



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Desarrollo de una aplicación de Realidad Aumentada para
representación de datos AIS en tiempo real*

Grado en Ingeniería Mecánica

ALUMNO: Pablo Díaz García
DIRECTORES: Miguel Rodelgo Lacruz
CURSO ACADÉMICO: 2024-2025

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Desarrollo de una aplicación de Realidad Aumentada para
representación de datos AIS en tiempo real*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

El presente trabajo se centra en el desarrollo de una aplicación de realidad aumentada para dispositivos Android, que represente en tiempo real los datos AIS recibidos de embarcaciones cercanas. La aplicación amplía los avances logrados en una fase anterior del proyecto, en la que se alcanzaron resultados funcionales, aunque con ciertas limitaciones. En esta segunda fase, se ha trabajado en la integración de datos AIS reales mediante una antena VHF y un receptor SDR, así como la mejora de la precisión de los marcadores de realidad aumentada.

Para lograr este objetivo, se ha llevado a cabo un estudio detallado del sistema AIS, analizando sus tipos de mensajes, el formato NMEA y las posibilidades de representación de realidad aumentada. La aplicación, desarrollada con Unity y ARCore, recibe los datos del receptor SDR, los decodifica y los integra como marcadores virtuales posicionados según las coordenadas geoespaciales de las embarcaciones.

Los resultados confirman el potencial de esta herramienta para reforzar la seguridad marítima y portuaria, permitiendo una monitorización visual en tiempo real. Además, el proyecto sienta las bases para futuras mejoras y aplicaciones en el ámbito naval, tanto civil como militar.

PALABRAS CLAVE

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

AGRADECIMIENTOS

En primer lugar, quiero expresar mi más sincero agradecimiento a mis compañeros de la promoción 425-155. Han sido los principales protagonistas de unos años inolvidables que me han marcado profundamente, tanto a nivel personal como profesional.

Agradezco especialmente a mi tutor, D. Miguel Rodelgo, por su paciencia, compromiso y dedicación durante todo el cuatrimestre, aclarando siempre mis dudas y guiándome en cada uno de los pasos del trabajo. Sin su guía, habría sido difícil mantener el ritmo que este proyecto requería.

Por último, mi agradecimiento más profundo es para mi familia. Gracias por educarme en valores, por ser siempre un referente y por ofrecerme, sin condiciones, el apoyo que he necesitado a lo largo de estos cinco años. Habéis sido, y seguís siendo, el faro que ilumina cada uno de mis pasos.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	6
1.1 Contexto y motivación	6
1.2 Objetivos	7
1.3 Estructura de la memoria	8
2 Estado del arte	10
2.1 AIS	10
2.1.1 Definición	10
2.1.2 Funcionamiento	11
2.1.3 Clases de AIS.....	12
2.1.4 Campos mensajes AIS	14
2.1.5 Antenas receptoras.....	15
2.1.6 Mensajes NMEA	16
2.1.7 Aplicación de AIS en la Armada	17
2.1.8 Sistemas de monitorización	18
2.1.9 Antecedentes.....	20
2.2 Realidad Aumentada (AR).....	20
2.2.1 Definición	20
2.2.2 Importancia en la actualidad	21
2.2.3 AR en la Armada	22
2.2.4 La Realidad Aumentada Geoespacial	23
2.3 Entorno de desarrollo	24
2.3.1 Unity Engine	24
2.3.2 ARCore	25
3 Desarrollo	26
3.1 Tecnologías y dispositivos empleados	26
3.1.1 Receptor SDR	26
3.1.2 Aplicación AIS-catcher	27
3.1.3 Dispositivo móvil.....	29
3.2 Entorno de desarrollo	30
3.2.1 Configuración de Unity	30

3.2.2 Marcador inicial.....	34
3.3 Primera versión	35
3.4 Recepción de datos AIS	39
3.5 Decodificación de datos AIS.....	39
3.6 Mejoras implementadas	41
3.6.1 Actualización dinámica de los marcadores.....	41
3.6.2 Rediseño visual del marcador	41
3.6.3 Comportamiento de los marcadores en pantalla	42
3.6.4 Simplificación estructural del código	43
3.6.5 Sistema de calibración y orientación del dispositivo.....	44
3.7 Código de <i>VPSManager</i> definitivo	46
4 Pruebas y resultados	52
4.1 Presentación de los resultados.....	52
4.1.1 Prueba empírica	52
4.1.2 Análisis de los resultados.....	57
5 Conclusiones y líneas futuras	63
5.1 Conclusiones y revisión de los objetivos	63
5.2 Líneas futuras	64
6 Bibliografía.....	65
Anexo I: Implicaciones Sociales, y/o Económicas, y/o Ambientales	68
Anexo II: Reflexiones Éticas y Sociales	69
Anexo III: Tipos de mensajes AIS y su contenido	70
Anexo IV Primera versión <i>VPSManager</i>	72
Anexo V <i>VPSManager</i> definitivo	75
Anexo VI: Código <i>AISReceiver</i>	86
Anexo VII: Código <i>VPSManagerUDP</i>	88
Anexo VIII: Código de procesamiento datos AIS.....	91
Anexo IX Código <i>INmeaAisMessageStreamProcessor</i>	96
Anexo X <i>NmeaAISTextFieldParser</i>	98
Anexo XI Vídeo	100

ÍNDICE DE FIGURAS

Figura 1-1 Ejemplo de HUD montado en aeronave para mostrar ayudas al vuelo [4]	7
Figura 2-1 Esquema funcionamiento sistema SOTDMA [10]	12
Figura 2-2 Tabla comparativa clases AIS [7].....	13
Figura 2-3 Mapa global página web <i>VesselFinder</i> con datos AIS [20]	19
Figura 2-4 Captura de la aplicación de AR de <i>Marine Traffic</i> [21]	20
Figura 2-5 Ejemplo de uso de la AR en la enseñanza médica [25]	21
Figura 2-6 HMS combinación de la tecnología HUD con el casco de los pilotos [26]	22
Figura 2-7. Gemelo digital en la Armada [28]	23
Figura 3-1 Antena conectada a la carcasa RTL-SDR con chip R820T2 y RTL2832U (Elaboración propia)	27
Figura 3-2 Captura de pantalla (izquierda) panel principal <i>AIS-catcher</i> (Elaboración propia)	28
Figura 3-3 Captura de pantalla (derecha) panel de mensajes formato NMEA (Elaboración propia)	28
Figura 3-4 Captura de pantalla (izquierda) de configuración de <i>AIS-catcher</i>	29
Figura 3-5 Captura de pantalla (derecha) de configuración de <i>AIS-catcher</i>	29
Figura 3-6 Dispositivo móvil y antena RTL-SDR.	30
Figura 3-7 Escena <i>Unity</i> con los diferentes componentes.....	32
Figura 3-8 Configuración de <i>ARCore Extensions</i>	33
Figura 3-9 Elementos del <i>VPSManager</i> con campos que son parte del <i>script</i>	34
Figura 3-10 Primer diseño de marcador	34
Figura 3-11 Funcionamiento código original (Elaboración propia).....	38
Figura 3-12 Nuevo marcador (Elaboración propia)	41
Figura 3-13 Captura de pantalla en la que se puede observar como el color cambia según la distancia	42
Figura 3-14 Captura que muestra cómo los marcadores se van desvaneciendo a medida que transcurre el tiempo.....	43
Figura 3-15 Texto de instrucciones para calibrar la brújula, con barra que se va completando de izquierda a derecha a medida que se realiza la calibración (Elaboración propia)	44
Figura 3-16 Texto que indica que se ha realizado la calibración correctamente (Elaboración propia)	44
Figura 3-17 Diagrama de flujo código <i>VPSManager</i> (Elaboración propia)	46
Figura 4-1 Primera parte prueba.....	53
Figura 4-2 Segunda parte prueba.....	54
Figura 4-3 Tercera parte prueba	54
Figura 4-4 Cuarta parte prueba.....	55

Figura 4-5 Quinta parte prueba.....	55
Figura 4-6 Sexta parte prueba	56
Figura 4-7 Séptima parte prueba	56
Figura 4-8 Octava parte prueba	57
Figura 4-9 Error de altura observado en los marcadores.....	58
Figura 4-10 Imagen tomada para la medición de error de los marcadores	59
Figura 4-11 Posición de los marcadores parcialmente corregida	60
Figura 4-12 Imagen tomada para la medición de error de los marcadores	61
Figura 4-13 Fallo en la colocación de los marcadores observado en distancias cortas.....	62
Figura 4-14 Aprovisionamiento en la mar	62

ÍNDICE DE TABLAS

Tabla 1 Tipos de barcos AIS [14]	15
Tabla 2 Relación nombres de las lanchas con su MMSI (Elaboración propia)	52
Tabla 3 Tipos de mensajes AIS y sus contenidos [37].....	71

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

Desde sus orígenes la seguridad marítima ha sido y sigue siendo uno de los aspectos más importantes para los navegantes. Esto no sólo implica contar con una visión clara del entorno en el que se navega, conociendo los posibles obstáculos, rutas seguras y las plataformas que se encuentran en los alrededores, si no también llevar a cabo una vigilancia y supervisión marítima tanto conjunta como individual por parte de cada nación de este mismo entorno.

Para garantizar esta seguridad es fundamental tener un control adecuado de las principales rutas comerciales, así como de los barcos que las transitan. Cada año más de 200.000 buques navegan exclusivamente por las líneas marítimas españolas, su seguridad, por tanto, es vital para la economía nacional e internacional. De esta manera, se incrementa la capacidad de anticipar actividades ilícitas o, en caso de navegación temeraria o peligrosa se reduce el riesgo de posibles abordajes. El uso de las tecnologías en las actividades cotidianas y trabajos es cada día más frecuente, y el sector marítimo no es excepción. La tecnología facilita el trabajo de garantizar una navegación segura siempre y cuando se mantengan los procedimientos tradicionales y no se dependa excesivamente de ella. Un ejemplo clave es el sistema AIS (*Automatic Identification System*), utilizado para transmitir mensajes de posicionamiento e información identificativa de buques. Este sistema se ha vuelto fundamental para la seguridad marítima, hasta el punto de que la OMI (Organización Marítima Internacional) lo ha establecido como obligatorio para todos los buques que operen bajo la normativa SOLAS (*Safety of Life at Sea*) [1].

La importancia del AIS en la actualidad es tal que se utiliza como fuente principal en sistemas de supervisión tan cruciales como el COVAM (Centro de Operaciones y Vigilancia de Acción Marítima), que se encarga de generar una RMP (*Recognised Maritime Picture*) de los espacios marítimos de interés nacional, proporcionando información crucial para los buques de la Armada [2]. Asimismo, el AIS constituye una fuente de información fundamental para aplicaciones relevantes para la navegación como *Marine Traffic* que basa su funcionamiento en este sistema, recolectando continuamente esta información y plasmándola en un mapa global. Esto permite una clasificación más precisa de las plataformas que se encuentran en navegación [3].

La motivación principal de este proyecto es aplicar las tecnologías AR (Realidad Aumentada) para representar los datos obtenidos del AIS, actualmente representados en 2-D en mapas virtuales, y llevarlo a un nivel superior al superponer los datos que nos proporcione el sistema con el mundo real. Con esta mejora se pretende obtener una representación que ayude a los usuarios a tener un mayor conocimiento de los contactos que sean capaces de ver a simple vista, o bien desde el barco o desde algún puesto de vigilancia de tráfico portuario. Al presentar datos AIS superpuestos con los contactos del mundo real,

mediante pantallas o incluso con algún tipo de HUD (*Heads Up Display*) (véase Figura 1-1), identificar barcos o peligros a la navegación se haría más sencillo, sobre todo en horas sin luz o días de mala visibilidad cuando no se pueden distinguir los blancos con claridad. Además, jugaría un papel importante en la seguridad marítima al mostrar la distancia a la que se encuentran los contactos, información relevante para algunas maniobras o la entrada y salida de embarcaciones de un puerto; y la velocidad de las mismas permitiendo controlar por ejemplo si se cumplen o no las normas de seguridad portuaria. Buscando acercar las nuevas tecnologías a la Armada e intentar darle un impulso a estas mismas tratando de modernizar los medios y también incluso, a poder ser, facilitar dentro de lo posible el trabajo diario en los barcos y otras unidades.



Figura 1-1 Ejemplo de HUD montado en aeronave para mostrar ayudas al vuelo [4]

Este proyecto continúa los pasos establecidos por el trabajo iniciado por el AF Ángel Gamboa [5] donde se sentaron las bases del proyecto. La aplicación, desarrollada para un teléfono móvil, lograba mostrar marcadores en forma de realidad aumentada; sin embargo, estos no presentaban datos reales sino datos ficticios preconfigurados. Estos marcadores que aparecían en la aplicación no llegaban a mantenerse en la posición exacta que se le había asignado y no se midió la precisión de la representación. Partiendo de los principios establecidos en el anterior, este proyecto busca mejorar la representación y precisión de la aplicación y, principalmente, integrar los datos recibidos de una fuente externa para mostrarlos en tiempo real. En particular se empleará un receptor SDR [6] conectado directamente al teléfono móvil.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado (TFG) es el desarrollo de una aplicación de AR que muestre en tiempo real mediante marcadores, los datos AIS esenciales de un contacto de forma que esta información sea más comprensible y pueda asociarse con mayor facilidad a cada contacto visual en el entorno real.

Para lograr este objetivo principal se plantean los siguientes objetivos específicos:

- Realizar un estudio del estado del arte para asentar conceptos cruciales para el desarrollo de la aplicación y del entorno en el que va a operar.
- Familiarizarse con las tecnologías involucradas en el trabajo, logrando un entendimiento profundo de estas.

- Integrar los datos recibidos a través de una antena y su receptor SDR en la aplicación, logrando una representación de los mismos en tiempo real.
- Corregir los errores de funcionamiento de la aplicación anterior, buscando conseguir una mejor representación de los datos y una aplicación más robusta.
- Evaluar el rendimiento de la aplicación durante su desarrollo, observando su funcionamiento y limitaciones en las pruebas realizadas.
- Mostrar el máximo de información posible de las embarcaciones, rumbo, velocidad, distancia, y lograr distinguir los marcadores por su nombre o en su defecto por el MMSI.

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, así como 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 se engloba el trabajo con la revisión del estado del arte respecto a:

- Los AIS, su importancia en el mundo marítimo y su seguridad, así como una explicación de los datos proporcionados y las diversas aplicaciones.
- La Realidad Aumentada, y las distintas herramientas utilizadas para el proyecto, como *ARCore* o *Unity*.

En el Capítulo 3 se detalla el desarrollo del TFG. Se justifica el uso del receptor SDR, y su *driver*, así como de la antena VHF, y las herramientas *Unity* y *ARCore*. Además, se presentan los tipos de datos AIS que se han buscado obtener, y el proceso que se ha seguido para lograr integrarlos en la aplicación. Se explica también su funcionamiento y se describe el enfoque adoptado para lograr la representación en tiempo real. Finalmente, se explican las mejoras implementadas para optimizar la aplicación y se detallan los experimentos realizados.

En el Capítulo 4, se muestran los resultados obtenidos y se procede a su análisis. Se exponen las prestaciones de la aplicación alcanzadas explicando la información mostrada, y se discuten sus limitaciones.

En el Capítulo 5 se habla de la conclusión 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 describe en qué consiste la realidad aumentada, cómo se ha desarrollado su tecnología y la aplicación de esta en la Armada, seguido de un estudio de los AIS y su aplicación para la navegación tanto en el mundo civil como en la Armada. Por último, se concluirá con una explicación del entorno *software* que se ha utilizado para el desarrollo de este proyecto.

2.1 AIS

2.1.1 Definición

El Sistema de Identificación Automática es el avance más importante para la seguridad marítima desde la invención del radar. Este nació como un sistema de ayuda para que los barcos comerciales fuesen capaces de saber la localización de otros buques en cualquier condición de visibilidad para evitar el abordaje, y para darle más información al timonel de su entorno marítimo [7].

El sistema AIS logra esto mediante el envío constante de información relativa a las embarcaciones, como su localización, velocidad o rumbo, entre otros. Esta información es recibida por todos los receptores AIS que se encuentren dentro del alcance adecuado para captar la señal. Cuando se emplea de manera combinada con las autoridades costeras se puede proporcionar mayor seguridad en la navegación, así como un control efectivo del tráfico marítimo con el objetivo de reducir los peligros asociados. El impulso de los sistemas AIS en la seguridad de la navegación marítima fue tal que, en 2002, se hizo obligatorio su uso para todo tipo de buques de pasajeros y de más de 300 toneladas. Así lo recoge el Convenio SOLAS de 1974, enmendado, en su capítulo V, ampliando la obligatoriedad de su uso para otros tipos y tamaño de buques en el 2004 [8].

Actualmente los requerimientos recogidos en SOLAS no exigen que los barcos dispongan de una pantalla o interfaz donde se muestren los datos AIS recibidos. En cambio, solo se requiere una pantalla mínima y un teclado que permitan visualizar la información relevante del AIS a los encargados de la navegación del buque.

Hoy en día además de las funciones que tiene AIS a bordo de los barcos también se ofrece la posibilidad de enviar mensajes de seguridad marítima, y avisos a la navegación, desde las llamadas estaciones AIS AtoN, o estaciones base. Estas estaciones son definidas por la Asociación Internacional de Ayudas a la Navegación Marítima y Autoridades de Faros (IALA) como dispositivos o sistemas externos a los buques diseñados para mejorar la seguridad y eficacia de la navegación y el tráfico. Las estaciones AtoN son dispositivos flotantes que envían la posición de las ayudas a la navegación, como puede ser una boya de un puerto o de peligro incluso antes de lo que sería capaz de detectarla un barco mediante radar. Esto permite a los barcos realizar maniobras de evasión en caso de tener una visibilidad reducida, brindando un tiempo de reacción mayor e incluso saber la posición en la que se encuentran.

2.1.2 Funcionamiento

AIS funciona mediante transpondedores que determinan su propia posición, velocidad y rumbo utilizando un receptor de GPS. La información que recoge el transpondedor se combina con otros datos relevantes para la navegación y se transmite a los transpondedores AIS cercanos sin la necesidad de intervención de una tercera parte o usuario.

Los transpondedores instalados en las embarcaciones reciben la información proveniente de otras embarcaciones cercanas y la utilizan para crear una representación gráfica del tráfico en el área. Este mismo transpondedor puede estar conectado a un PC o un *chart plotter*, dispositivos que muestran una carta digital junto con la información de la situación GPS. De esta manera el usuario recibe información sobre la situación marítima como si de un radar se tratara, visualizando los contactos y el entorno en el que está navegando. Para el funcionamiento del AIS no es necesario el uso de un radar, sin embargo, sí que es posible combinar ambos sistemas, lo que ayuda a verificar e identificar los contactos radar cercanos de manera mucho más efectiva. Una ventaja clara del AIS sobre el radar es que opera en la banda de frecuencias VHF, lo que permite que cualquier obstáculo geográfico, como puede ser una isla, sea sorteado y la señal llegue sin mayores inconvenientes.

Debido a que opera en la frecuencia VHF, el AIS tiene la capacidad de gestionar correctamente hasta 4500 reportes por minuto y actualizar la información en cuestión de segundos. La razón de que esta gestión de la información sea posible es el sistema SOTDMA (Acceso Múltiple por División del Tiempo Autoorganizado). Este sistema es el encargado de garantizar que el flujo de información entre las estaciones y entre barcos, con estaciones costeras y cualquier otro dispositivo, sea continuo y fiable, permitiendo de esta manera una rápida y fácil identificación de las estaciones en el entorno.

El funcionamiento del sistema reside en la asignación de intervalos de tiempo, exactamente de 26,6 milisegundos, durante los cuales los transpondedores deben transmitir toda la información que han recabado mientras su ventana de transmisión estuvo cerrada. De este modo, cada buque tiene asignado un intervalo de tiempo específico para poder transmitir, mientras que el resto del tiempo permanece en escucha para recibir los mensajes de los buques que se encuentran cerca. El sistema SOTDMA (véase la Figura 2-1) está diseñado para evitar las interferencias entre los intervalos de transmisión. Para lograr esto, si hay una interferencia de datos entre dos buques que quieren transmitir a la vez, el sistema le asignará a uno de ellos un intervalo aleatorio. Si se produjese, en el poco probable caso de que todos los intervalos esten ocupados, el sistema aplicaría la norma de “reutilización de slots”, refiriéndose a los intervalos anteriormente mencionados como slots. Esta norma consiste en asignarle al buque un intervalo de otro lejano, de esta forma se reduce la posibilidad de interferencia, lo que prioriza los buques que se encuentran a una menor distancia entre ellos para minimizar el riesgo de abordaje [9].

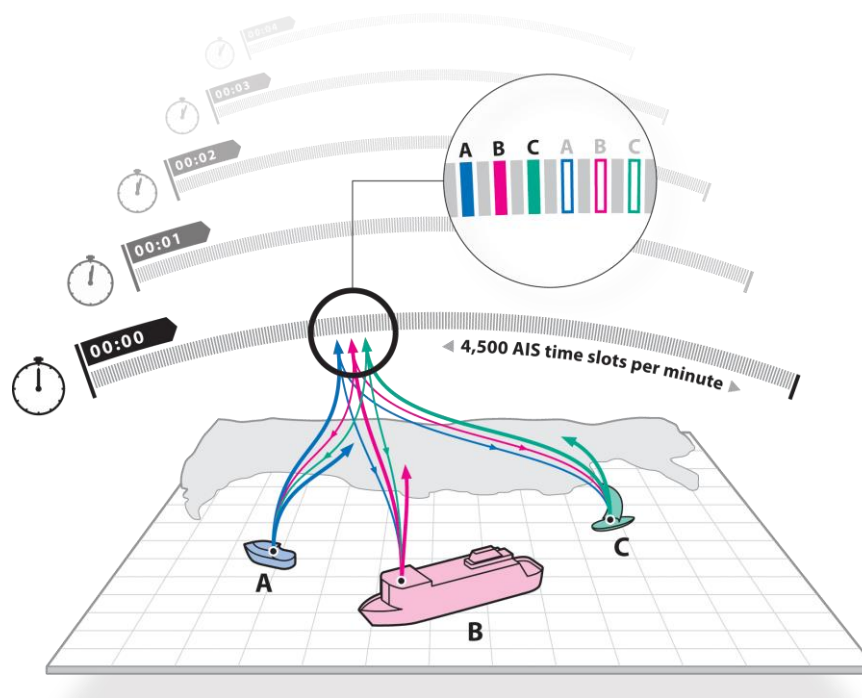


Figura 2-1 Esquema funcionamiento sistema SOTDMA [10]

2.1.3 Clases de AIS

El AIS dispone actualmente de tres clases de transpondedores; clase A, clase B y solo recepción (Figura 2-2). La clase A, obligatoria para una gran cantidad de buques de gran tonelaje, comerciales y todos los demás incluidos en el criterio SOLAS, fue la primera implementada. Estos transpondedores son los más complejos y costosos, y transmiten más información y con mayor frecuencia. Su instalación es la más compleja de todas, ya que cuentan con una antena GPS propia, además de un *display* independiente, así como salidas para los sistemas de visualización radar y de cartografía digital.

Los transpondedores de clase B, por otro lado, fueron diseñados para barcos recreativos, de pesca o embarcaciones más pequeñas que no están obligados a llevar un transpondedor de clase A. Aunque muchas de las capacidades de la clase B son similares a las que ofrece la clase A, presenta un precio más razonable y una instalación menos compleja. Muchos de estos sistemas son simplemente unidades que transmiten la información recibida directamente al *chart plotter*, radar e instrumentos conectados. La principal desventaja frente a los sistemas más avanzados es que tienen un alcance más limitado y las actualizaciones de la información de los barcos cercanos se realizan en intervalos de tiempo mayores [11].

Por último, están los transpondedores de solo recepción, que no emiten ningún tipo de información. Su funcionamiento es únicamente recibir datos de otros transpondedores de clase A o clase B y transmitirlos al instrumento conectado. Esta clase se usa principalmente en embarcaciones recreativas de pequeña envergadura o aeronaves de rescate.

AIS FEATURES AT A GLANCE	CLASS A	CLASS B	RECEIVER ONLY
Transmit own position and vessel data	✓	✓	✗
Receive position from Class A vessels	✓	✓	✓
Receive position from Class B vessels	✓	✓	✓
Receive name & call signal from Class A vessels	✓	✓	✓
Receive name & call signal from Class B vessels	✓*	✓	✓*
Includes certified GPS receiver	✓	✓	✗
Transmit "Safety related message"	✓	✓	✗
Meets international AIS standards	✓	✓	✗
Can be fitted to SOLAS vessels	✓	✗	✗
Can be fitted to non-SOLAS vessels	✓	✓	✓
Two parallel receivers for complete AIS reception	✓	✓	✗**
DSC receiver for automatic international channel control	✓	✓	✗

* Some Class A transponders and receive only devices may require a software update to receive the latest Class B vessel details messages.
** All Comar AIS receiver products have 2 parallel channels and can receive messages on both channels simultaneously.

Figura 2-2 Tabla comparativa clases AIS [7]

Existen, además, otros tipos de dispositivos AIS que no van instalados dentro de los barcos, pero que forman parte del ecosistema AIS. En esta categoría se incluyen los terminales AtoN mencionados anteriormente, instalados en boyas y en tierra, que transmiten información recabada sobre el tráfico, la meteorología o el estado del mar. También se encuentran las *Base Station* (estaciones costeras AIS), que apoyan a las estaciones costeras de control de tráfico y contribuyen al aumento de la seguridad marítima. En cuanto a la seguridad marítima existen los dispositivos SART (*Search And Rescue Transceiver*), que se utilizan para enviar un mensaje de socorro en situaciones de emergencia. Estos dispositivos emplean un sistema SOTDMA modificado, el PATDMA, que transmite datos identificativos, ocho veces por minuto en intervalos asignados y puede ser recibido por cualquier receptor AIS [9].

Como ya se ha observado, el AIS puede adaptarse a diferentes entornos y propósitos, lo que le da un uso específico en cada caso. Sin embargo, cada una de estas clases de AIS y tipos de dispositivos utilizan un sistema de transmisión distinto. Anteriormente se mencionó el sistema de transmisión SOTDMA, diseñado para los transpondedores de la clase A, pero existen otros cinco tipos de sistema de transmisión TDMA. Cada uno de ellos tiene sus características y funciones y son los siguientes [12]:

- **CSTDMA:** Es el sistema utilizado por los transpondedores de clase B. Este sistema permite por lo tanto que el transmisor sea de un costo reducido, pero sigue siendo completamente operativo con sistemas SOTDMA, otorgando siempre prioridad a estos últimos. Al igual que el sistema SOTDMA, el sistema CSTDMA hace uso de intervalos de tiempo asignados para cada estación AIS. Sin embargo, en este caso, para usar uno de estos "slots", el sistema CSTDMA se basa en el nivel de ruido de fondo. Asume un ruido constante en los intervalos y solo ocupa aquellos en los que el nivel de ruido no excede el umbral establecido. De esta forma, se asegura que el intervalo no está en uso y les cede la prioridad a los mensajes de clase A.
- **FATDMA:** Este es el sistema de transmisión utilizado por las estaciones AIS encargadas de enviar mensajes de ayuda a la navegación. Están diseñados de tal manera que los "slots" utilizados se seleccionan manualmente y se emite una señal al resto de los sistemas que bloquean los "slots" reservados para los sistemas FATDMA. De esta forma las estaciones AIS responsables de enviar los mensajes de ayuda a la navegación se aseguran siempre de que estos se transmitan de forma inmediata.
- **RATDMA:** Este sistema se utiliza como apoyo a los AIS de clase A. Su funcionamiento se basa en el uso de un mapa interno de los "slots" disponibles, con el objetivo de seleccionar

uno libre para enviar un único mensaje. La razón de que solo se envíe un mensaje, se debe a que su propósito se limita a acceder a la red de “slots” para posteriormente utilizar el sistema SOTDMA. No genera una señal de aviso de uso del “slot”. Por lo tanto, si se enviaran más de un mensaje de este tipo, podría producirse un colapso de los datos que comprometería el sistema.

- **PATDMA:** El presente sistema como se explicó previamente es el utilizado en los dispositivos SART. Su funcionamiento puede provocar colisiones de datos, un problema que se asume por la misión de emergencia que los dispositivos SART desempeñan. Sin embargo, no constituye un problema significativo, ya que los dispositivos no suelen operar frecuentemente en zonas de mucha densidad de tráfico.
- **ITDMA:** Al igual que el sistema RATDMA, sirve como apoyo a los sistemas SOTDMA. En este caso en lugar de enviar un mensaje para acceder a una red de slots, se utiliza para notificar de un cambio en el intervalo de tiempo entre los mensajes enviados o para indicar un periodo próximo de silencio. A diferencia del RATDMA, el mensaje sí que es anunciado antes de su envío, lo que elimina el riesgo de colisiones en caso de que se envíen múltiples mensajes.

2.1.4 Campos mensajes AIS

Existen varios tipos de mensajes AIS con sus correspondientes campos. En el presente proyecto, se abordarán únicamente algunos de ellos; no obstante, es necesario conocer, aunque de manera general, la clasificación que se asigna a los mensajes según la información que transmiten. A continuación, se detallan los diferentes campos existentes en los mensajes [13]:

1. **MMSI** (*Maritime Mobile Service Identity*): Serie de nueve dígitos transmitidos utilizados para identificar de forma única, las embarcaciones, estaciones costeras, estaciones terrenas costeras o llamadas grupales.
2. **IMO number** (*International Maritime Organization*): Identificador único asignado a buques o compañías registradas en la gestión de buques. El número “IMO” de los buques consta de siete dígitos.
3. **Callsign**: Distintivo de llamada que puede ser adoptado formalmente por una organización gubernamental o de manera informal por individuos y organizaciones. Generalmente se codifica para enmascarar la identidad de la estación.
4. **Name**: Nombre original del buque que transmite la señal.
5. **SOG** (*Speed Over Ground*): Velocidad absoluta de la embarcación, determinada mediante la medición de la velocidad respecto al fondo marino, de modo que las condiciones del mar no interfieran en la medición.
6. **COG** (*Course Over Ground*): Similar al SOG, indica el rumbo absoluto del buque, medido también respecto al fondo marino.
7. **Navstate**: El *navstate* refleja el estado de navegación del buque, representado por números, cada uno de los cuales tiene un significado específico. Este campo está compuesto por dieciséis números, que varían del 0 al 15.
8. **Country**: Indica la bandera del país al que pertenece administrativamente el buque, aunque no necesariamente coincida con la nacionalidad del mismo.
9. **Destination**: Define el puerto al que el buque tiene que arribar en su próxima recalada.
10. **Shiptype**: Este campo define el tipo de barco emisor de los datos AIS, son números del 10 al 99, cuyos significados se detallan en la Tabla 1.
11. **Draught**: Muestra el calado de la embarcación en metros.
12. **Dimensions**, a, b, c.: Se trata de cuatro campos independientes que muestran las dimensiones del buque en metros, así como la posición de la antena transmisora.
13. **Longitude**: Muestra la longitud geográfica en la que se encuentra el buque. Se presenta en grados, con decimales y con el correspondiente símbolo, (+) si está al este o (-) si está al oeste.

