MÁSTER DE AUTOMÁTICA
E INFORMÁTICA INDUSTRIAL
2012 – 2013

Nuevas tecnologías de manipulación usando sensorización integrada

AUTOR: Jesús Moreno Ceca
DIRECTOR: Martín Mellado Arteche

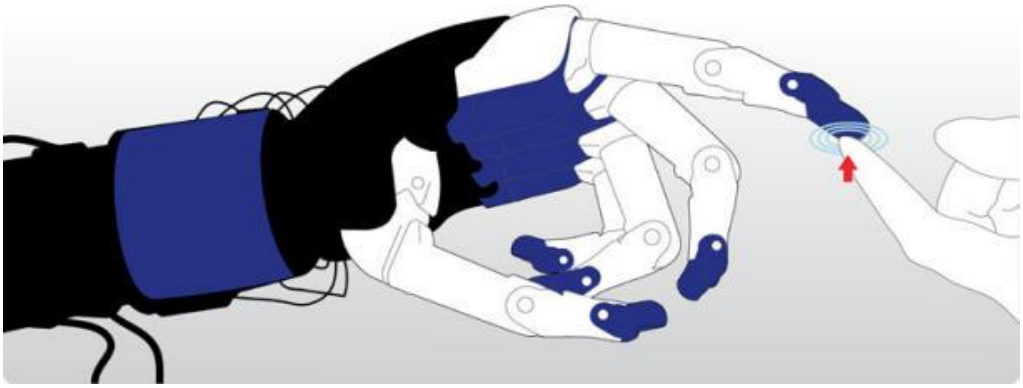
INTRODUCCIÓN Y OBJETIVOS

Actualmente, existe un tipo de sensor muy desconocido con el que se intentan simular las sensaciones humanas que provoca el sentido del tacto. Esta familia de sensores está siendo actualmente un tema de estudio en pleno auge.

Estos sensores, permitirán a las máquinas conocer en todo momento, dónde están tocando y con qué presión, por lo que a partir de ahora, podrían aparecer robots capaces de saber si alguien les está haciendo cosquillas, acariciando o dando una palmadita en la espalda. Además, algunos investigadores añaden sensores de temperatura, de forma que el robot también sea capaz de medir la temperatura del material que está tocando, para conseguir con ello además, una imitación más cercana al sentido humano.

Existen estudios activos en lo referente a la fabricación de estos sensores, como ejemplo, unos investigadores de Tokio, han elaborado una piel artificial que mide tanto presión como temperatura. Para ello, han utilizado circuitos electrónicos para como sensores de presión, y semiconductores como sensores de temperatura. Han empotrado estos sensores entre dos capas finas de plástico para crear una matriz en forma de red.

Por su parte, Peratech (www.peratech.com), está utilizando su material llamado QTC (Quantum Tunnelling Composite) para crear pieles sensitivas al tacto, para máquinas inteligentes del MIT (Massachusetts Institute of Technology). Este material es un material que tiene propiedades flexibles, muy económico, y conductor eléctrico, por lo que puede ofrecer a los robots nuevas formas de interactuar con el entorno que lo rodea.



En el ámbito industrial, este tipo de sensores también ofrece unas ventajas que muchos otros no pueden ofrecer. Leyendo en la bibliografía, existen diferentes aplicaciones en las que se está trabajando.

En lo que al marco de este proyecto, esta tesina sirve como introducción para un proyecto llamado "*Nuevas tecnologías de manipulación utilizando sensorización integrada para la estimación de propiedades y determinación automática de la calidad y sanidad de la producción agroalimentaria en líneas de inspección y manipulación* (MANI-DACSA)". Se trata de un Proyecto de Investigación Fundamental Orientada. INIA (Ref. RTA2012-00062-C04-02), financiado por el ministerio de economía y competitividad.

Las entidades participantes en este proyecto, son: ai2 de la UPV, en proyecto coordinado con IVIA, UMH y LabHuman de la UPV. Con una duración de tres años (desde Mayo de 2013 hasta Abril de 2016). Con Martín Mellado como investigador responsable.

ESTADO DEL ARTE

1 - SENSORES TÁCTILES

A continuación se describirán las modalidades de sensores más utilizadas en tareas de manipulación y agarre especial énfasis en sensores de contacto y tecnologías comercialmente disponibles, mencionando algunas opciones sin contacto.

En el proceso de manipulación, existen tres sentidos utilizados: vista, tacto y oído. El sentido de la vista ha sido ampliamente estudiado por los científicos, sin embargo, la visión para la manipulación es una pequeña parte de lo que se ha hecho en este campo. La visión es la señal más adecuada para localizar objetos en el campo de trabajo e identificar obstáculos a su alrededor. También puede ser utilizada para detectar contactos y para seguir la relación entre la mano y el objeto manipulado. Sin embargo, la visión no es 100% apropiada debido a errores como pueden ser los de calibración.

Para solucionar los problemas de precisión ofrecidos por el sentido de la vista, se pueden utilizar otros sentidos, como puede ser el sentido del tacto. En sistemas robóticos, el sentido del tacto es implementado utilizando tres tipos de sensores: táctiles, fuerza y cinestésicos. La información táctil está dedicada a características locales, mientras que la información visual para la interpretación del entorno global. La escucha es una característica complementaria que puede facilitar la detección de contactos aumentando la posibilidad del éxito tras la escucha del contacto. También puede ser utilizado para deducir el material del objeto y para adivinar la zona de contacto.

2 - TIPOS DE SENSORES TÁCTILES

Además de la interacción física con el objeto, existe otra clasificación de los diferentes tipos de sensores utilizados:

A. SENSORES SIN CONTACTO

Esta categoría la componen los sensores que no necesitan un contacto entre el sensor y el objeto para detectar un contacto con el mismo. Lo componen principalmente los sensores de visión y de audición, sin embargo existen otros tipos de sensores que pertenecen a esta categoría, como son los sensores IR, que también han sido utilizados para tareas de agarre y para dar algo de información antes de realizar la tarea de agarre, pero no estrictamente para detectar contactos.

Aunque existe una amplia variedad de tipos de sensores de visión, todos ellos comparten un elemento común: una cámara. Hay mucha información de los objetos y entorno que pueden extraerse a través de sensores visuales. Sin embargo, la información que es más relevante para el agarre y la manipulación está relacionada con la localización de objetos y obstáculos, y su estructura tridimensional. El enfoque clásico y común para obtener la información 3D del entorno es el sistema de reconstrucción estéreo. Este método se basa en la diferencia entre las imágenes de dos cámaras para extraer información de profundidad. De hecho, muchos de los robots humanoides con la habilidad de coger objetos, implementan un sistema de visión estéreo en su cabeza. Por otra parte, existen muchos sistemas de visión estéreo que implementan todos los algoritmos de reconstrucción en hardware.

Otra posibilidad es proyectar un patrón a la escena y observar su deformación para obtener una estructura en tres dimensiones. Este método es utilizado por el sensor de la Kinect. Desde su lanzamiento en Noviembre del 2010, su impacto en la comunidad de robótica, ha sido muy alta. Debido a su bajo precio, alto rendimiento, precisión y velocidad, muchos investigadores dedicados a la robótica han encontrado en la Kinect un método para facilitar la obtención de información 3D de la escena.

Existen otros sensores de rango que utilizan el láser, como los escáneres láser LIDAR, o cámaras de tiempo de vuelo. Estas alternativas tienen sentido utilizarlas en aplicaciones cuyas necesidades no pueden solucionarse por el sensor de la Kinect, como pueden ser entornos exteriores, bajo el agua, etc.

Los sensores de sonido apenas son utilizados por la comunidad robótica, pero pueden ser utilizados para detectar contactos, y el instante de tiempo en el que se han producido.

Es posible utilizar sensores visuales para determinar la calidad de agarre, pero no es una tarea sencilla. Una posible idea de detectar la estabilidad de agarre a partir de información visual podría ser seguir el objeto y la mano al mismo tiempo mientras se coge el objeto y detectar diferencias entre ambos movimientos.

B. SENSORES DE CONTACTO

Actualmente existen muchos estudios que intentan imita el sentido del tacto humano utilizando distintas tecnologías táctiles. Desafortunadamente, cada tipo de sensor está centrado en una aplicación particular, y no es posible lograr la versatilidad de los receptores humanos.

Es importante mencionar que los sensores táctiles, y los sensores de contacto se refieren a términos distintos. Por un lado, los sensores táctiles incluyen investigaciones de sensores que imiten la piel humana para medir distribuciones de presión, temperaturas, rugosidad, etc. Mientras que los sensores de contacto se refieren a aquellos que miden la fuerza generada a través de los puntos de contacto durante la manipulación. Ambas modalidades pertenecen al sentido del tacto.

Algunos sensores táctiles son capaces de medir ambos tipos de información, tanto táctil como de contacto, pero normalmente, cubren una pequeña superficie de la mano, como los extremos de los dedos, o la palma de la mano. De ser así, si el contacto con el objeto es ejercido en una parte de la mano que no se encuentra cubierta por los sensores, no existe información disponible. Sin embargo, un sensor de fuerza situado en la muñeca del robot, será capaz de dar información de la fuerza generada por los contactos en cualquier parte de la mano, asegurando que la detección de un contacto estará siempre disponible.

En la literatura de los robots, la realimentación táctil ha sido utilizada principalmente para manipulación dirigida por eventos, que se ocupa de cambiar entre diferentes tareas de acuerdo a una serie de condiciones. De todos los eventos que pueden ser detectados con sensores táctiles, es muy interesante mencionar los siguientes: percibir un contacto, un deslizamiento, tipo de contacto, cambios en las propiedades del objeto, etc.

La realimentación de fuerza representa una necesidad fundamental para el éxito de cualquier tarea que implique alguna interacción con el exterior, pero cuando se intenta utilizar para evaluar la calidad de un agarre, la información de fuerza proporcionada por un sensor en la muñeca no es muy importante. Por otro lado, los sensores de fuerza pueden ser utilizados para detectar posiciones de contacto si son situados en cada dedo, para así utilizar esa información de contacto para evaluar la estabilidad del agarre.

3 - SENSORES COMERCIALES

El despliegue de los robots en entornos no estructurados, requiere una adaptación ante eventos inesperados además de una administración de la incertidumbre de los entornos reales. Esto ha conducido a los diferentes tipos de sensores a ser totalmente necesarios en cualquier plataforma robótica que esté diseñada para entornos reales. A lo largo de la última década algunas compañías han sido creadas para trabajar en el negocio de los sensores táctiles. Estos sensores comerciales, permiten a los grupos de investigadores en robótica de todo el mundo implementar un sistema de sensor táctil en su plataforma actual sin la necesidad de diseñar e implementar sus propios sistemas de sensores. Existen diferentes tecnologías utilizadas para diseñar sensores táctiles, cada una con sus propias ventajas e inconvenientes.

Los sensores táctiles comerciales son diseñados normalmente como matrices de celdas con sensores llamadas “tactels”. Cada tactel es la unidad sensora y es sensitiva a la presión. La mayoría de las veces, los fabricantes tienen algunos tamaños definidos para sus sensores, pero a menudo, es posible pedir un sensor de tamaño determinado para que encaje en la aplicación que se esté desarrollando.

Las principales características de los sensores táctiles son:

1. Sensibilidad. La variación más pequeña de presión que puede ser detectada por el sensor.
2. Repetibilidad. La variación del valor que el sensor ha calculado para la misma presión aplicada.
3. Velocidad de comunicación. La velocidad a la que el sensor manda los valores.
4. Rango de presión. El rango de posibles valores que pueden medirse.
5. Resolución espacial. La densidad de tactels a lo largo de la superficie del sensor.

El material y la tecnología utilizados para construir sensores táctiles pueden influir en otras características que son muy comunes en este tipo de sensores: histéresis y filtrado paso bajo. Un ciclo de histéresis hace posible dos valores diferentes para la misma presión, dependiendo si la presión está aumentando o disminuyendo. La ocurrencia de este efecto es principalmente determinado por la tecnología utilizada para crear los sensores. Por otro lado, el efecto de filtro paso bajo es causado principalmente por el material utilizado para cubrir los sensores. El efecto de filtro paso bajo actúa como una operación de convolución sobre la presión que es generado por el sensor causando la perdida de detalles y distorsionando los datos.

Estos dos problemas se deben tener en cuenta antes de elegir un sensor para determinada aplicación. Por ejemplo, si los sensores táctiles van a ser utilizados para detectar detalles y texturas, el efecto de filtrado que aparece en la mayoría de sensores podría causar la pérdida de los detalles de la superficie. Por otro lado, si la aplicación está pensada para mejorar, detecta y realizar buenos agarres sin tener en cuenta detalles de la superficie a coger, un material suave en los extremos de los dedos, puede ayudar a obtener mejores resultados. Para la mejora y evaluación de agarres, la característica más importante que debe ofrecer el sensor es la sensibilidad. Si se dispone de un sensor con muy buena sensibilidad, se podrán detectar colisiones con pequeños objetos y así, actuar en consecuencia.

En la actualidad, existen muchas tecnologías disponibles para la fabricación de estos sensores: resistivos, capacitivos, piezoeléctricos, ultrasónicos, ópticos, etc. Para la mayoría de las tecnologías, existe un fabricante que distribuye una versión comercial de los mismos.

A. SENSORES RESISTIVOS

La tecnología resistiva es la solución más comúnmente utilizada. Generalmente, es muy barata, sin embargo, ofrece una precisión más baja de la que pueden ofrecer otras tecnologías. Esta comercialmente disponible por Interlink, y ha sido utilizada en sensores táctiles experimentales debido a su bajo precio, ruido y buena sensibilidad. El principal inconveniente del sensor de Interlink solamente hay una unidad sensitiva. Si la aplicación deseada necesita una matriz de sensores cubriendo la superficie, se necesita un montaje manual.

Estos sensores fueron utilizados en las primeras manos táctiles, como la mano DLR de tres dedos. La empresa Weiss Robotics, ha diseñado una matriz resistiva de 6x14 sensores con una resolución espacial de 3.4mm. Estos sensores son cubiertos y protegidos por un polímero que además, incrementa la fricción con los objetos. Por otro lado, este polímero actua como un filtro paso bajo, que esconde los posibles detales, como la textura, de las superficies en contacto. Los sensores de esta empresa, son los utilizados en la mano Schunk Dextrous.

B. SENSORES QTC

En este tipo de sensores, en vez de tener una resistencia sensitiva a la presión, lo que se tiene es un polímero que cambia gradualmente de ser un perfecto aislante, a ser un total conductor. Las mayores ventajas de estos sensores son que permiten más flexibilidad a la hora de diseñar la geometría del sensor y ofrecen una mejor sensibilidad. Esta solución táctil es implementada en la mano de Shadow Robotics, llamada Shadow Dextrous Hand. La resolución espacial es muy similar a otros sistemas táctiles.

C. SENSORES CAPACITIVOS

Con respecto a la tecnología capacitiva, las soluciones táctiles de Pressure Profile Systems son los sensores de presión comerciales más relevantes. Aunque que la tecnología capacitiva permite más densidad de tactels, las matrices de sensores de esta empresa ofrecen una resolución espacial similar a los sensores resistivos. Los sensores de esta empresa están integrados en la manos Barrett Hand, Twendy One y PR2. Los sensores capacitivos son muy sensibles, pero sufren una histéresis muy grande.

D. SENSORES ÓPTICOS

En el pasado, el sistema táctil más importante utilizando sistemas ópticos era Kinotex de Tactex. Desafortunadamente, en Noviembre del año 2008, la compañía se quedó en bancarrota y la producción de detuvo. Actualmente, la solución comercial para estos sistemas es Skilsens de Fastenica. Estos sistemas son muy sensitivos, flexibles y rápidos, pero también muy voluminosos. El tamaño es la principal razón que hace muy difícil utilizar estos sensores en los robots.

E. OTROS SENSORES TÁCTILES

En un diseño innovador, la empresa Syntouch ha diseñado un sensor multi-modal táctil con forma de dedo. Este sensor puede detectar la situación del contacto y la dirección de la fuerza utilizando electrodos. También puede detectar microvibraciones y temperaturas. El sensor está formado por un núcleo rígido al que se sujetan todos los sensores, y por un líquido conductor cubierto por una piel elástica. Cualquier cambio en la piel elástica, produce el desplazamiento del líquido, que puede ser detectado mediante unos electrodos para deducir la dirección de la fuerza y la localización del contacto.

4 - APLICACIONES

Actualmente, uno de los campos con más estudio, en lo que a este tipo de sensores se refiere, es la idea de dotar a los robots humanoides el sentido del tacto. Las nuevas capacidades, y un sistema de producción para construir sensores que permitan a los robots recibir el sentido del tacto, van a permitir el trabajo de los robots en condiciones que no van a tener restricciones, como sucede en la actualidad, así como su habilidad para comunicarse y cooperar, tanto entre robots, como entre robots y humanos.

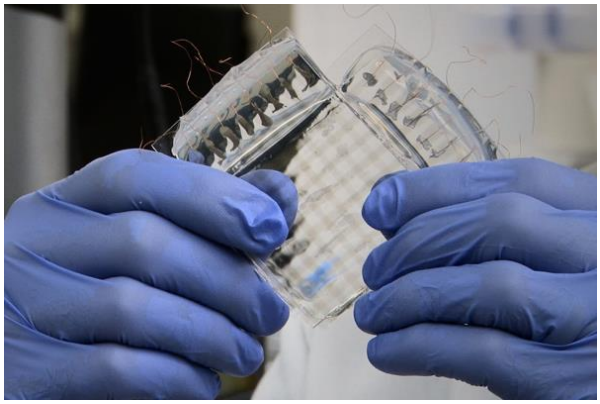
Existe un proyecto en desarrollo llamado ROBOSKIN en el que se están estudiando nuevas tecnologías de sensores y sistemas de administración que permitirán a los robots poseer un sentido del tacto artificial.

Dentro de este proyecto, se han hecho pruebas con el sentido del tacto. Se ha empezado realizando pruebas en el laboratorio para clasificar los tipos y los grados de tacto. Crearon un mapa geométrico utilizando contacto continuo entre el robot y el entorno para construir el cuerpo de la representación, parámetros con los que los datos pueden ser asimilados por el robot, para generar comportamientos.

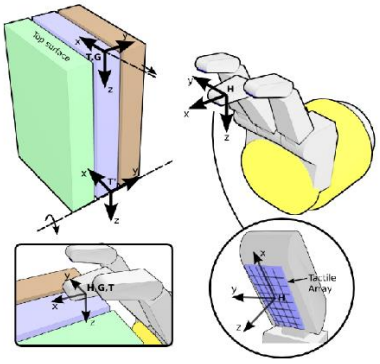
Además, este grupo, ha creado un robot llamado KASPAR, un robot humanoide que ha sido creado para ayudar a los niños autistas a comunicarse mejor.

El profesor Cannata expone: “Con nuestros sensores, el robot puede ser consciente de un contacto, y los datos pueden constituir una parte importante de la clasificación del contacto que hicimos – la distinción entre, por ejemplo, contactos deseados y no deseados.”

También, desde la universidad de Stanford, han creado una piel artificial flexible y elástica, con el objetivo de utilizarlas en prótesis sensitivas en robots, para varias aplicaciones médicas como pueden ser vendajes sensibles a la presión o en pantallas táctiles de ordenadores.



Desde la universidad de Castellón, llegan investigaciones sobre la calidad del agarre ejercido por el robot. Las pruebas, son realizadas con un brazo robótico que intenta coger un libro de una estantería que se encuentra entre más libros, para lo que se ayuda de un sensor del tipo en estudio. Su intención es coger el libro como lo haría un ser humano, utilizando un dedo de la mano para hacerlo girar sobre la estantería, para lo que se ejerce fuerza desde la parte de arriba del libro. La mano robótica que han utilizado consta de con sensores táctiles ubicados en cada dedo, formados por una matriz de 8 x 5 celdas.



5 - TÉCNICAS DE CLASIFICACIÓN, REDES NEURONALES

A. REDES NEURONALES NATURALES

Las redes neuronales se basan en el funcionamiento del sistema neuronal del cuerpo humano. En el cuerpo humano encontramos tres elementos fundamentales: los órganos receptores, el sistema nervioso y los órganos efectores. El proceso mediante el cual, el sistema humano convierte la información del exterior en una acción es la siguiente:

1. Los órganos receptores recogen la información del exterior.
2. El sistema nervioso transmite esta información, la analiza, y envía la información.
3. Los órganos efectores reciben la información del sistema nervioso y la convierten en una acción determinada.

La unidad fundamental del sistema nervioso es la neurona. Las neuronas, están compuestas por un núcleo, el axón, y de las dendritas. La información llega al núcleo de la neurona a través de las dendritas. Cuando ésta llega al núcleo, es procesada para generar una respuesta, que será propagada por el axón, para llegar a otras neuronas, formando en su conjunto lo que se conoce como red neuronal.

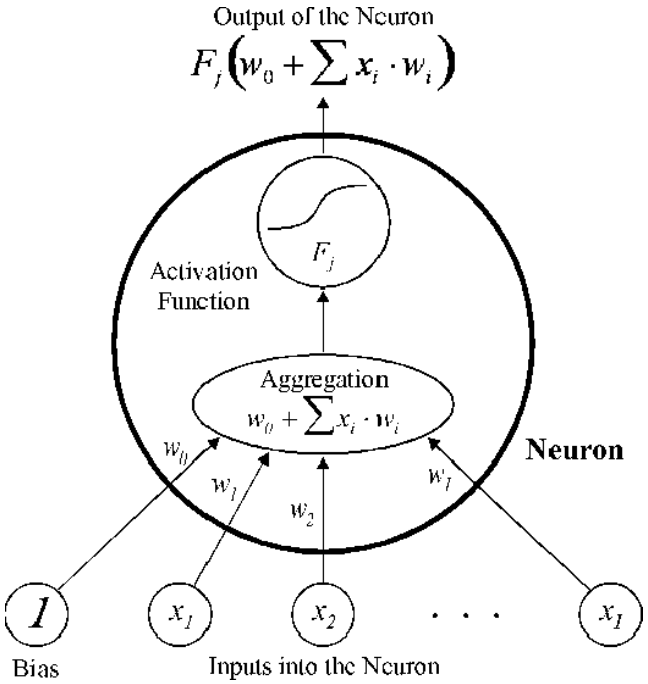
La característica que hace especialmente interesante a estas redes neuronales, es que la información que fluye desde el axón de una neurona hasta la dendrita de otra, a través de lo que se conoce como sinapsis, que no es más que una especie de espacio líquido que tiene una serie de características eléctricas que permiten cancelar o potenciar la transmisión de la información. Esta característica, está fuertemente relacionada con la habilidad humana del aprendizaje.

B. REDES NEURONALES ARTIFICIALES

Del estudio biológico de las redes neuronales, surge la idea de imitar este comportamiento en un computador. Del mismo modo que en las redes neuronales naturales, la unidad principal se llama neurona, y en este caso, tienen unas características que las hacen comportarse de forma similar a las neuronas naturales, estando conectadas entre sí para formar una red.

A cada neurona, le llega un valor de entrada, que es transformado por una función específica llamada función de activación. Una vez se realiza la transformación, ésta pasa a ser la salida de la neurona.

