



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Aplicación de una red de sensores inalámbricos
distribuidos de bajo
coste para estimar el posicionamiento de objetos en 3 dimensiones
mediante el uso de señales de Ultrasonido*

Grado en Ingeniería Mecánica

ALUMNO: Gil García, Ignacio
DIRECTORES: González Prieto, José Antonio
González Coma, José P.
CURSO ACADÉMICO: 2020-2021

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Aplicación de una red de sensores inalámbricos
distribuidos de bajo
coste para estimar el posicionamiento de objetos en 3 dimensiones
mediante el uso de señales de radio Ultrasonido*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General / Infantería de Marina

Universida_{de}Vigo

RESUMEN

El desarrollo de sistemas de posicionamiento y la digitalización de experimentos constituyen una de las herramientas clave que permite el seguimiento en tiempo real, y además dota de la capacidad de realizar análisis de los datos que hayan sido almacenados. En particular, el posicionamiento de alta precisión mediante una red de sensores de ultrasonido permite evaluar, por ejemplo, el comportamiento de sistemas de navegación y posicionamiento inercial en maquetas de barcos a escala o el estudio de las deformaciones (en ciertos ensayos de materiales) realizados, ambos, en entornos de laboratorio mediante la aplicación un sistema de bajo coste. Con el objetivo de validar la posibilidad de esta propuesta, se planteó la posibilidad de crear un prototipo operativo basado en módulos de ultrasonido que, mediante técnicas de cálculo de tiempo de vuelo y posicionamiento en 3 dimensiones, permitiese calcular la distancia entre un emisor y un sensor receptor

De esta forma el objetivo principal del TFG es la creación de una maqueta que permita realizar el análisis de las diferentes opciones, que permiten calcular el posicionamiento relativo de una serie de sensores distribuidos de bajo coste, y emplear esta información con múltiples receptores para estimar la posición y orientación en 3D de un objeto con respecto a unas bases emisoras de pulsos de ultrasonidos

Es importante destacar que el proceso de cálculo de tiempo de vuelo requiere del uso de sincronización temporal, la cual, en este TFG, se ha realizado empleando señales de radiofrecuencia, que se encargan de informar del disparo de los pulsos de ultrasonido, empleando la diferencia de llegada entre ambas señales para poder calcular la distancia a la que se sitúa el emisor del pulso. Con el fin de poder transmitir la información procedente de los sensores, se realizó una emisión de estos en tiempo real por vía Wi-Fi empleando un servidor con soporte para el protocolo MQTT para la comunicación multipunto a punto., de forma que un programa en Python se encarga de recibir esta información y almacenarla para su procesamiento posterior.

El primer paso del procesamiento de la información pasa por la reducción de los niveles de ruido presente, el cual se caracteriza por un ruido de baja frecuencia típico de los sensores, y la introducción de *outliers* en determinados instantes donde el receptor no llega a captar el pulso emitido. A continuación, se procede a generar un modelo de regresión que permita traducir el tiempo de vuelo calculado con respecto a la distancia real obtenida, para lo cual se realiza una serie de pruebas de toma de datos de calibración. Finalmente, se aplica un filtro de Kalman para obtener una estimación del ángulo de *roll* de un objeto apartide la información generada por dos receptores instalados sobre un objeto, y se comparan los resultados con los obtenidos mediante sensores inerciales (IMU).

PALABRAS CLAVE

Radiofrecuencia, Ultrasonidos, Posicionamiento3D, Sensores distribuidos

AGRADECIMIENTOS

En primer lugar, quería mostrar mi agradecimiento a toda mi familia, en especial a mis abuelos, no solo por el apoyo mostrado durante esta recta final dentro de mi estancia en la Escuela Naval Militar, sino por todo el apoyo recibido durante todos estos años en la escuela. Donde han intentado siempre animarme diariamente, consiguiendo que tratase de ser siempre mejor persona y militar en mis quehaceres diarios.

Agradecer también a mi novia Mayte por todo su apoyo diario, presente a mi lado desde casi el principio de mi singladura en esta escuela. Apoyo sin el cual me hubiese costado mantenerme siempre animoso y con ganas de continuar aventura. Gracia por su eterna paciencia y por convertirse en mi mayor apoyo desde que la conocí.

Por último, lugar, agradecer enormemente a mis dos tutores Don José Antonio González Prieto, y Don José P. González Coma ayudándome siempre que lo he precisado para la realización de este trabajo. Sobre todo, agradecerles haberme tratado como un compañero más dentro del trabajo, instruyéndome como avanzar y en qué aspectos centrarme para completar con éxito este proyecto. introduciéndome en el uso de Hardware tipo Arduino con el que poder realizar todos los sensores y dispositivos necesarios para la realización de los experimentos.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	7
1.1 Introducción	7
1.2 Objetivos	7
1.3 Estructura del TFG	8
2 Estado del arte	9
2.1 Introducción	9
2.2 Clasificación topológica.....	9
2.2.1 Sistemas Tokenless	9
2.2.2 Sistemas Token.....	9
2.2.3 Sistemas Distribuidos	9
2.3 Clasificación en base a la tecnología	10
2.3.1 Señales de onda electromagnética	10
2.3.2 Señales sónicas	12
2.4 Técnicas de medición de distancia	14
2.4.1 RSSI (Received Signal Strength Indicator) Amplitud de la señal recibida	14
2.4.2 ToF (Time of Flight) Tiempo de vuelo.....	15
2.4.3 AoA (Angle of Arrival) Angulo de llegada	15
2.5 Técnicas de posicionamiento	16
2.5.1 Trilateración en 2D	16
2.5.2 Trilateración en 3D	18
2.6 Técnicas de filtrado y estimación.....	20
2.6.1 Estimación Bayesiana.....	20
3 Desarrollo del TFG.....	25
3.1 Diseño del prototipo.....	25
3.1.1 Selección de Hardware	25
3.1.2 Selección de Software.....	32
3.2 Creación del proyecto	36
3.2.1 Creación de bases emisoras y receptoras	36
3.2.2 Desarrollo del Software	42

4 Resultados / Validación / Prueba.....	45
4.1 Calibración	45
4.2 Estimación de la posición del punto en 3D.....	50
4.3 Estimación de posicionamiento de un objeto en 3D	52
4.3.1 Test de calibración.....	53
4.3.2 Test de movimiento vertical	55
4.3.3 Test de roll	58
4.3.4 Test general.....	61
5 Conclusiones y líneas futuras	64
5.1 Conclusiones y propuestas de mejora	64
5.2 Líneas futuras.....	64
6 Bibliografía.....	66
Anexo I: Código Arduino bases emisoras	67
Anexo II: Código RX Arduino con ESP8266	69
Anexo III: Código PYTHON CAPTURA DE DATOS	72
Anexo IV: Código PYTHON ANALISIS Y FILTRADO DE DATOS	74
Anexo V: Código PYTHON posicionamiento 3d.....	84
Anexo VI: Código PYTHON regresión tiempo de vuelo a distancia	85
7 Anexo VII: Código PYTHON filtro gh.....	86

ÍNDICE DE FIGURAS

Figura 2-1 Tipos de sistema de localización [1].....	10
Figura 2-2 centro de Girona mediante técnicas LIDAR [9]	11
Figura 2-3 Comparación de tecnologías de posicionamiento por RF, añadir fuente	12
Figura 2-4 Sonar LWHP53, Indra	13
Figura 2-5 curva de atenuación de antenas inalámbricas [6].	14
Figura 2-6 Técnica de medición de distancia por ToF [2].	15
Figura 2-7 Técnica de medición de distancia por TDoA	¡Error! Marcador no definido.
Figura 2-8 Diagrama medición varianza de dos sensores [3].....	20
Figura 2-9 Estructura de un algoritmo recursivo de un filtro GH [3]	21
Figura 2-10 Comparación de la variación de H y la dx [3]	22
Figura 2-11 Comparativa de un UKF con la trayectoria real de un objeto estudiado [3]	23
Figura 2-12 Evolución de un filtro de partículas [3]	24
Figura 3-1 Esquema general del sistema de posicionamiento.....	25
Figura 3-2 Sensor HC-SR04, Insertar referencia AZ_DELIVERY	25
Figura 3-3 Brush ultrasonido.....	26
Figura 3-4 US-026.....	26
Figura 3-5	27
Figura 3-6 Chip NRF24L01, Insertar referencia AZ_DELIVERY .	¡Error! Marcador no definido.
Figura 3-7 esquema de pines del NRF24L01 [manual].....	28
Figura 3-8 Pinout de un Arduino NANO, fuente Arduino	29
Figura 3-9 esquema de pines del ESP32[página AZ-DELIVERY]	32
Figura 3-10 Logotipo Arduino [ARDUINO]	33
Figura 3-11 Entorno de programación Arduino	33
Figura 3-12 Logotipo MQTT [4].....	34
Figura 3-13 Estructura de mqtt [4].	35
Figura 3-14 Esquema TX mediante Fritzing.	37
Figura 3-15 Prototipo Base TX con US-026	38
Figura 3-16 Prototipo Base TX con HC-SR04.....	38
Figura 3-17 Esquema RX con ESP32 mediante Fritzing.	39
Figura 3-18 Esquema RX con ESP8266 mediante Fritzing	40
Figura 3-19, Sensor RX montaje en Protoboard	40
Figura 3-20 Estación de pruebas	41

Figura 3-21 Material empleado41

Figura 3-22 Estación de pruebas con dos RX42

Figura 3-23 Esquema de estación meteorológica mediante Fritzing.....42

Figura A1-0-1 Logotipo del Centro Universitario de la Defensa (si la figura está tomada de alguna fuente se indicará la misma)..... **¡Error! Marcador no definido.**

Figura A1-0-2 Logotipo del Centro Universitario de la Defensa (si la figura está tomada de alguna fuente se indicará la misma)..... **¡Error! Marcador no definido.**

ÍNDICE DE TABLAS

Tabla 3-1 Ejemplo de tabla (tomada de [1]).....	¡Error! Marcador no definido.
Tabla 4-1 Ejemplo de tabla (tomada de [1]).....	¡Error! Marcador no definido.

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

La necesidad de determinar con precisión y exactitud la posición de objetos en entornos tridimensionales con distintas topologías es uno de los campos de investigación fundamentales en la actualidad. Tanto es así que podemos encontrar dicha necesidad en aplicaciones de última generación tales como los sistemas de conducción autónoma, sistemas robóticos inteligentes en tareas de carácter industrial, o en localización de personas u objetos en el interior de edificios. Con el fin de dar respuesta a este tipo de problemas han surgido en los últimos años diferentes tipos de tecnologías que en gran medida dependen del entorno de trabajo, como puede ser GPS, IMU, Sistemas LIDAR, Sonar, etc. Debe tenerse en cuenta que, en la actualidad, las tecnologías comentadas suelen ser empleadas tanto de forma aislada como de forma colaborativa, lo que constituye los denominados sistemas de localización híbridos.

Sin embargo, en la literatura disponible se puede observar que uno de los campos menos estudiados dentro de este tipo de tecnologías es la localización de objetos tridimensionales en entornos de laboratorio. Algunos ejemplos de topología dentro de este tipo de entornos son los experimentos de dinámica de objetos flotantes, análisis de deformaciones plásticas en condiciones prefijadas, o dinámica de sistemas robóticos colaborativos o enjambres ejecutando trabajos específicos.

Las características de los entornos de laboratorio suponen un problema a la hora de encontrar soluciones de bajo coste que puedan ser empleadas en el diseño de experimentos. Surgiendo de esta forma la necesidad de buscar soluciones, que intentando aproximar al máximo la precisión de las de mayor coste, generen una respuesta precisa, flexible, robusta y económicamente aceptable. Con este fin se propone el empleo de redes distribuidas de sensores de bajo coste, de forma que podemos suplir la falta de precisión de uno de los elementos compensándolo con la información de otros. Para tramitar este volumen de información se emplearán algoritmos que aprovechen la redundancia de información de la red.

Por otro lado, la necesidad de integrar la información recibida de los sensores con entornos de visualización virtual que permitan un seguimiento en tiempo real del objeto a estudiar mediante equipos informáticos se ha convertido en los últimos años en una herramienta fundamental del proceso de análisis. Por ello este tipo de soluciones no solo persigue la representación de la información en tiempo real, si no que busca también crear entornos virtuales interactivos que permitan al usuario modificar los parámetros del experimento y observar los resultados obtenidos en directo. Estos sistemas también pueden evolucionar presentando información de otro tipo como gráficos estadísticos, índice de seguimiento de tolerancia a fallos, etc. Sin embargo, al igual que los sensores, requiere de disponer de herramientas de software que nos permitan realizar prototipos rápidos y de bajo coste de aplicaciones orientadas a la gestión de la información de los experimentos generados. Por consecuencia la necesidad de tener un sistema de bajo coste dentro de este tipo de características nos impulsa a seleccionar herramientas de software libre bajo licencias tipo GNU, sistema operativo de tipo Unix.

1.2 Objetivos

Como hemos comentado en el apartado anterior, la virtualización de objetos reales, que son posicionados con exactitud es una necesidad fundamental para el estudio y análisis de prototipos físicos

en laboratorios. A partir de esta necesidad surge el objetivo básico del TFG, en el que se pretende obtener una solución robusta de bajo coste al problema del posicionamiento en entornos de laboratorio.

En la actualidad existen multitud de proyectos que buscando los mismos resultados exploran las diferentes tecnologías existentes, (posicionamiento por tratamiento de imagen, medición de desfase de la señal que llega a las bases, etc.). Por ello el objetivo básico de este proyecto es explorar una solución muy económica basada en el empleo de antenas de radiofrecuencia y sensores de ultrasonido empleadas para medir la distancia entre múltiples bases emisoras y receptoras. A partir de esta información el objetivo final es la estimación de la posición lineal y angular del objeto con respecto a un sistema de ejes establecido, ya que calculando la distancia y posicionando distintos puntos de referencia del objeto nos permiten caracterizarlo y dibujarlo en un entorno virtual. Las distancias entre emisores y receptores se miden en función del tiempo de vuelo de la señal del ultrasonido, pero debido a la ausencia de sistemas robustos de sincronismo, es necesario introducir un método de sincronización en el protocolo que permita evaluar el tiempo entre instantes de emisión y el de recepción. Para resolver este problema se emplearán las señales de radiofrecuencia teniendo en cuenta que son ondas electromagnéticas que se desplazan a la velocidad de la luz, que es muy superior a la de los ultrasonidos.

De esta forma los objetivos principales del TFG son:

- Construcción física de un prototipo de localización (bases emisoras y sensores receptores).
- Desarrollo de experimentos de recogidas de datos.
- Creación de algoritmos asociados a: calibrado del sistema, y estimación de estados de posición y ángulos mediante la fusión de los datos de diferentes sensores.
- Análisis de los datos.

1.3 Estructura del TFG

Este TFG se estructura de la siguiente forma:

- Capítulo 1: Introducción y objetivos. Se introduce al lector en los aspectos generales y exponiendo a su vez los objetivos fijados para su desarrollo.
- Capítulo 2: Estado del arte. Se presente la tecnología y herramientas desarrolladas hasta el momento para la localización de objetos mediante señales radio y de ultrasonido, así como algunos proyectos relacionados que se encuentran en vías de desarrollo.
- Capítulo 3: Desarrollo del Trabajo de Fin de Grado exponiendo que hardware y software se han seleccionado y porque se ha hecho.
- Capítulo 4: Instalación y pruebas con el fin de observar la precisión obtenida además de la calibración del sistema.
- Capítulo 5: Conclusiones extraídas de los experimentos y posibles líneas futuras que permitan mejorar los resultados.
- Para finalizar se Anexará información de utilidad para el seguimiento del proyecto, así como los esquemas de circuitos y los distintos códigos desarrollados.

2 ESTADO DEL ARTE

2.1 Introducción

En este capítulo realizaremos una introducción en lo que respecta a los sistemas de localización de objetos tridimensionales en interiores, así como los algoritmos que permiten reducir el índice de error de los sensores de medida (calibración) y los que realizan la fusión de la información de los diferentes sensores para establecer la estimación de la posición y ángulos de orientación del objeto a localizar. Debemos tener en cuenta que la función principal de todo sistema de localización es la de situar de la forma más precisa posible un punto en el espacio con respecto a un sistema de referencia

2.2 Clasificación topológica

La clasificación topológica se realiza en base a consideraciones estructurales de la red de sensores distribuidos empleada. De esta forma, atendiendo a las diferentes estructuras, se distinguen entre las siguientes topologías de redes de sensores distribuidos [1]:

2.2.1 *Sistemas Tokenless*

En este tipo de sistemas, según [1], el objeto a localizar no presenta ningún tipo de sensor para localizarlo, por lo tanto, participa de forma pasiva en el entorno y no precisan la colaboración del objeto a localizar. Algunos ejemplos de aplicación de este tipo de redes son los sistemas de sonar submarino para localización de objetos subacuáticos, los sistemas lidar empleados para obtener mapeados de la orografía o del entorno de vehículos autónomos no tripulados y sistemas basados en visión artificial mediante cámaras en vehículos de conducción automática. Cada una de estas tecnologías presenta una serie de ventajas y desventajas, pero salvo los sistemas de visión artificial, para el resto de los ejemplos resulta difícil plantear su uso en una red distribuida de bajo coste, debido al alto precio de los sensores.

2.2.2 *Sistemas Token*

Los sistemas tipo Token, definidos en [1], requieren que el objeto a localizar porte sensores con los que sea capaz de posicionarse. Dentro de este tipo de sistemas podemos discernir entre dos tipos.

- **Sistemas asistidos:** Son aquellos sistemas en los que, además de los sensores situados en el objeto a localizar, se emplea una red distribuida de sensores situados en el medio cuya posición es conocida. De esta forma se calcula la posición relativa del objeto con respecto a los sensores situados en el entorno.
- **Sistemas no asistidos:** Toda la información necesaria para posicionarse es extraída del propio sensor o sensores instalados en el objeto.

2.2.3 *Sistemas Distribuidos*

Por último, en los sistemas distribuidos, como se indica en [1], no existe ninguna base de posición conocida, sino que nos encontramos en un entorno donde existen diversos nodos que intercambian información posicionándose entre sí. De esta manera un nodo obtendrá su posición con respecto a sus vecinos, de forma que realizando este proceso para todos los nodos de la red se obtiene la topología completa de la misma.



Figura 2-1 Tipos de sistema de localización (tomado de [1]).

2.3 Clasificación en base a la tecnología

Además de la clasificación por la topología se puede realizar una segunda atendiendo a la tecnología, entendida como el fenómeno físico, sobre la que funciona de aquí se pueden identificar dos grupos en función de la naturaleza de la onda.

2.3.1 Señales de onda electromagnética

Este tipo de ondas se basan en alteraciones de los campos electromagnéticos, los cuales son perpendiculares entre sí y a la dirección de transmisión. Como principal ventaja es su capacidad para enviarse a través del vacío, así como que presentan velocidades muy elevadas atendiendo a la ecuación $c = \frac{1}{\sqrt{E_o \mu_o}}$ donde solo dependen de la permeabilidad eléctrica (E_o) y magnética (μ_o) del espacio llegando a alcanzar la velocidad de $3 * 10^8 \text{ m/s}$ en el vacío. Como desventaja presentan que en ciertos medios como el agua presentan una caída de energía muy rápida en frecuencias altas.

2.3.1.1 Laser (LIDAR)

Este tipo de tecnología se basa en un sistema formado por un telemetro laser y un espejo móvil capaz de rotar en sus ejes x e y. Para obtener la topología se realizan movimientos conocidos del espejo, de esta forma se puede conocer el ángulo de salida del rayo. A continuación, para cada ángulo se realiza una medición mediante el telemetro laser, que recoge el echo asociado al rebote de la emisión de luz láser sobre el objeto de estudio. Como consecuencia, conociendo la velocidad de propagación se calcula la distancia del impacto asociada para cada dirección, con lo que se genera la denominada nube de puntos que representa la posición geométrica del objeto estudiado, y si hay suficientes puntos permite realizar la estimación de su superficie exterior.

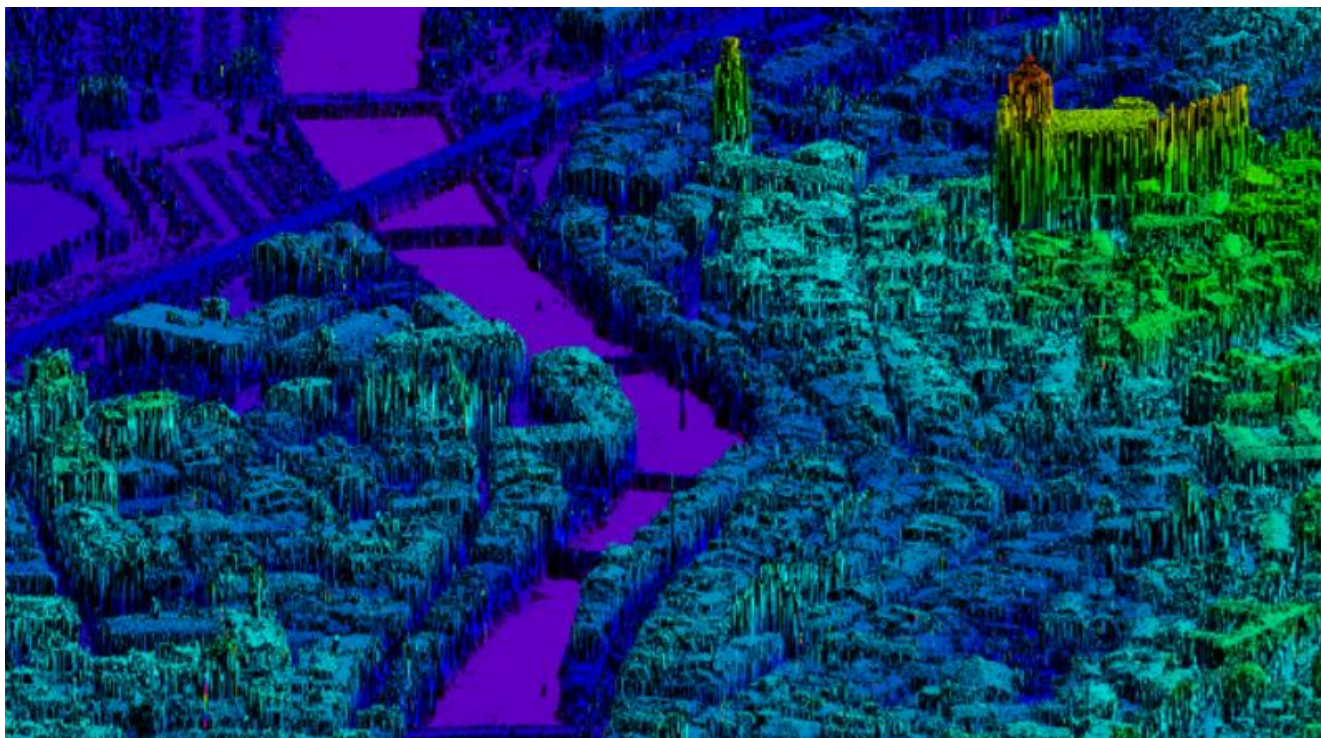


Figura 2-2 centro de Girona mediante técnicas LIDAR (tomada de [13])

Este tipo de tecnología se convierte en una herramienta de gran valor a la hora de realizar estudios topográficos del terreno o digitalizaciones de objetos. Por un lado, esta tecnología puede emplearse desde sistemas aéreos que permiten abarcar grandes superficies de terreno, o empleando trípodes fijos, lo que permite realizar digitalizaciones precisas de edificios o incluso elementos de menor tamaño

La ventaja principal de este tipo de sistemas es que permiten barridos muy precisos, de forma que empleando una potencia del láser adaptativa es posible descubrir edificaciones ocultas por vegetación muy densa, que a simple vista no se podrían apreciar. Sin embargo, este tipo de sistemas precisan de un tiempo relativamente elevado para la adquisición de datos, así como de un elevado procesamiento de los datos una vez obtenidos, lo que implica que una adquisición en tiempo real de bajo coste en este tipo de sistemas es difícil de conseguir.

2.3.1.2 UWB

La banda ultra ancha (UWB) es una tecnología de radio basada en los estándares IEEE 802.15.4a y 802.15.4z, que permite la medición muy precisa del tiempo de vuelo de la señal de radio, lo que lleva a una medición de distancia / ubicación con precisión de centímetros. Además de esta capacidad única, UWB ofrece capacidad de comunicación de datos mientras usa muy poca energía, lo que permite el funcionamiento durante años sin necesidad de recargarlo o reemplazarlo. Al combinar una ubicación y comunicación precisas, UWB también ofrece una nueva forma de comunicarse de forma segura a través de la red inalámbrica, lo que abre la puerta a nuevas formas de transacciones seguras.

UWB difiere sustancialmente de las estrechas frecuencias de banda de radio (RF) y tecnologías como el Bluetooth. UWB usa un gran ancho de banda del espectro de RF para transmitir información. Por lo tanto, UWB es capaz de transmitir más información en menos tiempo en comparación a Bluetooth.

Mientras que Bluetooth, Wi-Fi, teléfonos inalámbricos y demás dispositivos de radiofrecuencia están limitados a frecuencias sin licencia en los 900 MHz, 2.4 GHz y 5.1 GHz, UWB hace uso de un

espectro de frecuencia recientemente legalizado. UWB puede usar frecuencias que van desde 3.1 GHz hasta 10.6 GHz y cada canal de radio tiene una anchura de más de 500 MHz, dependiendo de su frecuencia central.

El hecho de estar compartiendo bandas de frecuencia con otros dispositivos ha motivado que, aunque esto les permite tener una alta productividad, han de estar situados relativamente cerca. Las ventajas que ofrece UWB son su bajo consumo (como emisor de ondas de radio), bajo coste y alta productividad, lo que marca esta tecnología como el futuro de las redes de localización.

TECHNOLOGY					
WHERE USED					
ACCURACY	 Centimeter	 1-5 meters	 5-15 meters	 Centimeter to 1 meter	 5-20 meters
RELIABILITY	★★★★★ Strong immunity to multi-path and interference	★☆☆☆☆ Very sensitive to multi-path, obstructions and interference	★☆☆☆☆ Very sensitive to multi-path, obstructions and interference	★★★★★	★★★★☆ Very sensitive to obstructions
RANGE / COVERAGE	 Typ. 70m Max 250m Typ. 250m² per anchor	 Typ. 15m Max 100m Typ. 25m² per beacon (for 2m accuracy)	 Typ. 50m Max 150m Typ. 100m² per access point (for 5m accuracy)	 Typ. 1m Max 5m Typ. 25m² per reader	N/A
DATA COMMUNICATIONS	 up to 27Mbps	 up to 2Mbps	 up to 1Gbps		
SECURITY (PHY LAYER)	★★★★★ Distance-Time bounded protocol	★☆☆☆☆ Can be spoofed using relay attack	★☆☆☆☆ Can be spoofed using relay attack	★☆☆☆☆ Can be spoofed using relay attack	N/A
LATENCY	★★★★★ Typ. <1ms to get XYZ	★☆☆☆☆ Typ. >3s to get XYZ	★☆☆☆☆ Typ. >3s to get XYZ	★☆☆☆☆ Typ. 1s to get XYZ	★★★★☆ Typ. 100ms to get XYZ
SCALABILITY DENSITY	★★★★★ >10's of thousands of tags	★★★☆☆ Hundreds to a thousand tags	★★★☆☆ Hundreds to a thousand tags	★★★★★ Unlimited	★★★★★ Unlimited
POWER & BATTERY	 5nJ/b TX - 9nJ/b RX Coin Cell	 15nJ/b RX/TX Coin Cell	 50nJ/b RX/TX Lithium Battery	 Passive	 Lithium Battery
TOTAL COST (infrastructure, tag, maintenance)					

Figura 2-3 Comparación de tecnologías de posicionamiento por RF

2.3.1.3 Señales de Radiofrecuencia

2.3.2 Señales sónicas

Este tipo de ondas usan como medio las perturbaciones de las características mecánicas del medio (densidad y presión) provocando alteraciones que se van desplazando con el tiempo sin provocar el desplazamiento sustancial de los átomos de la materia. Estas se pueden emplear para transportar energía, como una ola del mar, o información, como las ondas sonoras que se generan al hablar. Su principal desventaja radica en la dependencia de un medio material para poder expandirse de forma que, por ejemplo, este tipo de ondas no son capaces de viajar en el vacío. Por otra parte, existen las sustanciales diferencias de velocidad dependiendo del medio que use como soporte. Sin embargo, esas medidas se pueden modelizar y calcular mediante el conocimiento de ciertos factores del medio. Concretamente la velocidad del sonido viene definida por (Presión y la Densidad del aire).

2.3.2.1 Ultrasonidos

Onda mecánica cuya frecuencia se sitúa por encima del espectro audible por el ser humano, concretamente por encima de los 20 KHz. Actualmente sus aplicaciones son diversas y van desde aplicaciones en medicina, a medición de distancia mediante el tiempo de vuelo, pasando por utilidades en el estudio

Sin embargo, la principal limitación de este tipo de ondas para calcular en tiempo real la posición de un objeto viene de la mano de la velocidad del sonido. Esto se debe a que por la falta de velocidad con respecto a otro tipo de señales provoca que su tasa máxima de emisión este limitado. Suponiendo una velocidad de 344m/s en el aire y una distancia media de medición de aproximadamente 5 m se puede hacer un cálculo rápido se puede calcular que el tiempo mínimo para realizar el vuelo es de 0,014 s es decir se puede hablar de un ancho de banda aproximado de 70 Hz. Siendo siempre este variable con la distancia a la que se encuentren los receptores de los transmisores.

2.3.2.2 Sonar

Dentro del ámbito de las señales sónicas para el posicionamiento, tal vez uno de los sistemas más extendidos, sobre todo en el ámbito militar, es el Sonar. Este sistema emplea ondas sónicas para determinar la distancia del emisor a los objetos del medio.

La idea principal surge en la naturaleza donde animales como los murciélagos o los delfines utilizan la ecolocalización para navegar en condiciones de baja visibilidad, donde calculan la distancia a los objetos por el tiempo que tardan en recibir el eco emitido. Este tipo de localización podemos observarla tanto en medios fluidos, como los cetáceos, o gaseosos, como los murciélagos al volar por la noche.



Figura 2-4 Sonar LWHP53, Indra

Cuando se traslada esta técnica a la tecnología se pueden observar distintas aplicaciones, desde la localización, al espionaje pasando incluso por la comunicación submarina. Además, podemos distinguir entre dos clasificaciones diferentes. Por un lado, se puede clasificar en función de la fuente emisora

- **Activos:** el emisor y el receptor se encuentran en el sistema que pretende localizarse, pudiendo ser incluso en mismo elemento conociéndose entonces como transceptores.

- Pasivos: El elemento que pretende localizarse no emite ningún tipo de señal, sino que se dedica a escuchar las emisiones del medio, este sistema presenta la característica de ser más discreto por lo que se convierte en ideal para misiones encubiertas de los submarinos militares. Sin embargo, hace que sea más complicado posicionarse con respecto al resto del entorno.

La segunda calificación viene de la mano del tipo de onda emitida donde esta puede ser:

- Continua: Permite un mapeo continuo en todas las direcciones, pero implica el uso de modulaciones por ejemplo emisiones tipo CHIRP (onda triangular analógica) para poder calcular las distancias.
- Pulsada: Se van emitiendo pulsos espaciados en el tiempo, escuchando los rebotes de estos. Su ventaja principal se basa en su facilidad de implementación donde solo hace falta un emisor un receptor y un reloj para poder tener todos los datos necesarios.

2.4 Técnicas de medición de distancia

Dentro del ámbito de medición de distancia mediante el uso de ondas existen distintos métodos los cuales ofrecen características diversas, tanto en la señal más apropiada para emplear como en la precisión o complejidad de los algoritmos.

2.4.1 RSSI (*Received Signal Strength Indicator*) Amplitud de la señal recibida

Este tipo de técnica mide la potencia de la señal recibida que es comparada con una señal de características conocidas por el receptor. De esta forma, el receptor al recibir la señal la compara con un modelo de referencia de atenuación de señal en distancia, como el que puede verse en la Figura 2-5.

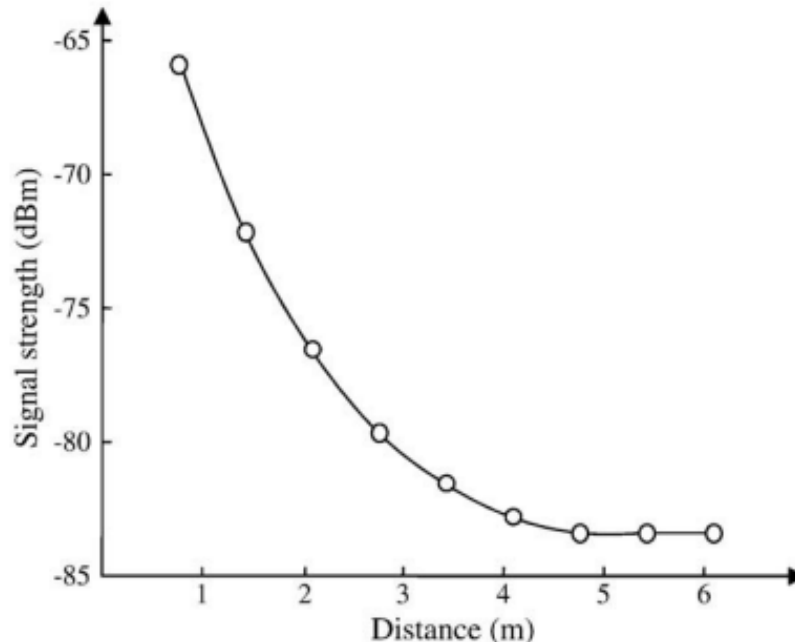


Figura 2-5 curva de atenuación de antenas inalámbricas [2]

El problema de este tipo de soluciones es que es muy sensible a las condiciones ambientales, requiriendo caracterizar de forma precisa el entorno, por lo que se necesita el uso de diversos sensores (presión, temperatura, humedad, etc.) y complejos algoritmos de calibración. En ningún caso este tipo de mecanismos permite obtener la localización de un objeto con la precisión y robustez que los experimentos básicos de laboratorio requieren [2]

2.4.2 ToF (Time of Flight) Tiempo de vuelo

Esta técnica consiste en calcular la distancia entre dos puntos mediante el tiempo de vuelo de la señal [3], como podemos ver en la Figura 2-6, en donde se aplican varias medidas obtenidas mediante esta técnica para realizar la trilateración de la posición de un objeto.

