Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Optimización de un controlador MPC mediante algoritmos metaheurísticos

Autor: Luis Miguel Martínez Gómez

Tutor/es: Iván García Daza

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado
Optimización de un controlador MPC mediante algoritmos metaheurísticos

Autor: Luis Miguel Martínez Gómez

Tutor/es: Iván García Daza

TRIBUNAL:

Presidente: Javier Alonso Ruiz

Vocal 1º: Francisco Manuel Márquez García

Vocal 2º: Iván García Daza

FECHA: 20/09/2021

Agradecimientos

A mis padres, por su apoyo continuado y sus lecciones de vida.
Al profesor Iván García Daza, por su ayuda para encontrar el camino adecuado.
A Carmen, por su inestimable compañía y cariño.
Y por último, a mí mismo, por haber luchado durante años.

Índice

Lista de figuras

Lista de algoritmos

Lista de acrónimos

ACO Ant Colony Optimization. 29, 33

BCO Bound Constrained Optimization. 27

BOA Base Optimization Algorithm. 30

CLONALG Clonal Selection Algorithm. 30

COBYLA Constrained Optimization by Linear Aproximation. 28, 54, 55

CRHPC Constrained Receding-Horizon Predictive Control. 4

CRO Chemical Reaction Optimization. 30

DMC Dynamic Matrix Control. 3

DTS Dynamic Tabu Search. 33

GA Genetic Algorithm. 33

GBMO Gases Brownian Motion Optimization. 30

GD Gradient Descent. 54

GLS Guided Local Search. 33

GP Gradient Projection. 27

GPU Graphic Processing Unit. 57

GSA Gravitation Search Algorithm. 31

HS Harmony Search. 30

ILS Iterated Local Search. 33

INVETT Intelligent Vehicles and Traffic Technologies. 41

IPM Interior-Point Method. 27

IRM Impulse Response Model. 8, 9, 11, 12

LCA League Championship Algorithm. 31

LP Linear Programming. 27

MLP MultiLayer Perceptron Network. 12

MMC Method of Music Composition. 30

MPC Model Predictive Control. 1–4, 6, 7, 14, 16, 41–44, 46, 55, 56

MPHC Model Predictive Heuristic Control. 3

NFLT No Free-Lunch Theorem. 21, 56, 57

NNARMAX Neural Network Nonlinear AutoRegressive, Moving Average, eXternal input. 13, 14

NNARX Neural network Nonlinear AutoRegressive, eXternal input. 13, 14

NNFIR Neural network Nonlinear Finite Impulse Response. 13

NNOE Neural network Nonlinear Output Error. 14

NP Nonlinear Programming. 28

POPMUSIC Partial Optimization Metaheuristic Under Special Intensification Conditions. 31

PPSO Phasor Particle Swarm Optimization. 37, 38, 40

PSO Particle Swarm Optimization. 1, 21, 29, 31, 33, 34, 37, 38, 46, 54–57

QP Quadratic Programming. 28, 43

RMPC Robust Mutlivariable Predictive Control. 4

SA Simulated Annealing. 31, 33

SCA Sine Cosine Algorithm. 31

SIORHC Stabilized Input/Output Receding Horizon Control. 4

SLP Sequential Linear Programming. 27, 28

SM Simplex Method. 27

SQP Sequential Quadratic Programming. 28

SRM Step Response Model. 8, 9, 11, 12

SSM State Space Model. 11, 12, 43

TFM Transfer Function Model. 9

TPU Tensor Processing Unit. 57

TS Tabu Search. 31, 33

VNPSO Variable Neighbourhood Particle Swarm Optimization. 33

VNS Variable Neighbourhood Search. 31, 33

WOA Whale Optimization Algorithm. 21

Resumen

El control por predicción de modelo (MPC) es una de las estrategias de control con mayor proyección de futuro por su utilidad en el mundo industrial. Uno de los problemas del control MPC es el de optimizar una función de coste que puede ser no convexa, lo que limita su aplicación. La metaheurística, una rama de la optimización, es conocida por su habilidad para manejar problemas con gran número de variables y alta complejidad. En este trabajo se explora la posibilidad de mejorar la etapa de optimización del controlador MPC empleando un algoritmo metaheurístico concreto, Particle Swarm Optimization.

Palabras clave: ingeniería de control, MPC, optimización, metaheurística, Particle Swarm Optimization.

Abstract

Model Predictive Control (MPC) is one of the most promising control strategies given its usefulness in industrial environments. However, it is heavily set back by the necessity of optimizing a non-convex function during its workflow, which can severly limit its capabilities. Metaheuristics is a sub-branch of optimization, widely known to be able to tackle high complexity and large problems. This work explores the posibility of enhancing the optimization stage of an MPC controller by implementing a specific metaheuristic algorithm, Particle Swarm Optimization.

Keywords: control engineering, MPC, optimization, metaheuristics, Particle Swarm Optimization.

1. Introducción.

La ingeniería de control moderna es, sin duda alguna, una de las disciplinas ingenieriles más importantes del mundo moderno. Ya sea en entornos industriales o sociales, es difícil encontrar una máquina, proceso o situación no esté gobernada por un algoritmo de control desarrollado de forma específica para satisfacer una necesidad o resolver un problema. Desde la producción alimenticia a los medios de transporte, pasando por las telecomunicaciones, la fabricación de medicamentos o las transacciones financieras, todas las actividades que realiza el ser humano hoy en día tienen un denominador común en que son procesos sujetos a modelización y control.

Durante los últimos 200 años se han desarrollado diversas y numerosas aproximaciones sistemáticas de control. En este trabajo de final de grado se explorará en detalle una de las más recientes: el Control por Predicción de Modelo, o MPC por sus siglas en inglés. Esta estrategia de control, surgida a finales de los años setenta, tiene el potencial de ser una de las más usadas en las próximas décadas, ya que su estructura y principio de trabajo permite su aplicación a todo tipo de procesos, sin importar su complejidad.

Por desgracia, uno de los límites del control MPC viene fuertemente determinado por la eficiencia de los procesos de optimización de funciones de los que disponemos hoy en día. Dentro del flujo de trabajo del controlador se encuentra la minimización de una función de coste que determina las secuencias de control a ejecutar, y es este proceso el que determina si un controlador MPC puede ser diseñado o no para una determinada aplicación, teniendo en cuenta principalmente las constricciones temporales que haya que tener en cuenta en la misma.

Es por ello que, además de detallar en qué consiste la mencionada estrategia de control, en este trabajo se presenta también la metaheurística, una disciplina albergada dentro de la optimización que se basa en la adaptación de comportamientos observados en diversas fuentes de inspiración con el objetivo de plantear nuevos y potentes métodos de optimización de funciones que no pueden ser atacadas de forma sistemática o determinista. La metaheurística es también una disciplina de futuro, ya que su reciente desarrollo -hacia mediados de los años 90, aunque hay algunos trabajos anteriores-, su curiosa fuente de inspiración y su probada efectividad la han convertido en uno de los tópicos más interesantes de la ciencia de computación.

En este documento se explora lo que se ha identificado como un nexo común entre el control MPC, que presenta una debilidad clara en su etapa de minimización, y la metaheurística, que se ofrece como una alternativa a las formas tradicionales de resolver los problemas de optimización más complejos; para ello, se estudiará la implementación del algoritmo Particle Swarm Optimization (PSO), considerado uno

de los pioneros dentro de la disciplina metaheurística, dentro de la arquitectura de un controlador MPC dedicado a un vehículo autónomo.

El trabajo está organizado como sigue: la sección 2 establece las bases teóricas del control MPC y sus posibles ventajas o desventajas con respecto a otros arquetipos de control. En la sección 3 se estudia la metaheurística como disciplina matemática, así como los diferentes tipos de problemas de optimización que se pueden resolver y cómo se caracterizan. La sección 4 presenta un estudio del estado del arte de la aplicación de los controladores MPC al ámbito de la conducción autónoma, y este es acompañado de una implementación propia de tal controlador con la modificación propuesta en el apartado anterior. Por último, la sección 5 discute los resultados del trabajo e identifica unas posibles líneas de investigación futuras.

2. Control por Predicción de Modelo (MPC)

El objetivo de esta sección es el de introducir la estrategia Model Predictive Control (MPC) como método de control para sistemas de complejidad variable. Para ello, se ofrecerá una somera introducción de alto nivel, acompañada de una breve descripción del desarrollo histórico de esta disciplina de control. Seguidamente se ofrecerá una descripción comprensiva de los aspectos principales que componen esta aproximación de control. Una vez asentadas las ideas fundamentales y el papel que juegan dentro de la configuración del sistema, en una sección posterior, se estudiará un problema concreto solucionado mediante un controlador MPC, como ejemplo de la polivalencia de esta aproximación.

2.1. Origen y evolución del control MPC

El origen de la estrategia MPC surge a finales de los años 70 como respuesta a las necesidades de la industria petroquímica de mejorar los procesos de refinado del petróleo. Las primeras aproximaciones aparecen en la literatura como solución al problema del control óptimo de tiempo mínimo y están íntimamente relacionadas con la programación lineal; son las conocidas como Model Predictive Heuristic Control (MPHC) [41] y Dynamic Matrix Control (DMC) [7]. Ambas estrategias están basadas en un modelo dinámico del proceso a controlar (respuesta a impulso y a escalón, respectivamente) para predecir el efecto de las acciones de control futuras sobre la salida del sistema. Dichas acciones de control son determinadas minimizando el error estimado, calculado como la diferencia entre la referencia a seguir y las salidas proyectadas. Para ello, se emplea la estrategia del *horizonte recesivo*, propuesta por Propoi en 1963 [39].

Estas estrategias, cuyo desarrollo fue impulsado por los avances en computación digital en la época, fueron rápidamente incorporadas dentro de los procesos de la industria petrolera debido a la simplicidad de su implementación y al uso de modelos basados en la respuesta a impulso o escalón, intuitivos y asentados en el mundo industrial como una forma sencilla y efectiva de representar una planta a controlar [15]. Cabe destacar que, ya en los albores del desarrollo del control MPC, los desarrollos empleados eran capaces de manejar sistemas multivariable con alta complejidad, siempre que las constricciones temporales no fueran una parte fundamental de las prestaciones del controlador.

Uno de los problemas principales del empleo de modelos bien asentados en la industria es la falta de fundamento formal de los mismos, en el sentido en que es prácticamente imposible determinar la estabilidad y la robustez del sistema salvo que el proceso a controlar y sus restricciones sean particularmente benignas. Como respuesta a esto surge la reformulación del control MPC en modelo de espacio de

estados [36], lo que permite emplear los teoremas clásicos de estabilidad y robustez, así como extender de forma efectiva y con un fundamento teórico sólido la estrategia MPC a sistemas no lineales o con perturbaciones estocásticas y ruido en las medidas. Esto permite concebir un controlador MPC como un compensador basado en un observador de estados cuya estabilidad y robustez vienen determinadas por los polos del observador y del regulador, que son parámetros fácilmente modificables dentro de la arquitectura del controlador [4].

Durante los años noventa aparecen dos nuevos métodos, el Constrained Receding-Horizon Predictive Control (CRHPC) [5] y el Stabilized Input/Output Receding Horizon Control (SIORHC) [37]; estos métodos aseguran la estabilidad del sistema imponiendo condiciones sobre la salida estimada a lo largo del horizonte finito de predicción. Estos resultados son especialmente importantes, ya que aunque el optimizador del controlador encuentre soluciones óptimas (es decir, alcance el mínimo de la función de error), no está garantizado que estas acciones de control estabilicen el sistema. En otras palabras, la solución de control óptima no tiene por qué ser estable [4]. En estas dos nuevas estrategias propuestas se incluyen restricciones sobre la salida, pero pueden emplearse otros mecanismos como funciones de Lyapunov o conjuntos invariantes. Mayne et al. resumen en [32] las condiciones generales de diseño de un controlador MPC estable.

En lo que respecta a la robustez del controlador se han obtenido resultados con aproximaciones en las que las incertidumbres e interferencias se modelan de forma explícita dentro del proceso, de forma que el optimizador encuentra la solución adecuada para el peor caso de dichas incertidumbres. En cualquier caso, durante los próximos años se realizarán cuantiosos avances en este aspecto, habida cuenta de la buena acogida del control MPC en la industria. La compañía Honeywell, por medio de su tecnología patentada Robust Mutlivariable Predictive Control (RMPC) ha promovido desde principios de siglo una solución comercial con robustez suficiente.

Como puede observarse en los datos aportados por Kozák en [27], hasta 2010 se reportan unas 2500 soluciones industriales proporcionadas por empresas del sector de la ingeniería de control. En la actualidad, este paradigma de trabajo está sufriendo una evolución casi constante, relacionada con los grandes saltos en potencia de cálculo vistos en la última década y con la interrelación con otras ramas de la computación, como la inteligencia artificial, las redes neuronales o la ciencia de datos.

2.2. Ideas principales

El control por predicción de modelo MPC no designa a una estrategia de control específica, sino a una arquitectura flexible de trabajo que permite el uso de

un modelo explícito para adaptar una señal de control a una referencia sin limitarse a la información del pasado. Sus principales características son [4]:

- Empleo de un modelo matemático del proceso para predecir su salida en un horizonte finito en el futuro.
- Cálculo de una secuencia de control que minimiza una función objetivo (en general cuadrática), y
- Empleo de la metodología del horizonte recesivo: en cada instante el horizonte de predicción se desplaza hacia el futuro, lo que supone la aplicación de la señal de control generada en el instante anterior.

El proceso de control, desde un punto de vista general, es como sigue [4]:

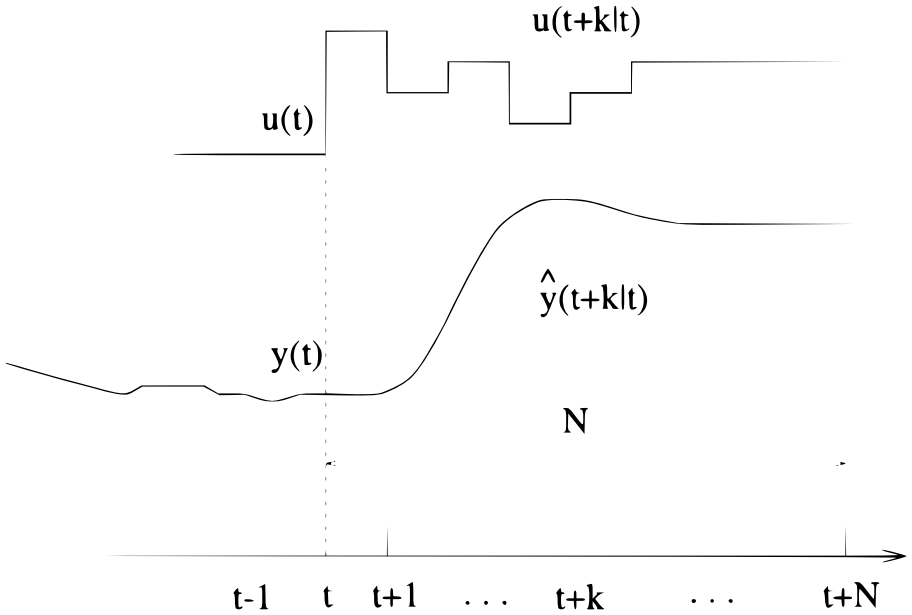

Figura 2.1: Representacion del control MPC
Fuente: [4]

1. En el instante t, se predicen las salidas del sistema de control para un horizonte de predicción N_p usando el modelo incorporado en la estructura del controlador. Para ello, las salidas $\hat{y}(t+k|\,t)$, $k = 1, \ldots, N$ [1] se calculan empleando la información conocida hasta el instante t (entradas y salidas), así como las futuras señales de control a enviar al sistema, $u(t+k|\,t)$, $k = 0, \ldots, N_p - 1$. En algunas implementaciones, los valores N y N_p difieren, ya que se asume que a partir de cierto instante en el tiempo la señal de control se mantiene constante ($N > N_p$).

[1] La notación indica la salida y predecida en el instante t para el instante $t + k$

2. El conjunto de señales futuras de control se calcula mediante la optimización de un criterio generalmente cuadrático (conocido como *función de coste, error* o *pérdida*), que tiene en cuenta, en su forma más simple, la diferencia entre las salidas predecidas $\hat{y}(t+k|\,t)$ y la referencia a seguir $w(t+k)$. Si el criterio es cuadrático, el modelo es lineal y el proceso no posee restricción alguna, puede encontrarse una solución analítica al problema, lo que hace del proceso de optimización de las secuencias de control una mera evaluación de una función determinada. Por desgracia, la mayoría de procesos en los que el control MPC es de aplicación no cumplen estos requisitos, por lo que es necesario emplear un método iterativo para minimizar el criterio de desempeño. Versiones más complejas de este criterio pueden incluir el esfuerzo de control o la diferencia entre acciones de control como elementos a minimizar, como se verá en una sección posterior.

3. Se envían las señales de control óptimas para el horizonte de control N_u $u(t+k|\,t)$, $k = 0, \ldots, N_u$. Cabe destacar que el horizonte de control debe ser comparativamente pequeño con respecto al de predicción; en caso contrario se estarían enviando señales de control calculadas a partir de predicciones con un error acumulado que puede hacerse lo suficientemente grande como para desestabilizar o saturar el sistema, o simplemente alejarse de forma considerable de la referencia a seguir. Cuanto más grande sea el horizonte de control, menor carga computacional se impone sobre el sistema, ya que el algoritmo de optimización y generación de señales de control debe ejecutarse menos veces, pero mayor es el error acorde al criterio de evaluación del controlador. Por ello, es necesario conseguir una solución de compromiso. En sistemas de altas prestaciones es común que el optimizador se ejecute en cada iteración, de forma que la única acción de control que se emplea de las calculadas en el paso anterior sea la primera.

El empleo de esta estrategia tiene una serie de ventajas con respecto a métodos más tradicionales de control. Entre ellas, destaca su flexibilidad, ya que puede adaptarse con facilidad tanto a procesos de dinámica simple como compleja, en la medida en que es suficiente con disponer de un modelo que refleje de forma fiel dichas dinámicas. Una consecuencia de esto es que la estrategia MPC puede aplicarse a procesos multivariable o con tiempos muertos.

Como cualquier otra aproximación, también tiene una serie de desventajas que deben ser consideradas a la hora de evaluar si el control MPC es la mejor estrategia para afrontar un proceso. La más importante de estas desventajas sea, quizá, la necesidad específica de un modelo de la planta a controlar. Esto obliga a realizar un proceso de identificación que puede ser costoso, poco preciso y en general

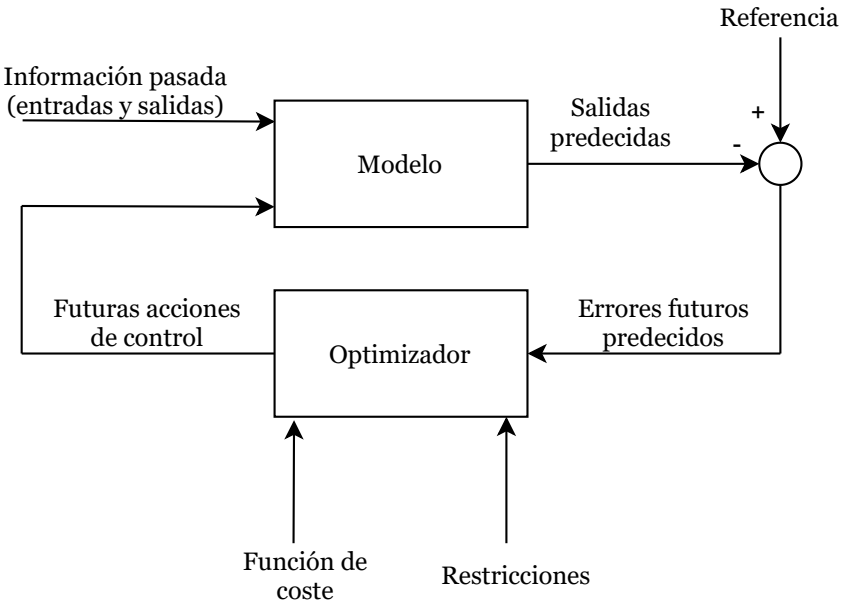

Figura 2.2: Esquema básico de un controlador MPC
Fuente: Adaptada de [4]

inconveniente. Por otro lado, entendiendo que este tipo de control tiene como objetivo afrontar dinámicas complejas y ofrecer un grado de adaptatividad importante a cambios temporales en el proceso, hay que destacar la carga computacional que supone resolver un problema de optimización con restricciones, en el peor de los casos en cada iteración de control. Esto reduce de forma drástica el ancho de banda de los procesos a los que el control MPC es aplicable. En definitiva, salvo que se cuente con una implementación rápida o con un *framework* adaptado, es complicado aplicar este control a procesos con constantes de tiempo o periodos de muestreo reducidos.

