UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN
MENCIÓN EN SISTEMAS DE TELECOMUNICACIÓN

SISTEMA PARA LA DETECCIÓN DEL ESTADO DE FUNCIONAMIENTO DE LOS ELEMENTOS ROTANTES DE UNA COSECHADORA A PARTIR DEL PROCESADO DE VIBRACIONES: ANÁLISIS DE LAS FRECUENCIAS MÁS RELEVANTES MEDIANTE ALGORITMOS EVOLUTIVOS

Autora:
Dña. Lidia Martínez Martínez

Tutores:
D. Rubén Ruiz González
Dr. D. Jaime Gómez Gil

Valladolid, 11 de septiembre de 2019

TÍTULO:	**Sistema para la detección del estado de funcionamiento de los elementos rotantes de una cosechadora a partir del procesado de vibraciones: análisis de las frecuencias más relevantes mediante algoritmos evolutivos**
AUTORA:	**Dña. Lidia Martínez Martínez**
TUTORES:	**D. Rubén Ruiz González** **Dr. D. Jaime Gómez Gil**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Alonso Alonso Alonso**
VOCAL:	**Dr. D. Jaime Gómez Gil**
SECRETARIO:	**Dr. D. Javier Manuel Aguiar Pérez**
FECHA:	**11 de septiembre de 2019**
CALIFICACIÓN:	

Resumen

En este Trabajo Fin de Grado, se ha diseñado un sistema que permite la detección del estado de funcionamiento de elementos rotantes de una cosechadora, como el motor, la trilla y el picador, a partir de señales de vibración procedentes de cuatro acelerómetros situados en distintos puntos del chasis de la cosechadora. Este sistema realiza la detección empleando 4 etapas: (*i*) la adquisición de datos de los sensores de vibración; (*ii*) el preprocesado de las señales adquiridas; (*iii*) una reducción de la dimensionalidad seleccionando las mejores frecuencias mediante el algoritmo *Harmony Search* (HS); y (*iv*) la estimación del estado de los elementos rotantes empleando el clasificador de los *k* vecinos más cercanos (kNN). Además, se incluyó un bloque de suavizado y otro de preselección de frecuencias para estudiar su influencia en el sistema. Los resultados obtenidos evidencian que, en sistemas para la detección de elementos rotantes de máquinas como el desarrollado en este trabajo: (*i*) un suavizado de las señales adquiridas y una preselección de las frecuencias mejora la precisión del sistema, que en las pruebas realizadas llegó a ser del 100% empleando tan solo 3 frecuencias; (*ii*) el algoritmo *Harmony Search* ayuda a mantener las precisiones sin requerir un tiempo prohibitivo, concretamente, los tiempos de ejecución disminuyeron hasta en un 90% empleando este algoritmo; y (*iii*) los acelerómetros pueden ser colocados en distintas partes de la máquina, en nuestros experimentos se obtuvieron resultados similares para diferentes ubicaciones. Para finalizar, otra aportación de este trabajo es que el rendimiento y la eficiencia del algoritmo *Harmony Search* mejoran al introducir una etapa de reducción de la dimensionalidad**.** Debido al gran número de frecuencias a tener en consideración, al incluir esta etapa, se consigue reducir la complejidad del sistema.

Palabras clave:

Algoritmo evolutivo, Fuerza Bruta (FB), *Harmony Search* (HS), *k* vecinos más cercanos (kNN), mantenimiento predictivo, reducción de la dimensionalidad.

Abstract

In this Final Degree Project, a system has been designed allowing the detection of the operating status of rotating elements of an agricultural harvester, such as the engine, the threshing and the chopper, from vibration signals obtained from four accelerometers located at different points on the harvester's chassis. This system performs the detection using 4 stages: (*i*) the acquisition of data from the vibration sensors; (*ii*) the preprocessing of the acquired signals; (*iii*) the dimensionality reduction by selecting the best frequencies using the Harmony Search (HS) algorithm; and (*iv*) the estimation of the operating status of the rotating elements using the *k*-Nearest Neighbors (kNN) classifier. In addition, a smoothing block and a frequency preselection block were included to study their influence on the system. The results obtained prove that, in systems for the detection of rotating elements of machines such as the one developed in this project: (*i*) a smoothing of the acquired signals and a preselection of the frequencies improves the accuracy of the system, which in the tests performed reached 100% using only 3 frequencies; (*ii*) the Harmony Search algorithm helps to maintain the accuracy without requiring prohibitive execution times, specifically, the times decreased by up to 90% using this algorithm; and (*iii*) accelerometers can be placed in several locations of the machine, due to the similar results that were obtained in our experiments when testing different locations. Finally, another contribution of this work is that the performance and efficiency of the Harmony Search algorithm are improved by introducing the stage of dimensionality reduction. Due to the larger number of frequencies to be considered, by including this stage the complexity of the system is reduced.

Keywords:

Evolutionary algorithm, Brute Force, Harmony Search (HS), *k*-Nearest Neighbors (kNN), predictive maintenance, dimensionality reduction.

Agradecimientos

En primer lugar, quiero dar las gracias a Jaime por ofrecerme la oportunidad de realizar este trabajo y por intentar facilitarme siempre las cosas. Tampoco me quiero olvidar de Rubén. Gracias por guiarme y ayudarme durante todo este proceso y por hacer que los días en el laboratorio fueran más amenos. Sin ti no hubiera sido lo mismo.

A todas las personas que he conocido durante esta etapa y que han hecho que la vida lejos de casa fuera más fácil, tanto a mis compañeros de teleco como a mis amigas de la resi. Espero que estas nuevas amistades se mantengan en el futuro. También quiero acordarme de mis amigas de León, porque a pesar de la distancia hemos seguido disfrutando de nuestros pequeños momentos.

Por último, quiero agradecer a mis padres por su cariño y apoyo, y en especial a mi hermano por ayudarme, aconsejarme, animarme y entenderme en todo momento. Sin vosotros todo esto no habría sido posible.

Muchas gracias a todos.

Acrónimos

ANN	*Artificial Neural Network*
ABC	*Artificial Bee Colony algorithm*
ACO	*Ant Colony Optimization*
DFT	*Discrete Fourier Transform*
EA	*Evolutionary Algorithm*
EP	*Evolutionary Programming*
ES	*Evolution Strategy*
FB	*Fuerza Bruta*
FFT	*Fast Fourier Transform*
GA	*Genetic Algorithm*
GP	*Genetic Programming*
HS	*Harmony Search*
HM	*Harmony Memory*
HMCR	*Harmony Memory Considering Rate*
kNN	*k-Nearest Neighbors*
PAR	*Pitch Adjusting Rate*
PSO	*Particle Swarm Optimization*
SI	*Swarm Intelligence*
SVM	*Support Vector Machine*

Índice abreviado

Índice general

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1 Ámbito del proyecto

El mantenimiento de una cosechadora consiste en las operaciones y cuidados necesarios para que ésta pueda seguir funcionando correctamente. Este mantenimiento puede ser de dos tipos:

- **Mantenimiento correctivo**, que consiste en reparar un elemento dañado o sustituirlo una vez que ha dejado de funcionar.
- **Mantenimiento preventivo**, en el cual se realizan revisiones y seguimientos del equipo con el fin de evitar daños o roturas en las piezas y alargar la vida útil del equipo. Además, el mantenimiento preventivo puede ser:
    - Periódico, es decir, la sustitución de cada componente se realiza trascurrido cierto tiempo, en base al kilometraje realizado o a las horas de trabajo.
    - Predictivo, que consiste en intentar determinar el momento idóneo en el que se deben de realizar las reparaciones o sustituciones antes de que la pieza se haya dañado o haya dejado de funcionar, lo cual se consigue mediante medidas que se toman a partir de las señales de vibración que produce la cosechadora.

Por otro lado, la monitorización de una cosechadora consiste en detectar posibles anomalías en los componentes que la conforman a partir de la observación de señales. Ésta se puede realizar con cables y sensores específicos para cada elemento que se monitoriza, como puede ser la velocidad de un eje. Otra forma sería mediante el procesado de las señales de vibración, lo que supone un avance respecto al método anterior, ya que además de eliminar cables y sensores se pueden monitorizar varios elementos a la vez.

Teniendo en cuenta las ventajas que ofrece el mantenimiento predictivo, una alternativa viable es diseñar un sistema que determine el estado de funcionamiento de los elementos rotantes (motor, trilla y picador) a partir de la monitorización de las señales de vibración producidas por la cosechadora. De esta forma, se consigue mejorar el funcionamiento y la eficiencia de los componentes de una cosechadora y reducir los costes.

1.2 Objetivos

El principal objetivo de este Trabajo Fin de Grado es diseñar y evaluar un sistema para la detección del estado de funcionamiento de los elementos rotantes de una cosechadora empleando señales de vibraciones. Este sistema tiene que ser capaz de determinar si el motor, la trilla o el picador están encendidos o apagados y, además, en el caso de la trilla y el picador, si están equilibrados o desequilibrados cuando están encendidos.

Adicionalmente, para alcanzar el objetivo inicial, se plantean los siguientes objetivos secundarios en este proyecto:

- Desarrollar un procedimiento que seleccione las frecuencias de mayor interés para cada problema.
- Ver las mejoras que se producen en el sistema cuando se utilizan algoritmos evolutivos en el mismo, ya que el propósito que se busca es aumentar la eficiencia y rapidez en la detección.
- Estudiar la influencia de la preselección y del suavizado en la selección de las frecuencias.
- Comparar la eficiencia y tiempos de ejecución entre un algoritmo evolutivo y algoritmos basados en fuerza bruta.
- Determinar si la posición del acelerómetro tiene alguna influencia en la eficiencia y precisión del esquema de procesado.

1.3 Fases y métodos

Para que el desarrollo del proyecto fuera más sencillo y eficiente se organizó en las distintas fases que se van a describir a continuación:

- Estudio de trabajos previos para adquirir los conocimientos necesarios a la hora de llevar a cabo el sistema propuesto.
- Estudio teórico de las diferentes técnicas de algoritmos evolutivos, así como de los distintos clasificadores.
- Adquisición de las señales de vibración mediante los acelerómetros situados en la cosechadora.
- Pruebas previas con distintos algoritmos evolutivos y clasificadores para elegir el que mejor se adaptaba a nuestro sistema.
- Desarrollo del sistema:
    - Programación en MATLAB® de las partes auxiliares del sistema (preprocesado, clasificador y algoritmo evolutivo).
    - Programación del sistema completo, unificando todas las partes del mismo.
- Evaluación del sistema:
    - Ejecución del sistema completo para los datos adquiridos.
    - Análisis de los resultados y obtención de las conclusiones del proyecto.
- Identificación de las líneas futuras de mejora en el sistema.
- Redacción de la memoria del Trabajo Fin de Grado.

1.4 Medios disponibles

Durante la realización del Trabajo Fin de Grado se han utilizado los siguientes medios:

- Una cosechadora New Holland TC56 con 4050 horas de trabajo.
- Cuatro acelerómetros piezoelétricos uniaxiales Brüel & Kjær 4507-B-006, situados en distintos puntos de la cosechadora para medir las señales de vibración procedentes de la cosechadora.
- Un módulo NI 9234 de adquisición de datos para señales de entrada analógicas de la compañía National Instruments (NI).
- Un chasis NI cDAD-9172 de adquisición de datos compacto para conectar el módulo del sistema de adquisición de datos a un ordenador de la compañía National Instruments.
- *Software NI Sound and Vibration Assistant*, usado para adquirir las señales de vibración de la empresa National Instruments.
- Un ordenador portátil Asus K72J, con 6 GB de memoria RAM, 118 GB de disco duro SSD y procesador M350 a 2.27 GHz. Se ha utilizado a lo largo de todo el proceso de desarrollo del proyecto, así como para la confección de la memoria.
- *Software* MATLAB®, el programa de cómputo numérico desarrollado por MathWorks, empleado para el procesamiento de las señales de vibración.

1.5 Organización de la memoria

La memoria de este Trabajo Fin de Grado se ha estructurado de la siguiente forma:

- En el presente capítulo, Capítulo 1, se detalla el ámbito del proyecto, es decir, se explica la problemática que origina el trabajo de investigación, y se realiza una presentación de las herramientas utilizadas, las fases y métodos seguidos y los objetivos que se persiguen alcanzar.
- El Capítulo 2 constituye un marco teórico necesario para poder entender y desarrollar el sistema planteado. Primero, se realiza una introducción teórica a los algoritmos evolutivos y a sus principales paradigmas. A continuación, se introduce la inteligencia de enjambre, así como sus principales algoritmos. Finalmente, se explicarán en profundidad los algoritmos empleados en el sistema desarrollado.
- En el Capítulo 3, se describe el diseño y desarrollo del sistema. En él se pone en contexto la necesidad de la realización del sistema propuesto y se detallan los materiales y métodos utilizados, describiendo cada etapa llevada a cabo: adquisición de datos, preprocesado, reducción de la dimensionalidad y estimación del estado de los elementos rotantes.

- En el Capítulo 4, se evalúa la eficacia del sistema desarrollado a partir de los resultados obtenidos. Para ello se analizan y discuten los siguientes casos: estudio de la influencia de la preselección, estudio de la influencia del suavizado, estudio de la influencia del algoritmo *Harmony Search* y estudio de la influencia de la ubicación del acelerómetro utilizado. Para finalizar se realiza una comparativa con los resultados obtenidos de un trabajo similar existente.
- Por último, en el Capítulo 5, se detallan las conclusiones obtenidas tras el desarrollo del Trabajo Fin de Grado y se proponen posibles líneas futuras de investigación.
- Además, como material adicional se incluyen los códigos del sistema desarrollado en el Anexo I.

Capítulo 2: Marco teórico

En el presente capítulo se realizará una introducción a los algoritmos evolutivos y se describirán sus principales paradigmas: estrategias evolutivas, programación evolutiva, programación genética y algoritmos genéticos. A continuación, se introducirá la inteligencia de enjambre y sus principales algoritmos y se finalizará con la explicación del funcionamiento del algoritmo *Harmony Search* (HS) y el clasificador kNN (*k-Nearest Neighbors*), que serán dos métodos que se utilizarán en el sistema a desarrollar.

2.1 Algoritmos evolutivos

Los algoritmos evolutivos (*Evolutionary Algorithm*, EA), son métodos o técnicas que basan su funcionamiento en la simulación de los procesos evolutivos de la naturaleza y son muy utilizados en problemas de optimización. Algunas de las ventajas de estos algoritmos frente a otros métodos clásicos son:

- El espacio de búsqueda sobre el que trabajan es grande.
- Con una única ejecución del algoritmo se pueden obtener varias soluciones alternativas.
- La búsqueda de soluciones es más rápida debido a que los EAs permiten la incorporación de conocimiento del entorno del problema.
- Se consiguen evitar los óptimos locales.

Los EAs han demostrado que son un mecanismo de búsqueda de soluciones robusto y potente, debido a las ventajas anteriormente comentadas y a que mediante la recombinación son capaces de explotar las similitudes entre las distintas soluciones. Todo ello, hace que los EAs sean adecuados para problemas con múltiples soluciones, es decir, para la optimización multiobjetivo [1].

Hay distintos tipos de EAs, pero todos ellos se basan en la simulación de la evolución de una población a partir de unas reglas de selección y haciendo uso de los operadores de mutación y recombinación. Inicialmente, se parte de un conjunto de individuos que serán las soluciones candidatas, y en base a una función objetivo o *fitness*, se seleccionan a aquellos individuos que minimicen la función objetivo. Una vez que se han seleccionado los mejores individuos, con los operadores de recombinación y mutación se crearán otros nuevos. El operador de recombinación o cruce se encarga de intercambiar la información genética de los individuos padre, dando lugar a los nuevos individuos a los que se les denominará descendientes. Por otro lado, el operador de mutación modifica la información genética de los nuevos individuos. Repitiendo este proceso un número fijo de veces o hasta cumplir un determinado objetivo, la finalidad que se pretende conseguir es incrementar la calidad media de la población [1], [2].

Normalmente los EAs inicializan su población aleatoriamente, aunque en otros casos como valores iniciales se emplea una aproximación a la solución. A continuación, se evalúa a cada individuo con la función objetivo que se ha definido y se realiza la selección. La selección se realiza en dos pasos, primeramente, se seleccionan a los padres y se cruzan entre ellos dando lugar a los hijos que se añadirán a la población, posteriormente a través de la mutación se pueden modificar a los hijos. Una vez aplicados los operadores de recombinación y mutación se evalúan a los hijos y finalmente se seleccionan aquellos individuos que deben sobrevivir en la población. Además, los peores individuos se pueden eliminar para irlos reemplazando por los nuevos [1], [2]. Todo este proceso se puede observar, de forma más esquemática, en la Figura 1.

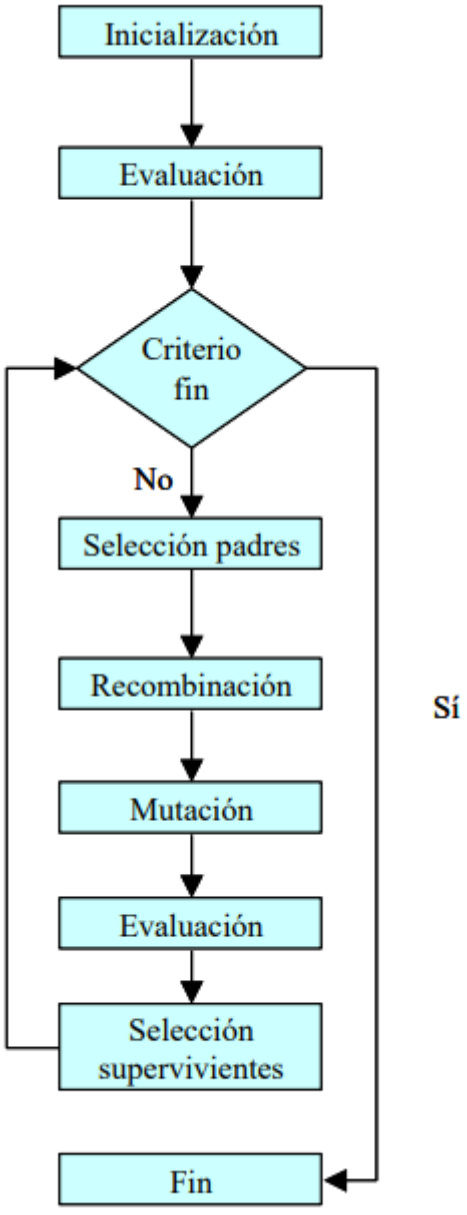

Figura 1. Diagrama de flujo del funcionamiento de un algoritmo evolutivo [1].