Las neuronas se conectan entre sí a través de una determinada arquitectura. Cada conexión entre neuronas, tiene un determinado peso, y cada neurona tiene una cantidad determinada de conexiones con otras neuronas que forman la entrada. Dado que tiene varias entradas, con varios pesos, se utiliza una función sumatorio para ponderar el valor de salida de la neurona anterior por el peso de dicha conexión. Este concepto, se ilustra en la siguiente imagen:



Anteriormente, se ha mencionado que las neuronas se conectan entre sí a través de una determinada arquitectura. Esta arquitectura está determinada por capas. Cada capa,está formada por un conjunto de neuronas. La información fluirá de las neuronas de una capa, a las neuronas de otra capa. En la arquitectura más sencilla, existen sólo dos capas, una de entrada y otra de salida, por lo que la información de entrada, será la entrada de las neuronas que formen la capa de entrada, y la salida de estas neuronas, pasará a formar la entrada de las neuronas que forman la capa de salida; esta información, será procesada por las correspondientes neuronas, para formar la salida de la red. Sin embargo, en arquitecturas más complejas, existen capas intermedias, denominadas capas ocultas.

C. PERCEPTRÓN MULTICAPA

Este tipo de redes se caracterizan por tener todas sus neuronas agrupadas en distintas capas. La primera capa (capa de entrada) se encarga únicamente de propagar por el resto de la red las entradas recibidas. Es la última capa (capa de salida), la que se encarga de proporcionar los valores de salida de la red. En las capas intermedias (capas ocultas), se realiza el procesamiento no lineal de los patrones recibidos.

Las conexiones del Perceptrón multicapa son hacia adelante. Generalmente, todas las neuronas de un nivel se conectan con otras neuronas de la capa inmediatamente posterior. A veces, dependiendo de la red, se encuentran conexiones de neuronas que no están en niveles consecutivos, o alguna de las conexiones entre dos neuronas de niveles consecutivos no existente, es decir, el peso asociado a dicha conexión es constante e igual a cero. Además, todas las neuronas de la red tienen un valor umbral asociado. Se suele tratar de una entrada cuyo valor es constante e igual a uno, y lo único que varía es el peso asociado a dicha conexión.

Las funciones de activación que se suelen utilizar en este tipo de redes son las que siguen:

a) Función identidad

$$f(x) = x$$

b) Función sigmoidal

$$f(x) = \frac{1}{1 + e^{-x}}$$

c) Función tangente hiperbólica

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

La principal diferencia entre la función sigmoidal y la función tangente hiperbólica es el rango de sus valores de salida. Mientras que en la primera el rango es [0,1], para la segunda es [-1,1].

D. ENTRENAMIENTO DE LAS REDES NEURONALES

Una de las principales características de las redes neuronales es su capacidad de aprendizaje. El entrenamiento de la redes muestra algunos paralelismo con el desarrollo intelectural de los seres humanos.

El objetivo del entranamiento de una red neuronal, es conseguir que una aplicación determinada, para un conjunto de entradas, produzca el conjunto de salidas deseadas. Consiste en la aplicación secuencial de diferentes conjuntos o vectores de entrada para que se ajusten los pesos de las conexiones entre las neuronas según un procedimiento determinado. Durante la sesión de entrenamiento, los pesos convergen gradualmente hacia los valores que hacen que cada entrada produzca el valor de salida deseado.

Los algoritmos de entrenamiento se pueden agrupar en dos grupos: supervisados y no supervisados.

- **Entrenamiento supervisado**

Estos algoritmos requieren el emparejamiento de cada vector de entrada con su correspondiente vector de salida. El entrenamiento consiste en presentar un vector de entrada a la red, calcular la salida de la red, campararla con la salida deseada, y utilizar el error para realimentar la red y cambiar los pesos de acuerdo con un algoritmo que tiende a minimizar el error.

Las parejas de vectores del conjunto de entrenamiento se aplican secuencialmente y de forma cíclica. Se calcula el error y el ajuste de los pesos para cada pareja hasta que el error para el conjunto de entrenamiento entero sea un valor pequeño y aceptable.

- **Entrenamiento no supervisado**

Los sistemas neuronales con entrenamiento supervisado han tenido éxito en muchas aplicaciones, sin embargo, tienen muchas críticas debido a que desde el punto de vista biológico no son muy lógicos. Resulta difícil creer que existe un mecanismo en el cerebro que compare las salidas deseadas con salidas reales.

Los sistemas no supervisados son modelos de aprendizaje más lógicos en los sistemas biológicos. Desarrollados por Kohonon (1984) y otros investigadores, estos sistemas de aprendizaje no supervisado no requieren de un vector de salidas deseadas, y por tanto no se realizan comparaciones entre las salidas reales y las salidas esperadas. El conjunto de vectores de entrenamiento consiste únicamente en vectores de entrada. El algoritmo de

entrenamiento modifica los pesos de la red, de forma que produzca vectores de salida consistentes. El proceso de entrenamiento extrae las propiedades estadísticas del conjunto de vectores de entrenamiento y agrupa en clases los vectores similares.

Existe una gran variedad de algoritmos de entrenamiento hoy en día, la gran mayoría de ellos han surgido de la evolución del modelo de aprendizaje no supervisado que propuso Hebb (1949). El modelo propuesto por Hebb se caracteriza por incrementar el valor del peso de la conexión si las dos neuronas unidas son activadas o disparadas.

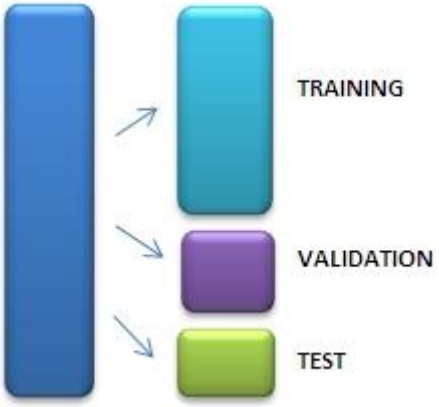
E. CONJUNTO DE ENTRENAMIENTO, VALIDACIÓN Y PRUEBA

Cuando realizamos modelos predictivos, hay tres conjuntos de datos fundamentales que se deben manejar:

1. Muestra de entrenamiento (Trainning): son los datos con los que se entrenan los modelos
2. Muestra de validación (Validation): selecciona el mejor de los modelos entrenados
3. Muestra de prueba (Test): entrega el error real cometido con el modelo seleccionado

Cuando tenemos suficientes datos, se puede subdividir los datos en estos tres conjuntos. Durante el proceso de selección del mejor modelo, los modelos se ajustan a los datos de entrenamiento, y el error de predicción para dichos modelos es obtenido mediante el uso de los datos de validación. Este error de predicción en los datos de validación se puede utilizar para decidir cuándo dar por terminado el proceso de selección. Finalmente, una vez que termina el proceso y se tiene seleccionado el modelo, se pueden utilizar los datos de prueba para evaluar la manera en que el modelo seleccionado se generaliza para los datos que no jugaron ningún papel en la selección del mismo.

Los autores del libro The Elemenst of Statistical Learning (2011), Hastie, Tibshirani y Friedman, señalan que es difícil dar una regla general sobre cuántas observaciones se deben asignar a cada conjunto, aunque indican que una división típica puede ser el 50 % para el conjunto de entrenamiento, y el 25 % para la validación y prueba, respectivamente.


TRAINING
VALIDATION
TEST

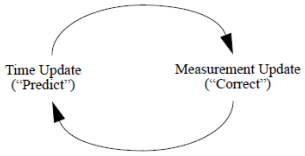
6 - FILTRO DE KALMAN

El conocido filtro de Kalman, tiene sus orígenes en el año 1969, cuando Rudolph E. Kalman, publicó su famoso paper describiendo una solución recursiva para el problema del filtrado lineal de datos discretos. Desde ese momento, debido a sus grandes ventajas de cómputo digital, el filtro de Kalman ha sido objetivo en numerosos estudios y aplicaciones, particularmente en el área de la navegación autónoma o asistida.

El filtro de Kalman, está compuesto por un conjunto de ecuaciones que ofrece una solución eficiente. Este filtro, es muy potente en muchos aspectos: soporta estimaciones de estados del pasado, del presente e incluso del futuro. Además, es capaz de hacerlo en situaciones en las que la precisión del sistema modelado es desconocida.

El filtro de Kalman estima un proceso utilizando una forma de control con realimentación: el filtro estima el estado del proceso en un instante de tiempo, y a continuación obtiene realimentación en la forma de mediciones (ruido). Por lo tanto, las ecuaciones que componen el filtro de Kalman pertenecen a dos grupos: ecuaciones de actualización temporal y ecuaciones de actualización de la medición. Las ecuaciones de actualización temporal son las responsables de proyectar en el tiempo el estado actual y las estimaciones de la covarianza del error para obtener las estimaciones a priori para el siguiente instante de tiempo. Las ecuaciones de actualización de la medición son responsables de la realimentación, por ejemplo, para la incorporación de nuevas medidas en la estimación a priori para obtener una estimación a posteriori.

Las ecuaciones de actualización temporal también pueden ser vistas como las ecuaciones de un predictor, mientras que las ecuaciones de actualización de la medida pueden ser vistas como ecuaciones de un corrector. En efecto, el algoritmo final de estimación se asemeja un algoritmo predictor-corrector para solucionar problemas numéricos como se muestra en la siguiente imagen:



Las ecuaciones que pertenecen a cada grupo son las mostradas a continuación:

ECUACIONES DE ACTUALIZACIÓN TEMPORAL

$$\hat{x}_{k+1}^{-} = A_k \hat{x}_k + Bu_k$$

$$P_{k+1}^{-} = A_k P_k A_k^T + Q_k$$

ECUACIONES DE ACTUALIZACIÓN DE MEDICIÓN

$$K_k = P_k^{-} H_k^T (H_k P_k^{-} H_k^T + R_K)^{-1}$$

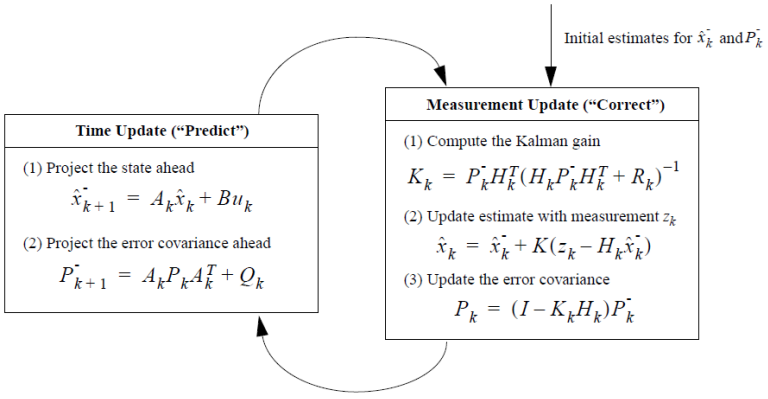
$$\hat{x}_k = \hat{x}_k^{-} + K(z_k - H_k \hat{x}_k^{-})$$

$$P_k = (I - K_k H_k) P_k^{-}$$

La primera tarea a realizar durante la actualización de la medida, es calcular la ganancia de Kalman, K_k. El siguiente paso es realizar la medición el proceso real para obtener z_k, y luego generar un estado de estimación a posteriori incorporando la medición tomada. Finalmente, hay que obtener una estimación de la covarianza del error.

Tras cada ciclo de actualización temporal y de la medición, el proceso se repite con las estimaciones a posteriori anteriores para predecir las estimaciones a priori. Esta naturaleza recursiva es una de las características mas llamativas del filtro de Kalman, haciendo más práctica la implementación de este filtro que, por ejemplo, la implementación del filtro de Weiner, que está diseñado para operar con todos los datos directamente en cada estimación.

El proceso completo que utiliza el filtro de Kalman, con las fórmulas necesarias, se expresa en la siguiente imagen:



EQUIPAMIENTO

1 - EQUIPOS HARDWARE

A. MOXA IOLOGIK E1200

Este dispositivo, suministrado por la empresa llamada Moxa, dispone de dos puertos Ethernet que permiten que la información viaje a otro dispositivo local Ethernet, o conectar varios dispositivos de este tipo en cadena.

Puede ser utilizado en aplicaciones como automatización industrial, seguridad y sistemas de vigilancia, monitorización de túneles, se puede hacer uso de la conexión Ethernet en cadena para construir redes de entradas y salidas a mediante cables Ethernet estándares y con protocolos comunes.

La conexión en Ethernet en cadena que ofrece este dispositivo, no solo incrementa el número de conexiones entre máquinas y paneles, sino que también minimiza el coste al no tener que comprar switches, y al mismo tiempo, reduce las tasas de trabajo y el cableado.

Características principales del producto:

- Comunicación activa con un servidor OPC patentado
- 2 puertos Ethernet para comunicaciones en cadena
- Fácil configuración y despliegue gracias a la utilidad ioSearch™
- Configuración sencilla vía buscador web
- Ahorre tiempo y coste de cableados gracias a la comunicación peer-to-peer
- Direccionamiento Modbus/TCP
- Simplicidad en la administración de entradas y salidas gracias a la librería MXIO, disponible en Windows y Linux.
- Amplio rango de temperaturas: -40 a 75°C
- Compatible con SNMPv1/v2c
- Certificación Atex para zona 2

B. INTELLIGENT SENSOR MODULE WTS 1025-34

El fabricante Weiss Robotics, ofrece un amplio rango de módulos de sensores táctiles inteligentes que integran procesado de señal, diseñados especialmente para las necesidades de los robots y la tecnología de garras. A pesar de su diseño totalmente encapsulado, y su robusta construcción, este sensor ofrece una matriz de sensores muy sensibles que permite una detección muy precisa de contactos de muy bajas fuerzas.

Concretamente, el sensor WTS 1025-34, ofrece un área sensible muy grande en un tamaño muy compacto. La superficie reforzada protege a la matriz del sensor incluso en un ambiente difícil. La tecnología patentada de muestreo permite una captura altamente dinámica de perfiles de presión con una selectividad excelente.

Para poner en marcha el sensor, se distribuye junto a una placa de evaluación, que permite realizar la conexión con el PC mediante cable USB o mediante cable RS-232. Además, también se distribuye junto a un software que permite realizar toda la configuración, y la visualización de los valores capturados por el sensor.

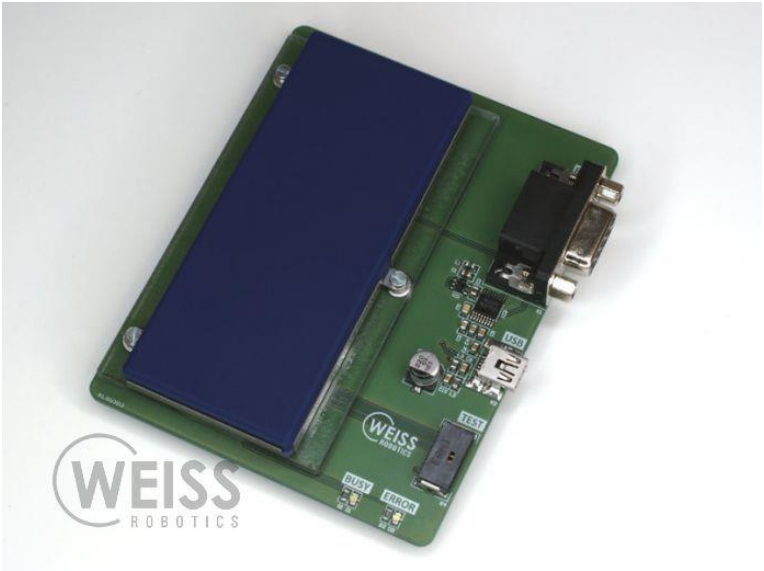
Características principales:

- Número de puntos de medición: 10 x 25
- Resolución espacial: 3.4 mm
- Frecuencia de muestreo: 90 frames/segundo mediante cable USB
- Resolución: 12 bit
- Voltaje de trabajo: 5 V

- Consumo de corriente: 50 mA
- Interfaces integradas: USB 2.0, UART
- Rango de temperatura: 0 a 40 ^{o}C
- Dimensiones: 91 x 49.4 x 6.9 mm
- Peso: 43 g

C. INTERFAZ DE COMUNICACIÓN CON EL SENSOR

Junto al sensor, se proporciona una tarjeta que proporciona la interfaz de comunicación entre el mismo y un computador. Esta tarjeta es la representada en la siguiente imagen:



Desde ella, se pueden realizar las primeras pruebas y configuraciones, y se hace necesaria, ya que el sensor trae una configuración inicial de comunicación para ser realizada mediante RS232. Independientemente del tipo de comunicación, la tensión de alimentación se recoge del conector USB, por lo que hay que conectar tanto el cable RS232, como el USB.

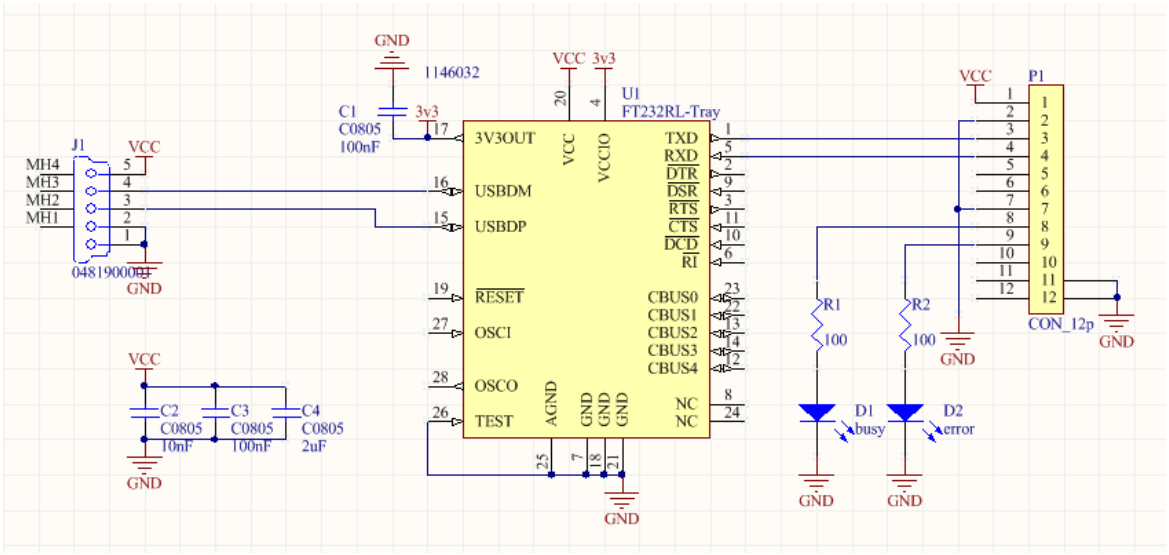
En primer lugar, y tras comprobar que todos los sensores de los que está compuesta la matriz funcionan correctamente con la ayuda del software del fabricante llamado WTS Commander, se ha realizado una configuración para poder prescindir del cable RS232. Para ello, el controlador del sensor tiene integrado un terminal con el que se puede realizar una comunicación mediante, por ejemplo, HyperTerminal, que habrá que configurar de la siguiente forma:

- BaudRate: 115200
- 8 bits de datos
- Sin paridad
- 1 bit de stop
- Sin Handshaking

A partir de aquí, el sensor detectará automáticamente la conexión y mostrará una pantalla de bienvenida.

Desde esta comunicación podremos configurar la interfaz de comunicación, pudiendo elegir entre USB o UART, debiendo elegir este último caso una de las siguientes velocidades: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 ó 460800.

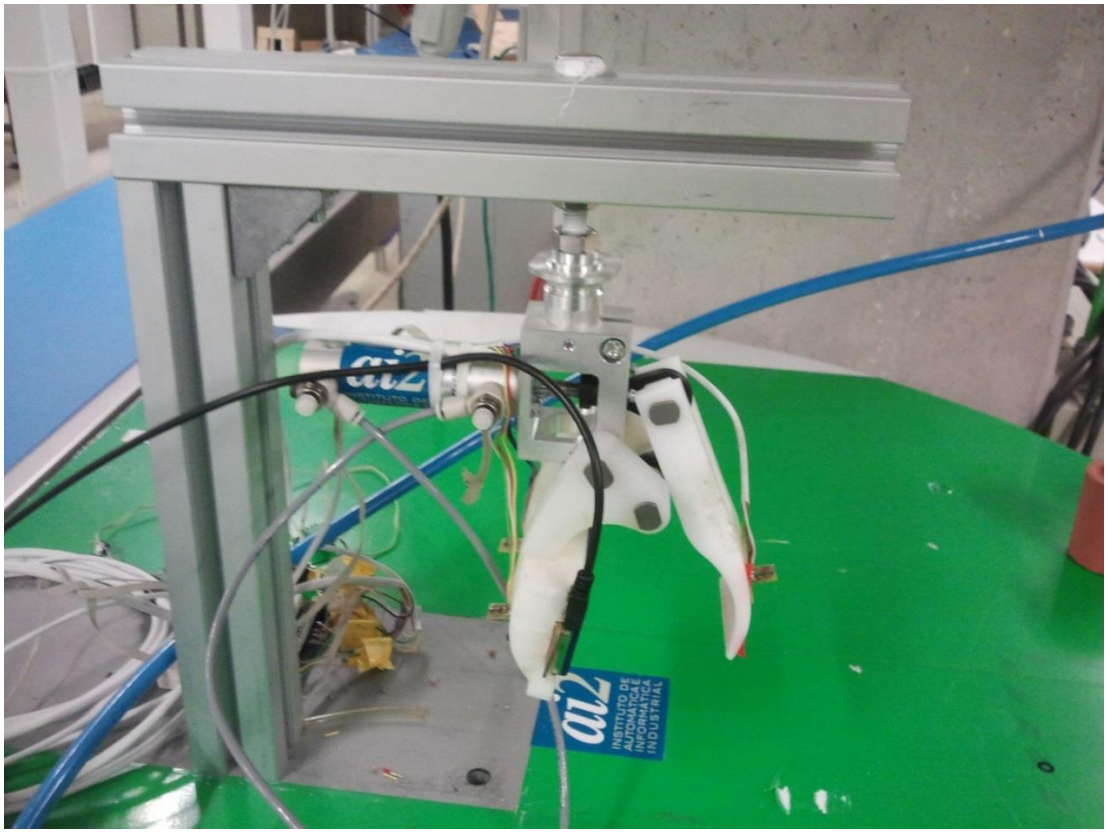
Dado que el objetivo es utilizar el sensor en una garra compatible con un robot industrial, una de las primeras tareas será eliminar esta interfaz de comunicación, y fabricar una que pueda ser utilizada mediante un cable USB, intentando además reducir el tamaño de la misma. Para ello, se ha diseñado la PCB cuyo esquema se muestra a continuación.



