Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de representación de rutas marítimas con detección de anomalías en tiempo real

Grado en Ingeniería Mecánica

ALUMNO: Mariano Perales Hernández

DIRECTORES: Miguel Rodelgo Lacruz

José María Núñez Ortuño

CURSO ACADÉMICO: 2016-2017

Universidad de Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de representación de rutas marítimas con detección de anomalías en tiempo real

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidad de Vigo

RESUMEN

El empleo de sistemas de inteligencia artificial (IA) para la detección de anomalías en rutas marítimas, ha demostrado ser de gran utilidad en el análisis y supervisión de la información del tráfico marítimo, sin embargo, para su utilización práctica es imprescindible que funcione en tiempo real.

Este trabajo pretende adaptar uno de estos sistemas de IA para que reciba, en tiempo real, los datos AIS (Automatic Identification System) de los buques que naveguen por una determinada zona, los procese y analice el grado de anomalía de las derrotas de dichos buques.

Para ello emplea una aplicación previamente desarrollada basada en la plataforma abierta de inteligencia artificial NuPIC. El resultado de este análisis puede visualizarse, también en tiempo real, gracias a la pasarela desarrollada que se integrada con el software de cartografía electrónica OpenCPN.

PALABRAS CLAVE

Anomalía, AIS, Tiempo Real, Tráfico Marítimo, OpenCPN y Python

AGRADECIMIENTOS

En primer lugar, quiero agradecer a los tutores de este trabajo, Miguel Rodelgo Lacruz y José María Núñez Ortuño, que han estado siempre disponibles para apoyarme y aconsejarme en el desarrollo de este trabajo, sirviendo como guía cuando los problemas detenían el avance en el proyecto.

En segundo lugar, debo agradecer a *Marine Traffic* el acceso a sus servicios que generosamente nos ha dado, sirviendo estos como base para el desarrollo de todo el trabajo, por incluirlo como aportación a su red de investigación.

En tercer lugar, también debo agradecerle a la comunidad OpenCPN, y en concreto a su desarrollador, Dave Register, por ofrecer de forma libre este programa que, además de una muy útil herramienta para la navegación, ha permitido el buen desarrollo de este trabajo.

Por último, quiero agradecer a mi familia, en concreto a mi madre, que durante toda mi vida ha estado ahí para enseñarme que todo fin se logra con un esfuerzo, y que también me ha apoyado durante el desarrollo del trabajo, manteniendo siempre interés y preocupación por los avances realizados.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

En la actualidad, la gran mayoría de los avances tecnológicos surgen como alternativa mecánica a la realización de un trabajo que antes era realizado por una o varias personas. En este aspecto, y en el ámbito de la navegación, la tecnología, principalmente en los últimos siglos, ha supuesto una absoluta revolución, que ha transformado las galeras de los antiguos fenicios, vikingos y romanos, impulsadas por velas y remos, en inmensos cargueros y portaaviones, propulsados por turbinas diésel.

Pero esta tendencia de mejora continua en las características de las embarcaciones no se limita a la propulsión y el tamaño de las mismas, sino que influye a todos y cada uno de los aspectos de la navegación. Es por esto que los dispositivos de cartografía electrónica, de comunicaciones marítimas, radares y semejantes se encuentran en continuo desarrollo e implementan cada vez capacidades mayores y de más utilidad.

Por otra parte, y de acuerdo con [1], el 90% del comercio internacional se realiza a través del medio marítimo. La cantidad de abordajes que se producen anualmente entre buques, así como el hecho de que la gran mayoría de los delitos de tráfico de productos ilegales, piratería o inmigración ilegal entre otros se lleven a cabo a través de este medio, hacen evidente la necesidad de gestionar y controlar el tráfico marítimo de alguna manera.

Para esto, existen organizaciones internacionales, como la OMI (Organización Marítima Internacional), que han distribuido normas basadas en pactos y convenios, como el convenio SOLAS [2], o el RIPAM (Reglamento Internacional para la Prevención de Abordajes en la Mar) y se han establecido DST (Dispositivos de Separación de Tráfico) y marcas de navegación para proporcionar referencias en las que los buques puedan basarse para evitar colisiones y otros accidentes en la mar. Además, sistemas como el AIS permiten la gestión del tráfico marítimo desde estaciones en tierra (VTS).

Además, para la seguridad marítima, la OTAN (Organización del Tratado del Atlántico Norte), desarrolló las conocidas como Operaciones de Seguridad Marítima (MSO), en las que el objetivo es evaluar los comportamientos de los buques con el fin de proteger las rutas marítimas y detectar posibles actividades ilegales. Estas operaciones dieron lugar al concepto de MSA (*Maritime Safety Awarness*), que consiste en el conocimiento y estudio del medio marítimo para actuar en él con mayor eficiencia.

En concreto, diversos buques de la Armada española llevan a cabo constantemente numerosas operaciones MSO. Para esto, aunque existen *software* de gestión de datos AIS, ninguno realiza análisis del comportamiento de los buques según sus rutas de forma automática, y es que la IA no está, todavía, muy integrada en el ámbito naval.

1.2 Motivación y objetivos

Debido a esta carencia de sistemas automáticos de análisis del comportamiento de buques, se ha desarrollado previamente una aplicación de detección de anomalías en rutas marítimas [3], basada en un *software* de IA (Inteligencia Artificial). Esta aplicación crea modelos de comportamiento en una zona basándose en la aportación de datos históricos del tránsito de buques por la misma. Una vez creado el modelo, es capaz de procesar archivos *csv* con datos AIS que contengan un número indefinido de rutas y compararlas con él para evaluar el grado de anomalía de las mismas, generando un nuevo archivo *csv* en el que almacena estos datos.

No obstante, este programa no es capaz de procesar los datos en tiempo real, sino que es necesario que un buque complete un tránsito para poder analizarlo. Por este motivo, tomando esta aplicación como herramienta, se pretende desarrollar un programa capaz de analizar el grado de anomalía de los buques que se encuentren navegando por una zona predeterminada en tiempo real. Para ello, el programa creará un modelo de comportamiento con anterioridad, y analizará las anomalías de nuevas posiciones recibidas de un servicio VTS (2.3). Además, el programa presentará en pantalla las trazas analizadas, diferenciadas por colores según su grado de anomalía, en un *software* de cartografía electrónica, OpenCPN.

Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos:

- Estudio de las posibilidades de visualización de anomalías con OpenCPN.
- Desarrollo de un sistema capaz de procesar datos AIS recibidos de *marinetraffic* para extraer rutas efectuadas por buques en una zona determinada.
- Análisis de la anormalidad de las derrotas de los buques extraídas mediante una aplicación de detección de anomalías.
- Presentación de las trazas estudiadas en un *software* de cartografía electrónica, estableciendo una diferenciación gradual según el grado de anomalía de las mismas.
- Ejecución de todos los procesos en tiempo real.

Las pruebas realizadas se harán en el DST de Finisterre, por ser el escenario en el que se basó el desarrollo de la aplicación de detección de anomalías.

1.3 Estructura de la memoria

Detalladas las motivaciones y objetivos del proyecto, después de haber sido contextualizado, en este apartado se realiza un desglose estructural de la memoria que se desarrollará según lo indicado a continuación.

El primer capítulo ha servido de contextualización e introducción a los objetivos del proyecto, así como de orientación en cuanto al camino que se va a seguir en su desarrollo.

En el segundo capítulo se realiza un análisis del estado del arte en los campos en los que se va a basar el trabajo. En primer lugar, se introduce al lector a la IA, presentando el *software* NuPIC y su aplicación *Geospatial Tracking*, desarrolladas por la empresa Numenta. A continuación, se hace referencia a los sistemas VTS, introduciendo a *marinetraffic* como gestor de datos AIS. El siguiente apartado describe la situación actual de los programas y tipos de cartografía electrónica, explicando con más detalle el *software* OpenCPN y, por último, se presentará un breve estudio del desarrollo de las bases de datos, concretando ciertas características de interés de MySQL, el sistema de gestión de bases de datos que se empleará en el trabajo.

En el tercer capítulo se expone el desarrollo del trabajo en sí. En primer lugar, se realiza una explicación detallada de la instalación del entorno y configuración del mismo, que irá seguida de un estudio de las posibilidades de introducción de datos en OpenCPN. A continuación, y para finalizar este capítulo, se desglosa y se explica el funcionamiento del programa.

En el cuarto capítulo se muestran las pruebas realizadas, junto con un análisis de los motivos que han llevado a tomar algunas decisiones en cuanto a gestión de los datos para solucionar los problemas encontrados.

En el quinto capítulo se detallan las conclusiones extraídas de las pruebas realizadas, así como un análisis de las posibles líneas futuras que se podrían seguir en el desarrollo de este trabajo.

A continuación, se añaden cuatro anexos. En el primero se presenta un diccionario de siglas y acrónimos utilizados en el trabajo, en el segundo se describen las opciones disponibles en la llamada a la aplicación detectora de anomalías, *maritimeanomalies.py*, en el tercero se muestra un ejemplo de los archivos recibidos de *marinetraffic* y en el cuarto un ejemplo del archivo escrito por el programa desarrollado, con la ruta efectuada por la traza.

2 ESTADO DEL ARTE

En esta sección se describen las tecnologías en las que se fundamenta este trabajo, empezando por la AI (*Artificial Intelligence*, Inteligencia Artificial), más concretamente el software que se ha utilizado, NuPIC, seguido de los sistemas VTS, los programas de cartografía electrónica, en concreto el OpenCPN y las bases de datos, específicamente MySQL.

2.1 Inteligencia Artificial

2.1.1 Definición

La AI es uno de los campos más modernos de la ingeniería. John McCarthy, profesor del departamento de Ciencias Informáticas de la Universidad de Stanford, la define como la "ciencia de construir máquinas inteligentes y, más concretamente, programas de ordenador inteligentes" [4].

También otros la han definido de formas semejantes, como Patrick H. Winston, profesor de AI y Ciencias Informáticas en el Instituto de Tecnología de Massachusetts, según el cual se trata del "estudio de las computaciones que hacen posible percibir, razonar y actuar" [5]; o Philipp C. Jackson, quien la consideraba la "capacidad de las máquinas de hacer cosas que las personas dirían que requieren inteligencia" [6].

Sin embargo, la descripción con más aceptación en la sociedad científica e informática actual es la de Stuart J. Russel y Peter Norvig [7], quienes requieren cuatro objetivos claros que buscar para construir un sistema de AI que funcione correctamente:

- Que piensen como humanos.

"La automatización de actividades que asociamos al pensamiento humano, actividades como toma de decisiones, resolución de problemas, aprendizaje..." (Bellman, 1978).

- Que piensen racionalmente.

"El estudio de las capacidades mentales mediante el uso de modelos computacionales" (Charniak y McDermott, 1985).

- Que actúen como humanos.

"El arte de crear máquinas que cumplan funciones que requieren el uso de la inteligencia cuando son realizadas por personas" (Kurzweil, 1990).

- Que actúen racionalmente.

"La rama de las ciencias informáticas que se preocupa de la automatización del comportamiento inteligente" (Luger y Stubblefield, 1993)

2.1.2 Historia de la AI

La gestación de la AI comienza en 1943, cuando Warren McCulloch y Walter Pitts crean el ahora reconocido como primer trabajo de este campo, un modelo de neuronas artificiales basado en la relación entre neuronas vecinas. Sugerían incluso que un modelo configurado adecuadamente podría llegar a aprender, lo que fue demostrado por Donald Hebb en 1949, mediante una simple regla que modificase las conexiones entre neuronas.

En 1956, una de las figuras más influyentes en este campo, John McCarthy, organizó un taller de dos meses en Dartmouth, donde, además de concienciar a la sociedad científica sobre el tema, se bautizó a la AI con ese nombre. Desde entonces, el desarrollo de programas de AI fue aumentando, destacando el *Logic Theorist (LT*) y el *General Problem Solver (GPS)*, de Allen Newell y Hebert Simon, ambos de gran éxito.

Las décadas de los 50 y los 60 fueron testigos del desarrollo de diferentes programas capaces de aprender y solucionar problemas basándose en razonamientos básicos y en conocimientos, como el programa "Generador de Consejos" de McCarthy que, en 1958, explicaría su funcionamiento en un conocido artículo, *Programs with Common Sense* [8]. En 1963, McCarthy creó el Laboratorio de AI en Stanford, donde M. L. Minsky supervisaría el desarrollo de los conocidos como "micromundos", con capacidades de resolución de problemas de álgebra y geometría o de cálculo integral.

Sin embargo, y a pesar de las optimistas predicciones de Hebert Simon, el desarrollo de la AI a partir de entonces fue más lento de lo esperado, debido a que los impedimentos al resolver problemas aumentaban exponencialmente con la complejidad de los mismos y a que algunos de ellos eran inabarcables con las limitaciones de las básicas estructuras utilizadas.

En los 70 se retomó el estudio análogo de la AI y la inteligencia humana, comenzando a centrar la atención en el conocimiento, implementado en forma de reglas. La mayor implicación de filósofos, doctores y demás miembros de la comunidad científica dio pie a una actualización del enfoque desde el que estudiar la AI, y a que, partir de la siguiente década, el avance en la investigación se acelerase.

Desde entonces, la AI se ha convertido en una industria, produciendo grandes beneficios a sus explotadores, aunque no sin períodos de cierto declive. A finales de los 80, se retomaron las redes neuronales como base del aprendizaje y la memoria, y esto impulsó a la AI a reintegrarse en el ámbito de los métodos científicos, del que había sido aislada por el escepticismo de la comunidad. Los problemas surgidos llevaron a una separación del estudio en diferentes ramas, lo que permitió un avance más veloz de forma paralela en cada una de ellas.

De 1995 en adelante, los sistemas inteligentes o *bots* han sido implantados progresivamente en aplicaciones de uso habitual entre la población, tales como buscadores de internet (Google), sistemas de piloto automático (Tesla), y muchas más.

En la actualidad, desarrollan gran variedad de capacidades y utilidades, entre las que destacan la planificación y el control autónomos, la diagnosis, la planificación logística, los juegos, la robótica y el procesamiento de lenguaje y la resolución de problemas [7].

2.2 Numenta

2.2.1 Introducción

Numenta es una empresa cuyo objetivo es liderar la nueva era de la AI, el cual define en su slogan: *Leading the New Era of Machine Intelligence*. Para ello basan su trabajo en la ejecución de ingeniería inversa con el funcionamiento del neocórtex, fuente de la inteligencia humana, para comprenderlo y, de esta forma, poder crear máquinas que se comporten igual que el cerebro humano.

Figura 2-1 Logotipo de Numenta [9]

Debido a que consideran que el avance en la AI va a tener un gran impacto en los años venideros, Numenta ofrece su tecnología de forma libre, por lo que sus proyectos son abiertos y fácilmente accesibles a través de su página web [9].

2.2.2 Historia

Jeff Hawkins, ingeniero informático por la Universidad de Cornell, inspirado por una frase de Francis Crick en un artículo de la revista *Scientific American* según el cual, a pesar de todos los datos que se conocían sobre el cerebro, no se sabía en absoluto cómo funcionaba, comenzó a concienciarse con la investigación en el campo de la AI. Para él, el cerebro humano era la fuente de todo el conocimiento, y su estudio era el mayor trabajo científico y filosófico que alguien podría llevar a cabo.

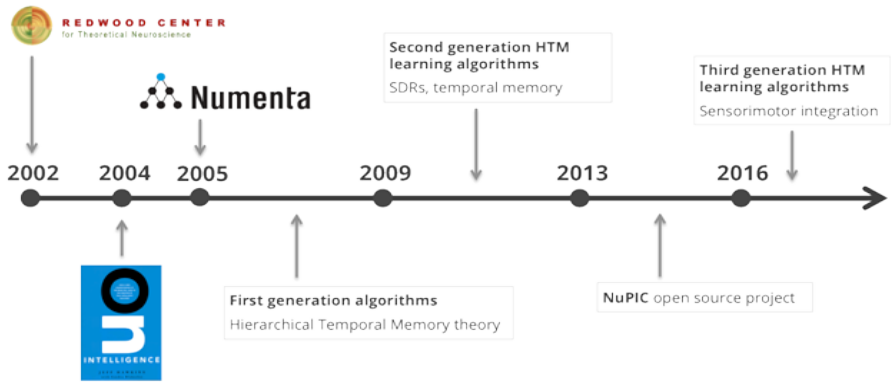

Figura 2-2 Historia de Numenta [9]

Por ello, junto con Donna Dubinsky, graduada en historia en la Universidad de Yale con un máster en Administración de Empresas por la Universidad de Harvard, y Dileep George, ingeniero eléctrico por la Universidad de Stanford, J. Hawkins fundó en febrero de 2005 la empresa Numenta en Redwood City, California.