Camacho y Bordons ofrecen una de las demostraciones más intuitivas detrás de los conceptos del control MPC en [4], comparándolo al proceso que sigue un conductor para manejar un coche. El conductor conoce la trayectoria que desea seguir, así como la que ha seguido y las acciones de control (pedales, marchas, volante) que ha ejecutado hasta el momento. A base de evaluar mentalmente qué ocurrirá conforme mueva el volante de una forma u otra, o pise el pedal del acelerador más o menos a fondo, es capaz de determinar una nueva serie de acciones que se ajustan de forma fiel a la trayectoria deseada. En este caso, el muestreo del modelo del coche y la optimización se realizan mentalmente en tiempo real, como consecuencia del funcionamiento heurístico del pensamiento humano y la extraordinaria capacidad para evaluar situaciones que se destila del aprendizaje y la experiencia. Este ejemplo tomará relevancia en las secciones posteriores.

2.3. Estructura del controlador

Como puede observarse en la figura 2.2, el controlador tiene dos componentes principales, el modelo y el optimizador, en el que se configura el criterio de rendimiento. En esta sección se analizarán el modelo y la función de coste. El proceso de optimización será discutido en la sección dedicada a la metaheurística.

2.3.1. Tipos de modelo empleados en el control MPC

Como ha sido comentado anteriormente, la necesidad del modelo en este arquetipo de control viene dada por el empleo de la estrategia de horizonte recesivo, como mecanismo de predicción de la respuesta del sistema en base a salidas y secuencias de control medidas y predecidas.

En primer lugar es conveniente dar una definición cerrada y concisa de modelo. Será tal cualquier constructo matemático del que, tras ser excitado con una o varias señales de entrada, pueden ser extraídas unas salidas que reflejan la dinámica de un sistema con una precisión arbitraria. En principio, el modelo puede ser tan complejo como se desee, pero suele ser necesario alcanzar un compromiso entre la fidelidad de sus salidas y el número de parámetros necesario para describirlas. En lo que respecta al control MPC, suelen emplearse modelos discretos con un periodo de muestreo muy inferior a las constantes de tiempo que determinan el comportamiento del sistema. Téngase en cuenta que no se ha hecho distinción alguna acerca de la tipología del modelo: pueden emplearse modelos lineales, no lineales, multivariables e incluso de caja negra.

El tipo de modelo conceptualmente más sencillo es el Impulse Response Model (IRM), el modelo basado en la respuesta a impulso de la planta. Es un modelo ampliamente aceptado en la industria, ya que es fácil describir e interiorizar qué influencia tiene la modificación de cualquier parámetro del mismo, pero el gran número de parámetros a obtener y su incapacidad para representar procesos inestables suponen desventajas considerables. La respuesta del modelo se obtiene convolucionando la señal de entrada con el polinomio de coeficientes [4]:

$$y(t) = \sum_{j=1}^{N} h_j u(t-i) = H(z^{-1})u(t) \tag{2.1}$$

donde N es el número de parámetros del modelo, h_j es la respuesta medida cuando la planta es excitada por una delta de Kronecker y $H(z^{-1}) = h_1 z^{-1} + h_2 z^{-2} + \ldots + h_N z^{-N}$, donde z^{-1} es el operador de retardo, de forma que $z^{-1}u(t) = u(t-1)$. Una representación del modelado IRM puede verse en la figura 2.3.

Otro modelo ampliamente adoptado en la industria es el Step Response Model (SRM), el modelado mediante respuesta a escalón. Es similar al IRM, pero la señal

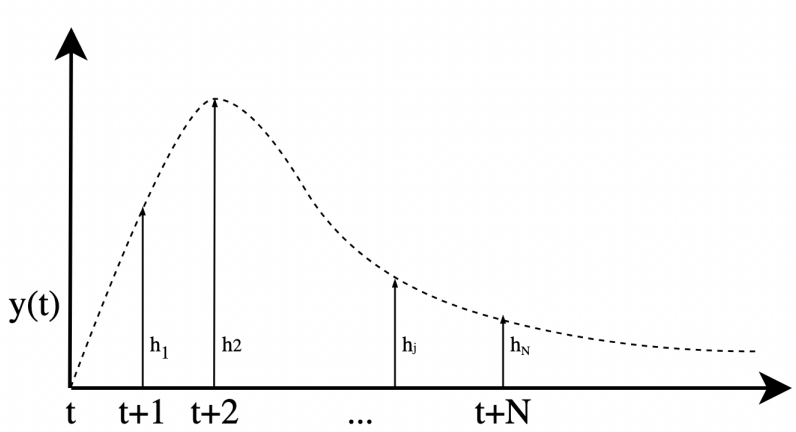

Figura 2.3: Modelado mediante respuesta a impulso
Fuente: Adaptada de [4]

de excitación del sistema es un escalón en vez de un impulso. Cuenta con las mismas ventajas y desventajas, entre las que destacan la imposibilidad de modelar procesos inestables y la gran cantidad de parámetros necesarios para obtener una respuesta fiel.

El modelo se formula con la siguiente ecuación:

$$y(t) = \sum_{j=1}^{N} g_j[u(t-j+1) - u(t-j)] = (1-z^{-1})G(z^{-1})u(t) \tag{2.2}$$

donde N es el número de parámetros, g_j es la respuesta medida ante una entrada escalón $u(t)$ en el instante $t+j$, y $G(z^{-1}) = g_1 z^{-1} + g_2 z^{-2} + \ldots + g_N z^{-N}$. La figura 2.4 es una representación de la obtención de los parámetros del modelo SRM

Si se considera un impulso unitario como una diferencia de dos escalones de sentidos contrarios desfasados un periodo de muestreo, los modelos IRM y SRM pueden relacionarse mediante:

$$h_j = g_j - g_{j-1} \qquad\qquad g_i = \sum_{j=1}^{i} h_j \quad i = 1, \ldots, N$$

Dentro de la teoría de control, una de las formas tradicionales de representar los sistemas y procesos es la función de transferencia. El Transfer Function Model (TFM) se basa en el empleo de polinomios para representar las dinámicas del proceso a modelar; estos polinomios son obtenidos mediante la transformada de Laplace (para sistemas continuos descritos mediante ecuaciones diferenciales) o la transformada Z (para sistemas discretos descritos mediante ecuaciones en diferencias). Si, por otro lado, no se dispone de una formulación del sistema en el dominio temporal,

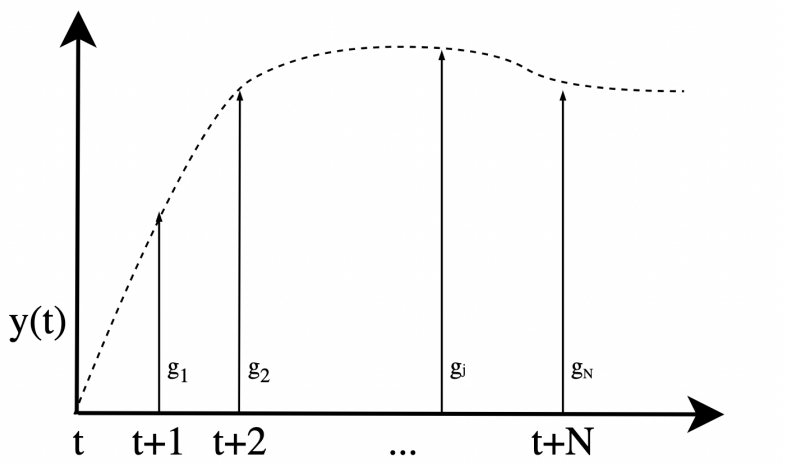

Figura 2.4: Modelado mediante respuesta a escalón
Fuente: Adaptada de [4]

pueden obtenerse mediante métodos de identificación de sistemas, como los modelos autorregresivos.

La salida del modelo viene dada, en su forma más simple, por [4]:

$$A(z^{-1})y(t) = B(z^{-1})u(t) \tag{2.3}$$

donde

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na}$$
$$B(z^{-1}) = b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{nb} z^{-nb}$$

La predicción, por tanto, se obtiene mediante:

$$\hat{y}(t+k|\,t) = \frac{B(z^{-1})}{A(z^{-1})} u(t+k|\,t) \tag{2.4}$$

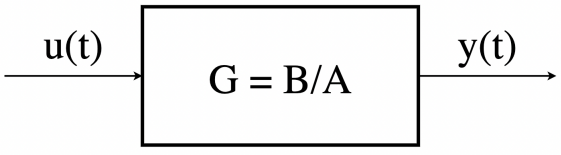

Figura 2.5: Esquema del modelo por función de transferencia

La representación por función de transferencia tiene la ventaja de poder representar sistemas inestables. Cabe destacar que, aunque la formulación parece algo compleja, es común que sistemas comunes en la industria puedan aproximarse de forma suficientemente fiel con polinomios de orden bajo (generalmente, no superior

a 4). Esto reduce de forma drástica el número de parámetros a determinar con respecto a los modelos IRM y SRM. Esta cualidad también es la principal desventaja de esta aproximación: es necesario conocer de antemano el grado de los polinomios A y B, pudiéndose obtener un modelo impreciso si no se determinan con rigurosidad. Además, la inclusión de tiempos muertos y retardos dentro del modelo requiere de una implementación más compleja.

El otro modelo que domina la literatura y la teoría de los sistemas de control es el State Space Model (SSM), o modelado mediante espacio de estados. La formulación del mismo es la siguiente:

$$x(t) = Ax(t-1) + Bu(t-1) \tag{2.5}$$

$$y(t) = Cx(t) \tag{2.6}$$

donde $x(t)$ es el *estado* del sistema, y A, B y C son las matrices de estado, entrada y salida, respectivamente. La formulación matricial permite extender las actuaciones de control a sistemas multivariable sin imponer complejidad adicional al sistema. Además, los teoremas de estabilidad y robustez son fácilmente aplicables a estos modelos, lo que facilita el análisis teórico preliminar a la implementación de una estrategia de control. Normalmente, las matrices se derivan de las ecuaciones diferenciales que representan el proceso a modelar, eligiendo una serie de variables temporales como estados, a partir de las cuales se describe la evolución del sistema. No obstante, es posible que las componentes del vector de estados no tengan un significante físico, haciendo que el modelo sea menos intuitivo y requiera de un análisis formal para determinar su validez. Por otro lado, la formulación propuesta en las ecuaciones 2.5 y 2.6 únicamente es válida si los estados del sistema son accesibles; en caso contrario, es necesario considerar la introducción de un observador. Además, no tiene en cuenta la posible influencia de fuentes de ruido del entorno del sistema, ni puede reaccionar ante perturbaciones, al no encontrarse dentro del modelo. Una revisión comprensiva del modelado y control en espacio de estados puede encontrarse en [42].

La predicción de la salida en un instante $t+k$ viene determinada por:

$$\hat{y}(t+k|\,t) = C\hat{x}(t+k|\,t) = C[A^k x(t) + \sum_{i=1}^{k} A^{i-1} Bu(t+k-i|\,t)] \tag{2.7}$$

Si bien es cierto que la mayoría de modelos están desarrollados con sistemas lineales en mente, a veces la aproximación lineal a un proceso de control no es suficiente, especialmente cuando las prestaciones del controlador a diseñar deben ser de alto rendimiento. Las representaciones basadas en ecuaciones diferenciales no lineales suelen ser complicadas de evaluar y resolver, lo que reduce el ancho de

banda del controlador y, por extensión, la posibilidad de aplicarlo a sistemas con constantes de tiempo pequeñas.

Una alternativa a esto es linealizar el conjunto de ecuaciones diferenciales en cada iteración, de forma que los coeficientes del modelo empleado sean dinámicos. Si el sistema permite una buena identificación, esta solución puede mejorar el desempeño del controlador, pero si los modelos son derivados de IRM o SRM, esta alternativa no es viable, por el alto número de parámetros que habría que determinar en cada iteración. En menor medida, esto también es cierto para el modelado SSM, si el número de estados es alto o estos no son accesibles. Otra solución -empleada de forma extensiva en el control de convertidores de electrónica de potencia, por ejemplo- puede ser la de asumir que las variaciones en torno a un punto de trabajo fundamental del sistema van a ser pequeñas, y trabajar con un modelo en pequeña señal [13].

Una de las mejores formas de adoptar un modelo no lineal con un rendimiento adecuado y una velocidad de ejecución lo suficientemente alta es el empleo de redes neuronales como modelos de caja negra [14].

La aproximación discutida será la de emplear una red neuronal multicapa basada en perceptrones, MLP, un tipo de arquitectura de red neuronal *feedforward* basada en el empleo de capas internas con funciones de activación no lineales. La aplicación de estas funciones de transferencia permite que la red aprenda relaciones no lineales entre los conjuntos de entrada y salida. Una explicación profunda del fucionamiento de esta arquitectura puede encontrarse en [31].

Dado que si la red se entrena de forma adecuada -es decir, con un cojunto de entrenamiento lo suficientemente grande y diverso- se consigue un comportamiento generalizador, la elección de la estructura del modelo se basa en seleccionar qué entradas se van a proporcionar, conocidas como *regresores*, y en determinar el número de capas ocultas de la red, así como el de neuronas dentro de las mismas.

La predicción del modelo, en un instante cualquiera t, viene dada por:

$$\hat{y}(t) = g(\phi(t), \theta) \tag{2.8}$$

donde $\phi(t)$ es el vector de regresores y θ es el vector de parámetros de la red neuronal (pesos de conexión entre neuronas y sesgos de las mismas).

Un esquema de la confección del modelo neuronal puede encontrarse en la figura 2.6. Se someten al sistema a identificar y al modelo a la misma señal de entrada $u(t)$, y las salidas, y e $\hat{y}$ se comparan a través de la función de entrenamiento, que suele adoptar la forma de un calculador de error cuadrático. La salida de esta función, E, da la información necesaria para actualizar el vector de parámetros θ de la red neuronal. Si bien suele ser común que el sistema real dependa de la entrada actual

y de las salidas o estados en instantes anteriores, esta cualidad no tiene por qué verse trasladada al modelo neuronal, ya que sus entradas vienen determinadas por la elección del vector de regresión $\phi(t)$.

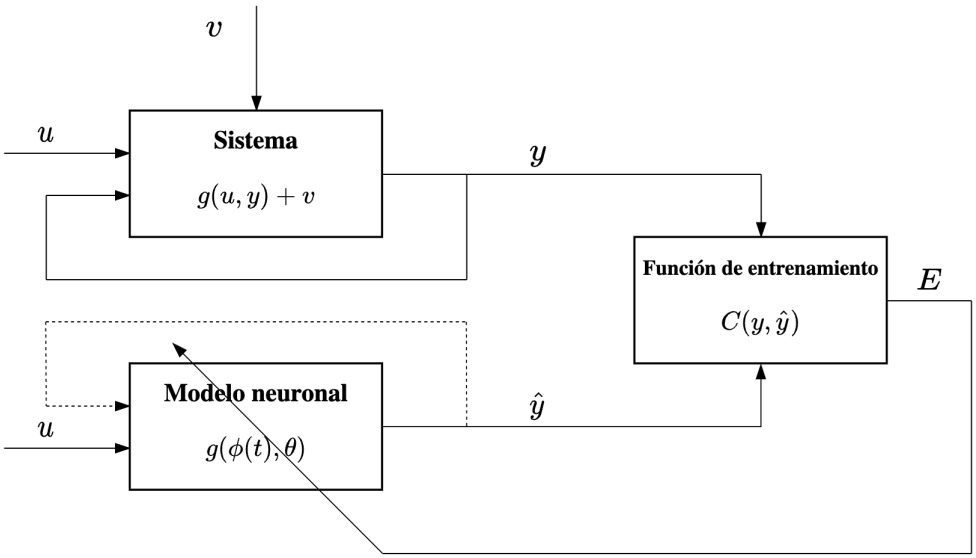

Figura 2.6: Entrenamiento del modelo neuronal.
Fuente: Adaptada de [14].

En función de dicha elección, se distingue entre los siguientes tipos de modelo:

- Neural network Nonlinear Finite Impulse Response (NNFIR). Emplea como vector de regresión únicamente las entradas al sistema: $\phi(t) = [u(t), u(t-1), \ldots, u(t-m)]$.

- Neural network Nonlinear AutoRegressive, eXternal input (NNARX). El vector de regresión se configura con las entradas al sistema, precedidas de las salidas anteriores del mismo: $\phi(t) = [y(t-1), y(t-2), \ldots, y(t-n), u(t), u(t-1), \ldots, u(t-m)]$. Esto ocasiona que la red neuronal tenga cierto sentido de corrección con respecto a la salida real del sistema, lo que permite que el modelo neuronal pueda reaccionar en cierta medida ante perturbaciones medidas en la salida.

- Neural Network Nonlinear AutoRegressive, Moving Average, eXternal input (NNARMAX). Una evolución del modelo anterior, introduce el error de predicción en la red neuronal además de las salidas anteriores del sistema y el histórico de la señal de entrada: $\phi(t) = [y(t-1), y(t-2), \ldots, y(t-n), u(t), u(t-1), \ldots, u(t-m), e(t-1), e(t-2), \ldots, e(t-q)]$. La inclusión del error de predicción, además de la salida del sistema original, le otorgan al modelo neuronal la capacidad de corregir las predicciones de forma más fiable que el modelo

NNARX. La principal desventaja con respecto a este es el aumento considerable de complejidad de la red neuronal, que a su vez conlleva una mayor dificultad a la hora de seleccionar los conjuntos de entrenamiento y validación. La figura 2.7 representa la arquitectura de un modelo NNARMAX.

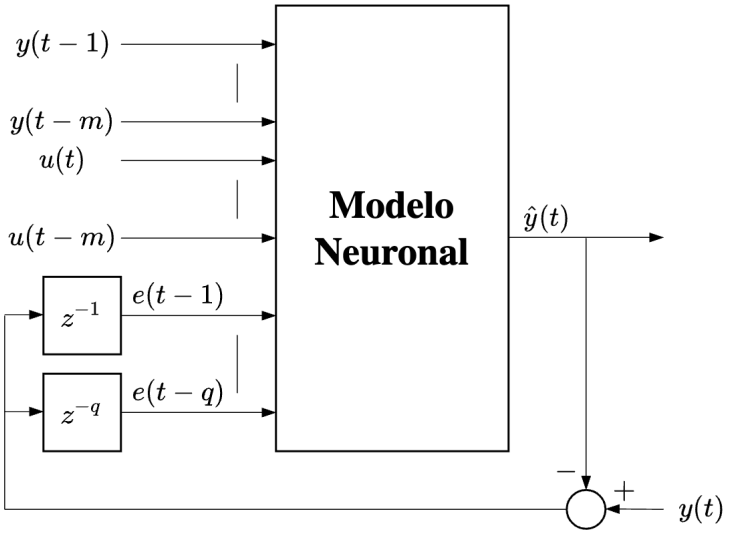

Figura 2.7: Esquema del modelo NNARMAX.
Fuente: Adaptada de [14].

Si bien es cierto que existen otros modelos que emplean otros vectores de regresión, como el Neural network Nonlinear Output Error (NNOE), los tres descritos son los que ofrecen un mejor rendimiento a la hora de emplearlos como predictor en un controlador MPC.

2.3.2. Función de coste

El objetivo de los controladores MPC es, en la mayoría de los casos, provocar que la salida y de un sistema de terminado siga una referencia r mediante una señal de control u. Como se ha explicado anteriormente, el mecanismo sobre el que se cimienta toda la acción de control es el paradigma del horizonte recesivo, por el que se predice la salida $\hat{y}$ del sistema gracias a un modelo del mismo, y se elige la mejor señal de control posible comparando los resultados de esa predicción con la referencia.

Esta comparación se realiza por medio de la conocida como *función de coste*, de *error* o de *pérdida*. Por simplicidad, a lo largo de este apartado será referida como función de coste. En la mayoría de aplicaciones -no únicamente referidas al control MPC, sino en cualquier disciplina en la que se quiera comparar una predicción con respecto a un valor real- se emplea una formulación cuadrática, de forma que pueda

asegurarse que la función sea cóncava y posea un mínimo global, que dará la menor diferencia entre la predicción y la secuencia real.

La función de coste toma la siguiente forma [4]:

$$J(N_1, N_2, N_p) = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(t+k|\,t) - w(t+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[u(t+j-1) - u(t+j-2)]^2 \tag{2.9}$$

El primer término de la ecuación 2.9 se ocupa de comparar la trayectoria predecida con la que se quiere seguir -ponderada con los coeficientes $\delta(j)$-, mientras que el segundo tiene en cuenta el esfuerzo de control entre iteraciones, ponderado mediante $\lambda(j)$. Hay algunas implementaciones en las que este segundo término no se tiene en cuenta, mientras que en otras lo que se optimiza es la propia señal de control, y no sus incrementos, con el objetivo de evitar problemas de saturación de las variables de control.

