



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Visualización geoespacial en gafas de realidad aumentada de
bajo coste*

Grado en Ingeniería Mecánica

ALUMNO: Mariano Jesús Esteban López

DIRECTOR: Miguel Rodelgo Lacruz

CURSO ACADÉMICO: 2025-2026

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Visualización geoespacial en gafas de realidad aumentada de
bajo coste*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

El presente trabajo consiste en el desarrollo de un sistema de Realidad Aumentada (RA) de bajo coste diseñado para mejorar la vigilancia marítima y la seguridad de la navegación. El objetivo principal ha sido visualizar datos del AIS superpuestos directamente sobre el mundo real mediante un visor óptico transparente, evitando que el usuario tenga que desviar la mirada hacia las pantallas convencionales y reduciendo la carga cognitiva que supone integrar mentalmente los símbolos cartográficos con el escenario real.

El desarrollo se ha dividido en tres fases técnicas. Primero, se configuró el hardware basado en las gafas XREAL Air 2 y la unidad de computación XREAL Beam Pro, integrando sus sensores de posición y orientación con el motor gráfico Unity. En segundo lugar, se implementó la recepción inalámbrica de datos AIS mediante un receptor SDR, desarrollando un algoritmo para decodificar los mensajes NMEA y extraer variables como el nombre, MMSI, rumbo y velocidad de los buques. Finalmente, se diseñó una interfaz que ancla estos datos geoespacialmente en el campo visual del usuario.

Las pruebas realizadas confirman que el sistema permite una identificación intuitiva de contactos marítimos en tiempo real. Aunque se identificaron limitaciones en la precisión de la brújula y el GPS, el proyecto demuestra la viabilidad de utilizar tecnología comercial de bajo coste para aplicaciones de ayuda a la navegación.

PALABRAS CLAVE

Realidad Aumentada, AIS, Unity, XREAL, Seguridad Marítima.

AGRADECIMIENTOS

En primer lugar me gustaría agradecer a todos los compañeros que me han acompañado a lo largo de estos años de estancia en la ENM, con especial cariño para la camareta 102 que ha hecho que el trabajo a realizar siempre sea más ameno.

Agradezco especialmente a mi tutor, D. Miguel Rodelgo Lacruz, el cual ha demostrado paciencia y compromiso durante este cuatrimestre, aclarando las interminables dudas que me surgían a cada momento.

Por último, no me puedo olvidar de la parte fundamental de este éxito, mi familia. A mis padres por ser mis referentes siempre y apoyarme tanto en los momentos buenos como en los más bajos, sin ellos y mis hermanos sé que no hubiese sido posible. Asimismo tampoco me puedo olvidar de los que ya no están, que sé que me han apoyado y serían los que más orgullosos estarían.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	7
1.1 Contexto y motivación	7
1.2 Objetivos	8
1.3 Estructura de la memoria	9
2 Estado del arte	11
2.1 Sistema de Identificación Automática (AIS)	11
2.2 Realidad aumentada	13
2.2.1 Evolución histórica y usos	13
2.2.2 Realidad aumentada geoespacial	14
2.3 <i>Hardware</i> de visualización	14
2.3.1 Arquitecturas ópticas	14
2.3.2 Tecnologías de micro-display	15
2.4 Sensores	16
2.4.1 Sensores de posicionamiento y rastreo	16
2.4.2 Grados de libertad.....	17
2.5 Ecosistema de <i>software</i> y herramientas de desarrollo	18
2.5.1 Frameworks y plugins de Realidad Aumentada	18
2.6 Análisis de mercado y selección de <i>hardware</i>	19
2.7 Unidad de computación	21
2.7.1 Dispositivos basados en Android.....	21
2.7.2 XREAL Beam Pro	21
2.8 Motores de Desarrollo (IDE)	22
3 Desarrollo del TFG.....	25
3.1 Arquitectura <i>hardware</i>	25
3.1.1 Gafas XREAL Air 2 y dispositivo XREAL Beam Pro.....	26
3.1.2 Sistema de recepción AIS: Antena VHF y Receptor SDR	27
3.2 Entorno de desarrollo	28
3.2.1 Configuración de Unity y XREAL SDK	28
3.2.2 Estructura de la escena y gestión de Scripts	30
3.2.3 Consideraciones y limitaciones en la captura del entorno inmersivo	31

3.3 Fase 1: Acceso y calibración de sensores de navegación	31
3.3.1 Activación y lectura del magnetómetro (Brújula)	31
3.3.2 Validación del sistema de posicionamiento global (GPS).....	32
3.4 Fase 2: Posicionamiento geoespacial estático y validación empírica	33
3.4.1 Prueba de concepto y calibración de precisión	33
3.4.2 Interfaz HUD y escalabilidad mediante múltiples marcadores	34
3.5 Fase 3: Integración de telemetría marítima dinámica (AIS)	36
3.5.1 Arquitectura de red y recepción asíncrona de datos (UDP).....	37
3.5.2 Decodificación del protocolo NMEA/AIVDM y procesamiento a nivel de bit	39
3.5.3 Fusión de sensores y desacoplamiento espacial en arquitectura tethered.....	41
3.5.4 Representación inmersiva y seguimiento espacial (Tracker)	42
3.5.5 Interfaz contextual y carga dinámica de activos gráficos	43
4 Resultados y Validación	45
4.1 Escenario de pruebas y metodología.....	45
4.2 Visualización y legibilidad de la interfaz.....	46
4.3 Precisión del posicionamiento espacial.....	46
4.4 Rendimiento del flujo de datos	49
5 Conclusiones y líneas futuras	51
5.1 Conclusiones	51
5.2 Líneas futuras	52
6 Bibliografía.....	55
Anexo I: Implicaciones Sociales, y/o Económicas, y/o Ambientales	59
Anexo II: Reflexiones Éticas y Sociales	61
Anexo III: Uso de la inteligencia artificial	63
Anexo IV: CompassTest.cs	65
Anexo V: Código SensorIntegrationHUD.cs	67
Anexo VI: Código GeoMarkerTest.cs.....	69
Anexo VII: Código MultiMarkerTracker.cs.....	73
Anexo VIII: Código UDP_AIS_Receiver.cs.....	77
Anexo IX: Código AISParser.cs	79
Anexo X: Código AIS_AR_Tracker.cs.....	83
Anexo XI: Código ShipMarkerUI.cs.....	89

ÍNDICE DE FIGURAS

Figura 1-1 Representación de la <i>Situational Awareness</i> (Generado por Inteligencia Artificial)	7
Figura 2-1 Diagrama conceptual de la arquitectura de red y transmisión del sistema AIS [40]	12
Figura 2-2 Sistema " <i>Sword of Damocles</i> " de Ivan Sutherland [41]	14
Figura 2-3 Esquema de la arquitectura óptica "Birdbath" mediante divisor de haz. [42]	15
Figura 2-4 Diagrama 3DoF vs 6DoF [43]	18
Figura 2-5 Gafas XREAL Air 2 [44]	19
Figura 3-1 Arquitectura del <i>hardware</i> (Elaboración propia)	25
Figura 3-2 Gafas XREAL Air 2 con el dispositivo XREAL Beam Pro (Elaboración propia)	26
Figura 3-3 Antena SDR (Elaboración propia)	27
Figura 3-4 Configuración Active Input Handling (Elaboración propia)	29
Figura 3-5 Configuración API gráfica (Elaboración propia)	29
Figura 3-6 Interfaz de la aplicación Unity (Elaboración propia)	30
Figura 3-7 Inicialización de la brújula y servicios de ubicación (Elaboración propia)	32
Figura 3-8 Representación de los datos sobre la pantalla del XREAL Beam Pro (Elaboración propia)	33
Figura 3-9 Implementación de la fórmula del Haversine para el cálculo de distancia y demora (Elaboración propia)	34
Figura 3-10 Definición de la estructura de datos <i>GeoWaypoint</i> para la gestión de marcadores (Elaboración propia)	35
Figura 3-11 Estructuración dinámica del panel de datos de navegación (HUD) (Elaboración propia)	36
Figura 3-12 Esquema gráfico de la disposición de red (Generado por inteligencia artificial)	37
Figura 3-13 Configuración del hilo secundario (Multithreading) para escucha UDP (Elaboración propia)	38
Figura 3-14 Control de concurrencia y extracción segura de tramas en el ciclo <i>Update</i> (Elaboración propia)	39
Figura 3-15 Algoritmo de conversión de carga útil AIS a formato binario de 6 bits (Elaboración propia)	39
Figura 3-16 Lógica de segmentación de bits para la extracción de variables MMSI y posición (Elaboración propia)	40
Figura 3-17 Método de decodificación de texto alfanumérico para nombres de buques (Elaboración propia)	41
Figura 3-18 Algoritmo de fusión de sensores para el cálculo del Rumbo de la Mirada (<i>Gaze Heading</i>) (Elaboración propia)	42
Figura 3-19 Lógica de desacoplamiento espacial y rotación tipo <i>Billboarding</i> (Elaboración propia)	42

Figura 3-20 Gestión de la base de datos de buques activos mediante Diccionarios (C#) (Elaboración propia)	43
Figura 3-21 Carga dinámica de activos gráficos (banderas) basada en el código MID (Elaboración propia)	44
Figura 4-1 Oficial alumno de la ENM durante la prueba de visualización (Elaboración propia)....	45
Figura 4-2 Interfaz lateral mostrando los contactos AIS detectados, capturada utilizando el filtro oscuro para mejorar el contraste (Elaboración propia)	46
Figura 4-3 Lanchas de instrucción de la Escuela Naval durante la maniobra de ataque (Elaboración propia)	47
Figura 4-4 Comparativa del efecto de actualización de la trama AIS (Elaboración propia).....	48
Figura 4-5 Superposición de marcadores (Elaboración propia)	48

ÍNDICE DE TABLAS

Tabla 1 Análisis de mercado del <i>hardware</i> (elaboración propia)	20
Tabla 2 Coordenadas geográficas de los marcadores empleados en la validación espacial. (Elaboración propia)	36

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

En el ámbito de la navegación marítima, la seguridad y la eficacia de las operaciones dependen fundamentalmente de la conciencia situacional (*situational awareness*) del equipo del puente de mando (Figura 1-1). Comprender el entorno, anticipar los movimientos del tráfico marítimo circundante y tomar decisiones ágiles son factores críticos. Para facilitar esta labor, el Sistema de Identificación Automática (AIS por sus siglas en inglés) se ha consolidado como una herramienta indispensable, proporcionando datos cinemáticos y de identificación en tiempo real sobre los buques cercanos.



Figura 1-1 Representación de la *Situational Awareness* (Generado por Inteligencia Artificial)

Históricamente, la información proveniente del AIS, así como del radar y las cartas náuticas electrónicas (ENC por sus siglas en inglés), se presenta en pantallas y consolas integradas en el puente. Sin embargo, esta disposición obliga al operador a desviar constantemente la mirada del horizonte marino para consultar los monitores. En el contexto de las Fuerzas Armadas y, más concretamente, en

las operaciones de la Armada, donde la rapidez de reacción en entornos tácticos o de navegación restringida es un factor determinante, esta división de la atención representa una vulnerabilidad potencial. A esto se suma la elevada carga cognitiva que supone para el operador el tener que integrar mentalmente los símbolos cartográficos mostrados en las pantallas con el escenario real, ralentizando la toma de decisiones.

La tecnología de Realidad Aumentada (RA) emerge como una solución disruptiva ante este desafío. Mediante el uso de visores ópticos transparentes (*Optical See-Through*), es posible superponer la información digital del AIS directamente sobre el campo visual del usuario, anclando los datos geográficamente en el horizonte real. Esto permite al usuario mantener la cabeza erguida creando un *Head-Up Display* (HUD), fusionando la observación visual del mar con los datos telemétricos sin perder detalle del entorno físico.

El presente Trabajo Fin de Grado (TFG) se plantea como la continuación y evolución técnica de diversas líneas de investigación desarrolladas previamente en este centro universitario, orientadas a la decodificación de señales marítimas y su visualización inmersiva [1] [2]. Estos proyectos previos sentaron las bases para la recepción y decodificación de las tramas de datos, empleando como entorno de desarrollo el motor gráfico Unity y el sistema operativo Android. No obstante, el avance tecnológico continuo exige adaptar estas soluciones a nuevo *hardware*. Tradicionalmente, los dispositivos de RA con capacidades avanzadas de integración, como las gafas Microsoft HoloLens, han presentado costes elevados y unas dimensiones que dificultan su uso operativo prolongado. No obstante, la evolución del mercado ha propiciado la aparición de *hardware* de RA más asequible, aunque con capacidades técnicas más limitadas que los sistemas de alta gama. Estas tecnologías *Commercial Off-The-Shelf* (COTS) de electrónica de consumo orientadas fundamentalmente a la visualización multimedia (videojuegos, reproducción de vídeo, etc.), ofrecen diseños más ligeros y ergonómicos. Algunos ejemplos representativos de estos dispositivos son los modelos desarrollados por marcas como XREAL, Rokid o Viture. Por consiguiente, resulta de interés investigar y desarrollar aplicaciones experimentales orientadas a evaluar las posibilidades reales de esta tecnología. El propósito es analizar su viabilidad en escenarios de uso dual (civil y militar), con posibles aplicaciones concretas tales como la monitorización de datos tácticos en el puente de mando, la asistencia a la navegación en aguas restringidas o el apoyo visual directo en operaciones de búsqueda y salvamento.

1.2 Objetivos

El objetivo principal de este Trabajo Fin de Grado es desarrollar, adaptar y validar un sistema de visualización de datos AIS en tiempo real utilizando gafas de realidad aumentada de bajo coste, mejorando la inmersión, la ergonomía y la precisión geoespacial respecto a los desarrollos previos.

Para alcanzar este propósito general, se definen los siguientes objetivos específicos:

1. **Analizar el estado del arte** de las tecnologías de visualización en realidad aumentada aplicables al entorno marítimo, justificando la selección de una arquitectura de *hardware* comercial de bajo coste.
2. **Implementar un sistema de posicionamiento geoespacial** preciso que utilice los sensores del dispositivo de computación para calcular la posición relativa de los blancos AIS y proyectarlos correctamente en el horizonte visual del usuario, mitigando los efectos del movimiento del buque.
3. **Adaptar y optimizar la lógica computacional** heredada de la fase anterior del proyecto para la ingesta y decodificación de tramas NMEA (*National Marine and Electronics Association*) procedentes del receptor AIS, asegurando su compatibilidad y fluidez en el nuevo ecosistema de *software* (Android/NebulaOS).

4. **Diseñar una interfaz de usuario (UI por sus siglas en inglés) ergonómica y no intrusiva**, que permita una lectura clara de los datos críticos como el *Maritime Mobile Service Identity* (MMSI), el rumbo o la velocidad entre otros, garantizando la transparencia necesaria para no obstaculizar la visión del mar real.
5. **Realizar pruebas de validación** del sistema integrado para evaluar su rendimiento técnico, la legibilidad bajo diferentes condiciones de luminosidad y la usabilidad del *hardware* en escenarios reales.

1.3 Estructura de la memoria

Tras haber definido el contexto del presente proyecto, y habiendo definido sus objetivos, se describe a continuación la estructura que seguirá la memoria, con el fin de organizar el desarrollo realizado y los resultados obtenidos de la manera más clara posible.

En el Capítulo 1 se presenta una introducción al trabajo, contextualizando el proyecto, exponiendo la motivación que lo impulsa y definiendo los objetivos que se persiguen con su desarrollo.

En el Capítulo 2 se presenta una revisión del marco teórico en el que se engloba el trabajo con la revisión del estado del arte respecto a:

- El sistema AIS, su importancia en el mundo marítimo y su seguridad, así como una explicación de la estructura de los datos NMEA proporcionados.
- La Realidad Aumentada, desglosando el *hardware* de visualización (arquitecturas ópticas y pantallas), los sensores de posicionamiento, un análisis de mercado de gafas de bajo coste (justificando la elección de las XREAL Air 2 y el dispositivo XREAL Beam Pro) y el ecosistema de *software* empleado.

En el Capítulo 3 se detalla el desarrollo del TFG. Se justifica y explica la configuración del *hardware* utilizado, así como la integración del entorno Unity con el SDK de XREAL. Además, se detalla el proceso seguido para la obtención y decodificación de las tramas de datos AIS, y se describe el enfoque adoptado para lograr la representación geoespacial en tiempo real anclada al horizonte. Finalmente, se explican las decisiones de diseño de la interfaz holográfica implementadas para optimizar la aplicación.

En el Capítulo 4, se muestran los resultados obtenidos y se procede a su análisis. Se exponen las prestaciones de la aplicación alcanzadas evaluando la precisión del posicionamiento virtual y la legibilidad de la información mostrada a través del visor, y se discuten sus limitaciones.

En el Capítulo 5 se presentan las conclusiones y las líneas futuras. Se plantea y discute el desarrollo completo de la aplicación y sus funcionalidades, con las posibles tareas que se podrían realizar para alcanzar una versión mejorada.

Por último, se adjunta la bibliografía utilizada a lo largo del trabajo y los anexos con información complementaria con el propósito de apoyar la comprensión del desarrollo realizado.

2 ESTADO DEL ARTE

El presente capítulo tiene como objetivo contextualizar las tecnologías y sistemas que componen este proyecto. Se abordará, en primer lugar, el sistema de transmisión de datos marítimos (AIS), fundamental para la obtención de la información a visualizar. Posteriormente, se analizará la tecnología *hardware* de RA, desglosando los tipos de pantallas y sensores que permiten la inmersión. A continuación, se realizará un estudio de mercado de los dispositivos de bajo coste actuales, justificando la elección de las gafas XREAL Air 2 combinadas con el dispositivo de computación XREAL Beam Pro. Finalmente, se evaluarán los entornos de desarrollo de *software* disponibles, argumentando la selección de Unity como motor gráfico.

2.1 Sistema de Identificación Automática (AIS)

La seguridad marítima moderna se sustenta en la capacidad de las embarcaciones para comunicarse y situarse mutuamente en un entorno global. El Sistema de Identificación Automática (AIS) es el estándar técnico internacional [3] que permite este intercambio de información en tiempo real, operando en la banda de frecuencia VHF (*Very High Frequency*).

El funcionamiento del AIS se basa en la transmisión autónoma y periódica de datos desde un transpondedor a bordo. Estos datos son recibidos por otros buques, estaciones costeras o satélites, permitiendo la creación de un mapa de tráfico marítimo dinámico (véase la Figura 2-1). La Organización Marítima Internacional (IMO por sus siglas en inglés), a través del convenio SOLAS (*Safety Of Life At Sea*), establece la obligatoriedad del sistema para buques de arqueo bruto superior a 300 toneladas en viajes internacionales, buques de carga de más de 500 toneladas y todos los buques de pasaje, independientemente de su tamaño [4]. Los transpondedores de Clase A, exigidos por esta normativa en la práctica totalidad de la flota comercial actualizan su posición con una frecuencia de entre 2 y 10 segundos durante la navegación. Adicionalmente, para embarcaciones de menor porte, se dispone de los transpondedores de Clase B; su uso es de carácter voluntario y transmiten la actualización de posición típicamente cada 30 segundos, con independencia de la velocidad del buque.

La información transmitida por el AIS se divide principalmente en dos categorías:

- **Datos estáticos:** Información que no varía durante la navegación, como el número MMSI, el número IMO, el nombre del buque, sus dimensiones (eslora y manga) y el tipo de embarcación.
- **Datos dinámicos:** Información que cambia constantemente, derivada de los sensores del buque, como la posición GPS (*Global Positioning System*) mostrada por latitud y longitud, el rumbo sobre el fondo (COG por sus siglas en inglés), la velocidad sobre el fondo (SOG por sus siglas en inglés) y el estado de navegación.

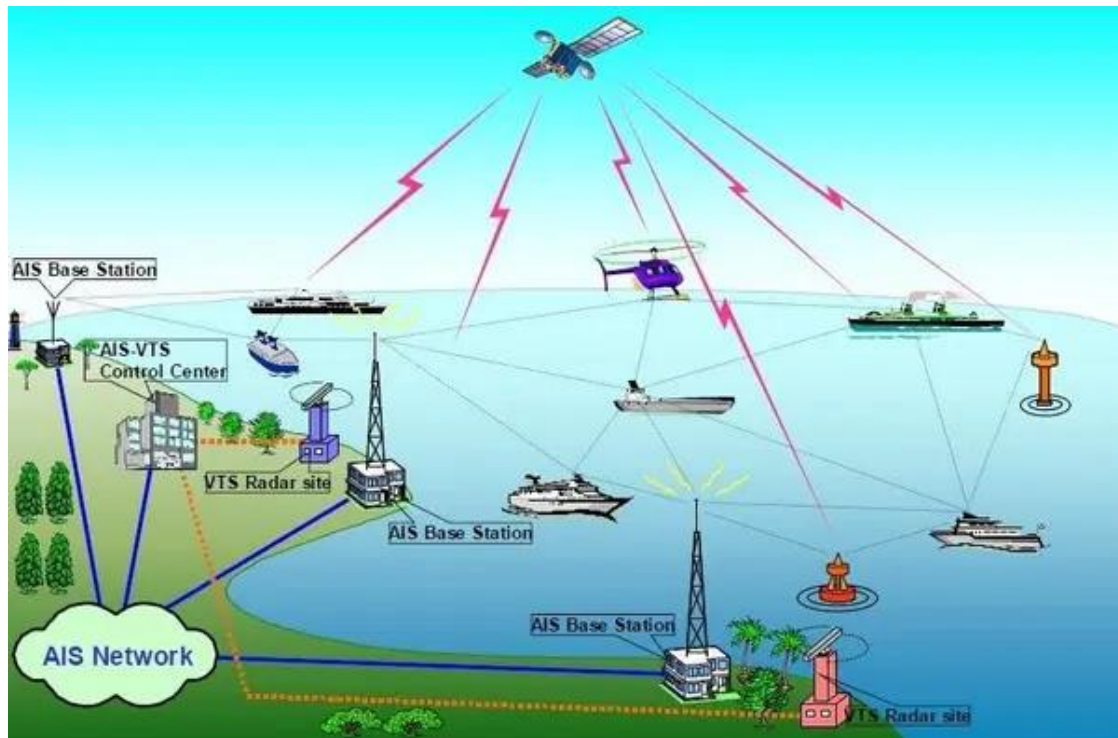


Figura 2-1 Diagrama conceptual de la arquitectura de red y transmisión del sistema AIS [40]

Para la extracción y utilización de estos datos en equipos informáticos, los receptores AIS encapsulan la información recibida utilizando el protocolo NMEA 0183 [5]. Este estándar de comunicación marina transmite la información en forma de tramas secuenciales de texto plano en caracteres ASCII (*American Standard Code for Information Interchange*), cuyos campos internos están separados por comas.

En el entorno AIS, existen dos tipos principales de tramas o sentencias que el sistema debe procesar:

- **VDM (*VHF Data-link Message*):** Contiene la información recibida de otras embarcaciones circundantes. Se identifica con el prefijo *!AIVDM*.
- **VDO (*VHF Data-link Own-vessel message*):** Contiene la información dinámica propia del buque que aloja el receptor, identificada mediante el prefijo *!AIVDO*.

Una trama estándar NMEA para AIS (*!AIVDM,1,1,A,13a1:sPP00PD>hRMDH@j0?v00000,0*04*) presenta una estructura rígida. La decodificación algorítmica de esta cadena exige procesar los siguientes campos clave:

1. **Número de fragmentos e identificador:** Indica en cuántas sentencias se ha dividido el mensaje original y el orden del fragmento actual. Los mensajes largos, como los datos estáticos de un buque, suelen dividirse en múltiples tramas.
2. **Canal de radio:** Identifica si los datos se recibieron a través del canal AIS A (161.975 MHz) o B (162.025 MHz).
3. **Carga útil (*Payload*):** Es el segmento crítico de la trama (en el ejemplo, la cadena alfanumérica *13a1:sPP...*). Esta sección está codificada mediante un formato estandarizado de ASCII de 6 bits (*Armored ASCII*), lo que permite comprimir datos binarios en caracteres imprimibles. Contiene la información real del buque (MMSI, coordenadas geográficas, velocidad, etc.).
4. **Suma de comprobación (*Checksum*):** Un código hexadecimal situado al final de la sentencia (ej. **04*) que permite al *software* receptor verificar la integridad estructural de la trama y detectar errores de transmisión.

