SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE MÉRIDA

ITM

# "SISTEMA AUTÓNOMO DE NAVEGACIÓN PARA UGVS USANDO TÉCNICAS DE VISIÓN Y ESTIMACIÓN DE POSICIÓN BASADO EN RSSI"

OPCIÓN:

**TESIS**

PARA OPTAR AL GRADO DE:

**MAESTRO EN INGENIERÍA**


PRESENTA:

**ANDRÉS JOSUÉ CASTRO ANGULO**

DIRECTOR:

**JOSÉ RAMÓN ATOCHE ENSEÑAT**

CODIRECTOR:

**ALEJANDRO ARTURO CASTILLO ATOCHE**



MÉRIDA, YUCATÁN, MÉXICO

06 DE DICIEMBRE DE 2019

Mérida, Yucatán, 6/noviembre/2019

**ASUNTO:** Solicitud de autorización de impresión.

**Daniel Arcángel López Sauri**
**Jefe de la División de Estudios**
**de Posgrado e Investigación**
**PRESENTE**

En virtud de que el C. **Andrés Josué Castro Angulo**, ha culminado satisfactoriamente la tesis: "**Sistema Autónomo de Navegación para UGVs usando Técnicas de Visión y Estimación de Posición basado en RSSI**", participando exitosamente en el Seminario de Investigación de la misma y cubierto los requisitos necesarios para optar al grado de "Maestro en Ingeniería" le solicitamos se le otorgue la Autorización correspondiente para realizar la impresión de su trabajo final, el cual avala como un producto de calidad la Comisión Revisora conformada por los que a continuación firman.

| DIRECTOR | CODIRECTOR |
|---|---|
| José Ramón Atoche Enseñat | Alejandro Arturo Castillo Atoche |
| REVISOR | REVISOR |
| Carlos Alberto Luján Ramírez | José Agustín Hernández Benítez |

**ATENTAMENTE**
*Excelencia en Educación Tecnológica*

**Sylvia María del Rosario Ruiz Casanova**
**Coordinadora de la Maestría en Ingeniería**
C.p. Archivo
DALS/SRC/fja

INSTITUTO TECNOLOGICO DE MERIDA
RECIBIDO
11 NOV 2019
DIVISION DE ESTUDIOS DE POSGRADO E INVESTIGACION

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Mérida

"2019, Año del Caudillo del Sur, Emiliano Zapata"

DEPENDENCIA: DIV. DE EST. DE POSG. E INV.
No. DE OFICIO: X-423/19
Mérida, Yucatán, 11/noviembre/2019

**ASUNTO:** AUTORIZACIÓN DE IMPRESIÓN

**C. ANDRÉS JOSUÉ CASTRO ANGULO**
**PASANTE DE LA MAESTRÍA EN INGENIERÍA**
**PRESENTE.**

De acuerdo al fallo emitido por su director **José Ramón Atoche Enseñat**, codirigido por Alejandro Arturo Castillo Atoche y la comisión revisora integrada por Carlos Alberto Luján Ramírez y José Agustín Hernández Benítez, considerando que cubre los requisitos establecidos en el Reglamento de Titulación de los Institutos Tecnológicos le autorizamos la impresión de su trabajo profesional con la TESIS:

**"SISTEMA AUTÓNOMO DE NAVEGACIÓN PARA UGVS USANDO TÉCNICAS DE VISIÓN Y ESTIMACIÓN DE POSICIÓN BASADO EN RSSI"**

**A T E N T A M E N T E**
*Excelencia en Educación Tecnológica®*

S.E.P.
INSTITUTO TECNOLOGICO
DE MERIDA
DIVISION DE ESTUDIOS DE
POSGRADO E INVESTIGACION

**DANIEL ARCÁNGEL LÓPEZ SAURI**
**JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN**

C.p. Archivo
DALS/fja

# Índice general

**Lista de tablas** VII

**Lista de figuras** VIII

**1. Introducción** 1
- 1.1. Planteamiento del Problema . . . 2
- 1.2. Objetivos . . . 3
  - 1.2.1. General . . . 3
  - 1.2.2. Específico . . . 3
- 1.3. Justificación . . . 4
- 1.4. Estado del Arte . . . 4

**2. Marco Teórico** 7
- 2.1. Bluetooth de bajo consumo . . . 7
  - 2.1.1. Perfiles de Acceso Genérico . . . 8
  - 2.1.2. Perfil de Atributo Genérico . . . 9
  - 2.1.3. Protocolo de Atributo . . . 9
  - 2.1.4. Configuración de las antenas y el explorador . . . 9
- 2.2. Estimación de distancia usando RSSI en BLE . . . 11
- 2.3. Trilateración . . . 12
- 2.4. Detección de obstáculos . . . 14
  - 2.4.1. K-medias . . . 15
  - 2.4.2. Detector de esquinas de Harris . . . 17
  - 2.4.3. Disparidad y profundidad . . . 19
  - 2.4.4. Suma de diferencias absolutas . . . 20

**3. Desarrollo** 21
- 3.1. Vehículo explorador . . . 21
- 3.2. BLE . . . 23
  - 3.2.1. Interfaz de adquisición de datos . . . 25
- 3.3. Trilateración . . . 26
  - 3.3.1. Cálculo de cantidad nodos . . . 28
- 3.4. Detección de obstáculos . . . 30

3.4.1. Filtrado y rectificación. . . . . 31
3.4.2. Puntos de interés . . . . . 31
3.4.3. Disparidad . . . . . 31
3.4.4. Clusters de profundidad . . . . . 31

**4. Resultados** **33**
4.1. RSSI . . . . . 33
4.2. Trilateración . . . . . 46
4.3. Detección de obstáculos . . . . . 48

**5. Conclusión** **54**

**Referencias** **56**

**Anexos** **58**

**Python** **59**
.1. Modelo . . . . . 59
.2. Análisis espacial . . . . . 61
.3. myPlots . . . . . 66
.4. myBle . . . . . 67

**PSOC4** **69**
.5. Antena . . . . . 69
.6. Explorador . . . . . 75

# Índice de tablas

2.1. BLE vs Bluetooth . . . . . 8
2.2. Configuración A: Emisor-Observador . . . . . 10
2.3. Configuración B: Periférico-Central . . . . . 11

4.1. Valores de A y n de las antenas . . . . . 39

# Índice de figuras


1.1. Modelo del proyecto. . . . . 3

2.1. Planteamiento geométrico para hallar el punto $(x, y)$ . . . . . 12
2.2. Flujo de procesamiento. . . . . 14
2.3. K-medias. . . . . 16
2.4. Detector de esquinas de Harris. . . . . 18
2.5. Modelo de la disparidad en cámaras estero. . . . . 19
2.6. Representación de la disparidad en un par estéreo. . . . . 20

3.1. Características del vehículo explorador . . . . . 22
3.2. Características y descripción de puertos del Proc BLE SMA . . . . . 23
3.3. Medición punto a punto de RSSI, para dj=1m,2m,3m,4m y 5m. . . . . 24
3.4. Diagrama de bloques de la configuración de hardware para los nodos . . . . . 25
3.5. Distribución de nodos en vértices de rejilla cuadrada. . . . . 26
3.6. Distribución de nodos en vértices de rejilla triangular. . . . . 27
3.7. Distribución de nodos en línea recta. . . . . 28
3.8. Cálculo de la cantidad de nodos necesarios para la distribución propuesta. . . . . 29
3.9. Mapa con coordenadas de puntos de medición probados para la trilateración . . 30

4.1. Datos puros de RSSI para posiciones de 1 a 5 metros . . . . . 34
4.2. Mediana de los datos con ventana de tamaño 100 . . . . . 34
4.3. Promedio de los datos con ventana de tamaño 100 . . . . . 35
4.4. Comparación de los filtros de mediana y media para la antena01 . . . . . 36
4.5. Distribución de frecuencias de valores de RSSI para una medición punto a punto de 1 metro. . . . . 37
4.6. Distribución de frecuencias de valores de RSSI para una medición punto a punto de 2 metros. . . . . 37
4.7. Distribución de frecuencias de valores de RSSI para una medición punto a punto de 3 metros. . . . . 38
4.8. Distribución de frecuencias de valores de RSSI para una medición punto a punto de 4 metros. . . . . 38
4.9. Distribución de frecuencias de valores de RSSI para una medición punto a punto de 5 metros. . . . . 39
4.10. Modelo RSSI-Distancia . . . . . 40

4.11. Medición punto a punto de 1 metro. . . . . 41
4.12. Medición punto a punto de 2 metros. . . . . 41
4.13. Medición punto a punto de 3 metros. . . . . 42
4.14. Medición punto a punto de 4 metros. . . . . 42
4.15. Medición punto a punto de 5 metros. . . . . 43
4.16. Error medio de la antena01 a 1m,2m,3m,4m y 5m. . . . . 44
4.17. Error medio de la antena02 a 1m,2m,3m,4m y 5m. . . . . 44
4.18. Error medio de la antena03 a 1m,2m,3m,4m y 5m. . . . . 45
4.19. Error medio de la antena04 a 1m,2m,3m,4m y 5m. . . . . 45
4.20. Error medio de la antena05 a 1m,2m,3m,4m y 5m. . . . . 46
4.21. Mapa con comparación de los puntos reales con los estimados . . . . . 47
4.22. Mapa de calor del error de la distancia por áreas. . . . . 48
4.23. Proceso actual en que se presentan y evalúan resultados. . . . . 49
4.24. UGV tomando una captura de lo que se encuentra en su camino. . . . . 49
4.25. Proceso actual en que se presentan y evalúan resultados. . . . . 50
4.26. Frame tomado con la cámara ZED. . . . . 50
4.27. Frame corregido para eliminar la distorsión. . . . . 51
4.28. Proceso actual en que se presentan y evalúan resultados. . . . . 51
4.29. Resultado de aplicar el detector de esquinas de Harris en la imagen izquierda. . 52
4.30. Proceso actual en que se presentan y evalúan resultados. . . . . 52
4.31. Objetos cercanos a la cámara. . . . . 53

# Capítulo 1

# Introducción

En este capítulo se hace una introducción al problema y se describen los objetivos del trabajo de tesis y la justificación de la misma. Por último, se presenta el estado del arte utilizado en el proyecto.

Hoy en día, la mayoría de la población mundial se concentra principalmente en las grandes ciudades, y esta concentración cada vez es mayor y será inevitable. Las poblaciones de todo el mundo están emigrando a las ciudades más grandes, debido a que buscan mejores oportunidades económicas. Para 2030, más del 60 % de la población vivirá en megaciudades (con más de 10 millones de habitantes), con una tasa de crecimiento exponencial. La agricultura de precisión asociado al concepto de ciudades inteligentes es una propuesta novedosa para el desarrollo de cultivos inteligentes en áreas rurales y urbanas diseñadas con la perspectiva de crear alta calidad de vida, y alcanzando un desarrollo económico sostenible mediante el avance en varios sectores tecnológicos, tales como el transporte inteligente y la agricultura de precisión. Transformar cualquier ciudad en una ciudad inteligente es una tendencia científica actual para el beneficio de los pueblos. En este sentido, los vehículos terrestres no tripulados (UGVs unmanned ground vehicle) tienen una amplia gama de aplicaciones en muchos campos de cualquier

ciudad inteligente. El uso de sistemas UGV en la agricultura y el transporte es necesario para mejorar la calidad de los servicios en las ciudades mediante rutas de inspección, recopilación de información, monitoreo de emergencia, y reconocimiento, entre otros. Sin embargo, el uso de dispositivos UGV se ve reducido por las dificultades relacionadas con la complejidad de su entorno, y las consecuencias potencialmente graves relacionadas con posibles accidentes. Por lo tanto, para ser fiable, un UGV necesita ser consciente de su entorno, tener la capacidad de entender la escena observada y de manejar su funcionamiento con respecto a las limitaciones provenientes del análisis de la escena. Mediante un sistema de visión, debe ser capaz de comprender y proporcionar una estrategia de conducción automatizada (definición de trayectoria), donde el GPS no está disponible o simplemente porque carece de precisión. En este contexto, la precisión de la detección de movimiento es un requisito básico del sistema, ya que es crucial para vincular la comprensión de la escena a la evolución espacial del UGV.

## 1.1. Planteamiento del Problema

Como parte de su expansión comercial y en la búsqueda de soluciones inteligentes, apoyadas en la tecnología, a los problemas actuales del país Spring Labs plantea el desarrollo de un UGV para monitoreo de cultivo mexicano y control de plagas. Este vehículo contará con un sistema de navegación autónoma, gestión inteligente de la batería, recolección de información del cultivo y conocimiento de su entorno.

En este trabajo de tesis se ataca el problema específico de la navegación autónoma del vehículo a través de una región delimitada por antenas BLE y la detención de obstáculo en su camino para tomar la decisión de seguir o abortar el seguimiento de la trayectoria.

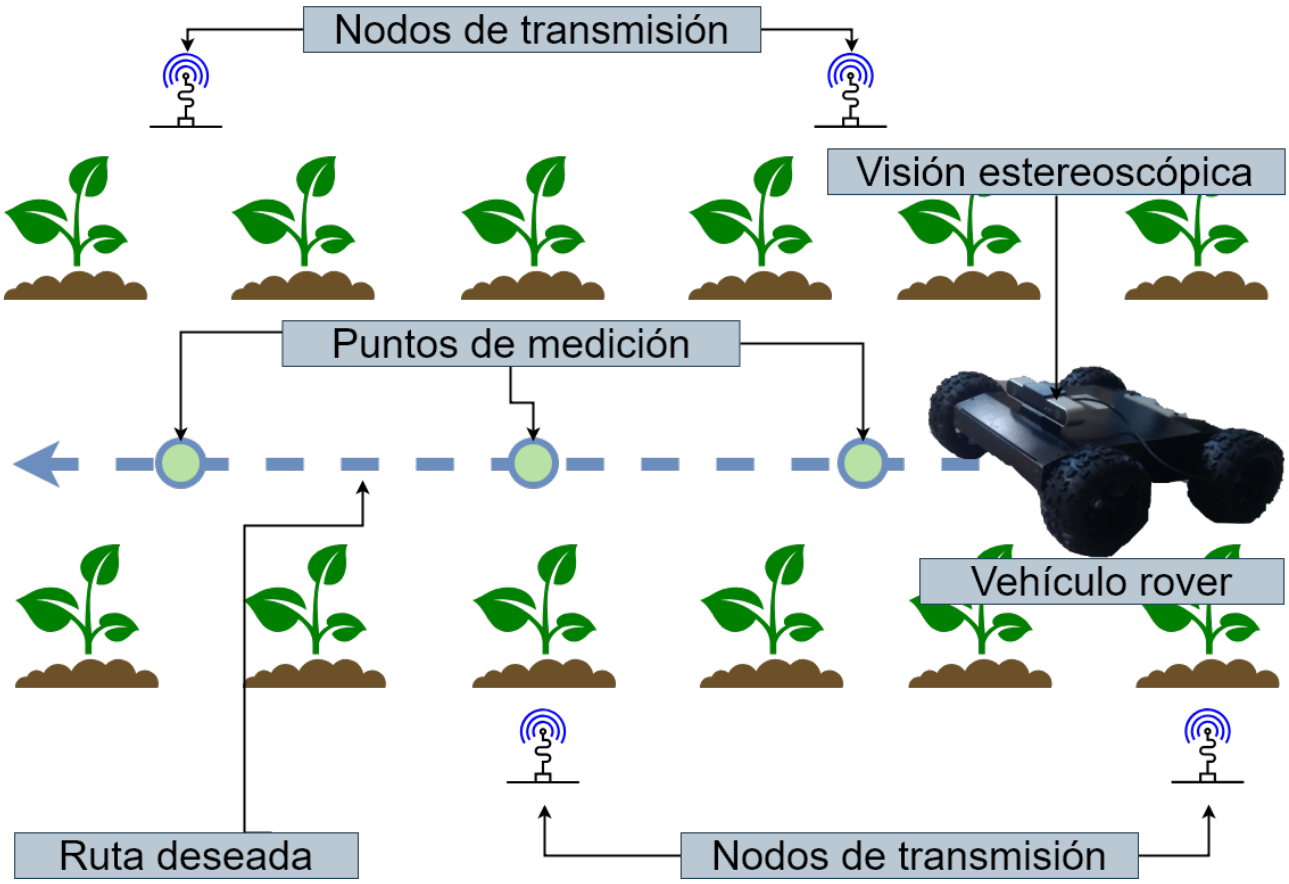



**Figura 1.1: Modelo del proyecto.**

## 1.2. Objetivos

### 1.2.1. General

Estimar el movimiento de UGVs basándose en visión computacional y el indicador de intensidad de la señal recibida (RSSI, por sus siglas en inglés). Ver Figura 1.1

### 1.2.2. Específico

- Modelar y estimar la posición de UGVs mediante la técnica de estimación basado en RSSI.
- Implementar los algoritmos de estimación de la posición de UGVs dada trayectorias de movimiento del UGV.

- Análisis de precisión de la estimación de la ruta y el recorrido de la trayectoria del UGVs.
- Implementación de algoritmos de visión computacional para la navegación de UGVs.

## 1.3. Justificación

El proyecto trata de la implementación de un sistema de seguimiento de trayectoria autónomo para UGVs utilizando un sistema de visión y de técnicas de estimación de movimiento basadas en RSSI para el desarrollo de ciudades inteligentes. Para la ejecución exitosa del proyecto, se requiere experiencia en dos áreas distintas. En primer lugar, para diseñar y evaluar la estimación de movimiento basada en RSSI integrada en una estación terrestre para UGVs, es necesario contar con un equipo de trabajo con sólidos conocimientos sobre comunicación inalámbrica y sistemas embebidos. En segundo lugar, para el diseño e implementación del sistema de visión, se requiere experiencia en procesamiento de señales e imágenes. Considerando estas características, se propone la colaboración de investigadores del ITM, y de la empresa "Spring Labs" ubicada en "Calle Ote. 67 número 2912 Asturias Ciudad de México", con experiencia en métodos de visión para la navegación de vehículos no tripulados y con experiencia en mecatrónica para el diseño e instrumentación con sistemas embebidos. Esto resultará en una colaboración efectiva para alcanzar los objetivos del proyecto.

## 1.4. Estado del Arte

Actualmente existen en la literatura muchos estudios que hablan sobre el uso de RSSI, en sistemas con antenas de transmisión y nodos receptores, para la estimación de la distancia y posicionamiento geográfico, en su mayoría estos estudios tratan sobre experimentos realizados en ambientes interiores [1] [2] [3]. Uno de los primeros estudios realizados sobre este tema es [4], en este trabajo se realizó la medición de la distancia entre una referencia fija como transmisor

y un receptor móvil; el modelo de propagación de señal utilizado es parecido al que se usa en este trabajo, de igual forma los resultados obtenidos son similares; por ultimo el error estimado obtenido es 1.2m.

Un factor importante en la estimación de la distancia es el modelo de distribución del RSSI, aunque varios estudios usan un modelo normal [5] [6], pero en [7] se propone el modelo de distribución de Weibull; para esto realizan un sistema de posicionamiento basado en un servidor y varios teléfonos móviles; además de hacer énfasis en el modelo de distribución de Weibull para RSSI este trabajo también hace énfasis en los tiempos de descubrimiento y conexión de dispositivos entre mediciones, que sirven para calcular el tiempo de muestreo del sistema.

A diferencia del artículo anterior, en [6] se propone el modelo de propagación de señal punto a punto con un filtro gaussiano. El sistema de posicionamiento presentado cuenta con 4 nodos fijos y un objetivo móvil, también cuenta con dos fases para localización: entrenamiento offline y luego actualización online; en este trabajo se propone un filtro de distancias ponderadas para disminuir el error.

Es importante considerar para la localización de nodos móviles factores como el consumo energético y el tiempo de muestro para la localización de nodos .En [8] se toman en cuenta estos puntos y describe un algoritmo basado en 50 muestras para la localización. El concepto de tomar más de una muestra por punto de medición, fue analizado en está tesis para 30 muestras por punto.

En [9] se presentan otros dos modelos para el tratamiento de la posición con RSSI. Los dos modelos de localización de interiores presentados son: alta precisión interior (HIL) y baja precisión interior (LIL). En ambos métodos se divide el área en secciones y a partir de estas se va determinando en cuál sección se localiza el objeto, la diferencia es que HIL genera regiones más pequeñas que LIL. Los resultados son precisos y se pudo determinar la posición del módulo en una región.

Son variados los dispositivos usados en las investigaciones para medir RSSI, por ejemplo, en [10]

se realizó una comunicación punto a punto utilizando la tecnología IEEE 802.15.4. Por otro lado en [5] se emplearon iBeacons, también en esta investigación se muestran diferencias de medir RSSI con diferente orientación de la antena. Este trabajo analiza algoritmos con entrenamiento tanto offline como online y ponderado iterativo KNN. En los resultados se aprecia un mapa de calor del error de la región limitada para el experimento.

En [11] se hace una verificación de un modelo uno a uno simulado con MATLAB, el modelo usa parámetros como ganancia de las antenas, frecuencia de transmisión, perdidas y posición del transmisor. Se tomaron en cuenta dos escenarios uno en un cuarto sin interferencia y otro en un corredor común. Se realizó una estimación del modelo del ruido en los canales.

# Capítulo 2

# Marco Teórico

En este capítulo se presentan los algoritmos, tecnologías y herramientas utilizados para el trabajo de tesis; este capítulo está dividido en cuatro partes. La primera parte describe el Bluetooth de bajo consumo, sus perfiles, protocolos y configuraciones; la segunda parte presenta el modelo de estimación de distancia usando RSSI en dispositivos BLE; en la tercera parte se habla sobre el algoritmo de trilateración; por último, en la parte 4, se describe el flujo para la detección de obstáculos.

## 2.1. Bluetooth de bajo consumo

El bluetooth de baja energía (BLE) es una tecnología de comunicación inalámbrica que opera a 2.4 GHz, la tabla 2.1 presenta una comparación entre el BLE y Bluetooth. Las características del protocolo BLE no permiten que transmita grandes cantidades de información, pero maximiza la vida del dispositivo y minimiza los tiempos de conexión [12]. En la tabla 2.1 se muestran las características principales de este protocolo en comparación del Bluetooth.

