Curso 2011/12
CIENCIAS Y TECNOLOGÍAS/2
I.S.B.N.: 978-84-15287-91-9

ÓSCAR CASANOVA GONZÁLEZ

Estudio e implementación de esquemas Neuro-Fuzzy aplicados al procesamiento de variables cinemáticas en el campo del diagnóstico y la navegación inercial

Director
G. NICOLÁS MARICHAL PLASENCIA

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

A Sandra, por su infinita paciencia,
apoyo y comprensión.

A mis padres y hermanos,
por estar siempre ahí.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento, en primer lugar, al Dr. D. Graciliano Nicolás Marichal Plasencia por la formación recibida durante estos años, además de la preocupación y motivación que ha puesto en este trabajo.

En segundo lugar quiero expresar mi más profundo agradecimiento al Dr. D. José Luis Gónzalez Mora de la Universidad de La Laguna, por poner los medios necesarios para la consecución de esta tesis.

También me gustaría expresar mi agradecimiento a los Profesores Dr. D. Mariano Artés Goméz de la Universidad Nacional a Distancia y al Dr. D. Juan Carlos García Prada de la Universidad Carlos III de Madrid por su inestimable colaboración.

Por último, quiero dar las gracias a todos mis compañeros del Grupo de Neuroquímica y Neuroimagen por su ayuda desinteresada en este trabajo. En especial a Carlos Merino Gracia y a Miguel Torres Gil, cuya colaboración ha sido decisiva en algunos aspectos técnicos de esta tesis.

INDICE

INTRODUCCIÓN

En esta memoria se presenta un trabajo de investigación focalizado en la aplicación de esquemas neuro-fuzzy en dos problemas clásicos de la ingeniería, como son el diagnóstico de piezas mecánicas defectuosas, en concreto rodamientos, y la corrección de la deriva de las variables cinemáticas obtenidas a través de un sistema de navegación inercial.

En el capítulo 1 se procede a introducir los fundamentos de las redes neuronales y de la lógica fuzzy. En el capítulo 2 por su parte se profundiza en los sistemas neuro-fuzzy que posteriormente serán objeto de aplicación en los problemas indicados. Concretamente, se hace especial hincapié en los denominados en la terminología inglesa como Neural Fuzzy Systems (NFS). En el capítulo 3 se abordan las tecnologías utilizadas para la determinación de aceleraciones en el contexto de la medida de vibraciones y la estimación de medidas inerciales. Particularmente, se describen los tipos de acelerómetros más comunes y sus aplicaciones. De forma especial se describen los acelerómetros microelectromecánicos (MEMS – Microelectromechanical system) cuyo bajo coste y robustez, así como capacidad de miniaturización han permitido recientemente su uso extensivo. En este capítulo se describe el acelerómetro Brüel & Kjaer 4383 utilizado posteriormente para la medición de vibraciones en rodamientos y la unidad Mtx de Xsens Inc, que incorpora acelerómetros y giroscopios con tecnología MEMS y fue utilizada como sistema de medida inercial en los trabajos de esta tesis. Por último, se presentan las bases de los algoritmos de visión por computador que serán usados en el capítulo 6, describiendo de forma general el método de Lucas-Kanade. En el capítulo 4 se describe la importancia del mantenimiento de los sistemas mecánicos. Inicialmente se da una visión general, para posteriormente centrarse en la determinación de defectos en rodamientos. En dicho capítulo se muestra el reciente interés que han suscitado las técnicas neuro-fuzzy en este contexto. En el capítulo 5 se expone el montaje experimental realizado para determinar fallos en rodamientos y se muestra la adaptación realizada del problema a los esquemas neuro-fuzzy. También se describen los ensayos con uno de los esquemas neuro-fuzzy más difundidos, concretamente el denominado ANFIS (Adaptive Neuro-Fuzzy Inference Systems). Posteriormente, se describe y aplica un esquema alternativo de sistema neuro-fuzzy, para finalmente comparar los resultados de ambos esquemas, poniendo de manifiesto las buenas propiedades del esquema neuro-fuzzy alternativo cuando se aumenta el número de

entradas. En el capítulo 6 se presenta la aplicación del sistema neuro-fuzzy alternativo expuesto en el capítulo 5 al problema de la fusión sensorial entre un algoritmo de visión y el algoritmo strapdown para la mejora de la estimación de las posiciones partiendo de la medida de las aceleraciones. En dicho capítulo se muestran varias aproximaciones al problema, destacando los resultados obtenidos en relación a las estrategias clásicas empleadas. Por último se presentan las conclusiones del trabajo.

1 REDES NEURONALES Y LÓGICA DIFUSA

En este capítulo se describen los conceptos básicos y características de las ***redes neuronales*** *y la* ***lógica difusa****. En la primera parte se da una introducción al concepto de red neuronal artificial, mencionando sus características y propiedades más notables. A continuación se muestra en detalle la estructura de una red neuronal artificial partiendo de la neurona básica y comentando algunos tipos de redes según su estructura. Para completar esta primera parte se describen las redes de Kohonen y las redes basadas en funciones de base radial, ya que desempeñan un papel importante en el desarrollo de esta tesis. En la segunda parte de este capítulo se hace una introducción a la lógica difusa mediante la descripción de un ejemplo concreto.*

1.1 Introducción

Dada la flexibilidad y adaptabilidad de los mecanismos cognitivos del modelo de control neuronal biológico, éstos han sido ampliamente estudiados con el fin de trasladarlos al campo de la computación para resolver problemas donde existen imprecisiones e incertidumbres en los datos. En este ámbito, surgen dos paradigmas computacionales: la lógica difusa y las redes neuronales artificiales.

Una red neuronal artificial es un sistema que simula en cierta medida el comportamiento de un conjunto de neuronas siguiendo un enfoque conexionista. Estos sistemas tienen multitud de ventajas, siendo la principal su capacidad de aprendizaje y adaptación al problema a resolver. La lógica difusa en cambio permite expresar conjuntos de reglas sobre conceptos imprecisos. Se adaptan muy bien a entornos humanos, donde se emplean expresiones poco determinadas como “un poco” o “demasiado”.

Estas técnicas, complementarias la una de la otra, permiten a las máquinas tratar datos ambiguos, incompletos o cuyo comportamiento no responde a un modelo matemático suficientemente sencillo o conocido.

1.2 Redes neuronales artificiales

Las *redes neuronales artificiales* (RNA) son un paradigma de computación basado en el aprendizaje automático, cuya estructura se inspira en el modelo biológico de las neuronas animales. Kohonen definió en 1989 [KOH89] las redes neuronales como "redes de elementos simples, usualmente adaptativos, masivamente interconectados en paralelo y con una organización jerárquica, que tratan de interactuar con algún sistema del mismo modo que lo hace el sistema nervioso biológico". Su uso se ha vuelto muy popular debido a su elevada capacidad para aproximar complejas funciones matemáticas y trabajar con datos incompletos o imprecisos.

En el año 1943, Warren McCulloch y Walter Pitts presentaron su trabajo "A logical calculus of the ideas immanent in nervous activity" [MCC43] donde se modela una red neuronal por primera vez, sentando algunas de las bases que aun hoy en día se conservan. En este modelo, una neurona es un dispositivo con *n* entradas y una única salida, pudiendo tener esta solamente dos estados: *activa* o *inactiva*. Una red de neuronas consistía por tanto en un grupo de nodos donde las salidas de unas neuronas estaban conectadas a las entradas de otras, funcionando de forma sincronizada. Posteriormente, han aparecido diversas estructuras de red, como el Perceptron en 1957, Adeline en 1960, Avalancha en 1967, Backpropagation en 1974, Hopfield y SOM en 1980, ART en 1986, etc. De esto podemos deducir que no es un concepto nuevo, si bien no ha sido hasta una época reciente cuando la tecnología ha evolucionado lo suficiente como para que se haya podido aplicar de forma práctica.

Las RNA poseen algunas características que hacen su aplicación sumamente atractiva:

- *Capacidad de aprendizaje*. Las RNA son capaces, mediante un proceso de entrenamiento, de aprender el comportamiento de un sistema. Este entrenamiento se lleva a cabo mostrándole las entradas al sistema y las salidas que produce.

- *Auto organización*. Una RNA, durante el proceso de aprendizaje, crea de forma automática su propia representación de la información.

- *Tolerancia a fallos*. Dado que las RNA almacenan la información de forma distribuida en sus neuronas, una red puede soportar la destrucción de parte de sus neuronas y conservar suficiente información como para seguir funcionando.

- *Flexibilidad*: Las RNA son capaces de manejar pequeñas alteraciones en los datos de entrada y funcionar correctamente. P.ej. Una red dedicada a la clasificación de imágenes dará la misma salida con una imagen de entrada limpia que con una imagen contaminada con ruido

Sin embargo, las RNA sufren también de algunos inconvenientes, sobre todo, derivados de su opacidad. Precisamente por su capacidad auto-organizativa y

paralelismo masivo, en una red compleja, el número de operaciones que supone el obtener un resultado es tan elevado que no es posible saber qué variables han contribuido en mayor o menor medida al resultado o qué reglas se han empleado para obtener una conclusión.

1.2.1 Estructura de una neurona artificial

Como se ha mencionado anteriormente, la *neurona artificial* está basada en el modelo biológico. Por tanto, una neurona artificial es un elemento básico que recoge la información de otras neuronas o del exterior de la red, la integra y procesa y produce una salida que o bien alimenta a otras neuronas artificiales, o es usada como salida del sistema.

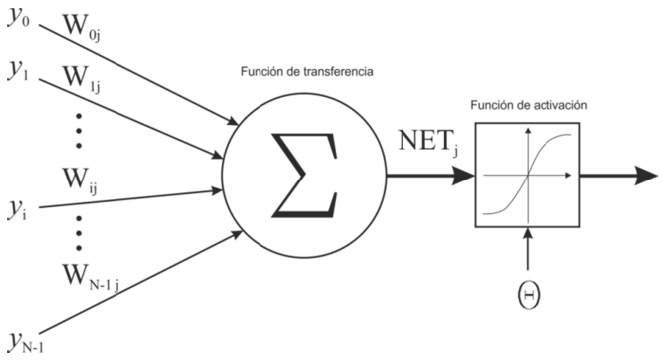


Figura 1.1. Neurona artificial

Las conexiones entre dos neuronas *i* y *j* tienen un peso asociado (W_{ij}). Este peso representa la fuerza de la sinapsis. Se dice que una sinapsis es fuerte si la información que transfiere contribuye en gran medida a la alteración del estado de la neurona receptora.

La salida producida por una neurona depende de dos funciones: la *función de transferencia* y la *función de activación*.

- La *función de transferencia,* también llamada función de propagación o función de excitación, consiste en la suma ponderada de las entradas de la neurona, que viene dada por la expresión:

$$NET_j = \sum_i y_i \cdot W_{ij} \qquad \textbf{(1.1)}$$

  Si el resultado supera un umbral, se considera que la conexión es excitatoria, y por tanto su resultado se transfiere a la función de activación. Si no supera dicho umbral, se denomina inhibitoria y la neurona no se activa.

- La *función de activación*, que se aplica al valor de salida de la función de transferencia. Habitualmente se emplea para acotar el valor de dicha función, adaptándolo al uso que se vaya a hacer de las salidas. La función de activación

puede también no estar presente, mostrando en la salida directamente el valor de la función de transferencia.

$$A_j = f(NET_j) \tag{1.2}$$

Como se puede ver en la ecuación (1.2), la activación de una neurona depende de sus entradas, de los pesos sinápticos y de la función de transferencia escogida.

1.2.2 Estructura de una red neuronal artificial (RNA)

Una neurona artificial por sí sola, al igual que en el ámbito biológico, no es capaz de resolver un problema. Para ello, las neuronas artificiales se deben agrupar, interconectándolas de forma correcta y asignando los pesos adecuados a las conexiones sinápticas. El comportamiento de la red dependerá en gran medida del número de neuronas y la organización de sus capas, y como consecuencia, del número de enlaces que las interconectan y los pesos asignados.

1.2.2.1 Organización de las neuronas

Dentro de una red, las neuronas se dividen en *capas*. Una capa es un conjunto de neuronas cuyas entradas provienen de la misma fuente, que puede ser otra capa de neuronas, y cuyas salidas se dirigen al mismo destino, que también puede ser otra capa de neuronas. Una RNA habitualmente se organiza en una *capa de neuronas de entrada*, una o varias *capas de neuronas ocultas* y *una capa de neuronas de salida*.

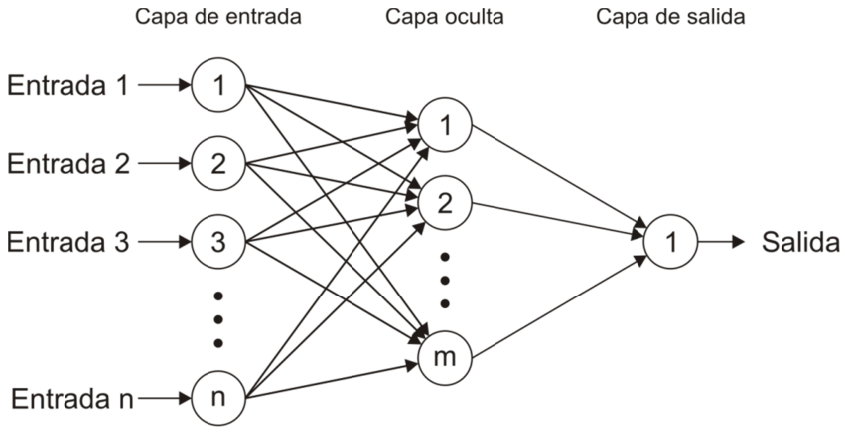


Figura 1.2. Red neuronal de 3 capas

- *Capa de entrada* es la que recibe los datos provenientes de las fuentes externas de la red.

- *Capa oculta*. No tiene conexión directa con el exterior. El número de capas ocultas puede variar dependiendo de la arquitectura de red escogida, pudiendo incluso no existir. El número de capas ocultas, así como la forma de interconexión, es lo que determina la tipología de la red.

- *Capa de salida* es la que transfiere la información al exterior de la red.

1.2.2.2 Estructura de conexión entre neuronas

La señal de salida de una neurona puede ser la entrada de otra neurona, o estar conectada a una de sus propias entradas. En el segundo caso se denomina *conexión autorecurrente*. En función de la propagación de la información entre las capas, podemos hablar de red de propagación hacia adelante o de propagación hacia atrás.

- *Red de propagación hacia adelante*, cuando no existen conexiones de una capa que tenga como destino una capa anterior, o neuronas del propio nivel. Este tipo de redes son muy rápidas, dado que la información solo circula en una única dirección y no hay tiempos de espera durante los cuales unas neuronas están actuando sobre otras hasta que su estado se estabiliza.

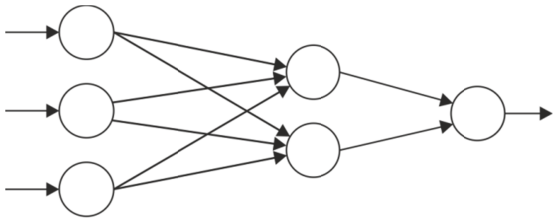


Figura 1.3. Red de propagación hacia adelante

- *Red con retropropagación*, cuando existe al menos una conexión entre la salida de una neurona y la entrada de otra neurona de la misma capa o una capa precedente. Al haber retroalimentación, las interacciones entre las neuronas son recurrentes. Esto provoca que ante un cambio en la entrada, se sucedan una serie de cambios sucesivos en las neuronas hasta alcanzar un estado estable, esto es, cuando no hay cambios en la salida de ninguna neurona.

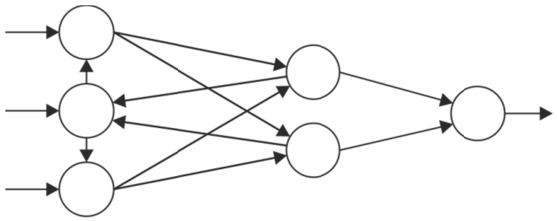


Figura 1.4. Red con retropropagación

1.2.2.3 Redes ACON y OCON

A la hora de enfrentarse al diseño de una RNA muy compleja, es posible adoptar dos enfoques distintos. Por un lado se puede optar por construir una única superred que contemple todas las soluciones posibles al problema propuesto. Esto se conoce como red *ACON* (All Class in One Network).

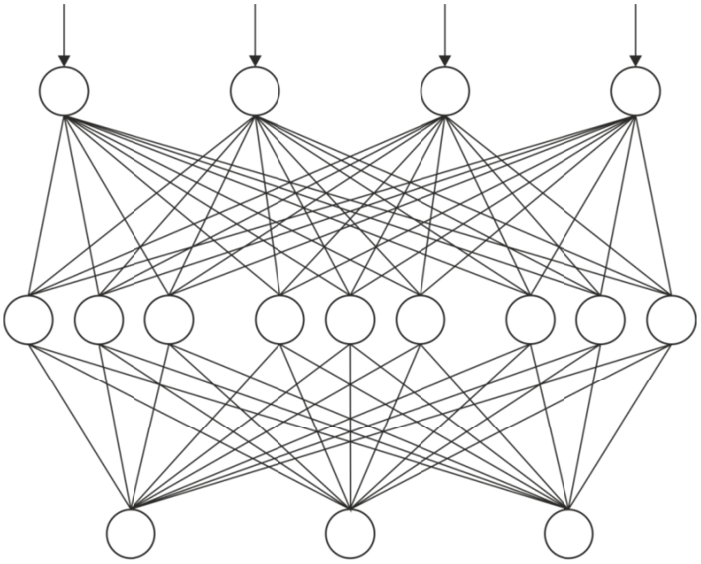


Figura 1.5. Red ACON

Como se puede ver en la figura 1.5, todas las neuronas de la capa oculta intervienen en la salida proporcionada por cada una de las neuronas de la capa de salida. Esto se traduce en una red con un elevado número de sinapsis, lo cual acarrea una mayor dificultad en la etapa de aprendizaje pudiendo llegar incluso a la imposibilidad de alcanzar un resultado satisfactorio.

En contraposición a la arquitectura ACON se define la arquitectura *OCON* (One Class in One Network).

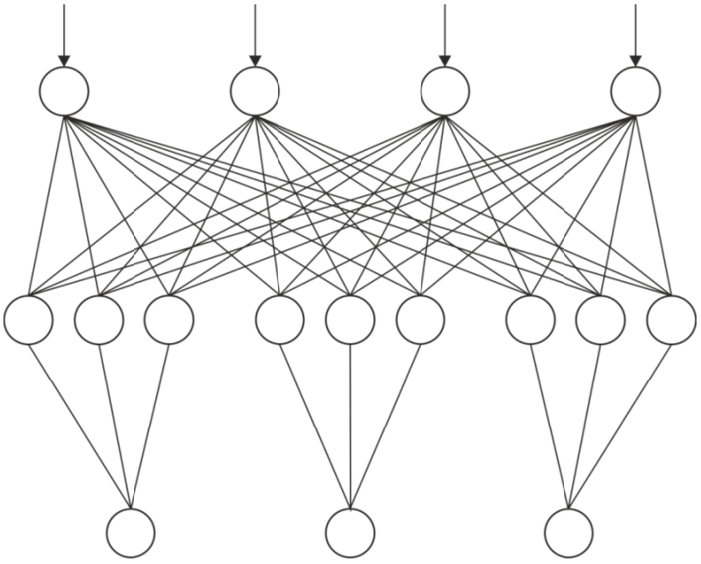


Figura 1.6. Red OCON

En la arquitectura OCON, el diseño se aborda de una forma totalmente diferente. En este caso se divide la red en varias subredes más pequeñas, cada una dedicada a resolver una parte de la solución. Si se observa la red de la figura 1.6, se aprecia cómo la superred de la figura 1.5 se ha dividido en tres subredes diferenciadas, estando cada una

dedicada a extraer solo una parte de la solución a partir de las mismas entradas. Cada una de estas redes es más pequeña y especializada y por tanto con mayores posibilidades de alcanzar un aprendizaje adecuado.

1.2.3 Aprendizaje

En el paradigma clásico de programación, el programador modela matemáticamente un problema y expresa su solución como un algoritmo que representa los pasos a seguir para obtener la solución deseada a partir de unos datos de entrada. En las RNA el enfoque es completamente distinto. Partiendo de un conjunto de datos de entrada suficientemente significativo, se somete la red a un proceso de aprendizaje automático denominado *entrenamiento*. Durante este proceso de entrenamiento se modifican los pesos de las conexiones sinápticas de acuerdo al algoritmo de aprendizaje seleccionado hasta conseguir la salida deseada ante la entrada propuesta. Por lo tanto, el conjunto de pesos de las conexiones sinápticas determina el conocimiento de la red.

1.2.3.1 Clases de aprendizaje

Existen cuatro clases de aprendizaje en función del método empleado:

- *Aprendizaje supervisado*. En este tipo de entrenamiento se le presenta a la red una serie de patrones (*conjunto de entrenamiento*) a la entrada de forma iterativa, especificando los patrones de salida correctos. En cada iteración se modifican los pesos de las sinapsis para que la salida de la red se ajuste lo más posible al patrón deseado.

- *Aprendizaje no supervisado o autoorganizativo*. En el aprendizaje no supervisado no existe un conjunto de patrones que indiquen la salida correcta, sino que se presentan los patrones de entrenamiento a la entrada, y es la propia red la encargada de extraer las características comunes entre ellos, produciendo una clasificación automática. Cabe indicar que en este tipo de aprendizaje, el número de patrones en el conjunto de entrenamiento suele ser elevado para que se puedan ajustar correctamente los pesos sinápticos.

- *Aprendizaje reforzado*. Se sitúa entre los dos tipos de entrenamiento anteriores. En el aprendizaje reforzado se presentan los patrones de entrenamiento a la entrada, pero no se comparan con un conjunto de patrones de salida correctos. Simplemente se indica si la salida es correcta o no.

- *Aprendizaje híbrido*. El aprendizaje híbrido se emplea en algunas redes multicapa y consiste en entrenar unas capas con el método supervisado y otras con el método no supervisado.

1.2.3.2 Problemas comunes en el entrenamiento

Como se ha visto, el proceso de entrenamiento tiende a minimiza el error con los patrones de entrenamiento. Es posible que este error tarde demasiado tiempo en

converger a un mínimo (donde se consideraría la red entrenada), que este error mínimo no sea suficientemente pequeño, o simplemente que a la hora de usar datos reales como entrada, los resultados de la red no sean satisfactorios. En los dos primeros casos, el problema puede estar en una mala elección de la topología de la red, o en el conjunto de patrones de entrenamiento, que no sea suficientemente representativo. En el último caso, sin embargo, el problema suele deberse al *sobreentrenamiento*.

El sobreentrenamiento se da en los casos donde la red tiene un nivel de complejidad muy alto o ha sido entrenada durante demasiados ciclos. Si la red es demasiado compleja, es decir, tiene un número demasiado elevado de neuronas y sinapsis, también tendrá una gran capacidad para almacenar en sus pesos sinápticos los valores necesarios para memorizar todos los patrones de entrenamiento. Debido a este fenómeno, la red responde perfectamente a los patrones de entrenamiento, pero no a las entradas reales. Se dice entonces que la red no tiene capacidad de *generalización*, que es una de las características fundamentales de este modelo de cómputo. Para paliar este problema se recurre al uso de tres conjuntos de patrones:

- *Conjunto de entrenamiento*. Son los patrones introducidos en la red de forma iterativa durante la fase de entrenamiento.

- *Conjunto de validación.* Se emplean para evitar el sobreentrenamiento. Durante el entrenamiento se comprueba el comportamiento de la red con el conjunto de validación a intervalos regulares. Como estos patrones no modifican los pesos sinápticos de la red, miden la capacidad de generalización de la misma de una forma indirecta. Cuando el error de validación comienza a crecer, es síntoma de que la red está empezando a sobreentrenarse y es el momento de detener el entrenamiento. En la práctica es difícil determinar este punto con absoluta exactitud, dado que es posible que el error de validación presente mínimos locales. Habitualmente se recurre al uso de la experiencia para determinar el punto óptimo de parada del entrenamiento.

- *Conjunto de test*. Debido a la forma en la que se emplea, es posible que el conjunto de validación termine condicionando el entrenamiento de la red. Aunque sus patrones no se usen para modificar los pesos de la red, sí que se detiene el entrenamiento cuando se consiguen valores de error aceptables con sus elementos. Esto hace necesario un tercer conjunto, que se emplea para evaluar la red una vez ha sido entrenada, y que está formado por patrones que no han sido presentados a la red durante la fase de entrenamiento. El error obtenido con este conjunto de test da, en la medida de lo posible, un valor representativo de la capacidad de generalización y adecuación de la red al problema real.

1.2.4 Mapas de características autoorganizativos (Redes de Kohonen)

Los *mapas de características autoorganizativos*, también conocidos como *Redes de Kohonen*, son un tipo de red neuronal desarrollado por Teuvo Kohonen. Este tipo de

red se organiza de forma autónoma durante el entrenamiento, de forma que se adapta lo mejor posible a los patrones de entrada. En las redes de Kohonen, las neuronas que representan patrones parecidos aparecen juntas en el espacio de salida, es decir, se crea cierta relación de orden topológico. Esta característica las hace especialmente aptas para la detección de grupos o clusters en los datos de entrada, de los cuales se desconoce su organización a priori.

1.2.4.1 Arquitectura

Las redes de Kohonen no tienen capas ocultas, solamente tienen una capa de entrada o de sensores, y una capa de salida o de cálculo. Cada neurona de la capa de entrada está unida mediante conexiones hacia adelante con todas las neuronas de la capa de salida, y cada neurona de la capa de salida está conectada con sus vecinas lateralmente.

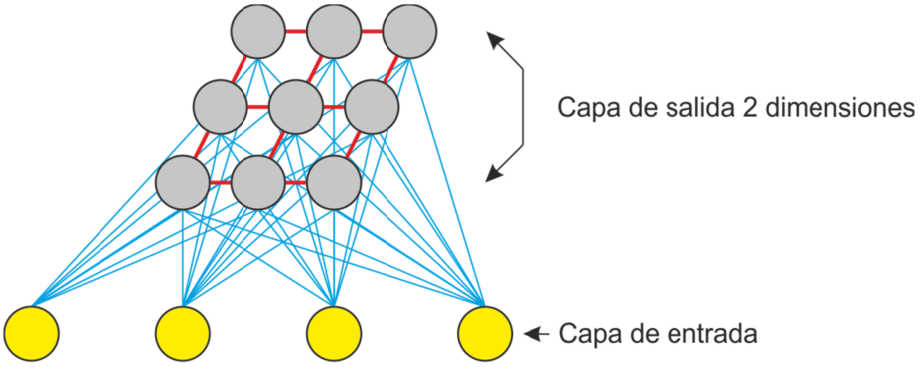


Figura 1.7. Red de Kohonen

Las neuronas de la capa de salida pueden estar conectadas de diferentes formas: lineal, cuadrada (como en la figura 1.7), hexagonal, etc… Esto define en gran medida el comportamiento de la red, puesto que, como se verá en el siguiente apartado, la relación de vecindad condiciona fuertemente el entrenamiento.

1.2.4.2 Aprendizaje

Las redes de Kohonen se basan en el aprendizaje competitivo. El aprendizaje competitivo es no supervisado, por lo tanto requiere que los patrones de entrada tengan un elevado grado de redundancia. En este tipo de entrenamiento, todas las neuronas de salida reciben la misma entrada. A continuación se selecciona una única neurona de salida "ganadora", la de mayor salida, y se refuerzan los pesos de ésta y de su vecindad.

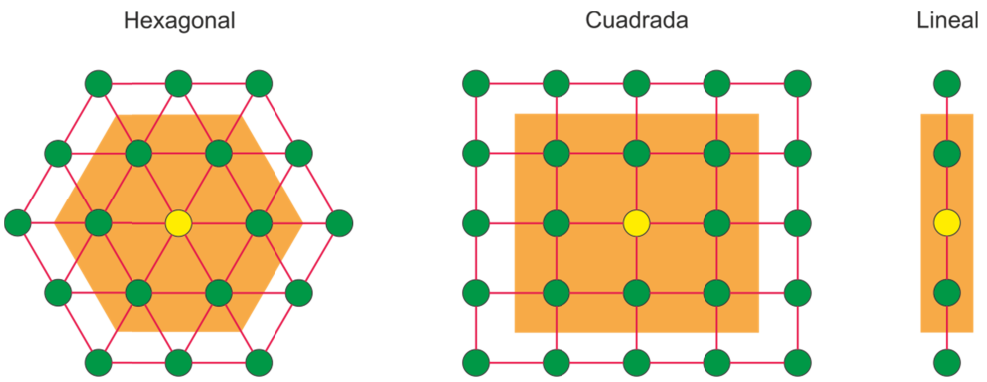


Figura 1.8. Tipos de vecindad

Aquí se pueden definir dos tipos de competición: blanda y dura. Se habla de competición dura cuando se modifica únicamente el peso de la neurona ganadora. En la competición blanda se modifica el peso de la neurona ganadora y en menor medida los pesos de las neuronas vecinas.

El algoritmo de entrenamiento básico se puede describir de la siguiente manera:

1. Se inicializan los pesos de forma aleatoria

$$W_i = [w_{i1}, w_{i2}, \dots, w_{in}] \tag{1.3}$$

2. Se presenta un vector de entrenamiento

$$X = [x_1, x_2, \dots, x_n] \tag{1.4}$$

3. Se escoge la neurona k más cercana al vector de entrada

$$k/\|X - W_k\| = \min_i\{\|X - W_i\|\} \tag{1.5}$$

4. Se modifican los pesos de la neurona ganadora y su vecindad, haciéndolos más próximos al vector de entrada

$$\Delta w_{ij} = \propto \cdot (x_j - w_{ij}) \tag{1.6}$$

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij} \tag{1.7}$$

α es la velocidad de aprendizaje

A medida que progresa el entrenamiento, los parámetros de vecindad y velocidad de aprendizaje se van modificando. La velocidad de aprendizaje se va reduciendo, así como la vecindad, pasando de un entrenamiento competitivo blando a uno casi totalmente duro. Un ejemplo de evolución de los parámetros puede ser el siguiente:

- Velocidad de aprendizaje:

$$\alpha_t = \alpha_0 \cdot \left(1 - \frac{t}{T}\right) \quad \textbf{(1.8)}$$

donde *T* es el número total de iteraciones, *t* es la iteración actual y α_0 es la velocidad de aprendizaje inicial.

- Vecindad:

$$\Omega_t = ent\left(\Omega_0 \cdot \left(1 - \frac{t}{T}\right)\right)$$
$$\Omega_0 \approx ent\left(\frac{1}{2}n\right) \quad \textbf{(1.9)}$$

donde *T* es el número total de iteraciones, *t* es la iteración actual, *n* es el número de neuronas de la capa y *ent(.)* es una función cuyo valor es la parte entera de la variable.

Sobre el algoritmo básico de entrenamiento se pueden aplicar multitud de variantes, como la incorporación de reglas que inhiban la activación de una neurona cuando es ganadora con demasiada frecuencia o la modificación de la topología del vecindario a medida que transcurre el entrenamiento.

1.2.5 Redes neuronales basadas en funciones de base radial (Radial Basis Function Networks – RBFN)

Las redes RBFN desarrolladas por Broomhead y Lowe [BRO88] presentan un modelo de red feed-forward empleado habitualmente como aproximador de funciones no lineales, o como clasificador en aquellos problemas donde las clases no son linealmente separables.

1.2.5.1 Función de base radial

Las funciones de base radial son funciones cuyo valor de salida depende de la distancia de la entrada a un punto denominado *centro*. La función de Gauss es un ejemplo de este tipo de funciones.

Las funciones de base radial tienen al menos dos parámetros: *centro* y *anchura*.

- El centro es el punto donde la función alcanza un extremo. En el caso de la función de Gauss, el centro es la media.
- La anchura indica el índice de variación de la función a medida que la variable se aleja del centro. En el caso de la función de Gauss, la anchura corresponde a la varianza.

Este tipo de funciones siempre son simétricas con respecto al centro.

1.2.5.2 Red neuronal basada en funciones de base radial

Las RBFN tienen una estructura de tres capas, como se puede observar en la figura 1.9. En este caso se representa una red de n entradas y una sola salida.

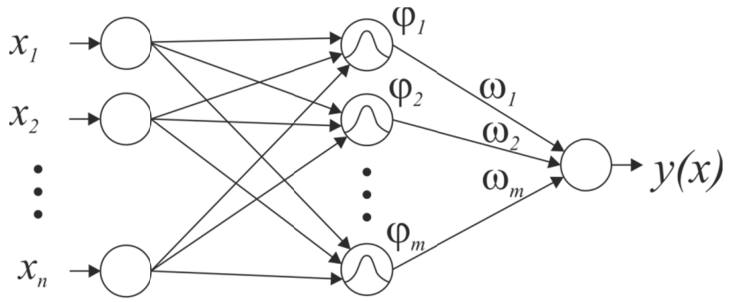


Figura 1.9. Estructura de una red de base radial con *n* entradas y una salida

- En la primera capa, cada neurona replica su entrada a todas las neuronas de la capa oculta. Dado que esta capa no hace ningún tipo de procesamiento, algunos autores no la consideran una capa en sí, describiendo este modelo de red como de dos capas.

- En la segunda capa de neuronas, la función de activación de cada neurona es una función de base radial. Esta capa realiza una transformación local y no lineal del espacio de entrada. Habitualmente se usa una función gaussiana como función de base radial.

$$\varphi_i(x) = \varphi_i(x, c^i, \sigma^i) = e^{\left(\frac{1}{2}\frac{\sum_k (x_k - c_k^i)^2}{{\sigma^i}^2}\right)}, \quad \begin{matrix} i = 1 \dots m \\ k = 1 \dots n \end{matrix} \tag{1.10}$$

  donde:

  - x es el vector de entrada y x_k su k-esima componente.
  - c^i representa el vector de centros de la función de base radial de la neurona i-ésima en la capa oculta. c_k^i es su k-ésima componente.
  - σ^i representa la anchura de la función de base radial de la neurona i-ésima en la capa oculta.

- En la capa de salida se lleva a cabo una combinación lineal de las salidas de la capa oculta. Esta combinación consiste en la suma ponderada de los resultados de las funciones de base radial multiplicados por unos pesos ω_i.

$$y(x) = \sum_{i=1}^{m} \omega_i \varphi_i(x) \quad \textbf{(1.11)}$$

Como se puede observar, para poder aplicar esta red es necesario determinar una serie de parámetros, que son:

- El número m de neuronas de la capa oculta.
- Los centros c^i de las funciones de base radial.
- Los parámetros de anchura σ^i de las funciones de base radial.
- El conjunto de pesos ω_i.

Estos parámetros se determinan mediante el procedimiento de aprendizaje de la red.

1.2.5.3 Aprendizaje de la RBFN

El método de entrenamiento habitual para este tipo de redes consiste en un entrenamiento híbrido en el que se emplea un método no supervisado para determinar los parámetros de la capa oculta y un algoritmo supervisado para los parámetros de la capa de salida.

1.2.5.3.1 Fase no supervisada

Dado que las neuronas de la capa oculta representan distintas zonas del espacio de entrada, el cálculo de los centros, anchuras e incluso el número de neuronas de la capa oculta se puede abordar como un problema de clasificación.

Para determinar los centros de cada función de base radial se puede usar un mapa autoorganizativo de Kohonen descrito en el apartado 1.2.4. Mediante este procedimiento, una vez presentando un conjunto significativo de vectores del espacio de entrada, los vectores de pesos de la red de Kohonen habrán tomado los valores representativos de cada clase del espacio de entrada, que en definitiva son sus centros.

Las anchuras o desviaciones de cada función de base radial se deben calcular de forma que los solapamientos entre funciones sean mínimos. Para ello se pueden aplicar distintas aproximaciones, como obtener la media de las distancias euclídeas del centro c_i a los p centros más cercanos:

$$\sigma^i = \frac{1}{p} \sum_{p} \left\| c_i - c_p \right\| \quad \textbf{(1.12)}$$

o calcular la media geométrica de la distancia del centro a sus dos vecinos más cercanos s y t:

$$\sigma^i = \sqrt{\|c_i - c_t\| \|c_i - c_s\|} \quad \textbf{(1.13)}$$

1.2.5.3.2 Fase supervisada

Para determinar los pesos ω_i se suele emplear el método de gradiente descendente, utilizando como medida del error la suma al cuadrado de los errores de las muestras del conjunto de entrenamiento:

$$E = \frac{1}{2}\sum_k (s_k - y_k)^2 \quad \textbf{(1.14)}$$

donde:

- y_k es la salida real de la red ante el patrón k-ésimo
- s_k es la salida deseada de la red ante el patrón k-ésimo

La corrección de los pesos es un proceso iterativo en el que se corrigen los pesos para cada una de las muestras del conjunto de aprendizaje según la regla:

$$\omega_i(t+1) = \omega_i(t) - \tau \frac{\partial E}{\partial \omega_i} \quad \textbf{(1.15)}$$

$$\frac{\partial E}{\partial \omega_i} = -(s_k - y_k)\, \varphi_i(x_k) \quad \textbf{(1.16)}$$

donde:

- τ es la velocidad de aprendizaje.
- k es el índice del patrón del conjunto de entrenamiento que está siendo presentado.
- i es el índice del peso que está siendo ajustado.

1.3 Lógica difusa

En el mundo real, el ser humano maneja conceptos que no son cuantificables con precisión y que por tanto no son manejables directamente por un ordenador. Si se desea aplicar reglas a este tipo de variables, es necesario recurrir a técnicas como la *lógica difusa*. La lógica difusa tiene la propiedad de trabajar con incertidumbre en los datos y la ventaja de poder expresar el conocimiento mediante un conjunto de reglas. Este concepto se puede ver mejor a través del siguiente ejemplo.

1.3.1 Ejemplo de un sistema fuzzy

Si se desea expresar la temperatura de una habitación, un ser humano puede decir que hace *un poco* de calor, que es *muy* fría, que es agradable, etc. Al intentar modelar la variable temperatura para controlar automáticamente un sistema de climatización, se presenta el problema de cómo definir formalmente qué es "un poco", qué es "adecuado", o qué es "mucho". Una primera aproximación a un control automático podría consistir en dividir el rango de temperatura en cinco partes y activar el climatizador en función del siguiente algoritmo:

```
Si (temperatura < 18°)
      activar calefacción al 100%
Si (temperatura >= 18°) y (temperatura < 21°)
      activar calefacción al 50%
Si (temperatura >= 21°) y (temperatura < 25°)
      desactivar calefacción
      desactivar aire acondicionado
Si (temperatura >= 25°) y (temperatura < 28°)
      activar aire acondicionado al 50%
Si (temperatura >= 28°)
      activar aire acondicionado al 100%
```

Sin duda, este control no representa la forma en la que un ser humano haría dicha labor. Una persona a 20,99ºC según las reglas expuestas se consideraría que está en un entorno frío, sin embargo no parece razonable que ya a 21º C se pueda determinar que la misma persona se encuentra en un ambiente cómodo, siendo la diferencia de temperatura de sólo 0,01ºC. Por otro lado, este algoritmo estaría permanentemente activando y desactivando los dispositivos cuando la temperatura se situase en los valores frontera de 21º y 25º, con el consiguiente derroche energético.

Para poder resolver este problema empleando lógica difusa, primero se deben definir los *conjuntos difusos* o *conjuntos fuzzy* a los cuales pertenece cada una de las variables con un cierto *grado de pertenencia*. La pertenencia a un conjunto se define mediante una *función de pertenencia*. A este proceso se le denomina coloquialmente *fuzzificación* de las variables.

- Para la variable de entrada "temperatura" se pueden definir tres conjuntos: mucho frío, frío, adecuada, calor y mucho calor

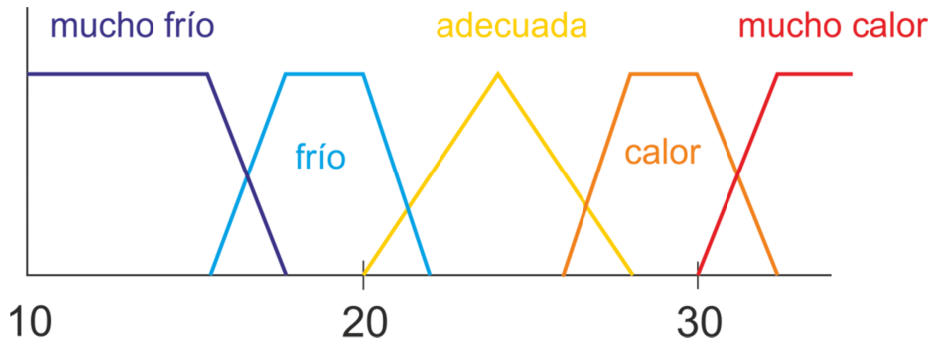


Figura 1.10. Funciones de pertenencia para la variable temperatura

- Para las variables de salida "calefacción" y "aire acondicionado", se definen dos conjuntos a modo de niveles de activación: bajo y alto

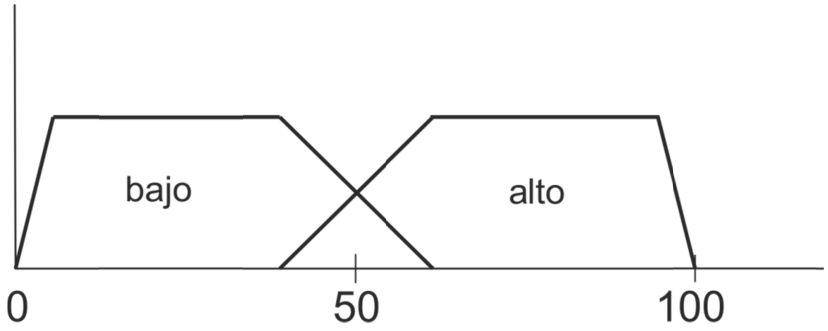


Figura 1.11. Funciones de pertenencia para las variable calefacción y aire acondicionado

Estas funciones indican la probabilidad de que una variable pertenezca a un conjunto dado. Generalmente, las funciones de pertenencia tienen forma triangular o trapezoidal.

Una vez definidos los conjuntos difusos y las funciones de pertenencia, se define un grupo de reglas difusas que gobernarán el sistema:

```
1. Si temperatura es muy fría entonces
      calefacción es alto
2. Si temperatura es poco fría entonces
      calefacción es bajo
3. Si temperatura es adecuada entonces
      no calentar
      no enfriar
4. Si temperatura es poco calor entonces
      aire acondicionado es bajo
5. Si temperatura es mucho calor entonces
      aire acondicionado es alto
```

Como último paso, hay que definir el *proceso de inferencia*, mediante el cual se obtiene la salida del sistema fuzzy. El proceso de inferencia consiste en determinar el

grado de pertenencia de las variables de salida a un conjunto dado, aplicando las reglas anteriormente definidas. En este caso, el proceso de inferencia retorna los valores de control para el aire acondicionado y el sistema de calefacción. Para llevar a cabo esta tarea, se aplica un *operador fuzzy* a los grados de pertenencia de las variables de entrada que se han usado en los antecedentes del conjunto de reglas. Esto da como resultado el grado de pertenencia de la variable de salida para esa regla.

Una vez concluido el proceso de inferencia, se obtiene un conjunto de grados de pertenencia para cada variable de salida. Sin embargo, en última instancia hay que transformar esta información a valores numéricos que puedan ser utilizados en un sistema físico. Esta transformación a valores numéricos recibe el nombre de *defuzzyficación*. El método más empleado es el del centroide, que consiste en multiplicar los valores de grados de pertenencia por las funciones de pertenencia, sumar las figuras geométricas resultantes y finalmente hallar el centroide de esta figura resultante.

En resumen, cualquier diseño de sistema fuzzy conlleva los siguientes pasos:

1. Definir las variables de interés y sus conjuntos fuzzy
2. Definir las reglas fuzzy que resuelven el problema
3. Aplicar el proceso de inferencia a los datos de las variables de entrada
4. Aplicar el método de defuzzyficación que permita obtener un valor concreto para las variables de salida.

1.3.2 Tipos de sistemas fuzzy

Según la clasificación de Mitra y Hayashi [MIT00], los sistemas fuzzy se pueden englobar en dos grandes categorías:

La primera categoría está formada por sistemas tipo Mamdani. Este tipo de sistemas se compone de modelos lingüísticos basados en conjuntos de reglas *if-then*, donde los antecedentes y consecuentes emplean variables difusas. Su comportamiento se puede expresar en lenguaje natural. En este tipo de sistemas fuzzy, el conocimiento se representa mediante un conjunto de reglas del tipo:

$$R^i: If\ x_1\ is\ A_1^i\ and\ x_2\ is\ A_2^i \ldots and\ x_n is\ A_m^i, then\ y^i\ is\ B^i \qquad \textbf{(1.17)}$$

donde $R^i(i = 1,2, \ldots, l)$ es la i-ésima regla, $x_j(J = 1,2, \ldots, n)$ es la entrada, y^i es la salida de la regla R^i, y $A_1^i, A_2^i, \ldots A_m^i, y\ B^i(i = 1,2, \ldots, l)$ son funciones de pertenencia normalmente asociadas a términos lingüísticos.

La segunda categoría, basada en sistemas tipo Sugeno, usa reglas con un antecedente fuzzy y un consecuente funcional:

$$R^i: If\ x_1\ is\ A_1^i\ and\ x_2\ is\ A_2^i \ldots and\ x_n is\ A_m^i, then\ y^i = a_0^i + a_1^i x_1 + \cdots + a_n^i x_n \quad \textbf{(1.18)}$$

Este enfoque aproxima una función no lineal con una función lineal a trozos, descomponiendo todo el espacio de entrada en varios espacios fuzzy y representando cada uno de ellos con una función lineal. Está demostrado [BUC99] que este modelo es capaz de aproximar cualquier función real continua hasta el grado de precisión deseado para un conjunto compacto de números reales. Este tipo de representación, por el contrario, no permite que las variables de salida sean descritas mediante lenguaje natural.

2 SISTEMAS NEURO-FUZZY

Este capítulo comienza describiendo el concepto de sistema neuro-fuzzy, así como las motivaciones del diseño de estos sistemas y sus capacidades. A continuación se describen los dos tipos principales de sistemas neuro-fuzzy existentes, describiendo más en profundidad el tipo de sistema que será usado en esta tesis mediante dos ejemplos, siendo uno de ellos el sistema de inferencia fuzzy basado en una red adaptativa.

2.1 Introducción

Tanto las RNA, como la lógica difusa, son dos herramientas fundamentales en el ámbito del *soft computing*. En este tipo de computación se trata de dar respuesta a aquellos problemas demasiado complejos para modelarlos con técnicas tradicionales. Muchas veces no se busca una solución totalmente precisa a esta clase de problemas, sino que se intenta obtener una solución aceptable a un coste razonable.

Como se ha visto en el capítulo anterior, las redes neuronales tienen la ventaja de poder aprender a resolver un problema sin tener un modelo matemático del mismo, simplemente viendo las entradas y salidas esperadas. Una RNA aprende y almacena su conocimiento de forma distribuida mediante la adaptación de los pesos sinápticos durante el proceso de entrenamiento, y es aquí donde reside su mayor virtud y a la vez su desventaja: dado que la red aprende de forma autónoma y su conocimiento está distribuido, no hay forma de expresar en un lenguaje comprensible el proceso que sigue para tomar una decisión. En definitiva, se comporta como una *caja negra*. Por otro lado, los sistemas fuzzy destacan en su capacidad para manejar conceptos expresados en lenguaje natural, pero necesitan el conocimiento previo de un experto que sea capaz de definir con reglas el proceso que emplea para tomar una decisión.

Algunos autores como Hayashi y Buckley [HAY93] y [HAY94] o Jang y Sun [JAN93a], han demostrado que en muchos casos se puede aproximar una RNA a un sistema fuzzy y viceversa, mostrando comportamientos equivalentes. Parece lógico por tanto combinar ambas técnicas aprovechando las capacidades propias de la lógica difusa, como son el manejo de incertidumbre y la expresión en lenguaje natural, para

extraer un conjunto de reglas que represente el conocimiento adquirido por una red neuronal artificial que tiene capacidad para adquirir conocimiento por si sola.

2.2 Tipos de sistemas Neuro-Fuzzy

Existen dos grandes categorías en los esquemas híbridos neuro-fuzzy. Por un lado se encuentran los denominados *fuzzy-neural network* (FNN) y por otro los *neural-fuzzy systems* (NFS).

2.2.1 Fuzzy Neural Networks

Los esquemas FNN consisten en redes neuronales que poseen elementos de tipo fuzzy, ya sean sus entradas, pesos sinápticos, funciones de transición y/o salidas. Mitra y Hayashi distinguen cuatro métodos para el desarrollo de las FNN [MIT00]:

1. Incorporar elementos fuzzy en la red neuronal, ya sea fuzzificando los datos de entrada, asignando etiquetas fuzzy a las muestras de entrenamiento, fuzzificando el proceso de aprendizaje o expresando las salida de la red en términos de conjuntos fuzzy.

2. Cambiar las características básicas de las neuronas. En vez de aplicar una suma de productos como es usual, las neuronas se pueden modificar para realizar operaciones empleadas en la teoría de conjuntos fuzzy, como la unión, intersección, etc.

3. Usar medidas de incertidumbre para modelar el error de la red neuronal.

4. Hacer que cada neurona se comporte como un proceso fuzzy, de forma que sus entradas y salida sean conjuntos difusos,

2.2.2 Neural Fuzzy Systems

En los sistemas NFS, las redes neuronales se diseñan siguiendo el formalismo aplicado a la lógica difusa. Estas redes implementan la lógica de los sistemas de decisión fuzzy para poder descubrir, una vez entrenadas, las reglas que subyacen en el sistema en base a las entradas y salidas esperadas. En esta tesis nuestro interés se centrará en este tipo de sistemas neuro-fuzzy, ya que al existir una correspondencia entre sus elementos y los elementos de un sistema fuzzy, el tratamiento se simplifica notablemente. A continuación se describirán dos ejemplos ilustrativos de este tipo de sistemas.

2.2.2.1 Sistema neuro-fuzzy construido a partir de una red tipo CPN (counterpropagation network)

Como ejemplo de estos sistemas, se puede describir el desarrollado por Junhong Nie [NIE95], o el empleado por Chang y Chen [CHA01], donde se usa una red CPN para extraer un conjunto de reglas fuzzy de tipo Mamdani a partir de datos disponibles

de tipo numérico. El problema planteado es un sistema dinámico fuzzy multivariable para el que hay disponible un conjunto de datos de entrada, con sus correspondientes salidas esperadas. Debido a la dificultad y poca eficiencia a la hora de aplicar métodos basados en el modelo relacional, donde se intenta resolver la matriz relacional del sistema fuzzy por medio de ecuaciones, queda de manifiesto que es necesario buscar alternativas.

Una de estas alternativas es modelar el sistema fuzzy a través de una red CPN (counterpropagation network). El comportamiento del sistema fuzzy se puede describir de la siguiente forma:

$$y_k = f_k(x_1, x_2, \dots, x_Q) \tag{2.1}$$

donde:

- y_k es la salida k-ésima
- $x_1,\dots,x_Q$ es un conjunto de variables que toman los valores de todas y cada una de las entradas y salidas del sistema en los instantes anteriores
- f_k es la función que establece la relación entre las entradas y las salidas.

La relación de entradas y salidas descritas por (2.1) puede ser expresada con un conjunto de reglas IF-THEN de forma análoga a la descrita anteriormente en la ecuación (1.17):

$$R^j: If\ X_1\ is\ A_1^j\ and \dots and\ X_Q\ is\ A_Q^j, then\ Y_1\ is\ B_1^j\ and \dots and\ Y_m\ is\ B_m^j \tag{2.2}$$

donde:

- X_l e Y_k son variables fuzzy que corresponden a x_1 y a y_k
- A^j_l y B^j_k son conjuntos fuzzy

Para determinar el mapeo f_k se emplea una red de tipo CPN (Counter Propagation Network). La red se emplea como un extractor de conocimiento. El proceso de aprendizaje autoorganizativo trata de descubrir las reglas que relacionan las entradas con las salidas, a partir de las regularidades encontradas en los patrones de datos. Una vez generadas las reglas, el algoritmo de razonamiento fuzzy será el encargado de proporcionar la generalización.

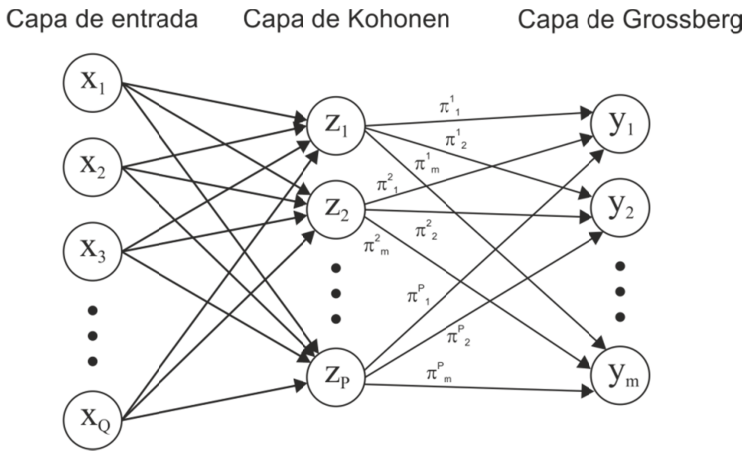


Figura 2.1. Counterpropagation network

Una red CPN está formada por tres capas: una capa de entrada con Q neuronas, una capa oculta de Kohonen con P neuronas y una capa de salida llamada capa de Grossberg formada por m neuronas. El objetivo de este tipo de red es aproximar funciones continuas del tipo $f: A \in R^Q \rightarrow R^m$ a través de un conjunto de muestras previo. El algoritmo de aprendizaje de la red CPN se divide en dos partes y es el siguiente:

- Entrenamiento de la capa de Kohonen:

1. En la capa de Kohonen se selecciona la unidad J ganadora empleando, por ejemplo, la distancia euclídea.
2. Se calcula la salida $z_j(t)$ de la capa de Kohonen empleando una función winner-take-all

$$z^j(t) = \begin{cases} 1 \; si \; j = J \\ 0 \; e.o.c. \end{cases}, j = 1 \dots P \tag{2.3}$$

3. Se actualizan los pesos de Kohonen

$$w^j(t) = w^j(t-1) + \propto_t \left[x^s(t) - w^j(t-1)\right] z^j \quad , j = 1 \dots P \tag{2.4}$$

donde $\propto_t \in (0,1)$ representa la tasa de entrenamiento, que decrece con el tiempo y x^s es un vector de muestra perteneciente al conjunto de entrenamiento.

- Una vez que la capa de Kohonen se estabiliza, se fijan los pesos w^j y comienza a aprender la capa de Grossberg en función de las salidas deseadas y^s ante un vector de peso w^j empleando la siguiente función de actualización:

$$\pi_k^j(t) = \pi_k^j(t-1) + \beta\left[y_k^s - \pi_k^j(t-1)\right] z^j \quad , \begin{matrix} j = 1 \dots P \\ k = 1 \dots m \end{matrix} \tag{2.5}$$

donde β es la constante que fija la velocidad de aprendizaje e y_k^s es la k-ésima componente de la muestra de entrenamiento y^s.

Cada nodo de la capa de Kohonen representa una regla. Las conexiones *w* entre las entradas y la capa de Kohonen representan la sentencia IF de las reglas. Las conexiones π entre la capa de Kohonen y la capa de Grossberg representan la cláusula THEN de las reglas fuzzy. Por lo tanto, cada regla del sistema fuzzy generado a partir de esta red será de la forma:

$$If\ x\ is\ w\ then\ y\ is\ \pi \tag{2.6}$$

2.2.2.2 Adaptive Network Based Fuzzy Inference System (ANFIS)

Otro ejemplo de sistema neuro-fuzzy de tipo NFS es el *Adaptive Network Based Fuzzy Inference System (ANFIS)*. Este sistema, desarrollado por Jang [JAN93b] es equivalente a un sistema fuzzy de tipo Sugeno. Está compuesto por una red feed-forward de cinco capas y tiene dos tipos de neuronas: uno cuya función de neurona es fija (representadas con forma circular en la figura 2.2), y otro cuya función de neurona es variable en función de ciertos parámetros (representadas con forma rectangular en la figura 2.2). Todas las neuronas de una misma capa son del mismo tipo. En la siguiente figura se puede observar la estructura de un sistema ANFIS de solo dos entradas con el fin de simplificar la descripción del esquema.

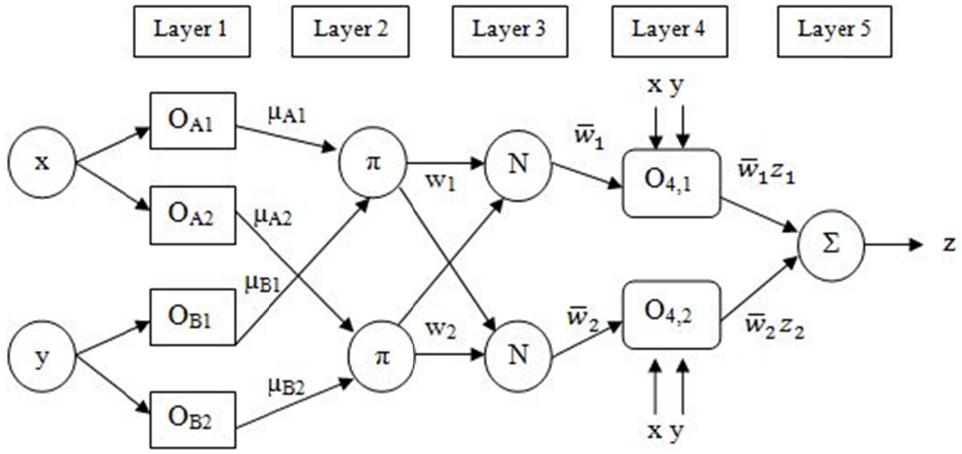


Figura 2.2. Sistema ANFIS de dos entradas

La función de cada capa es la siguiente:

- *Capa 1*: Las entradas se corresponden con las entradas al sistema (*x* e *y*) y la salida es el grado de pertenencia de cada variable al conjunto difuso A_i asociado al nodo. Las funciones de pertenencia son del tipo:

$$O_{Ai} = \mu_{Ai}(x) \tag{2.7}$$

$$O_{Bi} = \mu_{Bi}(y) \tag{2.8}$$

$i = 1, \dots, n$ *n*:número de funciones de pertenencia

Los parámetros variables de esta capa corresponden a los parámetros de las funciones de pertenencia de los conjuntos fuzzy, conocidos como parámetros del antecedente.

- *Capa 2*: Cada nodo calcula el grado de activación de la regla asociada al mismo. Estos nodos, conocidos como *nodos de reglas*, calculan el grado de activación de la regla mediante el producto de las señales de entrada (salida de los nodos de la capa 1).

$$O_{2,i} w_i = \mu_{Ai}(x)\mu_{Bi}(y), \qquad i = 1, \dots, n \tag{2.9}$$

- *Capa 3*: Cada nodo de la capa 3 calcula el grado de activación normalizado respecto a la suma de los grados de activación de la regla i.

$$O_{3,i} = \overline{w_i} = \frac{w_i}{\sum_{i=1}^{n} w_i}, \qquad i = 1, \dots, n \tag{2.10}$$

Cada nodo se corresponde con una de las reglas que se han introducido en el sistema, igual que los nodos de la capa anterior.

- *Capa 4*: La salida de los nodos se corresponde con el producto entre el grado de activación normalizado (salida de la capa 3) por la salida individual de cada regla (calculada por los nodos de esta capa).

$$O_{4,i} = \overline{w_i} z_i = \overline{w_i}(p_i x + q_i y + r_i), \qquad i = 1, \dots, n \tag{2.11}$$

p_i, q_i, r_i forman el conjunto de parámetros conocidos como *parámetros del consecuente* y son los coeficientes de las funciones lineales que forman el consecuente de las reglas. Son parámetros ajustables, como los de la capa 1.

- *Capa 5*: Tiene un único nodo que calcula la salida total del sistema (agregación) como la suma de todas sus señales de entradas individuales.

$$O_{5,i} = z = \sum_i \overline{w_i} z_i\,, \qquad i = 1, \dots, n \tag{2.12}$$

En el modelo ANFIS, al igual que en el caso anterior, existen dos conjuntos de parámetros a entrenar. Por un lado están los parámetros del antecedente, que definen las funciones de pertenencia y por otro lado están los parámetros del consecuente, que corresponden a los coeficientes de las funciones lineales del consecuente de las reglas.

Los algoritmos utilizados para optimizar los parámetros del antecedente suelen ser de gradiente descendente. Para los parámetros del consecuente sin embargo, los más empleados son de mínimos cuadrados. Debido a que el aprendizaje usa estos dos tipos de algoritmos para el entrenamiento, se denomina “aprendizaje híbrido”.

El ajuste de los parámetros de las funciones se lleva a cabo calculando la suma de los errores cuadráticos medios entre la salida deseada y la salida de la red. El gradiente de esta función de error se emplea para minimizar el propio error, ya que indica hacia donde se produce el mayor incremento del mismo. Para reducir el valor de error, los parámetros del antecedente se actualizan de manera iterativa. Esta actualización consiste en la substracción de un término que viene dado por el gradiente multiplicado por un cierto factor. A este factor se le conoce como *factor o velocidad de aprendizaje*. Mediante este procedimiento se van ajustando sucesivamente los parámetros del antecedente de forma que se va reduciendo el error producido por el sistema ANFIS.

Para poder aplicar este sistema de aprendizaje simultáneamente a todos los parámetros de la red, en cada época de entrenamiento se debe ejecutar un paso forward y otro backward:

- *Paso forward:* en la primera iteración se inicializan los parámetros de las funciones de pertenencia (capa 1) y se presenta un vector de entrada-salida, se calculan las salidas del nodo para cada capa de la red y se obtienen los parámetros del consecuente (capa 4) usando el método de mínimos cuadrados. Con los parámetros del consecuente identificados se calcula el error como la diferencia entre la salida obtenida en la red y la salida deseada (que se presentó en los pares de entrenamiento). Para medir el error de entrenamiento suele emplearse la suma de errores cuadráticos.

- *Paso backward:* se propagan las señales de error desde la salida, en dirección hacia las entradas. Para cada dato de entrenamiento se acumula el vector gradiente. Al final de este paso para todos los datos de entrenamiento, los parámetros en la capa 1 (parámetros de las funciones de pertenencia) se actualizan por el método gradiente descendente en el que, como se ha explicado anteriormente, se multiplica cada parámetro por un valor de velocidad de aprendizaje que es proporcional a la derivada del error respecto a ese parámetro (gradiente descendente).

Las reglas generadas mediante esta red, una vez entrenada, tendrán la forma:

$$If\ x\ is\ A_1 and\ y\ is\ B_1\ then\ z_1 = p_1x + q_1y + r_1 \quad \textbf{(2.13)}$$

$$If\ x\ is\ A_2 and\ y\ is\ B_2\ then\ z_2 = p_2x + q_2y + r_2$$

...

3 SISTEMAS DE DETERMINACIÓN DE VARIABLES CINEMÁTICAS

Este capítulo pretende dar una visión general de los sistemas empleados para medir variables cinemáticas como la velocidad, aceleración y orientación. Se comienza describiendo los tipos más usuales de acelerómetros y giroscopios, así como sus características más notables. Estos elementos están habitualmente presentes en sistemas empleados para la medición de vibraciones o la estimación de la posición de un cuerpo. A continuación se describen los fundamentos del algoritmo de Lucas-Kanade como método para medir el movimiento a partir de imágenes de vídeo.

3.1 Dispositivos de medidas inerciales

3.1.1 Acelerómetros

Un *acelerómetro* es un dispositivo destinado a medir la aceleración del cuerpo al que va unido. En su forma más simple está compuesto por una masa conocida (denominada masa sísmica) fijada a un dinamómetro que registra la fuerza ejercida sobre su eje al producirse el desplazamiento del sensor.

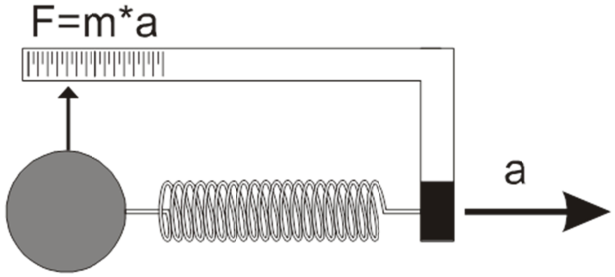


Figura 3.1. Acelerómetro mecánico

Conocida la masa y la fuerza registrada por el dinamómetro, y aplicando la segunda ley de Newton, es sencillo calcular la aceleración sufrida por el cuerpo al cual está fijado el acelerómetro.

Existen diversos tipos de acelerómetros atendiendo a la tecnología y diseño empleados en su fabricación. A continuación se enumeran algunos de los más empleados:

- *Acelerómetro mecánico*, como el descrito anteriormente. Este tipo de acelerómetros tienen la ventaja de su alta precisión en aceleraciones constantes, pero en cambio su coste es elevado y su respuesta lenta. Se suelen emplear en sistemas de navegación inercial.

- *Acelerómetro de efecto Hall*. Un acelerómetro de este tipo aprovecha el efecto Hall para medir el desplazamiento de la masa sísmica. El efecto Hall consiste en la aparición de un potencial eléctrico en un conductor por el que circula corriente, cuando este es atravesado por un campo magnético. Este potencial eléctrico se alinea en sentido transversal al de la circulación de la corriente.

  El acelerómetro de efecto Hall se compone de un sensor de este tipo fijado a la masa sísmica. Al producirse la aceleración, el sensor se desplaza dentro de un campo magnético y por tanto se produce una señal proporcional al desplazamiento de la masa.

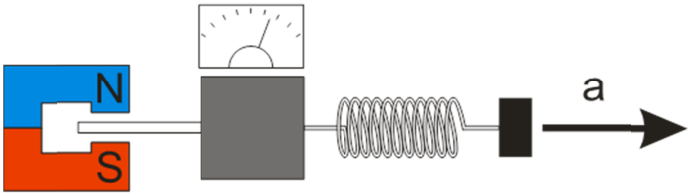


Figura 3.2. Acelerómetro de efecto Hall

- *Acelerómetros capacitivos*. En este caso, se varía la carga de un condensador fijando la masa a una de las placas que lo componen. Al moverse la masa se varía la distancia entre placas, produciendo variaciones en la carga. Otra forma de construcción consiste en colocar la masa entre ambas placas como si fuese un electrodo más del condensador. En este caso se forma un acelerómetro capacitivo diferencial.

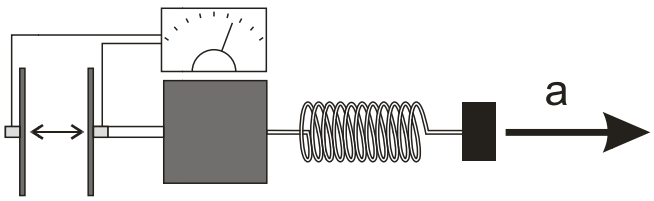


Figura 3.3. Acelerómetro capacitivo

Los acelerómetros capacitivos tienen la ventaja de poder medir aceleraciones continuas y poseen un alto grado de precisión. Su uso está muy difundido, abarcando desde sistemas industriales hasta sismógrafos.

- *Piezoeléctricos*. En los acelerómetros piezoeléctricos se aprovecha el hecho de que determinados materiales, al ser sometidos a presión o tensión, crean una carga eléctrica en su masa. En este tipo de acelerómetros, al producirse la aceleración, la masa sísmica presiona el cristal y se mide el potencial creado en sus extremos.

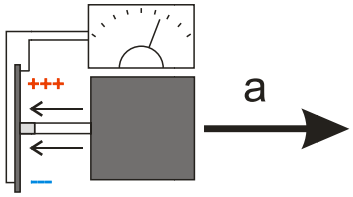


Figura 3.4. Acelerómetro piezoeléctrico

Estos acelerómetros habitualmente tienen una pobre respuesta en continua, y se utilizan sobre todo en medidas de vibraciones e impactos.

- *Piezoresistivos*. El funcionamiento de este tipo de acelerómetros es similar al de los piezoeléctricos, si bien en vez de usar un cristal piezoeléctrico se emplea un sustrato que varía su resistividad eléctrica dependiendo de la presión aplicada sobre el mismo. A diferencia de los piezoeléctricos, los acelerómetros piezoresistivos responden bien en continua, además suelen tener un alto grado de sensibilidad. Su aplicación es similar a los acelerómetros piezoeléctricos, es decir, pruebas de impacto, vibraciones, automoción…

- *Térmicos*. Están formados por una resistencia en un sustrato de silicio y dos termopares como sensores de temperatura a sus extremos (figura 3.5). Sobre el sustrato se deja una cavidad que alberga una burbuja de aire.

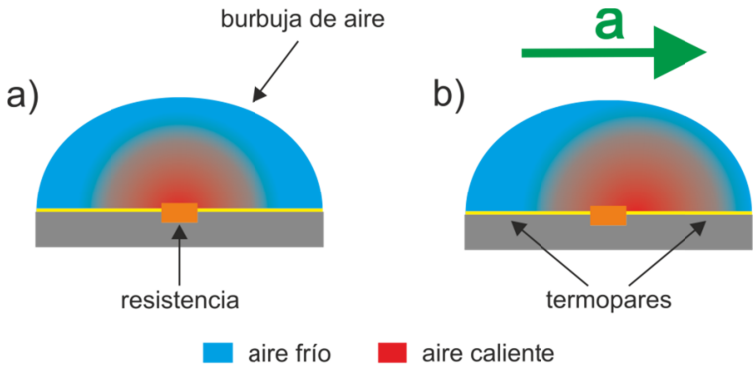


Figura 3.5. Acelerómetro térmico

Su principio de funcionamiento es muy distinto a los anteriores. Cuando se calienta la resistencia (figura 3.5a), en el interior de la burbuja se crea un diferencial de temperatura alrededor de la misma. Al producirse un desplazamiento (figura 3.5b), la masa de aire frío se desplaza en sentido opuesto al del movimiento, quedando registrado un diferencial de temperatura por los termopares. Este diferencial es función de la aceleración sufrida por el sensor.

La principal ventaja de estos acelerómetros es la ausencia total de partes móviles, eliminando problemas derivados de la inercia de la masa sísmica. Suelen estar fabricados mediante tecnología microelectromecánica.

3.1.2 Giróscopos

Un *giróscopo* es un dispositivo que nos permite medir la orientación de un cuerpo con respecto a un sistema de coordenadas, o bien el cambio de orientación por unidad de tiempo del mismo, es decir, su velocidad angular. Al igual que los acelerómetros, existen multitud de tecnologías y diseños en la fabricación de estos dispositivos. Como muestra, los más destacados son los siguientes:

a) *Mecánicos*. Consisten en una rueda que gira a gran velocidad, suspendida en el interior de dos cardanes. Si se producen variaciones de rotación en los cardanes, la rueda permanecerá fija debido al principio de conservación del momento angular. Midiendo la rotación de los cardanes con un sensor rotatorio se puede conocer la inclinación con respecto a la rueda giratoria, ya que esta permanece fija con respecto al sistema global de coordenadas. Este tipo de giróscopo mide directamente los ángulos de orientación del cuerpo con respecto a los ejes de referencia.

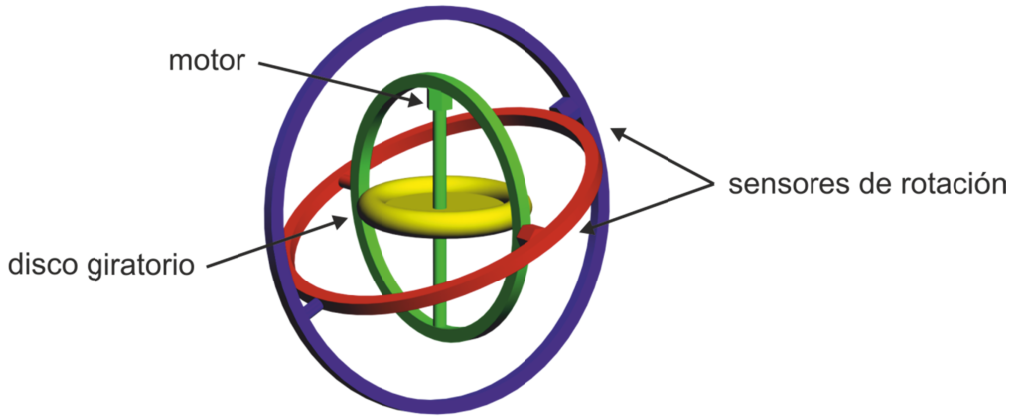


Figura 3.6. Giróscopo mecánico

b) *De fibra óptica*. En este caso se emplea una bobina de fibra óptica, donde se introducen dos haces de luz simultáneamente en direcciones opuestas. Al rotar la bobina en su eje, se producen cambios en el camino recorrido por cada haz de luz, produciéndose un desfase entre ambos haces que puede ser medido por un espectroscopio.

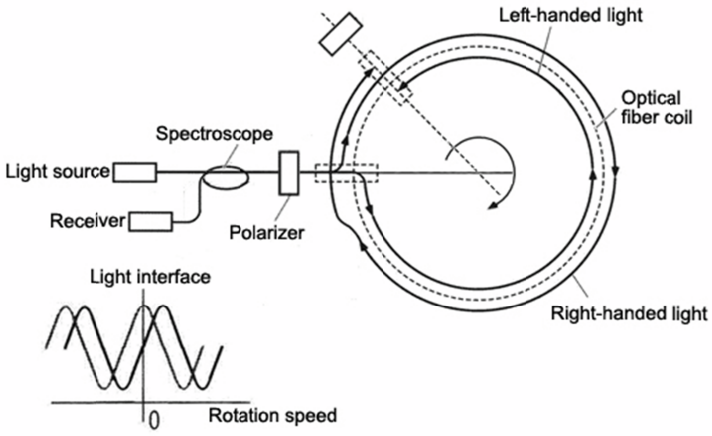


Figura 3.7. Giróscopo de fibra óptica

c) *De estructura vibratoria de Coriolis*. Este tipo de giróscopo mide la velocidad angular. Se compone de una masa que vibra en el plano de rotación a medir (figura 3.8a, flecha verde).

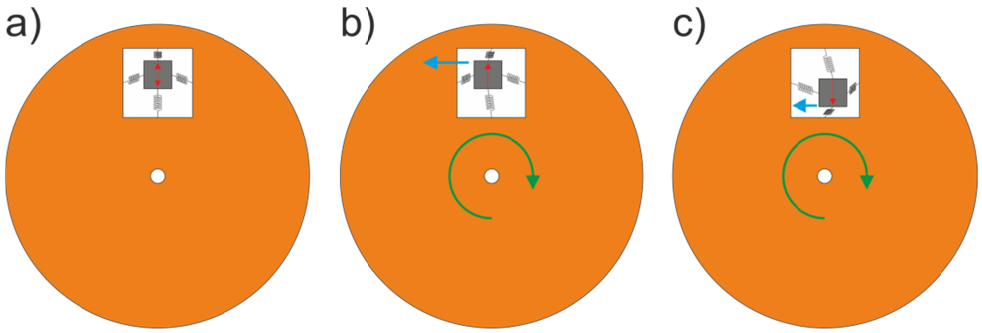


Figura 3.8. Giróscopo de estructura vibratoria

Al producirse una rotación, debido al efecto Coriolis, aparece una oscilación secundaria (figura 3.8b, flecha azul) en la cual la masa se ve sometida a una fuerza mayor cuando se encuentra en la parte más alejada del eje y a una menor fuerza cuando está más cerca de este (figura 3.8c, flecha azul). Esta oscilación secundaria es perpendicular a la primera pero en el mismo plano. Por tanto, si la masa vibratoria y el sustrato que la rodea forman un condensador, la carga del mismo variará en función de la rotación a la que sea sometido el sistema.

3.1.3 Sistemas microelectromecánicos (MEMS – Microelectromechanical system)

La *microelectromecánica* consiste en "la integración de elementos electrónicos y mecánicos en un sustrato de silicio empleando tecnología de microfabricación". Usando esta técnica se construyen sistemas mecánicos de una forma similar a como se fabrican circuitos integrados, empleando en muchos casos menos piezas que sus homólogos tradicionales. Los acelerómetros y giroscopios MEMS, si bien no son tan precisos como los tradicionales, tienen la ventaja de su simplicidad. Esta simplicidad se traduce en bajo

coste y robustez, así como capacidad de miniaturización. Los giróscopos mecánicos, por ejemplo, requieren de rodamientos de muy alta precisión y lubricantes especiales, lo que encarece su producción y no permite la miniaturización. Un giróscopo MEMS puede tener tan solo tres piezas y costar unos pocos euros. Los acelerómetros capacitivos y los giróscopos de estructura vibratoria son los más habituales en sistemas MEMS.

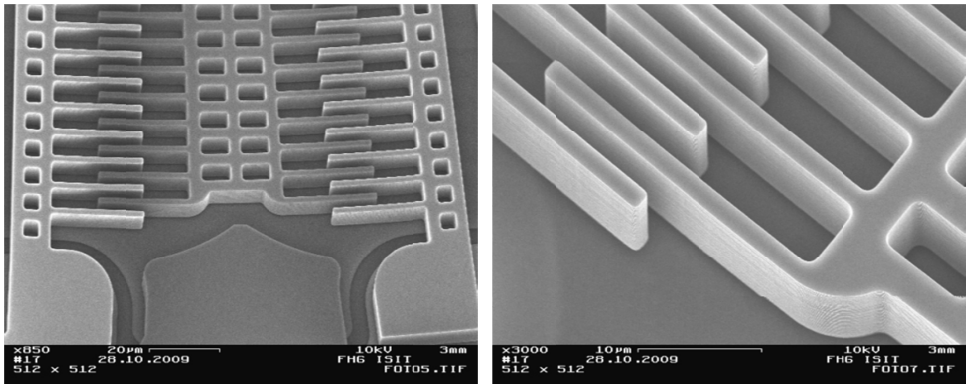


Figura 3.9. Imagen obtenida con microscopio de un acelerómetro capacitivo MEMS

3.1.4 Descripción de los sistemas de medida inercial utilizados en esta tesis

3.1.4.1 El acelerómetro piezoeléctrico Brüel & Kjaer 4383

El acelerómetro Brüel & Kjaer 4383 es un acelerómetro piezoeléctrico de propósito general, válido para la medida y análisis de vibraciones. Mide 14mm de diámetro y 21.5mm de altura.

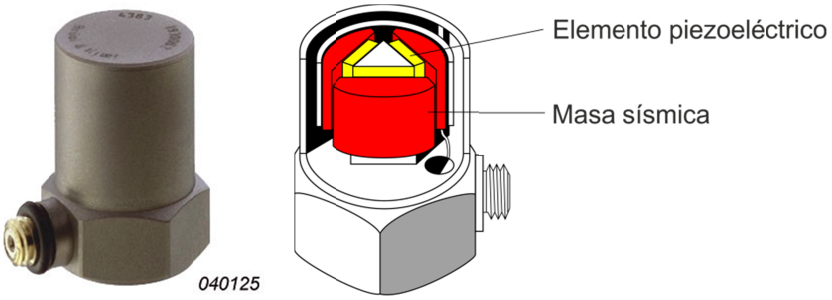


Figura 3.10. Acelerómetro Brüel & Kjaer 4383

Este diseño en concreto incluye tres elementos piezoeléctricos y tres masas dispuestas de forma triangular alrededor de un eje (figura 3.10). Esta disposición proporciona una serie de ventajas en comparación a otros diseños:

- Mejora de la relación entre masa y sensibilidad
- Aumenta la frecuencia de resonancia
- Proporciona un mayor aislamiento de las tensiones de la base

- Mejora la respuesta ante cambios de temperatura

Las especificaciones técnicas de este dispositivo se resumen en la siguiente tabla:

	Valor	Unidades
Frecuencia	0.1-8400	Hz
Temperatura	-74…250	ºC
Peso	17	gramos
Sensibilidad	31	pC/g
Ruido residual (RMS)	0.06	mg
Nivel de operación máximo	2000	g
Frecuencia de resonancia	28	Khz
Nivel máximo de choque	5000	g

Tabla 3.1. Especificaciones del sensor Brüel & Kjaer 4383

3.1.4.2 La unidad Mtx de Xsens Inc.

La unidad Mtx de Xsens Inc es un buen ejemplo de unidad inercial de plataforma analítica basada en sensores MEMS.

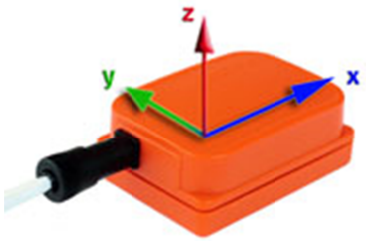


Figura 3.11. Unidad Mtx

En un encapsulado plástico de 38x53x21 mm (figura 3.11) se incluyen tres acelerómetros de tipo capacitivo y tres giróscopos de Coriolis dispuestos ortogonalmente, además de la electrónica de procesamiento de señales que realiza la integración. Como complemento a los acelerómetros y giróscopos, se incluye un sensor de temperatura y un magnetómetro 3D, destinados al refinamiento de las medidas mediante el empleo de fusión sensorial (figura 3.12).

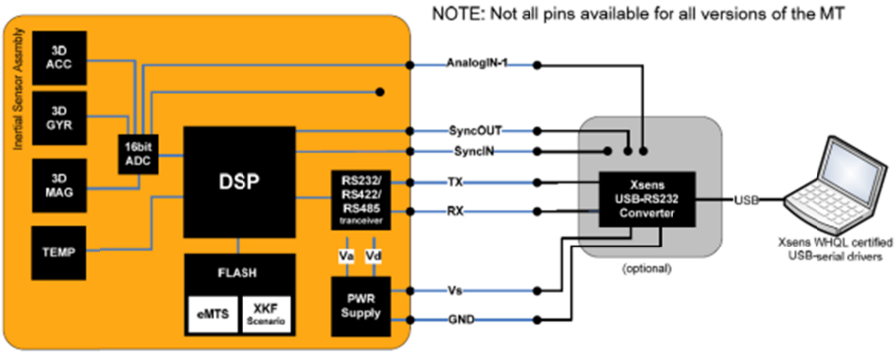


Figura 3.12. Esquema interno de la unidad Mtx

Para obtener la orientación absoluta con respecto al marco de referencia global, hay que integrar la lectura de los giróscopos. Para hacer esto, el fabricante proporciona un método de lectura a través del cual las medidas se integran fusionando los valores de los giroscopios con los valores de los magnetómetros y la medida del vector gravedad.

		Giro	Aceleración	Campo magnético	Temperatura
Unidad		[deg/s]	[m/s^2]	[mGauss]	[ºC]
Dimensiones		3 ejes	3 ejes	3 ejes	-
Fondo de escala	[unidades]	+/-300	+/-50	+/-750	-55…+125
Linearidad	[% de FE]	0.1	0.2	0.2	<1
Estabilidad	[unidades de 1σ]	1	0.02	0.1	0.5
Estabilidad del factor de escala	[% de 1σ]	-	0.03	0.5	-
Densidad de ruido	[unidades/ √Hz]	0.05	0.002	0.5	-
Error de alineamiento	[deg]	0.1	0.1	0.1	-
Ancho de banda	[Hz]	40	30	10	-
Resolución del A/D	[Bits]	16	16	16	12

Tabla 3.2. Especificaciones de la unidad Mtx

Para llevar a cabo la fusión sensorial se emplea un filtro de Kalman, en el que los valores de los magnetómetros se emplean en el filtrado de la rotación sobre el eje vertical y la medida del vector gravedad se usa para filtrar la inclinación en ambos ejes horizontales. Como resultado, se puede obtener una medida de la inclinación del sensor con un alto grado de precisión y con muy poca deriva a lo largo del tiempo. Los valores de aceleración lineal sin embargo, no son susceptibles de ser fusionados por falta de otra fuente de referencia.

En la tabla 3.2 se muestra un resumen de las especificaciones de la unidad Mtx.

3.2 Visión por computador

La *visión por computador o visión artificial* es una disciplina que trata el estudio y desarrollo de técnicas informatizadas para extraer información de una imagen o una serie de imágenes con el fin de resolver un problema concreto. Algunos campos de aplicación destacados dentro de esta disciplina son el reconocimiento de objetos, la restauración de imágenes, la reconstrucción tridimensional del entorno a partir de imágenes bidimensionales o la estimación de movimiento.

En la estimación de movimiento, una de las fuentes de información que se puede extraer de una secuencia de imágenes es el flujo óptico. El *flujo óptico* es el patrón de movimiento aparente en una secuencia de imágenes debido al movimiento relativo entre la cámara y su entorno.

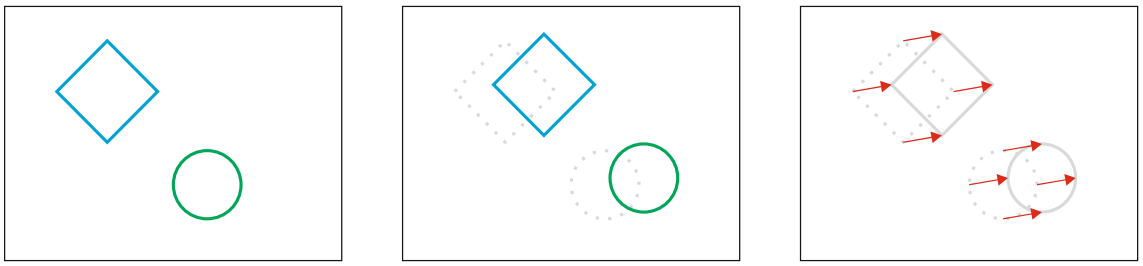


Figura 3.13. Flujo óptico producido por el movimiento de la cámara

Como se puede ver en la figura 3.13, si se hace un seguimiento de un conjunto de puntos característicos a lo largo de una secuencia de imágenes donde el dispositivo de captura se mueve con respecto a un entorno fijo, se puede definir un patrón de movimiento con vectores que marcan el flujo óptico o velocidad local. De la misma manera, este movimiento aparecerá si son los objetos los que se mueven en sentido opuesto.

Resulta obvio que para rescatar el flujo óptico de la imagen, se debe ser capaz de emparejar o registrar los puntos de una imagen con la siguiente. Para ello, uno de los métodos más conocido es el *Lucas-Kanade* [LKN81] que aún hoy sigue siendo uno de los más empleados debido a su robustez y relativo bajo coste computacional [BAR94].

3.2.1 Lucas-Kanade

El método de Lucas-Kanade [LKN81] es una solución al problema del registro de imágenes, esto es, averiguar el desplazamiento de una región de interés entre dos imágenes de un mismo escenario. Ambas imágenes pueden pertenecer a una secuencia en el tiempo o estar tomadas desde dos puntos distintos del espacio (p.ej. imágenes estereoscópicas). Se fundamenta en la idea de que para pequeños desplazamientos entre dos imágenes, este desplazamiento se puede aproximar a una derivada. En el caso de una dimensión se puede ilustrar de la siguiente manera:

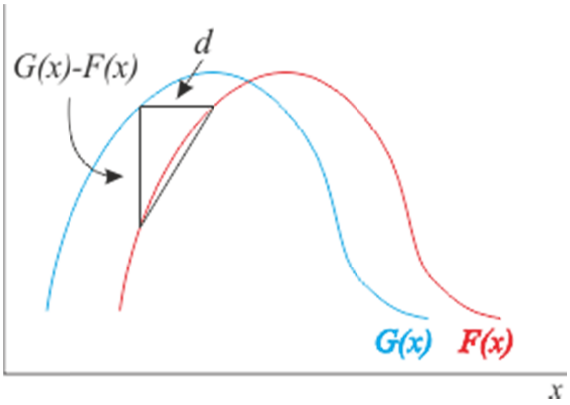


Figura 3.14. Aproximación del desplazamiento empleando la derivada

Como se puede observar, si *d* es suficientemente pequeño, podemos aproximar la derivada de *F(x)* de la siguiente forma:

$$F'(x) \approx \frac{F(x+d) - F(x)}{d} = \frac{G(x) - F(x)}{d} \tag{3.1}$$

y por lo tanto:

$$d \approx \frac{G(x) - F(x)}{F'(x)} \tag{3.2}$$

Esta expresión depende del pixel *x* que se esté evaluando. Para aumentar la fiabilidad y precisión del resultado se calcula la media ponderada para los distintos *x*. Esta ponderación se basa en minimizar aquellos resultados de *d* donde la función *F(x)* se aleje de su aproximación lineal. Para obtener ese factor *w(x)*, se emplea la segunda derivada de *F(x)*, que será tanto más pequeña cuanto más se aproxime a una función lineal.

$$F(x)'' \approx \frac{G'(x) - F'(x)}{d} \tag{3.3}$$

Si se invierte esta expresión, su valor será mayor cuanto más se aproxime a una función lineal. Además el factor *1/d* puede ser eliminado, ya que al ser constante no

afectará al comportamiento de la ponderación. Con todo, el factor *w(x)* queda expresado de la siguiente forma:

$$w(x) = \frac{1}{|G'(x) - F'(x)|} \tag{3.4}$$

Con lo que *d* queda finalmente:

$$d \approx \sum_x \left(\frac{w(x)[G(x) - F(x)]}{F'(x)} \right) / \sum_x w(x) \tag{3.5}$$

Una mejora del esquema de derivación anterior, que evita las divisiones por cero y además permite generalizar el método a más de una dimensión consiste en poner la ecuación (3.1) en la forma:

$$F(x + d) \approx F(x) + dF'(x) \tag{3.6}$$

A continuación se busca el valor *d* que minimiza la suma de diferencias al cuadrado entre las dos curvas *F(x+d)* y *G(x):*

$$E = \sum_x [F(x + d) - G(x)]^2 \tag{3.7}$$

Para encontrar el mínimo de esta función de error respecto a *d*, buscamos el valor que hace que su primera derivada sea cero:

$$0 = \frac{\partial E}{\partial d} \approx \frac{\partial}{\partial d} \sum_x [F(x) + dF'(x) - G(x)]^2 \tag{3.8}$$

Derivando respecto a *d* y despejando queda:

$$\sum_x 2F'(x)\left[F(x) + dF'^{(x)} - G(x)\right] \tag{3.9}$$

$$d \approx \frac{\sum_x F'(x)[G(x) - F(x)]}{\sum_x F'(x)^2} \tag{3.10}$$

Dentro de un esquema de Newton-Raphson y ponderando los valores en la misma forma que se ha indicado previamente, se obtiene la expresión final de la serie:

$$d_0 = 0,$$
$$d_{k+1} = d_k + \frac{\sum_x w(x)F'(x + h_k)[G(x) - F(x + h_k)]}{\sum_x w(x)F'(x + h_k)^2} \tag{3.11}$$

Como se ha mencionado anteriormente, el método de Lucas-Kanade permite encontrar correspondencias entre dos imágenes siempre y cuando los desplazamientos

sean muy pequeños. Cuando los vectores de desplazamientos son mayores, ya sea por un movimiento rápido de la cámara o por una combinación de movimientos entre cámara y objeto a seguir, es posible que el algoritmo no converja a una solución adecuada. Sin embargo, tal y como sugerían Lucas y Kanade en su artículo original [LKN81], suprimiendo las altas frecuencias espaciales, se aumenta el rango de convergencia. Además, una versión filtrada con un filtro paso-bajo se puede submuestrear sin perder información. Esto sin embargo provocaría que algunas características empleadas en el seguimiento quedasen difuminadas, por lo tanto, una aproximación grueso-fino (coarse-to-fine) parece lo más adecuado.

El uso de la visión como soporte a la estimación de variables cinemáticas está ampliamente difundido. Ejemplo de ello es el empleo de sistemas basados en visión artificial en el guiado de vehículos UAV (Unmanned Aerial Vehicle) como el presentado por Mondragón, Campoy, Olivares, Martínez y Mejías [MON10] donde se emplean distintos métodos de visión para el cálculo de rumbo, altitud, posición y movimiento de un helicóptero autoguiado o sistemas de odometría visual mucho más simples como el descrito en [CAM04]. En este último, se hace el cálculo de posición y orientación de un robot móvil empleando una simple webcam.

4 MANTENIMIENTO DE SISTEMAS MECÁNICOS

En este capítulo se muestran las nuevas tendencias en las políticas de mantenimiento de los sistemas mecánicos haciendo especial hincapié en las técnicas utilizadas para la detección automática de fallos. Concretamente el capítulo se centrará en la determinación de fallos en rodamientos. El orden de presentación en este capítulo se corresponde con el siguiente: Comenzamos haciendo un análisis global de las técnicas de mantenimiento utilizadas habitualmente y los recientes avances. Después nos centraremos en la exposición más detallada de los métodos empleados para la implementación de políticas de mantenimiento predictivo, indicando las diferentes fases de este tipo de tratamiento de fallos, así como las técnicas que muchos investigadores han utilizado. Posteriormente se hará un análisis más detallado de los diferentes tipos de rodamientos, focalizando el estudio hacia los posibles fallos que se pueden presentar, así como las condiciones en que estos se pueden dar. Por último, se abordarán los métodos utilizados para determinar el estado de los rodamientos y se indicará en qué contexto se enmarcan las investigaciones realizadas en esta tesis.

4.1 Introducción

En general los productos industriales cada vez son fabricados para ser más fiables y tener una mayor calidad. Sin embargo, estos productos, y en particular los sistemas mecánicos, se deterioran con el tiempo al estar sometidos a diferentes cargas. Este deterioro puede llegar a un estado en el cual la operación con estos sistemas mecánicos no es fiable y/o segura. Debido a este hecho se hace necesario establecer algún tipo de programa de mantenimiento. El tipo de mantenimiento más antiguo es el que se podría denominar mantenimiento sin planificación que simplemente espera a que se produzca la avería y actúa en ese momento. Dado que este tipo de mantenimiento resulta ineficiente, de hecho en componentes críticos resultaría en graves problemas de seguridad, otros planteamientos más usuales son los denominados programas de mantenimiento programado, que otros autores denominan mantenimiento preventivo. En este caso se establecen unos periodos acorde con las piezas objeto de mantenimiento, de tal forma que transcurrido ese periodo se realiza una revisión y se sustituyen las piezas según el periodo de tiempo prefijado para cada una de ellas. Este tipo de

mantenimiento comporta para muchas compañías un excesivo costo, en tanto en cuanto no toma en consideración el estado de deterioro de las piezas, sustituyendo en ocasiones piezas que todavía estarían operativas. Por otro lado el preestablecer periodos fijos de revisión contribuye en ocasiones a generar costos para las compañías innecesarios, dado que realmente en ese momento los sistemas mecánicos todavía pueden tener un cierto periodo adicional de tiempo en el que puedan funcionar correctamente. Más aun, en ocasiones se realizan revisiones completas de todos los sistemas cuando realmente en ciertos casos deberíamos centrarnos en alguno o algunos de ellos. Estos inconvenientes han llevado en estos últimos años a que se propongan cada vez más programas de mantenimiento basados en la monitorización on-line de los elementos mecánicos. A este tipo de mantenimiento se le conoce en la bibliografía especializada como mantenimiento basado en condición [PAU05][AND06]. En la terminología anglosajona se ha acuñado el término condition-based mantenience (CBM). Un aspecto clave de este tipo de programas de mantenimiento es que al considerar el estado de los sistemas mecánicos, el número de revisiones previstas se pueden reducir, con el consecuente ahorro económico.

4.2 Mantenimiento basado en las condiciones de los componentes del sistema

Este tipo de mantenimiento, de forma general, conlleva tres fases. En la primera fase se han de recoger datos que sean relevantes para la determinación de fallos. Posteriormente estos datos han de ser procesados de tal forma que nos permitan inferir si se ha producido un fallo y el tipo del mismo. Por último, este esquema conlleva el tomar decisiones de mantenimiento basadas en la información considerada en fases previas.

4.2.1 Adquisición de los datos

El primer paso consiste en la captura de los datos que posteriormente serán utilizados para determinar un posible fallo. En definitiva, el propósito es la adquisición de datos que puedan ser relevantes para este fin. Si bien existen datos relativos a que le ha sucedido a la máquina objeto de estudio y a que operaciones previas han sido acometidas, en nuestro caso nos centraremos en la medida de datos que sean relevantes en la determinación del estado de los componentes del sistema. Sobre todo desde el punto de vista de generar un sistema de detección de fallos que opere de forma automática. En este sentido lo más habitual es captar datos de vibraciones mecánicas, señales en el rango ultrasónico, análisis de los lubricantes, la temperatura, la presión, la humedad, etc. En este sentido hemos de precisar que el avance tecnológico respecto al diseño de sensores que permitan la adquisición de estas señales ha ampliado considerablemente su uso. Por su parte el avance en las tecnologías de comunicación, sobre todo en lo referente a tecnologías inalámbricas, han hecho más fácil su uso dentro de estos sistemas. En definitiva, estos avances han permitido abaratar el hardware y

software utilizados, lo que trae consigo una mayor implantación de este tipo de monitorización de fallos.

4.2.2 Tratamiento de los datos

Una vez se han obtenido los datos mediante un sistema de adquisición es necesario su tratamiento al objeto de determinar posibles fallos. En este sentido es importante determinar con qué tipo de datos vamos a operar. Inicialmente podemos distinguir tres tipos de datos en función de su estructura. Así un tipo de datos con el que nos encontraremos es el tipo valor. Concretamente se trata de un simple dato obtenido en un instante de tiempo específico. Como ejemplos podríamos indicar una temperatura, un valor de presión o humedad, etc. Por otro lado, existen datos que responden a un tipo de forma de onda. En este caso los datos responden a una serie temporal. Caso típicos lo constituyen las vibraciones. Por último, algunos autores hablan de datos de tipo multidimensional. Es el caso cuando se capturan datos en el mismo instante de tiempo pero los mismos tienen una estructura multidimensional. En este sentido podemos destacar como ejemplos las imágenes obtenidas con un termógrafo infrarrojo o las imágenes visuales.

El siguiente paso para diseñar un sistema de detección de fallos automática es extraer las características más relevantes de estos datos. A este proceso algunos autores lo conocen como extracción de características de los datos capturados directamente por los sensores. Por tanto estas señales han de ser procesadas. En el caso de valores multidimensionales como es el caso de imágenes el procedimiento de obtener información relevante se hace más complejo, de tal forma que no es tan inmediato tener a disposición la información necesaria para la detección de fallos. Este es uno de los motivos por el que las técnicas de diagnóstico y pronóstico son más usualmente utilizadas en el caso de datos en forma onda. En lo que prosigue nos centraremos en este tipo de datos.

Respecto al procesamiento de este tipo de datos existen numerosas técnicas de procesamiento. Por ello hemos de seleccionar la herramienta más apropiada según el caso concreto. En líneas generales estas técnicas se pueden agrupar en tres grandes grupos: el análisis en el dominio temporal, el análisis en el dominio frecuencial y el análisis en tiempo-frecuencia.

En el análisis en el dominio del tiempo, las técnicas habituales han sido el cálculo de características de la señal, tales como la media, los picos, el intervalo de pico a pico, la desviación estándar u otras variables estadísticas como la raíz cuadrática media, o el kurtosis. A estas características se les denomina habitualmente características en el domino del tiempo.

Si bien otros autores han hecho uso de estrategias más avanzadas basadas en la elaboración de modelos de series temporales. La idea básica en este caso ha sido el obtener un modelo mediante series temporales de tal forma que la forma de onda se

pueda corresponder a un modelo de serie temporal parametrizado y de esta manera poder extraer características utilizando este modelo paramétrico. Destaquemos como modelos ampliamente utilizados en la literatura los modelos autoregresivos (AR) o los modelos autoregresivos de media variable, conocidos como modelos ARMA. Así Zhang et al.[ZHA03] usó una representación de modelo basado en el espacio de estados de un modelo AR para analizar señales de vibración. Sin embargo, es importante apuntar que la aplicación de modelos AR o ARMA se hace difícil dada la complejidad en la modelización, sobre todo dado que necesitamos determinar el orden del modelo.

Otros autores han utilizado otras técnicas. Merece atención el uso de técnicas estadísticas multivariables conocidas como análisis de componentes principales (PCA) para el análisis de ondas de señal en el diagnóstico de fallos de engranajes.

Como se indicó en los párrafos previos otro de los tratamientos que se consideran en el caso de señales en forma de onda es el análisis en el dominio frecuencial. En este caso la base reside en tratar los datos como señal transformada en el dominio de las frecuencias. La principal ventaja que tiene este análisis respecto al análisis respecto al dominio temporal es el poder aislar e identificar fácilmente ciertas componentes frecuenciales de interés. El análisis más extendido ha sido el análisis espectral, basado en la aplicación de la transformada rápida de Fourier, conocida en la terminología inglesa como FFT (Fast Fourier Trasform). Generalmente se suele utilizar como herramienta de análisis espectral el espectro de frecuencia. Es decir, se toma el valor esperado de la transformada de Fourier multiplicada por su conjugada, tal como indica la siguiente formula.

$$P = E\left[X(f)X^{*}(f)\right] \tag{4.1}$$

Por su parte es de interés indicar que se suelen aplicar herramientas auxiliares sobre el espectro obtenido. Una de ellas es el análisis de envolvente que algunos autores denominan también demodulación de amplitud. En este sentido la transformada de Hilbert resulta una herramienta útil al objeto de obtener esta envolvente. De hecho dicha transformada ha sido utilizada por varios investigadores en diferentes aproximaciones encaminadas a la detección y diagnóstico de fallos en maquinaria [SHE04]. Esta transformada se ha utilizada en muchos casos para determinar la envolvente de la señal.

La transformada de Hilbert es útil en la determinación de atributos instantáneos de una serie temporal. De tal forma que la amplitud instantánea de una señal se puede calcular a partir del módulo de los números complejos obtenidos al aplicar la transformada de Hilbert. Asimismo, la frecuencia instantánea se puede calcular como la razón de cambio temporal de la fase de este número complejo. La transformada de Hilbert se puede calcular de la siguiente forma:

1) Se calcula la FFT de la señal temporal, remplazando aquellos coeficientes de la FFT que corresponden a frecuencias negativas por ceros.
2) Finalmente, se calcula la inversa de la FTT sobre esta nueva serie temporal.

Aunque el espectro de potencia es una de las técnicas de más amplia aceptación existen otras que pueden ser útiles en determinados casos. Así el Cepstrum que se define comúnmente como la transformada de Fourier inversa del logaritmo del espectro de potencia, tiene la capacidad de detectar armónicos y patrones de bandas laterales dentro del espectro de potencia.

Por último se ha de destacar el análisis de las señales en el tiempo y frecuencia. En el caso de señales de onda no estacionaria el análisis de frecuencia presenta limitaciones. Una de las alternativas es utilizar un análisis que involucre a la vez el tiempo y la frecuencia. Generalmente, el análisis de tiempo–frecuencia usa distribuciones de tiempo-frecuencia que representan la energía o potencia de la forma de onda como una función de dos variables, el tiempo y la frecuencia, con el objetivo de mostrar más precisamente patrones de fallo. Entre las técnicas que más se han utilizado podemos nombrar la transformada de Fourier de intervalo de tiempo corto, conocida como Short-time Fourier transform (STFT) o espectrograma (la potencia de las STFT) y la distribución de Wigner-Ville. En el caso de la primera técnica se trata de dividir la señal en varios segmentos con un intervalo de tiempo de ventana corto y aplicar la transformada de Fourier a cada segmento. En general, esta técnica resulta conveniente cuando las señales no oscilatorias tienen cambios lentos en su dinámica. Por su parte la distribución de Wigner-Ville no está basada en la segmentación y puede superar las limitaciones de la STFT. Sin embargo, al estar basada esta distribución en una transformada bilineal, esto conlleva ciertas interferencias de la transformación por si misma. Estos términos resultado de las interferencias hacen más difícil la interpretación de la distribución estimada. Otras transformadas tal es el caso de la distribución de Choi-Williams han sido diseñadas con el objetivo de evitar estos inconvenientes.

Otra transformada que se ha utilizado ampliamente para el análisis de tiempo-frecuencia es la transformada wavelet. A diferencia de las anteriores aproximaciones esta transformada es una representación de tiempo-escala de la señal. El análisis wavelet intenta expresar la señal en una serie de funciones de carácter oscilante con diferentes frecuencias a diferentes tiempos mediante dilataciones que se realizan por medio de un parámetro de escala y translaciones mediante un parámetro de tiempo. La transformada wavelet se ha aplicado con éxito en el análisis de datos en diagnóstico de fallos de engranajes, rodamientos y otros sistemas mecánicos.

En esta tesis nos centraremos en la representación de la señal en el dominio frecuencial en un intento por buscar características relevantes sin recurrir a un análisis más exhaustivo de las señales objeto de estudio. De hecho focalizaremos el estudio a los defectos de carácter incipiente producidos en rodamientos. En los siguientes apartados profundizaremos en la teoría existente en torno a este campo particular.

4.3 Rodamientos

En general hablamos de cojinetes de contacto, cojinetes de rodadura o rodamientos. Algunos autores los denominan cojinetes antifricción, destacando que estos cojinetes se conciben para su uso en ejes de transmisión, de tal forma que el movimiento entre el eje y la bancada de la máquina, se realiza mediante una serie de elementos intermedios que se mueven realizando un movimiento de rodadura, prácticamente sin ningún deslizamiento. De tal forma que en la práctica podemos indicar que la fricción que se produce es muy pequeña.

En la figura 4.1 presentamos un dibujo esquemático de un rodamiento de bolas. Tal como se puede apreciar un rodamiento de bolas típico está constituido por dos anillos concéntricos, denominados pista de rodadura interior (Inner Raceway) y pista de rodadura exterior (Outer Raceway). Adicionalmente existen unos elementos rodantes que se desplazan en torno a estas pistas. Los elementos rodantes pueden tener diferentes formas, siendo la más habitual la forma de bolas (Ball). Por otro lado, debemos especificar que estos elementos que se desplazan por las pistas lo hacen de forma guiada dentro de una especie de jaula (Cage) que cumple la misión de espaciar uniformemente los elementos rodantes, a la vez que prevenir su mutuo contacto. De esa forma se evita el deslizamiento entre los diferentes elementos rodantes. Dependiendo del elemento rodante que se utilice podemos hablar de rodamientos de bolas o de rodillos, diferenciando en este último caso entre rodillos cilíndricos y rodillos cónicos. En el caso de los rodillos cilíndricos, si estos poseen un diámetro lo suficientemente pequeño, a los mismos se les conoce con el nombre de cojinetes de agujas.

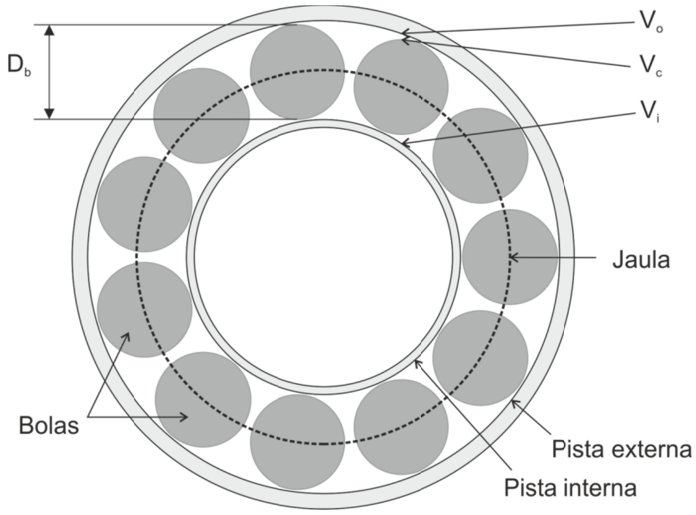


Figura 4.1. Estructura del rodamiento de bolas y variables características

Por otra parte, podemos hablar de rodamientos radiales o de empuje, según hayan sido diseñados para soportar carga radial o axial. Hemos de precisar que los cojinetes de bolas permiten soportar cierta carga axial, además de la carga radial para los que fueron concebidos, mientras que los cilíndricos no soportan carga de empuje. Los rodamientos cónicos ya se conciben desde un principio para soportar carga radial y

de empuje. En ocasiones, los rodamientos tienen varias pistas de rodadura paralelas, desplazándose por cada una de ellas los elementos rodantes correspondientes. En la figura 4.1, se muestra el caso de un rodamiento de bolas simple con una sola pista de rodadura, interna y externa.

Los rodamientos se diseñan para que se permitan sobre ellos diferentes movimientos. De hecho su dinámica se podría describir esencialmente como la combinación de cinco movimientos básicos. Cada uno de estos movimientos viene caracterizado por una frecuencia de respuesta, dado que responden a movimientos periódicos. Este hecho es el que hace que el funcionamiento de un rodamiento produzca vibraciones, cuyas características podrían ser extraídas mediante la técnica conocida como FFT (Fast Fourier transform). En esencia estas cinco frecuencias responden a:

1. La frecuencia de rotación del eje, que denominaremos *Fs*.
2. La frecuencia fundamental de la Jaula (Cage), que denotaremos como *Fc*.
3. La frecuencia de paso de las bolas por la pista interna, que denominaremos *FBFI* .
4. La frecuencia de paso de las bolas por la pista externa, que denominaremos *FBPO*.
5. La frecuencia rotacional de las bolas que denotaremos como *FB*

Estas frecuencias dependen de diferentes parámetros geométricos y cinemáticos asociados al rodamiento. Así destacamos, tal como se puede observar en la figura 4.1:

1. *Vi* la velocidad lineal de la pista de rodadura interior.
2. *Vc* la velocidad lineal del centro de la bola.
3. *Vo* la velocidad lineal de la pista de rodadura exterior.
4. *Db* el diámetro del elemento rodante, en este caso la bola.

Además dependen del diámetro de la jaula del rodamiento, medido desde un centro de bola al centro de bola opuesta y el ángulo de contacto del rodamiento.

4.3.1 Defectos en rodamientos

Los rodamientos son elementos que están sometidos a múltiples clases de esfuerzos que terminan provocando algún tipo de fallo en los mismos. Se podrían clasificar en una primera aproximación en: tensiones de carga estáticas y dinámicas, esfuerzos térmicos, vibraciones e impactos, condiciones ambientales de operación, corrientes eléctricas en el caso de encontrarse operando en motores eléctricos, etc. Mientras los factores indicados se encuentren dentro de los límites establecidos para cada uno de ellos, el rodamiento operará en condiciones normales, sin embargo, si

cualquier combinación de ellos excede estos límites, la duración de los rodamientos quedará drásticamente disminuida, provocando un fallo prematuro de los mismos que llevan consigo daños importantes en la maquinaria en la que están incorporados.

El fallo más común en un rodamiento es debido al efecto de la carga a la que se somete, debido a ello se van produciendo esfuerzos de contacto entre las pistas de rodadura y los elementos rodantes. De tal forma que se origina el fenómeno de las picaduras, que viene a producir el desprendimiento de pequeñas partículas de material, en forma de conos. Este fenómeno no es otro que el proceso de fatiga, debido a las tensiones de cortadura, presentes en zonas próximas a la superficie, originando grietas que terminan propagándose a las superficies, causando el desprendimiento de material. En la terminología inglesa se conoce con el nombre de flacking o spalling.

Generalmente este fenómeno viene acompañado de un aumento del ruido y vibraciones. En el apartado anterior, se mostró como en cualquier rodamiento normal se generan una serie de vibraciones, sin embargo, cuando el rodamiento entra en un proceso de fatiga, dichas vibraciones aumentan considerablemente. En el caso de rodamientos de bolas simples como el que se mostró en la figura 4.1, se puede indicar que la composición espectral de las vibraciones cambia. De hecho se puede hacer un análisis más detallado. En el caso de que los defectos se encuentren en la pista de rodadura, cada vez que la bola pasa golpea la zona defectuosa. De tal forma que se producirá una excitación relacionada con las frecuencias de paso de bola, es decir, FBFI o FBFO . Si el área de defecto es grande, se presentarán armónicos de estas frecuencias, lo que puede ser un indicativo de la gravedad del defecto. En el caso de que el defecto se produzca en el elemento rodante, la bola, generalmente se producen vibraciones con una frecuencia el doble de la frecuencia de rotación FB . Este fenómeno tiene que ver con el hecho de que la bola golpea la pista de rodadura interna y externa cada vez que rota sobre su propio eje. Al margen de las consideraciones realizadas, en la mayoría de los casos, lo habitual es que aparezcan fenómenos de modulación de estas frecuencias, tales como FBFI y FBFO, produciendo un espectro de la señal de vibración mucho más complejo. En ocasiones, si el área defectuosa de la bola es lo suficientemente grande, la frecuencia natural del sistema es excitada y modulada con el doble de la frecuencia de rotación.

Las consideraciones realizadas en el párrafo anterior nos llevan a considerar de manera lógica que una forma de observar el estado de los rodamientos es el uso de alguna herramienta de diagnóstico basada en el análisis de las vibraciones que se producen, en términos de composición espectral. La mayoría de herramientas de diagnóstico se basan en este análisis, siendo en estos momentos una línea de investigación en la comunidad científica, elaborar nuevas técnicas que permitan rentabilizar esta información desde punto de vista del mantenimiento predictivo de los rodamientos. A la vez que se producen vibraciones se produce ruido. Este ruido también sirve de base para otras herramientas de diagnóstico del estado del rodamiento. Este fallo no es de naturaleza catastrófica, la pieza puede seguir trabajando después de picada, sin embargo, se empeora el contacto. De forma genérica se entiende que se ha

producido un fallo cuando alguna de las superficies de contacto aparece dañada con área igual o superior a 6,5 mm^2.

Si el proceso de fatiga siguiese, dado que no se ha reemplazado el rodamiento, se comenzarán a producir cambios en las dimensiones críticas del mismo. Esto llevará al rodamiento a que se produzca más ruido, vibración, fricción, calor y desgaste, acompañado de más desprendimiento de material, de tal forma que el rodamiento comenzaría a rebasar los márgenes de seguridad, lo cual hace no prudente el proseguir operando la máquina. El paso final es el que se produzca un estado avanzado de desprendimiento de material, que acaba en un fallo catastrófico del rodamiento. Generalmente, los fabricantes tabulan la duración, fiabilidad y carga de los rodamientos. Así se suele hablar de duración de un rodamiento en términos de millones de revoluciones de operación.

4.4 Apoyo a la toma de decisiones

Previamente a la toma de decisiones, se hace necesario elaborar algoritmos que nos permitan pasar del espacio de características descrito en la sección 4.2.2 a lo que podríamos denominar espacio de fallos. Es decir, sería de interés buscar una correspondencia entre unas características concretas y un fallo determinado. A este proceso se le denomina genéricamente como reconocimiento de patrones. De hecho, tradicionalmente el personal de mantenimiento, a través del uso de herramientas tales como el gráfico del espectro de frecuencias de una señal de vibración u otra herramienta gráfica, es capaz de inferir el posible fallo. Sin embargo, estas operaciones requieren de un personal con alto nivel de especialización en el reconocimiento de estos fallos. Debido a ello resulta de gran interés el realizar este reconocimiento de patrones de forma automática. Dentro del marco de esta tesis nos centraremos en la determinación de fallos incipientes en rodamientos de bolas. En general, las diferentes aproximaciones al reconocimiento de fallos incipientes en rodamientos han girado en torno a aproximaciones basadas en métodos estadísticos y aproximaciones basadas en técnicas de inteligencia artificial.

En cuanto a las aproximaciones estadísticas, muchas de ellas se han basado en medir la desviación de la señal objeto de estudio con respecto a una señal de referencia correspondiente a un rodamiento normal, de tal forma que pudiésemos determinar si la señal está dentro de los límites deseables o fuera de estos. Por su parte, el análisis de cluster ha sido una técnica profusamente utilizada por la comunidad científica en este campo. Este tipo de análisis consiste en una clasificación estadística de tal forma que las señales se puedan agrupar en diferentes conjuntos de fallo en base a la similitud de sus características. De hecho, se busca minimizar la varianza dentro del mismo grupo y a la vez maximizar la varianza entre diferentes conjuntos. Se puede decir por tanto que el resultado de estas técnicas es la obtención de un cierto número de grupos heterogéneos con contenido homogéneo. En general, se suelen utilizar diferentes formas de medida de distancia o de similitud entre las señales. Así se han utilizado desde la distancia Euclidea, la distancia de Mahalanobis, la de Kullback-Leibler o la distancia Bayesiana.

En cuanto a las aproximaciones basadas en técnicas de inteligencia artificial, podemos citar los sistemas expertos o las redes neuronales, las cuales han sido utilizadas por muchos investigadores. Otros investigadores han utilizado también técnicas de lógica borrosa o algoritmos evolutivos. Sin embargo, recientemente, se vienen introduciendo los sistemas neuro-fuzzy como una clase de hibridación entre las redes neuronales y los sistemas borrosos. En cuanto a las redes neuronales se han utilizado generalmente métodos de entrenamiento basados en el denominado método del gradiente descendente, conocido en la terminología anglosajona como Backpropagation. Sin embargo, estas técnicas adolecen de dos importantes limitaciones. Por un lado la dificultad de determinar la estructura de la red para un problema concreto, así como el número de nodos, por otro, la lentitud en cuanto a la convergencia en los algoritmos de entrenamiento. Por su parte, los sistemas expertos tienen el hándicap de que a medida que aumenta el número de variables el número de reglas aumenta exponencialmente y consecuentemente el costo computacional. También es importante señalar que en la práctica, la monitorización de sistemas mecánicos conlleva la utilización de conocimiento con un cierto grado de inexactitud. Los expertos expresan sus conocimientos respecto a la detección de defectos introduciendo un cierto grado de incertidumbre. Este hecho ha potenciado que en estos últimos años los investigadores se hayan dirigido hacia técnicas basadas en lógica borrosa o lógica fuzzy. De esta forma, no se pierde la potencialidad de tener descrito el conocimiento mediante un conjunto de reglas, aunque sean reglas borrosas. Sin embargo, las redes neuronales, aunque en muchos casos arrojan resultados satisfactorios, adolecen de esta forma de expresar el resultado.

Si bien los sistemas borrosos tienen las ventajas señaladas previamente, en la mayoría de los casos resulta extremadamente difícil establecer las reglas borrosas que solucionan un problema particular de diagnóstico o pronóstico. Debido a ello, algunos investigadores empiezan a utilizar hibridaciones de las redes neuronales y los sistemas borrosos conocidos de manera general como sistemas neuro-fuzzy. En esta tesis serán estos sistemas en los que centraremos nuestro estudio.

5 ESQUEMAS NEURO-FUZZY APLICADOS A LA DETECCIÓN DE FALLOS

En este capítulo se describen las técnicas desarrolladas en esta tesis, aplicadas a la detección de fallos en rodamientos. Se comienza describiendo el montaje experimental sobre el que se han obtenido los datos, así como el procesamiento de los mismos para ser usados como entrada de un sistema neuro-fuzzy. A continuación se describe la aplicación del esquema ANFIS a los datos procesados y su resultado. Posteriormente se describe en profundidad un sistema neuro-fuzzy alternativo y tras aplicarlo a los datos obtenidos, se compara el comportamiento de ambas opciones discutiendo el resultado.

5.1 Introducción

Tal y como se ha indicado en el capítulo anterior, una de las técnicas más utilizadas para la determinación de defectos en piezas mecánicas es el análisis de las vibraciones que se generan cuando dichas piezas resultan defectuosas. Uno de los objetivos fundamentales de cualquier esquema de diagnóstico basado en el análisis de estas vibraciones es la determinación de las características que nos permiten predecir alguna clase de defecto incipiente. En este sentido, ya en el capítulo anterior se apuntaba que las técnicas denominadas inteligentes constituyen una herramienta de gran interés al para conseguir este objetivo, detectando los posibles defectos, así como la clasificación de los mismos. En este capítulo se plantea el uso de diferentes esquemas neuro-fuzzy para realizar esta tarea. Concretamente, el estudio se centrará en la determinación de diferentes defectos en rodamientos simples de bola y para ello se hará uso de la señal provista por el acelerómetro descrito en el apartado 3.1.4.1, que medirá la vibración objeto de estudio. En el siguiente apartado se describe el procedimiento experimental utilizado para tomar las señales de vibración asociadas a diferentes defectos.

5.2 Montaje experimental

En este trabajo se partió de un conjunto de datos obtenidos en el Laboratorio de Mecánica del Departamento de Mecánica de la UNED, fruto de la colaboración con los profesores Mariano Artés y Juan Carlos García Prada. En dicho laboratorio se utilizó la bancada que se muestra en la figura 5.1. Esta bancada consta de un eje soportado por dos rodamientos. En uno de los extremos del eje se sitúa un motor acoplado de forma elástica de 0.75 Kw. Este motor posee un regulador de velocidad electrónico que nos permite en todo momento controlar la velocidad del mismo. En el extremo opuesto del eje se sitúa una mordaza.

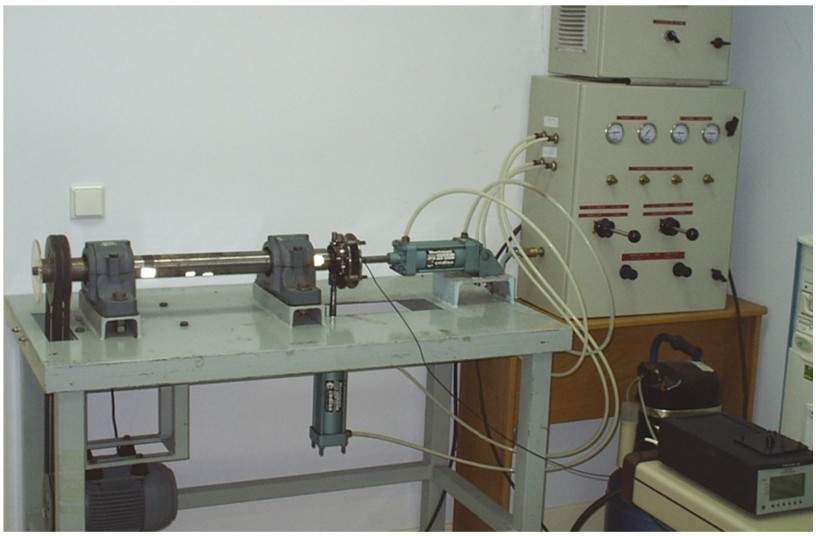


Figura 5.1. Bancada experimental

Esta mordaza se ha diseñado de tal forma que se puedan incorporar al eje diferentes rodamientos objeto de estudio. De hecho, adicionalmente se puede aplicar al rodamiento una carga axial y radial mediante un dispositivo de accionamiento neumático. De esta forma se puede hacer un estudio en condiciones de una cierta carga axial y radial tal como sucede en las aplicaciones habituales. En el estudio mostrado se utilizaron rodamientos FAG 7206B, un rodamiento simple de bolas. En cuanto a la velocidad de eje elegida esta fue de 1000 revoluciones por minuto, mientras que se aplicó una carga axial y radial de aproximadamente 2500 N.

5.2.1 Procesamiento de los datos

La presencia de fuerzas axiales o radiales en los ejes provoca que los rodamientos se conviertan en una fuente de vibración, aun estando estos en perfecto estado. Cuando un rodamiento sufre un desperfecto en alguno de sus componentes, esta vibración se incrementa, ya que el desperfecto provoca un impacto adicional en la mecánica. En este caso, la señal del acelerómetro se puede considerar una señal modulada en amplitud [BRA04], donde la portadora es la señal correspondiente a la agregación de las frecuencias de resonancia de los rodamientos y el resto de la mecánica y la señal moduladora es la producida por el desperfecto (figura 5.2 a).

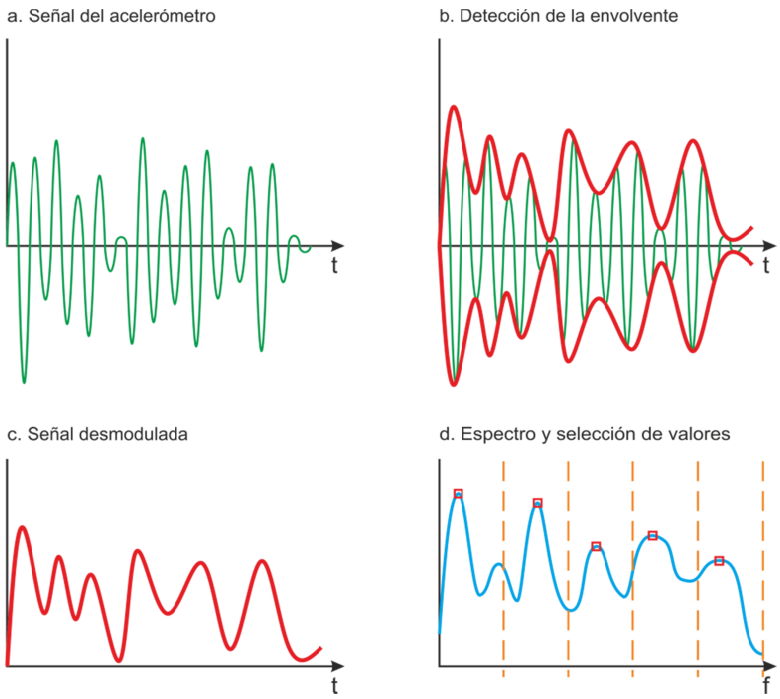


Figura 5.2. Procesado de la señal

Debido a este fenómeno, es necesario desmodular la señal del acelerómetro con el fin de rescatar únicamente las componentes correspondientes al desperfecto (figura 5.2 b y c). Para llevar a cabo esta tarea se usó la transformada de Hilbert [SHE07][SHE04]. Una vez que se dispuso de la señal desmodulada, se escogió el segmento desde los 30 hasta los 500Hz como el perteneciente a la señal generada por la avería en el rodamiento.

Para poder usar la señal como entrada a la red neuronal se le aplicó primero la transformada de Fourier. Con la señal en el dominio de la frecuencia, el espectro fue dividido en segmentos de igual ancho de banda y se escogió el máximo de cada segmento como valor representativo del mismo (figura 5.2d), obteniendo de esta manera los valores que conforman el vector de entrada al sistema neuro-fuzzy.

El conjunto de datos obtenido de esta manera se divide en cuatro grupos, cada uno con 10 elementos. El primer grupo corresponde a espectros de rodamientos en buen estado, el segundo está formado por espectros de rodamientos con fallos en las bolas, el tercero lo forman espectros de rodamientos con fallos en la pista interna y el cuarto está formado por los espectros de aquellos rodamientos que tienen fallos en la pista externa.

5.2.2 Aplicación del esquema ANFIS clásico

Como ya se ha visto en el apartado 2.2.2.2, el ANFIS es un sistema neuro-fuzzy con capacidad para aprender los parámetros de un sistema fuzzy de tipo Sugeno. Estos sistemas se caracterizan por tener en sus reglas un antecedente fuzzy y un consecuente funcional, lo que convierte al ANFIS en un buen candidato para resolver este tipo de problemas.

Como primera aproximación a la solución del problema se optó por emplear la implementación de Matlab del sistema ANFIS. Con el fin de simplificar el problema, el ensayo se llevó a cabo usando solo espectros correspondientes a rodamientos con fallos, con el objetivo de distinguir si el fallo era en pista o en bola. A su vez, estos espectros se dividieron en un conjunto de entrenamiento, formado por 18 espectros (6 con fallo en bola, 12 con fallo en pista), y un conjunto de test, formado por 12 espectros (4 con fallo en bola y 8 con fallo en pista). El sistema creado tenía una entrada por cada segmento del espectro de la señal de vibración y solo dos salidas. La salida 0 debería proporcionar un valor igual a 0 cuando se detecta un fallo en bola y un valor igual a 1 cuando es un fallo en pista. La salida 1 debería reflejar una salida complementaria a la 0, tal y como se resume en la tabla 5.1.

	Salida 0	**Salida 1**
Fallo en bola	0	1
Fallo en pista	1	0

Tabla 5.1. Respuesta esperada del sistema ANFIS

Se llevaron a cabo tres ensayos dividiendo los espectros en segmentos de 11, 35 y 55 frecuencias cada uno. Todos los ensayos fueron llevados a cabo en un ordenador Core2 Quad Q6600 a 2,4 Ghz con 4Gb de RAM.

En los dos primeros casos (11 y 35 frecuencias por segmento) no fue posible obtener un resultado debido al consumo de recursos por parte del algoritmo de entrenamiento, siendo esto advertido desde el propio entorno de Matlab en el momento de la ejecución del entrenamiento al crear las redes de 35 y 11 entradas respectivamente. En el tercer caso, empleando 55 frecuencias por segmento, y por tanto reduciendo el número de entradas a 7, se consiguió finalizar 1000 épocas de entrenamiento. Los resultados del ensayo con los patrones de test se muestran en la tabla 5.2.

Como se puede apreciar en la tabla 5.2, la respuesta del sistema ante los patrones 1, 5 y 12 no es adecuada. Además, el patrón nº 2 está extremadamente cerca de la frontera de decisión. Todo ello nos hace concluir que la capacidad de generalización de esta red no es adecuada. Probablemente su máxima limitación viene dada por la poca escalabilidad del esquema ante el número de entradas. Para conseguir un entrenamiento satisfactorio fue necesario reducir de forma drástica el número de entradas, lo que se traduce en una reducción notable de la información espectral manejada. Como existe una entrada por cada segmento del espectro, al haber menos segmentos, éstos deben ser de mayor anchura con lo que la representatividad de un solo valor dentro del segmento se ve reducida.

Nº de patrón	Fallo real	Salida 1 del sistema ANFIS (fallo de bola)	Salida 2 del sistema ANFIS (fallo de pista)
1	Fallo de bola	0.3785	0.6215
2	Fallo de bola	0.5001	0.4999
3	Fallo de bola	0.5786	0.4214
4	Fallo de bola	0.6217	0.3783
5	Fallo en pista interna	0.6676	0.3324
6	Fallo en pista interna	0.2828	0.7172
7	Fallo en pista interna	0.0290	0.9710
8	Fallo en pista interna	0.1061	0.8939
9	Fallo en pista externa	0.0004	1.0374
10	Fallo en pista externa	0.2454	0.7546
11	Fallo en pista externa	0.0000	0.8551
12	Fallo en pista externa	0.6379	0.3621

Tabla 5.2. Resultados del sistema ANFIS

5.3 Propuesta neuro-fuzzy alternativa

5.3.1 Estructura

Como alternativa al sistema ANFIS de Jang se ha empleado una modificación de la aproximación propuesta por Marichal et al. [MAR01], debido a su capacidad para el manejo de un alto número de entradas. Esta capacidad para el manejo de entradas permite usar un mayor número de bandas de frecuencia, concretamente 40, cada una con un ancho de 10 frecuencias, frente a las 55 frecuencias por banda que era necesario utilizar en el caso del ANFIS de Jang debido al alto consumo de recursos computacionales por parte de esta implementación.

Este sistema se plantea como una red de tres capas:

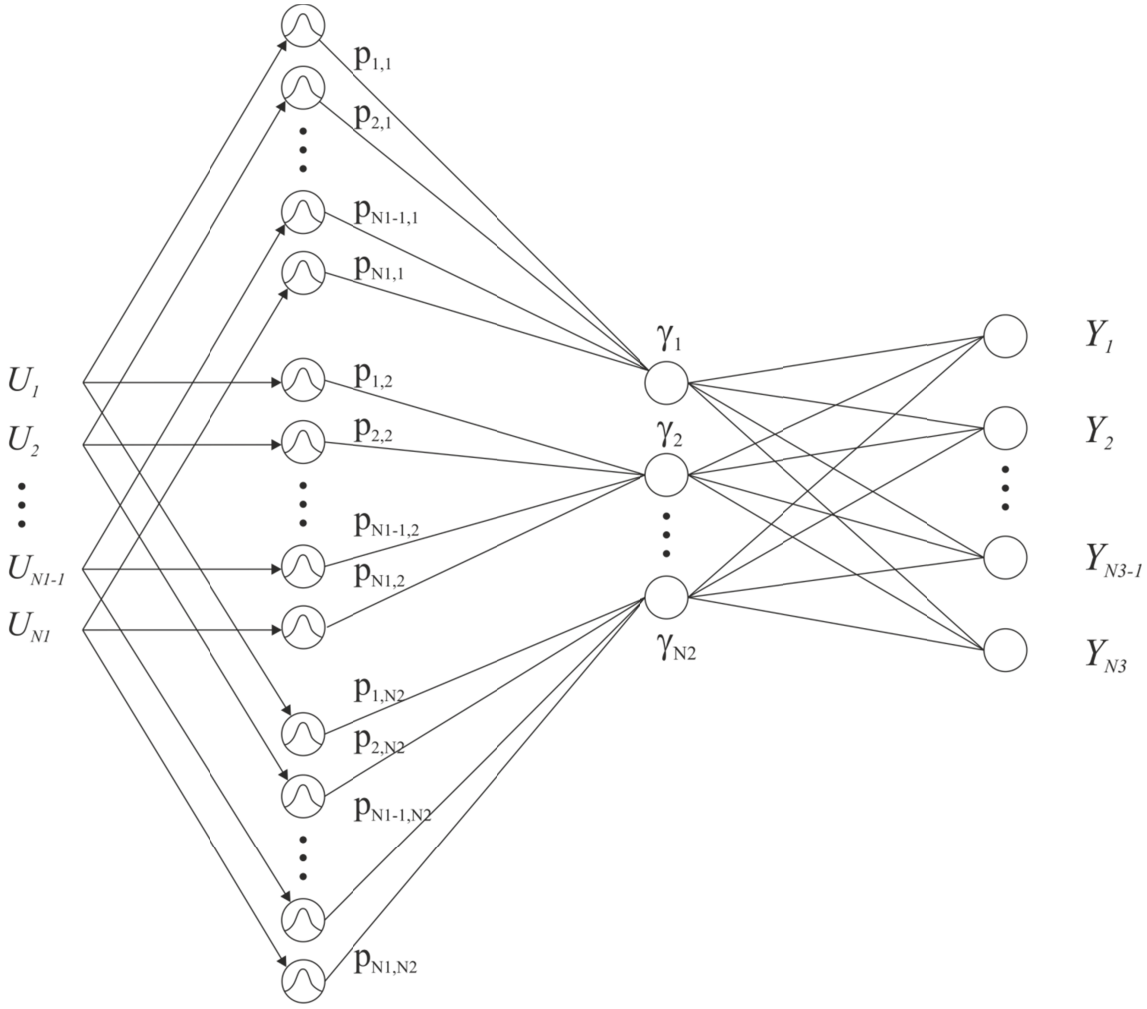


Figura 5.3. Sistema neuro-fuzzy aplicado

1. La primera capa está formada por neuronas de base radial cuyas entradas son las entradas del sistema neuro-fuzzy y sus salidas vienen dadas por la siguiente ecuación:

$$p_{ij} = e^{\left(-\frac{(U_i - m_{ij})^2}{\sigma_{ij}^2}\right)}, \begin{matrix} j = 1,2,\dots,N2 \\ i = 1,2,\dots,N1 \end{matrix} \qquad \textbf{(5.1)}$$

donde:

- *N1* es el número de entradas al sistema neuro-fuzzy y N2 es el número de neuronas de la capa oculta, que corresponde al número de reglas del sistema neuro-fuzzy.
- U_i es la i-ésima entrada al sistema neuro-fuzzy.
- m_{ij} es el centro de la función de pertenencia correspondiente a la entrada *i* y la neurona *j* de la capa oculta.

- σ_{ij} es el parámetro que indica la anchura de la campana de Gauss en la función de pertenencia correspondiente a la entrada *i* y la neurona *j* de la capa oculta.

- p_{ij} es la salida de la neurona de base radial, que indica el grado de pertenencia de la entrada i-ésima. Su valor será mayor cuanto más cerca esté U_i de m_{ij}.

  Esta capa de neuronas representa las funciones de pertenencia.

2. La segunda capa de neuronas representa las reglas del sistema. La salida de los nodos de esta capa se calcula según la expresión:

$$\gamma_j = min[p_{1j}, p_{2j}, \dots, p_{ij}, \dots, p_{N1j}], j = 1,2, \dots, N2 \quad \textbf{(5.2)}$$

   Esta capa indica el nivel de activación de cada regla, según las entradas al sistema, en virtud del grado de pertenencia de las entradas a dicha regla.

3. La tercera capa representa el proceso de defuzzyficación, proporcionando las salidas del sistema neuro-fuzzy. Está formada por una capa de neuronas lineales cuya salida viene dada por la siguiente expresión:

$$Y_k = \frac{\sum_j sv_{jk}\gamma_j}{\sum_j \gamma_j}, \begin{matrix} j = 1,2, \dots, N2 \\ k = 1,2, \dots, N3 \end{matrix} \quad \textbf{(5.3)}$$

   donde:

   - Y_k es el valor de la k-ésima salida del sistema neuro-fuzzy

   - sv_{jk} es el valor estimado que toman las reglas

5.3.2 Algoritmo de aprendizaje

Como se ha visto en el apartado anterior, el sistema neuro-fuzzy descrito depende de los siguientes parámetros: los centros de las funciones de pertenencia (m_{ij}), la anchura de las funciones de pertenencia (σ_{ij}) y los valores estimados de salida (sv_{jk}). Estos parámetros son determinados mediante el entrenamiento de la red del sistema neuro-fuzzy.

El algoritmo de entrenamiento de esta red tiene cuatro fases. En la primera se fija el parámetro $\sigma_{ij} = 1$ y se establecen los valores iniciales de los parámetros m_{ij} y sv_{jk}. En la segunda fase se optimiza el número de nodos de la capa oculta, que corresponde al número de reglas fuzzy. En la tercera fase se lleva a cabo un primer refinamiento de la anchura de las funciones de pertenencia (σ_{ij}). En la cuarta y última fase se ajustan definitivamente los diferentes parámetros del sistema neuro-fuzzy, tomando como valores iniciales los calculados en las anteriores etapas.

5.3.2.1 Primera fase

En la primera fase del entrenamiento se aplica un mapa de características de Kohonen para el cálculo inicial de los centros de las funciones de pertenencia y los valores de salida estimados. El número de vectores de pesos del mapa autoorganizativo es *N2*, y su dimensión es *N1 + N3* ya que los elementos de dicho vector de pesos corresponden a los centros de las funciones de pertenencia junto con las salidas esperadas.

$$W_j = \left(w_{1,j}, w_{2,j}, \dots, w_{N1,j}, w_{N1+1,j}, \dots, w_{N1+N3,j}\right) \qquad j = 1,2,\dots,N2 \tag{5.4}$$

Las entradas del mapa autoorganizativo serán por tanto:

$$V = (U_1, U_2, \dots, U_{N1}, YD_1, \dots, Y_{N3}) \tag{5.5}$$

donde $(U_1, U_2, \dots, U_{N1})$ es el vector de entrada del sistema neuro-fuzzy y $(YD_1, YD_2, \dots, YD_{N3})$ el vector de salidas deseadas.

Para determinar el nodo ganador y actualizar sus pesos en cada ciclo del algoritmo de Kohonen se aplica la siguiente regla:

$$\|W_x - V\|^2 = \min_j \left\|W_j - V\right\|^2, \quad \begin{matrix} j = 1,2,\dots,N2 \\ x = nodo\ ganador \end{matrix} \tag{5.6}$$

y los pesos se actualizan mediante el siguiente cálculo:

$$W_j(t+1) = W_j(t) + lr0 \cdot \left(\frac{mc - t}{mc}\right) \cdot exp\left(-\frac{(j-x)^2}{\sigma^2}\right) \cdot \left(V - W_j(t)\right) \qquad \begin{matrix} j = 1,2,\dots,N2 \\ t = 1,2,\dots,mc \end{matrix} \tag{5.7}$$

donde:

- σ es la varianza
- $W_j(t)$ es el j-ésimo vector para el instante t
- x es el nodo ganador
- $lr0$ es el índice de aprendizaje inicial
- mc es el número de ciclos de entrenamiento

Como se puede apreciar en la expresión (5.7), la actualización de los pesos del vector *W(t+1)* depende de tres términos que multiplican al índice de entrenamiento inicial (*lr0*):

- El término "$\left(\frac{mc-t}{mc}\right)$" hace que a medida que progresa el entrenamiento, la tasa de aprendizaje disminuya
- El término "$exp\left(-\frac{(j-x)^2}{\sigma^2}\right)$" hace que la tasa de aprendizaje disminuya cuanto más lejos está el nodo actual del nodo ganador. Define la vecindad del nodo ganador.
- El término "$\left(V - W_j(t)\right)$" hace que la tasa de aprendizaje sea menor cuanto más parecidos son el vector de entrada y el vector de pesos del nodo actual.

Después de haber presentado los pares de entrada-salida durante *mc* ciclos de entrenamiento, se da por concluida esta fase. Cada vector de pesos se asocia con una neurona de la capa oculta, ya que el vector contiene los centros de las funciones de pertenencia que relacionan un patrón de entrada con una o más reglas, así como la salida estimada de las mismas.

5.3.2.2 Segunda fase

Como se ha visto anteriormente, cada neurona de la capa oculta está relacionada con una regla del sistema neuro-fuzzy. En la primera fase del entrenamiento se consigue que una neurona de la capa oculta y las relacionadas con ésta, presenten una fuerte respuesta a determinados patrones de entrada. Sin embargo, es posible que dos nodos tengan una respuesta similar ante patrones similares, debido a que la distancia euclídea entre los pesos de ambos nodos es pequeña. En otras palabras, es posible que durante la primera fase se hayan originado nodos de reglas similares.

El objetivo de esta segunda fase es reducir el conjunto de nodos en la capa oculta, esto es, extraer un conjunto mínimo de reglas aplicando la distancia euclídea entre los pesos de los nodos, favoreciendo además la capacidad de generalización del sistema.

El algoritmo de optimización de reglas es el siguiente:

1. Para cada patrón *V* perteneciente al conjunto de entrenamiento utilizado en la etapa no supervisada:
   a. Calcular la distancia euclídea entre *V* y todos los W_j
   b. Escoger el nodo ganador como el más cercano a V y añadirlo al conjunto de nodos definitivos

$$W_w = \min_j \|W_j - V\|, \qquad j = 1,2,\dots,N2 \tag{5.8}$$

Una vez finalizado el algoritmo, los nodos definitivos representan el conjunto mínimo de reglas, pudiéndose asignar entonces los valores de las componentes del vector de pesos a los centros de las funciones de pertenencia y las salidas estimadas de la siguiente forma:

$$m_{ij} = w_{i,j}, \quad \begin{matrix} i = 1,2,\dots,N1 \\ j = 1,2,\dots,N2 \end{matrix} \tag{5.9}$$

$$sv_{jk} = w_{N1+k,j}, \quad \begin{matrix} j = 1,2,\dots,N2 \\ k = 1,2,\dots,N3 \end{matrix} \tag{5.10}$$

5.3.2.3 Tercera fase

En la tercera fase se hace un barrido del valor σ que había sido fijado a 1 en la primera etapa para todas las funciones de pertenencia. Este barrido, cuyos límites mínimo y máximo son escogidos por el operador, tiene el objetivo de encontrar el valor que mejor respuesta proporciona. Esta respuesta se evalúa presentando los patrones de entrenamiento y test a la red entrenada únicamente con el algoritmo no supervisado y calculando el error de su respuesta. El valor escogido para el parámetro σ es aquel que minimiza dicho error.

5.3.2.4 Cuarta fase

En esta última fase se refina el cálculo de los parámetros σ_{ij}, m_{ij} y sv_{jk} mediante un algoritmo de entrenamiento supervisado.

Dada la similitud entre una red de base radial y el sistema aquí descrito, para el entrenamiento se emplea el algoritmo LMS (*least mean-squares*). Este algoritmo se basa en la minimización de una función de error que consiste en una suma de cuadrados:

$$E = \frac{1}{2}\sum_{k=1}^{N3}(Y_k - \hat{y}_k)^2, \quad k = 1,\dots,N3 \tag{5.11}$$

donde $\hat{y}_k$ es la k-ésima salida deseada y Y_k es la k-ésima salida del sistema neuro-fuzzy.

Para determinar los parámetros del sistema se emplea de forma iterativa la siguiente expresión:

$$m(t+1) = m(t) - lr\frac{\partial E}{\partial m}, \quad t = 1,2,\dots,mf \tag{5.12}$$

donde *m(t)* es el valor del parámetro a determinar en la iteración *t*, *lr* es el índice de aprendizaje y *mf* el número de ciclos de entrenamiento.

Por tanto, para poder aplicar el algoritmo LMS, es necesario calcular las derivadas parciales de la función de error con respecto a cada parámetro. En este caso:

$$\frac{\partial E}{\partial sv_{jk}} = -(y_k - \hat{y}_k)\frac{\gamma_j}{\sum_j \gamma_j}, \quad \begin{matrix} j = 1, \dots, N2 \\ k = 1, \dots, N3 \end{matrix} \tag{5.13}$$

$$\frac{\partial E}{\partial m_{ij}} = -2\left[\sum_{k=1}^{N3}(y_k - \hat{y}_k)\cdot\frac{sv_{jk}\sum_{j=1}^{N2} p_{ij} - \sum_{j=1}^{N2} sv_{jk}p_{ij}}{\left(\sum_{j=1}^{N2} p_{ij}\right)^2}\right]\cdot \cdot p_{ij}\left[\frac{u_i - m_{ij}}{\sigma_{ij}^2}\right], \quad \begin{matrix} j = 1, \dots, N2 \\ k = 1, \dots, N3 \end{matrix} \tag{5.14}$$

$$\frac{\partial E}{\partial \sigma_{ij}} = -2\left[\sum_{k=1}^{N3}(y_k - \hat{y}_k)\cdot\frac{sv_{jk}\sum_{j=1}^{N2} p_{ij} - \sum_{j=1}^{N2} sv_{jk}p_{ij}}{\left(\sum_{j=1}^{N2} \gamma_{ij}\right)^2}\right]\cdot \cdot p_{ij}\left[\frac{\left(u_i - m_{ij}\right)^2}{\sigma_{ij}^3}\right], \quad \begin{matrix} j = 1, \dots, N2 \\ k = 1, \dots, N3 \end{matrix} \tag{5.15}$$

5.4 Aplicación de la aproximación Neuro-Fuzzy

Aplicando el esquema neuro-fuzzy descrito en el apartado anterior, se creó una red con 39 entradas y 2 salidas. Cada entrada del sistema corresponde al valor representativo de una banda en cada espectro patrón, estando estos previamente procesados según el método descrito en el apartado 5.2.1.

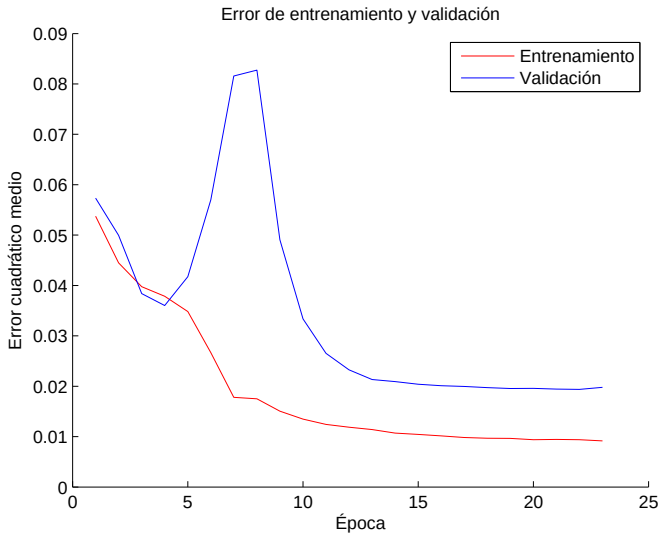


Figura 5.4. Progresión del error durante el entrenamiento

Esta red fue entrenada solamente con 23 ciclos de entrenamiento. Como se puede ver en la figura 5.4 el error converge rápidamente a partir de las 15 épocas

A continuación se muestra una tabla comparativa con los resultados tras aplicar la red a 12 patrones de fallo en los rodamientos.

Nº de patrón	Fallo real	Salida 1 del sistema NF (fallo de bola)	Salida 2 del sistema NF (fallo de pista)	Salida 1 de ANFIS (fallo de bola)	Salida 2 de ANFIS (fallo de pista)
1	Fallo de bola	0.96833	0.031667	0.3785	0.6215
2	Fallo de bola	0.67843	0.32157	1.4999	0.4999
3	Fallo de bola	0.97738	0.022619	0.5786	0.4214
4	Fallo de bola	0.98157	0.018433	0.6217	0.3783
5	Fallo en pista interna	-2.0101e-011	1	0.6676	0.3324
6	Fallo en pista interna	0.090459	0.90954	0.2828	0.7172
7	Fallo en pista interna	0	1	0.0290	0.9710
8	Fallo en pista interna	0.010791	0.98921	0.1061	0.8939
9	Fallo en pista externa	0.078308	0.92169	-0.0374	1.0374
10	Fallo en pista externa	-1.2294e-017	1	0.2454	0.7546
11	Fallo en pista externa	0	1	-0.8551	1.8551
12	Fallo en pista externa	0.13847	0.86153	0.6379	0.3621

Tabla 5.3. Resultados del sistema Neuro-Fuzzy y ANFIS

En la tabla 5.3 se comparan los resultados del sistema neuro-fuzzy una vez entrenado y los del ANFIS clásico ante los patrones del conjunto de test. Como se puede ver, el sistema neuro-fuzzy planteado ha dado resultados correctos en los doce patrones, frente a los tres fallos del sistema ANFIS clásico. Además, el grado de discriminación del sistema ANFIS es mucho menor, siendo el patrón número 2 el que hace más patente este hecho, donde, si bien la salida es correcta, está extremadamente cerca del valor frontera.

Esta mejora frente al ANFIS se puede justificar por la mayor cantidad de información manejada (39 bandas frente a 7). Precisamente esta es una de las cualidades de este sistema neuro-fuzzy, dado que en el ANFIS de Jang [JAN93b] el número de neuronas está fuertemente ligado al número de entradas y funciones de pertenencia. La estructura de la red crece exponencialmente con el número de entradas, lo que se traduce en un gran problema de cómputo cuando este es elevado. El sistema neuro-

fuzzy utilizado, por el contrario, permite definir de antemano un máximo de neuronas en la capa oculta, acotando de esta manera el tamaño máximo de la red generada.

Cabe mencionar también que los tiempos de entrenamiento de un sistema y otro difieren de forma sustancial, siendo unas cinco veces más cortos en el sistema neuro-fuzzy alternativo por las razones anteriormente mencionadas.

6 NAVEGACIÓN INERCIAL Y ESQUEMAS DE FUSIÓN SENSORIAL

Este capítulo comienza describiendo el concepto de navegación inercial, así como la problemática de dichos sistemas. A continuación se introducen algunas de las técnicas empleadas actualmente para paliar los efectos negativos de dicha problemática. Se continúa con la descripción en profundidad del problema de navegación abordado en esta tesis y la solución planteada mediante un sistema neuro-fuzzy .Tras la descripción del desarrollo experimental, se muestran los resultados de la aplicación del sistema neuro-fuzzy al problema planteado.

6.1 Navegación inercial

Una *unidad de medida inercial* es un dispositivo equipado con tres giróscopos y tres acelerómetros dispuestos ortogonalmente que miden la velocidad angular y la aceleración lineal en cada eje. Con estos sensores es posible calcular los cambios de aceleración de un cuerpo y por tanto conocer su posición incrementalmente con respecto a la posición inicial. Existen dos tipos de sistemas de navegación inercial: de plataforma estabilizada y de plataforma analítica.

En los *sistemas de plataforma estabilizada* se monta una plataforma en una suspensión de cardanes y a su vez se fija esta al cuerpo a medir (figura 6.1). La plataforma permanece estable gracias a una serie de motores que hacen girar los marcos para compensar las rotaciones detectadas por los giróscopos. Esto permite que la plataforma permanezca alineada con el sistema de coordenadas de referencia independientemente de la inclinación o giros que realice el vehículo. La medida de la rotación de los marcos de la suspensión cardánica con respecto a la plataforma permite conocer la inclinación del sistema. Para obtener la posición del dispositivo se calcula la integral doble de la señal de los acelerómetros, ya que estos permanecen siempre alineados con el marco de referencia global. Simplemente es necesario sustraer el valor de la aceleración de la gravedad al acelerómetro que mide la componente vertical.

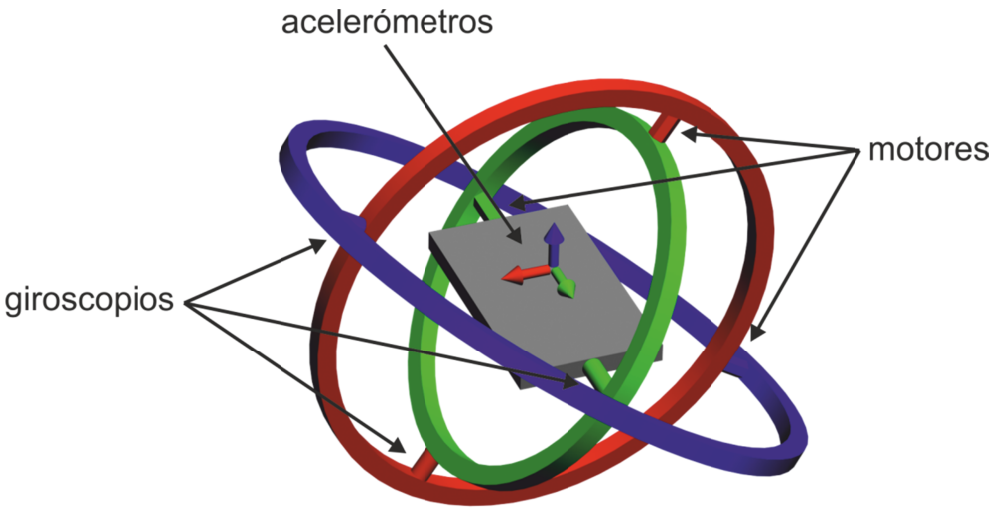


Figura 6.1. Unidad inercial de plataforma estabilizada

En los *sistemas de plataforma analítica* los sensores se montan directamente sobre el cuerpo a medir. Esto implica que las aceleraciones y giros registrados están definidos con respecto al marco de referencia del propio sensor (figura 6.2). Para poder conocer la posición y velocidad con respecto al marco de referencia global, primero hay que integrar la señal de los giróscopos para conocer la inclinación del cuerpo. Una vez obtenidos los ángulos de inclinación, es necesario proyectar los vectores de aceleración registrada sobre el marco de referencia global. Con la aceleración ya en coordenadas globales simplemente se sigue el mismo procedimiento de corrección de la gravedad y doble integración de la aceleración empleado en el sistema de plataforma estabilizada. Los requerimientos y complejidad mecánicos de este tipo de unidades inerciales son mucho menores que los sistemas de plataforma estabilizada. Esto permite que su tamaño sea mucho menor, y empleando dispositivos MEMS, se puede incluir una unidad inercial completa en un encapsulado de unos pocos milímetros cuadrados.

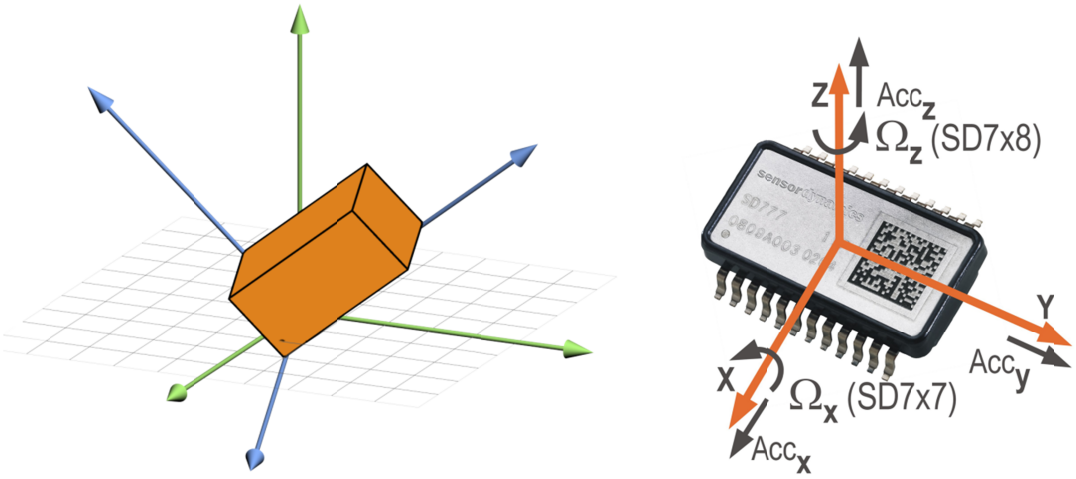


Figura 6.2. Unidad inercial de plataforma analítica

Tanto en los sistemas de plataforma estabilizada como en los sistemas de plataforma analítica, la navegación inercial presenta un inconveniente. Dado que las

medidas de los acelerómetros se integran para ir obteniendo incrementalmente la posición a partir de un origen, los errores cometidos en estas integraciones (errores de medida, de cálculo, etc.) también se van agregando. Esto provoca que el error cometido en la estimación de la posición crezca con el tiempo, sobre todo en unidades MEMS. Para paliar este problema se suelen emplear técnicas de fusión sensorial con otro tipo de sensores.

6.2 La fusión sensorial como solución al problema de la deriva

Como se ha mencionado, la navegación inercial por si sola presenta un problema inherente a su funcionamiento y es la acumulación del error de posición cuando no existen restricciones de movimiento [FOX05]. Este fenómeno se conoce como deriva, y puede provocar que en unidades de bajo coste el error de posición crezca a centenas de metros en menos de un minuto. Para solventarlo, lo habitual es emplear técnicas de *fusión sensorial*.

El esquema más clásico de fusión sensorial con una IMU está compuesto por una unidad de navegación inercial y un GPS, cuyos datos se combinan a través de un filtro de Kalman [KAL60], [WEL06]. En este modelo, extensivamente usado [ALI04], difundido y ampliado [CHR03][ABD06], la IMU proporciona datos acumulativos con una mayor frecuencia de refresco, proporcionando el rumbo con poco error a corto plazo. El GPS, sin embargo, proporciona la posición absoluta con un ratio de refresco mucho menor y con un error del orden de metros, pero sin deriva. Las medidas del GPS e IMU se combinan por tanto para obtener un sistema de navegación fiable, con información en tiempo real y un error de posición acotado.

Existen también esquemas de fusión sensorial con unidades de navegación inercial basados en visión artificial. En [HID10] se describe un esquema capaz de reducir la deriva de un sistema de navegación personal basado en IMU, empleando una cámara de vídeo colocada en la cintura apuntando al suelo. Mediante el seguimiento de puntos característicos en la imagen y empleando un filtro de Kalman para combinar la información se consigue un sistema capaz de mantener acotado el error de posición sobre el plano aun cuando hay ausencia de información de un GPS. Otro ejemplo de combinación IMU-Visión es el sistema de navegación de los MER (Mars Exploration Rovers) [MAI07] cuyo funcionamiento básico se basa en la odometría clásica combinada con una IMU. En aquellos terrenos arenosos o muy accidentados, donde la odometría introduce errores por deslizamiento de las ruedas, se emplea la información proveniente del sistema de visión como apoyo al cálculo de rumbo. Esta información se obtiene mediante seguimiento de puntos característicos de la imagen de un par de cámaras estereoscópicas. En las siguientes secciones se muestra un esquema de fusión cuya fuente de datos se basa en el sistema de navegación inercial Xsens Mtx y una cámara de bajo costo en un contexto tridimensional

6.3 Navegación basada en fusión sensorial mediante un esquema Neuro-Fuzzy

El esquema tradicional de navegación empleando fusión sensorial entre IMU y GPS [BEV06] sólo es aplicable en entornos abiertos, donde alcanza la cobertura de los satélites del sistema GPS. En escenarios cerrados es necesario buscar otras fuentes de información para complementar la información proporcionada por la IMU. En este apartado se presenta un esquema de fusión basado en IMU y visión artificial que trata de paliar los problemas de la navegación inercial en aquellos escenarios donde no es posible emplear un GPS. Para el desarrollo de los experimentos se ha empleado un sensor Xsens Mtx como el descrito en el apartado 3.1.4.2, que proporciona la aceleración e inclinación en los tres ejes.

6.3.1 Algoritmo de navegación para el sistema de plataforma analítica

El algoritmo de navegación empleado es el clásico *strapdown algorithm*. Este sencillo algoritmo consiste en integrar el valor de la aceleración para obtener la velocidad y a continuación integrar la velocidad para obtener la posición, siempre a partir de un estado de aceleración, velocidad y posición conocido.

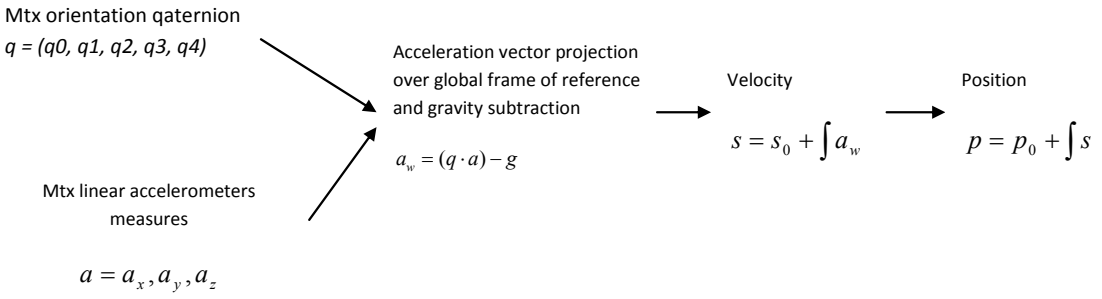


Figura 6.3. Algoritmo "strapdown"

El dispositivo IMU contiene tres acelerómetros lineales dispuestos de forma ortogonal (uno por eje). Esto hace que los valores obtenidos correspondan a las tres componentes cartesianas de la aceleración referidas al sistema de coordenadas local al dispositivo. Debido a esto, el vector de aceleración leído debe ser proyectado sobre los ejes del sistema de coordenadas global para obtener la aceleración en coordenadas "globales". Una vez proyectada la aceleración en el sistema global, hay que sustraer a la componente vertical el valor de la aceleración de la gravedad para tener la aceleración debida únicamente al movimiento del objeto. Con la aceleración real del dispositivo, ya es posible obtener la posición en coordenadas globales a través de la doble integral de cada componente.

6.3.1.1 Medida de posición y fuentes de ruido

Para medir la posición se cuenta con los valores leídos de los acelerómetros a intervalos regulares de tiempo (120 muestras por segundo). Empleando un esquema de integración numérica, se puede averiguar la posición mediante la integral doble de las medidas de aceleración. Como cualquier otro tipo de sensor, los acelerómetros están afectados por fuentes de ruido. Este ruido, al integrarse junto con la señal, se convierte en una deriva constante en la posición calculada.

Como ejemplo, se llevó a cabo un sencillo experimento en el cual se fijó el sensor a una superficie nivelada y completamente estática. Se tomaron muestras a razón de 120 por segundo y se integraron dos veces para calcular la posición. El resultado esperado hubiera sido que la posición expresada en coordenadas cartesianas fuese (0, 0, 0), o sea, que el sensor no se hubiese movido. Sin embargo, en la figura 6.4, donde las tres gráficas representan el desplazamiento frente al tiempo en cada eje (x, y, z de arriba hacia abajo), se puede ver que a los treinta segundos los valores de desplazamiento calculados son 4m. en el eje *x*, 3.5m en el eje *y* y poco más de 9m en el eje *z*. Si se calcula la distancia euclídea se puede comprobar que la posición calculada dista 10.7m del origen de coordenadas, en tan solo 30 segundos. Además, observando la tendencia de la gráfica, se comprueba fácilmente que este error crece cada vez más rápido.

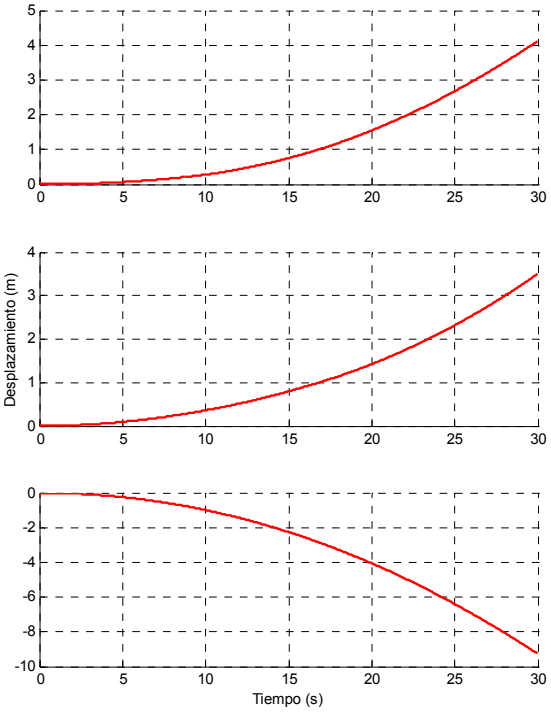


Figura 6.4. Deriva del sensor

Las principales fuentes de ruido que afectan a los acelerómetros son tres: ruido blanco gaussiano, sesgo en las medidas y variación del sesgo con el tiempo. El ruido blanco es inherente a cualquier tipo de sensor y a priori es susceptible de ser tratado con técnicas de filtrado convencionales (empleando por ejemplo un filtro paso-bajo). El sesgo sin embargo no es fácil de determinar, y al ser integrada la señal dos veces para calcular la posición, se convierte en un error de orden cuadrático, contribuyendo enormemente al error total. Intuitivamente se podría suponer que promediando la señal un periodo suficientemente largo, el valor obtenido sería una estimación del sesgo. Como se verá a continuación, esto no es así.

6.3.1.2 Desviación de Allan

La *Desviación* o *Varianza de Allan* es una técnica desarrollada para caracterizar las fuentes de ruido y la estabilidad en una señal. Originalmente se creó para el análisis de relojes de precisión, pero puede ser aplicada a cualquier tipo de señal [WOD07]. La ventaja de este método es que permite identificar de forma conveniente sobre una gráfica cuales son los tipos de ruido que subyacen. En este caso se empleará la Varianza de Allan para hacer el cálculo óptimo del *offset* (sesgo en la medida) de los acelerómetros, con el fin de poder sustraerlo correctamente.

La Varianza de Allan consiste en el cálculo de la varianza de los valores medios de la señal, tomados sobre intervalos de tamaño creciente siguiendo una escala logarítmica. El tamaño máximo del intervalo será de la novena parte de la duración total del tiempo que se tenga grabado, dado que con menos de 9 valores, el resultado de la Varianza de Allan pierde significado. El algoritmo es el siguiente:

1. Se graba un número de muestras de señal suficientemente grande
2. Desde t = 0,1 hasta t = (número de muestras / 9)
   a. Dividir la señal capturada en intervalos de tamaño *t*
   b. Calcular la media de cada intervalo, obteniendo una lista de medias: $a(t)_1$, $a(t)_2$, $a(t)_3$, … , $a(t)_n$
   c. Calcular la varianza de la lista de medias

$$AVAR(t) = \frac{1}{2\cdot(n-1)}\sum_i (a(t)_{i+1} - a(t)_i)^2$$

Si se representan los valores de la Desviación de Allan (raíz cuadrada de su varianza) frente a la duración del intervalo, se obtendrá una gráfica en la que se puede ver cómo quedan representadas las distintas componentes del ruido:

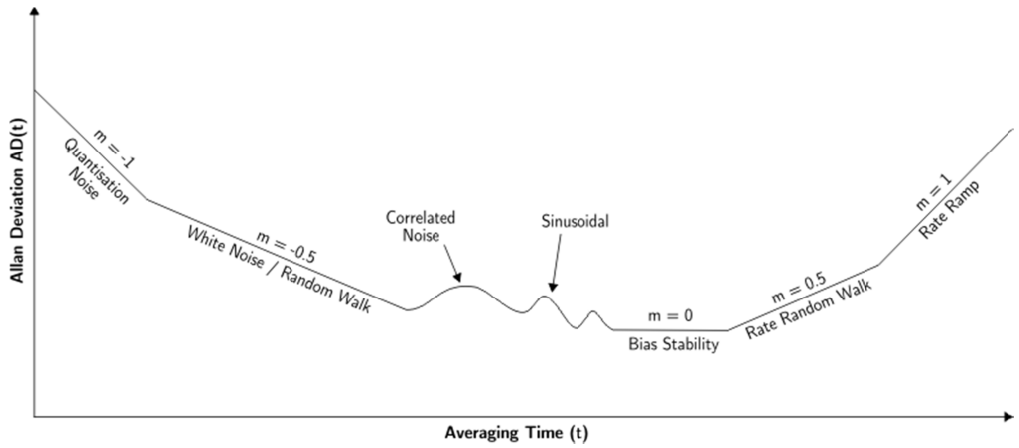


Figura 6.5. Desviación de Allan

Para determinar los valores de sesgo del sensor de que se dispone, se grabó una secuencia de lecturas de 12 horas de duración, tomando 120 muestras por segundo con el sensor colocado sobre una superficie fija y nivelada y con el eje Z orientado en la

vertical. Al resultado se le aplicó la técnica descrita con el siguiente resultado:

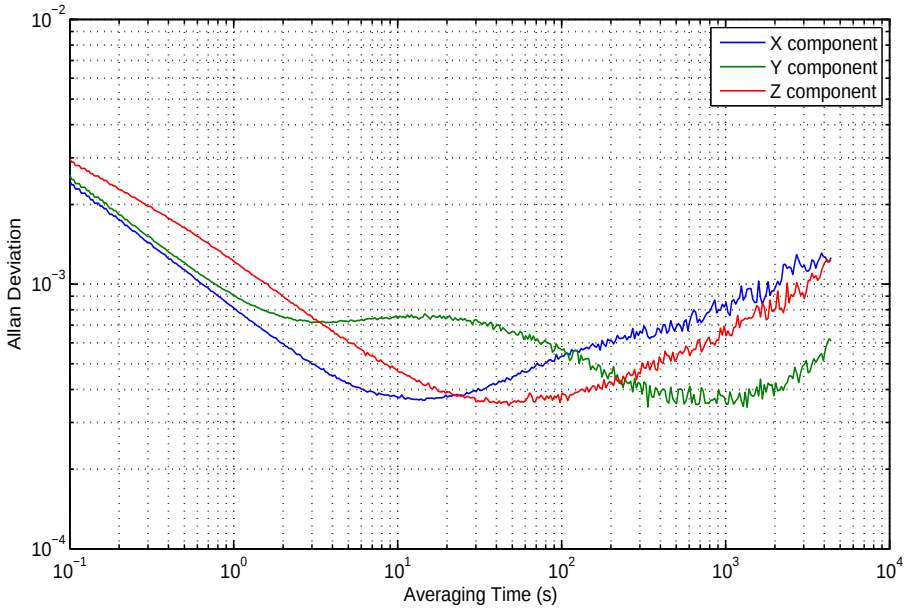


Figura 6.6. Gráfico de Allan para el sensor Mtx.

Como se puede observar en la figura, los mínimos en la gráfica para los ejes X, Y y Z se encuentran a 14,61, 1353,3 y 47,5 segundos respectivamente. Estos son los valores óptimos de duración del intervalo a promediar para calcular el offset. Como experimento, se promedió un intervalo de duración arbitraria suficientemente largo ($2x10^6$ muestras - 4,5 horas aproximadamente) con el fin de comparar los resultados con los obtenidos aplicando la técnica de la desviación de Allan.

Estos valores de offset fueron sustraídos a la señal empleada anteriormente para mostrar la deriva (figura 6.4). Tras integrar dos veces la señal de aceleración y representar los resultados, podemos ver en la figura 6.7, cómo con el offset arbitrario (línea verde) se logra reducir la deriva de 10,7m a solo 1,85m, lo cual es una mejora notable. Sin embargo, si se observa la deriva correspondiente a la aplicación de la Desviación de Allan (línea azul), el error de posición a los 30s se ve reducido a solo 0,43m, quedando demostrada la eficacia de esta técnica. Sin embargo, aunque efectiva, la aplicación de esta técnica

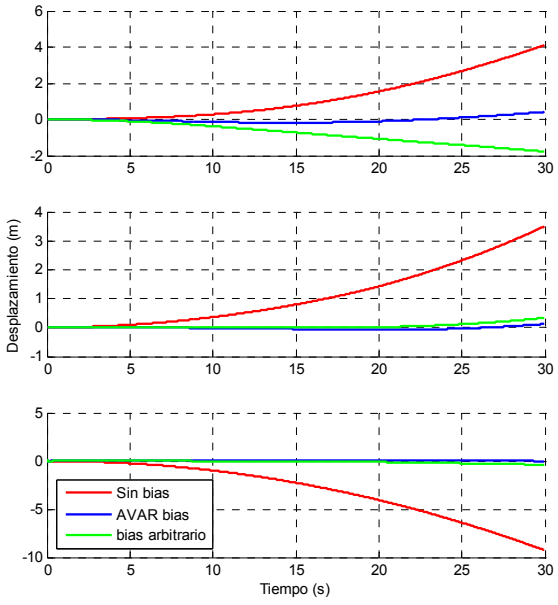


Figura 6.7. Comparativa de resultados

no es suficiente para establecer la posición a medio o largo plazo con precisión, dado que sigue existiendo deriva en las medidas.

6.3.2 Solución propuesta al problema de la deriva

La solución propuesta consiste en aplicar un esquema de fusión sensorial basado en un sistema neuro-fuzzy entre el dispositivo IMU y una cámara de vídeo convencional de bajo coste. La elección de una cámara como dispositivo adicional a la IMU se debe a que es un sensor que proporciona información espacial, aunque de forma indirecta, y tiene ventajas como su pequeño tamaño, es pasivo, de bajo coste, y fácilmente integrable en un prototipo.

Para poder aplicar el sistema neuro-fuzzy, el planteamiento seguido ha sido obtener por un lado una estimación de la posición aplicando el algoritmo mostrado en el apartado 6.3.1 a los datos de orientación y aceleración provistos por el sensor IMU. Por otro lado se ha calculado el flujo óptico de las imágenes con el fin de utilizarlo como indicador parcial de la dirección del movimiento. Estos datos, junto con la orientación obtenida directamente del sensor IMU conforman el conjunto de entrada para el sistema.

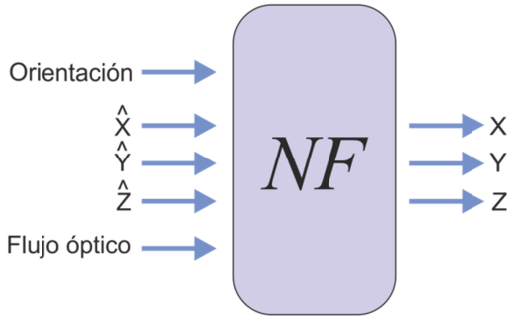


Figura 6.8. Planteamiento del sistema neuro-fuzzy para la estimación de posición

La salida esperada son unos valores de posición X, Y y Z corregidos y libres de deriva en el tiempo.

6.3.2.1 Montaje experimental

Para llevar a cabo los ensayos y experimentos requeridos en este trabajo, fue necesaria la construcción de un prototipo hardware y el desarrollo de algunas aplicaciones de software destinadas a la adquisición y procesado de los datos.

6.3.2.1.1 Dispositivo de captura

El prototipo aquí mostrado se construyó para ser usado en el ámbito del proyecto Espacio Acústico Virtual [ROD10]. El objetivo fue alojar en una gorra de tipo “flat cap” los elementos de un sistema de visión, posicionamiento y reproducción de sonido. El requisito de estar alojado en una gorra se debe al hecho de que está pensado para ser

usado por una persona ciega y era una condición imprescindible que el prototipo fuese visualmente discreto.

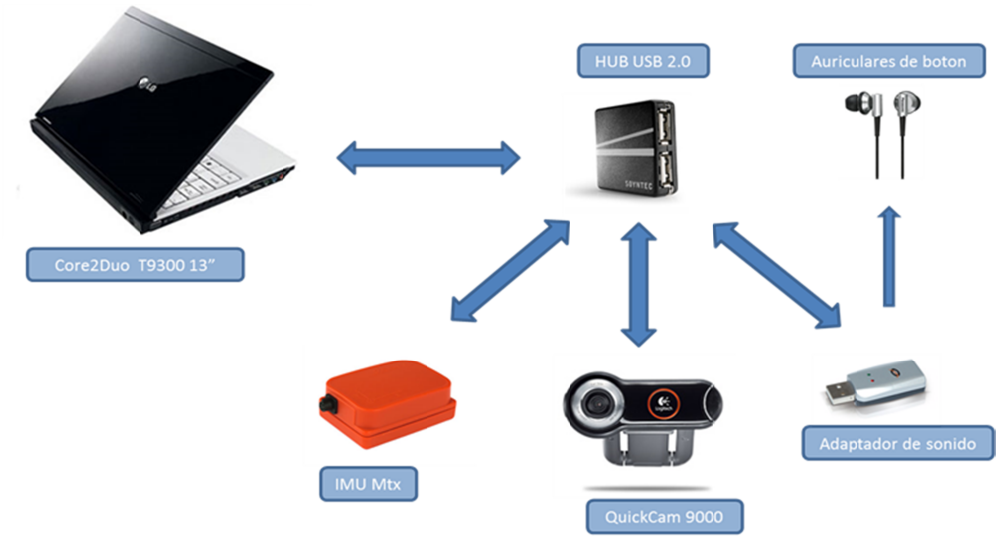


Figura 6.9. Elementos del prototipo

La construcción se llevó a cabo sobre una estructura de plástico ligero fabricada a medida en la que se instaló la placa de la cámara de vídeo, la placa de sonido y el cuerpo del sensor Mtx (ver apartado 3.1.4.2). Además se añadió un hub USB 2.0 con el fin de interconectar los dispositivos al ordenador que ejecuta la aplicación de captura. Dado que todos los dispositivos usan una interfaz de datos USB (el sensor inercial lleva un convertidor RS232-USB), se puede usar un solo cable para la alimentación y los datos de los tres dispositivos.

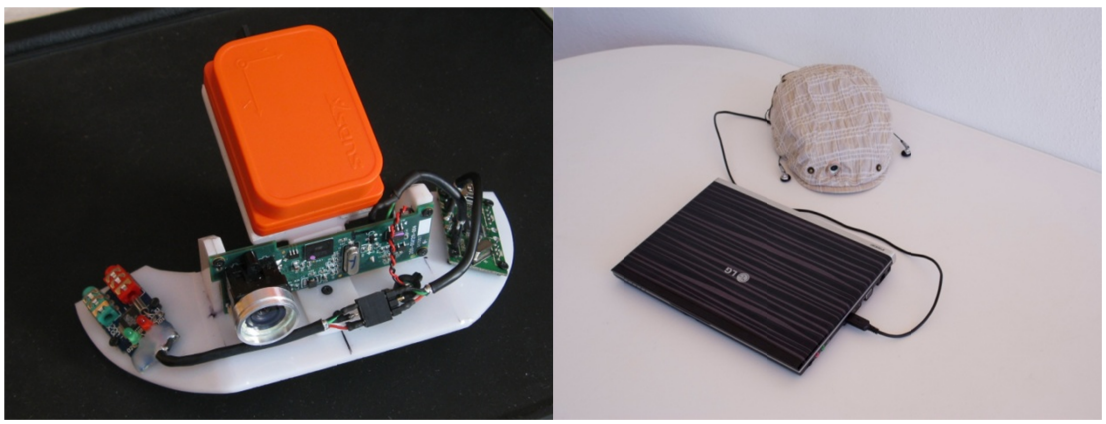


Figura 6.10. Aspecto final del prototipo de captura

La cámara empleada en este sistema es el modelo QuickCam 9000 de Logitech.

Figura 6.11. Cámara Logitech QuickCam 9000

Esta cámara de bajo coste posee un sensor CMOS de 2.0 megapixels y alcanza las 30 imágenes por segundo en la captura, además de estar equipada con una lente con autoenfoque fabricada por la empresa Carl Zeiss. En el momento de su elección, esta cámara disponía de las mejores especificaciones de entre las disponibles en el mercado en su segmento.

6.3.2.1.2 Software

Para poder llevar a cabo el entrenamiento del sistema neuro-fuzzy, fue necesario capturar primero un conjunto de datos, que en este caso son:

- Orientación y aceleración en los tres ejes del sensor Mtx.
- Flujo óptico de las imágenes de la cámara.
- Posición real obtenida mediante un sistema externo.

El software tenía que ser capaz de capturar simultáneamente de todas las fuentes de datos y grabarlas a fichero para su posterior uso. Dado que cada fuente de datos proporciona información a un ritmo distinto, es necesario grabar junto a cada dato, un código de tiempo que permita hacer una adecuada interpolación de los datos a posteriori.

Para cada uno de los dispositivos de captura se desarrolló una librería que contiene las funciones necesarias de inicialización y captura y que facilita la posterior reutilización del código. En la figura 6.12 se presenta el diagrama global del diseño del software, que se explicará a continuación.

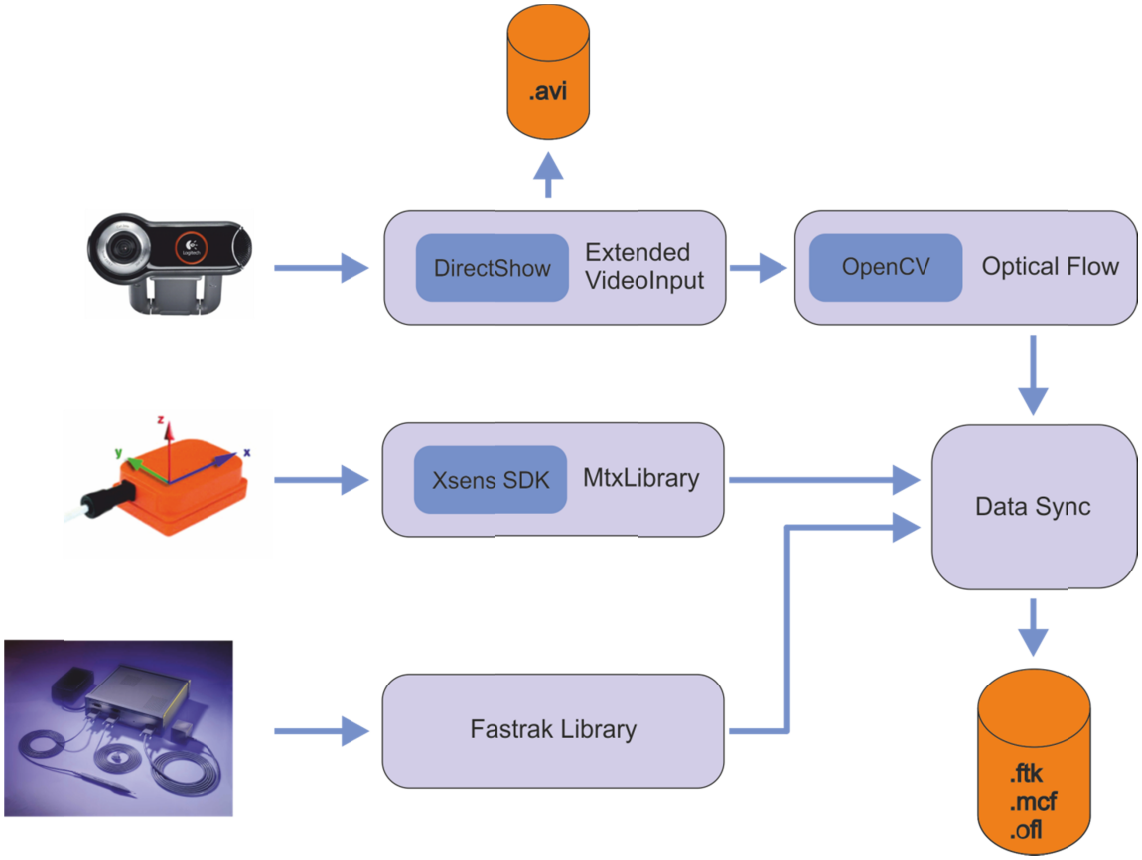


Figura 6.12. Diagrama del software de captura

6.3.2.1.2.1 Sistema de visión

Para acceder a la cámara y capturar la secuencia de vídeo se usó el entorno DirectShow del sistema operativo Microsoft Windows. Este entorno se maneja a través de una librería, desarrollada como una modificación hecha sobre la librería VideoInput de Theodore Watson [VIL] y que se ha denominado Extended VideoInput. Esta nueva librería extiende su funcionalidad, compatibilizándola con la librería OpenCV [OCV] y permitiendo crear conjuntos de filtros de DirectShow [MDS] personalizados de forma que, en este caso, se puede grabar la secuencia de vídeo a un fichero con formato AVI, a la vez que se envía cada fotograma al siguiente módulo de software para calcular el flujo óptico.

Para el cálculo del flujo óptico se ha empleado la variante piramidal del algoritmo de Lucas-Kanade implementada en la librería OpenCV. La solución que presenta Bouget [BOU99] en su implementación del algoritmo (ver apartado 3.2.1) es una aproximación piramidal, empleando varias versiones de cada imagen a distintas resoluciones. Es decir, para cada imagen I se crean varios niveles de submuestreo. El número de niveles necesario se obtiene de forma empírica. Para una imagen de 640x480 píxeles y una pirámide de cuatro niveles, se crearían versiones de 320x240, 160x120, 80x60 y 40x30 píxeles. En cada nivel de la imagen se refina el resultado del nivel anterior. Con esto se consigue un rango y velocidad de convergencia suficientemente grande, a la vez que se conserva la precisión en el cálculo del desplazamiento.

Cabe mencionar también, que esta implementación del algoritmo piramidal usa una lista dispersa de puntos en vez de una matriz densa equiespaciada, como sería normal. Los puntos de seguimiento se seleccionan buscando esquinas, dado que estos puntos son fácilmente identificables a pesar de los efectos de posibles traslaciones y rotaciones de la cámara. La propia implementación del algoritmo proporciona la información necesaria para descartar aquellos puntos que se han perdido en la imagen, ya sea por incapacidad de emparejarlos o porque salen del encuadre o quedan ocultos.

Como ya se ha mencionado en el apartado 3.2, el flujo óptico es el patrón de *movimiento aparente* que aparece en una imagen debido al movimiento relativo entre la cámara y su entorno. Por tanto, la aparición de estos patrones no es sólo debida al movimiento de la cámara, sino que también los objetos móviles que aparecen en la imagen producirán vectores de flujo óptico. Dado que lo que se intenta calcular es el movimiento de la cámara, es interesante minimizar la contribución que estos vectores debidos a objetos móviles dan al resultado final.

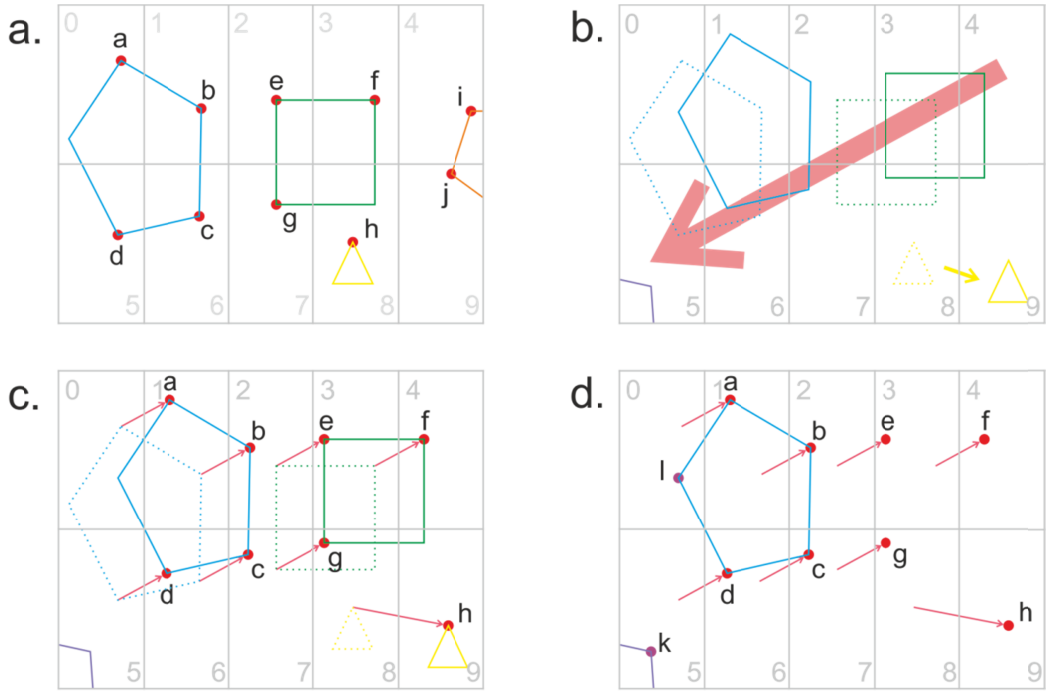


Figura 6.13. Cálculo del flujo óptico

Para reducir el efecto de los objetos móviles, se fuerza la dispersión de puntos de seguimiento por toda la imagen. Para ello se divide la imagen en un número de regiones fijo, en este caso 10. Dentro de cada región se busca un punto fácilmente identificable sobre el que se hará el seguimiento en sucesivas imágenes. Esto garantiza que la matriz de puntos para hacer el seguimiento este dispersa por toda la imagen, y no concentrada en una región donde, si existiese un objeto móvil, confundiría totalmente el cálculo del movimiento relativo de la cámara. El motivo de escoger 10 regiones se debe a que por cada región se genera un vector de flujo óptico entre imágenes, que se le pasa directamente al sistema neuro-fuzzy como entrada en forma de radio y ángulo (ver

figura 6.8). Aumentar el número de regiones en este planteamiento significaría aumentar el número de entradas, que ya es bastante elevado para este tipo de sistemas.

El procedimiento para mantener la matriz de puntos para el cálculo de los vectores de flujo óptico se puede observar en el siguiente ejemplo referido a la figura 6.13:

1. Se busca en cada región un punto adecuado para el seguimiento (*punto de tracking*) mediante la función ”GoodFeaturesToTrack” de la librería OpenCV. Estos puntos están representados como puntos rojos en la figura 6.13a

2. La cámara se mueve en dirección abajo-izquierda y el triángulo se mueve en dirección abajo-derecha (figura 6.13b). Esto produce un movimiento aparente en dirección contraria al desplazamiento de la cámara y en la misma dirección del desplazamiento del triángulo.

3. Se buscan los puntos de tracking en la nueva imagen (figura 6.13c) generándose los vectores de flujo óptico.

4. En aquellas regiones donde ya no hay puntos de tracking se buscan nuevos (regiones 0 y 5, figura 6.13d, representados en violeta) y se descartan los sobrantes en las regiones donde exista más de uno.

6.3.2.1.2.2 Adquisición de datos de aceleración e inclinación

El acceso al sensor se realiza a través de la interfaz de programación que proporciona el fabricante. Esta interfaz permite leer los datos del sensor tanto en bruto como calibrados. Cuando se accede a los datos en bruto se obtienen los valores de los conversores analógico-digitales directamente, sin ningún tipo de calibración previa. Sin embargo, lo más útil es leer los datos calibrados ya en metros por segundo (para los acelerómetros) o en grados por segundo (para los giroscopios).

El sensor dispone de tres giroscopios para medir el ratio de giro. Si se quiere obtener la inclinación absoluta, habrá que integrar la lectura de los giroscopios. Para hacer esto, el fabricante proporciona un método de acceso a través del cual las medidas se integran fusionando los valores de los giroscopios con los valores de los magnetómetros y la medida del vector gravedad. Para llevar a cabo la fusión sensorial emplea un filtro de Kalman [KAL60], en el que los valores de los magnetómetros se usan en el filtrado de la rotación sobre el eje vertical y la medida del vector gravedad se usa para filtrar la inclinación en ambos ejes horizontales. Como resultado, se puede obtener una medida de la inclinación del sensor con un alto grado de precisión y con muy poca deriva a lo largo del tiempo.

6.3.2.1.2.3 Adquisición de datos de posición

Para la construcción del conjunto de datos reales se empleó un sistema de posicionamiento 3D de tipo magnético, concretamente el modelo Fastrak de Polhemus. Este sistema está instalado en una sala acondicionada para el simulador de realidad virtual del grupo EAV [TOR10] (figura 6.14). El dispositivo es capaz de posicionar hasta cuatro sondas con seis grados de libertad cada una, en un radio de 5 m. alrededor de su antena. Las especificaciones teóricas de este sistema son:

- Información de posición x, y, z y ángulos de orientación.
- Precisión de 0.005mm.
- Rango de uso de 10m con la antena Long Ranger instalada
- Capacidad de muestreo de 120Hz (con una sola sonda instalada)
- Interfaz RS-232

Figura 6.14. Sala de RV del grupo EAV

En la práctica, los valores de precisión y rango distan mucho de los referidos por el fabricante. Esto se debe a que los sistemas de posicionamiento magnético sufren una serie de problemas bien conocidos. Estos sistemas son muy precisos en rangos cortos y en condiciones ideales, pero cualquier objeto metálico cerca de la antena o de las sondas, provoca una distorsión en el campo magnético producido. Esta distorsión genera un error demasiado grande como para que las medidas obtenidas sean usables.

Para paliar en la medida de lo posible los errores producidos por las interferencias en el campo magnético, se optó por crear un algoritmo de calibrado, que posteriormente quedaría integrado en la aplicación de simulación 3D del grupo EAV. Dado que algunas experiencias preliminares revelaban que los errores en las medidas de posición y orientación obedecían a un patrón, se podría obtener la lectura del sistema Fastrak en puntos conocidos y con esa información elaborar un modelo para corregir los valores devueltos.

El primer paso fue medir una cuadrícula regular de 80cm de lado usando un poste vertical donde se fija la sonda mediante una base fabricada al efecto (figura 6.15). Esta base mantiene la sonda siempre en la misma orientación, alineada ortogonalmente con las paredes y el suelo. Con este método se garantiza la regularidad en las medidas. En cada punto de la malla regular se registró tanto la medida real de la posición y orientación de la sonda, como la medida devuelta por el sistema. Como se puede ver en la figura 6.16, donde se muestran los puntos devueltos por el sistema Fastrak, la deformación del campo es sustancial, aumentando a medida que la sonda se aleja de la antena, que está encima de la malla, en su parte central.

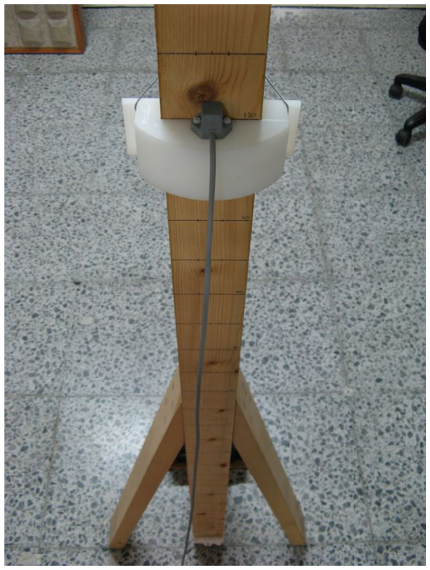


Figura 6.15. Poste de calibración

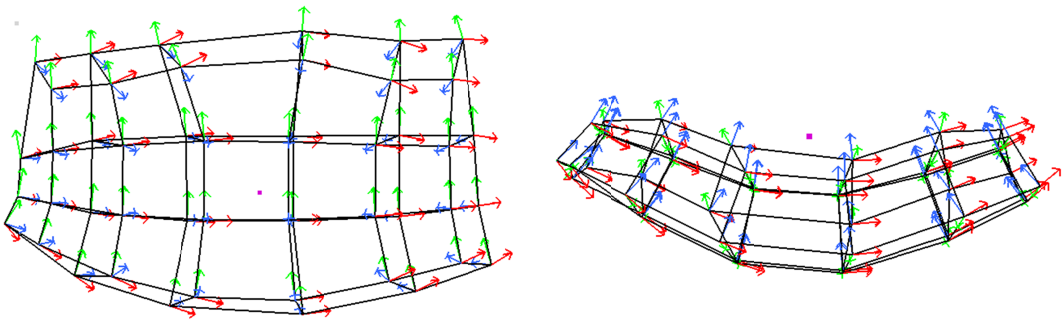


Figura 6.16. Vista superior (izq.) y lateral (dcha.) de la malla de calibración

Una vez obtenido el modelo de deformación, se puede establecer una correspondencia entre las coordenadas y orientación devueltas por el sistema Fastrak y la posición real mediante un método de interpolación.

Debido a que la malla de medidas es uniforme en el espacio real, pero está distorsionada en el espacio de medidas de Fastrak, no se puede aplicar una simple interpolación trilineal. La aproximación implementada consiste en buscar el cubo de la matriz de calibración que contiene el punto *p*. Una vez encontrado, se escogen dos triángulos en dos caras opuestas del cubo que tengan un punto de intersección con una recta paralela a alguno de los ejes de coordenadas del espacio real y que pase por el punto *p*. A partir de aquí ya se puede realizar la sucesión de interpolaciones lineales.

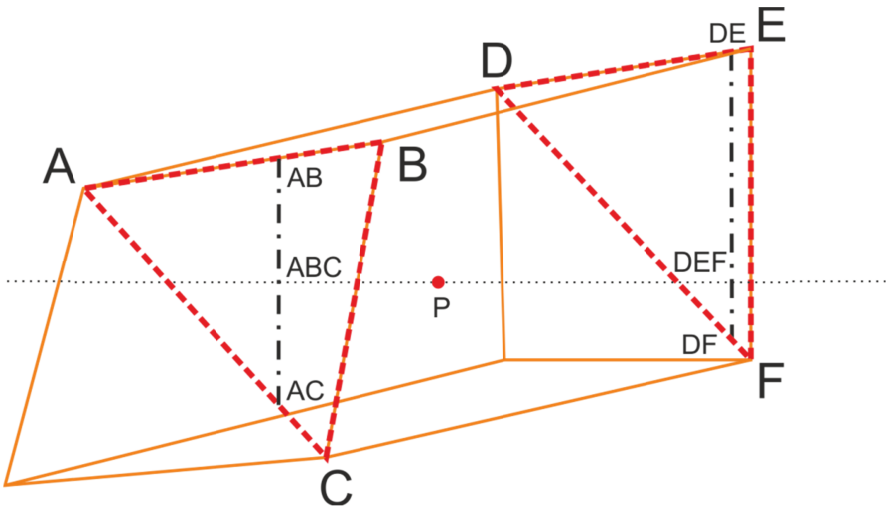


Figura 6.17. Interpolación de puntos

Expresado de forma algorítmica:

1. Se obtiene el punto P del sistema Fastrak
2. Se busca en la matriz de calibración el cubo que contiene el punto P
3. Se traza una recta paralela al eje X y que pase por el punto P. Se buscan dos triángulos en caras opuestas del cubo que intersecten con esta recta. Si no se encuentran, se prueba con una recta paralela al eje Y y si tampoco se encuentran, paralela al Z.
4. Con el punto de intersección entre la recta trazada por el punto P y los triángulos, se hallan los siguientes puntos:
    a. AB como interpolación entre A y B
    b. AC como interpolación entre A y C
    c. DE como interpolación entre D y E
    d. DF como interpolación entre D y F
    e. ABC como interpolación entre AB y AC
    f. DEF como interpolación entre DE y DF
5. P' en el espacio real se haya a partir de la interpolación entre los puntos ABC y DEF

6.3.2.1.2.4 Captura y almacenamiento de los datos

La captura de datos de las diferentes fuentes se lleva a cabo de forma concurrente empleando múltiples threads de ejecución. Para el manejo y control de los threads se ha utilizado la librería Threading Building Blocks [TBB] desarrollada por Intel.

El software de captura simultánea de datos genera tres ficheros con la siguiente extensión:

- *.ftk* para el fichero de capturas del sistema Fastrak. En este fichero se almacena cada muestra de posición y orientación junto a un código de tiempo de captura.

- *.ofl* para el fichero que contiene la información de flujo óptico. Se almacenan los vectores de pares ángulo-radio pertenecientes al flujo óptico de cada imagen de vídeo capturada. Para el vector de flujo de cada imagen se almacena el código de tiempo de captura.

- *.mcf* para el fichero de muestras del sensor Mtx. Se almacenan las muestras de los tres acelerómetros y la orientación, junto al código de tiempo de captura.

Además de estos ficheros, se guarda el vídeo obtenido de la cámara del prototipo para que pueda ser empleado como referencia si fuese necesario en un posterior análisis de los datos.

6.3.3 Aplicación del sistema neuro-fuzzy

Para aplicar el esquema neuro-fuzzy descrito en el apartado 5.3, primero fue necesario combinar la información de todos los sensores. Dado que cada sensor captura datos a distintos intervalos, antes de formar las matrices de datos de entrada se aplicó una interpolación mediante splines a los datos de los diferentes sensores para calcular los valores en los instantes de captura de imágenes, dado que estos son los de menor frecuencia de muestreo. Una vez obtenidos los datos interpolados, se aplicó el algoritmo strapdown a los datos de aceleración, obteniendo las posiciones estimadas para cada instante.

6.3.3.1 Posiciones absolutas

El primer ensayo se llevó a cabo usando directamente las posiciones estimadas mediante el algoritmo strapdown. Con el fin de simplificar el problema, se crearon tres sistemas neuro-fuzzy, uno para cada coordenada de salida.

Los vectores de entrada al sistema son de la forma:

$$U = (t, r_1, \dots, r_{10}, \theta_1, \dots, \theta_{10}, \hat{x}, \hat{y}, \hat{z}, q_w, q_x, q_y, q_z) \quad \textbf{(6.1)}$$

donde:

- t indica el tiempo de la captura en segundos

- $\theta_1, r_1, \dots, \theta_{10}, r_{10}$ son los vectores de flujo óptico expresados en forma polar

- $\hat{x}, \hat{y}, \hat{z}$ son las coordenadas de posición estimada, calculadas mediante el algoritmo strapdown

- q_w, q_x, q_y, q_z son las componentes del quaternion de rotación del sensor Mtx

La salida del sistema es una coordenada X, Y, o Z (dependiendo del sistema escogido) que se espera que esté corregida y libre de deriva.

6.3.3.1.1 Entrenamiento

Para el entrenamiento se han empleado 1701 patrones, de los cuales 1191 (70%) corresponden a patrones de entrenamiento y 510 (30%) corresponden a patrones de validación.

- Eje X

El entrenamiento para la red correspondiente al eje X se llevó a cabo con 20000 épocas en su etapa no supervisada y 2400 épocas en la etapa supervisada. En la figura 6.18 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.1 se muestran los errores mínimos alcanzados durante el entrenamiento.

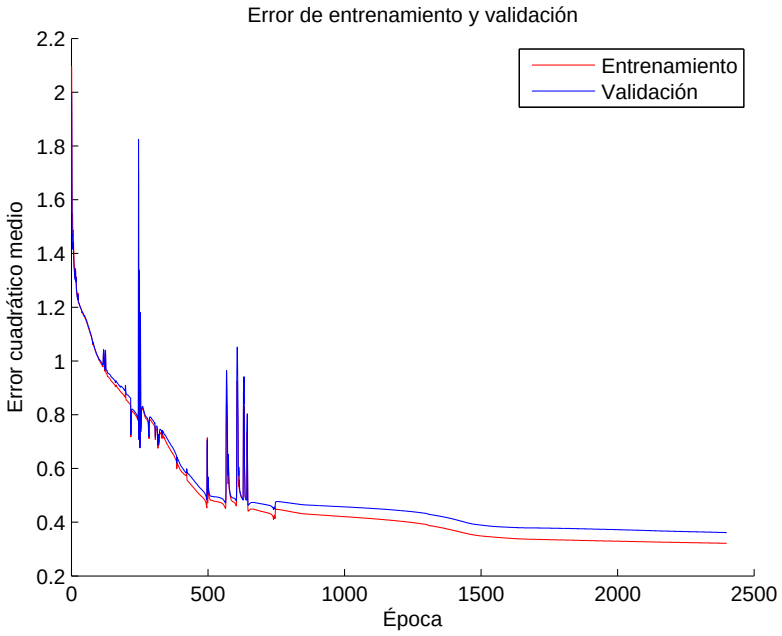


Figura 6.18. Progresión del error en el entrenamiento del eje X

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.3215	0.3615

Tabla 6.1. Errores mínimos del entrenamiento para el eje X

- Eje Y

Para la red correspondiente al eje Y se emplearon 20000 épocas en su etapa no supervisada y 1100 épocas en la etapa supervisada. En la figura 6.19 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.2 se muestran los errores mínimos alcanzados durante el entrenamiento.

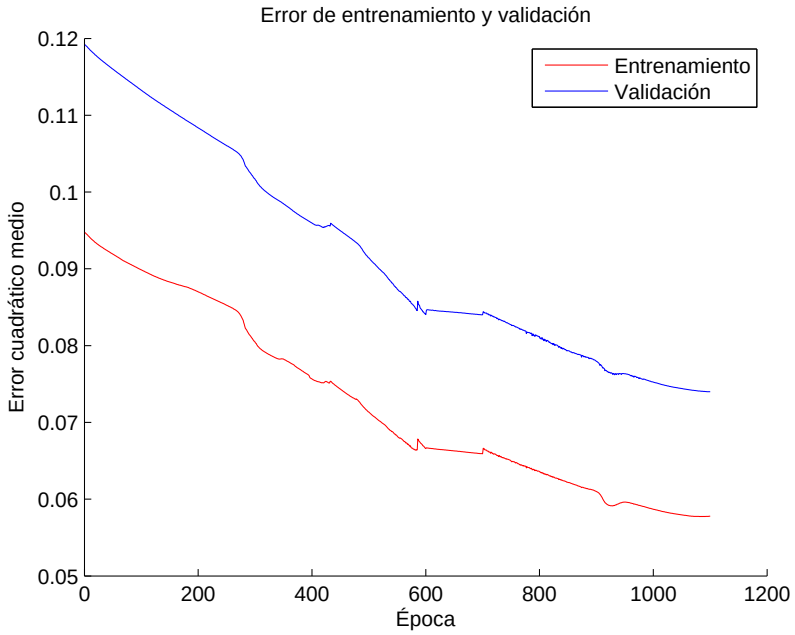


Figura 6.19. Progresión del error en el entrenamiento del eje Y

Cabe mencionar que aunque en el gráfico no se aprecie, el error se estabiliza brevemente en el último tramo de la curva. Tras el ciclo número 1100, la curva de error se volvía creciente independientemente de la velocidad de aprendizaje escogida, por lo cual se decidió finalizar el entrenamiento en este punto.

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.0577	0.0740

Tabla 6.2. Errores mínimos del entrenamiento para el eje Y

- Eje Z

Para la red correspondiente al eje Z se emplearon 20000 épocas en su etapa no supervisada y 1270 épocas en la etapa supervisada. En la figura 6.20 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.3 se muestran los errores mínimos alcanzados durante el entrenamiento.

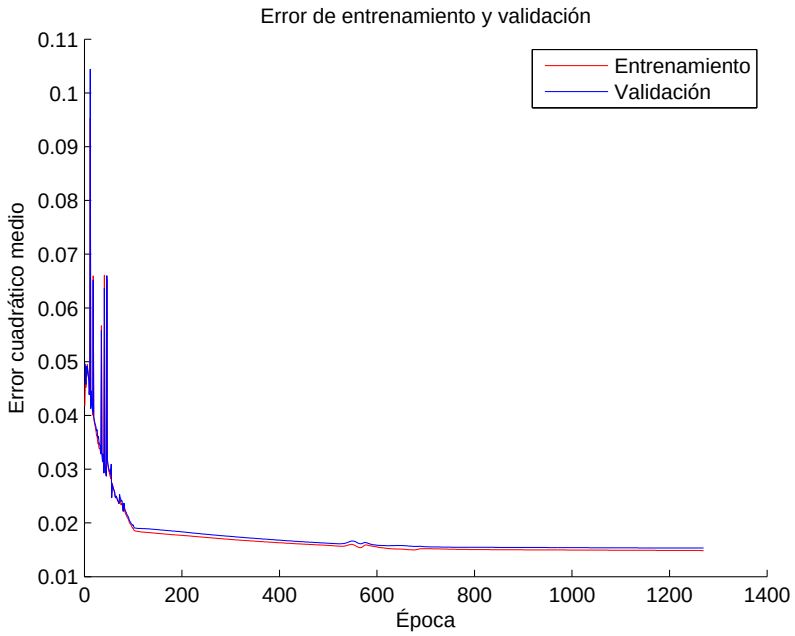


Figura 6.20. Progresión del error en el entrenamiento del eje Z

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.0149	0.0153

Tabla 6.3. Errores mínimos del entrenamiento para el eje Z

6.3.3.1.2 Resultados

A continuación se muestra una ruta de ejemplo en cada uno de los tres ejes, comparando el camino real con el calculado mediante el algoritmo strapdown y el mejorado mediante el sistema neuro-fuzzy. Junto a cada gráfica se muestra una tabla resumen de los errores cometidos con ambas técnicas.

Es necesario aclarar que el error mínimo del algoritmo strapdown siempre va a ser cero, debido a que la posición se calcula de forma acumulativa. Por tanto, en el instante inicial, cuando aún no hay valores acumulados, la posición siempre coincidirá con el origen de coordenadas, al igual que el camino real. Cabe también mencionar que el error máximo cometido depende del tiempo de evolución mostrado, dado que, como se puede ver en las figuras, el distanciamiento del camino real crece sustenacialmente con el tiempo.

El conjunto de test corresponde a 651 patrones obtenidos secuencialmente que corresponden a 21,7 segundos de ruta.

- Eje X

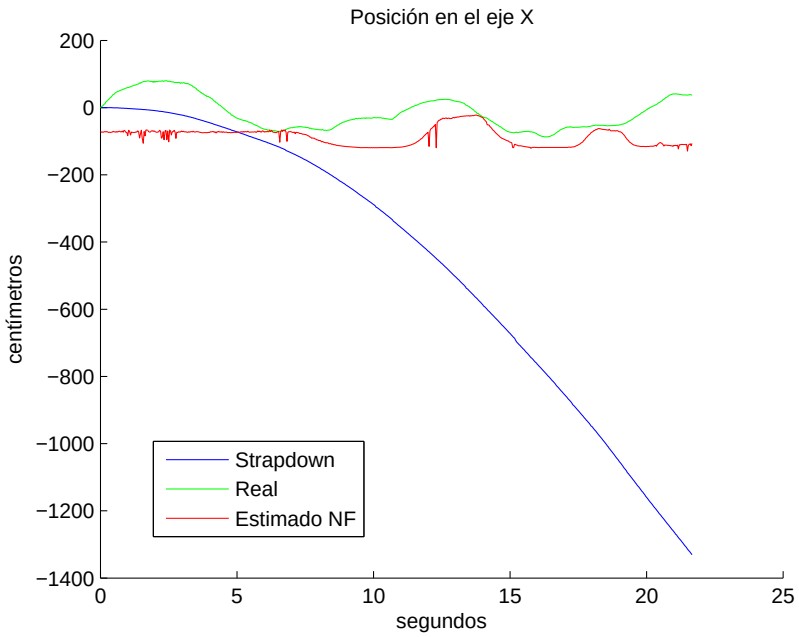


Figura 6.21.Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje X

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.34 cm.
Error máximo	1367.40 cm.	183.21 cm.
Error medio	441.90 cm.	71.77 cm.

Tabla 6.4. Errores en el eje X del sistema neuro-fuzzy vs. strapdown.

- Eje Y

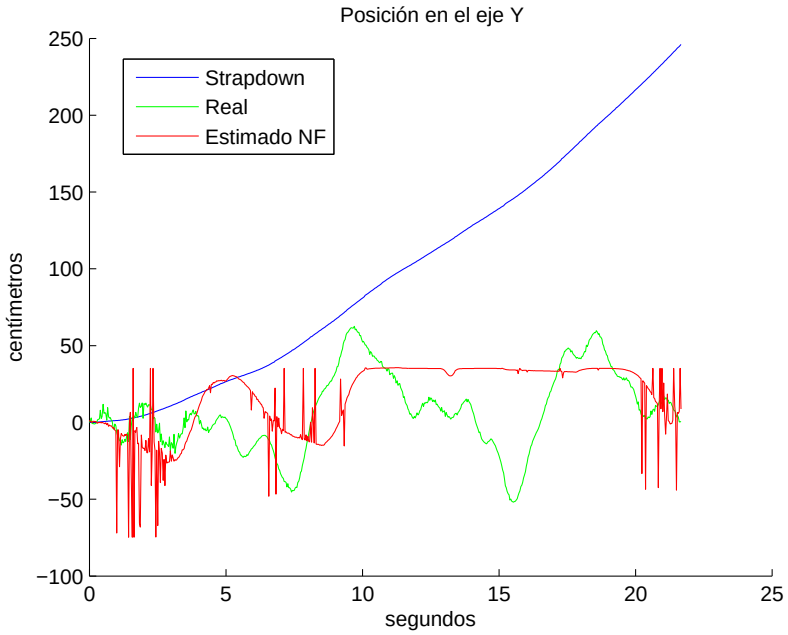


Figura 6.22. Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje Y

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.14 cm.
Error máximo	245.46 cm.	86.07 cm.
Error medio	90.68 cm.	24.43 cm.

Tabla 6.5. Errores en el eje Y del sistema neuro-fuzzy vs. strapdown.

- Eje Z

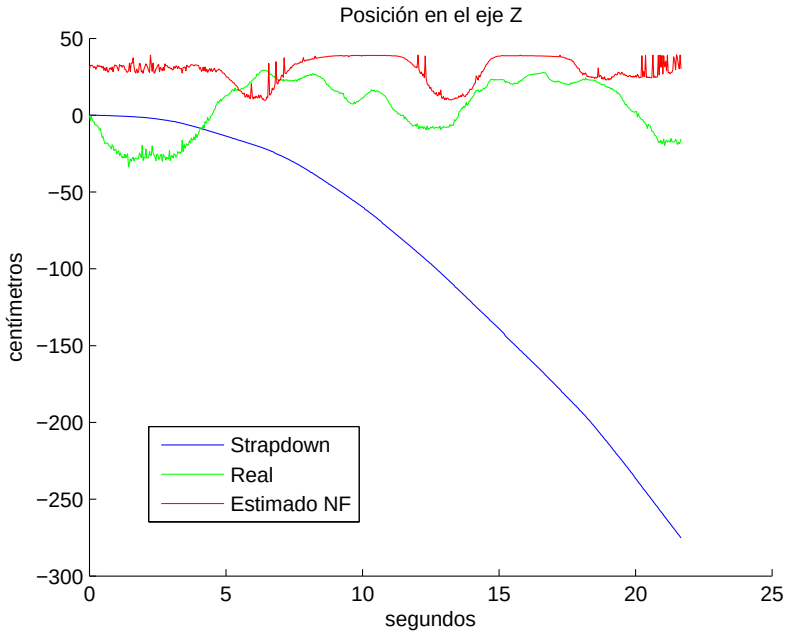


Figura 6.23. Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje Z

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.23 cm.
Error máximo	259.92 cm.	65.22 cm.
Error medio	106.23 cm.	25.62 cm.

Tabla 6.6. Errores en el eje Z del sistema neuro-fuzzy vs. Strapdown.

Como se puede observar en las gráficas, el ajuste entre las posiciones estimadas por la red y las reales dista de ser perfecto. Sin embargo, se mantiene en unos valores razonables frente al uso de la navegación puramente inercial. Además, de las gráficas se puede extraer la conclusión de que las rutas calculadas están libres de deriva.

Por otro lado, se ha usado como entrada la posición calculada con el algoritmo strapdown, que como bien se puede observar en las figuras anteriores, está definida por una función monótona decreciente (figura 6.21 y figura 6.23) o monótona creciente (figura 6.22). Esta peculiaridad podría estar causando que la red durante el entrenamiento estableciese una relación unívoca entre los valores X, Y y Z de entrada con los valores de salida entrenados, ya que al no repetirse valores en la entrada, podría ignorar los demás parámetros facilitados al sistema. Si tenemos en cuenta que los valores del subconjunto de validación y el subconjunto de entrenamiento se escogen del mismo conjunto de datos, siendo estos consecutivos, se pueden plantear dudas razonables sobre la idoneidad del método escogido frente a la capacidad de generalización de esta red.

6.3.3.2 Posiciones relativas

Viendo los resultados comentados en el apartado anterior, se decidió realizar un segundo ensayo. Esta vez se emplearon incrementos de posición, en vez de las posiciones absolutas. Dado que los incrementos corresponden a la primera integral de la aceleración, es decir, a la velocidad, cabe esperar que la deriva no sea tal que haga que constituyan una función monótona creciente o decreciente. Para confirmar esta hipótesis, se puede observar en la siguiente figura los incrementos de posición en cada eje.

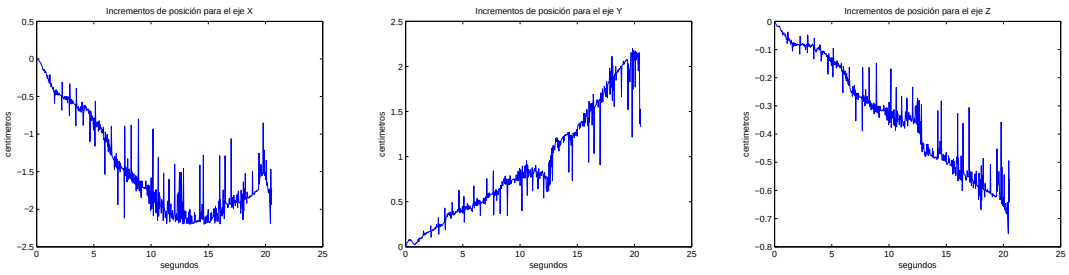


Figura 6.24. Incrementos de posición en cada eje

Los vectores de entrada al sistema son de la forma:

$$U = (t, r_1, \dots, r_{10}, \theta_1, \dots, \theta_{10}, d\hat{x}, d\hat{y}, d\hat{z}, q_w, q_x, q_y, q_z) \quad \textbf{(6.2)}$$

donde:

- t indica el tiempo de la captura en segundos.
- $\theta_1, r_1, \dots, \theta_{10}, r_{10}$ son los vectores de flujo óptico expresados en forma polar.

- $d\hat{x}, d\hat{y}, d\hat{z}$ son los incrementos de posición estimados, calculados como las diferencias entre las posiciones determinadas con el algoritmo strapdown.
- q_w, q_x, q_y, q_z son las componentes del quaternion de orientación del sensor Mtx.

6.3.3.2.1 Entrenamiento

- Eje X

Para el entrenamiento en el eje X se emplearon 616 patrones, de los cuales 431 (70%) formaron el conjunto de entrenamiento y 185 (30%) el de validación. El entrenamiento se llevó a cabo con 10000 épocas en su etapa no supervisada y 2200 épocas en la etapa supervisada. En la figura 6.25 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.7 se muestran los errores mínimos alcanzados durante el entrenamiento.

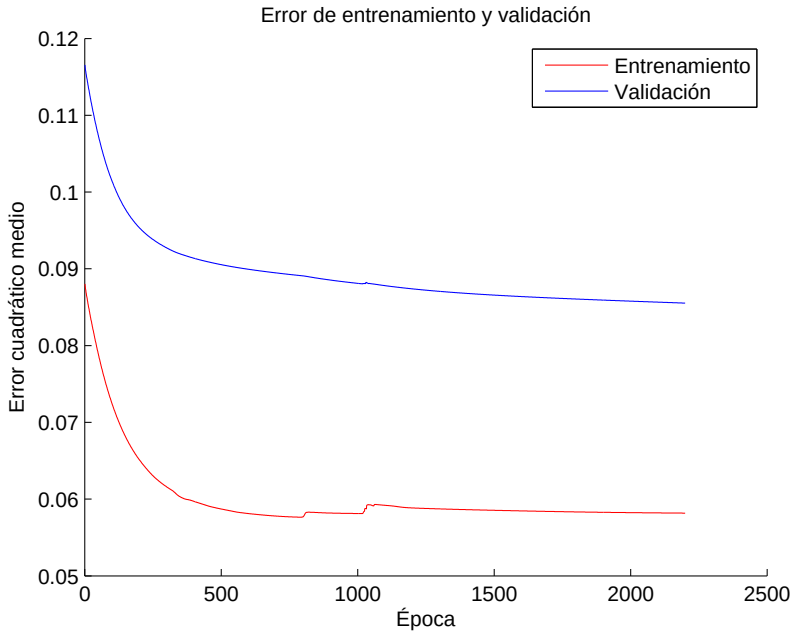


Figura 6.25. Gráfica de errores durante el entrenamiento para el eje X

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.058	0.085

Tabla 6.7. Errores mínimos del entrenamiento para el eje X

- Eje Y

Para el entrenamiento en el eje Y se emplearon 553 patrones, de los cuales 387 (70%) formaron el conjunto de entrenamiento y 166 (30%) el de validación. El

entrenamiento se llevó a cabo con 20000 épocas en su etapa no supervisada y 170 épocas en la etapa supervisada. En la figura 6.26 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.8 se muestran los errores mínimos alcanzados durante el entrenamiento.

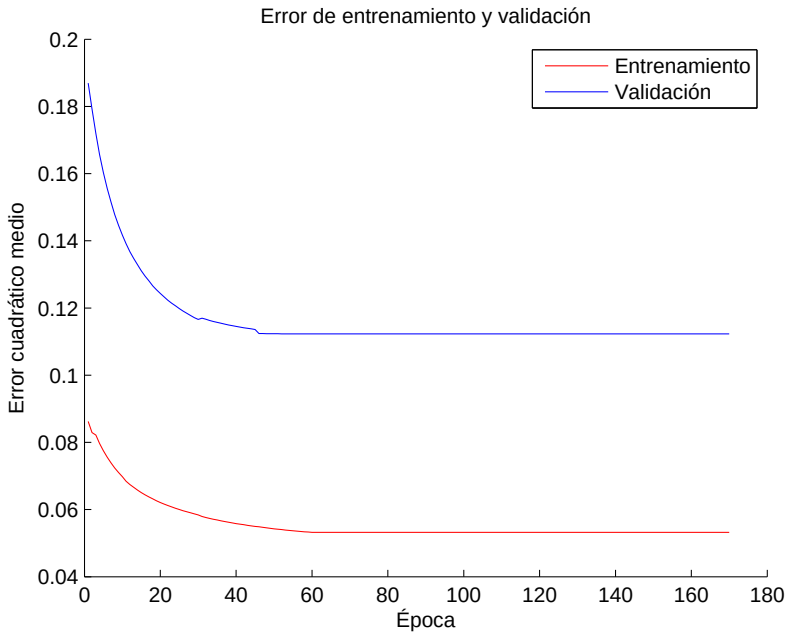


Figura 6.26. Gráfica de errores durante el entrenamiento para el eje Y

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.0532	0.1123

Tabla 6.8. Errores mínimos del entrenamiento para el eje Y

- Eje Z

Para el entrenamiento en el eje Z se emplearon 496 patrones, de los cuales 347 (70%) formaron el conjunto de entrenamiento y 149 (30%) el de validación. El entrenamiento se llevó a cabo con 20000 épocas en su etapa no supervisada y 1550 épocas en la etapa supervisada. En la figura 6.27 se puede observar la evolución del error de entrenamiento frente al de validación. En la tabla 6.9 se muestran los errores mínimos alcanzados durante el entrenamiento.

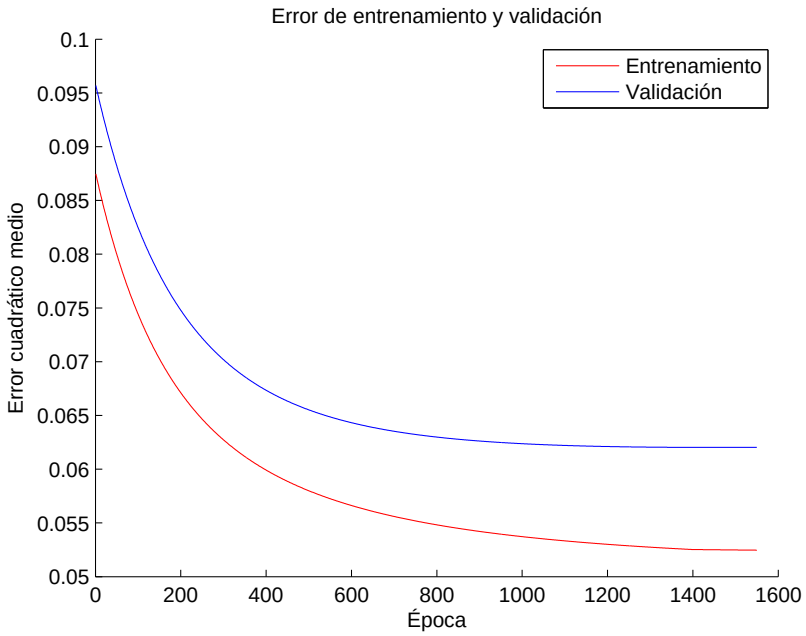


Figura 6.27. Gráfica de errores durante el entrenamiento para el eje Z

	Entrenamiento	Validación
Error cuadrático medio mínimo	0.0525	0.0620

Tabla 6.9. Errores mínimos del entrenamiento para el eje Z

6.3.3.2.2 Resultados

Al igual que en el apartado 6.3.3.1.2 se mostrará una ruta de ejemplo para cada eje, acompañándola de una tabla resumen de los errores cometidos. La ruta de test, igual que en el caso anterior, se corresponde a 651 muestras (21,7 segundos)

- Eje X

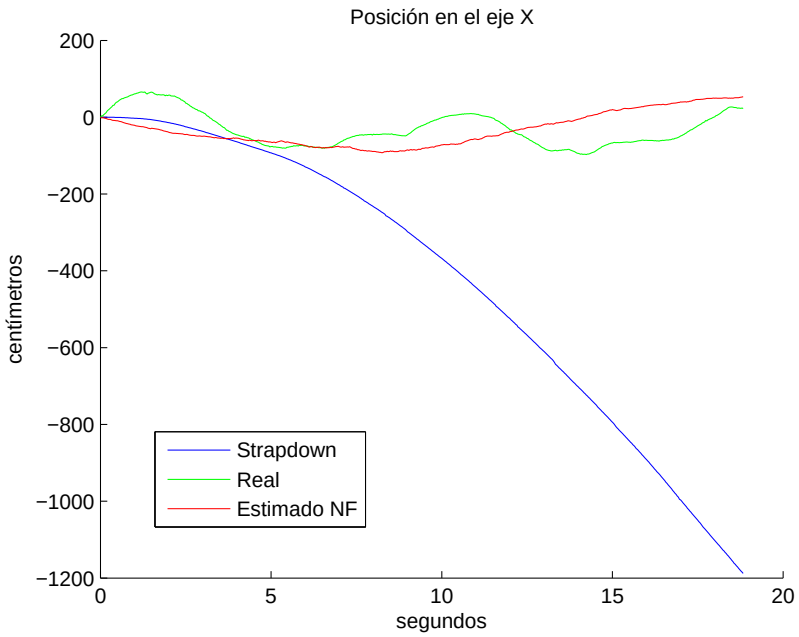


Figura 6.28. Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje X

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.03 cm.
Error máximo	1367.40 cm.	99.18 cm.
Error medio	441.90 cm.	52.48 cm.

Tabla 6.10. Errores en el eje X del sistema neuro-fuzzy vs. strapdown.

- Eje Y

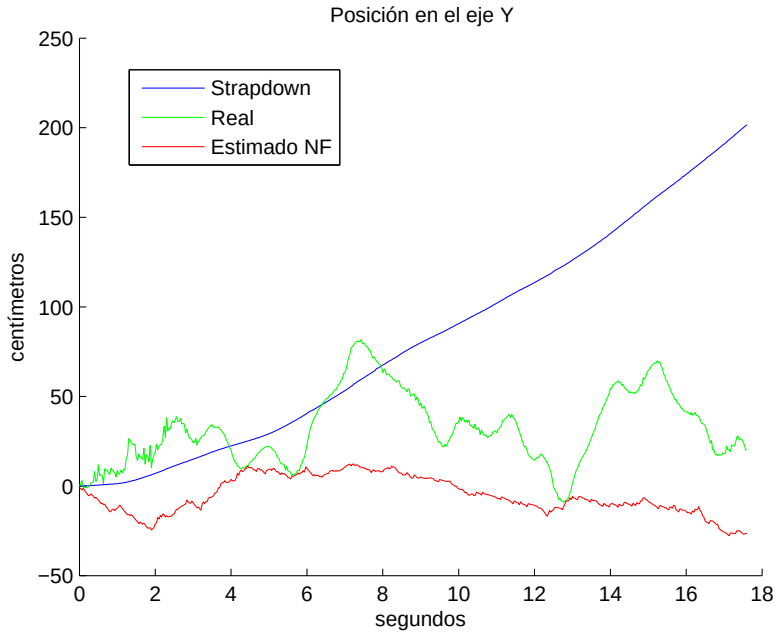


Figura 6.29. Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje Y

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.16 cm.
Error máximo	245.46 cm.	81.87 cm.
Error medio	90.68 cm.	37.41 cm.

Tabla 6.11. Errores en el eje Y del sistema neuro-fuzzy vs. strapdown solamente.

- Eje Z

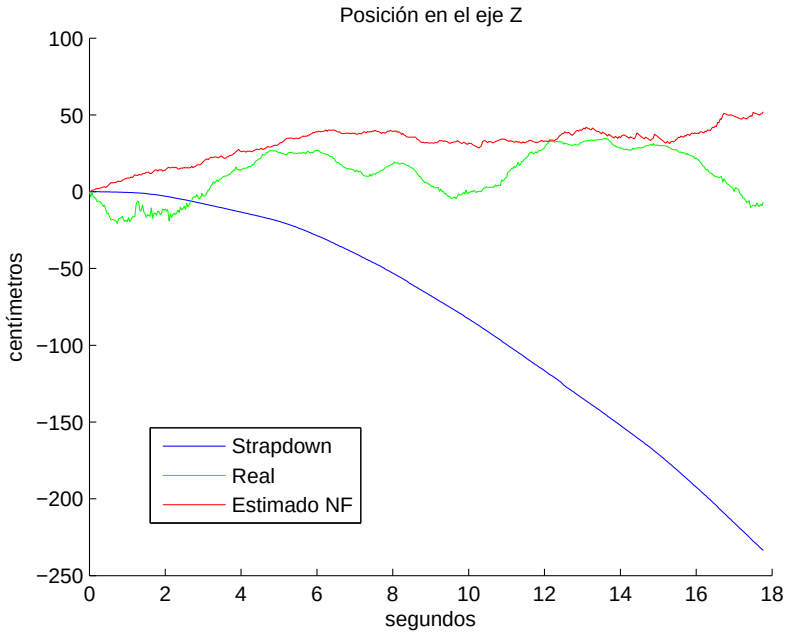


Figura 6.30. Ruta estimada por el sistema neuro-fuzzy vs strapdown en el eje Z

	Strapdown	Neuro-Fuzzy
Error mínimo	0.00 cm.	0.02 cm.
Error máximo	259.92 cm.	60.45 cm.
Error medio	106.23 cm.	19.16 cm.

Tabla 6.12. Errores en el eje Z del sistema neuro-fuzzy vs. strapdown solamente.

A continuación se muestran tres tablas donde se comparan los errores cometidos en el cálculo de posición empleando los métodos de posiciones absolutas y relativas frente al algoritmo strapdown. Como se puede observar, el empleo de posiciones relativas a la hora de entrenar y usar la red mejora los resultados de error tanto en el eje X, donde la mejora es sustancial, como en el eje Z, donde la reducción del error es menor. Sin embargo, en el caso del eje Y el error aumenta ligeramente. Esto puede ser debido a las contribuciones de alta frecuencia que se pueden observar en la figura 6.22, donde los valores que no siguen la tendencia pueden acercarse puntualmente a los valores de la posición real, contribuyendo a reducir el error en el cálculo del error medio.

	Strapdown	N-F pos. absolutas	N-F pos. relativas
Error mínimo	0.00 cm.	0.34 cm.	0.03 cm.
Error máximo	1367.40 cm.	183.21 cm.	99.18 cm.
Error medio	441.90 cm.	71.77 cm.	52.48 cm.

Tabla 6.13. Errores de posición empleando posiciones relativas y absolutas para el eje X

	Strapdown	N-F pos. absolutas	N-F pos. relativas
Error mínimo	0.00 cm.	0.14 cm.	0.16 cm.
Error máximo	245.46 cm.	86.07 cm.	81.87 cm.
Error medio	90.68 cm.	24.43 cm.	37.41 cm.

Tabla 6.14. Errores de posición empleando posiciones relativas y absolutas para el eje Y

	Strapdown	N-F pos. absolutas	N-F pos. relativas
Error mínimo	0.00 cm.	0.23 cm.	0.02 cm.
Error máximo	259.92 cm.	65.22 cm.	60.45 cm.
Error medio	106.23 cm.	25.62 cm.	19.16 cm.

Tabla 6.15. Errores de posición empleando posiciones relativas y absolutas para el eje Z

7 CONCLUSIONES

- En esta tesis se han abordado en particular dos problemas muy relacionadas con la medida de aceleraciones: La medición de vibraciones al objeto de determinar fallos en componentes mecánicos y la aminoración de derivas en la estimación de la posición a través de sistemas de navegación inercial. Este último problema tiene una importancia vital en la aplicación de otros algoritmos que se basan en una buena estimación de la posición.

- En ambos casos se ha partido del uso de acelerómetros microelectromecánicos (MEMS – Microelectromechanical system) cuyas características de bajo coste y capacidad de miniaturización los hacen idóneos para diferentes aplicaciones, siempre y cuando se puedan minimizar los errores a valores adecuados para la aplicación concreta de estos sistemas.

- Inicialmente, se ha tratado el problema del diagnóstico de fallos en rodamientos, focalizando la atención en los resultados obtenidos al aplicar un esquema bastante extendido como es el sistema neuro-fuzzy ANFIS (Adaptive Neuro-Fuzzy Inference Systems) y el esquema alternativo indicado en la tesis. La aplicación de estos esquemas a este problema ha revelado como el esquema neuro-fuzzy alternativo presenta buenos resultados en este contexto y sobre todo incorpora características innovadores respecto al sistema neuro-fuzzy ANFIS que se ponen de manifiesto claramente cuando el número de entradas al sistema se va incrementando.

- Vistas las propiedades del sistema neuro-fuzzy alternativo a través del problema del diagnóstico de rodamientos defectuosos, este esquema ha sido aplicado al problema de la corrección de la deriva en la estimación de la posición de los sistema de navegación inercial. Para ello, se ha utilizado un sistema de navegación inercial basado en el uso de acelerómetros MEMS y una cámara de vídeo con el fin de fusionar la información de ambos y mejorar la estimación de la posición utilizando dispositivos de bajo costo.

- Concretamente, se ha adaptado la información disponible procedente de estas fuentes al esquema neuro-fuzzy alternativo. Para posteriormente realizar varios ensayos en contextos experimentales en donde se han obtenido resultados satisfactorios.

- Se han evaluado dos aproximaciones al problema de la deriva, por un lado la determinación de la corrección de la derivada utilizando valores absolutos y por otro mediante una aproximación basada en valores incrementales. Tanto en un caso como otro los resultados han sido satisfactorios en relación a la metodología del strapdown clásico, aunque la aproximación incremental parece más adecuada dada su independencia de configuraciones particulares que pueden propiciar de manera artificial el aprendizaje.

8 APÉNDICE A: ESPECIFICACIONES TÉCNICAS

8.1 Polhemus Fastrak

POLHEMUS
INNOVATION IN MOTION™

FASTRAK

THE FAST AND EASY DIGITAL TRACKER

THE INDUSTRY STANDARD

The most accurate electromagnetic motion tracking system available, FASTRAK® is the perfect solution for accurately computing position and orientation through space. With real-time, 6 Degrees-of-Freedom (6DOF) tracking and virtually no latency, this award-winning system is ideal for head, hand, and instrument tracking, as well as biomedical motion and limb rotation, graphic and cursor control, stereotaxic localization, telerobotics, digitizing, and pointing. FASTRAK has been the workhorse of the industry since its introduction.

FEATURES

Real Time
Virtually no latency. Digital Signal Processing (DSP) technology provides 4ms latency updated at 120 Hz. Data is transmitted via USB or RS-232 to the host at up to 115.2 K Baud.

Improved Accuracy and Resolution
Accuracy of 0.03 inches RMS with a resolution of 0.0002 inches per inch makes this the most precise device of its kind.

Range
Standard operational range is 4 to 6 feet; 10 foot range is obtainable. The optional TX4 or Long Ranger™ transmitters allow significantly longer ranges of operation.

Reliable
The pioneer of 3D position/orientation measuring devices, in business since 1970. Real-time self-calibration ensures the unit never needs adjustment.

Multiple Output Formats
Position in Cartesian coordinates (inches or centimeters); orientation in direction cosines, Euler angles, or quaternions.

Angular Coverage
The receivers are all-attitude with no limits.

Drift-Free
Solid state electronics.

Two Solutions in One
FASTRAK is a 3D digitizer and a quad receiver motion tracker, making it perfect for a wide range of applications requiring high resolution, accuracy, and range. By computing the position and orientation of a small receiver as it moves through space, it provides dynamic, real-time measurements of position (X, Y, and Z Cartesian coordinates) and orientation (azimuth, elevation, and roll).

Up to Four Receivers
FASTRAK comes standard with a Microsoft Windows® GUI. With a single transmitter, FASTRAK accepts data from up to four receivers. Because FASTRAK uses proprietary low-frequency magnetic transducing technology, there's no need to maintain a clear line-of-sight between receivers and transmitters.

A/C Magnetics
Quiet and stable, the system is essentially unaffected by facility power grids. Update rates are always maintained, as A/C magnetics offer the best signal-to-noise ratios and incorporate sophisticated digital signal processing capabilities. In addition, adaptive filtering is available as a standard feature.

APPLICATIONS

Limited only by your imagination!

FASTRAK

COMPONENTS

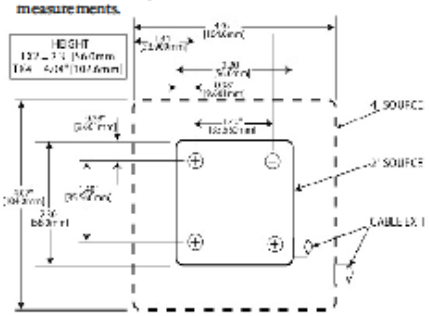
The FASTRAK system includes a System Electronics Unit (SEU), a power supply, one receiver, and one transmitter. You can easily expand the system's capabilities by adding up to three additional receivers.

System Electronics Unit

Contains the hardware and software necessary to generate and sense the magnetic fields, compute position and orientation, and interface with the host computer via USB, RS-232 or optional RS-422.
10.2 in. (25.9 cm) L x 11.5 in. (29.2 cm) W x 2.3 in. (5.8 cm) H

Source

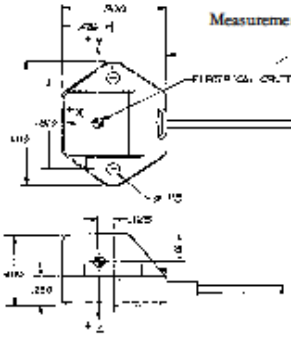
The source is the system's reference frame for sensor measurements.



Weight
TX2: 8.8 oz. (250 gm) **Thread size** 1/4" x 20
TX4: 1.60 lbs. (726gm) **Thread size** 1/4" x 20

Sensor

A lightweight, small cube, the sensor's position and orientation is precisely measured as it is moved.



Weight
0.32 oz. (9.1 gm)

POLHEMUS
INNOVATION IN MOTION™

The systems are not certified for medical or bio-medical use. Any reliance by medical or bio-medical use overcomplies of what medical companies have done with the systems after obtaining all necessary or appropriate medical certifications. Those of use/OEM must comply with all pertinent FDA/CE and all other regulatory requirements.

SPECIFICATIONS

Update Rate

120 updates/second divided by the number of receivers

Latency

4 milliseconds

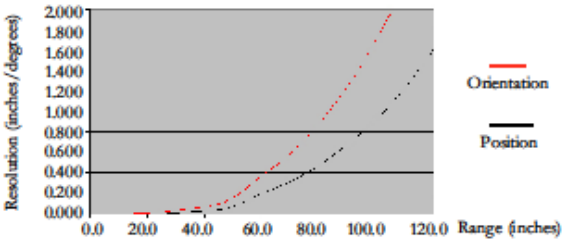
Static Accuracy

0.03 inches RMS for the X, Y, or Z position; 0.15° RMS for receiver orientation. The system will provide the specified performance when the receivers are within 30 inches of the transmitter. Operation over a range of up to 10 feet is possible with slightly reduced performance.

Interface

USB; RS-232 with selectable baud rates up to 115.2 K (optional RS-422)

Resolution vs. Range



Range (inches)	Position Resolution (inches)	Orientation Resolution (degrees)
12.0	0.00023	0.0026
24.0	0.0030	0.0147
36.0	0.019	0.0558
48.0	0.055	0.1266
72.0	0.346	0.369
120.0	1.605	2.960

Sync Input

TTL and CRT

Software Tools

GUI included
USB drivers for Microsoft Windows® XP/Vista included/Win7 included (32-bit and 64-bit). Linux®- open-source application available

Operating Temperature

10°C to 40°C at a relative humidity of 10% to 95%, noncondensing

Power Requirements

25 W, 100-240 VAC, 47-63 Hz

Regulatory

FCC Part 15, class A
CE: EN61326-1: 1997/A1:1998/A2:2001/A3:2003 emission
EN61326-1: 1997/A1:1998/A2:2001/A3:2003 Immunity

* Large metallic objects, such as desks or cabinets, located near the transmitter or receiver, may adversely affect the performance of the system.

www.polhemus.com
40 Hercules Drive • PO Box 560 • Colchester, Vermont 05446-0560
US and Canada 800.357.4777 • 802.655.3159 • fax 802.655.1439

FASTRAK is a registered trademark of Polhemus. Microsoft Windows is a registered trademark of Microsoft Corp. Linux is a registered trademark of Linus Torvalds.
Copyright © 2008 Polhemus FT: MS028 Rev. June 2011

8.2 Xsens Mtx

The MTx is a small and accurate 3DOF Orientation Tracker. It provides drift-free 3D orientation as well as kinematic data: 3D acceleration, 3D rate of turn (rate gyro) and 3D earth-magnetic field. The MTx is an excellent measurement unit for orientation measurement of human body segments and other applications requiring very low profile and light-weight sensor units.

PRODUCT OVERVIEW

Features

- Accurate full 360 degrees 3D orientation output
- Highly dynamic response combined with long-term stability (no drift)
- 3D acceleration, 3D rate of turn and 3D earth-magnetic field data
- Compact design
- High update rate
- Accepts synchronization pulses
- All solid state miniature MEMS inertial sensors inside
- Individually calibrated for temperature, 3D misalignment and sensor cross sensitivity

Fields of use

- Biomechanics
- Rehabilitation
- Sports science
- Virtual reality
- Ergonomics
- Animations

The MTx uses 3 rate gyros to track rapidly changing orientations in 3D and it measures the directions of gravity and magnetic north to provide a stable reference. The system's real-time algorithm fuses the sensor information to calculate accurate 3D orientation, with a highly dynamic response which remains stable over prolonged periods. With the MTx Software Development Kit, the MTx can be easily integrated in any system or (OEM) application.

A standalone MTx is available, as well an Xbus version. With the Xbus Master, Xsens' digital data bus, multiple MTx's can easily be used simultaneously, enabling ambulatory and cost-effective measurements of human body motion.

MTx TECHNICAL SPECIFICATIONS

Output
3D orientation (Quaternions/Matrix/Euler angles)
3D acceleration
3D rate-of-turn
3D earth-magnetic field (normalized)
Temperature

Orientation performance

Dynamic Range	all angles in 3D
Angular Resolution[1]	0.05 deg
Static Accuracy (Roll/Pitch)	<0.5 deg
Static Accuracy[2] (Heading)	<1 deg
Dynamic Accuracy[3]	2 deg RMS

Sensor performance

	Rate of turn	Acceleration	Magnetic field
Dimensions	3 axes	3 axes	3 axes
Full Scale (standard)	± 1200 deg/s	± 50 m/s²	± 750 mGauss
Linearity	0.1% of FS	0.2% of FS	0.2% of FS
Bias stability[4]	° deg/s	0.02 m/s²	0.1mGauss
Scale Factor stability[4]	-	0.03%	0.5%
Noise	0.05 deg/s/ √Hz	0.002 m/s²/√Hz	0.5 mGauss
Alignment error	0.1 deg	0.1 deg	0.1 deg
Bandwidth	40 Hz	30 Hz	10 Hz
Max update rate	512 Hz	512 Hz	512 Hz

Interfacing

Max. update rate	512 Hz (calibrated sensor data) 120 Hz (orientation data)
Operating voltage[5]	4.5 - 30 V
Power consumption	360 mW (orientation output)
Digital interface (standard)	RS-232 and USB (external converter) or 'Xbus'

Housing

Dimensions	38x53x21 mm (WxLxH)
Weight	30 g
Ambient temperature operating range[6]	-20... +55 °C
Specified performance operating range[4]	0.. +55 °C

Options and product code

Interface:
RS-232 (RS-232, sync in) 28
RS-485 (RS 485) 48
Xbus 49
(two connectors, only to be used with Xbus Master)

Full Scale Acceleration:
5g (50 m/s²) A53
18g (180 m/s²) A83

Full Scale Rate of Turn:
300 deg/s G35
1200 deg/s G25

Other options on request. Surcharges may apply.

Product code: MTx-## A## G##
Standard version: MTx-28 A53 G25
Standard Xbus version: MTx-49 A53 G25

[1] 1σ standard deviation of zero-mean angular random walk
[2] in homogenous magnetic environment
[3] may depend on type of motion
[4] deviation over operating temperature range (1σ) specifications subject to change without notice
[5] only valid for MTx's with device ID's > 2000, other units operate on 4.5 - 16 V max
[6] non-condensing environment

xsens

8.3 Acelerómetro Brüel & Kjaer 4383

PRODUCT DATA

Piezoelectric DeltaShear® Accelerometers Uni-Gain®, DeltaTron® and Special Types

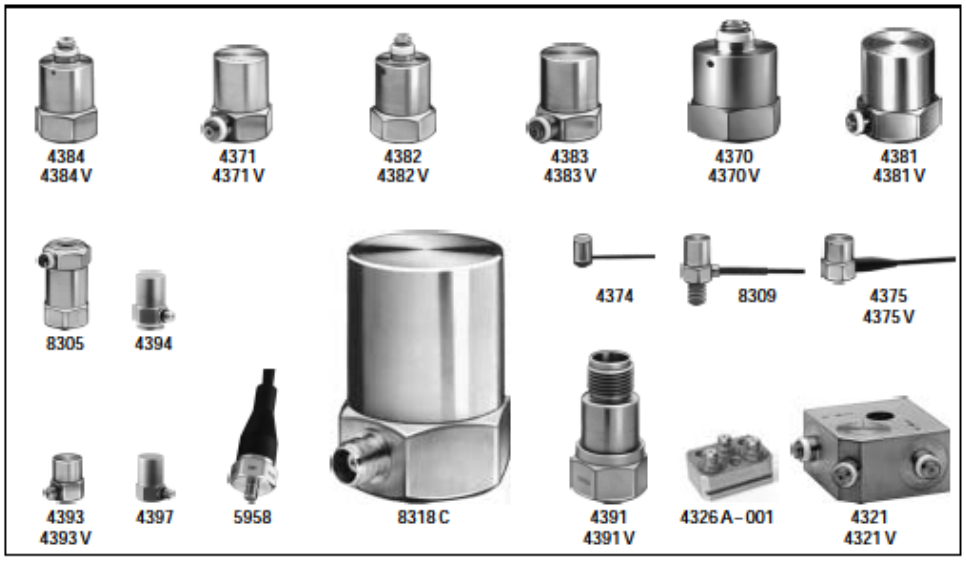
"V" Types: 4321 V, 4370 V, 4371 V, 4375 V, 4381 V, 4382 V, 4383 V, 4384 V, 4391 V and 4393 V
Uni-Gain Types: 4321, 4370, 4371, 4375, 4381, 4382, 4383, 4384, 4391 and 4393
Uni-Gain DeltaTron Types: 4394 and 4397
Special Types: 4326 A, 4326 A-001, 4374, 8305, 8309, 8318 C and 5958

USES

- Shock and vibration measurement and analysis
- Vibration monitoring
- Modal and structural analysis
- Vibration test control
- Production and quality control

FEATURES

- Competitively priced DeltaShear "V" Types, especially suitable for permanent setups
- Uni-Gain types for easy interchangeability
- DeltaTron types with integral preamplifier
- Acceleration ranges cover 20 μms^{-2} to 1000 kms^{-2}
- Frequency ranges cover from a fraction of a Hz to 60 kHz (+10% limit)
- Temperature ranges cover −74° to +250°C (−101 to +482°F)
- Low sensitivity to extraneous environmental influences including temperature fluctuations
- Low sensitivity to base bending effects
- Individual calibration supplied
- Artificially aged for long-term stability



Accelerometers

Brüel & Kjær

The Brüel & Kjær transducer range incorporates accelerometers suitable for most application requirements. In addition to the comprehensive range of piezoelectric accelerometers described in this Data Sheet, Brüel & Kjær supply accelerometers for heavy-duty industrial use and transducers specifically designed for special-purpose applications.

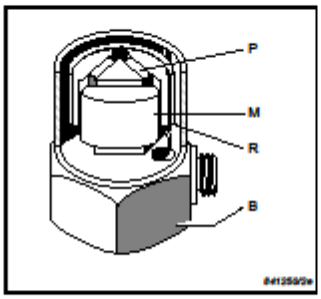


Fig. 1
The unique Brüel & Kjær DeltaShear design. M=Seismic Mass, P=Piezoelectric Element, B=Base and R=Clamping Ring

The active element of Brüel & Kjær accelerometers consists of piezoelectric discs or slices loaded by seismic masses and held in position by a clamping arrangement. When the accelerometer is subjected to vibration, the combined seismic mass exerts a variable force on the piezoelectric element. Due to the piezoelectric effect, this force produces a corresponding electrical charge.

For frequencies from DC up to approximately one third of the resonance frequency of the accelerometer assembly, the acceleration of the seismic mass is equal to the acceleration of the whole transducer. Consequently, the charge produced by the piezoelectric element is proportional to the acceleration to which the transducer is subjected.

The electrical signal output from Brüel & Kjær accelerometers is self-generated, though the types with built-in preamplifiers require an external power supply for this signal to be measured.

All the piezoelectric accelerometer types described in this Product Data sheet are supplied with an individual calibration chart and, in most cases, an individually measured frequency-response curve. Data from these charts are summarised in the Specifications.

"V" and Uni-Gain Types

Some of the piezoelectric accelerometers described in this Product Data sheet are available both as "V" types as well as Uni-Gain types. The DeltaShear without Uni-Gain types are recognized by the "V" suffix in the type name. The only difference between these two types is that all the specifications on the calibration chart for "V" types, except the sensitivity, are typical. In contrast the sensitivity and other parameters for the Uni-Gain accelerometers are guaranteed within tight tolerances for easy interchangeability without recalibration (see specifications on pages 20 and 21). Except for the sensitivity, everything in this Product Data applies to both types.

Uni-Gain Sensitivity

This designation indicates that the measured accelerometer sensitivity has been adjusted during manufacture to within 2% of a convenient value, for example (in 10 dB steps), 1, 3.16 or 10 pC/ms^{-2}.

Design and Construction

Fig. 2
Most accelerometers are supplied in a robust plastic box

All accelerometers, except Types 4321, 4321 V and 4326, measure uniaxial acceleration. These types measure acceleration in three mutually perpendicular directions.

With the exception of Triaxial Accelerometer Type 4326, Miniature Accelerometer Type 4374, Standard Reference Accelerometer Type 8305 and Shock Accelerometer Type 8309, all piezoelectric accelerometers in this data sheet use the DeltaShear design (see Fig. 1). Type 4374 uses the planar shear design, Type 8305 uses the inverted centre-mounted compression design and Type 8309 uses the centre-mounted compression design.

The piezoelectric elements of most of the accelerometers are PZ 23 lead zirconate titanate elements. The Shock Accelerometer Type 8309 has a specially formulated ferroelectric ceramic PZ 45. Miniature Accelerometer Type 4374 has a lead zirconate titanate element PZ 27.

The housing material of all the accelerometers is the same as the base material (given in the Specifications) except Type 4374, which has a nickel-chromium alloy housing.

Characteristics

Charge and Voltage Sensitivity

A piezoelectric accelerometer may be treated as a charge or voltage source. Its sensitivity is defined as the ratio of its output to the acceleration it is subjected to, and may be expressed in terms of charge per unit acceleration (e.g. pC/ms^{-2}) or in terms of voltage per unit acceleration (e.g., mV/ms^{-2}).

The sensitivities given in the individual calibration charts have been measured at 160 Hz with an acceleration of $100\,ms^{-2}$. For a 99.9% confidence level, the accuracy of the factory calibration is ±2% and includes the influence of the connecting cable supplied with each accelerometer. With the exception of Triaxial Accelerometers Types 4321, 4321 V, 4326 A and 4326 A–001, the direction of main axis sensitivity for these accelerometers is perpendicular to the base plane of the accelerometers. Types 4321, 4321 V, 4326 A and 4326 A–001 have three mutually perpendicular axes of sensitivity.

DeltaShear Accelerometers

The Delta design involves three piezoelectric elements and three masses arranged in a triangular configuration around a triangular centre post, as illustrated in Fig. 1. The Delta Shear design gives a high sensitivity-to-mass ratio compared to other designs, a relatively high resonance frequency and high isolation from base strains and temperature transients. The excellent overall characteristics of this design make it ideal for both general purpose accelerometers and more specialised types.

DeltaTron Accelerometers

DeltaTron accelerometers operate on a constant-current power supply and give output signals in the form of voltage modulation on the power supply line. Types 4394 has an insulated base. All DeltaTron accelerometers are individually calibrated Uni-Gain types.

Fig. 3 *Upper and lower frequency limits (10%) and sensitivities of accelerometers. † denotes a DeltaTron type where the sensitivity is given in mV/ms^{-2}. Frequency limits also apply to "V" types*

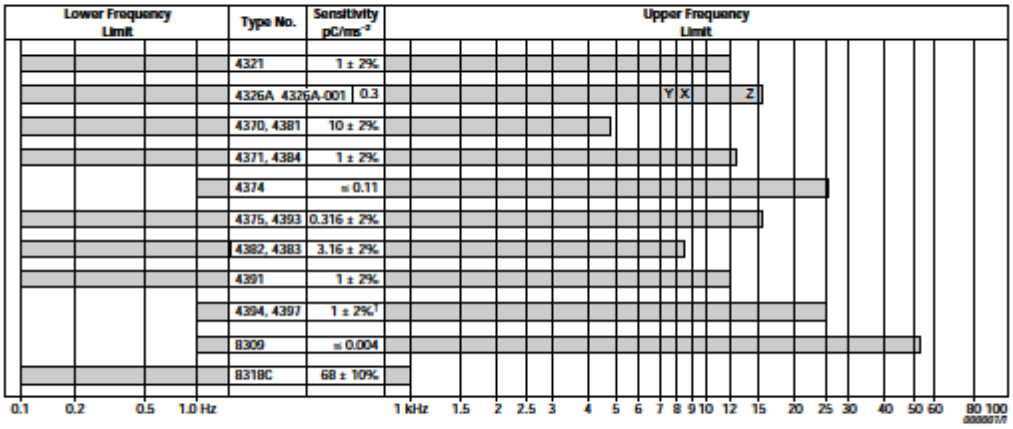
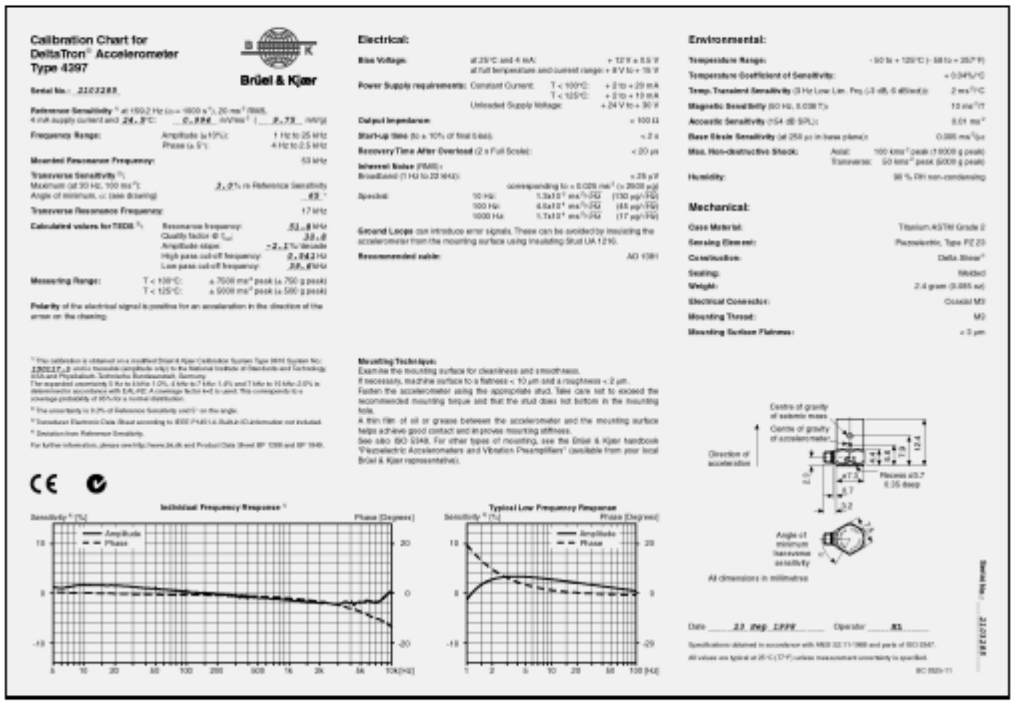


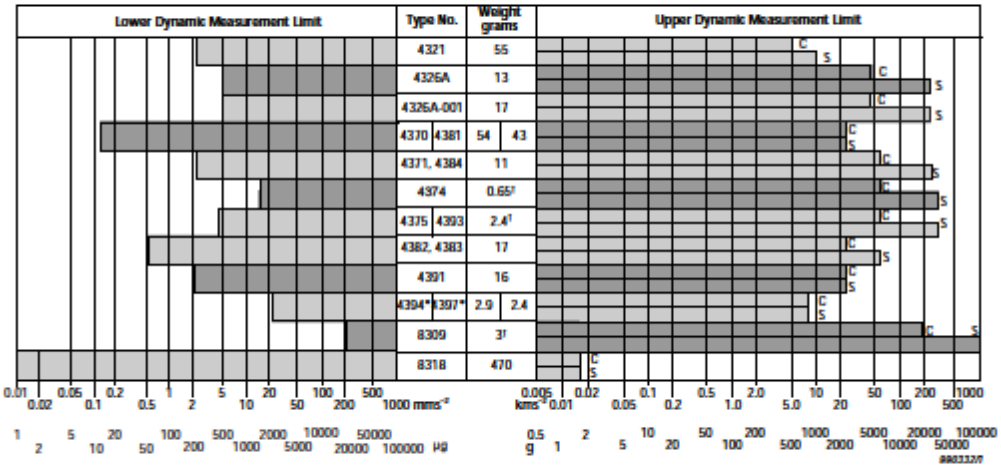
Fig. 4 *Example of the calibration chart supplied with Brüel & Kjær accelerometers*



Transverse Sensitivity

Accelerometers are slightly sensitive to acceleration normal to their main sensitivity axis. This transverse sensitivity is measured during the factory calibration process using a 30 Hz and 100 ms^{-2} excitation, and is given as a percentage of the corresponding main axis sensitivity.

Fig. 5 *Upper and lower dynamic measurement limits and weights of the accelerometers. Maximum limits (C = continuous sinusoidal vibration and S = shock) are peak values. Minimum limits (L = Lin 1 Hz to individual accelerometer + 10% upper frequency limit) are RMS values. The dynamic limits are typical residual noise on either DeltaTron types or charge types plus Brüel & Kjær Conditioning Amplifier Type 2692. † denotes cable weight excluded. * Upper limit for shock is measured in the axial direction. Limits also apply to "V" types*



Most Uni-Gain DeltaShear types have an indication of the angle of minimum transverse sensitivity.

Frequency Response

The upper frequency limits given in the specifications are calculated as 30% and 22% of the mounted resonance frequency to give errors of less than 10% and 5% respectively. These calculations assume that the accelerometer is properly fixed to the test specimen, as poor mounting can have a marked effect on the mounted resonance frequency.

The low-frequency response of an accelerometer depends primarily on the type of preamplifier used in the measurement setup. A detailed discussion of the effects of the measuring system on the low-frequency response of an accelerometer is given in the Brüel & Kjær "Piezoelectric Accelerometers and Vibration Preamplifiers Handbook".

All the standard piezoelectric accelerometer types are supplied with an individual calibration chart. With the exception of Types 4374, 4326 A, 4326 A–001 and all V-types, all types have individually measured frequency response curves.

DeltaTron types are supplied with individual frequency curves from 5 to 10000 Hz as well as typical curves below this range.

Transverse Resonance Frequency

Typical values for the transverse resonance frequency are obtained by vibrating the accelerometers mounted on the side of a steel or beryllium cube using Calibration Exciter Type 4290.

Phase Response and Damping

The low damping of Brüel & Kjær accelerometers leads to the single, well-defined resonance peak plotted on the individual frequency-response curves. Brüel & Kjær accelerometers can be used at frequencies up to 30% of their mounted resonance frequency

9 APÉNDICE B: CÓDIGO FUENTE DE LOS PROGRAMAS

9.1 Software de Captura

9.1.1 ExtendeVideoInput.h

```
#ifndef _EXTENDEDVIDEOINPUT
#define _EXTENDEDVIDEOINPUT

#include "all_includes.h"

class IMediaSeeking;

class extendedVideoInput: public videoInput {
private:
  int             numFilesAsDevicesConfigured;
  int             avgTimePerFrame;
  IMediaSeeking*  mediaSeeking;
  bool            initialized;

  bool setup(int deviceNumber, const WCHAR *fileName,
             bool initRunning);
  int startDevice(int deviceID, videoDevice * VD,
             const WCHAR *fileName, bool initRunning);
  int  startFileAsDevice(int deviceID, videoDevice * VD,
             const WCHAR *fileName, bool disableSync,
             bool initRunning, bool oneShot);
public:
  extendedVideoInput();
  ~extendedVideoInput();
  using videoInput::setupDevice;
  bool setupDevice(int deviceID, const WCHAR *fileName,
```

```
                  bool initRunning = true);
  bool setupDevice(int deviceID, int w, int h, const WCHAR *fileName,
bool initRunning = true);
  int setupFileAsDevice(const WCHAR *fileName, bool disableSync =
false, bool initRunning = true, bool oneShot = false);
  bool codecSetupDialog(int deviceID);
  bool run(int deviceID);
  bool pause(int deviceID);
  bool stop(int deviceID);
  bool setPlayRate(int deviceID, double rate);
  double getDurationInSeconds(int deviceID);
  LONGLONG getDurationInFrames(int deviceID);
  // Hace el seek y retorna el tiempo en reference time
  LONGLONG seekInFrames(int deviceID, LONGLONG frame);
  LONGLONG getPositionInFrames(int deviceID);

};

#endif
```

9.1.2 ExtendedVideoInput.cpp

```
#include "extendedVideoInput.h"
#include "Filters.h"
#include <DShow.h>
#pragma include_alias( "dxtrans.h", "qedit.h" )
#define __IDxtCompositor_INTERFACE_DEFINED__
#define __IDxtAlphaSetter_INTERFACE_DEFINED__
#define __IDxtJpeg_INTERFACE_DEFINED__
#define __IDxtKey_INTERFACE_DEFINED__
#include <qedit.h>

static const WCHAR CODEC_FRIENDLY_NAME[] = L"Lagarith lossless codec";
//static const WCHAR CODEC_FRIENDLY_NAME[] = L"Huffyuv v2.1.1";

extendedVideoInput::extendedVideoInput() {
  numFilesAsDevicesConfigured = 0;
  mediaSeeking = NULL;
}

extendedVideoInput::~extendedVideoInput() {
  SAFE_RELEASE(mediaSeeking);
```

```
}


// ----------------------------------------------------------------------
// Setup a device with the default settings
//
// ----------------------------------------------------------------------

bool extendedVideoInput::setupDevice(int deviceID, const WCHAR
*fileName, bool initRunning) {
  if(deviceID >= VI_MAX_CAMERAS || VDList[deviceID]->readyToCapture)
return false;

  if(setup(deviceID, fileName, initRunning))return true;
  return false;
}

// ----------------------------------------------------------------------
// Setup a device with the default connection but specify size
//
// ----------------------------------------------------------------------

bool extendedVideoInput::setupDevice(int deviceID, int w, int h, const
WCHAR *fileName, bool initRunning) {
  if(deviceID >= VI_MAX_CAMERAS || VDList[deviceID]->readyToCapture)
return false;

  setAttemptCaptureSize(deviceID,w,h);
  if(setup(deviceID, fileName, initRunning))return true;
  return false;
}

int extendedVideoInput::setupFileAsDevice(const WCHAR *fileName, bool
disableSync, bool initRunning, bool oneShot) {
  int newDevice = getDeviceCount() + numFilesAsDevicesConfigured;

  HRESULT hr = startFileAsDevice(newDevice, VDList[newDevice],
fileName, disableSync, initRunning, oneShot);
  if(hr == S_OK) {
    devicesFound = newDevice + 1;
    numFilesAsDevicesConfigured++;
    return newDevice;
```

```
  } else
    return -1;
}

bool extendedVideoInput::codecSetupDialog(int deviceID) {
  IBaseFilter *codecFilter = NULL;
  IAMVfwCompressDialogs *pCompDialog = NULL;
  HRESULT hr = E_FAIL;

  codecFilter = getFilterByName(VDList[deviceID]->pGraph,
CODEC_FRIENDLY_NAME);
  if (codecFilter != NULL) {
    // Codec configuration
    if ((codecFilter->QueryInterface(IID_IAMVfwCompressDialogs, (void
**)&pCompDialog) == S_OK) && (pCompDialog != NULL)) {
      if (SUCCEEDED(pCompDialog-
>ShowDialog(VfwCompressDialog_QueryConfig, NULL)))
          hr = pCompDialog->ShowDialog(VfwCompressDialog_Config,
NULL);
      SAFE_RELEASE(pCompDialog);
    }
    SAFE_RELEASE(codecFilter);
  }

  if (SUCCEEDED(hr))
    return true;
  else
    return false;
}

bool extendedVideoInput::setup(int deviceNumber, const WCHAR
*fileName, bool initRunning) {
  devicesFound = getDeviceCount();

  if(deviceNumber>devicesFound-1) {
    if(verbose)printf("SETUP: device[%i] not found - you have %i
devices available\n", deviceNumber, devicesFound);
    if(devicesFound>=0) if(verbose)printf("SETUP: this means that the
last device you can use is device[%i] \n",  devicesFound-1);
    return false;
  }
  if(VDList[deviceNumber]->readyToCapture) {
```

```
    if(verbose)printf("SETUP: can't setup, device %i is currently
being used\n",VDList[deviceNumber]->myID);
    return false;
  }
  HRESULT hr = startDevice(deviceNumber, VDList[deviceNumber],
fileName, initRunning);
  if(hr == S_OK)return true;
  else return false;
}


int extendedVideoInput::startDevice(int deviceID, videoDevice * VD,
const WCHAR *fileName, bool initRunning) {
  HRESULT hr                    = NULL;
  VD->myID                  = deviceID;
  VD->setupStarted      = true;
  CAPTURE_MODE          = PIN_CATEGORY_CAPTURE; //Don't worry - it ends
up being preview (which is faster)
  callbackSetCount      = 1;  //make sure callback method is not
changed after setup called
  IPin *pSourceOut[2];


  if(verbose)printf("SETUP: Setting up device %i\n",deviceID);


  // CREATE THE GRAPH BUILDER //
  // Create the filter graph manager and query for interfaces.
  hr = CoCreateInstance(CLSID_CaptureGraphBuilder2, NULL,
CLSCTX_INPROC_SERVER, IID_ICaptureGraphBuilder2, (void **)&VD-
>pCaptureGraph);
  if (FAILED(hr)) // FAILED is a macro that tests the return value
  {
    if(verbose)printf("ERROR - Could not create the Filter Graph
Manager\n");
    return hr;
  }


  //FITLER GRAPH MANAGER//
  // Create the Filter Graph Manager.
  hr = CoCreateInstance(CLSID_FilterGraph, 0,
CLSCTX_INPROC_SERVER,IID_IGraphBuilder, (void**)&VD->pGraph);
  if (FAILED(hr))
  {
    if(verbose)printf("ERROR - Could not add the graph builder!\n");
    stopDevice(deviceID);
```

```
    return hr;
  }

  //SET THE FILTERGRAPH//
  hr = VD->pCaptureGraph->SetFiltergraph(VD->pGraph);
  if (FAILED(hr)) {
    if(verbose)printf("ERROR - Could not set filtergraph\n");
    stopDevice(deviceID);
    return hr;
  }

  //MEDIA CONTROL (START/STOPS STREAM)//
  // Using QueryInterface on the graph builder,
  // Get the Media Control object.
  hr = VD->pGraph->QueryInterface(IID_IMediaControl, (void **)&VD-
>pControl);
  if (FAILED(hr)) {
    if(verbose)printf("ERROR - Could not create the Media Control
object\n");
    stopDevice(deviceID);
    return hr;
  }

  //FIND VIDEO DEVICE AND ADD TO GRAPH//
  //gets the device specified by the second argument.
  hr = getDevice(&VD->pVideoInputFilter, deviceID, VD->wDeviceName,
VD->nDeviceName);

  if (SUCCEEDED(hr)){
    if(verbose)printf("SETUP: %s\n", VD->nDeviceName);
    hr = VD->pGraph->AddFilter(VD->pVideoInputFilter, VD-
>wDeviceName);
  } else {
    if(verbose)printf("ERROR - Could not find specified video
device\n");
    stopDevice(deviceID);
    return hr;
  }
  //LOOK FOR PREVIEW PIN IF THERE IS NONE THEN WE USE CAPTURE PIN AND
THEN SMART TEE TO PREVIEW
  IAMStreamConfig *streamConfTest = NULL;
```

```
  hr = VD->pCaptureGraph->FindInterface(&PIN_CATEGORY_PREVIEW,
&MEDIATYPE_Video, VD->pVideoInputFilter, IID_IAMStreamConfig, (void
**)&streamConfTest);

  if(FAILED(hr)){

    if(verbose)printf("SETUP: Couldn't find preview pin using
SmartTee\n");

  } else {

    CAPTURE_MODE = PIN_CATEGORY_PREVIEW;

    streamConfTest->Release();

    streamConfTest = NULL;

  }


  //CROSSBAR (SELECT PHYSICAL INPUT TYPE)//

  //my own function that checks to see if the device can support a
crossbar and if so it routes it.

  //webcams tend not to have a crossbar so this function will also
detect a webcams and not apply the crossbar

  if(VD->useCrossbar) {

    if(verbose)printf("SETUP: Checking crossbar\n");

    routeCrossbar(&VD->pCaptureGraph, &VD->pVideoInputFilter, VD-
>connection, CAPTURE_MODE);

  }


  //we do this because webcams don't have a preview mode

  hr = VD->pCaptureGraph->FindInterface(&CAPTURE_MODE,
&MEDIATYPE_Video, VD->pVideoInputFilter, IID_IAMStreamConfig, (void
**)&VD->streamConf);

  if(FAILED(hr)){

    if(verbose)printf("ERROR: Couldn't config the stream!\n");

    stopDevice(deviceID);

    return hr;

  }


  //NOW LETS DEAL WITH GETTING THE RIGHT SIZE

  hr = VD->streamConf->GetFormat(&VD->pAmMediaType);

  if(FAILED(hr)){

    if(verbose)printf("ERROR: Couldn't getFormat for
pAmMediaType!\n");

    stopDevice(deviceID);

    return hr;

  }


  VIDEOINFOHEADER *pVih =  reinterpret_cast<VIDEOINFOHEADER*>(VD-
>pAmMediaType->pbFormat);
```

```
  int currentWidth      =  HEADER(pVih)->biWidth;
  int currentHeight     =  HEADER(pVih)->biHeight;

  bool customSize = VD->tryDiffSize;
  bool foundSize  = false;

  if(customSize){
    if(verbose)   printf("SETUP: Default Format is set to %i by %i \n",
currentWidth, currentHeight);

    char guidStr[8];
    for(int i = 0; i < VI_NUM_TYPES; i++){

      getMediaSubtypeAsString(mediaSubtypes[i], guidStr);

      if(verbose)printf("SETUP: trying format %s @ %i by %i\n",
guidStr, VD->tryWidth, VD->tryHeight);
      if( setSizeAndSubtype(VD, VD->tryWidth, VD->tryHeight,
mediaSubtypes[i]) ){
        VD->setSize(VD->tryWidth, VD->tryHeight);
        foundSize = true;
        break;
      }
    }

    //if we didn't find the requested size - lets try and find the
closest matching size
    if( foundSize == false ){
      if( verbose )printf("SETUP: couldn't find requested size -
searching for closest matching size\n");

      int closestWidth       = -1;
      int closestHeight      = -1;
      GUID newMediaSubtype;

      findClosestSizeAndSubtype(VD, VD->tryWidth, VD->tryHeight,
closestWidth, closestHeight, newMediaSubtype);

      if( closestWidth != -1 && closestHeight != -1){
        getMediaSubtypeAsString(newMediaSubtype, guidStr);

        if(verbose)printf("SETUP: closest supported size is %s @ %i
%i\n", guidStr, closestWidth, closestHeight);
```

```
        if( setSizeAndSubtype(VD, closestWidth, closestHeight,
newMediaSubtype) ){
          VD->setSize(closestWidth, closestHeight);
          foundSize = true;
        }
      }
    }
  }

  //if we didn't specify a custom size or if we did but couldn't find
it lets setup with the default settings
  if(customSize == false || foundSize == false){
    if( VD->requestedFrameTime != -1 ){
      pVih->AvgTimePerFrame   = VD->requestedFrameTime;
      hr = VD->streamConf->SetFormat(VD->pAmMediaType);
    }
    VD->setSize(currentWidth, currentHeight);
  }

  //SAMPLE GRABBER (ALLOWS US TO GRAB THE BUFFER)//
  // Create the Sample Grabber.
  pSourceOut[0] = GetPin(VD->pVideoInputFilter, PINDIR_OUTPUT);
  hr = addFilter(CLSID_SampleGrabber, L"SampleGrabber", VD->pGraph,
pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("Could not Create Sample Grabber -
addFilter()\n");
    stopDevice(deviceID);
    return hr;
  }

  VD->pGrabberF = getFilterByName(VD->pGraph, L"SampleGrabber");

  hr = VD->pGrabberF->QueryInterface(IID_ISampleGrabber, (void**)&VD-
>pGrabber);
  if (FAILED(hr)){
    if(verbose)printf("ERROR: Could not query SampleGrabber\n");
    stopDevice(deviceID);
    return hr;
  }
```

```
  //Set Params - One Shot should be false unless you want to capture
just one buffer
  hr = VD->pGrabber->SetOneShot(FALSE);
  if(bCallback){
    hr = VD->pGrabber->SetBufferSamples(FALSE);
  }else{
    hr = VD->pGrabber->SetBufferSamples(TRUE);
  }

  if(bCallback){
    //Tell the grabber to use our callback function - 0 is for
SampleCB and 1 for BufferCB
    //We use SampleCB
    hr = VD->pGrabber->SetCallback((ISampleGrabberCB *)(VD-
>sgCallback), 0);
    if (FAILED(hr)){
      if(verbose)printf("ERROR: problem setting callback\n");
      stopDevice(deviceID);
      return hr;
    }else{
      if(verbose)printf("SETUP: Capture callback set\n");
    }
  }

  // MEDIA CONVERSION
  // Get video properties from the stream's mediatype and apply to the
grabber (otherwise we don't get an RGB image)
  // zero the media type - lets try this :) - maybe this works?
  AM_MEDIA_TYPE mt;
  ZeroMemory(&mt,sizeof(AM_MEDIA_TYPE));

  mt.majortype    = MEDIATYPE_Video;
  mt.subtype      = MEDIASUBTYPE_RGB24;
  mt.formattype   = FORMAT_VideoInfo;

  //VD->pAmMediaType->subtype = VD->videoType;
  hr = VD->pGrabber->SetMediaType(&mt);

  //lets try freeing our stream conf here too
  //this will fail if the device is already running
  if(VD->streamConf){
    VD->streamConf->Release();
```

```
    VD->streamConf = NULL;
  }else{
    if(verbose)printf("ERROR: connecting device - prehaps it is
already being used?\n");
    stopDevice(deviceID);
    return S_FALSE;
  }

  // Null Render
  //hr = addFilter(CLSID_NullRenderer, NULL, VD->pGraph, &pSourceOut);
  //if (FAILED(hr)){
  //  if(verbose)printf("ERROR: Could not create Null Render\n");
  //  stopDevice(deviceID);
  //  return hr;
  //}

  // Compression filter
  //addFilterByEnum(CLSID_VideoCompressorCategory, "Huffyuv v2.1.1",
VD->pGraph, pSourceOut);
  IBaseFilter *pFilter = NULL;

  pFilter = addFilterByEnum(CLSID_VideoCompressorCategory,
CODEC_FRIENDLY_NAME, VD->pGraph, pSourceOut, 1, true);

 //   // Codec configuration

 // IAMVfwCompressDialogs *pCompDialog = NULL;

  //if ((pFilter->QueryInterface(IID_IAMVfwCompressDialogs, (void
**)&pCompDialog) == S_OK) && (pCompDialog != NULL))
 //   if (SUCCEEDED(pCompDialog-
>ShowDialog(VfwCompressDialog_QueryConfig, NULL))) {
  //    hr = pCompDialog->ShowDialog(VfwCompressDialog_Config, NULL);
  //        SAFE_RELEASE(pCompDialog);
 //   }
  SAFE_RELEASE(pFilter);

  //pSourceOut[0]->QueryInterface(IID_IAMVideoCompression,
(void**)&pCompress);
  //pCompress->GetInfo(

  // AVI Mux
```

```
  hr = addFilter(CLSID_AviDest, L"AVImux", VD->pGraph, pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("ERROR: Could not create filter - AVImux\n");
    stopDevice(deviceID);
    return hr;
  }

  // File Writer
  hr = addFileWriter(fileName, VD->pGraph, pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("ERROR: Could not create filter - File
writer\n");
    stopDevice(deviceID);
    return hr;
  }

  ////EXP - lets try setting the sync source to null - and make it run
as fast as possible
  //{
  //  IMediaFilter *pMediaFilter = 0;
  //  hr = VD->pGraph->QueryInterface(IID_IMediaFilter,
(void**)&pMediaFilter);
  //  if (FAILED(hr)){
  //    if(verbose)printf("ERROR: Could not get IID_IMediaFilter
interface\n");
  //  }else{
  //    pMediaFilter->SetSyncSource(NULL);
  //    pMediaFilter->Release();
  //  }
  //}

  if (initRunning) {
    //LETS RUN THE STREAM!
    hr = VD->pControl->Run();

    if (FAILED(hr)){
      if(verbose)printf("ERROR: Could not start graph\n");
      stopDevice(deviceID);
      return hr;
    }

    // MAKE SURE THE DEVICE IS SENDING VIDEO BEFORE WE FINISH
```

```
    if(!bCallback){
      long bufferSize = VD->videoSize;

      while( hr != S_OK){
        hr = VD->pGrabber->GetCurrentBuffer(&bufferSize, (long *)VD-
>pBuffer);
        Sleep(10);
      }
    }
  } else {
    //LETS RUN THE STREAM!
    hr = VD->pControl->StopWhenReady();
  }

  if(verbose)printf("SETUP: Device is setup and ready to
capture.\n\n");
  VD->readyToCapture = true;

  //Release filters - seen someone else do this
  //looks like it solved the freezes

  //if we release this then we don't have access to the settings
  //we release our video input filter but then reconnect with it
  //each time we need to use it

  VD->pVideoInputFilter->Release();
  VD->pVideoInputFilter = NULL;

  //VD->pGrabberF->Release();
  //VD->pGrabberF = NULL;

  //VD->pDestFilter->Release();
  //VD->pDestFilter = NULL;

  return S_OK;
}

int  extendedVideoInput::startFileAsDevice(int deviceID, videoDevice *
VD, const WCHAR *fileName,
                                           bool disableSync, bool
initRunning, bool oneShot) {
  HRESULT hr                  = NULL;
```

```
  VD->myID              = deviceID;
  VD->setupStarted      = true;
  callbackSetCount      = 1;  //make sure callback method is not
changed after setup called
  IPin *pSourceOut[2];

  if(verbose)printf("SETUP: Setting up device %i\n",deviceID);

  //FITLER GRAPH MANAGER//
  // Create the Filter Graph Manager.
  hr = CoCreateInstance(CLSID_FilterGraph, 0,
CLSCTX_INPROC_SERVER,IID_IGraphBuilder, (void**)&VD->pGraph);
  if (FAILED(hr)) {
    if(verbose)printf("ERROR - Could not add the graph builder!\n");
    stopDevice(deviceID);
    return hr;
  }

  //MEDIA CONTROL (START/STOPS STREAM)//
  // Using QueryInterface on the graph builder,
  // Get the Media Control object.
  hr = VD->pGraph->QueryInterface(IID_IMediaControl, (void **)&VD-
>pControl);
  if (FAILED(hr)) {
    if(verbose)printf("ERROR - Could not create the Media Control
object\n");
    stopDevice(deviceID);
    return hr;
  }

  // Source Filter
  hr = addSource(fileName, VD->pGraph, &(pSourceOut[0]));
  if (FAILED(hr)) {
    if(verbose)printf("ERROR - Could not create the source filter -
addFilter()\n");
    stopDevice(deviceID);
    return hr;
  }

  // Avi Splitter
  hr = addFilter(CLSID_AviSplitter, L"AviSplitter", VD->pGraph,
pSourceOut);
```

```
  if (FAILED(hr)){
    if(verbose)printf("Could not Create AVI Splitter -
addFilter()\n");
    stopDevice(deviceID);
    return hr;
  }

  // Avi Decompressor
  hr = addFilter(CLSID_AVIDec, L"AviDecompressor", VD->pGraph,
pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("Could not Create AVI decompressor -
addFilter()\n");
    stopDevice(deviceID);
    return hr;
  }

  IPin *decompressorOutPin = pSourceOut[0];

  //// Añadimos el filtro samplegrabber pero no lo conectamos porque
primero hay que cambiar
  //// el formato de entrada de color, para que el filtro de
conversion de color sepa a que
  //// formato tiene que convertir
  pSourceOut[0] = NULL;

  //SAMPLE GRABBER (ALLOWS US TO GRAB THE BUFFER)//
  // Create the Sample Grabber.
  hr = addFilter(CLSID_SampleGrabber, L"SampleGrabber", VD->pGraph,
pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("Could not Create Sample Grabber -
addFilter()\n");
    stopDevice(deviceID);
    return hr;
  }

  VD->pGrabberF = getFilterByName(VD->pGraph, L"SampleGrabber");

  hr = VD->pGrabberF->QueryInterface(IID_ISampleGrabber, (void**)&VD-
>pGrabber);
  if (FAILED(hr)){
    if(verbose)printf("ERROR: Could not query SampleGrabber\n");
```

```
    stopDevice(deviceID);
    return hr;
  }

  //Set Params - One Shot should be false unless you want to capture
just one buffer
  hr = VD->pGrabber->SetOneShot(oneShot);

  if(bCallback){
    hr = VD->pGrabber->SetBufferSamples(FALSE);
  }else{
    hr = VD->pGrabber->SetBufferSamples(TRUE);
  }

  if(bCallback){
    //Tell the grabber to use our callback function - 0 is for
SampleCB and 1 for BufferCB
    //We use SampleCB
    hr = VD->pGrabber->SetCallback((ISampleGrabberCB *)(VD-
>sgCallback), 0);
    if (FAILED(hr)){
      if(verbose)printf("ERROR: problem setting callback\n");
      stopDevice(deviceID);
      return hr;
    }else{
      if(verbose)printf("SETUP: Capture callback set\n");
    }
  }

  //MEDIA CONVERSION
  //Get video properties from the stream's mediatype and apply to the
grabber (otherwise we don't get an RGB image)
  //zero the media type - lets try this :) - maybe this works?
  AM_MEDIA_TYPE mt;
//  hr = VD->pGrabber->GetConnectedMediaType(&mt);
  ZeroMemory(&mt,sizeof(AM_MEDIA_TYPE));

  mt.majortype    = MEDIATYPE_Video;
  mt.subtype = MEDIASUBTYPE_RGB24;
  mt.formattype   = FORMAT_VideoInfo;

  //VD->pAmMediaType->subtype = VD->videoType;
```

```
  hr = VD->pGrabber->SetMediaType(&mt);


  // Una vez configurado el formato de entrada de SampleGrabber
conectamos el convertidor de
  // espacio de color al samplegrabber


  IBaseFilter *sampleGrabberFilter = getFilterByName(VD->pGraph,
L"SampleGrabber");
  IPin *sampleGrabberInputPin = GetPin(sampleGrabberFilter,
PINDIR_INPUT);
  hr = VD->pGraph->Connect(decompressorOutPin, sampleGrabberInputPin);
  if(FAILED(hr)) {
    printf("Error connecting samplegrabber with colour space
converter\n");
    return E_FAIL;
  }


  SAFE_RELEASE(decompressorOutPin);
  SAFE_RELEASE(sampleGrabberFilter);
  SAFE_RELEASE(sampleGrabberInputPin);


  // Una vez conectado el sample grabber obtenemos el tamaño de frame
y el framerate
  AM_MEDIA_TYPE l_MediaType;
  VD->pGrabber->GetConnectedMediaType(&l_MediaType);
  // Get a pointer to the video header.
  VIDEOINFOHEADER *pVideoHeader =
(VIDEOINFOHEADER*)l_MediaType.pbFormat;
  avgTimePerFrame = pVideoHeader->AvgTimePerFrame;


  VD->setSize(pVideoHeader->bmiHeader.biWidth, pVideoHeader-
>bmiHeader.biHeight);


  // Null Render
  hr = addFilter(CLSID_NullRenderer, NULL, VD->pGraph, pSourceOut);
  if (FAILED(hr)){
    if(verbose)printf("ERROR: Could not create Null Render\n");
    stopDevice(deviceID);
    return hr;
  }


  //// Video Render
```

```
  //hr = addFilter(CLSID_VideoRenderer, L"VideoRender", VD->pGraph,
pSourceOut);
  //if (FAILED(hr)){
  //  if(verbose)printf("Could not create Video Renderer -
addFilter()\n");
  //  stopDevice(deviceID);
  //  return hr;
  //}


  //EXP - lets try setting the sync source to null - and make it run
as fast as possible
  if (disableSync) {
    IMediaFilter *pMediaFilter = 0;
    hr = VD->pGraph->QueryInterface(IID_IMediaFilter,
(void**)&pMediaFilter);
    if (FAILED(hr)){
      if(verbose)printf("ERROR: Could not get IID_IMediaFilter
interface\n");
    }else{
      pMediaFilter->SetSyncSource(NULL);
      pMediaFilter->Release();
    }
  }


  mediaSeeking = NULL;
  hr = VD->pGraph->QueryInterface(IID_IMediaSeeking, (void
**)&mediaSeeking);
  if (FAILED(hr)) {
    if(verbose)printf("ERROR: Could not get IID_IMediaSeeking
interface\n");
    return hr;
  }


  //LETS RUN THE STREAM!
  if (initRunning) {
    hr = VD->pControl->Run();


    if (FAILED(hr)){
      if(verbose)printf("ERROR: Could not start graph\n");
      stopDevice(deviceID);
      return hr;
    }
```

```
    // MAKE SURE THE DEVICE IS SENDING VIDEO BEFORE WE FINISH
    if(!bCallback){
      long bufferSize = VD->videoSize;

      while( hr != S_OK){
        hr = VD->pGrabber->GetCurrentBuffer(&bufferSize, (long *)VD-
>pBuffer);
        Sleep(10);
      }
    }
  } else {
    //LETS RUN THE STREAM!
    hr = VD->pControl->StopWhenReady();
  }

  if(verbose)printf("SETUP: Device is setup and ready to
capture.\n\n");
  VD->readyToCapture = true;

  //Release filters - seen someone else do this
  //looks like it solved the freezes

  //if we release this then we don't have access to the settings
  //we release our video input filter but then reconnect with it
  //each time we need to use it

  //VD->pVideoInputFilter->Release();
  //VD->pVideoInputFilter = NULL;

  VD->pGrabberF->Release();
  VD->pGrabberF = NULL;

  //VD->pDestFilter->Release();
  //VD->pDestFilter = NULL;

  return S_OK;
}


bool extendedVideoInput::run(int deviceID) {
  HRESULT hr = VDList[deviceID]->pControl->Run();
```

```
  if (SUCCEEDED(hr))
    return true;
  else
    return false;
}

bool extendedVideoInput::pause(int deviceID) {
  HRESULT hr = VDList[deviceID]->pControl->Pause();
  if (SUCCEEDED(hr))
    return true;
  else
    return false;
}

bool extendedVideoInput::stop(int deviceID) {
  HRESULT hr;

  hr = VDList[deviceID]->pControl->Pause();
  if (FAILED(hr)) return false;

  LONGLONG c = 0;
  hr = mediaSeeking->SetPositions(&c, AM_SEEKING_AbsolutePositioning,
NULL, AM_SEEKING_NoPositioning);

  if (FAILED(hr)) {
    return false;
  } else
    return true;
}

bool extendedVideoInput::setPlayRate(int deviceID, double rate) {
  HRESULT hr;

  IMediaSeeking *mediaSeeking = NULL;
  hr = VDList[deviceID]->pGraph->QueryInterface(IID_IMediaSeeking,
(void **)&mediaSeeking);
  if (FAILED(hr)) return false;

  hr = mediaSeeking->SetRate(rate);

  SAFE_RELEASE(mediaSeeking);
```

```
  if (FAILED(hr)) {
    return false;
  } else
    return true;
}

double extendedVideoInput::getDurationInSeconds(int deviceID) {
  HRESULT hr = E_FAIL;;
  LONGLONG duration;

  if (mediaSeeking) {
    mediaSeeking->SetTimeFormat(&TIME_FORMAT_MEDIA_TIME);
    hr = mediaSeeking->GetDuration(&duration);
  }

  if (SUCCEEDED(hr))
    return (duration * 0.000001);
  else
    return -1.0;
}

LONGLONG extendedVideoInput::getDurationInFrames(int deviceID) {
  HRESULT hr = E_FAIL;
  LONGLONG duration;

  if (mediaSeeking) {
    mediaSeeking->SetTimeFormat(&TIME_FORMAT_FRAME);
    hr = mediaSeeking->GetDuration(&duration);
  }

  if (SUCCEEDED(hr))
    return duration;
  else
    return -1;
}

LONGLONG extendedVideoInput::seekInFrames(int deviceID, LONGLONG
frame) {
  HRESULT hr = E_FAIL;
  LONGLONG pos = frame;
```

```
  if (mediaSeeking) {
    mediaSeeking->SetTimeFormat(&TIME_FORMAT_FRAME);
    hr = mediaSeeking->SetPositions(&pos,
AM_SEEKING_AbsolutePositioning | AM_SEEKING_ReturnTime, NULL,
AM_SEEKING_NoPositioning);
  }

  if (SUCCEEDED(hr))
    return (pos);
  else
    return -1;
}

LONGLONG extendedVideoInput::getPositionInFrames(int deviceID) {
  HRESULT hr = E_FAIL;
  LONGLONG pos;

  if (mediaSeeking) {
    mediaSeeking->SetTimeFormat(&TIME_FORMAT_FRAME);
    hr = mediaSeeking->GetCurrentPosition(&pos);
  }

  if (SUCCEEDED(hr))
    return (pos);
  else
    return -1;
    }
```

9.1.3 Filters.h

```
#include <streams.h>
#include <Dshow.h>
#include <vector>
#include <atlstr.h>

#define SAFE_RELEASE(p) { if(p){(p)->Release();(p)=NULL;}}

IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir);
IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir, int n);
HRESULT addRenderer(const WCHAR* renderername, IGraphBuilder *pGraph,
IPin **outputPin);
```

```
HRESULT addSource(const WCHAR* sourcename, IGraphBuilder *pGraph, IPin
**outputPin);

HRESULT addFileWriter(const WCHAR* filename, IGraphBuilder *pGraph,
IPin **outputPin);

HRESULT addFilter(REFCLSID filterCLSID, const WCHAR* filtername,
IGraphBuilder *pgraph,
                IPin **outputPin, int numberOfOutput=1);

bool removeAllFilters(IGraphBuilder *pGraph);

std::vector<CString> enumFilters(IGraphBuilder *pGraph);

//IBaseFilter *getFilterByFriendlyName(REFCLSID CLSIDcategory, CString
friendlyName);

void enumFilters(REFCLSID CLSIDcategory, std::vector<CString>& names,
std::vector<CLSID>& clsidFilters);

IBaseFilter* addFilterByEnum(REFCLSID CLSIDcategory, CString
filtername, IGraphBuilder *pGraph,
                IPin **outputPin, int numberOfOutput=1, bool
returnIt=false);

IBaseFilter* getFilterByName(IGraphBuilder *pGraph, CString name);

bool changeCompressorOptions(IPin **outputPin, int numberOfOutput,
int& iKeyFrames, int& iPFrames, double& dQuality);

bool changeVideoCaptureOptions(IBaseFilter* pVideoSource, long lParam,
long lValue, long lFlags=0);

bool changeVideoCaptureOptionsByName(const WCHAR* wcVideoSource,
IFilterGraph* pGraph, long lParam, long lValue, long lFlags=0);
```

9.1.4 Filters.cpp

```
#include "filters.h"


HRESULT addFilter(REFCLSID filterCLSID, const WCHAR* filtername,
IGraphBuilder *pGraph,
                IPin **outputPin, int numberOfOutput) {
  HRESULT hr1, hr2;


  // Create the filter.


  IBaseFilter* baseFilter = NULL;
  char tmp[100];


  if(FAILED(CoCreateInstance(
    filterCLSID, NULL, CLSCTX_INPROC_SERVER,
    IID_IBaseFilter, (void**)&baseFilter)) || !baseFilter) {
    sprintf(tmp,"Unable to create %ls filter", filtername);
    ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
    return E_FAIL;
```

```
  }
  hr1 = pGraph->AddFilter(baseFilter, filtername);
  if (*outputPin != NULL) {
    // Obtain the input pin.
    IPin* inputPin= GetPin(baseFilter, PINDIR_INPUT);
    if (!inputPin) {
      sprintf(tmp,"Unable to obtain %ls input pin", filtername);
      ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
      return E_FAIL;
      }
    // Connect the filter to the ouput pin.
    hr2 = pGraph->Connect(*outputPin, inputPin);
    SAFE_RELEASE(*outputPin);
    SAFE_RELEASE(inputPin);
    if(FAILED(hr1) || FAILED(hr2)) {
      sprintf(tmp,"Unable to connect %ls filter", filtername);
      ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
      return E_FAIL;
    }
  }
  // Obtain the output pin(s).
  for (int i=0; i<numberOfOutput; i++) {
    outputPin[i]= 0;
    outputPin[i]= GetPin(baseFilter, PINDIR_OUTPUT, i+1);
    if (!outputPin[i]) {
      outputPin[i] = NULL;
      // sprintf(tmp,"Unable to obtain %s output pin (%d)",
filtername, i);
      // ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
      // return 0;
    }
  }
  SAFE_RELEASE(baseFilter);
  return S_OK;
}


HRESULT addRenderer(const WCHAR* renderername, IGraphBuilder *pGraph,
IPin **outputPin) {


      char tmp[100];
```

```
      IBaseFilter* pVideo = NULL;

      if(FAILED(CoCreateInstance(CLSID_VideoRenderer, NULL,
CLSCTX_INPROC_SERVER,
                IID_IBaseFilter, (void**)&pVideo)) || !pVideo)
      {
        sprintf(tmp,"Unable to create renderer %ls", renderername);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }


      IPin* pVideoIn= GetPin(pVideo, PINDIR_INPUT);
      if (!pVideoIn) {

        sprintf(tmp,"Unable to obtain %ls input pin", renderername);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }

      if(FAILED(pGraph->AddFilter( pVideo, renderername)) ||
         FAILED(pGraph->Connect(*outputPin, pVideoIn)) )
      {
        sprintf(tmp,"Unable to connect %ls renderer", renderername);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }

      SAFE_RELEASE(pVideo);
      SAFE_RELEASE(pVideoIn);
      SAFE_RELEASE(*outputPin);

      return S_OK;
}

HRESULT addSource(const WCHAR* sourcename, IGraphBuilder *pGraph, IPin
**outputPin) {

      char tmp[100];
```

```
      IBaseFilter*      pSource;

      if(FAILED(pGraph->AddSourceFilter(sourcename,0,&pSource)))
      {
        sprintf(tmp,"Unable to connect to %ls source", sourcename);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }

      *outputPin= 0;
      *outputPin= GetPin(pSource, PINDIR_OUTPUT);

      if (! *outputPin) {

        sprintf(tmp,"Unable to obtain source (%ls) input pin",
sourcename);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }

      SAFE_RELEASE(pSource);

      return S_OK;
}

HRESULT addFileWriter(const WCHAR* filename, IGraphBuilder *pGraph,
IPin **outputPin) {

      char tmp[100];
      IBaseFilter* pWriter = NULL;
      if(FAILED(CoCreateInstance(CLSID_FileWriter, NULL,
CLSCTX_INPROC_SERVER,
                IID_IBaseFilter, (void**)&pWriter)) || !pWriter)
      {
        sprintf(tmp,"Unable to create file writer (%ls)", filename);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }
```

```
      IFileSinkFilter2 *psink;
      pWriter->QueryInterface(IID_IFileSinkFilter2, (void **)&psink);

      psink->SetFileName(filename,NULL);
      psink->SetMode(AM_FILE_OVERWRITE);

      IPin* pWriterIn= GetPin(pWriter, PINDIR_INPUT);
      if (!pWriterIn) {

        sprintf(tmp,"Unable to obtain writer input pin (%ls)",
filename);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION
);
        return E_FAIL;
      }

      if(FAILED(pGraph->AddFilter( pWriter, filename)) ||
         FAILED(pGraph->Connect(*outputPin, pWriterIn)) )
      {
        ::MessageBox( NULL, "Unable to connect writer filter",
"Error", MB_OK | MB_ICONINFORMATION );
        return E_FAIL;
      }

      SAFE_RELEASE(pWriter);
      SAFE_RELEASE(pWriterIn);
      SAFE_RELEASE(*outputPin);

      return S_OK;
}

/*
 * Returns the next available pin.
 */
IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir)
{
    BOOL       bFound = FALSE;
    IEnumPins  *pEnum;
    IPin       *pPin;
    IPin       *pPin2;
```

```
    pFilter->EnumPins(&pEnum);
    while(pEnum->Next(1, &pPin, 0) == S_OK)
    {
        PIN_DIRECTION PinDirThis;
        pPin->QueryDirection(&PinDirThis);
        if (PinDir == PinDirThis) {

            pPin->ConnectedTo(&pPin2); // is this pin
                                       // connected to another pin ?
            if (pPin2 == 0) {
              bFound= TRUE;
              break;
            }
        }
        pPin->Release();
    }

    pEnum->Release();
    return (bFound ? pPin : NULL);
}

/*
 * Returns the n ieme pin (the first one being number 1).
 */
IPin *GetPin(IBaseFilter *pFilter, PIN_DIRECTION PinDir, int n)
{
    BOOL       bFound = FALSE;
    IEnumPins  *pEnum;
    IPin       *pPin;
    int i=1;

    pFilter->EnumPins(&pEnum);
    while(pEnum->Next(1, &pPin, 0) == S_OK)
    {
        PIN_DIRECTION PinDirThis;
        pPin->QueryDirection(&PinDirThis);
        if (PinDir == PinDirThis) {

            if (i == n) {
```

```
              bFound= TRUE;
              break;

            } else {

              i++;
            }
        }
        pPin->Release();
    }

    pEnum->Release();
    return (bFound ? pPin : NULL);
}

std::vector<CString> enumFilters(IGraphBuilder *pGraph) {

      IEnumFilters *pEnum = NULL;
      IBaseFilter *pFilter;
      ULONG cFetched;
      std::vector<CString> names;

      pGraph->EnumFilters(&pEnum);
      while(pEnum->Next(1, &pFilter, &cFetched) == S_OK)
      {
        FILTER_INFO FilterInfo;
        char szName[256];
        CString fname;

        pFilter->QueryFilterInfo(&FilterInfo);
        WideCharToMultiByte(CP_ACP, 0, FilterInfo.achName, -1, szName,
256, 0, 0);
        fname= szName;
        names.push_back(fname);

        SAFE_RELEASE(FilterInfo.pGraph);
        SAFE_RELEASE(pFilter);
      }

      SAFE_RELEASE(pEnum);
```

```
      return names;
}

IBaseFilter* getFilterByName(IGraphBuilder *pGraph, CString name) {
  IEnumFilters *pEnum = NULL;
  IBaseFilter *pFilter;
  ULONG cFetched;
  bool found= false;

  pGraph->EnumFilters(&pEnum);
  while(!found && pEnum->Next(1, &pFilter, &cFetched) == S_OK) {
    FILTER_INFO FilterInfo;
    char szName[256];
    CString fname;

    pFilter->QueryFilterInfo(&FilterInfo);
    WideCharToMultiByte(CP_ACP, 0, FilterInfo.achName, -1, szName,
256, 0, 0);
    fname= szName;

    SAFE_RELEASE(FilterInfo.pGraph);

    if (fname == name)
      found= true;
    else
      SAFE_RELEASE(pFilter);
  }

  SAFE_RELEASE(pEnum);

  if (found)
    return pFilter;
  else
    return NULL;
}

bool removeAllFilters(IGraphBuilder *pGraph) {

      IEnumFilters *pEnum = NULL;
      IBaseFilter *pFilter;
      ULONG cFetched;
```

```
      std::vector<IBaseFilter *> filters;
      bool done= TRUE;

      pGraph->EnumFilters(&pEnum);
      while(pEnum->Next(1, &pFilter, &cFetched) == S_OK) {

        filters.push_back(pFilter);
      }

      SAFE_RELEASE(pEnum);

      for (int i=0; i<filters.size(); i++) {

        if (done && FAILED(pGraph->RemoveFilter(filters[i]))) {

         ::MessageBox( NULL, "Unable to remove filter", "Error", MB_OK
| MB_ICONINFORMATION );
          done= FALSE;
          break;
        }

        SAFE_RELEASE(filters[i]);
      }

      return done;
}

//IBaseFilter *getFilterByFriendlyName(REFCLSID CLSIDcategory, CString
friendlyName) {
//  // Create the System Device Enumerator.
//  HRESULT hr;
//  IBaseFilter *filter;
//  bool filterFound = false;
//  ICreateDevEnum *pSysDevEnum = NULL;
//  hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
CLSCTX_INPROC_SERVER,
//                        IID_ICreateDevEnum, (void **)&pSysDevEnum);
//  // Obtain a class enumerator for the specified category.
//  IEnumMoniker *pEnumCat = NULL;
//  hr = pSysDevEnum->CreateClassEnumerator(CLSIDcategory, &pEnumCat,
0);
//  if (hr == S_OK)  {
```

```
//    // Enumerate the monikers.
//    IMoniker *pMoniker;
//    ULONG cFetched;
//    while ((pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK) &&
!filterFound) {
//      IPropertyBag *pPropBag;
//      pMoniker->BindToStorage(0, 0, IID_IPropertyBag, (void
**)&pPropBag);
//      // To retrieve the friendly name of the filter, do the
following:
//      VARIANT varName;
//      VariantInit(&varName);
//      hr = pPropBag->Read(L"FriendlyName", &varName, 0);
//      if (SUCCEEDED(hr)) {
//        CString str(varName.bstrVal);
//        if (friendlyName.Compare(str) == 0) {
//          pMoniker->BindToObject(NULL, NULL, IID_IBaseFilter,
(void**)&filter);
//          filterFound = true;
//        }
//        SysFreeString(varName.bstrVal);
//      }
//      // To create an instance of the filter, do the following:
//      //   IBaseFilter *pFilter;
//      //   pMoniker->BindToObject(NULL, NULL, IID_IBaseFilter,
(void**)&pFilter);
//      // Now add the filter to the graph. Remember to release
pFilter later.
//
//      // Clean up.
//      pPropBag->Release();
//      pMoniker->Release();
//    }
//    pEnumCat->Release();
//  }
//  pSysDevEnum->Release();
//
//  if (filterFound)
//    return filter;
//  else
//    return NULL;
//}
```

```
void enumFilters(REFCLSID CLSIDcategory, std::vector<CString>& names,
std::vector<CLSID>& clsidFilters) {
  // Create the System Device Enumerator.
  HRESULT hr;
  ICreateDevEnum *pSysDevEnum = NULL;
  hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
CLSCTX_INPROC_SERVER,
                        IID_ICreateDevEnum, (void **)&pSysDevEnum);
  // Obtain a class enumerator for the specified category.
  IEnumMoniker *pEnumCat = NULL;
  hr = pSysDevEnum->CreateClassEnumerator(CLSIDcategory, &pEnumCat,
0);
  if (hr == S_OK)  {
    // Enumerate the monikers.
    IMoniker *pMoniker;
    ULONG cFetched;
    while(pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK) {
      IPropertyBag *pPropBag;
      pMoniker->BindToStorage(0, 0, IID_IPropertyBag, (void
**)&pPropBag);
      // To retrieve the friendly name of the filter, do the
following:
      VARIANT varName;
      VariantInit(&varName);
      hr = pPropBag->Read(L"FriendlyName", &varName, 0);
      if (SUCCEEDED(hr)) {
        CString str(varName.bstrVal);
        names.push_back(str);
        SysFreeString(varName.bstrVal);
      }
      VariantClear(&varName);
      VARIANT varFilterClsid;
      varFilterClsid.vt = VT_BSTR;
      // Read CLSID string from property bag
      hr = pPropBag->Read(L"CLSID", &varFilterClsid, 0);
      if(SUCCEEDED(hr)) {
        CLSID clsidFilter;
        // Save filter CLSID
        if(CLSIDFromString(varFilterClsid.bstrVal, &clsidFilter) ==
S_OK) {
          clsidFilters.push_back(clsidFilter);
```

```
        }
        SysFreeString(varFilterClsid.bstrVal);
      } else {
          clsidFilters.push_back(GUID_NULL);
      }
      // To create an instance of the filter, do the following:
      //   IBaseFilter *pFilter;
      //   pMoniker->BindToObject(NULL, NULL, IID_IBaseFilter,
(void**)&pFilter);
      // Now add the filter to the graph. Remember to release pFilter
later.

      // Clean up.
      pPropBag->Release();
      pMoniker->Release();
    }
    pEnumCat->Release();
  }
  pSysDevEnum->Release();
}



IBaseFilter* addFilterByEnum(REFCLSID CLSIDcategory, CString
filtername, IGraphBuilder *pGraph,
               IPin **outputPin, int numberOfOutput, bool returnIt) {


  // Create the System Device Enumerator.
  HRESULT hr;
  char tmp[100];
  IBaseFilter* baseFilter = NULL;
  ICreateDevEnum *pSysDevEnum = NULL;
  hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
CLSCTX_INPROC_SERVER,
   IID_ICreateDevEnum, (void **)&pSysDevEnum);


  // Obtain a class enumerator for the specified category.
  IEnumMoniker *pEnumCat = NULL;
  hr = pSysDevEnum->CreateClassEnumerator(CLSIDcategory, &pEnumCat,
0);


  if (hr == S_OK)  {
```

```
   // Enumerate the monikers.
   IMoniker *pMoniker;
   ULONG cFetched;
   bool found= false;

   while(pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK && !found)
   {
     IPropertyBag *pPropBag;
     pMoniker->BindToStorage(0, 0, IID_IPropertyBag, (void
**)&pPropBag);

     // To retrieve the friendly name of the filter, do the following:
     VARIANT varName;
     VariantInit(&varName);
     CString str;
     hr = pPropBag->Read(L"FriendlyName", &varName, 0);
     if (SUCCEEDED(hr))
     {
       str= varName.bstrVal;

       SysFreeString(varName.bstrVal);
     }
     VariantClear(&varName);

     // To create an instance of the filter, do the following:

     if (str == filtername) {

       found= true;
       hr= pMoniker->BindToObject(NULL, NULL, IID_IBaseFilter,
(void**)&baseFilter);
     }

     // Clean up.
     pPropBag->Release();
     pMoniker->Release();
   }

   pEnumCat->Release();

   IPin* inputPin;
```

```
if (*outputPin != NULL) {

  // Obtain the input pin.

  inputPin= GetPin(baseFilter, PINDIR_INPUT);
  if (!inputPin) {

    sprintf(tmp,"Unable to obtain %ls input pin", filtername);
    ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
    return 0;
  }

  // Don't release the input pin here, we need it later.
  //SAFE_RELEASE(inputPin);
}

// Add the filter to the graph.

WCHAR *wfile= new WCHAR[filtername.GetLength()+1];
MultiByteToWideChar(CP_ACP, 0, filtername, -1, wfile,
  filtername.GetLength()+1);
if(FAILED(pGraph->AddFilter( baseFilter, wfile)))
{
  sprintf(tmp,"Unable to add %ls filter", filtername);
  ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
  return 0;
}

delete[] wfile;


if (*outputPin != NULL) {
  // Connect the filter to the ouput pin.

  if(FAILED(pGraph->Connect(*outputPin, inputPin)) )
  {

    sprintf(tmp,"Unable to connect %ls filter", filtername);
    ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
    return 0;
```

```
      }

      // Release both pins.  We will set a new outputpin to be
      // returned in the next section of the function
      SAFE_RELEASE(*outputPin);
      SAFE_RELEASE(inputPin);
    }

    // Obtain the output pin(s).
    // Change the output pin passed in to be the output
    // pin of the newly added filter.
    for (int i=0; i<numberOfOutput; i++) {

      outputPin[i]= 0;
      outputPin[i]= GetPin(baseFilter, PINDIR_OUTPUT, i+1);

      if (!outputPin[i]) {

        sprintf(tmp,"Unable to obtain %s output pin (%d)", filtername,
i);
        ::MessageBox( NULL, tmp, "Error", MB_OK | MB_ICONINFORMATION );
        return 0;
      }
    }
  }

  pSysDevEnum->Release();

  if (returnIt) {

    return baseFilter;

  } else {

    SAFE_RELEASE(baseFilter);
    return 0;
  }
}

bool changeCompressorOptions(IPin **outputPin, int numberOfOutput,
int& iKeyFrames, int& iPFrames, double& dQuality)
```

```
{
    // This function will check over each of the output pins in the
outputPin
    // array to see if any of them support the IAMVideoCompression
interface
    // If the interface is supported on a pin, the properties of the
compressor
    // will be found and displayed.  The given vaules for KeyFrames,
PFrames, and
    // quality will be set if the compressor supports the options, and
the given
    // values aren't -1.  If the value is -1 and the given option IS
supported,
    // the default value will be used.  FALSE will be returned if
there is a
    // problem obtaining the interface, or a value for an option has
been passed
    // but the compressor doesn't support that option.

    int i;
    HRESULT hr = E_FAIL;
    IPin *pPin = NULL;
    IAMVideoCompression *pCompress = NULL;

    long lCap;                      // Capability flags
    long lKeyFrameDef, lPFrameDef; // Default values
    double QualityDef;
    bool bResult = TRUE;

    for (i=0; i < numberOfOutput; i++)
    {
        pPin = outputPin[i];
        hr = pPin->QueryInterface(IID_IAMVideoCompression,
(void**)&pCompress);
        if (SUCCEEDED(hr))
            break;          // Interface found on pin, pCompress is
valid.
    }

    if (SUCCEEDED(hr))
    {

        int cbVersion=0, cbDesc=0; // Size in bytes, not characters!
```

```
        hr = pCompress->GetInfo(0, &cbVersion, 0, &cbDesc, 0, 0, 0,
0);
        if (SUCCEEDED(hr))
        {
            // Fudge the values here since it seems that the operation
of
            // IAMVideoCompression::GetInfo isn't quite as advertised.
The size
            // of the version and description are returned as zero,
but using zero
            // length buffers with 0 size buffer length inputs results
in an exception.
            // It seems safest to set both values to something
reasonably large.

            WCHAR *pszVersion = new WCHAR[512];
            WCHAR *pszDesc = new WCHAR[512];
            cbVersion = 256;
            cbDesc = 256;
            // Get default values and capabilities.
            hr = pCompress->GetInfo(pszVersion, &cbVersion, pszDesc,
&cbDesc, &lKeyFrameDef, &lPFrameDef,
                &QualityDef, &lCap);
            delete[] pszVersion;
            delete[] pszDesc;
        }
        else
            bResult = FALSE;

        if (SUCCEEDED(hr))
        {
            // For any values that aren't -1, attempt to set the
values
            // If the compressor doesn't support the option, return
false
            if (iKeyFrames != -1)
            {
                if (lCap & CompressionCaps_CanKeyFrame)
                {
                    hr = pCompress->put_KeyFrameRate(iKeyFrames);
                    if (FAILED(hr))
                        bResult = FALSE;
                }
```

```
            else
                bResult = FALSE;

        } else {

  iKeyFrames= lKeyFrameDef;
}

        if (iPFrames != -1)
        {
            if (lCap & CompressionCaps_CanBFrame)
            {
                hr = pCompress->put_PFramesPerKeyFrame(iPFrames);
                if (FAILED(hr))
                    bResult = FALSE;
            }
            else
                bResult = FALSE;

        } else {

  iPFrames= lPFrameDef;
}

        if (dQuality >= 0.0)
        {
            if (lCap & CompressionCaps_CanQuality)
            {
                hr = pCompress->put_Quality(dQuality);
                if (FAILED(hr))
                    bResult = FALSE;
            }
            else
                bResult = FALSE;

        } else {

  dQuality= QualityDef;
}
```

```
        }
        else
            bResult = FALSE;

        SAFE_RELEASE(pCompress);
        return bResult;
    }

    return false;
}

bool changeVideoCaptureOptions(IBaseFilter* pVideoSource, long lParam,
long lValue, long lFlags)
{
    // This function takes a pointer to the video capture filter and
    // sets the specified property to the specified value, using the
given
    // flags.  If the option cannot be set for some reason, FALSE is
returned,
    // otherwise the function returns TRUE.  FALSE may be returned in
the
    // case that the property is not available for this device, or
    // the given value is out of range.

    // lParam is the property from the VideoProcAmpProperty
enumeration that
    // you would like to change.  lValue contains the new value to set
for this
    // property.

    // The lFlags variable can have the value 0 or 1, with 1
indicating that the
    // specified parameter is controlled automatically, and 0
indicating otherwise.
    // This feature is not available for all parameters.  If the auto
setting is
    // used, the value specified it ignored.  If lFlags is not passed
to the function,
    // 0, or non-automatic, is assumed.

    IAMVideoProcAmp* pVideoProcAmp = NULL;
    HRESULT hr = E_FAIL;
```

```
    hr = pVideoSource->QueryInterface(IID_IAMVideoProcAmp,
(void**)&pVideoProcAmp);

    if (FAILED(hr))
        return FALSE;

    // Have interface to VideoProcAmp, attempt to change settings
    hr = pVideoProcAmp->Set(lParam, lValue, lFlags);

    if (FAILED(hr))
    {
        SAFE_RELEASE(pVideoProcAmp);
        return FALSE;
    }

    return TRUE;
}

bool changeVideoCaptureOptionsByName(const WCHAR* wcVideoSource,
IFilterGraph* pGraph, long lParam, long lValue, long lFlags)
{
    // This function is almost identical to changeVideoCaptureOptions,
except
    // that it wraps the finding of the video capture device, given
the name
    // of the device.

    bool bResult = FALSE;
    HRESULT hr = E_FAIL;

    IBaseFilter* pVideoSource = NULL;
    hr = pGraph->FindFilterByName(wcVideoSource, &pVideoSource);
    if (FAILED(hr))
        return FALSE;
    bResult= changeVideoCaptureOptions(pVideoSource, lParam, lValue,
lFlags);
    SAFE_RELEASE(pVideoSource);
    return bResult;
}
```

9.1.5 MtxSensor.h

```
#ifndef __MTXSENSOR_H__
```

```
#define __MTXSENSOR_H__

#include "all_includes.h"

// Opciones del encabezado
#define CONTAINS_UP_INFO 1;

typedef struct {
  FXVec3f acc, rawAcc;
  FXQuatf ori;
  double time;
} TMtxData;

typedef struct {
  unsigned int deviceId;
  unsigned int sampleRate;
  unsigned int options;
  float        upAcc[3];
  float        upOri[4];
  char         padding[88];  // Relleno hasta los 128 bytes
} TMtxCaptureFileHeader;

#pragma pack(1)
typedef struct {
  float acc[3];
  float ori[4];
  double timestamp;
} TMtxCaptureFileRecord;
#pragma pack()

typedef enum {DIRECT_DATA, PROCESS_DATA} tDataMode;
typedef enum {READ_DEVICE, READ_LOG, READ_CAPTURE} tReadMode;
typedef enum {WORLD_COORDINATE, SENSOR_COORDINATE} tFrameMode;

class MtxSensor {
  // Sensor variables
  long              cmtInstance;
  unsigned short    mtCount;
  unsigned short    mtxPortNumber;
  unsigned short    sampleFreq;
```

```
  CmtOutputSettings settings;
  CmtOutputMode     mode;
  CmtDeviceId       deviceIds[256];
  FXText                                *logBox;

  FXFile            captureFile;
  char              *fileBuffer;
  unsigned int      fileBufferSize;
  unsigned int      fileBufferCounter;
  tbb::tick_count   baseTimestamp;
  tbb::mutex        logMutex;

  tDataMode         dataMode;
  tbb::mutex        sensorDataMutex;
  tbb::tbb_thread       *readSensorThread;
  bool              continuousRead;

  TMtxData          sensorData;
  double            lastTime;
  FXVec3f           gVec;

  bool              deviceInitialized;
  bool              logEnabled;         // Indica si se esta
capturando en este momento
  bool              logCreated;         // Indica si se ha creado el
fichero de log
  bool              captureToMCF;       // Si True estamos usando
nuestro tipo de log
  FXString          logFileName;
  tReadMode         readMode;

  void logMessage(const char *format, ...);
  XsensResultValue doMtxScan();
  XsensResultValue doMtxSettings();
  int continuousSensorReading();
  XsensResultValue reset();
  XsensResultValue readFromCmt(TMtxData *mtxData);
  XsensResultValue readFromCaptureFile(TMtxData *mtxData);
public:
  MtxSensor(FXText *logBox = NULL, tDataMode dataMode = DIRECT_DATA,
const char *fileName = NULL);
  ~MtxSensor();
```

```
  XsensResultValue setBaseTimeStamp(const tbb::tick_count &timestamp);
  XsensResultValue resetBaseTimeStamp();
  XsensResultValue createLog(const char *fileNameToLog);
  XsensResultValue enableLog();
  XsensResultValue disableLog();
  XsensResultValue closeLog();
  XsensResultValue addToLog(const TMtxCaptureFileRecord *fileRecord);

  tReadMode getReadMode() {return readMode;};
  int init();
  void run();
//  int reset();
  XsensResultValue seek(long sample);
  XsensResultValue read(TMtxData *mtxData);
  friend void mtxThreadEncapsulator (MtxSensor *mainApp);
};

#endif
```

9.1.6 MtxSensor.cpp

```
#include "MtxSensor.h"
#define _USE_MATH_DEFINES
#include <math.h>


#define EXIT_ON_ERROR(res,comment) if (res != XRV_OK) {
logMessage("Error %d occurred in " comment ":
%s\n",res,cmtGetResultText(res)); return(res); }
#define LOG_ON_ERROR(res,comment) if (res != XRV_OK) {
logMessage("Error %d occurred in " comment ":
%s\n",res,cmtGetResultText(res)); }
#define RECORDS_IN_FILE_BUFFER 120


static const char serialNumber[] = "2020E-C56A3-5CCD0-05B02";
static const int ACC_HISTORY_SIZE = 10;
static const int G_MEAN_SAMPLES = 20;
static const double PI_DIV_180 = M_PI / 180.0;

static double round(double x) {
  return x < 0 ? -floor(fabs(x) + .5) : floor(x + .5);
}
static double trunc(double x) {
  return x < 0 ? ceil(x) : floor(x);
```

```
}
    //if ((fileName.length() < 4) ||
    //        (fileName.find(".cdp", fileName.length() - 4) !=
(fileName.length() - 4)))
    //            fileName.append(".cdp");
/*-------------------------------------------------------------------
------*/
MtxSensor::MtxSensor(FXText *logBox, tDataMode dataMode, const char
*fileName) {
  this->logBox = logBox;
  this->dataMode = dataMode;

  cmtInstance = -1;
  logEnabled = false;
  logCreated = false;
  deviceInitialized = false;
  captureToMCF = false;

  sensorData.rawAcc.set(0.0, 0.0, 0.0);
  sensorData.acc.set(0.0, 0.0, 0.0);
  sensorData.time = 0;

  if (fileName != NULL) {
    logFileName = fileName;
    if (logFileName.find(".mtb", logFileName.length() - 4) ==
(logFileName.length() - 4)) {
    // Si es un fichero de log mtb
      readMode = READ_LOG;
    } else if (logFileName.find(".mcf", logFileName.length() - 4) ==
(logFileName.length() - 4)) {
    // Si es un fichero de captura de datos
      readMode = READ_CAPTURE;
    }
  } else {
    readMode = READ_DEVICE;
  }
};

/*-------------------------------------------------------------------
------*/
MtxSensor::~MtxSensor() {
  // Paramos el thread de procesamiento de datos si estuviese
corriendo
```

```
  if (dataMode == PROCESS_DATA) {
    continuousRead = false;
    readSensorThread->join();
  }
  // Cerramos el fichero de captura de datos si estuviese abierto
  if (logCreated)
    closeLog();

  switch (readMode) {
    case READ_DEVICE:
      if (logEnabled) {
        if (captureToMCF && captureFile.isOpen())
          captureFile.close();
        else
          cmtCloseLogFile(cmtInstance, logFileName.text());
      }
      break;
    case READ_LOG:
      cmtCloseLogFile(cmtInstance, logFileName.text());
      break;
    case READ_CAPTURE:
      captureFile.close();
      break;
  }

  if (cmtInstance != -1)
    cmtClose(cmtInstance);
}

/*-------------------------------------------------------------------
------*/
int MtxSensor::init() {
  XsensResultValue res;

  if ((readMode == READ_DEVICE) || (readMode == READ_LOG)) {
    cmtInstance = cmtCreateInstance(serialNumber);
    if (cmtInstance != -1)
      logMessage("CMT instance created\n\n");
    else {
      logMessage("Creation of CMT instance failed, probably because of
an invalid serial number\n");
```

```
      return XRV_ERROR;
    }
  }

  switch (readMode) {
    case READ_DEVICE:
      res = doMtxScan();
      if (res != XRV_OK)
        return -1;
      res = doMtxSettings();
      if (res != XRV_OK)
        return -1;
      reset();
      break;
    case READ_LOG:
      logMessage("Opening device log file \"%s\"\n", logFileName);
      res = cmtOpenLogFile(cmtInstance, logFileName.text());
      EXIT_ON_ERROR(res, "cmtOpenLogFile");
      res = doMtxScan();
      EXIT_ON_ERROR(res, "doMtxScan");
      logEnabled = true;
      break;
    case READ_CAPTURE:
      TMtxCaptureFileHeader header;
      logMessage("Opening capture file \"%s\"\n", logFileName);
      if (captureFile.open(logFileName)) {
        captureFile.readBlock(&header, sizeof(TMtxCaptureFileHeader));
        sampleFreq = header.sampleRate;
        //if (header.options & CONTAINS_UP_INFO) {
        //  upQuat.w = header.upOri[0];
        //  upQuat.x = header.upOri[1];
        //  upQuat.y = header.upOri[2];
        //  upQuat.z = header.upOri[3];
        //  upAcc.set(header.upAcc[0], header.upAcc[1],
header.upAcc[2]);
        //  upVectorPresent = true;
        //}
        logEnabled = true;
      } else {
        return XRV_ERROR;
      }
```

```
      // leer la cabecera del fichero
      break;
    default:
      return -1;
  }

  deviceInitialized = true;
  return 0;
}

/*-------------------------------------------------------------------
------*/
void MtxSensor::run() {
  if (readMode == PROCESS_DATA)
    readSensorThread = new tbb::tbb_thread(mtxThreadEncapsulator,
this);
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::setBaseTimeStamp(const tbb::tick_count
&timestamp) {
  if (logEnabled) {
    return XRV_INVALIDOPERATION;
  } else {
    baseTimestamp = timestamp;
    return XRV_OK;
  }
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::resetBaseTimeStamp() {
  if (logEnabled) {
    return XRV_INVALIDOPERATION;
  } else {
    baseTimestamp = tbb::tick_count::now();
    return XRV_OK;
  }
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::createLog(const char *fileNameToLog) {
```

```
  XsensResultValue res;
  TMtxCaptureFileHeader fileHeader;


  // Si el dispositivo esta inicializado y no se ha abierto ya un log
de lectura o escritura
  if ((readMode == READ_DEVICE) && deviceInitialized && !logCreated) {
    logFileName = fileNameToLog;


    if (logFileName.find(".mcf", logFileName.length() - 4) ==
(logFileName.length() - 4)) {
      // Si es un fichero de captura
      // Crear el fichero
      if (captureFile.open(logFileName, FXIO::Writing)) {
        logMessage("Saving capture file \"%s\"\n", logFileName);
        // Escribir la cabecera
        fileHeader.deviceId = deviceIds[0];
        fileHeader.sampleRate = sampleFreq;
        memset(fileHeader.padding, 0, 88);
        captureFile.writeBlock(&fileHeader,
sizeof(TMtxCaptureFileHeader));
        fileBufferSize = sizeof(TMtxCaptureFileRecord) *
RECORDS_IN_FILE_BUFFER;
        fileBuffer = new char[fileBufferSize];
        fileBufferCounter = 0;
        captureToMCF = true;
        logCreated = true;
        resetBaseTimeStamp();
        return XRV_OK;
      } else {
        logMessage("Can't open file \"%s\" for writing.\n",
logFileName);
        return XRV_ERROR;
      }
    } else {
      // Si es un fichero de log del dispositivo
      res = cmtCreateLogFile(cmtInstance, mtxPortNumber,
logFileName.text());
      EXIT_ON_ERROR(res, "cmtCreateLogFile");
      logMessage("Saving log file \"%s\"\n", logFileName);


      res = cmtSetLogMode(cmtInstance, 1, deviceIds[0]);
      if (res != XRV_OK) {
```

```
        cmtCloseLogFile(cmtInstance, logFileName.text());
        EXIT_ON_ERROR(res, "cmtSetLogMode");
      } else {
        captureToMCF = false;
        logCreated = true;
        return XRV_OK;
      }
    }
  } else
    return XRV_INVALIDOPERATION;
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::enableLog() {
  if ((!captureToMCF) || logEnabled)
    return XRV_INVALIDOPERATION ;

  fileBufferCounter = 0;
  logEnabled = true;
  return XRV_OK;
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::disableLog() {
  tbb::mutex::scoped_lock(logMutex);

  if ((!captureToMCF) || (!logEnabled))
    return XRV_INVALIDOPERATION;

  logEnabled = false;
  // Si hay cosas pendientes por escribir...
  if (fileBufferCounter > 0) {
    captureFile.writeBlock(fileBuffer, fileBufferCounter);
    fileBufferCounter = 0;
  }
  return XRV_OK;
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::closeLog() {
  XsensResultValue res;
```

```
  if ((readMode == READ_DEVICE) && logCreated) {
    if (captureToMCF) {
      disableLog();
      captureToMCF = false;
      logCreated = false;
      // Si estamos usando el modo de captura de datos
      //logMessage("Closing capture file \"%s\" for writing.\n",
logFileName);
      captureFile.close();
      delete fileBuffer;
    } else {
      // Si usamos la funcion de log interna de la lib de Mtx
      res = cmtSetLogMode(cmtInstance, 0,deviceIds[0]);
      LOG_ON_ERROR(res, "cmtSetLogMode");
      res = cmtCloseLogFile(cmtInstance, logFileName.text());
      logCreated = false;
      logEnabled = false;
      return res;
    }
  } else
    return XRV_INVALIDOPERATION;
}
/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::addToLog(const TMtxCaptureFileRecord
*fileRecord) {
  tbb::mutex::scoped_lock(logMutex);


  if (!logEnabled)           // Si el dispositivo no estaba logueando
devuelve error
    return XRV_INVALIDOPERATION;


  memcpy(fileBuffer + fileBufferCounter, fileRecord,
sizeof(TMtxCaptureFileRecord));
  fileBufferCounter += sizeof(TMtxCaptureFileRecord);
  // Posición y orientacion P1,Q1,P2,Q2...
  if (fileBufferCounter == fileBufferSize) {
    captureFile.writeBlock(fileBuffer, fileBufferSize);
    fileBufferCounter = 0;
  }
  return XRV_OK;
```

```
}
/*-------------------------------------------------------------------
------*/
inline void MtxSensor::logMessage(const char *format, ...) {
  char logString[255];
  va_list args;

  if (logBox) {
    va_start(args, format);
    vsprintf(logString, format, args);
    va_end( args );
    logBox->appendText(logString);

    FXint lastPos = logBox->getLength();
    // Use rowStart() to find the position of the start of the row
    FXint lastRowStartPos = logBox->rowStart(lastPos);
    // Make that position visible
    logBox->makePositionVisible(lastRowStartPos);
  }
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::doMtxScan() {
  XsensResultValue res;
  CmtPortInfo portInfo[256];
  unsigned int portCount = 0;

  if (readMode == READ_DEVICE) {
    logMessage("Scanning for connected Xsens devices...\n");
    res = cmtScanPorts(portInfo, &portCount, 0);
    EXIT_ON_ERROR(res, "cmtScanPorts...");

    if (portCount == 0) {
      logMessage("No motion tracker found!!\n");
      return XRV_ERROR;
    };

    for(int i = 0; i < (int)portCount; i++)
      logMessage("Using COM port %d at %d baud\n\n",
```

```
                (long)portInfo[i].m_portNr, portInfo[i].m_baudrate);


    logMessage("Opening ports...");


    //open the port which the device is connected to and connect at
the device's baudrate.
    for(int p = 0; p < (int)portCount; p++){
      res = cmtOpenPort(cmtInstance, portInfo[p].m_portNr,
portInfo[p].m_baudrate);
      EXIT_ON_ERROR(res,"cmtOpenPort");
    }


    mtxPortNumber = portInfo[0].m_portNr;


    logMessage("done\n\n");
  }


  //get the Mt sensor count.
  logMessage("Retrieving MotionTracker count (excluding attached Xbus
Master(s))\n");
  res = cmtGetMtCount(cmtInstance,&mtCount);
  EXIT_ON_ERROR(res,"cmtGetMtCount");
  logMessage("MotionTracker count: %i\n\n",mtCount);


  // retrieve the device IDs
  logMessage("Retrieving MotionTrackers device ID(s)\n");
  for(unsigned int j = 0; j < mtCount; j++ ){
    res = cmtGetMtDeviceId(cmtInstance, &deviceIds[j], j);
    EXIT_ON_ERROR(res,"cmtGetDeviceId");
    logMessage("Device ID at index %i: %08x\n",j,(long) deviceIds[j]);
  }
  return XRV_OK;
}


/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::doMtxSettings() {
  XsensResultValue res;


  // set sensor to config sate
  res = cmtGotoConfig(cmtInstance);
```

```
  EXIT_ON_ERROR(res, "cmtGotoConfig");


  double gMag;
  res = cmtGetGravityMagnitude(cmtInstance, &gMag, deviceIds[0]);
  EXIT_ON_ERROR(res, "cmtGetGravityMagnitude");
  gVec = FXVec3f(.0, .0, gMag);
  logMessage("Gravity vector: (%f, %f, %f)\n", gVec.x, gVec.y,
gVec.z);


  res = cmtGetSampleFrequency(cmtInstance, &sampleFreq, deviceIds[0]);
  EXIT_ON_ERROR(res, "cmtGetSampleFrequency");
  logMessage("Sample frequency: %d\n", sampleFreq);


  // set the device output mode for the device(s)
  // settings = CMT_OUTPUTSETTINGS_ORIENTMODE_EULER |
CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT;
  settings = CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION |
CMT_OUTPUTSETTINGS_TIMESTAMP_SAMPLECNT;
  mode = CMT_OUTPUTMODE_ORIENT | CMT_OUTPUTMODE_CALIB;


  printf("Configuring your mode selection");
  for (int i=0; i < mtCount; i++) {
    if (cmtIdIsMtig(deviceIds[i])) {
      res = cmtSetDeviceMode(cmtInstance, mode, settings, sampleFreq,
deviceIds[i]);
    } else {
      res = cmtSetDeviceMode(cmtInstance, mode & 0xFF0F, settings,
sampleFreq, deviceIds[i]);
    }
    EXIT_ON_ERROR(res, "setDeviceMode");
  }


  // make sure that we get the freshest data
  logMessage("\nSetting queue mode so that we always get the latest
data\n\n");
  res = cmtSetQueueMode(cmtInstance,CMT_QM_LAST);
  //logMessage("\nSetting queue mode.\n\n");
  //res = cmtSetQueueMode(cmtInstance,CMT_QM_FIFO);
  EXIT_ON_ERROR(res,"cmtSetQueueMode");


  // Enable software calibration
  res = cmtSetSoftwareCalibration(cmtInstance, 1);
```

```
  LOG_ON_ERROR(res, "cmtSetSoftwareCalibration");
  // switch on Kalman filter
  res = cmtSetSoftwareXkf3Filtering(cmtInstance, 1);
  LOG_ON_ERROR(res, "cmtSetSoftwareXkf3Filtering");


  //CmtScenario scenarios[CMT_MAX_SCENARIOS_IN_MT+1];
  //res = cmtGetAvailableSoftwareScenarios(cmtInstance, scenarios,
deviceIds[0]);
  //LOG_ON_ERROR(res, "cmtGetAvailableSoftwareScenarios");
  //res = cmtSetSoftwareScenario(cmtInstance, scenarios[0].m_type,
deviceIds[0]);
  //LOG_ON_ERROR(res, "cmtSetSoftwareScenario");


  // start receiving data
  res = cmtGotoMeasurement(cmtInstance);
  EXIT_ON_ERROR(res, "cmtGotoMeasurement");


  Sleep(500);


  return XRV_OK;
}


/*-------------------------------------------------------------------
------*/
int MtxSensor::continuousSensorReading() {
  //XsensResultValue res;
  //unsigned short sample;
  //CmtVector accData;
  //CmtQuat   quatData;
  //double time;
  //FXVec3f gVec, lastSpd, lastAcc, curAcc, filAcc, tmpVec;
  //FXQuatf oriQuat;
  //
  //int accFilterIndex;
  //static const int FILTER_SIZE = 5;
  //FXVec3d accFilterVector[FILTER_SIZE];


  //for (int i = 0; i < FILTER_SIZE; i++)
  //   accFilterVector[i].set(.0, .0, .0);
  //accFilterIndex = 0;
```

```
  //FXQuatf rotQuat(FXVec3f(1.0, 1.0, 1.0), -(2.0/3.0) * PI);   //
Quaternion de rotacion
  //gVec.set(.0, gMag, .0);                                      //
Vector gravedad


  //curPos.set(.0, .0, .0);
  //curSpd.set(.0, .0, .0);
  //filAcc.set(.0, .0, .0);
  //tmpVec.set(.0, .0, .0);
  //lastAcc.set(.0, .0, .0);
  //lastSpd.set(.0, .0, .0);


  //// Reseteamos el sensor y obtenemos los valores iniciales
  ////res = cmtResetOrientation(cmtInstance,
CMT_RESETORIENTATION_GLOBAL, deviceIds[0]);
  ////LOG_ON_ERROR(res, "cmtResetOrientation");


  //res = cmtGetNextDataBundle(cmtInstance);
  //LOG_ON_ERROR(res, "cmtGetNextDataBundle");
  //res = cmtDataGetSampleCounter(cmtInstance, &sample, deviceIds[0]);
  //LOG_ON_ERROR(res, "cmtDataGetSampleCounter");
  //lastTime = double(sample) / double(sampleFreq); // tiempo en
segundos


  //continuousRead = true;


  //while (continuousRead) {
  //  res = cmtGetNextDataBundle(cmtInstance);
  //  if (res == XRV_OK) {
  //    // Lectura de los datos del sensor
  //      // Tiempo
  //    res = cmtDataGetSampleCounter(cmtInstance, &sample,
deviceIds[0]);
  //    LOG_ON_ERROR(res, "cmtDataGetSampleCounter");
  //      // Quaternion de orientación
  //    res = cmtDataGetOriQuat(cmtInstance, &quatData, deviceIds[0]);
  //    LOG_ON_ERROR(res, "cmtDataGetOriQuat");
  //    oriQuat.w = quatData.m_data[0];
  //    oriQuat.x = quatData.m_data[1];
  //    oriQuat.y = quatData.m_data[2];
  //    oriQuat.z = quatData.m_data[3];
  //    oriQuat = rotQuat * oriQuat;
```

```
  //      // Aceleración
  //    res = cmtDataGetCalAcc(cmtInstance, &accData, deviceIds[0]);
  //    LOG_ON_ERROR(res, "cmtDataGetCalAcc");
  //    curAcc.set(accData.m_data[0], accData.m_data[1],
accData.m_data[2]);
  //    curAcc = (oriQuat * curAcc) - gVec;  // Proyectamos la acc y
sustraemos G
  //      // Filtramos los datos de aceleracion con una media
  //    accFilterIndex = (accFilterIndex + 1) % FILTER_SIZE;
  //    filAcc = filAcc - accFilterVector[accFilterIndex] + (curAcc /
float(FILTER_SIZE));
  //    accFilterVector[accFilterIndex] = curAcc / float(FILTER_SIZE);


  //    filAcc.x = trunc(filAcc.x * 100.0) / 100.0;
  //    filAcc.y = trunc(filAcc.y * 100.0) / 100.0;
  //    filAcc.z = trunc(filAcc.z * 100.0) / 100.0;


  //    //filAcc = curAcc;


  //      // Tiempo en segundos
  //    time = double(sample) / double(sampleFreq); // tiempo en
segundos
  //    // Calculamos la posición en función de la aceleración medida
  //    // dT = t2 - t1; p = p0 + v0 * dT + a * (dt*dt) / 2;
  //    float dT = (time - lastTime);


  //    curSpd = curSpd + (float(dT / 2.0) * (lastAcc + filAcc));
  //    curPos = curPos + (float(dT / 2.0) * (lastSpd + curSpd));


  //    lastAcc = filAcc;
  //    lastSpd = curSpd;
  //    lastTime = time;
  //    { // Se asigna en exclusion mutua el nuevo paquete de datos
recalculados
  //      tbb::mutex::scoped_lock lock(sensorDataMutex);
  //      sensorData.acc = filAcc;
  //      sensorData.spd = curSpd;
  //      sensorData.pos = curPos;
  //      sensorData.ori = oriQuat;
  //      sensorData.time = time;
  //    }
  //  } else {
```

```
  //    Sleep(7); // 120Hz -> 8,3 ms
  //  }
  //}

  return 0;
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::seek(long sample) {
  long pos, posc;

  if (readMode == READ_CAPTURE) {
    pos = sizeof(TMtxCaptureFileHeader) + (sample *
sizeof(TMtxCaptureFileRecord));
    posc = captureFile.position(pos);
    if (pos == posc)
      return XRV_OK;
    else
      return XRV_ERROR;
  } else {
    return XRV_INVALIDOPERATION;
  }
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::readFromCmt(TMtxData *mtxData) {
  XsensResultValue  res;
  CmtVector         accData;
  CmtQuat           quatData;
  unsigned short    sample;

  res = cmtGetNextDataBundle(cmtInstance);

  tbb::tick_count timestamp = tbb::tick_count::now();
  tbb::tick_count::interval_t interval = timestamp - baseTimestamp;

  if (res == XRV_OK) {
    // Lectura de los datos del sensor
      // Tiempo
```

```
    res = cmtDataGetSampleCounter(cmtInstance, &sample, deviceIds[0]);
    LOG_ON_ERROR(res, "cmtDataGetSampleCounter");
    mtxData->time = interval.seconds();
      // Quaternion de orientación
    res = cmtDataGetOriQuat(cmtInstance, &quatData, deviceIds[0]);
    LOG_ON_ERROR(res, "cmtDataGetOriQuat");
    mtxData->ori.w = quatData.m_data[0];
    mtxData->ori.x = quatData.m_data[1];
    mtxData->ori.y = quatData.m_data[2];
    mtxData->ori.z = quatData.m_data[3];
      // Aceleración
    res = cmtDataGetCalAcc(cmtInstance, &accData, deviceIds[0]);
    LOG_ON_ERROR(res, "cmtDataGetCalAcc");
    mtxData->acc.set(accData.m_data[0], accData.m_data[1],
accData.m_data[2]);
    if (captureToMCF) {
      TMtxCaptureFileRecord record;
      record.acc[0] = accData.m_data[0];
      record.acc[1] = accData.m_data[1];
      record.acc[2] = accData.m_data[2];
      record.ori[0] = quatData.m_data[0];
      record.ori[1] = quatData.m_data[1];
      record.ori[2] = quatData.m_data[2];
      record.ori[3] = quatData.m_data[3];
      record.timestamp = interval.seconds();
      addToLog(&record);
    }
  }
  return res;
}


/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::readFromCaptureFile(TMtxData *mtxData) {
  TMtxCaptureFileRecord record;


  if (captureFile.readBlock(&record, sizeof(TMtxCaptureFileRecord)) ==
sizeof(TMtxCaptureFileRecord)) {
    mtxData->acc.x = record.acc[0];
    mtxData->acc.y = record.acc[1];
    mtxData->acc.z = record.acc[2];
```

```
    mtxData->ori.w = record.ori[0];
    mtxData->ori.x = record.ori[1];
    mtxData->ori.y = record.ori[2];
    mtxData->ori.z = record.ori[3];
    return XRV_OK;
  } else
    return XRV_ERROR;
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::read(TMtxData *mtxData) {
  XsensResultValue res;

  if (dataMode == DIRECT_DATA) {
    if (readMode == READ_CAPTURE)
      res = readFromCaptureFile(mtxData);
    else
      res = readFromCmt(mtxData);
      mtxData->acc = mtxData->acc - (mtxData->ori.conj() * gVec);
  } else {
    // Se lee el paquete de datos en exclusion mutua
    tbb::mutex::scoped_lock lock(sensorDataMutex);
    *mtxData = sensorData;
    res = XRV_OK;
  }

  //if (coordinateFrame == WORLD_COORDINATE) {
      // Calculamos la rotación actual con respecto al de la vertical
      //FXQuatd rot = mtxData->ori - upQuat;
      //FXVec3d acc = mtxData->acc;
      //mtxData->acc = acc - (rot * upAcc);
  //}
  return res;
}

/*-------------------------------------------------------------------
------*/
XsensResultValue MtxSensor::reset() {
  XsensResultValue res;
```

```
  res = cmtResetOrientation(cmtInstance, CMT_RESETORIENTATION_ALIGN,
deviceIds[0]);

  LOG_ON_ERROR(res, "cmtResetOrientation");

  return res;
}

/*-----------------------------------------------------------------
------*/
void mtxThreadEncapsulator (MtxSensor *mtxSensor) {
  mtxSensor->continuousSensorReading();
}
```

9.1.7 OpticaFlow.h

```
#pragma once
#include "all_includes.h"

typedef struct {
  unsigned int hd, vd, featuresPerQuadrant;
  char padding[128 - (3 * sizeof(unsigned int))];
} TOFFileHeader;

#pragma pack(1)
typedef struct {
  double time;
  unsigned int count;
  CvPoint2D32f prev[10];
  CvPoint2D32f curr[10];
} TOFFileRecord;
#pragma pack()

class OpticalVectorsList {
  int maxCount;
public:
  CvPoint2D32f *prevPoints, *currPoints;
  unsigned int count;

  OpticalVectorsList(int maxCount) {
    currPoints = (CvPoint2D32f*)cvAlloc(maxCount *
sizeof(CvPoint2D32f));
```

```
    prevPoints = (CvPoint2D32f*)cvAlloc(maxCount *
sizeof(CvPoint2D32f));
    this->maxCount = maxCount;
    count = 0;
  }

  ~OpticalVectorsList() {
    cvFree(&prevPoints);
    cvFree(&currPoints);
  }

  int getMaxCount() { return maxCount; };
};

class OpticalFlowEstimator {
  // parametros del lk
  double          quality;
  double          min_distance;
  int             win_size;
  int             hd, vd, featuresPerQuadrant;

  bool            initialized;
  bool            logEnabled;
  bool            logCreated;
  tbb::mutex      logMutex;
  FXFile          logFile;
  char            *fileBuffer;
  unsigned int    fileBufferSize;
  unsigned int    fileBufferCounter;
  tbb::tick_count baseTimestamp;

  IplImage        *_currGray, *_prevGray,
                  *_currPyramid, *_prevPyramid,
                  *_eig, *_temp,
                  *_swapTemp;
  CvPoint2D32f    *_points, *_swapPoints;
  char            *_lkStatus;
  int             *_pointsPerQuadrant;

  int addFeaturesToTrack(IplImage *grayImage, CvPoint2D32f *points,
```

```
                          int currCount, int desiredCount, int
minDistance,
                          const CvRect *roi = NULL);
  int sparseFeaturesInQuadrants(IplImage *grayImage,
                                CvPoint2D32f *points, int currCount,
                                int hd, int vd, int
featuresPerQuadrant);
  bool checkPointAndFix(CvPoint2D32f *point, CvSize imageSize);


public:
  OpticalFlowEstimator(CvSize imageSize, int hd, int vd, int
featuresPerQuadrant);
  ~OpticalFlowEstimator();
  void firstOpticalFlow (const IplImage *prevImage, const IplImage
*currImage,
                          OpticalVectorsList *opticalVectors);
  void getOpticalFlow (const IplImage *currImage, OpticalVectorsList
*opticalVectors);
  bool setBaseTimeStamp(const tbb::tick_count &timestamp);
  bool resetBaseTimeStamp();
  bool createLog(const char *fileName);
  bool enableLog();
  bool disableLog();
  bool closeLog();
  bool addToLog(const OpticalVectorsList *ovList);
  bool setBaseTimestamp(tbb::tick_count baseTimestamp);
};
```

9.1.8 OpticaFlow.cpp

```
#include <OpticalFlow.h>
#include <math.h>


static const int MAX_FEATURES_TO_FIND = 200;
static const int MIN_DISTANCE = 30;
static const int RECORDS_IN_OF_FILE_BUFFER = 120;
static const float INVALID_FLOW = 1000000.0;



/*-----------------------------------------------------------------
---------------------*/
OpticalFlowEstimator::OpticalFlowEstimator(CvSize imageSize, int hd,
int vd,
```

```
                                                int featuresPerQuadrant) {
  quality = 0.005;
  win_size = 5;
  this->hd = hd;
  this->vd = vd;
  this->featuresPerQuadrant = featuresPerQuadrant;

  initialized = false;
  logEnabled = false;
  logCreated = false;

  _currGray     = cvCreateImage(imageSize, 8, 1);
  _prevGray     = cvCreateImage(imageSize, 8, 1);
  _currPyramid  = cvCreateImage(imageSize, 8, 1);
  _prevPyramid  = cvCreateImage(imageSize, 8, 1);
  _eig          = cvCreateImage(imageSize, 32, 1);
  _temp         = cvCreateImage(imageSize, 32, 1);

  _points       = (CvPoint2D32f*)cvAlloc(MAX_FEATURES_TO_FIND *
sizeof(CvPoint2D32f));
  _lkStatus     = (char*)cvAlloc(imageSize.height * imageSize.width *
sizeof(char));

  _pointsPerQuadrant = (int *)malloc(100 * sizeof(int));
}

/*-------------------------------------------------------------------
---------------------*/
OpticalFlowEstimator::~OpticalFlowEstimator() {
  closeLog();
  cvReleaseImage(&_currGray);
  cvReleaseImage(&_prevGray);
  cvReleaseImage(&_currPyramid);
  cvReleaseImage(&_prevPyramid);
  cvReleaseImage(&_eig);
  cvReleaseImage(&_temp);

  cvFree(&_points);
  cvFree(&_lkStatus);

  free(_pointsPerQuadrant);
```

```
}

/*-------------------------------------------------------------------
--------------------*/
int OpticalFlowEstimator::addFeaturesToTrack(IplImage *grayImage,
CvPoint2D32f *points, int currCount, int desiredCount, int
minDistance, const CvRect *roi) {
  int count, foundPoints;
  int i, j, addedPoints;
  CvRect myROI;

  if (desiredCount <= currCount)
    return -1;

  count = (int)(((desiredCount - currCount) * 1.5) + 0.5);

  if (roi !=  NULL) {
    myROI = *roi;
    cvSetImageROI(grayImage, myROI);
    cvSetImageROI(_eig, myROI);
    cvSetImageROI(_temp, myROI);
  } else {
    myROI = cvRect(0, 0, grayImage->width, grayImage->height);
  }

  cvGoodFeaturesToTrack(grayImage, _eig, _temp, _points, &count,
                        quality, minDistance, 0, 3, 1, 0.04 );

  cvFindCornerSubPix(grayImage, _points, count,
cvSize(win_size,win_size), cvSize(-1,-1),

cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03));

  if (currCount == 0) {
    // Para inicializar una lista, se llena directamente
    if (count < desiredCount) {
      for (i = 0; i < count; i++) {
        points[i].x = _points[i].x + myROI.x;
        points[i].y = _points[i].y + myROI.y;
      }
      foundPoints = count;
    } else {
```

```
      for (i = 0; i < desiredCount; i++) {
        points[i].x = _points[i].x + myROI.x;
        points[i].y = _points[i].y + myROI.y;
      }
      foundPoints = desiredCount;
    }
  } else {
    // Para agregar a una lista existente hay que comprobar que
respeten la distancia
    i = 0;
    addedPoints = 0;
    while ((i < count) && (addedPoints < (desiredCount - currCount)))
{
      // nos aseguramos de que el punto a añadir no este demasiado
cerca de uno existente
      for (j = 0; j < currCount; j++) {
        int dist = (((_points[i].x + myROI.x) - points[j].x) *
((_points[i].x + myROI.x) - points[j].x)) +
                   (((_points[i].y + myROI.y) - points[j].y) *
((_points[i].y + myROI.y) - points[j].y));
        if (dist < (minDistance * minDistance))
          break;
      }
      // si el punto es válido
      if (j == currCount) {
        points[currCount + addedPoints].x = _points[i].x + myROI.x;
        points[currCount + addedPoints].y = _points[i].y + myROI.y;
        addedPoints++;
      }
      i++;
    }
    foundPoints = currCount + addedPoints;
  }

  if (roi !=  NULL) {
    cvResetImageROI(grayImage);
    cvResetImageROI(_eig);
    cvResetImageROI(_temp);
  }

  return foundPoints;
}
```

```
/*-------------------------------------------------------------------
--------------------*/
int OpticalFlowEstimator::sparseFeaturesInQuadrants(IplImage
*grayImage, CvPoint2D32f *points, int currCount, int hd, int vd, int
featuresPerQuadrant) {
  int validPoints;


  int vQuadrantSize = grayImage->height / vd;
  int hQuadrantSize = grayImage->width / hd;


  // reseteamos el contador de puntos por cuadrante
  for (int i = 0; i < (hd * vd); i++)
    _pointsPerQuadrant[i] = 0;
  // Contamos el numero de features por cuadrante que hay
  validPoints = 0;
  for (int i = 0; i < currCount; i++) {
    int currQuadrant = (int)(floor(points[i].y /
(float)vQuadrantSize)) * hd +
                       (int)(floor(points[i].x /
(float)hQuadrantSize));
    //if ((currQuadrant > ((hd * vd) - 1)) || (currQuadrant < 0))
    //  printf("currQuadrant > 11\n");
    // Eliminamos los puntos sobrantes en los cuadrantes
    if (_pointsPerQuadrant[currQuadrant] < featuresPerQuadrant) {
      points[validPoints++] = points[i];
      _pointsPerQuadrant[currQuadrant]++;
    }
  }
  // Para cada cuadrante, si hay que añadir features las agregamos
  for (int i = 0; i < vd; i++)
    for (int j = 0; j < hd; j++) {
      if (_pointsPerQuadrant[i * hd + j] < featuresPerQuadrant) {
        // Hacemas la mascara para el cuadrante
        CvRect roi = cvRect(j * hQuadrantSize, i * vQuadrantSize,
                            hQuadrantSize, vQuadrantSize);
        // Agregamos los puntos
        int desiredPoints = validPoints + featuresPerQuadrant -
_pointsPerQuadrant[i * hd + j];
        validPoints = addFeaturesToTrack(grayImage, points,
validPoints, desiredPoints, MIN_DISTANCE, &roi);
      }
```

```
    }


  //for (int i = 0; i < validPoints; i++)
  //  if ((points[i].x > 639) || (points[i].x < 0) || (points[i].y >
479) || (points[i].y < 0))
  //    printf ("Error\n");



  // Eliminamos los puntos que salgan de rango
  int j = 0;
  for (int i = 0; i < validPoints; i++) {
    if (checkPointAndFix(&(points[i]), cvSize(grayImage->width,
grayImage->height))) {
      points[j++] = points[i];
    }
  }
  validPoints = j;


  for (int i = 0; i < validPoints; i++)
    if ((points[i].x > 639) || (points[i].x < 0) || (points[i].y >
479) || (points[i].y < 0))
      printf ("Error\n");
  return validPoints;
}


/*-------------------------------------------------------------------
---------------------*/
bool OpticalFlowEstimator::checkPointAndFix(CvPoint2D32f *point,
CvSize imageSize) {
  // Si nos alejamos mas de un pixel del borde de la imagen retornamos
un false
  // indicando que hay un misstracking
  if ((point->x >= imageSize.width) || (point->x <= -1.0) ||
      (point->y >= imageSize.height) || (point->y <= -1.0))
    return false;
  // Clamp en x
  if (point->x > (imageSize.width - 1)) {
    point->x = (float)imageSize.width - 1.5;
  } else if (point->x < 0) {
    point->x = 0.5;
  }
  // Clamp en y
```

```
  if (point->y > (imageSize.height - 1)) {
    point->y = (float)imageSize.height - 1.5;
  } else if (point->y < 0) {
    point->y = 0.5;
  }

  return true;
}
/*-------------------------------------------------------------------
--------------------*/
void OpticalFlowEstimator::firstOpticalFlow (const IplImage
*prevImage, const IplImage *currImage, OpticalVectorsList
*opticalVectors) {
  // Convertimos las imagenes a escala de grises
  cvCvtColor(prevImage, _prevGray, CV_BGR2GRAY);
  cvCvtColor(currImage, _currGray, CV_BGR2GRAY);
  // Se crea la lista de puntos para el tracking
  opticalVectors->count = sparseFeaturesInQuadrants(_prevGray,
opticalVectors->prevPoints, 0, hd, vd, featuresPerQuadrant);
  // Se calcula el flujo optico
  cvCalcOpticalFlowPyrLK(_prevGray, _currGray, _prevPyramid,
_currPyramid,
                         opticalVectors->prevPoints, opticalVectors-
>currPoints,
                         opticalVectors->count,
cvSize(win_size,win_size), 3, _lkStatus, 0,

cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), 0);
  // Eliminamos los puntos que no tengan correspondencia
  int j = 0;
  for (int i = 0; i < opticalVectors->count; i++)
    if (_lkStatus[i])
      if (checkPointAndFix(&(opticalVectors->currPoints[i]),
cvSize(currImage->width, currImage->height))) {
        opticalVectors->prevPoints[j] = opticalVectors->prevPoints[i];
        opticalVectors->currPoints[j] = opticalVectors->currPoints[i];
        j++;
      }
  opticalVectors->count = j;
}

/*-------------------------------------------------------------------
--------------------*/
void OpticalFlowEstimator::getOpticalFlow (const IplImage *currImage,
```

```
OpticalVectorsList *opticalVectors) {
  // Los puntos e imagenes actuales pasan a ser los previos
  CV_SWAP(opticalVectors->currPoints, opticalVectors->prevPoints,
_swapPoints);
  CV_SWAP(_currGray, _prevGray, _swapTemp);
  CV_SWAP(_currPyramid, _prevPyramid, _swapTemp);

  // Convertimos la nueva imagen a escala de grises
  cvCvtColor(currImage, _currGray, CV_BGR2GRAY);

  // Revisamos la distribución de puntos en cuandrantes y hacemos las
modificaciones oportunas
  opticalVectors->count = sparseFeaturesInQuadrants(_prevGray,
opticalVectors->prevPoints, opticalVectors->count, hd, vd,
featuresPerQuadrant);

  // Se calcula el flujo optico
  cvCalcOpticalFlowPyrLK(_prevGray, _currGray, _prevPyramid,
_currPyramid, opticalVectors->prevPoints, opticalVectors->currPoints,
opticalVectors->count, cvSize(win_size,win_size), 3, _lkStatus, 0,
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,1.0),
CV_LKFLOW_PYR_A_READY);

  // Eliminamos los puntos que no tengan correspondencia
  int j = 0;
  for (int i = 0; i < opticalVectors->count; i++)
    if (_lkStatus[i])
      if (checkPointAndFix(&(opticalVectors->currPoints[i]),
cvSize(currImage->width, currImage->height))) {
        opticalVectors->prevPoints[j] = opticalVectors->prevPoints[i];
        opticalVectors->currPoints[j] = opticalVectors->currPoints[i];
        j++;
      }
  opticalVectors->count = j;

  if (logEnabled)
    addToLog(opticalVectors);
}

/*-------------------------------------------------------------------
--------------------*/
bool OpticalFlowEstimator::setBaseTimeStamp(const tbb::tick_count
&timestamp) {
  if (logEnabled) {
```

```
    return false;
  } else {
    baseTimestamp = timestamp;
    return true;
  }
}
/*---------------------------------------------------------------------
---------------------*/
bool OpticalFlowEstimator::resetBaseTimeStamp() {
  if (logEnabled) {
    return false;
  } else {
    baseTimestamp = tbb::tick_count::now();
    return true;
  }
}
/*---------------------------------------------------------------------
---------------------*/
bool OpticalFlowEstimator::createLog(const char *fileName) {
  TOFFileHeader fileHeader;

  if (logCreated)
    return false;

  if (logFile.open(fileName, FXIO::Writing)) {
    // Escribir la cabecera
    fileHeader.hd = hd;
    fileHeader.vd = vd;
    fileHeader.featuresPerQuadrant = featuresPerQuadrant;
    memset(fileHeader.padding, 0, 128 - (3 * sizeof(int)));
    logFile.writeBlock(&fileHeader, sizeof(TOFFileHeader));
    fileBufferSize = 0;
    fileBufferSize = (sizeof(double) + sizeof(unsigned int) +
                      sizeof(CvPoint2D32f) * 2 * vd * hd *
featuresPerQuadrant) *
                     RECORDS_IN_OF_FILE_BUFFER;
    fileBuffer = new char[fileBufferSize];
    fileBufferCounter = 0;
    resetBaseTimeStamp();
    logCreated = true;
    return true;
```

```
  } else {
    return false;
  }
}

/*-------------------------------------------------------------------
--------------------*/
bool OpticalFlowEstimator::enableLog() {
  if (!logCreated || logEnabled)
    return false;

  fileBufferCounter = 0;
  logEnabled = true;
  return true;
}

/*-------------------------------------------------------------------
--------------------*/
bool OpticalFlowEstimator::disableLog() {
  tbb::mutex::scoped_lock(logMutex);

  if (!logEnabled)
    return false;

  logEnabled = false;
  // Si hay cosas pendientes por escribir...
  if (fileBufferCounter > 0) {
    logFile.writeBlock(fileBuffer, fileBufferCounter);
    fileBufferCounter = 0;
  }
  return true;
}

/*-------------------------------------------------------------------
--------------------*/
bool OpticalFlowEstimator::closeLog() {
  if (!logCreated)
    return false;

  disableLog();
  logCreated = false;
```

```
  logFile.close();
  delete fileBuffer;
  return true;
}

/*-------------------------------------------------------------------
--------------------*/
bool OpticalFlowEstimator::addToLog(const OpticalVectorsList *ovList)
{
  //TOFFileRecord fileBufferRecord;

  tbb::mutex::scoped_lock(logMutex);

  if (!logEnabled) return false;

  tbb::tick_count timestamp = tbb::tick_count::now();
  tbb::tick_count::interval_t interval = timestamp - baseTimestamp;

  int bleble = sizeof(CvPoint2D32f);
  double time = interval.seconds();
  int featuresPerImage = vd * hd * featuresPerQuadrant;
  // instante de la muestra
  memcpy(fileBuffer + fileBufferCounter, &time, sizeof(double));
  fileBufferCounter += sizeof(double);
  // número de puntos válidos
  memcpy(fileBuffer + fileBufferCounter, &(ovList->count),
sizeof(unsigned int));
  fileBufferCounter += sizeof(unsigned int);
  // lista de puntos previos
  memcpy(fileBuffer + fileBufferCounter, ovList->prevPoints,
sizeof(CvPoint2D32f) * featuresPerImage);
  fileBufferCounter += (sizeof(CvPoint2D32f) * featuresPerImage);
  // lista de puntos actuales
  memcpy(fileBuffer + fileBufferCounter, ovList->currPoints,
sizeof(CvPoint2D32f) * featuresPerImage);
  fileBufferCounter += (sizeof(CvPoint2D32f) * featuresPerImage);
  /* test */
  if (fileBufferCounter == fileBufferSize) {
    int bytesWritten;
    bytesWritten = logFile.writeBlock(fileBuffer, fileBufferSize);
    fileBufferCounter = 0;
  }
```

```
  return true;

}
/*-------------------------------------------------------------------
---------------------*/
bool OpticalFlowEstimator::setBaseTimestamp(tbb::tick_count
baseTimestamp) {
  this->baseTimestamp = baseTimestamp;
  return true;
}
```

9.1.9 Fastrak.h

```
#ifndef FASTRAK_H
#define FASTRAK_H

#ifdef LIBFASTRAK_EXPORTS
#define LIBFASTRAK_API __declspec(dllexport)
#else
#define LIBFASTRAK_API __declspec(dllimport)
#endif

#include <boost/asio.hpp>
#include <tbb/tbb_thread.h>
#include <tbb/mutex.h>
#include <fx.h>
#include <fx3d.h>
#include "triplebuffer.hpp"

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define P_PACKET_SIZE        31
#define H_PACKET_SIZE        26

#define CUBE_SIZE       3 * 8


#define N_FSTRK_STATIONS                        4
#define P_H_INTERLEAVING                        5
```

```
#define MAX_GARBAGE_BUFF_SIZE       512

#define RECORDS_IN_FILE_BUFFER  120

typedef enum {
  FSTRK_OK                  = 0,
  FSTRK_GENERIC_ERROR       = -1,
  FSTRK_NOT_INITIALIZED     = -2,
  FSTRK_INVALID_OPERATION   = -3,
  FSTRK_INVALID_PACKET      = -4,
  FSTRK_HEMISPHERE_ERROR    = -5,
  FSTRK_FILE_ERROR          = -6,
} FstrkResult;

typedef struct {
  FXVec3f   p[N_FSTRK_STATIONS];
      FXQuatf           q[N_FSTRK_STATIONS];
      int retVal;
} TbuffData;

typedef struct {
  unsigned int activeStations;
  unsigned int padding[31];
} TFstrkFileHeader;

typedef struct {
      FXVec3f  pos;
      FXVec3f  realPos;
      FXQuatf  q;
} TCorrectionMeshPoint;

class LIBFASTRAK_API Fastrak {
            boost::asio::io_service *ioService;
            boost::asio::serial_port *serialPort;
    boost::asio::deadline_timer *asyncTimer;
            tbb::tbb_thread *thFastrakIO;
            TripleBuffer<TbuffData> *buffer;

            int stop;
            int xSize, ySize, zSize; // tamaño de la malla de
correccion
```

```
            TCorrectionMeshPoint *correctionMesh;
            int minX, maxX, minY, maxY, minZ, maxZ; // indices del cubo
            int correctionCubePoints[CUBE_SIZE];
            float correctionCube[CUBE_SIZE];
    bool readingError;
            FXVec3f lastPos[N_FSTRK_STATIONS];
            FXQuatf lastQ[N_FSTRK_STATIONS];

            int activeStations;
    int currStation;
    int checkHemisphere;

    char readBuff[P_PACKET_SIZE * 4];
            char readGarbageBuff[MAX_GARBAGE_BUFF_SIZE];

    FXFile captureFile;
    char *fileBuffer;
    unsigned int fileBufferSize;
    unsigned int fileBufferCounter;
    tbb::tick_count baseTimestamp;

    bool IOisRunning;
    bool recoveringState;
    bool deviceInitialized;
    bool logEnabled;
    bool logCreated;
    tbb::mutex logMutex;
    void fastrakIO();
    FstrkResult checkPPacket(const char* packet, int activeStations);
    FstrkResult checkHPacket(const char* packet, int nStation);
    FstrkResult processPPacket(const char* packet, int
activeStations);
    FstrkResult processHPacket(const char* packet, int currStation);
    void recoverHandler();
    void recoverWriteHandler(const boost::system::error_code& error);
    void recoverReadHandler(const boost::system::error_code& error,
std::size_t bytesTransferred);
    void dummyHandler(const boost::system::error_code& error,
std::size_t bytesTransferred);
    void processHAndRequestPHandler(const boost::system::error_code&
error,
                                    std::size_t bytesTransferred);
```

```
    void processPAndRequestHHandler(const boost::system::error_code&
error,
                                    std::size_t bytesTransferred);
    void processPAndRequestPHandler(const boost::system::error_code&
error,
                                    std::size_t bytesTransferred);
    void readHandler(const boost::system::error_code& error,
std::size_t bytesTransferred);
    void startFastrakCommunications();


    int buscaCubo(float x, float y, float z);
            int intersectaTriangulo(const FXVec3f &p, const FXVec3f &d,
                            const int *vr0, const int *vr1, const int
*vr2,
                            FXVec3f *intval, FXQuatf *qval, float *t);
            int intersectaSuperficie(const FXVec3f &point, const
FXVec3f &ray,
                             const int *v1, const int *v2, const int
*v3, const int *v4,
                             FXVec3f *intval, FXQuatf *qval, float
*t);
            int intersectaCubo(const FXVec3f &p, const FXVec3f &d,
                        const int *vertices,
                        FXVec3f *intval, FXQuatf *qval);
    int correctPoint(const FXVec3f &v, const FXQuatf &q, FXVec3f *cv,
FXQuatf *cq);


    FstrkResult addToLog(const char *data);


      public:
    int readingsCounter;


            int init(char *portName, int stations);
            FstrkResult run();
            int loadCorrectionMesh(char *filename, char t);
            int setMesh(TCorrectionMeshPoint *m, int xs, int ys, int
zs);
            int readRawCoords(FXVec3f *pos, FXQuatf *q);
            int readCoords(FXVec3f *cpos, FXQuatf *cq);
            int readCoords(FXVec3f *cpos, FXQuatf *cq, float *cube);
            int readCoords(FXVec3f *cpos, FXQuatf *cq, FXVec3f *rawpos,
FXQuatf *rawq);
            int readCoords(FXVec3f *cpos, FXQuatf *cq, FXVec3f *rawpos,
FXQuatf *rawq, float *cube);
```

```
    FstrkResult resetBaseTimeStamp();
    FstrkResult setBaseTimestamp(const tbb::tick_count &timestamp);
    FstrkResult createLog(const char *fileNameToLog);
    FstrkResult enableLog();
    FstrkResult disableLog();
    FstrkResult closeLog();
            void destroy();
};

class LIBFASTRAK_API coordsToModelMatrix {
private:
      FXVec3f origin;
public:
      coordsToModelMatrix(const FXVec3f &fastrakOrigin);
      void getMatrixFromRaw(const FXVec3f &pos, const FXQuatf &q,
float *m);
      void getMatrix(const FXVec3f &pos, const FXQuatf &q, float *m);
};

class LIBFASTRAK_API coordsToViewMatrix {
private:
      FXVec3f origin;
public:
      coordsToViewMatrix(const FXVec3f &fastrakOrigin);
      void getMatrixFromRaw(const FXVec3f &pos, const FXQuatf &q,
float *m);
      void getMatrix(const FXVec3f &pos, const FXQuatf &q, float *m);
};

#endif
```

9.1.10 Fastrak.cpp

```
// fastrak.cpp: define las funciones exportadas de la aplicación DLL.
#include "fastrak.h"
#include "windows.h"
#include <stdio.h>
#include <stdlib.h>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <GL/gl.h>
```

```
#define MAX_GARBAGE_SIZE 512;
static const char hemisphereInitCommand[4][30] = {
              "H1,0,0,-1\nH1,0,0,0\n",
              "H2,0,0,-1\nH2,0,0,0\n",
              "H3,0,0,-1\nH3,0,0,0\n",
              "H4,0,0,-1\nH4,0,0,0\n"};
static const char fastrakInitCommand[4][30] = {
              "R1\nO1,2,11\nuf",
              "R2\nO2,2,11\nuf",
              "R3\nO3,2,11\nuf",
              "R4\nO4,2,11\nuf"};
static const char hemisphereCommand[4][5] = {"H1\n", "H2\n", "H3\n",
"H4\n"};


static const TCorrectionMeshPoint CP0 = {FXVec3f(.0, .0, .0),
FXVec3f(.0, .0, .0), FXQuatf(.0, .0, .0, .0)};


#define CPOINT(a,b,c)   ((correctionMesh != NULL) ? correctionMesh[a +
(b * xSize) + (c * xSize *  ySize)] : CP0)


/*-----------------------------------------------------------------
------*/
int Fastrak::init(char *portName, int stations) {
  boost::system::error_code ec;

      activeStations = stations;
      correctionMesh = NULL;

      ioService = new boost::asio::io_service;
      // Open serial port (port options 115000,8,n,1)
      serialPort = new boost::asio::serial_port(*ioService, portName);
            // 115200
      boost::asio::serial_port_base::baud_rate baudRate(115200);
      serialPort->set_option(baudRate, ec);
  if (ec)
    printf ("Error: %s\n", ec.message().c_str());
            // 8
      boost::asio::serial_port_base::character_size charSize(8);
      serialPort->set_option(charSize, ec);
  if (ec)
    printf ("Error: %s\n", ec.message().c_str());
            // n
```

```
      boost::asio::serial_port_base::parity
_parity(boost::asio::serial_port_base::parity::none);
      serialPort->set_option(_parity, ec);
  if (ec)
    printf ("Error: %s\n", ec.message().c_str());
           // 1
      boost::asio::serial_port_base::stop_bits
stopBits(boost::asio::serial_port_base::stop_bits::one);
      serialPort->set_option(stopBits, ec);
  if (ec)
    printf ("Error: %s\n", ec.message().c_str());
           // no flow control
      boost::asio::serial_port_base::flow_control
flowControl(boost::asio::serial_port_base::flow_control::none);
      serialPort->set_option(flowControl, ec);
  if (ec)
    printf ("Error: %s\n", ec.message().c_str());


  // Timero para las operaciones de IO
  asyncTimer = new boost::asio::deadline_timer(*ioService);


  // Vaciamos los buffers de lectura para asegurarnos de que no hay
basura
//  syncPort();


      // Fastrak output configuration
      for (int i = 0; i < stations; i++) {
           Sleep(200);
    boost::asio::write(*serialPort,
                       boost::asio::buffer(hemisphereInitCommand[i]),
                       boost::asio::transfer_all(), ec);
    if (ec)
      printf ("Error inicializando hemisferio de sonda %d : %s\n", i,
ec.message().c_str());
           Sleep(200);
    boost::asio::write(*serialPort,
                       boost::asio::buffer(fastrakInitCommand[i]),
                       boost::asio::transfer_all(), ec);
    if (ec)
      printf ("Error inicializando lectura de sonda %d : %s\n", i,
ec.message().c_str());
      }
```

```
      buffer = new TripleBuffer<TbuffData>(1);
  currStation = 0;
  IOisRunning = false;
  deviceInitialized = true;
  logEnabled = false;
  logCreated = false;

  readingsCounter = 0;

      return 0;
}
/*-------------------------------------------------------------------
------*/
void Fastrak::destroy(void) {
      stop = 1;

  ioService->stop();
  while (ioService->poll_one() != 0)
    tbb::this_tbb_thread::yield();
      if (serialPort->is_open())
            serialPort->close();
  delete buffer;
      delete serialPort;
  delete asyncTimer;
      delete ioService;
  delete thFastrakIO;
  if (logCreated)
    closeLog();
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::createLog(const char *logFileName) {
  TFstrkFileHeader fileHeader;

  if (!deviceInitialized)   // Si el dispositivo no esta inicializado
devuelve error
    return FSTRK_NOT_INITIALIZED;
  if (logEnabled)           // Si el dispositivo ya logueando se
retorna error
    return FSTRK_INVALID_OPERATION;
```

```
  if (captureFile.open(logFileName, FXIO::Writing)) {
    fileBufferSize = (sizeof(double) + 7 * sizeof(float) *
activeStations) * RECORDS_IN_FILE_BUFFER;
    fileBuffer = new char[fileBufferSize];
    fileBufferCounter = 0;
    fileHeader.activeStations = activeStations;
    memset(fileHeader.padding, 0, sizeof(TFstrkFileHeader) - 4);
    captureFile.writeBlock(&fileHeader, sizeof(TFstrkFileHeader));
    resetBaseTimeStamp();
    logCreated = true;
    return FSTRK_OK;
  } else {
    return FSTRK_FILE_ERROR;
  }
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::enableLog() {
  if ((!logCreated) || logEnabled)
    return FSTRK_INVALID_OPERATION;

  fileBufferCounter = 0;
  logEnabled = true;
  return FSTRK_OK;
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::disableLog() {
  tbb::mutex::scoped_lock(logMutex);

  if (!logEnabled)           // Si el dispositivo no estaba logueando
devuelve error
    return FSTRK_INVALID_OPERATION;

  logEnabled = false;
  // Si hay cosas pendientes por escribir...
  if (fileBufferCounter > 0) {
    captureFile.writeBlock(fileBuffer, fileBufferCounter);
    fileBufferCounter = 0;
  }
  return FSTRK_OK;
}
```

```
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::closeLog() {
  if (!logCreated)
    return FSTRK_INVALID_OPERATION;

  disableLog();

  if (captureFile.close()) {
    fileBufferCounter = 0;
    delete fileBuffer;
    return FSTRK_OK;
  } else {
    return FSTRK_FILE_ERROR;
  }
}

/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::addToLog(const char *data) {
  tbb::mutex::scoped_lock(logMutex);

  if (!logEnabled)           // Si el dispositivo no estaba logueando
devuelve error
    return FSTRK_INVALID_OPERATION;

  readingsCounter++;

  tbb::tick_count timestamp = tbb::tick_count::now();
  tbb::tick_count::interval_t interval = timestamp - baseTimestamp;

  // Timestamp
  *((double *)(fileBuffer + fileBufferCounter)) = interval.seconds();
  fileBufferCounter += sizeof(double);
  // Posición y orientacion P1,Q1,P2,Q2...
  for (int k = 0; k < activeStations; k++) {
    //const float *ptr = (float *)(data + k * P_PACKET_SIZE + 3);
    memcpy(fileBuffer + fileBufferCounter, (data + 3 + k *
P_PACKET_SIZE), 7 * sizeof(float));
    fileBufferCounter += 7 * sizeof(float);
  }
  if (fileBufferCounter == fileBufferSize) {
```

```
    captureFile.writeBlock(fileBuffer, fileBufferSize);
    fileBufferCounter = 0;
  }
  return FSTRK_OK;
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::setBaseTimestamp(const tbb::tick_count
&timestamp) {
  if (logEnabled) {
    return FSTRK_INVALID_OPERATION;
  } else {
    baseTimestamp = timestamp;
    return FSTRK_OK;
  }
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::resetBaseTimeStamp() {
  if (logEnabled) {
    return FSTRK_INVALID_OPERATION;
  } else {
    baseTimestamp = tbb::tick_count::now();
    return FSTRK_OK;
  }
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::run() {
      // Primero dejamos el puerto limpito y además ponemos un par de
comandos
      // en cola para que el ioService no termine nada más arrancarlo
  recoveringState = true;
  ioService->post(boost::bind(&Fastrak::recoverHandler, this));

      thFastrakIO = NULL;
  thFastrakIO = new tbb::tbb_thread(boost::bind(&Fastrak::fastrakIO,
this));
  if (thFastrakIO == NULL)
      printf ("No se pudo crear el Thread de IO con Fastrak!!!");
```

```
  return FSTRK_OK;
}
/*-------------------------------------------------------------------
------*/
void Fastrak::fastrakIO() {
  if (IOisRunning == false) {
    IOisRunning = true;
    ioService->run();
    printf("IOService terminado\n");
    IOisRunning = false;
  }
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::checkPPacket(const char* packet, int
activeStations) {
  int i;
  FstrkResult errorCond;

  i = 0;
  errorCond = FSTRK_OK;
  while ((i < activeStations) && (errorCond != FSTRK_INVALID_PACKET))
    if ((packet[i * P_PACKET_SIZE + 0] == '0') &&
        (packet[i * P_PACKET_SIZE + 1] == ('0' + i + 1)) &&
        (packet[i * P_PACKET_SIZE + 2] == ' '))
      i++;
    else
      errorCond = FSTRK_INVALID_PACKET;
  return errorCond;
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::checkHPacket(const char* packet, int nStation) {
  if ((packet[0] == '2') && (packet[1] == ('1' + nStation)) &&
      (packet[2] == 'H') && (packet[3] == ' '))
    return FSTRK_OK;
  else
    return FSTRK_INVALID_PACKET;
}
/*-------------------------------------------------------------------
------*/
```

```
FstrkResult Fastrak::processPPacket(const char* packet, int
activeStations) {
      TbuffData *data;


  if (checkPPacket(packet, activeStations) == FSTRK_OK) {
    if (logEnabled)
      addToLog(packet);
        data = (TbuffData *)buffer->getWriteBuffer();
    if (data != NULL) {
      for (int k = 0; k < activeStations; k++) {
        const float *ptr = (float *)(packet + k * P_PACKET_SIZE + 3);
        data->p[k].set(*(ptr), *(ptr + 1), *(ptr + 2));
        data->q[k].set(*(ptr + 4), *(ptr + 5), *(ptr + 6), *(ptr +
3));
                data->retVal = FSTRK_OK;
      }
      return FSTRK_OK;
    } else
      return FSTRK_GENERIC_ERROR;
  } else {
    return FSTRK_INVALID_PACKET;
  }
}
/*-------------------------------------------------------------------
------*/
FstrkResult Fastrak::processHPacket(const char* packet, int
currStation) {
      TbuffData *data;
  float hz, angle;
  char localBuff[7];


  if (checkHPacket(packet, currStation) == FSTRK_OK) {
            localBuff[6] = '\0';
    strncpy(localBuff, packet + 18, 6);
            hz = atof(localBuff);
        angle = 360.0 * asin(hz)/(M_PI * 2.0);
        if ((hz > 0.0) && ((fabs(angle) > 15.0))) {
      printf("Inversión de hemisferio!!\n");
      boost::asio::async_write(*serialPort,
boost::asio::buffer("H1,0,0,-1\nH1,0,0,0\n", 10),
                               boost::asio::transfer_all(),
```

```
                                boost::bind(&Fastrak::dummyHandler,
this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
      return FSTRK_HEMISPHERE_ERROR;
    } else {
      return FSTRK_OK;
    }
  } else {
    return FSTRK_INVALID_PACKET;
      }
}
/*-------------------------------------------------------------------
------*/
void Fastrak::recoverReadHandler(const boost::system::error_code&
error,
                                 std::size_t bytesTransferred) {
  if (!error) {
    if (bytesTransferred == MAX_GARBAGE_BUFF_SIZE) {
      printf ("Recuperando, leídos %d bytes\n", bytesTransferred);
      serialPort->async_read_some(boost::asio::buffer(readGarbageBuff,
MAX_GARBAGE_BUFF_SIZE),

boost::bind(&Fastrak::recoverReadHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
            } else {
      recoveringState = false;
      readBuff[bytesTransferred] = '\0';
      printf ("Comunicaciones recuperadas, descartados %d bytes\n",
bytesTransferred);
      startFastrakCommunications();
    }
  } else {
    printf ("%s\n", error.message().c_str());
  }

}
/*-------------------------------------------------------------------
------*/
```

```
void Fastrak::recoverWriteHandler(const boost::system::error_code&
error) {
  if (!error) {
    boost::asio::async_write(*serialPort, boost::asio::buffer("*\n",
2),
                             boost::asio::transfer_all(),
                             boost::bind(&Fastrak::recoverReadHandler,
this,

boost::asio::placeholders::error,
                                         MAX_GARBAGE_BUFF_SIZE));
    printf ("Comenzando recuperación de comunicaciones\n");
  } else {
    printf ("%s\n", error.message().c_str());
  }
}
/*-------------------------------------------------------------------
------*/
void Fastrak::recoverHandler() {
  asyncTimer->cancel();

  asyncTimer->expires_from_now(boost::posix_time::milliseconds(50));
  asyncTimer->async_wait(boost::bind(&Fastrak::recoverWriteHandler,
this,

boost::asio::placeholders::error));
}
/*-------------------------------------------------------------------
------*/
void Fastrak::dummyHandler(const boost::system::error_code& error,
                           std::size_t bytesTransferred) {
  return;
}
/*-------------------------------------------------------------------
------*/
void Fastrak::processHAndRequestPHandler(const
boost::system::error_code& error,
                                         std::size_t bytesTransferred)
{
  if (recoveringState)
    return;

  if (!error) {
    if ((bytesTransferred != H_PACKET_SIZE) ||
```

```
        (processHPacket(readBuff, currStation) ==
FSTRK_INVALID_PACKET)) {
      printf ("Error: invalid H packet!!\n");
      recoveringState = true;
      ioService->post(boost::bind(&Fastrak::recoverHandler, this));
    } else {
                currStation = (currStation + 1) % activeStations;
      // printf ("Escribiendo comando P\n");
      boost::asio::async_write(*serialPort, boost::asio::buffer("P",
1),
                               boost::asio::transfer_all(),
                               boost::bind(&Fastrak::readHandler,
this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
    }
  } else {
    printf ("Error: %s\n", error.message().c_str());
  }
}
/*-------------------------------------------------------------------
------*/
void Fastrak::processPAndRequestHHandler(const
boost::system::error_code& error,
                                         std::size_t bytesTransferred)
{


  if (recoveringState)
    return;


  if (!error) {
    if ((bytesTransferred != (P_PACKET_SIZE * activeStations)) ||
        (processPPacket(readBuff, activeStations) ==
FSTRK_INVALID_PACKET)) {
      printf ("Error: invalid P packet!!\n");
      recoveringState = true;
      ioService->post(boost::bind(&Fastrak::recoverHandler, this));
    } else {
             // printf ("Escribiendo comando H\n");
             boost::asio::async_write(*serialPort,
boost::asio::buffer(hemisphereCommand[currStation], 3),
```

```
                    boost::asio::transfer_all(),

                    boost::bind(&Fastrak::readHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
    }
  } else {
    printf ("Error: %s\n", error.message().c_str());
  }
}
/*-------------------------------------------------------------------
------*/
void Fastrak::processPAndRequestPHandler(const
boost::system::error_code& error, std::size_t bytesTransferred) {

  if (recoveringState)
    return;

  if (!error) {
    if ((bytesTransferred != (P_PACKET_SIZE * activeStations)) ||
        (processPPacket(readBuff, activeStations) ==
FSTRK_INVALID_PACKET)) {
      printf ("Error: invalid P packet!!\n");
      recoveringState = true;
      ioService->post(boost::bind(&Fastrak::recoverHandler, this));
    } else {
      // printf ("Escribiendo comando P\n");
      boost::asio::async_write(*serialPort, boost::asio::buffer("P",
1),
                               boost::asio::transfer_all(),
                               boost::bind(&Fastrak::readHandler,
this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
    }
  } else {
    printf ("Error: %s\n", error.message().c_str());
  }
```

```
}
/*-------------------------------------------------------------------
------*/
void Fastrak::readHandler(const boost::system::error_code& error,
                          std::size_t bytesTransferred) {

  if (recoveringState)
    return;

  if (!error) {
    // printf ("Esperando para leer la respuesta de P\n");
    switch (checkHemisphere) {
      case 0:
        boost::asio::async_read(*serialPort,
                                boost::asio::buffer(readBuff,
P_PACKET_SIZE * activeStations),

boost::asio::transfer_all(),

boost::bind(&Fastrak::processPAndRequestHHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
                        break;
      case 1:
        boost::asio::async_read(*serialPort,
                                boost::asio::buffer(readBuff,
H_PACKET_SIZE),

boost::asio::transfer_all(),

boost::bind(&Fastrak::processHAndRequestPHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
                        break;
      default:
        boost::asio::async_read(*serialPort,
                                boost::asio::buffer(readBuff,
P_PACKET_SIZE * activeStations),

boost::asio::transfer_all(),
```

```
boost::bind(&Fastrak::processPAndRequestPHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
                        break;
    }
    checkHemisphere = (checkHemisphere + 1) % P_H_INTERLEAVING;
  } else {
    printf ("Error: %s\n", error.message().c_str());
  }
}
/*-----------------------------------------------------------------
------*/
void Fastrak::startFastrakCommunications() {
  // Comenzamos el pingo-pong de comandos con el Fastrak;
  // printf ("Escribiendo comando P\n");
  checkHemisphere = 0;
  boost::asio::async_write(*serialPort, boost::asio::buffer("P", 1),
                           boost::asio::transfer_all(),
                           boost::bind(&Fastrak::readHandler, this,

boost::asio::placeholders::error,

boost::asio::placeholders::bytes_transferred));
}

/*-----------------------------------------------------------------
------*/
int Fastrak::readRawCoords(FXVec3f *pos, FXQuatf *q) {
      TbuffData *tmpBuff;

      tmpBuff = (TbuffData *)buffer->getReadBuffer();
      if (tmpBuff != NULL) {
            for (int i = 0; i < activeStations; i++) {
                  pos[i] = tmpBuff->p[i];
                  q[i] = tmpBuff->q[i];
                  lastPos[i] = pos[i];
                  lastQ[i] = q[i];
            }
            return tmpBuff->retVal;
      }     else {
```

```
            for (int i = 0; i < activeStations; i++) {
                  pos[i] = lastPos[i];
                  q[i] = lastQ[i];
            }
            return 0;
      }
}
/*-------------------------------------------------------------------
------*/
int Fastrak::readCoords(FXVec3f *cpos, FXQuatf *cq) {

      TbuffData *tmpBuff;

      tmpBuff = (TbuffData *)buffer->getReadBuffer();
      if (tmpBuff != NULL) {
            for (int i = 0; i < activeStations; i++) {
                  if (correctPoint(tmpBuff->p[i], tmpBuff->q[i],
&cpos[i], &cq[i])) {
                        lastPos[i] = cpos[i];
                        lastQ[i] = cq[i];
                  }
            }
            return tmpBuff->retVal;
      }     else {
            for (int i = 0; i < activeStations; i++) {
                  cpos[i] = lastPos[i];
                  cq[i] = lastQ[i];
            }
            return 0;
      }
}
/*-------------------------------------------------------------------
------*/
int Fastrak::readCoords(FXVec3f *cpos, FXQuatf *cq, float *cube) {

            return 0;
}
/*-------------------------------------------------------------------
------*/
int Fastrak::readCoords(FXVec3f *cpos, FXQuatf *cq, FXVec3f *rawpos,
FXQuatf *rawq) {
```

```
            return 0;
}
/*-------------------------------------------------------------------
------*/
int Fastrak::readCoords(FXVec3f *cpos, FXQuatf *cq, FXVec3f *rawpos,
FXQuatf *rawq, float *cube) {

      TbuffData *tmpBuff;

      tmpBuff = (TbuffData *)buffer->getReadBuffer();
      if (tmpBuff != NULL) {
            for (int i = 0; i < activeStations; i++) {
                  rawpos[i] = tmpBuff->p[i];
                  rawq[i] = tmpBuff->q[i];
                  if (correctPoint(rawpos[i], rawq[i], &cpos[i],
&cq[i])) {
                        lastPos[i] = cpos[i];
                        lastQ[i] = cq[i];
                  }
                  for (int i = 0; i < 3*8; i++) {
                        cube[i] = correctionCube[i];
                  }
            }
            return tmpBuff->retVal;
      }     else {
            for (int i = 0; i < activeStations; i++) {
                  cpos[i] = lastPos[i];
                  cq[i] = lastQ[i];
            }
            return 0;
      }
}


/*-------------------------------------------------------------------
------*/
int Fastrak::buscaCubo(float x, float y, float z) {

      // busqueda del cubo contenedor
      // busqueda en el eje X de fastrak
```

```
      while ((minY > 0) && ((CPOINT(maxX,minY,minZ).pos.x < x) &&

      (CPOINT(maxX,minY,maxZ).pos.x < x) &&

      (CPOINT(maxX,maxY,minZ).pos.x < x) &&

      (CPOINT(maxX,maxY,maxZ).pos.x < x) )) {
            --maxY;
            --minY;
      }
      while ((maxY < (ySize - 1)) && ((CPOINT(minX,minY,minZ).pos.x >
x) &&

                                    (CPOINT(minX,minY,maxZ).pos.x > x)
&&

                                    (CPOINT(minX,maxY,minZ).pos.x > x)
&&

                                    (CPOINT(minX,maxY,maxZ).pos.x > x)
)) {
            ++maxY;
            ++minY;
      }
      // busqueda en el eje Y de fastrak
      while ((maxX < (xSize - 1)) && ((CPOINT(maxX,maxY,minZ).pos.y <
y) &&

                                    (CPOINT(maxX,maxY,maxZ).pos.y < y)
&&

                                    (CPOINT(minX,maxY,minZ).pos.y < y)
&&

                                    (CPOINT(minX,maxY,maxZ).pos.y < y)
)) {
            ++maxX;
            ++minX;
      }
      while ((minX > 0) && ((CPOINT(maxX,minY,minZ).pos.y > y) &&

      (CPOINT(maxX,minY,maxZ).pos.y > y) &&

      (CPOINT(minX,minY,minZ).pos.y > y) &&

      (CPOINT(minX,minY,maxZ).pos.y > y) )) {
            --maxX;
```

```
            --minX;
      }
      // busqueda en el eje Z
      while ((maxZ < (zSize - 1)) && ((CPOINT(maxX,maxY,maxZ).pos.z <
z) &&

                                    (CPOINT(maxX,minY,maxZ).pos.z < z)
&&

                                    (CPOINT(minX,maxY,maxZ).pos.z < z)
&&

                                    (CPOINT(minX,minY,maxZ).pos.z < z)
)) {
            ++maxZ;
            ++minZ;
      }
      while ((minZ > 0) && ((CPOINT(maxX,maxY,minZ).pos.z > z) &&

      (CPOINT(maxX,minY,minZ).pos.z > z) &&

      (CPOINT(minX,maxY,minZ).pos.z > z) &&

      (CPOINT(minX,minY,minZ).pos.z > z) )) {
            --maxZ;
            --minZ;
      }
      correctionCubePoints[0] = minX; correctionCubePoints[1] = minY;
correctionCubePoints[2] = minZ;
      correctionCubePoints[3] = maxX; correctionCubePoints[4] = minY;
correctionCubePoints[5] = minZ;
      correctionCubePoints[6] = minX; correctionCubePoints[7] = maxY;
correctionCubePoints[8] = minZ;
      correctionCubePoints[9] = maxX; correctionCubePoints[10] = maxY;
correctionCubePoints[11] = minZ;
      correctionCubePoints[12] = minX; correctionCubePoints[13] =
minY; correctionCubePoints[14] = maxZ;
      correctionCubePoints[15] = maxX; correctionCubePoints[16] =
minY; correctionCubePoints[17] = maxZ;
      correctionCubePoints[18] = minX; correctionCubePoints[19] =
maxY; correctionCubePoints[20] = maxZ;
      correctionCubePoints[21] = maxX; correctionCubePoints[22] =
maxY; correctionCubePoints[23] = maxZ;
      //printf("%d %d %d %d %d %d\n",minX,maxX,minY,maxY,minZ,maxZ);
      return 0;
}
```

```
/*-----------------------------------------------------------------
------*/
int Fastrak::correctPoint(const FXVec3f &v, const FXQuatf &q, FXVec3f
*cv, FXQuatf *cq) {

      FXVec3f final;
      FXVec3f point, ray;
      int nPoints = 0, i;
      int vertices[3*8];
      float distancias[8];
      float dQ0, dQ1, dQ2, dQ3;
      float tQ0, tQ1, tQ2, tQ3;
      float rQ0 = 0.0, rQ1 = -0.707, rQ2 = 0.0, rQ3 = -0.707;
      FXQuatf real, measured, raw, correct, aux, accq, aux2;

      if (correctionMesh == NULL)
            return 0;

      buscaCubo(v.x, v.y, v.z);
      accq.set(.0, .0, .0, .0);

      for (i = 0; i < 8; i++) {
    point = CPOINT(correctionCubePoints[i*3],
correctionCubePoints[i*3+1], correctionCubePoints[i*3+2]).pos - v;
            distancias[i] = point.x * point.x + point.y * point.y +
point.z * point.z;
      }

  float min = 100000000.0;
      for (int j = 0; j < 8; j++) {
            if (distancias[j] < min) {
                  min = distancias[j];
                  i = j;
            }
      }

  real.set(.0, 1.0, .0, .0);
  raw = q;

  aux2 = CPOINT(correctionCubePoints[i*3],
```

```
                correctionCubePoints[i*3+1],
                correctionCubePoints[i*3+2]).q.conj();
  point = v;
  cv->set(.0, .0, .0);
      cq->set(.0, .0, .0, .0);

  // direccion desde el origen
      ray = v;
      if (intersectaCubo(point, ray, correctionCubePoints, &final,
&measured)) {
                  //printf("Po: %f %f %f\n",final.x, final.y, final.z);
            //printf("INTERSECCION\n");
            *cv += final;
            accq += measured;
            ++nPoints;
      }
      // probar otro rayo
  ray.set(0.0, 1.0, 0.0);
      if (intersectaCubo(point, ray, correctionCubePoints, &final,
&measured)) {
            //printf("Py: %f %f %f\n",final.x, final.y, final.z);
            //printf("INTERSECCION\n");
            *cv += final;
        accq += measured;
      ++nPoints;
            //return 1;
      }
      // probar otro rayo
      ray.set(1.0, 0.0, 0.0);
      if (intersectaCubo(point, ray, correctionCubePoints, &final,
&measured)) {
      //printf("Px: %f %f %f\n",final.x, final.y, final.z);
            //printf("INTERSECCION\n");
            *cv += final;
    accq += measured;
    ++nPoints;
            //return 1;
      }
      // probar otro rayo
      ray.set(0.0, 0.0, 1.0);
      if (intersectaCubo(point, ray, correctionCubePoints, &final,
&measured)) {
```

```
      //printf("Pz: %f %f %f\n\n",final.x, final.y, final.z);
            //printf("INTERSECCION\n");
            *cv += final;
            accq += measured;
            ++nPoints;
      }

  if (nPoints == 0)
            return 0;

      *cv /= (float)nPoints;
      accq /= (float)nPoints;

  normalize(accq);
  accq.conj();
  //accq.conj();

  correct = real * accq;
  aux = correct * raw;

      *cq = aux;

  return 1;
}

/*-----------------------------------------------------------------
------*/
int Fastrak::intersectaCubo(const FXVec3f &p, const FXVec3f &d,
                            const int *vertices, FXVec3f *intval,
FXQuatf *qval) {

      FXVec3f points[2], int1, int2;
      FXQuatf quats[2], qtemp;
      int pointsIndex = 0;
      float tvals[2], t1, t2;
      const int *v1, *v2, *v3, *v4;

      // cara menor Z
      v1 = vertices;
      v2 = vertices + 3;
      v3 = vertices + 6;
```

```
      v4 = vertices + 9;
      if (pointsIndex < 2) {
            if (intersectaSuperficie(p, d, v1, v2, v3, v4, &int1,
&qtemp, &t1)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int1;
                  tvals[pointsIndex] = t1;
                  quats[pointsIndex] = qtemp;
                  ++pointsIndex;
            }
      }
      // cara mayor Z
      v1 = vertices + 12;
      v2 = vertices + 15;
      v3 = vertices + 18;
      v4 = vertices + 21;
      if (pointsIndex < 2) {
            if (intersectaSuperficie(p, d, v1, v2, v3, v4, &int2,
&qtemp, &t2)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int2;
                  tvals[pointsIndex] = t2;
                  quats[pointsIndex] = qtemp;
                  ++pointsIndex;
            }
      }
      // cara menor X
      v1 = vertices;
      v2 = vertices + 6;
      v3 = vertices + 12;
      v4 = vertices + 18;
      if (pointsIndex < 2) {
            if (intersectaSuperficie(p, d, v1, v2, v3, v4, &int1,
&qtemp, &t1)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int1;
                  tvals[pointsIndex] = t1;
                  quats[pointsIndex] = qtemp;
                  ++pointsIndex;
            }
      }
```

```
      // cara mayor X
      v1 = vertices + 3;
      v2 = vertices + 9;
      v3 = vertices + 15;
      v4 = vertices + 21;
      if (pointsIndex < 2) {
            if(   intersectaSuperficie(p, d, v1, v2, v3, v4, &int2,
&qtemp, &t2)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int2;
                  tvals[pointsIndex] = t2;
                  quats[pointsIndex] = qtemp;
                  ++pointsIndex;
            }
      }
      // cara menor Y
      v1 = vertices;
      v2 = vertices + 3;
      v3 = vertices + 12;
      v4 = vertices + 15;
      if (pointsIndex < 2) {
            if (intersectaSuperficie(p, d, v1, v2, v3, v4, &int1,
&qtemp, &t1)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int1;
                  tvals[pointsIndex] = t1;
                  quats[pointsIndex] = qtemp;
                  ++pointsIndex;
            }
      }
      // cara mayor Y
      v1 = vertices + 6;
      v2 = vertices + 9;
      v3 = vertices + 18;
      v4 = vertices + 21;
      if (pointsIndex < 2) {
            if (intersectaSuperficie(p, d, v1, v2, v3, v4, &int2,
&qtemp, &t2)) {
                  //ya tenemos un punto
                  points[pointsIndex] = int2;
                  tvals[pointsIndex] = t2;
```

```
                quats[pointsIndex] = qtemp;
                ++pointsIndex;
            }
        }
  if (pointsIndex < 2) {
            return false;
        }
  if (pointsIndex > 2) {
            printf("fuerteloco!!!\n");
        }

  // interpolar valor
      *intval = points[0] + (((points[1] - points[0]) / (tvals[1] -
tvals[0])) * (-tvals[0]));
      qval->set(quats[0] + ((quats[1] - quats[0]) / (tvals[1]-
tvals[0])) * (-tvals[0]));

  return true;
}
/*-----------------------------------------------------------------
------*/
int Fastrak::intersectaSuperficie(const FXVec3f &point, const FXVec3f
&ray,
                                  const int *v1, const int *v2, const
int *v3, const int *v4,
                                  FXVec3f *intval, FXQuatf *qval,
float *t) {

      FXVec3f accV;
      FXQuatf accQ;
      float accT;
      int nPoints = 0;

  accV.set(.0, .0, .0);
      accQ.set(.0, .0, .0, .0);
      accT = 0.0;
      // prueba todas las triangulaciones posibles
      if (intersectaTriangulo(point, ray, v1, v2, v3, intval, qval,
t)) {
            ++nPoints;
            accT += *t;
            accV += *intval;
            accQ += *qval;
```

```
      }
      if (intersectaTriangulo(point, ray, v4, v2, v3, intval, qval,
t)) {
            ++nPoints;
            accT += *t;
            accV += *intval;
            accQ += *qval;
      }
      if (intersectaTriangulo(point, ray, v1, v2, v4, intval, qval,
t)) {
            ++nPoints;
            accT += *t;
            accV += *intval;
            accQ += *qval;
      }
      if (intersectaTriangulo(point, ray, v1, v3, v4, intval, qval,
t)) {
            ++nPoints;
            accT += *t;
            accV += *intval;
            accQ += *qval;
      }
      if (nPoints == 0) {
    intval->set(.0, .0, .0);
            *t = 0.0;
            return 0;
      } else {
            *t = accT / (float)nPoints;
            *intval = accV / (float)nPoints;
    qval->set(accQ / (float)nPoints);
            return 1;
      }
}
/*-----------------------------------------------------------------
------*/
int Fastrak::intersectaTriangulo(const FXVec3f &p, const FXVec3f &d,
                                 const int *vr0, const int *vr1, const
int *vr2,
                                 FXVec3f *intval, FXQuatf *qval, float
*tval) {

      FXVec3f e1,e2,h,s,q,v0,v1,v2;
```

```
      float a,f,u,v;

  v0 = CPOINT(vr0[0], vr0[1], vr0[2]).pos;
      v1 = CPOINT(vr1[0], vr1[1], vr1[2]).pos;
      v2 = CPOINT(vr2[0], vr2[1], vr2[2]).pos;

  e1 = v1 - v0;
  e2 = v2 - v0;
  h = d ^ e2;
      a = e1 * h;

      if ((a > -0.00001) && (a < 0.00001))
            return 0;

      f = 1.0 / a;
  s = p - v0;
      u = f * (s * h);
      if ((u < 0.0) || (u > 1.0))
            return 0;

      q = s ^ e1;
      v = f * (d * q);
      if ((v < 0.0) || ((u + v) > 1.0))
            return 0;
      // interpolar valor
      FXVec3f v01, v21;
      FXQuatf q01, q21;
      // obtener v01
      v01 = CPOINT(vr0[0], vr0[1], vr0[2]).realPos +
                  (CPOINT(vr1[0], vr1[1], vr1[2]).realPos -
CPOINT(vr0[0], vr0[1], vr0[2]).realPos) * u;
      q01.set(CPOINT(vr0[0], vr0[1], vr0[2]).q +
                        (CPOINT(vr1[0], vr1[1], vr1[2]).q -
CPOINT(vr0[0], vr0[1], vr0[2]).q) * u);
      // obtener v21
      v21 = CPOINT(vr2[0], vr2[1], vr2[2]).realPos +
                       (CPOINT(vr1[0], vr1[1], vr1[2]).realPos -
CPOINT(vr2[0], vr2[1], vr2[2]).realPos) * u;
      q21.set(CPOINT(vr2[0], vr2[1], vr2[2]).q +
                        (CPOINT(vr1[0], vr1[1], vr1[2]).q -
CPOINT(vr2[0], vr2[1], vr2[2]).q) * u);
      // obtener v'
```

```
	float vp;
	vp = 1 - u;
	// interpolacion final
	*intval = v01 + ((v21 - v01) / vp) * v;
	qval->set(q01 + ((q21 - q01) / vp) * v);
	//printf("intval: %f %f %f\n", intval->x, intval->y, intval->z);
	// distancia del punto a la interseccion
	*tval = f * (e2 * q);
	return 1;
}
/*-------------------------------------------------------------------
------*/
int Fastrak::loadCorrectionMesh(char *filename, char t) {



	FILE *f;
	TCorrectionMeshPoint p;
	char type;
	int sizes[3];

	if (t == 't') {
		f = fopen(filename, "r");
		if (f == NULL) {
			printf("fichero incorrecto\n");
			return -1;
		}
		fscanf(f, "%c\t%d\t%d\t%d\n", &type, &xSize, &ySize,
&zSize);
		correctionMesh = new
TCorrectionMeshPoint[xSize*ySize*zSize];
		if (type == 't') {
			for (int i = 0; i < (xSize*ySize*zSize); i++) {
				fscanf(f,
"%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\n", &(p.realPos.x),
&(p.realPos.y), &(p.realPos.z),
&(p.pos.x), &(p.pos.y),
&(p.pos.z), &(p.q.w), &(p.q.x), &(p.q.y), &(p.q.z));
				correctionMesh[i] = p;
			}
		}
	} else if (t == 'b') {
		f = fopen(filename, "rb");
```

```
            if (f == NULL) {
                  printf("fichero incorrecto\n");
                  return -1;
            }
            int ret = 0;
            ret = fread(sizes, sizeof(int), 3, f);
            xSize = sizes[0];
            ySize = sizes[1];
            zSize = sizes[2];
            correctionMesh = new
TCorrectionMeshPoint[xSize*ySize*zSize];
            int count = xSize*ySize*zSize;
            int sss = fseek(f, 0, SEEK_END);
            sss = fseek(f, sizeof(int) * 3, SEEK_SET);
            // testing. NOT WORKING
//          while (count) {
//                ret = fread(correctionMesh,
sizeof(TCorrectionMeshPoint), xSize*ySize*zSize, f);
//                count -= ret;
//          }
      }
      minX = minY = minZ = 0;
      maxX = maxY = maxZ = 1;
      fclose(f);
}


/*-------------------------------------------------------------------
------*/
int Fastrak::setMesh(TCorrectionMeshPoint *m, int xs, int ys, int zs)
{


      minX = minY = minZ = 0;
      maxX = maxY = maxZ = 1;


      correctionMesh = m;
      xSize = xs;
      ySize = ys;
      zSize = zs;
      return 0;
}
/*-------------------------------------------------------------------
------*/
```

```
coordsToModelMatrix::coordsToModelMatrix(const FXVec3f &fastrakOrigin)
{
      origin = fastrakOrigin;
}
/*-------------------------------------------------------------------
------*/
void coordsToModelMatrix::getMatrixFromRaw(const FXVec3f &pos, const
FXQuatf &q, float *m) {

      float angle, rax, ray, raz;

      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
            glLoadIdentity();
            glTranslatef(origin.x, origin.y, origin.z);
            glRotatef(-90, 0, 0, 1);
            // fastrak origin axis
            glTranslatef(pos.x, pos.y, pos.z);
            rax = q.x / sqrt(1.0 - q.w * q.w);
            ray = q.y / sqrt(1.0 - q.w * q.w);
            raz = q.z / sqrt(1.0 - q.w * q.w);
            angle = (180.0f * acos(q.w) * 2.0f) / M_PI;
            glRotatef(angle, rax, ray, raz);
            glRotatef(180, 1, 0, 0);
            glGetFloatv(GL_MODELVIEW_MATRIX, m);
      glPopMatrix();
}
/*-------------------------------------------------------------------
------*/
void coordsToModelMatrix::getMatrix(const FXVec3f &pos, const FXQuatf
&q, float *m) {

      float angle, rax, ray, raz;

      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
            glLoadIdentity();
            // fastrak origin axis
            glTranslatef(pos.x, pos.y, pos.z);
            glRotatef(-90, 0, 0, 1);
            rax = q.x / sqrt(1.0 - q.w * q.w);
            ray = q.y / sqrt(1.0 - q.w * q.w);
```

```
            raz = q.z / sqrt(1.0 - q.w * q.w);
            angle = (180.0f * acos(q.w) * 2.0f) / M_PI;
            glRotatef(angle, rax, ray, raz);
            glRotatef(180, 1, 0, 0);
            glGetFloatv(GL_MODELVIEW_MATRIX, m);
      glPopMatrix();
}
/*-----------------------------------------------------------------
------*/
coordsToViewMatrix::coordsToViewMatrix(const FXVec3f &fastrakOrigin) {
      origin = fastrakOrigin;
}
/*-----------------------------------------------------------------
------*/
void coordsToViewMatrix::getMatrixFromRaw(const FXVec3f &pos, const
FXQuatf &q, float *m) {

      float angle, rax, ray, raz;

      rax = q.x / sqrt(1.0 - q.w * q.w);
      ray = q.y / sqrt(1.0 - q.w * q.w);
      raz = q.z / sqrt(1.0 - q.w * q.w);
      angle = (180.0f * acos(q.w) * 2.0f) / M_PI;

      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
            glLoadIdentity();
            glRotatef(-90, 1, 0, 0);
            glRotatef(90, 0, 0, 1);
            glRotatef(180, 1, 0, 0);
            glRotatef(-angle, rax, ray, raz);
            glTranslatef(-pos.x, -pos.y, -pos.z);
            glRotatef(90, 0, 0, 1);
            glTranslatef(-origin.x, -origin.y, -origin.z);
            glGetFloatv(GL_MODELVIEW_MATRIX, m);
      glPopMatrix();
}
/*-----------------------------------------------------------------
------*/
void coordsToViewMatrix::getMatrix(const FXVec3f &pos, const FXQuatf
&q, float *m) {
```

```
        float angle, rax, ray, raz;

        rax = q.x / sqrt(1.0 - q.w * q.w);
        ray = q.y / sqrt(1.0 - q.w * q.w);
        raz = q.z / sqrt(1.0 - q.w * q.w);
        angle = (180.0f * acos(q.w) * 2.0f) / M_PI;

        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
              glLoadIdentity();
              glRotatef(-90, 1, 0, 0);
              glRotatef(90, 0, 0, 1);
              glRotatef(180, 1, 0, 0);
              glRotatef(-angle, rax, ray, raz);
              glRotatef(90, 0, 0, 1);
              glTranslatef(-pos.x, -pos.y, -pos.z);
              glGetFloatv(GL_MODELVIEW_MATRIX, m);
        glPopMatrix();
}
```

9.2 Programas de Matlab para importación de datos

9.2.1 loadFastrakData.m

```
% function [time, acc, quat, samplerate] = loadFastrakData(filePath)
% -- Carga los elementos de un fichero de flujo optico
%      time  -> vector con los tiempos de cada muestra de puntos
%      pos   -> matriz que contiene las posiciones en cm en cada eje
%               cada columna tiene el vector correspondiente a una
muestra
%      quat  -> matriz que contiene los angulos de los vectores
%               de desplazamiento de los puntos. Cada columna
corresponde
%               a la lista de los angulos de desplazamiento entre los
puntos
%               de tracking de dos imagenes
function [time, pos, quat] = loadFastrakData(filePath)
sizeofHeader = 128;
ofFileID = fopen(filePath, 'r');
% Leemos la cabecera del fichero
```

```
% Calculamos el tamaño de los datos a leer
sizeofRegister = 8 + 4 * 3 + 4 * 4;
fseek(ofFileID, 0, 'eof');
sizeOfFile = ftell(ofFileID);
numberOfRegisters = (sizeOfFile - 128) / sizeofRegister;

% Cargamos las matrices de puntos
time = zeros([1, numberOfRegisters]);
pos = zeros([3, numberOfRegisters]);
quat = zeros([4, numberOfRegisters]);

% Leemos los tiempos de muestra en un vector
fseek(ofFileID, sizeofHeader, 'bof');
time = fread(ofFileID, numberOfRegisters, 'float64', 3 * 4 + 4 * 4);

% Leemos los vectores de posicion
fseek(ofFileID, sizeofHeader + 8, 'bof');
pos = fread(ofFileID, [3, numberOfRegisters], '3*float32', 4 * 4 + 8);

% Leemos todos quaternions de orientación
fseek(ofFileID, sizeofHeader + 8 + 3 * 4, 'bof');
quat = fread(ofFileID, [4, numberOfRegisters], '4*float32', 8 + 3 *
4);

fclose(ofFileID);
```

9.2.2 loadMtxData.m

```
% function [time, acc, quat, samplerate] = loadMtxData(filePath)
% -- Carga los elementos de un fichero de flujo optico
%      time  -> vector con los tiempos de cada muestra de puntos
%      acc   -> matriz que contiene las aceleraciones en m/s en cada
eje
%               cada columna tiene el vector correspondiente a una
muestra
%      quat  -> matriz que contiene los angulos de los vectores
%               de desplazamiento de los puntos. Cada columna
corresponde
%               a la lista de los angulos de desplazamiento entre los
puntos
%               de tracking de dos imagenes
function [time, acc, quat, samplerate] = loadMtxData(filePath)
sizeofHeader = 128;
```

```
ofFileID = fopen(filePath, 'r');
% Leemos la cabecera del fichero

% Calculamos el tamaño de los datos a leer
sizeofRegister = 4 * 3 + 4 * 4 + 8;
fseek(ofFileID, 0, 'eof');
sizeOfFile = ftell(ofFileID);
numberOfRegisters = (sizeOfFile - sizeofHeader) / sizeofRegister;

% Cargamos las matrices de puntos
time = zeros([1, numberOfRegisters]);
acc = zeros([3, numberOfRegisters]);
quat = zeros([4, numberOfRegisters]);
samplerate = zeros([1, 1]);

% Leemos el samplerate
fseek(ofFileID, 4, 'bof');
samplerate = fread(ofFileID, 1, 'uint32');

% Leemos los vectores de aceleración
fseek(ofFileID, sizeofHeader, 'bof');
acc = fread(ofFileID, [3, numberOfRegisters], '3*float32',
sizeofRegister - 3 * 4);

% Leemos todos quaternions de orientación
fseek(ofFileID, sizeofHeader + 3 * 4, 'bof');
quat = fread(ofFileID, [4, numberOfRegisters], '4*float32', 8 + 3 *
4);

% Leemos los tiempos de muestra en un vector
fseek(ofFileID, sizeofHeader + 3 * 4 + 4 * 4, 'bof');
time = fread(ofFileID, numberOfRegisters, 'float64', 3 * 4 + 4 * 4);

fclose(ofFileID);
```

9.2.3 loadOpticaFlowData.m

```
% function [time, count, r, theta] = loadOpticalFlowData(filePath)
% -- Carga los elementos de un fichero de flujo optico
%      time  -> vector con los tiempos de cada muestra de puntos
%      count -> vector con el numero de deplazamientos validos por
imagen
```

```
%      r     -> matriz que contiene los modulos de los vectores
%               de desplazamiento de los puntos. Cada columna
corresponde
%               a la lista de los desplazamientos entre los puntos de
%               de tracking de dos imagenes
%      theta -> matriz que contiene los angulos de los vectores
%               de desplazamiento de los puntos. Cada columna
corresponde
%               a la lista de los angulos de desplazamiento entre los
puntos
%               de tracking de dos imagenes
function [time, count, r, theta] = loadOpticalFlowData(filePath)
sizeofHeader = 128;
ofFileID = fopen(filePath, 'r');
% Leemos la cabecera del fichero
hd = fread(ofFileID, 1, 'uint32');
vd = fread(ofFileID, 1, 'uint32');
featuresPerQuadrant = fread(ofFileID, 1, 'uint32');
% Calculamos el tamaño de los datos a leer
featuresPerImage = double(hd * vd * featuresPerQuadrant);
% sizeofRegister = sizeof(time) + sizeof(counter) + sizeof(prevVector)
+ sizeof(currVector)
sizeofRegister = 8 + 4 + featuresPerImage * 8 + featuresPerImage * 8;
fseek(ofFileID, 0, 'eof');
sizeOfFile = ftell(ofFileID);
numberOfRegisters = (sizeOfFile - 128) / sizeofRegister;
% Cargamos las matrices de puntos
time = zeros([1, numberOfRegisters]);
count = zeros([1, numberOfRegisters]);
prevPoints = zeros([2 * featuresPerImage, numberOfRegisters]);
currPoints = zeros([2 * featuresPerImage, numberOfRegisters]);
% Leemos los tiempos de muestra en un vector
fseek(ofFileID, sizeofHeader, 'bof');
time = fread(ofFileID, numberOfRegisters, 'float64', sizeofRegister -
8);
% Leemos el numero de puntos válidos en cada lista de parejas
fseek(ofFileID, sizeofHeader + 8, 'bof');
count = fread(ofFileID, numberOfRegisters, 'uint32', sizeofRegister -
4);
% Leemos todos las X,Y del vector de puntos previos
fseek(ofFileID, sizeofHeader + 8 + 4, 'bof');
sizeofPointsVector = featuresPerImage * 2;
```

```
prevPoints = fread(ofFileID, [featuresPerImage * 2,
numberOfRegisters], '20*float32', (featuresPerImage * 2 * 4) + 8 + 4);

% Leemos todos las X,Y del vector de puntos previos

fseek(ofFileID, sizeofHeader + 8 + 4 + (featuresPerImage * 2 * 4),
'bof');

currPoints = fread(ofFileID, [featuresPerImage * 2,
numberOfRegisters], '20*float32', 8 + 4 + (featuresPerImage * 2 * 4));

fclose(ofFileID);

% Creamos una matriz de complejos para calcular despues módulo y
ángulo

% Como los puntos se guardaron en el orden (x1,y1)(x2,y2)... las filas

% impares de las matrices contienen las X y las pares las Y

% [theta, r] = cart2pol(x,y)

xDisp = currPoints(1:2:featuresPerImage * 2, :) -
prevPoints(1:2:featuresPerImage * 2, :);

yDisp = currPoints(2:2:featuresPerImage * 2, :) -
prevPoints(2:2:featuresPerImage * 2, :);

[theta, r] = cart2pol(xDisp, yDisp);
```

9.2.4 loadSensorsData.m

```
% [time, radius, theta, acc, quat, pos] =
loadSensorsData(fileNamePath, fileNamePrefix)

% Carga los datos de una captura reduciéndolos a los instantes en los

% que se tomaron las muestras de los vectores de flujo optico

function [time, radius, theta, acc, quat, pos] =
loadSensorsData(fileNamePath, fileNamePrefix)


%% Lectura de datos de fichero


% Nombres de fichero

opticalFlowFileName = [fileNamePath, fileNamePrefix, '.ofl'];

fastrakFileName = [fileNamePath, fileNamePrefix, '.ftk'];

mtxFileName = [fileNamePath, fileNamePrefix, '.mcf'];


% Carga de los datos de flujo optico

disp(['--> Loading optical flow data from: ', opticalFlowFileName, '
...']);

[ofTime, ~, ofRadius, ofTheta] =
loadOpticalFlowData(opticalFlowFileName);

disp([num2str(length(ofTime)), ' matching vectors loaded from ',
opticalFlowFileName]);


% Carga de los datos del sensor Mtx
```

```
disp(['--> Loading mtx sensor data from: ', mtxFileName, ' ...']);
[mtTime, mtAcc, mtQuat, ~] = loadMtxData(mtxFileName);
disp([num2str(length(mtTime)), ' measures loaded from ',
mtxFileName]);


% Carga de los datos del fastrak
disp(['--> Loading fastrak data from: ', fastrakFileName, ' ...']);
[ftTime, ftPos, ~] = loadFastrakData(fastrakFileName);
disp([num2str(length(ftTime)), ' positions loaded from ',
fastrakFileName]);


%% Flujo óptico


% La base de tiempos usada será la de los vectores de flujo optico
porque
% no son interpolables
time = ofTime;
radius = ofRadius';
theta = ofTheta';


%% Ajuste de vectores de flujo optico y quaterniones


% Ajustamos los vectores, eliminando aquellas muestras donde exista
% información de orientación pero no de flujo óptico y viceversa.


% Ajuste de los arrays por el principio
if time(1) < mtTime(1)
    i = 1;
    while time(i) < mtTime(1)
        i = i + 1;
    end
    time = time(i:length(time),:);
    radius = radius(i:length(radius),:);
    theta = theta(i:length(theta),:);
end


% Ajuste de los arrays por el final
if time(length(time)) > mtTime(length(mtTime))
    i = length(time);
    while time(i) > mtTime(length(mtTime))
        i = i - 1;
```

```
    end
    time = time(1:i,:);
    radius = radius(1:i,:);
    theta = theta(1:i,:);
end

disp(['--> Reducing data to ', num2str(length(time)), ' samples']);

%% Interpolación de los quaterniones
disp('----> Quaternion interpolation...');
% Calculamos los factores de interpolación
[k, distance] = dsearchn(mtTime, time);
distance2nearest = time - mtTime(k);
mtIndex = k - lt(distance2nearest, zeros(size(distance2nearest)));
distance = time - mtTime(mtIndex);
intervalLength = mtTime(2:length(mtTime)) - mtTime(1:length(mtTime)-
1);
slerpFactor = distance ./ intervalLength(mtIndex);

% Creamos el objeto quaternion y hacemos la interpolacion esferica
quat = zeros(length(mtIndex), 4);
for i = 1:length(mtIndex)
    q0 = quaternion([mtQuat(1,mtIndex(i)), mtQuat(2,mtIndex(i)),
mtQuat(3,mtIndex(i)), mtQuat(4,mtIndex(i))]);
    q1 = quaternion([mtQuat(1,mtIndex(i)+1), mtQuat(2,mtIndex(i)+1),
mtQuat(3,mtIndex(i)+1), mtQuat(4,mtIndex(i)+1)]);
    qInterpolated = qinterp(q0, q1, slerpFactor(i));
    quat(i,:) = [qInterpolated.s qInterpolated.v];
end;
% quat = quaternion(mtQuat(1,:), mtQuat(2,:), mtQuat(3,:),
mtQuat(4,:));
% quat = slerp(quat(mtIndex), quat(mtIndex+1), slerpFactor');
% quat = [quat.s; quat.x; quat.y; quat.z]';

%% Interpolación de la aceleración
disp('----> Acceleration interpolation...');
acc = interp1(mtTime, mtAcc', time, 'spline');

%% Interpolamos las posiciones
disp('----> Positions interpolation...');
pos = interp1(ftTime, ftPos', time, 'spline');
disp('--> End.');
```

9.3 Programas de entrenamiento y simulación

9.3.1 generalFicheroDatosRelativos.m

```
sourceFilesPath = '..\capturas\';
sourceNamePrefix = 'captura02';
destNamePrefix = 'matrices_datos';
fromData = 700;
toData = 2400;
%% Carga de los datos de los sensores
[time, radius, theta, acc, quat, pos] =
loadSensorsData(sourceFilesPath, sourceNamePrefix);
% Se seleccionan los que se van a usar
if toData < fromData;
    disp('--> Error, toData must be greater than fromData');
end;
if toData > length(time);
    disp('--> Error, toData exceeds avaliable data index.');
end;
time = time(fromData:toData);
radius = radius(fromData:toData,:);
theta = theta(fromData:toData,:);
acc = acc(fromData:toData,:);
quat = quat(fromData:toData,:);
pos = pos(fromData:toData,:);

%% Ajuste a condiciones iniciales de tiempo y posición fastrak
time = time - time(1);
pos(:,1) = pos(:,1) - pos(1,1);
pos(:,2) = pos(:,2) - pos(1,2);
pos(:,3) = pos(:,3) - pos(1,3);

%% Calculo de los incrementos de posicion
diffPos = zeros(size(pos));
diffPos(2:size(diffPos,1),:) = pos(2:size(pos,1),:) -
pos(1:(size(pos,1)-1),:);

%% Expresamos las aceleraciones en el marco de referencia global
gVec = [0.0 0.0 9.812687];
for i=1:size(quat, 1)
    q = quaternion(quat(i,:));
```

```
    qConj = inv(q);
    acc(i,:) = qConj * acc(i,:);
    acc(i,:) = acc(i,:) - gVec;
end;

%% Integración doble de la acc para obtener los incrementos en
posicion
disp('--> Acceleration integration...');
accPos = cumtrapz(time, acc);
accPos = cumtrapz(time, accPos);

diffAccPos = zeros(size(accPos));
diffAccPos(1,:) = [0 0 0];
diffAccPos(2:size(diffAccPos,1),:) = accPos(2:size(accPos,1),:) -
accPos(1:(size(accPos,1)-1),:);

%% Formacion de las matrices U e Y
disp('--> Creating input and output pattern matrix...');
U = cat(2, time, radius, theta, diffAccPos, quat);
U = U'; % dimension RxQ   R numero de entradas

%% Conjunto de datos para X
Y = diffPos(:,1); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_X.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m

%% Conjunto de datos para Y
Y = diffPos(:,2); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_Y.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m

%% Conjunto de datos para Z
Y = diffPos(:,3); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_Z.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m
```

```
clear time radius theta acc quat pos diffPos accPos diffAccPos
clear fromData toData q qConj gVec i
clear sourceFilesPath sourceNamePrefix destNamePrefix
```

9.3.2 generaFicheroDatosAbsolutos.m

```
sourceFilesPath = '..\capturas\';
sourceNamePrefix = 'captura02';
destNamePrefix = 'matrices_datos_test';
%fromData = 700;
%toData = 2400;
fromData = 50;
toData = 700;
%% Carga de los datos de los sensores
[time, radius, theta, acc, quat, pos] =
loadSensorsData(sourceFilesPath, sourceNamePrefix);
% Se seleccionan los que se van a usar
if toData < fromData;
    disp('--> Error, toData must be greater than fromData');
end;
if toData > length(time);
    disp('--> Error, toData exceeds avaliable data index.');
end;
time = time(fromData:toData);
radius = radius(fromData:toData,:);
theta = theta(fromData:toData,:);
acc = acc(fromData:toData,:);
quat = quat(fromData:toData,:);
pos = pos(fromData:toData,:);

%% Ajuste a condiciones iniciales de tiempo y posición fastrak
time = time - time(1);
pos(:,1) = pos(:,1) - pos(1,1);
pos(:,2) = pos(:,2) - pos(1,2);
pos(:,3) = pos(:,3) - pos(1,3);

%% Expresamos las aceleraciones en el marco de referencia global
gVec = [0.0 0.0 9.812687];
for i=1:size(quat, 1)
    q = quaternion(quat(i,:));
    qConj = inv(q);
```

```
    acc(i,:) = qConj * acc(i,:);
    acc(i,:) = acc(i,:) - gVec;
end;

%% Integración doble de la acc para obtener los incrementos en
posicion
disp('--> Acceleration integration...');
accPos = cumtrapz(time, acc);
accPos = cumtrapz(time, accPos);

%% Formacion de las matrices U e Y
disp('--> Creating input and output pattern matrix...');
U = cat(2, time, radius, theta, accPos, quat);
U = U'; % dimension RxQ   R numero de entradas
clear time radius theta acc quat accPos fromData toData q qConj gVec i

%% Creación de los ficheros de datos
% Fichero para X
Y = pos(:,1); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_X.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m
% Fichero para Y
Y = pos(:,2); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_Y.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m
% Fichero para Z
Y = pos(:,3); % Y solo contiene la coordenada X
Y = Y'; % dimension SxQ   S numero de salidas Q numero de patrones
fichero_datos = [destNamePrefix '_Z.mat'];
disp(['--> Saving to ' fichero_datos '...']);
save(fichero_datos, 'fichero_datos', 'U', 'Y') % fichero para busca.m

clear pos sourceFilesPath sourceNamePrefix destNamePrefix
```

9.3.3 initializeSelfOrganizing.m

```
% initializeSelfOrganizing.m
%   fichero que define parametros por defecto de sistema Neuro-Fuzzy.
```

```
%   Observese que es independiente de entradas y salidas y su
dimension
%   U entradas e Y salidas
%   porcentaje_tranning indica el porcentaje de valores que se toman
para
%   entrenamiento dado entre 0 y 100 por ciento
% se parte de la variable matriz_krobustos que tiene el siguiente
% formato de fila:

porcentaje_tranning = 70;
seleccion=[];

nPatrones = size(U,2);
n = round(nPatrones * porcentaje_tranning / 100);
i = 1;
while i <= n
    r = ceil(nPatrones .* rand(1));
    if isempty(intersect(seleccion, r))
        i = i + 1;
        seleccion=cat(1,seleccion, r);
    end
end;

% seleccionamos los patrones del conjunto de entrenamiento y
generalizacion.
seleccion = seleccion';
defecto_generalizacion = setdiff(1:nPatrones,seleccion);

% y el numero de salidas del sistema Neuro-Fuzzy.

numero_salida = size(Y, 1); % Numero de salidas

Unuevo=U;
Ynuevo=Y;
maximo_U=max(max(U)); % valor maximo para la U
maximo_Y=max(max(Y)); % valor maximo para la Y
clear U Y;
selection;



U=U/maximo_U; % normalizacion de datos por el maximo valor de la
```

```
% entrada U
Y=Y/maximo_Y; % normalizacion de datos por el maximo valor de la
% entrada Y

num_total=n; % numero total de comandos tomados en seleccion.
clear Unuevo Ynuevo indice1 Ybuf


sigma0=1.75
N2=75; % N2=105 numero de reglas

lr1=1
mf=10 % frecuencia de presentacion de resultados
me1= 20000 % me1= 500 numero de epocas
```

9.3.4 selfOrganizing.m

```
%
% sistema neuro-fuzzy que determina parametros de tal sistema
% mediante entrenamiento
%
% entrada:
%           U entrada de dimension (RxQ)
%           Y salida de dimension (SxQ)
%           N2 numero de neuronas de la red auto-organizativa
%          sv     valores estimados para los nodos de reglas.
%          sigma  anchuras de las funciones de pertenencia.

% salidas:
%         w0 vector de pesos resultante.
%

% entradas al sistema

[R,Q] = size(U); % dimension (RxQ)
[S,Q1] = size(Y);  % dimension (SxQ)
if Q1 ~= Q
    error('entrada y salida tienen que tener igual Q');
end
```

```
Umin = min(U');
Umin = Umin';  % dimension RX1

Umax = max(U');
Umax = Umax';  % dimension Rx1

% salidas del sistema

[S,Q1] = size(Y);  % dimension (SxQ)
if Q1 ~= Q
    error('entrada y salida tienen que tener igual Q');
end

Ymin = min(Y');
Ymin = Ymin';  % dimension (Sx1)
Ymax = max(Y');
Ymax = Ymax'; % dimension (Sx1)

N1 = R; % numero de entradas
N3 = S; % numero de salidas.
I = [U; Y];

Imin = [Umin; Ymin]; % dimension (R+S)x1
Imax = [Umax; Ymax]; % dimension (R+S)x1
Ilimite = [Imin Imax]; % dimension ((R+S)x2)

net = newsom(Ilimite, N2);
net.trainParam.show = mf;
net.trainParam.epochs = me1;
net = train(net,I);
w0 = net.IW{1,1}
```

9.3.5 ruleSelection.m

```
% seleccion_reglasv3.m adaptada a solo patrones defecto.
%  una vez que se ha realizado el entrenamiento segun algoritmo
%  auto-organizativo esta funcion analiza el resultado obtenido,
calculando
%   regla(vector peso), se acerca mas las componentes de la entrada.
Ademas
%   elimina reglas comunes a ambos grupos hasta quedarse con conjuntos
%   disjuntos de reglas y genera nuevas.
```

```
%
%
%   variables de entrada:
%              w0 vector de pesos producto de red de Kohonen despues
de ser entrenados.
%   salidas:
%
%         w_reducido    vector de pesos resultado de proceso de
seleccion.
%
%         nodos_sobrevivientes
%                       los nodos de reglas que realmente intervienen
en
%                       el proceso de generación de salidas.
%         N2      nuevo numero de reglas.
%
%
alfa=1; % valor por el que se multiplica la entrada U.

%load inicia.mat;  % carga U normalizada, Y, seleccion,
                  % defecto_generalizacion, ...
                  % seleccionamos los patrones del conjunto de
entrenamiento.
                  % y el numero de salidas del sistema Neuro-Fuzzy.
I = [alfa * U' Y'];
[R, Q] = size(U);
[S, Q1] = size(Y);
N1 = R; % numero de entradas
N3 = S; % numero de salidas
[N2, Q2] = size(w0);

w_seleccion = w0;

for i = 1:num_total
    for j = 1:N2
        distancias(i,j) = norm(w_seleccion(j,:) - I(i,:));
    end
end

for i = 1:num_total
    [minimo(i), nodo(i)] = min(distancias(i,:));
```

```
    disp(['cercania ganador: ', num2str(minimo(i)), ' nodo(Regla):',
num2str(nodo(i)) ]);
end
nodos_sobrevivientes = unique(nodo);
N2 = max(size(nodos_sobrevivientes));
for i = 1:N2
    w_reducido(i,:) = w0(nodos_sobrevivientes(i),:);
end
```

9.3.6 findSigma.m

```
% findSigma.m
%
% calculo manual del sigma más adecuado para proseguir entrenamiento.


minimo=1;
maximo=100;
factor=0.0001;
for i=minimo:maximo


sigmafactor=factor*i;
[error,errorgen,problema_entradas,m,sv,sigma]=calcErrorNF(w_reducido,s
eleccion,defecto_generalizacion,fichero_datos,sigmafactor);
disp(['sigma=' num2str(sigmafactor) ' error= ' num2str(error) '
error_gen= ' num2str(errorgen) ...
       ' reglas NaN =' num2str(problema_entradas)]);
 matriz_errores(i,:)=[sigmafactor error errorgen];


end
disp(' ');
disp(' ');
disp(' resumen del barrido de sigma');
disp('-----------------------------');
disp(' ');
disp(['rango de sigma entre ' num2str(factor*minimo) ' y '
num2str(factor*maximo)]);
[valorerrormin1,i]=min(matriz_errores(:,2));
ment=i;
if (N3>3)
    [valorerrormin4,l]=min(matriz_errores(:,5));
end
if (N3>2)
  [valorerrormin3,k]=min(matriz_errores(:,4));
```

```
end
if (N3>1)
   [valorerrormin2,j]=min(matriz_errores(:,3));
 end
disp(['valor de error minimo Salida 1: ' num2str(matriz_errores(i,2))
...
    ' sigma=' num2str(matriz_errores(i,1)) ' errgen= '
num2str(matriz_errores(i,(2+N3):(1+2*N3))) ]);
if (N3>3)
   disp(['valor de error minimo Salida 4: '
num2str(matriz_errores(l,5)) ...
    ' sigma=' num2str(matriz_errores(l,1)) ' errgen= '
num2str(matriz_errores(l,(2+N3):(1+2*N3))) ]);
end
if (N3>2)
    disp(['valor de error minimo Salida 3: '
num2str(matriz_errores(k,4)) ...
    ' sigma=' num2str(matriz_errores(k,1)) ' errgen= '
num2str(matriz_errores(k,(2+N3):(1+2*N3))) ]);
end
if (N3>1)
  disp(['valor de error minimo Salida 2: '
num2str(matriz_errores(j,3)) ...
    ' sigma=' num2str(matriz_errores(j,1)) ' errgen= '
num2str(matriz_errores(j,(2+N3):(1+2*N3))) ]);
end
if (N3>3)
    [valorerrormin4,l]=min(matriz_errores(:,6));
end
if (N3>2)
  [valorerrormin3,k]=min(matriz_errores(:,5));
end
if (N3>1)
   [valorerrormin2,j]=min(matriz_errores(:,4));
 end
[valorerrormin,i]=min(matriz_errores(:,3));
mgen=i;
disp(['valor de error generalizacion minimo 1 : '
num2str(matriz_errores(i,2+N3)) ...
    ' sigma=' num2str(matriz_errores(i,1)) ' error='
num2str(matriz_errores(i,2:(1+N3))) ]);


if (N3>3)
```

```
  disp(['valor de error generalizacion minimo 2 : '
num2str(matriz_errores(l,2+N3)) ...
    ' sigma=' num2str(matriz_errores(l,1)) ' error='
num2str(matriz_errores(l,2:(1+N3))) ]);
end
if (N3>2)
    disp(['valor de error generalizacion minimo 3 : '
num2str(matriz_errores(k,2+N3)) ...
    ' sigma=' num2str(matriz_errores(k,1)) ' error='
num2str(matriz_errores(k,2:(1+N3))) ]);
end
if (N3>1)
  disp(['valor de error generalizacion minimo 2 : '
num2str(matriz_errores(j,2+N3)) ...
    ' sigma=' num2str(matriz_errores(j,1)) ' error='
num2str(matriz_errores(j,2:(1+N3))) ]);
 End
```

9.3.7 calcErrorNF.m

```
function
[error,errorgen,problema_entradas,m,sv,sigma]=calculo_error_NFv1(w_ext
endido,seleccion,defecto_generalizacion,fichero_datos,sigmafactor)
% variables de entrada
%  w_extendido    los pesos de la red neuro-fuzzy ya entrenada con el
%                 método no supervisado.
% seleccion      patrones seleccionados en entrenamiento.
% seleccion_generalizacion  patrones seleccionados para test de
%                           generalizacion.
% fichero_datos  nombre del fichero con los datos fuentes sin
normalizar.
% sigmafactor    valor de sigma tomado manualmente.


% variables de salida
% error       error cuadratico medio derivado del conjunto de
entrenamiento.
%             tiene dimension 1xN3
%
% errorgen    error cuadratico medio derivado del conjunto de test.
%             tiene dimension 1xN3
% problema_entradas  conjunto de patrones que dan problemas de
division por
%                    cero en el test
% m           centros de las funciones de pertenencia segun
entrenamiento
```

```
%             no supervisado
% sv          valores estimados segun entrenamiento no supervisado
% sigma       valor de sigma tomado manualmente, es igual a
sigmafactor dimensionado para sistema N-F.


w=w_extendido;
eval(['load ./',fichero_datos]); % datos no normalizados


% seleccionamos los patrones del conjunto de entrenamiento.
% y el numero de salidas del sistema Neuro-Fuzzy.


numero_salida=2; % adapta la matriz Y a tener solo un numero de
% salidas numero_salida.


Unuevo=U;
Ynuevo=Y;
maximo_U=max(max(U)); % valor maximo para la U
maximo_Y=max(max(Y)); % valor maximo para la Y
clear U Y;
selection;
% introducimos en una variable aparte los datos utilizados para
% generalizacion que seran utilizados en el entrenamiento como forma
de
% testear la bondad del entrenamiento, el como va variando la
% generalizacion.
U=U/maximo_U; % normalizacion de datos por el maximo valor de la
% entrada U
Y=Y/maximo_Y; % normalizacion de datos por el maximo valor de la
% entrada Y


seleccion_generalizacion=[defecto_generalizacion];
numero_salida=1; % adapta la matriz Y a tener solo un numero de
% salidas numero_salida.


Unew_generalizacion=Unuevo;
Ynew_generalizacion=Ynuevo;


selectionGen;
```

```
Ugeneralizacion=Ugeneralizacion/maximo_U; % normalizacion de datos por
el maximo valor de la entrada U
Ygeneralizacion=Ygeneralizacion/maximo_Y; % normalizacion de datos por
el maximo valor de la entrada U


clear Unuevo Ynuevo indice1 Ybuf Unew_generalizacion
Ynew_generalizacion maximoUgeneralizacion
clear maximoU maximoabsoluto


if nargin==3
    sigmafactor=1;
end


%
%
%     w  vectores de pesos obtenidos anteriormente.
%     U entradas a la red
%     Y salidas de la red
%     sigmafactor escalar por el que se multiplica la matriz de 1's
que
%                 define la matriz sigma del sistema N-F.
%
%  la salida es:
%
%     error  error cuadratico medio obtenido con minimizacion.
%     w      vector de pesos despues de minimizacion.
%     N2     numero de reglas despues de minimizacion.
%


% entradas al sistema neuro-fuzzy
alfa=1; % factor que multiplica a U (Por defecto es 1 dado que U suele
estar normalizado).
[N2,dimw]=size(w); % N2 numero de reglas consideradas.
[R,Q]=size(U); % dimension (RxQ)
N1=R;
Umin=min(U');
Umin=Umin';


Umax=max(U');
Umax=Umax';


% salidas del sistema neuro-fuzzy
```

```
[S,Q1]=size(Y);  % dimension (SxQ)
N3=S;
if Q1~=Q
    error('entrada y salida tienen que tener igual Q');
end

Ymin=min(Y');
Ymin=Ymin';
Ymax=max(Y');
Ymax=Ymax';

% asignacion de las centros de las funciones de
% pertenencia.

for i=1:N1
    for j=1:N2
        m(i,j)=w(j,i);
    end
end

% asignacion de los valores estimados para los
% nodos de reglas.

for k=1:N3
    for j=1:N2
        sv(j,k)=w(j,N1+k);
    end
end

% Analisis del error que se produce.

sigma=sigmafactor*ones(N1,N2); % inicializacion de las sigmas

[R,Q]=size(U);
[S,Q1]=size(Y);
if Q~=Q1
    error('dimension de U e Y incorrecta para calcular error');
end
```

```
% presentation phase
problema_entradas=[];
for q1=1:Q
    ypr=simnf(U(:,q1),m,sv,sigma,alfa);
    eseg1(q1)=Y(1,q1)-ypr(:,1);
    if (N3>1)
        esegf2(q1)=Y(2,q1)-ypr(:,2);
    end
    if (N3>2)
        esegf3(q1)=Y(3,q1)-ypr(:,3);
    end
    if (N3>3)
        esegf4(q1)=Y(4,q1)-ypr(:,4);
    end

        % se toma nota de las entradas de test que dan problema al
aparecer
    % divisiones por cero.
    if (N3>3)
        if ypr(:,4)>=99
            problema_entradas=[problema_entradas seleccion(q1)];
        end
    end
    if (N3>2)
        if ypr(:,3)>=99
            problema_entradas=[problema_entradas seleccion(q1)];
        end
    end
    if (N3>1)
        if ypr(:,2)>=99
            problema_entradas=[problema_entradas seleccion(q1)];
        end
    end
    if ypr(:,1)>=99
        problema_entradas=[problema_entradas seleccion(q1)];
    end

end

% Observese que se pesa con cero todas las salidas menos
% la primera.
```

```
if (N3>3)
    error(4)=sumsqr(esegf4);
end
if (N3>2)
    error(3)=sumsqr(esegf3);
end
if (N3>1)
    error(2)=sumsqr(esegf2);
end
error(1)=sumsqr(eseg1);

% error off-line de datos de generalizacion

[Rgeneralizacion,Qgeneralizacion]=size(Ugeneralizacion);
[Sgeneralizacion,Q1generalizacion]=size(Ygeneralizacion);
for q1=1:Qgeneralizacion
    ypr=simnf(Ugeneralizacion(:,q1),m,sv,sigma,alfa);
    eseg1generalizacion(q1)=Ygeneralizacion(1,q1)-ypr(:,1);
    if (N3>3)
        eseg4generalizacion(q1)=Ygeneralizacion(4,q1)-ypr(:,4);
    end
    if (N3>2)
        eseg3generalizacion(q1)=Ygeneralizacion(3,q1)-ypr(:,3);
    end
    if (N3>1)
        eseg2generalizacion(q1)=Ygeneralizacion(2,q1)-ypr(:,2);
    end

    % se toma nota de las entradas de test que dan problema al
aparecer
    % divisiones por cero.
    if (N3>3)
        if ypr(:,4)>=99
            problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
        end
    end
    if (N3>2)
        if ypr(:,3)>=99
```

```
            problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
        end
    end
    if (N3>1)
        if ypr(:,2)>=99
            problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
        end
    end
    if ypr(:,1)>=99
        problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
    end
end


if (N3>3)
    errorgen(4)=sumsqr(eseg4generalizacion);
end
if (N3>2)
    errorgen(3)=sumsqr(eseg3generalizacion);
end
if (N3>1)
    errorgen(2)=sumsqr(eseg2generalizacion);
end
errorgen(1)=sumsqr(eseg1generalizacion);
```

9.3.8 initializeRBN.m

```
%   fichero que define parametros por defecto de sistema Neuro-Fuzzy.
%   se utiliza despues de aplicar algoritmos desarrollados de
seleccion de
%   reglas.


   % la Y en el fichero .mat tiene tantas salidas como casos
diferentes
   % y asocia a cada caso un 1 cuando es el patron.


 eval(['load ./',fichero_datos]); % datos no normalizados



    % seleccionamos los patrones del conjunto de entrenamiento.
    % y el numero de salidas del sistema Neuro-Fuzzy.
```

```
  numero_salida=1; % adapta la matriz Y a tener solo un numero de
                   % salidas numero_salida.


  Unuevo=U;
  Ynuevo=Y;
  maximo_U=max(max(U)); % valor maximo para la U
  maximo_Y=max(max(Y)); % valor maximo para la Y
  clear U Y;
 selection;
   % introducimos en una variable aparte los datos utilizados para
   % generalizacion que seran utilizados en el entrenamiento como
forma de
   % testear la bondad del entrenamiento, el como va variando la
   % generalizacion.
  U=U/maximo_U; % normalizacion de datos por el maximo valor de la
                   % entrada U
  Y=Y/maximo_Y; % normalizacion de datos por el maximo valor de la
                   % entrada Y
  selectionGen=[defecto_generalizacion];
  numero_salida=1; % adapta la matriz Y a tener solo un numero de
                   % salidas numero_salida.

  Unew_generalizacion=Unuevo;
  Ynew_generalizacion=Ynuevo;

  selectionGen;

   Ugeneralizacion=Ugeneralizacion/maximo_U; % normalizacion de datos
por el maximo valor de la entrada U
  Ygeneralizacion=Ygeneralizacion/maximo_Y; % normalizacion de datos
por el maximo valor de la entrada U

  clear Unuevo Ynuevo indice1 Ybuf Unew_generalizacion
Ynew_generalizacion maximoUgeneralizacion
  clear maximoU maximoabsoluto

    lr=1e-006  % velocidad de aprendizaje
    me=200 % numero de entrenamientos
```

9.3.9 rbn.m

```
% rbn.m
%                          solo realiza el entrenamiento de la red de
base
%                          radial segun el metodos de los minimos
%                          cuadrados.

numero_guarda=20000; % guarda resultado en result_backup cada
                     % numero_guarda de epocas


                     % valores de dimension utilizados por el programa

[R,Q]=size(U);
[S,Q1]=size(Y);
N1=R; % numero de entradas
N3=S; % numero de salidas

if ~exist('minicial')
  minicial=0;
end

% Reservamos el espacio para las variables

e1=zeros(1,me); % salida 1
eseg1t=zeros(1,me);
eseg1tgeneralizacion=zeros(1,me);
if (N3>1)
 e2=zeros(1,me); % salida 2
 esegf2t=zeros(1,me);
 esegf2tgeneralizacion=zeros(1,me);
end
if (N3>2)
 e3=zeros(1,me); % salida 3
 esegf3t=zeros(1,me);
 esegf3tgeneralizacion=zeros(1,me);
end
if (N3>3)
 e4=zeros(1,me); % salida 4
 esegf4t=zeros(1,me);
```

```
 esegf4tgeneralizacion=zeros(1,me);
end
problema_entradas_entrenamiento=[];

for ent=1:me
  for q1=1:Q
    %% Presentation phase
    for i=1:N1
      for j=1:N2
        P(i,j)=exp(-((alfa*U(i,q1)-m(i,j))^2)/(sigma(i,j)^2));
      end
    end
    for j=1:N2
      ro(j)=min(P(:,j));
    end
    if (sum(ro)==0)

problema_entradas_entrenamiento=[problema_entradas_entrenamiento
seleccion(q1)];
    end
    for k=1:N3
      sy(k)=sum(sv(:,k).*ro')/sum(ro);
    end
    e=Y(:,q1)-sy';  % calculo error on-line.
    eseg(q1)=e(1); %salida 1
    if (N3>1)
      eseg2(q1)=e(2); % salida 2
    end
    if (N3>2)
      eseg3(q1)=e(3); % salida 3
    end
    if (N3>3)
      eseg4(q1)=e(4); % salida 4
    end

    %% Learning phase
    for j=1:N2
      for k=1:N3
        sv(j,k)=sv(j,k)+lr*e(k)*ro(j)/sum(ro);
      end
    end
```

```
  for j=1:N2
    for i=1:N1
      if ro(j)==P(i,j)
        q(i,j)=1;
      else
        q(i,j)=0;
      end
      % actualizacion centro de funcion de pertenencia
      buf1=0;
      for k=1:N3
        buf2 = sv(j,k)*sum(ro)-sum(sv(:,k).*((P(i,:))'));
        buf1 = buf1+e(k)*buf2;
      end
      den = sum(P(i,:))^2;
      buf1 = buf1*P(i,j)/den;
      buf2 = buf1*((alfa*U(i,q1)-m(i,j))/(sigma(i,j)^2));
      m(i,j) = m(i,j)+2*lr*q(i,j)*buf2;
      % actualizacion de la varianza
      buf3 = buf1*(alfa*(U(i,q1)-m(i,j))^2)/(sigma(i,j)^3);
      sigma(i,j) = sigma(i,j)+2*lr*q(i,j)*buf3;
    end
  end
end

%% Error cuadratico

% error on-line.
e1(ent)=sqrt(sum(eseg.^2)); % salida 1
if (N3>1)
  e2(ent)=sqrt(sum(eseg2.^2)); % salida 2
end
if (N3>2)
  e3(ent)=sqrt(sum(eseg3.^2)); % salida 3
end
if (N3>3)
  e4(ent)=sqrt(sum(eseg4.^2)); % salida 4
end

% error off-line
for q1=1:Q
```

```
    ypr=simnf(U(:,q1),m,sv,sigma,alfa);
    if ypr(:,1)>=99

problema_entradas_entrenamiento=[problema_entradas_entrenamiento
seleccion(q1)];
      end
    eseg1(q1)=Y(1,q1)-ypr(:,1);
    if (N3>1)
      esegf2(q1)=Y(2,q1)-ypr(:,2);
    end
    if (N3>2)
      esegf3(q1)=Y(3,q1)-ypr(:,3);
    end
    if (N3>3)
      esegf4(q1)=Y(4,q1)-ypr(:,4);
    end
  end
  eseg1t(ent)=sumsqr(eseg1);
  if (N3>1)
    esegf2t(ent)=sumsqr(esegf2);
  end
  if (N3>2)
    esegf3t(ent)=sumsqr(esegf3);
  end
  if (N3>3)
    esegf4t(ent)=sumsqr(esegf4);
  end

  % error off-line de datos de generalizacion
  [Rgeneralizacion,Qgeneralizacion]=size(Ugeneralizacion);
  [Sgeneralizacion,Q1generalizacion]=size(Ygeneralizacion);
  problema_entradas=[];
  for q1=1:Qgeneralizacion
    ypr=simnf(Ugeneralizacion(:,q1),m,sv,sigma,alfa);
    eseg1generalizacion(q1)=Ygeneralizacion(1,q1)-ypr(:,1);
    % se toma nota de las entradas de test que dan problema al aparece
    % divisiones por cero.
    if N3>1
      decision=FALSE;
      for k=1:N3
        decision1=ypr(:,k)>=99;
```

```
        decision=decision||decision1;
      end
      if decision
        problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
      end
    else
      if ypr(:,1)>=99
        problema_entradas=[problema_entradas
seleccion_generalizacion(q1)];
      end
    end
    if (N3>1)
      esegf2generalizacion(q1)=Ygeneralizacion(2,q1)-ypr(:,2);
    end
    if (N3>2)
      esegf3generalizacion(q1)=Ygeneralizacion(3,q1)-ypr(:,3);
    end
    if (N3>3)
      esegf4generalizacion(q1)=Ygeneralizacion(4,q1)-ypr(:,4);
    end
  end

  eseg1tgeneralizacion(ent)=sumsqr(eseg1generalizacion);
  if (N3>1)
    esegf2tgeneralizacion(ent)=sumsqr(esegf2generalizacion);
  end
  if (N3>2)
    esegf3tgeneralizacion(ent)=sumsqr(esegf3generalizacion);
  end
  if (N3>3)
    esegf4tgeneralizacion(ent)=sumsqr(esegf4generalizacion);
  end

  % visualizando errores off-line.
  if rem(ent,10)==0
    if isempty(problema_entradas)
      fprintf('Salida 1 epo= %.0f, er= %g er_gen=%g
\n',ent,eseg1t(ent),eseg1tgeneralizacion(ent))
    else
      dimension=max(size(problema_entradas));
```

```
      fprintf('Salida 1 epo= %.0f, er= %g er_gen=%g
patrones_error=',ent,eseg1t(ent),eseg1tgeneralizacion(ent))
      for pro=1:dimension
        fprintf('%i',problema_entradas(pro));
      end
      fprintf('\n')
    end
    if (N3>1)
      fprintf('Salida 2 epo= %.0f, er= %g er_gen=%g
\n',ent,esegf2t(ent),esegf2tgeneralizacion(ent))
    end
    if (N3>2)
      fprintf('Salida 3 epo= %.0f, er= %g er_gen=%g
\n',ent,esegf3t(ent),esegf3tgeneralizacion(ent))
    end
    if (N3>3)
      fprintf('Salida 4 epo= %.0f, er= %g er_gen=%g
\n',ent,esegf4t(ent),esegf4tgeneralizacion(ent))
    end
  end


  if rem(ent,numero_guarda)==0
    a11=['Salida 1 sum squared error(acabado
entrenamiento)=',num2str(eseg1t(ent))];
    if (N3>1)
      a32=['Salida 2 sum squared error1(acabado
entrenamiento)=',num2str(esegf2t(ent))];
    end
    if (N3>2)
      a33=['Salida 3 sum squared error1(acabado
entrenamiento)=',num2str(esegf3(ent))];
    end
    if (N3>3)
      a34=['Salida 2 sum squared error1(acabado
entrenamiento)=',num2str(esegf4(ent))];
    end


    a3=[' --- Kohonen self-organizing feature map ----'];
    a4=[' '];
    a5=['N1=',num2str(N1),' % numero de entradas'];
    a6=['N2=',num2str(N2),' % numero de reglas fuzzy'];
    a7=['N3=',num2str(N3),' % numero de salidas.'];
    a8=['numero de epocas=',num2str(ent)];
```

```
    a9=['frecuencia de presentacion=',num2str(mf)];
    a10=[' velocidad de aprendizaje inicial=',num2str(lr1)];
    a11=['alfa=',num2str(alfa),' % alfa que multiplica a U'];
    a12=['sigma0=',num2str(sigma0),' % sigma en funcion de vecindad'];
    a13=[' '];
    a14=[' --- Rule-extracting phase ---'];
    a15=[' '];


    a17=[' '];
    a18=['----   LMS algorithm sobre ----'];
    a19=[' '];
    a20=['numero de epocas= ',num2str(ent)];
    a21=['lr=',num2str(lr)];
    a22=[' m sv sigma son los resultados del entrenamiento'];

readme=strvcat(a11,a21,a22,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a1
5,...
                  a17,a18,a19,a20,a21,a22);
    if (N3>3)
      readme=strvcat(a34,readme);
      clear a34
    end
    if (N3>2)
      readme=strvcat(a33,readme);
      clear a33;
    end
    if (N3>1)
      readme=strvcat(a32,readme);
      clear a32;
    end
    readme=strvcat(a11,readme);
    clear a11 a21 a22  a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15
    clear a16 a17 a18 a19 a20 a21 a22
    clear eseg eseg2 eseg3 eseg1 esegf2 esegf3 esegf4
    readme
    if ~exist('mfinal')
      mfinal=ent;
    end
    save -v4 result_bakup.mat;
  end
end
```

```
%% Error cuadratico al final del entrenamiento
a31=['Salida 1 sum squared error(acabado
entrenamiento)=',num2str(eseg1t(ent))];
if (N3>1)
  a32=['Salida 1 sum squared error generalizacion (acabado
entrenamiento)=',num2str(eseg1tgeneralizacion(ent))];
end
if (N3>2)
  a33=['Salida 3 sum squared error1(acabado
entrenamiento)=',num2str(esegf3(ent))];
end
if (N3>3)
  a34=['Salida 2 sum squared error1(acabado
entrenamiento)=',num2str(esegf4(ent))];
end


% Mostrando el error cuadratico medio de cada salida;
figure(1);
plot(eseg1t / size(U, 2));
xlabel('epocas');
ylabel('error cuadratico de salida 1');


title(['error cuadratico de salida para patrones de entrenamiento para
' fichero_datos]);
figure(2);
plot(eseg1tgeneralizacion / size(Ugeneralizacion, 2) ,'--');
xlabel('epocas');
ylabel('error cuadratico de salida 1')
if ~isempty(problema_entradas)
      dimension=max(size(problema_entradas));
  cadena=['error patrones de test problema patrones: '];
  for pro=1:dimension
    cadena=[ cadena num2str(problema_entradas(pro))];
  end
  title(cadena);
else
  title(['error patrones de test para ' fichero_datos]);
end


if (N3>1)
  figure(3);
```

```
  plot(esegf2t);
  xlabel('epocas');
  ylabel('error cuadratico de salida 2');
  title('error cuadratico de salida para patrones de entrenamiento');

  figure(4);
  plot(esegf2tgeneralizacion,'--');
  xlabel('epocas');
  ylabel('error cuadratico de salida 1')
  if ~isempty(problema_entradas)
    dimension=max(size(problema_entradas));
    cadena=['error patrones de test problema patrones: '];
    for pro=1:dimension
      cadena=[ cadena num2str(problema_entradas(pro))];
    end
    title(cadena);
  else
    title('error patrones de test');
  end
end

if (N3>2)
  figure(5);
  plot(esegf3t);
  xlabel('epocas');
  ylabel('error cuadratico de salida 3');
  title('error cuadratico de salida para patrones de entrenamiento');
  % hold
  % plot(esegf3tgeneralizacion,'--');
  % title('error cuadratico de salida: - (patrones de entrenamiento) -
- (patrones de test)');
end

if (N3>3)
  figure(6);
  plot(esegf4t);
  xlabel('epocas');
  ylabel('error cuadratico de salida 4');
  title('error cuadratico de salida para patrones de entrenamiento');
  % hold
  % plot(esegf4tgeneralizacion,'--');
```

```
  % title('error cuadratico de salida: - (patrones de entrenamiento) -
- (patrones de test)');
end


a3=[' --- Kohonen self-organizing feature map ----'];
a4=[' '];
a5=['N1=',num2str(N1),' % numero de entradas'];
a6=['N2=',num2str(N2),' % numero de reglas fuzzy'];
a7=['N3=',num2str(N3),' % numero de salidas.'];
a8=['numero de epocas=',num2str(me1)];
a9=['frecuencia de presentacion=',num2str(mf)];
a10=[' velocidad de aprendizaje inicial=',num2str(lr1)];
a11=['alfa=',num2str(alfa),' % alfa que multiplica a U'];
a12=['sigma0=',num2str(sigma0),' % sigma en funcion de vecindad'];
a13=[' '];
a14=[' --- Rule-extracting phase ---'];
a15=[' '];
a17=[' '];
a18=['----   LMS algorithm sobre ----'];
a19=[' '];
a20=['numero de epocas= ',num2str(me)];
a21=['lr=',num2str(lr)];
a22=[' m sv sigma son los resultados del entrenamiento'];
readme=strvcat(a11,a21,a22,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a1
5,...
               a17,a18,a19,a20,a21,a22);
if (N3>3)
  readme=strvcat(a34,readme);
  clear a34
end
if (N3>2)
  readme=strvcat(a33,readme);
  clear a33;
end
if (N3>1)
  readme=strvcat(a32,readme);
  clear a32;
end
  readme=strvcat(a31,readme);

clear a11 a21 a22  a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15
```

```
clear a16 a17 a18 a19 a20 a21 a22
clear eseg eseg2 eseg3 eseg1 esegf2 esegf3 esegf4

readme
```

9.3.10 selection.m

```
% asignación de entradas y salidas seleccionadas para entrenamiento

for indice1=1:max(size(seleccion))
    U(:,indice1)=Unuevo(:,seleccion(indice1));
    Y(:,indice1)=Ynuevo(:,seleccion(indice1));
End
```

9.3.11 selectionGen.m

```
  % se ha adaptado vector generalizacion

for indice1=1:max(size(seleccion_generalizacion))

Ugeneralizacion(:,indice1)=Unew_generalizacion(:,seleccion_generalizac
ion(indice1));

Ygeneralizacion(:,indice1)=Ynew_generalizacion(:,seleccion_generalizac
ion(indice1));

  End
```

9.3.12 simnf.m

```
function [sy]=simnf(U,m,sv,sigma,alfa)



% function [sy]=simnf(U,m,sv,sigma,alfa)
%
%  sistema neuro-fuzzy ya entrenado. Extrae valores para un caso
%  concreto.
%
% presentation phase

  [N2,N3]=size(sv);
  [N1,N2]=size(sigma);
  [N1,N2]=size(m);
```

```
  for i=1:N1
   for j=1:N2
    P(i,j)=exp(-((alfa*U(i)-m(i,j))^2)/(sigma(i,j)^2));
   end
  end

  for j=1:N2
   ro(j)=min(P(:,j));
  end

  for k=1:N3
   den=sum(ro);
   if den==0
       sy(k)=1000;
   else
     sy(k)=sum(sv(:,k).*ro')/sum(ro);
   end

  end
```

9.3.13 simnf_so.m

```
function [sy]=simnf_so(U,m,sv,sigma,alfa)


% function [sy]=simnf(U,m,sv,sigma,alfa)
%
%  sistema neuro-fuzzy ya entrenado. Extrae valores para un caso
%  concreto.
%
% presentation phase

  [N2,N3]=size(sv);
  [N1,N2]=size(sigma);
  [N1,N2]=size(m);



  for i=1:N1
   for j=1:N2
```

```
   P(i,j)=exp(-((alfa*(U(i)-m(i,j)))^2)/(sigma(i,j)^2));
 end
end

for j=1:N2
 ro(j)=min(P(:,j));
end

for k=1:N3
 sy(k)=sum(sv(:,k).*ro')/sum(ro);
end
```

10 BIBLIOGRAFÍA

[ABD06] W. Abdel-Hamid, T. Abdelazim, N. El-Sheimy y G. Lachapelle. Improvement of MEMS-IMU/GPS performance using fuzzy modeling. GPS Solutions, Volume 10, Number 1, 1-11, DOI: 10.1007/s10291-005-0146-6

[ALI04] Alison K. Brown and Yan Lu, *NAVSYS Corporation.* Performance Test Results of an Integrated GPS/MEMS Inertial Navigation Package. Proceedings of ION GNSS 2004, Long Beach, California, September 2004

[AMI99] Amidi, O., Kanade, T., and Fujita, K. (1999). A visual odometer for autonomous helicopter flight. Journal of Robotics and Autonomous Systems, 28:185-193.

[AND06] Andrew K.S. Jardine, Daming Lin, Dragan Banjevic, A review on machinery diagnostics and prognostics implementing condition-based maintenance, Mechanical Systems and Signal Processing, Volume 20, Issue 7, October 2006, Pages 1483-1510, ISSN 0888-3270

[AZA95] A. Azarbayejani, A. Pentland, (1995) Recursive estimation of motion, structure, and focal length, IEEE Trans. Pattern Analysis and Machine Intelligence 17 (6) 562-575.

[BAR94] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. 1994. Performance of Optical Flow Techniques. The International Journal of Computer Vision 12(1):43–77.

[BEV06] D. Bevly, J. Ryu, and J. C. Gerdes, "Integrating INS sensors with GPS measurements for continuous estimation of vehicle sideslip, roll, and tire cornering stiffness," IEEE Trans. Intell. Transp. Syst., vol. 7, no. 4, pp. 483–493, Dec. 2006.

[BOU99] Bouguet, Jean-Yves 1999. Pyramidal Implementation of the Lucas-Kanade Feature Tracker, OpenCV Documentation, Microprocessor Research Labs, Intel Corporation

[BRA01] S.G. Braun, D.J. Ewins, S.S. Rao, Encyclopedia of Vibration, Academic Press, New York, 2001.

[BRO88] Broomhead, D. S. ; Lowe, David. "Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks", Memorandum report. Royal Signales and Radar Establishment Malvern (United Kingdom)(1988)

[BUC93] J. J. Buckley, Y. Hayashi y E. Czogala, "On the equivalence of neural nets and fuzzy expert systems", Fuzzy Sets Syst., vol. 53, no. 2, pp. 129-134, 1993.

[BUC99] J.J. Buckley and T. Feuring, "Fuzzy and Neural: Interactions and Applications", ser. Studies in Fuzziness and Soft Computing. Heidelberg, Germany: Physica-Verlag, 1999.

[CAM04] Campbell, J.; Sukthankar, R.; and Nourbakhsh, I. 2004. "Visual odometry using commodity optical flow". Proceedings of the American Association of Artificial Intelligence (AAAI).

[CAS11] O. Casanova, G.N. Marichal, J.L. González Mora , A. Hernández. "A low cost intelligent localization system based on a camera and an inertial measurement unit (IMU)." In Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-2011) and "les rencontres francophones sur la Logique Floue et ses Applications" (LFA-2011), 2011, Aix-les-Bains, France. ISBN: 978-90-78677-00-0

[CHA01] Chang F.-J.; Chen Y.-C. "A counterpropagation fuzzy-neural network modeling approach to real time streamflow prediction" Journal of Hydrology, vol. 245, pp. 153-164(12), 2001

[CHE96] Chen C.H., (1996) ." Fuzzy Logic and Neural Networks Handbook", McGraw-Hill.S.

[CHR03] Christopher Hide, Terry Moore and Martin Smith (2003). Adaptive Kalman Filtering for Low-cost INS/GPS. Journal of Navigation, 56, pp 143-152

[FOX05] Foxlin, E., November 2005. Pedestrian tracking with shoemounted inertial sensors. IEEE Computer Graphics and Applications 25, 38–46.

[GRE02] R. Gregor, M. Lutzeler, M. Pellkofer, K. H. Siedersberger, and E. D. Dickmanns, "EMS-Vision: A perceptual system for autonomous vehicles," IEEE Trans. Intell. Transp. Syst., vol. 3, no. 1, pp. 48–59, Mar. 2002.

[HAY94] Y. Hayashi y J. J. Buckley, "Aproximation between fuzzy expert systems and neural networks", Int. J. Approx. Reas., vol. 10, pp. 63-73, 1994

[HID10] Hide, C.; Botterill, T.; Andreotti, M. Low cost vision-aided IMU for pedestrian navigation. Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS) 2010. ISBN 978-1-4244-7880-4

[JAN93a] J. S. R. Jang and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems", IEEE Transactions on Neural Networks, vol. 4, pp. 156-159, 1993

[JAN93b] J.S.R. Jang "ANFIS: Adaptive-Network-Based Fuzzy Inference System", IEEE Trans. Systems, Man, Cybernetics, 23(5/6):665-685, 1993.

[KAL60] Kalman, R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems," Transaction of the ASME-Journal of Basic Engineering, pp. 35-45 (March 1960).

[KIN00] Volodymyr V. Kindratenko. A survey of electromagnetic position tracker calibration techniques. *Virtual Reality: Research, Development and Applications*, 5(3):169-182, 2000

[KOH89] T. Kohonen. Self-organization and associative memory. Springer Verlag, New York, 1989.

[LKN81] Lucas, B. D., and Kanade, T. 1981. An iterative image registration technique with an application to stereo vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81), 674–679.

[MAI07] Maimone, M.; Cheng, Y.; Matthies, L. (2007). "Two years of Visual Odometry on the Mars Exploration Rovers". *Journal of Field Robotics* 24 (3): 169–186.

[MAR01] G.N. Marichal, L. Acosta, L. Moreno, J.A. Méndez, J.J. Rodrigo, M. Sigut. "Obstacle Avoidance for a Mobile Robot: A Neuro Fuzzy Approach". Fuzzy Sets and Systems, Volumen: (124). 171-179, 2001

[MAR11] Marichal, G. N., Artés Mariano, García Prada J. C.,Casanova O. Extraction of rules for faulty bearing classification by a Neuro-Fuzzy approach. Mechanical Systems and Signal Processing, Volume 25, Issue 6, p. 2073-2082.

[MDS] http://msdn.microsoft.com/en-us/library/dd375454%28v=vs.85%29.aspx

[MIT00] Sushita Mitra and Y. Hayashi, "Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework", IEEE Transactions on Neural Networks, vol 11, pp 748, 2000

[MON10] Iván F. Mondragón, Pascual Campoy, Miguel Olivares, Carol Martínez, Luis Mejias. 2010 Autonomous Robots, Volume 29, Number 1, 17-34, DOI: 10.1007/s10514-010-9183-2

[MURA05] L. Muratet, S. Doncieux, Y. Briere, J.-A. Meyer, (2005) "A contribution to vision-based autonomous helicopter flight in urban environments", Robotics and Autonomous Systems (Elsevier) 50 (4) 195-209.

[NIE95] J. Nie, “Constructing fuzzy model by self-organizing counterpropagation network” IEEE Transactions on Systems, Man, and Cybernetics, vol. 25, pp. 963-970, 1995.

[OCV] http://opencv.willowgarage.com/wiki

[OLI07] Woodman Oliver J. An introduction to inertial navigation. Technical Report. - Cambridge: University of Cambridge, 2007. - ISSN 1476-2986.

[PAU05] Paul D. Samuel, Darryll J. Pines, A review of vibration-based techniques for helicopter transmission diagnostics, Journal of Sound and Vibration, Volume 282, Issues 1-2, 6 April 2005, Pages 475-508, ISSN 0022-460X

[ROD10] Antonio Francisco Rodriguez-Hernandez, Carlos Merino, Oscar Casanova, Cristian Modrono, Miguel Angel Torres, Raquel Montserrat, Gorka Navarrete, Enrique Burunat, Jose Luis Gonzalez-Mora. Sensory Substitution for Visually Disabled People: Computer Solutions. WSEAS TRANSACTIONS on BIOLOGY and BIOMEDICINE. Issue 1, Volume 7, p.p. 1 -10, January 2010.

[SHE04] Y.-T. Sheen, C.-K. Hung, “Constructing a wavelet-based envelope function for vibration signal analysis”, Mechanical Systems and Signal Processing 18 (1) (2004) 119–126

[SHE07] Yuh-Tay Sheen, “An analysis method for the vibration signal with amplitude modulation in a bearing system”, Journal of Sound and Vibration 303 (3–5) (2007)

[TAN99] N. Tandon, A. Choudhury, “A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings”, Tribology International 32 (1999) 469–480

[TBB] http://threadingbuildingblocks.org/

[TOR10] Torres-Gil, M. A., Casanova-Gonzalez, O., Gonzalez-Mora, J. L. “Applications of Virtual Reality for Visually Impaired People” WSEAS TRANSACTIONS on COMPUTERS, ISSN: 1109-2750, iss. 2, vol 9, pp. 184-193, 2010

[VIL] http://muonics.net/school/spring05/videoInput

[WEI90] M. Wei and K. P. Schwarz, “Testing a decentralized filter for GPS/INS integration,” in Proc. IEEE Plans Position Location Navig. Symp., Mar. 1990, pp. 429–435.

[WEL06] G. Welch, G. Bishop, “An introduction to the Kalman Filter”, UNC-Chapel Hill, TR 95-041, July 24, 2006.

[ZHA03] S. Zhang, T. Asakura, X.L. Xu, B.J. Xu, “Fault diagnosis system for rotary machine based on fuzzy neural networks”, JSME International Journal. Series C: Mechanical Systems, Machine Elements and Manufacturing (2003) 203546 (2003) 2035