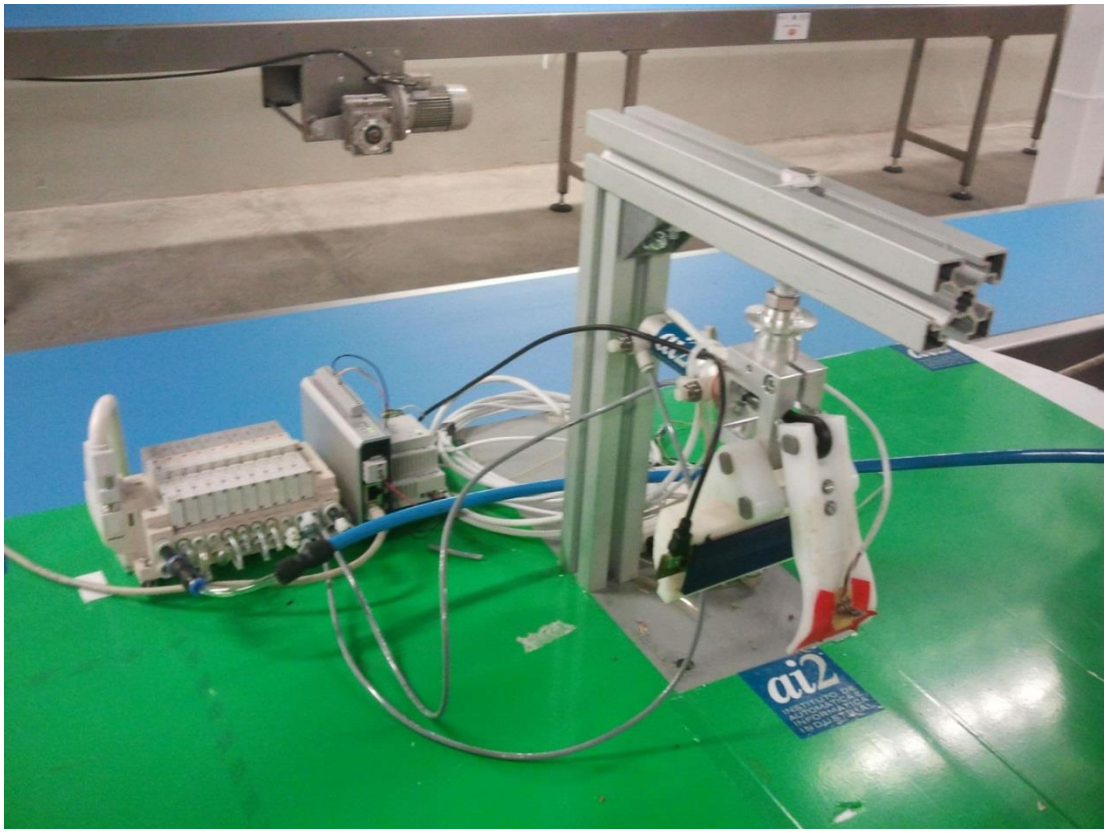
La única parte a mencionar de este circuito, es el circuito integrado FTDI, que es un puente entre comunicación USB a comunicación UART. Este circuito, además ofrece la facilidad de crear en el computador un puerto de comunicación virtual, desde el que se puede trabajar como si de un puerto UART se tratase, por lo que en este caso, nos permitirá comunicar con nuestro sensor con un cable USB.

D. SOPORTE DEL SENSOR

Una vez se ha resuelto la primera tarea, toca situar el sensor en algún soporte que permita cierto control sobre la pieza. Para ello, se ha utilizado la garra que se muestra en la siguiente figura:



En la figura anterior, se puede apreciar también el soporte al que se sujeta la garra, tratándose éste de dos perfiles de aluminio rígidos.

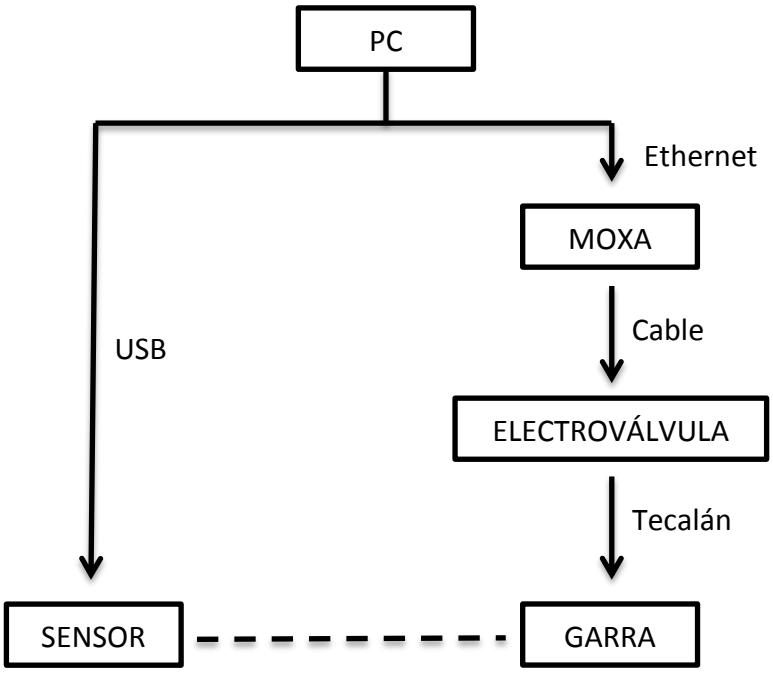


En la imagen anterior, se puede ver el actuador incluido en la garra que permite la apertura y el cierre de la misma, siendo éste un pistón neumático. Dicho pistón, se controla mediante una electroválvula pilotada a través del Moxa ioLogik E1200 (parte izquierda de la imagen), controlado desde el ordenador mediante un cable Ethernet.

E. DIAGRAMA HARDWARE

A continuación, mostraremos un diagrama que ayude a entender, de forma clara, como interactuarán, en nuestro proyecto, los distintos elementos hardware.

Con forma rectangular, se expresarán los componentes hardware. Mientras que en forma de líneas, se indicarán las conexiones entre los elementos. Si la línea es discontinua, la relación a la que se refiere no será una conexión, sino una relación física. En este caso, la relación física (línea discontinua), indica que el sensor se encuentra sujeto en la garra. Si por el contrario, la línea es continua, el método utilizado de comunicación se detallará con un comentario.



2 - PROGRAMAS

A. ECLIPSE

Este programa es un entorno de desarrollo integrado, de código abierto y multiplataforma. Mayoritariamente se utiliza para desarrollar lo que se conoce como Aplicaciones de cliente enriquecido, opuesto a las aplicaciones Cliente-liviano, basadas en navegadores. Es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. No es más que un entorno de desarrollo integrado (IDE) en el que encontrarás todas las herramientas y funciones necesarias para tu trabajo, recogidas además en una atractiva interfaz que lo hace fácil y agradable de usar.

B. MATLAB

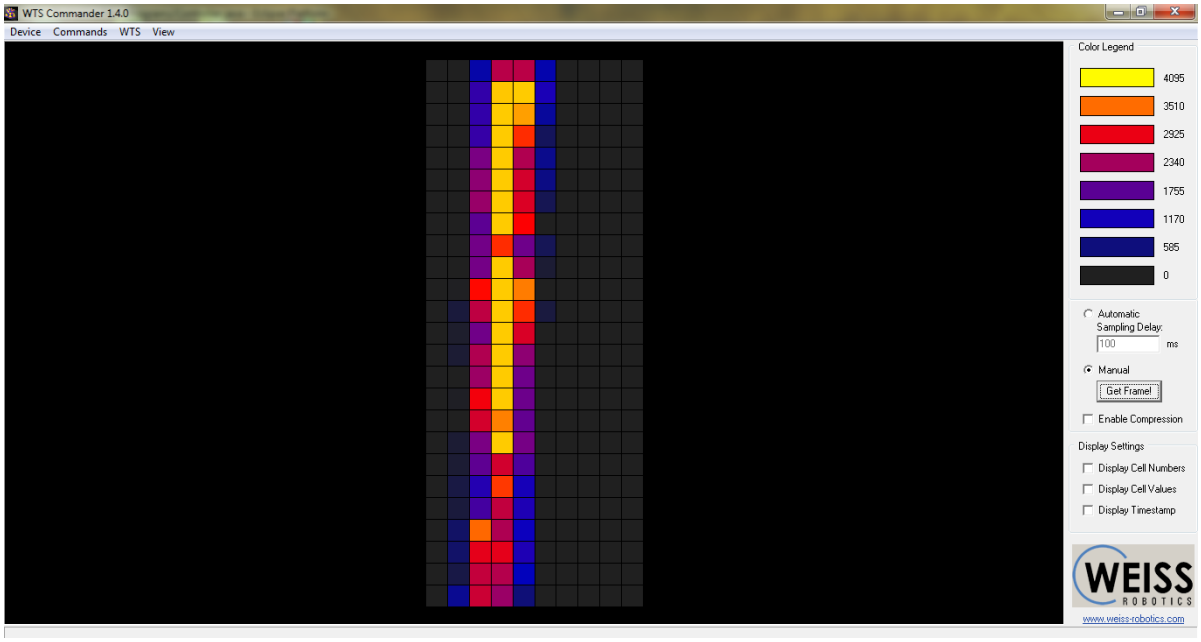
Es un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, la visualización y la programación. Mediante MATLAB, es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. El lenguaje, las herramientas y las funciones matermáticas incorporadas permiten explorar diversos enfoques y llegar a una solución antes que con hojas de cálculo o lenguajes de programación tradicionales, como pueden ser C/C++ o Java.

MATLAB se puede utilizar en una gran variedad de aplicaciones, tales como procesamiento de señales y comunicaciones, procesamiento de imagen y vídeo, sistemas de control, pruebas y medidas, finanzas computacionales y biología computacional. Más de un millón de ingenieros y científicos de la industria y la educación utilizan MATLAB, el lenguaje de cálculo técnico.

C. WTS COMMANDER

Este software viene junto con el sensor de la empresa Weiss Robotics. Con él, es posible realizar adquisiciones de datos para comprobar, de forma gráfica (como se puede ver en la siguiente imagen), el correcto funcionamiento del sensor.

Además se pueden ajustar parámetros del procesador integrado en el sensor, como pueden ser ganancia, threshold, matriz de adquisición, etc.


WTS Commander 1.4.0
Device Commands WTS View
Color Legend
4095
3510
2925
2340
1755
1170
585
0
Automatic
Sampling Delay:
100
ms
Manual
Get Frame!
Enable Compression
Display Settings
Display Cell Numbers
Display Cell Values
Display Timestamp
WEISS
ROBOTICS
www.weiss-robotics.com

3 - LIBRERÍAS SOFTWARE

A. JAMOD (JAVA MODBUS LIBRARY)

Esta librería está desarrollada para todo aquel que necesita acceder o compartir datos utilizando protocolos Modbus.

Se trata de una implementación 100 % realizada en Java, para ser utilizada para utilizarlas tanto en maestros como esclavos en alguna de las siguientes formas:

A) Serie: ASCII, RTU, BIN
B) IP: TCP, UDP

El diseño de esta librería está totalmente orientado a objetos, basada en abstracciones que deberían permitir una fácil comprensión, reusabilidad y extensibilidad, ya que uno de los objetivos principales de este proyecto es la generación de código que puede ser utilizado en una gran variedad de plataformas de Java y de dispositivos.

- **Ejemplo de uso:**

En primer lugar, necesitamos un esquema simple de un programa en Java, importando todas las librerías jamod:

```
import java.net.*;
import java.io.*;
import net.wimpi.modbus.*;
import net.wimpi.modbus.msg.*;
import net.wimpi.modbus.io.*;
import net.wimpi.modbus.net.*;
import net.wimpi.modbus.util.*;

public class DITest {

  public static void main(String[] args) {
    try {
      ...
      ...
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }//main

}//class DITest
```

Ahora, añadimos las instancias y las variables que el programa necesitará en su ejecución:

```
/* The important instances of the classes mentioned before */


TCPMasterConnection con = null; //the connection
ModbusTCPTransaction trans = null; //the transaction
ReadInputDiscretesRequest req = null; //the request
ReadInputDiscretesResponse res = null; //the response

/* Variables for storing the parameters */
InetAddress addr = null; //the slave's address
int port = Modbus.DEFAULT_PORT;
int ref = 0; //the reference; offset where to start reading from
int count = 0; //the number of DI's to read
int repeat = 1; //a loop for repeating the transaction
```

Ahora, la aplicación necesita leer algunos parámetros:

1. Dirección
2. Registro
3. Bitcount
4. Repetir

```
//1. Setup the parameters
if (args.length < 3) {
 System.exit(1);
} else {
 try {
  String astr = args[0];
  int idx = astr.indexOf(':');
  if(idx &gt; 0) {
   port = Integer.parseInt(astr.substring(idx+1));
   astr = astr.substring(0,idx);
  }
  addr = InetAddress.getByName(astr);
  ref = Integer.decode(args[1]).intValue();
  count = Integer.decode(args[2]).intValue();
  if (args.length == 4) {
   repeat = Integer.parseInt(args[3]);
  }
 } catch (Exception ex) {
  ex.printStackTrace();
```

```
    System.exit(1);
   }
  }
```

El siguiente código, será utilizado para configurar y abrir la conexión, así como para preparar una petición y una transacción.

```
//2. Open the connection
con = new TCPMasterConnection(addr);
con.setPort(port);
con.connect();

//3. Prepare the request
req = new ReadInputDiscretesRequest(ref, count);

//4. Prepare the transaction
trans = new ModbusTCPTransaction(con);
trans.setRequest(req);
```

Ahora, estamos en condiciones para ejecutar la transacción que se ha preparado en el código anterior, repitiéndolo el número de veces que se desee (registro repeat), y más tarde, a método de limpieza, cerrar la conexión:

```
//5. Execute the transaction repeat times
int k = 0;
do {
  trans.execute();
  res = (ReadInputDiscretesResponse) trans.getResponse();
  System.out.println("Digital Inputs Status=" + res.getDiscretes().toString());
  k++;
} while (k < repeat);

 //6. Close the connection
 con.close();
```

B. MATLABCONTROL

Esta librería tiene como objetivo permitir comunicación entre el entorno de desarrollo de MATLAB y Java. Permite ejecutar cualquier comando en MATLAB, así como

mandar y recibir variables desde y hacia Java. Esta interacción, se puede realizar tanto desde dentro de MATLAB, como desde fuera (cualquier otro programa).

- **Ejemplo de uso:**

En la página del proyecto, existe una wiki con ejemplos de uso, muy sencilla de consultar. De ella, podemos extraer el siguiente ejemplo de código:

```
public static void main(String[] args) throws MatlabConnectionException,
MatlabInvocationException
{
   //Create a proxy, which we will use to control MATLAB
   MatlabProxyFactory factory = new MatlabProxyFactory();
   MatlabProxy proxy = factory.getProxy();

   //Create and print a 2D double array
   double[][] array = new double[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
   System.out.println("Original: ");
   for(int i = 0; i < array.length; i++)
   {
      System.out.println(Arrays.toString(array[i]));
   }



   //Send the array to MATLAB, transpose it, then retrieve it and convert it to a 2D double
array
   MatlabTypeConverter processor = new MatlabTypeConverter(proxy);
   processor.setNumericArray("arrayM", new MatlabNumericArray(array, null));
   proxy.eval("arrayM = transpose(arrayM)");
   double[][] transposedArray = processor.getNumericArray("arrayM").getRealArray2D();

    //Print the returned array, now transposed
    System.out.println("Transposed: ");
    for(int i = 0; i < transposedArray.length; i++)
    {
       System.out.println(Arrays.toString(transposedArray[i]));
    }

   //Disconnect the proxy from MATLAB
   proxy.disconnect();
}
```

En el código anterior, se puede ver cómo en primer lugar se crea un proxy para abrir la aplicación MATLAB. A continuación, se crea una matriz de 3x3, se imprime, y se convierte esta matriz a un tipo de datos compatible para la comunicación entra Java y MATLAB.

En el fragmento de código siguiente:

```
processor.setNumericArray("arrayM", new MatlabNumericArray(array, null));
proxy.eval("arrayM = transpose(arrayM)");
```

Se puede ver como se realiza el envío dela variable “array” (de Java), que en matlab aparecerá con el nombre “arrayM” y cómo se ejecuta el código en MATLAB para generar la matriz traspuesta.

De igual forma, con el siguiente código:

```
double[][] transposedArray = processor.getNumericArray("array").getRealArray2D();
```

Se realiza el envío desde MATLAB a Java de la matriz traspuesta, para pasar a imprimir los valores.

Como se puede ver, resulta una implementación muy sencilla de comunicar ambas plataformas.

C. NEUROPH STUDIO

Este proyecto, sin lugar a dudas, es uno de los más completos utilizados en el desarrollo del estudio que nos ocupa. Se trata de un proyecto para el desarrollo de redes neuronales en el lenguaje Java.

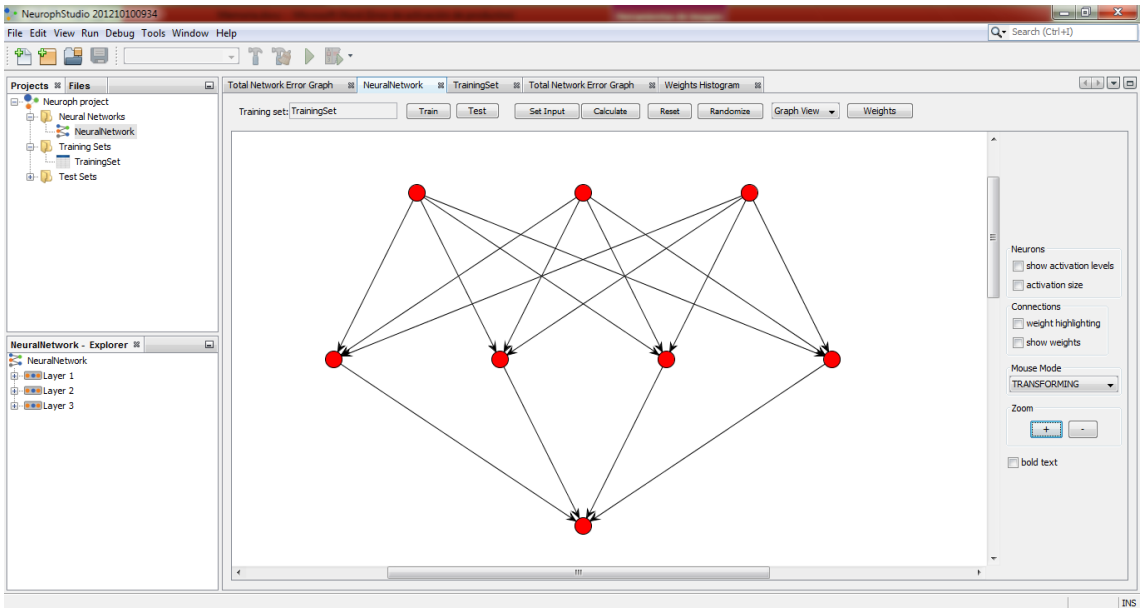
Dispone tanto de las librerías necesarias para poder incluir el control de estas redes en cualquier proyecto, como una aplicación gráfica desde la que generar distintos tipos de proyectos basados en redes neuronales, como pueden ser reconocimiento de imágenes, reconocimiento de texto, o reconocimiento de caracteres escritos a mano, entre otros. Además de la creación de la red neuronal, se permite realizar el entrenamiento de la misma, y el guardado en un archivo para poder utilizarlo en el programa. Los tipos de redes neuronales que permite crear son los siguientes:

1. Adaline

2. Perceptrón
3. Perceptrón multicapa
4. Hopfield
5. BAM
6. Kohonen
7. Hebbian supervisada
8. Hebbian sin supervisar
9. Maxnet
10. Red competitiva
11. RBF
12. InStar
13. OutStar

Como puede verse, es una aplicación que permite el desarrollo de múltiples tipos de redes neuronales, de forma clara y sencilla.

La interfaz de la aplicación puede verse en la siguiente imagen:



- **Ejemplo de uso:**

Para ver la forma de uso de este software, haremos un pequeño ejemplo en el que simularemos un escenario en el que un coche, tiene que esperar una señal para poder moverse. Estas señales provienen de tres luces: verde, amarilla y roja. El objetivo de este ejemplo será mostrar cómo crear una red, entrenarla, importarla a Java y utilizarla en una aplicación.

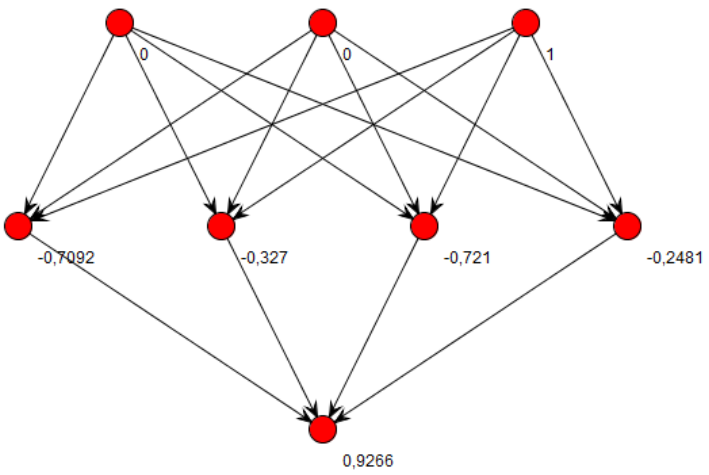
Los estados de las señales serán:

1. Roja: no se podrá mover
2. Amarilla: enciende motor
3. Verde: puede moverse

La red tendrá tres entradas, y utilizaremos una arquitectura de Perceptrón multicapa. Una capa oculta con cuatro neuronas será la encargada de decidir la salida de la red. La salida de la red estará dada por una sola neurona, y dará un valor que indique si el coche podrá moverse o no.

Los pasos a seguir son los siguientes:

1. Creamos la red, el conjunto de entrenamiento y entrenamos la red. Las características que se han escogido para la red son:
    a. Perceptrón multicapa
    b. Función de transferencia: tangente hiperbólica
    c. Tipo de entranemiento: Backpropagation with Momentum
2. Una vez tenemos la red entrenada, podemos probar el resultado introduciendo manualmente un vector de entrada. Para ello, introduciremos el vector 0 0 1, representando este vector los valores de las luces roja amarilla y verde, por lo que como resultado, deberíamos obtener es un 1, para poder movernos. El caso, se muestra en la siguiente imagen:



3. Una vez tenemos la red completamente entrenada y funcional, para poder utilizarla en nuestro proyecto Java, pasaremos a guardarla en un fichero .nnet (en nuestro

caso lo llamaremos 'redNeuronal.nnet'), que será el que más tarde utilizaremos en nuestro ejemplo.

4. Como código de ejemplo, el siguiente muestra cómo cargar nuestra red neuronal, y calcular las tres posibles salidas:

```
public class PruebaRedNeuronal
{

   public static void main(String[] args)
   {
        NeuralNetwork red = NeuralNetwork.load("redNeuronal.nnet");

      calculate(red,1,0,0);
      calculate(red,0,1,0);
      calculate(red,0,0,1);

   }

   private void calculate(NeuralNetwork network, double... input)
   {
      network.setInput(input);
      network.calculate();
      Vector output = network.getOutput();
      Double answer = output.get(0);
      System.out.println(answer);
   }
}
```

Cuyo resultado de ejecución será:

```
-1.6360230873976706E-6
-4.140786100885251E-6
0.9684448970000741
```

DESARROLLO

1 - PROGRAMACIÓN SENSOR

Para poder comunicar con el sensor y utilizando el lenguaje de programación Java, se ha realizado una serie de clases que se ejecutan en distintos hilos, para poder obtener la mayor velocidad de cómputo posible.

