**ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

# "MEJORAMIENTO DE IMÁGENES SONAR OBTENIDAS MEDIANTE BARRIDO MECANICO"

**TESIS DE GRADO**

**Previo a la obtención del Titulo de:**

## INGENIERO EN ELECTRONICA Y TELECOMUNICACIONES

**Presentado por:**

**Vidal Estuardo Ayala Carabajo**

**GUAYAQUIL – ECUADOR**

**Año 2006**

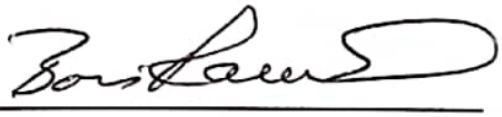
# AGRADECIMIENTO

A Aquel en quien todo consiste
A mis padres
A mis hermanos

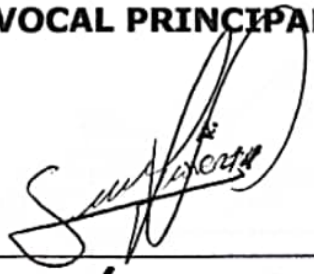
## TRIBUNAL DE GRADUACION

**Ing. Holger Cevallos U.**
**SUB-DECANO DE LA FIEC**
**PRESIDENTE**

**Ing. Pedro Vargas G.**
**DIRECTOR de TESIS**



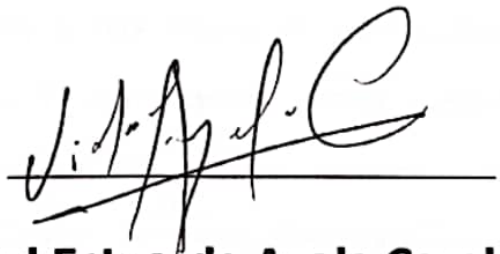
**Ing. Boris Ramos S.**
**VOCAL PRINCIPAL**



**Ing. Gómer Rubio R.**
**VOCAL PRINCIPAL**

# DECLARACION EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado me corresponde exclusivamente y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral".



**Vidal Estuardo Ayala Carabajo**

# RESUMEN

En esta tesis hemos desarrollado un conjunto de métodos de procesamiento de imágenes acústicas, con el fin de mejorar visualmente dichas imágenes y detectar automáticamente objetos en la escena que representa la imagen. El objetivo perseguido es el de incrementar las potencialidades de los sistemas sonar *haz de abanico* (fan-shaped beam) y *haz de lapiz* (pencil-beam) empleados durante operaciones submarinas.

En lo que concierne al sistema sonar *haz de abanico*, los métodos desarrollados buscan incrementar la calidad visual de las imágenes generadas. El primer método que se ha desarrollado es un procedimiento de "barrido y conversión" *(scan conversion)*, para pasar de un sistema de coordenadas polares (con el cual trabaja el sistema sonar) a una matriz de píxeles densa y regular, la cual representa la imagen. A continuación hemos desarrollado un paso de interpolación eficiente, trabajando en las mismas coordenadas polares del sistema sonar, que realiza el cálculo del valor que hay que asignar a cada píxel en base a la media pesada de las muestras acústicas vecinas. Además se ha usado una ley de asignación dinámica de brillo del píxel a fin de explotar el rango de brillo disponible. La aplicación conjunta del método de interpolación y de la ley de asignación de brillo optimizada nos ha permitido mejorar la calidad de la

imagen, proveendo así de una herramienta simple y efectiva para generar imágenes acústicas particularmente refinadas. Finalmente el desarrollo de etapas posteriores de procesamiento han permitido reducir problemas típicos de empobrecimiento de la imagen que afectan este tipo de sistemas, como el ruido speckle, los ecos múltiples y el bajo contraste.

Con respecto al sistema sonar *haz de lapiz*, desarrollamos y evaluamos 2 métodos que permiten la detección de un objeto simple contenido en la región bajo análisis. El primer método trabaja en coordenadas rectangulares y se basa completamente en la estrategia de Plantilla Correspondiente (*Template Matching*), mientras que el otro explota los datos referidos al sistema de coordenadas polares originales de los sensores sonar.

El trabajo se desarrolló a nivel de software. Se obtuvieron resultados satisfactorios en términos de precisión en la localización del objeto y en términos de carga computacional. Los requerimientos de tiempo real pueden ser fácilmente satisfechos usando un computador personal.

Las prestaciones de las técnicas propuestas han sido evaluadas usando datos reales recolectados por sensores sonar durante diferentes pruebas en el mar.

# INDICE GENERAL

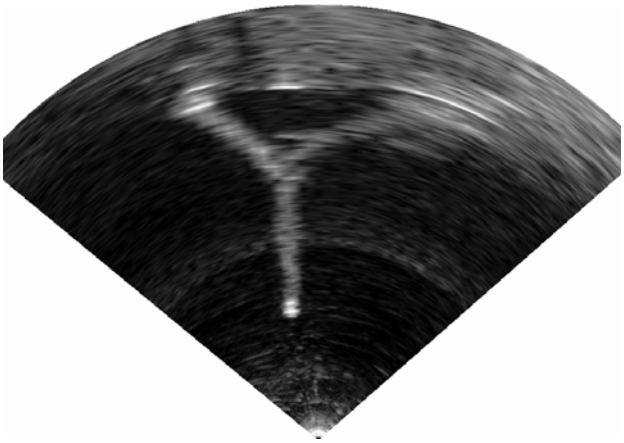
# INTRODUCCION

El procesamiento de imágenes acústicas (*Acoustic Imaging*) es un campo de investigación dirigido al estudio de técnicas para la formación y el procesamiento de imágenes a partir de señales acústicas. Estas técnicas se usan típicamente en aplicaciones que tienen que ver con investigación submarina y análisis de imágenes medicas. En general, estas aplicaciones requieren que la escena bajo investigación sea previamente insonificada por una señal acústica, y que los ecos retornados sean recogidos por el sistema acústico.

En esta tesis se generan, se analizan y se procesan dos tipos de imágenes acústicas, obtenidas mediante dos sistemas sonar de barrido mecánico diferentes: los sistemas *haz de abanico* y *haz de lapiz*. El objetivo perseguido es el de incrementar las potencialidades de estos sistemas empleados durante operaciones submarinas.

Las imágenes producidas por los sistemas sonar *haz de abanico* (conocidos como *imaging* sonar*)* son imágenes a niveles de gris que representan una sección de la escena submarina o una vista oblicua de ésta, de acuerdo a la orientación del sensor sonar. Las imágenes producidas por los sistemas sonar *haz de lapiz*

(también conocidos como *profiling* sonar), son los perfiles exteriores de las escenas estudiadas, los cuales se representan como imágenes binarias. Ver la figura 1.



*(a)*

*(b)*

*Figura 1. Imágenes acústicas. (a) Imagen a niveles de gris obtenida con un sistema acústico haz de abanico. (b) Imagen binaria obtenida con un sistema haz de lapiz*

## OBJETIVOS DE LA TESIS

Para el desarrollo de esta tesis hemos trabajado con dos tipos de datos acústicos, obtenidos con 2 sistemas sonar diferentes. El objetivo final perseguido es el de incrementar las potencialidades de estos sistemas. Los objetivos específicos, sin embargo, dependen del tipo de datos analizados.

1. Con respecto a los sistemas sonar de tipo *haz de abanico*[1], se ha intentado mejorar la calidad visual de las imágenes acústicas actuando desde las señales de "bajo nivel", es decir desde las muestras acústicas obtenidas por el transductor sonar usado en recepción. Los métodos desarrollados apuntan a 3 objetivos en este ámbito:

- Mejoramiento de la luminosidad de la imagen
- Eliminación del ruido speckle y los ecos múltiples
- Mejoramiento del contraste

2. Para los sistemas de tipo *haz de lapiz*, usados en modo dual[2], intentamos desarrollar métodos que permitan la localización automática de objetos presentes en la escena submarina barrida por el sensor sonar, a partir de los datos del perfil submarino (ver figura 1).

[1] Véase el capitulo 1: Estado del Arte
[2] Véase el capitulo 1 para mayor detalle

## CONTEXTO DE APLICACIÓN

La tesis forma parte de un gran proyecto financiado por la empresa SAIPEM S.A. con sede en Milán[3]. Fue desarrollada en Italia, en la Universidad de Génova, en el departamento de Ingeniería Biofísica y Electrónica (DIBE). Participaron en el desarrollo de la tesis, Vidal Estuardo Ayala Carabajo (quien presenta el trabajo), estudiante de la ESPOL, como parte de del programa de movilidad internacional CINDA y Matteo Garofalo, estudiante de la Universidad de Génova, bajo la tutoría en Italia del PhD. Andrea Trucco, investigador de la Universidad de Génova.

El objetivo del proyecto dentro del que se enmarca la tesis es el de extender la operabilidad de los sistemas VOR (*Vehículos Operados Remotamente*) en ambientes submarinos con escasa visibilidad debida a la turbiedad del agua, sobretodo a baja profundidad. En este tipo de ambientes el uso de las telecámaras para observar la escena presenta serias dificultades precisamente por la baja visibilidad, mientras que los sistemas sonar aprovechan las propiedades mecánicas del sonido para obtener imágenes allá donde no pueden llegar las telecámaras. El uso de sistemas sonar de alta frecuencia permite también disponer de mediciones precisas acerca de la posición y la orientación de pequeños objetos así como de otros particulares de la escena. En general, el proyecto está orientado a:

- identificar y localizar objetos y otros particulares de la escena barrida
- ejecutar manipulaciones precisas sobre partes mecánicas a través de brazos robóticas, de los cuales está dotado el VOR (figura 2).

[3] Para mayor detalle ver el sitio web de la empresa: www.saipen.eni.it

- monitorear el desplazamiento relativo entre piezas mecánicas y en particular el acercamiento de partes que posteriormente deberán ser enganchadas o soldadas, tales como tuberías.

Los experimentos a partir de los cuales se han adquirido los datos analizados fueron realizados montando el dispositivo sonar sobre un vehiculo VOR (Figura 2).

*Figura 2. Vehiculo Operado Remotamente (VOR). En la parte inferior se instalan los transductores sonar.*

En el caso de los sistemas *haz de abanico* se adquirieron datos acústicos de dos diferentes clases de objetos, una tubería suspendida verticalmente en el agua y una estructura definida como "estrella-triangulo" posada sobre el fondo marino (ver la figura 3). En ambos casos la turbiedad del agua era bastante elevada.


*(a)* *(b)*

*Figura 3. Estructuras usadas para realizar las pruebas submarinas con el sistema haz de abanico. (a) estructura "estrella-triangulo"; (b) tubería*

## SÍNTESIS DEL TRABAJO DESARROLLADO

La tesis se desarrolla nivel de software, utilizando el lenguaje C++ para crear un programa que recibe la información de bajo nivel provista por los sensores sonar (las señales muestreadas) y genera a partir de éstas un conjunto de imágenes en formato bitmap (.bmp).

La primera fase del trabajo consistió en la decodificación de los datos, a partir del formato en que son provistos por el sistema sonar. Esta decodificación consiste esencialmente en la extracción de la información necesaria (tiempos de vuelo, frecuencia de muestreo de las señales acústicas, amplitud de los ecos, ángulos de emisión, etc.) desde las señales de bajo nivel, para posteriormente construir la imagen acústica a niveles de gris en un caso (*haz de abanico*) o para generar una imagen binaria en el otro caso (*haz de lapiz*).

Partiendo de las imágenes así formadas se ha realizado un análisis visual de éstas utilizando herramientas existentes para el procesamiento de imágenes ópticas (análisis de histogramas) y se ha procedido a desarrollar un sinnúmero de funciones que actúan directamente en la formación de la imagen acústica para mejorarlas en un caso y para detectar objetos en el otro caso.

Para las imágenes obtenidas por el sistema *haz de abanico* se intentó mejorar su calidad visual mediante la aplicación de una mezcla de técnicas de procesamiento desarrolladas ad-hoc y otras convencionales. El conjunto de estas técnicas ha sido probada usando datos reales tomados mediante un sistema sonar haz de abanico durante algunas pruebas en el mar, y los resultados obtenidos han sido comparados con los resultados producidos por el sistema sonar original (el fabricante del sistema sonar provee también un software para generación de imágenes acústicas).

Para los sistemas sonar *haz de lapiz*, se intentó detectar automáticamente la presencia de un objeto simple en la escena (el tubo de la figura 3) mediante el reconocimiento de una de sus secciones (conocida *a priori*). Para lograr esto se desarrollaron dos técnicas diferentes. También en este caso, estas técnicas han sido probadas usando datos reales tomados por un sistema sonar de tipo *haz de lapiz* durante pruebas en el mar.

Cabe indicar que la investigación, el análisis de las imágenes y los resultados, y la creación del programa en C++, son fruto del trabajo en conjunto de ambos estudiantes, bajo la supervisión de los tutores.

## ORGANIZACIÓN DE LA TESIS

El **capitulo 1** describe el Estado del Arte de los sistemas sonar de barrido mecánico a alta frecuencia, explicando brevemente su división y describiendo los métodos existentes para el mejoramiento de la calidad de las imágenes acústicas y la detección de objetos.

En el **capitulo 2** se describen los métodos desarrollados en esta tesis para la generación de las imágenes acústica y para su mejoramiento.

El **capitulo 3** describe la generación del perfil submarino y los dos métodos de detección de objetos propuestos.

En el **capitulo 4** se muestran los resultados obtenidos aplicando sobre datos reales tanto las técnicas de mejoramiento de imágenes como las técnicas de detección de objetos. Se reportan comparaciones y discusiones sobre los resultados que incluyen también un análisis del peso computacional.

Finalmente el último capitulo muestra las conclusiones y delinea los posibles desarrollos futuros.

# CAPITULO 1

## ESTADO DEL ARTE

En aplicaciones submarinas el término **visión acústica** puede ser definido como el conjunto de algoritmos y métodos que apuntan a la localización y reconocimiento de objetos sumergidos (típicamente hechos por el hombre) a partir de imágenes computarizadas y, por lo tanto, a la reconstrucción e interpretación de una escena submarina. El rango máximo de investigación de las aplicaciones que permiten esta interpretación de la escena bajo observación varía de acuerdo a los sensores específicos usados y en particular a la frecuencia a la cual trabaja el sensor. Generalmente, las altas frecuencias (desde alrededor de 100 kilohertz hasta unos poco megahertz) son utilizadas para un rango de visibilidad que va desde algunos centímetros hasta 100 metros, dejando fuera todas las aplicaciones (a bajas frecuencias y rangos mas extensos) específicamente dedicadas al estudio del fondo marino y al trazado de mapas.

En este capítulo se presenta una breve clasificación de los sistemas sonar más ampliamente usados y los métodos de procesamiento de imágenes usados para extraer información a partir de los datos obtenidos, así como para reconocer la escena observada. El objetivo de este capitulo es el de dar una breve descripción del desarrollo actual de los sistemas de acústica submarina.

## 1.1 CLASIFICACIÓN DE LOS SISTEMAS SONAR

En el agua la velocidad de transmisión del sonido depende de muchos factores (salinidad, temperatura, presión, etc.). En general se puede observar, sin embargo, que ésta sufre una atenuación muy fuerte. Este fenómeno está fuertemente ligado al rango de frecuencias utilizadas. Las distancias alcanzables de transmisión a alta frecuencia, de hecho, se reducen enormemente en agua de mar: típicamente se alcanzan 50 a 200 metros. Existe por tanto una relación de compromiso entre la resolución de las imágenes producidas por un sonar y las distancias alcanzables de los mismos impulsos acústicos que la han producido: al crecer la frecuencia mejora la resolución pero al mismo tiempo disminuye el alcance del impulso transmitido. Este fenómeno es esquematizado en la Figura 1.1.1:

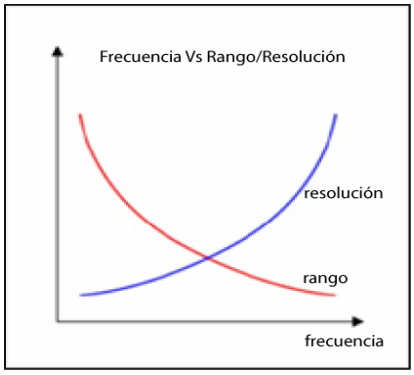



*Figura 1.1.1. Comportamiento de la resolución y del alcance en los sistemas de acústica submarina al variar la frecuencia*

Se puede hacer una primera división lógica de los sistemas sonar en dos categorías generales:

- Sistemas sonar de alta frecuencia
- Sistemas sonar de baja frecuencia

En este capitulo se describe la clasificación de los sistemas sonar de alta frecuencia, puesto que son los sistemas inherentes al trabajo desarrollado en esta tesis. Una posible clasificación ulterior de los sistemas de alta frecuencia es:

- Sonar mono-haz
- Sonar multi-haz 2D

### 1.1.1 SONAR MONO-HAZ

Los sonar ***mono-haz*** son llamados así justamente porque generan un solo haz acústico por cada impulso (*ping*) emitido. Estos sistemas son simples y generan una imagen gracias al barrido completo del sector de interés. El barrido se puede producir mediante dos formas:

1. variación del ángulo de apuntamiento del haz (barrido mecánico)
2. desplazamiento del vehículo en el cual se encuentra fijado el sonar

Normalmente la variación del ángulo de apuntamiento (llamado ángulo de *steering*) se realiza a través de una rotación mecánica, de ahí el nombre de "barrido mecánico".

Los sistemas mono-haz pueden ser subdivididos ulteriormente en las siguientes topologías:

***a) Sonar de barrido lateral***

Estos sistemas permiten posicionar el sonar sobre un lado del vehiculo que lo transporta. Tienen la capacidad de poder analizar y por lo tanto de generar imágenes de regiones bastante extensas del fondo marino. Se valen del movimiento del vehículo para generar imágenes 2D. Por cada *impulso* viene "insonificada" una sección del fondo, perpendicular al medio sobre el cual el sonar se encuentra fijado (figura 1.1.2).

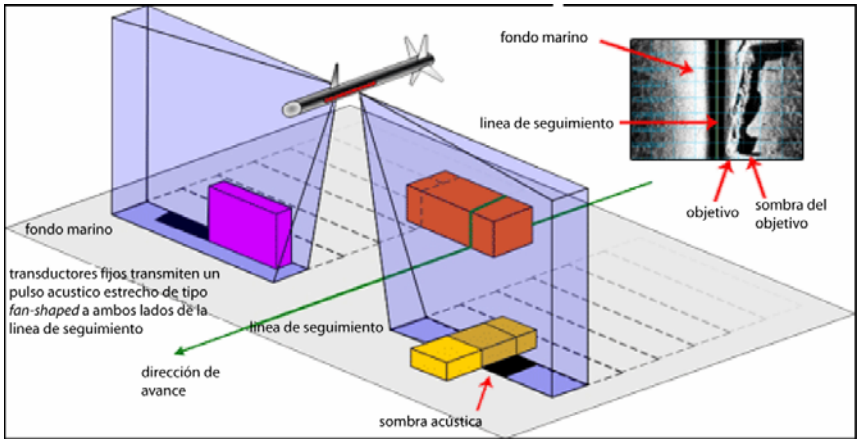



*Figura 1.1.2 Sistema sonar de barrido lateral*

**b) *Sonar de haz unico***

Estos son dispositivos usados generalmente para generar imágenes o mapas a través del barrido del fondo, caracterizados frecuentemente por una alta frecuencia de trabajo y por una buena resolución. Se pueden identificar 2 ulteriores sub-categorías dentro de esta tipología de sonar, que son además las topologías utilizadas para recoger los datos estudiados y elaborados en esta tesis:

1. ***Sonar haz de lapiz***

El haz utilizado es de forma cónica y se caracterizan por una apertura angular pequeña. Pueden ser utilizados para crear imágenes 3D, porque por cada *impulso* transmitido estos sistemas recogen la información acerca de la distancia y la fuerza reflectante de un eventual objeto alcanzado por el haz.

Estos sonar vienen comúnmente llamados **perfiladores**, porque gracias al barrido del haz sobre un plano permiten crear el perfil de la escena bajo observación (figura 1.1.3).

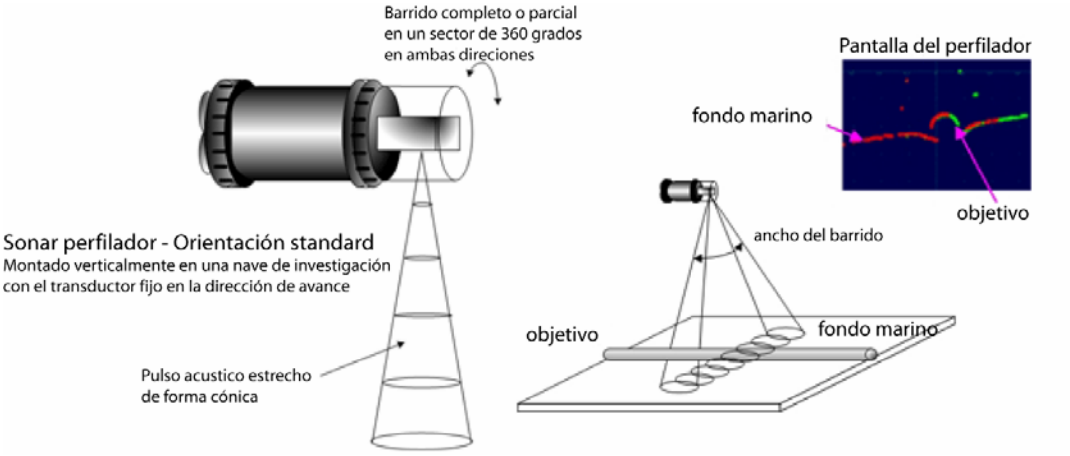



*Figura 1.1.3 Sistemas sonar haz de lapiz. El haz usado en estos sistemas es cónico y de una baja apertura angular. La pantalla a la derecha muestra el perfil obtenido con el sistema descrito*

### *2. Sonar haz de abanico*

Utilizan un haz piramidal de sección rectangular con apertura angular pequeña solo en una de las direcciones, creando un haz que podríamos definir sutil pero ancho (Figura 1.1.4). Es por esta característica que son frecuentemente utilizados como *sonar de barrido lateral*. Se conocen también cono **imaging** sonar.

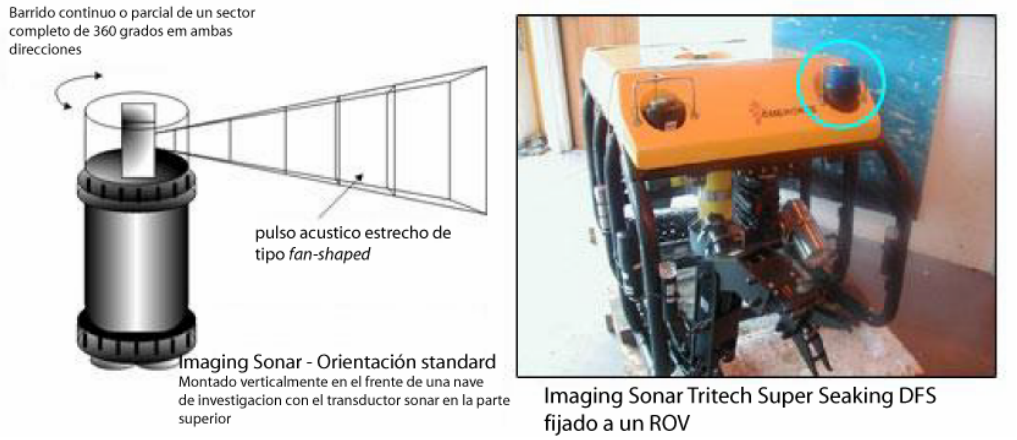



*Figura 1.1.4 Sistemas sonar haz de abanico. El haz es de tipo piramidal con sección rectangular, estrecho de un lado y ancho en el otro*

## 1.2 TRATAMIENTO DE IMÁGENES ACÚSTICAS

*Acoustic imaging* (procesamiento de imágenes acústicas) es el término que describe al campo de investigación dedicado al estudio de técnicas dirigidas a la formación y al procesamiento de imágenes generadas desde señales de bajo nivel adquiridas por un sistema acústico. Estas técnicas se usan en aplicaciones típicas que tienen que ver con investigación submarina y análisis de imágenes médicas, y son explotadas también en aplicaciones de robótica. En general, todas estas aplicaciones requieren que la escena bajo investigación sea previamente "insonificada" por una señal acústica y que los ecos retornados sean recibidos por el sistema (sensor activo).

### 1.2.1 GENERACIÓN DE IMÁGENES ACÚSTICAS

Similar a los sistemas ópticos, los sistemas acústicos pueden generar una imagen mediante el procesamiento de las ondas retornadas (ecos) por los objetos de una escena, o sea, las ondas reflejadas por los objetos "iluminados" y que regresan hacia el transductor que las ha generado. La relativa facilidad de medición del tiempo de vuelo (*time-of-flight*) de una señal acústica hace posible generar no sólo imágenes 2D similares a las ópticas, sino también estimaciones de rango que pueden ser usadas para producir un mapa real 3D. Obviamente, para iniciar el proceso, la escena debería ser "iluminada" por la emisión de un impulso acústico. Los ecos retornados pueden entonces ser procesados para crear una imagen de la escena. La operación de procesamiento del eco puede ser llevada a cabo mediante diferentes aproximaciones:

- Los **sistemas de *formación del haz (beam forming)*** recogen una sola vez los ecos retornados, mediante un arreglo de sensores. Luego los ecos son elaborados (pesados y desplazados temporalmente) de tal manera de amplificar la señal que viene de una dirección fija (*steering direction*) y de reducir todas las señales que vienen de cualquier otra dirección. La señal adquirida lleva así información acerca de la estructura de la escena solamente en la dirección de *apuntamiento*. Con el fin de formar una imagen de la escena es posible repetir la operación de *beamforming* para varias direcciones de apuntamiento del haz adyacentes.

- Los **lentes acústicos** trabajan como los lentes ópticos: los ecos retornados se enfocan en un plano donde una retina 2D de sensores transforman la imagen acústica en señales eléctricas. Gracias a la facilidad de medición del

tiempo de vuelo de un impulso acústico, uno puede generar no solo imágenes 2D sino también estimaciones en rango que pueden ser utilizadas para producir un mapa real 3D.

- Los **sistemas holográficos** parten de los ecos adquiridos por un arreglo de sensores, pero están dirigidos a la reconstrucción de la estructura de una escena mediante la re-propagación de las señales recibidas. La holografía acústica es un caso especial de difracción inversa y se realiza a través de la inversión de las ecuaciones de propagación y dispersión. Una imagen no se genera por medio de una operación de barrido, sino que el algoritmo holográfico produce toda la imagen completa al mismo tiempo.

Los métodos descritos arriba permiten la creación de una imagen 2D, así como una imagen 3D con un poco mas de elaboración. Para el caso de las imágenes 2D el método mas sencillo es el de *formación del haz*, el cual, en su caso más simple, consiste solamente en un sensor y no en una hilera de sensores. La generación de la imagen 2D se produce mediante una operación de barrido de la zona de interés. Para un mayor detalle véase (4).

La información que nos interesa de los ecos recibidos se encuentra contenida en la amplitud de la señal y en el tiempo que éstos emplean para retornar al transductor (tiempo de vuelo). Esta última información nos permite estimar la distancia a la que se encuentra el objeto. La operación de estima de la distancia puede hacerse en modos diferentes. Un método común para medir la distancia desde un objeto que ha retornado un eco es la búsqueda del pico máximo de la envolvente de la señal en recepción (*beam signal*). Otro método posible es tomar

el primer pico que excede un cierto valor de umbral, en vez de buscar el máximo pico absoluto de la señal recibida (ver Figura 1.2.1).

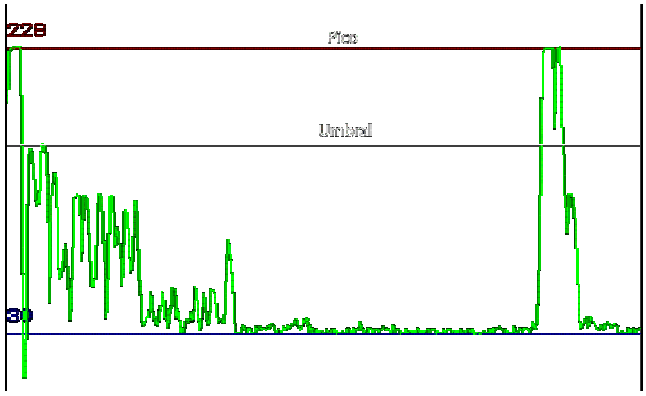



*Figura 1.2.1 Señal de bajo nivel (beam signal) obtenida por un sistema sonar. El valor del eco para cada ping puede ser tomado como el valor de pico de la señal o como el primer valor que supera un cierto umbral*

**Creación de las celdas de resolución**

La información obtenida en recepción consiste en un conjunto de señales que representan a los ecos retornados por la escena bajo análisis. En este punto es necesario clarificar como estas señales (*beam signals*) pueden ser explotadas para generar una imagen. Para este fin es necesario definir la resolución en rango y la resolución angular (3).

La ***resolución en rango*** se define como la *mínima distancia entre 2 cuerpos difusores (puestos en la misma dirección de haz), alcanzados por un impulso acústico, necesaria para poder distinguir sus contribuciones separadamente en la señal recibida.* Se debe, por tanto, poder registrar distintamente los dos ecos en

recepción (Figura. 1.2.2). La resolución en rango es inversamente proporcional al ancho de banda del pulso emitido.

$$R = \frac{v}{2 \cdot B} \tag{1.A}$$

donde *R* es la resolución en rango, *v* es la velocidad del sonido en el agua y *B* es la banda de la señal transmitida. La ecuación (1.A) es valida bajo las siguientes hipótesis: la señal transmitida debe tener una amplitud constante y la frecuencia de muestreo usada por el sistema satisface el criterio de Nyquist.

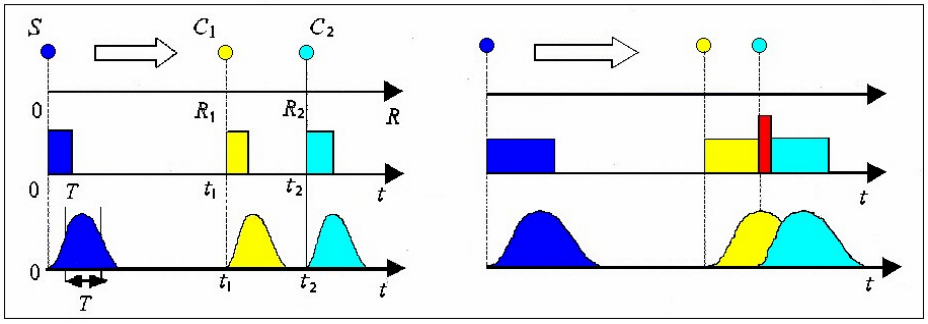



*Figura 1.2.2 Resolución en rango. El mínimo espaciamiento temporal entre 2 ecos ($t_1$ y $t_2$) se traduce en una distancia espacial dentro de la cual no es posible distinguir 2 objetos.*

Análogamente, la ***resolución angular*** es el *mínimo espaciamiento angular entre dos cuerpos difusores, entre ellos iguales y puestos a la misma distancia desde el centro del sensor, necesario para distinguir sus respectivas contribuciones*. Depende de la longitud de onda, las dimensiones del sensor (o del arreglo de sensores) y del ángulo de incidencia.

Independientemente del tipo de sistema usado para la formación de las imágenes acústicas, la información obtenida en recepción se organiza siempre en una densa cuadricula de "celdas de resolución" de diferentes dimensiones que cubren el área de interés completamente.

Una ***celda de resolución*** puede ser definida como el área limitada por las resoluciones en rango y angular dentro de la cual no es posible separar las contribuciones de los ecos (4). Evidentemente, la dimensión de las celdas de resolución no es constante en el área de interés (debido a que la resolución angular varia con la distancia), como se muestra en la figura 1.2.3.

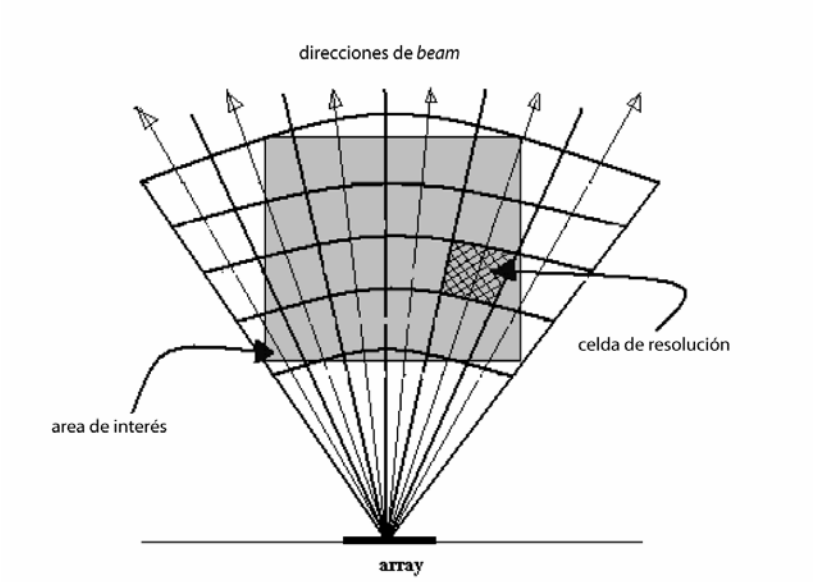



*Figura 1.2.3 Esquema de un sistema de imagen 2D: el área de interés es cubierta completamente por medio de una colección de celdas de resolución de diferentes dimensiones.*

La información esencial acerca de cada celda se encuentra en las coordenadas del centro de la celda y en la amplitud acústica o intensidad de la muestra de la señal relativa a aquel punto (proporcional a la reflectividad de la escena). Por

otra parte, las coordenadas del centro de la celda se expresan frecuentemente por medio de coordenadas polares.

**Uso de las celdas de resolución para crear las imágenes**

Las celdas de resolución pueden ser proyectadas al interno de un sistema de coordenadas cartesianas en una cuadricula regular 2D de píxeles (elementos de área) de dimensiones constantes. Esta operación de transformación entre los dos sistemas de coordenadas viene comúnmente llamado *barrido y conversion* (que será explicada al detalle en el capitulo 2). Para evitar perdidas en la resolución de los datos, cada píxel debería ser más pequeño que la más pequeña celda de resolución para permitir a uno o más píxeles ser contenidos dentro de una celda de resolución dada. Para este fin, hay dos posibles aproximaciones:

1. buscar el píxel que contiene el centro de una celda de resolución dada, asignar la amplitud acústica de la celda a tal píxel, repetir el procedimiento para todas las celdas y, finalmente, interpolar con el objetivo de asignar un valor a cada píxel que no contiene el centro de una celda;

2. calcular cuantos píxeles entran en una celda dada en base a las dimensiones de la celda, asignar la amplitud acústica de la celda a los píxeles, repetir esta operación para todas las celdas, y, finalmente, revisar si algunos píxeles están sin asignar e interpolar.

### 1.2.2 PROCESAMIENTO DE IMÁGENES ACÚSTICAS: FILTRADO, MEJORAMIENTO DE IMÁGENES, SEGMENTACIÓN

Cualquiera que sea el método de representación de la imagen adoptado, es necesaria una etapa de procesamiento preliminar con el objetivo de realizar operaciones básicas útiles para mejorar la calidad de la imagen y la comprensión humana. Estos métodos son normalmente simples y son frecuentemente utilizados para mejorar velozmente la calidad de la imagen (4). En realidad, la elección del método o del conjunto de elaboración, viene realizada de acuerdo a las características de los datos de elaborar que típicamente dependen del tipo de sensor utilizado.

Es posible, sucesivamente, aplicar otras elaboraciones ligeramente mas complicadas como el filtrado. Los filtros (como por ejemplo, los filtros medianos o los filtros pasa-bajos) utilizan "mascaras" que se sobreponen a los píxeles de la imagen: se aplican frecuentemente para reducir el ruido.

Después de realizar los métodos de procesamiento antes descritos, la segmentación y la reconstrucción se pueden llevar a cabo para identificar las regiones más significantes. Los métodos de reconocimiento o clasificación pueden ser subsecuentemente aplicados para identificar realmente los objetos de interés en una imagen, y para visualizarlos para una más fácil comprensión humana.

Un esquema de procesamiento de imágenes típico se muestra en la figura 1.2.4, donde el esquema propuesto puede ser interpretado como una división de todos los procesos en tres niveles: bajo, medio y alto. Aquí se proponen algunas aproximaciones, pero no pueden ser identificadas técnicas estándar.

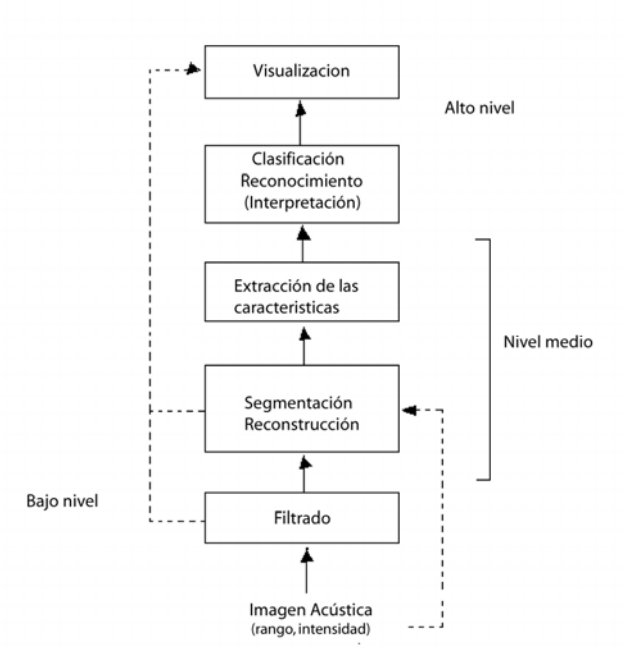



*Figura 1.2.4 Representación de tipo jerárquico para el procesamiento y la comprensión de imágenes acústicas*

En la tabla I se muestra el detalle de cada uno de los procesos.

| **FASE DEL PROCESAMIENTO** | **OBJETIVO** | **APROXIMACION** |
|---|---|---|
| Filtrado (Filtering) | Mejoramiento de la calidad de la imagen (remoción de ruido, reducción de los efectos de los lóbulos laterales, mejoramiento del contraste) | Thresholding<br>Filtros FIR<br>Restauración<br>Aproximaciones near-sensor |
| Segmentación (Segmentation) | Identificación de subconjuntos significantes de datos (regiones) | Métodos estadísticos<br>Aproximación fuzzy<br>Redes neurales<br>Análisis de textura |
| Reconstrucción (Reconstruction) | Recuperación de las superficies reales del objeto o perfil del fondo a partir de datos 3D | Métodos estadísticos<br>Métodos geométricos (fijación de superficies, procesamiento volumétrico, componentes conectados)<br>Aproximaciones de minimización de energía |

| | | |
|---|---|---|
| Reconocimiento (Recognition)<br><br>Clasificación (Classification) | Detección e identificación de objetos de interés. Asignación de un significado semántico a los datos segmentados | Reconocimiento del patrón basado en el futuro<br>Firma en frecuencia<br>Inteligencia artificial<br>Visión por computadora |
| Modelado del entorno (Enviroment Modelling) | Reconstrucción de la escena y recuperación de la posición (para aplicaciones en vehículos submarinos) | Inteligencia artificial<br>Visión por computadora<br>Registro de mapas |
| Visualización (Visualization) | Mostrar al operador | Gráficos por computadora (2D y 3D)<br>Visión por computadora<br>Realidad virtual aumentada<br>Modelado VRML |

*Tabla I. Resumen de las fases del procesamiento de datos para imágenes acústicas*

**Métodos de umbralizacion y filtrado**

En todo procesamiento preliminar, en general, es necesario eliminar ecos falsos para remover interferencias y ruidos, compensar el ángulo de incidencia de las ondas acústicas, la velocidad del sonido y la posición del dispositivo que lleva el sensor.

Algunos de estos problemas (por ej. estimación de la velocidad del sonido, posición del barco y compensación del movimiento) no se toman en cuenta en los sistemas de alta frecuencia y corto rango. En estos casos, se usan técnicas de procesamiento de imágenes para mejorar la amplitud/intensidad de la calidad de la imagen, principalmente apuntados a la reducción del ruido speckle y al mejoramiento del contraste, normalmente precedidos por correcciones geométricas.

El ruido speckle se afronta típicamente mediante la aplicación de filtros FIR (respuesta al impulso finita) de un tamaño apropiado (desde 3x3 hasta 7x7), de esta manera cada píxel es restaurado mediante la combinación lineal pesada de sus vecinos. Desafortunadamente, la aplicación de técnicas de remoción del ruido y *smoothing* usualmente lleva a desenfocar la imagen, y, por consiguiente, a reducir la información útil para la interpretación.

El primer modo y el más simple para discriminar entre los ecos reales retro-difundidos por los objetos presentes en la escena y las interferencias, es determinar un nivel de umbral apropiado. De esta manera se asume heuristicamente que los ecos tienen una respuesta mas fuerte (o diferente) que las interferencias, aun cuando esto no sea siempre cierto. Para evitar que unas pocas respuestas de amplitud muy altas produzcan un valor de umbral demasiado elevado, debidas a un efecto especular, la máxima amplitud puede ser calculada como el promedio sobre un cierto porcentaje de celdas que tienen las más altas amplitudes. Este proceso es similar a la aplicación de un filtro de umbral a una imagen, de tal manera que los píxeles bajo el umbral son ignorados y solo los píxeles sobre el umbral se visualizan.

Otro método simple consiste en el uso de filtros de tipo mascara, definiendo heuristicamente un umbral basados en la diferencia entre cada píxel y sus vecinos, pero obviamente, esta aproximación exhibe severas limitaciones y provee un bajo rendimiento. En la práctica, los filtros de este tipo generan un proceso de suavizamiento mediante el reemplazo del valor del píxel bajo examen con diferentes tipos de promedios de los píxeles vecinos, frecuentemente especificando algunos umbrales manualmente.

Los filtros medianos, también usados para imágenes ópticas, constituyen un buen método para reducir el ruido mientras preservan la información de alta frecuencia.

Los métodos de mejoramiento del contraste se pueden adoptar también para mejorar la calidad visual y facilitar procesamiento subsiguiente: los operadores Laplacianos, gradientes y otros son útiles para mejorar el contraste.

Las aproximaciones estadísticas se pueden aplicar también para medir los datos directamente con el objetivo de estimar la información real. Esta metodología consiste en el modelado estadístico del proceso físico de adquisición, y consecuentemente usando técnicas adecuadas de "inversión" para remover el ruido.

En la practica, en términos de reducción de ruido y mejoramiento de la calidad, los filtros FIR y la umbralizacion han probado permitir un buen compromiso entre complejidad computacional y desempeño; mientras que si se busca restauración de la imagen con precisión, los métodos que incluyen modelos estadísticos producen mejores resultados, pero a un costo de complejidad y carga computacional mas alto.