**Tabla 2.1: BLE vs Bluetooth**

| Caracteristica | Bluetooth | BLE |
|---|---|---|
| Velocidad de transmición. | 2.1 Mbps | 0.26 Mbps |
| Consumo energético porcentual | 100 % | 1 % - 5 % |
| Latencia de transmición | 100ms | 3ms |

### 2.1.1. Perfiles de Acceso Genérico

Los procedimientos genéricos relacionados con el descubrimiento de dispositivos Bluetooth y los aspectos de administración de enlaces de la conexión a dispositivos Bluetooth son definidos por el Perfil de acceso genérico (GAP). Además, este perfil incluye requisitos de formato comunes para los parámetros accesibles en el nivel de la interfaz de usuario. Existen 4 perfiles de acceso genérico:

- Broadcaster: Un dispositivo que funciona en la función de transmisor puede enviar eventos publicitarios. Tiene un transmisor y puede tener un receptor.
- Observer role: Un dispositivo que funciona en la función de Observador es un dispositivo que recibe eventos publicitarios. Tiene un receptor y puede tener un transmisor.
- Peripheral role: Dispositivo que se anuncia mediante el uso de paquetes de publicidad conectables y se convierte en esclavo una vez conectado; acepta el establecimiento de un enlace físico. Un dispositivo que funcione en el rol de «Periférico» estará en el rol de «Esclavo» en el estado de conexión de la capa de enlace. Un Periférico tiene un transmisor y un receptor.
- Central role: Define un dispositivo que inicia conexiones a periféricos y, por lo tanto, se convertirá en maestro cuando se conecte; admite la función Central e inicia el establecimiento de una conexión física. Un dispositivo que funcione en el rol «central» estará en el rol «principal» en la conexión de la capa de enlace. Una central tiene un transmisor y un receptor.

### 2.1.2. Perfil de Atributo Genérico

El Perfil de atributo genérico define un marco de servicio genérico utilizando la capa de protocolo ATT. Este marco define los procedimientos y formatos de los servicios y sus características. Define los procedimientos para el servicio, la característica y la descripción, la lectura, escritura, notificación e indicación de las características, así como la configuración de la difusión de las características. Los de perfiles de atributo genérico son:

- GATT Client: Este es el dispositivo que quiere datos. Inicia comandos y peticiones hacia el servidor GATT. Puede recibir respuestas, indicaciones y datos de notificaciones enviados por el servidor GATT.
- GATT Server: este es el dispositivo que tiene los datos y acepta los comandos y solicitudes entrantes del Cliente GATT y envía respuestas, indicaciones y notificaciones a un Cliente GATT.

### 2.1.3. Protocolo de Atributo

La capa de protocolo de atributo define una arquitectura de clientes y servidor por encima del canal de transporte lógico BLE. El protocolo de atributos permite que un dispositivo denominado Servidor GATT exponga un conjunto de atributos y sus valores asociados a un dispositivo par denominado Cliente GATT. Estos atributos expuestos por el servidor GATT pueden ser descubiertos, leídos y escritos por un cliente GATT, y pueden ser indicados y notificados por el servidor GATT. Todas las transacciones sobre atributos son atómicas.

### 2.1.4. Configuración de las antenas y el explorador

El primer paso consiste en conseguir las mediciones elegir los perfiles con los que se estarán comunicando las antenas y el explorador. Es importante resaltar que lo que se desea es que el explorador emita el paquete de conexión, para que las antenas inician la conexión o midan

el RSSI a partir de este paquete. Existen dos posibles configuraciones que cumplen con esta condición.

### Emisor-Observador

En esta configuración no existe conexión entre las antenas y el explorador. Las antenas se encuentran en un estado constante de búsqueda de dispositivos, una vez que se encuentra el explorador se toma lectura del RSSI mediante el paquete de publicidad; por su parte el explorador se encuentra todo el tiempo emitiendo paquetes de publicidad. La tabla 2.2 muestra el GAP asociado al tipo de nodo BLE y la cantidad de nodos de cada tipo necesarios. Esta configuración presentó problemas con las antenas al momento de encontrar el explorador debido a que existían demasiados dispositivos en modo emisor; los tiempos de recolección de datos fueron muy inferiores a establecer una conexión (5 segundos).

**Tabla 2.2: Configuración A: Emisor-Observador**

| Tipo | GAP | Cantidad |
|---|---|---|
| Explorador. | Broadcaster | 1 |
| Antena | Observer | 5 |

### Periférico-Central.

La configuración Periférico-Central puede comportase como una configuración Emisor-Observador y además permite establecer conexión entre una antena y el explorador. Aprovechando la conexión entre la antena y el explorador es posible tomar un mayor número de mediciones. La tabla 2.3 muestra el GAP asociado al tipo de nodo BLE y la cantidad de nodos de cada tipo necesarios. Se eligió esta configuración por la ventaja en tiempo que implica tener una conexión entre antena y explorador.

**Tabla 2.3: Configuración B: Periférico-Central**

| Tipo | GAP | Cantidad |
|---|---|---|
| Explorador. | Periférico | 1 |
| Antena | Central | 5 |

## 2.2. Estimación de distancia usando RSSI en BLE

Distintos trabajos presentan la relación que existe entre el RSSI y la distancia, para este trabajo se usa el modelo propuesto en [6] descrito en la ecuación 2.2.

$$P = -10nlog10(d) + A \tag{2.1}$$

En donde $P$ representa la intensidad de señal recibida; $d$ es la distancia que existe entre el emisor y receptor; $A$ es el valor de la señal recibido a 1 metro de distancia; y $n$ es el factor de atenuación. Despejando $d$ de la ecuación 2.2, se obtiene:

$$d = 10^{\frac{P-A}{-10n}} \tag{2.2}$$

Para encontrar los valores de $d$ y $A$ que minimicen el error se usará el modelo de mínimos cuadrados (LS). Los pasos para seguir son los siguientes:

1. Definir el número de muestras $m$ y la distancia $d$ para cada muestra.
2. Recolectar los valores de RSSI para cada muestra y asociarlos con su respectiva distancia.
3. Resolver los siguientes cambios de la ecuación 2.2 para $\beta_0$ y $\beta_1$.

$$\beta_0 = \frac{n\sum X_i^2 \sum Y_i - \sum X_i \sum X_i Y_i}{n\sum X_i^2 - (\sum X_i)^2} \tag{2.3}$$

$$\beta_1 = \frac{n\sum X_i \sum Y_i - \sum X_i \sum Y_i}{n\sum X_i^2 - (\sum X_i)^2} \tag{2.4}$$

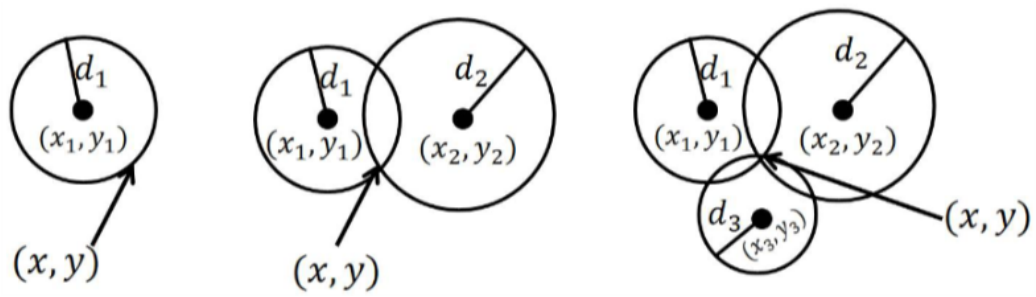



**Figura 2.1: Planteamiento geométrico para hallar el punto** $(x, y)$

## 2.3. Trilateración

La trilateración es un método matemático para determinar la posición de un punto conociendo su distancia a tres puntos de referencia. A continuación se describe el método de acuerdo a [13] [8].

De la Figura 2.1 se obtiene la ecuación 2.5 la cual muestra la distancia $d_i$ para cada punto $(x_i, y_i)$ con respecto al punto $(x, y)$:

$$d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}. \tag{2.5}$$

El error $e_i$ que existe entre cada distancia estimada $r_i$ y la distancia real $d_i$ esta dado por

$$e_i = r_i - \sqrt{(x - x_i)^2 + (y - y_i)^2}. \tag{2.6}$$

Dado que se desea minimizar el error a 0, entonces:

$$e_i = r_i - \sqrt{(x - x_i)^2 + (y - y_i)^2} = 0. \tag{2.7}$$

Elevando ambos lados al cuadrado y despajando para $(x, y)$, se obtiene:

$$(x^2 + y^2) + x(-2x_i) + y(-2x_i) = r_i^2 - x_i^2 - y_i^2 \tag{2.8}$$

Para eliminar los términos cuadráticos restamos la ecuación 2.8 evaluada para la $k$-ésima distancia,

$$2x(x_k - x_i) + 2y(y_k - y_i) = r_i^2 - r_k^2 - x_i^2 - y_i^2 + x_k^2 + y_k^2 \tag{2.9}$$

La ecuación 2.9 es lineal, por lo tanto el sistema también es lineal y se representa como:

$$Ax = B \tag{2.10}$$

en donde,

$$A = \begin{bmatrix} 2(x_k - x_1) & 2(y_k - y_1) \\ 2(x_k - x_2) & 2(y_k - y_2) \\ \vdots & \vdots \\ 2(x_k - x_{k-1}) & 2(y_k - y_{k-1}) \end{bmatrix} \tag{2.11}$$

$$B = \begin{bmatrix} r_1^2 - r_k^2 - x_1^2 - y_1^2 + x_k^2 + y_k^2 \\ r_2^2 - r_k^2 - x_2^2 - y_2^2 + x_k^2 + y_k^2 \\ \vdots \\ r_{k-1}^2 - r_k^2 - x_{k-1}^2 - y_{k-1}^2 + x_k^2 + y_k^2 \end{bmatrix} \tag{2.12}$$

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.13}$$

La solución por mínimos cuadrados de $Ax = B$ es

$$x = (A^T A)^{-1} - A^T B \tag{2.14}$$

Por lo que las coordenadas de um punto $(x, y)$ se pueden calcular a partir de la ecuación anterior.

## 2.4. Detección de obstáculos

Para el proceso de obtención de obstáculos son necesarias 3 etapas de procesamiento, ver Figura 2.2.

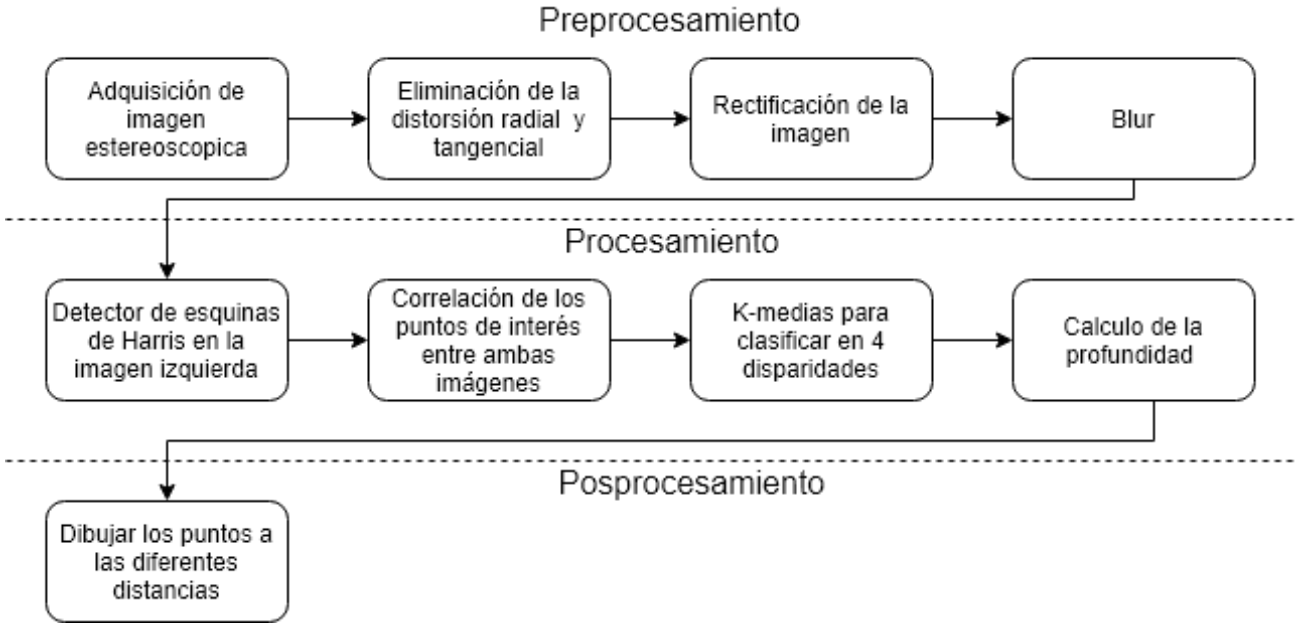



**Figura 2.2: Flujo de procesamiento.**

La primera etapa corresponde al preprocesamiento de la imagen, en esta se realizan los siguientes pasos:

1. Adquirir la imagen - Frame de la cámara estereoscópica.

2. Corregir distorsiones - Tangencial y radial debidas al lente de la imagen.

3. Rectificación - Operación para que los pixeles estén alineados con respecto al eje horizontal.

4. Filtrado - Filtro blur para suavizar el ruido en las imágenes.

La segunda etapa es el procesamiento del frame capturado, en esta etapa se calcula disparidad y se averigua si existen o no obstáculos. Los pasos a en esta etapa son:

1. Detección de puntos de interés - Mediante el detector de esquinas de Harris se hallan puntos significativos de la imagen; este procesamiento solo se lleve en el lado izquierdo del frame.

2. Cálculo de la disparidad - Usando el método de suma de diferencias absolutas se hallan los puntos correspondientes en lado derecho del frame, después se calcula la disparidad entre los puntos encontrados y lo originales.

3. Niveles de disparidad - Usando K-medias se agrupan las disparidades en 4 niveles.

4. Cálculo de profundidad - Se calcula la profundidad para los 4 niveles de disparidad.

La última etapa corresponde al post-procesamiento, en este caso es la toma de decisiones con la información obtenida de la etapa anterior.

### 2.4.1. K-medias

Es un algoritmo de agrupación de elementos o clustering. Este algoritmo resuelve un problema de optimización de mínima distancia entre los elementos a agrupar y los clusters o centros de grupo [14]. El modelo matemático que representa este problema es el siguiente.

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x_j \in S_i} \| x_j - \mu_i \|^2 \tag{2.15}$$

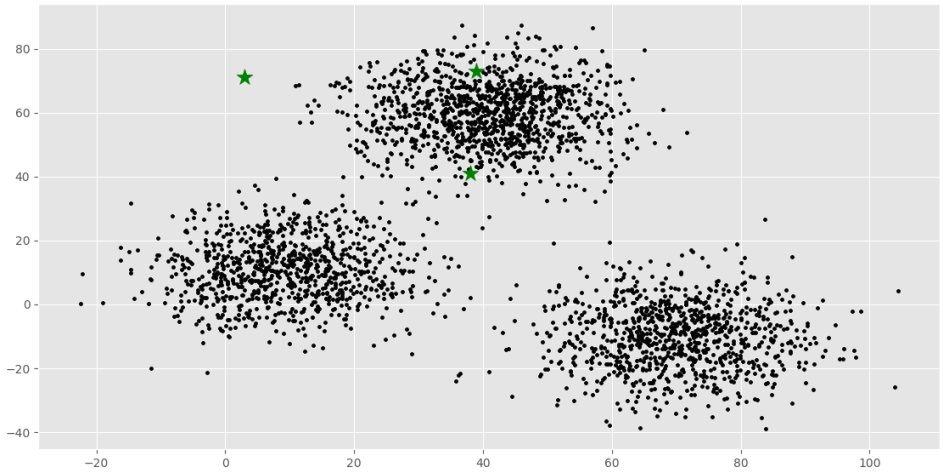
donde $\mu_i$ es la media de puntos en $S_j$

El algoritmo para implementar las K-medias es el siguiente:

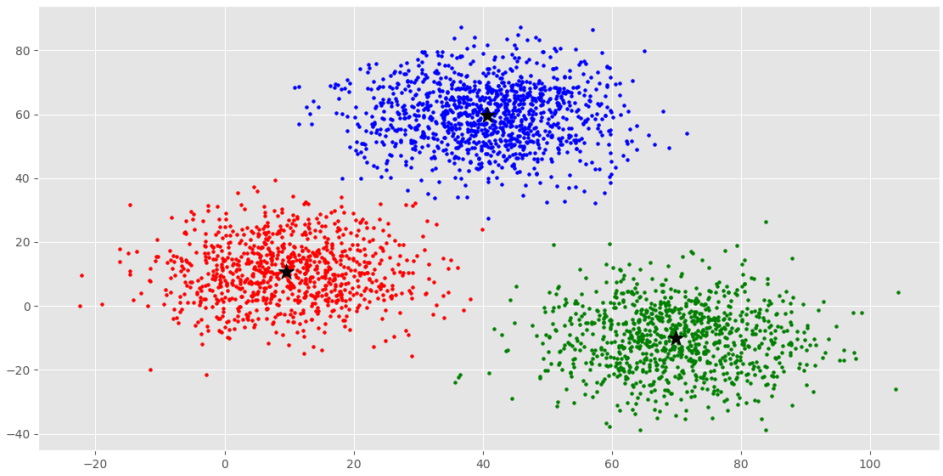
```
1. Sitúa k puntos en el espacio de objetos. Estos puntos serán los centroides
   iniciales.
2. Asigna a cada objeto el centroide más cercano a él.
3. Calcula la posición de los k centroides a partir de los objetos asignados a
   ellos.
```

```
4. Repetir hasta que la diferencia entre cada centroide y su posición anterior
   sea menor al error deseado.
```

**Código. 2.1: k-medias.**




(a) Centroides iniciales




(b) Centroides finales

**Figura 2.3: K-medias.**

Una implementación de este algoritmo se puede encontrar en openCV en la función **kmeans()**.

Los parámetros de entrada son:

- samples - Array de una columna tipo float32.
- nclusters - Número de centroides o clusters.
- criteria - Criterio para detener el algoritmo.
- attempts - Indicador para especificar el número de veces que se ejecuta el algoritmo utilizando diferentes etiquetas iniciales.
- flags - Indicador para inicializar con centroides aleatorios.

### 2.4.2. Detector de esquinas de Harris

Una esquina es una región de la imagen con cambios de intensidad en diferentes direcciones. Las esquinas se ven poco afectadas por rotaciones y escalamientos lo que las hace buenos puntos de interés. Harris y Stephens [15] presentan un modelo basado en la diferencia de intensidad para un desplazamiento de $(x, y)$ en todas las direcciones de una imagen $I$ y una región $(u, v)$.

$$S(x, y) = \sum_u \sum_v = w(x, y)(I(u + x, v + y) - I(u, v))^2 \tag{2.16}$$

El detector de esquinas de Harris se encuentra implementando en OpenCV en la función **cornerHarris()** esta requiere los siguientes parámetros de entrada.

- img - Imagen de entrada, debe ser en escala de grisis y tipo float32.
- blockSize - Tamaño de la ventana para la detección de esquinas.
- ksize - Apertura del derivado Sobel utilizado.
- k - Parámetro libre en la Ecuación del detector de Harris.

(a) Original

(b) Detector de esquinas de Harris

**Figura 2.4: Detector de esquinas de Harris.**

### 2.4.3. Disparidad y profundidad

En un par de imágenes estéreo rectificadas la disparidad es el desplazamiento en azimut que existe entre los pixeles de las imágenes. Es posible estimar la distancia que existen entre la cámara y un objeto en el mundo real utilizando la disparidad [16]. En la Figura 2.5 se presenta un modelo matemático que relaciona la disparidad de una imagen con la profundidad con un objeto en el mundo real, aplicando semejanza de triángulos en el diagrama se obtiene:

$$disparity = x - x\prime = \frac{Bf}{Z} \tag{2.17}$$

Despejando la distancia $Z$ de la ecuación 2.17

$$Z = \frac{Bf}{disparity} \tag{2.18}$$

La ecuación 2.18 presenta la distancia en relación a la disparidad de la imagen.

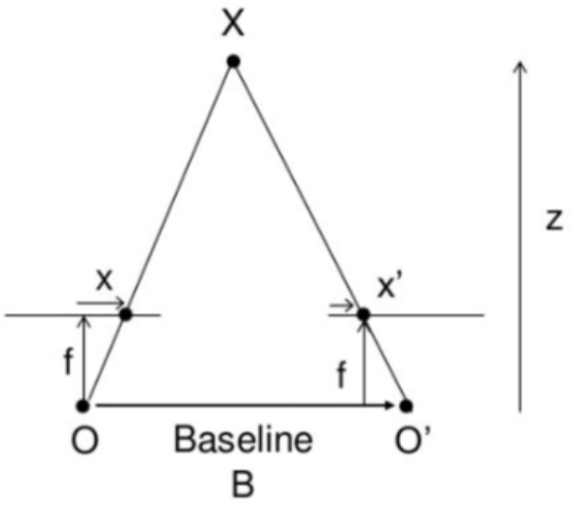



**Figura 2.5: Modelo de la disparidad en cámaras estero.**

### 2.4.4. Suma de diferencias absolutas

La suma de diferencias absolutas (SAD, por sus siglas en inglés) es un método para encontrar similitud entre bloques o ventanas de uno o varios fotogramas. Para una ventana centrada en un pixel de interés se calcula la diferencia absoluta con ventanas de tamaño equivalente en un región de interés, al finalizar el recorrido por las diferencias que sumen el menor valor corresponden a la ventana con mayor similitud [17]. Esto se ilustra en la Figura 2.6 en donde $x$ y $x'$ son los fotogramas a comparar y la disparidad esta representa por la diferencia entre pixeles.

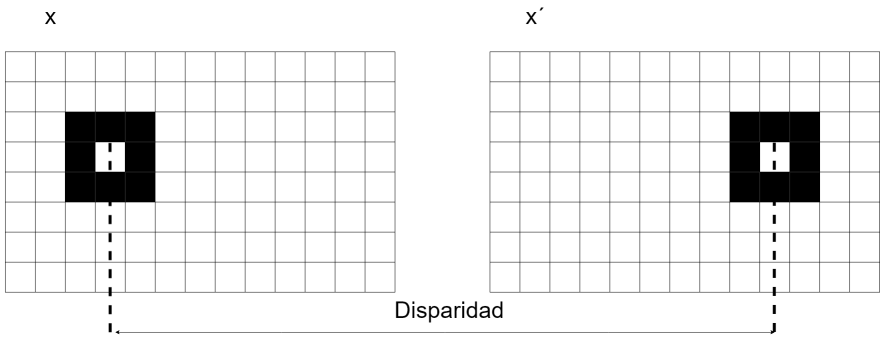



**Figura 2.6: Representación de la disparidad en un par estéreo.**

En la ecuacion 2.19 muestra como calcular la disparidad para cada pixel de dos frames $I_1$ e $I_2$.

$$\sum_{(i,j)\in W} \mid I_1(i,j) - I_2(x+i, y+j) \mid \tag{2.19}$$

El procedimiento para implementar el algoritmo SAD es el siguiente:

```
1. Repetir para cada punto de interés.
1.1 Seleccionar una ventana w1 centrada el punto de interés.
1.2 Repetir para cada ventana Wi en la región de búsqueda.
1.2.1 Hacer a_i = Sum(|w1−wi|)
1.3 Elegir el a_i mínimo.
```

**Código. 2.2: SAD.**

# Capítulo 3

# Desarrollo

Este capítulo describe el hardware empleado para el UGV así como el software elaborado para el mismo. Al igual que el capítulo 2, este está dividido en 4 partes. En el primer apartado se describen los elementos que conforman el UGV; la segunda parte describe la calibración de los módulos BLE; en la parte tres del capítulo se propone un arreglo espacial de nodos BLE para minimizar la cantidad de estos para un área específica; el último apartado presenta el pseudocódigo para la detección de obstáculos con Python 3 y OpenCV 3.4

## 3.1. Vehículo explorador

Para el vehículo de exploración, ver Figura 3.1, se decidió construir un carro 4x4 con motores DC de 12V y 2A. La tarjeta de procesamiento de este equipo es una Odroid Xu 4 la cual cuenta con un procesador Octacore Exynos5422, tarjeta gráfica Mali-T628 y 2GB de memoria RAM; también cuenta con un coprocesador CY5674 PRoC™BLE SMA; una cámara de visión estereoscópica ZED; una unidad de medida inercial (IMU); y un sistema de alimentación y carga.