Las clases que se han creado son las siguientes:

- SensorAccess.java
- ReaderThread.java
- WriterThread.java
- Message.java
- MessageSemaphore.java
- CompressedDataManagement.java

Pasaremos a explicar los métodos y los objetivos de cada una de ellas:

A. SENSORACCESS.CLASS

Esta clase se puede considerar la principal de todas las clases programadas para la comunicación entre el computador y el sensor. Su constructor, es el encargado de buscar, entre todas las conexiones serie disponibles en el ordenador, cual corresponde al sensor. Los métodos que contiene esta clase son:

Constructor: cuando se crea una instancia de esta clase, el constructor es el encargado de buscar entre todos los dispositivos conectados al ordenador mediante comunicación serie, el que responda satisfactoriamente a una petición de información del sistema. Dado que siguiendo este método, hay que mandar un mensaje de petición y esperar a que el dispositivo responda correctamente, para ahorrar tiempo de espera, se ha realizado una comprobación previa para quitar todos los dispositivos ajenos. Esta comprobación se trata de buscar sólo el dispositivo que se encuentre conectado en el puerto COM19, ya que el sensor siempre queda conectado en este puerto virtual.

Read: este método, comprueba si existen bytes disponibles en el puerto de comunicaciones, y en caso de existir crea un objeto de tipo Message para devolverlo a la clase invocadora.

Write: envía por el puerto de comunicación el mensaje de tipo Message pasado como argumento.

B. READERTHREAD.CLASS

Esta clase tiene como objetivo comprobar en todo momento si el sensor ha enviado algún mensaje al computador. Para ello, hereda de la clase Thread para ejecutarse concurrentemente.

Solamente dispone del método run, que se encarga de llamar al método read de la clase SensorAccess (anteriormente mencionada). En caso de que este método devuelva algún mensaje, se comprueba si este mensaje es un mensaje estándar o si se trata de un mensaje de lectura periódica del sensor.

En caso de tratarse de un mensaje estándar, éste se pasa al semáforo de sincronización, que se verá más tarde.

En caso contrario, primero se comprueba si el mensaje utiliza la compresión proporcionada por el fabricante del sensor, para pasarlo a la clase encargada de administrar estos mensajes, o si no lo utiliza, para guardar los valores devueltos por el sensor en un fichero de texto.

C. WRITERTHREAD.CLASS

De igual forma que se ha programado una clase para leer del sensor, también se ha creado una para mandar mensajes al mismo, siendo esta clase la encargada de ello. Cuando cualquiera de las clases requiera enviar un mensaje al sensor, bien sea para configurar algún parámetro, o para realizar alguna petición de algún registro, debe utilizar esta clase para ello. Los métodos utilizados son los siguientes:

addMessage: es el utilizado por las clases para agregar un mensaje a la cola interna de la clase.

takeMessage: este método privado, es el que utiliza este hilo para comprobar si alguien ha añadido un mensaje para enviar al sensor. En caso de que la cola contenga algún mensaje, este es devuelto.

Dado que está programado como hilo independiente, se comprueba constantemente si hay mensajes en la cola mediante el método takeMessage. Si este devuelve un mensaje, se llama a la clase sensorAccess para que se encargue de hacer llegar el mensaje al sensor.

D. MESSAGE.CLASS

Esta clase es la que contiene toda la información de bajo nivel de los mensajes. Contiene los campos que componen el mensaje que hay que mandar al sensor, como son:

- Id
- Longitud
- Checksum
- Estatus
- Cuerpo

Existen tres constructores de esta clase:

- Constructor con dos datos, el id y los datos: calculará todos los demás campos para establecer los valores necesarios a la hora de realizar el envío del mensaje.
- Constructor con un dato, el id: necesario para crear algunos mensajes concretos que no necesitan cuerpo.
- Constructor con un dato, un vector de bytes: es el que se utiliza cuando se recibe un mensaje del sensor. Se encarga de crear todos los datos del mensaje a partir de los bytes que ha mandado el sensor.

Algunos métodos interesantes para el uso correcto del sensor son:

SaveDataArray: se encarga de guardar los datos del cuerpo del mensaje al fichero cuyo nombre se pasa como argumento.

DataToDoubleArray: se utiliza cuando se quieren convertir los bytes del cuerpo del mensaje al tipo de datos double, para un uso más sencillo de los mismos, sobre todo cuando se trata de una lectura de los puntos de la matriz.

getDoubleValures: devuelve el vector que se ha creado con el método DataToDoubleArray.

E. COMPRESSEDDATAMANAGEMENT.CLASS

Esta clase se ha programado para intentar liberar las tareas de recepción de los mensajes que el sensor envía periódicamente (en caso de haber realizado una configuración para recibir estas lecturas).

El proceso que permite liberar la carga es:

1. La clase que se encarga de realizar la lectura del sensor, recibe un mensaje.
2. Esta clase envía el mensaje a la clase compressedDataManagement
3. Cuando esta clase recibe un mensaje nuevo, realiza el cómputo que sea necesario, mientras que la clase que realiza la lectura del sensor, puede seguir leyendo mensajes del sensor.

Una variable interna, concretamente una cola, permite que se puedan recibir mensajes más rápido de lo que se pueden procesar, por lo tanto, este hilo comprobará constantemente si se han añadido mensajes nuevos, para en caso de haberlos, guardar los valores de los que estemos interesados, en este caso, los 250 valores de la matriz del sensor.

F. MESSAGESEMAPHORE.CLASS

Esta clase ha sido implementada para sincronizar la llegada de las respuestas a mensajes desde el computador hasta el sensor.

Cuando el computador envía algún tipo de mensaje, sea del tipo que sea, el sensor siempre responde con un mensaje de confirmación, en el que indica el éxito de la transacción, e incluso, en los mensajes que lo requieran, incluir los datos que responden a la petición de algún registro.

Es por esto, que se ha implementado esta clase, para que cuando desde algún punto del programa se envíe un mensaje al sensor, se espere a que llegue la respuesta del mismo, de forma que no se pueda continuar la ejecución del programa sin saber si la transacción ha sido satisfactoria o por el contrario, ha sufrido algún tipo de error.

Para llevar a cabo esta tarea, la clase MessageSemaphore sólo tiene dos métodos:

GetMessage: permite recoger el mensaje cuyo origen es el sensor, por lo que debe utilizarlo aquel cliente que haya enviado cualquier mensaje al sensor.

SetMessage: de igual forma que el método anterior, sirve para añadir un mensaje, en este caso, es el utilizado por el sensor.

Para permitir que la ejecución del cliente que ha mandado cualquier mensaje al sensor no continúe hasta que éste responda con algún mensaje, se utilizan dos semáforos. De la literatura, podemos extraer la siguiente definición de semáforo:

"A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the `Semaphore` just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource" (docs.oracle.com)

En español:

Semáforo contador. Conceptualmente, un semáforo mantiene una serie de permisos. Cada llamada al método acquire() bloquea si es necesario hasta que se encuentre disponible algún permiso, para cuando lo esté cogerlo. Cada llamada al método reléase() añade un permiso, normalmente desbloqueando alguna llamada bloqueada (mediante el método acquire()). Sin embargo, actualmente no se utilizan objetos de permisos; el objeto Semaphore mantiene la cuenta de permisos disponibles y actúa en consecuencia.

Los semáforos son utilizados para restringir el número de hilos que pueden acceder a cualquier recurso (sea físico o lógico).

Como se ha mencionado antes, esta clase utiliza dos semáforos. Viendo la definición anterior, se podrá comprender el uso de uno de los semáforos para bloquear el hilo que intenta coger un mensaje, así como el otro semáforo bloqueará al que quiera dejar un mensaje mientras no se haya cogido el mensaje que se ha depositado anteriormente.

2 - PROGRAMACIÓN MOXA

Dado que el servidor Moxa tiene como tarea la apertura y el cierre de la pinza que mantiene el sensor, se ha programado una clase con dos métodos static:

Activar: se utiliza para cerrar la garra que sostiene el sensor. Para realizar este proceso, en primer lugar se crea una conexión con el servidor mediante el protocolo TCP/Modbus (mediante la librería Jamod), a continuación se realiza una escritura con contiene la salida que debe modificar, y el valor al que debe actualizar la salida, para finalmente ejecutarla y cerrar la conexión.

Desactivar: se utiliza de la misma forma que el método activar, con la salvedad de que el valor que se escribe en el registro es el contrario, por lo que de forma análoga, la acción sobre la garra será contraria (apertura).

3 - CLASES LAUNCHER Y CONTROLLER

Además de las clases explicadas con anteriormente, se han creado otras dos que son las encargadas de controlar la aplicación.

En un principio, el objetivo de esta aplicación fue crear una interfaz gráfica desde la que realizar todas las pruebas, permitiendo informar visualmente al exterior sobre el tipo de material que se encuentra sobre el sensor. Sin embargo, debido al problema del tiempo disponible para finalizar la aplicación, a mitad de proyecto se ha descartado esta idea, adoptando el mismo objetivo pero eliminando la interfaz gráfica, por lo que el nuevo objetivo se simplifica a crear una aplicación Java que realice las capturas de las muestras, las guarde en un fichero, y se sincronice con la aplicación MATLAB para que desde ella se realice todo el cómputo encargado de realizar la distinción entre materiales y de informar al exterior sobre los mismos.

Es por esto, que se creó una clase encargada de realizar todas las instancias de las clases que se van a necesitar a lo largo de la ejecución del programa. Esta clase, se llama Launcher.java, y contiene únicamente el método main, encargado de instanciar las siguientes clases:

- MessageSemaphore
- SensorAccess
- CompressedDataManagement
- WriterThread
- ReaderThread
- Controller

Prestando atención a las clases instanciadas desde el método main, nos podremos dar cuenta de que a partir de aquí, se pondrán en ejecución cuatro hilos, pertenecientes a las clases CompressedDataManagement, SensorReader, SensorWriter y Controller.

Hasta ahora, no se ha comentado nada acerca de la nueva clase comentada llamada Controller. Esta clase es el corazón que le da vida a la comunicación entre el sensor, el actuador de la garra que mantiene al sensor, y la aplicación MATLAB.

La tarea más importante que se realiza en el constructor de esta clase, es la creación de la pasarela entre la aplicación MATLAB y este programa, esto quiere decir, que será desde esta clase, desde la que se ejecutarán todas las tareas necesarias en el entorno MATLAB.

Una vez se ha creado esta pasarela, comienza la ejecución del programa, ejecutando los siguientes pasos:

1. Se establece la ganancia del sensor a un valor de 26
2. Se establece el umbral del sensor a un valor de 48
3. Se realiza una tara de los valores del sensor, para ponerlos todos a cero
4. Se realiza el número de tomas deseado.

El proceso que permite tomar una muestra es el siguiente:

1. Comenzar el proceso de envío, por parte del sensor, de lecturas periódicamente. El tiempo que el sensor debe esperar entre envío y envío se establece a 250 milisegundos.
2. Se realiza una llamada a la clase Moxa para llevar a cabo el cierre de la garra.
3. Dado que el sensor ya está mandando la lectura de los valores al computador, la ejecución del hilo Controller se desactiva durante un segundo, tiempo que tara en realizar la toma de la muestra.
4. Cuando ha finalizado el tiempo de captura, se realiza una llamada a la clase Moxa para llevar a cabo la apertura de la garra.
5. Se realiza una transacción con el sensor con el objetivo de parar el envío de los valores periódicamente.
6. Se hace uso de la pasarela con la aplicación MATLAB para que realice todo el cómputo necesario.

Como se ha visto antes, durante el envío de las lecturas periódicas por parte del sensor, la ejecución del hilo Controller pasa a estar en estado de reposo. Es en este momento, cuando entran en acción los hilos SensorReader y CompressedDataManagement.

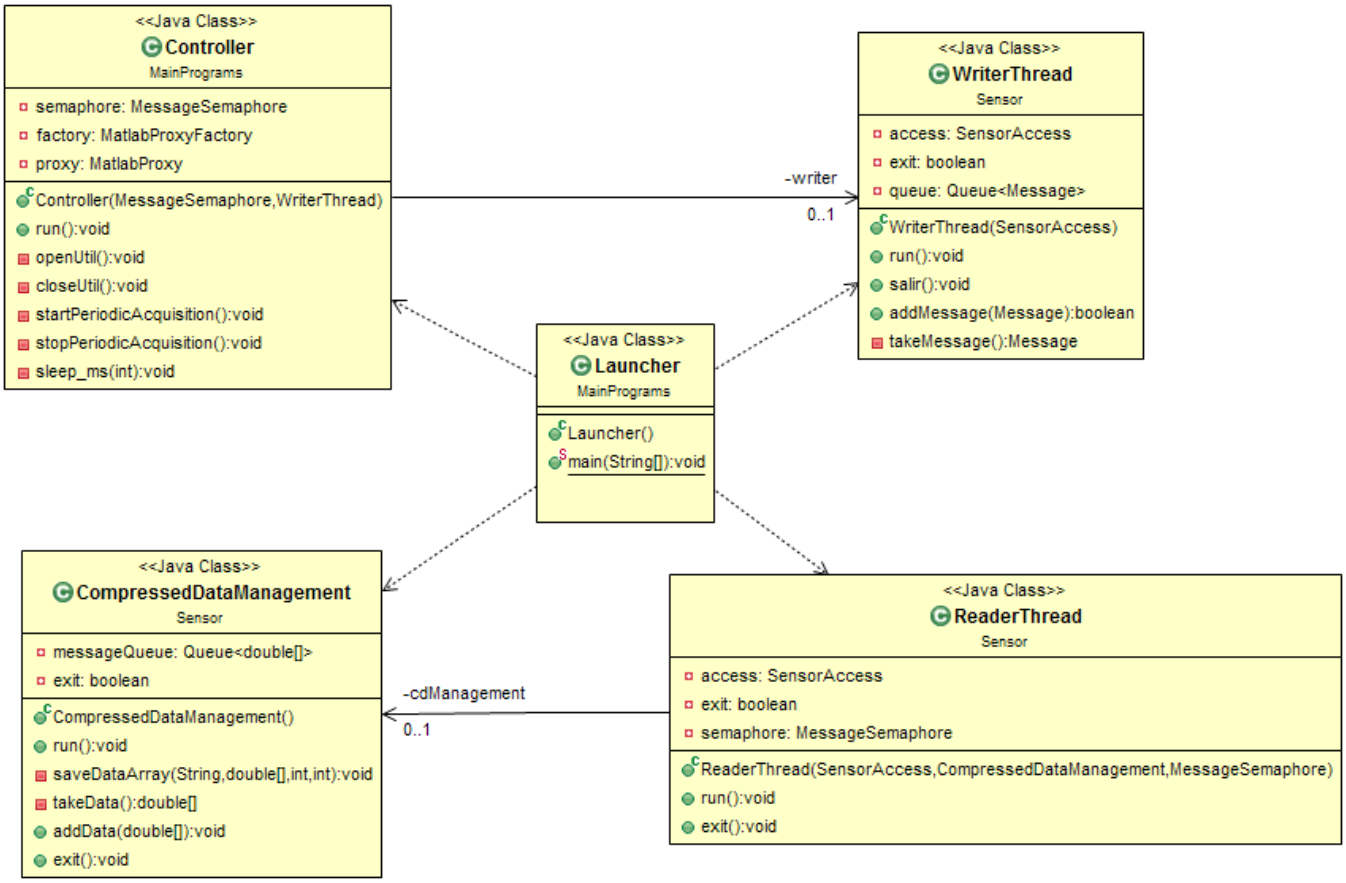
El hilo SensorReader realiza la lectura constantemente de los mensajes que le llegan del sensor, por lo que en cada lectura, este se encargará de comprobar que el mensaje es correcto (comprobando el checksum del mensaje recibido) y en caso afirmativo, pasar el contenido al hilo CompressedDataManagement, para así, poder continuar con la recepción de nuevos mensajes.

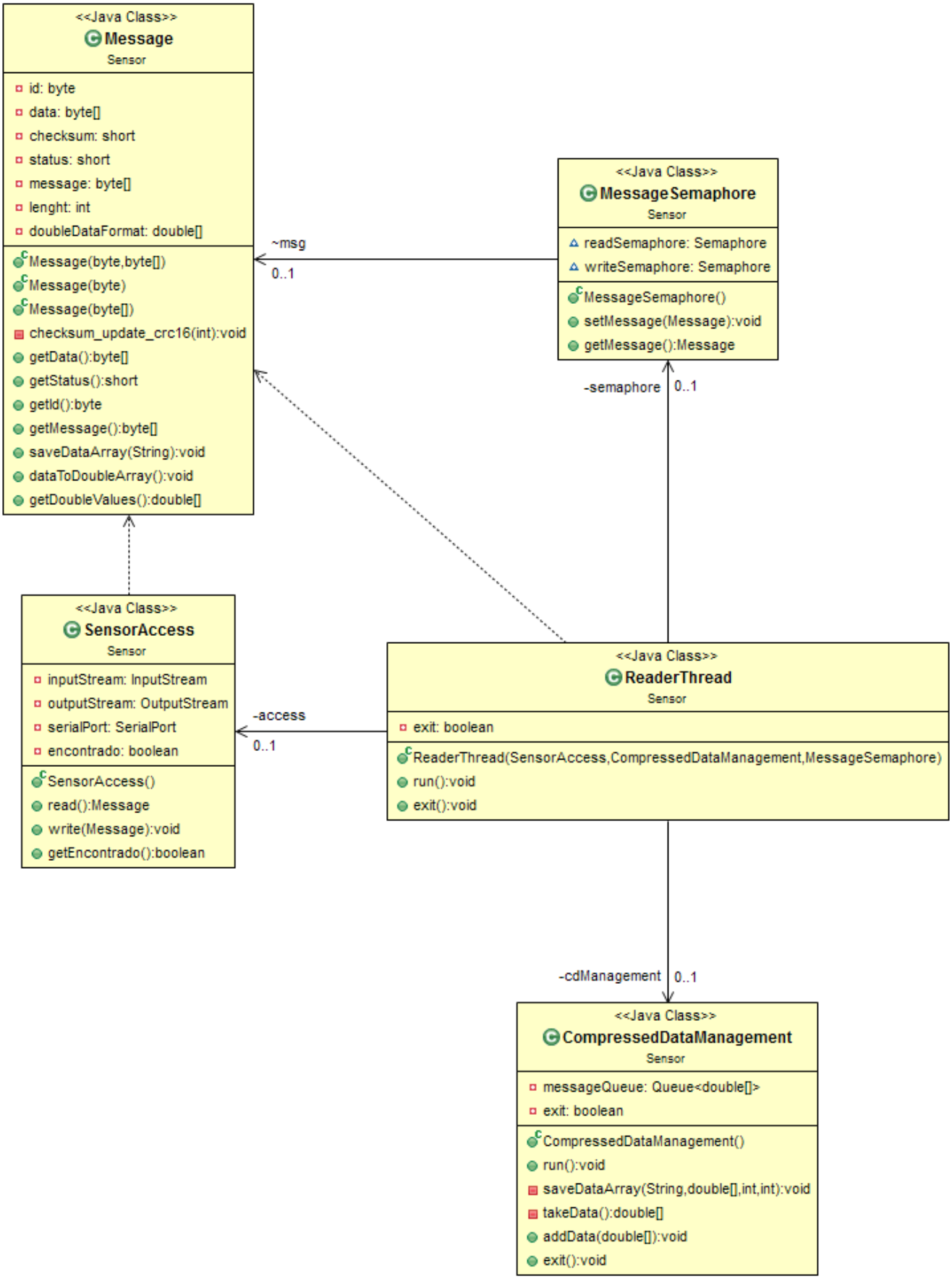
Por otra parte, el hilo CompresedDataManagement, comprueba constantemente la llegada de nuevos mensajes por parte del hilo antes mencionado, para llevar a cabo el salvado de los valores en un fichero. En este fichero, únicamente se guardan los valores de la matriz que son necesarios para llevar a cabo una conclusión del material que se ha utilizado

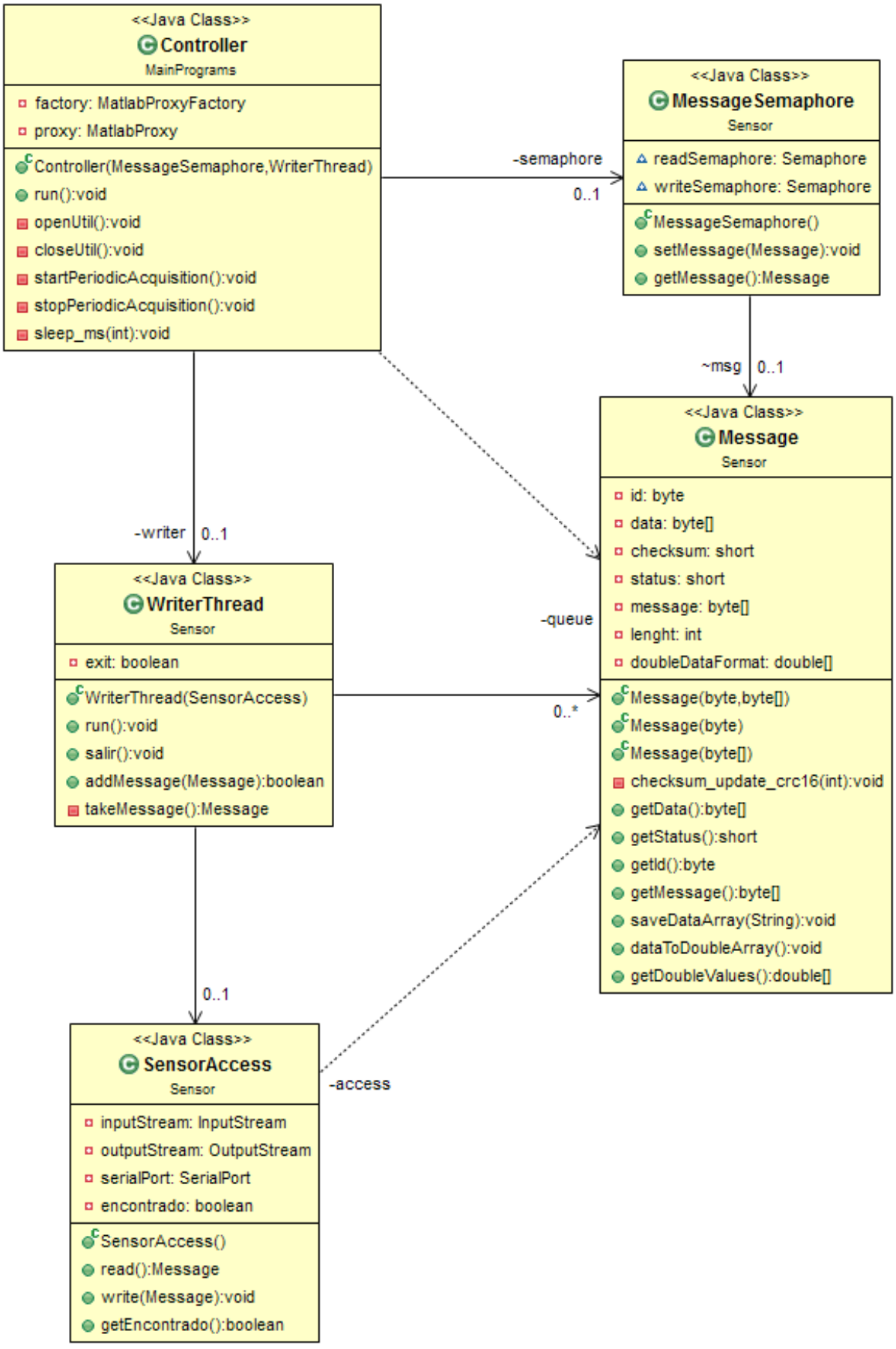
para tomar la muestra. Este fichero será el que utilice la aplicación MATLAB para procesar y sacar conclusiones.

4 - DIAGRAMAS DE CLASES

A continuación, mostraremos los diagramas de las clases utilizadas en el programa realizado con Java. Para más claridad, y dado que en una sóla página no cabe, se ha dividido en tres partes. La primera de ellas muestra la clase principal, Launcher.class, con las clases que instancia. El segundo diagrama muestra la clase Controller.class, que se encarga de controlar las acciones y los mensajes que van a enviarse con el sensor y con el Moxa. Por último, se incluye un diagrama que muestra las relaciones entre las clases utilizadas para realizar la lectura de los mensajes del sensor.




<<Java Class>>
Message
Sensor
id: byte
data: byte[]
checksum: short
status: short
message: byte[]
lenght: int
doubleDataFormat: double[]
Message(byte,byte[])
Message(byte)
Message(byte[])
checksum_update_crc16(int):void
getData():byte[]
getStatus():short
getId():byte
getMessage():byte[]
saveDataArray(String):void
dataToDoubleArray():void
getDoubleValues():double[]
<<Java Class>>
MessageSemaphore
Sensor
readSemaphore: Semaphore
writeSemaphore: Semaphore
MessageSemaphore()
setMessage(Message):void
getMessage():Message
~msg
0..1
-semaphore
0..1
<<Java Class>>
SensorAccess
Sensor
inputStream: InputStream
outputStream: OutputStream
serialPort: SerialPort
encontrado: boolean
SensorAccess()
read():Message
write(Message):void
getEncontrado():boolean
<<Java Class>>
ReaderThread
Sensor
exit: boolean
ReaderThread(SensorAccess,CompressedDataManagement,MessageSemaphore)
run():void
exit():void
-access
0..1
-cdManagement
0..1
<<Java Class>>
CompressedDataManagement
Sensor
messageQueue: Queue<double[]>
exit: boolean
CompressedDataManagement()
run():void
saveDataArray(String,double[],int,int):void
takeData():double[]
addData(double[]):void
exit():void


<<Java Class>>
Controller
MainPrograms
factory: MatlabProxyFactory
proxy: MatlabProxy
Controller(MessageSemaphore,WriterThread)
run():void
openUtil():void
closeUtil():void
startPeriodicAcquisition():void
stopPeriodicAcquisition():void
sleep_ms(int):void
-semaphore
0..1
<<Java Class>>
MessageSemaphore
Sensor
readSemaphore: Semaphore
writeSemaphore: Semaphore
MessageSemaphore()
setMessage(Message):void
getMessage():Message
~msg
0..1
<<Java Class>>
Message
Sensor
id: byte
data: byte[]
checksum: short
status: short
message: byte[]
lenght: int
doubleDataFormat: double[]
Message(byte,byte[])
Message(byte)
Message(byte[])
checksum_update_crc16(int):void
getData():byte[]
getStatus():short
getId():byte
getMessage():byte[]
saveDataArray(String):void
dataToDoubleArray():void
getDoubleValues():double[]
-writer
0..1
<<Java Class>>
WriterThread
Sensor
exit: boolean
WriterThread(SensorAccess)
run():void
salir():void
addMessage(Message):boolean
takeMessage():Message
-queue
0..*
0..1
<<Java Class>>
SensorAccess
Sensor
inputStream: InputStream
outputStream: OutputStream
serialPort: SerialPort
encontrado: boolean
SensorAccess()
read():Message
write(Message):void
getEncontrado():boolean
-access

5 - PROCESADO MATLAB

Cuando el hilo de la clase Controller llama a MATLAB, lo que hace es ejecutar un programa guardado en un fichero .m llamado testFile.m. En este fichero, se han guardado las sentencias necesarias para realizar los siguientes pasos:

1. Cargar la red neuronal previamente creada y guardada en un fichero.
2. Cargar el fichero de datos que ha creado el hilo CompressedDataManagement.
3. Crear un único vector con los valores de la toma.
4. Introducir estos valores en la red neuronal para comprobar el resultado.
5. Informar por pantalla del resultado del material.

Expliquemos el paso 3, en el que se crea un único vector con los valores de la toma. Dado que en el fichero que contiene los valores de la toma de datos no contiene una única lectura, sino que contienen desde el instante previo al cierre de la garra, en el que la mayoría de los valores tienen un valor cero, hasta unos instantes después de realizar la apertura de la garra, se hace necesario un proceso que seleccione los valores de la lectura de un mismo sensor que resultan útiles para la red neuronal. Este proceso, resulta tan sencillo como seleccionar el máximo valor de todos los representados en el fichero, por lo que si, por ejemplo, un fichero llega en forma de matriz de 8x50 (8 lecturas en instantes de tiempo distintos y 50 valores, los relativos a las 5 columnas centrales del sensor), este proceso dará como resultado un vector de 50 elementos, siendo cada uno de ellos el máximo valor de los 8 posible valores guardados en el fichero.

6 - CONFIGURANDO EL SENSOR

Una vez se tiene definido el programa que se encargará de capturar valores del sensor, toca hacer capturas para ver cómo se comporta éste en el tiempo, y ante distintos materiales.

En primer lugar, se han hecho pruebas para establecer la configuración de la ganancia del sensor. En segundo lugar se han hecho pruebas que nos permitan tomar una decisión sobre el tiempo que la garra permanece abierta, en reposo, para por último, realizar más pruebas que permitan establecer el tiempo que la garra se encuentra cerrada, tomando muestras.

A. GANANCIA DEL SENSOR

Este valor, como se ha visto antes, ajusta la sensibilidad de la presión de la matriz. La ganancia, se puede establecer con un valor comprendido entre 0 y 255, siendo el valor cero el más insensible (ganancia más baja), y el valor 255 el más sensible (ganancia más grande).

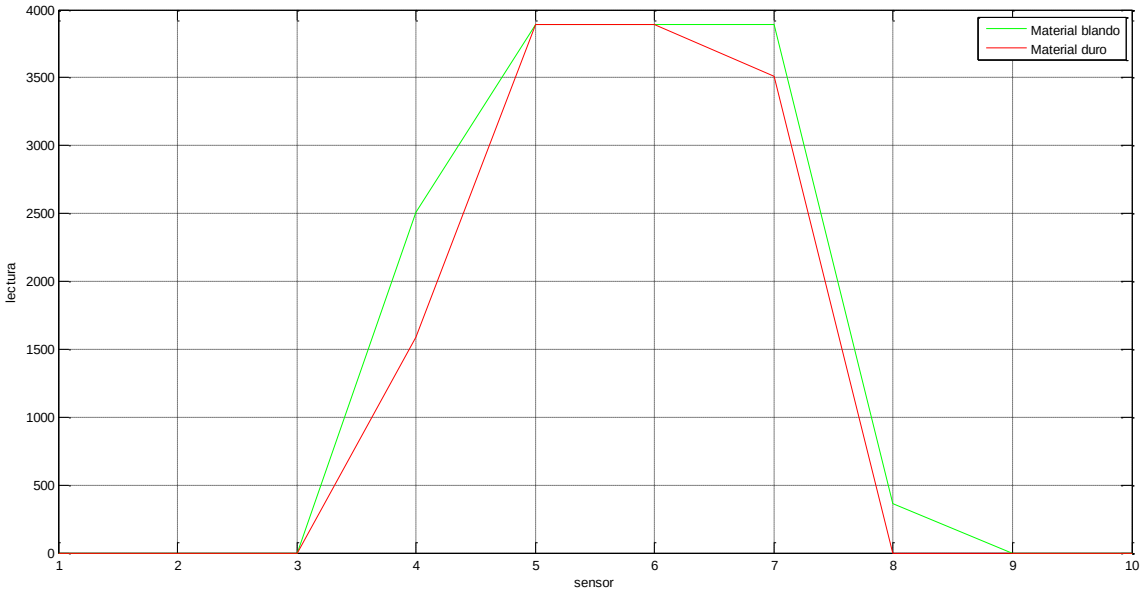
Una vez tenemos clara la definición de la ganancia en el sensor, y sabemos cómo puede afectar a nuestro resultado, se hace necesario realizar un programa que realice varias tomas de muestras, variando la ganancia en cada una de ellas.

Antes de proceder a la ejecución de dicho programa, se han hecho algunas pruebas con la ayuda del software WTS Commander, cambiando el valor de la ganancia con la garra cerrada (con el mismo material en todos los casos) y observando cómo cambia la lectura de los valores ante este cambio. Mediante este procedimiento, se ha observado que la ganancia debía estar entre los valores 40 y 60.

Con estos valores, se ha ejecutado un programa que realice un ciclo de toma de muestra con ganancias comprendidas entre 40 y 60, a los dos materiales que ofrecen los extremos de las densidades (el más duro y el más blando).

Con esto, una vez se han tomado las muestras en ambos materiales, se ha creado un programa en MATLAB para buscar la mayor diferencia entre ambos materiales, dando el máximo valor con una ganancia establecida de 57, por lo que adoptaremos este valor como la ganancia a utilizar en las siguientes tomas. En la siguiente imagen, podemos observar una gráfica en la que se observan las diferencias entre los dos materiales.

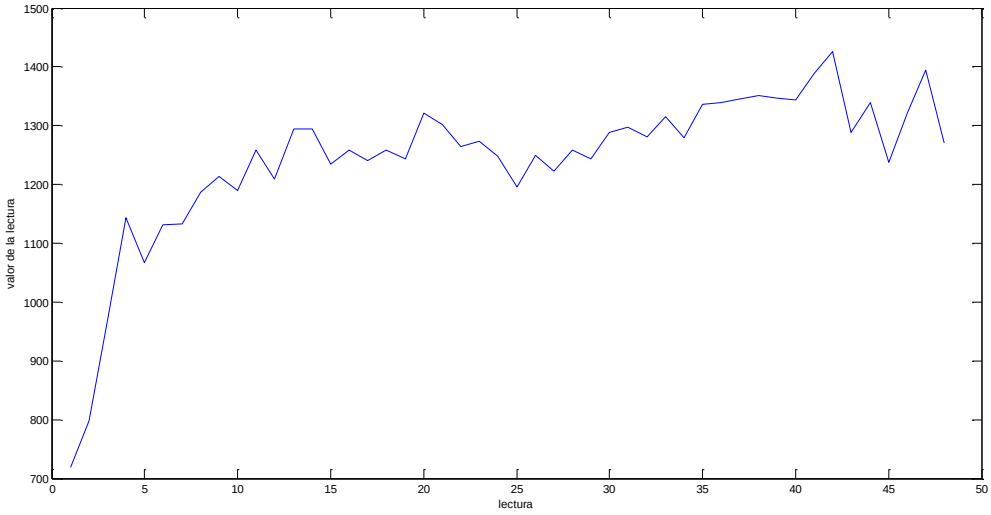
En esta imagen, podemos observar el primer error cometido en el desarrollo de este proyecto. Si observamos la gráfica, observaremos que ambos materiales saturan la respuesta del sensor, dando como resultado el mismo valor en la parte central de la lectura. Este problema produce una pérdida muy importante de información a la hora de diferenciar materiales.



B. TIEMPO ABIERTO

Una vez hemos configurado la ganancia del sensor, hemos realizado pruebas para establecer el tiempo que la garra debe permanecer abierta para obtener un buen resultado. Para ello, se ha creado un programa en Java que realice una lectura del sensor durante 5 segundos, variando el tiempo que el programa espera entre lectura y lectura (tiempo que la garra permanece abierta). En esta prueba, los tiempos que la garra se ha mantenido abierta han ido variando desde los 10 milisegundos hasta los 1000 milisegundos, incrementando en un valor de 20 milisegundos.

Con la ejecución de este programa, obtenemos la siguiente gráfica:



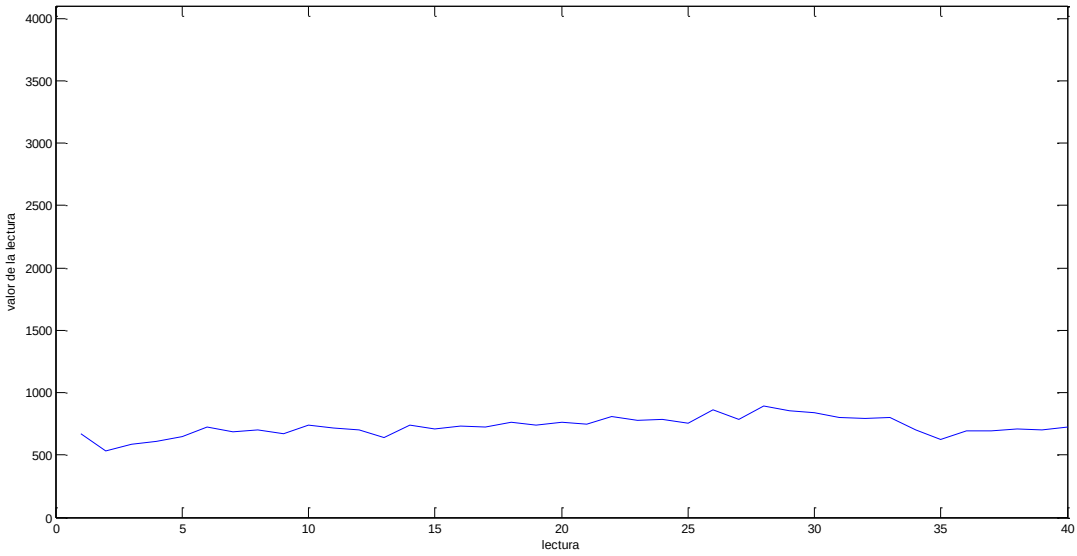
De los valores probados, se ha escogido un tiempo de apertura que se ha seleccionado son 500 milisegundos, ya que es más o menos en este instante de tiempo cuando la lectura comienza a ser lineal.

Al igual que en el punto anterior, a la hora de seleccionar el tiempo de apertura también se han cometido errores. El primero y más grave de ellos es no haberse dado cuenta de la curva que presenta la respuesta del sensor, lo que hace, que sea necesario establecer un tiempo más alto para que el sensor estabilice su respuesta (en la versión final del programa, se ha utilizado un tiempo de 1 minuto). De este error, nos damos cuenta también de la falta de una prueba de repetitividad, en la que nos habríamos dado cuenta de este problema. El segundo error que se ha cometido viene dado al establecer un rango de prueba tan pequeño, ya que de haber establecido unos límites mayores a la hora de realizar las pruebas, se podría haber observado la respuesta que presenta el sensor.

C. TIEMPO CERRADO

Esta prueba se ha realizado de la misma forma que la anterior, fijando el tiempo que la garra se encuentra abierta al valor deducido en la prueba b. Las tomas, se han realizado variando el tiempo que la garra permanece cerrada desde los 1000 milisegundos hasta los 5000 milisegudos, variando entre muestra y muestra 100 milisegundos.

En la siguiente imagen se puede ver la gráfica:



Como se puede apreciar, la respuesta es muy lineal (algo que en realidad no corresponde a la salida real del sensor). Es por esto, que se ha escogido el mínimo tiempo, es decir, 1 segundo, para establecer el tiempo que la garra debe permanecer cerrada al tomar la muestra.

En este punto, el error ha sido un fallo de programación. Revisando la programación, nos damos cuenta de que el tiempo que hemos variado en la prueba no tiene repercusión en la lectura, ya que este retardo se realiza tras la lectura del sensor, y no antes, como cabría esperar.

7 - DISEÑO DE LA RED NEURONAL

Una vez hemos configurado los parámetros del sensor, tendremos listo el programa de Java que nos permitirá realizar la toma de muestras correctamente. El siguiente problema a resolver es el diseño de la red neuronal, desde la adquisición de los datos, pasando por la transformación de los mismos hasta la generación de nuevos parámetros y el posterior entrenamiento de la red.

A. LOS DATOS DE ENTRADA DE LA RED NEURONAL

Cuando leemos en la bibliografía sobre las redes neuronales, en la mayoría de los casos vemos cómo los datos que se utilizan como entradas a una red neuronal, vienen dados no por los datos originales (en este caso la salida del sensor), sino por valores estadísticos que se calculan a partir de los originales.

En este caso en concreto, tras una toma de 150 muestras (con el programa configurado con los parámetros explicados anteriormente), se han calculado distintos valores estadísticos sobre cada una de las muestras, concretamente:

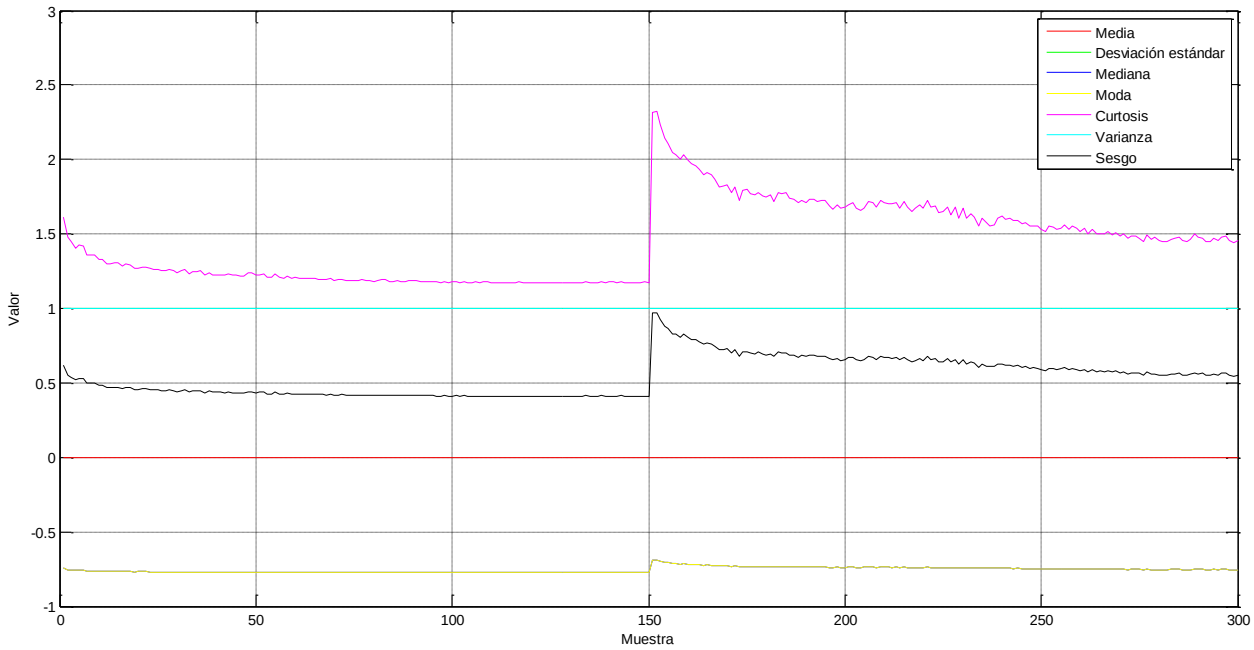
- Media
- Desviación estándar
- Mediana
- Moda
- Curtosis
- Varianza
- Sesgo

En primer lugar, previamente al cálculo de los valores estadísticos, se ha seguido un proceso de normalización de los valores, para lo que se ha hecho uso de la siguiente ecuación:

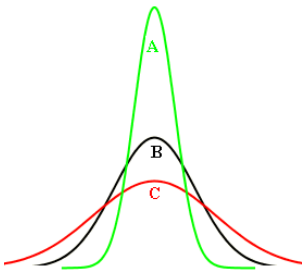
$$y_k = \frac{x_k - m_x}{\sigma_x}$$

Donde m_x y σ_x son el valor medio y la desviación estándar de la variable x. Con esta transformación, obtendremos una nueva variable cuyo valor medio valdrá 0, y cuya varianza será 1.

Tras la obtención de las nuevas variables, se pueden calcular los valores estadísticos. En la siguiente imagen, se pueden ver estos valores para dos materiales distintos (densidad 20 y 60).



Como se puede ver en la imagen anterior, sólo existen dos valores que diferencien una muestra de la otra, siendo estos la curtosis y el sesgo. Si nos paramos a pensar en lo sucedido, nos daremos cuenta de que este comportamiento es algo normal, dado que ambas variables expresan, la forma que tiene la curva (ver siguiente imagen).



De la imagen en la que se muestran los valores estadísticos de las muestras, sacamos la conclusión de que lo mejor será establecer como entradas de la red neuronal los valores de la curtosis y del sesgo.

B. DISEÑO CON NEUROPHSTUDIO

En un principio, cuando el objetivo del proyecto pasaba por crear toda la aplicación en el lenguaje de programación Java, se realizaron todas las pruebas con el uso de dicho lenguaje.

Desde el software Eclipse, se realizaron todas las capturas, además del cálculo de los valores estadísticos. El proceso que seguía el programa era el siguiente:

1- Se realizaba una única lectura del sensor.
2- El hilo encargado de leer los mensajes del sensor creaba un nuevo mensaje.
3- Durante la creación de este nuevo mensaje, se realizaba el cálculo de los valores estadísticos (curtosis y sesgo), con las lecturas de una única columna del sensor, concretamente, la del medio.
4- Estos valores, eran guardados en un fichero.

Una vez tenemos todos los valores que establecen las entradas de la red neuronal, se utiliza la interfaz gráfica de NeurophStudio para crear la red neuronal.

Dado que la creación de esta red neuronal necesitaba muchas pruebas, se implementó toda la creación de la red neuronal (entrenamiento incluido), en un programa independiente Java. Este programa, creaba redes neuronales variando el número de neuronas pertenecientes a la capa oculta de la red, realizaba el entrenamiento, y comprobando durante cada iteración de éste, cuál era el error obtenido.

En primer lugar, se propuso una red para distinguir dos materiales, concretamente, los materiales de densidades 20 y 40. Para ello, se realizó una adquisición de 1000 muestras de ambos materiales, para más tarde ejecutar el programa antes mencionado, en este caso, simplemente con una red con 15 neuronas en la capa oculta.

Tras la ejecución, nos damos cuenta del número de iteraciones necesarias para obtener un error menor a 0.0001, siendo este número de 21060. Como salida deseada de la red, se ha utilizado un único valor, siendo este 0 para el material de densidad 20, y 1 para el material de densidad 60.

Una vez se tiene la red entrenada, se comprueba el resultado de la red con los valores reservados para test. Con este propósito, se han reservado 750 muestras, 375 de cada material. El resultado de la red ha sido muy bueno, dando una tasa de aciertos del 100 %.

Dado que se han obtenido buenos resultados con esta red, seguimos con la creación de nuevas redes, ahora, para distinguir entre los tres materiales. Para ello, se ha realizado una modificación en el programa que realiza el diseño de la red neuronal. Ahora se han creado distintas redes, variando el número de neuronas que componen la capa oculta en un valor que va desde las 10 neuronas hasta las 20 neuronas, incrementando entre prueba y prueba en una unidad el número de estas. Cuando se obtiene una red entrenada, se ha comprobado automáticamente cual era la tasa de aciertos de la red con las muestras reservadas para test, si el resultado es mejor que alguno de los anteriores, se actualiza el número de neuronas que mejor resultado ofrece. A final del programa, se informa cual es la mejor opción.

Ahora, se tienen tres materiales, por lo que el resultado de la red no puede quedarse en un valor. Se han actualizado el número de neuronas que dan el resultado de la clasificación, ahora no lo indica un valor, sino dos. La combinación deseada es la siguiente:

- Material densidad 20 0 0
- Material densidad 40 0 1
- Material densidad 60 1 0

El resultado de la ejecución, ha sido un error de 9.94 E-4 utilizando para ello 17 neuronas en la capa oculta.

Con estos resultados, se ha paralizado la mejora de la red neuronal para comprobar el resultado ante muestras a tiempo real, para lo que se ha creado un programa en Java que permita tomar una muestra, y mostrar en modo texto el tipo del material utilizado. Sin embargo, tras realizar esta prueba, se han visto unos resultados muy dispares a los mostrados anteriormente. En este caso, la tasa de aciertos sufrió un descenso muy algo, cayendo por debajo del 30 %.

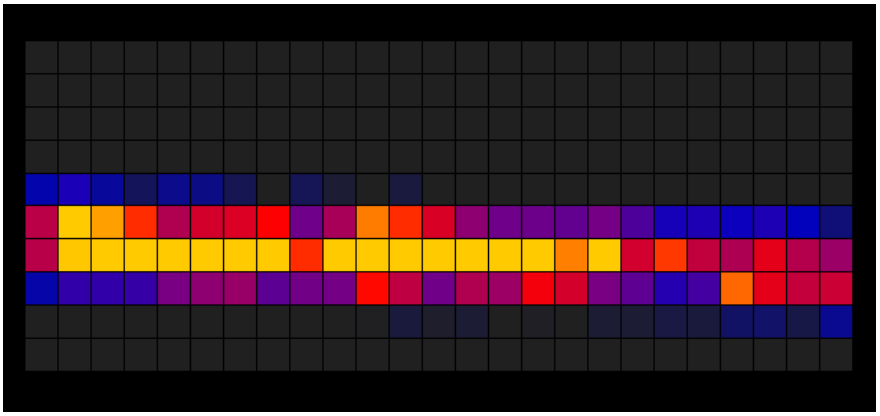
C. ALGORITMO DE MARZULLO

Dadas los malos resultados obtenidos hasta ahora, se ha decidido aplicar algoritmos que ayuden a mejorar los resultados obtenidos anteriormente. Uno de estos algoritmos, es el llamado Algoritmo de Marzullo.

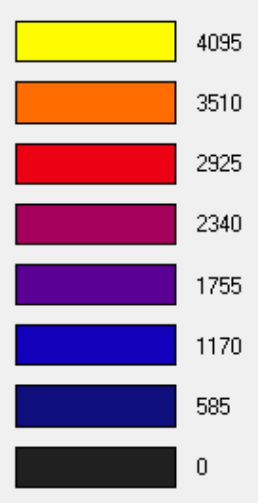
El planteamiento de la matriz del sensor que se ha hecho para poder aplicar este algoritmo es el siguiente: dado que la matriz del sensor está compuesto por 10 filas de 25 columnas cada una, y el material se encuentra situado a lo largo del sensor, se puede pensar

que cada fila, está compuesta por 25 sensores redundantes, que ofrecen una medición que debería ser la misma.

En la siguiente imagen, podemos ver las lecturas del sensor que ofrece el software WTS Commander:



Este software ofrece un visualización gráfica con colores que representan la fuerza que es está ejerciendo sobre cada uno de los sensores que componen la matriz. La gama de colores que representa la presión es la que sigue:



Por lo tanto, una celda de la matriz de color amarillo, representa una celda sobre la que se está ejerciendo la mayor presión que el sensor es capaz de leer, mientras que una celda de color negro, hace referencia a una celda en la que no se está ejerciendo ninguna presión.

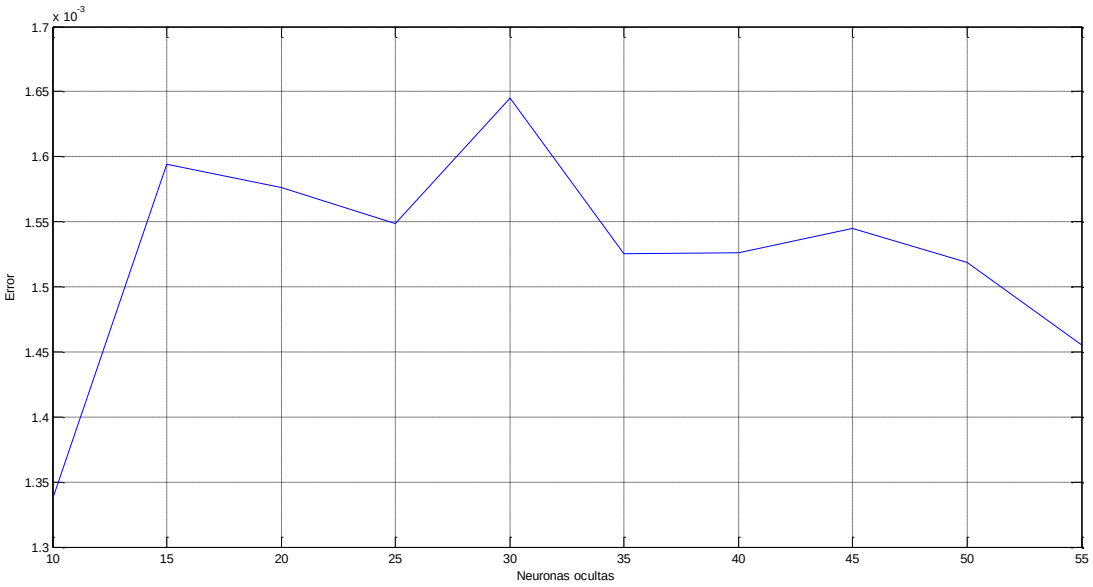
En la imagen en la que se muestra el resultado de la presión que ejerce un cilindro de los utilizados en el laboratorio, se puede ver cómo existe una fila que representa el centro

del material, dando la mayoría de las celdas que componen esta fila unos valores muy cercanos entre ellos.

Es por esto, que se puede aplicar el razonamiento explicado con anterioridad, en el que cada fila ofrece un conjunto de sensores que, una vez aplicado algún algoritmo capaz de realizar algún tipo de operación que dé como resultado un único valor cuya confianza sea mayor que cualquiera de las confianzas ofrecidas por cada uno de los sensores por sí solos.

A la hora de aplicar el algoritmo, se ha utilizado el lenguaje de programación Java para realizar un programa que lee de un archivo en el que se encuentran los valores de la matriz completa del sensor. Con estos valores, se ha aplicado el algoritmo de Marzullo para calcular cual es el rango de valores que tiene el máximo número de solapamientos. Para ello, ya que de cada sensor se tiene un único valor, se ha dado un margen a cada señal, por la cual, cada sensor tiene un margen cuyo centro es el valor ofrecido por la lectura, y cuyos límites se calculan sumando el valor del margen a dicha lectura. Una vez tenemos calculado el rango de valores que tiene el máximo número de solapamientos en cada fila, sacamos el valor del conjunto calculando la media de este rango, con lo que nos queda una única columna calculada a partir de toda la matriz del sensor, que representa el material. Esta columna, es guardada en otro fichero que contiene todos los valores calculados de todas las lecturas.

Una vez tenemos creado el fichero, pasamos a ejecutar la aplicación que genera las redes neuronales, esta vez introduciendo una pequeña modificación por la cual, guardamos en un fichero los errores obtenidos con cada una de las redes. Estos errores se pueden ver en la siguiente imagen:



Como se puede ver, el mejor resultado es el obtenido utilizando 10 neuronas. Las propiedades del diseño de esta red son:

- Clasificación de dos materiales de densidades 40 y 60.
- Salida esperada: 0 para material de densidad 40 y 1 para material de densidad 60.
- Número de muestras para entrenamiento: 750 para cada material.
- Número de muestras para test: 300 para cada material.
- Error obtenido en el entrenamiento: 1.33E-3
- Número de errores en el test: 8 de 600

Una vez se ha obtenido la red neuronal con unos resultados aceptables, se ha pasado, una vez más, a comprobar los resultados de la red con datos reales, haciendo pruebas en tiempo real. Los resultados ofrecidos por esta red neuronal, aplicando a las muestras el algoritmo de Marzullo, han vuelto a ser malos, obteniendo una solución incapaz de distinguir correctamente entre los materiales de los cilindros de prueba.

D. FILTRO DE KALMAN

Como una vez más, se han obtenido resultados incorrectos, se ha pensado en aplicar el filtro de Kalman a las muestras, siguiendo la misma filosofía que cuando aplicábamos el algoritmo de Marzullo.

Ahora, nuestra intención al aplicar este filtro, sigue siendo tratar una fila como un conjunto de sensores que está realizando mediciones que deberían dar el mismo resultado, sin embargo, entre las mediciones de los distintos sensores, existe un ruido que se intenta eliminar aplicando este algoritmo.

Se sabe que este algoritmo es muy complejo, y que puede ofrecer mejoras muy significativas en este campo, sin embargo, la situación, nos impide poder aplicar el filtro de forma realmente efectiva. Esta situación es la ofrecida por el accionamiento de la garra, dado que este accionamiento es neumático, del tipo todo o nada, no se puede especificar un modelo de la misma para obtener buenos resultados. Tampoco se puede aplicar el modelo del sensor, ya que el fabricante no lo pone a nuestra disposición.

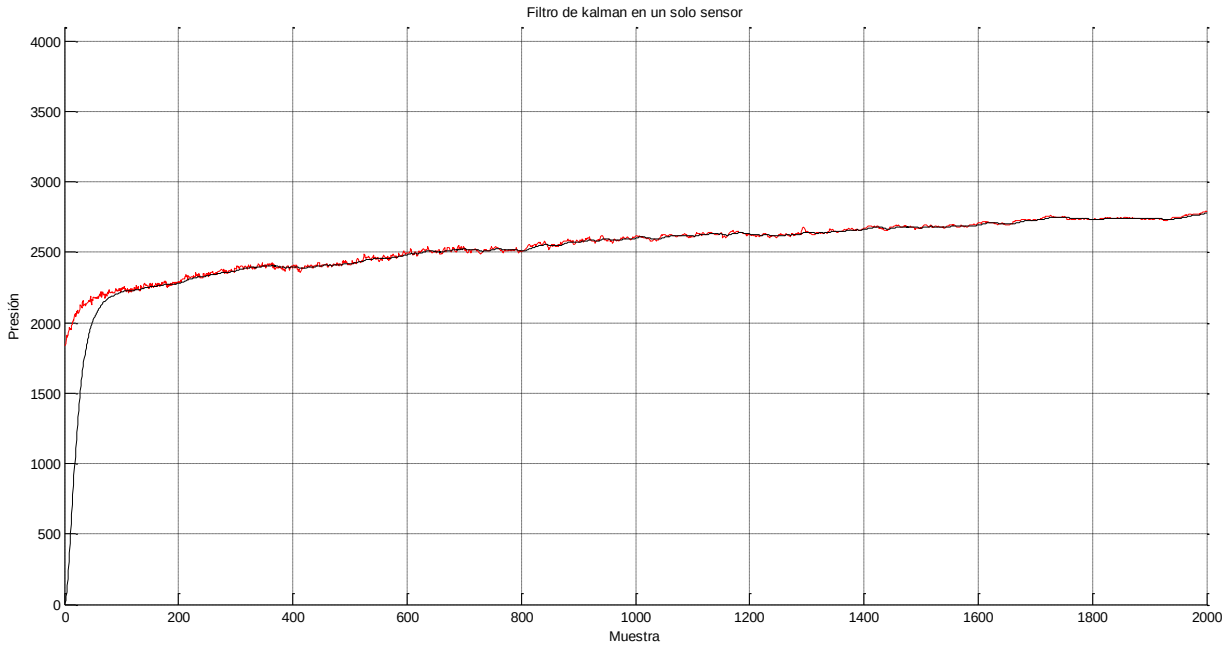
Con todo esto, el filtro que se ha aplicado es el más sencillo que el algoritmo permite aplicar, en el que las matrices utilizadas son:

- A = 1
- B = 0
- P = 0.0001
- X = 0
- Q = 0.005
- H = 1
- R = 1

Como es de esperar, los resultados de este algoritmo, siguiendo la filosofía explicada, no funciona, al igual que en el caso del algoritmo de Marzullo, los resultados son muy malos en pruebas realizadas en tiempo real.

Una vez se sabe que esta forma de aplicar el algoritmo no funciona, se cambia la forma en la que se realizan las lecturas del sensor. A partir de este momento, en vez de tomar una lectura cada vez que se cierra la garra que sostiene el sensor, se realiza una adquisición de 2000 muestras durante, con un retardo entre toma y toma de 200 milisegundos.

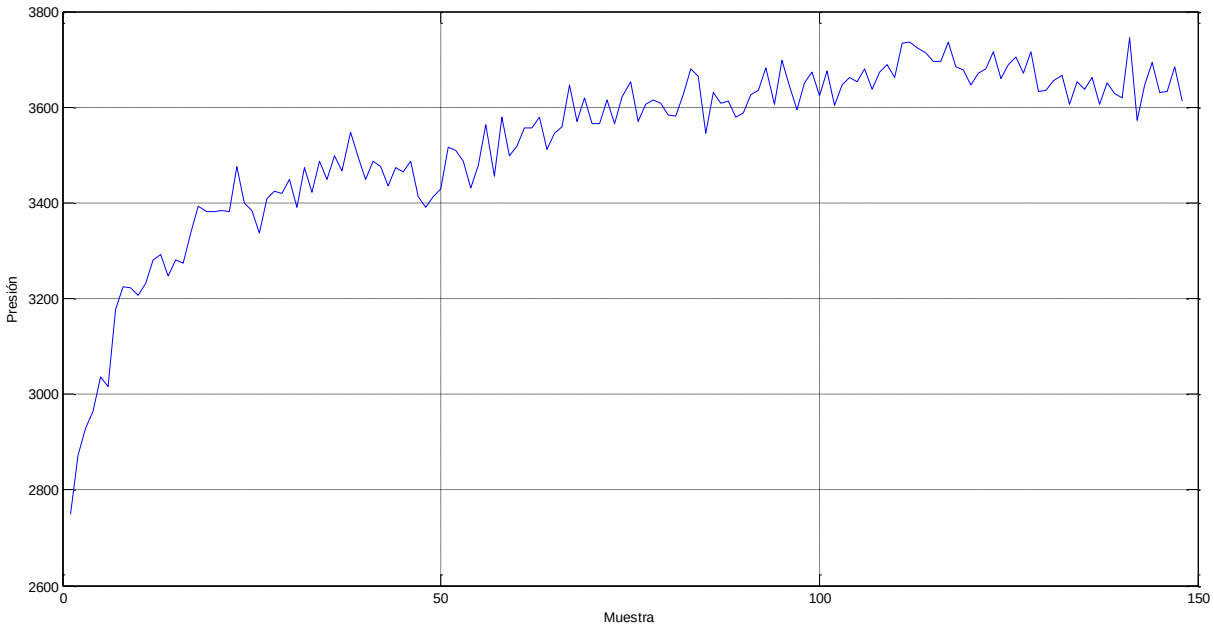
Con este método, se ha obtenido una adquisición con 2000 lecturas de los 250 sensores que componen el sensor. En este momento, se aplica el filtro de Kalmana uno de los sensores a lo largo del tiempo, obteniendo una gráfica como la que se muestra en la imagen siguiente:



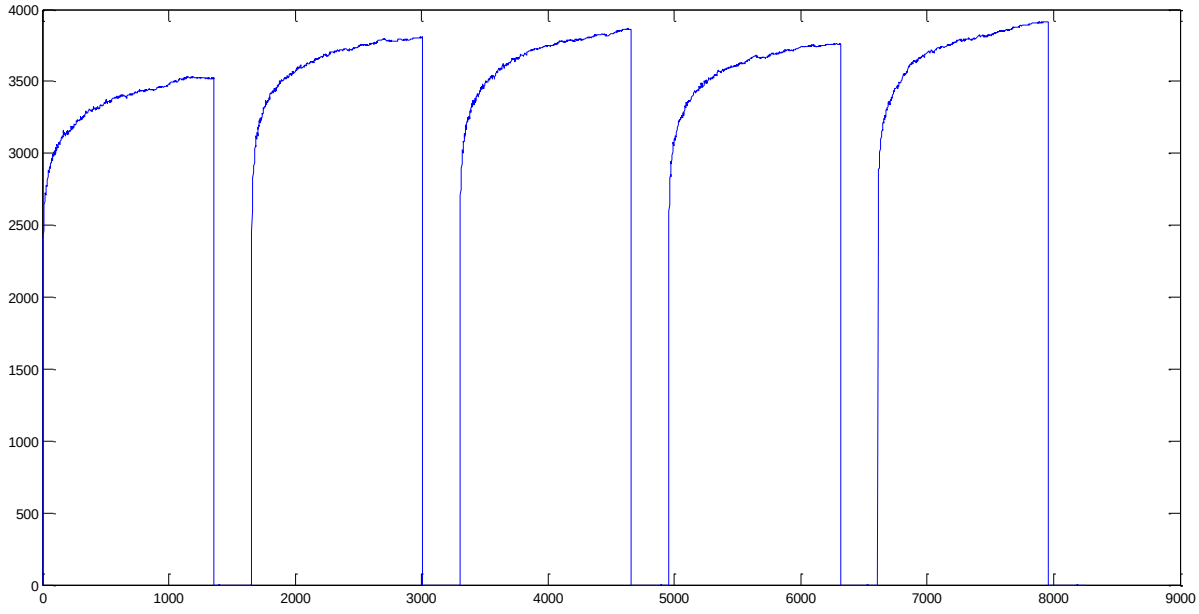
En este momento, al observar la gráfica, podemos ver que su respuesta no es como se esperaba. Lo que se esperaba, era que tuviera una respuesta lineal en el tiempo, sin embargo, como se puede ver en la imagen, el valor crece alrededor de 1000 unidades.

E. CURVA DE REFERENCIA

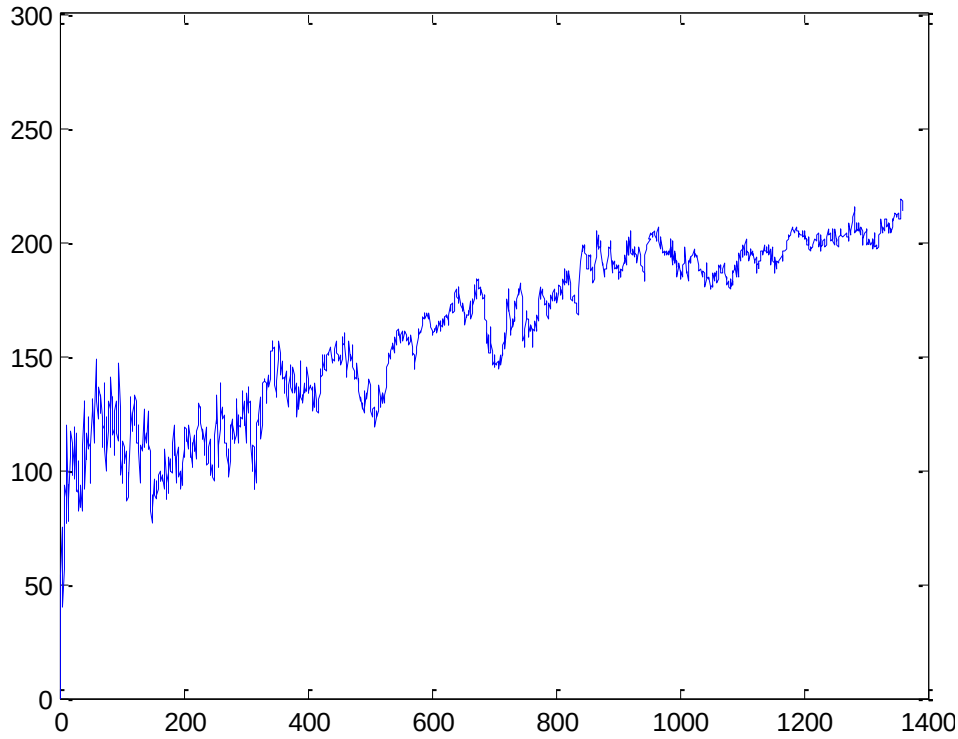
Es en este momento, cuando nos damos cuenta del problema que se ha estado sufriendo desde el principio: la escasa linealidad que ofrece la respuesta del sensor. Esta escasa linealidad, hace que las muestras del mismo material varíen de un instante de tiempo a otro, es decir, si graficamos los valores que han adquirido, por ejemplo, 150 muestras del mismo sensor, con el mismo material, veremos la gráfica que se muestra en la imagen siguiente:



Es cuando realizamos esta prueba, cuando nos damos realmente cuenta de nuestro problema, el que hemos ido arrastrando desde el principio. Como podemos ver en la imagen anterior, el valor de la presión de un mismo punto de la matriz va creciendo en el tiempo. Si realizamos una prueba más, en la que realizamos medidas esperando un tiempo de 5 minutos entre lectura y lectura, con una duración de cada una de ellas de 1 segundo, podremos ver la respuesta del sensor:



Como se puede ver en la imagen anterior en este caso, podremos obtener resultados más lineales si conocemos la curva que ofrece el sensor en el tiempo que éste se mantiene en contacto con la pieza. Para ello, realizamos una captura de 6 minutos. Una vez realizada la toma, la procesaremos con MATLAB para obtener la curva de respuesta, en la que el instante inicial tendrá valor 0, y que irá creciendo en la forma que marca el sensor. Dicha curva puede verse en la imagen siguiente:



Con esta prueba, obtendremos como resultado 10 gráficas, una por cada sensor de la columna del centro (seguimos pensando que únicamente con los sensores que componen la columna del centro de la matriz podemos obtener resultados).

El objetivo de generar estas curvas, es obtener el comportamiento que hay que restar a la lectura que ofrece el sensor para obtener una respuesta lo más lineal posible. Sin embargo, no podremos utilizar directamente esta curva, dado que el sensor presenta ruido de una lectura a otra, como se puede ver en la imagen anterior, presentando picos en su resultado. Para ello, se han probado dos soluciones: primero se ha utilizado una interpolación de los puntos que componen la gráfica, y segundo se ha utilizado el algoritmo de regresión lineal para obtener otra solución.

