony.html

[14] Universidad de Pensylvania – Estados Unidos, (2006, Junio), “Real Time Scheduling”, Soft Temporal Constraints [Online], Disponible en: http://informatica.uv.es/it3guia/ARS/practicas/Funciones.pdf

[15] Lawrence Livermore National Laboratory – Estados Unidos, (2013, Enero), “POSIX”, POSIX Thread [Online], Disponible en: http://www.cis.upenn.edu/~lee/06cse480/lec-real-time-scheduling.pdf

[16] YoLinux – Estados Unidos, (2013, Junio), “POSIX Threads”, POSIX Thread Libraries [Online], Disponible en: http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

ANEXOS

Anexo 3.1: Código del programa transmisor.-

Código del programa ejecutado en la computadora conectada al xbee ubicado sobre el objeto a localizar, que permite a este xbee almacenar un dato “Xi” cada 50 [mseg] en su buffer de datos y transmitirlo automáticamente en modo broadcast a todos los 3 xbees configurados en este modo. La cantidad máxima de datos que se transmiten se define por la constante NUM_DATOS del programa, que para este código se definió como 10.

```
//Librerías utilizadas para la compilacion del código del transmisor
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <termios.h>
4 #include <strings.h>
5 #include <string.h>
6 #include <stdlib.h>
7 #include <fcntl.h>
8 #include <stdio.h>

10 #define NUM_DATOS   10
11 #define TAM_BUFFER  255

13 int fd, c, res, I = 0, m = 0;

18 char *serialPort="";
19 char *USBserialPort0="/dev/ttyUSB0";
20 char *USBserialPort1="/dev/ttyUSB1";
21 char *USBserialPort2="/dev/ttyUSB2";
22 char *USBserialPort3="/dev/ttyUSB3";
23 char *USBserialPort4="/dev/ttyUSB4";

25 char *u0="USB0";
26 char *u1="USB1";
27 char *u2="USB2";
28 char *u3="USB3";
29 char *u4="USB4";

31 char *cadena;
33 char buf[TAM_BUFFER] = {0};
35 double inicio, final;
37 struct timeval time; //Estructura para especificar un intervalo de tiempo
39 struct termios oldtio, newtio; /* estructura termios donde se almacenan:
                                  tcflag_t c_iflag, parametros de modo entrada
                                  tcflag_t c_oflag, parametros de modo salida
                                  tcflag_t c_cflag, parametros de modo control
                                  tcflag_t c_lflag, parametros de modo local
                                  cc_t c_line,      disciplina de la linea
                            */
49 main(int argc, char **argv)
50 {
```

52 //1).- CONFIGURACIÓN DE PUERTO PARA LECTURA DE BUFFER DE XBEE

```
54  //Abrir el puerto del computador donde se encuentra conectado el xbee
55  if (argc != 2)
56      {
57          fprintf(stderr, "ERROR AL LEER EL ARCHIVO\n");
58              exit (-1);
59      }
60      if(!strcmp(argv[1],u0)) serialPort=USBserialPort0;
61      if(!strcmp(argv[1],u1)) serialPort=USBserialPort1;
62      if(!strcmp(argv[1],u2)) serialPort=USBserialPort2;
63      if(!strcmp(argv[1],u3)) serialPort=USBserialPort3;
64      if(!strcmp(argv[1],u4)) serialPort=USBserialPort4;
65      if(!strcmp(serialPort,""))
66  {
67    fprintf(stderr, "ERROR, VERIFIQUE SI EL PUERTO ES: USB0,USB1 \n");
68        exit(-1);
69  }
70  if ((fd = open(serialPort, O_RDWR | O_NOCTTY)) < 0)
71      {
72   fprintf(stderr,"ERROR,NO Detecto xbee en puerto:%s \n",serialPort);
73   exit(-1);
74  }
75  else
76      {macrosigma
77      if (!fd)
78          {
79              // Abre el dispositivo para lectura y escritura
80              fd = open(serialPort, O_RDWR | O_NOCTTY);
81              tcflush(fd, TCIOFLUSH);
82      }
83          // La lectura se realiza inmediatamente
84          printf("EXITO, XBEE DETECTADO EN EL PUERTO: %s \n", serialPort);
85      }

86      tcgetattr(fd, &oldtio);//Almacena la configuracion actual del puerto

88      //Limpieza de termios para receptar nuevos parámetros del puerto
89      bzero(&newtio, sizeof(newtio));
90      newtio.c_cflag = CLOCAL | CREAD;
91      cfsetispeed(&newtio, 9600);
92      cfsetospeed(&newtio, 9600);

93      // 8n1 (8bit, no paridad,1 bit de parada)
94      newtio.c_cflag &= ~PARENB;
95      newtio.c_cflag &= ~CSTOPB;
96      newtio.c_cflag &= ~CSIZE;
97      newtio.c_cflag |= CS8;//IGNPAR: Ignora bytes con error de paridad

99      newtio.c_iflag  = 0;
100     newtio.c_oflag  = 0;
101     newtio.c_lflag  = 0;//Pone el modo entrada (no-canonico, sin eco,...)

103     newtio.c_cc[VINTR] = 0;//Ctrl-c
104     newtio.c_cc[VTIME] = 0;//Temporizador entre caracter, no usado
105     newtio.c_cc[VMIN]  = 1;//Bloquea lectura hasta llegada de 1er caracter

107     tcsetattr(fd,TCSANOW,&newtio);
```