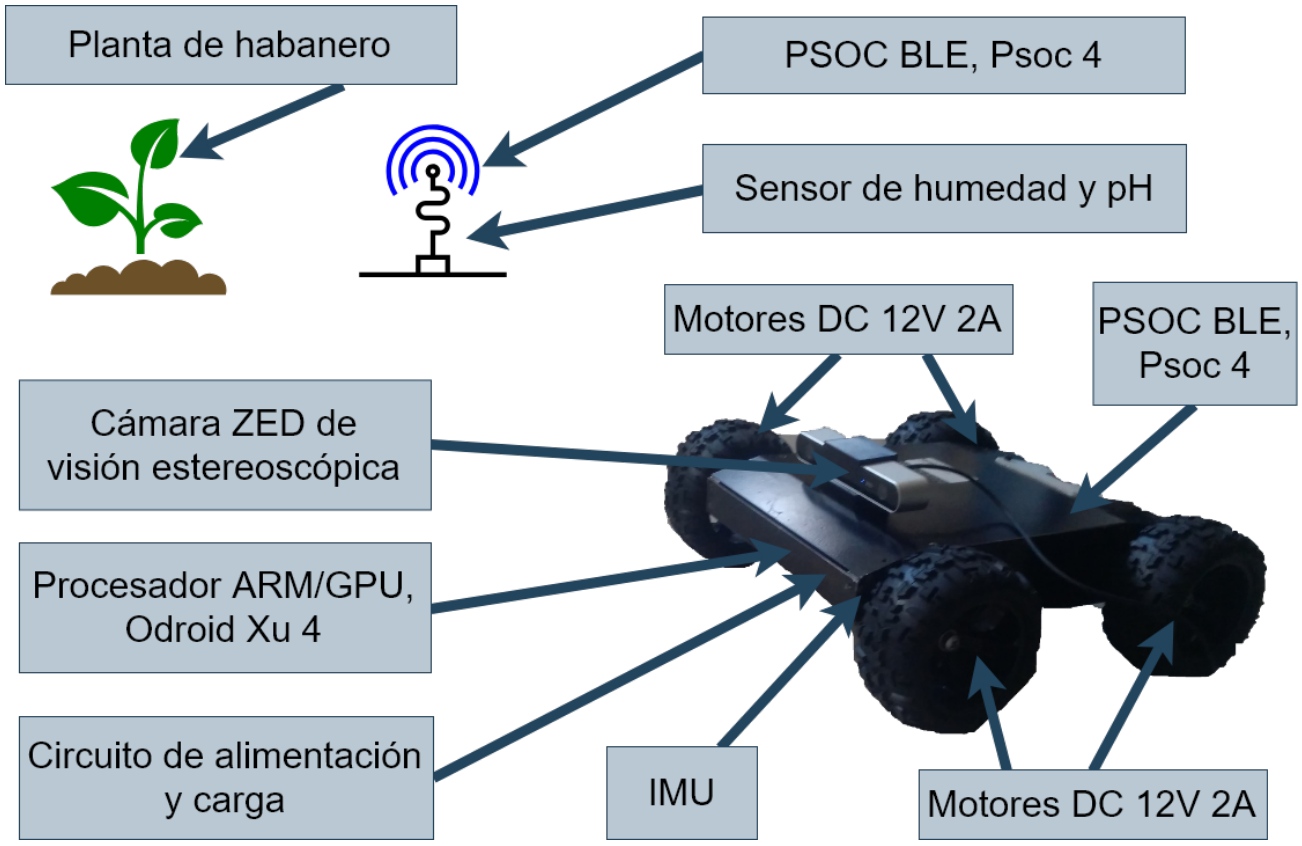



**Figura 3.1: Características del vehículo explorador**

## 3.2. BLE

Para la medición de RSSI se utilizaron seis dispositivos CY5674 PRoC™BLE SMA de CYPRESS. Esto módulos cuentan con SOC compuesto por un procesador ARM®Cortex®-M0 a 48-MHz, radio BLE, dos bloques de comunicación serial (SCBs), un ADC de 12-bit, cuatro Timer/Counter/PWMs, cuatro PWMs adicionales, $I^2S$ para audio digital y un controlador para LCD. La Figura 3.2 las características y descripción de los distintos puertos de entrada y salida del Proc BLE SMA.

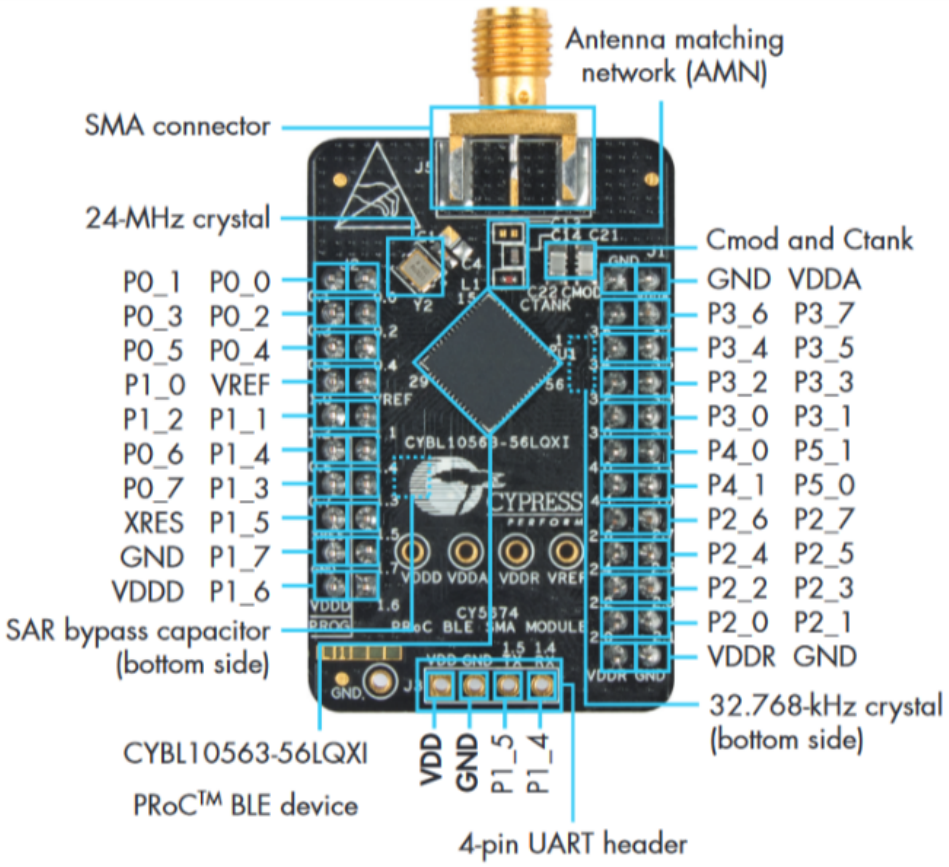



**Figura 3.2: Características y descripción de puertos del Proc BLE SMA**

### Configuración de los módulos BLE

Para la calibración se realizaron mil mediciones de RSSI punto a punto entre el explorador y cada una de las antenas a distancias de un metro, dos metros, tres metros, cuatro metros y 5 metros; siendo un total de cinco mil datos por antena; ver Figura 3.3

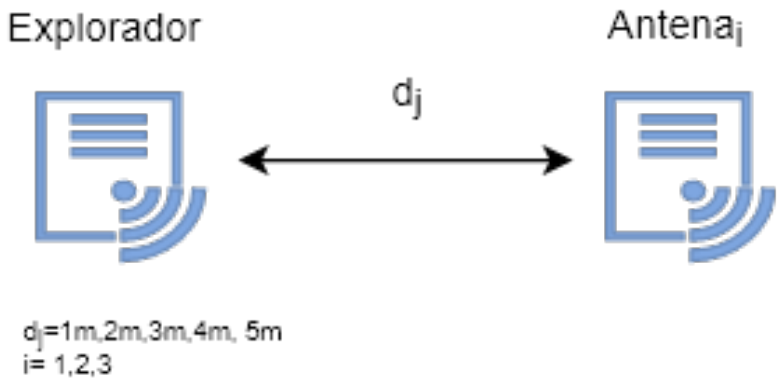



**Figura 3.3: Medición punto a punto de RSSI, para dj=1m,2m,3m,4m y 5m.**

Para desarrollar un firmware en psoc es necesario configurar y diseñar el hardware del SOC para ser utilizado, esto puede ser hecho mediante código o diagrama de bloques. En este caso se uso el diagrama de bloques para configurar los nodos. En la Figura 3.4 se ve la configuración de hardware para las antenas, estas cuentan con: la radio BLE; dos leds indicadores uno para indicar encendido y otro para mostrar conexión; comunicación UART a 115200 bps para transmitir las lecturas al radio Wifi. El explorador tiene una configuración similar pero este no cuenta con el módulo UART. Se codificaron dos firmwares, uno para cada tipo de nodo. El firmware del nodo central se encarga de buscar y establecer comunicación con un nodo periférico, una vez establecida la comunicación mide el RSSI y envía el valor por UART, por último termina la conexión e inicia la búsqueda de otro nodo. Por su parte el nodo periférico está en espera del nodo central.

La adquisición de datos se llevó mediante el desarrolló una API REST con el lenguaje NodeJs y el framework Express. Express es un framework para creación de aplicaciones web en NodeJs.

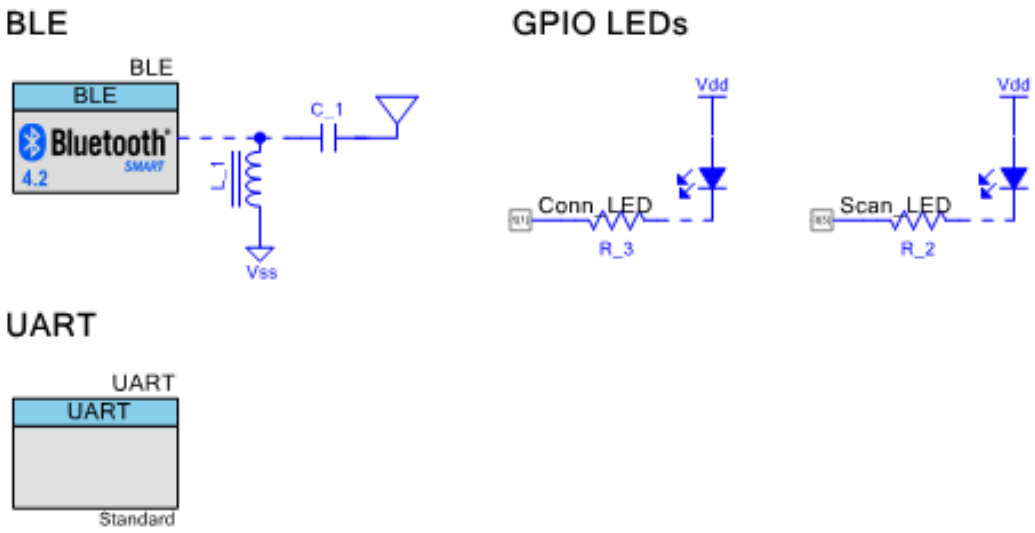



**Figura 3.4: Diagrama de bloques de la configuración de hardware para los nodos**

La API cuenta con dos rutas: *connect* y *data*. La ruta *connect* es tipo post y no requiere objeto de entrada, esta ruta es para comprobar la comunicación entre el módulo central y la API. La ruta *data* también es tipo post y requiere un objeto entrada el cual tiene dos campos id y rssi.

### 3.2.1. Interfaz de adquisición de datos

La adquisición de datos se llevó mediante el desarrolló una API REST con el lenguaje NodeJs y el framework Express. Express es un framework para creación de aplicaciones web en NodeJs. La API cuenta con dos rutas *connect* y *data*. La ruta *connect* es tipo post y no requiere objeto de entrada, esta ruta es para comprobar la comunicación entre el módulo central y la API. La ruta *data* también es tipo post y requiere un objeto entrada el cual tiene dos campos id y RSSI.

## 3.3. Trilateración

Con el fin de minimizar la cantidad de nodos para cubrir una región, se analizaron tres distribuciones de antenas. Para cada arreglo se forman 4 tipos de zonas las cuales dependen de la cantidad señales de antenas en ese punto. También se calcula la densidad de nodos por m2, para esta medida se considera una superficie rectangular la cual encierra las zonas de cobertura 3 y 4. Por último se calcula el porcentaje de cobertura útil en razón de la cobertura total.

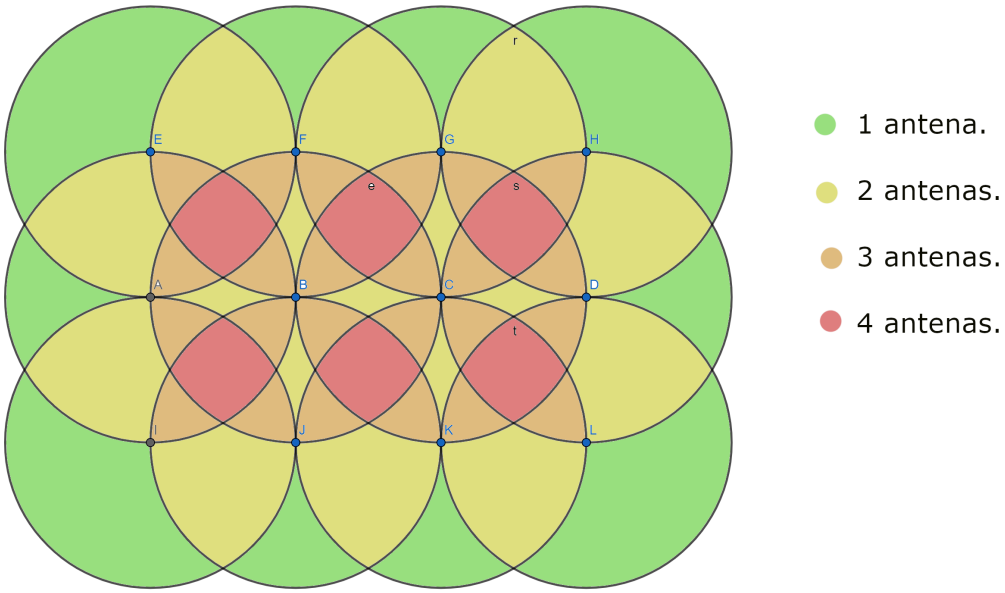



**Figura 3.5: Distribución de nodos en vértices de rejilla cuadrada.**

La primera distribución consiste en acomodar los nodos en una rejilla conformada por cuadrados cuyo lado es igual a la distancia máxima de separación entre antenas, ver Figura 3.5. En esta distribución la densidad de nodos es 2 $nodos/m^2$ y el porcentaje de área útil es 27.23 %. Es importante notar que esta es al distribución con menor porcentaje de área útil.

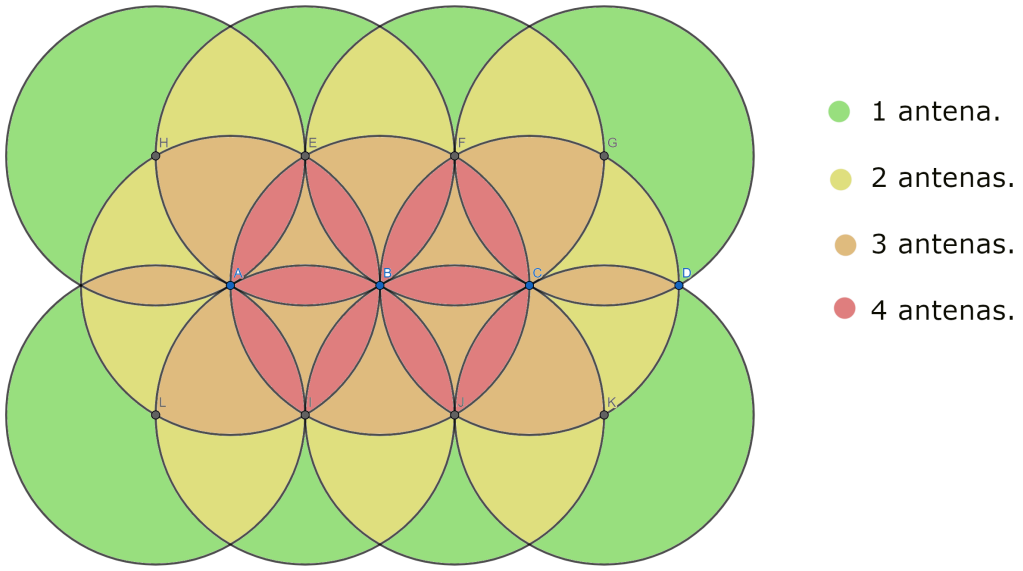



**Figura 3.6: Distribución de nodos en vértices de rejilla triangular.**

La segunda propuesta es una rejilla hecha con triángulos equiláteros cuyo lado mide la máxima separación posible entre antenas; esta rejilla presenta una ventaja con respecto a una rejilla cuadrada en que las regiones con poca cobertura de RSSI se encuentran en los alrededores de la región de interés, ver Figura 3.6. Para esta distribución la densidad de nodos es 2.12 $nodos/m^2$ y tiene un porcentaje de cobertura útil del 33.54 %.

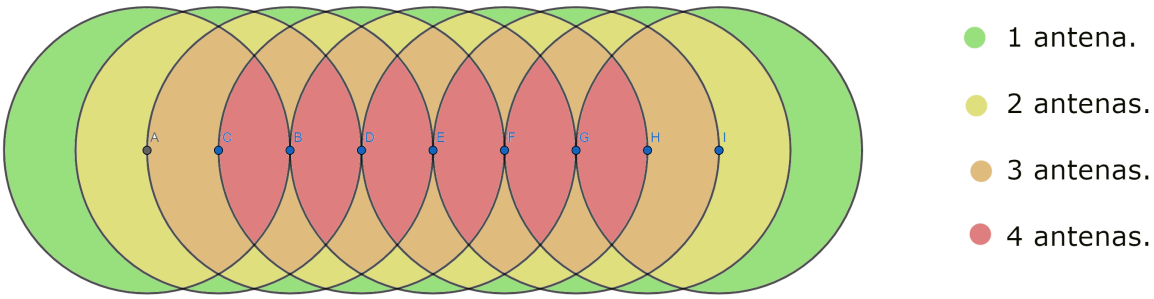



**Figura 3.7: Distribución de nodos en línea recta.**

Por último se presenta una distribución en la que todos los nodos se encuentran alineados, ver Figura 3.7, en este caso la densidad de nodos es 1.25 $nodos/m^2$ y el porcentaje de cobertura útil es del 44.88 %. Aunque esta rejilla presenta el mayor porcentaje de cobertura útil, no puede ser usada ya que para implementar el algoritmo de trilateración es necesario que todos los nodos no se encuentren en la misma recta.

### 3.3.1. Cálculo de cantidad nodos

Partiendo de la distribución propuesta por la Figura 3.6 y conociendo la distancia máxima de separación que puede existir entre dispositivos para su correcto funcionamiento se puede calcular el área de cobertura y la cantidad de nodos para cubrirla. Suponiendo que la región de interés está compuesta por $k$ canales de largo $L$ y ancho $a$; entonces $d_{max}$ es la distancia máxima entre nodos; y $b$ la distancia que existe entre canales, ver Figura 3.8.

