INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

"Metodología del diseño de sensores virtuales basados en datos"

TESIS

Que para obtener el grado de

Maestría en Ciencias en Ingeniería de Cómputo con Opción en Sistemas Digitales

Presenta:

Ing. Eduardo Meneses Bello

Asesores:

Dr. Víctor Hugo Ponce Ponce

Dr. Herón Molina Lozano

Septiembre 2015

Resumen

"METODOLOGÍA DEL DISEÑO DE SENSORES VIRTUALES BASADOS EN DATOS"

Palabras Clave: Sensores Virtuales, Medición e instrumentación, Inteligencia artificial, Instrumentación Virtual, Control Automático.

Resumen:

En este trabajo se presenta un enfoque sistemático sobre la metodología de diseño para sensores virtuales basados en datos, los cuales son usados como una alternativa flexible y económica para la medición de variables en procesos donde la implementación de un sensor físico es difícil o imposible. Se presentan dos técnicas muy usadas para el desarrollo de sensores virtuales basados en datos. La primera de ellas son las redes neuronales artificiales (RNA), método de aprendizaje supervisado muy usado para el desarrollo de sensores virtuales por su capacidad de "aprender" a partir de ejemplos dados. Después se presenta la técnica del agrupamiento difuso (FC), algoritmo de agrupamiento no supervisado que se ha usado recientemente en el desarrollo de sensores virtuales por su capacidad de asociar datos sin dar un patrón de relación entre ellos a la entrada. Finalmente, como caso de estudio se lleva a cabo el desarrollo de un sensor virtual capaz de detectar objetos en las pinzas de un brazo robótico y clasificarlos de acuerdo a su forma y tamaño.

Abstract:

This work presents a systematic approach to the design methodology for data-driven soft sensors, which are used as a flexible and economic alternative for measuring variables in processes where the implementation of a physical sensor is difficult or impossible. Two commun methods for the development of virtual sensors are presented. The first method presented are the artificial neural networks (ANN), supervised learning method widely used for the development of virtual sensors for their ability to " learn" from given examples. Then the technique of fuzzy clustering (FC) is presented, unsupervised clustering algorithm which has been recently used in the development of virtual sensors for their ability to associate data without giving a pattern of relationship between them to the input. Finally as case study the development of a virtual sensor capable of detecting objects in the gripper of a robotic arm and classify them according to their shape and size is presented.

Agradecimientos

A **Dios** por estar conmigo en cada momento y por poner en mi camino a las personas indicadas en el momento apropiado.

Al **IPN** por permitirme formarme como persona y profesionista; así como al **CIC** por permitirme cursar mis estudios de maestría.

Al **CONACYT** por el apoyo económico otorgado durante la realización de mi maestría.

Al **Dr. Víctor Hugo Ponce Ponce** por el tiempo dedicado para el desarrollo y conclusión de esta tesis.

Al **Dr. Herón Molina Lozano** por sus aportaciones en la revisión de este trabajo.

A la **Dra. Elsa Rubio Espino** por su apoyo e interés en el desarrollo de este trabajo. Por su consideración y por sus útiles consejos.

Al **Dr. Chistian Becker-Asano** por darme la oportunidad de formar parte de un proyecto tan importante en mi vida. Por las enseñanzas y por el tiempo dedicado en la realización de este trabajo.

Así mismo agradezco a mis profesores, compañeros y a toda aquella persona que de alguna manera contribuyó al término de esta tesis.

Y, por supuesto, el agradecimiento más profundo y sentido va para mi familia, que siempre me ha brindado su apoyo, y es mi inspiración para seguir adelante.

Notación

SV	Sensor Virtual
VI	Instrumento Virtual
ACP	Análisis de Componentes Principales
RCP	Regresión por Componentes Principales
MCP	Mínimos Cuadrados Parciales
RMCP	Regresión por Mínimos Cuadrados Parciales
SVM	Maquina de Soporte Vectorial
RNA	Red Neuronal Artificial
RNR	Red Neuronal Recurrente
PMC	Perceptrón Multicapa
LMS	Mínimos Cuadrados Promedio
ECM	Error Cuadrático Medio
FC	Agrupamiento Difuso
FCM	Fuzzy C-Means

Índice

Índice de figuras

Índice de tablas

Objetivos

Objetivo general

Presentar una metodología de diseño de sensores virtuales basados en datos, para la estimación de variables dentro de un sistema dinámico utilizando técnicas de aprendizaje automático.

Objetivos Particulares

1. Describir los principios básicos de funcionamiento, así como características y aplicaciones de los sensores virtuales.
2. Describir una metodología de diseño para sensores virtuales basados en datos.
3. Diseñar y construir un sistema físico donde se implementará un sensor virtual (brazo robótico).
4. Diseñar e implementar un sensor virtual capaz de detectar y clasificar piezas de ajedrez en las pinzas de un brazo robótico.
5. Validar el sensor virtual, clasificando piezas de ajedrez con las pinzas de un brazo robótico.

Parte I

Panorama General

Capítulo 1

Introducción

El desarrollo de la instrumentación moderna ha venido creciendo a partir de los diferentes tipos de sensores que aparecen día a día en el mercado, la mayoría de estos reproducen con mayor calidad las señales censadas y describen en rangos lineales más amplios las variaciones de las señales a censar, por otra parte esto requiere de un mejor soporte del sistema de interfaz con la conexión de medición en tiempo real. Estos sistemas presentan dos problemas, uno la posibilidad de que la medición de alguna variable no pueda realizarse en forma física y otro que el costo se incrementa ya que se considera tanto el sistema de medición como de los sensores.

Una nueva alternativa para el censo de variables, se presenta con el desarrollo de los llamados sensores suaves o virtuales, estos son sensores que determinan la variación de las variables a partir de un algoritmo de computación y mediante el uso de una base de datos tomada de un proceso en particular. Estos sensores consideran que la evolución dinámica de todas las variables guarda una estrecha relación con respecto a cada una de las variables y con la evolución total del proceso, así que al conocer alguna de ellas se puede estimar el valor de cualquier otra. Por otra parte, el proceso de identificación y estimación permiten reducir ruido y algunos efectos no deseables en las mediciones del proceso.

Este capítulo es una introducción general a la instrumentación virtual y a los sensores virtuales. Se da una breve descripción de los tipos de sensores virtuales y las técnicas que hoy en día se utilizan para su desarrollo, así como también se presenta el desarrollo a lo largo de la historia de la instrumentación virtual y su impacto en la tecnología de medición.

1.1. Introducción a la instrumentación virtual

Un dispositivo virtual es un objeto que posee la capacidad de producir un efecto sin estar presente físicamente. Es por esto, que la denominada instrumentación virtual ha revolucionado el mercado de la instrumentación, porque no es necesario disponer físicamente de los instrumentos para realizar aplicaciones.

La instrumentación virtual es un campo interdisciplinario que combina tecnologías de censado, de hardware y software con el fin de crear instrumentos flexibles y sofisticados para aplicaciones de control y monitoreo. Hay varias definiciones de un instrumento virtual disponible en la literatura. Santori define un instrumento virtual como "un instrumento cuya función y capacidades en general se determinan en el software"[56]. Goldberg describe que "un instrumento virtual se compone de algunas subunidades especializadas, computadoras de uso general, software, y un poco de conocimiento"[21]. Aunque informal, la siguiente definición captura la idea básica de la instrumentación virtual y de los conceptos virtuales en general "cualquier computadora puede simular cualquier otra computadora si simplemente le cargamos el software que simule la otra computadora"[14]. Esta universalidad introduce una de las propiedades básicas de un instrumento virtual, su capacidad de ser modificado a través de software, permitiendo a un usuario modificar su función en cualquier momento para adaptarse a una amplia gama de aplicaciones. El concepto de instrumentación virtual nació a finales de 1970, cuando la tecnología del microprocesador permitió cambiar la función de una maquina simplemente cambiando su software [56]. La flexibilidad es posible, ya que las capacidades de un instrumento virtual dependen muy poco del hardware.

Desde principios de los años 80, y siguiendo la tendencia de la tecnología y del mercado, varias compañías comenzaron a desarrollar sistemas para implementar aplicaciones basadas en instrumentación virtual, esto debido principalmente a la eficiencia y beneficios de esta nueva tecnologías, permitiendo de esta forma que el usuario configure y genere sus propios sistemas logrando de esta forma: alto desempeño del sistema, flexibilidad, reutilización y reconfiguración. A la par con estos beneficios se logra una notoria disminución de costos de desarrollo, costos de mantenimiento, etc.

1.2. Introducción a los sensores virtuales

Las plantas industriales suelen estar equipadas con un gran número de sensores. El propósito principal de los sensores es proporcionar datos para el proceso de monitoreo y control. Sin embargo, hace aproximadamente dos décadas, los científicos comenzaron a hacer uso de la gran cantidad de datos que están siendo medidos y almacenados en los procesos industriales, mediante la construcción de modelos predictivos basados en estos datos. En el contexto de la industria de procesos, estos modelos predictivos son llamados soft sensors. Este término en ingles es una combinación de palabras, "soft"debido a que los modelos son programas de computadora, y "sensor", porque los modelos ofrecen información similar a sus contrapartes de hardware. Otros términos comunes para este tipo de sensores son sensores de inferencia [15, 51] y sensores virtuales, como son llamados en [25]. A partir de este momento, se utilizará solamente el término sensor virtual (SV) para referirse a este tipo de sensores en el presente trabajo.

En particular, en la industria de procesos, los sensores virtuales se han establecido como herramientas útiles, que pueden brindar información importante acerca de los procesos. Principalmente, se pueden distinguir dos tipos de sensores virtuales, conocidos como, sensores virtuales basados en modelos (model-driven) y sensores virtuales basados en datos (data-driven) [41]. Hay otros términos que pueden ser utilizados para estos dos tipos de modelos, como por ejemplo, modelos paramétricos vs. modelos no paramétricos, de caja blanca vs. de caja negra [17], y modelos fenomenológicos vs modelos empíricos [64]. Los sensores virtuales basados en modelo utilizan los principios físicos y químicos propios del proceso. En general, si se conoce el comportamiento del proceso y tales datos están disponibles, se utilizan sensores virtuales basados en modelo. El problema es que, en los procesos industriales, a menudo esto no sucede, porque los procesos son demasiado complejos para ser descritos por modelos rigurosos en forma de ecuaciones matemáticas o químicas. Un ejemplo destacado de estos tipos de procesos son los procesos bioquímicos, como la producción de penicilina [35]. Por otra parte, los sensores virtuales basados en datos utilizan los datos históricos del proceso para construir un algoritmo que permite relacionar las variables disponibles con la variable que se desea estimar.

1.3. Antecedentes

La historia de la instrumentación virtual se caracteriza por un aumento continuo de la flexibilidad y la escalabilidad de los equipos de medición. Desde los primeros instrumentos eléctricos controlados manualmente, el campo de la instrumentación ha ido evolucionando, hasta llegar a tener los equipos sofisticados controlados por computadora que se tienen hoy en día. Se puede decir que la Instrumentación ha tenido las siguientes fases:

1. Dispositivos de medición analógicos,
2. Dispositivos de adquisición de datos y de procesamiento,
3. Procesamiento digital en computadoras de propósito general, e
4. Instrumentación virtual distribuida.

La primera fase está representada por los primeros dispositivos de medición analógicos, tales como osciloscopios o sistemas de electroencefalografía (EEG). Estos sistemas se especializaban específicamente en una tarea, e incluían muchos otros dispositivos para su funcionamiento tales como fuentes de energía, sensores, traductores y displays [20]. Este tipo de dispositivos requerían de ajustes manuales, y presentaban sus resultados en varios contadores, indicadores, monitores o en papel. Por lo tanto los datos no eran almacenados en su totalidad, o de lo contrario un operador tenía que copiar físicamente los datos en una hoja de datos. Realizar procedimientos de pruebas complejas o automatizadas era bastante complicado o imposible, ya que todo tenía que ser ajustado manualmente.

La segunda fase se inició en la década de 1950, como resultado de las demandas en el campo del control industrial. Los instrumentos incorporaban sistemas de control rudimentarios, como relevadores, detectores de velocidad e integradores. En esta etapa se creó el controlador proporcional integral derivativo (PID), lo que permitió una mayor flexibilidad de los procedimientos de prueba y automatización de algunas fases del proceso de medición [21]. Los Instrumentos comenzaron a digitalizar las señales de medición, permitiendo el procesamiento digital de los datos, e introduciendo un control más complejo o decisiones analíticas. Sin embargo, los requerimientos del procesamiento en tiempo real todavía eran demasiados altos

para este tipo de instrumentos. Los instrumentos aún seguían siendo cajas para propósitos y clientes específicos.

En la tercera fase, los instrumentos de medición se basaron en la computadora. Comenzaron a incluir interfaces que permitieron la comunicación entre el instrumento y la computadora. Esta relación comenzó con el bus de interfaz de propósito general (GPIB) creado por Hewlett-Packard (HP) en 1960, entonces llamado HPIB, para fines de comunicar instrumentos de control con equipos HP. Inicialmente, las computadoras se utilizaban principalmente como instrumentos fuera de línea. Antes de procesar los datos primero los recolectaban en discos y después se llevaba a cabo su análisis [39].

A medida que la velocidad y las capacidades de las computadoras personales (PC) fueron avanzando exponencialmente, computadoras de uso general se hicieron lo suficientemente rápidas para mediciones complejas en tiempo real. Fue entonces cuando las computadoras comenzaron a utilizarse en aplicaciones en línea que requerían de medición y control en tiempo real. Las computadoras de propósito general de la mayoría de los fabricantes, ya incorporaban todo el hardware y la mayor parte del software requerido por los instrumentos de ese entonces. Las principales ventajas de las computadoras personales estándar eran sus precios más bajos, impulsados por el gran mercado, la disponibilidad y la estandarización.

Aunque el rendimiento de las computadoras pronto se convirtió en lo suficientemente alto, todavía no eran fáciles de usar para la mayoría de los usuarios. Casi todos los programas para instrumentos de control fueron escritos en BASIC, porque había sido el lenguaje más utilizado para los controladores de instrumentos dedicados. Lo que requería que ingenieros y otros usuarios, primero se convirtieran en programadores antes de poder operar los instrumentos, así que fue muy difícil explotar el potencial que la instrumentación computarizada podría traer. Por lo tanto, un evento importante en la historia de la instrumentación virtual sucedió en 1986, cuando National Instruments introdujo LabVIEW 1.0 en una plataforma de PC [56]. LabVIEW introdujo las interfaces gráficas de usuario y programación visual dentro de la instrumentación computarizada, uniendo la simplicidad de una operación utilizando una interfaz de usuario con el aumento en las capacidades de las computadoras. Hoy en día, la PC es la plataforma sobre la cual se realizan la mayoría de las mediciones, y las interfaces gráficas han hecho de la instrumentación una tarea más amigable para el usuario.

Como resultado, la instrumentación virtual hizo posible la disminución en el precio de los instrumentos, debido a que el instrumento virtual depende muy poco de hardware dedicado, y el cliente podía utilizar su propia PC, mientras que un fabricante de instrumentos suministraba sólo lo que el usuario no podía obtener en el mercado general.

La cuarta y última fase se hizo posible gracias al desarrollo de las redes locales y globales de computadoras. Con la mayoría de los instrumentos computarizados, los avances en las telecomunicaciones y las tecnologías de red, se hizo posible la distribución física de los componentes de instrumentos virtuales en los sistemas de monitoreo y control de procesos. Y con esto se facilitó la localización de fallos en sistemas de medición, reduciendo costos y productividad en las plantas.

1.4. Sensores virtuales como instrumentos virtuales

Los instrumentos virtuales (VIs) se pueden considerar como una clase más amplia de la que representan los sensores virtuales. De hecho, los sensores virtuales se enfocan en la estimación de variables en procesos mediante el uso de modelos matemáticos o datos obtenidos del sistema, sustituyendo algunos sensores físicos y usando los datos adquiridos a partir de los disponibles algunos otros.

Por su parte, los VIs están basados en software que realiza cualquiera de las acciones típicas implicadas en una medición y/o problema de control, mediante la utilización de la instrumentación disponible, computadoras y software. Esta acción puede incluir, o no, las capacidades de modelado típico de sensores virtuales.

Los VIs son el resultado de la rápida difusión que han tenido las computadoras de bajo costo en los últimos 20 años en cualquier campo de aplicación de ingeniería, junto con el desarrollo de software [56].

Tanto los instrumentos virtuales como los sensores virtuales representan un modelo alternativo a los instrumentos tradicionales y permiten personalizar las capacidades de las instalaciones de medición para la aplicación del usuario y de tomar el control de la forma de utilizar y presentar los resultados.

Capítulo 2

Planteamiento del problema

En este capítulo se presenta en primera instancia un panorama donde se plantean los problemas a los que se enfrenta el control de variables en procesos industriales. Y más adelante se propone y justifica el uso de sensores virtuales como una herramienta viable en la sustitución de sensores físicos en el proceso de monitoreo y control de procesos industriales.

En los últimos años, una de las áreas con mayor demanda tecnológica en el campo industrial, se refiere a la innovación, implementación y aplicación de nuevas metodologías en el control de los procesos industriales. La industria local, nacional e internacional ha venido diversificado sus procesos de producción, en los que está implícita la tecnología necesaria para que éstos sean rentables y sus productos puedan alcanzar la calidad necesaria en un mercado cada vez más competitivo. En los procesos industriales existen gran cantidad de variables físicas, químicas, biológicas etc., estas variables deben ser cuantificadas por elementos sensores con gran precisión, para después ser acondicionadas y realimentadas nuevamente a los procesos para posteriormente modificar esas variables a los valores deseados o requeridos por el producto durante su elaboración. En ocasiones esas variables tienen dependencia temporal unas de otras en la dinámica del proceso, por lo que se puede recurrir a medir otras variables alternativas, y a estimar las de interés por medio de diferentes técnicas de control. Las tareas de conversión de las variables físicas a su equivalente eléctrica como señal acondicionada, así como su manejo y realimentación a los procesos mediante elementos de control, es llevada a cabo por la instrumentación requerida por los algoritmos y estrategias de control clásico o moderno, que son adaptados para cada proceso en particular. Dichas tareas se han vuelto complejas, ya que cada vez surgen más procesos en los que están implícitas no-linealidades inherentes, las cuales dificultan la obtención de un modelo matemático completo del sistema.

Este modelo es necesario para realizar simulaciones en computadora del sistema de control en lazo cerrado, por lo que en caso de no ser posible obtenerlo se puede realizar su linealización en un punto de operación, sin embargo cuando esta representación lineal no es suficiente, debido a la complejidad propia del sistema no es posible aplicar los métodos convencionales de control [37].

En el sector industrial existen procesos, como los hornos o las plantas de tratamiento de residuos, que deben trabajar de manera ininterrumpida y cuyas condiciones de operación son complejas; en estos casos se requieren sistemas avanzados de supervisión, control y detección de fallas para garantizar un funcionamiento de acuerdo a las necesidades de los usuarios. Una tendencia actual es la automatización de procesos, lo cual requiere de sensores que permitan medir las variables clave y que generen señales eléctricas para ser procesadas en medios digitales. En este sentido, existen diversos factores importantes a considerar en el proceso de medición; por ejemplo: a) hay variables que no pueden medirse bajo esta perspectiva por falta de sensores adecuados, b) las condiciones hostiles de operación exigen altos estándares de calidad en los sensores y rigurosos procedimientos de mantenimiento, lo que implica costos elevados, c) algunas variables no pueden ser medidas en línea y necesitan análisis de laboratorio; esto induce retardos que afectan seriamente a los sistemas de supervisión y control [9].

2.1. Inconvenientes existentes

Las industrias cada día se enfrentan con la elección de adecuadas políticas de producción que son el resultado de una serie de compromisos que involucran diferentes restricciones. Los precios y la calidad final de los productos son, por supuesto, dos factores relevantes y competitivos, que pueden determinar el éxito de mercado de una industria. Estrictamente relacionado con aspectos tales como son los temas de energía y consumo de materias primas, sobre todo por el precio cada vez mayor del petróleo. Por otra parte, el cumplimiento de las normas de seguridad (de acuerdo con varios estudios, el manejo inadecuado de situaciones anormales representa una causa importante de pérdida en la industria) y los problemas de contaminación ambiental contribuyen a aumentar la complejidad del escenario descrito.

Desafortunadamente, algunos dispositivos de medición son requeridos para trabajar en

ambientes hostiles que, por un lado, deben satisfacer las estrictas normas de diseño, de otro modo requerirá de constante mantenimiento o ajustes para trabajar adecuadamente. En cualquier caso, la ocurrencia de fallos inesperados no puede evitarse totalmente. Así también, algunas herramientas de medición pueden introducir un retraso significativo en la aplicación, lo que puede provocar que no se cumplan adecuadamente las políticas de calidad. Instalar y mantener una red de medición dedicada al monitoreo de una gran planta no es nada barato y el presupuesto requerido puede afectar significativamente los costos totales del funcionamiento de la planta, estos costos son generalmente sesgados por lo que se reduce el número total de variables monitoreadas y/o la frecuencia de las observaciones, aunque en muchas situaciones industriales la falta de sensores en la línea para monitorear algunas de las variables del proceso puede presentar potenciales problemas de operatividad. Un sinfín de casos pueden ser mencionados donde es imposible instalar un dispositivo de medición en línea debido a las limitaciones en las tecnologías de medición. También en estos casos, las variables que son indicadores clave para conocer el rendimiento del proceso se determinan por análisis de laboratorio fuera de línea [17].

2.2. Justificación

Los sensores virtuales son una importante herramienta en diferentes campos de la industria, incluyendo refinerías, plantas químicas, cementeras, plantas de energía, procesadoras de alimentos, plantas nucleares, monitoreo de contaminantes, solo por dar algunos ejemplos. Son usados para resolver un sin fin de problemas tales como medición de sistemas retroalimentados, predicción en tiempo real para control de plantas, validación de sensores y diagnósticos de fallas.

Los modelos matemáticos diseñados para estimar las variables relevantes en un proceso pueden ayudar a reducir la necesidad de usar dispositivos de medición, mejorar la fiabilidad del sistema y desarrollar estrictas políticas para el control de calidad.

Existen un sinfín de razones de porque los sensores virtuales pueden ser provechosamente utilizados en aplicaciones industriales o en otras áreas de ingeniería, en la actualidad se están convirtiendo en herramientas de rutina con la tendencia de migrar las herramientas de lazo abierto hacia sensores de lazo cerrado inferenciales, y/o esquemas de control adaptable.

Por otra parte, la amplia disponibilidad de analizadores en línea y los sistemas digitales que se utilizan tanto para monitoreo y control dan a los diseñadores y operadores las herramientas necesarias para el diseño e implementación de los sensores virtuales con un mínimo o incluso nulo incremento de los costos iniciales [17].

Capítulo 3

Marco teórico

En este capítulo se describen los fundamentos teóricos necesarios para el desarrollo de este trabajo.

3.1. Conceptos básicos

Desviación estándar (σ): Es una medida de dispersión usada en estadística que dice cuánto tienden a alejarse los valores concretos del promedio en una distribución.

Varianza (σ^2): La varianza de una variable aleatoria es una medida de dispersión definida como la esperanza del cuadrado de la desviación de dicha variable respecto a su media.

Cluster o Grupo: Conjunto de personas, animales o cosas que están juntos o reunidos o que tienen alguna característica común.

Análisis de regresión: Es un proceso estadístico para la estimación de las relaciones entre variables. Ayuda a entender cómo el valor típico de la variable dependiente cambia cuando cualquiera de las variables independientes es variada.

Análisis de Componentes Principales (ACP): Es una técnica estadística de síntesis de la información, o reducción de la dimensión (número de variables). Es decir, ante un banco de datos con muchas variables, el objetivo será reducirlas a un menor número perdiendo la menor cantidad de información posible.

Mínimos Cuadrados (MC): Es una técnica de análisis numérico enmarcada dentro de la optimización matemática, en la que, dados un conjunto de pares ordenados: variable independiente, variable dependiente, y una familia de funciones, se intenta encontrar la función continua, dentro de dicha familia, que mejor se aproxime a los datos (un "mejor ajuste"), de

acuerdo con el criterio de mínimo error cuadrático.

Red Neuronal Artificial (RNA): Las redes neuronales artificiales son modelos bastantes simplificados de las redes de neuronas que forman el cerebro. Y, al igual que este, intenta "aprender" a partir de los datos que se le suministran. Se puede decir que una red neuronal es un modelo computacional, paralelo, compuesto de unidades procesadoras adaptativas con una alta interconexión entre ellas.

Error Cuadrático Medio (ECM): En estadística, el error cuadrático medio es una forma de evaluar la diferencia entre un estimador y el valor real de la cantidad que se quiere calcular. El ECM mide el promedio del cuadrado del "error", siendo el error el valor en la que el estimador difiere de la cantidad a ser estimada.

3.2. Sensor Virtual

Los sensores en general son utilizados para monitorear las variables de un proceso en tiempo real. Si el proceso tiene muchas variables, y todas ellas deben ser monitoreadas, entonces se necesitará el mismo número de sensores para dichas variables. Los sensores son costosos, requieren mantenimiento, ser calibrados y pueden llegar a ser muy delicados, además de que necesitan de un "hardware"para conectarse con las unidades de monitoreo.

El objetivo de los sensores virtuales, también conocidos en el medio de la ingeniería de control como "Observadores" o "Estimadores de estado", es poder prescindir de sensores físicos en un proceso, sin ver afectada la productividad del proceso y reduciendo costos. Los sensores virtuales son programas de computadora que a partir de las mediciones obtenidas por una cierta cantidad de sensores reales, realizan una estimación de las variables del proceso que no cuenta con su respectivo sensor. El programa reproduce el sistema observado dándole la información de cuáles son los estímulos (entradas) que recibe el proceso real. Una segunda fuente de información útil para el sensor virtual son los datos que recibe del monitoreo de los resultados del sistema (salidas).

Un sensor virtual calcula una estimación de una cierta variable de interés, basado en un modelo matemático y con mediciones de otras variables. La estimación puede ser utilizada cuando falta un sensor físico, o como una herramienta para validar una medición de un sensor físico poco fiable. En la mayoría de aplicaciones del mundo real, la estimación del

sensor virtual no será tan precisa como un sensor físico bien calibrado. Pero en ocasiones solo se necesita que la predicción se aproxime al valor real para que sea tomada como una medida correcta. Algunas aplicaciones comunes de los sensores virtuales pueden ser la predicción en línea, monitoreo y detección de fallos en procesos, reemplazo de sensores físicos, respaldo de datos para sensores físicos, etc.

La precisión y confiabilidad del sensor virtual dependerá en gran medida del conocimiento que se tenga sobre el proceso real, ya que en base a esto, se diseña el algoritmo de estimación. No obstante, en ciertos casos una gran precisión de estimación no es requerida, ya que los resultados de un sistema de monitoreo pueden ser alimentados a un sistema de control robusto que sólo necesite de una aproximación a los valores reales del proceso para mantenerlo controlado. De igual forma, los algoritmos de estimación mientras son desarrollados suelen ser evaluados en su desempeño comparando su estimación con respecto a mediciones reales.

Los sensores virtuales han encontrado un amplio campo de acción en la industria de procesos, pero es en la industria química donde han tenido su mayor auge. Principalmente en la operación de reactores químicos en donde comúnmente se necesita el monitoreo de diversas variables químicas durante el proceso. En algunos otros procesos como el caso de reactores bioquímicos el uso de observadores no es sólo cuestión de economía sino también de factibilidad, ya que resulta prácticamente imposible el contar con sensores que puedan identificar en línea las diferentes variables microbiológicas involucradas en el sistema reactivo.

Los sensores virtuales han denotado una revolución en la ingeniería de control. Su estudio durante los próximos años seguirá revolucionando la industria de procesos y se extenderán a otras áreas de ingeniería donde su uso se vuelva cada vez más práctico.

3.2.1. Tipos de Sensores Virtuales

Generalmente existen dos tipos diferentes de sensores virtuales, los basados en modelo (Model-driven) y los basados en datos (Data-driven). Aunque hay algunos sensores virtuales basados en modelo que utilizan el filtro de Kalman extendido [67] u observadores adaptativos [4] (ver por ejemplo [11, 33]), este tipo de sensores virtuales están comúnmente basados en los modelos del primer principio (FPM por sus siglas en inglés) [62]. Los modelos del primer principio describen el fondo físico y químico del proceso. Estos modelos se han desarrollado principalmente para la planificación y el diseño de las plantas de procesamiento, y por lo

general se centran únicamente en la descripción de los estados ideales de los procesos. Esto es sólo uno de sus inconvenientes, lo que hace difícil basar sensores virtuales prácticos en ellos. Los sensores virtuales basados en datos ganaron cada vez más en popularidad, debido a que pueden utilizarse sin la necesidad del conocimiento de un modelo del proceso, a este tipo de métodos también se les denomina de caja negra cuando no es posible acceder al conocimiento del modelo, de otra manera recibirán el nombre de caja gris y caja blanca cuando se tiene conocimiento parcial y total del modelo respectivamente (ver figura 3.1).

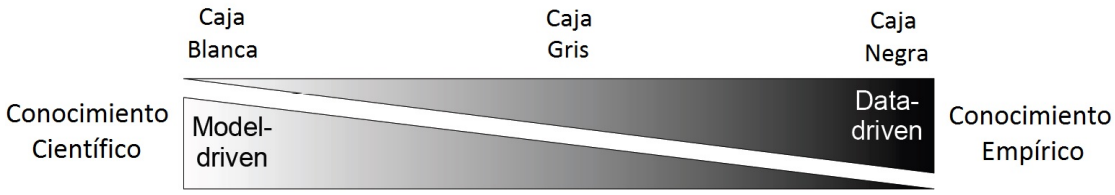


Figura 3.1: Comparación de sensores basados en modelo y en datos

Actualmente han surgido técnicas de modelado para sensores basados en datos, tales como la lógica difusa, métodos estadísticos de regresión múltiple, redes neuronales y algoritmos genéticos, o combinación de ellas. Estas herramientas se basan principalmente en la utilización de conocimiento experto, entrenamiento y aprendizaje programado o estructura de los procesos evolutivos.

La siguiente sección de este trabajo presenta las líneas generales para la metodología de diseño de un sensor virtual basado en datos, con un énfasis especial en dos técnicas de modelado (Redes Neuronales Artificiales (RNA) y Agrupamiento Difuso(FC)). Esta metodología se implementa después como caso de estudio, para el diseño de un sensor virtual táctil en las pinzas de un brazo robótico.

3.3. Metodología de diseño de los sensores virtuales basados en datos

Esta sección describe los pasos típicos para el desarrollo de los sensores virtuales basados en datos, así como los problemas que se presentan en cada uno de ellos, proponiendo posibles soluciones. El procedimiento que se presenta es más bien general y por lo tanto se puede aplicar a cualquiera de las áreas de aplicación que se discuten en la sección 3.4. Una visión general de la metodología se presenta en la figura 3.2.

En las siguientes subsecciones se realiza una descripción de cada una de las etapas de desarrollo de los sensores virtuales basados en datos presentadas en la figura 3.2.

3.3.1. Descripción del proceso

Antes de comenzar a diseñar un sensor virtual, lo primero que se debe conocer, es el funcionamiento del proceso donde se va a implementar, aunque obvio, este paso es imprescindible, debido a que si no se tiene un conocimiento previo del proceso sería imposible implementar un sensor virtual. Aquí, el experto del proceso, como en los pasos subsecuentes, juega un papel importante pues se necesita de su conocimiento acerca del proceso para poder planear la construcción de un sensor virtual.

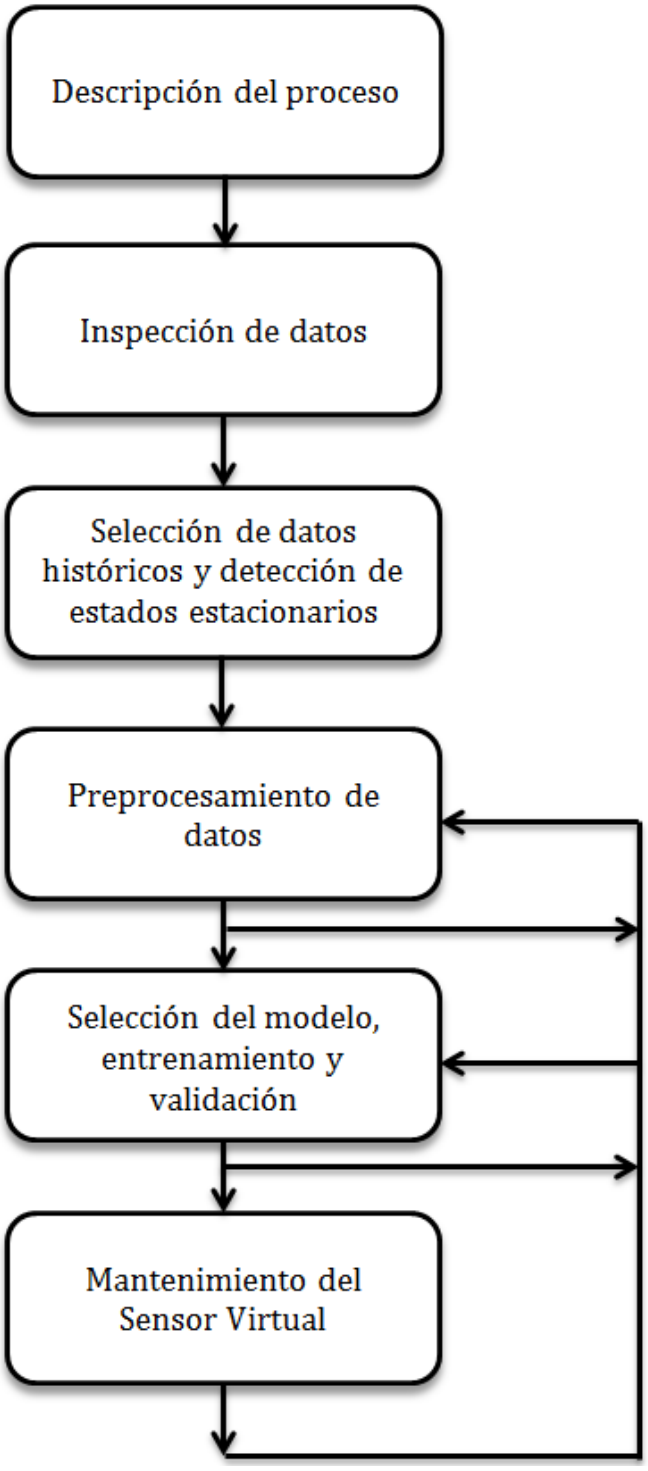


Figura 3.2: Metodología para el desarrollo de Sensores Virtuales basados en datos

Muchos autores dan por hecho este paso, y generalmente no se presenta como un paso de diseño, pero en este trabajo se considera que es un paso fundamental para empezar a concebir lo que será el sensor virtual. En este paso se deben de contestar a preguntas simples, como por ejemplo, ¿en dónde?, ¿cómo? y ¿por qué? se implementará el sensor virtual.

En esta etapa se puede saber si es viable o no la implementación de un sensor virtual. Se deben dar las razones por las cuales se desea implementar el sensor virtual y se hace un planteamiento previo de como se realizará esta implementación, mostrando así los datos que se tienen disponibles para el diseño del sensor virtual.

3.3.2. Inspección de datos

Durante este paso, se lleva a cabo la primera inspección de los datos. El objetivo de este paso es obtener una visión general de la estructura de los datos e identificar cualquier problema obvio que puede ser resuelto en esta etapa (como por ejemplo datos incompletos). El próximo objetivo de esta etapa es evaluar las necesidades de la complejidad del modelo. Un desarrollador de sensores virtuales experimentado puede, ya desde esta fase, tomar una decisión razonable de qué modelo usar, por ejemplo en el caso de una predicción en línea, se podría utilizar un modelo de regresión simple, un modelo de regresión ACP más complejo o una red neuronal no-lineal para construir el sensor virtual. En algunos casos, la decisión del tipo de modelo en esta etapa no es la correcta, por lo tanto, los modelos y su desempeño deben ser siempre evaluados y comparados con modelos alternativos en las etapas de desarrollo posteriores.

Se presta especial atención en la evaluación de la variable de salida. Se comprueba, si hay suficiente variación en la variable de salida y si puede ser modelada.

3.3.3. Selección de datos históricos y detección de estados estacionarios

En este paso, se seleccionan los datos que se utilizarán para el entrenamiento y evaluación del modelo. Después, los valores estacionarios de los datos tienen que ser identificados y seleccionados. En la gran mayoría de los casos, la modelización sólo se ocupará de los estados estacionarios del proceso. La identificación de los estados estacionarios se realiza generalmente por notación manual de los datos.

En [61] la detección del estado estacionario en procesos continuos se discute y se aplica un enfoque basado en la transformada de wavelet para realizar esta tarea.

3.3.4. Preprocesamiento de datos

El objetivo de este paso es transformar los datos de tal manera, que se puedan procesar eficazmente por un modelo. Un ejemplo de un paso típico de pre-procesamiento es la normalización de los datos con media cero y varianza unitaria (como se requiere en el método de ACP). En el caso de los datos que se producen en la industria hay varios pasos necesarios para su pre-procesamiento, indicados en la figura 3.2 por el bucle alrededor del bloque "Pre-procesamiento de datos", por mencionar algunos se tienen; tratamiento de datos faltantes, detección y reemplazo de valores atípicos, tratamiento del desplazamiento de datos, selección de variables relevantes (extracción de características), detección de retrasos entre variables, así como el tratamiento de colinealidad y ruido. La mayoría de los pasos que se indican son tratados de forma manual o necesitan al menos una inspección supervisada de los resultados. El pre-procesamiento de los datos se realiza generalmente de forma iterativa, por ejemplo, después de la normalización y tratamiento de datos faltantes que normalmente se realiza sólo una vez, la eliminación de datos atípicos y la extracción de características se aplican varias veces hasta que el desarrollador considera que los datos están listo para ser utilizados en el entrenamiento y evaluación del modelo. Por el momento, el pre-procesamiento de los datos es un paso que requiere una gran cantidad de trabajo manual y conocimiento experto sobre el proceso. En la figura 3.3 se pueden observar los problemas más comunes que presentan los datos obtenidos de la industria, y serán discutidos a continuación en esta sección.

Datos faltantes

Los datos faltantes son muestras individuales, en los que una o más variables tienen un valor que no refleja el estado real de la cantidad medida física. Las variables afectadas suelen tener valores como $\pm \infty$, 0.