14. **Latitude:** Indica la longitud geográfica en la que se encuentra el buque. Se presenta en grados, con decimales y con el correspondiente símbolo, (+) si está al norte o (-) si está al sur.
15. **ETA (Estimated Time of Arrival):** Este campo proporciona la hora y la fecha estimadas de llegada del buque.

Código	Tipo de barco	Código	Tipo de barco
10-19	<i>Reserved</i>	55	<i>Law Enforce</i>
20-28	<i>Wind in Ground</i>	56	<i>Local Vessel</i>
29	<i>SAR Aircraft</i>	57	<i>Local Vessel</i>
30	<i>Fishing Tug</i>	58	<i>Medical Trans</i>
31	<i>Tug</i>	59	<i>Special Craft</i>
32	<i>Tug</i>	60-69	<i>Passenger</i>
33	<i>Dredger</i>	70	<i>Cargo</i>
34	<i>Dive Vessel</i>	71	<i>Cargo- Hazard A (Major)</i>
35	<i>Military Ops</i>	72	<i>Cargo- Hazard B</i>
36	<i>Sailing Vessel</i>	73	<i>Cargo- Hazard C (Minor)</i>
37	<i>Pleasure Craft</i>	74	<i>Cargo- Hazard D (Recognizable)</i>
38	<i>Reserved</i>	75-79	<i>Cargo</i>
39	<i>Reserved</i>	80	<i>Tanker</i>
40-49	<i>High Speed Craft</i>	81	<i>Tanker – Hazard A (Major)</i>
50	<i>Pilot Vessel</i>	82	<i>Tanker – Hazard B</i>
51	<i>SAR</i>	83	<i>Tanker – Hazard C (Minor)</i>
52	<i>Tug</i>	84	<i>Tanker – Hazard D (Recognizable)</i>
53	<i>Port Tender</i>	85-89	<i>Tanker</i>
54	<i>Anti-Pollution</i>	90-99	<i>Other</i>

Tabla 1 Tipos de barcos AIS [14]

Como se puede observar, los AIS proporcionan una gran cantidad de datos con el objetivo de ofrecer la máxima información posible sobre una embarcación. En el presente proyecto, únicamente se utilizarán los campos que se consideran más relevantes para la identificación de un buque, resaltados en negrita en el listado anterior.

2.1.5 Antenas receptoras

El sistema AIS, como ya se ha comentado previamente, es fundamental para la seguridad marítima, permitiendo el intercambio de información entre embarcaciones y estaciones costeras. Para recibir estos datos, se utilizan principalmente antenas VHF, optimizadas para las frecuencias de 161.975 MHz y 162.025 MHz, que emplean los mensajes AIS. Estas antenas están diseñadas para ofrecer una alta

ganancia en estas frecuencias, mejorando la recepción de las señales y reduciendo interferencias. Algunas embarcaciones también emplean antenas combinadas VHF/GPS, integrando la recepción de datos AIS y señales de posicionamiento global.

En los últimos años, los receptores SDR (Radio Definidas por software) han ganado popularidad en aplicaciones de recepción de datos AIS. Estos receptores permiten una mayor flexibilidad y adaptabilidad en la recepción de señales, ya que el procesamiento de estas se realiza mediante software, facilitando la actualización y personalización de las funciones de recepción. Esta tecnología permite a los usuarios ajustar parámetros y modos de operación, utilizando el mismo dispositivo para distintas aplicaciones.

Los receptores RTL-SDR son una versión asequible y compacta de los receptores SDR, basados en tecnología USB de TV digital (DVB-T). Estas antenas permiten la recepción de señales en un amplio rango de frecuencias, desde 100 kHz hasta 1,7 GHz y transforman las señales analógicas en digitales mediante un ADC (Convertidor de Analógico a Digital). Utilizan un muestreo en cuadratura I/Q, lo que optimiza el procesamiento de las señales y mejora la precisión de la recepción. Su versatilidad, bajo costo y capacidad para ser configuradas mediante software las hace ideales para una amplia variedad de aplicaciones. En este proyecto, se especificará con más detalle el tipo exacto de antena RTL-SDR [15] utilizada, proporcionando información sobre su funcionamiento y especificaciones en el capítulo tres.

2.1.6 Mensajes NMEA

Para la realización de este proyecto, se emplea una antena con un receptor de datos SDR, cuyos datos son reenviados por una aplicación en el teléfono que actúa como un *driver* del dispositivo. En todo momento, la información AIS se transmite utilizando el formato NMEA para los mensajes. Lo cual es un aspecto de gran importancia a tener en cuenta para su posterior decodificación.

En este proyecto, se centra la atención en el estándar NMEA 0183, reconocido como una versión ampliamente utilizada para la comunicación electrónica marítima. Aunque existe una versión más reciente NMEA 2000, esta ha sido adoptada principalmente en embarcaciones recreativas de pequeño tamaño. NMEA 0183 permite la transmisión de datos entre dispositivos electrónicos marinos, como receptores GPS, ecosondas y AIS. Los datos se envían en pequeñas unidades denominadas “sentencias”, cada una de las cuales no debe exceder los ochenta caracteres. La NMEA (*National Marine Electronics Association*) es la entidad responsable de definir y mantener estos estándares. Su misión principal es promover la estandarización y el desarrollo técnico en la industria de la electrónica marítima, asegurando que los dispositivos de diferentes fabricantes puedan comunicarse eficazmente entre sí. La adopción de estos estándares ha sido fundamental para garantizar la compatibilidad y la comunicación eficiente entre los diversos sistemas electrónicos presentes en las embarcaciones, mejorando la seguridad y la eficacia de las operaciones marítimas.

Los dispositivos encargados de emitir la información se denominan “*Talkers*”. Ejemplos de estos son los receptores GPS o los sondadores, dispositivos de los cuales el AIS obtiene los datos que luego transmite. Por otro lado, los equipos electrónicos que reciben esta información, conocidos como “*Listeners*”, incluyen dispositivos como los *chart plotters* o los radares. La relación de comunicación entre los dispositivos que operan bajo el formato NMEA sigue siempre el patrón de un único “*Talker*” y varios “*Listeners*”. La velocidad de transmisión de datos en este formato es de 4800 baudios (bits por segundo). Dentro del formato NMEA la versión NMEA 2000 a diferencia de sus predecesores incluye un *checksum* obligatorio en el mensaje.

Todos los mensajes NMEA se manda en formato de texto que sigue un formato “ASCII”, el cual se explica a continuación[16]:

- La frase siempre empieza o bien con el símbolo \$ o con ;
- Un código de dos dígitos “yy” continua con el mensaje informando el tipo de instrumento.
- A continuación, sigue un código de tres dígitos “xxx”, informando el tipo de datos enviados.

- Sigue una coma y los contenidos que se quieren enviar, esto depende de los datos que se quieran enviar y su valor.
- La última parte del mensaje sería el checksum.

Un mensaje con formato NMEA general sería:

\$SDDBT, Water Depth, feet, Water depth, Meters, Water Depth, Fathoms*Checksum

\$SDDBT,556.3, f,169.6, M,92.7, F*37

Siendo f, *feets*; M, metros; y F, *Fathoms*.

Si la sentencia NMEA es usada para AIS, el mensaje queda de la siguiente manera:

!AIVDM,1,1,,A,177l?m9000`Pk`<i`kh0ISd00R;,0*30 (Datos de otras embarcaciones)

!AIVDO,1,1,,A,13P6u?0P7c3E2fP4000000uF001Q00000,0*74 (Datos de la propia embarcación)

Siendo:

- !AIVDO / !AIVDM identificadores de sentencia AIS de la propia embarcación (*Vessel Data Output*) o de otras embarcaciones respectivamente.
- El primer 1 es un indicador del número total de fragmentos en los que se divide el mensaje completo.
- El segundo 1 indica que es el primer fragmento del mensaje.
- El campo vacío, se utiliza para identificar secuencialmente los mensajes que se envían en múltiples fragmentos. En caso de ser único, este campo puede estar vacío.
- La letra A especifica el canal de radio utilizado para la transmisión, en este caso el canal A
- 177l?m9000`Pk`<i`kh0ISd00R constituye la carga útil que contiene los datos AIS codificados en formato binario de 6 bits.
- 0*30 La última parte del mensaje, muestra el valor del *checksum* o suma de verificación.

2.1.7 Aplicación de AIS en la Armada

La Armada, siguiendo el cauce que marca la tecnología, no se queda atrás en la adopción de estos sistemas para mejorar la seguridad en la navegación. Todos los buques de la Armada están equipados con AIS, tal como lo establece el convenio SOLAS y hacen uso de esta tecnología para garantizar una navegación con conocimiento del entorno que les rodea. Pero este no es el único uso que se hace de estos sistemas dentro de la Armada.

Existen dos instituciones militares dentro de la Armada que participan junto al resto de organismos en organizar y controlar el tráfico marítimo de nuestras aguas territoriales y de la ZEE (Zona Económica Exclusiva). Como misión aparte, las instituciones militares se encargan de dar seguridad y controlar las unidades militares desplegadas mediante apoyo directo a sus misiones en aguas de interés nacional. Las dos instituciones militares encargadas de realizar este papel son el ya comentado anteriormente COVAM y el CMVSM.

El COVAM es un instrumento del que se vale la FAM (Fuerza de Acción Marítima) que obtiene información de varias fuentes, entre ellas AIS, para obtener una imagen precisa de lo que sucede en las aguas marítimas de interés. Con la información obtenida el COVAM realiza una fusión de datos y análisis de estos para crear la imagen más precisa y actualizada posible. Imagen que pone a disposición a todos los buques de la flota en apoyo a las misiones en las que se encuentren. Aunque este proyecto se centrarán más en la parte naval de la Armada, el COVAM también apoya a otros organismos nacionales encargados de otro tipo de misiones en ámbito marítimo, como Salvamento Marítimo (SEMAR) de la Guardia Civil, la Sociedad Estatal de Salvamento Marítimo (SASEMAR) o la Secretaría General de Pesca. Este centro de operaciones se encuentra ubicado en Cartagena y mantiene una vigilancia constante

las veinticuatro horas del día, ofreciendo una respuesta ante cualquier incidente inmediato y a cualquier hora del día, buscando una cobertura perfecta para las unidades que lo necesiten. Aunque reciben los datos de varias fuentes de información, la más importante y las que se utiliza con prioridad siguen siendo el AIS por la obligatoriedad de transmitir estos datos según lo establecido en el convenio SOLAS.

Por su parte, el CMSVM (Centro de Mando de Seguridad y Vigilancia Marítima) [17], integrado dentro de la estructura operativa de la Armada, es el encargado del planeamiento, conducción y seguimiento de las operaciones de vigilancia y seguridad en los espacios marítimos de soberanía, responsabilidad e interés nacional. Este órgano, para cumplir con su cometido, colabora con distintas entidades tanto del mundo militar como del mundo civil, entidades que incluyen todos los servicios de costas que operan con datos AIS para realizar su seguimiento y control y mandar los mensajes correspondientes de cada estación.

2.1.8 Sistemas de monitorización

Los sistemas AIS, han demostrado su eficacia en el diseño de sistemas que permiten, a las diversas entidades involucradas, llevar a cabo un control más preciso del tráfico marítimo. Esto ha contribuido a mejorar la seguridad en el ámbito marítimo al posibilitar una monitorización más detallada. Actualmente, y desde hace ya varios años, estos sistemas están disponibles en plataformas digitales, algunas de las cuales ofrecen, incluso de forma gratuita, mapas virtuales globales con la información AIS de millones de embarcaciones. Esta accesibilidad ha transformado en gran medida la forma en la que se navega, eliminando la incertidumbre durante la travesía. Hoy en día resulta muy extraño encontrarse con un contacto u obstáculo no conocido, condición que permite una navegación más segura y confiada al contar con un conocimiento más amplio del entorno que se transita.

Tradicionalmente, la responsabilidad del rumbo y velocidad de un buque recaía en el Comandante de la embarcación, que utilizaba señales visuales con banderas como señal que se estaba acercando a un puerto. Con la evolución de la radio comenzó a incrementarse la dependencia de esta tecnología por las facilidades en la comunicación que ofrecía. Posteriormente, durante la Segunda Guerra Mundial se introdujo el radar en el mundo marítimo y la combinación de radio y radar dio lugar al primer VTS. VTS son los *Vessel Traffic Systems*, sistemas ubicados en la costa cuyo objetivo principal es proporcionar información y advertencias para la navegación, así como gestionar el tráfico marítimo, controlando la entrada y salida de embarcaciones en puertos a lo largo de vías marítimas específicas.

Fue tal el avance de los AIS, que la IMO (*International Maritime Organization*) los reconoció como cruciales para la seguridad en la navegación en 1968. Esto impulsó su desarrollo, y fue en 1985 cuando se reconocen los VTS como sistema necesario para la entrada o salida de un puerto, llegando al punto, en el que, establecer un VTS se volvió un sinónimo de la necesidad de pilotaje para las maniobras, y de zona de alta densidad de tráfico y peligrosidad para la navegación. Es en 1997 cuando se recogen estos sistemas en el convenio SOLAS, volviendo obligatorio el uso de los VTS para los gobiernos a los que pertenecieran ciertas costas. Algunas de las condiciones establecidas para que sea obligatorio la presencia de un VTS son, que la zona se encuentre dentro de las aguas territoriales de un país y que exista una alta densidad de tráfico. Además, la implementación de un VTS debe garantizar que no se infrinja ninguna normativa en su utilización. Para cumplir con esta tarea, estos tipos de sistemas se valen de herramientas como los AIS que desempeñan un papel fundamental en el funcionamiento de estos sistemas [18].

En el mundo marítimo, las páginas web como *Marine Traffic* o *VesselFinder* son herramientas de gran utilidad, ya que ofrecen una representación visual clara del tráfico marítimo, lo que permite a los usuarios conocer la cantidad de tráfico en su ruta o identificar contactos que podrían ser difíciles de distinguir solo a simple vista. Aunque existen otros competidores en este mercado, como son *SeaRM360* o *FleetMon* estos no han logrado el mismo nivel de popularidad ni tienen el impacto de las plataformas mencionadas. Cada una de estas aplicaciones emplea su propio software para generar mapas interactivos, pero los datos que se utilizan para una visión global del tráfico marítimo provienen principalmente de los AIS, mostrando no solo la localización de los barcos, si no también sus nombres, velocidad y datos

de sus rutas, tales como destinos y puntos de salida. *VesselFinder* (véase la Figura 2-3) por ejemplo, lleva más allá su utilidad realizando una distinción incluso de colores según el tipo de barco que está transmitiendo en AIS, sabiendo de un solo vistazo la densidad de tráfico de un tipo específico de buque.

No es sorprendente que los datos AIS sean fundamentales en proyectos relacionados con la identificación de buques o con la seguridad marítima. Un ejemplo representativo de esta afirmación es el reciente proyecto SIRENA [19] llevado a cabo por el CUD (Centro Universitario de la Defensa). Este proyecto surgió, de la mano de la Armada, con el objetivo de aplicar técnicas basadas en IA (Inteligencia Artificial) para la mejora del CEM (Conocimiento del Entorno Marítimo). El proyecto se centra en un análisis en tiempo real de los datos proporcionados por los AIS. Al tratarse de un análisis en tiempo real existe cierta limitación en cuanto a la cantidad de datos que pueden ser analizados. El propósito de este análisis es asistir al COVAM en su misión de monitorear y garantizar la seguridad en el entorno marítimo. SIRENA ya ha sido probada con éxito demostrando su capacidad para realizar análisis precisos en tiempo real e incluso detectar escenarios anómalos que podrían comprometer la seguridad marítima.

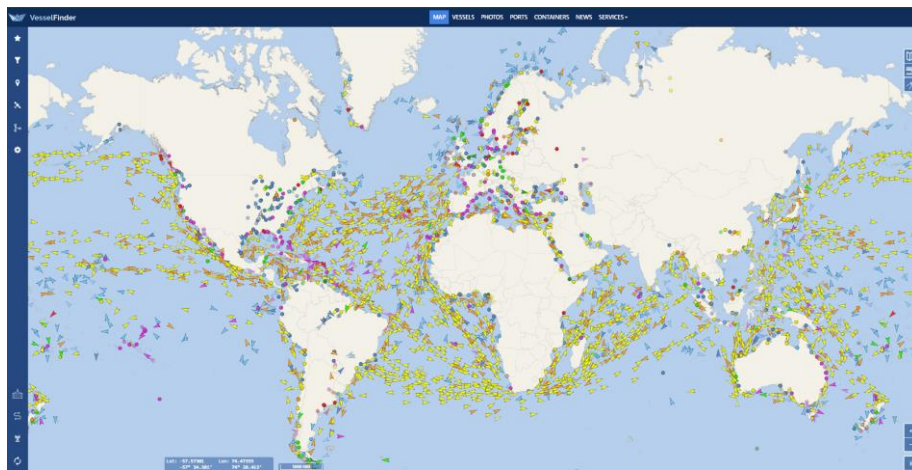


Figura 2-3 Mapa global página web *VesselFinder* con datos AIS [20]

La tecnología avanza rápidamente en los sistemas de monitorización e identificación existentes. Ya son varias las plataformas que siguiendo el avance tecnológico que vivimos, adoptan tecnologías como la AR para ofrecer servicios más avanzados, superando así el modelo tradicional de representación de datos 2D sobre un plano. Un claro ejemplo de esta tendencia es la aplicación desarrollada por Marine Traffic (véase la Figura 2-4) que presenta un funcionamiento similar al que se busca en este proyecto.

La aplicación recopila los datos AIS transmitidos por los barcos, y mediante la integración de la brújula magnética del dispositivo del usuario posiciona el dispositivo en el entorno. A continuación, muestra la información recibida ya traducida al usuario, mostrando una imagen clara de los datos de la embarcación mientras se observa al mismo en tiempo real. A mayores, la aplicación proporciona detalles sobre la ubicación de los barcos más cercanos al usuario, e informa de la actualidad de los datos del barco.



Figura 2-4 Captura de la aplicación de AR de *Marine Traffic* [21]

2.1.9 Antecedentes

El presente proyecto surge como continuación de su predecesor, que consistió en el desarrollo de una aplicación destinada a la representación de datos AIS en realidad aumentada. El objetivo principal de dicho proyecto era desarrollar un demostrador que, a partir de unos datos previamente establecidos, crease un objeto marcador y lo representase en una ubicación geográfica concreta mediante AR. Aunque dicho objetivo se cumplió, la precisión en la localización de los marcadores no alcanzó los niveles esperados. Además, los marcadores se creaban con datos ficticios preestablecidos. No obstante, este trabajo sentó las bases necesarias para el desarrollo de una aplicación más avanzada que permita representar marcadores en su localización con mayor precisión e integrar datos AIS, recibidos por una antena en tiempo real.

A parte del desarrollo del prototipo inicial de la aplicación, el trabajo de investigación realizado en el proyecto precursor ha ayudado en el avance de este proyecto, dado la similitud de la información requerida. Esta similitud ha permitido reutilizar los conocimientos adquiridos, lo que ha facilitado una investigación más sólida y la construcción de un contexto adecuado para el desarrollo del trabajo actual.

2.2 Realidad Aumentada (AR)

2.2.1 Definición

Según Ronald Azuma, en su trabajo *A Survey of Augmented Reality* [22], la Realidad Aumentada puede considerarse una variación del concepto de Entorno Virtual (VE) o Realidad Virtual (VR). Mientras que las tecnologías de los VE sumergen al usuario en un entorno completamente artificial, aislándolo por completo de la realidad, la AR se centra en complementar la realidad con elementos virtuales, tratando de conseguir el objetivo de que el usuario perciba una coexistencia entre ambos mundos (véase la Figura 2-5).

En términos generales la AR, puede definirse como la integración en tiempo real de datos digitales e información sensorial humana, que aparentemente se superpone sobre un espacio físico real [23]. A pesar de que este concepto pueda parecer relativamente reciente, su desarrollo comenzó en la década de los noventa, aunque no experimentó un impulso significativo hasta la llegada de los teléfonos inteligentes. Empresas como *Apple*, con *AR kit*, y *Google*, con *ARCore* no comenzaron a invertir de manera sustancial en estas tecnologías hasta el año 2017. Sin embargo, el ámbito de la AR no se limita únicamente a los dispositivos móviles; a principios de siglo emergieron variantes que, enfocadas en otros sentidos, como por ejemplo una pulsera para gente invidente que vibraba según la posición del usuario, indicándole de esta forma donde se encontraba el norte.

La AR puede considerarse como un punto intermedio entre la VR y la telepresencia. Sin embargo, es crucial mantener una distinción clara entre la AR y la VR. Como se explicó anteriormente, la diferencia clave entre ambas tecnologías es que, en el caso de la VR se requiere un casco o unas gafas para evadirse de cualquier estímulo del mundo real y lograr una experiencia inmersiva total en el mundo virtual. Mientras que en la VR nos introducimos en un mundo completamente virtual, en la AR este mundo digital es traído al mundo real para ser superpuesto en el entorno físico del usuario [24]. Actualmente existen tecnologías de AR que emplean dispositivos como gafas de VR los cuales muestran un video en tiempo real del mundo físico e integran elementos virtuales en la visualización, como ocurre con las gafas AR de *Apple*. Un ejemplo popular y exitoso de la AR es el conocido juego *Pokémon Go* que utiliza esta tecnología combinada con la localización geoespacial para proporcionar al jugador una experiencia inmersiva. Este tipo de aplicaciones se apoya en las mencionadas anteriormente *ARCore* y *ARkit* para posicionar los objetos 3D en lugares geométricos reales.

Por motivo de diferenciar las tecnologías de AR de herramientas como los HMDs (*Head Mounted Display*) nos centraremos en definir el sistema AR en tres puntos clave siguiendo el modelo de Ronald Azuma [22]:

1. Combina lo real y lo virtual: La AR fusiona elementos virtuales con el entorno físico, permitiendo que los usuarios interactúen con ambos de manera simultánea.
2. Es interactivo en tiempo real: Los sistemas de AR permiten que los usuarios interactúen con los objetos virtuales mientras están en movimiento.
3. Utiliza el 3-D. La AR emplea modelos tridimensionales para superponer objetos virtuales en el mundo real.

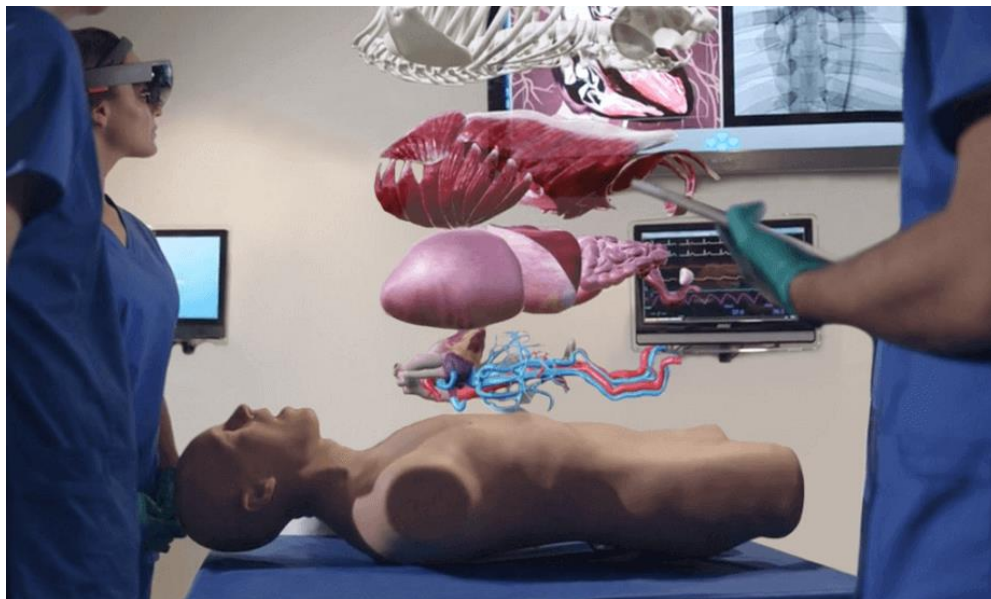


Figura 2-5 Ejemplo de uso de la AR en la enseñanza médica [25]

2.2.2 Importancia en la actualidad

Es común asociar el concepto de AR con los videojuegos o el entretenimiento, probablemente debido a la popularización de este concepto en la cultura popular, especialmente a través de películas o libros de ciencia ficción. Ejemplos de estos son películas como “*Minority Report*” o “*Iron Man*” y libros de ciencia ficción. Gracias a estos ejemplos se tiene una imagen de la AR como una tecnología futurista asociada a mundos imaginarios o a visiones distópicas, lo que lleva a pensar que esta tecnología aún está lejos de formar parte de nuestra vida cotidiana.

Sin embargo, el uso de la AR se ha extendido a una amplia variedad de sectores, tanto civiles como militares. En el ámbito de la medicina, por ejemplo, sirve como ayuda a los médicos al proporcionarles

visualizaciones de partes específicas del cuerpo humano o al integrar análisis de imágenes, como los de resonancias magnéticas, con lo que ven en tiempo real. En el sector de las reparaciones o la manufactura, la AR resulta útil cuando el objeto a crear o reparar tiene mucha complejidad, en estos casos permite a los operarios acceder a descripciones de las partes o incluso a guías paso a paso. Una de las aplicaciones más relevantes para este proyecto se encuentra en el ámbito militar, donde los pilotos de aviones y helicópteros han utilizado durante años sistemas como los HUD y los HMS (*Helmet-Mounted Sight*) para superponer gráficos y vectores a su campo visual, combinando de esta forma el vuelo visual e instrumental (véase la Figura 2-6) [22].



Figura 2-6 HMS combinación de la tecnología HUD con el casco de los pilotos [26]

Tanto en aplicaciones ya implantadas como futuras en desarrollo, el uso de la AR en diversos ámbitos de nuestras vidas podría ofrecer soluciones a problemas que ni siquiera se han planteado aún. Gracias a estas tecnologías se abre un amplio abanico de posibilidades, muchas de las cuales pueden no haberse imaginado hoy en día.

2.2.3 AR en la Armada

El concepto de AR aplicado a la Armada aún no se encuentra completamente desarrollado ni ha sido explorado en profundidad. No obstante, existen varios proyectos que, aunque no se consideren estrictamente como aplicaciones de AR, se aproximan a este tipo de tecnologías. La Armada dispone de diversos simuladores que generan entornos virtuales para la enseñanza de principios básicos de instrucción. Uno de los ejemplos más recientes es la incorporación de HMDs para el aprendizaje de maniobra de los alumnos de la Escuela Naval Militar. Estos dispositivos permiten a los profesores impartir una enseñanza más visual y dinámica, lo que facilita la transmisión de conocimientos por parte de los instructores. A través de esta tecnología, los estudiantes pueden aprender y practicar maniobras de manera más inmersiva, mejorando la comprensión y la adquisición de habilidades. Este tipo de iniciativas representan un paso significativo hacia la integración de tecnologías avanzadas en la formación de personal naval.

El futuro de la AR en la Armada se vislumbra cada vez más cercano y tangible con la incorporación del proyecto del gemelo digital, que estará disponible en las futuras unidades de la Armada, como son las F-110 (véase la Figura 2-7). El Gemelo digital consiste en la representación virtual de un buque real que recibe información importante de sus sistemas en tiempo real. Representación que permite ofrecer soporte durante las operaciones y para el mantenimiento del buque.

La idea principal que existe para las futuras nuevas unidades es la de crear un sistema que garantice mantener una mejora continua del producto durante todo su ciclo de vida, mediante pruebas y herramientas que optimicen tanto el diseño como el mantenimiento. Además, se pretende que el sistema sea capaz de almacenar, analizar y consolidar datos relevantes sobre la estática y dinamismo de los

buques de la misma clase. Este sistema también se integrará dentro de la unidad para dar un soporte a las aplicaciones operativas, logísticas y de alistamiento a bordo interactuando con sus sistemas [27]. Con esta innovación, se espera mejorar la eficiencia operativa y el mantenimiento preventivo, maximizando la disponibilidad de las unidades y optimizando la toma de decisiones durante las operaciones.

Dentro del servicio logístico que el gemelo digital puede aportar a las unidades, las tecnologías de VR y AR buscan facilitar y guiar a las dotaciones en la gestión y la realización de estas tareas. En el caso de AR, la organización toma un nuevo camino con la incorporación de esta. Puede facilitar el almacenamiento y la distribución de recursos, presentando la información de manera clara y accesible, lo que simplifica su manejo por parte del usuario.

Una parte importante del gemelo digital es el adiestramiento de las dotaciones. La incorporación de AR, unido al gemelo digital podría permitir a las dotaciones practicar e instruirse sin necesidad de estar a bordo de la unidad. Característica que plantea opciones interesantes, como tener dos dotaciones para un mismo barco con la misma formación que se vayan relevando para lograr una mayor efectividad dentro del barco además de mejorar las condiciones de vida de las dotaciones.

Por último y con la vista en el futuro, un desarrollo de la iniciativa del gemelo digital podría generar nuevos puestos de trabajo con la misión de ayudar a la navegación, supliendo problemas o simplemente ofreciendo soluciones a las cuales sea más complicado llegar desde el buque. En cuanto al sostenimiento, un operario desde tierra podría mediante VR o AR recibir una imagen o simulación del problema que está ocurriendo en el barco y ayudar en la reparación.