Los parámetros N_1 y N_2 determinan el horizonte de predicción, que de forma intuitiva se traduce en el intervalo de tiempo en el que se desea que el sistema siga la referencia $w(t)$. Una primera aproximación podría asegurar que el valor N_1 debería ser, en la mayoría de ocasiones, igual a 1, pero la posibilidad de manejar el horizonte de predicción conforme a las características particulares del sistema a controlar permite, por ejemplo, que en procesos con un tiempo muerto t_d se elija N_1 de forma que no se incluya dentro de la optimización un periodo de tiempo en el que el sistema físicamente no puede responder, lo que perjudicaría al proceso de optimización. Además, en procesos de fase no mínima, puede elegirse N_1 de forma que los instantes de respuesta inversa no influyan en la obtención de la ley de control. El parámetro N_u determina la longitud de la acción de control, es decir, hasta qué instante de tiempo se considera el control del sistema.

Las secuencias $\delta(j)$ y $\lambda(j)$ balancean la importancia de cada uno de los términos en el proceso de optimización, y controlan el comportamiento del sistema. Lo más común es emplear valores constantes para ellas, de forma que si δ es comparativamente grande sobre λ, se priorizará un seguimiento fiel de la trayectoria sobre la obtención de una ley de control suave y sin grandes saltos, mientras que al contrario se preservará en la medida de lo posible la energía empleada por los actuadores, a costa de un seguimiento de referencia con un error acumulado mayor.

Otra forma de configurar las secuencias de ponderación es mediante una función exponencial:

$$\delta(j) = \rho^{N_2 - j}$$

Si se elige un valor de ρ entre 0 y 1, los errores alejados del instante actual se ven más penalizados que los errores cercanos, lo que da lugar a acciones de control suaves que convergen hacia la trayectoria de referencia en instantes lejanos. Si, por el contrario,

se elige un valor de ρ superior a 1, los errores próximos sufren una gran penalización, ocasionando una respuesta rápida del sistema, a costa de proponer una secuencia de control irregular y que provoca un gran esfuerzo sobre los actuadores. La secuencia $\lambda(j)$ puede configurarse de forma similar, creando sistemas muy responsivos si ρ es mayor que 1, y sistemas más conservadores si ρ está entre 0 y 1.

La elección de las secuencias de ponderación es una de las formas más sencillas de ajuste del controlador MPC. Ya sea con valores constantes, funciones exponenciales, o secuencias intermitentes fabricadas a mano para procesos particularmente peliagudos, la elección de secuencias adecuadas juega una parte fundamental en el desempeño final del sistema de control, y su efecto intuitivo en la función de coste es uno de los puntos fuertes de esta aproximación de control.

La señal $w(t)$ representa la trayectoria a seguir por el sistema; esta señal no tiene por qué coincidir con la referencia de entrada. Dado que en la mayoría de procesos a los que se aplica la estrategia MPC se conoce la referencia con profundidad, puede manipularse el primer término de la función de coste, de forma que la transición en la que se alcanza la referencia desde una condición inicial arbitraria $y(t=0)$ del sistema se realice de forma óptima.

Una de las formas más comunes de afrontar este problema es la siguiente:

$$w(t) = y(t) \quad w(t+k) = \alpha w(t+k-1) + (1-\alpha) r(t+k) \quad k = 1, \ldots, N \tag{2.10}$$

donde α es un parámetro entre 0 y 1. Cuanto más cercano sea este parámetro a 1, más suave será la transición entre $y(t)$ y $r(t+k)$. Una representación de esto puede verse en la figura 2.8; la trayectoria w_1, con un valor de α pequeño, permite una respuesta muy rápida del sistema, mientras que w_2, con un valor alto de α, ocasiona una respuesta más suave.

Una vez caracterizada la función de coste, ha de considerarse el carácter real de los procesos de control, en la medida en que los actuadores no son ideales y, en general, los estados altos de energía suelen ser no deseables cuando se habla de sistemas de control. Es por ello que deben definirse restricciones en la función de coste, de forma que puedan evitarse las limitaciones de saturación de actuadores o plantas, ya sea en el valor absoluto de los esfuerzos o en el *slew rate* de los mismos. Las restricciones asociadas a un problema de optimización serán definidas con precisión en el próximo capítulo, conviene introducirlas como parte integral del controlador MPC. Suelen tomar la siguiente forma:

$$\begin{aligned} u_{min} \leq u(t) \leq u_{max} & \qquad \forall t \\ du_{min} \leq u(t) - u(t-1) \leq du_{max} & \qquad \forall t \\ y_{min} \leq y(t) \leq y_{max} & \qquad \forall t \end{aligned}$$

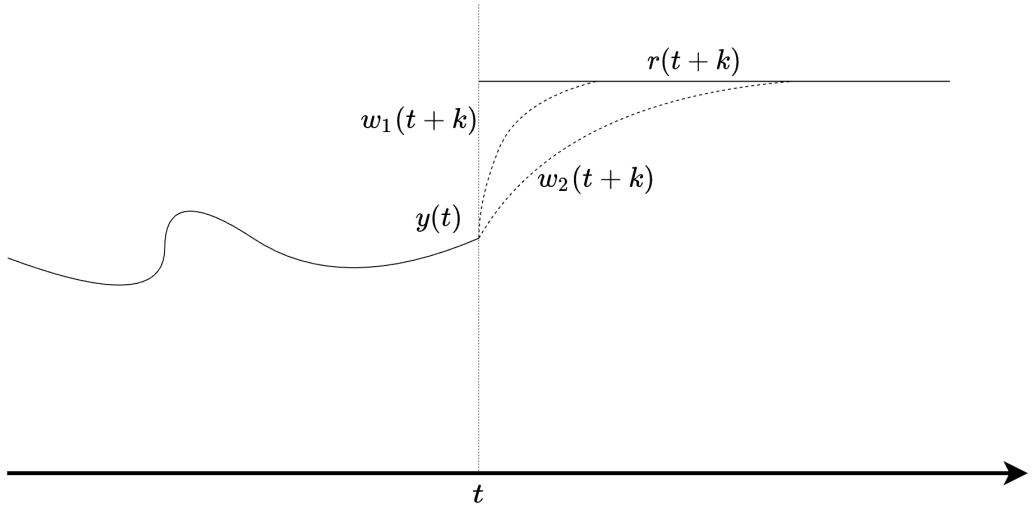

Figura 2.8: Seguimiento de la señal de referencia a través de la función de coste. Fuente: [4]

Las restricciones asociadas a la función de coste son una parte fundamental del proceso de optimización de la función de coste, ya que aseguran que el margen de operación del sistema se encuentra dentro de límites seguros o alcanzables. El problema de la saturación requiere de un análisis cuidadoso, pero dada la naturaleza fundamentalmente computacional de este arquetipo de control, no es necesario diseñar controladores *anti windup*, ya que este problema puede atacarse monitorizando la magnitud de las acciones de control a través del código y actuando en consecuencia si se detecta que esta es excesiva o demasiado pequeña.

2.3.3. Obtención de la ley de control

Como se ha venido comentando a lo largo de toda esta sección, para obtener la ley de control $u(t)$ es necesario minimizar la función de coste J, lo que a su vez conlleva el cálculo de las predicciones $\hat{y}(t+k|\,t)$ empleando el modelo del proceso a controlar.

Si el modelo es lineal, y el problema no tiene restricciones -es decir, la posible saturación de las variables de control no es importante, y no hay ningún aspecto externo a considerar más allá de la obtención de la solución óptima-, existe una solución analítica a la optimización del funcional J, y la obtención de la ley de control se limita a la implementación computacional de la ecuación en la que ésta resulta, de forma que la obtención de la ley de control es trivial. Por desgracia, este no suele ser el caso, lo que obliga al empleo de un optimizador iterativo.

Sea cual sea el método de minimización que decida emplearse, es de esperar un problema con $N_2 - N_1 + 1$ variables, lo que supone una carga computacional importante si se fuese a elegir un horizonte de predicción moderadamente elevado

-lo cual suele ser necesario en la mayoría de aplicaciones industriales-. Por ello, es necesario estructurar la ley de control de forma que pueda reducirse el número de variables independientes del problema, lo que aumentaría el ancho de banda del controlador al reducir el tiempo necesario para optimizar la función de coste.

Uno de los métodos más comunes es el de asumir que, más allá del horizonte de control N_u, permanece constante, esto es:

$$\Delta u(t+j-1) = 0 \qquad\qquad j > N_u$$

Cabe destacar que este método es uno de los más sencillos, y que existen otros como representar la señal de control como una combinación lineal de funciones previamente calculadas de forma que la salida del sistema se ajuste a la referencia [4].

Además de reducir la carga computacional del controlador, la estructuración de la ley de control conforme a un patrón predeterminado contribuye a mejorar el comportamiento del sistema, en la medida en que puede evitar que se desarrollen componentes de alta frecuencia en la señal de control en instantes lejanos al actual, mejorando en definitiva la robustez del sistema.

3. Metaheurística

El objetivo de este capítulo es el de presentar las estrategias metaheurísticas de optimización. En primer lugar se realizará una definición comprensiva de qué constituye la disciplina metaheurística. Tras ello se propondrán una serie de clasificaciones taxonómicas de la inmensa cantidad de algoritmos y variantes que se han desarrollado a lo largo de los años, y se describirá con detalle una de las estrategias metaheurísticas clásicas, así como una variación de la misma de reciente publicación.

3.1. ¿Qué es la metaheurística?

La gran mayoría de los problemas de toma de decisiones que pueden encontrarse en ámbitos científicos, económicos y sociales están basados en la comparación de dos o más alternativas y en alguna forma de evaluación de las mismas, de forma que pueda elegirse la que mejor satisfaga las condiciones y necesidades que se destilan del problema.

Ante un problema aparentemente sencillo, como pudiera parecer dimensionar una viga de madera para emplearla en la fabricación de un mueble artesanal, la experiencia de un carpintero que ha fabricado decenas de artículos con anterioridad es herramienta suficiente para obtener un resultado lo suficientemente aceptable. Esta aplicación de conocimiento previo para alcanzar la mejor decisión posible es lo que se conoce como *heurística*, la aplicación del aprendizaje de experiencias previas en situaciones similares para generar un comportamiento como respuesta a un problema para el que no se dispone de una solución. Si únicamente debemos fabricar unas pocas unidades, no es crítico que la solución adoptada no sea la que emplea el mínimo absoluto de material. De hecho, pudiera ser que en la fabricación del objeto se tengan en cuenta características cualitativas, como la disponibilidad del material y la forma de origen del mismo.

Supóngase ahora que se quiere producir un modelo de mueble en cantidades industriales para su comercialización. Para ello, disponemos de una función que permite relacionar cuantas variables quieran considerarse (tipo, dureza y densidad de la madera, carga sobre el mueble, humedad y temperatura de la estancia en la que va a situarse) con las dimensiones de la viga necesaria. Si se quiere escoger la decisión óptima -es decir, la que emplea la menor cantidad de madera para satisfacer las condiciones del problema- es necesario encontrar el mínimo de la función que relaciona las variables de entrada con la de salida, de forma que se emplee la menor cantidad de madera posible en cada unidad producida.

La primera aproximación de cualquier diseñador lo suficientemente ingenuo para plantearla sería diferenciar el modelo matemático explícito y resolver el problema de forma analítica; se dispone de una serie de funciones reales de variable

real, a priori continuas, y su tratamiento, por más que tedioso, no debería ser complicado. En un mundo ideal, este método arrojaría el mejor resultado: siempre la mejor solución obtenida de forma instantánea. Por desgracia, la grandísima mayoría de problemas que deben ser resueltos en el mundo moderno ni siquiera disponen de una formulación matemática explícita, ni mucho menos de una variedad fácilmente diferenciable y optimizable.

La siguiente idea lógica sería la de emplear un método iterativo para obtener las dimensiones críticas de la viga. La idea detrás de estos métodos es simple: se evalúa el desempeño de una solución (es decir, si cumple o no con las condiciones de carga) inicial -generalmente aleatoria-; si la solución propuesta cumple un requisito de terminación (que en este caso, por ejemplo, podría ser soportar la carga máxima prevista más una cantidad arbitraria de tolerancia), la solución óptima ha sido encontrada. Si, por el contrario, la solución no satisface el requisito de terminación, se manipula de forma determinista para obtener una nueva propuesta, que vuelve a evaluarse contra el criterio. La manipulación de la solución en una iteración para obtener la propuesta de la siguiente puede o no incluir evaluaciones de derivadas primeras. Entre los más empleados están los métodos de interpolación (sin cómputo de derivadas), el descenso de gradiente (primera derivada) o el método de Newton (hessiano).

En general, el mayor inconveniente de estos algoritmos deterministas está en el coste computacional asociado; el cálculo de la primera derivada y sucesivas es una tarea computacionalmente prohibitiva cuando la dimensión del problema crece, lo que puede ocasionar que deban emplearse soluciones no óptimas por haber alcanzado el máximo número de iteraciones que pueden permitirse. En el problema que viene sirviendo de ejemplo esta característica podría ser asumible en el sentido en el que el tiempo de ejecución no es crítico ya que el problema debe resolverse sólo una vez, pero en escenarios en los que la optimización debe realizarse *online*, puede suponer un inconveniente insuperable. Además, no hay forma de incorporar el conocimiento preliminar en el desarrollo de la optimización más allá de elegir cuidadosamente la solución inicial.

Supóngase, ahora, que se realiza una investigación extensiva del estado del arte de los algoritmos de optimización disponibles y sus implementaciones computacionales, y que ninguno de ellos parece cumplir los requisitos de exactitud, convergencia o eficiencia computacional requeridos. La respuesta a este problema es la metaheurística.

El término *metaheurística* hace referencia a una disciplina de la ciencia de la computación que recoge todos los algoritmos estocásticos con componentes aleatorias cuya idea originaria supone la implementación de un mecanismo o comportamien-

to determinado de forma que sea fácilmente adaptable al problema que se quiere resolver.

La mayoría algoritmos metaheurísticos hacen uso de dos mecanismos determinados para encontrar una solución a un problema definido sobre un espacio de búsqueda: exploración y explotación. La exploración es el proceso por el cual se muestrea de forma general un espacio, con el objetivo de identificar qué regiones del mismo pueden albergar la solución buscada. Por otro lado, la explotación es la búsqueda concienzuda en esas regiones identificadas, de forma que se obtenga como solución el punto óptimo que se ubica dentro de la misma.

La diferencia entre algoritmos radica en la inspiración que emplean para implementar los comportamientos definidos en el párrafo anterior. Uno de los algoritmos más peculiares es el Whale Optimization Algorithm (WOA), que toma su inspiración en el comportamiento de las ballenas jorobadas cuando cazan [34]. Este algoritmo diferencia con claridad los mecanismos de exploración y explotación, asignando un patrón comportamental a cada uno de ellos y haciendo implementaciones independientes reguladas por un parámetro que toma un valor aleatorio en cada iteración. Otro algoritmo que incorpora ambos mecanismos de forma diferenciada es Particle Swarm Optimization (PSO), en el que se pondera la importancia de los dos fenómenos mediante dos hiperparámetros del algoritmo en la ecuación fundamental del mismo. Asimismo, existen otros algoritmos que incorporan la búsqueda local y global en un único procedimiento combinado, aunque la dificultad de parametrizar el comportamiento de los mismos hace que se encuentren en desuso frente a estrategias más estructuradas.

Otro concepto importante dentro de la disciplina metaheurística es el llamado No Free-Lunch Theorem [51]. Este teorema, demostrado por Wolpert y Macready en 1997, asegura que, si dado un conjunto de problemas F_1, un algoritmo de búsqueda A tiene un mejor desempeño medio que otro algoritmo B -es decir, encuentra mejores soluciones-, existe otro conjunto de problemas F_2 diferentes para los que B muestra un desempeño medio mejor que A.

La interpretación de este teorema es inmediata: no existe ningún algoritmo que sea superior a todos los demás en absolutamente todos los problemas de búsqueda de solución óptima existentes. Esta conclusión es la principal razón tras la extensiva biblioteca de algoritmos que se han ido revelando en la literatura desde mediados de los años 90 hasta hoy. Una implicación peculiar del NFLT es que existe un conjunto de problemas F_r para los cuales el método de búsqueda aleatoria a través de fuerza bruta es el mejor para encontrar una solución óptima.

En definitiva, la metaheurística es una herramienta más dentro de la optimización que permite atacar problemas complejos o de grandes dimensiones. Las ventajas con respecto a los optimizadores tradicionales son varias: en primer lugar,

suelen tener un coste computacional asociado bastante menor que las alternativas deterministas, en la medida en que no requieren del cálculo de derivadas. Por otro lado, el empleo de técnicas estocásticas favorece una mejor exploración del espacio de búsqueda que en las alternativas tradicionales basadas en manipulaciones deterministas. Por último, los algoritmos metaheurísticos suelen trabajar mejor ante problemas con un gran número de variables independientes, convergiendo en un menor número de iteraciones y encontrando soluciones más refinadas que otras alternativas de optimización.

3.2. Optimización

3.2.1. Conceptos básicos

El objetivo de esta sección es el de definir una serie de conceptos y términos necesarios para caracterizar los problemas de optimización. Muchos de estos conceptos han sido adaptados de las definiciones propuestas por Bozorg-Haddad, Solgi y Loáiciga en [45].

Se conoce como *optimización* al proceso de búsqueda organizada de una especificación de diseño de un sistema que le permita operar bajo una serie de limitaciones o restricciones. En un contexto matemático, el proceso de optimización busca el máximo o mínimo de una función objetivo definida sobre un conjunto de variables y cuyo conjunto de destino está ordenado.

Más formalmente, una definición del problema puede ser la siguiente:

$$\text{Optimizar } f(X) \text{ con } X = (x_1, x_2, \ldots, x_i, \ldots, x_N)$$

Restringida a:

$$\begin{aligned} g_j(X) &< b_j, && \forall\, j = 1, \ldots, M \\ x_i^L &< x_i < x_i^U, && \forall\, i = 1, \ldots, N \end{aligned}$$

Donde $f(X)$ es la *función objetivo*, $X = (x_1, x_2, \ldots, x_i, \ldots, x_N)$ es el vector de *variables de decisión* de dimensión N, y representa una posible solución al problema de optimización, $g_j(X) < b_j$ es la j-ésima de las m *funciones de restricción*, y x_i^L y x_i^U definen las *fronteras* inferior y superior de cada una de las variables de decisión.

Las soluciones al problema de optimización se agrupan en el *espacio de decisión*, un espacio N-dimensional delimitado por las restricciones y las fronteras en el que cada posible solución es un punto X. El problema consiste, pues, en buscar el X^* que produce el valor mínimo de $f(X)$ cumpliendo con las restricciones $g_j(X)$. En general, en el mundo ingenieril, los problemas de optimización suelen estar re-

lacionados con la minimización de un conjunto de variables. En cualquier caso, y sin pérdida de generalidad, puede asumirse que optimizar consiste en minimizar un conjunto de una o varias funciones objetivo, ya que cualquier problema de maximización puede transformarse en su contrario multiplicando las funciones objetivo por -1.

Se dice que un punto es *mínimo local* si existe un $\varepsilon > 0$ tal que:

$$f(X^*) \leq f(X), \quad X^* - \varepsilon \leq X \leq X^* + \varepsilon$$

y se conoce como *mínimo global* al menor de los mínimos locales dentro del espacio de búsqueda. Los mínimos locales, por tanto, están limitados a una vecindad arbitraria de X^*, mientras que los mínimos globales arrojan el mejor de los resultados de la(s) función(es) objetivo. Cabe destacar que la cantidad de mínimos globales asociados a una función objetivo y un espacio de búsqueda puede ser ilimitada, y que su existencia está garantizada por medio del teorema generalizado de Weierstrass.

Es común que encontrar de forma exacta el mínimo global de una función objetivo sea una tarea computacional prohibitiva, y que la respuesta al problema deba ser una solución relativamente cercana a la óptima, pero no la mejor. Para ello, puede definirse el siguiente criterio de terminación: X^λ es una solución *cuasi-óptima* si

$$|f(X') - f(X^*)| < \lambda, \quad \lambda > 0,$$

donde el parámetro λ recibe el nombre de *factor de tolerancia*.

Un ejemplo de problema de optimización podría ser el siguiente:

$$Minimizar\, y(x) = x^2 - 1$$

sujeta a

$$\begin{gathered} x^2 + y^2 < 2, \\ y \geq x, \\ -\frac{3}{2} < y < \frac{3}{2} \end{gathered}$$

Este ejemplo es deliberadamente sencillo con el objetivo de posibilitar la representación gráfica del mismo, la cual puede observarse en la figura 3.1. El punto mínimo que cumple las restricciones propuestas es $X^* = (-0{,}732, -0{,}732)$.