Los datos faltantes en contexto industrial tienen diversas causas. Las más comunes son la falla de un sensor, su mantenimiento o eliminación. Como ya se ha mencionado, las plantas industriales están fuertemente instrumentadas para el propósito de control de procesos, por lo tanto, los datos registrados también consisten en un gran número de variables. En tal

escenario, hay una cierta probabilidad de que algunos de los sensores de vez en cuando produzcan un error. Otras posibles causas de la falta de datos están relacionadas con la transmisión de los datos entre los sensores y las bases de datos, errores en la base de datos, problema para acceder a la base de datos, etc.

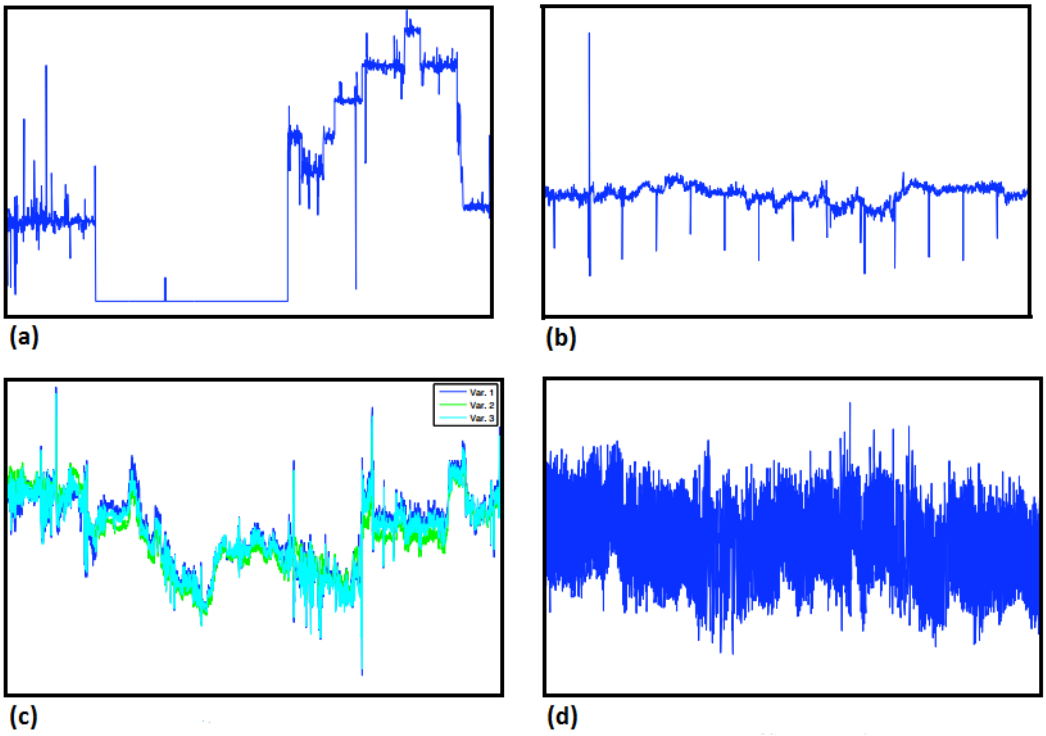


Figura 3.3: Problemas comunes encontrados en datos que se producen en la industria: (a) datos faltantes; (b) valores atípicos; (c) colinealidad; (d) ruido de medición

Dado que la mayoría de las técnicas aplicadas al censado virtual basado en datos no se puede tratar con datos faltantes, se tiene que encontrar una solución al respecto, un enfoque, que es muy primitivo y no se recomienda, pero todavía se aplica comúnmente en escenarios prácticos, es reemplazar los valores perdidos con valores promedio de la variable afectada. Otro enfoque no óptimo es saltarse las muestras de datos que contienen los valores faltantes. Un enfoque más eficiente para el tratamiento de datos faltantes tiene en cuenta la estadística multivariante de los datos y por lo tanto hace que la reconstrucción de los valores faltantes dependan de las otras variables disponibles de la muestra afectada (por ejemplo, un enfoque multivariado de máxima verosimilitud para el reemplazo de datos faltantes [66]). Estos tipos

de enfoques están relacionados a la detección de fallos en sensores y la reconstrucción. De forma general, se pueden distinguir dos enfoques principales para el tratamiento de datos faltantes [57], (1) La sustitución simple, donde los datos faltantes se sustituyen en un solo paso (utilizando, por ejemplo, los valores medios) y (2) la sustitución múltiple, que son técnicas iterativas donde se realizan varios pasos se sustitución.

Valores atípicos

Los valores atípicos son valores de los sensores que se desvían de lo típico, o de los rangos de los valores medios. Se puede distinguir dos tipos de valores atípicos, valores atípicos obvios y no obvios [47]. Los valores atípicos obvios son aquellos valores que violan las limitaciones físicas o tecnológicas. Por ejemplo, la presión absoluta no puede llegar a valores negativos o el sensor de flujo no podrá entregar valores que excedan las limitaciones tecnológicas del sensor. Para ser capaz de detectar este tipo de valores atípicos de manera eficiente, el sistema tiene que estar provisto de los valores límite como información primordial. En contraste con esto, los valores atípicos no obvios son aún más difíciles de identificar, ya que no violan ninguna limitación, pero tampoco reflejan los estados correctos de las variables.

La detección de valores atípicos, como parte del pre-procesamiento de datos sigue siendo muy crítica para el desarrollo de sensores virtuales, porque un valor atípico desapercibido podría tener un efecto negativo en el rendimiento del modelo. Por ejemplo, un solo valor atípico puede ser crítico para un ACP [65, 58]. Otro problema de la detección es que, incluso cuando se aplican medidas automáticas de detección de valores atípicos, por lo general los resultados tienen que ser validados manualmente por el desarrollador del modelo. El objetivo de la inspección manual es detectar cualquier valor atípico enmascarado (valores atípicos etiquetados como valores correctos) e intercambio de valores atípicos (valores correctos etiquetados como valores atípicos) [43].

Los enfoques para la detección de valores atípicos se basan en la estadística de los datos históricos. El enfoque más simple es el algoritmo de detección de valores atípico (véase, por ejemplo, [3, 44]), que se basa en observaciones univariantes de la distribución de la variable. Una versión más robusta de este enfoque es el identificador Hampel [13], que utiliza la desviación media absoluta (DMA) de los valores [43, 44] para el cálculo de los límites.

Colinealidad

Otro problema común al que se afronta el censado virtual está relacionado con la estructura de los datos. Por lo general, los datos medidos en industrias están fuertemente alineados. Esto es resultado de la redundancia en la distribución de sensores, por ejemplo, dos sensores de temperatura vecinos, entregarán generalmente mediciones correlacionadas. A tales casos a menudo se les llaman *data rich but information poor*[12]. Sin embargo, para el censado virtual los requisitos son diferentes, en este caso sólo se requieren variables informativas.

Hay dos maneras de lidiar con el problema de colinealidad. Una forma es mediante la transformación de las múltiples variables de entrada en un nuevo espacio reducido con menos colinealidad como se hace en el caso del ACP y de los MCP. Estas dos técnicas son las más populares para hacer frente a la colinealidad de datos en la industria de procesos. Otra manera de manejar colinealidad es seleccionar un subconjunto de variables de entrada que sean menos colineales. Estos enfoques se resumen de la gran cantidad de métodos de selección de variables que existen en la investigación de aprendizaje automático, una revisión general de estos métodos se presenta en [27].

Ruido de medición

El ruido de medición es otro efecto común que se observa en los datos de la industria. La mayoría de los enfoques de desarrollo de sensores virtuales están tratando de hacer frente al ruido de medición durante la etapa de pre-procesamiento de datos. Esto se logra principalmente mediante la aplicación de un filtro como un paso del preprocesamiento.

El ACP es también una herramienta útil para hacer frente al ruido por medición. Como el método de mínimos cuadrados promedio (LMS), que puede tratar el ruido de medición, siempre y cuando se pueda suponer como ruido Gaussiano, es decir, distribuido al azar con un valor medio de cero [3].

Zamprogna et al. han demostrado la robustez del método de MCP hacia el ruido de medición en [16]. Los autores han demostrado que el error de predicción de un sensor virtual por MCP no varía mucho cuando se incrementa el nivel de ruido. La explicación de este hecho es que el ruido influye principalmente en las variables latentes de orden superior que normalmente se omiten en la aplicación práctica.

Desplazamiento de datos

Dependiendo de la causa, se puede distinguir dos tipos de desplazamientos de datos, los desplazamientos por proceso y los desplazamientos por sensor. Las causas del desplazamiento por proceso son los cambios en el proceso mismo o de algunas condiciones externas del proceso. Las plantas de procesamiento consisten en un gran número de elementos mecánicos que se someten al desgaste constante durante la operación de la planta. Esto puede tener un efecto sobre el proceso en sí mismo, por ejemplo, el flujo entre dos partes del proceso puede disminuir debido al desgaste de las bombas mecánicas. Otra causa del desplazamiento en los valores también puede tener influencias externas como las condiciones cambiantes del medio ambiente (por ejemplo, la influencia del tiempo), la pureza de los materiales, la eliminación del catalizador, etc. Estos factores no sólo tienen una influencia en los datos, sino que también afectan el estado del proceso. Por lo tanto los desplazamientos deben ser detectados, reportados y se deben de tomar acciones adecuadas para eliminar su causa. Esto es diferente en el caso del desplazamiento por sensores, que son causados por los cambios en los dispositivos de medición y no por el propio proceso. El punto crítico de este tipo de desplazamientos, es que mientras que se observan variaciones en los datos medidos, no se reflejan cambios en el proceso. Por lo tanto, en el caso del desplazamiento por sensores, la acción a tomar debe ser la recalibración de los dispositivos de medición o la adaptación del sensor virtual sin llevar a cabo una acción correctiva en el proceso.

En cuanto a los efectos sobre los datos del proceso, se pueden observar cambios en las medias y las varianzas de las variables individuales, así como cambios en la estructura de correlación de los datos [63].

Distinguir entre los dos desplazamientos discutidos es un reto y una vez más, se necesita una gran cantidad de conocimiento experto con el fin de tomar las medidas adecuadas. Otro aspecto difícil de tratar con el desplazamiento de los datos es el hecho de que los cambios pueden progresar muy lentamente y pueden influir entre sí, y por lo tanto tener forma no lineal, lo que los hace difíciles de detectar y compensar.

El enfoque más común para hacer frente a la dinámica en los datos es la aplicación de *moving window techniques*. En este caso, el modelo se actualiza de manera periódica usando un número definido de muestras recientes. Algunos ejemplos de la aplicación de esta técnica

en el contexto de la modelización del sensor virtual se encuentran en [70, 72, 48, 7]. Otros enfoques para la adaptación del sensor virtual se discuten en la subsección 3.3.6.

3.3.5. Selección del modelo, entrenamiento y validación

Esta fase es crítica para el desarrollo del sensor virtual. Debido a que el modelo es el motor del sensor virtual, y la selección del tipo de modelo adecuado es fundamental para el buen funcionamiento del sensor. Hasta el momento, no existe un enfoque teórico unificado para esta tarea y por lo tanto el tipo de modelo y sus parámetros se seleccionan a menudo de una manera específica para cada sensor. La selección del modelo esta a veces sujeta a experiencias pasadas y preferencias personales del desarrollador, lo que puede llegar a ser una desventaja para el sensor virtual. Esto se puede observar en el dominio de aplicaciones de sensores virtuales publicados donde muchos de los autores se centran en gran medida en un tipo de modelo (por ejemplo, MCP), que está en su campo de experiencia.

Sin embargo, a pesar de la falta de un enfoque teórico común para la selección del modelo, hay algunas técnicas que se pueden adoptar para esta tarea. Un posible enfoque es comenzar con un tipo simple de modelo o estructura (por ejemplo, el modelo de regresión lineal) y poco a poco aumentar la complejidad del modelo, siempre y cuando se pueda observar una mejora significante en el modelo [23]. Durante la realización de esta tarea es importante evaluar el desempeño del modelo sobre los datos independientes [60] y [55]. El mismo enfoque se puede aplicar también a la selección de los parámetros de los métodos de pre-procesamiento, como por ejemplo la selección de variables.

Además, para algunos procesos industriales, puede ser difícil obtener suficiente datos históricos para el desarrollo del modelo. En tales casos, es ventajoso recurrir a técnicas estadísticas de estimación de error, como la validación cruzada *K-fold* [34]. Este método hace uso óptimo de los datos disponibles dividiéndolos de una manera tal que todas las muestras se utilizan para la validación del rendimiento del modelo. Otra alternativa en estas circunstancias es aplicar métodos de re-muestreo estadístico como por ejemplo *bagging* [6] y *boosting* [71]. En el caso del primer método, conjuntos de datos de entrenamiento se generan aleatoriamente a partir de los datos disponibles y se entrena un modelo para cada uno de los conjuntos. El modelo final se obtiene promediando las predicciones de los modelos particulares. En contraste con esto, en el caso de *boosting*, utiliza varios modelos diferentes de forma

iterativa asignando pesos a los datos de entrenamiento y luego promedia el resultado utilizando un enfoque de promedio ponderado. Además, en el caso de *boosting*, los errores en cada iteración se utilizan para actualizar los pesos de cada conjunto de entrenamiento de manera que se incremente el peso de las predicciones erróneas y se reduzca el peso de las predicciones correctas.

Después de encontrar la estructura óptima del modelo y entrenar el modelo, el sensor virtual tiene que ser evaluado con datos independientes [55]. Hay varias herramientas para la evaluación del rendimiento del modelo. En el caso de la evaluación numérica de rendimiento, el más popular es el Error Cuadrático Medio (ECM), que mide la distancia media entre el valor obtenido y el valor correcto. Otra forma de evaluar el rendimiento es utilizando la representación visual de las predicciones. Por ejemplo, el *four-plot analysis* es una herramienta muy útil, ya que proporciona información útil acerca de la relación entre las predicciones y los valores correctos, así mismo lo hace el análisis de los residuos de predicción [17]. Una desventaja de los métodos visuales es que requieren una asistencia del desarrollador del modelo y de la decisión final del rendimiento del modelo queda a juicio del desarrollador.

Una discusión más detallada sobre la selección y la validación del modelo se proporciona en [17] donde, además de la discusión de varias técnicas para la selección del modelo y la validación, los autores del libro hacen hincapié en la necesidad de la aplicación de los conocimientos del proceso durante la fase de desarrollo del sensor virtual.

3.3.6. Mantenimiento del Sensor Virtual

Después de desarrollar y validar el sensor virtual, tiene que mantenerse y ajustarse sobre ciertos valores base. El mantenimiento es necesario debido al desplazamiento de los datos y otros cambios de los datos (ver subsección 3.3.4), que hacen que el rendimiento del sensor virtual se valla deteriorando y tendrá que ser compensado mediante la adaptación o re-desarrollo del modelo.

Actualmente la mayoría de los sensores virtuales no prevén mecanismos automatizados para su mantenimiento. Este hecho junto con la evidencia discutida anteriormente del desplazamiento de los datos, da como resultado la necesidad de llevar a cabo un control de calidad y un mantenimiento de los sensores virtuales de forma manual, lo que representa un factor de costo significativo para la aplicación de sensores virtuales. Peor aún, a menudo no existe

una medida objetiva para evaluar el nivel de calidad del sensor virtual, entonces la decisión de, si un modelo funciona bien o no, depende de la percepción subjetiva del operador, basado en la interpretación visual de la desviación entre el valor correcto y su predicción.

Sin embargo, hay varios enfoques de adaptación en la literatura relacionada con los sensores virtuales. La mayoría de estos enfoques se basan en versiones adaptativas del ACP o de los MCP, como *Moving Window PCA* [70] o el *ACP recursivo* [63] (véase la subsección **??** para ACP y la subsección **??** para MCP). Todos estos métodos se basan en la adaptación periódica o continua de la base del componente principal. Sensores virtuales neurodifusos, como en [30], a menudo proporcionan mecanismos de adaptación automática. Estos mecanismos se basan en el despliegue de nuevas unidades en la estructura neuronal del modelo una vez que se encuentra un nuevo estado de los datos. Un enfoque relacionado con los métodos neurodifusos también proporcionan posibilidades de adaptación y aprendizaje local [10]. Un sensor virtual adaptativo desarrollado en este marco se publicó en [33].

A pesar de los métodos para la adaptación automática del sensor virtual, el operador del modelo sigue desempeñando un papel importante, ya que mediante su juicio y su conocimiento del proceso decide acerca de la forma en que se seleccionan los parámetros de los métodos de adaptación (por ejemplo, la longitud de la ventana en el caso de la técnica *Moving Window PCA*, o un umbral para el despliegue de un nuevo campo receptivo en el caso de los métodos difusos).

3.3.7. Metodologías relacionada

La metodología discutida en esta sección, aunque es la más utilizada, no es la única vía posible para el desarrollo de un sensor virtual. Por ejemplo, Warne et al. [32] presentan una metodología alternativa para el desarrollo de sensores virtuales o sensores inferenciales (como los llama Warne). Es un poco menos detallada, pero sigue siendo coherente con la metodología que aquí se presenta. Se centra en tres etapas principales: (i) recopilación y acondicionamiento de datos, (ii) la selección de variables influyentes y (iii) el desarrollo de la correlación. Estos tres pasos corresponden con "Selección de los datos históricos", "Preprocesamiento de datos" y "Selección del modelo, entrenamiento y validación"de la figura 3.2.

Otro trabajo que se debe mencionar es la metodología para desarrollo de sensores virtuales en Fortuna [17]. Sin embargo, no hay ninguna diferencia significativa a la metodología que se

presenta en esta sección.

Han y Lee, en [25] presenta una metodología bastante general para el desarrollo de sensores virtuales tomando en cuenta la metodología de gestión de procesos Six-Sigma (ver [24] para más detalles sobre Six-Sigma).

En [52], además de una metodología de sensores virtuales de tres pasos generales que consiste en (i) la comprensión del proceso, (ii) preprocesamiento de datos y (iii) la determinación del modelo, se discute una metodología más especializada para el desarrollo de modelos basados en el procedimiento de suavizado multivariante.

3.4. Aplicaciones de los sensores virtuales

Las aplicaciones de los sensores virtuales pueden encontrarse principalmente en muchos campos de la industria de procesos, pero su fiabilidad los ha llevado a ser utilizados también en otras áreas como la robótica, pero es en la industria de procesos donde han encontrado su mayor auge. Los ejemplos más típicos son la industria química, la industria del papel y la industria del acero. Los tipos de aplicaciones más comunes se presentan en las siguientes secciones; cabe mencionar que se muestran solo una parte de las tantas aplicaciones que existen para este tipo de sensores.

3.4.1. Predicción en linea

La aplicación más común de sensores virtuales es la predicción de valores que no se pueden medir en línea usando mediciones automatizadas. Esto puede ser por razones tecnológicas (por ejemplo, no hay equipo disponible para la medición requerida), razones económicas (por ejemplo, el equipo necesario es demasiado caro), etc. Esto a menudo se aplica a valores críticos que están relacionados con la calidad del producto final. En tales escenarios, los sensores virtuales pueden proporcionar información útil acerca de los valores de interés y en caso en que la predicción del sensor virtual cumpla con los estándares dados, puede ser incorporado al control automatizado del proceso. Los sensores virtuales han sido ampliamente utilizados en los procesos de fermentación, polimerización y refinería. El denominador común de estos procesos es que su dinámica no puede describirse fácilmente en términos de rigurosos modelos, y que a menudo no hay manera de recopilar la información necesaria en línea.

Desde el punto de vista del aprendizaje automático estos problemas son equivalentes a la regresión supervisada. Los modelos basados en datos como su nombre lo indica, se basan en la obtención de datos históricos del proceso. Estos datos se componen de las mediciones pasadas de la planta que forman el espacio de datos de entrada del sensor virtual. Y como salida se obtienen mediciones que antes requerían de un análisis previo en laboratorio.

3.4.2. Monitoreo y detección de fallos en procesos

Otra área de aplicación de los sensores virtuales es el monitoreo de procesos. Los modelos pueden ser entrenados ya sea para describir/analizar el funcionamiento normal del proceso o para reconocer fallos posibles del proceso. Comúnmente, las técnicas de control de procesos se basan en técnicas estadísticas multivariantes como ACP, o más precisamente en *Hotelling´s* T^2 [26] y *Q-Statistics* [29]. Estas técnicas tienen por un lado la ventaja de considerar todas las características de entrada, por ejemplo, utilizando la estadística multivariante, y por otro lado proporcionar información acerca de un posible fallo en el sistema por medio de la estadística de monitoreo [59]. Otro método popular para el monitoreo de procesos son los Mapas Auto-organizados (ver sección 4.3).

3.4.3. Detección y reconstrucción de fallos en sensores

La gran mayoría de las técnicas de modelado aplicadas dentro de la industria de procesos como sensores virtuales, no son capaces de lidiar con datos de sensores defectuosos por principio de funcionamiento, por lo tanto, hay una necesidad de identificar y reemplazar los sensor defectuosos y los fallos en el proceso antes de la construcción del modelo y la aplicación del mismo.

Los fallos en sensores y en procesos se detectan y controlan mediante el ACP en [49] y [50]. Los fallos se detectan en el espacio residual del ACP. Esto tiene la ventaja de que se puede, por un lado, identificar las fallas en sensores o en procesos, y por otra parte, mediante la proyección del estado de fallo al espacio original también se pueden encontrar qué sensor o conjunto de sensores son los responsables de la falla. Mediante la manipulación del espacio residual del ACP también se puede lograr una reconstrucción de la falla. En el trabajo también se definen las condiciones de detectabilidad, identificabilidad y reconstructibilidad de fallos.

Para la tarea de detección de fallos en procesos se debe describir la "dirección de la falla", lo que requiere del conocimiento del proceso. Para la detección de fallos en sensores no hay necesidad de tal conocimiento. El enfoque propuesto se evalúa en términos de un proceso continuo de calderas industriales.

En [8] el enfoque anterior se amplía a los procesos dinámicos. La extensión a los procesos dinámicos se logra mediante el uso del ACP de tiempo-retardado (TLPCA) en lugar del método tradicional del ACP estático. Aunque hay una necesidad de eliminar las variables correlacionadas a partir del conjunto de datos, el método presentado afirma que es adecuado para procesos altamente dinámicos, lo que se demuestra en un conjunto de datos industriales.

Otro trabajo para la detección de fallos en sensores basado en ACP y para el diagnóstico del sensores virtuales, se publicó en [54]. El Sensor virtual utiliza la *Q-Statistics* para detectar fallos y los sensores responsables de los mismos. El proceso que se describe es un sistema de enfriadores centrífugos. Los mismos autores publicaron otro sensor virtual para detección de fallos ([53]), esta vez monitoreando una unidad de tratamiento de aire (UTA). Con el fin de hacer frente a la no linealidad del proceso el modelo se divide en dos modelos separados. Además, el modelo se amplía con el uso de un sistema experto simple que maneja las señales de los dos sub-modelos del ACP.

3.4.4. Respaldo de datos para sensores físicos

Los sensores virtuales pueden actuar también como sensores de respaldo para los sensores físicos, que tienden a fallar mucho o que requieren de un mantenimiento continuo. En esta situación, para prevenir una perturbación del sistema, pueden desarrollarse sensores virtuales capaces de reemplazar a los sensores físicos durante su indisponibilidad [25].

3.4.5. Estimación de variables en aplicaciones fuera de la industria de procesos

Los sensores virtuales pueden ser utilizados para estimar variables de un sensor físico que no se puede implementar en un sistema, pero este sistema no necesariamente tiene que pertenecer a la industria de procesos como los ejemplos que se han tratado hasta ahora. Este sistema puede ser cualquier sistema físico dinámico donde existan variables independientes,

y donde se pueda establecer una relación entre las variables independientes disponibles y las variables deseadas. Para esto es indispensable contar con información acerca de la relación existente entre las distintas variables. En este contexto los sensores virtuales estiman la dependencia estadística entre magnitudes, para lo que deben emplearse conjuntos de entrenamientos que incluyan lecturas pasadas válidas. Por tanto, el sensor no "aprende"la física del problema, sino el comportamiento del sensor físico que provee los ejemplos de entrenamiento.

Son pocos los ejemplos de sensores virtuales basados en datos que se han desarrollado para estimar variables fuera de la industria de procesos. Pero como se ha dicho anteriormente, cada día este tipo de sensores toman más fuerza en distintas áreas, y ya se están utilizando como una alternativa real para el censo de variables en automóviles y robots por ejemplo. A continuación se describen algunos ejemplos interesantes donde se hace uso de sensores virtuales basados en datos como estimadores de variables fuera de la industria de procesos.

En [38] se describen dos sensores para estimar variables en el motor de un automóvil, orientados a optimizar las prestaciones y disminuir las emisiones de un motor de combustión interna. El primero estima la posición del pico de presión en cada cilindro de un motor de combustión interna, esta presión se mide tradicionalmente mediante sensores piezoeléctricos instalados dentro del cilindro, cuyo inconveniente es que son demasiados caros. El segundo de estos sensores virtuales estima la relación de aire-combustible en cada cilindro del motor, esta magnitud podría medirse mediante un sensor UEGO (Universal Exhaust Gas Oxygen) instalado en el tubo de escape; sin embargo, además de ser caro, la dinámica de este sensor es demasiada lenta. Además, este método no sería efectivo para el control de cada cilindro a no ser que se monte un sensor por cilindro.

Debido a estas dificultades, ambas magnitudes, se estiman a partir de la corriente de ionización en la bujía, con la que están relacionadas. De esta manera se consigue conocer ambas magnitudes a un costo muy bajo y de manera independiente en cada cilindro. El inconveniente es el alto nivel de ruido presente en la ionización, por lo tanto, se utilizan redes neuronales para el modelo del sensor, debido a que tienen un buen comportamiento ante el ruido.

Otro ejemplo se puede encontrar en [2], donde se implementa un sensor virtual de vacío en la ventosa de la pata de un robot. Se presenta el problema que se tiene en la ventosa colocada en la pata de un robot, donde no se sabe si la ventosa ya está completamente sujeta

a una superficie, por lo tanto, se propone el diseño de un sensor virtual que pueda estimar el vacío de la ventosa y de esta forma, determinar indirectamente si la ventosa esta o no sujeta a la superficie. Por lo que se analizan las señales que se pueden adquirir del sistema, tales como: velocidad y corriente del motor, posición del pistón y tiempo, los cuales actúan como parámetros de entrada de una red neuronal implementada al sensor virtual de vacío.

3.4.6. Resumen de las aplicaciones de los sensores virtuales

A pesar de encontrarse todavía en la fase inicial de su desarrollo, los sensores virtuales empiezan a introducirse de manera comercial en ciertas aplicaciones. Esto es debido en parte a la reciente aparición de herramientas que facilitan su diseño y mantenimiento. En la actualidad la mayoría de los sensores virtuales se pueden encontrar en la industria de procesos, pero solo es cuestión de tiempo para que se extiendan hacia otro tipo de áreas.

En la tabla 3.1 se proporciona una lista con las aplicaciones de sensores virtuales y se resumen sus propiedades más importantes.

La lista de aplicaciones de sensores virtuales presentada en esta sección no está completa, ya que la cantidad de aplicaciones de sensores virtuales publicados es demasiado grande para ser cubierta totalmente. En lugar de esto, este trabajo se centra en una parte en las publicaciones recientes y, por otra parte en los enfoques no tradicionales.

Suponiendo que los ejemplos presentados son una muestra representativa de los últimos sensores virtuales, la distribución de los métodos actuales de censado virtual se presenta en la figura 3.4. La figura muestra claramente la tendencia actual en el censado virtual. Los métodos más populares para la construcción de sensores virtuales son las técnicas de estadísticas multivariante, es decir, el ACP y los MCP, que en conjunto abarcan el 38 % de las aplicaciones presentadas en esta revisión. Otra de las técnicas comúnmente aplicadas en los sensores virtuales, son los métodos basados en redes neuronales como PMC, RNR, etc. Sin embargo, algunas de las aplicaciones más recientes se basan en métodos que han ido encontrando su camino dentro de áreas de aplicación mucho más amplias. Estos son, por ejemplo, los métodos neuro-difusos, que tienen la ventaja de proporcionar un mecanismo intrínseco para la adaptación/evolución, así como las SVM que tienen su justificación en la teoría del aprendizaje automático y han demostrado que tienen muy buena capacidad de generalización a través de un número de diferentes áreas de aplicación.

Publicación	Métodos aplicados	Aplic.	Descripción del proceso
Casali et al. (1998)	SRM (ARMAX)	OP	Estimación del tamaño de partículas en una planta de molienda
Park and Han (2000)	ACP/MCP+LWR	OP	Estimación de la temperatura del diésel en la columna de petróleo crudo
Kadlec and Gabrys (2008a)	RLM	OP	Predicción de oxidación térmica de NOx
Devogelaere et al. (2002)	PMC	OP	Estimación de calidad de azúcar
Jos de Assis and Maciel Filho (2000)	PMC, FPM, eKF	OP	Estimación de la biomasa en un proceso de fermentación
Qin (1997)	PMC, RNMCP	OP	Refinería
Meleiro and Finho (2000)	PMC	OP	Soporte al control del proceso de producción de etanol
Park and Han (2000)	PMC+-Sistema Experto	OP	Control de contenido de silicio en la producción de acero
Fortuna et al. (2005)	PMC	OP	Predicción de la concentración de C4 y C5 en un proceso de refinería
Desai et al. (2006)	PMC, RBFN, SVR	OP	Dos procesos bioquímicos simulados
Wang et al. (2006)	RBFN	OP	modelado del procesos de separación de membrana
Kadlec and Gabrys (2008b)	Conjunto de PMCs	OP	Secadora industrial
James et al. (2002)	PMC, RBF, PMC/RBF+FPM	OP	Predicción de la concentración de biomasa
Su et al. (1998)	RNR+FPM	OP	Predicción del grado de curación en el proceso de materiales compuestos de fibras
Chen et al. (2004)	RNR	OP	Predicción de la concentración de biomasa
Chen et al. (2004)	RNR	OP	Predicción de la longitud del flujo de fusión en el proceso de moldeo por inyección
Yang and Chai (1997)	RNR	OP	Tres procesos simulados simples
Fellner et al. (2003)	RNA Generalizada	OP	Predicción de la concentración de diacetilo
Lin et al. (2007)	ACP	OP	Estimación del producto en hornos de cemento, supervisión de NOx
Qin et al. (1997)	ACP	OP, DFS	Monitoreo de emisiones de aire
Zamprogna et al. (2004b)	MCP, ACP	OP	Columna de destilación simulada
Dayal and MacGregor (1997)	MCPPE	OP	Reactor agitado, circuito de flotación
Qin (1998)	MCPR	OP	Predicción del Número de octanos en un proceso de refinería
Feng et al. (2003)	MC-SVM	OP	Tasa de absorción de la gasolina en FCC
Yan et al. (2004)	MC-SVM	OP	Detección del punto de congelación del diésel en FCC
Merikoski et al. (2001)	ANFIS	OP	Estimación de la viscosidad del caucho
Warne et al. (2004)	PCA+ANFIS	OP	Análisis del sustrato de polímero recubierto
Luo and Shao (2006)	NFS+AG	OP	Detección del punto de congelación del diésel en FCC
Arazo-Bravo et al. (2004)	NFS	OP	Bioproceso de producción de penicilina
Wang and Rong (1997)	NFS	OP	Predicción de la pureza de propileno en una columna de destilación
Macias and Zhou (2006)	NFS	OP	Destilación de crudo en el proceso de la refinería
Dong et al. (1995)	ACPN+RNMCP	OP	Predicción de NOx en los gases de escape
Li et al. (2005)	PSO+PMC	OP	Columna de destilación de etileno
Kalos et al. (2003)	RN analítica+SVM+PG	OP	Estimación del nivel de interfaz en una unidad de neutralización
Chen et al. (2000)	FPM+RBFN	OP	Población microbiana en un biorreactor
Rao et al. (1993)	Sensor Virtual Inteligente	OP	Sistema de fabricación de pasta de sulfito
Gonzalez et al. (2003)	SRM, TS, FC, MCP, WBM	OP	Grado de concentrado de cobre en un proceso áspero
Li et al. (2000)	ACPR	MP	Proceso rápido de recocido térmico
Nomikos and MacGregor (1995b)	ACP	MP	Proceso de polimerización
Rotem et al. (2000)	MBACP	DFP	Compresor de etileno
Wang et al. (2005)	FMWACP	MP	Simulador del proceso FCC
Amazouz and Pantea (2006)	ACP+MCP	MP	Secado de maderas
Zhang and Lennox (2004)	MCP	OP, MP	Simulación del proces de producción de penicilina
He et al. (2005)	ADF	MP	Proceso de fabricación del poliéster
Alhoniemi (1999)	SOM	MP, OP	Proceso de producción del acero
Yang et al. (2000)	FPM+RNA	DFP	Reactor FCC
Kampjarvi et al. (2008)	ACP, SOM, RBF	MP, DFP	Proceso de agrietamiento del etileno
Marjanovic et al. (2006)	MMCP	MP	Detección del punto final del proceso
Dunia and Qin (1998a)	ACP	DFP, DFS	Proceso de una caldera
Lee et al. (2004)	ACPTR	DFP, DFS	Proceso de polimerización
Wang and Cui (2005)	ACP	DFS	Proceso de enfriador centrífugo
Wang and Xiao (2004)	ACP	DFS+MP	Unidad de tratamiento de aire

Tabla 3.1: Lista de publicaciones de sensores virtuales [46] (Ver Tabla 3.2 para abreviaciones)

Abreviación del método	Descripción
ACP	Análisis de Componentes Principales
ACPNL	Análisis de Componentes Principales No-Lineales
ACPR	Análisis de Componentes Principales Recursivos
ACPTR	Análisis Componente Principales de Tiempo Retardado
ADF	Análisis Discriminante de Fisher
AG	Algoritmo Genético
AnaRN	Red Neuronal Analítica
eKF	Filtro de Kalman Extendido
FC	Lógica Difusa Combinacional
FMWACP	Análisis de componentes Principales por Fast Moving Window
FPM	Modelo del Primer Principio
LWR	Regresión Ponderada Localmente
MC	Mínimos Cuadrados
MCP	Mínimos Cuadrados Parciales
MCPPE	Mínimos Cuadrados Parciales Ponderados Exponencialmente
MCPR	Mínimos Cuadrados Parciales Recursivos
MBACP	Modelo Basado en el Análisis de Componentes Principales
MMCP	Mínimos Cuadrados Parciales Multi-Vía
NFS	Sistema Neuro-Difuso
PG	Programación Genética
RMCP	Regresión por Mínimos Cuadrados Parciales
PMC	Perceptrón Multi-Capa
PSO	Optimización por Nube de Partículas
RBF	Función de Base Radial
RCP	Regresión por Componentes Principales
RLM	Regresión Lineal Múltiple
RNA	Red Neuronal Artificial
RNMCP	Red Neuronal con Mínimos Cuadrados Parciales
RNR	Redes Neuronales Recurrentes
SOM	Mapa Auto-Organizado
SRM	Método de Regresión paso a paso
SVM	Maquina de Soporte Vectorial
SVR	Regresión de Soporte Vectorial
TS	Modelo Takagi and Sugeno
WBM	Modelo Basado en Wavelet
Tipos de Aplicaciones	
OP	Predicción en linea
DFP	Detección de fallos en Procesos
MP	Monitoreo de Procesos
DFS	Detección de fallo en Sensores

Tabla 3.2: Lista de Abreviaciones

Un punto común de la mayoría de los sensores virtuales presentados en esta sección, es la necesidad de un conocimiento previo del proceso. Si dejamos de lado los sensores virtuales basados en modelos, que están fuera del alcance de esta revisión, se pueden distinguir diferentes niveles de información de los procesos. Un tipo de información es la construcción de características adicionales que describan algunas propiedades relacionadas con el proceso. Con esto se espera que estas características se correlacionen con la variable que se desee estimar y por lo tanto tengan un efecto positivo en su modelado. Otra forma de aplicar el conocimiento del proceso a los sensores virtuales basados en datos es durante las etapas iniciales de modelado (ver sección 3.3). Especialmente los pasos pre-procesamiento requieren una gran atención del desarrollador del modelo, que a menudo tiene que entrevistar a los expertos de procesos con el fin de llevar a cabo la selección de variables, evaluar los resultados del pre-procesamiento, etc.

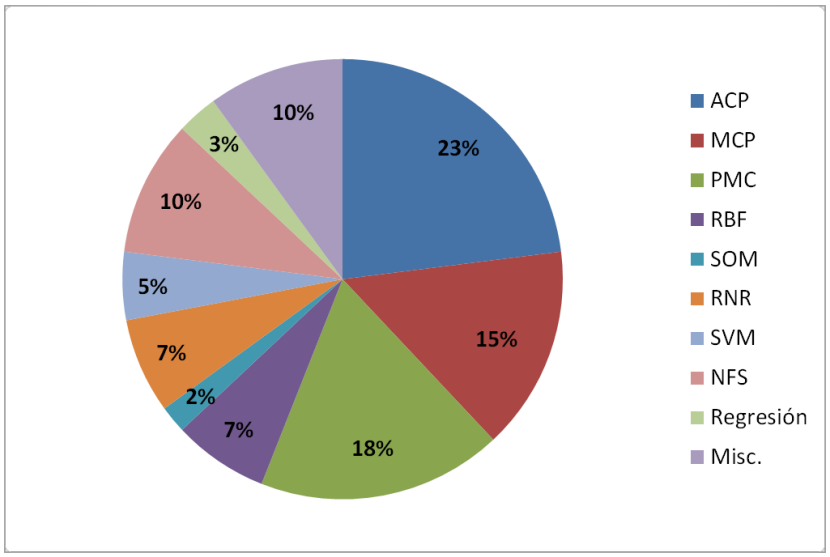


Figura 3.4: Distribución de los métodos de aprendizaje automático en sensores virtuales [46]

3.5. Técnicas de desarrollo para sensores virtuales basados en datos

Si bien las secciones anteriores se centraron en el censado virtual desde el punto de vista de la industria de procesos, en esta sección el mismo tema se tratará, pero desde el punto

de vista del aprendizaje automático. El aprendizaje automático proporciona las herramientas necesarias para el desarrollo y mantenimiento de sensores virtuales.

En esta sección se describen dos métodos muy usados para el desarrollo de sensores virtuales utilizando técnicas de aprendizaje automático. El primer método presentado son las redes neuronales artificiales (RNA) y en particular, su forma usada para el modelado no lineal llamado perceptrón multicapa (PMC). El segundo método presentado es el agrupamiento difuso (FC) y en particular, su variante mas utilizada, el algoritmo Fuzzy C-Means (FCM).

3.5.1. Redes neuronales artificiales (RNA)

La idea original de las RNA [5] era construir modelos computacionales motivados por el funcionamiento de las neuronas biológicas que son las unidades básicas de procesamiento de la información en los sistemas nerviosos. El objetivo tanto de la neurona biológica como de la neurona artificial es recoger señales en la entrada, para procesar esta información, y luego mostrarla a la salida. En el caso de sensores virtuales, el tipo de red neuronal más comúnmente aplicada es el perceptrón multicapa (PMC).