Los principales paradigmas o algoritmos heurísticos de los EAs son: estrategias evolutivas, programación evolutiva, algoritmos genéticos y programación genética. En los siguientes apartados se explicará detalladamente el funcionamiento de cada uno de ellos.

2.1.1 Estrategias evolutivas

Las estrategias evolutivas (*Evolution Strategy*, ES) fueron desarrolladas por Rechenberg. Éstas utilizaban un tamaño de población igual a uno y hacían uso de la selección y mutación. Más tarde, Schewefel al trabajar con poblaciones mayores introdujo la recombinación. Además, comparó las estrategias evolutivas con las técnicas de optimización tradicionales. La gran ventaja de las ESs es que están pensadas para trabajar sobre espacios continuos, esto quiere decir que su codificación se hace sobre vectores de números reales [1].

2.1.2 Programación evolutiva

La programación evolutiva (*Evolutionary Programming*, EP) fue desarrollada por Lawrence Fogel con el propósito de resolver tareas de predicción de la evolución de las máquinas de estados finitos. En estas máquinas, las tablas de transición de estado se modifican mediante mutaciones aleatorias uniformes en el alfabeto correspondiente. Los operadores principales que utiliza la programación evolutiva son la selección y la mutación. Estas técnicas se orientaron hacia la evolución de la inteligencia artificial, para que desarrollaran la habilidad de predecir cambios en el entorno [3], [4].

2.1.3 Algoritmos genéticos

Una clase de algoritmo evolutivo son los algoritmos genéticos (*Genetic Algorithm*, GA), los cuales son muy efectivos en la optimización de procesos no lineales. Además, para su diseño es suficiente con tener *a priori* un mínimo conocimiento del sistema y pueden aplicarse con éxito a un amplio espectro de problemas [2].

Los algoritmos genéticos fueron desarrollados por Holland en los años 70 mientras investigaba sobre la forma de aplicar al campo de la ciencia computacional las teorías de la evolución. Su idea se fundamentaba en usar poblaciones de individuos que representan soluciones a ciertos problemas mediante cadenas de ceros y unos. A partir de estas poblaciones, y a través de técnicas que simulaban la selección natural, como el sobrecruzamiento y la mutación, se buscaba mejorar las soluciones obtenidas [5].

Un GA empieza con una población de cromosomas, que se generan de manera aleatoria, y éstos van mejorando gracias a la aplicación de los operadores genéticos. La evolución de la población se produce a través de la selección natural. Para calcular el valor de adaptación de un cromosoma, se hace uso de una función de adaptación o evaluación que devuelve un valor numérico proporcional a la utilidad o adaptación de la solución que representa. En todas las iteraciones, llamadas generaciones, se evalúa la adecuación de los cromosomas como soluciones y, en base a esta evaluación, se forma una nueva población de cromosomas a partir del mecanismo de selección y de los operadores genéticos de cruce y mutación [6]. Este

proceso se puede observar tanto en la Figura 2, de manera gráfica, como en la Figura 3, en forma de diagrama de flujo.

- El **operador selección** se encarga de elegir a los individuos que se van a reproducir. Una vez que se han aplicado los operadores de cruce y mutación, el conjunto de individuos seleccionados engendrarán una nueva generación [1].
- El **operador de cruce** se encarga de la recombinación del material genético de los cromosomas padres, es decir, combina los genes de dos cromosomas padres para crear uno o dos cromosomas hijos. Debido a esto, al operador de cruce también se le denomina operador de recombinación [6].
- El **operador de mutación** altera de manera aleatoria uno o más genes de un cromosoma con la finalidad de que la variabilidad estructural de la población sea mayor. El papel de este operador en los algoritmos genéticos es restaurar material genético perdido o introducir material genético no explorado en la población. De este modo, se asegura de que la probabilidad de alcanzar cualquier punto del espacio de búsqueda nunca sea nula [6].

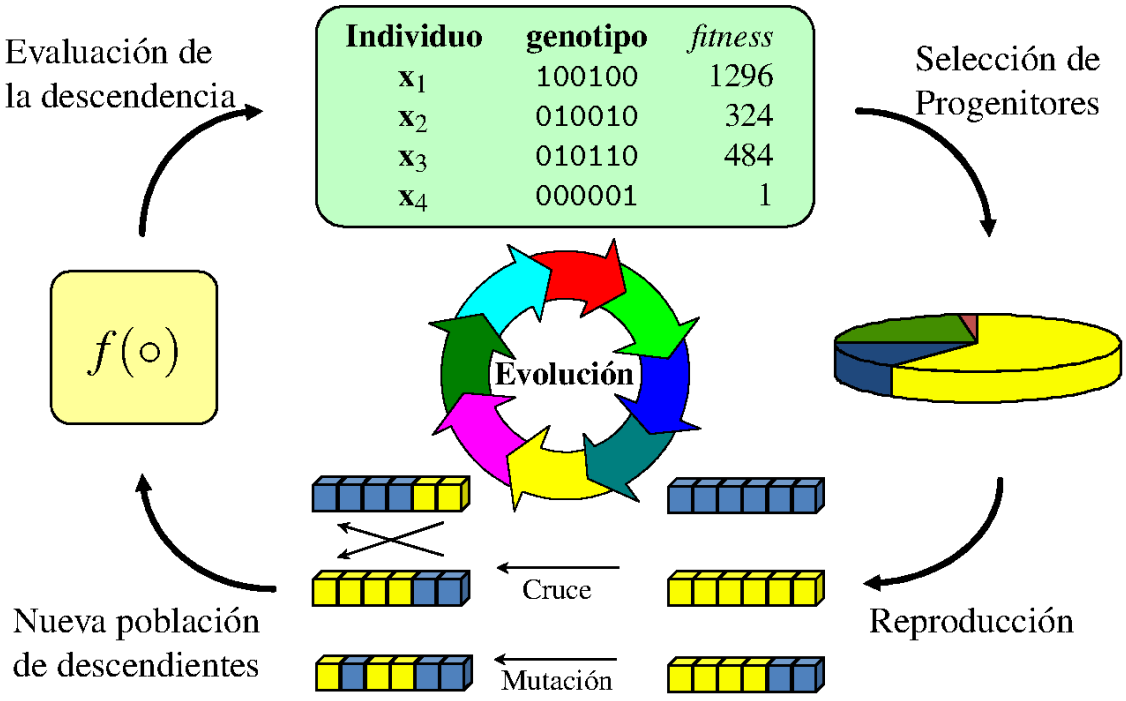

Figura 2. Representación gráfica de un algoritmo genético [7].

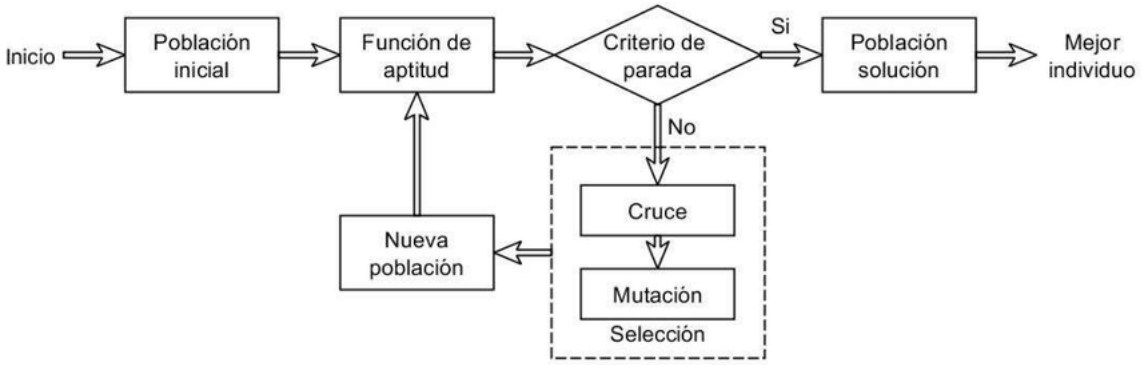

Figura 3. Ejemplo de diagrama de flujo del funcionamiento de un algoritmo genético [8].

Una de las principales ventajas que tienen los GAs es su capacidad para explotar la información almacenada sobre un espacio de búsqueda y, de esta forma, dirigir las sucesivas búsquedas hacia los mejores subespacios. Debido a esto, los GAs se aplican sobre espacios grandes, complejos y parcialmente definidos donde las técnicas de búsqueda clásicas no son adecuadas [6].

Por el contrario, existe una gran desventaja, que es el problema de la convergencia prematura hacia zonas del espacio de búsqueda que no contienen el óptimo global. Esto ocurre porque la base del funcionamiento de estos algoritmos radica en el mantenimiento del equilibrio entre explotar lo que es mejor actualmente a través del cruce, y explorar posibilidades que pueden convertirse en algo mucho mejor mediante la mutación. Cuando este equilibrio no es proporcional, se llega a la pérdida de diversidad de la población y, por tanto, se converge prematuramente [6].

2.1.4 Programación genética

La programación genética (*Genetic Programming*, GP) fue desarrollada por John Koza, el cual sugería que el programa deseado debía de evolucionar por sí mismo durante el proceso de ejecución. Esto quiere decir que se debería buscar el mejor programa dentro del espacio de posibles programas y la programación genética es el medio para hacer esta búsqueda. Al usar la GP para un problema existen tres pasos principales a seguir: selección de terminales y de una función, identificación de la función de evaluación y selección de los parámetros del sistema y de la condición de finalización [4].

2.2 Inteligencia de enjambre

La inteligencia de enjambre (*Swarm Intelligence*, SI) fue desarrollada en 1989 por Gerardo Beni y Jin Wang. Se trata de una rama de la inteligencia artificial que está inspirada en sistemas biológicos en los que pueden observarse comportamientos colectivos. Es un grupo de técnicas que se basan en el estudio del comportamiento colectivo en sistemas autoorganizados y descentralizados, los cuales normalmente están compuestos por una población de agentes computacionales simples capaces de percibir y modificar su ambiente

de forma local. Las interacciones locales entre los agentes habitualmente llevan a la emergencia de un comportamiento global. Este tipo de comportamiento se encuentra en la naturaleza: concretamente, se puede observar en algunos grupos de animales como las bandadas de aves, las colonias de hormigas, los enjambres de abejas, etc. [9], [10].

La auto-organización es una de las características de los comportamientos en la naturaleza. Ésta requiere de interacciones entre individuos, cuando uno de ellos modifica el entorno, este cambio afecta tanto a su propio comportamiento como al del resto de la colonia. A este mecanismo se le conoce como estigmergia y principalmente es utilizado por los insectos. Otra de las características, aparte de la interacción entre individuos, es el hecho de que las decisiones que tomen los individuos conserven una cierta aleatoriedad. De esa forma se amplía la capacidad de exploración y es posible descubrir alternativas nuevas que, de seguir la estrategia habitual, no sería posible encontrar [10].

Algunos de los algoritmos más importantes basados en la inteligencia de enjambre que se pueden encontrar son:

- El algoritmo de **optimización por enjambre de partículas (*Particle Swarm Optimization*, PSO)** fue desarrollada por Kennedy. Se trata de un método de optimización inspirado en las bandadas de pájaros o en los bancos de peces, pensado para aplicarse a funciones no lineales en espacios continuos y discretos [9], [10].

Figura 4. Imagen de una bandada de pájaros, la cual inspira el funcionamiento del algoritmo de optimización por enjambre de partículas

A partir de un espacio de soluciones, y dado un criterio de calidad o *fitness*, se intenta mejorar una solución candidata. La idea es moverse por el espacio de búsqueda en base a una velocidad que engloba tanto la inercia o movimiento de la partícula en el pasado, como los resultados de la propia partícula y los de sus vecinos. La finalidad es encontrar la mejor solución global, aunque no se puede asegurar que sea la óptima en ninguno de los casos [9], [11]. El funcionamiento de este algoritmo se puede observar tanto en la Figura 5, mediante un esquema, como en la Figura 6, en forma de diagrama de flujo.

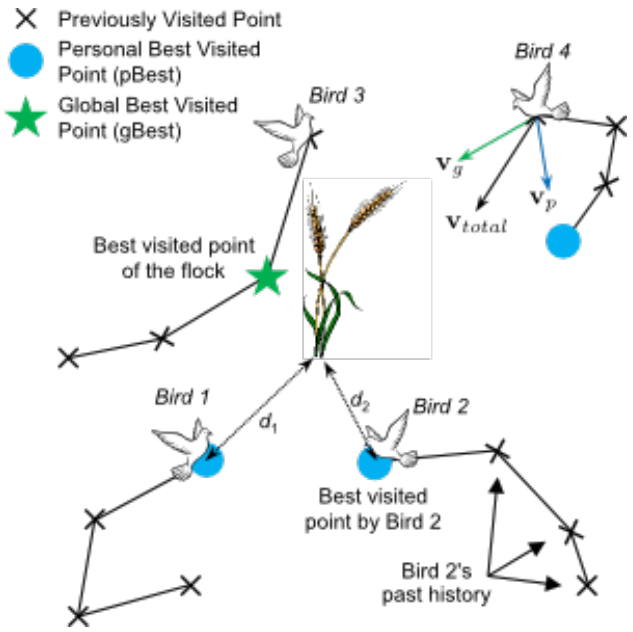

Figura 5. Esquema representativo del funcionamiento del algoritmo de optimización por enjambre de partículas [12].

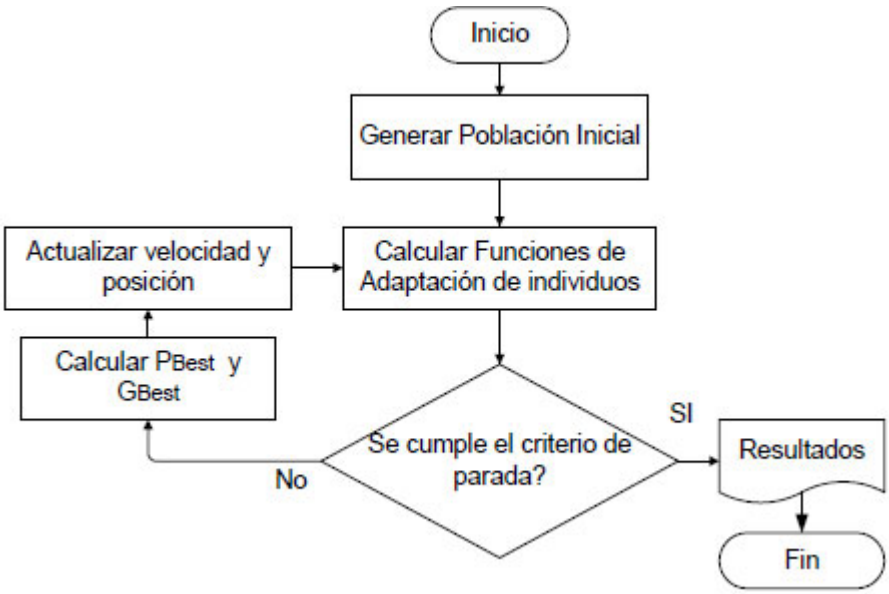

Figura 6. Diagrama de flujo del funcionamiento del algoritmo de optimización por enjambre de partículas, siendo P_{Best} la mejor solución explorada por una partícula y G_{Best} la mejor solución explorada por todas las partículas [13].

- La **optimización por colonia de hormigas (*Ant Colony Optimization*, ACO)** se trata de una familia de algoritmos derivados de un estudio realizado por Dorigo *et al.* [14]. Está inspirado en el comportamiento social de las hormigas [9].

Figura 7. Imagen de unas hormigas, las cuales inspiran el funcionamiento del algoritmo de optimización por colonia de hormigas.

- Las hormigas usan unas sustancias químicas llamadas feromonas para comunicarse. Esta comunicación se produce debido a que las feromonas depositadas por una hormiga al avanzar por un camino en busca de comida ejercen una acción sobre la decisión de las hormigas posteriores. Éstas eligen con mayor probabilidad aquel camino que contenga una mayor concentración de sustancia, permitiendo que encuentren tanto la localización de las fuentes de alimento como su nido. De este modo, los mejores caminos atraerán a una mayor cantidad de hormigas que, a su vez, reforzarán esta trayectoria depositando una nueva cantidad de feromonas, dando así lugar a una retroalimentación positiva. Sin embargo, en los trayectos sub-óptimos se irá evaporando la feromona al haber un menor número de hormigas que circulen por ellos, evitando de esta manera converger en óptimos locales [9], [10]. El funcionamiento de este algoritmo se puede observar tanto en la Figura 8, mediante un esquema, como en la Figura 9, en forma de diagrama de flujo.

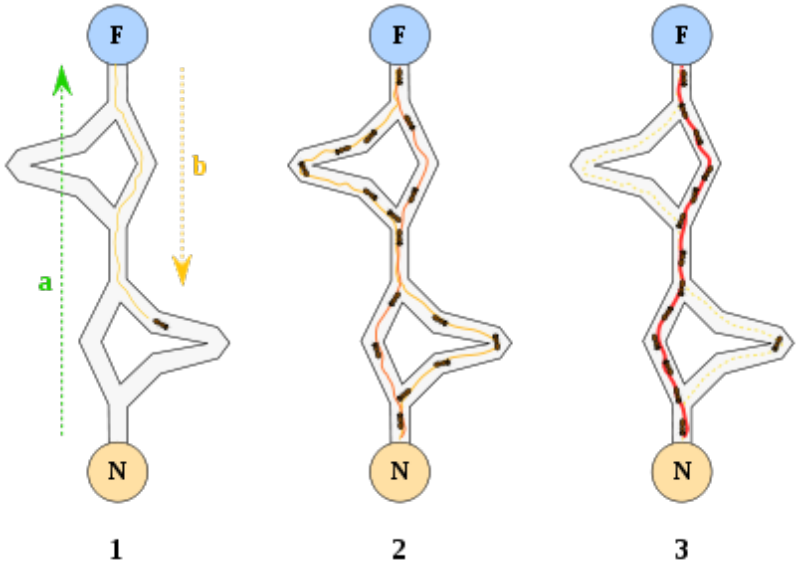

Figura 8. Esquema representativo del funcionamiento del algoritmo de optimización por colonia de hormigas. En el primer paso una hormiga encuentra una fuente de alimento (F) y regresa a la colonia (N) por un camino b dejando un rastro de feromonas. A continuación, el resto de hormigas siguen los posibles caminos hasta la fuente de alimento. Por último, se observa que el camino más corto es el escogido por todas las hormigas debido a que ha acumulado un mayor rastro de feromonas [15].