Para la materialización de este TFG, la decodificación de la carga útil de los mensajes AIS es esencial. El *software* de la aplicación debe validar la integridad del mensaje NMEA, decodificar los caracteres de 6 bits a su valor binario correspondiente, aplicar máscaras de bits para extraer las magnitudes físicas reales y transformar esta información numérica en objetos virtuales geolocalizados en el horizonte del usuario.

Tradicionalmente, una vez procesada, esta información se representa en pantallas de navegación convencionales, como *plotters* o el Sistema de Información y Visualización de Cartas Electrónicas (ECDIS por sus siglas en inglés). El uso continuado de estos equipos requiere que el operador desvíe la mirada del exterior hacia los instrumentos del puente de mando. Esta dinámica de alternancia visual entre el horizonte y las pantallas conlleva un aumento de la carga cognitiva y puede repercutir en el tiempo de reacción ante posibles situaciones de riesgo. Para mitigar estas limitaciones operativas, resulta de interés explorar alternativas de visualización que integren los datos de navegación directamente en el campo de visión natural del usuario.

2.2 Realidad aumentada

Para comprender la integración de datos AIS en el campo visual del usuario, es necesario establecer primero el marco conceptual de la tecnología base. La Realidad Aumentada (RA) se define, según el continuo de virtualidad de Milgram y Kishino [6], como una tecnología que superpone información digital (imágenes, texto, modelos 3D) sobre la percepción directa del mundo físico en tiempo real. A diferencia de la Realidad Virtual (RV), que aísla al usuario en un entorno sintético, la RA enriquece la realidad existente.

Ronald Azuma [7], en su definición canónica de 1997, establece tres características indispensables para un sistema de RA:

1. Combina lo real y lo virtual.
2. Es interactiva en tiempo real.
3. Está registrada en tres dimensiones (los objetos virtuales se alinean geoméricamente con el mundo físico).

2.2.1 Evolución histórica y usos

Aunque los orígenes se remontan a la década de 1960 con el sistema "*Sword of Damocles*" de Ivan Sutherland (véase la Figura 2-2) (el primer visor montado en la cabeza), la tecnología ha experimentado su mayor auge en la última década gracias a la miniaturización de componentes. Inicialmente utilizada en el sector militar (HUDs en aviación de combate) y mantenimiento industrial (proyección de diagramas de cableado en Boeing), la RA ha permeado en sectores como la medicina (guiado quirúrgico), la logística y la navegación [8].



Figura 2-2 Sistema "*Sword of Damocles*" de Ivan Sutherland [41]

2.2.2 Realidad aumentada geoespacial

Dentro del espectro de la RA, este proyecto se enmarca específicamente en la RA Geoespacial (*Location-Based AR*). A diferencia de la RA tradicional, que ancla objetos virtuales a marcadores visuales (códigos QR) o superficies planas detectadas (mesas, suelos), la RA Geoespacial utiliza el sistema de coordenadas geográficas globales (latitud, longitud y altitud) como referencia. Esta variante resulta especialmente interesante en el ámbito marítimo ya que permite proyectar la posición de un buque (obtenida vía AIS) en su ubicación física exacta en el horizonte, independientemente de la distancia al observador, facilitando así la integración visual de los datos de navegación con el entorno físico.

2.3 Hardware de visualización

En términos generales, el *hardware* de RA combina subsistemas de visualización (*displays*) para mostrar la información sintética, con un conjunto de sensores diseñados para conocer la posición y orientación del usuario en el espacio, dotando al sistema de distintos grados de libertad [9]. Centrándonos en la parte visual, la experiencia en RA depende de dos componentes que trabajan de manera conjunta: la fuente de imagen (*micro-display*) y el combinador óptico siendo este el elemento físico que permite al usuario ver el mundo real y el virtual simultáneamente, mezclando ambas fuentes de luz. Cabe destacar que, en los sistemas de realidad virtual puros, que emplean pantallas completamente opacas, este combinador no existe, ya que no hay integración con la luz del entorno natural [10].

2.3.1 Arquitecturas ópticas

La industria actual de la realidad extendida (XR por sus siglas en inglés) presenta tres enfoques principales para la visualización de contenido digital:

1. **Pantallas Opacas (Realidad Virtual):** Aíslan completamente al usuario del entorno físico mediante paneles LCD (*Liquid Crystal Display*) u OLED (*Organic Light Emitting Diode*) cerrados. Aunque ofrecen una inmersión total, no son aptas para tareas de navegación o vigilancia en puente, donde la pérdida de referencia visual del entorno real no es aceptable.
2. **Video See-Through (VST o Passthrough):** Esta tecnología no emplea ópticas transparentes. Captura el mundo real mediante cámaras externas de alta resolución y lo renderiza en tiempo real en una pantalla opaca frente a los ojos del usuario, mezclándolo digitalmente con los gráficos virtuales. Dispositivos como las Meta Quest 3 utilizan este sistema. Aunque facilita la gestión de oclusiones (objetos virtuales que ocultan reales), introduce una latencia

"*Motion-to-Photon*" que puede inducir cinetosis (*motion sickness*). Además, presenta un modo de fallo crítico: ante una pérdida de energía, el usuario queda visualmente aislado ("ciego"), lo cual puede llegar a limitar su uso primario en operaciones de seguridad marítima.

3. **Optical See-Through (OST o Transparentes):** Esta arquitectura utiliza elementos ópticos semitransparentes que permiten al usuario ver el mundo real directamente a través de ellos, mientras se proyecta la luz artificial sobre su retina o sobre el combinador. Dentro de las OST, destacan dos diseños:

- **Óptica "Birdbath" (Baño de pájaro):** Su diseño catadióptrico pliega la trayectoria de la luz procedente de un micro-display (generalmente Micro-OLED situado en la parte superior) hacia un divisor de haz (*beam splitter*) y un espejo cóncavo semitransparente (véase la Figura 2-3). La principal ventaja del diseño *Birdbath* es su calidad de imagen superior y coste contenido en comparación con las guías de onda (*waveguides*). Permite un campo de visión (FOV por sus siglas en inglés) amplio (típicamente 45°-50°) con excelente uniformidad de color. Sin embargo, su principal desventaja es la eficiencia óptica: debido a que la luz debe atravesar el divisor de haz dos veces (perdiendo teóricamente el 50% de intensidad en cada paso), se requiere un *micro-display* de muy alto brillo para compensar las pérdidas y que la imagen sea visible en exteriores. Además, el conjunto óptico es físicamente más grueso que una guía de onda, lo que limita la reducción del tamaño de las gafas [11].

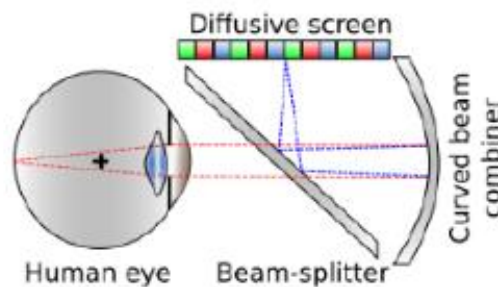


Figura 2-3 Esquema de la arquitectura óptica "Birdbath" mediante divisor de haz. [42]

- **Guías de Onda (Waveguides):** Utilizan una lámina delgada de vidrio o polímero donde la luz viaja mediante reflexión interna total (TIR por sus siglas en inglés). Permiten factores de forma muy delgados, pero su fabricación es costosa y suelen presentar menor eficiencia lumínica y uniformidad de color.

2.3.2 Tecnologías de micro-display

La fuente que genera la imagen antes de pasar por la óptica es igual de importante a la hora de definir las gafas de RA. Las tecnologías actuales compiten en brillo, contraste y eficiencia energética [12]:

- **LCD (Liquid Crystal Display):** Esta tecnología tradicional requiere una fuente de luz trasera (*backlight*) que ilumina todo el panel constantemente. Una capa de cristal líquido actúa como obturador, bloqueando o dejando pasar la luz para formar los colores. Lo cual representa un problema en RA, ya que, en un sistema óptico transparente, el color negro se utiliza para representar transparencia (el sistema no proyecta luz, dejando ver el mundo real). Los paneles LCD, al tener una retroiluminación siempre encendida, no pueden lograr un "negro absoluto"; siempre emiten una luz residual grisácea. Esto provoca que el usuario vea un recuadro brillante flotando frente a sus ojos (efecto "*glow*"), rompiendo la inmersión y reduciendo el contraste de los datos superpuestos.

- **Micro-OLED (*Organic Light Emitting Diode*):** A diferencia de las pantallas LCD tradicionales, los píxeles OLED son auto emisivos; es decir, cada píxel genera su propia luz, esto encuentra su relevancia para el proyecto: en el factor que, en RA, el color "negro" se percibe como transparencia. Los paneles Micro-OLED logran un contraste teóricamente infinito (negros puros), lo que significa que el fondo de la interfaz desaparece completamente, dejando solo los datos AIS flotando en el horizonte sin un recuadro grisáceo visible ("*glow*") que es común en tecnologías retroiluminadas.
- **LCoS (*Liquid Crystal on Silicon*):** Históricamente muy común en RA (usado en HoloLens 1 o Magic Leap 1). Funciona reflejando luz de una fuente externa (LED o láser) a través de una capa de cristal líquido sobre un chip de silicio. Esta tecnología encuentra una limitación, requiere una fuente de iluminación constante ("*light engine*") que añade volumen y consumo. Además, es difícil lograr negros perfectos, lo que puede resultar en un efecto de "ventana semitransparente" que reduce la inmersión.
- **Micro-LED (*Light Emitting Diode*):** Considerada el futuro de la visualización. Ofrece la autoemisividad del OLED pero con brillo inorgánico mucho mayor (millones de nits), ideal para uso bajo luz solar directa. Sin embargo, su fabricación masiva es actualmente prohibitiva y la tecnología aún enfrenta desafíos para producir paneles a todo color eficientes a precios de consumo.

2.4 Sensores

2.4.1 Sensores de posicionamiento y rastreo

Para lograr la coherencia espacial —que un dato AIS permanezca anclado a la posición geográfica del barco y no a la cabeza del usuario— es imprescindible determinar la pose (posición y orientación) de las gafas en tiempo real. Esto se logra mediante la fusión sensorial de diversos componentes [13]:

- **Sensores Relativos (IMU):** La Unidad de Medición Inercial (IMU por sus siglas en inglés) es el componente básico. Integra acelerómetros (miden aceleración lineal) y giroscopios (miden velocidad angular). Proporcionan una respuesta inmediata y de baja latencia a los movimientos rápidos de la cabeza, fundamental para evitar mareos, pero sufren de *drift* (deriva acumulada) con el tiempo si no se corrigen con referencias externas.
- **Sistemas de Rastreo Seguimiento y Mapeo Espacial:** Para complementar a la IMU, mitigar su error acumulado y dotar al sistema de capacidad para entender la traslación física en el entorno (habilitando los 6 Grados de Libertad), se emplean tecnologías de seguimiento basadas en visión artificial o emisiones externas [14]:
 - **SLAM (*Simultaneous Localization and Mapping*):** La evolución más significativa en el estado del arte reciente es la incorporación de cámaras de seguimiento (*tracking cameras*) en dispositivos de bajo coste. Estas cámaras, que pueden ser RGB (*Red, Green and Blue*), monocromas o infrarrojas, permiten la implementación de algoritmos de SLAM. El SLAM visual permite al dispositivo mapear el entorno, detectando superficies, suelos y paredes mediante la identificación de puntos característicos de alto contraste, así como realizar la odometría visual, es decir, calcular el desplazamiento físico del usuario (traslación) analizando cómo se mueven estos puntos característicos entre fotogramas consecutivos. Es importante notar que, en un entorno estructurado como el puente de mando (*bridge*), el SLAM visual funciona correctamente gracias a la presencia de referencias geométricas estáticas (ventanas, consolas, mamparos) que permiten la extracción de puntos característicos fiables. Sin embargo, al proyectar la visión hacia mar abierto, el sistema se enfrenta a un entorno desestructurado. La falta de puntos de referencia fijos, sumada a las alteraciones visuales producidas por el oleaje y los reflejos (*dynamic invalid features*), provocan que los algoritmos de odometría visual pierdan

eficacia y acumulen errores de deriva rápidamente [15] [16]. Por este motivo, para el posicionamiento y la orientación en el exterior marítimo, el sistema no puede depender del SLAM visual, siendo necesario priorizar la fusión de datos de sensores inerciales (IMU) y de posicionamiento global (GPS) [17].

- **Lighthouse:** Tecnología de rastreo "*Outside-In*" utilizada en realidad virtual de alta gama (e.g., HTC Vive). Emplea estaciones base externas que emiten barridos láser captados por fotodiodos en las gafas. Aunque ofrece una precisión milimétrica, su dependencia de infraestructuras externas fijas e instaladas limita su viabilidad para sistemas portátiles de uso en puente de mando o exteriores.
- **Sensores Absolutos:** Mientras que los sistemas anteriores miden cambios relativos de movimiento o mapean el entorno geométrico inmediato, los sensores absolutos son aquellos capaces de vincular el dispositivo con marcos de referencia globales invariables. En aplicaciones de RA Geoespacial, su presencia es insustituible, ya que permiten la alineación del mundo virtual con la geografía real:
 - **Magnetómetro:** Actúa como brújula digital proporcionando la referencia del Norte magnético. Es vital para la alineación inicial del sistema de coordenadas virtual con el geográfico.
 - **GPS/GNSS (*Global Navigation Satellite System*):** A diferencia de la RA en interiores, la visualización de datos AIS requiere geolocalización absoluta (latitud, longitud y altitud). Las gafas ligeras suelen delegar esta función en la unidad de cómputo conectada (*smartphone* o *puck*) para reducir peso y calor.

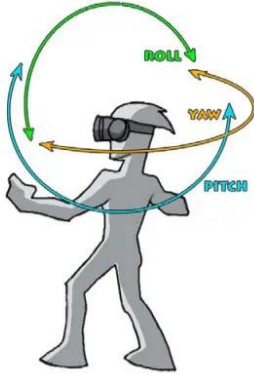
2.4.2 Grados de libertad

El concepto de "Grados de Libertad" (DoF, por sus siglas en inglés) define la capacidad del sistema para rastrear el movimiento del usuario en el espacio tridimensional. Esta clasificación es determinante para la experiencia de inmersión y la utilidad operativa de la aplicación.

- **3DoF (Tres Grados de Libertad: Rotacional):** En un entorno 3DoF, el usuario es un observador estático. Si el usuario camina hacia adelante, los objetos virtuales se mueven con él, manteniendo la distancia relativa. Para la visualización de datos AIS a larga distancia, el 3DoF suele ser suficiente, ya que las etiquetas de los barcos están proyectadas en el horizonte. El sistema rastrea únicamente los movimientos de rotación sobre los tres ejes cartesianos (ver Figura 2-4):
 - *Pitch* (Cabeceo), mirar arriba/abajo.
 - *Yaw* (Guiñada), mirar izquierda/derecha.
 - *Roll* (Alabeo), inclinar la cabeza hacia los hombros.
- **6DoF (Seis Grados de Libertad: Posicional):** El 6DoF permite que los elementos virtuales (como una carta náutica holográfica o un panel de control flotante) permanezcan anclados en una posición física fija dentro del puente de mando, independientemente de si el usuario camina alrededor de ellos o se acerca para leer un detalle. Esto mejora drásticamente la ergonomía y reduce la fatiga visual y el mareo (*cybersickness*), ya que el movimiento visual coincide perfectamente con la percepción vestibular del oído interno. En este entorno 6DoF, además del movimiento rotacional, se rastrean los 3 movimientos traslacionales (ver Figura 2-4):
 - *Surge*, moverse hacia adelante/atrás.
 - *Sway*, moverse hacia izquierda/derecha.
 - *Heave*, moverse hacia arriba/abajo (agacharse/levantarse).
- **9DoF:** Término comúnmente utilizado en marketing para referirse a una IMU que fusiona datos de 3 ejes de acelerómetro, 3 de giroscopio y 3 de magnetómetro. Técnicamente, en

términos de movimiento espacial, el resultado sigue siendo una experiencia 6DoF, pero con corrección de deriva absoluta gracias al magnetómetro.

3 degrees of freedom (3-DoF)



6 degrees of freedom (6-DoF)

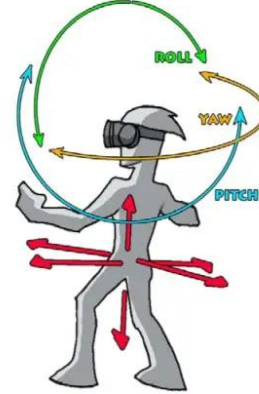


Figura 2-4 Diagrama 3DoF vs 6DoF [43]

2.5 Ecosistema de *software* y herramientas de desarrollo

El desarrollo de una aplicación de Realidad Aumentada (RA) eficaz requiere la orquestación de diversos niveles de *software*, desde el sistema operativo hasta los *plugins* específicos.

2.5.1 Frameworks y plugins de Realidad Aumentada

Para la construcción de la aplicación, existen diversas herramientas en el mercado. A continuación, se evalúan las más relevantes y su pertinencia para el proyecto:

- **ARCore (Google):** Es el kit de desarrollo de *software* (SDK por sus siglas en inglés) estándar de Google para RA en Android. Se basa en la detección de planos (suelos, mesas) y el seguimiento de movimiento mediante la cámara del móvil. Resulta especialmente relevante por su capacidad para generar "anclas geoespaciales" (*Geospatial API*) y ofrecer una ubicación de alta precisión gracias a su Sistema de Posicionamiento Visual (VPS por sus siglas en inglés). Sin embargo, presenta la limitación de estar diseñado principalmente para RA en pantallas de teléfonos móviles (*handheld AR*) y no estar optimizado de forma nativa para gafas ópticas estereoscópicas.
- **ARKit (Apple):** El homólogo de *ARCore* para el ecosistema iOS. Ofrece capacidades avanzadas de seguimiento, pero es incompatible con el *hardware* basado en Android seleccionado para este trabajo.
- **Vuforia Engine:** Uno de los SDK más veteranos. Destaca en el reconocimiento de imágenes (marcadores) y objetos 3D. Es muy utilizado en industria para mantenimiento (por ejemplo, reconocer una pieza de motor), pero su enfoque en "marcadores visuales cercanos" lo hace poco útil para la visualización de datos AIS en el horizonte (mar abierto), donde no hay referencias visuales estáticas que reconocer.
- **Unity MARS (Mixed and Augmented Reality Studio):** Es una extensión de entorno de autoría para Unity que permite simular entornos reales dentro del editor. Facilita el diseño de aplicaciones "inteligentes" que se adaptan al espacio físico. Sin embargo, su alto coste de licencia y complejidad lo relegan a producciones comerciales de gran escala.
- **XREAL SDK:** Es el SDK propietario de XREAL. A diferencia de *ARCore*, el XREAL SDK está optimizado para acceder directamente a la IMU de las gafas y a las funciones de Nebula.

Permite controlar la visualización estereoscópica y acceder a los eventos de los sensores con latencia mínima, requisito indispensable para sincronizar la vista con los datos AIS dinámicos.

Cabe mencionar de forma independiente a *AR Foundation*. Dado que no es un *framework* o SDK en sentido estricto, sino una capa de abstracción unificada propia de Unity, actúa como un puente que permite escribir el código una sola vez y desplegarlo en múltiples plataformas subyacentes (como *ARCore* o *ARKit*). Su uso es fundamental para la arquitectura moderna de las aplicaciones de RA.

2.6 Análisis de mercado y selección de *hardware*

El mercado actual de gafas de RA se encuentra en una fase de desarrollo continuo, dividiéndose principalmente entre dispositivos *standalone* (autónomos, provistos de procesador y batería integrados) y sistemas *tethered* (dependientes de una unidad de computación externa, como un teléfono inteligente o un módulo de procesamiento dedicado). En el caso de estos últimos, su funcionamiento exige la conexión a equipos anfitriones compatibles; habitualmente, dispositivos con sistema operativo Android equipados con un puerto USB 3.1 o superior que dispongan de soporte para salida de vídeo mediante *DisplayPort*. Para el desarrollo de este proyecto, se han evaluado diversas alternativas representativas del estado del arte, considerando su viabilidad para la visualización de datos AIS. A continuación, se describen los dispositivos analizados:

1. **XREAL Air 2** (Figura 2-5): Dispositivo tipo *tethered*. Destaca por su equilibrio entre calidad de imagen y portabilidad. Emplea paneles Sony Micro-OLED que ofrecen una densidad de píxeles líder en la industria y un gran contraste, factor determinante para la transparencia en RA [18]. Al carecer de batería pesada y procesador, su ergonomía es superior para turnos de guardia prolongados (72g). Su ecosistema de desarrollo a través del XREAL SDK es el más maduro para la integración con Unity en Android [19].



Figura 2-5 Gafas XREAL Air 2 [44]

2. **XREAL Air 2 Ultra**: Representa la evolución profesional del modelo anterior. Aunque mantiene el factor de forma ligero de unas gafas (titanio, 80g), incorpora dos cámaras de visión artificial en el frontal. Esto le dota de capacidades 6DoF reales, permitiendo *hand tracking* (control gestual sin mandos) y mapeo del entorno (SLAM) [20]. No obstante, considerando que el caso de uso de este proyecto es la visualización de datos AIS proyectados en el horizonte a larga distancia, estas características adicionales de seguimiento espacial no resultan imprescindibles. Por consiguiente, el incremento de precio respecto al modelo estándar no queda justificado para esta aplicación concreta.

3. **TCL RayNeo X2:** Es un dispositivo *standalone* robusto, equipado con el procesador Qualcomm Snapdragon XR2 [21]. A diferencia de las XREAL (que usan óptica *Birdbath*), las RayNeo X2 utilizan guías de onda ópticas difractivas (*waveguides*) con pantallas Micro-LED binoculares a todo color [22]. Esto les permite alcanzar niveles de brillo muy superiores, ideales para exteriores soleados. Sin embargo, su arquitectura *standalone* implica un mayor peso, un diseño más voluminoso y una autonomía limitada por la batería integrada en la montura.
4. **RayNeo Air 3S Pro:** Competidor directo de las XREAL Air 2 en el segmento *tethered*. Su principal ventaja competitiva es la luminosidad, alcanzando picos de hasta 1200 nits en el ojo, lo que garantiza una legibilidad excelente incluso bajo luz solar directa en el mar [23]. Aunque sus especificaciones visuales son sobresalientes, su SDK y comunidad de desarrollo en occidente aún están consolidándose frente a la adopción masiva del *hardware* de XREAL.
5. **Meta Quest 3:** El estándar actual en Realidad Mixta VST. Es un dispositivo *standalone* de gran potencia [24]. A pesar de sus capacidades inmersivas, su naturaleza de "casco cerrado" (opaco) lo descarta para operaciones de navegación, ya que, en caso de fallo del sistema de cámaras, el usuario dejaría de ver el entorno, teniendo que prescindir por completo de la herramienta.
6. **Microsoft HoloLens 2:** Representa el estándar tradicional en el sector profesional e industrial de la Realidad Mixta *Optical See-Through* (OST). Se trata de un dispositivo *standalone* equipado con un *hardware* avanzado para el mapeo espacial y la interacción holográfica. No obstante, su elevado coste de adquisición, superior a los 3000 euros, justifica la necesidad de explorar alternativas de bajo coste como las planteadas en este proyecto. Adicionalmente, la reciente discontinuación de esta línea de visores por parte del fabricante limita su viabilidad para implementaciones a largo plazo.