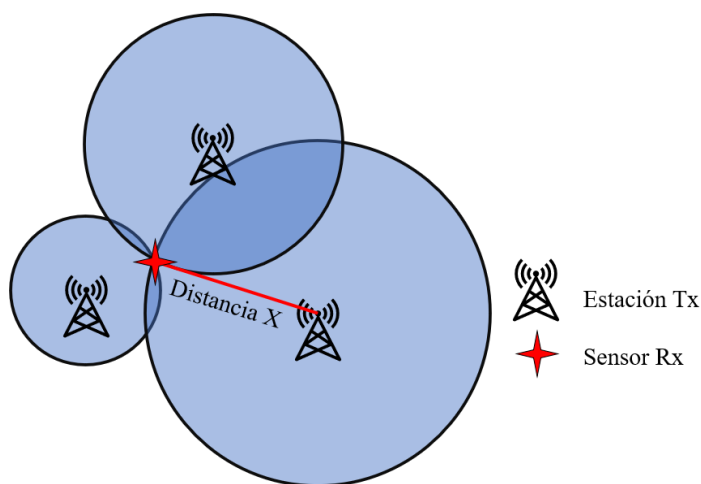


Figura 2-6 Técnica de medición de distancia por ToF.

Para calcular el tiempo de vuelo necesitaremos la velocidad de la señal en el medio para obtener medidas precisas, así como de otros datos adicionales (temperatura, presión, humedad, etc.) que permitan generar un modelo preciso de la velocidad de propagación. Sin embargo, la principal problemática viene de la necesidad de crear un mecanismo de sincronización entre emisor y receptor, pero que no requiera de sincronizar previamente sus relojes. Para ello se hace uso de algún tipo de técnica que permita sincronizar el inicio de la emisión de pulsos de la señal previamente al lanzamiento de estos pulsos. Un ejemplo de esto es el uso de señales RF (Radio Frecuencia) empleadas para iniciar la sincronía del protocolo de detección que se basa en la emisión de ultrasonido de forma que por medio de la diferencia de tiempo entre la llegada de ambas señales [4]. Es posible establecer la distancia asumiendo que la velocidad del ultrasonido es despreciable con respecto a la de la radio frecuencia.

Una vez calculadas las distancias entre el sensor a posicionar y las estaciones bases pueden emplear diferentes métodos de fusión para realizar la estimación de la posición, como por ejemplo el método de la trilateración. Método usado, por ejemplo, por los sistemas de geolocalización (GPS, Glonas, Galileo...) [4].

Otra alternativa es la medición directa del tiempo de vuelo de señales de radiofrecuencia, pero debido a las altas velocidades de este tipo de señales y a que en el entorno de laboratorio las distancias son de 1 a 6 metros, se necesita hardware capaz de generar sincronismos de alta velocidad y evaluar la información en tiempos muy pequeños (nanosegundos) que no se encuentran disponibles como sistemas de bajo coste.

2.4.3 AoA (Angle of Arrival) Angulo de llegada

Con esta técnica en lugar de utilizar la distancia entre los transmisores y el receptor, se usa el ángulo de incidencia de la señal. Para ello un receptor realiza una estimación del ángulo de llegada de una señal por medio de la diferencia de tiempo, la amplitud, o la fase obtenida en dos antenas receptoras que se

encuentran en una posición relativa conocida. Una vez obtenidos los ángulos de llegada se procede a realizar una triangulación de la señal para estimar la posición del emisor.

Los algoritmos con base en la técnica de AoA presentan, en general peores rendimientos respecto a lo de tiempo de vuelo, además de que suelen ser algoritmos más complejos que los empleados en otro tipo de técnicas

2.4.3.1 TDoA (Time Difference of Arival) diferencia del tiempo de llegada

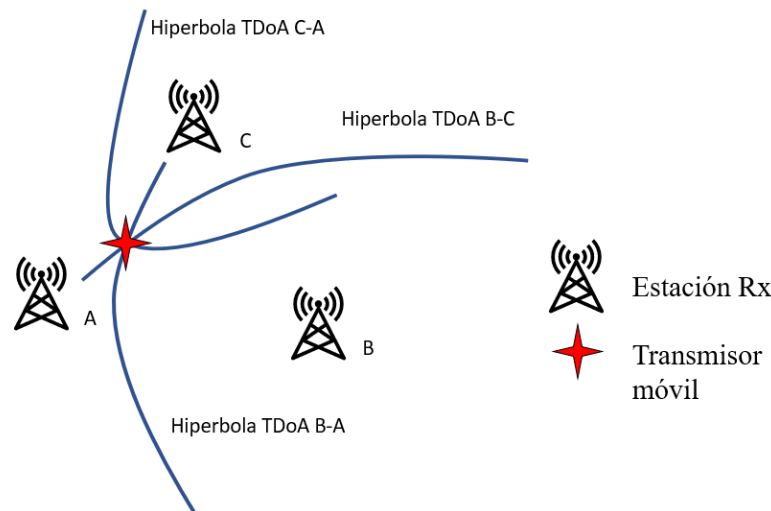


Figura 2-7 Técnica de medición de distancia por TDoA

Este método utiliza las diferencias del tiempo de llegada de una señal entre un emisor y varios receptores. A partir de los datos de la distancia obtenida desde varios receptores es posible obtener la posición del objeto móvil. Para ello se necesita un sensor base que inicie una y dos sensores subordinados que miden la diferencia de tiempo entre la llegada, de forma que, conociendo la situación de las estaciones se puede resolver las ecuaciones matemáticas que posicionen el objeto.

La principal ventaja de estos sistemas es que no precisan de la coordinación entre las estaciones y el elemento a localizar, sin embargo, necesitan más balizas y obtienen peor resolución que la ofrecida por otro tipo de técnicas. Podemos ver un ejemplo de este tipo de técnicas en [4], donde trata de posicionar un punto sonoro mediante la diferencia de llegada de la onda mecánica. El principal problema de la aplicación de este tipo de sistemas es la necesidad de un mecanismo de sincronización altamente preciso y robusto entre los relojes de las estaciones receptoras, lo cual implica requerimientos difíciles de obtener con sensores de bajo coste.

2.5 Técnicas de posicionamiento

Una vez determinadas las distancias entre el nodo a posicionar y los nodos de posición conocida se precisan realizar los cálculos necesarios para poder convertir los datos obtenidos en posiciones, para ello plantearemos dos problemas uno bidimensional, empleado por ejemplo en problemas de localización de personal en un edificio. Y otro tridimensional que resuelve los problemas de posicionamiento espacial que queremos solucionar para este proyecto.

2.5.1 Trilateración en 2D

Cuando el problema es bidimensional para poder estimar la posición de un nodo desconocido es mediante el cálculo de la trilateración en 2 dimensiones, para ello se precisa conocer la situación relativa

entre 3 nodos generando de esta manera un sistema de referencia. Con el fin de simplificar los cálculos se establece un sistema con las siguientes coordenadas

- Nodo A (0, 0)
- Nodo B (h, 0)
- Nodo C (0, k)

Donde h y k son constantes conocidas.

El siguiente paso es obtener la distancia mediante cualquiera de las técnicas de posicionamiento indicadas en 2.4. Una vez obtenidas estas se generan las ecuaciones de 3 circunferencias con centro en los nodos y de radio igual a la distancia entre este y el punto a calcular.

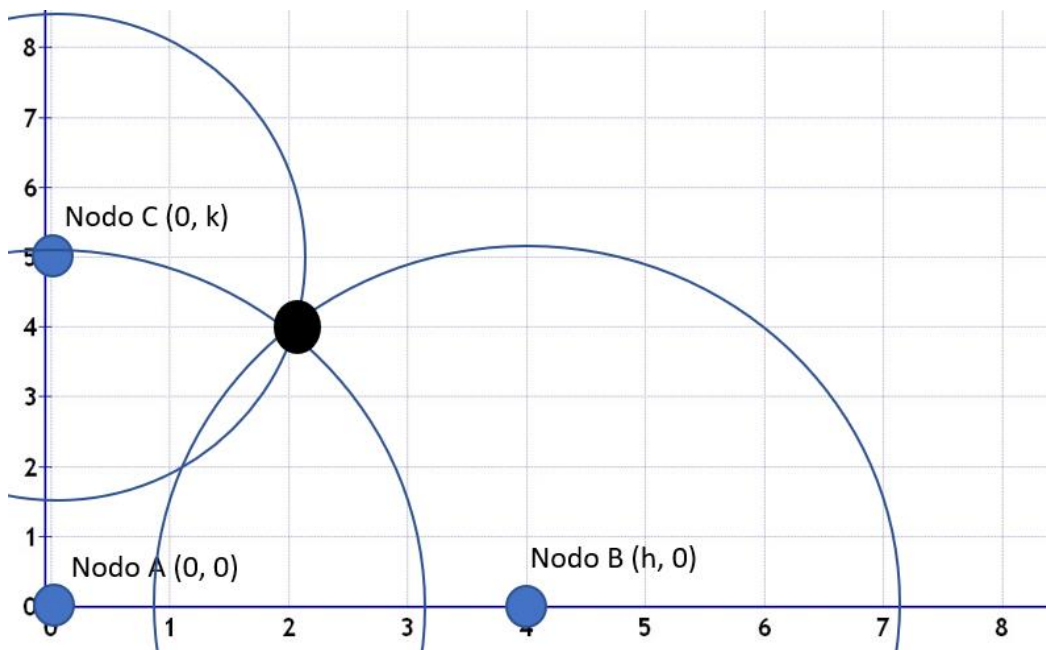


Figura 2-8 estimación grafica por trilateración

- Nodo A: $x^2 + y^2 = r_{A-0}^2$
- Nodo B: $(x + h)^2 + y^2 = r_{B-0}^2$
- Nodo C: $x^2 + (y + k)^2 = r_{C-0}^2$

A continuación, se despeja la x^2 y la y^2 en el Nodo A.

- Nodo A: $x^2 = r_{A-0}^2 - y^2$, $y^2 = r_{A-0}^2 - x^2$,
- Nodo B: $(x + h)^2 + y^2 = r_{B-0}^2$
- Nodo C: $x^2 + (y + k)^2 = r_{C-0}^2$

De esta manera ya se pueden sustituir ambos valores en los Nodos B y C obteniendo ecuaciones con una única incógnita.

- Nodo B: $(x + h)^2 + r_{A-0}^2 - x^2 = r_{B-0}^2$
- Nodo C: $r_{A-0}^2 - y^2 + (y + k)^2 = r_{C-0}^2$

Resolviendo ambas ecuaciones se llega al resultado:

- Nodo B: $x = \frac{r_{B-0}^2 - r_{A-0}^2 - h^2}{2h}$

- Nodo C: $y = \frac{r_{C-0}^2 - r_{A-0}^2 - k^2}{2k}$

De esta manera como todos los parámetros son conocidos y definen dos rectas lineales una centrada en un valor de x y la otra en un valor de Y, el resultado perseguido es la intersección entre ambos Figura 2-9 Representación gráfica del resultado analítico de la trilateración bidimensional

- Nodo 0: $(\frac{r_{B-0}^2 - r_{A-0}^2 - h^2}{2h}, \frac{r_{C-0}^2 - r_{A-0}^2 - k^2}{2k})$

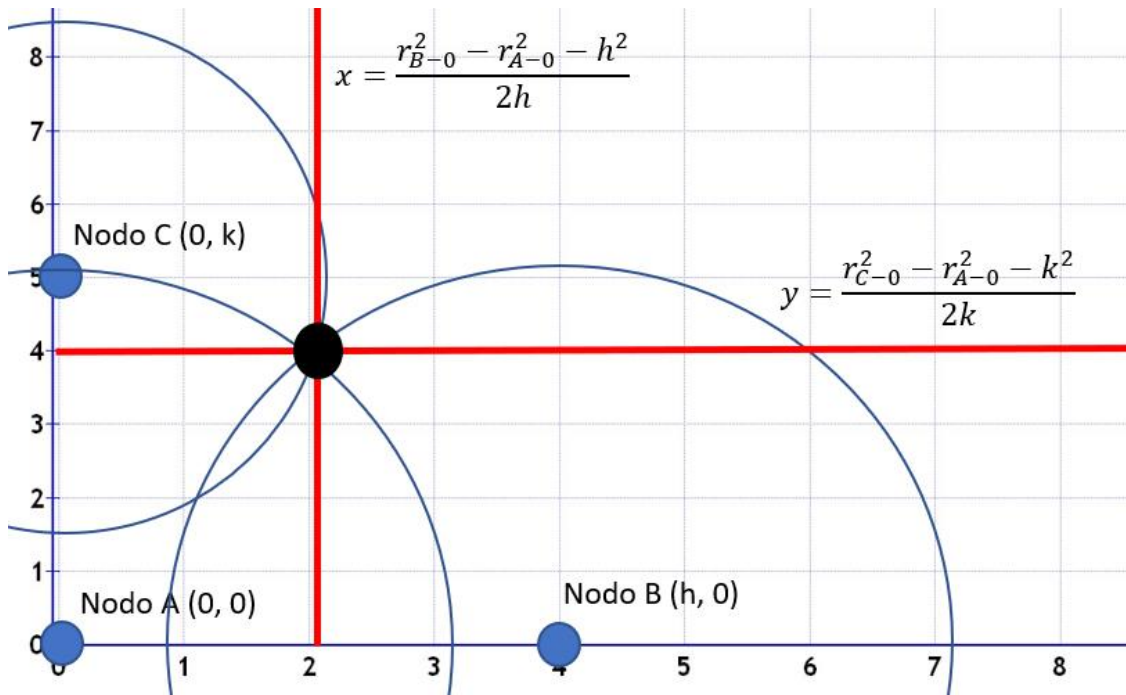


Figura 2-9 Representación gráfica del resultado analítico de la trilateración bidimensional

A mayores se podrían obtener resultados con el uso de solo dos nodos conocidos cruzando la recta obtenida con la ecuación de una de las circunferencias conocidas. El problema surge en que en este caso el resultado es doble es decir se precisa seleccionar una de las dos soluciones mediante un criterio valido. Por ejemplo, si se conoce que todas las posiciones se encuentran en el primer cuadrante nos quedaremos siempre con el resultado positivo de X e Y.

2.5.2 Trilateración en 3D

Para este tipo de casos de manera genérica se precisan un mínimo de 4 nodos no alineados, es decir que no estén incluidos en el mismo plano. Para solucionar este tipo de situaciones se puede seguir una dinámica muy similar a la establecida en 2.5.1. Para ello lo primero a plantear son las mismas ecuaciones, pero en este caso para los tres ejes marcando igualmente un nodo en el origen de coordenadas, por ello el sistema de referencia quedaría de la siguiente forma.

- Nodo A (0, 0, 0)
- Nodo B (h, 0, 0)
- Nodo C (0, k, 0)
- Nodo D (0, 0, t)

De igual forma que en 2.5.1 se precisa de conocer la distancia de Nodo a calcular a los cuatro puntos, pudiendo así representar ecuaciones de esferas con centro en cada nodo y de radio igual a la distancia de ese nodo conocido al Nodo 0. Teniendo el siguiente sistema:

- Nodo A: $x^2 + y^2 + z^2 = r_{A-0}^2$

- Nodo B: $(x + h)^2 + y^2 + z^2 = r_{B-0}^2$
- Nodo C: $x^2 + (y + k)^2 + z^2 = r_{C-0}^2$
- Nodo D: $x^2 + y^2 + (z + t)^2 = r_{D-0}^2$

A continuación, se procede a despejar x, y, z de la ecuación del Nodo A.

- Nodo A: $x^2 = r_{A-0}^2 - y^2 - z^2; y^2 = r_{A-0}^2 - x^2 - z^2; z^2 = r_{A-0}^2 - y^2 - x^2$
- Nodo B: $(x + h)^2 + y^2 + z^2 = r_{B-0}^2$
- Nodo C: $x^2 + (y + k)^2 + z^2 = r_{C-0}^2$
- Nodo D: $x^2 + y^2 + (z + t)^2 = r_{D-0}^2$

Y se sustituyen los valores de x, y, z en las ecuaciones de los nodos B, C, y D

- Nodo B: $(x + h)^2 + r_{A-0}^2 - x^2 - z^2 + z^2 = r_{B-0}^2$
- Nodo C: $x^2 + (y + k)^2 + r_{A-0}^2 - y^2 - x^2 = r_{C-0}^2$
- Nodo D: $r_{A-0}^2 - y^2 - z^2 + y^2 + (z + t)^2 = r_{D-0}^2$

Simplificando se llega al siguiente resultado:

- Nodo B: $h^2 + 2hx + r_{A-0}^2 = r_{B-0}^2$
- Nodo C: $k^2 + 2ky + r_{A-0}^2 = r_{C-0}^2$
- Nodo D: $t^2 + 2tz + r_{A-0}^2 = r_{D-0}^2$

De esta manera despejando en cada ecuación las variables obtenemos las ecuaciones de 3 planos ortogonales entre sí.

- Nodo B: $x = \frac{r_{B-0}^2 - h^2 - r_{A-0}^2}{2h}$
- Nodo C: $y = \frac{r_{C-0}^2 - k^2 - r_{A-0}^2}{2k}$
- Nodo D: $z = \frac{r_{D-0}^2 - t^2 - r_{A-0}^2}{2t}$

Entonces el punto de corte de los tres planos se encuentra en las coordenadas:

- Nodo 0: $(\frac{r_{B-0}^2 - h^2 - r_{A-0}^2}{2h}, \frac{r_{C-0}^2 - k^2 - r_{A-0}^2}{2k}, \frac{r_{D-0}^2 - t^2 - r_{A-0}^2}{2t})$

Al igual que en 2.5.1 se puede hacer una estimación empleando solo 3 Nodos conocidos para ello se necesita poder discernir cuál de las dos soluciones es correcta. En sistemas como el GPS este tipo de acciones resulta relativamente sencilla puesto que resulta improbable que el punto a localizar se encuentre en el espacio. Para la explicación matemática se parte de las ecuaciones obtenidas con anterioridad desechando los resultados del Nodo D. Por lo que se parte del siguiente estado.

- Nodo A: $x^2 + y^2 + z^2 = r_{A-0}^2$
- Nodo B: $x = \frac{r_{B-0}^2 - h^2 - r_{A-0}^2}{2h}$
- Nodo C: $y = \frac{r_{C-0}^2 - k^2 - r_{A-0}^2}{2k}$

En este caso se tiene dos ecuaciones que representan planos, y por tanto se tiene la ecuación de una recta en 3 dimensiones. Y la ecuación de una esfera centrada en el Nodo A. Como la única incógnita que tenemos es la variable z se puede despejar esta de la ecuación del Nodo A y sustituir la x y la y por los valores ya obtenidos. Teniendo de esta manera.

- Nodo A $z = \sqrt{r_{A-0}^2 - (\frac{r_{B-0}^2 - h^2 - r_{A-0}^2}{2h})^2 - (\frac{r_{C-0}^2 - k^2 - r_{A-0}^2}{2k})^2}$

Como se adelantó anteriormente en este caso la solución es doble pues tenemos una negativa y otra positiva debido a la presencia de la raíz cuadrada.

2.6 Técnicas de filtrado y estimación

Cuando tenemos sistemas poco robustos y necesitamos mejorar sus resultados, existen dos posibles soluciones. Por un lado, podemos emplear sensores de mejor calidad, que obtengan medidas con menor varianza y por otro lado podemos usar modelos matemáticos que permitan mejorar sus características. La primera de las soluciones suele estar asociada a un incremento considerable de los costes del sistema, y la segunda permite el desarrollo de investigación asociada al análisis de los datos con costes mínimos. Dentro de estas herramientas matemáticas se puede hacer una distinción entre dos grupos (Estimaciones Bayesianas, y otro tipo de estimadores).

2.6.1 Estimación Bayesiana

La estimación bayesiana está basada en una interpretación subjetiva de la probabilidad, para lo cual se considera que esta presenta un grado determinado con respecto a la incertidumbre real.

Para ello un parámetro es visto como una variable aleatoria a la que, antes de la evidencia muestral, se le asigna una distribución a priori de probabilidad. De esta manera con cada nueva medida se realiza una corrección a esta predicción tratando de esta forma con forme se van realizando medidas obtener predicciones futuras más precisas.

2.6.1.1 Filtros Kalman

La primera de estas herramientas son los denominados filtros Kalman, creados por Rudolf Kalman en 1960. Este tipo de modelos sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal cuando el sistema está sometido a ruido Gaussiano, y realiza esta tarea de forma recursiva. A pesar de su importancia su problema reside en las restricciones a la hora de ser aplicado (lo cual le resta robustez), Por ello cuando se aplicó por primera vez, en la misión Apollo de la Nasa, lo hizo en su forma de filtro de Kalman extendido, que permite aplicar el algoritmo a sistemas dinámicos no lineales continuamente diferenciables.

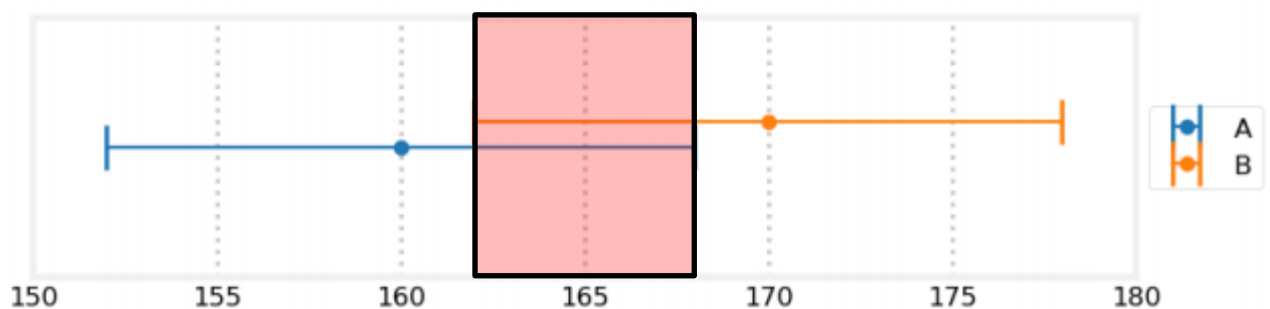


Figura 2-10 Diagrama medición varianza de dos sensores [5]

¿Cómo funcionan este tipo de filtros? Para poder explicarlo partiremos del supuesto que poseemos un sensor con una varianza relativamente alta. De esta forma podemos apreciar un cambio sustancial entre una y otra medición. Como consecuencia no podríamos realizar ninguna medida fiable del fenómeno a estudiar. Pero ¿qué ocurre cuando en lugar de una única medición realizamos más de una con sensores distintos? En la Figura 2-10 podemos ver la representación en un diagrama de la medición de dos sensores.

En este caso Roger R. Labbe Jr. [5] representa la medición de dos sensores A y B con una varianza de 8 centrados en 160 y 170 respectivamente. Inicialmente si solo tenemos una de las mediciones y sabiendo que nuestro sensor tiene una varianza de 8 al obtener la medida de 160 solo podríamos afirmar

que la medida real se encuentra entre [152,168]. Sin embargo, si ahora realizamos una segunda medición podríamos reducir el rango de donde se encuentra este valor real a la intersección entre ambas medidas en este caso de [162,168], zona sombreada en rojo en la Figura 2-10.

Una vez planteado como podemos aumentar la precisión de una medida usando más de un sensor de forma simultánea que ocurriría si no poseemos más de un sensor, ¿Se puede realizar una aproximación a este sistema con un único sensor? Para responder a esta duda surgen los filtros con memoria planteando que en lugar de realizar mediciones simultaneas de varios sensores, podemos realizar la aproximación de otro sensor con la predicción de la medida basada en la información obtenida anteriormente.

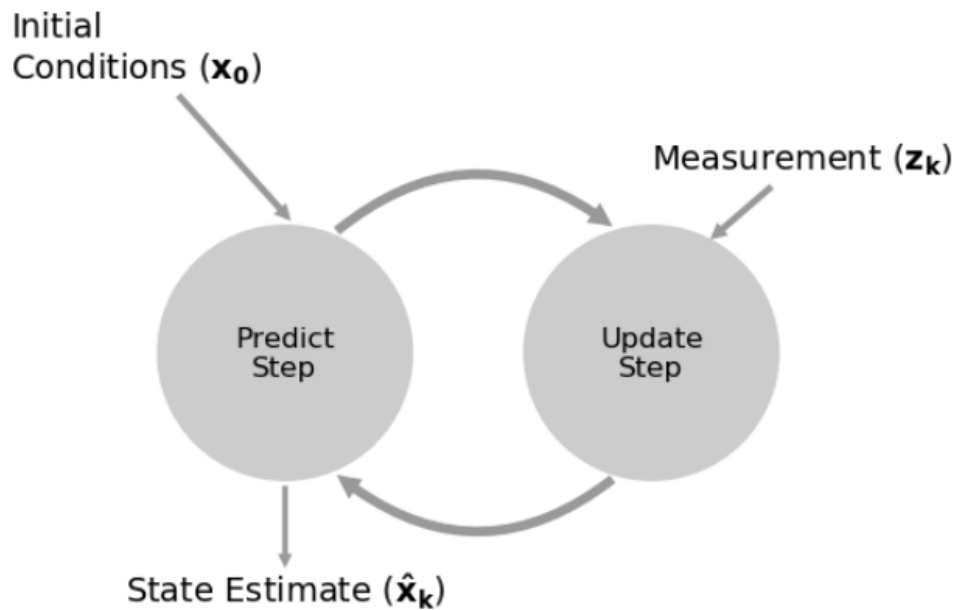


Figura 2-11 Estructura de un algoritmo recursivo de un filtro GH (tomada de [5])

Planteado este nuevo sistema, si las mediciones no son simultaneas deberíamos esperar que el valor real haya podido variar de una a otra medición. De esta forma, para poder calcular esa predicción se hace necesario plantear una hipótesis inicial que nos ayude a predecir cómo va a variar ese valor con respecto al tiempo. A esto se le conoce como el modelo de movimiento del sistema de estudio. Los filtros que realizan este tipo de aproximaciones basadas en el estado actual y el estado anterior (aquellos que permiten una implementación de tipo recursivo) son los conocidos como filtros G-H. Donde G hace referencia a la escala usada para la medida [5] y H representa la escala de cambio de esta [5]. Los filtros de Bayesianos como el filtro de Kalman forman parte de este tipo de soluciones, que calculan G y H en base a diferentes criterios.

En líneas generales, la constante G responde a como el filtro se adapta a la varianza de las medidas, es decir a su diversidad. Por ello en sistemas que presentan mucho ruido conviene usar valores bajos de G . Sin embargo, disminuir en exceso el valor de G implica que el filtro tarda más en adaptarse a los cambios en el medio, es decir, es menos sensible a las altas frecuencias de la señal de entrada. Este efecto puede observarse en la Figura 2-12, en donde valores mayores de G responden con mayor velocidad a la variación en la señal de excitación del sistema. Representando de esta forma un factor de escala entre el valor de la nueva medida que es observada y la predicción realizada por el modelo de movimiento.

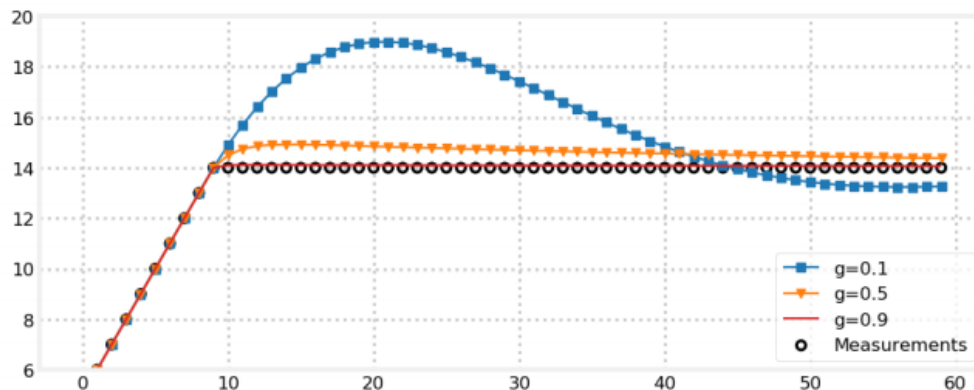


Figura 2-12 Comparación de la variación de H y la dx (tomada de [5])

Por otro lado, la constante H se asocia al nivel de confianza con respecto al modelo de movimiento y a las nuevas observaciones obtenidas, es decir la hipótesis acerca de cómo se producen los cambios en el sistema de estudio con el valor oculto que debemos estimar. De esta forma, un valor elevado de H implica que el modelo de movimiento planteado se modifica considerablemente a partir de las nuevas observaciones. Debemos tener en cuenta que a mayor precisión de la predicción que hemos realizado, mejor es el comportamiento de nuestra medida, y por lo tanto esta información debe incorporarse al modelo de movimiento empleado para predecir futuras observaciones

2.6.1.2 Filtros sigma point

El filtro de Kalman extendido realiza el cálculo del Jacobiano (la expresión del hiperplano tangente a una hipersuperficie en un punto) para aproximar a funciones lineales. Sin embargo, esta aproximación puede ser muy compleja para ciertas funciones no lineales o incluso no ser realizable analíticamente al tratarse de funciones no diferenciables en el punto. Este hecho provoca que en ocasiones la computación sea excesivamente lenta para tener respuestas a tiempo real. Para solucionar este tipo de problemas del EKF (Filtro de Kalman extendido) surgen los filtros Sigma-Point donde se intenta aproximar el Jacobiano mediante el muestreo de un número definido de puntos. A este nuevo tipo de filtros, enmarcados dentro de la familia de los filtros de Kalman, se les denomina como UKF (Unscented Kalman Filtre)

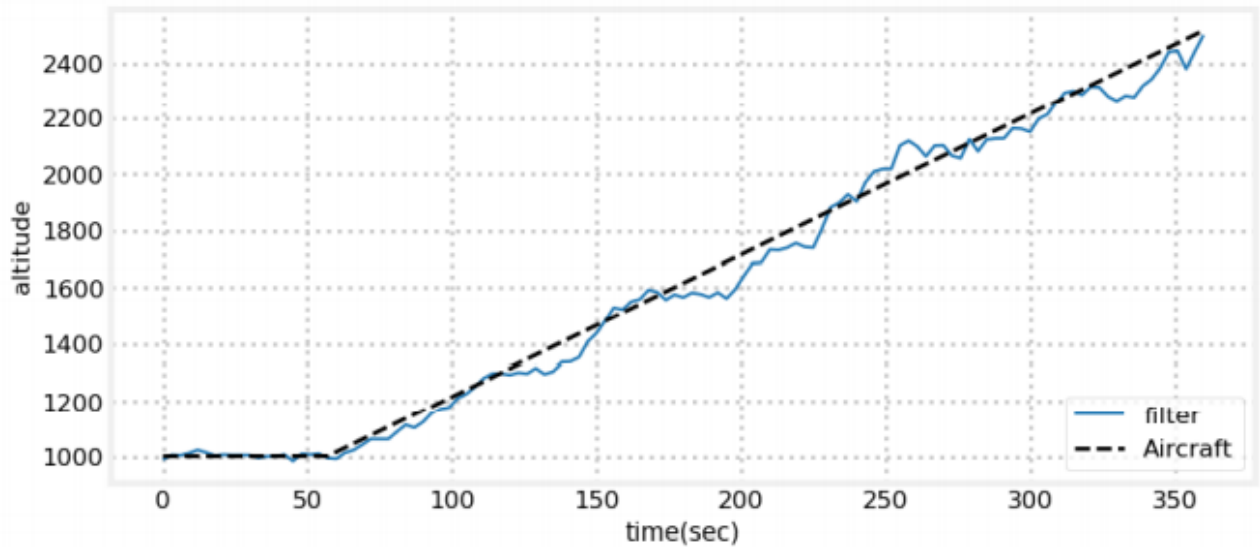


Figura 2-13 Comparativa de un UKF con la trayectoria real de un objeto estudiado (tomada de [5])

De esta manera, se permite el tratamiento de problemas no lineales como problemas lineales sin necesidad de grandes pérdidas de tiempo en la computación o de la necesidad de aplicar técnicas más complejas. Como es natural este tipo de filtro también tiene sus limitaciones de aplicación, sobre todo en lo relativo a las características del ruido y al tipo de no linealidades existentes en el modelo de movimiento planteado.

El filtro básicamente trabaja generando el muestreo de los denominados Sigma Point, que son localizaciones que, en base a unos criterios estadísticos, permiten obtener una buena aproximación al valor del Jacobiano en el punto de trabajo actual. Posteriormente se aplica una transformada a cada punto de forma que la información obtenida se emplea en la nueva predicción formada por los nuevos Sigma Point. A continuación, se repite la medida para realizar una nueva transformada Unscented, para obtener el nuevo estado y de nuevo repetir el proceso. Esta transformada rápida asociada a los Sigma Point, permite al algoritmo UKF ser más eficiente que el EKF para un gran número de sistemas no lineales. Además, al tener en cuenta razonamientos estadísticos para generar estas transformadas, permite obtener respuestas más robustas que otro tipo de filtros.