### 1.2.3 DETECCIÓN DE OBJETOS

Después de la formación de la imagen y el filtrado, se pueden aplicar métodos más estructurados de post-procesamiento, en particular técnicas de segmentación y reconstrucción, para tareas de alto nivel, como clasificación y

reconocimiento de objetos. Un esquema de procesamiento de datos típico se mostró en la figura 1.2.4, y el significado de algunas fases y las técnicas relacionadas se resumen en la tabla 1.1.

La naturaleza de los objetos detectados se puede estimar de algunas maneras, partiendo del procesamiento de los datos desde las señales puras hasta los algoritmos de más alto nivel. Algunos métodos se basan en análisis en frecuencia, en los cuales, la llamada "firma en frecuencia" es diferente para diferentes objetos y también para diferentes aspectos del mismo objeto, de manera que puede ser útil para discriminar entre los objetos y sus apariencias. Otros métodos se basan en una reconstrucción precisa, la cual permite la extracción de características del objeto para ser usadas para clasificación y reconocimiento de patrones.

En algunos sistemas propuestos el proceso consiste en la explotación de las características temporales extraídas directamente de una secuencia de imágenes. Esto nos permite discriminar entre ecos. Las imágenes cartesianas derivadas de los ecos retornados son primero filtradas y segmentadas usando la combinación de un procedimiento de *umbralizacion*, un filtro mediano y una técnica de *crecimiento de la region* para extrapolar la información útil. Después de la identificación de las regiones significantes, los descriptores de las figuras (longitud, área, ejes) y las características topológicas se extraen de una única imagen y son rastreadas en la secuencia. Las características temporales se extraen directamente del comportamiento de las características estáticas del objeto, usando mediciones estadísticas. Finalmente se realiza una clasificación supervisada usando funciones discriminantes lineares, las cuales son aplicadas

para clasificar las diferentes clases de entidades (Ej., buzos, cadenas, timones) presentes en la escena.

En general se puede afirmar que no existen técnicas estándar para la detección de objetos en una escena submarina. Algunas aproximaciones trabajan a un bajo nivel, es decir, a nivel de señales de *haz*, mientras que los métodos más fáciles pero menos eficientes se aplican sobre la imagen acústica ya formada.

# CAPITULO 2

## TÉCNICAS DE GENERACIÓN Y MÉTODOS DE MEJORAMIENTO DE IMÁGENES ACÚSTICAS GENERADAS POR SISTEMAS SONAR DE TIPO *HAZ DE ABANICO*

Como ya fue introducido en el capitulo precedente, los sistemas sonar de interés en esta tesis son dispositivos mas bien simples, que utilizan un barrido mecánico para variar el ángulo de incidencia del haz acústico emitido. Pertenecen a la categoría ***haz unico***, presentada en detalle en el capitulo 1, y se clasifican en el siguiente modo:

- sonar *haz de abanico*
- sonar *haz de lapiz*

Las imágenes acústicas que serán presentadas en esta tesis fueron creadas a partir de los datos adquiridos por los sistemas sonar instalados en vehículos VOR Los VOR son pequeños vehículos submarinos controlados a distancia que, entre otras características, tienen la capacidad de transportar un sistema sonar. En la figura 2.1 se muestra como ejemplo una foto de estos sistemas.

*Figura 2.1 Transductor sonar haz de abanico instalado en un vehiculo submarino VOR*

En este capitulo entraremos mas en el detalle en el conjunto de las técnicas aplicadas a los datos adquiridos por un sistema sonar de tipo *haz de abanico*.

Veremos como se ha procedido a una distribución dinámica y optimizada de los niveles de luminosidad, al desarrollo de una técnica de interpolación a partir de las señales de bajo nivel, a la reducción de ruido speckle, al énfasis de las sombras y finalmente a la eliminación de los ecos múltiples y a las colas de ecos intensos a partir de la imagen formada.

Los métodos desarrollados tienden principalmente a tres objetivos:

- mejoramiento de la luminosidad
- reducción del ruido speckle y eliminación de los ecos multiples
- mejoramiento del contraste

## 2.1 DESCRIPCIÓN DE LOS SISTEMAS SONAR *HAZ DE ABANICO*

Este tipo de sistema sonar forma parte de la categoría definida como *haz unico.* El nombre en si evidencia el hecho de que estos sonar producen, para cada ping emitido, un único haz acústico tanto en transmisión como en recepción, y son caracterizados casi siempre de una elevada frecuencia de trabajo y una buena resolución. Utilizan un haz piramidal de sección rectangular, con una apertura angular particularmente estrecha solo en una de las dos direcciones. Se forma así un haz que podríamos definir sutil pero ancho (figura 2.1.1). Este tipo de sistema viene utilizado para producir imágenes del fondo marino.

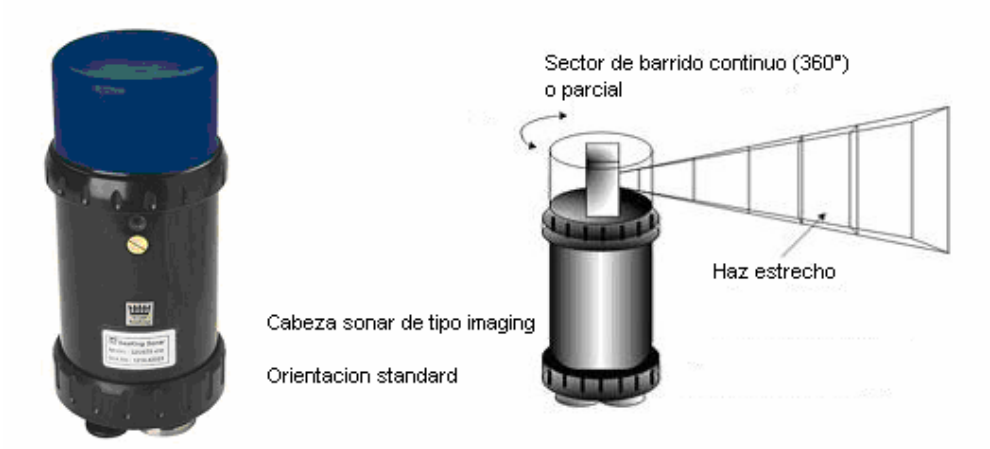



*Figura 2.1.1 Sistemas sonar haz de abanico. El haz es de tipo piramidal con sección rectangular, con apertura pequeña en una dirección y amplia en la dirección perpendicular.*

El sonar que ha sido utilizado es el *Super Seaking DFS* producido por la sociedad británica TRITECH. Este dispositivo tiene la posibilidad de trabajar a dos frecuencias diferentes: 325 y 675 Khz. Los datos utilizados en el presente trabajo

de tesis fueron adquiridos utilizando la frecuencia mas baja, que permite un rango máximo de cerca de 300 metros.

Indicamos a continuación algunos de los parámetros más importantes que caracterizan el funcionamiento de esta clase de sonar:

| | |
|---|---|
| Frecuencia de trabajo: | 325 y 675 Khz. |
| Apertura vertical del haz (beamwidth): | 20° [325] |
| | 40° [675] |
| Apertura horizontal del haz: | 3.0° [325] |
| | 1.5° [675] |
| Rango máximo: | 300 m [325] |
| | 100 m [675] |
| Rango mínimo: | 0.4 m |
| Resolución en rango: | 5 - 40 m |
| Anchura del impulso transmitido: | 20 – 300 mseg |
| Tamaño de paso mecánico: | 0.225°, 0.45º, 0.9º, 1.8º |
| Sector investigado: | hasta 360° |

## 2.2 ORGANIZACIÓN DE LA INFORMACIÓN Y EXTRACCION DE LOS DATOS

Todos los métodos de elaboración, ideados y propuestos en la presente tesis, son verificados por la creación de un código en C++ desarrollado utilizando Microsoft Visual Studio C++ 6.0 en ambiente Windows que permite la generación de imágenes, a partir de la información recogida por los transductores sonar. Estas informaciones están presentes en las señales de bajo nivel

contenidas en los archivos de extensión ***v4log*** que han sido puestos a nuestra disposición.

Las señales de bajo nivel adquiridas por ambas topologías de sonar son una colección de señales relativas a cada ángulo de apuntamiento. Los datos relativos a estas señales están contenidos en los archivos de extensión v4log. Los archivos de extensión v4log son llamados comúnmente "log file".

Todo el proceso de adquisición de los datos a partir de los *log file* se efectua en la primera parte del código desarrollado en C++, en el cuerpo principal del programa, dentro de la función ***void OnBtnProcess()*** contenida en CSeaNetLogExtractDlg.cpp (ver el apéndice A).

**Estructuras de los archivos *log***

Para el sistema sonar de tipo *haz de abanico*, el formato de los archivos es tal de poder memorizar en formato binario todas las señales de bajo nivel adquiridas. El *log file* está formado por el encabezado llamado ***log header*** (80 bytes), seguido de una secuencia de encabezados adicional llamados ***rec header*** (46 bytes), que están asociados a grupos de datos y llevan la información necesaria para la correcta interpretación de los datos. La estructura del *log file* está ejemplificada en la figura 2.1.2.

El *log header* es único para cada archivo, y cada archivo puede contener las informaciones relativas a muchas imágenes. Al interno del *log header* se memoriza información acerca de, por ejemplo, la versión del software de

adquisición, punteros al inicio de los datos, al inicio del *registry key*, y al inicio de eventuales opciones (*extra-data*). Está también presente un campo *checksum* para el control de los errores.

El *registry key* es un archivo textual colocado casi al inicio del *log file*: en éste se guarda información fundamental para la correcta interpretación de los datos adquiridos, como la posición del sonar (posición relativa al medio de transporte). El campo *extra-data*, en cambio, contiene información adicional no estrictamente utilizada para la creación de la imagen: entre otras hay un temporizador inicializado en el momento de la emisión del haz (*beam*), la duración del *ping*, e informaciones relativas a las frecuencias utilizadas.

Se reporta aquí a continuación, una tabla resumida de los campos constituyentes del ***log header***:

| **BYTE** | **DESCRIPCIÓN** | **EJEMPLO** |
|---|---|---|
| 1-32 | código ASCII para una prueba sobre el header | "SeaNetV4 Log File…." |
| 33-36 | Versión | 10H (=1.0) |
| 37-40 | Offset del registro | 50H (=80). Offset start |
| 41-44 | Offset de los datos | 1346DH (=78957) |
| 45-48 | Datos registrados | 657H(1623)n° datos registrados |
| 49-52 | Offset de configuración | 816E7H (=530151) |
| 53-56 | Offset de los *extra-data* | 8180AH (=530442) Posición de los *extra-data* |
| 57-60 | Offset del índice | 8180AH (=530442) |
| 61-64 | Offset del *checksum* | 8180AH (=530442) |

| | | |
|---|---|---|
| 65-72 | Tiempo de apertura del LogFile | n° de días desde el 30/12/1899 |
| 73-80 | Tiempo de cerrado del LogFile | n° de días desde el 30/12/1899 |

*Tabla II . Descripción de los campos contenidos en el Log Header*

Todos los campos *offset* son usados para encontrar el inicio de ciertas secciones al interno del archivo: su valor puede ser interpretado como el número de bytes que distancian la información buscada desde el inicio del archivo.

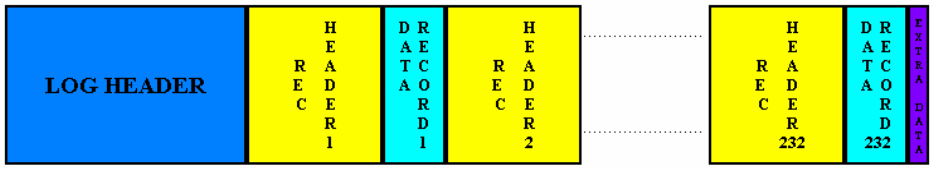



*Figura 2.1.2 Estructura del log file para los datos adquiridos por un sistema sonar haz de abanico.*

En el campo *data record* son memorizados los datos verdaderos, que consisten en las muestras acústicas obtenidas luego de la operación de emisión del haz y adquisición de los ecos. En la figura 2.1.2 se puede notar como cada *data record* está precedido del propio ***rec header*** que transporta información especifica solo de una determinada dirección de apuntamiento.

En la tabla III se muestran al detalle las informaciones contenidas en el ***rec header*** y como éstas vienen memorizadas byte por byte:

| BYTE | DESCRIPCIÓN | EJEMPLO |
|---|---|---|
| 1,2 | Longitud del mensaje binario | 0171H = 369 bytes |
| 3-10 | Fecha | n° de días desde el 30/12/1899 |
| 11 | Nodo transmisor | Siempre 02 para Single Sonar |
| 12 | Nodo receptor | Siempre FFH |
| 13 | Datos sonar | Siempre 02H |
| 14 | Secuencia de paquetes | Siempre 80H |
| 15 | Numero del nodo | Copia del byte 11 |
| 16,17 | Longitud del mensaje binario – 15 | 0162H = 354 bytes |
| **18** | **Tipo de transductor** | 02H = Sonar |
| **19** | **Estado** | 90H |
| 20 | código de barrido | 0H (0=barrido normal, 1=al limite izquierdo, 2=al limite derecho) |
| **21,22** | **Head control** | 2103H |
| **23,24** | **Escala del rango en decímetros** | 05H = 80 decímetros |
| 25-28 | Portadora del transmisor | 05666666H |
| 29 | Ganancia (/255 unidades) | 54H = 84 = 40% |
| 30,31 | Slope | 007DH = 125 |
| 32 | A-D del receptor | 70H = 112 |
| 33 | A-D Low del receptor | 10H = 16 |
| 34,35 | *Offset* del *header* | 0 = ignorar |
| 36,37 | Intervalo AD | 2EH = 46 |
| **38,39** | **Limite izquierdo (1/16 de grado)** | 01H = 1 |
| **40,41** | **Limite derecho (1/16 de grado)** | 18FFH = 6399 |
| **42** | **Intervalo de paso del motor** | 10H = (en 1/16 de grado) |
| **43,44** | **Angulo del transductor** | 05B0H = 1456 (en 1/16 de grado) |
| **45,46** | **n° de bytes de los datos** | 00E8H = 232 bytes de datos |

| | | |
|---|---|---|
| **47** | **1° Byte de datos** | 2CH = 00101100 (1° bin) |
| **48** | **2° Byte de datos** | 66H = 01100110 (2° bin) |
| **49** | **3° Byte de datos** | 6FH = 01101111 (3° bin) |
| **..** | | |
| **278** | **232° Byte de datos** | 20H = 00100000 (232° bin) |
| **279** | **Datos extra (*extraDetects mensaje*)** | |

*Tabla III. Parte de un archivo v4log. Contenido del rec header (hasta el byte 46) y datos relativos asociados: data record (desde el byte 47 al 278).*

En el *rec header* está presente la información adicional necesaria para desarrollar la operación de creación de la imagen: sin ellas no sería posible el uso y la interpretación de los datos memorizados. Como es posible ver de la Tabla 2.1.2, algunas de estas informaciones son: el tipo de transductor sonar (para permitir distinguir entre los diversos tipos de sonar existentes), la frecuencia de la portadora, los ángulos de apuntamiento utilizados, el paso mecánico del ángulo de barrido.

Informaciones particularmente importantes, contenidas en el *rec header*, para el uso y la elaboración de los datos son memorizados al interno del campo *head controls* (bytes 21 y 22). Por este motivo detallamos en la tabla IV la estructura de estos bytes, bit por bit.

| **BIT** | **NOMBRE ASOCIADO AL BIT** | **INFORMACIÓN QUE LLEVA** |
|---|---|---|
| Bit 0 | Adc8on | 0=muestras representadas por 4 bits, 1=a 8 bits |
| Bit 1 | Cont | 0=Barrido de un sector, 1=Continuo |
| Bit 2 | Scanright | Dirección de barrido (0=izquierda, 1=derecha) |

| **Bit 3** | **Invert** | **1=transductor sonar montado invertido** |
|---|---|---|
| Bit 4 | Motorff | 1=motor de barrido apagado |
| Bit 5 | Txoff | Bit para test en la transmisión |
| Bit 6 | Toggleadcmux | Usado para sonar especial |
| Bit 7 | Chan2 | Especifica cual de las 2 frecuencias es usada |
| Bit 8 | Raw | Bit de control |
| Bit 9 | Hasmot | 1=existe un motor para el barrido |
| Bit 10 | Applyoffset | Permite cambiar la dirección de barrido |
| Bit 11 | Pingpong | 1=es posible alternar entre las 2 frecuencias |
| Bit 12 | StareLLim | 0 (default)=la dirección de barrido no es fija |
| Bit 13 | ReplyASL | Bit de default=1 |
| Bit 14 | ReplyThr | 0; Reservado para funciones especiales |
| Bit 15 | IgnoreSensor | Bit utilizado para diagnostico |

*Tabla IV . Contenido de un campo del rec header llamado head control.*

Resulta evidente que el conocimiento de algunos de estos bits es fundamental para la correcta interpretación de los datos. Por ejemplo el bit *invert* (bit3) permite entender si el transductor sonar se encuentra montado hacia el fondo marino o en dirección de la superficie del agua, el conocimiento del bit *Adc8on* (bit0) aclara cuantos bits vienen empleados para representar cada muestra.

**Modalidad de adquisición de los datos**

Completamos esta breve introducción a los sistemas sonar ejemplificando en que modo estos adquieren los datos relativos a un escenario de interés.

Para cada dirección de apuntamiento el transductor sonar emite una señal acústica de breve duración (*ping*). El transductor en recepción muestrea a intervalos regulares la señal retrodifundida, come se muestra en la figura 2.1.3. La amplitud de cada muestra viene memorizada normalmente utilizando 8 bits y este dato se almacena en el campo *data record* (bytes del 47 al 278 visibles en la tabla III).

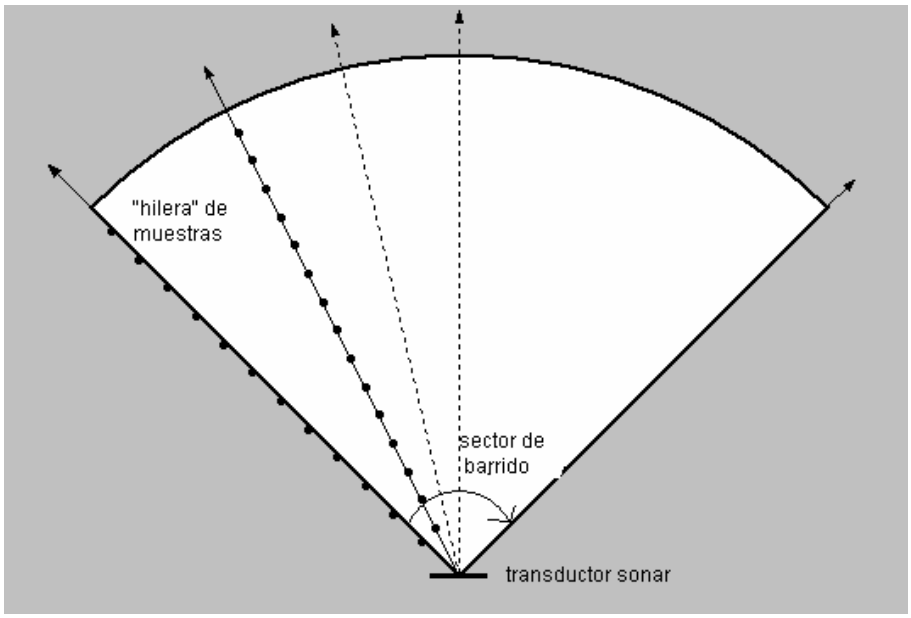



*Figura 2.1.3 Envío de pings a lo largo de diversas direcciones de barrido y muestreo de la señal de retorno.*

Esta operación hace posible memorizar una serie de muestras relativas a un determinado ángulo de apuntamiento. Por cada "hilera" (conjunto de muestras a lo largo de una determinada dirección) se memoriza un *rec header*.

Cada muestra, siendo representada por 8 bits, está identificada en decimales con un número entre 0 y 255: durante la operación de creación de la imagen tales valores serán usados directamente para asignar los píxeles de la imagen.

Como consecuencia los datos recogidos son suficientes para crear una imagen acústica a niveles de gris, cuya resolución dependerá de la frecuencia de muestreo y del rango en una dimensión, y de la apertura del haz emitido en la otra.

Se ha visto así cuales son las informaciones necesarias para la creación de una imagen acústica y como los datos adquiridos por un sistema *haz de abanico* son suficientes para la creación de una imagen del fondo o de los eventuales objetos que se encuentran en dirección de la proa del medio.

El proceso de creación de la matriz de imagen involucra todos los pasos que pasaremos a describir. Sin embargo, la creación de la imagen en formato bitmap se realiza al final de todo el proceso mediante la función ***int SalvaBitmap (const char *filename, int **immagineFinale, int profilatore)*** que se encuentra en SeaNetLogExtractFunzioni.cpp

## 2.3 TÉCNICAS PARA EL MEJORAMIENTO DE LAS IMÁGENES

Los datos puestos a disposición por la SONSUB para el desarrollo de esta tesis son datos relativos tanto a experimentos en el mar como a pruebas y tests desarrollados en lagos y piscinas: ha sido posible así probar muchas de las configuraciones y las funcionalidades del aparato sonar.

En el caso de los sonar de tipo *haz de abanico* el sistema ha adquirido datos relativos a la presencia de dos tipos de objetos diferentes, una tubería suspendida verticalmente en el agua y una estructura definida como "estrella-

triangulo" posada sobre el fondo marino (ver la figura 3 de la introducción). En ambos casos la turbiedad del agua era muy elevada.

Una visión esquemática de todas las elaboraciones que se han ejecutado sobre las señales y luego sobre las imágenes producidas, es la siguiente:

- gestión dinámica y optimizada de los niveles de luminosidad
- *barrido y conversion* e interpolación
- reducción del ruido *speckle* (filtro de Frost)
- mejoramiento del contraste (*contrast enhancement*)

Las imágenes producidas por el sistema sonar *haz de abanico* son imágenes a niveles de gris que representan una sección de la escena o una vista oblicua de esta, de acuerdo a la orientación del transductor sonar. Para este tipo de imágenes ha sido desarrollado un conjunto de técnicas dirigidas al mejoramiento de la calidad visual. Tales técnicas pueden ser subdivididas en dos categorías según el nivel al cual los datos son elaborados.

El primer conjunto de técnicas trabaja sobre las señales de bajo nivel (es decir de las señales generadas por el transductor sonar usado en recepción). Pertenecen a esta categoría la gestión dinámica y optimizada de los niveles de luminosidad y la interpolación basada en la media pesada de las muestras vecinas. Estas operaciones son realizadas antes de la creación de la imagen.

El segundo conjunto de técnicas obra directamente sobre la imagen. Para formarla es necesario usar un procedimiento de *barrido y conversion* para pasar desde el sistema de coordenadas polares del transductor sonar a una cuadricula regular de píxeles. Pertenecen a este conjunto la técnica dirigida a la reducción del ruido speckle, basada en el uso del filtro Frost (1) y una técnica dirigida almejoramiento del contraste (2) basada en un método de mejoramiento del contraste propuesto recientemente en la literatura para imágenes ópticas.

### 2.3.1 GESTIÓN DINÁMICA Y OPTIMIZADA DE LOS NIVELES DE LUMINOSIDAD

Los datos adquiridos por un sistema sonar *haz de abanico* son almacenados en los archivos log. En estos archivos vienen memorizadas las muestras relativas a cada dirección de barrido. Se tienen así múltiples "hileras" de muestras. Cada muestra es un número de 8 bit y representa en el sistema decimal un valor de intensidad (reflejada en el transductor) entre 0 y 255: se pueden representar en total 256 niveles de amplitud. Cada uno de estos valores de intensidad debe ser interpretado en una escala lineal y, durante la operación de creación de la imagen, puede ser interpretado como un nivel de gris.

El primer procesamiento, dirigido al mejoramiento de la representación visual de estos datos, ha sido justamente una optimización dinámica de la distribución de estos niveles de intensidad. El termino "dinámica" se usa para indicar la capacidad del software de ejecutar este "ajuste" de los niveles de intensidad originales en modo optimo para cada imagen. Esta elaboración está dirigida, por

tanto, a modificar los valores de intensidad para hacer que estos se dispongan en modo tal de ocupar mejor todo el rango de valores que estos pueden asumir (0 – 255). La función en la que se implementa esta operación es la siguiente: ***void sogliatura (int **matrice)***. Es necesario, sin embargo, un paso previo de creación de una matriz primitiva que represente la imagen, cuyos valores son los niveles de gris obtenidos de los archivos *log*. Tal proceso lo realiza principalmente la función ***int **matriceCampioni(int nimmagine, int *dati)***. Ambas funciones se encuentran en el archivo SeaNetLogExtractFunzioni.cpp (ver apéndice A).

Gráficamente la operación de ajuste de los niveles de luminosidad puede ser interpretada como una función que asigna a cada valor de intensidad en ingreso ($z_{in}$) un valor de intensidad en la salida ($z_{out}$). El problema es encontrar la función que permita tener al final los mejores resultados en la distribución de los valores.

Los datos en ingreso son codificados en una escala de 0 a 255, donde 255 representa el eco de mayor intensidad. En primer lugar es necesario calcular el valor de ingreso más pequeño (LOW) y el más grande (HIGH). LOW es calculado como el mínimo valor presente en los datos en ingreso y, experimentalmente es siempre igual a cero. El cálculo de HIGH se realiza de la siguiente manera:

1. se ordenan los valores de intensidad originales (en ingreso) desde el más pequeño al más alto. Esta operación fue realizada a través de un algoritmo de ordenamiento de tipo ***OrdenamientoRapido (QuickSort)***.

2. se escoge el número de valores sobre los cuales efectuar la media (ver el punto 3) en el modo siguiente:

$$num\_el = \frac{N}{100}$$

donde N es el número total de datos en ingreso; se ha elegido después de muchos experimentos calcular la media sobre el 1% de los valores en ingreso.

3**.** se calcula el valor medio (*mean*) de los *num_el* valores en ingreso mas altos.

4**.** por ultimo HIGH se obtiene como el entero más cercano al valor medio calculado:

HIGH= (int) mean

De esta manera los valores en ingreso iguales a LOW vienen puestos a cero, mientras que aquellos mayores de HIGH vienen puestos a 255. En cuanto a los valores comprendidos entre LOW y HIGH, estos vienen mapeados en el intervalo 0-255 según la siguiente relación:

$$z_{out} = 255 \cdot \left( \frac{z_{in} - LOW}{HIGH - LOW} \right) \qquad (2.A)$$

Veamos gráficamente el comportamiento de la curva de asignación que sigue la ley de entrada-salida descrita por la ecuación (2.A):

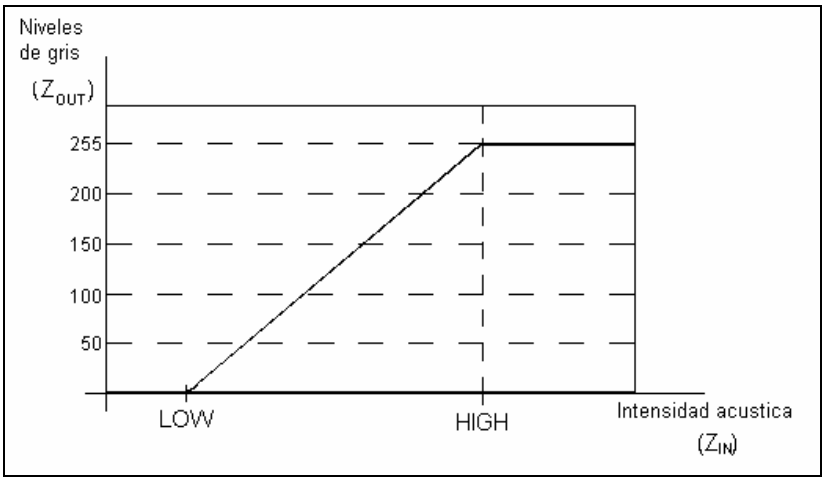



*Figura 2.2.1 Curva de asignación entrada-salida descrita por la formula (2.A): ley de asignación lineal.*

Como resulta evidente tanto del grafico como de la ecuación (2.A) se ha procedido a una asignación lineal de los niveles de intensidad de salida ($z_{out}$) en relación con aquellos de ingreso ($z_{in}$).

Una asignación de este tipo de los niveles de gris en el rango dinámico no es la asignación que valoriza más la información a disposición. Una curva diversa puede mejorar la calidad visual de la imagen.

Experimentalmente se ha observado como se tienen resultados mejores escogiendo una ley de asignación diferente. Se han probado muchas posibles modificaciones de la curva de entrada-salida: la ley de asignación que permite obtener los mejores resultados se expresa en la ecuación (2.B) y es visible en la figura 2.2.2:

$$z_{out} = 255 \cdot \left( \frac{z_{in} - LOW}{HIGH - LOW} \right)^{\gamma} \qquad (2.B)$$

donde γ es un parámetro con valores reales positivos. γ = 1.7 ha sido considerada una buena elección en base a la calidad media de los resultados obtenidos.

Es importante notar como esta nueva formulación de la ley de asignación es general y comprende también el caso precedente: es suficiente fijar el valor de $\gamma$ en 1 para obtener de nuevo la ecuación de la recta (2.A). En síntesis se puede afirmar que poner $\gamma=1$ produce un efecto de estrechamiento lineal, $\gamma<1$ enfatiza la luminosidad de la imagen, mientras que un valor de $\gamma>1$ atenúa la luminosidad de la imagen, en modo especial en relación a los valores centrales del intervalo LOW÷HIGH.

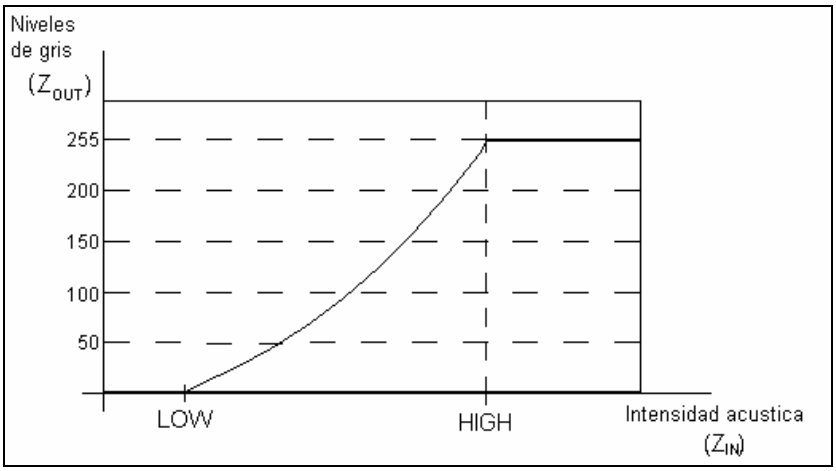



*Figura 2.2.2 Curva de asignación entrada-salida descrita por la formula (2.B).*

### 2.3.2 *BARRIDO Y CONVERSION*

Prescindiendo del tipo de sistema utilizado para la formación de la imagen acústica, la información a disposición se organiza en una cuadricula densa de

celdas de resolución de dimensiones diferentes que van a cubrir el área entera de interés. Lo que es esencial en cada celda son las coordenadas de tal celda y la amplitud o intensidad.

El sonar memoriza la información sobre la intensidad del eco de retorno y para cada muestra provee también la información sobre el ángulo de incidencia y sobre el periodo de muestreo de la señal (en los *rec header*). Precisamente esta última información permite calcular la distancia a la cual se refiere el valor de muestra memorizado, dada la velocidad de propagación del sonido en el medio (v):

$$\rho = \frac{v \cdot T}{2} \qquad (2.C)$$

Supongamos que tenemos la i-ésima muestra después de T segundos. T, por lo tanto, será el tiempo transcurrido desde el envío del *ping* hasta el instante de muestreo: es el tiempo que emplea la onda acústica para recorrer la distancia $\rho$ y regresar.

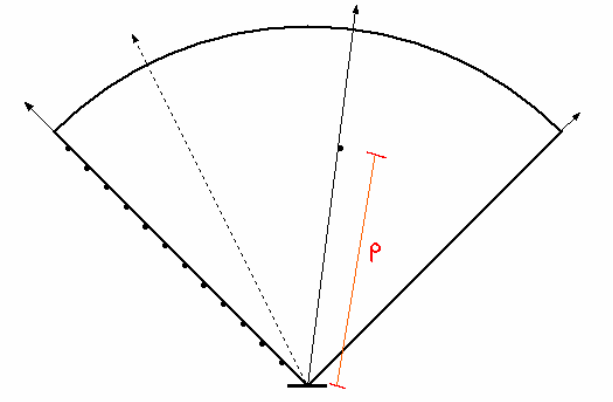



*Figura 2.2.3 La distancia $\rho$ de la muestra desde el transductor se deduce fácilmente del tiempo T empleado por la onda para ir y regresar al sensor sonar.*

Se ha elegido llamar $\rho$ a la distancia porque es natural pensar en estos datos en el dominio de coordenadas polares. $\rho$ (rho), por tanto, puede ser identificada como la información de distancia del objeto reflectante desde el sensor y $\theta$ (theta) como la información sobre el ángulo de incidencia.

Las coordenadas del centro de las celdas, precisamente por las razones descritas arribas, son expresadas en coordenadas polares y se hace necesaria una operación de *barrido y conversion* para pasar a una malla regular de píxeles. La función que realiza el proceso se define como ***int **scan_conv(double **vettori,int **matriceCampImm)***. Es además indispensable una técnica de interpolación con el fin de obtener una malla densa de píxeles. Se pueden identificar en esta operación dos pasos:

1. las coordenadas cartesianas (x, y) de todos los píxeles de la pantalla vienen expresadas en coordenadas polares, considerando ambos sistemas de referencia con origen en el vértice del sector circular, que no es otra cosa que la posición del transductor [la operación viene realizada por la función ***double **posizioneMetri (double *ping, int start, int nimmagine, double Ts,int *tipo)***] (figura 2.2.4). Las nuevas coordenadas de los píxeles del monitor son denominadas $(\hat{\rho},\hat{\theta})$. Tales coordenadas en general no tendrán una correspondencia directa con las coordenadas (ρ,θ) relativas a los datos (muestras) a disposición (figura 2.2.4);

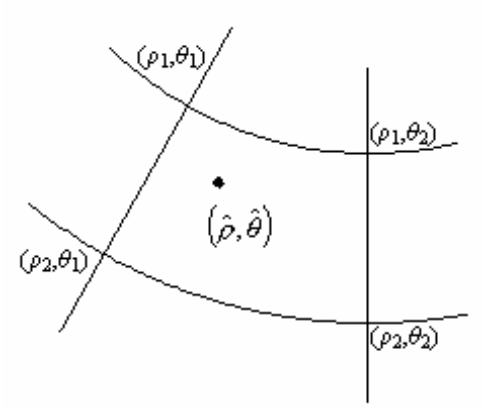



*Figura 2.2.4 El punto de coordenadas $(\hat{\rho},\hat{\theta})$ corresponde a un píxel del monitor, mientras que los cuatro puntos a su alrededor son puntos de los cuales se conocen los valores de amplitud, pero para los cuales, en general, no se tiene una correspondencia directa con un píxel.*

2. al punto de coordenadas $(\hat{\rho},\hat{\theta})$ viene asignada una amplitud calculada como la media pesada de las cuatro muestras mas cercanas a éste, por medio de la ecuación (2.D) y según lo representado en la figura 2.2.5:

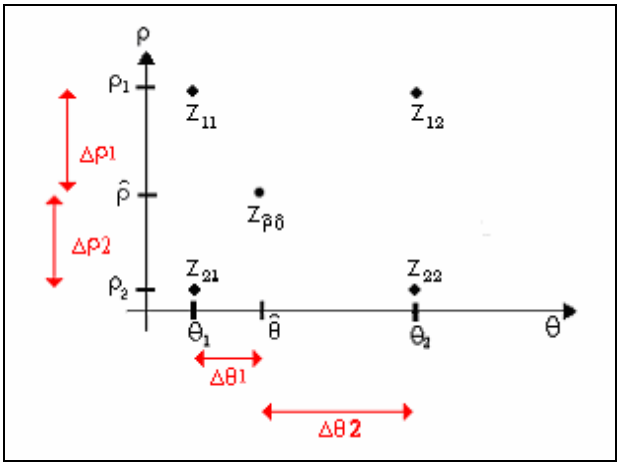



*Figura 2.2.5 Esquematización de la técnica de interpolación basada en la media pesada de las muestras vecinas.*

$$z_{\hat{\rho}\hat{\theta}} = \frac{\dfrac{z_{11}\Delta\theta_2 + z_{12}\Delta\theta_1}{\Delta\theta_1 + \Delta\theta_2}\Delta\rho_2 \; + \; \dfrac{z_{21}\Delta\theta_2 + z_{22}\Delta\theta_1}{\Delta\theta_1 + \Delta\theta_2}\Delta\rho_1}{\Delta\rho_1 \; + \; \Delta\rho_2} \qquad (2.D)$$

y, en general, tendremos: $\Delta\theta_i = \left|\theta_i - \hat{\theta}\right|$ y $\Delta\rho_i = \left|\rho_i - \hat{\rho}\right|$, donde en el caso tomado en consideración en la figura 2.2.4 y en la ecuación (2.D), i = 1,2.

Lo que ha sido descrito en el precedente punto 2 va aplicado a todos los píxeles del monitor que, expresados en coordenadas polares, son tales de tener cuatro muestras cercanas. Esto significa que los píxeles correspondientes a los bordes no vienen asignados, o sea asumen un valor igual a cero.

La aplicación conjunta de la optimización de la asignación de los niveles de gris y de esta técnica de interpolación estudiada *ad-hoc*, permite la generación de una imagen de calidad superior a aquella producida originalmente, mejorando también la comprensión humana de la escena investigada.

### 2.3.3 REDUCCIÓN DEL RUIDO SPECKLE (APLICACIÓN DEL FILTRO DE FROST)

Al final de la operación de *barrido y conversion,* se ha elaborado y organizado, por tanto, la información en una imagen a niveles de gris. Las imágenes acústicas, a causa de las dificultades de los escenarios submarinos en los cuales vienen adquiridos los datos, están muy sujetas al ruido.

El primer procesamiento que se ha realizado sobre la imagen ha sido la eliminación del ruido speckle. Este tipo de ruido aflige muy a menudo las imágenes acústicas, creando altos valores de intensidad en puntos de la imagen en los cuales no deberían estar presentes. Este fenómeno repercute visiblemente sobre la imagen con la aparición de muchos puntos que en realidad son "falsas alarmas".

Se ha aplicado una aproximación dirigida a la reducción del ruido speckle, basada en el filtro de Frost (1). La operación de reducción del ruido speckle se produce haciendo pasar sobre la imagen formada (o sea sobre la matriz que ahora corresponde a la imagen) una mascara de dimensiones $(2\cdot N)+1 \times (2\cdot N)+1$ y calculando la información estadística de todos los píxeles contenidos en la mascara, como la media y la varianza local. La asignación del parámetro N viene realizada al inicio de la función misma: los valores de N que han sido tomados en consideración son 1, 2, 3. Al aumentar N aumenta el peso computacional y por tanto el tiempo necesario para la ejecución de esta operación. El aumento de N tiene repercusiones también en la calidad de los resultados. En particular, a un aumento de N corresponde ciertamente una reducción del ruido speckle, pero tal elección viene acompañada de un efecto de "desenfoque" de la imagen y por tanto de una perdida de nitidez sobretodo en los detalles. Por tal razón, es importante fijar el valor de N en modo tal de obtener un buen compromiso entre reducción del ruido speckle y mantenimiento de los detalles y, sobre la base de la calidad media de los resultados obtenidos, el mejor compromiso se ha considerado alcanzado escogiendo N=2.

En el código este filtro ha sido implementado mediante la función ***void filtroFrost (int **matrice)***. La operación básica de este proceso consiste en calcular el nuevo valor que hay que asignar al píxel central de la mascara ($\mathbf{r_{ij}}$), valor que depende de la información estadística local. En el filtro de Frost, el valor a asignar al píxel central es calculado a través de la siguiente formula:

$$r_{IJ} = \frac{\sum_{i=I-N}^{I+N} \sum_{j=J-N}^{J+N} \left( z_{ij}\, m_{ij} \right)}{\sum_{i=I-N}^{I+N} \sum_{j=J-N}^{J+N} m_{ij}} \qquad \text{(2.E)}$$

donde $\mathbf{z_{kl}}$ es el valor de cada píxel que se encuentra en la mascara y $\mathbf{m_{kl}}$ es el coeficiente de peso para cada píxel de la mascara.

$$m_{ij} = \exp\left(-A\ T_{i-I,\, j-J}\right) \qquad \text{(2.F)}$$

donde $\mathbf{T_{k-i,\, l-j}}$ es el valor absoluto de la distancia, en píxeles, desde el píxel central (i, j) a sus vecinos (k, l) al interior de la mascara, y

$$A = D\left(\frac{\sigma}{\mu}\right)^2 \qquad \text{(2.G)}$$

donde **σ** y **μ** son respectivamente la desviación standard y la media local (dentro de la mascara); **D** fue puesto a 1 como ha sido sugerido en la mayor parte de los casos tratados en la literatura. Grandes valores de **D** preservan mejor los contornos pero reducen el efecto "suavizado" (*smoothing*), mientras que pequeños valores de **D** aumentan el *smoothing* pero no preservan los contornos.

**D** = 0 corresponde a un filtro medio, donde los coeficientes de peso para cada píxel perteneciente a la mascara son constantes e iguales a 1/ [(2N+1) · (2N+1)].

Ves necesario afirmar que la calidad de las imágenes analizadas es ya muy buena: la elaboración efectivamente efectuada por el filtro de Frost se mantiene apreciable, aun cuando con frecuencia no es particularmente evidente.

### 2.3.4 MEJORAMIENTO DEL CONTRASTE

Para obtener una imagen más nitida en donde se logren diferenciar los objetos presentes del fondo, es necesario mejorar el contraste.

Para este fin se aplicó una técnica de *mejoramiento del contraste* recientemente propuesta en literatura para imágenes ópticas (2). Este método de mejoramiento del contraste no es común para aplicaciones que conciernen a imágenes acústicas, pero se ha relevado en cambio como un método muy potente y capaz de proveer mejoramientos de la calidad de la imagen muy evidentes aun a primera vista.

Este método se basa en una particular ecualización del histograma que permite elaboraciones muy potentes. La idea de base de esta nueva aproximación es la de buscar de aprovechar los efectos positivos tanto de las aproximaciones basados en la ecualización del histograma global de la imagen (HE), como de la ecualización adaptiva (AHE) que trabaja dividiendo las imágenes en secciones.

Utilizando la primera aproximación (HE) se podrían tener áreas de la imagen en las cuales el contraste incluso empeora causando el oscurecimiento de algunas particulares zonas, aunque el nivel visual de la elaboración completa no resulta desagradable. Aplicando el segundo, en cambio, se valorizan los detalles de cada área tomada en consideración, pero puede surgir el problema opuesto: evidenciar exageradamente el contraste de algunos particulares de la imagen, haciéndola parecer muy artificial.

Este método particular de mejoramiento del contraste, con el fin de conciliar las dos aproximaciones clásicas (HE y AHE), tiene necesidad de extrapolar tanto las informaciones estadísticas locales como las globales de la imagen. Es necesario calcular la distribución de las diferencias de intensidad entre parejas de píxeles adyacentes. Los posibles pares de píxeles son identificables en la figura 2.2.6, compuesta de los valores 80-80 y 80-175: esta última pareja se refiere a un borde de la imagen más bien evidente, pero que se quisiera hacer resaltar aun más.

Este proceso ha sido implementado en la función ***void netExpansionForce(int **immagineFinale)***, dentro del archivo SeaNetLogExtractFunzioni.cpp.

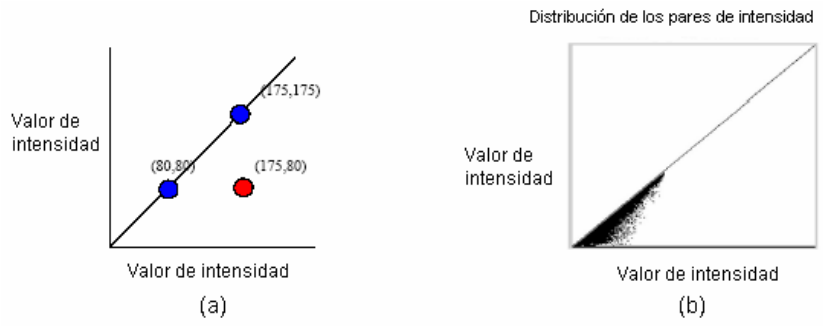



*Figura 2.2.6 (a) Parejas de valores de intensidad de píxeles adyacentes; (b) ejemplo de distribución de las parejas de intensidad de una imagen*

El algoritmo consiste en aplicar una curva de expansión (***función de mapeo de intensidad***) que asigne a cada intensidad en ingreso un valor en la salida. Por ejemplo, si la curva de expansión de intensidad fuese como en la figura 2.2.7 (a), los pares de píxeles mostrados en la figura 2.2.6 se "alejarían" entre ellos, creando así un estrechamiento del rango dinámico: figura 2.2.7 (b). Es esta simple operación la que permite modificar el contraste.

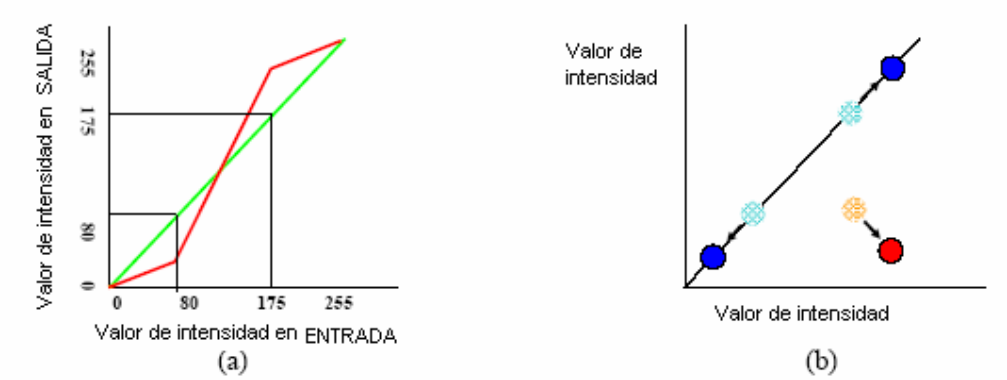



*Figura 2.2.7 (a) ejemplo de funcion de mapeo de intensidad; (b) efecto de la aplicación de la funcion de mapeo de intensidad sobre los pares de intensidad*

La generación de la curva se hace dinámicamente y permite una aproximación adaptiva: se genera una curva específica por cada imagen.

La operación de creación de esta curva es articulada:

A. se toman en consideración todas las parejas de píxeles, se genera un tren de impulsos de amplitud unitaria llamado ***fuerza de expansion*** entre los valores de las dos intensidades: siempre con referencia al ejemplo de la figura 2.2.6 (a), se genera un tren de impulsos entre los valores 80 y 175. La suma de todos los trenes de impulsos forma el así llamado ***tren de fuerzas de expansion***.

B. en realidad no todos los pares de valores crean una *fuerza de expansion*: esto sucede solo si la diferencia entre los dos valores es superior a un umbral elegido *a priori*. Si el umbral no viene superado, entonces las muestras se refieren a una zona de bajo contraste y no se tiene necesidad de valorizarlo. Por esta razón viene creado entre los pares de píxeles cuya diferencia de intensidad no supera el umbral, un tren de impulsos de amplitud **g** llamado ***fuerza de anti-expansion*** que viene substraído al *tren de fuerzas de expansion*. Esta operación permite evitar también de exaltar el eventual ruido indeseado. Los eventuales impulsos presentes en el *tren de fuerzas de expansion* que tienen amplitud negativa son llevados a 0. Entonces podemos escribir que para 0<=i<=255, el vector *net expansion force* se obtiene así:

*net expansion force[i] = expansion force[i] - **g***anti-expansion force[i]*

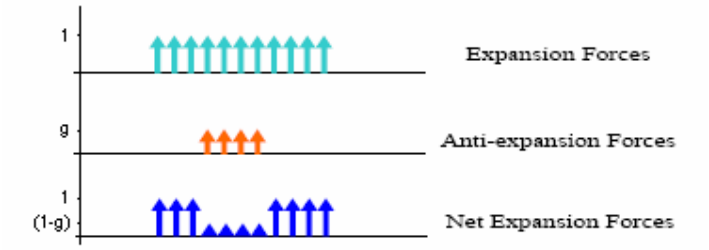



*Figura 2.2.8 Creación del vector net expansion force de la suma pesada de los vectores expansion force y anti-expansion force.*

C. se aplica una corrección a la curva (definida ***funcion de mapeo***) con el fin de reducir el rango dinámico de las amplitudes: haciendo así es menos artificial el "estrechamiento" de los valores de intensidad y, como consecuencia, el impacto visual de la imagen misma. La operación realizada es una simple elevación a la potencia donde el exponente depende de un parámetro **M**. Si

llamamos X a la curva de *tren de fuerzas de expansion* y Y es la *funcion de mapeo,* entonces:

$$Y = X^{1/M} \qquad (2.L)$$

D. se crea la ***funcion de mapeo normalizada***, normalizando la *funcion de mapeo* entre 0 y 255.

E. finalmente estamos listos para crear la *funcion de mapeo de intensidad*: se realiza una suma pesada de un parámetro **k** de la curva identidad (la curva de asignación de los niveles de gris teniendo inalterados los de ingreso) y de la integral del tren de impulsos final memorizado en el *tren de fuerzas de expansion*.

*intensity mapping function = **k***(normalized mapping function) +*

*(1-**k**)*(original mapping function)*

En la figura 2.2.9 están esquematizados los pasajes expuestos:

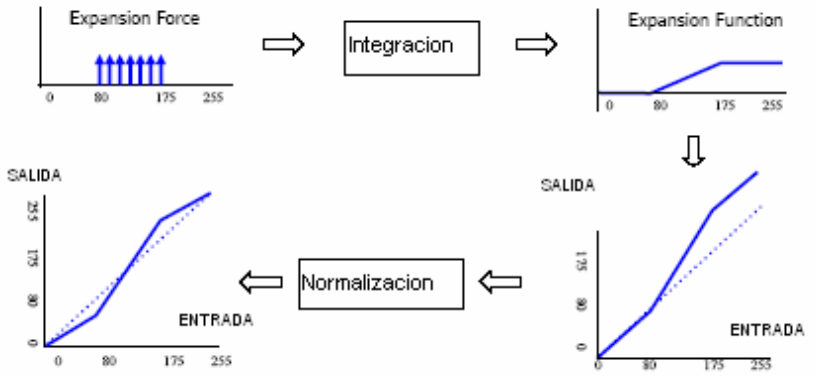



*Figura 2.2.9 Esquematización de los pasajes necesarios para la creación de la funcion de mapeo de intensidad.*

Notemos como se puede intervenir en la formación de la *funcion de mapeo de intensidad* y, por tanto, en la modificación de los niveles de intensidad, a través de la elección de los valores de cuatro parámetros: **g**, **threshold**, **M** y **k**.

Este método es muy potente. Debido a la elección de diferentes valores de estos cuatro parámetros se obtienen resultados muy diferentes. Los parámetros fueron escogidos experimentalmente. Para los datos que han sido provistos, las siguientes configuraciones de parámetros han generado en promedio buenos resultados:

- **g** que controla la amplitud de la *fuerza de anti-expansion* fue fijado en 0.6 (o en alternativa 0.2);
- **threshold**, que determina si las dos muestras representan un borde o no, a 15;
- **M** que determina el valor del exponente para crear la *funcion de mapeo* de la curva de *tren de fuerzas de expansion* (formula 2.L), a 2 o a 4;
- K, utilizada para pesar la media entre la *funcion de mapeo normalizada* y la original *funcion de mapeo*, a 0.8;

Vale la pena subrayar como, además de los óptimos resultados provistos, uno de los puntos de fuerza de este algoritmo reside también en la simplicidad y en la consiguiente velocidad de realización de las imágenes.

# CAPITULO 3

## TECNICAS DE GENERACION DE IMÁGENES Y DETECCION DE OBJETOS PARA SISTEMAS SONAR *HAZ DE LAPIZ*

En este capitulo será presentado el trabajo desarrollado sobre los sistemas sonar de tipo *haz de lapiz*. Las elaboraciones ejecutadas fueron orientadas a la localización automática de objetos simples (una tubería en una trinchera) en la escena.

Para este fin fueron desarrolladas dos técnicas de detección de objetos, ambas basadas en la estrategia de *plantilla correspondiente* (template matching). Dentro del programa principal en C++, en el archivo SeaNetLogExtractDlg.cpp se escoge cual de las dos técnicas utilizar, llamando a la función respectiva.

En este capitulo el término imagen no indicará ya una representación de la escena a niveles de gris, sino que indicará el perfil del fondo marino. Por esta razón, normalmente se definen los sistemas *haz de lapiz* como **perfiladores** y los sistemas de tipo *haz de abanico* como **imaging**.

## 3.1 DESCRIPCIÓN DE LOS SISTEMAS SONAR *HAZ DE LAPIZ*

Las imágenes producidas por los sistemas sonar de tipo *haz de lapiz* son los perfiles exteriores de las escenas barridas. Este particular tipo de sonar permite obtener, para cada impulso transmitido, la información relativa a la distancia y a la fuerza reflectante de los objetos puestos en la dirección a lo largo de la cual está apuntado el haz (figura 3.1.1). Haciendo recorrer el haz en varias direcciones se logra entonces obtener una imagen del perfil de cierto sector del fondo submarino. Los sistemas *haz de lapiz* pertenecen al grupo de los sistemas *haz unico*, que usan barrido mecánico, como se vio en el capitulo 1. En el caso del *haz de lapiz*, el haz emitido es de forma cónica, con una apertura angular pequeña.

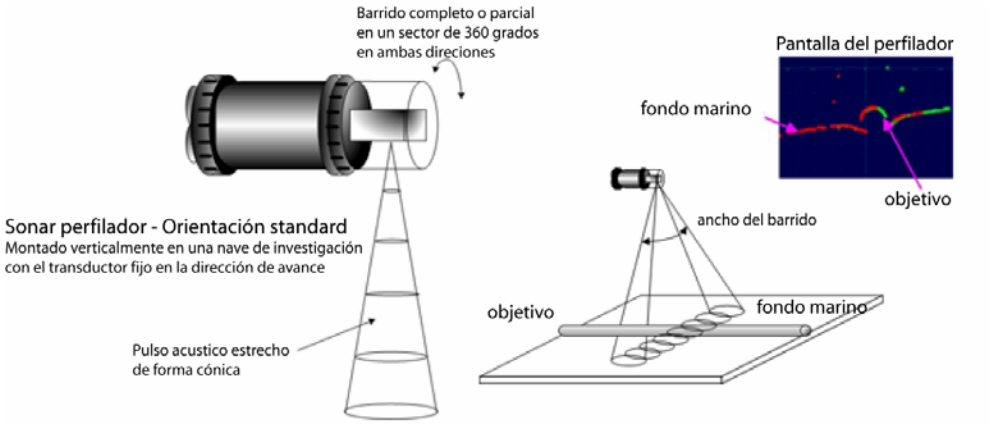



*Figura 3.1.1 Barrido del fondo marino con el sistema sonar haz de lapiz*

El sonar utilizado en esta investigación para recolectar los datos es el "Super Seaking DFP" producido, como en el caso del Super Seaking DFS (*haz de abanico*), por la TRITECH.

Este sistema tiene la posibilidad de trabajar a 2 frecuencias diferentes: 0.6 Mhz y 1.1 Mhz. Los datos con los que se desarrolla esta tesis provienen de un sistema que trabaja a ambas frecuencias.

Los datos analizados han sido obtenidos con un sistema perfilador compuesto de dos sensores sonar *haz de lapiz* que barren la escena bajo examen desde 2 puntos de vista diferentes puestos a lo largo de una línea de referencia común (*sensor doble*). Los 2 barridos se sobreponen parcialmente y los dos perfiles son producidos independientemente por los dos transductores. Se puede, por tanto, decidir si se evalúan las dos imágenes separadamente o si se utilizan los datos adquiridos por ambos para crear una única imagen representando ambos sectores investigados. Se tiene así la posibilidad de obtener una imagen de un sector del fondo más amplio del que sería posible obtener por medio del uso de un único transductor. (Figura 3.1.2).

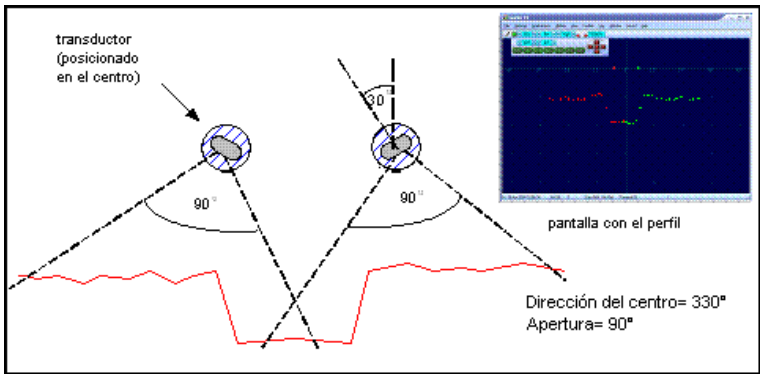



*Figura 3.1.2 Uso de dos sensores de tipo haz de lapiz para generar una imagen más extensa del fondo marino*

Cabe aclarar que para poder "fundir" los datos relativos a ambos transductores, es necesaria información adicional relativa a los ángulos de barrido y sobretodo a la posición reciproca de los dos transductores sonar. Uno de los dos transductores debe ser configurado como *maestro* y el otro como *esclavo*. Estas

informaciones se encuentran en los archivos *log* (*log file*), cuya estructura será presentada en modo detallado en los próximos párrafos.

Algunos de los parámetros que caracterizan el funcionamiento del sistema sonar *haz de lapiz* de interés se muestran en seguida:

| | |
|---|---|
| Apertura del haz: | 2º de forma cónica (600kHz) |
| | 1º de forma cónica (1.1MHz) |
| Alcance máximo: | 80 m (600kHz) |
| | 40 m (1.1MHz) |
| Alcance mínimo: | 0.3 m |
| Resolución en rango: | 1mm. |
| Anchura del impulso: | 20 – 200 μsec |
| Ancho de banda del sistema: | 30kHz |
| Resolución mecánica | 0.45º |
| Tamaño de paso mecánico | 0.45º, 0.9º, 1.35º, 1.8º |
| Sector barrido | variable hasta 360º |

Este sistema soporta operaciones no solo con uno o dos transductores sonar, sino también con 3 o 4 sensores activos contemporáneamente.

## 3.2 EXTRACCIÓN DE LOS DATOS Y GENERACIÓN DEL PERFIL ACÚSTICO

### 3.2.1 ORGANIZACIÓN DE LA INFORMACIÓN

Toda la información recogida por los sensores sonar durante el proceso de barrido viene almacenada como señales de bajo nivel en los archivos *v4log,* como

fue explicado en el capitulo 2. Por cada impulso transmitido la información memorizada es el tiempo que transcurre desde el envío de la señal hasta la consecuente recepción de la onda de retorno. La información asociada a estos ecos retro-difundidos, consiste, por tanto, en el tiempo de vuelo (tiempo empleado por la señal en regresar al transductor).

Claramente la señal de retorno no será un impulso perfecto sino una señal variable en el tiempo: se crea, por tanto, el problema de fijar un criterio para decidir cuando la onda se puede considerar que ha sido recibida. En relación a lo que se introdujo en el capitulo 1 y a los dos criterios introducidos para ejecutar esta operación, es necesario precisar que todas las señales de interés fueron obtenidas definiendo el instante de su recepción como aquel correspondiente a la llegada del primer pico de la señal que excede un cierto valor umbral y no aquel que corresponde al pico absoluto. Es fácil ver como esta información temporal puede ser usada para medir la distancia desde el objeto reflectante: veremos con más detalle esta operación en el párrafo 3.2.3.

La información necesaria para formar un perfil se obtiene utilizando toda la información recolectada por las varias emisiones que delimitan un cierto sector, como se ve en la figura 3.1.1, en la cual el sensor sonar apunta en un conjunto de direcciones adyacentes hasta completar el sector del que se desea obtener el perfil.

**Estructura de los archivos *log* (*log file*)**

La estructura de los *log file* es tal que permite memorizar en formato binario todos los datos que se obtienen de la operación de barrido. En común con los *log*

*file* presentados por los sistemas de tipo *imaging*, la estructura de estos archivos presenta solamente los 80 bytes iniciales (*log header*), cuyos campos fueron ya ilustrados en la sección 2.1.1.

Cada archivo contiene una serie de "paquetes" llamados ***registro de barrido del perfil (profiler scan record)*** y al final del archivo existe la posibilidad de añadir el campo *extra-data,* con funcionalidades reservadas para usos futuros. La estructura de los archivos *log* está esquematizada en la figura 3.2.1.

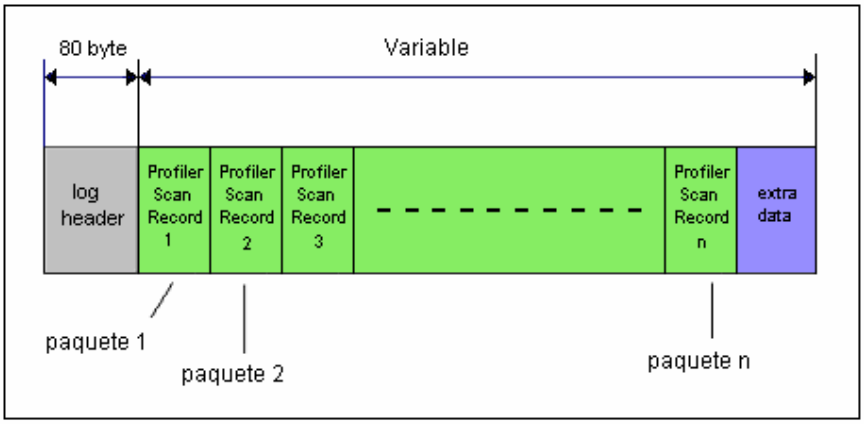



*Figura 3.2.1 Estructura de un archivo log. Después del header, se encuentran los registros especiales seguidos de los paquetes que portan la información.*

Cada paquete contiene la información relativa a un perfil entero. El numero total de paquetes contenidos en el *log file* será, por tanto, igual al numero de imágenes que es posible obtener.

Cada *registro de barrido del perfil* consta de un encabezado de 42 bytes seguido de los valores de tiempo relativos a la recepción de cada señal de retorno, por cada ángulo iluminado. Estos datos contienen la información de distancia desde

el sensor sonar hasta el perfil, para cada impulso. Cada valor temporal viene memorizado en 2 bytes.

Análogamente a lo que ocurre en los sistemas *haz de abanico*, la información que porta el encabezado general y los encabezados de cada paquete son de suma importancia para la decodificación posterior de éstos y la generación del perfil acústico.

El trabajo desarrollado en esta tesis permite extraer de los *log file* tanto los datos relativos a cada dirección de incidencia como a la información más general, necesaria para el uso de estas últimas: en particular las informaciones que se refieren a la posición relativa de los dos transductores sonar cuando el sistema de adquisición funciona en modalidad *sensor dual*.

La siguiente tabla muestra  la estructura que presenta cada paquete, el cual está compuesto de un encabezado formado por 42 bytes y un número indefinido de bytes que representan el tiempo almacenado en forma hexadecimal, 2 bytes por impulso.

| **BYTE** | **DESCRIPCIÓN** | **EJEMPLO** |
|---|---|---|
| 1,2 | Longitud binaria del mensaje | 00BAH = 186 bytes |
| 3-10 | Fecha | No. de días desde el 30/12/1899 |
| **11** | **Nodo transmisor** | **14H = Nodo 20, 15H = Nodo 21** |
| 12 | Nodo receptor | Siempre FFH |
| 13 | Mensaje de datos del sonar | Siempre 02H |
| 14 | Secuencia de paquete | Siempre 80H |
| 15 | Numero de nodo | Copia del byte 11 |
| 16,17 | Longitud binaria del mensaje -15 | 00ABH = 171 bytes |

| | | |
|---|---|---|
| **18** | **Tipo de cabeza** | **05H = Profiler** |
| **19** | **Estado** | **80H** |
| 20 | Código de barrido | 00H; barrido normal |
| **21,22** | **Head controls** | **4386** |
| **23,24** | **Escala en rango en decímetros** | **14H = 20 decímetros** |
| 25-28 | Parámetro del transmisor | 9AE147AH |
| 29 | Ganancia (/255 unidades) | 64H = 100 = 47% |
| 30,31 | Slope | 0096H = 150 |
| 32 | A-D Span del receptor | 32H = 50 |
| 33 | A-D Low del receptor | 14H = 20 |
| **34,35** | **Limite izquierdo (1/16 grad)** | **960H = 2400** |
| **36,37** | **Limite derecho (1/16 grad)** | **F90H = 3984** |
| **38** | **Intervalo de paso del motor (1/16 grad)** | **18H = 24** |
| 39,40 | Tiempo de barrido | F6AH = 3946 |
| **41,42** | **Numero de pings en el barrido** | **0043H = 67 puntos** |
| | | |
| **43,44** | **1º Ping** | **0256H = 598 (en microsec)** |
| **45,46** | **2º Ping** | **02B5H = 693** |
| **47,48** | **3º Ping** | **02ECH = 748** |
| **.** | | |
| **.** | | |
| **.** | | |
| **175,176** | **67º Ping** | **0000H = No eco** |
| | | |
| **177-186** | **Extra Record** | |
| | | |

*Tabla V. Contenido de cada paquete (profiler scan record)*

Los campos en negrillas de la tabla V son de fundamental importancia para la reconstrucción del perfil. Veamos cada uno de ellos con mayor detalle:

***Nodo transmisor**:* es el identificador de cada transductor. Son valores escritos desde la fábrica. Cuando se trabaja en modalidad dual, como en este caso, cada una de los 2 sensores tiene un valor de nodo transmisor diferente, por ejemplo 20 y 21.

***Tipo de cabeza**:* identifica el tipo de sonar utilizado, en el caso del profiler el código es 05

***Estado*:** el byte de estado indica las condiciones de error en el sensor, que se producen durante la emisión-recepción del ping.

***Head Controls**:* son 16 bits con información importante sobre la configuración del sensor sonar. Cada bit lleva una información específica, que se describe en la tabla VI.

| **BIT** | **DESCRIPCIÓN** | **EJEMPLO** |
|---|---|---|
| Bit 0 | AGC | 0=AGC Off, 1=AGC On |
| Bit 1 | Scanalt | 0=barrido en una dirección, 1=barrido alternado |
| Bit 2 | Scanright | Dirección de barrido; 0=Izquierda, 1=Derecha |
| Bit 3 | Invert | Inversión de la cabeza 0=Normal, 1=Invertida |
| Bit 4 | Motoff | Motor apagado para pruebas 0=MotorOn, 1=MotorOff |
| Bit 5 | Txoff | Transmisor apagado para pruebas 0=TxOn, 1=TxOff |
| Bit 6 | Prf_t10 | Unidad de tiempo 0=μsec**,** 1=μsec *10 |
| Bit 7 | Chan2 | Frecuencia usada 0=canal1, 1=canal2 |
| Bit 8 | First | Eco considerado 0=el de mayor valor pico, 1= el primero recibido |

| | | |
|---|---|---|
| Bit 9 | Hasmot | Estado del motor durante el barrido 0=OFF, 1= ON |
| Bit 10 | PingSync | Sincronización entre master y slave 0=Off, 1=On |
| Bit 11 | ScanSync | Sincronización de inicio 0=Master inicia, 1= Master y Slave al mismo tiempo |
| Bit 12 | StareLLim | Dirección fija definida por Llim. 1= No hay barrido, apunta a Llim |
| Bit 13 | Master | 1= Master, 0=Slave |
| Bit 14 | Mirror | Barrido invertido para el slave 1=On, 0=Off |
| Bit 15 | IgnoreSensor | Ignorar el sensor en caso de error 1=On, 0=Off |

*Tabla VI. Significado de cada bit del campo Head Controls de cada paquete*

Cada uno de estos bits debe ser continuamente probado para saber la condición en que se encuentra el transductor sonar. De éstos depende la correcta generación y posición del perfil en la imagen.

***Escala en rango en decímetros**:* el alcance expresado en decímetros.

***Límite izquierdo, Límite derecho**:* son los ángulos desde donde comienza y termina el barrido del perfil. Dependiendo de la configuración del sensor sonar, el barrido puede iniciar en Límite izquierdo y terminar en Límite derecho o viceversa, según como se definan los bits 1 y 2 del campo *head control* (Figura 3.2.2).

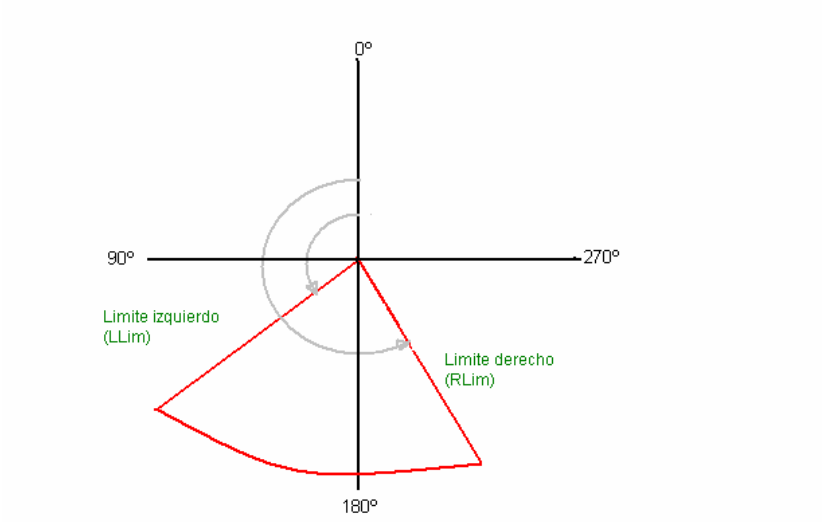



*Figura 3.2.2 Sector de barrido definido por los Límites izquierdo y derecho*

***Intervalo de paso del motor***: espaciamiento en grados entre emisiones adyacentes.

***Numero de pings en el barrido****:* indica el número de impulsos que han sido usados para cubrir todo el sector considerado. Se obtiene también dividiendo la apertura (Rlim-Llim) entre el intervalo de paso del motor.

***1° Ping, 2° Ping,...:*** información relativa al tiempo de ida y regreso de la onda acústica.

El software desarrollado ejecuta la operación de generación del perfil acústico en 3 etapas:

A. extracción de los datos a partir de los archivos *log* y generación de dos vectores que contienen los ángulos de incidencia y el tiempo de vuelo para cada impulso (funciones ***double *thetaProfiler(int nimmagine, double***

***pingA, double *pingB, BYTE AisMaster, BYTE scanRight[])*** y ***double* vettoreCampioni(int u,int *vettoreDatiA,int *vettoreDatiB)***).

B. operación de *barrido y conversion* para pasar del dominio polar al dominio cartesiano (función ***int**scan_convProfiler(double*thetaRuotata,double* vettoreCampImm, BYTE AisMaster, double* posizioneTubo)***).

C. construcción de la matriz de píxeles y generación de la subsiguiente imagen, que se implementa dentro de la función anterior.

### 3.2.2 EXTRACCIÓN DE DATOS

Es necesario aclarar que, puesto que el sistema trabaja en modalidad *sensor dual*, serán obtenidos dos perfiles contemporáneamente, uno por cada transductor sonar. Estos perfiles vendrán memorizados al interno del mismo *log file* en modo "alternado". Por esta razón, la primera operación realizada ha sido la creación de dos vectores (definidos como vectorA y vectorB) en los cuales se guardan separadamente las informaciones que se refieren a los dos transductores. Como consecuencia, es necesario asociar los datos al respectivo sistema sonar que los ha generado: esto es posible a través de la lectura de los bit 11 y 13 del campo *head control* (Tabla VI). La estructura de los dos vectores se ejemplifica a continuación (figura 3.2.3):

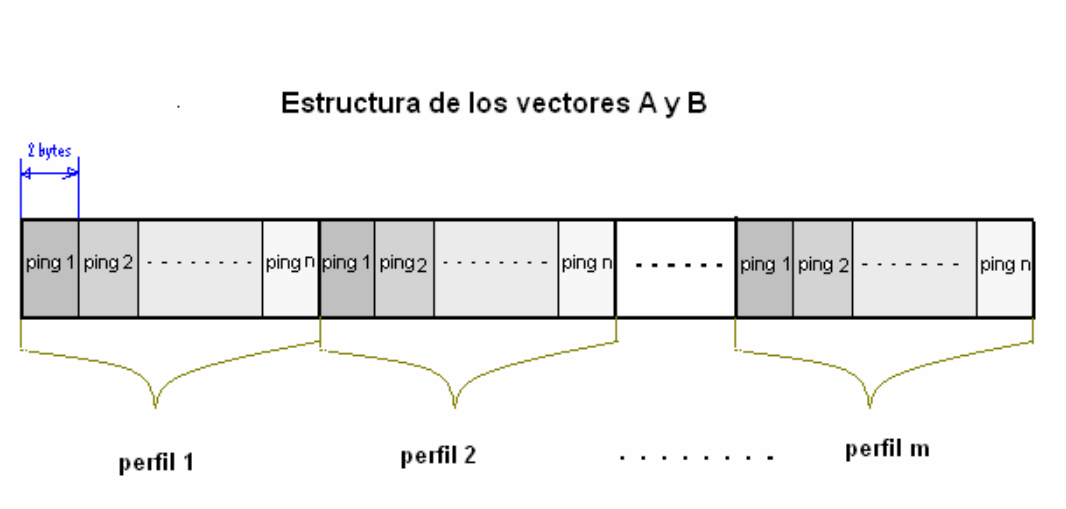



*Figura 3.2.3 Estructura de los vectores A y B, que contienen los tiempos de vuelo para cada impulso (ping).*

La lectura de estos datos se implementa dentro de la función ***void OnBtnProcess()*** del programa principal. Cada uno de estos vectores, al final de este primer paso, contendrá todos los perfiles producidos por uno solo de los sensores sonar. La información contenida estará constituida exclusivamente de los valores de tiempo expresados en microsegundos o en microsegundos*10, en dependencia del bit 6 del campo *Head Control*. Los dos vectores, lógicamente, tendrán la misma dimensión.

La información que contienen los vectores A y B no es suficiente para la creación del perfil, puesto que no se tiene aún ninguna información sobre la orientación espacial: no se puede asociar una posición en el espacio a las distancias individuadas. En los paquetes *registro de barrido del perfil* se encuentra la información sobre el ángulo de incidencia relativo a cada emision. Es preciso

generar, por tanto, dos vectores de dimensiones idénticas a la de los vectores A y B, en los cuales venga memorizada esta información angular.

Para la creación de estos vectores hay que considerar los siguientes parámetros:

- inicio del barrido (Limite izquierdo o Limite derecho)
- dirección de barrido inicial (de izquierda a derecha o de derecha a izquierda)
- tipo de barrido (en una sola dirección o alternado)

Finalmente notamos que el valor del Límite izquierdo, del Límite derecho o del paso mecánico son siempre los mismos para todos los impulsos contenidos en un único *log file*, aun cuando estos podrían teóricamente variar de perfil en perfil. Esto es debido a que cada *log file* se refiere al barrido de un perfil de interés del cual vienen hechos un sinnúmero de barridos (y por tanto, un sinnúmero de imágenes): todos los datos contenidos en un único archivo son conceptualmente referidos a un único sector.

### 3.2.3 *BARRIDO Y CONVERSION* Y CREACIÓN DE LA MATRIZ DE PÍXELES

En modo totalmente análogo a lo que se ha visto para los sistemas de tipo *imaging*, los datos memorizados por los sistemas perfiladores vienen expresados en un sistema de coordenadas polares con el origen de los ejes coincidiendo con la posición del transductor.

Si se afronta el problema de realizar una imagen del perfil submarino bajo análisis, seria útil referir estos datos a un dominio de coordenadas cartesianas. Se quiere, de hecho, crear una matriz en la cual cada elemento representa un píxel de la imagen que se quiere representar. El valor de cada elemento de esta matriz se obtiene precisamente a partir de los datos expresados en $(\rho,\theta)$.

Es necesaria una operación de *barrido y conversion*, la cual fue conceptualmente introducida en el capitulo 2. Esta operación, sin embargo, resulta ser mucho más simplificada que aquella vista en el caso del sonar de tipo *haz de abanico*. Justamente porque se desea representar solamente un perfil del fondo submarino, no es necesario asignar un valor de intensidad a cada píxel, sino que a partir de una matriz de ceros es suficiente "encender" solo los píxeles que se considera que representan el fondo. Todo el algoritmo se implementa en la función ***int**scan_convProfiler(double* thetaRuotata,double* vettoreCampImm, BYTE AisMaster, double* posizioneTubo)***.

Necesitamos primero precisar como es posible transformar la información de tiempo contenida en los paquetes (tiempos de vuelo) a la información de distancia desde los transductores. También en este caso se tiene, análogamente a lo que se ha visto para el caso de los sistemas *haz de abanico*:

$$\rho = \frac{v \cdot T}{2} \tag{3.A}$$

donde *T* es el tiempo de ida y regreso de la onda emitida por el transductor, *v* es la velocidad del sonido en el mar (la velocidad del sonido en el agua salada ha

sido estimada en 1477.5 m/s) y $\rho$ es la distancia calculada en metros (figura 3.2.4).

El paso sucesivo es definir un punto de referencia absoluto para ambos sensores sonar, puesto que los datos guardados en los vectores de ángulos y distancias se refieren a ángulos y distancias con respecto a *cada uno* de los sensores, así como están los datos no pueden ser utilizados conjuntamente.

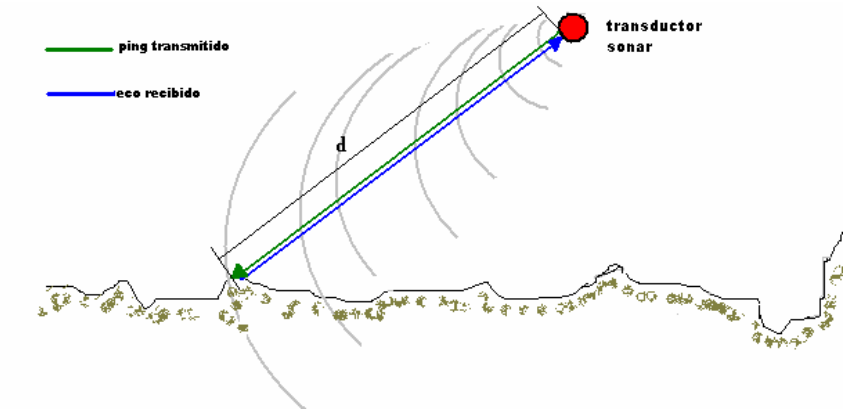



*Figura 3.2.4 Cálculo de la distancia recorrida por el impulso acústico. El impulso transmitido encuentra un obstáculo en su camino y regresa en forma de eco. El tiempo T obtenido es el doble del tiempo utilizado para llegar al obstáculo.*

El objetivo es entonces el de crear un punto origen de coordenadas (0,0) en un dominio de coordenadas cartesianas (X, Y), a partir del cual todos los puntos del perfil puedan ser referidos.

Este origen de coordenadas existe intrínsecamente y se encuentra definido en la información contenida en el *log header*: en él se encuentra presente la información sobre las posiciones de los sensores Master y Slave, con respecto al vehiculo que los transporta. Ha sido creada una función para obtener estos datos: ***void posizioneTeste(FILE *f)***.

El paso sucesivo es definir cual de los dos vectores (A o B) corresponden al transductor definido como Maestro o como Esclavo. En este punto, definiendo las coordenadas cartesianas del transductor sonar identificado como Maestro (xMaster, yMaster) y las de aquel identificado como Esclavo (xSlave, ySlave), podemos escribir:

$$X_i = \rho_i * sin(\theta_i) + xMaster\ (xSlave); \qquad (3.B)$$

$$Y_i = \rho_i * cos(\theta_i) + yMaster\ (ySlave);$$

donde $(X_i, Y_i)$ son las coordenadas del i-esimo punto del perfil, identificado por un rango $\rho$ y un ángulo $\theta$ (en coordenadas polares). Todas las cantidades se expresan en metros (Figura 3.2.5).

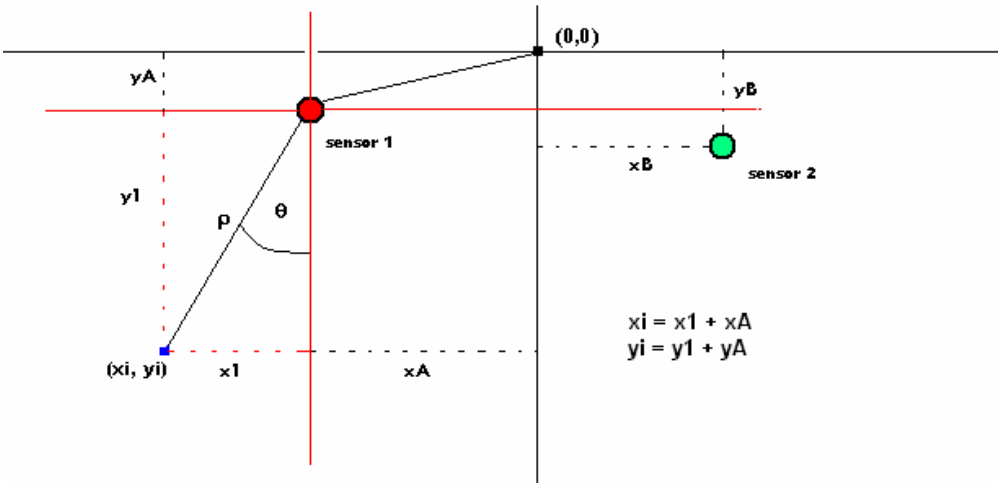



*Figura 3.2.5 Posicionamiento de un punto del perfil con respecto a un origen de coordenadas común a ambos sensores*

Hemos así obtenido un conjunto de puntos cuyos valores en coordenadas cartesianas representan distancias en metros en la escena. El último paso es representar este conjunto de puntos como la imagen del perfil. Se crea entonces una matriz de píxeles, en donde cada píxel representa un rectángulo de unas ciertas dimensiones reales, vistas a escala.

La dimensión de la sección de la escena bajo examen que cada píxel debe representar es dependiente de la resolución del sistema sonar. De hecho, la resolución de la imagen depende de la resolución del sonar. En realidad es suficiente con conocer la resolución en rango porque ésta es normalmente peor que la resolución angular y, por tanto, es este valor el que debe imponer la dimensión real de un píxel. El cálculo de la resolución en rango se realiza fácilmente de acuerdo con la siguiente formula:

$$R = \frac{v}{2 \cdot B} \tag{3.C}$$

El valor de la banda (*B*) se puede derivar fácilmente de los parámetros contenidos en el *log header*.

Un cálculo necesario es la dimensión total de la matriz. Esta depende de la apertura de los sectores investigados por los dos transductores sonar (Figura 3.2.6). La imagen debe ser construida en modo tal de poder representar la unión de los dos perfiles.

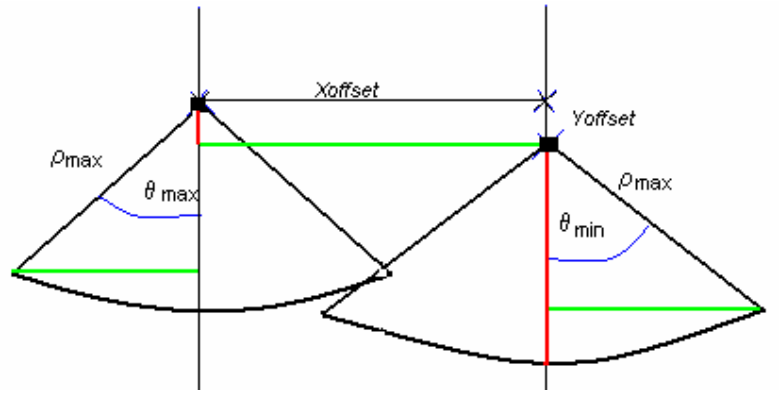



*Figura 3.2.6 Cálculo de las dimensiones de la matriz de píxeles. Es necesario tomar en cuenta la diferencia de posiciones entre los sensores sonar (Yoffset, Xoffset) para calcular el tamaño máximo.*

El calculo de las dimensiones de la matriz se realiza fácilmente por medio del calculo de la diferencia de las coordenadas (cartesianas) asociadas a cada transductor sonar (Xoffset y Yoffset) y gracias al conocimiento del ángulo máximo y mínimo de apertura del sector investigado por cada transductor (Figura 3.2.7).

Una vez calculadas la dimensión de la matriz y la dimensión de cada píxel, podemos representar la imagen como una matriz. Cada píxel está representado por 8 bits: se puede asociar a cada uno de estos un valor de amplitud comprendido entre 0 y 255. Inicialmente se inicializan todos los píxeles de la matriz con cero. La imagen se crea luego asignando el valor máximo (255) a todos los cuyas coordenadas correspondan a aquellos puntos del perfil que fueron calculados previamente.

Se ha creado así una imagen binaria cuyos valores representan la ausencia o la presencia del perfil.

La creación del perfil a partir de los valores en coordenadas cartesianas se resume en los siguientes pasos:

- inicialización de los píxeles de la matriz con el valor de 0
- individuación de la posición de los píxeles al interno de la matriz que representan el perfil del sector del fondo investigado
- asignación del valor 255 a cada uno de los píxeles identificados

En la práctica los dos perfiles creados se distinguen en la imagen con los colores rojo y verde, cada uno correspondiente a un sensor sonar. Esto se logra mediante el cambio de algunos parámetros en el mapa de bits que representa la imagen. Se crea así una única imagen en la cual es aun posible individuar la proveniencia de la información y asociar cada punto al respectivo sensor sonar (figura 3.2.7)

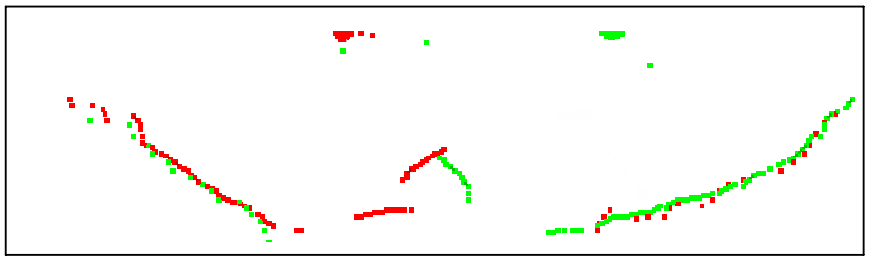


*Figura 3.2.7 Imagen generada por el software producido durante la tesis. Se pueden identificar bien, gracias al uso de dos colores diferentes, los dos transductores sonar y sus respectivos perfiles.*

## 3.3 MÉTODOS DE DETECCIÓN DE OBJETOS

En este trabajo han sido desarrolladas dos técnicas dirigidas a detectar de manera automática la presencia de un objeto simple y localizarlo individuando una de sus secciones, conocida *a priori*.

Antes de entrar en el detalle de estas técnicas, es importante subrayar que los procesos realizados fueron posibles gracias al uso del software desarrollado y descrito hasta aquí. La producción de un software propietario ha permitido no solo elaborar y optimizar los datos adquiridos por el sistema sonar, sino que también ha provisto la oportunidad de desarrollar nuevas funcionalidades como la localización automática de una tubería, la que será descrita en lo que sigue de este capitulo.

Específicamente, el objetivo del trabajo realizado fue la localización automática de una tubería (presentada en la figura 3.3.1) presente en las escenas investigadas y de las cuales fueron producidas imágenes a través del empleo de sonar de tipo *haz de lapiz* en configuración *sensor dual*.

*Figura 3.3.1 Tubería presente en la escena bajo examen.*

Una maquina excavadora está trabajando en el fondo marino y está creando una fosa necesaria para albergar un tubo sobre el fondo mismo. A causa del movimiento del agua creado por la maquina, viene elevada mucha arena, lo que hace inútil el uso de eventuales telecámaras; se hace necesario el uso de sistemas sonar. Aunque los contextos aplicativos son múltiples, la situación apenas descrita es una de las situaciones reales en las cuales la SONSUB ha recogido datos sonar y por la cual se ha sentido la exigencia de realizar un método de detección de objetos.

Aun cuando las elaboraciones desarrolladas por las dos funciones implementadas se diferencian muchísimo, ambos aprovechan una estrategia de *plantilla correspondiente*. También el método propuesto en la sección 3.3.2, de hecho, utiliza esta aproximación. La gran diferencia que hace diferentes los algoritmos y los consecuentes resultados, consiste en la selección de los puntos sobre los cuales aplicar la plantilla. Es por tanto esta operación previa la que juega un rol fundamental al final de los resultados.

Un último punto en común es el primer paso seguido por ambos métodos, o sea el primer proceso realizado sobre los datos adquiridos. Este consiste en la eliminación de algunos de aquellos valores que en la imagen se transformarán en puntos de ruido. Los datos relativos a la presencia de "objetos" muy cercanos al transductor sonar, deben ser eliminados puesto que, en el contexto aplicativo de interés, estos se pueden considerar referidos a ruido causado por la presencia de tierra levantada desde el fondo marino.

Para resolver el problema de la presencia de estos puntos no deseados, se define un valor de umbral (*sogliaRumore)* relativo al mínimo rango que un punto

detectado debe tener para no ser considerado ruido. Este valor varía de barrido en barrido y depende del valor de rango máximo (*maxPortata*), que depende, a su vez, de las configuraciones del sonar escogidas por el usuario (por ejemplo, la frecuencia de trabajo del sonar) y que puede variar de perfil en perfil:

$$sogliaRumore = \frac{\max Portata}{M} \qquad (3.3A)$$

donde *maxPortata* es el valor de rango máximo expresado en metros y *M* es una variable definida por el usuario, cuyo valor puede ser calculado en base a consideraciones geométricas. El valor de *M*, sin embargo, fue fijado después de varias pruebas igual a 12, de tal manera que el valor de *sogliaRumore* será igual a (*maxPortata*/12). Este valor ha permitido obtener buenos resultados.

Puesto que la información adquirida por los transductores es precisamente el valor de distancia ($\rho$) desde el punto detectado por el sensor sonar, es suficiente controlar que este valor de rango sea mayor que *sogliaRumore*. En la implementación de ambos métodos, esta operación viene realizada en un dominio de coordenadas polares. Si un determinado valor de $\rho$ no satisface la condición, entonces éste viene considerado automáticamente ruido y es descartado. La situación apenas descrita está ejemplificada en la Figura 3.3.2.

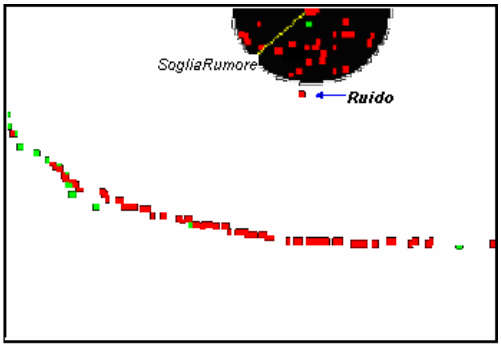



*Figura 3.3.2 Operación de reducción del ruido. Si el valor de $\rho$ considerado es menor que sogliaRumore, este viene considerado ruido y viene consecuentemente eliminado.*

### 3.3.1 DETECCIONES CONGRUENTES

Antes de proceder a la presentación detallada del método reportamos un ejemplo de lo que se desea obtener. Como se puede observar en la figura 3.3.3 (imagen típica producida por un sonar *haz de lapiz* en configuración *sensor dual*) las imágenes resultantes pueden ser pensadas como una sección del fondo marino que representan un perfil de éste.

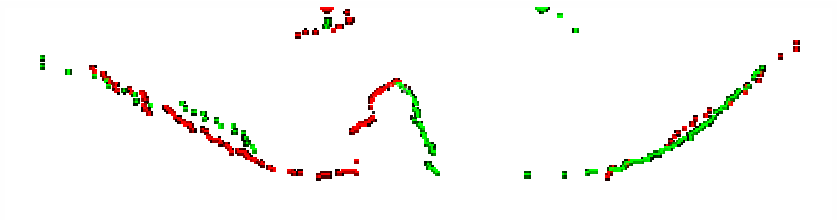


*Figura 3.3.3 Imagen típica de un perfil producida por el sonar*

Observemos ahora como se presenta la imagen al término de la operación de detección de objetos (Figura 3.3.4). Como se puede observar, se ha añadido en

modo automático el diseño de la sección de la tubería buscada (y, por tanto, localizada).

*Figura 3.3.4 Modificaciones aportadas a la imagen presentada en la figura 3.3.3, sucesivas a la aplicación del método de detección de objetos*

La idea a la base de esta técnica es la de buscar dentro de la imagen una forma conocida *a priori*. En primer lugar, por tanto, es necesario crear, en algún modo, una representación del objeto buscado, luego de lo cual se usa esta última para encontrar una correspondencia al interior de la imagen.

La técnica propuesta trabaja enteramente en el dominio "transformado" de las coordenadas cartesianas. La información de interés se encuentra, por tanto, en las coordenadas (X, Y) de los puntos que constituyen el perfil.

Otra característica importante de este método es que no mantiene separadas las informaciones que provienen de los dos diferentes transductores sonar, sino que trata estos datos en modo agregado, sin hacer ninguna distinción de su proveniencia. Vienen así creados dos vectores (**X** e **Y**) de igual dimensión, conteniendo las coordenadas de todos los puntos del perfil. No nos detendremos en esta operación de transformación de un dominio a otro (*barrido y conversion*) porque ésta ya fue expuesta en pasajes anteriores.

El vector X así formado, contendrá, por tanto, las coordenadas en *x* de todos los ecos adquiridos (y, equivalentemente, el vector Y contendrá las coordenadas en *y* relativas). Notemos como estos valores no son equiespaciados sino que, al contrario, y observando las imágenes producidas (por ejemplo, la figura 3.3.3), existen frecuentemente "saltos" entre valores adyacentes, lo que repercute visiblemente mediante la presencia de grandes zonas de la imagen en las que no está presente ningún punto.

La función que implementa la técnica propuesta no tiene necesidad de conocer nada mas que el diámetro del tubo del cual se esta buscando la sección. Este parámetro (*diametroTubo*) debe ser decidido *a priori* y debe ser asignado antes de iniciar el procedimiento de localización. Otra característica de este algoritmo es que se aplica a cada punto del perfil.

**Algoritmo propuesto**

El desarrollo de cada uno de los metodos de deteccion de objetos se ha realizado independiente y contemporáneamente, implementándose ambos en dos funciones llamadas ObjectDetector: ***double *objectDetector (double* thetaRuotata, double* vettoreCampImm, BYTE AisMaster)***, las cuales reciben como parámetros de ingreso los vectores de angulo (thetaRuotata), y de tiempos de vuelo (vettoreCampImm), así como un bit que define cual de los dos transductores es el *master*.

Hemos llamado al punto de partida del algoritmo, *start* (ver la figura 3.3.5). La primera operación realizada es la individuación, al interior del vector X, de aquellos puntos que no se distancian más de *diametroTubo* desde *start*. Todas las elaboraciones siguientes vienen realizadas a partir de un punto inicial e involucran solamente este particular conjunto de puntos. La idea a la base de

este método es la búsqueda, a lo largo del perfil, de un objeto que sea correspondiente a la forma que se está buscando. Esta operación se realiza a partir de *start*, que viene considerado como el punto extremo de una semicircunferencia de diámetro *diametroTubo.* Se calcula luego la distancia (en el eje de las Y) que hay entre la posición de los puntos del conjunto considerado y la posición ideal de los puntos que constituirían una semicircunferencia perfecta. El punto *start* que, en el conjunto, habrá generado el error medio (suma de las distancias calculadas sobre todos los puntos a considerar) menor, será por tanto considerado el punto inicial de la tubería. Si la distancia existente entre los puntos del perfil y esta semicircunferencia es demasiado grande (supera un determinado valor), entonces significa que el objeto no está presente.

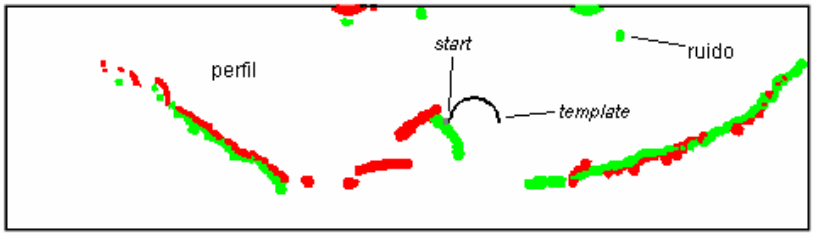



*Figura 3.3.5 Posición del elemento start con respecto al template.*

Idealmente se busca aquella parte del perfil que "asemeja" mas a la forma de la tubería buscada, esto es, buscamos la región que minimiza el *mis-matching* (error de correspondencia) y por tanto el error cometido en el aproximar el objeto buscado con aquella particular región del perfil.

La forma buscada es una semicircunferencia, puesto que la imagen representa el perfil del fondo marino y la tubería se encuentra posada sobre el fondo. El hecho de escoger una semicircunferencia y no una circunferencia deriva del

procedimiento particular de selección de los ecos operada por el sistema perfilador, donde solo el primer eco, el de la superficie superior del tubo, aun en el caso en el que la tubería "emergiese" de la fosa viene memorizado.

Procedemos ahora a la descripción por puntos de esta técnica de detección de objetos. Esta se articula en los siguientes cinco pasos:

1. reducción del ruido y "fusión" de los datos provenientes de los dos sensores sonar.

2. reorganización de las informaciones de los perfiles de modo tal que las coordenadas cartesianas x de los puntos que lo constituyen sean ordenadas en modo creciente.

3. limitación de los puntos sobre los cuales realizar la elaboración.

4. operación de *plantilla correspondiente*.

5. individuación del error mínimo y cálculo de las coordenadas del tubo.

**1. Reducción del ruido y "fusión" de los datos provenientes de los dos transductores sonar**

La primera operación necesaria es la creación de dos vectores **X** y **Y** de dimensiones iguales al numero total de emisiones (suma de los elementos de los vectores ping A y B introducidos en la sección 3.2.2). De estos vectores vienen eliminados todos los valores relativos a las distancias (desde el transductor sonar) inferiores a los valores de umbral prefijados (*sogliaRumore*) que se

presume que sean debidos a la presencia de ruido antes que la de un objeto verdadero. Fue explicada en la sección anterior el motivo de este procedimiento.

**2. Reorganización de la información del perfil en modo tal que las coordenadas cartesianas x de los puntos que lo constituyen sean ordenados en modo creciente.**

El paso sucesivo está constituido por la reorganización del vector **X** según un ordenamiento creciente. Esta operación fue hecha a través de un ordenamiento de tipo *ordenamiento Burbuja (BubbleSort)*. Este algoritmo, aun cuando no es siempre el más eficiente, ofrece buenas prestaciones sobre todo si es utilizado en vectores cuyas dimensiones no son particularmente grandes. En el caso en el que existieran eventuales valores de x coincidentes, uno de los dos puntos debe ser eliminado.

A causa de la evidente correlación entre la información contenida en los dos vectores X y aquella contenida en Y, es necesario seguir una idéntica actualización también de la organización del vector Y.

**3. Limitación de los puntos sobre los que se debe seguir el proceso**

Como será claro al final de la exposición de todos los pasos que constituyen el método, aun si el algoritmo toma en cuenta todos los puntos pertenecientes al perfil, no todos serán seleccionados como puntos de *start*. En particular, los últimos puntos no serán seleccionados, precisamente porque este particular punto representa el hipotético extremo de una semicircunferencia. (figuras 3.3.5

y 3.3.6). El último elemento sobre el cual actúa el algoritmo se ha definido como *ultimoElemento.*

## 4. Operación de *plantilla correspondiente*

En este punto se tienen todos los elementos necesarios para la aplicación en si del metodo de *plantilla correspondiente*. Podemos subdividir lógicamente esta operación en cuatro pasos ulteriores. Estos son realizados para cada elemento del vector X, genéricamente definido como $x_i$: se tomarán en consideración, por tanto, todos los elementos comprendidos entre el primero y aquel identificado como *ultimoElemento* (calculado en el paso 3). Para cada iteración se considerará por tanto un elemento de partida diferente (*start*):

4.1 En primer lugar es necesario trasladar el *template* de la semicircunferencia en correspondencia con los puntos de interés. Hemos ejemplificado en la figura 3.3.5 la posición que el elemento *start* ($x_i$, $y_i$) tiene con respecto a la del modelo. Es necesario identificar las coordenadas de la semicircunferencia ideal: es suficiente calcular las coordenadas del centro de ésta ($X_{centro}$, $Y_{centro}$), una vez conocido el radio (*diametroTubo*/2).

Dadas las coordenadas del punto que se está considerando ($x_i$, $y_i$), las coordenadas del centro pueden ser calculadas como sigue:

$$\begin{aligned} Xcentro &= x_i + \frac{diametroTubo}{2} \\ Ycentro &= y_i \end{aligned} \qquad (3.3B)$$

donde lógicamente $y_i$ es el elemento del vector Y correspondiente a $x_i$.

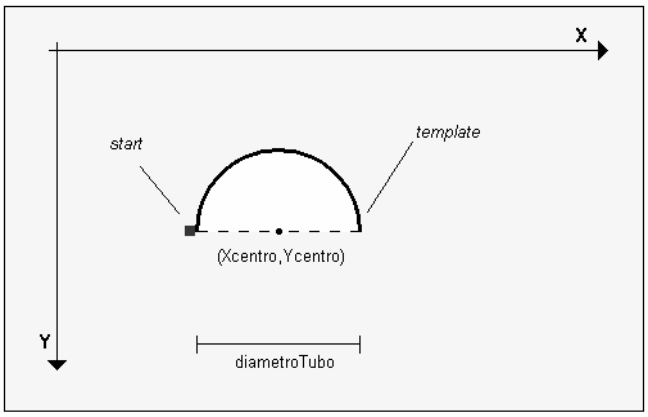



*Figura 3.3.6 Coordenadas del centro de la plantilla (template).*

4.2 Una vez calculadas las coordenadas del centro de la semicircunferencia, el paso sucesivo consiste en identificar los elementos del vector X que recaen en el mismo intervalo "cubierto" por la semicircunferencia.

Se consideran, por tanto, todos los elementos ($x_m$) sucesivos a $x_i$ tales que

$$(x_m - x_i) < diametroTubo \qquad (3.3C)$$

y los correspondientes elementos ($y_m$) del vector Y tales que

$$|y_m - Y_{centro}| < L \qquad (3.3D)$$

La ecuación (3.3D) afirma que la coordenada y de los puntos considerados debe distar mas de 0.3 metros desde el centro de la palntilla. El fin de esta operación es el de no permitir a los puntos de

ruido, que no forman parte del perfil bajo examen, ser tomados en consideración en la operación de *correspondencia*. Puesto que, de hecho, se busca proveer una medida de similitud entre la plantilla y una parte del perfil, también considerar solo un punto "fuera de lugar" produce resultados falsos.

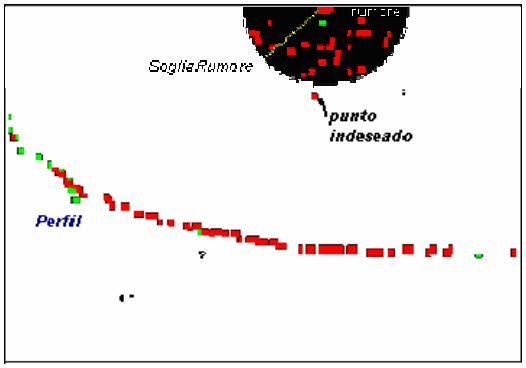



*Figura 3.3.7 Punto de ruido indeseado sobrante al final de la operación de reducción de ruido (paso 1).Por la presencia de estos puntos se hace necesario aplicar (3.3D).*

El valor de la variable L fue puesto igual a 0.3 metros. Es evidente que su valor debía ser elegido mayor del radio de la semicircunferencia (en nuestro caso 0.275 m), para permitir a todos los eventuales puntos que representan el perfil ser considerados. La elección de un valor mayor de la dimensión exacta del radio fue hecha para permitir una cierta tolerancia. Esta es necesaria puesto que la precisión de la adquisición de los datos no es perfecta. Pensemos solamente en el movimiento del vehiculo sobre el cual se encuentran puestos los sonar, que no puede estar perfectamente detenido en el agua: los puntos, de hecho, no estarán jamás dispuestos perfectamente a lo largo de un arco de circunferencia.

El valor asignado al parámetro L es de considerarse experimental, y variable según el diámetro de la tubería.

4.3 Realizamos ahora la operación de *correspondencia* con la semicircunferencia centrada en ($X_{centro}$, $Y_{centro}$). Esta operación ofrece una estima de la similitud entre las partes del perfil considerado (el conjunto de los puntos ($x_m$, $y_m$) que respetan la condición expresada por la ecuación 3.3C). La idea de base es la de calcular, para cada $x_m$, el valor de *y* que éste debería tener si representara un semicircunferencia ideal (llamado $y_{justo}$). Hecho esto se puede calcular la diferencia (dife) entre la coordenada real $y_m$ y aquella ideal $y_{justo}$.

Para cada $y_m$ viene calculada, por tanto, la siguiente cantidad:

$$\text{dife} = |y_m - y_{justo}| \qquad (3.3E)$$

$$y_{giusto} = \sqrt{R^2 - (x_m - \text{Xcentro})^2} + \text{Ycentro} \qquad (3.3F)$$

donde *R* equivale a (*diametroTubo*/2).

La medida de *error de correspondencia*, en conclusión, viene dada por la variable *error* calculada en el siguiente modo:

$$error = \frac{1}{M}\sum_{m=1}^{M} \text{dife}_m \qquad (3.3G)$$

donde *M* es el numero de elementos ($x_m$, $y_m$) considerados en el punto 4.2

4.4 Estas operaciones vienen desarrolladas para cada punto *start*. Cada elaboración producirá un valor de *error*: estos deberán luego ser comparados entre ellos para determinar cual punto *start* ha generado el error más pequeño.

La primera operación necesaria es la memorización de la variable *error* en un vector (*vettoreErrore*) que tenga la correspondencia con el relativo punto *start* que lo ha generado.

Sin embargo no siempre esta operación viene realizada. Fueron puestas algunas condiciones que deben ser respetadas para permitir un funcionamiento mas preciso del procedimiento de detección de objetos. En particular fueron definidas 5 condiciones que son indispensables para que el método produzca buenos resultados:

$$M \geq 14; \tag{3.3H}$$

$$numPuntosDerecha > 7; \tag{3.3I}$$

$$numPuntosIzquierda > 7; \tag{3.3L}$$

$$decrece > 3; \tag{3.3M}$$

$$crece > 3; \tag{3.3N}$$

La condición (3.3H) impone que los puntos que representan el perfil sean al menos 14. Se impone por tanto un numero mínimo de puntos sobre los cuales calcular el parámetro *error*: sin esta condición, aun solo un punto

(muy cercano a la forma ideal) podría ser "intercambiado", al final del algoritmo, como el perfil de la tubería.

Las condiciones (3.3I) y (3.3L) definen dos nuevos parámetros *numPuntosDerecha* y *numPuntosIzquierda*. Estos representan, respectivamente, el conjunto de puntos considerados ($x_m$, $y_m$) a la derecha y a la izquierda del centro ($X_{centro}$, $Y_{centro}$) de la plantilla: *numPuntosDerecha* son aquellos puntos tales que ($x_m$>Xcentro) y *numPuntosIzquierda* aquellos tales que ($x_m \leq$Xcentro). Estas condiciones imponen que la parte del perfil considerado tenga un número de puntos mínimos y que sea además uniformemente distribuido desde el centro de la tubería. Los valores de tales parámetros, también en este caso, fueron elegidos experimentalmente.

Las ultimas dos condiciones representan una ultima limitación sobre la distribución de los puntos tomados en examen. Se ha impuesto, a través del uso de dos ulteriores parámetros (*crece* e *decrece*), una condición sobre la posición relativa que estos puntos deberían tener en el caso de que representaran verdaderamente el perfil de una tubería. Los puntos a la derecha del centro de la tubería (*numPuntosDerecha*), de hecho, deben tener un comportamiento decreciente, mientras que los puntos a la izquierda (*numPunosIzquierda*) deben tener un comportamiento creciente.

Todas estas condiciones permiten poner limitaciones sobre los puntos que vienen usados para crear el parámetro *error*. En el caso de que una sola de estas condiciones no sea satisfecha, un error constante igual a 1000 (que es un valor enorme) viene memorizado en *vettoreErrore*. Esto

permitirá descartar esta distribución de puntos como posible tubería, en cuanto el error asociado a esta será demasiado grande.

Se repiten los pasos desde el 4.1 al 4.4 para cada elemento del vector X (xi) considerable como start (o sea los elementos comprendidos entre el primero y ultimoElemento).

## 5. Identificación del error mínimo y cálculo de las coordenadas del tubo

El ultimo paso consiste en la búsqueda del elemento *start* que ha generado el valor del parámetro *error* menor. El punto en cuestión tendrá coordenadas ($X_{start}$, $Y_{start}$). Viene buscado en el vector *vettoreErrore* la cantidad mínima. Es precisamente la posición de tal elemento que identificará las coordenadas ($\bar{x}$, $\bar{y}$) de un punto definido *top*, utilizadas para diseñar la tubería en la imagen.

$$\bar{x} = Xstart + \frac{diametroTubo}{2}$$
$$\bar{y} = Ystart - \frac{diametroTubo}{2} \qquad (3.3O)$$

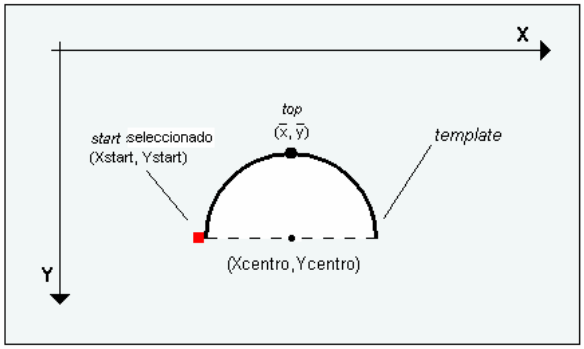



*Figura 3.3.8 Punto start relativo al parámetro error menor. Posición del punto top necesario para el diseño de la tubería*

Antes de concluir la presentación de esta técnica vale la pena subrayar el hecho de que ésta es el resultado de la experimentación de diferentes aproximaciones. Los primeros puntos del algoritmo (desde el punto 1 al 3) se mantuvieron siempre inalterados, pero los puntos que definen como ocurre el verdadero procedimiento de *plantilla correspondiente* son muy distintos a como fueron ideados inicialmente.

### 3.3.2 TÉCNICA BASADA EN LA DISCONTINUIDAD EN RANGO

El segundo método desarrollado trabaja directamente con los datos originales expresados en coordenadas polares. El método en cuestión se basa en la identificación de discontinuidades en rango entre emisiones adyacentes, usando separadamente la información retornada por cada transductor sonar. Como fue mencionado anteriormente, este algoritmo se implementa en la función ***double *objectDetector (double* thetaRuotata, double* vettoreCampImm, BYTE AisMaster)*** del archivo SeaNetLogExtractFunzioni.cpp.

El principio usado es que la presencia de un objeto en la escena bajo examen provoca notables variaciones de los valores de $\rho$ de las muestras relativas a ángulos de barrido adyacentes: se puede aprovechar esta información para detectar en la imagen de perfil la presencia de objetos. Este concepto se esquematiza en la figura 3.3.10.

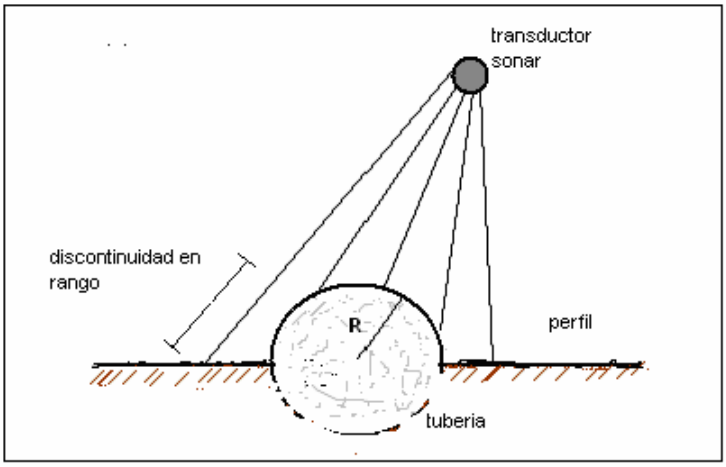



*Figura 3.3.10 Discontinuidad en rango debida a presencia de un objeto en la escena bajo investigación.*

La presencia de dos sensores sonar y, consecuentemente, de dos conjuntos de datos, permite además de encontrar (si son analizados separadamente los dos conjuntos de datos) dos discontinuidades, una en relación a cada transductor, debidas a la presencia del mismo objeto. La presencia de un cuerpo extraño en el perfil submarino, por tanto, causará dos "saltos" de los valores de rango, uno por cada transductor.

El programa desarrollado permite evaluar la presencia de estos puntos de discontinuidad, llamados "candidatos": estos podrían señalar la presencia de un objeto. Cada uno de estos puntos, como se verá mejor a continuación, es candidato para ser el punto en la cima del perfil de la tubería, al que llamaremos *top* (figura 3.3.11).

Los candidatos, referidos a ambos sensores, vienen comparados entre si: si se identifica una pareja de puntos candidato que satisface determinadas condiciones

de proximidad en el dominio cartesiano, entonces se adopta como una posible localización del *top* de la tubería.

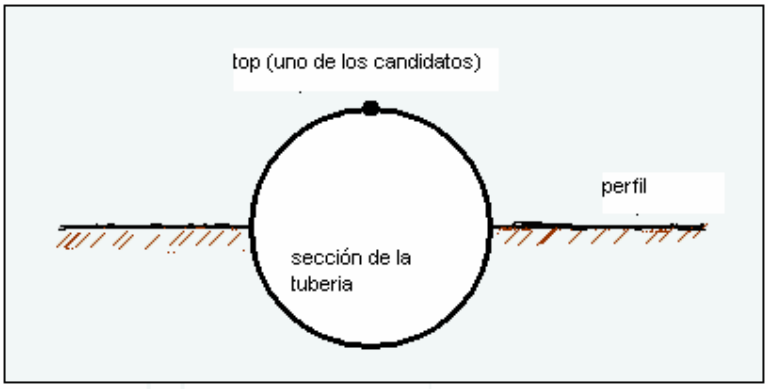



*Figura 3.3.11 El punto top, que define la posición de la tubería, es uno de los candidatos.*

Vale la pena subrayar que los puntos candidatos no necesariamente identificarán la presencia de un objeto: su trabajo consiste en proveer una indicación sobre la posible presencia de un objeto en aquel punto. Uno solo de estos puntos (si el objeto está presente) vendrá luego definido como *top*.

Todos los pasos del algoritmo propuesto, sucesivos a la identificación de estos candidatos, están dirigidos a verificar esta hipótesis. Para lograr discriminar entre todas las posibles posiciones de la tubería indicadas por los puntos candidato también se adopta un procedimiento de *plantilla correspondiente*: en este caso el *modelo* debe ser colocado en cada punto candidato. El paso sucesivo será una medida de similitud entre el modelo y el perfil considerado y la sucesiva elección de un punto candidato como *top*.

El proceso completo puede ser dividido en cinco pasos:

**1.** Reducción del ruido y eliminación de los valores nulos

**2.** Búsqueda de los "pares de discontinuidad"

**3.** Confronto cartesiano entre los candidatos identificados para ambos sensores sonar.

**4.** Correspondencia usando como *plantilla* una semicircunferencia ideal.

**5.** Discriminación del error y cálculo de las coordenadas del tubo.

Como en el caso precedente, el objeto específico que se busca sobre el fondo marino es una tubería de diámetro *diametroTubo*. Veamos al detalle cada uno de los pasos del algoritmo.

### 1. Reducción del ruido y eliminación de los valores nulos

El primer paso consiste en la reducción del ruido. Todas las muestras cuyo $\rho$ no supera el valor de *sogliaRumore*, vienen descartadas, como fue explicado al inicio del capitulo.

El paso sucesivo consiste en la eliminación de los valores memorizados por el sistema sonar que no llevan información acerca del perfil: durante el proceso de adquisición de las señales (elaboración de los ecos), si el transductor no ha recibido una respuesta al envío de un ping dentro de un intervalo de tiempo preestablecido, entonces se asocia a la falta de respuesta un valor nulo. Un valor nulo así obtenido es una falsa medición y no debe ser tomado en cuenta para el resto del proceso. Vienen así eliminados los valores nulos.

## 2. Búsqueda de los "pares de discontinuidad"

La figura 3.3.12 ilustra el principio usado para lograr reconocer un objeto posado en el fondo marino. Se pueden observar los puntos relativos a direcciones de barrido adyacentes A, B y C, D y notamos como la diferencia en rango entre los puntos C y D es mucho mayor que la diferencia en rango entre los puntos A y B.

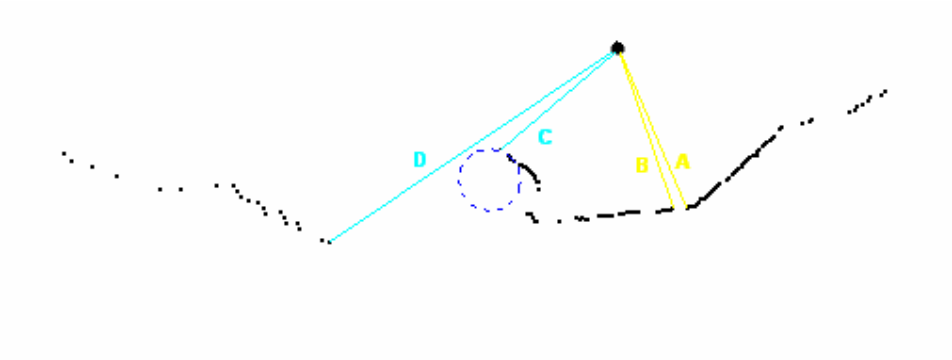



*Figura 3.3.12 Diferentes trayectorias de los pings. Las trayectorias A, B y C, D corresponden a pings adyacentes.*

El método aquí propuesto se basa en esta característica particular que se verifica en el dominio de las coordenadas polares.

Es evidente que cada par de muestras adyacentes, teniendo dos valores de $\rho$ diferentes, será caracterizado de una discontinuidad en rango. La técnica propuesta busca, sin embargo, solo aquellos valores de discontinuidad considerados relevantes. Por esta razón fue necesario definir un valor de umbral respecto al cual establecer si la diferencia en rango considerada puede señalar la presencia (en aquel punto) de un objeto. El valor de este umbral (*threshold*) ha sido definido como sigue:

$$threshold = \frac{diametroTubo}{n} \tag{3.3Q}$$

en donde *n* es un número adimensional que representa y determina el nivel de discriminación que se usa para escoger los "candidatos" en relación con el diámetro del tubo: para calcular este valor nos podemos basar en simples consideraciones geométricas. El valor de *threshold* provee, en consecuencia, el valor que permite considerar la diferencia en rango (entre pings adyacentes) como un "salto" debido a la presencia de un objeto dentro de la escena investigada. Si se verifica la condición expresada por la ecuación (3.3Q) se ha encontrado un candidato. Podemos afirmar por tanto que si:

$$\left|\rho_{i+1} - \rho_i\right| > threshold \tag{3.3R}$$

entonces las muestras relativas a los valores en rango $\rho_i$ y $\rho_{i+1}$, a las que llamaremos $C_i$ y $C_{i+1}$, son consideradas candidatos.

Se puede notar también que mientras mayor es la distancia a la que se encuentra el objeto desde el transductor sonar, mayor será la discontinuidad producida en el valor de rango asociado a las muestras.

La figura 3.3.13 muestra 4 imágenes diferentes producidas por un sonar de tipo *haz de lapiz*, luego del procesamiento y la aplicación del método de detección de objetos. También en estas imágenes se verifica un "salto" entre el *impulso* que alcanza el objeto puesto en el fondo y el *impulso* que golpea el fondo mismo. La muestra que nos interesa es la que corresponde al *impulso* que golpea al objeto: este se caracteriza por un valor de $\rho$ (rango) más pequeño que el de los pings adyacentes, es decir, es el *ping* más cercano al sensor sonar.

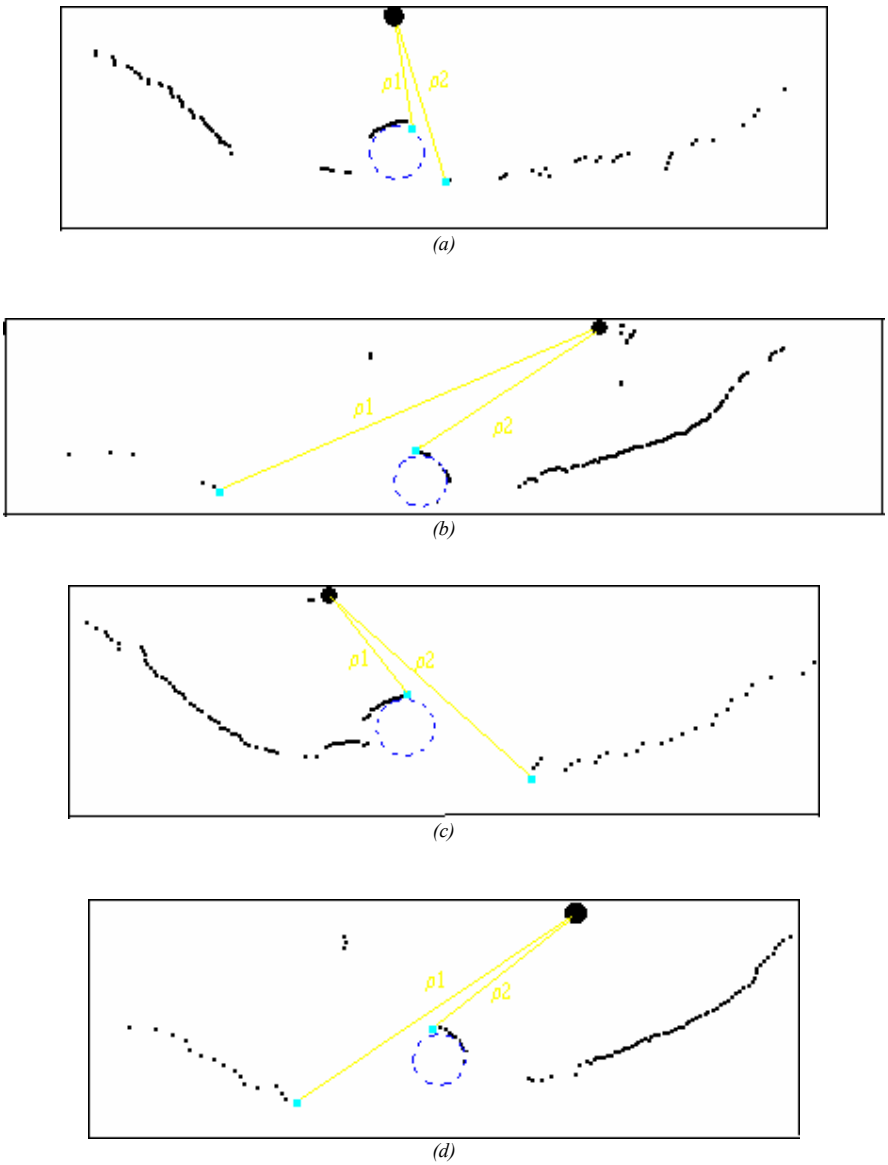



*Figura 3.3.13 Imágenes obtenida por el sistema sonar haz de lapiz. Se observa que existe una discontinuidad en rango entre emisiones(pings) adyacentes cuando hay un objeto presente.*

Es necesaria otra operación para discriminar y escoger un solo ping para cada par de discontinuidad (definido por las dos muestras $C_i$, $C_{i+1}$). Al final se escogen como candidatos solamente aquellas muestras cuyo rango es el mas pequeño de

cada par. Al final de este proceso, por cada par de discontinuidad, se identificará un único candidato.

### 3. Confronto (en el dominio cartesiano) entre los candidatos individuados por ambos sensores sonar

Hasta el momento hemos trabajado con la información de cada transductor sonar por separado. Esto es debido a que la información sobre el rango ($\rho$) y el ángulo ($\theta$) se memoriza separadamente para cada uno de éstos en los archivos *log*.

La eficacia del método propuesto se basa precisamente en esta característica. Si tomamos como punto de referencia para determinar las coordenadas de la tubería el punto *top*, mostrado en la figura 3.3.14, independientemente de la posición en la que se encuentra el transductor que lo detecta, éstas coordenadas deberían ser las mismas (usando un punto de referencia común para todos los transductores). Como consecuencia, la técnica aquí propuesta debería revelar las mismas coordenadas, considerando un cierto valor de tolerancia, a partir de los dos diferentes sensores sonar. Así, el hecho de que el sistema sonar *haz de lapiz* trabaje en modalidad *sensor dual*, nos permite buscar separadamente el punto *top* en los datos adquiridos por ambos transductores. Se tiene, en consecuencia, un instrumento de verificación de los potenciales puntos que identifican la tubería: se busca una pareja de candidatos (relativos a sensores sonar diferentes) cercanos entre ellos.

Basándonos en este principio, el confronto cartesiano consiste en un criterio ulterior para discriminar los candidatos. Se espera que de este confronto surja un único par de candidatos relativos al mismo punto en el dominio cartesiano: el punto *top* (figura 3.3.14).

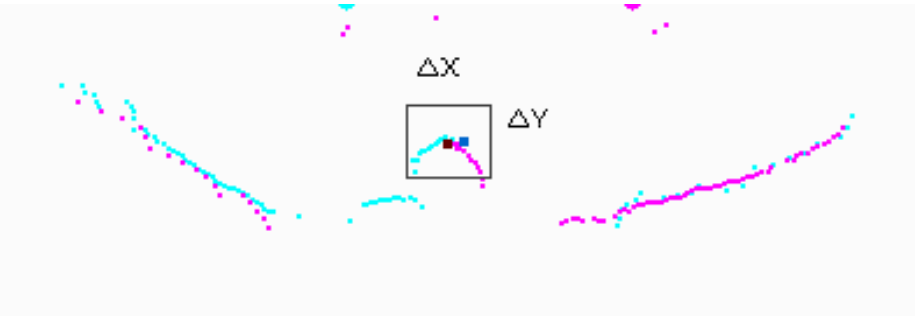



*Figura 3.3.14 Confronto cartesiano de los puntos candidato. El candidato obtenido por una de los sensores sonar debería coincidir idealmente en el espacio con el candidato obtenido por el segundo sensor.*

El procedimiento que se sigue es el siguiente. En primer lugar se transforman las coordenadas polares de cada candidato (de ambos sensores sonar) en sus respectivas coordenadas cartesianas. Se define entonces una región de tolerancia (el cuadrado mostrado en la figura 3.3.14) dentro de la cual se realiza el confronto entre candidatos pertenecientes a sensores diferentes. Si existen dos candidatos pertenecientes a sensores sonar diferentes dentro de esta área, se considera que estos han sido generados precisamente por la presencia de un objeto. La operación de creación de esta área se logra a través de la definición de un parámetro que define el valor de tolerancia aceptable sobre la distancia que puede existir entre dos candidatos para ser considerados referidos a la tubería. Este parámetro expresa un valor de umbral que limita el error máximo que se puede cometer en relación con el valor del diámetro del tubo.

$$\Delta X = \Delta Y = \frac{diametroTubo}{q} \qquad (3.3S)$$

donde q es un valor adimensional que debe ser establecido a priori.

Si se definen las coordenadas del i-ésimo candidato relativo al primer transductor sonar como ($X'_i$, $Y'_i$) y las coordenadas del j-esimo candidato relativo al segundo transductor como ($X''_j$, $Y''_j$), entonces la primera verificación (de cercanía de las muestras $C_i$ y $C_j$) que se efectúa es la siguiente:

$$X'_i - \Delta x < X''_j < X'_i + \Delta x$$

$$Y'_i - \Delta y < Y''_j < Y'_i + \Delta y \qquad (3.3T)$$

Solamente en el caso en el que ambas condiciones sean satisfechas los candidatos $C_i$ y $C_j$ (para el primero y el segundo transductor respectivamente) pasan a la siguiente fase del proceso. En caso contrario, se descartan, puesto que se consideran demasiado alejados entre si como para poder ser ecos relativos al mismo objeto

En la practica, en la mayor parte de los casos estudiados, después del confronto cartesiano queda un solo candidato que será llevado a los pasos de verificación 4 y 5. El software permite también individuar más de un objeto si se encuentra presente. En este caso, al final de este paso será identificado más de un candidato.

### *4. Correspondencia* usando como *plantilla* una semicircunferencia ideal

Una vez detectada la presencia de un objeto en el fondo marino, es necesario determinar si éste corresponde efectivamente a una tubería de un diámetro específico o si se trata de un objeto cualquiera. Si no se ha individuado ningún candidato al final del punto 3, este paso no es necesario y el método termina en el paso anterior.

Este paso, por tanto, tiene como objetivo determinar si el objeto identificado tiene exactamente la forma deseada. También en este caso usamos la técnica de *plantilla correspondiente*. En nuestros requerimientos específicos debemos hacer la correspondencia con una tubería de un diámetro específico posicionada en el fondo marino. El diámetro (*diametroTubo*) de dicha tubería es un parámetro conocido y es la única información que tenemos para determinar la posición de ésta en la escena.

Se procede a un confronto en coordenadas cartesianas entre un conjunto de muestras asociadas a un candidato y una semicircunferencia de radio igual a (*diametroTubo* / 2). Para hacer esto debemos escoger un intervalo de muestras conteniendo al candidato y confrontar cada punto contenido en éste con la ecuación de la circunferencia. Una vez realizado el confronto, es suficiente calcular el error mínimo cometido y finalmente calcular la posición (en coordenadas cartesianas) correspondiente al punto *top*. Veamos en el detalle los pasos que realizan concretamente estas operaciones:

### *4. a Encontrar un intervalo para cada candidato*

La figura 3.3.15 muestra un candidato seleccionado y las muestras contenidas dentro de un intervalo al que llamaremos *entornoCandidato*. Para realizar la correspondencia es necesario escoger un conjunto de muestras vecinas que, en teoría, deberían representar al tubo:

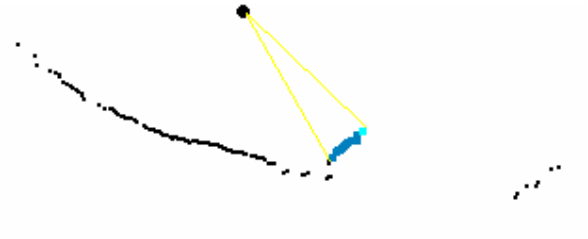


*Figura 3.3.15 Candidato (en celeste) y relativas muestras vecinas seleccionadas (en azul)*

Este paso consiste en la identificación de las muestras contenidas dentro de *entornoCandidato*. Este intervalo está definido por *B emisiones* adyacentes al candidato (con respecto a una sola dirección). La dirección en la cual escoger estas muestras depende de la posición relativa del sensor sonar y de la tubería. Debemos notar que, por el particular contexto aplicativo, cada transductor podrá "ver" solamente la mitad de la tubería (la mitad izquierda o derecha): la otra mitad estará escondida completamente y es justamente la razón por la que se crean las discontinuidades en rango. La mitad visible está formada por muestras localizadas entre el candidato (el punto mas alto) y el transductor sonar (figura 3.3.15).

El valor de *B* es útil para calcular la información sobre el ángulo de apertura del sector relativo a *entorno Candidato*:

$$\Phi = \beta * B \qquad (3.3U)$$

donde $\beta$ es el intervalo de paso mecánico usado por el sistema y B es el número de muestras vecinas tomadas en consideración. $\Phi$ será, por tanto, la apertura angular que comprende todas las *B* muestras.

Conociendo los valores de apertura angular, podemos definir el sector angular preciso a través de la simple operación expresada en la ecuación (3.3V). Se define así un sector circular ($\Delta\theta$ ) dentro del cual se encuentran todas las muestras que deberían ser tomadas en consideración

$$\Delta\theta = (\theta c \pm \Phi) \qquad (3.3V)$$

siendo $\theta c$ el ángulo del ping relativo al candidato y $\Delta\theta$ el sector circular considerado. El signo en la ecuación dependerá, como fue dicho, de la posición del sensor sonar.

El procedimiento realizado para el candidato de uno de los sensores sonar se repite para el otro sensor. Debemos considerar que en este caso el signo de la ecuación (3.3V) será opuesto al caso anterior, puesto que es opuesto el sentido angular en que el sensor recoge las muestras.

Una vez definido y creado el intervalo que será usado para hacer la correspondencia (*matching*), todas las coordenadas de las muestras correspondientes a ambos sensores indistintamente deben ser convertidas a coordenadas cartesianas y luego ordenadas en modo creciente. De esta manera obtendremos el perfil hipotético de la tubería uniendo la información de ambos sensores.

***4. b Encontrar los máximos y los mínimos y llevar el perfil al origen de coordenadas.***

Para hacer el confronto en coordenadas cartesianas es necesario obtener el valor de las coordenadas del centro de la tubería (a partir del perfil obtenido) para luego poder comparar este perfil con la ecuación de la circunferencia con centro en el centro de la tubería. Una manera alternativa de hacer esta comparación es la de llevar todos los valores en coordenadas cartesianas al origen de coordenadas (0,0).

Para comprender mejor este concepto se muestra la figura 3.3.17

*Figura 3.3.17 Perfil final de la tubería obtenido (en negro) usando la información de ambos sensores sonar.*

En la imagen el perfil de la tubería obtenido después del paso previo se puede encontrar en cualquier posición dentro de la escena. En este punto el objetivo es el de "trasladar" el perfil obtenido entorno al origen de coordenadas (0,0) porque el *matching* se hace con una semicircunferencia R (*diametroTubo* /2) centrada precisamente en (0,0).

Para tal fin, calculamos el valor máximo de *y* (*Ymax*) y el valor medio de *x* (*Xmed*) entre todas las muestras que forman el perfil (ver la figura 3.3.18). *Ymax* representa en este caso el valor de offset en *y*, mientras que *Xmed* el de

offset en *x*. Substrayendo estos valores de offset a los valores en coordenadas cartesianas de cada una de las muestras del perfil se obtiene un desplazamiento de éstos hasta llevarlos al origen de coordenadas (0,0), con un cierto margen de error. Es decir que los nuevos valores en *y* y en *x* serán:

$$X_i^{\sim} = X_i - Xmed$$

$$Y_i^{\sim} = Ymax - Y_i \qquad (3.3W)$$

donde ($X_i$, $Y_i$) son las coordenadas de la i-ésima muestra que forma el perfil del tubo.

### *4. c Confrontar el perfil obtenido con la ecuación del circulo*

El corazón de la técnica de *plantilla correspondiente* es el confronto entre el perfil que se obtuvo previamente y la ecuación de la circunferencia de radio R, centrada en el origen. La ecuación de la circunferencia es:

$$x^2 + y^2 = R^2 \qquad (3.3X)$$

Para cada $X_i^{\sim}$ se calcula $Y^{\cdot\cdot}$ de acuerdo a la ecuación de la circunferencia. Obtenemos así un valor calculado (ideal) y un valor real (el del perfil del tubo). Llamemos $Y^{\cdot\cdot}$ al valor calculado y $Y^{\sim}$ al valor real. El confronto entre estos 2 valores se realiza de la siguiente manera:

$$\xi_i = \left| Y^{''}_i - Y_i \right|$$

donde $Y_i = (R2 - X_i^2)^{1/2}$ *(3.3Y)*

El criterio usado en este método para calcular el error entre el perfil real y la semicircunferencia, consiste en el cálculo de una "diferencia de diferencias". La figura 3.3.19 ayuda a entender mejor el principio usado. En la figura se encuentran posicionados 2 semicírculos en las coordenadas (0,0) y (Xc, Yc). El valor calculado de Y dada la X del semicírculo superior es:

$$Y_i = (R^2 - X_i^2)^{1/2}$$

y para el semicírculo inferior es:

$$Y_i' = (R^2 - X_i^2)^{1/2} + Yc$$

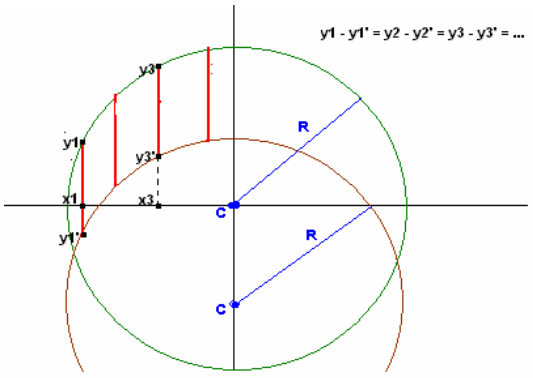



*Figura 3.3.19 Método de la "diferencia de las diferencias" para el calculo del error.*

Esto significa que si hacemos la comparación entre las diferencias en *y* obtenidas ($Y_i - Y_i'$) para todas las muestras, en el caso ideal estas diferencias deben ser iguales. Haciendo por tanto la diferencia entre pares de muestras (($Y_1 - Y_1'$) - ($Y_2 - Y_2'$)…) el resultado de esta operación deberá ser cero. En el caso real el resultado será un valor pequeño, próximo a cero.

Para cada punto ($X_i$, $Y_i$) se calcula $\varepsilon_i$ como se vio precedentemente. La variable error es calculada en el siguiente modo:

$$E = \sum_{i=1}^{I-1} \left| \xi_i - \xi_{i+1} \right|$$

donde I es el número de puntos que constituyen el perfil.

La medida de *error de correspondencia* (mis-matching) adoptada tiene por tanto en consideración la curvatura de la distribución de puntos, admitiendo también un desplazamiento entre la plantilla y la potencial tubería. Esto deriva del hecho que no se conoce *a priori* cuan enterrada en la fosa se encuentra la tubería, es decir, si esa esta semienterrada o enterrada completamente.

### 3. Extracción del error mínimo y cálculo de las coordenadas de la tubería.

El resultado del proceso anterior es la generación de un conjunto de valores (un conjunto de ternas) que representan el error calculado para cada uno de los candidatos así como los valores de *Xmed* y *Ymax* correspondientes a cada uno.

El paso final de este método de detección de objetos es la selección de un único candidato mediante la búsqueda del error mínimo. Al final de los pasos precedentemente descritos, no solo se ha verificado la posible presencia de un objeto dentro de la escena, sino que ha sido también controlado que éste coincida realmente con una tubería de diámetro *diametroTubo*. Una vez individuado el error mínimo, que debe ser también inferior a un cierto valor, se usan los correspondientes *Xmed* y *Ymax* y se calculan las coordenadas de la tubería ( $\overline{x}$ , $\overline{y}$ ) en el siguiente modo:

$$\bar{x} = X_{med}$$

$$\bar{y} = Y_{max}$$

En el siguiente capitulo se mostraran los resultados finales obtenidos con este método, así como del primer método propuesto.

# CAPITULO 4

## RESULTADOS

El objetivo del proyecto de investigación dentro del cual se inserta esta tesis es el de extender la operabilidad de los sistemas VOR (*Vehiculo Operado Remotamente*) en ambientes con escasa visibilidad, debida sobretodo a la turbiedad del agua. El uso de las telecámaras, en estos contextos, se hace vano y es necesario el empleo de sistemas sonar.

Es en este contexto aplicativo que se inserta el trabajo presentado hasta aquí, que ha consistido en la creación de un código en lenguaje C++ en grado de generar imágenes en formato bitmap (bmp) a partir de la información de "bajo nivel" recogida por los sistemas sonar utilizados por la SONSUB. En este capitulo serán presentados los resultados obtenidos luego de las elaboraciones expuestas en los capítulos precedentes sobre estas señales.

Es importante subrayar el hecho de que los sonar utilizados por la SONSUB son producidos por la TRITECH que, además de proveer del sonar, ha provisto también un software en grado de procesar los ecos de retorno del proceso de insonificación y de producir las imágenes relativas. Por esta razón los resultados

presentados en este capitulo serán también comparados con las imágenes producidas por este software en modo de dar una mejor comprensión de los resultados alcanzados.

Antes de pasar a la presentación de las imágenes en si, recordamos una vez mas que el trabajo desarrollado en esta tesis puede ser subdividido en dos grandes secciones según el sistema de adquisición y, por tanto, según el tipo de sonar usado en la operación de investigación de la escena de interés.

En este capitulo mostraremos algunos ejemplos de imágenes producidas tanto a partir de los datos adquiridos por los sistemas sonar *haz de abanico* (sección 4.1) como a partir de los datos adquiridos por los sistemas de tipo *haz de lapiz* (sección 4.2). Serán por tanto expuestos, paso a paso, las mejoras y las ventajas aportadas por el uso de las técnicas desarrolladas, comparando las imágenes originales con aquellas producidas en este tesis.

## 4.1 HAZ DE ABANICO

Se expuso con detalle en el capitulo 2, como, con respecto a los sistemas sonar de tipo *haz de abanico*, se ha procedido al mejoramiento de la calidad visual de las imágenes acústicas a partir de señales de bajo nivel, o sea las señales directamente adquiridas por el transductor sonar.

Para tal fin se procedió a una distribución dinámica y optimizada de los niveles de luminosidad, al desarrollo de una técnica de interpolación a partir de señales de bajo nivel, a la reducción del ruido speckle, al énfasis de las sombras acústicas y

finalmente a la eliminación de los ecos múltiples y colas de ecos intensos. El conjunto de estas técnicas fue experimentado en datos reales adquiridos por un sistema sonar *haz de abanico* durante algunas pruebas en el mar.

El sistema sonar *haz de abanico* usado ha obtenido datos relativos a la presencia de dos tipos de objetos diferentes, una tubería suspendida verticalmente en el agua, y una estructura definida como "estrella-triangulo" puesta sobre el fondo marino (Figura 4.1.1). En ambos casos la turbiedad del agua era bastante elevada.


*(a)* *(b)*

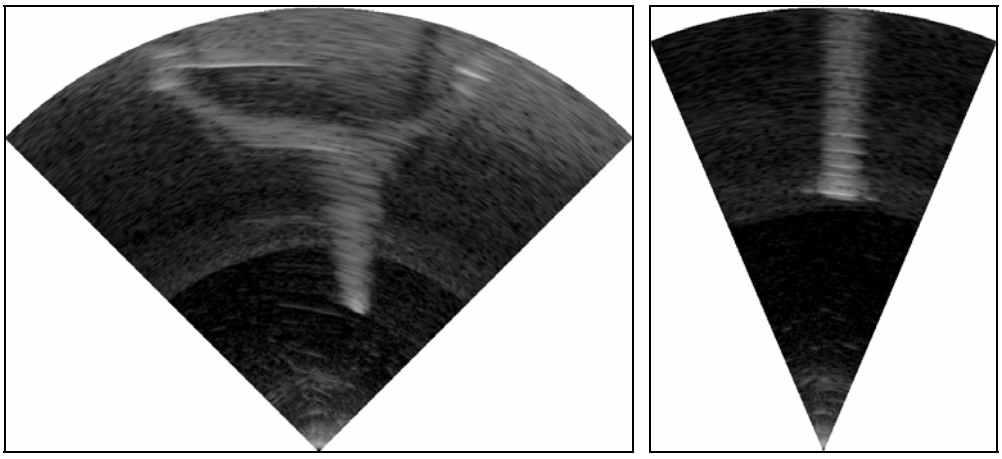
*Figura 4.1.1 Estructuras usadas para realizar las pruebas submarinas en las cuales fue empleado el sistema sonar haz de abanico. (a) estructura "estrella-triangulo" (b) tubería*

Las imágenes producidas por el sistema sonar de tipo *haz de abanico*, son imágenes a niveles de gris, o sea los píxeles de la imagen asumen valores entre 0 y 255 (256 tonalidades de gris), donde el 255 representa el eco mas fuerte. Tales valores se interpretan en una escala lineal.

La operación más simple posible, a partir de los datos de bajo nivel, consiste en el paso del dominio de las coordenadas polares en el que son memorizados los

datos, al dominio cartesiano (scan-conversion), en el cual vienen representados para ser visualizados.

En la Figura 4.1.2 se muestra una imagen formada luego del procedimiento de scan-conversión e interpolación. Como se puede observar, no obstante la presencia de mucho ruido, en la imagen es visible la forma que caracteriza a la estructura "estrella-triangulo" (Figura 4.1.2 (a)) y la tubería (Figura 4.1.2 (b)).



(a) (b)

*Figura 4.1.2 Imágenes de la estructura "estrella-triangulo" y de la tubería mostradas en la figura 4.4.1. Tales imágenes fueron obtenidas usando solo la operación de barrido y conversion e interpolación de los datos de bajo nivel.*

La técnica de interpolación utilizada para producir la imagen de la Figura 4.1.2 trabaja en el mismo sistema de coordenadas polares del sistema sonar y permite calcular los valores que hay que asignar a cada píxel en base a la media pesada de las cuatro muestras acústicas más cercanas. Los detalles sobre esta técnica fueron presentados en el capitulo 2.

Los pasos fundamentales, dirigidos al mejoramiento de la calidad de la imagen y de la mayor comprensión de la escena bajo examen (presentados en el capitulo 2), fueron:

- Gestión dinámica y optimizada de los niveles de gris.
- Reducción del ruido speckle (filtro de Frost)
- Mejoramiento del contraste

**Gestión dinámica y optimizada de los niveles de luminosidad**

La primera elaboración aplicada fue dirigida a optimizar el uso del rango de valores que los píxeles pueden asumir, en modo tal de aprovechar mejor toda su extensión: esta operación ocurre dinámicamente, es decir, imagen por imagen. Para hacer esto, como fue expuesto en la sección 2.2.1, se ejecuta una operación de estrechamiento de los valores de las muestras, gracias a la definición de dos parámetros LOW y HIGH. Recordemos que HIGH está calculado como la media del 1% de los valores de intensidad mas altos presentes en la imagen considerada. En la Figura 4.1.3 se pueden observar las mismas imágenes de la Figura 4.1.2 obtenidas sucesivamente a la aplicación de esta nueva elaboración, donde la distribución de los valores comprendidos entre LOW y HIGH es lineal.

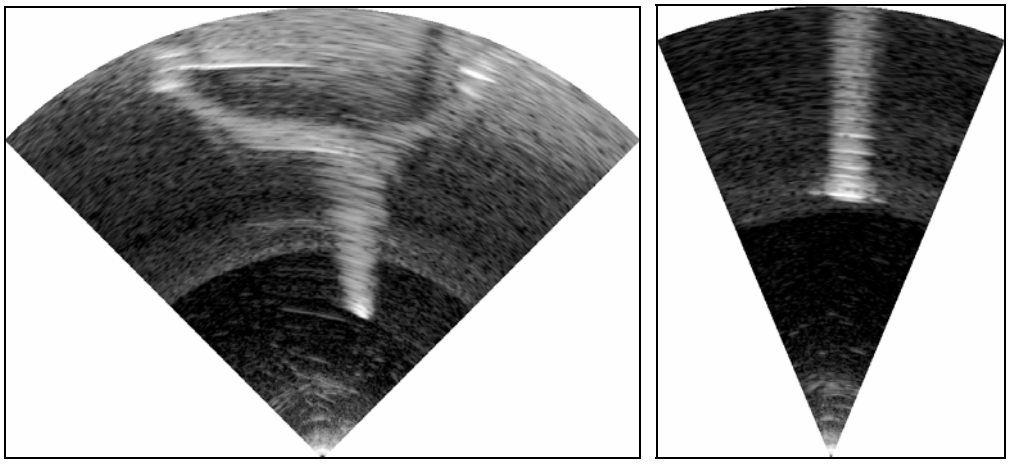


*Figura 4.1.3 Imágenes obtenidas sucesivamente a la aplicación de una ley de asignación de los valores de intensidad de los píxeles lineal (entre los valores LOW y HIGH).*

Han sido además tomados en consideración diferentes comportamientos de la curva de asignación de los valores comprendidos entre LOW y HIGH.

En la Figura 4.1.4 se pueden observar dos imágenes producidas a través de la aplicación de una ley de asignación diferente a la lineal.

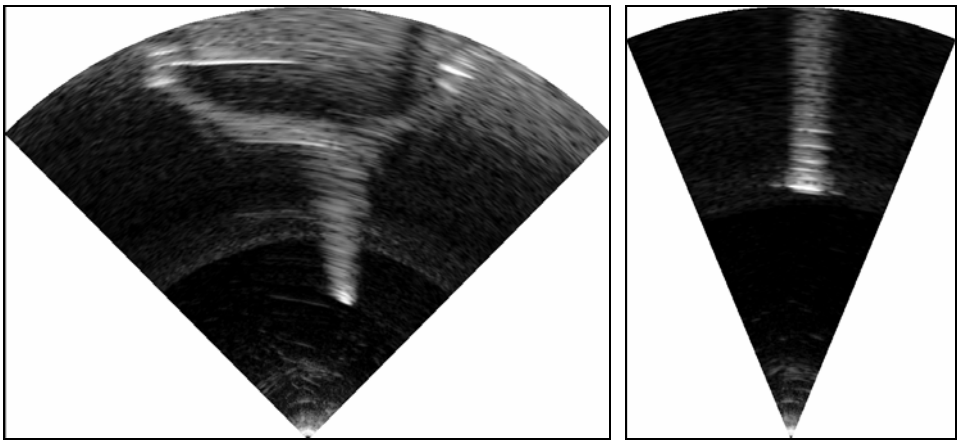
Esta operación fue dirigida a exaltar la diferencia que existe entre los ecos debidos a la presencia del objeto y aquellos puntos de la imagen generados por causa de la presencia de ecos múltiples, no queridos, que causan un efecto ruidoso sobre la imagen, reduciendo su comprensión.

El comportamiento real de la curva de asignación de los niveles de gris, depende del valor del parámetro $\gamma$ (ver la ecuación (2.B) en el párrafo 2.2.1). La elección de su valor fue hecha empíricamente: se ha elegido el valor que, en promedio, ha parecido producir los mejores resultados, también de acuerdo a las elaboraciones ejecutadas sucesivamente. Las imágenes mostradas se refieren a

un valor de $\gamma$ igual a 1.7. Respecto a las imágenes originales, con tal elección se ha producido una atenuación de la luminosidad.

El grafico que representa la curva que ha generado los resultados presentes en la Figura 4.1.4, fue mostrado en la sección 2.2.1 (Figura 2.2.1).

De un atento análisis de las imágenes mostradas en la Figura 4.1.4, se puede observar como las mejoras introducidas son evidentes y han permitido reducir la presencia en la imagen de todos aquellos puntos que no llevan información sobre los objetos contenidos en la imagen. Al mismo tiempo lo que lleva información sobre el objeto se preserva, por ejemplo la sombra acústica presente en la Figura 4.1.4 (a). La sombra acústica ofrece la posibilidad de estimar la altura de los objetos bajo investigación. Si esta información se perdiera, de hecho, la sola visión de la imagen producida por el sonar no proveería indicación alguna sobre la altura de los objetos representados.



*(a)* *(b)*

*Figura 4.1.4 Imágenes obtenidas sucesivamente a la aplicación de la curva de asignación de los niveles de gris con $\gamma$=1.7. (a) estructura "estrella-triangulo"; (b) tubería*

**Reducción del ruido speckle (filtro de Frost)**

Las imágenes propuestas son ya imágenes de calidad tal de permitir la interpretación de la escena bajo examen, aun cuando son evidentes en éstas, algunos disturbios debidos a, por ejemplo, la presencia de ecos múltiples.

Un ulterior disturbio del cual son afectadas las imágenes propuestas, como todas las imágenes acústicas, es aquel relativo a la presencia de ruido de tipo speckle (puntos aislados). En realidad, en los datos provistos por la SONSUB, este fenómeno no es particularmente evidente al ojo humano, sino que está presente y podría comprometer la eficiencia de un sistema automático de *detection*.

La operación de reducción del ruido speckle realizada mediante el uso de un filtro adaptivo, el filtro de Frost (1). Los detalles sobre esta técnica fueron mostrados en la sección 2.2.3

En la Figura 4.1.5 se muestran dos imágenes, referidas a las mismas escenas mostradas en las Figuras 4.1.4 (a) y (b), obtenidas luego de esta elaboración.

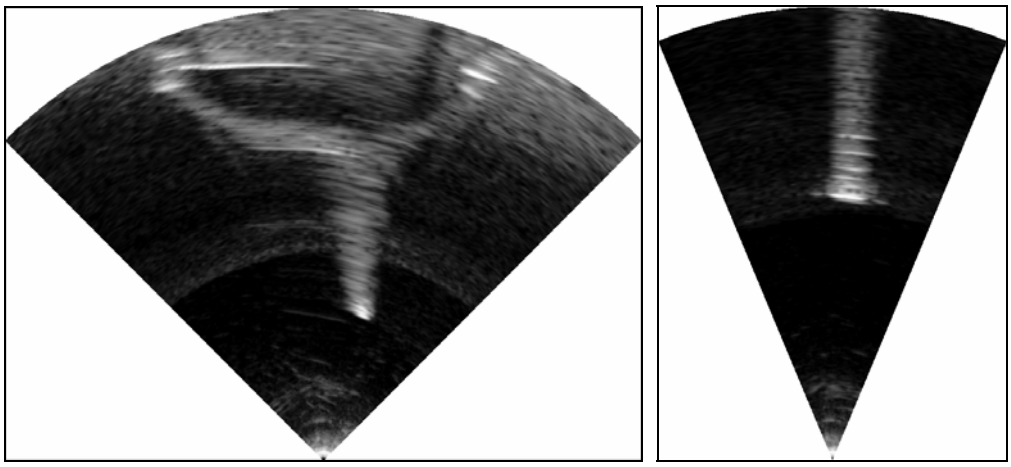


*Figura 4.1.5 Gestión dinámica y optimizada de los niveles de gris ($\gamma$=1.7) y aplicación del filtro de Frost (3×3) para la reducción del ruido.*

Un parámetro crítico en el filtro de Frost es la dimensión de la máscara que se hace correr sobre la imagen.

En las siguientes imágenes se puede notar como es mas marcada la reducción del ruido speckle al crecer la dimensión del filtro, pero como ésta comporta una desagradable perdida en la percepción de los detalles y está acompañada de un efecto de "desenfoque" de la imagen. Para filtros de dimensión 3×3 y 5×5, estos efectos son más bien contenidos, mientras que inician a ser muy marcados para filtros de dimensiones superiores. Son un ejemplo de esto las imágenes mostradas en la Figura 4.1.6 obtenidas aplicando un filtro de 7x7.

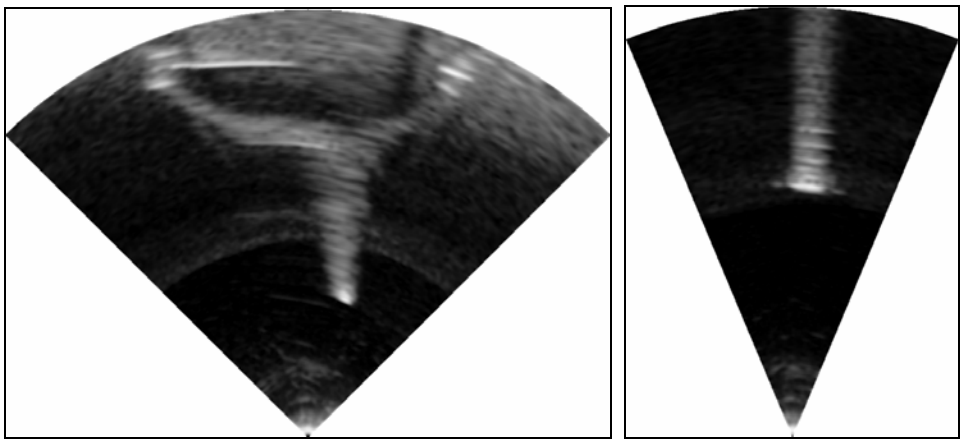


*Figura 4.1.6 Gestión dinámica y optimizada de los niveles de gris ($\gamma$=1.7) y aplicación del filtro de Frost (7×7) para la reducción del ruido speckle*

Un atento análisis de los resultados presentados en la figura 4.1.5 muestra una eficaz reducción del speckle: puntos únicos aislados fueron eliminados al pasar el filtro, mientras que se ha creado una especie de efecto de "homogenización" de los otros puntos.

La aplicación de esta técnica lleva una contribución importante al mejoramiento de la imagen, en particular cuando esta se usa en manera combinada con la técnica de mejoramiento del contraste presentada a continuación que, de algún modo, compensa los efectos no deseados legados al uso del filtro de Frost.

**Mejoramiento del contraste**

La última elaboración realizada consiste en la aplicación de una técnica recientemente propuesta en la literatura científica para aplicaciones ópticas convencionales, dirigida al mejoramiento del contraste (2). Este es un método muy potente que permite, al variar los valores escogidos para sus parámetros, obtener casi siempre óptimos resultados aun cuando sean muy diferentes entre ellos.

El mayor problema para la aplicación de esta técnica, por tanto, ha sido encontrar una configuración de estos parámetros tal de obtener buenos resultados independientemente de los datos a procesar. Se ha buscado así una combinación de los parámetros (*M*, *g*, *k, threshold*) tal de influir positivamente en la presentación de la imagen y, al mismo tiempo, de no introducir ecos indeseados.

Los valores de los parámetros elegidos ($M$=4, $g$=0.2, $k$=0.8, *threshold*=15), son tales de conciliar estos dos aspectos: si de una parte se puede notar un notable mejoramiento de la calidad de las imágenes, por otra parte no se encuentran efectos tales de hacer la imagen innatural. Existen, sin embargo, muchas otras

configuraciones que han provisto asimismo óptimos resultados (por ejemplo *M*=2, *g*=0.6, *k*=0.8, *threshold*=15).

Se muestran en la figura 4.1.7 los resultados producidos por la aplicación de esta técnica utilizada junto a la técnica de optimización de los niveles de luminosidad, cuyos resultados fueron reportados en la Figura 4.1.4.

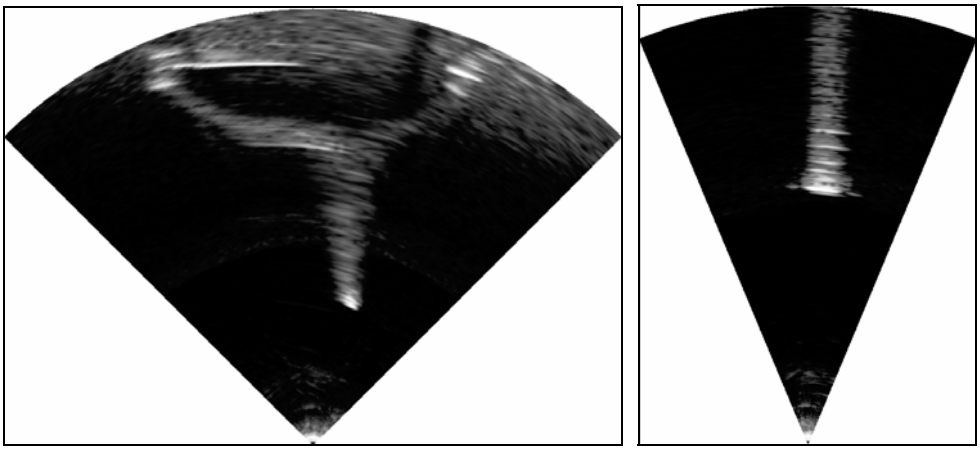


*Figura 4.1.7 Gestión dinámica y optimizada de los niveles de gris ($\gamma$=1.7) y aplicación de la técnica de mejoramiento del contraste.*

En la Figura 4.1.8 se muestran, en cambio, las imágenes producidas por la utilización combinada de la distribución optimizada de los niveles de luminosidad, de la aplicación del filtro de Frost (3×3) y de la técnica de mejoramiento del contraste:

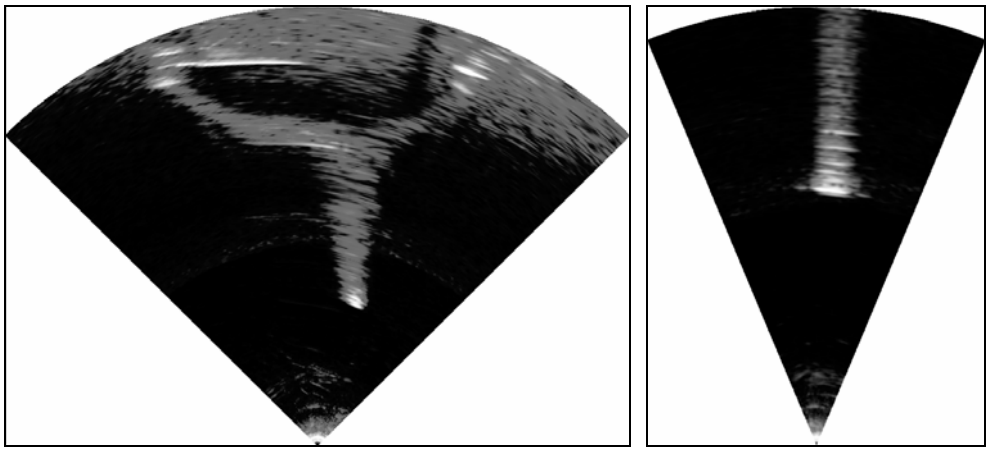


*Figura 4.1.8 Gestión dinámica y optimizada de los niveles de gris ($\gamma$=1.7), aplicación del filtro de Frost (3×3) y método de mejoramiento del contraste.*

En la Figura 4.1.9, en cambio, se reportan los resultados al variar solamente la dimensión del filtro de Frost (7x7). Es evidente el efecto de "desenfoque" introducido, debido al aumento de las dimensiones del filtro, con respecto a aquel mostrado en la Figura 4.1.6. Tal efecto es sin embargo reducido por la sucesiva aplicación de la técnica de mejoramiento del contraste.

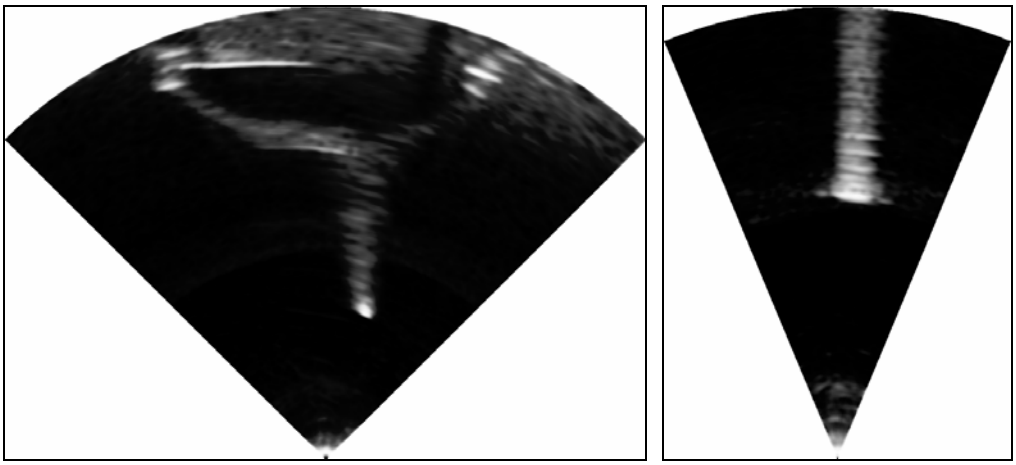
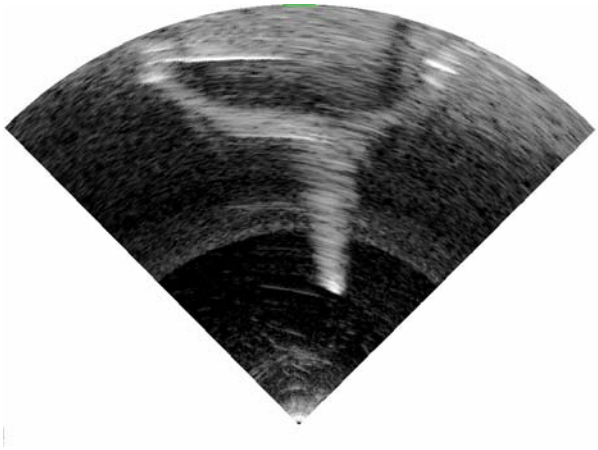


*Figura 4.1.9 Gestión dinámica y optimizada de los niveles de luminosidad, aplicación del filtro de Frost (filtro 7×7) y método de mejoramiento del contraste.*

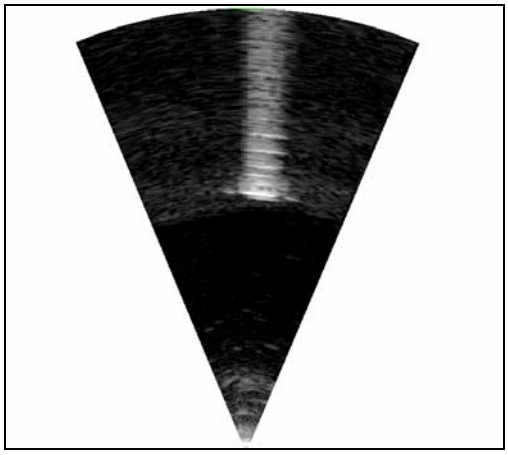
En la secuencia de imágenes propuesta, es evidente el mejoramiento aportado con respecto a las imágenes de partida (Figura 4.1.2). Ha sido reducido eficazmente el ruido de fondo bien visible todavía en la Figura 4.1.3, donde la única elaboración ejecutada consiste en la gestión dinámica de los niveles de luminosidad, mediante una ley de asignación lineal.

Es importante subrayar una vez más como viene mantenida la información contenida en la imagen original. En el caso especifico de la estructura bajo investigación, se debe subrayar el mantenimiento de la sombra acústica.

Se reportan finalmente las imágenes originales producidas por el sistema sonar de la TRITECH, de manera que sean evidentes los mejoramientos realmente aportados por las técnicas desarrolladas en el trabajo desarrollado. (Figura 4.1.10)



*(a)*

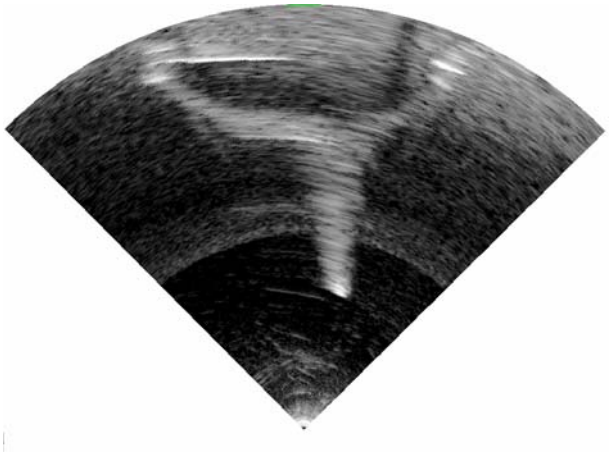


*(b)*

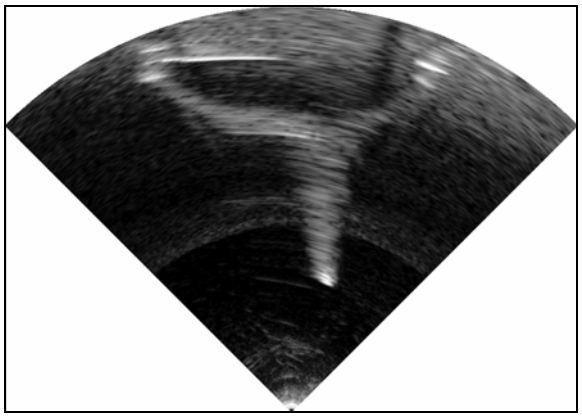
*Figura 4.1.10 Imagen original de la estructura "estrella-triangulo" (a) y de la tubería (b) generadas por el sonar haz de abanico*

En forma general se puede afirmar que la calidad de las imágenes ha mejorado luego de las elaboraciones ejecutadas. Con el fin de hacer más fácil la visualización de las mejoras aportadas con cada técnica desarrollada, a continuación vienen mostrados y confrontados algunos de los resultados ya presentados.

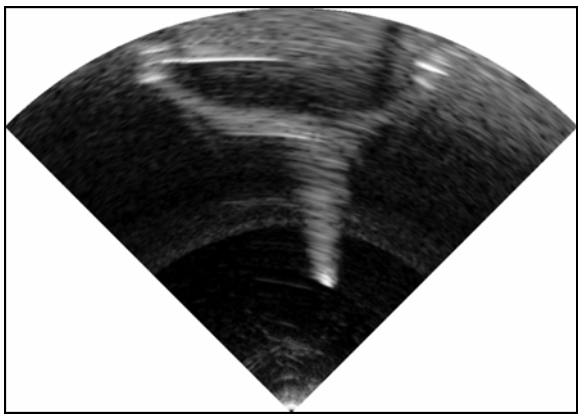
En particular, en la Figura 4.1.11, se muestran cuatro imágenes relativas a la estructura "estrella-triangulo". La primera (4.4.11 (a)) es la imagen original producida por el sonar, la segunda (4.4.11 (b)) es la sucesiva a la aplicación de la curva de optimización de los niveles de gris ($\gamma$=1.7), la tercera (4.4.11 (c)) es relativa al uso del filtro de Frost (3x3) adicional a la optimización de los niveles de gris, y la cuarta (4.4.11 (d)) es relativa al uso combinado de todas las técnicas vistas arriba (con $\gamma$=1.7, filtro de Frost 3×3 y parámetros de método de *contrast enhancement*: *M*=4, *g*=0.2, *k*=0.8, *threshold*=15):



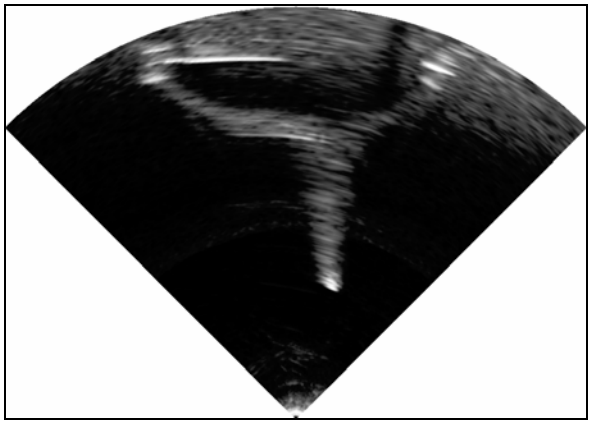
*(a)*



*(b)*



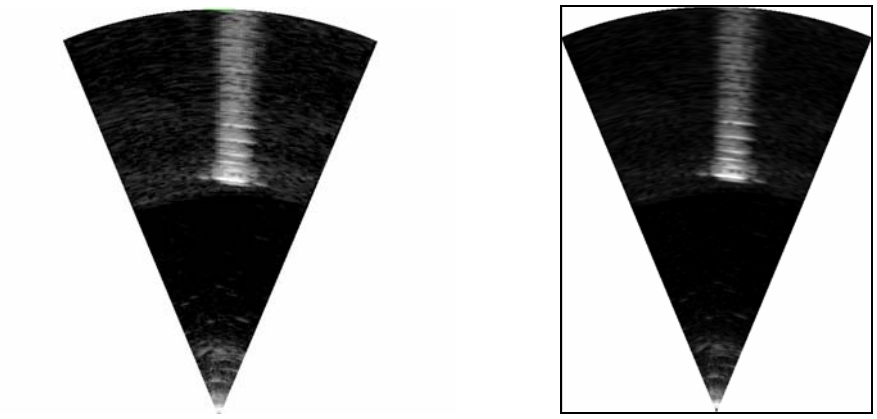
*(c)*



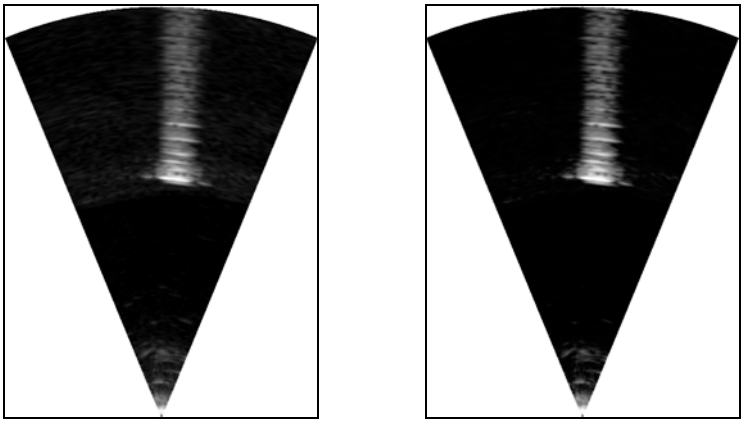
*(d)*

*Figura 4.11* *Confronto de las imágenes originales de la estructura "estrella-triangulo" (a)con las imágenes obtenidas aplicando la optimización de los niveles de gris(b), añadiendo la reducción del ruido speckle (c) y el método de contrast enhancement (d).*

En la Figura 4.1.12 se muestran las imágenes relativas a la tubería puesta verticalmente en el agua, luego de las mismas elaboraciones realizadas para producir las cuatro imágenes de la Figura 4.1.11:



*(a)* *(b)*



*(c)* *(d)*

*Figura 4.12 Confronto de la imagen original de la tubería (a) con las imágenes obtenidas aplicando la optimización de los niveles de gris (b), añadiendo la reducción del ruido speckle (c) y el método de mejoramiento del contraste (d).*

En lo que respecta a los sistemas sonar de tipo *haz de abanico*, se ha logrado alcanzar el objetivo de producir una imagen acústica de una calidad mejor de aquella propuesta en la pantalla del software provisto por la TRITECH.

Trabajando directamente sobre señales de bajo nivel, además, se ha podido mejorar la calidad de las imágenes, manejando los datos diversamente de lo previsto por el software de la TRITECH: por ejemplo, la gestión dinámica y optimizada de los niveles de luminosidad de los píxeles hace que estos vengan utilizados mejor para presentar en la pantalla los datos acústicos de partida. Todas estas operaciones, globalmente, permiten una mejor lectura de las informaciones contenidas en los *log file*, en los cuales son almacenadas las muestras de las señales recibidas por el sistema sonar.

## 4.2 *HAZ DE LAPIZ*

En esta sección serán mostrados los resultados de las elaboraciones ejecutadas sobre los datos adquiridos por los sistemas sonar de tipo *haz de lapiz*.

Es importante evidenciar que el término imagen, en este contexto, no indica una representación de la escena a tonalidades de gris como para los sistemas *haz de abanico*, sino que indica una imagen binaria del perfil de la escena investigada, motivo por el cual estos sistemas sonar han sido definidos como perfiladores.

Los procesos seguidos fueron orientados exclusivamente a la localización automática de objetos de forma particular, la tubería de la Figura 4.1.1 (b), presente en la imagen del perfil producida a partir de los datos adquiridos, mediante la individuación de su sección conocida *a priori*.

Las técnicas fueron probadas en datos reales adquiridos por un sistema sonar de tipo *haz de lapiz,* funcionando en modalidad *sensor dual*, durante pruebas en el mar, obteniendo resultados satisfactorios en términos de precisión en la localización del objeto y de carga computacional.

La escena de interés, por tanto, ha sido "iluminada" desde dos puntos de vista diferentes, puestos a lo largo de una línea de base común. Los sectores de barrido de los dos transductores se sobreponen parcialmente (los dos perfiles producidos pueden también ser examinados distintamente) con el objetivo de iluminar mejor la región bajo observación.

Todos los perfiles mostrados fueron creados a través de la unión de la información de los dos sensores sonar con el cuidado de "colorear" los dos perfiles en modo diferente, de manera tal de poder hacer evidente cuales puntos fueron adquiridos por un sensor sonar y cuales por el otro.

En lo que concierne el contexto en el cual han sido adquiridos los datos de interés, se ha precisado que la escena bajo investigación del sistema es relativa a la excavación de una fosa y al posicionamiento de una tubería en su interno, sobre el fondo submarino.

### 4.2.1 DETECCIONES CONGRUENTES

A continuación vienen reportadas solo algunas de las centenares de imágenes que han sido producidas gracias al uso del software desarrollado. En la Figura 4.2.1 es visible uno de los perfiles producidos en el trabajo realizado: se puede notar claramente el perfil del fondo marino, los dos transductores sonar, y la presencia de la tubería en el centro de la escena. La tubería se reconoce fácilmente a simple vista.

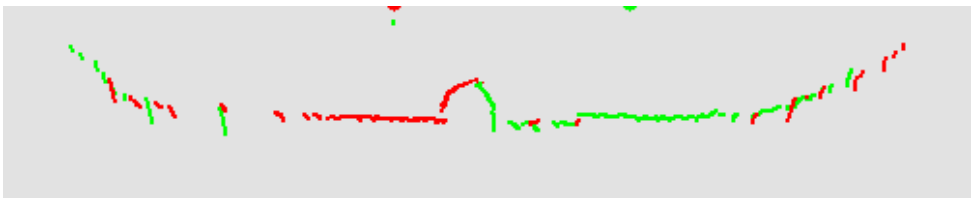


*Figura 4.2.1 Imagen de un perfil. Se pueden notar los dos sensores sonar (en rojo y verde) en alto, y la semicircunferencia (comprendida entre los dos sensores sonar) del perfil de la tubería de interés.*

En la Figura 4.2.2 es visible la misma imagen del perfil mostrada en la Figura 4.2.1 luego de la aplicación de la versión del método de deteccion de objetos presentado en esta sección y que fue denominado “detecciones congruentes”.

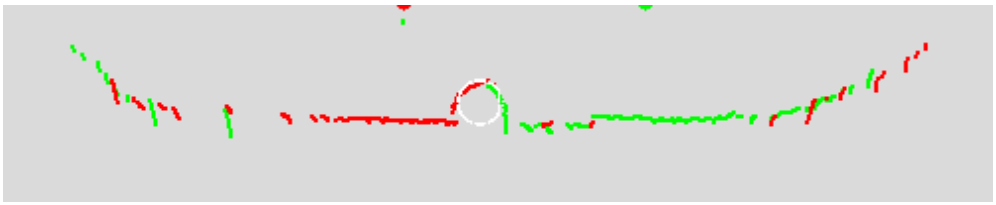


*Figura 4.2.2 Imagen del mismo perfil mostrado en la Figura 4.2.1, producido a continuación del procedimiento de detecciones congruentes*

Desgraciadamente no es muy frecuente obtener perfiles de la escena investigada así detallados: frecuentemente se tiene que trabajar con datos mucho mas esparcidos o situaciones donde identificar la posición de la tubería es una operación complicada también a simple vista. En la Figura 4.2.3, por ejemplo, se muestra un perfil de una escena que ha producido un resultado errado por parte del software desarrollado. Como se puede observar, se pueden tener dificultades para definir la posición de la tubería al interior de la fosa también a simple vista:

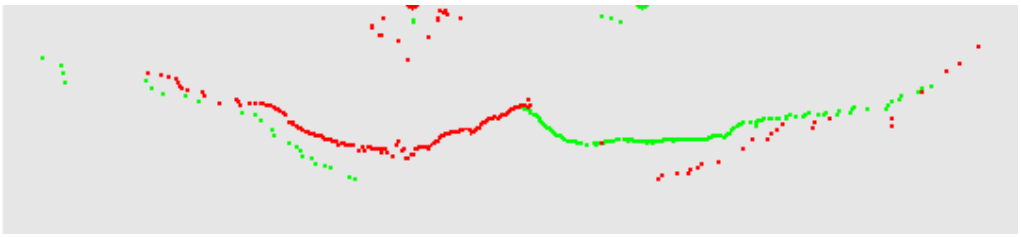


*Figura 4.2.3 Situación difícil de afrontar para el procedimiento de detecciones congruentes. La posición de la tubería no es así evidente como en el perfil bajo examen en la Figura 4.2.1*

En la Figura 4.2.4 se reporta el caso donde esta versión de deteccion de objetos falla. Se nota como el perfil seleccionado por el software es realmente muy similar a la semicircunferencia buscada: esta similitud es tal de inducir un error

en la técnica aquí propuesta. La versión de *object detection* propuesta en la sección siguiente, precisamente por su peculiar modo de clasificación de las posibles posiciones de la tubería, puntos en los cuales se manifiesta una cierta discontinuidad en rango, es menos sensible a este tipo de situaciones pero está inducida al error en otras circunstancias, como veremos mejor en la sección sucesiva.

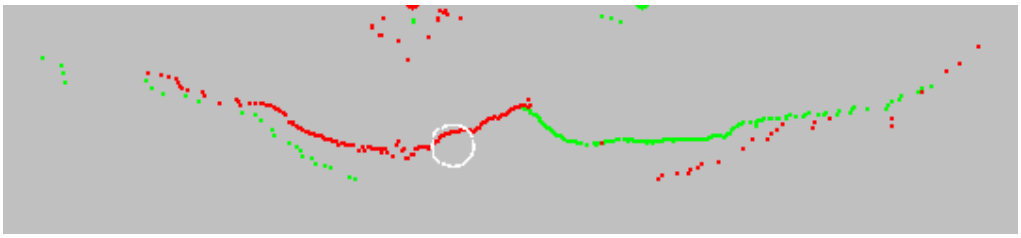


*Figura 4.2.4 Ejemplo de error de la técnica de detecciones congruentes, debido a la ambigüedad de la escena bajo investigación.*

La imagen de la Figura 4.2.5, en cambio, ejemplifica como los datos pueden ser distribuidos dentro de la imagen en modo absolutamente disperso. En la Figura 4.2.6, luego, se ha evidenciado como pueden estar presentes también muchos "huecos". En ambos casos, sin embargo, se ha logrado identificar correctamente la posición de la tubería.

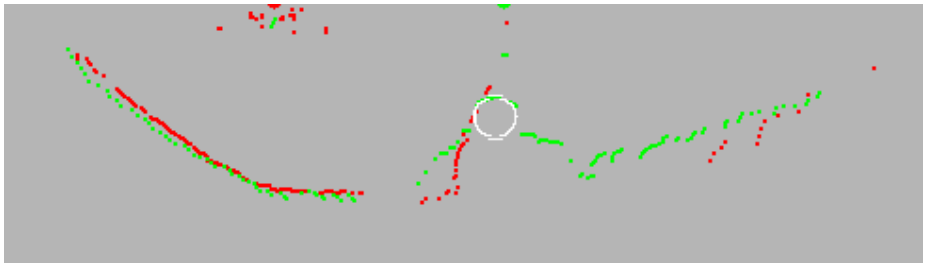


*Figura 4.2.5 Imagen de un perfil. Nótese la dispersión de los puntos que lo componen.*

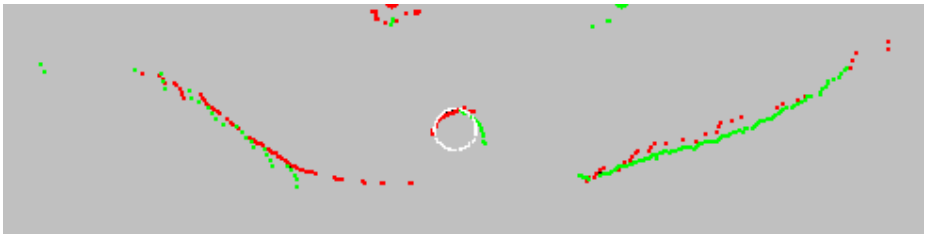


*Figura 4.2.6 Imagen en la cual son particularmente evidentes "huecos" debido a la dispersión de los datos del perfil de tubería, sin embargo es aun evidente y ha sido identificada correctamente.*

Como se puede observar, las imágenes del perfil no siempre son de fácil interpretación. Además, frecuentemente están presentes en la imagen muchos puntos debidos a la presencia de tierra y polvo en las cercanías del transductor sonar. Estos puntos son indeseados y deben ser excluidos de las elaboraciones sucesivas para que sea posible el correcto funcionamiento de la técnica desarrollada (Figura 4.2.7).

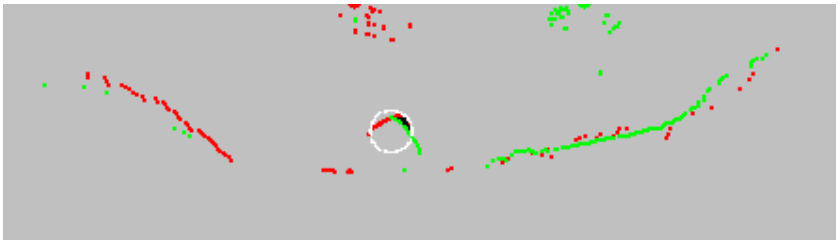


*Figura 4.2.7 Imagen del perfil donde es evidente la presencia de muchos puntos de ruido en las cercanías de los dos sensores sonar.*

Esta técnica ha sido probada en mas de 300 imágenes de perfiles producidas por sistemas sonar funcionando en modalidad *sensor dual*. En base a las pruebas realizadas se puede concluir que la precisión de este método es de cerca del 90%. En el 6% de los casos, en cambio, se ha localizado una posición errada, mientras que en el restante 4% no se ha producido la localización.

### 4.2.2 *DISCONTINUIDAD EN RANGO*

La segunda versión de la técnica de detección de objetos propuesta utiliza información sobre la distancia ($\rho$) de cada punto desde el transductor sonar para poder seleccionar los puntos candidatos que representan la tubería.

Aun cuando esta aproximación es muy diferente de aquella expuesta en el párrafo precedente, los resultados obtenidos, sea en términos de precisión como en términos de porcentual de error, son muy similares. En la Figura 4.2.8 se muestra la misma imagen presentada en la Figura 4.2.1 luego del uso de la técnica en cuestión. También en este caso, la posición de la tubería fue identificada correctamente:

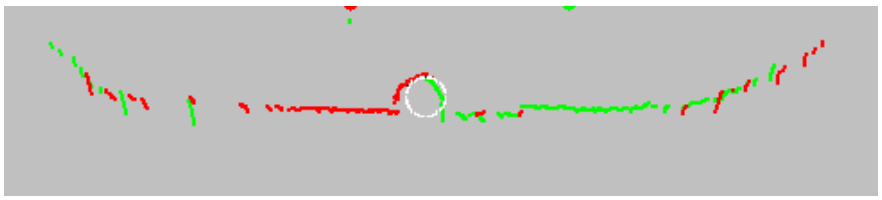


*Figura 4.2.8 Imagen del perfil de la misma escena mostrada en la Figura 4.2.1. Resultado obtenido luego del procedimiento de discontinuidad en rango.*

En las Figuras 4.2.9 y 4.2.10, como ejemplo, son mostradas dos ulteriores imágenes de perfiles en las cuales viene correctamente localizada la tubería:

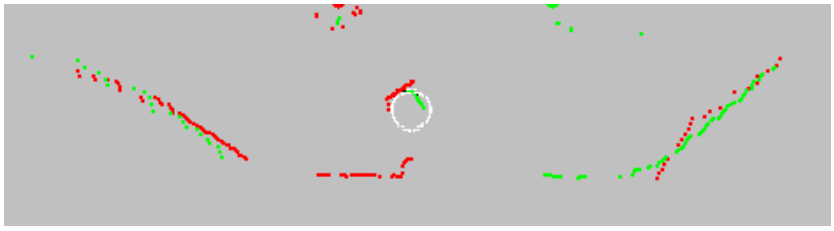


*Figura 4.2.9 Ejemplo de imagen de perfil en la cual la tubería viene localizada correctamente luego del procedimiento de discontinuidad en rango.*

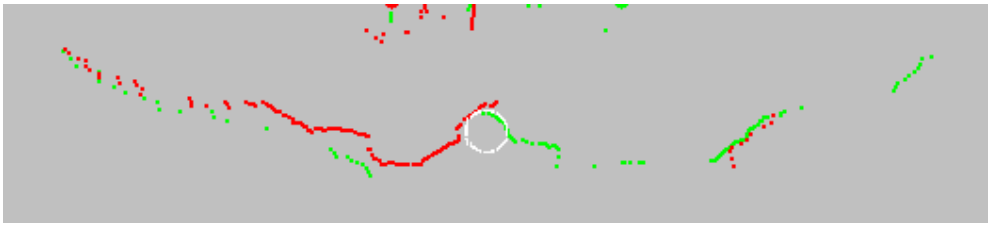


*Figura 4.2.10 Ejemplo de imagen de un perfil en la cual la tubería viene localizada correctamente luego del procedimiento de discontinuidad en rango.*

La necesidad de desarrollar dos versiones diferentes de una técnica de deteccion de objetos, es debida a las particulares dificultades de interpretación de los datos adquiridos, causada, en particular de su dispersión.

Ninguna de las dos versiones ha provisto una porcentual de éxito del 100%, pero los resultados obtenidos son mas que satisfactorios en ambos casos. También esta técnica basada en la discontinuidad en rango, a veces está sujeta al error. En la Figura 4.2.11 se tiene un ejemplo de esto:

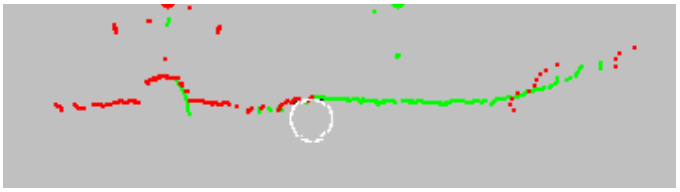


*Figura 4.2.11 Ejemplo de error del procedimiento de discontinuidad en rango*

Como se puede notar, esta versión es inducida a cometer un error en situaciones diferentes con respecto a las vistas en el caso de detecciones incongruentes. En este sentido, la Figura 4.2.11 muestra un caso particularmente critico: la tubería está justamente bajo uno de los dos sensores sonar, no creando para esta aquella situación de discontinuidad en rango en las muestras que la representan.

Este hecho produce una posible dificultad en la selección correcta de los puntos candidatos y una consecuente posible situación de error, justamente como en el caso de la escena bajo investigación mostrada en la Figura 4.2.11.

En conclusión es justo recordar como también esta versión del método de deteccion de objetos ha sido probada en mas de 300 imágenes de perfil adquiridas por el sistema sonar *haz de lapiz* funcionando en modalidad sensor dual. De estas se ha recavado una estima de la porcentual de localizaciones correctas de alrededor del 87%, de la localización de la tubería en una posición errada de cerca del 5% y de la no localización, del 8%.

Las porcentuales de error reportadas tanto en esta sección como en la anterior, confirman que las dos versiones del método propuesto tienen niveles de prestación muy similares. Al mismo tiempo se debe subrayar como estas dos versiones, no obstante se basen en una única aproximación (*template matching*), trabajan en modo profundamente diverso. Esto permite tener en la casi totalidad de los casos uno de los dos métodos en grado de localizar la posición de la tubería buscada, precisamente porque donde falla uno no necesariamente falla el otro. Las características de las dos técnicas, por tanto, son diferentes: la primera es mucho mas sensible a los parámetros de los cuales depende y es por tanto necesario un fijado de estos muy preciso, la segunda, en cambio, aun cuando es muy robusta, necesita que los sectores de barrido de los dos sensores sonar se sobrepongan y es sensible a las situaciones, como aquella mostrada en la Figura 4.2.11, donde la tubería se encuentra exactamente bajo uno de los dos sensores sonar.

# CONCLUSIONES Y RECOMENDACIONES

1. El objetivo perseguido en el curso de esta tesis, de potenciar los sistemas sonar de barrido mecanico empleados en sistemas VOR, ha sido plenamente alcanzado gracias al buen nivel de los resultados obtenidos, trabajando con datos reales adquiridos durante algunos experimentos en el mar y probando las tecnicas de mejoramiento de imágenes y deteccion de objetos en dichos datos.

2. En lo que respecta a los sistemas sonar de tipo *haz de abanico*, se ha procedido al mejoramiento de la calidad visual de las imágenes acústicas a partir de las señales de "bajo nivel", o sea de las señales adquiridas por el transductor sonar usado en recepción.

2.1 Para estos sistemas, el mejoramiento de la calidad visual de las imágenes ha sido posible gracias al desarrollo de una serie de elaboraciones de los datos adquiridos por el sistema sonar, entre las que se encuentran una ley de asignación dinámica del valor de intensidad de los píxeles, a través del análisis de los ecos acústicos utilizados para crear la imagen, que ha permitido el

aprovechamiento optimo del rango de intensidad disponible. Una segunda elaboración fue desarrollada a partir de los datos acústicos optimizados, resultantes del uso de la ley de asignación dinámica, y ésta ha sido una técnica de interpolación de los datos en el específico sistema de coordenadas polares del sonar. La elección de trabajar en el sistema de coordenadas polares, nativo del sonar, ha permitido la puesta a punto de una interpolación mas fina de la que normalmente se obtiene trabajando en el sistema de coordenadas cartesianas e involucrando directamente los píxeles de la imagen, como normalmente viene hecho en la mayor parte de los sistemas de *imaging* acústico.

2.2 Los resultados obtenidos muestran en forma general una mayor homogeneidad en la imagen y, por tanto, una mejor calidad visual. Vale la pena subrayar que la elección de trabajar en coordenadas polares, hecha posible gracias a la buena frecuencia de muestreo de los datos y a la pequeñez del paso mecánico, ha permitido reducir la carga computacional relacionada a las técnicas de interpolación standard, operando en el dominio cartesiano. En este caso, de hecho, una simple interpolación basada en la media pesada comporta el cálculo de las distancias desde el píxel al que hay que asignar el valor hasta sus vecinos, distancias que para ser calculadas requieren el uso de raíces cuadradas, indudablemente pesadas desde el punto de vista computacional.

2.3 Técnicas posteriores de elaboración de los datos, operando sobre la imagen ya formada, fueron puestas a punto con el fin de contrastar la presencia del ruido speckle y de mejorar el contraste. En el primer caso se ha recurrido al uso de filtros adaptivos anti-speckle, específicamente el filtro de Frost, que se ha revelado como el mejor compromiso entre complejidad y prestaciones.

2.4 Con el fin de mejorar el contraste se aplicó una técnica avanzada, recientemente propuesta para aplicaciones ópticas (2). Esta se basa en la así llamada "*distribución de pares de intensidad*" y permite aprovechar tanto las ventajas de las aproximaciones globales (ecualización del histograma) como las ventajas comunes de las aproximaciones locales (ecualización del histograma adaptivo).

2.4 Algunas de estas técnicas, específicamente el filtrado adaptivo anti-speckle y el mejoramiento del contraste, involucran algunos parámetros de cuya regulación depende la calidad de los resultados obtenidos. Vale la pena subrayar como la elección de tales parámetros ha sido razonada con el fin de obtener en promedio buenos resultados en la practica y se ha revelado robusta con respecto a la amplia gama de los datos adquiridos y elaborados.

3. En lo que se refiere a los sistemas de tipo *haz de lapiz*, usados en modalidad *sensor dual*, se procedió a desarrollar métodos dirigidos a la localización automática de objetos presentes en la escena, a partir de datos obtenidos relativos al perfil, o sea a partir de la información relativa a la distancia y a la fuerza reflectante de un objeto puesto en la dirección hacia la cual está apuntado el haz.

3.1 Para tal fin se desarrollaron dos versiones de una técnica de detección de objetos, ambas basadas en la estrategia de *plantilla correspondiente*. La primera trabaja exclusivamente en el dominio de las coordenadas cartesianas, la segunda, en cambio, aprovecha las propiedades del sistema de coordenadas

polares nativas del sistema sonar. La necesidad de idear y desarrollar dos versiones diferentes deriva de las notables dificultades encontradas debidas sustancialmente a la naturaleza dispersa de los datos a disposición.

3.2 Las dos técnicas fueron evaluadas en más de 300 imágenes de perfiles producidos por los sistemas sonar funcionando en modalidad *sensor dual*. Dada la amplia gama de datos recogidos y elaborados fue posible proveer una estima del porcentaje de correcta localización para las dos diferentes versiones. En el caso de la primera versión, el procedimiento de detecciones congruentes, el porcentual de correcta localización es de cerca del 90%. En el 6% de los casos, en cambio, se ha cometido un error colocando la tubería en una posición no correcta, mientras que en el restante 4% no se ha producido la localización.

3.3 Relativamente a la segunda versión, basada en la discontinuidad en rango, la porcentual de localizaciones correctas se ha estimado en torno al 87%, la localización de la tubería en una posición errada es, en cambio, de cerca del 5% y la no localización es de cerca del 8%.

3.4 En general, ambas versiones de la técnica de detección de objetos han provisto resultados satisfactorios, mostrando ser una alternativa valida la una con respecto a la otra. En la mayor parte de los casos examinados, al menos una de las dos versiones desarrolladas ha permitido localizar la tubería correctamente.

4. El trabajo original de la generacion de las imágenes acusticas se efectuó usando MATLAB como plataforma de desarrollo. Es un lenguaje más facil de programar y más amigable, sin embargo, recomendamos el uso del lenguaje C o C++ para desarrollar estas aplicaciones, puesto que el tiempo requerido para

producir una imagen se reduce drásticamente. En aplicaciones que requieran tiempo real, el lenguaje C, C++ ofrece velocidades de procesamiento aceptables.

5. Todas las tecnicas de procesamiento de imágenes presentadas involucran parámetros criticos que han sido configurados manualmente en base a los mejores resultados obtenidos luego de innumerables pruebas. Se recomienda un mayor analisis en la eleccion de dichos parámetros, siendo la mayor parte de éstos calculables mediante aproximaciones geometricas o estadisticas, como por ejemplo, los valores de los umbrales para el ruido en los metodos de deteccion de objetos.

6. El trabajo desarrollado demuestra la eficacia de las técnicas de procesamiento de imágenes opticas aplicadas en imágenes acústicas; así como la eficacia de las técnicas de detección de objetos basadas en la tecnica de *plantilla correspondiente*. Las nuevas técnicas propuestas e implementadas muestran, sin embargo, que en el campo de las imágenes acústicas existen muchas aproximaciones e hipótesis. La mayor parte de éstas apuntan hacia el futuro en este campo: la extracción automatica de información en una escena.

# APENDICES

# APENDICE A

# CODIGO DEL PROGRAMA DESARROLLADO EN VISUAL C++

La implementación de todas las técnicas y métodos de procesamiento de las imágenes acústicas se realizó mediante un programa creado en lenguaje C++. El programa, denominado **SeaNetLogExtract**, consta esencialmente de el cuerpo principal, denominado **SeaNetLogExtractDlg.cpp**, y un archivo donde se encuentran implementadas todas las funciones, denominado **SeaNetLogExtractFunzioni.cpp**, con su respectivo archivo de encabezados, el **SeaNetLogExtractDlg.h**, en donde se declaran las funciones.

## Programa principal

El programa principal consta de casi 900 líneas de código. En éste se crea la interfase desde la cual se cargan los archivos de tipo LOG y se realiza el procesamiento básico de los datos, como la extracción, colocación en vectores y extracción de información esencial. Posteriormente, dentro del mismo programa, se llama a cada una de las funciones que realizaran desde la creación de la matriz, pasando por el procesamiento de las muestras y finalmente la creación del archivo .bmp donde se muestra la imagen final procesada.

### *SeaNetLogExtractDlg.cpp*

```
// SeaNetLogExtractDlg.cpp : implementation file
//

#include "stdafx.h"
#include "V4M2Defs.h"
#include "v4headv3.h"
#include "SeaNetReaderDefs.h"
#include "SeaNetLogExtract.h"
#include "SeaNetLogExtractDlg.h"
#include <math.h>


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSeaNetLogExtractDlg dialog

CSeaNetLogExtractDlg::CSeaNetLogExtractDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSeaNetLogExtractDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSeaNetLogExtractDlg)
	m_sFileName = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSeaNetLogExtractDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSeaNetLogExtractDlg)
	DDX_Text(pDX, IDC_EDIT_LOGFILE, m_sFileName);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSeaNetLogExtractDlg, CDialog)
	//{{AFX_MSG_MAP(CSeaNetLogExtractDlg)
	ON_WM_SYSCOMMAND()
```

```
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BTN_BROWSE, OnBtnBrowse)
	ON_BN_CLICKED(IDC_BTN_PROCESS, OnBtnProcess)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSeaNetLogExtractDlg message handlers

BOOL CSeaNetLogExtractDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSeaNetLogExtractDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSeaNetLogExtractDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
```

```
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSeaNetLogExtractDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CSeaNetLogExtractDlg::OnBtnBrowse()
{
        // Open dialog box for browsing for source file
        CFileDialog dlgFile(TRUE,"*.bin",0,OFN_FILEMUSTEXIST |  OFN_PATHMUSTEXIST);
        // OFN_FORCESHOWHIDDEN |


        if (dlgFile.DoModal()==IDOK)
        {
                m_sFileName  = dlgFile.GetPathName();
                UpdateData(FALSE);
        }

}

void CSeaNetLogExtractDlg::OnBtnProcess()
{
        LPVOID pMsg;

        m_max = 0;
        double aperturaA, delta_thetaA,periodo_campionamentoA,maxPortataA;
        double aperturaB, delta_thetaB,periodo_campionamentoB,maxPortataB;
        //double numeroPingScanA, numeroPingScanB;
        int num_pingA,numCampioniA,numeroImmaginiA=0;
        int num_pingB,numCampioniB,numeroImmaginiB=0;
        int numeroImmagini=0;
        //numeroImmagini=0;
        int *vettoreDati,*vettoreDatiA, *vettoreDatiB;
        int tipoA[3],tipoB[3];
        double *thetaPingA, *thetaPingB, *thetaPing;

        profilatore=0;
        num_ping = 0;
        num_pingA = 0;
        num_pingB = 0;
        // Get file name (possibly changed manually)
        UpdateData();

        BYTE numSonar;
        int tst1=1, tst2=1, mat=1;
        WORD control,mask;
        DWORD dim;

        FILE *f,*g;

        if (m_sFileName.IsEmpty())                      // if string is empty return...
                return;

        if (!(f=fopen(m_sFileName,"rb")))
        {
                FormatMessage(     FORMAT_MESSAGE_FROM_SYSTEM                                   |
FORMAT_MESSAGE_ALLOCATE_BUFFER |
                                                        FORMAT_MESSAGE_IGNORE_INSERTS, // flags
```

```
                                                0,                                                      //
source, ignored
                                                GetLastError(),
                                                MAKELANGID(LANG_NEUTRAL,  SUBLANG_DEFAULT),  //
Default language
                                                (LPTSTR) &pMsg,
                                                0,
                                                NULL);
                // Display the string.
                AfxMessageBox( (LPCTSTR)pMsg, MB_OK | MB_ICONSTOP );
                // Free the buffer.
                LocalFree( pMsg );
                // exit
                return;
        }

        g=f;                                    //leggo il registry key per prendere le posizioni delle teste
        posizioneTeste(g);
        fseek(g,0,SEEK_SET);

        ThASNRec   data;
        // fileMsg    rec;
        DWORD dwRec = 0;

        LogFileHead fileHead;
        LogFileHeadRec recHead;
        LogFileSonarRec recSonar;
        LogFileProfilerRec recProfiler;
        DWORD dwLen;
        BYTE AisMaster=0;
        BYTE scanRight[2];

        // Read something....
        if (fread(&fileHead,sizeof(fileHead),1,f)!=1)
        {

                AfxMessageBox("Not enough bytes to read (1)",MB_ICONSTOP);
                fclose(f);
                return;
        }

        // try to move to start of data bytes
        if (fseek(f,fileHead.dwDataOffset,SEEK_SET))
        {
                AfxMessageBox("Cannot move to data offset",MB_ICONSTOP);
                fclose(f);
                return;
        }

        // Process data
        while((!feof(f)) && (dwRec < fileHead.dwDataRecs))
        {


                if (fread(&recHead,sizeof(recHead),1,f)!=1)
                {
                        if (feof(f))
                        {
                        // we are at the end of file
                        break;
                        }

                        AfxMessageBox("Not enough bytes to read (2)",MB_ICONSTOP);
                        BOOL bEnd = feof(f);
                        fclose(f);
                        return;
                }

                switch(recHead.nHeadType)
                {
                        case 2:   // Sonar
                                  dwLen = sizeof(recSonar);
```

```
                                        if (fread(&recSonar,dwLen,1,f)!=1) //Includes Extradetects message
                                        {
                                                AfxMessageBox("Not enough bytes to read
(3)",MB_ICONWARNING);
                                        }
                                        else

                                        break;
                                case 5: // Profiler
                                        dwLen = sizeof(recProfiler);
                                        if (fread(&recProfiler,dwLen,1,f)!=1)   // Includes Extra detects message
                                        {
                                                AfxMessageBox("Not enough bytes to read
(4)",MB_ICONWARNING);
                                        }
                                        break;
                                default: // ????
                                        AfxMessageBox("Head type not supported", MB_ICONSTOP);
                                        break;
                        }

                        // How many data bytes?
                        // if (fread(&data,rec.sonHdr.AsnHdr.dbytes,1,f)!=1)
                        // if (fread(&data,280+sizeof(_timeb)-sizeof(rec),1,f)!=1)
                        // if (fread(&data,337+sizeof(_timeb)-sizeof(rec),1,f)!=1)
                        if (fread(&data,recHead.wMsgLen-sizeof(recHead)-dwLen,1,f)!=1)   // Includes Extra detects
message
                        {

                                AfxMessageBox("Not enough bytes to read (5)",MB_ICONWARNING);
                                // fclose(f);
                                // return;
                        }
                        else
                        {
                                if(mat==1) //controllo il campo nTxNode per prendere il numero associato
                                {                       // alla testa e quindi verificare se il sonar è single/dual
                                        numSonar=recHead.nTxNode;
                                        mat=2;
                                        switch(numSonar)
                                        {
                                                case 2:
                                                        tipo[0]=1;          //0=profiling   1=immaging
                                                        break;

                                                case 20:
                                                        tipo[0]=0;          //0=profiling   1=immaging
                                                        break;

                                                case 21:
                                                        tipo[0]=0;          //0=profiling   1=immaging
                                                        break;

                                                default:
                                                        //finestra di ERRORE
                                                        CString msg;
                                                        msg.Format("Mi spiace amico..non riconosciamo la tua
testa sonar..ah ah");
                                                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
                                                        break;
                                        }
                                }


                                switch(recHead.nHeadType)
                                {
                                case 2:   //nel caso sia usato come un IMAGING

                                        //gestire due teste sonar
                                        if(recHead.nTxNode==numSonar)
                                        {
                                                if(tst1==1)
```

```
                                        {       numCampioniA = recSonar.wScanlineDataBytes;
        //dovrebbe essere dinamico
                                                delta_thetaA=(recSonar.nMotorStep)*0.9/16;
        //dovrebbe essere dinamico
                                                left_limA =(recSonar.wLeftLimit)*0.9/16;
                //dovrebbe essere dinamico
                                                right_limA =(recSonar.wRightLimit)*0.9/16;
        //dovrebbe essere dinamico
                                                aperturaA=(right_limA-left_limA);
                        //dovrebbe essere dinamico
                                                periodo_campionamentoA=     (recSonar.wADInterval);
//dovrebbe essere dinamico

                                                control=recHead.wHeadControls;
                                                mask=2;
                                                control=control&mask;       //provo       il       bit
cont(continuo)..
                                                tipoA[2]=control;           //0=non         continuo
1=continuo

                                                control=recHead.wHeadControls;
                                                mask=8;
                                                control=control&mask;       //provo il bit invert..
                                                if(control==8)      control=1;
                                                tipoA[1]=control;           //0=non         invertito
1=invertito

                                                dim=fileHead.dwDataRecs*numCampioniA;
                                                vettoreDatiA=new int [dim];

                                                thetaPingA=new          double[fileHead.dwDataRecs];
        //l'argomento è il numero di ping
                                        }
                                                // one more record
                                        //dwRec++; //coincide con il numero di ping.....Stefania

                                        num_ping= num_pingA;

                                        ProcessRecord(&recSonar,data.asn,vettoreDatiA,thetaPingA);
                // Process this packet...
                                        num_pingA++; //Stefania
                                        tst1=2;

                                }
                                else
                                {
                                        if(tst2==1)
                                        {       numCampioniB = recSonar.wScanlineDataBytes;
        //dovrebbe essere dinamico
                                                delta_thetaB=(recSonar.nMotorStep)*0.9/16;
        //dovrebbe essere dinamico
                                                left_limB =(recSonar.wLeftLimit)*0.9/16;
                //dovrebbe essere dinamico
                                                right_limB =(recSonar.wRightLimit)*0.9/16;
        //dovrebbe essere dinamico
                                                aperturaB=(right_limB-left_limB);
                        //dovrebbe essere dinamico
                                                periodo_campionamentoB  =  (recSonar.wADInterval);
//dovrebbe essere dinamico

                                                control=recHead.wHeadControls;
                                                mask=2;
                                                control=control&mask;       //provo       il       bit
cont(continuo)..
                                                tipoB[2]=control;           //0=non         continuo
1=continuo

                                                control=recHead.wHeadControls;
                                                mask=8;
                                                control=control&mask;       //provo il bit invert..
                                                if(control==8)      control=1;
```

```
                                        tipoB[1]=control;               //0=non             invertito
1=invertito

                                        dim=fileHead.dwDataRecs*numCampioniB;
                                        vettoreDatiB=new int [dim];
                                        for(int u=0;u<dim;u++)      vettoreDatiA[u]=0;

                                        thetaPingB=new              double[fileHead.dwDataRecs];
        //l'argomento è il numero di ping
                                }
                                // one more record
                                num_ping= num_pingB;


        ProcessRecord(&recSonar,data.asn,vettoreDatiB,thetaPingB);
        // Process this packet...
                                num_pingB++; //Stefania
                                tst2=2;
                        }

                break;

                case 5:
                        profilatore=1;

                        if(recHead.nTxNode==numSonar)                   //gestire due teste sonar
                        {
                                if(tst1==1)
                                {
                                        delta_thetaA=(recProfiler.nMotorStep)*0.9/16;

                                        left_limA =(recProfiler.wLeftLimit)*0.9/16;

                                        right_limA =(recProfiler.wRightLimit)*0.9/16;

                                        aperturaA=(right_limA-left_limA);

                                        numeroPingScanA=recProfiler.nPings;
                                        maxPortataA=recHead.wRangeScale*0.1;


                                        tipoA[2]=0;                     //0=non             continuo
1=continuo

                                        control=recHead.wHeadControls;
                                        mask=8;
                                        control=control&mask;           //provo il bit invert..
                                        if(control==8)      control=1;
                                        tipoA[1]=control;               //0=non             invertito
1=invertito

                                        control=recHead.wHeadControls;
                                        mask=4;
                                        control=control&mask;           //provo il bit scanright.
                                        if(control==4)
                                                scanRight[0]=1;
                                        else
                                                scanRight[0]=0;

                                        if(recHead.nTxNode==20)     //se   il   primo   pacchetto
corrisponde al master
                                                AisMaster=1;
                                        dim=fileHead.dwDataRecs*numeroPingScanA;     //*2
                                        xchè ogni ping è di 2 byte

                                        vettoreDatiA=new int [dim];
                                        thetaPingA=new              double[numeroPingScanA];
        //l'argomento è il numero di ping
                                }

                                numeroImmagini= numeroImmaginiA;
```

```
ProcessRecord(&recProfiler,data.asn,vettoreDatiA,numeroImmagini);              // Process this packet...
                                        numeroImmaginiA++;
                                        tst1=2;

                                }
                                else
                                {
                                        if(tst2==1)
                                        {       delta_thetaB=(recProfiler.nMotorStep)*0.9/16;

                                                left_limB =(recProfiler.wLeftLimit)*0.9/16;

                                                right_limB =(recProfiler.wRightLimit)*0.9/16;

                                                aperturaB=(right_limB-left_limB);

                                                numeroPingScanB=recProfiler.nPings;
                                                maxPortataB=recHead.wRangeScale*0.1;

                                                tipoB[2]=0;          //0=non continuo  1=continuo

                                                control=recHead.wHeadControls;
                                                mask=8;
                                                control=control&mask;         //provo il bit invert..
                                                if(control==8)       control=1;
                                                tipoB[1]=control; //0=non invertito  1=invertito

                                                control=recHead.wHeadControls;
                                                mask=4;
                                                control=control&mask;         //provo il bit scanright.
                                                if(control==4)
                                                        scanRight[1]=1;
                                                else
                                                        scanRight[1]=0;

                                                dim=fileHead.dwDataRecs*numeroPingScanB;

                                                vettoreDatiB=new int [dim];
                                                thetaPingB=new double[numeroPingScanB];
                                        //l'argomento è il numero di ping
                                        }
                                        // one more record

                                        numeroImmagini= numeroImmaginiB;

                                        ProcessRecord(&recProfiler,data.asn,vettoreDatiB,numeroImmagin
                                        i);          // Process this packet...
                                        numeroImmaginiB++;
                                        tst2=2;
                                }

                        break;
                        }
                        dwRec++;
                }

        }

        CString msg;

        msg.Format("Success! %u out of %u records decoded",dwRec, fileHead.dwDataRecs);
        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);


////////////////////////////////NUOVO/////////////////////////////////////////////////////


//////////////////////////INSERISCO PROGRAMMA/////////////////////////////////////////

        pi= 3.1416;
        CString filename = m_sFileName;
```

```
CString indice;
CString time;


if(profilatore==0)
{
	num_ping= num_pingA;
	periodo_campionamento= periodo_campionamentoA;
	numCampioni= numCampioniA;
	apertura= aperturaA;
	delta_theta= delta_thetaA;
	thetaPing= thetaPingA;
	vettoreDati= vettoreDatiA;
	tipo[1]= tipoA[1];
	tipo[2]= tipoA[2];
	left_lim= left_limA;
	right_lim= right_limA;

	int vid=0;

	do{
		clock_t start= clock();

		numPingSettore=(apertura/delta_theta)+1;
		double pingCorrente = thetaPing[0];	// metto il primo ping
		int in = 1;
		while(thetaPing[in]!=pingCorrente)	//finchè non trovo 2 ping uguali
		{
		  pingCorrente = thetaPing[in];
		  in = in+1;
		}
			pingStart = in;	// è l'ultimo ping da scartare..devo infatti scartare pingStart ping
					// anche se thetaPing[pingStart] è il primo ping dell'immagine
			int numero_immagini= (num_ping-pingStart)/numPingSettore;
			// arrotonda all'intero INFERIORE + vicino...
			//divide il numero di ping totale per quelli contenuti in una immagine = num di immagini che si ottengono

		int **matriceCampImm;
		int **immagineFinale;
		int a=0;
		int n;


	//for (int u=12; u<13; u++)//numero_immagini;u++)//richiamo la funzione...
	for (int u=0; u<numero_immagini;u++)		// richiamo la funzione...
		{
			matriceCampImm=matriceCampioni(u,vettoreDati);

			sogliatura (matriceCampImm);

			double **vettori;
			int	t=0,i,j;

			vettori=posizioneMetri (thetaPing,pingStart,u,periodo_campionamento,tipo);

			immagineFinale=scan_conv(vettori,matriceCampImm);

			if(vettori[3][0]>=0)		t=1;
			creazioneImmagine(immagineFinale, tipo, t);

		////*********FILTRO**********************************


			//filtroFrost(immagineFinale);


		///**************CONTRASTO*****************************
```

```
//netExpansionForce(immagineFinale);

///**************TORNO LO SFONDO A 255**********************


for(i=0; i<numCampioni; i++)
	for(j=0; j<dimensioneAscissa; j++)
		if(immagineFinale[i][j]>255)
	immagineFinale[i][j]=255;


///***FUNZIONE per CREARE un BITMAP a partire di una MATRICE di PIXEL

if(u<9)
	filename = "immagine 0";
else
	filename = "immagine ";

indice.Format("%d",u+1);
filename += indice;

////////VIDAL/////////////////due teste sonar

if(vid==1)
	filename += "b";

////////VIDAL/////////////////

filename += ".bmp";

SalvaBitmap(filename, immagineFinale, profilatore);

///************FINAL**********************************

free (vettori[0]);
free (vettori[1]);
free (vettori[2]);
free (vettori[3]);
free (vettori);

for(i=0; i<numCampioni; i++)
free (matriceCampImm[i]);// dealloca le righe
free(matriceCampImm); // dealloca il vettore di puntatori alle righe

for(i=0; i<numCampioni; i++)
	free (immagineFinale[i]);// dealloca le righe
	free (immagineFinale);

a++;

}

delete [] vettoreDati;
delete [] thetaPing;

clock_t ends= clock();

double tempo= (ends-start)/CLOCKS_PER_SEC;

msg.Format("Success! %d images decoded in %g sec",a, tempo);
AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

if(tst2>1)			//per il caso di due teste sonar
{
	apertura= aperturaB;
	delta_theta= delta_thetaB;
	thetaPing= thetaPingB;
	num_ping= num_pingB;
	vettoreDati= vettoreDatiB;
	periodo_campionamento= periodo_campionamentoB;
	tipo[1]= tipoB[1];
	tipo[2]= tipoB[2];
```

```
                                        left_lim= left_limB;
                                        right_lim= right_limB;
                                        numCampioni= numCampioniB;
                                }

                        vid++;

                        }while(vid<tst2); //se si ha trovato un'altra testa

                }
                else
                {
                        //assumiamo sempre 2 teste per questo caso
/*
                        msg.Format("Status=%x",recHead.nStatus);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        control=recHead.wHeadControls;
                        control=control&2;
                        msg.Format("controll= %d",control);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);


                        control=recHead.wHeadControls;
                        control=control&4;
                        msg.Format("controlll= %d",control);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        control=recHead.wHeadControls;
                        control=control&4096;
                        msg.Format("fra= %d",control);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);


                        msg.Format("AisMaster= %d",AisMaster);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);


                        msg.Format("SweepCode=%d",recHead.nSweepCode);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        msg.Format("rotMaster= %g",rotMaster);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        msg.Format("rotSlave= %g",rotSlave);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        msg.Format("Xmaster=%d Ymaster=%d",Xmaster,Ymaster);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        msg.Format("Xslave= %d Yslave=%d",Xslave,Yslave);
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/

                        int numero_immagini;

                        //il vettore thetaPing e riempito da LeftLimit a RightLimit per tutte le immagini...dunque bisogna
                        girarlo
                        //ogni immagine...
                        for (int e=0;e<numeroPingScanA;e++)
                        {
                                if(aperturaA>0)
                                        thetaPingA[e]=(left_limA+(e*delta_thetaA));//     riempiamo     i     vettori
                                        thetaPingA e thetaPingB                 numeroPingScan
                                else
                                        thetaPingA[e]=(left_limA-(e*delta_thetaA));
                        }

                        for (e=0;e<numeroPingScanB;e++)
                        {
                                if(aperturaB>0)
                                        thetaPingB[e]=(left_limB+(e*delta_thetaB));
                                else
```

```
                                thetaPingB[e]=(left_limB-(e*delta_thetaB));
                        }


                        /////********bisogna guardare bene questo*****************
                        tipo[1]= tipoA[1];
                        //tipo[2]= tipoA[2];
                        left_lim= left_limA;
                        right_lim= right_limA;
                        //numeroPingScan= numeroPingScanA;
                        //thetaPing= thetaPingA;


                        if(maxPortataA>=maxPortataB)          //scelgo la portata più grande
                                maxPortata=maxPortataA;
                        else        maxPortata=maxPortataB;


                        if(numeroImmaginiA<=numeroImmaginiB)         //scelgo il minore numero de immagini
                                numero_immagini= numeroImmaginiA;
                        else        numero_immagini= numeroImmaginiB;



                        msg.Format("fine...");
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                        clock_t start= clock();

                        double *vettoreCampImm;
                        double *thetaRuotata;
                        int **immagineFinale;
                        int a=0;
                        int n;

                        FILE *p;

                        p=fopen("C:\\tulum.txt","w");

                        diametroTubo=0.543;          //diametro del tubo nel profilo

                        int temp;
                        if(numero_immagini>1000)
                                temp=1000;
                        else
                                temp=numero_immagini;

                        for (int u=0; u<temp;u++)
                        //for (int u=1; u<2; u++)
                        {
                                int t=0;

                                vettoreCampImm=vettoreCampioni(u,vettoreDatiA,vettoreDatiB);
                                //è un vettore che contiene i rho di una immagine per una testa ed in //seguito i rho
                                della immagene della seconda testa
/*
                                msg.Format("vettoreCampioni bene!");
                                AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
        */
                                thetaRuotata = thetaProfiler(u,thetaPingA,thetaPingB, AisMaster, scanRight);
                                //è un vettore che contiene il valore di theta per ogni testa, uno a seguito dell'altra


                                for(int i=0; i<numeroPingScanA+numeroPingScanB; i++)
                                        fprintf(p,"\nvettore %d= %g",i,vettoreCampImm[i]);
                                fprintf(p,"\n\n\n");



                                double
                   *posizioneTubo=objectDetector(thetaRuotata,vettoreCampImm,AisMaster,diametroTubo);
```

```
/*
                        msg.Format("objectDetector bene!");
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/
        immagineFinale=scan_convProfiler(thetaRuotata,vettoreCampImm,AisMaster,posizioneTubo;

        ///***FUNZIONE per CREARE un BITMAP a partire di una MATRICE di PIXEL

                        creazioneImmagine(immagineFinale, tipo, t);

                        if(u<9)
                                filename = "immagine 0";
                        else
                                filename = "immagine ";

                        indice.Format("%d",u+1);
                        filename += indice;

                        filename += ".bmp";

                        SalvaBitmap(filename, immagineFinale, profilatore);

        ///***FUNZIONE per CREARE un BITMAP a partire di una MATRICE di PIXEL

                        free (thetaRuotata);
                        free(vettoreCampImm);

                        for(i=0; i<dimensioneOrdinate; i++)
                        free (immagineFinale[i]);// dealloca le righe
                        free (immagineFinale);

                        a++;
                }

                delete [] vettoreDatiA;
                delete [] thetaPingA;
                delete [] vettoreDatiB;
                delete [] thetaPingB;

                clock_t ends= clock();

                double tempo= (ends-start)/CLOCKS_PER_SEC;

                msg.Format("Success! %d images decoded in %g sec",a, tempo);
                AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

                fclose(p);

        }

}
```