Característica	XREAL Air 2	XREAL Air 2 Ultra	RayNeo Air 3S Pro	TCL RayNeo X2	Meta Quest 3
Tipo de Óptica	<i>Birdbath</i>	<i>Birdbath</i>	<i>Birdbath</i>	Guía de onda (<i>Waveguide</i>)	<i>Video See-Through</i> (VST)
Tipo de Dispositivo	<i>Tethered</i> (Dependiente)	<i>Tethered</i> (Dependiente)	<i>Tethered</i> (Dependiente)	<i>Standalone</i> (Autónomo)	<i>Standalone</i> (Autónomo)
Tecnología de Pantalla	Micro-OLED	Micro-OLED	Micro-OLED	Micro-LED	LCD (<i>Pancake</i>)
*Luminosidad (Ojo) **	~500 nits	~500 nits	~1200 nits	~1500 nits	N/A (Opaca)**
Sensores	IMU, Magnetómetro, Prox.	IMU, SLAM, Mag.	IMU, Sensor Prox.	IMU, Cámaras, GPS	SLAM, Profundidad, IMU
Grados de Libertad	3DoF	6DoF + Hand Tracking	3DoF	6DoF (SLAM Visual)	6DoF Nativo
SDK Principal	XREAL SDK (Unity)	XREAL SDK (Unity)	RayNeo SDK	RayNeo SDK	Meta XR SDK
Precio Aprox.	~400€ (+ Beam Pro)	799€	350€	850€	550€

Tabla 1 Análisis de mercado del *hardware* (elaboración propia)

Atendiendo a los datos técnicos y económicos recopilados en la Tabla 1, la selección de las gafas XREAL Air 2 se fundamenta en las siguientes consideraciones:

1. **Exploración de la tecnología OST:** En la visualización de realidad aumentada actual existen dos opciones predominantes y válidas: los sistemas *Video See-Through* (VST o *passthrough*), como Meta Quest 3, y los visores *Optical See-Through* (OST), como las XREAL. Ambas tecnologías presentan ventajas e inconvenientes. El presente proyecto tiene como objetivo explorar y validar las posibilidades prácticas que ofrece un visor OST para la asistencia a la navegación marítima.

2. **Disponibilidad de sensores:** Una limitación crítica de los dispositivos VST autónomos líderes del mercado, como las Meta Quest 3, es que carecen de sensores nativos indispensables para la orientación geográfica global, como el módulo GPS y la brújula (magnetómetro). Esto descarta su viabilidad directa para renderizar elementos referenciados geográficamente en alta mar sin depender de integraciones complejas de *hardware* de terceros.
3. **Coste y madurez del ecosistema:** Dentro del abanico de visores puramente OST evaluados, las XREAL Air 2 destacan por ser la alternativa más económica, alineándose directamente con la premisa de "bajo coste" del título del proyecto. Asimismo, su entorno de desarrollo (XREAL SDK) se sitúa como el más maduro, documentado y estable en occidente para su integración con el motor Unity, mitigando los riesgos técnicos durante la fase de desarrollo.

2.7 Unidad de computación

Como se ha descrito anteriormente, el empleo de gafas de RA de tipo *tethered* precisa de una unidad de computación externa encargada del procesamiento de datos espaciales y la generación de la señal de vídeo. Para abordar esta necesidad técnica, se distinguen principalmente dos aproximaciones tecnológicas:

2.7.1 Dispositivos basados en Android

La aproximación más extendida para dar soporte a las gafas *tethered* consiste en la utilización de teléfonos inteligentes de gama alta que operan bajo el sistema operativo Android. Para garantizar el funcionamiento, estos terminales deben cumplir con requisitos *hardware* específicos; en particular, es necesaria la presencia de un puerto USB-C 3.1 (o de especificación superior) que soporte salida de vídeo mediante el protocolo *DisplayPort Alternate Mode*.

Entre los dispositivos del mercado que cumplen con estas características destacan terminales de altas prestaciones como la serie Samsung Galaxy S (incluyendo modelos recientes como el Samsung Galaxy S25), así como otros equipos equivalentes orientados a la gama alta. A nivel técnico, estos *smartphones* integran procesadores de gran capacidad computacional y memorias RAM (*Random Access Memory*) de alta velocidad, lo que les permite gestionar la renderización de los modelos tridimensionales con una latencia reducida.

Las principales ventajas de emplear esta configuración radican en el aprovechamiento de un dispositivo multipropósito, evitando la adquisición de *hardware* adicional. Asimismo, estos terminales ofrecen conectividad de red avanzada integrada (como 5G o Wi-Fi), lo cual resulta de gran utilidad para la recepción de flujos de datos externos en aplicaciones conectadas. No obstante, cabe señalar que la ejecución continua de aplicaciones de RA supone una elevada carga de procesamiento que repercute directamente en un consumo acelerado de la batería del dispositivo.

2.7.2 XREAL Beam Pro

El XREAL Beam Pro se introduce en el mercado como un dispositivo de computación espacial dedicado, diseñado para liberar al teléfono móvil personal del usuario de la carga de procesamiento AR.

Características Generales: El dispositivo opera bajo el sistema NebulaOS (basado en Android 14) y presenta un factor de forma similar al de un smartphone, con una pantalla de 6.5 pulgadas que actúa como interfaz táctil y *trackpad* espacial. Su arquitectura de *hardware* destaca por la inclusión de dos puertos USB-C independientes: uno dedicado a la conexión de las gafas (con salida de vídeo *DisplayPort Alternate Mode*) y otro para la alimentación eléctrica o transferencia de datos, permitiendo el uso continuado durante guardias largas.

Variantes de Conectividad y Posicionamiento (Wi-Fi vs. 5G): Actualmente, este dispositivo se comercializa en dos variantes técnicas cuya diferencia radica en la conectividad y en los sensores de posicionamiento:

- **Modelo 5G:** Esta variante integra un módem celular y un módulo GNSS (GPS, GLONASS, Galileo) nativo. Desde el punto de vista de la navegación autónoma, representa la solución *hardware* ideal, ya que el propio dispositivo es capaz de obtener su geolocalización (latitud y longitud) sin depender de fuentes externas, simplificando la arquitectura del sistema.
- **Modelo Wi-Fi:** A diferencia del modelo 5G, carece de módulo GPS interno. Esta limitación de *hardware* implica que el dispositivo no puede determinar su posición geográfica de forma autónoma empleando constelaciones satelitales.

Implicaciones Técnicas para el Proyecto: La utilización del modelo Wi-Fi implica que el dispositivo no cuenta con un módulo GNSS nativo para determinar su ubicación de forma autónoma mediante satélites. No obstante, esta particularidad técnica se solventa de manera sencilla y eficaz en la práctica operativa. Para que la aplicación pueda adquirir sus coordenadas y situar los blancos AIS correctamente en el horizonte, el Beam Pro requiere únicamente enlazarse a una red externa. Esto se logra de forma directa compartiendo la conexión y los servicios de ubicación desde un teléfono móvil convencional (mediante punto de acceso o *tethering*), o bien conectando el dispositivo directamente a la red Wi-Fi del buque. De este modo, el sistema obtiene la geolocalización necesaria sin añadir complejidad arquitectónica ni requerir la compra de *hardware* de posicionamiento adicional.

Procesamiento Espacial: Independientemente de la versión, ambas variantes comparten la capacidad de procesar la fusión de sensores de la IMU de las gafas. Esto permite ejecutar algoritmos de estabilización de imagen (3DoF/6DoF) y funciones de *Smooth Follow* (seguimiento suavizado), esenciales para mitigar el efecto del balanceo del buque en la visualización de datos.

2.8 Motores de Desarrollo (IDE)

Para la creación de la lógica de la aplicación y la renderización de los barcos sobre el horizonte, se han valorado las siguientes alternativas:

- **Unity:** Es el estándar de facto en la industria de la AR. Su elección se fundamenta en la compatibilidad directa con el XREAL SDK, el cual está diseñado específicamente para integrarse con Nebula. Unity gestiona la renderización 3D, la física y la conversión de coordenadas geográficas (GPS) a coordenadas cartesianas (*World Space*) de forma nativa. La amplia documentación existente y su arquitectura basada en componentes (C#) facilitan la implementación de la lógica de decodificación de mensajes AIS.
- **Unreal Engine:** Motor gráfico de alta fidelidad que utiliza C++ o *Blueprints*. Aunque ofrece una potencia gráfica superior, su curva de aprendizaje es más pronunciada. Además, la integración del SDK de XREAL en Unreal es históricamente menos prioritaria para el fabricante que la versión de Unity, lo que podría suponer riesgos técnicos y falta de soporte para funciones específicas del Beam Pro.
- **Android Studio:** Entorno oficial para desarrollo Android (Java/Kotlin). Si bien es excelente para apps 2D convencionales, carece de un motor de renderizado 3D espacial nativo. Desarrollar una visualización de realidad aumentada desde cero en Android Studio requeriría implementar librerías gráficas de bajo nivel (como OpenGL o Vulkan) y reescribir manualmente los algoritmos de fusión de sensores que Nebula y Unity ya resuelven, lo cual es ineficiente para el alcance de este TFG.
- **Godot:** Motor de código abierto (*Open Source*) ligero. Aunque es una alternativa prometedora y gratuita, el soporte oficial de XREAL para Godot es limitado o inexistente en comparación con Unity. La falta de un SDK oficial optimizado implicaría dificultades para acceder correctamente a los datos de la IMU de las gafas o utilizar las funciones de la pantalla 3D.

La arquitectura de *software* seleccionada se fundamenta en la integración del motor gráfico Unity y el entorno Nebula (mediante el SDK de XREAL). La elección de Unity como plataforma principal de desarrollo se justifica por diversos motivos técnicos y operativos. En primer lugar, permite aprovechar y dar continuidad a los trabajos previos. Adicionalmente, Unity constituye un estándar consolidado en la industria para la creación de aplicaciones de XR, ofreciendo una alta compatibilidad nativa con sistemas Android y disponiendo de soporte oficial directo a través de las bibliotecas de desarrollo proporcionadas por el fabricante de las gafas.

Bajo este esquema de trabajo, Unity opera como el motor computacional encargado de procesar la lógica de los datos AIS y de renderizar la simbología marítima correspondiente. De forma complementaria, el sistema Nebula, en ejecución sobre el módulo Beam Pro, gestiona la interacción a bajo nivel con el *hardware* de las gafas y la computación espacial para la estabilización de la imagen, facilitando así un entorno visual estable para el operador.

3 DESARROLLO DEL TFG

En este capítulo se describe el proceso de desarrollo e implementación de la aplicación. Una vez definidos el entorno de trabajo, la configuración base del motor gráfico y el *hardware* seleccionado, la metodología del proyecto se ha estructurado en tres fases principales de desarrollo. En primer lugar, se abordó la integración de los sensores de posicionamiento y orientación (como el receptor GPS y el magnetómetro) para determinar de forma continua la ubicación y perspectiva del usuario. En una segunda etapa, se implementó la lógica matemática y espacial para la colocación de marcadores geoespaciales en la escena virtual. Por último, se desarrolló el módulo de adquisición y decodificación de las tramas de datos AIS, procediendo a su correspondiente posicionamiento geoespacial para la representación visual de los contactos marítimos. Esta secuencia de trabajo ha permitido articular de manera progresiva las distintas funcionalidades del sistema.

3.1 Arquitectura *hardware*

La arquitectura de *hardware* se ha dividido en dos subsistemas principales como se puede ver en la Figura 3-1: el bloque de visualización y computación espacial, y el bloque de adquisición de datos AIS. Se ha prescindido del uso de teléfonos móviles convencionales, optando por una configuración de *hardware* dedicado que garantice una mayor precisión inmersiva y capacidad de procesamiento [25].

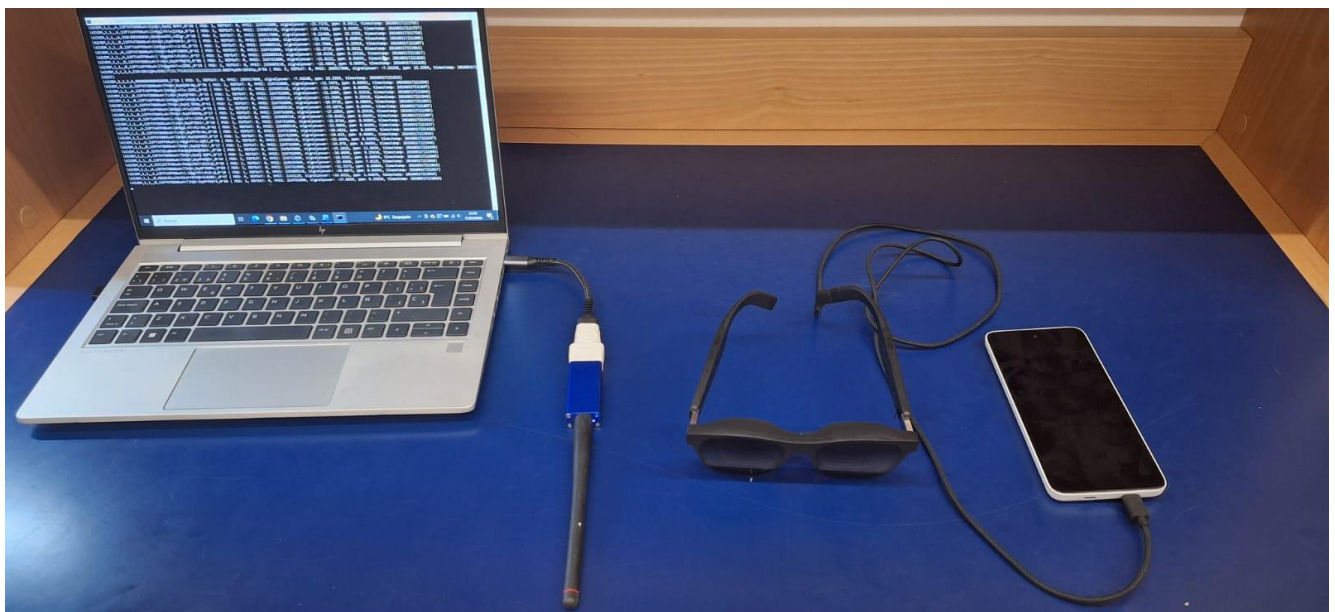


Figura 3-1 Arquitectura del *hardware* (Elaboración propia)

3.1.1 Gafas XREAL Air 2 y dispositivo XREAL Beam Pro

El desarrollo del *software* y la ejecución de las pruebas empíricas se han llevado a cabo utilizando las gafas de realidad aumentada XREAL Air 2 emparejadas con el terminal XREAL Beam Pro (Figura 3-2). Este visor, equipado con pantallas de tecnología Micro-OLED, permite la proyección nítida de los elementos virtuales sobre el entorno real del usuario, manteniendo un factor de forma ligero que resulta crucial para su uso prolongado en exteriores.



Figura 3-2 Gafas XREAL Air 2 con el dispositivo XREAL Beam Pro (Elaboración propia)

Por su parte, el dispositivo XREAL Beam Pro está diseñado específicamente para operar como unidad de procesamiento principal (*Compute Unit*) bajo el paradigma de la computación espacial. Este módulo opera bajo el entorno XREAL Nebula, el cual actúa como un *middleware* o capa de abstracción espacial ejecutada sobre una arquitectura Android modificada.

El XREAL SDK asume la proyección de la interfaz en el espacio tridimensional. Para lograrlo, este sistema interpreta de forma continua los datos sensoriales recibidos de las gafas y adapta la imagen renderizada en tiempo real. Este proceso de transformación compensa el movimiento del usuario, permitiendo que los objetos virtuales mantengan una posición estática aparente con respecto al entorno físico. En el contexto de este proyecto, Nebula proporciona el soporte operativo necesario para el despliegue de la aplicación desarrollada en Unity en modo de Realidad Aumentada, actuando como el entorno y lanzador (*launcher*) principal, mientras que el SDK es el encargado de asumir la gestión de la estereoscopia y asimilar la carga computacional del renderizado.

Para lograr la correcta ubicación de los buques virtuales en el horizonte del usuario, se ha implementado un módulo de fusión de sensores que integra las coordenadas globales proporcionadas por el receptor GPS y los datos de orientación espacial suministrados por la Unidad de Medición Inercial (IMU), con especial relevancia del magnetómetro (brújula digital). Esta fusión de datos permite

establecer un sistema de coordenadas locales preciso y estabilizar la orientación del campo de visión del usuario frente a los movimientos de su cabeza [26].

3.1.2 Sistema de recepción AIS: Antena VHF y Receptor SDR

Para la captura de tráfico marítimo real, la arquitectura del sistema de adquisición se ha estructurado en dos bloques principales. El primero consiste en un subsistema de recepción compuesto por una antena sintonizada en las frecuencias de la banda VHF marina (típicamente 161.975 MHz y 162.025 MHz). Esta antena se encuentra conectada a un receptor SDR (*Software Defined Radio* o Radio Definida por *Software*) como se puede ver en la Figura 3-3.

El uso de esta tecnología permite desplazar el procesamiento de las señales de radiofrecuencia del dominio del *hardware* al dominio del *software*. De este modo, el receptor se encarga únicamente de captar y digitalizar la señal analógica, generando un flujo de datos I/Q (fase y cuadratura por sus siglas en inglés) en bruto.

El segundo bloque corresponde al equipo informático, que actúa como nodo decodificador y eslabón intermedio con el sistema de Realidad Aumentada. El receptor SDR se conecta mediante una interfaz USB a este ordenador estándar, el cual asume las tareas de procesamiento digital de señales (DSP, por sus siglas en inglés).



Figura 3-3 Antena SDR (Elaboración propia)

En este equipo informático se ejecuta *AIS-catcher*, una herramienta de *software* especializada en la recepción de tráfico marítimo [27]. Su función principal consiste en procesar el flujo de datos I/Q proveniente del SDR, aplicar los filtros correspondientes y demodular la señal GMSK (*Gaussian Minimum Shift Keying*) [3] característica de las transmisiones AIS.

Una vez demodulada la señal, el programa extrae la carga útil y transforma la información binaria en tramas de texto estructuradas bajo el estándar NMEA 0183 [5], específicamente en formato AIVDM. Esta traducción convierte las variaciones electromagnéticas en paquetes de datos inteligibles (incluyendo la posición, velocidad, rumbo e identificador MMSI de cada buque), dejándolos listos para ser transmitidos y procesados por el entorno gráfico en el dispositivo XREAL Beam Pro.

3.2 Entorno de desarrollo

En este apartado se detalla la configuración del *software* y las herramientas de desarrollo empleadas para la construcción de la aplicación. La adecuada parametrización del motor gráfico resulta necesaria para establecer una comunicación estable entre el *hardware* de las gafas XREAL, los sensores del dispositivo Beam Pro y el entorno virtual tridimensional.

3.2.1 Configuración de Unity y XREAL SDK

El motor gráfico seleccionado para el desarrollo y despliegue de la aplicación ha sido Unity. Para garantizar la estabilidad del proyecto y la correcta compatibilidad con las bibliotecas externas requeridas, la instalación y la programación se han llevado a cabo utilizando específicamente la versión 6000.3.1f1. A diferencia del proyecto predecesor —el cual basaba su funcionamiento en *ARCore* de Google para teléfonos móviles—, esta iteración emplea la arquitectura estándar de *AR Foundation* combinada de forma nativa con el proveedor específico XREAL SDK.

Para garantizar la compatibilidad entre el *hardware* de computación espacial y el *software*, se han aplicado las siguientes configuraciones necesarias en los parámetros de compilación (*Player Settings*) y en la jerarquía de la escena:

- **Configuración del *Rig* de Cámaras:** Para la estructuración del sistema de visualización se ha adoptado el componente estándar *XR Origin* (o *AR Rig*) proporcionado por Unity, evitando el uso de arquitecturas propietarias cerradas. En el entorno de este motor gráfico, el objeto cámara (*Main Camera*) actúa como el punto de vista virtual del usuario, siendo el elemento matemático responsable de capturar la escena tridimensional y proyectarla en la pantalla.

En esta arquitectura, la integración con el visor se realiza a través del XREAL SDK, el cual se vincula directamente con dicha cámara principal. Este complemento (*plugin*) intercepta el proceso de renderizado estándar y asume la gestión de la separación de la imagen de forma automática, generando y enviando el *buffer* estereoscópico correspondiente a cada una de las lentes de las gafas XREAL Air 2. Mantener esta jerarquía estandarizada resulta necesario para prevenir anomalías de renderizado en el visor, tales como la ausencia de proyección de la señal de vídeo (pantalla negra)."

- **Configuración del *Active Input Handling*:** En lo que respecta a la gestión de las entradas del sistema, durante las fases iniciales se forzó la utilización del sistema clásico en la modalidad "*Old*". Sin embargo, esta configuración generaba un conflicto crítico de *hardware*: el sistema clásico permitía el acceso ininterrumpido a la API (*Application Programming Interface*) nativa de la brújula del dispositivo anfitrión (*Input.compass*), pero bloqueaba la lectura de la telemetría inercial avanzada (*XR Pose*) generada por las gafas XREAL Air 2. Para habilitar la lectura simultánea de ambos flujos de datos —el magnetómetro del *smartphone* y el giroscopio del visor— fue imperativo reconfigurar este parámetro a la modalidad "*Both*" (véase la Figura 3-4). Esta decisión arquitectónica permitió a Unity procesar concurrentemente los *inputs* clásicos del sistema operativo Android y los sistemas de seguimiento espacial modernos del XREAL SDK.

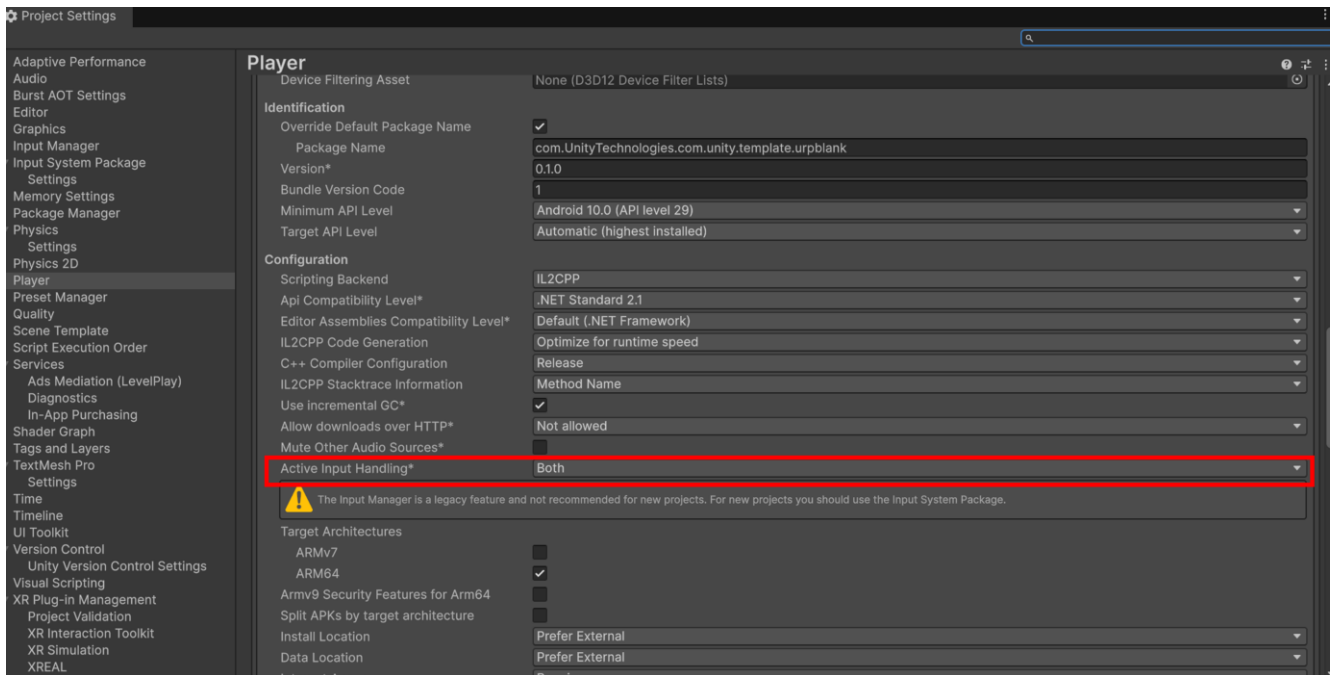


Figura 3-4 Configuración Active Input Handling (Elaboración propia)

- **API Gráfica:** Una API gráfica es el conjunto de bibliotecas de *software* que actúa como puente de comunicación de bajo nivel entre el motor de renderizado de Unity y la unidad de procesamiento gráfico (GPU por sus siglas en inglés) del *hardware*. Para el desarrollo de este proyecto, se ha seleccionado OpenGL ES3 (véase la Figura 3-5) como la interfaz de renderizado principal. En consecuencia, se ha retirado Vulkan de la lista de compilación nativa. Esta decisión técnica tiene como finalidad favorecer la estabilidad visual y prevenir posibles conflictos de compatibilidad gráfica al ejecutar el entorno inmersivo sobre el sistema operativo Nebula (basado en Android) del dispositivo XREAL Beam Pro.