2.6.1.3 Filtros de partículas

Este tipo de filtro son una evolución natural de los ejemplos anteriores suponiendo que se presentan nuevos problemas en los que es necesario realizar estimaciones sobre múltiples elementos de forma simultánea, o en donde se pueden generar situaciones en donde dos posiciones diferentes de un mismo objeto generan observaciones idénticas. Este tipo de situaciones requiere una serie de características que el filtro debe de ser capaz de responder, tal y como se comenta en [5]

- **Multimodal:** Significa que la distribución de la probabilidad en la toma de las decisiones no debe ser concentrada en un único punto del espacio de decisiones, tal y como ocurre en el filtro de Kalman, sino que esta debe ser distribuida.
- **Oclusión:** Ocurre cuando la posición de un objeto no permite realizar una observación completa del objeto de estudio, lo cual genera situaciones en donde se trabaja con información parcial del entorno.
- **Comportamiento no lineal**

- **Ruido no gaussiano:** El tratamiento de sistemas en donde el ruido no sea Gaussiano no puede ser tratado por la familia de filtros vistos con anterioridad.
- **Multivariable:** debe de ser capaz de realizar estimaciones de varias variables forma simultánea.
- **Modelo del proceso desconocido:** En situaciones en las que desconocemos por completo el modelo de proceso (black box) los modelos distribuidos, como el filtro de partículas, permiten comportamientos adaptativos que generen respuestas más robustas.

Como consecuencia para poder resolver este nuevo tipo de problemas surgen los filtros de partículas donde a diferencia que los filtros de Kalman en lugar de concentrar la probabilidad en un único punto la distribuyen. De esta forma el modelo se adapta mejor a posibles errores. En este caso lo que hace el filtro es repartir donde se encuentra el objeto en diversos puntos dentro del espacio de control, dotándoles de peso probabilísticos a cada uno.

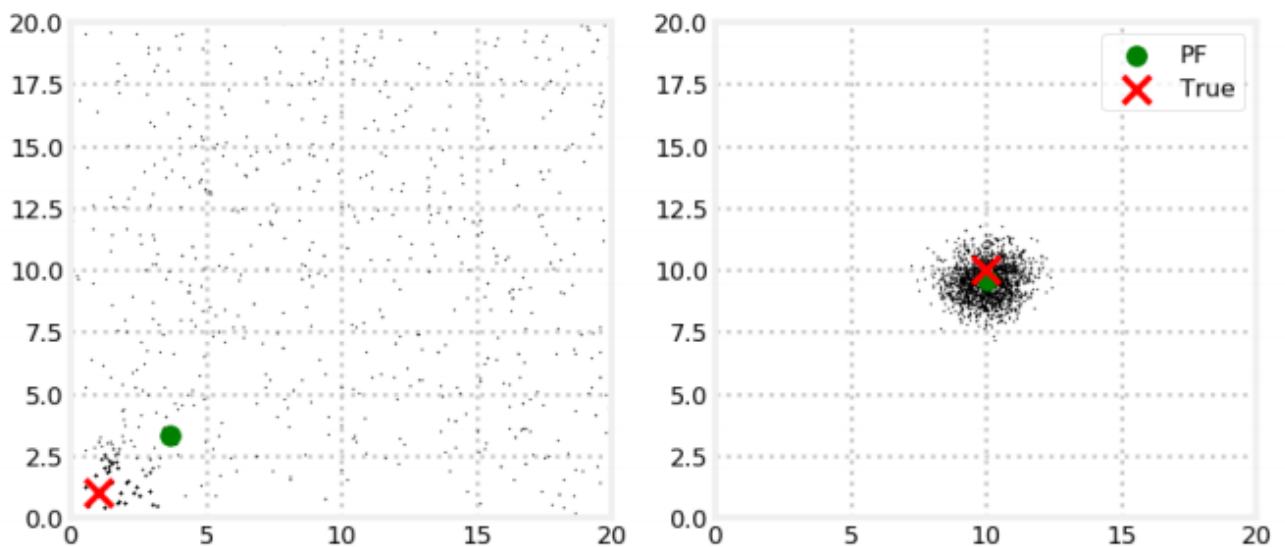


Figura 2-14 Evolución de un filtro de partículas (tomada de [5])

A continuación, se plantea la duda de como trabajar con una nube de puntos en lugar de un punto único a la hora de realizar la nueva predicción y compararlo con la medida obtenida. En primer lugar, podemos suponer como se va a desplazar el objetivo y aplicar este modelo para todos los puntos, sin embargo, las predicciones no son perfectas, por lo que deberemos tener en cuenta el ruido y las posibles perturbaciones a cada nueva predicción. El segundo factor a tener en cuenta es el método para realizar la medición y como eso afecta a la distribución de probabilidad. Una vez obtenida la nueva medida se realiza una nueva distribución de probabilidad dando más probabilidad a aquellos puntos que se han quedado más próximos a la nueva medida.

Por último, queda realizar la nueva estimación. Sin embargo, ahora se tiene una nube de puntos, por ello la nueva estimación se puede considerar como la suma de los pesos de las partículas. Así se considera que el nuevo estado estimado es el centro de gravedad de dichas partículas.

3 DESARROLLO DEL TFG

3.1 Diseño del prototipo

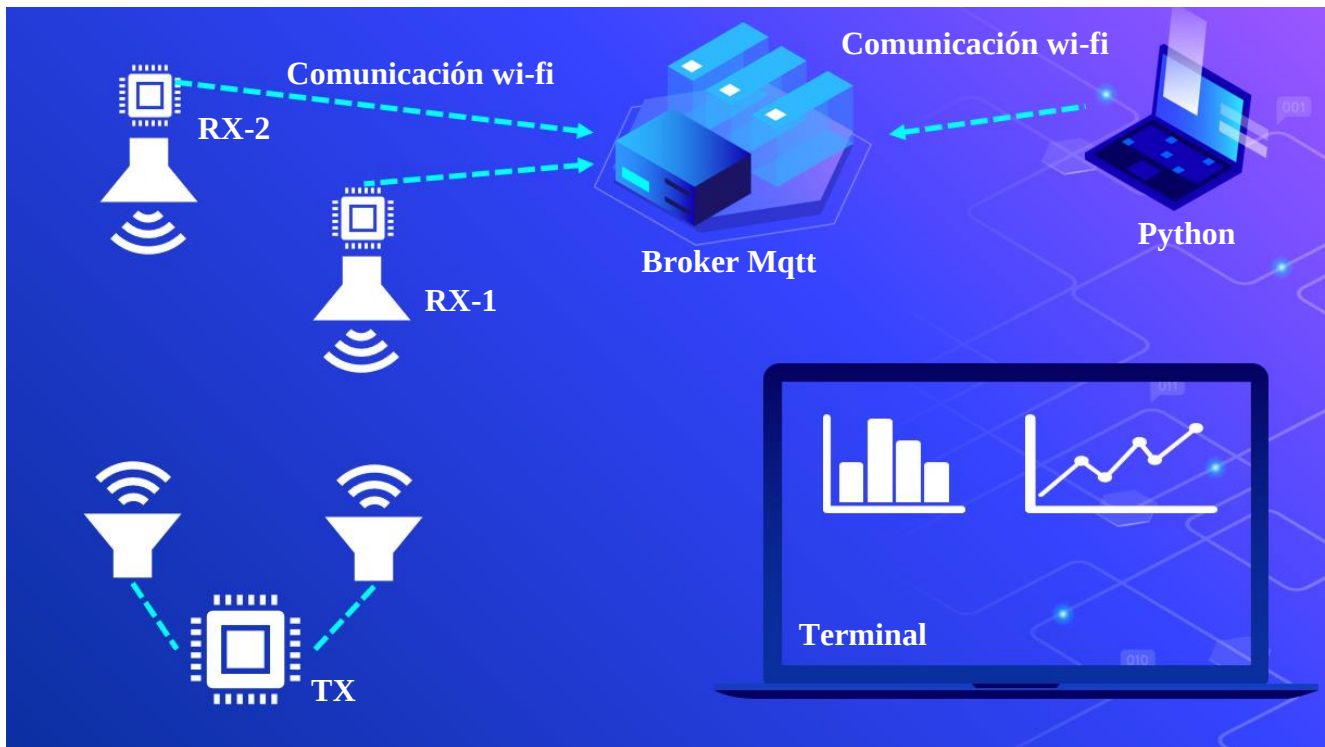


Figura 3-1 Esquema general del sistema de posicionamiento (elaboración propia)

Para la producción del prototipo se estableció un sistema distribuido formado por bloques aislados, lo que permita en un futuro modificar, mejorar o ampliar el sistema.

3.1.1 Selección de Hardware

El primer paso fue seleccionar los módulos, chips y sistemas que, por sus características técnicas, así como su precio los hacía la mejor solución para el desarrollo del proyecto. A continuación, se listan estos componentes, así como se indican sus características principales aportadas por el fabricante como el precio por unidad que ofertan.

3.1.1.1 HC-SR04

Se trata de un sensor de ultrasonidos digital dotado de un transmisor y un receptor y que dependiendo del modelo puede operar entre 3,3V y 5V. Presenta una precisión inicial entre 1 y 2 cm dependiendo de



Figura 3-2 Sensor HC-SR04, [14]

la distancia y con un alcance máximo de 4,5 m. El sensor trabaja con una resolución estándar de 0,3cm y usa una frecuencia de trabajo entorno a los 40Khz. El modulo presenta un precio entorno a los 0,85€.

Pensado para la medición de distancia mediante sistema digital consta de 4 Pinout (Vcc, GND, Trig, Echo). Los dos primeros sirven para la alimentación del módulo, mientras que el Trig se usa para controlar de forma digital el transmisor y Echo para recibir la señal. A continuación, se observa la gráfica tomada con osciloscopio del pulso eléctrico que provoca el ultrasonido Figura.

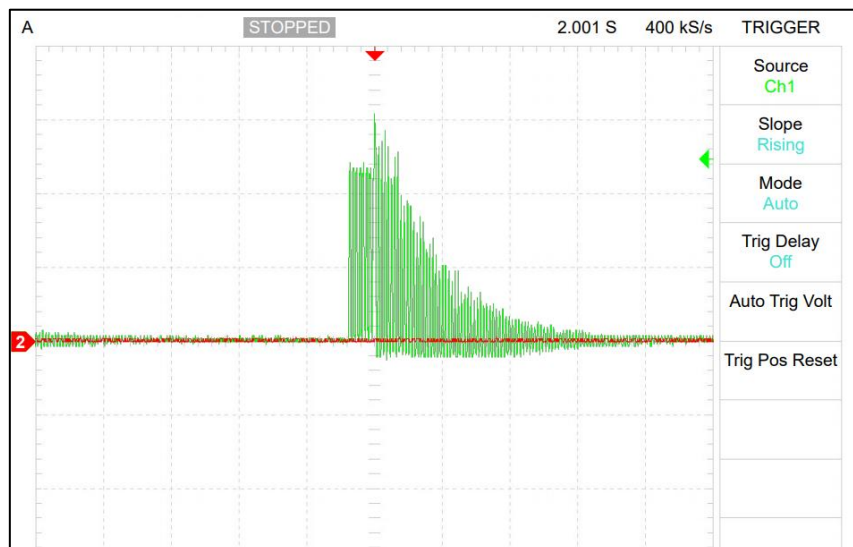


Figura 3-3 Brush ultrasonido tomado con osciloscopio

3.1.1.2 US-026

Sensor de distancia que puede ser alimentado con 3,3V o 5V lo que permite conectar de forma directa las salidas del sensor con el Chip ESP32 y el Arduino NANO. El sensor está formado por un transmisor y un receptor de ultrasonidos. Cuyas características son:



Figura 3-4 US-026

- Distancia de detección: 2m-4,5m.
- Precisión: 1cm-2cm.
- Angulo de apertura de los transductores: inferior a 15°.
- Frecuencia de trabajo: 40KHz.
- Precio: entorno a los 0,85€

El sensor está pensado inicialmente para la medición de distancia del sensor a un blanco objetivo. Para ello emplea 4 pines de entrada y salida, donde 2 de ellos son el Vcc y el GND preparados para la alimentación, mientras que los otros dos son el Trigger y el Echo. El funcionamiento es mediante un pulso de disparo de 10us por el pin Trigger para posteriormente enviar un *brush* de 8 pulsos de ultrasonido a 40KHz. Para posteriormente contar el tiempo de recepción del rebote.

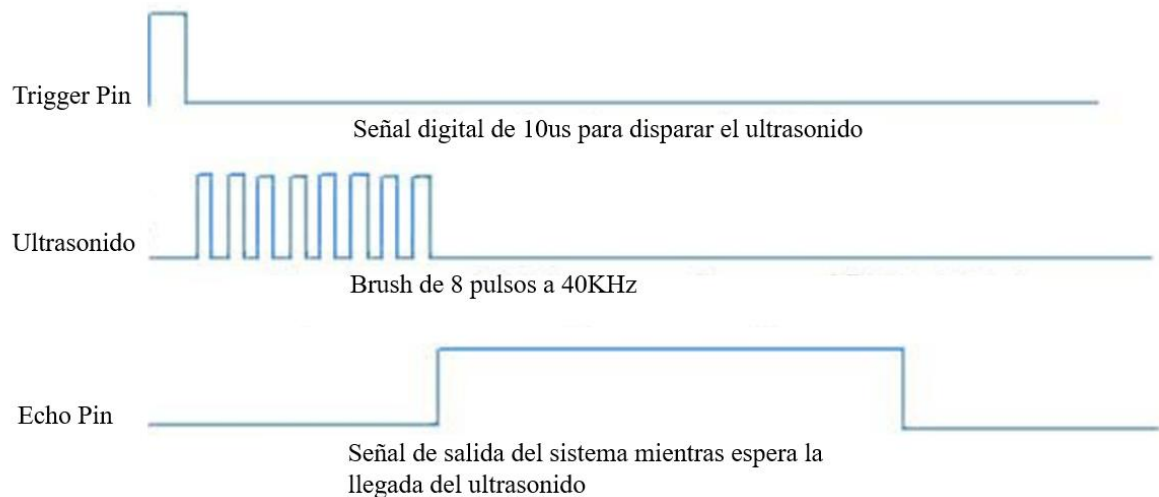


Figura 3-5 Secuencia de funcionamiento de los módulos de ultrasonido

3.1.1.3 NRF24L01

El NRF24L01 es un módulo de radiofrecuencia con capacidad “half-duplex”, por lo que permite una comunicación bidireccional no simultánea.

Características según el fabricante [6]

- Rango de frecuencia: 2,4GHz-2,525GHz con 125 canales de 1MHz.
- máxima emisión de datos: 2Mb/s.
- Modulación: GFSK.
- Rango de voltaje: 1,9V-3.6V.
- Corriente de operación máxima: 13.5mA.
- Corriente de operación mínima: 26uA.
- Entrada lógica: tolerados 5V.
- Alcance de comunicación: máximo 100m en LOS (line of sight)



Sin embargo, con la antena básica permite alcances entre 20 y 30 m en entornos de laboratorio siempre y cuando no se sitúen obstáculos para la comunicación. El chip presenta 8 Pinout (Vcc, GND, CE, CSN, SCK, MOSI, MISO e, IRQ). Los cuales los dos primeros son entradas para la alimentación y los 6 últimos son entradas/salidas lógicas que permiten el control del módulo mediante chips controladores como Arduino.

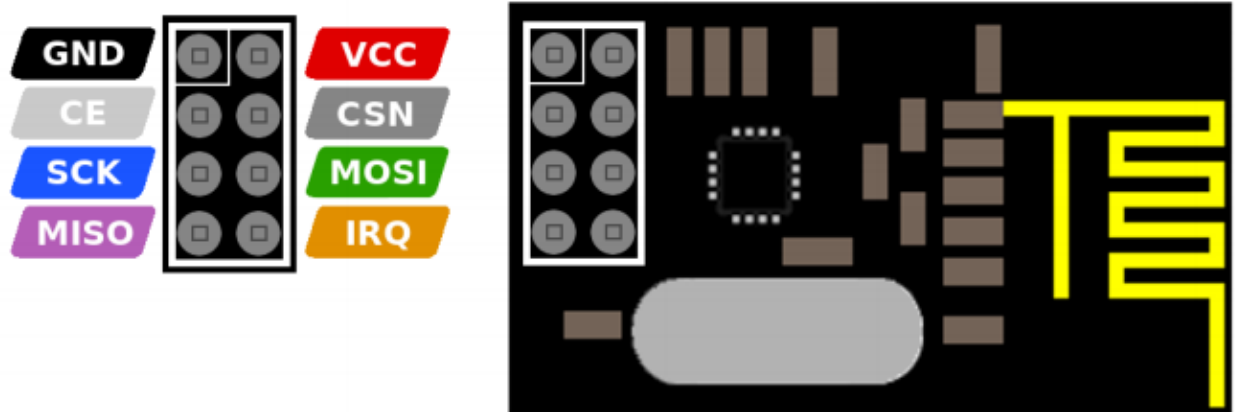


Figura 3-6 esquema de pines del NRF24L01 [6]

- GND es el pin de conexión a tierra.
- VCC aporta la alimentación necesaria al módulo, pudiéndolo conectar a una salida de 3,3V de un microcontrolador.
- CE (Chip Enable)
- CSN (Chip Select Not)
- SCK (Serial Clock)
- MOSI (Master Out Slave In) es la entrada SPI al NRF24L01.
- MISO (Master In Slave Out) es la salida SPI del NRF24L01.
- IRQ es un pin de interrupción que alerta de nueva información pendiente de procesar, este pin no es necesario conectarlo para un uso básico del módulo.

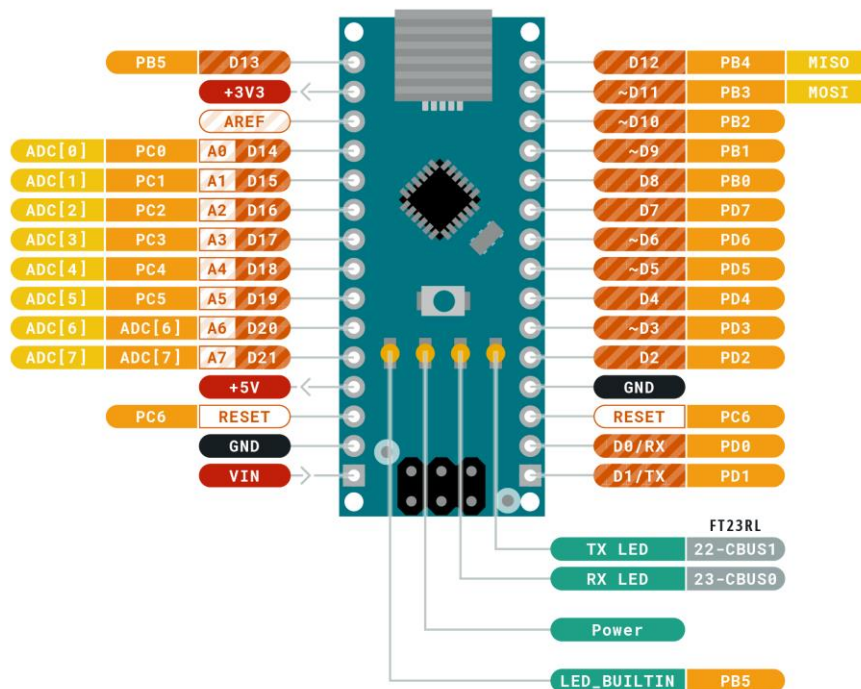
El precio aproximado del módulo se sitúa en torno a los 1.83€ por unidad pudiendo variar por el proveedor y el número de módulos que se compran.

3.1.1.4 Arduino Nano

Microcontrolador basado en ATMMega328, que se caracteriza por una buena flexibilidad con un reducido tamaño. Está preparado para una alimentación entre 3V y 5V y posee 16 pines de salida, que siguen la distribución de la Figura 3-5.



ARDUINO NANO



Ground	Internal Pin	Digital Pin	Microcontroller's Port
Power	SWD Pin	Analog Pin	
LED	Other Pin	Default	



This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Figura 3-7 Pinout de un Arduino NANO, (tomada de [15])

El Arduino presenta una memoria *flash* de 32KB, una velocidad de reloj de 16Mhz, un tamaño de 18x45mm y un peso de 7g por lo que lo hace ideal para nuestra aplicación. Además de que su precio de mercado oscila entre 1€- 2€.

3.1.1.5 ESP 8226

Este módulo es una plataforma Iot (*internet of things*) con código abierto y de bajo coste con un precio medio de 4€ en el mercado. Este microcontrolador posee capacidad de conexión Wi-Fi de 2,4Hz mediante IEEE 802.11 bgn. El módulo presenta una pila TCP/IP completa lo que le permite la lectura, control y procesamiento de datos. Esto le confiere capacidad de soportar los protocolos de IoT MQTT. Algunas de sus características principales son:

- Procesador de 32 bits
- Velocidad de 80MHz (máximo de 160MHz)
- 32 KBits RAM
- 16pins GPIO
- PWM en todos los pines
- Conversor de analógico a digital de 10 bits
- Voltaje de operación (3-3,6) V

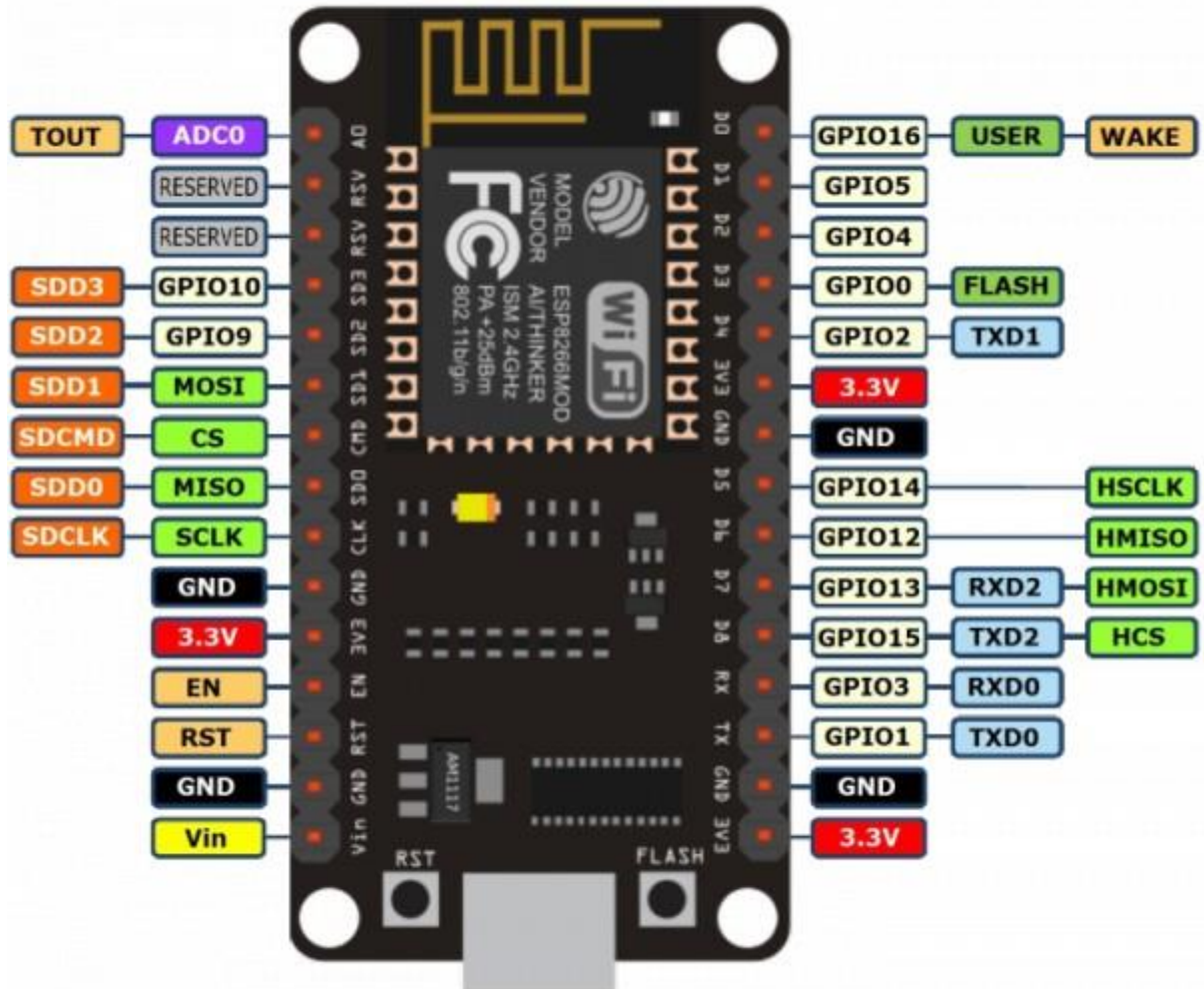


Figura 3-8 Módulo ESP 8266

Estas características lo convierten en una posible solución para el control de los distintos módulos necesarios para la elaboración del prototipo. Además, este microcontrolador da capacidad, no solo de control, sino que además permite realizar comunicación con un servidor común mediante tecnología MQTT.

3.1.1.6 ESP 32

El ESP32 es un chip heredero del ESP 8266 donde el principal cambio es un módulo de 4,2 BLE (Bluetooth Low Energy) incorporado. Las características más importantes de este Chip según el manual del fabricante [7]

- Programación y fuente de alimentación por cable USB-B
- Procesador ESP-WROOM-32.

- W-lan 802.11 b/g/n.
- Bluetooth 4.2 / BLE.
- 160MHz Tensilica L108 32 bits CPU de doble núcleo.
- 512 KB SRAM y 16 MB de memoria flash.
- 32 pines de E/S digitales (3,3V).
- 6 pines analógicos.
- 3xUART, 2xSPI, 2xI2C
- CP2102 Interfaz USB a UART
- Programable a través de Arduino Code, Lua, MicroPython, etc.
- Precio: 8€.

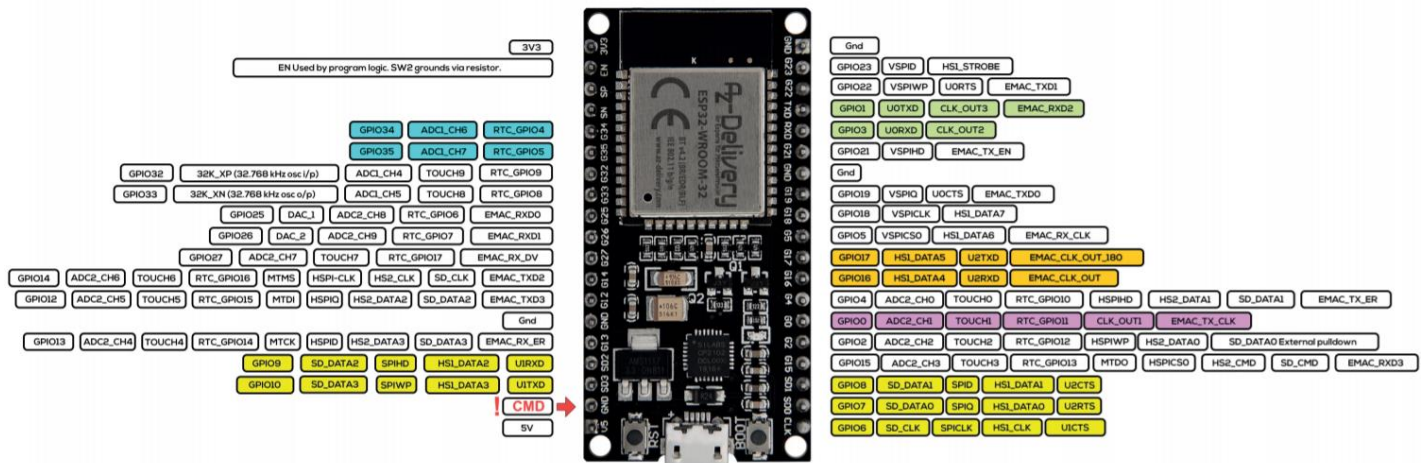


Figura 3-9 esquema de pines del ESP32 (tomada de [14])

La ventaja que presenta este Chip frente a otros es su capacidad de conexión Bluetooth y Wi-Fi simultáneas lo que nos permite enviar la información a un servidor a tiempo real sin necesidad de usar otro servidor adicional. Esto dota al sistema de una alta flexibilidad por no precisar de cables para volcar la información, evitando el uso de módulos específicos para realizar dicha comunicación.

3.1.2 Selección de Software

Una vez seleccionado el Hardware a emplear es necesario determinar que software permite realizar las operaciones deseadas de la forma más eficaz. Para ello primero se precisa controlar los Chips, donde se seleccionó el IDE de Arduino. A continuación, se precisa de un bróker (servidor) tipo mqtt-mosquito que realice la comunicación entre los chips y una aplicación. Por último, cubrir la necesidad del tratamiento y presentación de datos, para lo cual se usó Python con el entorno de programación ANACONDA.

3.1.2.1 IED Arduino



**Figura 3-10 Logotipo
Arduino (tomada de [15])**

Para el control de los microcontroladores se empleó la plataforma Arduino. Esta se pensó inicialmente como una plataforma de aprendizaje en el uso de microcontroladores, sin embargo, hoy en día es tan extenso su uso que se emplea desde aplicaciones educativas, como en el desarrollo de productos de índole científica pasando por aplicaciones IOT (internet de las cosas).

Sin lugar a duda las ventajas más importantes de esta aplicación son la de disponer de una plataforma económica sin coste para el desarrollador o el usuario. Así como la de disponer de un sistema multiplataforma con una librería abierta y extensible, que permiten a la comunidad ir actualizando en función de los nuevos sensores y dispositivos que van apareciendo en el mercado. Además, se trata de un entorno de programación multiplataforma, lo que permite emplearlo en distintos sistemas operativos, desde Linux hasta MAC pasando por Windows, sin que presente problemas de compatibilidad al emplear

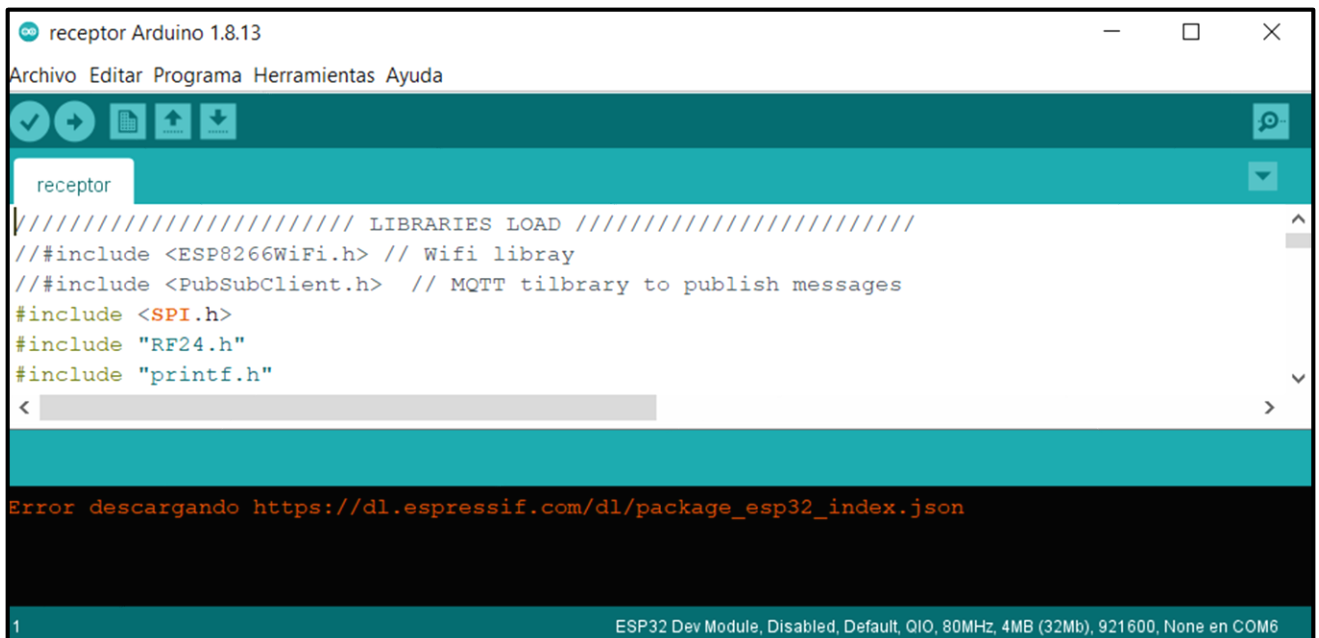


Figura 3-11 Entorno de programación Arduino (tomada del entorno directamente)

programas en uno u otro sistema indistintamente.

Su lenguaje de programación se basa en C/Cpp. Además, su IDE detecta de forma automática las placas de Arduino y su entorno de programación permite la compilación sin necesidad de descargar

librerías extras. Sin embargo, presenta un administrador de descargas que nos permite acceder a todas las creaciones de la comunidad dotando así de una gran flexibilidad a la plataforma, y permitiendo el uso de esta en placas que no pertenezcan a la familia de Arduino. A mayores dispone de un monitor serie para poder comprobar el funcionamiento correcto de los programas creados. Todo esto lo hace de un entorno de programación ideal para el uso y desarrollo de microcontroladores como el ESP32, empleado en este proyecto.

3.1.2.2 *Mosquito mqtt*



Figura 3-12 Logotipo MQTT (tomada de [3])

Una vez seleccionado el entorno para programar los Chips ESP32 surge el siguiente escalón del sistema. Se necesita un protocolo que nos permita volcar la información obtenida por los chips de forma independiente a una base de datos común, desde la que un programa sea capaz de acceder y administrarlos. Es aquí donde entra en juego mqtt.

En palabras de Llamas en 2019, *“MQTT son las siglas MQ Telemetry Transport. Es un **protocolo de comunicación M2M (machine-to-machine) de tipo message queue**. Está basado en la pila **TCP/IP** como base para la comunicación. En el caso de MQTT cada conexión se mantiene abierta y se “reutiliza” en cada comunicación.”* [3].

MQTT es un sistema de comunicación mediante mensajería “push” que se estructura en un sistema publicador /suscriptor. En definitiva, desde un servidor o “Broker” se reciben los mensajes de todos los publicadores y a su vez se distribuyen a los distintos suscriptores. Para controlar la distribución de mensajería y que esta sea lo más eficiente posible se puede establecer un sistema de temas mediante el cual solo pueden acceder los equipos que han recibido permiso por el “Broker”. Este funcionamiento dinámico ha convertido a mqtt en uno de los pilares [3] fundamentales en el IOT permitiendo a que diversos sensores y actuadores se conecten de manera simultánea a un servidor central.