El progreso de Numenta en cuanto a la AI puede categorizarse en tres fases (Figura 2-2):

- Pruebas en la teoría HTM y desarrollo de algoritmos de primera generación, fase orientada a la definición de una orientación de cara al futuro.
- Desarrollo de algoritmos HTM de aprendizaje de segunda generación con correlaciones biológicas más fuertes y una hoja de ruta para trabajos futuros. En esta fase se continúa explorando aplicaciones, se construye un "motor de predicción" y se comienza con el proyecto de código abierto.
- Investigación de algoritmos HTM de aprendizaje de tercera generación, actualmente centrada en la integración de la capacidad sensomotora.

2.2.3 Fundamentos de la HTM (Memoria Jerárquica Temporal)

La HTM es una tecnología de AI, desarrollada por Numenta, que pretende capturar e imitar las propiedades estructurales y algorítmicas del neocórtex, donde se basa la inteligencia del cerebro de un mamífero. Es un trabajo teórico que se encuentra en continua actualización y estudio, siendo fácilmente accesible en Internet [9], y que permite entender las numerosas capacidades del neocórtex.

En cierto modo, se trata de un tipo de red neuronal, ya que recrea modelos neuronales (llamados celdas en la teoría HTM [10]), pero se diferencia de las demás en sus características estructurales y jerárquicas. Además, como su propio nombre indica, se basa fundamentalmente en la memoria, tramitando una gran variedad de datos, patrones y secuencias por sí misma, cuyo almacenamiento gestiona de forma lógica y automática y clasifica jerárquica y temporalmente, al contrario que la memoria informática clásica, administrada directamente por el usuario.

Las capacidades de las redes HTM son generales, de forma que puede orientarse a diferentes tipos de entradas, ya sean datos sensoriales no humanos, como radar o infrarrojos, o información pura, como datos meteorológicos, patrones de tráfico o texto.

Actualmente, los modelos HTM son ejecutados por ordenadores normales incorporando las funciones clave de la jerarquía, el tiempo y la distribución dispersa, pero Numenta se cree que, con el tiempo, se creará *hardware* especializado para la construcción de redes HTM.

Los principios en los que se basa la teoría de Jeff Hawkins son los siguientes [11]:

- **Regiones**

Los biólogos dividen el neocórtex en áreas o regiones de estructura similar (generalmente 6 capas de células y una capa no celular) y tamaño variable que están organizadas jerárquicamente. Cada región recibe y procesa diferentes tipos de información.

Las regiones HTM imitan esta distribución, equivaliendo únicamente a una porción de una región neocortical, aunque capaces de deducir y predecir a partir de oleadas de datos.

- **Jerarquía**

Las regiones se organizan jerárquicamente según el grado de interconexión entre las células del neocórtex. Para imitar esta organización, las regiones HTM se interrelacionan entre ellas en orden ascendente y descendente, dando lugar a una estructura jerárquica en la que el mismo grupo de elementos en una región converge siempre en el mismo elemento de la región orgánicamente superior, produciéndose también la divergencia al desplazarnos hacia abajo en la estructura jerárquica.

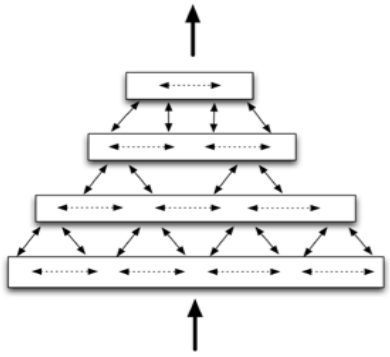

Figura 2-3 Diagrama simplificado de regiones HTM en jerarquía de cuatro niveles [11]

Esta capacidad se obtiene gracias a la interconexión y comunicación bidireccional entre los elementos dentro de cada región, así como de cada una de ellas con la inmediatamente superior y la inmediatamente inferior (Figura 2-3).

- **Representaciones Dispersas Distribuidas (SDR)**

Al igual que en el cerebro, en el que la información es siempre representada por un pequeño porcentaje de neuronas activas, el modelo HTM transforma sus *inputs* en representaciones dispersas distribuidas, de forma que, por ejemplo, una entrada de 10000 bits puede ser representada por 200 celdas activas.

Aunque pueda parecer que esto genera grandes pérdidas de información, en realidad es lo que posibilita el funcionamiento de la red HTM, además de la relación entre las celdas activas.

- **El papel del tiempo**

El papel del tiempo es crucial para el funcionamiento de las redes HTM. El aprendizaje se basa en estudio de patrones secuenciales, y en reconocer que patrón sigue a cuál, por lo que el tiempo en el que cada secuencia empieza y acaba es necesario para distinguirlas y relacionarlas entre sí y poder, de esta forma, crear modelos capaces de deducir que elemento vendrá a continuación.

2.2.4 NuPIC

NuPIC es uno de los proyectos de código abierto de Numenta, basado en la teoría HTM y que implementa algoritmos CLA (*Cortical Learning Algorithm)*. Contiene una comunidad destinada al aporte de avances en investigación o en funcionalidades por parte de los usuarios, y permite el uso del código por parte de ellos también con fines comerciales. Existen gran variedad de aplicaciones basadas en NuPIC, orientadas a reconocimiento de objetos, predicción de tráfico, seguimiento geoespacial, detección de anomalías y muchas más capacidades.

Los algoritmos CLA, y entre ellos NuPIC, tienen los siguientes componentes [12]:

- Componentes sensoriales, para recibir datos de diferentes fuentes, tales como archivos CSV, bases de datos o diferentes API's.
- Codificadores, que transforman los datos recibidos en largas cadenas binarias, relativamente dispersas, similares a las SDR.
- Una jerarquía en las regiones, cada una de las cuales aprende a identificar patrones tanto temporales como espaciales en las entradas, así como a representar las salidas en forma de SDR de columnas activas, o de predicciones de futuras entradas.
- Un clasificador que extrae información valiosa de las salidas del sistema, incluyendo la clasificación y el nivel de anomalía de la última entrada y predicciones para los próximos pasos.

Entre las capacidades de NuPIC, cabe destacar la detección de anomalías (*anomaly score*). Para obtener del modelo el nivel de anomalía, el usuario debe especificarlo como *TemporalAnomaly Model*, que calcula el grado de anomalía de cada nueva entrada según la secuencia que esté siguiendo. El valor del *anomalyScore* se computa según la siguiente fórmula [13]:

$$anomalyScore = \frac{|\, A_t \, - \, (P_{t-1} \cap A_t) \,|}{|\, A_t \,|}$$

Siendo A_t el número de columnas activas en el instante t y P_{t-1} las columnas predichas en el instante t.

2.2.5 NuPIC Geospatial Tracking

Geospatial Tracking [14] es una de las de las aplicaciones de más utilidad de NuPIC, que se basa en la teoría HTM para crear modelos con patrones de velocidad y movimiento y poder reconocer y detectar anomalías geográficas en tiempo real.

Los datos de situación GPS, velocidad y fecha y hora son codificados por el *Geospatial Coordinate Encoder*, desarrollado por NuPIC convirtiéndose cada dato en una SDR. La construcción de estas

cadenas se consigue creando una cuadrícula de bits en torno a la zona del mapa en la que se mueve el objeto de estudio.

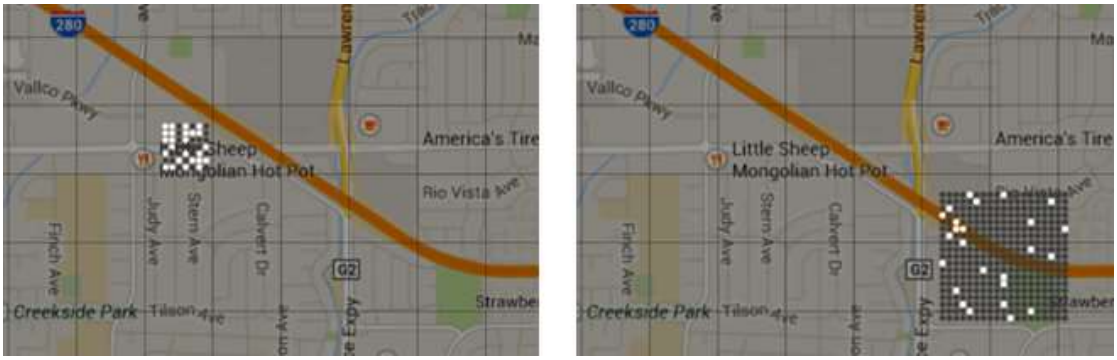

Figura 2-4 Codificación de SDR según localización y velocidad: (a) lento, (b) rápido.

Dentro de esta cuadrícula, el tamaño de cada cuadro viene definido por un parámetro del codificador (`scale`), y a cada uno se le asigna un valor numérico aleatorio entre 0 y 1. En torno a la posición del objeto, se dibuja una caja cuyo radio de cuadros depende de la velocidad a la que se mueve el contacto (Figura 2-4). Dentro de esta caja, se escogen los `w` cuadros de mayor valor, y se activan los bits correspondientes en la SDR (Figura 2-5). El número de bits activos siempre será el mismo, definido inicialmente por un parámetro del codificador (`w`).

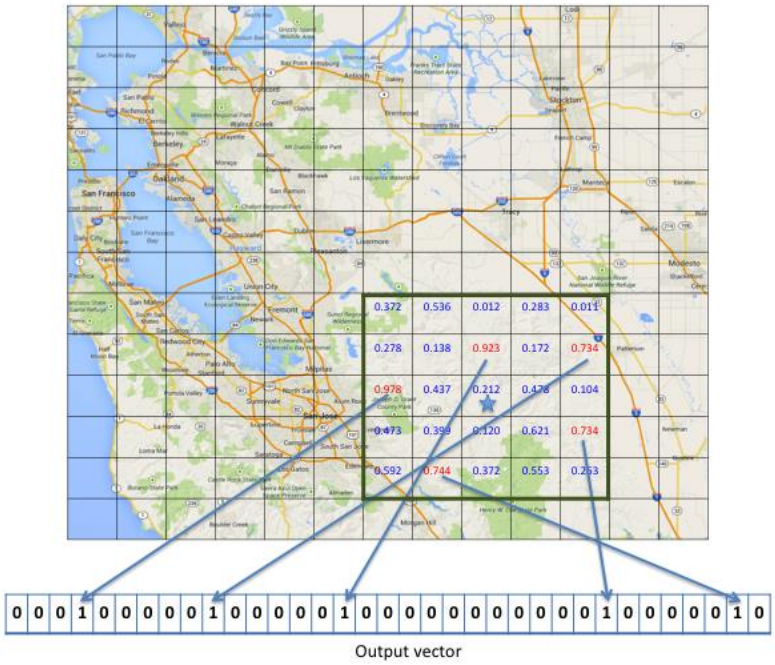

Figura 2-5 Codificación de una SDR [15]

Tras esta codificación, los datos en formato SDR son procesados por una serie de algoritmos CLA que permiten el aprendizaje de patrones de movimiento por parte del modelo, que adquiere así la capacidad de identificación de anomalías.

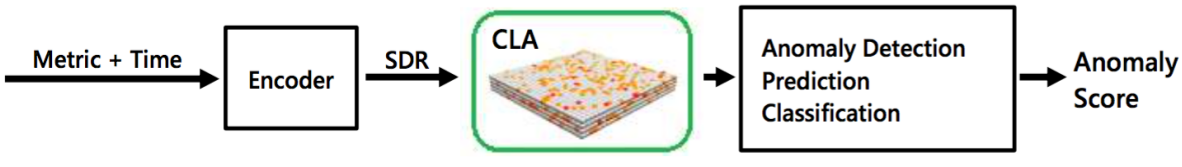

Figura 2-6 Proceso de creación de modelos de movimiento y detección de anomalías [14]

2.3 Sistemas VTS (*Vessel Traffic Services*)

2.3.1 Introducción

Los servicios VTS son herramientas para la monitorización del tráfico marítimo, basada en la gestión de datos AIS, radar, Circuito Cerrado de Televisión (CCTV), y radiofonía VHF. Son administrados en estaciones establecidas en tierra, distribuidas para dar una amplia cobertura a la zona costera.

Su función es la de proporcionar información de relevancia para la navegación en su zona de acción, como peligros, identidad e intenciones de los buques o condiciones meteorológicas; organización del tráfico mediante vigilancia de los DST (Dispositivo de Separación de Tráfico) o establecimiento de velocidades límite que considere la autoridad VTS; y la asistencia a la navegación para buques en condiciones de peligro o necesidad.

2.3.2 AIS (Automatic Identification System)

El AIS es un sistema de comunicaciones para seguridad en la navegación estandarizado por la ITU (*International Telecommunication Union*) y adoptada por la OMI, que permite que los buques que monten un transceptor puedan enviar y recibir información de otras embarcaciones o de estaciones en tierra, como su identidad internacional, tipo de barco, posición, rumbo, velocidad o estado de navegación entre otros datos, y cuya función es la ayuda en la navegación.

Existen varios tipos de dispositivos AIS, siendo los más importantes el de clase A y el de clase B. El de clase A, de mayor complejidad, permite, además del intercambio de información sobre buques, la transmisión y recepción de datos de seguridad en la navegación, así como datos meteorológicos. El de clase B es más simple y no puede transmitir datos de seguridad en la navegación, aunque sí recibirlos.

Cualquier buque de más de 300 toneladas en viaje internacional o de más de 150 toneladas y que lleve más de 12 pasajeros en viaje internacional, debe llevar montado un transceptor AIS. En el capítulo V del convenio SOLAS [2], se especifica qué tipo de dispositivo debe llevar cada uno.

2.3.3 Marine Traffic

Marine Traffic es un servicio web de gestión y presentación de datos AIS, que recibe de la red independiente de estaciones AIS terrestres más grande del mundo, de la que la ENM forma parte como contribuidora, combinada con el satélite AIS de la empresa Orbcomm [16]. Actualmente, la red de *Marine Traffic* cubre 180.000 barcos diariamente.

Esta página web cuenta con una serie de servicios para usuarios, algunos gratuitos, como la búsqueda filtrada de embarcaciones, y otros de pago, como la petición a tiempo real de datos AIS en una zona, datos particulares de un buque, datos históricos de una zona, etc. Estos últimos se solicitan mediante diferentes APIs, lo que puede hacerse de forma gratuita si se es contribuidor AIS o si se pretende utilizarlos para investigación. Para este trabajo se ha solicitado la API PS05, para la solicitud en tiempo real de posiciones de buques en una zona predefinida [17].

2.4 Sistemas de cartografía electrónica

2.4.1 Introducción a la cartografía electrónica

En la segunda mitad del siglo XX se empezaron a registrar datos hidrográficos en formato digital por parte de, entre otros servicios hidrográficos, el Instituto Hidrográfico de la Marina. En los años 80, algunos servicios hidrográficos comenzaron a considerar la posibilidad de crear cartas en formato electrónico que pudieran ser utilizadas a bordo de los buques de la misma forma que las cartas en papel.

Los sistemas de presentación de cartografía electrónica son definidos por la OHI (Organización Hidrográfica Internacional) como cualquier equipo electrónico capaz de presentar en la pantalla de un

ordenador la posición de un buque superpuesta sobre la imagen de una carta [18]. Existen, entre estos sistemas, tres tipos bien diferenciados:

- ECS (*Electronic Chart System)*.

Sistema de cartografía electrónica que se puede usar como ayuda a la navegación, pero no cumple todos los requisitos de la Organización Marítima Internacional (OMI) sobre ECDIS y no pretende cumplir los del capítulo V del convenio SOLAS [2] sobre llevar cartas.

- RCDS (*Raster Chart Display System*).

Similares a los ECS pero que utilizan cartas de tipo *raster*.

- ECDIS (*Electronic Chart Display and Information System*).