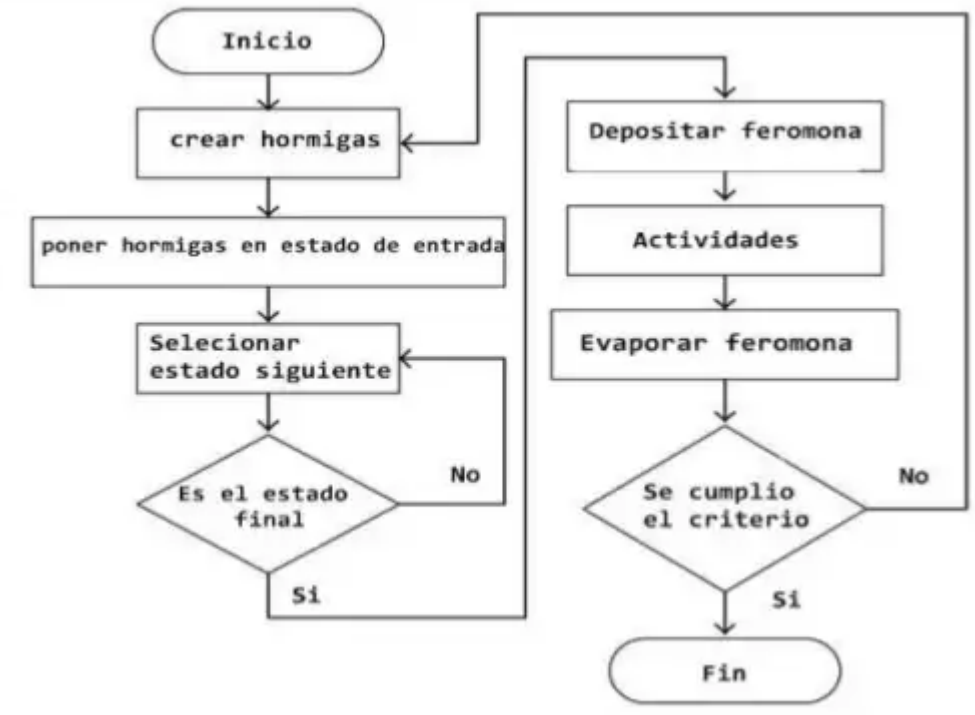

Figura 9. Diagrama de flujo del funcionamiento del algoritmo de optimización por colonia de hormigas [15].

- El **algoritmo de colonia artificial de abejas (*Artificial Bee Colony algorithm*, ABC)**, fue propuesto por Karaboga en 2005. Consiste en un método de optimización inspirado en los enjambres de abejas [9].

Figura 10. Imagen de unas abejas, las cuales inspiran el funcionamiento del algoritmo de colonia artificial de abejas.

- Se basa en la forma de búsqueda de alimento de las abejas, las cuales se dividen en abejas empleadas o trabajadoras, abejas espectadoras y abejas exploradoras. Las distintas fuentes de alimento se reparten en el espacio de búsqueda con un número equivalente de abejas cada una. Cada fuente está a una distancia y contiene una cantidad de néctar diferente que equivalen al *fitness*. Las abejas trabajadoras conocen la rentabilidad y ubicación de la fuente de alimento que tienen asignada. Las abejas espectadoras esperan la llegada de las abejas trabajadoras para que compartan la información sobre cada fuente de alimento mediante una danza y así escoger la de mejor *fitness*. Por último, las abejas exploradoras buscan nuevas fuentes de alimento aleatoriamente permitiendo ampliar el espacio de búsqueda [9]. El funcionamiento de este algoritmo se puede observar tanto en la Figura 11, mediante un esquema de los elementos que lo conforman, como en la Figura 12, en forma de diagrama de flujo.

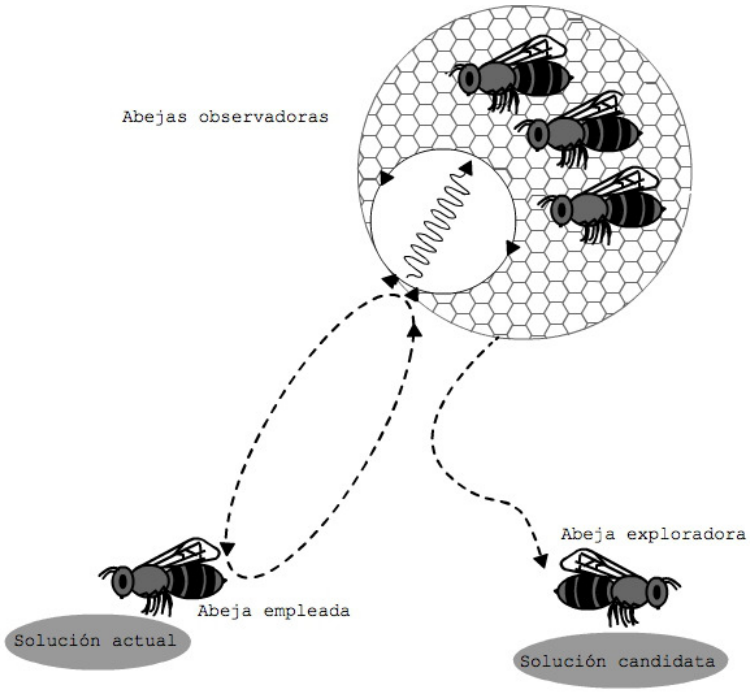

Figura 11. Esquema representativo de los distintos elementos que conforman el algoritmo de colonia artificial de abejas [16].

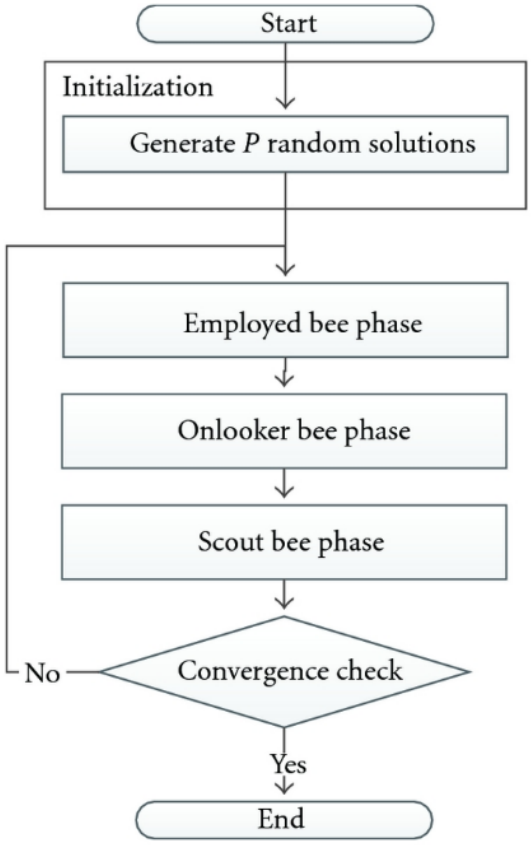

Figura 12. Diagrama de flujo del funcionamiento del algoritmo de colonia artificial de abejas [17].

2.3 Algoritmos empleados

A la hora de desarrollar el sistema propuesto en este TFG, se probaron distintos algoritmos como el ACO, el ABC, el *Harmony Search*, así como el clasificador basado en centroide y el kNN. Finalmente, se utilizaron el *Harmony Search* y el kNN debido a que eran los que mejores resultados alcanzaban. Por ello, son los que se detallan más en profundidad a continuación.

2.3.1 *Harmony Search*

La búsqueda armónica (*Harmony Search*, HS) es un algoritmo meta-heurístico cuyo funcionamiento se basa en la improvisación musical [18]. En el 2001, Zong Woo Geem *et al.* [3] propusieron la versión de este algoritmo para problemas discretos, y posteriormente, en el 2005, Kang Seok Lee y Zong Woo Geem presentaron la versión para problemas continuos [19].

Los métodos heurísticos existentes hasta ese momento imitaban los fenómenos naturales, por lo que cabía pensar se podría encontrar un nuevo algoritmo en otros fenómenos naturales o en los artificiales, como el musical. En este caso, la búsqueda de la mejor armonía musical podía servir como modelo para diseñar una nueva técnica [3].

En la naturaleza, la armonía es una relación especial entre varias ondas de sonido que se propagan a frecuencias distintas. En cambio, la armonía musical es una combinación de sonidos que, desde un punto de vista estético, se consideran agradables al oído humano [3].

Los algoritmos de optimización buscan el mejor estado, es decir, que el coste del óptimo global sea mínimo o que el beneficio o eficiencia sean máximos. En el caso de las actuaciones musicales, el mejor estado sería encontrar la mejor armonía posible, la cual viene determinada por la estimación estética del conjunto de los sonidos producidos. Estos sonidos se pueden mejorar con la práctica si lo que se quiere es tener una valoración estética más alta. Una breve presentación de estas observaciones se muestra en la Tabla 1 [3].

COMPARISON FACTOR	OPTIMIZATION PROCESS	PERFORMANCE PROCESS
Best state	Global Optimum	Fantastic Harmony
Estimated by	Objective Function	Aesthetic Standard
Estimated with	Values of Variables	Pitches of Instruments
Process unit	Each Iteration	Each Practice

Tabla 1. Comparación entre el proceso de optimización y el proceso de una actuación musical [3].

El nuevo algoritmo que proponen Zong Woo Geem *et al.* [3] basado en la armonía musical se llama *Harmony Search* (HS). Los pasos a seguir en el procedimiento del algoritmo HS son [3]:

- **Paso 1:** Inicializar una memoria de armonía (*Harmony Memory*, HM).
- **Paso 2:** Improvisar una nueva armonía de HM.
- **Paso 3:** Si la nueva armonía es mejor que la peor armonía en la HM, incluir la nueva armonía en HM y quitar la peor armonía.
- **Paso 4:** Volver al paso 2 hasta que se cumplan los criterios de finalización.

Un ejemplo de funcionamiento del algoritmo HS se describe a continuación. Considere un trío de jazz compuesto por violín, saxofón y teclado. Inicialmente, la memoria se rellena con armonías aleatorias: (Do, Mi, Sol), (Do, Fa, La) y (Si, Re, Sol) que están ordenadas de mayor a menor según la estimación estética, como se puede ver en la Figura 13. A continuación, en el procedimiento de improvisación, los tres instrumentos producen una nueva armonía; por ejemplo (Do, Re, La): el violín de entre las notas {Do, Do, Si} selecciona la {Do}; el saxofón escoge la nota {Re} entre {Mi, Fa, Re}; y el teclado elige la nota {La} entre {Sol, La, Sol}. Cada nota en la HM tiene la misma oportunidad de ser seleccionada, por ejemplo, en el caso del saxofón, cada una de las notas Mi, Fa o Re en HM tiene una probabilidad de selección de 33.3%. Si la armonía recién creada (Do, Re, La) es mejor que cualquiera de las armonías existentes en la HM, la nueva armonía se incluye en HM y la peor armonía (en este ejemplo, (Si, Re, Sol)) se elimina. Este proceso se repite hasta que se obtienen resultados satisfactorios (casi óptimos) [3].

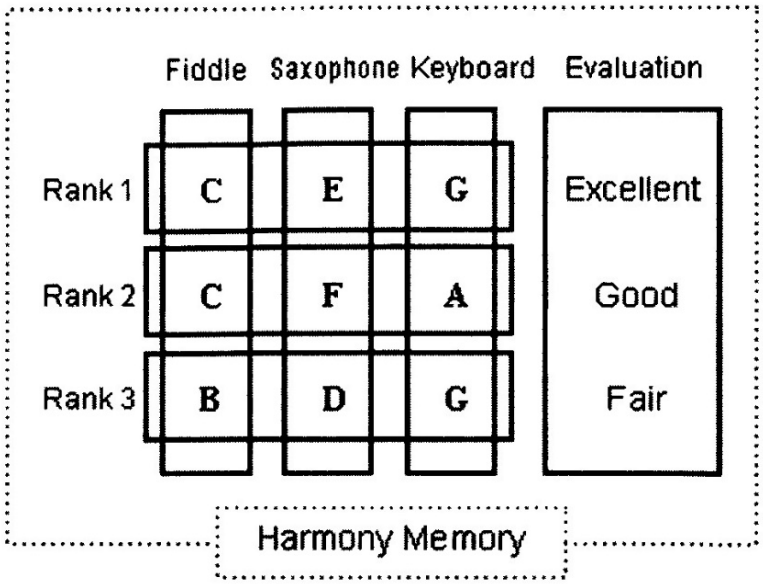

Figura 13. Estructura de la memoria de armonía donde C, D, E, F, G, A y B se corresponden con las notas Do, Re, Mi, Fa, Sol, La y Si respectivamente [3].

Cuando alguna de las partes de la solución global no existe inicialmente en la HM, la búsqueda de armonía inicia un parámetro: la tasa de consideración de memoria de armonía (*Harmony Memory Considering Rate*, HMCR), que varía de 0 a 1. Si un valor generado de manera uniforme entre 0 y 1 está por encima del valor actual de la HMCR, el algoritmo HS no elegirá una nota de la memoria, sino que escogerá una nota al azar de entre de todo el rango

reproducible. Por ejemplo, una HMCR de 0.95 significa que, en el siguiente paso, el algoritmo elegirá un valor variable de la HM con un 95% de probabilidad [3].

Para mejorar las soluciones y escapar de los óptimos locales, se puede introducir una opción que imita el ajuste de tono de cada instrumento para afinar el conjunto. El mecanismo de ajuste de tono se diseña como un cambio a valores vecinos dentro de un rango de valores posibles. Por ejemplo, si hay seis valores posibles, como {1, 3, 4, 6, 7, 9}, el {6} se puede mover hacia los vecinos {4} o {7} en el proceso de ajuste de tono. Por ejemplo, una tasa de ajuste de tono (*Pitch Adjusting Rate*, PAR) de 0.10 significa que el algoritmo elige un valor vecino con un 10% de probabilidad (un valor superior con un 5% o un valor inferior con un 5%) [3].

Supongamos que el conjunto de valores posibles de un instrumento es {Do, Re, Mi, Fa, Sol}, la HMCR es 0.95, la PAR es 0.10, y el instrumento tiene {Do, Mi, Sol} en la HM. En el proceso de improvisación, el algoritmo HS elige uniformemente una nota de {Do, Mi, Sol} con un 95% de probabilidad o una nota de {Do, Re, Mi, Fa, Sol} con un 5% de probabilidad, y cuando se selecciona {Mi} se puede cambiar a {Re} o {Fa} con un 10% de probabilidad [3].

Tal y como se ha explicado, el algoritmo HS (Búsqueda Armónica) incorpora, por naturaleza, la estructura de los métodos heurísticos existentes. Por ejemplo, administra varios vectores simultáneamente de una manera similar a los algoritmos genéticos (GAs). Sin embargo, la principal diferencia entre los GAs y el algoritmo HS es que este último crea un nuevo vector a partir de todos los vectores existentes (todas las armonías que hay en la memoria), mientras que los GA crean el nuevo vector a partir de solo dos de los vectores existentes (los padres) [3]. Además, el procesamiento es mucho más rápido puesto que no realiza cálculos matemáticos complejos. Esto hace que el tiempo de convergencia sea la gran diferencia del algoritmo HS con la mayor parte de los algoritmos meta-heurísticos existentes, ya que se reduce considerablemente. Otra ventaja es que puede manejar tanto variables discretas como continuas y que obvia los óptimos locales para intentar aproximarse lo máximo posible al óptimo global [18].

2.3.2 kNN

El método de los *k* vecinos más cercanos (*k-Nearest Neighbors*, kNN) sirve para resolver problemas de clasificación. Estima la probabilidad de la clase *a posteriori*, esto quiere decir que asigna una muestra *x* a la clase que más se repita de entre sus *k* vecinos más cercanos. El criterio de clasificación que se sigue es una medida de similitud o distancia. En la fase de entrenamiento el algoritmo almacena las etiquetas de las distintas y clases los vectores característicos de los datos usados como ejemplos en esta fase. En la fase de clasificación se utiliza un vector en el espacio de rasgos para evaluar un ejemplo del que aún no se sabe la clase a la que pertenece. A continuación, se calcula la distancia entre los vectores almacenados y la del nuevo vector y se seleccionan los *k* vecinos más cercanos. Si la distancia entre los vectores es alta quiere decir que éstos son muy diferentes entre sí, por el contrario, si la distancia es baja significa que los vectores son muy similares. El nuevo ejemplo se clasifica según la clase más repetida entre los vecinos más próximos [20].

A pesar de que hay muchas funciones de distancia, la mayoría de los algoritmos de clasificación de kNN utilizan la distancia euclidiana, que se define como:

$$dist(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Donde p y q son puntos del espacio n-dimensional.

Debido a su simplicidad, el kNN es uno de los clasificadores más utilizados. El principal problema de este método es la dificultad de determinar el valor de *k*. Por una parte, si adquiere un valor alto, se corre el riesgo de hacer la clasificación de acuerdo con la mayoría y no según el parecido. Por el contrario, si el valor de *k* es bajo, puede que la clasificación sea incorrecta dado que los datos seleccionados no han sido suficientes como instancias de comparación [20]. En la Figura 14 se muestra un ejemplo de cómo se asignaría la clase a un nuevo elemento con el clasificador kNN.

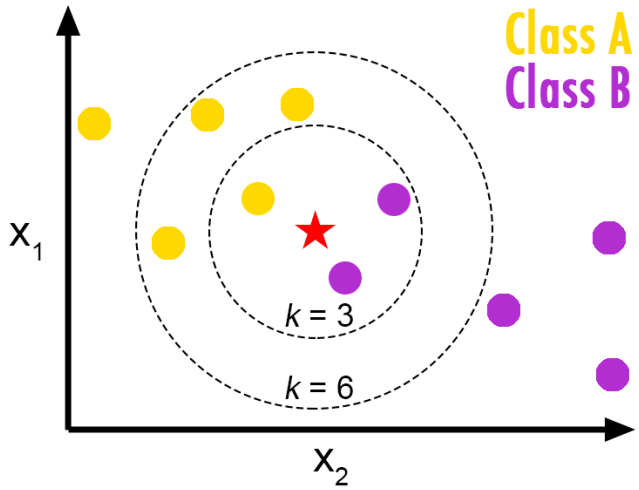

Figura 14. Representación de un ejemplo del clasificador de los *k* vecinos más cercanos. El elemento que se quiere clasificar es la estrella roja. En el caso de k=3, se clasificaría en la Clase B (morada), ya que hay 2 círculos morados frente a 1 amarillo dentro de la línea discontinua interna. En cambio, para k=6, sería clasificado como Clase A (amarilla), debido a que ahora hay 4 círculos amarillos frente a 2 morados dentro de la línea discontinua externa [21].

Capítulo 3: Diseño y desarrollo del sistema

En este capítulo se pone en contexto la necesidad del sistema propuesto y se detallan los materiales y métodos utilizados, describiendo cada etapa llevada a cabo: adquisición de datos, preprocesado, reducción de la dimensionalidad y estimación del estado de los elementos rotantes.