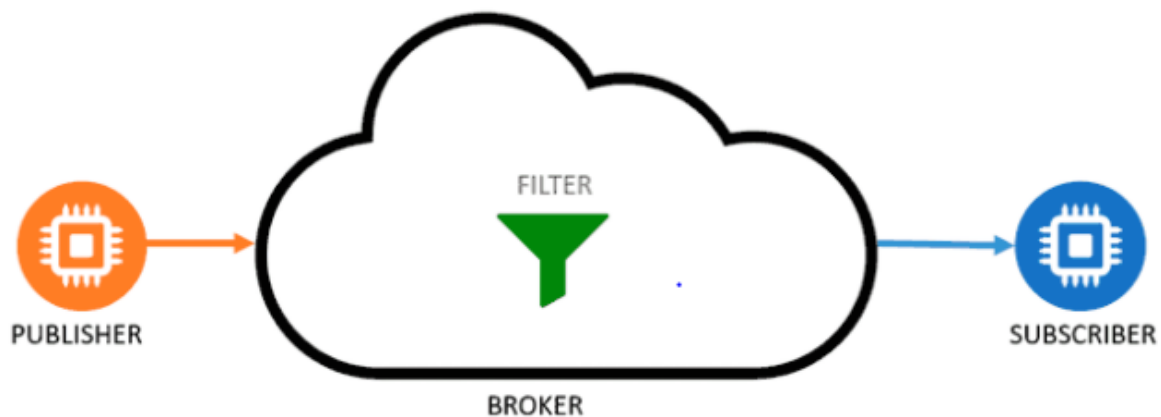


Figura 3-13 Estructura de mqtt (tomada de [3]).

El funcionamiento del protocolo es muy simple donde se distinguen 2 formas de conexión. Una para el publicador, donde el cliente envía un mensaje “CONNECT” y el servidor responde con un mensaje ACK que contiene el resultado de la conexión (aceptada o rechazada). A partir de ahí la conexión se mantiene abierta hasta que finalice el envío de información. En el caso del suscriptor el funcionamiento es idéntico a excepción del mensaje inicial que en este caso es una petición “SUBSCRIBE”. Para asegurar el mantenimiento de conexión se realizan de forma periódica protocolos tipo “PING” donde se envían mensajes sin información para ver el tiempo de transmisión de un punto a otro. Finalmente, para cerrar la conexión los clientes envían un mensaje “DISCONNECT”.

3.1.2.3 Anaconda

Finalmente, para presentar y guardar los datos obtenidos se seleccionó el entorno de programación de Anaconda basado en Python QT. Este tipo de entornos con un elevado número de librerías lo que permite una programación fácil y sencilla mediante un sistema por bloques.

Algunas de las ventajas de esta plataforma es que corre sobre programación Python-QT uno de los lenguajes de programación más comunes en la actualidad. Además, permite la presentación de grafica de datos lo que dará una lectura de datos más interactiva. Como ventaja añadida y relacionada con el precio del prototipo se presenta que Anaconda es una distribución libre y con código abierto, es decir cualquier usuario tiene permisos de administrador, pudiendo de esta manera acceder al código fuente y siendo capaces de optimizar o modificar este. Este Entorno tiene usos principalmente en tratamiento de datos y *machine Learning*, es decir que los ordenadores sean capaces de aprender procesos mediante el entrenamiento repetitivo.

- **Python**

Python es el programa empleado para aunar y fusionar todos los elementos de Anaconda. Se caracteriza por presentar un lenguaje de programación interpretado, con una filosofía



Figura 3-14 Logotipo de Python (tomado de [16])

centrada en la legibilidad del código, facilitando de esta manera al programador como a un futuro usuario entender el funcionamiento de este. A mayores el lenguaje es multiparadigma soportando programación orientada a objetos, programación funcional e incluso interpretativa.

Su éxito a nivel mundial dentro del mundo de la programación se debe a que se trata de un lenguaje amigable siendo sencillo, bien estructurado y legible. Además, permite la elaboración de complejos algoritmos en un número reducido de líneas de código, y su extensa comunidad, así como la información colgada en la red permite una programación con un número relativamente reducido de conocimientos acudiendo para compensar esta falta al elevado número de documentación y foros.

A mayores El entorno dispone de frameworks gráficos que se pueden explotar para obtener representaciones en pantalla todos los datos enviados por los distintos sensores.

- **PyQtGraph**

Con el fin de poder representar los resultados obtenidos, así como poder realizar graficas que permitan comparar estos resultados y evaluar la calidad de los experimentos se recurrirá a la librería de PyQtGraph. Esta librería está constituida sobre NumPy, desarrollada para aplicaciones científico-matemáticas, lo que le confiere una gran capacidad para el tratamiento de un elevado número de datos como en este caso. PyQtGraph se distribuye bajo la licencia de código abierto del MIT.

3.2 Creación del proyecto

La opción seleccionada ha sido la de un sistema token asistido, en el cual se usan señales de ultrasonido para medir los tiempos de vuelo, señales de radiofrecuencia para sincronizar los transmisores con los receptores, y señales Wi-Fi para cargar los datos recogidos en un servidor. Posteriormente se accederá desde un PC para realizar el análisis, procesado de información y presentación de resultados en tiempo real.

3.2.1 Creación de bases emisoras y receptoras

Para la creación de ambos sistemas se ha empleado el sistema Protoboard el cual permite la creación rápida de prototipos para la experimentación de circuitos, dotando de flexibilidad al usuario y permitiendo realizar tantas modificaciones como sea necesarias, hasta que se alcance un resultado definitivo que pueda ser definitivamente llevado a un sistema integrado.

3.2.1.1 Base emisora

Para la confección de la base emisora se emplearon 4 sensores de ultrasonidos HC-SR04 alimentados con 5V, un módulo de RF NRF24L01 alimentado con 3,3V y un chip controlador Arduino Nano con salidas de alimentación de 3,3V y 5V. El sistema es alimentado la base mediante un puerto USB mini B con una pila de Litio de 3,7V situada en una porta pilas de adaptación de voltaje con salida de 3,3V.

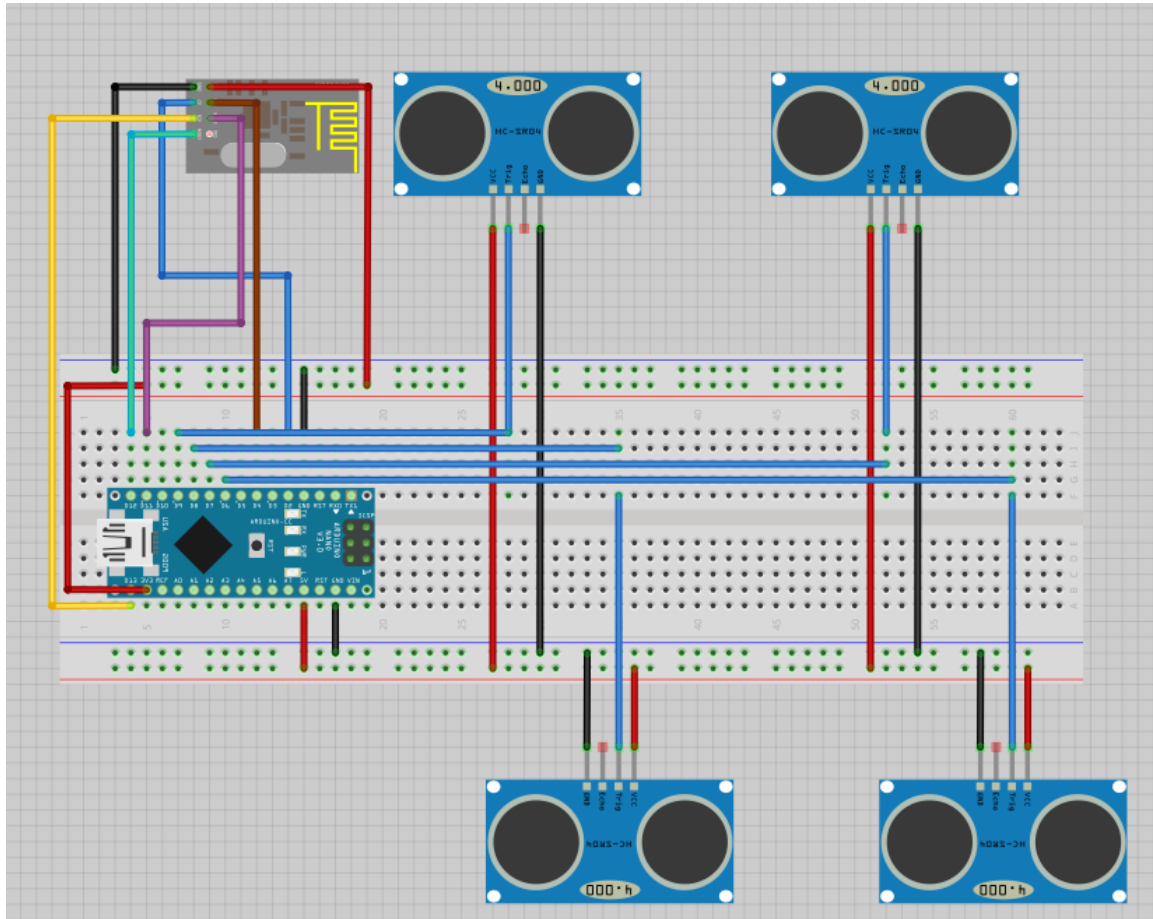


Figura 3-15 Esquema TX mediante Fritzing (elaboración propia).

Tras el montaje de los sistemas según el esquema de la Figura 3-12 se integraron todos los componentes en una plataforma de Etinilvinilavetato, comúnmente conocido como Goma Eva, para que hiciera de soporte rígido y fácilmente transportable. Permitiendo de esta forma, que sea posible realizar experimentos en distintos entornos de trabajo adaptando las plataformas emisoras y receptoras creadas.

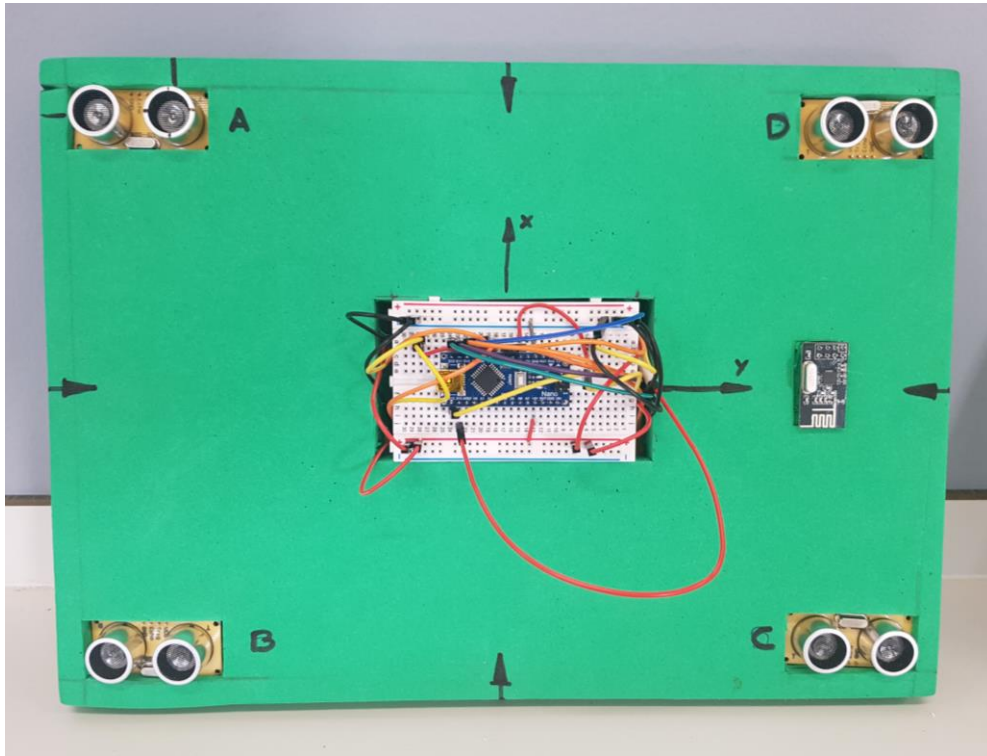


Figura 3-17 Prototipo Base TX con US-026

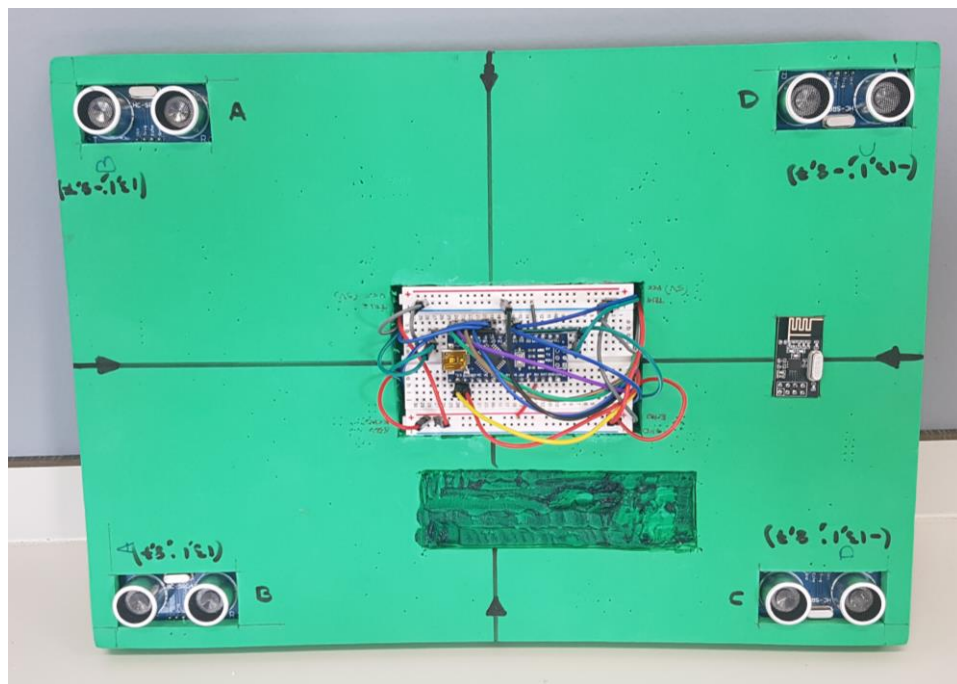


Figura 3-16 Prototipo Base TX con HC-SR04

El montaje final puede verse en la Figura 3-1, en donde la posición de los cuatros sensores que trabajan como emisores se situaron en las esquinas y repartidos de forma simétrica.

El primer prototipo desarrollado consta de 4 ultrasonidos que irán transmitiendo de forma secuencial con una indicación previa, antes del envío de cada pulso de ultrasonidos por parte de la antena de RF, de forma que este mensaje actúa como sistema de sincronización entre los emisores y los receptores. Este montaje inicial podrá ser modificado añadiendo más ultrasonidos para aumentar la información a costa de reducir nuestro ancho de banda, debido a la ejecución secuencial no simultanea de la emisión de pulsos de ultrasonidos. Una posible alternativa futura para evitar esta pérdida de ancho de banda sería la realización del disparo simultáneo de todos los emisores, pero las interferencias entre señales de los ultrasonidos emitidos podría provocar que se precisen otro tipo de técnicas para estimar un modelo aceptable.

3.2.1.2 Base receptora

En el caso de la base receptora se realizaron tres montajes distintos. Para dos de ellos se empleó un único sensor de ultrasonido HC-SR04/US-026 a 5V, al cual se le inutilizó el transmisor, mediante una tapa, para que solo fuera capaz de recibir información. Para poder comunicarlo con el ESP-32 se insertó un adaptador de corriente entre 5V y 3,3V. Además, se empleó un módulo de radiofrecuencia NRF24L01, que recibe el aviso de emisión por parte de la base, y un microcontrolador ESP32 alimentado con 3,3V y con antena WI-FI para que se pueda conectar mediante esta vía a un servidor MQTT que reciben los datos obtenidos por la recepción de las señales de ultrasonidos. Por último, el sistema ira alimentado por una pila de 3,7V sobre una porta pilas de adaptación de voltaje a 3,3V. El esquema de montaje seguido es el que se puede ver en la Figura 3-16.

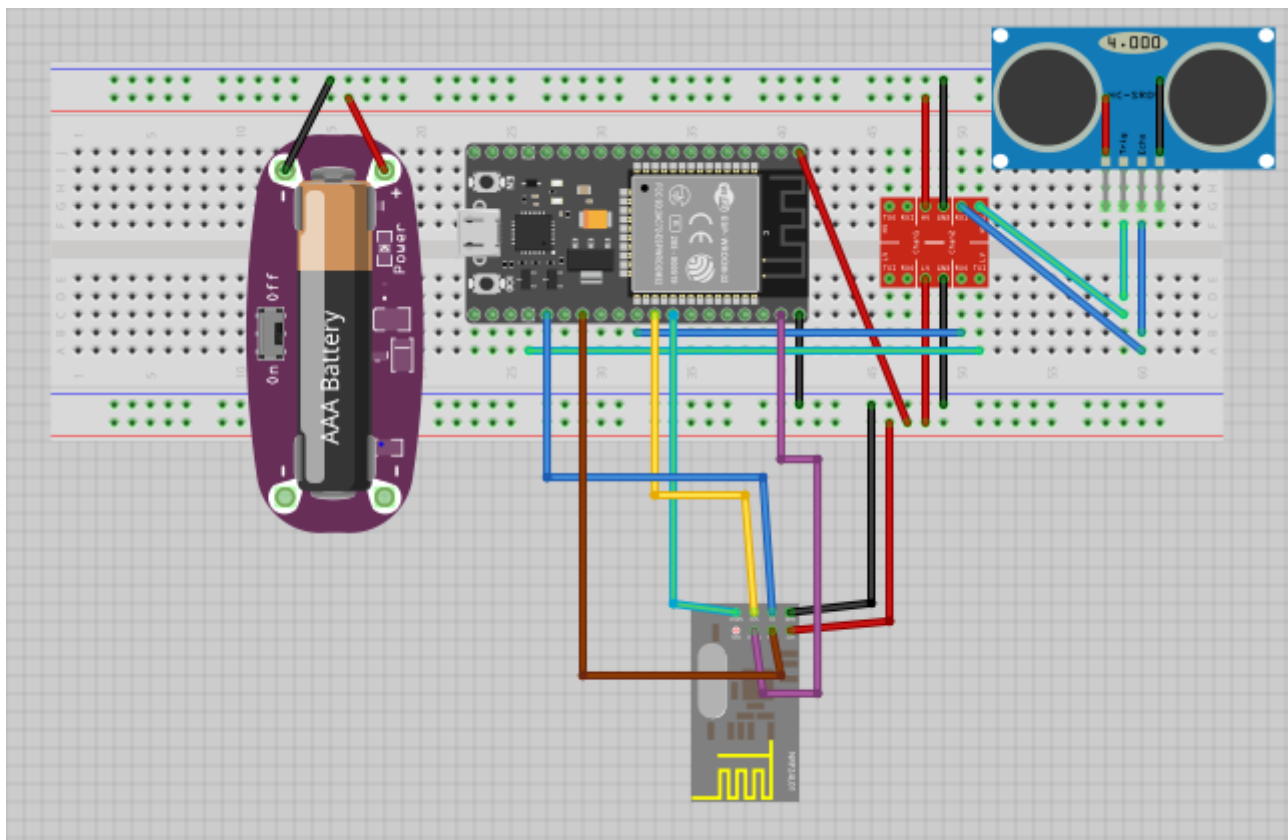


Figura 3-18 Esquema RX con ESP32 mediante Fritzing (elaboración propia).

El tercer montaje que se realizó siguió un esquema muy similar, donde se empleó un US-026 alimentado a 3,3V y controlado por un ESP-8266. De esta forma no se precisa el adaptador de nivel de 5 a 3,3 simplificando el montaje, aunque según especificaciones introduciendo más errores en las mediciones. En la **¡Error! No se encuentra el origen de la referencia.** se puede observar el montaje realizado.

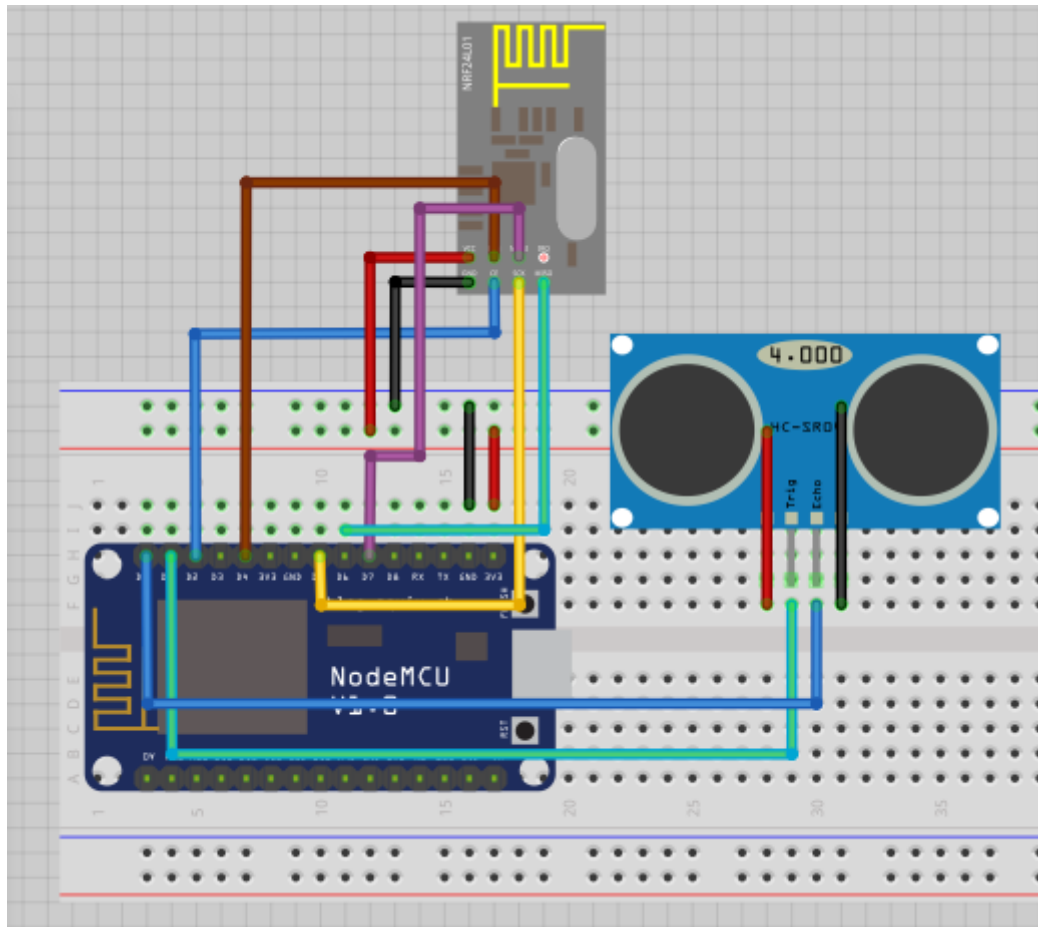


Figura 3-19 Esquema RX con ESP8266 mediante Fritzing (elaboración propia)

Todas las instalaciones se realizaron sobre una Protoboard, dotando de capacidad al sistema de sufrir modificaciones durante el desarrollo del proyecto, así como poder colocarlo en objetos estáticos o dinámicos. Permitiendo de esta forma que el sistema presente la flexibilidad necesaria para los experimentos de calibración. El resultado final es el que se puede observar en la Figura 3-21.

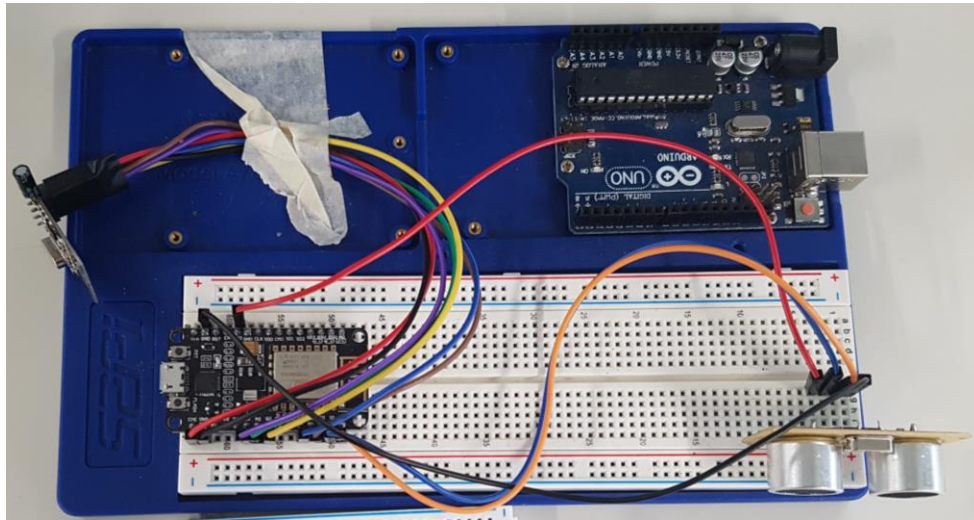


Figura 3-21 Imagen tomada del prototipo realizado en Protoboard con microcontrolador ESP8266

Debemos tener en cuenta que el objetivo es emplear varios receptores independientes ligados a un objeto para realizar la estimación de su posición y orientación en dimensiones. Debido a esto, se crearon 4 bases receptoras que se situaran en los puntos más relevantes del objeto cuya posición se desea determinar. En un futuro se pretenderá reducir el tamaño del montaje usando circuitería soldada, así como un receptor independiente de ultrasonidos sin emplear el módulo completo del HC-SR04/US-026.

Con el fin de poseer una estación de pruebas dinámica se realizó un montaje en poliestireno expandido (EPS) con 5 posiciones para los receptores. De esta forma la estación permite realizar experimentos de calibración estáticos, donde se puede modificar la posición de los receptores, así como el empleo simultaneo de los mismo. Este montaje nos permite el desarrollo de diferentes pruebas

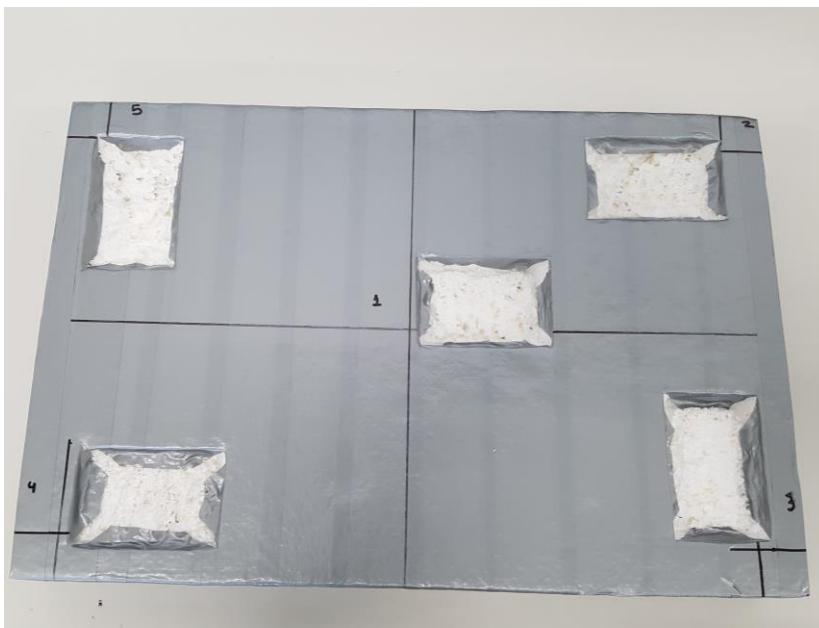


Figura 3-22 Estación de pruebas (elaboración propia)

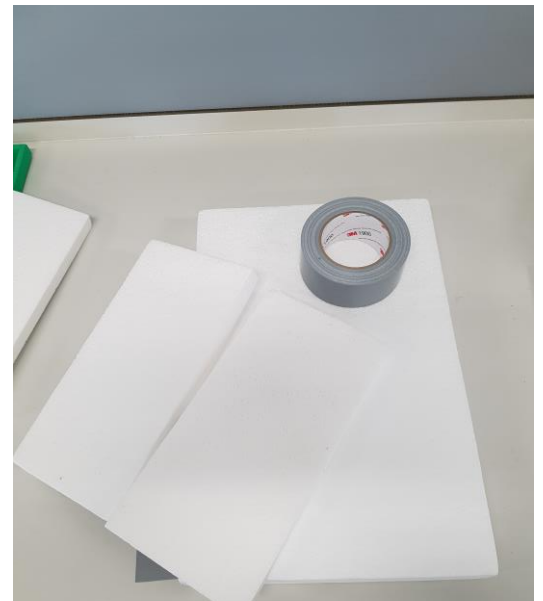


Figura 3-23 Material empleado (elaboración propia)

mediante las cuales poder calibrar y comprobar la eficacia inicial de los sensores desarrollados, así como la respuesta del servidor a un volumen elevado de datos.

Una vez confeccionado la estación como se puede ver en la imagen se crearon 5 posiciones para la Protoboard quedando el receptor del módulo de ultrasonidos sobre la cruz de cada posición. Así la estación permite generar experimentos conociendo la posición exacta del receptor generando el “*Ground truth*” o cero conocidos con el que poder comparar los resultados de la calibración y permitiendo determinar el tipo de ruido y listar los posibles filtros más apropiados a aplicar. En la Figura 3-20 se puede observar cómo quedarían las estaciones RX una vez colocadas par los experimentos.

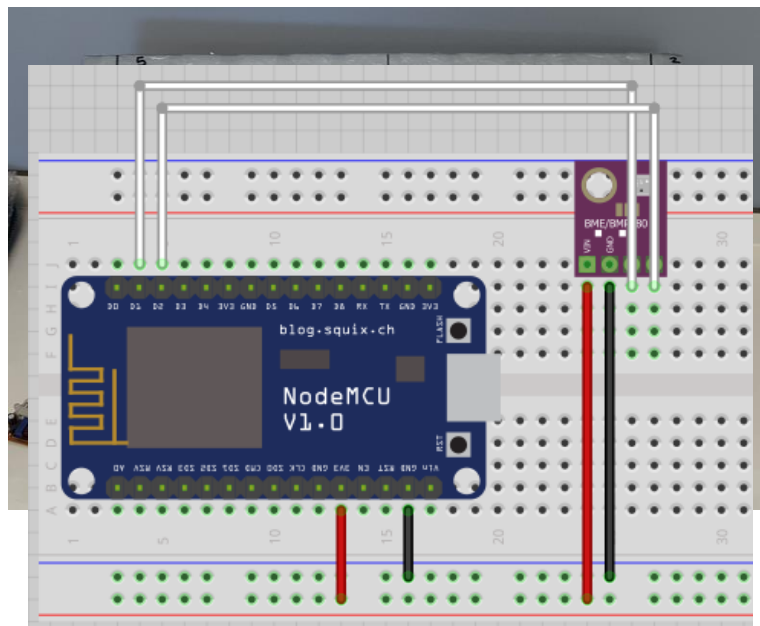


Figura 3-25 Esquema de estación meteorológica mediante Fritzing (elaboración propia)

3.2.1.3 Estación meteorológica

Por último, se desarrolló una pequeña estación meteorológica con el fin de poder tomar datos de temperatura presión y altitud. Para ello se empleó un Chip ESP8266 como controlador. Para la toma de datos se usó un ... creando así un sistema muy simple con capacidad de conexión a Wi-Fi y así poder descargar los datos obtenidos en nuestro servidor mqtt. Guardando de esta manera no solo la información de las estaciones receptoras y emisoras, sino que, en qué condiciones de presión y temperatura se han obtenido los resultados. A continuación, en Figura 3-25 se puede apreciar el esquema de montaje seguido.

3.2.2 Desarrollo del Software

A continuación, se expondrá de nivel más bajo a más alto los distintos programas desarrollados para la realización de las pruebas de calibración, así como para la toma de datos.

3.2.2.1 Arduino

El primer código que se desarrollo fue mediante el IDE de Arduino. Fue empleado para generar tres programas distintos. Con el fin de controlar los distintos chips, y que estos pudieran funcionar de forma independiente, así como cargar todos los datos obtenidos en un servidor mqtt.

Por un lado, se puede identificar el Código referido al TX, el cual fue preparado para los Arduino Nano. En él se generó un sistema de bucle tipo *for* que permitiera ir cambiando de forma secuencial entre los 4 pins Trigger de los módulos HC-SR04. De esta forma en cada vuelta del bucle se lanza un

mensaje RF indicando que sensor va a emitir y a continuación se envía un pulso de ultrasonido. Inicialmente no se tuvo en cuenta que la estación pudiera ser controlada de forma remota mediante el servidor inicial, sino que se inicia mediante conexión de la batería. En este momento el programa desarrollado realiza una primera función de *setup()* donde se abre la conexión mqtt con el *broker*, así como se comprueba que el hardware que emplea no presenta problemas y permiten la comunicación entre ellos con el Arduino Nano. A continuación, se reproduce la función *loop()* de manera cíclica

A continuación, se desarrolló el programa que controla las estaciones RX. En esta ocasión se planteó un funcionamiento autónomo de cada pareja HC-SR04/US-026 y NRF24L01, con el fin de que no haya interferencia entre las señales recibidas. De esta manera cada punto del que se pretende obtener la posición ira emitiendo los datos recibidos de forma autónoma a un servidor común, indicando de que estación recibe el pulso, quien es, y cuanto tiempo ha tardado en llegar dicho pulso. La sincronización de las estaciones TX y RX se realizará mediante la señal de RF midiendo la diferencia de tiempo entre la llegada de la señal de radio frecuencia con la señal de ultrasonido. Para ello el chip esperara a una señal de radio frecuencia que le indique que TX va a emitir. En ese momento empezara a escuchar el canal “echo” del ultrasonido. Una vez que el pulso de ultrasonido llegue el chip enviara un mensaje por mqtt donde indicara que estación TX ha enviado el pulso, quien es dentro de los RX de la red y cuánto tiempo ha tardado entre la RF y el ultrasonido.