Perceptrón Multicapa (PMC)

Un perceptrón multicapa (PMC) es una red neuronal con alimentación hacia delante (ver Figura 3.5), que consiste en una capa de entrada, una capa de salida, y por lo menos una capa oculta. El papel de la capa de entrada es recoger los datos de entrada y proporcionarlos a la capa oculta para su procesamiento posterior. El número de neuronas en la capa de entrada es equivalente a la dimensionalidad de los datos de entrada. Cada una de las neuronas de la capa de entrada está conectada a cada neurona de la capa oculta, y las conexiones entre las neuronas llevan pesos. El papel de las neuronas en la capa oculta es recoger las señales de la salida de la capa anterior; multiplicarlas por los pesos de conexión; construir una suma de ellos, y procesarlos utilizando las funciones de activación g^{oculta}.

$$x_i^{oculta} = g^{oculta}\left(\sum_j w_{j,i}^{oculta} x_j^{entrada}\right), \tag{3.1}$$

donde $x_j^{entrada}$ es la j-iésima variable de la muestra de entrada, $w_{j,i}^{oculta}$ es el peso entre

la j-iésima entrada y la i-iésima neurona oculta y x_i^{oculta} es la salida de la i-iésima neurona oculta. La función de activación puede ser cualquier función diferenciable no lineal. Muy frecuentemente, se utilizan las funciones sigmoides, tales como:

$$g^{oculta}(x) = \frac{1}{1 + exp(-x)}. \tag{3.2}$$

Una vez que las señales son procesadas por la capa oculta se pasan a la siguiente capa oculta, en caso de que haya más de una capa oculta, o de otro modo se pasan a la capa de salida. La capa de salida puede tener una o más neuronas. En el caso de típicos modelos de regresión, la capa de salida consta de una sola neurona:

$$y^{salida} = g^{salida}\left(\sum_j w_i^{salida} x_i^{oculta}\right), \tag{3.3}$$

con y^{salida} como la salida del perceptrón multicapa, w_i^{salida} el peso entre la i-iésima neurona oculta y la neurona de salida, y g^{salida} como la función de activación de la neurona de salida.

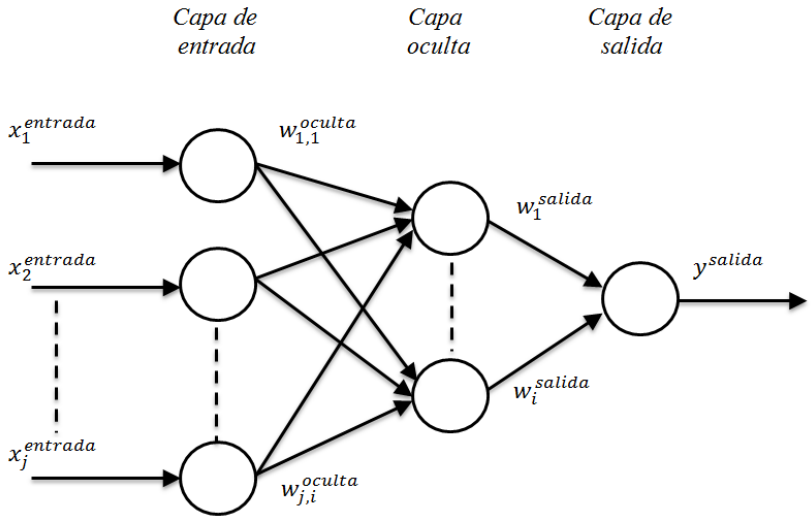


Figura 3.5: Estructura del perceptrón multicapa

Una propiedad teórica notable del PMC es que son aproximadores universales, lo que

significa que con suficientes datos de entrenamiento y una estructura lo suficientemente compleja, pueden ser entrenados para aproximar cualquier función con una precisión determinada [19].

Función de activación

La función de activación se utiliza para limitar el rango de valores de la respuesta de la neurona. Generalmente los rangos de valores se limitan a [0,1] o [-1,1], sin embargo otros rangos son posibles de acuerdo a la aplicación o problema a resolver. Las más usadas son:

- Función sigmoidal (no lineal)

$$logsig(x) = \frac{1}{1 + e^{-x}}. \tag{3.4}$$

- Función sigmoidal tangente hiperbólica

$$tansig(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3.5}$$

- Función lineal

$$pureline(x) = x. \tag{3.6}$$

Como ya se ha comentado anteriormente, se puede utilizar cualquier función siempre y cuando sea continua y diferenciable.

En las capas ocultas del PMC suele utilizarse funciones de activación no lineales, y en la capa de salida se pueden utilizar de tipo lineal o no lineal.

En el caso de utilizar un PMC para realizar aproximaciones de funciones, se recomienda usar en las neuronas de la capa de salida una función lineal. Si el PMC se utiliza para realizar clasificación, se recomienda utilizar la función sigmoidal tangente hiperbólica en la capa oculta y una función sigmoidal en la capa de salida.

Algoritmo de aprendizaje

El aprendizaje del PMC (y en general de RNA) se logra mediante la aplicación de un algoritmo para encontrar los pesos óptimos entre las neuronas. Los más populares de estos algoritmos son los algoritmos de retropropagación (backpropagation) [68].

Antes de aplicar cualquier algoritmo de aprendizaje, se debe saber que cada algoritmo tiene características especiales que lo hacen adecuado a determinadas circunstancias. El diseñador de la red en función a ciertas circunstancias en particular (problema a resolver, tipo de plataforma disponible, etc.) debe escoger debe escoger el algoritmo más adecuado.

Los algoritmos de retropropagación clásicos para entrenar redes neuronales son el gradiente descendiente, el algoritmo Quasi-Newton y el algoritmo de Levenberg-Marquardt.

Algoritmo de gradiente descendiente. El algoritmo de gradiente descendiente es uno de los más utilizados en el estado de la técnica. Si se define $E(w)$, como la función de error en función de los pesos w de la red, el aprendizaje busca un mínimo global de esta función de error.

De forma iterativa, se puede expresar como que, dados los pesos de la red $w(0)$ para el instante $n = 0$ y un factor de aprendizaje μ, se calcula la dirección de mayor variación de la función de error, que viene dada por el gradiente $\Delta E(w)$. Posteriormente, se actualizan los pesos, en los del sentido contrario a los de la variación de la función de error:

$$w(n+1) = w(n) - \mu \cdot \Delta E(w). \tag{3.7}$$

Algoritmo Quasi-Newton. Puede que la dirección del gradiente no sea la que más rápido converja hacia el mínimo de la función de error. De esta manera, existen modelos que actualizan sus pesos en base a la expresión:

$$w(n) = w(n-1) + \mu(n) \cdot d(n), \tag{3.8}$$

donde $d(n)$, indica la dirección de búsqueda de la iteración n.

Esta dirección, es la conjugada de las $(n-1)$ direcciones anteriores, por lo que, se puede generalizar la expresión anterior como:

$$w(n) = w(n-1) - H(w(n-1))^{-1} \cdot \Delta E(w), \tag{3.9}$$

donde H, es la matriz Hessiana.

El método de Newton puede que no converja (si el punto inicial no está suficientemente cerca del óptimo), pero si lo hace, su rapidez, es del orden de diez a cien veces superior a la del gradiente descendiente.

Algoritmo Levenberg-Marquardt. El algoritmo de Levenberg-Marquardt es una modificación del método de Newton, que elimina el cálculo de la matriz Hessiana. La actualización de los pesos sigue la expresión:

$$w(n) = w(n-1) + \mu(n) \cdot d(n). \tag{3.10}$$

Al mismo tiempo, la magnitud del cambio de los pesos viene dada por la expresión:

$$\Delta w(n) = -\mu \cdot M(w) \cdot \Delta E(w). \tag{3.11}$$

Si $M(w) = I$, se convierte en el algoritmo de gradiente descendiente.

Si $M(w) = H - 1(w)$, se tiene el método de Newton.

Se suele establecer $M(w)$ en un valor de compromiso, $M(w) = [\alpha \cdot I + H(w(n))]^{-1}$.

Por lo tanto, la actualización de los pesos sigue la expresión:

$$\Delta w(n) = w(n-1) - \mu(n) \cdot [\alpha \cdot I + H(w(n))]^{-1} \cdot \Delta E(w), \tag{3.12}$$

donde I es la matriz identidad y α un factor de amortiguamiento.

El valor de μ es inicialmente puesto a algún valor, normalmente $\mu = 10^{-3}$. Si el valor de $\Delta E(w)$ obtenido resolviendo las ecuaciones conduce a una reducción del error, entonces

el incremento es aceptado y μ es dividido por un factor de decremento para la siguiente iteración. Por otro lado si el valor de $\Delta E(w)$ conduce a un aumento del error, entonces μ es multiplicado por un factor de incremento y se resuelven de nuevo las ecuaciones, este proceso continúa hasta que el valor de $\Delta E(w)$ encontrado da lugar a un decremento del error. Este proceso de resolver repetidamente las ecuaciones para diferentes valores de μ hasta encontrar un valor aceptable de $\Delta E(w)$ es lo que constituye una iteración del algoritmo de LM.

En comparación con los algoritmos de gradiente descendiente, el algoritmo de Levenberg-Marquardt requiere un mayor esfuerzo computacional, pero su convergencia es más rápida, además de que tiene un excelente desempeño en el entrenamiento de redes neuronales donde el rendimiento de la red este determinado por el error cuadrático medio.

3.5.2. Agrupamiento Difuso (FC)

El agrupamiento difuso (en inglés, fuzzy clustering) pertenece a los algoritmos de aprendizaje automático no supervisado, donde no es necesario un conocimiento a priori para el proceso de aprendizaje. El objetivo es obtener información útil sobre la estructura de un conjunto de patrones dado, los cuales son divididos en grupos (clusters) [45].

Este tipo de algoritmos surge de la necesidad de resolver una deficiencia del agrupamiento exlusivo, que considera que cada elemento se puede agrupar inequívocamente con los elementos de su cluster y que, por lo tanto, no se asemeja al resto de los elementos. Tras la introducción de la lógica difusa por Zadeh en 1965 surgió una solución para este problema, caracterizando la similitud de cada elemento a cada uno de los grupos. Esto se logra representando la similitud entre un elemento y un grupo por una función, llamada función de pertenencia, que toma valores entre cero y uno. Los valores cercanos a uno indican una mayor similitud, mientras que los cercanos a cero indican una menor similitud. Por lo tanto, el problema del agrupamiento difuso se reduce a encontrar una caracterización de este tipo que sea óptima.

Una técnica difusa bastante conocida y que ha cobrado importancia en la tarea de clustering o agrupamiento es el algoritmo C-Medias Difuso (en ingles, Fuzzy C-Means (FCM)) el cuál es una extensión difusa del conocido C-Means.

Fuzzy C-Means (FCM)

En muchas situaciones cotidianas ocurre el caso que un dato está lo suficientemente cerca de dos clusters de tal manera que es difícil etiquetarlo en uno o en otro, esto se debe a la relativa frecuencia con la cuál un dato particular presenta características pertenecientes a clusters distintos y como consecuencia no es fácilmente clasificado; *fuzzy c-means (FCM)* es un algoritmo que se desarrolló con el objetivo de solucionar tales inconvenientes.

Este algoritmo funciona mediante la asignación de grados de pertenencia a cada dato correspondiente a cada centro de un cluster sobre la base de la distancia entre el centro de la agrupación y el dato. Mientras mas cerca estén los datos del centro del cluster, mayor es su pertenencia hacia un determinado grupo. Claramente, la suma de la pertenencia de cada dato debe ser igual a uno. Después de cada iteración los grados de pertenencia y los centros se actualizan de acuerdo con la fórmula [28]:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{ik}} \right)^{\frac{2}{m}-1}}, \tag{3.13}$$

$$v_j = \frac{\left(\sum_{i=1}^{n} (\mu_{ij})^m x_i \right)}{\left(\sum_{i=1}^{n} (\mu_{ij})^m \right)}, \tag{3.14}$$

donde:

n es el número de datos.

v_j representa el j^{iesimo} centro del cluster.

m es el indice de fuzificación $m \, \epsilon \, (1, \infty)$

c representa el número de centros de clusters.

μ_{ij} representa la pertenencia del i^{iesimo} dato al j^{iesimo} centro de clusters.

d_{ij} representa la distancia euclidiana entre el i^{iesimo} dato y el j^{iesimo} centro de cluster.

El objetivo principal del algoritmo *fuzzy c-means* es minimizar:

$$J(U,V) = \sum_{i=1}^{n}\sum_{j=1}^{c}(\mu_{ij})^m \|x_i - v_j\|^2, \tag{3.15}$$

donde:

$\|x_i - v_j\|$ es la distancia euclidiana entre el i^{iesimo} dato y el j^{iesimo} centro de cluster.

Algoritmo *fuzzy c-means*

Sea $X = x_1, x_2, x_3, ..., x_n$ el conjunto de datos y $V = v_1, v_2, v_3, ..., v_c$ el conjunto de centros.

1. Seleccionar el número de centros 'c' de manera aleatoria.
2. Calcular la pertenencia difusa 'μ_{ij}' usando:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m}-1}}, \tag{3.16}$$

3. Estimar los centros difusos 'v_j' utilizando:

$$v_j = \frac{\left(\sum_{i=1}^{n}(\mu_{ij})^m x_i\right)}{\left(\sum_{i=1}^{n}(\mu_{ij})^m\right)}, \tag{3.17}$$

4. Repetir el paso 2 y 3, hasta que se alcance el mínimo valor de la función objetivo 'J' o algún criterio de paro

Alguno de los criterios de paro mas utilizados son:

- Un número máximo de iteraciones.
- Que la variación en la matriz U sea muy pequeña: $\|U^{k+1} - U^k\| < \beta$, donde:

  k es la iteración.

  β es el criterio de paro entre [0,1].

  $U = (\mu_{ij})_{n*c}$ es la matriz difusa de pertenencia.

Ventajas

1. Da mejor resultado para el conjunto de datos traslapados.

2. A diferencia de otros algoritmos, donde un dato debe pertenecer exclusivamente a un grupo o cluster, este algoritmo asigna un valor de pertenencia a cada dato como consecuencia un dato puede pertenecer a más de un grupo.

Desventajas

1. Previamente se debe especificar un número de centros.

2. Con menor valor de β se obtiene un mejor resultado, pero a expensas de un mayor número de iteraciónes.

Parte II

Desarrollo

Capítulo 4

Caso de estudio

Como caso de estudio se presenta un nuevo y alternativo método para medir la interacción entre las pinzas de un brazo robótico y los objetos que sostiene, detectando formas y tamaños de los objetos. En lugar de utilizar una celda de carga para medir la fuerza de interacción resultante, se propone una medida por retroalimentación indirecta de fuerza, midiendo la corriente consumida por un actuador. Y en vez de utilizar una cámara para reconocimiento de formas y tamaños de los objetos se implementa un sensor virtual capaz de reconocer y clasificar objetos. La ventaja de este enfoque es que no solo permite conocer si existe interacción entre un objeto y las pinzas del brazo, sino que también se podrá conocer la forma y el tamaño del objeto que se sostiene. Por otra parte, la solución propuesta implica la reducción de costos al eliminar sensores físicos. El sensor virtual se implementa en las pinzas de un brazo robótico capaz de jugar ajedrez contra humanos, con el fin de estimar el tipo de pieza de ajedrez que se está sujetando.

En este capítulo se construye un sensor virtual táctil capaz de estimar que pieza de ajedrez se sujeta con las pinzas de un brazo robótico, siguiendo la metodología de diseño propuesta en el marco teórico (ver Sección 3.3), cuyos pasos son descritos a continuación:

4.1 Descripción del proceso.

4.2 Inspección de datos.

4.3 Selección de datos históricos y detección de estados estacionarios.

4.4 Preprocesamiento de datos.

4.5 Selección del modelo, entrenamiento y validación.

4.6 Mantenimiento del sensor virtual.

4.1. Descripción del proceso

En esta sección se explica el proceso de detección y sujeción de las piezas de ajedrez con las pinzas del brazo robótico y se dan a conocer las variables que se tienen disponibles para ser medidas.

4.1.1. Descripción del brazo robótico

El brazo robótico que se utiliza para sujetar las piezas de ajedrez lleva por nombre MARCO (Multimodal Agent RoboChess Operator) y forma parte de un proyecto realizado en Alemania durante una estancia de investigación, el brazo es capaz de jugar al ajedrez contra humanos de forma autónoma, esto es posible gracias al uso de un tablero de ajedrez digital (DGT USB Walnut e-Board) para conocer la posición de las piezas y un algoritmo de búsqueda heurística (Tom Kerrigan's Simple Chess Program (TSCP)) para jugar ajedrez. El brazo fue diseñado para mover las piezas a través de todo el tablero de ajedrez, tiene una longitud de 60 cm y puede levantar objetos de hasta 200 g (ver Apéndice 1. Dimensiones del brazo robótico). Para construir el brazo robótico, fueron utilizados cinco servomotores Dynamixel de cuatro familias diferentes, cada uno con características diferentes (ver Apéndice 2. Especificaciones de la familia Dynamixel). El servomotor MX-28 Dynamixel fue usado para la base y la muñeca del robot, el codo usa un Dynamixel MX-64, el hombro utiliza un Dynamixel MX-106 y para las pinzas fue utilizado un Dynamixel AX-12A (ver Figura 4.1).

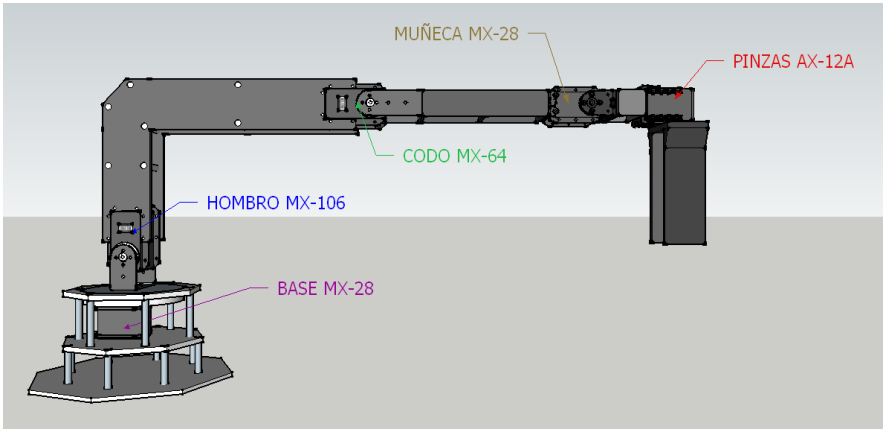


Figura 4.1: Brazo róbotico

Cada servomotor utilizado tiene la capacidad de controlar su velocidad, temperatura, posición, torque, voltaje y corriente soportada. El algoritmo de control utilizado para mantener la posición de cada servo puede ajustarse individualmente, permitiendo el control retroalimentado de la velocidad y la corriente que soporta cada actuador.

Las pinzas del brazo fueron construidas específicamente para sostener todas las piezas del ajedrez, independientemente de su forma o tamaño, para esto, se utilizó un diseño de agarre flexible por medio de una goma que forma una estructura que se ajusta a los objetos. La pinza normalmente tiene dos posiciones, ya sea abierta o cerrada.

El servomotor utilizado para la pinzas es un Dynamixel AX-12A, posee un microcontrolador integrado que entiende 50 comandos, la mayoría de los cuales fijan o leen parámetros que definen su comportamiento. Estos parámetros leídos pueden ser la posición actual, la corriente consumida, o la variación de la temperatura del servo con la carga aplicada en el mismo, lo que permite un control retroalimentado sofisticado controlando el torque que soporta cada articulación del robot.

En la lista siguiente podemos encontrar las principales características del Dynamixel AX-12A.

Características del AX-12A:

- Peso: 54.6 gr
- Dimensión: 32 mm x 50 mm x 40 mm
- Resolución: 0.29°
- Ratio de reducción: 254 : 1
- Torque motor: 1.52 N.m (a 12.0 V, 1.5 A)
- Velocidad sin carga: 59 rpm (a 12 V)
- Grados de giro: 0° 300°
- Rotación continua
- Temperatura de trabajo: -5ºC +70ºC
- Voltaje de operación: 9 12 V (Voltaje de operación recomendado 11.1 V)
- Señal de comandos: paquete digital
- Tipo de protocolo: comunicación serie asíncrona half duplex (8 bit,1 stop, no parity)
- Conexión física: TTL Level Multi Drop (conector tipo daisy chain)

- ID: 254 ID (0 253)
- Velocidad de comunicación: 7343 bps 1 Mbps
- Feedback: posición, temperatura, corriente, voltaje de entrada y velocidad.
- Material: plástico

4.1.2. Detección de piezas

El diseño de las piezas de ajedrez utilizadas son del tipo *Staunton*, en honor al campeón de ajedrez Howark Staunton (1810 - 1874). Es el diseño usado para las competiciones oficiales de ajedrez [1].

Figura 4.2: Diseño de piezas de ajedrez *Staunton*

Existen medidas estándar para este diseño, mostradas en la tabla 4.1.

Pieza	**Altura**	**Diámetro de la base**
Rey	9.5 cm	3.5 cm
Reina	8 cm	3.5 cm
Alfil	6.5 cm	3.1 cm
Caballo	6.3 cm	3.1 cm
Torre	5 cm	3.1 cm
Peón	4.4 cm	2.6 cm

Tabla 4.1: Detalles de las piezas de ajedrez

[1] Standards of Chess Equipment and tournament venue for FIDE Tournaments, FIDE, retrieved 2009-02-22

Como se puede apreciar en la Figura 4.2 y en la Tabla 4.1, todas las piezas tienen diferentes formas y tamaños, características que permitirán que las piezas puedan ser clasificadas.

Para la detección de las piezas de ajedrez en las pinzas del brazo robótico se utiliza una realimentación indirecta de fuerza, para ello se mide la corriente que consume el servomotor que controla las pinzas mientras sujeta una pieza de ajedrez.

Una de las ventajas que tiene el servomotor AX-12A es que puede proporcionar sus parámetros de forma digital, por lo tanto, solo es necesario enviar un comando de petición para obtener el valor de la corriente consumida por el servomotor en ese instante.

De la figura 4.3 a la figura 4.9 se presentan las gráficas del comportamiento de la corriente con respecto al tiempo durante la sujeción de cada pieza de ajedrez.

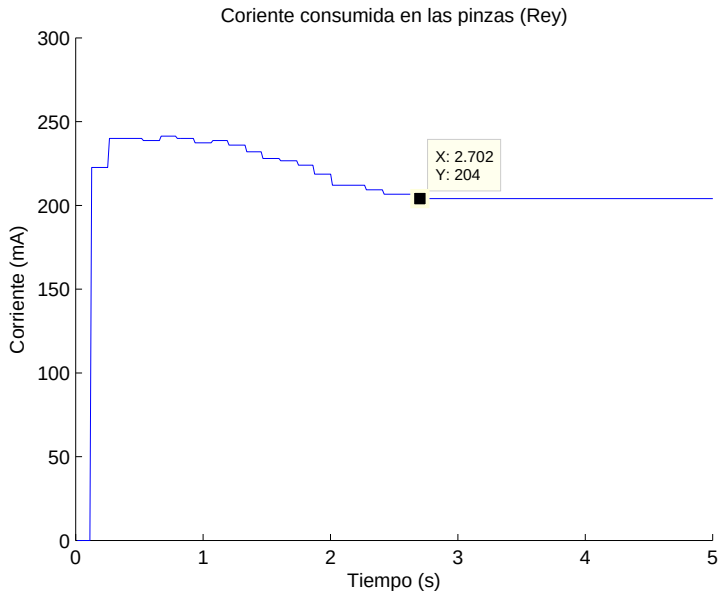


Figura 4.3: Gráfica de corriente-tiempo para rey

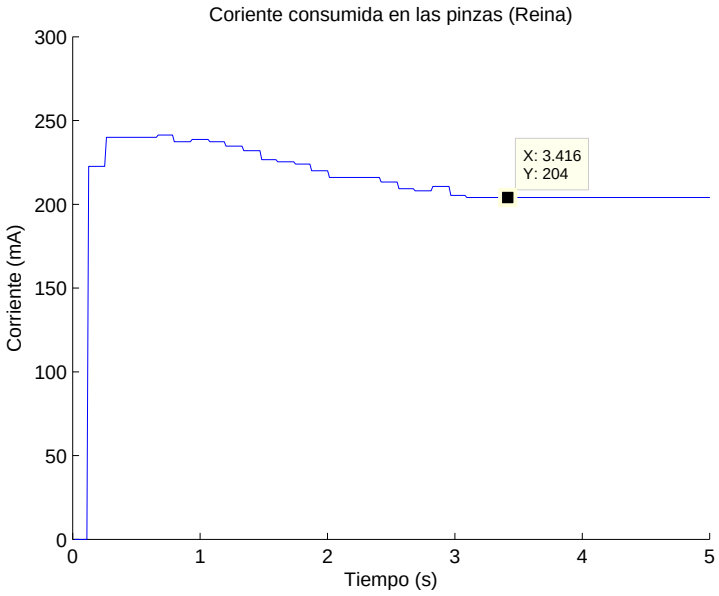


Figura 4.4: Gráfica de corriente-tiempo para reina

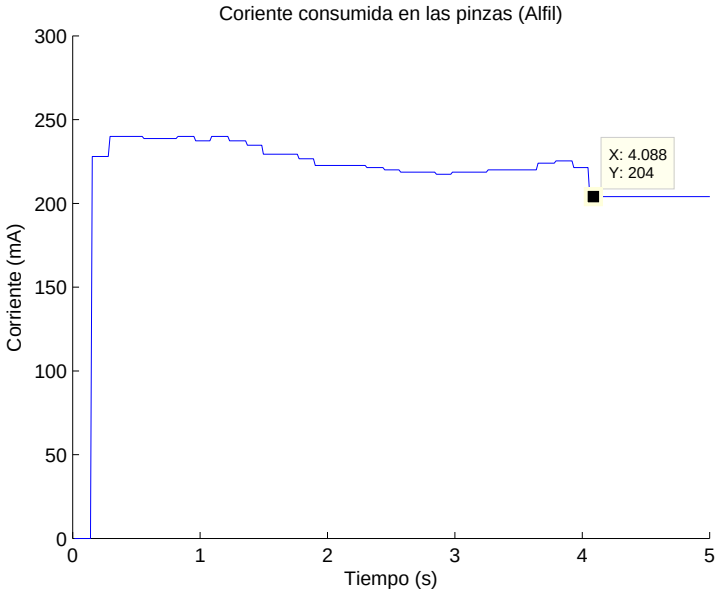


Figura 4.5: Gráfica de corriente-tiempo para alfil

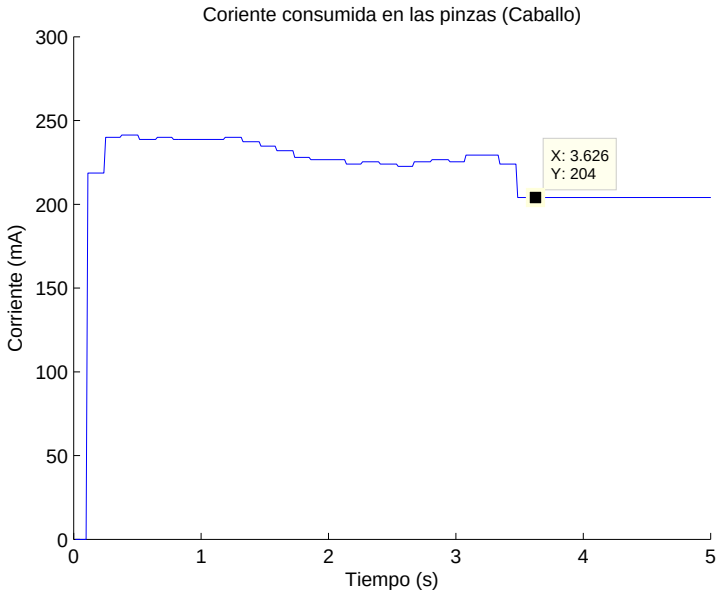


Figura 4.6: Gráfica de corriente-tiempo para caballo

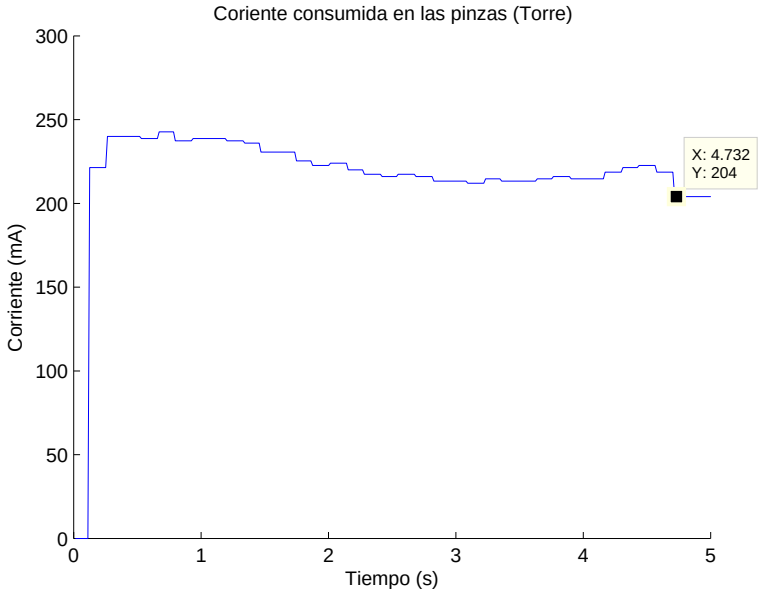


Figura 4.7: Gráfica de corriente-tiempo para torre

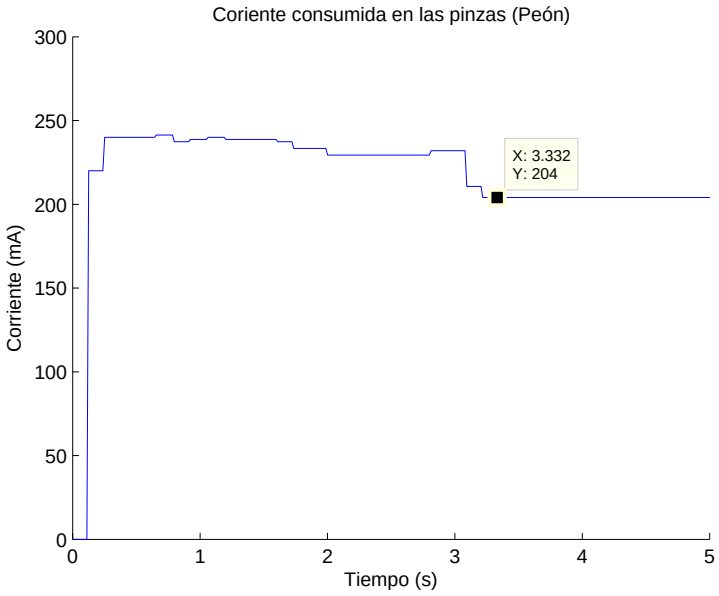


Figura 4.8: Gráfica de corriente-tiempo para peón

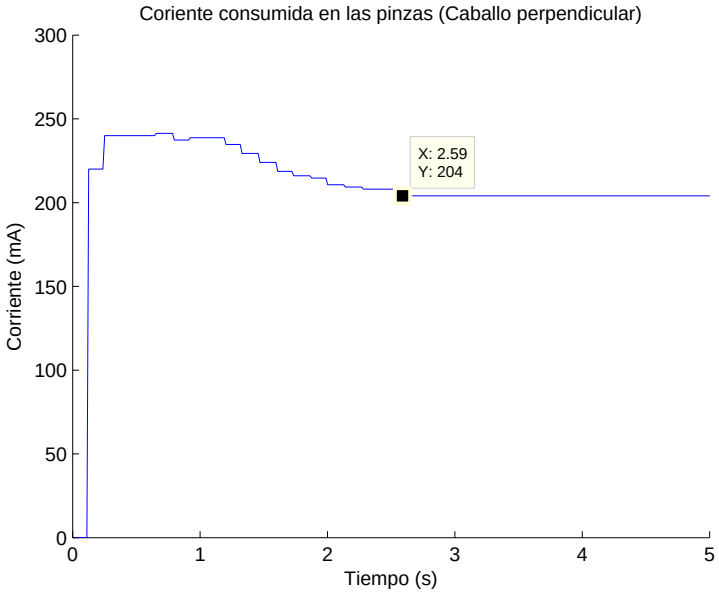


Figura 4.9: Gráfica de corriente-tiempo para caballo perpendicular

Como se puede apreciar en las gráficas anteriores (Figura 4.3 a 4.9), la corriente converge en el mismo punto para todas las piezas (204 mA), pero el tiempo de convergencia es di-

ferente para cada una de las piezas. Existe un caso particular para el caballo, el cual es la única pieza del ajedrez que no es simétrica, por lo tanto puede ser sostenido por las pinzas de dos formas diferentes; ya sea perpendicular a las pinzas o paralelo a las pinzas. En la tabla 4.2 se muestra el tiempo de convergencia de la corriente para cada una de las piezas de ajedrez.

Pieza	**Tiempo de convergencia de la corriente**
Rey	2.70 s
Reina	3.41 s
Alfil	4.08 s
Caballo	3.62 s
Torre	4.73 s
Peón	3.33 s
Caballo perpendicular	2.59 s

Tabla 4.2: Tiempo de convergencia de la corriente para cada pieza de ajedrez

Debido a la diferencia en forma y tamaño de cada pieza de ajedrez, el tiempo de convergencia es diferente para cada una de ellas, información que puede ser utilizada más adelante para la clasificación de las piezas en el sensor virtual. Por ahora el proceso de detección solamente se centra en predecir cuando existe una pieza en las pinzas, sin importar que pieza sea, y esto se logra estableciendo el punto de convergencia como indicador de que existe una pieza en las pinzas, es decir, cuando la corriente alcanza los 204 mA y se mantiene en ese valor durante un tiempo predeterminado (50 ms), el algoritmo de detección (ver Figura 4.10) deduce la existencia de una pieza en las pinzas y detiene el giro del servomotor para que deje de ejercer fuerza sobre la pieza.

En el diagrama de flujo de la figura 4.10 se explica el proceso de detección de las piezas de ajedrez. En primera instancia se comienzan a cerrar las pinzas del brazo robótico, mientras esto sucede la corriente del servomotor es monitoreada, cuando la corriente alcanza el valor de 204 mA se incrementa un contador a uno, la corriente vuelve a ser leída y mientras el valor de la corriente permanezca en 204 mA, el contador seguirá aumentando hasta alcanzar cinco incrementos consecutivos, con esto se asegura la detección de una pieza y se descarta la posibilidad de un error de medición. Una vez que se han alcanzado los cinco incrementos consecutivos, se detiene el servomotor y se reinician los valores. El código para la detección

de las piezas se muestra en Apéndice 3. Código de detección de piezas con ARDUINO.

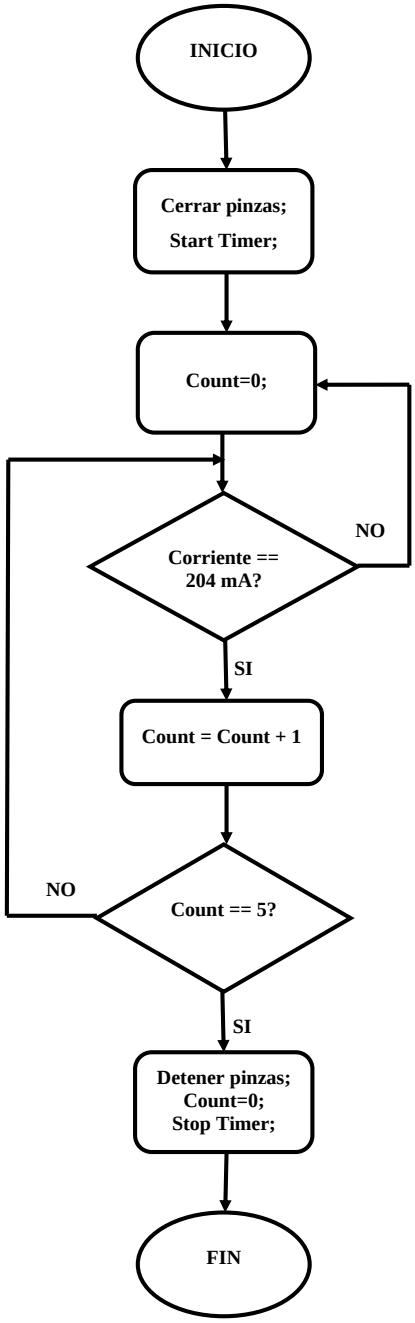


Figura 4.10: Diagrama de flujo del algoritmo de detección de piezas

4.2. Inspección de datos

En esta sección se analizarán los datos que se tienen disponibles para la implementación del sensor virtual, también se dan sugerencias sobre el tipo de modelo que se utilizará para el desarrollo del sensor virtual.

En la sección anterior se conocieron las variables disponibles que se pueden obtener directamente del servomotor y que se analizarán en esta sección, las cuales son:

- ⋆ Posición.
- ⋆ Temperatura.
- ⋆ Corriente.
- ⋆ Voltaje.
- ⋆ Velocidad.
- ⋆ Tiempo de detección.

Posición: La posición del servomotor es una variable que sin duda proporciona información valiosa para la clasificación de las piezas, debido a que la diferencia de tamaños y formas entre las piezas, da como resultado que las pinzas se detengan en diferentes posiciones durante el proceso de detección de piezas. La posición del servomotor es de 150° cuando la pinza se encuentra abierta y 0° cuando la pinza se encuentra cerrada, y dependiendo de la pieza que se sujeta, la posición de las pinzas varía para cada una de las piezas. De la figura 4.11 a la figura 4.17 se presentan las gráficas de posición durante el proceso de sujeción de cada una de las piezas de ajedrez.