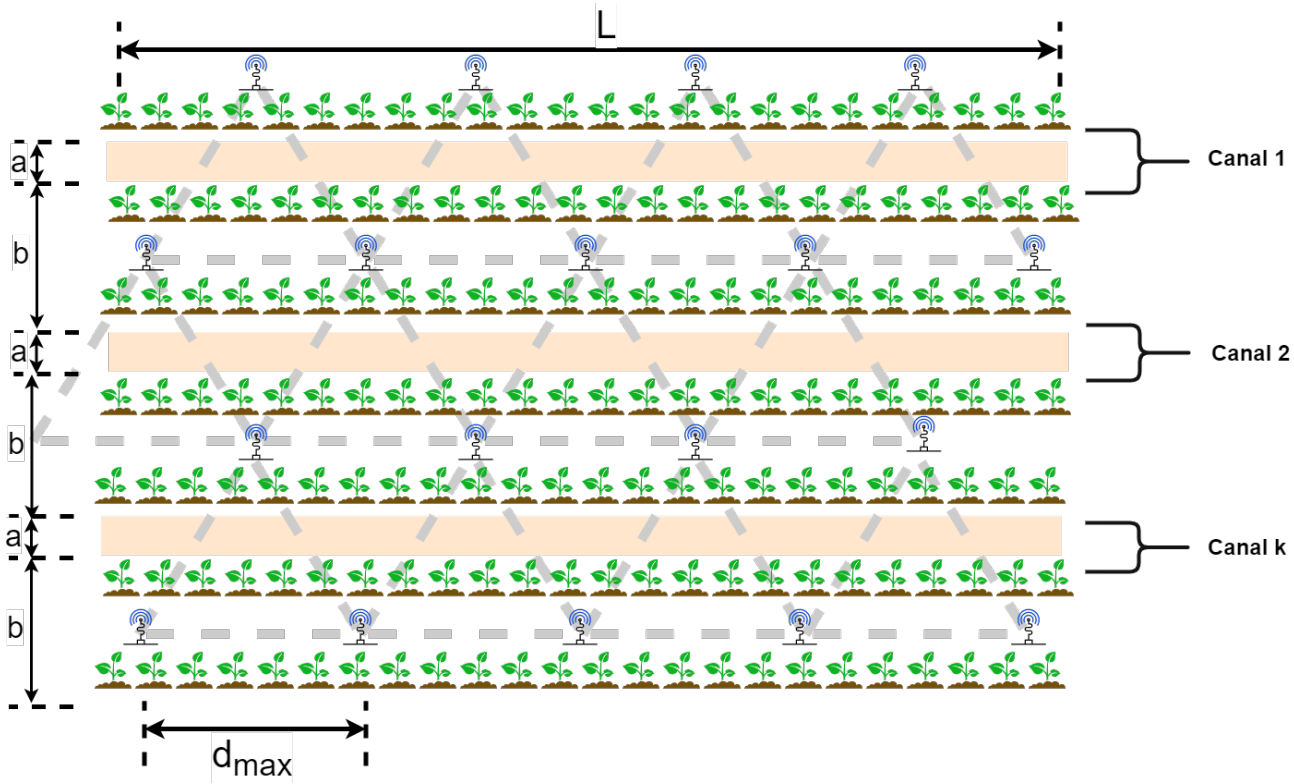



**Figura 3.8: Cálculo de la cantidad de nodos necesarios para la distribución propuesta.**

Entonces siempre que $a + b \leq \sqrt{3}d_{max}$ la cantidad de nodos $n$ necesaria para cubrir la región es:

$$n = (k + 1)(\frac{L}{d_{max}} + 1) \tag{3.1}$$

La distribución de los sensores se realizó siguiendo la propuesta de rejilla triangular con lado 1m y la distancia del lado del triángulo se eligió con los resultados de la caracterización de los nodos. Se acomodaron cuatro antenas en los vértices de un trapecio regular con una base mayor de 2 metros, base menor de 1 metro y ángulo menor de 60 grados; se puso una antena en el punto medio de la base mayor, el origen de este plano se puso en el punto medio de la base menor. El recorrido del robot se dio sobre 27 puntos de prueba ubicados en el interior del trapecio, ver Figura 3.9. Cada medición cuenta con 100 valores de RSSI para cada antena la comunicación entre la antena y el explorador es Point-to-Point, pero la comunicación entre las

antenas y el dataloger es tipo estrella. El tiempo promedio para la toma de 100 valores fue de 2.85 segundos dando un total de 14.25 segundos por punto.

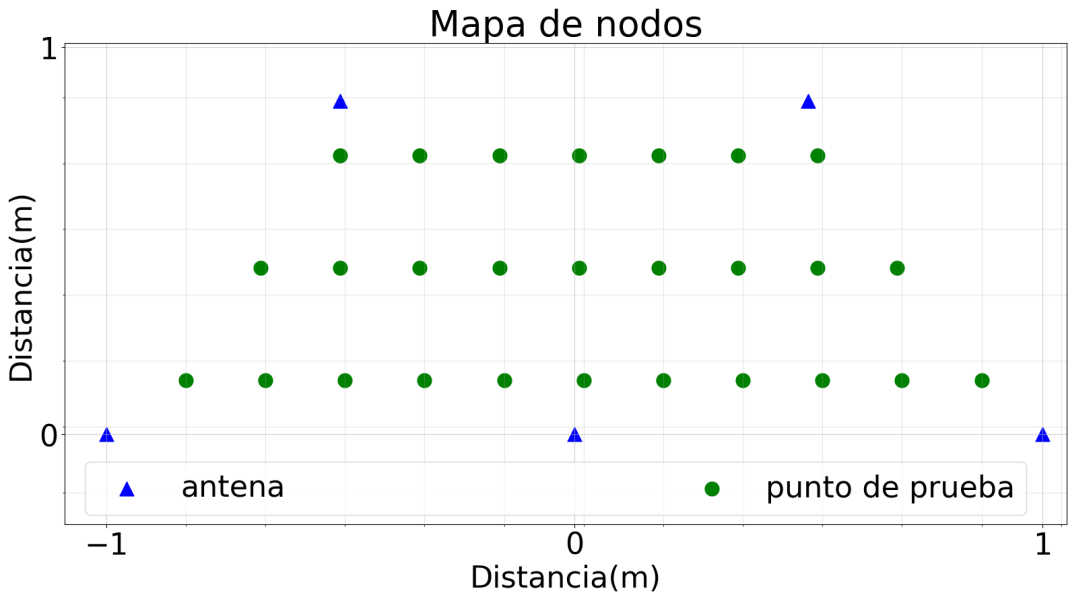



**Figura 3.9: Mapa con coordenadas de puntos de medición probados para la trilateración**

## 3.4. Detección de obstáculos

La función de detección de obstáculos se implementó en python 3.0 con OpenCV 3.4, siguiendo el proceso descrito en la metodología y la figura 2.2. A continuación se muestra el pseudocódigo del código desarrollado.

```
1. Capturar un frame.
2. Dividir la imagen en 2 imágenes, imagenR e imagenL.
3. Corregir la distorsión y rectificar las imágenes imagenR e imagenL
4. Calcular los puntos de interés con el detector de esquinas de Harris en la
   imagenL.
5. Calcular la disparidad entre imgenR  e imagenL usando SAD.
6. Crear 3 clusters de disparidad usando K–medias.
7. Calcular la profundidad para los clusters.
```

```
8. Mostrar los puntos más cercanos a la cámara.
9. Repetir desde 1 para el siguiente frame.
```

**Código. 3.1: Algoritmo Implementado.**

### 3.4.1. Filtrado y rectificación.

La eliminación de las distorsiones radial y tangencial de la imagen así como la rectificación se realizaron con la función **undistort()**, esta función recibe como parámetros de entrada una matriz con los valores intrínsecos de la cámara y la imagen original. Para hallar los valores intrínsecos de la cámara previamente se realizo una calibración usando una cuadricula tipo tablero de ajedrez.

### 3.4.2. Puntos de interés

Para hallar los puntos de interés se usó la función **cornerHarris()** la cual devuelve una imagen en escala de grises con las posibles esquinas. Para cada esquina devuelve una nube de puntos por lo que es necesario calcular el centroide de esta nube, esto se realiza con la función **connectedComponentsWithStats()**

### 3.4.3. Disparidad

La disparidad se calculó usando el método SAD. La función compara para una región de interés de la imagen imagenR todas las ventanas $w_i$ con una ventana $W$ tomada de la imagen imagenL y centrada en un punto de interés $p$

### 3.4.4. Clusters de profundidad

Con el fin de agrupar los objetos encontrados en 3 posibles distancias se usó la función **kmeans()**, la cual entre otros parámetros requiere el número de clusters o centroides y un

array de con los elmentos a clasificar.

# Capítulo 4

# Resultados

En este capítulo se presentan los resultados de la calibración de los nodos BLE, la trilateración de vehículo UGV y el algoritmo de detección de obstáculos.

## 4.1. RSSI

Con el fin de conseguir el modelo que mejor se verificó la generación de coeficientes $A$ y $n$ a partir de los datos de entradas puros (RAW, por su nombre en Inglés) como se ilustra en la Figura 4.1; la mediana de los datos con un filtro de ventana de tamaño 100, Figura 4.2; y los datos promediados con un filtro de ventana de tamaño 100, Figura 4.3.

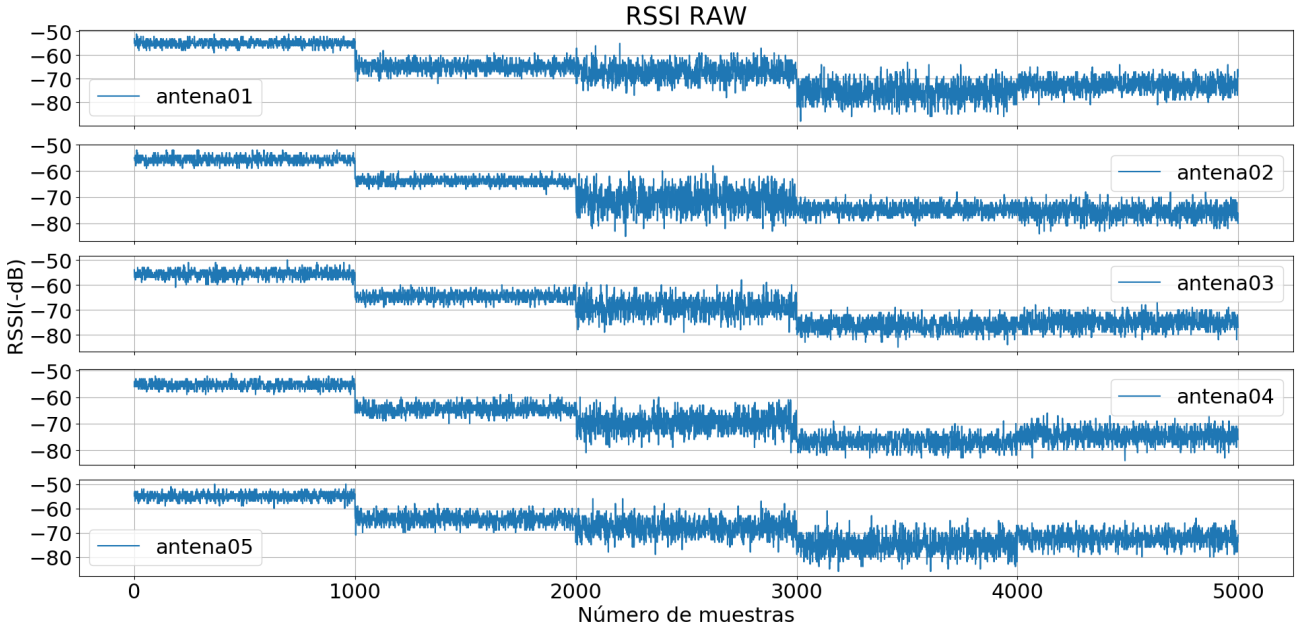



**Figura 4.1: Datos puros de RSSI para posiciones de 1 a 5 metros**

En la Figura 4.1 se aprecia que existe un cambio notable en el valor del RSSI para las diferentes distancias probadas, sobre todo en los primeros cuatro metros (muestas 0 a 4000)

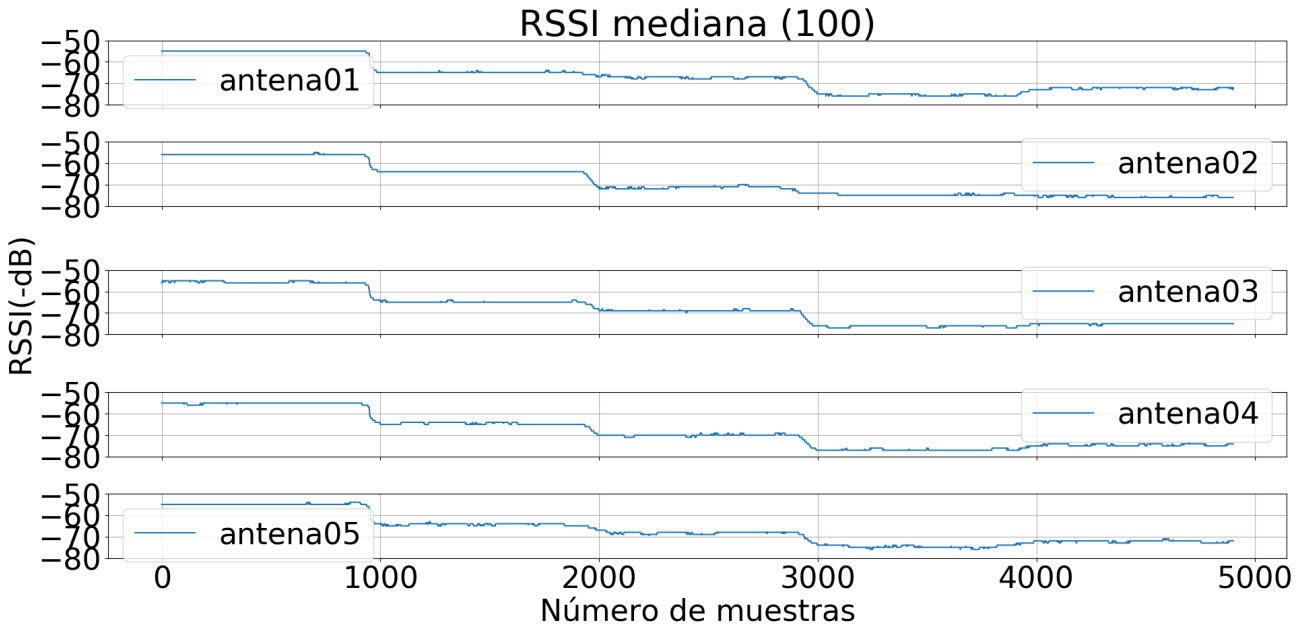



**Figura 4.2: Mediana de los datos con ventana de tamaño 100**

Al aplicar el filtro de mediana, ver Figura 4.2, los datos obtenidos presentan valores más estables para una misma distancia, lo que indica que presentan una baja desviación estándar.

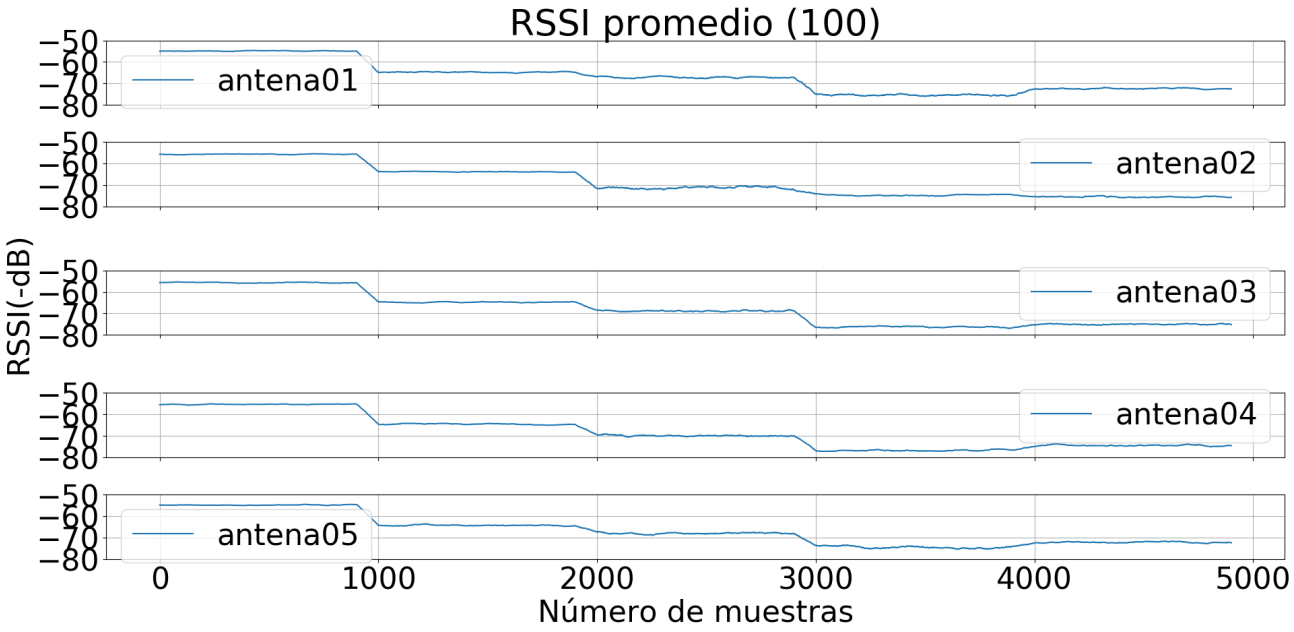



**Figura 4.3: Promedio de los datos con ventana de tamaño 100**

El comportamiento de los datos con el filtro de media,ver Figura 4.3, también muestran un comportamiento estable, pero con mayor variación que el filtro de mediana.

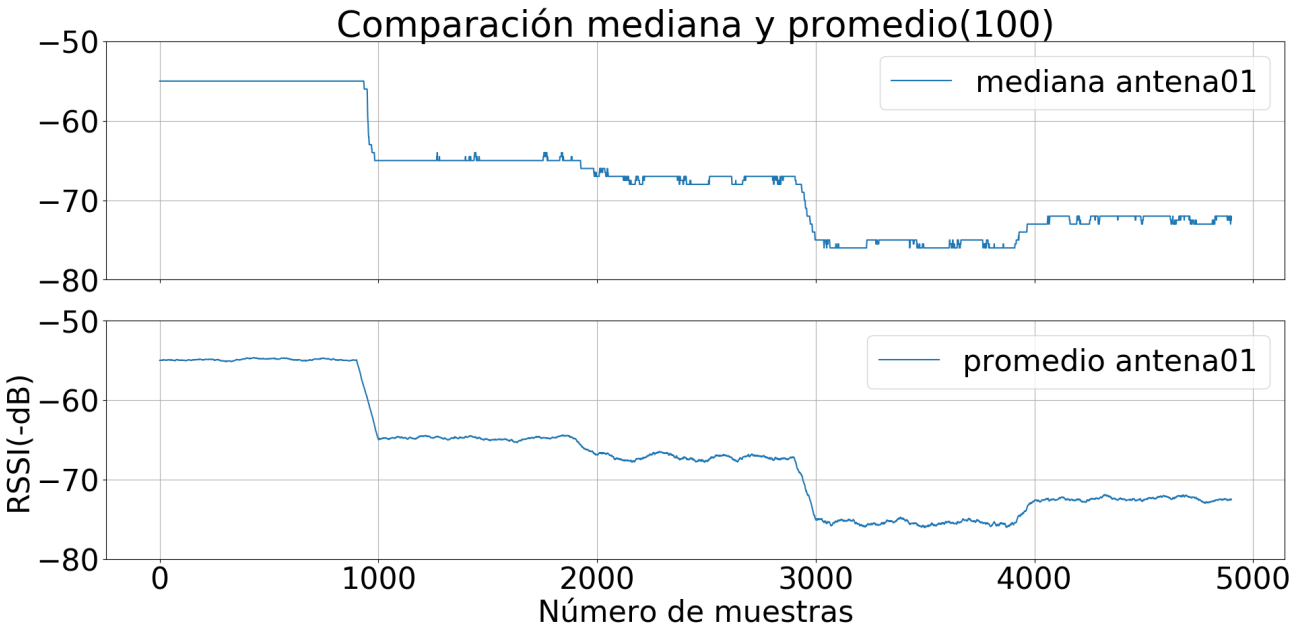



**Figura 4.4: Comparación de los filtros de mediana y media para la antena01**

La Figura 4.4 compara el comportamiento de las curvas creadas por el filtro de mediana y media para la antena01. Podemos ver que la mediana representa mejor el comportamiento estático que tenía la antenas cuando se realizaron las mediciones. Por este motivo se decidió usar este filtro para la obtención de la estimación de distancia.

Para confirmar que el modelo propuesto se apegué al comportamiento de las antenas lo primero que se realizó fue comprobar que el comportamiento del canal sea un modelo normal. La distribución de frecuencias de RSSI a distancias de 1 a 5 metros para la antena01 se muestra en las figuras: Figura 4.5 a Figura 4.9.

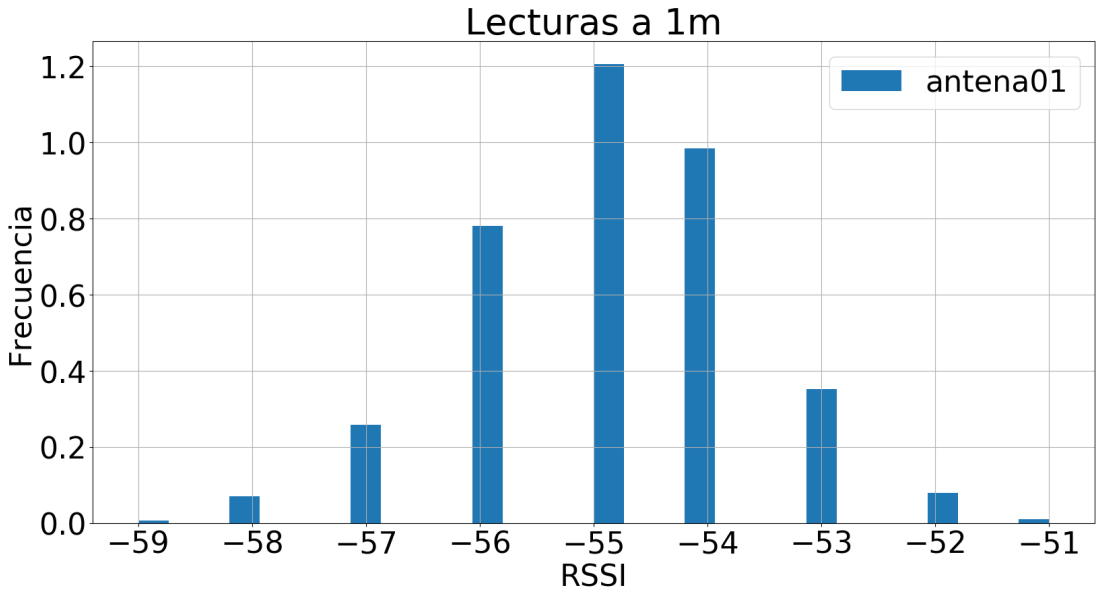



**Figura 4.5: Distribución de frecuencias de valores de RSSI para una medición punto a punto de 1 metro.**

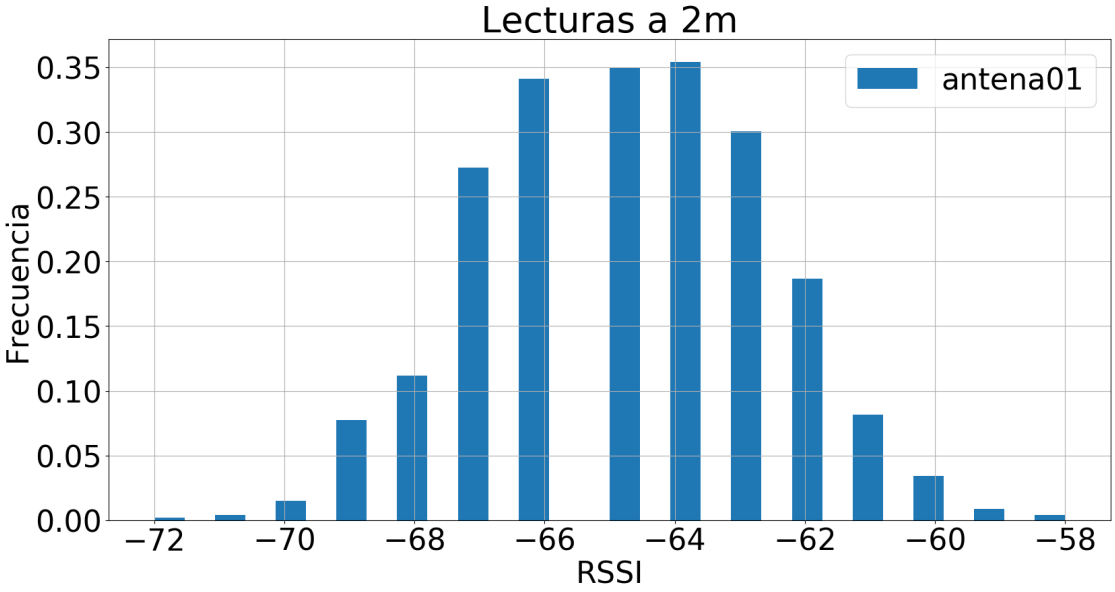



**Figura 4.6: Distribución de frecuencias de valores de RSSI para una medición punto a punto de 2 metros.**

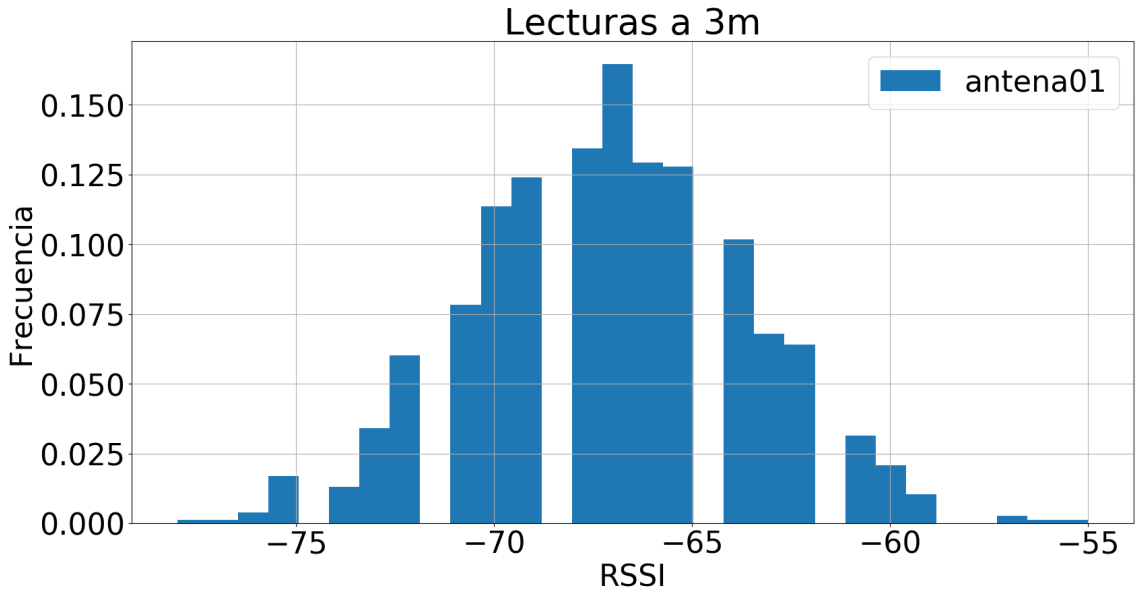



**Figura 4.7: Distribución de frecuencias de valores de RSSI para una medición punto a punto de 3 metros.**

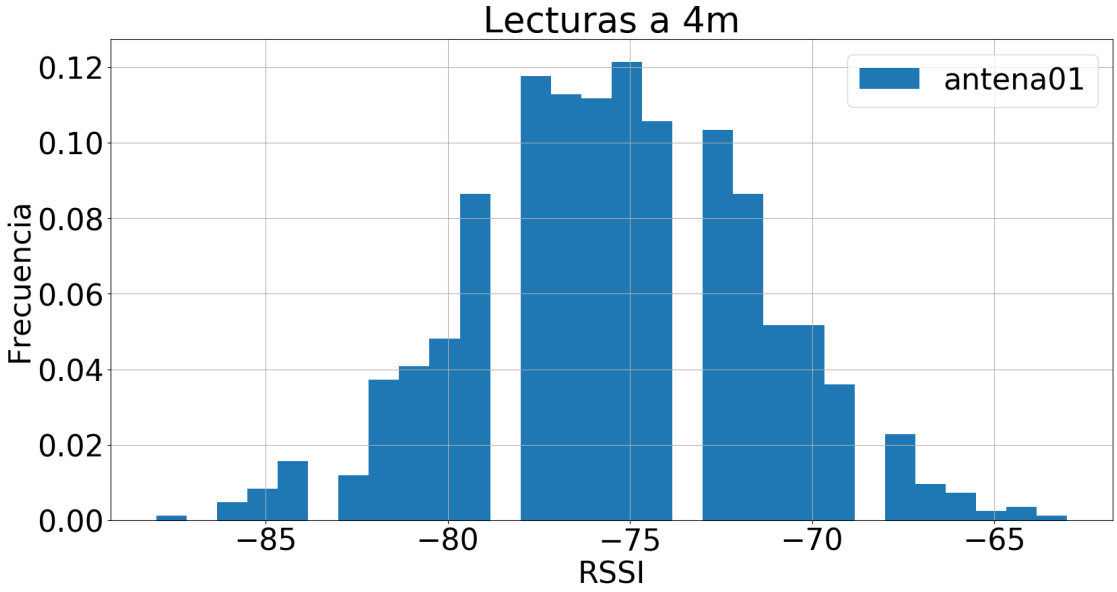



**Figura 4.8: Distribución de frecuencias de valores de RSSI para una medición punto a punto de 4 metros.**

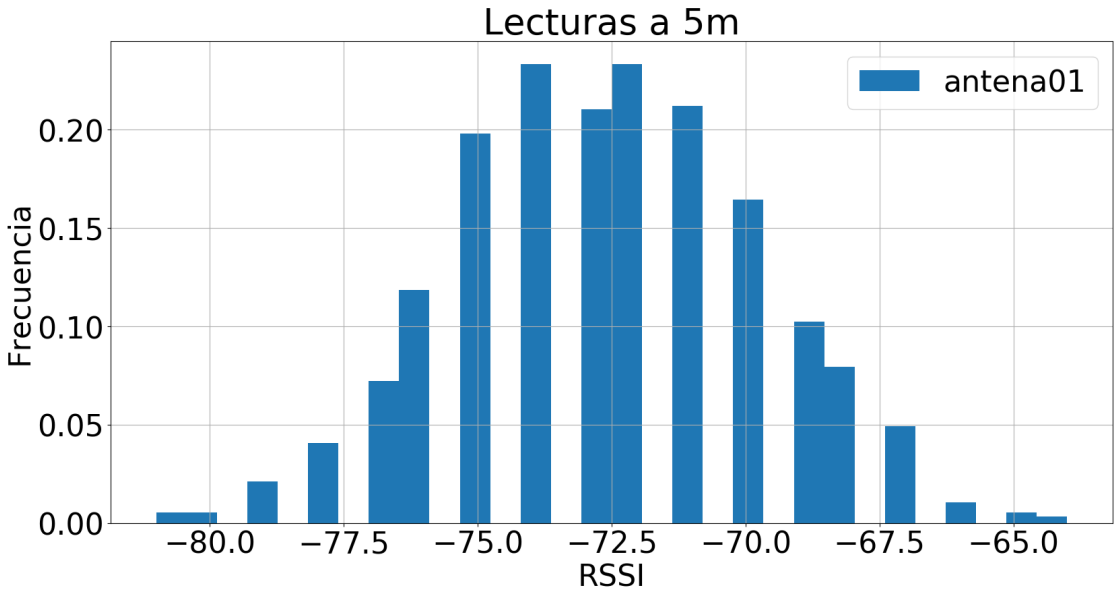



**Figura 4.9: Distribución de frecuencias de valores de RSSI para una medición punto a punto de 5 metros.**

A pesar de que la distribución no es igual para cada punto de medición en todos los casos el comportamiento es normal. En estas gráficas es más clara la relación que existe entre la distancia y la precisión de los datos, a menor distancia mayor precisión, la curva normal está claramente definida en la Figura 4.9.

Las parajes de valores $A$, $n$ para cada una de las antenas se muestra en la Tabla 4.1, al tratarse de un mismo modelo de BLE los valores de los coeficientes son muy parecidos lo que muestra la facilidad de replicación del proyecto.

**Tabla 4.1: Valores de A y n de las antenas**

| Antena | A | n |
|---|---|---|
| antena01 | -57.51 | 2.32 |
| antena02 | -58.26 | 2.61 |
| antena03 | -58.01 | 2.57 |
| antena04 | -57.62 | 2.62 |
| antena05 | -57.43 | 2.26 |

En la Figura 4.10 se presentan los modelos RSSI-Distancia obtenidos con los coeficientes de

la Tabla 4.1 para cada una de las antenas. En esta grafita se puede apreciar que para distancias menores a un metro, el comportamiento de las antenas es prácticamente el mismo.

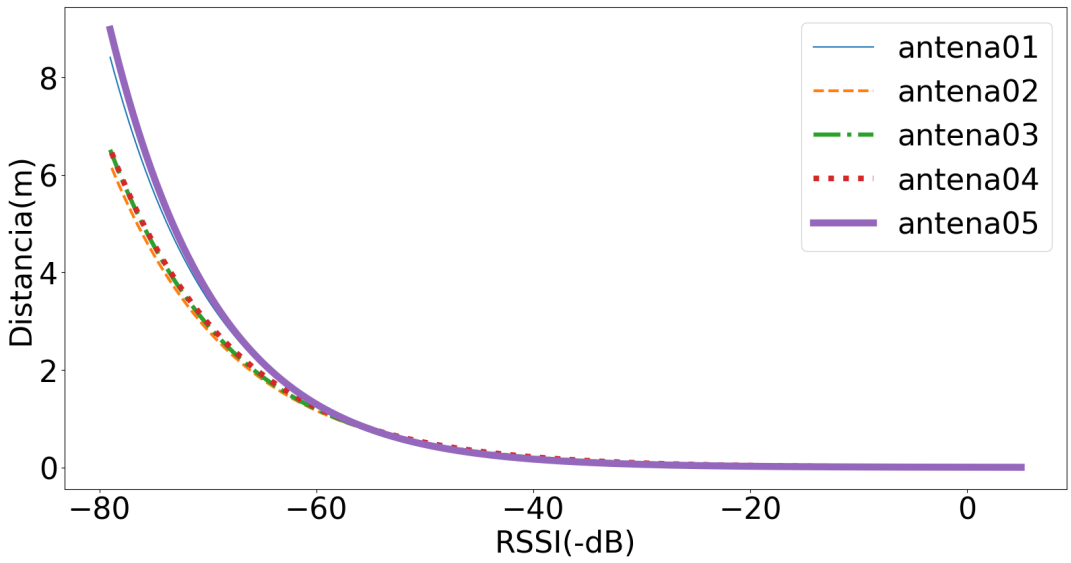



**Figura 4.10: Modelo RSSI-Distancia**

Para comparar las distancia estimada contra la real a las mediciones de RSSI filtradas se les aplicó el modelo descrito en la ecuación 2.5, los resultados de comparación se muestran en las figuras: Figura 4.11 a Figura4.15.

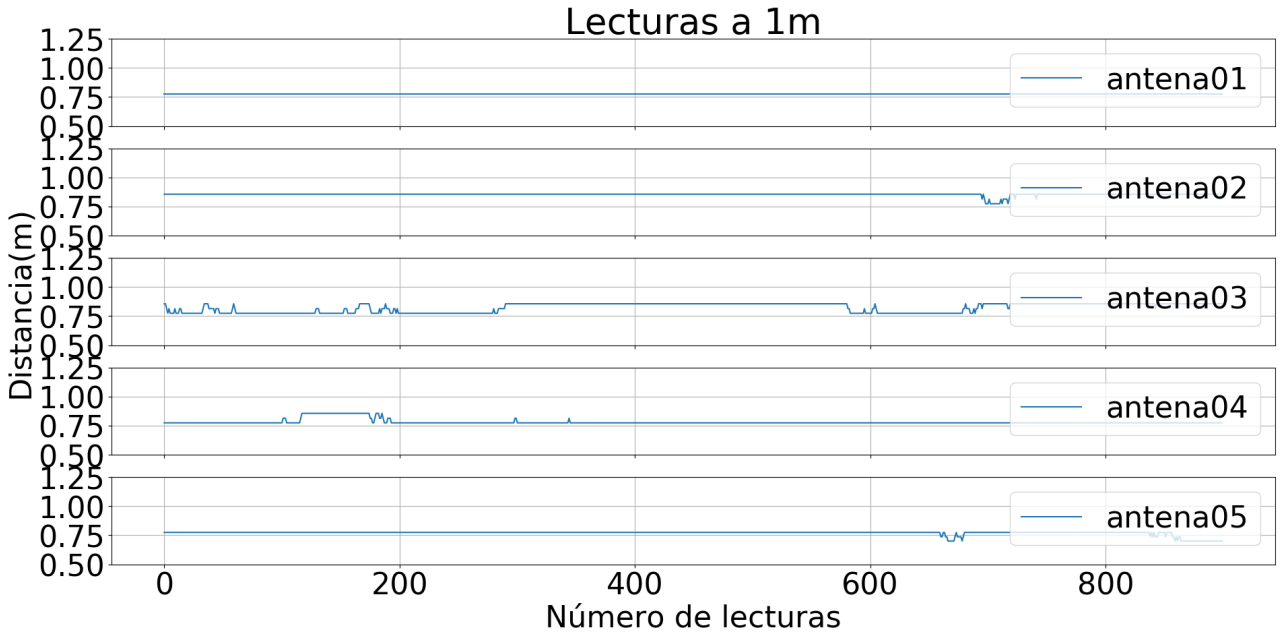



**Figura 4.11: Medición punto a punto de 1 metro.**

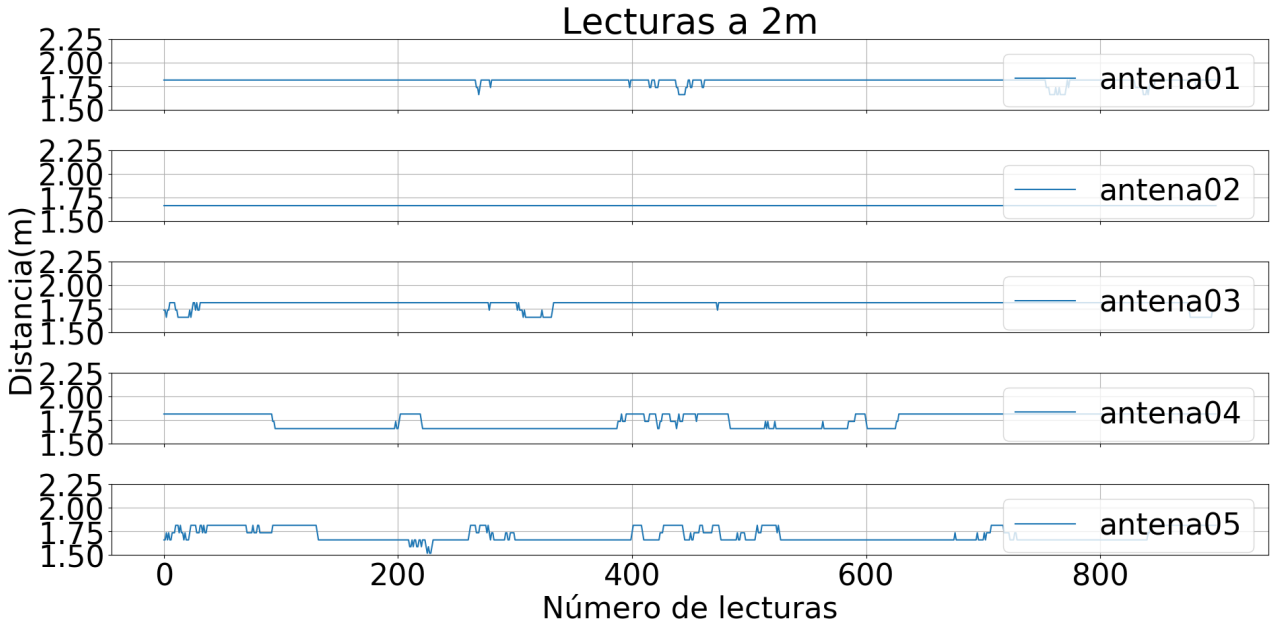



**Figura 4.12: Medición punto a punto de 2 metros.**

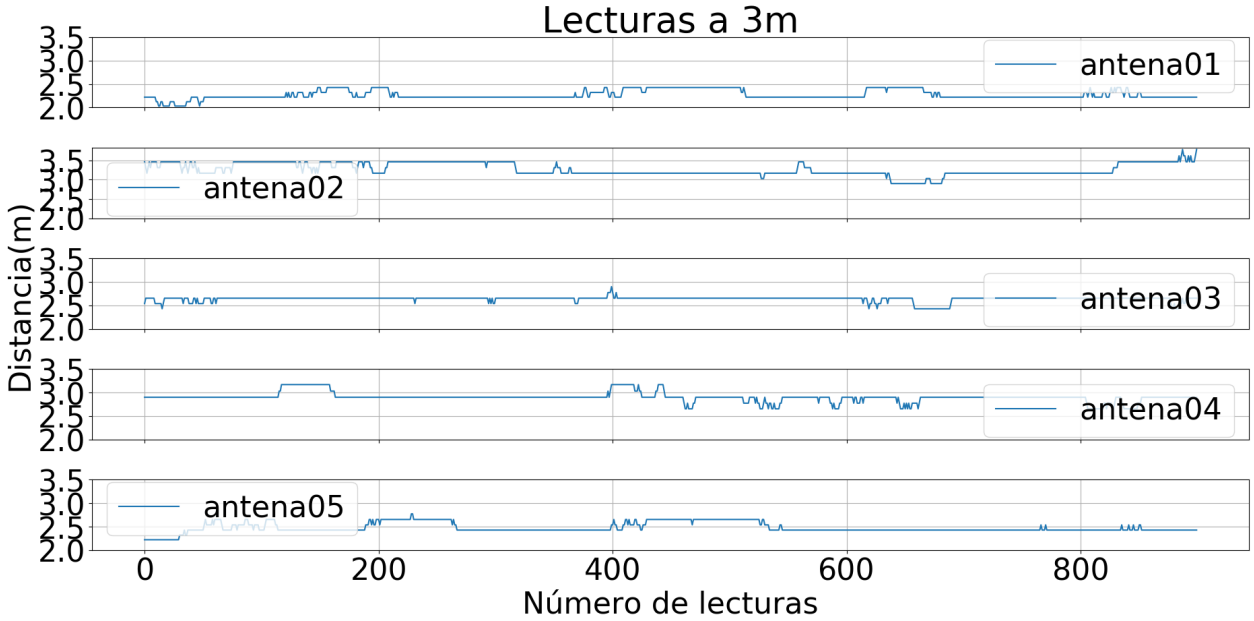



**Figura 4.13: Medición punto a punto de 3 metros.**

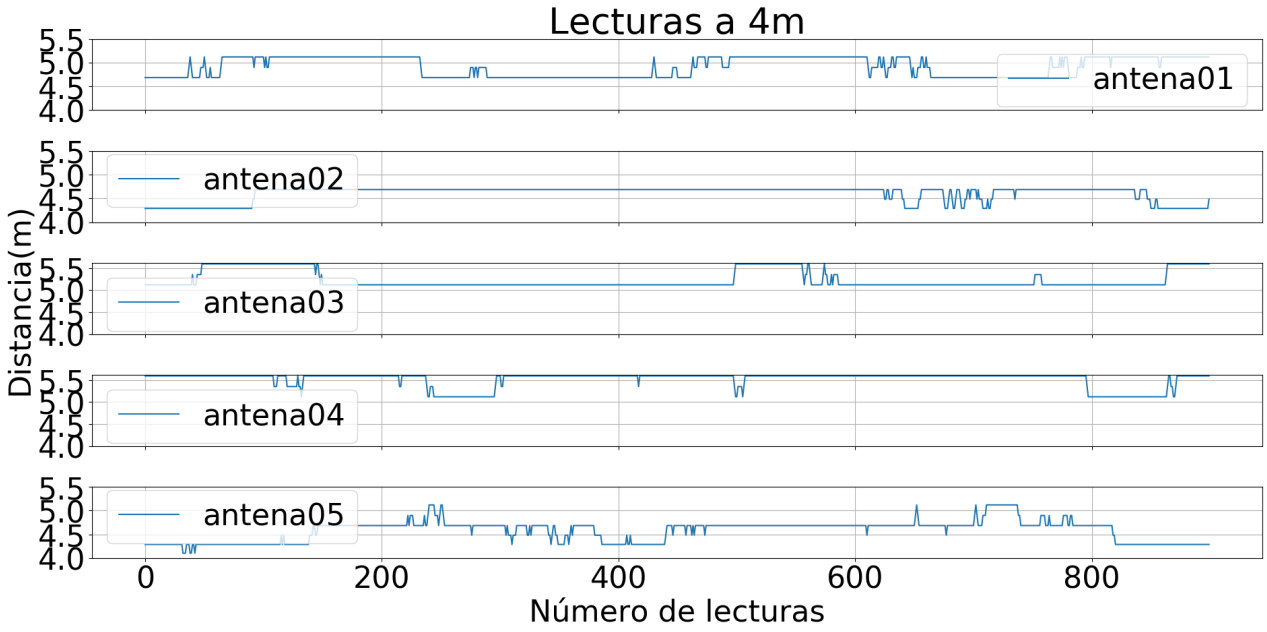



**Figura 4.14: Medición punto a punto de 4 metros.**

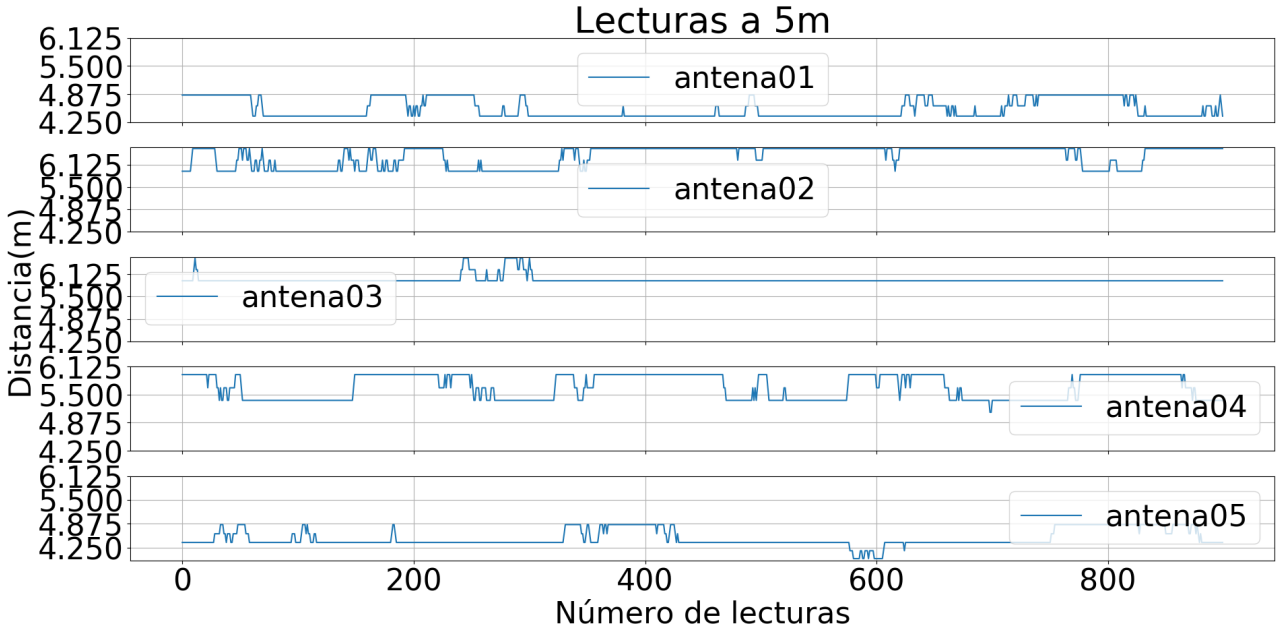



**Figura 4.15: Medición punto a punto de 5 metros.**

Los resultados no difieren mucho de los vistos con RSSI, es decir, la estimación de la distancia es más precisa para distancias menores a cuatro metros.

Por ultimo las figuras: Figura 4.16 a Figura4.20; muestran el error medio de cada antena para las distancias de 1 metro a 5 metros. En estas gráficas para cada valor de error medio también se muestra la desviación entandar, la cual esta representada con las rectas verticales con punta de flecha.

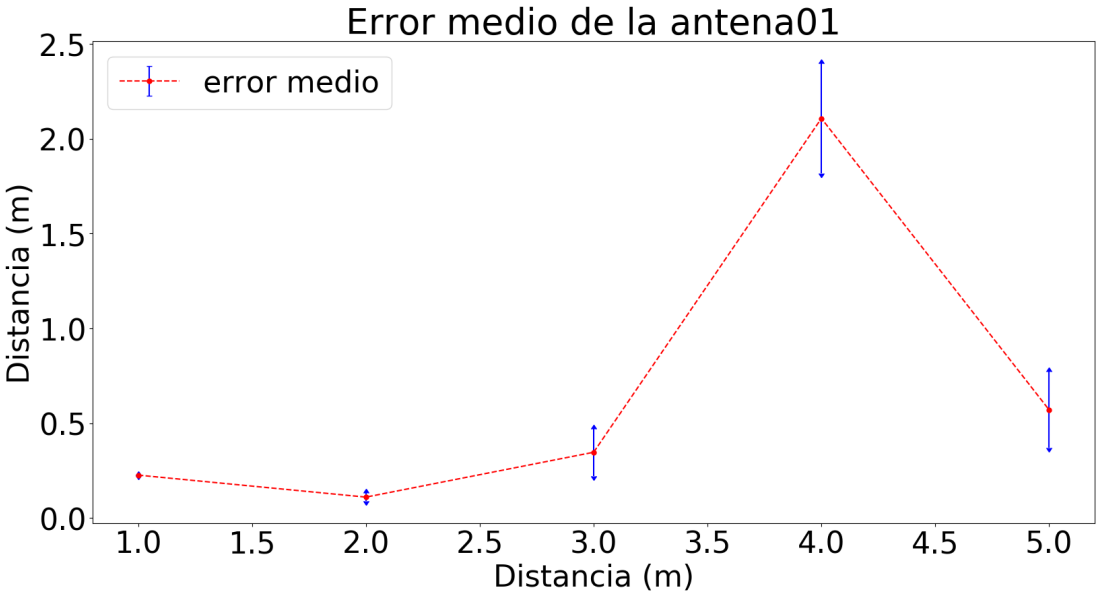



**Figura 4.16: Error medio de la antena01 a 1m,2m,3m,4m y 5m.**

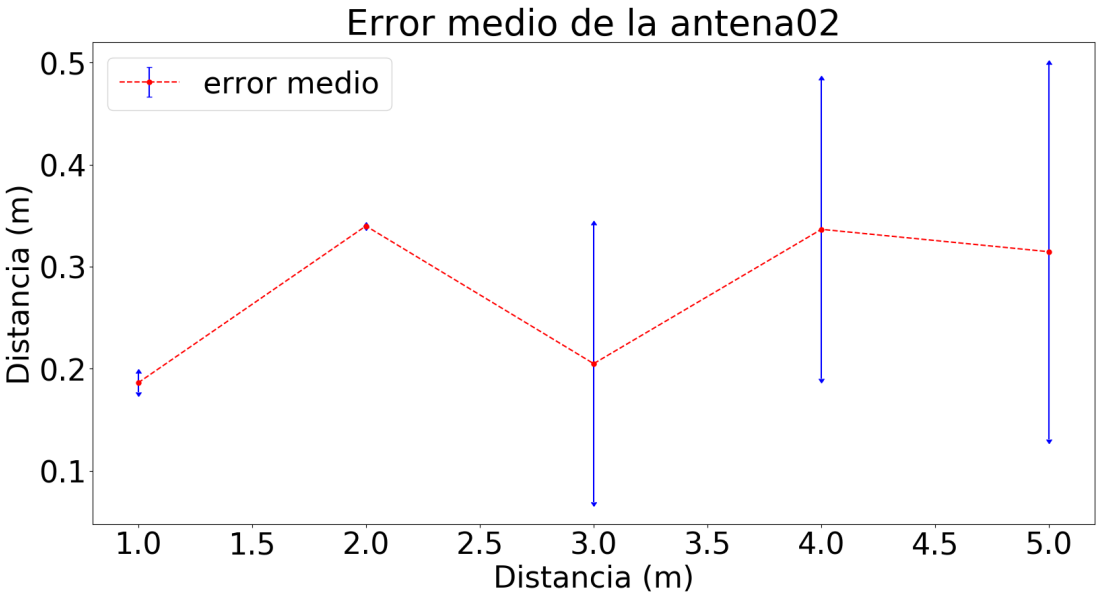



**Figura 4.17: Error medio de la antena02 a 1m,2m,3m,4m y 5m.**

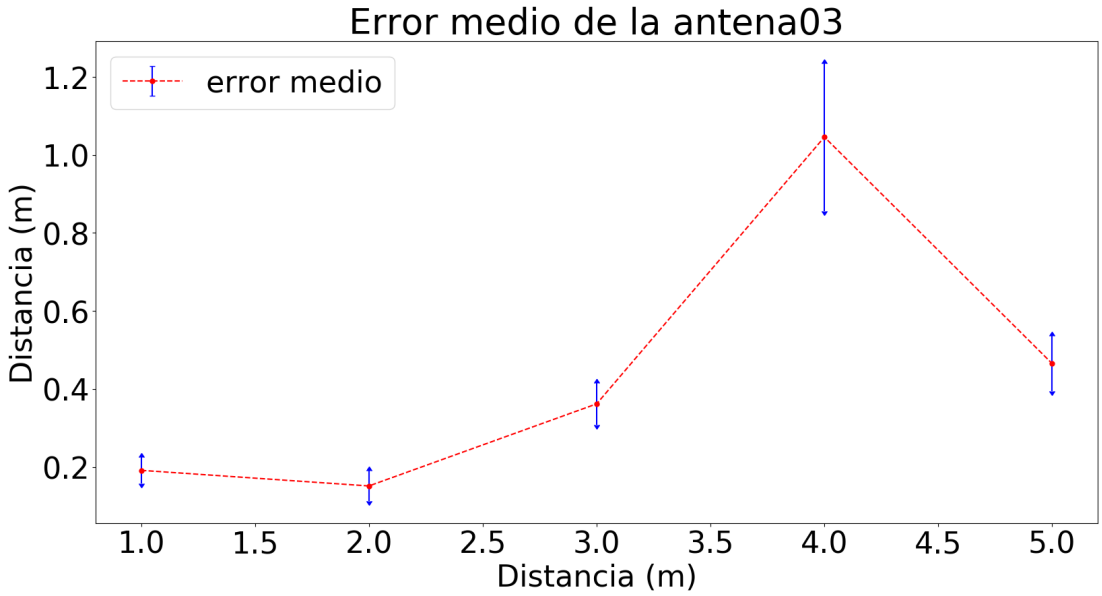



**Figura 4.18: Error medio de la antena03 a 1m,2m,3m,4m y 5m.**

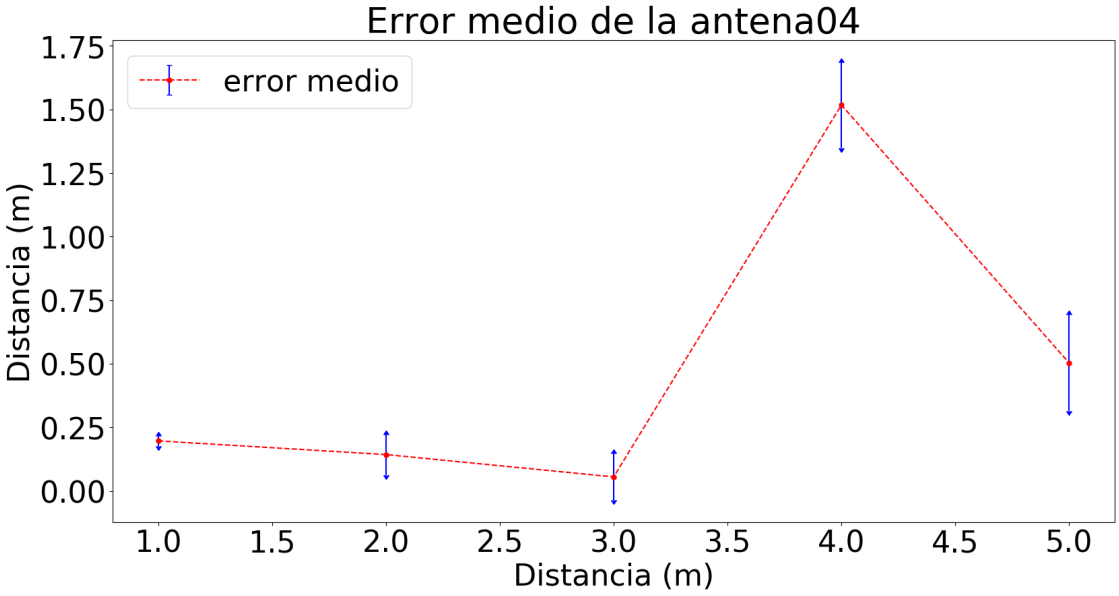



**Figura 4.19: Error medio de la antena04 a 1m,2m,3m,4m y 5m.**

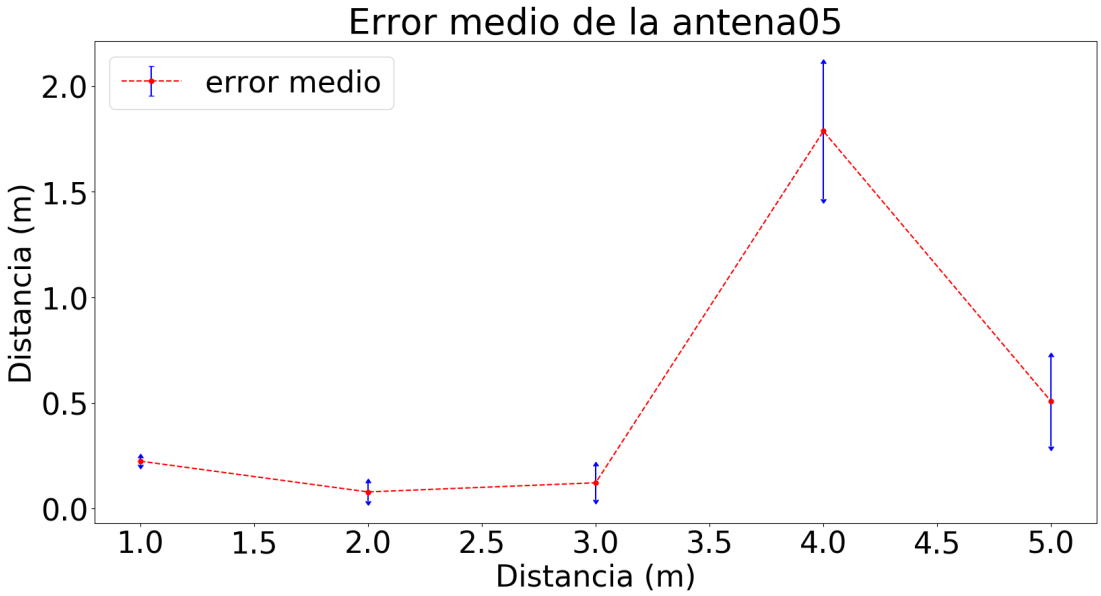



**Figura 4.20: Error medio de la antena05 a 1m,2m,3m,4m y 5m.**

Al igual que con las figuras anteriores, estas muestran un alto error al pasar los tres metros.

## 4.2. Trilateración

La Figura 4.21 muestra la distancia (la linea punteada) que existe entre los puntos reales, en donde se llevaron las mediciones (los puntos), y los puntos estimados con el modelo RSSI-Distancia (las estrellas); el error máximo obtenido fue de 0.25m. Se puede observar que cercano al origen del plano se tienen los menores errores, esto debido a que la antena situada en este punto es la que que cumple la menor distancia con respecto a cada una de las otras antenas.

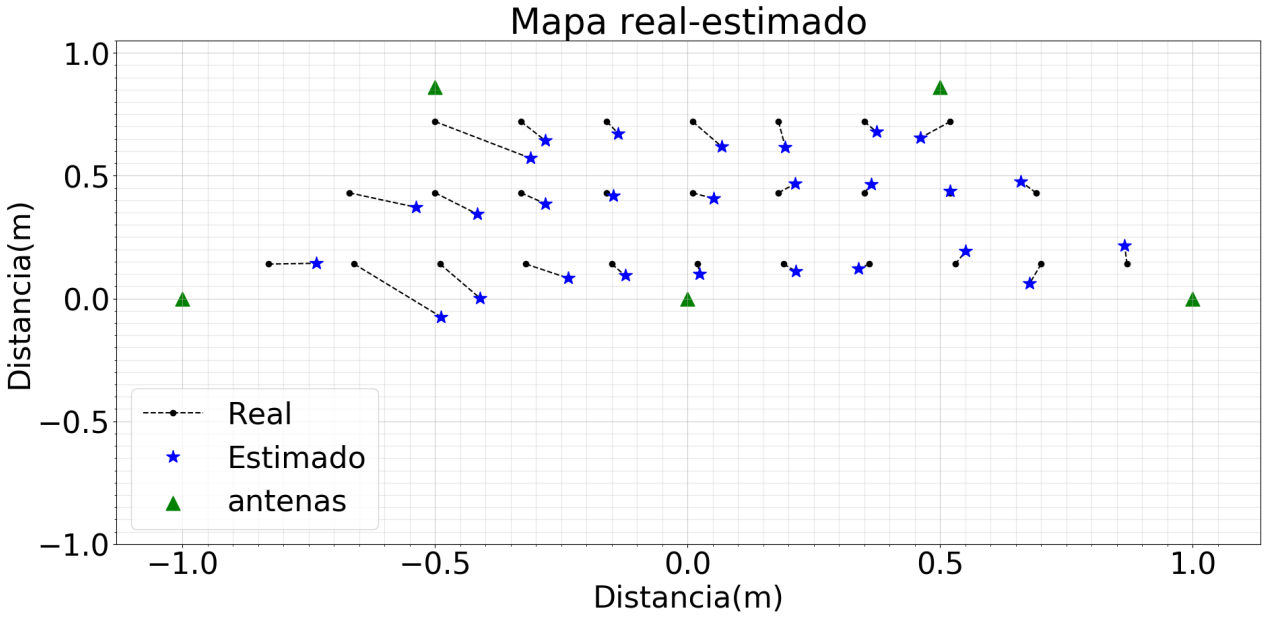



**Figura 4.21: Mapa con comparación de los puntos reales con los estimados**

Usando los errores en cada punto de medición se generó un mapa de error para la región de movimiento del robot, ver Figura 4.22, aunque si es notable que existe un bajo error en el centro del trapecio no se esperaba tener el mayor error cercano de los BLE. Se espera poder disminuir el error conseguido usando el filtro de Kalman.

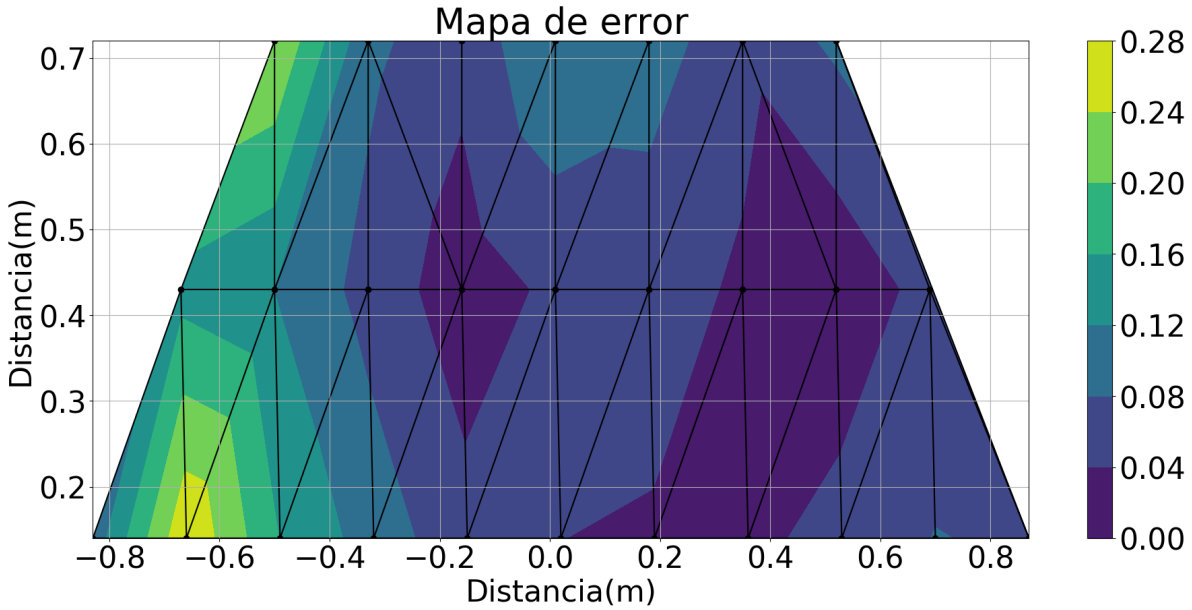



**Figura 4.22: Mapa de calor del error de la distancia por áreas.**

## 4.3. Detección de obstáculos

Para este desarrollo las imágenes fueron tomadas con la cámara estereoscópica ZED, el procesamiento de las imágenes se realizó con la plataforma Odroid Xu 4 la cual cuenta con un procesador Samsung Exynos5422, 2GB de memoria RAM y el sistema operativo Ubuntu 16.04 LTS.

Los resultados presentados están divididos en cada una de las etapas mostradas en el apartado 3.4. A continuación, se muestran los resultados para los pasos mas relevantes de cada etapa. Como muestra la Figura4.23, el primer paso a seguir es la adquisición de un frame.

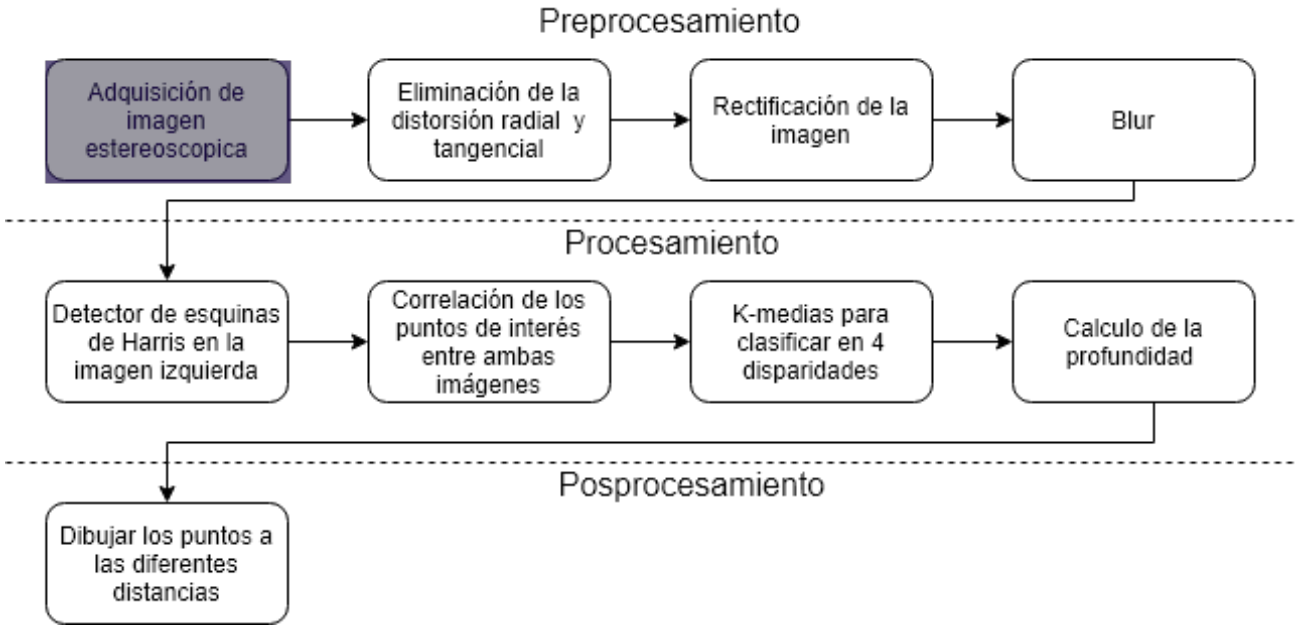



**Figura 4.23: Proceso actual en que se presentan y evalúan resultados.**

La Figura 4.24 representa como es la captura de este frame para el UGV, como es de esperarse la imagen esta compuesta por un par estero. En esta imagen es notaria la distorsión radial que existe para cada elemento de la misma.

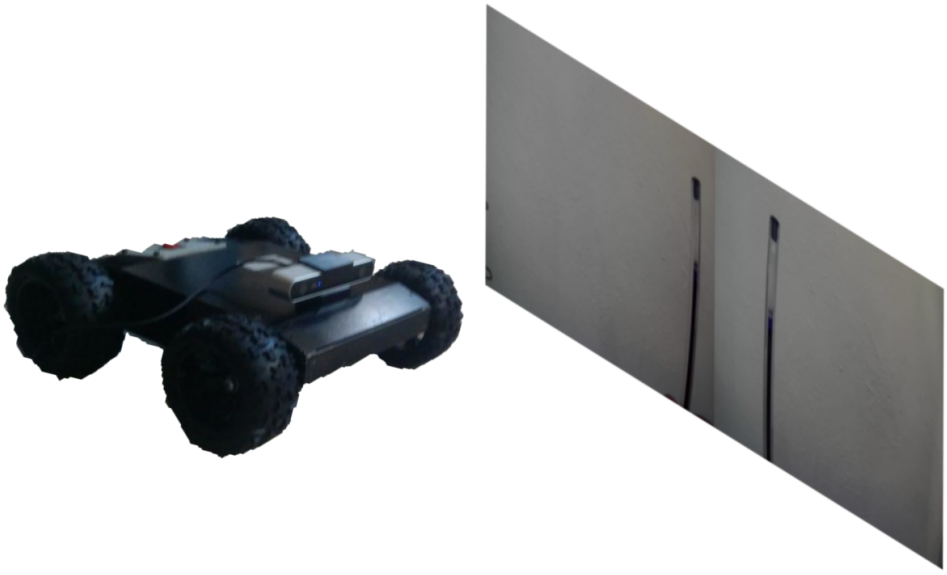


**Figura 4.24: UGV tomando una captura de lo que se encuentra en su camino.**

Siguiendo con los pasos del preprocesamiento, como se ven el a Figura 4.25 es turno de analizar los resultados de .Eliminación de la distorsión radial y tangencialz Rectificación de la imagen".

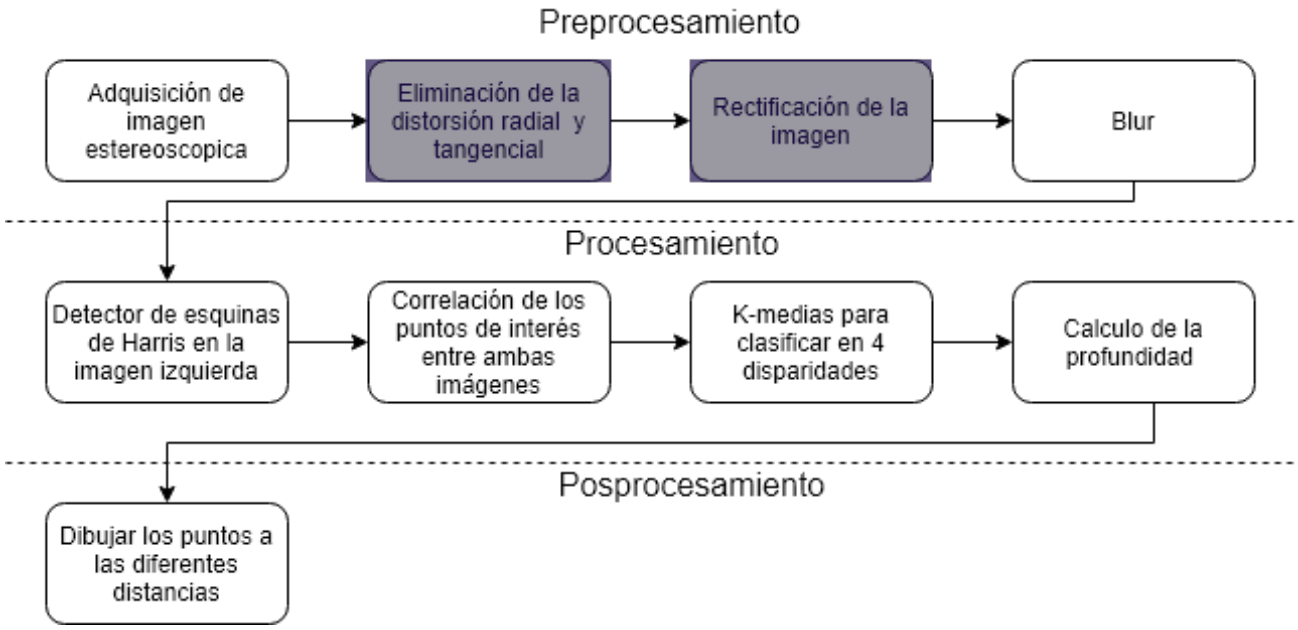



**Figura 4.25: Proceso actual en que se presentan y evalúan resultados.**

La función **undisort()** debe ser capaz de remover la distorsión presente en la imagen de entrada, la cual en este caso se trata de una deformación radial como se aprecia en la Figura 4.26

**Figura 4.26: Frame tomado con la cámara ZED.**

El resultado para este frame se muestra en la 4.27, es importante notar como la curvatura de los elementos verticales fue eliminada.

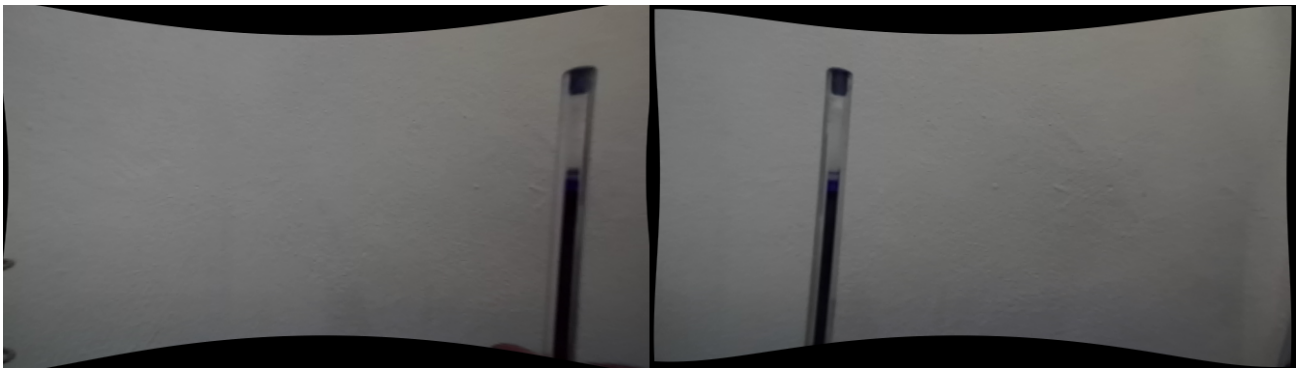


**Figura 4.27: Frame corregido para eliminar la distorsión.**

Siguiendo con el mapa de procesos el ultimo paso del preprosecamiento es el blur, el cual no requiere ver resultados, por lo que seguiremos con el procesamiento. LA etapa de procesamiento inicia con "Detector de esquinas de Harris en la imagen izquierda", ver Figura 4.28

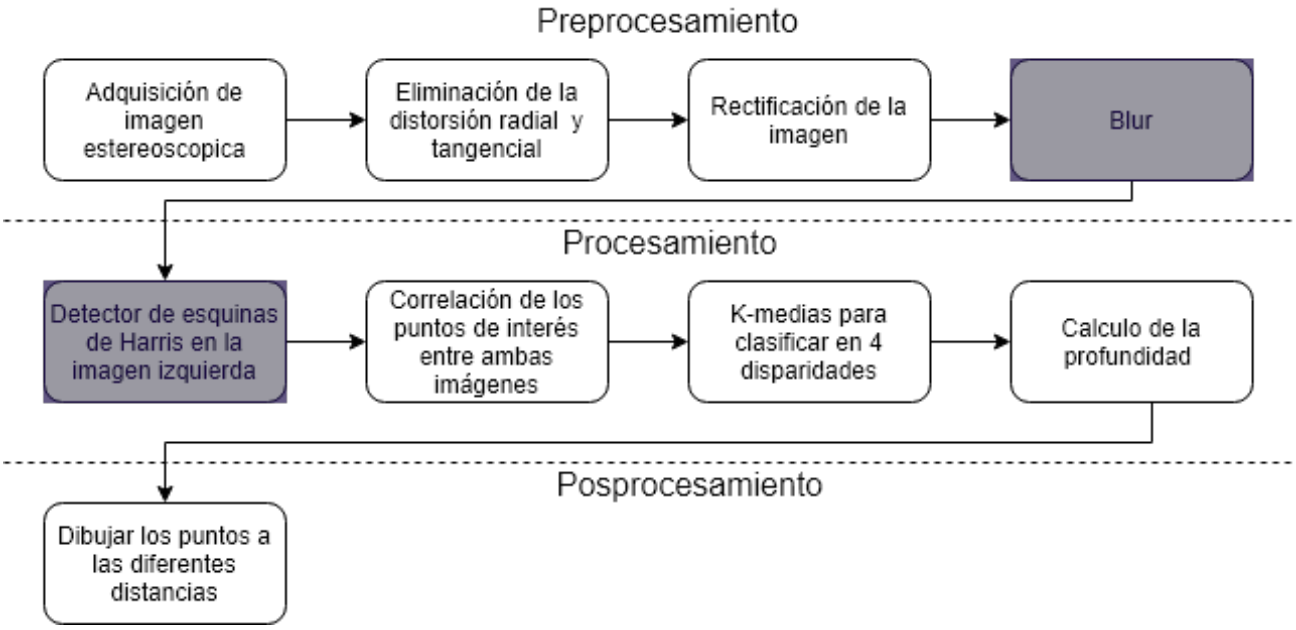



**Figura 4.28: Proceso actual en que se presentan y evalúan resultados.**

La Figura 4.29 muestra el resultado de aplicar la función **cornerHarris()** en la imagen izquierda. Esta función regresa una nube de puntos que corresponden a posibles esquinas en la imagen de entrada. Es esta nube de puntos la que se usará en el paso siguiente para calcular la disparidad.

**Figura 4.29: Resultado de aplicar el detector de esquinas de Harris en la imagen izquierda.**

Una vez que se tienen los puntos de interés, en este caso las esquinas obtenidas con **cornerHarris()**, se procede a calcular la disparidad y profundidad de estos puntos; La Figura 4.30 muestra los taras que se llevan para obtener esta información.

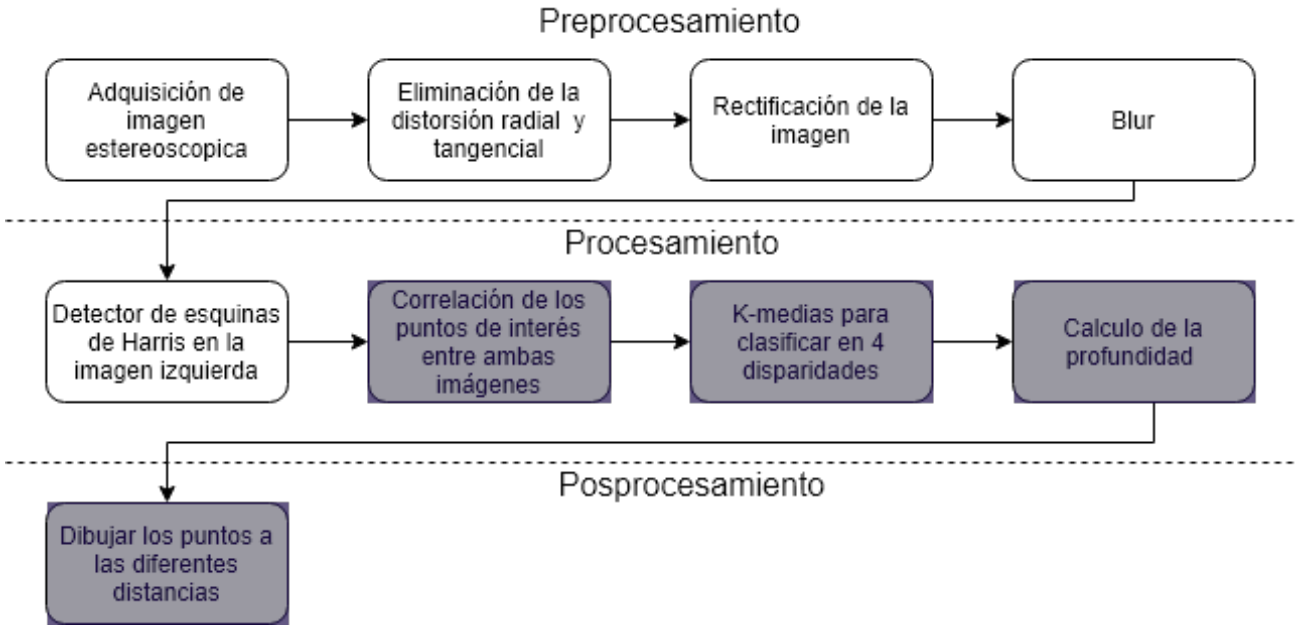



**Figura 4.30: Proceso actual en que se presentan y evalúan resultados.**

La Figura 4.31 muestra el resultado de aplicar el calculo de profundidad y la agrupación de distancias con k-means, en la imagen se encierran en un círculo rojo los puntos màs cercanos a la cámara. Es importante notar que no todos los puntos encontrados con el detector de esquinas de Harris, ver Figura 4.29, se encuentran como objetos cercanos en la Figura Figura 4.31.

**Figura 4.31: Objetos cercanos a la cámara.**

# Capítulo 5

# Conclusión

En este trabajo de tesis se modeló y estimó la posición de un UGV mediante el algoritmo de trilateración, el cual recibió como entradas las distancias estimadas con el RSSI proveniente de una red de antenas BLE.

También se realizó un análisis espacial del error en la trayectoria del vehículo el cual muestra la importancia de la geometría elegida para la elección de los nodos; ya que las zonas que reportaron el menor error coinciden con las esperadas por la geometría propuesta. Por otro lado se presentó un inconveniente en el tiempo de muestro, ya que como bien reportan en otras trabajos [12], este se ve directamente afectado por el tiempo de emparejamiento de los módulos BLE, por lo que es recomendable limitar la cantidad máxima de antenas que se usan para el algoritmo de trilateración. Por ultimo se observó que debido al comportamiento exponencial del RSSI el rango útil del sistema esta muy limitado en comparación de la distancia real máxima a la que se pueden comunicar dos dispositivos.

Se implementaron algoritmos de visión computacional para la detención de obstáculos. Este algoritmo encuentra los objetos cercanos mediante el uso de puntos de interés, en concreto las esquinas encontradas en la imagen, y aplica una clasificación según la cercanía de los objetos.

Para trabajos futuros se propone mejorar la precisión usando estimadores, como el Filtro de

Kalman, y agregar información proveniente de otros sensores por ejemplo, un IMU.

# Referencias


[1] J. Jung, D. Kang, J. Choi, and C. Bae. D2d distance measurement using kalman filter algorithm for distance-based service in an office environment. In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 221–224, July 2015.

[2] A. K. M. M. Hossain and W. Soh. A comprehensive study of bluetooth signal parameters for localization. In *2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–5, Sep. 2007.

[3] J. Rodas, T. M. Fernaandez, D. I. Iglesia, and C. J. Escudero. Multiple antennas bluetooth system for rssi stabilization. In *2007 4th International Symposium on Wireless Communication Systems*, pages 652–656, Oct 2007.

[4] Sheng Zhou and John K Pollard. Position measurement using bluetooth. *IEEE Transactions on Consumer Electronics*, 52(2):555–558, 2006.

[5] Yiran Peng, Wentao Fan, Xin Dong, and Xing Zhang. An iterative weighted knn (iw-knn) based indoor localization method in bluetooth low energy (ble) environment. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 794–800. IEEE, 2016.

[6] Zhu Jianyong, Haiyong Luo, Chen Zili, and Li Zhaohui. Rssi based bluetooth low energy indoor positioning. pages 526–533, 10 2014.

[7] Ling Pei, Ruizhi Chen, Jingbin Liu, Tomi Tenhunen, Heidi Kuusniemi, and Yuwei Chen. Inquiry-based bluetooth indoor positioning via rssi probability distributions. In *2010 Second International Conference on Advances in Satellite and Space Communications*, pages 151–156. IEEE, 2010.

[8] Yu Gu, Lianghu Quan, Fuji Ren, and Jie Li. Fast indoor localization of smart hand-held devices using bluetooth. In *2014 10th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 186–194. IEEE, 2014.

[9] Yixin Wang, Qiang Ye, Jie Cheng, and Lei Wang. Rssi-based bluetooth indoor localization. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 165–171. IEEE, 2015.

[10] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. Performance study of ieee 802.15.4 using measurements and simulations. In *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, volume 1, pages 487–492, April 2006.

[11] Jakub Neburka, Zdenek Tlamsa, Vlastimil Benes, Ladislav Polak, Ondrej Kaller, Libor Bolecek, Jiri Sebesta, and Tomas Kratochvil. Study of the performance of rssi based bluetooth smart indoor positioning. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 121–125. IEEE, 2016.

[12] David Contreras, Mario Castro Ponce, and David Sánchez de la Torre. Performance evaluation of bluetooth low energy in indoor positioning systems. *Transactions on Emerging Telecommunications Technologies*, 28, 08 2014.

[13] R. Al Alawi. Rssi based location estimation in wireless sensors networks. In *2011 17th IEEE International Conference on Networks*, pages 118–122, Dec 2011.

[14] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Information Theory*, 28:129–136, 1982.

[15] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[16] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[17] Stephan Wong, Stamatis Vassiliadis, and Sorin Cotofana. A sum of absolute differences implementation in fpga hardware. In *Proceedings. 28th Euromicro Conference*, pages 183–188. IEEE, 2002.

# Anexos

# Python

## .1. Modelo

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from myBle import ble
import myPlots as myplt

# Leer la inforación de entrada a partir de un archivo csv
d_raw=pd.read_csv("./calib.csv")

# Aplicando filtros a los datos de entrada
win_size=100
d_filter=pd.DataFrame( columns=d_raw.columns) # Filtro de promedio
d_filter2=pd.DataFrame( columns=d_raw.columns) # Filtro de medinana
for i in range(0,5000-win_size+1):
    x=[]
    y=[]
    for j,c_name in enumerate(d_raw.columns):
        x.append(d_raw[c_name][i:i+win_size].median())
        y.append(d_raw[c_name][i:i+win_size].mean())
    d_filter=d_filter.append(pd.DataFrame([x],columns=d_raw.columns))
    d_filter2=d_filter2.append(pd.DataFrame([y],columns=d_raw.columns))

# Mostrar información de entrada
from matplotlib.patches import Rectangle
from matplotlib.collections import PatchCollection

patches = []
fig, ax = plt.subplots()
for i in range(5):
    rectangle = Rectangle((i*1000,-90),1000,40,fill=False)
    patches.append(rectangle)
    ax.annotate(str(i+1)+" metros", (400+(i*1000),-88))
p = PatchCollection(patches, alpha=0.1, color="blue",)
ax.add_collection(p)
fig.canvas.set_window_title('filtro_raw')
plt.xlabel('Número de muestras')
plt.ylabel('RSSI(-dB)')
```

```
plt.title("RSSI RAW")
for name in d_raw.columns.values:
    if name!="dst":
        plt.plot(range(0,len(d_raw[name])),d_raw[name],label=name+" raw")
plt.legend()
plt.show()


# Preparando los datos para graficarlos
d0,d1,d2=[],[],[]
for i in range(1,6):
    ant="antena0{}".format(i)
    d0.append({"x":range(0,len(d_filter2[ant])),"y":d_filter2[ant],"label":ant+"
    promedio"})
    d1.append({"x":range(0,len(d_filter[ant])),"y":d_filter[ant],"label":ant+"
    mediana"})
    if i==1:
        d2.append({"x":range(0,len(d_filter2[ant])),"y":d_filter2[ant],"label":
    ant+" promedio"})
        d2.append({"x":range(0,len(d_filter[ant])),"y":d_filter[ant],"label":ant+
    " mediana"})

dataToShow=[
    {"figName":"filtro_mean","title":"RSSI promedio ("+str(win_size)+")","data":
    d0},
    {"figName":"filtro_median","title":"RSSI mediana ("+str(win_size)+")","data":
    d1},
    {"figName":"mean vs median","title":"Comparación mediana y promedio ("+str(
    win_size)+")","data":d2}
]

# Graficando
for data in dataToShow:
    myplt.inputData(data["figName"],data["title"],'Número de muestras','RSSI(-dB)
    ',data["data"])

# Mostrar los coeficientes A y n
model_ble,model_ble_mean,model_ble_median=[],[],[]
titles=["RAW","PROMEDIO","MEDIANA"]
for title in titles:
    print(title)
    for name in d_raw.columns.values:
        if name!="dst" and title=="RAW":
            model_ble.append(ble(name,d_raw[[name,'dst']],5))
            print("{} : {}, {}".format(name, model_ble[-1].A, model_ble[-1].n))
        if name!="dst" and title=="PROMEDIO":
            model_ble_mean.append(ble(name,d_filter2[[name,'dst']],5))
            print("{} : {}, {}".format(name, model_ble_mean[-1].A, model_ble_mean
    [-1].n))
        if name!="dst" and title=="MEDIANA":
            model_ble_median.append(ble(name,d_filter[[name,'dst']],5))
            print("{} : {}, {}".format(name, model_ble_median[-1].A,
```

```
    model_ble_median[-1].n))

# Creando graficas de modelo
x=np.linspace(5,-79,200)
dist_model=[]
plt.figure("modelo raw")
for antena_raw in model_ble:
    dist=np.array([antena_raw.getDist(xi) for xi in x])
    plt.plot(x,dist,label=antena_raw.label)