Por último, se generó un pequeño programa para controlar la estación meteorológica obteniendo datos de temperatura, presión, y altitud con el fin de poder tener unos valores que definan el medio. Teniendo en cuenta estas variaciones se permite calcular con mayor precisión la velocidad del sonido. Esto permitirá normalizar los distintos experimentos.

3.2.2.2 MQTT

Para el funcionamiento del servidor mqtt se precisó conectarse tanto desde los sistemas de Arduino que funcionaran como publicadores como desde el programa generado con Python que realizara las funciones de lectura.

Para ello se lanza el servidor mqtt desde un ordenador PC o portátil. Para poder establecer la comunicación entre todos los elementos se necesita que se encuentren todos en la misma red Wi-Fi. Una vez abierta la red se necesita programar en los distintos programas que van a acceder a que IP van a enviar o solicitar la información necesaria.

Una vez iniciado el servidor la información se ira guardando en distintos *topics* o *subtopics* indicados por los distintos publicadores. Los suscriptores entonces solicitarán acceder a esos *topics* o *subtopics* donde leerán la información que se encuentra. En este caso se generó un *topic* /TFG del que cuelgan dos “*subtopics*” uno /TFG/ULTRASOUND, destinado a almacenar la información de las distintas estaciones, y el otro TFG/PROTOCOL, que, aunque inicialmente para las tareas de calibración no se usan deja una vía abierta para poder controlar los sensores desde un terminal.

3.2.2.3 Python

Por último, con el fin de controlar los experimentos, así como presentar la información recibida se crearon dos programas tipo “.py”. uno con el fin de generar archivos “.csv”. Con la información recibida y otro encargado de coger dichos ficheros y presentarlos por pantalla mediante gráficas, pudiendo así comprobar la validez de los resultados. Para realizar estas evaluaciones se guarda en los ficheros .csv el conocido como *rawdata*, o información sin procesar. De esta manera se recogen los datos de:

- Estación emisora
- Estación receptora
- Tiempo de pulso

- Tiempo de llegada del mensaje

4 RESULTADOS / VALIDACIÓN / PRUEBA

Durante este tema pasaremos a introducir las pruebas realizadas con el sistema de medición de distancias mediante ultrasonidos que ha sido construido. Para ello se ha montado un set sencillo en donde se sitúa la base emisora en una posición fija, y en donde los receptores son situados o desplazados según el tipo de prueba a realizar.

Comenzaremos por introducir los experimentos relativos a la calibración de los parámetros que nos permitan calcular distancias a partir de los tiempos de pulso obtenidos en los receptores de ultrasonidos. A continuación, interesa realizar el análisis de la sensibilidad y precisión obtenidas a la hora de situar en 3 dimensiones la posición de un punto asociado al receptor de ultrasonidos, y por último pasaremos a aplicar este sistema para realizar la estimación del ángulo de *roll* de un objeto al que se adhieren dos receptores en posiciones fijas establecidas.

4.1 Calibración

Para efectuar la calibración del sistema propuesto se han tomado una serie de medidas de un receptor situado en posiciones conocidas con respecto a las posiciones de los cuatro emisores situados en la plataforma base. Aunque el objetivo final de un estudio más profundo sería el de crear una gran base de datos empleando múltiples receptores y tomando medidas desde una gran cantidad de posiciones en el espacio 3D de trabajo, en este TFG el objetivo es, sobre todo, evaluar las posibilidades de este tipo de sistemas de bajo coste para la localización con precisión de objetos en 3D. Debido a esto, vamos a realizar la toma de datos en una pequeña serie de posiciones entre 50.0 cm y 110.0 cm (con intervalos de 10 cm) y posicionando el receptor en coordenadas ($x=0.0, y=0.0$), de forma que se desplaza únicamente en la coordenada z . Durante estas pruebas el receptor se sitúa apuntando a la posición central de la base,



Figura 4-1 Montaje del experimento de calibración

ya que, aunque sería interesante disponer de una base de datos que incluyese desviaciones angulares conocidas, esto requiere de disponer de ciertos recurso y tiempos fuera del alcance de un TFG.

Una vez obtenidos los datos de cada una de las posiciones fijas, el proceso para realizar la calibración es el siguiente:

- Se obtiene, para cada emisor, el valor medio y la desviación típica del valor de tiempo de pulso (en microsegundos) en cada prueba.
- A partir del análisis anterior se implementa un filtro doble:
 - En primer lugar, se añade un filtro de *outliers* que se encarga de eliminar valores anómalos en un sentido estadístico (los que superan el valor medio ± 2 std). Esto es necesario debido a que este tipo de dispositivos bajo coste tiene tendencia a generar valores fuera del rango de medida actual en ciertas ocasiones.

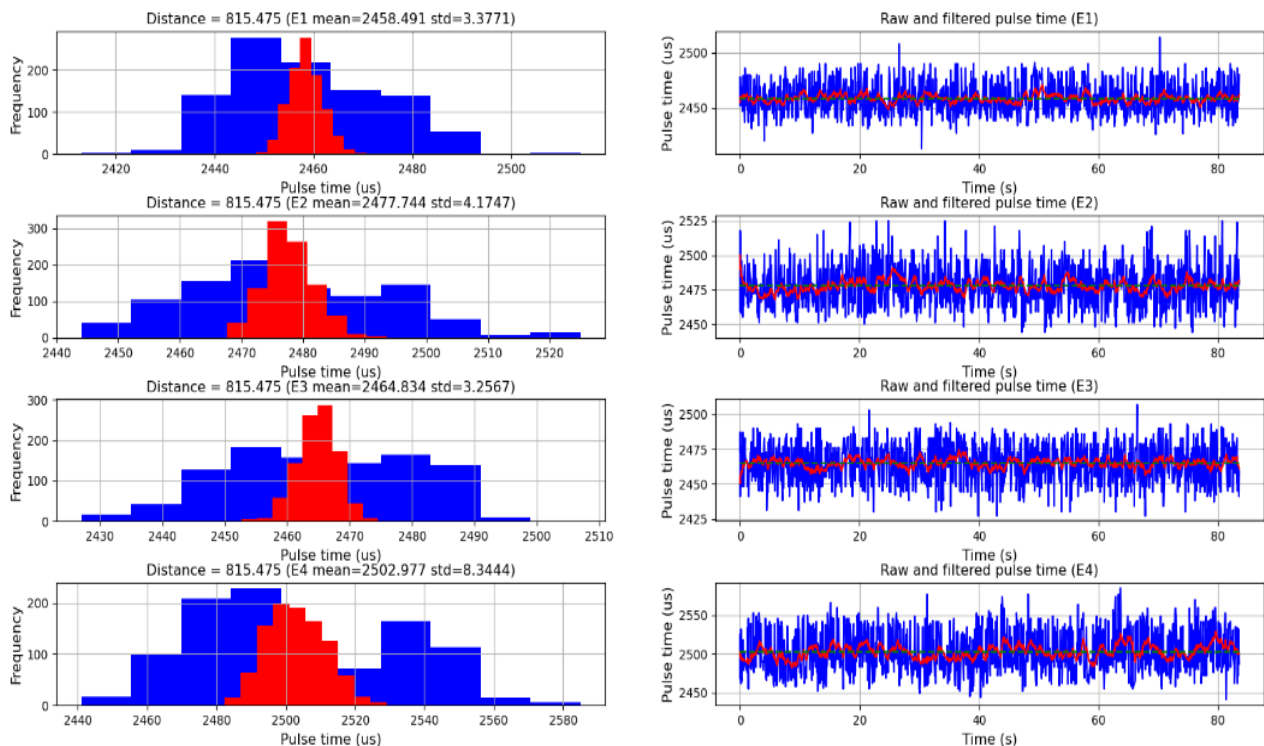


Figura 4-2 Tiempo de pulso sin filtrar y filtrado para $z = 500$ mm

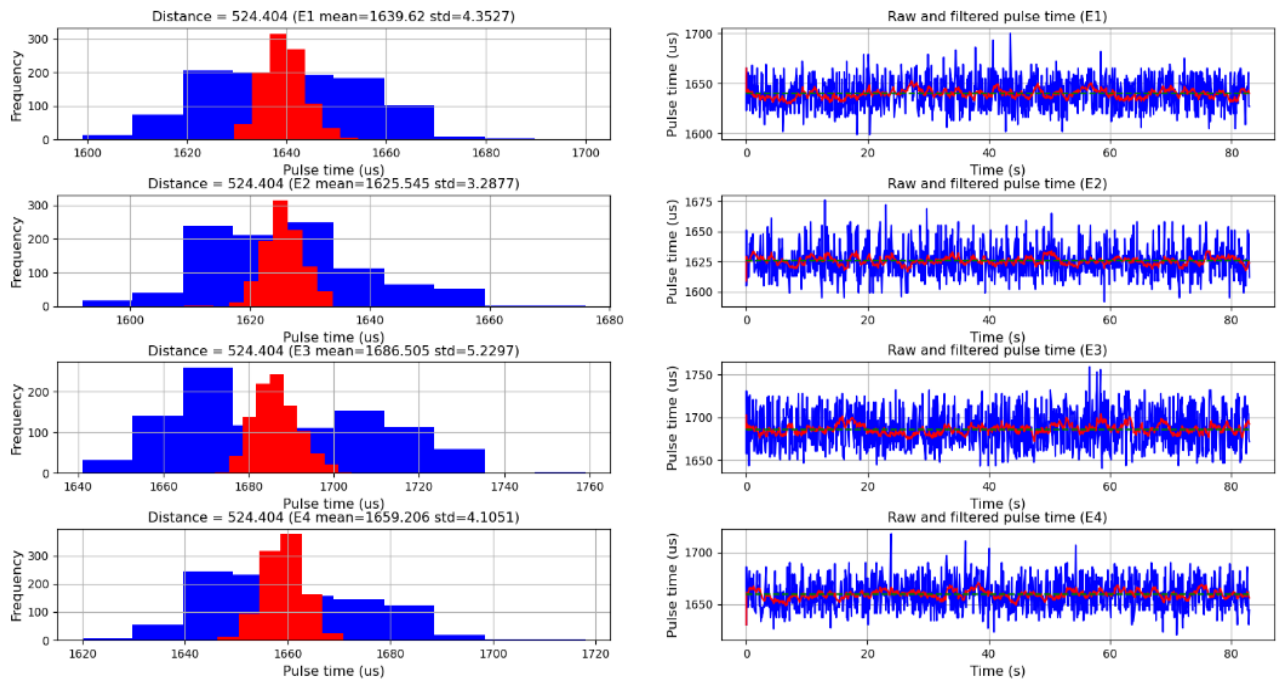


Figura 4-3 Tiempo de pulso sin filtrar y filtrado para $z = 800$ mm

A continuación, se vuelve a calcular el valor medio de las señales y la desviación típica. A partir de estos valores se aplica un filtro GH con valores $G=0.1$, $H= 0.02$ para disminuir los niveles de ruido y obtener un valor filtrado del tiempo de duración del pulso de ultrasonido. Las figuras 4.1 y 4.2 muestran los resultados obtenidos para las posiciones con valores $z = 500$ mm y $z = 800$ mm, donde podemos apreciar que el filtro propuesto permite reducir la presencia de ruido en los valores obtenidos y mantiene el valor medio que la señal genera.

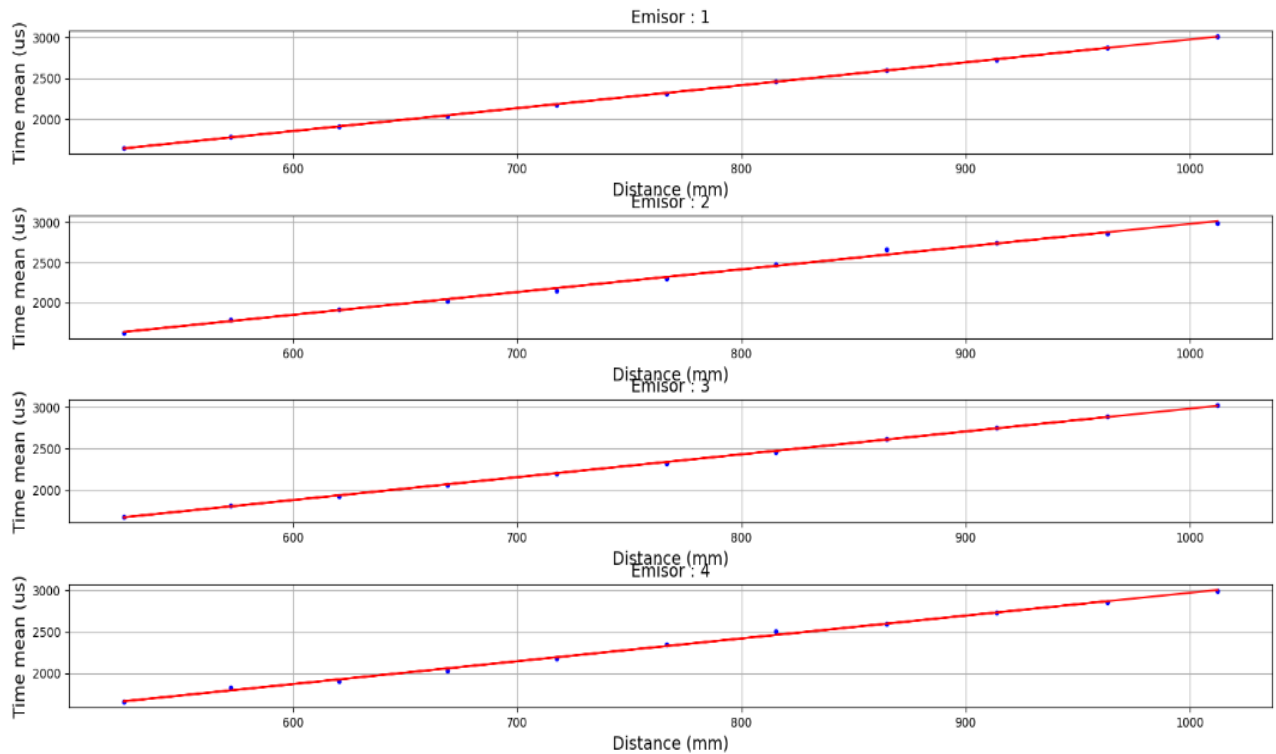


Figura 4-4 Resultados obtenidos mediante modelos de regresión lineal para cada sensor.

- El siguiente paso realizado en el cálculo de parámetros que permitan obtener la distancia a partir de las mediciones del tiempo de pulso. Debemos tener en cuenta que este tiempo no solo incluye el tiempo de vuelo de los pulsos de ultrasonido, sino también el tiempo de procesamiento en cada dispositivo receptor. Para realizar este cálculo se propone un modelo de regresión lineal para cada uno de los sensores, de forma que se obtienen los resultados que se muestran en la Figura 4.3, en donde podemos observar que la hipótesis lineal se ajuste con bastante exactitud los datos obtenidos.

Tabla 1 valores de la regresión para cada sensor

<i>Emisor</i>	<i>Slope</i>	<i>Bias</i>
1	2.8131440051608676	163.93372407514607
2	2.8260951730402732	154.21525379930108
3	2.748175662673981	234.14440993704193
4	2.7482768446032093	223.47234756128228

Empleando los valores de esta tabla es posible obtener el valor de la distancia, en mm, para cada sensor a partir de su *bias* y su *slope* mediante la siguiente fórmula lineal:

$$distance = \frac{pulse\ time - bias}{slope}$$

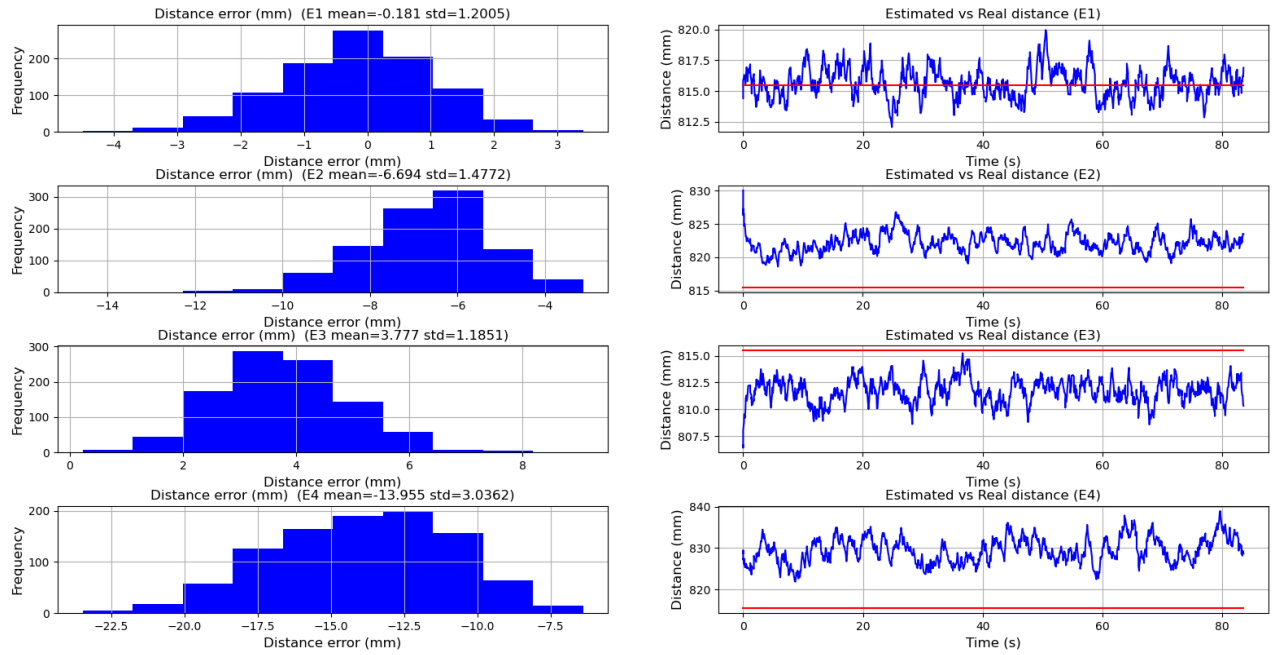


Figura 4-5 Distancia sin filtrar y filtrada para $z = 500$ mm

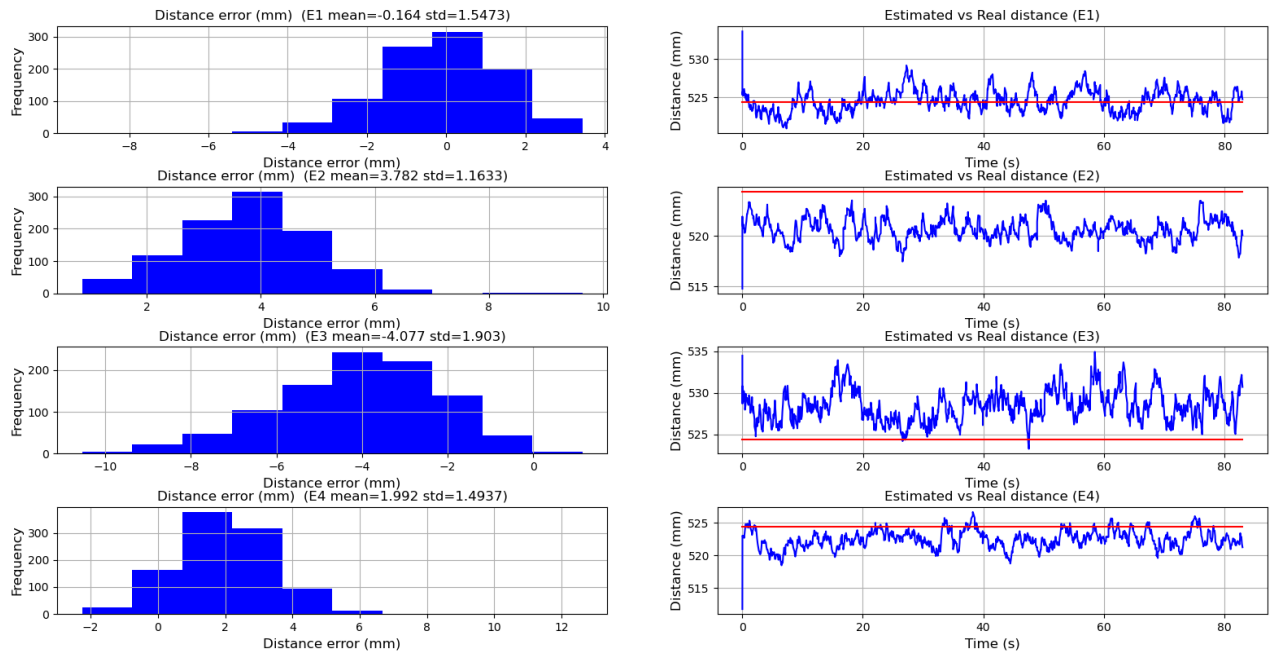


Figura 4-6 Distancia sin filtrar y filtrada para $z = 800$ mm

La Figura 4-5 y la Figura 4-6 muestran los resultados obtenidos, en las dos pruebas comentadas anteriormente ($z=500$ y $z = 800$), de las distancias estimadas a las que se encuentra cada emisor del receptor.

4.2 Estimación de la posición del punto en 3D

El siguiente paso consiste en aplicar los resultados obtenidos para realizar la estimación de la posición 3D del punto del receptor para los datos generados. Para ello se procede a aplicar el siguiente procedimiento:

- A medida que llegan un nuevo valor del tiempo de pulso enviado desde un emisor, se filtra empleando el filtrado de *outliers* y a continuación el filtro GH.
- A continuación, se recalcula la posición del punto en 3 dimensiones aplicando el algoritmo de *trilateration* introducido en el capítulo anterior cuando se aplica con 3 puntos de referencia. Debido a que se disponen de 4 emisores, y por lo tanto 4 distancias, para obtener la posición del punto del receptor, es posible aplicar diferentes tipos de algoritmos para resolver el problema de optimización asociado. En este trabajo vamos a aplicar un procedimiento sencillo, de forma que se obtiene la posición del receptor con respecto a cada una de las posibles combinaciones de 3 emisores (4 combinaciones en total) cada vez que se actualiza la información recibida desde uno de los emisores, y finalmente se calcula el valor medio de los resultados obtenidos.

El procedimiento propuesto se aplica a todos los datos obtenidos, de forma que los resultados obtenidos para los valores $z=500$, 700 y 1000 mm, se muestran en las Figuras 4-7, 4-8 y 4-9.

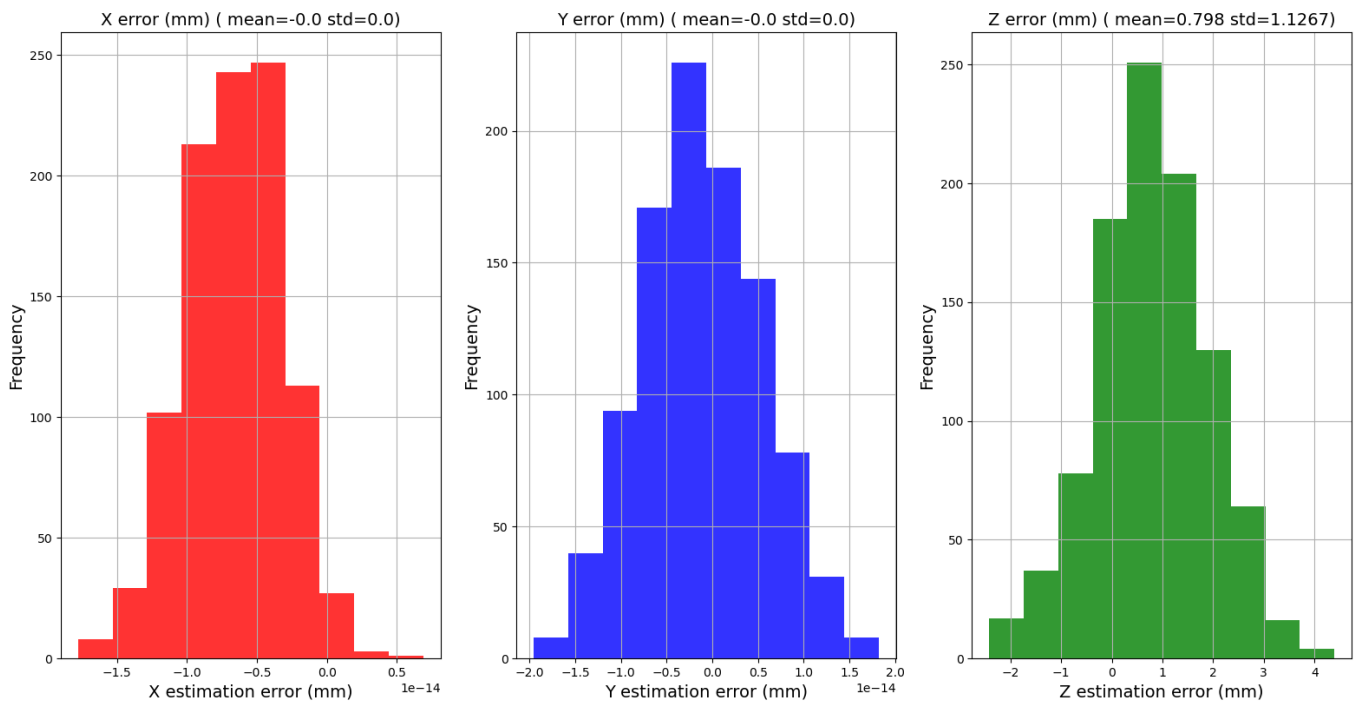


Figura 4-7 Error global en la estimación de las coordenadas 3D para $z = 500$ mm

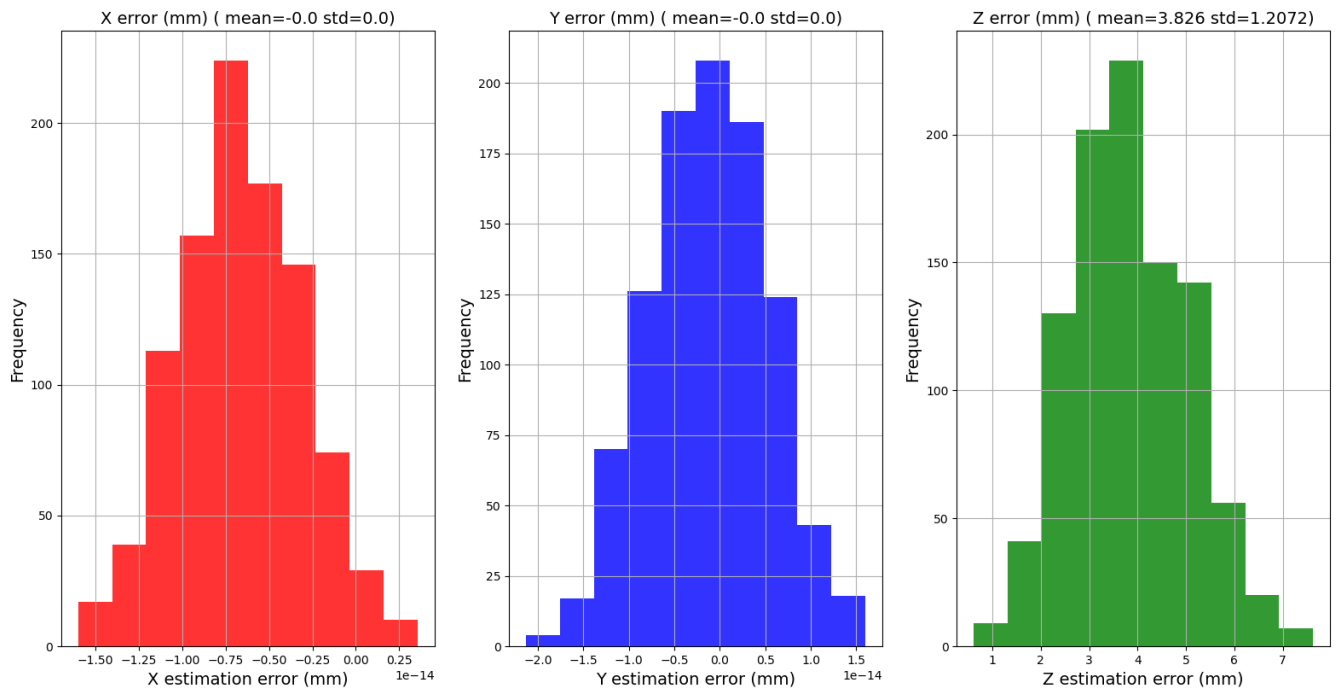


Figura 4-8 Error global en la estimación de las coordenadas 3D para $z = 750$ mm

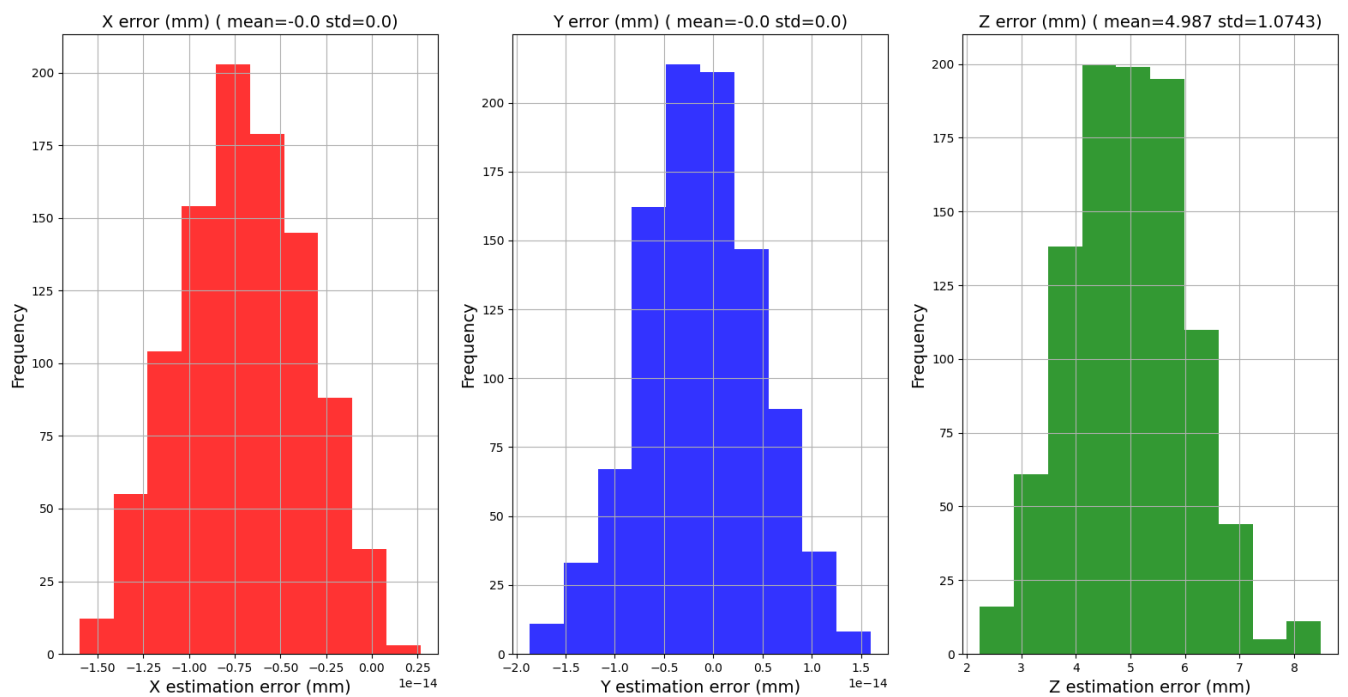


Figura 4-9 Error global en la estimación de las coordenadas 3D para $z = 1000$

Podemos observar que los valores medios y la desviación típica obtenidos en los ejes x e y son prácticamente nulos en todos los casos. En el caso de la coordenada Z el error es del orden de pocos milímetros y es posible apreciar que crece con la distancia, por lo que con suficientes datos podría ser posible aplicar un nuevo modelo de regresión que ayude a corregir la estimación de la coordenada Z, o incluso utilizar técnicas de *machine learning* para corregir esta pequeña desviación.