## FUNCIONES

Todos los metodos y tecnicas propuestas para el procesamiento de las imágenes acusticas fueron implementados en un total de 20 funciones, que ocupan aproximadamente 3000 lineas de codigo. Las funciones se encuentran dentro del archivo ***SeaNetLogExtractFunzioni.cpp*** y las declaraciones de estas en el respectivo archivo .h.

### SeaNetLogExtractFunzioni.cpp

```
#include "stdafx.h"
#include "V4M2Defs.h"
#include "v4headv3.h"
#include "SeaNetReaderDefs.h"
#include "SeaNetLogExtract.h"
#include "SeaNetLogExtractDlg.h"
#include <math.h>


///************************FUNZIONI*********************************


///*************************PROCESS RECORD*****************************

BOOL CSeaNetLogExtractDlg::ProcessRecord(LogFileSonarRec *rec, BYTE *pSonarData,int *vettoreDati,double
*thetaPing)
{
	BYTE data[1500];

	double bearing_deg = (rec->wXdrBearing)*0.9/16; //Stefania...angolo del ping in gradi
	int SonarDataLen = rec->wScanlineDataBytes;		// Sonar data is in pSonarData


	memcpy(data,pSonarData,SonarDataLen);	//copia sonarDataLen byte da pSonarData nella memoria
	puntata da data

	for (int y=0;y<SonarDataLen;y++)
	{
		vettoreDati[SonarDataLen*num_ping+y]=data[y];
	}

	thetaPing[num_ping]=bearing_deg;

	return TRUE;
}

///*************************PROCESS RECORD*****************************


void CSeaNetLogExtractDlg::ProcessRecord(LogFileProfilerRec *rec, BYTE *pSonarData,int *vettoreDati, int
numeroImmagini)
{
	BYTE data[4096];
	int bin;
	int dim;

	int SonarDataLen = rec->nPings;				// Sonar data is in pSonarData
	//ci sono 2 byte e l'extra data

	memcpy(data,pSonarData,(SonarDataLen*2));	//copia sonarDataLen byte da pSonarData nella
	memoria puntata da data
```

```
	for (int y=0;y<SonarDataLen;y++)
	{
		bin=256*data[y*2+1]+data[y*2];	//raggruppo 2a2 i byte per prendere il valore giusto del
		ping
		vettoreDati[SonarDataLen*numeroImmagini+y]=bin;
	}
}


///*************************PROCESS RECORD*****************************



///**********************MATRICE CAMPIONI*******************************

int ** CSeaNetLogExtractDlg::matriceCampioni(int nimmagine, int *dati)
{

	int i,j,k,ncampioni=numCampioni;
	int **matrice;
	int pngset=numPingSettore;
	int start= pingStart+1;

	int prcolonna= start+ nimmagine*pngset; //indice della prima colonna della immagine
	int ulcolonna= prcolonna+ pngset-1;	//indice dell'ultima colonna della immagine
	int campini= (prcolonna-1)*ncampioni;
	int campfin= (ulcolonna+1)*ncampioni;

	//creazione della matrice dei campioni

	matrice = (int **) malloc(sizeof(int *)*ncampioni); // alloca il vettore di puntatori alle righe
	for(i = 0; i <ncampioni; i++)
	matrice[i] = (int *) malloc (sizeof(int)*pngset); // alloca le righe

	k=campini;

	for (i=0; i<pngset; i++)
	{	for(j=0; j<ncampioni; j++)
		{
			if(k<=campfin)
			{
				matrice[j][i]=dati[k];
				k++;
			}
			else{			//se i campioni non bastano per riempire l'ultima immagine
				matrice[0][0]=0;
				break;}
		}
	}

	return (matrice);
}

///******************MATRICE CAMPIONI*******************************



///**********************SOGLIATURA*******************************

int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

void CSeaNetLogExtractDlg::sogliatura (int **matrice)
{
	double intv, mean, gamma=1.7, numer, div;
	int i,j,num=0,M=numCampioni,N=numPingSettore,k=0;
	int *temp, num_el,HIGH,LOW;
```

```
int nValori[256];

//FILE *curva;
//curva= fopen("CurvaGamma.xls","wb");

//creo un vettore temporaneo per copiare la matrice originale
temp = (int *) malloc(sizeof(int)*M*N); // alloca dinamicamente M*N int

for (i = 0; i<M ; i++)
{       for(j=0; j<N;j++)
        {
                temp[k]=matrice[i][j];
                k++;
        }
}

qsort(temp, M*N, sizeof(int), compare);//sort dei valori della matrice in ordine ascendente

num_el = 0.01*M*N;

for (i = M*N-num_el; i<M*N ; i++)      //cerco la media degli ultimi num_el elementi
        num=num+temp[i];
mean = num/num_el;

HIGH = (int) mean;//calcolo i valori di assegnare a LOW e HIGH
LOW = temp[0];

free(temp); // libera la memoria precedentemente allocata

intv=HIGH-LOW;

//faccio la inserzione dei valori sogliati
for(i=0; i<256; i++)
{
        div=(double)(1/intv);
        numer= ((i-LOW)*div);
        numer= pow(numer,gamma);
        numer= (int)(numer*255);
        nValori[i]= numer;

}



for (i = 0; i<M ; i++)
{
        for (j=0; j<N; j++)
        {
                if ((matrice[i][j]<=LOW))
                matrice[i][j]=0;
                else    if ((matrice[i][j]>=HIGH))
                        matrice[i][j]= 255;
                                else    if ((LOW <matrice[i][j])&&(matrice[i][j]<HIGH))
                                                matrice[i][j] = nValori[matrice[i][j]];

        }
}

/*
for(i=0; i<256; i++)
{
        if (i<=LOW)
                fprintf(curva,"0\n");
        else    if (i>=HIGH)
                                fprintf(curva,"255\n");
                        else
                                fprintf(curva,"%d\n",nValori[i]);
}
fclose(curva);
*/
```

```
}

///**********************SOGLIATURA*****************************



///*************************POSIZIONE METRI*********************************

double ** CSeaNetLogExtractDlg::posizioneMetri (double *ping, int start, int nimmagine, double Ts,int *tipo)
{
	int ncampioni= numCampioni;
	int pngset=numPingSettore;
	int i,j,temp2, apertura;
	double theta_max,theta_min,x_neg,x_pos,delta_x,delta_z,temp,delta_rho = Ts*640*pow(10,-
9)*(1477.5/2);
	int leng_x, pngini;
	double *rho, *theta, *zeta, *ascissa,**vettori;
	double rho_max = (ncampioni-1)*delta_rho;  //portata degli echi ricevuti


	deltaRho=delta_rho;
	apertura = abs(right_lim-left_lim);

	rho = (double *) malloc(sizeof(double)*(ncampioni)); //***creazione del vettore dei valori di rho in metri
	for(i=0;i<ncampioni;i++)
		rho[i]=(i+1)*delta_rho;

	pngini = start+nimmagine*pngset; //***creazione del vettore dei valori di theta (ping) in gradi***///

	theta = (double *) malloc(sizeof(double)*pngset);

	if(tipo[2]==0)			//per sonar NON CONTINUOUS
	{
		if (tipo[0]==1)		//nel caso in cui il sonar sia tipo IMAGING
		{
			if (tipo[1]==1)		//modo INVERTED
			{
				if (left_lim>=90)	//settore verso l'alto
				{
					for(i=0; i<pngset; i++)
					theta[i] = 180-ping[pngini+i];
				}
				else				//settore verso il basso
				{
					for(i=0; i<pngset; i++)
					theta[i] = -ping[pngini+i];
				}
			}
			else				//modo NON INVERTED
			{
				if (left_lim>=90)	//settore verso il basso
				{
					for(i=0; i<pngset; i++)
					theta[i] = 180-ping[pngini+i];
				}
				else				//settore verso l'alto
				{
					for(i=0; i<pngset; i++)
					theta[i] = ping[pngini+i];
				}
			}
		}
		else				//sonar tipo PROFILING
		{
			//modo INVERTED & NON INVERTED
			if (left_lim>=90)	//settore verso il basso
			{
				for(i=0; i<pngset; i++)
				theta[i] = 180-ping[pngini+i];
			}
			else				//settore verso l'alto
```

```
                {
                        for(i=0; i<pngset; i++)
                        theta[i] = ping[pngini+i];
                }
        }

        for(i=0;i<pngset;i++)                   //cambio degli angoli a  valori negativi
        {
                if((360>theta[i])&&(theta[i]>=270))
                        theta[i]=theta[i]-360;
                if((theta[i]>-360)&&(theta[i]<=-270))
                        theta[i]=360+theta[i];
        }
}

if(theta[0]<0)
{       theta_min=theta[0];                             //valori max e min di theta
        theta_max=theta[pngset-1];
}else
        {       theta_max=theta[0];                             //valori max e min di theta
                theta_min=theta[pngset-1];
        }

zeta = (double *) malloc(sizeof(double)*(ncampioni));  //***creazione del vettore dei valori di z

for(i=0;i<ncampioni;i++)                //copio rho in z
                zeta[i] = rho[i];

delta_z=delta_rho;

//creazione del vettore dei valori di x

x_neg = rho_max*sin(theta[0]*3.1416/180);      //***creazione del vettore dei valori di x
x_pos =  rho_max*sin(theta[pngset-1]*3.1416/180);

delta_x = delta_z;  //PIXEL QUADRATI

leng_x = (abs(x_pos/delta_x)+abs(x_neg/delta_x)+1);

ascissa = (double *) malloc(sizeof(double)*leng_x);

temp2 = abs(x_neg/delta_x);

j=temp2;

for(i=0;i<=temp2;i++)                   //definisco x a partire del valore centrale
{
        if(theta[0]<0)
                ascissa[i] = -delta_x*j;
        else
                ascissa[i] = delta_x*j;
                j--;
}

temp2 = abs(x_pos/delta_x);

j=1;
do
{
        if(theta[0]<0)
                ascissa[i]= delta_x*j;
        else
                ascissa[i]= -delta_x*j;
        i++;
        j++;
}while(j<=temp2);


vettori = (double **) malloc(sizeof(double)*4);   //Creazione della matrice con i 4 vettori

vettori[0] = ascissa;
```

```
		vettori[1] = zeta;
		vettori[2] = rho;
		vettori[3] = theta;

		dimensioneAscissa=leng_x;
		thetaMax=theta_max;
		thetaMin=theta_min;
		deltaZ=delta_z;

		return (vettori);
}

///*************************POSIZIONE METRI*********************************



////////*************************TROVA INDICI*********************************

int* CSeaNetLogExtractDlg::trovaIndici(double rho_cap,double theta_cap,double **vettori)
{
		int lenght_z=numCampioni, length_theta=numPingSettore;
		int *vettoreInd;
		int count_z=0;
		int count_theta=0;
		int *pind_z,*pind_theta;
		int i, j;

		vettoreInd = (int *) malloc(sizeof(int)*4);
		pind_z = (int *) malloc(sizeof(int)*2);
		pind_theta = (int *) malloc(sizeof(int)*2);


		for(i=0;i<lenght_z;i++)
		{
			//if(((rho_cap-deltaZ)<=vettori[1][i]) && (vettori[1][i]<(rho_cap+deltaZ)))
			if(((vettori[1][i]-(rho_cap-deltaZ))>0.0000000001) && (((rho_cap+deltaZ)-
vettori[1][i])>=0.0000000001))
			{
				pind_z[count_z]=i; //ho levato il +1 per passare la funzione a C e non a matlab
				count_z++;
			}
		}

		if(count_z==1)
		{
			if (pind_z[0]==1)   pind_z[1] = 2;       // se l'unico indice è il primo z
			else
			{
				pind_z[1]=pind_z[0];
				if(pind_z[0]!=0)     pind_z[0]=pind_z[0]-1;
				else                                    pind_z[0]=pind_z[0]+1;
			}
		}

		for(j=0;j<length_theta;j++)
		{
			//if(((theta_cap-delta_theta)<vettori[3][j])&&(vettori[3][j]<(theta_cap+delta_theta)))
			if(((vettori[3][j]-(theta_cap-delta_theta))>=0.0000000001)&&(((theta_cap+delta_theta)-
			vettori[3][j])>0.0000000001))
			{
		//fprintf(prova2,"theta[j]=%g\ntheta_cap=%g\ndelta_theta=%g\nj=%d\n",vettori[3][j],theta_cap,delta_
		theta,j);

				pind_theta[count_theta]=j;
	count_theta++;
			}
		}

		//fprintf(prova2,"\ncount_z=%d\ncount_theta=%d\nj=%d\n",count_z,count_theta,j);

		if(count_theta==1)
```

```
	{
		if (pind_theta[0]==1)		pind_theta[1] = 2;  // se l'unico indice è il primo z
		else
		{
			pind_theta[1]=pind_theta[0];
			if(pind_theta[0]!=0)		pind_theta[0]=pind_theta[0]-1;
			else 					pind_theta[0]=pind_theta[0]+1;
		}
	}

	vettoreInd[0]=pind_z[0];
	vettoreInd[1]=pind_z[1];
	vettoreInd[2]=pind_theta[0];
	vettoreInd[3]=pind_theta[1];

	count_theta=0;
	count_z=0;

	free (pind_z);
	free (pind_theta);

	return (vettoreInd);
}


////////*************************TROVA INDICI*********************************



/////*****************************SCAN CONV*************************************


int ** CSeaNetLogExtractDlg::scan_conv(double **vettori,int **matriceCampImm)
{
	double rho_cap,theta_cap,rho_1,rho_2,theta_1,theta_2,theta_sum,amp_rho1,amp_rho2,x_str1,x_str2;
	double z_str1, z_str2;
	int length_z=numCampioni,amp11,amp12,amp21,amp22,AMP;
	int **matrice;
	int *ll;
	int *vettoreIndici;



	matrice = (int **) malloc(sizeof(int *)*length_z); // alloca il vettore di puntatori alle righe
	for(int i = 0; i <length_z; i++)
		matrice[i] = (int *) malloc (sizeof(int)*dimensioneAscissa);

	for( i=0; i<length_z; i++)
	{
		for(int j=0;j<dimensioneAscissa; j++)
					matrice[i][j]=300;
	}


	ll = (int *) malloc(sizeof(int)*dimensioneAscissa);

	if(vettori[3][0]<0)
	{
		 z_str1=(vettori[0][dimensioneAscissa-1]/tan(thetaMax*pi/180))/deltaZ;
		 z_str2=(vettori[0][0]/tan(thetaMin*pi/180))/deltaZ;
	}
	else
	{
		 z_str1=(vettori[0][0]/tan(thetaMax*pi/180))/deltaZ;
		 z_str2=(vettori[0][dimensioneAscissa-1]/tan(thetaMin*pi/180))/deltaZ;
	}

	int z_str=z_str1;

	if(z_str2<z_str1)    z_str=z_str2;
```

```
for (int j=0;j<z_str;j++)
{
	int s=0;
	x_str1=(vettori[1][j]*tan(thetaMin*pi/180));
	x_str2=(vettori[1][j]*tan(thetaMax*pi/180));

	for(int h=0;h<dimensioneAscissa;h++)// ho bisogno della dimensione massima di X
	{
		if((vettori[0][h]>=x_str1) && (vettori[0][h]<=x_str2))
		{
			ll[s]=h;   // ritorna gli indici del vettore di x i cui elementi verificano quella
			condizione
			s++;

		}
	}


	for (int i=ll[0];i<=ll[s-1];i++)
	{

			rho_cap=sqrt(pow(vettori[0][i],2)+pow(vettori[1][j],2));
			theta_cap=atan(vettori[0][i]/vettori[1][j]);
			theta_cap=theta_cap*180/pi;

			vettoreIndici=trovaIndici(rho_cap,theta_cap,vettori);


			amp11 = matriceCampImm[vettoreIndici[0]][vettoreIndici[2]];
			amp12 = matriceCampImm[vettoreIndici[0]][vettoreIndici[3]];
			amp22 = matriceCampImm[vettoreIndici[1]][vettoreIndici[3]];
			 amp21 = matriceCampImm[vettoreIndici[1]][vettoreIndici[2]];


			rho_1=(vettori[2][vettoreIndici[0]]-rho_cap);  // abs fa il valore assoluto..è
			la differenza tra il rho del campione e quello del rho cappello
			rho_2=(vettori[2][vettoreIndici[1]]-rho_cap);  // esistono due rho soli ma
			ognuno dei due si riferisce a due campioni..in totale sono 4!

			if (rho_1<0)		rho_1=-rho_1;
			if (rho_2<0)		rho_2=-rho_2;

		theta_1=(vettori[3][vettoreIndici[2]]-theta_cap);
		theta_2=(vettori[3][vettoreIndici[3]]-theta_cap);

			if (theta_1<0)	theta_1=-theta_1;
			if (theta_2<0)	theta_2=-theta_2;

			theta_sum = theta_1+theta_2;


			amp_rho1 = (amp11*theta_2 + amp12*theta_1)/theta_sum;   // con
			queste 3 righe pesa i 4 campioni!
			amp_rho2 = (amp21*theta_2 + amp22*theta_1)/theta_sum;


	AMP = (amp_rho1*rho_2 + amp_rho2*rho_1)/(rho_1+rho_2);
	matrice[length_z-j-1][i]=AMP;


			free(vettoreIndici);
		}

	//fclose(prova);

}

for (j=z_str;j<length_z;j++)
{
	for (i=0;i<dimensioneAscissa;i++)
	{
```

```
                rho_cap=sqrt(pow(vettori[0][i],2)+pow(vettori[1][j],2));
                theta_cap=atan(vettori[0][i]/vettori[1][j]);
                theta_cap=theta_cap/pi*180;


                if ((rho_cap<=(numCampioni)*deltaRho)&&(thetaMin<=theta_cap)&&(theta_cap<=thetaMax))
                        {
                                        vettoreIndici=trovaIndici(rho_cap,theta_cap,vettori);


                        amp11 = matriceCampImm[vettoreIndici[0]][vettoreIndici[2]];
                        amp12 = matriceCampImm[vettoreIndici[0]][vettoreIndici[3]];
                        amp22 = matriceCampImm[vettoreIndici[1]][vettoreIndici[3]];
                        amp21 = matriceCampImm[vettoreIndici[1]][vettoreIndici[2]];


                        rho_1=(vettori[2][vettoreIndici[0]]-rho_cap);  // abs fa il valore assoluto..è la
                        differenza tra il rho del campione e quello del rho cappello
                        rho_2=(vettori[2][vettoreIndici[1]]-rho_cap);  // esistono due rho soli ma ognuno dei
                        due si riferisce a due campioni..in totale sono 4!
                                        if (rho_1<0)        rho_1=-rho_1;
                                        if (rho_2<0)        rho_2=-rho_2;

                        theta_1=(vettori[3][vettoreIndici[2]]-theta_cap);
                        theta_2=(vettori[3][vettoreIndici[3]]-theta_cap);

                                        if (theta_1<0)      theta_1=-theta_1;
                                        if (theta_2<0)      theta_2=-theta_2;

                                        theta_sum = theta_1+theta_2;

                        amp_rho1 = (amp11*theta_2 + amp12*theta_1)/theta_sum;   // con queste 3 righe
                        pesa i 4 campioni!
                        amp_rho2 = (amp21*theta_2 + amp22*theta_1)/theta_sum;

                        AMP = (amp_rho1*rho_2 + amp_rho2*rho_1)/(rho_1+rho_2);
                        matrice[length_z-j-1][i]=AMP;

                                        free(vettoreIndici);

                        }
                }
        }

        //fclose(prova);

        free (ll);
        return (matrice);

}

/////**************************SCAN CONV************************************



//****************************CREAZIONE IMMAGINE*****************************

void CSeaNetLogExtractDlg::creazioneImmagine(int **immagineFinale, int *tipo, int thetapos)
{

        double Llim= left_lim;
        int i, j, x=dimensioneAscissa,y=numCampioni;
        int *pA1, *pA2, **A;
        int temp;

        A=immagineFinale;

        //i casi quando si deve girare l'immagine (righe), dipendendo della posizione e tipo del sonar
```

```
	if((tipo[0]&&tipo[1]&&Llim<=90)||(tipo[0]&&!tipo[1]&&Llim>90)||(!tipo[0]&&tipo[1]&&Llim>90)||(!tipo[0
]&&!tipo[1]&&Llim>90))
	{
		for(j=0; j<y/2; j++)
		{
			pA1 = &A[j][0];
			pA2 = &A[y-1-j][0];

			for (i = 0; i < x; i++)
			{
				temp = *pA1;
				*pA1++ = *pA2;
				*pA2++ = temp;
			}
		}
	}

	//i casi quando l'immagine deve essere girata (colonne) in dependenza del numero di immagine


	if((!thetapos&&tipo[0])||(thetapos&&!tipo[0]))
	{
		for(i=0; i<x/2; i++)
		{
			for (j = 0; j < y; j++)
			{
				pA1 = &A[j][i];
				pA2 = &A[j][x-i-1];
				temp = *pA1;
				*pA1 = *pA2;
				*pA2 = temp;
			}
		}
	}

}

//*****************************CREAZIONE IMMAGINE*****************************



//*********************************SAVEDIBITMAP*****************************
int CSeaNetLogExtractDlg::SalvaBitmap(const char *filename, int **immagineFinale, int profilatore)
{
	int		Ascissa1, dimensione, Ascissa2;
	int		i,j,k;
	int *dati;
	BYTE *bits;
	BITMAPINFO		info;
	FILE *fp;		/* Open file pointer */
	int size;		/* Size of file */
	int sizeheader;
	 int infosize;	/* Size of bitmap info */
	int bitsize;	/* Size of bitmap pixels */
	BITMAPFILEHEADER header;		/* File header */
	RGBQUAD Colors[256];


	Ascissa1= dimensioneAscissa;
	Ascissa2=Ascissa1;
	dimensione=Ascissa1*numCampioni;

	//fare i dati divisibili per 4
	while(Ascissa2%4!=0)
		Ascissa2++;


	dati = (int *)malloc(Ascissa1*numCampioni*sizeof(int));
	for(i=0;i<numCampioni;i++)
		for(j=0;j<Ascissa1;j++)
			dati[i*Ascissa1+j]=immagineFinale[numCampioni-i-1][j];
```

```
	bits=(BYTE *)malloc(Ascissa2*numCampioni);

	j=0;
	for(i=0;i<Ascissa1*numCampioni;i++)
	{
		if(i<Ascissa1||i%Ascissa1!=0)
		{
				bits[j]=(BYTE)dati[i];
		}else
			{	k=0;
				while(k<Ascissa2-Ascissa1)
				{	bits[j]=0;
					j++;	k++;
				}

				bits[j]=(BYTE)dati[i];
			}
		j++;
	}

	//creazione della struttura tipo BITMAPINFO

	info.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
	info.bmiHeader.biWidth = Ascissa1;
	info.bmiHeader.biHeight = numCampioni;
	info.bmiHeader.biPlanes = 1;
	info.bmiHeader.biBitCount = 8;
	info.bmiHeader.biCompression = BI_RGB;
	info.bmiHeader.biSizeImage = Ascissa2*numCampioni;
	info.bmiHeader.biClrImportant = 256;
	info.bmiHeader.biClrUsed= 256;
	info.bmiHeader.biXPelsPerMeter = 0;
	info.bmiHeader.biYPelsPerMeter = 0;

	for(i=0; i<256;i++) {
		Colors[i].rgbBlue = (BYTE)i;
		Colors[i].rgbGreen = (BYTE)i;
		Colors[i].rgbRed = (BYTE)i;
		Colors[i].rgbReserved = 0;
	}

	switch(profilatore){
	case 0:
		break;
	case 1:
		Colors[100].rgbBlue = 255;
		Colors[100].rgbGreen = 0;
		Colors[100].rgbRed = 0;

		Colors[110].rgbBlue = 0;
		Colors[110].rgbGreen = 255;
		Colors[110].rgbRed = 0;

		Colors[101].rgbBlue = 0;
		Colors[101].rgbGreen = 0;
		Colors[101].rgbRed = 255;
		break;
	}

/* Try opening the file; use "wb" mode to write this *binary* file. */
if ((fp = fopen(filename, "wb")) == NULL)
   return (-1);

/* Figure out the bitmap size */

	bitsize = info.bmiHeader.biSizeImage;


/* Figure out the header size */

infosize = sizeof(BITMAPINFOHEADER)+256*sizeof(RGBQUAD);
```

```
            sizeheader = sizeof(BITMAPFILEHEADER);
    size = sizeheader + infosize + bitsize;

    /* Write the file header, bitmap information, and bitmap pixel data... */
    header.bfType      = 'MB'; /* Non-portable... sigh */
    header.bfSize      = size;
    header.bfReserved1 = 0;
    header.bfReserved2 = 0;
    header.bfOffBits   = sizeheader + infosize;



    if (fwrite(&header, 1, sizeheader, fp) < sizeheader)
                        {
       // Couldn't write the file header - return...
       fclose(fp);
       return (-1);
       }

    if (fwrite(&info, 1, sizeof(BITMAPINFOHEADER), fp) < sizeof(BITMAPINFOHEADER))
       {
       // Couldn't write the bitmap header - return...
       fclose(fp);
       return (-1);
                }

            if (fwrite(&Colors, 1, sizeof(Colors), fp) < sizeof(Colors))
       {
       // Couldn't write the bitmap header - return...
       fclose(fp);
       return (-1);
                }

    if (fwrite(bits,1,bitsize,fp) < bitsize)
       {
       // Couldn't write the bitmap - return...
       fclose(fp);
       return (-1);
       }

    /* OK, everything went fine - return... */
    fclose(fp);
            free(bits);
            free(dati);
            return (0);


}

//*********************************SAVEDIBITMAP*****************************



//*********************************FILTROFROST*****************************

void CSeaNetLogExtractDlg::filtroFrost(int **matrice)
{
            int N=2, R= numCampioni, C= dimensioneAscissa;
            int i, j, k, l, m;
            int pixels[1000];
            int **matriceTemp;
            double pesi[1000];
            double mean = 0.0;
            double sum;
            double stdDev;
            double stdDevSum;
            double x;
            double variance;
            double A,  D=1.0;
            double sumpesi;
            double tot1;
```

```
//creazione di un'immagine temporale per salvare i dati
matriceTemp = (int **) malloc(sizeof(int *)*R); // alloca il vettore di puntatori alle righe
for(i = 0; i <R; i++)
matriceTemp[i] = (int *) malloc (sizeof(int)*C); // alloca le righe

for(i=0; i<R; i++)
        for(j=0; j<C; j++)
                matriceTemp[i][j]=matrice[i][j];



for(i=N; i<R-N; i++)        //uso N per trascurare i bordi
        for(j=N; j<C-N; j++)        //uso N per trascurare i bordi
        {
                if(matriceTemp[i][j]<256)
                {
                        m=0;
                        sumpesi=0.0;
                        tot1=0.0;
                        sum=0.0;
                        stdDevSum=0.0;
                        for(k=i-N; k<=i+N; k++)
                                for(l=j-N; l<=j+N; l++)
                                {
                                        pesi[m]= sqrt(pow(k-i,2)+pow(l-j,2));  //Calcolo la
                                        distanza tra ogni pixel della maschera e il pixel centrale
                                        pixels[m]= matriceTemp[k][l];
                                        //Salvo tutti i pixel della maschera
                                        m++;
                                }

                        //**calcolo la media

                        for (int a = 0; a < m; a++)
                                sum = sum + pixels[a];
                        mean = sum / static_cast<double>(m);

                        //**calcolo la deviazione standar

                        for (a = 0; a < m; a++) {
                                x = pixels[a] - mean;
                                stdDevSum = stdDevSum + (x * x);
                                }
                        variance = stdDevSum / static_cast<double>(m-1);
                        stdDev = sqrt( variance );

                        A= D*pow((stdDev/mean),2);

                        for(k=0; k< m; k++){
                                pesi[k]= pow(2.71828, -(A*pesi[k]));   //i pesi di ogni pixel
                                sumpesi=sumpesi+pesi[k];               //la somma dei pesi
                                tot1= pesi[k]*(double)pixels[k]+tot1;  //moltiplico i pesi per ogni
                                pixel
                                }

                                matrice[i][j]= tot1/sumpesi;
                                if(matrice[i][j]>255)           matrice[i][j]=255;

                        }

                }

                for(i=0; i<R; i++)  free (matriceTemp[i]);// dealloca le righe
        free(matriceTemp); // dealloca il vettore di puntatori alle righe

}

//*********************************FILTROFROST*****************************
```

```
///*******************************NETEXPANSIONFORCE*************************

void CSeaNetLogExtractDlg::netExpansionForce(int **immagineFinale)
{

	double NetExpansionForce[256]; //devo allocarlo dinamicamente...di dim 256
	int i, j;

	for(i = 0; i<256; i++)			NetExpansionForce[i]=0.0;

		double M=4,g=0.2,k=0.8,somma=0.0,maxx=0;
		int ultimoPixel,primoPixel,soglia=15;


		for (i = 0; i<numCampioni; i++)
		{
			for (int j=0; j<dimensioneAscissa; j++)
			{
				if(immagineFinale[i][j]<256)
				{

//DEVO METTERE UN'ALTRA CONDIZIONE IN MODO DA NON PRENDERE LO SFONDO FATTO DA 1
					if ((i-1)>=0)
					// se non esco dall'immagine...confronto (i,j) con (i-1,j)
					{
						if(immagineFinale[i-1][j]!=300)
						{
						  if (immagineFinale[i][j]<immagineFinale[i-1][j])
					//capisco qual'è il pixel più piccolo e quello più grande
							{

				primoPixel=immagineFinale[i][j];
				ultimoPixel=immagineFinale[i-1][j];
							}
							else
							{
							primoPixel=immagineFinale[i-1][j];

							ultimoPixel=immagineFinale[i][j];
							}

							if(ultimoPixel-primoPixel>soglia) //considero
							un edge e non rumore
							{

							for(int y=primoPixel;y<=ultimoPixel;y++) //
								{

								NetExpansionForce[y]++;
								}
							}
							else // se è rumore
							{
					for(int y=primoPixel;y<=ultimoPixel;y++) //
								{

	NetExpansionForce[y]=NetExpansionForce[y]-g;
								}
							}
						}

						if ((j-1)>=0  && immagineFinale[i-1][j-1]!=300)  // se
						non esco dall'immagine...confronto (i,j) con (i-1,j-1)
						{
						if (immagineFinale[i][j]<immagineFinale[i-1][j-1])
					//capisco qual'è il pixel più piccolo e quello più grande
							{

							primoPixel=immagineFinale[i][j];
							ultimoPixel=immagineFinale[i-1][j-1];
							}
```

```
                                                                        else
                                                                        {
                                                                        primoPixel=immagineFinale[i-1][j-1];

                                                                        ultimoPixel=immagineFinale[i][j];
                                                                        }

                                                                        if(ultimoPixel-primoPixel>soglia) //considero
                                                                        un edge e non rumore
                                                                        {

                                                                for(int y=primoPixel;y<=ultimoPixel;y++) //
                                                                                {

                                                                                NetExpansionForce[y]++;
                                                                                }
                                                                        }
                                                                        else // se è rumore
                                                                        {
                                                                                for(int
                                                                        y=primoPixel;y<=ultimoPixel;y++) //
                                                                                {

                                                                NetExpansionForce[y]=NetExpansionForce[y]-g;
                                                                                }
                                                                        }
                                                                }                       // chiudo il (j-1)


                                                                        if((j+1)<dimensioneAscissa && immagineFinale[i-
                                                        1][j+1]!=300) //...confronto (i,j) con (i-1,j+1)
                                                                        {
                                                                                if (immagineFinale[i][j]<immagineFinale[i-
                                                                        1][j+1])
                                                                //capisco qual'è il pixel più piccolo e quello più grande
                                                                                {

                        primoPixel=immagineFinale[i][j];
                                                                                        ultimoPixel=immagineFinale[i-
                1][j+1];
                                                                        }
                                                                        else
                                                                        {
                                                                        primoPixel=immagineFinale[i-1][j+1];

                                                                        ultimoPixel=immagineFinale[i][j];
                                                                        }

                                                                        if(ultimoPixel-primoPixel>soglia) //considero
                                                                        un edge e non rumore
                                                                        {
                                                                                for(int
                                                                        y=primoPixel;y<ultimoPixel;y++) //
                                                                                {

                                                                                NetExpansionForce[y]++;
                                                                                }
                                                                        }
                                                                        else // se è rumore
                                                                        {
                                                                        for(int y=primoPixel;y<ultimoPixel;y++) //
                                                                                {

                                                                NetExpansionForce[y]=NetExpansionForce[y]-g;
                                                                                }
                                                                        }
                                                                }               //chiudo il (j+1)

                                                        }               // chiudo (i-1)

                                                        if((j-1)>=0 && immagineFinale[i][j-1]!=300)
                                                        {
```

```
                                        if (immagineFinale[i][j]<immagineFinale[i][j-1])
                                        //capisco qual'è il pixel più piccolo e quello più
                                        grande...confronto (i,j) con (i,j-1)
                                        {
                                                primoPixel=immagineFinale[i][j];
                                                ultimoPixel=immagineFinale[i][j-1];
                                        }
                                        else
                                        {
                                                primoPixel=immagineFinale[i][j-1];
                                                ultimoPixel=immagineFinale[i][j];
                                        }

                                        if(ultimoPixel-primoPixel>soglia) //considero un edge e
                                        non rumore
                                        {
                                                for(int y=primoPixel;y<=ultimoPixel;y++) //
                                                {
                                                        NetExpansionForce[y]++;
                                                }
                                        }
                                        else // se è rumore
                                        {
                                                for(int y=primoPixel;y<=ultimoPixel;y++) //
                                                {

                                        NetExpansionForce[y]=NetExpansionForce[y]-g;
                                                }
                                        }
                                }       // chiudo il (j-1)

                        } // chiudo l'if dei 300!
                }       // chiudo i due for!
        }


        for (i=0; i<256; i++)                   //nessun valore deve essere negativo
        {
                if(NetExpansionForce[i]<0)      NetExpansionForce[i]=0.0;
        }

        for(i=0; i<256; i++)                    //implemento la MAPPING FUNCTION
        {
                NetExpansionForce[i]=(double)pow(NetExpansionForce[i],(double)(1/M));
        }

        for(i=0; i<256; i++)                    //faccio l'integrale
        {
                somma=(double)(somma+NetExpansionForce[i]);
                NetExpansionForce[i]=somma;
        }

        for(i=0; i<256;i++) // trovo il max della funzione (che potrebbe non essere all'ultimo elemento)
        {
                maxx=NetExpansionForce[0];
                if(NetExpansionForce[i]>maxx)           maxx=NetExpansionForce[i];
        }

        for(i=0; i<256;i++)                     //normalizzo l'integrale...ma mantengo valori double
        nella curva!!!
        {

NetExpansionForce[i]=(double)(NetExpansionForce[i]*(255.0)/(double)maxx);
        }



        for(i=0; i<256;i++)                     //sommo all'integrale la retta bisettrice (pesandola con
        un fattore k)
        {
```

```
		NetExpansionForce[i]=(double)(NetExpansionForce[i]*(double)k)+(i*(double)(1-k));
			}

			for (i = 0; i<numCampioni; i++)
				for (int j=0; j<dimensioneAscissa; j++)
					if(immagineFinale[i][j]<256)		immagineFinale[i][j]=(int)
(NetExpansionForce[immagineFinale[i][j]]);

}
///*****************************NETEXPANSIONFORCE*************************



///*****************************THETAPROFILER*************************
double *CSeaNetLogExtractDlg::thetaProfiler(int nimmagine, double *pingA, double *pingB, BYTE AisMaster, BYTE
scanRight[])
{
		double *theta, *p1, *p2;
		double temp;
		int pngsetA, pngsetB;
		int i;
		double rotA, rotB;
		CString msg;

		////////////////////////////////////////////////////
/*		FILE *p;
		p=fopen("C:\\Sonar\\edfu.txt","w");*/
		////////////////////////////////////////////////////


		pngsetA=numeroPingScanA;
		pngsetB= numeroPingScanB;
		int pngset=pngsetA+pngsetB;

		if(AisMaster)
		{
			rotA=rotMaster;
			rotB=rotSlave;
		}
		else{
				rotA=rotSlave;
				rotB=rotMaster;
			}


		theta = (double *) malloc(sizeof(double)*pngset);

		//**************Per pingA*********************++++


		//modo INVERTED & NON INVERTED
			if (left_limA>=90)  //settore verso il basso
				for(i=0; i<pngsetA; i++)
					//theta[i] = 180-(pingA[i]+rotA);
					theta[i] = -180+(pingA[i]+rotA);

			else				//settore verso l'alto
				for(i=0; i<pngsetA; i++)
			//		theta[i] = (pingA[i]+rotA);
				theta[i] = -(pingA[i]+rotA);



		//**************Per pingB*********************++++


		//modo INVERTED & NON INVERTED
		if (left_limB>=90)  //settore verso il basso
```

```
                    for(i=0; i<pngsetB; i++)
                              //theta[pngsetA+i] = 180-(pingB[i]+rotB);
                              theta[pngsetA+i] = -180+(pingB[i]+rotB);

          else                        //settore verso l'alto
                    for(i=0; i<pngsetB; i++)
                              //theta[pngsetA+i] = (pingB[i]+rotB);
                              theta[pngsetA+i] = -(pingB[i]+rotB);


          /*        ////////////////////////////////////////////////
                    for(i=0; i<pngset; i++)
                              fprintf(p,"\nthetaRuotata1 %d=%g",i,theta[i]);
                    fprintf(p,"\n\n\n");
                    ////////////////////////////////////////////////*/

/*        for(i=0;i<pngset;i++)               //cambio degli angoli a  valori negativi
                    {
                              if(theta[i]>90)
                                        theta[i]=-360+theta[i];
                              else if(theta[i]<-90)
                                        theta[i]=-(360+theta[i]);
                    }*/

if(theta[0]>90)
          for(i=0;i<pngsetA;i++)              //cambio degli angoli a  valori negativi
                    theta[i]=-360+theta[i];
else if(theta[0]<-90)
                    for(i=0;i<pngsetA;i++)              //cambio degli angoli a  valori negativi
                              theta[i]=-(360+theta[i]);

if(theta[pngsetA]>90)
          for(i=pngsetA;i<pngset;i++)         //cambio degli angoli a  valori negativi
                    theta[i]=-360+theta[i];
else if(theta[pngsetA]<-90)
                    for(i=pngsetA;i<pngset;i++)         //cambio degli angoli a  valori negativi
                              theta[i]=-(360+theta[i]);

int pp=0;
for(i=0;i<pngset;i++)
{
          if(theta[i]>90 || theta[i]<-90)               pp=1;
          if(theta[i]>90)             theta[i]=90;
          if(theta[i]<-90)      theta[i]=-90;
}

if(pp==1)
{
          msg.Format("Il settore non è elaborato correttamente! Cmq l'immagine dovrebbe essere simile a quella
elaborata!");
          AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
}

/*        ////////////////////////////////////////////////////////
                    for(i=0; i<pngset; i++)
                              fprintf(p,"\nthetaRuotata2 %d=%g",i,theta[i]);
                    fprintf(p,"\n\n\n");
          ////////////////////////////////////////////////////////////////*/


          if((scanRight[0]==0 && nimmagine%2==0) || (scanRight[0]==1 && nimmagine%2!=0))//dipendendo
del numero di immagine devo invertire il vettore
          {
                    for(i=0; i<pngsetA/2; i++)
                    {
                              p1 = &theta[i];
                              p2 = &theta[pngsetA-1-i];
                              temp = *p1;
                              *p1 = *p2;
                              *p2 = temp;
                    }
          }
```

```
		if((scanRight[1]==0 && nimmagine%2==0) || (scanRight[1]==1 && nimmagine%2!=0))//dipendendo
del numero di immagine devo invertire il vettore
		{
			for(i=pngsetA; i<(pngsetA+(pngsetB/2)); i++)
			{
				p1 = &theta[i];
				p2 = &theta[pngset+pngsetA-1-i];
				temp = *p1;
				*p1 = *p2;
				*p2 = temp;
			}
		}

/*		//////////////////////////////////////////////////////
			for(i=0; i<pngset; i++)
					fprintf(p,"\nthetaRuotata3 %d=%g",i,theta[i]);
				fprintf(p,"\n\n\n");
		//////////////////////////////////////////////////////

		////////////////////////
		fclose(p);
		////////////////////////*/

		return (theta);
}
///*****************************THETAPROFILER*************************



///*****************************VETTORECAMPIONI*************************

double* CSeaNetLogExtractDlg::vettoreCampioni(int u,int *vettoreDatiA,int *vettoreDatiB)
{
		double *vettore;
		int dimensione= numeroPingScanA+numeroPingScanB;

		//creazione del vettore degli echi

		vettore = (double *) malloc (sizeof(double)*dimensione);

		for(int a=0;a<numeroPingScanA;a++)
			vettore[a]=(vettoreDatiA[(numeroPingScanA*u)+a]*(1477.5*0.000001/2.0));

		for(a=0;a<numeroPingScanB;a++)
		vettore[numeroPingScanA+a]=(vettoreDatiB[(numeroPingScanB*u)+a]*(1477.5*0.000001/2.0));


		return (vettore);
}

///*****************************VETTORECAMPIONI*************************



///*****************************SCANCONVPROFILER*************************
int** CSeaNetLogExtractDlg::scan_convProfiler(double* thetaRuotata,double* vettoreCampImm, BYTE AisMaster,
double* posizioneTubo)
{
		//Creo la matrice dell'immagine
		int** matrix;
		double X0,Y0,X1,Y1,X2,Y2,XA,XB,YA,YB,x,y,ic,ip;
		int ics,ips,a,b;
		double thetaMaxA,thetaMinA,thetaMaxB,thetaMinB,thetaMax,thetaMin;
		int dimensione= numeroPingScanA+numeroPingScanB;

		CString msg;
```

```
//devo trovare il angolo più piccolo e il più grande tra i due vettori di theta
if(thetaRuotata[0]<thetaRuotata[numeroPingScanA-1])
{
	thetaMaxA=thetaRuotata[numeroPingScanA-1]*pi/180;
	thetaMinA=thetaRuotata[0]*pi/180;
}
else
{
	thetaMaxA=thetaRuotata[0]*pi/180;
	thetaMinA=thetaRuotata[numeroPingScanA-1]*pi/180;
}

if(thetaRuotata[numeroPingScanA]<thetaRuotata[dimensione-1])
{
	thetaMaxB=thetaRuotata[dimensione-1]*pi/180;
	thetaMinB=thetaRuotata[numeroPingScanA]*pi/180;
}
else
{
	thetaMaxB=thetaRuotata[numeroPingScanA]*pi/180;
	thetaMinB=thetaRuotata[dimensione-1]*pi/180;
}

if(thetaMaxA>=thetaMaxB)
	thetaMax=thetaMaxA;
else
	thetaMax=thetaMaxB;

if(thetaMinA<=thetaMinB)
	thetaMin=thetaMinA;
else
	thetaMin=thetaMinB;


double TxPulse=(maxPortata+10.0)*25.0/10.0;		//in microsecondi
if (TxPulse>200)    TxPulse=200;
if (TxPulse<20)		TxPulse=20;					//non ci arriva mai!!

double risoluzioneRange=((TxPulse*pow(10,-6)*1477.5)/2.0);	//in metri
//	double risoluzioneLaterale=maxPortata*sin(pi/180); //sin(2*pi/180);		rhoMin*sin(pi/180);

double risoluzione=risoluzioneRange;

//calcoliamo la diferenza tra le due posizioni (due teste)
double Xoffset= 0.001*(Xmaster-Xslave);
double Yoffset= 0.001*(Ymaster-Yslave);

/*
msg.Format("maxPortata= %g thetaMin=%g sin(thetaMin)=%g cos(thetaMin)=%g,
sin(45)=%g",maxPortata,thetaMin,sin(thetaMin),cos(thetaMin),sin(45));
AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/

//fissa un origine
X0= ((-maxPortata)*sin(thetaMin))/risoluzione;

if(Yoffset>=0)
	Y0= (maxPortata+Yoffset)/risoluzione;
else
	Y0= (maxPortata-Yoffset)/risoluzione;


//fissa il punto della testa MASTER(X1) e SLAVE(X2)
if(Xoffset>=0)	//master + a destra
{
	X1=X0+(Xoffset/risoluzione);
	X2=X0;
}else
	{
		Xoffset=-Xoffset;
		X2=X0+(Xoffset/risoluzione);
		X1=X0;
```

```
		}

	//fissa il punto della testa MASTER(Y1) e SLAVE(Y2)
	if(Yoffset>=0)		//master + in alto
	{
		Y1=Y0-((Yoffset)/risoluzione);
		Y2=Y0;
	}else
		{
			Yoffset=-Yoffset;
			Y2=Y0-(Yoffset/risoluzione);
			Y1=Y0;
		}

	/*msg.Format("X0= %g Y0= %g X1=%g  X2=%g Y1=%g Y2=%g ",X0,Y0,X1,X2,Y1,Y2);
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/


	double totaleAscissa= ((maxPortata*sin(thetaMax)) - (maxPortata*sin(thetaMin)) + Xoffset)/risoluzione;

	double totaleOrdinata=(maxPortata+Yoffset)/risoluzione;

/*
	msg.Format("totaleAscissa=%g  totaleOrdinata%g",totaleAscissa,totaleOrdinata);
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/

	double dimPixel=1;

	do{				//adatto la lunghezza della immagine allo schermo
		dimPixel=dimPixel+0.5;
		dimensioneAscissa=totaleAscissa*dimPixel+1;
		dimensioneOrdinate=totaleOrdinata*dimPixel+1;
	}while(dimensioneAscissa<800.0&&dimensioneOrdinate<500.0);



	numCampioni=dimensioneOrdinate;

	/*
	msg.Format("dimensioneAscissa=%d  dimensioneOrdinate%d",dimensioneAscissa,dimensioneOrdinate);
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/

	//creiamo la matrice dell'immagine e la riempiamo con un valore nullo

	matrix=(int**)malloc(sizeof(int*)*(dimensioneOrdinate));  //alloco le righe
	for(int i=0;i<dimensioneOrdinate;i++)		matrix[i]=(int*) malloc(sizeof(int)*dimensioneAscissa);



	for(i=0;i<dimensioneOrdinate;i++)
	{
		for(int f=0;f<dimensioneAscissa;f++)
		{
			matrix[i][f]=0;
		}
	}


	//inserisco i punti corrispondenti alle due teste

	ics=X1*dimPixel;
	ips=Y1*dimPixel;
	matrix[ips-2][ics]=110;
	for(int j=2; j>0; j--)
		for(int k=-j-1; k<=j+1; k++)
			if(ips-j>=0&&ics-k>=0)		matrix[ips+j-2][ics+k]=110;

	ics=X2*dimPixel;
	ips=Y2*dimPixel;
	matrix[ips-2][ics]=101;
```

```
for(j=2; j>0; j--)
		for(int k=-j-1; k<=j+1; k++)
			if(ips-j>=0&&ics-k>=0)		matrix[ips+j-2][ics+k]=101;

XA=X1;
YA=Y1;
XB=X2;
YB=Y2;
int colorA=110,
int colorB=101;
if(!AisMaster)
{
		XA=X2;
		YA=Y2;
		XB=X1;
		YB=Y1;
		colorA=101,
		colorB=110;
}


int dimDot=2;			//dimensione di ogni singolo pixel della immagine nello schermo

//Mettiamo i punti della prima testa

for(i=0;i<numeroPingScanA;i++)
{
		//mett i punti nell'immagine
		x=((vettoreCampImm[i]*sin(thetaRuotata[i]*pi/180))/risoluzione);
		y=((vettoreCampImm[i]*cos(thetaRuotata[i]*pi/180))/risoluzione);

		x=XA+x;
		y=YA-y;

		ics=x*dimPixel;			//dovrei approx all'intero + vicino non al + piccolo
		ips=y*dimPixel;

		if(ics<dimensioneAscissa && ics>=0 && ips>=0)
			for(int j=0; j<dimDot; j++)
				for(int k=0; k<dimDot; k++)
					if(ips-j>=0&&ics-k>=0)		matrix[ips-j][ics-k]=colorA;
}

//mettiamo i punti della seconda testa

for(i=numeroPingScanA;i<dimensione;i++)
{
		//mett i punti nell'immagine
		x=((vettoreCampImm[i]*sin(thetaRuotata[i]*pi/180))/risoluzione);
		y=((vettoreCampImm[i]*cos(thetaRuotata[i]*pi/180))/risoluzione);

		x=XB+x;
		y=YB-y;

		ics=x*dimPixel;			//dovrei approx all'intero + vicino non al + piccolo
		ips=y*dimPixel;

		if(ics<dimensioneAscissa && ics>=0 && ips>=0)
			for(int j=0; j<dimDot; j++)
				for(int k=0; k<dimDot; k++)
					if(ips-j>=0&&ics-k>=0)		matrix[ips-j][ics-k]=colorB;
}



//disegniamo il tubo, nel caso nel cui essista
if(posizioneTubo[0]!=0 || posizioneTubo[1]!=0)
{
		double h,k;
		int Xmin,Xmax,Ymin,Ymax;
		double R2;
		double xy2;
```

```
		X0=X1-Xmaster*0.001/risoluzione;		//coordinate dell'origine!
		Y0=Y1+Ymaster*0.001/risoluzione;

		h=X0+posizioneTubo[0]/risoluzione;		//coordinate del centro del tubo
		k=Y0-(posizioneTubo[1]+diametroTubo/2)/risoluzione;

		Xmin=dimPixel*(h-(diametroTubo/2)/risoluzione);//coordinate degli stremi della circunferencia
		Ymin=dimPixel*(k-(diametroTubo/2)/risoluzione);
		Xmax=dimPixel*diametroTubo/risoluzione+Xmin;
		Ymax=dimPixel*diametroTubo/risoluzione+Ymin;

		for(a=Xmin-10; a<=Xmax+10; a++)
			for(b=Ymin-10; b<=Ymax+10; b++)
			{
				x=a*risoluzione/dimPixel;
				y=b*risoluzione/dimPixel;
				R2=pow(diametroTubo/2,2);
				xy2=pow(x-h*risoluzione,2)+pow(y-k*risoluzione,2);

				if((R2-0.01)<xy2 && xy2<=(R2+0.01))
				{
					ics=x*dimPixel/risoluzione;
					ips=y*dimPixel/risoluzione;
					if(ics<dimensioneAscissa && ics>=0 && ips>=0)
					matrix[ips][ics]=255;
				}


			}

	/*
		msg.Format("X0=%g Y0=%g x=%g y=%g risoluzione=%g",X0,Y0,x,y,risoluzione);
		AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/

	}

	free(posizioneTubo);

	return(matrix);
}
///*******************************SCANCONVPROFILER*************************



///*******************************POSIZIONETESTE*************************

void CSeaNetLogExtractDlg::posizioneTeste(FILE *f)
{
	char car;
	char vettore[1000],stringaTrovata[100],stringaDaTrovare[100];
	char position0[17],rotation0[17],position1[17],rotation1[17];
	char Xmas[9],Ymas[9],Xsl[9],Ysl[9];
	int i=0;
	int controllo=10;

	fseek (f, 92 , SEEK_SET );	//salto la prima riga che mi da dei problemi

	if(f==0)
	{
		printf("Non si è riusciti ad aprire il file! Sicuro che ci sia?");
		return;
	}

	for(int q=0;q<6;q++)
	{
		if(q==0)  strcpy(stringaDaTrovare,"CurrentApp");
		if(q==1)  strcpy(stringaDaTrovare,"Touch Down (Dual Head)\\Profiler");
		if(q==2)  strcpy(stringaDaTrovare,"Position0");
			  //Cerco la posizione!!!
		if(q==3)  strcpy(stringaDaTrovare,"Rotation0");
		if(q==4)  strcpy(stringaDaTrovare,"Position1");
			  //Cerco la posizione!!!
```

```
		if(q==5)  strcpy(stringaDaTrovare,"Rotation1");

		while(feof(f)==0 && controllo==10)		//ricerca all'interno di tutto il file!
		{
			//finchè la funzione ricerca non ha trovato quel che cercava
			fscanf(f,"%c",&car);
			while(i<1000 && car!='\n')		//carico una riga
			{
				vettore[i]=car;
				fscanf(f,"%c",&car);
				i++;
			}
			vettore[i]='\0';

			controllo=ricerca(vettore,stringaTrovata,stringaDaTrovare,q);
			i=0;
		}
		controllo=10;

		if(q==2)			//carico la position0
		{
			for(int w=0;w<16;w++)
				position0[w]=stringaTrovata[w+((w)/2)];

			position0[16]='\0';
		}

		if(q==3)			//carico la rotationn0
		{
			for(int w=0;w<16;w++)
				rotation0[w]=stringaTrovata[w+((w)/2)];

			rotation0[16]='\0';
		}

		if(q==4)			//carico la position1
		{
			for(int w=0;w<16;w++)
				position1[w]=stringaTrovata[w+((w)/2)];

			position1[16]='\0';
		}

		if(q==5)			//carico la rotation1
		{
			for(int w=0;w<16;w++)
				rotation1[w]=stringaTrovata[w+((w)/2)];

			rotation1[16]='\0';
		}
}

for(int s=0;s<8;)			//trasformo in interi le posizioni
{
		Xmas[s]=position0[7-s-1];
		Xmas[s+1]=position0[7-s];
		Xsl[s]=position1[7-s-1];
		Xsl[s+1]=position1[7-s];

		Ymas[s]=position0[15-s-1];
		Ymas[s+1]=position0[15-s];
		Ysl[s]=position1[15-s-1];
		Ysl[s+1]=position1[15-s];
		s=s+2;


		Xmas[8]='\0';
		Ymas[8]='\0';
		Xsl[8]='\0';
		Ysl[8]='\0';
}
```

```
	for(s=0;s<8;)				//giro i vettori delle rotazioni
	{
		car=rotation0[14-s];
		rotation0[14-s]=rotation0[s];
		rotation0[s]=car;
		car=rotation0[15-s];
		rotation0[15-s]=rotation0[s+1];
		rotation0[s+1]=car;

		car=rotation1[14-s];
		rotation1[14-s]=rotation1[s];
		rotation1[s]=car;
		car=rotation1[15-s];
		rotation1[15-s]=rotation1[s+1];
		rotation1[s+1]=car;
		s=s+2;
	}

	Xmaster=covertiEsadec(Xmas);
	Ymaster=covertiEsadec(Ymas);
	Xslave=covertiEsadec(Xsl);
	Yslave=covertiEsadec(Ysl);

	rotMaster=covertiEsadecInDouble(rotation0);
	rotSlave=covertiEsadecInDouble(rotation1);

}


///*****************************POSIZIONETESTE*******************************



///*****************************RICERCA**************************************

int CSeaNetLogExtractDlg::ricerca(char *vettoreDiRiga,char* stringaTrovata,char* stringaDaTrovare,int q)
{
	char carat;
	int dimDaTrovare=strlen(stringaDaTrovare);
	int dimRiga=strlen(vettoreDiRiga);
	int flag=10,h=0,controllo=10,tuttoGiusto=0;

	//adesso devo cercare dentro pezziDiString se c'è stringDaTrovare

	int i=0;

	while(i<dimRiga)		//controllo tutta la riga ma non sforo la dim del vettore
	{
		if(vettoreDiRiga[i]==stringaDaTrovare[0])	//se la prima lettera è uguale
		{

			flag=0;
	//suppongo che sia la prima lettera della stringa giusta
			for(int r=0;r<dimDaTrovare;r++)			//controllo tutte le lettere
			{
				if((i+r)<dimRiga)	//sto attento a non sforare la dim del vettore
				{
					if(vettoreDiRiga[i+r]!=stringaDaTrovare[r])	//se c'è anche 1
					solo carattere diverso
					{
						flag=10;
						tuttoGiusto++;
					}
					else	if(tuttoGiusto==0 && r==(dimDaTrovare-1))
				//se è la stringa cercata
								flag=0;

					if(tuttoGiusto!=0 && r==(dimDaTrovare-1))
				//finito il for se non è la stringa cercata
						i++;
				}
```

```
                                        else
                                        //se sforo la dim del vettore
                                        {
                                                flag=10;
                                                i++;
                                        }
                                                //era un falso allarme! Eravamo alla fine della riga!

                                }

                                if(flag==0)                     //se ho trovato la stringa che mi serve!
                                {
                                        if(q==0  || q==1)
                                        {
                                                //devo copiare in stringaTrovata le lettere successive...ma ci
                                                saranno tutte dentro al vettore?
                                                carat=vettoreDiRiga[i+dimDaTrovare+3];

                                                while(carat!='"' && h<100)
                                                {
                                                        stringaTrovata[h]=carat;
                                                        carat=vettoreDiRiga[i+dimDaTrovare+3+h+1];
                                                        h++;
                                                }
                                                stringaTrovata[h]='\0';
                                        }

                                                if(q==2  || q==3 || q==4  || q==5)
                                //prendo Position0 e Rotation0
                                        {
                                                carat=vettoreDiRiga[i+15];

                                                while(carat!='\n' && h<100)
                                                {
                                                        stringaTrovata[h]=carat;
                                                        carat=vettoreDiRiga[i+15+h+1];
                                                        h++;
                                                }
                                                stringaTrovata[h]='\0';

                                        }

                                        controllo=0;
                                        h=0;
                                        flag=10;
                                        i=dimRiga;
                                }

                        }
                        else                            //se la prima lettera non è uguale vado avanti nel vettore
                                i++;

                        tuttoGiusto=0;
                }

                return controllo;
}

///*******************************RICERCA*************************



///*******************************DACHARADINT*************************

int* CSeaNetLogExtractDlg::daCharAdInt(char vettoreEsadec[])
{
        int dim=strlen(vettoreEsadec);
        int *vetDiInteri=(int*)malloc(sizeof(int)*dim);

        for(int u=0;u<dim;u++)
        {
                switch(vettoreEsadec[u])
```

```
		{
			case '0':
				vetDiInteri[u]=0;
				break;
			case '1':
				vetDiInteri[u]=1;
				break;
			case '2':
				vetDiInteri[u]=2;
				break;
			case '3':
				vetDiInteri[u]=3;
				break;
			case '4':
				vetDiInteri[u]=4;
				break;
			case '5':
				vetDiInteri[u]=5;
				break;
			case '6':
				vetDiInteri[u]=6;
				break;
			case '7':
				vetDiInteri[u]=7;
				break;
			case '8':
				vetDiInteri[u]=8;
				break;
			case '9':
				vetDiInteri[u]=9;
				break;
			case 'A':
				vetDiInteri[u]=10;
				break;
			case 'B':
				vetDiInteri[u]=11;
				break;
			case 'C':
				vetDiInteri[u]=12;
				break;
			case 'D':
				vetDiInteri[u]=13;
				break;
			case 'E':
				vetDiInteri[u]=14;
				break;
			case 'F':
				vetDiInteri[u]=15;
				break;
			default:
				break;
		}
	}

	return vetDiInteri;
}

///*******************************DACHARADINT*************************


///*******************************CONVERTIESADEC*************************

int CSeaNetLogExtractDlg::covertiEsadec(char vettoreEsadec[])
{
	int risultato=0;
	double potenza;
	int dim=strlen(vettoreEsadec);
	int *vetDiInteri;

	vetDiInteri=daCharAdInt(vettoreEsadec);
```

```
	for(int s=0;s<dim;s++)
	{
		potenza=pow( 2.0, (4.0*(7.0-s)) );
		risultato+=vetDiInteri[s]*potenza;
	}

	free(vetDiInteri);
	return risultato;
}

///*****************************CONVERTIESADEC*************************


///*****************************CONVERTIESACINDOUBLE*************************


double CSeaNetLogExtractDlg::covertiEsadecInDouble(char vettoreEsadec[])
{
	double mantissa=0.0,potenza,esponente,risultato,frazione=0.0;
	int dim=strlen(vettoreEsadec);
	int primo,x=0,numDiQuatBitInteressati,numBitRestanti,espo,num,s;
	int *vetDiInteri;//=(int*)malloc(sizeof(int)*dim);
	int mask,maskk,a=0;
	int segno;

	vetDiInteri=daCharAdInt(vettoreEsadec);

	mask=8;                                                         //calcolo il segno
	segno=(vetDiInteri[0] & mask);

	primo=vetDiInteri[0] & 7;                       //calcolo l'esponente
	esponente= primo*pow(2, 8)+vetDiInteri[1]*pow(2, 4)+vetDiInteri[2];
	esponente=esponente-1023.0;

	numDiQuatBitInteressati=esponente/4;

	for(s=3;s<numDiQuatBitInteressati+3;s++)                        //calcolo la mantissa
	{
		potenza=pow( 2, (-4* (s-2) ) );
		mantissa+=(vetDiInteri[s]*potenza);
	}

	espo=esponente;
	numBitRestanti=espo%4;                          //prendo i bit restanti

	if(numBitRestanti==3)
	{
		mask=14;
		maskk=1;
	}
	if(numBitRestanti==2)
	{
		mask=12;
		maskk=3;
	}
	if(numBitRestanti==1)
	{
		mask=8;
		maskk=7;
	}
	num=vetDiInteri[s]&mask;

	if(numBitRestanti!=0)                   mantissa+=(num*pow( 2, (-4* (4-s) ) ));
	//aggiungo i bit restanti..
	mantissa=mantissa+1;


	if(numBitRestanti!=0)                   //Devo solo aggiungere la parte decimale!!!
	{
		num=num=vetDiInteri[s]&maskk;
		frazione=(num*pow( 2, (-4) ));
```

```
		a=1;
	}

	for(s=3+numDiQuatBitInteressati+a;s<16;s++)			//calcolo la mantissa
	{
		potenza=pow( 2, (-4* (s-2-numDiQuatBitInteressati) ) );
		frazione+=(vetDiInteri[s]*potenza);
	}

	risultato=mantissa*pow(2,esponente);
	risultato=risultato+frazione;


	free(vetDiInteri);
	return risultato;
}


///*******************************CONVERTIESACINDOUBLE*************************


///*******************************OBJECTDETECTOR*************************

double *CSeaNetLogExtractDlg::objectDetector(double* thetaRuotata,double* vettoreCampImm,BYTE AisMaster,
double Dimens)
{
	double sogliaA= Dimens/2,sogliaB=Dimens/2;  ///bisogna farlo dinamico!
	int dimensione=(numeroPingScanA+numeroPingScanB);
	double x,y;
	double *X,*Y;
	double minSoglia=maxPortata/10;		//bisogna farlo dinamico!
	double dif;
	int indice1, indice2,indice3, indice;
	double prim, sec;
	int *indiceCandidati;
	int *indiceRho;
	int i=0,j,jA=0,jB=0;
	double *posizione= (double*)malloc(sizeof(double)*2);
	CString msg;

	/*
	msg.Format("maxPortata=%g", maxPortata);
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/


	indiceRho= (int *)malloc(sizeof(int)*dimensione);//vettore degli indici!
	indiceCandidati= (int *)malloc(sizeof(int)*dimensione/2);


	X= (double *)malloc(sizeof(double)*dimensione);
	Y= (double *)malloc(sizeof(double)*dimensione);


	///*******COORDINATE CARTESIANE*************************
	//cambio a coordinate cartesiane con rispecto allo (0,0)
	i=0;
	while(i<dimensione)
	{
		x=( vettoreCampImm[i]*sin(thetaRuotata[i]*pi/180) );
		y=( vettoreCampImm[i]*cos(thetaRuotata[i]*pi/180) );

		if(i<numeroPingScanA)
		{
			if(AisMaster)
			{
				x=x+Xmaster*0.001;
				y=y+Ymaster*0.001;

			}else
				{
					x=x+Xslave*0.001;
```

```
                                        y=y+Yslave*0.001;
                                }
                        }else
                                {
                                if(!AisMaster)
                                {
                                        x=x+Xmaster*0.001;
                                        y=y+Ymaster*0.001;

                                }else
                                        {
                                                x=x+Xslave*0.001;
                                                y=y+Yslave*0.001;
                                        }
                                }

                        X[i]=x;
                        Y[i]=y;

                        i++;
                }


                ///*******ASSEGNAZIONE DI RHO CANDIDATI**************************
                //creo il vettore di indici per rho, togliendo i valori 0 e quelli sotto la soglia

                for(i=0; i<dimensione; i++)
                        indiceRho[i]=0;

                for(i=0; i<dimensione; i++)
                {
                        indice=i;
                        prim=vettoreCampImm[indice];
                        //sec=Y[indice];

                        if(prim==0.0 || prim<minSoglia/* || sec<minSoglia*/)      //se il valore non è valido metto -1
                        nel indice corrispondente
                                indiceRho[i]=-1;
                }

                i=0;
                j=0;
                while(i<dimensione)           //trovo le coppie di discontinuità
                {
                        indice=indiceRho[i];
                        if(indice!=-1)        //soltanto per i valori che non sono stati discriminati
                        {
                                if(i<numeroPingScanA)
                                {
                                        indice1=i;
                                        indice2=indice1+1;
                                        while(indice2<numeroPingScanA && indiceRho[indice2]==-1)
                                                indice2++;
                                        if(indice2!=numeroPingScanA)
                                        {
                                                prim=vettoreCampImm[indice1];
                                                sec=vettoreCampImm[indice2];
                                                dif=prim-sec;        //faccio la diferenza tra i valori vicini
                                                if(dif>=0)
                                                        dif=dif;
                                                else
                                                        dif=-dif;

                                                if(dif>=sogliaA)     //se sono più grandi della soglia sono
candidati
                                                {
                                                        indiceRho[indice1]=1;          //ogni indice 1
rappresenta un punto candidato
                                                        indiceRho[indice2]=1;
                                                        indiceCandidati[j]=indice1;
                                                        indiceCandidati[j+1]=indice2;
                                                        j=j+2;
```

```
                                        jA++;

                                /*                      msg.Format("dif=%g i=%d j=%d",dif,i,j);
                                AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/

                                        }
                                }
                                i=indice2-1;

                        }else
                                {
                                        indice1=i;
                                        indice2=indice1+1;
                                        while(indice2<dimensione && indiceRho[indice2]==-1)
                                                indice2++;
                                        if(indice2!=dimensione)
                                        {
                                                prim=vettoreCampImm[indice1];
                                                sec=vettoreCampImm[indice2];
                                                dif=prim-sec;
                                                if(dif>=0)
                                                        dif=dif;
                                                else
                                                        dif=-dif;

                                                if(dif>=sogliaB)
                                                {
                                                        indiceRho[indice1]=1;
                                                        indiceRho[indice2]=1;
                                                        indiceCandidati[j]=indice1;
                                                        indiceCandidati[j+1]=indice2;
                                                        j=j+2;
                                                        jB++;
                                                }
                                        }
                                        i=indice2-1;
                                }
                        }
                        i++;
                }



                /*
                msg.Format("j= %d jA= %d jB= %d",j,jA,jB);
                AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/
                if(jA==0 || jB==0) //bisogna prevedere quando non c'è tubo
                {

                /*      msg.Format("non c'è nessun oggetto!!!!");
                        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/
                        posizione[0]=0;
                        posizione[1]=0;
                        free(indiceRho);
                        free(indiceCandidati);

                        return(posizione);

                }



                ///*******CANDIDATO PIU VICINO**************************
                int totCandidati=2*jA+2*jB;

                //trovo il valore più vicino alla testa per ogni coppia (valore candidato)
                for(i=0; i<totCandidati;)
                {
                        indice1=indiceCandidati[i];
                        indice2=indiceCandidati[i+1];
```

```
		if((vettoreCampImm[indice1]<=vettoreCampImm[indice2]))
		{
			indiceRho[indice1]=2;
			indiceCandidati[i+1]=-indiceCandidati[i+1];
		}
		else
			{
				indiceRho[indice2]=2;
				indiceCandidati[i]=-indiceCandidati[i];
			}
	i=i+2;
	}


/*
	msg.Format("holaaaaaaaaa mundo");
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);
*/



	///*******CONFRONTO IN COORDINATE CARTESIANE**************************
	double deltaX=Dimens/2;	//bisogna calcolarlo dinamicamente
	double deltaY=Dimens/2;
	int flag=3;

	//faccio il confronto tra le X e le Y per i punti trovati
	for(i=0; i<numeroPingScanA; i++)
		for(j=numeroPingScanA; j<dimensione; j++)
		{
			indice1=indiceRho[i];
			indice2=indiceRho[j];
			if(indice1==2 && indice2==2)

				if((X[j]-deltaX)<X[i] && X[i]<(X[j]+deltaX)) //se la x trovata

					if((Y[j]-deltaY)<Y[i] && Y[i]<(Y[j]+deltaY))
					{
						indiceRho[i]=flag;
						indiceRho[j]=flag;
						flag++;
					}
		}

	if(flag==3)
	{
		/*msg.Format("non c'è nessun oggetto 2!!!!!!!!");
		AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/
		posizione[0]=0;
		posizione[1]=0;
		free(indiceRho);
		free(indiceCandidati);
		free(X);
		free(Y);

		return(posizione);

	}



/*
	msg.Format("Viva el Ecuador!!!!!!!! flag=%d", flag);
	AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);

*/
	//****************TEMPLATE MATCHING*********************/////////////////////////

	int N=10;//bisogna calcolarlo bene!
	double *Xfin, *Yfin, *CoXY;
```

```
Xfin= (double *)malloc(sizeof(double)*2*N);
Yfin= (double *)malloc(sizeof(double)*2*N);
CoXY= (double *)malloc(sizeof(double)*3*(flag-3));


int cont=3;
int flag2;
int k,m,k1,k2;
double R=Dimens/2;
double yprima, yseconda;
int Nelementi;
double Xmedio, Xmin, Xmax;
double Ymax,Ymin,Ymedio;
double errore;
int iCoXY=0;
double dif1, dif2;
int d,l;
double X1,X2,Xoffset,rho1,rho2;



do{

//FASE1: trovo per ogni coppia di candidati i N valori più vicini alla testa a partire del candidato
        k=0;
        l=0;    //quante volte entra per ogni coppia(dovrebbe essere 2!)
        for(i=0; i<totCandidati;)
        {
                indice1=indiceCandidati[i];
                indice2=indiceCandidati[i+1];
                if(indice1>0)       //se il primo indice è positivo è il più vicino
                {
                        j=0;
                        m=0;

                        if(indiceRho[indice1]==cont)
                        {
                                if(l>1){  k=0;    break;   }

                                if(l==0)
                                {       X1=X[indice1];
                                        rho1=vettoreCampImm[indice1];
                                }else
                                        {       X2=X[indice1];
                                                rho2=vettoreCampImm[indice1];
                                        }

                                while(m<N)//for(j=0; j<N; j++)
                                {
                                if((indice1-j)<0 || (indiceRho[indice1-j]>0 && j!=0))
                        break;

                                        if(indiceRho[indice1-j]==0 || j==0)
                                        {
                                                Xfin[k]=X[indice1-j]; //prendo i valori a
                                                sinistra
                                                Yfin[k]=Y[indice1-j];
                                                k++;
                                                //k1++;
                                                m++;
                                        }
                                        j++;
                                }
                                l++;
                        }
                }
                else                //se il primo indice è negativo è il più lontano
                {
                        j=0;
                        m=0;

                        if(indiceRho[indice2]==cont)
```

```
                                        {
                                                if(l>1){   k=0;      break;    }

                                                if(l==0)
                                                {          X1=X[indice2];
                                                           rho1=vettoreCampImm[indice2];
                                                }else
                                                           {          X2=X[indice2];

                                                           rho2=vettoreCampImm[indice2];
                                                           }

                                                while(m<N)//for(j=0; j<N; j++)
                                                {
                                                           if((indice2+j)>=dimensione ||
                                                           (indiceRho[indice2+j]>0 && j!=0))      break;

                                                           if(indiceRho[indice2+j]==0 || j==0)
                                                           {
                                                                      Xfin[k]=X[indice2+j];
                                                                      //prendo i valori a destra
                                                                      Yfin[k]=Y[indice2+j];
                                                                      k++;
                                                                      m++;
                                                           }
                                                           j++;
                                                }
                                                l++;
                                        }
                                }
                                i=i+2;
}


/*msg.Format("k=%d",k);
AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/

if(k>2*N-3)
{
          //FASE2: portare tutti i valori all'origine (0,0)
          Nelementi=k;

          //trovo la Y più grande e la tolgo a tutti i valori Y

          Ymax=Yfin[0];
          Ymin=Yfin[0];
          Ymedio=0.0;

          for(i=0; i<Nelementi; i++)
          {
                    if(Ymax<Yfin[i])
                              Ymax=Yfin[i];
                    if(Ymin>Yfin[i])
                              Ymin=Yfin[i];
                    Ymedio=Ymedio+Yfin[i];
          }

          Ymedio=Ymedio/Nelementi;

          for(i=0; i<Nelementi; i++)
                    Yfin[i]=Ymax-Yfin[i];

          //trovo la X media e la tolgo a tutte le X

          Xmedio=0.0;

          Xmax=Xfin[0];
          Xmin=Xfin[0];

          for(i=0; i<Nelementi; i++)
          {
                    if(Xmax<Xfin[i])
```

```
                                        Xmax=Xfin[i];
                                if(Xmin>Xfin[i])
                                        Xmin=Xfin[i];

                        }

                        if(rho1>rho2)
                        {
                                Xoffset=X2;
                                X2=X1;
                                X1=Xoffset;
                        }
                        Xoffset=(X1-X2);



                        Xmedio=(Xmax+Xmin-Xoffset)/2;

                        for(i=0; i<Nelementi; i++)
                                Xfin[i]=Xfin[i]-Xmedio;



                        //FASE3: correlazione tra i valori ottenuti e i valori teorici per un tubo

                        errore=0.0;
                        //d=Nelementi;

                        for(i=0; i<Nelementi-1; i++)
                        {
                                dif1=pow(R,2)-pow(Xfin[i],2);
                                dif2=pow(R,2)-pow(Xfin[i+1],2);

                                /*if(dif1<0)            dif1=-dif1;
                                if(dif2<0) dif2=-dif2;*/

                                if(dif1>=0 && dif2>=0)        //se la x è dentro il cerchio!
                                {
                                        yprima=sqrt(dif1); //valori per un cerchio di raggio R con centro in
                                        (0,0)
                                        yseconda=sqrt(dif2);

                                        dif1=Yfin[i]-yprima;
                                        dif2=Yfin[i+1]-yseconda;
                                        dif=dif1-dif2;          //faccio la diferenza delle diferenze!

                                        if(dif<0)  dif=-dif;

                                        errore=errore+dif;

                                }else      //aggiungiamo un errore grande per potere scartare dopo se non
                                corrisponde a un tubo
                                        {
                                                errore=errore+0.1;
                                                /*Xmedio=Xmedio-(Xfin[i]/d);
                                                //devo ricalcolare la media
                                                Xmedio=Xmedio*d/(d-1);
                                                d--;*/
                                        }
                        }

                        CoXY[iCoXY]=errore;
                        CoXY[iCoXY+1]=Xmedio;
                        CoXY[iCoXY+2]=Ymedio;
                        iCoXY=iCoXY+3;
                }

                cont++;

        }while(cont<flag);

        //*************TEMPLATE MATCHING*********************//////////////////////////
```

```
		//Trovo il valore con il errore più basso e che corrisponderebbe al tubo

		double minCo=CoXY[0];
		int iMin=0;

		for(i=0; i<iCoXY;)
		{
			if(minCo>CoXY[i])
			{
				minCo=CoXY[i];
				iMin=i;
			}
			i=i+3;
		}


		////////////////////////////////////////
/*		FILE *p;
		p=fopen("C:\\Sonar\\dati.txt","w");


		for(i=0; i<dimensione; i++)
			fprintf(p,"\nindiceRho %d=%d",i,indiceRho[i]);

		fprintf(p,"\n\n");

		for(i=0; i<totCandidati; i++)
			fprintf(p,"\nindiceCandidati %d=%d",i,indiceCandidati[i]);

		for(i=0; i<dimensione; i++)
		{fprintf(p,"\nX %d=%g",i,X[i]);
		fprintf(p,"\t\tY %d=%g",i,Y[i]);}


		fprintf(p,"\n\n");

		for(i=0; i<k; i++)
			fprintf(p,"\nXfin %d=%g",i,Xfin[i]);

		fprintf(p,"\n\n");

		for(i=0; i<k; i++)
			fprintf(p,"\nYfin %d=%g",i,Yfin[i]);

		for(i=0; i<iCoXY; i++)
			fprintf(p,"\nCoXY %d=%g",i,CoXY[i]);

		fclose(p);*/

		////////////////////////////////////////////////////////////////////////

		if(minCo>0.12*Nelementi/2 || iCoXY==0)
		{
		/*msg.Format("non c'è nessun tubo 3!!!!!!!!");
			AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/
			posizione[0]=0;
			posizione[1]=0;
			free(indiceRho);
			free(indiceCandidati);
			free(X);
			free(Y);
			free(Xfin);
			free(Yfin);
			free(CoXY);

			return(posizione);
		}

		posizione[0]=CoXY[iMin+1];
		posizione[1]=CoXY[iMin+2];
```

```
/*                      msg.Format("X=%g Y=%g", posizione[0],posizione[1]);
        AfxMessageBox(msg,MB_OK | MB_ICONINFORMATION);*/


        free(indiceRho);
        free(X);                                        //dealloco X e Y che ormai non servono più
        free(Y);
        free(indiceCandidati);
        free(Xfin);
        free(Yfin);
        free(CoXY);


        return(posizione);


}

///*******************************OBJECTDETECTOR*************************
```

# BIBLIOGRAFIA

**1.** V.S. Frost, J.A. Stiles, K.S. Shanmugan, and J.C. Holtzman, "A model for radar images and its application to adaptive filtering of multiplicative noise", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-4(2), pp. 157-166, 1982.

**2.** T-C. Jen, B. Hsieh, and S-J. Wang, "Image contrast enhancement based on intensity-pair distribution, " *IEEE Int. Conf. Image Processing*, Genoa, Italy, pp. 913-916, September 2005.

**3.** X. Lurton, *An introduction to Underwater Acoustics: principles and applications*, Springer, Praxis Publishing, Chichester, UK, 2002.

**4.** V. Murino and A. Trucco, "Three-Dimensional Image Generation and Processing in Underwater Acoustic Vision," Proceedings of the IEEE, Vol. 88, pp. 1903-1946, December 2000.

**5.** J.P. Fish y H.A. Carr, "SOUND UNDERWATER IMAGES: A guide to the Generation and Interpretation of Side Scan Sonar Data". 1991

**6.** SuperSeaking DFP Datasheet. TRITECH.

**7.** Super Seaking DFS Datasheet. TRITECH.