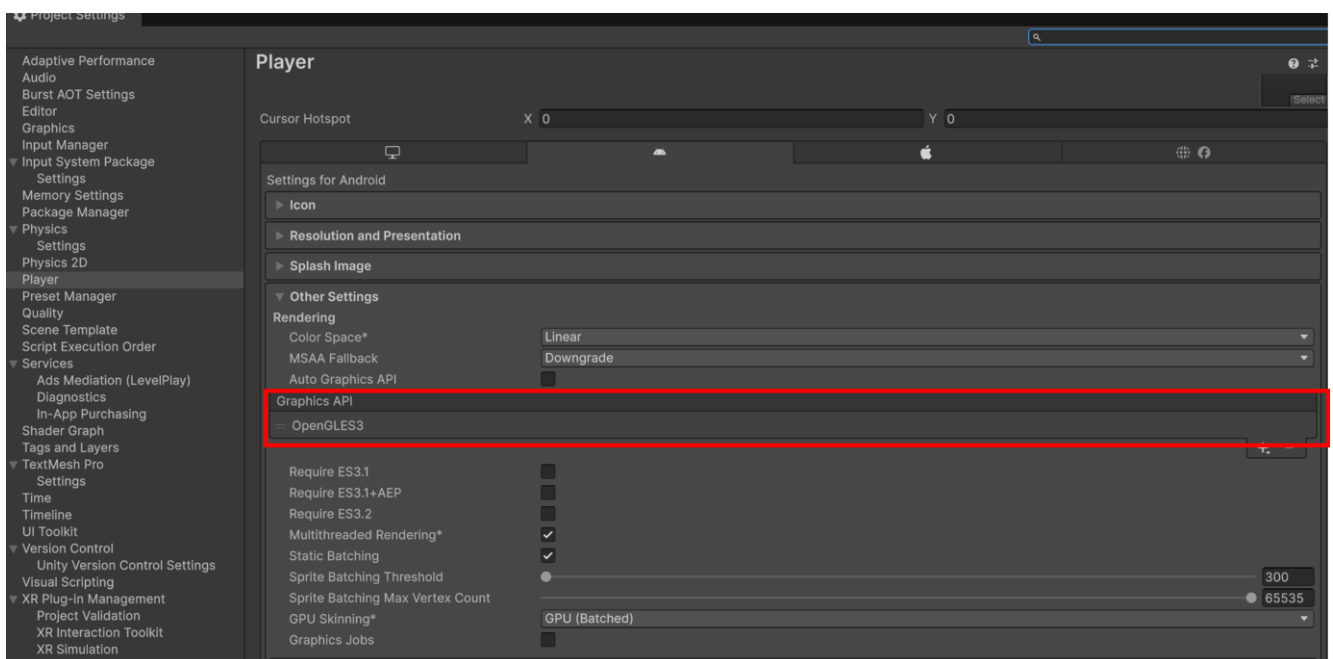


Figura 3-5 Configuración API gráfica (Elaboración propia)

Adicionalmente a la configuración de renderizado, el correcto despliegue de la aplicación ha requerido el ajuste de otros parámetros funcionales en el editor. Específicamente, se ha habilitado el *plugin* oficial de XREAL dentro del gestor de proveedores de Realidad Extendida (*XR Plug-in Management*), un paso necesario para que Unity reconozca e interactúe con los sensores de las gafas. Asimismo, se ha configurado el nivel de API de Android (*Minimum y Target API Level*) dentro de los ajustes de compilación, adecuando las versiones a los requisitos técnicos exigidos por el fabricante para habilitar el acceso a los recursos de red y localización del dispositivo.

3.2.2 Estructura de la escena y gestión de Scripts

En el desarrollo con Unity, el entorno de trabajo se organiza mediante escenas (*Scenes*). Una escena actúa como un contenedor espacial y lógico donde se disponen todos los elementos que conforman la aplicación, denominados genéricamente *GameObjects*. Para gestionar este entorno, la interfaz del motor gráfico se divide en varios paneles de control (Figura 3-6). Los paneles principales son la jerarquía (*Hierarchy*), que muestra una lista estructurada de todos los objetos presentes en la escena; la vista de escena (*Scene*), que permite la edición espacial y visual; y el inspector (*Inspector*), que detalla las propiedades y coordenadas del objeto seleccionado.

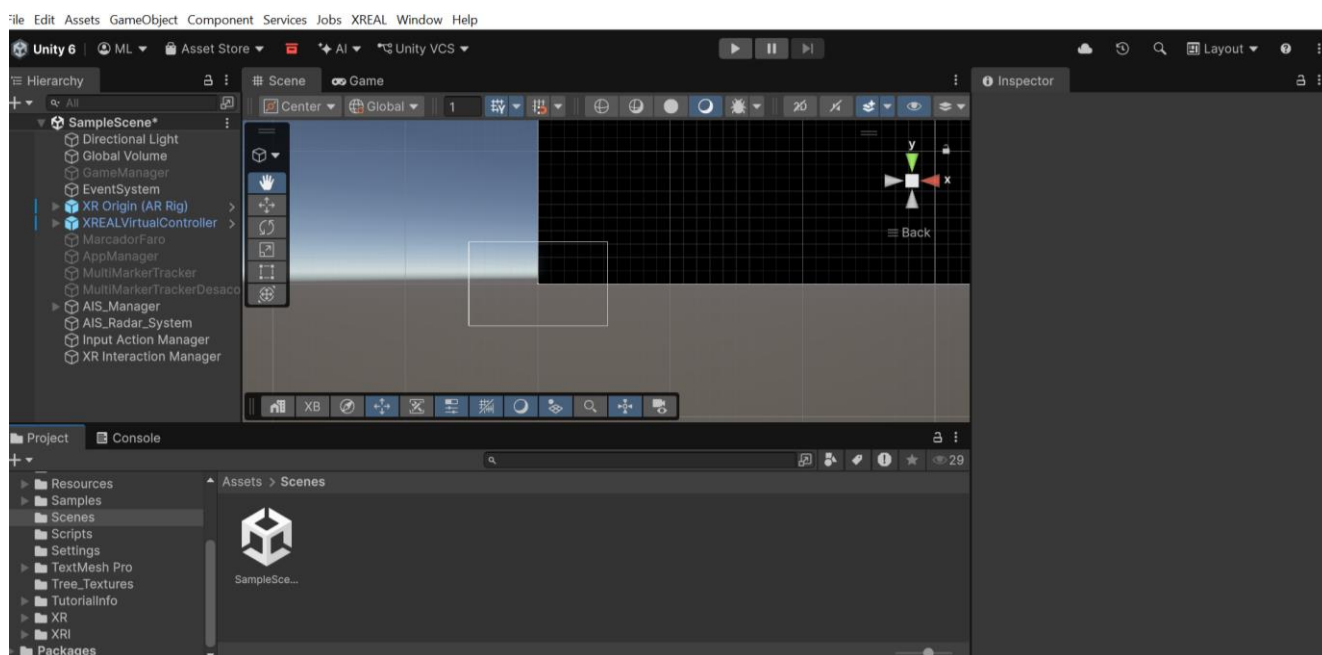


Figura 3-6 Interfaz de la aplicación Unity (Elaboración propia)

La arquitectura base configurada para este proyecto incluye los *GameObjects* necesarios para el entorno de Realidad Aumentada. El elemento central es el *XR Origin* (o *AR Rig*), un contenedor que representa la ubicación del observador e integra la cámara principal (*Main Camera*). A este se suman elementos auxiliares como fuentes de iluminación (*Directional Light*) y lienzos de interfaz (*Canvas*) configurados en el espacio tridimensional (*World Space*).

Por su parte, la programación de la aplicación se implementa mediante la redacción de *scripts* en lenguaje C#. En la arquitectura basada en componentes de Unity, estos *scripts* no se ejecutan de forma aislada, sino que se adjuntan a los *GameObjects* a través del panel inspector. Al vincular un código a un objeto de la escena, el *script* adquiere la capacidad de leer variables externas y modificar las propiedades del objeto en tiempo de ejecución. En el marco de este trabajo, la gestión de *scripts* permite procesar la telemetría de los sensores, ejecutar los cálculos trigonométricos para la conversión de coordenadas

geográficas a espaciales, y actualizar dinámicamente la posición y rotación de los modelos tridimensionales en el campo de visión del usuario [28].

3.2.3 Consideraciones y limitaciones en la captura del entorno inmersivo

Un aspecto técnico crítico que considerar durante la fase de desarrollo y la posterior evaluación de resultados es la naturaleza óptica del *hardware* de visualización empleado. Las gafas XREAL Air 2 operan bajo el paradigma de RA *Optical See-Through* (OST). En este tipo de sistemas, el usuario percibe el entorno físico directamente a través de lentes translúcidas, mientras que los elementos virtuales son proyectados mediante las pantallas Micro-OLED hacia el ojo del observador [10].

Debido a esta arquitectura de *hardware*, la composición final del entorno mixto (la suma visual del mundo real y los datos AIS superpuestos) ocurre exclusivamente en la retina del usuario. El dispositivo XREAL Beam Pro, encargado del renderizado, no procesa ni almacena una señal de vídeo del entorno físico que pueda combinarse por *software* con la capa digital.

Como consecuencia directa, el sistema operativo carece de la capacidad nativa para realizar capturas de pantalla (*screenshots*) o grabaciones de vídeo que reflejen fielmente la experiencia inmersiva completa. Cualquier intento de captura a nivel de *software* únicamente registrará el *buffer* de imagen estéreo generado por Unity (los modelos tridimensionales sobre un fondo negro o transparente). Esta limitación técnica inherente a los dispositivos OST obliga a que, para la documentación visual de los resultados empíricos y la demostración de la correcta alineación espacial de los objetivos, sea necesario recurrir a la fotografía externa, situando el objetivo de una cámara directamente tras la lente del visor inmersivo.

3.3 Fase 1: Acceso y calibración de sensores de navegación

El desarrollo del sistema geoespacial se planteó bajo una metodología incremental. La etapa inicial consistió en la validación del acceso a la telemetría interna del dispositivo XREAL Beam Pro. Antes de abordar la representación espacial de los marcadores planos, resultaba necesario comprobar que el motor de Unity pudiera leer, de forma estable y con los permisos adecuados, la orientación y la posición global del usuario.

3.3.1 Activación y lectura del magnetómetro (Brújula)

El primer objetivo de *software* fue desarrollar un módulo capaz de interpretar la orientación del usuario respecto al polo norte. Esta variable resulta determinante, ya que el sistema de coordenadas locales de Unity debe rotar dinámicamente para alinear el eje Z virtual con el norte verdadero.

Para la implementación de esta lectura, se configuró una escena de pruebas preliminar en Unity. En ella, se instanció un objeto contenedor básico (*GameObject*) al cual se le vinculó, a modo de componente, el *script* de desarrollo *CompassTest.cs* (Anexo IV: *CompassTest.cs*). Asimismo, para comprobar visualmente la recepción de los datos en tiempo real, se incorporó un elemento de texto a la interfaz de la escena mediante el componente *Text Mesh Pro*. Esta estructura permite que el código interroge a los sensores del dispositivo en cada ciclo de ejecución y proyecte los valores obtenidos directamente en pantalla. Durante la evaluación de este módulo, se documentó una particularidad técnica en la interfaz de programación (API) de Unity: la activación estándar del magnetómetro (*Input.compass.enabled = true*) únicamente proporciona el rumbo magnético (*magneticHeading*). Sin embargo, para evitar desviaciones en los cálculos trigonométricos posteriores, el sistema requiere el rumbo verdadero (*trueHeading*). Para que el sistema operativo Android pueda calcular este norte

verdadero, es necesario arrancar simultáneamente los servicios de localización, de modo que el dispositivo pueda compensar la declinación magnética local (Figura 3-7).

```
IEnumerator Start()
{
    if (txt != null)
        txt.text = "Iniciando...";

    // 1) Activar brújula
    Input.compass.enabled = true;

    // 2) Intentar arrancar localización para forzar permisos y ayudar a trueHeading
    if (SystemInfo.supportsLocationService)
    {
        Input.location.Start();
    }
}
```

Figura 3-7 Inicialización de la brújula y servicios de ubicación (Elaboración propia)

Una vez estabilizado el sensor tras un breve periodo de inicialización, el flujo de ejecución permite extraer en tiempo real el valor de orientación absoluta en grados (0° a 360°) junto con su margen de precisión (*headingAccuracy*). Para trasladar esta información del nivel de *software* a la escena visual, los valores obtenidos se enlazan directamente con los *GameObjects* del entorno. Por un lado, la cifra de orientación y su margen de error se proyectan de forma continua en el panel de interfaz de usuario (*Canvas*) mediante componentes de texto, permitiendo al observador conocer su rumbo exacto. Por otro lado, este dato angular se aplica a la componente de rotación (*Transform*) del objeto *XR Origin*, sentando así la base geométrica para alinear correctamente el entorno virtual tridimensional con el norte verdadero [29].

3.3.2 Validación del sistema de posicionamiento global (GPS)

Confirmada la viabilidad de la lectura de orientación espacial, el siguiente paso consistió en la extracción de las coordenadas geográficas del observador. Con este fin, se desarrolló una iteración más avanzada del controlador, implementada en el *script SensorIntegrationHUD.cs* (Anexo V: Código SensorIntegrationHUD.cs).

Dado que la aplicación se ejecuta sobre una arquitectura basada en Android (XREAL Beam Pro), el primer requerimiento abordado en este módulo fue la gestión de permisos en tiempo de ejecución. El código implementa una llamada a la API del sistema operativo para verificar y solicitar explícitamente el permiso de ubicación de alta precisión (*Permission.FineLocation*).

Una vez concedido el acceso, se procede a la activación del servicio de localización de Unity mediante la instrucción *Input.location.Start(5f, 5f)*. En esta inicialización se establecieron empíricamente parámetros de actualización de 5 metros, tanto para la distancia de refresco como para la precisión requerida. Esta configuración busca un balance operativo entre la exactitud del posicionamiento continuo y la optimización del consumo energético del dispositivo portátil.

Finalmente, tras verificar que el estado del servicio de localización pasa a estar activo (*LocationServiceStatus.Running*), el flujo de ejecución extrae las variables de posicionamiento: latitud, longitud, altitud y precisión horizontal (*horizontalAccuracy*).

Para integrar esta telemetría en el entorno visual, la información obtenida se vincula directamente con los objetos de la escena. Por una parte, las coordenadas geográficas actualizadas se proyectan en el panel de interfaz (*Canvas*) mediante elementos de texto, ofreciendo al usuario una lectura en tiempo real de su ubicación exacta (Figura 3-8). Por otra parte, a nivel de arquitectura espacial, la combinación de

estos datos de posición con el vector de orientación de la brújula se aplica sobre el *XR Origin* para establecer el origen de coordenadas absoluto (0,0,0) del sistema. Fijar esta referencia física entre el mundo real y el entorno virtual resulta necesario para proceder a la instanciación geométrica y al posicionado preciso de los modelos representativos en las siguientes fases del desarrollo.



Figura 3-8 Representación de los datos sobre la pantalla del XREAL Beam Pro (Elaboración propia)

3.4 Fase 2: Posicionamiento geoespacial estático y validación empírica

Tras asegurar la correcta inicialización y lectura de la telemetría del dispositivo en la primera fase, el desarrollo avanzó hacia la validación del motor de renderizado geoespacial. El objetivo principal de esta etapa empírica fue lograr la transformación de coordenadas geográficas absolutas (latitud y longitud) a coordenadas cartesianas locales (X, Y, Z) relativas a la cámara, así como el diseño de una interfaz visual que permitiera monitorizar los cálculos matemáticos en tiempo real.

3.4.1 Prueba de concepto y calibración de precisión

Una vez establecido el origen de coordenadas del usuario, se procedió a implementar la lógica de conversión matemática de coordenadas geográficas (latitud y longitud) a coordenadas espaciales locales dentro del entorno de Unity. Este algoritmo de transformación se desarrolló e integró en el componente *script GeoMarkerTest.cs* (Anexo VI: Código GeoMarkerTest.cs).

Posteriormente, para evaluar la precisión de este algoritmo de posicionamiento de forma controlada y determinista, se diseñó una prueba de validación en la escena utilizando un punto geográfico estático de referencia conocido. Con este propósito, se seleccionó el Faro de Tenlo Chico, situado en las coordenadas de latitud 42.3986° N y longitud 8.7058° W, procediendo a instanciar un modelo representativo en dichas coordenadas.

La función principal del *script GeoMarkerTest.cs* consistía en calcular la relación espacial entre la coordenada del observador y la del objetivo. Para ello, implementa la fórmula del *Haversine* (Figura 3-9), obteniendo la distancia ortodrómica y la demora o rumbo inicial (*bearing*).

Más allá del cálculo analítico, el propósito fundamental de este componente es su integración con la escena de RA. Una vez obtenidas la distancia y la demora, el *script* traduce estos valores polares a coordenadas cartesianas (ejes X, Y, Z) locales dentro de Unity. Con esta información espacial, el sistema utiliza un objeto prefabricado (*Prefab*) previamente configurado en el editor, que contiene el diseño visual del marcador o panel de datos. Mediante las funciones de instanciación del motor gráfico, el código genera dinámicamente una copia de este *Prefab* y la posiciona en la escena tridimensional. Esta colocación se realiza tomando como punto de referencia central el *XR Origin*, garantizando que la

representación virtual quede anclada con precisión en la ubicación geográfica real del Faro de Tenlo Chico.

```
// Funciones matemáticas auxiliares (Corregidas para Doble Precisión)
private float CalculateDistance(float lat1, float lon1, float lat2, float lon2)
{
    double R = 6371e3; // Radio de la Tierra en metros

    // Conversión a Radianes usando Double
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaPhi = (lat2 - lat1) * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;

    // Fórmula del Haversine en Doble Precisión
    double a = Math.Sin(deltaPhi / 2.0) * Math.Sin(deltaPhi / 2.0) +
        Math.Cos(phi1) * Math.Cos(phi2) *
        Math.Sin(deltaLambda / 2.0) * Math.Sin(deltaLambda / 2.0);
    double c = 2.0 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a));

    return (float)(R * c);
}

private float CalculateBearing(float lat1, float lon1, float lat2, float lon2)
{
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;

    double y = Math.Sin(deltaLambda) * Math.Cos(phi2);
    double x = Math.Cos(phi1) * Math.Sin(phi2) -
        Math.Sin(phi1) * Math.Cos(phi2) * Math.Cos(deltaLambda);

    double theta = Math.Atan2(y, x);

    // Retornamos el valor en grados normalizado a 0-360
    return (float)((theta * 180.0 / Math.PI + 360.0) % 360.0);
}
```

Figura 3-9 Implementación de la fórmula del Haversine para el cálculo de distancia y demora (Elaboración propia)

Durante la primera validación de campo de este algoritmo, se detectó una desviación significativa en la proyección espacial: el sistema calculaba y renderizaba el marcador a una distancia aproximada de 400 metros, mientras que la medición de contraste realizada mediante la herramienta satelital Google Earth arrojaba una distancia real de 1.400 metros.

El análisis de esta discrepancia reveló que el fallo no provenía de la fórmula matemática, sino de un error humano en la introducción manual de las coordenadas GPS del faro (desajuste en el formato de grados decimales frente a grados y minutos). La corrección de este parámetro solventó la desviación, coincidiendo plenamente el cálculo del *software* con la topografía real. Esta incidencia evidenció la alta sensibilidad del sistema geoespacial, demostrando que el mínimo error de formato o falta de precisión en las coordenadas de origen y destino genera desviaciones severas en el posicionamiento dentro del entorno virtual.

3.4.2 Interfaz HUD y escalabilidad mediante múltiples marcadores

Una vez validado el posicionamiento espacial con un único objetivo estático, la siguiente fase consistió en adaptar el sistema para soportar un entorno dinámico con múltiples elementos simultáneos. Esta estructuración resulta necesaria como etapa previa a la integración de los datos reales del tráfico marítimo.

A nivel de arquitectura en Unity, esta transición implica pasar de un elemento fijo a un modelo de instanciación dinámica en la escena. Para lograrlo, se diseñó un objeto prefabricado base (*Prefab*) que agrupa tanto el modelo visual que representa a la embarcación como su panel de información flotante (*Canvas*). El control de este entorno recae sobre un objeto gestor (*GameObject*) central; mediante su *script* asociado, el sistema es capaz de generar en tiempo de ejecución una copia independiente de este *Prefab* por cada nueva embarcación detectada, asignándole a cada instancia sus coordenadas espaciales individuales calculadas a partir de la señal geomática.

Para gestionar la información de múltiples objetivos de forma estructurada, se definió la clase serializable *GeoWaypoint* (Figura 3-10). Esta estructura de datos encapsula las propiedades estáticas de cada contacto (identificador, latitud, longitud) junto con sus variables en tiempo de ejecución (distancia calculada, rumbo y la referencia al *GameObject* tridimensional asociado).

El flujo de representación visual de estos datos se ejecuta mediante las funciones estándar de la API de Unity y el soporte del XREAL SDK. En primer lugar, cuando el sistema recibe las coordenadas de un nuevo objetivo, utiliza la función nativa *Instantiate()* para generar un clon del *Prefab* base en la escena. A continuación, el *script* gestor convierte las coordenadas polares (distancia y demora) en coordenadas cartesianas de Unity (*Vector3*). Este vector espacial se asigna directamente a la propiedad *transform.position* del marcador instanciado, tomando como punto de referencia el *XR Origin* para asegurar su correcta ubicación en el entorno de RA. Por último, durante el ciclo continuo del motor gráfico (dentro de la función *Update()*), el sistema recalcula y actualiza la posición de todos los marcadores activos para mantener la coherencia espacial frente a los movimientos de cabeza del usuario captados por el visor.

```
public class GeoWaypoint
{
    public string waypointName;
    public double latitude;
    public double longitude;
    [HideInInspector] public GameObject markerInstance;
    [HideInInspector] public double currentDistance;
    [HideInInspector] public double currentBearing;

    // Constructor para facilitar la creación por código
    public GeoWaypoint(string name, double lat, double lon)
    {
        waypointName = name;
        latitude = lat;
        longitude = lon;
    }
}
```

Figura 3-10 Definición de la estructura de datos *GeoWaypoint* para la gestión de marcadores (Elaboración propia)

Para evaluar el comportamiento del sistema y su precisión en diferentes rangos y ángulos de visión, se definió un conjunto de puntos geoespaciales reales. Tomando como zona de operaciones el entorno marítimo cercano, se introdujeron en la lista de objetivos inicial (*List<GeoWaypoint>*) diversas marcas de navegación y elementos topográficos conocidos. En la Tabla 2 se detallan las coordenadas absolutas empleadas durante esta prueba.

Nombre del Marcador	Latitud	Longitud
Faro de Tenlo Chico	42.40803° N	8.70782° W
Puerta de Carlos I	42.39369° N	8.70298° W
Punta de Atán	42.39631° N	8.39631° W
Muelle de Torpedos (faro)	42.39897° N	8.70499° W

Tabla 2 Coordenadas geográficas de los marcadores empleados en la validación espacial. (Elaboración propia)

Para monitorizar los datos procesados en tiempo real sin depender de la consola de depuración del editor, se implementó una interfaz de usuario superpuesta tipo HUD. En cuanto a su integración en el entorno de Realidad Aumentada, fue necesario adaptar la configuración predeterminada del motor gráfico. Por defecto, los lienzos de interfaz en Unity (*Canvas*) se generan en las modalidades *Screen Space - Overlay* o *Screen Space - Camera*, lo que provoca que el panel bidimensional se renderice únicamente en la pantalla física del dispositivo móvil (XREAL Beam Pro) y resulte invisible a través de las lentes de las gafas.