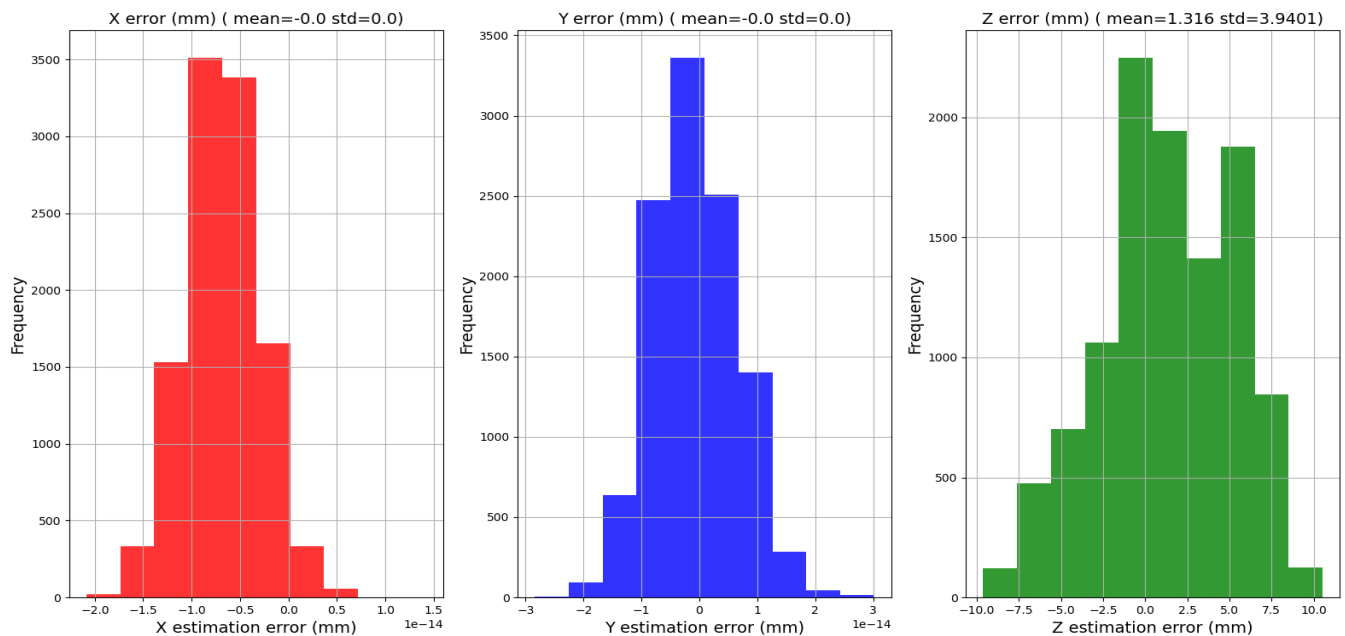


Figura 4-10 Error global en la estimación de las coordenadas 3D para todos los ejemplos

La Figura 4-10 Error global en la estimación de las coordenadas 3D para todos los ejemplos muestra el resultado final obtenido para todas las posiciones del receptor, en donde podemos apreciar que la tendencia mostrada en los ejemplos anteriores para posiciones específicas se mantiene en los resultados globales. Estos resultados iniciales son prometedores y muestran las posibilidades que este tipo de sistemas puede ofrecer, pero debemos tener en cuenta la necesidad de obtener más datos en un número mayor de situaciones, en las que la orientación del receptor pueda ser variable. Además, hay que tener en cuenta que se está evaluando el sistema en posiciones estáticas del receptor, por lo que para aplicarse en objetos en movimiento se necesitan realizar nuevos experimentos de validación que tengan en cuenta este tipo de situaciones.

4.3 Estimación de posicionamiento de un objeto en 3D

Tal y como se ha indicado en los objetivos del proyecto, el objetivo final del sistema creado es su evaluación para realizar estimaciones sobre las posición y orientación de un objeto que transporte varios receptores en posiciones determinadas. Para llevar a cabo este tipo de pruebas se procedió a realizar un montaje sobre el elemento denominado como **bloque en el (TFG11) Aplicación de una red de sensores inalámbricos distribuidos de bajo coste para el análisis de las propiedades dinámicas en 3 dimensiones de barcos en posiciones estacionarias** [8], presentado este mismo año en el CUD por parte de mi compañero Rafael Montero de Espinosa Ales. Este **bloque** dispone de 4 unidades inerciales que han sido instaladas para realizar estimaciones del ángulo de *pitch* y *roll* como objetivo final del TFG11 mencionado, e inicialmente instalan dos receptores ultrasonido a babor y estribor, de forma que permita realizar la estimación del ángulo de *roll*. Para ello se emplearon dos unidades ESP32, clonando la que había sido empleada en el proceso de toma de datos de calibración, pero surgieron varios problemas técnicos que se detallan a continuación:

- En primer lugar, los micros ESP32 comenzaron a fallar en las conexiones a la red *wifi* para el envío de datos. Tras analizar los resultados y buscar información por Internet, se averiguó que se debía a un problema de la librería de acceso a redes *wifi* que afecta a un gran número de modelos del micro seleccionado, entre los que se incluye el que teníamos disponible para el TFG. Debido a esto se optó por trasladar el montaje a los micros ESP8266, que ya habían sido probados en el TFG11 con excelentes resultados.
- Una vez reconstruido el montaje, se obtuvieron unos resultados iniciales que no se correspondían con los que, obtenidos durante la fase de calibración, ya que, aunque los valores medios se mantenían, la cantidad y magnitud del ruido añadido era muy considerable. Se realizaron pruebas con varios tipos de receptores (de 3V y 5V) para lo cual se necesitaron los elementos de conversión de 3V a 5 V, lo que permitía la comunicación de información digital entre ambos módulos sin que se produzcan subidas de tensión en el chip controlador. En todos los casos los errores se reproducían de forma persistente, hasta que finalmente se comprobó que el problema se debía al uso de dos receptores de forma simultánea, ya que al cancelar el disparo de uno de ellos el otro funcionaba solo tal y como era esperado. Es posible, esto es solo una hipótesis que no ha podido ser comprobada, que el uso de ultrasonidos en donde el receptor y el emisor están integrados (como es el caso HC-SRC04), y que necesiten el disparo del transmisor antes de recibir la señal emitida desde la base, generen interferencias que introduzcan niveles de ruido inasumibles. Debido a este problema, y a la falta de tiempo para buscar otro tipo de solución técnica se optó por realizar un montaje en donde un único ESP8266 es empleado para leer las señales de dos receptores. De esta forma, aunque el ancho de banda del sistema se reduce a la mitad, es posible recibir las señales en cada receptor de forma independiente y sin generar el problema indicado. Además, debe tenerse en cuenta que debido a este cambio es necesario introducir en el protocolo de transmisión los cambios necesarios que permitan determinar en cada pulso enviado, cuál debe ser el receptor que debe ser activado, lo cual supuso un trabajo adicional de adaptación del código C del receptor.

4.3.1 Test de calibración

En esta prueba inicial el objeto en donde se han instalado las unidades inerciales y los dos receptores de ultrasonido se encuentra en posición de reposo siendo la aceleración en el eje Z la única variable que debe tener un valor medio no nulo igual a la aceleración de la gravedad. En las Figuras 4.9 y 4.10 podemos observar los componentes de las aceleraciones lineales y las velocidades angulares que miden las IMUs tras haber aplicado un procedimiento de filtrado (filtro GH) y una corrección de *bias* u *offset* (los detalles de este filtrado pueden consultar en el TFG 11), por lo que los valores medios obtenidos coinciden con los esperados en esta posición.

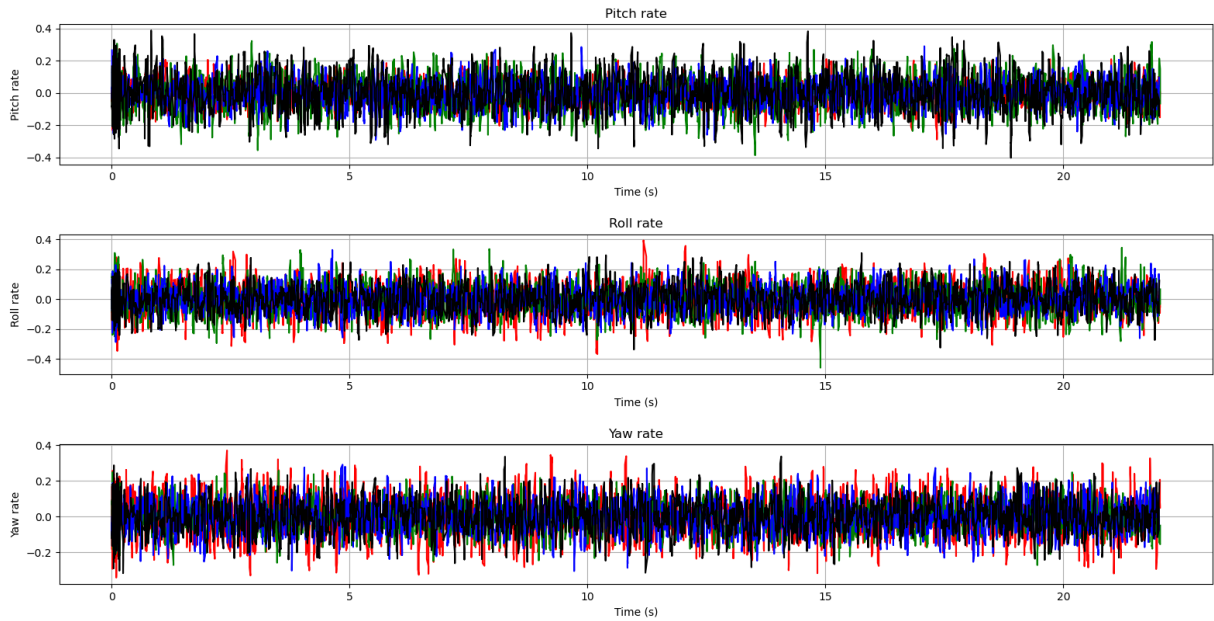


Figura 4-11 Aceleraciones de las unidades inerciales con objeto en reposo

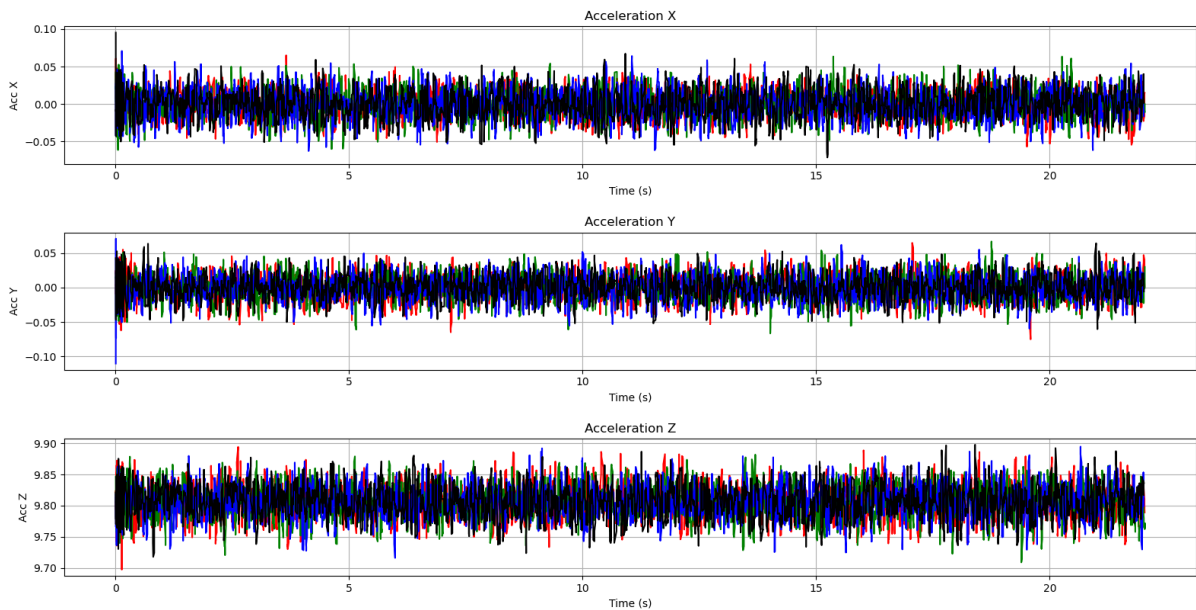


Figura 4-12 Velocidades angulares de las unidades inerciales con objeto en reposo.

La Figura 4-13 y la Figura 4-14 muestran las distancias obtenidas para los dos receptores de ultrasonidos con respecto a las bases emisoras de pulsos, tras aplicar los procedimientos de filtrado introducidos en el capítulo anterior del TFG. Podemos apreciar que las variaciones de distancia son mínimas, y se encuentra por debajo de los 3mm en ambos casos, lo cual induce a pensar que la precisión en cálculos será suficiente para estimar, incluso, pequeñas variaciones del ángulo de roll del objeto.

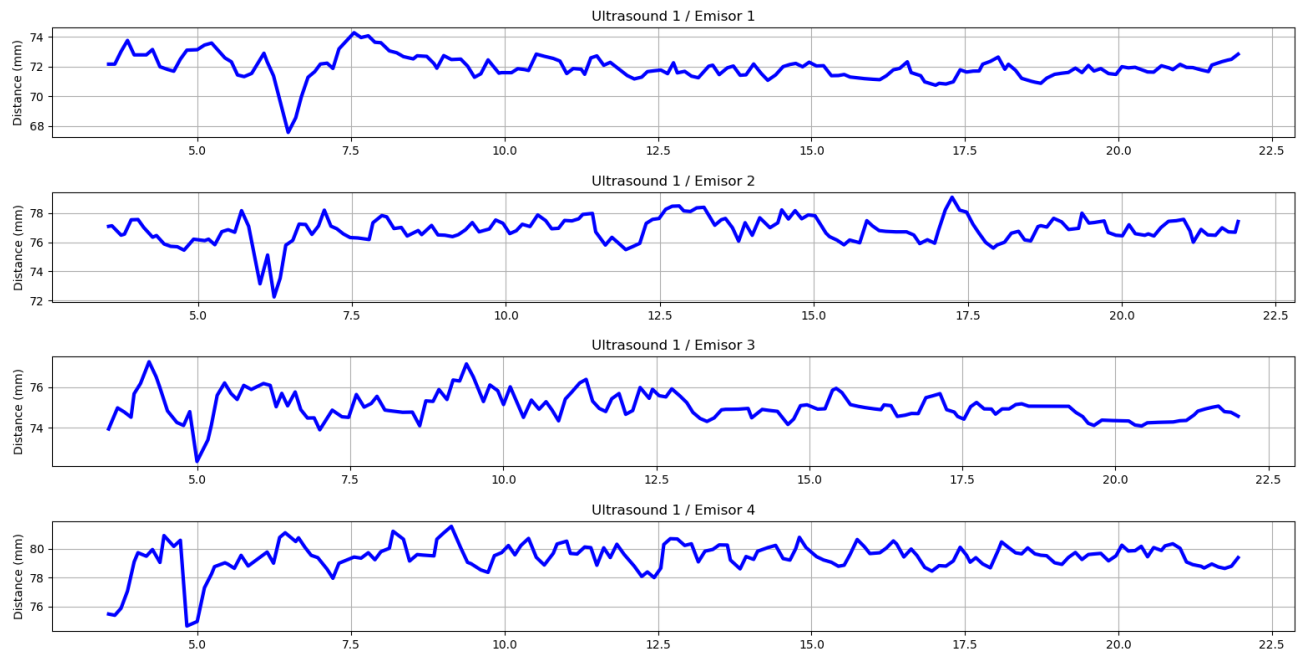


Figura 4-13 Distancias del receptor 1 a los emisores para la prueba de calibración.

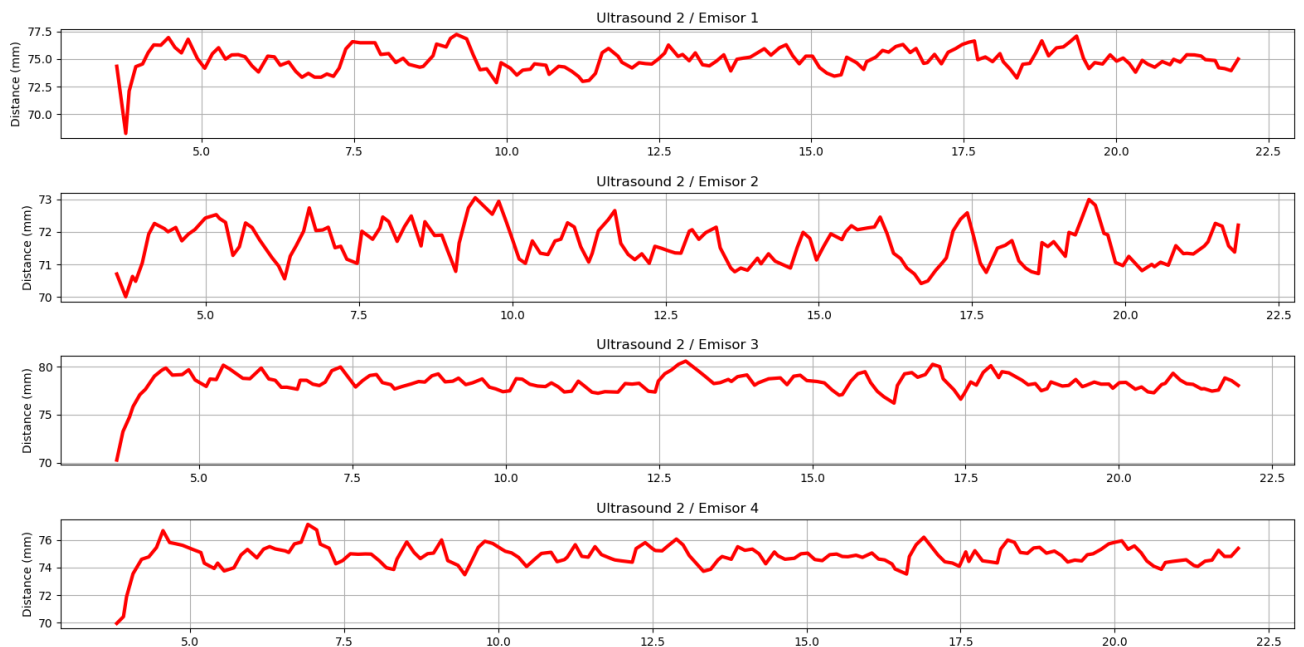


Figura 4-14 Distancias del receptor 2 a los emisores para la prueba de calibración

4.3.2 Test de movimiento vertical

En esta prueba se desplaza el objeto manualmente de forma aproximadamente vertical, con pequeñas modificaciones en sus ángulos de orientación. Se trata, por tanto, de la primera prueba del sistema en movimiento, de forma que se pretende evaluar la sensibilidad de la respuesta del sistema de posicionamiento por ultrasonidos en el primer caso en que se incluye movimiento en el objeto del que

se pretende obtener su posición y orientación. Las Figuras 4.13 y 4.14 muestran las aceleraciones lineales y velocidades angulares obtenidas de las cuatro unidades inerciales, en donde podemos apreciar claramente las variaciones en la aceleración del eje Z.

Figura 4.16: Distancias del receptor 2 a los emisores para la prueba de movimiento vertical.

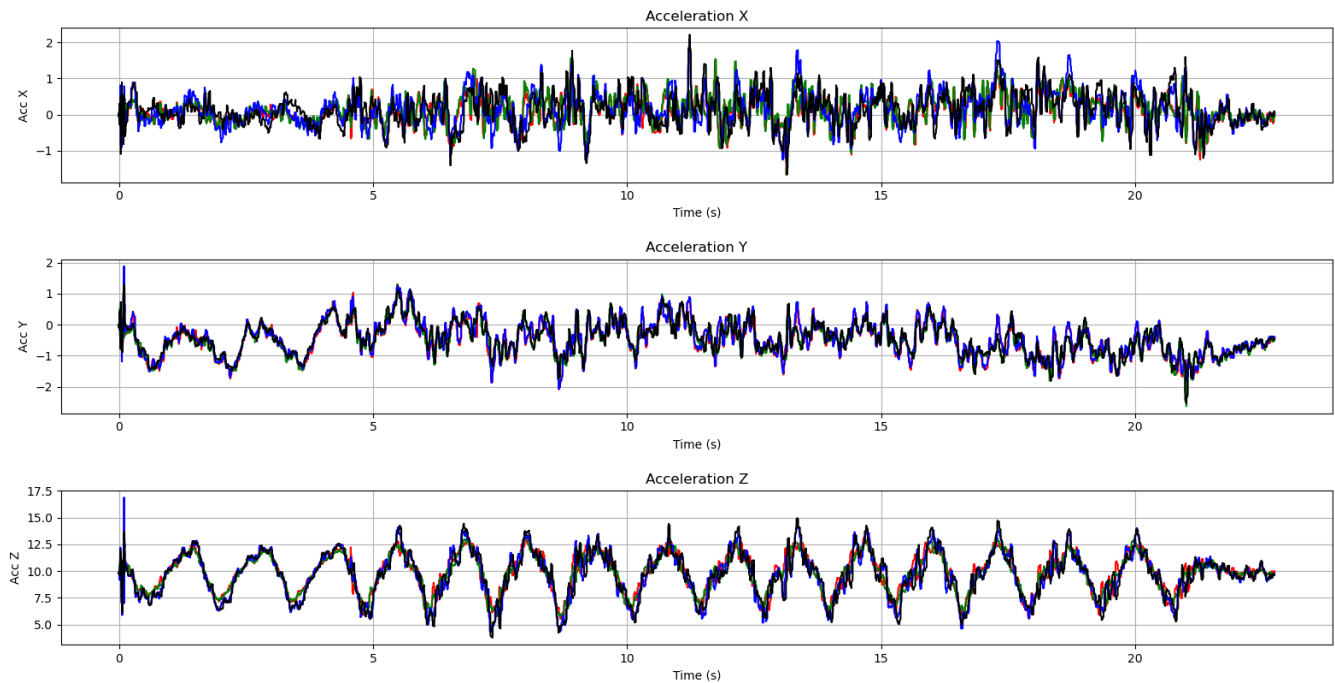


Figura 4-15 Aceleraciones de las unidades inerciales para la prueba de movimiento vertical.

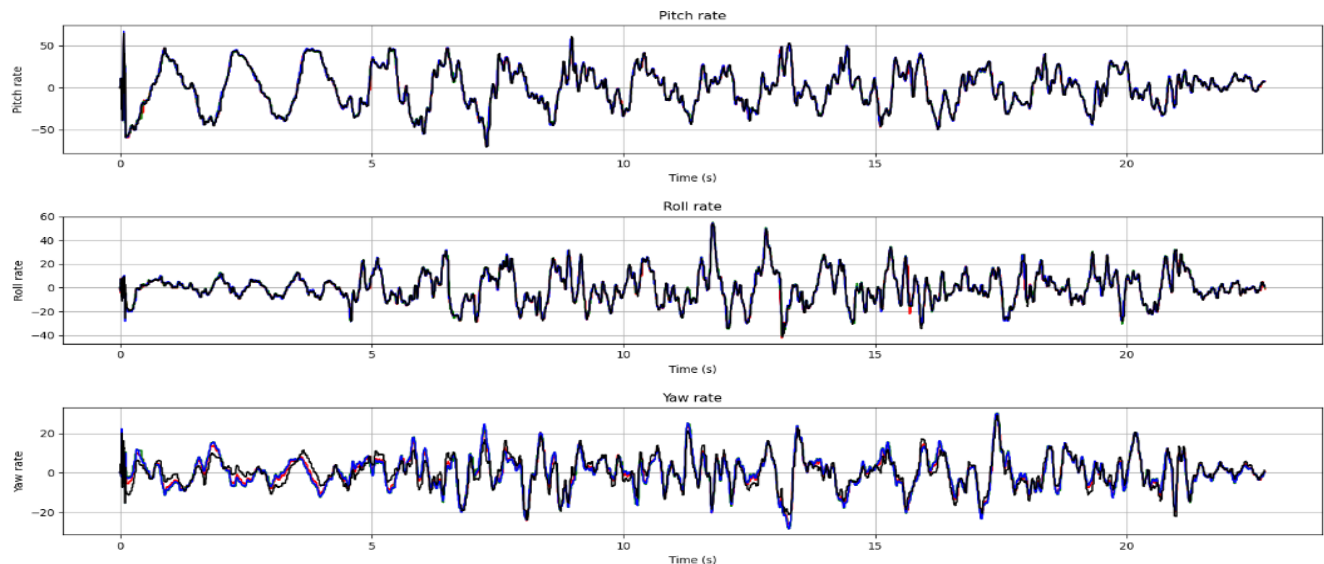


Figura 4-16 Velocidades angulares de las unidades inerciales para la prueba de movimiento vertical.

Las **¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.** muestran las distancias de los receptores a las bases emisoras, en donde se puede apreciar el patrón de desplazamientos verticales que provocan las oscilaciones sincronizadas. Estas oscilaciones reflejan la sensibilidad del sistema, e inducen a pensar que va a permitir realizar el seguimiento de

objetos que se mueven con velocidades dentro del rango de las velocidades que han sido ejecutas durante la prueba.

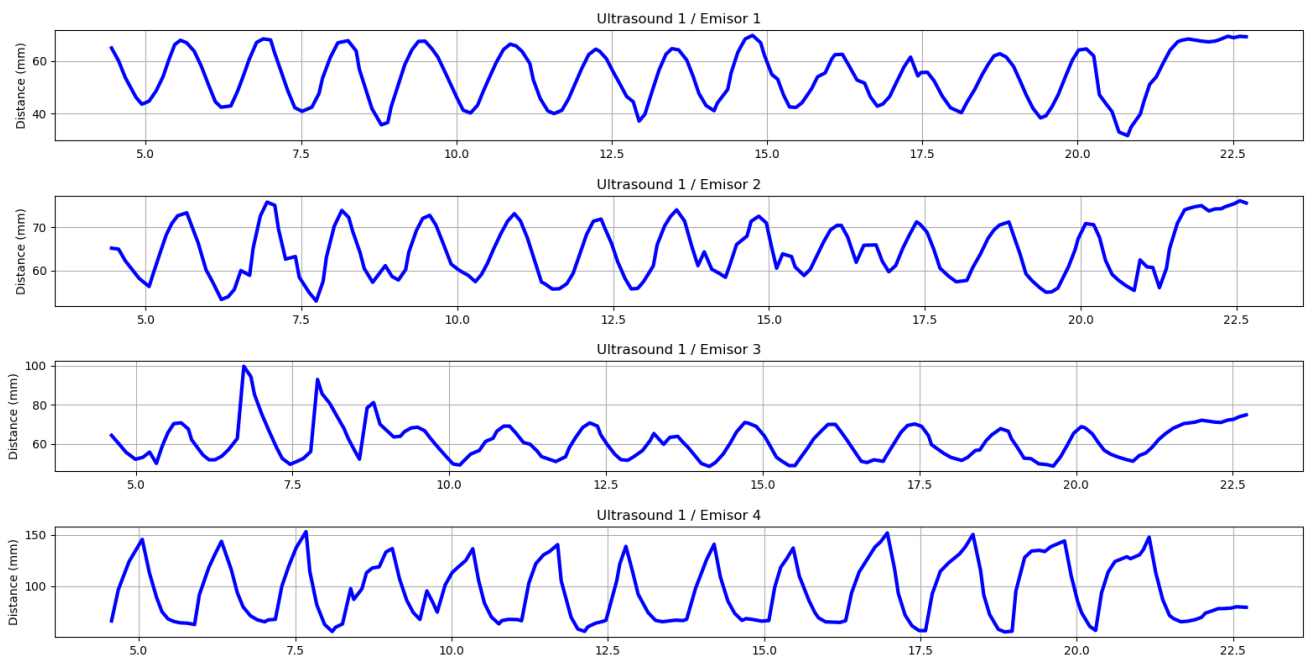


Figura 4-17 Distancias del receptor 2 a los emisores para la prueba de movimiento vertical.

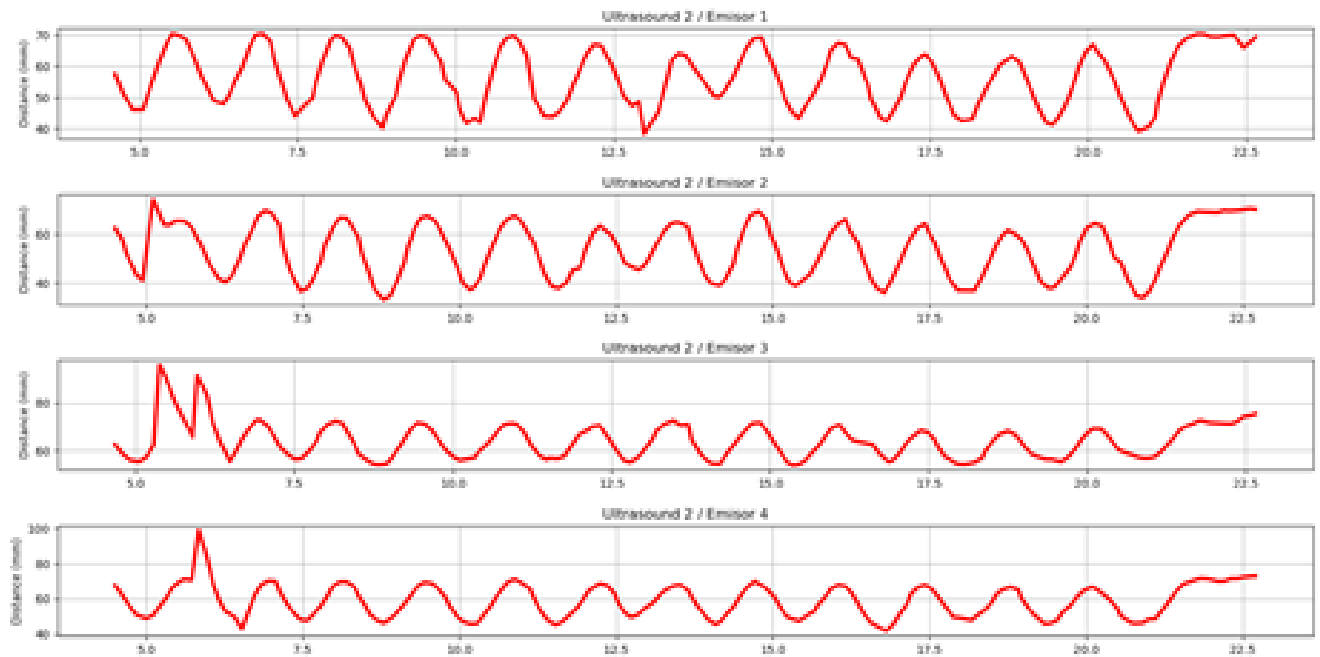


Figura 4-18 Distancias del receptor 2 a los emisores para la prueba de movimiento vertical.

4.3.3 Test de roll

Esta prueba marca uno de los objetivos básicos en la realización del TFG, esto es, el uso de receptores de ultrasonidos para realizar estimaciones acerca de las propiedades de la posición y orientación del objeto en 3D. Concretamente nos vamos a centrar en lo relativo a la estimación del ángulo de *roll* o balanceo, ya que se considera muy importante para el estudio de los sistemas de control de estabilidad cuando el objeto de estudio es un barco navegando.

Para ello el objeto es balanceado de forma lateral, de forma que se busca resaltar cambios en el ángulo de *roll* que se desea estimar mediante la aplicación de un filtro de Kalman. Debido a la presencia de una función *arctan* en las ecuaciones del modelo de estimación, no es posible aplicar un filtro de Kalman directamente sobre este sistema, ya que la función *arctan* no es de tipo lineal. Ante la falta de tiempo para realizar un estudio adecuado de la aplicación de un filtro de Kalman extendido (que requiere del cálculo del Jacobiano del modelo y la adaptación del código empleado para el filtro lineal) u otro tipo de solución con soporte para modelos no lineales (como los filtros *sigma point* o los filtros de partículas) se optó finalmente por realizar una aproximación lineal, de forma que se propone un modelo de la forma:

$$\text{Roll} = K (z_1 - z_2)$$

Donde K es una constante a calcular, y (z_1, z_2) son las coordenadas en el eje Z de los receptores que forman parte de la solución del sistema de posicionamiento en 3d (*trilateration*) introducido en el capítulo anterior, y que suponen la entrada de observación del filtro de Kalman.

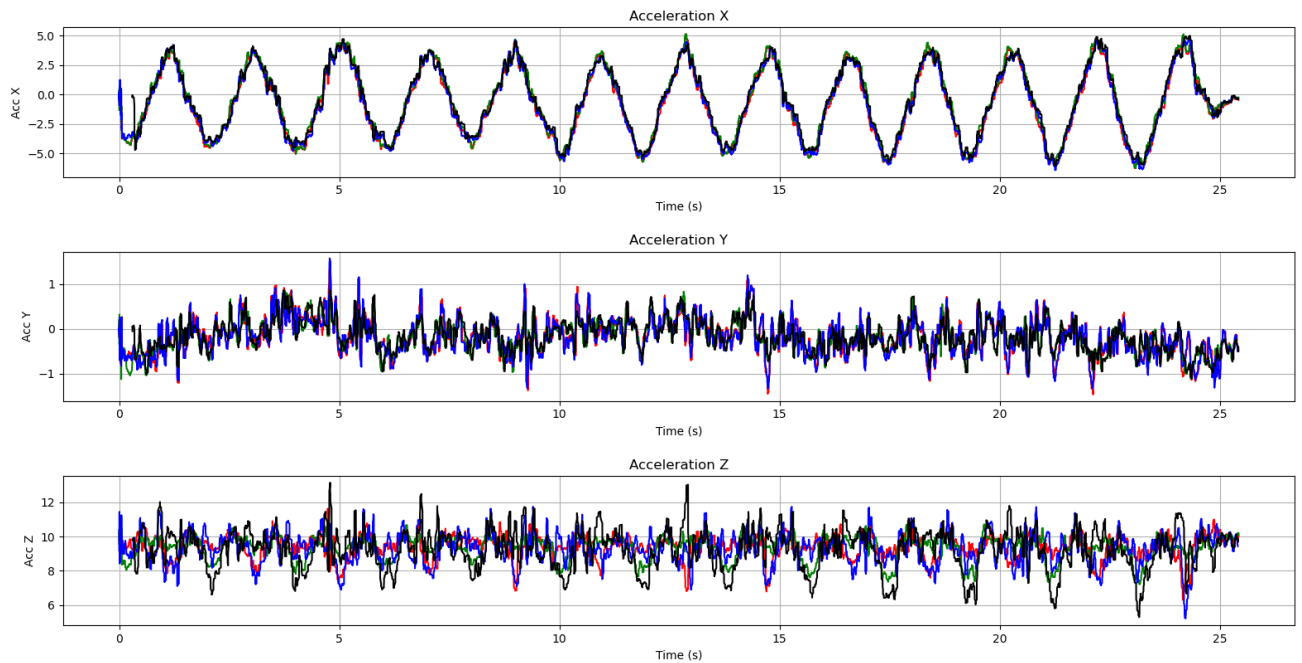


Figura 4-19 Aceleraciones de las unidades inerciales para la prueba de balanceo longitudinal.