112 //2).- CONFIGURACIÓN DE XBEE PARA ENVIO DE DATO ALMACENADO EN SU BUFFER

```
114      while(m!=2)
115     {
116          gettimeofday(&time, NULL);//Hora del sistema en seg y mseg
117          inicio = time.tv_sec+(time.tv_usec/1000000.0;//Empezo el programa
118          cadena = (char*) malloc ( sizeof(char) * 3);
119          system("clear");
120          printf("\n\n\t  TRANSMISOR\n");
121          printf("\n\t\t\t\t     DATO\n");

123          while(i < NUM_DATOS)
124          {
125                       sprintf(cadena, "X%d", i);
126                            write(fd, cadena, 2);
127                            fprintf(stderr, "\n\t\t\t\t      %s", cadena);
128                            usleep(50000);//El dispositivo se duerme durante 50 mseg
129                            i++;
130          }
131          printf("\n\n");
132          i=0;

134          gettimeofday(&time, NULL);//Hora del sistema en seg y mseg
135          final = time.tv_sec+(time.tv_usec/1000000.0);//Termino el programa

137          //Tiempo que se tarda el programa en hacer que el xbee envie Xi
138          fprintf(stderr,"\t\t\t\t\tTIEMPO: %.6lf [Seg]\n", final-inicio);

140          usleep(10000);//Se duerme 1 segundo el dispositivo.
141          m++;
142     }


145     tcsetattr(fd,TCSANOW,&oldtio);//Restaura configuracion del puerto
146     close(fd);




151 }
```

Anexo 3.2: Código del programa receptor.-

Código del programa ejecutado en los computadores conectados a los xbee ubicados en los extremos mas altos del lugar de localización para que puedan leer del buffer de un xbee un dato “Xi” transmitido por otro xbee, extraer la RSSI del dato y enviarla por socket TCP al computador central. Esto se realiza durante los 50 [mseg] que el programa transmisor se ejecuta.

```
//Librerías utilizadas para la compilacion del código del receptor
1   #include <sys/types.h>
2   #include <sys/select.h>
3   #include <termios.h>
4   #include <strings.h>
5   #include <string.h>
6   #include <stdlib.h>
7   #include <fcntl.h>
8   #include <stdio.h>
9   #include <netinet/in.h>
10 #include <errno.h>
11 #include <metodos.h>

13 #define NUM_DATOS   10
14 #define TAM_BUFFER  255

13   int         fd, c, res, i = 0, m = 0, rv1, rv2,  Error = 0, numError = 0;
14  fd_set set1, set2;
15   double    RSSIPromedio = 0, tInicial1, tInicial2, tFinal1, tFinal2, tEjecucion;


18 char *serialPort="";
19 char *USBserialPort0="/dev/ttyUSB0";
20 char *USBserialPort1="/dev/ttyUSB1";
21 char *USBserialPort2="/dev/ttyUSB2";
22 char *USBserialPort3="/dev/ttyUSB3";
23 char *USBserialPort4="/dev/ttyUSB4";

25 char *u0="USB0";
26 char *u1="USB1";
27 char *u2="USB2";
28 char *u3="USB3";
29 char *u4="USB4";

31 double  tDatosRecibidos [NUM_DATOS];

33 char datosRecibidos [NUM_DATOS] [TAM_BUFFER] ={0},  RSSIHexadecimal [NUM_DATOS] [TAM_BUFFER] ={0};
34 char XBEEBuffer [TAM_BUFFER] ={0};


37 struct timeval timeOut1, timeOut2; //Estructura para especificar intervalos de tiempo

39 struct termios oldtio, newtio; /* estructura termios donde se almacenan:
                                    tcflag_t c_iflag, parametros de modo entrada
                                    tcflag_t c_oflag, parametros de modo salida
                                    tcflag_t c_cflag, parametros de modo control
                                    tcflag_t c_lflag, parametros de modo local
                                    cc_t c_line,      disciplina de la linea
                                */


49 main(int argc, char **argv)
50 {
```