3.1 Contexto

El mantenimiento de una cosechadora debe de ser frecuente para evitar que el mal funcionamiento de un componente mecánico derive en una reparación todavía más costosa. Por ello, ante la necesidad de mejorar el mantenimiento predictivo de una cosechadora, surge la necesidad de crear un sistema que detecte el estado de funcionamiento de los elementos rotantes de ésta, mediante una señal de vibración adquirida de la máquina.

El análisis de vibraciones es un método no intrusivo que, principalmente, se emplea en las máquinas para las revisiones del estado de funcionamiento de los elementos rotantes [22]–[24]. Esto se debe a que las señales de vibración pueden reflejar este aspecto. En el caso de maquinaria sin aislamiento de vibraciones, estas señales se propagan a lo largo de la estructura con baja atenuación. Esto permite monitorizar ciertos elementos rotantes mediante la colocación de un acelerómetro situado en la estructura de la máquina [25]. Sin embargo, la propagación de vibraciones tiene un efecto negativo, ya que también transmite señales de vibración procedentes de otros elementos de la estructura que se pueden añadir a la señal de interés, lo que supone una dificultad añadida a la hora de extraer la información relevante [26].

Las señales de vibración de los elementos rotantes normalmente se analizan en el dominio de la frecuencia, ya que existe una relación entre la frecuencia de rotación del elemento y los picos significativos del espectro [22]. Debido a esta relación, los expertos pueden estimar el estado de funcionamiento de los elementos de la máquina buscando patrones en el espectro de la señal. Sin embargo, para llevar esto a cabo, se requiere un análisis experto de las señales de vibración y un conocimiento detallado de los elementos de la máquina, que no siempre está disponible. Para evitar este problema, se han propuesto sistemas automatizados que utilizan el análisis frecuencial para estimar el estado de los elementos de la máquina cuando no hay expertos disponibles [27], [28]. Estos sistemas incorporan conocimientos básicos del elemento de la máquina para extraer características de la señal del espectro y estimar así su estado en base a esas características mediante técnicas de *machine learning*.

Con el propósito de mejorar el mantenimiento predictivo de una cosechadora, se han realizado trabajos de investigación previos como el de Ruiz-Gonzalez *et al.* [29], en el que, para conseguir este objetivo, se hacía uso de la técnica de extracción de características y de un clasificador basado en Máquinas de Vector de Soporte (*Support Vector Machines*, SVM). En

este caso, la señal de vibración fue preprocesada para, a continuación, extraer 12 características de ella, tras lo cual, un método de búsqueda exhaustiva seleccionó las más adecuadas. Posteriormente, se evaluó la precisión del clasificador basado en SVM haciendo uso de la validación cruzada *Leave-One-Out*, utilizando como datos de entrada las características seleccionadas. En la misma línea de investigación, Martínez-Martínez *et al.* [30] proponen para el mismo problema un método basado en una Red Neuronal Artificial (*Artificial Neural Network*, ANN). Los coeficientes positivos del espectro en frecuencia de la señal de vibración eran las entradas de la ANN. Además, se implementaron 4 métodos de aprendizaje basados en algoritmos genéticos para ajustar los pesos y sesgos de la ANN. Con la mejor implementación del método de aprendizaje lo que se consiguió era mejorar la tasa de éxito y reducir el tiempo necesario para realizar el ajuste de la ANN.

El proyecto desarrollado en el presente Trabajo Fin de Grado continúa esta última línea de investigación. El principal objetivo que se plantea es la reducción de la complejidad y el tiempo de ejecución del método propuesto por Martínez-Martínez *et al.* [30]. Para ello, se ha desarrollado un nuevo método de selección de frecuencias basado en algoritmos evolutivos, que consigue estimar el estado de funcionamiento de los elementos rotantes de una cosechadora a partir de una señal de vibración procedente de un único punto de la cosechadora.

3.2 Materiales y métodos

En esta sección, se presentan los materiales y métodos empleados en el sistema desarrollado. Las principales etapas de las que consta el sistema desarrollado en este Trabajo Fin de Grado se muestran en la Figura 15 y son: adquisición de datos, preprocesado, reducción de la dimensionalidad y estimación del estado de los elementos rotantes de la cosechadora. En las siguientes subsecciones se explica con detalle los materiales y métodos empleados en cada una de ellas.

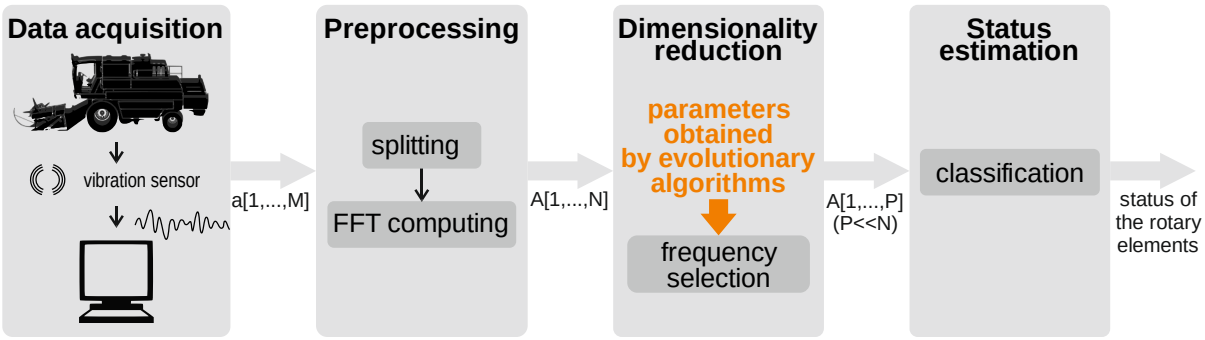

Figura 15. Diagrama de bloques que resume las principales etapas de procesamiento del sistema desarrollado.

3.2.1 Etapa de adquisición de datos

Los datos utilizados en el trabajo fueron adquiridos por una cosechadora New Holland TC56 con 4050 horas de trabajo. Las señales de vibración fueron adquiridas cuando la cosechadora estaba parada y operando en modo trilla. Para medir las señales de vibración se

emplearon 4 acelerómetros piezoelétricos uniaxiales Brüel & Kjær 4507-B-006 situados en distintos puntos del chasis de la cosechadora, tal y como se puede ver en la Figura 16. Los sensores se colocaron usando un montaje adhesivo siguiendo las pautas de Scheffer *et al.* [22].

Figura 16. Esquema de la cosechadora en el que la cruz azul representa la localización de la trilla, la cruz amarilla representa la localización del motor, la cruz naranja representa la localización del picador y los símbolos rojos representan las distintas ubicaciones en las que se colocaron los acelerómetros en el chasis de la cosechadora.

Las señales de vibración fueron adquiridas usando el *software NI Sound and Vibration Assistant* y el sistema de adquisición de datos (DAQ) de *National Instruments* (NI). Éste estaba compuesto de un módulo de adquisición de datos de para señales de entrada analógicas NI 9234 y un chasis de adquisición de datos compacto NI cDAD-9172 para conectar el módulo DAQ a un ordenador.

Varias condiciones de operación fueron simuladas en la cosechadora para poder adquirir los datos de todas las combinaciones de las siguientes condiciones: (*i*) velocidad del motor (ralentí/máximo); (*ii*) estado de operación de la trilla (encendida/apagada); (*iii*) estado de equilibrio de la trilla (equilibrada/desequilibrada) cuando la trilla está en estado encendida; (*iv*) estado de operación del picador (encendido/apagado); y (*v*) estado de equilibrio del picador (equilibrado/desequilibrado) cuando el picador está en estado encendido. Además, tanto la trilla como el picador fueron desequilibrados a propósito. En el caso del picador se hizo mediante la rotura de una cuchilla, ya que una causa frecuente de los desequilibrios producidos es la rotura de la cuchilla contra las piedras. La trilla, en cambio, puede desequilibrarse cuando sus barras sufren un desgaste no uniforme debido al uso, así que se optó por añadir un peso excéntrico que simulase este defecto.

De este modo, un total de 18 procesos de adquisición de datos distintos fueron simulados, grabando 60 segundos de operación de la cosechadora para cada proceso y usando una frecuencia de muestreo de 1706 Hz, para obtener un total de 102360 muestras.

3.2.2 Etapa de preprocesado

Los datos de las series temporales adquiridos de los acelerómetros fueron preprocesados para adaptarlos a la siguiente etapa. Este proceso de adaptación se realizó en 3 fases: *splitting*, transformación en frecuencia y eliminación de la información redundante.

En el *splitting*, se partía de 4 señales de vibración procedentes de cada uno de los 4 acelerómetros, cada una tenía un total de 102360 muestras y una duración de 60 segundos. Cada señal se dividió en 10 épocas, de 6 segundos cada una, por lo que cada época constaba de 10236 muestras.

Posteriormente, se realizó la transformación en frecuencia, la cual consistía en calcular la Transformada de Fourier Discreta (*Discrete Fourier Transform*, DFT) de cada una de las épocas. En este caso, la DFT se calculó haciendo uso de la Transformada Rápida de Fourier (*Fast Fourier Transform*, FFT), obteniendo a la salida una señal de la misma longitud que la secuencia de entrada.

Por último, se eliminó la información redundante de la señal obtenida tras la FFT. Como la señal de vibración es una señal real, al hacer la FFT se obtiene un espectro simétrico. Debido a esto, se eliminó la parte negativa del espectro ya que contenía información redundante y se mantuvo únicamente la parte positiva, que eran 5118 muestras.

3.2.3 Etapa de reducción de la dimensionalidad

Una vez que los datos fueron preprocesados, se llevó a cabo la etapa de reducción de la dimensionalidad, en la cual un algoritmo evolutivo seleccionaba las mejores frecuencias. La Figura 17 muestra los distintos procesos realizados en esta etapa.

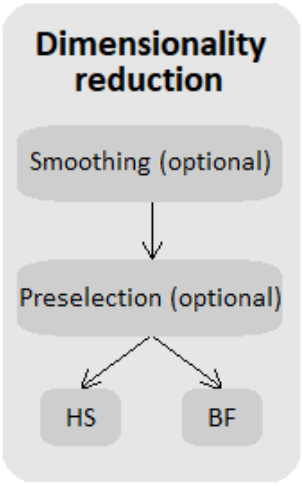

Figura 17. Diagrama de bloques que resume los procesos llevados a cabo en la etapa de reducción de la dimensionalidad. Primero se hizo un suavizado y posteriormente una preselección de frecuencias (ambos bloques son opcionales). Finalmente, se podía elegir entre realizar el proceso mediante el algoritmo *Harmony Search* (HS) o mediante *Brute Force* (BF).

Primero, se diseñó un bloque de suavizado, el cual se podía activar y desactivar y se situaba tanto en la parte de preselección de frecuencias como en la de reducción de la dimensionalidad. El suavizado se realizó mediante la convolución de la señal con una matriz de 5 unos y posteriormente se normalizó multiplicando por 1/5. De esta forma se eliminaron los picos indeseados al atenuarse con las frecuencias del entorno. Además, al tener la opción de activarlo y desactivarlo se podía comprobar si la precisión mejoraba o no al emplearlo.

Posteriormente, se hizo una preselección de entre las 5118 frecuencias que había tras realizar la FFT, ya que se realizó una selección de las más relevantes. Para ello, se calcularon la media y la varianza de las frecuencias con el fin de compararlas mediante un umbral. Aquellas componentes cuya media fuera mayor que un umbral, dado por una constante

multiplicada por la varianza, se eliminaron. El criterio para escoger el umbral fue que, al hacer la preselección, se obtuvieran en torno a 300-400 frecuencias en todos los casos considerados.

Finalmente, para reducir la dimensionalidad se usó un algoritmo evolutivo. De esta forma, se seleccionó un subconjunto de frecuencias de entre las preseleccionadas anteriormente, que serán las empleadas como entradas del clasificador. Inicialmente, se probaron los algoritmos de optimización por colonia de hormigas (ACO) y de colonia artificial de abejas (ABC), junto con el algoritmo de búsqueda armónica (HS). La simplicidad del algoritmo HS, y la mejora del rendimiento respecto a los otros algoritmos heurísticos, son algunas de las razones que justificaron el empleo de HS en detrimento de los algoritmos ACO y ABC. HS basa su funcionamiento en el proceso de búsqueda de la mejor armonía en una interpretación musical, por ello, denomina armónicos a sus potenciales soluciones. Los armónicos, en general, están compuestos por varios elementos, que en este caso son frecuencias. Además, el algoritmo HS define una memoria de armonía (HM) en la que se van almacenando las mejores soluciones encontradas hasta el momento. En el problema que se plantea en este TFG, el número de soluciones almacenadas durante el procedimiento fue de 50, y el número de frecuencias consideradas en la solución final en los distintos casos simulados varió entre 1 y 3, ya que con 3 frecuencias en la mayoría de los problemas analizados ya se alcanzaba el 100% de precisión. En cada iteración del algoritmo HS se crea una nueva solución o armónico y se calcula su aptitud, es decir, cómo de buena es dicha solución. A continuación, esta solución se guarda en la memoria si su aptitud es mayor que alguna de las que ya están almacenadas, y se borra la solución de la memoria con menor aptitud, o se elimina si no tiene una aptitud lo suficientemente alta o si ya está en la memoria. Además, a la hora de crear una nueva solución o armónico en cada iteración, el algoritmo trabaja frecuencia a frecuencia, pudiendo tomar una frecuencia aleatoriamente, elegir una de las que ya están entre las soluciones de la memoria, o buscar alguna frecuencia vecina a las ya existentes. La tasa de consideración de memoria de armonía (HMCR) elegida fue de 0.75, por lo que la probabilidad de escoger una frecuencia de todas las posibles era de un 25%, y el 75% era la probabilidad de seleccionar una frecuencia de entre las que están en la HM. La tasa de ajuste de tono (PAR) usada fue de 0.10, lo que quiere decir que con una probabilidad de un 10% se cambiaba a una frecuencia vecina. Las frecuencias vecinas a las que se podía mover se determinaban con una ventana de adyacencia, en la que el tamaño indicaba el número de frecuencias vecinas a las que se podía mover, en nuestro caso el tamaño elegido fue 5. El número de iteraciones del algoritmo se fijó en 5000 porque con ese número ya se observaba que el HS convergía.

3.2.4 Etapa de estimación del estado (clasificador)

Para establecer la clase a la que pertenecía cada muestra, inicialmente se optó por usar un clasificador basado en centroides, el cual determinaba si el dato pertenecía a una clase u otra en función de su proximidad a los promedios de cada clase. Primero se utilizaban unos datos para entrenar y se calculaba el promedio según la clase. Luego, con los datos restantes se calculaba la distancia a cada centroide y se determinaba que la clase a la que pertenecía

era aquella cuyo centroide estuviera más cercano. Pero, como los resultados obtenidos no eran los esperados, se llegó a la conclusión de que, debido a las características de nuestros datos, este clasificador no era el más adecuado. En la Figura 18 se puede observar un ejemplo de datos mal clasificados cuando éstos no están en grupos o en clústers adyacentes. La solución a este problema fue utilizar un nuevo clasificador, el de los *k* vecinos más cercanos (kNN), el cual sí se ajustaba a nuestras características. Este algoritmo determina la clase a la que pertenece un nuevo dato en función de la clase más frecuente a la que pertenecen los *k* vecinos más cercanos. En nuestro caso se escogió *k*=5. De esta forma, los datos sí que se clasificaban correctamente y los resultados obtenidos mejoraban considerablemente.

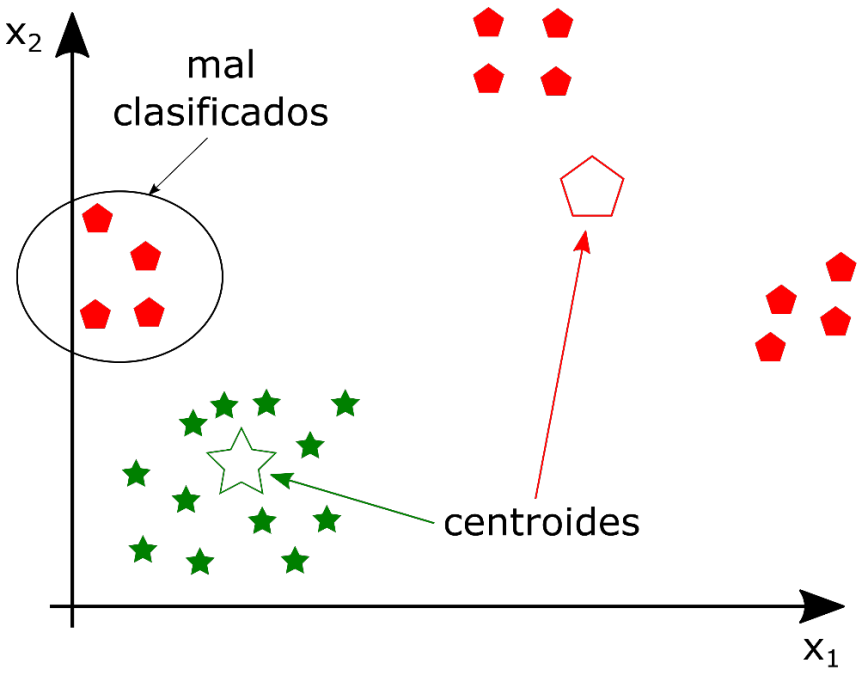

Figura 18. Representación de un ejemplo en el que se demuestra que el clasificador basado en centroides no se adapta a datos que están en grupos o clústers no adyacentes. En este caso, una muestra del grupo marcado como "mal clasificados" con el clasificador basado en centroides se asignaría al grupo verde (estrellas) de manera incorrecta. Sin embargo, con el clasificador de los *k* vecinos más cercanos esto no ocurriría.

Capítulo 4: Resultados

Para evaluar la eficacia del sistema creado se han analizado distintos casos: estudio de la influencia de la preselección, estudio de la influencia del suavizado, estudio de la influencia del algoritmo *Harmony Search* y estudio de la influencia de la ubicación del acelerómetro utilizado. Los resultados obtenidos tras estos análisis son los que se presentan a continuación, junto con una comparativa de los resultados obtenidos en un trabajo previo.

4.1 Estudio de la influencia de la preselección

Primeramente, se evalúo la mejora que suponía el hecho de hacer una preselección de las frecuencias más relevantes, frente a no hacer una preselección y ejecutar el algoritmo con las 5118 frecuencias obtenidas. Las frecuencias más relevantes se determinaron mediante el cálculo de la media y la varianza, las cuales se comparaban mediante un umbral. Se eliminaron aquellas componentes cuyo promedio no superaba este umbral, calculado tal y como se ha comentado en el capítulo anterior. El umbral se fijó para que en cada problema analizado se obtuvieran entre 300 y 400 frecuencias preseleccionadas. Los resultados sin hacer uso de la preselección se pueden ver en la Tabla 2 y en la Figura 19.