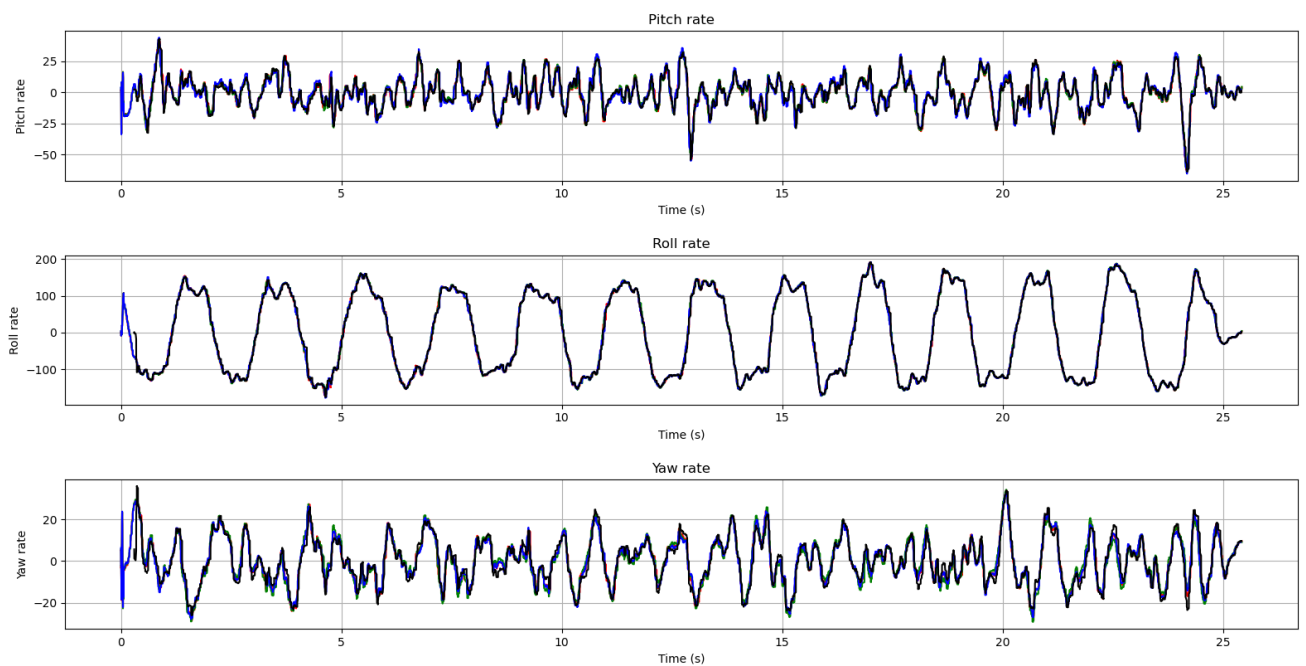


Figura 4-20 Velocidades angulares de las unidades inerciales para la prueba balanceo longitudinal.

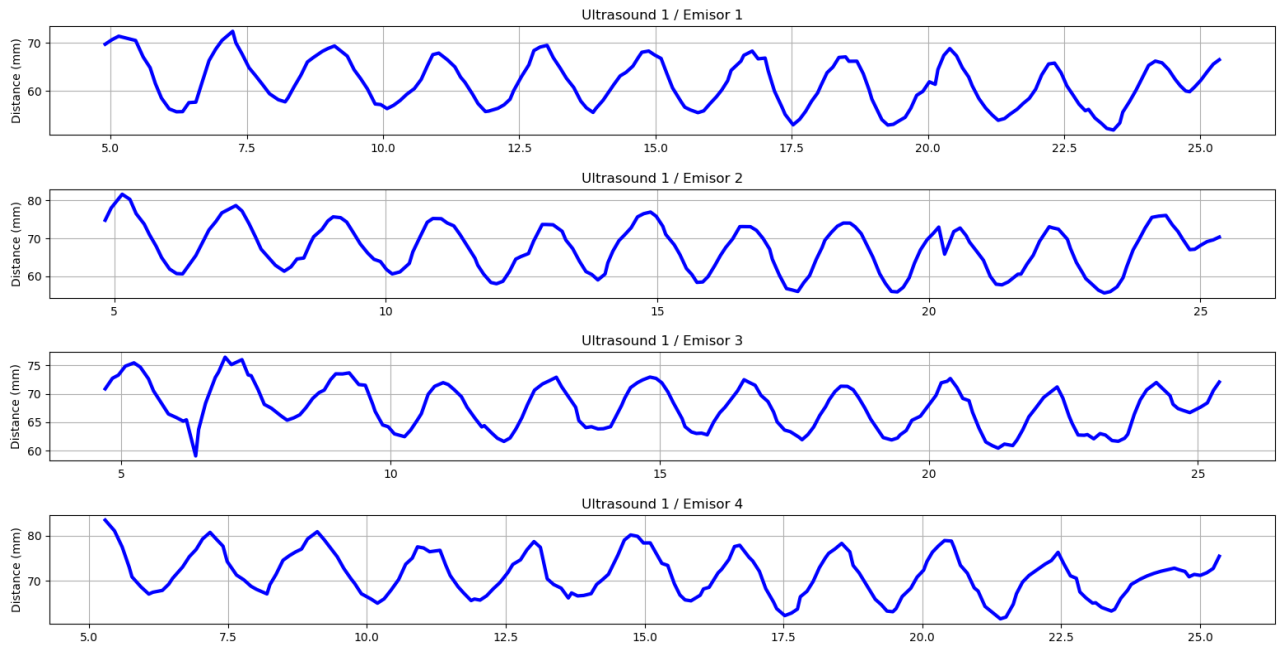


Figura 4-21 Distancias del receptor 2 a los emisores para la prueba de balanceo longitudinal.

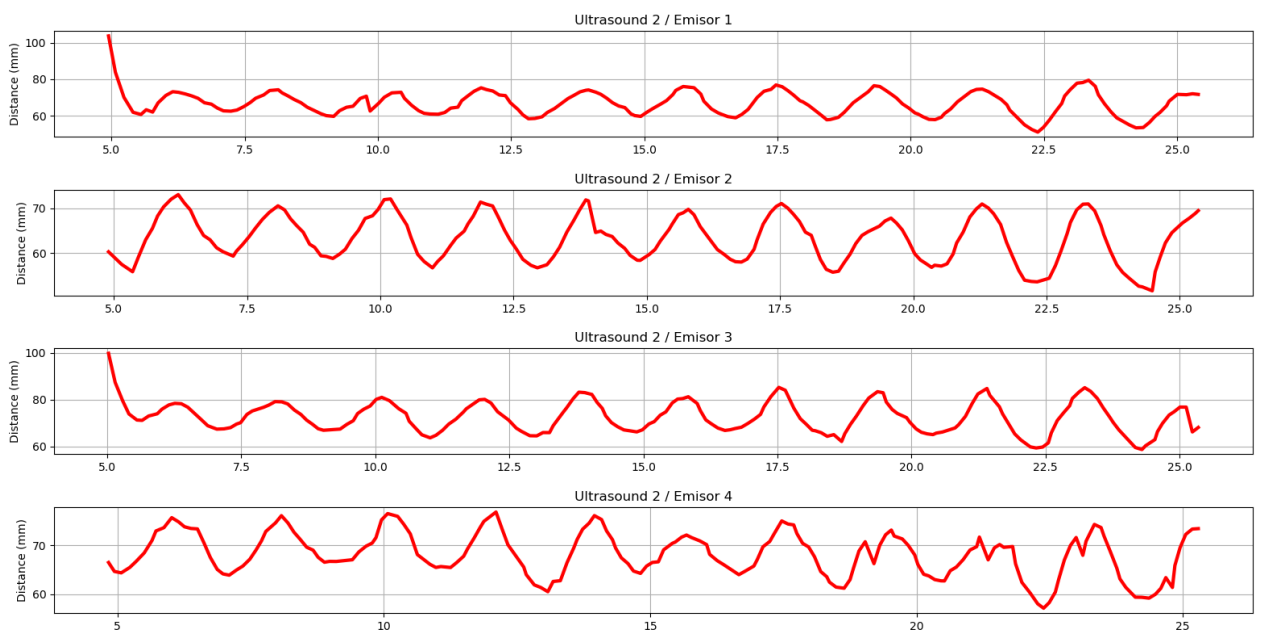


Figura 4-22 Distancias del receptor 2 a los emisores para la prueba de balanceo longitudinal.

Las Figuras 4.18 a 4.21 muestran los resultados de las aceleraciones, velocidades angulares y distancias de los receptores a las bases emisoras. Podemos ver claramente que el patrón del movimiento de balanceo lateral en la velocidad del *roll*, y también en las distancias de los receptores que cambian de forma sincronizada.

La **¡Error! No se encuentra el origen de la referencia.** muestra las coordenadas (z_1, z_2) así como el resultado final de la estimación del ángulo de *roll* a partir, únicamente, de la información de (z_1, z_2). Podemos ver que tras un inicio oscilatorio (asociado a la fase de adaptación de las matrices del algoritmo de estimación) el resultado obtenido es similar al que se obtiene con cada de las cuatro unidades inerciales. Puede apreciarse un ligero retardo (lógico, ya que la frecuencia de emisión de las IMUs es superior a la de los ultrasonidos), aunque es probable que un algoritmo más complejo que, por ejemplo, emplee las tres coordenadas de cada receptor, pueda mejorar los resultados la precisión de la estimación generada. En líneas generales, y para ser una primera prueba los resultados parecen ser muy prometedores y sugieren que con más datos y algoritmos más complejos es posible generar un sistema de posicionamiento de alta precisión con la red de sensores de bajo coste propuesta.

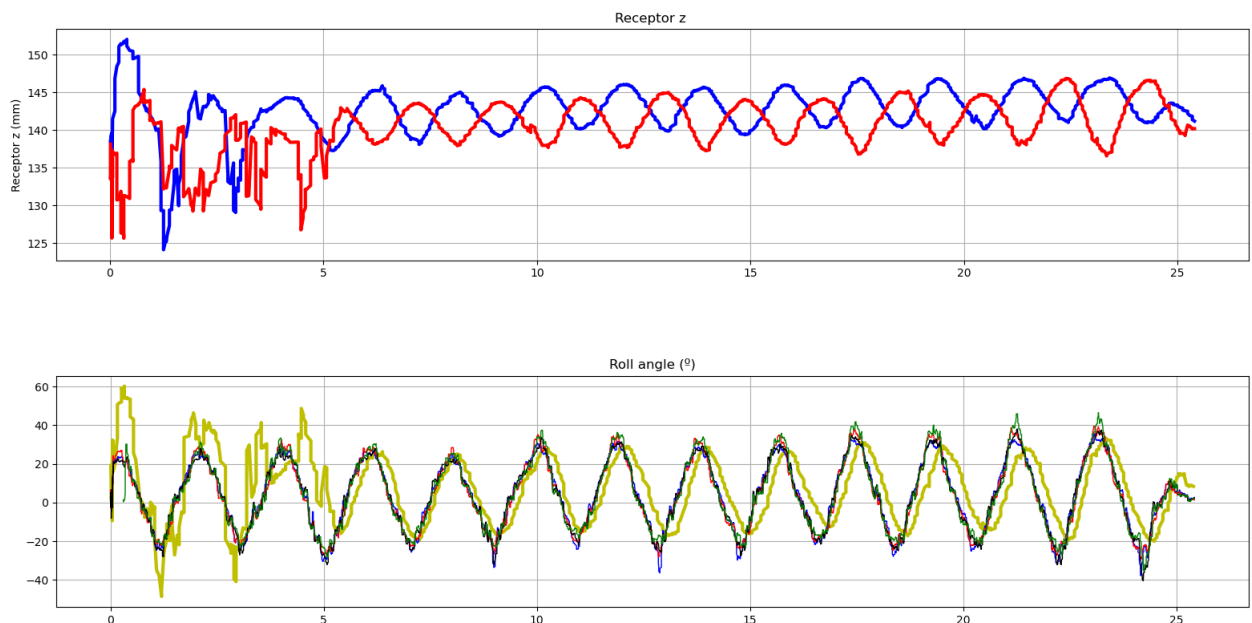


Figura 4-23 Coordenada z de los receptores y estimación del ángulo de roll del ultrasonido y de las unidades inerciales para la prueba de balanceo longitudinal.

4.3.4 Test general

En esta última prueba el objeto se mueve de forma genérica en una trayectoria en 3D sufriendo cambios en todas sus variables de posición y orientación. Debemos tener en cuenta que cierto tipo de movimientos pueden llegar a generar pérdida de la señal de pulsos de los receptores, ya que la distancia a la que se realiza el test no es suficientemente alta como permitir grandes oscilaciones de los ángulos sí que se pierda la “cobertura” del receptor de ultrasonidos.

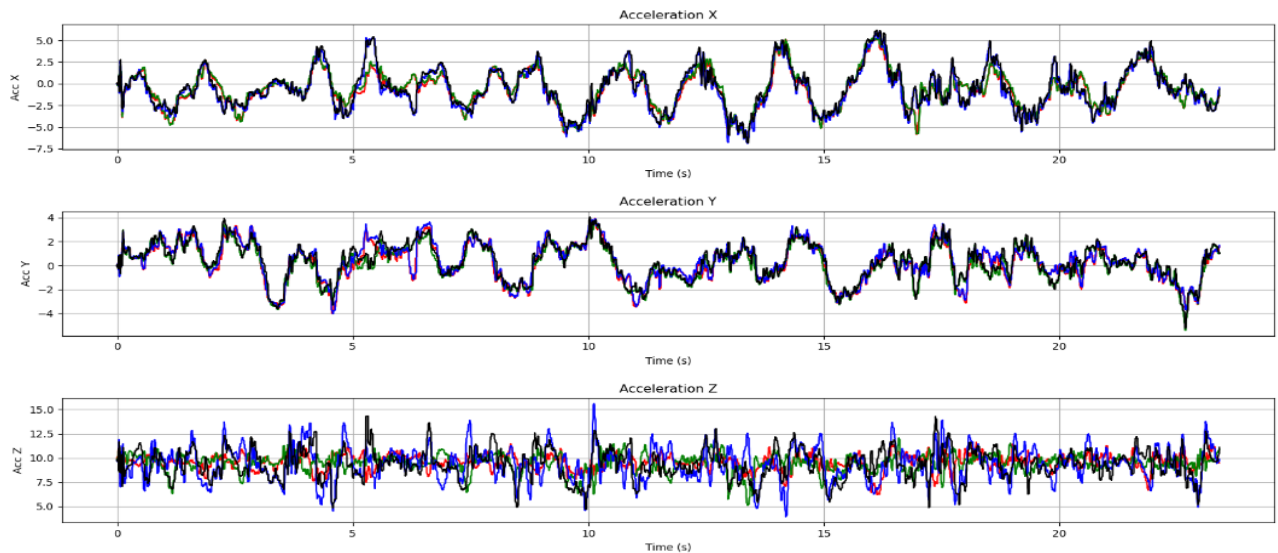


Figura 4-24 Aceleraciones de las unidades inerciales para la prueba de movimiento general.

Además, debe tenerse en cuenta que, con la simplificación lineal realizada, es decir, empleando únicamente (z_1, z_2) no es posible generar la matriz de rotación que permita expresar el ángulo del *roll* con respecto al obtenido en las unidades inerciales (debemos tener en cuenta que en 3 dimensiones una rotación del *yaw* seguida de una rotación del *pitch* genera un ángulo de *roll* no nulo), por lo que es de esperar que esto se refleje en las estimaciones en ciertos instantes de la prueba.

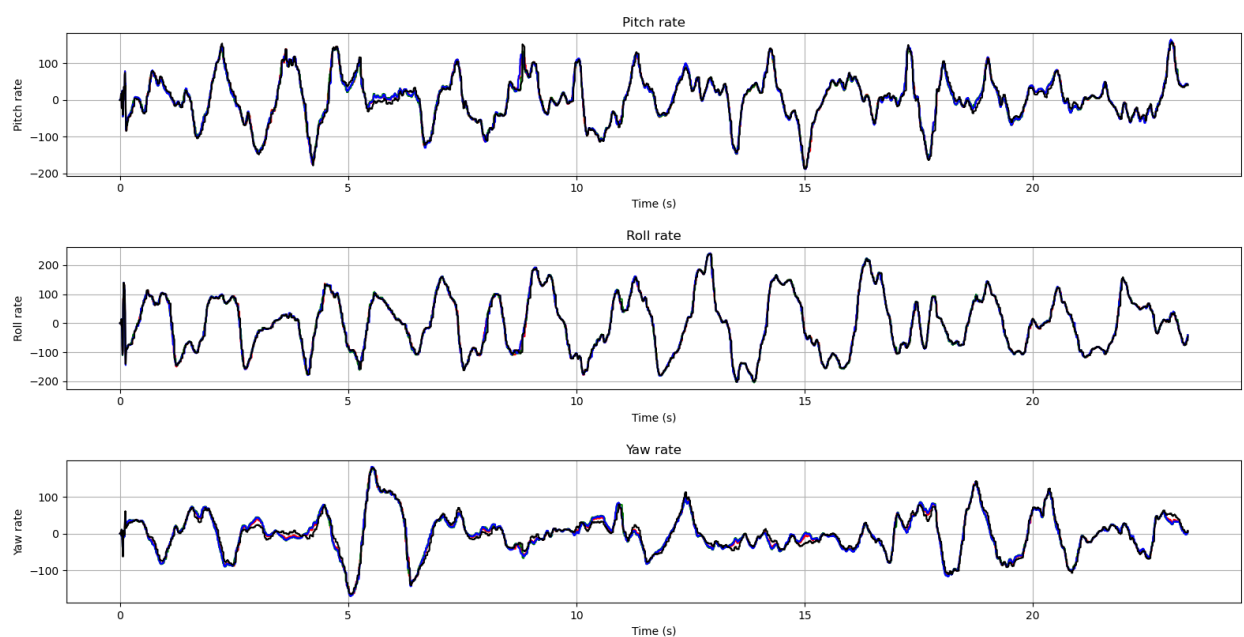


Figura 4-25 Velocidades angulares de las unidades inerciales para la prueba de movimiento general.

Las Figuras 4.22 y 4.23 muestran los datos obtenidos de las unidades inerciales, y la Figura 4.24 el resultado de la estimación del ángulo de *roll*. Podemos apreciar que en este caso aparecen ciertas divergencias entre los resultados de ambos sistemas que confirman el resultado esperado debido a la aplicación de la simplificación lineal. Por tanto, el siguiente paso natural será el desarrollo de un estimador que incluya elementos no lineales y que permita corregir las diferencias en el mayor número de situaciones posible.

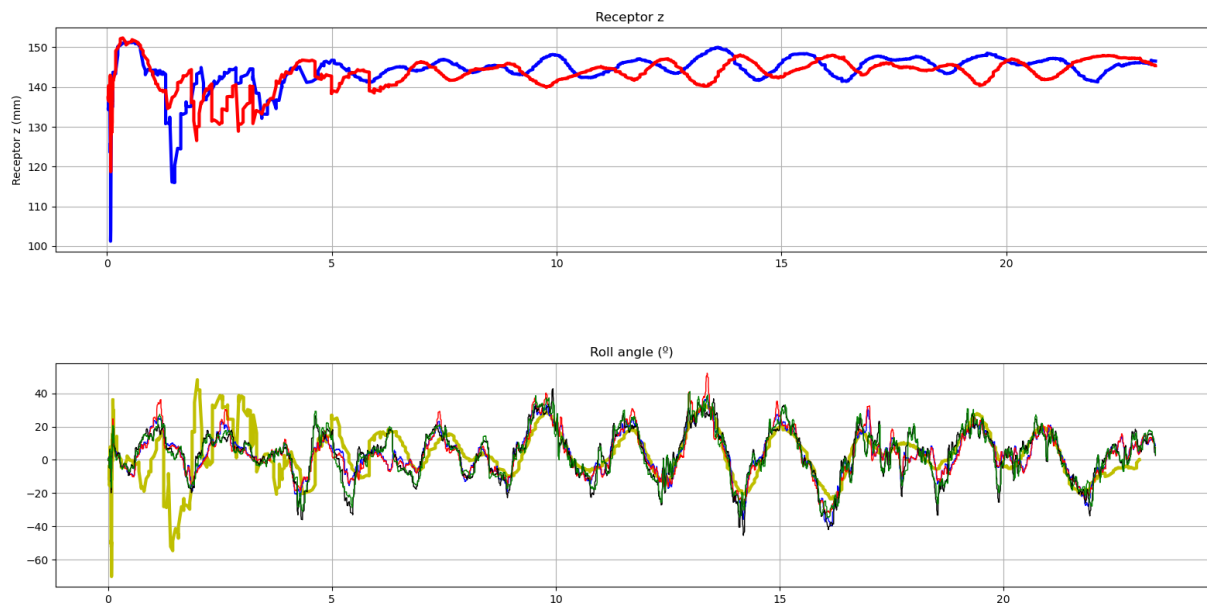


Figura 4-26 Velocidades angulares de las unidades inerciales para la prueba de movimiento general.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones y propuestas de mejora

Tras los experimentos realizados, hemos podido comprobar que los resultados son muy prometedores para unas distancias de aproximadamente 2-3m, tanto en lo relativo al posicionamiento en un único punto como a la fusión de datos de dos receptores para estimar una propiedad de la orientación de un objeto en 3D. A pesar de que los datos recibidos sin procesar presentan ruido de alta frecuencia y generación de *outliers* debido a la pérdida de captación de un pulso emitido, se ha comprobado que, tras aplicar técnicas sencillas de filtrado y estimación, es posible obtener resultados aceptables mediante la propuesta presentada en este TFG.

Por otro lado, es necesario destacar que la conexión entre los ESP32 y el servidor MQTT generó un elevado número de problemas, lo que dificultó la toma de datos y obligó a tener que realizar cambios de última hora sustituyéndolos por los ESP8266, con el fin de obtener una mayor fiabilidad. Además, tras realizar varias pruebas pudo comprobarse que el uso de dos receptores conectados a dos micros diferentes introducía en los datos de las pruebas una cantidad de ruido que no era consistente con las pruebas que solo emplean un único receptor, por lo que se optó por emplear único micro para ambos receptores. Se intuye que el problema lo pueden generar interferencias entre ambos receptores debido a la necesidad de disparar sus transmisores para activar el modo de recepción del *Echo*, pero habría que crear un sistema con receptores y emisores individuales y desarrollar la circuitería y código para su uso.

Pese a haber obtenido unos resultados realmente prometedores, hay que comentar que en lo referido a los prototipos creados, se pudo comprobar que el montaje inicial (aunque dotó de gran flexibilidad a la hora de la realización de las pruebas de calibración y test de movimiento realizados) resultó ser un tanto precario en ciertos aspectos, que en cierta medida es asociado al bajo coste de los materiales empleados. En futuras ocasiones se recomienda la construcción de una estructura más fiable y robusta que permita realizar una mejor comparación de los datos obtenidos mejorando la calibración y como consecuencia mejorando los estimadores. También se recomienda el uso de receptores y emisores de ultrasonidos individuales, es decir, sin emplear los módulos que llevan integrados ambos componentes, como se ha hecho durante este TFG.

5.2 Líneas futuras

Con este trabajo se marca un punto de partida realmente importante para la realización de sistemas de localización en entornos de laboratorio con un precio de producción realmente bajo comparado con otros sistemas.

Como continuación más próxima a este TFG se plantea reducir el tamaño de los sensores empleando PCB impresas que permitan reducir los montajes de los receptores. Además, se propone cambiar los módulos US-026 por receptores de ultrasonido y desarrollar cajas estancas que puedan albergar los sensores facilitando los experimentos en entornos acuáticos.

Tras esto, se debería continuar con la integración del proyecto con el TFG11 desarrollado por Rafael Montero de Espinosa Alés, ya sea como punto de comparación para saber si las IMU's pueden generar predicciones acertadas de la trayectoria o como elemento complementario para la integración de la información y poder realizar mejores predicciones. Por esta vía y con el fin de poder mejorar dichas predicciones se propone estudiar otro tipo de filtrados más modernos o incluso el empleo de técnicas de *machine learning* que con entrenamientos de los procesadores permitan mejorar los resultados obtenidos.

Por último, sería deseable mejorar el procesamiento de datos pudiendo realizar un seguimiento instantáneo de los resultados con el fin de poder seguir los experimentos en tiempo real. Esto permitirá que se integre en laboratorios de investigación con el objetivo de poder digitalizar de manera automática los experimentos que se realicen de cinemática y dinámica, pudiendo de esta forma llevar un registro dinámico y facilitando la comparación de la información con otros ensayos. Como aplicación más próxima se podría realizar en laboratorios como los del CUD donde con una inversión de capital relativamente baja se podría realizar el seguimiento de futuros trabajos de investigación.

6 BIBLIOGRAFÍA

- [1] F. D. R. Pereda, *Localizacion y autocalibrado simultaneo con sistemas de posicionamiento local ultrasonicos*, Alcalá: -, 2011.
- [2] C.-H. Ko, «RFID 3D Location sensing algorithms,» ELSEVIER, 2010.
- [3] L. Llamas, «Luis Llamas,» Luis Llamas, 17 Abril 2019. [En línea]. Available: <https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/>. [Último acceso: 26 Febrero 2021].
- [4] M. P. Pérez, *Evaluacion y comparacion de algoritmos de posicionamiento basados en tecnologia Ultra Wide Band*, Universitat de les illes Balears, 2019.
- [5] R. R. L. Jr., *Kalman and Bayesian Filters in Python*, -: -, 2020.
- [6] AZ-Delivery, «AZ-Delivery nRF24L01 manual,» de *AZ-Delivery nRF24L01 manual*.
- [7] AZ-Delivery, *Manual ESP32*, AZ.Delivery, 2021.
- [8] R. M. d. E. Ales, *Aplicacion de una red de sensores inalambricos distribuidos de bajo coste para el analisis de las propiedades dinamicas en 3 dimensiones de barcos en posiciones estacionarias*, Marín: ENM, 2021.
- [9] «Web de La Moncloa,» [En línea]. Available: <http://www.lamoncloa.gob.es>. [Último acceso: 13 enero 2015].
- [10] J. Rodríguez y V. Fernández, *Cómo redactar el estado del arte de un trabajo*, Editorial Genios, 2010.
- [11] P. Martínez y A. García, *Cómo escribir una buena memoria de TFG*, Publicaciones del 2000, 2013.
- [12] A. Pérez, *Cómo escribir una bibliografía*, Nuevas publicaciones.
- [13] W. A. Ruiz, «Experiencias y aplicaciones del LIDAR,» Institut Cartogràfic de Catalunya, Barcelona, 2003.
- [14] «AZ-Delivery,» [En línea]. Available: <https://www.az-delivery.net/>. [Último acceso: 15 marzo 2021].
- [15] «Arduino,» [En línea]. Available: <https://www.arduino.cc/>. [Último acceso: 15 Marzo 2021].
- [16] Python, [En línea]. Available: <https://www.python.org/>. [Último acceso: 15 Febrero 2021].

ANEXO I: CÓDIGO ARDUINO BASES EMISORAS

```
#include <nRF24L01.h>
#include "RF24.h"
#include <RF24_config.h>

uint8_t radio_addresses[][6] = {"RFTFG"};
uint8_t radio_data[1] = {1};
bool radio_state;

// ARDUINO NANO
const uint8_t ultrasound_trigger_pins[4] = {8, 7, 4, 5};
const uint8_t ultrasound_echo_pins[4] = {6, 6, 3, 3};
uint8_t pin_index = 0;
uint8_t receptor_index = 0;

// ARDUINO NANO/UNO
RF24 radio(9,10);

void setup() {
  Serial.begin(115200);
  while (!Serial);

  // Configure ultrasound emisor
  pinMode(ultrasound_echo_pins[0], INPUT);
  pinMode(ultrasound_echo_pins[2], INPUT);
  pinMode(ultrasound_trigger_pins[0], OUTPUT);
  pinMode(ultrasound_trigger_pins[1], OUTPUT);
  pinMode(ultrasound_trigger_pins[2], OUTPUT);
  pinMode(ultrasound_trigger_pins[3], OUTPUT);

  // Configure radio frequency transmissor
  radio.begin();
  radio.setChannel(8);
  radio.setPALevel(RF24_PA_LOW);
  radio.setDataRate(RF24_1MBPS);
  radio.setCRCLength(RF24_CRC_16);
  radio.openWritingPipe(radio_addresses[0]);
  radio.powerUp();
  radio.stopListening();
}

void loop() {
  // Send RF message
  radio_data[0] = receptor_index + 1;
```

```

    radio.writeFast(&radio_data,1);
    radio_state = radio.txStandBy();

    delayMicroseconds(5);
    digitalWrite(ultrasound_trigger_pins[pin_index], HIGH);
    delayMicroseconds(10);
    digitalWrite(ultrasound_trigger_pins[pin_index], LOW);

    pin_index = pin_index + 1;
    if (pin_index > 3){
        pin_index = 0;
    }
    receptor_index = receptor_index + 1;
    if (receptor_index > 7){
        receptor_index = 0;
    }
    delay(10);
}

```

ANEXO II: CÓDIGO RX ARDUINO CON ESP8266

```
////////// LIBRARIES LOAD ////////////
#include <ESP8266WiFi.h> // Wifi library
#include <PubSubClient.h> // MQTT library to publish messages
#include <SPI.h>
#include <nRF24L01.h>
#include "RF24.h"
#include "printf.h"
#include <RF24_config.h>

////////// SENSOR CONFIGURATION ////////////
uint8_t sensor_id = 1; // Sensor unique identification
uint16_t sensor_loop_delay = 0; // Time between data send in milliseconds

////////// WIFI CONFIGURATION ////////////
// Wifi network name
const char* wifi_ssid = "???????";

// Wifi password
const char* wifi_password = "???????";

////////// MQTT CONFIGURATION ////////////
char mqtt_hostname[] = "192.168.43.164"; // MQTT IP address of machine
const char* mqtt_data_topic = "TFG/ULTRASOUND"; // MQTT token to publish IMU data
const char* mqtt_protocol_topic = "TFG/PROTOCOL"; // MQTT token to read and publish IMU protocol
int mqtt_port = 1883; // Default MQTT port
String mqtt_message = ""; // MQTT message to be send o received

////////// RADIO FREQUENCY CONFIGURATION ////////////
uint8_t radio_addresses[][6] = {"RFTFG"};
uint8_t radio_data[1] = {1};

// ESP32
uint8_t radio_ce_pin = 2;
uint8_t radio_csn_pin = 4;

////////// ULTRASOUND CONFIGURATION ////////////
// ESP8266
const uint8_t ultrasound_trigger_pin_1 = D1;
const uint8_t ultrasound_echo_pin_1 = D0;

const uint8_t ultrasound_trigger_pin_2 = D3;
const uint8_t ultrasound_echo_pin_2 = D0;

uint8_t ultrasound_trigger_pin = 0;
uint8_t ultrasound_echo_pin = 0;

////////// PROJECT OBJECTS ////////////
WiFiClient wifiClient;
PubSubClient MQTTClient(wifiClient);
RF24 radio(radio_ce_pin, radio_csn_pin);

void setup()
{
  Serial.begin(115200);
  while (!Serial);

  MQTTClient.setServer(mqtt_hostname, mqtt_port);
  mqttConnect();

  // Configure ultrasound emisor
  pinMode(ultrasound_trigger_pin_1, OUTPUT);
  pinMode(ultrasound_echo_pin_1, INPUT);
  pinMode(ultrasound_trigger_pin_2, OUTPUT);
  pinMode(ultrasound_echo_pin_2, INPUT);

  // Configure radio frequency transmissor
  radio.begin(); // Setup and configure rf radio
```

```
radio.setChannel(8);
radio.setPALevel(RF24_PA_LOW);
radio.setDataRate(RF24_1MBPS);
radio.setCRCLength(RF24_CRC_16);
radio.openReadingPipe(1,radio_addresses[0]);
radio.startListening();
radio.powerUp();
}

void loop()
{
  if (radio.available()){
    radio.read(&radio_data,1);
    uint8_t emisor_id = (uint8_t) radio_data[0];
    if ((emisor_id == 2) or (emisor_id == 4) or (emisor_id == 5) or (emisor_id == 7)) {
      ultrasound_trigger_pin = ultrasound_trigger_pin_1;
      ultrasound_echo_pin = ultrasound_echo_pin_1;
      sensor_id = 1;
    }else{
      ultrasound_trigger_pin = ultrasound_trigger_pin_2;
      ultrasound_echo_pin = ultrasound_echo_pin_2;
      sensor_id = 2;
    }

    // Sensor unique identification
    digitalWrite(ultrasound_trigger_pin, HIGH);
    delayMicroseconds(10);
    digitalWrite(ultrasound_trigger_pin, LOW);
    long pulse_time = pulseIn(ultrasound_echo_pin, HIGH, 10000); //, pulse_wait_time*1.0e3

    if (sensor_id == 1){
      if (emisor_id == 5){
        emisor_id = 1;
      }else if(emisor_id == 7){
        emisor_id = 3;
      }
      mqtt_message = String(emisor_id) + ",11," + String(pulse_time);
    }else{
      if (emisor_id == 6){
        emisor_id = 2;
      }else if(emisor_id == 8){
        emisor_id = 4;
      }
      mqtt_message = String(emisor_id) + ",22," + String(pulse_time);
    }
    mqttSendMessage();
  }
}

////////// MQTT FUNCTIONS //////////

void mqttSendMessage()
{
  char attributes[1000];
  mqtt_message.toCharArray( attributes, 1000 );
  MQTTClient.publish(mqtt_data_topic, attributes);
}

void mqttConnect()
{
  while (!MQTTClient.connected()) {
    WiFi.mode(WIFI_STA);
    if ( WiFi.status() != WL_CONNECTED) {
      Serial.println("Connecting to Wifi");
      WiFi.begin(wifi_ssid, wifi_password);
      while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
      }
      Serial.println(".");
      Serial.println("Connected to Wifi !!!");
    }
    Serial.println("Connecting to MQTT server");
    if (MQTTClient.connect("ultrasound", mqtt_data_topic, NULL) )
    {
      Serial.println("Connected to MQTT server !!! ");
      MQTTClient.subscribe(mqtt_protocol_topic);
    }
  }
}
```

```
    }  
    else {  
        Serial.println( " ... retrying in 1 second ... " );  
        delay( 1000 );  
    }  
}  
}
```