52 //1).- CONFIGURACIÓN DE PUERTO PARA LECTURA DE BUFFER DE XBEE

```
54  //Abrir el puerto del computador donde se encuentra conectado el xbee
55  if (argc != 2)
56       {
57            fprintf(stderr, "ERROR AL ESTABLECER COMUNICACIÓN CON EL XBEE \n");
58                 exit (-1);
59       }
60       if(!strcmp(argv[1],u0)) serialPort=USBserialPort0;
61       if(!strcmp(argv[1],u1)) serialPort=USBserialPort1;
62       if(!strcmp(argv[1],u2)) serialPort=USBserialPort2;
63       if(!strcmp(argv[1],u3)) serialPort=USBserialPort3;
64       if(!strcmp(argv[1],u4)) serialPort=USBserialPort4;
65       if(!strcmp(serialPort,""))
66  {
67    fprintf(stderr, "ERROR, VERIFIQUE SI EL PUERTO ES: USB0,USB1 \n");
68         exit(-1);
69  }
70  if ((fd = open(serialPort, O_RDWR | O_NOCTTY | O_NDELAY)) ==  -1)
71       {
72   fprintf(stderr,"ERROR,NO Detecto xbee en puerto:%s \n",serialPort);
73   exit(-1);
74  }
75  else
76       {
77       if (!fd)
78            {
79                 // Abre el dispositivo para lectura y escritura
80                 fd = open(serialPort, O_RDWR | O_NOCTTY | O_NDELAY);
81            }
82            fcntl (fd, F_SETFL, 0);
83            // La lectura se realiza inmediatamente
84            printf("EXITO, XBEE DETECTADO EN EL PUERTO: %s \n", serialPort);
85       }

86       tcgetattr(fd, &oldtio);//Almacena la configuracion actual del puerto

88       //Limpieza de termios para receptar nuevos parámetros del puerto
89       bzero(&newtio, sizeof(newtio));
90       newtio.c_cflag = CRTSCTS | CS8 | CLOCAL | CREAD;
91       cfsetispeed(&newtio, 9600);
92       cfsetospeed(&newtio, 9600);

93       // 8n1 (8bit, no paridad,1 bit de parada)
94       newtio.c_cflag &= ~PARENB;
95       newtio.c_cflag &= ~CSTOPB;
96       newtio.c_cflag &= ~CSIZE;
97       newtio.c_cflag |= CS8;//IGNPAR: Ignora bytes con error de paridad

99       newtio.c_iflag  = 0;
100      newtio.c_oflag  = 0;
101      newtio.c_lflag  = 0;//Pone el modo entrada (no-canonico, sin eco,...)

103      newtio.c_cc[VINTR] = 0;//Ctrl-c
104      newtio.c_cc[VTIME] = 0;//Temporizador entre caracter, no usado
105      newtio.c_cc[VMIN]  = 1;//Bloquea lectura hasta llegada de 1er caracter
106      tcflush(fd,TCIFLUSH);
107      tcsetattr(fd,TCSANOW,&newtio);
```

112 //2).- CONFIGURACIÓN DE XBEE PARA RECEPCIÓN DE DATO

```
114 while(1) {
116   tInicial1=calcularSegundos();
117   timeOut1.tv_sec=0;
118   timeOut1.tv_usec=4001000;
119   FD_ZERO(&set1);
120   FD_SET(fd,&set1);
121   rv1=select(fd+1,&set1,NULL,NULL,&timeOut1);

128   if(rv1==0){
130      numError = NUM_DATOS;
131      tFinal1 = calcularSegundos();
132      tEjecucion = tFinal1-tInicial1;
133      interfazReceptor(RSSIHexadecimal,datosRecibidos,tDatosRecibidos,&RSSIPromedio);
135      fprintf(stderr,"\n\t\t\t\t\t%d Datos Perdidos\n",numError);
136   }
129   if(rv1>=1){
131       numError=0;
133       while(i< NUM_DATOS){
135             timeOut2.tv_sec=0;
136             timeOut2.tv_usec=55000;
137             FD_ZERO(&set2);
138             FD_SET(fd,&set2);
139             rv2=select(fd+1,&set2,NULL,NULL,&timeOut2);
145             if(rv2==0){
146                  strcpy(RSSIHexadecimal[i],"0");
147                  numError++;
148                  i++;
149             }
150             if(rv2>=1){
151                  tInicial2=calcularSegundos();

153                  /** EN MODO TRANSPARENTE **/
154                  res = read(fd,XBEEBuffer,tamBuffer);
155                  strcpy(datosRecibidos[i],XBEEBuffer);
156                  memset(XBEEBuffer,'\0',tamBuffer);

158                  /** EN MODO COMANDO **/
159                  write(fd,"+++",strlen("+++"));
160                  usleep(14000);
161                  res=read(fd,XBEEBuffer,tamBuffer);
162                  memset(XBEEBuffer,'\0',tamBuffer);
163                  write(fd,"atdb\r",strlen("atdb\r"));
164                  usleep(14000);
165                  res=read(fd,XBEEBuffer,tamBuffer);
166                  strcpy(RSSIHexadecimal[i],XBEEBuffer);
167                  memset(XBEEBuffer,'\0',tamBuffer);
168                  write(fd,"atcn\r",strlen("atcn\r"));
169                  usleep(14000);
170                  res=read(fd,XBEEBuffer,tamBuffer);
171                  memset(XBEEBuffer,'\0',tamBuffer);
172                  tFinal2=calcularSegundos();
173                  tDatosRecibidos[i]=tFinal2-tInicial2;
174                  i++;
175              }
176              FD_CLR(fd,&set2);
177         }
178     interfazReceptor(RSSIHexadecimal,datosRecibidos,tDatosRecibidos,&RSSIPromedio);
179     FD_CLR(fd,&set1);
180     tFinal1=calcularSegundos();
181     Error=numDatos-numError;
182     tEjecucion=tFinal1-tInicial1;
183
185     for(i=0;i<numDatos;i++){
186           memset(datosRecibidos[i],'\0',tamBuffer);
187           tDatosRecibidos[i]=0;
188     }
189     i=0,Error=0,tFinal1=0,tFinal2=0,numError=0,tInicial1=0,tInicial2=0,tEjecucion=0;
190     }
191   }
192   tcsetattr(fd,TCSANOW,&oldtio);//Restaura configuracion del puerto
193   close(fd);
194 }
```

Anexo 3.3: Código del programa central.-

Este código tiene como función hacer el computador central espere por recibir las 3 RSSI provenientes de los xbee fijos para luego generar 3 hilos que calculen las distancias de cada xbee fijo al objeto y aplica trilateración para estimar la posición (x,y) del objeto.

```
//Librerías utilizadas para la compilacion del código del receptor
1    #include <sys/socket.h>
2    #include <netinet/in.h>
3    #include <sys/types.h>
4    #include <funciones.h>
5    #include <sys/time.h>
6    #include <pthread.h>
7    #include <unistd.h>
8    #include <signal.h>
9    #include <sys/un.h>
10   #include <string.h>
11   #include <stdlib.h>
12   #include <netdb.h>
13   #include <stdio.h>
14   #include <errno.h>
15   #include <math.h>
16   #include <time.h>

18   #define  _REENTRANT  /* Para que los hilos no se cuelgen al accede a las funciones de la biblioteca.*/
19   #define   CLIENTES  3   /* Constante que define el número máximo de clientes a atender por el socket.*/

13   int                  clientesConectados;
14   pthread_mutex_t      semaforo = PTHREAD_MUTEX_INITIALIZER; /* Semáforo para serializar
                                                                  accedo a las variables*/
16   double               distancia1, distancia2, distancia3;
```

```
/** Función del HILO: Va a "obtener" la distancia en base al valor del RSSI.**/

void *obtenerDistancia(void *arg){
   int dato1 = 0;       /* BUFFER usado para leer datos del socket */
   int tid;             /* Variable usada para guardar el id del hilo */
   double distancia;    /* Variable temporal */
   long sd2=(long)arg;  /* Descriptor de archivo para el cliente */
   char dato2[20]={0};  /* BUFFER usado para leer datos del socket */
   fd_set set;          /* Descriptor de lectura para funcion select */

   while(1){
     FD_ZERO(&set); /* Inicialización de descriptor de lectura */
     FD_SET(sd2,&set); /* Se añade en la funcion select el socket del cliente */
     select(sd2+1,&set,NULL,NULL,NULL);/*Hilo dormido hasta que sd2 tenga algo que leer.
                                        (hasta que el cliente envíe algún mensaje)*/
     pthread_mutex_lock(&semaforo);/*Variable no disponible, el proceso se bloquea*/

     if(FD_ISSET(sd2, &set)){  /*Cliente ha enviado algún mensaje nuevo*/
        pthread_mutex_unlock(&semaforo);/*No bloquear mutex en obtención de datos*/
        tid=(int)pthread self(); /* ID del hilo */
        printf("TID del Hilo: %d\n", tid);

         if((leerDatosSocket(sd2, dato2, sizeof(dato2))>0)){
            /*Se comprueba que cliente envia mensajes para luego obtener la distancia en
            base al RSSI y asignarla a la variable que sera pasada al hilo principal */
            leerDatosSocket(sd2, (char *)&dato1, sizeof(dato1));

            if(!strcmp(dato2,"XBEE-1")){
               pthread mutex lock(&semaforo);/*Se bloquea el mutex*/
               distancia=calcularDistancia((double)dato1);/*Almacenar valor de función*/
               distancia1=distancia;/*Copiar valor obtenido de distancia en distancia1*/
               pthread mutex unlock(&semaforo);/*Desbloquear mutex pues no se necesita*/
            }

            if(!strcmp(dato2,"XBEE-2")){
               pthread mutex lock(&semaforo); /*Se bloquea el mutex*/
               distancia=calcularDistancia((double)dato1);/*Almacenar valor de función*/
               distancia2=distancia;/*Copiar valor obtenido de distancia en distancia2*/
               pthread_mutex_unlock(&semaforo);/*Desbloquear mutex pues no se necesita*/
            }

            if(!strcmp(dato2,"XBEE-3")){
                pthread mutex lock(&semaforo); /*Se bloquea el mutex*/
                distancia=calcularDistancia((double)dato1);/*Almacena valor de función*/
               distancia3=distancia;/*Copiar valor obtenido de distancia en distancia3*/
               pthread_mutex_unlock(&semaforo);/*Desbloquear mutex pues no se necesita*/
            }
         }
         else{
                printf("Cliente ha cerrado la Conexion\n");
               pthread_mutex_lock(&semaforo);/*Se bloquea el mutex*/
              clientesConectados--;/*Se disminuye la cantidad de clientes conectados*/
               pthread_mutex_unlock(&semaforo);/*Desbloquear mutex pues no se necesita*/
              FD_CLR(sd2, &set); /*Se limpia el descriptor de lectura*/
               close(sd2); /*Se cierra el descriptor del cliente*/
              pthread_exit(&sd2); /*Se destruye el hilo*/
         }
     }
   }
}
```

```
/* HILO o FUNCION PRINCIPAL */
int main(){
     struct timeval timeOut;            /* Tiempo usado en la funcion select*/
     fd set set;                        /* Descriptor de la funcion select */
     int rv;                            /* Usada en la funcion select */
     int sd1;                           /* Descriptor del socket servidor */
     int sd2;                           /* Descriptor para socket clientes */
     int numClientesConectados=0;       /* Numero de clientes conectados */
     double d1=0,d2=0,d3=0, *posicion4; /* Variable temporal */
     double  posicion1[2]={0,0};        /* Posición de las circunferencias y la */
     double  posicion2[2]={5,0};        /* posición en la que se encontrará el objeto */
     double  posicion3[2]={0,5};

     pthread t  hilo;       /*Estructura para crear un hilo, Atributos y parametros*/
     pthread attr t       atributos;
     struct sched_param   parametro;
     pthread_attr_init(&atributos); /*Se inicializan los atributos del hilo*/
     pthread_attr_setschedpolicy(&atributos, SCHED_RR); /*Planificación ROUND ROBIN*/
     pthread_attr_setschedparam(&atributos, &parametro);/*Prioridad 10 para cada hilo*/
     parametro.sched priority =10;

     if((sd1=abrirConexion("RSSI"))==-1){ /*Abre socket para que un cliente pueda enviar
                                          y recibir mensajes a través de él.
                                          El servicio RSSI se debe encontrar levantado
                                          en el archivo /etc/services */
        perror("No se puede abrir el socket en el Servidor");
     }
     else {
       printf("PROGRAMA CENTRAL INICIADO\n\n");
       while(1) { /*Bucle Infinito:Espera a ver si existen más CLIENTES para conectar*/
           timeOut.tv sec=0;
           timeOut.tv usec=55000;
           FD ZERO(&set);  /*Se inicializa el descriptor de lectura*/
           FD_SET(sd1,&set); /*Se añade en la funcion select el socket cliente*/
           rv = select(sd1+1, &set, NULL, NULL, &timeOut);/* Proceso dormido hasta que
                                                    sd1 tenga algo que decir.(nuevo
                                                    cliente quiera conectarse)*/
           if(rv==0){ /*Comprobación de 3 clientes permitidos, se estima la posición*/
                if(clientesConectados==CLIENTES){
                    pthread_mutex_lock(&semaforo); /*Se bloquea el mutex*/
                    d1=distancia1;
                    d2=distancia2;
                    d3=distancia3;
                    pthread_mutex_unlock(&semaforo); /*Desbloquear mutex*/
                   /* Pasar distancias obtenidas, para estimar la posición del objeto*/
                   posicion4=estimarPosicion(posicion1,d1,posicion2,d2,posicion3,d3);
                   printf("Coordenada-X: %.2f\n", posicion4[0]);/*Posición estimada*/
                   printf("Coordenada-Y: %.2f\n", posicion4[1]);
                }
           }
           if(rv>=1){ /*Comprobar si cliente nuevo desea conectarse y se le admite, si
                      se supera numero de clientes permitidos se cierra la conexión*/
                sd2=aceptarConexion(sd1); /*Se aceptan las conexiones*/
                pthread_mutex_lock(&semaforo); /*Se bloquea el mutex*/
                clientesConectados++;  /*Se aumenta la cantidad de clientes conectados*/
                pthread_mutex_unlock(&semaforo);/*Desbloquea mutex pues no se necesita*/

                if(clientesConectados>CLIENTES){ /*Se desconecta cliente no aceptado*/
                   printf("Cliente Desconectado\n");/*Descriptor del cliente se cierra*/
                   close(sd2);
                   pthread_mutex_lock(&semaforo); /*Se bloquea el mutex*/
                   clientesConectados--;/*Disminuye la cantidad de clientes conectados*/
                   pthread_mutex_unlock(&semaforo);
                }
                else{ /*Si un cliente es aceptado se le asigna un HILO*/
                    pthread create(&hilo,&atributos,obtenerDistancia,(void*)(long)sd2);
                    pthread_detach(hilo);/*Liberar recursos para el Sistema Operativo*/
                }
            }
        }
        FD CLR(sd1, &set);  /*Se limpia el descriptor de lectura*/
        close(sd1); /*Se cierra el descriptor del socket servidor*/
        pthread_mutex_destroy(&semaforo); /*Se destruye el semáforo*/
        return 0;
}
```