Tipo de clasificador	Nº frecuencias preselección	Nº frecuencias solución	*Fitness* máximo	Tiempo de ejecución (min)
Motor	5118	1	0.9944	1.7260
Trilla	5118	1	1.0000	1.6637
Trilla Equilibrada	5118	1	0.8250	1.2110
Picador	5118	1	0.8278	2.0045
Picador Equilibrado	5118	1	0.8250	1.7801

Tabla 2. Resultados de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, sin el uso de la preselección. En todos los casos se obtuvo una solución de 1 frecuencia, además se muestra la precisión máxima alcanzada y el tiempo de ejecución.

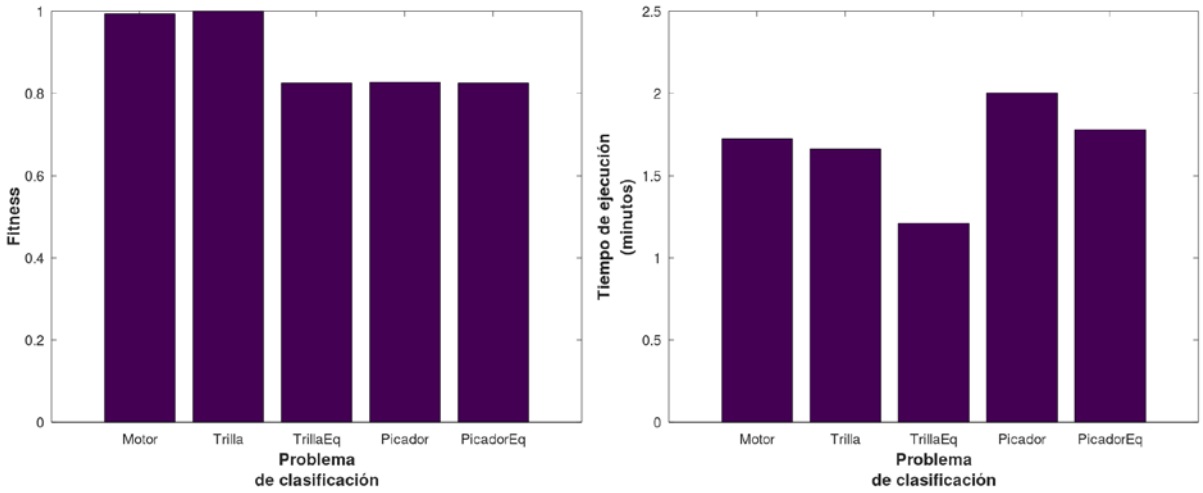

Figura 19. Resultados de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, sin el uso de la preselección. En todos los casos se obtuvo una solución de 1 frecuencia. En la imagen de la izquierda se muestra una comparativa de la precisión máxima alcanzada y en la imagen de la derecha una comparativa del tiempo de ejecución.

A continuación, en la Tabla 3 y en la Figura 20 se pueden observar los resultados empleando la preselección.

Tipo de clasificador	Nº frecuencias preselección	Nº frecuencias solución	*Fitness* máximo	Tiempo de ejecución (min)
Motor	365	1	0.9944	0.1029
Trilla	361	1	1.0000	0.1091
Trilla Equilibrada	380	1	0.8250	0.0946
Picador	371	1	0.8278	0.1168
Picador Equilibrado	346	1	0.8000	0.0789

Tabla 3. Resultados de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, haciendo uso de la preselección. En todos los casos se obtuvo una solución de 1 frecuencia, además se muestra la precisión máxima alcanzada y el tiempo de ejecución.

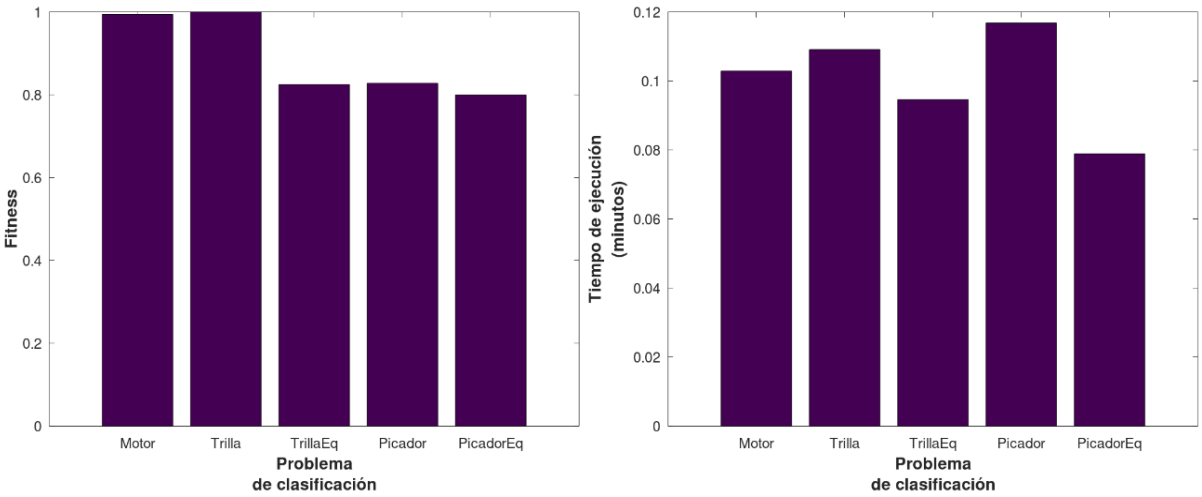

Figura 20. Resultados de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, haciendo uso de la preselección. En todos los casos se obtuvo una solución de 1 frecuencia. En la imagen de la izquierda se muestra una comparativa de la precisión máxima alcanzada y en la imagen de la derecha una comparativa del tiempo de ejecución.

Por último, en la Figura 21 se presenta una comparativa entre los resultados obtenidos con y sin preselección.

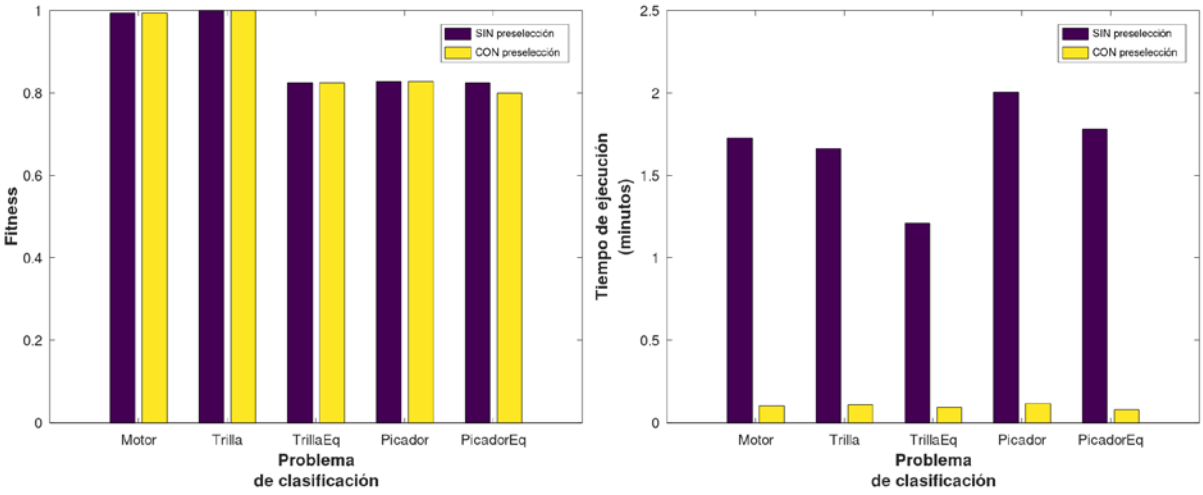

Figura 21. Comparativa de resultados de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, analizando el empleo de la preselección y sin ella. En todos los casos se obtuvo una solución de 1 frecuencia. En la imagen de la izquierda se muestra una comparativa de la precisión máxima alcanzada y en la imagen de la derecha una comparativa del tiempo de ejecución.

Los resultados obtenidos muestran que la precisión (*fitness*) es prácticamente la misma en ambos casos, sin embargo, el tiempo de ejecución se reduce considerablemente al hacer uso de la preselección. Debido a esto, en los siguientes casos a analizar se usará la preselección, ya que, como se ha comprobado, además de reducir el tiempo de ejecución, la precisión no se ve afectada en gran medida.

4.2 Estudio de la influencia del suavizado

A continuación, se decidió analizar si el bloque de suavizado, que se podía activar y desactivar, producía alguna mejora en los resultados al ejecutarlo con el *Harmony Search*. El bloque de suavizado se introdujo tanto en la etapa de preselección como a la hora de ejecutar el algoritmo. La comparativa de resultados, haciendo uso del suavizado y sin él, se puede ver en la Tabla 4 y en la Figura 22.

Nº frecuencias solución	Tipo de clasificador	*Fitness* máximo con suavizado	*Fitness* máximo sin suavizado
1	Motor	1.0000	0.9944
	Trilla	1.0000	1.0000
	Trilla Equilibrada	0.8083	0.8250
	Picador	0.8556	0.8278
	Picador Equilibrado	0.9250	0.8000
2	Motor	1.0000	1.0000
	Trilla	1.0000	1.0000
	Trilla Equilibrada	1.0000	0.9833
	Picador	0.9889	0.9222
	Picador Equilibrado	1.0000	0.9917
3	Motor	1.0000	1.0000
	Trilla	1.0000	1.0000
	Trilla Equilibrada	1.0000	0.9917
	Picador	1.0000	0.9667
	Picador Equilibrado	1.0000	1.0000

Tabla 4. Resultados del algoritmo *Harmony Search* utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, comparando la precisión alcanzada con el bloque del suavizado y sin él. Además, se analiza la evolución de la precisión conforme se aumenta el número de frecuencias obtenidas como solución.

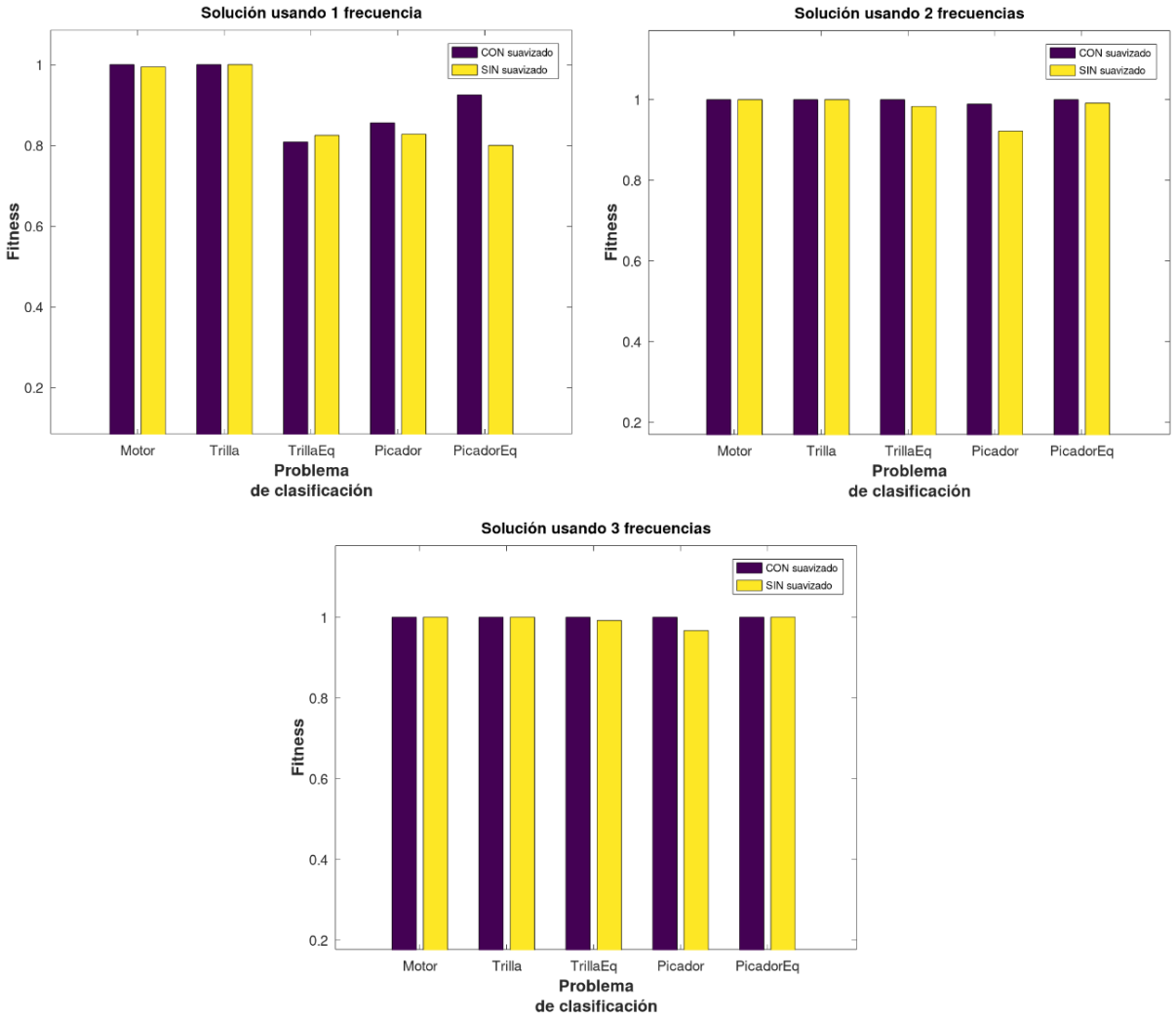

Figura 22. Resultados del algoritmo *Harmony Search* utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, comparando la precisión alcanzada con el bloque del suavizado y sin él. En la imagen de arriba a la izquierda se muestra una comparativa de la precisión máxima alcanzada obteniendo como solución 1 frecuencia, en la imagen de arriba a la derecha con una solución de 2 frecuencias y en la imagen de abajo con una solución de 3 frecuencias.

Tal y como se puede ver, el *fitness* mejora con el suavizado. Si los resultados sin aplicar el bloque del suavizado ya eran buenos, al aplicarlo mejoran notablemente, llegándose a obtener con una solución de 2 frecuencias un *fitness* del 100% en 4 de las 5 condiciones a analizar (velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla y estado de equilibrio del picador), frente a 2 condiciones (velocidad del motor y estado de operación de la trilla) sin el suavizado. Esto es debido a que, con el suavizado, lo que se consigue es que en una misma frecuencia se junta información de las frecuencias adyacentes y se reduce el ruido. Además, se observa que con una solución de 3 frecuencias y el suavizado se alcanza el 100% de precisión en todas las condiciones (motor, trilla, trilla equilibrada, picador y picador equilibrado), por lo que en el resto de los casos a analizar se utilizará el bloque de suavizado.

4.3 Estudio de la influencia del algoritmo *Harmony Search*

En este caso, se decidió comparar los resultados obtenidos con el algoritmo de *Harmony Search* y con la Fuerza Bruta. Ambos estaban bajo las mismas condiciones: se utilizó la preselección y el suavizado, y como solución se fijó a 2 el número de frecuencias seleccionadas. La comparativa de resultados, entre la Fuerza Bruta (FB) y *Harmony Search* (HS) se puede observar tanto en la Tabla 5 como en la Figura 23.

Tipo de clasificador	*Fitness* FB	*Fitness* HS	Tiempo ejecución FB (min)	Tiempo ejecución HS (min)
Motor	1.0000	1.0000	18.4162	2.3857
Trilla	1.0000	1.0000	22.8537	2.1960
Trilla Equilibrada	1.0000	1.0000	8.9466	1.3872
Picador	0.9889	0.9889	24.1017	2.3003
Picador Equilibrado	1.0000	1.0000	7.6887	1.3908

Tabla 5. Resultados del algoritmo *Harmony Search* y de la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, comparando la precisión alcanzada y el tiempo de ejecución entre ambos.

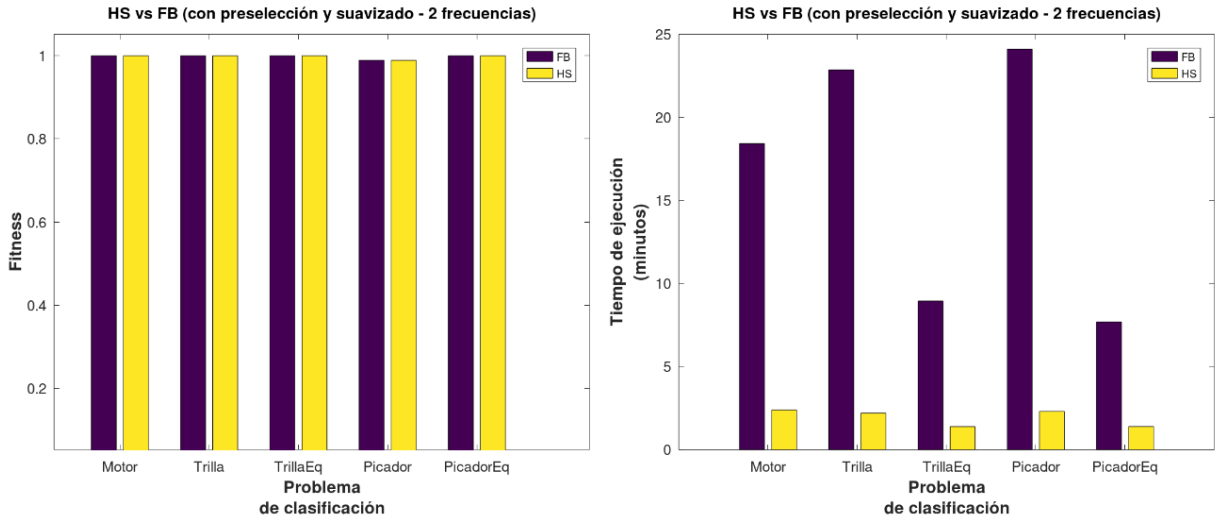

Figura 23. Comparativa de resultados entre el algoritmo *Harmony Search* y la Fuerza Bruta utilizando únicamente el canal 1 para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador. En la imagen de la izquierda se muestra una comparativa de la precisión máxima alcanzada y en la imagen de la derecha una comparativa del tiempo de ejecución.