ANEXO III: CÓDIGO PYTHON CAPTURA DE DATOS

```
# -*- coding: utf-8 -*-
import traceback, time, sys
sys.path.insert(0, 'C:\\Users\\jagaprieto\\Desktop\\JUNE')
from MQTT import MQTT
import numpy as np

#####
sensor_type = 'hcsr04'
dir_name = '2021_03_15_attitude'
file_name = str(sensor_type) + "_calibration.csv"
print("file_name=", file_name)
data_size = 25000

#####
imu_data = np.zeros(shape=(data_size, 8), dtype=float) # id, time, acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z
ultrasound_data = np.zeros(shape=(data_size, 4), dtype=float) # time, emisor_id, receptor_id, pulse_time (microseconds)

MQTTObj = MQTT()
MQTTObj.connect('192.168.43.164', 1883)
MQTTObj.subscribe(["TFG/ULTRASOUND",1], ("TFG/IMU",1), ("TFG/PROTOCOL",1))
MQTTObj.publish("TFG/PROTOCOL", "1")

ultrasound_counter = 0
imu_counter = 0
start_time = time.time()
end_ultrasound = False
end_imu = False

while imu_counter < data_size:
    mqtt_data_txt = MQTTObj.read()
    if not mqtt_data_txt is None:
        mqtt_data = mqtt_data_txt.split(",")

        if ultrasound_counter < data_size:
            if (len(mqtt_data) == 3): # ULTRASOUND
                pulse_time = float(mqtt_data[2])
                if (pulse_time > 5): # pulse time
                    reception_time = time.time() - start_time
                    ultrasound_data[ultrasound_counter, 0] = reception_time
                    ultrasound_data[ultrasound_counter, 1] = int(mqtt_data[0])
                    ultrasound_data[ultrasound_counter, 2] = int(mqtt_data[1])
                    ultrasound_data[ultrasound_counter, 3] = pulse_time
                    ultrasound_counter +=1
            else:
                end_ultrasound = True
```

```
if imu_counter < data_size and ultrasound_counter > 0:
    if (len(mqtt_data) == 11): # IMU
        reception_time = time.time() - start_time
        imu_data[imu_counter, 0] = float(mqtt_data[0]) # imu_id
        imu_data[imu_counter, 1] = float(reception_time)
        imu_data[imu_counter, 2] = float(mqtt_data[2]) # acc_x
        imu_data[imu_counter, 3] = float(mqtt_data[3]) # acc_y
        imu_data[imu_counter, 4] = float(mqtt_data[4]) # acc_z
        imu_data[imu_counter, 5] = float(mqtt_data[5]) # gyro_x
        imu_data[imu_counter, 6] = float(mqtt_data[6]) # gyro_y
        imu_data[imu_counter, 7] = float(mqtt_data[7]) # gyro_z
        if (imu_counter % 100) == 0:
            print(imu_counter)
        imu_counter += 1
    else:
        end_imu = True

MQTTObj.publish("TFG/PROTOCOL", "0")
file_path = str('./' + dir_name + '/imu_' + file_name)
print('imu_file_path=', file_path)
np.savetxt(file_path, imu_data, delimiter=",")
ultrasound_data = ultrasound_data[0:ultrasound_counter,:]
file_path = str('./' + dir_name + '/ultrasound_' + file_name)
print('ultrasound_file_path=', file_path)
np.savetxt(file_path, ultrasound_data, delimiter=",")
```

ANEXO IV: CÓDIGO PYTHON ANALISIS Y FILTRADO DE DATOS

```
# -*- coding: utf-8 -*-
import traceback, time, sys
import numpy as np
import matplotlib.pyplot as plt
from getEmisorCoordinates import getEmisorCoordinates
from getReceptorCoordinates import getReceptorCoordinates
from numpy import genfromtxt
from trilateration3Points import trilateration3Points
from estimateIMUAttitude import estimateIMUAttitude
from filterGH import filterGH

#####

sensor_type = 'hcsr04'
dir_name = '2021_03_15_attitude'
file_name = str(sensor_type) + "_test_roll.csv"
print("file_name=", file_name)
G_ULTRASOUND = 0.5
H_ULTRASOUND = 0.2
G_IMU = 0.2
H_IMU = 0.1
gravity = 9.80665
TIME_TO_DISTANCE = 34.3
L = 15 # (mm) Distance between ultrasound sensors
window_size = 50
std_gain = 3.00

#.....IMU1.....
acc_x_mean_1= 0.04727375451262006
acc_y_mean_1= -0.003547088539291741
acc_z_mean_1= 0.92285721401702 - 1.0
gyro_x_mean_1= 4.073209619898725
gyro_y_mean_1= 1.1108420109596284
gyro_z_mean_1= -0.9747000984894009
#.....IMU2.....
acc_x_mean_2= 0.06777491875152056
acc_y_mean_2= -0.006266157614658059
acc_z_mean_2= 0.912008397560039 - 1.0
gyro_x_mean_2= 1.5973582521741383
gyro_y_mean_2= -2.162789788954913
gyro_z_mean_2= -5.606075918642236
#.....IMU3.....
acc_x_mean_3= 0.06831485417988224
acc_y_mean_3= 0.012476720928444021
acc_z_mean_3= 1.0245745236953465 - 1.0
```

```
gyro_x_mean_3= -4.46269413203103
gyro_y_mean_3= -1.0847469719074663
gyro_z_mean_3= -3.9634130386110713
#.....IMU4.....
acc_x_mean_4= -0.05757752489718176
acc_y_mean_4= -0.01679080801734216
acc_z_mean_4= 0.8775619993605672 - 1.0
gyro_x_mean_4= 0.5961860129473068
gyro_y_mean_4= -0.7030186409417608
gyro_z_mean_4= -2.178134404666765

imu_calibration_noise_data = []
imu_calibration_noise_data.append([0.030387, 0.030614, 0.047509, 0.136699, 0.185545, 0.208693])
imu_calibration_noise_data.append([0.031595, 0.028858, 0.04105, 0.207361, 0.203021, 0.221259])
imu_calibration_noise_data.append([0.030249, 0.032747, 0.050443, 0.205358, 0.207641, 0.182644])
imu_calibration_noise_data.append([0.033905, 0.032539, 0.049021, 0.201353, 0.160368, 0.160808])

#####
ultrasound_file_path = str('./' + dir_name + 'ultrasound_' + file_name) # time, emisor_id, receptor_id, pulse_time (microseconds)
ultrasound_data = genfromtxt(ultrasound_file_path, delimiter=',')
imu_file_path = str('./' + dir_name + 'imu_' + file_name)
imu_data = genfromtxt(imu_file_path, delimiter=',') # id, time, acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z

##### ULTRASOUND #####
ultrasound_data_size = ultrasound_data.shape[0]
#print('ultrasound_data_size=',ultrasound_data_size, 'imu_data_size=' , imu_data_size)
ultrasound_times_1 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_pulse_durations_1 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_pulse_distances_1 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_coordinates_x_1 = []
ultrasound_coordinates_y_1 = []
ultrasound_coordinates_z_1 = []
ultrasound_coordinates_time_1 = []
ultrasound_times_2 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_pulse_durations_2 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_pulse_distances_2 = {0:[], 1:[], 2:[], 3:[]}
ultrasound_coordinates_x_2 = []
ultrasound_coordinates_y_2 = []
ultrasound_coordinates_z_2 = []
ultrasound_coordinates_time_2 = []

ultrasound_estimated_angles = []
ultrasound_estimated_angles_times = []

pulse_duration_dx_1 = [0,0,0,0]
```

```

pulse_duration_dx_2 = [0,0,0,0]
pulse_duration_est_1 = [0,0,0,0]
pulse_duration_est_2 = [0,0,0,0]
receptor_index_1 = [0,0,0,0]
receptor_index_2 = [0,0,0,0]
time_1_prev = [0,0,0,0]
time_2_prev = [0,0,0,0]
trilateration_3d_points_1 = [0,0,0,0]
trilateration_3d_distances_1 = [0,0,0,0]
trilateration_3d_points_2 = [0,0,0,0]
trilateration_3d_distances_2 = [0,0,0,0]
combinations = [[0,1,2], [0,1,3], [0,2,3], [1,2,3]]
for ultrasound_index in range(ultrasound_data_size): # time, emisor_id, receptor_id, pulse_time (microseconds)
    time = ultrasound_data[ultrasound_index, 0]
    emisor_id = int(ultrasound_data[ultrasound_index, 1]) - 1
    receptor_id = int(ultrasound_data[ultrasound_index, 2])
    pulse_duration = ultrasound_data[ultrasound_index, 3]*0.001 #ms
    (emisor_x, emisor_y, emisor_z) = getEmisorCoordinates(emisor_id)

    if receptor_id == 11:
        ##### GH Filter #####
        if receptor_index_1[emisor_id] > 0:
            if len(ultrasound_pulse_durations_1[emisor_id]) > window_size:
                mean_data = ultrasound_pulse_durations_1[emisor_id][-window_size:]
                pulse_mean = np.mean(ultrasound_pulse_durations_1[emisor_id][-window_size:])
                pulse_std = np.std(ultrasound_pulse_durations_1[emisor_id][-window_size:])
                if abs(pulse_duration-pulse_mean) > std_gain*pulse_std:
                    pulse_duration = pulse_duration_est_1[emisor_id]

            dt = time - time_1_prev[emisor_id]
            pulse_duration_x_pred = pulse_duration_est_1[emisor_id] + (pulse_duration_dx_1[emisor_id]*dt)
            residual = pulse_duration - pulse_duration_x_pred
            pulse_duration_dx_1[emisor_id] = pulse_duration_dx_1[emisor_id] + (H_ULTRASOUND * residual / dt)
            pulse_duration_est_1[emisor_id] = pulse_duration_x_pred + G_ULTRASOUND * residual
        else:
            pulse_duration_est_1[emisor_id] = pulse_duration

    time_1_prev[emisor_id] = time
    receptor_index_1[emisor_id] += 1
    ultrasound_times_1[emisor_id].append(time)
    ultrasound_pulse_durations_1[emisor_id].append(pulse_duration_est_1[emisor_id])
    pulse_distance = pulse_duration_est_1[emisor_id]*TIME_TO_DISTANCE
    ultrasound_pulse_distances_1[emisor_id].append(pulse_distance)

##### COORDINATES #####
trilateration_3d_points_1[emisor_id] = (emisor_x, emisor_y, emisor_z)
trilateration_3d_distances_1[emisor_id] = pulse_distance

```

```
if receptor_index_1[emisor_id] > 3:
    comb_index = 0
    receptor_x_estimates = []
    receptor_y_estimates = []
    receptor_z_estimates = []
    for combination in combinations:
        emisor_1 = combination[0]
        emisor_2 = combination[1]
        emisor_3 = combination[2]
        (receptor_x_estimate, receptor_y_estimate, receptor_z_estimate) = trilateration3Points(trilateration_3d_points_1[emisor_1],
trilateration_3d_points_1[emisor_2], trilateration_3d_points_1[emisor_3],
trilateration_3d_distances_1[emisor_1], trilateration_3d_distances_1[emisor_2], trilateration_3d_distances_1[emisor_3])
        if comb_index == 1:
            receptor_y_estimate = -1.0*receptor_y_estimate
        elif comb_index == 2:
            receptor_x_estimate = -1.0*receptor_x_estimate
            receptor_z_estimate = -1.0*receptor_z_estimate
        elif comb_index == 3:
            receptor_x_estimate = -1.0*receptor_x_estimate
            receptor_y_estimate = -1.0*receptor_y_estimate
            receptor_z_estimate = -1.0*receptor_z_estimate
        comb_index += 1
        receptor_x_estimates.append(receptor_x_estimate)
        receptor_y_estimates.append(receptor_y_estimate)
        receptor_z_estimates.append(receptor_z_estimate)

    ultrasound_coordinates_x_1.append(np.mean(receptor_x_estimates))
    ultrasound_coordinates_y_1.append(np.mean(receptor_y_estimates))
    ultrasound_coordinates_z_1.append(np.mean(receptor_z_estimates))
    ultrasound_coordinates_time_1.append(time)
else:
    ##### GH Filter #####
    if receptor_index_2[emisor_id] > 0:
        if len(ultrasound_pulse_durations_2[emisor_id]) > window_size:
            pulse_mean = np.mean(ultrasound_pulse_durations_2[emisor_id][-window_size:])
            pulse_std = np.std(ultrasound_pulse_durations_2[emisor_id][-window_size:])
            if abs(pulse_duration-pulse_mean) > std_gain*pulse_std:
                pulse_duration = pulse_duration_est_2[emisor_id]

        dt = time - time_2_prev[emisor_id]
        pulse_duration_x_pred = pulse_duration_est_2[emisor_id] + (pulse_duration_dx_2[emisor_id]*dt)
        residual = pulse_duration - pulse_duration_x_pred
        pulse_duration_dx_2[emisor_id] = pulse_duration_dx_2[emisor_id] + ( H_ULTRASOUND * residual / dt)
        pulse_duration_est_2[emisor_id] = pulse_duration_x_pred + G_ULTRASOUND * residual
    else:
```

```
pulse_duration_est_2[emisor_id] = pulse_duration
time_2_prev[emisor_id] = time
receptor_index_2[emisor_id] += 1
ultrasound_times_2[emisor_id].append(time)
ultrasound_pulse_durations_2[emisor_id].append(pulse_duration_est_2[emisor_id])
pulse_distance = pulse_duration_est_2[emisor_id]*TIME_TO_DISTANCE
ultrasound_pulse_distances_2[emisor_id].append(pulse_distance)

##### COORDINATES #####
trilateration_3d_points_2[emisor_id] = (emisor_x, emisor_y, emisor_z)
trilateration_3d_distances_2[emisor_id] = pulse_distance
if receptor_index_2[emisor_id] > 3:
    comb_index = 0
    receptor_x_estimates = []
    receptor_y_estimates = []
    receptor_z_estimates = []
    for combination in combinations:
        emisor_1 = combination[0]
        emisor_2 = combination[1]
        emisor_3 = combination[2]
        (receptor_x_estimate, receptor_y_estimate, receptor_z_estimate) = trilateration3Points(trilateration_3d_points_2[emisor_1],
trilateration_3d_points_2[emisor_2], trilateration_3d_points_2[emisor_3],
trilateration_3d_distances_2[emisor_1], trilateration_3d_distances_2[emisor_2], trilateration_3d_distances_2[emisor_3])
        if comb_index == 1:
            receptor_y_estimate = -1.0*receptor_y_estimate
        elif comb_index == 2:
            receptor_x_estimate = -1.0*receptor_x_estimate
            receptor_z_estimate = -1.0*receptor_z_estimate
        elif comb_index == 3:
            receptor_x_estimate = -1.0*receptor_x_estimate
            receptor_y_estimate = -1.0*receptor_y_estimate
            receptor_z_estimate = -1.0*receptor_z_estimate
        comb_index += 1
        receptor_x_estimates.append(receptor_x_estimate)
        receptor_y_estimates.append(receptor_y_estimate)
        receptor_z_estimates.append(receptor_z_estimate - 1.25)

    ultrasound_coordinates_x_2.append(np.mean(receptor_x_estimates))
    ultrasound_coordinates_y_2.append(np.mean(receptor_y_estimates))
    ultrasound_coordinates_z_2.append(np.mean(receptor_z_estimates))
    ultrasound_coordinates_time_2.append(time)

##### ULTRASOUND #####
imu_data_1 = imu_data[np.where(imu_data[:,0] == 1), :][0]
```

```
imu_data_1[:,2] = (imu_data_1[:,2] - 1.0*acc_x_mean_1)*gravity
imu_data_1[:,3] = (imu_data_1[:,3] - 1.0*acc_y_mean_1)*gravity
imu_data_1[:,4] = (imu_data_1[:,4] - 1.0*acc_z_mean_1)*gravity
imu_data_1[:,5] = imu_data_1[:,5] - 1.0*gyro_x_mean_1
imu_data_1[:,6] = imu_data_1[:,6] - 1.0*gyro_y_mean_1
imu_data_1[:,7] = imu_data_1[:,7] - 1.0*gyro_z_mean_1
imu_data_1[:,2] = filterGH(imu_data_1[:,0], imu_data_1[:,2], imu_data_1[:,2][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_1[:,3] = filterGH(imu_data_1[:,0], imu_data_1[:,3], imu_data_1[:,3][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_1[:,4] = filterGH(imu_data_1[:,0], imu_data_1[:,4], imu_data_1[:,4][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_1[:,5] = filterGH(imu_data_1[:,0], imu_data_1[:,5], imu_data_1[:,5][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_1[:,6] = filterGH(imu_data_1[:,0], imu_data_1[:,6], imu_data_1[:,6][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_1[:,7] = filterGH(imu_data_1[:,0], imu_data_1[:,7], imu_data_1[:,7][0], dx = 0.0, g=G_IMU, h=H_IMU)
```

```
imu_data_2 = imu_data[np.where(imu_data[:,0] == 2), :][0]
imu_data_2[:,2] = (imu_data_2[:,2] - 1.0*acc_x_mean_2)*gravity
imu_data_2[:,3] = (imu_data_2[:,3] - 1.0*acc_y_mean_2)*gravity
imu_data_2[:,4] = (imu_data_2[:,4] - 1.0*acc_z_mean_2)*gravity
imu_data_2[:,5] = imu_data_2[:,5] - 1.0*gyro_x_mean_2
imu_data_2[:,6] = imu_data_2[:,6] - 1.0*gyro_y_mean_2
imu_data_2[:,7] = imu_data_2[:,7] - 1.0*gyro_z_mean_2
imu_data_2[:,2] = filterGH(imu_data_2[:,0], imu_data_2[:,2], imu_data_2[:,2][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_2[:,3] = filterGH(imu_data_2[:,0], imu_data_2[:,3], imu_data_2[:,3][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_2[:,4] = filterGH(imu_data_2[:,0], imu_data_2[:,4], imu_data_2[:,4][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_2[:,5] = filterGH(imu_data_2[:,0], imu_data_2[:,5], imu_data_2[:,5][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_2[:,6] = filterGH(imu_data_2[:,0], imu_data_2[:,6], imu_data_2[:,6][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_2[:,7] = filterGH(imu_data_2[:,0], imu_data_2[:,7], imu_data_2[:,7][0], dx = 0.0, g=G_IMU, h=H_IMU)
```

```
imu_data_3 = imu_data[np.where(imu_data[:,0] == 3), :][0]
imu_data_3[:,2] = (imu_data_3[:,2] - 1.0*acc_x_mean_3)*gravity
imu_data_3[:,3] = (imu_data_3[:,3] - 1.0*acc_y_mean_3)*gravity
imu_data_3[:,4] = (imu_data_3[:,4] - 1.0*acc_z_mean_3)*gravity
imu_data_3[:,5] = imu_data_3[:,5] - 1.0*gyro_x_mean_3
imu_data_3[:,6] = imu_data_3[:,6] - 1.0*gyro_y_mean_3
imu_data_3[:,7] = imu_data_3[:,7] - 1.0*gyro_z_mean_3
imu_data_3[:,2] = filterGH(imu_data_3[:,0], imu_data_3[:,2], imu_data_3[:,2][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_3[:,3] = filterGH(imu_data_3[:,0], imu_data_3[:,3], imu_data_3[:,3][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_3[:,4] = filterGH(imu_data_3[:,0], imu_data_3[:,4], imu_data_3[:,4][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_3[:,5] = filterGH(imu_data_3[:,0], imu_data_3[:,5], imu_data_3[:,5][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_3[:,6] = filterGH(imu_data_3[:,0], imu_data_3[:,6], imu_data_3[:,6][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_3[:,7] = filterGH(imu_data_3[:,0], imu_data_3[:,7], imu_data_3[:,7][0], dx = 0.0, g=G_IMU, h=H_IMU)
```

```
imu_data_4 = imu_data[np.where(imu_data[:,0] == 4), :][0]
```

```

imu_data_4[:,2] = (imu_data_4[:,2] - 1.0*acc_x_mean_4)*gravity
imu_data_4[:,3] = (imu_data_4[:,3] - 1.0*acc_y_mean_4)*gravity
imu_data_4[:,4] = (imu_data_4[:,4] - 1.0*acc_z_mean_4)*gravity
imu_data_4[:,5] = imu_data_4[:,5] - 1.0*gyro_x_mean_4
imu_data_4[:,6] = imu_data_4[:,6] - 1.0*gyro_y_mean_4
imu_data_4[:,7] = imu_data_4[:,7] - 1.0*gyro_z_mean_4

imu_data_4[:,2] = filterGH(imu_data_4[:,0], imu_data_4[:,2], imu_data_4[:,2][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_4[:,3] = filterGH(imu_data_4[:,0], imu_data_4[:,3], imu_data_4[:,3][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_4[:,4] = filterGH(imu_data_4[:,0], imu_data_4[:,4], imu_data_4[:,4][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_4[:,5] = filterGH(imu_data_4[:,0], imu_data_4[:,5], imu_data_4[:,5][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_4[:,6] = filterGH(imu_data_4[:,0], imu_data_4[:,6], imu_data_4[:,6][0], dx = 0.0, g=G_IMU, h=H_IMU)
imu_data_4[:,7] = filterGH(imu_data_4[:,0], imu_data_4[:,7], imu_data_4[:,7][0], dx = 0.0, g=G_IMU, h=H_IMU)

print('.....IMU1.....')
print('acc_x_mean_1=', np.mean(imu_data_1[:,2]), ' acc_y_mean_1=', np.mean(imu_data_1[:,3]), ' acc_z_mean_1=',
np.mean(imu_data_1[:,4]))
print('gyro_x_mean_1=', np.mean(imu_data_1[:,5]), ' gyro_y_mean_1=', np.mean(imu_data_1[:,6]), ' gyro_z_mean_1=',
np.mean(imu_data_1[:,7]))

print('.....IMU2.....')
print('acc_x_mean_2=', np.mean(imu_data_2[:,2]), ' acc_y_mean_2=', np.mean(imu_data_2[:,3]), ' acc_z_mean_2=',
np.mean(imu_data_2[:,4]))
print('gyro_x_mean_2=', np.mean(imu_data_2[:,5]), ' gyro_y_mean_2=', np.mean(imu_data_2[:,6]), ' gyro_z_mean_2=',
np.mean(imu_data_2[:,7]))

print('.....IMU3.....')
print('acc_x_mean_3=', np.mean(imu_data_3[:,2]), ' acc_y_mean_3=', np.mean(imu_data_3[:,3]), ' acc_z_mean_3=',
np.mean(imu_data_3[:,4]))
print('gyro_x_mean_3=', np.mean(imu_data_3[:,5]), ' gyro_y_mean_3=', np.mean(imu_data_3[:,6]), ' gyro_z_mean_3=',
np.mean(imu_data_3[:,7]))

print('.....IMU4.....')
print('acc_x_mean_4=', np.mean(imu_data_4[:,2]), ' acc_y_mean_4=', np.mean(imu_data_4[:,3]), ' acc_z_mean_4=',
np.mean(imu_data_4[:,4]))
print('gyro_x_mean_4=', np.mean(imu_data_4[:,5]), ' gyro_y_mean_4=', np.mean(imu_data_4[:,6]), ' gyro_z_mean_4=',
np.mean(imu_data_4[:,7]))

imu_1_attitude = estimateIMUAttitude(sensor_data = imu_data_1, noise_data = imu_calibration_noise_data[0])
imu_2_attitude = estimateIMUAttitude(sensor_data = imu_data_2, noise_data = imu_calibration_noise_data[1])
imu_3_attitude = estimateIMUAttitude(sensor_data = imu_data_3, noise_data = imu_calibration_noise_data[2])
imu_4_attitude = estimateIMUAttitude(sensor_data = imu_data_4, noise_data = imu_calibration_noise_data[3])

##### PLOT #####
# plot Ultrasound pulses/distances

```

```
plt.figure(1)
for emisor_id in range(4):
    plt.subplot(4,1,emisor_id+1)
    #plt.plot(ultrasound_times_1[emisor_id][window_size:],      ultrasound_pulse_durations_1[emisor_id][window_size:],      color='b',
linewidth=3)
    plt.plot(ultrasound_times_1[emisor_id][window_size:],      ultrasound_pulse_distances_1[emisor_id][window_size:],      color='b',
linewidth=3)
    #plt.ylabel('Pulse time (ms)')
    plt.ylabel('Distance (mm)')
    plt.title('Ultrasound 1 / Emisor ' + str(emisor_id+1))
    plt.subplots_adjust(hspace=0.5)
plt.grid(True)

plt.figure(2)
for emisor_id in range(4):
    plt.subplot(4,1,emisor_id+1)
    #plt.plot(ultrasound_times_2[emisor_id][window_size:
    # ], ultrasound_pulse_durations_2[emisor_id][window_size:], color='r', linewidth=3)
    plt.plot(ultrasound_times_2[emisor_id][window_size:],      ultrasound_pulse_distances_2[emisor_id][window_size:],      color='r',
linewidth=3)
    #plt.ylabel('Pulse time (ms)')
    plt.ylabel('Distance (mm)')
    plt.title('Ultrasound 2 / Emisor ' + str(emisor_id+1))
    plt.subplots_adjust(hspace=0.5)
plt.grid(True)

# plot Z and roll
plt.figure(3)
plt.subplot(2,1,1)
plt.plot(ultrasound_coordinates_time_1, ultrasound_coordinates_z_1, color='b', linewidth=3)
plt.plot(ultrasound_coordinates_time_2, ultrasound_coordinates_z_2, color='r', linewidth=3)
plt.ylabel('Receptor z (mm)')
plt.title('Receptor z ')
plt.subplots_adjust(hspace=0.5)
plt.grid(True)
plt.subplot(2,1,2)
#plt.plot([y - x for x, y in zip(ultrasound_coordinates_z_1, ultrasound_coordinates_z_2)], color='b', linewidth=3)
estimated_angle = [-1.0*np.arctan2((y - x), L)*180.0/np.pi for x, y in zip(ultrasound_coordinates_z_1, ultrasound_coordinates_z_2)]
plt.plot(ultrasound_coordinates_time_2[0:len(estimated_angle)], estimated_angle, color='y', linewidth=3)
plt.plot(imu_data_1[:,1], imu_1_attitude[:,1], color='b', linewidth=1)
plt.plot(imu_data_2[:,1], imu_2_attitude[:,1], color='r', linewidth=1)
plt.plot(imu_data_3[:,1], imu_3_attitude[:,1], color='k', linewidth=1)
plt.plot(imu_data_4[:,1], imu_4_attitude[:,1], color='g', linewidth=1)
plt.title('Roll angle (°)')
plt.subplots_adjust(hspace=0.5)
```

```
plt.grid(True)
```

```
# plot IMU
```

```
plt.figure(4)
```

```
plt.subplot(3,1,1)
```

```
plt.plot(imu_data_1[:,1], imu_data_1[:,2], color='r', label='Ultrasound 1')
```

```
plt.plot(imu_data_2[:,1], imu_data_2[:,2], color='g', label='Ultrasound 2')
```

```
plt.plot(imu_data_3[:,1], imu_data_3[:,2], color='b', label='Ultrasound 3')
```

```
plt.plot(imu_data_4[:,1], imu_data_4[:,2], color='k', label='Ultrasound 4')
```

```
plt.xlabel("Time (s)")
```

```
plt.ylabel('Acc X')
```

```
plt.title("Acceleration X")
```

```
plt.subplots_adjust(hspace=0.5)
```

```
plt.grid(True)
```

```
plt.subplot(3,1,2)
```

```
plt.plot(imu_data_1[:,1], imu_data_1[:,3], color='r', label='Ultrasound 1')
```

```
plt.plot(imu_data_2[:,1], imu_data_2[:,3], color='g', label='Ultrasound 2')
```

```
plt.plot(imu_data_3[:,1], imu_data_3[:,3], color='b', label='Ultrasound 3')
```

```
plt.plot(imu_data_4[:,1], imu_data_4[:,3], color='k', label='Ultrasound 4')
```

```
plt.xlabel("Time (s)")
```

```
plt.ylabel('Acc Y')
```

```
plt.title("Acceleration Y")
```

```
plt.subplots_adjust(hspace=0.5)
```

```
plt.grid(True)
```

```
plt.subplot(3,1,3)
```

```
plt.plot(imu_data_1[:,1], imu_data_1[:,4], color='r', label='Ultrasound 1')
```

```
plt.plot(imu_data_2[:,1], imu_data_2[:,4], color='g', label='Ultrasound 2')
```

```
plt.plot(imu_data_3[:,1], imu_data_3[:,4], color='b', label='Ultrasound 3')
```

```
plt.plot(imu_data_4[:,1], imu_data_4[:,4], color='k', label='Ultrasound 4')
```

```
plt.xlabel("Time (s)")
```

```
plt.ylabel('Acc Z')
```

```
plt.title("Acceleration Z")
```

```
plt.subplots_adjust(hspace=0.5)
```

```
plt.grid(True)
```

```
plt.figure(5)
```

```
plt.subplot(3,1,1)
```

```
plt.plot(imu_data_1[:,1], imu_data_1[:,5], color='r', label='Ultrasound 1')
```

```
plt.plot(imu_data_2[:,1], imu_data_2[:,5], color='g', label='Ultrasound 2')
```

```
plt.plot(imu_data_3[:,1], imu_data_3[:,5], color='b', label='Ultrasound 3')
```

```
plt.plot(imu_data_4[:,1], imu_data_4[:,5], color='k', label='Ultrasound 4')
```

```
plt.xlabel("Time (s)")
```

```
plt.ylabel('Pitch rate')
```

```
plt.title('Pitch rate')
plt.subplots_adjust(hspace=0.5)
plt.grid(True)

plt.subplot(3,1,2)
plt.plot(imu_data_1[:,1], imu_data_1[:,6], color='r', label='Ultrasound 1')
plt.plot(imu_data_2[:,1], imu_data_2[:,6], color='g', label='Ultrasound 2')
plt.plot(imu_data_3[:,1], imu_data_3[:,6], color='b', label='Ultrasound 3')
plt.plot(imu_data_4[:,1], imu_data_4[:,6], color='k', label='Ultrasound 4')
plt.xlabel('Time (s)')
plt.ylabel('Roll rate')
plt.title('Roll rate')
plt.subplots_adjust(hspace=0.5)
plt.grid(True)

plt.subplot(3,1,3)
plt.plot(imu_data_1[:,1], imu_data_1[:,7], color='r', label='Ultrasound 1')
plt.plot(imu_data_2[:,1], imu_data_2[:,7], color='g', label='Ultrasound 2')
plt.plot(imu_data_3[:,1], imu_data_3[:,7], color='b', label='Ultrasound 3')
plt.plot(imu_data_4[:,1], imu_data_4[:,7], color='k', label='Ultrasound 4')
plt.xlabel('Time (s)')
plt.ylabel('Yaw rate')
plt.title('Yaw rate')
plt.subplots_adjust(hspace=0.5)
plt.grid(True)

plt.show()
```

ANEXO V: CÓDIGO PYTHON POSICIONAMIENTO 3D

```
# -*- coding: utf-8 -*-
import sys
import numpy as np

def trilateration3Points(P1, P2, P3, r1, r2, r3):

    p1 = np.array([0, 0, 0])
    p2 = np.array([P2[0] - P1[0], P2[1] - P1[1], P2[2] - P1[2]])
    p3 = np.array([P3[0] - P1[0], P3[1] - P1[1], P3[2] - P1[2]])
    v1 = p2 - p1
    v2 = p3 - p1

    Xn = (v1)/np.linalg.norm(v1)
    tmp = np.cross(v1, v2)
    Zn = (tmp)/np.linalg.norm(tmp)
    Yn = np.cross(Xn, Zn)
    i = np.dot(Xn, v2)
    d = np.dot(Xn, v1)
    j = np.dot(Yn, v2)
    X = ((r1**2)-(r2**2)+(d**2))/(2*d)
    Y = (((r1**2)-(r3**2)+(i**2)+(j**2))/(2*j))-((i/j)*(X))
    Z1 = np.sqrt(abs(r1**2-X**2-Y**2))
    Z2 = np.sqrt(abs(r1**2-X**2-Y**2))*(-1)
    K1 = P1 + X*Xn + Y * Yn + Z1 * Zn
    K2 = p1 + X * Xn + Y * Yn - Z2 * Zn
    return K1
```

ANEXO VI: CÓDIGO PYTHON REGRESIÓN TIEMPO DE VUELO A DISTANCIA

```
# -*- coding: utf-8 -*-
from os import listdir
import numpy as np
#from numpy import genfromtxt
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

def generateRegression(regression_data, show_plot = False):
    x = regression_data[:,6].reshape(-1, 1)
    y = regression_data[:,7].reshape(-1, 1)
    model = LinearRegression()
    model.fit(x, y)
    slope = model.coef_[0][0]
    bias = model.intercept_[0]
    print('slope=', slope, ' bias=' , bias)
    if show_plot:
        predicted_pulse_time = bias[0] + slope[0]*x
        plt.figure(10)
        plt.subplot(1,1,1)
        plt.plot(x, y, '.b')
        plt.plot(x, predicted_pulse_time, '-r')
        plt.grid(True)
        plt.xlabel('Distance (mm)')
        plt.ylabel('Pulse time mean (ms)')
        plt.title('Distance mean regression')
        plt.subplots_adjust(hspace=0.5)
        plt.show()
    return (bias, slope)
```

7 ANEXO VII: CÓDIGO PYTHON FILTRO GH

```
# -*- coding: utf-8 -*-
import numpy as np

def isNaN(num):
    return num != num

def filterGH(time, data, x0, dx, g, h):
    x_est = x0
    results = []
    index = 0
    len_data = len(data)
    for z in data:

        # prediction step
        if index < len_data - 1:
            dt = time[index+1] - time[index]
            x_pred = x_est + (dx*dt)

        # update step
        residual = z - x_pred
        dx = dx + h * (residual) / dt
        x_est = x_pred + g * residual
        results.append(x_est)
    return np.array(results)
```