IMPLANTACIÓN DE UN SENSOR SONAR PARA EL CONTROL DE LA NAVEGACIÓN DE UN VEHÍCULO SUBMARINO

Emilio García Fidalgo

Proyecto Final de Carrera

Ingeniería en Informática

Directores:

Alberto Ortiz Rodríguez

Javier Antich Tobaruela

Universitat de les Illes Balears (UIB)

2007

Agradecimientos

Antes de entrar en materia, me gustaría dedicar estas primeras líneas del presente documento a todas aquellas personas que han contribuido a la realización de este proyecto, ya sea por su labor, por su apoyo o por ambas cosas.

En primer lugar quiero dar las gracias a mis dos directores de proyecto, Alberto Ortiz y Javier Antich, por ofrecerme la oportunidad de desarrollar este trabajo y por la gran cantidad de horas que han dedicado a resolver mis continuas dudas.

A mi familia y amigos, en especial a mis padres, ya que gracias a su apoyo (sobretodo en los malos momentos) he conseguido llegar hasta aquí y me he convertido en la persona que soy.

A aquellos compañeros de facultad que formaron parte de mis años de estudiante, junto a los que tantas horas pasé y que tanto hablamos de este momento.

A todos ellos, y a aquellas personas de las que me pueda olvidar y que deberían estar aquí, mil gracias.

Índice

1. Introducción **1**

1.1. Robótica submarina 1
1.2. Vehículos submarinos 3
1.3. Aplicaciones 5
1.4. Interacción con el entorno 6
1.5. Objetivos del proyecto 7
1.6. Estructura del documento 7

2. Sónar Tritech Miniking **9**

2.1. Introducción 9
2.2. Propagación del sonido 10
2.3. Tipos de sónar 13
2.4. Características técnicas 13
2.5. Funcionamiento 16
2.6. Interpretación de datos 18
2.7. Protocolo SeaNet 24
2.7.1. Descripción general 25
2.7.2. Clasificación de tramas 28
2.7.3. Formato general de las tramas 28
2.7.4. Descripción de las tramas 30

3. Diseño e implementación **33**

3.1. Arquitectura de capas 33
3.2. Capa 1: comunicación serie 35
3.3. Capa 2: comandos y mensajes 37
3.3.1. Clases abstractas 38
3.3.2. Mensajes 39
3.3.3. Comandos 40
3.4. Capa 3: MiniKing 41
3.4.1. Funciones de acceso público a la librería 42
3.5. Ejemplo de utilización 46

4. Ejemplo de utilización **49**
4.1. Características generales . . . 49
4.1.1. Funcionamiento general . . . 50
4.1.2. Salir de la aplicación . . . 54
4.1.3. Configuraciones . . . 54
4.1.4. Valor límite . . . 55
4.1.5. Reinicio del sónar . . . 55
4.1.6. Mapas de color . . . 57
4.1.7. Capturas de pantalla . . . 57
4.1.8. Grabación de sesiones . . . 57
4.1.9. Análisis de *scanlines* . . . 57
4.1.10. Ayuda . . . 59
4.1.11. Exploración de sectores . . . 59
4.2. Diseño e implementación . . . 59
4.2.1. Colores . . . 62
4.2.2. Análisis de *scanlines* . . . 63
4.2.3. Configuraciones . . . 64
4.2.4. Grabación de sesiones . . . 66
4.3. Resultados . . . 67

5. Implantación en un simulador **71**
5.1. Control de robots móviles . . . 71
5.2. Paradigmas . . . 72
5.2.1. Control deliberativo . . . 73
5.2.2. Control reactivo . . . 75
5.2.3. Control híbrido . . . 78
5.3. NEMO$_{CAT}$ . . . 79
5.3.1. El entorno submarino . . . 81
5.3.2. Vehículos autónomos submarinos . . . 81
5.3.3. Comportamientos disponibles . . . 82
5.4. Adaptación . . . 84
5.5. Resultados . . . 89
5.5.1. Primer experimento . . . 89
5.5.2. Segundo experimento . . . 89

6. Conclusiones **95**
6.1. Consecución de los objetivos . . . 95
6.2. Experiencias del proyecto . . . 96
6.3. Trabajo futuro . . . 98

Bibliografía **99**

A. Tramas del protocolo *SeaNet* **101**
A.1. Comandos . . . 101
A.1.1. MtSendVersion . . . 101
A.1.2. MtSendBBUser . . . 101
A.1.3. MtReboot . . . 101
A.1.4. MtSendData . . . 102
A.1.5. MtHeadCommand . . . 102
A.2. Mensajes . . . 103
A.2.1. MtVersionData . . . 103
A.2.2. MtAlive . . . 103
A.2.3. MtHeadData . . . 104
A.2.4. MtBBUserData . . . 105

B. Tecnología y herramientas utilizadas **109**

C. Librería **113**
C.1. Acceso al puerto serie . . . 113
C.2. Protocolo *SeaNet* . . . 120
C.2.1. Comandos . . . 124
C.2.2. Mensajes . . . 142
C.3. *Miniking* . . . 160

D. Código fuente del *Miniking Viewer* **175**

Índice de figuras

1.1. ROV para el control de instalaciones submarinas 3
1.2. AUV Phoenix 4

2.1. Posibles caminos del sonido en un océano 12
2.2. Sonar Tritech Miniking 14
2.3. Control de rango dinámico 20
2.4. Escenario de ejemplo 22
2.5. Barrido del entorno del sónar 23
2.6. Scanlines 23
2.7. Intensidad de eco de los *bins* del pulso 24
2.8. Protocolo SeaNet 26

3.1. Arquitectura *software* de la librería 36
3.2. Jerarquía de clases 39
3.3. Configuración angular del sónar 43

4.1. Selector de puerto serie 51
4.2. Ventana de carga 51
4.3. Ventana principal del programa 52
4.4. Barra de herramientas y menús 54
4.5. Filtro de valor límite 55
4.6. Diferentes mapas de color de la aplicación 56
4.7. Representación gráfica de los valores de una *scanline* 58
4.8. Exploración de un sector 60
4.9. Principales clases de la aplicación 62
4.10. Estructuras dinámicas de datos para almacenar *scanlines* 64
4.11. Primera imagen del barrido del depósito de agua 68
4.12. Segunda imagen del barrido del depósito de agua 69
4.13. Tercera imagen del barrido del depósito de agua 69
4.14. Cuarta imagen del barrido del depósito de agua 70
4.15. Quinta imagen del barrido del depósito de agua 70

5.1. Esquema del modelo de control deliberativo 74
5.2. Esquema del modelo de control reactivo 75

5.3. Generación de un vector de salida a través de campos de potencial 77
5.4. Esquema del modelo de control híbrido 78
5.5. Vista general del simulador 80
5.6. Matriz del entorno submarino 81
5.7. Sensores disponibles en la aplicación 86
5.8. Valor de los *bins* de la trama *MtHeadData* 87
5.9. Representación gráfica del barrido del sónar Miniking en el simulador $NEMO_{CAT}$ 88
5.10. Entorno virtual utilizado en el primer experimento 90
5.11. Evitación de un obstáculo dentro del entorno virtual en el primer experimento . . 91
5.12. Trayectoria seguida por el vehículo en el segundo experimento 92
5.13. Entorno virtual del segundo experimento 93
5.14. Trayectoria seguida por el vehículo en el segundo experimento 94

Índice de tablas

2.1. Características técnicas del sónar Miniking 15
2.2. Identificadores de trama 29

3.1. Configuración por defecto del dispositivo 44

Capítulo 1

Introducción

El objetivo de este primer capítulo es realizar una pequeña introducción a la robótica submarina. En él, se describen los tipos de vehículos submarinos utilizados en la industria, los sensores que pueden llevar incorporados y los campos de aplicación más comunes. Para terminar, se analizan los objetivos del proyecto y se comenta la estructura del presente documento.

1.1. Robótica submarina

Dentro de la rama de la tecnología, podríamos definir la **robótica** como la ciencia que estudia el diseño y la construcción de máquinas capaces de realizar tareas de forma automática. Su interés es indudable cuando se trata de tareas repetitivas y/o peligrosas para los seres humanos, aunque hoy en día podemos encontrar máquinas diseñadas para otras tareas menos relevantes, como es el caso de, por ejemplo, máquinas de compañía y de ocio de la firma japonesa *Sony*. Dichas máquinas se conocen con el nombre de **robots**. Más especificamente, la robótica se encarga del diseño, fabricación, control y programación de robots. Toma conceptos de diferentes materias, como pueden ser la mecánica, las matemáticas, la física, la informática, la psicología o la filosofía y es, por tanto, una asociación de ciencias.

El término robot fue acuñado por el escritor checoslovaco Karel Kapek en su libro *Rossum´s Universal Robots*. Procede del checo *Robota*, que significa esclavo o trabajador. Fue popularizado

por el escritor Isaac Asimov en su novela *I Robot* en 1950. Existen diferentes definiciones de la palabra **robot**, ya que ninguna de ellas es capaz de englobar a todos los tipos existentes y utilizados. Tratando de dar una definición actual, podríamos decir que un robot es una máquina programada para moverse, manipular objetos y realizar tareas a la vez que interactúa con el entorno en el que se encuentra.

Desde sus inicios hasta nuestros días, la robótica ha ido explorando diferentes campos, dando lugar a varias líneas de investigación. Este proyecto se engloba dentro del marco de una ellas: la **robótica submarina**.

Como su nombre indica, la robótica submarina es la rama de la robótica dedicada al estudio y diseño de máquinas para su utilización dentro de medios acuáticos. Su importancia radica en la dificultad (o imposibilidad) por parte del ser humano para realizar ciertas tareas en el agua, debido a problemas de aire, presión o distancia, entre otros.

Dos hechos curiosos y relevantes para la popularización de los robots submarinos ocurrieron en el verano de 1985. El primero fue el accidente de un vuelo de Air India en el océano Atlántico, cerca de las costas de Irlanda. Un vehículo submarino guiado remotamente, utilizado normalmente para el tendido de cable, fue utilizado para encontrar y recobrar la caja negra del avión. El segundo hecho fue el descubrimiento del *Titanic* en el fondo de un cañón, cuatro kilómetros bajo la superficie, donde había permanecido desde que en 1912 chocase con un iceberg. Un vehículo submarino fue utilizado para encontrar, explorar y filmar el hallazgo.

Uno de los principales usos de estos vehículos submarinos es la inspección y mantenimiento de instalaciones sumergidas tales como tuberías que conducen petróleo o gas o el cableado para telecomunicaciones y transporte de energía eléctrica. Con un carácter más científico, se utilizan en investigaciones geológicas y geofísicas del suelo marino. Más adelante se hará un mayor hincapié en las posibles aplicaciones.

1.2. Vehículos submarinos

Los vehículos utilizados en medios subacuáticos son conocidos como *URV*[1] o como *UUV*[2]. Los dos términos hacen referencia a máquinas submarinas no tripuladas, con el objetivo de minimizar la presencia de los humanos en un entorno tan poco adecuado como el agua. Básicamente, podemos dividir los vehículos submarinos en dos grandes grupos:

- **ROV:** *Remotely Operated Vehicle*
- ***AUV:*** *Autonomous Underwater Vehicle*

Los **vehículos remotamente operados** utilizan telerobótica y telepresencia durante el control y la navegación. Esto quiere decir que, en todo momento, hay un operador humano interactuando con la nave, analizando la información obtenida y enviando órdenes de funcionamiento. Está conectado al barco a través de un **cordón umbilical**, formado por un grupo de cables que llevan corriente eléctrica y señales de vídeo y datos entre el vehículo y el operador de a bordo. Son útiles cuando la posición del destino es incierta o cuando las condiciones del entorno submarino hacen peligrar una misión tripulada.

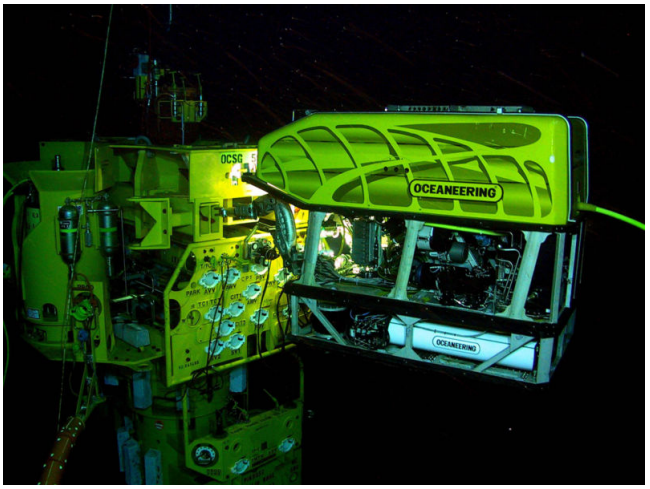

Figura 1.1: ROV para el control de instalaciones submarinas

[1] *Underwater Robotic Vehicle*
[2] *Unmanned Underwater Vehicle*

La principal desventaja que presentan los *ROV* es el cable umbilical, ya que restringe el alcance de las misiones a las cercanías del barco desde el que se está operando. Otra desventaja es el arrastre que produce sobre la nave, siendo necesarios, por tanto, motores de mayor capacidad. No obstante, los *ROV* que se utilizan cerca de la superficie suelen utilizar enlaces de radiofrecuencia.

En cuanto a los **vehículos autónomos submarinos**, son vehículos no tripulados y sin cable umbilical. Por tanto, transportan su propia fuente de energía y basan su comportamiento en el software que se está ejecutando en el ordenador de a bordo, que expresa la misión a ejecutar. El no depender del cable posibilita misiones a mayor distancia y de mayor profundidad, siendo a su vez inmune a las condiciones atmosféricas. Además, se reducen gastos, ya que no es necesario un operador para interactuar con la máquina.

Las mismas ventajas de un *AUV* son a su vez sus limitaciones. La duración de las misiones está sujeta a la duración del sistema de alimentación. Además, al no haber un operador humano, el vehículo depende íntegramente de cómo ha sido programado, no siendo trivial responder en tiempo real a posibles imprevistos, debido a limitaciones del *hardware* y/o del *software*.

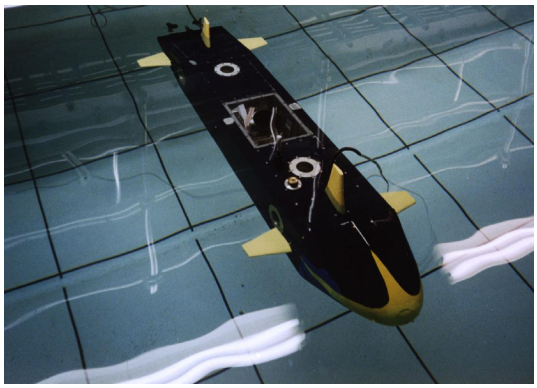

Figura 1.2: AUV Phoenix

1.3. Aplicaciones de la robótica submarina

El océano ocupa aproximadamente el 70 % de la superficie de la tierra. Posee una gran cantidad de recursos vivos y minerales, tiene grandes reservas energéticas, afecta al clima y en él tienen también lugar ciertos fenómenos de interés como los terremotos. Por lo tanto, es evidente que su exploración está más que justificada. Por todo ello, la robótica submarina tiene una gran cantidad de aplicaciones, entre las que destacan [1]:

- **Aplicaciones puramente científicas**. Son aquellas cuya finalidad es mejorar el conocimiento sobre los océanos, los procesos físicos y biológicos que ahí tienen lugar, la flora, la fauna, etc. Se puede incluir también en este grupo el estudio de la relación del clima con el estado de los océanos, así como la arqueología submarina.

- **Instalación y mantenimiento de estructuras submarinas**. Se refiere al montaje y mantenimiento posterior de tuberías y/o cables u otras instalaciones necesarias relacionadas con una actividad concreta. El mayor partido a estas aplicaciones lo obtienen las compañías petroleras.

- **Control de la flora y fauna submarina**. Otra de las grandes áreas de intervención es el control de la vida submarina. Mediante un AUV se puede obtener una medida más exacta del número de individuos de colonias de peces, algas, moluscos, etc.

- **Conservación del entorno**. Se refiere a la monitorización y polución oceánica. Podría llevarse a cabo a través de un AUV con los sensores químicos adecuados. De esta forma sería posible controlar el vertido de materiales contaminantes para actuar cuanto antes.

- **Asistencia a operaciones marinas**. Un AUV puede ser muy útil en la obtención de mapas del fondo oceánico de forma continuada. Asimismo, su utilización será muy valiosa también en naufragios, dentro de las tareas de salvamento, búsqueda y obtención de la caja negra, etc.

1.4. Interacción con el entorno: los sensores

En el apartado 1.2 hemos mencionado los dos grandes grupos de vehículos submarinos que existen: los *ROV* y los *AUV*. En el primer grupo, la trayectoria del aparato es guiada por un operador en un barco de soporte en tiempo real. Sin embargo, los *AUV* funcionan de forma autónoma, y por tanto, deben ser capaces de autoajustar su dirección y orientación. Es decir, deben **obtener información del entorno** para poder tomar decisiones por sí mismos. Esto se realiza a través de su **equipamiento sensorial**.

En un *AUV* se puede distinguir dos tipos diferentes de sensores:

- **Sensores de sistema**, para llevar cabo la navegación, evitación de obstáculos y auto-diagnóstico del vehículo.
- **Sensores de misión**, requeridos para obtener la información del entorno de operación y así poder llevar a cabo los objetivos de la misión.

Dentro de los **sensores de sistema**, los *sensores de navegación* son los que proporcionan al vehículo la capacidad de conocer su posición y orientación. Por otro lado, los *sensores de evitación de obstáculos* típicamente proporcionan información acerca de la distancia a la que se encuentran los objetos. Combinando ambos tipos de sensor el *AUV* es capaz de situarse en el espacio, evitando obstáculos. Dentro del grupo de sensores de sistema encontramos también los *sensores de auto-diagnóstico*, encargados de comprobar el correcto funcionamiento del vehículo. El último tipo que se puede englobar en esta categoría son los *sensores de comunicación*. A pesar de que un vehículo sea autónomo, en algún momento requerirá la interacción del usuario. Esta es la funcionalidad básica de estos últimos.

En cuanto a los **sensores de misión**, cada tarea determina los tipos de sensores a utilizar, por lo que hay tantos sensores como posibilidades diferentes para una misión. No obstante, la mayoría de las operaciones se llevan a cabo con un **sonar**. Este dispositivo es el principal objeto de estudio de este proyecto. Sin embargo, en nuestro caso se utilizará como *sensor de evitación*

de obstáculos y no como *sensor de misión*. Otros sensores utilizados en este apartado son los *sensores de campo magnético*, *sensores químicos* y *sensores biológicos*.

1.5. Objetivos del proyecto

Los objetivos de este proyecto son:

- Diseñar e implementar una librería de interfase con el sónar *Miniking* de *Tritech*. Esta librería permitirá a cualquier programador que necesite interactuar con el dispositivo desarrollar sus aplicaciones sin necesidad de conocer los detalles del protocolo sobre el que opera dicho sónar.

- Programar una aplicación que haga uso de la librería mencionada en el punto anterior. En concreto, se va a desarrollar una interfaz gráfica de usuario para visualizar los datos procedentes del sónar. Debe permitir configurar el dispositivo desde el mismo programa e incluir una serie de utilidades como capturas de pantalla, grabación y reproducción de sesiones, etc.

- Implantar la librería dentro de un entorno de simulación con el objetivo de mostrar utilidad en el control de la navegación de un vehículo submarino, concretamente en tareas de evitación de obstáculos.

1.6. Estructura del documento

- En el **capítulo 1**, nos hemos centrado en hacer una pequeña introducción a la robótica submarina y sus aplicaciones. Se han explicado los diferentes tipos de vehículos que se utilizan en ella y los principales sensores con los que éstos operan. Al final, se han comentado los objetivos básicos que se pretenden alcanzar con este proyecto.

- En el **capítulo 2** se presenta el sónar *Miniking* de *Tritech*. Se detallan sus características técnicas, así como el método que utiliza para detectar obstáculos. Se explica el protocolo a

través del cual se comunica con un computador, conocido como *SeaNet*. Además, incluye una introducción a los sistemas sónar en general y a la propagación del sonido en el agua.

- En el **capítulo 3** se muestra el diseño e implementación de la librería de interfase con el dispositivo. Se detalla la estructura jerárquica empleada, junto con los principales métodos para interactuar con el sónar. Concluye con un ejemplo de cómo utilizar esta librería.

- En el **capítulo 4** se presentan las características y la implementación de una aplicación que hace uso de la librería programada. Es una interfaz gráfica de usuario que explora el entorno en el que se halla el sónar y presenta los resultados de una forma visual. El programa permite además configurar el dispositivo en tiempo real.

- En el **capítulo 5** se muestra la implantación de la librería dentro de un entorno virtual, el simulador de arquitecturas de control para robots submarinos denominado $NEMO_{CAT}$. El objetivo fundamental de esta parte es simular el funcionamiento del sónar *Miniking* dentro del propio simulador, de forma que se pueda utilizar en tareas de evitación de obstáculos para el control de la navegación de un vehículo submarino. En este apartado se realiza además una pequeña introducción al control de un robot móvil.

- En el **capítulo 6** se analiza el trabajo realizado y se extraen conclusiones, experiencias personales y opiniones acerca del desarrollo de este proyecto.

- Finalmente se incluyen una serie de **apéndices**, en los que aparecen los formatos de trama del protocolo *SeaNet*, las tecnologías y herramientas utilizadas en alguna fase de este proyecto y el código fuente de la librería y la aplicación gráfica.

Capítulo 2

Sónar Tritech Miniking

En el capítulo anterior nos hemos situado dentro del campo de la robótica submarina. El proyecto está basado en experiencias con un sónar comercial concreto. En este capítulo se presenta éste sónar, su mecanismo de funcionamiento y se describe el protocolo de comunicación a través del cual se puede interactuar con él. Antes, se realiza una pequeña introducción a los sistemas sónar y a la propagación del sonido en el agua.

2.1. Introducción

El término sónar (*SOund Navigation And Ranging*) hace referencia al uso de ondas acústicas para determinar la distancia a los obstáculos circundantes más próximos. En este sentido, mantiene ciertas similitudes con el *radar*, el cual emplea ondas electromagnéticas para el mismo fin. Los sistemas sónar son ampliamente utilizados en entornos submarinos debido a las dificultades que plantea el agua para la propagación de la radiación electromagnética.

Generalmente están compuestos por un emisor y un receptor. El emisor transmite un haz de impulsos ultrasónicos. Cuando los impulsos interaccionan con un objeto, éstos se reflejan y forman una señal de eco que es captada por el receptor, que amplifica la energía de las ondas recibidas y genera una señal, que puede representarse en la pantalla de un computador o ser tratada a un nivel superior de proceso.

Los trabajos de Daniel Colloden en 1822, que utilizó una campana bajo el agua para calcular la velocidad del sonido en el medio, condujeron a la invención de los dispositivos sónar por parte de otros inventores. El primero fue diseñado por el arquitecto naval **Lewis Nixon** en 1906 para la detección de *icebergs*. Sin embargo, el aumento de interés por estos sistemas tuvo lugar durante la Primera Guerra Mundial, debido a la necesidad de detectar submarinos. El francés **Paul Langévin** creó en 1915 el primer modelo para submarinos, basándose en las propiedades piezoeléctricas del cuarzo. Este sónar ha servidor de base para la construcción de modelos posteriores.

En las siguientes secciones se describirán de forma global las características de la transmisión del sonido en el agua y se analizarán los diferentes tipos de sónar utilizados en el mercado.

2.2. Propagación del sonido en un medio elástico

El sonido está formado por ondas que se propagan a través de un medio que puede ser sólido, líquido o gaseoso. Dichas ondas son longitudinales, ya que las partículas materiales que las transmiten oscilan en la dirección de propagación del movimiento, en contraposición con las ondas transversales de otros fenómenos físicos en los que las partículas oscilan perpendicularmente a la dirección de propagación de la onda.

Una analogía al fenómeno ondulatorio de la propagación se aprecia al arrojar una piedra al agua. Se genera una serie de ondas que se propagan y se dispersan en todas las direcciones, con simetría esférica. Esto es debido a que las partículas del agua transmiten su movimiento a las contiguas sucesivamente en todo el volumen del agua. Análogamente, el sonido llega a nuestros oídos gracias a que las partículas que componen el medio en que estamos vibran y transmiten su oscilación a sus «vecinas» más próximas.

Por lo tanto, está claro que el sonido necesita de un medio transmisor para propagarse, siendo imposible en el vacío, ya que en este medio sólo se pueden propagar ondas que no requieran de

sustrato material, como las electromagnéticas. Cuando el movimiento de una partícula del medio se transmite a las partículas vecinas, aunque con cierto retraso en el comienzo del movimiento, se dice que se está propagando un sonido.

Las **ondas sonoras** son un tipo de onda elástica que tiene como principales características dos propiedades de la materia:

- **Elasticidad:** Cuando una partícula se desplaza de su posición de equilibrio por la acción de un estímulo externo, se ve sometida a una fuerza de recuperación, ya que es un medio elástico, que hace que tienda a volver a dicha posición.
- **Inercia:** Cuando un elemento desplazado vuelve a su posición original debido a la elasticidad del medio, la inercia hace que no se detenga y la cantidad de movimiento que tiene lo desplaza a una posición opuesta a la primera.

El océano es un medio irregular y heterogéneo limitado en su parte superior por la superfície y en su parte inferior por una frontera penetrable, el fondo marino [4]. En general, el sonido dentro del agua del mar sufre variaciones de velocidad debido, entre otros, a los cambios de temperatura y se atenúa con el cuadrado de la frecuencia, aunque mucho menos que la radiación electromagnética. Debido a la relación entre atenuación y frecuencia, los sistemas de vigilancia y monitorización acústica utilizan bajas frecuencias, para poder alcanzar así mayores distancias. Los dispositivos de alta frecuencia se utilizan en sistemas de corto alcance y alta resolución.

En la figura 2.1 se muestran los posibles caminos que puede tomar una onda acústica dentro del océano [4], partiendo de dos puntos diferentes, uno a 80 metros y el otro a 500 metros de profundidad, y desplazándose de izquierda a derecha.

En el caso A, la onda queda atrapada por la superficie debido al tipo de variaciones que sufre la velocidad del sonido cuando la profundidad es escasa. En el camino B, la onda sale de una fuente a gran profundidad con un ángulo de salida pequeño respecto a la horizontal. En este caso la onda se propaga sin interacción ninguna con la superficie ni con el fondo. A medida que diverge

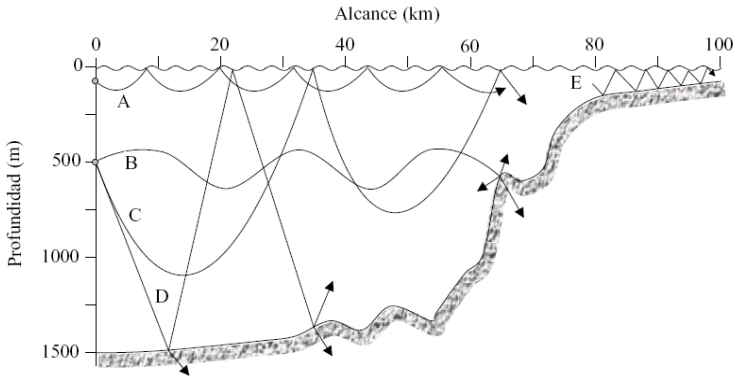

Figura 2.1: Posibles caminos del sonido en un océano

el ángulo de salida respecto a la horizontal, se produce la situación C, un fenómeno periódico en el que hay interacción con la superficie pero no con el fondo. Finalmente el caso D también es periódico, aunque con un ciclo más corto que el caso C y con interacción en el fondo. Por otro lado, el camino E, representado en la parte derecha del dibujo, se corresponde con aguas poco profundas, donde existe una gran variabilidad de la estructura del fondo combinado con una gran variabilidad de la velocidad del sonido, dando lugar a condiciones de propagación complicadas.

Como se ha podido observar, la propagación del sonido dentro del medio oceánico está sometida a perturbaciones considerables en velocidad y dirección. Son debidas fundamentalmente a los cambios de temperatura, aunque también influyen las olas, las mareas y las corrientes. Aparte, como ya se ha mencionado antes, las ondas sufren una atenuación en su recorrido. Los principales mecanismos que producen este efecto son:

- La *absorción volumétrica*, que se incrementa con la frecuencia y es el mecanismo de pérdida dominante en el camino B de la figura 2.1.

- La *reflexión con pérdida*, que da lugar a la pérdida de energía debido a la interacción del sonido con el fondo.

- Las *pérdidas por dispersión*, que son debidas a las irregularidades de la superficie y a la rugosidad del fondo. La reflexión genera múltiples ondas, con lo que la onda reflejada es la

superposición aleatoria de formas de ondas retrasadas en el tiempo. Esto da lugar a ruido multiplicativo denominado *speckle*, que es mayor cuanto mayor es la frecuencia.

2.3. Tipos de sónar

Existen dos grandes grupos de sónar de acuerdo con su forma de operar: los **activos** y los **pasivos**.

Los **activos** envían un pulso de sonido, conocido normalmente como *ping*, y escuchan posibles ecos de esta señal. Para largas distancias, estos tipos de sónar utilizan frecuencias bajas, mientras que para distancias cortas se pueden emplear frecuencias más altas, consiguiéndose una mayor resolución. Los receptores calculan la distancia al objeto a partir del tiempo que ha tardado el eco en volver desde que el pulso fue emitido. Cuando el transmisor y el receptor se encuentran en el mismo lugar se dice que el sónar opera en modo *monoestático*; cuando se hallan separados, entonces opera en modo *biestático*.

Los **pasivos** no emiten ningún pulso, sino que escuchan el entorno en el que se encuentran. Estos sistemas suelen tener una extensa base de datos para identificar barcos y acciones a partir de los datos obtenidos del medio. Los sistemas de sónar activos actuales tiene también capacidad de funcionar como sónar pasivo con ciertas limitaciones.

2.4. Características técnicas del sónar Miniking

El sónar objeto de este proyecto se denomina **Miniking** y es fabricado por la empresa escocesa **Tritech International Limited**. Destaca por su pequeño tamaño, lo cual permite insertarlo en vehículos *ROV* y *AUV* de tamaño reducido. Actúa fundamentalmente como sónar activo, enviando pulsos y esperando el eco producido por éstos dentro de un rango temporal determinado. El sónar *Miniking* permite realizar exploraciones de 360° ó restringidas a un sector angular configurable gracias a la incorporación de un motor al que está adosado el transductor dentro del cabezal. Se utiliza en sistemas genéricos de medida de distancia, para aplicaciones de evitación

de obstáculos y para reconocimiento de objetos.

Figura 2.2: Sonar Tritech Miniking

Para poder visualizar los resultados, es necesario un computador (portátil o de sobremesa) que tenga un puerto serie, ya que es a través de esta conexión como el sónar envía sus resultados y escucha posibles órdenes. El dispositivo incluye un puerto de estas características configurable para dos protocolos: *RS232* y *RS485*. Aunque ambos sirven para la transmisión serie, presentan ciertas diferencias. El *RS232* permite la comunicación con un dispositivo a menos de 45 metros. El *RS485*, sin embargo, permite la conexión entre 31 dispositivos (como máximo) siempre y cuando la longitud total no supere los 1200 metros de longitud. Comúnmente, el más utilizado e incluido en la mayoría de los ordenadores convencionales es el primero, como consecuencia del precio. Por este motivo y debido a que la transmisión sólo se realizará entre el cabezal y un ordenador, el puerto del sónar se ha configurado con la primera opción.

El modelo lleva incorporado otro puerto conocido como *AUX*, utilizado para conectar dispositivos adicionales a través de los que se pueden obtener y enviar datos, como por ejemplo brújulas, sensores de presión y GPS. No es menester de este proyecto tratar con esta entrada adicional del cabezal.

La tabla 2.1 muestra información detallada acerca del dispositivo.

Especificaciones	
Frecuencia operacional	675 kHz
Tamaño de haz, vertical	40°
Tamaño de haz, horizontal	3°
Rango	100 metros
Ancho de banda del sistema	35 kHz
Sectores de scan	360° continuos, 180° en una dirección o sectores angulares
Velocidad de paso	1.8°, 0.9°, 0.45° o 0.225°
Cambio de dirección instantáneo	Sí
Medida de imagen	Sí
Operación invertida del cabezal	Sí
Alimentación	12, 24, 48 VDC @ 6 VA
Comunicación	RS232/RS485 optoaislado
Longitud del cable	Máximo 1200 metros (RS 485)
Control	Ordenador con comunicaciones serie estándar
Software	Tritech SeaNet(OEM)/Protocolo de comandos a bajo nivel
Longitud máxima	180 mm
Anchura máxima	88 mm
Altura máxima	76 mm
Peso en aire	1.1 kg
Peso en agua	0.5 kg
Profundidad máxima de operación	1000 metros
Material	Aluminio
Temperatura de operación	-10℃ a 35℃
Temperatura de almacenamiento	-20℃ a 50℃

Tabla 2.1: Características técnicas del sónar Miniking

2.5. Modo de funcionamiento

Ya hemos mencionado que el sónar envía pulsos y escucha los posibles ecos producidos a raíz de su interacción con el entorno. Estos ecos llegan al dispositivo en forma de señal ultrasónica, la cual no tiene ningún valor por sí sola. Por tanto, es importante explicar el mecanismo mediante el cual el cabezal procesa esta señal y devuelve información útil, a partir de la que se podrá determinar la distancia a la que se encuentran los objetos.

En primer lugar debemos diferenciar los conceptos de *ping* y *pulso*. Se conoce como *ping* a la señal acústica que envía el sónar de la cual se espera recibir el eco. Un *ping* puede estar formado por varios *pulsos*. Otro concepto importante es el de *scanline*. Se puede definir como el conjunto de intensidades registradas por el sensor en una dirección concreta.

En el proceso entran en juego una serie de parámetros configurables a través del protocolo de comunicación del sónar por el puerto serie (ver sección 2.7). El primero de ellos es el **rango**, que se define como la distancia máxima a la que el dispositivo espera detectar obstáculos. Es decir, define la distancia máxima que puede estar un objeto para que sea detectable por el sónar. El segundo parámetro importante es el **número de *bins***. La idea es discretizar el primer parámetro, el rango, en un número determinado de pequeñas unidades que representen una distancia concreta. Estos intervalos son denominados *bins* por *Tritech*. El valor de la intensidad del eco para cada *bin* es devuelto en una trama de información que contiene todas las muestras correspondientes a una *scanline*. Tras estas definiciones, estamos en condiciones de explicar en qué consiste el proceso. En primer lugar veamos cómo se discretiza el tiempo total de vuelo del pulso.

Lo primero es calcular el máximo tiempo de vuelo del pulso. Sabemos la distancia que vamos a analizar (rango) y la velocidad aproximada del sonido en el agua[1]. Por ello, podemos calcular el tiempo máximo de vuelo del ultrasonido, que obtenemos de:

[1]Esta velocidad depende de diferentes factores del medio, como la presión y la temperatura. En condiciones generales, se utiliza 1550 m/s.

$$t_f = 2 \times \frac{e}{v}\,, \tag{2.1}$$

donde t_f es el tiempo que tarda el pulso en ir y volver la distancia correspondiente al rango, e es el rango configurado en el sónar y v es la velocidad del sonido en el agua en m/s. Necesitamos guardar las intensidades de los ecos durante ese tiempo. El problema es que este tiempo es una variable contínua, y por tanto no representable de forma digital, por lo que es necesario discretizarla. Según el número de *bins*:

$$t_b = \frac{t_f}{n}\,, \tag{2.2}$$

donde n es el número de *bins* configurado en el sónar y t_b el tiempo que transcurre en ir y volver dentro del un mismo *bin*. Así, hemos conseguido subdividir el tiempo de vuelo en intervalos de t_b unidades de tiempo. Sin embargo, para rangos muy elevados, es posible que este t_b tenga un valor alto, siendo poco precisa la localización del obstáculo. Para solucionar esto, el dispositivo subdivide nuevamente el tiempo de *bin* en intervalos de 640 ns:

$$muestras = \frac{t_b}{640\ ns}\,, \tag{2.3}$$

donde *muestras* indica el número de unidades de 640 ns en las que se divide cada *bin*. Por lo tanto, el tiempo de vuelo de un *ping* está dividido en un número entero de *bins*, y a su vez, cada *bin* está muestreado en unidades de 640 ns. Ésta es la jerarquía con la que trabaja el cabezal. Internamente consta de dos contadores, uno de tiempo y otro de intervalos de 640 ns. Tras haber enviado un pulso se espera un cierto tiempo, llamado *tiempo de anulación de eco*[2], para no escuchar la propia señal que se acaba de enviar. En este momento se inicia el contador temporal a la vez que el sensor escucha el medio. Cada 640 ns, se incrementa el contador de intervalos y se registra el valor de la señal en ese momento. Cuando se ha completado el número de intervalos correspondientes a un *bin*, se ponderan los valores de las muestras que contiene dicho *bin*, se registra su valor, se reinician los contadores y se vuelve a aplicar el proceso para el siguiente *bin* del pulso. Una vez completados todos los *bins*, con las intensidades se compone un

[2]En inglés, *blanking time*

paquete *mtHeadData*, explicado en la sección 2.7, que se envía a través de la conexión serie al computador.

2.6. Interpretación de datos

Una vez que el ordenador recibe la trama correspondiente, debe interpretar los datos que contiene para poder determinar los posibles obstáculos y la distancia a la que se encuentran. Las intensidades de cada *bin* están registradas dentro del paquete enviado por el cabezal. Dependiendo de los umbrales de intensidad que se deseen aplicar, un valor puede significar que en ese *bin* hay un objeto. Los *bins* representan tiempo. Por tanto, debemos realizar algunos cálculos para obtener la distancia que representan.

Despejando de la fórmula 2.1 obtenemos cómo calcular el rango de un *bin*:

$$e_k = \frac{v \times t_k}{2}\,, \tag{2.4}$$

donde t_k es el tiempo en ir y volver al *bin* k y e_k la distancia a dicho *bin*.

En este caso, la única incógnita variable es el tiempo, que es específico dependiendo de la posición de *bin* que queremos calcular. El tiempo en segundos será:

$$t_k = k \times adinterval \times 640\ ns\,, \tag{2.5}$$

donde k representa el número de *bin* concreto del cual se desea obtener la distancia y *adinterval* es el número de intervalos de 640 ns en los que se divide cada *bin*. Este último parámetro depende del rango y del número de *bins* (ver sección 2.5), y debe configurarse en el cabezal.

Por defecto, el sónar devuelve un byte por cada *bin*. De esta forma, cada intensidad está codificada con 8 bits y puede tomar 256 valores. Sin embargo, el dispositivo se puede configurar de tal manera que empaquete 2 muestras en un byte, reduciéndose el número de valores de inten-

sidad posibles a 16, ya que cada muestra se codifica con 4 bits. Este cambio aumenta la rapidez del proceso a costa de perder precisión en la codificación de intensidades.

La trama de información contiene además una serie de parámetros que dan información acerca de la *scanline*. Entre ellos, destaca *bearing*, que indica la posición del cabezal en el momento de enviar el pulso. Las unidades con las que trabaja son gradianes, siendo necesaria una transformación para obtener grados (90° = 100 gradianes).

Se puede modificar la sensibilidad del dispositivo a través de un mecanismo de control de rango dinámico incluido en él, que en definitiva actúa como un filtro del eco recibido. Los límites del filtro se configuran en el sónar en decibelios (dBs). El receptor es capaz de aceptar señales de hasta 80dB, que indica que la relación entre la potencia máxima aceptada y la mínima puede ser como mucho 80. Esto es:

$$RD = 10 \times \log\left(\frac{P_{max}}{P_{min}}\right), \tag{2.6}$$

donde P_{max} y P_{min} son las potencias máxima y mínima aceptadas por el sónar, respectivamente. Existen una serie de valores configurables en el cabezal que nos permiten modificar estos límites:

- *ADLow:* Establece la relación que debe existir entre la potencia de la señal de eco y la potencia más baja aceptada por el sónar para que la señal se considere válida. Es decir:

$$ADLow = 10 \times \log\left(\frac{P_{sig}}{P_{min}}\right), \tag{2.7}$$

  donde P_{sig} es la potencia de la señal recibida. El incremento de este valor permite reducir la sensibilidad del sónar, filtrando posibles ruidos del entorno. Si despejamos de la fórmula 2.7 la relación entre potencia y ruido, obtenemos:

$$\frac{P_{sig}}{P_{min}} = 10^{\frac{ADLow}{10}}, \tag{2.8}$$

  obteniendo así el número de veces que, como mínimo, la potencia del eco recibido debe

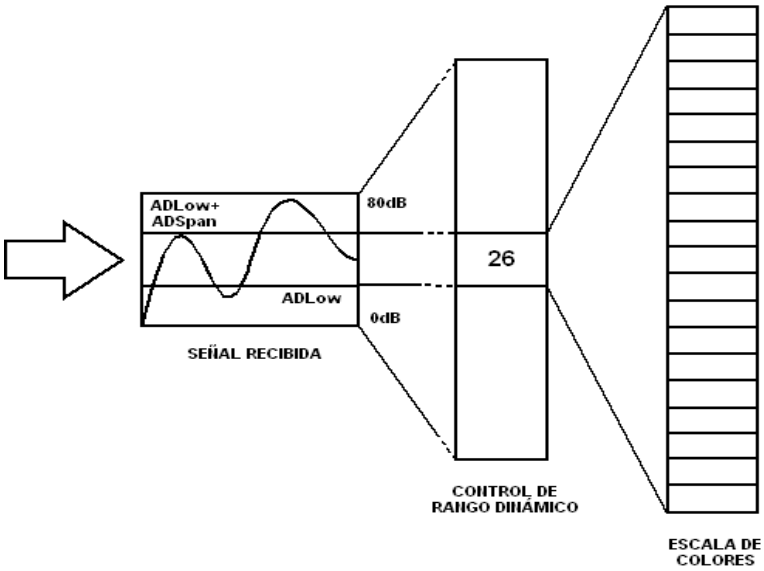

Figura 2.3: Control de rango dinámico

superar a la mínima potencia detectable por el sónar para que la señal sea aceptada. Por ejemplo, si establecemos *ADLow* a 40 dB y sustituimos este valor en la fórmula anterior, obtenemos:

$$\frac{P_{sig}}{P_{min}} = 10^4,$$

que indica que sólo se aceptarán señales tales que como mínimo sean 10000 veces más intensas que la señal mínima detectable.

- *ADSpan:* Indica la amplitud en decibelios del filtro, partiendo del límite inferior. Una vez establecido este valor, el sónar detectará señales entre *ADLow* y *ADLow + ADSpan*.

Estos dos parámetros controlan el *mapping* entre el nivel de señal recibida y su valor digital. En caso de estar trabajando con 8 bits por *bin*, un valor de *ADLow*(dB) se codificará con el valor 0, mientras que un valor de *ADLow + ADSpan*(dB) se codificará con el valor 255. Por ejemplo, si configuramos el sónar con valores *ADLow* de 13dB y *ADSpan* de 12dB, una señal de 13dB se codifica con el valor 0 mientras que una señal de 25dB se codifica con el valor 255.

Ejemplo

Para concluir esta sección, vamos a poner un ejemplo práctico y gráfico sobre cómo detectar un obstáculo a partir de los datos que devuelve el sónar. Estamos operando con un dispositivo que vamos a configurar para un rango de *100 metros* y el número de *bins* a *250*. Devolverá un byte por muestra (por defecto) y se utilizará una velocidad de sonido de *1467 m/s*.

En primer lugar, debemos calcular el número de intervalos de 640 ns que tendrá cada muestra, ya que debemos configurar este parámetro en el sónar. Para ello, necesitamos el tiempo total de vuelo del pulso, que obtenemos de aplicar la fórmula 2.1:

$$t_f = 2 \times \frac{e}{v} = 2 \times \frac{100}{1467} = 0{,}136\ s$$

El siguiente paso es dividir ese tiempo de vuelo (t_f) en tiempos de *bin*, lo cual se hace aplicando la fórmula 2.2:

$$t_b = \frac{t_f}{n} = \frac{0{,}136}{250} = 0{,}000545\ s/bin$$

Una vez calculado el tiempo que debe durar cada *bin* (t_b), necesitamos fijar el número de muestras de 640 ns en que se dividirá cada uno. Se utiliza la fórmula 2.3:

$$muestras = \frac{t_b}{640ns} = \frac{0{,}000545}{640 \times 10^{-9}} \approx 852\ muestras$$

Lo siguiente es configurar el sónar con las opciones deseadas, incluido este último valor calculado. Dicho proceso se explica en la siguiente sección. De momento se supondrá que se ha llevado a cabo con éxito y que el cabezal está operando sin ningún tipo de problema. Supongamos ahora la situación de la figura 2.4, la cual representa que hay un objeto a 225° a una distancia aproximada de 70 metros.

El sistema empieza a realizar *pings* como muestra la figura 2.5. Las señales de eco recibidas van pasando por el filtro de control de rango dinámico, para comprobar si se encuentran dentro de los límites preestablecidos por el usuario. Los *pings* 50, 51 y 52 son los que incluyen el eco pro-

ducido por el objeto A (ver figura 2.6), y por tanto, las *scanlines* a analizar con más detenimiento.

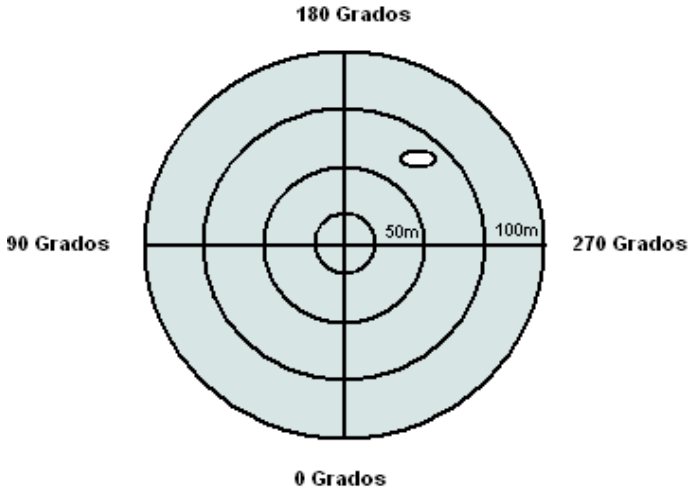

Figura 2.4: Escenario de ejemplo

En este caso, prestemos atención al pulso 50. En la figura 2.7 se puede comprobar como, tras aplicar el filtro, son los *bins* próximos al 175 los que tienen un suficiente nivel de intensidad de señal, lo cual puede significar la presencia de un obstáculo. Para saber a qué distancia se encuentra, primero necesitamos saber el tiempo que representa ese *bin*. A partir de la fórmula 2.5, obtenemos:

$$t_k = k \times adinterval \times 640\ ns = 175 \times 852 \times 640 \times 10^{-9} = 0{,}0954\ s$$

Con este resultado podemos obtener la distancia recorrida en ese tiempo. Habrá que dividirla entre dos, ya que esa cifra indica el tiempo en ir y volver del *bin*. Así pues, a partir de 2.4, calculamos la distancia:

$$e = \frac{v \times t}{2} = \frac{1467 \times 0{,}0954}{2} = 69{,}993\ m$$

Ésta es la forma en la que se deben interpretar los datos obtenidos del sónar. En la mayoría de casos se utiliza una aplicación para gestionar la comunicación entre el dispositivo y el ordenador. Por tanto, no es necesario que el usuario calcule estas cifras, ya que se puede programar la máquina para que lo haga. En multitud de ocasiones se deseará mostrar todos los valores del

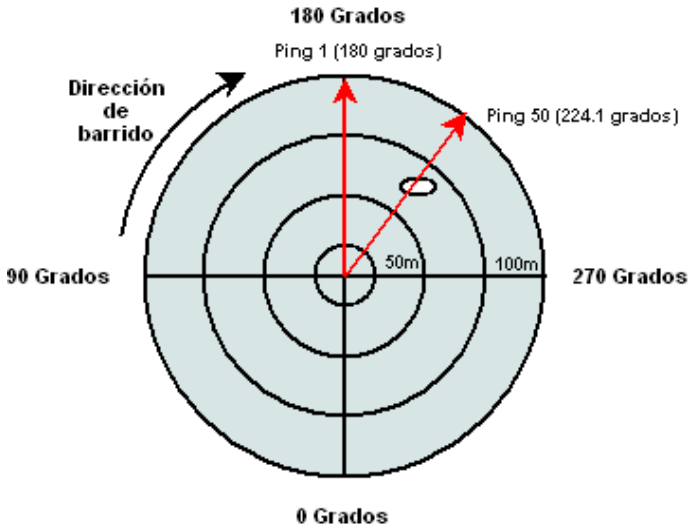

Figura 2.5: Barrido del entorno del sónar

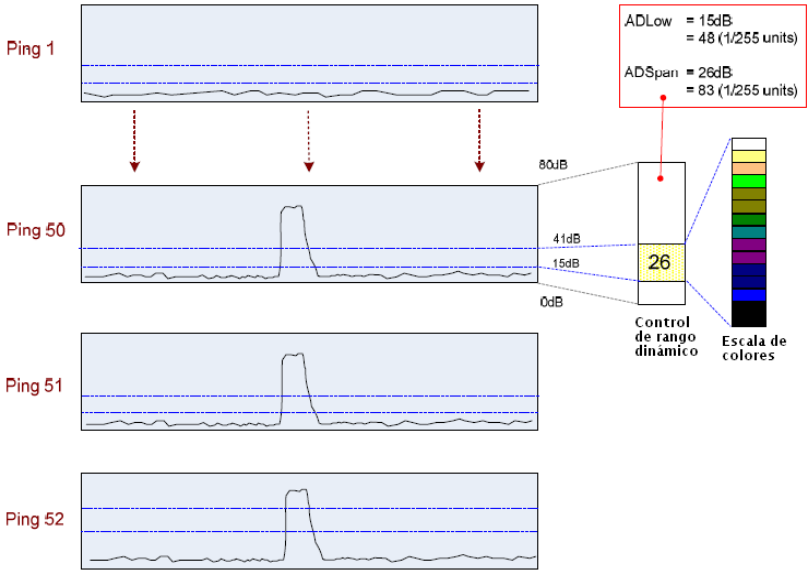

Figura 2.6: Scanlines

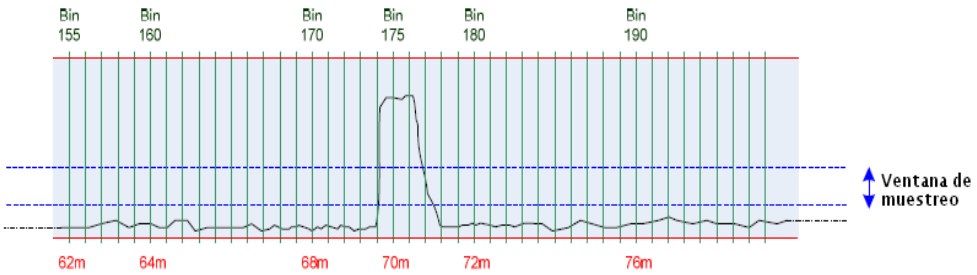

Figura 2.7: Intensidad de eco de los *bins* del pulso

pulso de forma gráfica. En estos casos se suele representar la *scanline* de acuerdo con un mapa de color codificado por niveles de intensidad.

2.7. Protocolo SeaNet

Para que el sónar y el computador conectado al puerto serie interactúen de forma ordenada, es necesario un protocolo de comunicación, conocido por ambos. El primer gran objetivo de este proyecto es la programación de una librería que implemente este protocolo y que sirva para que las aplicaciones de usuario puedan comunicarse con el cabezal. A continuación se detalla el protocolo que utiliza el sónar *Miniking*.

Todos los dispositivos de *Tritech* utilizan un protocolo propietario denominado **SeaNet**, diseñado para interactuar con ellos. Es una comunicación del estilo «comando-respuesta», que opera de una forma similar a las ARCNET LAN[3]. El sónar envía tramas de información que la aplicación de usuario debe captar para obtener datos del entorno y tomar decisiones a partir de ellos. De forma inversa, el programa puede enviar paquetes para configurar el dispositivo o hacer peticiones al cabezal.

Anteriormente, el protocolo era conocido como *SONV3*. Había dos versiones: *V1.4* y *V1.5*. El protocolo *V1.5* es mucho más parecido a *SeaNet* que la versión *V1.4*. *SeaNet* es un avance de

[3]Arquitectura de red de área local desarrollado por *Datapoint Corporation* que utiliza una técnica de acceso de paso de testigo como *Token Ring*. La topología física es en forma de estrella, utilizando cable coaxial y *hubs* pasivos o activos.

la versión *V1.5*, por lo que en ocasiones se le conoce como *SONV4*. Las principales diferencias con respecto a *V1.4* son:

- Todas las tramas de información empiezan con el carácter «@» en lugar de «%».
- *SeaNet* incluye en la cabecera de cada paquete dos campos identificadores, SID y DID, que indican la fuente y el destino de la trama.
- Todos los comandos terminan con un carácter especial de valor 0x0A. Debemos prestar atención a la longitud de la trama, ya que este número también puede aparecer en los datos. No podemos asegurar que la aparición de este carácter sea el final del paquete.
- En *V1.4* era el programador el encargado de iniciar la comunicación con el sónar. Ahora, el dispositivo comienza a lanzar paquetes nada más conectarse a la alimentación a razón de 1 mensaje por segundo. Una vez la aplicación del usuario ha captado una de estas tramas, pasa a tener el control.

2.7.1. Descripción general

En la figura 2.8 se puede ver un diagrama de flujo del funcionamiento básico del protocolo. Consiste en intercambios de tramas de información entre el ordenador y el sónar a través del puerto serie.

El dispositivo empieza a enviar paquetes *mtAlive* desde el momento en que se conecta a la alimentación, a razón de uno por segundo. En esta trama se incluye información diversa sobre el sónar, como la posición del motor o la existencia de una configuración válida en el cabezal. Seguirá enviando estas secuencias mientras no tenga los parámetros correctos para su funcionamiento.

Nada más ejecutarse, la aplicación de usuario debe recibir un *mtAlive*. En caso de no ser así, puede indicar que el sónar está trabajando con *V1.4*, en la que la propia aplicación era la encargada de iniciar la comunicación. El siguiente paso consiste en que el programa envía una orden

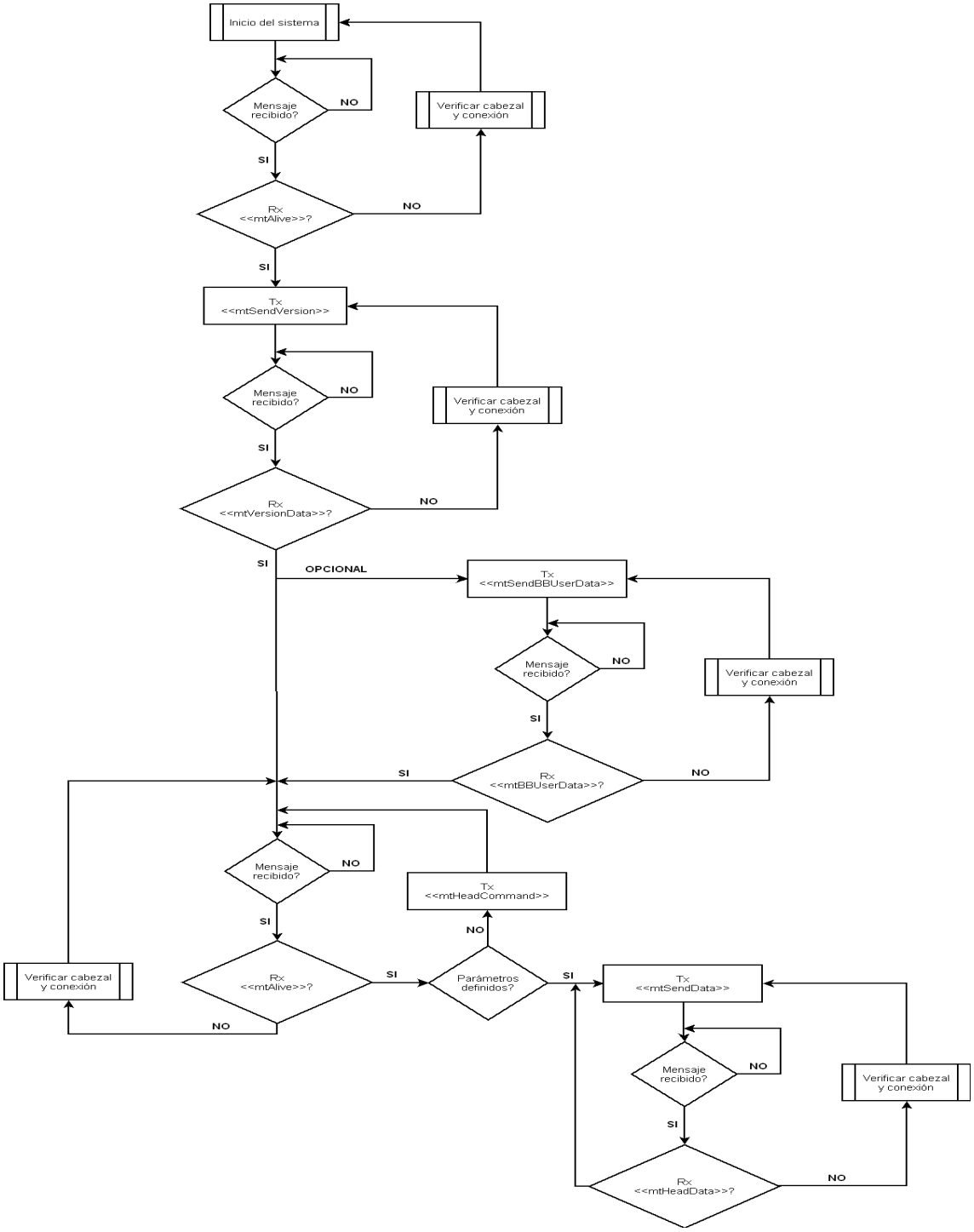

Figura 2.8: Protocolo SeaNet

llamada *mtSendVersion*, que interroga al dispositivo acerca de la versión de software instalada en él, y espera su respuesta, que viene dada en una trama *mtVersionData*. Opcionalmente se puede enviar el comando *mtSendBBUser*, al que el sónar responde con *mtBBUserData*. Éste último contiene información acerca de parámetros físicos contenidos en el sónar, como la existencia de un puerto AUX.

Tras esto, el programa de usuario debe comprobar si están configurados los parámetros de funcionamiento en el dispositivo. Está indicado en las tramas *mtAlive* recibidas anteriormente. En caso afirmativo, el sónar está preparado para operar correctamente. De no ser así, es necesario enviarle estos parámetros. Se hace a través del comando *mtHeadCommand*, que incluye todas las opciones necesarias para su configuración. La aplicación debe recibir un nuevo *mtAlive* para comprobar que la operación se ha llevado a cabo con éxito.

Se debe repetir este proceso hasta que el sónar tenga una configuración correcta. En ese momento, estaremos en condiciones de interactuar con él.

Para obtener una *scanline* se envía la trama *mtSendData*. El cabezal responde con *mtHeadData*, que contiene todas las intensidades de los *bins* de la *scanline* (ver sección 2.5). Si el puerto serie está configurado como *RS232*, el sónar devuelve dos tramas de respuesta, la correspondiente a la *scanline* actual y la correspondiente a la siguiente. En caso de que esté configurado con *RS485*, devuelve únicamente una.

Por último, cabe destacar que se puede reiniciar el dispositivo en cualquier momento, enviando una trama *mtReboot*. Tras esto, la configuración es eliminada, y por tanto es necesario volver a enviar un paquete *mtHeadCommand* para reconfigurar el sónar.

Este es el funcionamiento básico del protocolo. Existen una serie de tramas que se utilizan para tareas avanzadas que no entran dentro del ámbito del proyecto y, por tanto, no se han tenido en cuenta.

2.7.2. Clasificación de tramas

En la sección anterior hemos visto todas las tramas que circulan por el puerto serie dentro del protocolo. Algunas se envían desde el ordenador con destino al sónar y otras lo hacen en dirección contraria. Por tanto, de acuerdo con este criterio, podemos clasificarlas en dos grupos:

- **Comandos:** Son aquellos que van desde la aplicación de usuario al sónar. Contienen órdenes de interacción con el dispositivo. En este grupo se incluyen:
  - *mtSendData*
  - *mtReboot*
  - *mtHeadCommand*
  - *mtSendBBUser*
  - *mtSendVersion*
- **Mensajes:** Discurren desde el cabezal hasta el ordenador. Contienen información útil y suelen ser enviados como respuesta a algún comando. Pertenecen a este grupo:
  - *mtAlive*
  - *mtHeadData*
  - *mtBBUserData*
  - *mtVersionData*

2.7.3. Formato general de las tramas

Cada trama del protocolo contiene una información determinada, organizada de una cierta forma. Es decir, cada paquete tiene estructura y formato propio. En esta sección no se pretende dar una descripción exhaustiva de cada uno de estos formatos, sino mostrar una serie de campos comunes a todas las tramas del protocolo.

Estos campos forman, en su mayoría, la cabecera de los paquetes. Son los siguientes:

Trama	Identificador
mtVersionData	1
mtHeadData	2
mtBBUserData	6
mtReboot	16
mtHeadCommand	19
mtSendVersion	23
mtSendBBUser	24
mtSendData	25

Tabla 2.2: Identificadores de trama

- **HDR:** Carácter de principio de trama («@»).
- **LTH H:** Representación carácter a carácter de la representación hexadecimal de la longitud del paquete. Son 4 bytes.
- **LTH B:** Representación binaria de la longitud del paquete. Son 2 bytes.
- **SID:** Indica la fuente del mensaje. Dentro del protocolo, cada nodo está identificado con un número. El sónar suele ser el número 2, mientras que el ordenador del usuario debería ser configurado para actuar como nodo 255.
- **DID:** Indica el destino del mensaje (ver explicación anterior).
- **COUNT:** Tamaño en bytes desde el siguiente campo hasta el final del paquete, sin incluir el carácter de final de trama.
- **MSG:** Contiene la información propia del paquete. El primer byte de este campo es un identificador único. Cada trama tiene un número entero que la identifica en el protocolo. En la tabla 2.2 se pueden ver los identificadores para cada paquete.
- **LF:** Carácter que representa el final del paquete. No forma parte de la cabecera. Tiene un valor de 0Ah.

2.7.4. Descripción de las tramas

Para terminar este capítulo, se va a hacer una descripción de las tramas que componen el protocolo. Ya se han mencionado las funcionalidades básicas de cada una al explicar el proceso de diálogo entre el cabezal y el computador. En este apartado se intentará profundizar un poco más en ellas. En cualquier caso, en el apéndice A se puede encontrar una descripción pormenorizada de cada tipo de trama.

Comandos

- *mtSendData:* Esta orden se envía para obtener una *scanline*. Al recibir este paquete, el sónar explora a lo largo de dos orientaciones consecutivas mediante dos operaciones transmisión-recepción, y deja preparado el cabezal para la siguiente petición. Dependiendo de la configuración que se tenga del puerto serie devolverá uno o dos mensajes *mtHeadData*. Es una trama de 18 bytes que incluye además la hora actual en milisegundos para configurarla en el dispositivo receptor. Nada más arrancar, el sónar establece la hora a 00:00:00. Con el sucesivo envío de esta trama se va actualizando este valor.

- *mtReboot:* Este paquete reinicia el cabezal. Tras esto, se deben volver a enviar los parámetros de configuración, ya que han sido borrados. Es especialmente útil cuando no es posible realizar un reinicio físico del sónar.

- *mtHeadCommand:* Se utiliza para dar al cabezal instrucciones sobre cómo debe operar. Sin esta información, no es capaz de funcionar y no responde a las peticiones de exploraciones. Incluye opciones como ganancia, tamaño de muestreo y el rango, entre otras. Cualquier actualización en la configuración requiere que se vuelva a enviar esta trama. Recibe como respuesta dos *mtAlive*. Su tamaño es variable, ya que dependiendo del modelo de *Tritech* con el que estemos trabajando incluye más o menos campos. En el caso del sónar *Miniking*, está formado por 66 bytes.

- *mtSendBBUser:* Se envía para preguntar sobre diferentes opciones instaladas y programadas en el sónar, como por ejemplo la existencia de un puerto AUX, configuración de sensores

de presión/temperatura y telemetría. Es una secuencia de 14 bytes, que es respondida con el mensaje *mtBBUserData*.

- *mtSendVersion:* Se utiliza para interrogar al cabezal acerca de la versión del software instalada en él y puede ser útil para futuras actualizaciones provistas por *Tritech*. Se debe enviar antes de realizar cualquier operación de escaneo. Es una trama de 14 bytes, similar al resto de comandos salvo en su identificador. El receptor contesta a través de un mensaje *mtVersionData*.

Mensajes

- *mtAlive:* Sirve como respuesta a *mtHeadCommand*. El sónar envía este mensaje de forma reiterativa nada más conectarse a la alimentación con una frecuencia de 1 Hz. Incluye información sobre la existencia de parámetros en el sónar, así como datos importantes, por ejemplo la posición del motor en un momento determinado. Son 22 bytes de información.

- *mtHeadData:* Devuelve las intensidades de los *bins* de una *scanline*. Se genera en respuesta a un comando *mtSendData*. Dependiendo de como esté configurado el sónar (comando *mtHeadCommand*), se devuelve un byte por *bin* o se empaquetan en un byte dos *bins*. Su tamaño es variable, ya que depende del número de *bins* que configuremos en el sónar.

- *mtBBUserData:* Se genera en respuesta al comando *mtSendBBUser*. Contiene varias configuraciones y opciones programadas o incluidas en el sónar. Incluye información general, telemetría de los puertos del cabezal y características básicas del dispositivo.

- *mtVersionData:* Son enviadas por el cabezal en respuesta al comando *mtSendVersion*. Contiene un campo que indica la versión del software, que debe ser almacenada en el programa de usuario para realizar las comprobaciones correspondientes. Lleva además información sobre la CPU instalada, que suele incluir un identificador único. Tiene un tamaño de 22 bytes.

Capítulo 3

Diseño e implementación de una librería de interfase con el sónar Miniking de Tritech

En este capítulo se explica el diseño y la implementación de una librería de interfase con el sónar, la cual ha sido diseñada de acuerdo con una arquitectura por capas. Se comentan las funcionalidades básicas de cada uno de estos niveles y las clases que los componen. El capítulo concluye detallando los métodos necesarios para interactuar con el dispositivo, así como ejemplos de utilización de la librería.

3.1. Arquitectura de capas

En el capítulo anterior se ha detallado el funcionamiento del protocolo *SeaNet*, utilizado en todos los dispositivos de *Tritech* y en particular en el sónar objeto de estudio en este proyecto, el modelo *Miniking*. Para poder interactuar con él, es necesario que el computador con el cual está conectado a través del puerto serie ejecute una aplicación que entienda este protocolo. Esta tarea implica la ejecución de procesos como el intercambio de tramas de información con el dispositivo en el orden correcto así como el procesamiento de los datos que llegan por el puerto.

La implementación de una librería de interacción con el sónar que implemente las funcionalidades necesarias evita que el programador tenga que programar estos procedimientos cada vez que desee crear una aplicación que requiera interacción con el sónar. Es más, puede que no desee conocer el funcionamiento del protocolo y únicamente necesite obtener información del dispositivo. En este caso, la creación de la librería evita al programador tener que conocer los detalles de implementación del protocolo, lo cual le permite centrarse únicamente en el desarrollo de su aplicación.

Este intercambio de tramas entre el computador y el sónar implica la necesidad de leer y escribir a través de un puerto serie. Éstos últimos son, por tanto, procedimientos a más bajo nivel, que son utilizados por funciones de carácter más general. A su vez, dentro del protocolo existen una serie procesos que necesitan de una secuencia de intercambio de tramas entre el dispositivo y el computador, que se deben implementar para liberar al programador de esta tarea. Por ejemplo, para reiniciar el sónar debemos enviar un comando *mtReboot* y posteriormente *mtHeadCommand* de forma iterativa hasta que el dispositivo nos indique que tiene una configuración correcta para su funcionamiento. Dichos procesos hacen uso de los métodos de intercambio de paquetes y son, por ello, de un nivel de abstracción mayor.

Debido a esta relación de dependencia entre procesos y al propio funcionamiento del protocolo *SeaNet*, se ha optado por estructurar la librería de acuerdo con un *modelo de capas*, dividiéndola en diferentes niveles de abstracción, con funciones perfectamente definidas, donde cada capa hace uso del nivel inmediatamente inferior, simplificándose así el desarrollo de la librería. Cada nivel está compuesto por una serie de clases que implementan los procedimientos necesarios para dar servicio a la capa superior.

Estructurar la librería mediante una *arquitectura de capas* facilita la tarea del programador, que en un momento determinado únicamente debe prestar atención al desarrollo de una capa concreta, sin preocuparse por el resto de niveles. Además, se aisla la lógica de la aplicación en componentes separados, reutilizables en otras aplicaciones. Por último, permite modificar la im-

plementación de las capas sin que éste hecho afecte al resto de niveles.

Se ha optado por programar la librería con el lenguaje *C++*, una extensión del lenguaje *C* que permite trabajar tanto a alto como a bajo nivel de una forma eficiente. Permite el desarrollo de aplicaciones bajo el paradigma de la *orientación a objetos*, por lo que así hemos podido aprovechar las características que éste nos ofrece.

Así pues, la arquitectura software de la librería está estructurada en tres capas:

- **Capa 1:** Es el nivel más bajo de los tres y se encarga de proporcionar una serie de primitivas que permitan a la capa 2 acceder al puerto serie del computador.
- **Capa 2:** En este nivel intermedio se implementan los comandos y mensajes que componen el protocolo *SeaNet* (ver sección 2.7.2). Cada clase incluye los atributos necesarios para almacenar todos los valores de la trama, así como métodos para recibirla o enviarla, dependiendo de su tipo.
- **Capa 3:** Es el nivel más abstracto de la jerarquía. El usuario interactúa con el sónar mediante los métodos programados en esta capa. Las funciones de este nivel implican la gestión de una secuencia de tramas a través del protocolo RS232 entre el dispositivo y el computador. Dichas funciones son totalmente transparentes para el programador.

3.2. Capa 1: comunicación serie

La principal función de esta capa es proporcionar al nivel superior una serie de métodos para permitir el acceso al puerto *RS232*. Mediante los procedimientos incluidos en este apartado se pueden leer del puerto serie bytes de información enviados por el sónar, así como transmitir a través de dicho puerto secuencias de bytes al dispositivo.

Esta capa está formada por una única clase llamada *COMPort*. Dispone de atributos y métodos para configurar los parámetros necesarios en una comunicación serie, como por ejemplo la

Figura 3.1: Arquitectura *software* de la librería

tasa de bit o la paridad. Destacan los métodos *read* y *write*, que permiten leer y escribir bytes sobre el puerto.

La clase *COMPort* está basada en el API *Win32* del sistema operativo *Windows*. Esta interfaz proporciona una serie de primitivas residentes en bibliotecas (generalmente dinámicas) que permiten a una aplicación la interacción con el sistema. Dichas primitivas están escritas en lenguaje *C*, por lo que pueden ser llamadas desde una clase programada en *C++* sin ningún problema.

La necesidad de disponer de la interfaz *Win32* para emplear esta clase limita el uso de la librería a entornos *Windows*, ya que dicha interfaz no se encuentra disponible en otras plataformas. Sin embargo, gracias a la arquitectura dividida en capas de la librería, para utilizarla en otra plataforma bastaría con volver a implementar los métodos de la clase *COMPort* mediante las primitivas de acceso de dicha plataforma sin necesidad de modificar el resto de los niveles software.

Debido a que es el nivel más bajo de la arquitectura, el programador no necesita conocer la existencia de esta capa para emplear la librería. Sin embargo, es perfectamente reutilizable en otras aplicaciones. Por ello, a continuación se muestra un ejemplo sencillo de cómo utilizar la clase *COMPort* para acceder al puerto serie:

```
#include "COMPort.h"
#include <iostream>

using namespace std;

int main(void) {

    char buffer[20];

    COMPort puerto("COM1", br9600, db8, None, sb1, 0, false);

    puerto.write("Hola mundo");
    puerto.read(buffer,20);
}
```

El programa crea un objeto *COMPort* llamando al constructor de la clase, envía la cadena de carácteres «Hola mundo» a través del puerto *COM1* del computador y recibe 20 carácteres de dicho puerto. Los parámetros que recibe dicho constructor son el puerto serie por el que se desea conectar con el sónar, la tasa de bits, los bits de datos, la paridad, los bits de parada, el *timeout* de la conexión en segundos y la opción de activar el control de flujo en la comunicación, respectivamente.

3.3. Capa 2: comandos y mensajes

En este nivel se representan, a través de clases, cada una de las tramas que componen el protocolo *SeaNet*. Dichas clases incluyen las estructuras de datos necesarias para almacenar todos los campos que forman la trama en cuestión, así como procedimientos para modificar y acceder a sus valores.

Se han abstraído los procesos de envío y recepción de cada tipo de paquete dentro de las propias clases, mediante métodos llamados *send* y *receive*, respectivamente. Dicha abstracción

simplifica enormemente la tarea de intercambiar bytes a través del puerto serie. Para recibir un mensaje del sónar basta con crear un objeto del tipo necesario y llamar a su método *receive*, que se encarga de la lectura completa y ordenada de los bytes que forman la trama y almacena su valor en variables definidas dentro de la clase. Del mismo modo, para enviar un comando al dispositivo únicamente debemos crear el objeto y llamar al procedimiento *send*, que se encarga de componer el paquete y transmitirlo a través del puerto de forma ordenada.

En la sección 2.7.3 se mostraron una serie de campos que son comunes a todas las tramas del protocolo. Por otro lado, existen clases que comparten ciertos métodos, como es el caso de *send* para los comandos y *receive* para los mensajes. Debido a estas causas, se ha decidido implementar en este nivel una estructura jerárquica de clases, aprovechando las ventajas que la programación orientada a objetos nos ofrece. Las clases situadas en un nivel de abstracción mayor incluyen atributos y métodos que son heredados por las clases derivadas. A parte, cada clase define sus propias estructuras y métodos según su formato. En la figura 3.2 se muestra una representación gráfica de la jerarquía de clases sobre la que se ha estructurado esta capa. A continuación se explican las características básicas de cada una de ellas.

3.3.1. Clases abstractas

- *Packet:* Es la raíz de la jerarquía de clases. Incluye todos los campos mencionados en la sección 2.7.3, comunes a todos los paquetes que componen el protocolo *SeaNet*, y provee a su vez primitivas de acceso a estos valores. Dentro de esta clase se ha declarado un procedimiento abstracto llamado *printPacket* cuya función es mostrar por pantalla todos los atributos que componen una trama. Se ha definido como procedimiento abstracto debido a que cada paquete incluye campos propios, y por tanto debe implementarse para cada formato de trama concreto.

- *Message:* Esta clase abstracta deriva de *Packet*, por lo que hereda todos sus atributos y métodos. Declara el procedimiento abstracto *receive*, común a todos los mensajes y necesario para recibir una trama enviada por el sónar. Se define como procedimiento abstracto debido a que cada mensaje es de un tamaño diferente e incluye campos específicos de la

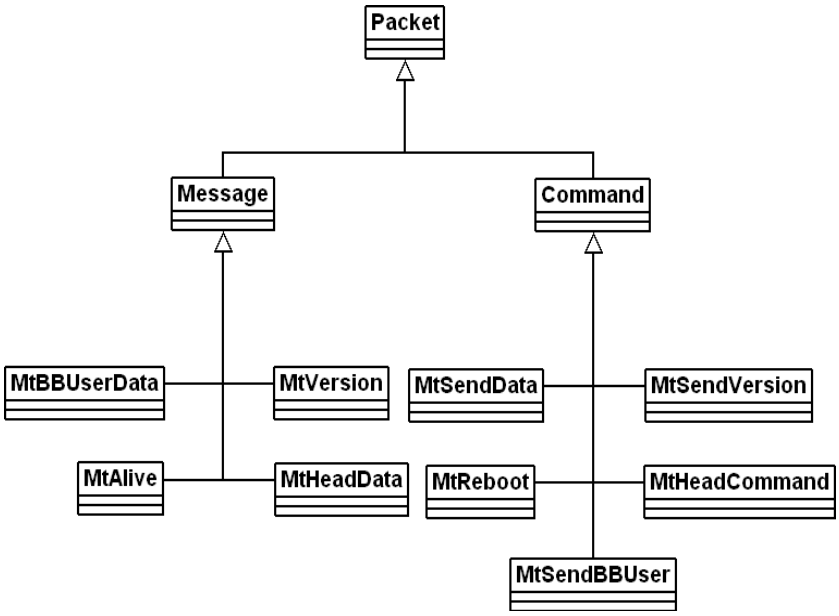

Figura 3.2: Jerarquía de clases

trama. Por tanto, debe ser programado para cada caso.

- *Command:* Es una clase abstracta en la que se incluyen todas las estructuras comunes a los comandos. Es muy similar a la clase anterior. La diferencia más notable es que define el método abstracto *send*, el cual construye la secuencia de bytes indicada en el formato de la trama y la envía a través del puerto serie de forma ordenada.

3.3.2. Mensajes

- *MtVersionData:* Esta clase representa la trama enviada por el sónar en respuesta a una petición de envío de versión. Contiene los campos de datos necesarios para almacenar el número de versión del software instalado y un identificador único del procesador del dispositivo, junto con métodos de lectura para acceder a ellos. Además, hereda todos los campos de *Message* y se han implementado los métodos *printPacket* y *receive* para este caso concreto. De ahora en adelante no se mencionará este hecho, ya que todas las clases heredan estos campos y reimplementan estos procedimientos.

- *MtBBUserData:* Representa la trama de respuesta del sónar al comando *mtSendBBUser*.

Incluye diferentes características generales del sónar, telemetría y funciones de acceso. Almacena esta información en variables, accesibles a través de métodos, que se hallan divididas en tres grandes bloques:

- *Características generales:* Este bloque es esencialmente una cabecera que incluye información del sistema, como el identificador que está utilizando el dispositivo dentro del protocolo (ver sección 2.7.3), entre otros.
- *Configuración telemétrica:* En este bloque de parámetros se definen las configuraciones telemétricas de cada puerto del sónar.
- *Perfil del sónar:* Este grupo de variables nos informa acerca de ciertas características físicas del dispositivo, como la existencia de un puerto auxiliar o de un motor.

- *MtAlive:* Esta clase se utiliza para recibir y procesar un mensaje *mtAlive*. Destaca el método *hasParams*, que nos permite saber si el dispositivo tiene una configuración de parámetros correcta para su funcionamiento. Además, incluye diversa información útil, como la posición angular del motor o el tiempo configurado en el cabezal.

- *MtHeadData:* Esta clase representa al mensaje devuelto por el sónar en respuesta a una petición de exploración. Incluye los valores de intensidad de los *bins* de un *scanline*. El tamaño de la trama es variable y depende de la configuración del dispositivo. Por tanto, el método *receive* es capaz de reservar memoria de forma dinámica una vez que ha leído el campo que indica la longitud en bytes de los *bins*. Estos datos son accesibles a través del método *getDataBytes*. Otros métodos destacados son *getBearing* para obtener la posición en grados del *scanline* y *getDataLength*, que nos permite saber la longitud en bytes de los datos almacenados en la propia clase.

3.3.3. Comandos

- *MtReboot:* Esta clase corresponde al comando que se debe enviar al sónar para reiniciarlo. Hereda los métodos *printPacket* y *send*. De nuevo, este hecho no se mencionará a partir de este momento, ya que todos los comandos reimplementan estos procedimientos. El método

send crea la trama en un *buffer* interno a la clase y lo envía de forma secuencial a través del puerto serie.

- *MtSendVersion:* Esta clase se utiliza para crear y enviar un comando *mtSendVersion*, que permite obtener información acerca de la versión de software instalada en el dispositivo. Únicamente difiere de la clase anterior en el identificador de trama enviado al sónar (ver tabla 2.2).

- *MtSendBBUser:* Esta estructura interpreta el comando utilizado dentro del protocolo para obtener información general acerca del cabezal. Es idéntico a los dos anteriores con la excepción del valor del identificador de trama.

- *MtSendData:* Nos permite enviar al sónar una petición de exploración. La diferencia con respecto a las anteriores es que el método *send* calcula la hora actual del día en milisegundos, ya que es un valor requerido dentro del formato de la propia trama.

- *MtHeadCommand:* Esta clase corresponde al comando utilizado para enviar una configuración al sónar. Para ello, la clase recibe por parámetro todos los valores necesarios para crear el paquete de configuración, como el rango o la ganancia, entre otros. El método *send* crea la trama en el *buffer* a partir de dichos valores y envía el paquete byte a byte a través del puerto.

3.4. Capa 3: MiniKing

En esta capa se encuentran las primitivas de más alto nivel de abstracción. Sus funciones principales son automatizar una serie de procesos que aparecen dentro del protocolo *SeaNet* y servir de interacción con la librería al usuario, que sólo necesita conocer los métodos aquí definidos para poder utilizarla.

Está formada por una única clase llamada *MiniKing*, que debe ser incluída e instanciada por el programador en su proyecto. Dicha clase posee como atributos objetos definidos en el nivel inferior, desde los que se envían y reciben las tramas necesarias para interactuar con el sónar.

También incluye los parámetros de configuración del dispositivo. Cualquier cambio en el valor de dichos parámetros se realiza en la propia clase, siendo necesario llamar al método *updateConfig* para que se lleve a cabo sobre el cabezal. Se ha implementado de esta forma debido a que para modificar la configuración del sónar es necesario enviar un comando *mtHeadCommand* cada vez. Si el usuario realiza demasiadas peticiones consecutivas, puede saturar al dispositivo y ralentizar su funcionamiento. Con este método se pueden realizar todos los cambios necesarios de una vez y enviar la nueva configuración al sónar en un único comando.

Como ya se ha mencionado, el protocolo *SeaNet* incluye una serie de tareas que requieren una secuencia de intercambio de tramas (ver figura 2.8). Dentro de esta capa existen procedimientos que representan dichas tareas. Por ejemplo, para poder enviar peticiones de exploración al cabezal es necesario todo un proceso que incluye el envío de la versión del software instalado y la configuración del dispositivo a través de *mtHeadCommand*. El programador sólo necesita llamar al método *initSonar* para llevar a cabo esta secuencia de intercambio de tramas y no necesita conocer la forma como se lleva a cabo el proceso. En la siguiente sección se explican las principales funciones que proporciona la librería para poder interactuar con el sónar.

3.4.1. Funciones de acceso público a la librería

Configuración del dispositivo

A través de estas funciones se pueden configurar los parámetros disponibles en el sónar. Los métodos tienen un equivalente *get*, que devuelve el valor del parámetro en cuestión.

- *set8Bits(bool):* Indica al dispositivo el número de bits por *bin* que debe devolver en las tramas de respuesta a peticiones de exploración. Con un valor *true*, se devolverá 1 *bin* por byte. En caso contrario, se devuelven 2 *bins* por byte.

- *setContinuous(bool):* Si se establece este valor a *true*, el dispositivo explora 360 grados. Con el valor a *false*, hace el barrido dentro de la sección que se encuentra entre unos límites llamados genéricamente *izquierdo* y *derecho*, explicados más abajo.

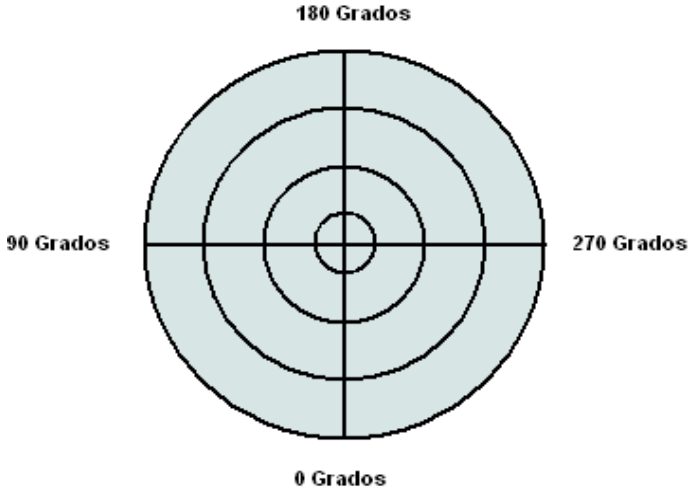

Figura 3.3: Configuración angular del sónar

- *setStare(bool):* Cuando esta opción está activa, el cabezal explora de forma iterativa la dirección señalada en el límite izquierdo.
- *setRange(int):* Configura la distancia máxima a la que el dispositivo espera detectar obstáculos. El parámetro viene dado en metros.
- *setLeftLim(int):* Indica el límite izquierdo de exploración. El parámetro viene dado en grados, de acuerdo a la configuración angular mostrada en la figura 3.3. Se pueden explorar secciones combinando este valor con el límite derecho y la función *setContinuous*.
- *setRightLim(int):* Establece el límite derecho para la exploración. El parámetro viene dado en grados, de acuerdo a la configuración angular mostrada en la figura 3.3.
- *setADLow(int):* Indica el valor mínimo de la relación entre la potencia de la señal de eco y la potencia más baja aceptada por el sónar para que la señal se considere válida. Es el límite inferior del mecanismo de control de rango dinámico, dado en decibelios.
- *setADSpan(int):* Representa el rango dinámico sobre *AdLow*, con unidades de decibelios.
- *setGain(int):* Indica la ganancia del dispositivo. El parámetro es un valor entre 0 y 99 en unidades de tanto por cien, por lo que representa ganancias entre 1.00 y 1.99.

Parámetros por defecto	
Frecuencia	675 kHz
Rango	30 metros
Límite izquierdo	0°
Límite derecho	0°
Modo de barrido	Continuo
ADSpan	12dB
ADLow	13dB
Ganancia	40 %
Resolución	*Medium*
Bits por bin	8
Número de bins	300

Tabla 3.1: Configuración por defecto del dispositivo

- *setResolution(Resolution):* Indica la distancia en grados entre dos operaciones de exploración consecutivas. Los valores que pueden pasarse por parámetro son un tipo enumerado definido en la propia clase, que puede tomar los siguientes valores: *Low*$(1, 8°)$, *Medium*(0,90°), *High*(0,45°) y *Ultimate*(0,225°).

- *setBins(int):* Indica el número de *bins* de que ha de constar cada *scanline*. El máximo valor que puede tomar es 800.

- *setDefaultConfig():* Esta función reestablece los parámetros de configuración a su valor inicial. Por defecto, al crear un objeto de la clase *MiniKing*, dichos parámetros se inicializan. En la tabla 3.1 se muestran los valores por defecto de estas variables.

Procesos

Estas funciones implican una secuencia de intercambio de tramas entre la aplicación y el sónar. Abstraen todo detalle de implementación del protocolo al programador.

- *initSonar():* Inicializa el sónar, dejándolo preparado para recibir peticiones de exploración. Es necesario ejecutar este método antes que ningún otro, o de lo contrario no se podrá interactuar con el dispositivo. Internamente, obtiene la versión de software instalada (comando *mtSendVersion*), interroga al cabezal acerca de sus características técnicas (co-

mando *mtSendBBUser*) y envía tramas *mtHeadCommand* de forma iterativa hasta que el dispositivo indica en un *mtAlive* que está preparado para actuar.

- *reboot():* Este procedimiento reinicia el sónar. Envía un comando *mtReboot*, restablece los parámetros por defecto dentro del objeto *MiniKing* y envía comandos *mtHeadCommand* hasta que el dispositivo está listo para recibir peticiones de exploración.

- *updateConfig():* Actualiza en el cabezal los cambios realizados en los parámetros de configuración. Todos los métodos definidos en la sección anterior requieren una llamada posterior a este método. Internamente, envía tramas *mtHeadCommand* hasta que el dispositivo indica que se ha configurado correctamente.

- *getScanLine():* Este método se utiliza para realizar una petición de exploración al dispositivo y obtener una *scanline*. Devuelve un puntero a *BYTE*, un tipo de datos definido dentro de la propia librería, que apunta a la posición de memoria en la que se almacena el valor del primer *bin* de la *scanline*. Se puede acceder al valor de todos los *bins* con el correspondiente incremento de este puntero. Además, este procedimiento es capaz de gestionar las dos tramas *mtHeadData* que devuelve el sónar en respuesta a un comando *mtSendData*. Este último comando sólo se utiliza en caso de que dos *scanlines* hayan sido devueltas al usuario.

- *getDataLength():* Devuelve la longitud en bytes de la última *scanline* obtenida a través del método *getScanLine*.

- *getPosition():* Devuelve la orientación del sónar en el momento de muestrear la última *scanline* obtenida. Tanto este valor como el anterior varían con las sucesivas llamadas a *getScanLine*.

Hasta aquí se han definido los principales métodos necesarios para el funcionamiento de la librería. Existen muchos otros que nos permiten obtener información acerca del dispositivo o configurar algunas características más específicas. Estos procedimientos se han detallado en la documentación de la librería, incluída en el disco compacto que se entrega con esta memoria.

3.5. Ejemplo de utilización

Para ilustrar el funcionamiento y uso de la librería, en este apartado se presenta un ejemplo práctico y sencillo de cómo se pueden utilizar los métodos explicados en el apartado anterior. El programa configura todos los parámetros necesarios para un escenario ficticio, obtiene una *scanline* y muestra el valor de todos sus *bins*.

En primer lugar debemos incluir la librería dentro del fichero fuente:

```
#include "MiniKing.h"
```

en caso de tener las clases en el mismo directorio que el programa. En su defecto, debemos escribir la ruta completa a la librería. El siguiente paso es crear una instancia del objeto *MiniKing*:

```
MiniKing mk("COM1", 0);
```

donde el primer parámetro indica el puerto por el que se quiere establecer la comunicación con el dispositivo y el segundo el *timeout* de la comunicación serie. El valor cero indica que no hay límite de tiempo en la transferencia. Para preparar la comunicación *RS232*:

```
mk.initSonar();
```

que deja al cabezal preparado para recibir peticiones de exploración. En este punto, podemos configurar los parámetros del sónar de acuerdo con nuestras necesidades. En este caso, se configura para que explore de forma sectorial entre los 90° y los 180°, en un rango de 25 metros y con 200 *bins* por *scanline*:

```
mk.setRange(25);
mk.setBins(200);
mk.setContinuous(false);
mk.setLeftLim(90);
mk.setRightLim(180);
```

que se corresponden con las funciones de configuración del cabezal. Estos cambios han sido realizados en la propia instancia de la clase *MiniKing*. Para que surjan efecto en el dispositivo:

```
mk.updateConfig();
```

que deja al sónar configurado correctamente para operar de acuerdo con los parámetros deseados. El último paso es escribir el valor de cada uno de los *bins* de una *scanline*. A través de las siguientes líneas de código:

```
BYTE *data = mk.getScanLine();

printf("POSICIÓN: %.2f\n", mk.getPosition());
for (int i = 0; i < mk.getDataLength(); i++)
    printf("%d ", data[i]);
```

se imprime el ángulo del *scanline* explorado y se escribe el valor de intensidad de cada uno de los *bins*. Es importante notar como se utiliza el método *getDataLength* para saber cuál es el número de *bins* devueltos en la trama *mtHeadData*. Este ejemplo ilustra de una forma muy básica cómo interactuar con el sónar. En el siguiente capítulo se presenta una aplicación mucho más extensa que utiliza la librería programada.

Capítulo 4

Ejemplo de utilización de la librería: una interfaz gráfica de visualización de datos sónar

El objetivo fundamental de este capítulo es presentar una interfaz gráfica de usuario que utiliza la librería desarrollada en este proyecto. En una primera parte, se comentan las características incorporadas en el programa, junto con capturas de pantalla para mostrar el aspecto visual de la aplicación. El capítulo continúa explicando el diseño y la implementación, y concluye presentando resultados reales obtenidos después de realizar pruebas en un escenario real.

4.1. Características generales

Tras presentarse las principales funciones de interacción con la librería diseñada y haber mostrado un pequeño ejemplo para facilitar la comprensión del funcionamiento y uso de la misma, el objetivo que se ha acometido en el marco de este proyecto es implementar una aplicación a gran escala que requiera interacción con el sónar.

El objetivo principal ha sido crear una interfaz de usuario que permita visualizar el entorno

de una forma gráfica, a partir de la información que se obtiene del dispositivo. El programa permite configurar el cabezal en tiempo real, haciendo efectivo cualquier cambio en los parámetros de funcionamiento de una forma transparente al usuario.

La aplicación equivale a un nivel superior de abstracción al representado por la capa tres de la librería, descrita en la sección 3.4. Gracias a esta aplicación, el usuario puede interactuar con el sónar sin conocer el funcionamiento de la librería y, por tanto, sin conocer el funcionamiento del protocolo *SeaNet* bajo el que opera.

Debido a la naturaleza de la librería de interacción, la aplicación se ha desarrollado en *C++*. Por otro lado, para implementar las funcionalidades visuales de la misma se ha optado por utilizar una librería ya existente que proporcione un conjunto de clases que permitan mostrar ventanas, dibujar gráficos, responder a eventos del usuario, etc. Tras barajar varias opciones, se ha elegido *wxWidgets* para esta tarea. La librería *wxWidgets* permite al usuario la creación de interfases gráficas de usuario, proporcionando una gran cantidad de funcionalidades, desde el sencillo diseño de pantallas hasta la gestión de aplicaciones multihilo.

En esta sección se van a mostrar las funcionalidades básicas implementadas en la aplicación. Se ha decidido llamar al programa *MiniKing Viewer*, por razones obvias.

4.1.1. Funcionamiento general

La aplicación se con un selector de puerto serie. Es un menú deslizante en el que se debe indicar el puerto del computador al cual hemos conectado el sónar. En caso de que dicho puerto no exista o sea incorrecto, el programa da un mensaje de error y finaliza su ejecución. Ésta ventana de selección de puerto se puede ver en la figura 4.1.

Tras esto, aparece una ventana de carga (figura 4.2). Durante el tiempo que se muestra dicha ventana, el programa inicia la comunicación con el cabezal, obteniendo toda la información sobre él y enviándole una configuración por defecto, la cual se puede ver en la tabla 3.1. En estos

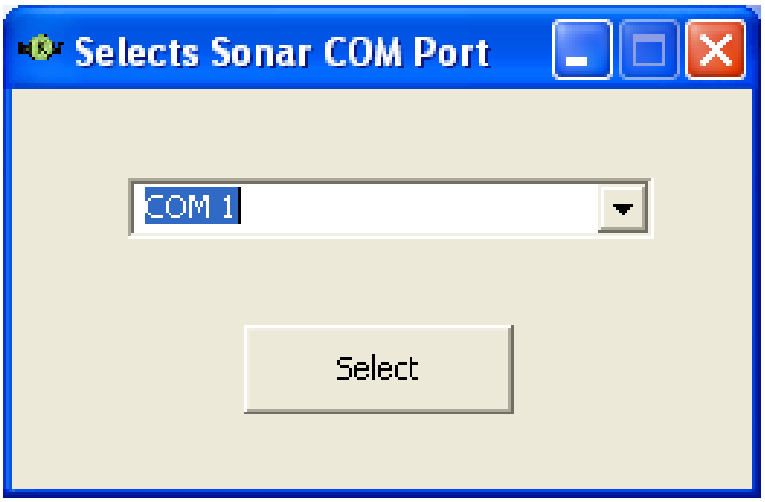

Figura 4.1: Selector de puerto serie

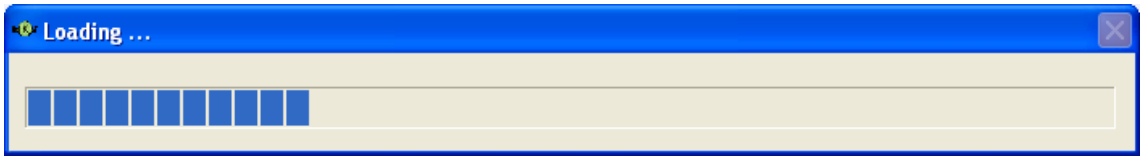

Figura 4.2: Ventana de carga

momentos el sónar está en condiciones de recibir peticiones de exploración.

Con la comunicación activa entre el sónar y la aplicación, el programa muestra su ventana principal (ver figura 4.3). Automáticamente, comienza a realizar peticiones de exploración de forma iterativa y a representar las respuestas del dispositivo por pantalla.

En la parte central de la ventana principal se encuentra el *área de visualización de resultados*. En ella, el programa va mostrando de forma gráfica cada una de las *scanlines* que el sónar devuelve. Sobre dicha área se muestran superpuestos una serie de círculos concéntricos, dónde cada uno representa una distancia concreta en metros, dependiendo del rango configurado en el cabezal. Esta distancia se indica con un número cercano a cada círculo en la posición 0°, aproximadamente. El centro de los círculos concéntricos representa el sónar, y las distancias son relativas, por tanto, al mismo. La aplicación asigna un color a cada *bin* de la *scanline* de

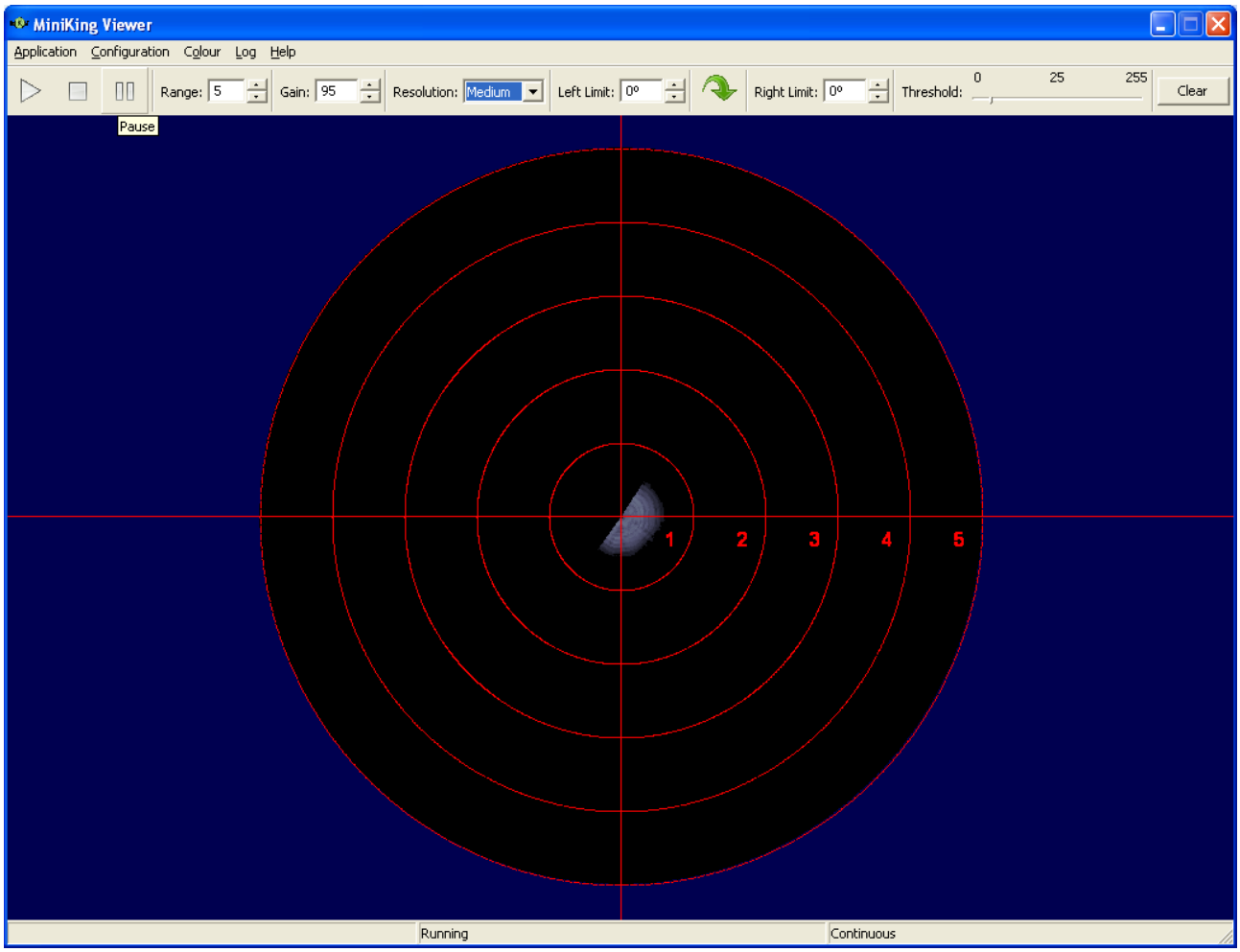

Figura 4.3: Ventana principal del programa

acuerdo con un mapa de color configurable. A mayor valor de *bin*, más intensidad de color. Es importante destacar sobre estas imágenes que se tomaron con el dispositivo operando en aire, y por tanto los resultados no son válidos, ya que el sónar sólo opera correctamente dentro del agua, puesto que ha sido diseñado para operar con velocides de propagación próximas a 1500 m/s.

En la parte superior de la ventana se encuentran la barra de herramientas y los menús. Desde la *barra de herramientas* se pueden configurar los parámetros de funcionamiento del sónar. Cualquier cambio en estos parámetros detiene la exploración, siendo necesario pulsar la opción *play* para que se hagan efectivos dichos cambios en el cabezal y el programa vuelva a realizar

peticiones de exploración de acuerdo con la nueva configuración. La barra de herramientas incluye las siguientes opciones:

- *Play:* Hace efectivo sobre el cabezal cualquier cambio realizado en los parámetros de configuración y reanuda la realización de peticiones de exploración al dispositivo en caso de que estuvieran detenidas.
- *Stop:* Detiene la aplicación y borra el *área de visualización de resultados*.
- *Pause:* Detiene la aplicación *sin* borrar el *área de visualización de resultados*.
- *Range:* Indica la distancia máxima a la que el dispositivo espera encontrar obstáculos. La distancia que representa cada círculo concéntrico en el *área de visualización de resultados* se actualiza de forma automática al modificar este parámetro. El rango puede variar entre 0 y 100 metros.
- *Gain:* Indica la ganancia del dispositivo. Oscila entre el 10 % y el 95 %.
- *Resolution:* En este menú deslizante se puede seleccionar la distancia en grados entre dos operaciones de exploración consecutivas. Los posibles valores son *Low*(1,8°), *Medium*(0,90°), *High*(0,45°) y *Ultimate*(0,225°). Es en definitiva la resolución de paso del motor que hace girar el transductor dentro del cabezal.
- *Left limit:* Indica el límite izquierdo en operaciones de exploración sectorizadas. Viene dado entre 0° y 359°. Se ignora en caso de que el cabezal esté explorando el entorno de forma contínua.
- *Continuous:* Esta opción permite alternar entre el modo de exploración contínuo y el modo de exploración sectorizado. Los límites sectoriales se marcan a través de las opciones *Right limit* y *Left Limit*.
- *Right limit:* Indica el límite derecho en operaciones de exploración sectorizadas. Viene dado entre 0° y 359°. Se ignora en caso de que el cabezal esté explorando el entorno de forma contínua.

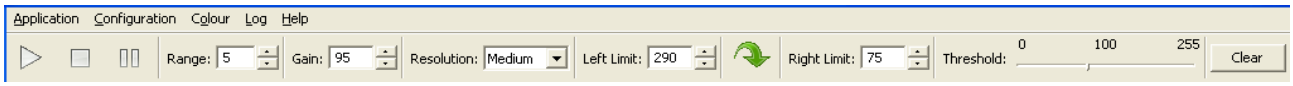

Figura 4.4: Barra de herramientas y menús

- *Threshold:* Este parámetro permite aplicar un valor umbral en la visualización de los resultados, sirviendo como filtro. Todo *bin* que supere el valor configurado en esta opción será mostrado con su color correspondiente acorde al mapa de color seleccionado en la aplicación. En caso contrario, se mostrará en negro.
- *Clear:* Esta opción permite borrar el *área de visualización de resultados* sin necesidad de detener las peticiones de exploración.

Los *menús* de la aplicación nos aportan una gran cantidad de funcionalidades, que se mencionan a continuación clasificadas de acuerdo con su finalidad (ver figura 4.4).

4.1.2. Salir de la aplicación

Para finalizar la aplicación se puede seleccionar la opción *Exit* del menú *Application*. También se puede cerrar el programa haciendo *clic* sobre la casilla marcada con una × en la esquina superior derecha de la ventana principal.

4.1.3. Configuraciones

El programa incluye la opción de almacenar en un fichero los valores de los parámetros configurados en el cabezal para posteriormente poder volver a cargarlos. Esta característica, por ejemplo, evita al usuario reconfigurar el dispositivo en caso de que sea necesario repetir una exploración en un mismo escenario para el cual ya había sido programado.

Una vez se ha configurado el sónar con los parámetros deseados, se puede almacenar dicha configuración seleccionando la opción *Save Configuration* del menú *Configuration*. Aparecerá una ventana para elegir el directorio donde se desea almacenar el fichero. Los archivos de configuración llevan la extensión *.mkv* e incluyen contenido *XML*[1] con el valor de cada parámetro.

[1] *eXtensible Markup Language*

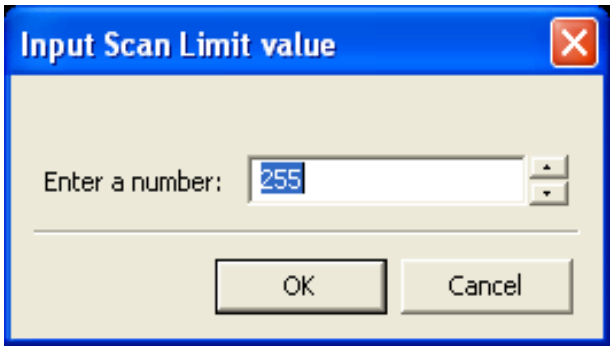

Figura 4.5: Filtro de valor límite

Para cargar una configuración almacenada anteriormente, se debe seleccionar la opción *Load Configuration* del menú *Configuration*. En la ventana de selección, debemos elegir el archivo con extensión *.mkv* que se desea cargar. Una vez se ha configurado el dispositivo con las opciones indicadas en el fichero, el sónar comienza a operar.

4.1.4. Valor límite

Esta opción es un segundo filtro que permite eliminar ruido de los resultados para detectar mejor los contornos. Si seleccionamos la opción *Set limit value* del menú *Configuration* aparecerá la ventana que se muestra en la figura 4.5. Con esta opción activa, el programa representa en color negro todos los *bins* de una *scanline* hasta encontrar el primero que supere el valor indicado, el cual se marca con intensidad de color máxima. El resto de *bins* de la *scanline* se marcan en negro. Para deshabilitar esta opción, se debe elegir la opción *Unset limit value* del menú *Configuration*.

4.1.5. Reinicio del sónar

La aplicación permite reiniciar el sónar en cualquier momento. Para ello, se debe elegir la opción *Reboot* del menú *Configuration*. Una vez que el sónar se ha reiniciado, la aplicación vuelve a enviarle la configuración por defecto (ver tabla 3.1), y los parámetros configurados hasta ese momento se descartan.

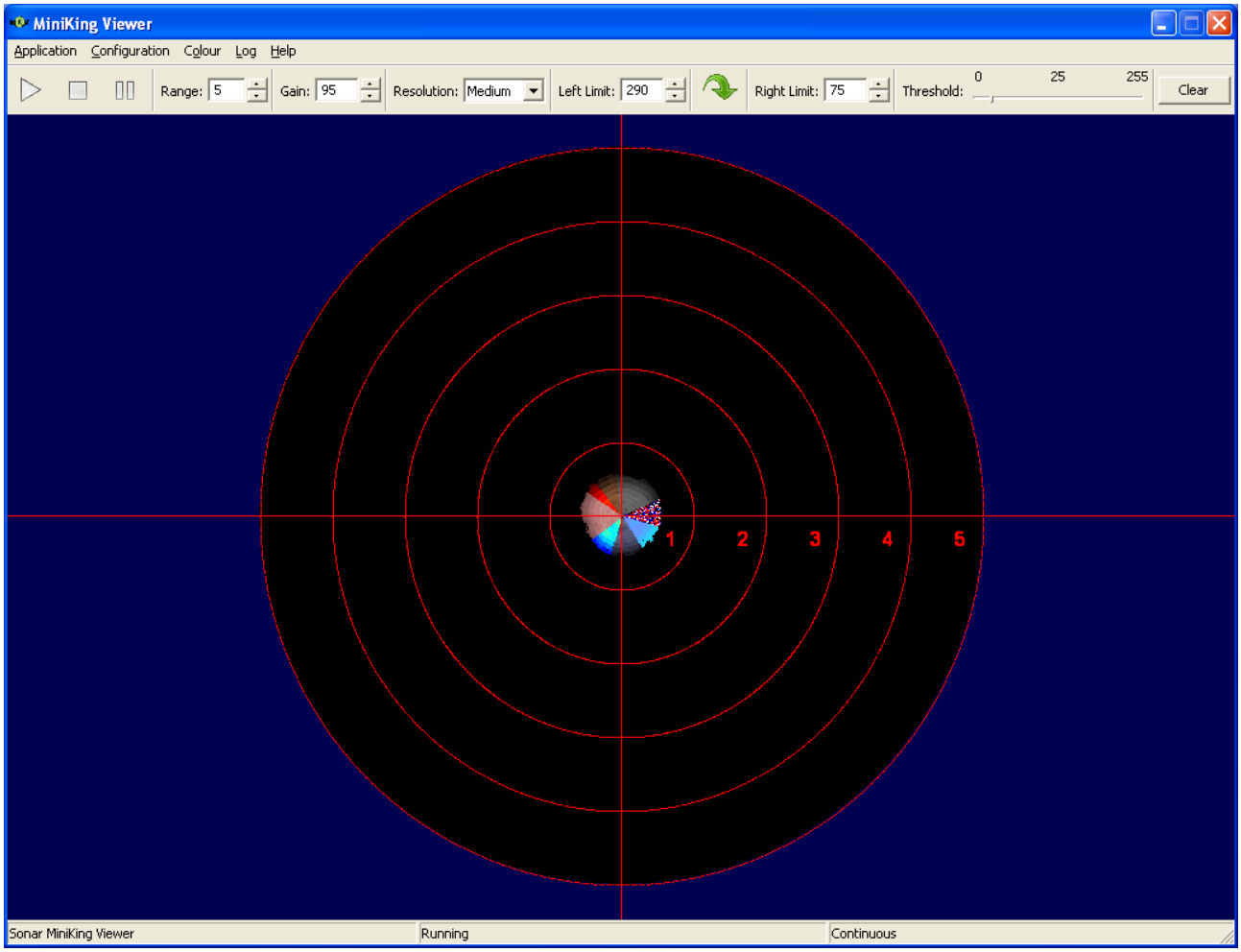

Figura 4.6: Diferentes mapas de color de la aplicación

4.1.6. Mapas de color

El programa permite mostrar las intensidades del eco recibido en diferentes tonalidades, de acuerdo con unos mapas de color predefinidos por la aplicación. Dichos mapas de color se pueden seleccionar en el menú *Colour*. En la figura 4.6 se muestra un ejemplo de algunas de las combinaciones disponibles.

4.1.7. Capturas de pantalla

Para obtener una captura de pantalla del *área de visualización de resultados* debemos seleccionar la opción *Snapshot* del menú *Log*. Una vez se ha seleccionado el directorio destino de la captura, la imagen se almacena en formato *.bmp*.

4.1.8. Grabación de sesiones

La aplicación permite almacenar sesiones de exploración en un fichero para que posteriormente puedan ser reproducidas. Para iniciar la grabación de una sesión debemos seleccionar la opción *Record* del menú *Log*. El siguiente paso es elegir el fichero en el que se va a guardar. Dichos ficheros llevan la extensión *.mlf*. Tras esto, todos los resultados y cambios de configuración que se llevan a cabo quedan registrados en el archivo de sesión. Para detener la grabación, debemos seleccionar la opción *Stop* del menú *Log*.

Un archivo de sesión puede ser visualizado en cualquier momento a través de la opción *Play* del menú *Log*. Tras la reproducción correspondiente, la aplicación se detiene. Es necesario seleccionar la opción *play* para volver a reproducirla. El formato de estos ficheros se trata más adelante en este documento.

4.1.9. Análisis de *scanlines*

En cualquier momento se puede analizar el valor de los *bins* de una *scanline* haciendo doble *clic* sobre el *área de visualización de resultados*. El programa abre una ventana como la que se muestra en la figura 4.7 para la *scanline* sobre la que se encuentra el cursor en el momento de

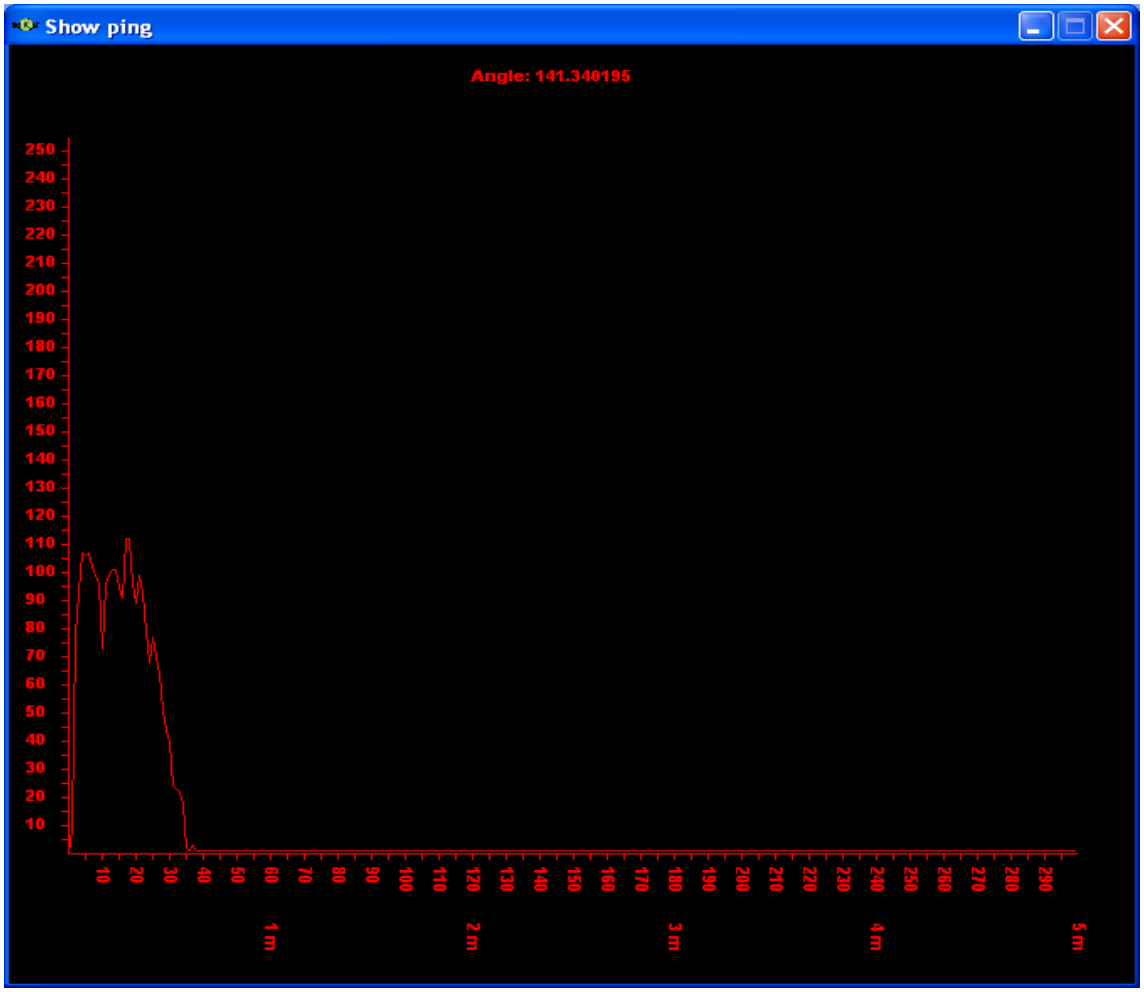

Figura 4.7: Representación gráfica de los valores de una *scanline*

hacer la petición. En el eje horizontal de la gráfica se encuentran todos los *bins* ordenados de la *scanline* junto con la distancia en metros que representa de acuerdo con la configuración del cabezal. En el eje vertical, se muestra el valor recibido por el dispositivo para cada *bin*, que oscila entre 0 y 255. En la parte superior de la ventanta se muestra el ángulo de la *scanline* en grados.

4.1.10. Ayuda

Se puede obtener ayuda sobre el funcionamiento del programa a través de la opción *help* del menú *Help*. Contiene información básica acerca de las opciones que provee la aplicación.

4.1.11. Exploración de sectores

Como ya se ha mencionado, para explorar sectores debemos seleccionar la opción *continuous* de la barra de herramientas para indicarle al dispositivo que no es un análisis contínuo y configurar los límites izquierdo y derecho a través de los correspondientes controles. Si no se configuran estos límites, la exploración no funcionará correctamente. En la figura 4.8 se muestra un análisis sectorizado entre 290° y 75°, pasando por 0°.

4.2. Diseño e implementación

El objetivo fundamental de esta sección es mostrar a grandes rasgos el diseño de la aplicación *MiniKing Viewer*. No pretende ser una descripción exhaustiva del programa, pero sí explicar la estructura básica de clases y comentar detalles de implementación acerca de las principales características que lleva incorporadas.

Toda aplicación basada en la librería *wxWidgets* debe iniciarse a través de una clase derivada de *wxApp*. En este caso dicha clase se ha llamado *MainApp*, e incluye el método *OnInit*, que crea las estructuras necesarias para la ejecución del programa.

La aplicación es capaz de responder a los eventos lanzados por el usuario al mismo tiempo que envía peticiones de exploración al sónar y representa las respuestas en el *área de visualiza-*

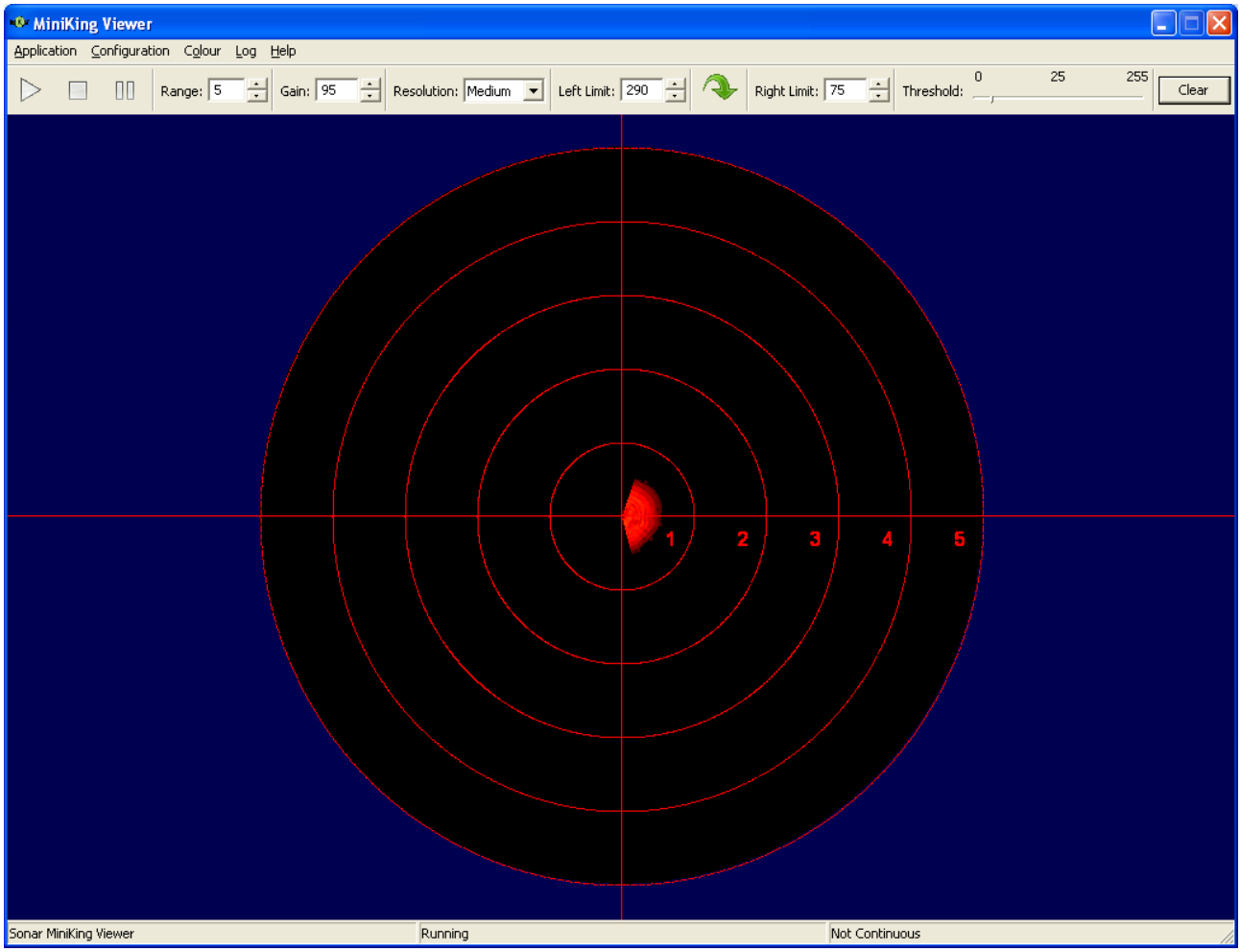

Figura 4.8: Exploración de un sector

ción de resultados. Para obtener esta concurrencia, es necesario desarrollar el programa a través de una estructura multihilo. Así pues, la aplicación se divide en dos hilos básicos de ejecución, cuyas clases son *MainFrame* y *PaintThread* y los cuales son creados desde la clase principal del programa, *MainApp* (ver figura 4.9).

La clase *MainFrame* se encarga de la creación del entorno visual del programa y de la gestión de eventos de usuario. Cada evento que se produce en la aplicación es tratado por un método incluído en esta clase. Este hilo tiene mayor prioridad que *PaintThread*, debido a que se deben tratar los eventos del usuario a la mayor brevedad posible.

La clase *PaintThread* mantiene una comunicación activa con el sónar en todo momento. Para iniciarla se llama al método *Entry*, que entra en un bucle infinito a través del cual se envían peticiones de exploración al sónar y se representan sus respuestas en la ventana de la aplicación de forma ininterrumpida.

A pesar de esta separación de funciones, los dos hilos de ejecución no son totalmente independientes, ya que en ciertas ocasiones uno necesita funcionalidades del otro. Además se han utilizado semáforos de exclusión mútua para evitar asegurar una correcta sincronización entre los dos hilos.

Las dos ventanas iniciales del programa, las correspondientes a la selección del puerto serie y la carga de la aplicación, son dos diálogos implementados bajo las clases *PortDialog* e *InitialDialog*, respectivamente, y no tienen sentido más allá de la configuración inicial del programa.

En el apartado anterior se comentó que cualquier modificación en los parámetros de la barra de herramientas provocaba la detención de la aplicación y era necesario pulsar el botón *play* para llevar a cabo estos cambios sobre el cabezal. A efectos de implementación, la clase *MainFrame* contiene como atributo una instancia de la clase *MiniKing* (ver sección 3.4). Cada vez que el usuario modifica algún parámetro en la barra de herramientas, el evento correspondiente realiza los cambios sobre dicho atributo dentro de la propia clase y detiene el hilo de visualización

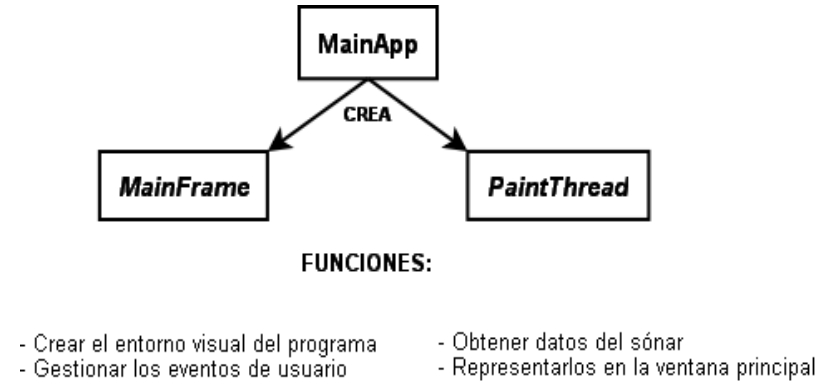

Figura 4.9: Principales clases de la aplicación

mediante la llamada al método *Pause*. El hecho de seleccionar *play* en la barra provoca que se ejecute el método *updateConfig* de la clase *MiniKing*, que hace efectivos los cambios en el sónar, y despierta al hilo de visualización a través de la llamada a la función *Resume*. En los siguientes apartados se comentan las características de implementación más destacadas del programa.

4.2.1. Colores

El hilo de ejecución encargado de mostrar por pantalla las *scanlines* selecciona el color de cada *bin* de acuerdo con un *array* de 256 posiciones definido dentro de la propia clase *PaintThread*. Cada posición almacena el color que toman los *bins* cuya intensidad es igual a la posición que ocupa en dicho *array*. Por ejemplo, un *bin* con una intensidad de valor 200, será mostrado con el color que se encuentra almacenado en la posición 200 del vector de colores.

La característica de cambiar el mapa de color, disponible en la aplicación, únicamente modifica los valores almacenados en este vector de acuerdo con la opción seleccionada por el usuario. Dichos valores se obtienen a partir de ficheros de texto, que guardan las intensidades de rojo, verde y azul (*RGB*) de forma secuencial para cada posición del *array*. Así pues, cuando el usuario elige cambiar el mapa de color a través del menú *Colour*, el procedimiento que responde al evento llama al método *SetColourMap* de la clase *PaintThread*, indicándole por parámetro qué fichero debe leer. Dicho método abre el archivo, lo lee de forma secuencial y almacena los valores en el vector de colores.

El hilo *PaintThread* comprueba para cada *bin* si su valor supera el umbral indicado por el parámetro *threshold*. En caso contrario, el *bin* en cuestión es mostrado en color negro. Por otro lado, al activar el valor límite para mejorar la detección de contornos (ver sección 4.1.4), *PaintThread* gestiona a través de variables *booleanas* la representación de la *scanline*.

4.2.2. Análisis de *scanlines*

Para la opción de visualización de *scanlines* (ver apartado 4.1.9), el programa necesita almacenar la información que le llega del dispositivo para disponer de ella en el mismo momento de recibir la solicitud de análisis por parte del usuario. Ésta no es una tarea trivial, ya que el ángulo de las *scanlines* no es un conjunto discreto y puede variar de una vuelta a otra. Otro inconveniente es la gran cantidad de datos a manejar.

Por tanto, se ha optado por implementar una tabla de *hash*, ordenada en base a la parte entera del ángulo de la *scanline*. Es decir, se ha definido un *array* de 360 posiciones, donde cada posición es una estructura *Vector*, clase que permite la gestión de datos dinámica de una forma sencilla a través de listas. Los datos que se van a almacenar dentro de dichas estructuras *Vector* son registros que representan *scanlines* y que contienen los datos sobre los *bins* procedentes del sónar, el ángulo y el número de vuelta en el que fue tomada la *scanline* en cuestión.

Cada vez que el hilo *PaintThread* recibe una *scanline* del cabezal, ésta es almacenada dentro de su posición de *hash* correspondiente. Por ejemplo, si recibe una *scanline* con ángulo 163.12, guarda la información en la posición 163 del *array* de vectores, la cual está formada por una lista dinámica de *scanlines* cuya parte entera del ángulo es 163. Tras esto, la aplicación elimina las *scanlines* del ángulo precedente que pertenecen a vueltas anteriores a la actual mediante la llamada a la función *deletePings* (en nuestro ejemplo la posición 162 del vector). Esta operación se lleva a cabo mediante una variable global que almacena el número de vueltas dadas por el dispositivo. Cada *scanline* guarda el número de vuelta en la cual fue obtenido, permitiendo así saber si dicho *scanline* pertenece a vueltas anteriores. Este mecanismo nos permite tener en todo momento información actualizada acerca de las *scanlines* obtenidas del sónar sin sobrecargar la

memoria del computador en exceso.

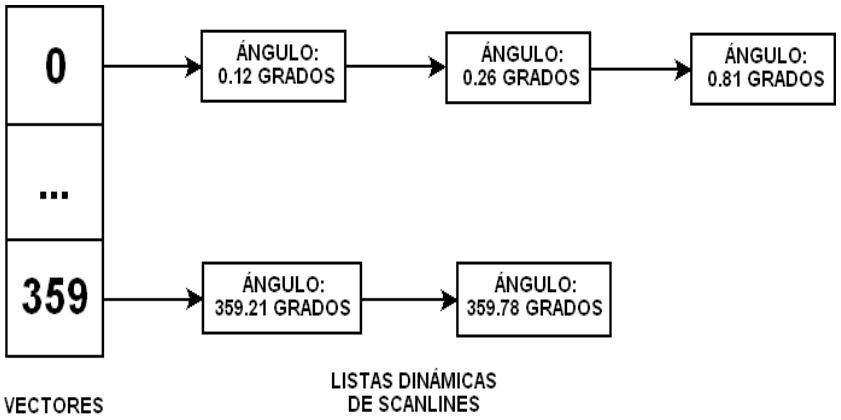

Figura 4.10: Estructuras dinámicas de datos para almacenar *scanlines*

Cuando la aplicación recibe el evento correspondiente, el método encargado de su gestión calcula el ángulo solicitado por el usuario a partir de la posición en la que ha realizado doble *clic*, y llama a la función *showPing*, pasándole por parámetro el ángulo calculado. Dicha función accede a la tabla de *hash* y encuentra la *scanline* más próxima a la demandada por el usuario. El último paso es crear un objeto de la clase *ShowPingDialog*, cuya misión es crear la ventana gráfica a partir de los datos obtenidos.

4.2.3. Configuraciones

Para la gestión de ficheros de configuración se ha optado por utilizar *XML*. Se caracteriza por ser un lenguaje de marcas que permite dar mayor importancia a los datos que al propio formato visual. Gracias a *XML* podemos seguir una estructura jerárquica dentro de un documento, simplificando la tarea posterior de lectura.

Para poder tratar ficheros *XML* dentro de *C++* se ha utilizado una librería llamada *TinyXML*. Como su nombre indica, es un pequeño conjunto de clases que nos permiten crear, modificar y leer documentos *XML*. Las principales ventajas que presenta frente a otras librerías son su sencillez y su pequeño tamaño. Como principal desventaja, nos encontramos la posibilidad de no ser

lo suficientemente completa en algunas situaciones, dependiendo de la envergadura del proyecto. En el caso que nos ocupa, cumple sobradamente su función.

Por tanto, tras haber seleccionado el fichero de configuración en el cual queremos guardar el valor de los parámetros actuales, la aplicación crea un documento *XML* en memoria, obtiene dichos valores y los almacena como nodos en el documento.

Cuando se desea cargar una configuración, la propia librería *TinyXML* nos permite acceder a los nodos del documento y obtener sus atributos, que representan los valores a cargar dentro de los parámetros del dispositivo. Tras haber configurado dichos valores, se actualizan sobre el cabezal mediante la llamada al procedimiento *updateConfig* y el programa continua con su ejecución.

Los valores que se almacenan en los ficheros de configuración son el rango, la ganancia, la resolución, los límites izquierdo y derecho y el umbral de representación. A continuación se muestra un ejemplo de fichero de configuración creado por la aplicación:

```
<?xml version="1.0" ?>
<!--This file is auto-generated by MiniKing Viewer.-->
<configuration name="config.mkv">
    <range value="5" />
    <gain value="95" />
    <resolution value="1" />
    <left value="0" />
    <right value="0" />
    <threshold value="25" />
</configuration>
```

4.2.4. Grabación de sesiones

Para almacenar ficheros de sesión, en un primer momento se pensó en utilizar *XML* como en el caso de las configuraciones. Sin embargo, esta opción no se ha podido llevar a cabo debido a que la estructura de un documento *XML* se crea en memoria y un archivo de este tipo necesita una gran cantidad de espacio de almacenamiento, que termina por desbordar a la memoria disponible para la aplicación. Por tanto, estos *logs* se crean a través de los mecanismos de gestión de ficheros que *C++* provee por defecto, que almacenan los datos en disco de forma directa.

El encargado de almacenar esta información es el hilo *PaintThread*, que lo hace en los casos en que está activa una variable *booleana* definida en su interior. El hilo que gestiona los eventos, *MainFrame*, es quien modifica esta variable a través del método *setSave*, e indica al programa cuál es la ruta del fichero en el que debe almacenar la información mediante la función *setDocument*. En el momento en que el usuario selecciona *Stop* en el menú *Log*, se desactiva la opción de almacenamiento y se cierra el fichero, dando lugar a un archivo de sesión.

Cuando el usuario desea reproducir un archivo de sesión, seleccionando la opción *Play* del menú *Log*, la aplicación detiene el hilo de visualización de datos y llama al método *paintDocument* incluido en *PaintThread*, pasándole por parámetro el fichero a mostrar. Esta función abre el archivo, reproduce su contenido en el *área de visualización de resultados* de forma secuencial y detiene la aplicación, siendo necesario seleccionar la opción *play* para que el programa vuelva a realizar peticiones de exploración.

El formato de las *scanline* en el fichero se compone del ángulo, el valor de los *bins* de la *scanline* y la palabra clave *STOP*. Estos datos se encuentran dispuestos de forma secuencial y separados por espacios. Por ejemplo:

```
182.7 X a n q t t q v w j [ a i d V F N Q L ? > > 7 0 ... t t q a n STOP
183.6 X b l p u t q w w k Y a i f Y H L Q L @ = = 7 0 ... - $ N L Q STOP
```

donde se muestran dos *scanlines* consecutivos, correspondientes a los ángulos 182.7° y 183.6°. Se

observa como en el momento de grabar esta sesión la resolución del programa estaba configurada a *Medium* (0,9°). La aplicación sigue representando *scanlines* hasta que encuentre la palabra clave *ENDOFFILE* en el fichero, que indica el final de la sesión. También puede encontrarse con la palabra clave *RANGE* seguida de un número entero, que le indica que en ese momento se cambió la configuración de dicho parámetro y debe tenerse en cuenta a efectos de visualización.

Cómo último detalle hay que destacar que en ciertas ocasiones el sónar devuelve el carácter *0x1A* como valor de intensidad de un *bin*, el cual también es utilizado como marca de fin de fichero en entornos *Windows*. Si se almacena este valor como intensidad de un *bin*, la aplicación no leerá más allá de él. Por tanto, se ha solucionado el problema almacenando la cadena *$$* en el fichero en lugar del valor del *0x1A*, y se ha tenido en cuenta este hecho en el acceso al vector de colores, representando el *bin* correspondiente mediante el color de la posición 26.

4.3. Resultados

En este último apartado del capítulo se pretende mostrar el funcionamiento de la aplicación dentro de un entorno real. El lugar elegido es un depósito de agua de forma rectangular situado dentro del campus universitario, cerca de las instalaciones deportivas. Las pruebas se han realizado desde una plataforma situada en una esquina de dicho depósito.

En la primera imagen (figura 4.11) se puede apreciar cómo la aplicación detecta la forma rectangular del depósito, e indica que sus medidas son 25 metros de ancho y 40 metros de largo, aproximadamente. En la esquina opuesta a la posición del sónar, se observa gran cantidad de ruido en las *scanlines* mostradas. Este hecho es debido a la escasa profundidad del lugar, lo que nos obliga a modificar la ganancia del dispositivo para obtener mejores resultados.

En la segunda imagen (figura 4.12) se ha rotado el dispositivo 90 grados y se ha reducido la ganancia de la aplicación. La intensidad del ruido se ha reducido bastante respecto a la imagen anterior. Sin embargo, el objetivo del experimento es mostrar el contorno del depósito lo mejor

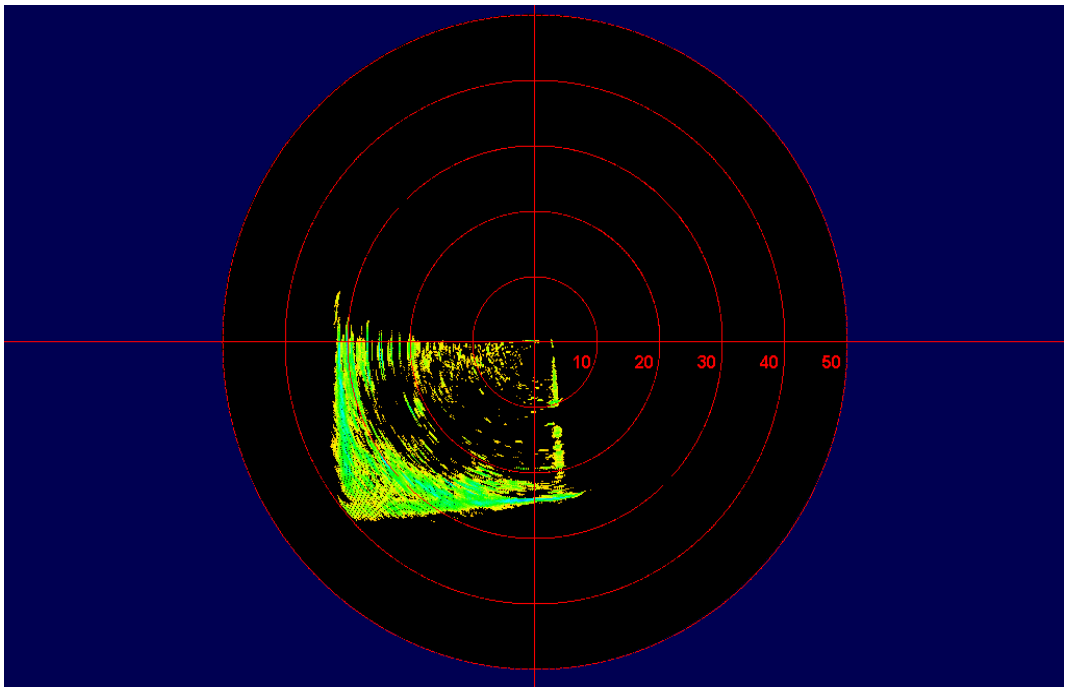

Figura 4.11: Primera imagen del barrido del depósito de agua

delineado posible, por lo que aún debemos realizar algunos cambios en la configuración del cabezal para ello.

En las dos imágenes siguientes (figuras 4.13 y 4.14) se ha ido rotando el sónar y reduciendo la ganancia, tratando de ajustar los parámetros al máximo para obtener el contorno del depósito de la forma más clara posible. Se observa como se reduce la intensidad del ruido en cada caso, aunque sin llegar a obtener explícitamente la forma deseada.

Finalmente, se obtiene una representación mejorada utilizando las opciones de filtro que la propia aplicación proporciona. Combinando de forma correcta el ajuste de la ganancia del dispositivo y el parámetro de valor umbral (*threshold*), se obtiene un contorno más preciso del depósito, como se observa en la figura 4.15.

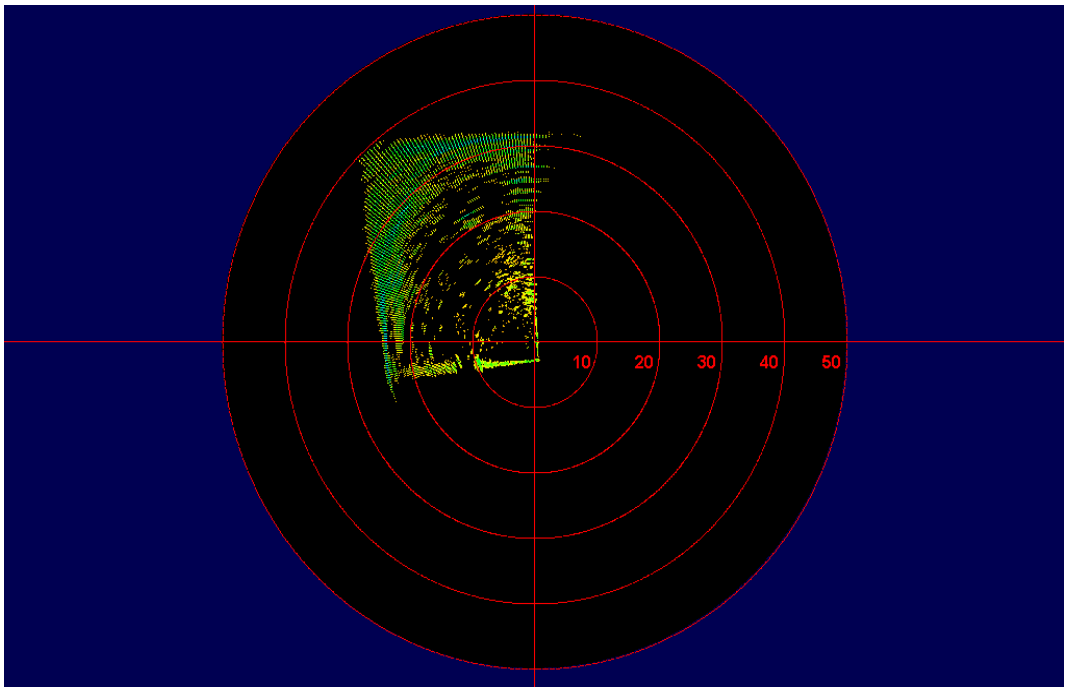

Figura 4.12: Segunda imagen del barrido del depósito de agua

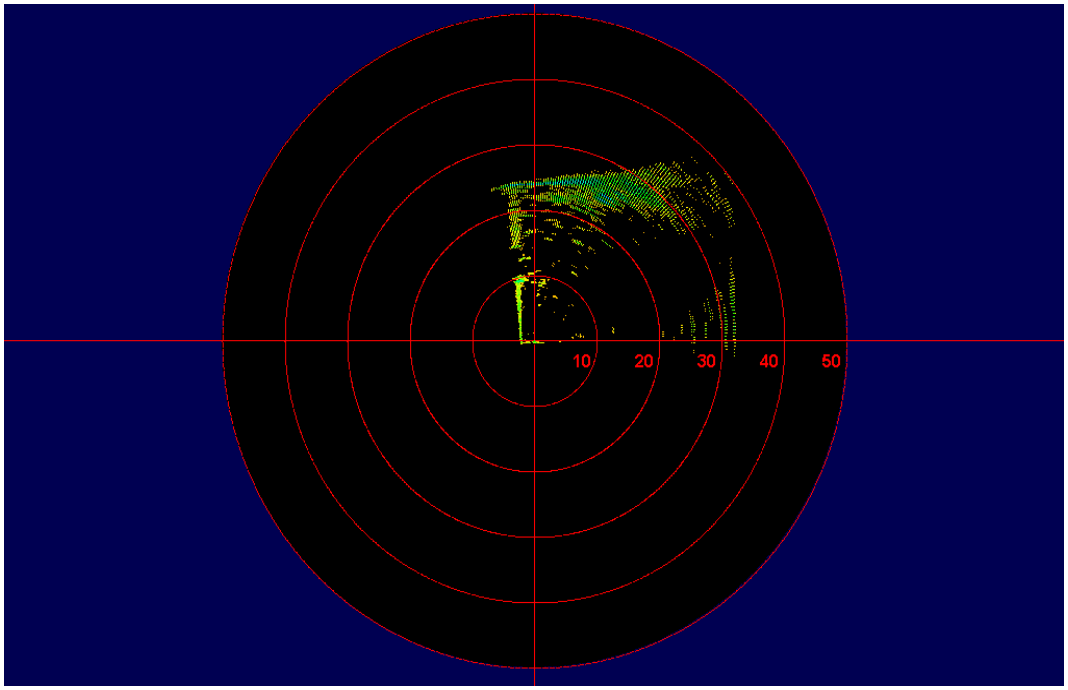

Figura 4.13: Tercera imagen del barrido del depósito de agua

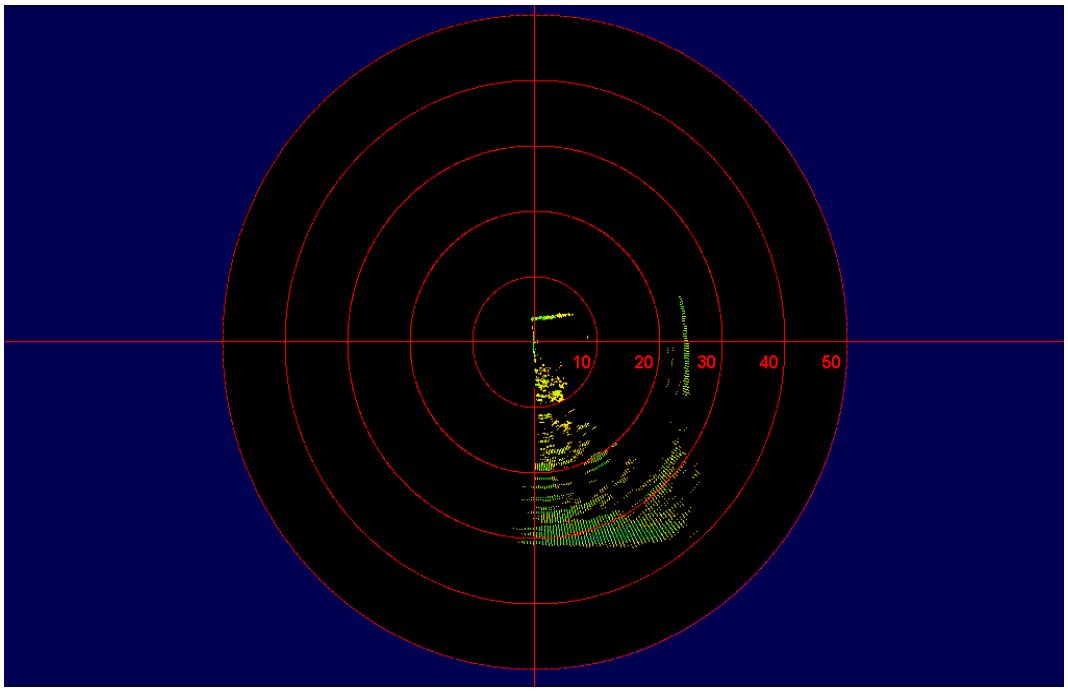

Figura 4.14: Cuarta imagen del barrido del depósito de agua

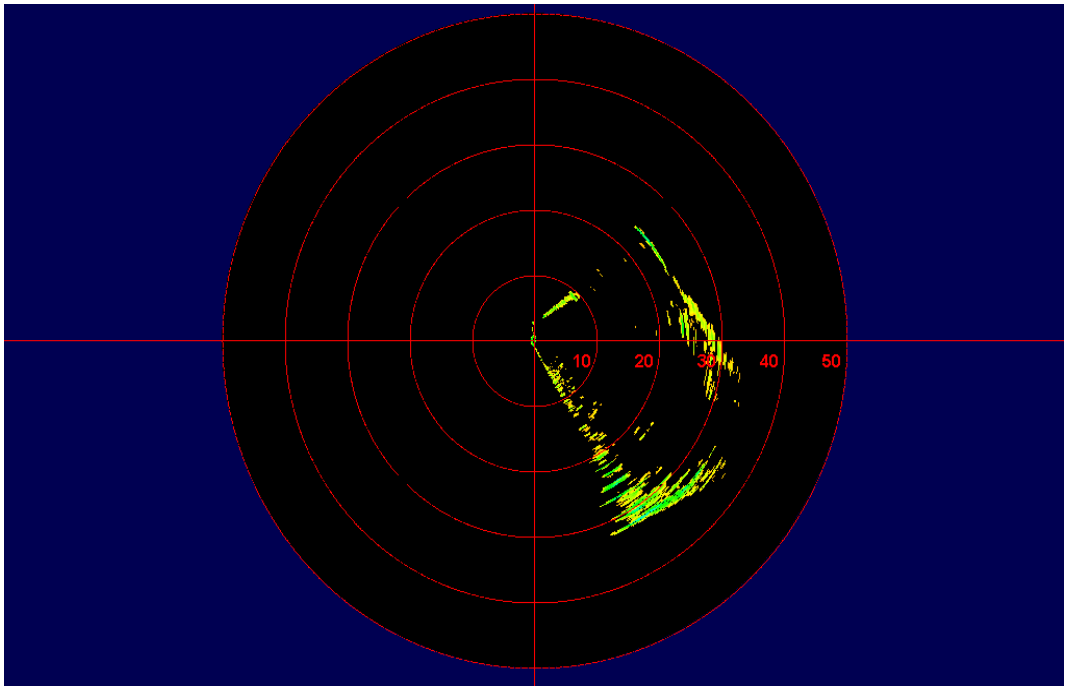

Figura 4.15: Quinta imagen del barrido del depósito de agua

Capítulo 5

Implantación de la librería dentro de un entorno de simulación

En esta sección se presenta la herramienta $NEMO_{CAT}$, un entorno de simulación submarino para el análisis de arquitecturas de control de robots autónomos. Se muestran las modificaciones necesarias para permitir simular el funcionamiento del sónar *Miniking* en dicha herramienta y posibilitar el acceso al mismo a través de la librería desarrollada en este proyecto. El capítulo concluye con ejemplos de simulación en $NEMO_{CAT}$.

5.1. Control de robots móviles

Como bien ya se ha mencionado, el objetivo de esta parte del proyecto es implantar la librería dentro de un entorno de simulación llamado $NEMO_{CAT}$. Esta herramienta permite comprobar el funcionamiento de arquitecturas de control para robots móviles de tipo reactivo e híbrido. Antes de explicar los detalles básicos de este simulador, se va a realizar una introducción teórica al control de vehículos de estas características.

Se conoce como control de un robot al proceso encargado de obtener información del entorno a través del equipamiento sensorial del vehículo, procesarla correctamente de forma que le permita decidir las acciones más adecuadas a llevar a cabo por el robot y ejecutar dichas acciones

a través de los efectores.

En robótica, la palabra *control* tiene diferentes significados. Por un lado, debe asegurar que el movimiento del vehículo es estable y que se produce de acuerdo con ciertos criterios de control, como por ejemplo mantener el robot de forma horizontal a pesar de las irregularidades del terreno o comprobar que la orientación es la deseada. Esto se conoce como *control de bajo nivel.*

Por otro lado, el control de un robot debe asegurar la capacidad de realizar ciertas tareas, como llegar a un punto objetivo, navegar en un entorno, evitar obstáculos o construir un mapa. Este tipo de control se denomina *control de alto nivel.*

Es necesario que una arquitectura de control tenga en cuenta estos dos puntos de vista, ya que es importante tanto llevar a cabo las tareas de alto nivel como dirigir la trayectoria del vehículo de forma correcta. En el diseño de aviones, por ejemplo, se suele distinguir entre:

- Control *de* la trayectoria (alto nivel): Mantener la trayectoria deseada hacia el objetivo a través de GPS, sistemas de visión, mapas, etc.
- Control *sobre* la trayectoria (bajo nivel): Mientras el vehículo navega hacia su destino, se espera que mantenga su orientación estable, sin oscilar alrededor de ninguno de sus ejes.

Por tanto, el control de un robot debe abarcar el rango completo de niveles, desde los aspectos de más bajo nivel, como mantener un movimiento estable, hasta los aspectos de más alto nivel, como la planificación, la toma de decisiones y el razonamiento requerido para realizar cierta tarea. Para ello, es importante una buena organización del software que se está ejecutando en el sistema robótico, en lo que se conoce como *arquitectura de control de un robot.*

5.2. Paradigmas de control de un robot

Existen diferentes configuraciones de control aplicables a robots móviles. No obstante, no existe ningún consenso acerca de la arquitectura de control idónea. Se debe estudiar cada caso

en particular y analizar las ventajas e inconvenientes de cada configuración para determinar qué arquitectura se adecúa mejor a una situación concreta.

Actualmente, se reconocen tres modelos:

- Arquitecturas **deliberativas**.
- Arquitecturas **reactivas**.
- Arquitecturas **híbridas** o deliberativo-reactivas.

La diferencia fundamental entre las tres arquitecturas radica en la relación existente entre las tres primitivas en las que comúnmente se acepta que se descompone cualquier acción robótica de alto nivel: *sense*, *plan*, *act*. Difieren también en la forma en la que la información sensorial es procesada y distribuida a lo largo del sistema.

La primitiva *sense* tiene como misión obtener información del entorno a partir de los sensores y crear un mapa a partir de ella. En la fase de *plan* se genera un plan de ejecución acorde a la misión a cumplir y el modelo del entorno realizado anteriormente. La etapa *act* traduce el plan de ejecución en una secuencia de comandos apropiada para los actuadores.

5.2.1. Control deliberativo

Fue el primer modelo de control de un robot. Estos primeros intentos están fuertemente influenciados por el optimismo de la comunidad investigadora en la potencia del razonamiento simbólico y la Inteligencia Artificial. Se fundamenta en realizar ciclos completos *SPA* (*sense*, *plan*, *act*) de forma iterativa, como se puede apreciar en la figura 5.1. En este modelo la información fluye en un único sentido, por lo que no incluye realimentación. Los movimientos del robot se definen a base de un detallado proceso de planificación mediante técnicas de Inteligencia Artificial.

La parte más destacada de este paradigma es el *planificador*. Es una entidad que razona sobre los símbolos en los que está basado el modelo que representa el entorno. Por ello, las tareas más

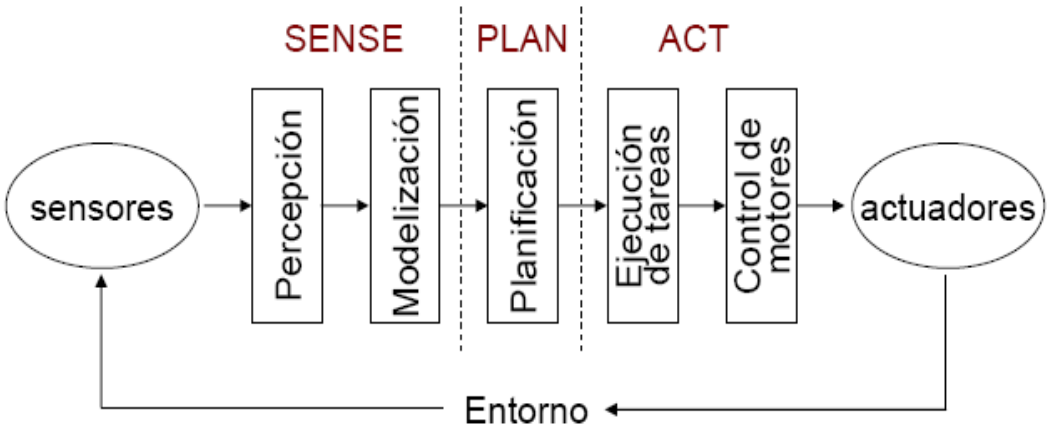

Figura 5.1: Esquema del modelo de control deliberativo

importantes a realizar son la modelización y la planificación, que a su vez implican un elevado coste temporal.

Existen dos variantes de este tipo de arquitectura:

- *Control jerárquico:* Jerarquía de niveles que emplean/proporcionan servicios de/a niveles superiores/inferiores.

- *Control centralizado:* Conjunto de módulos comunicados por un almacén central de información.

A finales de los años 80 se identificaron una serie de inconvenientes intrínsecos asociados a estas arquitecturas. Entre dichos inconvenientes podemos destacar la dificultad de mantener sincronizada la representación del entorno con el mundo real. Este hecho es debido a que existen ciertos eventos que pueden ocurrir con una frecuencia superior a la frecuencia de ejecución del ciclo *SPA*. Otro inconveniente destacable es el coste de las tareas de modelización y planificación, como ya se ha comentado anteriormente. En resumen, este paradigma no se adapta a posibles cambios en el entorno y no es válido por tanto dentro de aquellos modelos con carácter dinámico.

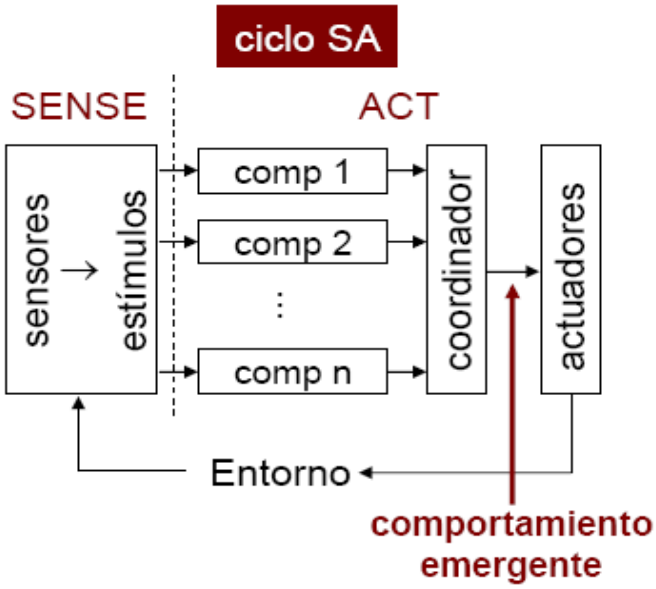

Figura 5.2: Esquema del modelo de control reactivo

5.2.2. Control reactivo

Este segundo paradigma surge debido al desencanto que se produce con las arquitecturas deliberativas en cuanto a capacidad de tratar con la complejidad del mundo y la imposibilidad de actuar en tiempo real. La idea clave de esta configuración es que *el mejor modelo del mundo es el mundo*, ya que siempre está actualizado y contiene toda la información necesaria para tomar decisiones de carácter inmediato.

Por tanto, los esfuerzos van dedicados al subsistema sensorial y no a la planificación de las tareas. Debemos obtener toda la información del mundo de la forma más apropiada para un mejor proceso y observar dicho entorno frecuentemente, ya que puede cambiar en cortos períodos de tiempo. Se debe poner gran énfasis en la calidad de la información capturada por los sensores.

Este tipo de arquitectura de control ejecuta ciclos *SA* de forma iterativa, eliminando la pesada parte de planificación, y toma decisiones de forma *reactiva* de acuerdo con los estímulos recibidos por los sensores en un momento determinado. En este punto radica la importancia de

las percepciones sensoriales.

La arquitectura se estructura como una colección de niveles que se ejecutan de forma paralela. A estos niveles se les denomina *comportamientos*. Dichos comportamientos se ejecutan de forma independiente unos de otros sin supervisión (planificación) y sin una representación global del entorno, y generan su salida a partir de la información obtenida de los sensores. Cada una de estas capas es responsable de un único comportamiento. Ejemplos de dichos comportamientos son ir a un objetivo, esquivar obstáculos, mantener la distancia al fondo, mantener la profundidad, etc.

Las decisiones del robot dependen mucho de la relación que existe entre los diferentes comportamientos que ejecuta su arquitectura. En este sentido, entra en juego el *coordinador*, que es la capa del sistema encargada de decidir los movimientos del vehículo a partir del valor de los comportamientos en un momento concreto. Existen dos posibles mecanismos de coordinación de comportamientos:

- *Competitivo:* El coordinador decide en cada momento qué comportamiento es más adecuado para cada situación concreta.
- *Cooperativo:* El comportamiento emergente es una combinación ponderada de los comportamientos individuales.

Como ejemplo al primer tipo de coordinación, se puede señalar los *sistemas subsumidos*, propuestos por Rodney Brooks en 1986. La idea es asignar una prioridad a cada comportamiento, de forma que las respuestas de los niveles superiores predominen sobre las respuestas de los niveles más bajos. Los niveles más altos deben estar orientados a garantizar la supervivencia del vehículo, como por ejemplo evitar obstáculos. El comportamiento resultante en un momento concreto es aquél procedente del comportamiento activo con mayor prioridad.

Como ejemplo de coordinación cooperativa se pueden mencionar los *motor schemas*, definidos por Ronald Arkin en 1989. Un *motor schema* representa una unidad básica de comportamiento.

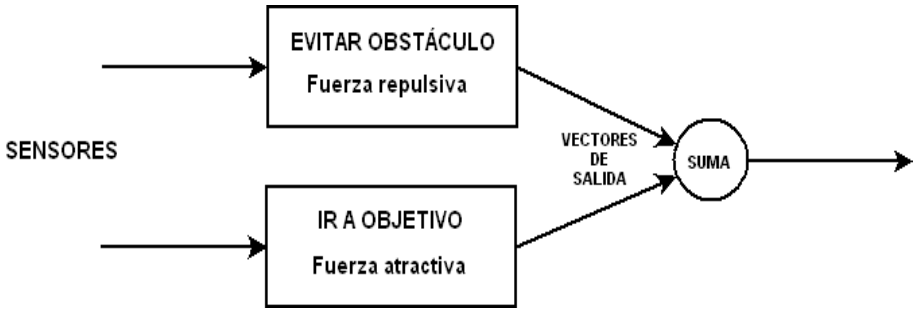

Figura 5.3: Generación de un vector de salida a través de campos de potencial

Cada una de estas unidades opera de forma asíncrona y concurrentemente con los otros y reacciona proporcionalmente a la información sensorial percibida. La salida de cada *motor schema* es un vector 2D/3D que define la dirección de movimiento que debería seguir el robot. Cada uno de los comportamientos tiene asociado una ganancia que determina su contribución relativa a la dirección del movimiento del robot. El vector resultante de este tipo de coordinación es la suma vectorial ponderada y normalizada de los vectores que provienen de los comportamientos de que consta la arquitectura del sistema robótico. Los vectores se crean a partir de un campo de potencial, gradiente de la fuerza que regula el movimiento del robot. En la figura 5.3 se puede ver un ejemplo de generación de vectores a partir de campos de potencial.

Esta arquitectura resuelve los problemas presentados por el control deliberativo, consiguiendo resolver tareas relativamente complejas con éxito rápidamente y con poca información sensorial. Además reduce la carga de las comunicaciones, ya que cada capa sólo procesa la información que le es relevante.

En cuanto a sus limitaciones, destaca la dificultad en encontrar un método sistemático de descomposición de un problema en comportamientos. La ordenación de las prioridades de las tareas, en el caso de coordinación competitiva, y la valoración de las ganancias en el caso cooperativo no son fáciles de definir. Por último, la ausencia de una visión global del entorno puede limitar su aplicación, debido a la falta de memoria. A priori, no es trivial, por tanto, implementar tareas complejas mediante arquitecturas puramente reactivas.

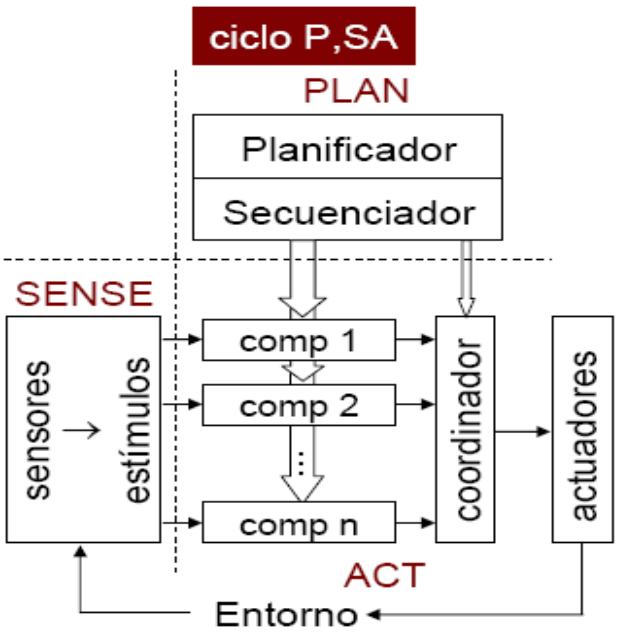

Figura 5.4: Esquema del modelo de control híbrido

5.2.3. Control híbrido

Este último modelo de control pretende aunar las ventajas de los dos planteamientos anteriores para conseguir robots robustos, inteligentes y flexibles. Del primer modelo toma la planificación dentro de entornos controlados, mientras que del segundo la reactividad y la simplicidad, los cuales facilitan el movimiento en entornos dinámicos.

Estos sistemas híbridos incorporan razonamiento deliberativo y ejecución basada en comportamientos. Existe un procesamiento a diferentes niveles y con distinta frecuencia de ejecución de los módulos que componen el sistema. Los módulos *reactivos* tratan situaciones impredecibles y aportan al sistema respuestas en tiempo real. Los módulos *deliberativos* generan y mantienen un modelo del entorno que les permite planificar el movimiento.

Esta arquitectura ha sido propuesta por varios grupos de investigación de forma simultánea. No obstante, a nivel de detalle aparecen diferentes estructuras debido a que no es evidente cómo unir los niveles reactivos y deliberativos. La configuración más típica consta normalmente de tres

niveles:

- *Planificador:* Se encarga de realizar un modelo del entorno y un plan de ejecución a partir de él.
- *Secuenciador:* Descompone el plan de ejecución creado anteriormente en tareas.
- *Reactivo:* Implementa las tareas y las ejecuta de forma coordinada, proporcionando una respuesta rápida del robot.

Los sistemas híbridos proporcionan una vía para la incorporación de métodos de inteligencia artificial y representación simbólica en arquitecturas reactivas. Hay fuertes evidencias de que en el mundo animal se reproducen esquemas híbridos deliberativo-reactivos.

Entre los principales inconvenientes, se puede mencionar la falta de consenso en el número de niveles a implementar y en la interfase entre los niveles reactivos y deliberativos.

5.3. Herramienta de simulación: $NEMO_{CAT}$

En el campo de la robótica, existen una serie de ventajas derivadas del uso de herramientas de simulación. La pérdida de detalles debido al uso de simuladores se compensa con ahorro en esfuerzo, riesgos y dinero, necesarios para llevar a cabo ciertos experimentos. Sin embargo, estas técnicas de simulación no sustituyen por completo a la experimentación, sino que la complementan.

A continuación se presenta un simulador 3D denominado $NEMO_{CAT}$ (*Navigational Environment MOdeller, Control Architecture Tester*), desarrollado en la Universidad de las Islas Baleares con vistas a permitir la validación y puesta a punto de arquitecturas de control reactivas e híbridas para un AUV (ver sección 1.2). Permite definir la estructura sensorial del vehículo, además de los comportamientos que obtienen información de dichos sensores. La figura 5.5 muestra una vista general de $NEMO_{CAT}$.

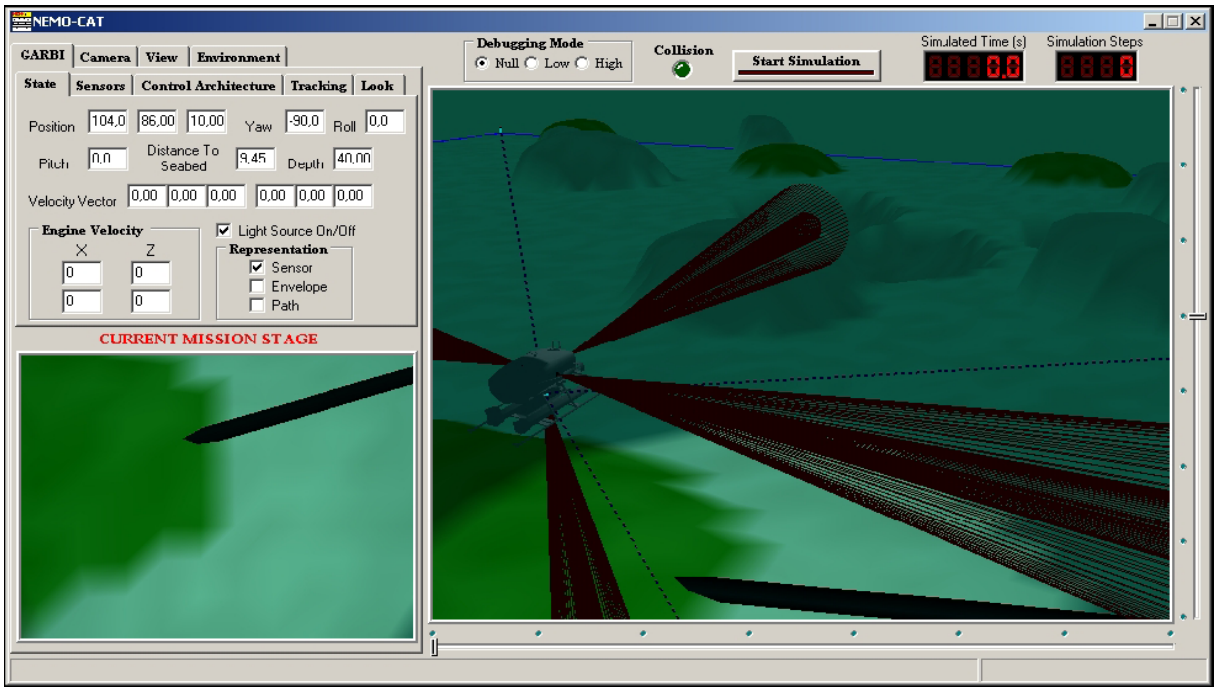

Figura 5.5: Vista general del simulador

Para el diseño de esta herramienta se ha requerido del uso de una metodología de desarrollo de *software*, concretamente *RUP* (*Rational Unified Process*), que suele ser utilizada junto a *UML*. Este último es un lenguaje de construcción de modelos de propósito general que se puede utilizar para especificar, visualizar, construir y documentar las características que componen un sistema de software complejo como $NEMO_{CAT}$. En cuanto a la implementación del simulador, se ha utilizado el lenguaje *C++* para el desarrollo general y la librería gráfica *OpenGL* para mostrar las imágenes del simulador.

Los modelos de *AUV* disponibles en la aplicación, denominados *GARBI* y *URIS*, están basados en dos vehículos diseñados y construidos por el grupo de investigación de visión por computador y robótica de la Universidad de Girona.

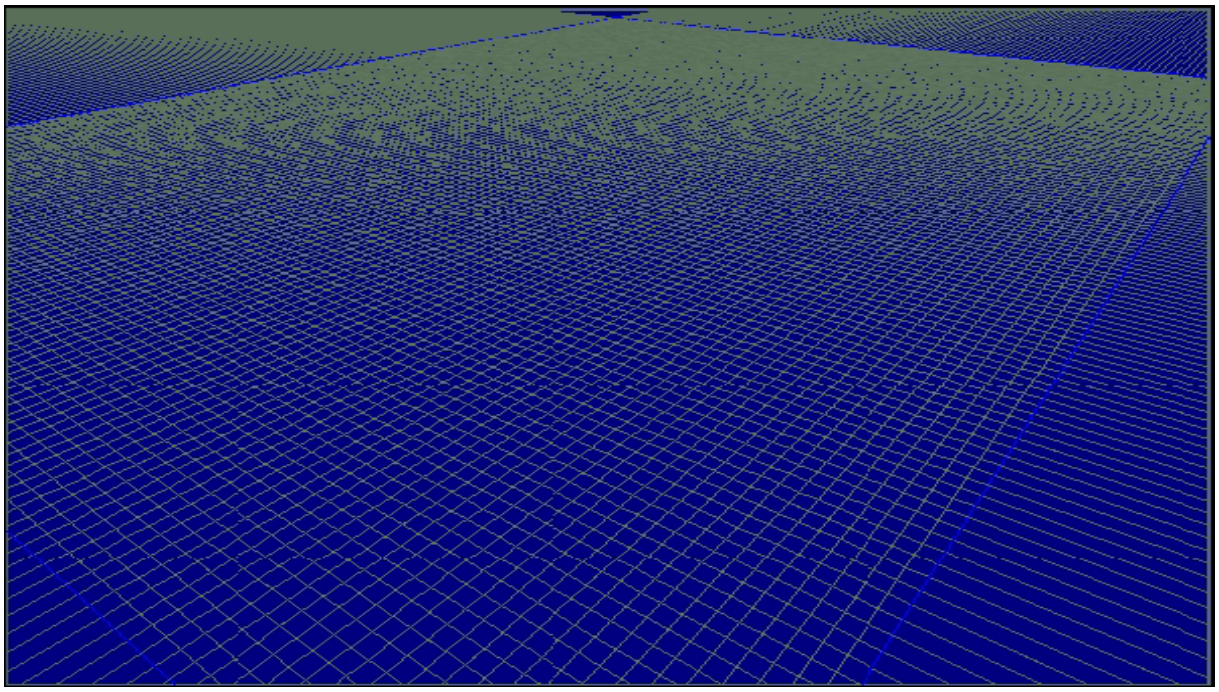

Figura 5.6: Matriz del entorno submarino

5.3.1. El entorno submarino

El entorno submarino se modela a partir de una matriz de puntos cuya extensión y resolución pueden ser configurados manualmente para cada misión concreta, como se observa en la figura 5.6. Inicialmente todos los puntos se encuentran dispuestos a la misma altura. La aplicación permite definir rocas, fosas y algas como obstáculos, que se pueden añadir en cualquier posición del fondo marino, modificando las coordenadas verticales de dicha matriz. La detección de los obstáculos se realiza a partir del cálculo de las intersecciones de los haces de los sensores simulados con los objetos que forman el entorno. Existen otra serie de objetos que pueden situarse sobre la superficie marina, como cables y tuberías.

5.3.2. Vehículos autónomos submarinos

Los vehículos autónomos submarinos (AUV) han sido incluídos en la aplicación a través de los parámetros hidrodinámicos del modelo que representan, su apariencia visual y su equipamiento sensorial. No existe límite para el número de $AUVs$ simulados de forma simultánea.

La dinámica de los vehículos se ha establecido en base a una serie de ecuaciones no lineales que tienen en cuenta los seis grados de libertad de los *AUVs*. Los parámetros de los dos modelos representados hasta el momento han sido estimados a partir de un conjunto de pruebas reales en diferentes entornos.

En cuanto a la configuración sensorial disponible, la aplicación dispone de tres tipos de sensores que pueden usarse en un *AUV*, con el objetivo de adecuarlo a la misión que debe llevar a cabo:

- *Sónars:* Utilizados para la detección de obstáculos dentro del simulador. El haz que generan posee forma cónica. La resolución y la máxima distancia a la que detectan obstáculos son parámetros configurables al crear un sensor de este tipo.
- *Brújulas:* Nos permiten obtener la orientación del vehículo respecto a un sistema de coordenadas base.
- *Cámaras:* Se utilizan para visualizar el entorno, ya sea desde la posición del vehículo o desde una posición global. Se pueden implementar algoritmos de visión por computador a partir de las imágenes obtenidas por estos sensores.

El simulador incorpora además un sistema de posicionamiento de tipo *Long Base Line* (*LBL*), que determina la posición del vehículo de acuerdo con un conjunto de sensores, por lo menos tres, situados en el fondo marino.

El objetivo de esta parte del proyecto es la definición un sónar del modelo *Miniking* dentro del equipamiento sensorial de un *AUV* simulado, y obtener información del entorno de simulación a partir de la librería presentada en el capítulo 3.

5.3.3. Comportamientos disponibles

Con el objetivo de facilitar al máximo la tarea del programador, la aplicación incluye una completa librería de clases que implementan una serie de comportamientos sencillos. De todos ellos, podemos destacar los siguientes:

- *Evitar obstáculos:* Permite al vehículo evitar barreras de navegación como rocas, algas o incluso otros posibles vehículos. La respuesta de este comportamiento es un vector en dirección opuesta al obstáculo, cuya magnitud es variable de acuerdo con la distancia que hay entre el AUV y el objeto. Como se verá más adelante, uno de los subobjetivos de esta parte del proyecto es modificar este comportamiento, configurándolo para que obtenga datos directamente del sónar a través de la librería. Este nivel está representado en la aplicación mediante la clase *TAvoidObstacles*.

- *Evitar el pasado:* Es un intento de superar los problemas de atrapamiento en pozos de pontencial que derivan del uso de arquitecturas reactivas. Para este propósito, se utiliza un mapa local de la ruta recorrida recientemente por el AUV. Cuando el vehículo se encuentra durante un largo período de tiempo en un área concreta, el comportamiento se activa generando un vector que favorece la exploración de nuevas rutas en el entorno. En este caso, la magnitud del vector es proporcional al tamaño del área en el que se encuentra atrapado. Se encuentra representado dentro del programa mediante la clase *TAvoidPast*.

- *Ir a un punto:* Este comportamiento dirige el vehículo a un cierto punto tridimensional definido por el usuario generando un vector, constante en magnitud, cuya dirección une la posición actual del AUV con dicho punto. Está implementado de una forma algo más compleja, ya que permite incluir una lista de puntos a los que acceder acorde a el orden en que han sido definidos. Cuando el robot se halla en una posición lo suficientemente cercana al punto en cuestión, el objetivo pasa a ser el siguiente que se encuentra en la lista. Este comportamiento se ejecuta en la aplicación bajo el nombre de *TGoToGoal*.

- *Permanecer en una región:* Se utiliza para prevenir la exploración de entornos exteriores al área deseada. Se activa únicamente en aquellos casos en los que el vehículo se encuentra lo suficientemente cerca de los límites deseados. En este caso, se genera un vector que aleja el AUV todo lo posible y cuya magnitud es directamente proporcional a la distancia a la que se encuentra del límite. Está implementado dentro de la clase *TStayOnRegion*.

- *Detección y seguimiento de un cable submarino:* Mueve estratégicamente el vehículo a lo largo de todo el área de exploración en busca de un cable. Una vez el vehículo se ha estabi-

lizado verticalmente, inicia un movimiento en *zigzag* hasta encontrar evidencias suficientes de la existencia de cables submarinos. Tras haberlo detectado, se inician dos procesos que se ejecutan de forma secuencial: el primero trata de mantener el cable orientado verticalmente en las imágenes obtenidas a través de la cámara de a bordo mientras que el segundo lo mantiene en el centro de la imagen. En condiciones normales, el cable puede desaparecer debido a problemas en el entorno. En esta situación, el comportamiento posee un mecanismo de recuperación consistente en volver a realizar los movimientos en *zigzag* pero dentro de un área más reducida a partir de la ruta recorrida recientemente por el vehículo. Este comportamiento está representado dentro del simulador mediante la clase *TTrackCable*.

- *Mantener la distancia al fondo marino:* Permite mantener la distancia al fondo marino constante con el objetivo de mantener constante la longitud del cable en las imágenes capturadas por la cámara. La clase incluida en la aplicación se llama *TKeepDistanceToSeabed*.
- *Vuelta al principio:* Este comportamiento permite al vehículo regresar al principio de la misión. Una vez en ese punto, eleva el AUV hasta alcanzar la superficie. El comportamiento está representado por la clase *THoming*.

5.4. Adaptación de la librería

En esta sección se presentan todas las modificaciones que se han realizado para poder utilizar la librería dentro del simulador $NEMO_{CAT}$. Este conjunto de cambios son necesarios para que, una vez comprobado el correcto comportamiento del vehículo en el simulador, se pueda transportar con los mínimos riesgos a un entorno real sin realizar variaciones importantes en el código fuente.

Por tanto, con esta última parte del proyecto se pretende demostrar la utilidad de la librería en el control de la navegación de un robot submarino, concretamente en su comportamiento de evitación de obstáculos. Nos permitirá obtener información del entorno a partir de un sónar *Miniking* simulado y modificar la trayectoria del vehículo de forma consecuente.

La librería se comunica con el dispositivo real a través de un puerto serie *RS232*. En la situación dada, al tratarse de un entorno virtual, no se dispone del sónar, y por tanto esta comunicación debe simularse. Para ello se ha implementado una clase llamada *TSerialPort*, que se encarga de esta tarea. Dicha clase proporciona una serie de primitivas[1] las cuales permiten leer y escribir tramas del protocolo *SeaNet*, de forma que la librería y el cabezal virtual puedan intercambiar información a pesar de la ausencia de un puerto serie. Las primitivas de escritura reservan memoria de forma dinámica de acuerdo con la longitud del paquete en cuestión.

Las funciones y procedimientos definidos en la librería envían y reciben datos del cabezal. En un entorno simulado no disponemos del dispositivo, por ello, al igual que en el caso del puerto *RS232*, se debe implementar una clase que imite su comportamiento, y que envíe la información necesaria a un objeto *TSerialPort* común. Dicha clase se ha llamado *TMiniking*, e implementa una serie de métodos que envían a la librería las tramas de información, conocidas como *Mensajes* (ver sección 3.3), necesarias para la simulación de protocolo *SeaNet*.

La clase *TMiniking* deriva de uno de los sensores predefinidos en la aplicación, *TEchoSounder* (ver figura 5.7). Este último implementa la funcionalidad básica de un sónar común, y proporciona un procedimiento llamado *GetDstObstaculo*, que devuelve la distancia a la que se encuentra el obstáculo más cercano, si es que lo hay, en la dirección actual en la que apunta el sensor en el momento de obtener la medida. A partir de esta función se puede crear la trama *MtHeadData* correspondiente del protocolo *SeaNet*, que será enviada a través del objeto *TSerialPort* al vehículo. Con la distancia devuelta por la función *GetDstObstaculo*, se calcula el número de *bin* que representa, se le marca con el máximo valor de intensidad de señal y se crea el paquete de respuesta aplicando una distribución *Gaussiana* con media en dicho número de *bin*. El número de *bin* (k) que representa una distancia concreta se puede calcular a partir de la fórmula:

$$k = \frac{2 \times e_k}{v \times adinterval \times 640\ ns}, \tag{5.1}$$

donde e_k representa la distancia detectada por *GetDstObstaculo*, v la velocidad del sonido en el

[1] *read(), readNBytes(), write(), writeNBytes()*

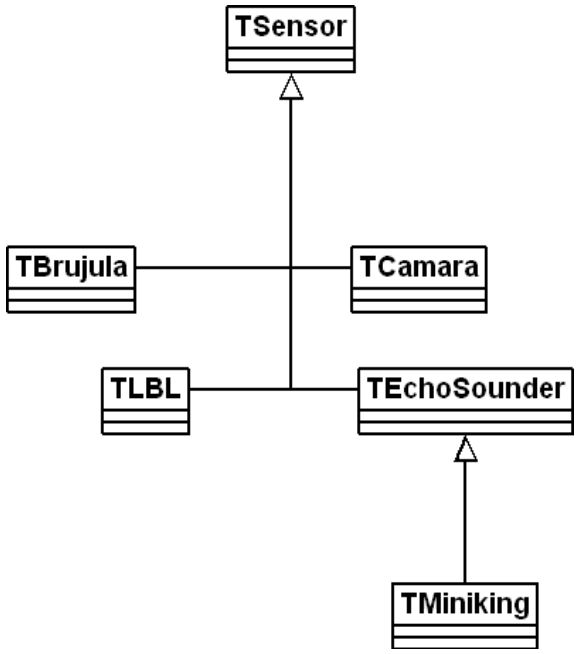

Figura 5.7: Sensores disponibles en la aplicación

agua y *adinterval* es el número de intervalos de 640 ns en los que se divide cada *bin*.

Por ejemplo, si el sónar virtual está configurado para trabajar con 300 *bins*, el parámetro *adinterval* vale 208 y la función *GetDstObstáculo* indica que se ha detectado un obstáculo a 10 metros en la orientación actual, el número de *bin* que deberá marcarse con valor de intensidad máxima en la trama de respuesta *MtHeadData* es:

$$\frac{2 \times 10}{1500 \times 208 \times 640\ ns} = 100{,}16\,,$$

que, tras redondear, se queda en el *bin* número 100. Por tanto, los valores en decimal de los *bins* en el mensaje serán los mostrados en la figura 5.8.

La función *GetDstObstaculo* devuelve el valor -1 en aquellas situaciones en las que no detecta ningún obstáculo. En este caso, la trama *MtHeadData* que se crea como respuesta incluye todos los valores de los *bins* con el nivel de intensidad 0.

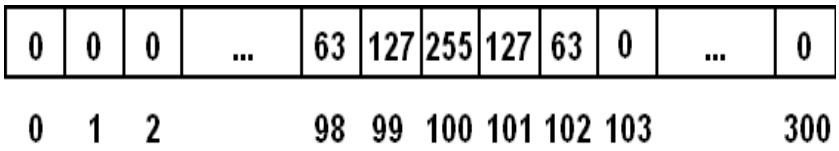

Figura 5.8: Valor de los *bins* de la trama *MtHeadData*

En el otro lado de la comunicación debe haber una instancia de la librería que acceda a estos datos a través del objeto *TSerialPort*, los interprete y modifique la trayectoria del vehículo en relación a ellos. Los comportamientos de un AUV en el simulador se gestionan desde la capa de coordinación, representada mediante la clase abstracta *TCoordinador*. De dicha clase derivan dos subclases correspondientes a los dos mecanismos de coordinación típicos: cooperativo (*TCCooperativo*) y competitivo (*TCCompetitivo*). Estos objetos combinan las salidas de una serie de comportamientos básicos, como los mostrados en la sección anterior, a través de la llamada a sus métodos *GenSalida*, que es un procedimiento encargado de generar un vector de salida de acuerdo con la funcionalidad del comportamiento en cuestión.

Por tanto, ha sido necesario crear un comportamiento simple que accediera a través de la librería a los datos del sónar y a partir de ellos generase un vector de movimiento. Esta clase se ha llamado *TAOMiniking*, y accede al dispositivo virtual a través de las correspondientes funciones programadas para ello. Los comportamientos disponibles en la aplicación derivan de una clase abstracta llamada *TBehavior*, incluido el comportamiento de evitación de obstáculos, *TAvoidObstacles*. La clase de acceso al sónar *TAOMiniking* desarrollada deriva de esta última, heredando todos los métodos necesarios para su correcta gestión.

Como hemos mencionado anteriormente, de todos los métodos implementados en *TAOMiniking* destaca *GenSalida*, que accede al dispositivo, analiza la información obtenida y genera un vector de salida de acuerdo con los obstáculos detectados. Para mejorar la eficiencia del comportamiento, se realiza un barrido completo del entorno en lugar de generar un vector por cada *ping* en cada llamada al procedimiento. De esta forma, la salida es la suma ponderada de todos los vectores obtenidos durante dicho barrido. Así pues, el procedimiento calcula el número de

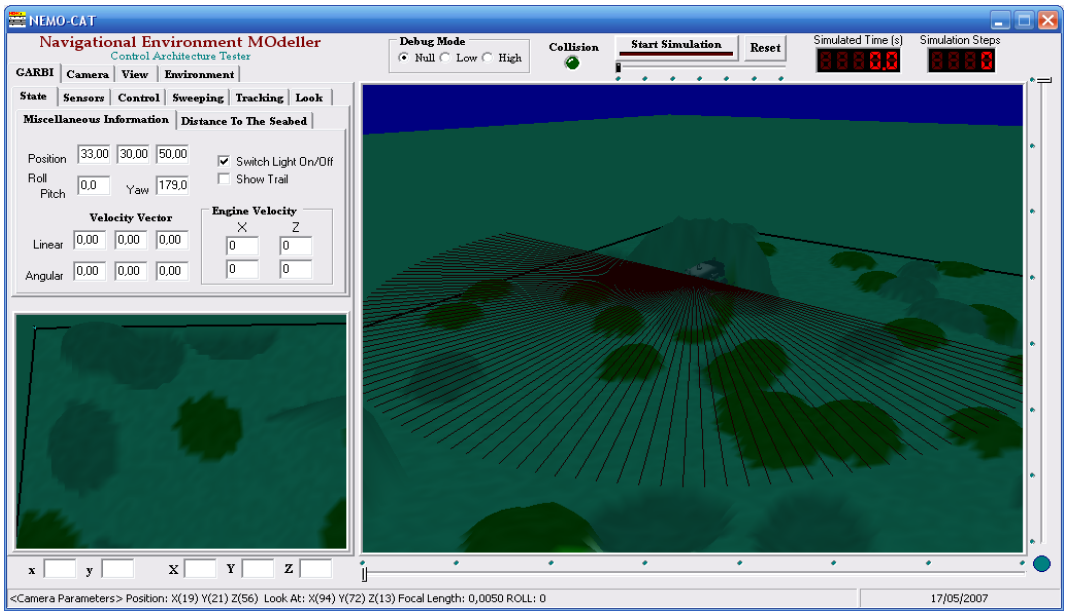

Figura 5.9: Representación gráfica del barrido del sónar Miniking en el simulador $NEMO_{CAT}$

pings que deberá realizar de acuerdo con los parámetros de límite incluidos en el cabezal virtual, y realiza las correspondientes llamadas a la función *getScanLine* de la librería. Cada una de las tramas devueltas es procesada y se detecta la distancia al obstáculo a partir de los valores de intensidad de *bins* que incluye.

Por último, ha sido necesario redefinir ciertas características del código de la librería original, para permitirle interactuar con el sónar virtual. En la capa más baja de la jerarquía, la capa de acceso al puerto serie, se han reescrito las funciones de lectura y escritura, permitiendo la posibilidad de actuar sobre un objeto de tipo *TSerialPort*. En la capa superior de la arquitectura software de la librería se han reescrito los métodos fundamentales para permitir sincronizar el funcionamiento del dispositivo virtual con las funciones de la librería que implementan partes del protocolo *SeaNet*.

5.5. Resultados

Se han realizado una serie de pruebas que muestran la utilidad de la librería dentro de la arquitectura de un *AUV*. El modelo de robot elegido para los experimentos es *GARBI*, que incluye un sónar *Miniking* como único dispositivo para la detección de obstáculos. El cabezal se ha configurado para trabajar en un rango de 180°, como muestra la figura 5.9.

5.5.1. Primer experimento

El primer sistema de control propuesto intenta llevar a cabo una sencilla tarea: alcanzar una secuencia de puntos considerados *objetivo* por el usuario dentro de un entorno con obstáculos. En este caso, una arquitectura reactiva pura es suficiente para llevar a cabo el experimento.

El entorno para esta primera prueba se muestra en la figura 5.10. Como obstáculos destacan dos grandes rocas. Detrás de cada una de ellas se encuentra un punto objetivo, lo que obliga al vehículo a evitarlas para poder llegar a ellos. Durante la simulación, se puede apreciar como el robot detecta la presencia de dichas rocas y modifica su trayectoria de forma consecuente para llegar al punto objetivo. Esta situación se muestra en la figura 5.11.

De todos los comportamientos utilizados en este experimento, destacan *Ir a un punto* y por supuesto el comportamiento de *Evitación de obstáculos* desarrollado en esta última parte del proyecto, que interactúa con el sónar virtual *Miniking*. La trayectoria seguida por el vehículo se puede apreciar en la figura 5.12.

5.5.2. Segundo experimento

En este segundo experimento se pretende realizar el seguimiento de un cable depositado en el fondo marino. Las tareas de evitación de obstáculos realizadas en el apartado anterior siguen siendo igualmente válidas. En este experimento, el vehículo realiza el movimiento en *zig zag* característico del comportamiento de detección del cable.

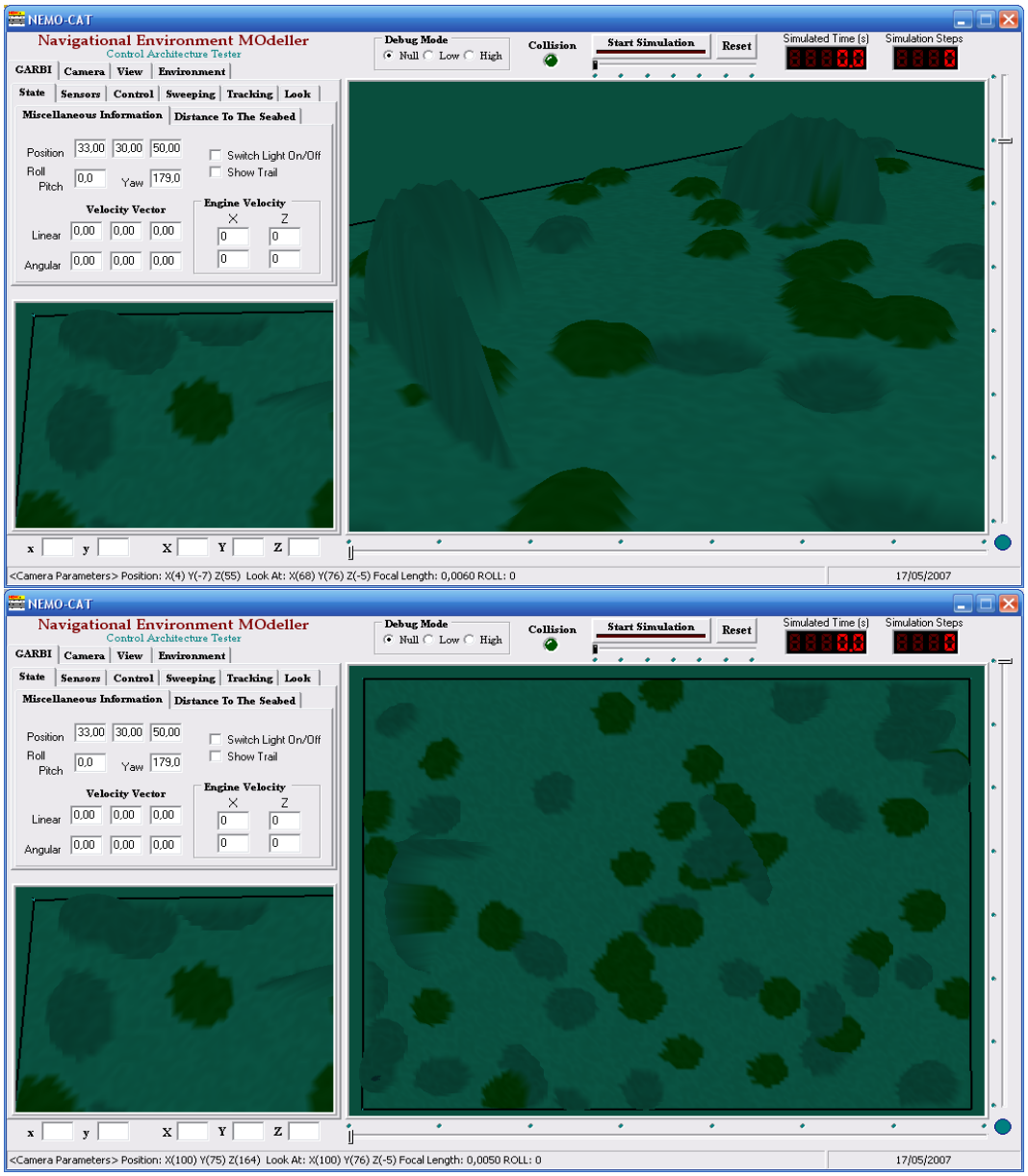

Figura 5.10: Entorno virtual utilizado en el primer experimento

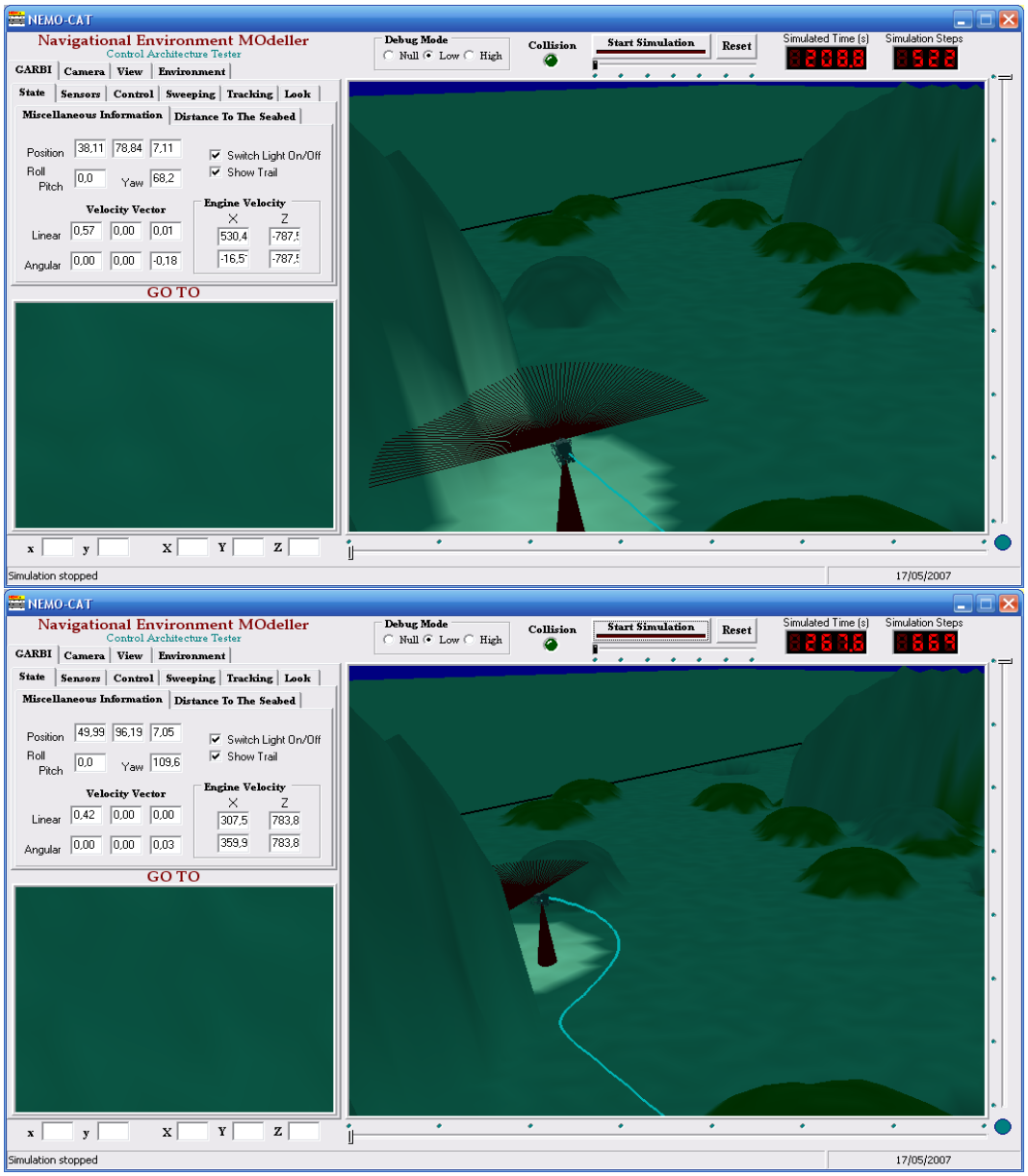

Figura 5.11: Evitación de un obstáculo dentro del entorno virtual en el primer experimento

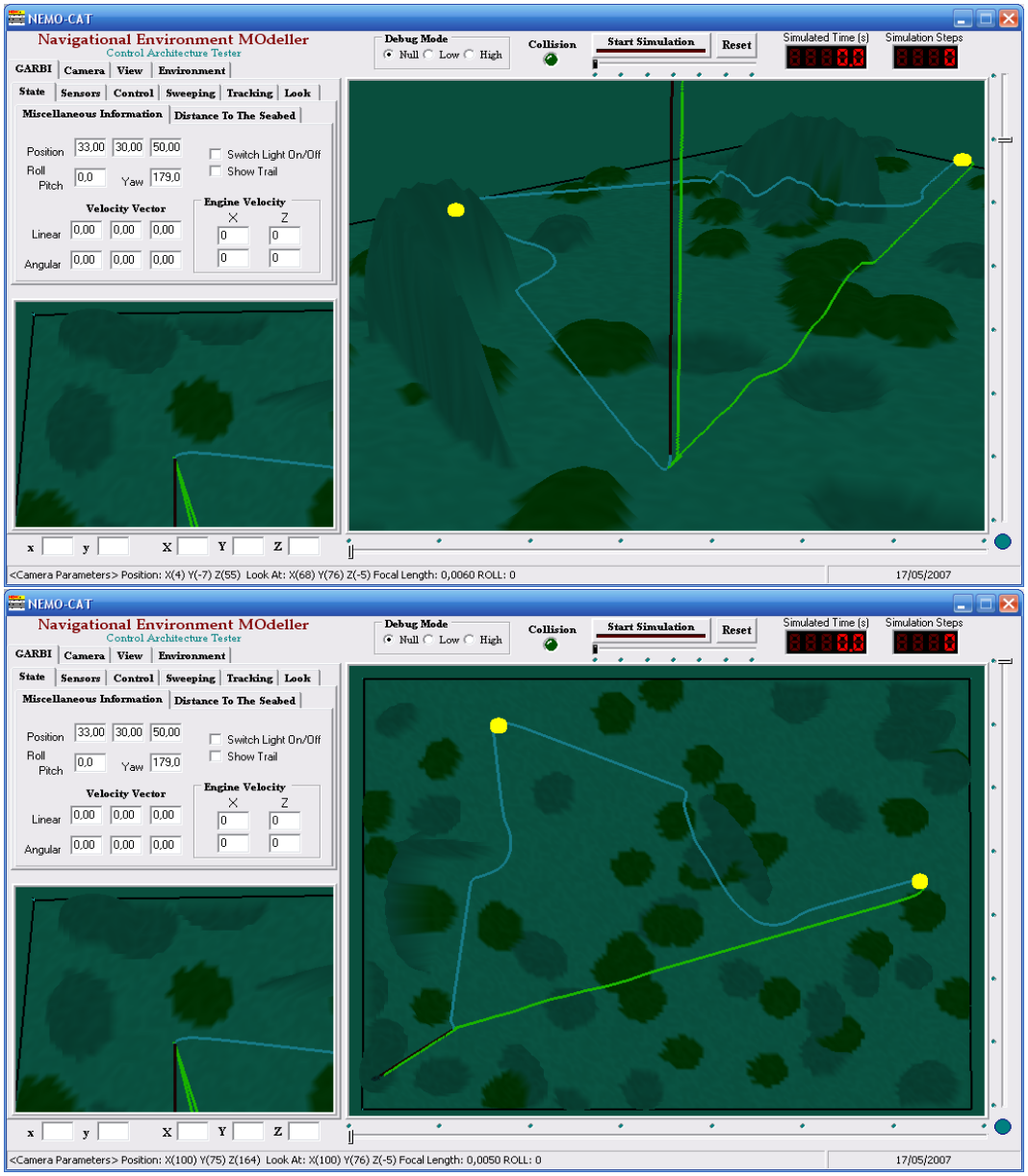

Figura 5.12: Trayectoria seguida por el vehículo en el primer experimento. La línea roja representa el descenso al fondo marino, la azul se corresponde con la trayectoria seguida hasta alcanzar los dos puntos objetivos y la verde la ruta de retorno al punto inicial de la simulación. Los puntos objetivos están marcados por círculos de color amarillo.

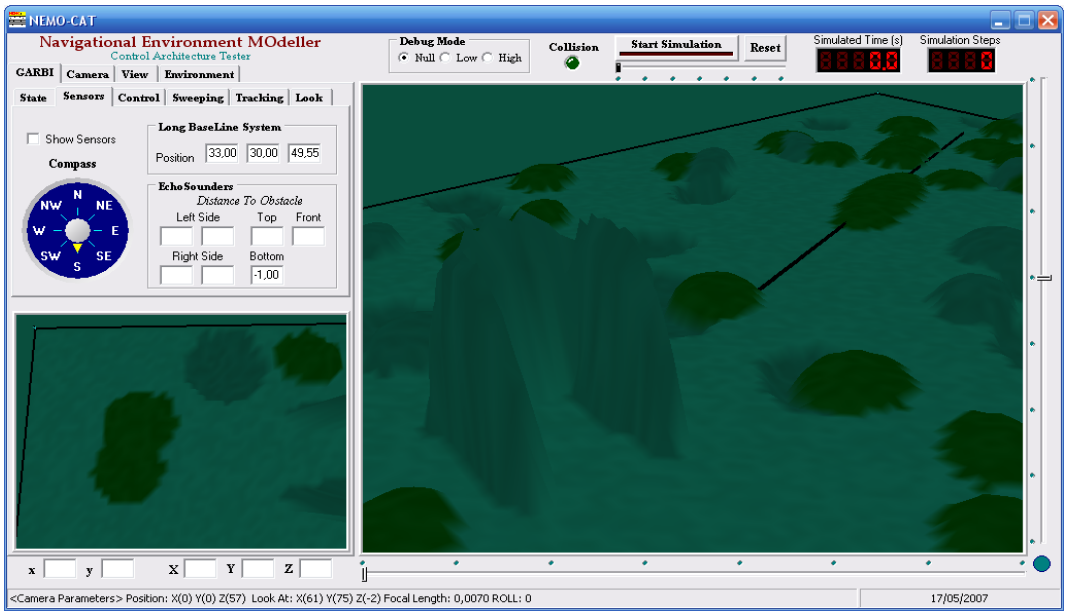

Figura 5.13: Entorno virtual del segundo experimento

Se ha añadido una roca al entorno para completar una estructura en cañón rectangular, como se aprecia en la figura 5.13. El objetivo es comprobar el funcionamiento del comportamiento *Evitar el pasado*, que debe permitir al vehículo escapar del pozo de potencial que se genera en esa situación.

Una parte del cable ha sido sepultado bajo un alga. Con ello se pretende analizar el comportamiento del vehículo al perder de su línea de visión dicho cable. Durante el experimento se puede comprobar como al encontrarse en esta situación, el robot retoma su movimiento en *zig zag* pero de una forma restringida, analizando las zonas más próximas de acuerdo con su trayectoria.

Los comportamientos más destacables de esta arquitectura son el de *Detección y seguimiento de un cable submarino*, *Evitar el pasado* y el comportamiento de *Evitación de obstáculos* utilizado en el apartado anterior. La trayectoria final del vehículo puede verse en la figura 5.14.

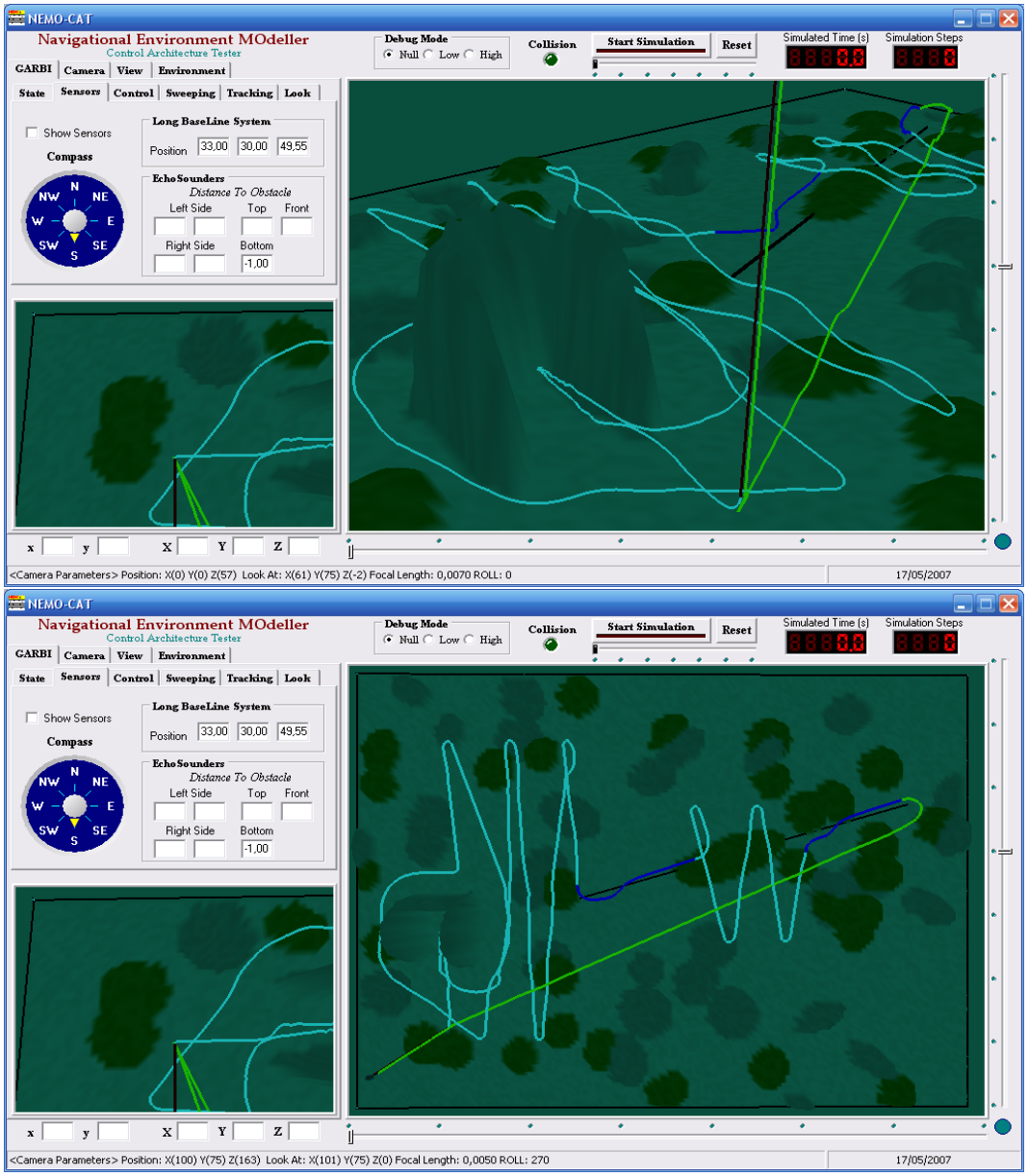

Figura 5.14: Trayectoria seguida por el vehículo en el segundo experimento. El color rojo representa el descenso al fondo marino, el color azul claro la trayectoria seguida por el vehículo, el color azul oscuro se corresponde con el seguimiento del cable submarino y el verde con la ruta de retorno al punto inicial de la simulación.

Capítulo 6

Experiencia y conclusiones

En este último capítulo se recogen las impresiones, experiencias personales, opiniones y conclusiones obtenidas tras el desarrollo de este proyecto.

6.1. Consecución de los objetivos

En este trabajo se ha mostrado cómo implantar y poner a punto un sensor sónar para controlar la trayectoria de un vehículo submarino dentro de un entorno con obstáculos. La interacción con el dispositivo se ha realizado gracias a una librería de acceso, mientras que el control de la trayectoria se ha conseguido configurando los comportamientos que componen la arquitectura software del robot.

En relación a los objetivos que se establecieron en la etapa inicial del proyecto, se tiene la convicción de haberlos alcanzado de una forma clara. Gracias a la librería realizada se pueden desarrollar aplicaciones que necesiten acceso al sónar *Miniking* de una forma sencilla, intuitiva y eficiente. Las funciones implementadas son lo suficientemente simples, desde el punto de vista del usuario, como para permitir su utilización en un corto período de aprendizaje.

Como bien se ha mostrado en este documento, la comunicación entre el dispositivo y un computador a través del puerto serie implica todo un protocolo de intercambio de tramas cono-

cido como *SeaNet*. La librería permite ocultar todos estos detalles de bajo nivel, permitiendo así la interacción con el sónar sin necesidad de conocer dicho protocolo.

Además, la estructuración de la librería en una jerarquía por capas permite modificar cualquiera de sus niveles sin que afecte al resto. Este hecho se ha comprobado en la última parte del proyecto, en la que se ha redefinido la capa más baja de la arquitectura para permitir a la librería acceder a un puerto serie simulado sin que haya sido necesario modificar los niveles superiores.

El funcionamiento y la utilidad de la librería se ha mostrado con el desarrollo de la aplicación *Miniking Viewer*, que permite observar el entorno en el que se encuentra el sónar de una forma gráfica. Se han incluido además una serie de funcionalidades que permiten configurar el dispositivo en tiempo real. Dicha aplicación puede instalarse en un computador y utilizarse para visualizar el entorno en el que se encuentra el cabezal en un momento concreto.

Como último objetivo, se ha conseguido implantar la librería dentro de un entorno simulado, y se ha mostrado su correcto funcionamiento y utilidad en el control de la navegación de un vehículo submarino. En primer lugar, se dispone desde este momento de un tipo de sensor sónar dentro de la aplicación $NEMO_{CAT}$, que permite incluirlo en un vehículo AUV para realizar tareas de evitación de obstáculos en un entorno simulado. Este hecho evita la necesidad de definir diferentes sensores ultrasónicos monohaz para poder abarcar todo el rango angular del robot, como era necesario anteriormente. Por otro lado, los experimentos llevados a cabo en la última parte permiten comprobar el correcto funcionamiento de la arquitectura software por capas planteada en la librería, y muestran la posibilidad de realizar pruebas de funcionamiento del sónar dentro de robots sin necesidad de llevarlo a un entorno real.

6.2. Experiencias del proyecto

Existen una serie de objetivos que, si bien no se consideran propios del proyecto, se han llevado a cabo y han colaborado en la consecución de los tres principales. En primer lugar, ha sido

necesario comprender el funcionamiento interno del sónar, su modo de actuar, su configuración, etc. Por otro lado, para implementar la librería se ha necesitado conocer con exactitud el mecanismo del protocolo *SeaNet*, con vistas a implementar los procesos que se dan dentro del mismo. Estas tareas no han sido triviales, debido a la escasa e incompleta documentación proporcionada por *Tritech*.

Es necesario remarcar también el conocimiento adquirido en el campo de la robótica submarina, tanto teórico como práctico. En el plano teórico se ha investigado sobre los posibles tipos de robots submarinos y las posibles aplicaciones de esta rama de la robótica, además de profundizar en el control de alto nivel para vehículos robóticos. En la práctica, se ha conseguido programar un protocolo de interacción con un dispositivo sónar así como utilizar la información obtenida de él para controlar la trayectoria de un robot basado en comportamientos.

La principal dificultad para implantar la librería en el simulador ha sido la extensión de la propia aplicación. $NEMO_{CAT}$ consta de más de cien clases, por lo que ha sido necesario analizarla con detenimiento para comprender su funcionamiento y poder así cumplir nuestro objetivo.

De acuerdo con los resultados obtenidos, no sería dificil implantar la librería en una arquitectura real. Sería necesario analizar el funcionamiento de dicha arquitectura y programar el comportamiento de evitación de obstáculos para obtener los datos a partir del dispositivo, como se ha hecho en la última parte del proyecto. Es posible que el vehículo lleve un computador a bordo y que los resultados se deseen mostrar en la cónsola del operador. En esa situación será necesario reescribir una parte de la aplicación *Miniking Viewer* para permitir obtener datos a través de una conexión *Ethernet*. La librería obtendrá la información del sónar desde el computador insertado en el vehículo, y enviará los resultados al computador del operador a través de la conexión de área local.

Como experiencia personal cabe mencionar que el proyecto me ha permitido poner en práctica conocimientos, técnicas y procedimientos aprendidos durante la carrera de Ingeniería Informática.

En concreto, y debido a la temática, he podido comprender mejor ciertos conceptos de asignaturas como Robótica, y ponerlos en práctica. Además me ha permitido profundizar en la programación en el lenguaje *C++* y el diseño de interfases gráficas.

6.3. Trabajo futuro

Dentro de las líneas de trabajo futuro, podemos destacar la posible actualización de la librería para trabajar con otros modelos de dispositivos *Tritech*. En la actualidad es capaz de interactuar únicamente con el modelo *Miniking*, debido a las diferencias estructurales que presenta con otros modelos, como el modelo *Seaking*, por ejemplo.

Actualmente la librería sólo funciona bajo la plataforma Windows. Por tanto, otra posible mejora es adaptar la librería para trabajar con otros sistemas operativos, como Linux o Mac OS. Para ello, únicamente sería necesario reescribir la capa de acceso al puerto serie con las primitivas correspondientes de cada sistema operativo.

Respecto a la última parte del proyecto, una tarea pendiente que no se ha podido llevar a cabo por falta de un vehículo real operativo es comprobar el funcionamiento de la librería dentro de un entorno real, programando los comportamientos adecuados para el vehículo utilizado en el experimento. Sería interesante investigar acerca de cómo se comporta un robot submarino de acuerdo con la información que obtiene del dispositivo sónar en diferentes entornos a los utilizados en este proyecto, permitiendo así mejorar la capacidad y funcionalidad de la librería.

Bibliografía

[1] *Aplicación de técnicas de visión por computador a entornos submarinos*
Memoria de investigación, Departamento de Matemáticas e Informática, UIB
Alberto Ortiz Rodríguez (1998)

[2] *Fundamentos de Robótica*
McGraw Hill, 1997
Antonio Barrientos

[3] *Desarrollo de un sistema de sensorización por ultrasonido para aplicaciones de alta fiabilidad*
Proyecto de final de carrera, Escuela Politécnica Superior, UIB
Alejandro Bennásar Sevillá (2006)

[4] *In acoustic signal processing for ocean exploration*
NATO ASI Series, 1992
Kluwer Academic Publishers
Finn Jensen

[5] *Proyecto wxWidgets*
`http://www.wxwidgets.org`

[6] *Wikipedia, la enciclopedia libre*
`http://es.wikipedia.org/wiki`

[7] *Software notes for controlling and operating RS-232 Sonar Heads*
Tritech International

[8] *Decoding sonar scanline data*
Tritech International

[9] *Tritech MiniKing OEM Sonar, Installation and Operators manual*
Tritech International

[10] *An underwater simulation environment for testing autonomous robot control architectures*
Actas de *IFAC conference on control applications in marine systems*
Ancona(Italia), Julio 2004
Javier Antich, Alberto Ortiz

Apéndice A

Tramas del protocolo *SeaNet*

A.1. Comandos

A.1.1. MtSendVersion

40	30	30	30	38	08	00	FF	02	03	17	80	02	0A
Hdr '@'	Hex Length = 8 bytes				Bin Length = 8 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 3	**mtS end Ver'**	Seq = End	Nde 02	LF

A.1.2. MtSendBBUser

40	30	30	30	38	08	00	FF	02	03	18	80	02	0A
Hdr '@'	Hex Length = 8 bytes				Bin Length = 8 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 3	**mtS end BB'**	Seq = End	Nde 02	LF

A.1.3. MtReboot

40	30	30	30	38	08	00	FF	02	03	10	80	02	0A
Hdr '@'	Hex Length = 8 bytes				Bin Length = 8 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 3	**mtR eBo ot**	Seq = End	Nde 02	LF

A.1.4. MtSendData

40	30	30	30	43	0C	00	FF	02	07	19	80	02
Hdr '@'	Hex Length = 12 bytes				Bin Length = 12 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 7	**mtS end D'ta**	Seq = End	Nde 02
CA	**64**	**B0**	**03**	**0A**								
Current Time = 61891786 msec = 17:11:31.79				LF								

A.1.5. MtHeadCommand

40	30	30	34	43	4C	00	FF	02	47	13	80	02	1D
Hdr '@'	Hex Length = 76 bytes				Bin Length = 76 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 71	**mtH 'dC md**	Seq = End	Nde 02	V3B Par-ams
07	**23**	**02**	**99**	**99**	**99**	**02**	**66**	**66**	**66**	**05**	**A3**	**70**	**3D**
HdCtrl * = 8967		HdT ype = 02	TXN, Ch1 (325kHz) = 43620761				TXN, Ch2 (675kHz) = 90596966				RXN, Ch1 (325kHz) = 104689827		
06	**70**	**3D**	**0A**	**09**	**13**	**01**	**E8**	**03**	**00**	**00**	**E0**	**18**	**53**
	RXN, Ch2 (675kHz) = 151666032				TxPulse-Len = 275 usec		Range-Scale = 100m		LeftLim = 0 (1/16 Grad)		RightLim = 6368 (1/16 Grad)		ADS p'n = 83
30	**6B**	**6B**	**5A**	**00**	**7D**	**00**	**19**	**10**	**54**	**03**	**FA**	**00**	**E8**
ADL ow = 48	Iga-in, Ch1	Iga-in, Ch2	Slope, Ch1 = 90		Slope, Ch2 = 125		Mo' Tme = 25	Step Size = 16	ADInterval = 852		Nbins = 250		Max ADb uf
03	**64**	**00**	**40**	**06**	**01**	**00**	**00**	**00**	**53**	**53**	**30**	**30**	**6B**
= 100 0	Lockout = 100 usec		MinorAxis-Dir = 1600 (1/16 Grad)		Maj' Axis Pan	Ctl2 = 0	ScanZ = 0		AD Sp'n Ch1	AD Sp'n Ch2	AD Low Ch1	AD Low Ch2	Iga-in Ch1
6B	**00**	**00**	**5A**	**00**	**7D**	**00**	**00**	**00**	**00**	**00**	**0A**		
Iga-in, Ch2	Adc Set, Ch1	Adc Set, Ch2	Slope, Ch1 = 90		Slope, Ch2 = 125		Slope Delay, Ch1 = 0		Slope Delay, Ch2 = 0		LF		

A.2. Mensajes

A.2.1. MtVersionData

40	30	30	31	33	13	00	02	FF	0E	01	80	02	31
Hdr '@'	Hex Length = 19 bytes				Bin Length = 19 bytes		Tx Nde 02	Rx Nde 255	No. Byte 14	**mtVer'n D'ta**	Seq = End	Nde 02	CPU ID
11	**0D**	**8C**	**83**	**A8**	**00**	**00**	**3C**	**88**	**02**	**0A**			
CPU ID = 8C0D1131H			Program Length = 43139				Checksum = 34876		Nde 02	LF			

A.2.2. MtAlive

40	30	30	31	30	10	00	02	FF	0B	04	80	02	80
Hdr '@'	Hex Length = 16 bytes				Bin Length = 16 bytes		Tx Nde 02	Rx Nde 255	No. Byte = 11	**mtAlive**	Seq = End	Nde 02	No Params
C4	**37**	**00**	**00**	**80**	**0C**	**CA**	**0A**						
Head Time = 14276 msec				Motor Pos = 3200 (1/16 Grad)		He-Ad Inf*	LF						

A.2.3. MtHeadData

40	30	31	32	38	28	01	02	FF	00	02	80	02	19
Hdr '@'	Hex Length = 296 bytes				Bin Length = 296 bytes		Tx Nde 02	Rx Nde 255	Sin-gle pckt	**mtH ead D'ta**	Seq = End	Nde 02	->
01	**02**	**10**	**00**	**07**	**23**	**E8**	**03**	**99**	**99**	**99**	**02**	**6B**	**5A**
Cou nt = 281	= Son h'd	Stat -us	Sw-eep	HdCtrl = 8967		Range = 100m		TxN = 1717986821				Gai n = 51%	->
00	**53**	**32**	**00**	**00**	**54**	**03**	**00**	**00**	**E0**	**18**	**10**	**80**	**0A**
Slo-pe= 90	ADS p'n	ADL ow	Heading Offset = 0		AD Interval = 852		L.Limit = 0		R.Limit = 6368 (1/16 Grad)		Ste-ps = 16	Bearing = 3984 (1/16 Grad)	
FA	**00**	**00**	**00**	**00**			**00**	**FF**	**FF**	**83**		**00**	**00**
Dbytes = 250		Bin 1 = 0	Bin 2 = 0	Bin 3 = 0	→	→	Bin 174 = 0	Bin 175 =255	Bin 176 =255	Bin 177 =131	→	Bin 249 = 0	Bin 250 = 0

0A
LF

A.2.4. MtBBUserData

40	**30**	**30**	**34**	**43**	**4C**	**00**	**FF**	**02**	**47**	**13**	**80**	**02**	**1D**
Hdr '@'	Hex Length = 76 bytes				Bin Length = 76 bytes		Tx Nde 255	Rx Nde 02	No. Byte = 71	**Mt H'd Cmd**	Seq = End	Nde 02	V3B Para -ms
83	**23**	**02**	**99**	**99**	**99**	**02**	**66**	**66**	**66**	**05**	**A3**	**70**	**3D**
HdCtrl * = 9091		HdType = 02	TXN, Ch1 = 43620761				TXN, Ch2 = 90596966				RXN, Ch1 = 104689827		
06	**70**	**3D**	**0A**	**09**	**28**	**00**	**3C**	**00**	**01**	**00**	**FF**	**18**	**51**
	RXN, Ch2 = 151666032				TxPulseLen = 40 usec		RangeScale = 6 metres		LeftLim = 1 (1/16 Grad)		RightLim = 6399 (1/16 Grad)		AD Sp'n = 81
08	**54**	**54**	**5A**	**00**	**7D**	**00**	**19**	**10**	**8D**	**00**	**5A**	**00**	**E8**
AD Low = 8	Igain, Ch1	Igain, Ch2	Slope, Ch1 = 90		Slope, Ch2 = 125		Mo' Tme = 25	Step Size = 16	ADInterval = 141		Nbins = 90		Max ADbuf
03	**97**	**03**	**40**	**06**	**01**	**00**	**00**	**00**	**50**	**51**	**09**	**08**	**54**
Lo = '0' = 8	Slu1 D'taBit Hi = '0' = 8 bits		Has Mot' r = 1	D1	Has Aux = 0	D2	Has Roll = 0	D3	Roll Coeff Mul = 0		Roll Coeff Div = 0		Roll Coeff ->
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
Offs = 0	Has P/T = 0	D4	Press Coeff Mul = 0		Press Coeff Div = 0		Press Coeff Offs = 0		Temp Coeff Mul = 0		Temp Coeff Div = 0		Tmp Coeff ->
00	**00**	**00**	**00**	**00**	**06**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
Offs = 0	Fire 2Ch = 0	D5	Emu lv18 = 0	D6	B'rd Rev = 6	D7	V18 Node = 0		Aux 1sec = 0	D10	Dual Axis = 0	D11	Stabilis'd = 0

00	**00**	**00**	**45**	**01**	**A3**	**02**	**45**	**01**	**A3**	**02**	**5A**	**00**	**7D**
D12	Rs-485 = 0	D13	Tx, Ch1 = 325		Tx, Ch2 = 675		Rx, Ch1 = 325		Rx, Ch2 = 675		Slope, Ch1 = 90		Slope, ->
00	**00**	**00**	**00**	**00**	**64**	**00**	**64**	**00**	**00**	**00**	**C8**	**00**	**20**
Ch2 = 125	PrfTime-Cal, Ch1 = 0		PrfTime-Cal, Ch2 = 0		SonTime-Cal, Ch1 = 100		SonTime-Cal, Ch2 = 100		V4 Mde = 0	D14	MotSteps (Pan) = 200		Max Speed, ->
03	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
Pan = 800	MechLL, Pan = 0 Cont. Rot'n		MechRL, Pan = 0 Cont. Rot'n		MotSteps, Tilt = 0		Max Speed, Tilt = 0		MechLL, Tilt = 0 Cont. Rot'n		MechRL, Tilt = 0 Cont. Rot'n		Txdcr1, Ch1
00	**00**	**00**	**01**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
-> Pofs = 0	Txdcr1, Ch1 TOfs = 0		Txdcr1, Ch1 Type = 1		Txdcr1, Ch1 Gain = 0		Txdcr1, Ch1 HBm = 0		Txdcr1, Ch1 VBm = 0		Txdcr1, Ch2 Pofs = 0		Txdcr1, Ch2
00	**01**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
-> Tofs = 0	Txdcr1, Ch2 Type = 1		Txdcr1, Ch2 Gain = 0		Txdcr1, Ch2 HBm = 0		Txdcr1, Ch2 VBm = 0		Txdcr2, Ch1 Pofs = 0		Txdcr2, Ch1 TOfs = 0		Txdcr2, Ch1
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
-> Type =0	Txdcr2, Ch1 Gain = 0		Txdcr2, Ch1 HBm = 0		Txdcr2, Ch1 VBm = 0		Txdcr2, Ch2 Pofs = 0		Txdcr2, Ch2 TOfs = 0		Txdcr2, Ch2 Type = 0		Txdcr2, Ch2
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
-> Gain = 0	Txdcr2, Ch2 HBm = 0		Txdcr2, Ch2 VBm = 0		ELScanCode = 0		Dual Synth =0	D15	ADLow data = 0		ParaTxRec = 0		
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
	MotorAmp Gain = 0		1st Empty	2nd Empty	…	…	…	…	…	…	…	…	…

00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
…	…	…	…	…	…	…	…	…	…	…	…	…	…
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
…	…	…	…	…	…	…	…	…	…	…	…	…	…
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**
…	…	…	…	…	…	…	…	…	…	…	…	…	…
00	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**00**	**0A**		
…	…	…	…	…	…	…	…	…	…	64^{th} Empty	LF		

Apéndice B

Tecnología y herramientas utilizadas

- **C++:** Es un lenguaje de programación diseñado a mediados de los años 80 por Bjarne Stroustrup como extensión al lenguaje *C*. Soporta abstracción de datos, programación orientada a objetos y programación genérica. Posee además una serie de características difíciles de encontrar en otros lenguajes, como la posibilidad de redefinir los operadores y la identificación de los objetos en tiempo de ejecución. Está considerado uno de los lenguajes más potentes, al permitir trabajar tanto a alto como a bajo nivel. En este proyecto se ha utilizado como lenguaje general de programación.

- **Dev-C++:** *Bloodshed Dev-C++* es un entorno de desarrollo integrado (*IDE* por sus siglas en inglés) para programar en lenguaje C/C++. Usa MinGW que es una versión de GCC (GNU Compiler Colletion) como su compilador. Dev-C++ puede además ser usado en combinación con Cygwin y cualquier compilador basado en *GCC*. El entorno está desarrollado en el lenguaje *Delphi* de *Borland*. Ha sido el *IDE* utilizado en las dos primeras partes del proyecto.

- **Doxygen:** Es un generador de documentación para *C++*, *C*, *Java*, *Objective-C*, *Python*, *IDL* (versiones Corba y Microsoft) y en cierta medida para *PHP*, *C#* y *D*. Dado que es fácilmente adaptable, funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X. Es un acrónimo de *dox*(document) *gen*(generator), generador de documentos. Se ha utilizado en la documentación de la librería, incluida en el *CD* del proyecto.

- **WxWidgets:** son unas librerías multiplataforma, *freeware/opensource* para el desarrollo de interfases gráficas programadas en lenguaje *C++*, que usan una licencia L-GPL, similar a la GPL. Las *wxWidgets* proporcionan una interfase gráfica basada en las bibliotecas ya existentes en el sistema (nativas), con lo que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos. Se ha utilizado para programar las características gráficas del entorno visual *Miniking Viewer*, la segunda parte de este proyecto.

- **XML:** siglas en inglés de *eXtensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (*W3C*). Es una simplificación y adaptación del *SGML* y permite definir la gramática de lenguajes específicos (de la misma manera que *HTML* es a su vez un lenguaje definido por *SGML*). Por lo tanto *XML* no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. El uso de esta tecnología en el proyecto queda reducido a la gestión de configuraciones del programa *Miniking Viewer*.

- **TinyXML:** Es un sencillo *parser* de *XML* para *C++* fácilmente integrable en cualquier aplicación. Es la librería que ha permitido al programa *Miniking Viewer* interactuar con *XML*.

- **OpenGL:** *OpenGL* es una especificación estándar que define una *API* multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Fue desarrollada por *Silicon Graphics Inc.* (SGI) en 1992. En este proyecto, su uso se ha reducido a pequeñas modificaciones realizadas sobre el simulador $NEMO_{CAT}$.

- **LaTeX:** Es un procesador de textos basado en un lenguaje de marcado formado por un gran conjunto de macros de *TeX*, escritas inicialmente por Leslie Lamport (*LamportTeX*) en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea. LaTeX es software libre bajo licencia LPPL. Toda esta memoria ha sido redactada a través de dicho procesador de textos. La distribución LaTeX utilizada ha sido *MiKTeX*.

- **TEXnic Center:** Es un entorno integrado para el desarrollo de documentos LaTeX sobre Windows. Incluye todas las herramientas necesarias para facilitar la tarea del redactor. Se ha utilizado dicho *IDE* para la utilización de LaTeX durante la redacción de este documento.

Apéndice C

Código fuente de la librería

C.1. Capa 1: Acceso al puerto serie

COMPort.h

```
/*
================================================================================
  Name: COMPort.h
  Author: Emilio García Fidalgo
  Date: 05/10/06 14:00
  Description: Class for a RS232 Communication.
  Based on UAB BBDSoft ( http://www.bbdsoft.com/ ).
================================================================================
*/

#ifndef _COMPORT_H
#define _COMPORT_H

using namespace std;

// Parity

typedef enum {

        None = 0,
        Odd,
        Even,
        Mark,
        Space
} Parity;

// Data Bits

typedef enum {
```

```
        db4 = 4,
        db5,
        db6,
        db7,
        db8
} DataBits;

// Stop Bits

typedef enum {

        sb1 = 0,
        sb15,
        sb2
} StopBits;

// Bit Rates

typedef enum {

        brdefault = 0,
        br110 = 110,
        br300 = 300,
        br600 = 600,
        br1200 = 1200,
        br2400 = 2400,
        br4800 = 4800,
        br9600 = 9600,
        br19200 = 19200,
        br38400 = 38400,
        br56000 = 56000,
        br57600 = 57600,
        br115200 = 115200,
        br256000 = 256000
} BitRate;

class COMPort {

      private:
              BitRate br;
              DataBits db;
              Parity par;
              StopBits sb;
              unsigned thePortHandle;
              char *theDCB;

              void getState() const;
              COMPort& setState();
      public:

              // Contructor & Destructor

              COMPort(char *, BitRate, DataBits, Parity, StopBits, int, bool);
              ~COMPort();

              // General functions
```

```
                unsigned long getBitRate(void);
                void setBitRate(BitRate);

                DataBits getDataBits(void);
                void setDataBits(DataBits);

                Parity getParity(void);
                void setParity(Parity);

                StopBits getStopBits(void);
                void setStopBits(StopBits);

                COMPort & setHandshaking(bool);

                COMPort& setBlockingMode (
                          unsigned long inReadInterval = 0
                        , unsigned long inReadMultiplyer = 0
                        , unsigned long inReadConstant = 0
                );

                // I/O functions

                unsigned char read(void);
                unsigned long read(unsigned char *, int ch);
                void write(unsigned char);
};

#endif
```

COMPort.cpp

```
/*
================================================================================
  Name: COMPort.cpp
  Author: Emilio García Fidalgo
  Date: 05/10/06 14:37
  Description: Class for a RS232 Communication.
  Based on UAB BBDSoft ( http://www.bbdsoft.com/ ).
================================================================================
*/

#include "COMPort.h"
#include <windows.h>
#include <stdexcept>

// Constructor

COMPort::COMPort(char *Name, BitRate br, DataBits db, Parity par, StopBits sb, int timeout, bool hs)
: theDCB(NULL) {

                // Open COM Port as a file
```

```
                thePortHandle = (unsigned ) CreateFile ( Name
                                , GENERIC_READ | GENERIC_WRITE
                                , 0
                                , NULL
                                , OPEN_EXISTING
                                , FILE_FLAG_NO_BUFFERING
                                , NULL
                                );

                if (thePortHandle == (unsigned)HFILE_ERROR) {
                   throw runtime_error("COMPort: failed to open.");
                }

                // Configure port

                theDCB = new char[sizeof(DCB)];
                getState();
                if (timeout == 0)
                   setBlockingMode();
                else
                    setBlockingMode(timeout,2,500);
                setBitRate(br);
                setParity(par);
                setDataBits(db);
                setStopBits(sb);
                setHandshaking(hs);
}

// Destructor

COMPort::~COMPort() {

     delete [] theDCB;

     if (CloseHandle ((HANDLE)thePortHandle) == FALSE ) {
        throw runtime_error ("COMPort: failed to close.");
     }
}

// Return state of port

void COMPort::getState () const {

     if (!GetCommState ((HANDLE)thePortHandle, (LPDCB)theDCB)) {
        throw runtime_error ("COMPort: could not retrieve serial port state.");
     }
}

// Set state of port

COMPort& COMPort::setState () {

          if (!SetCommState((HANDLE)thePortHandle, (LPDCB)theDCB)) {
               throw runtime_error ("COMPort: could not modify serial port state.");
          }
          return *this;
```

```
}

// Set Timeouts

COMPort& COMPort::setBlockingMode (
                          unsigned long inReadInterval
                        , unsigned long inReadMultiplyer
                        , unsigned long inReadConstant) {

          COMMTIMEOUTS commTimeout;

          if (!GetCommTimeouts((HANDLE(thePortHandle)), &commTimeout)) {
             throw runtime_error ("COMPort: failed to retrieve timeouts.");
          }

          commTimeout.ReadIntervalTimeout = inReadInterval;

          if ( inReadInterval==MAXDWORD ) {
             commTimeout.ReadTotalTimeoutMultiplier = 0;
             commTimeout.ReadTotalTimeoutConstant = 0;
          }
          else {
               commTimeout.ReadTotalTimeoutMultiplier = inReadMultiplyer;
               commTimeout.ReadTotalTimeoutConstant = inReadConstant;
          }

          if (!SetCommTimeouts((HANDLE(thePortHandle)), &commTimeout)) {
             throw runtime_error ("COMPort: failed to modify timeouts.");
          }
          return *this;
}

// Set/Unset serial control

COMPort & COMPort::setHandshaking (bool inHandshaking) {

         DCB & aDCB = *((LPDCB)theDCB);
         if (inHandshaking) {
            aDCB.fOutxCtsFlow = TRUE;
            aDCB.fOutxDsrFlow = FALSE;
            aDCB.fRtsControl = RTS_CONTROL_HANDSHAKE;
         }
         else {
              aDCB.fOutxCtsFlow = FALSE;
              aDCB.fOutxDsrFlow = FALSE;
              aDCB.fRtsControl = RTS_CONTROL_ENABLE;
         }
         return setState();
}

// Return port's bit rate

unsigned long COMPort::getBitRate(void) {

         DCB & aDCB = *((LPDCB)theDCB);
         return aDCB.BaudRate;
```

```
}

// Set port's bit rate

void COMPort::setBitRate(BitRate br) {

     DCB & aDCB = *((LPDCB)theDCB);
     aDCB.BaudRate = br;
}

// Return port's data bits

DataBits COMPort::getDataBits(void) {

          DCB & aDCB = *((LPDCB)theDCB);
          return (DataBits)aDCB.ByteSize;
}

// Set port's data bits

void COMPort::setDataBits(DataBits db) {

     DCB & aDCB = *((LPDCB)theDCB);
     aDCB.ByteSize = db;
}

// Return port's parity

Parity COMPort::getParity(void) {

        DCB & aDCB = *((LPDCB)theDCB);
        return (Parity)aDCB.Parity;
}

// Set port's parity

void COMPort::setParity(Parity par) {

     DCB & aDCB = *((LPDCB)theDCB);
     aDCB.Parity = par;
}

// Return port's stop bits

StopBits COMPort::getStopBits(void) {

          DCB & aDCB = *((LPDCB)theDCB);
          return (StopBits)aDCB.StopBits;
}

// Set port's stop bits

void COMPort::setStopBits(StopBits sb){

     DCB & aDCB = *((LPDCB)theDCB);
     aDCB.StopBits = sb;
```

```
}

// Read a character

unsigned char COMPort::read(void) {

     unsigned char buffer;
     DWORD charsRead = 0;

     do {
          if (!ReadFile ((HANDLE(thePortHandle))
                    , &buffer
                    , sizeof(unsigned char)
                    , &charsRead
                    , NULL
                    )) {
                       throw runtime_error ("COMPort: read failed.");
                    }
     } while (!charsRead);

     return  buffer;
}

// Write a character

void COMPort::write(unsigned char ch) {

     char buffer = ch;
     DWORD charsWritten = 0;

     if (!WriteFile((HANDLE(thePortHandle))
                , &buffer
                , sizeof(char)
                , &charsWritten
                , NULL
                )) {
                   throw runtime_error ("COMPort: write failed.");
                }
}

// Read 'nch' bytes from port and store them in 'buffer'

unsigned long COMPort::read(unsigned char *buffer, int nch) {

     DWORD charsRead = 0;
     if (!ReadFile((HANDLE(thePortHandle))
                , buffer
                , nch
                , &charsRead
                , NULL
                )) {
                   throw runtime_error ("COMPort: read failed.");
                }

     return charsRead;
}
```

C.2. Capa 2: Mensajes y Comandos

Packet.h

```
/*
=============================================================================
  Name: Packet.h
  Author: Emilio García Fidalgo
  Date: 09/10/06 19:02
  Description: Abstract class Packet
=============================================================================
*/

#include "SonarDefs.h"
#include <stdexcept>
#include <iostream>
#include <iomanip>
#include <cstdio>

using namespace std;

#ifndef _PACKET_H
#define _PACKET_H

class Packet {

      protected:
            BYTE hdr;           // Character header
            BYTE hexln[4];      // Packet's length in ASCII
            BYTE binln[2];      // Packet's length in binary
            BYTE sid;           // Source ID
            BYTE did;           // Destination ID
            BYTE countmsg;      // Length in bytes of MSG
            BYTE ptype;         // Packet's type
            BYTE sequence;      // Sequence number (for multi-packet mode)
            BYTE node;          // Copy of byte 8 / 9
            BYTE lfeed;         // Packet temination's character

            virtual void printPacket(bool) = 0;

      public:
            // Access functions

            BYTE getHdr(void);
            BYTE *getHexLn(void);
            BYTE *getBinLn(void);
            BYTE getSid(void);
            BYTE getDid(void);
            BYTE getCountMsg(void);
            BYTE getType(void);
```

```
                BYTE getSequence(void);
                BYTE getNode(void);
                friend class MiniKing;
};

#endif
```

Packet.cpp

```
/*
=============================================================================
  Name: Packet.h
  Author: Emilio García Fidalgo
  Date: 26/10/06 16:57
  Description: Abstract class Packet
=============================================================================
*/

#include "Packet.h"

BYTE Packet::getHdr(void) { return hdr; };

BYTE *Packet::getHexLn(void) { return hexln; };

BYTE *Packet::getBinLn(void) { return binln; };

BYTE Packet::getSid(void) { return sid; };

BYTE Packet::getDid(void) { return did; };

BYTE Packet::getCountMsg(void) { return countmsg; };

BYTE Packet::getType(void) { return ptype; };

BYTE Packet::getSequence(void) { return sequence; };

BYTE Packet::getNode(void) { return node; };
```

Command.h

```
/*
=============================================================================
  Name: Command.h
  Author: Emilio García Fidalgo
  Date: 09/10/06 19:22
  Description: Abstract class Command
=============================================================================
*/

#ifndef _COMMAND_H
#define _COMMAND_H
```

```
#include "Packet.h"
#include "COMPort.h"

class Command : public Packet {

      protected:
                void base256(BYTE *, int, int);            // Pass an integer to Hexadecimal
                virtual void send(COMPort *) = 0;          // Send through a Serial port
};

#endif
```

Command.cpp

```
/*
===========================================================================
  Name: Command.cpp
  Author: Emilio García Fidalgo
  Date: 11/10/06 19:46
  Description: Implementation of the 'base 256' function
===========================================================================
*/

#include "Command.h"

// Pass 'number' to Hexadecimal and store it to 'r'

void Command::base256(BYTE *r, int number, int howmany) {

     int remainder;
     int quotient = number;
     BYTE *p;

     p = r;
     for (int i = 0; i < howmany; i++) {
         remainder = quotient % 256;
         quotient = quotient / 256;
         if (quotient || remainder)
             *p = remainder;
         else
             *p = 0;
         p++;
    }
}
```

Message.h

```
/*
===========================================================================
  Name: Message.h
```

```
   Author: Emilio García Fidalgo
   Date: 09/10/06 19:44
   Description: Abstract class Message
============================================================================
*/

#ifndef _MESSAGE_H
#define _MESSAGE_H

#include "Packet.h"
#include "COMPort.h"

class Message : public Packet {

      protected:
                int base10(BYTE *, int);                // Pass an Hexadecimal to integer
                virtual void receive(COMPort *) = 0;    // Receive from a Serial port
};

#endif
```

Message.cpp

```
/*
============================================================================
   Name: Message.cpp
   Author: Emilio García Fidalgo
   Date: 09/10/06 19:44
   Description: Implementation of the 'synchronize' and 'base10' functions
============================================================================
*/

#include "Message.h"
#include <cmath>

// Pass an Hexadecimal to Integer

int Message::base10(BYTE *buffer, int howmany) {

     int result = 0;
     BYTE *p = buffer;

     for (int i = 0; i < howmany; i++) {
         result += (*p) * (int)pow(256.0, i);
         p++;
    }
    return result;
}
```

C.2.1. Comandos

MtHeadCommand.h

```
/*
=============================================================================
   Name: MtHeadCommand.h
   Author: Emilio García Fidalgo
   Date: 11/10/06 19:02
   Description: Implementation of 'mtHeadCommand' command
=============================================================================
*/

#ifndef _MTHEADCOMMAND_H
#define _MTHEADCOMMAND_H

#include "Command.h"

// This structure stores all parameter information

typedef struct {

        BYTE hdctrl[2];             // Controlling head operation
        BYTE hdtype;                // Device or Head type. Default: Imaging Sonar
        BYTE tx1[4];                // Channel 1 Transmit Frequency
        BYTE tx2[4];                // Channel 2 Transmit Frequency
        BYTE rx1[4];                // Channel 1 Receive Frecuency
        BYTE rx2[4];                // Channel 2 Receive Frecuency
        BYTE txpulselen[2];         // Length of the sonar trasmit pulse
        BYTE rangescale[2];         // Maximun distance
        BYTE leftlim[2];            // Left Limit
        BYTE rightlim[2];           // Right Limit
        BYTE adspan;                // Control the mapping of the received sonar echo amplitude
        BYTE adlow;                 // Control the mapping of the received sonar echo amplitude
        BYTE igain1;                // Initial Gain of the receiver, Channel 1
        BYTE igain2;                // Initial Gain of the receiver, Channel 2
        BYTE slope1[2];             // Compensation in channel 1
        BYTE slope2[2];             // Compensation in channel 2
        BYTE motime;                // High speed limit of scanning motor in unit of 10 nanosecs
        BYTE stepsize;              // Sonar Resolution
        BYTE adinterval[2];         // Define the sampling interval of each Range Bin
        BYTE nbins[2];              // Number of bin to create response packet
        BYTE maxadbuff[2];          // Default Value
        BYTE lockout[2];            // Blankig Time
        BYTE minoraxis[2];          // Default Value
        BYTE majaxispan;            // Default Value
        BYTE ctl2;                  // Additional sonar control functions
        BYTE scanz[2];              // For special devices
} HeadParameterInfo;

class MtHeadCommand : public Command {

      protected:
                BYTE mtheadcommandtype;    // Specifies wheter packet includes a V3B Block
                HeadParameterInfo hpi;     // Structure that contains all parameters
```

```
                Config *cf;                 // Configuration to send

        public:
                MtHeadCommand(Config *);
                void printPacket(bool);
                void send(COMPort *);
                void setConfig(Config *); // Sets the 'Config' structure where get the data
};

#endif
```

MtHeadCommand.cpp

```
/*
=============================================================================
  Name: MtHeadCommand.cpp
  Author: Emilio García Fidalgo
  Date: 12/10/06 15:29
  Description: Implementation of 'mtHeadCommand' command
=============================================================================
*/

#include "MtHeadCommand.h"
#include <cmath>

// Constructor

MtHeadCommand::MtHeadCommand(Config *cfg) {

        // Header

      hdr = INITCHAR;

      // Hexadecimal length

      hexln[0] = 0x30;
      hexln[1] = 0x30;
      hexln[2] = 0x33;
      hexln[3] = 0x43;

      // Binary length

      binln[0] = 0x3C;
      binln[1] = 0x00;

      // Addresses

      sid = PCADDR;
      did = SNADDR;

      // Message Field Count

      countmsg = 0x37;
```

```
        // Message Field

        ptype = mtHeadCommand;
        sequence = 0x80;
        node = SNADDR;

        // Head Command type

        mtheadcommandtype = 0x01;

        // Configuration structure

        cf = cfg;

        // Footer

        lfeed = ENDCHAR;
}

// Shows information contained in packet´s fields

void MtHeadCommand::printPacket(bool complete) {

        cout << endl << "--mtHeadCommand--␣Packet" << endl << endl;

        if (complete) {
                printf("Header:␣%02X\n", hdr);
                printf("Hex␣Length:␣%02X␣%02X␣%02X␣%02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
                printf("Bin␣Length:␣%02X␣%02X\n", binln[0], binln[1]);
                printf("SID:␣%02X\n", sid);
                printf("DID:␣%02X\n", did);
                printf("Count␣MSG:␣%02X\n", countmsg);
                printf("Packet␣Type:␣%02X\n", ptype);
                printf("Sequence:␣%02X\n", sequence);
                printf("Node:␣%02X\n", node);
                printf("Head␣Command␣Type:␣%02X\n\n", mtheadcommandtype);
                printf("Hdctrl:␣%02X␣%02X\n", hpi.hdctrl[0], hpi.hdctrl[1]);
                printf("HdType:␣%02X\n", hpi.hdtype);
                printf("Tx␣Ch1:␣%02X␣%02X␣%02X␣%02X\n", hpi.tx1[0], hpi.tx1[1], hpi.tx1[2], hpi.tx1[3]);
                printf("Tx␣Ch2:␣%02X␣%02X␣%02X␣%02X\n", hpi.tx2[0], hpi.tx2[1], hpi.tx2[2], hpi.tx2[3]);
                printf("Rx␣Ch1:␣%02X␣%02X␣%02X␣%02X\n", hpi.rx1[0], hpi.rx1[1], hpi.rx1[2], hpi.rx1[3]);
                printf("Rx␣Ch2:␣%02X␣%02X␣%02X␣%02X\n", hpi.rx2[0], hpi.rx2[1], hpi.rx2[2], hpi.rx2[3]);
                printf("Pulse␣Length:␣%02X␣%02X\n", hpi.txpulselen[0], hpi.txpulselen[1]);
                printf("Range␣Scale:␣%02X␣%02X\n", hpi.rangescale[0], hpi.rangescale[1]);
                printf("Left␣Lim:␣%02X␣%02X\n", hpi.leftlim[0], hpi.leftlim[1]);
                printf("Right␣Lim:␣%02X␣%02X\n", hpi.rightlim[0], hpi.rightlim[1]);
                printf("ADSpan:␣%02X\n", hpi.adspan);
                printf("ADLow:␣%02X\n", hpi.adlow);
                printf("Gain␣1:␣%02X\n", hpi.igain1);
                printf("Gain␣2:␣%02X\n", hpi.igain2);
                printf("Slope␣Ch1:␣%02X␣%02X\n", hpi.slope1[0], hpi.slope1[1]);
                printf("Slope␣Ch2:␣%02X␣%02X\n", hpi.slope2[0], hpi.slope2[1]);
                printf("MoTime:␣%02X\n", hpi.motime);
                printf("Step␣Size:␣%02X\n", hpi.stepsize);
                printf("ADInterval:␣%02X␣%02X\n", hpi.adinterval[0], hpi.adinterval[1]);
```

```
            printf("NBins: %02X %02X\n", hpi.nbins[0], hpi.nbins[1]);
            printf("MAX AD Buff: %02X %02X\n", hpi.maxadbuff[0], hpi.maxadbuff[1]);
            printf("Lockout: %02X %02X\n", hpi.lockout[0], hpi.lockout[1]);
            printf("Minor Axis Dir: %02X %02X\n", hpi.minoraxis[0], hpi.minoraxis[1]);
            printf("Major Axis Pan: %02X\n", hpi.majaxispan);
            printf("Ctl2: %02X\n", hpi.ctl2);
            printf("ScanZ: %02X %02X\n", hpi.scanz[0], hpi.scanz[1]);

            cout << endl << "--Line Feed--" << endl;
        }
}

// Rounds a double to integer

int roundNumber(double number) {

    return int(number + 0.5);
}

void MtHeadCommand::send(COMPort *cport) {

     BYTE frame[66];

     // Header

     frame[0] = hdr;

     // Hexadecimal length

     frame[1] = hexln[0];
     frame[2] = hexln[1];
     frame[3] = hexln[2];
     frame[4] = hexln[3];

     // Binary length

     frame[5] = binln[0];
     frame[6] = binln[1];

     // Addresses

     frame[7] = sid;
     frame[8] = did;

     // Message Field Count

     frame[9] = countmsg;

     // Message Field

     frame[10] = ptype;
     frame[11] = sequence;
     frame[12] = node;

     // MtHeadCommand Type
```

```
frame[13] = mtheadcommandtype;

// Calculate 'hdtrl' value

int total = 0;
for (int i = 0; i < 16; i++)
    total += cf->hdctrlbits[i] * (int)pow(2.0,i);

base256(&frame[14], total, 2);
base256(hpi.hdctrl, total, 2);

// Sonar Type

frame[16] = hpi.hdtype = cf->sonartype;

// TxN and RxN parameters

double constant = pow(2.0,32) / 32e6;

int txn = roundNumber(cf->frequency * 1000 * constant);
int rxn = roundNumber(((cf->frequency * 1000) + 455000) * constant);

base256(&frame[17], txn, 4);
base256(&frame[21], txn, 4);
base256(&frame[25], rxn, 4);
base256(&frame[29], rxn, 4);

base256(hpi.tx1, txn, 4);
base256(hpi.tx2, txn, 4);
base256(hpi.rx1, rxn, 4);
base256(hpi.rx2, rxn, 4);

// Tx Pulse Length Value

int plen = ((cf->rangescale + 10) * 25) / 10;

base256(&frame[33], plen, 2);

base256(hpi.txpulselen, plen, 2);

// Range Scale

base256(&frame[35], cf->rangescale * 10, 2);

base256(hpi.rangescale, cf->rangescale * 10, 2);

// LeftLim and RighLim

base256(&frame[37], cf->leftlim * 16, 2);

base256(hpi.leftlim, cf->leftlim * 16, 2);

base256(&frame[39], cf->rightlim * 16, 2);

base256(hpi.rightlim, cf->rightlim * 16, 2);
```

```
// ADSpan and ADLow

BYTE adsp = roundNumber(cf->adspan * 255) / 80;
BYTE adlw = roundNumber(cf->adlow * 255) / 80;

frame[41] = adsp;
frame[42] = adlw;

hpi.adspan = adsp;
hpi.adlow  = adlw;

// Initial Gain

BYTE gn = roundNumber((cf->gain * 210) / 100);

frame[43] = gn;
frame[44] = gn;

hpi.igain1 = gn;
hpi.igain2 = gn;

// Slope Settings

int slp;
switch (cf->frequency) {
        case f200:
              slp = 70;
              break;
        case f325:
              slp = 90;
              break;
        case f580:
              slp = 110;
              break;
        case f675:
              slp = 125;
              break;
        case f795:
              slp = 130;
              break;
        case f935:
              slp = 140;
              break;
        case f1210:
              slp = 150;
              break;
        case f2000:
              slp = 180;
              break;
        default:
              slp = 0;
              break;
}

base256(&frame[45], slp, 2);
base256(&frame[47], slp, 2);
```

```
base256(hpi.slope1, slp, 2);
base256(hpi.slope2, slp, 2);

// MoTime

frame[49] = 0x19;
hpi.motime = 0x19;

// Resolution

frame[50] = cf->resolution;
hpi.stepsize = cf->resolution;

// ADInterval

double adint = (cf->rangescale * 2.0) / 1500.0;
adint /= cf->bins;
adint /= 640e-9;

base256(&frame[51], roundNumber(adint), 2);
base256(hpi.adinterval, roundNumber(adint), 2);

// NBins

base256(&frame[53], cf->bins, 2);
base256(hpi.nbins, cf->bins, 2);

// MaxADBuff

base256(&frame[55], 1000, 2);
base256(hpi.maxadbuff, 1000, 2);

// Lockout

base256(&frame[57], 100, 2);
base256(hpi.lockout, 100, 2);

// Minor Axis

base256(&frame[59], 1600, 2);
base256(hpi.minoraxis, 1600, 2);

// Major Axis Pan

frame[61] = 0x01;
hpi.majaxispan = 0x01;

// Ctrl2

frame[62] = 0;
hpi.ctl2 = 0;

// ScanZ

base256(&frame[63], 0, 2);
```

```
        base256(hpi.scanz, 0 ,2);

        // Footer

        frame[65] = lfeed;

        for (int i = 0; i < 66; i++)
            cport->write(frame[i]);
}

// Changes configuration structure associated with MtHeadCommand

void MtHeadCommand::setConfig(Config *cfg) { cf = cfg; }
```

MtReboot.h

```
/*
=============================================================================
  Name: MtReboot.h
  Author: Emilio García Fidalgo
  Date: 11/10/06 17:50
  Description: Implementation of 'mtReboot' command
=============================================================================
*/

#ifndef _MTREBOOT_H
#define _MTREBOOT_H

#include "Command.h"

class MtReboot : public Command {

      public:
            MtReboot();
            void printPacket(bool);
            void send(COMPort *);
};

#endif
```

MtReboot.cpp

```
/*
=============================================================================
  Name: MtReboot.cpp
  Author: Emilio García Fidalgo
  Date: 11/10/06 17:52
  Description: Implementation of 'mtReboot' command
=============================================================================
*/
```

```
#include "MtReboot.h"

// Constructor

MtReboot::MtReboot() {

       // Header

      hdr = INITCHAR;

      // Hexadecimal length

      hexln[0] = 0x30;
      hexln[1] = 0x30;
      hexln[2] = 0x30;
      hexln[3] = 0x38;

      // Binary length

      binln[0] = 0x08;
      binln[1] = 0x00;

      // Addresses

      sid = PCADDR;
      did = SNADDR;

      // Message Field Count

      countmsg = 0x03;

      // Message Field

      ptype = mtReboot;
      sequence = 0x80;
      node = SNADDR;

      // Footer

      lfeed = ENDCHAR;
}

// Shows information contained in packet´s fields

void MtReboot::printPacket(bool complete) {

      cout << endl << "--mtReboot-- Packet" << endl << endl;

      if (complete) {
            printf("Header: %02X\n", hdr);
            printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
            printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
            printf("SID: %02X\n", sid);
            printf("DID: %02X\n", did);
            printf("Count MSG: %02X\n", countmsg);
            printf("Packet Type: %02X\n", ptype);
```

```
            printf("Sequence: %02X\n", sequence);
            printf("Node: %02X\n", node);

            cout << endl << "--Line Feed--" << endl;
        }
}

// Sends an mtSendReboot packet through 'cport'

void MtReboot::send(COMPort *cport) {

    BYTE frame[14];

    // Header

    frame[0] = hdr;

    // Hexadecimal length

    frame[1] = hexln[0];
    frame[2] = hexln[1];
    frame[3] = hexln[2];
    frame[4] = hexln[3];

    // Binary length

    frame[5] = binln[0];
    frame[6] = binln[1];

    // Addresses

    frame[7] = sid;
    frame[8] = did;

    // Message Field Count

    frame[9] = countmsg;

    // Message Field

    frame[10] = ptype;
    frame[11] = sequence;
    frame[12] = node;

    // Footer

    frame[13] = lfeed;

    for (int i = 0; i < 14; i++)
        cport->write(frame[i]);
}
```

MtSendBBUser.h

```
/*
===========================================================================
  Name: MtSendBBUser.h
  Author: Emilio García Fidalgo
  Date: 10/10/06 20:34
  Description: Implementation of 'mtSendBBUser' command
===========================================================================
*/

#ifndef _MTSENDBBUSER_H
#define _MTSENDBBUSER_H

#include "Command.h"

class MtSendBBUser : public Command {

      public:
            MtSendBBUser();
            void printPacket(bool);
            void send(COMPort *);
};

#endif
```

MtSendBBUser.cpp

```
/*
===========================================================================
  Name: MtSendBBUser.cpp
  Author: Emilio García Fidalgo
  Date: 11/10/06 17:48
  Description: Implementation of 'mtSendBBUser' command
===========================================================================
*/

#include "MtSendBBUser.h"

// Constructor

MtSendBBUser::MtSendBBUser() {

       // Header

      hdr = INITCHAR;

      // Hexadecimal length

      hexln[0] = 0x30;
      hexln[1] = 0x30;
      hexln[2] = 0x30;
      hexln[3] = 0x38;

      // Binary length
```

```
        binln[0] = 0x08;
        binln[1] = 0x00;

        // Addresses

        sid = PCADDR;
        did = SNADDR;

        // Message Field Count

        countmsg = 0x03;

        // Message Field

        ptype = mtSendBBUser;
        sequence = 0x80;
        node = SNADDR;

        // Footer

        lfeed = ENDCHAR;
}

// Shows information contained in packet´s fields

void MtSendBBUser::printPacket(bool complete) {

        cout << endl << "--mtSendBBUser-- Packet" << endl << endl;

        if (complete) {
                printf("Header: %02X\n", hdr);
                printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
                printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
                printf("SID: %02X\n", sid);
                printf("DID: %02X\n", did);
                printf("Count MSG: %02X\n", countmsg);
                printf("Packet Type: %02X\n", ptype);
                printf("Sequence: %02X\n", sequence);
                printf("Node: %02X\n", node);

                cout << endl << "--Line Feed--" << endl;
        }
}

// Sends an mtSendBBUser packet through 'cport'

void MtSendBBUser::send(COMPort *cport) {

        BYTE frame[14];

        // Header

        frame[0] = hdr;

        // Hexadecimal length
```

```
    frame[1] = hexln[0];
    frame[2] = hexln[1];
    frame[3] = hexln[2];
    frame[4] = hexln[3];

    // Binary length

    frame[5] = binln[0];
    frame[6] = binln[1];

    // Addresses

    frame[7] = sid;
    frame[8] = did;

    // Message Field Count

    frame[9] = countmsg;

    // Message Field

    frame[10] = ptype;
    frame[11] = sequence;
    frame[12] = node;

    // Footer

    frame[13] = lfeed;

    for (int i = 0; i < 14; i++)
        cport->write(frame[i]);
}
```

MtSendData.h

```
/*
============================================================================
  Name: MtSendData.h
  Author: Emilio García Fidalgo
  Date: 11/10/06 18:23
  Description: Implementation of 'mtSendData' command
============================================================================
*/

#ifndef _MTSENDDATA_H
#define _MTSENDDATA_H

#include "Command.h"

class MtSendData : public Command {

    public:
```

```
            MtSendData();
            void printPacket(bool);
            void send(COMPort *);
};

#endif
```

MtSendData.cpp

```
/*
=============================================================================
  Name: MtSendData.cpp
  Author: Emilio García Fidalgo
  Date: 11/10/06 18:34
  Description: Implementation of 'mtSendData' command
=============================================================================
*/

#include "MtSendData.h"
#include <ctime>

// Constructor

MtSendData::MtSendData() {

      // Header

     hdr = INITCHAR;

     // Hexadecimal length

     hexln[0] = 0x30;
     hexln[1] = 0x30;
     hexln[2] = 0x30;
     hexln[3] = 0x43;

     // Binary length

     binln[0] = 0x0C;
     binln[1] = 0x00;

     // Addresses

     sid = PCADDR;
     did = SNADDR;

     // Message Field Count

     countmsg = 0x07;

     // Message Field

     ptype = mtSendData;
```

```
        sequence = 0x80;
        node = SNADDR;

        // Footer

        lfeed = ENDCHAR;
}

// Shows information contained in packet´s fields

void MtSendData::printPacket(bool complete) {

        cout << endl << "--mtSendData-- Packet" << endl << endl;

        if (complete) {
                printf("Header: %02X\n", hdr);
                printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
                printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
                printf("SID: %02X\n", sid);
                printf("DID: %02X\n", did);
                printf("Count MSG: %02X\n", countmsg);
                printf("Packet Type: %02X\n", ptype);
                printf("Sequence: %02X\n", sequence);
                printf("Node: %02X\n", node);

                cout << endl << "--Line Feed--" << endl;
        }
}

// Sends an mtSendData packet through 'cport'

void MtSendData::send(COMPort *cport) {

        BYTE frame[18];

        // Header

        frame[0] = hdr;

        // Hexadecimal length

        frame[1] = hexln[0];
        frame[2] = hexln[1];
        frame[3] = hexln[2];
        frame[4] = hexln[3];

        // Binary length

        frame[5] = binln[0];
        frame[6] = binln[1];

        // Addresses

        frame[7] = sid;
        frame[8] = did;
```

```
        // Message Field Count

        frame[9] = countmsg;

        // Message Field

        frame[10] = ptype;
        frame[11] = sequence;
        frame[12] = node;

        // Calculate local time in miliseconds

        time_t tt;
        struct tm *thetime;

        tt = time(NULL);
        thetime = localtime(&tt);

        int milisecs = ((thetime->tm_hour * 3600) + (thetime->tm_min * 60) + (thetime->tm_sec)) * 1000;

        // Pass this number to Base 256 and store it in 'frame'

        base256(&frame[13], milisecs, 4);

        // Footer

        frame[17] = lfeed;

        for (int i = 0; i < 18; i++)
            cport->write(frame[i]);
}
```

MtSendVersion.h

```
/*
=============================================================================
  Name: MtSendVersion.h
  Author: Emilio García Fidalgo
  Date: 10/10/06 18:29
  Description: Implementation of 'mtSendVersion' command
=============================================================================
*/

#ifndef _MTSENDVERSION_H
#define _MTSENDVERSION_H

#include "Command.h"

class MtSendVersion : public Command {

    public:
        MtSendVersion();
        void printPacket(bool);
```

```
            void send(COMPort *);
};

#endif
```

MtSendVersion.cpp

```
/*
=====================================================================
  Name: MtSendVersion.cpp
  Author: Emilio García Fidalgo
  Date: 10/10/06 18:58
  Description: Implementation of 'mtSendVersion' command
=====================================================================
*/

#include "MtSendVersion.h"

// Constructor

MtSendVersion::MtSendVersion() {

      // Header

     hdr = INITCHAR;

     // Hexadecimal length

     hexln[0] = 0x30;
     hexln[1] = 0x30;
     hexln[2] = 0x30;
     hexln[3] = 0x38;

     // Binary length

     binln[0] = 0x08;
     binln[1] = 0x00;

     // Addresses

     sid = PCADDR;
     did = SNADDR;

     // Message Field Count

     countmsg = 0x03;

     // Message Field

     ptype = mtSendVersion;
     sequence = 0x80;
     node = SNADDR;
```

```
      // Footer

      lfeed = ENDCHAR;
}

// Shows information contained in packet´s fields

void MtSendVersion::printPacket(bool complete) {

      cout << endl << "--mtSendVersion-- Packet" << endl << endl;

      if (complete) {
            printf("Header: %02X\n", hdr);
            printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
            printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
            printf("SID: %02X\n", sid);
            printf("DID: %02X\n", did);
            printf("Count MSG: %02X\n", countmsg);
            printf("Packet Type: %02X\n", ptype);
            printf("Sequence: %02X\n", sequence);
            printf("Node: %02X\n", node);

            cout << endl << "--Line Feed--" << endl;
      }
}

// Sends an mtSendVersion packet through 'cport'

void MtSendVersion::send(COMPort *cport) {

      BYTE frame[14];

      // Header

      frame[0] = hdr;

      // Hexadecimal length

      frame[1] = hexln[0];
      frame[2] = hexln[1];
      frame[3] = hexln[2];
      frame[4] = hexln[3];

      // Binary length

      frame[5] = binln[0];
      frame[6] = binln[1];

      // Addresses

      frame[7] = sid;
      frame[8] = did;

      // Message Field Count

      frame[9] = countmsg;
```

```
        // Message Field

        frame[10] = ptype;
        frame[11] = sequence;
        frame[12] = node;

        // Footer

        frame[13] = lfeed;

        for (int i = 0; i < 14; i++)
            cport->write(frame[i]);
}
```

C.2.2. Mensajes

MtAlive.h

```
/*
=============================================================================
  Name: MtAlive.h
  Author: Emilio García Fidalgo
  Date: 13/10/06 18:53
  Description: Implementation of 'mtAlive' Message
=============================================================================
*/

#ifndef _MTALIVE_H
#define _MTALIVE_H

#include "Message.h"

class MtAlive : public Message {

    protected:
            BYTE willsend;              // The device has params or not
            BYTE headtime[4];           // Time in ms since head power on
            BYTE motorpos[2];           // Current position in 1/16 Gradians
            BYTE headinf;               // Information about sonar
    public:
            void printPacket(bool);
            void receive(COMPort *);

            // Access Functions

            bool hasParams(void);       // Returns true if Sonar had params
            int getHeadTime(void);      // Time in sonar
            int getMotorPos(void);      // Motor position

            // Hdctrl info

            bool isCentering(void);     // Returns true if Sonar is in centering operation
```

```
                bool isCentred(void);       // Returns true if Sonar is in centre position
                bool isMotoring(void);      // Returns true if Sonar´s motor is in moving
                bool isMotorOn(void);       // Returns true if Sonar´s motor is on
                bool isDir(void);           // Returns true if transducer is off centre
                bool isScan(void);          // Returns true if Sonar is in a scanning operation
                bool noParams(void);        // Returns true if Sonar hasn´t got params
                bool paramsReceived(void);  // Returns true if params has been received by the sonar
};

#endif
```

MtAlive.cpp

```
/*
=============================================================================
  Name: MtAlive.cpp
  Author: Emilio García Fidalgo
  Date: 13/10/06 19:04
  Description: Implementation of 'mtAlive' Message
=============================================================================
*/

#include "MtAlive.h"

// Shows information contained in packet´s fields

void MtAlive::printPacket(bool complete) {

     cout << endl << "--mtAlive--␣Packet" << endl << endl;

     if (complete) {
          printf("Header:␣%02X\n", hdr);
          printf("Hex␣Length:␣%02X␣%02X␣%02X␣%02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
          printf("Bin␣Length:␣%02X␣%02X\n", binln[0], binln[1]);
          printf("SID:␣%02X\n", sid);
          printf("DID:␣%02X\n", did);
          printf("Count␣MSG:␣%02X\n", countmsg);
          printf("Packet␣Type:␣%02X\n", ptype);
          printf("Sequence:␣%02X\n", sequence);
          printf("Node:␣%02X\n", node);
          printf("Params:␣%02X\n", willsend);
          printf("Head␣Time:␣%02X␣%02X␣%02X␣%02X\n", headtime[0],
          headtime[1], headtime[2], headtime[3]);
          printf("Motor␣Pos:␣%02X␣%02X\n", motorpos[0], motorpos[1]);
          printf("Head␣Info:␣%02X\n", headinf);

          cout << endl << "--Line␣Feed--" << endl;
     }
}

// Receive and store 'MtAlive' Message

void MtAlive::receive(COMPort *cport) {
```

```
            try {

                // Params

                willsend = cport->read();

                // Head Time

                cport->read(headtime, 4);

                // Motor Position

                cport->read(motorpos, 2);

                // Information about Motor

                headinf = cport->read();

                // Line Feed

                lfeed = cport->read();
            }
            catch (runtime_error &e) {          // Error in timeouts or bytes
                  cout << e.what() << endl;
            }
}

bool MtAlive::hasParams(void) {

     if (willsend == 0)
        return true;
     else
         return false;
}

int MtAlive::getHeadTime(void) {

    return base10(headtime, 4);
}

int MtAlive::getMotorPos(void) {

    return base10(motorpos, 2);
}

bool MtAlive::isCentering(void) {

     if ((headinf & 0x01) == 0)
        return false;
     else
        return true;
}

bool MtAlive::isCentred(void) {
```

```
    if ((headinf & 0x02) == 0)
       return false;
    else
       return true;
}

bool MtAlive::isMotoring(void) {

    if ((headinf & 0x04) == 0)
       return false;
    else
       return true;
}

bool MtAlive::isMotorOn(void) {

    if ((headinf & 0x08) == 0)
       return false;
    else
       return true;
}

bool MtAlive::isDir(void) {

    if ((headinf & 0x10) == 0)
       return false;
    else
       return true;
}

bool MtAlive::isScan(void) {

    if ((headinf & 0x20) == 0)
       return false;
    else
       return true;
}

bool MtAlive::noParams(void) {

    if ((headinf & 0x40) == 0)
       return false;
    else
       return true;
}

bool MtAlive::paramsReceived(void) {

    if ((headinf & 0x80) == 0)
       return false;
    else
       return true;
}
```

MtBBUserData.h

```
/*
=============================================================================
  Name: MtBBUserData.h
  Author: Emilio García Fidalgo
  Date: 14/10/06 13:46
  Description: Implementation of 'mtBBUserData' Message
=============================================================================
*/

#ifndef _MTBBUSERDATA_H
#define _MTBBUSERDATA_H

#include "Message.h"

class MtBBUserData : public Message {

      protected:
                SystemBBBlock  syb;          // General features
                BaudBBBlock    bbb;          // Telemetry settings
                SonarBlock     snb;          // Sonar Profiler Block
                friend class MiniKing;       // To access SonarBlock
      public:
                void printPacket(bool);
                void receive(COMPort *);

                // Access Functions

                // System

                int getHeadClock(void);
                int getBlocksLength(void);

                // Telemetry

                   // LAN

                LANBitRate getLanBaudLow(void);
                LANBitRate getLanBaudHigh(void);

                LANSensitivity getSensitivityLow(void);
                LANSensitivity getSensitivityHigh(void);

                int getLanTimeout(void);

                    // RS232 Port

                BitRate getA0BaudLow(void);
                BitRate getA0BaudHigh(void);

                Parity getA0ParityLow(void);
                Parity getA0ParityHigh(void);

                DataBits getA0DataBitsLow(void);
                DataBits getA0DataBitsHigh(void);
```

```
                    // AUX Port

                BitRate getA1BaudLow(void);
                BitRate getA1BaudHigh(void);

                Parity getA1ParityLow(void);
                Parity getA1ParityHigh(void);

                DataBits getA1DataBitsLow(void);
                DataBits getA1DataBitsHigh(void);

                // General Sonar Options

                bool hasMotor(void);
                bool hasAux(void);
                bool hasRollSensor(void);
};

#endif
```

MtBBUserData.cpp

```
/*
=============================================================================
  Name: MtBBUserData.cpp
  Author: Emilio García Fidalgo
  Date: 15/10/06 1:04
  Description: Implementation of 'MtBBUserData' Message
=============================================================================
*/

#include "MtBBUserData.h"

// Shows information contained in packet´s fields

void MtBBUserData::printPacket(bool complete) {

     cout << endl << "--mtBBUserData-- Packet" << endl << endl;

     if (complete) {
          printf("Header: %02X\n", hdr);
          printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
          printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
          printf("SID: %02X\n", sid);
          printf("DID: %02X\n", did);
          printf("Count MSG: %02X\n", countmsg);
          printf("Packet Type: %02X\n", ptype);
          printf("Sequence: %02X\n", sequence);
          printf("Node: %02X\n", node);
          printf("CK: %02X %02X\n", syb.ck[0], syb.ck[1]);
          printf("UserBBLen: %02X %02X\n\n", syb.userbbln[0], syb.userbbln[1]);
          printf("RS232 Baud Low: %02X %02X\n", bbb.a0baudlow[0], bbb.a0baudlow[1]);
```

```
            printf("RS232 Baud High: %02X %02X\n", bbb.a0baudhigh[0], bbb.a0baudhigh[1]);
            printf("RS232 Parity Low: %02X %02X\n", bbb.a0paritylow[0], bbb.a0paritylow[1]);
            printf("RS232 Parity High: %02X %02X\n", bbb.a0parityhigh[0], bbb.a0parityhigh[1]);
            printf("RS232 Data Bits Low: %02X %02X\n", bbb.a0databitslow[0], bbb.a0databitslow[1]);
            printf("RS232 Data Bits High: %02X %02X\n", bbb.a0databitshigh[0], bbb.a0databitshigh[1]);
            printf("Has Motor: %02X\n", snb.hasmotor);
            printf("Has Aux: %02X\n", snb.hasaux);
            printf("Has Roll Sensor: %02X\n", snb.hasrollsensor);

            cout << endl << "--Line Feed--" << endl;
        }
}

// Receive and store 'MtBBUserData' Message

void MtBBUserData::receive(COMPort *cport) {

            try {

                // System BB Block

                syb.surfacectrl = cport->read();
                syb.d1 = cport->read();
                cport->read(syb.ck,2);
                cport->read(syb.userbbln,2);

                for (int i = 0; i < 6; i++)
                    cport->read();

                // Baud Block

                cport->read(bbb.lanbaudlow, 2);
                cport->read(bbb.lanbaudhigh, 2);
                cport->read(bbb.lansensitivitylow, 2);
                cport->read(bbb.lansensitivityhigh, 2);
                cport->read(bbb.lantimeout, 2);

                cport->read(bbb.a0baudlow, 2);
                cport->read(bbb.a0baudhigh, 2);
                cport->read(bbb.a0paritylow, 2);
                cport->read(bbb.a0parityhigh, 2);
                cport->read(bbb.a0databitslow, 2);
                cport->read(bbb.a0databitshigh, 2);

                cport->read(bbb.a1baudlow, 2);
                cport->read(bbb.a1baudhigh, 2);
                cport->read(bbb.a1paritylow, 2);
                cport->read(bbb.a1parityhigh, 2);
                cport->read(bbb.a1databitslow, 2);
                cport->read(bbb.a1databitshigh, 2);

                // Sonar Block

                snb.hasmotor = cport->read();
                snb.d1 = cport->read();
                snb.hasaux = cport->read();
```

```
snb.d2 = cport->read();
snb.hasrollsensor = cport->read();
snb.d3 = cport->read();

cport->read(snb.rollmultiplier, 2);
cport->read(snb.rolldivisor, 2);
cport->read(snb.rolloffset, 2);

snb.haspresure = cport->read();
snb.d4 = cport->read();

cport->read(snb.pressmultiplier, 2);
cport->read(snb.pressdivisor, 2);
cport->read(snb.pressoffset, 2);

cport->read(snb.tempmultiplier, 2);
cport->read(snb.tempdivisor, 2);
cport->read(snb.tempoffset, 2);

snb.fireboth = cport->read();
snb.d5 = cport->read();
snb.emulatev18 = cport->read();
snb.d6 = cport->read();
snb.boardrevision = cport->read();
snb.d7 = cport->read();
cport->read(snb.emulatev18node, 2);
snb.aux1sec = cport->read();
snb.d10 = cport->read();
snb.dualaxis = cport->read();
snb.d11 = cport->read();
snb.compassstabilised = cport->read();
snb.d12 = cport->read();
snb.rs485 = cport->read();
snb.d13 = cport->read();

cport->read(snb.tx1, 2);
cport->read(snb.tx2, 2);
cport->read(snb.rx1, 2);
cport->read(snb.rx2, 2);
cport->read(snb.slope1, 2);
cport->read(snb.slope2, 2);
cport->read(snb.prftimecal1, 2);
cport->read(snb.prftimecal2, 2);
cport->read(snb.sontimecal1, 2);
cport->read(snb.sontimecal2, 2);

snb.v4mode = cport->read();
snb.d14 = cport->read();

cport->read(snb.motorpanconstant, 2);
cport->read(snb.maxspeedpan, 2);
cport->read(snb.mechllpan, 2);
cport->read(snb.mechrlpan, 2);

cport->read(snb.motortiltconstant, 2);
cport->read(snb.maxspeedtilt, 2);
```

```
                cport->read(snb.mechlltilt, 2);
                cport->read(snb.mechrltilt, 2);

                cport->read(snb.txdcr1chan1, 12);
                cport->read(snb.txdcr1chan2, 12);
                cport->read(snb.txdcr2chan1, 12);
                cport->read(snb.txdcr2chan2, 12);

                cport->read(snb.elscancode, 2);

                snb.dualsynth = cport->read();
                snb.d15 = cport->read();

                cport->read(snb.specials, 8);

                for (int i = 0; i < 64; i++)
                    cport->read();

                // Line Feed

                lfeed = cport->read();
            }
            catch (runtime_error &e) {          // Error in timeouts or bytes
                  cout << e.what() << endl;
            }
}

int MtBBUserData::getHeadClock(void) { return base10(syb.ck, 2); }
int MtBBUserData::getBlocksLength(void) { return base10(syb.userbbln, 2); }

// Telemetry

LANBitRate MtBBUserData::getLanBaudLow(void)
{ return (LANBitRate)base10(bbb.lanbaudlow, 2); }
LANBitRate MtBBUserData::getLanBaudHigh(void)
{ return (LANBitRate)base10(bbb.lanbaudhigh, 2); }

LANSensitivity MtBBUserData::getSensitivityLow(void)
{ return (LANSensitivity)base10(bbb.lansensitivitylow, 2); }
LANSensitivity MtBBUserData::getSensitivityHigh(void)
{ return (LANSensitivity)base10(bbb.lansensitivityhigh, 2); }

int MtBBUserData::getLanTimeout(void) { return base10(bbb.lantimeout, 2); }

BitRate MtBBUserData::getA0BaudLow(void){

        switch (base10(bbb.a0baudlow, 2)) {
                case 0x01:
                        return br2400;
                case 0x02:
                        return br4800;
                case 0x03:
                        return br9600;
                case 0x04:
                        return br19200;
                case 0x05:
```

```
                        return br38400;
                case 0x06:
                        return br57600;
                case 0x07:
                        return br115200;
                default:
                        return brdefault;
        }
}

BitRate MtBBUserData::getA0BaudHigh(void){

        switch (base10(bbb.a0baudhigh, 2)) {
                case 0x01:
                        return br2400;
                case 0x02:
                        return br4800;
                case 0x03:
                        return br9600;
                case 0x04:
                        return br19200;
                case 0x05:
                        return br38400;
                case 0x06:
                        return br57600;
                case 0x07:
                        return br115200;
                default:
                        return brdefault;
        }
}

Parity MtBBUserData::getA0ParityLow(void)
{ return (Parity)base10(bbb.a0paritylow, 2); }
Parity MtBBUserData::getA0ParityHigh(void)
{ return (Parity)base10(bbb.a0parityhigh, 2); }

DataBits MtBBUserData::getA0DataBitsLow(void){

          if (base10(bbb.a0databitslow, 2) == 0)
              return db8;
          else
              return db7;
}

DataBits MtBBUserData::getA0DataBitsHigh(void){

          if (base10(bbb.a0databitshigh, 2) == 0)
              return db8;
          else
              return db7;
}

BitRate MtBBUserData::getA1BaudLow(void){

        switch (base10(bbb.a1baudlow, 2)) {
```

```
                case 0x01:
                        return br2400;
                case 0x02:
                        return br4800;
                case 0x03:
                        return br9600;
                case 0x04:
                        return br19200;
                case 0x05:
                        return br38400;
                case 0x06:
                        return br57600;
                case 0x07:
                        return br115200;
                default:
                        return brdefault;
        }
}

BitRate MtBBUserData::getA1BaudHigh(void){

        switch (base10(bbb.a1baudhigh, 2)) {
                case 0x01:
                        return br2400;
                case 0x02:
                        return br4800;
                case 0x03:
                        return br9600;
                case 0x04:
                        return br19200;
                case 0x05:
                        return br38400;
                case 0x06:
                        return br57600;
                case 0x07:
                        return br115200;
                default:
                        return brdefault;
        }
}

Parity MtBBUserData::getA1ParityLow(void)
{ return (Parity)base10(bbb.a1paritylow, 2); }
Parity MtBBUserData::getA1ParityHigh(void)
{ return (Parity)base10(bbb.a1parityhigh, 2); }

DataBits MtBBUserData::getA1DataBitsLow(void) {

          if (base10(bbb.a1databitslow, 2) == 0)
              return db8;
          else
              return db7;
}

DataBits MtBBUserData::getA1DataBitsHigh(void){
```

```
        if (base10(bbb.a1databitshigh, 2) == 0)
            return db8;
        else
            return db7;
}

// General Sonar Options

bool MtBBUserData::hasMotor(void){ return snb.hasmotor; }
bool MtBBUserData::hasAux(void) { return snb.hasaux; }
bool MtBBUserData::hasRollSensor(void) { return snb.hasrollsensor; }
```

MtHeadData.h

```
/*
=============================================================================
  Name: MtHeadData.h
  Author: Emilio García Fidalgo
  Date: 15/10/06 13:21
  Description: Implementation of 'mtHeadData' Message
=============================================================================
*/

#ifndef _MTHEADDATA_H
#define _MTHEADDATA_H

// Device Params

#include "Message.h"

typedef struct {

        BYTE totalbytecount[2];
        BYTE devicetype;
        BYTE headstatus;
        BYTE sweepcode;
        BYTE hdctrl[2];
        BYTE rangescale[2];
        BYTE txntransmitter[4];
        BYTE gainsetting;
        BYTE slopesetting[2];
        BYTE adspan;
        BYTE adlow;
        BYTE headingoffset[2];
        BYTE adinterval[2];
        BYTE leftlimit[2];
        BYTE rightlimit[2];
        BYTE stepangle;
        BYTE bearing[2];
        BYTE dbytes[2];
} DevicePars;

class MtHeadData : public Message {
```

```
    protected:
            DevicePars dp;                  // Copy of Sonar's Params
            BYTE *rd;                       // Data
    public:
            MtHeadData();
            void printPacket(bool);
            void receive(COMPort *);

            // Access Functions

            SweepCode getSweepCode(void);   // Return Sweep Code about scan
            int getBearing(void);           // Return Sonar's orientation
            int getDataLength(void);        // Return Number of Information Bytes
            BYTE *getDataBytes(void);       // Return Bytes of Data
};

#endif
```

MtHeadData.cpp

```
/*
========================================================================
  Name: MtHeadData.cpp
  Author: Emilio García Fidalgo
  Date: 15/10/06 14:02
  Description: Implementation of 'mtHeadData' Message
========================================================================
*/

#include "MtHeadData.h"

// Constructor

MtHeadData::MtHeadData() : rd(NULL) {}

// Shows information contained in packet's fields

void MtHeadData::printPacket(bool complete) {

    cout << endl << "--mtHeadData-- Packet" << endl << endl;

    if (complete) {
        printf("Header: %02X\n", hdr);
        printf("Hex Length: %02X %02X %02X %02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
        printf("Bin Length: %02X %02X\n", binln[0], binln[1]);
        printf("SID: %02X\n", sid);
        printf("DID: %02X\n", did);
        printf("Start Seq: %02X\n", countmsg);
        printf("Packet Type: %02X\n", ptype);
        printf("Sequence: %02X\n", sequence);
        printf("Node: %02X\n\n", node);
        printf("Total Byte Count: %02X %02X\n", dp.totalbytecount[0], dp.totalbytecount[1]);
```

```
            printf("Head Type: %02X\n", dp.devicetype);
            printf("Head Status: %02X\n", dp.headstatus);
            printf("Sweep Code: %02X\n", dp.sweepcode);
            printf("Head Type: %02X\n", dp.devicetype);
            printf("Hdctrl: %02X %02X\n", dp.hdctrl[0], dp.hdctrl[1]);
            printf("Range Scale: %02X %02X\n", dp.rangescale[0], dp.rangescale[1]);
            printf("TxN: %02X %02X %02X %02X\n",
            dp.txntransmitter[0], dp.txntransmitter[1],
            dp.txntransmitter[2], dp.txntransmitter[3]);
            printf("Gain: %02X\n", dp.gainsetting);
            printf("Slope: %02X %02X\n", dp.slopesetting[0], dp.slopesetting[1]);
            printf("AdSpan: %02X\n", dp.adspan);
            printf("ADLow: %02X\n", dp.adlow);
            printf("Heading OffSet: %02X %02X\n", dp.headingoffset[0], dp.headingoffset[1]);
            printf("Left Limit: %02X %02X\n", dp.leftlimit[0], dp.leftlimit[1]);
            printf("Right Limit: %02X %02X\n", dp.rightlimit[0], dp.rightlimit[1]);
            printf("Steps: %02X\n", dp.stepangle);
            printf("Bearing: %02X %02X\n", dp.bearing[0], dp.bearing[1]);
            printf("DBytes: %02X %02X\n\n", dp.dbytes[0], dp.dbytes[1]);

            cout << "Data: " << endl;

            int cnt = base10(dp.dbytes, 2);
            for (int i = 0; i < cnt; i++)
                printf("%02X ", *(rd + i));

            cout << endl << endl << "--Line Feed--" << endl;
        }
}

// Receive and store 'MtHeadData' Message

void MtHeadData::receive(COMPort *cport) {

        try {

            // Params info

            cport->read(dp.totalbytecount, 2);
            dp.devicetype = cport->read();
            dp.headstatus = cport->read();
            dp.sweepcode = cport->read();

            cport->read(dp.hdctrl, 2);
            cport->read(dp.rangescale, 2);
            cport->read(dp.txntransmitter, 4);
            dp.gainsetting = cport->read();
            cport->read(dp.slopesetting, 2);
            dp.adspan = cport->read();
            dp.adlow = cport->read();
            cport->read(dp.headingoffset, 2);
            cport->read(dp.adinterval, 2);
            cport->read(dp.leftlimit, 2);
            cport->read(dp.rightlimit, 2);
            dp.stepangle = cport->read();
            cport->read(dp.bearing, 2);
```

```
            cport->read(dp.dbytes, 2);

            // Data Bytes

            int cnt = base10(dp.dbytes, 2);

            if (rd != NULL)
               delete [] rd;

            rd = new BYTE[cnt];
            for (int i = 0; i < cnt; i++)
                rd[i] = cport->read();

            // Line Feed

            lfeed = cport->read();
        }
        catch (runtime_error &e) {          // Error in timeouts or bytes
              cout << e.what() << endl;
        }
}

SweepCode MtHeadData::getSweepCode(void) { return (SweepCode)dp.sweepcode; }

int MtHeadData::getBearing(void) { return base10(dp.bearing, 2); }

int MtHeadData::getDataLength(void) { return base10(dp.dbytes, 2); }

BYTE *MtHeadData::getDataBytes(void) { return rd; }
```

MtVersionData.h

```
/*
=============================================================================
  Name: MtVersionData.h
  Author: Emilio García Fidalgo
  Date: 13/10/06 17:05
  Description: Implementation of 'mtVersionData' Message
=============================================================================
*/

#ifndef _MTVERSIONDATA_H
#define _MTVERSIONDATA_H

#include "Message.h"

// CPUID fields

typedef struct {

        BYTE frequencies;
        BYTE boardtype;
        BYTE serialnumber[2];
```

```
} CPUID;

class MtVersionData : public Message {

      protected:
                CPUID cpuid;          // Information about Sonar
                BYTE programln[4];    // Version
                BYTE checksum[2];     // Checksum
                BYTE nodeid;          // Node ID
      public:
                void printPacket(bool);
                void receive(COMPort *);

                // Access Functions

                Frequency getFrequency(void); // Returns Packet Frequency
                BoardType getBoardType(void); // Return BoardType
                int getSerialNumber(void);    // Returns CPU Serial Number in sonar
                int getProgramLength(void);   // Returns Program Length
                int getChecksum(void);        // Returns Checksum
};

#endif
```

MtVersionData.cpp

```
/*
=============================================================================
  Name: MtVersionData.cpp
  Author: Emilio García Fidalgo
  Date: 13/10/06 17:55
  Description: Implementation of 'mtVersionData' Message
=============================================================================
*/

#include "MtVersionData.h"

// Shows information contained in packet´s fields

void MtVersionData::printPacket(bool complete) {

     cout << endl << "--mtVersionData--␣Packet" << endl << endl;

     if (complete) {
          printf("Header:␣%02X\n", hdr);
          printf("Hex␣Length:␣%02X␣%02X␣%02X␣%02X\n", hexln[0], hexln[1], hexln[2], hexln[3]);
          printf("Bin␣Length:␣%02X␣%02X\n", binln[0], binln[1]);
          printf("SID:␣%02X\n", sid);
          printf("DID:␣%02X\n", did);
          printf("Count␣MSG:␣%02X\n", countmsg);
          printf("Packet␣Type:␣%02X\n", ptype);
          printf("Sequence:␣%02X\n", sequence);
          printf("Node:␣%02X\n", node);
```

```
            printf("CPUID Frequencies: %02X\n", cpuid.frequencies);
            printf("CPUID BoardType: %02X\n", cpuid.boardtype);
            printf("CPUID Serial Number: %02X %02X\n",
            cpuid.serialnumber[0], cpuid.serialnumber[1]);
            printf("Program Legth: %02X %02X %02X %02X\n",
            programln[0], programln[1], programln[2], programln[3]);
            printf("Checksum: %02X %02X\n", checksum[0], checksum[1]);
            printf("Node ID: %02X\n", nodeid);

            cout << endl << "--Line Feed--" << endl;
      }
}

// Receive and store 'MtVersionData' Message

void MtVersionData::receive(COMPort *cport) {

          try {

               cpuid.frequencies = cport->read();

               cpuid.boardtype = cport->read();

               cport->read(cpuid.serialnumber, 2);

               // Program Length

               cport->read(programln, 4);

               // Checksum

               cport->read(checksum, 2);

               // Node ID

               nodeid = cport->read();

               // Line Feed

               lfeed = cport->read();
          }
          catch (runtime_error &e) {          // Error in timeouts or bytes
                cout << e.what() << endl;
          }
}

// Analizes 'frequency' BYTE and returns enumerate type

Frequency MtVersionData::getFrequency(void) {

          switch(cpuid.frequencies) {
                case 0x01:
                     return f325;
                case 0x02:
                     return f580;
                case 0x03:
```

```
                    return f675;
                case 0x04:
                    return f795;
                case 0x05:
                    return f935;
                case 0x06:
                    return f1210;
                case 0x07:
                    return f200;
                case 0x08:
                    return f1700;
                case 0x09:
                    return f2000;
                case 0x0A:
                    return f500;
                case 0x0B:
                    return f1500;
                case 0x0C:
                    return f295;
                    default:
                        return f0;
        }
}

// Analizes 'BoardType' BYTE and returns enumerate

BoardType MtVersionData::getBoardType(void) {

        BYTE mask = 0x0F;
        BYTE res = cpuid.boardtype & mask;

        switch (res) {
            case 0:
                return AIF;
            case 0x01:
                return Sonar;
            case 0x02:
                return Bathy;
            default:
                return Others;
        }
}

int MtVersionData::getSerialNumber(void) { return base10(cpuid.serialnumber, 2); }

int MtVersionData::getProgramLength(void) { return base10(programln, 4); }

int MtVersionData::getChecksum(void) { return base10(checksum, 2); }
```

C.3. Capa 3: Miniking

MiniKing.h

```
/*
=============================================================================
  Name: MiniKing.h
  Author: Emilio García Fidalgo
  Date: 30/10/06 16:40
  Description: Main Library class
=============================================================================

Default Configuration for Sonar:
        Imaging Sonar
        Frequency: 675 Khz
        Range Scale: 30 Metres
        Scan Limits: Left = 0 ; Right = 0 (in gradians)
        ADSpan = 12 Db
        ADLow = 13 Db
        Initial Gain: 40%
        Resolution: Medium
        Bins: 300
        8 Data Bits
        Scan continuous
        Clockwise
        Normal mounted
        Motor Enabled
        Transmitter Enabled
        Only One Channel Operation
        Sonar has scanning motor
        Do not apply offsets
        Not pingpong, only one channel
        Don´t stare in a fixed direction
        Don´t ignore sensor
*/

#ifndef _MINIKING_H
#define _MINIKING_H

#include "SonarDefs.h"
#include "COMPort.h"
#include "IncludePackets.h"
#include "Message.h"
#include "Command.h"

class MiniKing {

      private:
                COMPort cp;             // RS232 Sonar Communication
                Config conf;            // Configuration sonar parameters
                VersionData vd;         // Version Data Information
                bool needData;          // Determine if it´s needed send an MtSendData command
                bool verbose;           // Shows all information about receiving and sending packets
                bool allReceived;       // To receive data in head1 or head2
```

```
        // Packets

        // Commands

        MtHeadCommand mtheadcommand;
        MtReboot mtreboot;
        MtSendData mtsenddata;
        MtSendVersion mtsendversion;
        MtSendBBUser mtsendbbuser;

        // Messages

        MtVersionData mtversiondata;
        MtAlive mtalive;
        MtBBUserData mtbbuserdata;
        MtHeadData mtheaddata1;
        MtHeadData mtheaddata2;
        PacketType receive(COMPort *cp);       // General receive

public:
        MiniKing(char *, int);              // Constructor

        // Configuration Access functions

        void set8Bits(bool);                // Data bits of responses
        bool get8Bits(void);

        void setContinuous(bool);           // Sector Scan or Continuous
        bool getContinuous(void);

        void setInverted(bool);             // When sonar head is mounted inverted
        bool getInverted(void);

        void setMotorDisabled(bool);        // Disable motor
        bool getMotorDisabled(void);

        void setTransmitDisabled(bool); // Disable sonar trasmitter
        bool getTransmitDisabled(void);

        void setApplyOffset(bool);          // Direction is dinamically modified
        bool getApplyOffset(void);

        void setStare(bool);                // Stare in a fixed direction, marked as 'LeftLim'
        bool getStare(void);

        void setSonarType(SonarType);       // Imaging, Sidescan or Profiling
        SonarType getSonarType(void);

        void setFrequency(Frequency);       // Frequency in Khz
        Frequency getFrequency(void);

        void setRange(int);                 // Sets Range Scale value in metres
        int getRange(void);

        void setLeftLim(int);               // Sets Left Limit for a Sector Scan
        float getLeftLim(void);
```

```
void setRightLim(int);          // Sets Right Limit for a Sector Scan
float getRightLim(void);

void setADSpan(int);            // Parameter in DB
int getADSpan(void);            // Parameter in DB

void setADLow(int);             // Parameter in DB
int getADLow(void);             // Parameter in DB

void setGain(int);              // Set Initial gain head. In percentage (%)
int getGain(void);

void setResolution(Resolution); // Head Step Angle Size
Resolution getResolution(void);

void setBins(int);              // Number of Bins for each ping
int getBins(void);

// Version data access functions

BoardType getBoardType(void);   // Return BoardType
int getSerialNumber(void);      // Returns CPU Serial Number in sonar
int getProgramLength(void);     // Returns Program Length
int getChecksum(void);          // Returns Checksum

// 'MtAlive' access functions

bool hasParams(void);
int getHeadTime(void);
float getMotorPos(void);          // In degrees
bool isCentering(void);
bool isCentred(void);
bool isMotoring(void);
bool isMotorOn(void);
bool isDir(void);
bool isScan(void);
bool configReceived(void);

// 'MtBBUserData' access functions

    // LAN

LANBitRate getLanBaudLow(void);
LANBitRate getLanBaudHigh(void);

LANSensitivity getSensitivityLow(void);
LANSensitivity getSensitivityHigh(void);

int getLanTimeout(void);

    // RS232 Telemetry

BitRate getCOMBaudLow(void);
BitRate getCOMBaudHigh(void);
```

```
        Parity getCOMParityLow(void);
        Parity getCOMParityHigh(void);

        DataBits getCOMDataBitsLow(void);
        DataBits getCOMDataBitsHigh(void);

            // AUX Telemetry

        BitRate getAUXBaudLow(void);
        BitRate getAUXBaudHigh(void);

        Parity getAUXParityLow(void);
        Parity getAUXParityHigh(void);

        DataBits getAUXDataBitsLow(void);
        DataBits getAUXDataBitsHigh(void);

        bool hasMotor(void);
        bool hasAux(void);
        bool hasRollSensor(void);

        SonarBlock *otherInfo(void);          // Access to another information of Sonar

        // 'MtHeadData access functions

        float getPosition(void);              // Return Sonar's orientation
        int getDataLength(void);              // Return Number of Information Bytes

        // General

        void setDefaultConfig(void);          // Set sonar parametres to default
        void initSonar(void);                 // Basic initialization procedure
        void reboot(void);                    // Reboots sonar
        void updateConfig(void);              // Update sonar's parametres
        BYTE *getScanLine(void);              // Get a data scan
        void setVerboseMode(bool);            // Enables / Disables verbose mode

        bool isLeftLimit(void);               // Returns true if position is the left limit
        bool isRightLimit(void);              // Returns true if position is the right limit
};

#endif
```

MiniKing.cpp

```
/*
============================================================================
  Name: MiniKing.h
  Author: Emilio García Fidalgo
  Date: 30/10/06 18:22
  Description: Main Library class
============================================================================
*/
```

```
#include "MiniKing.h"

MiniKing::MiniKing(char *port, int timeout) :
                   cp(port, br115200, db8, None, sb1, timeout, false),
                   mtheadcommand(&conf),
                   mtreboot(),
                   mtsenddata(),
                   mtsendversion(),
                   mtsendbbuser(),
                   mtversiondata(),
                   mtalive(),
                   mtbbuserdata(),
                   mtheaddata1(),
                   mtheaddata2() {

      // Default Settings

      setDefaultConfig();

      // Need send 'MtSendData' command

      needData = true;

      // To receive mtHeadData Packets

      allReceived = true;

      // Verbose Mode

      verbose = false;
}

// Configuration access functions

void MiniKing::set8Bits(bool op) { conf.hdctrlbits[0] = op; }

bool MiniKing::get8Bits(void) { return conf.hdctrlbits[0]; }

void MiniKing::setContinuous(bool op) { conf.hdctrlbits[1] = op; }

bool MiniKing::getContinuous(void) { return conf.hdctrlbits[1]; }

void MiniKing::setInverted(bool op) { conf.hdctrlbits[3] = op; }

bool MiniKing::getInverted(void) { return conf.hdctrlbits[3]; }

void MiniKing::setMotorDisabled(bool op) { conf.hdctrlbits[4] = op; }

bool MiniKing::getMotorDisabled(void) { return conf.hdctrlbits[4]; }

void MiniKing::setTransmitDisabled(bool op) { conf.hdctrlbits[5] = op; }

bool MiniKing::getTransmitDisabled(void) { return conf.hdctrlbits[5]; }

void MiniKing::setApplyOffset(bool op) { conf.hdctrlbits[10] = op;}
```

```
bool MiniKing::getApplyOffset(void) { return conf.hdctrlbits[10]; }

void MiniKing::setStare(bool op) { conf.hdctrlbits[12] = op;}

bool MiniKing::getStare(void) { return conf.hdctrlbits[12]; }

void MiniKing::setSonarType(SonarType st) { conf.sonartype = st; }

SonarType MiniKing::getSonarType(void) { return conf.sonartype; }

void MiniKing::setFrequency(Frequency freq) { conf.frequency = freq; }

Frequency MiniKing::getFrequency(void) { return conf.frequency; }

void MiniKing::setRange(int range) { conf.rangescale = range; }

int MiniKing::getRange(void) { return conf.rangescale; }

void MiniKing::setLeftLim(int lim) { conf.leftlim = (lim * 10) / 9; }

float MiniKing::getLeftLim(void) { return (conf.leftlim * 9.0) / 10.0; }

void MiniKing::setRightLim(int lim) { conf.rightlim = (lim * 10) / 9; }

float MiniKing::getRightLim(void) { return (conf.rightlim * 9.0) / 10.0; }

void MiniKing::setADSpan(int ads) { conf.adspan = ads; }

int MiniKing::getADSpan(void) { return conf.adspan; }

void MiniKing::setADLow(int adl) { conf.adlow = adl; }

int MiniKing::getADLow(void) { return conf.adlow; }

void MiniKing::setGain(int gn) { conf.gain = gn; }

int MiniKing::getGain(void) { return conf.gain; }

void MiniKing::setResolution(Resolution rs) { conf.resolution = rs; }

Resolution MiniKing::getResolution(void) { return conf.resolution; }

void MiniKing::setBins(int bn) { conf.bins = bn; }

int MiniKing::getBins(void) { return conf.bins; }

// Version data access functions

BoardType MiniKing::getBoardType(void) { return vd.bt; }

int MiniKing::getSerialNumber(void) { return vd.serialnumber; }

int MiniKing::getProgramLength(void) { return vd.programlength; }

int MiniKing::getChecksum(void) { return vd.checksum; }
```

```
// 'MtAlive' access functions

bool MiniKing::hasParams(void) { return mtalive.hasParams(); }

int MiniKing::getHeadTime(void) { return mtalive.getHeadTime(); }

float MiniKing::getMotorPos(void)
{ return (((mtalive.getMotorPos() / 16.0) * 9.0) / 10.0); } // In grads

bool MiniKing::isCentering(void) { return mtalive.isCentering(); }

bool MiniKing::isCentred(void) { return mtalive.isCentred(); }

bool MiniKing::isMotoring(void) { return mtalive.isMotoring(); }

bool MiniKing::isMotorOn(void) { return mtalive.isMotorOn(); }

bool MiniKing::isDir(void) { return mtalive.isDir(); }

bool MiniKing::isScan(void) { return mtalive.isScan(); }

bool MiniKing::configReceived(void) { return mtalive.paramsReceived(); }

// 'MtBBUserData' access functions

LANBitRate MiniKing::getLanBaudLow(void) { return mtbbuserdata.getLanBaudLow(); }

LANBitRate MiniKing::getLanBaudHigh(void) { return mtbbuserdata.getLanBaudHigh(); }

LANSensitivity MiniKing::getSensitivityLow(void) { return mtbbuserdata.getSensitivityLow(); }

LANSensitivity MiniKing::getSensitivityHigh(void) { return mtbbuserdata.getSensitivityHigh(); }

int MiniKing::getLanTimeout(void) { return mtbbuserdata.getLanTimeout(); }

BitRate MiniKing::getCOMBaudLow(void) { return mtbbuserdata.getA0BaudLow(); }

BitRate MiniKing::getCOMBaudHigh(void) { return mtbbuserdata.getA0BaudHigh(); }

Parity MiniKing::getCOMParityLow(void) { return mtbbuserdata.getA0ParityLow(); }

Parity MiniKing::getCOMParityHigh(void) { return mtbbuserdata.getA0ParityHigh(); }

DataBits MiniKing::getCOMDataBitsLow(void) { return mtbbuserdata.getA0DataBitsLow(); }

DataBits MiniKing::getCOMDataBitsHigh(void) { return mtbbuserdata.getA0DataBitsHigh(); }

BitRate MiniKing::getAUXBaudLow(void) { return mtbbuserdata.getA1BaudLow(); }

BitRate MiniKing::getAUXBaudHigh(void) { return mtbbuserdata.getA1BaudHigh(); }

Parity MiniKing::getAUXParityLow(void) { return mtbbuserdata.getA1ParityLow(); }

Parity MiniKing::getAUXParityHigh(void) { return mtbbuserdata.getA1ParityHigh(); }
```

```
DataBits MiniKing::getAUXDataBitsLow(void) { return mtbbuserdata.getA1DataBitsLow(); }

DataBits MiniKing::getAUXDataBitsHigh(void) { return mtbbuserdata.getA1DataBitsHigh(); }

bool MiniKing::hasMotor(void) { return mtbbuserdata.hasMotor(); }

bool MiniKing::hasAux(void) { return mtbbuserdata.hasAux(); }

bool MiniKing::hasRollSensor(void) { return mtbbuserdata.hasRollSensor(); }

SonarBlock *MiniKing::otherInfo(void) { return &(mtbbuserdata.snb); }

// 'MtHeadData' access functions

float MiniKing::getPosition(void) {

    if (needData)
        return ((((float)mtheaddata2.getBearing() / 16.0) * 9.0) / 10.0);       // In grads
    else
        return ((((float)mtheaddata1.getBearing() / 16.0) * 9.0) / 10.0);       // In grads
}

int MiniKing::getDataLength(void) {

    if (needData)
        return mtheaddata2.getDataLength();
    else
        return mtheaddata1.getDataLength();
}

// General functions

void MiniKing::setDefaultConfig() { // Need updateConfig() after to take effect

     // Hdctrl bits

     conf.hdctrlbits[0] = true;     // 8 Data Bits
     conf.hdctrlbits[1] = true;     // Scan continuous
     conf.hdctrlbits[2] = true;     // Clockwise
     conf.hdctrlbits[3] = false;    // Normal mounted
     conf.hdctrlbits[4] = false;    // Motor Enabled
     conf.hdctrlbits[5] = false;    // Transmitter Enabled
     conf.hdctrlbits[6] = false;    // Default, Always 0
     conf.hdctrlbits[7] = false;    // Only One Channel Operation
     conf.hdctrlbits[8] = true;     // Default, Always 1
     conf.hdctrlbits[9] = true;     // Sonar has scanning motor
     conf.hdctrlbits[10] = false;   // Do not apply offsets
     conf.hdctrlbits[11] = false;   // Not pingpong, only one channel
     conf.hdctrlbits[12] = false;   // Don't stare in a fixed direction
     conf.hdctrlbits[13] = true;    // Default, Always 1
     conf.hdctrlbits[14] = false;   // Default, Always 0
     conf.hdctrlbits[15] = false;   // Don't ignore sensor

     // Head Type

     conf.sonartype = ImagingSonar;
```

```
        // Head Frequency

        conf.frequency = f675;

        // Range Scale

        conf.rangescale = 30;

        // Scan Limits

        conf.leftlim = 0;
        conf.rightlim = 0;

        // ADSpan and ADLow (in Db's)

        conf.adspan = 12;
        conf.adlow = 13;

        // Initial Gain

        conf.gain = 40;

        // Resolution

        conf.resolution = Medium;

        // Number of Range Bins for each ping

        conf.bins = 300;
}

void MiniKing::initSonar(void) {        // Basic initialization procedure

        while (receive(&cp) != mtAlive);

        mtsendversion.send(&cp);
        if (verbose) mtsendversion.printPacket(true);

        while (receive(&cp) != mtVersionData);

        vd.fr = mtversiondata.getFrequency();
        vd.bt = mtversiondata.getBoardType();
        vd.serialnumber = mtversiondata.getSerialNumber();
        vd.programlength = mtversiondata.getProgramLength();
        vd.checksum = mtversiondata.getChecksum();

        mtsendbbuser.send(&cp);
        if (verbose) mtsendversion.printPacket(true);

        while (receive(&cp) != mtBBUserData);

        while (receive(&cp) != mtAlive);

        while (!hasParams()) {
```

```
            mtheadcommand.send(&cp);
            if (verbose) mtheadcommand.printPacket(true);
            while (receive(&cp) != mtAlive);
        }
}

void MiniKing::reboot(void) {            // Reboots sonar

        setDefaultConfig();

        mtreboot.send(&cp);
        if (verbose) mtreboot.printPacket(true);
        while (receive(&cp) != mtAlive);

        while (hasParams()) receive(&cp);

        while (!hasParams()) {

            mtheadcommand.send(&cp);
            if (verbose) mtheadcommand.printPacket(true);

            while (receive(&cp) != mtAlive);
        }
}

void MiniKing::updateConfig(void) {      // Update sonar´s parametres

        do {

           mtheadcommand.send(&cp);
           if (verbose) mtheadcommand.printPacket(true);
           while (receive(&cp) != mtAlive);
        } while (!hasParams());
}

BYTE *MiniKing::getScanLine(void) {     // Get a data scan

        if (needData) {

            mtsenddata.send(&cp);
            if (verbose) mtsenddata.printPacket(true);

            while (receive(&cp) != mtHeadData);

            allReceived = false;

            while (receive(&cp) != mtHeadData);

            allReceived = true;

            needData = false;

            return mtheaddata1.getDataBytes();
        }
        else {
```

```
            needData = true;

            return mtheaddata2.getDataBytes();
        }
}

void MiniKing::setVerboseMode(bool op) {

        verbose = op;
}

bool MiniKing::isLeftLimit(void) {

        if (needData) {

                if (mtheaddata1.getSweepCode() == ScanLeftLimit)
                        return true;
                else
                        return false;
        }
        else {

                if (mtheaddata2.getSweepCode() == ScanLeftLimit)
                        return true;
                else
                        return false;
        }
}

bool MiniKing::isRightLimit(void) {

        if (needData) {

                if (mtheaddata1.getSweepCode() == ScanRightLimit)
                        return true;
                else
                        return false;
        }
        else {

                if (mtheaddata2.getSweepCode() == ScanRightLimit)
                        return true;
                else
                        return false;
        }
}

// Waits for Initial Character

BYTE synchronize(COMPort *cport) {

        BYTE ch;

        try {

            while((ch = cport->read()) != '@');
```

```
            return ch;
        }
        catch (runtime_error &e) {          // Error in timeouts or bytes
            cout << e.what() << endl;
            return false;
        }
}

// General receive

PacketType MiniKing::receive(COMPort *cport) {

        BYTE inchar = synchronize(cport);
        BYTE hdr;
        BYTE hexln[4];
        BYTE binln[2];
        BYTE sid;
        BYTE did;
        BYTE countmsg;
        BYTE ptype;
        BYTE sequence;
        BYTE node;

        if (!inchar) {
           cout << "Can't receive message" << endl;
        }
        else if (inchar == '@') {
             try {

                  // Header

                  hdr = inchar;

                  // Hex Length

                  cport->read(hexln, 4);

                  // Bin Length

                  cport->read(binln, 2);

                  // Source ID and Destination ID

                  sid = cport->read();
                  did = cport->read();

                  // Count Message

                  countmsg = cport->read();

                  // Packet Type

                  ptype = cport->read();

                  // Sequence Number
```

```
            sequence = cport->read();

            // Copy of Byte 8

            node = cport->read();

            Message *msg = NULL;

            switch (ptype) {
                    case mtAlive:
                         msg = &mtalive;
                         mtalive.receive(&cp);
                         break;
                    case mtBBUserData:
                         msg = &mtbbuserdata;
                         mtbbuserdata.receive(&cp);
                         break;
                    case mtVersionData:
                         msg = &mtversiondata;
                         mtversiondata.receive(&cp);
                         break;
                    case mtHeadData:
                         if (allReceived) {
                            msg = &mtheaddata1;
                            mtheaddata1.receive(&cp);
                         }
                         else {
                            msg = &mtheaddata2;
                            mtheaddata2.receive(&cp);
                         }
                         break;
                    default:
                         break;
            }
            msg->hdr = inchar;
            for (int i = 0; i < 4; i++)
                 msg->hexln[i] = hexln[i];
            for (int i = 0; i < 2; i++)
                 msg->binln[i] = binln[i];
            msg->sid = sid;
            msg->did = did;
            msg->countmsg = countmsg;
            msg->ptype = ptype;
            msg->sequence = sequence;
            msg->node = node;
            if (verbose) msg->printPacket(true);
            return (PacketType)ptype;
        }
        catch (runtime_error &e) {          // Error in timeouts or bytes
              cout << e.what() << endl;
        }
    }
    else {
        cout << "Unknown error receiving message" << endl;
    }
    return errorPacket;
```

```
}
```

Apéndice D

Código fuente del *Miniking Viewer*

MainApp.h

```
/*
=============================================================================
  Name: MainApp.h
  Author: Emilio García Fidalgo
  Date: 15/12/06 10:29
  Description: Initial wxWidgets application class
=============================================================================
*/

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP

#include <wx/wx.h>
#endif

#include "MainFrame.h"
#include "InitialDialog.h"

#ifndef MAINAPP_H
#define MAINAPP_H

class MainApp : public wxApp {

    virtual bool OnInit(void);  // Start method
};

#endif
```

MainApp.cpp

```
/*
=============================================================================
  Name: MainApp.cpp
  Author: Emilio García Fidalgo
  Date: 15/12/06 10:35
  Description: Initial wxWidgets application class
=============================================================================
*/

#include "MainApp.h"
#include "PaintThread.h"

DECLARE_APP(MainApp)

IMPLEMENT_APP(MainApp)

const int NPIX = 300;                  // Circle radio. Same as bins.

// Initial application method

bool MainApp::OnInit(void) {

    // Create new frame

    MainFrame *mframe = new MainFrame(NULL);

    PaintThread *pt = new PaintThread(mframe);

     // New thread for painting in frame

    if (pt->Create() != wxTHREAD_NO_ERROR )
    {
        wxLogError(wxT("Can't create thread!"));
    }

    // Reduces thread priority

    wxThread *thr = pt;
    thr->SetPriority(WXTHREAD_MIN_PRIORITY);

    // Assigns thread to frame

    mframe->setPaintThread(pt);

    MiniKing *mk =  mframe->getSonar();
    pt->setSonar(mk);

    wxMutex *mutex = new wxMutex();
    pt->setMutex(mutex);
    mframe->setMutex(mutex);

    // Loading dialog

    InitialDialog *dlg = new InitialDialog(mframe);
```

```
    dlg->Show();

    // Initializing Sonar

    dlg->setValue(25);

    mk->initSonar();

    dlg->setValue(50);

    // Configuring Sonar

    mk->setBins(NPIX);

    dlg->setValue(75);

    mk->updateConfig();

    dlg->setValue(100);

    dlg->Destroy();
    delete dlg;

    mframe->Show();

    // Starts the thread

    pt->Run();

    return true;
}
```

MainFrame.h

```
/*
=============================================================================
  Name: MainFrame.h
  Author: Emilio García Fidalgo
  Date: 15/12/06 11:53
  Description: MainFrame class declaration
=============================================================================
*/

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include <wx/wx.h>
        #include <wx/frame.h>
#else
        #include <wx/wxprec.h>
#endif
```

```
#include "tinyxml/tinyxml.h"
#include "tinyxml/tinystr.h"

#include "PaintThread.h"
#include "PortDialog.h"
#include <wx/spinctrl.h>
#include <wx/combobox.h>
#include <wx/slider.h>
#include <wx/filedlg.h>
#include "wx/html/htmlwin.h"
#include "wx/statline.h"
#include "wx/numdlg.h"
#include <iostream>
#include <fstream>

#ifndef MAINFRAME_H
#define MAINFRAME_H

#undef MainFrame_STYLE
#define MainFrame_STYLE wxCAPTION | wxSYSTEM_MENU | wxMINIMIZE_BOX | wxCLOSE_BOX | wxCLIP_CHILDREN

class MainFrame : public wxFrame
{
        private:

            DECLARE_EVENT_TABLE();                      // Event Table
                    PaintThread *pt;
            MiniKing *sn;
            wxMutex *mutex;
                    enum
                    {
                            ID_DUMMY_VALUE_,
                            ID_SAVECONF = wxID_HIGHEST + 1,
                            ID_LOADCONF,
                            ID_BONE,
                            ID_COOL,
                            ID_COPPER,
                            ID_FLAG,
                            ID_GRAY,
                            ID_HOT,
                            ID_HSV,
                            ID_JET,
                            ID_PINK,
                            ID_SNAPSHOT,
                            ID_PLAY,
                            ID_RECORD,
                            ID_TBSTART,
                            ID_TBSTOP,
                            ID_TBPAUSE,
                            ID_SPINRANGE,
                            ID_SPINGAIN,
                            ID_COMBOBX,
                            ID_TBCONTINUOUS,
                            ID_SPINLEFTLIM,
                            ID_SPINRIGHTLIM,
```

```
                    ID_REBOOT,
                    ID_SLIDER,
                    ID_CLEAR,
                    ID_LIMIT
    };

            void OnClose(wxCloseEvent& event);              // Responses close event
            void OnQuit(wxCommandEvent& event);             // Responses close event

            void OnSaveConf(wxCommandEvent& event);         // Save configuration event
            void OnLoadConf(wxCommandEvent& event);         // Load configuration event
            void OnLimit(wxCommandEvent& event);
            void OnReboot(wxCommandEvent& event);           // Reboot sonar event

            void OnBone(wxCommandEvent& event);             // Colour selected 1
            void OnCool(wxCommandEvent& event);             // Colour selected 2
            void OnCopper(wxCommandEvent& event);           // Colour selected 3
            void OnFlag(wxCommandEvent& event);             // Colour selected 4
            void OnGray(wxCommandEvent& event);             // Colour selected 5
            void OnHot(wxCommandEvent& event);              // Colour selected 6
            void OnHsv(wxCommandEvent& event);              // Colour selected 7
            void OnJet(wxCommandEvent& event);              // Colour selected 8
            void OnPink(wxCommandEvent& event);             // Colour selected 9

            void OnSnapshot(wxCommandEvent& event);         // Takes an Snapshot
            void OnLogPlay(wxCommandEvent& event);          // Play file
            void OnLogRecord(wxCommandEvent& event);        // Start record

            void OnHelp(wxCommandEvent& event);             // Help Event
            void OnAbout(wxCommandEvent& event);            // About Event

            void OnStart(wxCommandEvent& event);            // Start Toolbar Event
            void OnStop(wxCommandEvent& event);             // Stop Toolbar Event
            void OnPause(wxCommandEvent& event);            // Pause Toolbar Event
            void OnContinuous(wxCommandEvent& event);       // Continuous Toolbar Event
            void OnSpinRange(wxSpinEvent& event);           // Spin distance
            void OnSpinGain(wxSpinEvent& event);            // Spin Gain
            void OnComboBox(wxCommandEvent& event);         // Changing resolution
            void OnSpinLeft(wxSpinEvent& event);            // Spin left limit
            void OnSpinRight(wxSpinEvent& event);           // Spin right limit
            void OnSlide(wxCommandEvent& event);            // Threshold limit
            void OnClear(wxCommandEvent& event);            // Clear Button
            void OnMouseDClick(wxMouseEvent& event);        // Double-Click event

            void OnErase(wxEraseEvent& event);              // Erase event
            void OnPaint(wxPaintEvent& event);              // Painting event

            void CreateGUIControls();                       // Internal component creator

public:

    // Constructor

            MainFrame(wxWindow *parent, wxWindowID id = 1,
    const wxString &title = wxT("Circle Scanner"),
    const wxPoint& pos = wxDefaultPosition,
```

```
            const wxSize& size = wxDefaultSize, long style = MainFrame_STYLE);
            virtual ~MainFrame();                              // Destructor
                    void setPaintThread(PaintThread *pthread);      // Sets the PaintThread
                    void setMutex(wxMutex *);                          // Mutual Exclusion
            MiniKing *getSonar();                              // Returns Sonar Pointer
};

#endif
```

MainFrame.cpp

```
/*
=============================================================================
  Name: MainFrame.cpp
  Author: Emilio García Fidalgo
  Date: 15/12/06 12:26
  Description: MainFrame class implementation
=============================================================================
*/

#include "MainFrame.h"

#include "start.xpm"
#include "stop.xpm"
#include "pause.xpm"
#include "continuous.xpm"
#include "mkv.xpm"

// Events Table

BEGIN_EVENT_TABLE(MainFrame,wxFrame)
        EVT_CLOSE(MainFrame::OnClose)                          // Close operation
        EVT_MENU(wxID_EXIT, MainFrame::OnQuit)                 // Close operation through menu
        EVT_MENU(ID_SAVECONF, MainFrame::OnSaveConf)           // Save configuration event
        EVT_MENU(ID_LOADCONF, MainFrame::OnLoadConf)           // Load configuration event
        EVT_MENU(ID_LIMIT, MainFrame::OnLimit)                // Sets a limit scan
        EVT_MENU(ID_REBOOT, MainFrame::OnReboot)               // Reboot event
        EVT_MENU(ID_BONE, MainFrame::OnBone)                   // Colour select 1
        EVT_MENU(ID_COOL, MainFrame::OnCool)                   // Colour select 2
        EVT_MENU(ID_COPPER, MainFrame::OnCopper)               // Colour select 3
        EVT_MENU(ID_FLAG, MainFrame::OnFlag)                   // Colour select 4
        EVT_MENU(ID_GRAY, MainFrame::OnGray)                   // Colour select 5
        EVT_MENU(ID_HOT, MainFrame::OnHot)                     // Colour select 6
        EVT_MENU(ID_HSV, MainFrame::OnHsv)                     // Colour select 7
        EVT_MENU(ID_JET, MainFrame::OnJet)                     // Colour select 8
        EVT_MENU(ID_PINK, MainFrame::OnPink)                   // Colour select 9
        EVT_MENU(ID_SNAPSHOT, MainFrame::OnSnapshot)           // Snapshot
        EVT_MENU(ID_PLAY, MainFrame::OnLogPlay)                // Record stop
        EVT_MENU(ID_RECORD, MainFrame::OnLogRecord)            // Record select
        EVT_MENU(wxID_HELP, MainFrame::OnHelp)                 // Help information
        EVT_MENU(wxID_ABOUT, MainFrame::OnAbout)               // About information
        EVT_MENU(ID_TBSTART, MainFrame::OnStart)               // Toolbar start
        EVT_MENU(ID_TBSTOP, MainFrame::OnStop)                 // Toolbar stop
```

```
        EVT_MENU(ID_TBPAUSE, MainFrame::OnPause)                    // Toolbar pause
        EVT_MENU(ID_TBCONTINUOUS, MainFrame::OnContinuous)          // Toolbar continuous
        EVT_SPINCTRL(ID_SPINRANGE, MainFrame::OnSpinRange)          // Range Spin
    EVT_SPINCTRL(ID_SPINGAIN, MainFrame::OnSpinGain)              // Gain Spin
        EVT_COMBOBOX(ID_COMBOBX, MainFrame::OnComboBox)             // Change Resolution
    EVT_SPINCTRL(ID_SPINLEFTLIM, MainFrame::OnSpinLeft)           // Left limit spin
    EVT_SPINCTRL(ID_SPINRIGHTLIM, MainFrame::OnSpinRight)         // Right limit spin
        EVT_SLIDER(ID_SLIDER, MainFrame::OnSlide)                   // Threshold event
        EVT_BUTTON(ID_CLEAR, MainFrame::OnClear)                    // Clear Button
        EVT_ERASE_BACKGROUND(MainFrame::OnErase)                    // Erase event
    EVT_LEFT_DCLICK(MainFrame::OnMouseDClick)                     // Double-Click event
END_EVENT_TABLE()

// Constructor

MainFrame::MainFrame(wxWindow *parent, wxWindowID id,
const wxString &title, const wxPoint &position, const wxSize& size, long style)
: wxFrame(parent, id, title, position, size, style)
{
    pt = NULL;
        CreateGUIControls();
}

// Destructor

MainFrame::~MainFrame()
{
}

// Sets pt variable

void MainFrame::setPaintThread(PaintThread *pthread)
{

    pt = pthread;
}

// Mutual Exclusion

void MainFrame::setMutex(wxMutex *mt)
{

    mutex = mt;
}

wxToolBar *toolbar;

// General structures

wxMenuItem *snapshot;
wxMenuItem *logplay;
wxMenuItem *logrecord;

wxMenuItem *scanlimitvalue;

wxSpinCtrl *spinrange;
```

```
wxSpinCtrl *spingain;
wxSpinCtrl *spinleftlim;
wxSpinCtrl *spinrightlim;

wxComboBox *comboresol;

wxSlider* threshold;

ofstream *logdoc;

// Frame Initialization

void MainFrame::CreateGUIControls()
{

    // Selects COMPort

    PortDialog *dialog = new PortDialog(NULL);
    dialog->ShowModal();

    // Creates frame's interface

    wxIcon icon(mkv_xpm);

        SetTitle(wxT("MiniKing Viewer"));
        SetIcon(icon);
        SetSize(8,8,1024,768);

        SetBackgroundColour(wxColour(0, 0, 80));

        Center();

        // Create Menus

        wxMenuBar *menubar = new wxMenuBar();

        wxMenu *appmenu = new wxMenu;
        wxMenu *confmenu = new wxMenu;
        wxMenu *colourmenu = new wxMenu;
        wxMenu *logmenu = new wxMenu;
        wxMenu *helpmenu = new wxMenu;

        // Application Menu

        appmenu->Append(wxID_EXIT, wxT("E&xit\tCtrl+X"), wxT("Quit this program"));

        // Configuration Menu

        scanlimitvalue = new wxMenuItem(confmenu, ID_LIMIT,
    wxT("Set limit value ..."), wxT("Sets a scan limit value"));

        confmenu->Append(ID_SAVECONF, wxT("&Save configuration ...\tCtrl+S"),
    wxT("Save configuration to a file"));
        confmenu->Append(ID_LOADCONF, wxT("&Load configuration ...\tCtrl+L"),
    wxT("Load configuration from a file"));
        confmenu->AppendSeparator();
```

```
        confmenu->Append(scanlimitvalue);
        confmenu->AppendSeparator();
        confmenu->Append(ID_REBOOT, wxT("&Reboot ...\tCtrl+R"), wxT("Reboots sonar"));

        // Colour Menu

        colourmenu->AppendRadioItem(ID_BONE, wxT("Bone"));
        colourmenu->AppendRadioItem(ID_COOL, wxT("Cool"));
        colourmenu->AppendRadioItem(ID_COPPER, wxT("Copper"));
        colourmenu->AppendRadioItem(ID_FLAG, wxT("Flag"));
        colourmenu->AppendRadioItem(ID_GRAY, wxT("Gray"));
        colourmenu->AppendRadioItem(ID_HOT, wxT("Hot"));
        colourmenu->AppendRadioItem(ID_HSV, wxT("HSV"));
        colourmenu->AppendRadioItem(ID_JET, wxT("Jet"));
        colourmenu->AppendRadioItem(ID_PINK, wxT("Pink"));

        // LOG Menu

        snapshot = new wxMenuItem(logmenu, ID_SNAPSHOT,
    wxT("Sn&apshot ..."), wxT("Window Capture"));
        logplay = new wxMenuItem(logmenu, ID_PLAY, wxT("&Play ..."), wxT("Play Log"));
        logrecord = new wxMenuItem(logmenu, ID_RECORD, wxT("&Record ..."),
    wxT("Selects a file to store this session"));

        logmenu->Append(snapshot);
        logmenu->AppendSeparator();
        logmenu->Append(logplay);
        logmenu->Append(logrecord);

        // Help Menu

        helpmenu->Append(wxID_HELP, wxT("&Help"), wxT("Useful information"));
        helpmenu->Append(wxID_ABOUT, wxT("&About"), wxT("Program information"));

        // General menu bar

        menubar->Append(appmenu, wxT("&Application"));
        menubar->Append(confmenu, wxT("&Configuration"));
        menubar->Append(colourmenu, wxT("C&olour"));
        menubar->Append(logmenu, wxT("&Log"));
    menubar->Append(helpmenu, wxT("&Help"));

        SetMenuBar(menubar);

        // Create ToolBar

    toolbar = new wxToolBar(this, wxID_ANY, wxDefaultPosition,
    wxDefaultSize, wxTB_HORIZONTAL | wxNO_BORDER | wxTB_FLAT);

        wxBitmap bmpStart(start_xpm);
        wxBitmap bmpStop(stop_xpm);
        wxBitmap bmpPause(pause_xpm);
        wxBitmap bmpContinuous(continuous_xpm);

    wxArrayString strings;
```

```
strings.Add(wxT("Low"));
strings.Add(wxT("Medium"));
strings.Add(wxT("High"));
strings.Add(wxT("Ultimate"));

    spinrange = new wxSpinCtrl(toolbar, ID_SPINRANGE,
wxT("30M"), wxDefaultPosition, wxSize(50, 20),
wxSP_ARROW_KEYS | wxSP_WRAP, 5, 100, 30);
    spingain = new wxSpinCtrl(toolbar, ID_SPINGAIN,
wxT("40%"), wxDefaultPosition, wxSize(55, 20),
wxSP_ARROW_KEYS | wxSP_WRAP, 10, 95, 40);
comboresol = new wxComboBox(toolbar, ID_COMBOBX,
wxT("Medium"), wxDefaultPosition, wxDefaultSize, strings,
wxCB_DROPDOWN | wxCB_READONLY);
spinleftlim = new wxSpinCtrl(toolbar, ID_SPINLEFTLIM,
wxT("0º"), wxDefaultPosition, wxSize(55, 20),
wxSP_ARROW_KEYS | wxSP_WRAP, 0, 359, 0);
    spinrightlim = new wxSpinCtrl(toolbar, ID_SPINRIGHTLIM,
wxT("0º"), wxDefaultPosition, wxSize(55, 20),
wxSP_ARROW_KEYS | wxSP_WRAP, 0, 359, 0);

toolbar->SetToolBitmapSize(wxSize(32, 32));

    toolbar->AddTool(ID_TBSTART, bmpStart, wxT("Start"));
    toolbar->AddTool(ID_TBSTOP, bmpStop, wxT("Stop"));
    toolbar->AddTool(ID_TBPAUSE, bmpPause, wxT("Pause"));
    toolbar->AddSeparator();

wxStaticText *labeltext1 = new wxStaticText(toolbar, wxID_ANY,
wxT("␣Range:␣"), wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext1->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext1);
    toolbar->AddControl(spinrange);
    toolbar->AddSeparator();

wxStaticText *labeltext2 = new wxStaticText(toolbar, wxID_ANY, wxT("␣Gain:␣"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext2->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext2);
    toolbar->AddControl(spingain);
    toolbar->AddSeparator();

wxStaticText *labeltext3 = new wxStaticText(toolbar, wxID_ANY, wxT("␣Resolution:␣"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext3->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext3);
    toolbar->AddControl(comboresol);
toolbar->AddSeparator();

    wxStaticText *labeltext4 = new wxStaticText(toolbar, wxID_ANY, wxT("␣Left␣Limit:␣"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext4->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext4);
    toolbar->AddControl(spinleftlim);
    toolbar->AddSeparator();
```

```
toolbar->AddTool(ID_TBCONTINUOUS, bmpContinuous, wxT("Continuous"));
toolbar->AddSeparator();

    wxStaticText *labeltext5 = new wxStaticText(toolbar, wxID_ANY, wxT("␣Right␣Limit:␣"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext5->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext5);
    toolbar->AddControl(spinrightlim);
    toolbar->AddSeparator();

threshold = new wxSlider(toolbar, ID_SLIDER, 25, 0, 255, wxDefaultPosition,
wxSize(150, -1), wxSL_HORIZONTAL|wxSL_LABELS|wxSL_SELRANGE);
    wxStaticText *labeltext6 = new wxStaticText(toolbar, wxID_ANY, wxT("␣Threshold:␣"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
labeltext6->SetBackgroundColour(toolbar->GetBackgroundColour());
threshold->SetBackgroundColour(toolbar->GetBackgroundColour());
toolbar->AddControl(labeltext6);
toolbar->AddControl(threshold);
toolbar->AddSeparator();

wxButton *clearbutton = new wxButton(toolbar, ID_CLEAR, wxT("Clear"),
wxDefaultPosition, wxSize(60, -1));
    toolbar->AddControl(clearbutton);
    toolbar->Realize();

    this->SetToolBar(toolbar);

CreateStatusBar(3);
SetStatusText(wxT("Sonar␣MiniKing␣Viewer"));
SetStatusText(wxT("Running"), 1);
SetStatusText(wxT("Continuous"), 2);

// Creating Sonar

switch (dialog->getSelected()) {

    case 1:
        sn = new MiniKing("COM1", 0);
        break;
    case 2:
        sn = new MiniKing("COM2", 0);
        break;
    case 3:
        sn = new MiniKing("COM3", 0);
        break;
    case 4:
        sn = new MiniKing("COM4", 0);
        break;
    case 5:
        sn = new MiniKing("COM5", 0);
        break;
    case 6:
        sn = new MiniKing("COM6", 0);
        break;
    case 7:
        sn = new MiniKing("COM7", 0);
```

```
            break;
        case 8:
            sn = new MiniKing("COM8", 0);
            break;
        case 9:
            sn = new MiniKing("COM9", 0);
            break;
        case 10:
            sn = new MiniKing("COM10", 0);
            break;
        case 11:
            sn = new MiniKing("COM11", 0);
            break;
        case 12:
            sn = new MiniKing("COM12", 0);
            break;
        default:
            wxMessageBox("Incorrect port", wxMessageBoxCaptionStr,
            wxOK | wxCENTRE |wxICON_ERROR);
            exit(0);
    }
    delete dialog;
}

// Return Sonar Pointer

MiniKing *MainFrame::getSonar() {

    return sn;
}

// Response to CLOSE Event

void MainFrame::OnClose(wxCloseEvent& event)
{
    pt->Kill();          // Stops the thread
    delete pt;           // Free thread memory
        Destroy();           // Close Frame
}

// Response to CLOSE Event

void MainFrame::OnQuit(wxCommandEvent& event)
{

    pt->Kill();          // Stops the thread
    delete pt;           // Free thread memory
        Destroy();           // Close Frame
}

// Saving configuration

void MainFrame::OnSaveConf(wxCommandEvent& event)
{

    mutex->Lock();
```

```
    wxFileDialog dialog(this, wxT("Save Configuration"), wxEmptyString,
    wxEmptyString, wxT("MiniKing Viewer Files (*.mkv) | *.mkv"),
    wxSAVE | wxOVERWRITE_PROMPT);
    if (dialog.ShowModal() == wxID_OK)
    {
        TiXmlDocument doc;

        TiXmlDeclaration *dcl = new TiXmlDeclaration("1.0", "", "");
        doc.LinkEndChild(dcl);

        TiXmlComment *cmt =
        new TiXmlComment("This file is auto-generated by MiniKing Viewer.");
        doc.LinkEndChild(cmt);

        TiXmlElement *conf = new TiXmlElement("configuration");
        conf->SetAttribute("name", dialog.GetFilename());

        TiXmlElement *element = new TiXmlElement("range");
        element->SetAttribute("value", spinrange->GetValue());
        conf->LinkEndChild(element);

        element = new TiXmlElement("gain");
        element->SetAttribute("value", spingain->GetValue());
        conf->LinkEndChild(element);

        element = new TiXmlElement("resolution");
        element->SetAttribute("value", comboresol->GetCurrentSelection());
        conf->LinkEndChild(element);

        element = new TiXmlElement("left");
        element->SetAttribute("value", spinleftlim->GetValue());
        conf->LinkEndChild(element);

        element = new TiXmlElement("right");
        element->SetAttribute("value", spinrightlim->GetValue());
        conf->LinkEndChild(element);

        element = new TiXmlElement("threshold");
        element->SetAttribute("value", threshold->GetValue());
        conf->LinkEndChild(element);

        doc.LinkEndChild(conf);

        doc.SaveFile(dialog.GetPath());
    }
    mutex->Unlock();
}

// Loads a sonar configuration from a file

void MainFrame::OnLoadConf(wxCommandEvent& event)
{

    wxClientDC dc(this);

    mutex->Lock();
```

```
wxFileDialog dialog(this, wxT("Open Configuration"), wxEmptyString,
wxEmptyString, wxT("MiniKing Viewer Files (*.mkv) | *.mkv"),
wxOPEN | wxFILE_MUST_EXIST);
TiXmlElement *element;
if (dialog.ShowModal() == wxID_OK)
{
    TiXmlDocument doc(dialog.GetPath());
    if (!doc.LoadFile()) {
        wxMessageBox(wxT("Cannot load this file"));
        mutex->Unlock();
        return;
    }

    TiXmlHandle hdoc(&doc);
    int range, gain, resolution, left, right, thresh;

    // Read Range

    element = hdoc.FirstChild("configuration").FirstChild("range").ToElement();
    if (!element) {
        wxMessageBox("This file is corrupted");
        mutex->Unlock();
        return;
    }

    if (element->QueryIntAttribute("value", &range) != TIXML_SUCCESS) {
        wxMessageBox("This file is corrupted");
        mutex->Unlock();
        return;
    }

    // Read Gain

    element = hdoc.FirstChild("configuration").FirstChild("gain").ToElement();
    if (!element) {
        wxMessageBox("This file is corrupted");
        mutex->Unlock();
        return;
    }

    if (element->QueryIntAttribute("value", &gain) != TIXML_SUCCESS) {
        wxMessageBox("This file is corrupted");
        mutex->Unlock();
        return;
    }

    // Read Resolution

    element = hdoc.FirstChild("configuration").FirstChild("resolution").ToElement();
    if (!element) {
        wxMessageBox("This file is corrupted");
        mutex->Unlock();
        return;
    }
```

```
if (element->QueryIntAttribute("value", &resolution) != TIXML_SUCCESS) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

// Read left

element = hdoc.FirstChild("configuration").FirstChild("left").ToElement();
if (!element) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

if (element->QueryIntAttribute("value", &left) != TIXML_SUCCESS) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

// Read Right

element = hdoc.FirstChild("configuration").FirstChild("right").ToElement();
if (!element) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

if (element->QueryIntAttribute("value", &right) != TIXML_SUCCESS) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

// Read Threshold

element = hdoc.FirstChild("configuration").FirstChild("threshold").ToElement();
if (!element) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

if (element->QueryIntAttribute("value", &thresh) != TIXML_SUCCESS) {
    wxMessageBox("This file is corrupted");
    mutex->Unlock();
    return;
}

// Set Text

int number;
double inc = (double)range / 5.0;
double add = 0.0;
```

```
wxString circ1, circ2, circ3, circ4, circ5;

add += inc;
number = (int)add;

circ1.Printf("%d",number);
pt->setCirc(1, circ1);

add += inc;
number = (int)add;

circ2.Printf("%d",number);
pt->setCirc(2, circ2);

add += inc;
number = (int)add;

circ3.Printf("%d",number);
pt->setCirc(3, circ3);

add += inc;
number = (int)add;

circ4.Printf("%d",number);
pt->setCirc(4, circ4);

add += inc;
number = (int)add;

circ5.Printf("%d",number);
pt->setCirc(5, circ5);

dc.Clear();
pt->paintBase(&dc);

spinrange->SetValue(range);
sn->setRange(range);

if (pt->getSave())
    (*logdoc) << "RANGE_" << (double)spinrange->GetValue() << "\n";

spingain->SetValue(gain);
sn->setGain(gain);

comboresol->SetSelection(resolution);
switch(resolution) {
    case 0:
        sn->setResolution(Low);
        break;
    case 2:
        sn->setResolution(High);
        break;
    case 3:
        sn->setResolution(Ultimate);
        break;
    default:
```

```
                sn->setResolution(Medium);
                break;
        }

        spinleftlim->SetValue(left);
        sn->setLeftLim(left);

        spinrightlim->SetValue(right);
        sn->setRightLim(right);

        threshold->SetValue(thresh);
        pt->setThreshold(thresh);

        // Updating Sonar configuration

        sn->updateConfig();
    }
    mutex->Unlock();
}

// Sets limit scan value

void MainFrame::OnLimit(wxCommandEvent& event)
{

    mutex->Lock();

    if (!pt->getLimitScan()) {

        wxNumberEntryDialog dialog(this, wxT(""),
        wxT("Enter a number:"), wxT("Input Scan Limit value"), 255, 0, 255);
        if (dialog.ShowModal() == wxID_OK)
        {
            long value = dialog.GetValue();
            pt->setLimitScan(true);
            pt->setLimitScanValue((int)value);
            scanlimitvalue->SetText(wxT("Unset limit value"));
        }
    }
    else {

        pt->setLimitScan(false);
        scanlimitvalue->SetText("Set limit value ...");
    }
    mutex->Unlock();
}

// Reboots the sonar

void MainFrame::OnReboot(wxCommandEvent& event)
{

    wxClientDC dc(this);

    mutex->Lock();
    wxMessageDialog dialog(this, wxT("Are you sure to reboot MiniKing Viewer?"),
```

```
        wxT("Reboot"),
        wxNO_DEFAULT| wxYES_NO | wxICON_INFORMATION);

    if(dialog.ShowModal() == wxID_YES) {
        SetStatusText("Rebooting sonar ...", 1);
        pt->setCirc(1, "6");
        pt->setCirc(2, "12");
        pt->setCirc(3, "18");
        pt->setCirc(4, "24");
        pt->setCirc(5, "30");
        if (pt->getSave())
            (*logdoc) << "RANGE 30.0\n";
        dc.Clear();
        pt->paintBase(&dc);
        spinrange->SetValue(30);
        spingain->SetValue(40);
        comboresol->Select(1);
        spinleftlim->SetValue(0);
        spinrightlim->SetValue(0);
        threshold->SetValue(25);
        pt->setThreshold(25);
        sn->reboot();
        if (pt->IsPaused())
            pt->Resume();
        SetStatusText("Running", 1);
    }
    mutex->Unlock();
}

// Start: change colormap methods

void MainFrame::OnBone(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/bone.txt");
    mutex->Unlock();
}

void MainFrame::OnCool(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/cool.txt");
    mutex->Unlock();
}

void MainFrame::OnCopper(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/copper.txt");
    mutex->Unlock();
}

void MainFrame::OnFlag(wxCommandEvent& event)
```

```
{

    mutex->Lock();
    pt->setColorMap("./colormaps/flag.txt");
    mutex->Unlock();
}

void MainFrame::OnGray(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/gray.txt");
    mutex->Unlock();
}

void MainFrame::OnHot(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/hot.txt");
    mutex->Unlock();
}

void MainFrame::OnHsv(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/hsv.txt");
    mutex->Unlock();
}

void MainFrame::OnJet(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/jet.txt");
    mutex->Unlock();
}

void MainFrame::OnPink(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setColorMap("./colormaps/pink.txt");
    mutex->Unlock();
}

// End: change colormap methods

// Takes an screenshot

void MainFrame::OnSnapshot(wxCommandEvent& event)
{

    wxClientDC dc(this);
```

```
    mutex->Lock();

    wxSize sizeDC = dc.GetSize();
    wxSize sizeTB = toolbar->GetToolSize();

    wxBitmap *bitmap = new wxBitmap(sizeDC.x, sizeDC.y - sizeTB.y);
    wxMemoryDC memDC;
    memDC.SelectObject(*bitmap);
    memDC.Blit(0, 0, sizeDC.x, sizeDC.y - sizeTB.y, &dc, 0, sizeTB.y + 25);
    memDC.SelectObject(wxNullBitmap);

    wxFileDialog dialog(this, wxT("Save image"), wxEmptyString,
    wxEmptyString, wxT("Bitmap Files (*.bmp)|*.bmp"),
    wxSAVE | wxOVERWRITE_PROMPT);
        if (dialog.ShowModal() == wxID_OK)
        {
            bitmap->SaveFile(dialog.GetPath(), wxBITMAP_TYPE_BMP);
        }
    mutex->Unlock();
}

// Play a log file

void MainFrame::OnLogPlay(wxCommandEvent& event)
{

    mutex->Lock();

    wxFileDialog dialog(this, wxT("Play Log"), wxEmptyString, wxEmptyString,
    wxT("MiniKing Log Files (*.mlf)|*.mlf"), wxOPEN | wxFILE_MUST_EXIST);
    if (dialog.ShowModal() == wxID_OK)
    {
        if (!pt->IsPaused())
            pt->Pause();

        SetStatusText("Playing ...", 1);

        ifstream *doc = new ifstream(dialog.GetPath());

        pt->paintDocument(doc);

        SetStatusText("Stopped", 1);
    }
    mutex->Unlock();
}

// Starts a log record

void MainFrame::OnLogRecord(wxCommandEvent& event)
{

    mutex->Lock();
    if (!pt->getSave()) {
        wxFileDialog dialog(this, wxT("Save Log"), wxEmptyString,
        wxEmptyString, wxT("MiniKing Log Files (*.mlf)|*.mlf"),
        wxSAVE | wxOVERWRITE_PROMPT);
```

```
        if (dialog.ShowModal() == wxID_OK)
        {
            pt->setSave(true);
            logdoc = new ofstream(dialog.GetPath());
            (*logdoc) << "RANGE " << (double)spinrange->GetValue() << "\n";
            pt->setDocument(logdoc);

            logrecord->SetText(wxT("Stop &record"));
            this->SetStatusText(wxT("Recording ..."), 1);
        }
    }
    else {
            pt->setSave(false);
            pt->setDocument(NULL);
            this->SetStatusText(wxT("Saving file ..."), 1);
            (*logdoc) << "ENDOFFILE\n";
            logdoc->close();
            logrecord->SetText("&Record ...");
            this->SetStatusText(wxT("Saved"), 1);
            wxSleep(2);
            if (pt->IsPaused())
                this->SetStatusText(wxT("Stopped"), 1);
            else
                this->SetStatusText(wxT("Running"), 1);
    }
    mutex->Unlock();
}

// Show help

void MainFrame::OnHelp(wxCommandEvent& event)
{

    mutex->Lock();

    wxBoxSizer *topsizer;
    wxHtmlWindow *html;
    wxDialog dlg(this, wxID_ANY, wxString(_("Help")));

    topsizer = new wxBoxSizer(wxVERTICAL);

    html = new wxHtmlWindow(&dlg, wxID_ANY, wxDefaultPosition, wxSize(500, 500), wxHW_SCROLLBAR_AUTO);
    html -> SetBorders(0);
    html -> LoadPage(wxT("help.html"));
    html -> SetSize(html -> GetInternalRepresentation() -> GetWidth(),
                    html -> GetInternalRepresentation() -> GetHeight());

    topsizer -> Add(html, 1, wxALL, 10);

#if wxUSE_STATLINE
    topsizer -> Add(new wxStaticLine(&dlg, wxID_ANY), 0, wxEXPAND | wxLEFT | wxRIGHT, 10);
#endif

    wxButton *bu1 = new wxButton(&dlg, wxID_OK, _("OK"));
    bu1 -> SetDefault();
```

```
    topsizer -> Add(bu1, 0, wxALL | wxALIGN_RIGHT, 15);

    dlg.SetSizer(topsizer);
    topsizer -> Fit(&dlg);

    dlg.Center();

    dlg.ShowModal();

    mutex->Unlock();
}

// Show about

void MainFrame::OnAbout(wxCommandEvent& event)
{

    mutex->Lock();

    wxBoxSizer *topsizer;
    wxHtmlWindow *html;
    wxDialog dlg(this, wxID_ANY, wxString(_("About")));

    topsizer = new wxBoxSizer(wxVERTICAL);

    html = new wxHtmlWindow(&dlg, wxID_ANY, wxDefaultPosition,
    wxSize(380, 160), wxHW_SCROLLBAR_NEVER);
    html -> SetBorders(0);
    html -> LoadPage(wxT("about.html"));
    html -> SetSize(html -> GetInternalRepresentation() -> GetWidth(),
                    html -> GetInternalRepresentation() -> GetHeight());

    topsizer -> Add(html, 1, wxALL, 10);

#if wxUSE_STATLINE
    topsizer -> Add(new wxStaticLine(&dlg, wxID_ANY), 0,
    wxEXPAND | wxLEFT | wxRIGHT, 10);
#endif

    wxButton *bu1 = new wxButton(&dlg, wxID_OK, _("OK"));
    bu1 -> SetDefault();

    topsizer -> Add(bu1, 0, wxALL | wxALIGN_RIGHT, 15);

    dlg.SetSizer(topsizer);
    topsizer -> Fit(&dlg);

    dlg.Center();

    dlg.ShowModal();

    mutex->Unlock();
}

// Start scan button
```

```
void MainFrame::OnStart(wxCommandEvent& event)
{

    wxClientDC dc(this);

    mutex->Lock();


    if (pt->IsPaused()) {
        sn->updateConfig();
        pt->Resume();
        if (pt->getRepaint()) {
            dc.Clear();
            pt->paintBase(&dc);
            pt->setRepaint(false);
        }
    }
    SetStatusText("Running", 1);
    mutex->Unlock();
}

// Stop scan button

void MainFrame::OnStop(wxCommandEvent& event)
{

    wxClientDC dc(this);

    mutex->Lock();
    if (!pt->IsPaused())
        pt->Pause();
    dc.Clear();
    pt->paintBase(&dc);
    SetStatusText("Stopped", 1);
    mutex->Unlock();
}

// Pause scan button

void MainFrame::OnPause(wxCommandEvent& event)
{

    mutex->Lock();
    if (!pt->IsPaused())
        pt->Pause();
    SetStatusText("Paused", 1);
    mutex->Unlock();
}

// Set/Unset continuous scan

void MainFrame::OnContinuous(wxCommandEvent& event)
{

    mutex->Lock();
    if (sn->getContinuous()) {
```

```
        sn->setContinuous(false);
        SetStatusText(wxT("Not Continuous"), 2);
    }
    else {
        sn->setContinuous(true);
        SetStatusText(wxT("Continuous"), 2);
    }
    sn->updateConfig();
    mutex->Unlock();
}

// Modify range

void MainFrame::OnSpinRange(wxSpinEvent& event)
{
    wxClientDC dc(this);
    mutex->Lock();
    int value = spinrange->GetValue();
    sn->setRange(value);

    int number;
    double inc = (double)value / 5.0;
    double add = 0.0;
    wxString circ1, circ2, circ3, circ4, circ5;

    add += inc;
    number = (int)add;

    circ1.Printf("%d",number);
    pt->setCirc(1, circ1);

    add += inc;
    number = (int)add;

    circ2.Printf("%d",number);
    pt->setCirc(2, circ2);

    add += inc;
    number = (int)add;

    circ3.Printf("%d",number);
    pt->setCirc(3, circ3);

    add += inc;
    number = (int)add;

    circ4.Printf("%d",number);
    pt->setCirc(4, circ4);

    add += inc;
    number = (int)add;

    circ5.Printf("%d",number);
    pt->setCirc(5, circ5);

    if (!pt->IsPaused()) {
```

```
            pt->Pause();
            SetStatusText("Paused", 1);
        }
        dc.Clear();
        pt->paintBase(&dc);

        if (pt->getSave())
            (*logdoc) << "RANGE_" << (double)spinrange->GetValue() << "\n";
        mutex->Unlock();
}

// Modify gain

void MainFrame::OnSpinGain(wxSpinEvent& event)
{

        mutex->Lock();
        sn->setGain(spingain->GetValue());
        if (!pt->IsPaused()) {
            pt->Pause();
            SetStatusText("Paused", 1);
        }
        mutex->Unlock();
}

// Modify Resolution

void MainFrame::OnComboBox(wxCommandEvent& event)
{

        mutex->Lock();
        switch(event.GetSelection()) {
            case 0:
                sn->setResolution(Low);
                break;
            case 2:
                sn->setResolution(High);
                break;
            case 3:
                sn->setResolution(Ultimate);
                break;
            default:
                sn->setResolution(Medium);
                break;
        }
        sn->updateConfig();
        mutex->Unlock();
}

// Modify left limit

void MainFrame::OnSpinLeft(wxSpinEvent& event)
{

        mutex->Lock();
        sn->setLeftLim(spinleftlim->GetValue());
```

```
    if (!pt->IsPaused()) {
        pt->Pause();
        SetStatusText("Paused", 1);
    }
    mutex->Unlock();
}

// Modify right limit

void MainFrame::OnSpinRight(wxSpinEvent& event)
{

    mutex->Lock();
    sn->setRightLim(spinrightlim->GetValue());
    if (!pt->IsPaused()) {
        pt->Pause();
        SetStatusText("Paused", 1);
    }
    mutex->Unlock();
}

// Modify threshold

void MainFrame::OnSlide(wxCommandEvent& event)
{

    mutex->Lock();
    pt->setThreshold(threshold->GetValue());
    mutex->Unlock();
}

// Clear button

void MainFrame::OnClear(wxCommandEvent& event)
{

    wxClientDC dc(this);

    mutex->Lock();
    wxMessageDialog dialog(this, wxT("Are you sure to clear screen?"),
      wxT("Clear"),
      wxNO_DEFAULT| wxYES_NO | wxICON_INFORMATION);

    if (dialog.ShowModal() == wxID_YES) {
        dc.Clear();
        pt->paintBase(&dc);
    }

    mutex->Unlock();
}

// Erase event

void MainFrame::OnErase(wxEraseEvent& event)
{
    mutex->Lock();
```

```
    wxClientDC *dc = (wxClientDC *)event.GetDC();

    dc->SetBackground(wxBrush(wxColour(0, 0, 80)));

    dc->Clear();
    pt->paintBase(dc);

    mutex->Unlock();
}

// Show ping statistics

void MainFrame::OnMouseDClick(wxMouseEvent& event) {

    wxClientDC dc(this);

    mutex->Lock();

    wxCoord w, h;
    dc.GetSize(&w, &h);

    wxPoint pnt(event.GetPosition());
    pnt.x -= w / 2;
    pnt.y -= h / 2;

    if (sqrt(pnt.x * pnt.x + pnt.y * pnt.y) < 300) {

        float angle = (180.0 * atan2(pnt.y, pnt.x)) / 3.14159265;

        if (angle < 0)
            angle += 360.0;

        pt->showPing(angle);
    }

    mutex->Unlock();
}

// Paint event

void MainFrame::OnPaint(wxPaintEvent& event)
{}
```

PaintThread.h

```
/*
=============================================================================
  Name: PaintThread.h
  Author: Emilio García Fidalgo
  Date: 15/12/06 13:09
  Description: PaintThread class declaration
=============================================================================
```

```
*/

#include <wx/wxprec.h>

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#include "tinyxml/tinyxml.h"
#include "tinyxml/tinystr.h"

#ifndef WX_PRECOMP

#include <wx/wx.h>
#endif

#include "../Sonar/MiniKing.h"
#include <iostream>
#include <fstream>

#ifndef PAINTTHREAD_H
#define PAINTTHREAD_H

class PaintThread : public wxThread {

    public:
            PaintThread(wxFrame *);                 // Constructor
            void setSonar(MiniKing *);              // Sets sonar pointer to obtain information
            void setMutex(wxMutex *);               // Mutual Exclusion
            void setThreshold(unsigned int);        // Change threshold value
            void setColorMap(char *);               // Set painting colors from a file
            void paintBase(wxClientDC *);           // Paints main frame
            void setDocument(ofstream *);           // Set stream for store data
            ofstream *getDocument(void);            // Returns the store data stream
            void setSave(bool);                     // Set 'Save' option
            bool getSave(void);                     // Return 'Save' option value
            void paintDocument(ifstream *);         // Paints a log file
            void setCirc(int, wxString);            // Modifiy distance value in circle
            bool getRepaint(void);                  // Return if a repaint is needed in main window
            void setRepaint(bool);                  // Sets 'repaint' option to a value
            void showPing(float);                   // Show statistics about a ping
            void setLimitScan(bool);                // Sets the limit scan
            bool getLimitScan(void);                // Returns if is selected limit scan
            void setLimitScanValue(int);            // Sets the limit scan value

    private:

            MiniKing *sn;
            wxFrame *frame;
            wxMutex *mutex;
            unsigned int threshold;
            bool save;
            bool limitscan;
            int limitscanvalue;
            ofstream *doc;
            wxString circ1, circ2, circ3, circ4, circ5;
```

```
            bool repaint;

            virtual void *Entry();
};

#endif
```

PaintThread.cpp

```
/*
=============================================================================
  Name: PaintThread.cpp
  Author: Emilio García Fidalgo
  Date: 15/12/06 16:33
  Description: PaintThread class implementation
=============================================================================
*/

#include "PaintThread.h"
#include "ShowPingDialog.h"
#include <fstream>
#include <iostream>
#include <cmath>
#include <cstring>
#include <cstdio>
#include <vector>
#include <climits>

const int NPIX = 300;                   // Circle radio. Same as bins.
wxColour color[256];

// Ping structure

typedef struct {

    double angle;
    BYTE data[NPIX];
    unsigned lap;
} Ping;

vector<Ping> pings[360];
unsigned laps;

// Constructor

PaintThread::PaintThread(wxFrame *fr) : wxThread(wxTHREAD_JOINABLE)
{
    frame = fr;
    threshold = 25;
    save = false;
    doc = NULL;
    for (int i = 0; i < 256; i++)
        color[i].Set(0, 0, 0);
```

```
    setColorMap("./colormaps/bone.txt");
    repaint = false;
    circ1 = "6";
    circ2 = "12";
    circ3 = "18";
    circ4 = "24";
    circ5 = "30";
    laps = 0;
    limitscan = false;
    limitscanvalue = 255;
}

// Paints main window base

void PaintThread::paintBase(wxClientDC *dc) {

    wxCoord w, h;

    // Sets the device context origin

    dc->GetSize(&w, &h);
    dc->SetDeviceOrigin(w / 2, h / 2 + 40);

    // Draw background circle

    dc->SetPen(*wxBLACK_PEN);
    dc->SetBrush(*wxBLACK_BRUSH);
    dc->DrawCircle(0, 0, NPIX);

    dc->SetPen(*wxRED_PEN);

    // Draw CrossHair

    dc->CrossHair(0, 0);

    // Paints concentric circles

    double inc = 0.225;
    int x, y;
    for (int i = NPIX / 5; i <= NPIX; i = i + NPIX / 5) {
        for (double j = 0.0; j < 360.0; j = j + inc) {
            x = (int)ceil(i * (cos((j * 3.14159265) / 180.0)));
            y = (int)ceil(i * (sin((j * 3.14159265) / 180.0)));
            dc->DrawPoint(x, y);
        }
    }

    // Draw Text

    wxFont font(12, wxFONTFAMILY_SWISS, wxNORMAL, wxBOLD);
    dc->SetFont(font);
    dc->SetTextForeground(*wxRED);
    dc->SetBackgroundMode(wxTRANSPARENT);
    dc->DrawText(circ1, wxPoint(37,10));
    dc->DrawText(circ2, wxPoint(97,10));
    dc->DrawText(circ3, wxPoint(157,10));
```

```
    dc->DrawText(circ4, wxPoint(217,10));
    dc->DrawText(circ5, wxPoint(277,10));
}

// Upgrade previous angle pings in vector 'pings'

void deletePings(int number) {

        vector<Ping>::iterator index = pings[number].begin();
        Ping p;

        while (index != pings[number].end()) {

            p = (*index);
            if (p.lap != laps) {
                index = pings[number].erase(index);
                free(&p);
            }
            else
                index++;
        }
}

// Shows a

void PaintThread::showPing(float angle) {

    int hash = (int)angle;
    double maxdif = UINT_MAX;
    int i = 0;

    if (pings[hash].size() != 0) {

        vector<Ping>::iterator index = pings[hash].begin();
        Ping p;

        int count = 0;

        while (index != pings[hash].end()) {

            p = (*index);
            if (abs(angle - p.angle) < maxdif) {
                maxdif = abs(angle - p.angle);
                i = count;
            }
            index++;
            count++;
        }

        p = pings[hash].at(i);
        ShowPingDialog *spd = new ShowPingDialog(p.data, angle, sn->getRange(), NULL);
        spd->Show();
    }
}

// Storing general variables
```

```
// Starting thread function

void *PaintThread::Entry() {

    wxClientDC dc(frame);
    paintBase(&dc);

    BYTE *data;
    int ndatabytes;
    float angle, angle_old = 0.0;
    bool clock, clock_old = true;
    Ping newelement;
    int hash;

    int x, y;
    int hashremove;
    while (true) {

        mutex->Lock();

        // Draw basic document

        dc.SetPen(*wxRED_PEN);
        dc.CrossHair(0, 0);

        // Draw Text

        wxFont font(12, wxFONTFAMILY_SWISS, wxNORMAL, wxBOLD);
        dc.SetFont(font);
        dc.SetTextForeground(*wxRED);
        dc.SetBackgroundMode(wxTRANSPARENT);
        dc.DrawText(circ1, wxPoint(37,10));
        dc.DrawText(circ2, wxPoint(97,10));
        dc.DrawText(circ3, wxPoint(157,10));
        dc.DrawText(circ4, wxPoint(217,10));
        dc.DrawText(circ5, wxPoint(277,10));

        // Get Data

        data = sn->getScanLine();
        ndatabytes = sn->getDataLength();
        angle = sn->getPosition();

        // Store data

        if (angle >= angle_old)
            clock = true;
        else
            clock = false;

        hash = (int)angle;
        newelement.angle = angle;
        newelement.lap = laps;
        for (int i = 0; i < NPIX; i++)
            newelement.data[i] = data[i];
```

```
pings[hash].push_back(newelement);

if (hash == 0) {

    if (clock)
        hashremove = 359;
    else
        hashremove = 1;
}
else if (hash == 359) {

    if (clock)
        hashremove = 0;
    else
        hashremove = 358;
}
else {

    if (clock)
        hashremove = hash - 1;
    else
        hashremove = hash + 1;
}

deletePings(hashremove);

if (sn->getContinuous()) {

    if (hash == 0 || hash == 1) {

        laps++;
        laps %= UINT_MAX;
    }
}
else {

    if (!clock && clock_old) {
        laps++;
        laps %= UINT_MAX;
    }
}

angle_old = angle;
clock_old = clock;

// End Storing

// Draw Ping

if (angle == 0.0 || angle == 90.0 || angle == 180.0 || angle == 270.0) {
    mutex->Unlock();
    continue;
}
else {
    if (save)
```

```
                (*doc) << angle << "␣";

            bool paintblack;
            bool end = false;
            if (limitscan)
                paintblack = true;
            else
                paintblack = false;

            // Data line

            for (int i = 0; i < ndatabytes; i++) {
                if (i == 60 || i == 120 || i == 180 || i == 240 || i == 300) {
                    continue;
                }
                else {
                    if (limitscan && ((unsigned)data[i] >= (unsigned)limitscanvalue) && !end) {
                        paintblack = false;
                        end = true;
                    }

                    if ((unsigned)data[i] < threshold || paintblack)
                        dc.SetPen(*wxBLACK_PEN);
                    else
                        dc.SetPen(color[(int)data[i]]);

                    if (limitscan && end && ((unsigned)data[i] < (unsigned)limitscanvalue))
                            paintblack = true;

                    x = (int)ceil(i * (cos((angle * 3.14159265) / 180.0)));
                    y = (int)ceil(i * (sin((angle * 3.14159265) / 180.0)));
                    dc.DrawPoint(x,y);
                }
                if (save) {
                    if (data[i] == 0x1A)
                        (*doc) << "$$␣";
                    else {
                        doc->put(data[i]);
                        doc->put('␣');
                    }
                }
            }
            if (save)
                (*doc) << "STOP\n";
        }
        mutex->Unlock();
    }
    while (true);
    return NULL;
}

// Plays a log file

void PaintThread::paintDocument(ifstream *file)
{
```

```
wxClientDC dc(frame);

dc.Clear();
paintBase(&dc);

int x, y;
double angleaux;
char line[50];
BYTE aux[350];
int index = 0;
char c;
float angle, angle_old = 0.0;
bool clock, clock_old = true;
Ping newelement;
int hash, hashremove;

wxString circ1_old = circ1;
wxString circ2_old = circ2;
wxString circ3_old = circ3;
wxString circ4_old = circ4;
wxString circ5_old = circ5;

mutex->Lock();

while (true) {

    dc.SetPen(*wxRED_PEN);
    dc.CrossHair(0,0);

    wxFont font(12, wxFONTFAMILY_SWISS, wxNORMAL, wxBOLD);
    dc.SetFont(font);
    dc.SetTextForeground(*wxRED);
    dc.SetBackgroundMode(wxTRANSPARENT);

    dc.DrawText(circ1, wxPoint(37,10));
    dc.DrawText(circ2, wxPoint(97,10));
    dc.DrawText(circ3, wxPoint(157,10));
    dc.DrawText(circ4, wxPoint(217,10));
    dc.DrawText(circ5, wxPoint(277,10));

    int i = 0;
    do {
        file->get(c);
        line[i] = c;
        i++;
    } while (c != ' ' && c != '\n');
    line[i - 1] = '\0';

    if (strcmp(line, "ENDOFFILE") == 0)
        break;
    else if (strcmp(line, "RANGE") == 0) {
        int i = 0;
        wxString c1, c2, c3, c4, c5;
        do {
            file->get(c);
            line[i] = c;
```

```
            i++;
        } while (c != '\n');
        line[i - 1] = '\0';

        int value = (int)atof(line);
        wxString msg;

        double inc = (double)value / 5.0;
        double add = 0.0;

        add += inc;
        int number = (int)add;

        c1.Printf("%d",number);
        setCirc(1, c1);

        add += inc;
        number = (int)add;

        c2.Printf("%d",number);
        setCirc(2, c2);

        add += inc;
        number = (int)add;

        c3.Printf("%d",number);
        setCirc(3, c3);

        add += inc;
        number = (int)add;

        c4.Printf("%d",number);
        setCirc(4, c4);

        add += inc;
        number = (int)add;

        c5.Printf("%d",number);
        setCirc(5, c5);

        dc.Clear();
        paintBase(&dc);
    }
    else {
        angleaux = (double)atof(line);
        angle = angleaux;
        index = 0;

        int j = -1;

        bool paintblack;
        bool end = false;
        if (limitscan)
            paintblack = true;
        else
            paintblack = false;
```

```
while (true) {

    i = 0;
    do {
        file->get(c);
        line[i] = c;
        i++;
    } while (c != ' ' && c != '\n');
    line[i - 1] = '\0';

    if (strcmp(line, "STOP") == 0) {

        // Store data

        if (angle >= angle_old)
            clock = true;
        else
            clock = false;

        hash = (int)angle;
        newelement.angle = angle;
        newelement.lap = laps;
        for (int i = 0; i < NPIX; i++)
            newelement.data[i] = aux[i];

        pings[hash].push_back(newelement);

        if (hash == 0) {

            if (clock)
                hashremove = 359;
            else
                hashremove = 1;
        }
        else if (hash == 359) {

            if (clock)
                hashremove = 0;
            else
                hashremove = 358;
            }
        else {

            if (clock)
                hashremove = hash - 1;
            else
                hashremove = hash + 1;
        }

        deletePings(hashremove);

        if (sn->getContinuous()) {

            if (hash == 0 || hash == 1) {
```

```
                laps++;
                laps %= UINT_MAX;
            }
        }
        else {

            if (!clock && clock_old) {
                laps++;
                laps %= UINT_MAX;
            }
        }

        angle_old = angle;
        clock_old = clock;

        // End Storing

        break;
    }
    if (strcmp(line, "$$") == 0) {

        aux[index++] = line[0];

        j++;
        if (j == 60 || j == 120 || j == 180 || j == 240 || j == 300)
            continue;
        else {

            if (limitscan && ((unsigned)line[0] >= (unsigned)limitscanvalue) && !end) {
                paintblack = false;
                end = true;
            }

            if ((unsigned)line[0] < threshold || paintblack)
                dc.SetPen(*wxBLACK_PEN);
            else
                dc.SetPen(color[0x1A]);

            if (limitscan && end && ((unsigned)line[0] < (unsigned)limitscanvalue))
                paintblack = true;

            x = (int)ceil(j * (cos((angleaux * 3.14159265) / 180.0)));
            y = (int)ceil(j * (sin((angleaux * 3.14159265) / 180.0)));
            dc.DrawPoint(x,y);
        }
    }
    else {

        aux[index++] = line[0];

        j++;
        if (j == 60 || j == 120 || j == 180 || j == 240 || j == 300)
            continue;
        else {

            if (limitscan && ((unsigned)line[0] >= (unsigned)limitscanvalue) && !end) {
```

```
                                paintblack = false;
                                end = true;
                            }

                            if ((unsigned)line[0] < threshold || paintblack)
                                dc.SetPen(*wxBLACK_PEN);
                            else if ((unsigned)line[0] > 255)
                                dc.SetPen(color[254]);
                            else
                                dc.SetPen(color[(int)line[0]]);

                            if (limitscan && end && ((unsigned)line[0] < (unsigned)limitscanvalue))
                                paintblack = true;

                            x = (int)ceil(j * (cos((angleaux * 3.14159265) / 180.0)));
                            y = (int)ceil(j * (sin((angleaux * 3.14159265) / 180.0)));
                            dc.DrawPoint(x,y);
                        }
                    }
                }
            }
        }

        setCirc(1, circ1_old);
        setCirc(2, circ2_old);
        setCirc(3, circ3_old);
        setCirc(4, circ4_old);
        setCirc(5, circ5_old);

        repaint = true;

        mutex->Unlock();
}

// Sets the sonar pointer to obtain data

void PaintThread::setSonar(MiniKing *mk) {

    sn = mk;
}

// Mutual Exclusion

void PaintThread::setMutex(wxMutex *mt) {

    mutex = mt;
}

// Change threshold value

void PaintThread::setThreshold(unsigned int th)
{
    threshold = th;
}

// Set the limit scan boolean option
```

```
void PaintThread::setLimitScan(bool scn)
{
    limitscan = scn;
}

// Returns the limit scan option

bool PaintThread::getLimitScan(void)
{

    return limitscan;
}

// Sets de limit scan value

void PaintThread::setLimitScanValue(int vl)
{

    limitscanvalue = vl;
}

// Sets 'Save' variable

void PaintThread::setSave(bool sv)
{

    save = sv;
}

// Returns 'Save' variable

bool PaintThread::getSave(void)
{

    return save;
}

// Returns 'Repaint' variable

bool PaintThread::getRepaint(void)
{

    return repaint;
}

// Sets 'Repaint' variable

void PaintThread::setRepaint(bool op) {

    repaint = op;
}

// Return saving document

ofstream *PaintThread::getDocument(void)
```

```
{

    return doc;
}

// Set saving document

void PaintThread::setDocument(ofstream *txd)
{

    doc = txd;
}

// Set circle's value in metres

void PaintThread::setCirc(int pos, wxString value)
{

    switch (pos) {

        case 1:
            circ1 = value;
            break;
        case 2:
            circ2 = value;
            break;
        case 3:
            circ3 = value;
            break;
        case 4:
            circ4 = value;
            break;
        case 5:
            circ5 = value;
            break;
    }
}

// Round Number

int RoundNumber(double number) {

    return int(number + 0.5);
}

// Change color map

void PaintThread::setColorMap(char *file)
{

    ifstream fs(file);
    char line[200];
    char *value;
    double db;
    unsigned char red, green, blue;
```

```
    for (int i = 0; i < 256; i++)
    {

        // Get Line

        fs.getline(line, 200);

        // Red

        value = strtok(line, " ");
        db = atof(value);
        db *= 255.0;
        red = (unsigned char)RoundNumber(db);

        // Green

        value = strtok(NULL, " ");
        db = atof(value);
        db *= 255.0;
        green = (unsigned char)RoundNumber(db);

        // Blue

        value = strtok(NULL, " ");
        db = atof(value);
        db *= 255.0;
        blue = (unsigned char)RoundNumber(db);

        // Setting Colour

        color[i].Set(red, green, blue);
    }
}
```

PortDialog.h

```
/*
=============================================================================
  Name: PortDialog.h
  Author: Emilio García Fidalgo
  Date: 29/12/06 13:32
  Description: PortDialog Class Selection
=============================================================================
*/

#ifndef __PORTDIALOG_h__
#define __PORTDIALOG_h__

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
```

```
        #include <wx/wx.h>
        #include <wx/dialog.h>
#else
        #include <wx/wxprec.h>
#endif

#include <wx/button.h>
#include <wx/combobox.h>

#undef PortDialog_STYLE
#define PortDialog_STYLE wxCAPTION | wxSYSTEM_MENU | wxDIALOG_NO_PARENT | wxMINIMIZE_BOX | wxCLOSE_BOX

class PortDialog : public wxDialog
{
        private:

                DECLARE_EVENT_TABLE();
                wxButton *WxButton1;
                wxComboBox *WxComboBox1;
                enum
                {
                        ID_WXBUTTON1 = 1002,
                        ID_WXCOMBOBOX1 = 1001
                };
                void OnOK(wxCommandEvent& event);           // Action button
                void OnClose(wxCloseEvent& event);          // Close
                void CreateGUIControls();
                int selected;

        public:

        // Constructor

                PortDialog(wxWindow *parent, wxWindowID id = 1,
        const wxString &title = wxT("Selects Sonar COM Port"),
        const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize, long style = PortDialog_STYLE);
                virtual ~PortDialog();     // Destructor
                int getSelected(void);     // Gets selected value
};

#endif
```

PortDialog.cpp

```
/*
=============================================================================
  Name: PortDialog.h
  Author: Emilio García Fidalgo
  Date: 29/12/06 13:35
  Description: PortDialog Class Implementation
=============================================================================
*/
```

```
#include "PortDialog.h"

#include "mkv.xpm"

// EVENT TABLE

BEGIN_EVENT_TABLE(PortDialog,wxDialog)
    EVT_BUTTON(ID_WXBUTTON1, PortDialog::OnOK)
        EVT_CLOSE(PortDialog::OnClose)
END_EVENT_TABLE()

// Constructor

PortDialog::PortDialog(wxWindow *parent, wxWindowID id,
const wxString &title, const wxPoint &position, const wxSize& size, long style)
: wxDialog(parent, id, title, position, size, style)
{
    selected = 0;
        CreateGUIControls();
}

// Destructor

PortDialog::~PortDialog()
{
}

// Creating controls

void PortDialog::CreateGUIControls()
{
    wxIcon icon(mkv_xpm);

        SetTitle(wxT("Selects Sonar COM Port"));
        SetIcon(icon);
        SetSize(8,8,261,171);
        Center();

        WxButton1 = new wxButton(this, ID_WXBUTTON1, wxT("Select"),
    wxPoint(80,82), wxSize(93,31), 0, wxDefaultValidator, wxT("Select"));

        wxArrayString strings;
        strings.Add(wxT("-- COM Port --"));
        strings.Add(wxT("COM 1"));
        strings.Add(wxT("COM 2"));
        strings.Add(wxT("COM 3"));
        strings.Add(wxT("COM 4"));
        strings.Add(wxT("COM 5"));
        strings.Add(wxT("COM 6"));
        strings.Add(wxT("COM 7"));
        strings.Add(wxT("COM 8"));
        strings.Add(wxT("COM 9"));
        strings.Add(wxT("COM 10"));
        strings.Add(wxT("COM 11"));
        strings.Add(wxT("COM 12"));
```

```
        WxComboBox1 = new wxComboBox(this, ID_WXCOMBOBOX1, wxT("-- COM Port --"),
    wxPoint(40,31), wxSize(181,21), strings, 0, wxDefaultValidator, wxT("-- COM Port --"));
}

// Close window

void PortDialog::OnClose(wxCloseEvent& event)
{
        Destroy();
}

// Click on 'OK' button

void PortDialog::OnOK(wxCommandEvent& event)
{

    selected = WxComboBox1->GetSelection();
    Destroy();
}

// Returns selected value

int PortDialog::getSelected()
{

    return selected;
}
```

ShowPingDialog.h

```
/*
=============================================================================
  Name: ShowPingDialog.h
  Author: Emilio García Fidalgo
  Date: 19/1/06 12:34
  Description: ShowPingDialog class declaration
=============================================================================
*/

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include <wx/wx.h>
        #include <wx/frame.h>
#else
        #include <wx/wxprec.h>
#endif

#include "../Sonar/MiniKing.h"

#ifndef SHOWPINGDIALOG_H
```

```
#define SHOWPINGDIALOG_H

#undef ShowPingDialog_STYLE
#define ShowPingDialog_STYLE wxCAPTION | wxSYSTEM_MENU | wxCLOSE_BOX |
wxMINIMIZE_BOX | wxCLIP_CHILDREN | wxFULL_REPAINT_ON_RESIZE

class ShowPingDialog : public wxFrame
{
        private:

                DECLARE_EVENT_TABLE();
        BYTE data[300];                         // Scan values
        float angle;                            // Ping angle
        int range;                              // Range distance
                void OnClose(wxCloseEvent& event);      // Close window
                void CreateGUIControls();               // Creating controls
                enum
                {
                        ID_DUMMY_VALUE_
                };

                void OnErase(wxEraseEvent& event);       // Erase event

        public:

        // Constructor

                ShowPingDialog(BYTE *, float, int, wxWindow *parent,
        wxWindowID id = 1, const wxString &title = wxT("Show ping"),
        const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize, long style = ShowPingDialog_STYLE);
        virtual ~ShowPingDialog();      // Destructor
            void draw(wxClientDC *dc);      // Draw data
};

#endif
```

ShowPingDialog.cpp

```
/*
=============================================================================
  Name: ShowPingDialog.cpp
  Author: Emilio García Fidalgo
  Date: 19/1/06 13:42
  Description: ShowPingDialog class implementation
=============================================================================
*/

#include "ShowPingDialog.h"
#include "mkv.xpm"

// EVENT TABLE
```

```
BEGIN_EVENT_TABLE(ShowPingDialog,wxFrame)
        EVT_ERASE_BACKGROUND(ShowPingDialog::OnErase)
        EVT_CLOSE(ShowPingDialog::OnClose)
END_EVENT_TABLE()

// Constructor

ShowPingDialog::ShowPingDialog(BYTE *pngs, float ang, int rng,
wxWindow *parent, wxWindowID id, const wxString &title,
const wxPoint &position, const wxSize& size, long style)
: wxFrame(parent, id, title, position, size, style)
{
    for (int i = 0; i < 300; i++)
        data[i] = pngs[i];
    angle = ang;
    range = rng;
        CreateGUIControls();
}

// Destructor

ShowPingDialog::~ShowPingDialog()
{
}

// Draw data

void ShowPingDialog::draw(wxClientDC *dc)
{

        dc->SetDeviceOrigin(35, 575);
        dc->SetPen(*wxRED_PEN);

        dc->DrawLine(0, 0, 600, 0);
    dc->DrawLine(0, 0, 0, -510);

    wxFont font(8, wxFONTFAMILY_SWISS, wxNORMAL, wxBOLD);
    dc->SetFont(font);
    dc->SetTextForeground(*wxRED);
    dc->SetBackgroundMode(wxTRANSPARENT);

    for (int i = 5; i < 300; i = i + 5)
        dc->DrawLine(i * 2, 0, i * 2, 5);

    wxString msg;

    for (int i = 10; i < 300; i = i + 10) {
        msg.Printf("%d", i);
        dc->DrawRotatedText(msg, wxPoint((i * 2) + 8, 10), -90.0);
    }

    // Painting distance in metres

    double inc = (double)range / 5.0;
    double add = 0.0;
```

```
    add += inc;
    int number = (int)add;

    msg.Printf("%d m",number);
    dc->DrawRotatedText(msg, wxPoint(128, 50), -90.0);

    add += inc;
    number = (int)add;

    msg.Printf("%d m",number);
    dc->DrawRotatedText(msg, wxPoint(248, 50), -90.0);

    add += inc;
    number = (int)add;

    msg.Printf("%d m",number);
    dc->DrawRotatedText(msg, wxPoint(368, 50), -90.0);

    add += inc;
    number = (int)add;

    msg.Printf("%d m",number);
    dc->DrawRotatedText(msg, wxPoint(488, 50), -90.0);

    add += inc;
    number = (int)add;

    msg.Printf("%d m",number);
    dc->DrawRotatedText(msg, wxPoint(608, 50), -90.0);

    for (int i = 5; i < 255; i = i + 5)
        dc->DrawLine(0, -(i * 2), -5, -(i * 2));

    for (int i = 10; i < 255; i = i + 10) {
        msg.Printf("%d", i);
        dc->DrawText(msg, wxPoint(-25, -(i * 2) -8));
    }

    msg.Printf("Angle: %d", angle);
    dc->DrawText(msg, wxPoint((2 * 120), -560));

    int x_old = 0;
    int y_old = 0;
    int x, y;

    for (int i = 0; i < 300; i++) {
        x = i * 2;
        y = -(int)data[i] * 2;
        dc->DrawPoint(x, y);
        dc->DrawLine(x_old, y_old, x, y);
        x_old = x;
        y_old = y;
    }
}

// Creating controls
```

```
void ShowPingDialog::CreateGUIControls()
{

        SetTitle(wxT("Show ping"));
        SetIcon(wxNullIcon);
        SetSize(8,8,675,700);
        SetBackgroundColour(wxColour(0, 0, 0));

    wxIcon icon(mkv_xpm);
    SetIcon(icon);
        Center();
}

// Close window

void ShowPingDialog::OnClose(wxCloseEvent& event)
{
        Destroy();
}

// Erase event

void ShowPingDialog::OnErase(wxEraseEvent& event) {

    wxClientDC *dc = (wxClientDC *)event.GetDC();

    dc->SetBackground(wxBrush(*wxBLACK_BRUSH));
    dc->Clear();
    draw(dc);
}
```

InitialDialog.h

```
/*
=============================================================================
  Name: InitialDialog.h
  Author: Emilio García Fidalgo
  Date: 29/12/06 14:38
  Description: Loading Dialog Description
=============================================================================
*/

#ifndef __INITIALDIALOG_h__
#define __INITIALDIALOG_h__

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include <wx/wx.h>
        #include <wx/dialog.h>
```

```
        #include "wx/statbmp.h"
        #include "wx/gauge.h"
#else
        #include <wx/wxprec.h>
#endif

#define InitialDialog_STYLE wxCAPTION | wxSYSTEM_MENU | wxDIALOG_NO_PARENT

class InitialDialog : public wxDialog
{
        private:

                DECLARE_EVENT_TABLE();
                void OnClose(wxCloseEvent& event);
                void CreateGUIControls();
                wxGauge *loadbar;               // Loading bar
                enum
                {
                        ID_STATIC
                };

        public:

        // Constructor

                InitialDialog(wxWindow *parent, wxWindowID id = 1,
        const wxString &title = wxT("Loading ..."),
        const wxPoint& pos = wxDefaultPosition,
        const wxSize& size = wxDefaultSize, long style = InitialDialog_STYLE);
                virtual ~InitialDialog();       // Destructor
                void setValue(int);             // Sets Gauge value
                int getValue(void);             // Return Gauge value
};

#endif
```

InitialDialog.cpp

```
/*
=============================================================================
  Name: InitialDialog.h
  Author: Emilio García Fidalgo
  Date: 29/12/06 15:44
  Description: Loading Dialog implementation
=============================================================================
*/

#include "InitialDialog.h"
#include "mkv.xpm"

// EVENT TABLE

BEGIN_EVENT_TABLE(InitialDialog,wxDialog)
```

```
        EVT_CLOSE(InitialDialog::OnClose)
END_EVENT_TABLE()

// Constructor

InitialDialog::InitialDialog(wxWindow *parent, wxWindowID id,
const wxString &title, const wxPoint &position, const wxSize& size, long style)
: wxDialog(parent, id, title, position, size, style)
{

        CreateGUIControls();
}

// Destructor

InitialDialog::~InitialDialog()
{
}

// Creating controls

void InitialDialog::CreateGUIControls()
{

        SetTitle(wxT("Loading ..."));
        SetIcon(mkv_xpm);
        SetSize(8,8,704,90);
        Center();

    loadbar = new wxGauge(this, ID_STATIC, 100, wxPoint(10, 20), wxSize(676,25), wxGA_HORIZONTAL);
}

// Close event

void InitialDialog::OnClose(wxCloseEvent& event)
{
        Destroy();
}

// Set loading bar value

void InitialDialog::setValue(int vl)
{
    loadbar->SetValue(vl);
}

// Get loading bar value

int InitialDialog::getValue(void)
{
    return loadbar->GetValue();
}
```