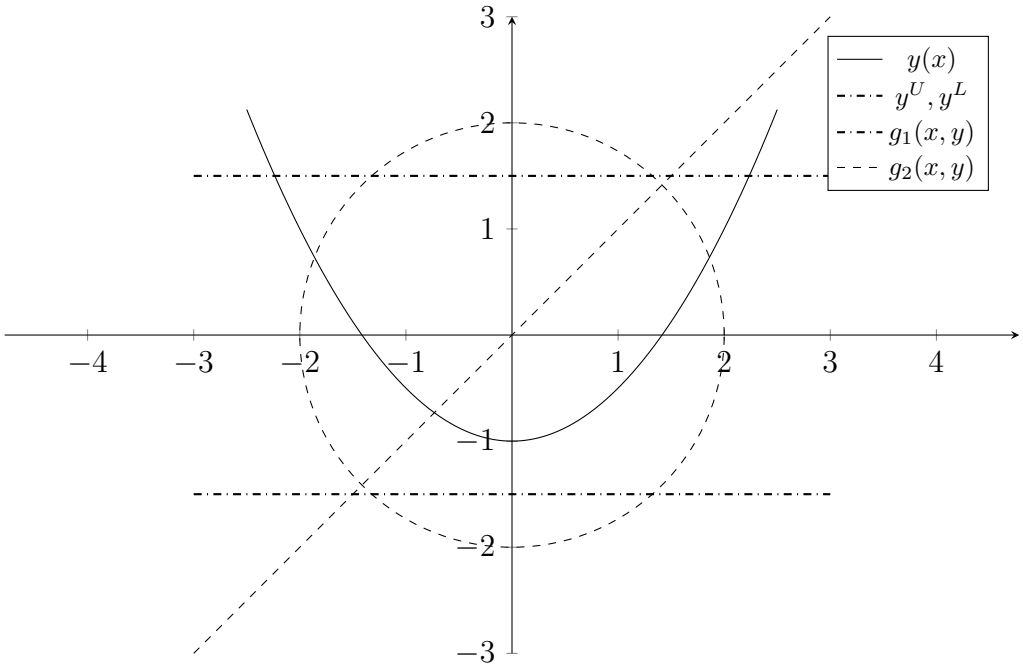

Figura 3.1: Representación gráfica de problema ejemplo.

3.2.2. Tipos de problemas de optimización

Una vez sentados los conceptos básicos de los problemas de optimización, se ofrece una breve introducción a los diferentes tipos de problemas de optimización y su formulación matemática.

En función de las características del problema, pueden hacerse las siguientes distinciones [8][9][19]:

- Problemas continuos o discretos. Los problemas continuos son aquellos que toman valores en conjuntos densos, en general $\mathbb{R}^n$. En cambio, los problemas discretos están definidos sobre conjuntos contables. Los problemas continuos presentan una dificultad sensiblemente inferior a la hora de ser resueltos, ya que la continuidad de las funciones en un punto determinado X^* permite obtener información acerca de un entorno cercano del mismo.

- Problemas con o sin restricciones. Los problemas sin restricciones son aquellos que están definidos y admiten soluciones en todo el espacio de búsqueda definido. Los problemas con restricciones, en cambio, deben satisfacer criterios adicionales además de presentar una solución óptima. Este tipo de restricciones suelen ser formuladas como inecuaciones, y reflejan límites explícitos en las variables independientes del problema. Los problemas con restricciones suelen ser más complicados de resolver, ya que requieren la evaluación de las mismas para determinar la validez de la solución. Las restricciones pueden eliminarse

sustituyéndolas por un término de penalización en la función de coste que se utilice para evaluar la idoneidad de la solución propuesta por el algoritmo.

- Problemas con ninguna, una o varias funciones objetivo. Si bien es cierto que la mayoría de problemas de optimización constan de al menos una función objetivo, existe una variedad de problemas denominada de *viabilidad*, en el que se pretende encontrar un conjunto de variables que satisfagan una serie de restricciones, sin una función objetivo en particular que minimizar. Los problemas de optimización con una única función objetivo son los más comunes, mientras que los que evalúan varias funciones suelen buscar relacionar características diferentes (por ejemplo, riesgo y beneficio en la selección de una estrategia mercantil), minimizando algunas y maximizando otras. Es común que los problemas con varias funciones se reduzcan a una única función mediante una combinación lineal o transformando alguna de las funciones objetivo en restricciones para las demás.

- Problemas deterministas o estocásticos. En relación al carácter de los parámetros que definen el problema, se distingue entre optimización determinista, en la que los datos que se conocen del problema son conocidos con precisión, o estocástica, en la que dichos parámetros están definidos con una incertidumbre determinada, que puede deberse a errores de medida -no confundir con las estrategias estocásticas de optimización, que emplean métodos aleatorios en su ejecución para manipular las soluciones tentativas-.

- Problemas convexos o no convexos. Los problemas de optimización convexos son aquellos en los que, dado un espacio de búsqueda y un conjunto de restricciones, puede obtenerse una única solución, que hace de óptimo global, o probar que ninguna de las soluciones presentes en el espacio de búsqueda es viable, es decir, ninguna cumple los requisitos impuestos por las restricciones. En cambio, en los problemas no convexos, el espacio de búsqueda contiene óptimos locales, provocados por la topología de la función objetivo. La existencia de problemas no convexos supone un gran quebradero de cabeza en la disciplina de la optimización, ya que exige una exploración concienzuda del espacio de búsqueda para evitar convergencias en soluciones no óptimas.

A pesar de que se han expuesto varios criterios para clasificar los problemas de optimización, el principal empleado para decidir qué técnica de resolución se emplea es la presencia de restricciones. Dado que en el mundo ingenieril todos los problemas definidos las poseen, la introducción de los diferentes problemas a resolver se basarán en aquellos que las incluyen.

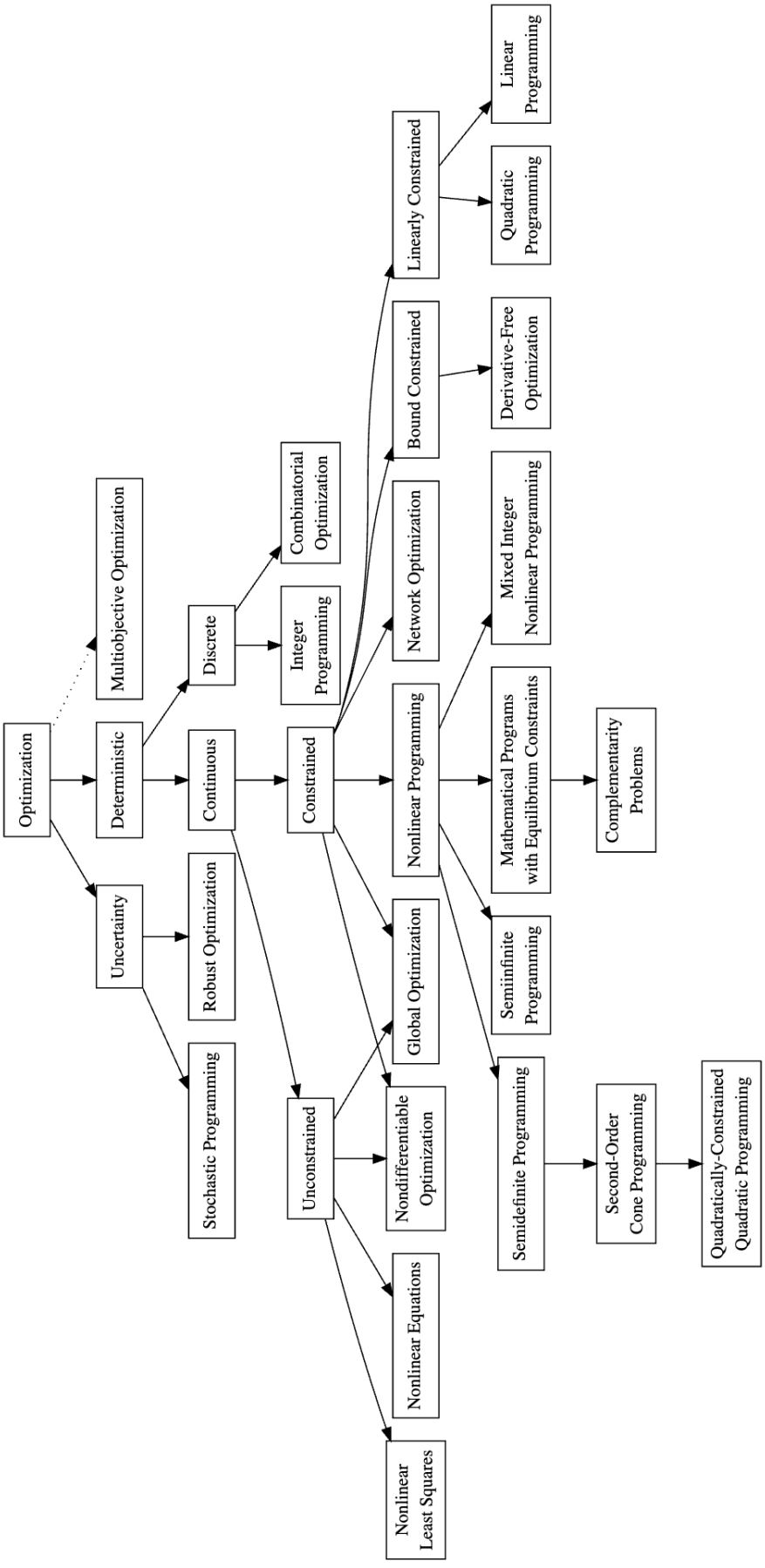

Figura 3.2: Clasificación taxonómica de los subcampos de la optimización
Fuente: [8] [9] [19]

- Bound Constrained Optimization (BCO). Este término agrupa todos los problemas en los que se busca minimizar una función objetivo sujeta a condiciones del tipo frontera, de la forma:

$$\text{Minimizar } f(X) \text{ con } X = (x_1, x_2, \ldots, x_i, \ldots, x_N)$$

Restringida a:

$$x_i^L < x_i < x_i^U, \quad \forall\, i = 1, \ldots, N$$

Juegan un papel importante dentro de la implementación de software y la generación de algoritmos de complejidad superior, ya que muchos de ellos reducen la solución del problema de origen a la resolución de una serie de problemas transformados a partir de las restricciones y función objetivo inicial. Entre los algoritmos más conocidos que aplican esta aproximación destacan el método de Newton y el Gradient Projection (GP).

- Linear Programming (LP). Esta familia de problemas tiene como objetivo minimizar funciones objetivo lineales sujetas a restricciones del mismo tipo. La forma estándar del problema a resolver es la siguiente:

$$\text{Minimizar } C^T X \text{ con } X = (x_1, x_2, \ldots, x_i, \ldots, x_N)^T$$

Restringida a:

$$\begin{aligned} AX &= B \\ x_i &\geq 0, \quad \forall\, i = 1, \ldots, N \end{aligned}$$

donde C es el vector de ponderación de coste y A y B son la matriz y el vector independiente de restricciones, respectivamente. Suele ser común que A tenga menos filas que columnas -es decir, el problema tiene menos restricciones que variables independientes-. Aunque todos los problemas pueden ser convertidos a la forma canónica, muchas implementaciones del LP permiten la inclusión de fronteras, restricciones con valores inferior y superior, e incluso pueden resolver problemas de maximización multiplicando C por -1. Las técnicas de solución más empleadas son Simplex Method (SM) y Interior-Point Method (IPM). Ambas representan una familia de desarrollos algorítmicos con implementaciones específicas.

- Sequential Linear Programming (SLP). Concebida a medio camino entre la programación lineal y no lineal, se basa en la resolución de problemas con

no linealidades -ya sea en la función objetivo o en las restricciones- mediante un proceso de linealización que hace uso del desarrollo de Taylor de orden uno. En general, se empieza desde una estimación de la solución óptima y se van realizando iteraciones sucesivas del proceso hasta que se cumple un criterio de terminación. La aproximación de linealización requiere de cuidado en el tratamiento de las soluciones, ya que pueden excederse las fronteras del problema original o no cumplir de forma estricta las restricciones no lineales, lo que hace necesario el empleo de regiones de confianza para asegurar la convergencia. Un optimizador usualmente empleado para resolver problemas de este tipo es Constrained Optimization by Linear Aproximation (COBYLA).

- Quadratic Programming (QP). Este problema consiste en minimizar funciones cuadráticas sujetas a restricciones lineales, y su formulación se ajusta a:

$$\text{Minimizar } \frac{1}{2}X^T QX + C^T X \text{ con } X = (x_1, x_2, \ldots, x_i, \ldots, x_N)$$

Restringida a:

$$AX \leq B$$

donde $Q \in R^{n \times n}$ es una matriz simétrica, C es el vector de ponderación de los términos de primer orden, A es la matriz de restricciones, y B el vector de restricciones independiente.

La dificultad de resolución de este problema viene determinada por la matriz Q. Si es semidefinida positiva en el espacio viable determinado por las fronteras y las restricciones, el problema es convexo y hay una única solución óptima. Por el contrario, si Q tiene algún autovalor negativo, el problema es no convexo, lo que obliga a la consideración de óptimos locales y posibles problemas de convergencia sobre ellos.

El problema cuadrático es sumamente importante, ya que resuelve por si solo el problema del control óptimo, y además sirve como base para resolver problemas más complejos mediante el Sequential Quadratic Programming (SQP), haciendo uso de una idea similar a la del SLP pero con desarrollos de Taylor de orden 2.

- Nonlinear Programming (NP). La programación no lineal afronta problemas como el definido al comienzo de esta sección, en los que tanto la función objetivo como las restricciones son no lineales. Puede afirmarse que todos los problemas descritos en esta sección constituyen un caso particular de NP, imponiendo características específicas de linealidad o forma cuadrática sobre la función objetivo o las restricciones.

Aunque aquí se ha hecho una introducción breve de algunos de los tipos más importantes de problemas dentro de la optimización, existen otras muchas ramas, como se muestra en la figura 3.2. Las referencias relacionadas con NEOS Server ([8], [9] y [19]) ofrecen una buena recapitulación de tipos de problemas de optimización y soluciones propuestas a los mismos. NEOS Server es una iniciativa de la Universidad de Wisconsin cuyo objetivo es ofrecer una plataforma *online* y gratuita en la que se puede acceder a más de 60 optimizadores punteros.

3.3. Clasificación de los algoritmos metaheurísticos

3.3.1. Fuente de inspiración

La clasificación que se propone en función del mecanismo subyacente de inspiración es la siguiente [2]:

- Algoritmos inspirados en la biología.
  - Algoritmos genéticos. Los algoritmos de inspiración genética suponen uno de los primeros estudios formales dentro del campo de la metaheurística. Introducidos por Holland en 1962 (y analizados en profundidad en [21], de 1984), circulan en torno a la idea de evolución propuesta por Darwin en el siglo XIX. Están basados en un conjunto de soluciones iniciales -agentes- que constituyen una población. Estos se desplazan por el espacio de búsqueda mediante el empleo de operadores genéticos, como el apareo (dos agentes combinan información de cada una de sus d-dimensiones para formar un tercero) o mutación (un agente sufre una o varias modificaciones aleatorias en su posición). En cada iteración (más conocida en el entorno de estos algoritmos como generación), un porcentaje de la población desaparece por no haberse adaptado lo suficiente, es decir, por no mejorar su posición en términos de la búsqueda de la solución óptima.
  - Algoritmos basados en en la inteligencia de enjambre. Estos algoritmos están basados en los principios de descentralización e intercambio de información observados en bandadas de pájaros o enjambres de insectos. Muchas especies emplean mecanismos de comunicación entre agentes del enjambre para alcanzar objetivos como zonas ricas en alimento o para recorrer un camino de vuelta seguro al lugar de congregación. La implementación de de estos comportamientos en algoritmos metaheurísticos ha dado lugar a dos de los optimizadores más estudiados y empleados en la disciplina, Particle Swarm Optimization (PSO) y Ant Colony Optimization (ACO).

    - Algoritmos basados en el sistema inmune. Estos optimizadores toman su inspiración de mecanismos observados en la inmunología, implementando el concepto de anticuerpo como solución candidata y el de antígeno como función objetivo. Los anticuerpos evolucionan durante la ejecución del algoritmo mediante clonación, mutación y selección, y sus mejores posiciones históricas se almacenan en una célula de memoria. Una de las metaheurísticas más conocidas dentro de este grupo es el Clonal Selection Algorithm (CLONALG) [48].

- Algoritmos inspirados en la química. Estos algoritmos incorporan ideas relacionadas con los principios de intercambio de energía en las reacciones químicas, así como las constantes de equilibrio de las mismas o los potenciales químicos que intervienen en ellas. Otra posibilidad es incorporar los patrones de movimiento browniano de partículas fluidas que se ven sometidas a un régimen turbulento. En la mayoría de estos algoritmos se emplea el concepto de temperatura para regular la velocidad de los agentes, con un efecto similar al que se discutirá más adelante del parámetro de inercia en algoritmos de inteligencia de enjambre. En este grupo de algoritmos destacan Chemical Reaction Optimization (CRO) [28] y Gases Brownian Motion Optimization (GBMO) [1].

- Algoritmos inspirados en la música. Los dos algoritmos más famosos que se incluyen en esta categoría son Harmony Search (HS) y Method of Music Composition (MMC) [35]. En HS se emplea el concepto de improvisación musical para generar conjuntos de instrumentos como soluciones cuyo criterio de evaluación se basa en la armonía producida por la combinación de los mismos. Cuanto mejor es la armonía generada por un conjunto de instrumentos, más posibilidades hay de que esa solución sea incluida en el banco de memoria del algoritmo. Es un algoritmo complejo en su inspiración y en la implementación de la misma. La lectura de [16] ofrece una explicación mucho más profunda que la introducida aquí. Por otro lado, MMC asocia a cada solución un una composición formada por las d dimensiones del problema. Cada una de estas composiciones intercambia información con otras en base a unos criterios determinados, lo que provoca un comportamiento similar a aquel de los operadores genéticos.

- Algoritmos inspirados en las matemáticas. No hay una característica común en este grupo, más allá de tomar inspiración de diversos conceptos matemáticos de complejidad variable. Dentro de estos algoritmos podemos encontrar el Base Optimization Algorithm (BOA) [44], un algoritmo basado en manipulaciones aritméticas (suma, resta, multiplicación y división) de una población inicial de

soluciones aleatorias. Cada una de ellas pasa por el proceso aritmético, arrojando cuatro tentativas nuevas conforme a un parámetro de desplazamiento, las cuales son evaluadas conforme al criterio de validez. Sólo la mejor de las soluciones procesadas sobrevive el proceso, que se repite iteración tras iteración hasta que se cumple un criterio de terminación. Otro optimizador destacado es el Sine Cosine Algorithm (SCA) [33], el cual manipula una población inicial aleatoria mediante las funciones seno y coseno añadiendo cuatro parámetros aleatorios extraídos de una distribución $U(0, 1)$.

- Algoritmos basados en la física. Al igual que en el apartado anterior, no hay un concepto común que permee todos los algoritmos que pueden agruparse bajo este epígrafe. Una de las metaheurísticas más importantes es la Simulated Annealing (SA) [26], un algoritmo de trayectoria que recrea el movimiento molecular de las partículas de un material metálico cuando es sometido a un enfriamiento rápido. Otro algoritmo destacado dentro de esta categoría es el Gravitation Search Algorithm (GSA) [40], en el que cada agente de la población inicial es considerado un objeto con posición, masa inercial, y masa gravitacional tanto activa como pasiva. La posición representa la solución candidata asociada al agente, y los otros tres conceptos intervienen en la evaluación de la validez de dicha solución: cuanto mayor sea esta, más poder gravitacional se confiere al objeto, lo que atrae a los demás hacia él, implementando un concepto similar al intercambio social de información visto en PSO.

- Algoritmos basados en comportamientos sociales. Este grupo hace uso de una de las fuentes de inspiración más curiosas en la disciplina metaheurística. Probablemente el algoritmo más conocido relacionado con el comportamiento social sea el League Championship Algorithm (LCA) [23]. Este algoritmo asocia cada solución a un equipo deportivo, los cuales compiten por pares en partidos. Aplicando unos criterios específicos, se decide cuál de los dos equipos es más fuerte, es decir, cuál de las dos soluciones es más óptima, y el equipo débil reformula su plantilla, cambiando su posición dentro del espacio de búsqueda.