La precisión obtenida, tanto con la Fuerza Bruta como con *el Harmony Search*, fue exactamente la misma. Sin embargo, tal y como cabía esperar, los tiempos de ejecución fueron claramente superiores con la Fuerza Bruta. Con el *Harmony Search* se conseguía una reducción del tiempo de ejecución de entre 82% y 90%. Además, cabe destacar que en la solución conforme aumenta el número de frecuencias, la diferencia de los tiempos de ejecución entre la Fuerza Bruta y el algoritmo HS se agudiza, llegando a ser inviable la ejecución de la Fuerza

Bruta cogiendo 4 frecuencias. Sin embargo, el algoritmo HS realiza un buen papel sin requerir un tiempo prohibitivo. Esto se debe a que la Fuerza Bruta combina todas las posibles soluciones y el algoritmo HS explora parte de las soluciones y se queda con la mejor.

4.4 Estudio de la influencia de la ubicación del acelerómetro utilizado

Por último, se realizó una comparativa entre los distintos acelerómetros/sensores que se habían situado en la cosechadora, con el objetivo de determinar si la ubicación de éstos era determinante. Todos los canales se analizaron bajo las mismas características: se utilizó la preselección y el suavizado, y se seleccionó una solución de 3 frecuencias. En la Tabla 6 y en la Figura 24 se puede ver una comparativa entre los resultados obtenidos con cada uno de los 4 canales.

Canales	Tipo de clasificador	*Fitness* máximo
Ch1	Motor	1.0000
	Trilla	1.0000
	Trilla Equilibrada	1.0000
	Picador	1.0000
	Picador Equilibrado	1.0000
Ch2	Motor	1.0000
	Trilla	1.0000
	Trilla Equilibrada	1.0000
	Picador	0.9944
	Picador Equilibrado	1.0000
Ch3	Motor	1.0000
	Trilla	1.0000
	Trilla Equilibrada	0.9583
	Picador	1.0000
	Picador Equilibrado	1.0000
Ch4	Motor	1.0000
	Trilla	1.0000
	Trilla Equilibrada	1.0000
	Picador	1.0000
	Picador Equilibrado	1.0000

Tabla 6. Resultados del algoritmo *Harmony Search* para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, comparando la precisión alcanzada entre los 4 acelerómetros situados en la cosechadora.

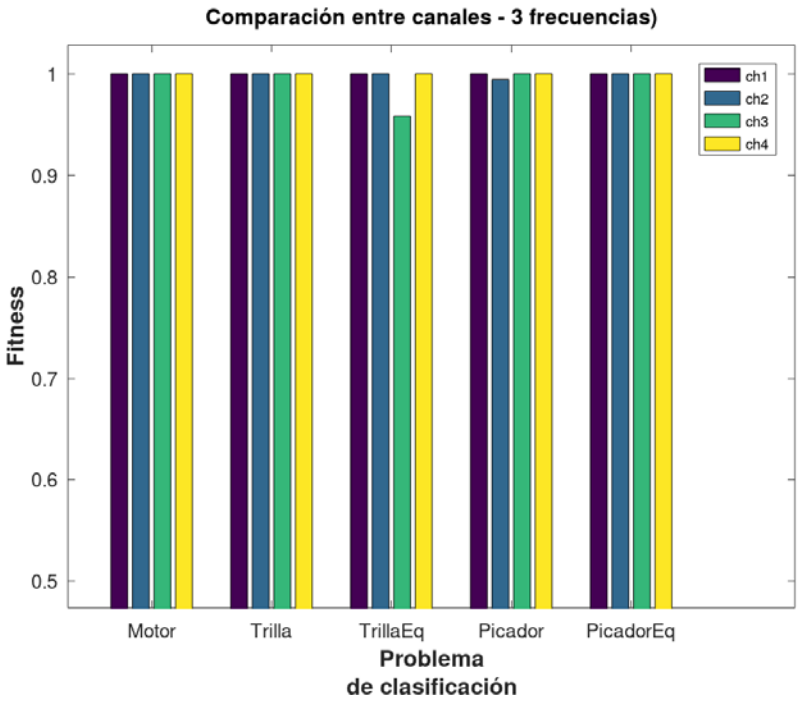

Figura 24. Resultados del algoritmo *Harmony Search* para cada condición analizada: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador, comparando la precisión alcanzada entre los 4 acelerómetros situados en la cosechadora.

Por los resultados obtenidos se puede concluir que da igual la ubicación del acelerómetro, ya que las precisiones obtenidas son prácticamente las mismas.

4.5 Comparación de resultados con trabajo previo

Tal y como se había planteado anteriormente, uno de los objetivos que se buscaba, en el sistema desarrollado en este Trabajo Fin de Grado era la reducción de la complejidad y del tiempo de ejecución del método propuesto en el trabajo previo de Martínez-Martínez *et al.* [30]. Haciendo una comparación de los resultados, se puede decir que este objetivo se ha cumplido, ya que tanto con la Fuerza Bruta como con el algoritmo HS se ha reducido la complejidad, lo que conlleva una disminución del tiempo de ejecución. Haciendo uso de la preselección y del suavizado y obteniendo una solución de 2 frecuencias se necesitaron tiempos de entre 8 y 24 min para encontrar las frecuencias con la Fuerza Bruta, pero la ejecución del algoritmo HS no llegó a alcanzar los 3 min en ninguna de las condiciones analizadas: velocidad del motor, estado de operación de la trilla, estado de equilibrio de la trilla, estado de operación del picador y estado de equilibrio del picador. Esto significa que con el algoritmo HS se consiguió una reducción del tiempo de ejecución de entre 82% y 90%, respecto a la Fuerza Bruta. En cambio, en el método propuesto por Martínez-Martínez *et al.* [30] los tiempos de entrenamiento eran mucho más elevados, del orden de horas. Esto se debe, por una parte, a que las redes neuronales son mucho más complejas que el algoritmo HS y el clasificador kNN empleados en este TFG. Por otro lado, con las redes neuronales, se trabajaba con todas las frecuencias, es decir, no se hacía preselección, por lo que el sistema se hacía muy grande y complejo al tener que ajustar muchos parámetros y, por lo tanto, el

proceso de ajuste de éstos llevaba más tiempo. Además, también se produce una mejora en la precisión, ya que con el algoritmo HS, haciendo uso de la preselección y el suavizado y fijando a 2 el número de frecuencias obtenidas como solución, se ha conseguido una precisión del 100% en todos los casos menos en el picador, que es del 98.89%. En cambio, con las redes neuronales la precisión más alta que se alcanzaba era del 100%, pero únicamente en el caso de estudio de la velocidad del motor, y la más baja fue del 80%, correspondiente a la condición de equilibrio de la trilla. Los resultados de Martínez-Martínez *et al.* [30] se obtuvieron con una red neuronal de 8 neuronas en la capa oculta.

Capítulo 5: Conclusiones y líneas futuras

A lo largo de este Trabajo Fin de Grado se ha realizado un sistema para detectar las frecuencias más relevantes de los elementos rotantes de una cosechadora con el objetivo de determinar su estado de funcionamiento. Para ello, se han utilizado el algoritmo *Harmony Search* y el clasificador kNN, y se han analizado distintos casos de estudio para evaluar el papel del uso de la preselección, del suavizado y de la ubicación del sensor, así como las mejoras producidas al utilizar el algoritmo HS frente a la Fuerza Bruta. Tras los resultados obtenidos se han extraído las siguientes conclusiones y posibles líneas futuras.

5.1 Conclusiones

Los resultados obtenidos evidencian que, en sistemas para la detección del estado de funcionamiento de elementos rotantes de máquinas como el desarrollado en este trabajo:

- **Un suavizado de las señales adquiridas y una preselección de las frecuencias mejora la precisión del sistema.** Con la preselección se consigue que, sin alterar la precisión, se produzca una disminución en los tiempos de ejecución, ya que no se trabaja con todas las frecuencias. Por otro lado, haciendo uso del suavizado, se ha logrado una mejora de la precisión (*fitness*), llegando a alcanzar con una solución de 3 frecuencias precisiones del 100% en todas las condiciones analizadas.
- **El empleo de un algoritmo evolutivo para la selección de frecuencias mejora la precisión y disminuye los tiempos de ejecución.** Con un algoritmo evolutivo se ha conseguido disminuir los tiempos de ejecución sin alterar la precisión. Con soluciones de 3 frecuencias y haciendo uso del suavizado y la preselección, la precisión obtenida ha sido del 100% en todas las condiciones analizadas.
- **El algoritmo *Harmony Search* ayuda a mantener las precisiones sin requerir un tiempo prohibitivo.** Al comparar la Fuerza Bruta con el algoritmo HS, se ha visto que la precisión no varía y que los tiempos de ejecución disminuyen entre un 82% y un 90% haciendo uso del *Harmony Search*. Además, esta diferencia se agudiza según se añaden frecuencias en la solución, llegando a ser inviable la ejecución de la Fuerza Bruta con 4 frecuencias.
- **Los acelerómetros pueden ser colocados en distintas partes de la cosechadora.** En las pruebas realizadas obtuvimos resultados similares para diferentes ubicaciones del sensor en la cosechadora.

A raíz de estas conclusiones y una vez analizado el trabajo realizado, también se puede evidenciar que:

- **El rendimiento y la eficiencia del algoritmo *Harmony Search* mejoran al introducir una etapa de reducción de la dimensionalidad.** Debido al gran número de frecuencias a tener en consideración en este trabajo, al incluir esta etapa, se

conseguía disminuir el número de combinaciones de las posibles soluciones y, con ello, reducir la complejidad del sistema y mejorar el tiempo de convergencia.

- **Con un algoritmo evolutivo y distintos métodos de preprocesado de la señal de vibración se reduce la dimensionalidad y la complejidad del sistema.** Esto se debe a que a partir de este proceso se conseguían seleccionar las frecuencias de mayor interés para cada problema.

5.2 Líneas futuras

Las posibles líneas futuras que se han pensado para continuar o completar este Trabajo Fin de Grado se plantean a continuación:

- Justificar las frecuencias seleccionadas por el algoritmo y relacionarlas con armónicos de las velocidades de giro de los elementos rotantes bajo análisis.
- Determinar si la posición del acelerómetro tiene alguna influencia en las frecuencias seleccionadas.
- Utilizar el sistema desarrollado con los datos procedentes de una cosechadora funcionando en condiciones de trabajo reales.
- Probar otros algoritmos evolutivos.
- Probar otros clasificadores.
- Estudiar la influencia de los parámetros HM, PAR, HMCR del algoritmo HS, así como el número de iteraciones para optimizar sus condiciones de funcionamiento.

Referencias

[1] F. J. Soltero, “Aplicaciones de los algoritmos evolutivos al análisis de procesos económicos,” Universidad Complutense de Madrid, 2014.

[2] J. A. García, “Análisis E Implementación De Algoritmos Evolutivos Para La Optimización De Simulaciones En Ingeniería Civil,” Universidad Católica San Antonio de Murcia, 2014.

[3] Z. W. Geem, J. H. Kim, and G. V. Loganathan, “A New Heuristic Optimization Algorithm: Harmony Search,” *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.

[4] J. Gamarra, “Optimización en el diseño de resortes helicoidales de compresión basado en algoritmos genéticos,” Universidad Nacional del Centro del Perú, 2009.

[5] C. Luque, “Predicción Local mediante Algoritmos Evolutivos,” Universidad Carlos III de Madrid, 2009.

[6] E. Yeguas, “Un modelo de rendimiento de algoritmos evolutivos aplicados a la selección de la solución deseada,” Universidad de Granada, 2009.

[7] F. Sancho-Caparrini, “Algoritmos genéticos,” 2018. [Online]. Available: http://www.cs.us.es/~fsancho/?e=65. [Accessed: 20-Aug-2019].

[8] M. Bonelli and H. F. Begliardo, “Optimización de armaduras planas mediante diseño paramétrico y algoritmos genéticos: efectos de la no correspondencia objeto real - modelo idealizado,” *Mecánica Comput.*, vol. 34, pp. 501–515, 2016.

[9] A. Imas, “Algoritmos inspirados en swarm intelligence para el enrutamiento en redes de telecomunicaciones,” Universidad politécnica de Madrid, 2013.

[10] M. A. Muñoz, J. A. López, and E. F. Caicedo, “Inteligencia de enjambres: sociedades para la solución de problemas (una revisión),” *Rev. Ing. e Investig.*, vol. 28, no. 2, pp. 119–130, 2008.

[11] S. Molina and G. Leguizamón, “Algoritmos de Inteligencia de Enjambres Orientados a Map Reduce,” Universidad Nacional de San Luis, 2015.

[12] J. Kovitz, “Optimization in Electromagnetics.” [Online]. Available: http://joshkovitz.com/research/projects/optimization-in-electromagnetics/. [Accessed: 20-Aug-2019].

[13] D. Gutiérrez, W. M. Villa, and J. M. López-Lezama, “Flujo Óptimo Reactivo mediante Optimización por Enjambre de Partículas,” *Inf. Tecnológica*, vol. 28, no. 5, pp. 215–224, 2017.

[14] M. Dorigo, V. Maniezzo, and A. Colorni, “The ant system: An autocatalytic optimizing process,” 1991.

[15] F. Sancho-Caparrini, “Algoritmos de hormigas y el problema del viajante,” 2018. [Online]. Available: http://www.cs.us.es/~fsancho/?e=71. [Accessed: 20-Aug-2019].

[16] M. D. Araoz, “Uso de tolerancias dinámicas para el manejo de restricciones en problemas de optimización usando la colonia artificial de abejas,” Instituto Tecnológico de Orizaba, 2013.

[17] A. Randazzo, “Swarm optimization methods in microwave imaging,” *Int. J. Microw. Sci. Technol.*, vol. 2012, p. 12, 2012.

[18] C. Cobos, J. Perez, and D. Estupiñan, “A Survey of Harmony Search.Una Revisión de la Búsqueda Armónica,” *Rev. Av. en Sist. e Informática (RASI).*, vol. 8, no. 2, pp. 67–80, 2011.

[19] K. S. Lee and Z. W. Geem, “A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice,” *Comput. Methods Appl. Mech. Eng.*, vol. 194, no. 36–38, pp. 3902–3933, 2005.

[20] D. Delgado, M. Rainer, L. Hernández, R. Orozco, and J. Lorenzo, “Algoritmos de aprendizaje automático para la clasificación de neuronas piramidales afectadas por el envejecimiento,” *Rev. Cuba. Informática Médica*, vol. 8, no. 3, pp. 559–571, 2016.

[21] L. A. Salcedo, “Introducción al Machine Learning #9 - K Vecinos más cercanos (Clasificación y Regresión),” 2018. [Online]. Available: http://www.pythondiario.com/2018/01/introduccion-al-machine-learning-9-k.html. [Accessed: 20-Aug-2019].

[22] C. Scheffer and P. Girdhar, *Machinery Vibration Analysis & Predictive Maintenance*. 2004.

[23] W. Q. Wang, F. Ismail, and M. Farid Golnaraghi, “Assessment of gear damage monitoring techniques using vibration measurements,” *Mech. Syst. Signal Process.*, 2001.

[24] Z. K. Peng and F. L. Chu, “Application of the wavelet transform in machine condition monitoring and fault diagnostics: A review with bibliography,” *Mechanical Systems and Signal Processing*. 2004.

[25] Z. Li, X. Yan, Z. Tian, C. Yuan, Z. Peng, and L. Li, “Blind vibration component separation and nonlinear feature extraction applied to the nonstationary vibration signals for the gearbox multi-fault diagnosis,” *Meas. J. Int. Meas. Confed.*, 2013.

[26] A. Albarbar, F. Gu, and A. D. Ball, “Diesel engine fuel injection monitoring using acoustic measurements and independent component analysis,” *Meas. J. Int. Meas. Confed.*, 2010.

[27] G. Cheng, Y. L. Cheng, L. H. Shen, J. B. Qiu, and S. Zhang, “Gear fault identification based on Hilbert-Huang transform and SOM neural network,” *Meas. J. Int. Meas. Confed.*, 2013.

[28] V. Sugumaran and K. I. Ramachandran, “Effect of number of features on classification of roller bearing faults using SVM and PSVM,” *Expert Syst. Appl.*, 2011.

[29] R. Ruiz-Gonzalez, J. Gomez-Gil, F. J. Gomez-Gil, and V. Martínez-Martínez, “An SVM-Based classifier for estimating the state of various rotating components in Agro-Industrial machinery with a vibration signal acquired from a single point on the machine chassis,” *Sensors (Switzerland)*, vol. 14, no. 11, pp. 20713–20735, 2014.

[30] V. Martínez-Martínez, F. J. Gomez-Gil, J. Gomez-Gil, and R. Ruiz-Gonzalez, “An Artificial Neural Network based expert system fitted with Genetic Algorithms for detecting the status of several rotary components in agro-industrial machines using a single vibration signal,” *Expert Syst. Appl.*, vol. 42, no. 17–18, pp. 6433–6441, 2015.

Anexo I: Códigos desarrollados en el TFG

En este anexo se recogen los códigos realizados en este TFG correspondientes a las siguientes partes: preselección, algoritmo *Harmony Search*, Fuerza Bruta y clasificador kNN.