Sistema de información náutica que, con los equipos de respaldo apropiados, ha sido probado, aprobado y certificado conforme cumple con las Normas de Funcionamiento de ECDIS de la OMI, y que, por lo tanto, cumplen los requisitos del capítulo V del convenio SOLAS.

2.4.2 Formatos de cartografía electrónica

Dentro de la cartografía electrónica, encontramos como clasificación principal la que se basa en el formato de las propias cartas, que las divide en dos grupos: cartas *raster* y cartas vectoriales [19].

Los archivos cartográficos *raster* contienen únicamente imágenes georreferenciadas, es decir, asociadas lógicamente a puntos geográficos; compuestas por *pixels*, de los que se conocen sus coordenadas (latitud y longitud) y su color, dependiendo la resolución de la concentración de *pixels* por pulgada. La información que se puede obtener de estas cartas es esencialmente visual y adquirida por el usuario, ya que el software no puede hacerle saber si lo que está representando es mar, tierra, una boya o un naufragio.

Los principales tipos de cartas *raster* son las ARCS (*Admiralty Raster Chart Service*), del servicio británico de cartografía con cobertura mundial, y las BSB, de origen norteamericano.

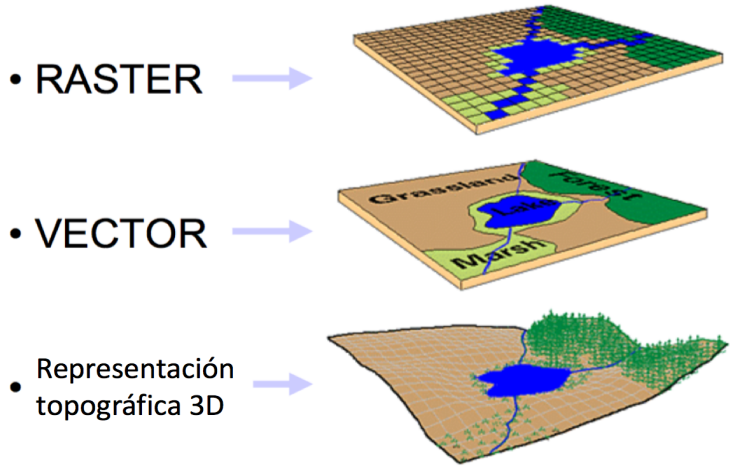

Figura 2-7 Diferencias visuales entre cartas *raster* y vectoriales

Por otra parte, los archivos vectoriales son bases de datos de mayor o menor complejidad en la que se hallan registradas las características geoespaciales de los objetos, tanto de forma como de posición, de manera que el *software* utilizado podrá representar la información de manera correcta. De esta forma, el programa gestionará información en lugar de imágenes, pudiendo alertar automáticamente al usuario cuando se aproxime a veriles bajos, naufragios, marcas de navegación o incluso a otros buques (Figura 2-7).

Los tipos de cartas vectoriales más importantes son:

- ENC (*Electronic Nautical Chart*). Realizadas por servicios estatales (formato S57), son el sustituto legal de las cartas de papel.
- DNC. Cartografía náutica del ministerio de defensa de los EEUU, en formato vectorial VPF, y de uso muy extendido en los buques de guerra.
- C-Map. Formato vectorial comercial de cobertura mundial.
- Transas. Formato vectorial comercial de cobertura mundial.

	Raster	**Vectorial**
Ventajas	Facilidad de producción Aspecto familiar para el usuario	Mayor cantidad de información Consulta automática (alarmas) Varias simbologías y colores No se ve afectada por el zoom Menos espacio en el disco
Inconvenientes	Información exclusivamente visual Deformación ante el zoom Espacio ocupado en el disco	Dificultad de producción

Tabla 2-1 Ventajas e inconvenientes de los formatos *Raster* y Vectorial [19]

En este sentido, y como se puede comprobar en la Tabla 2-1, la utilización de las cartas vectoriales es sustancialmente más ventajosa que la de cartas de tipo *raster*, y es por esto que son también las que se emplean más habitualmente.

2.4.3 OpenCPN

OpenCPN (*Open Chart Plotter Navigation*) es un *software* abierto de cartografía electrónica disponible en la web [20] que responde a la definición de los ECS, ya que no cumple con todos los requisitos del convenio SOLAS. Cumple con la función esencial de planificación y navegación con cartas náuticas en la pantalla de un ordenador. La última versión de OpenCPN, la 4.4.0, fue lanzada en julio de 2016.

Fue desarrollado por Dave Register, un informático que en torno al año 2000 comenzó a navegar con su mujer y, descontento con los sistemas de navegación comerciales, decidió crear un programa por sí mismo y para su propio uso. Años después, en 2008, decidió ofertarlo de forma gratuita en internet, situándose en la actualidad entre los ECS con más usuarios, y contando con un amplio equipo de desarrolladores voluntarios. Además, ofrece al público la posibilidad de colaborar en el desarrollo del *software* mediante la implementación de *plugins*, para el desarrollo de los cuales ofrece una API y ayuda en la web [21].

OpenCPN soporta las plataformas Windows, Linux y Mac OSX, esta última introducida con la última versión, que implementa también, entre otras utilidades, la capacidad de seguimiento de MOB (*Man Over Board*) y desarrollo del de blancos AIS, así como la visualización de derrotas y datos de radar (ARPA).

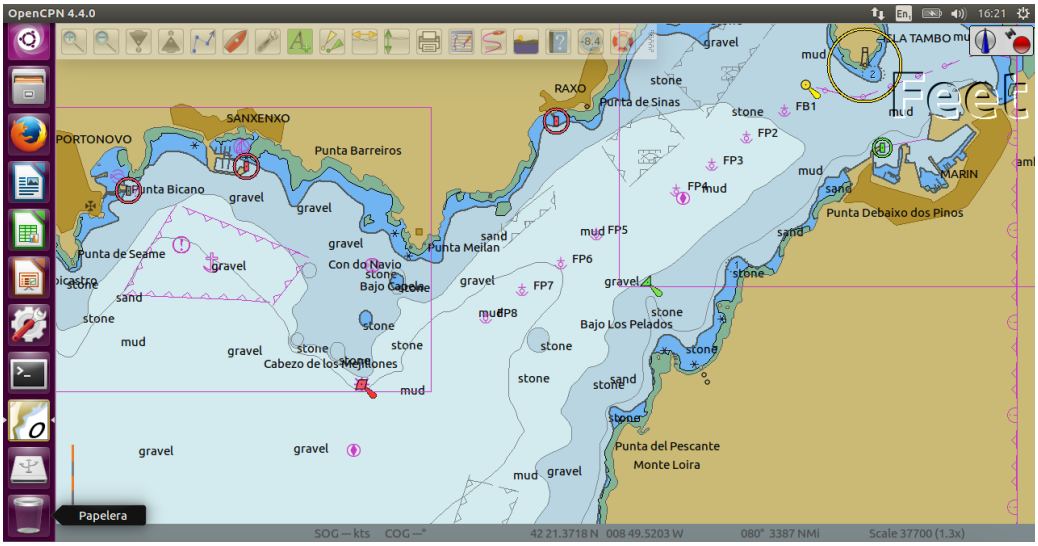

Figura 2-8 Presentación de OpenCPN, ría de Pontevedra

En cuanto a la cartografía soportada, este *software* admite los formatos *raster* BSB 2 y 3, así como las ENC S57 y S63 (esta última es una versión encriptada de la anterior), y también las C-Map, en concreto el formato gratuito de cobertura mundial CM93, que es el que se ha utilizado en el desarrollo de este trabajo [22].

La utilización del OpenCPN en navegación se basa en la entrada de datos desde un dispositivo GPS, a partir de los cuales puede registrar los datos de la ruta seguida por la embarcación. Sin embargo, con las mejoras en cuanto a la entrada de datos AIS y radar de la última versión, se ha optimizado la capacidad de configurar alertas de posibles rumbos de colisión con otros buques o de proximidad a las marcas de navegación o veriles de baja profundidad.

Además, este programa tiene su mayor utilidad en la planificación, pudiendo dibujarse rutas, que pueden administrarse intuitivamente con la función del "gestor de rutas", y también rutas activas, sobre las que, en navegación, el usuario podrá recibir recomendaciones por parte del programa, como la distancia, rumbo y ETA al próximo *waypoint*.

También, y en adición a las funcionalidades estrictamente destinadas a la navegación, OpenCPN cuenta en la barra de herramientas de la pantalla predeterminada (Figura 2-8), con opciones de funcionamiento, como la de auto-seguimiento, la visualización o no de corrientes, mareas y contactos AIS, el seguimiento automático de las trazas, la selección de modo de presentación diurno o nocturno o la de marcado de MOB.

2.5 Bases de datos

2.5.1 Introducción

En 1890, Herman Hollerith desarrolló un sistema de almacenamiento de información en tarjetas perforadas para el censo, en el que cada tarjeta recogía los datos de cada vivienda representados de forma binaria, lo que se considera como el inicio del avance tecnológico de las bases de datos. Conforme la cantidad de información que almacenar aumentaba, este sistema fue quedándose obsoleto, por lo que, a partir de los años 50, el avance se aceleró, apareciendo en 1955 una cinta magnética que podía almacenar tanta información como 10.000 tarjetas perforadas [23].

Con los avances en lenguaje informático de la segunda mitad del siglo XX, el desarrollo de este campo fue dando lugar a bases de datos en red, ordenadas y con datos independientes, y más tarde, a bases de datos relacionales en las que el cliente podía interactuar con el servidor y actuar directamente sobre los datos. Además, se trabajó en una representación gráfica más cómoda para el usuario.

En la actualidad, los *software* de bases de datos están ampliamente optimizados en cuanto a la capacidad de almacenamiento y administración de la información. Además, admiten prácticamente cualquier tipo de datos, y son, generalmente, muy intuitivos y accesibles para los usuarios.

2.5.2 MySQL

MySQL es el sistema de gestión de bases de datos de *software* libre más popular del mundo. Está programado esencialmente en lenguaje C y C++, e implementa una administración relacional de los datos. Se basa, como se puede intuir por su nombre, en el lenguaje SQL (*Structured Query Language*), que es uno de los más comunes entre los lenguajes de dominio de los sistemas de gestión de bases de datos. La última versión disponible es la 5.7.17, lanzada en diciembre de 2016.

MySQL muestra una gran variedad de capacidades, permitiendo crear tablas de gran tamaño (hasta 64 índices por tabla, y un ancho de límite de 1000 bytes) dentro de numerosas bases de datos. El propio programa aporta, entre otras utilidades, la de seleccionar datos que cumplan ciertos requisitos en una tabla, ordenar los datos según una columna u otra y, por supuesto, introducir, borrar y sustituir datos dentro de estas tablas.

Además, existe la posibilidad de acceder al programa desde otro programa escrito por un cliente cualquiera de forma muy intuitiva, lo que será de gran utilidad en el desarrollo de este trabajo.

2.6 Estudio de entrada de datos en OpenCPN

OpenCPN es un programa de código libre preparado para recibir datos de diferentes tipos de equipos, ya sean GPS, AIS, radar o muchos otros. Además, permite al usuario colaborar en la creación de *plugins* a través de una API aportada por ellos mismos. También existe la posibilidad de crear conexiones de tipo UDP o TCP, para recibir datos a través de internet, como puede verse en la Figura 2-9.

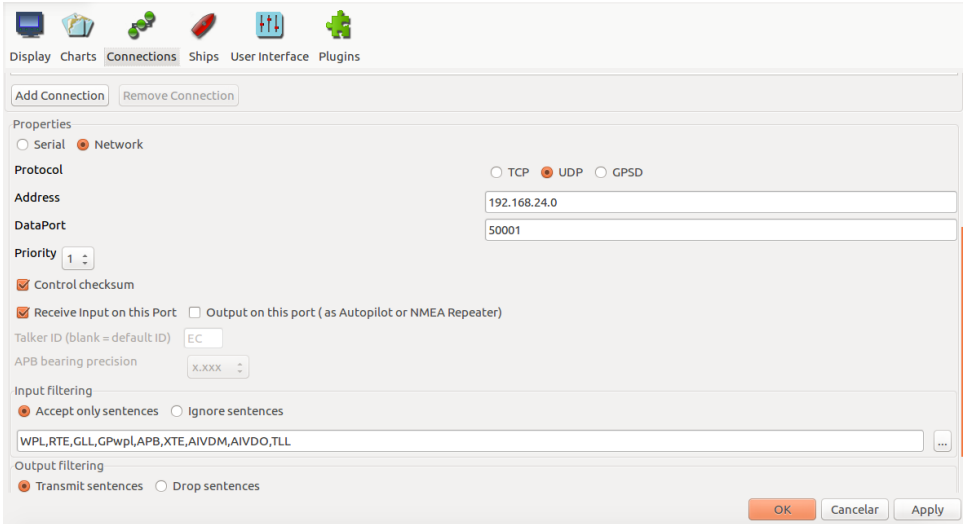

Figura 2-9 Ejemplo de conexión UDP en OpenCPN

El objetivo del trabajo es crear un programa capaz hacer un seguimiento en tiempo real de los contactos marítimos que naveguen por una zona concreta, analizar sus rutas para detectar anomalías y presentarlos en pantalla en OpenCPN de manera que puedan diferenciarse los que siguen rutas anómalas de los que no. La intención inicial es la de buscar una forma de enviar los datos al *software* de cartografía a través de la cual podamos definir el color en que van a presentarse en pantalla, creando una escala con varios colores (verde, amarillo, naranja, rojo, etc), que permitan diferenciar las trazas según su grado de anomalía. Si esto no fuera posible, otra posibilidad sería la de presentar los datos de distintas

formas según sean anómalos o no, por ejemplo, como contactos radar si no son anómalos y como *waypoints* si no lo son.

Para cumplir con este objetivo, se ha hecho un análisis de las posibilidades de este *software* cartográfico y, teniendo en cuenta la complejidad que supone la construcción de un *plugin*, y que los datos con que se trabajará serán extraídos de internet, en concreto de la web de *marinetraffic*, se ha tomado la decisión de realizar la entrega de datos a través de un enlace UDP. Se utilizarán sentencias de tipo NMEA 0183, ya que el *software* de OpenCPN ya está programado para recibirlas, procesarlas y presentarlas. Para elegir el tipo de sentencia más adecuado se ha hecho un estudio que se encuentra detallado a continuación.

2.6.1 Estándar NMEA 0183

NMEA (*National Marine Electronics Association*) es una asociación interesada en la utilización de equipos y protocolos electrónicos para la navegación. El estándar NMEA 0183 define una interfaz de datos y un protocolo de comunicación entre instrumentos de navegación. Fue lanzado por primera vez en 1983 y, desde entonces, ha estado en constante actualización por parte de la empresa. En el año 2001, la empresa lanzó el estándar NMEA 2000, mucho más avanzado y funcional, pero que no está todavía implementado en programas como Open CPN. La última versión lanzada es la 4.10.

El modelo NMEA 0183 está diseñado para permitir la comunicación entre un emisor y varios destinatarios en un solo circuito. Los datos se envían en forma de sentencias, compuestas solo por caracteres de tipo ASCII. Todas estas sentencias empiezan con el símbolo "$" y terminan con un "*" seguido del valor del *checksum* NMEA, que se explicará más adelante. Se dividen, esencialmente, en tres grupos [24]:

- **Sentencias de emisor.**

El formato de estas sentencias es el siguiente: `$ttsss,d1,d2,...,*<checksum>`, donde las dos primeras letras que siguen al "$" definen el tipo de emisor (*Talker Identifiers,* Tabla 2-2), y las tres siguientes el tipo de sentencia. A continuación, y separados por comas, se sitúan los diferentes datos en función de qué tipo de sentencia se haya elegido. Son las que se utilizarán, ya que pretendemos emitir una serie de datos.

Talker Identifier	**Significado**
AG	Piloto Automático General (*Autopilot General*)
EP	EPIRB
GP	GPS
HC	Proa (*Heading - Magnetic Course*)
RA	Radar y/o ARPA
WI	Herramientas meteorológicas (*Weather Instruments*)

Tabla 2-2 Ejemplos de *Talker Identifiers* [24]

- **Sentencias de propiedad.**

Se diferencian de las anteriores en que son desarrolladas por un particular, y en que en lugar de las dos letras que definen el tipo de emisor, se sitúa una "P" tras el "$", seguida de los datos que correspondan.