- Algoritmos sin fuente de inspiración. A pesar de que la mayoría de algoritmos metaheurísticos tienen una idea subyacente generalmente extraída del entorno natural o del comportamiento humano, existen algoritmos que han sido pensados explícitamente como metaheurísticos con el único objetivo de su implementación, sin un concepto inspirador detrás. Algunos de ellos son Tabu Search (TS) [18], Variable Neighbourhood Search (VNS) [20] o Partial Optimization Metaheuristic Under Special Intensification Conditions (POPMUSIC) [46].

3.3.2. Clasificación funcional

Blum y Roli proponen en [3] un sistema de clasificación basado en las características de operación del algoritmo. Esta taxonomía produce una clasificación tabular que permite distinguir, en base al principio de funcionamiento, si un algoritmo puede ser beneficioso para resolver un problema particular o no, siempre que se tenga un conocimiento adecuado sobre el mismo y sobre el espacio de búsqueda. Cabe destacar que cada una de estas características podría establecerse como patrón de clasificación único desde un punto de vista formal.

- Según número de soluciones empleado en la búsqueda de la solución óptima, se distingue entre algoritmos de *población* o de *agente único*. Los algoritmos de población emplean un conjunto de puntos definidos sobre el espacio de búsqueda, y los manipulan conforme a unos criterios determinados para encontrar la solución. Por su parte, los algoritmos de agente único describen una trayectoria por el espacio de soluciones. Una ventaja que pueden aportar los algoritmos de trayectoria con respecto a los de población es la posibilidad de definir con facilidad un criterio de terminación relativo a la variación de posición de la solución, en vez de esperar a ejecutar el total de las iteraciones asignadas. Si bien es cierto que esto también es posible en los algoritmos de población, es considerablemente más complejo, ya que no es posible determinar la influencia que tendrá la posición actual de los agentes sobre ejecuciones futuras del bucle de optimización. Por su parte, los algoritmos de población aseguran una mejor exploración del espacio de búsqueda, lo que puede llevar a soluciones de mejor calidad.

- En función del tratamiento que hace el algoritmo de la función objetivo a optimizar, se habla de algoritmos con función objetivo *estática* o *dinámica*. Uno de los posibles beneficio de alterar la función sobre la que se busca el óptimo es el de escapar de posibles convergencias tempranas en mínimos locales mediante la modificación de la forma del espacio de búsqueda, lo que además posibilita la inclusión de información que se haya podido obtener durante la ejecución del algoritmo en el proceso.

- Con respecto a la topología del espacio de búsqueda, se distingue entre algoritmos de *vecindad única* o *variable*. En los primeros, los agentes son libres para deambular por todo el espacio de búsqueda, siendo únicamente limitados por las restricciones del problema y las fronteras del espacio de búsqueda. Esta topología permanece, además, constante durante la ejecución del optimizador. Por otro lado, los algoritmos de vecindad variable proponen una subdivisión de la población en conjuntos más pequeños confinados en una región del espacio

Algoritmo	Nº de soluciones	Función objetivo	Vecindades	Uso de memoria
Particle Swarm Optimization (PSO) [24]	Población	Estática	Única	Sí
Tabu Search (TS) [18]	Trayectoria	Estática	Única	Sí
Simulated Annealing (SA) [26]	Trayectoria	Estática	Única	No
Variable Neighbourhood Search (VNS) [20]	Trayectoria	Estática	Múltiples	Sí
Iterated Local Search (ILS) [30]	Trayectoria	Estática	Única	No
Guided Local Search (GLS) [49]	Trayectoria	Dinámica	Única	Sí
Genetic Algorithm (GA) [21]	Población	Estática	Única	No
Dynamic Tabu Search (DTS) [43]	Trayectoria	Dinámica	Única	No
Ant Colony Optimization (ACO) [6]	Población	Estática	Única	Sí

Tabla 3.1: Clasificación funcional de algunos algoritmos metaheurísticos.

de búsqueda conforme a unos criterios determinados. La definición dinámica de estos criterios permite la modificación topológica de la región de confinamiento, lo que aporta beneficios en la diversificación de la búsqueda.

- En función de si el optimizador emplea información obtenida durante su ejecución, se distingue entre algoritmos *con memoria* o sin ella. Los algoritmos con memoria emplean información del histórico de búsqueda, como puede ser la mejor solución alcanzada o regiones en las que se ha identificado un mínimo local. El empleo de memoria, entre otros beneficios, asegura que el algoritmo siempre devuelve la mejor solución que ha conocido, en vez de perder una posible solución óptima por la influencia de aspectos que provoquen un refinamiento poco adecuado de la solución. Por el contrario, los algoritmos sin memoria emplean un proceso markoviano en el que únicamente se utiliza la información del estado actual del proceso de búsqueda.

La tabla 3.1 recoge una clasificación funcional de algoritmos metaheurísticos representativos. Es importante destacar que esta clasificación no es cerrada, en la medida en que la mayoría de algoritmos son modificables en un sentido u otro para incorporar mecanismos presentes en otros. Por ejemplo, existen versiones de PSO, como Variable Neighbourhood Particle Swarm Optimization (VNPSO) [29], que incorporan el concepto de vecindad variable con el objetivo de mejorar el rendimiento del algoritmo original.

3.4. Particle Swarm Optimization (PSO)

En este apartado se presenta uno de los algoritmos clásicos dentro de la metaheurística, estudiado desde mediados de los noventa hasta la época actual. Para ello, se hará una revisión del algoritmo propuesto originalmente con una serie de modificaciones necesarias para garantizar su funcionamiento, seguida de una modificación moderna que elimina parte del proceso de ajuste del algoritmo.

3.4.1. Algoritmo canónico

La Optimización por Enjambre de Partículas (PSO, del inglés Particle Swarm Optimization) es un método de optimización de funciones no lineales propuesto por Kennedy y Eberhart en 1995 [24]. En un principio, este algoritmo surge a partir de una simulación de comportamientos sociales detectados en bandadas de pájaros y bancos de peces. Tras una serie de experimentos en los que se demostró que la aplicación directa de los conceptos biológicos que rigen esos comportamientos llevaba a problemas de congregación de los agentes de la población, lo que impide una exploración efectiva del espacio de búsqueda, se decidió introducir una serie de elementos aleatorios, de forma que el comportamiento del algoritmo se asemeje más al de un enjambre que al de una bandada o un banco. El algoritmo ha visto varias evoluciones [38], y la expuesta en esta sección recogerá las características más importantes de las mismas.

Sea un problema de optimización *D-dimensional* sin restricciones con una función objetivo $f(X)$, sujeta a un espacio de búsqueda arbitrario. Para aplicar el algoritmo PSO, en primer lugar debe inicializarse una *población* de partículas, P, de tamaño N, y asignarse una posición aleatoria a cada una dentro del espacio de búsqueda. La elección del tamaño de la población depende de varios factores, entre los cuales destacan la dificultad percibida del problema o información empírica de la que se disponga sobre el mismo. Un buen número suele encontrarse entre 20 y 50, aunque esto puede cambiar radicalmente en función del problema y de la potencia de computación disponible. Asimismo, la elección del número de iteraciones, K, obedece a los mismos criterios.

Para un agente i de la población se tienen los siguientes parámetros:

- Posición: $\vec{x_i}$. Representa el valor del agente i en cada una de las d componentes del problema en la iteración t.
- Velocidad: $\vec{v_i}$. Representa el cambio de posición de la partícula i entre las iteraciones t y $t+1$.
- Mejor posición local: $p\vec{best}_i$. Representa la mejor posición histórica del agente i, es decir, la que devuelve el valor más pequeño de la función objetivo. Es común que en la literatura se conozca a la evaluación de la función objetivo como *fitness value*.
- Mejor posición global: $g\vec{best}$. Representa la mejor posición alcanzada por cualquiera de los agentes de la población. Si bien este parámetro es compartido entre todas las partículas, es necesario en los cálculos que modifican la velocidad de cada agente individual.

Algoritmo 1 Ejecución del algoritmo PSO

1: Inicializar la población P con posiciones y velocidades aleatorias.
2: **while** $t < K$ **do**
3: Evaluar la función objetivo, $f(\vec{x_i})$ para cada partícula. Si $f(\vec{x_i}) < f(\vec{pbest_i})$, entonces $\vec{pbest_i} \leftarrow \vec{x_i}$.
4: Encontrar el mejor agente de la población en la iteración t, es decir, $\vec{x^*}$ tal que $f(\vec{x^*}) \leq f(\vec{x}) \, \forall \vec{x} \in P$. Si $f(\vec{x^*}) < f(\vec{gbest})$, entonces $\vec{gbest} \leftarrow \vec{x^*}$.
5: Actualizar la velocidad de cada agente $\vec{v_i}$ conforme a la ecuación:

$$\vec{v_i} \leftarrow c_1\vec{r_1} \otimes (\vec{pbest} - \vec{x_i}) + c_2\vec{r_2} \otimes (\vec{gbest} - \vec{x_i}) + w\vec{v_i} \tag{3.1}$$

6: Actualizar la posición de cada agente $\vec{x_i}$ conforme a la ecuación:

$$\vec{x_i} \leftarrow \vec{x_i} + \vec{v_i} \tag{3.2}$$

7: Aumentar el contador de iteraciones: $t \leftarrow t + 1$
8: **end while**
9: Devolver $\vec{gbest}$ como solución del problema.

La ejecución del algoritmo, adaptada de [38] es como sigue:

Donde el parámetro w se conoce como *inercia*, cuya definición puede adoptar varias formas, tanto estáticas como dinámicas, los coeficicentes c_1 y c_2 son los coeficientes *local* y *global* respectivamente, y los vectores $\vec{r_1}$ y $\vec{r_2}$ son vectores de dimensión D cuyas componentes se generan en cada iteración a partir de una distribución uniforme $U(0, 1)$. El símbolo $\otimes$ denota el producto de Hadamard, o producto elemento a elemento.

Como puede observarse, la ecuación 3.1 es el corazón del algoritmo. Cada uno de los tres sumandos del miembro derecho corresponden a un mecanismo metaheurístico claro y tienen una función definida, a saber:

1. El término $c_1\vec{r_1} \otimes (\vec{pbest} - \vec{x_i})$ determina el comportamiento de la partícula con respecto a sí misma, es decir, cuán atraída está a la mejor posición que ha conocido a lo largo de la ejecución del algoritmo. En algunos artículos se conoce como término de nostalgia o individual. La inclusión del término aleatorio $\vec{r_1}$ ayuda a mejorar la estabilidad de las partículas y favorece una explotación errática y oscilante del espacio de búsqueda

2. El término $c_2\vec{r_2} \otimes (\vec{gbest} - \vec{x_i})$ representa el mecanismo de intercambio de información entre una partícula y el resto del enjambre, y aporta el aspecto social del algoritmo. Determina cuán atraídas están las partículas del enjambre a la mejor posición alcanzada por la mejor de ellas. Al igual que en el término anterior, la inclusión de $\vec{r_2}$ convierte las trayectorias de exploración en erráticas, lo que mejora la calidad de la búsqueda.

3. El término $w\vec{v_i}$ representa la inercia del agente. El coeficiente de inercia pondera la importancia de la información obtenida en la iteración anterior a la hora de continuar explorando el espacio de búsqueda si la velocidad anterior era grande, o de explotar la región local si la velocidad anterior era pequeña. Además, en las iteraciones iniciales favorece la exploración, ya que en la inicialización aleatoria de la posición de los agentes puede darse la situación de que la partícula comience desde un mínimo local.

Tanto el término individual como el colectivo pueden interpretarse como fuerzas externas que actúan sobre una partícula. Con el objetivo de hacer de esta interpretación una que tenga un significante común en el mundo real, puede entenderse que cada partícula está conectada a su mejor posición con un muelle cuya constante elástica media es $k_1 = c_1/2$, mientras que el muelle que la conecta a la mejor posición histórica alcanzada por la población tiene una constante elástica media $k_2 = c_2/2$ [38].

La manipulación de los coeficientes local y global determina, por tanto, la ponderación entre los mecanismos de exploración y explotación y es la principal entrada de conocimiento preexistente al algoritmo. Si se prevé un espacio de búsqueda grande, y apenas hay información sobre dónde puede encontrarse la solución, convendrá tener un coeficiente c_2 comparativamente grande sobre c_1. En cambio, si es conocido que la solución del problema es relativamente cercana a una solución anterior, puede inicializarse una partícula dicho punto y conferir un peso mayor al coeficiente local sobre el global. En definitiva, el ajuste de ambos coeficientes es crucial para determinar la estabilidad del algoritmo.

Una convención que parece haberse hecho un estándar dentro de la literatura relacionada con este optimizador es la de mantener $c_1 + c_2 < 4$. Hay otras aproximaciones que incluyen otros parámetros que permiten superar esta desigualdad y refinar aún más la elección de los parámetros, pero para la grandísima mayoría de problemas para los que se ha demostrado que este algoritmo tiene un buen desempeño, esta regla práctica es adecuada.

Continuando con el símil físico anterior, el parámetro de inercia da una idea de la fluidez del medio en el que se mueven las partículas, esto es, cuán permisivo es el medio a los cambios de posición de las mismas. Si se entienden los dos primeros términos de 3.1 como una fuerza externa a la partícula $\vec{f_i}$, la ecuación de aceleración puede reescribirse como:

$$\Delta\vec{v_i} = \vec{f_i} - (1 - w)\vec{v_i} \tag{3.3}$$

Atendiendo a esta ecuación, la constante $(1 - w)$ actúa como un coeficiente de rozamiento, penalizando velocidades grandes cuanto más pequeño sea w.

La elección del coeficiente w requiere de especial atención. A primera vista, parece obvio que un valor estático no puede ofrecer resultados satisfactorios, porque si bien al inicio de la ejecución del algoritmo es interesante que las partículas puedan tener velocidades altas (de forma que puedan vencer los posibles mínimos locales en los que puedan verse atrapadas), en las iteraciones finales interesa que los agentes se centren más en la explotación de una región concreta del espacio de búsqueda con velocidades reducidas que eviten la posible salida del mínimo global a encontrar. Una posible solución a este problema es elegir el coeficiente de inercia conforme a:

$$w = w_0 + (w_f - w_0)\frac{t}{K} \tag{3.4}$$

donde w_0 y w_f son los coeficientes de inercia inicial y final, respectivamente, t es el número de iteración actual y K el número total de iteraciones. Valores comunes para los coeficientes inicial y final son 0.9 y 0.4, respectivamente. En [11], Eberhart y Shi proponen el empleo de un sistema borroso para elegir de forma adecuada el término de inercia, reportando resultados muy superiores a la implementación original o al decrecimiento lineal. Los mismos autores, en [12], proponen elegir w conforme a una distribución $U(0{,}5, 1)$.

3.4.2. Phasor Particle Swarm Optimization (PPSO)

Phasor Particle Swarm Optimization (PPSO) es una modificación del algoritmo PSO canónico propuesta por Ghasemi, Akbari et al. [17], basada en el modelado de los parámetros de control de la ecuación 3.1 mediante un ángulo de fase θ [17]. Mediante esta modificación, el algoritmo se convierte en un optimizador adaptativo, trigonométrico y que no requiere de parámetros que determinen el comportamiento de las partículas.

En una primera instancia, puede parecer contraintuitivo el hecho de eliminar parámetros de configuración de una estrategia de optimización cuyo punto fuerte es la posibilidad de la inclusión de información y conocimiento previo para mejorar la calidad de la solución. Para poder tener una idea adecuada de por qué se eliminan los parámetros en este algoritmo, debe considerarse el cometido de los coeficientes de ajuste dentro del optimizador original; como se ha descrito en la sección anterior, el principal objetivo de los coeficientes es ponderar los mecanismos de exploración y explotación, así como evitar problemas de convergencia en mínimos locales. Los proponentes de la modificación aseguran haber modelado estos comportamientos mediante la inclusión de un ángulo fasorial y una manipulación adecuada del mismo

en la ecuación de actualización de velocidad empleándolo como argumento de varias funciones. [2]

El problema a emplear para ilustrar la ejecución de este algoritmo es el mismo que en la subsección anterior: un problema de minimización de una función $f(X)$ *d-dimensional*, sin restricciones, con un espacio de búsqueda acotado arbitrario. Asimismo, al igual que en el anterior algoritmo, se distinguen los siguientes parámetros para cada uno de los miembros i de la población P:

- Posición: $\vec{x_i}$. Representa el valor del agente i en cada una de las d componentes del problema. La componente d del agente i en la iteración t se denotará como x_{id}^t
- Velocidad: $\vec{v_i}$. Representa el cambio de posición de la partícula i entre las iteraciones t y $t+1$.
- Mejor posición local: $\vec{pbest_i}$. Representa la mejor posición histórica del agente i, es decir, la que devuelve el valor más pequeño de la función objetivo.
- Mejor posición global: $\vec{gbest}$. Representa la mejor posición alcanzada por cualquiera de los agentes de la población. Si bien este parámetro es compartido entre todas las partículas, es necesario en los cálculos que modifican la velocidad de cada agente individual.
- Ángulo de fase: θ_i. Parámetro fasorial que regula el comportamiento del agente i en la iteración t.

La ejecución del optimizador PPSO puede observarse en el algoritmo 2. En lugar de emplear notación vectorial como en el optimizador PSO, en este caso se ha optado por denotar las componentes de dicho vector, asumiendo que el tratamiento en el algoritmo es el mismo para todas ellas.

La naturaleza intermitente de las funciones seno y coseno, unidas al empleo de su valor absoluto, permite modificar los términos de nostalgia y social de la ecuación 3.5 de forma que en algunas iteraciones uno toma un valor mucho más importante que sobre otro, mientras que en la mayoría de bucles de ejecución se mantienen comparativamente similares. Cabe destacar que, a pesar de que en la implementación original de PSO la suma de los coeficientes local y global no podía superar el valor de 4 por riesgo de introducir el algoritmo en una dinámica inestable, esta limitación desaparece en PPSO, ya que se adjudican límites duros a la velocidad de los agentes en cada una de las dimensiones del problema. Además, estos límites

[2] Dado que en esta modificación se emplea de forma extensiva el número de iteración actual, se incluirá en la notación como el superíndice t.

Algoritmo 2 Ejecución del algoritmo PPSO

1: Inicializar la población P con los siguientes parámetros:

$$\begin{aligned} x_{id}^{t=1} &= x_{min,d} + U(0,1)(x_{max,d} - x_{min,d}) \\ \theta_i^{t=1} &= U(0, 2\pi) \\ v_{max,id}^{t=1} &= 0{,}5(x_{max,d} - x_{min,d}) \end{aligned}$$

2: Obtener los valores iniciales de $pbest_i$ y $gbest$ para cada partícula:

$$\vec{pbest}_i^{t=1} = \vec{x_i} \qquad \vec{gbest}^{t=1} = \text{mín}\, f(\vec{x_i}) \quad i = 1, \ldots, N$$

3: **while** $t < K$ **do**

4: Calcular la velocidad de cada agente $\vec{v_{id}^t}$ conforme a la ecuación:

$$v_{id}^t \leftarrow |\cos\theta_i^t|^{2\sin\theta_i^t} \times (pbest_{id} - x_{id}^t) + |\sin\theta_i^t|^{2\cos\theta_i^t} \times (gbest_d - x_{id}^t) \tag{3.5}$$

5: Comparar la velocidad v_{id}^t con respecto a $v_{max,id}^t$:

$$v_{id}^t \leftarrow \text{mín}(\text{máx}(v_{id}^t, -v_{max,id}^t), v_{max,id}^t)$$

6: Actualizar la posición de cada agente x_{id} conforme a la ecuación:

$$x_{id}^{t+1} \leftarrow x_{id}^t + v_{id}^t \tag{3.6}$$

7: Comparar la posición x_{id}^{t+1} con respecto a las fronteras del espacio de búsqueda:

$$x_{id}^{t+1} \leftarrow \text{mín}(\text{máx}(x_{id}^{t+1}, x_{min,d}), x_{max,d})$$

8: Actualizar θ_i y $v_{max,id}$ conforme a:

$$\theta_i^{t+1} \leftarrow \theta_i^t + 2\pi|\cos(\theta_i^t) + \sin(\theta_i^t)| \tag{3.7}$$

$$v_{max,id}^{t+1} \leftarrow |\cos(\theta_i^{t+1})|^2 \times (x_{max,d} - x_{min,d}) \tag{3.8}$$

9: Evaluar la función objetivo, $f(\vec{x_i})$ para cada partícula. Si $f(\vec{x_i}) \leq f(\vec{pbest}_i)$, entonces $\vec{pbest}_i \leftarrow \vec{x_i}$.

10: Encontrar el mejor agente de la población en la iteración t, es decir, $\vec{x^*}$ tal que $f(\vec{x^*}) \leq f(\vec{x})\ \forall \vec{x} \in P$. Si $f(\vec{x^*}) \leq f(\vec{gbest})$, entonces $\vec{gbest} \leftarrow \vec{x^*}$.