# Comparando los 3 modelos
dataToPlot=[
    {"x":x,"y":np.array([model_ble[0].getDist(xi) for xi in x]), "label":"
    antena01 raw"},
    {"x":x,"y":np.array([model_ble_mean[0].getDist(xi) for xi in x]), "label":"
    antena01 promedio"},
    {"x":x,"y":np.array([model_ble_median[0].getDist(xi) for xi in x]), "label":"
    antena01 mediana"}
]

myplt.inputData("comparación de modelos","Modelo raw & promedio & mediana",'
    Distancia(m)','RSSI(-dB)',dataToPlot)
```

**Código. 1: modelo.py**

## .2. Análisis espacial

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.patches import Circle
from matplotlib.collections import PatchCollection
from matplotlib.colors import ListedColormap
import matplotlib.tri as mtri
from matplotlib.mlab import griddata
from myBle import ble2 as ble
from myPlots import bleMap, bleMap2, gridMap

def getCircleList(data,device,p):
    circle_list=[]
    d=data[(np.isclose(data['x'],p[0]) & np.isclose(data['y'],p[1]))]
    for i in range(len(device)):
        x=d[device[i].label]
        x=np.median(x)
        dist = device[i].getDist(x)
        circle_list.append(dist)
    return circle_list


# Inicializamos los dispotivos
```

```
device=[
    ble(-57.21659695646536, 2.2802612330784644,"antena01",[-0.5,0.86]),
    ble(-58.26297356001721, 2.5467995985572505,"antena02",[0.5,0.86]),
    ble(-57.936110973030566, 2.5186283601844943,"antena03",[-1.0,0]),
    ble(-58.26297356001721, 2.5467995985572505,"antena04",[0,0]),
    ble(-57.936110973030566, 2.5186283601844943,"antena05",[1.0,0])
    ]


# Puntos propuestos
grid_points=[]
for i in range(7):
    grid_points.append([-0.5+i*0.17, 0.72])
for i in range(9):
    grid_points.append([-0.67+i*0.17, 0.43])
for i in range(11):
    grid_points.append([-0.83+i*0.17,0.14])

fig, ax = plt.subplots()
ax = bleMap(device,ax)
ax = gridMap(grid_points,ax)
fig.canvas.set_window_title('Mapa de nodos')

# Ajuste de los ejes de la rejilla
major_ticks = np.arange(-1, 1.1, 1)
minor_ticks = np.arange(-1, 1.1, .17)

ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)
ax.set_yticks(major_ticks)
ax.set_yticks(minor_ticks, minor=True)

# Rejilla en ambos ejes
ax.grid(which='both')

# Transparencia de la rejilla
ax.grid(which='minor', alpha=0.3)
ax.grid(which='major', alpha=0.6)

plt.xlabel('Distancia(m)')
plt.ylabel('Distancia(m)')
plt.title("Mapa de nodos")
ax.grid(True)
plt.show()


# Cargar datos
d_raw=pd.read_csv("./map.csv")

# Aplicando filtro de mediana
win_size=100
d_filter=pd.DataFrame( columns=d_raw.columns)
```

```
for i in range(0,len(d_raw.index) -win_size+1):
    x=[]
    for j,c_name in enumerate(d_raw.columns):
        x.append(d_raw[c_name][i:i+win_size].median())
    d_filter=d_filter.append(pd.DataFrame([x],columns=d_raw.columns))

# Valores de Salida
X,Y,X_real,Y_real,err_X,err_Y,err_dst =[],[],[],[],[],[],[]

# Trilateración
device_temp=device.copy()
for j in range(len(grid_points)):

    k=len(device_temp)-1
    d=getCircleList(d_filter,device_temp, grid_points[j])

    a=np.zeros((k,2))
    b=np.zeros((k,1))

    for i in range(k):
        a[i]=(2*(device_temp[k].p[0]-device_temp[i].p[0]),2*(device_temp[k].p[1]-
    device_temp[i].p[1]))
        b[i]=(d[i]**2)-(d[k]**2)-(device_temp[i].p[0]**2)-(device_temp[i].p
    [1]**2)+(device_temp[k].p[0]**2)+(device_temp[k].p[1]**2)

    # Resolvemos la ecuación lineal Ax=b
    x=np.linalg.lstsq(a,b,-1)[0]
    # Agregamos resultados a los array de salida
    err_X.append(x[0]-grid_points[j][0])
    err_Y.append(x[1]-grid_points[j][1])
    X_real.append(grid_points[j][0])
    Y_real.append(grid_points[j][1])
    err_dst.append(np.sqrt((x[0]-grid_points[j][0])*(x[0]-grid_points[j][0])+(x
    [1]-grid_points[j][1])*(x[1]-grid_points[j][1])))
    X.append(x[0])
    Y.append(x[1])

# Error en los puntos
err_points=[]
for i,err in enumerate(err_dst):
     err_points.append([grid_points[i][0],grid_points[i][1], err[0]])


# Mapa Real- Estimado
fig, ax = plt.subplots()
ax.set_xlabel("Distancia(m)")
ax.set_ylabel("Distancia(m)")
ax.set_title("Mapa real-estimado")

# Puntos realeas vs estimados
for i in range(len(X_real)):
    if i== len(X_real)-1:
```

```
            ax.plot([X_real[i],X[i]],[Y_real[i],Y[i]],'--ok',label="Real")
            ax.plot(X[i],Y[i],"o",color='b',label="Estimado")
        else:
            ax.plot([X_real[i],X[i]],[Y_real[i],Y[i]],'--ok')
            ax.plot(X[i],Y[i],"o",color='b')

# Dispositivos
for i,dev in enumerate(device):
    if i==len(device)-1:
        ax.scatter(dev.p[0],dev.p[1],marker="^",color='g',label="antenas")
    else:
        ax.scatter(dev.p[0],dev.p[1],marker="^",color='g')
    ax.annotate(dev.label, (dev.p[0],dev.p[1]))

# Major ticks every 20, minor ticks every 5
major_ticks = np.arange(-1, 1.1, .50)
minor_ticks = np.arange(-1, 1.1, .05)

ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)
ax.set_yticks(major_ticks)
ax.set_yticks(minor_ticks, minor=True)

# And a corresponding grid
ax.grid(which='both')

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.3)
ax.grid(which='major', alpha=0.6)

ax.grid(True)
ax.legend()
plt.show()


# Grafica Error
plt.figure("Trilateración Error X")
plt.xlabel('Posición')
plt.ylabel('Distancia(m)')
plt.title("Error X")
plt.plot(err_X,'--ok')

plt.figure("Trilateración Error Y")
plt.xlabel('Posición')
plt.ylabel('Distancia(m)')
plt.title("Error Y")
plt.plot(err_Y,'--ok')


# Grafica de calor
x,y,z = [],[],[]
for err in err_points:
```

```
    x.append(err[0])
    y.append(err[1])
    z.append(err[2])

triang = mtri.Triangulation(x, y)
fig, ax = plt.subplots()
im = ax.tricontourf(triang, z)
ax.triplot(triang, 'ko-')
fig.canvas.set_window_title('Mapa de error')
plt.xlabel('Distancia(m)')
plt.ylabel('Distancia(m)')
plt.title("Mapa de error")
ax.grid(True)

# Create colorbar
fig.colorbar(im, ax=ax)
plt.show()

from scipy.interpolate import griddata
points = np.zeros((len(err_points),2))
values  = np.zeros(len(err_points))
for i, err in enumerate(err_points):
    points[i]=(err[0],err[1])
    values[i]=err[2]

fig, ax = plt.subplots()
grid_x, grid_y = np.mgrid[-1:1:200j, 0:1:200j]
grid_z1 = griddata(points, values, (grid_x, grid_y), method='cubic')
im = ax.contourf(grid_x, grid_y, grid_z1,100)
ax = gridMap(grid_points,ax)
fig.colorbar(im, ax=ax)
fig.canvas.set_window_title('Mapa de error')
plt.xlabel('Distancia(m)')
plt.ylabel('Distancia(m)')
plt.title("Mapa de error")
plt.show()

# Grafica de Error en el plano
points = np.zeros((len(err_points),2))
values  = np.zeros(len(err_points))
for i, err in enumerate(err_points):
    points[i]=(err[0],err[1])
    values[i]=err[2]

fig, ax = plt.subplots()
grid_x, grid_y = np.mgrid[-1:1:200j, 0:1:200j]
grid_z1 = griddata(points, values, (grid_x, grid_y), method='cubic')
im = ax.contourf(grid_x, grid_y, grid_z1,100)
ax = gridMap(grid_points,ax)
fig.colorbar(im, ax=ax)
ax = bleMap2(device,ax)
fig.canvas.set_window_title('Mapa de error')
```

```
plt.xlabel('Distancia(m)')
plt.ylabel('Distancia(m)')
plt.title("Mapa de error")
plt.show()
```

**Código. 2: analisisEspacial.py**

## .3. myPlots

```
from matplotlib import pyplot as plt
from matplotlib.patches import Circle
from matplotlib.collections import PatchCollection

# Función de ayuda para graficar varias curvas en una misma imagen.
def inputData(figName,title,xlabel,ylabel,data):
    plt.figure(figName)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    for plot in data:
        plt.plot(plot["x"],plot["y"],label=plot["label"])
    plt.legend()
    plt.show()

# Define la función para dibujar el mapa espacial
def bleMap(ble_list, ax):
    patches = []
    for ble in ble_list:
        circle = Circle((ble.p[0],ble.p[1]),1)
        patches.append(circle)
    p = PatchCollection(patches, alpha=0.4)
    for ble in ble_list:
        ax.scatter(ble.p[0],ble.p[1],marker="o",color="b")
        ax.annotate(ble.label, (ble.p[0],ble.p[1]))
    return ax

def bleMap2(ble_list, ax):
    patches = []
    for ble in ble_list:
        circle = Circle((ble.p[0],ble.p[1]),1,linewidth=2.0)
        patches.append(circle)
    p = PatchCollection(patches, alpha=0.05,edgecolor='black', linewidth=2.0)
    ax.add_collection(p)
    for ble in ble_list:
        ax.scatter(ble.p[0],ble.p[1],marker="o",color="b")
        ax.annotate(ble.label, (ble.p[0],ble.p[1]))
    return ax

# Define la función para dibujar la nube de puntos
def gridMap(points,ax):
    for p in points:
```

```
        ax.scatter(p[0], p[1],marker="o",color='g')
    return ax
```

**Código. 3: myPlots.py**

## .4. myBle

```
import numpy as np

class ble:
    """
    Clase para crear los coeficientes de calibración de los dispositivos BLE
    y Calculas las distancias a partir del RSSI
    """
    def __init__(self,label,data,maxDist):
        self.label=label
        self.data=data
        self.maxDist=maxDist+1
        self.A=0
        self.n=0
        self.getParams()

    def getParams(self):
        N=np.shape(self.data)[0]
        XY=0
        XX=0
        XXY=0
        X=0
        Y=0
        x=10*np.log10(self.maxDist-(self.data['dst']/100))
        y=self.data[self.label]
        X=X+np.sum(x)
        Y=Y+np.sum(y)
        XY=XY+np.sum(x*y)
        XX=XX+np.sum(x*x)
        XXY=XXY+np.sum(x*x*y)

        den=N*XX-(X*X)

        self.n=(N*XY-X*Y)/den
        self.A=(N*XXY-X*XY)/den

    def getDist(self,rssi):
        d=pow(10,(rssi-self.A)/(-10*self.n))
        return d

class ble2:
    """
    Clase para calular las distancias dado el RSSI y los coeficinetes A y n
    """
    def __init__(self,A,n,label,p):
```

```
        self.A=A
        self.n=n
        self.label=label
        self.p=p

    def getDist(self,rssi):
        d=pow(10,(rssi-self.A)/(-10*self.n))
        return d
```

**Código. 4: myBle.py**

# PSOC4

## .5. Antena

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/

#if !defined(MAIN_H)

    #define MAIN_H

    #include <project.h>
    #include "app_BlE.h"
    #include "stdbool.h"


    /***************************************
    *   Conditional compilation parameters
    ***************************************/
//    #define     FLOW_CONTROL
    #define DEBUG_ENABLED           (1)
    #define     PRINT_MESSAGE_LOG               (1)
//    #define     LOW_POWER_MODE

    /***************************************
    *       Function Prototypes
    ***************************************/
    void AppCallBack(uint32 , void *);

#endif
```

```
/* [] END OF FILE */
```

**Código. 5: main.h**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include "main.h"

int main()
{
    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    CYBLE_API_RESULT_T      bleApiResult;

    CyGlobalIntEnable; /* Uncomment this line to enable global interrupts. */

    //Vincula el handler de eventos
    UART_Start();
    bleApiResult = CyBle_Start(AppCallBack);

    if(bleApiResult == CYBLE_ERROR_OK){
        //Mensaje que todo esta bien
        #ifdef PRINT_MESSAGE_LOG
            UART_UartPutString("\n\r
    ************************************************************");
            UART_UartPutString("\n\r***************** BLE UART example project
    *****************");
            UART_UartPutString("\n\r
    ************************************************************\n\r");
            UART_UartPutString("\n\rDevice role \t: CENTRAL");
        #endif

    }

    CyBle_ProcessEvents();
    for(;;)
    {
        HandleBleProcessing();
        CyBle_ProcessEvents();


    }
}
```

```
/* [] END OF FILE */
```

**Código. 6: main.c**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/

#if !defined(APP_BLE_H)
    #define APP_BLE_h

    #include <project.h>

    void HandleBleProcessing(void);
    void readRSSI(void);
#endif

/* [] END OF FILE */
```

**Código. 7: appBLE.h**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/

#include "app_BLE.h"
#include <stdio.h>
#include <main.h>

volatile static bool peerDeviceFound  = false;
volatile static bool endMission  = false;

static CYBLE_GAP_BD_ADDR_T       peerAddr;
CYBLE_GATTS_HANDLE_VALUE_NTF_T    SensorDataNTF;
uint8 myId=2;
```

```
void AppCallBack(uint32 event, void *eventParam)
{
    CYBLE_GAPC_ADV_REPORT_T                *advReport;

    switch (event)
    {
        case CYBLE_EVT_STACK_ON:
            break;
        case CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT:

            advReport = (CYBLE_GAPC_ADV_REPORT_T *) eventParam;

            if((advReport->eventType == CYBLE_GAPC_SCAN_RSP) && (true) \
                    && (advReport->data[1] == 0x01) && (advReport->data[2] == 0
    x06)  \
                    && (advReport->data[3] == 0x02) && (advReport->data[4] == 0
    x08) \
                    && (advReport->data[5] == 0x43))
            {
               //readRSSI();
              //Copy on Memory
              peerDeviceFound = true;
              memcpy(peerAddr.bdAddr, advReport->peerBdAddr, sizeof(peerAddr.
    bdAddr));
              peerAddr.type = advReport->peerAddrType;

            #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\r Device Found!!! \n\r ");
            #endif

            }

            break;

        case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:

            /* RESET all flags */
            peerDeviceFound  = false;
            endMission  = false;


            #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\r DISCONNECTED!!! \n\r ");
            #endif

            CyBle_GapcStartScan(CYBLE_SCANNING_FAST);
            CyBle_ProcessEvents();


            break;
```

```
        case CYBLE_EVT_GAP_DEVICE_CONNECTED:
            break;

        case CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP:

            if(CyBle_GetState() == CYBLE_STATE_DISCONNECTED)
            {

            #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\r ADVERTISEMENT START!!! \n\r ");
            #endif

            }
            else{
              #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\r ADVERTISEMENT STOP!!! \n\r ");
            #endif
            }
            break;
        default:
            break;
    }
}

void HandleBleProcessing(void)
{
    CYBLE_API_RESULT_T      cyble_api_result;

    switch (cyBle_state)
    {
        case CYBLE_STATE_SCANNING:
            if(peerDeviceFound)    CyBle_GapcStopScan();
            break;

        case CYBLE_STATE_CONNECTED:

            SensorDataNTF.attrHandle =
   CYBLE_APPEARANCE_HEART_RATE_SENSOR_HEART_RATE_BELT;
            SensorDataNTF.value.val = &myId;
          SensorDataNTF.value.len = sizeof(myId);

            cyble_api_result=CyBle_GattsNotification (cyBle_connHandle, &
   SensorDataNTF);

            if(CYBLE_ERROR_OK == cyble_api_result){
             #ifdef PRINT_MESSAGE_LOG
              UART_UartPutString("\n\r SENT NOTIFICATION!!! \n\r ");
             #endif

            }
            else if(CYBLE_ERROR_INVALID_PARAMETER == cyble_api_result){
             #ifdef PRINT_MESSAGE_LOG
```

```
            UART_UartPutString("\n\r connHandle value does not represent any
existing entry in the Stack !!! \n\r ");
            #endif

          }
          else if(CYBLE_ERROR_INVALID_OPERATION == cyble_api_result){
           #ifdef PRINT_MESSAGE_LOG
            UART_UartPutString("\n\r This operation is not permitted!!! \n\r ")
;
           #endif
          //Disconnect
          peerDeviceFound=false;
          CyBle_GapDisconnect(cyBle_connHandle.bdHandle);
          CyBle_ProcessEvents();


          }
          else if(CYBLE_ERROR_MEMORY_ALLOCATION_FAILED == cyble_api_result){
           #ifdef PRINT_MESSAGE_LOG
            UART_UartPutString("\n\r Memory allocation failed!!! \n\r ");
           #endif

          }



          break;

      case CYBLE_STATE_DISCONNECTED:
      {
          if(peerDeviceFound)
          {

              cyble_api_result = CyBle_GapcConnectDevice(&peerAddr);

        if(CYBLE_ERROR_OK == cyble_api_result)
        {

                #ifdef PRINT_MESSAGE_LOG
                    UART_UartPutString("\n\r CONNECT DEVICE!!! \n\r ");
                #endif

        }
          }
          else
          {
              CyBle_GapcStartScan(CYBLE_SCANNING_FAST);
              CyBle_ProcessEvents();

          }
          break;
      }
```

```
        default:
            break;
    }
}

void readRSSI(void){

    int8 rssi=CyBle_GetRssi();
    char strData[8];
//    char strAdrress[3];
//    for (int i = 0 ; i < 3 ; ++i) strAdrress[i]=peerAddr.bdAddr[i]+'0';//
   advReport->peerBdAddr[i];
    //sprintf(strAdrress,"%X",peerAddr.bdAddr);
    UART_UartPutString("\n\r");
//    UART_UartPutString(strAdrress);
//    UART_UartPutString(",");
    sprintf(strData,"%d",rssi);
    UART_UartPutString(strData);

}

/* [] END OF FILE */
```

**Código. 8: appBLE.c**

## .6. Explorador

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/

#if !defined(MAIN_H)

    #define MAIN_H

    #include <project.h>
    #include "app_BlE.h"
    #include "stdbool.h"


    /***************************************
    *    Conditional compilation parameters
```

```
    ***************************************/
//    #define      FLOW_CONTROL
    #define DEBUG_ENABLED            (1)
    #define      PRINT_MESSAGE_LOG                  (1)
//    #define      LOW_POWER_MODE

    /***************************************
    *       Function Prototypes
    ***************************************/
    void AppCallBack(uint32 , void *);

#endif


/* [] END OF FILE */
```

**Código. 9: main.h**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include "main.h"

int main()
{
    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    CYBLE_API_RESULT_T      bleApiResult;

    CyGlobalIntEnable; /* Uncomment this line to enable global interrupts. */

    //Vincula el handler de eventos
    UART_Start();
    bleApiResult = CyBle_Start(AppCallBack);

    if(bleApiResult == CYBLE_ERROR_OK){
        //Mensaje que todo esta bien
        #ifdef PRINT_MESSAGE_LOG
            UART_UartPutString("\n\r
   ************************************************************");
            UART_UartPutString("\n\r***************** BLE UART example project
   *****************");
            UART_UartPutString("\n\r
   ************************************************************\n\r");
            UART_UartPutString("\n\rDevice role \t: Peripherical");
```

```
        #endif

    }

    CyBle_ProcessEvents();
    for(;;)
    {
        HandleBleProcessing();
        CyBle_ProcessEvents();
    }
}

/* [] END OF FILE */
```

**Código. 10: main.c**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/

#if !defined(APP_BLE_H)
    #define APP_BLE_h

    #include <project.h>

    void HandleBleProcessing(void);
    void readRSSI(int8);
#endif

/* [] END OF FILE */
```

**Código. 11: appBLE.h**

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
```

```
#include "app_BLE.h"
#include <stdio.h>
#include <main.h>

volatile static bool peerDeviceFound  = false;
volatile static bool endMission  = false;

static CYBLE_GAP_BD_ADDR_T      peerAddr;
int8 deviceId=2;

void AppCallBack(uint32 event, void *eventParam)
{
    //CYBLE_GATT_ERR_CODE_T           errorCode;
    //CYBLE_GATTS_WRITE_REQ_PARAM_T   *writeReqParam;
    CYBLE_GAPC_ADV_REPORT_T               *advReport;
    CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T *UART_RX_data;

    switch (event)
    {
        case CYBLE_EVT_STACK_ON:
            break;

        case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:

            #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\r DISCONNECTED!!! \n\r ");
                //while(0 != (UART_SpiUartGetTxBufferSize() +
   UART_GET_TX_FIFO_SR_VALID));
            #endif

            /* RESET Uart and flush all buffers */
            UART_Stop();
            UART_SpiUartClearTxBuffer();
            UART_SpiUartClearRxBuffer();
            UART_Start();
            break;

        case CYBLE_EVT_GAPC_SCAN_PROGRESS_RESULT:

            advReport = (CYBLE_GAPC_ADV_REPORT_T *) eventParam;

            /* check if report has manfacturing data corresponding to the
   intended matching peer */
            if((advReport->eventType == CYBLE_GAPC_SCAN_RSP) && (advReport->
   dataLen == 0x06) \
                    && (advReport->data[1] == 0xff) && (advReport->data[2] == 0
   x31)  \
                    && (advReport->data[3] == 0x01) && (advReport->data[4] == 0
   x3b) \
                    && (advReport->data[5] == 0x04))
            {
```

```
                //Copy on Memory
                memcpy(peerAddr.bdAddr, advReport->peerBdAddr, sizeof(peerAddr.
    bdAddr));
                peerAddr.type = advReport->peerAddrType;

                //CyBle_GapcStopScan();
            }

            break;

        case CYBLE_EVT_GATT_CONNECT_IND:

            #ifdef PRINT_MESSAGE_LOG
                UART_UartPutString("\n\rConnection established");
            #endif

            break;

        case CYBLE_EVT_GATTC_HANDLE_VALUE_NTF:
            UART_RX_data = (CYBLE_GATTC_HANDLE_VALUE_NTF_PARAM_T *) eventParam;
            deviceId=*UART_RX_data->handleValPair.value.val;
            readRSSI(deviceId);
            break;

        default:
            break;
    }
}

void HandleBleProcessing(void)
{
        //uint8 txDataClientConfigDesc[2];

    switch (cyBle_state)
    {
        case CYBLE_STATE_ADVERTISING:
            break;

        case CYBLE_STATE_CONNECTED:



            break;

        case CYBLE_STATE_DISCONNECTED:



            CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);

            break;
```

```
        case CYBLE_STATE_INITIALIZING:
        case CYBLE_STATE_STOPPED:
        default:
            break;
    }
}


void readRSSI(int8 id){

    int8 rssi=CyBle_GetRssi();
    char strData[8];
    char strAdrress[8];
    //for (int i = 0 ; i < 3 ; ++i) strAdrress[i]=peerAddr.bdAddr[i]+'0';//
   advReport->peerBdAddr[i];
    sprintf(strAdrress,"%d",id);
    UART_UartPutString("\n\r");
    UART_UartPutString(strAdrress);
    UART_UartPutString(",");
    sprintf(strData,"%d",rssi);
    UART_UartPutString(strData);

}

/* [] END OF FILE */
```

**Código. 12: appBLE.c**