- **Interpolación**

La intención de crear una interpolación de la curva de referencia, es obtener una curva en la que el ruido se haya eliminado, teniendo además, una serie de puntos que son los que utilizaremos para restar y acomodar la respuesta del sensor a partir de la curva de referencia. Con esta solución, se piensa que computacionalmente el comportamiento va a ser muy bueno, ya que una vez calculados los puntos, éstos se podrán guardar en un fichero, y el programa que realiza la lectura del sensor tan sólo tendrá que leer estos puntos del fichero, para crearse una tabla con los valores, por lo tanto, cada vez que realice una lectura, no tendrá que hacer ningún cálculo adicional.

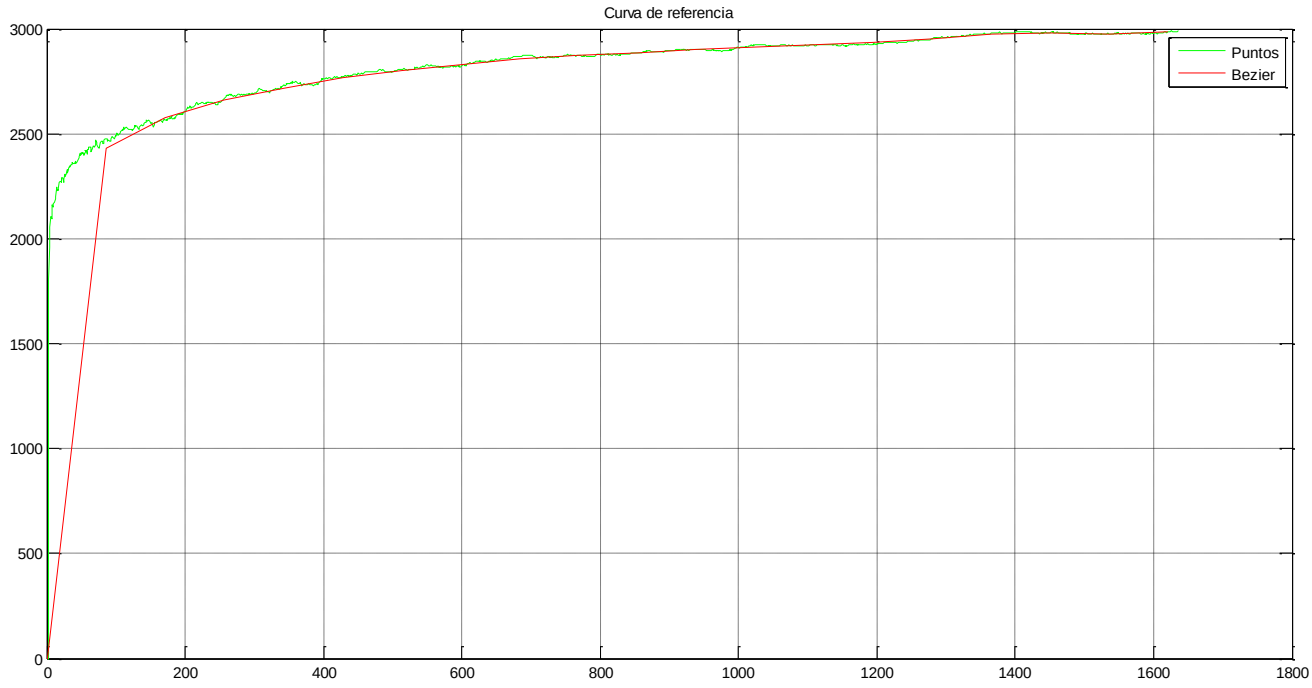
Utilizando MATLAB, se han calculado los puntos de esta curva. Para ello, se ha utilizado una función encontrada en la página de MathWorks (www.mathworks.es), que de forma gratuita ofrece una serie de funciones para poder utilizar en tus programas, con licencia BSD. En el siguiente fragmento de código, se puede ver la función mencionada:

```
% % % --------------------------------
% % % Author: begtostudy
% % % Email : begtostudy@gmail.com
% % % --------------------------------
function Q=Bezier(P,t)
% Bezier interpolation for given points.
%
%Example:
% P=[292 280 321 356;
% 196 153 140 148;
% -56 75 140 248];
%
% t=linspace(0,1,100);
% Q3D=Bezier(P,t);
%
% figure
% plot3(Q3D(1,:),Q3D(2,:),Q3D(3,:),'b','LineWidth',2),
% hold on
% plot3(P(1,:),P(2,:),P(3,:),'g:','LineWidth',2)       % plot control polygon
% plot3(P(1,:),P(2,:),P(3,:),'ro','LineWidth',2)     % plot control points
% view(3);
% box;

for k=1:length(t)
   Q(:,k)=[0 0 0]';
   for j=1:size(P,2)
      Q(:,k)=Q(:,k)+P(:,j)*Bernstein(size(P,2)-1,j-1,t(k));
   end
```

```
end
end

function B=Bernstein(n,j,t)
   B=factorial(n)/(factorial(j)*factorial(n-j))*(t^j)*(1-t)^(n-j);
end
```

En este código, se puede ver además, un ejemplo de cómo usarla. En él, se puede comprobar la simplicidad de uso que tiene esta función, ya que tan sólo hay que pasar los puntos que representan la curva en 3 dimensiones, y una variable de puntos que representan los puntos en los que se va a realizar el cálculo.

Con esta función, se han obtenido los resultados mostrados en la imagen siguiente:



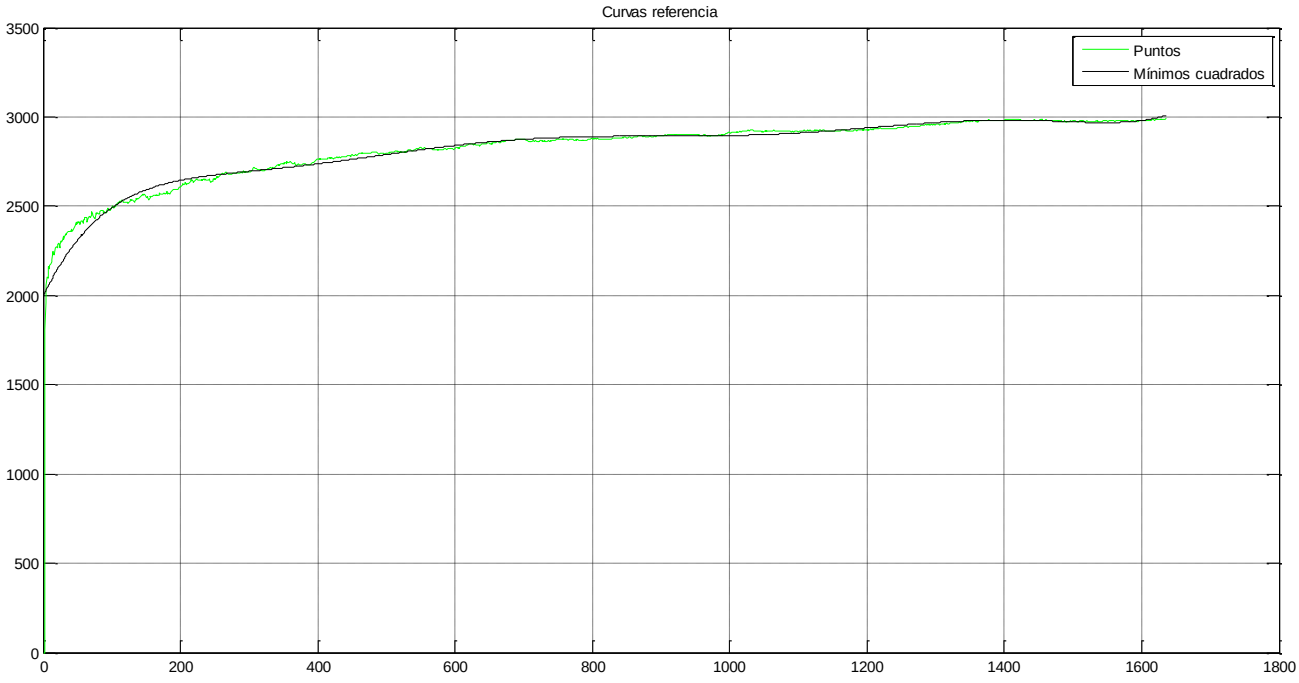
Se puede apreciar en esta imagen, el comportamiento fiel de la curva de referencia a los puntos de control capturados por el sensor. Sin embargo, en la parte inicial de la curva, el comportamiento no llega a ser lo ajustado que cabría esperar. Esto, cuando se aplique a la lectura del sensor, dará como resultado un pico de valores altos, a los que se les ha restado un valor menor del que debería para lograr un resultado más lineal.

- **Mínimos cuadrados**

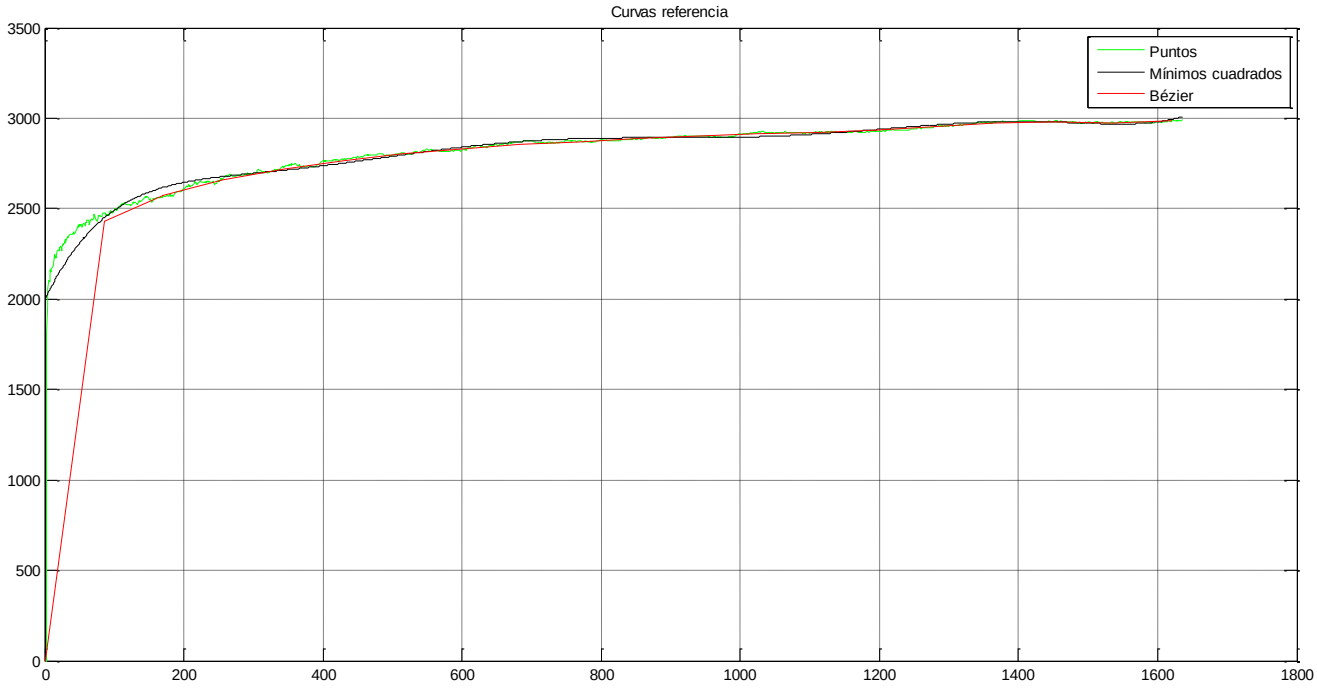
Con esta solución, la filosofía sigue la misma línea que con la curva Bézier. Con esta opción, se pretende calcular una fórmula que represente la curva que elimine el ruido de los puntos de la lectura del sensor, para poder utilizar esta curva cuando se realicen nuevas lecturas y se pueda conseguir, de esta forma, una mejora en la respuesta del sensor.

El entorno de desarrollo MATLAB, ofrece entre sus comandos uno dedicado a este efecto, crear a partir de una serie de puntos, una fórmula que represente el resultado de aplicar mínimos cuadrados a dichos puntos. Este comando se llama 'polyfit'. Esta función, toma como argumentos los valores que representan el eje X de los puntos que representan la curva a partir de la cual queremos aplicar el algoritmo, el valor de dichos puntos en el eje Y, y un valor que expresa el grado de la función resultado.

En nuestro caso, se han probado funciones de distintos grados, viendo en cada caso, cómo se ajusta la función resultado a la curva que nos interesa. Finalmente, con una función de grado 7, se obtiene una curva que ofrece una respuesta muy similar a la ofrecida por el sensor, que nosotros utilizaremos como referencia. En la siguiente imagen, se puede ver cómo se ajustan ambas curvas:



Una comparación de ambas curvas, la conseguida mediante Bézier, y la calculada mediante mínimos cuadrados, se puede ver en la siguiente imagen:



En la imagen anterior, se puede ver cómo la parte inicial es la que ofrece una diferencia más sustancial de las dos soluciones. Es en esta parte, donde la curva conseguida mediante mínimos cuadrados ofrece una solución mejor que la conseguida mediante Bézier.

F. DISEÑO CON MATLAB

Para realizar diseño de las nuevas redes neuronales, se ha cambiado el entorno de desarrollo que estábamos utilizando (NeurophStudio). Ahora, debido a las buenas opiniones dadas por conocidos, amigos y compañeros, se ha pasado a utilizar MATLAB. La verdad, es que después de utilizar ambos entornos, y conociendo las ventajas y los inconvenientes entre amos programas, he de reconocer que el toolbox ofrecido por MATLAB es más amigable, y sobre todo, más rápido a la hora de realizar el entrenamiento que el competidor NeurophStudio.

Además, a la hora de hacer el entrenamiento, MATLAB ofrece unos resultados mejores, ya que los resultados mostrados, y los conseguidos en las pruebas a tiempo real que se han realizado tienen una semejanza muy buena.

Una vez se ha tomado la decisión de continuar con la creación de una red neuronal cuyos datos de entrada sean pre-procesados utilizando la función de mínimos cuadrados,

nos ponemos a estudiar las opciones que tendremos para crear la red, la metodología a seguir desde que se realiza la lectura de la matriz del sensor, hasta que los datos que contienen el resultado indican qué material se está tratando.

En este punto, es donde nos encontramos la problemática que surge de cómo utilizar el algoritmo de los mínimos cuadrados. Las tomas del sensor se deberían tener unas características muy lentas para poder aplicar esta metodología. Estas características son:

- Mantener cerrada la garra durante 5 minutos realizando continuamente lecturas del sensor
- Lectura del sensor cada 200 milisegundos
- Espera entre ciclos de lecturas de 1 minuto

Pensando en estas características, se hace muy difícil el uso de este sensor en entornos reales, ya que para poder hacer este uso, se necesitarían ciclos mucho más cortos de los obtenidos.

Con esto, se sabe que si se llegan a obtener buenos resultados, la siguiente prueba tendría como objetivo disminuir los tiempos lo máximo posible para poder obtener resultados aceptables.

En este punto nos planteamos las siguientes posibilidades:

1. Una posible metodología a seguir puede ser realizar el entrenamiento de la red con muestras que se han linealizado con la función de mínimos cuadrados estudiada. Para ello, el programa encargado de realizar las muestras debería comprobar, en cada lectura del sensor, qué valor da como resultado la red neuronal. Si la red neuronal da distintos resultados, es decir, indica que el material utilizado en un mismo ciclo de toma de muestra, se pueden establecer unos valores que sean necesario alcanzar hasta poder afirmar que el material que se está utilizando es de un tipo dado con cierta confianza. Hasta que no se alcance este límite, se seguirá tomando lecturas del lector.

   Con esta solución, los pasos a seguir en este momento serían los que siguen:

   1- Realizar una serie de tomas a cada uno de los tres materiales de los que estamos interesados.

   2- Pre-procesar estas lecturas, para conseguir unas muestras más lineales en el tiempo, a partir de la función conseguida con mínimos cuadrados.

3- Crear y entrenar una red con estos valores hasta conseguir unos resultados aceptables.
4- Modificar el programa que realiza las lecturas, para que en cada una de ellas, linealice la muestra.
5- Una vez se tiene la muestra linealizada en el tiempo, se debe buscar un método de comunicación entre Java y MATLAB que permita conocer el resultado de la red neuronal anteriormente entrenada.
6- Si tras una serie de resultados, en forma de porcentaje, se obtiene un nivel alto de confianza, dar la muestra como reconocida y seguir con la siguiente.

Con este procedimiento, nos encontramos el problema de buscar una pasarela entre MATLAB y Java. Realizando una búsqueda en internet, vemos que hay varias opciones, pero se desconoce el rendimiento conseguido con esta comunicación, y teniendo en cuenta la velocidad que debe conseguir el proceso, es descartada.

2. Para solucionar el problema que surge con la necesidad de una velocidad de transmisión tan alta, surge otra opción. En esta segunda opción, la idea es guardar todas las lecturas del sensor en un fichero, para posteriormente, cuando se terminada de realizar la toma que, en principio, tarda 5 minutos, realizar una llamada a MATLAB, para que éste se ocupe tanto del preprocesado, como de la clasificación del material del que se están tomando las muestras.

   Esta vez, los pasos a seguir son los siguientes:

   1. Realizar una serie de tomas a cada uno de los tres materiales de los que estamos interesados.
   2. Pre-procesar estas lecturas, para conseguir unas muestras más lineales en el tiempo, a partir de la función conseguida con mínimos cuadrados.
   3. Crear y entrenar una red con estos valores hasta conseguir unos resultados aceptables.
   4. Modificar el programa que realiza las lecturas, para que una vez finalice el tiempo de toma de muestras, realice una llamada a MATLAB para que siga con el proceso.

5. MATLAB debe pre-procesar las muestras, pasarlas por la red neuronal para determinar el material que se ha utilizado, y volver a llamar al programa Java sobre el resultado.

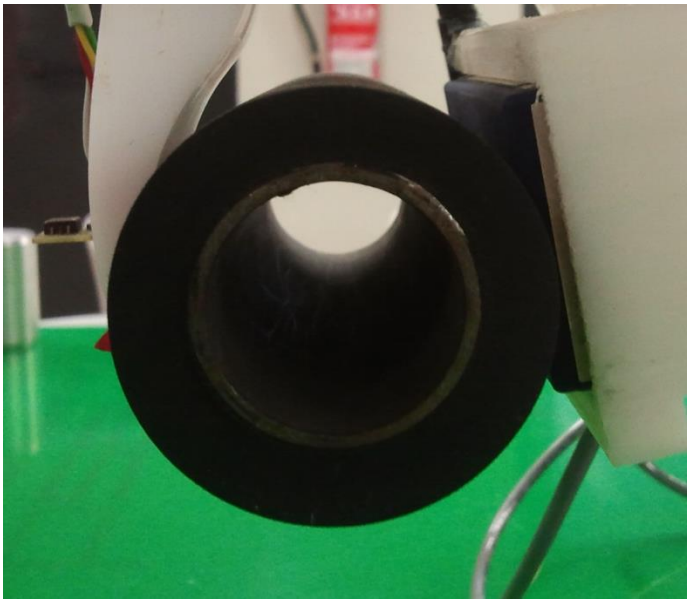
Con esta solución, se elimina el problema que surge al utilizar la primera idea, sin embargo, ofrece otro problema (menos grave que el anterior). En este caso, todas las adquisiciones de las muestras deberán llevar una cierta cantidad de tiempo predefinido. Si el algoritmo es bueno y no necesita tal cantidad de tiempo, no habrá forma de solucionarlo una vez el programa se esté ejecutando.

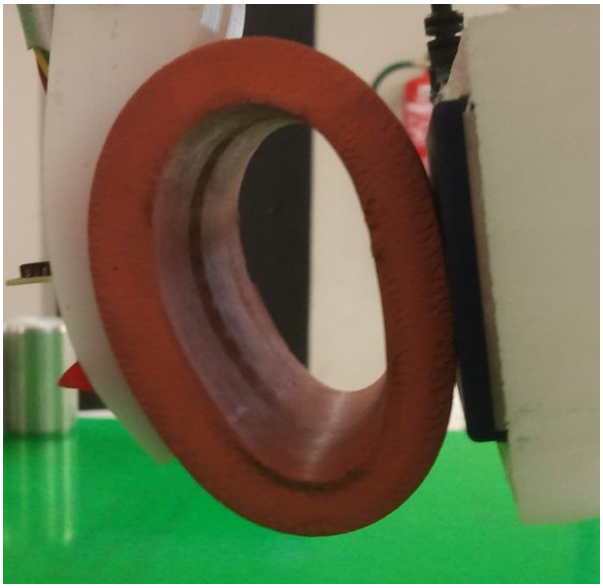
Tras el estudio de las posibilidades que tenemos para realizar la implementación, nos decantamos por la segunda opción. Sin embargo, también nos damos cuenta del siguiente razonamiento:

Con esta metodología, tenemos que esperar 1 minuto entre ciclos para conseguir que el sensor estilice su respuesta inicial, ya que a partir de esta respuesta inicial, se pueden utilizar algoritmos que minimicen la no linealidad. Tras esta espera, debemos esperar otros 5 minutos realizando lecturas del sensor, para, una vez terminado el tiempo de lecturas, pre-procesar las muestras para eliminar el problema que ofrece el sensor: la linealidad.

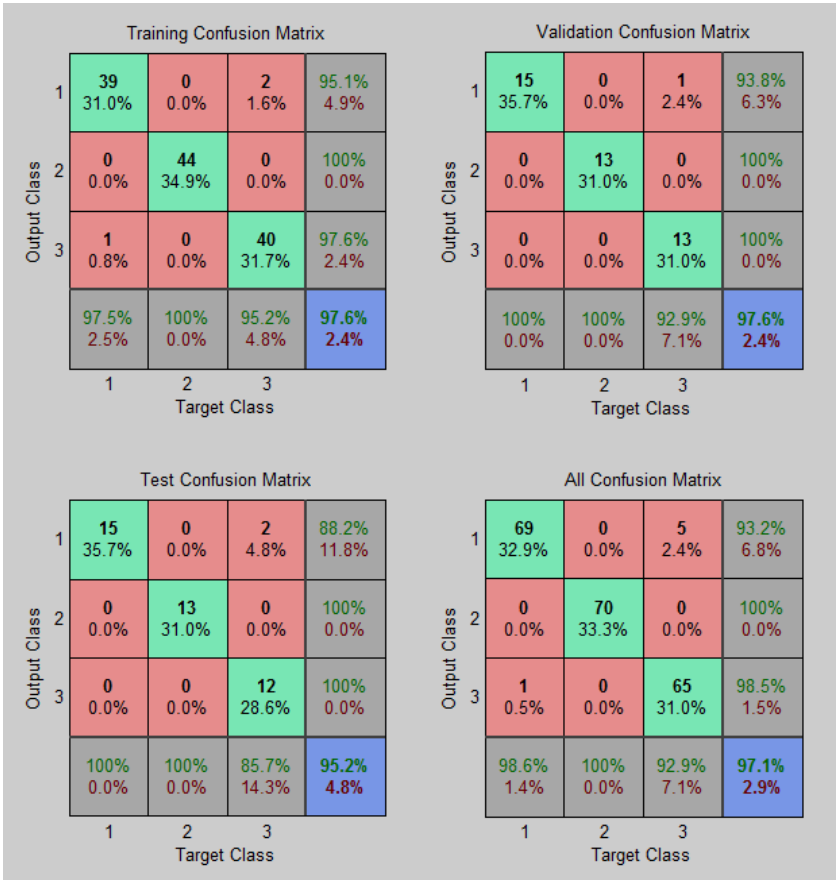
Dado que la espera de 1 minuto entre ciclo y ciclo no la vamos a eliminar (al menos de momento), y tras este tiempo, los valores del mismo material serán similares, es posible realizar un procedimiento mediante el cual, el tiempo del ciclo en el que se están guardando las lecturas del sensor sea muy breve (del orden del segundo). De esta forma, el tiempo de ciclo entre tomas se verá reducido de los 6 minutos, con la solución planteada anteriormente, al minuto y 1 segundo, con la solución planteada ahora.

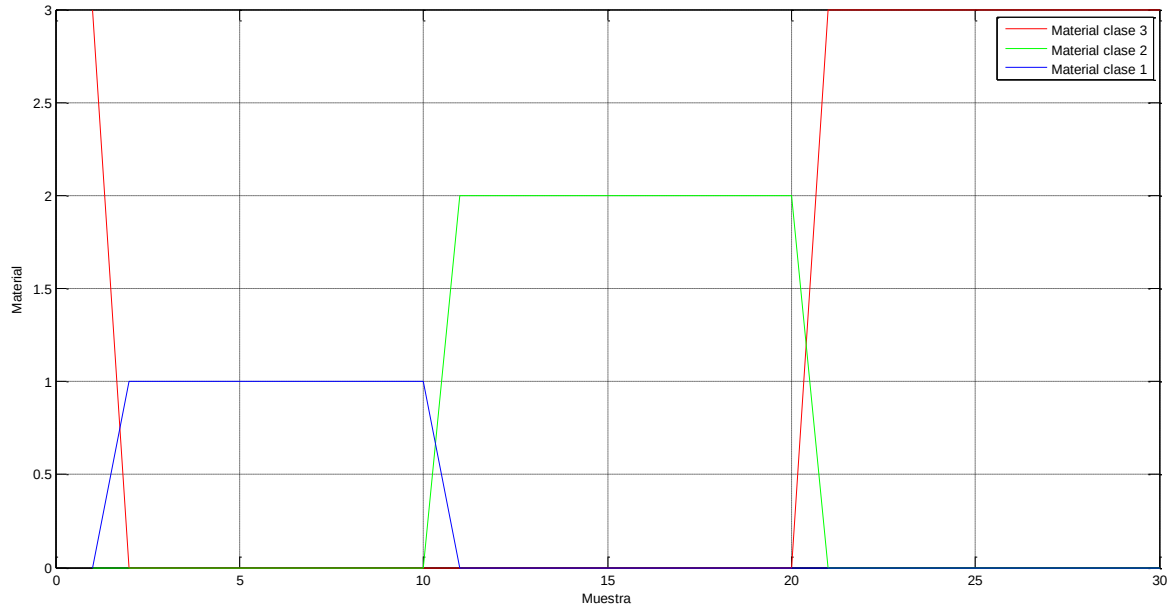
Además, con esta solución, y con el objetivo de mejorar la confiabilidad de la respuesta del programa, se han cambiado los materiales que se utilizaban para realizar las pruebas, tanto para la red neuronal, como para las pruebas en tiempo real. Los nuevos materiales, no tienen un núcleo sólido, por lo que al ejercer presión sobre ellos, sufren una deformación mayor. Con este incremento en la deformación sufrida por el material, se piensa los resultados pueden mejorar notablemente debido al incremento de la cantidad de tactels que son utilizados en cada lectura. En la siguiente imagen, se puede ver la diferencia de la deformación ofrecida por ambos materiales en la garra.





Siguiendo la metodología que resulta del estudio anterior, nos ponemos a generar la red neuronal, obteniendo los resultados que vemos a continuación:





Como se puede ver en las imágenes, los resultados son muy buenos. En la primera imagen, vemos que la tasa de aciertos es del 97.1%, mientras que la prueba con las muestras

de prueba, indican que de 30 muestras que se han utilizado para estos propósitos, 29 han resultado clasificadas correctamente.

Para la realización de esta red neuronal, se han utilizado las siguientes características:

- Muestras de cada material para entrenamiento y validación: 70 de cada material
- Muestras para prueba: 10 de cada material
- Número de neuronas en la capa oculta: 11
- Número de columnas de la matriz del sensor utilizadas: 5
- Preprocesado de las entradas de la red neuronal: ninguno

Con estos resultados, nos atrevemos a realizar el mismo procedimiento para los materiales que nos habíamos propuesto desde el principio, los cilindros que tienen un núcleo rígido y que ofrecen, por ello, una deformación menor en el sensor.

En el diseño de esta red neuronal, tan sólo se ha variado el número de muestra utilizada para entrenamiento y validación, y para prueba. En este caso, se han utilizado 80 muestras de cada material para entrenamiento y validación, y 20 de cada uno de ellos para prueba.

RESULTADOS FINALES

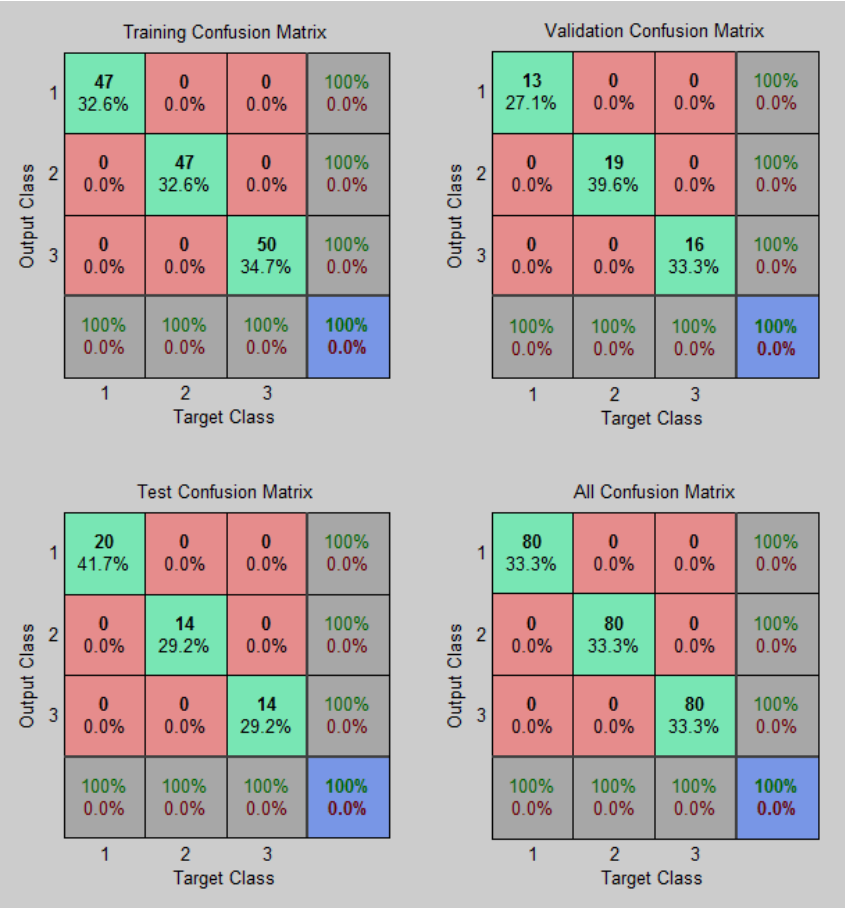
En definitiva, han sido muchas las opciones que se han utilizado para intentar llegar a una solución correcta del problema. Para ello, se ha realizado una búsqueda de algoritmos que puedan ayudar a resolver el problema con una fiabilidad lo más alta posible. En el camino, se han encontrado problemas de diverso tipo.

Algunas opciones que se han probado y han sido descartadas por no obtener un resultado correcto, podrían ser utilizadas a partir de este punto en el que se queda el proyecto para poder mejorar la respuesta. Por ejemplo, la aplicación del algoritmo de Marzullo resultó no dar solución a los problemas encontrados, sin embargo, ahora que la solución es buena, puede ayudar a mejorar la respuesta del clasificador.

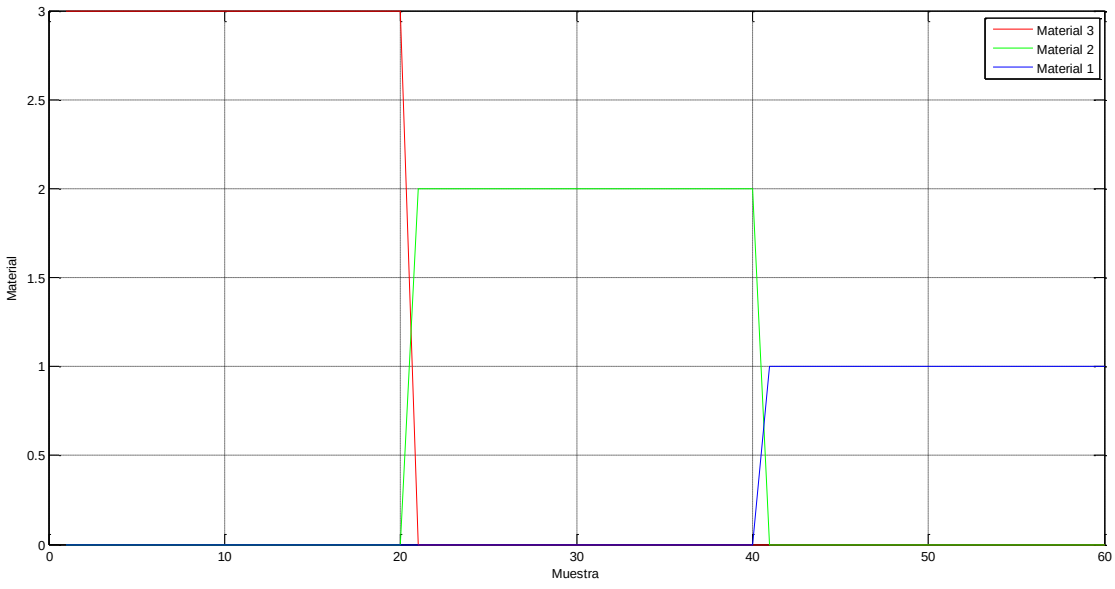
En el camino hacia la solución del problema, se ha llegado a pensar que el sensor utilizado no era capaz de obtener los resultados buscados debido a la resolución que ofrece su matriz. Esto es debido, a que cada celda de la matriz tiene un tamaño de 3.4 milímetros. Dada la deformación ofrecida por los materiales (cilindros rígidos), la respuesta ofrecida por el sensor tan sólo tiene unas 5 filas con información, lo que supone además una gran pérdida de información. Afortunadamente, esta idea fue descartada una vez se comprobó que introduciendo parte de la información proporcionada por el sensor en la red neuronal, se obtienen mejores resultados que haciendo cualquier modificación sobre estos valores con los materiales que ofrecen una deformación considerable, por lo que se optó por probar los materiales más rígidos y por lo tanto, con menos deformación, con resultados satisfactorios.

También se pensó que los resultados esperados no se iban a conseguir con el uso de una garra neumática sin control de fuerza. Sin embargo, esto no ha sido así, y todas las pruebas realizadas con esta garra, tras llegar a la solución algorítmicamente correcta, han ofrecido buenos resultados. No obstante, con un control de fuerza en la garra los resultados pueden llegar a ser mejores.

Tras todas estas pruebas y tiempo invertido, se ha llegado a una solución que ofrece unos resultados buenos, llegando a dar, en teoría, una tasa de aciertos del 100%. El resultado que nos muestra MATLAB a la hora de generar la red neuronal para la clasificación de cilindros de núcleo rígido es el que sigue:



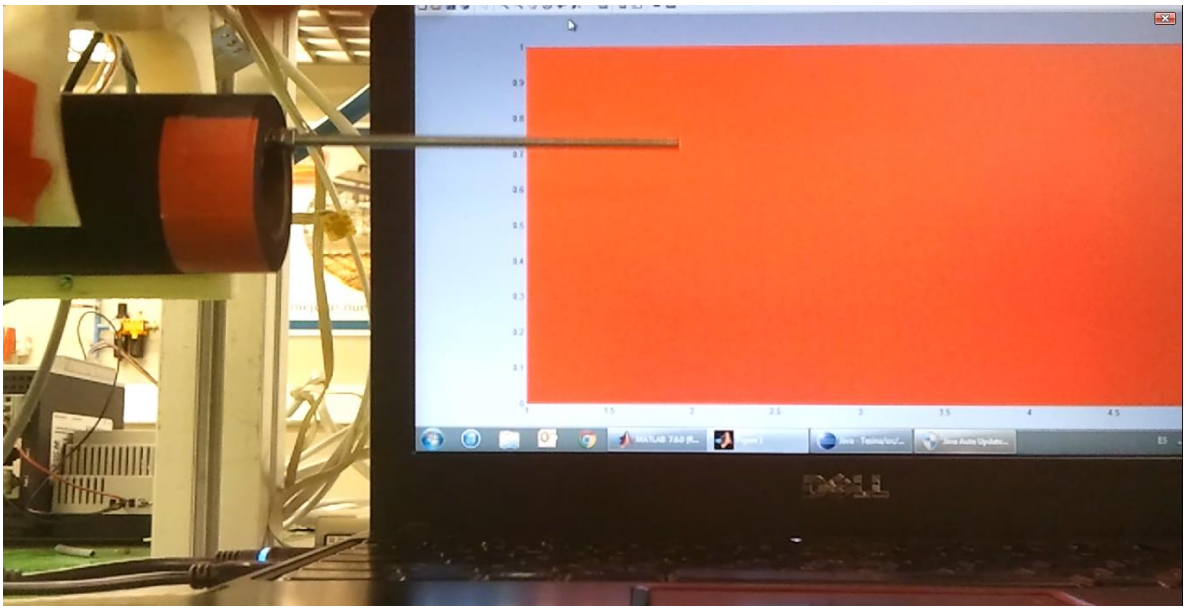
Mientras que si realizamos una prueba en la que se muestre el resultado generado por esta red neuronal con las muestras seleccionadas para prueba, nos encontramos con el resultado que se muestra en la siguiente imagen:



Con estos resultados, realizamos las modificaciones pertinentes al programa de Java que se encarga de realizar la captura de datos. Ahora, el programa sigue el siguiente procedimiento cada vez que quiere realizar un ciclo de toma de muestra:

a. Manda un mensaje al sensor para que éste active el envío de los valores de los sensores periódicamente cada 200 milisegundos.
b. Comienza a guardar todos los valores recibidos por el sensor en un fichero.
c. Cierra la garra.
d. Realiza una espera de 1 segundo.
e. Abre la garra.
f. Manda un mensaje al sensor para que éste detenga el envío de los valores periódicamente.
g. Realiza una llamada a MATLAB para que éste realice la comprobación con la red neuronal, e informe visualmente del material del que se trata.

En la siguiente imagen, se puede ver una fotografía tomada mientras el sistema se encuentra realizando pruebas en tiempo real, en el momento en el que ha dado como resultado que el material utilizado es el representado mediante el color rojo.



Sin embargo, cuando hacemos las pruebas en tiempo real, vemos que la tasa de aciertos que nos muestra MATLAB cuando realizamos la red neuronal no es del todo cierta. En la realidad, la clasificación no acierta en todas las muestras, ofreciendo algún que otro fallo en el reconocimiento. No obstante, esta solución se considera una buena base desde la que continuar con el desarrollo.

CONCLUSIONES Y TRABAJOS FUTUROS

Dados los resultados tan favorables que hemos obtenido, en primer lugar se puede afirmar que es totalmente factible la utilización de sensores táctiles para conocer la firmeza de piezas. Dado todo lo que se está estudiando sobre estos sensores, y las poca investigación hasta el momento sobre el uso de estos sensores, se puede considerar dicho resultado como un muy buen comienzo para la aplicación de estos sensores en la industria de la alimentación.

Quizás, una de las industrias con más necesidad de la utilización de estos sensores sea la industria agroalimentaria, ya que, a pesar de la gran automatización que han sufrido la mayor parte de los procesos de fabricación en casi cualquier ámbito, esta industria está todavía sin explotar. Para ejemplarizar un suceso que corrobore lo anterior, basta decir que actualmente, en las industrias dedicadas al envasado y manipulación de frutas y hortalizas como pueden ser tomates o pepinos, existen contratos laborales cuyo único objetivo es utilizar el sentido táctil para tocar cada producto y tomar una decisión acerca sobre si el producto es apto para consumo alimenticio o no. Esto no solo supone un coste alto para la dirección de la empresa, sino también para los empleados que a ello se dediquen, ya que son muchas horas las que se pasan de pie sin moverse, lo que supone dolores innecesarios en la zona lumbar, además del daño psicológico que supone tal desmotivación.

Sin embargo, como se ha mencionado repetidas veces, aún queda mucho por investigar en este campo, ya que desde distintos recursos, se ha visto que este tipo de sensor, por muy buenos resultados que pueda obtener, sufre de falta de robustez y vida útil muy corta. En algunos casos, esta vida útil puede suponer no poder rentabilizar el coste del sensor.

De los resultados obtenidos en el desarrollo del proyecto detallado, podemos concluir, como punto más importante, que para la clasificación de este tipo de materiales no es necesaria la utilización de un sensor que ofrezca una matriz tan grande.

Los resultados finales, se han obtenido utilizando tan sólo 5 columnas de las 25 ofrecidas por el sensor, por lo que se podría utilizar un sensor más pequeño.

Buscando en la web del fabricante (www.weiss-robotics.de), podemos ver la gama de sensores que ofrecen con las mismas características. De ahí, vemos que existe un sensor más pequeño, con una matriz de 4x6. No sabemos si sería viable, dadas las reducidas

dimensiones del mismo, pero seguramente, probando, y realizando buenas mediciones, se podría llegar a una buena clasificación de los materiales. El uso de un sensor más pequeño, tiene ventajas, entre las que se encuentran las siguientes:

- Ahorro en la inversión necesaria para adquirir este tipo de sensores.
- Reducción del tamaño de la garra necesaria para mantener el sensor.
- Aumento en la velocidad de comunicación entre el computador y el sensor.

Además, con el trabajo desarrollado se abren las puertas a muchas otras investigaciones relacionadas con este tipo de sensores y con el mundo de la alimentación.

Para continuar con este trabajo, lo primero que se debe hacer son pruebas para comprobar la eficacia al clasificar materiales que tengan densidades más parecidas a las utilizadas. Además, el cuello de botella de la implementación actual se encuentra en el tiempo necesario para realizar varias lecturas del mismo material con valores que sigan unos resultados coherentes. Para ello, quizás la solución más óptima sería realizar un modelado de la respuesta, con lo que sabríamos en todo momento, cuánto hay que corregir la respuesta de cada valor teniendo en cuenta el tiempo que ha pasado desde la última lectura.

Una vez esté dominada la clasificación de los distintos cilindros, sería buena idea cambiar de materiales que ofrezcan algo de incertidumbre, que sean distintos unos de otros. La idea final del proyecto es llegar a una solución que permita eliminar la necesidad de utilizar empleados que tengan como único objetivo coger hortalizas y frutas para comprobar cuáles están en buenas condiciones de maduración y cuáles no, por lo que lo ideal sería pasar de estos cilindros a algún tipo concreto de hortaliza que tenga forma similar, como podría ser un calabacín. Este tipo de material, tiene la dificultad añadida de la variación de las medidas, además de la variación de la forma, lo que puede llevar a cambiar totalmente la filosofía del problema.

Para llevar a cabo un proyecto como el mencionado, también habría que plantearse un sistema adicional al sensor para complementar la respuesta. Una solución típica que sería viable es utilizar visión artificial. Dado la implantación de un sistema de estas características se llevaría a cabo utilizando un robot industrial como efector del sensor, y una cinta transportadora como sistema de desplazamiento de, en este caso, hortalizas, el sistema de visión artificial ayudaría al robot a determinar la posición del siguiente elemento a clasificar, incluso, con la ayuda de alguna cámara de visión especial, determinar si el elemento se encuentra en malas condiciones para ser descartad automáticamente.

Además de este sistema como complemento al sensor táctil, también se pueden añadir más sensores a la garra que mantiene el sensor. La tarea de estos sensores en este caso, sería la de complementar y dar confianza a la señal proporcionada por el sensor. Tales sensores podrían ser del tipo acelerómetros, ya que existen aproximaciones similares a las utilizadas en este proyecto utilizando este tipo de sensores.

REFERENCIAS

[1] Zulhadi Zakaria, Nor Ashidi Mat Isa y Shahrel A. Suandi. A study on neural network training algorithm for multiface detection in static images.

[2] L L Bologna, J Pinoteau, J-B Passot, J A Garrido, J Vogel, E Ros Vidal y A Arleo. A closed- loop neurobotic system for fine touch sensing. 24 Julio 2013

[3] F Lotte, M Congedo, A Lécuyer, F Lamarche y Arnaldi. A review of classification algorithms for EEG-based Brain-Computer interfaces. Journal of Neural Engineering 4 (2007).

[4] Antonio Morales, Mario Prats y Javier Felip. Sensors and methods for the evaluation of grasping. Robotic Intelligence Lab. University Jaume I of Castellón.

[5] ioLogik E1200 Series User´s Manual.

[6] Greg Welch y Gary Bishop. An introduction to the kalman filter. Department of Conputer Science. University of North Carolina at Chapel Hill.

[7] Mohamed LAARAIEDH. Implementation of Kalman filter with Python languge. IETR Labs, University of Rennes l.

[8] Xabier Basogain Olabe. Redes neuronales y sus aplicaciones. Dpto. Ingeniería de Sistemas y Automática. Escuela superior de ingeniería de Bilbao.

[9] Pete S. Maybeck. Stochastic models, estimation and control, Volume 1. Chapter 1. Department of electrical engineering.

[10] Nils J. Nilsson. Introducion to machine learning. An early draft of a proposed textbook. Robotics Laboratory. Department of Computer Science. Stanford University.

[11] Alfonso Moreno Rodríguez. Desarrollo de una interfaz gráfica de redes neuronales usando Matlab. Universidad Carlos III de Madrid.

[12] Student Dave´s Tutorials. Kalman filter with Matlab code. Studentdavestutorials.weebly.com.

[13] Wolfgang Härdle y Heiko Lehmann. Neural Networks. 28 Julio 2004. sfb649.wiwi.hu-berlin.de.

[14] WebMining Consultores. Entrenamiento, validación y prueba. 06 Julio 2011. www.webmining.cl.

[15] Wilfried Elmenreich y Robert Leidenfrost. Fusion of Heterogeneous Sensors Data. Sixth International Workshop on Intelligent Solutions in Embedded Systems, pages 191-200, Regensburg, Germany, 2008.

[16] www.mathworks.es

[17] Keith Marzullo. Tolerating Failures of Continuous-Valued sensors. ACM transactions on Computer Systems, Vol. 0, No. 4, November 1990, Pages 284-304.

[18] Carla Abdo Brohem, Laura Beatriz da Silva Cardeal, Manoela Tiago, María S. Soengas, Silvia Berlanga de Moraes Barros y Silvya Stuchi Maria-Engler. Artificial skin in perspective: Concepts and Applications. February 2011.