11: Aumentar el contador de iteraciones: $t \leftarrow t + 1$

12: **end while**

13: Devolver $\vec{gbest}$ como solución del problema.

son dinámicos y se adaptan al estado de ejecución del algoritmo, asegurando su estabilidad.

Otra característica importante en la ecuación 3.1 es que no tiene en cuenta el término de inercia $w\vec{v_i^t}$. La eliminación de este término, unida a la refactorización de los coeficientes c_1 y c_2 en las funciones $p(\theta_i^t) = |\cos\theta_i^t|^{2\sin\theta_i^t}$ y $g(\theta_i^t) = |\sin\theta_i^t|^{2\cos\theta_i^t}$, elimina por completo la necesidad de ajustar parámetros intrínsecos del algoritmo, limitándose el ajuste únicamente a elegir un tamaño de población y un número de iteraciones que se ajusten a las particularidades del problema a resolver.

Es importante destacar que, además de los parámetros intrínsecos al algoritmo, esta modificación también elimina los términos aleatorios que se incluían en la propuesta original como forma de potenciar los mecanismos de exploración y explotación. Más allá de la inicialización de la posición inicial y el ángulo de fase de los agentes, no se incluye ninguna otra distribución uniforme, lo que confiere un carácter ciertamente determinista al proceso de búsqueda, en la medida en que una misma inicialización de la población, enfrentada al mismo problema, obtendrá la misma solución como resultado.

Con respecto a las mejoras que supone el incremento en complejidad del algoritmo con respecto a la función del mismo, los proponentes concluyen en [17] que PPSO funciona mejor que la implementación original y gran parte de las modificaciones que se han propuesto desde 1995. Por otro lado, afirman que el empleo de la estrategia fasorial añade estabilidad y robustez al algoritmo cuando el número de variables independientes aumenta.

Por último, y quizá la conclusión más importante que puede obtenerse del análisis de la aproximación de fase, es la posibilidad de incluir este mecanismo en otros algoritmos diferentes combinando estrategias propuestas por otros autores, de forma que la eliminación de la necesidad de ajuste y el aumento de robustez ocasionan que las versiones que incorporan el mecanismo de fase siempre obtienen mejores soluciones que las que no lo hacen.

4. Aplicaciones en conducción autónoma

En este capítulo se presentarán algunas implementaciones desarrolladas en el mundo ingenieril de los conceptos desarrollados anteriormente enfocados en la disciplina de la conducción autónoma. En primer lugar se estudiarán dos instancias de control MPC para verificar la viabilidad de esta estrategia para controlar un vehículo sin conductor, y posteriormente se presentará en detalle una implementación propia, desarrollada sobre la plataforma de simulación de control MPC del grupo de investigación Intelligent Vehicles and Traffic Technologies (INVETT) de la Universidad de Alcalá.

4.1. Empleo de controladores MPC en conducción autónoma

El empleo del arquetipo MPC ha ganado tracción durante los últimos años en la industria automovilística, perfilándose como una de las mejores opciones de futuro en lo que se refiere al control lateral y longitudinal de la posición de un vehículo autónomo a lo largo de una trayectoria a seguir. La principal razón detrás de esta evolución se debe, principalmente, a la mejora de los sistemas de cómputo disponibles en los ámbitos industrial y de investigación. Como se ha discutido en secciones anteriores, una de las principales características del control MPC es que su ancho de banda está severamente limitado por las capacidades del *hardware* que se emplea en su implementación o, visto de otra forma, por las constantes de tiempo que dominan el proceso de control a afrontar.

En la mayoría de aplicaciones del control MPC, el cuello de botella se encuentra en la etapa de optimización. Las evaluaciones de los modelos no suelen conllevar altas cargas computacionales a no ser que incluyan procesos de linealización -lo cual es relativamente común, puesto que el control MPC no lineal, pese a estar desarrollado y ser efectivo, es poco eficiente- o parámetros dinámicos que dependan de medidas realizadas sobre el propio proceso. En este último caso, puede aumentarse la velocidad del controlador con sensores más rápidos o una mejor electrónica de proceso de la señal proporcionada por el sensor. La optimización de las secuencias de control, por otro lado, representa un problema computacional grave si la topología de la función de coste no es benigna.

En lo relativo a la conducción autónoma, las restricciones temporales son absolutamente cruciales para garantizar la seguridad del proceso. Si se fuese a comparar un sistema de control contra un conductor real, se observaría que el periodo de muestreo necesario para designar secuencias de control viables está en el entorno de los 50 milisegundos. Por supuesto, esto depende fuertemente en las condiciones

del entorno; en rectas sin tráfico quizá pueda acometerse una acción de control cada medio segundo, mientras que en entornos urbanos con densidad de tráfico media, peatones y restricciones viales, el periodo de planificación de acciones puede ser inferior a los 20 milisegundos. En principio, parecería que puede adaptarse el periodo de computación a las condiciones externas, pero un análisis en tiempo real de las mismas derrota el propósito que quiere conseguirse, que es el de hacer el controlador más rápido. La solución es, pues, que el controlador lo más rápido posible, considerando que la seguridad del proceso de control prima sobre otros aspectos como la eficiencia energética.

El problema de la conducción autónoma debe considerar en todo instante, al menos, tres variables de control: el giro de volante, la acción sobre el acelerador, y la acción sobre el freno. El controlador MPC que se proponga para solucionarlo devuelve secuencias de control sobre estas tres variables, de forma que el seguimiento de la trayectoria que se le proponga sea viable. Es obvio que este problema, en lo que a la optimización de la función de coste se refiere, es multi-restringido. No sólo debe considerarse la posible saturación de las variables de control, sino que además hay que incorporar dentro de las consideraciones del seguimiento de trayectoria la normativa vial, los límites de velocidad, otros usuarios de la carretera y posibles situaciones instantáneas de peligro, entre otros muchos aspectos.

El seguimiento de trayectorias en entornos benignos es un problema ya resuelto y que cuenta con soluciones comerciales basadas en un control de bajo nivel que requiere una estrecha integración entre la mecánica del vehículo y los sistemas de control. Si se considera un controlador MPC, el problema se encuentra en un buen estado, con soluciones ciertamente viables implementadas en vehículos de calle en formato de prototipo. Por desgracia, no puede decirse lo mismo de otros aspectos que conforman la conducción autónoma, como puede ser la identificación de usuarios vulnerables de la vía de forma efectiva para garantizar su protección o la realización de maniobras de adelantamiento. Dos de estos problemas restantes se describen en los siguientes párrafos.

Un gran problema que se manifiesta en la literatura especializada en controladores MPC autónomos es la falta de sensación de seguridad, asociada a movimientos erráticos del volante de pequeña amplitud, provocados por la intención del controlador de situarse siempre exactamente donde indica la referencia, y a acciones diferenciales sobre el freno y el acelerador que provocan un balanceo constante de la inercia del vehículo. Esta problemática puede solucionarse en diversos puntos de la etapa de optimización.

Una de ellas es la inclusión de restricciones explícitas dentro del optimizador que se ocupen de imponer límites a la variación de las señales de control, es decir, imponer restricciones sobre las derivadas del giro del volante y la acción del acele-

rador y el freno. Esta aproximación garantiza la seguridad y la comodidad de los ocupantes del vehículo, puesto que en ningún momento pueden superarse los umbrales que se consideren en estas restricciones. Si las imposiciones sobre estas acciones se hacen de forma dinámica -analizando el estado del vehículo y haciendo uso de las predicciones del controlador MPC para determinar estados futuros, así como de la información de la trayectoria futura que debe seguir el vehículo-, es posible ajustarlas a lo largo del tiempo para que se adecúen a las necesidades del sistema de control. Esta solución es fácil de implementar en optimizadores que permiten restricciones, pero la dinamización de las mismas requiere de un estudio complejo y puede suponer un incremento importante del tiempo de cómputo, así que no siempre es deseable.

Otra posible solución es la inclusión de un término en la función de coste que tenga en cuenta las variaciones de las señales de control y penalice componentes de alta frecuencia de las mismas. De esta forma se permite ponderar la importancia del que podemos llamar *término de confort* frente a otros aspectos del proceso de control del vehículo autónomo, de forma que pierda importancia frente al seguimiento de referencia cuando el error sobre la misma sea grande, y que prime sobre este cuando la trayectoria transcurra por zonas de fácil navegación -rectas o curvas suaves-. Esta implementación es más simple que la anterior, pero añade un término más al proceso de optimización. Además, es necesario considerar cómo se realiza la ponderación de unos términos frente a otros.

Los autores de [50] proponen en su trabajo *Path tracking control for autonomous vehicles based on an improved MPC* una aproximación que hace uso de un controlador MPC para garantizar el seguimiento de referencias de trayectoria. Una de las características principales de esta propuesta es que hace uso de un sistema borroso para determinar de forma adaptativa los pesos de la función de coste.

Los autores describen un modelo dinámico del vehículo basado en el estudio de las fuerzas longitudinales y laterales que sufren los neumáticos. Este modelo da lugar a una representación en espacio de estados -SSM- fuertemente no lineal y continuo. La naturaleza predominantemente computacional del control MPC obliga a la discretización del modelo, y la poca eficiencia del control no lineal, a su linealización. La construcción de una función de coste alrededor de este modelo da lugar a un funcional J cuadrático, cuya optimización se afronta mediante la estrategia Quadratic Programming.

Los autores proponen el desarrollo de un sistema experto borroso de primer orden que relacione la posición lateral del vehículo y su orientación con respecto a las definidas en la trayectoria mediante un proceso de borrosificación, inferencia borrosa y desborrosificación cuya salida modifique de forma adaptativa los pesos de la función de coste. Un desarrollo completo de este sistema y su aplicación en el entorno de simulación CarSim-Matlab/Simulink puede encontrarse en [50].

Entre las conclusiones más importantes de la incorporación de la lógica borrosa dentro del flujo de trabajo del controlador MPC, los autores destacan la suavidad de las secuencias de control -lo que se traduce en una experiencia mucho más confortable para el pasajero- y el hecho de que la estabilidad lateral del vehículo está garantizada en el seguimiento de la trayectoria gracias a la acción del sistema borroso.

Otro gran problema del control MPC autónomo es el tratamiento de la posibilidad de colisiones. En [25] se propone una formulación de control MPC en la que las matrices de coste permanecen estáticas, pero la generación de la trayectoria local a seguir viene determinada por un campo potencial. En este trabajo se hace uso de un modelo que asemeja el comportamiento del vehículo al de una bicicleta, discretizado mediante un bloqueador de orden cero y linealizado mediante un modelo incremental, siguiendo una idea similar al modelo en pequeña señal de convertidores de potencia descrito en [13].

La generación de este campo potencial asigna fuentes de campo a los límites de la carretera y a los vehículos del entorno, realizando mediciones de posición y velocidad relativa adaptadas mediante un cambio de sistema de referencia en el que el vehículo a controlar permanece estático. Asimismo, se asigna un sumidero de campo al centro del carril, para asegurar el seguimiento de la trayectoria. El campo resultante se calcula mediante superposición, y la trayectoria se elige generando una función de coste que incluye la minimización del campo y la interpolación mediante *splines* cúbicos entre los puntos del mismo de menor coste.

Esta trayectoria se emplea como referencia de un controlador MPC, cuya función de coste incluye consideraciones adicionales sobre los actuadores -restricciones sobre el valor absoluto del giro de volante, así como de la derivada del mismo- y sobre la estabilidad del vehículo -límites sobre el ángulo de deslizamiento y la derivada de la orientación-.

Los autores simulan el comportamiento de esta aproximación en el mismo entorno que en la referencia anterior, CarSim-Matlab/Simulink. Los resultados de sus experimentos reportan que esta aproximación al tratamiento del riesgo de colisión es efectiva, evitando que las trayectorias de los ocupantes de la carretera se crucen provocando posibles accidentes.

4.2. Implementación propia

4.2.1. Controlador MPC

El controlador MPC empleado incluye un modelo cinemático de un vehículo basado en el concepto de una bicicleta, y no emplea información dinámica del comportamiento de las ruedas. Este modelo es válido para velocidades de hasta unos

$8m/s$, cantidad suficiente para el entorno de simulación que se va a manejar y el concepto que quiere demostrarse con la misma. Un esquema puede consultarse en las figuras 4.1 y 4.2

El modelo empleado consiste en las siguientes ecuaciones:

$$\begin{cases} \tan\delta & = \frac{L}{R} \\ \dot{\theta} & = \frac{v}{R} = \frac{v\cos\beta\tan\delta}{L} \\ \dot{x}_r & = v\cos(\theta+\beta) \\ \dot{y}_r & = v\sin(\theta+\beta) \\ \beta & = \tan^{-1}(\frac{l_r\tan(\delta)}{L}) \end{cases} \tag{4.1}$$

donde x_r e y_r representan la posición longitudinal y lateral del vehículo respectivamente, θ representa el ángulo de guiñada -más conocido en el mundo ingenieril por su nombre en inglés, *yaw*- X, δ simboliza el giro de las ruedas -relacionado de forma directamente proporcional con el del volante- y β el ángulo de deslizamiento, definido como la diferencia entre la dirección a la que apuntan las ruedas y en la que realmente se mueven.

La variable sobre la que se actúa es δ, relacionada con el giro de volante por un multpilicador de 17. No se consideran acciones sobre acelerador o freno, ya que el objetivo es mantener una velocidad constante.

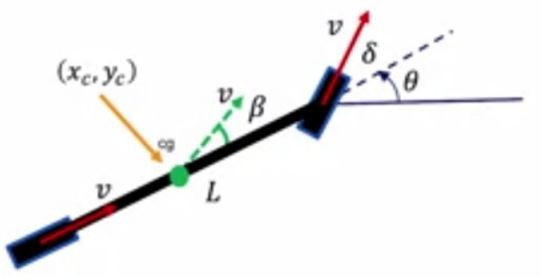

Figura 4.1: Modelo del vehículo referenciado a su centro de gravedad.

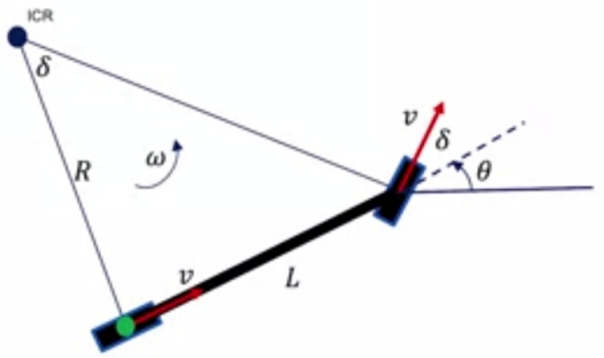

Figura 4.2: Modelo del vehículo referenciado a las ruedas traseras.

Por su parte, la función de coste toma la forma de:

$$J = \sum_{j=N_1}^{N_2} \alpha[\hat{y}(t+k|\,t) - y_{ref}(t+j)]^2 + \sum_{j=1}^{N_u} \kappa[\theta(t+j) - \theta_{ref}(t+j)]^2 + \sum_{j=1}^{N_u} \lambda[\theta(t+j) - \theta(t+j-1)]^2 \tag{4.2}$$

donde $N_1 = 1$, $N_2 = 3$, $N_u = 3$, α se corresponde con el peso asociado al error de posición, κ con el de orientación o *yaw*, y λ pondera la contribución de la velocidad de rotación del vehículo dentro de la función de coste, empleando uno de los mecanismos descritos en la sección anterior para mejorar la estabilidad del vehículo y evitar componentes de alta frecuencia en la señal de control.

4.2.2. Entorno de trabajo y simulación

La implementación del algoritmo PSO dentro del controlador MPC se ha realizado sobre el simulador de comportamiento de vehículos CARLA [10], una aplicación de código abierto y gratuita diseñada para facilitar la investigación y el desarrollo dentro de la conducción autónoma. El controlador MPC desarrollado está escrito en lenguaje Python, y todo el sistema se ha ejecutado sobre las versiones 18.04 y 20.04 del sistema operativo Ubuntu.

La simulación se ha configurado de forma que el hardware sobre el que corre no tenga influencia alguna sobre los resultados de la misma. Se ha elegido un periodo de 100 milisegundos para cada iteración de la simulación, lo que se traduce en que el mundo se para cada 100 milisegundos para realizar los cálculos necesarios relativos al controlador; esto permite además hacer mediciones fiables del tiempo de cómputo de la estrategia de control propuesta, uno de los puntos diferenciales que probarán si es viable como solución a la conducción autónoma o no.

El recorrido elegido consta de tres rectas unidas entre sí por dos curvas abiertas. A primera vista puede parecer que no es un recorrido adecuado para determinar si una estrategia de control, pero en el apartado de conclusiones se demuestra por qué el recorrido no tiene influencia alguna en el desempeño del controlador.

4.3. Desarrollo experimental

Durante la implementación del algoritmo PSO en el controlador MPC mencionado se realizaron una serie de ensayos para determinar los mejores parámetros en la configuración del optimizador. Esta subsección recoge las conclusiones más importantes de estos ensayos.

- Tamaño de población. El tamaño de población es uno de los parámetros más importantes del algoritmo PSO. A priori, es fácil caer en la suposición de

que cuanto más grande sea este parámetro, mejores soluciones proporciona el algoritmo, pero en la práctica esto no se cumple.

Para valores pequeños -menores que 10-, se observa que la población no es lo suficientemente grande como para que el espacio de búsqueda sea explorado de forma efectiva, lo que a su vez perjudica la explotación. El tiempo de ejecución es pequeño, pero la solución devuelta por el optimizador no es la óptima, ya que no se alcanza el mínimo de la función de coste. Esto se traduce en un comportamiento inestable del vehículo, que oscila alrededor del centro del carril sin seguir la trayectoria satisfactoriamente.

Valores entre 10 y 50 proporcionan el mejor comportamiento al suponer un compromiso entre la velocidad de ejecución y la precisión de las soluciones. La suavidad del comportamiento del vehículo y la continuidad de las soluciones de control llevan a concluir que esta es la franja óptima de funcionamiento para este problema.

El comportamiento del algoritmo es sensiblemente peor a la franja anterior cuando el tamaño de población supera el valor de 50. Si bien es cierto que la calidad de las soluciones mejora, la mejora es demasiado marginal como para que sea beneficioso. Por el contrario, el tiempo de ejecución aumenta de forma excesiva, al igual que la cantidad de memoria empleada.

Una posible estrategia para emplear tamaños de población grandes puede ser la de implementar subpoblaciones que se comuniquen entre ellas con el mecanismo social de transmisión de información, mientras que son los mejores individuos de cada subpoblación los que interactúan entre ellos de forma global. Esto reduce la carga computacional de comparar todos los individuos con todos los demás y mejora la transmisión de información entre conjuntos. En cualquier caso, sería necesaria una implementación en *hardware* demasiado potente como para que sea práctico su uso.

El valor elegido como óptimo en el comportamiento del algoritmo es de 20 agentes en la población.

- Número de iteraciones. Al igual que el tamaño de población, el número de iteraciones es crucial a la hora de valorar el desempeño del algoritmo. Es obvio que cuanto mayor sea este parámetro, mayor será el tiempo de ejecución, por lo que es necesaria una elección cuidadosa para que el comportamiento sea el deseable.

  Si el algoritmo se ejecuta menos de 10 veces por cada iteración de control, se observa que las soluciones son muy pobres, porque no hay tiempo suficiente para encontrar el mínimo global ni de explotar de forma efectiva una solución

no óptima que devuelva un comportamiento siquiera razonable. Además, si se inicializase una partícula dentro de una zona con un mínimo local, no habría tiempo suficiente para escapar de ella, ya que el decrecimiento del parámetro de inercia sería demasiado rápido. En recta el comportamiento quizá pueda ser aceptable ya que donde más variabilidad se aprecia es en la tercera solución de la secuencia de control, pero en curva el comportamiento es completamente inestable.

Cuando el número de iteraciones se encuentra entre 10 y 100 se alcanza un buen compromiso entre velocidad de ejecución y búsqueda del mínimo global. El comportamiento del vehículo se suaviza al disponer de soluciones más refinadas y la trayectoria es seguida con apenas unos centímetros de error en recta. En curva el error lateral aumenta hasta los 80 centímetros con respecto al centro del carril, valor más que asumible teniendo en cuenta las dimensiones del mismo.

Por último, cuando el número de iteraciones es superior a 100, el algoritmo se vuelve lento y torpe, ya que el tiempo de ejecución aumenta hasta niveles inaceptables, mientras que el comportamiento no es excesivamente mejor que en la franja anterior.