Para lograr que la interfaz se integre visualmente, el componente *Canvas* se configuró en la modalidad *World Space*. Al convertir la interfaz en un objeto tridimensional dentro de la escena, fue posible emparentarla jerárquicamente a la cámara principal del *XR Origin* y posicionarla espacialmente a una distancia focal cómoda en el margen superior izquierdo del campo de visión. De este modo, la interfaz acompaña los movimientos de la cabeza del usuario sin obstruir la visualización del entorno físico ni de los marcadores proyectados.

La actualización de los datos mostrados en este panel recae sobre el *script MultiMarkerTracker.cs* (Anexo VII: Código MultiMarkerTracker.cs). Este componente asume la responsabilidad de iterar en cada fotograma sobre la lista de objetivos (*List<GeoWaypoint>*), calculando matemáticamente su posición y estructurando la información del HUD (Figura 3-11) para mostrar de forma continua:

1. **Telemetría del usuario:** Coordenadas GPS actuales y rumbo verdadero (*true heading*).
2. **Telemetría de objetivos:** Un listado iterativo que muestra el nombre identificativo de cada marcador junto a su respectiva distancia y demora (*bearing*) actualizados en tiempo real.

```

    debugData += "<b>{wp.waypointName}</b>: {wp.currentDistance:F0}m | {wp.currentBearing:F1}°\n";
}

txt.text =
    "<size=110%><b><color=#00BFFF>DATOS DE NAVEGACIÓN</color></b></size>\n" +
    "-----\n" +
    "<b>Rumbo:</b> {trueHeading:000.0}°\n" +
    "<b>Lat:</b> {currentLat:F5}°\n" +
    "<b>Lon:</b> {currentLon:F5}°\n" +
    "-----\n" +
    "<color=#00FF00>{debugData}</color>";

```

Figura 3-11 Estructuración dinámica del panel de datos de navegación (HUD) (Elaboración propia)

Los resultados de esta prueba geométrica confirmaron que el motor de renderizado era capaz de mantener la coherencia espacial para múltiples objetivos simultáneos, actualizando correctamente el HUD inmersivo y validando que la arquitectura gráfica estaba lista para recibir flujos de datos asíncronos externos.

3.5 Fase 3: Integración de telemetría marítima dinámica (AIS)

Tras validar la escalabilidad del motor geoespacial estático, el proyecto abordó su objetivo final: la sustitución de los marcadores topográficos predefinidos por embarcaciones reales en movimiento. Esta

fase implicó la conexión del entorno de RA con el SDR descrito en la arquitectura de *hardware*, estableciendo un flujo continuo de telemetría marítima.

3.5.1 Arquitectura de red y recepción asíncrona de datos (UDP)

Dado que la decodificación de las señales de radio (paso de RF a tramas NMEA 0183) se ejecuta en un equipo informático externo actuando como nodo servidor mediante el *software AIS-catcher*, se requirió implementar una pasarela de comunicación en red para inyectar estos datos en las gafas XREAL Air 2 como se puede ver en la Figura 3-12.

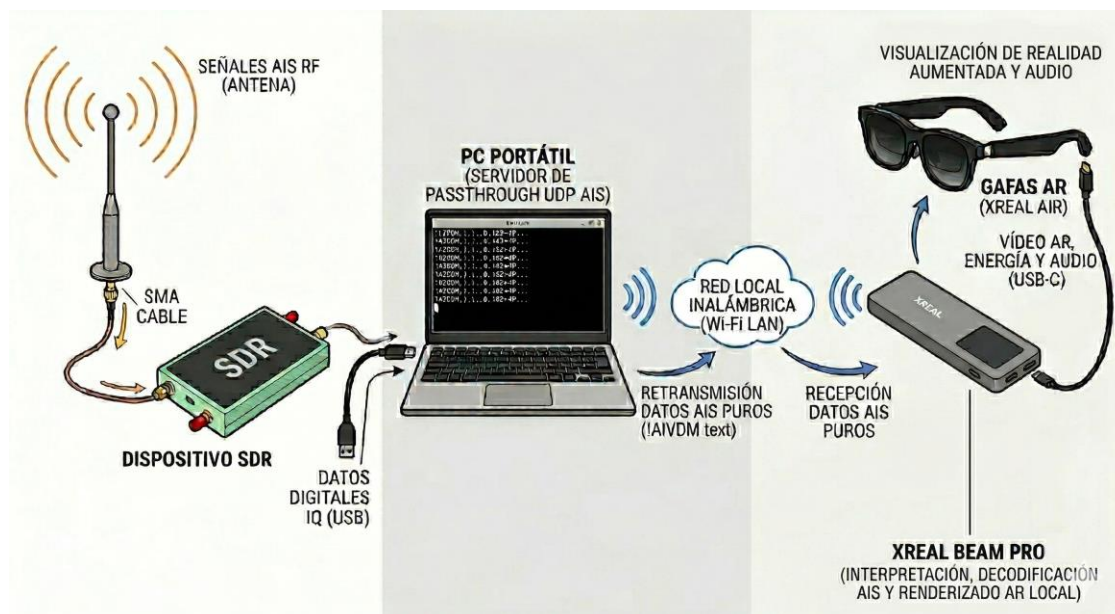


Figura 3-12 Esquema gráfico de la disposición de red (Generado por inteligencia artificial)

Se estableció una red de área local (WLAN por sus siglas en inglés) entre el equipo decodificador y el dispositivo XREAL Beam Pro. Para la inicialización del servicio de transmisión, es necesario ejecutar el *software AIS-catcher* a través de la interfaz de línea de comandos (CMD) del sistema operativo del ordenador. El operador debe invocar el archivo ejecutable especificando la ruta local del programa y concatenando los parámetros de red necesarios para habilitar el reenvío de datos (*data forwarding*).

La sintaxis del comando empleado sigue la estructura:

```
"[Ruta_del_archivo]\AIS-catcher.exe" -u [IP_Destino] [Puerto]
```

Para este proyecto, se empleó el modificador `-u` para forzar la salida de datos mediante el protocolo UDP (*User Datagram Protocol*), apuntando a la dirección IP privada que el *router* local asignó al dispositivo Beam Pro, y designando el puerto de escucha 10112. Un ejemplo de ejecución real durante las pruebas de campo fue:

```
"C:\Users\maest\Desktop\AIS catcher\AIS-catcher.exe" -u 192.168.1.33 10112
```

La selección del protocolo UDP frente a TCP (*Transmission Control Protocol*) responde a los requerimientos técnicos del sistema: en aplicaciones de telemetría en tiempo real, la baja latencia que proporciona UDP resulta preferible a la entrega garantizada de paquetes propia de TCP. Si un datagrama con la posición de un buque se pierde en la red, el *software* simplemente procesará la siguiente

actualización de coordenadas escasos segundos después, previniendo así la acumulación de datos obsoletos en el *buffer* de recepción. Adicionalmente, al operar como un protocolo sin conexión, UDP facilita enormemente la recuperación del flujo de datos frente a posibles microcortes de red, evitando las complejas lógicas de reconexión y establecimiento de sesión (*handshake*) que exigiría el protocolo TCP [30].

En el entorno de Unity, la recepción de estos paquetes se implementó a través del componente de *software* *UDP_AIS_Receiver.cs* (Anexo VIII: Código *UDP_AIS_Receiver.cs*). El principal desafío técnico en este punto radicaba en la naturaleza síncrona del motor gráfico: si el programa detiene su ejecución para "escuchar" la red de forma indefinida, la tasa de fotogramas (*framerate*) cae drásticamente, provocando la congelación de la imagen en el visor OST y causando mareos en el usuario.

Para solventar esta limitación, la arquitectura de recepción se diseñó basándose en la programación concurrente (*Multithreading*) como se puede ver en la Figura 3-13.

```
void Start()
{
    try
    {
        // Forzamos explícitamente a usar IPv4 (AddressFamily.InterNetwork)
        udpClient = new UdpClient(listenPort, AddressFamily.InterNetwork);
        isListening = true;

        listenThread = new Thread(ListenForData);
        listenThread.IsBackground = true;
        listenThread.Start();

        Debug.Log($"[AIS] Escuchando red local en el puerto {listenPort}...");
        if (debugText != null) debugText.text = $"Escuchando puerto {listenPort}...";
    }
    catch (Exception e)
    {
        Debug.LogError($"[AIS] Error de puerto: {e.Message}");
        if (debugText != null) debugText.text = $"Error: {e.Message}";
    }
}
```

Figura 3-13 Configuración del hilo secundario (Multithreading) para escucha UDP (Elaboración propia)

El *script* despliega un hilo secundario en segundo plano (*listenThread*) dedicado exclusivamente a la monitorización del puerto 10112. Cuando un datagrama *AIVDM* es interceptado, se lee como una cadena de caracteres (*string*). Sin embargo, la API gráfica de Unity prohíbe estrictamente la instanciación de objetos o la modificación de la interfaz desde hilos secundarios.

Para garantizar la seguridad en la concurrencia (*thread-safety*), se implementó un mecanismo de exclusión mutua mediante un bloque *lock*. Este bloque asegura que la variable compartida (*lastReceivedSentence*) se actualice sin colisiones de memoria. Posteriormente, en el ciclo principal de la aplicación (*Update()*) (Figura 3-14), el hilo gráfico de Unity comprueba de forma segura si existen nuevas tramas pendientes, procediendo a su extracción para la futura decodificación sin comprometer la fluidez del entorno inmersivo.

```

void Update()
{
    string sentenceToProcess = "";

    lock (lockObject)
    {
        if (!string.IsNullOrEmpty(lastReceivedSentence))
        {
            sentenceToProcess = lastReceivedSentence;
            lastReceivedSentence = "";
        }
    }
}

```

Figura 3-14 Control de concurrencia y extracción segura de tramas en el ciclo *Update* (Elaboración propia)

3.5.2 Decodificación del protocolo NMEA/AIVDM y procesamiento a nivel de bit

Una vez garantizado el flujo continuo de datos asíncronos en el hilo principal de Unity, el siguiente paso consistió en la traducción de las tramas de radiofrecuencia a variables cinemáticas y estáticas inteligibles. Las tramas emitidas por el *software AIS-catcher* se adhieren al estándar NMEA 0183, específicamente bajo la encapsulación *AIVDM*.

Este formato de texto plano estructura la información en campos separados por comas, donde el bloque crítico es la carga útil (*payload*). Para maximizar el ancho de banda del canal de radiofrecuencia VHF, esta carga útil no se transmite en texto legible, sino comprimida mediante un esquema de codificación *6-bit* ASCII.

Para procesar estas cadenas, se desarrolló la clase estática *AISParser.cs* (Anexo IX: Código AISParser.cs). El primer estadio de decodificación de este módulo (*PayloadToBinary*) consiste en revertir la compresión (Figura 3-15): itera sobre cada carácter de la carga útil, le aplica el desplazamiento aritmético estándar del protocolo (restando 48 y ajustando según su valor en la tabla ASCII) y lo convierte en una cadena continua de ceros y unos.

```

private static string PayloadToBinary(string payload)
{
    string binary = "";
    foreach (char c in payload)
    {
        int val = c - 48;
        if (val > 40) val -= 8;
        binary += Convert.ToString(val, 2).PadLeft(6, '0');
    }
    return binary;
}

```

Figura 3-15 Algoritmo de conversión de carga útil AIS a formato binario de 6 bits (Elaboración propia)

Una vez obtenida la trama binaria completa, la arquitectura del decodificador evalúa los primeros 6 bits para determinar el tipo de mensaje. El sistema se diseñó para soportar los reportes de posición tanto de buques comerciales (Clase A: Mensajes Tipo 1, 2 y 3) como de embarcaciones menores o de recreo (Clase B: Mensaje Tipo 18).

Dependiendo del tipo de mensaje, el algoritmo aplica un mapa de *bits* estandarizado (*bit mapping*) para extraer las variables. Por ejemplo, en los mensajes de Clase A, el número de identificación marítima (MMSI) se extrae de los bits 8 al 38, mientras que la latitud y la longitud se extraen leyendo bloques de 27 y 28 bits respectivamente (Figura 3-16). Tras la conversión del valor binario entero, las coordenadas se dividen por un factor de escala de 600.000 para obtener los grados decimales finales. De manera análoga, la velocidad (SOG) y el rumbo (COG) se extraen y se dividen por 10 para recuperar su valor real en nudos y grados.

```
int messageType = Convert.ToInt32(binaryString.Substring(0, 6), 2);

AISData data = new AISData();
data.MMSI = Convert.ToInt32(binaryString.Substring(8, 30), 2);

// Tipos 1, 2, 3 (Clase A) y 18 (Clase B) - Posición
if (messageType == 1 || messageType == 2 || messageType == 3 || messageType == 18)
{
    data.isPositionReport = true;
    int lonStart = (messageType == 18) ? 57 : 61;
    int latStart = (messageType == 18) ? 85 : 89;

    data.Longitude = DecodeCoordinate(binaryString.Substring(lonStart, 28));
    data.Latitude = DecodeCoordinate(binaryString.Substring(latStart, 27));

    if (messageType != 18)
    {
        data.SOG = Convert.ToInt32(binaryString.Substring(50, 10), 2) / 10.0;
        data.COG = Convert.ToInt32(binaryString.Substring(116, 12), 2) / 10.0;
    }

    if (data.Latitude >= 91.0 || data.Longitude >= 181.0) return null;
    return data;
}
```

Figura 3-16 Lógica de segmentación de bits para la extracción de variables MMSI y posición (Elaboración propia)

El análisis de los datos posicionales funcionó de manera fluida, ya que su tamaño permite su transmisión en una única línea de texto. Sin embargo, la extracción de los datos estáticos (como el nombre de la embarcación, contenido en el Mensaje Tipo 5 para Clase A y Tipo 24 para Clase B) presentó un desafío arquitectónico severo.

Dada la longitud de estas cadenas de texto, la carga útil supera el límite máximo estandarizado de 82 caracteres por línea del formato NMEA. Como consecuencia, el estándar segmenta estos mensajes y los transmite fragmentados en múltiples líneas secuenciales, un fenómeno conocido como *interleaving* (entrelazado). En las pruebas preliminares, el algoritmo descartaba estas tramas estáticas considerándolas corruptas, provocando que los marcadores en RA carecieran de nombre real.

Para solventar esta limitación técnica, se implementó una máquina de estados mediante un sistema de búfer temporal (*Dictionary<string, string> partialMessages*). La lógica de análisis segmenta la cabecera NMEA para extraer el número total de fragmentos, el fragmento actual y un identificador secuencial de mensaje (*Message ID*). Si detecta que una trama consta de dos fragmentos y recibe el primero, almacena la carga útil en la memoria utilizando el ID como clave única y detiene el procesamiento. Al interceptar el segundo fragmento con el mismo ID, el *parser* recupera la primera parte, concatena ambas cargas útiles y elimina la entrada de la memoria para evitar desbordamientos.

Finalmente, con la carga estática íntegra reconstruida (o al recibir la Parte A del Mensaje 24 que contiene el nombre), el sistema procede a su descompresión. Para ello, se implementó el método *Decode6BitText* (Figura 3-17), el cual itera sobre el binario en bloques de 6 bits y cruza el valor entero resultante con la tabla oficial de caracteres alfanuméricos del sistema AIS

(@ABCDEFGHIJKLMNOPQRSTUVWXYZ...). Esta traducción final desvela exitosamente el nombre completo del buque.

```
private static string Decode6BitText(string binString)
{
    const string charMap = "@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_ !\"#$%&'()*+,-./0123456789:;<=>?";

    string result = "";

    for (int i = 0; i + 6 <= binString.Length; i += 6)
    {
        string chunk = binString.Substring(i, 6);
        int val = Convert.ToInt32(chunk, 2);

        // Si el valor está dentro del rango, mapeamos el carácter
        if (val >= 0 && val < charMap.Length)
        {
            result += charMap[val];
        }
    }

    // Limpiamos los caracteres '@' de relleno y los espacios sobrantes
    return result.Replace("@", "").Trim();
}
```

Figura 3-17 Método de decodificación de texto alfanumérico para nombres de buques (Elaboración propia)

3.5.3 Fusión de sensores y desacoplamiento espacial en arquitectura tethered

Durante la fase de integración del sistema de visualización, se identificó una problemática inherente a las arquitecturas de RA conectadas por cable (*tethered*). Al estar la unidad de procesamiento (dispositivo anfitrión XREAL Beam Pro) y el visor óptico separados físicamente, los sistemas de referencia inercial presentaban un desacoplamiento que afectaba severamente a la proyección geométrica de los datos AIS. En la práctica, los marcadores virtuales se comportaban inicialmente como un HUD estático, desplazándose de forma solidaria con el campo de visión del usuario.

A nivel técnico, esta desviación ocurría porque el cálculo de posición tomaba como referencia directa el vector frontal de la cámara (*arCamera.forward*) y el rumbo absoluto del magnetómetro del dispositivo anfitrión, omitiendo la guiñada (*yaw*) independiente que se produce al girar la cabeza con el visor OST.

Para resolver este conflicto y anclar correctamente los elementos al entorno tridimensional global (*World Space*), se implementó una doble corrección a nivel algorítmico y matemático dentro del controlador *AIS_AR_Tracker.cs* (Anexo X: Código *AIS_AR_Tracker.cs*):

- **Fusión de los datos de orientación (*Sensor Fusion*):** Dado que el visor carece de brújula integrada, fue necesario fusionar la telemetría del magnetómetro del dispositivo anfitrión con el IMU de las gafas. Para lograrlo, el algoritmo cuenta con una rutina de calibración (*SyncRoutine*) que requiere que el usuario mire al frente durante tres segundos para capturar el desfase inicial del giroscopio (*yawOffset*) (Figura 3-18). A partir de este valor, se procesa en cada fotograma el "Rumbo de la Mirada" (*gaze heading*), combinando el rumbo magnético suavizado del dispositivo (*smoothedHeading*) con la rotación relativa actual de la cámara principal. Para garantizar su validez trigonométrica, el resultado se normaliza en una circunferencia de 360 grados. Esta fusión espacial garantiza que, incluso manteniendo el dispositivo de procesamiento en una posición estática (por ejemplo, apoyado en la consola del puente de mando), el sistema

sea capaz de registrar con precisión hacia qué azimut real está dirigiendo su mirada el operador a través de las lentes.

```
// --- CÁLCULO DE TELEMETRÍA (GAFAS + BEAM PRO) ---
float headYaw = arCamera.eulerAngles.y;
float gazeHeading = (smoothedHeading + headYaw) % 360f;
if (gazeHeading < 0) gazeHeading += 360f;

double currentLat = Input.location.lastData.latitude;
double currentLon = Input.location.lastData.longitude;
```

Figura 3-18 Algoritmo de fusión de sensores para el cálculo del Rumbo de la Mirada (*Gaze Heading*) (Elaboración propia)

- **Desacoplamiento de las coordenadas virtuales y *Billboarding*:** Para la instanciación espacial de los objetivos AIS, se modificó el vector de proyección geométrica. Se sustituyó la dependencia del vector direccional de la cámara por el vector normalizado absoluto del mundo virtual (*Vector3.forward*). Posteriormente, se calculó la demora relativa entre cada buque y el norte magnético, proyectando la posición tridimensional del objetivo mediante operaciones con cuaterniones sobre el eje Z global de la escena. Finalmente, para garantizar la legibilidad de la información táctica de la interfaz (MMSI, distancia y velocidad) independientemente del ángulo de observación, se aplicó una corrección de rotación a cada panel de usuario. Mediante la técnica de *billboarding* [31], se forzó matemáticamente a que las normales de los objetos bidimensionales intersecasen en todo momento de forma perpendicular con la posición actualizada de la cámara (Figura 3-19).

```
// --- LÓGICA DE DESACOPLAMIENTO ESPACIAL MODIFICADA ---
// Proyección estable utilizando Vector3.forward (Anclaje al mundo virtual) en lugar de arCamera.forward
float relativeBearing = (float)ship.currentBearing - smoothedHeading;
Vector3 directionToTarget = Quaternion.Euler(0, relativeBearing, 0) * Vector3.forward;
directionToTarget.y = 0;
directionToTarget.Normalize();

// Posicionamos el Canvas
ship.markerInstance.transform.position = arCamera.position + (directionToTarget * visualDistance);

// SOLUCIÓN EFECTO ESPEJO: Que el Canvas mire directamente a la cámara
ship.markerInstance.transform.rotation = Quaternion.LookRotation(ship.markerInstance.transform.position - arCamera.position);
```

Figura 3-19 Lógica de desacoplamiento espacial y rotación tipo *Billboarding* (Elaboración propia)

3.5.4 Representación inmersiva y seguimiento espacial (*Tracker*)

El último eslabón de la arquitectura de *software* consistió en la integración de los datos procesados dentro del motor gráfico para su renderizado tridimensional. Para orquestar este proceso, se desarrolló el controlador principal *AIS_AR_Tracker.cs*, el cual actúa como nexo entre el flujo de datos decodificados (*AISData*) y el algoritmo de posicionamiento geoespacial validado en las fases previas.

Dado que las emisiones de radio AIS son asíncronas y repetitivas, el sistema requería una estructura de datos capaz de discernir entre la aparición de un nuevo buque en el área y la actualización cinemática de un buque ya conocido. Para resolver esta dicotomía, se implementó un diccionario en C# (*Dictionary<int, GeoWaypoint>*), utilizando el identificador único MMSI de 9 dígitos como clave de indexación (*Key*) (Figura 3-20).

En el ciclo de ejecución principal, cuando el sistema recibe un paquete de datos validado, consulta este diccionario. Si el MMSI no existe en los registros, el motor instancia un nuevo modelo

tridimensional (*prefab*) para representar al buque en el entorno. Cabe destacar que, con el objetivo de mantener la coherencia visual e investigadora, se recuperó la malla geométrica del marcador diseñada en la fase predecesora del proyecto. Sin embargo, este recurso gráfico tuvo que ser optimizado y reconvertido técnicamente para su uso específico en RA inmersiva, ajustando sus materiales paramétricos y su escala para maximizar la visibilidad a través de las lentes OST de las gafas XREAL Air 2.

```

if (data != null)
{
    if (!trackedShips.ContainsKey(data.MMSI) && data.isPositionReport)
    {
        // Barco nuevo: Creamos el Canvas
        GeoWaypoint newShip = new GeoWaypoint();
        newShip.latitude = data.Latitude;
        newShip.longitude = data.Longitude;
        newShip.sog = data.SOG;
        newShip.cog = data.COG;

        newShip.markerInstance = Instantiate(markerPrefab);
        newShip.uiController = newShip.markerInstance.GetComponent<ShipMarkerUI>();

        trackedShips.Add(data.MMSI, newShip);
    }
    else if (trackedShips.ContainsKey(data.MMSI))
    {
        if (data.isPositionReport)
        {
            trackedShips[data.MMSI].latitude = data.Latitude;
            trackedShips[data.MMSI].longitude = data.Longitude;
            trackedShips[data.MMSI].sog = data.SOG;
            trackedShips[data.MMSI].cog = data.COG;
        }
        else if (data.isStaticReport)
        {
            trackedShips[data.MMSI].shipName = data.ShipName;
        }
    }
}

```

Figura 3-20 Gestión de la base de datos de buques activos mediante Dicionarios (C#) (Elaboración propia)

Por el contrario, si el MMSI ya existe en el diccionario, el sistema simplemente actualiza sus coordenadas geográficas, su velocidad (SOG) y su rumbo (COG), optimizando drásticamente la carga computacional al evitar la destrucción y creación constante de objetos gráficos (*GameObjects*). Posteriormente, el algoritmo itera sobre todos los buques registrados, recalculando en tiempo real la distancia y la demora hacia cada objetivo mediante la fórmula del *Haversine*, y trasladando el modelo en el espacio virtual hacia la cámara.

3.5.5 Interfaz contextual y carga dinámica de activos gráficos

Para maximizar la utilidad operativa de la aplicación en un entorno marítimo real, la representación visual no podía limitarse a posicionar la geometría del marcador espacial. Se requería dotar a cada buque virtual de un panel de información flotante (UI) que mostrase en tiempo real su nombre, distancia, velocidad, rumbo y bandera nacional.

Para gestionar esta interfaz bidimensional emparentada a cada modelo 3D, se creó el componente *ShipMarkerUI.cs* (Anexo XI: Código *ShipMarkerUI.cs*). El mayor desafío técnico en esta etapa fue la gestión de los recursos gráficos. Cargar de forma permanente en la memoria RAM del visor XREAL Beam Pro las texturas de las banderas de todos los países del mundo supondría un consumo de recursos ineficiente que penalizaría la fluidez gráfica del sistema de Realidad Aumentada.

Para representar visualmente la nacionalidad de cada embarcación en la interfaz del marcador, se implementó un método de asignación directa basado en el código MMSI. Según el estándar internacional de telecomunicaciones de la UIT, los tres primeros dígitos de cualquier código MMSI corresponden al MID (*Maritime Identification Digits*), el cual identifica unívocamente el país de registro del buque.

Aprovechando esta estructura, las imágenes de las banderas se importaron al directorio *Resources/flags* del proyecto de Unity, utilizando su respectivo código MID como nombre de archivo. De este modo, durante la actualización de la interfaz, el componente *ShipMarkerUI.cs* extrae el prefijo numérico del barco y utiliza la función nativa *Resources.Load<Sprite>()* para buscar y cargar dinámicamente la bandera correspondiente en el panel de información tridimensional (Figura 3-21).

```
if (imgFlag != null)
{
    string mmsiStr = mmsi.ToString();
    if (mmsiStr.Length >= 3)
    {
        string midCode = mmsiStr.Substring(0, 3);
        Sprite flagSprite = Resources.Load<Sprite>("flags/" + midCode);

        if (flagSprite != null)
        {
            imgFlag.sprite = flagSprite;
            imgFlag.color = Color.white;
        }
        else
        {
            imgFlag.color = new Color(1, 1, 1, 0);
        }
    }
}
```

Figura 3-21 Carga dinámica de activos gráficos (banderas) basada en el código MID (Elaboración propia)

Cuando un buque es instanciado en el entorno inmersivo, el *script* extrae la subcadena de los tres primeros caracteres de su MMSI e invoca a la API nativa de carga dinámica. Este método busca en el almacenamiento físico del dispositivo únicamente la textura de la bandera requerida y la inyecta en el panel visual del buque, manteniendo la huella de memoria al mínimo indispensable.

La integración de este último componente dio como resultado un entorno de visualización geoespacial robusto y completamente operativo. El sistema validó empíricamente su capacidad para captar señales de radiofrecuencia asíncronas, decodificar tramas binarias en hilos secundarios, resolver el posicionamiento trigonométrico de múltiples objetivos en tiempo real y superponer en el horizonte del usuario modelos tridimensionales con datos marítimos precisos, culminando con éxito el desarrollo de la aplicación.

4 RESULTADOS Y VALIDACIÓN

En este capítulo se presentan los resultados obtenidos tras la integración final del *hardware* y el *software*, así como el análisis de las pruebas de campo realizadas. El objetivo principal de esta fase ha sido comprobar si el sistema es capaz de representar los datos AIS sobre el entorno real de forma coherente, evaluando tanto la usabilidad de la interfaz bajo condiciones de luz natural como el comportamiento de los sensores de orientación del dispositivo.

4.1 Escenario de pruebas y metodología

Debido a la falta de tiempo material para coordinar un ensayo a bordo de las lanchas de instrucción de la Escuela Naval Militar (ENM) y ejecutar dichas ensayo a flote, se optó por buscar una alternativa terrestre que ofreciese una buena panorámica del tráfico marítimo. Por ello, se eligió la terraza del Cuartel Marqués de la Victoria, cuya cota elevada proporciona una línea de visión despejada hacia la ría, facilitando la recepción de las señales VHF y la confirmación visual de los barcos, tal y como se ilustra en la Figura 4-1.



Figura 4-1 Oficial alumno de la ENM durante la prueba de visualización (Elaboración propia)

Durante la jornada, las condiciones meteorológicas fueron de cielo despejado y alta luminosidad. Además, se aprovechó el horario de navegación de las lanchas de instrucción, lo que garantizó un flujo constante de objetivos dinámicos sobre los que probar la recepción de datos AIS.

4.2 Visualización y legibilidad de la interfaz

Uno de los aspectos más importantes a evaluar era la capacidad del usuario para leer los datos superpuestos sobre el mundo real. Es importante señalar que las imágenes que ilustran este documento se obtuvieron colocando la cámara de un teléfono móvil directamente tras el prisma de las gafas. Este método artesanal provoca ciertos problemas de enfoque y aberraciones cromáticas en los textos; no obstante, la percepción visual directa del usuario es notablemente más nítida.

En cuanto a la interfaz de usuario, el panel lateral decodificó y representó de forma exitosa los datos de las embarcaciones detectadas, logrando extraer el nombre, MMSI, distancia y demora de cada contacto. En condiciones de alta luminosidad, la paleta de colores (texto verde y blanco) permitió una lectura aceptable (Figura 4-2). Para atenuar la luz ambiental, se probó el uso del cristal oscurecedor incluido con las gafas.

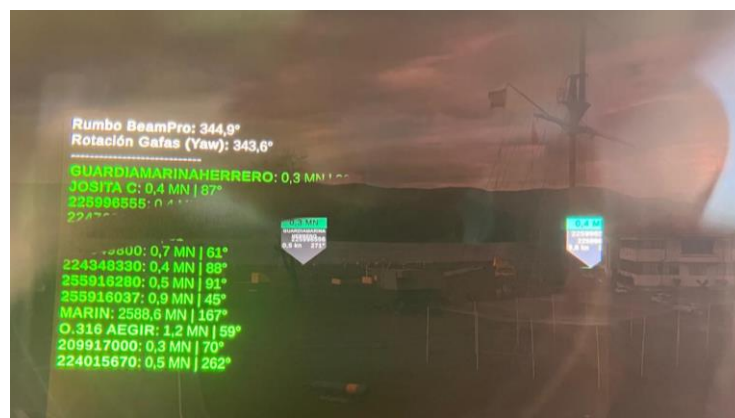


Figura 4-2 Interfaz lateral mostrando los contactos AIS detectados, capturada utilizando el filtro oscuro para mejorar el contraste (Elaboración propia)

Si bien el filtro oscuro mejoró de forma evidente el contraste de los datos, su uso resultó excesivo al punto de bloquear parcialmente la visibilidad directa de las embarcaciones físicas, por lo que su empleo debería restringirse a situaciones de deslumbramiento extremo.

Por otro lado, durante los ensayos se identificó un problema significativo de saturación en la interfaz (conocido como *clutter* visual). Al no filtrar los objetivos, el sistema detectó y listó no solo las lanchas en movimiento, sino también numerosos barcos atracados y fondeados en las proximidades. Esto provocó que la lista de contactos creciera de tal manera que acabó desplazando el indicador superior de rumbo fuera del campo de visión de las gafas. Esta sobresaturación evidencia la necesidad futura de implementar filtros por estado de navegación o por rango de distancia.

4.3 Precisión del posicionamiento espacial

El objetivo principal de la aplicación era lograr que los indicadores gráficos de cada buque se posicionaran exactamente sobre la silueta real del barco en el horizonte. Cabe destacar que al no poseer altura el mensaje AIS, todos los marcadores se proyectan a una altura constante (al nivel del ojo del observador), omitiendo la altitud y centrándose exclusivamente en el azimut.

Como se mencionó previamente, la evaluación del error de posicionamiento ha sido empírica y cualitativa. Durante la observación, se constató que el sistema ubica los marcadores en la dirección general de los buques, permitiendo su identificación. Sin embargo, se observó que los marcadores estaban ligeramente desfasados respecto a la posición exacta de las embarcaciones físicas, presentando

una desviación angular estimada visualmente en torno a 10°-15°. Este mismo efecto, derivado fundamentalmente de las limitaciones de precisión e interferencias propias del magnetómetro, ya había sido detectado y documentado en trabajos previos de esta misma línea de investigación [2].

Para analizar este desfase, es necesario considerar las condiciones geométricas de la prueba. La comprobación visual se realizó con blancos situados a distancias relativamente cortas (inferiores a 1 milla náutica). A estas distancias, cualquier imprecisión lineal del sensor GPS del dispositivo local (por ejemplo, un error inherente de posicionamiento de unos 15 metros) se traduce en un error angular de paralaje muy marcado en el campo de visión. Se prevé que, a distancias mayores, este desvío visual sea mucho menos apreciable, ajustándose mejor a la línea de visión del buque.

Además del margen de error del GPS y de la brújula interna, otra causa directa de este desfase es la propia naturaleza del sistema AIS. Las posiciones se actualizan únicamente al recibir una nueva trama VHF (lo que puede demorarse varios segundos dependiendo del buque). Al carecer la aplicación de un algoritmo de interpolación predictiva que estime el movimiento entre recepciones, los marcadores actualizan su posición a saltos, lo que genera un retardo visual respecto a la posición real de un buque en navegación. Este fenómeno se evidencia claramente en la Figura 4-3: al observar los buques que se encuentran estáticos (atracados en el muelle), el desfase de los marcadores es mínimo, siendo atribuible exclusivamente a la imprecisión del *hardware*. Por el contrario, en los blancos en movimiento, la discrepancia es notablemente mayor. Esta comparativa confirma de manera empírica que el desvío total percibido es la suma del error estático de los sensores (GPS y magnetómetro) y el error dinámico derivado de la latencia de actualización del protocolo AIS.



Figura 4-3 Lanchas de instrucción de la Escuela Naval durante la maniobra de atraque (Elaboración propia)

Este fenómeno se pudo documentar gráficamente durante las pruebas al observar a las lanchas en movimiento. Como se ilustra en la comparativa de la Figura 4-4, el grado de precisión geométrica varía notablemente en función del momento de la captura.

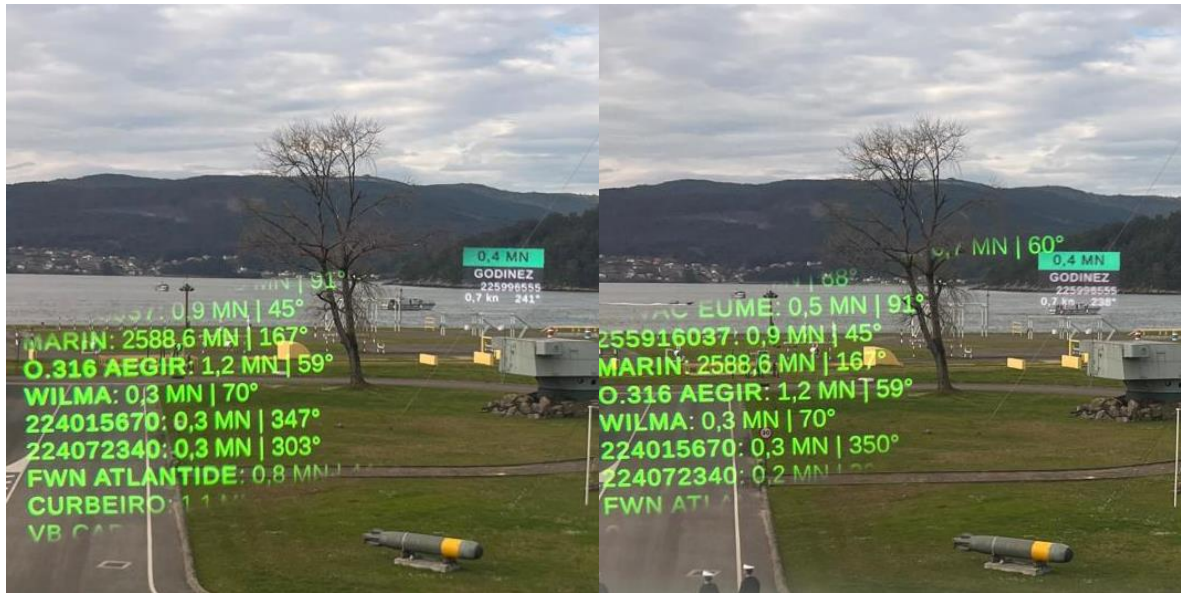


Figura 4-4 Comparativa del efecto de actualización de la trama AIS (Elaboración propia)

En la imagen de la derecha se puede apreciar cómo, en el instante exacto en el que el dispositivo recibe y procesa la nueva trama AIS, el marcador se reposiciona y la discrepancia visual frente a la embarcación se reduce de forma drástica. Esto confirma que una parte importante del error lateral observado se debe a la latencia de actualización del propio protocolo de comunicaciones y no a una falta de precisión de los sensores del visor.

Finalmente, derivado del exceso de contactos mencionados en el apartado anterior, se comprobó que cuando las embarcaciones estaban físicamente muy próximas entre sí, o alineadas en la misma línea de visión, sus marcadores gráficos se cruzaban y superponían, como se puede ver en la Figura 4-5. Esto provocó que en ciertos momentos la información resultase ligeramente incomprensible, reafirmando la necesidad de optimizar la gestión del espacio en pantalla. A pesar de estos inconvenientes, la estabilidad de los marcadores (*world-locked*) al mover la cabeza resultó coherente y fluida.

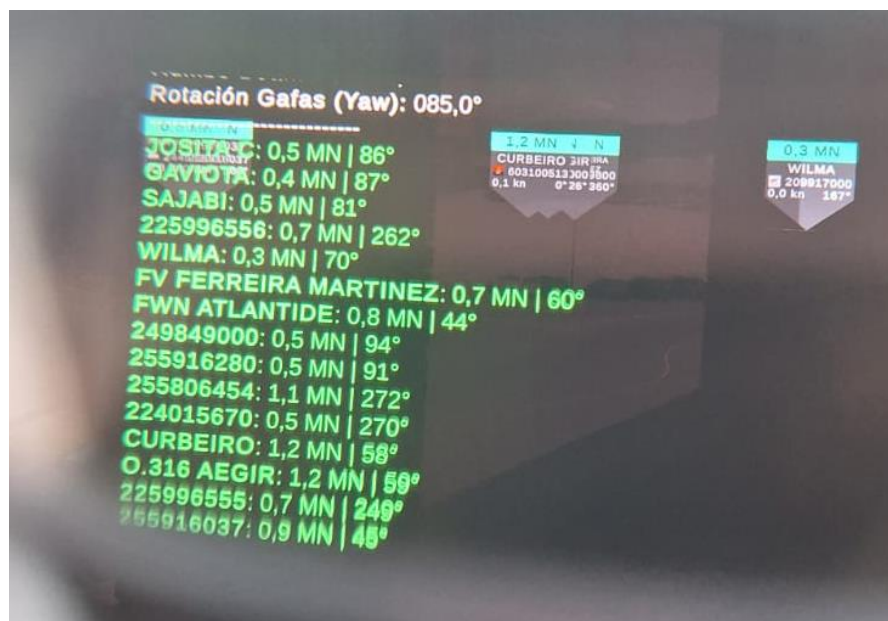


Figura 4-5 Superposición de marcadores (Elaboración propia)

4.4 Rendimiento del flujo de datos

En cuanto al ciclo de trabajo del sistema (desde que la antena recibe la trama AIS hasta que se renderiza en las gafas), el funcionamiento fue adecuado. Utilizando el dispositivo XREAL Beam Pro como unidad de procesamiento, la actualización de las posiciones se produjo de manera fiable tras cada recepción de datos.

La fluidez gráfica general se mantuvo constante, un factor vital en la realidad aumentada para evitar mareos o desorientación en el usuario. Se demostró que el *hardware* de bajo coste empleado posee la capacidad computacional suficiente para manejar el flujo de datos NMEA y renderizar la escena en estéreo sin cuellos de botella apreciables.

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este último capítulo se expone la valoración final del proyecto, analizando el grado de cumplimiento de los objetivos marcados al inicio de este. Asimismo, a partir de las limitaciones identificadas durante las pruebas de campo, se proponen diversas líneas de desarrollo futuro para continuar mejorando el sistema.

5.1 Conclusiones

El objetivo principal de este Trabajo Fin de Grado ha consistido en desarrollar y validar un sistema de visualización geoespacial inmersiva empleando gafas de realidad aumentada de bajo coste. Para materializar este propósito y evaluar su viabilidad operativa, se definió como caso de uso práctico la integración y representación espacial de datos tácticos AIS en tiempo real, superando así las limitaciones de fases anteriores del proyecto.

Tras concluir las etapas de diseño, integración de *hardware* y pruebas de campo, se extraen las siguientes conclusiones:

1. **Validación de la arquitectura de *hardware*:** La transición hacia un dispositivo específico de computación espacial (*Spatial Computing*), como el XREAL Beam Pro, ha resultado ser un éxito para la representación geoespacial. El sistema ha demostrado tener la capacidad computacional necesaria para gestionar simultáneamente la decodificación de tramas NMEA y el renderizado gráfico en Unity, sin generar cuellos de botella ni latencias perceptibles.
2. **Estabilidad del entorno virtual:** La integración y fusión de los datos del magnetómetro y la Unidad de Medición Inercial (IMU) ha permitido instanciar la información gráfica de manera estable sobre el entorno físico (*world-locked*). Los marcadores mantienen su coherencia espacial frente a los movimientos del operador, facilitando la correlación visual rápida entre el dato digital y el blanco real.
3. **Limitaciones ópticas del *hardware* de bajo coste:** Se ha comprobado empíricamente que las pantallas *optical see-through* comerciales presentan dificultades de contraste bajo luz solar directa. Aunque el uso de un filtro oscurecedor pasivo mitiga el problema de legibilidad de la interfaz, su opacidad excesiva puede llegar a comprometer la confirmación visual directa del entorno. Esto evidencia que el uso de *hardware* de bajo coste en el ámbito naval requiere asumir ciertos compromisos operativos.
4. **Análisis del error de posicionamiento geoespacial:** Durante las pruebas físicas se identificó un ligero desfase geométrico entre los marcadores virtuales y los buques. El análisis de este fenómeno concluye que el error no deriva únicamente de la imprecisión inherente del

magnetómetro o del GPS a distancias cortas (paralaje), sino fundamentalmente de la latencia del protocolo AIS. La baja tasa de actualización de las tramas de radio frente al movimiento continuo de las embarcaciones genera un desfase visual dinámico ineludible si no se aplican correcciones por *software*.

5. **Saturación de la interfaz de usuario:** La prueba en un entorno real con alto tráfico marítimo puso de manifiesto un problema de saturación visual (*clutter*). La acumulación de etiquetas de buques dinámicos, fondeados y atracados llega a solapar la información, dificultando la lectura y desplazando elementos vitales de la interfaz fuera del campo de visión del operador.

En síntesis, el proyecto ha logrado demostrar la viabilidad técnica de implementar una herramienta de visualización geoespacial de bajo coste que resulta funcional como asistencia a la navegación, sentando una base sólida sobre la que iterar en futuras investigaciones.

5.2 Líneas futuras

A la vista de los resultados obtenidos y de las limitaciones detectadas durante la fase de validación, se proponen las siguientes líneas de trabajo para dar continuidad al proyecto:

- **Integración con sistemas ECDIS y cartografía náutica:** Con el objetivo de enriquecer al máximo la conciencia situacional del navegante, se propone la integración de la aplicación con un sistema ECDIS (*Electronic Chart Display and Information System*). Esto permitiría superponer en el visor no solo los blancos AIS, sino también la información de las cartas náuticas electrónicas (ENC), mostrando peligros aislados, boyas, zonas de exclusión y las derrotas (rutas) planificadas directamente sobre el horizonte real.
- **Cálculo de cinemática y alarmas anticolidión visuales:** Aprovechando los datos de rumbo y velocidad decodificados de las tramas AIS, el sistema podría calcular automáticamente parámetros críticos como el CPA (*Closest Point of Approach*) y el TCPA (*Time to CPA*). Si estos valores caen por debajo de un umbral de seguridad predefinido, la interfaz podría cambiar el color del marcador del buque implicado (por ejemplo, a rojo) y emitir una alerta visual y representar gráficamente las derrotas previstas junto con el posible punto de colisión, aumentando de forma proactiva la seguridad en la navegación.
- **Evolución y escalabilidad del *hardware*:** Para superar las limitaciones ópticas detectadas y dar el salto al siguiente nivel tecnológico, se plantea la evaluación de visores de realidad aumentada de prestaciones superiores. Si se desea mantener y rentabilizar la infraestructura de computación espacial ya desarrollada, se podrían integrar modelos más avanzados como las XREAL Air 2 Ultra, que ofrecen un mejor seguimiento espacial (6DoF) y mayor campo de visión. Alternativamente, si se asume un incremento en el coste, se podría explorar la migración del proyecto hacia gafas de arquitectura *standalone*, eliminando por completo la dependencia de un dispositivo anfitrión cableado.
- **Desarrollo de un algoritmo de interpolación predictiva (*Dead Reckoning*):** Para solucionar el desfase visual de los contactos en movimiento causado por la latencia de actualización del AIS, se propone implementar un modelo matemático que estime la posición geoespacial en tiempo real de la embarcación basándose en su último rumbo y velocidad conocidos, logrando un movimiento fluido de los marcadores.
- **Implementación de filtros de visualización dinámicos:** Para resolver el problema de la saturación visual (*clutter*), resulta prioritario desarrollar un menú interactivo que permita al usuario filtrar los contactos. Se podrían aplicar filtros por estado de navegación (ocultando barcos atracados o fondeados) o por rango de distancia, limpiando el campo de visión de información irrelevante.
- **Pruebas dinámicas en navegación:** Es fundamental realizar una campaña de pruebas a bordo de una embarcación en alta mar para evaluar la respuesta del algoritmo de estabilización frente

a los movimientos propios de la dinámica del buque (balanceo, cabeceo y guiñada), así como comprobar la precisión de la actualización geoespacial con el dispositivo anfitrión en movimiento continuo.

6 BIBLIOGRAFÍA

- [1] Á. Gamboa Ramos Izquierdo, «Desarrollo de aplicación para la identificación de embarcaciones y elementos del entorno marítimo mediante realidad aumentada (Trabajo de fin de grado),» 2022. [En línea]. Available: <https://calderon.cud.uvigo.es/handle/123456789/679>.
- [2] P. Díaz García, «Desarrollo de una aplicación de Realidad Aumentada para representación de datos AIS en tiempo real (Trabajo Fin de Grado),» 2025. [En línea]. Available: <https://calderon.cud.uvigo.es/handle/123456789/924>.
- [3] International Telecommunication Union, «Technical characteristics for an automatic identification system using time division multiple access in the VHF maritime mobile band,» ITU, Geneva, Switzerland, 2014.
- [4] Organización Marítima Internacional, «Convenio internacional para la seguridad de la vida humana en el mar (SOLAS),» OMI, Londres, UK, 1974.
- [5] National Marine Electronics Association (NMEA), «NMEA 0183 Standard for Interfacing Marine Electronic Devices,» National Marine Electronics Association, 2008.
- [6] P. Milgram y F. Kishino, «A Taxonomy of Mixed Reality Visual Displays,» *IEICE Transactions on Information and Systems*, vol. 77, nº 12, pp. 1321-1329, 1994.
- [7] R. T. Azuma, «A Survey of Augmented Reality,» *Presence: Teleoperators and Virtual Environments*, vol. 4, nº 355-385, p. 6, 1997.
- [8] D. Schmalstieg y T. Höllerer, *Augmented Reality: Principles and Practice*, Addison-Wesley Professional, 2016.
- [9] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani y M. Ivkovic, «Augmented reality technologies, systems and applications,» *Multimedia Tools and Applications*, vol. 51, nº 1, pp. 341-377, 2011.
- [10] J. P. Rolland y H. Fuchs, «Optical versus video see-through head-mounted displays in medical visualization,» *Presence: Teleoperators and Virtual Environments*, vol. 9, nº 3, pp. 287-309, 2000.

- [11] B. Kress y T. Starner, «A review of head-mounted displays (HMD) technologies and applications for consumer electronics,» de *Proceedings of SPIE*, 2013.
- [12] E. G. Lee, J. S. Chen y Y. P. Huang, «High-resolution microdisplay technologies for augmented reality,» *Journal of Information Display*, vol. 21, nº 4, pp. 177-187, 2020.
- [13] V. H. M. Heinen, «Tracking systems in Virtual Reality: A review of current technologies,» *Journal of XR Research*, vol. 2, nº 45-58, p. 4, 2023.
- [14] G. Klein y D. Murray, «Parallel Tracking and Mapping for Small AR Workspaces,» de *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.
- [15] D. Li, H. Dong, X. Chen, B. Li, S. Zhang y X. Wu, «WS-SLAM: a monocular visual SLAM algorithm considering dynamic invalid features of water surface for USV,» *Systems Science & Control Engineering*, vol. 13, nº 1, 2025.
- [16] J. e. a. Silva, «Visual and Visual-Inertial SLAM for UGV Navigation in Unstructured Natural Environments: A Survey of Challenges and Deep Learning Advances,» *Sensors*, vol. 23, nº 15, 2023.
- [17] E. Vargas, «Robust Underwater and Surface Visual SLAM Fusing Acoustic Sensing and Inertial Data,» de *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [18] Sony Semiconductor Solutions Corporation, «Sony Semiconductor,» 2023. [En línea]. Available: <https://www.sony-semicon.com/en/products/microdisplay/oled.html>. [Último acceso: 25 02 2026].
- [19] XREAL, «XREAL Air 2 - Especificaciones Técnicas,» 2023. [En línea]. Available: <https://www.xreal.com/es/air2/>. [Último acceso: 25 02 2026].
- [20] XREAL, «XREAL Air 2 Ultra: The ultimate spatial computing glasses,» 2024. [En línea]. Available: <https://www.xreal.com/air2ultra/>. [Último acceso: 25 02 2026].
- [21] Qualcomm Technologies, «Snapdragon XR2 5G Platform,» 2023. [En línea]. Available: <https://www.qualcomm.com/products/mobile/snapdragon/xr-vr-ar/snapdragon-xr2-5g-platform>. [Último acceso: 26 02 2026].
- [22] RayNeo, «TCL RayNeo X2 - True Smart AR Glasses with Micro-LED,» 2024. [En línea]. Available: <https://www.rayneo.com/products/tcl-rayneo-x2>. [Último acceso: 26 02 2026].
- [23] RayNeo, «RayNeo XR Glasses Specifications,» 2024. [En línea]. Available: <https://www.rayneo.com/>. [Último acceso: 26 02 2026].
- [24] Meta Platforms, Inc., «Meta Quest 3: Características y especificaciones de la realidad mixta,» 2023. [En línea]. Available: <https://www.meta.com/quest/quest-3/>. [Último acceso: 26 02 2026].
- [25] S. Balakrishnan, M. Hameed y G. Aswin, «Interaction of Spatial Computing In Augmented Reality,» de *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2021.
- [26] B. C. Arslan y A. A. Alatan, «Motion model transitions in GPS-IMU sensor fusion for user tracking in augmented reality,» de *2015 23rd Signal Processing and Communications Applications Conference (SIU)*, Malatya, Turkey, 2015.

- [27] T. W. C. Brown y S. R. Saunders, «Performance of an AIS receiver based on software defined radio,» de *2009 International Conference on Electromagnetics in Advanced Applications (ICEAA*, Turin, Italy, 2009.
- [28] J. Linowes, *Unity Virtual Reality Projects: Learn Virtual Reality by developing more than 10 engaging projects with Unity 2018*, Birmingham, UK: Packt Publishing, 2018.
- [29] P. A. Zandbergen, «Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning,» *Transactions in GIS*, vol. 13, nº s1, pp. 5-25, Transactions in GIS.
- [30] W. R. Stevens, *UNIX Network Programming, Volume 1: The Sockets Networking API*, Boston, MA: Addison-Wesley, 2003.
- [31] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki y S. Hillaire, *Real-Time Rendering*, Boca Raton, FL: A K Peters/CRC Press, 2018.
- [32] «Web de La Moncloa,» [En línea]. Available: <http://www.lamoncloa.gob.es>. [Último acceso: 13 enero 2015].
- [33] J. Rodríguez y V. Fernández, *Cómo redactar el estado del arte de un trabajo*, Editorial Genios, 2010.
- [34] P. Martínez y A. García, *Cómo escribir una buena memoria de TFG*, Publicaciones del 2000, 2013.
- [35] A. Pérez, *Cómo escribir una bibliografía*, Nuevas publicaciones.
- [36] N. C. Karmakar, S. V. Aswath y M. R. N. Patra, «A Software Defined Radio based AIS Receiver for Maritime Surveillance,» de *2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE)*, Bhubaneswar, India, 2018.
- [37] U. Technologies, «Input Manager - Unity User Manual,» Unity Documentation, 10 Mayo 2024. [En línea]. Available: <https://docs.unity3d.com/Manual/class-InputManager.html>. [Último acceso: 14 Marzo 2026].
- [38] J. P. Rolland y H. Fuchs, «Optical versus video see-through head-mounted displays in medical visualization,» *Presence: Teleoperators and Virtual Environments*, vol. 9, nº 3, pp. 287-309, 2000.
- [39] M. A. Livingston, L. J. Rosenblum, S. J. Julier, D. Brown, Y. Baillot, J. E. Swan, J. L. Gabbard y D. Hix, «An augmented reality system for military operations in urban terrain,» de *Proceedings of SPIE*, 2002.
- [40] A. Wahyudiono, «El sistema AIS aerotransportado (Ilustración),» *Yachting.com*, 14 09 2023. [En línea]. Available: <https://www.yachting.com/es-es/news/what-is-ais-and-how-it-works>. [Último acceso: 17 03 2026].
- [41] C. Servin, «Proyecto IDIS,» 26 08 1968. [En línea]. Available: <https://proyectoidis.org/espada-de-damocles/>. [Último acceso: 17 03 2026].
- [42] Y. Itoh, T. Kaminokado y K. Akşit, «Beaming Displays,» *IEEE Transactions on Visualization and Computer Graphics*, p. 4, 2021.
- [43] Qualcomm Technologies, Inc., «Qualcomm,» 08 2017. [En línea]. Available: <https://www.qualcomm.com/content/dam/qcomm-martech/dm->

assets/documents/presentation_-_on-device_motion_tracking_for_immersive_vr.pdf.
[Último acceso: 03 17 2026].

- [44] XREAL, «Gafas de Realidad Aumentada XREAL Air 2 (Imagen en línea),» [En línea]. Available: <https://m.media-amazon.com/images/I/51Lpgv4wTZL.jpg>. [Último acceso: 25 02 2026].

ANEXO I: IMPLICACIONES SOCIALES, Y/O ECONÓMICAS, Y/O AMBIENTALES

Desde el punto de vista social y humano, el impacto fundamental del proyecto recae en la mejora directa de la seguridad de la vida humana en la mar. La navegación, especialmente en aguas restringidas o con alta densidad de tráfico, exige una carga cognitiva muy elevada por parte del oficial de guardia, que debe alternar constantemente su vista entre el radar, la carta electrónica y el entorno físico. El sistema de realidad aumentada desarrollado mitiga esta fatiga mental al superponer la información táctica directamente sobre la línea de visión del navegante. Esta ergonomía en la presentación de los datos agiliza la toma de decisiones en situaciones de alto estrés y reduce la probabilidad del error humano, que constituye históricamente la principal causa de los siniestros marítimos.

Esta mejora en la seguridad de la navegación actúa, de forma intrínseca, como una medida de protección medioambiental de primer orden. La prevención activa de colisiones marítimas es la herramienta más eficaz para evitar desastres ecológicos en el ecosistema marino, previniendo vertidos de hidrocarburos o pérdidas de cargas altamente contaminantes. Adicionalmente, si se evalúa el impacto desde la perspectiva del ciclo de vida del *hardware*, el uso de dispositivos comerciales ligeros (*Spatial Computing*) conlleva un consumo energético y una huella de carbono de fabricación muy inferiores a los de las pesadas consolas tradicionales de puente. No obstante, cabe destacar que una futura adopción a gran escala de esta tecnología exigiría contemplar la gestión del fin de vida útil de los equipos, aplicando las normativas vigentes de reciclaje de residuos electrónicos para el tratamiento de sus baterías y componentes ópticos.

Finalmente, el factor económico es el eje vertebrador que hace viable la implementación de todos estos beneficios. Mientras que el mercado actual de visores de realidad aumentada de grado industrial o militar maneja presupuestos sumamente restrictivos, este proyecto demuestra empíricamente que es posible lograr resultados operativos fiables mediante la integración de *hardware* COTS (como las gafas XREAL) y *software* de desarrollo accesible (Unity). Esta drástica reducción en los costes de adquisición, desarrollo y mantenimiento democratiza el acceso a la seguridad marítima. De este modo, no solo se facilita la modernización escalonada y asequible de los equipos de navegación en el ámbito de la Defensa, sino que se abre la posibilidad de extender esta tecnología de asistencia a embarcaciones civiles y de recreo con menores recursos, optimizando de forma sobresaliente la relación coste-beneficio.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

En el presente anexo se expone una reflexión crítica sobre las implicaciones éticas y el impacto social derivados del desarrollo y la posible implementación del sistema de realidad aumentada para la visualización de datos AIS en el entorno marítimo.

Desde una perspectiva ética, la introducción de nuevas tecnologías inmersivas en el puente de mando plantea el debate fundamental sobre la dependencia tecnológica y la delegación de responsabilidades. Es imperativo subrayar que la aplicación desarrollada se concibe estrictamente como una herramienta de asistencia a la navegación, destinada a complementar y enriquecer la conciencia situacional del operador. En ningún caso este sistema exime al oficial de guardia de sus obligaciones, ni actúa como sustituto del juicio humano, la vigilancia visual directa o el estricto cumplimiento del Reglamento Internacional para Prevenir Abordajes (RIPA). La responsabilidad última de la toma de decisiones y de la seguridad de la embarcación recae invariablemente en el navegante.

Ligado a esta responsabilidad operativa, surge un compromiso ético ineludible en la fase de diseño de la interfaz de usuario. Como se constató durante las pruebas físicas con el problema de la saturación visual (*clutter*), proporcionar un exceso de datos puede generar el efecto diametralmente opuesto al deseado: la distracción del operador, la fatiga visual o la pérdida de contacto con la realidad física. Por consiguiente, el desarrollo tecnológico debe regirse por un principio de prudencia y ergonomía cognitiva, garantizando que la información mostrada sea siempre clara, concisa y filtrada según su nivel de criticidad, evitando que la tecnología se convierta en un factor de riesgo añadido.

En el plano social, el impacto de este proyecto es netamente positivo y transformador. La industria marítima tradicional se caracteriza por un acceso muy asimétrico a las tecnologías de vanguardia en materia de seguridad, quedando los sistemas integrados de alta gama reservados habitualmente a grandes buques mercantes o unidades militares. Al cimentar este desarrollo sobre *hardware* de computación espacial de bajo coste y *software* accesible, se fomenta una verdadera democratización de la seguridad en la mar. Esta filosofía de diseño permite que sectores náuticos con menores recursos económicos, tales como la pesca artesanal, los servicios de salvamento locales o la navegación de recreo, puedan acceder a herramientas de prevención de accidentes que antes resultaban inalcanzables. En definitiva, el proyecto no solo busca un avance puramente técnico, sino que persigue una mejora tangible en las condiciones laborales de los profesionales de la mar, reduciendo su estrés cognitivo y contribuyendo activamente a salvaguardar la vida humana en un entorno inherentemente hostil.

ANEXO III: USO DE LA INTELIGENCIA ARTIFICIAL

En el marco de la elaboración del presente Trabajo Fin de Grado, se hace constar que se ha empleado la herramienta de Inteligencia Artificial (IA) generativa Gemini, desarrollada por Google, como un recurso de apoyo tecnológico y metodológico en diversas fases del proyecto. El uso de esta tecnología se ha concebido bajo un prisma de asistencia técnica, donde la IA ha actuado como un facilitador para optimizar procesos específicos, siempre supeditada al criterio, la validación y la autoría intelectual del alumno.

En lo que respecta a la redacción de la memoria, Gemini ha sido utilizada para el refinamiento del lenguaje y la mejora de la coherencia gramatical, ayudando a adecuar el estilo a los estándares de rigor académico y profesional exigidos, pero partiendo siempre de las ideas, conceptos y estructuras originales planteadas por el autor. Asimismo, en el ámbito del desarrollo de *software*, se ha recurrido a la metodología de *vibe coding*, empleando la IA como un soporte para la generación de estructuras de código base, la resolución de errores sintácticos y la optimización de algoritmos complejos para el procesamiento de los datos AIS. No obstante, la arquitectura del sistema, la lógica de programación y la integración final del *software* en el entorno de RA han sido fruto de la toma de decisiones directa del autor, quien ha supervisado y verificado cada línea de código para asegurar su correcto funcionamiento. Finalmente, la IA también ha servido de apoyo en la generación de recursos visuales e imágenes complementarias que facilitan la comprensión de los conceptos teóricos expuestos.

Se ratifica, por tanto, que el uso de Gemini no ha sustituido en ningún caso el proceso de aprendizaje ni la responsabilidad técnica del autor, siendo este el único responsable de la veracidad de los datos, la ejecución del *hardware* y las conclusiones derivadas de la investigación.

ANEXO IV: COMPASSTEST.CS

```
using System.Collections;
using UnityEngine;
using TMPro;

public class CompassDebug : MonoBehaviour
{
    public TextMeshProUGUI txt;

    IEnumerator Start()
    {
        if (txt != null)
            txt.text = "Iniciando...";

        // 1) Activar brújula
        Input.compass.enabled = true;

        // 2) Intentar arrancar localización para forzar permisos y ayudar a trueHeading
        if (SystemInfo.supportsLocationService)
        {
            Input.location.Start();
        }

        // 3) Esperar un poco a que el sensor arranque
        int maxWait = 10;
        while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0)
        {
            if (txt != null)
                txt.text = $"Iniciando localización... {maxWait}";
            yield return new WaitForSeconds(1f);
            maxWait--;
        }

        if (txt != null)
        {
            txt.text =
                "Listo\n" +
                $"supportsLocation: {SystemInfo.supportsLocationService}\n" +
                $"locationStatus: {Input.location.status}\n" +
                $"compassEnabled: {Input.compass.enabled}\n";
        }
    }

    void Update()
    {
        if (txt == null) return;

        txt.text =
            $"compassEnabled: {Input.compass.enabled}\n" +
            $"timestamp: {Input.compass.timestamp:F2}\n" +
            $"headingAccuracy: {Input.compass.headingAccuracy:F2}\n" +
            $"magHeading: {Input.compass.magneticHeading:F1}\n" +
            $"trueHeading: {Input.compass.trueHeading:F1}\n" +
            $"raw: {Input.compass.rawVector.x:F2}, {Input.compass.rawVector.y:F2},
{Input.compass.rawVector.z:F2}\n" +
            $"locationStatus: {Input.location.status}";
    }
}
```


ANEXO V: CÓDIGO SENSORINTEGRATIONHUD.CS

```
using System.Collections;
using UnityEngine;
using TMPro;
#if UNITY_ANDROID
using UnityEngine.Android;
#endif

public class SensorIntegrationHUD : MonoBehaviour
{
    public TextMeshProUGUI txt;

    IEnumerator Start()
    {
        if (txt != null)
            txt.text = "<color=#FFA500>Iniciando protocolo de sensores...</color>";

        // 1. Solicitud de permisos explícita para el sistema operativo Android
#if UNITY_ANDROID
        if (!Permission.HasUserAuthorizedPermission(Permission.FineLocation))
        {
            Permission.RequestUserPermission(Permission.FineLocation);
            // Se otorga un margen de tiempo para que el usuario acepte el diálogo
            yield return new WaitForSeconds(3f);
        }
#endif

        // 2. Comprobación de hardware
        if (!SystemInfo.supportsLocationService)
        {
            if (txt != null)
                txt.text = "<color=#FF0000>Error: El dispositivo carece de receptor  
GPS.</color>";
            yield break;
        }

        // 3. Activación de sensores
        Input.compass.enabled = true;

        // Se inicializa el servicio de ubicación con parámetros de precisión y distancia de
        // actualización (5 metros)
        Input.location.Start(5f, 5f);

        // 4. Bucle de espera para la adquisición de satélites (Fijación de posición)
        int maxWait = 20;
        while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0)
        {
            if (txt != null)
                txt.text = $"<color=#FFFF00>Adquiriendo señal GPS... Tiempo restante:
{maxWait} s</color>";
            yield return new WaitForSeconds(1f);
            maxWait--;
        }

        // 5. Evaluación final del estado de inicialización
        if (maxWait < 1)
        {
            if (txt != null)
                txt.text = "<color=#FF0000>Tiempo de espera agotado. No se logró fijar la  
posición GPS.</color>";
            yield break;
        }
    }
}
```

```
    }

    if (Input.location.status == LocationServiceStatus.Failed)
    {
        if (txt != null)
            txt.text = "<color=#FF0000>Error: El servicio de ubicación falló al intentar
inicializarse.</color>";
        yield break;
    }
}

void Update()
{
    if (txt == null) return;

    // Variables de la Brújula (Magnetómetro)
    float magHeading = Input.compass.magneticHeading;
    float trueHeading = Input.compass.trueHeading;
    float compassAcc = Input.compass.headingAccuracy;

    // Variables del GPS
    string gpsData = "";

    if (Input.location.status == LocationServiceStatus.Running)
    {
        float lat = Input.location.lastData.latitude;
        float lon = Input.location.lastData.longitude;
        float alt = Input.location.lastData.altitude;
        float acc = Input.location.lastData.horizontalAccuracy;

        // Formateo de las coordenadas geográficas con alineación de columnas
        gpsData =
            $"Latitud:           {lat,10:F5}°\n" +
            $"Longitud:          {lon,10:F5}°\n" +
            $"Altitud (ASL):       {alt,10:F1} m\n" +
            $"Precisión:           {acc,10:F1} m";
    }
    else
    {
        gpsData = "Calculando posición espacial...\n\n\n";
    }

    // Representación final en la interfaz de RA con formato limpio
    txt.text =
        "<size=120%><b><color=#FFA500>DIRECCION Y POSICIONAMIENTO</color></b></size>\n\n"
+
        "<b>[ MÓDULO IMU / BRÚJULA ]</b>\n" +
        $"Rumbo Verdadero:   {trueHeading,10:F1}°\n" +
        $"Rumbo Magnético:    {magHeading,10:F1}°\n" +
        $"Precisión:           {compassAcc,10:F1}°\n\n" +
        "<b>[ MÓDULO GPS ]</b>\n" +
        $" {gpsData}";
    }
}
```

ANEXO VI: CÓDIGO GEOMARKERTEST.CS

```
using System.Collections;
using UnityEngine;
using TMPro;
using System;
#if UNITY_ANDROID
using UnityEngine.Android;
#endif

public class GeoMarkerTest : MonoBehaviour
{
    [Header("Interfaz y Cámara")]
    public TextMeshProUGUI txt;
    public Transform arCamera;
    [Header("Marcador 3D (Faro)")]
    public Transform waypointMarker;
    public float visualDistance = 20f;

    [Header("Coordenadas Objetivo (Faro Tenlo Chico)")]
    public float targetLat = 42.3986f;
    public float targetLon = -8.7058f;

    // Constante del radio de la Tierra en metros
    private const float R = 6371e3f;

    IEnumerator Start()
    {
        if (txt != null) txt.text = "<color=#FFA500>Iniciando sensores y marcador...</color>";
    }

#if UNITY_ANDROID
    if (!Permission.HasUserAuthorizedPermission(Permission.FineLocation))
    {
        Permission.RequestUserPermission(Permission.FineLocation);
        yield return new WaitForSeconds(3f);
    }
#endif

    if (!SystemInfo.supportsLocationService) yield break;

    Input.compass.enabled = true;
    Input.location.Start(5f, 5f);

    int maxWait = 20;
    while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0)
    {
        yield return new WaitForSeconds(1f);
        maxWait--;
    }

    void Update()
    {
        if (Input.location.status != LocationServiceStatus.Running) return;

        // 1. Lectura de la posición actual
        float currentLat = Input.location.lastData.latitude;
        float currentLon = Input.location.lastData.longitude;
        float trueHeading = Input.compass.trueHeading;

        // 2. Cálculo de la Distancia (Haversine) y Marcación (Bearing)
        float distance = CalculateDistance(currentLat, currentLon, targetLat, targetLon);
```

```
float targetBearing = CalculateBearing(currentLat, currentLon, targetLat, targetLon);

// 3. Posicionamiento del Marcador 3D en Realidad Aumentada
if (waypointMarker != null && arCamera != null)
{
    // Calculamos la diferencia entre hacia dónde debes mirar (target) y hacia dónde
    miras (heading)
    float relativeBearing = targetBearing - trueHeading;

    // Rotamos el vector frontal de la cámara en el eje Y según esa diferencia
    Vector3 directionToTarget = Quaternion.Euler(0, relativeBearing, 0) *
arCamera.forward;

    // Forzamos que la altura (Y) sea la misma que la cámara para que no se entierre
    ni vuele demasiado
    directionToTarget.y = 0;
    directionToTarget.Normalize();

    // Colocamos el marcador a una distancia visible fija frente al usuario en esa
    dirección
    waypointMarker.position = arCamera.position + (directionToTarget *
visualDistance);

    // Hacemos que el marcador siempre mire hacia el usuario
    waypointMarker.LookAt(arCamera);
}

// 4. Actualización de la Interfaz
// 4. Actualización de la Interfaz (Formato Tabla HUD)
txt.text =
    "<size=110%><b><color=#00BFFF>DATOS DE NAVEGACIÓN</color></b></size>\n" +
    "-----\n" +
    $"<b>Demora (Rumbo):</b>    {trueHeading:000.0}°\n" +
    $"<b>Latitud:</b>           {currentLat:F5}°\n" +
    $"<b>Longitud:</b>          {currentLon:F5}°\n" +
    $"<b>Dist. al Faro:</b>     {distance:F0} m";
}

// Funciones matemáticas auxiliares (Corregidas para Doble Precisión)
private float CalculateDistance(float lat1, float lon1, float lat2, float lon2)
{
    double R = 6371e3; // Radio de la Tierra en metros

    // Conversión a Radianes usando Double
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaPhi = (lat2 - lat1) * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;

    // Fórmula del Haversine en Doble Precisión
    double a = Math.Sin(deltaPhi / 2.0) * Math.Sin(deltaPhi / 2.0) +
        Math.Cos(phi1) * Math.Cos(phi2) *
        Math.Sin(deltaLambda / 2.0) * Math.Sin(deltaLambda / 2.0);
    double c = 2.0 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a));

    return (float)(R * c);
}

private float CalculateBearing(float lat1, float lon1, float lat2, float lon2)
{
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;
```

```
double y = Math.Sin(deltaLambda) * Math.Cos(phi2);
double x = Math.Cos(phi1) * Math.Sin(phi2) -
          Math.Sin(phi1) * Math.Cos(phi2) * Math.Cos(deltaLambda);

double theta = Math.Atan2(y, x);

// Retornamos el valor en grados normalizado a 0-360
return (float)((theta * 180.0 / Math.PI + 360.0) % 360.0);
}
```


ANEXO VII: CÓDIGO MULTIMARKERTRACKER.CS

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
#if UNITY_ANDROID
using UnityEngine.Android;
#endif

[System.Serializable]
public class GeoWaypoint
{
    public string waypointName;
    public double latitude;
    public double longitude;
    [HideInInspector] public GameObject markerInstance;
    [HideInInspector] public double currentDistance;
    [HideInInspector] public double currentBearing;

    // Constructor para facilitar la creación por código
    public GeoWaypoint(string name, double lat, double lon)
    {
        waypointName = name;
        latitude = lat;
        longitude = lon;
    }
}

public class MultiMarkerTracker : MonoBehaviour
{
    [Header("Interfaz y Cámara")]
    public TextMeshProUGUI txt;
    public Transform arCamera;

    [Header("Configuración de Marcadores")]
    public GameObject markerPrefab;
    public float visualDistance = 20f;

    [Header("Lista de Objetivos")]
    public List<GeoWaypoint> waypoints = new List<GeoWaypoint>();

    private const double R = 6371e3;

    IEnumerator Start()
    {
        // --- CARGA AUTOMÁTICA DE PUNTOS ---
        if (waypoints.Count == 0)
        {
            waypoints.Add(new GeoWaypoint("Tenlochico", 42.40803, -8.70782));
            waypoints.Add(new GeoWaypoint("Carlos I", 42.39369, -8.70298));
            waypoints.Add(new GeoWaypoint("Atán", 42.39631, -8.71074));
            waypoints.Add(new GeoWaypoint("Torpedos", 42.39897, -8.70499));
        }

        if (txt != null) txt.text = "<color=#FFA500>Iniciando sensores...</color>";
    }

#if UNITY_ANDROID
    if (!Permission.HasUserAuthorizedPermission(Permission.FineLocation))
    {
        Permission.RequestUserPermission(Permission.FineLocation);
    }
#endif
}
```

```
        yield return new WaitForSeconds(3f);
    }
#endif

    if (!SystemInfo.supportsLocationService) yield break;

    Input.compass.enabled = true;
    Input.location.Start(5f, 5f);

    int maxWait = 20;
    while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0)
    {
        yield return new WaitForSeconds(1f);
        maxWait--;
    }

    // --- INICIALIZACIÓN DE LOS MARCADORES FÍSICOS ---
    if (Input.location.status == LocationServiceStatus.Running && markerPrefab != null)
    {
        foreach (var wp in waypoints)
        {
            wp.markerInstance = Instantiate(markerPrefab);

            TextMeshPro text3D = wp.markerInstance.GetComponentInChildren<TextMeshPro>();
            if (text3D != null)
            {
                text3D.text = wp.waypointName;
            }
        }
    }
}

void Update()
{
    if (Input.location.status != LocationServiceStatus.Running) return;

    double currentLat = Input.location.lastData.latitude;
    double currentLon = Input.location.lastData.longitude;
    float trueHeading = Input.compass.trueHeading;

    string debugData = "";

    foreach (var wp in waypoints)
    {
        if (wp.markerInstance == null) continue;

        wp.currentDistance = CalculateDistance(currentLat, currentLon, wp.latitude,
wp.longitude);
        wp.currentBearing = CalculateBearing(currentLat, currentLon, wp.latitude,
wp.longitude);

        float relativeBearing = (float)wp.currentBearing - trueHeading;
        Vector3 directionToTarget = Quaternion.Euler(0, relativeBearing, 0) *
arCamera.forward;
        directionToTarget.y = 0;
        directionToTarget.Normalize();

        wp.markerInstance.transform.position = arCamera.position + (directionToTarget *
visualDistance);
        wp.markerInstance.transform.LookAt(arCamera);
        wp.markerInstance.transform.Rotate(0, 180, 0);

        debugData += $"<b>{wp.waypointName}</b>: {wp.currentDistance:F0}m |
{wp.currentBearing:F1}°\n";
    }
}
```

```

    }

    txt.text =
        "<size=110%><b><color=#00BFFF>DATOS DE NAVEGACIÓN</color></b></size>\n" +
        "-----\n" +
        $"<b>Rumbo:</b>    {trueHeading:000.0}°\n" +
        $"<b>Lat:</b>      {currentLat:F5}°\n" +
        $"<b>Lon:</b>      {currentLon:F5}°\n" +
        "-----\n" +
        $"<color=#00FF00>{debugData}</color>";
}

private double CalculateDistance(double lat1, double lon1, double lat2, double lon2)
{
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaPhi = (lat2 - lat1) * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;

    double a = Math.Sin(deltaPhi / 2.0) * Math.Sin(deltaPhi / 2.0) +
        Math.Cos(phi1) * Math.Cos(phi2) *
        Math.Sin(deltaLambda / 2.0) * Math.Sin(deltaLambda / 2.0);
    double c = 2.0 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a));
    return R * c;
}

private double CalculateBearing(double lat1, double lon1, double lat2, double lon2)
{
    double phi1 = lat1 * Math.PI / 180.0;
    double phi2 = lat2 * Math.PI / 180.0;
    double deltaLambda = (lon2 - lon1) * Math.PI / 180.0;

    double y = Math.Sin(deltaLambda) * Math.Cos(phi2);
    double x = Math.Cos(phi1) * Math.Sin(phi2) -
        Math.Sin(phi1) * Math.Cos(phi2) * Math.Cos(deltaLambda);

    double theta = Math.Atan2(y, x);
    return (theta * 180.0 / Math.PI + 360.0) % 360.0;
}
}

```


ANEXO VIII: CÓDIGO UDP_AIS_RECEIVER.CS

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using UnityEngine;
using TMPro;

public class UDP_AIS_Receiver : MonoBehaviour
{
    [Header("Configuración de Red")]
    public int listenPort = 10112;

    [Header("UI Debug")]
    public TextMeshProUGUI debugText;

    private UdpClient udpClient;
    private Thread listenThread;
    private bool isListening = false;
    private string lastReceivedSentence = "";
    private readonly object lockObject = new object();

    void Start()
    {
        try
        {
            // Forzamos explícitamente a usar IPv4 (AddressFamily.InterNetwork)
            udpClient = new UdpClient(listenPort, AddressFamily.InterNetwork);
            isListening = true;

            listenThread = new Thread(ListenForData);
            listenThread.IsBackground = true;
            listenThread.Start();

            Debug.Log($"[AIS] Escuchando red local en el puerto {listenPort}...");
            if (debugText != null) debugText.text = $"Escuchando puerto {listenPort}...";
        }
        catch (Exception e)
        {
            Debug.LogError($"[AIS] Error de puerto: {e.Message}");
            if (debugText != null) debugText.text = $"Error: {e.Message}";
        }
    }

    private void ListenForData()
    {
        // 0 significa que escuche de cualquier IP que le hable
        IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, 0);

        while (isListening)
        {
            try
            {
                byte[] receiveBytes = udpClient.Receive(ref remoteEndPoint);
                string returnData = Encoding.ASCII.GetString(receiveBytes);

                lock (lockObject)
                {
                    lastReceivedSentence = returnData;
                }
            }
        }
    }
}
```

```
        }
        catch (SocketException)
    }
}

void Update()
{
    string sentenceToProcess = "";

    lock (lockObject)
    {
        if (!string.IsNullOrEmpty(lastReceivedSentence))
        {
            sentenceToProcess = lastReceivedSentence;
            lastReceivedSentence = "";
        }
    }
    // Si hemos recibido una sentencia nueva, la decodificamos
    if (!string.IsNullOrEmpty(sentenceToProcess))
    {
        AISData barcoDecodificado = AISParser.ParseAIVDM(sentenceToProcess);

        if (barcoDecodificado != null)
        {
            string mensajeExito = $"[BARCO CAZADO] MMSI: {barcoDecodificado.MMSI} | Lat: {barcoDecodificado.Latitude:F5} | Lon: {barcoDecodificado.Longitude:F5}";

            Debug.Log(mensajeExito);

            if (debugText != null)
            {
                debugText.text = mensajeExito;
            }
        }
    }
}

void OnDestroy()
{
    isListening = false;
    if (udpClient != null)
    {
        udpClient.Close();
    }
    if (listenThread != null && listenThread.IsAlive)
    {
        listenThread.Abort();
    }
    Debug.Log("[AIS] Puerto liberado correctamente.");
}
}
```

ANEXO IX: CÓDIGO AISPARSER.CS

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class AISData
{
    public int MMSI;
    public double Latitude;
    public double Longitude;
    public double SOG;
    public double COG;

    public bool isPositionReport;
    public bool isStaticReport;
    public string ShipName = "";
}

public static class AISParser
{
    // Diccionario para almacenar partes de mensajes largos sin que se pisen entre barcos
    private static Dictionary<string, string> partialMessages = new Dictionary<string,
string>();

    public static AISData ParseAIVDM(string nmeaSentence)
    {
        try
        {
            string[] parts = nmeaSentence.Split(',');
            if (parts.Length < 6) return null;

            string totalFragments = parts[1];
            string currentFragment = parts[2];
            string messageId = parts[3]; // ID secuencial
            string channel = parts[4]; // Canal de radio (A o B)

            // Creamos una llave única sumando el ID y el Canal para evitar colisiones
            string dictKey = messageId + "_" + channel;
            string payload = parts[5];

            // --- RECONSTRUCCIÓN DE MENSAJES LARGOS (Multilínea) ---
            if (totalFragments == "2")
            {
                if (currentFragment == "1")
                {
                    // Guardamos el fragmento 1 usando la llave única y esperamos
                    partialMessages[dictKey] = payload;
                    return null;
                }
                else if (currentFragment == "2")
                {
                    if (partialMessages.ContainsKey(dictKey))
                    {
                        // Encontramos la parte 1, las soldamos y borramos la memoria
                        payload = partialMessages[dictKey] + payload;
                        partialMessages.Remove(dictKey);
                    }
                    else
                    {
                        return null; // Nos llegó la parte 2 pero perdimos la 1
                    }
                }
            }
        }
    }
}
```

```
    }
}
else if (totalFragments != "1")
{
    return null; // Ignorar si son 3 o más trozos
}

// --- DECODIFICACIÓN BINARIA ---
string binaryString = PayloadToBinary(payload);
if (binaryString.Length < 6) return null;

int messageType = Convert.ToInt32(binaryString.Substring(0, 6), 2);

AISData data = new AISData();
data.MMSI = Convert.ToInt32(binaryString.Substring(8, 30), 2);

// Tipos 1, 2, 3 (Clase A) y 18 (Clase B) - Reportes de Posición y Velocidad
if (messageType == 1 || messageType == 2 || messageType == 3 || messageType == 18)
{
    data.isPositionReport = true;
    int lonStart = (messageType == 18) ? 57 : 61;
    int latStart = (messageType == 18) ? 85 : 89;

    data.Longitude = DecodeCoordinate(binaryString.Substring(lonStart, 28));
    data.Latitude = DecodeCoordinate(binaryString.Substring(latStart, 27));

    if (messageType != 18)
    {
        data.SOG = Convert.ToInt32(binaryString.Substring(50, 10), 2) / 10.0;
        data.COG = Convert.ToInt32(binaryString.Substring(116, 12), 2) / 10.0;
    }

    if (data.Latitude >= 91.0 || data.Longitude >= 181.0) return null;
    return data;
}

// Tipo 5 - Datos Estáticos CLASE A (Buques grandes comerciales)
if (messageType == 5)
{
    if (binaryString.Length >= 232)
    {
        data.isStaticReport = true;
        // El nombre empieza en el bit 112 y dura 120 bits (20 caracteres)
        data.ShipName = Decode6BitText(binaryString.Substring(112, 120));
        return data;
    }
}

// Tipo 24 - Datos Estáticos CLASE B (Pesqueros, veleros, barcos de recreo)
if (messageType == 24)
{
    if (binaryString.Length >= 160)
    {
        // El Tipo 24 tiene Parte A (0) y Parte B (1). El nombre viaja en la Parte

        int partNumber = Convert.ToInt32(binaryString.Substring(38, 2), 2);
        if (partNumber == 0)
        {
            data.isStaticReport = true;
            // En el Tipo 24A, el nombre empieza en el bit 40
            data.ShipName = Decode6BitText(binaryString.Substring(40, 120));
            return data;
        }
    }
}
```

A.

```

    }
}
catch (Exception) { /* Ignorar errores de trama para no colgar Unity si el WiFi falla
*/ }

return null;
}

// --- FUNCIONES INTERNAS DE MATEMÁTICAS Y TRADUCCIÓN ---

private static string PayloadToBinary(string payload)
{
    string binary = "";
    foreach (char c in payload)
    {
        int val = c - 48;
        if (val > 40) val -= 8;
        binary += Convert.ToString(val, 2).PadLeft(6, '0');
    }
    return binary;
}

private static double DecodeCoordinate(string binString)
{
    long val = Convert.ToInt64(binString, 2);
    if (binString[0] == '1') val = val - (1L << binString.Length);
    return val / 600000.0;
}

private static string Decode6BitText(string binString)
{
    // La tabla oficial AIS de 6 bits para decodificar caracteres sin fallos
    const string charMap = "@ABCDEFGHJKLMNPQRSTUVWXYZ[\\]^_ !\"#$%&'()*+,-
./0123456789;<=>?";

    string result = "";

    for (int i = 0; i + 6 <= binString.Length; i += 6)
    {
        string chunk = binString.Substring(i, 6);
        int val = Convert.ToInt32(chunk, 2);

        // Si el valor está dentro del rango, mapeamos el carácter
        if (val >= 0 && val < charMap.Length)
        {
            result += charMap[val];
        }
    }

    // Limpiamos los caracteres '@' de relleno y los espacios sobrantes para dejar el
    nombre limpio
    return result.Replace("@", "").Trim();
}
}

```


ANEXO X: CÓDIGO AIS_AR_TRACKER.CS

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.Concurrent;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using UnityEngine;
using TMPro;

#if UNITY_ANDROID
using UnityEngine.Android;
#endif

public class GeoWaypoint
{
    public string shipName = "";
    public double latitude;
    public double longitude;
    public double sog;
    public double cog;
    public GameObject markerInstance;
    public ShipMarkerUI uiController;
    public double currentDistance;
    public double currentBearing;
}

public class AIS_AR_Tracker : MonoBehaviour
{
    [Header("Interfaz y Cámara")]
    public TextMeshProUGUI txt;
    public Transform arCamera;

    [Header("Configuración de Marcadores")]
    public GameObject markerPrefab;
    public float visualDistance = 20f;

    [Header("Estabilización de Sensores")]
    [Range(0.5f, 10f)]
    public float compassSmoothing = 2.0f;
    private float smoothedHeading;

    [Header("Sincronización Automática")]
    public TextMeshProUGUI txtSyncMessage;
    private float yawOffset = 0f;
    private bool isSyncing = false;

    [Header("Red")]
    public int listenPort = 10112;

    private UdpClient udpClient;
    private Thread listenThread;
    private bool isListening = false;

    // --- Cola concurrente para evitar colisiones de datos ---
    private ConcurrentQueue<string> udpMessages = new ConcurrentQueue<string>();

    private Dictionary<int, GeoWaypoint> trackedShips = new Dictionary<int, GeoWaypoint>();
    private const double R = 6371e3;
```

```
IEnumerator Start()
{
    if (txt != null) txt.text = "<color=#FFA500>Iniciando sensores GPS y  
Brújula...</color>";
    if (txtSyncMessage != null) txtSyncMessage.gameObject.SetActive(false);

#if UNITY_ANDROID
    if (!Permission.HasUserAuthorizedPermission(Permission.FineLocation))
    {
        Permission.RequestUserPermission(Permission.FineLocation);
        yield return new WaitForSeconds(3f);
    }
#endif

    if (SystemInfo.supportsLocationService)
    {
        Input.compass.enabled = true;
        Input.location.Start(5f, 5f);

        int maxWait = 20;
        while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0)
        {
            yield return new WaitForSeconds(1f);
            maxWait--;
        }
    }

    StartUDPReceiver();
    smoothedHeading = Input.compass.trueHeading;

    // ---AUTO-CALIBRACIÓN AL ARRANCAR ---
    StartCoroutine(SyncRoutine());
}

private void StartUDPReceiver()
{
    udpClient = new UdpClient(listenPort, AddressFamily.InterNetwork);
    isListening = true;
    listenThread = new Thread(ListenForData);
    listenThread.IsBackground = true;
    listenThread.Start();
}

private void ListenForData()
{
    IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, 0);
    while (isListening)
    {
        try
        {
            byte[] receiveBytes = udpClient.Receive(ref remoteEndPoint);
            string returnData = Encoding.ASCII.GetString(receiveBytes);

            // ---Separar mensajes pegados y encolarlos ---
            string[] lines = returnData.Split(new[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries);
            foreach (string line in lines)
            {
                udpMessages.Enqueue(line);
            }
        }
        catch (SocketException) { }
    }
}
```

```

public void StartSyncRoutine()
{
    if (!isSyncing) StartCoroutine(SyncRoutine());
}

private IEnumerator SyncRoutine()
{
    isSyncing = true;
    if (txtSyncMessage != null)
    {
        txtSyncMessage.gameObject.SetActive(true);

        for (int i = 3; i > 0; i--)
        {
            txtSyncMessage.text = $"Alineando visor con el dispositivo...\nMire al  
frente\n<size=150%><b>{i}</b></size>";
            yield return new WaitForSeconds(1f);
        }

        // Momento Cero: Capturar el desfase actual del giroscopio
        yawOffset = arCamera.eulerAngles.y;

        txtSyncMessage.text = "<color=#0FF00>¡Sistema Calibrado!</color>";
        yield return new WaitForSeconds(1.5f);
        txtSyncMessage.gameObject.SetActive(false);
    }
    else
    {
        yield return new WaitForSeconds(3f);
        yawOffset = arCamera.eulerAngles.y;
    }
    isSyncing = false;
}

void Update()
{
    if (Input.location.status != LocationServiceStatus.Running) return;

    smoothedHeading = Mathf.LerpAngle(smoothedHeading, Input.compass.trueHeading,
    Time.deltaTime * compassSmoothing);
    smoothedHeading = Mathf.Repeat(smoothedHeading, 360f);

    float rawHeadYaw = arCamera.eulerAngles.y;

    float correctedYaw = Mathf.DeltaAngle(yawOffset, rawHeadYaw);
    if (correctedYaw < 0) correctedYaw += 360f;

    float gazeHeading = (smoothedHeading + correctedYaw) % 360f;

    double currentLat = Input.location.lastData.latitude;
    double currentLon = Input.location.lastData.longitude;

    // ---Procesar la cola entera de mensajes ---
    ProcessNMEASentences();

    string uiTableData = "";

    foreach (var kvp in trackedShips)
    {
        int mmsi = kvp.Key;
        GeoWaypoint ship = kvp.Value;
    }

```

```
        ship.currentDistance = CalculateDistance(currentLat, currentLon, ship.latitude,
ship.longitude);
        ship.currentBearing = CalculateBearing(currentLat, currentLon, ship.latitude,
ship.longitude);

        float relativeBearing = (float)ship.currentBearing - smoothedHeading;
        float worldAngle = yawOffset + relativeBearing;

        Vector3 directionToTarget = Quaternion.Euler(0, worldAngle, 0) * Vector3.forward;
        directionToTarget.y = 0;
        directionToTarget.Normalize();

        ship.markerInstance.transform.position = arCamera.position + (directionToTarget *
visualDistance);
        ship.markerInstance.transform.rotation =
Quaternion.LookRotation(ship.markerInstance.transform.position - arCamera.position);

        if (ship.uiController != null)
        {
            double distMN = ship.currentDistance / 1852.0;
            ship.uiController.UpdateData(ship.shipName, distMN, mmsi, ship.sog, ship.cog);
        }

        string displayName = string.IsNullOrEmpty(ship.shipName) ? mmsi.ToString() :
ship.shipName;
        uiTableData += $"<b>{displayName}</b>: {(ship.currentDistance / 1852):F1} MN |
{ship.currentBearing:F0}°\n";
    }

    if (txt != null)
    {
        txt.text =
            "<size=110%><b><color=#00BFFF>RADAR AIS TÁCTICO</color></b></size>\n" +
            $"<b>Rumbo Mirada:</b> {gazeHeading:000.0}°\n" +
            $"<b>Rumbo BeamPro:</b> {smoothedHeading:000.0}°\n" +
            $"<b>Rotación Gafas (Yaw):</b> {correctedYaw:000.0}°\n" +
            "-----\n" +
            $"<color=#00FF00>{uiTableData}</color>";
    }
}

private void ProcessNMEASentences()
{
    // ---Vaciar la cola procesando todos los mensajes juntos ---
    while (udpMessages.TryDequeue(out string sentenceToProcess))
    {
        if (string.IsNullOrEmpty(sentenceToProcess)) continue;

        AISData data = AISParser.ParseAIVDM(sentenceToProcess);

        if (data != null)
        {
            // Instanciamos el barco si no lo teníamos registrado
            if (!trackedShips.ContainsKey(data.MMSI))
            {
                GeoWaypoint newShip = new GeoWaypoint();

                // Si el primer mensaje es el de posición, inicializamos las coordenadas
                if (data.isPositionReport)
                {
                    newShip.latitude = data.Latitude;
                    newShip.longitude = data.Longitude;
                    newShip.sog = data.SOG;
                    newShip.cog = data.COG;
```

```

    }

    newShip.markerInstance = Instantiate(markerPrefab);
    newShip.uiController =
newShip.markerInstance.GetComponent<ShipMarkerUI>();

    trackedShips.Add(data.MMSI, newShip);
}

// Actualizamos los datos si el barco ya existe (o acaba de ser creado)
if (data.isPositionReport)
{
    trackedShips[data.MMSI].latitude = data.Latitude;
    trackedShips[data.MMSI].longitude = data.Longitude;
    trackedShips[data.MMSI].sog = data.SOG;
    trackedShips[data.MMSI].cog = data.COG;
}

if (data.isStaticReport)
{
    trackedShips[data.MMSI].shipName = data.ShipName;
}
}
}

private double CalculateDistance(double lat1, double lon1, double lat2, double lon2)
{
    double phi1 = lat1 * Math.PI / 180.0; double phi2 = lat2 * Math.PI / 180.0; double
deltaPhi = (lat2 - lat1) * Math.PI / 180.0; double deltaLambda = (lon2 - lon1) * Math.PI /
180.0; double a = Math.Sin(deltaPhi / 2.0) * Math.Sin(deltaPhi / 2.0) + Math.Cos(phi1) *
Math.Cos(phi2) * Math.Sin(deltaLambda / 2.0) * Math.Sin(deltaLambda / 2.0); double c = 2.0 *
Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a)); return R * c;
}

private double CalculateBearing(double lat1, double lon1, double lat2, double lon2)
{
    double phi1 = lat1 * Math.PI / 180.0; double phi2 = lat2 * Math.PI / 180.0; double
deltaLambda = (lon2 - lon1) * Math.PI / 180.0; double y = Math.Sin(deltaLambda) *
Math.Cos(phi2); double x = Math.Cos(phi1) * Math.Sin(phi2) - Math.Sin(phi1) * Math.Cos(phi2) *
Math.Cos(deltaLambda); double theta = Math.Atan2(y, x); return (theta * 180.0 / Math.PI +
360.0) % 360.0;
}

void OnDestroy()
{
    isListening = false;
    if (udpClient != null) udpClient.Close();
    if (listenThread != null && listenThread.IsAlive) listenThread.Abort();
}
}

```


ANEXO XI: CÓDIGO SHIPMARKERUI.CS

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ShipMarkerUI : MonoBehaviour
{
    public TextMeshProUGUI txtName;
    public TextMeshProUGUI txtDistance;
    public TextMeshProUGUI txtMMSI;
    public TextMeshProUGUI txtSOG;
    public TextMeshProUGUI txtCOG;
    public Image imgFlag;

    public void UpdateData(string shipName, double distanceMN, int mmsi, double sog, double
cog)
    {
        // El nombre machaca al MMSI en el título si lo tenemos
        if (txtName != null) txtName.text = string.IsNullOrEmpty(shipName) ? mmsi.ToString() :
shipName;

        if (txtDistance != null) txtDistance.text = $"{distanceMN:F1} MN";
        if (txtMMSI != null) txtMMSI.text = $"{mmsi}";
        if (txtSOG != null) txtSOG.text = $"{sog:F1} kn";
        if (txtCOG != null) txtCOG.text = $"{cog:F0}°";

        // Lógica de banderas
        if (imgFlag != null)
        {
            string mmsiStr = mmsi.ToString();
            if (mmsiStr.Length >= 3)
            {
                string midCode = mmsiStr.Substring(0, 3);
                Sprite flagSprite = Resources.Load<Sprite>("flags/" + midCode);

                if (flagSprite != null)
                {
                    imgFlag.sprite = flagSprite;
                    imgFlag.color = Color.white;
                }
                else
                {
                    imgFlag.color = new Color(1, 1, 1, 0);
                }
            }
        }
    }
}
```