Figura 2-7. Gemelo digital en la Armada [28]

2.2.4 La Realidad Aumentada Geoespacial

Dentro del amplio espectro de aplicaciones y tecnologías de la AR, existe la AR Geoespacial. A diferencia de otras modalidades de AR, que posicionan objetos virtuales en relación con elementos locales como un código QR o la ubicación dentro de una habitación, la AR geoespacial permite la colocación de estos objetos en ubicaciones geográficas específicas. Esta capacidad de asociar objetos virtuales con coordenadas precisas en el espacio físico proporciona una mayor versatilidad y facilita la implementación de la tecnología en una amplia gama de contextos.

Existen diversos usos actuales para la tecnología de AR geoespacial. Un ejemplo claro es su aplicación en contextos como el de los excursionistas o militares, quienes pueden beneficiarse de una herramienta que, en lugar de depender de mapas o instrucciones en 2D, les proporcione indicaciones visuales en tiempo real para alcanzar su destino. Esta tecnología podría integrarse en una aplicación que

reciba información geográfica a través de GPS y, con base en estos datos, ofrezca las indicaciones pertinentes al usuario para llegar a su objetivo deseado. Desde el punto de vista táctico la ventaja de seguir indicaciones para usar la ruta menos complicada o más rápida para llegar a una localización estratégica podría significar la diferencia entre el éxito o el fracaso de las operaciones.

El posicionamiento de los objetos en AR geoespacial es posible gracias a una combinación de tecnologías de localización, como el GPS y las brújulas electrónicas, así como imágenes de servicios como *Street View* que ofrecen contexto visual de los alrededores. Estas tecnologías permiten que el sistema calcule la ubicación y orientación exacta del dispositivo móvil y, por ende, pueda colocar los objetos virtuales con precisión en el mundo físico. El GPS proporciona las coordenadas geográficas del dispositivo, mientras que la brújula ayuda a determinar la orientación en el espacio. Las imágenes de *Street View*, por su parte, permiten mejorar el posicionamiento comparando las imágenes capturadas por la cámara con las almacenadas en el servicio. La precisión de estas tecnologías depende de factores como la calidad de la señal GPS y la resolución de las imágenes obtenidas, lo que puede influir en la efectividad de la superposición de los objetos virtuales. Según investigaciones realizadas en este campo, el uso combinado de estas tecnologías ha demostrado ser efectivo para lograr una experiencia de AR geoespacial precisa y coherente, incluso en entornos urbanos densos [29] [30].

Para desarrollar este tipo de aplicaciones es necesario el uso de librerías que ofrecen soporte para AR geoespacial. Entre las opciones más comunes se encuentran *ARCore* y *ARKit*, que son las librerías desarrolladas respectivamente para *Android* y *iOS*, y que soportan AR geoespacial. Dado que este proyecto está enfocado en el desarrollo de una aplicación para dispositivos Android nos centraremos en la librería *ARCore*, la cual se detalla más adelante.

2.3 Entorno de desarrollo

2.3.1 Unity Engine

Para llevar a cabo un proyecto de esta índole, es fundamental contar con un entorno de desarrollo adecuado, que ofrezca un conjunto de herramientas, *software* y tecnologías necesarias para alcanzar el resultado deseado. Un desafío clave en esta fase es seleccionar la plataforma de desarrollo idónea para *Android*, con el objetivo de garantizar la máxima compatibilidad de la aplicación creada. Esto implica que tanto su funcionamiento como los recursos requeridos para su creación se integren de manera óptima con la plataforma elegida.

Existen diversas opciones disponibles actualmente para el desarrollo de aplicaciones, siendo *Android Studio*, *Unreal Engine* y *Unity* las más destacadas. Al analizarlas, todas resultan viables para la creación de un proyecto compatible con dispositivos *Android*, dado que ofrecen soporte para esta plataforma. Partiendo de esa base y al compararlas, *Unity* se posiciona como la opción más adecuada para el tipo de aplicación que se quiere desarrollar, tal y como se concluyó en el proyecto previo [5]. Su ventaja frente a las otras dos radica en la posibilidad de diseñar y modelar dentro de la propia plataforma, característica esencial para poder realizar los objetos virtuales que expongan información de forma clara y concisa. A mayores, *Unreal Engine* es un programa muy centrado en el desarrollo de videojuegos de gran realismo y rendimiento. En su momento se planteó como opción plausible, pero fue descartado debido a la escasa documentación disponible en relación con su integración con *ARCore*.

Unity Engine destaca por ofrecer un entorno de desarrollo intuitivo utilizando lenguaje C#, lo que facilita la creación de aplicaciones tanto en 2D como en 3D. Entre sus principales ventajas se encuentran la posibilidad de realizar un diseño detallado de las aplicaciones sin complicar excesivamente su uso, así como su gran compatibilidad con un gran número de plataformas. *Unity* soporta una amplia variedad de *plugins*, lo que permite integrar las últimas tecnologías de AR. De las distintas posibilidades como *Vuforia* o *ARKit*, este proyecto se centrará en el uso de *ARCore*, que fue la seleccionada para la aplicación.

2.3.2 ARCore

ARCore es una plataforma desarrollada por *Google* para crear experiencias de AR. Utiliza diferentes APIs (Interfaz de Programación de Aplicaciones), que permiten al dispositivo móvil percibir, comprender e interactuar con el entorno y la información que recibe. Esta plataforma se basa en tres conceptos claves para integrar contenido virtual con el entorno físico, permitiendo su visualización superpuesta en tiempo real a través de la cámara del dispositivo móvil.

En primer lugar, *ARCore* permite que el teléfono siga y comprenda su movimiento en relación con el mundo físico. En segundo lugar, proporciona la capacidad de detectar el tamaño y la ubicación de diversas superficies. Finalmente, *ARCore* tiene la capacidad de estimar las condiciones de iluminación del entorno, lo que facilita una integración más realista del contenido virtual con el mundo real [31].

Además de estas funciones básicas, *ARCore* puede beneficiarse de herramientas como *Google Street View*, que proporciona imágenes panorámicas detalladas de ubicaciones del mundo real. La integración de *Street View* dentro de aplicaciones basadas en *ARCore* permite superponer contenido virtual de manera más precisa sobre el entorno real al mejorar la ubicación de los marcadores, mejorando la experiencia del usuario. Al combinar las imágenes de *Street View* con la capacidad de localización de *ARCore*, las aplicaciones pueden utilizar mapas y modelos de entornos urbanos y rurales con un alto nivel de precisión, lo que resulta fundamental para aplicaciones que necesitan posicionar objetos virtuales en lugares específicos.

El SDK (Kit de Desarrollo de *Software*) de *ARCore* desarrollado por *Google* es un conjunto de herramientas para desarrollo de aplicaciones y depende de APIs para llevar a cabo sus funciones. Además de las APIs previamente mencionadas, existen tres APIs fundamentales que permiten a *ARCore* integrarse de manera completa con nuestra posición real y con los objetos virtuales que se desean posicionar en el entorno. Estas APIs son:

- Seguimiento de movimiento: API que permite a la aplicación rastrear tanto la posición como la orientación del teléfono en el mundo físico.
- Sesión: Se encarga de que *ARCore* esté conectado para realizar el seguimiento de movimiento.
- Anclaje: Esta API permite a los desarrolladores anclar los objetos al mundo físico con ubicaciones precisas.

Las tres APIs mencionadas anteriormente permiten la conexión de las aplicaciones con la red y la ubicación del dispositivo. Estas herramientas resultan fundamentales para alcanzar el objetivo de la aplicación, ya que permiten posicionar marcadores directamente sobre las embarcaciones, vinculándose con su ubicación geoespacial y la información AIS asociada.

3 DESARROLLO

En este capítulo, tras haber realizado una breve contextualización de la importancia de la identificación para la seguridad marítima, así como una explicación del funcionamiento de los AIS y los programas a utilizar, se procederá a dar una explicación del entorno de trabajo del proyecto y posteriormente, se describe el proceso de desarrollo de la aplicación y su funcionamiento.

3.1 Tecnologías y dispositivos empleados

Como es de esperar en un proyecto de estas características son varias las tecnologías o dispositivos necesarios para poder realizarlo con éxito. En este apartado se describirán los distintos elementos empleados y se justifica su elección. Algunos de los dispositivos siguen lo ya establecido en el proyecto anterior que se continua en este trabajo.

3.1.1 Receptor SDR

En este proyecto, no se utiliza un receptor AIS estándar, sino un RTL-SDR (véase la Figura 3-1 Antena conectada a la carcasa RTL-SDR con chip R820T2 y RTL2832U (Elaboración propia)., que dispositivo originalmente diseñado para la recepción TV HD o DVB-T TV (*Digital Video Broadcasting-Terrestrial*). Nuestra antena receptora trabajará desde 100 KHz a 1,7 GHz, y es del tipo HF Q Branch *direct sampling*. Para entender bien el funcionamiento y especificación de la antena hay que saber que los receptores SDR transforman una señal analógica en una digital, concretamente por un ADC que transforma las señales de radio captadas en electricidad, la cuál es posteriormente convertida en valores numéricos representativos de la señal original. Los receptores RTL-SDR emplean dos ADC debido al uso de un muestreo de cuadratura, I/Q, que transforma la señal recibida en dos ondas desfasadas 90 grados. Este enfoque simplifica el ajuste de las amplitudes y facilita la obtención de la fase deseada, a diferencia del muestreo convencional de una sola onda. Para su funcionamiento, los receptores RTL-SDR incorporan un chip RTL2832U, que actúa como demodulador digital, y un chip R820T2, que se encarga de convertir la frecuencia recibida a una frecuencia intermedia constantes. Dado que se trata de una antena de “*direct sampling*”, los chips procesan directamente la señal de radio, sin necesidad de un convertidor de aumento o reducción [6] [32].



Figura 3-1 Antena conectada a la carcasa RTL-SDR con chip R820T2 y RTL2832U (Elaboración propia)

La antena utilizada para este proyecto es una antena de quince centímetros, recubierta de goma que opera en un rango de frecuencia VHF 136-174 MHz. VHF hace referencia a las frecuencias que van desde los 30 MHz a los 300 MHz, estas son las ideales para transmitir mensajes en largas distancias, gracias a la poca absorción de señal en la atmósfera y los objetos que tiene, que, a mayores ayuda a mantener la calidad de la señal [33].

Las características de la antena VHF la hace ideal para la transmisión de datos en espacios marítimos, razón por la que el rango de frecuencia VHF es el utilizado por los AIS, justificando el uso de una antena VHF para el trabajo.

3.1.2 Aplicación AIS-catcher

AIS-catcher [34] es un programa y una comunidad de usuarios entusiastas del AIS que se dedican al seguimiento del movimiento marítimo. Esta aplicación incorpora un *driver* que es capaz de utilizar antenas con receptores RTL-SDR, similares a los empleados en estaciones AIS de alta efectividad. Esto permite al usuario detectar, visualizar y analizar los movimientos de las embarcaciones, así como compartir los datos con la comunidad de *AIS-catcher* y otras aplicaciones mediante un socket UDP. El software con el que trabaja esta aplicación además de ser fuente abierta y es compatible con una amplia gama de dispositivos. En la Figura 3-2 se puede observar la interfaz de usuario en el teléfono móvil.

Como se puede ver en la Figura 3-2 hay un panel de control en la parte de debajo de la aplicación que permite definir varias funciones. En el extremo izquierdo se encuentra el *start/stop* de la aplicación que determina cuando se quiere empezar a recibir datos AIS y cuando parar. Más hacia el extremo derecho, en el penúltimo logo, se tiene el botón digital de selección de la de recepción, en la Figura 3-2 se muestra cómo se tiene seleccionada la opción de RTL-SDR. Los puntos azules que aparecen en la pantalla son los contactos AIS que está recibiendo la aplicación en ese momento.

Si se centra la mirada en el panel superior, como muestra la Figura 3-2 el usuario se encuentra en el modo mapa, que permite ver un mapa de los alrededores junto con las posiciones AIS representadas como círculos de los barcos en los alrededores. Por otro lado, siguiendo el panel de control hacia la derecha se muestra la opción de NMEA. Dentro de esta selección existe una pantalla (

Figura 3-3) donde se plasma en formato NMEA la información AIS de los barcos que tendrá que ser decodificada en la aplicación para su funcionamiento.

Antes de comenzar a utilizar la aplicación, este panel de la aplicación es la manera más rápida de comprobar si se están recibiendo los mensajes necesarios para que se creen los marcadores. Fijando la atención en los marcadores se observa como justo antes del mensaje con formato NMEA aparece el tipo de transpondedor que el barco utiliza, en este caso señales de transpondedores tipo A y tipo B, clasificación que ya se trató en el anterior capítulo.

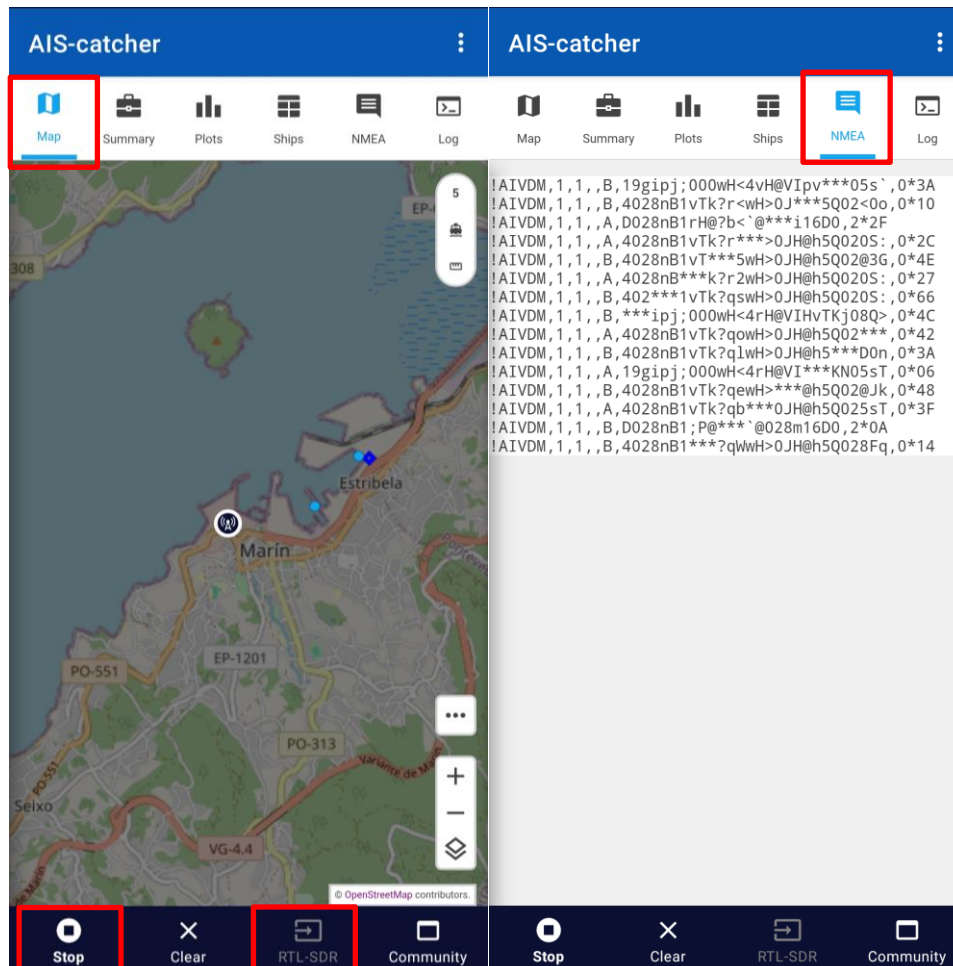


Figura 3-2 Captura de pantalla (izquierda) panel principal AIS-catcher (Elaboración propia)

Figura 3-3 Captura de pantalla (derecha) panel de mensajes formato NMEA (Elaboración propia)

Es importante configurar dentro de AIS-catcher el desajuste de frecuencia del receptor RTL-SDR, mostrado en la Figura 3-4 cómo *Frequency Correction* (PPM). El número uno establecido es el indicado como 1 PPM (Partes Por Millón), mostrado en el receptor RTL-SDR en la Figura 3-1. Un valor de PPM incorrecto puede causar una desviación en la frecuencia de sintonización, lo que afecta la exactitud de la recepción de señales, por lo que es fundamental calibrar correctamente el valor de PPM en la aplicación AIS catcher.

Por último, para que se puedan reenviar los datos AIS desde AIS-catcher es necesario configurar el servidor de salida de los mismos. Como se muestra en la Figura 3-5 existen varios servidores UDP para escoger, en este trabajo se escoge el servidor uno. Dentro del servidor es necesario configurar la dirección IP, que en este trabajo es la dirección local o *localhost* del propio dispositivo (127.0.0.1), y el puerto de salida, puerto que debe coincidir con el indicado dentro del script de nuestra aplicación. Una vez configurado el reenvío de los datos en AIS catcher estos se envían a la aplicación desarrollada mediante el *Socket UDP* antes mencionado. Los *sockets* permiten la comunicación entre aplicaciones, y

en este proyecto se emplea el protocolo UDP por su velocidad y eficiencia, ideal para transmitir los mensajes NMEA en tiempo real, aunque no garantiza la entrega ni el orden de los datos.

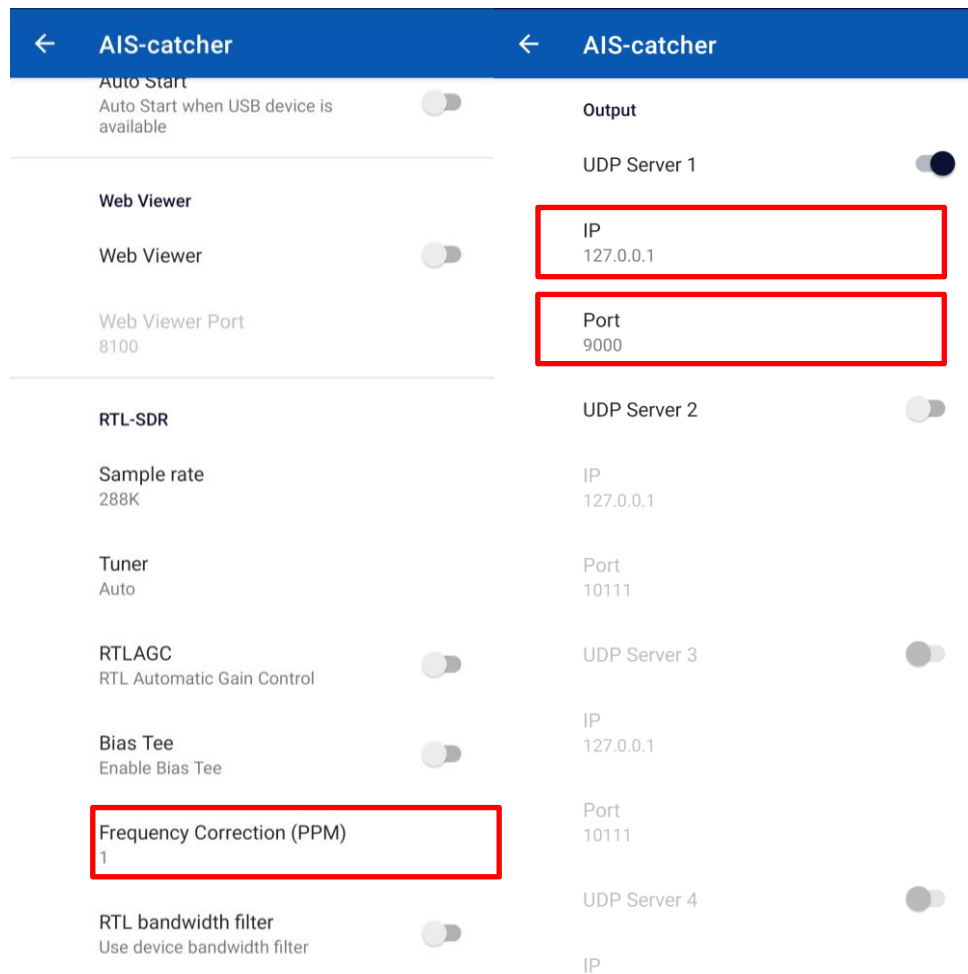


Figura 3-4 Captura de pantalla (izquierda) de configuración de *AIS-catcher*

Figura 3-5 Captura de pantalla (derecha) de configuración de *AIS-catcher*

3.1.3 Dispositivo móvil

Todo el desarrollo de la aplicación se ha realizado para que esta funcione adecuadamente en el dispositivo asignado para el proyecto, en concreto, un Xiaomi Redmi Note 11. Este teléfono tiene un sistema operativo *Android*, lo que justifica el uso de *ARCore* en el desarrollo de la aplicación. La especificación del tipo de teléfono móvil a utilizar es importante ya que al desarrollar la aplicación hay que distinguir que tipo de nivel de API puede soportar el dispositivo y tener un teléfono antiguo podría limitar el mismo desarrollo de la aplicación. En la Figura 3-6 se muestra cómo se conecta la antena y el receptor RTL.SDR mediante conexión USB formando el dispositivo completo y autónomo utilizado para el funcionamiento de la aplicación.



Figura 3-6 Dispositivo móvil y antena RTL-SDR.

3.2 Entorno de desarrollo

Como se explicó en el apartado anterior, para la realización del proyecto es necesario el uso de una herramienta de desarrollo. La selección de la herramienta adecuada es importante para que nuestro proyecto cumpla con los requisitos que se buscan. Por las razones explicadas en el estado del arte el motor seleccionado en nuestro caso es *Unity Engine*, que utiliza C# como lenguaje de programación. A continuación, se describe el proceso de configuración en Unity para el desarrollo de una aplicación de realidad aumentada geoespacial en *Android*.

3.2.1 Configuración de Unity

La plataforma de Unity es la elegida para el proyecto, en concreto la versión de 2021, la misma que el proyecto anterior. En el inicio del proyecto se probó el uso de otras versiones de Unity, más modernas como la 6.0.0.0 del 2024. Finalmente se decidió utilizar la versión del 2021 al ser la que más documentación tenía sobre cómo desarrollar una aplicación de estas características.

Una vez instalada la versión del motor seleccionada, el siguiente paso consiste en configurar el proyecto nuevo para seleccionar Android como plataforma objetivo (*target*). Esto se hace en la opción *switch platform* del apartado de *build* de Unity. Hay varias opciones como IOS o WebGL, pero el proyecto se centrará solo en *Android*. La siguiente opción que hay que modificar del programa se encuentra en *player settings*, al cual se llegan también desde *build*, donde se debe seleccionar también que la plataforma objetivo será *Android*. En este mismo apartado, dentro de la opción de *Android* se modificarán algunos parámetros para que la aplicación funcione el dispositivo *Android*, como el nivel de compatibilidad de API, la cual dependiendo de la antigüedad del dispositivo será mayor o menor.

Antes de comenzar a describir cómo se configura el entorno de *Unity* para crear el proyecto, es necesario conocer tres elementos clave dentro del desarrollo:

- *Arcore Extensions*

Como se explicó en el capítulo dos *ARCore* hace uso de APIs para que el dispositivo siga el movimiento de la cámara, sea capaz de identificar tamaños y formas de objetos y además pueda diferenciar los distintos tipos de iluminación. De *ARCore* dentro del proyecto es necesario descargar el paquete de Google *ARCore Extensions* correspondiente a la versión de *Unity* que se esté usando. Este paquete habilita funciones adicionales de *ARCore* para *AR Foundation*. La funcionalidad adicional que este paquete ofrece para nuestra aplicación es la de *Cloud Anchor*, esta función permite que se coloquen

objetos de AR con ubicaciones reales mediante el uso de conexión a internet, característica que debe cumplirse para que funcione la aplicación. Para descargar e instalar este paquete se introduce la URL de *Github*, en el *package manager* de *Unity*. Es importante que se utilice la versión adecuada con respecto a la versión que se utiliza de *Unity*. Una instalación incorrecta de este paquete puede dar lugar a problemas y que no se pueda establecer una escena correcta del programa lo que conlleva a errores en el desarrollo.

- *API Key*

Tal como se explicó en el capítulo dos una API (*Application Programmed Interface*) es una pieza de código que permite a dos aplicaciones comunicarse entre sí para intercambiar información y funcionalidades. En nuestra aplicación, *ARCore* hace uso de las APIs de *Google* para procesar la información recibida por el dispositivo. Para poder utilizar las APIs de *ARCore* y por ende las funciones AR geoespacial de *Google*, es necesario crear un proyecto en la *Google cloud platform console*. Una vez se ha creado este proyecto, en el apartado de credenciales, dentro de la misma página se da la posibilidad de crear una API privada. De esta forma se genera una clave privada única que debe ser copiada y pegada en el apartado en *Project settings* de *Unity* donde seleccionando *ARCore extensions* y *API Key* en la estrategia de *Android*, se nos abrirá la opción de pegar nuestra clave privada o *API Key*. Una vez colocado el código que copiamos de la página web ya tendríamos nuestro proyecto configurado para empezar a desarrollarlo.

- *Visual Studio*

Por último, dentro del entorno de trabajo hay que mencionar el programa utilizado para desarrollar el código necesario para realizar este proyecto ha sido *Visual Studio*, programa que puede instalarse desde la plataforma de desarrollo *Unity*. Esta aplicación permite escribir y modificar cualquier *script* incluido dentro del proyecto de *Unity* y que estos cambios se guarden al instante, además de compilarlos en cuanto son guardados. Como peculiaridad dentro de *Unity* todos los *scripts* que estén dentro del proyecto, aunque no estén en la escena se compilan en todo momento. *Visual Studio* y *Unity* facilitan el desarrollo del proyecto, ya que, por ejemplo, los errores de compilación se muestran directamente en la consola de *Unity*.

Dentro de *Unity* se encuentran varios elementos importantes que componen el proyecto. Un elemento principal es la llamada escena donde se definen el resto de componentes, tales como los paquetes de desarrollo o *plugins* que se utilizan, la cámara que se va a utilizar, los objetos que se quieren representar y dentro de estos los *scripts* con su código correspondiente.

Cuando se comienza un proyecto nuevo en *Unity*, este crea una escena predeterminada con los componentes de *main camera* y *directional light*. Ambas no son necesarias y se pueden eliminar. Antes de comenzar a crear la escena es necesario descargar los paquetes con las extensiones o *plugins* necesarios. Estos paquetes permiten el desarrollo de aplicaciones de AR e incluyen, *AR foundation*, *ARCore* con sus *Google Extensions* y *XR Tools* (si no está previamente instalada). Estos paquetes incluyen los siguientes elementos los que incorporamos a la escena vacía creamos el elemento *AR Session*, *ARCore Extension* y *AR Session Origin* (véase la Figura 3-7).

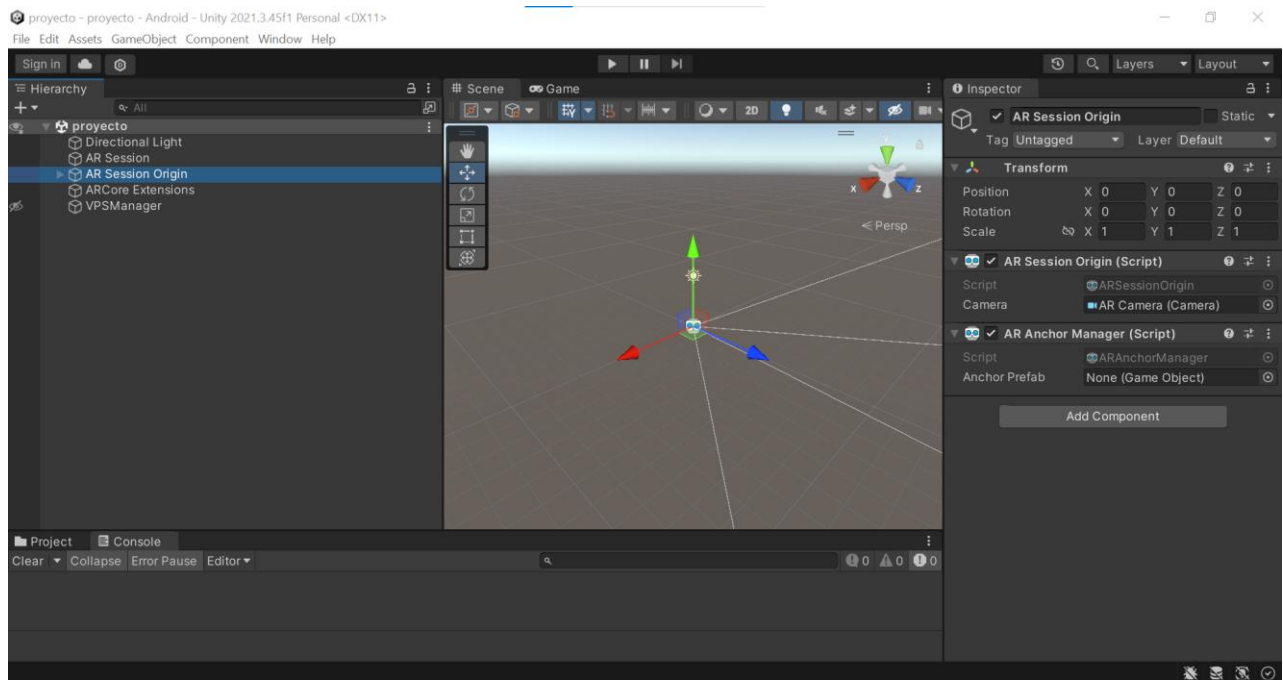


Figura 3-7 Escena Unity con los diferentes componentes

- *AR Session Origin*: Es el elemento de la escena que va a definir la cámara que se va a usar. Es crucial para el proyecto puesto que es la encargada de que la aplicación utilice la cámara del teléfono móvil, que se muestra como un objeto hijo llamado *AR camera*. Dentro de este elemento, se crea el script (código) *AR Anchor Manager*, que es el encargado de ubicar objetos de AR dentro de la aplicación.
- *ARCore Extensions*: Es un elemento fundamental para realizar un seguimiento de objetos de forma geoespacial. Dentro de este elemento, se configuran los demás elementos, como se muestra en la Figura 3-8, de modo que puedan generarse correctamente considerando tanto la geolocalización asignada a los objetos como la del propio dispositivo. *ARCore Extensions* agrega funcionalidad adicional a los paquetes de *AR Foundation* que es el paquete encargado de permitir crear aplicaciones de AR.

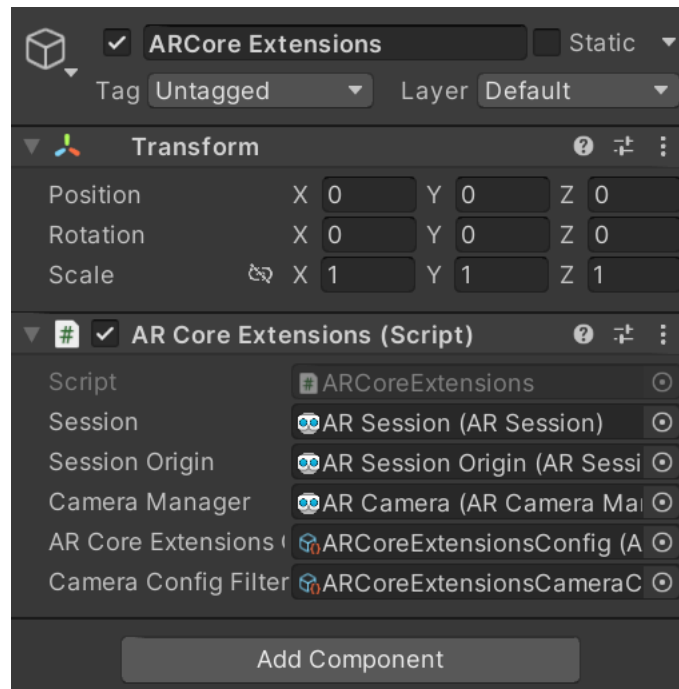


Figura 3-8 Configuración de *ARCore Extensions*

- *VPSManager*: Por último, para poder crear los marcadores que se quieren emplazar en las posiciones AIS recibidas más adelante, se debe crear un objeto 3D vacío que va a albergar el *script* encargado de crear y controlar los marcadores, y también el *prefab* u objeto modelo del marcador como se observa en la Figura 3-9. Dentro del *VPSManager* se debe enlazar el elemento *AR Earth Manager*, elemento clave para la geolocalización. Es importante activar la geolocalización dentro de los *player settings* de este elemento. También se asigna en el editor *AR Anchor Manager*, fundamental para que los marcadores dispongan de un anclaje en el mundo físico.

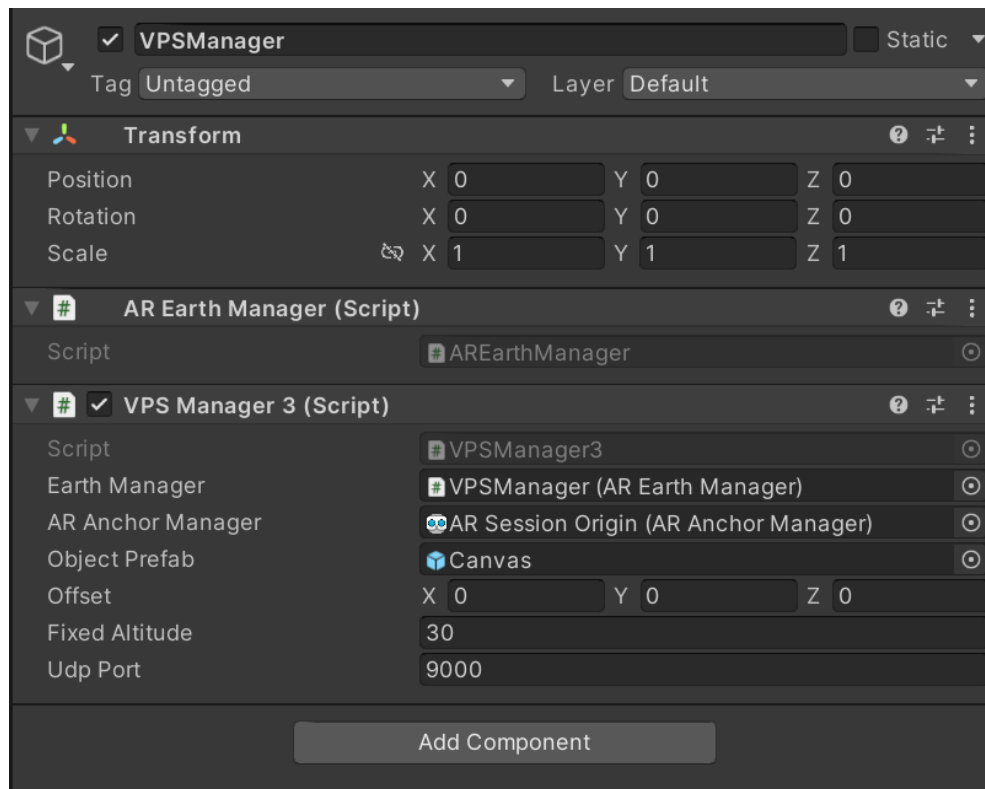


Figura 3-9 Elementos del *VPSManager* con campos que son parte del script

3.2.2 Marcador inicial

El diseño del marcador es un aspecto crucial dentro del proyecto, su forma y tamaño deben ser adecuados para permitir ver los datos que se quieren mostrar. Además, su color debe permitir que el contraste entre las letras y del fondo sea lo suficiente para permitir distinguir con claridad los datos mostrados. El marcador de la Figura 3-10 muestra el primer diseño. Esta primera versión solo muestra la distancia y el nombre del barco. A medida que el proyecto evolucione el marcador lo hará a su vez, adaptándose a las necesidades que aparezcan por el nuevo tipo de información que se quiera plasmar. Para lograr esto, *Unity* ofrece herramientas de diseño y modelado que permiten modificar el *prefab* o modelo para obtener el resultado deseado.

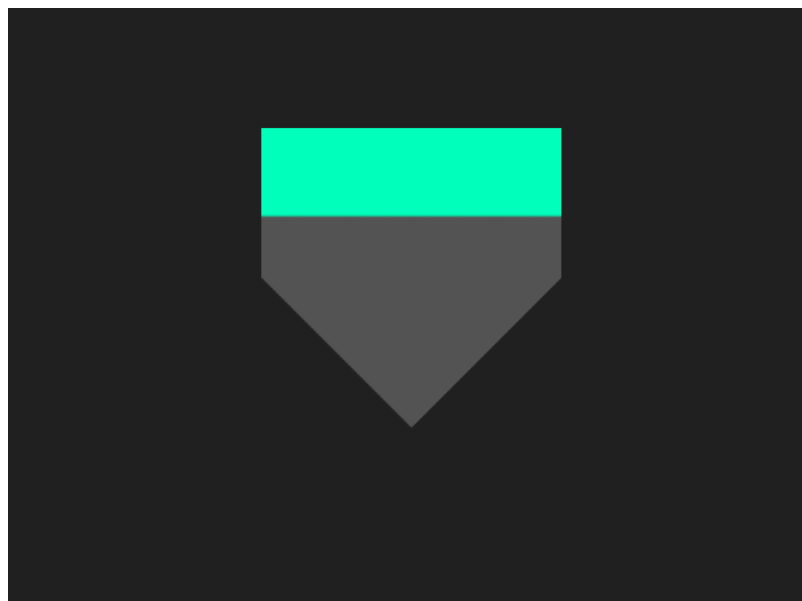


Figura 3-10 Primer diseño de marcador

3.3 Primera versión

Este proyecto al tratarse de una continuación de uno previamente existente contaba con una guía inicial para la creación de la aplicación. La escena que se debía gestionar se basa en un prelude del proyecto, construido siguiendo tutoriales y guías disponibles en línea, específicamente de la referencia [35]. Tras elegir *Unity* como plataforma, se recreó la escena desde el principio, para poder obtener un mayor conocimiento del entorno en el que se iba a trabajar. El *script VPSManager* se implementó a partir del mencionado en [35] con algunas modificaciones realizadas. Esta primera versión del código permitía, mediante la introducción manual de una lista de nombres, sus geolocalizaciones y alturas, crear marcadores con datos y ubicaciones predefinidas. Para que las coordenadas que se introducían pudiesen ser leídas por el programa tenían que estar en decimales y la altura que se usaba tenía que ser calculada mediante *Google Earth*.

Este primer código proporciona características claves para el proyecto, como la capacidad de mantener los marcadores visibles en todo momento en la pantalla y la de guiar al usuario hacia la dirección en la que debe mover la cámara para centrar los marcadores en la pantalla. A continuación, se abordarán los aspectos más relevantes de este código, que establecen las bases principales sobre las cuales se ha desarrollado el proyecto.

Como se puede observar en el Anexo IV Primera versión *VPSManager* el código comienza estableciendo la clase pública *VPSManager*. Este aspecto es crucial dentro de *Unity* puesto que este nombre de la clase debe coincidir con el objeto *VPSManager* que hemos creado dentro de la escena para que el *script* se pueda compilar sin problemas. En el resto del código se definen diversas variables que acompañarán al proyecto a lo largo de su desarrollo:

- *AREarthManager*: Variable que gestiona la funcionalidad de *ARCore*.
- *ARAnchorManager*: Gestiona los anclajes AR permitiendo colocar los objetos en el espacio físico.
- *List <EarthPosition>*: Lista donde se van a guardar las posiciones geográficas donde se colocarán los objetos.
- *GameObject ObjectPrefab*: El *prefab* o modelo del objeto que se instanciará en las posiciones indicadas.
- *Vector3 offset*: Vector que sirve para ajustar la posición del icono en la pantalla.
- *List<Marker> markers*: Lista interna donde se almacenarán todos los marcadores que se creen.

También se establecen estructuras internas necesarias para poder establecer las funciones que se quieren implementar más adelante y poder crear los marcadores necesarios.

- *Earthposition*: Representa el marcador con una posición geográfica con latitud, longitud, altitud y el nombre que se le quiera dar.

```
[Serializable]
public struct EarthPosition
{
    public double Latitude;
    public double Longitude;
    public double Altitude;
    public string name;
}
```

- *Marker*: Donde se guarda el *prefab* que se quiere representar y el anclaje necesario para anclar los *prefabs* a posiciones fijas del mundo real.

```
[Serializable]
private struct Marker
{
    public ARGeospatialAnchor Anchor;
    public GameObject Object;
}
```

En la segunda parte del código se encuentran los métodos que constituyen el cuerpo principal del código. Estos métodos que se detallan en el Anexo IV Primera versión *VPSManager* son los siguientes:

- *Start*: Método inicial que se ejecuta al comienzo del código, se llama al método *VerifyGeospatialSupport* que comprueba si el dispositivo soporta el modo geoespacial.

```
void Start()
{
    VerifyGeospatialSupport();
}
```

- *VerifyGeospatialSupport*: Este método se encarga de comprobar si el dispositivo realmente soporta el modo geoespacial, para ello crea tres casos; si el dispositivo es capaz de soportar el modo geoespacial, llama al método *PlaceObjects*, para situar los marcadores; si no se tiene respuesta, la función *Invoke* espera cinco segundos tras los cuáles vuelve a llamar a *VerifyGeospatialSupport*; y, por último, en caso de que el dispositivo no pueda soportarlo, lo registra como mensaje y no hace nada más.

```
private void VerifyGeospatialSupport()
{
    var result =
        earthManager.IsGeospatialModeSupported(GeospatialMode.Enabled);

    switch (result)
    {
        case FeatureSupported.Supported:
            Debug.Log("Ready to VPS");
            PlaceObjects();
            break;
        case FeatureSupported.Unknown:
            Debug.Log("Unknown...");
            Invoke("VerifyGeospatialSupport", 5.0f);
            break;
        case FeatureSupported.Unsupported:
            Debug.Log("Unsupported");
            break;
    }
}
```

- *PlaceObjects*: Comprueba si el estado de seguimiento de la Tierra es *Tracking* en ese momento con *earthManager EarthTrackingState*. Con *geospatialPose* obtiene la posición geoespacial de la cámara y verifica que la precisión vertical y horizontal es aceptable (menor a cinco metros según como está planteado en el código). A partir de aquí se crean tres posibles casos:
 1. Si la precisión es adecuada: Se obtiene la rotación de la cámara, mediante *Quaternion rotation*. Itera cada *Earthposition* previamente guardado en *earthpositions*, crea un *Marker* y añade un anclaje geoespacial en las coordenadas especificadas, instancia el *prefab* del objeto y actualiza su nombre, y por último añade el nuevo *marker* a la lista de *markers*.
 2. Si la precisión no resulta ser la suficiente entonces espera dos segundos y lo vuelve a intentar.
 3. Si el estado de seguimiento de la Tierra es *NONE*, vuelve a intentarlo tras cinco segundos.

```
private void PlaceObjects()
{
    if (earthManager.EarthTrackingState ==
TrackingState.Tracking)
    {
        var geospatialPose = earthManager.CameraGeospatialPose;

        if (geospatialPose.HorizontalAccuracy <= 5 &&
geospatialPose.VerticalAccuracy <= 5)
        {
            Quaternion rotation = geospatialPose.EunRotation;
            foreach (var obj in earthpositions)
            {
                var marker = new Marker();
                marker.Anchor =
ARAnchorManagerExtensions.AddAnchor(aRAnchorManager, obj.Latitude,
obj.Longitude, obj.Altitude, rotation);
                marker.Object = Instantiate(ObjectPrefab);

                marker.Object.transform.Find("Image/Name").GetComponent<TMP_Text>()
.text = obj.name;
                markers.Add(marker);
            }
        }
        else
        {
            Invoke("PlaceObjects", 2.0f);
        }
    }
    else if (earthManager.EarthTrackingState ==
TrackingState.None)
    {
        Invoke("PlaceObjects", 5.0f);
    }
}
```

- *Update*: En la parte final del código se encuentra el método *Update*. Los métodos *Update* de todos los objetos en *Unity* se llaman una vez por *frame*. Este método itera sobre cada *marker* dentro de la lista. Lo primero que hace es obtener el componente de imagen del *prefab*, que contiene el objeto del marcador con los campos nombre y la distancia, para manipular su posición en la pantalla. A continuación, calcula los límites de X e Y para asegurar que el icono no se salga en ningún momento de la pantalla. En *Vector2* utiliza *WorldToScreenPoint* para convertir una posición real del mundo en coordenadas de la pantalla. Lo siguiente que hace el código es crear una condicional en la cual calcula si el objeto se encuentra delante o detrás de la cámara, si el resultado es menor que cero, entonces el objeto se encuentra detrás. Al cumplirse la condición establecida, el código ajusta la posición de la imagen para que el marcador aparezca en el borde de la pantalla. Además, con *Mathf.Clamp* limita el valor de la posición de la imagen entre un rango de máximos y mínimos, en este caso los límites de la pantalla. Por último, actualiza la posición del icono en el interfaz con pos, y en *marker Text* calcula y muestra la distancia en metros desde la cámara al anclaje.

```

private void Update()
{
    foreach (var marker in markers)
    {
        var img = marker.Object.GetComponentInChildren<Image>();
        float minX = img.GetPixelAdjustedRect().width / 2;
        float maxX = Screen.width - minX;
        float minY = img.GetPixelAdjustedRect().height / 2;
        float maxY = Screen.height - minY;
        Vector2 pos =
        Camera.main.WorldToScreenPoint(marker.Anchor.transform.position + offset);

        if (Vector3.Dot((marker.Anchor.transform.position -
        Camera.main.transform.position), Camera.main.transform.forward) < 0)
        {
            if (pos.x < Screen.width / 2)
            {
                pos.x = maxX;
            }
            else
            {
                pos.x = minX;
            }
        }
        pos.x = Mathf.Clamp(pos.x, minX, maxX);
        pos.y = Mathf.Clamp(pos.y, minY, maxY);
        img.transform.position = pos;
        marker.Object.transform.Find("Image/Distance").GetComponent<TMP_Text>().text =
        ((int)Vector3.Distance(marker.Anchor.transform.position,
        Camera.main.transform.position)).ToString() + "m";
    }
}

```

Una vez implementado el script *VPSManager*, se procede a acceder a *Build Settings* desde la opción *File* en la plataforma *Unity*. En esta ventana, se selecciona la opción *Build* para generar la aplicación la cual posteriormente se instala en el dispositivo *Android* para su prueba y ejecución. Una vez se ha realizado la instalación de la aplicación en el dispositivo y tras darle los permisos pertinentes para funcionar adecuadamente, la imagen que obtenemos es la que podemos observar en la Figura 3-11 Como se puede observar, los marcadores que están fuera del foco de la cámara se colocan en el lado de la pantalla según hacia donde se debe apuntar el dispositivo visualizarlos en su localización. Cada marcador muestra en la parte inferior el nombre asignado al marcador de esa geolocalización, junto con la distancia en metros entre la localización del usuario y la del marcador.



Figura 3-11 Funcionamiento código original (Elaboración propia)

3.4 Recepción de datos AIS

Tras implementar, compilar, instalar y validar la aplicación base, el primer paso en el desarrollo consistió en crear las funciones necesarias para recibir los mensajes NMEA que contienen los datos AIS, generados por la aplicación *AIS-catcher*, mediante el uso de un *socket* UDP. A continuación, se describen los pasos más relevantes del proceso seguido.

Se desarrolló el método *UDPListen*, componente esencial en el proyecto, y común en aplicaciones que requieren transmisión eficiente y en tiempo real, como los sistemas de navegación marítima basados en AIS. Este método permite la recepción continua de datos desde una fuente remota a través de la red, utilizando el protocolo UDP.

Como se ha mencionado anteriormente en este capítulo, el protocolo hace uso de los *sockets* UDP que permiten establecer un canal de comunicación que facilita la recepción continua de datos. Dentro del código se crea un *socket* UDP en el puerto específico configurado (en el caso del proyecto, el puerto 9000) para escuchar y recibir los paquetes de datos enviados desde la fuente remota (en el caso del proyecto la aplicación *AIS-catcher*) de forma continua.

Los primeros esfuerzos se centraron únicamente en recibir los datos AIS de forma correcta, sin decodificar los datos recibidos. Primero se creó el código *AISReceiver* (véase el Anexo VI: Código *AISReceiver*) utilizando un código externo como solución de recepción de datos AIS, solución que quedó descartada al presentar demasiados problemas en su funcionamiento. Posteriormente, se creó el *script* *VPSManagerUDP*, (véase el Anexo VII: Código *VPSManagerUDP*), al recibir una sentencia NMEA con éxito establece un marcador en unas coordenadas fijas ya especificadas. Este código recoge las mejoras implementadas por el código *AISReceiver*, y los implanta en su propio código con el objetivo de establecer las bases de escucha de los mensajes AIS.

El funcionamiento de este código consiste en utilizar el método *UDPListener* que comienza a escuchar datos del puerto 9000. Estos datos, en formato NMEA recibidos de la aplicación *AIScatcher* se guardan en una cola *thread-safe* (*ConcurrentQueue*) para que puedan ser procesados posteriormente. Cada vez que se recibe un mensaje válido, el código procesa la sentencia NMEA y crea un marcador geoespacial en el mundo AR. La creación del marcador y su actualización se controlan mediante el código original del método *Update*. A mayores el funcionamiento del código se optimiza mediante el uso de multihilos, es decir, *VPSManagerUDP* utiliza un hilo separado (*Thread udpListenerThread*) para escuchar mensajes UDP de manera asíncrona sin bloquear el hilo principal.

Cabe destacar que la implementación completa e integrada en el código final de estos scripts y su funcionamiento detallado se recogen en el apartado final de este capítulo, con el fin de evitar redundancias en este apartado.

3.5 Decodificación de datos AIS

Tras haber logrado encontrar una solución para la escucha de datos AIS, el siguiente reto planteado consistía en conseguir decodificar correctamente las sentencias NMEA. En concreto, el objetivo principal en esta fase era interpretar los mensajes dinámicos del sistema AIS, que contienen información cinemática y de posicionamiento fundamental para representar gráficamente las embarcaciones mediante marcadores georreferenciados en el entorno de realidad aumentada. No obstante, estos mensajes no incluyen información identificativa como el nombre del buque, por lo que, en una fase posterior, se intentó también la decodificación de los mensajes estáticos del sistema AIS, con el fin de enriquecer la visualización de los contactos. La Tabla 3 recoge los distintos tipos de mensajes AIS y su contenido, lo cual fue clave para distinguir qué información era prioritaria decodificar inicialmente.

Para decodificar los mensajes, se plantearon dos alternativas: desarrollar un decodificador desde cero de los mensajes más relevantes a partir de la especificación de NMEA o utilizar las funciones de alguna librería previamente desarrollada. Tras algunas pruebas iniciales se determinó que los conocimientos de

programación y el esfuerzo requerido para desarrollar un decodificador desde cero sobrepasan el alcance de este proyecto. Por esta razón, se decidió investigar en línea, explorando aplicaciones y comunidades que ya cuentan con programas diseñados específicamente para este tipo de tareas. El objetivo era encontrar una librería que pudiera ofrecer una solución viable o, al menos, proporcionar orientación sobre cómo mejorar el código existente. La mayoría de los programas encontrados como por ejemplo el mismo *AIS.catcher* están desarrollados en otros lenguajes de programación y utilizan una cantidad gigantesca de scripts para lograr que sus avanzadas aplicaciones e interfaces funcionen de la forma que lo hacen. Dado que nuestro proyecto tiene un alcance más limitado, las soluciones encontradas superan ampliamente los objetivos establecidos, lo que generaba una desconexión considerable entre las necesidades del proyecto y las soluciones existentes.

Este patrón de búsqueda se repitió hasta que se encontró un artículo en el que se describe el desarrollo y las mejoras implementadas a la aplicación de *OceanMind*. Dicho artículo expone de forma clara y simplificada los distintos scripts implementados y la función de cada uno [36]. Otro punto a favor del código de esta aplicación es que utilizan C# como lenguaje de programación, característica clave para incorporar los métodos buscados de decodificación a nuestro *script* de *Unity*. Entre los diversos *scripts* expuestos en el artículo se encontraron las soluciones que se necesitaban en el proyecto para poder decodificar los datos AIS de la sentencia NMEA, véase el código del Anexo VIII: Código de procesamiento datos AIS, que utiliza los métodos de la interfaz del Anexo IX Código *INmeaAisMessageStreamProcessor*. Estos ofrecen un procesamiento o “parseo” de los datos dinámicos de las sentencias NMEA, contenidos en los mensajes AIVDM en una representación ASCII (*American Standard Code for Information Interchange*). Dentro del mensaje AIVDM los caracteres del mensaje AIS se representan utilizando una codificación de seis bits por carácter, codificación que se convierte en una secuencia ASCII legible.

Continuando con la decodificación de las sentencias NMEA, aparece la problemática de obtener datos como el nombre del barco, que no se incluyen en los mensajes dinámicos. Como se puede observar en el Anexo III: Tipos de mensajes AIS y su contenido, los mensajes AIS se dividen en varios tipos, y no es posible obtener todos los datos al mismo tiempo. La solución que ofrece el artículo[36] consiste en el uso de una herramienta de decodificación como el *NmeaAisTextFieldParser*, que se encarga específicamente de extraer el nombre del barco de los mensajes AIS. Este *parser* que trabaja tomando los bits del mensaje que contiene el nombre del barco se incorporó en el código para intentar representar estos datos también en los marcadores. Se probaron dos alternativas:

- Una decodificación similar a la que ofrecía el código del artículo [36] recogido en el Anexo X *NmeaAisTextFieldParser*. Su funcionamiento es sencillo, el *parser* recibe un conjunto de bits que representan los caracteres del nombre del barco, extrae los bits en bloques de seis, y los convierte en valores ASCII para representar los caracteres legibles.
- Una decodificación de los bits correspondientes al nombre de la embarcación que sigue un proceso similar al explicado anteriormente, difiriendo en que, en este caso, los valores decodificados se traducen a formato ASCII mediante una tabla de mapeo. Esta tabla asigna un carácter a cada combinación de seis bits, cubriendo valores del cero al sesenta y seis, donde cada grupo de seis bits se convierte en su correspondiente carácter ASCII.

```
private static readonly char[] SixBitToCharTable = {
    '@', 'A', 'B', 'C', 'D', 'E', 'F', 'F', '4', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
    'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
    '!', '!', 'A', '$', '%', 'W', '\\', '(', '!', '*', '+', '-', '.', '/',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?'
};
```

Sin embargo, tras varios intentos, esta idea fue descartada debido a la complejidad que implicaría modificar el proyecto para representar estos tipos de datos. La mayoría de las librerías disponibles en línea para la decodificación de mensajes AIS están compuestas por una gran cantidad de *scripts*, que, en conjunto, permiten mostrar toda la información deseada. Para alcanzar el nivel de aplicaciones populares

como *Marine Traffic* o *VesselFinder* sería necesario un periodo de tiempo considerable para desarrollar las librerías necesarias y garantizar su funcionamiento adecuado.

3.6 Mejoras implementadas

El objetivo principal del proyecto fue integrar la información AIS recibida para la creación de los marcadores en la aplicación en tiempo real. Una vez logrado el objetivo principal, como mejoras adicionales, se plantearon varias optimizaciones para enriquecer la funcionalidad del sistema, entre las cuales se incluye la actualización dinámica de los marcadores en tiempo real, garantizando que siempre se refleje la información más precisa y actualizada. Las mejoras de este código son:

3.6.1 Actualización dinámica de los marcadores

Se logra una actualización de los marcadores que sustituye los anteriores y genere uno nuevo con los datos subsiguientes recibidos del mismo barco, esta mejora se ve reflejada dentro del método *Update()* explicado en la sección 3.7.

Además, se establece un mínimo de precisión de treinta metros para los marcadores. De esta manera el funcionamiento de la aplicación asegura una precisión de los marcadores mínima y por tanto un error máximo. Este aspecto es clave para la medición del error de la aplicación. Esta línea de código es añadida como condición dentro del método *Update ()*.

```
...  
if (geospatialPose.HorizontalAccuracy <= 30 && geospatialPose.VerticalAccuracy <= 30)  
{  
...  
}
```

3.6.2 Rediseño visual del marcador

Se crea un *prefab* distinto donde se vea más claro los datos que se quieren mostrar de las embarcaciones (Figura 3-12).



Figura 3-12 Nuevo marcador (Elaboración propia)

Además, Con el objetivo de facilitar la interpretación visual de los datos, se establece un sistema de coloreado que muestra la distancia de los marcadores en color rojo cuando se encuentran a menos de mil ochocientos metros y de color verde cuando la distancia es mayor (Figura 3-13). De esta forma, es más sencillo distinguir qué barcos se encuentran en un rango de menos de una milla náutica de nuestra posición, información crucial para evitar abordajes.

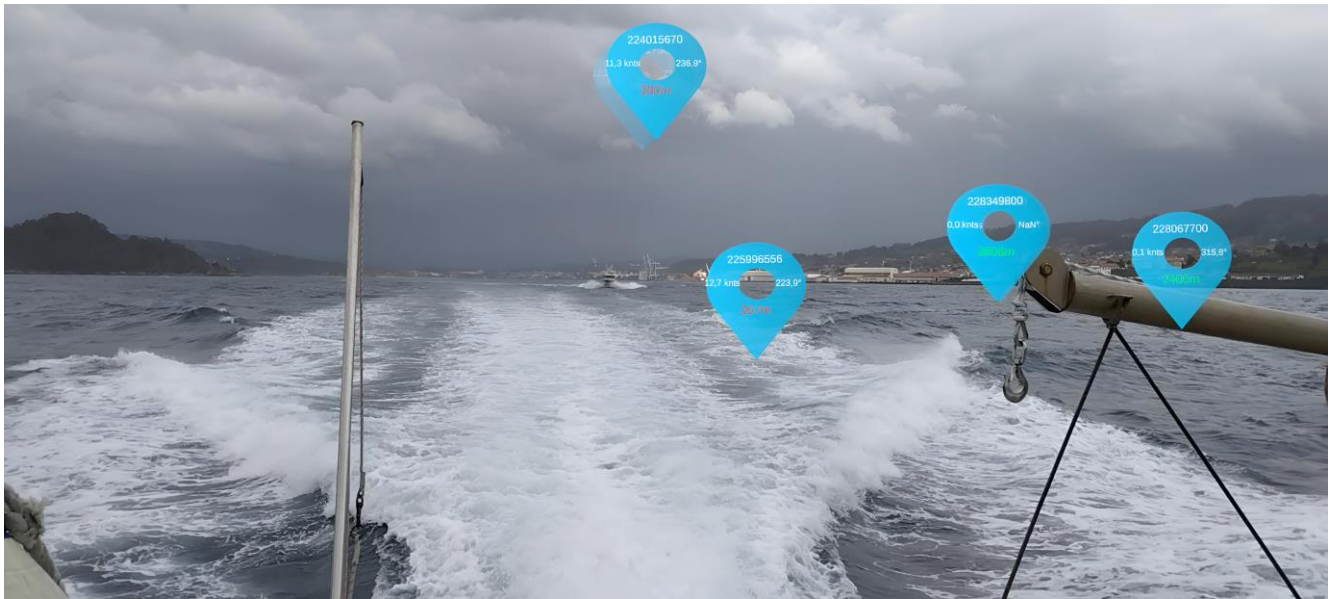


Figura 3-13 Captura de pantalla en la que se puede observar como el color cambia según la distancia

Esta mejora en la aplicación se implementa como un fragmento dentro del método *Update()*, utilizando una estructura condicional con un *if* y un *else*. Como se puede observar en el fragmento del código, cuando la distancia es menor a mil ochocientos metros solo el primer campo (que representa el color rojo) recibe su valor completo, mientras que los otros dos campos (verde y azul) permanecen inactivos. Por otro lado, si la distancia es superior a mil ochocientos metros, es el campo del color verde el que recibe el valor completo.

```
{
    distanceText.text = ((int)distance).ToString() + "m";
    if (distance < 1800f)
    {
        distanceText.color = new Color(1f, 0f, 0f, distanceText.color.a);
    }
    else
    {
        distanceText.color = new Color(0f, 1f, 0f, distanceText.color.a);
    }
}
```

3.6.3 Comportamiento de los marcadores en pantalla

Se elimina del código la funcionalidad que impedía que los marcadores desaparecieran de la pantalla. Esta modificación se realizó con el fin de mejorar la experiencia del usuario, ya que, al mantener los marcadores visibles de manera constante en la pantalla, la aplicación podría generar confusión al usuario, especialmente cuando los datos AIS variaban.

El sistema también gestiona el ciclo de vida de los marcadores, con un tiempo de expiración configurado. Pasado este tiempo, los marcadores se desvanecen suavemente utilizando un efecto de desvanecimiento gradual, lo que mejora la experiencia visual al eliminar los elementos obsoletos de la escena, además de dar una noción de actualidad de los marcadores. Este comportamiento se controla mediante un sistema de actualización que verifica periódicamente la antigüedad de los marcadores y activa su eliminación cuando es necesario (Figura 3-14).



Figura 3-14 Captura que muestra cómo los marcadores se van desvaneciendo a medida que transcurre el tiempo

El método creado para agregar esta mejora es del *FadeOut*, que actúa como una corutina que realiza una transición visual de desvanecimiento del objeto *marker*, reduciendo progresivamente la opacidad de su imagen y sus componentes de texto. Utiliza un valor *fadeSpeed* para controlar la velocidad de desvanecimiento. Primero, obtiene el componente *Image* y los componentes *TMP_Text* del objeto. Luego, guarda los colores originales de la imagen y los textos. En el ciclo *while*, la opacidad de la imagen y de los textos disminuye lentamente, manteniéndose por encima de un valor mínimo de opacidad (*minAlpha*), aplicando el cambio en la componente *alfa* de cada uno. La corutina espera un cuadro en cada iteración (*yield return null*) hasta que la opacidad alcanza el valor mínimo deseado.

```
private IEnumerator FadeOut(GameObject marker, float fadeSpeed)
{
    Image img = marker.GetComponentInChildren<Image>();
    TMP_Text[] textComponents = marker.GetComponentInChildren<TMP_Text>();
    Color originalImageColor = img.color;
    Color[] originalTextColors = textComponents.Select(t => t.color).ToArray();
    float minAlpha = 0.3f;
    while (img.color.a > minAlpha)
    {
        float alpha = Mathf.Max(minAlpha, img.color.a - fadeSpeed * Time.deltaTime);
        img.color = new Color(originalImageColor.r, originalImageColor.g,
originalImageColor.b, alpha);
        foreach (var text in textComponents)
        {
            text.color = new Color(text.color.r, text.color.g, text.color.b, alpha);
        }
        yield return null;
    }
}
```

3.6.4 Simplificación estructural del código

Dado que el objetivo principal era eliminar la intervención del usuario en la creación de los marcadores, se suprime por completo el método *PlaceObjects*. En su lugar, se integra su funcionalidad directamente en el método *Update* (), lo que permitió optimizar el código y hacerlo más eficiente. Esta modificación simplificó la estructura del programa al reducir la cantidad de métodos independientes, mejorando así la claridad y el rendimiento general de la aplicación.

3.6.5 Sistema de calibración y orientación del dispositivo

El código también incorpora un proceso de calibración que ajusta la orientación del dispositivo utilizando la brújula interna. Este proceso, realizado en un hilo separado, permite al usuario calibrar el dispositivo de manera interactiva, garantizando una orientación precisa para la colocación de los marcadores en el espacio real. Durante la calibración, se proporcionan instrucciones visuales y un indicador de progreso para facilitar la experiencia del usuario (Figura 3-15).

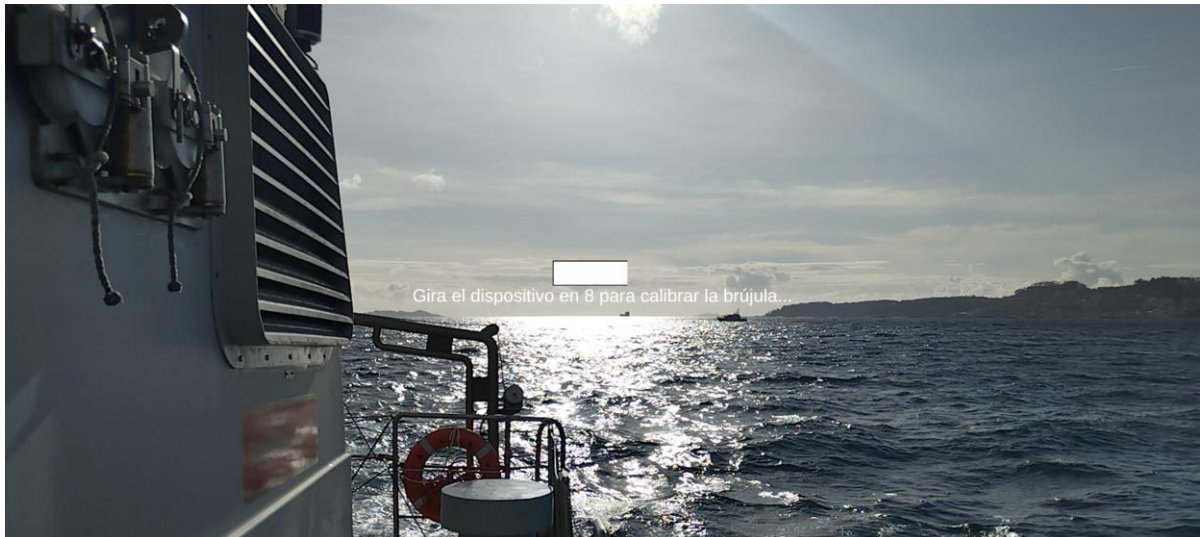


Figura 3-15 Texto de instrucciones para calibrar la brújula, con barra que se va completando de izquierda a derecha a medida que se realiza la calibración (Elaboración propia)



Figura 3-16 Texto que indica que se ha realizado la calibración correctamente (Elaboración propia)

Para incorporar esta mejora se creó el método *StartCalibration* que gestiona el proceso de calibración de la brújula del dispositivo, comenzando con la verificación de si la calibración ya se ha completado. Si no, activa el proceso de calibración y actualiza la interfaz de usuario mediante *UnityMainThreadDispatcher.Instance().Enqueue()*, mostrando instrucciones y un indicador de proceso. Durante la calibración, el método monitorea los cambios en el encabezado de la brújula con *Input.compass.trueHeading* y calcula el cambio de dirección (*deltaHeading*). Si el cambio supera un umbral (*calibrationTreshold*), incrementa el contador de rotaciones y actualiza el indicador de progreso. Una vez alcanzado el número máximo de rotaciones, se llama a *FinishCalibration()* para finalizar y mostrar el mensaje que se muestra en la Figura 3-16. Si no se completa, se solicita la recalibración con

RequestRecalibration(). El bucle de calibración se ejecuta con pausas controladas mediante *Thread.Sleep(100)* para dar tiempo a las rotaciones.

```
private void StartCalibration()
{
    if (!calibrationCompleted)
    {
        isCalibrating = true;
        rotationCount = 0;
        previousHeading = Input.compass.trueHeading;

        UnityMainThreadDispatcher.Instance().Enqueue(() => {
            if (calibrationInstructionsText != null)
            {
                calibrationInstructionsText.text = "Gira el dispositivo en 8 para calibrar
la brújula...";
                calibrationInstructionsText.gameObject.SetActive(true);
            }

            if (calibrationIndicatorImage != null)
            {
                calibrationIndicatorImage.gameObject.SetActive(true);
                calibrationIndicatorImage.fillAmount = 0f;
            }
        });
        while (isCalibrating && rotationCount < calibrationMaxRotations)
        {
            float currentHeading = Input.compass.trueHeading;
            float deltaHeading = Mathf.Abs(currentHeading - previousHeading);

            if (deltaHeading > calibrationThreshold)
            {
                rotationCount++;
                previousHeading = currentHeading;
                float progressValue = rotationCount / (float)calibrationMaxRotations;
                UnityMainThreadDispatcher.Instance().Enqueue(() => {
                    if (calibrationIndicatorImage != null)
                    {
                        calibrationIndicatorImage.fillAmount = progressValue;
                    }
                });
            }
            if (rotationCount >= calibrationMaxRotations)
            {
                UnityMainThreadDispatcher.Instance().Enqueue(() => {
                    FinishCalibration();
                });
                break;
            }
            Thread.Sleep(100);
        }
        if (rotationCount < calibrationMaxRotations)
        {
            UnityMainThreadDispatcher.Instance().Enqueue(() => {
                RequestRecalibration();
            });
        }
    }
}
```

3.7 Código de *VPSManager* definitivo

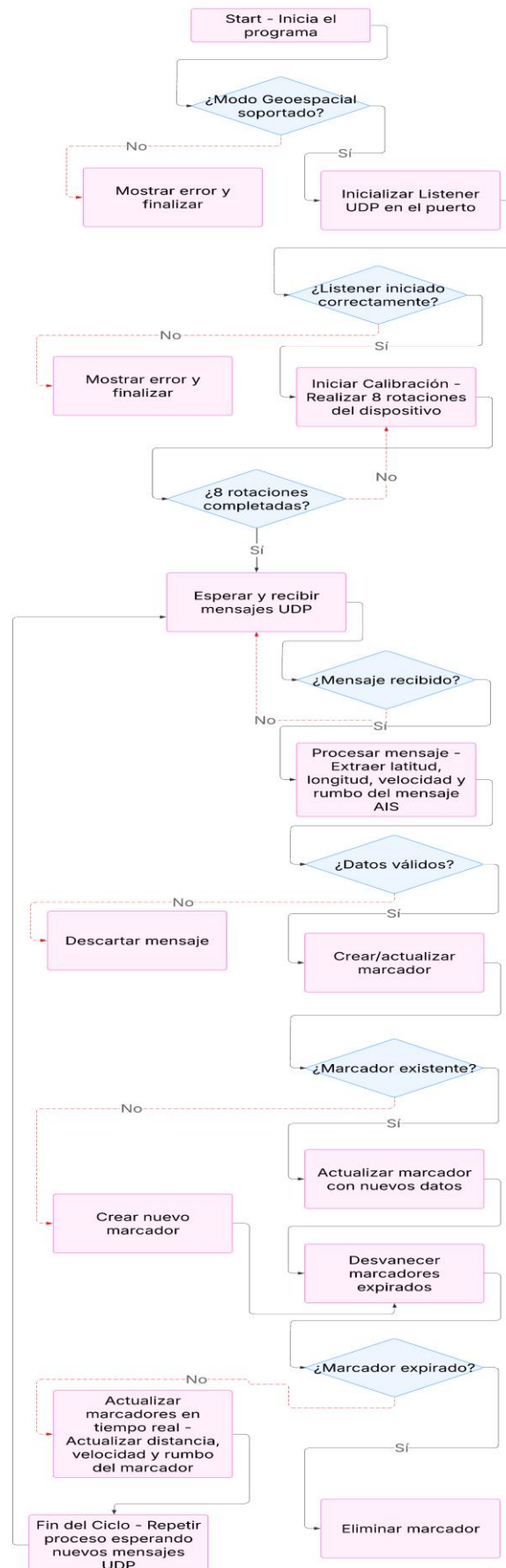


Figura 3-17 Diagrama de flujo código *VPSManager* (Elaboración propia)

En la Figura 3-17 se presenta un diagrama de flujo que explica el funcionamiento del código completo detallado en el Anexo V *VPSManager* definitivo. Este código, basado en los métodos previamente descritos en este capítulo, tiene como objetivo decodificar los datos AIS recibidos por el receptor RTL.SDR. Una vez decodificados, el sistema crea un marcador para representar la información obtenida, como la ubicación y características de las embarcaciones. Además, el código es capaz de identificar si los nuevos datos corresponden a un marcador anterior y crea uno nuevo con los datos actualizados, garantizando que la visualización siempre refleje la información más reciente, destruyendo el marcador antiguo.

La gestión de los mensajes UDP, que son recibidos y decodificados de forma asíncrona, es otra parte crucial del sistema. El programa escucha continuamente los mensajes UDP enviados por el receptor RTL.SDR, a través de un *socket* UDP.

Tras establecer el *socket* UDP en el puerto 9000 con la función *UdpClient (udpPort)*, el método *UDPListen* (mostrado a continuación) entra en un ciclo continuo donde se mantiene escuchando los paquetes de datos enviados desde AIS cácher. Para recibir los datos, se utiliza la función *udpClient.Receive (ref remoteEndPoint)*, que captura los paquetes entrantes y los almacena en el arreglo *.data*. Posteriormente, los datos recibidos son convertidos de bytes a una cadena UTF-8 (*Unicode Transformation Format – 8 bits*) mediante la función *Encoding.UTF8.GetString(data)*. Esta cadena se agrega a la cola de mensajes utilizando la función *udpMessages.Enqueue (message)* para su posterior procesamiento. Esto permite que el sistema mantenga una visualización dinámica y precisa, adaptándose a los cambios en tiempo real de la información AIS.

```
private void UDPListen()
{
    UdpClient udpClient = new UdpClient(udpPort);
    IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, udpPort);
    while (udpListenerRunning)
    {
        try
        {
            byte[] data = udpClient.Receive(ref remoteEndPoint);
            string message = Encoding.UTF8.GetString(data);
            udpMessages.Enqueue(message);
        }
        catch (SocketException ex)
        {
            Debug.LogError("Socket exception: " + ex.Message);
        }
        catch (Exception ex)
        {
            Debug.LogError("Unexpected exception: " + ex.Message);
        }
    }
    udpClient.Close();
}
```

En el código descrito, la decodificación de los datos AIS es un proceso fundamental que se lleva a cabo mediante la interpretación de las sentencias NMEA, explicadas en el capítulo dos, que son las que contienen información de las embarcaciones. Los datos recibidos están en formato texto, que luego se divide en fragmentos específicos para extraer la información relevante. El código realiza la conversión de los datos AIS, decodificando las diferentes secciones que corresponden a parámetros como la latitud, longitud, velocidad o rumbo. Este proceso de conversión se realiza utilizando técnicas como la representación de complemento a dos, que permite interpretar correctamente valores numéricos negativos, y la conversión de caracteres de seis bits en texto ASCII legible.

El método *DecodeAisPosition* procesa una sentencia NMEA para extraer y decodificar información relacionada con la posición de una embarcación. Primero, la sentencia se divide en partes mediante la función *Split ()*, y se valida si tiene la longitud adecuada. A continuación, se extrae la carga útil con *ConvertPayloadToBinary()*, que convierte los datos a su representación binaria. La función *Convert.ToInt32 ()* se utiliza para interpretar el tipo de mensaje AIS, y se verifica si está en un rango

válido (1-3). Luego, se extraen los bits correspondientes al MMSI, latitud, longitud, velocidad y rumbo utilizando *Substring()*, los cuales se interpretan y convierten en valores numéricos mediante funciones como *TwosComplementToInt()* y *DecodeSpeed()*. El método devuelve un objeto *AisPosition* con la información decodificada de la embarcación, que incluye latitud, longitud, MMSI, velocidad y rumbo.

```
private AisPosition DecodeAisPosition(string nmeaSentence)
{
    string[] parts = nmeaSentence.Split(',');
    if (parts.Length < 6)
    {
        return new AisPosition { Latitude = double.NaN, Longitude = double.NaN, ShipName = "No name", MMSI = "Unknown" };
    }
    string payload = parts[5];
    string bits = ConvertPayloadToBinary(payload);
    int messageType = Convert.ToInt32(bits.Substring(0, 6), 2);
    if (messageType < 1 || messageType > 3)
    {
        return new AisPosition { Latitude = double.NaN, Longitude = double.NaN, ShipName = "No name", MMSI = "Unknown" };
    }
    string mmsiBits = bits.Substring(8, 30);
    int mmsiInt = Convert.ToInt32(mmsiBits, 2);
    string mmsi = mmsiInt.ToString();
    string lonBits = bits.Substring(61, 28);
    string latBits = bits.Substring(89, 27);
    int rawLon = TwosComplementToInt(lonBits);
    int rawLat = TwosComplementToInt(latBits);
    double longitude = rawLon / 600000.0;
    double latitude = rawLat / 600000.0;
    string sogBits = bits.Substring(50, 10);
    string cogBits = bits.Substring(116, 12);
    double speed = DecodeSpeed(sogBits);
    double course = DecodeCourse(cogBits);
    string shipName = DecodeShipName(bits);
    return new AisPosition { Latitude = latitude, Longitude = longitude, ShipName = shipName, MMSI = mmsi, Speed = speed, Course = course };
}
```

- *ConvertPayloadToBinary*: Método que convierte una carga útil en formato ASCII a una cadena binaria, donde cada carácter se procesa y se representa como un valor binario de seis bits, asegurando que todos los valores tengan la longitud adecuada.

```
private string ConvertPayloadToBinary(string payload)
{
    StringBuilder bitString = new StringBuilder();
    foreach (char c in payload)
    {
        int sixBit = c - 48;
        if (sixBit > 40) sixBit -= 8;
        bitString.Append(Convert.ToString(sixBit, 2).PadLeft(6, '0'));
    }
    return bitString.ToString();
}
```

- *TwosComplementToInt*: Este método convierte una cadena de bits en un número entero, interpretando correctamente los valores en complemento a dos (una representación binaria de números negativos) para representar número negativos cuando corresponda.

```
private int TwosComplementToInt(string bits)
{
    if (bits[0] == '0')
        return Convert.ToInt32(bits, 2);
    else
    {
        string inverted = new string(bits.Select(b => b == '0' ? '1' : '0').ToArray());
        int intValue = Convert.ToInt32(inverted, 2) + 1;
        return -intValue;
    }
}
```

- *DecodeSpeed*: Es el método que convierte una cadena de bits que representa la velocidad de una embarcación en un valor numérico. Retornando NaN si el valor es inválido, o dividiendo el valor numérico obtenido por diez para obtener la velocidad en nudos.

```
private double DecodeSpeed(string sogBits)
{
    int sogRaw = Convert.ToInt32(sogBits, 2);
    if (sogRaw == 1023)
    {
        return double.NaN;
    }
    return sogRaw / 10.0;
}
```

- *DecodeCourse*: Es el método que convierte una cadena de bits que representa el rumbo de una embarcación en un valor numérico. Retornando NaN si el valor es inválido, o dividiendo el valor numérico obtenido por diez para obtener el rumbo en grados.

```
private double DecodeCourse(string cogBits)
{
    int cogRaw = Convert.ToInt32(cogBits, 2);
    if (cogRaw == 3600)
    {
        return double.NaN;
    }
    return cogRaw / 10.0;
}
```

- *AisPosition*: Es la estructura que define el tipo de dato que almacena la información de posición de la embarcación, además de su MMSI, velocidad, rumbo y nombre.

```
public struct AisPosition
{
    public double Latitude;
    public double Longitude;
    public string ShipName;
    public string MMSI;
    public double Speed;
    public double Course;
}
```

Una vez que los datos han sido extraídos y decodificados, el sistema evalúa si se corresponden a un marcador existente o a uno nuevo. Evaluación tras la cual ocurre lo ya explicado en esta sección. Esto asegura que la visualización se mantenga actualizada y precisa en tiempo real, proporcionando a los usuarios información relevante y dinámica de las embarcaciones que se encuentran en la zona.

Esta función de la aplicación es gestionada por el método *Update ()*, que realiza un procesamiento continuo de los datos AIS recibidos, siempre que la calibración esté completa. Primero, intenta extraer las sentencias NMEA de la cola *udpMessages* utilizando el método *TryDequeue ()*. Si el estado de *Earth TrackingState* se encuentra en *Tracking*, verifica si la precisión geoespacial es adecuada. Si las condiciones son válidas, decodifica la posición AIS, llamando al método *DecodeAISPosition*.

A continuación, el sistema evalúa si las coordenadas geoespaciales son válidas. Si los valores son válidos, el proceso continúa verificando si ya existe un marcador para la embarcación, buscando en la lista de *markers* el MMSI de la nueva posición. Si se encuentra un marcador con el mismo MMSI, el marcador anterior se destruye y se elimina de la lista *markers*. Esto asegura que solo haya un marcador activo por cada embarcación en los alrededores. Para colocar un nuevo marcador, se utiliza *ARAnchor*, que se genera en función de las coordenadas geoespaciales de la embarcación. Esto permite situar el marcador en el espacio 3D de la escena.

El marcador se instancia a través de un *prefab* o modelo y se actualizan los valores correspondientes a la embarcación, tales como el MMSI, la velocidad (SOG) y el rumbo (COG) en los textos de la interfaz

del usuario, además se inicia la corutina con el método *FadeOut()*. Si las condiciones de precisión no se cumplen o el *EarthTracking* no está activo, el método *Update ()* se invoca de nuevo después de un breve retardo, utilizando *Invoke()* para intentar procesar la siguiente sentencia NMEA.

Este proceso de actualización continua y reemplazo de los marcadores permite que la interfaz se mantenga actualizada en tiempo real, proporcionando al usuario una visualización precisa de las embarcaciones y sus movimientos en el entorno.

A continuación, se puede ver la primera parte del método *Update ()* recientemente explicado.

```
void Update()
{
    if (calibrationCompleted)
    {
        while (udpMessages.TryDequeue(out string nmeaSentence))
        {
            if (earthManager.EarthTrackingState == TrackingState.Tracking)
            {
                var geospatialPose = earthManager.CameraGeospatialPose;
                if (geospatialPose.HorizontalAccuracy <= 30 && geospatialPose.VerticalAccuracy
<= 30)
                {
                    AisPosition pos = DecodeAisPosition(nmeaSentence);
                    if (double.IsNaN(pos.Latitude) || double.IsNaN(pos.Longitude))
                    {
                        continue;
                    }
                    Marker existingMarker = markers.FirstOrDefault(m =>
m.Object.transform.Find("Image/Name").GetComponent<TMP_Text>().text == pos.MMSI);
                    if (existingMarker.Object != null)
                    {
                        Destroy(existingMarker.Object);
                        markers.Remove(existingMarker);
                    }
                    Quaternion rotation = earthManager.CameraGeospatialPose.EunRotation;
                    var anchor = ARAnchorManagerExtensions.AddAnchor(aRArAnchorManager,
pos.Latitude, pos.Longitude, fixedAltitude, rotation);
                    var markerObj = Instantiate(ObjectPrefab);
                    markerObj.transform.Find("Image/Name").GetComponent<TMP_Text>().text =
pos.MMSI;
                    markerObj.transform.Find("Image/Distance").GetComponent<TMP_Text>().text =
"0m";
                    markerObj.transform.Find("Image/SOG").GetComponent<TMP_Text>().text =
pos.Speed.ToString("F1") + " knts";
                    markerObj.transform.Find("Image/COG").GetComponent<TMP_Text>().text =
pos.Course.ToString("F1") + "°";
                    Marker newMarker = new Marker { Anchor = anchor, Object = markerObj,
Position = pos }; // Almacenar la posición AIS
                    markers.Add(newMarker);
                    StartCoroutine(FadeOut(markerObj, 0.05f));
                }
                else
                {
                    Invoke("Update", 2.0f);
                }
            }
            else if (earthManager.EarthTrackingState == TrackingState.None)
            {
                Invoke("Update", 5.0f);
            }
        }
    }
}
```

A continuación, se puede observar cómo finaliza el fragmento final del método *Update ()*. Este fragmento del código recorre todos los marcadores activos en la lista *markers* y actualiza su visualización en la interfaz del usuario según su posición geoespacial. Para cada marcador, calcula su posición en la pantalla a partir de su anchor en el mundo y la convierte en coordenadas de pantalla, actualizando la posición de la imagen asociada y mostrando los textos correspondientes si el marcador es visible en la pantalla, comprobando si su coordenada Z es positiva. Además, calcula la distancia entre el marcador y la cámara, y ajusta el color del texto en función de esta. También actualiza los valores de la velocidad (SOG) y el rumbo (COG) del marcador, si estos están presentes en el objeto.

```

foreach (var marker in markers)
{
    var img = marker.Object.GetComponentInChildren<Image>();
    var textComponents = marker.Object.GetComponentInChildren<TMP_Text>();

    Vector3 worldPosition = marker.Anchor.transform.position + offset;
    Vector3 screenPoint = Camera.main.WorldToScreenPoint(worldPosition);
    if (screenPoint.z > 0)
    {
        img.transform.position = new Vector2(screenPoint.x, screenPoint.y);
        img.enabled = true;
        foreach (var text in textComponents)
        {
            text.enabled = true;
        }
    }
    else
    {
        img.enabled = false;
        foreach (var text in textComponents)
        {
            text.enabled = false;
        }
    }
    float distance = Vector3.Distance(marker.Anchor.transform.position,
    Camera.main.transform.position);
    TMP_Text distanceText =
    marker.Object.transform.Find("Image/Distance")?.GetComponent<TMP_Text>();
    if (distanceText != null)
    {
        distanceText.text = ((int)distance).ToString() + "m";
        if (distance < 1800f)
        {
            distanceText.color = new Color(1f, 0f, 0f, distanceText.color.a);
        }
        else
        {
            distanceText.color = new Color(0f, 1f, 0f, distanceText.color.a);
        }

        if (marker.Object.transform.Find("Image/SOG") != null)
        {
            marker.Object.transform.Find("Image/SOG").GetComponent<TMP_Text>().text =
            marker.Position.Speed.ToString("F1") + " knts";
        }

        if (marker.Object.transform.Find("Image/COG") != null)
        {
            marker.Object.transform.Find("Image/COG").GetComponent<TMP_Text>().text =
            marker.Position.Course.ToString("F1") + "°";
        }
    }
}
}
}

```

Este proceso de decodificación y recepción se realiza de forma asíncrona, lo que permite que el sistema reciba y procese múltiples mensajes AIS sin interrumpir el funcionamiento del resto del sistema. Gracias a esta estructura, el código es capaz de manejar una gran cantidad de datos de forma rápida y precisa, lo que resulta esencial para mantener una representación actualizada de las embarcaciones en todo momento.

En resumen, el código no solo decodifica y procesa los datos AIS de manera eficiente, sino que también gestiona la visualización de los marcadores en un entorno AR, manteniendo la sincronización con los cambios en los datos en tiempo real y optimizando el rendimiento a través de la eliminación de marcadores antiguos y la gestión de orientación del dispositivo.

4 PRUEBAS Y RESULTADOS

En este capítulo se muestra el funcionamiento de la aplicación desarrollada y se evalúan cualitativa y cuantitativamente sus prestaciones. Se evidencia que la aplicación es capaz de asociar marcadores a la posición real de las embarcaciones, fallando en cierta medida en cuanto a precisión. No obstante, gracias a estos marcadores es posible visualizar de forma inmediata información relevante sobre cada embarcación directamente desde la pantalla del dispositivo.

Para la presentación de los resultados se ha realizado un experimento embarcando en las lanchas de instrucción de la Escuela Naval Militar, mostrando la funcionalidad de la aplicación representando los respectivos datos AIS de cada una de las lanchas.

4.1 Presentación de los resultados

Para facilitar la comprensión de los resultados mostrados a continuación es importante conocer que las figuras que se van a ver a continuación son capturas de videos tomados desde la lancha GM Rull, y que las lanchas que se pueden observar responden a la relación nombre-MMSI mostrada en la siguiente tabla:

Nombre	MMSI
GM Barrutia	224015670
GM Chereguini	224201730
GM Salas	224072340
GM Herrero	225996556

Tabla 2 Relación nombres de las lanchas con su MMSI (Elaboración propia)

4.1.1 Prueba empírica

Para realizar una medición fiable del funcionamiento de la aplicación, en un ambiente donde la aplicación no sea capaz de utilizar objetos de los alrededores ni los servicios de mapeado de Google como *Streetview*, se embarcó en la lancha de instrucción GM Rull y se utilizaron los datos AIS provenientes del resto de lanchas. Debido al disminuido tráfico marítimo de embarcaciones de gran envergadura a lo largo de la ría, se pudo realizar una valoración más precisa del funcionamiento, puesto que solo se tenían los marcadores que se creaban por las lanchas de instrucción.

A continuación, se va a proceder a un análisis de la prueba realizada mediante una secuencia de imágenes que muestran el funcionamiento de la aplicación. Esta secuencia de imágenes es una

recopilación de capturas de pantalla obtenidas del vídeo que se puede encontrar en el Anexo XI Vídeo, donde se puede observar con mayor claridad el funcionamiento real de la versión definitiva de la aplicación.

En la Figura 4-1 se observa cómo se crean tres marcadores que pertenecen, de izquierda a derecha, a la GM Barrutia, la GM Chereguini y a la GM Salas, dada la relación que se menciona anteriormente en este mismo capítulo en la Tabla 2. Cada marcador muestra el MMSI, rumbo, velocidad y la distancia a la que se encuentra cada contacto. Los tres marcadores presentan un color rojo en su distancia al encontrarse a una distancia menor de mil ochocientos metros. La imagen muestra en su fondo la bocana de la ría de Pontevedra, que se encuentra al Oeste de la posición desde la que se toma el vídeo. Teniendo esta posición en cuenta, se puede observar como la lancha con la silueta más clara, la GM Salas, bajo el marcador colocado a la derecha de los otros dos, aparece con la información de ir navegando con un rumbo prácticamente Sur, información que coincide con el rumbo real de la GM Salas.

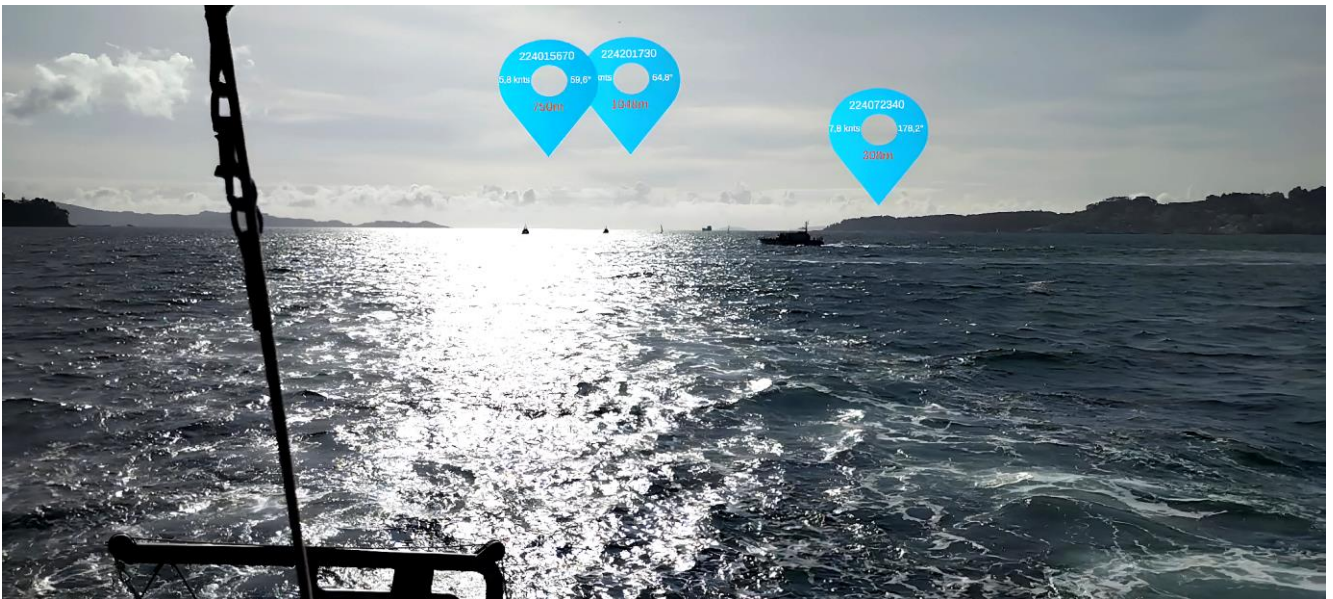


Figura 4-1 Primera parte prueba

En la Figura 4-2 se puede observar como la GM Salas con rumbo Sur comienza a cortar la proa de las otras dos lanchas de instrucción y a medida que esta avanza el marcador también lo hace. Por esta razón es que ahora se distingue menos distancia entre los marcadores, ya que el marcador correspondiente a la GM Salas está siguiendo el movimiento de su correspondiente. También se puede observar como la distancia, rumbo y velocidad de cada marcador ha cambiado al actualizarse.

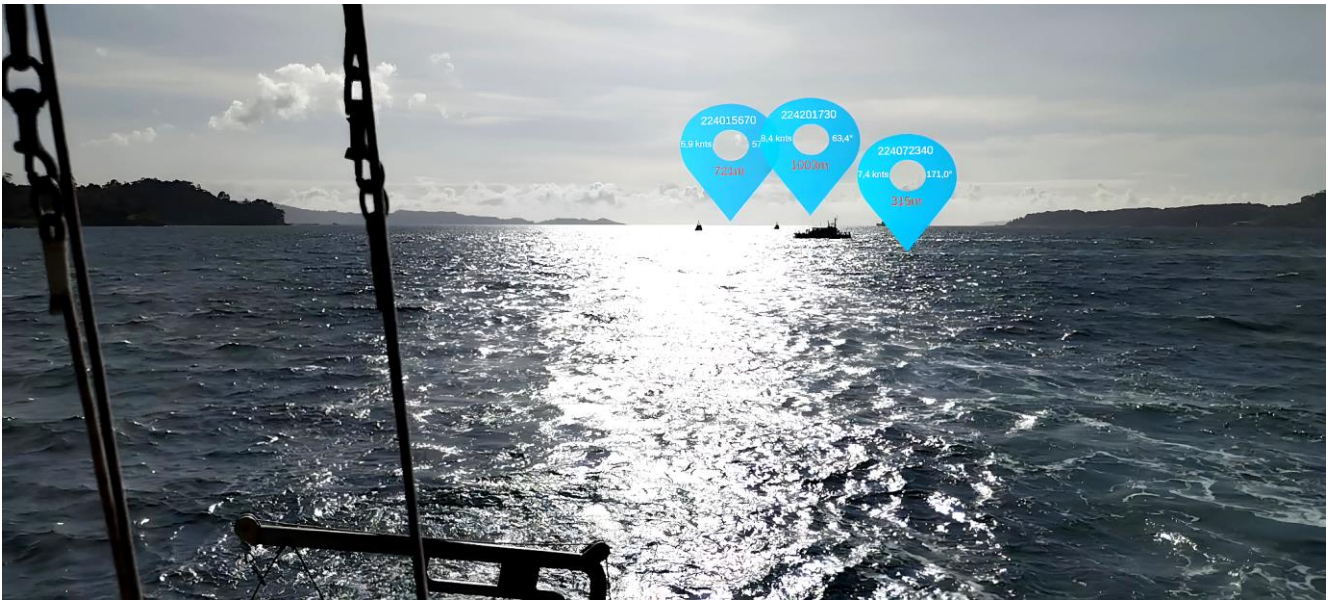


Figura 4-2 Segunda parte prueba

Continuando con la escena, en la Figura 4-3 se muestra cómo, mientras la GM Salas avanza, el marcador, que se actualiza dependiendo de la frecuencia con la que la embarcación mande sus datos AIS, se va quedando retrasado respecto a la posición real de la GM Salas. Pero, para que esto no suponga una posible confusión de la verdadera posición de una embarcación, los marcadores van aumentando su transparencia como señal de que el marcador no es reciente. Esto mismo ocurre en este momento, puesto que la posición de los marcadores es la misma que ocupaban en la Figura 4-2.



Figura 4-3 Tercera parte prueba

En la Figura 4-4, la GM Salas, desde la posición del usuario, se encuentra navegando entre las otras dos lanchas de instrucción y los marcadores ocupan la misma posición al actualizarse. Debido a que ocupan la misma localización no se distinguen con claridad los datos de cada marcador, pudiéndose distinguir la posición del marcador de la GM Salas únicamente por la distancia, más cercana que la de las otras dos lanchas de instrucción.



Figura 4-4 Cuarta parte prueba

Una vez la GM Salas corta la proa de las otras dos lanchas, comienza a cambiar su rumbo, véase la Figura 4-5. Los otros dos marcadores, pese a que en las capturas de pantalla no muestran un cambio significativo en la actualización de su posición y datos, si se actualizan, mostrando una apariencia menos translúcida cada vez que las lanchas de instrucción mandan nuevos datos AIS. Un aspecto crucial de los marcadores es que, aunque la información que ofrecen es solo actualizada cuando se reciben nuevos mensajes AIS, su distancia a la que se encuentran del usuario sí es actualizada y calculada de forma constante. Esta característica se puede observar de forma clara en el vídeo del Anexo XI Vídeo, pero es observable también en la Figura 4-6, continuación inmediata de la Figura 4-5, y en la que se ve como la GM Barrutia y la GM Chereguini se encuentran a menor distancia.



Figura 4-5 Quinta parte prueba

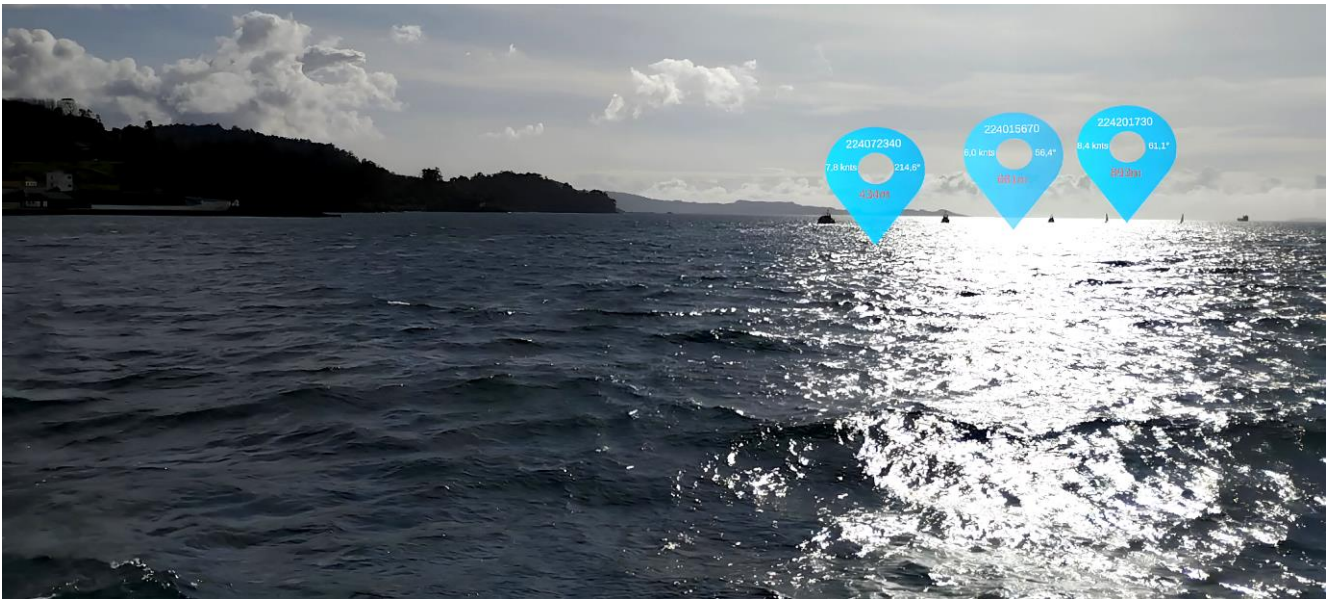


Figura 4-6 Sexta parte prueba

A medida que transcurre la escena, el usuario desde la GM Rull también modifica su posición y la lancha adopta un rumbo prácticamente inverso al que llevan las lanchas que se encuentran en la Figura 4-7. Figura en la que, además, se puede observar cómo los marcadores, que inicialmente se colocaban a la derecha respecto a la posición del usuario, pasan a situarse a la izquierda, alineándose mejor con la ubicación real de las lanchas de las que se reciben los datos. Esto ocurre debido a que la localización de los marcadores depende del estado de posicionamiento del dispositivo y de los ajustes del mismo realizados por *ARCore* respecto al entorno que compone la imagen que se obtiene del dispositivo. Esto conlleva que a veces se observe una diferencia en la precisión y el error de la aplicación.



Figura 4-7 Séptima parte prueba

Por último, en la Figura 4-8 los marcadores muestran una posición respecto a las lanchas de instrucción más exacta, esto sucede tras un periodo de tiempo en el que la posición relativa del usuario respecto a las lanchas no varía demasiado, permitiendo a la aplicación tras las necesarias actualizaciones, corregir la posición de los marcadores, acercándose de manera más precisa a su localización real.



Figura 4-8 Octava parte prueba

4.1.2 Análisis de los resultados

Durante la misma prueba realizada a bordo de la GM además de validar el funcionamiento general de la aplicación, se llevó a cabo un análisis más detallado para medir la calidad de los resultados obtenidos. Es necesario destacar que la aplicación aún no ofrece un funcionamiento perfecto en todas las ocasiones que se emplea, pero sí que es capaz de cumplir sus funciones con excelentes resultados en la mayor parte de las ocasiones.

Respecto a los datos AIS de las embarcaciones no muestra signo de tener ningún problema en cuanto a la decodificación de estos y su correspondiente emplazamiento en los marcadores. Tanto el MMSI, la velocidad y el rumbo se corresponde en todo momento con el que tienen las embarcaciones en el mundo físico y se observa una correcta actualización de las dos últimas cada vez que llegan nuevos datos AIS. En cuanto al MMSI, se muestra correctamente sobre el marcador correspondiente, no se observa en ningún momento un incorrecto nombramiento de un marcador con el MMSI de otra unidad ni la variación de algún número por algún carácter no correspondiente.

El primer problema observable se presenta en la altura de los marcadores. La aplicación actual utiliza una altura predeterminada para todos los marcadores sin ofrecer ninguna opción para ajustar dinámicamente la altura en función de la embarcación. Como se puede ver en la Figura 4-9, la altura de colocación del marcador no corresponde con la altura real de la lancha de instrucción. Además, dado que la aplicación utiliza *ARCore*, su funcionamiento depende del rastreo del movimiento del dispositivo y, cuando es posible, de la detección de superficies planas para integrar marcadores virtuales en el mundo físico. Sin embargo, durante las pruebas en el entorno marino, se presentan dificultades importantes debido a la escasez de puntos de referencia visuales estables y la ausencia de superficies planas, elementos que resultan clave para el rastreo preciso y estable de los objetos en la escena.

En el entorno marino, la falta de infraestructura fija, como edificios o carreteras, limita la capacidad del sistema para alinear correctamente los objetos virtuales con el mundo físico. Esta inexactitud, sumada a la imprecisión en la geolocalización debido a factores como la variabilidad de la señal GPS en el mar, podría resultar en una colocación errónea de los marcadores. Como consecuencia, esto podría generar confusión al intentar identificar a qué embarcación corresponde cada marcador, dificultando la distinción entre contactos cercanos y comprometiendo la efectividad de la aplicación en situaciones críticas.

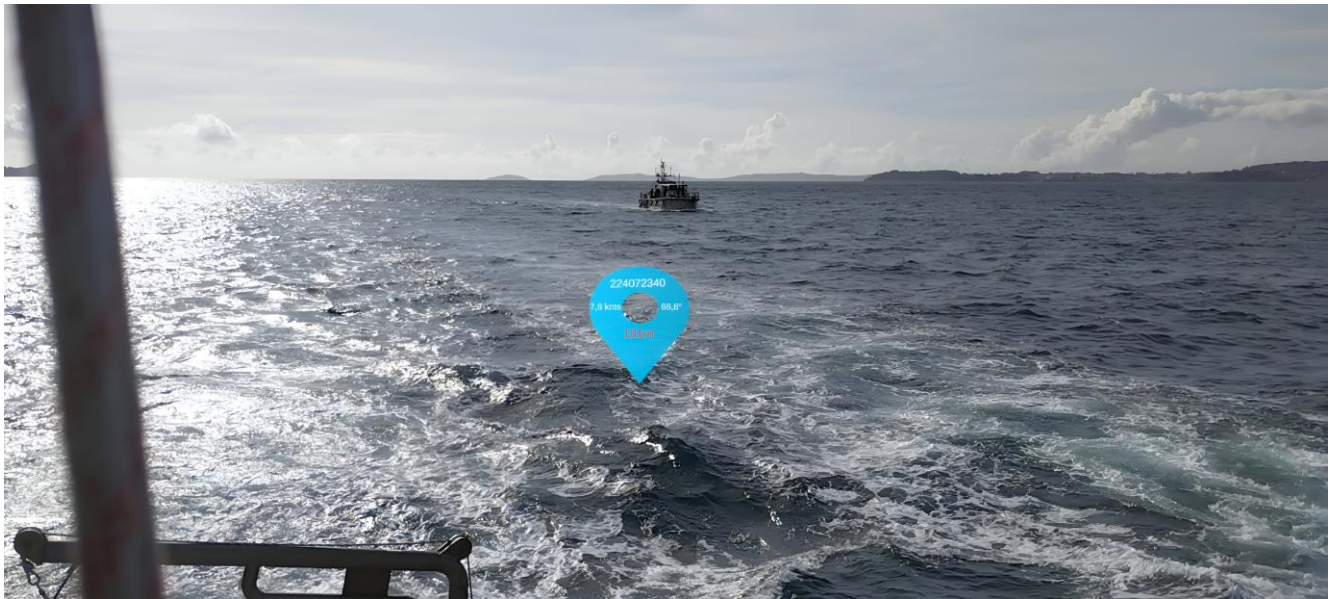


Figura 4-9 Error de altura observado en los marcadores

El segundo error observado durante la prueba en la mar está relacionado con la colocación de los marcadores respecto a la posición que les corresponde. Como se explicó en el capítulo tres, dentro del código del *Anexo V VPSManager* definitivo, el máximo error que se permite en el funcionamiento de la aplicación es de treinta metros en su posición GPS. Para comprobar sobre cómo se comporta este margen en condiciones reales, se realizó un análisis estimativo durante la prueba, utilizando como base la posición aparente de los marcadores en pantalla.

Antes de realizar el análisis, es importante señalar que en este trabajo no se ha realizado una evaluación específica del funcionamiento de la brújula del dispositivo. No obstante, su influencia en la colocación de los marcadores es crítica, ya que la orientación del dispositivo determina directamente la dirección en la que se proyectan los objetos en el entorno de realidad aumentada. Por ello, posibles errores de calibración o interferencias magnéticas podrían haber contribuido a parte de la desviación observada en la colocación de los marcadores.

En la Figura 4-10 se muestran dos lanchas de instrucción que siguen un mismo rumbo, así como sus marcadores correspondientes. Como se puede apreciar, los marcadores se encuentran desplazados hacia la izquierda respecto a la posición real de las lanchas. Para evaluar este desfase, se obtuvieron los siguientes datos durante la prueba:

- Rumbo GM Rull: 130°
- Rumbo GM Salas y GM Herrero: 040°
- Distancia a GM Salas: 219 metros
- Distancia a GM Herrero: 228 metros
- Distancia entre GM Salas y GM Herrero: 64 metros
- Demora a la GM Salas: 130°
- Demora a la GM Herrero: 145°
- Demora al marcador GM Herrero: 130°
- Demora al marcador GM Salas: 115°

Aprovechando la situación representada en la Figura 4-10, se estimó el ángulo de desviación que se aprecia en la colocación de los marcadores. Ambos marcadores tienen una desviación de aproximadamente quince grados (teniendo en cuenta que la demora a las lanchas puede no ser exacta y variar debido a un error instrumental). Desviación calculada utilizando los datos mencionados anteriormente:

- $|130^{\circ}-145^{\circ}|=15^{\circ} \rightarrow$ Desviación marcador GM Herrero
- $|130^{\circ}-115^{\circ}|=15^{\circ} \rightarrow$ Desviación marcador GM Salas



Figura 4-10 Imagen tomada para la medición de error de los marcadores

Aunque en el código se utiliza una condición de umbral de 30 metros como filtro de precisión para aceptar o no un marcador, este análisis no pretende verificar directamente esa condición, ya que no se está midiendo la precisión GPS del dato recibido. En su lugar, se realiza una estimación de la desviación angular, ya que es el parámetro más representativo del error percibido visualmente en pantalla. En situaciones donde la distancia a los contactos es considerable, el error angular tiene un mayor impacto en la representación que el error de posicionamiento GPS. Por ejemplo, un error de orientación de quince grados puede provocar una desviación visual más apreciable que un desplazamiento de veinte metros. No obstante, en distancias más cortas, incluso esos veinte metros pueden generar problemas de representación, especialmente si provocan que el marcador aparezca por completo en una dirección errónea.

Para contextualizar el impacto que puede tener un error angular en términos de desviación espacial aparente, se puede utilizar trigonometría básica. Aplicando la fórmula del desplazamiento angular:

Error distancia = distancia x seno (error angular)

- GM Salas: Error distancia = 219 metros x seno (15°) = 56,68 metros
- GM Herrero: Error distancia = 228 metros x seno (15°) = 59 metros

Estos valores no representan directamente una violación del umbral de precisión establecido en el código, sino que sirven para comprender el efecto que un pequeño error de orientación puede tener en la proyección visual, especialmente a largas distancias. Esta reflexión permite identificar mejor qué factores afectan a la colocación de los marcadores y valorar su impacto en la experiencia del usuario y en la fiabilidad del sistema.

Pese al error previamente expuesto, durante la misma situación, la Figura 4-11 muestra cómo los marcadores corrigen parcialmente su desviación respecto a la posición real de las lanchas de instrucción. Aunque en este momento no se realizó una medición, puede observarse visualmente que los marcadores se colocaron en una posición más cercana a su posición correspondiente. Con un error angular presumiblemente inferior a los quince grados observados anteriormente.

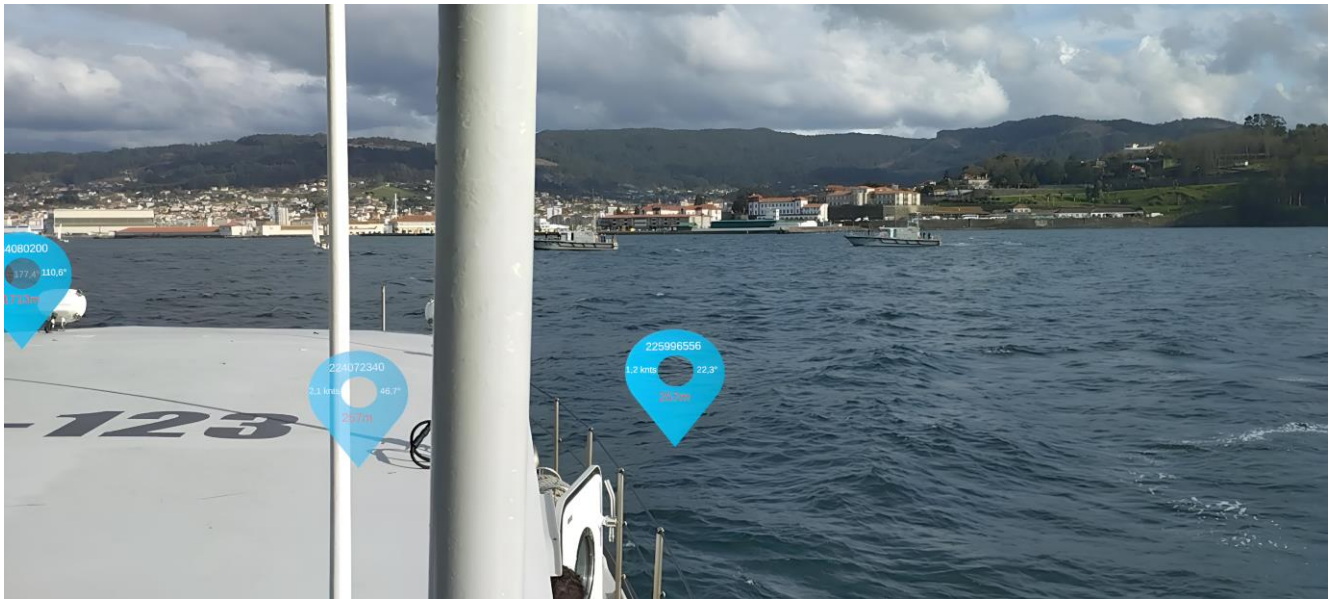


Figura 4-11 Posición de los marcadores parcialmente corregida

Es importante destacar que, durante la prueba en el mar, se buscó deliberadamente una situación en la que se pudiera observar un error significativo en la colocación de los marcadores. Aun así, la aplicación mostró un comportamiento estable la mayor parte del tiempo, especialmente en presencia de contactos de mayor parte del tiempo, especialmente en presencia de contactos de mayor envergadura, ya que estos ofrecían mejores referencias visuales para el correcto posicionamiento de los elementos de realidad aumentada mediante *Google ARCore*. También se debe destacar que se experimentaba un incremento en los errores respecto a la colocación de los marcadores proporcional a la cercanía de los contactos, mostrando un peor funcionamiento en cuanto mayor era la cercanía. Esta condición da pie a los siguientes errores observados.

Para realizar un análisis más detallado de la desviación angular en la colocación de los marcadores, se seleccionó la situación mostrada en la Figura 4-12. Esta situación fue escogida debido a que el marcador coincide con la posición de un contacto cercano a la GM Barrutia (destacado mediante un círculo rojo en la Figura 4-12), y la proximidad entre ambas embarcaciones permite realizar mediciones más precisas. Al coincidir visualmente la posición del marcador con la del contacto real se pudieron tomar los siguientes datos:

- Demora a la GM Barrutia: 297°
- Demora a contacto cercano/ marcador: 302°

Realizando los mismos cálculos que en la anterior medida:

- $|297^\circ - 302^\circ| = 5^\circ \rightarrow$ Desviación marcador GM Barrutia

En esta situación se observa un error angular sensiblemente menor que en el caso anterior. Esta desviación de cinco grados es consistente con el comportamiento general observado durante la mayoría de las pruebas, lo que sugiere que, en condiciones estables, el sistema ofrece una precisión aceptable en la colocación de los marcadores.



Figura 4-12 Imagen tomada para la medición de error de los marcadores

El tercer error observado en la prueba se corresponde al cálculo de la distancia del marcador al usuario. Pese a que este dato se calcula de forma constante mientras haya un marcador colocado, se pudo observar cómo, en determinados momentos la distancia mostrada no correspondía con la real. Un ejemplo de este error se muestra de forma clara en la Figura 4-10. Tomando como ejemplo la lancha de instrucción GM Salas, su marcador indica una distancia respecto al usuario de 261 metros cuando la distancia real era de 219 metros. Aunque es cierto que este error puede estar afectado al error previamente comentado de la colocación de los marcadores y la inexactitud de posición. También debe tenerse en cuenta que la distancia medida desde el radar de la GM Rull hacia la GM Salas puede diferir de la registrada por la aplicación. Estas discrepancias se deben, entre otros factores, a que la aplicación calcula la distancia desde la última posición AIS recibida, mientras que el radar mide hasta el primer punto de contacto de sus ondas con la otra embarcación. Además, elementos como la eslora de la lancha pueden influir en estas diferencias. Durante la prueba se pudo observar un cálculo de la distancia preciso en la mayor parte de los casos, dando errores de máximo diez metros, mostrando solo errores de mayor magnitud en momentos que la propia aplicación no funcionaba correctamente.

Por último, el cuarto error de funcionamiento observado ocurría cuando la distancia entre las lanchas era de ciento cincuenta metros o menor. En este caso la aplicación no calculaba la distancia correctamente y la colocación de los marcadores se veía completamente alterada. En la Figura 4-13 se puede observar cómo se colocan tres marcadores en la misma posición, pertenecientes a la GM Salas, la GM Herrero y a la propia GM Rull donde el usuario estaba embarcado. En todos ellos se muestra una distancia de doscientos metros respecto al usuario. Esto se produce cuando se está realizando una maniobra de aprovisionamiento, durante la cual la distancia entre las lanchas es mínima (véase la Figura 4-14).

Este error puede explicarse por la propia naturaleza del posicionamiento por GPS. Cuando dos embarcaciones están muy próximas entre sí, incluso pequeños errores en la localización (del orden de 5 a 10 metros, habituales en receptores GNSS estándar) pueden provocar desviaciones importantes en la representación relativa entre los contactos. En estos casos, el margen de error del sistema se vuelve comparable o incluso superior a la distancia real entre los objetos, generando proyecciones incorrectas o confusas en la pantalla. A diferencia de lo que ocurre en distancias largas, donde domina el error en la orientación del dispositivo (brújula), en distancias cortas lo que prima es el error de posicionamiento, y este afecta de forma directa al cálculo de la dirección y la colocación de los marcadores. Esto explica que los tres marcadores se proyecten con el mismo sesgo (por ejemplo, todos hacia la izquierda), reflejando una desviación común originada por el GPS.

Este tipo de error puede comprometer la fiabilidad de la aplicación en entornos operativos críticos. Uno de los posibles usos sería su empleo como herramienta de apoyo visual en maniobras de precisión,

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones y revisión de los objetivos

De acuerdo con los objetivos fijados al comienzo del trabajo se concluye que el objetivo principal del mismo se ha logrado de manera satisfactoria. Se ha conseguido desarrollar una aplicación que utiliza la AR para crear marcadores posicionados mediante la localización AIS recibida de las embarcaciones cercanas mostrando información clave de los mismos en tiempo real. Esto permite de un vistazo a los usuarios de la aplicación, identificar contactos y conocer su posición y cinemática.

La integración de los datos AIS recibidos mediante la antena de VHF junto con el receptor RTL-SDR se ha logrado con éxito, permitiendo mostrar los datos AIS de las embarcaciones de los alrededores en tiempo real.

Se han corregido los errores de la anterior versión, con una mejor representación de sus datos, con elementos como un nuevo marcador, y se ha logrado un mejor funcionamiento de la misma incorporando métodos que permiten una mejor colocación y representación de los marcadores, reduciendo el riesgo de confusión en cuanto a la visualización de estos.

Además, se ha realizado una medición de las prestaciones de la aplicación para evaluar la precisión de esta, y se han ido corrigiendo todos los errores que se han ido encontrando nuevos durante el desarrollo, incorporando métodos de calibración o estableciendo límites en cuanto a la imprecisión de la colocación de los marcadores.

La investigación previa ha sido fundamental. El conocimiento de los diferentes mensajes AIS, así como su formato específico ha sido clave para encontrar el método correcto de tratamiento de los mismos para así poder cumplir con los objetivos establecidos. El estudio del entorno de desarrollo del trabajo ha permitido la familiarización necesaria para poder crear la aplicación con las herramientas adecuadas para el tratamiento de elementos geoespaciales.

En cuanto al alcance del proyecto, el trabajo se ha visto en cierta parte limitado al encontrar una dificultad mayor de la esperada para la decodificación de datos como el nombre de las embarcaciones. Este objetivo secundario hubiese proporcionado a los marcadores una información muy interesante. El nivel necesario del lenguaje de programación C# adquirió niveles superiores a los que se disponía en el tramo final del proyecto, requiriendo mucha investigación en línea para encontrar soluciones a los problemas encontrados. Como principal causa de limitación del proyecto está el dominio del entorno de trabajo, el cual presentó problemas nuevos a lo largo del proyecto y limitó el tiempo disponible para desarrollar mejor la aplicación.

Los avances tecnológicos en la sociedad actual traen consigo nuevas herramientas que permiten una navegación cada vez más segura. Los diferentes países y organizaciones se están viendo forzados a desarrollar nuevas formas de explotar estos avances para lograr disponer de los más actualizados sistemas. Tecnologías como la AR o la inteligencia artificial se postulan como buques insignia dentro de los campos de estudio que están logrando avances que hasta hace pocos años serían inimaginables.

La principal conclusión que se extrae del desarrollo del presente TFG es el uso de la AR como una herramienta innovadora para mejorar la ejecución de tareas y procesos, especialmente a través de su integración en aplicaciones prácticas en el ámbito marino. Por lo tanto, con la realización de este trabajo se ha pretendido contribuir de forma directa al proceso de digitalización que, instituciones como la Armada, ya han iniciado y que seguirán afrontado en los próximos años.

Finalmente, a título personal, el desarrollo de este TFG me ha servido para conocer más a fondo el mundo de la AR, e infundirme curiosidad por sus posibles aplicaciones, tanto en el mundo civil como en la Armada. Ahora soy un mayor conocedor de todo el entorno que rodea a la seguridad marítima y de los sistemas AIS, así como del tipo de información que transmite. Por último, el trabajo me ha aportado una visión antes desconocida del proceso de desarrollo de una aplicación de esta índole, que, aun siendo una versión primitiva, me ha enseñado el esfuerzo necesario para conseguir un resultado satisfactorio.

5.2 Líneas futuras

Una vez se ha conseguido crear una aplicación que recibe los datos AIS mediante una antena y un receptor RTL-SDR, y los utiliza para crear marcadores de AR que muestran información importante de los contactos más cercanos, se proponen las siguientes líneas futuras que permiten continuar con el desarrollo de este trabajo y aplicar mejoras sobre el mismo:

- En primer lugar, mejorar la decodificación de los datos AIS, logrando representar datos dinámicos de gran relevancia como el nombre de las embarcaciones, su tipo o bandera. Para ello, probablemente será necesario integrar alguna librería específica o el uso de una base de datos registral vinculada al número IMO.
- Optimizar la posición de los marcadores logrando una colocación de los mismos más precisa y con una altura correspondiente a la del contacto del cual esté representando la información.
- Estudiar la integración de un sistema de reconocimiento de imagen que permita detectar visualmente las embarcaciones en la imagen captada por la cámara y así corregir la colocación de los marcadores AR en pantalla, mejorando su alineación con el contacto real.
- Incorporar un método de cálculo de la distancia más preciso que el actual, identificar la causa y encontrar solución al problema de su cálculo en la cercanía, además de cambiar sus unidades a yardas, que es la unidad adecuada en el mar.
- Ajustar la posición de los marcadores según la distancia, no permitir que se coloquen marcadores detrás de otros que por distancia se encuentren más lejanos.
- Mejorar la visibilidad de los textos según el fondo, por ejemplo, si es de día o de noche, e incluir una leyenda que aclare la información mostrada por los marcadores.
- Refinar la presentación de los marcadores, incluyendo diseños diferenciados por tipo de barco, y lograr crear una aplicación que mantenga un marcador activo sobre el contacto que se vaya actualizando sin desaparecer.
- Estudiar la posibilidad de crear versiones distintas de la misma aplicación para ser utilizadas en distintos sistemas operativos como IOS, ampliando su alcance y portabilidad.
- Evaluar la posibilidad de migrar el proyecto a versiones más modernas de Unity que puedan ofrecer herramientas distintas y más actualizadas para la creación de la aplicación.
- Continuar con el estudio de aplicaciones de la AR en la seguridad marítima y portuaria, desarrollando una posible integración de versiones futuras de este trabajo en tecnologías como los HUD o cámaras de vigilancia.

6 BIBLIOGRAFÍA

- [1] Ministerio de Fomento, *Real Decreto 210/2004, de 6 de febrero, por el que se establece un sistema de seguimiento y de información sobre el tráfico marítimo*. BOE-A-2004-2752. 2004, pp. 6868-6878. Accedido: 4 de febrero de 2025. [En línea]. Disponible en: <https://www.boe.es/eli/es/rd/2004/02/06/210>
- [2] Armada Española, «ENCOMAR - Entorno Colaborativo Marítimo de la Armada». Accedido: 4 de febrero de 2025. [En línea]. Disponible en: <https://encomar.covam.es/bienvenido>
- [3] Kpler, «Data Services». Accedido: 4 de febrero de 2025. [En línea]. Disponible en: <https://www.kpler.com/product/maritime/data-services>
- [4] W. Bellamy III, «Quality Considerations for Aviation Head-up Displays (HUDs)», Avionics International. Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <https://www.aviationtoday.com/2021/01/13/quality-considerations-aviation-head-displays-huds/>
- [5] 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, Universidad de Vigo, Escuela Naval Militar, 2022.
- [6] Aswathi M., Gandhiraj R., y Soman K.P., «Application and Analysis of Smart Meter Data along with RTL SDR and GNU Radio», *Procedia Technology*, vol. 21, pp. 317-325, 2015, doi: 10.1016/j.protcy.2015.10.039.
- [7] «What is AIS?», Comar Systems. Accedido: 17 de febrero de 2025. [En línea]. Disponible en: <https://comarsystems.com/what-is-ais/>
- [8] IALA, «Uso del sistema de AIS en los servicios de ayudas a la navegación marítima», International Association of Marine Aids to Navigation and Lighthouse Authorities Association Internationale de Signalisation Maritime, dic. 2021. Accedido: 17 de febrero de 2025. [En línea]. Disponible en: <https://www.puertos.es/sites/default/files/2024-03/R0126-A-126-USO%20DEL%20AIS%20EN%20LOS%20SERVICIOS%20DE%20ATON.pdf>
- [9] P. A. Niembro Álvarez, *Futuro del VHF: Sistema de intercambio de datos VDES*, Universidad de Oviedo, Oviedo, 2020. Accedido: 18 de febrero de 2025. [En línea]. Disponible en: https://digibuo.uniovi.es/dspace/bitstream/handle/10651/56207/TFM_PedroAntonioNiembroAlvarez.pdf?sequence=3
- [10] All About AIS, «Technical fundamentals». Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <http://www.allaboutais.com/index.php/en/technical-info/technical-fundamentals>

- [11] International Telecommunication Union, «Technical characteristics for an automatic identification system using time division multiple access in the VHF maritime mobile frequency band», feb. 2014. Accedido: 18 de febrero de 2025. [En línea]. Disponible en: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1371-5-201402-I!!PDF-E.pdf
- [12] All About AIS, «Transmission types». Accedido: 20 de febrero de 2025. [En línea]. Disponible en: <http://www.allaboutais.com/index.php/en/technical-info/transmission-types>
- [13] AISHub, «AIS data API (XML / JSON / CSV Webservice) | AISHub». Accedido: 3 de marzo de 2025. [En línea]. Disponible en: <https://www.aishub.net/api>
- [14] V. Kontas, «What is the significance of the AIS Shiptype number? | MarineTraffic». Accedido: 3 de marzo de 2025. [En línea]. Disponible en: <https://support.marinetraffic.com/en/articles/9552866-what-is-the-significance-of-the-ais-shiptype-number>
- [15] «About RTL-SDR». Accedido: 17 de marzo de 2025. [En línea]. Disponible en: <https://www.rtl-sdr.com/about-rtl-sdr/>
- [16] Actisense, *Everything-you-need-to-know-about-NMEA-0183*. 2021. Accedido: 24 de febrero de 2025. [En línea]. Disponible en: <https://actisense.com/wp-content/uploads/2021/01/Everything-you-need-to-know-about-NMEA-0183-1.pdf?srltid=AfmBOoobOs4hyT8JL6iQ1PQE3Zswj0Ov3N-TwKVZSeZgZMS4knUiVPD2>
- [17] Ministerio de Defensa, «BOE-A-2015-1232 Orden DEF/166/2015, de 21 de enero, por la que se desarrolla la organización básica de las Fuerzas Armadas.» Accedido: 24 de febrero de 2025. [En línea]. Disponible en: <https://www.boe.es/buscar/act.php?id=BOE-A-2015-1232>
- [18] IMO, «Vessel Traffic Services». Accedido: 24 de febrero de 2025. [En línea]. Disponible en: <https://www.imo.org/es/OurWork/Safety/Paginas/VesselTrafficServices.aspx>
- [19] M. Rodelgo Lacruz, B. Barragans Martínez, F. García, S. Raña, y S. García, «Real-time AIS data analysis for anomaly detection in the maritime environment», Defence University Center, Spanish Naval Academy, 2022.
- [20] VesselFinder, «Ship & Container Tracking - VesselFinder». Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <https://www.vesselfinder.com/>
- [21] V. Kontas, «Augmented Reality on Mobile App | MarineTraffic». Accedido: 12 de marzo de 2025. [En línea]. Disponible en: <https://support.marinetraffic.com/en/articles/9552818-augmented-reality-on-mobile-app>
- [22] R. T. Azuma, «A Survey of Augmented Reality», *Teleoperators and Virtual Environments*, vol. 6(4), pp. 355-385., ago. 1997.
- [23] J. Linowes y K. Babilinski, *Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia*. Packt Publishing Ltd, 2017.
- [24] Stringnet, «Realidad Aumentada: Qué es, cómo funciona + ejemplos de campañas de marketing AR», Stringnet | Blog Corporativo. Accedido: 11 de febrero de 2025. [En línea]. Disponible en: <https://www.stringnet.pe/blog/realidad-aumentada-que-es/>
- [25] A. Morozova, «Augmented reality in healthcare education — Jasoren». Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <https://www.jasoren.com/ar-in-healthcare-education/>
- [26] P. River Nas, «Navy asks Lockheed Martin to build electro-optical helmet-mounted displays for F-35 combat jet air crews», Military Aerospace. Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <https://www.militaryaerospace.com/sensors/article/14302758/lockheed-martin-helmet-mounted-displays-electro-optical-f-35>

- [27] A. García Romero y Á. Santos García, «La ciberdefensa del gemelo digital en los buques de la armada», *Revista General de Marina*, vol. Parte 05, p. 257, 2023.
- [28] Telefónica, «Telefónica implantará en Navantia una innovadora plataforma digital para el diseño, construcción y apoyo al ciclo de vida de los buques de la armada», Telefónica. Accedido: 27 de febrero de 2025. [En línea]. Disponible en: <https://www.telefonica.com/es/sala-comunicacion/prensa/telefonica-implantara-en-navantia-una-innovadora-plataforma-digital-para-el-diseno-construccion-y-apoyo-al-ciclo-de-vida-de-los-buques-de-la-armada/>
- [29] Google, «Crea experiencias de RA envolventes, basadas en la ubicación y a escala global con la API de ARCore Geospatial», Google for Developers. Accedido: 12 de marzo de 2025. [En línea]. Disponible en: <https://developers.google.com/ar/develop/geospatial?hl=es-419>
- [30] M. J. Salim, «GIScience for Sustainable Development», *JGIS*, vol. 09, n.º 04, pp. 482-492, 2017, doi: 10.4236/jgis.2017.94030.
- [31] Google, «Descripción general de ARCore y los entornos de desarrollo compatibles», Google for Developers. Accedido: 17 de febrero de 2025. [En línea]. Disponible en: <https://developers.google.com/ar/develop?hl=es-419>
- [32] Pieter-Tjerk de Boer, «How RTL-SDR dongles work». Accedido: 17 de marzo de 2025. [En línea]. Disponible en: <https://pa3fwm.nl/technotes/tn20.html>
- [33] U. J. Ekah, E. Obi, y I. Ewona, «Tropospheric Influence on Low-band Very High Frequency (VHF) Radio Waves», *AJARR*, pp. 25-36, oct. 2022, doi: 10.9734/ajarr/2022/v16i11436.
- [34] AIS-catcher, «AIS-catcher.org». Accedido: 6 de marzo de 2025. [En línea]. Disponible en: <https://aiscatcher.org/about>
- [35] Unity Adventure, ¿Cómo usar Geolocalización VPS en Realidad Aumentada? ARCore Geospatial API, (4 de junio de 2022). Accedido: 8 de marzo de 2025. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=-i8UOiB_G0E
- [36] I. Griffiths, «Introducing Ais.Net - High-Performance Parsing in C# | endjin». Accedido: 10 de marzo de 2025. [En línea]. Disponible en: <https://endjin.com/blog/2020/03/introducing-ais-dotnet-high-performance-ais-parsing-in-csharp>
- [37] The ITU Radiocommunication Assembly, «Technical characteristics for a universal shipborne automatic identification system using time division multiple access in the vhf maritime mobile band.», 1998.

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

El uso de la realidad aumentada y nuevas tecnologías de visualización tienen un impacto económico significativo al contribuir a la mejora de la seguridad marítima, especialmente en entornos portuarios y costeros donde el tráfico naval es elevado. Más de doscientos mil buques transitan anualmente por las líneas de comunicación marítima españolas, lo que representa una porción crítica del comercio internacional. La posibilidad de visualizar en tiempo real la posición, rumbo y velocidad de embarcaciones cercanas mediante realidad aumentada aporta una capa adicional de conciencia situacional. Esta mejora permite anticipar riesgos, reducir incidentes de colisión y optimizar los tiempos de maniobra en puerto, lo que, en conjunto, puede traducirse en una reducción de costes asociados a siniestros, retrasos logísticos y daños materiales. La adopción de este tipo de tecnologías, de bajo coste y alta escalabilidad, resulta especialmente atractiva en el ámbito comercial y militar como herramienta de apoyo a la navegación segura.

Una mejor monitorización del tráfico marítimo mediante tecnologías de visualización en tiempo real puede contribuir a la detección temprana de comportamientos anómalos, como rutas no autorizadas, fondeos en zonas protegidas o prácticas de navegación ineficientes. Estos comportamientos, además de ilegales, pueden tener consecuencias medioambientales graves, especialmente en ecosistemas costeros vulnerables. El uso de esta herramienta permite reducir estos riesgos, promoviendo una navegación más sostenible y respetuosa con el medio ambiente.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

El trabajo presentado se sitúa en una frontera tecnológica con potencial de doble uso, tanto en entornos civiles como militares. Este hecho implica una reflexión ética sobre la responsabilidad frente a posibles aplicaciones no previstas o usos que puedan derivar en la vigilancia intrusiva o el control no justificado de la actividad marítima. Si bien el objetivo del proyecto es reforzar la seguridad y la eficiencia en la navegación, resulta pertinente considerar mecanismos de gobernanza y supervisión que orienten su uso dentro de marcos legales y éticos adecuados, especialmente cuando se trate de contextos operativos sensibles.

Desde una perspectiva social, el desarrollo de esta aplicación contribuye positivamente a la mejora de la seguridad en el entorno marítimo, favoreciendo a una navegación más informada y reduciendo el riesgo de colisiones o incidentes. Esta herramienta puede tener un impacto directo en la protección de vidas humanas y en la eficacia de las operaciones portuarias y de patrullaje costero. No obstante, es importante señalar que un uso excesivamente confiado o una dependencia total de este tipo de sistemas podría derivar en una falsa sensación de seguridad, lo que, en situaciones críticas, podría aumentar el riesgo de error humano o provocar accidentes evitables. Por ello, debe fomentarse siempre un uso complementario y supervisado de estas tecnologías, integrándolas como apoyo a la toma de decisiones y no como sustituto del criterio profesional.

ANEXO III: TIPOS DE MENSAJES AIS Y SU CONTENIDO

Tipo de Mensaje AIS	Descripción	Campos Principales
Mensaje 1 (Posición)	Posición de un barco	- MMSI - Latitud - Longitud - Velocidad sobre el suelo (SOG) - Rumbo sobre el suelo (COG) - Hora del mensaje
Mensaje 2 (Posición)	Posición de un barco, sin información sobre la velocidad	- MMSI - Latitud - Longitud - Rumbo sobre el suelo (COG) - Hora del mensaje
Mensaje 3 (Posición)	Posición de un barco, sin información sobre el rumbo	- MMSI - Latitud - Longitud - Velocidad sobre el suelo (SOG) - Hora del mensaje
Mensaje 5 (Información estática)	Información estática de un barco	- MMSI - Nombre del barco - Tipo de embarcación - Dimensiones del barco - Posición del GPS (si está disponible)
Mensaje 6 (Posición - transmitido por un barco que se mueve lentamente o está detenido)	Posición de un barco con baja velocidad o detenido	- MMSI - Latitud - Longitud - Hora del mensaje
Mensaje 8 (Información de seguridad)	Mensaje de seguridad entre barcos	- MMSI (de origen y destino) - Información de seguridad, mensajes de alerta
Mensaje 14 (Información de comunicación)	Mensaje de comunicación de llamada entre barcos	- MMSI - Canal de comunicación - Tipo de mensaje
Mensaje 18 (Posición de referencia)	Posición de un barco con información de referencia	- MMSI - Latitud - Longitud - Tipo de referencia (puerto, etc.)

Mensaje 24 (Información de tripulación y carga)	Información sobre la tripulación y la carga del barco	- MMSI - Número de tripulantes - Tipo de carga - Peso de la carga - Capacidades del barco
--	--	---

Tabla 3 Tipos de mensajes AIS y sus contenidos [37]

ANEXO IV PRIMERA VERSIÓN *VPSMANAGER*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using Google.XR.ARCoreExtensions;
using System;
using TMLPro;

using UnityEngine.UI;

public class VPSManager : MonoBehaviour
{
    [SerializeField] private AREarthManager earthManager;

    [Serializable]
    public struct EarthPosition
    {
        public double Latitude;
        public double Longitude;
        public double Altitude;
        public string name;
    }

    [Serializable]
    private struct Marker
    {
        public ARGeospatialAnchor Anchor;
        public GameObject Object;
    }

    [SerializeField] private ARAnchorManager aRAnchorManager;
    [SerializeField] private List<EarthPosition> earthpositions = new
List<EarthPosition>();

    private List<Marker> markers = new List<Marker>();

    public GameObject ObjectPrefab;

    // To adjust the position of the icon
    public Vector3 offset;

    // Start is called before the first frame update
    void Start()
    {
        VerifyGeospatialSupport();
    }

    private void VerifyGeospatialSupport()
    {
        var result =
earthManager.IsGeospatialModeSupported(GeospatialMode.Enabled);

        switch (result)
        {
            case FeatureSupported.Supported:
                Debug.Log("Ready to VPS");
                PlaceObjects();
                break;
        }
    }
}
```

```

        case FeatureSupported.Unknown:
            Debug.Log("Unknown...");
            Invoke("VerifyGeospatialSupport", 5.0f);
            break;
        case FeatureSupported.Unsupported:
            Debug.Log("Unsupported");
            break;
    }

}

private void PlaceObjects()
{
    if (earthManager.EarthTrackingState == TrackingState.Tracking)
    {
        var geospatialPose = earthManager.CameraGeospatialPose;

        // Esto no se controlaba inicialmente
        if (geospatialPose.HorizontalAccuracy <= 5 &&
geospatialPose.VerticalAccuracy <= 5)
        {
            Quaternion rotation = geospatialPose.EunRotation;
            foreach (var obj in earthpositions)
            {
                var marker = new Marker();
                //Quaternion rotation = geospatialPose.EunRotation;
                marker.Anchor =
ARAnchorManagerExtensions.AddAnchor(aRAnchorManager, obj.Latitude, obj.Longitude,
obj.Altitude, rotation);
                //marker.Anchor =
ARAnchorManagerExtensions.AddAnchor(aRAnchorManager, obj.Latitude, obj.Longitude,
obj.Altitude, Quaternion.identity);
                marker.Object = Instantiate(ObjectPrefab);

marker.Object.transform.Find("Image/Name").GetComponent<TMP_Text>().text =
obj.name;

                markers.Add(marker);
            }
        }
        else
        {
            // Si la precisión no es suficiente, vuelve a intentarlo después de un
tiempo
            Invoke("PlaceObjects", 2.0f);
        }
    }
    else if (earthManager.EarthTrackingState == TrackingState.None)
    {
        Invoke("PlaceObjects", 5.0f);
    }
}

private void Update()
{
    foreach (var marker in markers)
    {
        var img = marker.Object.GetComponentInChildren<Image>();
        // Giving limits to the icon so it sticks on the screen

```

```
// Below calculations witht the assumption that the icon anchor point
is in the middle
// Minimum X position: half of the icon width
float minX = img.GetPixelAdjustedRect().width / 2;
// Maximum X position: screen width - half of the icon width
float maxX = Screen.width - minX;

// Minimum Y position: half of the height
float minY = img.GetPixelAdjustedRect().height / 2;
// Maximum Y position: screen height - half of the icon height
float maxY = Screen.height - minY;

// Temporary variable to store the converted position from 3D world
point to 2D screen point
Vector2 pos =
Camera.main.WorldToScreenPoint(marker.Anchor.transform.position + offset);

// Check if the target is behind us, to only show the icon once the
target is in front
//           if (Vector3.Dot((marker.Anchor.transform.position -
transform.position), transform.forward) < 0)
if (Vector3.Dot((marker.Anchor.transform.position -
Camera.main.transform.position), Camera.main.transform.forward) < 0)
{
    // Check if the target is on the left side of the screen
    if (pos.x < Screen.width / 2)
    {
        // Place it on the right (Since it's behind the player, it's
the opposite)
        pos.x = maxX;
    }
    else
    {
        // Place it on the left side
        pos.x = minX;
    }
}

// Limit the X and Y positions
pos.x = Mathf.Clamp(pos.x, minX, maxX);
pos.y = Mathf.Clamp(pos.y, minY, maxY);

// Update the marker's position
img.transform.position = pos;
// Change the meter text to the distance with the meter unit 'm'

marker.Object.transform.Find("Image/Distance").GetComponent<TMP_Text>().text =
((int)Vector3.Distance(marker.Anchor.transform.position,
Camera.main.transform.position)).ToString() + "m";
}
}
}
```

ANEXO V VPSMANAGER DEFINITIVO

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Collections.Concurrent;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using Google.XR.ARCoreExtensions;
using TMPro;
using UnityEngine.UI;
using System.Linq;
using UnityEngine.Timeline;
using System.Collections;

public class VPSManager3 : MonoBehaviour
{
    [SerializeField] private AREarthManager earthManager;
    [SerializeField] private ARAnchorManager aRAnchorManager;
    public GameObject ObjectPrefab;
    public Vector3 offset;
    public double fixedAltitude = 30.0;
    public int udpPort = 9000;
    private ConcurrentQueue<string> udpMessages = new ConcurrentQueue<string>();
    private Thread udpListenerThread;
    private bool udpListenerRunning = false;
    private List<Marker> markers = new List<Marker>();
    private Dictionary<string, string> mmsiToShipName = new Dictionary<string,
string>();
    private bool markersPlaced = false;
    private Dictionary<string, string> savedFragments = new Dictionary<string,
string>();

    // Variables para la calibración de la brújula
    private bool isCalibrating = false;
    private bool calibrationCompleted = false;
    private int rotationCount = 0;
    private float calibrationThreshold = 60f;
    private float previousHeading = 0f;
    private float calibrationMaxRotations = 8;
    private Thread calibrationThread;

    // Referencias UI para calibración
    public TextMeshProUGUI calibrationInstructionsText; // Texto de instrucciones
    public Image calibrationIndicatorImage; // Indicador de progreso
    (barra o círculo)

    private struct Marker
    {
        public ARGeospatialAnchor Anchor;
        public GameObject Object;
        public AisPosition Position;
    }

    private static readonly char[] SixBitToCharTable = {
        '@', 'A', 'B', 'C', 'D', 'E', 'F', 'F', '4', 'I', 'J', 'K', 'L', 'M', 'N',
        'O',
```

```
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']',  
'^', '-',  
' ', '!', 'A', '$', '%', 'W', '\\', '(', ')', '*', '+', ',', '-', '.', '/',  
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',  
'?'  
};  
  
void Start()  
{  
    // Activar la brújula  
    Input.compass.enabled = true;  
  
    // Iniciar la calibración de la brújula antes de verificar el soporte  
geoespacial  
    StartCalibrationProcess();  
  
    VerifyGeospatialSupport();  
  
    // Solo iniciamos el UDP listener si la calibración ya está completa  
    if (calibrationCompleted)  
    {  
        StartUDPListener();  
    }  
}  
  
void OnDestroy()  
{  
    StopUDPListener();  
  
    // Detener el hilo de calibración si está en ejecución  
    if (calibrationThread != null && calibrationThread.IsAlive)  
    {  
        calibrationThread.Abort();  
        calibrationThread = null;  
    }  
}  
  
private void StartCalibrationProcess()  
{  
    if (!calibrationCompleted)  
    {  
        // Comenzar la calibración en un hilo separado  
        calibrationThread = new Thread(StartCalibration);  
        calibrationThread.IsBackground = true;  
        calibrationThread.Start();  
    }  
}  
  
private void StartCalibration()  
{  
    if (!calibrationCompleted)  
    {  
        isCalibrating = true;  
        rotationCount = 0;  
        previousHeading = Input.compass.trueHeading; // Obtener la orientación  
inicial  
  
        UnityMainThreadDispatcher.Instance().Enqueue(() => {  
            if (calibrationInstructionsText != null)  
            {  
                calibrationInstructionsText.text = "Gira el dispositivo en 8  
para calibrar la brújula...";  
                calibrationInstructionsText.gameObject.SetActive(true);  
            }  
        });  
    }  
}
```

```

    }

    if (calibrationIndicatorImage != null)
    {
        calibrationIndicatorImage.gameObject.SetActive(true);
        calibrationIndicatorImage.fillAmount = 0f;
    }
});

// Realizar el proceso de calibración en un hilo separado
while (isCalibrating && rotationCount < calibrationMaxRotations)
{
    float currentHeading = Input.compass.trueHeading;
    float deltaHeading = Mathf.Abs(currentHeading - previousHeading);

    if (deltaHeading > calibrationThreshold)
    {
        // Incrementar el contador de rotaciones
        rotationCount++;
        previousHeading = currentHeading;

        // Actualizar el indicador de progreso visual en el hilo
principal
        float progressValue = rotationCount /
(float)calibrationMaxRotations;
        UnityMainThreadDispatcher.Instance().Enqueue(() => {
            if (calibrationIndicatorImage != null)
            {
                calibrationIndicatorImage.fillAmount = progressValue;
            }
        });
    }

    // Si ya se alcanzaron las rotaciones necesarias
    if (rotationCount >= calibrationMaxRotations)
    {
        UnityMainThreadDispatcher.Instance().Enqueue(() => {
            FinishCalibration();
        });
        break;
    }

    Thread.Sleep(100); // Pequeña pausa para no sobrecargar el hilo de
calibración
}

// Si no se completó la calibración después de un tiempo considerable
if (rotationCount < calibrationMaxRotations)
{
    UnityMainThreadDispatcher.Instance().Enqueue(() => {
        RequestRecalibration(); // Solicitar nueva calibración si no
se completó correctamente
    });
}

}

private void FinishCalibration()
{
    isCalibrating = false;
    calibrationCompleted = true;
    Debug.Log("Calibración completada con éxito.");
}

```

```
// Actualizar texto y ocultar indicadores
if (calibrationInstructionsText != null)
{
    calibrationInstructionsText.text = "¡Calibración completada!";
    StartCoroutine(HideCalibrationTextAfterDelay(2f));
}
if (calibrationIndicatorImage != null)
{
    calibrationIndicatorImage.gameObject.SetActive(false); // Ocultar la
barra de progreso
}

// Una vez completada la calibración, iniciamos el UDP listener
StartUDPListener();
}

private IEnumerator HideCalibrationTextAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay); // Esperar el tiempo definido
    if (calibrationInstructionsText != null)
    {
        calibrationInstructionsText.gameObject.SetActive(false); // Desactivar
el texto
    }
}

private void RequestRecalibration()
{
    Debug.Log("La calibración no se completó correctamente. Por favor, gira el
dispositivo en 8.");

    // Actualizar el texto de instrucciones
    if (calibrationInstructionsText != null)
    {
        calibrationInstructionsText.text = "Calibración incompleta. Por favor,
gira el dispositivo nuevamente.";
    }

    // Reiniciar el proceso de calibración
    isCalibrating = false;

    // Esperar un momento antes de reiniciar la calibración
    StartCoroutine(RestartCalibrationAfterDelay(3f));
}

private IEnumerator RestartCalibrationAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    StartCalibrationProcess();
}

private void VerifyGeospatialSupport()
{
    var result =
earthManager.IsGeospatialModeSupported(GeospatialMode.Enabled);
    switch (result)
    {
        case FeatureSupported.Supported:
            Debug.Log("Ready to VPS");
            break;
        case FeatureSupported.Unknown:
            Debug.Log("Unknown...");
    }
}
```

```

        Invoke("VerifyGeospatialSupport", 5.0f);
        break;
    case FeatureSupported.Unsupported:
        Debug.Log("Unsupported");
        break;
    }
}

private void StartUDPListener()
{
    udpListenerRunning = true;
    udpListenerThread = new Thread(UDPListen);
    udpListenerThread.IsBackground = true;
    udpListenerThread.Start();
}

private void StopUDPListener()
{
    udpListenerRunning = false;
    if (udpListenerThread != null)
    {
        udpListenerThread.Abort();
        udpListenerThread = null;
    }
}

private void UDPListen()
{
    UdpClient udpClient = new UdpClient(udpPort);
    IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, udpPort);
    while (udpListenerRunning)
    {
        try
        {
            byte[] data = udpClient.Receive(ref remoteEndPoint);
            string message = Encoding.UTF8.GetString(data);
            udpMessages.Enqueue(message);
        }
        catch (SocketException ex)
        {
            Debug.LogError("Socket exception: " + ex.Message);
        }
        catch (Exception ex)
        {
            Debug.LogError("Unexpected exception: " + ex.Message);
        }
    }
    udpClient.Close();
}

private IEnumerator FadeOut(GameObject marker, float fadeSpeed)
{
    // Obtener los componentes de imagen y texto
    Image img = marker.GetComponent<Image>();
    TMP_Text[] textComponents = marker.GetComponents<TMP_Text>();

    // Guardar el color original de la imagen y el texto
    Color originalImageColor = img.color;
    Color[] originalTextColors = textComponents.Select(t =>
t.color).ToArray();

```

```
// Solo reducir la opacidad hasta un valor mínimo, pero no desactivar el
objeto
float minAlpha = 0.3f; // Valor mínimo de opacidad

while (img.color.a > minAlpha)
{
    float alpha = Mathf.Max(minAlpha, img.color.a - fadeSpeed *
Time.deltaTime);
    img.color = new Color(originalImageColor.r, originalImageColor.g,
originalImageColor.b, alpha);

    // Mantener la misma opacidad para todos los textos
    foreach (var text in textComponents)
    {
        text.color = new Color(text.color.r, text.color.g, text.color.b,
alpha);
    }

    yield return null;
}

// No desactivamos el objeto, solo lo dejamos semi-transparente

void Update()
{
    // Solo procesar los mensajes UDP si la calibración está completa
    if (calibrationCompleted)
    {
        while (udpMessages.TryDequeue(out string nmeaSentence))
        {
            if (earthManager.EarthTrackingState == TrackingState.Tracking)
            {
                var geospatialPose = earthManager.CameraGeospatialPose;
                if (geospatialPose.HorizontalAccuracy <= 30 &&
geospatialPose.VerticalAccuracy <= 30)
                {
                    AisPosition pos = DecodeAisPosition(nmeaSentence);
                    if (double.IsNaN(pos.Latitude) ||
double.IsNaN(pos.Longitude))
                    {
                        continue;
                    }
                    Marker existingMarker = markers.FirstOrDefault(m =>
m.Object.transform.Find("Image/Name").GetComponent<TMP_Text>().text == pos.MMSI);
                    if (existingMarker.Object != null)
                    {
                        Destroy(existingMarker.Object);
                        markers.Remove(existingMarker);
                    }
                    Quaternion rotation =
earthManager.CameraGeospatialPose.EulerRotation;
                    var anchor =
ARAnchorManagerExtensions.AddAnchor(aRAnchorManager, pos.Latitude, pos.Longitude,
fixedAltitude, rotation);
                    var markerObj = Instantiate(ObjectPrefab);

                    markerObj.transform.Find("Image/Name").GetComponent<TMP_Text>().text = pos.MMSI;
                    markerObj.transform.Find("Image/Distance").GetComponent<TMP_Text>().text = "0m";
                }
            }
        }
    }
}
```

```
markerObj.transform.Find("Image/SOG").GetComponent<TMP_Text>().text =
pos.Speed.ToString("F1") + " knts";

markerObj.transform.Find("Image/COG").GetComponent<TMP_Text>().text =
pos.Course.ToString("F1") + "°";
        Marker newMarker = new Marker { Anchor = anchor, Object =
markerObj, Position = pos }; // Almacenar la posición AIS
        markers.Add(newMarker);
        StartCoroutine(FadeOut(markerObj, 0.05f)); // Velocidad de
desvanecimiento
    }
    else
    {
        // Si la precisión no es suficiente, vuelve a intentarlo
después de un tiempo
        Invoke("Update", 2.0f);
    }
}
else if (earthManager.EarthTrackingState == TrackingState.None)
{
    Invoke("Update", 5.0f);
}
}
foreach (var marker in markers)
{
    var img = marker.Object.GetComponentInChildren<Image>();
    var textComponents =
marker.Object.GetComponentsInChildren<TMP_Text>(); // Obtener todos los
componentes de texto dentro del marcador

    Vector3 worldPosition = marker.Anchor.transform.position + offset;
// La posición del marcador en el mundo
    Vector3 screenPoint =
Camera.main.WorldToScreenPoint(worldPosition); // Convertir la posición del mundo
a pantalla

    if (screenPoint.z > 0) // Si el marcador está en frente de la
cámara
    {
        // El objeto está en frente de la cámara, actualiza la
posición de la UI
        img.transform.position = new Vector2(screenPoint.x,
screenPoint.y);
        img.enabled = true; // Asegúrate de que la imagen se vea si
está en frente de la cámara
        foreach (var text in textComponents)
        {
            text.enabled = true; // Hacer visible el texto
        }
    }
    else
    {
        // El objeto está detrás de la cámara
        img.enabled = false; // Desactiva la imagen (UI)
        foreach (var text in textComponents)
        {
            text.enabled = false; // Desactivar el texto
        }
    }
}
```

```
        float distance =
Vector3.Distance(marker.Anchor.transform.position,
Camera.main.transform.position);
        TMP_Text distanceText =
marker.Object.transform.Find("Image/Distance")?.GetComponent<TMP_Text>();
        if (distanceText != null)
        {
            distanceText.text = ((int)distance).ToString() + "m";

            // Cambiar el color según la distancia
            if (distance < 1800f)
            {
                // Distancia menor a 1800 metros - color rojo
                distanceText.color = new Color(1f, 0f, 0f,
distanceText.color.a); // Rojo con la misma alpha actual
            }
            else
            {
                // Distancia mayor o igual a 1800 metros - color verde
                distanceText.color = new Color(0f, 1f, 0f,
distanceText.color.a); // Verde con la misma alpha actual
            }

            // Actualizar los valores de Speed y Course directamente desde
el marcador
            if (marker.Object.transform.Find("Image/SOG") != null)
            {
                marker.Object.transform.Find("Image/SOG").GetComponent<TMP_Text>().text =
marker.Position.Speed.ToString("F1") + " knts";
            }

            if (marker.Object.transform.Find("Image/COG") != null)
            {
                marker.Object.transform.Find("Image/COG").GetComponent<TMP_Text>().text =
marker.Position.Course.ToString("F1") + "°";
            }
        }
    }
}

private AisPosition DecodeAisPosition(string nmeaSentence)
{
    string[] parts = nmeaSentence.Split(',');
    if (parts.Length < 6)
    {
        return new AisPosition { Latitude = double.NaN, Longitude =
double.NaN, ShipName = "No name", MMSI = "Unknown" };
    }
    string payload = parts[5];
    string bits = ConvertPayloadToBinary(payload);
    int messageType = Convert.ToInt32(bits.Substring(0, 6), 2);
    if (messageType < 1 || messageType > 3)
    {
        return new AisPosition { Latitude = double.NaN, Longitude =
double.NaN, ShipName = "No name", MMSI = "Unknown" };
    }
    // Decodificación de MMSI (ahora está correctamente extraído desde los
bits de la carga útil)
```

```

        string mmsiBits = bits.Substring(8, 30); // Los 30 bits que corresponden
al MMSI
        int mmsiInt = Convert.ToInt32(mmsiBits, 2);
        string mmsi = mmsiInt.ToString(); // El MMSI como string
        string lonBits = bits.Substring(61, 28);
        string latBits = bits.Substring(89, 27);
        int rawLon = TwosComplementToInt(lonBits);
        int rawLat = TwosComplementToInt(latBits);
        double longitude = rawLon / 600000.0;
        double latitude = rawLat / 600000.0;
        string sogBits = bits.Substring(50, 10);
        string cogBits = bits.Substring(116, 12);
        double speed = DecodeSpeed(sogBits);
        double course = DecodeCourse(cogBits);
        string shipName = DecodeShipName(bits);
        return new AisPosition { Latitude = latitude, Longitude = longitude,
ShipName = shipName, MMSI = mmsi, Speed = speed, Course = course };
    }

    private string ConvertPayloadToBinary(string payload)
    {
        StringBuilder bitString = new StringBuilder();
        foreach (char c in payload)
        {
            int sixBit = c - 48;
            if (sixBit > 40) sixBit -= 8;
            bitString.Append(Convert.ToString(sixBit, 2).PadLeft(6, '0'));
        }
        return bitString.ToString();
    }

    private int TwosComplementToInt(string bits)
    {
        if (bits[0] == '0')
            return Convert.ToInt32(bits, 2);
        else
        {
            string inverted = new string(bits.Select(b => b == '0' ? '1' :
'0').ToArray());
            int intValue = Convert.ToInt32(inverted, 2) + 1;
            return -intValue;
        }
    }

    private double DecodeSpeed(string sogBits)
    {
        int sogRaw = Convert.ToInt32(sogBits, 2);
        if (sogRaw == 1023)
        {
            return double.NaN;
        }
        return sogRaw / 10.0;
    }

    private double DecodeCourse(string cogBits)
    {
        int cogRaw = Convert.ToInt32(cogBits, 2);
        if (cogRaw == 3600) // Valor especial que indica "no disponible"
        {
            return double.NaN;
        }
        return cogRaw / 10.0; // El rumbo se expresa en décimas de grado
    }

```

```
}

private string DecodeShipName(string bits)
{
    StringBuilder shipName = new StringBuilder();
    int startIndex = 8;
    int length = 120;
    for (int i = startIndex; i < startIndex + length; i += 6)
    {
        string sixBitChar = bits.Substring(i, 6);
        int charCode = Convert.ToInt32(sixBitChar, 2);
        if (charCode == 0)
        {
            break; // Detener la decodificación si encontramos un caracter de
relleno
        }
        if (charCode < SixBitToCharTable.Length)
        {
            shipName.Append(SixBitToCharTable[charCode]);
        }
        else
        {
            shipName.Append('?'); // Si el carácter no es visible, poner un
marcador
        }
    }
    return shipName.ToString().Trim();
}

public struct AisPosition
{
    public double Latitude;
    public double Longitude;
    public string ShipName;
    public string MMSI;
    public double Speed; // Velocidad sobre el fondo (SOG)
    public double Course; // Curso sobre el fondo (COG)
}

// Clase auxiliar para ejecutar acciones en el hilo principal (necesaria para
actualizar la UI)
public class UnityMainThreadDispatcher : MonoBehaviour
{
    private static readonly Queue<Action> _executionQueue = new Queue<Action>();
    private static UnityMainThreadDispatcher _instance = null;

    public static UnityMainThreadDispatcher Instance()
    {
        if (_instance == null)
        {
            _instance = FindObjectOfType<UnityMainThreadDispatcher>();
            if (_instance == null)
            {
                GameObject go = new GameObject("UnityMainThreadDispatcher");
                _instance = go.AddComponent<UnityMainThreadDispatcher>();
                DontDestroyOnLoad(go);
            }
        }
        return _instance;
    }

    public void Enqueue(Action action)
```

```
{
    lock (_executionQueue)
    {
        _executionQueue.Enqueue(action);
    }
}

void Update()
{
    lock (_executionQueue)
    {
        while (_executionQueue.Count > 0)
        {
            _executionQueue.Dequeue().Invoke();
        }
    }
}
```

ANEXO VI: CÓDIGO AISRECEIVER

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

public class AISReceiver : MonoBehaviour
{
    public int udpPort = 9000;
    private UdpClient udpClient;
    private bool isListening = false;
    public static Action<AISData> OnAISDataReceived;
    void Start()
    {
        try
        {
            udpClient = new UdpClient(udpPort);
            isListening = true;
            Debug.Log("Iniciando escucha UDP en el puerto " + udpPort);
            udpClient.BeginReceive(ReceiveCallback, null);
        }
        catch (Exception ex)
        {
            Debug.LogError("Error al iniciar UDP: " + ex.Message);
        }
    }
    void ReceiveCallback(IAsyncResult ar)
    {
        try
        {
            IPEndPoint remoteEP = new IPEndPoint(IPAddress.Any, udpPort);
            byte[] data = udpClient.EndReceive(ar, ref remoteEP);
            string message = Encoding.ASCII.GetString(data);
            Debug.Log("Mensaje recibido desde " + remoteEP.Address + ": " +
message);
            ProcessNMEAMessage(message);
            if (isListening)
            {
                udpClient.BeginReceive(ReceiveCallback, null);
            }
        }
        catch (Exception ex)
        {
            Debug.LogError("Error en ReceiveCallback: " + ex.Message);
        }
    }
    void ProcessNMEAMessage(string message)
    {
        if (message.StartsWith("!AIVDM") || message.StartsWith("!AIVDO"))
        {
            string[] parts = message.Split(',');
            if (parts.Length >= 6)
            {
                string payload = parts[5];
                Debug.Log("Payload AIS: " + payload);
                AISData aisData = DecodeAISPayload(payload);
                if (aisData != null)
                {
                    OnAISDataReceived?.Invoke(aisData);
                }
            }
        }
    }
}
```

```
    }
}
else
{
    Debug.Log("Mensaje no AIS recibido: " + message);
}
}

public class AISData
{
    public int mmsi;
    public double latitude;
    public double longitude;
    public string name;
}

AISData DecodeAISPayload(string payload)
{
    StringBuilder bitString = new StringBuilder();
    foreach (char c in payload)
    {
        int sixBit = c - 48;
        if (sixBit > 40) sixBit -= 8;
        string bits = Convert.ToString(sixBit, 2).PadLeft(6, '0');
        bitString.Append(bits);
    }
    string bitsTotal = bitString.ToString();
    Debug.Log("Cadena de bits: " + bitsTotal);
    try
    {
        AISData data = new AISData();
        int mmsi = Convert.ToInt32(bitsTotal.Substring(8, 30), 2);
        data.mmsi = mmsi;
        data.name = "Barco AIS";

        data.latitude = 0.0;
        data.longitude = 0.0;
        return data;
    }
    catch (Exception ex)
    {
        Debug.LogError("Error al decodificar payload AIS: " + ex.Message);
        return null;
    }
}

void OnApplicationQuit()
{
    isListening = false;
    if (udpClient != null)
    {
        udpClient.Close();
    }
}
}
```

ANEXO VII: CÓDIGO VPSMANAGERUDP

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using Google.XR.ARCoreExtensions;
using System;
using TMPro;
using UnityEngine.UI;

using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Collections.Concurrent;

public class VPSManagerUDP : MonoBehaviour
{
    [SerializeField] private AREarthManager earthManager;
    [SerializeField] private ARAnchorManager aRAnchorManager;
    public GameObject ObjectPrefab;
    public Vector3 offset;

    // Coordenadas fijas para colocar el marcador (puedes modificarlas según tu
    escena)
    public double fixedLatitude = 42.3937031168094;
    public double fixedLongitude = -8.70299426592422;
    public double fixedAltitude = 30.0;

    // Puerto UDP para escuchar
    public int udpPort = 9000;

    // Cola thread-safe para almacenar los mensajes NMEA recibidos
    private ConcurrentQueue<string> udpMessages = new ConcurrentQueue<string>();

    // Hilo para la escucha UDP
    private Thread udpListenerThread;
    private bool udpListenerRunning = false;

    // Lista para almacenar los marcadores creados
    private List<Marker> markers = new List<Marker>();

    // Estructura para almacenar el ancla y el objeto asociado
    private struct Marker
    {
        public ARGeospatialAnchor Anchor;
        public GameObject Object;
    }

    void Start()
    {
        VerifyGeospatialSupport();
        StartUDPListener();
    }

    void OnDestroy()
    {
        StopUDPListener();
    }
}
```

```

private void VerifyGeospatialSupport()
{
    var result =
earthManager.IsGeospatialModeSupported(GeospatialMode.Enabled);
    switch (result)
    {
        case FeatureSupported.Supported:
            Debug.Log("Ready to VPS");
            break;
        case FeatureSupported.Unknown:
            Debug.Log("Unknown...");
            Invoke("VerifyGeospatialSupport", 5.0f);
            break;
        case FeatureSupported.Unsupported:
            Debug.Log("Unsupported");
            break;
    }
}

// Inicia el hilo de escucha UDP
private void StartUDPListener()
{
    udpListenerRunning = true;
    udpListenerThread = new Thread(UDPListen);
    udpListenerThread.IsBackground = true;
    udpListenerThread.Start();
}

// Detiene el hilo de escucha UDP
private void StopUDPListener()
{
    udpListenerRunning = false;
    if (udpListenerThread != null)
    {
        udpListenerThread.Abort();
        udpListenerThread = null;
    }
}

// Bucle de escucha UDP en un hilo separado
private void UDPListen()
{
    UdpClient udpClient = new UdpClient(udpPort);
    IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, udpPort);

    while (udpListenerRunning)
    {
        try
        {
            byte[] data = udpClient.Receive(ref remoteEndPoint);
            string message = Encoding.UTF8.GetString(data);
            // Encola la sentencia NMEA recibida
            udpMessages.Enqueue(message);
        }
        catch (SocketException ex)
        {
            Debug.Log("Socket exception: " + ex);
        }
    }
    udpClient.Close();
}

```

```
// En Update se revisa la cola de mensajes y se crean los marcadores
correspondientes
void Update()
{
    // Procesar todos los mensajes recibidos
    while (udpMessages.TryDequeue(out string nmeaSentence))
    {
        // Solo creamos el marcador si el EarthTracking está en Tracking
        //if (earthManager.EarthTrackingState == TrackingState.Tracking)
        //{
            // Utilizamos la rotación de la cámara o Quaternion.identity según
se requiera
            Quaternion rotation =
earthManager.CameraGeospatialPose.EulerRotation;
            // Crea un ancla en la posición fija
            var anchor = ARAnchorManagerExtensions.AddAnchor(aRAnchorManager,
fixedLatitude, fixedLongitude, fixedAltitude, rotation);
            // Instancia el objeto del marcador
            var markerObj = Instantiate(ObjectPrefab);
            // Asigna el texto del marcador como la sentencia NMEA completa

markerObj.transform.Find("Image/Name").GetComponent<TMP_Text>().text =
nmeaSentence;

            // Agrega el marcador a la lista
            Marker marker = new Marker { Anchor = anchor, Object = markerObj
};

            markers.Add(marker);
        //}
    }

    // Actualiza la posición y la distancia de cada marcador (para el
canvas/UI)
    foreach (var marker in markers)
    {
        var img = marker.Object.GetComponentInChildren<Image>();
        Vector2 pos =
Camera.main.WorldToScreenPoint(marker.Anchor.transform.position + offset);
        img.transform.position = pos;
        float distance = Vector3.Distance(marker.Anchor.transform.position,
Camera.main.transform.position);

marker.Object.transform.Find("Image/Distance").GetComponent<TMP_Text>().text =
((int)distance).ToString() + "m";
    }
}
```

ANEXO VIII: CÓDIGO DE PROCESAMIENTO DATOS AIS

```
// <copyright file="NmeaStreamParser.cs" company="Endjin Limited">
// Copyright (c) Endjin Limited. All rights reserved.
// </copyright>

namespace Ais.Net
{
    using System;
    using System.Buffers;
    using System.IO;
    using System.IO.Pipelines;
    using System.Threading.Tasks;

    /// <summary>
    /// Processes streams containing lines of ASCII-encoded text, each containing
    an NMEA message.
    /// </summary>
    public static class NmeaStreamParser
    {
        /// <summary>
        /// Process the contents of a file one AIS message at a time.
        /// </summary>
        /// <param name="path">Path of the file to process.</param>
        /// <param name="processor">Handler for the AIS messages.</param>
        /// <returns>A task that completes when the stream has been
    processed.</returns>
        /// <remarks>
        /// This reassembles AIS messages that have been split over multiple NMEA
    lines.
        /// </remarks>
        public static Task ParseFileAsync(
            string path,
            INmeaAisMessageStreamProcessor processor)
        {
            return ParseFileAsync(path, processor, new NmeaParserOptions());
        }

        /// <summary>
        /// Process the contents of a file one AIS message at a time.
        /// </summary>
        /// <param name="path">Path of the file to process.</param>
        /// <param name="processor">Handler for the AIS messages.</param>
        /// <param name="options">Configures parser behaviour.</param>
        /// <returns>A task that completes when the stream has been
    processed.</returns>
        /// <remarks>
        /// This reassembles AIS messages that have been split over multiple NMEA
    lines.
        /// </remarks>
        public static async Task ParseFileAsync(
            string path,
            INmeaAisMessageStreamProcessor processor,
            NmeaParserOptions options)
        {
            using var adapter = new NmeaLineToAisStreamAdapter(processor,
    options);
            await ParseFileAsync(path, adapter, options).ConfigureAwait(false);
        }

        /// <summary>
```

```
    /// Process the contents of a file.
    /// </summary>
    /// <param name="path">Path of the file to process.</param>
    /// <param name="processor">Handler for the parsed lines.</param>
    /// <returns>A task that completes when the stream has been
processed.</returns>
    public static Task ParseFileAsync(
        string path,
        INmeaLineStreamProcessor processor)
    {
        return ParseFileAsync(path, processor, new NmeaParserOptions());
    }

    /// <summary>
    /// Process the contents of a file.
    /// </summary>
    /// <param name="path">Path of the file to process.</param>
    /// <param name="processor">Handler for the parsed lines.</param>
    /// <param name="options">Configures parser behaviour.</param>
    /// <returns>A task that completes when the stream has been
processed.</returns>
    public static async Task ParseFileAsync(
        string path,
        INmeaLineStreamProcessor processor,
        NmeaParserOptions options)
    {
        // This turns off internal file stream buffering
        using var file = new FileStream(path, FileMode.Open, FileAccess.Read,
FileShare.ReadWrite, bufferSize: 1, useAsync: true);
        await ParseStreamAsync(file, processor,
options).ConfigureAwait(false);
    }

    /// <summary>
    /// Process the contents of a stream.
    /// </summary>
    /// <param name="stream">The stream to process.</param>
    /// <param name="processor">Handler for the parsed lines.</param>
    /// <returns>A task that completes when the stream has been
processed.</returns>
    public static Task ParseStreamAsync(
        Stream stream,
        INmeaLineStreamProcessor processor)
    {
        return ParseStreamAsync(stream, processor, new NmeaParserOptions());
    }

    /// <summary>
    /// Process the contents of a stream.
    /// </summary>
    /// <param name="stream">The stream to process.</param>
    /// <param name="processor">Handler for the parsed lines.</param>
    /// <param name="options">Configures parser behaviour.</param>
    /// <returns>A task that completes when the stream has been
processed.</returns>
    public static async Task ParseStreamAsync(
        Stream stream,
        INmeaLineStreamProcessor processor,
        NmeaParserOptions options)
    {
        int lines = 0;
        int ticksAtStart = Environment.TickCount;
        int ticksAtLastLineCount = ticksAtStart;
```

```

try
{
    PipeReader reader = CreateFileReader(stream);

    byte[] splitLineBuffer = new byte[1000];
    while (true)
    {
        ReadResult result = await
reader.ReadAsync().ConfigureAwait(false);
        ReadOnlySequence<byte> remainingSequence =
ProcessBuffer(result);

        reader.AdvanceTo(remainingSequence.Start,
remainingSequence.End);

        if (result.IsCompleted)
        {
            break;
        }
    }

    const int LineCountInterval = 100000;
    int finalTicks = Environment.TickCount;
    int totalTicks = finalTicks - ticksAtStart;

    processor.Progress(
        true,
        lines,
        totalTicks,
        lines % LineCountInterval,
        finalTicks - ticksAtLastLineCount);

    ReadOnlySequence<byte> ProcessBuffer(
        in ReadResult result)
    {
        ReadOnlySpan<byte> lineSpan;
        SequencePosition? position = null;

        ReadOnlySequence<byte> remainingSequence = result.Buffer;

        while ((position = remainingSequence.PositionOf((byte)'\n') ??
(remainingSequence.IsEmpty || !result.IsCompleted ? default(SequencePosition?) :
remainingSequence.End)) != null)
        {
            ReadOnlySequence<byte> line =
remainingSequence.Slice(remainingSequence.Start, position.Value);

            if (line.IsSingleSegment)
            {
                lineSpan = line.First.Span;
            }
            else
            {
                Span<byte> reassemblySpan = splitLineBuffer;
                line.CopyTo(reassemblySpan);
                lineSpan = reassemblySpan.Slice(0, (int)line.Length);
            }

            if (lineSpan.Length > 0 && lineSpan[lineSpan.Length - 1]
== (byte)'\r')
            {

```

```
        lineSpan = lineSpan.Slice(0, lineSpan.Length - 1);
    }

    if (lineSpan.Length > 0)
    {
        try
        {
            var parsedLine = new NmeaLineParser(lineSpan,
options.ThrowWhenTagBlockContainsUnknownFields);

            processor.OnNext(parsedLine, lines + 1);
        }
        catch (Exception x)
        {
            processor.OnError(lineSpan, x, lines + 1);
        }
    }

    remainingSequence =
position.Value.Equals(remainingSequence.End)
    ? remainingSequence.Slice(remainingSequence.End)
    :
remainingSequence.Slice(remainingSequence.GetPosition(1, position.Value));

    if (++lines % LineCountInterval == 0)
    {
        int currentTicks = Environment.TickCount;
        int ticksSinceLastLineCount = currentTicks -
ticksAtLastLineCount;

        processor.Progress(
            false,
            lines,
            currentTicks - ticksAtStart,
            LineCountInterval,
            ticksSinceLastLineCount);
        ticksAtLastLineCount = currentTicks;
    }
}

return remainingSequence;
}
finally
{
    processor.OnCompleted();
}
}

private static PipeReader CreateFileReader(Stream file)
{
    async Task ProcessFileAsync(PipeWriter writer)
    {
        try
        {
            while (true)
            {
                Memory<byte> memory = writer.GetMemory();

                // See https://github.com/
dotnet/corefx/blob/master/src/Common/src/CoreLib/System/IO/Stream.cs#L379
                // for how Stream.ReadAsync handles Memory<byte>.

                #if NETSTANDARD2_0
```

```
System.Runtime.InteropServices.MemoryMarshal.TryGetArray(memory, out
ArraySegment<byte> memoryArray);
    int read = await file.ReadAsync(memoryArray.Array,
memoryArray.Offset, memoryArray.Count).ConfigureAwait(false);
#else
    int read = await
file.ReadAsync(memory).ConfigureAwait(false);
#endif

    if (read == 0)
    {
        break;
    }

    writer.Advance(read);

    await writer.FlushAsync().ConfigureAwait(false);
}
}
catch (Exception ex)
{
    writer.Complete(ex);
}
finally
{
    writer.Complete();
}
}

// A 1MB buffer seems to be about the sweet spot on Ian's desktop.
Further research
// required to work out what is the right value anywhere else.
var pipe = new Pipe(new PipeOptions(minimumSegmentSize: 1024 * 1024));
_ = ProcessFileAsync(pipe.Writer);

return pipe.Reader;
}
}
```

ANEXO IX CÓDIGO

INMEA AIS MESSAGE STREAM PROCESSOR

```
// <copyright file="INmeaAisMessageStreamProcessor.cs" company="Endjin Limited">
// Copyright (c) Endjin Limited. All rights reserved.
// </copyright>

namespace Ais.Net
{
    using System;

    /// <summary>
    /// Receives AIS messages parsed from an NMEA file by <see
    cref="NmeaStreamParser"/>.
    /// </summary>
    /// <remarks>
    /// In cases where AIS messages have been fragmented across multiple lines,
    they will be
    /// reassembled and passed to implementors of this interface as single
    messages.
    /// </remarks>
    public interface INmeaAisMessageStreamProcessor
    {
        /// <summary>
        /// Called for each complete AIS message.
        /// </summary>
        /// <param name="firstLine">The parsed line.</param>
        /// <param name="asciiPayload">The full payload.</param>
        /// <param name="padding">The number of bits of padding at the end of the
    payload.</param>
        /// <remarks>
        /// Where the source message was not fragmented, the <c>asciiPayload</c>
    will refer
        /// directly to the original message's payload in situ. If the AIS message
    was split
        /// across multiple NMEA sentences, the various sections of the payload
    will have been
        /// copied into a reassembly buffer, so that it can be presented as a
    single contiguous
        /// block of data.
        /// </remarks>
        void OnNext(
            in NmeaLineParser firstLine,
            in ReadOnlySpan<byte> asciiPayload,
            uint padding);

        /// <summary>
        /// Called when a line cannot be parsed, e.g. it does not contain a well-
    formed NMEA
        /// message.
        /// </summary>
        /// <param name="line">The line that cannot be parsed.</param>
        /// <param name="error">An exception describing the problem.</param>
        /// <param name="lineNumber">The 1-based line number on which the error
    was detected.</param>
        void OnError(in ReadOnlySpan<byte> line, Exception error, int lineNumber);

        /// <summary>
        /// Called when all lines have been processed.
        /// </summary>
        void OnCompleted();
    }
}
```

```
    /// <summary>
    /// Called regularly to provide progress reports, and always called at the
end.
    /// </summary>
    /// <param name="done">True if all processing is complete.</param>
    /// <param name="totalNmeaLines">The number of NMEA lines
processed.</param>
    /// <param name="totalAisMessages">The number of AIS messages
processed.</param>
    /// <param name="totalTicks">The number of milliseconds of
processing.</param>
    /// <param name="nmeaLinesSinceLastUpdate">The number of lines since the
last progress notification.</param>
    /// <param name="aisMessagesSinceLastUpdate">The number of AIS messages
since the last progress notification.</param>
    /// <param name="ticksSinceLastUpdate">The number of ticks since the last
progress notification.</param>
    void Progress(
        bool done,
        int totalNmeaLines,
        int totalAisMessages,
        int totalTicks,
        int nmeaLinesSinceLastUpdate,
        int aisMessagesSinceLastUpdate,
        int ticksSinceLastUpdate);
}
```

ANEXO X NMEA AIS TEXT FIELD PARSER

```
// <copyright file="NmeaAisTextFieldParser.cs" company="Endjin Limited">
// Copyright (c) Endjin Limited. All rights reserved.
// </copyright>

namespace Ais.Net
{
    using System;

    /// <summary>
    /// Parses a text field such as a ship's name stored in an AIS payload of an
    NMEA message.
    /// </summary>
    public readonly ref struct NmeaAisTextFieldParser
    {
        private readonly NmeaAisBitVectorParser bits;
        private readonly uint bitOffset;
        private readonly uint bitLength;

        /// <summary>
        /// Creates a <see cref="NmeaAisTextFieldParser"/>.
        /// </summary>
        /// <param name="bits">The bit vector (in 6-bit ASCII encoding) containing
the name.</param>
        /// <param name="bitLength">The length of the name field in bits.</param>
        /// <param name="bitOffset">The bit offset of the name field within the
bit vector.</param>
        public NmeaAisTextFieldParser(NmeaAisBitVectorParser bits, uint bitLength,
uint bitOffset)
        {
            this.bits = bits;
            this.bitOffset = bitOffset;
            this.bitLength = bitLength;
        }

        /// <summary>
        /// Gets the number of characters in this field.
        /// </summary>
        public uint CharacterCount => this.bitLength / 6;

        /// <summary>
        /// Gets the ASCII value of the character at the specified position.
        /// </summary>
        /// <param name="index">Character position.</param>
        /// <returns>The ASCII value of the character at the specified
position.</returns>
        public byte GetAscii(uint index)
        {
            uint bitIndexInField = index * 6;

            if ((bitIndexInField + 6) > this.bitLength)
            {
                throw new ArgumentOutOfRangeException(
                    nameof(index),
                    "Character index would read off end of name field");
            }

            byte aisValue = (byte)this.bits.GetUnsignedInteger(6, this.bitOffset +
bitIndexInField);

            return AisStrings.AisCharacterToAsciiValue(aisValue);
        }
    }
}
```

```
    }

    /// <summary>
    /// Writes the characters into a buffer in ASCII format.
    /// </summary>
    /// <param name="targetBuffer">
    /// The target buffer, which must be at least as long as <see
    cref="CharacterCount"/>.
    /// </param>
    /// <remarks>
    /// The reason we don't just provide a simple property that returns a
    string is that it
    /// would cause a heap allocation, defeating the entire point of using
    <c>ref struct</c>.
    /// </remarks>
    public void WriteAsAscii(in Span<byte> targetBuffer)
    {
        for (int i = 0; i < targetBuffer.Length; ++i)
        {
            targetBuffer[i] = this.GetAscii((uint)i);
        }
    }
}
```

ANEXO XI VÍDEO

En el siguiente QR se puede acceder a una prueba visual del funcionamiento de la aplicación.