A1.1 Código de la preselección

```
%Preselección con el promedio y varianza para todos distinguiendo entre
ralentí y máximo y equilibrado y desequilibrado


%Cargamos datos de trabajo y parámetros
close all;clear all;clc;pause(0.1);tic;     %Cerramos las ventanas abiertas
y limpiamos el workspace
load('./../../999 Ruben/DatosFrec.mat');


%Obtención de datos de trabajo

%Matriz medidas
canal=1;
medidasMatriz=zeros(length(medidasEnFrec),size(medidasEnFrec{1}.ch,1));
for i=1:length(medidasEnFrec)
    medidasMatriz(i,:)=medidasEnFrec{i}.ch(:,canal);
end

%Inicializamos las variables
datos={};
W_conv=5;
matriz_clase1=[];
matriz_clase2=[];
matriz_clase3=[];
matriz_clase4=[];
matriz_clase5=[];
matriz_clase6=[];
matriz_clase7=[];
matriz_clase8=[];
matriz_clase9=[];
matriz_clase10=[];
matriz_clase11=[];
matriz_clase12=[];
matriz_clase13=[];
matriz_clase14=[];
matriz_clase15=[];
matriz_clase16=[];
matriz_clase17=[];
matriz_clase18=[];

%Suavizado
for i=1:length(medidasEnFrec)
    for j=1:size(medidasEnFrec{i}.ch,2)

medidasEnFrec{i}.ch(:,j)=conv(medidasEnFrec{i}.ch(:,j),ones(W_conv,1)./W_co
nv,'same');
    end
end
```

```
%Orden de información
for i=1:length(medidasEnFrec)

    datos{i}= struct('clase',1,'vector',[]);
    datos{i}.vector = medidasEnFrec{i}.ch(:,canal);

    if strcmp(medidasEnFrec{i}.motor,'ralenti')
        if strcmp(medidasEnFrec{i}.trilla,'on')
            if strcmp(medidasEnFrec{i}.trilla_eq,'si')
                if strcmp(medidasEnFrec{i}.picador,'on')
                    if strcmp(medidasEnFrec{i}.picador_eq,'si')
                        datos{i}.clase=1; %Ral trilla_eq picador_eq
                    else
                        datos{i}.clase=2; %Ral trilla_eq picador_deseq
                    end
                else
                    datos{i}.clase=3; %Ral trilla_eq picador_off
                end
            else
                if strcmp(medidasEnFrec{i}.picador,'on')
                    if strcmp(medidasEnFrec{i}.picador_eq,'si')
                        datos{i}.clase=4; %Ral trilla_deseq picador_eq
                    else
                        datos{i}.clase=5; %Ral trilla_deseq picador_deseq
                    end
                else
                    datos{i}.clase=6; %Ral trilla_deseq picador_off
                end
            end
        else
            if strcmp(medidasEnFrec{i}.picador,'on')
                if strcmp(medidasEnFrec{i}.picador_eq,'si')
                    datos{i}.clase=7; %Ral trilla_off picador_eq
                else
                    datos{i}.clase=8; %Ral trilla_off picador_deseq
                end
            else
                datos{i}.clase=9; %Ral trilla_off picador_off
            end
        end

    elseif strcmp(medidasEnFrec{i}.motor,'maximo')
        if strcmp(medidasEnFrec{i}.trilla,'on')
            if strcmp(medidasEnFrec{i}.trilla_eq,'si')
                if strcmp(medidasEnFrec{i}.picador,'on')
                    if strcmp(medidasEnFrec{i}.picador_eq,'si')
                        datos{i}.clase=10; %Max trilla_eq picador_eq
                    else
                        datos{i}.clase=11; %Max trilla_eq picador_deseq
                    end
                else
                    datos{i}.clase=12; %Max trilla_eq picador_off
                end
            else
                if strcmp(medidasEnFrec{i}.picador,'on')
                    if strcmp(medidasEnFrec{i}.picador_eq,'si')
                        datos{i}.clase=13; %Max trilla_deseq picador_eq
                    else
                        datos{i}.clase=14; %Max trilla_deseq picador_deseq
                    end
                else
```

```
                    datos{i}.clase=15; %Max trilla_deseq picador_off
                end
            end
        else
            if strcmp(medidasEnFrec{i}.picador,'on')
                if strcmp(medidasEnFrec{i}.picador_eq,'si')
                    datos{i}.clase=16; %Max trilla_off picador_eq
                else
                    datos{i}.clase=17; %Max trilla_off picador_deseq
                end
            else
                datos{i}.clase=18; %Max trilla_off picador_off
            end
        end
    end
end


%Juntamos todos los datos de cada clase en una matriz
for i=1:length(datos)

    if datos{i}.clase==1
        matriz_clase1=[matriz_clase1,datos{i}.vector];
    elseif datos{i}.clase==2
        matriz_clase2=[matriz_clase2,datos{i}.vector];
    elseif datos{i}.clase==3
        matriz_clase3=[matriz_clase3,datos{i}.vector];
    elseif datos{i}.clase==4
        matriz_clase4=[matriz_clase4,datos{i}.vector];
    elseif datos{i}.clase==5
        matriz_clase5=[matriz_clase5,datos{i}.vector];
    elseif datos{i}.clase==6
        matriz_clase6=[matriz_clase6,datos{i}.vector];
    elseif datos{i}.clase==7
        matriz_clase7=[matriz_clase7,datos{i}.vector];
    elseif datos{i}.clase==8
        matriz_clase8=[matriz_clase8,datos{i}.vector];
    elseif datos{i}.clase==9
        matriz_clase9=[matriz_clase9,datos{i}.vector];
    elseif datos{i}.clase==10
        matriz_clase10=[matriz_clase10,datos{i}.vector];
    elseif datos{i}.clase==11
        matriz_clase11=[matriz_clase11,datos{i}.vector];
    elseif datos{i}.clase==12
        matriz_clase12=[matriz_clase12,datos{i}.vector];
    elseif datos{i}.clase==13
        matriz_clase13=[matriz_clase13,datos{i}.vector];
    elseif datos{i}.clase==14
        matriz_clase14=[matriz_clase14,datos{i}.vector];
    elseif datos{i}.clase==15
        matriz_clase15=[matriz_clase15,datos{i}.vector];
    elseif datos{i}.clase==16
        matriz_clase16=[matriz_clase16,datos{i}.vector];
    elseif datos{i}.clase==17
        matriz_clase17=[matriz_clase17,datos{i}.vector];
    else
        matriz_clase18=[matriz_clase18,datos{i}.vector];
    end
end
```

```
%Calculamos los promedios y las varianzas
prom_clase1=mean(matriz_clase1,2);
prom_clase2=mean(matriz_clase2,2);
prom_clase3=mean(matriz_clase3,2);
prom_clase4=mean(matriz_clase4,2);
prom_clase5=mean(matriz_clase5,2);
prom_clase6=mean(matriz_clase6,2);
prom_clase7=mean(matriz_clase7,2);
prom_clase8=mean(matriz_clase8,2);
prom_clase9=mean(matriz_clase9,2);
prom_clase10=mean(matriz_clase10,2);
prom_clase11=mean(matriz_clase11,2);
prom_clase12=mean(matriz_clase12,2);
prom_clase13=mean(matriz_clase13,2);
prom_clase14=mean(matriz_clase14,2);
prom_clase15=mean(matriz_clase15,2);
prom_clase16=mean(matriz_clase16,2);
prom_clase17=mean(matriz_clase17,2);
prom_clase18=mean(matriz_clase18,2);

var_clase1=std(matriz_clase1,0,2);
var_clase2=std(matriz_clase2,0,2);
var_clase3=std(matriz_clase3,0,2);
var_clase4=std(matriz_clase4,0,2);
var_clase5=std(matriz_clase5,0,2);
var_clase6=std(matriz_clase6,0,2);
var_clase7=std(matriz_clase7,0,2);
var_clase8=std(matriz_clase8,0,2);
var_clase9=std(matriz_clase9,0,2);
var_clase10=std(matriz_clase10,0,2);
var_clase11=std(matriz_clase11,0,2);
var_clase12=std(matriz_clase12,0,2);
var_clase13=std(matriz_clase13,0,2);
var_clase14=std(matriz_clase14,0,2);
var_clase15=std(matriz_clase15,0,2);
var_clase16=std(matriz_clase16,0,2);
var_clase17=std(matriz_clase17,0,2);
var_clase18=std(matriz_clase18,0,2);


%MOTOR
dif_ral_max1=abs(prom_clase1-prom_clase10);
dif_ral_max2=abs(prom_clase2-prom_clase11);
dif_ral_max3=abs(prom_clase3-prom_clase12);
dif_ral_max4=abs(prom_clase4-prom_clase13);
dif_ral_max5=abs(prom_clase5-prom_clase14);
dif_ral_max6=abs(prom_clase6-prom_clase15);
dif_ral_max7=abs(prom_clase7-prom_clase16);
dif_ral_max8=abs(prom_clase8-prom_clase17);
dif_ral_max9=abs(prom_clase9-prom_clase18);

maximo_ral_max1=max(var_clase1, var_clase10);
maximo_ral_max2=max(var_clase2, var_clase11);
maximo_ral_max3=max(var_clase3, var_clase12);
maximo_ral_max4=max(var_clase4, var_clase13);
maximo_ral_max5=max(var_clase5, var_clase14);
maximo_ral_max6=max(var_clase6, var_clase15);
maximo_ral_max7=max(var_clase7, var_clase16);
maximo_ral_max8=max(var_clase8, var_clase17);
maximo_ral_max9=max(var_clase9, var_clase18);
```

```
umbral_motor=7;

ind_ral_max1=find(dif_ral_max1>(umbral_motor*maximo_ral_max1));
ind_ral_max2=find(dif_ral_max2>(umbral_motor*maximo_ral_max2));
ind_ral_max3=find(dif_ral_max3>(umbral_motor*maximo_ral_max3));
ind_ral_max4=find(dif_ral_max4>(umbral_motor*maximo_ral_max4));
ind_ral_max5=find(dif_ral_max5>(umbral_motor*maximo_ral_max5));
ind_ral_max6=find(dif_ral_max6>(umbral_motor*maximo_ral_max6));
ind_ral_max7=find(dif_ral_max7>(umbral_motor*maximo_ral_max7));
ind_ral_max8=find(dif_ral_max8>(umbral_motor*maximo_ral_max8));
ind_ral_max9=find(dif_ral_max9>(umbral_motor*maximo_ral_max9));

prom_motor=[ind_ral_max1; ind_ral_max2; ind_ral_max3; ind_ral_max4;
ind_ral_max5; ind_ral_max6; ind_ral_max7; ind_ral_max8; ind_ral_max9];

promedio_motor=unique(prom_motor);


%TRILLA
dif_trilla1=abs(prom_clase7-prom_clase1);
dif_trilla2=abs(prom_clase7-prom_clase4);
dif_trilla3=abs(prom_clase8-prom_clase2);
dif_trilla4=abs(prom_clase8-prom_clase5);
dif_trilla5=abs(prom_clase9-prom_clase3);
dif_trilla6=abs(prom_clase9-prom_clase6);
dif_trilla7=abs(prom_clase16-prom_clase10);
dif_trilla8=abs(prom_clase16-prom_clase13);
dif_trilla9=abs(prom_clase17-prom_clase11);
dif_trilla10=abs(prom_clase17-prom_clase14);
dif_trilla11=abs(prom_clase18-prom_clase12);
dif_trilla12=abs(prom_clase18-prom_clase15);

maximo_trilla1=max(var_clase7, var_clase1);
maximo_trilla2=max(var_clase7, var_clase4);
maximo_trilla3=max(var_clase8, var_clase2);
maximo_trilla4=max(var_clase8, var_clase5);
maximo_trilla5=max(var_clase9, var_clase3);
maximo_trilla6=max(var_clase9, var_clase6);
maximo_trilla7=max(var_clase16, var_clase10);
maximo_trilla8=max(var_clase16, var_clase13);
maximo_trilla9=max(var_clase17, var_clase11);
maximo_trilla10=max(var_clase17, var_clase14);
maximo_trilla11=max(var_clase18, var_clase12);
maximo_trilla12=max(var_clase18, var_clase15);

umbral_trilla=8;

ind_trilla1=find(dif_trilla1>(umbral_trilla*maximo_trilla1));
ind_trilla2=find(dif_trilla1>(umbral_trilla*maximo_trilla2));
ind_trilla3=find(dif_trilla3>(umbral_trilla*maximo_trilla3));
ind_trilla4=find(dif_trilla4>(umbral_trilla*maximo_trilla4));
ind_trilla5=find(dif_trilla5>(umbral_trilla*maximo_trilla5));
ind_trilla6=find(dif_trilla6>(umbral_trilla*maximo_trilla6));
ind_trilla7=find(dif_trilla7>(umbral_trilla*maximo_trilla7));
ind_trilla8=find(dif_trilla8>(umbral_trilla*maximo_trilla8));
ind_trilla9=find(dif_trilla9>(umbral_trilla*maximo_trilla9));
ind_trilla10=find(dif_trilla10>(umbral_trilla*maximo_trilla10));
ind_trilla11=find(dif_trilla11>(umbral_trilla*maximo_trilla11));
ind_trilla12=find(dif_trilla12>(umbral_trilla*maximo_trilla12));
```

```
prom_trilla=[ind_trilla1; ind_trilla2; ind_trilla3; ind_trilla4;
ind_trilla5; ind_trilla6; ind_trilla7; ind_trilla8; ind_trilla9;
ind_trilla10; ind_trilla11; ind_trilla12];

promedio_trilla=unique(prom_trilla);


%TRILLA EQ
dif_trilla_eq1=abs(prom_clase1-prom_clase4);
dif_trilla_eq2=abs(prom_clase2-prom_clase5);
dif_trilla_eq3=abs(prom_clase3-prom_clase6);
dif_trilla_eq4=abs(prom_clase10-prom_clase13);
dif_trilla_eq5=abs(prom_clase11-prom_clase14);
dif_trilla_eq6=abs(prom_clase12-prom_clase15);

maximo_trilla_eq1=max(var_clase1, var_clase4);
maximo_trilla_eq2=max(var_clase2, var_clase5);
maximo_trilla_eq3=max(var_clase3, var_clase6);
maximo_trilla_eq4=max(var_clase10, var_clase13);
maximo_trilla_eq5=max(var_clase11, var_clase14);
maximo_trilla_eq6=max(var_clase12, var_clase15);


umbral_trilla_eq=4.5;

ind_trilla_eq1=find(dif_trilla_eq1>(umbral_trilla_eq*maximo_trilla_eq1));
ind_trilla_eq2=find(dif_trilla_eq1>(umbral_trilla_eq*maximo_trilla_eq2));
ind_trilla_eq3=find(dif_trilla_eq3>(umbral_trilla_eq*maximo_trilla_eq3));
ind_trilla_eq4=find(dif_trilla_eq4>(umbral_trilla_eq*maximo_trilla_eq4));
ind_trilla_eq5=find(dif_trilla_eq5>(umbral_trilla_eq*maximo_trilla_eq5));
ind_trilla_eq6=find(dif_trilla_eq6>(umbral_trilla_eq*maximo_trilla_eq6));

prom_trilla_eq=[ind_trilla_eq1; ind_trilla_eq2; ind_trilla_eq3;
ind_trilla_eq4; ind_trilla_eq5; ind_trilla_eq6];

promedio_trilla_eq=unique(prom_trilla_eq);


%PICADOR
dif_picador1=abs(prom_clase3-prom_clase1);
dif_picador2=abs(prom_clase3-prom_clase2);
dif_picador3=abs(prom_clase6-prom_clase4);
dif_picador4=abs(prom_clase6-prom_clase5);
dif_picador5=abs(prom_clase9-prom_clase7);
dif_picador6=abs(prom_clase9-prom_clase8);
dif_picador7=abs(prom_clase12-prom_clase10);
dif_picador8=abs(prom_clase12-prom_clase11);
dif_picador9=abs(prom_clase15-prom_clase13);
dif_picador10=abs(prom_clase15-prom_clase14);
dif_picador11=abs(prom_clase18-prom_clase16);
dif_picador12=abs(prom_clase18-prom_clase17);

maximo_picador1=max(var_clase3, var_clase1);
maximo_picador2=max(var_clase3, var_clase2);
maximo_picador3=max(var_clase6, var_clase4);
maximo_picador4=max(var_clase6, var_clase5);
maximo_picador5=max(var_clase9, var_clase7);
maximo_picador6=max(var_clase9, var_clase8);
maximo_picador7=max(var_clase12, var_clase10);
```

```
maximo_picador8=max(var_clase12, var_clase11);
maximo_picador9=max(var_clase15, var_clase13);
maximo_picador10=max(var_clase15, var_clase14);
maximo_picador11=max(var_clase18, var_clase16);
maximo_picador12=max(var_clase18, var_clase17);

umbral_picador=7;

ind_picador1=find(dif_picador1>(umbral_picador*maximo_picador1));
ind_picador2=find(dif_picador1>(umbral_picador*maximo_picador2));
ind_picador3=find(dif_picador3>(umbral_picador*maximo_picador3));
ind_picador4=find(dif_picador4>(umbral_picador*maximo_picador4));
ind_picador5=find(dif_picador5>(umbral_picador*maximo_picador5));
ind_picador6=find(dif_picador6>(umbral_picador*maximo_picador6));
ind_picador7=find(dif_picador7>(umbral_picador*maximo_picador7));
ind_picador8=find(dif_picador8>(umbral_picador*maximo_picador8));
ind_picador9=find(dif_picador9>(umbral_picador*maximo_picador9));
ind_picador10=find(dif_picador10>(umbral_picador*maximo_picador10));
ind_picador11=find(dif_picador11>(umbral_picador*maximo_picador11));
ind_picador12=find(dif_picador12>(umbral_picador*maximo_picador12));

prom_picador=[ind_picador1; ind_picador2; ind_picador3; ind_picador4;
ind_picador5; ind_picador6; ind_picador7; ind_picador8; ind_picador9;
ind_picador10; ind_picador11; ind_picador12];

promedio_picador=unique(prom_picador);


%PICADOR EQ
dif_picador_eq1=abs(prom_clase1-prom_clase2);
dif_picador_eq2=abs(prom_clase4-prom_clase5);
dif_picador_eq3=abs(prom_clase7-prom_clase8);
dif_picador_eq4=abs(prom_clase10-prom_clase11);
dif_picador_eq5=abs(prom_clase13-prom_clase14);
dif_picador_eq6=abs(prom_clase16-prom_clase17);

maximo_picador_eq1=max(var_clase1, var_clase2);
maximo_picador_eq2=max(var_clase4, var_clase5);
maximo_picador_eq3=max(var_clase7, var_clase8);
maximo_picador_eq4=max(var_clase10, var_clase11);
maximo_picador_eq5=max(var_clase13, var_clase14);
maximo_picador_eq6=max(var_clase16, var_clase17);


umbral_picador_eq=6.5;

ind_picador_eq1=find(dif_picador_eq1>(umbral_picador_eq*maximo_picador_eq1)
);
ind_picador_eq2=find(dif_picador_eq1>(umbral_picador_eq*maximo_picador_eq2)
);
ind_picador_eq3=find(dif_picador_eq3>(umbral_picador_eq*maximo_picador_eq3)
);
ind_picador_eq4=find(dif_picador_eq4>(umbral_picador_eq*maximo_picador_eq4)
);
ind_picador_eq5=find(dif_picador_eq5>(umbral_picador_eq*maximo_picador_eq5)
);
ind_picador_eq6=find(dif_picador_eq6>(umbral_picador_eq*maximo_picador_eq6)
);
```

```
prom_picador_eq=[ind_picador_eq1; ind_picador_eq2; ind_picador_eq3;
ind_picador_eq4; ind_picador_eq5; ind_picador_eq6];

promedio_picador_eq=unique(prom_picador_eq);


%Guardamos los datos
vector_ind_presel_frec=promedio_motor;
save 'vector_ind_presel_frec - ind_Motor_sin_suav.mat'
vector_ind_presel_frec

vector_ind_presel_frec=promedio_trilla;
save 'vector_ind_presel_frec - ind_Trilla_sin_suav.mat'
vector_ind_presel_frec

vector_ind_presel_frec=promedio_trilla_eq;
save 'vector_ind_presel_frec - ind_TrillaEq_suav_ch3.mat'
vector_ind_presel_frec

vector_ind_presel_frec=promedio_picador;
save 'vector_ind_presel_frec - ind_Picador_sin_suav.mat'
vector_ind_presel_frec

vector_ind_presel_frec=promedio_picador_eq;
save 'vector_ind_presel_frec - ind_PicadorEq_sin_suav.mat'
vector_ind_presel_frec
```

A1.2 Código del algoritmo Harmony Search

```
% Algoritmo HS con clasificador kNN para las diferentes opciones
% consideradas


%% Inicialización
pause(0.001);tic

close all;clear all;clc;    %Cerramos las ventanas y limpiamos el workspace

load('./../../999 Ruben/DatosFrec.mat');

tipo_clasificador='trillaEq';  % Tipo de clasificador:
% 'motor', 'picador', 'picadorEq', 'trilla', 'trillaEq'

fs=1706;       % Frecuencia de muestreo (Hz)
num_vecinos=5;              % Número de vecinos del clasificador KNN
N_freq=3;
W_conv=5;
canal=3;

XLS_file='HS_Clas_Datos_gen_v4 - resultados.xlsx';
guardarXLS=true;
presel_frec_file='vector_ind_presel_frec - ind_TrillaEq_suav_ch3.mat';


%Suavizado
for i=1:length(medidasEnFrec)
    for j=1:size(medidasEnFrec{i}.ch,2)
```

```
medidasEnFrec{i}.ch(:,j)=conv(medidasEnFrec{i}.ch(:,j),ones(W_conv,1)./W_co
nv,'same');
    end
end


%% Preselección de frecuencias
load(presel_frec_file);
% Genero vector de frecuencias (en Hz)
frec_preselec=(vector_ind_presel_frec-
1).*fs./(2*(size(medidasEnFrec{1}.ch,1)-1));
% Genero variable con datos de trabajo
for i=1:length(medidasEnFrec)
    medidasEnFrec{i}.ch=medidasEnFrec{i}.ch(vector_ind_presel_frec,:);
end

num_vars=size(medidasEnFrec{1}.ch,1); %Inicializamos el número de variables


%% Selección de variables en función del problema a realizar
datos={};
contador=0;
for i=1:length(medidasEnFrec)
    switch tipo_clasificador
        case 'motor'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.motor,'ralenti'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'trilla'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.trilla,'on'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'trillaEq'
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                contador=contador+1;
                datos{contador}= struct('clase',1,'vector',[]);
                datos{contador}.vector = medidasEnFrec{i}.ch(:,canal);
                if(strcmp(medidasEnFrec{i}.trilla_eq,'si'))
                    datos{contador}.clase=1;
                else
                    datos{contador}.clase=2;
                end
            end
        case 'picador'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'picadorEq'
```

```
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                contador=contador+1;
                datos{contador}= struct('clase',1,'vector',[]);
                datos{contador}.vector = medidasEnFrec{i}.ch(:,canal);
                if(strcmp(medidasEnFrec{i}.picador_eq,'si'))
                    datos{contador}.clase=1;
                else
                    datos{contador}.clase=2;
                end
            end
    end
end



%% Inicializo variables de HS y la memoria
mem_size=50;
memoria_HS=zeros(mem_size,N_freq);
fitness_HS=zeros(mem_size,1);
for nn_mem=1:mem_size
    memoria_HS(nn_mem,:)=sort(randperm(num_vars,N_freq));
    FA_aux=false(1,num_vars);
    FA_aux(memoria_HS(nn_mem,:))=true;
    fitness_HS(nn_mem)=clasificador_knn_mejorado(datos,FA_aux,num_vecinos);
end



N_iter=5e3;     % Número de iteraciones del algoritmo
HMCR=0.75;      % Harmony Memory Considering Rate: probabilidad de tomar un
dato anterior o inventarse uno nuevo)
PAR=0.1;        % Pit Adjusting Rate: Probabilidad de cambiar a soluciones
adyacentes
W_PAR=5;        % "Ventana de adyacencia": frecuencias vecinas a las que se
puede mover
N_plot=1e3;     % Número de iteraciones tras las que muestra resultados



% Ordenamos aptitudes y memoria y mostramos por pantalla
[fitness_HS,I_aux]=sort(fitness_HS,'descend');
memoria_HS=memoria_HS(I_aux,:);
fprintf(1,'Iteracion 0\t\t\tfitness max = %.3f\t\t\tfitness mean =
%.3f\n',...
    fitness_HS(1),mean(fitness_HS));


% Vector aleatorio fijo para el fitness, cálculo del rendimiento de HS e
% implementación de diferentes tipos de búsqueda para nueva solución
calcular_rendimiento_HS=true;
if calcular_rendimiento_HS
    tipo_solucion_rendimiento=zeros(N_iter,1);
    memoria_HS_auxiliar=[memoria_HS;-1*ones(N_iter,N_freq)];
end
tipo_busqueda_nueva_sol='global'; % 'local' si busca en la posición o
%                                 % 'global' si busca en toda la memoria
```

```
%% Bucle de iteraciones del algoritmo HS
for nn_i=1:N_iter

    % En cada iteración generamos una nueva solución y vemos si se
    % puede meter en la memoria
    nueva_sol=zeros(1,N_freq);
    memoria_HS_unique_aux=unique(memoria_HS);
    for nn_freq=1:N_freq    % La nueva solución la inicializamos elemento a
elemento
        if rand<HMCR    % Elegimos un valor de esa posición de la memoria
            switch tipo_busqueda_nueva_sol
                case 'local'

nueva_sol(nn_freq)=memoria_HS(randperm(mem_size,1),nn_freq);
                case 'global'

nueva_sol(nn_freq)=memoria_HS_unique_aux(randperm(length(memoria_HS_unique_
aux),1));
            end
        else            % Elegimos un valor de entre todos los posibles
            nueva_sol(nn_freq)=randperm(num_vars,1);
        end
        if rand<PAR     % Aplico PAR para "mutación"
            nueva_sol_PAR_aux=nueva_sol(nn_freq)+randperm(W_PAR*2+1,1)-
W_PAR-1;
            while (nueva_sol_PAR_aux<1)||(nueva_sol_PAR_aux>num_vars)
                nueva_sol_PAR_aux=nueva_sol(nn_freq)+randperm(W_PAR*2+1,1)-
W_PAR-1;
            end
            nueva_sol(nn_freq)=nueva_sol_PAR_aux;
        end

        while length(unique(nueva_sol(1:nn_freq)))<nn_freq  % Si el valor
no es válido seguimos iterando
            if rand<HMCR    % Elegimos un valor de es posición de la
memoria
                switch tipo_busqueda_nueva_sol
                    case 'local'

nueva_sol(nn_freq)=memoria_HS(randperm(mem_size,1),nn_freq);
                    case 'global'

nueva_sol(nn_freq)=memoria_HS_unique_aux(randperm(length(memoria_HS_unique_
aux),1));
                end
            else            % Elegimos un valor de entre todos los posibles
                nueva_sol(nn_freq)=randperm(num_vars,1);
            end
            if rand<PAR     % Aplico PAR para "mutación"
                nueva_sol_PAR_aux=nueva_sol(nn_freq)+randperm(W_PAR*2+1,1)-
W_PAR-1;
                while (nueva_sol_PAR_aux<1)||(nueva_sol_PAR_aux>num_vars)

nueva_sol_PAR_aux=nueva_sol(nn_freq)+randperm(W_PAR*2+1,1)-W_PAR-1;
                end
                nueva_sol(nn_freq)=nueva_sol_PAR_aux;
            end
        end
    end
    nueva_sol=sort(nueva_sol);    % Ordeno la solución
```

```
    % Vector aleatorio fijo + rendimiento      %

    if calcular_rendimiento_HS
        % Si solución repetida en HS
        if min(sum((memoria_HS-repmat(nueva_sol,[mem_size,1])).^2,2))==0
            % solución ya presente en memoria_HS
            tipo_solucion_rendimiento(nn_i)=1;
        elseif min(sum((memoria_HS_auxiliar-repmat(nueva_sol,...
                [size(memoria_HS_auxiliar,1),1])).^2,2))==0
            % solución ya probada anteriormente y no seleccionada
            tipo_solucion_rendimiento(nn_i)=2;
        else
            FA_aux=false(1,num_vars);
            FA_aux(nueva_sol)=true;
            fitness_HS_aux=clasificador_knn_mejorado(...
                datos,FA_aux,num_vecinos);
            if fitness_HS_aux>min(fitness_HS)
                % solución nueva que entra en memoria_HS
                tipo_solucion_rendimiento(nn_i)=3;
            else
                % solución nueva que no entra en memoria_HS
                tipo_solucion_rendimiento(nn_i)=4;
            end
        end
        %
        % Almaceno solución en memoria auxiliar
        memoria_HS_auxiliar(mem_size+nn_i,:)=nueva_sol;
    end



    % Miro si la solución está repetida (si está no hago nada)
    if min(sum((memoria_HS-repmat(nueva_sol,[mem_size,1])).^2,2))>0
        % Evalúo la nueva solución
        FA_aux=false(1,num_vars);
        FA_aux(nueva_sol)=true;



        % Vector aleatorio fijo en clasificador
        fitness_HS(end+1)=clasificador_knn_mejorado(...
            datos,FA_aux,num_vecinos);



        memoria_HS(end+1,:)=nueva_sol;
        % Me quedo con la nueva solución si es mejor que alguna de la
memoria
        [fitness_HS,I_aux]=sort(fitness_HS,'descend');
        memoria_HS=memoria_HS(I_aux(1:end-1),:);
        fitness_HS=fitness_HS(1:end-1);
    end

    % Muestro evolución por pantalla
    if mod(nn_i,N_plot)==0
        fprintf(1,'Iter. %8.0f\t\tfitness max = %.3f\t\tmean = %.3f\t\tmin
= %.3f\t\t%.4f seg.\n',...
            nn_i,fitness_HS(1),mean(fitness_HS),fitness_HS(end),toc);
    end
end
```

```
%% Muestro y almaceno resultados
if calcular_rendimiento_HS
    memoria_HS;

[fitness_HS(1:5),zeros(5,1),memoria_HS(1:5,:),zeros(5,1),frec_preselec(memo
ria_HS(1:5,:))]

bar([sum(tipo_solucion_rendimiento==1),sum(tipo_solucion_rendimiento==2),su
m(tipo_solucion_rendimiento==3),sum(tipo_solucion_rendimiento==4)]./length(
tipo_solucion_rendimiento))
    set(gca,'XtickLabel',{'sol. en mem HS','sol. repetida','sol. nueva
buena','sol. nueva mala'})
end

toc

if guardarXLS
    if calcular_rendimiento_HS

datos_XLS={datestr(datetime),tipo_clasificador,presel_frec_file,num_vars,nu
m_vecinos,mem_size,N_iter,HMCR,PAR,W_PAR,sum(tipo_solucion_rendimiento==1)/
N_iter,sum(tipo_solucion_rendimiento==2)/N_iter,sum(tipo_solucion_rendimien
to==3)/N_iter,sum(tipo_solucion_rendimiento==4)/N_iter,N_freq,fitness_HS(1)
};
    else

datos_XLS={datestr(datetime),tipo_clasificador,presel_frec_file,num_vars,nu
m_vecinos,mem_size,N_iter,HMCR,PAR,W_PAR,'','','','',N_freq,fitness_HS(1)};
    end
    for nn=1:size(memoria_HS,2)
        datos_XLS{end+1}=frec_preselec(memoria_HS(1,nn));
    end
    xlswrite(XLS_file,datos_XLS,'resultados
HS',sprintf('A%d',size(xlsread(XLS_file,'resultados HS'),1)+3));
end

toc
```

A1.3 Código de la Fuerza Bruta

```
% Código equivalente al del algoritmo pero probando todas las
% opciones por fuerza bruta


%% Inicialización
pause(0.001);tic

close all;clear all;clc;    %Cerramos las ventanas y limpiamos el workspace

load('./../../999 Ruben/DatosFrec.mat');


tipo_clasificador='picador';  % Tipo de clasificador:
% 'motor', 'picador', 'picadorEq', 'trilla', 'trillaEq'

fs=1706.665210312354;       % Frecuencia de muestreo (Hz)
num_vecinos=5;              % Número de vecinos del clasificador KNN
N_freq=2;
```

```
W_conv=5;
canal=1;

XLS_file='fuerzaBruta.xlsx';
guardarXLS=true;
presel_frec_file='vector_ind_presel_frec - ind_Picador_suav_ch1.mat';


%Suavizado
for i=1:length(medidasEnFrec)
    for j=1:size(medidasEnFrec{i}.ch,2)

medidasEnFrec{i}.ch(:,j)=conv(medidasEnFrec{i}.ch(:,j),ones(W_conv,1)./W_co
nv,'same');
    end
end


%% Preselección de frecuencias
load(presel_frec_file);
% Genero vector de frecuencias (en Hz)
frec_preselec=(vector_ind_presel_frec-
1).*fs./(2*(size(medidasEnFrec{1}.ch,1)-1));
% Genero variable con datos de trabajo
for i=1:length(medidasEnFrec)
    medidasEnFrec{i}.ch=medidasEnFrec{i}.ch(vector_ind_presel_frec,:);
end

num_vars=size(medidasEnFrec{1}.ch,1); %Inicializamos el número de variables


%% Selección de variables en función del problema a realizar
datos={};
contador=0;
for i=1:length(medidasEnFrec)
    switch tipo_clasificador
        case 'motor'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.motor,'ralenti'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'trilla'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.trilla,'on'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'trillaEq'
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                contador=contador+1;
                datos{contador}= struct('clase',1,'vector',[]);
                datos{contador}.vector = medidasEnFrec{i}.ch(:,canal);
                if(strcmp(medidasEnFrec{i}.trilla_eq,'si'))
                    datos{contador}.clase=1;
                else
```

```
                    datos{contador}.clase=2;
                end
            end
        case 'picador'
            datos{i}= struct('clase',1,'vector',[]);
            datos{i}.vector = medidasEnFrec{i}.ch(:,canal);
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                datos{i}.clase=1;
            else
                datos{i}.clase=2;
            end
        case 'picadorEq'
            if(strcmp(medidasEnFrec{i}.picador,'on'))
                contador=contador+1;
                datos{contador}= struct('clase',1,'vector',[]);
                datos{contador}.vector = medidasEnFrec{i}.ch(:,canal);
                if(strcmp(medidasEnFrec{i}.picador_eq,'si'))
                    datos{contador}.clase=1;
                else
                    datos{contador}.clase=2;
                end
            end
    end
end



%% Inicializo variables para almacenar soluciones
N_soluciones=nchoosek(num_vars,N_freq);
memoria_fuerzaBruta=zeros(N_soluciones,N_freq);
fitness_fuerzaBruta=zeros(N_soluciones,1);



%% Bucle de iteraciones del algoritmo de Fuerza Bruta
N_FB=num_vars;
k_FB=N_freq;
nueva_sol=1:k_FB;
columna_FB=k_FB;
fila_FB=1;

% Evaluar nueva_sol
FA_aux=false(1,num_vars);
FA_aux(nueva_sol)=true;
fitness_fuerzaBruta(fila_FB)=clasificador_knn_mejorado(datos,FA_aux,num_vec
inos);
memoria_fuerzaBruta(fila_FB,:)=frec_preselec(nueva_sol);

% Iterar
t_aux=0;
delta_t=60; % tiempo: cada cuanto te muestra la información (en seg.)
while sum(nueva_sol)<sum(N_FB-k_FB+1:N_FB)
    if nueva_sol(columna_FB)<N_FB+columna_FB-k_FB
        nueva_sol(columna_FB)=nueva_sol(columna_FB)+1;
        nueva_sol(columna_FB+1:k_FB)=nueva_sol(columna_FB)+(1:k_FB-
columna_FB);
        columna_FB=k_FB;
        fila_FB=fila_FB+1;
        % Evaluar nueva_sol
        FA_aux=false(1,num_vars);
```

```
        FA_aux(nueva_sol)=true;

fitness_fuerzaBruta(fila_FB)=clasificador_knn_mejorado(datos,FA_aux,num_vec
inos);
        memoria_fuerzaBruta(fila_FB,:)=frec_preselec(nueva_sol);
        if toc-t_aux>delta_t
            fprintf(1,'%2.0f%%\t%.0f
min\n',100*fila_FB/N_soluciones,toc/60);
            t_aux=toc;
        end
    else
        columna_FB=columna_FB-1;
    end
end
tiempo_fin=toc;



% Escribir en XLS
[~,sheets_XLS]=xlsfinfo(XLS_file);
xlswrite(XLS_file,{'Fecha/hora';'Tipo clasif.';'nombre fichero - frec.
presel.';'num. frec. - frec. presel.';'num. vecinos (KNN)';'num. frec.
sol.';'tiempo ejec. (min)'},length(sheets_XLS)+1,'A1');
xlswrite(XLS_file,{datestr(datetime);tipo_clasificador;presel_frec_file;num
_vars;num_vecinos;N_freq;tiempo_fin/60},length(sheets_XLS)+1,'B1');
xlswrite(XLS_file,{'fitness','frecuencias'},length(sheets_XLS)+1,'A10');
xlswrite(XLS_file,fitness_fuerzaBruta,length(sheets_XLS)+1,'A11');
xlswrite(XLS_file,memoria_fuerzaBruta,length(sheets_XLS)+1,'B11');
toc
```

A1.4 Código del clasificador kNN

```
%Clasificador kNN con Leave One Out

function [tasa_exito] =
clasificador_knn_mejorado(datos,variables_seleccionadas,num_vecinos)

%Calculamos la longitud de los datos de entrada
indice_final=length(datos);
tasa_exito=0;
% Declaramos un parámetro para realizar Leave-N-Out
lNo=1;

X = zeros(length(datos),sum(variables_seleccionadas));
Y = zeros(length(datos),1);
for i=1:length(datos)
    X(i,:) = datos{i}.vector(variables_seleccionadas);
    Y(i,:) = datos{i}.clase;
end

for j=1:indice_final/lNo
    indices_testear=1+lNo*(j-1):lNo*j;
    indices_testear2=false(indice_final,1);
    indices_testear2(indices_testear)=true;

    Xtrain=X(~indices_testear2,:);
    Ytrain=Y(~indices_testear2,:);
```

```
    Xnew = X(indices_testear,:);
    Xnew=repmat(Xnew,[indice_final-1 1]);

    distancia=sum((Xtrain-Xnew).^2,2);

    [distancia_ordenada,indices_orden]=sort(distancia);

    labelPredicted=mode(Ytrain(indices_orden(1:num_vecinos)));

    labelTrue = Y(indices_testear,:);

    %Calculamos los éxitos y fracasos de la clasificación
    tasa_exito2=sum(labelPredicted==labelTrue)/length(labelPredicted);

    %Promediamos el tanto por uno de éxitos para obtener la CVA usando
    %Leave-One-Out
    tasa_exito=(j-1)/j*tasa_exito+1/j*tasa_exito2;

end
tasa_exito;

end
```