Una buena forma de emplear la información extraída de esta experimentación puede ser implementar una forma de elección dinámica del número de iteraciones. En recta se puede trabajar con valores pequeños, pero en curva son necesarios valores más altos. Esto puede hacerse haciendo un estudio de la derivada del *yaw* de la referencia mediante un sistema borroso.

Tras la experimentación, se ha encontrado que 20 iteraciones es un valor adecuado cuando se considera el compromiso entre la calidad de las soluciones y el tiempo de ejecución.

- Coeficientes local y global. Durante el estudio de parámetros, se ha detectado que el impacto de estos coeficientes -siempre referidos a la aplicación mencionada- tiene un impacto inferior al que cabría esperar, dado el rol que mantienen en la ecuación de actualización de velocidad (3.1).

  Si ambos coeficientes son similares, las partículas exploran su entorno de forma efectiva a la vez que emplean la información global disponible para determinar si se encuentran dentro de un mínimo global. Este comportamiento prácticamente asegura la devolución de la solución óptima del algoritmo, por más que la metaheurística no pueda asegurar optimalidad en la solución.

  Si el coeficiente local es comparativamente superior al global, se favorece mucho la explotación sobre la exploración. En el problema tratado, este compor-

tamiento es favorable, ya que la continuidad y la suavidad de la trayectoria garantizan que el mínimo global de la función de coste de la iteración actual se encontrará cerca del que se haya encontrado en la iteración anterior. No obstante, el beneficio puede verse reducido si la forma de la función no es benigna, ya que de forma efectiva se elimina la posibilidad de escapar de mínimos locales salvo que la inicialización aleatoria de velocidad sea particularmente favorable, lo que puede provocar problemas de convergencia temprana.

Por último, si el coeficiente global es comparativamente grande sobre el local, las partículas se encuentran demasiado atraídas por la que se encuentra en la mejor posición histórica, provocando una explotación deficiente y soluciones mal refinadas. De los tres casos estudiados, este es el que peor comportamiento exhibe.

Finalmente, atendiendo a los resultados de la experimentación, se decide un valor de 2,3 para el coeficiente local y de 1,1 para el global, manteniendo la condición $c_1 + c_2 < 4$ que se describe en la literatura como recomendada para mejorar la estabilidad del algoritmo.

- Coeficiente de inercia. Al igual que el parámetro anterior, las particularidades del problema a tratar aseguran que no es necesaria una exploración particularmente eficiente del espacio de búsqueda, lo que resta importancia a la elección del rango de este parámetro. La implementación de un decrecimiento lineal entre 0,9 y 0,4 [38] arroja un buen comportamiento, con una evolución a lo largo de la ejecución que se ajusta a las necesidades del problema.

4.4. Solución de control

Tras haber determinado los parámetros óptimos de ejecución, se sigue un procedimiento similar para determinar los pesos en la ecuación 4.2, detectándose que el comportamiento del controlador es aceptable siempre que los valores elegidos sean similares.

Si se prima cualquiera de los términos sobre los otros dos, se observa un aumento de la inestabilidad lateral del vehículo, ya que aunque pueda reducirse el error lateral beneficiando este término sobre los de orientación, se generan componentes oscilatorias en el giro de los neumáticos que no son deseables. Algo similar ocurre si se prioriza cualquiera de los dos términos de orientación: el comportamiento del vehículo se suaviza, pero el error lateral se hace grande y el seguimiento de la trayectoria no es satisfactorio.

Finalmente, se decide un valor de α de 2, un valor de κ de 1,5 y un valor de λ de 3,5. Esta elección viene motivada principalmente por la necesidad de reducir las

componentes de alta frecuencia de la secuencia de control, reguladas por λ, mientras que se mantiene un seguimiento saludable de las referencias de posición y orientación.

Como puede observarse en la figura 4.3, el seguimiento de la trayectoria en términos de posición hace que el recorrido descrito por el vehículo sea prácticamente indistinguible del que marca la referencia.

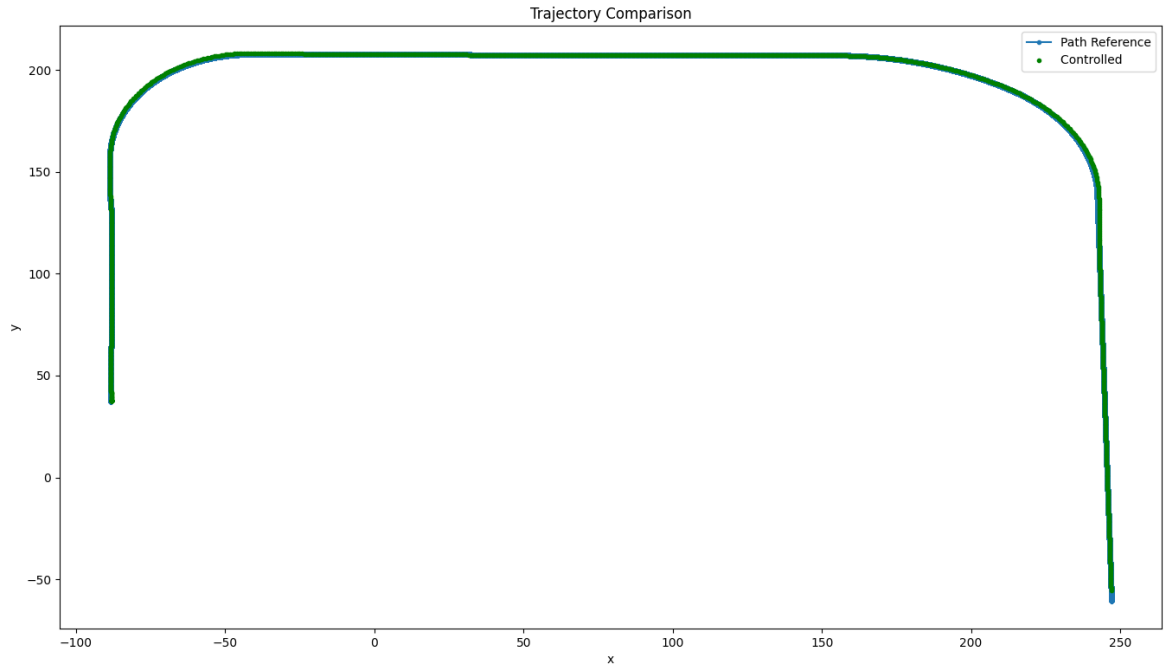

Figura 4.3: Trayectoria del vehículo.

Esta conclusión se refuerza a la vista de los errores laterales mostrados en la figura 4.4, que quedan acotados entre los 0,2 metros hacia la izquierda del centro del carril y 1,08 metros hacia la derecha. Además, se observa que la tendencia del controlador es a hacer converger la trayectoria en el centro del carril, pero no de forma que se ponga en peligro la estabilidad del vehículo. En lo que se refiere al error lateral, puede concluirse que el controlador muestra un comportamiento excelente.

Asimismo, si se observa el comportamiento del vehículo en términos de orientación, la figura 4.5 permite determinar que el seguimiento de la referencia de *yaw* es sobresaliente; destacan particularmente los instantes alrededor del segundo 75 de ejecución, en los que el coche ejecuta una maniobra de apertura para tomar la primera curva con mejor orientación.

La figura 4.6, que muestra el error de orientación, refuerza lo expuesto en el párrafo anterior. Los resultados obtenidos, con un error máximo de 0,19 radianes, son prácticamente inmejorables, teniendo en cuenta que, durante la ejecución en recta, el error medio es de 0,014 radianes y que en curva el error máximo supone una desviación de la orientación de referencia despreciable.

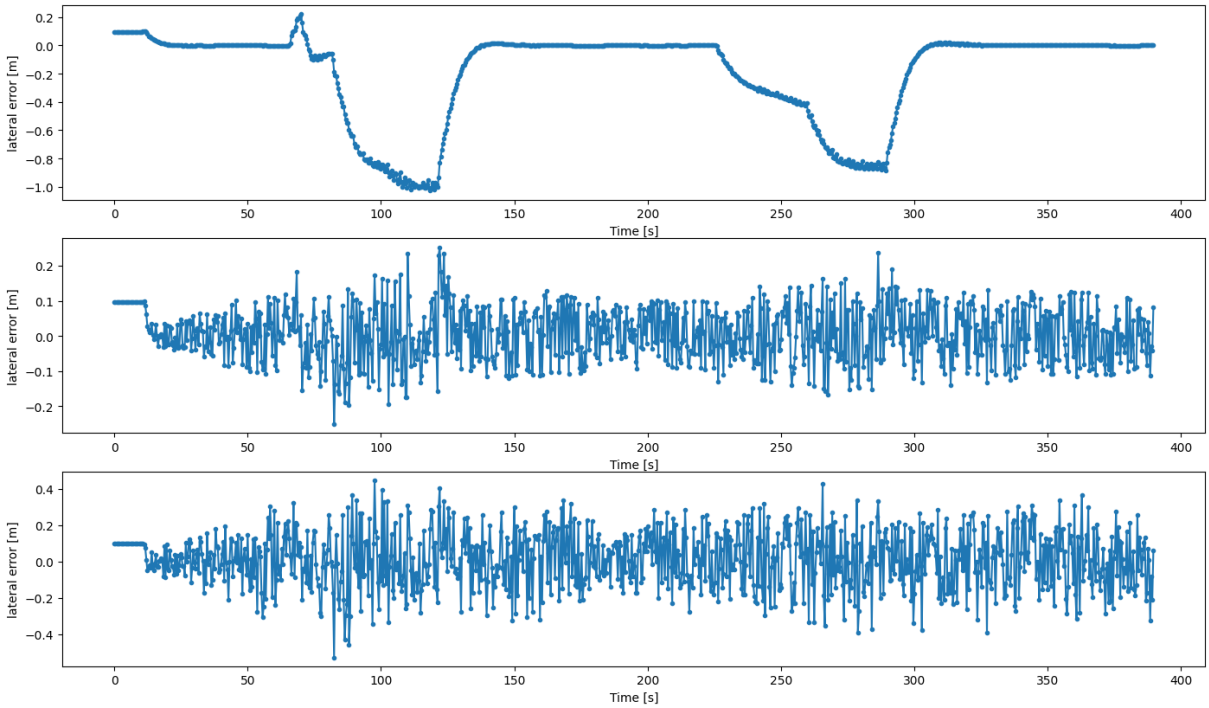

Figura 4.4: Error lateral predicho en la secuencia de control.

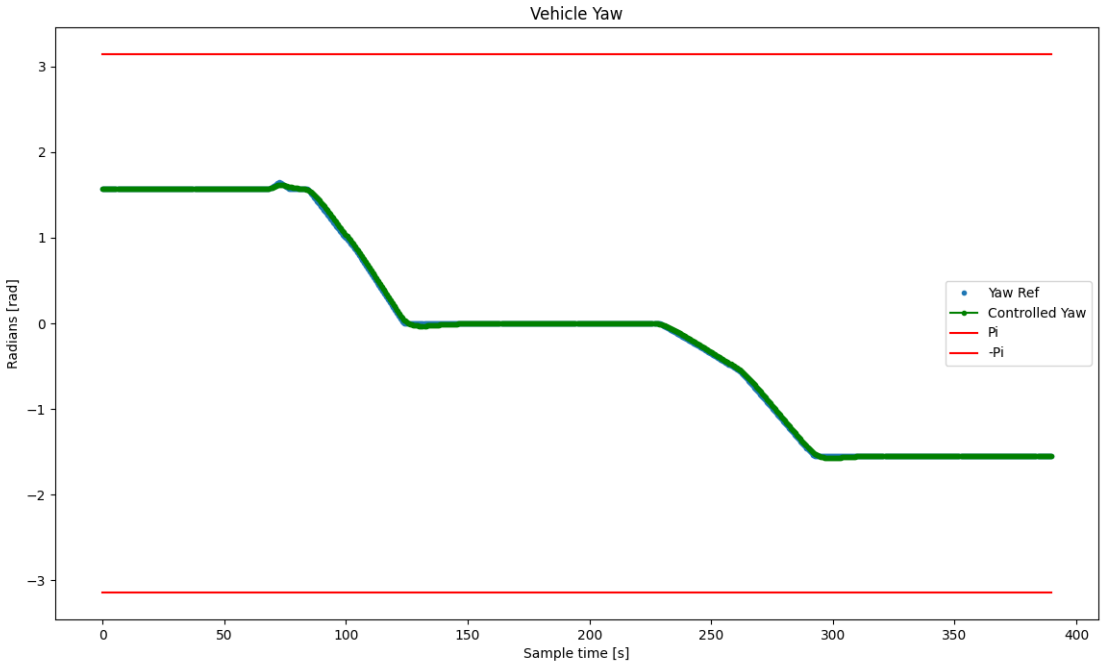

Figura 4.5: Orientación del vehículo.

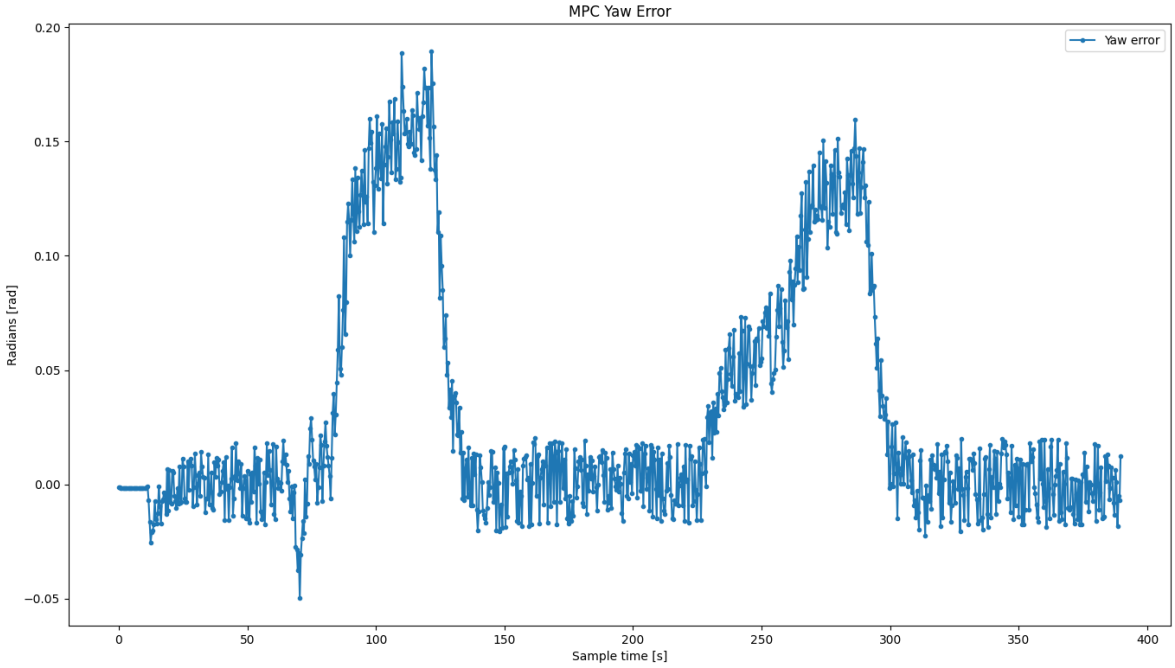

Figura 4.6: Error de orientación.

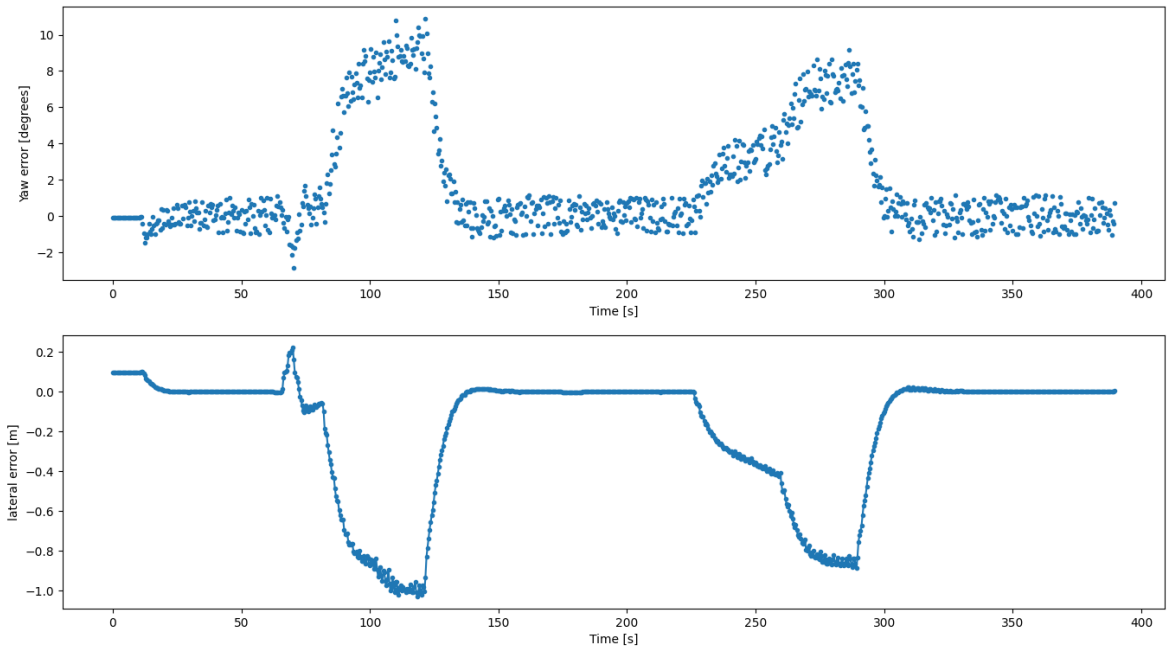

Figura 4.7: Comparación de los errores lateral y de orientación

Como conclusión al análisis de la ejecución del controlador se presenta la figura 4.7. En ella se observa cómo los errores de orientación y posición son complementarios, lo que se traduce en que cuando el vehículo se desvía hacia un lado con respecto al centro del carril, el controlador modifica la orientación mediante el giro de neumático, de forma que el error lateral tienda a cero conforme a lo mostrado en la figura 4.3.

5. Conclusiones y trabajos futuros

Durante este Trabajo de Fin de Grado se han expuesto las ideas principales del Control por Predicción de Modelo (MPC) y de la resolución de problemas de optimización de funciones mediante el empleo de algoritmos metaheurísticos. Además, se ha mostrado la integración de un técnica metaheurística concreta, la Particle Swarm Optimization (PSO) dentro de la estructura de un controlador MPC, como minimizador de la función de coste que evalúa las tentativas de control propuestas por el sistema. El objetivo de este apartado es el de juzgar si dicha integración es correcta, es decir, las acciones de control determinadas por el sistema son aceptables y, por otro lado, si el empleo de los algoritmos metaheurísticos dentro del proceso de optimización del controlador supone un beneficio sobre los algoritmos tradicionales, como pueden ser el Gradient Descent o el algoritmo COBYLA.

5.1. Análisis del controlador PSO

A la luz de lo expuesto en la sección anterior pueden extraerse una serie de ventajas y desventajas del controlador PSO con respecto a la alternativa que emplea optimizadores tradicionales.

En primer lugar, destaca la posibilidad de introducción de información sobre el problema de la que se dispone anteriormente a su ejecución. En la aplicación mostrada sobre un vehículo autónomo, esto se traduce en la elección de los coeficientes local y global del algoritmo, entendiendo que las soluciones óptimas no se desplazan de manera extrema en el tiempo.

Por otro lado, debe señalarse que las soluciones devueltas por el algoritmo son fácilmente controladas por los parámetros de la función de coste y del optimizador, lo que permite modificar el comportamiento del vehículo con apenas unas cuantas modificaciones numéricas al algoritmo de control. Esto permite una muy sencilla implementación de parámetros dinámicos determinados a través de sistemas complementarios como pueden ser redes neuronales o sistemas borrosos, que analicen la situación del vehículo y las referencias futuras y modifiquen el comportamiento del vehículo para adaptarse a posibles problemas que se detecten.

Por último, puede afirmarse que el tiempo de ejecución del algoritmo tiene carácter determinista, ya que al no haberse implementado un criterio de terminación basado en la variación relativa entre iteraciones de la secuencia de control propuesta, el optimizador siempre se ejecuta de forma completa. Esto hace que pueda tenerse en cuenta un periodo de tiempo fijo en la designación temporal del controlador dedicado a la optimización de la función de coste. No obstante, como se verá en las próximas líneas, esta ventaja pierde significado cuando se analiza el tiempo de ejecución por sí mismo.

La principal desventaja de la implementación del algoritmo PSO en el controlador MPC es precisamente esa, su tiempo de ejecución. Con un periodo de ejecución medio de 700 milisegundos, este controlador es exageradamente lento cuando se compara contra uno con una configuración idéntica pero con un optimizador basado en el algoritmo COBYLA, cuyo tiempo de ejecución, a pesar de tener mucha variación con respecto a la topología de la función de coste, no excede en ningún caso los 20 milisegundos.