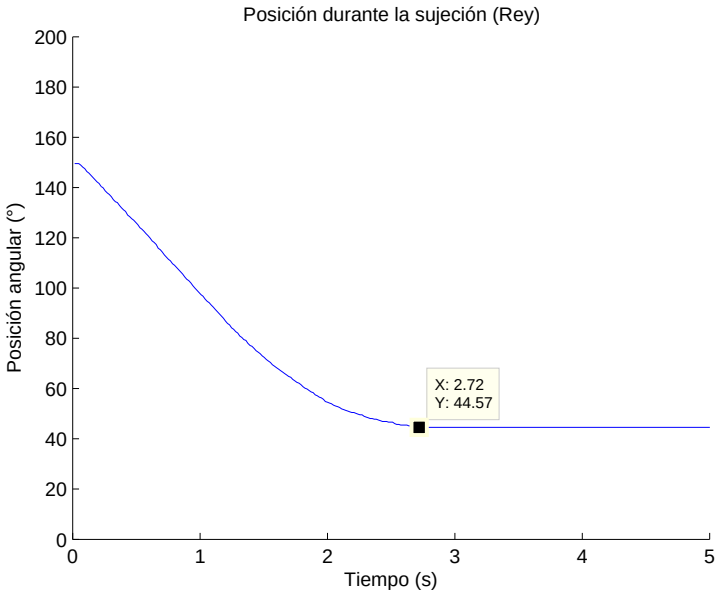


Figura 4.11: Gráfica de posición-tiempo para rey

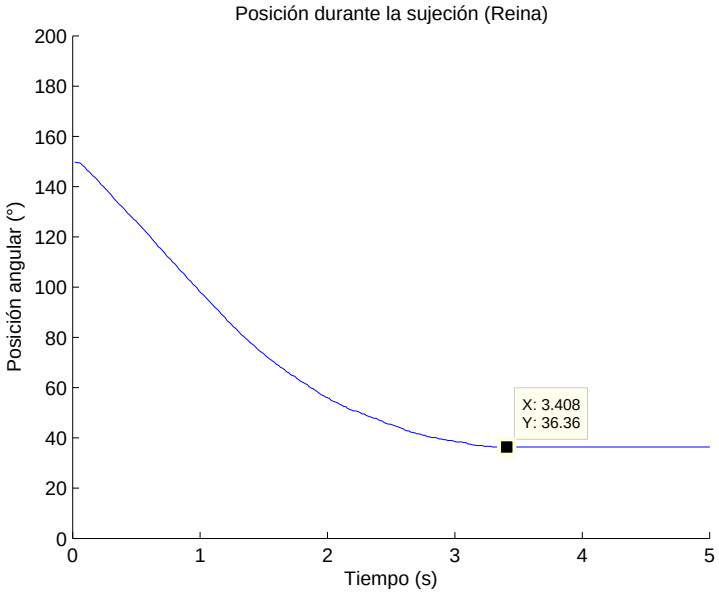


Figura 4.12: Gráfica de posición-tiempo para reina

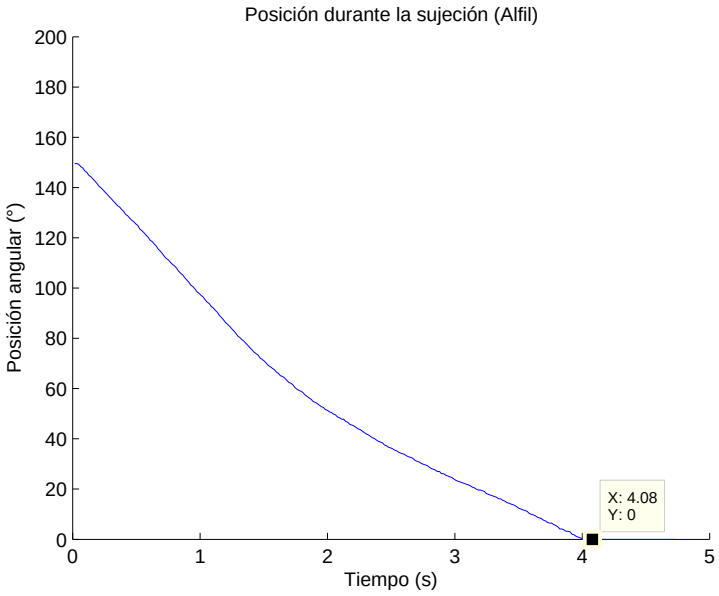


Figura 4.13: Gráfica de posición-tiempo para alfil

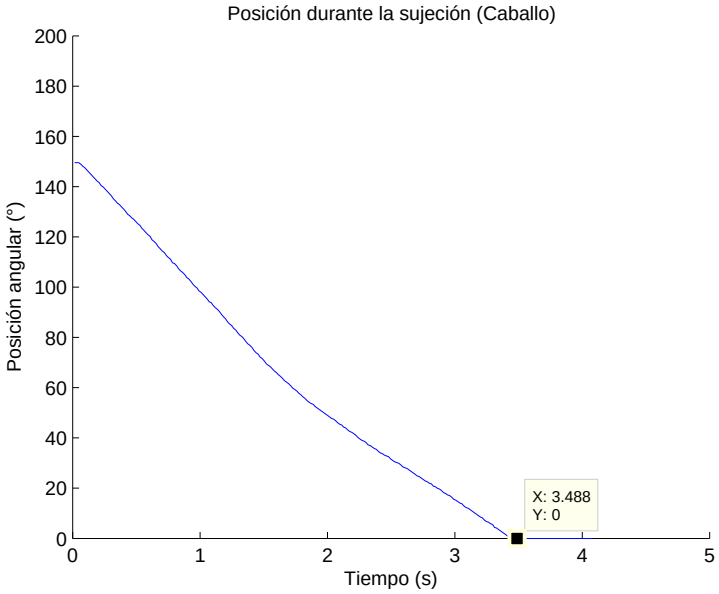


Figura 4.14: Gráfica de posición-tiempo para caballo

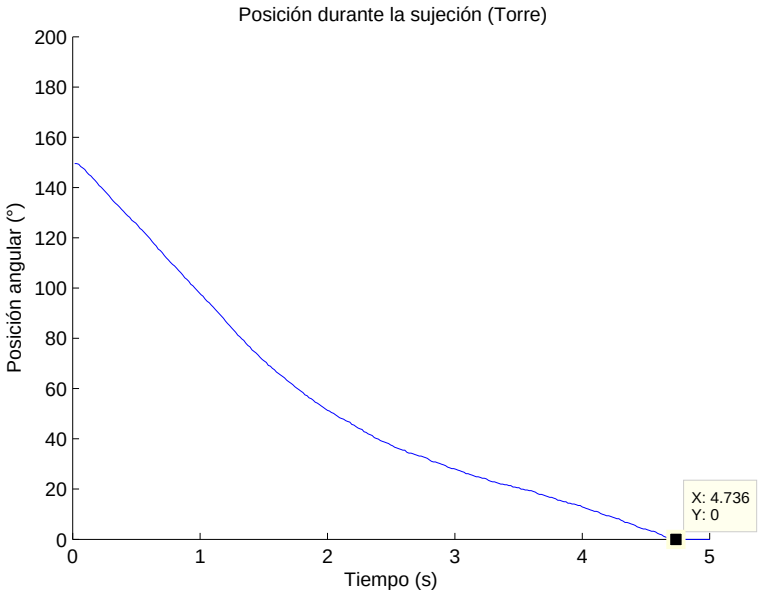


Figura 4.15: Gráfica de posición-tiempo para torre

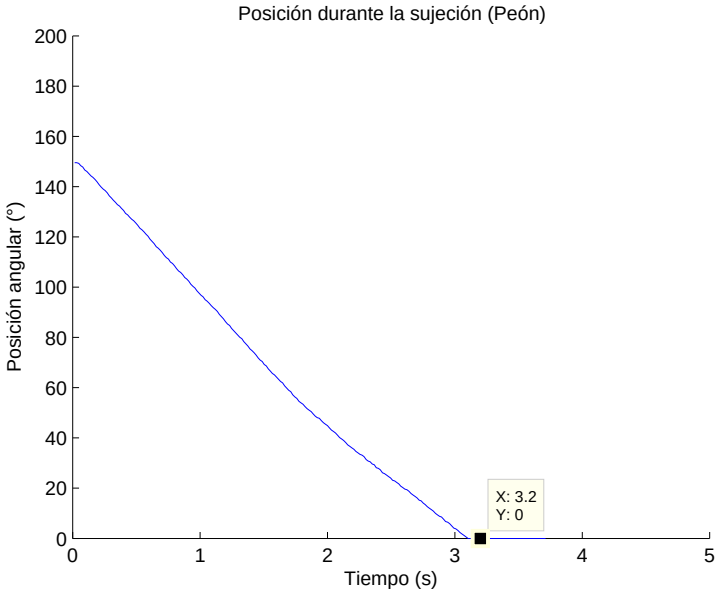


Figura 4.16: Gráfica de posición-tiempo para peón

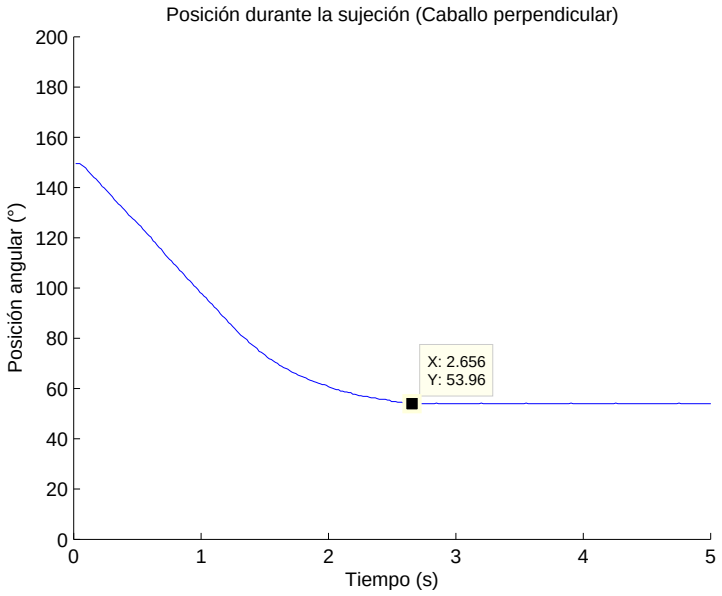


Figura 4.17: Gráfica de posición-tiempo para caballo perpendicular

En las gráficas anteriores (Figura 4.11 a 4.17) se observa que para cada pieza, la posición llega a un punto de convergencia diferente, en la tabla siguiente se observa la posición final de detección para cada pieza:

Pieza	**Posición**
Rey	44.57°
Reina	36.36°
Alfil	0°
Caballo	0°
Torre	0°
Peón	0°
Caballo perpendicular	53.96°

Tabla 4.3: Posición de detección para cada pieza del ajedrez

Las piezas más grandes; como el rey y la reina, detienen las pinzas antes de llegar a la posición cero, así como el caso del caballo perpendicular, pero para las piezas más pequeñas; como el alfil, el caballo, la torre y el peón, las pinzas se cierran por completo, esto debido al diseño flexible de las pinzas, por lo tanto para la clasificación de estas cuatro piezas es necesario encontrar otra variable que tenga la suficiente variación en sus valores.

Temperatura: Durante la sujeción de las piezas no se aprecian cambios significativos en la temperatura del servomotor, el aumento o disminución de la temperatura ocurre de forma tan lenta, que es imposible detectar cambios en instantes de tiempo tan cortos, como en la sujeción de las piezas. Sin embargo es una variable que debe ser monitoreada constantemente para garantizar el correcto funcionamiento del servomotor. La temperatura de operación normal del servomotor tiene un promedio de 40°C.

Corriente: La corriente siempre converge a un punto en común para todas las piezas (ver sección 4.1), por lo tanto es una variable con poca información, debido a que no se puede extraer ninguna característica de ella, además de tener un comportamiento inestable.

Voltaje: La alimentación de los servomotores es por medio de una fuente de 12V a 10A. Durante la operación normal los servomotores tienen una diferencia de potencial de 11.4V y esta tensión se mantiene constante, por lo tanto el voltaje no posee información valiosa para la clasificación.

Velocidad: La velocidad es una variable que puede proporcionar información importante acerca del comportamiento del servomotor durante la sujeción de las piezas de ajedrez, de la figura 4.18 a la figura 4.24 se presentan las gráficas de velocidad angular (en rpm) durante la sujeción de cada una de las piezas de ajedrez.

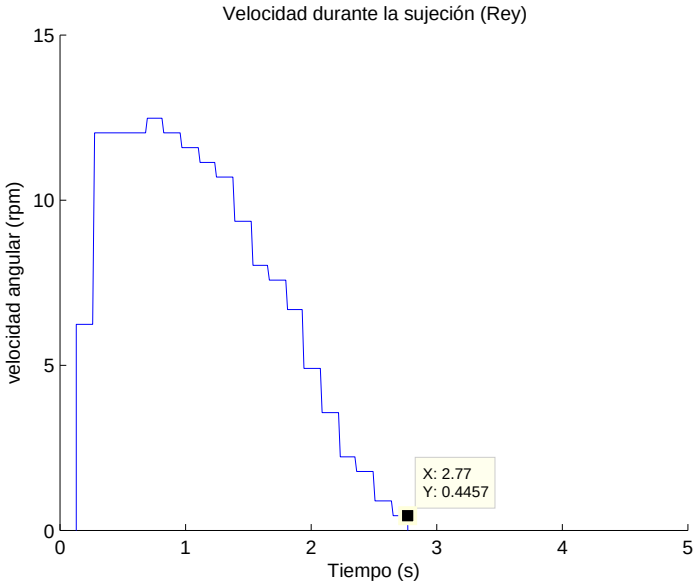


Figura 4.18: Gráfica de velocidad-tiempo para rey

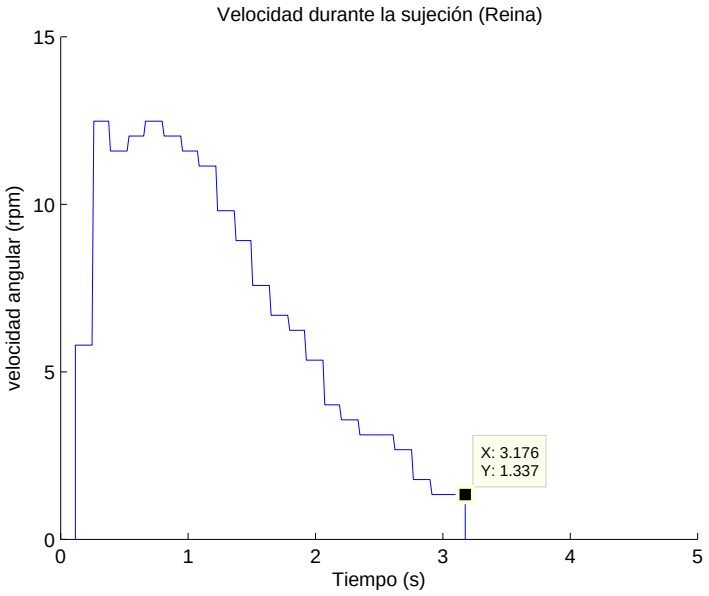


Figura 4.19: Gráfica de velocidad-tiempo para reina

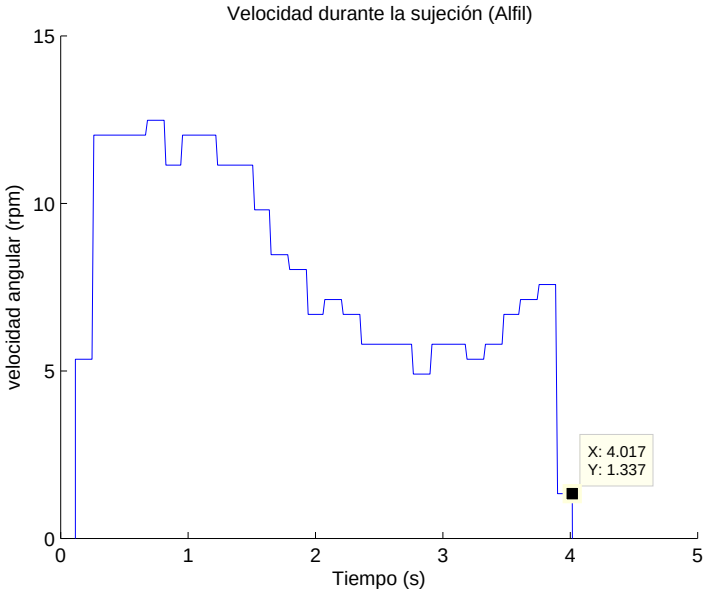


Figura 4.20: Gráfica de velocidad-tiempo para alfil

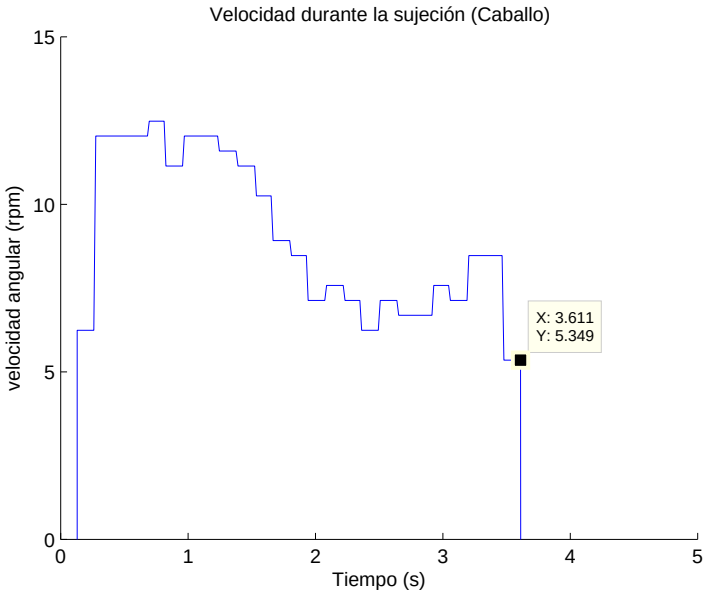


Figura 4.21: Gráfica de velocidad-tiempo para caballo

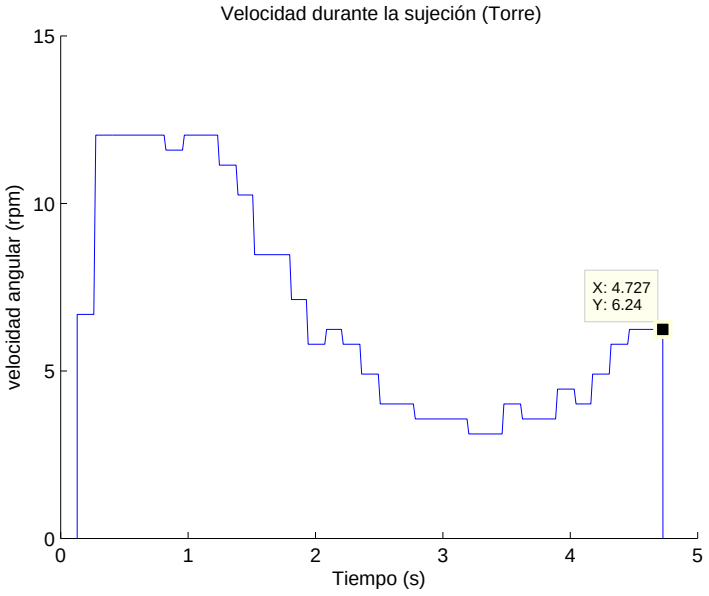


Figura 4.22: Gráfica de velocidad-tiempo para torre

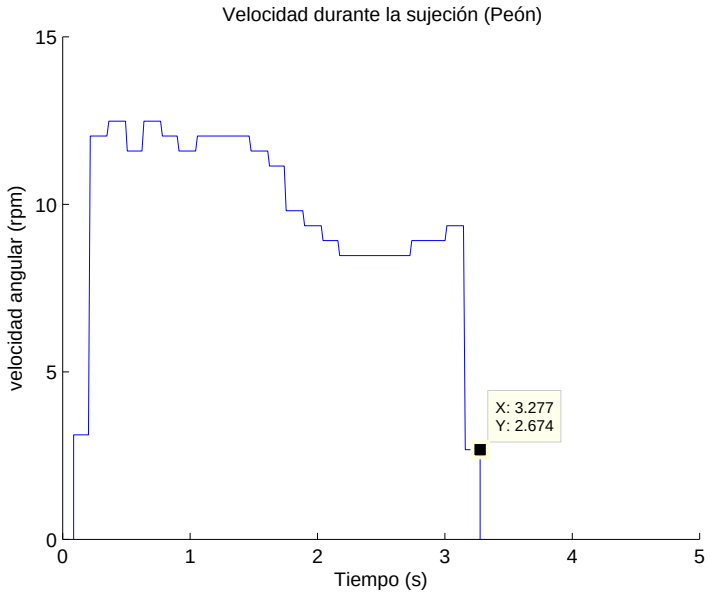


Figura 4.23: Gráfica de velocidad-tiempo para peón

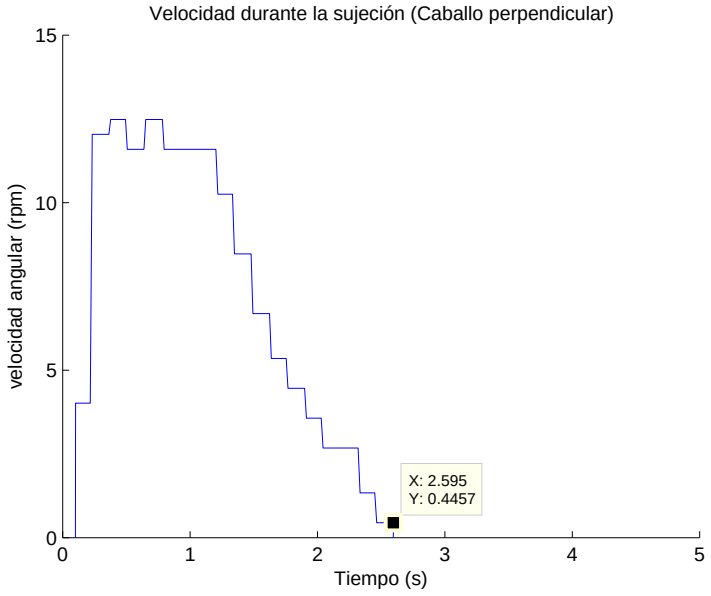


Figura 4.24: Gráfica de velocidad-tiempo para caballo perpendicular

En las gráficas anteriores (Figura 4.18 a 4.24) se puede observar que la velocidad alcanza un punto máximo para todas las piezas (12 rpm aproximadamente) y va decreciendo conforme

las pinzas van detectando cada pieza de ajedrez hasta llegar a un valor de cero, lo cual significa que el servomotor ha detenido su movimiento por completo y la pieza ha sido sujetada. Debido a la variación inestable en el comportamiento de la velocidad, es decir que presenta valores aleatorios que no siempre se comportan de la misma manera en cada iteración, no es viable utilizar la velocidad para extraer información, sin embargo se aprecia que el tiempo donde la velocidad decrece hasta un valor cero, es diferente para cada una de las piezas, por lo tanto se puede extraer información valiosa de dicha característica. En la tabla 4.4 se muestra el tiempo en que la velocidad converge a cero para cada una de las piezas.

Pieza	**Tiempo de convergencia a cero de la velocidad**
Rey	2.78 s
Reina	3.41 s
Alfil	4.03 s
Caballo	3.62 s
Torre	4.74 s
Peón	3.29 s
Caballo perpendicular	2.61 s

Tabla 4.4: Tiempo de convergencia a cero de la velocidad para cada pieza de ajedrez

Tiempo de detección: Es el tiempo en el que el algoritmo de detección determina la presencia de una pieza en las pinzas del brazo robótico y manda una señal al servomotor para detener su movimiento. Esta variable no se obtiene directamente del servomotor como las anteriores, pero se puede obtener de otras variables como la posición o la velocidad.

Las tablas obtenidas de la corriente (ver Tabla 4.2) y de la velocidad (ver Tabla 4.4), muestran tiempos de convergencia muy parecidos entre ellos, ambos valores en realidad son el tiempo de detección, pero obtenidos de diferentes variables. En la tabla 4.5 se hace una comparación entre los tiempos de convergencia tanto de la corriente como de la velocidad.

Pieza	Tiempo de convergencia de la corriente	Tiempo de convergencia a cero de la velocidad
Rey	2.70 s	2.78 s
Reina	3.41 s	3.41 s
Alfil	4.08 s	4.03 s
Caballo	3.62 s	3.62 s
Torre	4.73 s	4.74 s
Peón	3.33 s	3.29 s
Caballo perpendicular	2.59 s	2.61 s

Tabla 4.5: Comparación tiempos de convergencia

En la tabla 4.5 se puede observar que existe gran similitud entre ambos tiempos de convergencia, y que hay suficiente variación entre cada uno de los valores mostrados para cada pieza, por lo tanto el tiempo de detección es una variable a la que se le puede extraer mucha información para la clasificación de las piezas.

4.3. Selección de datos históricos y detección de estados estacionarios

En este paso se lleva a cabo la selección de las variables para el entrenamiento y la evaluación del modelo, unas vez seleccionadas las variables, se identifican y seleccionan los estados estacionarios de estas variables.

En el paso anterior se analizaron las variables disponibles para la implementación del sensor virtual, entre ellas destacan dos variables que proporcionan información valiosa para la clasificación de las piezas, la primera es la posición de detección y la segunda el tiempo de detección, por lo tanto ambas variables son seleccionadas como las variables de entrada del sensor virtual a implementar.

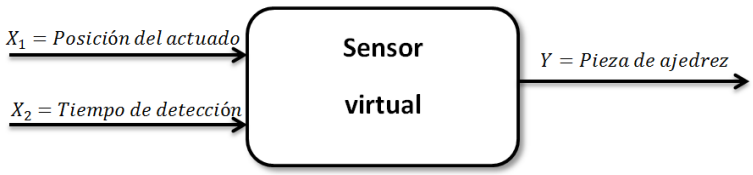


Figura 4.25: Variables de entrada del sensor virtual

Para el entrenamiento y la evaluación del modelo, se selecciona una muestra de 100 mediciones de posición de detección y 100 mediciones de tiempo de detección para cada pieza de ajedrez. Los datos fueron adquiridos de forma manual durante el proceso de sujeción de las piezas, para ello fue necesario sujetar una pieza de ajedrez de forma iterativa con las pinzas del brazo robótico e ir almacenando los datos en un archivo con extensión .txt, la pieza era cambiada de posición durante cada iteración para obtener resultados más reales y variados. En total se obtuvieron 700 mediciones de posición de detección y 700 mediciones para el tiempo de detección. De la figura 4.26 a la figura 4.32 se presentan las gráficas de posición de detección con una muestra de 100 mediciones para cada pieza de ajedrez.

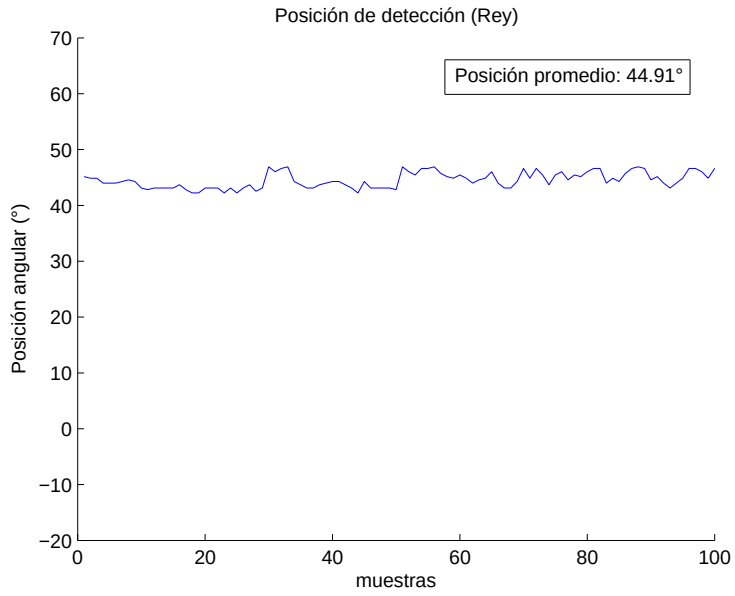


Figura 4.26: Gráfica de posición para rey

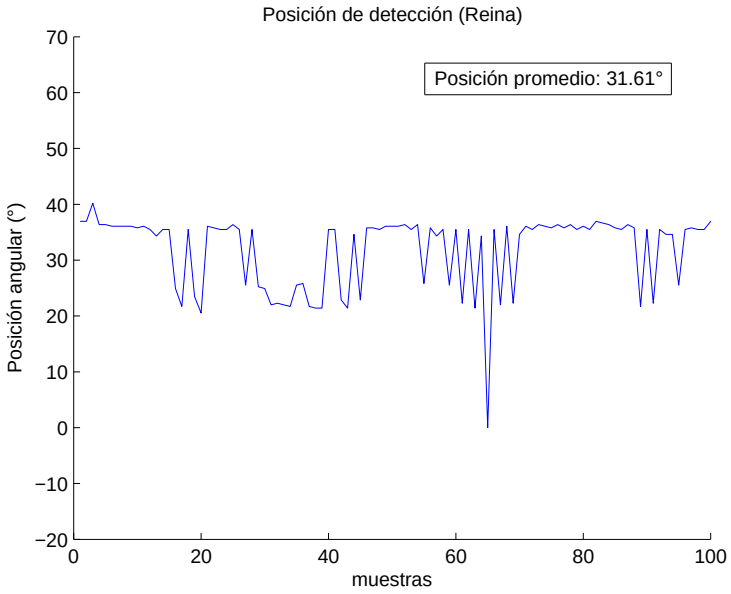


Figura 4.27: Gráfica de posición para reina

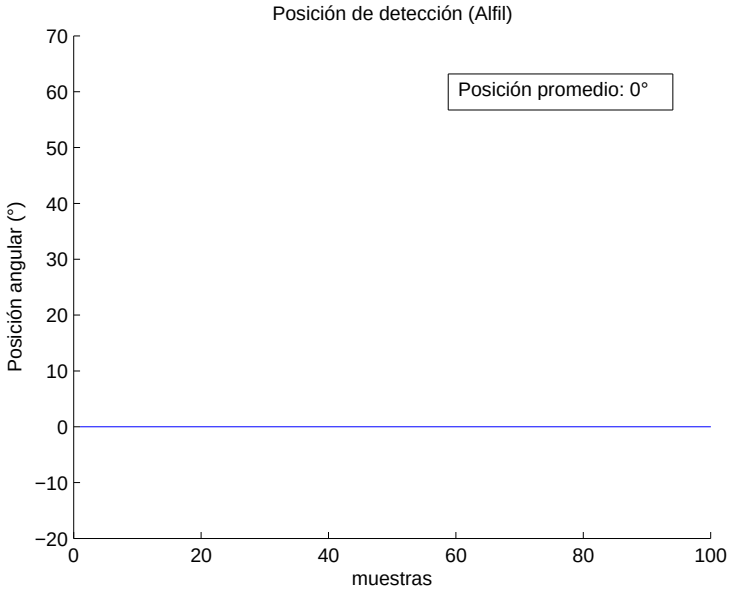


Figura 4.28: Gráfica de posición para alfil

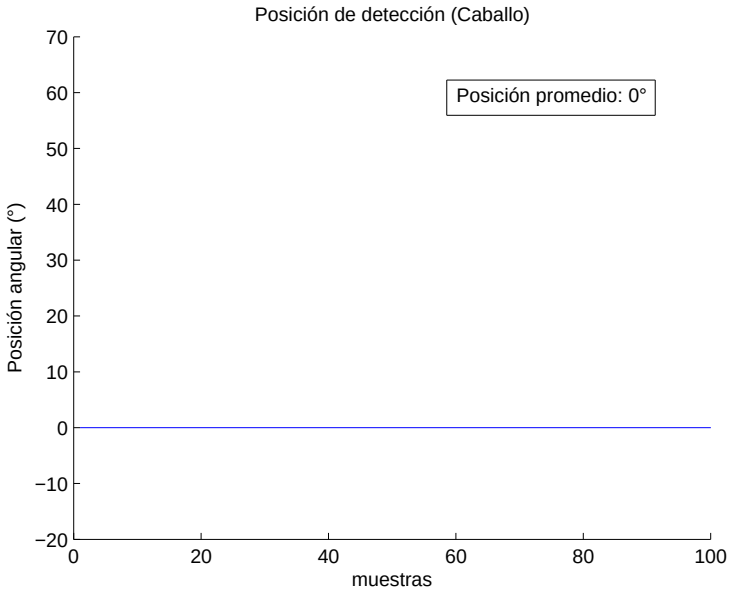


Figura 4.29: Gráfica de posición para caballo

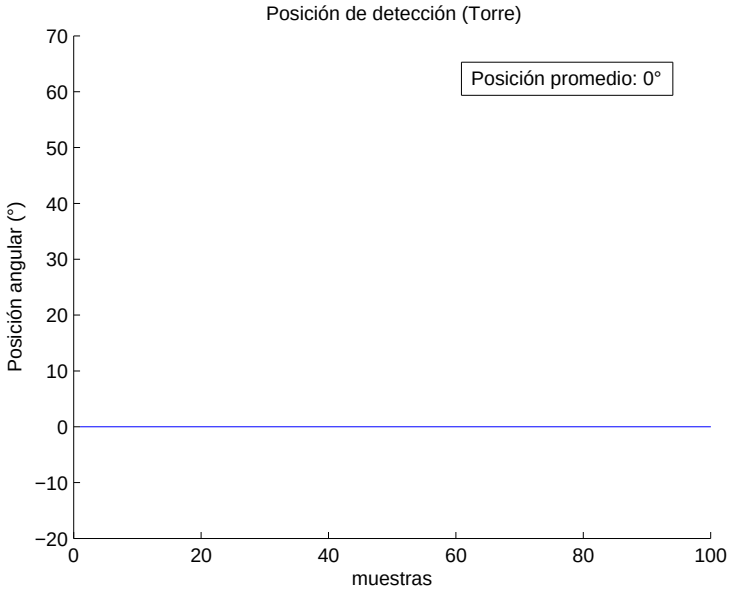


Figura 4.30: Gráfica de posición para torre

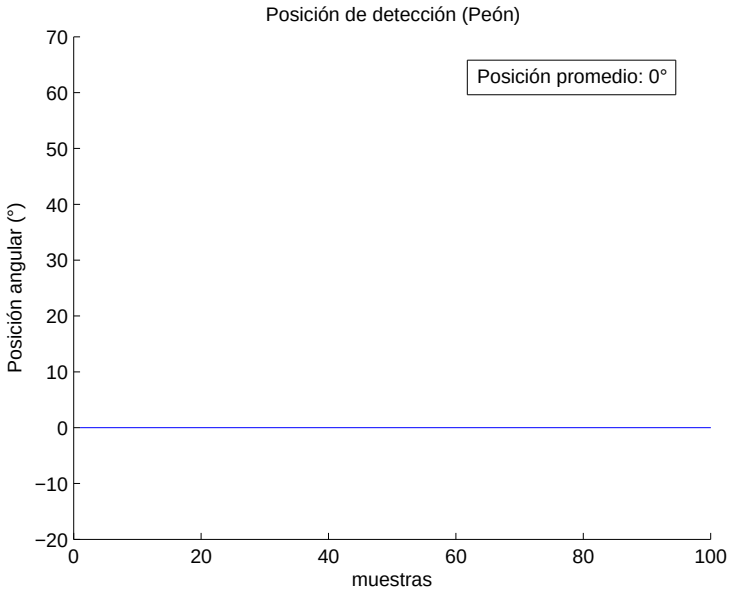


Figura 4.31: Gráfica de posición para peón

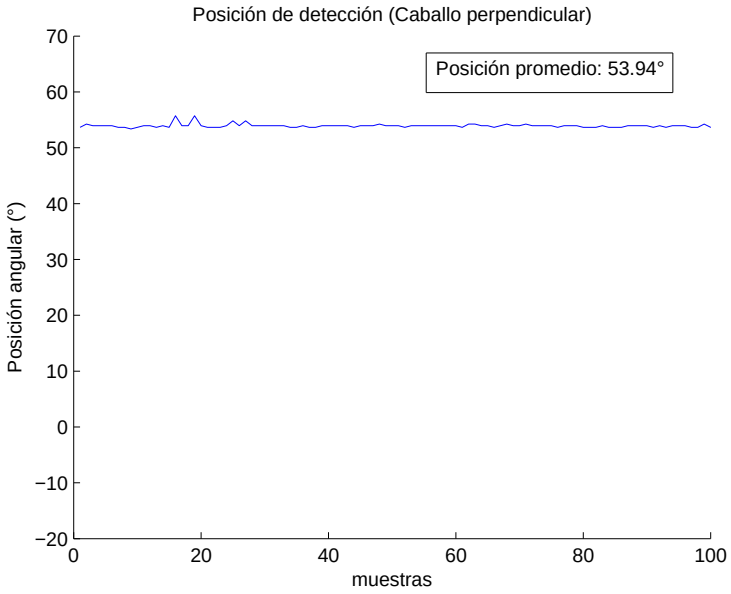


Figura 4.32: Gráfica de posición para caballo perpendicular

En la tabla 4.6 se muestra la información de las gráficas de posición de forma sintetizada, mostrando el valor mínimo y máximo de cada muestra, así como el valor promedio de las mediciones.

Pieza	**Valor mínimo**	**Valor máximo**	**Posición de detección promedio**
Rey	42.22°	46.92°	44.91°
Reina	0°	36.95°	31.58°
Alfil	0°	0°	0°
Caballo	0°	0°	0°
Torre	0°	0°	0°
Peón	0°	0°	0°
Caballo perpendicular	53.37°	55.71°	53.94°

Tabla 4.6: Resumen: posición de detección

De la figura 4.33 a la figura 4.39 se muestran las gráficas del tiempo de detección con una muestra de 100 mediciones para cada pieza.

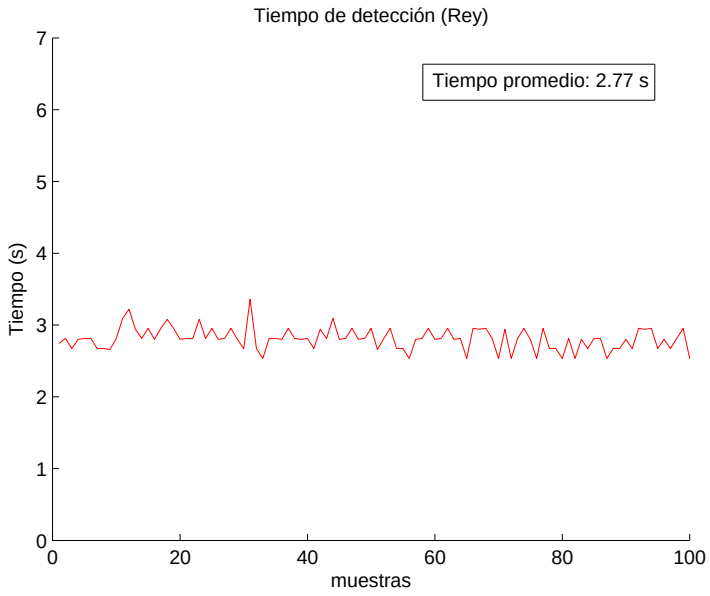


Figura 4.33: Gráfica del tiempo de detección para rey

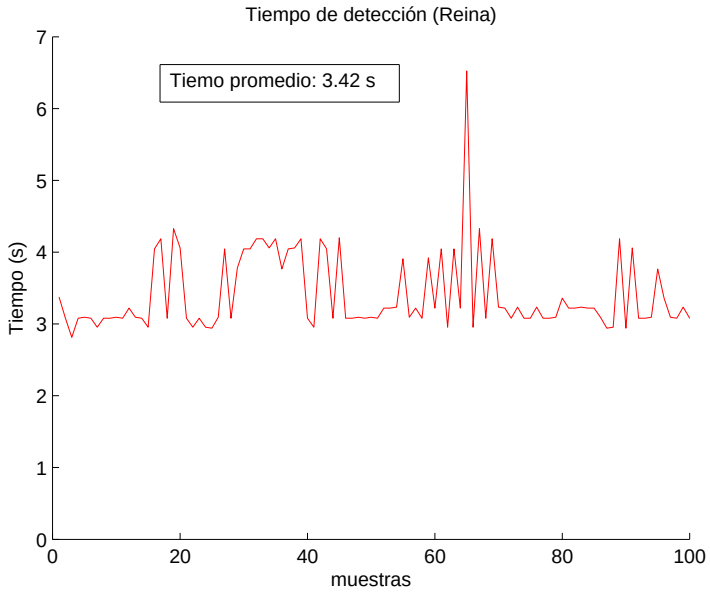


Figura 4.34: Gráfica del tiempo de detección para reina

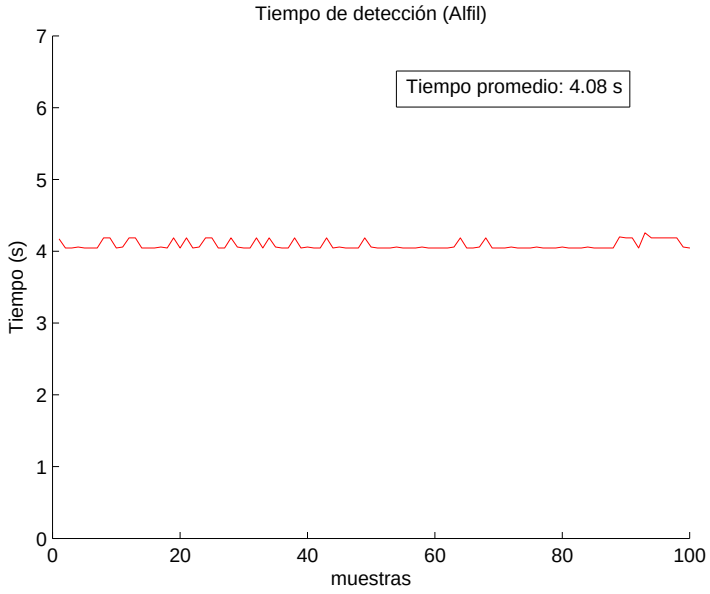


Figura 4.35: Gráfica del tiempo de detección para alfil

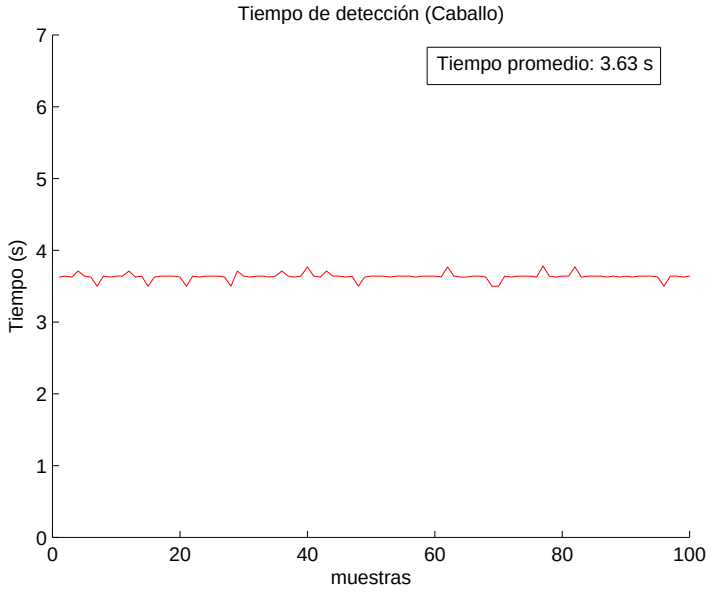


Figura 4.36: Gráfica del tiempo de detección para caballo

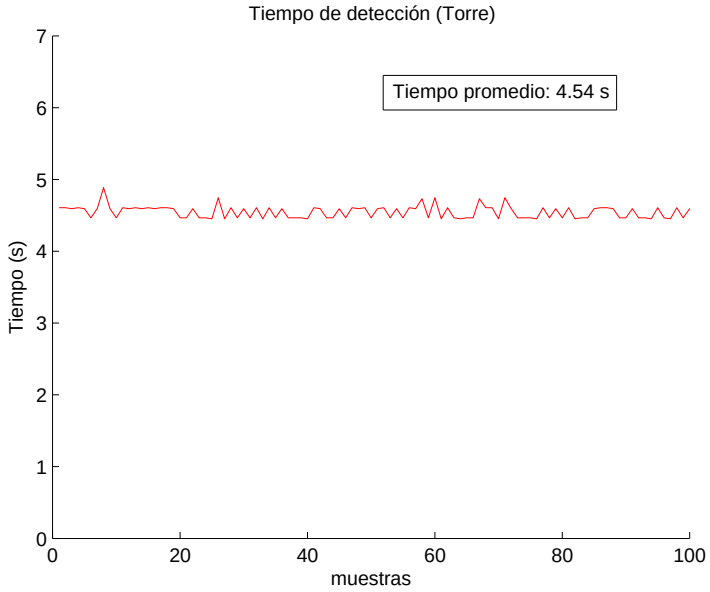


Figura 4.37: Gráfica del tiempo de detección para torre

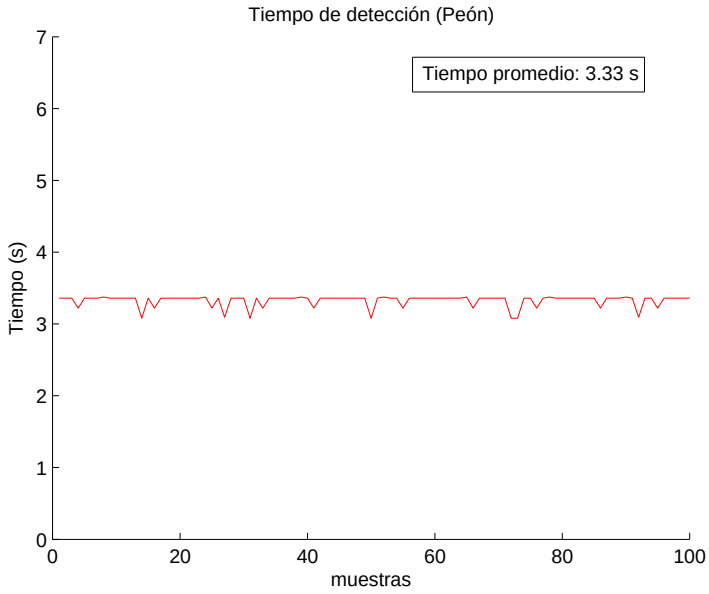


Figura 4.38: Gráfica del tiempo de detección para peón

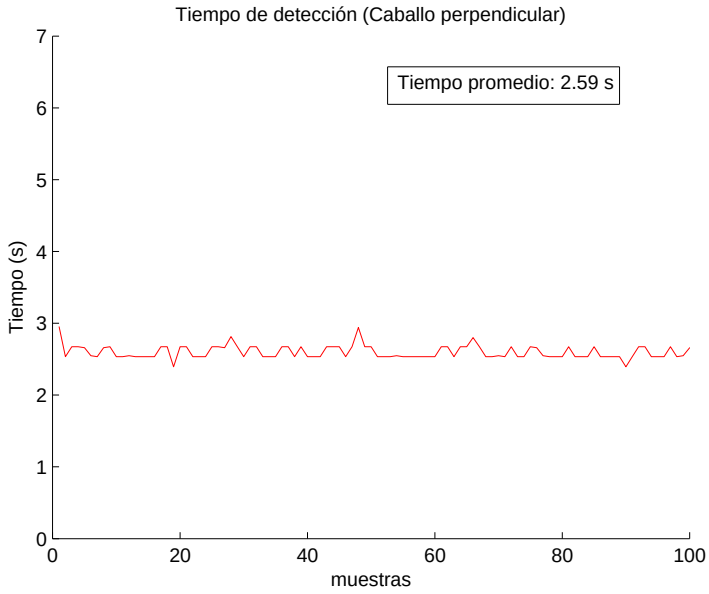


Figura 4.39: Gráfica del tiempo de detección para caballo perpendicular

En la tabla 4.7 se muestra la información de las gráficas de tiempo de forma sintetizada, mostrando el tiempo mínimo y máximo, así como el tiempo de detección promedio para cada pieza.

Pieza	**Valor mínimo**	**Valor máximo**	**Tiempo de detección promedio**
Rey	2.53 s	3.36s	2.77 s
Reina	2.94 s	6.52 s	3.42 s
Alfil	4.04 s	4.25 s	4.08 s
Caballo	3.50 s	3.78 s	3.63 s
Torre	4.45 s	4.88 s	4.54 s
Peón	3.08 s	3.37 s	3.33 s
Caballo perpendicular	2.39 s	2.95 s	2.59 s

Tabla 4.7: Resumen: tiempo de detección

En las tablas 4.6 y 4.7 se muestran los rangos de variación de los valores de posición y tiempo, así como los estados estacionarios (promedio) para cada pieza. Algunas mediciones tienen rangos más amplios que otros, por ejemplo, la reina (ver Figuras 4.27 y 4.34) tiene rangos muy amplios para la posición y tiempo, y esto se debe a su forma. La reina en particular presenta protuberancias en la parte superior, estas protuberancias o relieves corresponden a los picos de la corona de la reina, por lo tanto, la posición y el tiempo de detección dependerán de la orientación de la corona. La reina tiene un rango de valores muy amplio y puede presentar problemas tales como colinealidad, para esto es necesario realizar un tratamiento y acondicionamiento de datos antes del entrenamiento y validación del modelo.

4.4. Preprocesamiento de datos

En este paso se preparan los datos para que puedan ser procesados eficazmente por el modelo a elegir, para ello se analizan los datos que se tienen, buscando algunos de los problemas más usuales en datos, presentados en la sección Preprocesamiento de datos.

Primero se analizará si existen datos faltantes o valores atípicos en los datos. Con una inspección visual de las gráficas (Figuras 4.26 a 4.39) obtenidas en el paso anterior, se puede apreciar un valor atípico en la gráfica de posición de detección de la reina (ver Figura 4.40).

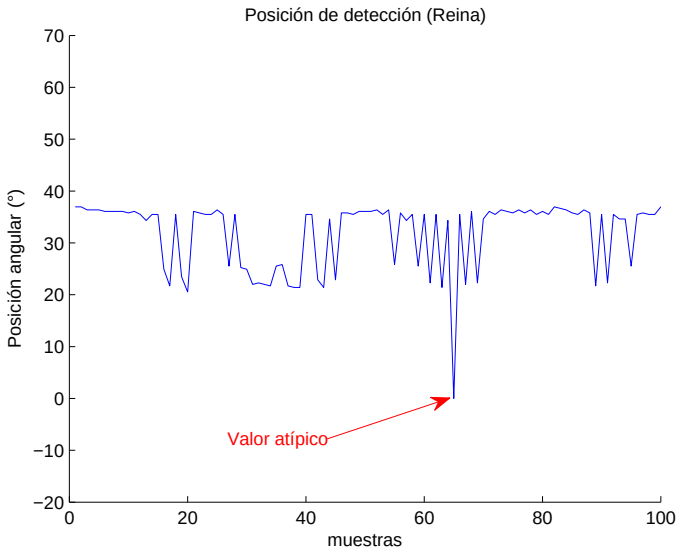


Figura 4.40: Valor atípico en la gráfica de posición de la reina

La forma más común de eliminar el valor atípico es sustituyéndolo por el valor de posición promedio de las mediciones, pero sin tomar en cuenta el valor atípico, para este caso el valor promedio de posición es de 31.83°. Al sustituir el valor atípico por dicho valor, la gráfica queda como se muestra en la figura 4.41.

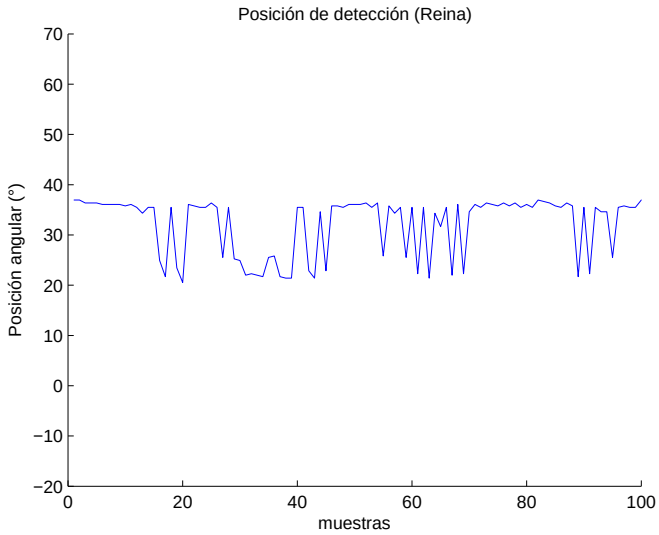


Figura 4.41: Eliminación de valor atípico en la gráfica de posición de la reina

Continuando con la inspección visual se puede encontrar el mismo problema en la gráfica de tiempo de detección de la reina (ver Figura 4.42).

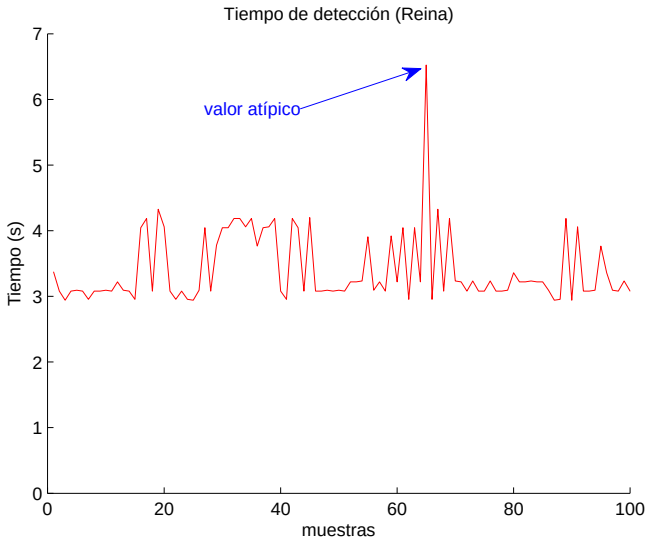


Figura 4.42: Valor atípico en la gráfica de tiempo de detección de la reina

A continuación se realiza el mismo paso de sustitución por el valor promedio sin tomar en cuenta el valor atípico, el cual en este caso es de 3.33 s. El resultado se muestra en 4.43.

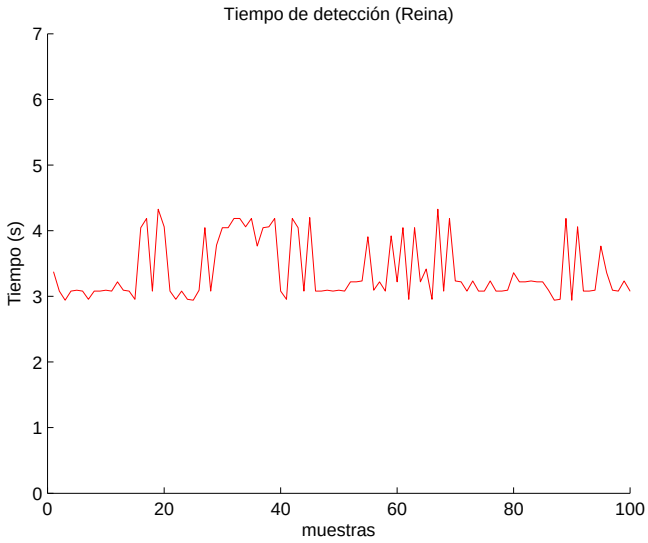


Figura 4.43: Eliminación del valor atípico en la gráfica de tiempo de detección de la reina

El tratamiento de los datos es esencial para el desarrollo del sensor virtual, debido a que cualquier valor atípico o valor faltante puede traer un efecto negativo en el rendimiento del modelo y por consecuencia provocar un mal funcionamiento del sensor virtual.

Después de realizar el tratamiento de los datos, los valores mínimos, máximos y promedio de la reina cambian tanto para la gráfica de posición como la de tiempo de detección. Los nuevos valores de los datos tras ser procesados, se muestran en las tablas 4.8 y 4.9, ya sin los datos atípicos.

Pieza	**Valor mínimo**	**Valor máximo**	**Posición de detección promedio**
Rey	42.22°	46.92°	44.91°
Reina	20.52°	36.95°	31.83°
Alfil	0°	0°	0°
Caballo	0°	0°	0°
Torre	0°	0°	0°
Peón	0°	0°	0°
Caballo perpendicular	53.37°	55.71°	53.94°

Tabla 4.8: Posición de detección

Pieza	**Valor mínimo**	**Valor máximo**	**Tiempo de detección promedio**
Rey	2.53 s	3.36s	2.77 s
Reina	2.94 s	4.32 s	3.40 s
Alfil	4.04 s	4.25 s	4.08 s
Caballo	3.50 s	3.78 s	3.63 s
Torre	4.45 s	4.88 s	4.54 s
Peón	3.08 s	3.37 s	3.33 s
Caballo perpendicular	2.39 s	2.95 s	2.59 s

Tabla 4.9: Tiempo de detección

Ahora se analizará la presencia de colinealidad en los datos. Si se observa la tabla 4.8, se aprecia que existe multicolinealidad en los datos del alfil, el caballo, la torre y el peón, las cuatro piezas presentan correlación en la posición de detección, por lo que se sugiere seleccionar un conjunto de variables de entrada diferente que sea menos colineal, tal y como se sugiere en el tratamiento de la colinealidad en la sección Preprocesamiento de datos, razón por la cual se ha elegido al tiempo de detección como variable de entrada al modelo. El tiempo de detección describe de una manera más amplia el comportamiento de las pinzas durante la

sujeción de todas las piezas (ver Tabla 4.9). La posición de detección describe solamente el comportamiento del rey, la reina y el caballo perpendicular (ver Tabla 4.8).

La colinealidad también se presenta con la reina, la cual describe rangos de valores muy amplios, valores que pueden presentar correlación con los datos de otras variables. Pero la correlación de los datos de la reina solo se presenta en el tiempo de detección, es decir el rango de los valores del tiempo de detección de la reina van de 2.94 s a 4.32 s, y el rango del tiempo de detección del alfil, el caballo y el peón, se encuentran dentro de estos valores (Ver Tabla 4.9) por lo que se dice que existe correlación entre los datos de la reina y los datos del alfil, el caballo y el peón en la variable de tiempo de detección. En la variable de posición no existe esta colinealidad entre la reina y las tres piezas mencionadas, la reina tiene un valor promedio de 31.83° y las demás piezas tienen un valor promedio de 0°, por lo tanto no existe correlación entre la reina y las tres piezas implicadas, de manera que se soluciona el problema de colinealidad nuevamente utilizando otra variable de entrada.

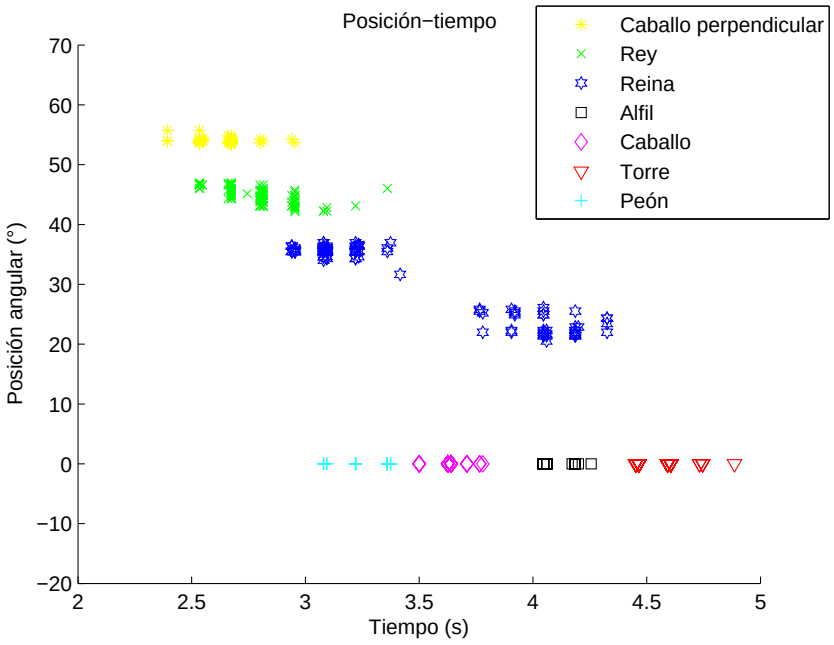


Figura 4.44: Gráfica de relación posición-tiempo

Para entender mejor cada variable de entrada y la relación que existe entre ellas, se presenta una gráfica que relaciona la posición de detección con el tiempo de detección para

cada pieza (ver Figura 4.44), donde se puede apreciar que cada entrada tiene un criterio de clasificación diferente para cada pieza. En la misma gráfica se puede observar que no existe colinealidad entre los datos, es decir ninguno de los datos se correlaciona entre sí.

Una vez que se tienen las variables de entrada (posición y tiempo) preprocesadas, se continúa con la selección, el entrenamiento y la validación del modelo.

4.5. Selección del modelo, entrenamiento y validación

La selección del modelo es un paso crítico para el desarrollo del sensor virtual, debido a que el modelo es el componente principal del sensor virtual, y por lo tanto, el buen funcionamiento del sensor dependerá de una buena selección del modelo.

4.5.1. Selección del modelo

Analizando la gráfica de relación entre las variables de entrada (ver Figura 4.44), se puede observar que cada una de las piezas tiene un espacio de rasgos muy distintivos, estos rasgos permiten que las piezas puedan ser asociadas a una clases o a un grupo de datos únicos. Debido a dichas características, se propone el uso de una red neuronal como modelo, ya que son las técnicas de aprendizaje más usadas para el reconocimiento de objetos o patrones. En específico se propone la implementación de un perceptrón multicapa (PMC), utilizado como clasificador de clases.

Un perceptrón multicapa puede formar regiones convexas en el espacio de datos, estas regiones convexas se forman mediante la intersección de líneas rectas trazadas a través de todo el espacio de datos.

En la figura 4.45 se muestra de forma gráfica que se pueden formar regiones convexas en el plano posición-tiempo, separando los datos en clases que pueden ser clasificadas.

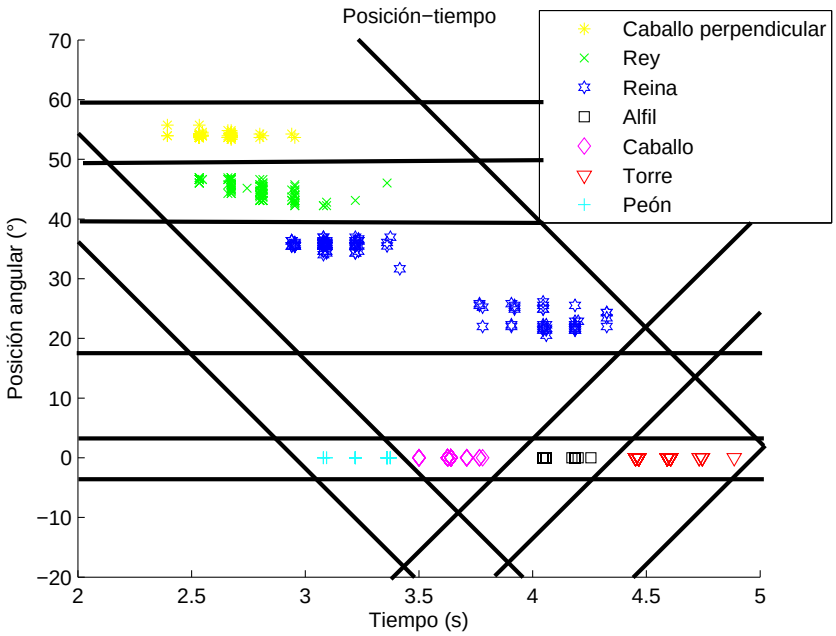


Figura 4.45: Regiones de los datos

Estructura del modelo

La red neuronal a implementar será una red para clasificación en multicategoría, debido a que se tienen siete clases para ser clasificadas (ver Figura 4.45). Típicamente se usa una neurona de salida para representar cada clase.

Para la clasificación en multicategoría existen dos tipos de arquitecturas, la "All-Class-in-One-Network" (ACON), esto es, todas las clases en una red y la "One-Class-in-One-Network" (OCON), esto es, una red para cada clase, y se utiliza cuando se tiene un gran número de clases.

Para la estructura del perceptrón multicapa a implementar se utiliza una configuración ACON, es decir que todas las clases estarán en una única red, debido a que se considera que siete clases no es un número alto de salidas. En una estructura ACON, la única red presente tiene que "satisfacer"todas estas clases, así que el número de neuronas en la capa oculta dependerá del número de clases a clasificar.

Con la gráfica de regiones (ver Figura 4.45) se puede determinar la estructura de la red neuronal a implementar. El número de elementos del vector entrada será igual al número

de neuronas que se utilizarán en la capa de entrada, en este caso la posición y el tiempo de detección son los elementos de entrada, por lo tanto la capa de entrada tendrá dos neuronas.

Como ya se ha estudiado en la sección Redes neuronales artificiales (RNA), en un perceptrón multicapa puede haber una o más capas ocultas entre las capas de entrada y salida. La mayoría de los problemas prácticos se resuelven con una sola capa oculta, se recomienda usar una sola capa oculta como primera elección y si el problema no se resuelve, entonces se debe usar una segunda capa oculta.

El número adecuado de neuronas en la capa oculta se determina a través de la experimentación. Muy pocas neuronas en la capa oculta impedirían el correcto mapeo de la entrada a la salida, y muchas neuronas en la capa oculta podrían causar memorización de los datos sin extraer las características para la generalización. Para calcular el número de neuronas en la capa oculta se utiliza la siguiente regla general:

$$h = \left(\frac{2}{3}\right) * (n + m),$$

donde h es el número de neuronas en la capa oculta, n es el número de entradas y m es el número de salidas. Por consecuente se tiene:

$$h = \left(\frac{2}{3}\right) * (2 + 7) = 6,$$

Se utilizarán seis neuronas en la capa oculta como primera instancia. Si no se logra entrenar adecuadamente la red con el número de neuronas propuesto, se aumentará el número de neuronas hasta obtener un entrenamiento óptimo.

Una vez definida la estructura de la red neuronal, se puede proceder con el entrenamiento. La configuración inicial propuesta de la red neuronal se muestra en la figura 4.46.

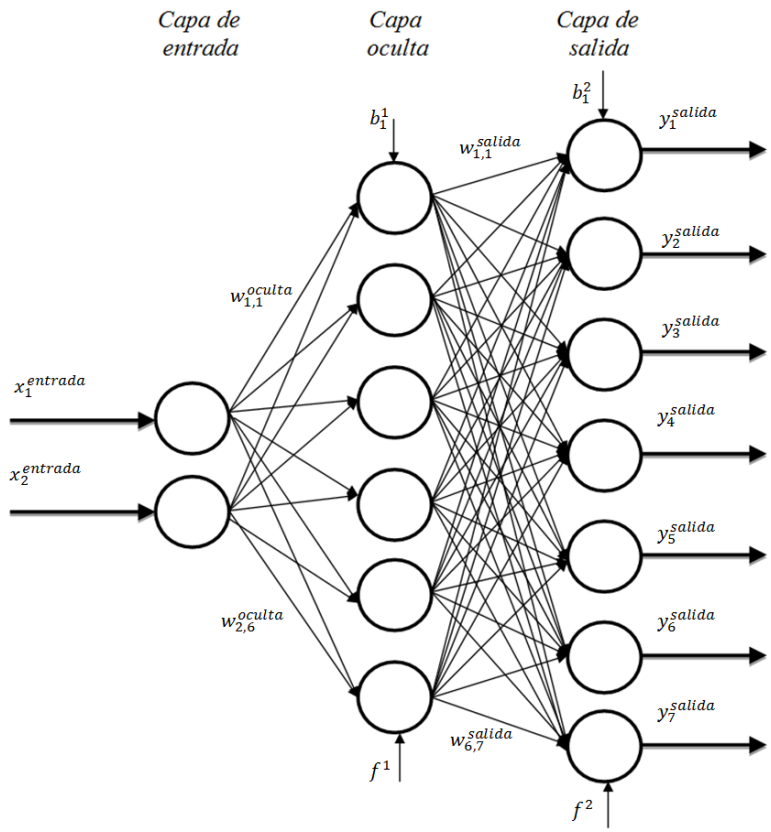


Figura 4.46: Estructura de la red neuronal: Perceptrón Multicapa con una capa oculta

4.5.2. Entrenamiento

Para el entrenamiento de la red neuronal se utiliza el *Neural Network Toolbox* de MATLAB como herramienta de desarrollo. La herramienta de MATLAB permite desarrollar y entrenar diferentes tipos de redes neuronales de una forma relativamente sencilla, solo se debe ingresar una estructura de red, los parámetros de entrenamiento y los datos para el entrenamiento, la validación y las pruebas.

Un PMC opera con vectores numéricos y normalizados, por lo tanto, todos los datos deben estar codificados como números y tener un rango de valores predeterminados. Para un PMC La normalización se plantea en rangos de [0,1] (variable de distribución normal) o [-1,1] (variable uniformemente distribuida). Tal elección debe basarse en un comportamiento natural, en función a los datos a tratar, por ejemplo, si se tienen solamente valores positivos puede seleccionarse tanto la normalización [0,1] como [-1,1], en este caso se selecciona esta

última por tener un rango de distribución de valores más amplio. Por lo tanto, los datos de entrada y salida tendrán un rango de valores de [-1,1]. La codificación de las piezas se realiza como se muestra en la tabla 4.10, donde la posición del "1" indica la salida de la clase correspondiente.

Pieza	**N° de clase**	**Valor codificado**
Rey	1	[1 -1 -1 -1 -1 -1 -1]
Reina	2	[-1 1 -1 -1 -1 -1 -1]
Alfil	3	[-1 -1 1 -1 -1 -1 -1]
Caballo	4	[-1 -1 -1 1 -1 -1 -1]
Torre	5	[-1 -1 -1 -1 1 -1 -1]
Peón	6	[-1 -1 -1 -1 -1 1 -1]
Caballo perpendicular	7	[-1 -1 -1 -1 -1 -1 1]

Tabla 4.10: Codificación de clases

Además de normalizados, los datos de la red neuronal deben ser ingresados de forma matricial para poder ser procesados por el Toolbox de MATLAB, por lo tanto se agrupan de la siguiente forma; los datos de entrada de la red neuronal se colocan en una matriz de 2x700, donde 2 representa el número de entradas y 700 son el total de muestras tomadas. Los pesos de la capa oculta se colocan en una matriz de 6x2, donde 6 representa el número de neuronas en la capa oculta y 2 el número de entradas. Los pesos de la capa de salida son colocados en un matriz de 7x6, donde 7 representa el número de salidas y 6 el número de neuronas en la capa oculta. Las bias de la capa oculta se colocan en una matriz de 6x1 y las bias de la capa de salida en una matriz de 7x1. La salida está dada por una matriz de 7x700, donde 7 son las salidas de la red neuronal, y 700 la salida correspondiente para cada muestra de entrada.

Matriz de entrada (2x700):

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,700} \\ x_{2,1} & x_{2,2} & \dots & x_{2,700} \end{bmatrix}$$

Matriz de pesos en capa oculta (6x2):

$$W^{oculta} = \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \\ w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \\ w_{1,5} & w_{2,5} \\ w_{1,6} & w_{2,6} \end{bmatrix}$$

Matriz de pesos en capa de salida (7x6):

$$W^{salida} = \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} & w_{4,1} & w_{5,1} & w_{6,1} \\ w_{1,2} & w_{2,2} & w_{3,2} & w_{4,2} & w_{5,2} & w_{6,2} \\ w_{1,3} & w_{2,3} & w_{3,3} & w_{4,3} & w_{5,3} & w_{6,3} \\ w_{1,4} & w_{2,4} & w_{3,4} & w_{4,4} & w_{5,4} & w_{6,4} \\ w_{1,5} & w_{2,5} & w_{3,5} & w_{4,5} & w_{5,5} & w_{6,5} \\ w_{1,6} & w_{2,6} & w_{3,6} & w_{4,6} & w_{5,6} & w_{6,6} \\ w_{1,7} & w_{2,7} & w_{3,7} & w_{4,7} & w_{5,7} & w_{6,7} \end{bmatrix}$$

Matriz de bias en capa oculta (6x1):

$$B^{oculta} = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_4^1 \\ b_5^1 \\ b_6^1 \end{bmatrix}$$

Matriz de bias en capa de salida (7x1):

$$B^{salida} = \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \\ b_4^2 \\ b_5^2 \\ b_6^2 \\ b_7^2 \end{bmatrix}$$

Matriz de salida (7x700):

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} & y_{1,5} & \cdots & y_{1,700} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} & y_{2,5} & \cdots & y_{2,700} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} & y_{3,5} & \cdots & y_{3,700} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} & y_{4,5} & \cdots & y_{4,700} \\ y_{5,1} & y_{5,2} & y_{5,3} & y_{5,4} & y_{5,5} & \cdots & y_{5,700} \\ y_{6,1} & y_{6,2} & y_{6,3} & y_{6,4} & y_{6,5} & \cdots & y_{6,700} \\ y_{7,1} & y_{7,2} & y_{7,3} & y_{7,4} & y_{7,5} & \cdots & y_{7,700} \end{bmatrix}$$

Los pesos y bias son inicializados de forma automática y aleatoria por el Toolbox de MATLAB, utilizando números aleatorios distribuidos uniformemente que mejor se adaptan a las entradas y salidas de la red neuronal.

Para la división de los datos de entrenamiento, validación y pruebas, se utiliza una distribución típica de los datos del 70 % para el entrenamiento, 15 % para la validación y 15 % para la prueba. A partir del empleo de la división de datos en los grupos mencionados, es posible aplicar una técnica para evitar el sobreaprendizaje: *early-stopping*. Durante el proceso iterativo de optimización de los parámetros de la red, se comparan los errores obtenidos con los datos de entrenamiento y con los datos de validación. En el caso de que durante sucesivas iteraciones, el error con los datos de entrenamientos disminuya, mientras que el error con los datos de validación aumente, se detiene el proceso de ajuste, como un criterio de parada adicional. El número sucesivo de incrementos del error de validación se establece en 6.

De los tres algoritmos de aprendizaje presentados en la sección Redes neuronales ar-

tificiales (RNA) para el entrenamiento de redes neuronales, se selecciona el algoritmo de Leven-Marquardt (LM), debido principalmente a su rapidez de convergencia y buen desempeño, además de explotar las ventajas de los otros dos métodos presentados (Gauss-Newton y Gradiente descendiente). El algoritmo Leven-Marquardt combina la velocidad del algoritmo Gauss-Newton con la estabilidad del gradiente descendiente lo que lo hace el algoritmo más robusto de los métodos presentados.

El criterio escogido para medir la eficiencia del proceso de entrenamiento es el del error cuadrático medio (ECM). El ECM es el resultado de dividir la sumatoria de los cuadrados de la diferencia de los valores predichos menos los valores observados, entre el número total de datos y son determinados después de que los pesos son ajustados. El ECM es el criterio más usado para medir el rendimiento de las redes neuronales durante el entrenamiento.

La función de activación que se usa tanto para la capa oculta como para la capa de salida es la sigmoidal tangente hiperbólica. La función *tansig* tiene rangos de valores de [-1,1] tal y como los datos de entrada y salida de la red neuronal a implementar, por lo tanto es la función que mejor se adapta a la particularidad de los datos.

Otro factor importante para el entrenamiento de la red neuronal, es el factor de aprendizaje, el cual determina la rapidez del algoritmo pero también su exactitud. Cuanto mayor sea, se necesitarán menos iteraciones pero es muy probable que no se llegue al mínimo error de evaluación. Cuanto menor sea, más lento será el aprendizaje, pero es más probable que se alcance el mínimo error propuesto. El factor de aprendizaje se establece en .001.

Las iteraciones del entrenamiento deben detenerse en algún momento. Por lo general se utilizan varios criterios distintos para definir las condiciones de paro. El proceso iterativo se detiene cuando alguna de estas condiciones se cumple. Existen diversos criterios posibles: cantidad de iteraciones, tiempo de ejecución, criterios de convergencia (que la función de error no se modifique en una magnitud mayor que un valor predeterminado entre iteraciones sucesivas, por ejemplo). Sin embargo el entrenamiento puede detenerse antes de tiempo o mucho después del momento adecuado, por ejemplo presentar sobre-entrenamiento, y es responsabilidad del diseñador saber diagnosticar estas situaciones y tomar las medidas correctivas indicadas. Es por ello que se implementan técnicas como el *early stopping* para evitar este tipo de inconvenientes. Los criterios de paro de la red neuronal a implementar son presentados en la tabla 4.11.

Criterio de paro	Valor
Número máximo de épocas	1000
Tiempo de entrenamiento	Infinito
Error mínimo a alcanzar	.01
Mínimo rendimiento del gradiente	1e-10
Número sucesivo del error de validación	6
Máximo factor de aprendizaje	1e10

Tabla 4.11: Criterios de paro

Una vez establecidos todos los parámetros de la red neuronal, solo es cuestión de ingresarlos en el editor de un entorno de desarrollo MATLAB en forma de código y ejecutar el programa para que la red neuronal comience a entrenarse hasta alcanzar el rendimiento deseado, abriéndose una ventana con la información de dicho proceso (ver Figura 4.47).

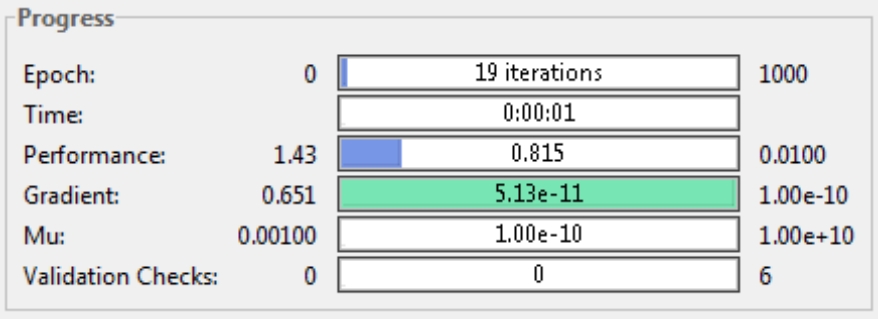


Figura 4.47: Progreso de entrenamiento con una capa oculta con 6 neuronas

En la ventana de progreso (ver Figura 4.47), se puede observar que el proceso se detuvo al alcanzar el mínimo rendimiento del gradiente ($1e-10$), pero el error deseado (.01) no se alcanzó, por lo que el desempeño de la red neuronal no será de lo más preciso, por lo tanto, se propone realizar el mismo procedimiento pero aumentando el número de neuronas en la capa oculta a 10 y observar la tabla de progreso.

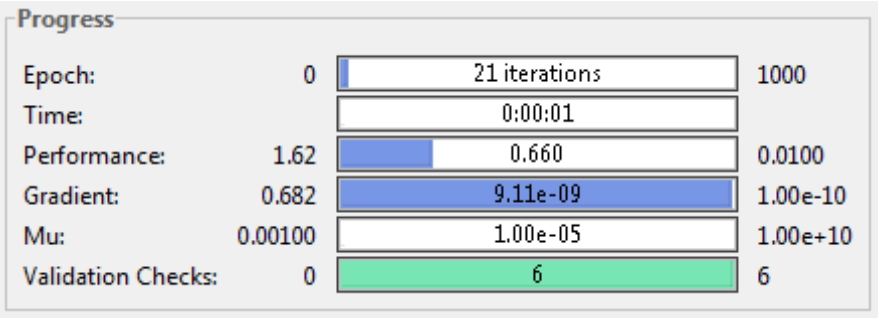


Figura 4.48: Progreso de entrenamiento con una capa oculta de 10 neuronas

Al aumentar el número de neuronas a 10 en la capa oculta, se observa (ver Figura 4.48) que hubo un sobre-entrenamiento de la red, el proceso se interrumpió al alcanzar seis incrementos consecutivos en la evaluación del error de validación, y el error de rendimiento no se redujo lo suficiente. Como ya se mencionó anteriormente en la sección Selección del modelo, seguir aumentando el número de neuronas en la capa oculta podría causar memorización de los datos, es decir que la red no sería capaz de evaluar datos que no se le han presentado anteriormente. Debido a que una capa oculta no logra satisfacer los requisitos de aprendizaje, se sugiere aumentar el número de capas ocultas a dos, con seis neuronas en cada una de ellas y utilizar la función *tansig* para la segunda capa oculta.

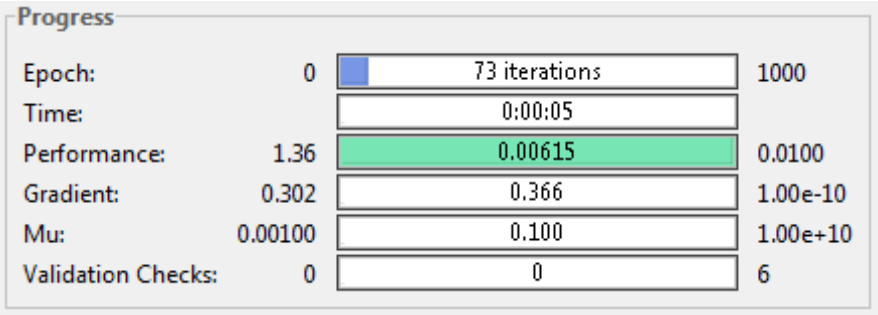


Figura 4.49: Progreso de entrenamiento con dos capas ocultas con 6 neuronas en cada capa

Al utilizar dos capas ocultas el proceso fue un poco más lento, pero los resultados son satisfactorios, con 73 iteraciones el algoritmo de aprendizaje logró llegar al error mínimo propuesto. En la figura 4.49 se puede observar que el proceso de aprendizaje se detuvo cuando el error de rendimiento alcanzó el valor de .006, valor suficiente para determinar que la RNA

se ha entrenado satisfactoriamente, ahora solo es cuestión de probar la red con valores reales y evaluar su comportamiento.

4.5.3. Evaluación

En la gráfica de rendimiento (ver Figura 4.50) se puede observar el comportamiento del error de evaluación (Error cuadrático medio) durante cada iteración del proceso de aprendizaje, se aprecia claramente como disminuye hasta alcanzar e incluso pasar el mínimo error de rendimiento preestablecido (.01).

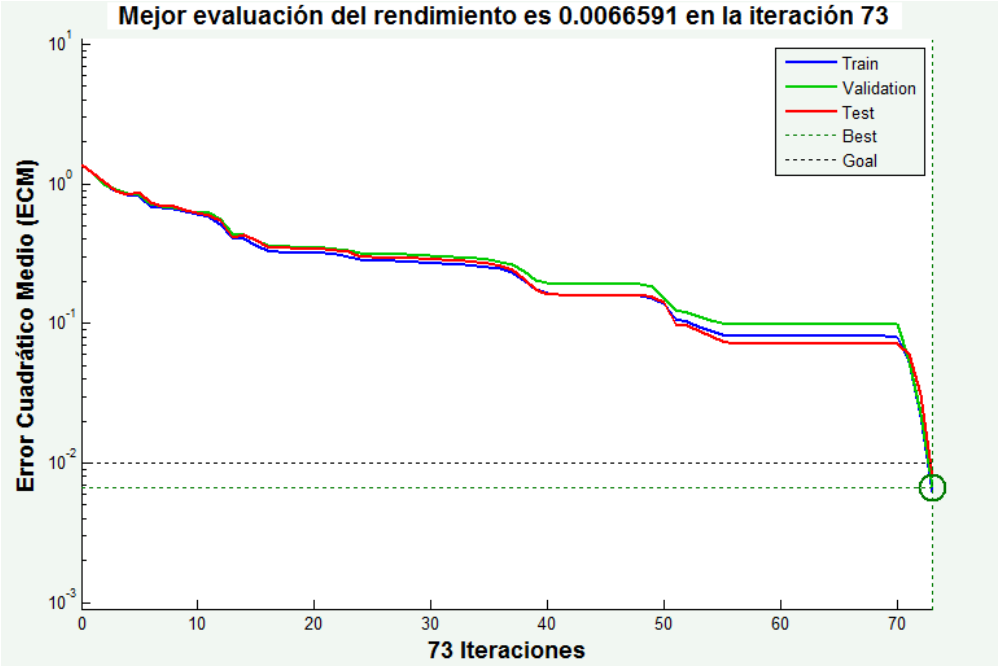


Figura 4.50: Rendimiento de entrenamiento

Las curvas de aprendizaje mostradas en la figura 4.50 muestran la posibilidad de un considerable efecto de sobreajuste (o sobreaprendizaje) si no se detiene el entrenamiento de la red en el momento oportuno. Este es cuando la curva verde (rendimiento del modelo medido sobre el conjunto de validación) alcanza el error mínimo (marcado por la línea negra horizontal). En este caso el entrenamiento fue detenido exactamente después de que se sobrepaso el error mínimo establecido (.01) y con esto se evitó el sobreaprendizaje de la red neuronal.

La forma estándar de evaluar un clasificador es mediante su matriz de confusión, esto es,

una matriz M de dimensión $C \times C$ para un problema de C clases y tal que m_{ij} dé el valor del número de patrones de la clase i que el clasificador asigna a la clase j. En este caso la Matriz M es una matriz de 7×7 donde cada fila representa la clase de salida y las columnas la clase real (ver Figura 4.51).

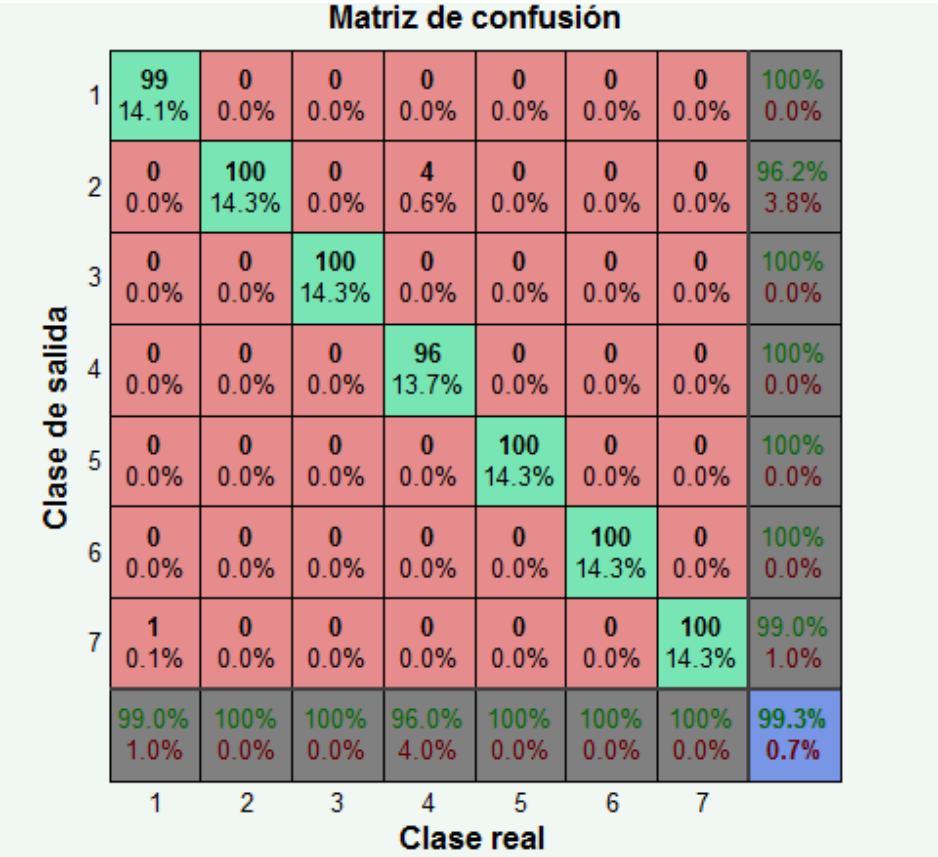


Figura 4.51: Matriz de confusión

Analizando la matriz de confusión, se obtienen los resultados de fiabilidad total del la RNA que fueron del 99.3 %. Un valor aceptable para el rendimiento de una red neuronal, considerando que siete clases tienen que ser clasificadas, en la matriz de confusión (ver Figura 4.51) se aprecia que en una ocasión, la red confundió una pieza de clase 1 (Rey) con una pieza de clase 7 (Caballo perpendicular) y en cuatro ocasiones confundió una pieza de clase 4 (Caballo) con una pieza de clase 2 (Reina), sin embargo el porcentaje de clasificaciones incorrectas fue de solo .70 %.

Para la evaluación de la red neuronal, se aplica un patrón real a la entrada y se ve en la

salida la clase en la que se ha clasificado. Para la primera prueba, se ingresa un vector de entrada $[44.07; 2.75]$ que corresponde a una posición angular de $44.07°$ con un tiempo de 2.75 segundos valores que se encuentran dentro de la clase 1 que es el rey (ver Tablas 4.8 y 4.9).

```
>> yout = sim(net,[44.07;2.75])

yout =

    0.9019
   -1.0000
   -0.9997
   -1.0000
   -1.0000
   -0.9905
   -0.9997
```

En este caso, de las siete salidas todas tienen un valor muy cercano a menos uno, excepto la salida correspondiente a la clase uno, que tiene un valor muy cercano a uno, siendo esta la clase reconocida.

Ahora se prueba la red neuronal con un vector de entrada $[54.08; 2.75]$ que corresponde a una posición angular de $54.08°$ con un tiempo de 2.75 segundos valores que se encuentran dentro de la clase 7 que es el caballo perpendicular (ver Tablas 4.8 y 4.9).

```
>> yout = sim(net,[54.08;2.75])

yout =

   -1.0000
   -1.0000
   -0.9999
   -1.0000
   -1.0000
   -0.9428
    1.0000
```

En este caso la salida correspondiente a la clase siete fue la clase reconocida. Ambas clasificaciones fueron correctas pero solo se usaron rangos que están dentro de los valores conocidos de la red neuronal. En el siguiente ejemplo se evalúa la extrapolación de la red neuronal, es decir que tan bien reacciona con datos que no fueron entregados para su entrenamiento. El vector propuesto es [48.08 ; 2.27], una posición angular de 48.08° que no se encuentra dentro de ninguna clase y un tiempo de 2.27 segundos que tampoco está dentro de ninguna clase.

```
>> yout = sim(net,[48.08;2.27])

yout =

    0.8966
   -1.0000
   -0.9997
   -1.0000
   -1.0000
   -0.9865
   -0.9995
```

A pesar de que los valores dados a la red neuronal no se encontraban dentro del rango de ninguna clase, la red neuronal aproximó la salida a la clase más cercana y esta es la clase 1, clase del rey que tiene un rango de 42.22° a 46.92° para la posición angular y 2.53 s a 3.36 s para el tiempo.

La extrapolación consiste en el proceso de estimar más allá del intervalo de observación original, el valor de la variable en base a su relación con otra variable. El perceptrón multicapa se caracteriza por extrapolar bien las salidas si se realiza un buen aprendizaje. Y en este ejemplo se pudo comprobar que dicha propiedad es bien soportada por la RNA desarrollada.

El código final de la implementación de la red neuronal en MATLAB se puede ver en el Apéndice 4. Código para la implementación de la red neuronal en MATLAB y la estructura final de la red neuronal implementada se muestra en la figura 4.52. Los pesos y bias finales se muestran en el Apéndice 5. Pesos y Bias finales de la red neuronal.

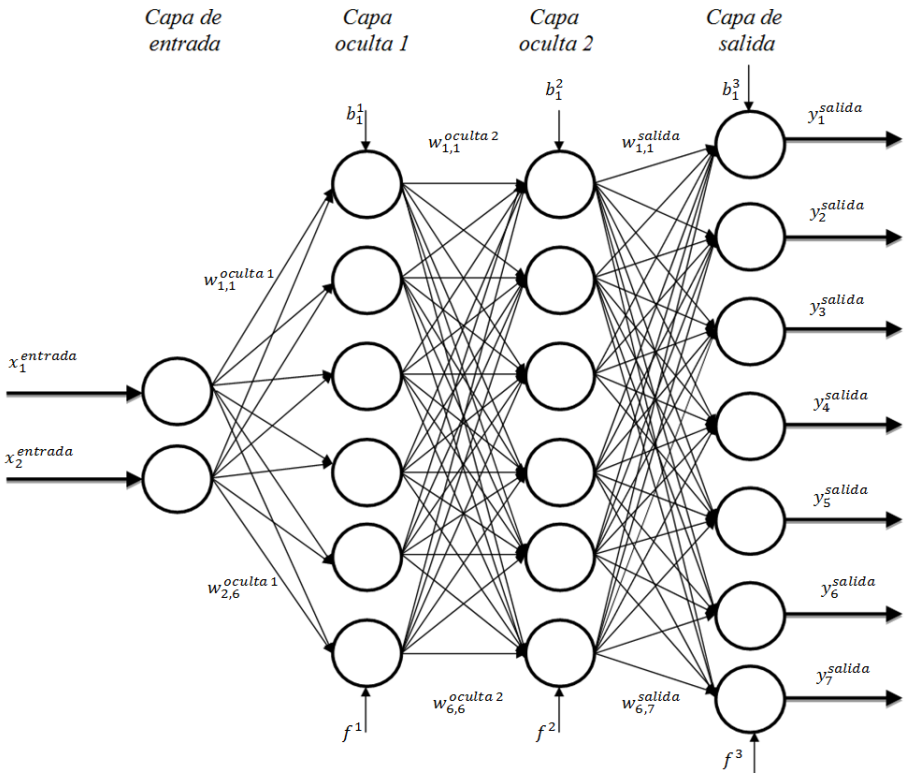


Figura 4.52: Estructura de la red neuronal: Perceptrón Multicapa con dos capas ocultas

Después de haber encontrado la estructura optima del modelo y verificar su buen funcionamiento, solo es cuestión de implementar el modelo junto al brazo robótico para que este último proporcione los datos de entrada a la red neuronal en tiempo real y se pueda hablar de un sensor virtual completo. Para dicha tarea se utiliza el *Instrument Control Toolbox* de MATLAB para controlar puertos seriales y poder comunicar el brazo robótico con la PC.

4.6. Mantenimiento del sensor virtual

Tal y como ocurre con los sensores físicos, el desajuste de medición es un factor que también afecta a los sensores virtuales, razón por la cual, es necesario realizar un mantenimiento periódico a los sensores y verificar que sus mediciones se encuentren dentro de un rango de valores convencionales. El sensor virtual desarrollado en este trabajo no está exento de este tipo de desajustes, las causas del desajuste en el sensor virtual táctil desarrollado pueden deberse a diferentes factores, entre ellos se encuentra el desplazamiento de los datos. Los datos pueden

desplazarse debido al desgaste de las piezas que intervienen en el proceso de sujeción, o por ejemplo a la lubricación de las pinzas de agarre, estos factores pueden modificar la velocidad de sujeción de las pinzas, y por consecuente se modifica el tiempo de detección de las piezas de ajedrez. Dichos factores son difíciles de controlar, pero se pueden implementar técnicas de autoajuste o adaptación que permiten el buen funcionamiento del sensor. Sin embargo como se menciona en Mantenimiento del Sensor Virtual, depende del juicio y conocimiento del diseñador implementar un correcto método de adaptación.

Se recomienda la realización de un mantenimiento periódico para evitar posibles desajustes del sistema, desafortunadamente este mantenimiento como ya se ha mencionado, necesita del conocimiento del proceso para llevarse a cabo correctamente. En el caso del sensor virtual desarrollado en este trabajo se necesita de una supervisión visual para determinar si se está llevando a cabo una correcta clasificación de las piezas.

Capítulo 5

Pruebas

En esta sección se detallan las pruebas realizadas al sensor virtual, así como los resultados obtenidos en cada una de ellas. Las pruebas consisten en introducir valores reales al sensor virtual y verificar su correcto funcionamiento.

Para realizar las pruebas se conecta el brazo robótico a Matlab por medio de comunicación serial, se inicia el algoritmo de sujeción y se obtienen las entradas (tiempo de detección y posición de detección), las cuales son procesadas inmediatamente por el sensor virtual y se estima la variable de salida. A continuación las pruebas realizadas para cada pieza.

Para el rey se obtuvieron las siguientes entradas con sus salidas correspondientes:

Rey:

Xin =	Xin =
45.0704	44.4839
2.6880	2.6740
yout =	**yout =**
0.9897	1.0000
−1.0106	−1.0111
−0.9902	−0.9850
−1.0163	−0.9850
−1.0017	−1.0196
−0.9985	−0.9973
−0.9765	−0.9889

Como se puede apreciar en yout, todas las salidas tienen un valor cercano a -1 excepto la primera línea que tiene un valor cercano a 1 que corresponde a la salida del rey, por lo tanto se ha realizado una correcta clasificación.

Ahora se sujeta la reina y se obtienen las siguientes entradas: y salidas del sensor virtual:

Reina:

```
Xin =                    Xin =

   22.2874                  21.9941
    3.5700                   3.5840

yout =                   yout =

   -1.0026                  -1.0026
    0.9986                   0.9984
   -0.9940                  -0.9940
   -1.0129                  -1.0130
   -0.9988                  -0.9988
   -0.9904                  -0.9901
   -1.0000                  -1.0000
```

Como se puede apreciar en las salidas, nuevamente se ha realizado una correcta clasificación, debido a que la línea 2 de las salidas es la que tiene un valor cercano a 1 y que corresponde a la reina.

Las entradas y salidas al sujetar el alfil son las siguientes:

Alfil:

```
Xin =                    Xin =

         0                        0
    4.1816                   4.2140
```

```
yout =

   −1.1181
   −0.9588
    0.9999
   −1.0805
   −0.9548
   −0.9689
   −0.9216
```

```
yout =

   −1.1443
   −0.9617
    0.8769
   −1.0367
   −0.8764
   −0.9635
   −0.8969
```

Como se aprecia en las salidas, ahora la tercera fila es la que tiene valores que se acercan más al valor de 1 y la tercer fila es la salida que representa al alfil, por lo tanto se realizó una clasificación correcta.

Las entradas y salidas obtenidas para el caballo se muestran a continuación:

Caballo:

```
Xin =

         0
    3.7660
```

```
Xin =

         0
    3.6540
```

```
yout =

   −1.0298
   −0.7354
   −0.4596
    0.3904
   −1.0353
   −1.1256
   −0.9849
```

```
yout =

   −1.0158
   −0.9753
   −0.8447
    0.9085
   −0.9870
   −1.0806
   −0.9650
```

La cuarta fila es la salida correspondiente al caballo y es la que tiene valores cercanos a 1, por lo tanto, se ha realizado una clasificación correcta.

Para la torre se obtuvieron las siguientes entradas y salidas:

Torre:

```
Xin =

     0
4.7040
```

```
Xin =

     0
4.6480
```

```
yout =

-1.0217
-1.0402
-0.8806
-1.0371
 0.9405
-1.0016
-0.9697
```

```
yout =

-1.0158
-1.0231
-0.8715
-1.0344
 0.9128
-1.0016
-0.9767
```

La quinta fila es la correspondiente a la torre y se observa que es la fila que tiene valores cercanos a 1. Entonces, se ha realizado una clasificación correcta.

Para el peón se tienen las siguientes entradas y salidas:

Peón:

```
Xin =

     0
3.2340
```

```
Xin =

     0
3.3600
```

```
yout =

-0.9992
-0.9856
-1.0022
-1.1476
```

```
yout =

-1.0241
-0.9883
-1.0054
-0.9358
```

−1.0029	−1.0021
1.1334	0.9285
−1.0054	−0.9807

La fila del peón es la sexta, y se aprecia en las salidas que efectivamente en la sexta fila se encuentran los valores más cercanos a 1, por lo tanto se llevo a cabo una clasificación correcta.

Por último se obtuvieron las siguientes entradas y salidas al sujetar un caballo perpendicularmente:

Caballo perpendicular:

Xin =	Xin =
53.5792	54.1657
2.6040	2.4360

yout =	yout =
−0.9397	−0.9919
−0.9860	−0.9872
−1.0066	−1.0074
−0.9968	−0.9981
−1.0123	−1.0108
−1.0019	−0.9992
0.9523	1.0038

La última fila corresponde a un caballo perpendicular, y es la fila que tiene los valores cercanos a uno, por tanto el caballo perpendicular ha sido correctamente clasificado.

Las pruebas realizadas dan por sentado que el sensor virtual funciona de manera correcta en tiempo real, y que está listo para detectar y clasificar las 6 piezas que componen un tablero de ajedrez.

Capítulo 6

Validación

En este capítulo se valida el sensor virtual desarrollado en esta tesis, utilizando un método alternativo para el desarrollo de sensores virtuales basados en datos. El método que se utiliza es el agrupamiento difuso y en específico el algoritmo *Fuzzy C-Means*, el cual se caracteriza por dar grados de pertenencia a cada dato con respecto a cada grupo, lo cual es muy útil cuando los datos están traslapados.

En la figura 6.1 se puede ver que los datos pueden ser agrupados en siete diferentes clusters, los cuales corresponden a cada pieza del ajedrez.

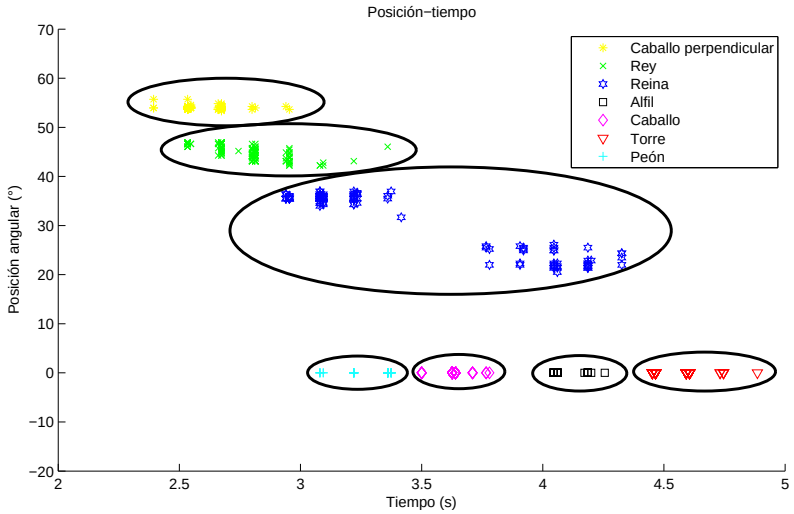


Figura 6.1: Agrupamiento difuso de los datos

En Técnicas de desarrollo para sensores virtuales basados en datos, se da una introducción

del agrupamiento difuso y se describe el algoritmo *Fuzzy C-Means* de manera detallada.

Para la implementación del algoritmo se ha utilizado Matlab, herramienta con la cual se programa el algoritmo y se presentan los resultados de forma gráfica. Existen dos formas en Matlab para resolver el algoritmo, usando la línea de comandos y/o usando una interfaz gráfica. En este caso se usará el primer método.

Para encontrar los centros de los clusters, se programa la función "$fcm1$", la cual es descrita a continuación:

$[center, U, obj_fcm] = fcm1(data, cluster_n)$

Los argumentos de esta función son:

1. *data* - Todos los datos a ser agrupados.
2. $cluster_n$ - Número de clusters (más de uno).
3. *center* - La matriz de centros.
4. U - La matriz resultante.
5. obj_fcn - El valor de la función objetivo para cada iteración.

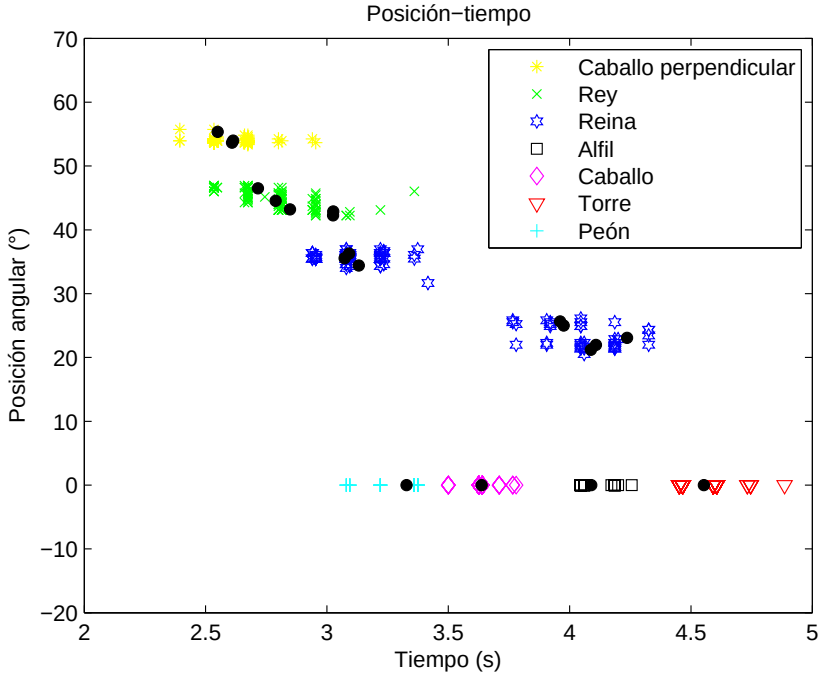


Figura 6.2: Agrupamiento difuso de los datos

Una desventaja ya antes mencionada del algoritmo *Fuzzy C-Means*, es que se debe predefinir un número de clusters y el resultado de una buena clasificación de los datos depende mucho del número de clusters que se ingresan al algoritmo. Por conveniencia y después de tratar con diferentes valores, se llega a la conclusión que 20 es el número de clusters que mejor resultados da al algoritmo. Para el criterio de paro, se utiliza una β de 0.001.

El código en Matlab se muestra en Apéndice 7. Código para la implementación del algoritmo Fuzzy C-Means en MATLAB, y los resultados se muestran en la figura 6.2.

En la figura 6.2 se puede observar que se han creado varios clusters, los cuales definen de forma correcta cada clase, algunas clases tienen más de un cluster, esto debido al número de clusters con relación al número de clases y a la distribución de los datos, pero con esto se le da una mayor robustez al sistema.

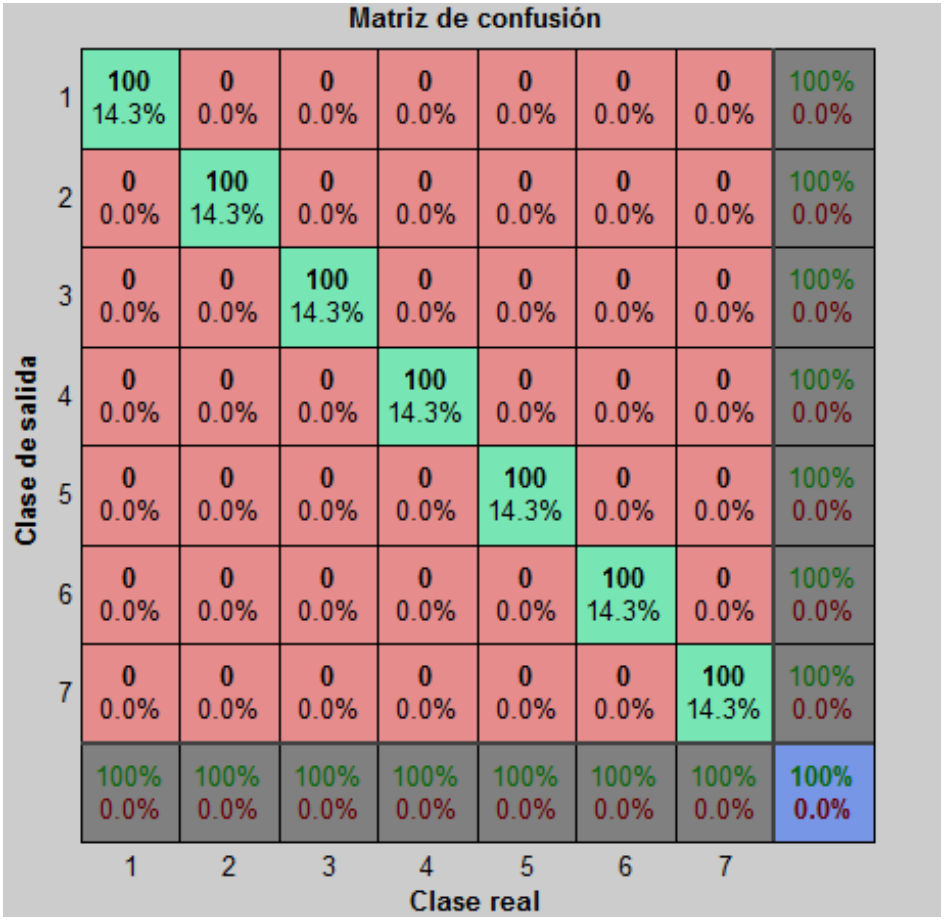


Figura 6.3: Matriz de confusión para el algoritmo Fuzzy C-Means

Por último se evalúan el total de los datos con cada uno de los centros obtenidos, para ello se utiliza una matriz de confusión donde se compara la clase real con la clase obtenida.

Analizando la matriz de confusión (ver Figura 6.3), se obtienen los resultados de fiabilidad total del algoritmo Fuzzy C-Means que fueron del 100 %. Un valor mayor al obtenido con respecto al rendimiento de la red neuronal y un tiempo de computo menor al de la red neuronal (el tiempo de computo del algoritmo Fuzzy C-Means fue de 2.6 segundos, mientras que el tiempo de computo del perceptrón multicapa fue de 5 segundos). Estos resultados muestran que los centros fueron distribuidos de manera correcta, para poder agrupar cada una de las clases a la que pertenecen. Es importante enfatizar el hecho de que el número de centros seleccionados para llevar a cabo el algoritmo es un factor importante para el desempeño del mismo.

	Ventajas	**Desventajas**
Fuzzy C-means	· Método no supervisado (las entradas se relacionan a una salida sin un conocimiento a priori). · Tiempo de procesamiento (en pocas iteraciones se obtiene una buena aproximación. · Buen clasificador cuando hay datos muy similares (da valores de pertenencia a los datos).	· Si los clúster están muy juntos se requerirá de un mayor número de centros.
Perceptron Multicapa	· Extrapolan de una manera eficiente (cuando se ingresa un dato que no fue considerado en la etapa de entrenamiento, se puede clasificar de forma correcta). · Tolerancia a fallos (las redes pueden reconocer patrones con ruido, distorsionados o incompletos)	· Mayor tiempo de procesamiento. · falta de reglas definitorias que ayuden a construir una red para un problema dado.

Tabla 6.1: Fuzzy C-means vs Perceptron Multicapa

En la tabla 6.1 se muestran las ventajas y desventajas de ambos métodos, tanto del Fuzzy C-Means como del perceptrón multicapa.

Con esto se concluye que se ha llevado a cabo una clasificación adecuada de los datos con los parámetros preestablecidos. Al mismo tiempo se pueden comparar los resultados con los obtenidos por el método de redes neuronales artificiales, donde ambos métodos clasifican las clases de manera correcta, pero un método utilizando un aprendizaje supervisado (Perceptrón Multicapa) y el otro un aprendizaje no supervisado (*Fuzzy C-Means*). Ambos métodos pueden tener sus ventajas y desventajas, mientras que el aprendizaje supervisado asume que sabemos qué se quiere predecir (se dice que los datos están etiquetados), mientras que el no supervisado segmenta los datos para buscar las variables a predecir (es decir, lo datos no están etiquetados), pero en este caso ambos enfoques dan un resultado positivo para la clasificación de las piezas del ajedrez.

Conclusiones

En este trabajo, se ha demostrado la viabilidad de un sensor virtual táctil, implementado sobre las pinzas de un brazo robótico para clasificación de objetos. Sin embargo, el presente documento proporciona directrices generales para el diseño de sensores virtuales basadas en datos, es decir, que la metodología de diseño propuesta en este trabajo puede utilizarse para el desarrollo de sensores virtuales basado en datos para cualquier tipo de aplicación.

Cuando se sabe poco acerca de la naturaleza de los fenómenos a modelar, los sensores virtuales basados en datos o de caja negra, son una herramienta viable para modelar este tipo de fenómenos y cuya efectividad ha sido probada en los últimos años, mediante su implementación exitosa como estimadores de variables. Este tipo de sensores estiman una cantidad de interés por medio de un conjunto de variables que se tienen disponibles sobre ejemplos resueltos. Una cuestión importante en el modelado basado en datos, es encontrar las variables de entrada adecuadas que se relacionen mejor a una salida deseada. La selección del modelo es sin duda un paso crucial en el desarrollo de un sensor virtual, pues es el que determina la salida del sensor virtual al relacionarla con sus entradas disponibles.

Los sensores virtuales son una herramienta poderosa para la estimación de variables. Una herramienta que cada día es más usada en diferentes áreas de la instrumentación. En este trabajo se presentó un claro ejemplo de que los sensores virtuales no solo pueden ser implementados en la industria de procesos, sino que pueden ser utilizados como una opción de bajo costo para detección y clasificación de objetos en la robótica.

Una de las ventajas del uso de las redes neuronales para clasificación es que el diseño de la red está basado en datos reales, es decir, que se trabaja con señales reales y se obtienen resultados igualmente reales. Sin embargo el diseño de la red neuronal depende del problema que se desea solucionar, actualmente no existe un diseño o estructura general que pueda

aplicarse a todos los problemas. Por ejemplo, el número de neuronas en la capa oculta es una cuestión que depende del problema que se está tratando. Se debe tener en cuenta que las neuronas en la capa oculta trabajan como extractores de patrones globales, ya que combinan las salidas de las neuronas de la primera capa operándolas en una región particular del espacio de entrada. Sin embargo, el uso de neuronas de sobra puede acarrear problemas de mala generalización y por el contrario la falta de neuronas en la capa oculta puede provocar perdida de caracterización.

También el número de capas ocultas es un factor importante a tomar en cuenta en el diseño de una red neuronal, y este número está determinado por el problema a resolver, por ejemplo, para clasificación una red neuronal generalmente trabaja con dos capas ocultas, debido a que en ocasiones una capa no es suficiente para extraer las características de todos los datos, y una segunda capa podría ayudar a realizar un mejor trabajo. Durante el diseño de la red neuronal se propusieron diferentes configuraciones y fueron probadas para evaluar su desempeño. Por ejemplo, en el caso de la selección del algoritmo de aprendizaje, se llegó a la conclusión que el algoritmo de Levenberg-Marquardt tiene el tiempo de convergencia más rápido al error deseado, y fue por esta razón que se seleccionó como el algoritmo de aprendizaje para la RNA.

El diseño y construcción de un brazo robótico fue el mayor reto en la realización de este trabajo, debido a que conjugó diferentes áreas de la ingeniería como son el diseño mecánico, la electrónica, el control y la programación. El brazo robótico forma parte del proyecto que lleva por nombre MARCO (Multimodal Autonomous Robotic Chess Opponent), un jugador virtual de ajedrez que consiste en un brazo robótico y un agente virtual que muestra emociones similares a las de un humano durante un juego de ajedrez. MARCO fue construido en la universidad de Albert Ludwigs de Friburgo Alemania, durante una estancia de investigación de Octubre del 2013 a Abril del 2014. Como resultado de dicho trabajo se realizó un documento (ver Apéndice 6. Paper submitted in the international Conference on Human-Agent Interaction (HAI 2014)) para ser presentado en la segunda conferencia internacional HAI 2014 (Humant-Agent Interaction), que se llevará a cabo en Tsukuba Japón.

De acuerdo a los resultados obtenidos en las pruebas realizadas al sensor virtual, se determina que se llevó a cabo un correcto diseño y selección del modelo. Así como una correcta selección de las variables independientes (entradas) que determinan la variable dependiente

(salida). Por último, se concluye que el desarrollo de un sensor virtual para la detección y clasificación de piezas de ajedrez mediante las pinzas de un brazo robótico se llevó a cabo con éxito, y el sensor puede ser utilizado de manera confiable durante un juego de ajedrez real.

Trabajo futuro

A continuación se enlistan algunos puntos que describen trabajos que pueden desarrollarse más adelante relacionados al trabajo presentado en esta tesis.

- Realizar algoritmos de autoajuste. Implementar algoritmos para evitar mediciones erróneas. Por ejemplo, en un desajuste de la velocidad, implementar un algoritmo que mida la velocidad de sujeción de las pinzas y que modifique los rangos de valores para cada pieza realizando un mapeo con los nuevos parámetros de velocidad.

- Clasificar formas más complejas. Al agregar robustez al sensor virtual, se permitirá clasificar diferentes tipos de objetos con formas y tamaños muy variados, que permitirá revolucionar la interacción de los robots con el mundo real.

- Agregar sensibilidad a las pinzas del brazo robótico. Implementar un algoritmo de detección que permita la sujeción de piezas más delicadas al tacto, para esto será necesario un control más preciso para la sujeción de las piezas.

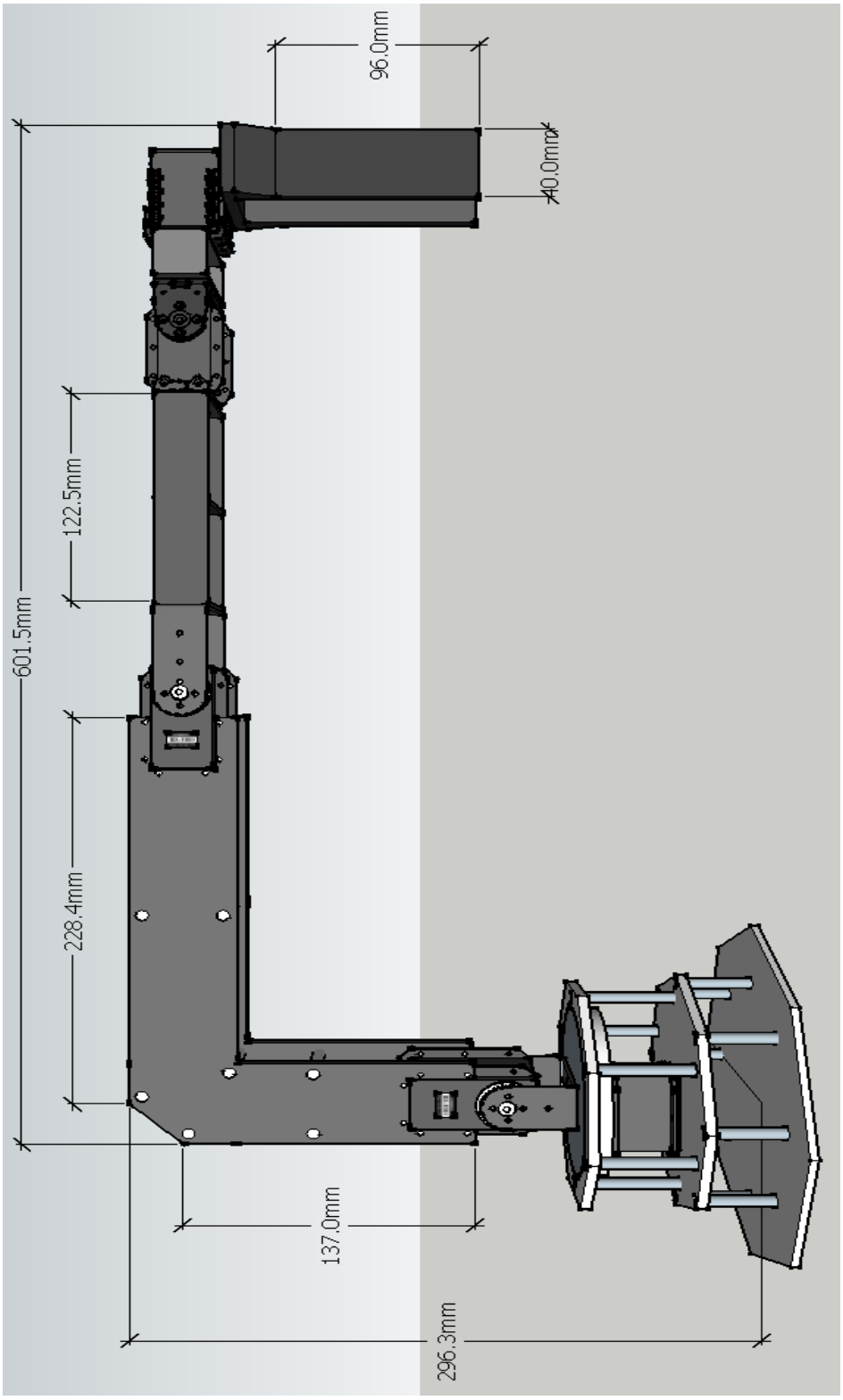
- Probar nuevos algoritmos para la clasificación de datos. Seguir explorando las posibilidades que el agrupamiento difuso ofrece en la clasificación de datos, y así desarrollar nuevas técnicas para el desarrollo de sensores virtuales basados en datos.

Parte III

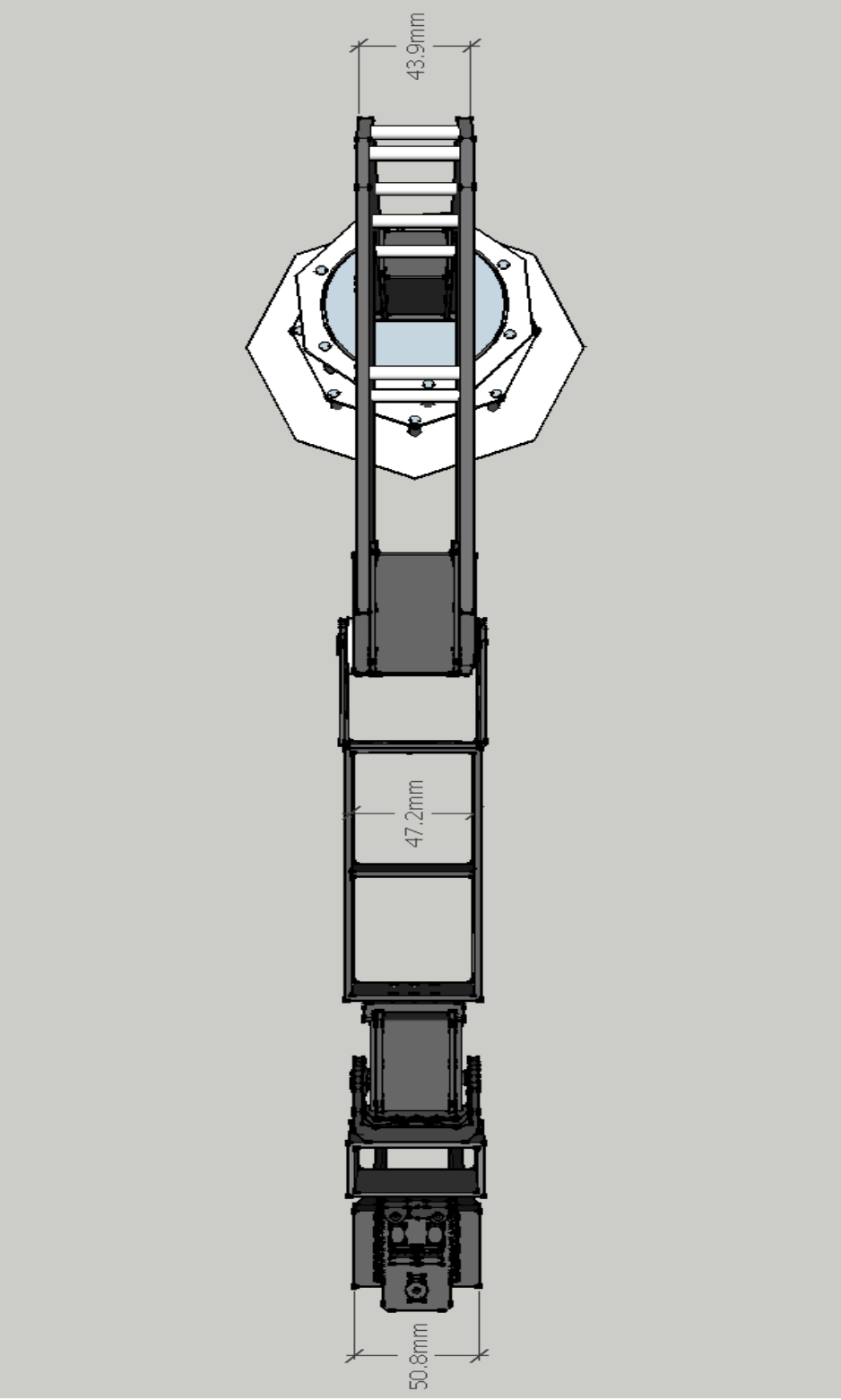
Apéndices

Apéndice 1. Dimensiones del brazo robótico

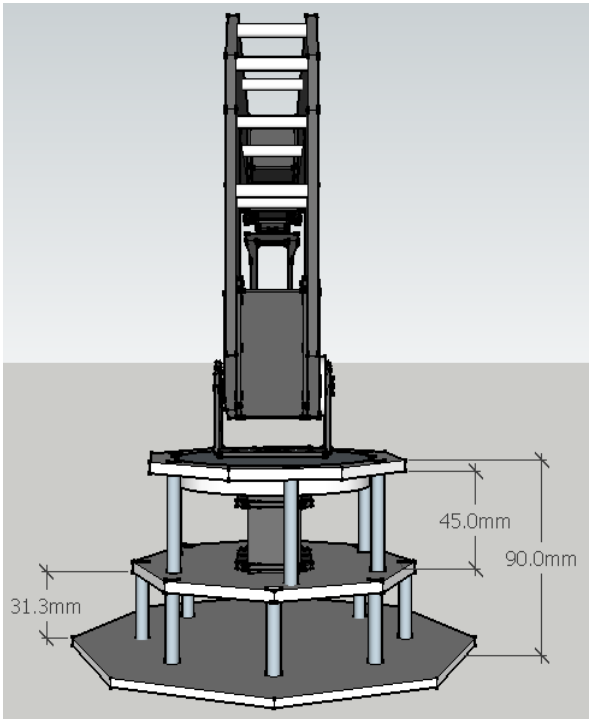
Vista lateral derecha:



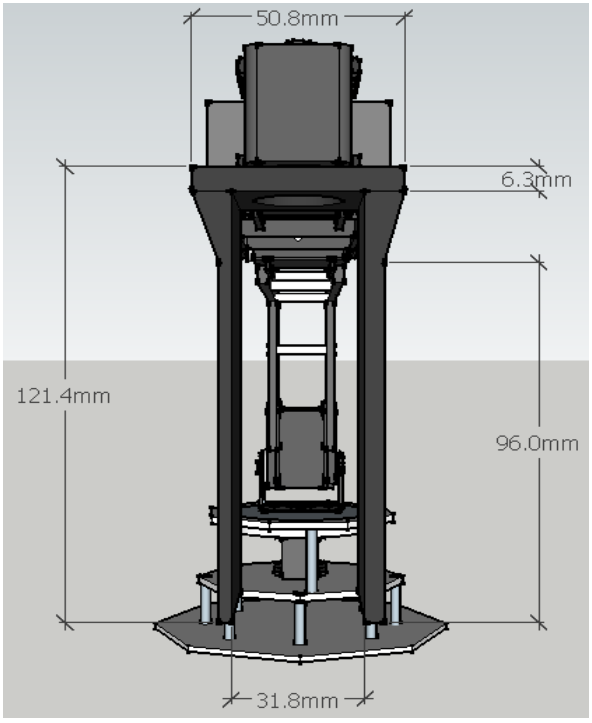
Vista superior:



Vista trasera:

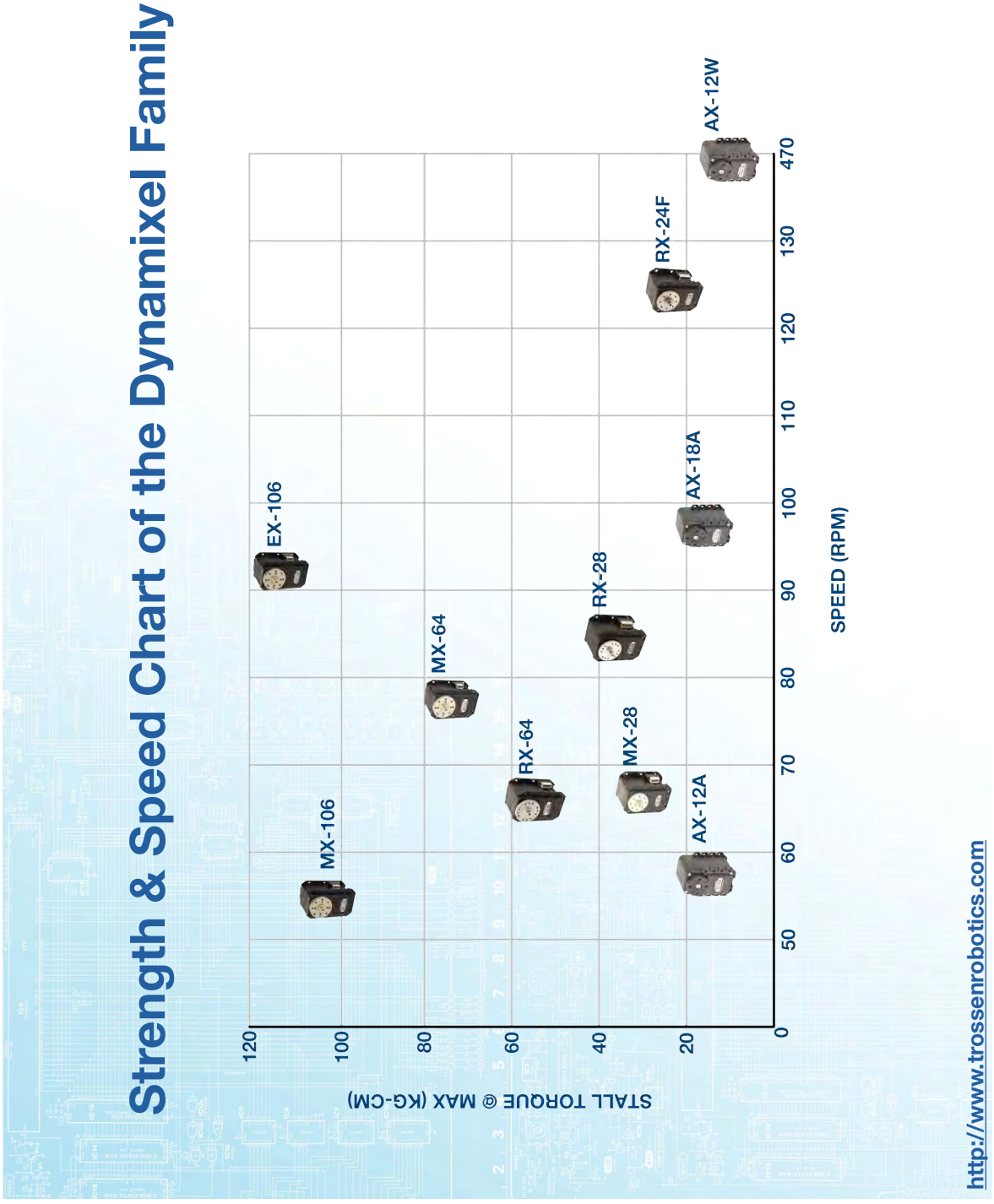


Vista frontal:



Apéndice 2. Especificaciones de la familia Dynamixel

Tabla comparativa:



Dynamixel AX-12A:

Dynamixel
AX Series

AX Series Dyanmixels & Brackets
(Click to View Full Line)

Model	AX-12A (Visit Product Page)	AX-12W (Visit Product Page)	AX-18A (Visit Product Page)
Stall Torque @ Max Voltage	1.5N.m (16.5 kg-cm)	0.2N.m (2.0 kg-cm)	1.8N.m (18 kg-cm)
Speed (RPM)	59	470	97
Nominal Operating Voltage	12v	12v	12v
Stall Current Draw	1.5A	1.4A	2.2A
Dimensions	32x50x40 mm	32x50x40 mm	32x50x40 mm
Weight	54.6g	52.9g	54.5g
Resolution	0.29°	0.29°	0.29°
Operating Angle	300	300	300
Gear Reduction	254 : 1	32 : 1	254 : 1
Geartrain Material	Eng. Plastic	Eng. Plastic	Eng. Plastic
Onboard CPU	ATMega8 (ATMEGA8-16AU@16MHZ, 8 Bit)	ATMega8 (ATMEGA8-16AU@16MHZ, 8 Bit)	ATMega8 (ATMEGA8-16AU@16MHZ, 8 Bit)
Position Sensor	Potentiometer	Potentiometer	Potentiometer
Com Protocol	TTL	TTL	TTL
Com Speed	1mbps	1mbps	1mbps
Compliance/PID	Compliance	Compliance	Compliance
Dimensional Drawing	PDF	PDF	PDF

http://www.trossenrobotics.com

Dynamixel MX-28:

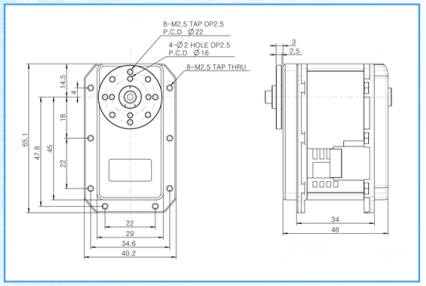
24/28 Series Dyanmixels & Brackets
(Click to View Full Line)

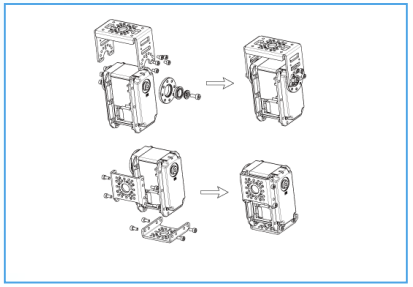
Model	RX-24F (Visit Product Page)	RX-28 (Visit Product Page)	MX-28 (Visit Product Page)
Stall Torque @ Max Voltage	2.6N.m (26.5 kg-cm)	3.7N.m (37.7 kg-cm)	3.1N.m (31.6 kg-cm)
Speed (RPM)	126	85	67
Nominal Operating Voltage	12v	12-18.5v	11.1-14.8v
Stall Current Draw	2.4A	1.9A	1.7A
Dimensions	35.6x50.6x35.5 mm	35.6x50.6x35.5 mm	35.6x50.6x35.5 mm
Weight	67g	72g	72g
Resolution	0.29°	0.29°	0.088°
Operating Angle	300	300	360
Gear Reduction	193 : 1	193 : 1	193 : 1
Geartrain Material	Hardened Steel	Hardened Steel	Hardened Steel
Onboard CPU	ATMega8 (ATMEGA8-16AU @ 16MHZ, 8 Bit)	ATMega8 (ATMEGA8-16AU @ 16MHZ, 8 Bit)	Cortex M3 (STM32F103C8 @ 72MHZ, 32 Bit)
Position Sensor	Potentiometer	Potentiometer	Magnetic Encoder
Com Protocol	RS-485	TTL	TTL/RS-485
Com Speed	1mbps	1mbps	3mbps
Compliance/PID	Compliance	Compliance	PID
Dimensional Drawing	PDF	PDF	PDF

Dynamixel MX-64:

Model	RX-64 (Visit Product Page)	MX-64 (Visit Product Page)
Stall Torque @ Max Voltage	5.3N.m (54 kg-cm)	7.3N.m (74.4 kg-cm)
Speed (RPM)	64	78
Nominal Operating Voltage	12-18.5v	11.1-14.8v
Stall Current Draw	2.6A	5.2A
Dimensions	40.2x61.1x41 mm	40.2x61.1x41 mm
Weight	125g	126g
Resolution	0.29°	0.088°
Operating Angle	300	360
Gear Reduction	200 : 1	200 : 1
Geartrain Material	Hardened Steel	Hardened Steel
Onboard CPU	ATMega8 (ATMEGA8-16AU @ 16MHZ, 8 Bit)	Cortex M3 (STM32F103C8 @ 72MHZ, 32 Bit)
Position Sensor	Potentiometer	Magnetic Encoder
Com Protocol	RS-485	TTL/RS-485
Com Speed	1mbps	3mbps
Compliance/PID	Compliance	PID
Dimensional Drawing	PDF	PDF

Dynamixel
106 Series





106 Series Dyanmixels & Brackets
(Click to View Full Line)

Model	EX-106+ (Visit Product Page)	MX-106 (Visit Product Page)
Stall Torque @ Max Voltage	10.9N.m (111 kg-cm)	10.0N.m (101 kg-cm)
Speed (RPM)	91	55
Nominal Operating Voltage	12-18.5v	11.1-14.8v
Stall Current Draw	7A	6.3A
Dimensions	40.2x65.1x46 mm	40.2x65.1x46 mm
Weight	154g	153g
Resolution	0.29°	0.088°
Operating Angle	251	360
Gear Reduction	184 : 1	225 : 1
Geartrain Material	Hardened Steel	Hardened Steel
Onboard CPU	ATMega8 (ATMEGA8-16AU @ 16MHZ, 8 Bit)	Cortex M3 (STM32F103C8 @ 72MHZ, 32 Bit)
Position Sensor	Magnetic Encoder	Magnetic Encoder
Com Protocol	RS-485	TTL/RS-485
Com Speed	1mbps	3mbps
Compliance/PID	Compliance	PID
Dimensional Drawing	PDF	PDF

http://www.trossenrobotics.com

Apéndice 3. Código de detección de piezas con ARDUINO

```
//programa que detecta las piezas en el gripper
     del brazo robótico
#include <DynamixelSerial3.h>

//Declaración de variables
int i;
int Position6;
int Current6,cont;
int incomingByte[10];
unsigned long timer=0,timer2=0;

//Inicialización de comunicación con
    servomotores y puerto serial
void setup()
{
Serial.begin(9600);
Dynamixel.begin(1000000,2);
delay(1000);
Serial.println("Ready");
Dynamixel.setEndless(6,OFF);
}

void loop()
```

```
{

 while (1)
        {
         if (Serial.available() > 0)//Verifica
              existencia de datos en el buffer
             de recepción
                {
                int availableBytes = Serial.
                    available();//Número de
                    datos leidos
                for (i=0;i<availableBytes;i++)
                    {
               incomingByte[i] = Serial.read();//
                   Almacena datos
                    }
                 if (incomingByte[0]==0x02 &&
                     incomingByte[0]==0x04 &&
                     incomingByte[0]==0x02 &&
                     availableBytes==3)//
                     Verifica si es comando de
                     cerrar pinzas
                    {
                    break;
                }
                 else//no es un comando de
                     cerrar pinzas
                    {
                    availableBytes=0;//Reset
                        de número de datos
                        leidos
                    }
             }
        }
        Closing();//Llamado a funcion para
            cerrar gripper
        delay(2000);
```

```
            Opening();//Llamado a función para
                abrir gripper
            delay(2000);

}

void Closing()
{
  Dynamixel.setEndless(6,ON);//Activa modo
      rotación continua
  Dynamixel.turn(6, RIGTH, 200);//Comienza a
      cerrar las pinzas del brazo robótico
  //reset de variables
  cont=0;//Reseteo contador
  timer=2000;//Tiempo limite para sujetar una
      pieza
  timer2=0;//Reset de tiempo de detección

  while(timer>0)//Permanecer aqui mientras el
      tiempo no llegue a cero
       {
       timer=timer-1;//Reduce el timer un valor
            de uno
       timer2++; //Comienza conteo del tiempo
           de recepción
       Position6 = Dynamixel.readPosition(6);
           //Lee posición del servomotor que
           controla las pinzas
       Current6 = Dynamixel.readLoad(6); //Lee
           corriente del servomotor que controla
            las pinzas
       delay(10);//retardo de 10ms
       if (Current6==1224)//la corriente ya
           llego a 204 mA?
          {
          cont=cont+1;//aumento contador
          }
```

```
        if (Current6 < 1224 && Current6 > 1224)
            //Si la corriente no es 201 mA
            {
            cont=0;//reseteo contador
            }
        if (cont > 5)//Se leyó 5 veces
            consecutivas la corriente en 204 mA?
            {
            Dynamixel.turn(6, LEFT, 0);//Detiene
                servomotor que controla las pinzas
            break;//salir
            }
        }

  Serial.print(timer2, DEC);//Manda tiempo de
      detección
  Serial.print(',');
  Serial.println(Position6, DEC);//Manda
      posición de detección

}

 void Opening()
 {
 Dynamixel.setEndless(6,OFF);//Abre las pinzas
     del brazo robótico
 }
```

Apéndice 4. Código para la implementación de la red neuronal en MATLAB

```
% Entradas y salidas
net.inputs= inputs;
net.outputs=targets;

hiddenlayer1=6;% neronas en capa oculta 1
hiddenlayer2=6;% neronas en capa oculta 2

% Perceptrón multicapa
net = feedforwardnet([hiddenlayer1 hiddenlayer2
    ]);

% Función de activación en la capa oculta 1
net.layers{1}.transferFcn = 'tansig';
% Función de activación en la capa oculta 2
net.layers{2}.transferFcn = 'tansig';
% Función de activación en la capa de salida
net.layers{3}.transferFcn = 'tansig';

% Normalización de los datos [-1 1]
net.inputs{1}.processFcns = {'
    removeconstantrows','mapminmax'};
```

```
net.outputs{2}.processFcns = {'
    removeconstantrows','mapminmax'};

% configuración de la red (entradas y salidas)
net = configure(net,inputs,targets);

% División de los datos para el entrenamiento,
    validación y prueba
net.divideFcn = 'dividerand';
net.divideMode = 'sample';
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Algoritmo de entrenamiento
net.trainFcn = 'trainlm';  % Levenberg-
    Marquardt

% Tipo de error para evaluar rendimiento de la
    red
net.performFcn = 'mse';  % Error cuadrático
    medio

% Gráficas a desplegar
net.plotFcns = {'plotperform','plottrainstate',
    'ploterrhist', ...
  'plotregression', 'plotroc', 'plotconfusion',
      'ploterrcorr'};

% Parametros de entrenamiento
net.trainParam.epochs = 1000;% Número máximo de
     épocas
net.trainParam.time = inf;% Tiempo de
    entrenamiento en segundos
net.trainParam.goal = 0.01;% Error a alcanzar
net.trainParam.min_grad=1e-10;  % Mínimo
    rendimiento del gradiente
```

```
net.trainParam.mu_max = 1e10;; % Máximo valor
    del factor de aprendizaje
net.trainParam.mu = .001; % Factor de
    aprendizaje
net.trainParam.max_fail = 6; % Número máximo de
    fallas al validar

% Entrenar la red
[net,tr,Y,E] = train(net,inputs,targets);

% probar la red
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)


% Calculo del desempeño de la red con los datos
     de entrenamiento, validacion y prueba
trainTargets = targets .* tr.trainMask{1};
valTargets = targets  .* tr.valMask{1};
testTargets = targets  .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,
    outputs)
valPerformance = perform(net,valTargets,outputs
    )
testPerformance = perform(net,testTargets,
    outputs)

% Porcentaje de clasificaciones correctas e
    incorrectas.
[c,cm] = confusion(targets,outputs)

fprintf('Porcentaje de clasificaciones
    correctas   : %f %%\n', 100 * (1-c));
fprintf('Porcentaje de clasificaciones
    incorrectas : %f %%\n', 100 * c);

% View the Network
```

```
view(net)

% Simulación
rey=[44.91 ; 2.53];
reina=[21.99 ; 4.04];
alfil=[0 ; 4.04];
caballo=[0 ; 3.62];
torre=[0 ; 4.60];
peon = [0; 3.36];
p = [53.66; 2.674];
todos=[44.91 21.99 0 0 0 0 53.66; 2.53 4.04
    4.04 3.62 4.60 3.36 2.674];

% Salida de la red neuronal
yout = sim(net,todos)
```

Apéndice 5. Pesos y Bias finales de la red neuronal

Matriz de pesos en capa oculta 1 (6x2):

$$W^{oculta1} = \begin{bmatrix} 2.4779 & 2.2196 \\ -0.1646 & 4.4485 \\ -4.0436 & -0.8049 \\ 3.7952 & -2.8936 \\ 2.0907 & 4.2366 \\ 2.8696 & -0.9423 \end{bmatrix}$$

Matriz de pesos en capa oculta 2 (6x6):

$$W^{oculta2} = \begin{bmatrix} -3.4550 & 8.9704 & 0.0707 & 2.3692 & -2.2221 & -4.1011 \\ 0.2761 & -6.0673 & 1.2870 & -2.2649 & -0.6326 & -8.1295 \\ 1.5617 & 3.6535 & -2.0781 & -3.4762 & -0.5039 & -4.0474 \\ -0.1010 & -3.4336 & -4.7945 & 2.2742 & 0.0201 & -2.0374 \\ 2.5988 & -1.9430 & -4.7775 & -1.2666 & 1.9945 & 1.1088 \\ 0.0713 & 0.9948 & -0.0366 & -1.0837 & -1.6238 & 1.2941 \end{bmatrix}$$

Matriz de pesos en capa de salida (7x6):

$$W^{salida} = \begin{bmatrix} -6.3676 & 2.9340 & 4.4172 & 8.4119 & -6.4081 & -4.5191 \\ -0.8811 & -7.6203 & -6.2861 & -7.0106 & -0.2720 & -5.8193 \\ -2.5848 & -8.6200 & 8.3830 & -0.5007 & -0.5396 & -2.7477 \\ 8.3785 & 7.0493 & -7.2965 & -0.1269 & 2.6555 & -5.9629 \\ 0.3993 & 9.3089 & 7.5222 & 2.2276 & 2.1170 & -2.4849 \\ -8.2764 & 2.4099 & -1.9609 & -0.5902 & 0.3422 & -1.9636 \\ -2.1596 & 0.4263 & -0.6662 & 3.6146 & 6.8643 & 0.8523 \end{bmatrix}$$

Matriz de bias en capa oculta 1 (6x1):

$$B^{oculta1} = \begin{bmatrix} -3.7881 \\ -0.1910 \\ 1.6536 \\ -0.9970 \\ -3.4315 \\ 3.1441 \end{bmatrix}$$

Matriz de bias en capa oculta 2 (6x1):

$$B^{oculta2} = \begin{bmatrix} 4.5301 \\ -0.3428 \\ -1.4056 \\ -0.7964 \\ -1.0908 \\ 2.7984 \end{bmatrix}$$

Matriz de bias en capa de salida (7x1):

$$B^{salida} = \begin{bmatrix} -4.3640 \\ -7.3913 \\ -3.1294 \\ -6.2120 \\ -2.2223 \\ -3.1307 \\ -3.2264 \end{bmatrix}$$

Apéndice 6. Paper submitted in the international Conference on Human-Agent Interaction (HAI 2014)

The hybrid agent MARCO

A Multimodal Autonomous Robotic Chess Opponent

anonymous for blind review

ABSTRACT

This paper introduces MARCO, a hybrid, chess playing agent equipped with a custom-built robotic arm and a hybrid virtual agent displaying emotions. MARCO was built to investigate the hypothesis that such hybrid agents capable of displaying emotions make human-computer interaction more personal and enjoyable and, thus, potentially more effective. Both the hardware and software components of the system's architecture are described. The hardware components consist of eight Dynamixel servos, an Arduino-based control board, a 5.6 inch display, and a DGT chessboard with a digital clock. The software components run concurrently as separate processes. The main components are the virtual agent framework MARC, the WASABI Affect Simulation architecture, and the TSCP chess engine. It is described in detail how the agent's emotions result from changes in the evaluations provided by the chess engine and how they are expressed by virtual agent. The paper concludes with an outline of how we plan to evaluate the effect of MARCO's different components on player enjoyment and how we aim to systematically assess and increase its overall level of human-likeness in the future.

ACM Classification Keywords

I.2.9 Computing Methodologies: Artificial Intelligence—*Robotics*

General Terms

Chess, Robotics, Human-agent interaction, Hybrid agent, Affective Computing

INTRODUCTION

Chess has been called the "Drosophila of artificial intelligence" [1] meaning that in the same way as the drosophila melanogaster has become the model organism for biological research, chess served at least for many years as a standard problem for artificial intelligence research. When in 1997 Garry Kasparov, who was ranked first at that time, lost against IBM's supercomputer "Deep Blue" [9], this problem was assumed to be solved and chess engines would nowadays outclass the best players. Altogether this triggered researchers to shift their attention to other games, such as Go. Today, for a casual chess player it can be rather frustrating to play against the computer, because he will lose most of the times and the computer moves its pieces with seemingly no hesitation.

With the advent of humanoid agents, both robotic and virtual or even hybrid as presented here, chess offers a good opportunity for system evaluation. Robotics researchers find an interesting challenge for both hard- and software design in grabbing and moving chess pieces [14]. Researchers in the fields of human-computer interaction and affective computing use chess as a situational context, which does not need to be explained to most people letting them instantly dive into the interactive experience. Questions such as how different embodiments might change player satisfaction and how the integration of a virtual agent expressing emotions influences a human's stance towards a computer system have recently been investigated. These are rooted in the humans' tendency to treat machines as social actors and this effect is expected to be stronger the more human-like the machine is designed to appear [15]. Accordingly, with our setup we aim to contribute to solving this puzzle.

The remainder of the paper is structured as follows. After discussing related work in the next section, our general motivation is explained and two research questions are introduced. Then the Elo rating will be explained together with how the employed chess engine evaluates board positions. Subsequently, MARCO's hardware components are detailed, before the interconnection of its software components is laid out. Then, the complete system is explained. Finally, we present our ideas concerning experimental protocols for evaluating MARCO. We conclude this paper with a general discussion.

RELATED WORK

This section describes research projects involving chess playing robots [14, 16, 12]. They aim to answer different research questions and, therefore, they employ systems of different size and complexity.

"Gambit" is a good example for an engineer's solution to an autonomous chess-playing robotic system [14]. With their "robot manipulator system" the authors created a "moderate in cost" (i.e. 18K USD) manipulator that is able to play chess with arbitrary chess sets on a variety of boards without the need to model the pieces. Although their system does not have any anthropomorphic features, it includes a "natural spoken language interface" to communicate with the human opponent. Most importantly, "Gambit" tracks both the board and the human opponent in real time so that the board does not need to

be fixed in front of the robot. With its available six degrees of freedom (DoF) and the USB camera mounted on top of its gripper the robot arm reliably grasps a wide array of different chess pieces, even if they are placed poorly. In result, it outperformed all robotic opponents at the 2010 AAAI Small Scale Manipulation Challenge. Unfortunately, no data on human players' enjoyment is available.

In contrast to the remarkable technical achievements behind the development of "Gambit", the "iCat" from Philips was combined with a DGT chess board to investigate the influence of embodiment on player enjoyment in robotic chess [12]. The authors conducted a small-scale empirical trial with the emotional iCat opponent either presented in its virtual or robotic form. Using a modified version of the GameFlow model [18], it was found that overall the virtual version is less enjoyable than the robotic one. A subsequent long term study [13] with the robotic iCat playing chess repeatedly against five children showed, however, that these children lost interest in the robot. Presumably, iCat's complete lack of any manipulation capability together with its cartoon-like appearance let the children ignore the robot completely after the initial curiosity is satisfied.

Similar to our approach, Sajó et al. [16] present a "hybrid system" called "Turk-2" that consists of a "mechanically simple" robot arm and a rather simple 2D talking head presented on a computer display. "Turk-2" can analyze three emotional facial expressions, namely *sad*, *neutral*, and *happy*, and additional image processing enables the system to monitor the chess board. Interestingly, the authors decided to artificially prolong the system's "thinking time", details of which are unfortunately not reported. The transitions between the talking head's facial expressions *neutral*, *sad*, *happy*, and *bored* are controlled by a state machine that takes the human's emotion (as derived from its facial expression) and the game state into account. Similar to our approach, the talking head will change into a bored expression after some time without input has passed. An empirical study on the effect of the presence of the talking head revealed that without the talking head the players mostly ignored the robotic arm, which was positioned to the right of the board, even during the turn of "Turk-2". With the talking head in front of them, however, the players not only looked at the talking head but also started smiling and laughing.

Regarding the effects of a virtual agent's facial expression of emotions on human performance in a cognitive task, an empirical trial resulted in no significant differences [7]. In addition, the study showed that for such a serious task it made no difference, if the agent's emotions were generated based on a set of hard-coded rules or by making use of a sophisticated and complex emotion simulation architecture. The authors speculate that a less cognitively demanding and more playful task might be better suited to search for such effects.

MOTIVATION AND RESEARCH QUESTIONS

These previous results in combination motivated us to include the following features in MARCO, our Multimodal, Autonomous, Robotic Chess Opponent:

1. A low-cost robotic arm that enables MARCO to autonomously move the chess pieces instead of having to rely on the human opponent's assistance (as in [12])

2. A custom built, small sized, robotic display presenting a highly anthropomorphic virtual agent's head to realize a hybrid embodiment combining the best of both worlds (combining [12] and [16])

3. A flexible software architecture that relies on an established emotion simulation architecture as one of its core modules (following up on [7])

The resulting MARCO system will help answering research questions that are motivated by the previous work presented above:

1. Is it more enjoyable to play chess against the robotic arm with or without the virtual agent next to it?

2. Is it more enjoyable to play against the hybrid agent (i.e. the robotic arm with the virtual agent) when the agent expresses emotions as compared to when it remains equally active but emotionally neutral?

The first questions will provide a baseline for the hardware components of our system and will be compared with those reported in [16] with regard to "Turk-2". It is not taken for granted that a more complex system will always be preferable to a simpler system from the perspective of a human player. The second question, however, is targeting the role that artificial emotions might or might not play and it is motivated by previous results [7]. Perhaps, the sole integration of a moving display with a virtual agent commenting the game is already sufficient to maximize player enjoyment?

Finally, MARCO allows us to tackle systematically the general question of how and when "mindfullness" is ascribed to machines [15]. Is the most human-like and emotional agent evaluated as more social than the less complex/human-like versions of it? How does this depend on the human chess player's ranking in chess, if at all?

BACKGROUND AND PRELIMINARIES

Elo rating

The skill of chess players is usually measured in terms of a single integer value, the so-called Elo Rating [11]. It represents the relative strength of a player, the higher the better, and it increases or decreases with his or her chess match results.

The Elo rating is considered reliable to determine the winning chances of one player against another by taking the difference of their respective Elo ratings. Given two players A and B and their respective Elo ratings

R_A and R_B, the expected result E_A for the player A is determined by the following equation:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \quad (1)$$

whereby the two extremes 0 and 1 are to be interpreted as a defeat or a win, respectively, and the exact value 0.5 represents a draw. After a game is played, the rating of a player is modified. With $S_A \in \{0, 0.5, 1\}$ being the score of the player A in a game, $R'_A = R_A + K(S_A - E_A)$ will be the new rating where K is a coefficient ranging from 10 to 30 in the FIDE system. Currently, ELO rating in chess goes from 1000 (complete beginner) to 2880 (Magnus Carlsen World Champion). The coefficient K represents the pace at which a rating can evolve.

EXAMPLE 1. *Paul has a rating of* 1650 *and George has a rating of* 1760*, both with K=15. Paul expects a result of* 0.35*. The game ended in a draw meaning that Paul's new rating is* $1650 + 15 * (0.5 - 0.35) = 1652$.

We expect that differences in the evaluations of our system might correlate with or even depend on the ELO ratings of the human players. In addition, we aim to use our system as a virtual coach for novice players to improve their chess skills and the ELO rating provide a standard means to compare player strength before and after training. Admittedly, this is future work.

Chess Engine

The TSCP chess engine evaluates the position using an alpha-beta algorithm with a depth d given as parameter based on a number of criteria like: pieces left on the board, activity of these pieces, security of the king, etc. The greater the depth the more precise is the evaluation. The position evaluation function results in a real number e ranging from $[-\infty, +\infty]$ where 0 means that the position is equal, $-\infty$ that black is winning and $+\infty$ that white is winning. A +1 valuation roughly represents the advantage equivalent to a pawn, +3 to a knight or a bishop, and so on according to the standard valuation of chess pieces. We denote by $e_{t,d}$ the evaluation given by the chess engine at move t with depth d. We write e when it is clear from the context. In practice, once $|e| \geq 5$ the game is more or less decided.

We chose the TSCP chess engine [2] for its simplicity and in order to make our results comparable to previous work on the iCat playing chess [12], for which the same engine was used.

The communication between the user and the TSCP chess engine is handled by the XBoard Chess Engine Communication Protocol [3]. Originally implemented as a means to facilitate communication between the GNU XBoard Chess GUI and underlying chess engines, this plain text protocol allows for easy information exchange in a human readable form.

HARDWARE DESCRIPTION

The complete setup is presented in Figure 1. The hardware used comprises a 5.6 inch pan-tilt display to present the virtual agent's face, a robotic arm to the left the agent to move the chess pieces, and a digital chess board (DGT USB Rosewood) with a chess clock. Each of these components will be described in detail below.

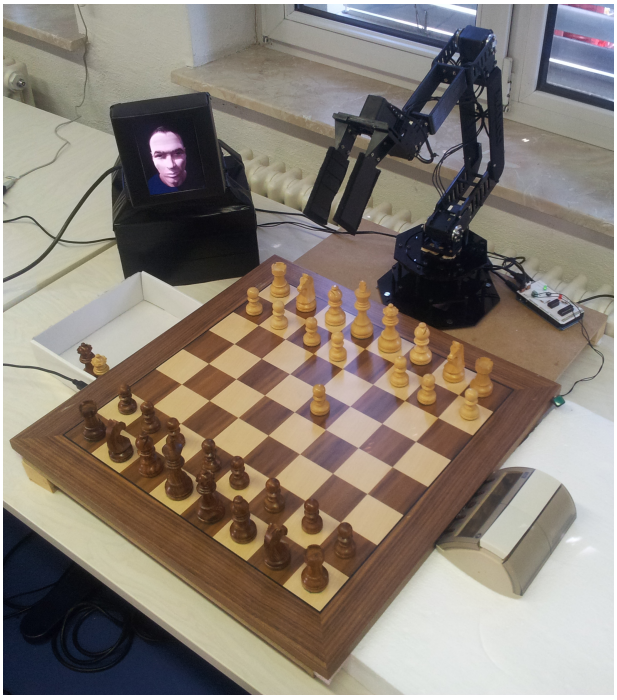
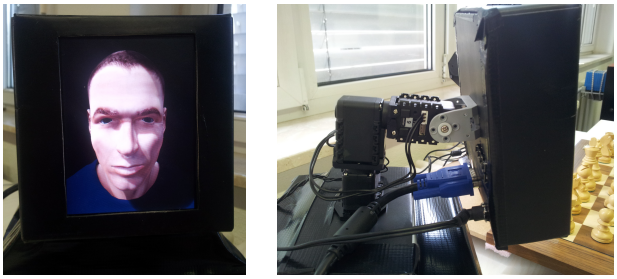


Figure 1. The pan-tilt agent display, the robotic arm, and the digital chess board together with the digital chess clock

The pan-tilt agent display



(a) Frontal view (b) Side view

Figure 2. The virtual agent provided by the MARC component as it is presented on a 5.6 inch pan-tilt display

The pan-tilt display component features a 5.6 inch upright TFT LCD display with a physical resolution of 640×480 pixels and 16bit color depth, cp. Fig. 2(a). It is positioned to the left of the robotic arm to give the impression that these two components belong together and it is mounted high enough that the virtual agent could potentially overlook the complete chess board. Three Dynamixel AX-12A servos (cp. Fig. 2(b)) are connected to an Arduino-based control board to change the display

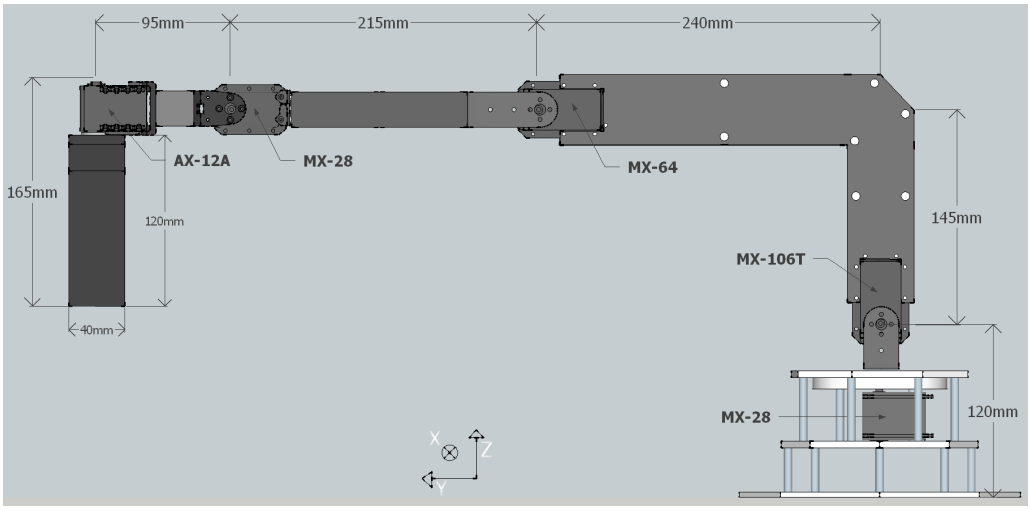


Figure 3. A schematic of the robotic arm with annotations of link lengths and Dynamixel servos used for each joint position

orientation during the game along all three axes. Currently, however, only pan and tilt are used to turn the display up, down, left, and right.

The robotic arm

The design of the robot's arm is based on the "WidowX Robotic Arm Kit Mark II" [4] available from Trossen Robotics. The rotational base remained unchanged (cp. Fig. 4), but the arm itself needed to be extended and the gripper modified. The extensions for the arm were printed with a 3D printer allowing us to also personalize the robot by imprinting its name "MARCO" on the side of its first link.

(a) The robot's rest configuration (b) Picking up a piece close to its base

Figure 4. Two configurations showing the flexibility of our robotic arm

Five Dynamixel servos of four different families move the robot's arm, cp. Fig. 3. For the base and wrist two MX-28 servos are used. An MX-64 servo moves the robot's elbow and an MX-106 servo its shoulder. The gripper is opened and closed by an AX-12A servo.

The gripper jaws were specifically built to grab the Staunton chess pieces on the DGT board regardless of their height or size. We designed a hollow gripper to form a structure that conforms with the objects. The gripper has only two states, either opened or closed, cp. Fig. 5.

(a) Open state (b) Closed state

Figure 5. The two states of the robot's custom designed gripper picking up a white bishop

With a maximum reach of $550mm$ the robotic arm can reach all 64 squares of the $480mm \times 480mm$ DGT tournament chess board. Due to the configuration of its three parallel joints (MX-106T, MX-64, and MX-28 in Fig. 3) it can fold into a rest position (cp. Fig. 4(a)) as well as also pick and place any piece in the first row closest to its base (cp. Fig. 4(b)).

The DGT digital chess board

The DGT chess board is a wooden board with standard Staunton pieces and $55mm \times 55mm$ squares. Each piece is equipped with a unique RFID chip that makes it recognizable. The board is connected to the computer with

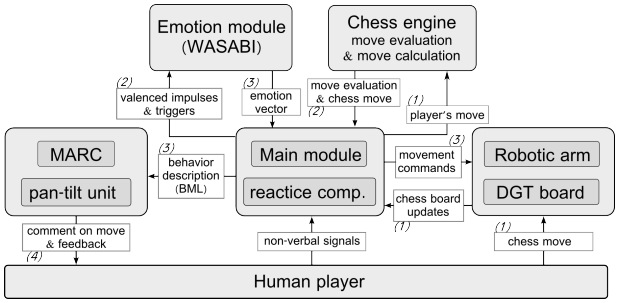


Figure 6. An outline of the software modules and their connections

a USB cable, and it transmits the position in FEN format to the engine every time a change is performed. The main module of our software framework (cp. Fig. 6) handles all human player's moves.

SOFTWARE DESCRIPTION

Except for the external MARC framework, the complete system is implemented in C++ using the Qt SDK [5] to enable cross-platform functionality. Communication with the hardware parts (i.e. the DGT chess board and the Arduino board) is realized by relaying their output to the Qt-specific event loop.

Overview of system components

The system consists of five main components which are linked together by the main module to form the chess playing agent (cp. Fig. 6):

- The DGT board to detect moving pieces
- The TSCP chess engine for position evaluation and move calculation
- The emotion engine WASABI to simulate MARCO's emotions
- The robotic arm to move the chess pieces on the board
- The MARC framework to create the agent's visual appearance on the pan-tilt display

The main information flow originates, on the one hand, from the DGT chess board, which detects when pieces are lifted or put down, and, on the other hand, from the chess engine, which reacts to a human player moving by calculating MARCO's next move. The result is sent as movement commands to the robotic arm and the virtual agent receives a behavior description using the behavior markup language (BML) [19].

Whenever chess pieces are moved on the DGT board, it concurrently updates the main module accordingly.

Deriving emotional states

The integration of emotions is achieved through the WASABI Affect Simulation architecture [8] as one of MARCO's software components. It concurrently updates the main module with the agent's dynamically changing emotional states in terms of *emotion vectors* that contain ordered lists of emotion/intensity pairs. As input WASABI needs *valenced impulses* and expectation-based emotions (e.g., *surprised*) need to be *triggered* before they can gain positive intensity. How these two types of inputs are derived from the sequence of game states is described next.

Valenced impulses

The main module transforms the consecutive evaluations of the chess board configurations e_t (at times t during the game) into *valenced impulses* $val(e_t)$ according to the Equation 2.

$$val(e_t) = k \times tanh\left(\frac{e_t}{r}\right) \tag{2}$$

In Equation 2, k is a scaling factor and by increasing the denominator $r \in [1, \infty]$ the skewness of the hyperbolic tangent is reduced until a quasi-linear mapping ($val(e_t) = k \times e_t$) is achieved. The hyperbolic tangent is introduced to let us emphasize small values of e_t relative to bigger values of e_t.

For example, choosing $k = 50$ and $r = 2$:

$$\begin{aligned} val(e_t) = 50 \times tanh\left(\frac{e_t}{2}\right) \in (2.5, 25], \\ \forall e_t \in \{x \in \mathbb{R} \mid 0.1 \leq x < 1.1\} \end{aligned} \tag{3}$$

Thus, with these constants any value of e_t between 0.1 and 1.1 results in a weak to medium valenced impulse. Observe that $|val(e_t)| \cong 50, \forall e_t \in \{x \in \mathbb{R} \mid |x| > 5\}$, meaning that a winning (or loosing) board configuration results in the maximum impulse of 50 (or minimum impulse of -50, respectively).

Depending on who plays white, the sign of the scaling factor k is adjusted as to map favorable board positions for MARCO to positively valenced impulses and vice versa. That is, if MARCO plays white k is positive, otherwise it is negative. For the time being, MARCO always plays white letting it perform the first half-move.

Inside the emotion module the *valenced impulses* drive the concurrent simulation of the agent's emotion dynamics. In summary, a positive (negative) impulse has the short term effect of increasing (decreasing) the agent's *emotional valence*, which in turn influences the agent's *mood* in the same direction as a long term effect. A mathematical transformation into *pleasure* and *arousal* values enables WASABI to map the emotion dynamics on emotional states with intensities (see [6, 8] for details).

Triggering prospect-based emotions

In its default configuration, WASABI simulates the primary emotions *annoyed*, *angry*, *bored*, *concentrated*, *depressed*, *fearful*, *happy*, *sad*, and *surprised* as well as the secondary emotions *relief*, *fears-confirmed*, and *hope*. Five of these 12 emotions (*fearful*, *surprised*, *relief*, *fears-confirmed*, and *hope*) rely on an agent's ability to build expectations about future events, i.e., they are so-called

trigger	*if..*
fear	$e_{t-1} - e_t > \epsilon$
surprise	$\lvert e_{t-1} - e_t \rvert > \epsilon$
fears-confirmed	$fear_{t-1} \wedge (e_{t-1} - e_t < \epsilon)$
hope	$e_{t,d} - e_{t,d-2} > \epsilon$
relief	$fear_{t-1} \wedge (e_t - e_{t-2} < \epsilon)$

Table 1. The conditions under which the prospect-based emotions are triggered in WASABI based on the changes of evaluations over time with ϵ and depth d as custom parameters

Figure 7. The virtual agent expressing *anger*, *neutral*, and *joy* (left to right)

prospect-based emotions. For example, one is only surprised about an event, if it is contrary to one's previous expectations, or one fears future events, only if one has reason to expects that bad event to happen (see [8] for further details). Accordingly, in WASABI each of these emotions is configured with zero *base intensity* and they need to be *triggered* (cp. Fig. 6) to give them a chance to gain positive intensity.

An experienced, human chess player will evaluate the available moves for his opponent. He will probably realize that his last move was less good than previously evaluated, because now his evaluation reaches one level deeper into the search tree than before. Accordingly, he might start to fear that his opponent realizes her opportunity as well. If his evaluation of the situation after the opponent's move is stable then his fears are confirmed: the opponent made the right move. On the other hand if the evaluation comes back to what it was before, i.e., before the agent made his last move, then the opponent missed the opportunity and the agent is relieved. This evaluation can be in between this two values and in that case, the agent is neither relieved nor seeing his fear comes true but will still receive the negative impulse from the drop. Formally, Table 1 provides details on how the changing evaluations trigger prospect-based emotions in WASABI.

Notably, the value e_t represents the future directed evaluation of the situation from the robot's perspective. For example, the formula $e_{t-1} - e_t > \epsilon$ lets the main module trigger *fear* whenever a significant drop in the evaluation function appeared from the previous move to the current one. That is, MARCO realizes at time t that the future seems much worse than evaluated before (in time $t-1$). If subsequently, after the next half-move in $t+1$, the value e_{t-1} turns out to have been correct in the light of the new value e_t (or the situation got even worse than expected), then *fears-confirmed* will be triggered. On the contrary, if it turned out to be much better than expected, *relief* will be triggered. *Surprise* is always triggered when the evaluation changes significantly from one half-move to the next. Finally, *hope* is triggered whenever not taking the full depth of the search tree into account would mean that the key move in the position is hard to reach (requires a computation at depth at least d).[1]

Feeding back into the main module

It is important to note that, in addition to an emotion being triggered, the *pleasure*, *arousal*, and *dominance* (PAD) values driven by the emotion dynamics must be close enough to that emotion for it to become a member of the *emotion vector* with positive intensity. Thus, although *surprise* will always be triggered together with *fear*, they will not always both be present in the *emotion vector*, because they occupy different regions in PAD space.

From the *emotion vector* the emotion with the highest intensity is compiled into the BML description driving the MARC framework. The agent comments on particular events like, for example, complimenting the player after it lost a game or stating that the position is now simplified after exchanging the queen.

The virtual agent provided by the MARC framework

The MARC framework [10] is used to animate the virtual agent, which is displayed on the 5.6 inch pan-tilt display next to the robotic arm. The emotional facial expressions (see Fig. 7 for examples) that are provided by the main module as part of the BML description are combined inside the MARC framework to create lip-sync animations of emotional verbal utterances. Thanks to the integration of the open-source text-to-speech synthesis OpenMARY [17] the agent's emotion also influences the agent's auditory speech.

CONCLUSION

This paper introduced MARCO, a chess playing hybrid agent equipped with a robotic arm and a screen displaying a virtual agent capable of emotional facial expressions. The hardware comprises a DGT chessboard with a digital clock, a pan-tilt agent display, and a custom-built robotic arm. Our modular software architecture connects the MARC agent framework with the TSCP chess engine, the WASABI Affect Simulation architecture and the hardware components allowing MARCO to

[1] An evaluation function is usually set up to an even number, thus the last level of the search tree equals the last two half-moves.

play chess against human players autonomously. Furthermore, a detailed description of how emotions are simulated based on the chess engine's evaluation function was presented by which the MARC agent is made to express emotions dynamically while giving verbal comments regarding the game state at the same time.

The concrete behavior of the virtual agent still needs to be implemented. For example, we need to decide which kind of comments are to be given with which timing during the game and how virtual gaze and robotic head movements are to be combined to give the impression of a believable, hybrid agent.

In order to answer the initially stated two research questions, we plan to conduct an empirical study following a between subjects design. At first, one group of participants will play against MARCO with the pan-tilt display turned off. Nevertheless, the invisible agent's comments will remain audible in this condition. In the second condition, another group of participants will play against MARCO with an unemotional agent presented on the pan-tilt display. For the third condition, a group of participants will play against the WASABI-driven agent presented on the pan-tilt display. In all three conditions, player enjoyment will be assessed using the GameFlow [18] questionnaire and video recordings of the human players will be analyzed inspired by [16]. We expect to find significant differences between conditions with the most complete setup (condition three) being most fun for the players.

Nass and Moon claim that imperfect technologies mimicking human characteristics might even increase "the saliency of the computer's 'nonhumanness'." [15, p. 97] In line with their ideas and in addition to the approach outlined above, we plan to compare human-human interaction with human-agent interaction when competing in chess to measure and incessantly improve MARCO's level of human-likeness. This will help to understand how human behavior might be split into computationally tractable components and then realized in robotic agents to improve human-computer interaction.

REFERENCES

1. `http://aitopics.org/topic/chess`.
2. `http://www.tckerrigan.com/Chess/TSCP`.
3. `http://www.gnu.org/software/xboard/engine-intf.html`.
4. `http://www.trossenrobotics.com/widowxrobotarm`.
5. Qt: Cross-platform application and UI framework. http://qt-project.org/, May 2014.
6. removed for blind review
7. removed for blind review
8. removed for blind review
9. M. Campbell, A. H. Jr., and F. hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1 – 2):57 – 83, 2002.
10. M. Courgeon, J.-C. Martin, and C. Jacquemin. MARC: a Multimodal Affective and Reactive Character. In *Proc. 1st Workshop on AFFective Interaction in Natural Environements*, 2008.
11. A. E. Elo. *The rating of chessplayers, past and present*. Arco Pub., New York, 1978.
12. I. Leite, C. Martinho, A. Pereira, and A. Paiva. icat: an affective game buddy based on anticipatory mechanisms. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, AAMAS '08, pages 1229–1232, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
13. I. Leite, C. Martinho, A. Pereira, and A. Paiva. As time goes by: Long-term evaluation of social presence in robotic companions. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 669–674, Sept 2009.
14. C. Matuszek, B. Mayton, R. Aimi, M. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J. Smith, and D. Fox. Gambit: An autonomous chess-playing robotic system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4291–4297, May 2011.
15. C. Nass and Y. Moon. Machines and mindlessness: Social responses to computers. *Journal of Social Issues*, 56(1):81–103, 2000.
16. L. Sajó, Z. Ruttkay, and A. Fazekas. Turk-2, a multi-modal chess player. *International Journal of Human-Computer Studies*, 69(7–8):483–495, 2011.
17. M. Schröder. OpenMARY sources. https://github.com/marytts/marytts, April 2013.
18. P. Sweetser and P. Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.
19. H. Vilhjálmsson, N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. N. Marshall, C. Pelachaud, Z. Ruttkay, K. Thóisson, H. Welbergen, and R. J. Werf. The behavior markup language: Recent developments and challenges. In *Intelligent Virtual Agents*, volume 4722 of *LNCS*, pages 99–111. Springer Berlin Heidelberg, 2007.

Apéndice 7. Código para la implementación del algoritmo Fuzzy C-Means en MATLAB

```
clear all
close all

load datos_ajedrez.mat    %Cargar datos de las
    piezas de ajedrez

%Tamaño de cada clase
n1=length(alfil);
n2=length(caballo_cruzado);
n3=length(caballo);
n4=length(peon);
n5=length(reina);
n6=length(rey);
n7=length(torre);

%Número total de datos de entrenamiento (la
    mitad del total de muestras)
n=fix(n1/2)+fix(n2/2)+fix(n3/2)+fix(n4/2)+fix(
    n5/2)+fix(n6/2)+fix(n7/2);

%Datos de entrenaiento
```

```
datos=[alfil(1:fix(n1/2),:);caballo_cruzado(1:
    fix(n2/2),:);caballo(1:fix(n3/2),:);peon(1:
    fix(n4/2),:);reina(1:fix(n5/2),:);rey(1:fix(
    n6/2),:);torre(1:fix(n7/2),:)];

%Permutación de los datos de entrenamiento
permuta=randperm(length(datos));
data=datos;
for r=1:length(datos)
    data(r,:)=datos(permuta(r),:);
end
datos=data;

grupos=20;    %Número de grupos
m=2;     %Varianza
u=rand(n,grupos);    %Matriz inicial de grados
    de pertenencia

%Método de agrupamiento c-medias difuso
[centros,U1,error]=fcm_1(datos,grupos,m);

%Distancias euclidianas entre la matriz de cada
     prototipo y el alfil
distancias1=[];
for p=(fix(n1/2)+1):n1
    agrega1=[];
    for r=1:grupos
        agrega1=[agrega1 sqrt(sum((centros(r,:)
            -alfil(p,:)).^2))];
    end
    distancias1=[distancias1;agrega1];
end
[Y1,I1]=min(distancias1');

%Distancias euclidianas entre la matriz de cada
     prototipo y el caballo-cruzado
distancias2=[];
for p=(fix(n2/2)+1):n2
```

```
    agrega2=[];
    for r=1:grupos
        agrega2=[agrega2 sqrt(sum((centros(r,:)
            -caballo_cruzado(p,:)).^2))];
    end
    distancias2=[distancias2;agrega2];
end
[Y2,I2]=min(distancias2');

%Distancias euclidianas entre la matriz de cada
    prototipo y el caballo
distancias3=[];
for p=(fix(n3/2)+1):n3
    agrega3=[];
    for r=1:grupos
        agrega3=[agrega3 sqrt(sum((centros(r,:)
            -caballo(p,:)).^2))];
    end
    distancias3=[distancias3;agrega3];
end
[Y3,I3]=min(distancias3');

%Distancias euclidianas entre la matriz de cada
    prototipo y el peón
distancias4=[];
for p=(fix(n4/2)+1):n4
    agrega4=[];
    for r=1:grupos
        agrega4=[agrega4 sqrt(sum((centros(r,:)
            -peon(p,:)).^2))];
    end
    distancias4=[distancias4;agrega4];
end
[Y4,I4]=min(distancias4');

%Distancias euclidianas entre la matriz de cada
    prototipo y la reina
distancias5=[];
```

```
for p=(fix(n5/2)+1):n5
    agrega5=[];
    for r=1:grupos
        agrega5=[agrega5 sqrt(sum((centros(r,:)
            -reina(p,:)).^2))];
    end
    distancias5=[distancias5;agrega5];
end
[Y5,I5]=min(distancias5');

%Distancias euclidianas entre la matriz de cada
    prototipo y el rey
distancias6=[];
for p=(fix(n6/2)+1):n6
    agrega6=[];
    for r=1:grupos
        agrega6=[agrega6 sqrt(sum((centros(r,:)
            -rey(p,:)).^2))];
    end
    distancias6=[distancias6;agrega6];
end
[Y6,I6]=min(distancias6');

%Distancias euclidianas entre la matriz de cada
    prototipo y la torre
distancias7=[];
for p=(fix(n7/2)+1):n7
    agrega7=[];
    for r=1:grupos
        agrega7=[agrega7 sqrt(sum((centros(r,:)
            -torre(p,:)).^2))];
    end
    distancias7=[distancias7;agrega7];
end
[Y7,I7]=min(distancias7');

%Gráfica de los datos y la posición de cada
   prototipo
```

```
plot(caballo_cruzado(:,1),caballo_cruzado(:,2),
    '*','Color',[1,1,0])

hold on
plot(rey(:,1),rey(:,2),'xg')
plot(reina(:,1),reina(:,2),'hb')
plot(alfil(:,1),alfil(:,2),'sk')
plot(caballo(:,1),caballo(:,2),'dm')
plot(torre(:,1),torre(:,2),'vr')
plot(peon(:,1),peon(:,2),'+','Color',[.1,1,1])
plot(centros(:,1),centros(:,2),'ok','
    MarkerFaceColor',[0 0 0],'MarkerSize',5),
    title('Posición-tiempo'),legend('Caballo
    perpendicular','Rey','Reina','Alfil','
    Caballo','Torre','Peón')
hold off
min_xy=min(datos);
max_xy=max(datos);
axis([min_xy(1),max_xy(1),min_xy(2)-1,max_xy(2)
    +1])
axis([2 5 -20 70])
xlabel('Tiempo (s)')
ylabel('Posición angular (°)')
```

Funcion fcm1

```
function [v_new,U,error]=fcm1(X,grupos,m)
[n1,m1]=size(X);
U=rand(grupos,n1);
tolerancia=0.001;
maximo=100;
v_old=rand(grupos,m1);
d=rand(grupos,m1);
iteracion=0;
A=eye(m1,m1);
while iteracion<=maximo
    iteracion=iteracion+1;
```

```
%Centros
for r=1:grupos
    for p=1:m1
        v_new(r,p)=(sum((U(r,:).^m)'.*X(:,p
            ))))/(sum(U(r,:).^m));
    end
end
%U
%Considerando que A=I
%for k=1:n1
    %Distancias
 for r=1:grupos
      for q=1:n1
          d(r,q)=(X(q,:)-v_new(r,:))*(X(q
              ,:)-v_new(r,:))';
      end
 end

 for r=1:grupos
    for q=1:n1
         if d(r,q) ~= 0
            suma=0;
              for l=1:grupos
                   if d(l,q) ~= 0
                       suma=suma+(d(r,q)/d(
                           l,q))^(2/(m-1));
                    else
                        suma=suma;
                    end
                end
                U(r,q)=1/suma;
             else
                U(r,q)=0;
             end
         end
    end
error=max(max(abs(v_new-v_old)));
if  error <= tolerancia break,end
```

```
        v_old=v_new;
    end
```

Bibliografía

[1] G. Alonso. *Introducción a la inteligencia artificial.* Multimedia ediciones S.A., Barcelona, 1993.

[2] D. E. Guillén B. A. Galan and D. F. Tello. *Estudio, diseño y construcción de una pata para un robot cuadrúpedo escalador.* Cuenca, Ecuador, 2009.

[3] J. Knudsen B. Lin, B. Recker and S.B. Jorgensen. *A systematic approach for soft sensor development.* Computers and Chemical Engineering, 31(5), 419 - 425, 2007.

[4] G. Bastin and D. Dochain. *On-line estimation and adaptive control of bioreactors.* Elsevier, Amsterdam, 1990.

[5] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, U.K., 1995.

[6] L. Breiman. *Bagging predictors.* Machine Learning, 24 (2), pp. 123-140, 1996.

[7] J.F. MacGregor B.S. Dayal. *Recursive exponentially weighted PLS and its applications to adaptive control and prediction.* Journal of Process Control, 7 (3), pp. 169-179, 1997.

[8] I-B. Lee C. Lee, S.W. Choi. *Sensor fault identification based on time-lagged PCA in dynamic processes.* Chemometrics and Intelligent Laboratory Systems, 70 (2), pp. 165-178, 2004.

[9] A.M. Salvador Carlos. *Fundamentos y Aplicaciones de los Sensores Virtuales.* Centro de Investigación y Estudios Avanzados, Instituto Politécnico Nacional, 2008.

[10] S. Schaal C.G. Atkeson, A.W. Moore. *Locally weighted learning.* Artificial Intelligence Review, 11 (1), pp. 11-73, 1997.

[11] A. Chruy. *Software sensors in bioprocess engineering.* Journal of Biotechnologyr, 52(3):193-199, 1997.

[12] T.J. McAvoy D. Dong. *Nonlinear principal component analysis-based on principal curves and neural networks.* Computers and Chemical Engineering, 20 (1), pp. 65-78, 1996.

[13] L. Davies and U. Gather. *The identification of multiple outliers.* Journal of the American Statistical Association, 88(423), 782-792, 1993.

[14] P.J. Denning. *Origin of Virtual Machines and Other Virtualities.* IEEE Annals of the History of Computing 23(3), 73, 2001.

[15] L. Chiang E. Jordaan, A. Kordon and G. Smits. *Robust Inferential Sensors Based on Ensemble of Predictors Generated by Genetic Programming.* In Proceedings of International Conference on Parallel Problem Solving from Nature, paginas 522-531, Birmingham, UK, 2004.

[16] M. Barolo E. Zamprogna and D.E. Seborg. *Development of a soft sensor for a batch distillation column using linear and nonlinear PLS regression techniques.* Control Engineering Practice, 12, 917-929, 2004.

[17] L. Fortuna. *Soft Sensors for Monitoring and Control Industrial Processes.* Springer, Londres, Inglaterra, 2007.

[18] K.R Foster. *Software tools.* IEEE Spectrum 53, 1: 52-56.

[19] K. Funahashi. *On the approximate realization of continuous mapping by neural networks.* Neural Networks, 2(3), 183-192, 1989.

[20] L. A. Geddes and L. E. Baker. *Principles of Applied Biomedical Instrumentation.* 3rd Edition, John Wiley and Sons, 1989.

[21] H. Goldberg. *What Is Virtual Instrumentation?* IEEE Instrumentation and Measurement Magazine 3(4), 10-13, 2000.

[22] G.D. Gonzalez. *Soft sensors for processing plants.* Proceedings of the second international conference on intelligent processing and manufacturing of materials, 1999.

[23] W.S. Gosset. *The probable error of a mean.* Biometrika, 6 (1), pp. 1 - 25, 1908.

[24] P. Fingar H. Smith. *Business process management: The third wave.* Meghan-Kiffer Press, Tampa, USA, 2003.

[25] C. Han and Y. H. Lee. *Intelligent integrated plant operation system for Six Sigma.* Annual Reviews in Control, 26(1):27 - 43, 2002.

[26] H. Hotelling. *The generalization of student´s ratio.* The Annals of Mathematical Statistics, 2(3), pp. 360-378, 1931.

[27] A. Elisseeff I. Guyon. *An introduction to variable and feature selection.* Journal of Machine Learning Research, 3 (7-8), pp. 1157-1182, 2003.

[28] W Full J.C. Bezdek, R. Ehrlich. *FCM: The Fuzzy c-Means Clustering Algorithm.* Computers and Geosciences 10 (2-3): pp. 191-203., 1984.

[29] G.S. Mudholkar J.E. Jackson. *Control procedures for residuals associated with principal component analysis.* Technometrics, 21 (3), pp. 341-349, 1979.

[30] P.X. Zhou J.J. Macias. *A method for predicting quality of the crude oil distillation.* Proceedings of the 2006 international symposium on evolving fuzzy systems, pp. 214-220, 2006.

[31] I. T. Jolliffe. *Principal Component Analysis.* Springer, New York, 2002.

[32] S. Rezvani L. Maguire K. Warne, G. Prasad. *Statistical and computational intelligence techniques for inferential model development: A comparative evaluation and a novel proposition for fusion.* Engineering Applications of Artificial Intelligence, 17 (8), pp. 871-885, 2004.

[33] P. Kadlec and B. Gabrys. *Adaptive Local Learning Soft Sensor for Inferential Control Support.* In Proceedings of International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA 2008), Vienna, Austria, 2008. IEEE.

[34] R. Kohavi. *A study of cross-validation and bootstrap for accuracy estimation and model selection.* Proceedings of the fourteenth international joint conference on artificial intelligence, Vol. 2, pp. 1137-1145, 1995.

[35] M.J. Lopez-Nieto Y. A. Dimitriadis M. J. Arauzo-Bravo. J. M. Cano-Izquierdo, E. Gomez-Sanchez and J. Lopez-Coronado. *Automatization of a penicillin production process with soft sensors and adaptive controller based on neuro fuzzy systems.* Control Engineering Practice, 12(9), 1073-1090, 2004.

[36] T. G. Manuel. *Análisis de componentes principales.* UOC.

[37] D. Miranda. *Instrumentación electrónica, sensores y control inteligente.* Centro de Física Aplicada y Tecnología Avanzada de la UNAM, 2010.

[38] A. Linde M. Larsson N. Wickstrom, M. Traveniku and B. Svensson. *Estimating pressure peak position and air-fuel ratio using the ionization current and artificial neural networks.* IEEE, Center for Computer Systems Architecture (CCA) Halmstad, Sweden, 1998.

[39] F. Nebeker. *Golden Accomplishments in Biomedical Engineering.* IEEE Engineering in Medicine and Biology Magazine 21(3), 17-47, 2002.

[40] K. Esbensen P. Geladi. *Regression on multivariate images: Principal component regression for modeling, prediction and visual diagnostic tools.* Journal of Chemometrics, 5 (97), p. 111, 1991.

[41] B. Gabrys P. Kadlec and S. Strandt. *Data-driven soft sensor in the process industry.* Computers and chemical Engineering, 33(4), 795-814, 2009.

[42] Yong-Lae Park. *Bio-Inspired Soft Robotics: New Ways of Sensing, Actuation, and Integration.* Robotics Institute and the School of Computer Science, Carnegie Mellon University, 2014.

[43] R. K. Pearson. *Exploring process datas.* Journal of Process Control 11(2), 179-194, 2001.

[44] R. K. Pearson. *Outliers in process modeling and identification.* IEEE Transactions on Control Systems Technology, 10(1), 55-63, 2002.

[45] P. Perner. *Machine Learning and Data Mining in Pattern Recognition.* LNAI 5632, Pages 337-346, 2009.

[46] Sibylle Strandt Petr Kadlec, Bogdan Gabrys. *Data-driven Soft Sensors in the process industry.* Elsevier, Pages 795-814, 2009.

[47] S. J. Qin. *Neural networks for intelligent sensors and control - Practical issues and some solutions.* Neural Systems for Control, Academic Press, Chapter 8, 213 - 234, San Diego, CA, 1997.

[48] S.J. Qin. *Recursive PLS algorithms for adaptive data modeling.* Computers and Chemical Engineering, 22 (4-5), pp. 503-514, 1998.

[49] S.J. Qin R. Dunia. *Joint diagnosis of process and sensor faults using principal component analysis.* Control Engineering Practice, 6 (4), pp. 457-469, 1998.

[50] S.J. Qin R. Dunia. *Subspace approach to multidimensional identification and reconstruction.* AIChE Journal, 44 (8), p. 1813, 1998.

[51] H. Yue S. J. Qin and R. Dunia. *Self-validating inferential sensors with application to air emission monitoring.* Industrial and Engineering Chemistry Research, 36:1675-1685, 1997.

[52] C. Han S. Park. *A nonlinear soft sensor based on multivariate smoothing procedure for quality estimation in distillation columns.* Computers and Chemical Engineering, 24 (2-7), pp. 871-877, 2000.

[53] F. Xiao S. Wang. *AHU sensor fault diagnosis using principal component analysis method.* Energy and Buildings, 36 (2), pp. 147-160, 2004.

[54] J. Cui S. Wang. *Sensor-fault detection, diagnosis and estimation for centrifugal chiller systems using principal-component analysis method.* Applied Energy, 82 (3), pp. 197-213, 2005.

[55] C. Kulikowski S. Weiss. *Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[56] M. Santori. *An instrument that isn't really.* IEEE Spectrum 27(8), 36-39, 1990.

[57] J. Scheffer. *Dealing with missing data.* Research Letters in the Information and Mathematical Sciences, 3(1), 153-160, 2002.

[58] S. Serneels and T. Verdonck. *Principal component analysis for data containing outliers and missing elements.* Computational Statistics and Data Analysis, 52(3), 1712-1727, San Diego, CA, 2008.

[59] A.J. Morris I.B. Lee S.W. Choi, E.B. Martin. *Adaptive multivariate statistical process control for monitoring time-varying processes.* Industrial and Engineering Chemistry Research, 45, pp. 3108-3118, 2006.

[60] J. Friedman T. Hastie, R. Tibshirani. *The elements of statistical learning: Data mining, inference, and prediction.* Springer, 2001.

[61] X. He P. Stuart T. Jiang, B. Chen. *Application of steady-state detection method based on wavelet transform.* Computers and Chemical Engineering, 27 (4), pp. 569-578, 2003.

[62] L. P. Russo V. Prasad, M. Schley and B. Wayne Bequette. *Product property and production rate control of styrene polymerization.* Journal of Process Control, 12(3):353-372, 2002.

[63] S. Valle-Cervantes S.J. Qin W. Li, H.H. Yue. *Recursive PCA for adaptive process monitoring.* Journal of Process Control, 10 (5), pp. 471-486, 2000.

[64] H. Shao W. Yan and X. Wang. *Soft sensing modeling based on support vector machine and Bayesian model selection.* Computers and chemical Engineering, 28(8), 1489-1498, 2004.

[65] B. Walczak and D. L. Massart. *Robust principal components regression as a detection tool for outliers.* Chemometrics and Intelligent Laboratory Systems, 27(1), 41 - 54, 1995.

[66] B. Walczak and D. L. Massart. *Dealing with missing data: Part ii.* Chemometrics and Intelligent Laboratory Systems, 58(1), 29-42, 2001.

[67] G. Welch and G. Bishop. *An Introduction to the Kalman Filter. Technical Report Technical Report TR 95-041.* University of North Carolina-Department of Computer Science, 1995.

[68] P.J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD dissertation, 1974.

[69] H. Wold. *Nonlinear estimation by iterative least squares procedures, in Research Papers in Statistics*. Festschrift for J. Neyman, F. David (eds.), pp. 411-444, Wiley, 1966.

[70] G.W. Irwin X. Wang, U. Kruger. *Process monitoring approach using fast moving window PCA*. Industrial and Engineering Chemistry Research, 44 (15), pp. 5691-5702, 2005.

[71] R.E. Schapire Y. Freund. *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55 (1), pp. 119-139, 1997.

[72] L. Zhao and T. Chai. *Adaptive moving window MPCA for online batch monitoring*. In Proceedings of the 5th Asian control conference, 2004.