- **Sentencias de consulta.**

Utilizadas por el usuario para solicitar un tipo de sentencia en concreto de un emisor. Su formato es el siguiente: `$ttllQ,sss,*<checksum>`, siendo las dos primeras letras el tipo de emisor del solicitante y las dos siguientes el tipo de dispositivo consultado. El único campo de datos define el tipo de sentencia solicitada.

2.6.2 Sentencias Waypoint

- **WPL (*Waypoint Location*)**

```
        1       2 3       4 5    6
        |       | |       | |    |
$--WPL,llll.ll,a,LLLLL.LL,a,c--c*chksm
```

Siendo:

1) Latitud. Las dos primeras cifras son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

2) Nombre de la latitud, Norte o Sur (N o S).

3) Longitud. Las tres primeras cifra son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

4) Nombre de la longitud, Este u Oeste (E o W).

5) Nombre del WP.

6) *Checksum* NMEA.

Este tipo de sentencia permite definir la posición GPS exacta y dar un nombre al objeto, que podría ser, en nuestro caso, su MMSI. Sin embargo, OpenCPN no procesa estas sentencias como entradas, por lo que no son de utilidad en el desarrollo de este proyecto.

- **GPwpl (*GPS Waypoint Location*)**

Utiliza el mismo formato que la anterior, diferenciándose en que sólo puede hacer entrada con el tipo de emisor GP, y que el tipo de sentencia se escribe con letras minúsculas.

La diferencia esencial en cuanto a utilidad es que sí que puede ser procesada como entrada. A pesar de esto, este tipo de sentencia sólo puede ser procesada por el *software* de OpenCPN cuando existe una ruta activa para recomendar al usuario en la aproximación al próximo WP.

2.6.3 Sentencia GLL (Geographic Position - Latitude/Longitude)

```
        1       2 3       4  5       6  7
        |       | |       |  |       |  |
$--GLL,llll.ll,a,yyyyy.yy,a,hhmmss.ss,A*hh
```

Siendo:

1) Latitud. Las dos primeras cifras son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

2) Nombre de la latitud, Norte o Sur (N o S).

3) Longitud. Las tres primeras cifra son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

4) Nombre de la longitud, Este u Oeste (E o W).

5) Marca temporal (UTC).

6) Estado (A - Dato válido, V - Dato Inválido).

7) *Checksum* NMEA.

Este formato describe la posición del propio buque, por lo que no es válido para identificar a diferentes trazas.

2.6.4 Sentencias AIS

Este tipo de sentencias, aunque tienen el mismo tipo de *checksum* al final, se caracterizan por ser las únicas que comienzan con el símbolo "!" en lugar de "$".

- **AIVDM (*AIS Identification Vessel Data Message*)**

```
        1 2 3 4 5      6 7
        | | | | |      | |
!AIVDM,n,n,n,a,c---c,n*chksm
```

Siendo:

1) Número de fragmentos que componen el mensaje acumulado actual.

2) Número secuencial de fragmento dentro del mensaje acumulado actual.

3) Número secuencial de identificación si se trata de un mensaje multisentencia.

4) Definición del canal de radio AIS (A o B).

5) Datos AIS extendidos del contacto, codificados en ASCII. Un ejemplo de cómo se aportarían estos datos es el siguiente: `177KQJ5000G?TO`K>RA1wUbN0TKH`.

6) Número de bits rellenados al final del campo 5 para completar la codificación ASCII.

7) *Checksum* NMEA.

La parte codificada de este mensaje permite incluir toda la información AIS de un contacto, como la posición GPS, el MMSI, el estado de navegación, el nombre del buque o el puerto de destino. Sin embargo, la complejidad de esta codificación hace que no sea un formato muy adecuado para cumplir con el objetivo del trabajo.

Además, OpenCPN procesa estos datos según la información que contengan, implementando una diferenciación por colores en la presentación de los contactos según su nivel de identificación, por lo que se perdería la posibilidad de distinguir entre contactos que sigan rutas anómalas de los que no lo hagan.

- **AIVDO (*AIS Identification Vessel Data Ownship*)**

El formato es exactamente igual al anterior, pero es una sentencia de salida de información del propio buque, dirigida a estaciones VTS para su transmisión.

2.6.5 Sentencias de trazas en seguimiento

- **TTM (*Tracked Target Message*)**

```
        1  2   3 4   5   6   7 8   9   10 11  12  13 14
        |  |   | |   |   |   | |   |   |  |   |  |  |
$--TTM,xx,d.d,u,s.s,c.c,x.x,u,d.d,t.t,c,c---c,a,a*chksm
```

Siendo:

1) Número de traza (00-99).

2) Distancia a la traza.

3) Demora desde el propio buque.

4) Unidades de la demora.

5) Velocidad de la traza.

6) Rumbo de la traza.

7) Unidades del rumbo.

8) Distancia de CPA.

9) TCPA.

10) "-": TCPA aumentando, vacío: TCPA disminuyendo.

11) Nombre de la traza.

12) Estado de la traza.

13) R: traza utilizada como referencia, vacío: ninguna función.

14) *Checksum* NMEA

Este formato de sentencia nos permite transmitir una gran cantidad de datos asociados a una traza. Podemos asignar un número de traza, de forma que cada vez que llegue una sentencia con ese mismo número, OpenCPN sustituirá la posición anterior del contacto por la nueva, lo que nos permite hacer un seguimiento en tiempo real. También se puede definir el nombre de la misma, para lo que elegiríamos el número MMSI del buque. Permite, además, enviar los datos de rumbo y velocidad del contacto, así como el CPA y TCPA y el estado de navegación, pero el *software* cartográfico no está aún preparado para procesar y presentar este tipo de datos.

A pesar de estas capacidades, este tipo de sentencia cuenta con la desventaja de que la posición de la traza hay que definirla en relación a la del propio buque, lo que nos obligaría a tener que calcular esta posición relativa, lo que añadiría complejidad al código y sustraería la capacidad de ejecutar el programa sin conocer la propia situación GPS. Además, no permite hacer ninguna distinción por colores de las trazas que se representen, por lo que no será de utilidad.

- **TLL (Target Latitude and Longitude)**

```
          1  2        3 4      5 6     7           8 9  10
          |  |        | |      | |     |           | |  |
$--TLL,xx,llll.ll,a,LLLLL.LL,a,c--c,hhmmss.ss,a,a*chksm
```

Siendo:

1) Número de traza (00-99).

2) Latitud. Las dos primeras cifras son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

3) Nombre de la latitud, Norte o Sur (N o S).

4) Longitud. Las tres primeras cifra son los grados, las dos siguientes, los minutos y, después del punto, las décimas de minuto.

5) Nombre de la longitud, Este u Oeste (E o W).

6) Nombre de la traza.

7) Marca temporal.

8) Tipo de traza (T: *Tracking*, en seguimiento; Q: *Query*, adquisición)

9) R: traza utilizada como referencia, vacío: ninguna función.

10) *Checksum* NMEA

En este caso, la sentencia también nos permite hacer seguimiento de hasta 100 trazas simultáneamente, diferenciándolas con un número de traza y asignar un nombre a cada una. Sin embargo, al contrario de lo que ocurre con la sentencia TTM, la posición en este formato se define como latitud y longitud, lo que es conveniente ya que esos son los datos que se van a recibir y, de esta forma, no se depende de la propia posición para el buen funcionamiento del programa.

Además, el octavo campo de datos de esta sentencia nos va a dar una capacidad determinante a la hora de decidir que sentencia utilizar. Según que letra defina este valor, OpenCPN presentará la traza de una forma distinta: de color verde si la letra es la T y de color amarillo si la letra es la Q. Esto nos permitirá diferenciar visualmente y de manera rápida e intuitiva qué contactos desarrollan comportamientos anómalos y cuáles no, cumpliendo con el objetivo principal del proyecto.

2.6.6 Tipo de sentencia elegida

El objetivo del programa a realizar es el de presentar en la pantalla de OpenCPN una serie de contactos, por lo que habrá que enviarlos al programa. Es decir, actuaremos como emisores y, en consecuencia, deberemos utilizar sentencias de emisor. Entre los *Talker Identifiers* disponibles, el RA y el GP, se procesan de la misma forma para ser presentados y ambos cumplen con los requisitos que exigen los objetivos del programa. De entre los dos se ha elegido la de radar.

Para elegir el tipo de sentencia, se plantearon dos requisitos indispensables que debía cumplir la sentencia elegida. El primero era la capacidad de cumplir con el objetivo de presentar diferenciadamente los contactos según el grado de anomalía de su derrota y el segundo el de permitir la identificación del buque con su MMSI.

Las sentencias WPL, AIVDM, TTM y TLL cumplen con la segunda condición, ya que permiten asignar un nombre al dato de entrada, que sería el número MMSI, con el que podríamos identificar a cualquier barco. Sin embargo, de ellas, sólo la TLL nos permite cambiar a nuestro parecer el color en que será presentada la traza.

Es por esto que se ha elegido el tipo de sentencia TLL que, aunque no permite que se haga una diferenciación gradual de la anomalía, si nos permite reconocer visualmente y de forma rápida qué contactos superan el nivel de anomalía predefinido por el programa.

Por lo tanto, el formato de sentencia seleccionado para la entrada de datos en OpenCPN será el siguiente:

```
$RATLL,xx,llll.ll,a,LLLLL.LL,a,c--c,hhmmss.ss,a,a*chksm
```

2.6.7 Checksum NMEA

El *checksum* NMEA, como cualquier *checksum* en cualquier tipo de transmisión de datos, tiene la finalidad de confirmar al receptor que el mensaje que ha recibido tiene el contenido correcto. Depende, por tanto, del contenido del mensaje, y servirá al receptor para descartar aquellos cuyo valor de comprobación correcto sea distinto del que se ha recibido, pero no para corregirlos.

En el caso de NMEA, se trata de dos cifras hexadecimales calculados mediante un cifrado basado en el operador binario XOR, que se aplica con una clave a todos los elementos de la sentencia, a excepción del "$" o "!" inicial y del "*" final.

3 DESARROLLO DEL TFG

3.1 Diseño de la aplicación

En este apartado se definirá brevemente el funcionamiento de la aplicación, así como los problemas que se han presentado para su desarrollo, y como se han solucionado.

3.1.1 Funcionamiento de la aplicación

La aplicación deberá ser capaz de leer una serie de datos AIS de una zona geográfica predeterminada, proporcionados por una API de *marinetraffic* y de procesarlos uno a uno para detectar su grado de anomalía, utilizando un programa previamente desarrollado [3], y construir una sentencia NMEA de tipo TLL que se envíe posteriormente al programa de cartografía electrónica OpenCPN. Este programa presentará las trazas de forma diferenciada según su grado de anomalía, verde si no se considera anómala y amarillo en caso de que sí que se considere anómala.

Todo este proceso se repetirá continuamente, para que cumpla la función de presentar los datos de la zona en tiempo real.

3.1.2 Problemas presentados

Los problemas que ha habido en cuanto al desarrollo del programa se centran, básicamente, en que es necesario procesar los datos de dos formas distintas, es decir, la aplicación detectora de anomalías procesa los datos en un formato distinto al que se va a emplear en la construcción de la sentencia TLL. Además, la aplicación detectora de anomalías suele generar una falsa alarma en las posiciones iniciales de cada ruta.

Los problemas presentados se encuentran definidos a continuación:

- Diferencias de formato entre los datos recibidos de la API, la sentencia TLL y los datos tratados por la aplicación detectora de anomalías.

Los datos AIS recibidos de la API de *marinetraffic* tienen el formato habitual de este tipo de mensajes, compuestos por siete columnas con los datos, en este orden, de MMSI, latitud, longitud, velocidad, rumbo, estado y marca temporal, como se ve en el Anexo III. Un ejemplo se encuentra a continuación:

355463000,43.047500,-9.804667,96,4,0,2017-03-06T22:29:54

La sentencia TLL se escribe, como se ha explicado en el apartado 2.6, con datos bastante distintos, aunque basados en los anteriores. Es de destacar que se descartan los datos de rumbo, velocidad y estado, y que es necesario introducir un número de traza, una letra para determinar el color y el

checksum, además de variar el formato en que se introduce la latitud, la longitud y la marca temporal. A continuación, se muestra la sentencia que se enviaría como resultado del procesado de esta traza:

```
$RATLL,00,4328.50,N,00948.28,W,355463000,222954.00,T*0C
```

Por último, la aplicación detectora de anomalías tramita los datos en el mismo formato que se reciben de la API. Sin embargo, es necesario pasarle todos los datos registrados de solamente una traza, por lo que el archivo procesado por la aplicación no puede ser el que se recibe, sino que se tiene que escribir uno nuevo con este registro de datos. Un ejemplo de este archivo se encuentra en el Anexo IV, y tienen el mismo formato que el de los datos AIS recibidos de la API.

- Falsa alarma en la detección de anomalías.

La aplicación detectora de anomalías tiene en cuenta las posiciones anteriores de la traza al realizar el análisis de su grado de anomalía, por lo que, al enviar la primera posición de una traza a la aplicación, esta devuelve un grado de anomalía muy elevado, por no disponer de una referencia anterior de la ruta. Para que el grado de anomalía detectado sea preciso, es necesario que la ruta procesada tenga, más de tres posiciones.

3.1.3 Solución de los problemas

Para solucionar los problemas definidos en el apartado anterior, se han llevado a cabo las decisiones que se presentan a continuación, referidas cada una al problema que solucionan:

- Diferencia en el formato de datos.

Dado que es posible que una traza sea recibida más de una vez, no pueden introducirse todos los datos recibidos en una base de datos automáticamente sin comprobar si ya han sido almacenados. Para comprobar esto se tramitan los datos previamente, por lo que pueden procesarse en este punto para adecuarlos a los formatos establecidos.

Para esto, se ha tomado la decisión de almacenar los datos en dos tablas distintas dentro de la base de datos creada, una que almacene los datos en el formato en el que llegan, para la detección de anomalías, y otra en el formato adecuado para la construcción de la sentencia TLL.

Para construir la sentencia NMEA, hay que procesar los datos de latitud, longitud y marca temporal, además de deducir el número de traza, el nombre de la latitud (N o S) y el de la longitud (E o W), la marca que definirá el color de la traza, que dependerá de su grado de anomalía, y el cálculo del *checksum*. Este procesado de datos se presenta en el apartado 3.3.4, detallado para cada dato procesado y deducido, a excepción del de la marca de color de la traza y el del cálculo del *checksum*, que se definen en el apartado 3.3.8.

- Falsa alarma de anomalía.

Para evitar este problema, se ha tomado la sencilla medida de no procesar las trazas cuyas rutas registradas sean demasiado cortas. Para eso, el programa comprobará la longitud del archivo contenedor de la ruta estudiada antes de procesarlo para detectar su grado de anomalía. Si comprueba que se han registrado más de 2 posiciones de la traza además de la estudiada, ejecutará la aplicación de detección de anomalías sobre la ruta. Si, por el contrario, comprueba que no se han registrado suficientes posiciones, se salta el paso del análisis y la presenta como no anómala.

3.2 Instalación y configuración del entorno

3.2.1 Sistema operativo

En este trabajo, principalmente por el motivo de que la plataforma NuPIC de Numenta ha sido desarrollada desde sus inicios en Ubuntu e IOS y por la facilidad y accesibilidad de este sistema operativo, se ha tomado la decisión de trabajar en Ubuntu, concretamente en la versión 16.04, instalada en una máquina virtual (VirtualBox).

3.2.2 NuPIC

El proceso de instalación de NuPIC se encuentra detallado en la Wiki de NuPIC en Github [24], así como en su vídeo tutorial [25]. También se puede seguir en el TFG en el que se basa este trabajo [3], llevado a cabo por el Alférez de Navío Carlos Arenas Pérez-Seoane.

Las dependencias requeridas para la instalación se encuentran detalladas en el Github de NuPIC [26] y son las siguientes:

- Python 2.7
- pip
- setuptools
- wheel
- numpy
- C++ compiler

Ubuntu cuenta ya con un compilador C++, el gcc.

Para instalar Python 2.7, en caso de que no se haya hecho anteriormente, se pueden seguir los pasos de [27]:

```
hg clone https://hg.python.org/cpython
hg update 2.7
sudo apt-get build-dep python2.7
sudo apt-get install python-dev
```

La instalación de pip, setuptools y wheel se llevará a cabo introduciendo el siguiente comando en el terminal:

```
curl https://bootstrap.pypa.io.get-pip.py | sudo python
```

Una vez finalizada la instalación, el sistema imprimirá en pantalla lo siguiente:

```
Successfully installed pip-8.0.2 setuptools-20.1.1 wheel-0.29.0
```

La versión actual de NuPIC, la 0.5.5, ha incluido la instalación de los *bindings* [3] en la instalación general del *software*, por lo que el siguiente paso será el de instalar NuPIC como tal, utilizando el siguiente comando:

```
pip install nupic --user
```

Al finalizar la instalación, el sistema devolverá un listado de las dependencias instaladas, entre las que se incluye numpy, los anteriormente mencionados *bindings*, o PyMySQL.

Llegados a este punto, NuPIC estará instalado. Para comprobar que la instalación se ha llevado a cabo correctamente, se van a ejecutar una serie de tests que se encuentran en el directorio de NuPIC en Github. Para obtener estos tests, instalaremos git y clonaremos el directorio anteriormente nombrado.

```
sudo apt-get install git
git clone https://github.com/numenta/nupic.git
```

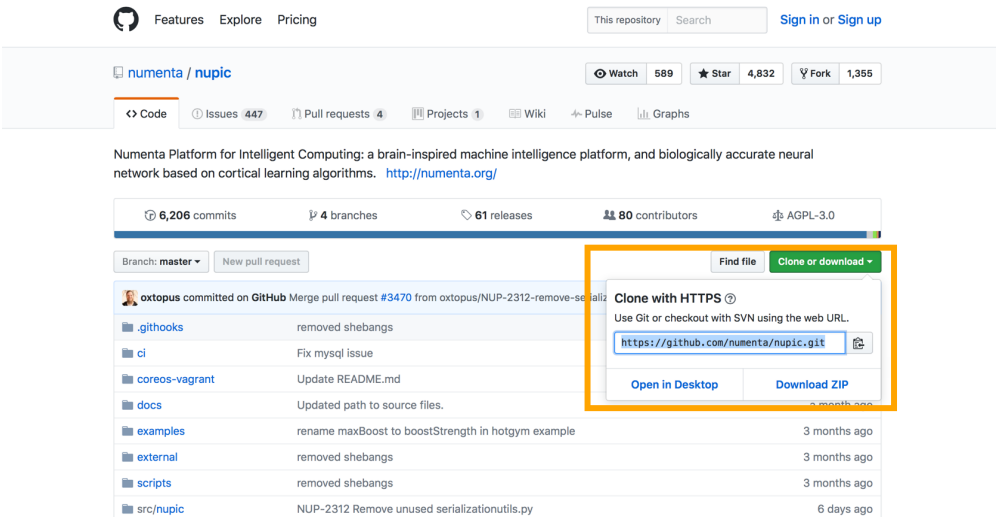

Figura 3-1 Ubicación del directorio de NuPIC en Github

Una vez clonado el directorio, llamado "nupic", para correr los tests, se ejecutan las siguientes líneas de comandos:

```
cd nupic
export NUPIC=`pwd`
export PATH=~/.local/bin:$PATH
./scripts/run_nupic_tests.py -u
```

En pantalla, los tests se imprimirán conforme sean pasados, junto con su nombre y resultado. Es habitual que algunos se salten, a pesar de que la gran mayoría se pasen sin ningún problema. Al finalizar todos, el sistema devuelve en el terminal el número de tests pasados y saltados, así como el tiempo que se ha tardado en llevarlos a cabo.

A continuación, y para poder hacer uso de la aplicación *Geospatial Tracking*, en la que se basa el programa de detección de anomalías, debemos clonar el directorio de Github de esta aplicación e instalar sus requerimientos, usando los siguientes comandos:

```
git clone https://github.com/numenta/nupic.geospatial.git
pip install -r requirements.txt
```

El proceso completo es susceptible de cambiar por el desarrollo de actualizaciones en NuPIC, o incluso requerir alguna actualización en Ubuntu.

3.2.3 OpenCPN

Para la instalación de este *software* de cartografía electrónica, se han seguido las indicaciones de los desarrolladores de OpenCPN en su página web [28]. Se realiza a través de una PPA. Las instrucciones se pueden encontrar en el Manual de Usuario de OpenCPN [29], y se reproducen a continuación:

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:opencpn/opencpn
sudo apt-get update
sudo apt-get install opencpn
```

Finalizada la instalación, para poder utilizar el programa eficientemente es necesario instalar la cartografía. En este caso, se han utilizado, como se explicó anteriormente, las cartas CM93 del formato C-Maps, por ser gratuitas y estar ya disponibles.

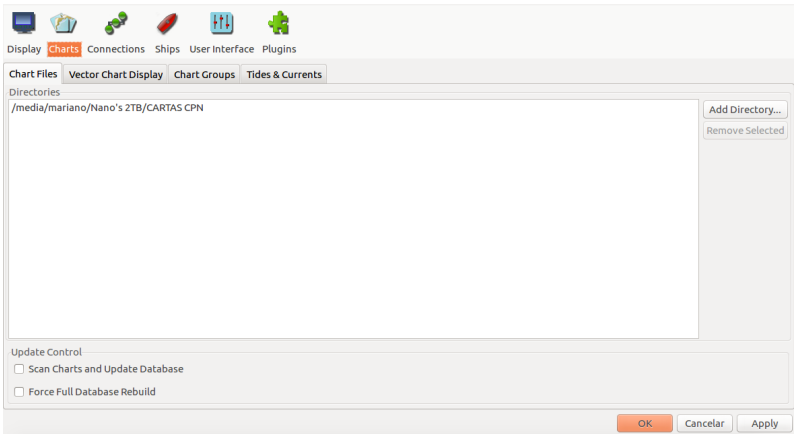

Figura 3-2 Instalación de las cartas en OpenCPN

Para instalarlas, es necesario entrar en la ventana configuración de OpenCPN, concretamente en la pestaña *Charts* y, una vez allí, pulsar *Add directory* y seleccionar la carpeta donde se encuentran almacenadas las cartas, como se observa en la Figura 3-2. A continuación, pulsaremos *Apply*, y el programa leerá las cartas y las presentará en la pantalla principal, pudiéndose hacer ya uso de ellas.

Además de la cartografía, necesitaremos configurar una conexión predeterminada que se encuentre activada siempre que ejecutemos el programa. Para ello, entraremos en la ventana de configuración, en la pestaña *Connections*, pulsaremos en añadir conexión y configuraremos los datos de la conexión:

- *Properties: Network*
- *Protocol:* UDP
- *Address:* 192.168.24.0
- *Dataport*: 50001
- *Control checksum*: Activado
- *Receive input on this Port*: Activado
- *Input filtering*: *Accept only sentences*: Activado. Se incluirá en el campo de texto la sentencia TLL

El resultado será semejante al que se observa en la Figura 3-3.

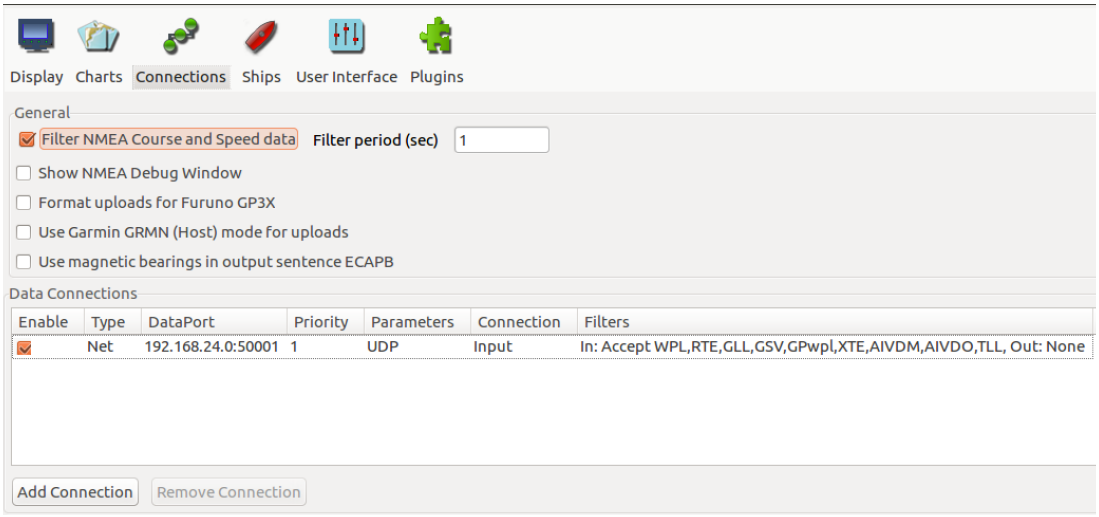

Figura 3-3 Conexión UDP en OpenCPN

Por último, en la ventana de configuración, en la pestaña *Ships*, activaremos los campos de *Remove lost target after (min)*, estableciendo un valor de 10, *Show target COG predictor arrow, length (min)*, con 5 minutos, y *Show target tracks, length (min)*, con 30 minutos. De esta forma se presentarán en

pantalla el vector de velocidad sobre el fondo y la derrota seguida por las trazas, y se borrarán de la misma las trazas que lleven más de 10 minutos sin ser actualizadas.

Llegados a este punto, OpenCPN estará configurado correctamente para el funcionamiento del programa.

3.2.4 MySQL

Para instalar el *software* gestor de bases de datos de MySQL solamente es necesario introducir el siguiente comando en el terminal:

```
sudo apt-get install mysql-server
```

Una vez instalado, será necesario crear una base de datos, que será la que utilizaremos para almacenar la información que se reciba de los contactos en estudio. Para ello accederemos desde el terminal con el siguiente comando:

```
mysql -u root -p
```

El sistema solicitará una contraseña si se introduce la opción `-p`, que en este caso se ha establecido como `marianotfg`.

A continuación, se creará la base de datos, que en este caso se llamará "AIS_register", usando el siguiente comando:

```
create database AIS_register;
```

El siguiente paso será creas las tablas que se deseen utilizar. Para el desarrollo de este trabajo se van a utilizar dos tablas. La primera, de nombre "mmsi", almacenará los datos de cada traza registrada en el formato adecuado para la escritura del mensaje que se enviará a OpenCPN. La segunda almacenará también los datos de cada traza, pero en el formato adecuado para su posterior procesado por la aplicación de detección de anomalías. Se crearán empleando los siguientes comandos:

```
create table mmsi (track int, latdeg varchar(2), latmin varchar(20),
                   latnom char, londeg varchar(3), lonmin varchar(20),
                   lonnom char, mmsi varchar(9), datetime varchar(6));

create table AIS_reg(MMSI varchar(9), LAT varchar(10), LON varchar(12),
                     SPEED varchar(3), COURSE varchar(3), STATUS int,
                     TIMESTAMP varchar(20));
```

Dentro de los paréntesis se definen, entre comas, los nombres de las columnas y el tipo de variables que irán dentro de cada una de ellas, separadas por un espacio.

Al finalizar este proceso, todo el entorno del programa habrá sido creado, y se podrá ejecutar el programa.

3.3 Ejecución del programa

En este apartado se desglosará el código del programa, explicando de forma breve las acciones que lleva a cabo cada una de las funciones que lo componen y, en algunos casos, los motivos por los cuales se han tomado ciertas decisiones en cuanto a la forma de ejecutar estas acciones.

En la Figura 3-4 se muestra un esquema del funcionamiento básico de esta aplicación que, al ser inicializada, se conecta a la base de datos para vaciarlas y empezar un nuevo registro. Una vez hecho esto, inicia un bucle temporal, que depende de la variable *time*, implementada con valor 0. Dentro de este bucle, el programa comienza haciendo una llamada a la API de *marinetraffic*, de la que extrae el archivo *AISdata.csv*, que contiene los datos AIS de las trazas presentes en la zona. Uno por uno, procesa estos datos, comprobando si ya han sido registrados en las bases de datos y, en caso negativo, los registra en las tablas de la base de datos MySQL, analiza su grado de anomalía haciendo una llamada a la aplicación *maritimeanomalies.py*, y escribe la sentencia TLL y el mensaje, enviando este último a

OpenCPN. Una vez enviado, o en caso de se compruebe que la traza ya ha sido registrada anteriormente, el programa pasa a procesar la siguiente traza, repitiendo el proceso descrito, hasta que termina de leer el archivo *csv*.

Finalizado este proceso, el programa espera 60 segundos y lo reinicia, desde la llamada a la API de *marinetraffic* hasta el envío del mensaje correspondiente a la última traza registrada, que se repite continuamente, debido a que en ningún momento se modifica el valor de la variable *time*.

En los próximos apartados, se desglosa el funcionamiento de cada una de las funciones que permiten el buen desarrollo de este programa.

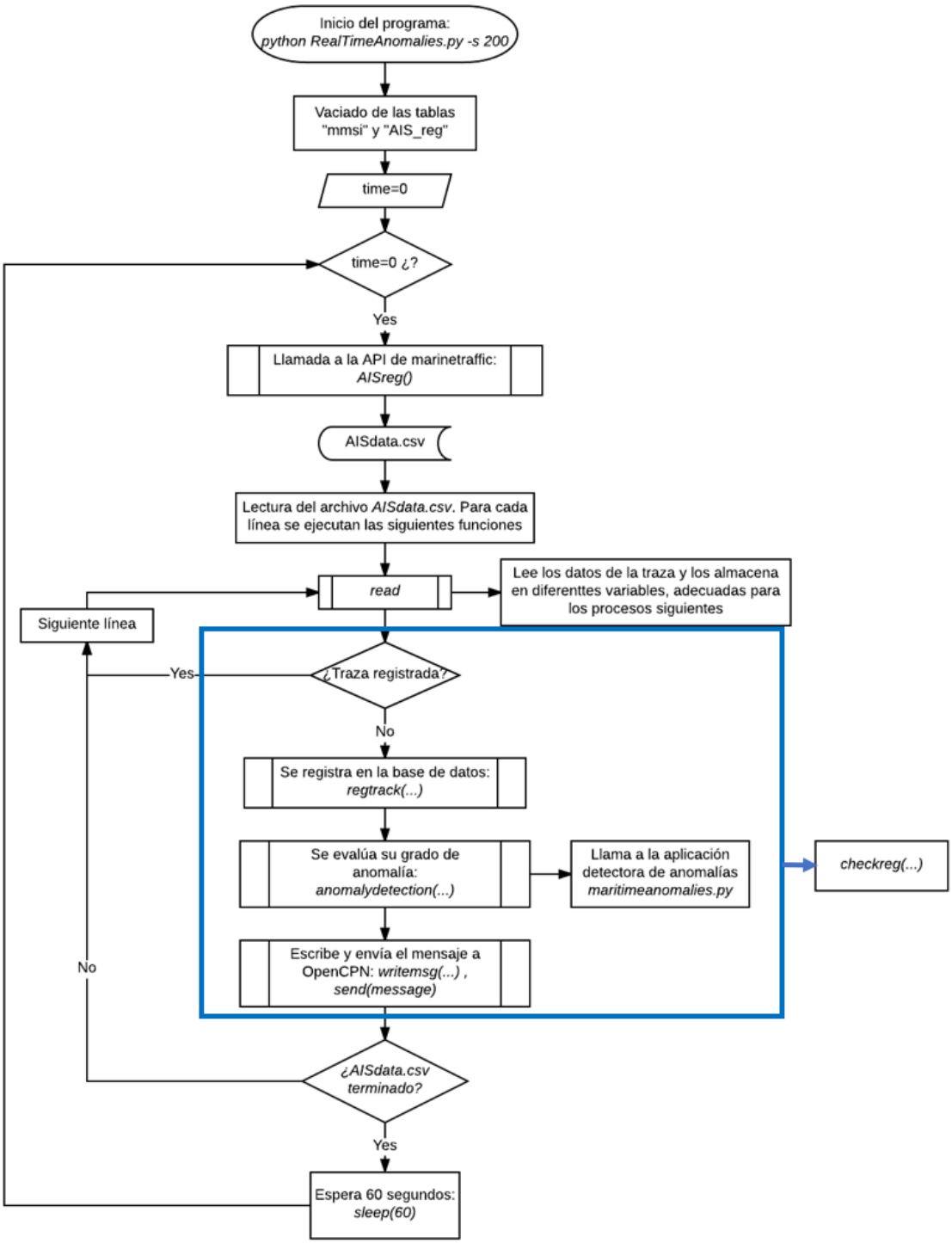

Figura 3-4 Diagrama de flujo conceptual del programa

3.3.1 Creación del modelo

Previamente a la llamada al programa, será necesario crear un modelo para detectar las anomalías de los datos que se estudien al ejecutar el programa. Para ello, solamente será necesario llamar directamente a la aplicación *maritimeanomalies.py* introduciendo la opción `-i`, que iniciará un nuevo modelo. Los datos introducidos para la creación del mismo fueron los contenidos en un fichero *csv*, llamado *exportvessels.csv*, que comprendían un registro histórico extendido de tráfico en la zona del DST de Finisterre. El comando introducido sería el siguiente:

```
python maritimeanomalies.py exportvessels.csv -i -s 200
```

El archivo contiene más de 40000 líneas, por lo que el procesado de los mismos duró aproximadamente una hora, debido a la menor velocidad de procesado de la máquina virtual.

3.3.2 Llamada al programa

Para llamar al programa, es necesario hacerlo desde el directorio en el que este se encuentra. Para ejecutar la aplicación, es necesario indicar previamente que se va a utilizar Python. A continuación, se introduce el nombre del programa, seguido de las opciones que se desee introducir, cuyas funciones se detallan en el Anexo II.

En este caso, se introduce solamente la escala, con el valor 200, que es el que fue seleccionado como óptimo para la ejecución de la aplicación *maritimeanomalies.py*:

```
python RealTimeAnomalies.py -s 200
```

A continuación, se describe el funcionamiento general del programa, que será desglosado y explicado con mayor detalle en los siguientes apartados.

Este archivo, en primer lugar, importa las librerías *csv*, *MySQLdb*, *operator*, *socket*, *urllib2*, *os* y *sys*, además de las funciones *OptionParser* de la librería *optparse*, *sleep* de la librería *time*, *maritimeanomalies* de *maritimeanomalies.py* y *writeinput* de *Nupicformatdb.py*. A continuación, se conecta al servidor de bases de datos MySQL y vacía las tablas *mmsi* y *AIS_reg* para comenzar un nuevo registro de trazas.

Acto seguido, abre un bucle *while* que depende de la variable *time*, establecida a 0 anteriormente. Dentro de este bucle, hace la llamada a la API de *marinetraffic*, escribiendo los datos en un archivo *csv*, llamado *AISdata.csv*. que comienza a ser leído mediante la función *read*.

Para cada traza recibida se realiza una comprobación, definida en la función *checkreg*, para cerciorar que no se haya registrado anteriormente y, si es así, se registra en las dos bases de datos y se escribe un archivo *csv* con todas las trazas que han sido almacenadas anteriormente con el mismo número MMSI, llamado *trackroute.csv*.

Este archivo es el que sirve de entrada para la función de detección de anomalías, *maritimeanomalies*, que se llama a continuación. Esta función procesa la ruta del contacto y devuelve un archivo *csv* en el que incluye un valor de grado de anomalía para cada posición de la ruta, llamado *anomaly_scores*.

El programa principal extrae del fichero el valor de anomalía de la última posición, que es la que se está procesando, y en función del mismo decide el color en que va a presentar la traza. Una vez decidido, construye el mensaje y lo envía a OpenCPN, que presenta el contacto en pantalla.

A continuación, procesa una por una las demás trazas que se han recibido, de la manera arriba descrita. Cuando termina de procesarlas, espera 60 segundos para reiniciar el bucle que constituye la base del funcionamiento del programa en tiempo real.

3.3.3 Adquisición de datos

Para la adquisición de datos, el programa realiza una llamada a la API PS05 de *marinetraffic*. Para ello se utiliza la librería *urllib2*, abriendo primero la dirección URL mediante la función *urlopen*, y creando a continuación la variable "salida", en la que se escribirá su contenido con la función *read()* (Figura 3-5).

```
def AISreg():
        #Llamada a la API de marinetraffic
        url = urllib2.urlopen('http://services.marinetraffic.com/api/exportvessels/
                        '78ea04b7152ed4aa0b2771a2a4dd5765c3d34470/timpespan:2/protocol:csv')

        salida=url.read()
        #Escritura del archivo 'salida.csv'
        with open('AISdata.csv', 'w') as p:
                for row in salida:
                        p.write(row)
```

Figura 3-5 Función de adquisición de datos AISreg()

A continuación, se abrirá el archivo *AISdata.csv* para su escritura y se abrirá un bucle *for* que leerá los datos de la variable "salida" y los introducirá en el archivo.

3.3.4 Procesado inicial de los datos de cada traza

Para hacer la lectura de datos, el programa abre el archivo *AISdata.csv* para lectura, con la función *open* y la utilidad *rb*. A continuación, abre un bucle de lectura de filas del archivo, de tal forma que para cada una de ellas, que corresponde a un solo registro de una traza, ejecuta la función *read*.

En esta función, se extraen directamente los datos de MMSI, velocidad, rumbo y estado. Sin embargo, llama a las funciones *latitude()*, *longitude()*, y *timereader()* para obtener la latitud, la longitud y la marca temporal en los formatos adecuados, y a la función *tracknumber(mmsi)*, para seleccionar el número de traza en función del MMSI (Figura 3-6).

```
def read(prueba):
        #mmsi
        mmsi=(row['MMSI'])
        #speed
        speed=(row[' SPEED'])
        #course
        course=(row[' COURSE'])
        #status
        status=(row[' STATUS'])

        #llamada a funcion de la latitud
        latvalue,latdeg,latmin,latnom=latitude()

        #llamada a funcion de la longitud
        lonvalue,londeg,lonmin,lonnom=longitude()

        #llamada a funcion del datetime
        datetime,timestamp=timereader()

        #llamada a funcion de track number
        trackN=tracknumber(mmsi)

        return mmsi,speed,course,status,latvalue,latdeg,latmin,latnom,lonvalue,londeg,lonmin,lonnom,datetime,timestamp,trackN
```

Figura 3-6 Función de lectura de datos de *AISdata.csv*

Las funciones de lectura y procesado de la latitud y la longitud son muy parecidas. En ellas se extrae el valor del archivo *csv*, para luego deducir el nombre de la latitud (Norte o Sur) y el de la longitud (Este u Oeste). A continuación, se procesan los valores para extraer por separado cuatro variables que cada función devolverá: los valores decimales (*latvalue* y *lonvalue*), los grados (*latdeg* y *londeg*), los minutos (*latmin* y lonmin) y los nombres (*latnom* y *lonnom*) (Figura 3-7).

```python
def latitude():
        #latitud en formato float
        latvalue=(row[' LAT'])
        lat=float(latvalue)
        #elegimos norte (N) o sur (S) segun latitud positiva o negativa
        if lat > 0 :
                latnom='N'
        else:
                latnom='S'
        #Construccion de string de latitud en formato para sentencia TLL
        latval=abs(lat)
        latdeg=abs(int(lat))
        latmin=(latval-latdeg)*60
        if latmin<10:
                latmin='0%(latm)f' %\
                        {"latm":latmin}
        else:
                latmin='%(latm)f'%\
                        {"latm":latmin}
        return latvalue,latdeg,latmin,latnom
```

```python
def longitude():
        #longitud en formato float
        lonvalue=(row[' LON'])
        lon=float(lonvalue)
        #elegimos este(E) u oeste(W) segun longitud positiva o negativa
        if lon > 0 :
                lonnom='E'
        else :
                lonnom='W'
        #Construccion de string de longitud en formato necesario para sentencia TLL
        lonval=abs(lon)
        londeg=abs(int(lon))
        lonmin=(lonval-londeg)*60
        if lonmin<10:
                lonmin='0%(lonm)f' %\
                        {"lonm":lonmin}
        else:
                lonmin='%(lonm)f'%\
                        {"lonm":lonmin}
        return lonvalue,londeg,lonmin,lonnom
```

Figura 3-7 Funciones de procesado de latitud (a) y longitud (b)

En cuanto a la marca temporal, es necesario extraerla también en dos formatos, uno para el mensaje y otro para la detección de anomalías. Para ello se van a extraer dos variables: *datetime* para la construcción de la sentencia TLL (`hhmmss.ss`) y *timestamp* para su procesado por parte de la función *maritimeanomalies* (`aaaa-mm-ddThh:mm:ss`) (Figura 3-8).

```python
def timereader():
        #Datetime
        timestamp=(row[' TIMESTAMP'])

        datetime='%(dt1)c%(dt2)c%(dt3)c%(dt4)c%(dt5)c%(dt6)c.00' %\
            {"dt1":timestamp[11],"dt2":timestamp[12],"dt3":timestamp[14],
             "dt4":timestamp[15],"dt5":timestamp[17],"dt6":timestamp[18]}
        return datetime,timestamp
```

Figura 3-8 Función de procesado de la marca temporal

Por último, se elige el número de traza, y para ello se lleva a cabo una comprobación en la base de datos "mmsi" de si se ha registrado ya alguna traza con el mismo número MMSI. En caso afirmativo, como se ve en la Figura 3-9, se escoge el número de traza asignado a ese MMSI y, en caso negativo, se selecciona el número de traza más alto registrado y se le asigna el siguiente al contacto.

```python
def tracknumber(mmsi):
        #Comprobacion de registro de traza segun su mmsi para elegir su track number
        cur=con.cursor()
        cur.execute("select count(*) from mmsi where mmsi=%s" % mmsi)
        exist=cur.fetchone()
        exist=exist[0]

        #Si no hay ningun contacto registrado con el mmsi recibido,
        #elegimos el numero de traza siguiente al ultimo registrado
        if exist == 0:
                with con:
                        cur.execute("select max(track) from mmsi")
                        maxtrack=cur.fetchone()
                        lasttrack=maxtrack[0]
                        #Si no es la primera traza registrada,
                        #se suma 1 al numero de la ultima traza registrada
                        if lasttrack is not None:
                                trackN=lasttrack+1
                        #Si es la primera traza, se le asigna el numero de traza 1
                        else:
                                trackN=1
        #Si ya se ha registrado alguna vez un contacto con este mmsi,
        #copiamos su numero de traza
        else :
                with con:
                        cur.execute("select track from mmsi where mmsi=%s" % mmsi)
                        Nowtrack=cur.fetchone()
                        trackN=Nowtrack[0]
        return trackN
```

Figura 3-9 Función de selección de número de traza

3.3.5 Comprobación de registro

Una vez procesados los datos de la traza, se llamará a la función *checkreg*, que comprobará si ya ha sido registrada en la base de datos y, en caso afirmativo, la descarta y el programa aborda el procesado de la siguiente traza. Si, por el contrario, la traza no ha sido registrada, se llama, por este orden, a las

funciones *regtrack*, para introducirla en la base de datos; *maritimeanomalies*, con ciertas excepciones, como se explicará más adelante, para analizar el grado de anomalía de la ruta; *writemsg*, para la construcción de la sentencia NMEA TLL, y *send* para el envío de la misma a OpenCPN (Figura 3-10).

Si la traza se envía como no anómala, el programa imprime en el terminal lo siguiente: "Traza número X (MMSI xxxxxxxxx) actualizada como NO anómala". Si, por el contrario, se evalúa como anómala, mostrará el mensaje: "Traza número X (MMSI xxxxxxxxx) actualizada como ANÓMALA".

```
def checkreg(mmsi,speed,course,status,latvalue,latdeg,latmin,latnom,lonvalue,londeg,lonmin,lonnom,datetime,timestamp):
        #Comprobamos si esta registrada
        cur=con.cursor()
        cur.execute("select count(*) from mmsi where mmsi=%s and latdeg=%s and"
                    " format(latmin,4)=format(%s,4) and latnom=%s and londeg=%s"
                    " and format(lonmin,4)=format(%s,4) and lonnom=%s",
                    (mmsi,latdeg,latmin,latnom,londeg,lonmin,lonnom))

        registered=cur.fetchone()
        registered='%(reg)s' %\
                {"reg":registered[0]}

        #Si no ha sido registrada, la insertamos en la base de datos,
        #construimos la sentencia TLL y la enviamos a OpenCPN
        if registered == '0':
            #llamada a la funcion de registro en la base de datos
            regtrack(trackN,latvalue,latdeg,latmin,latnom,lonvalue,londeg,
                     lonmin,lonnom,mmsi,speed,course,status,datetime,timestamp)
            #llamada a la funcion de deteccion de anomalias
            anomalyscore=anomalydetection(mmsi)
            #llamada a la funcion de construccion del mensaje
            message,color=writemsg(trackN,latdeg,latmin,latnom,londeg,
                                   lonmin,lonnom,mmsi,datetime,anomalyscore)
            #llamada a la funcion de envio del contacto al OpenCPN
            send(message)
            if color=='Q':
                print 'Traza numero %s, (MMSI %s) actualizada como ANOMALA' % (trackN,mmsi)
            elif color=='T':
                print 'Traza numero %s, (MMSI %s) actualizada como NO anomala' % (trackN,mmsi)
```

Figura 3-10 Función de comprobación de registro

3.3.6 Almacenamiento en base de datos

El almacenamiento en las bases de datos se realiza llamando a la función *regtrack*, que emplea la librería MySQLdb, apuntando a la base de datos con la función *cursor()*, que se conecta al servidor con la función *connect*, y ejecutando las órdenes con la función *execute* (Figura 3-11).

```
def regtrack(trackN,latvalue,latdeg,latmin,latnom,lonvalue,londeg,lonmin,lonnom,mmsi,speed,course,status,datetime,timestamp):
        with con:
                cur=con.cursor()
                cur.execute("insert into mmsi(track,latdeg,latmin,latnom,londeg,lonmin,lonnom,mmsi,datetime) "
                            "values(%s,%s,%s,%s,%s,%s,%s,%s,%s)",
                            (trackN,latdeg,latmin,latnom,londeg,lonmin,lonnom,mmsi,datetime))
                cur.execute('insert into AIS_reg(mmsi,lat,lon,speed,course,status,timestamp) '
                            'values(%s,%s,%s,%s,%s,%s,%s)', (mmsi,latvalue,lonvalue,speed,course,status,timestamp))
```

Figura 3-11 Función de registro de trazas en la base de datos

3.3.7 Detección de anomalías

En este apartado, se ha tomado la precaución de no evaluar el grado de anomalía de la traza hasta que se tengan varias posiciones que definan una ruta, en concreto, se empieza a evaluar el cuarto registro, ya que en el grado de anomalía calculado por la aplicación se tienen en cuenta las posiciones anteriores. Para esto, se llama a la función *anomalydetection* (Figura 3-12), que, a su vez, llama a la función *writeinput* (Figura 3-13), que escribe un archivo *csv*, llamado *trackroute.csv*, con el formato adecuado para la aplicación de detección de anomalías. A continuación, se inicializa la variable *anomalyscore* con valor *None* y se cuenta el número de líneas que tiene el archivo *trackroute*.csv. Si tiene más de 3, se arranca el módulo *maritimeanomalies* con este archivo como entrada. Si, por el contrario, se han registrado menos de 4 trazas, se devuelve la variable *anomalyscore* con valor *None*.

```
def anomalydetection(mmsi):
        #llamada a la funcion de escritura del inputPath
        writeinput(mmsi)
        anomalyscore=None
        if len(open("trackroute.csv").readlines()) > 3:
                input_path=os.path.abspath("trackroute.csv")
                maritimeanomalies(input_path,
                                  options.outputDir,
                                  options.useTimeEncoders,
                                  options.scale,
                                  not options.manualSequence,
                                  options.initial)

        anomaly_path=os.path.join(os.path.abspath("output"),"anomaly_scores.csv")
        with open(anomaly_path,"rb") as anomaly:
                reader=csv.DictReader(anomaly, delimiter=',')
                for row in reader:
                        anomalyscore=(row['anomaly_score'])

        return anomalyscore
```

Figura 3-12 Función de detección de anomalías

La aplicación de detección de anomalías escribirá un nuevo archivo *csv*, *anomaly_scores.csv*, que será leído por medio de un bucle *for*, asignando el valor del grado de anomalía del último registro a la variable *anomalyscore*.

```
def writeinput(mmsi):
        with con:
                cur=con.cursor()
                cur.execute("select * from AIS_reg where mmsi=%s" % mmsi)
                selection=cur.fetchall()

        with open('trackroute.csv',"w") as ruta:
                writer=csv.writer(ruta)
                i=0
                for row in selection:
                        line=[]
                        for item in row:
                                line.append(str(item))
                        writer.writerow(line)
                        i+=1
```

Figura 3-13 Función de escritura del archivo *trackroute.csv*

3.3.8 Construcción del mensaje y envío de datos a OpenCPN

A continuación, se llama a la función *writemsg*, que lleva a cabo tres acciones. La primera es elegir el color con que se va a representar la traza en función de su grado de anomalía, que será verde (T) si es menor que 0.5, y amarillo (Q) si es mayor. En segundo lugar, restará 1 al número de traza, para poder aprovechar los 100 disponibles, que van de 00 a 99 (Figura 3-14).

```
def writemsg(trackN,latdeg,latmin,latnom,londeg,lonmin,lonnom,mmsi,datetime,anomalyscore):
        #Elegimos el color de la traza segun su grado de anomalia
        if anomalyscore==None:
                color='T'
        else:
                anomalyscore=float(anomalyscore)
                if anomalyscore > 0.5:
                        color='Q'
                else:
                        color='T'

        #Escribimos la sentencia TLL
        track=trackN-1
        sentence= 'RATLL,%(track)02d,%(latD)02d%(latm)s,%(latN)c,%(lonD)03d%(lonm)s,%(lonN)c,%(mmsi)s,%(time)s,%(color)c,' %\
                {"track":track, "latD":latdeg, "latm":latmin, "latN":latnom, "lonD":londeg, "lonm":lonmin, "lonN":lonnom,
"mmsi":mmsi, "time":datetime, "color":color}

        #Calculamos el checksum NMEA y construimos el mensaje definitivo a enviar
        csum = checksum(sentence)

        message = '$%(sent)s*%(csum)02X' %\
                {"sent":sentence,"csum":csum}
        return message,color
```

Figura 3-14 Función de escritura del mensaje TLL

Por último, calculará el *checksum* llamando a la función del mismo nombre (Figura 3-15), después de haber construido la sentencia encadenando los caracteres correspondientes. Una vez calculado, escribirá la sentencia completa, añadiendo el símbolo "$" al principio y el "*" seguido del *checksum* al final.

```
def checksum(sentence):

        csum=reduce(operator.xor, (ord(s) for s in sentence), 0)
        return csum
```

Figura 3-15 Función de cálculo del *checksum*

Esta función devolverá la variable de tipo *str* llamada *message*, que será enviado a OpenCPN mediante la función *send* (Figura 3-16), dentro de la cual se definen los parámetros de la conexión creada con anterioridad en OpenCPN, el puerto UDP y la dirección IP, y se envía el mensaje, utilizando la función *socket.sendto* de la librería *socket*, al *software* de cartografía electrónica, que lo presenta en pantalla.

```
def send(message):
        #introducimos direccion IP y puerto UDP de destino
        UDP_IP='192.168.24.0'
        UDP_PORT=50001
        #Definimos el tipo de socket
        sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        #Enviamos el mensaje
        sock.sendto(message,(UDP_IP,UDP_PORT))
```

Figura 3-16 Función de envío del mensaje a OpenCPN como datagrama UDP

3.3.9 Tiempo real

El objetivo principal del proyecto, el de presentar los datos en tiempo real, se consigue introduciendo todo el programa en un bucle temporal. Para ello, después de vaciar las dos tablas para reiniciar el registro de dato, se declara una variable *time* con valor 0 y se abre un bucle *while* que se repetirá siempre que se cumpla que el valor de *time* sea 0. Cuando se completa el ciclo de lectura y procesado de cada llamada a la API de *marinetraffic*, se ejecuta la utilidad *sleep* de la librería *time*, que detiene la ejecución del programa durante 60 segundos (Figura 3-17).

```
con=mdb.connect('localhost','root','marianotfg','AIS_register')

cur=con.cursor()
cur.execute("delete from mmsi where track>0")
cur.execute("delete from AIS_reg where mmsi>0")

time=0
while time==0:
        #Llamada a la funcion de recopilacion de datos AIS
        AISreg()

        with open('AISdata.csv',"rb") as aisdata:
                reader=csv.DictReader(aisdata, delimiter=',')

                for row in reader:
                    #llamada a la funcion de lectura de datos del csv

mmsi,speed,course,status,latvalue,latdeg,latmin,latnom,lonvalue,londeg,lonmin,lonnom,datetime,timestamp,trackN = read()

                        #Comprobamos si la entrada de datos ya ha sido registrada y actuamos en consecuencia
                        checkreg
(mmsi,speed,course,status,latvalue,latdeg,latmin,latnom,lonvalue,londeg,lonmin,lonnom,datetime,timestamp)


        sleep(60)
```

Figura 3-17 Función principal con bucle temporal

4 PRUEBAS Y VALIDACIÓN DE LA APLICACIÓN

4.1 Verificación en el DST de Finisterre

4.1.1 Pruebas con la configuración normal del programa

En varias ocasiones, se probó el programa en condiciones normales, para evaluar su funcionamiento general. Debido a la escasez de anomalías en las rutas habituales de los buques que cruzan este dispositivo, el resultado más habitual era el de la no detección de ninguna anomalía, de forma que los contactos recibidos se mostraban en pantalla en color verde, es decir, como contacto no anómalo. En la Figura 4-1 se muestra la prueba realizada el día 5 de marzo de 2017, en torno a las 20:00 horas.

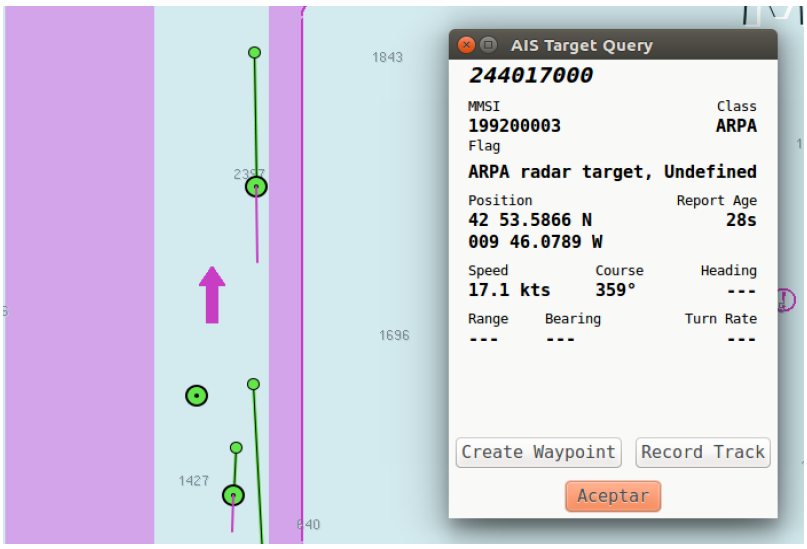

Figura 4-1 Prueba número 1 con cuadro de datos AIS en pantalla

Estas pruebas, a pesar de ello, nos sirven para comprobar la presentación de los datos en el cuadro de datos de traza. En la Figura 4-1 vemos el cuadro de datos de la traza que se encuentra en la parte superior de la imagen, tal y como los procesa OpenCPN. Es necesario remarcar que, en el campo MMSI no se sitúa el verdadero número MMSI de la traza. Este *software* cartográfico no posee un cuadro de presentación de trazas radar, por lo que las presenta como contacto AIS, asignándole como MMSI un número que es el resultado de sumar 199200000 a su número de traza. Esto nos permite, en este caso, identificar la traza como la número 4, teniendo en cuenta que la número 1 tendría el número de traza 00. Justo encima de este campo, se observa el nombre que el programa *RealTimeAnomalies.py* le asigna a la traza, que es el MMSI real del contacto.

En una ocasión, efectuando pruebas con la configuración normal, se dio el caso de un buque que estaba desarrollando una trayectoria anormal en la salida noreste del dispositivo. Esta anomalía fue detectada por la aplicación de detección de anomalías y procesada de forma correcta, en amarillo, por el programa desarrollado, como se observa en la Figura 4-2. Esta prueba se realizó el día 5 de marzo de 2017, en torno a las 23:00 horas.

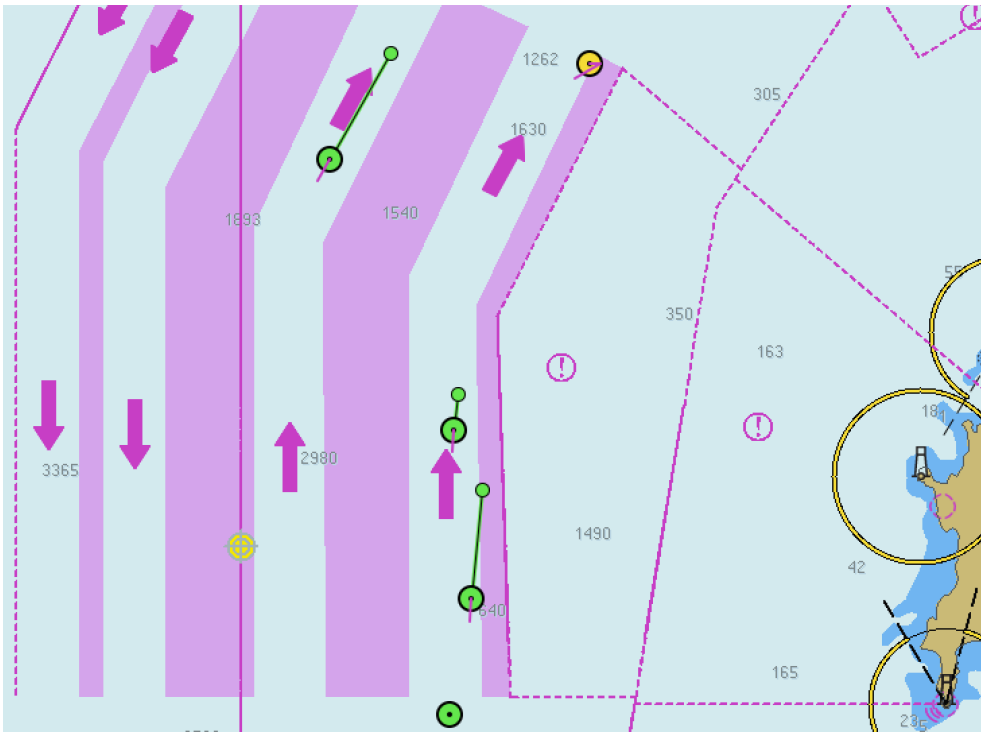

Figura 4-2 Anomalía real detectada en prueba número 2

En la Figura 4-3 se muestra más ampliada la misma traza anómala. Esta imagen nos permite también comprobar la presentación automática del COG (*Course Over Ground*) que realiza OpenCPN tras calcular la velocidad de la traza utilizando su posición anterior, así como la presentación en pantalla de la derrota seguida por el buque, la cual, como se puede observar, es ciertamente anómala.

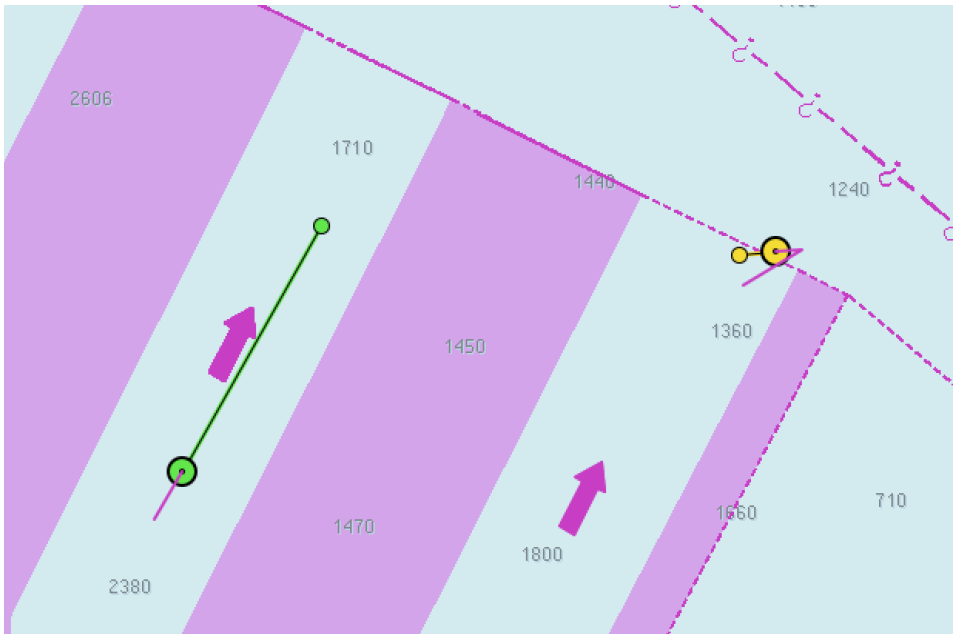

Figura 4-3 Anomalía real detectada en prueba número 2 (2)

4.1.2 Pruebas con reducción del margen aceptable de anomalía

Para verificar que el resultado obtenido en las pruebas con la configuración de parámetros normal del programa, se ha tomado la decisión de testearlo bajando el umbral de anomalía. De esta forma, trazas con un índice de anomalía muy bajo serán presentadas como anómalas, lo que servirá para confirmar el funcionamiento del programa.

En la primera prueba, realizada el día 6 de marzo de 2017 en torno a las 09:00 horas, se redujo el margen de 0.5 a 0.3. Los resultados, que se muestran en la Figura 4-4, no fueron muy diferentes a las pruebas anteriores, salvo por un contacto, cuyo rumbo era realmente anómalo, que se sitúa en la parte superior derecha de la imagen.

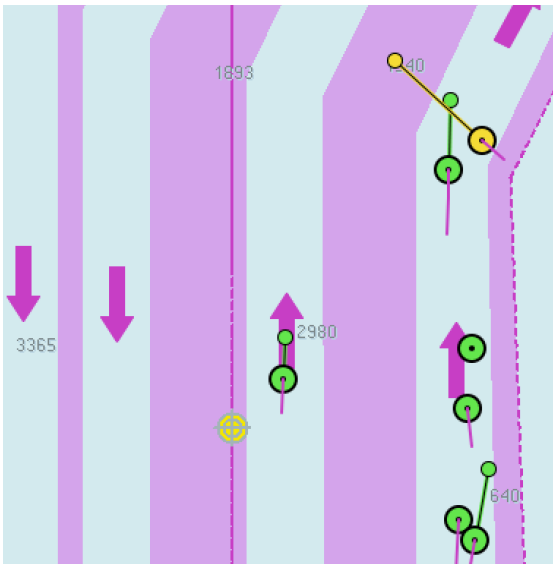

Figura 4-4 Prueba 1 con margen de anomalía 0.3

En la segunda prueba, realizada el día 6 de marzo de 2017 en torno a las 12:00, se redujo el margen de anomalía a 0.1, lo que generó un mayor porcentaje de trazas anómalas. Se observa en la Figura 4-5 que trazas siguiendo derrotas muy regulares obtienen resultado de anomalía, lo que cerciora que la reducción del margen de anomalía cumple con su función de probar la aplicación.

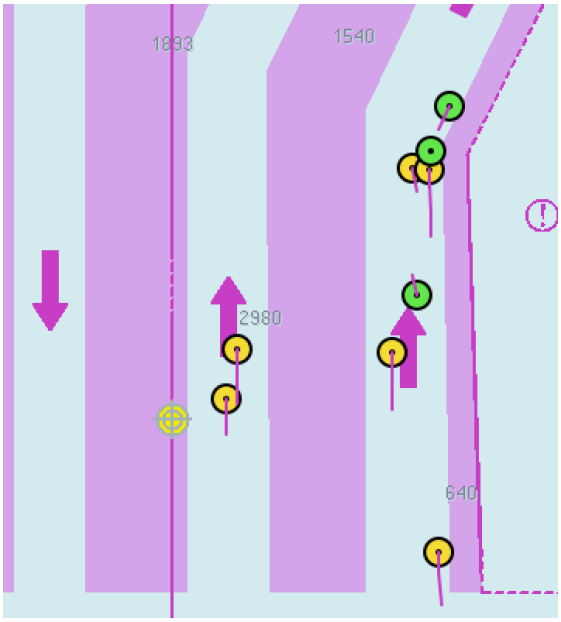

Figura 4-5 Prueba 2 con margen de anomalía 0.1

4.2 Análisis de los resultados

La valoración global de los resultados, teniendo en cuenta las pruebas realizadas, es satisfactoria.

En primer lugar, la llamada a la aplicación detectora de anomalías se efectúa con éxito, y esta procesa los datos de la ruta de la traza en cuestión, devolviendo al programa principal un valor de anomalía que este puede evaluar.

En segundo lugar, el programa es capaz de, después de haber evaluado el valor de anomalía de la traza, seleccionar el color de presentación de la misma.

Por último, se cumple con la función de realizar este análisis en tiempo real, lo que queda demostrado por la generación de las derrotas de las trazas y el cálculo de su velocidad por parte de OpenCPN. La actualización en tiempo real de las situaciones de cada una se consigue reiniciando el proceso continuamente y llamando de nuevo a la API de *marinetraffic*.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Revisión de los objetivos

Comprobando los objetivos fijados al inicio del trabajo, llegamos a la conclusión de que, desde un punto de vista general, se han alcanzado de forma satisfactoria. Se cumple con el objetivo principal de realizar el análisis de datos en tiempo real y con el de presentar las trazas en pantalla diferenciándolos por colores según su grado de anomalía.

Sin embargo, dentro de estos objetivos, esencialmente cumplidos, encontramos ciertos aspectos a mejorar, que se muestran a continuación:

- Tiempo real:

Aunque el bucle temporal permite que este objetivo se cumpla con un funcionamiento normal, el hecho de detener la aplicación 60 segundos después de cada procesado presenta un pequeño inconveniente que se debe a la ejecución de la aplicación detectora de anomalías. Se trata de la poca escalabilidad del proceso. Cuanto más larga es la ruta procesada por la aplicación, más tarda esta en realizar el análisis de anomalías, por lo que se alarga cada vez más el proceso. Aunque se trate de tiempos muy cortos, del orden de décimas de segundo, a medida que avance la aplicación, irá perdiendo la capacidad de presentar los datos en tiempo real.

Para solucionar este problema, se podrían incluir, por ejemplo, solamente las últimas 5 posiciones del blanco a la hora de crear el archivo que servirá de entrada para el detector de anomalías, lo que evitaría esta acumulación de retardo, recuperando la capacidad absoluta del desarrollo en tiempo real. Sin embargo, esta modificación requeriría un estudio para decidir el número de posiciones que optimizaría el resultado de la aplicación detectora de anomalías, el cual no se ha realizado por no disponer de tiempo suficiente.

- Diferenciación por colores:

En este caso, el objetivo se ha cumplido, ya que se distinguen visualmente las trazas anómalas de las que no lo son. Sin embargo, debido a la poca capacidad de decisión sobre la presentación que nos da OpenCPN, que solo nos da la posibilidad de elegir entre amarillo y verde, no se ha conseguido establecer una escala de colores que permita diferenciar, entre trazas anómalas, cual presenta un grado mayor de anomalía.

5.2 Líneas futuras

Como continuación del trabajo en el que se basa este proyecto [3], se ha conseguido avanzar en la integración de la IA en sistemas de gestión de datos marítimos. Sin embargo, a pesar de los buenos

resultados obtenidos, es evidente que existe mucho margen de mejora, tanto en el desarrollo de este programa, como en la integración de la IA en el ámbito marítimo.

Para optimizar la presentación, se recomienda estudiar la posibilidad de desarrollar un *plugin* para el programa de cartografía electrónica OpenCPN que implemente esta aplicación y que permita introducir una mayor variedad de colores en la presentación, para diferenciar con mayor eficacia las trazas anómalas y cumplir de forma aún mejor con este objetivo marcado al inicio del trabajo.

Dentro de este trabajo con los *plugins* de OpenCPN, es necesario recomendar la implementación de la capacidad de introducir datos cinemáticos y de identificación de los buques en el programa de alguna forma más ortodoxa, para que fueran presentados como tal. Si se estudiase esta posibilidad, sería interesante valorar, quizá más adelante, la posibilidad de implementar también en el *plugin* una manera de diferenciar los buques por su tipo, presentando de manera distinta, por ejemplo, un carguero, un ferry y un velero.

Como puede observarse, existe una gran margen de actuación en torno a OpenCPN con el objetivo de optimizar las funciones del programa desarrollado en este trabajo.

Además, para poder extender el uso del programa a varias zonas, podría estudiarse la posibilidad de implementar el almacenamiento de modelos CLA de diferentes zonas, de forma que, al llamar a la aplicación detectora de anomalías, se indique la zona geográfica en la que se está realizando el estudio y la aplicación seleccione, en función de esta zona, el modelo que va a utilizar.

6 BIBLIOGRAFÍA

En esta sección figurarán todas las referencias que han sido empleadas para el desarrollo de este trabajo.

[1] Purestone, «International Chamber of Shipping. Representing the Global Shipping Industry,» 2017. [En línea]. Available: http://www.ics-shipping.org/shipping-facts/shipping-facts.

[2] OMI, «Convenio SOLAS,» OMI, [En línea]. Available: http://www.imo.org/es/about/conventions/listofconventions/paginas/international-convention-for-the-safety-of-life-at-sea-(solas),-1974.aspx. [Último acceso: Febrero 2017].

[3] C. Arenas Pérez-Seoane, *Desarrollo de un sistema de inteligencia artificial para la supervisión y detección de anomalías en rutas marítimas,* Marín, Pontevedra, 2016.

[4] J. McCarthy, What is artificial Intelligence?, Online, 12 Noviembre 2007.

[5] P. H. Winston, Artificial Intelligence, Massachusetts, 1993.

[6] P. C. Jackson, Introduction to Artificial Intelligence, New York: Dover Publications, Inc., 1974.

[7] S. J. Russel y P. Norvig, Artificial Intelligence: A modern approach, New Jersey: Prentice-Hall, Inc., 1995.

[8] J. McCarthy, «Programs with common sense,» *Computation & intelligence,* pp. 479-492, 1995.

[9] Numenta, «Numenta,» Numenta, [En línea]. Available: http://numenta.com. [Último acceso: Enero 2017].

[10] J. Hawkins y S. Blakeslee, On intelligence, Times Books, 2004.

[11] J. Hawkins, «Hierarchical Temporal Memory (HTM) Whitepaper,» 2011. [En línea]. Available: http://numenta.com.s3-website-us-west-2.amazonaws.com/papers-videos-and-more/resources/hierarchical-temporal-memory-white-paper/. [Último acceso: Enero 2017].

[12] F. Byrne, «CLA for ML AI Researchers,» Numenta, Abril 2014. [En línea]. Available: https://github.com/numenta/nupic/wiki/CLA-for-ML-AI-Researchers. [Último acceso: Enero 2017].

[13] S. Ahmad, «Anomaly Detection and Anomaly Scores,» Numenta, Marzo 2014. [En línea]. Available: https://github.com/numenta/nupic/wiki/Anomaly-Detection-and-Anomaly-Scores. [Último acceso: Enero 2017].

[14] Geospatial Tracking . [En línea]. Available: http://numenta.com/assets/pdf/whitepapers/Geospatial%20Tracking%20White%20Paper.pdf. [Último acceso: Febrero 2017].

[15] Devpost, «Numenta HTM Challenge,» [En línea]. Available: https://htmchallenge.devpost.com/submissions. [Último acceso: Febrero 2017].

[16] M. Traffic, «Marine Traffic Blog,» [En línea]. Available: https://www.marinetraffic.com/blog/power-of-the-ais-network/. [Último acceso: Febrero 2017].

[17] M. Traffic, «API for AIS Data,» [En línea]. Available: https://www.marinetraffic.com/en/ais-api-services. [Último acceso: Febrero 2017].

[18] Organización Hidrográfica Internacional, Las cartas electrónicas de navegación y las prescripciones de transporte: Hechos, 2010.

[19] C. Sgto 1º Hidrógrafo Cid Álvarez, La Carta Electrónica, Cádiz: Escuela de Hidrografía, IHM, 2005.

[20] OpenCPN, «OpenCPN,» [En línea]. Available: https://opencpn.org/index.html. [Último acceso: Enero 2017].

[21] OpenCPN, «Developer Manual,» [En línea]. Available: https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual. [Último acceso: Febrero 2017].

[22] OpenCPN, «Resources,» [En línea]. Available: https://opencpn.org/OpenCPN/info/resources.html. [Último acceso: Febrero 2017].

[23] J. Gray, «Evolution of Data Management,» *Microsoft Research,* 1996.

[24] Numenta, «Installing and Building NuPIC,» [En línea]. Available: https://github.com/numenta/nupic/wiki/Installing-and-Building-NuPIC. [Último acceso: Enero 2017].

[25] Numenta, «Installing NuPIC on Ubuntu 15.10,» [En línea]. Available: https://www.youtube.com/watch?v=rN-57iBvcT4. [Último acceso: Enero 2017].

[26] Numenta, «NuPIC Github,» [En línea]. Available: https://github.com/numenta/nupic. [Último acceso: Enero 2017].

[27] P. S. Foundation, «Python Developer's Guide,» [En línea]. Available: https://docs.python.org/devguide/setup.html#build-dependencies.

[28] OpenCPN, «Download OpenCPN,» Diciembre 2016. [En línea]. Available: https://opencpn.org/OpenCPN/info/downloadopencpn.html.

[29] OpenCPN, «User Manual,» Diciembre 2016. [En línea]. Available: https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:opencpn_user_manual:getting_started:opencpn_installation:ubuntu_ppa.

[30] K. Betke, The NMEA 0183 Protocol, 2000.

ANEXO I: DICCIONARIO DE SIGLAS Y ACRÓNIMOS

Este anexo servirá de registro de las siglas y acrónimos utilizados a lo largo del trabajo. El significado de las siglas aparece preferentemente en español, con su traducción al inglés entre paréntesis. En caso de que no exista una traducción oficial al castellano, se pondrá únicamente el significado en la lengua inglesa.

AI: Inteligencia Artificial (*Artificial Intelligence*)

AIS: Sistema de Identificación Automática (*Automatic Identification System*)

API: Interfaz de Programación de Aplicaciones (*Application Programming Interface*)

ARPA: *Automatic Radar Plotting Aid*

ASCII: *American Standard Code for Information Interchange*

CLA: *Cortical Learning Algorithm*

COG: *Course Over Ground*

CPA: *Closest Point of Approach*

DST: Dispositivo de Separación de Tráfico

ECS: Sistema de Cartografía Electrónica (*Electronic Chart System*)

ECDIS: Sistema de Información y Presentación de Cartografía Electrónica (*Electronic Chart Display and Information System*)

ETA: *Estimated Time of Arrival*

GPS: Sistema de Posicionamiento Global (*Global Positioning System*)

HTM: Memoria Jerárquica Temporal (*Hierarchical Temporal Memory*)

IMO: *International Maritime Organization*

ITU: *International Telecommunication Union*

MSA: *Maritime Safety Awareness*

MSO: *Maritime Security Operations*

NMEA: *National Marine Electronics Association*

NuPIC: *Numenta Platform for Intelligent Computing*

OHI: Organización Hidrográfica Internacional

OpenCPN: *Open Chart Plotter Navigation*

OMI: Organización Marítima Internacional

OTAN: Organización del Tratado del Atlántico Norte

RCDS: Sistema de Presentación de Cartografía *Raster* (*Raster Chart Display System*)

RIPAM: Reglamento Internacional para la Prevención de Abordajes en la Mar.

SDR: Representaciones Dispersas Distribuidas (*Sparse Distributed Representations*)

SOLAS: Convenio Internacional para la Seguridad de la Vida Humana en la Mar (*International Convention for the Safety Of Life At Sea*)

SQL:	*Structured Query Language*
TCP:	*Transmission Control Protocol*
TCPA:	*Time for Closest Point of Approach*
TFG:	Trabajo de Fin de Grado
UDP:	*User Datagram Protocol*
WP:	*Waypoint*

ANEXO II: CORRESPONDENCIA DE OPCIONES EN EL PROGRAMA MARITIMEANOMALIES.PY

En este anexo se incluyen las opciones disponibles en la llamada del programa *maritimeanomalies.py*, que se introducen al hacer la llamada al programa principal, *RealTimeAnomalies.py*, tras identificarlo como tal.

- *Help* (`-h, --help`):

Muestra un mensaje de ayuda y cierra el programa.

- *Manual Sequence* (`-m, --manual-sequence`):

Divide las rutas por nombre, en lugar de hacerlo por saltos temporales.

- *Time encoders* (`-t, --time-encoders`):

Añade el codificador "momento del día" a los *model params*.

- *Verbose* (`-v, --verbose`):

Muestra las líneas de *debbuging*.

- *Output-dir* (`-o outputdir, --output-dir=outputdir`):

Selecciona el directorio de salida.

- *Scale* (`-s scale, --scale=scale`):

Si no se introduce, se toman 5 metros por defecto.

- *Initial* (`-i, --initial`):

Indica si hay que iniciar un nuevo modelo.

ANEXO III: EJEMPLO DE ARCHICO AISDATA.CSV CON DATOS DE LA API DE *MARINETRAFFIC*

MMSI, LAT, LON, SPEED, COURSE, STATUS, TIMESTAMP
304502000,42.886560,-9.953132,140,352,0,2017-03-06T22:29:30
235058838,43.275040,-9.943318,129,21,0,2017-03-06T22:30:18
538002845,43.265500,-9.915667,114,27,0,2017-03-06T22:30:01
355463000,43.047500,-9.804667,96,4,0,2017-03-06T22:29:54
304291000,43.026080,-9.791913,84,357,0,2017-03-06T22:30:39
305880000,43.023870,-9.767715,131,1,0,2017-03-06T22:30:27
235052314,43.271620,-9.761867,112,25,0,2017-03-06T22:29:43
224499000,43.244720,-9.742608,168,23,0,2017-03-06T22:30:31
244366000,43.238220,-9.732656,90,29,0,2017-03-06T22:28:22
224080680,43.345170,-9.701000,96,26,0,2017-03-06T22:30:29
244615000,43.343000,-9.656500,101,29,0,2017-03-06T22:30:39
305836000,43.402150,-9.616703,100,27,0,2017-03-06T22:30:24

ANEXO IV: EJEMPLO DEL ARCHIVO TRACKROUTE.PY CON LOS DATOS DE LA RUTA DE UNA TRAZA

MMSI,LAT,LON,SPEED,COURSE,STATUS,TIMESTAMP
244615000,43.120670,-9.768333,94,0,0,2017-03-06T21:07:55
244615000,43.136830,-9.768500,93,358,0,2017-03-06T21:10:54
244615000,43.158830,-9.768833,93,359,0,2017-03-06T21:15:59
244615000,43.152330,-9.768666,95,359,0,2017-03-06T21:16:58
244615000,43.163830,-9.769000,91,359,0,2017-03-06T21:17:58
244615000,43.167670,-9.769000,87,6,0,2017-03-06T21:22:57
244615000,43.182830,-9.764167,93,15,0,2017-03-06T21:28:57
244615000,43.211170,-9.749000,96,27,0,2017-03-06T21:36:59
244615000,43.219330,-9.743500,97,26,0,2017-03-06T21:40:18
244615000,43.212330,-9.748167,95,29,0,2017-03-06T21:43:59
244615000,43.227000,-9.738500,97,27,0,2017-03-06T21:47:01
244615000,43.240330,-9.729834,101,24,0,2017-03-06T21:48:48
244615000,43.247500,-9.725333,102,25,0,2017-03-06T21:51:39
244615000,43.269330,-9.711000,94,25,0,2017-03-06T22:00:29
244615000,43.271670,-9.709333,95,28,0,2017-03-06T22:07:59
244615000,43.286170,-9.698667,102,27,0,2017-03-06T22:10:59
244615000,43.306330,-9.683833,99,29,0,2017-03-06T22:15:48
244615000,43.309170,-9.681833,101,28,0,2017-03-06T22:16:58
244615000,43.316000,-9.676833,105,29,0,2017-03-06T22:19:39
244615000,43.329500,-9.666667,103,29,0,2017-03-06T22:25:09
244615000,43.333330,-9.663834,100,32,0,2017-03-06T22:26:39
244615000,43.330830,-9.665667,103,30,0,2017-03-06T22:28:58
244615000,43.343000,-9.656500,101,29,0,2017-03-06T22:30:39