Es posible que esta desventaja se deba a la implementación realizada y a su eficiencia computacional; es conocido que el lenguaje Python no destaca por su velocidad de ejecución, pero dado que el optimizador basado en COBYLA se encuentra escrito en este lenguaje, debe concluirse la lentitud del PSO implementado no es una cuestión sistémica. Es mucho más probable que la implementación realizada se muestre como una opción pobre en términos de computación por falta de experiencia en la programación de este tipo de algoritmos, pero la realidad es que la velocidad de ejecución no es un punto fuerte de los algoritmos metaheurísticos.

A pesar de que en la literatura no se hace mención a ello, una de las conclusiones más importantes que pueden extraerse de este trabajo de fin de grado es que los algoritmos metaheurísticos no están diseñados para ser una alternativa viable cuando se implementan directamente en un problema de optimización *online*, es decir, que tiene que ejecutarse de forma repetitiva y secuencial dentro de otro proceso. Se conciben como herramientas muy precisas, muy adaptables a los problemas que pretenden ser solucionados, pero únicamente aptas para problemas que cumplen una de estas dos condiciones: únicamente deben resolverse una vez o no tienen constricciones temporales severas que determinen su comportamiento de forma fundamental.

Otra desventaja fundamental de la implementación propuesta es que no dispone de una forma sencilla de incluir restricciones dentro del proceso de optimización. Es cierto que pueden incluirse como términos de la función de coste como se ha realizado para la derivada de la orientación, pero esta alternativa puede no ser viable cuando las restricciones deban ser de obligado cumplimiento, independientemente de los demás parámetros que quieran considerarse en el control del vehículo. Un ejemplo claro es el límite de velocidad de 40 kilómetros por hora en las ciudades: no puede permitirse que el controlador pueda decidir por sí mismo superar esa velocidad si lo juzga necesario para converger cuanto antes al centro del carril.

Estas conclusiones arrojan un resultado claro sobre la propuesta realizada en el resumen de este trabajo: no ha conseguido mejorarse la velocidad de ejecución de un controlador MPC dedicado a conducción autónoma mediante la implementación de un algoritmo PSO en su etapa de optimización. Quizá pueda interpretarse como

una consecuencia desalentadora tras el trabajo realizado para llegar hasta ella, pero pueden obtenerse ciertos puntos clave de la experiencia generada con este desarrollo.

El hecho de que la implementación *online* no sea viable no quiere decir que no haya posibilidades de que los algoritmos metaheurísticos puedan revolucionar el control MPC.

Una posible estrategia de implementación podría ser la de emplear el método de simulación descrito en este trabajo para generar una base de datos extensa en la que se registren cientos de miles de situaciones de conducción diferentes, con el objetivo de generar una suerte de *look-up table* a través de la que extrapolar soluciones de control mediante análisis de información obtenida por sensores, visión artificial y sistemas borrosos.

Por otro lado, es posible debido a la existencia del No Free-Lunch Theorem que la elección de PSO como optimizador para este problema concreto haya sido desafortunada. Puede ser, incluso, que no sea recomendable como optimizador para ningún problema en el que se emplee un controlador MPC. Estas afirmaciones requieren de un estudio profundo en el que se empleen otros algoritmos metaheurísticos diferentes y aplicados a problemas de otros ámbitos.

5.2. Trabajos futuros

Dentro de las posibles líneas de investigación que pueden surgir a partir de lo expuesto en este documento hay cuatro que merecen una especial atención: la adición de restricciones al problema de optimización del controlador MPC, la vectorización y paralelización del algoritmo PSO, la aplicación de una solución similar a problemas de control con un tiempo de muestreo ampliamente superior y la exploración de la extensa biblioteca de algoritmos metaheurísticos desarrollados en las últimas décadas.

En primer lugar, la adición de restricciones al proceso de optimización aumentaría de forma significativa las perspectivas de aplicación práctica de la solución de control expuesta. Tomando como ejemplo lo discutido en la sección anterior, la inclusión del algoritmo PSO en el *workframe* del control MPC estudiada a lo largo de este documento no permite la adición de condiciones de contorno al problema de optimización; en el caso de la conducción autónoma, esto se traduce en la imposibilidad de incluir restricciones como obstáculos en la carretera o limitaciones duras en la velocidad de giro del volante. En otras aplicaciones más clásicas del control MPC, como en los procesos de mezclado de la industria cementera, puede suponer una total incertidumbre en parámetros tales como el ratio de mezcla o la composición del material.

En la implementación presentada pueden incluirse restricciones de frontera, pero no aquellas que, a pesar de ser posibles por no llevar asociadas una saturación de las variables de control. Estas restricciones, que son íntegras al proceso de control, habilitarían la aplicación real de una aproximación diferente a la minimización de la función de error del controlador, lo que podría resultar en acciones de control que exijan menos esfuerzo por parte de los actuadores o que proporcionen una solución más refinada que las obtenidas del empleo de optimizadores tradicionales.

Las dos siguientes propuestas tratan la principal desventaja extraída de la subsección anterior. Por un lado, la vectorización y paralelización del algoritmo PSO y su posterior ejecución en GPU permitiría una reducción exponencial del tiempo de cómputo asociado a la evaluación del mismo, decenas de veces inferior [47]. Además, los desarrollos tecnológicos de los últimos años y la proliferación de plataformas de cálculo especializadas en manipulación matricial, como las Tensor Processing Unit (TPU) [22] podrían mejorar aún más el tiempo de ejecución, si la vectorización permite una ejecución asimilable al cálculo matricial y el tamaño de población es lo suficientemente grande. Si se consigue un beneficio lo suficientemente grande con una mejor optimización del código y el empleo de hardware de la potencia suficiente, es probable que pueda alcanzarse un modelo de controlador PSO que permita un empleo *online*, en tiempo real.

De forma paralela a la adaptación del algoritmo a plataformas de cómputo alternativas, puede estudiarse la aplicación de la estructura de control propuesta a procesos que requieran de unas prestaciones temporales sensiblemente inferiores a las que exige la conducción autónoma, como pueden los presentes en la industria alimenticia o en reactores petroleros. De esta forma las constricciones temporales pierden importancia frente al refinamiento de la solución, habilitando el controlador PSO como una solución viable en procesos como los mencionados.

Por último, teniendo en cuenta el No Free-Lunch Theorem (NFLT) [51] [52], es de esperar que, dentro de la extensa proliferación de algoritmos metaheurísticos que la literatura científica pueda encontrarse uno con un desempeño mejor que el Particle Swarm Optimization en la minimización de funciones topológicamente asimilables a las empleadas como funciones de error en los controladores MPC. Puede ocurrir, incluso, que la elección de técnica metaheuristica dependa por completo del proceso de control para el que se proponga la solución. Una exploración dedicada de este fenómeno podría llevar a la creación de un protocolo de elección sistemática de algoritmos en función de las características del problema original o, al menos, a la acumulación de experiencia que refleje qué técnicas se han mostrado útiles para según que problemas.

A. Anexo: presupuesto de desarrollo

Las tablas A.1, A.2, A.3 y A.4 recogen los costes calculados del trabajo realizado durante el proceso de este Trabajo de Fin de Grado.

A.1. Equipamiento empleado

Equipo	Precio (€)	Amortización	Tiempo de uso	Coste (€)
Ordenador personal HP Pavilion Laptop 15-cx0006ns	955	4 años	6 meses	119,38
Procesador Intel Core i7-8700	220	4 años	2 meses	9,16
Memoria RAM DDR4 3000 MHz 16 GB	60	4 años	2 meses	2,50
Tarjeta gráfica nVidia TITAN RTX	2.750	4 años	2 meses	114,58
Monitor LG 24"	120	4 años	2 meses	5,00
Monitor Samsung 22"	90	6 años	6 meses	7,50
Combo ratón y teclado Logitech	18	4 años	2 meses	0,75
			TOTAL	258,87

Tabla A.1: Costes del equipo empleado en el desarrollo del trabajo.

A.2. Software y programas informáticos

Item	Precio (€)	Tiempo de uso	Coste (€)
Ubuntu 18.04 - 20.04	No aplica	6 meses	0
Python 3.8	No aplica	6 meses	0
PyCharm - Community Edition	Gratuito	6 meses	0
Overleaf	Gratuito	6 meses	0
Sublime Text 3 - Trial	Gratuito	6 meses	0
Plataforma MPC - INVETT	Propiedad UAH	6 meses	0
Simulador CARLA 0.9.8	Gratuito	6 meses	0
		TOTAL	0

Tabla A.2: Costes del software empleado en el desarrollo del trabajo.

A.3. Mano de obra

Concepto	Precio (€/h)	Horas	Coste (€)
Investigación	21,33	150	3.199,50
Implementación	21,33	150	3.199,50
Redacción de proyecto	12,00	100	1.200,00
		TOTAL	7.599,00

Tabla A.3: Costes de personal en el desarrollo del trabajo.

A.4. Costes totales

Concepto	**Importe (€)**
Equipo	258,87
Software	0,00
Mano de obra	7.599,00
TOTAL	7.857,87

Tabla A.4: Coste total del trabajo.

Bibliografía

[1] Marjan Abdechiri, Mohammad Reza Meybodi y Helena Bahrami. «Gases Brownian motion optimization: an algorithm for optimization (GBMO)». En: *Applied Soft Computing* 13.5 (2013), págs. 2932-2946.

[2] Mohamed Abdel-Basset, Laila Abdel-Fatah y Arun Kumar Sangaiah. «Chapter 10 - Metaheuristic Algorithms: A Comprehensive Review». En: *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*. Ed. por Arun Kumar Sangaiah, Michael Sheng y Zhiyong Zhang. Intelligent Data-Centric Systems. Academic Press, 2018, págs. 185-231. ISBN: 978-0-12-813314-9. DOI: https://doi.org/10.1016/B978-0-12-813314-9.00010-4. URL: https://www.sciencedirect.com/science/article/pii/B9780128133149000104.

[3] Christian Blum y Andrea Roli. «Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison». En: *ACM Comput. Surv.* 35 (ene. de 2001), págs. 268-308. DOI: 10.1145/937503.937505.

[4] EF Camacho. *Model Predictive Control, 2ed*. Ed. por Springer. Springer, 2007.

[5] D. Clarke y R. Scattolini. «Constrained receding-horizon predictive control». En: 1991.

[6] A. Colorni y col. «Distributed Optimization by Ant Colonies». En: 1992.

[7] C. R. Cutler y B. L. Ramaker. «Dynamic matrix control: A computer control algorithm». En: *Joint Automatic Control Conference* 17 (1980), pág. 72. DOI: 10.1109/JACC.1980.4232009.

[8] J. Czyzyk, M.P. Mesnier y J.J. More. «The NEOS Server». En: *IEEE Computational Science and Engineering* 5.3 (1998), págs. 68-75. DOI: 10.1109/99.714603.

[9] Elizabeth D. Dolan. *The NEOS Server 4.0 Administrative Guide*. Technical Memorandum ANL/MCS-TM-250. Mathematics y Computer Science Division, Argonne National Laboratory, 2001.

[10] Alexey Dosovitskiy y col. «CARLA: An Open Urban Driving Simulator». En: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, págs. 1-16.

[11] R.C. Eberhart e Y. Shi. «Comparing inertia weights and constriction factors in particle swarm optimization». En: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*. Vol. 1. 2000, 84-88 vol.1. DOI: 10.1109/CEC.2000.870279.

[12] R.C. Eberhart y Yuhui Shi. «Tracking and optimizing dynamic systems with particle swarms». En: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. Vol. 1. 2001, 94-100 vol. 1. DOI: 10.1109/CEC.2001.934376.

[13] Robert W. Erickson y Dragan Maksimovic. *Fundamentals of Power Electronics*. Springer, 2001. ISBN: 978-0-306-48048-5.

[14] David Fernández Llorca. *Module 4: Neural Control*. Control Inteligente (600038). Universidad de Alcalá - Departamento de Automática, 2020.

[15] Carlos E García, David M Prett y Manfred Morari. «Model predictive control: Theory and practice—A survey». En: *Automatica* 25.3 (1989), págs. 335-348. ISSN: 0005-1098.

[16] Zong Woo Geem, Joong Hoon Kim y Gobichettipalayam Vasudevan Loganathan. «A new heuristic optimization algorithm: harmony search». En: *simulation* 76.2 (2001), págs. 60-68.

[17] Mojtaba Ghasemi y col. «Phasor particle swarm optimization: a simple and efficient variant of PSO». En: *Soft Computing* 23.19 (2019), págs. 9701-9718. DOI: 10.1007/s00500-018-3536-8. URL: https://doi.org/10.1007/s00500-018-3536-8.

[18] Fred Glover y Manuel Laguna. *Tabu search I*. Vol. 1. Ene. de 1999. ISBN: 978-0-7923-9965-0. DOI: 10.1287/ijoc.1.3.190.

[19] William Gropp y Jorge J. Moré. «Optimization Environments and the NEOS Server». En: *Approximation Theory and Optimization*. Ed. por Martin D. Buhman y Arieh Iserles. Cambridge University Press, 1997, págs. 167-182.

[20] Pierre Hansen, Nenad Mladenović y JoséA. Moreno Pérez. «Variable neighbourhood search: methods and applications». En: *Annals of Operations Research* 175.1 (2010), págs. 367-407.

[21] John H. Holland. «Genetic Algorithms and Adaptation». En: *Adaptive Control of Ill-Defined Systems*. Ed. por Oliver G. Selfridge, Edwina L. Rissland y Michael A. Arbib. Boston, MA: Springer US, 1984, págs. 317-333. ISBN: 978-1-4684-8941-5. DOI: 10.1007/978-1-4684-8941-5_21. URL: https://doi.org/10.1007/978-1-4684-8941-5_21.

[22] Norman Jouppi y col. «Motivation for and Evaluation of the First Tensor Processing Unit». En: *IEEE Micro* 38.3 (2018), págs. 10-19. DOI: 10.1109/MM.2018.032271057.

[23] Ali Husseinzadeh Kashan. «League championship algorithm: a new algorithm for numerical function optimization». En: *2009 international conference of soft computing and pattern recognition*. IEEE. 2009, págs. 43-48.

[24] J. Kennedy y R. Eberhart. «Particle swarm optimization». En: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942-1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[25] Dongchan Kim, Hayoung Kim y Kunsoo Huh. «Local trajectory planning and control for autonomous vehicles using the adaptive potential field». En: *2017 IEEE Conference on Control Technology and Applications (CCTA)*. 2017, págs. 987-993. DOI: 10.1109/CCTA.2017.8062588.

[26] Scott Kirkpatrick, C. Gelatt y M. Vecchi. «Optimization by Simulated Annealing». En: *Science (New York, N.Y.)* 220 (jun. de 1983), págs. 671-80. DOI: 10.1126/science.220.4598.671.

[27] Štefan Kozák. «State-of-the-art in control engineering». En: *Journal of Electrical Systems and Information Technology* 1.1 (2014), págs. 1-9. DOI: https://doi.org/10.1016/j.jesit.2014.03.002. URL: https://www.sciencedirect.com/science/article/pii/S2314717214000038.

[28] Albert YS Lam y Victor OK Li. «Chemical reaction optimization: a tutorial». En: *Memetic Computing* 4.1 (2012), págs. 3-17.

[29] Hongbo Liu, Ajith Abraham y Crina Grosan. «A novel Variable Neighborhood Particle Swarm Optimization for multi-objective Flexible Job-Shop Scheduling Problems». En: *2007 2nd International Conference on Digital Information Management*. Vol. 1. 2007, págs. 138-145. DOI: 10.1109/ICDIM.2007.4444214.

[30] Helena Lourenço, Olivier Martin y Thomas Stützle. «Iterated Local Search». En: ene. de 2003, págs. 321-353. ISBN: 978-1-4020-7263-5. DOI: 10.1007/0-306-48056-5_11.

[31] Bonifacio Martín-del-Brío y Alfredo Sanz. *Redes neuronales y sistemas borrosos*. Ene. de 2006. ISBN: 978-84-7897-743-7.

[32] D.Q. Mayne y col. «Constrained model predictive control: Stability and optimality». En: *Automatica* 36.6 (2000), págs. 789-814. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(99)00214-9. URL: https://www.sciencedirect.com/science/article/pii/S0005109899002149.

[33] Seyedali Mirjalili. «SCA: a sine cosine algorithm for solving optimization problems». En: *Knowledge-based systems* 96 (2016), págs. 120-133.

[34] Seyedali Mirjalili y Andrew Lewis. «The whale optimization algorithm». En: *Advances in engineering software* 95 (2016), págs. 51-67.

[35] Román Anselmo Mora-Gutiérrez, Javier Ramírez-Rodríguez y Eric Alfredo Rincón-García. «An optimization algorithm inspired by musical composition». En: *Artificial Intelligence Review* 41.3 (2014), págs. 301-315.

[36] M. Morari. «Model Predictive Control: Multivariable Control Technique of Choice in the 1990s?» En: 1993.

[37] E. Mosca, J.M. Lemos y J. Zhang. «Stabilizing I/O receding horizon control». En: *29th IEEE Conference on Decision and Control*. 1990, 2518-2523 vol.4. DOI: 10.1109/CDC.1990.203454.

[38] Riccardo Poli, James Kennedy y Tim Blackwell. «Particle swarm optimization». En: *Swarm Intelligence* 1.1 (2007), págs. 33-57. DOI: 10.1007/s11721-007-0002-0. URL: https://doi.org/10.1007/s11721-007-0002-0.

[39] A. I. Propoi. «Application of linear programming methods for the synthesis of automatic sampled-data systems». En: *Avtomat. i Telemekh.* 24.7 (1962), págs. 912-920.

[40] Esmat Rashedi, Hossein Nezamabadi-Pour y Saeid Saryazdi. «GSA: a gravitational search algorithm». En: *Information sciences* 179.13 (2009), págs. 2232-2248.

[41] J. Richalet y col. «Model predictive heuristic control: Applications to industrial processes». En: *Automatica* 14.5 (1978), págs. 413-428. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(78)90001-8. URL: https://www.sciencedirect.com/science/article/pii/0005109878900018.

[42] Douglas A. Lawrence Robert L. Williams II. *Linear State-Space Control Systems*. John Wiley & Sons, Ltd, 2007. ISBN: 9780470117873. DOI: https://doi.org/10.1002/9780470117873.

[43] Stefano Saetta y Lorenzo Tiacci. «A new methodology for applying simulation driven metaheuristics to the balancing of security inspection lines». En: ene. de 2006, 6 pp.-. ISBN: 0-7803-9519-0. DOI: 10.1109/WSC.2005.1574281.

[44] Sameh A Salem. «BOA: A novel optimization algorithm». En: *2012 International Conference on Engineering and Technology (ICET)*. IEEE. 2012, págs. 1-5.

[45] Ryan Solgi, Omid Bozorg-Haddad y Hugo Loaiciga. *Meta-heuristic and Evolutionary Algorithms for Engineering Optimization*. Nov. de 2017. ISBN: 9781119387053. DOI: 10.1002/9781119387053.

[46] Éric D Taillard y Stefan Voss. «POPMUSIC—Partial optimization metaheuristic under special intensification conditions». En: *Essays and surveys in metaheuristics*. Springer, 2002, págs. 613-629.

[47] Ying Tan y You Zhou. «Parallel Particle Swarm Optimization Algorithm Based on Graphic Processing Units». En: *Handbook of Swarm Intelligence: Concepts, Principles and Applications*. Ed. por Bijaya Ketan Panigrahi, Yuhui Shi y Meng-Hiot Lim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, págs. 133-154. ISBN: 978-3-642-17390-5. DOI: 10.1007/978-3-642-17390-5_6. URL: https://doi.org/10.1007/978-3-642-17390-5_6.

[48] Berna Haktanirlar Ulutas y Sadan Kulturel-Konak. «A review of clonal selection algorithm and its applications». En: *Artificial Intelligence Review* 36.2 (2011), págs. 117-138.

[49] Chris Voudouris y Edward Tsang. «Function optimization using guided local search». En: *University of Essex, Technical Report CSM-249, Colchester, UK* (1995).

[50] Hengyang Wang y col. «Path Tracking Control for Autonomous Vehicles Based on an Improved MPC». En: *IEEE Access* 7 (2019), págs. 161064-161073. DOI: 10.1109/ACCESS.2019.2944894.

[51] D. H. Wolpert y W. G. Macready. «No free lunch theorems for optimization». En: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), págs. 67-82. DOI: 10.1109/4235.585893.

[52] David H. Wolpert. *What the No Free Lunch Theorems Really Mean; How to Improve Search Algorithms*. 2012.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá