eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO DE FIN DE MÁSTER

IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL PREDICTIVO INTELIGENTE PARA COMPLEJOS PROCESOS MULTIVARIABLES

Alumno: Zabaljauregi, Lekerika, Asier
Director/Directora (1): Irigoyen, Gordo, Eloy

Curso: 2021-2022

Fecha: Bilbao, 18 de septiembre, 2022

Resumen Laburpena Abstract

Ante las crecientes exigencias de fabricación, la necesidad de operar bajo especificaciones de rendimiento estrictas y el advenimiento de sistemas innovadores de dinámicas acopladas y no lineales, crece el interés por desarrollar estrategias de control que hagan frente a dichos requerimientos.

En el trabajo a continuación presenta el estudio y la implementación de una compleja estrategia de control predictivo inteligente para sistemas multivariables no lineales (iMO-NMPC), sobre plataformas de simulación y de tiempo real, mediante *MATLAB®* y el API de *C MEX S-Functions* de *Simulink®*.

Palabras Clave: control predictivo no lineal, iMO-NMPC, MATLAB/Simulink, *S-Function*.

Fabrikazio-eskakizun gero eta handiagoen aurrean, espezifikazio zorrotzpean jarduteko beharren eta dinamika akoplatutako sistema berritzaile ez-linealen aurrean, aipatutako horiei aurpegi emango dien kontrol-estrategiak garatzeko interesa areagotzen da.

Ondorengo lanak, aldagai anitzeko sistema ez-linealentzako iragarpen kontrol-estrategia adimendun baten (iMO-NMPC) azterketa eta inplementazioa aurkezten ditu. Estrategiaren inplementazioa simulazio eta denbora errealeko plataformen gainean egingo da, *MATLAB®* eta *Simulink®-en C MEX S-Function* delako API bidez.

Gako-hitzak:iragarpen kontrol-estrategia adimendun ez-lineala , iMO-NMPC, MATLAB/-Simulink, *S-Function*.

With the increasing demands of manufacturing, the need to operate under strict performance specifications and the advent of innovative coupled and nonlinear dynamics systems, there is a growing interest in developing control strategies to meet these requirements.

The following work presents the study and implementation of a complex intelligent predictive control strategy for multivariable nonlinear systems (iMO-NMPC), on simulation and real-time platforms, using *MATLAB®* and the *C MEX S-Functions* API of *Simulink®*.

Keywords: nonlinear model predictive control, iMO-NMPC, MATLAB/Simulink, *S-Function*.

Índice

Lista de figuras

Lista de tablas

Lista de scripts

Lista de acrónimos

EIB Escuela de Ingeniería de Bilbao

GICI Grupo de Investigación de Control Inteligente

TFM Trabajo de Fin de Máster

MIMO Multiple Input Multiple Output

SISO Single Input Single Output

FLC Fuzzy Logic Control

iMO-NMPC Intelligent Multi-objective Nonlinear Model Predictive Control

HiL Hardware in the Loop

RT/TR Real Time/ Tiempo Real

NN Neural Network

NARX Nonlinear AutoRegressive with eXogenous input

ES Evolutionary Strategy

EA/MOEA (Multiobjective) Evolutionary Algorithm

GA/AG Genetic Algorithm/Algoritmo Genético

PSO Particle Swarm Optimization

NSGA Non-dominated Sorting Genetic Algorithm

MPC Model Predictive Control

DMC Dynamic Matrix Control

1. Introducción

El comportamiento no lineal aparece en muchos problemas de ingeniería. Algunos de los principales campos en los que aparece son: la ingeniería mecánica, en la que algunos ejemplos notorios son la rigidez y el amortiguamiento de las estructuras y las frecuencias de resonancia que no se exhiben de forma lineal a lo largo del régimen de funcionamiento; la robótica, donde los brazos y estructuras robóticas están sometidos a grandes no linealidades debido a que sus ecuaciones de movimiento están representadas por leyes trigonométricas y debido a las variaciones de la carga transportada entre otros motivos; los robots aéreos, como los cuadricópteros y otros robots con diferentes configuraciones de vuelo, en los que las fuerzas de empuje están fuertemente acopladas y relacionadas de forma no lineal con la orientación; finalmente, los procesos químicos, donde los pasos individuales de un proceso de fabricación complejo, como las columnas de destilación, que por sí mismas tienen un comportamiento dinámico no lineal, por no hablar de la combinación de varias unidades que componen toda una fabricación. Además, muchos sistemas biológicos, ya sean sistemas sensoriales (vista, oído, tacto), sistemas depredador-presa o ecuaciones de crecimiento bacteriano, no siguen los principios de superposición.

En definitiva, las propiedades que un sistema no lineal puede reunir, son las siguientes: incumplimiento del principio de superposición (linealidad y homogeneidad); múltiples puntos de equilibrio aislados; presencia de propiedades como ciclo límite, bifurcación y comportamiento caótico. Es más, las soluciones de los sistemas no lineales pueden no existir para todos los instantes de tiempo. Por tanto, la modelización de estos sistemas con modelos matemáticos plantea importantes retos, ya que las ecuaciones de movimiento no lineales no suelen tener una solución analítica, y en muchas ocasiones el modelo del sistema puede estar incompleto debido a las dificultades para modelizar ciertos fenómenos como: la fricción no lineal, la incertidumbre paramétrica o señales no medibles, necesarias para completar el modelo.

Aunque muchos procesos manifiesten comportamiento no lineal, en la práctica debido a que se opera en la vecindad del estado estacionario, una linearización del modelo es suficiente para representarlo. Sin embargo, existen algunos casos importantes en los que la no-linealidad es tan severa que afecta a la dinámica de lazo cerrado y por lo tanto un modelo lineal no es suficiente. Adicionalmente, en aquellos sistemas en las que se trabaja en lotes o en una operación transitoria una ley de control lineal puede no resultar suficientemente satisfactoria.

Otro desafío estrechamente ligado al control no lineal es el ámbito de la optimización y control multiobjetivo. La optimización multiobjetivo se ha aplicado en muchos campos de la ciencia, como la ingeniería, la economía y la logística, donde se requiere tomar decisiones óptimas en presencia de compromisos entre dos o más objetivos en conflicto. Para un problema de optimización multiobjetivo no trivial no existe una solución única que optimice simultáneamente cada objetivo; existen varias soluciones que no son mejores entre sí, es decir, son soluciones dominantes en el espacio objetivo. En este caso en el campo de la ingeniería de control, se hace uso de un agente experto o *decision maker* para seleccionar una solución que

basada en el conocimiento y experiencia de la planta se considera apropiada para la evolución del sistema. Por tanto, el reto y la oportunidad que presentan la modelización y control de este tipo de sistemas ofrecen un ámbito de estudio digno e importante.

En este trabajo se expone el estudio de una estrategia inteligente de control predictivo y procedimiento seguido para implementarlo en plataformas de simulación y ejecución en tiempo real. Primero se expone el **contexto** describiendo la problemática a abordar seguido de los **objetivos y el alcance** del trabajo; se exhiben a continuación los posibles **beneficios técnicos y económicos**. Engrosando la problemática, se presenta un **estado del arte** abordando los apartados de control predictivo, optimización y modelado de sistemas referente a la estrategia de control que se pretende implementar. Sucesivamente, la **descripción de la solución** incluye el trabajo realizado cronológicamente. Posteriormente se incluye una **planificación** junto a su **presupuesto** inicial para y su seguimiento. Finalmente se exponen las **conclusiones y las líneas futuras** de investigación a seguir. Adicionalmente se incluyen los apartados presupuestarios, de planificación y anexos del código desarrollado como parte de un trabajo de investigación en ingeniería.

2. Contexto

En la línea de investigación del Grupo de Investigación de Control Inteligente (GICI) de la EIB ya se cuenta con un desarrollo de un algoritmo que conjuga una estrategia MPC (*Model Predictive Control*/Control Predictivo basado en Modelos) no lineal con un optimizador de algoritmos genéticos, *fuzzy decision maker* (tomador de decisiones basado en lógica difusa) y una red neuronal NARX para modelar el sistema real.[1] La estrategia iMO-NMPC (*intelligent Multi-Objective Nonlinear Model Predictive Control*) proviene de una línea de investigación incluyendo los artículos [37],[33],[32] [67] consolidándose en [68] y extendiéndose en [38], [24], [18], [34]. Para estudiar el comportamiento de la estrategia y el efecto de diferentes parámetros en la misma, es necesario contar con una plataforma de simulación y posteriormente ejecución en tiempo real. En el trabajo presente se muestra el esfuerzo para alcanzar dicho objetivo. Asimismo, se implementa por primera vez el paso de SISO a MIMO, lo cual es totalmente novedoso en la evolución, estudio y mejora del iMO-NMPC. Adicionalmente, para seguir investigando esta estrategia de control y permitir una fácil intercambiabilidad de sus partes estructurales, surge la necesidad de desarrollar un programa que actúe como interfaz con el entorno de simulación y permita transiciones fluidas en los cambios del optimizador, modelo de predicción o *decision maker*/agente experto.

Cada una de las técnicas que componen la compleja estrategia de control propuesta, debido a su naturaleza inherente, aportan diversas ventajas que permiten que el sistema en su conjunto sea flexible, óptimo y robusto; el MPC tiene un gran rendimiento en sistemas dinámicos tanto simples como complejos, así como en sistemas con tiempos de retardo largos o de fase no mínima, ya que cuenta intrínsecamente con compensación de los tiempos de retardo y de las perturbaciones medibles. Además, la adición de restricciones en la optimización es conceptualmente sencilla [4]. En el campo de la optimización, técnicas como AG (Algoritmos Genéticos) o PSO (*Particle Swarm Optimization*) proporcionan un método metaheurístico para una búsqueda global en espacios de objetivos complejos de alta dimensión. Las modificaciones sobre los algoritmos genéticos básicos, como el NSGA-II [14], no sólo mejoran la complejidad computacional y tiempo, sino que también introducen un incentivo innato para preservar la diversidad en las soluciones y promover la convergencia.

En el tema del modelado de sistemas, la necesidad de identificar sistemas no lineales va más allá del campo de aplicación del control. Ya que la creciente demanda de un mayor rendimiento y eficiencia, empuja a los sistemas a un régimen de funcionamiento no lineal, por lo que se requieren modelos no lineales para su diseño y control. En estas aplicaciones, no siempre es necesaria una alta precisión, ya que los resultados cualitativos pueden ser muy útiles para aislar los términos dominantes. Así, los errores estructurales del modelo (es decir las deficiencias en el método de descripción del modelo elegido) son más importantes que las perturbaciones

[1] Como nota aclaratoria, "estrategia" se refiere al conjunto de enfoques/técnicas y sus posibles combinaciones empleadas para abordar el problema de control. En cambio, una instancia de implementación de la estrategia, es ya un algoritmo que a su vez contendrá sub-algoritmos que se encarguen de la lógica MPC, optimización, etc.

de ruido; y por ello es de mayor peso que los métodos de identificación y representación del modelo estén bien ajustadas para captar el correcto comportamiento del mismo.

Si por otro lado, como ya se ha mencionado, la desviación de la linealidad no es tan alta, se puede realizar una aproximación de la planta y emplear un modelo linearizado alrededor de varios puntos de operación donde el error cometido es despreciable y alternar de un punto de operación al otro según convenga. En cambio para no linealidades más severas existen diferentes enfoques que hacen uso de modelos *Wiener*, redes neuronales, modelos *Volterra*, modelos *Hammerstein*, modelos NARX, modelos *fuzzy*, etc. [4]. En la técnica iMO-NMPC propuesta se hace uso de una red neuronal NARX, debido a la gran flexibilidad, comportamiento conservativo y facilidad de uso que dan las redes neuronales dinámicas.

Concretando, en la necesidad de un mecanismo de prototipado y ejecución para poner a prueba y realizar modificaciones sobre la compleja estrategia de control se ve idóneo el empleo de una plataforma de simulación como Simulink. En este trabajo de fin de máster se pretende estudiar cada componente integral de una evolución de la estrategia de control propuesta y realizar una implementación de la misma sobre la plataforma de simulación MATLAB/Simulink ®.

3. Objetivos y alcance

El objetivo final del presente TFM es obtener un código en C de una evolución de la compleja estrategia de control iMO-NMPC que permita realizar simulaciones eficientes sobre la plataforma Simulink ®, así como ser compilable para su ejecución sobre plataformas de tiempo real y permitir una generación de código. Primero, para tener una buena comprensión de la estrategia y de las herramientas a emplear, es necesario estudiar el funcionamiento de los diferentes componentes que componen la estrategia de control y los medios que se emplearán para su desarrollo. Después, una vez familiar con los medios, se continua con el desarrollo del código. Finalmente, se valida la implementación con pruebas. Así se divide el objetivo principal en los siguientes sub-objetivos.

3.1. Estudio del estado del arte y de la estrategia iMO-NMPC mediante scripts de MATLAB ®

Este sub-objetivo trata de conseguir el conocimiento necesario de la estrategia en general, así como los diferentes componentes. Tal y como se ha mencionado anteriormente, el iMO-NMPC (*Intelligent Multi-objetive Nonlinear Model Predictive Control*) tiene una base de control predictivo al cual se añaden metodologías de *soft-computing* para hacer frente a complejos problemas multivariables con objetivos de control contrapuestos. Este primer paso trata de entender las redes neuronales empleadas como modelo de predicción, el algoritmo genético como optimizador y el papel del *Decision Maker*.

Este primer objetivo parcial contiene el análisis del estado del arte para alcanzar una visión general del ámbito de control inteligente, arrojar luz a los problemas de control industrial y conocer las posibles opciones de mejora y alternativas en el campo del control inteligente.

Como parte de una línea de investigación, el trabajo a desarrollar no parte de una base vacía; Juanjo Valera y el grupo de investigación de control inteligente de UPV/EHU cuenta con *scripts* de prototipado de anteriores desarrollos. Estos *scripts* están escritos en el lenguaje de programación de MATLAB, son interpretados en su entorno de edición y no tienen fácil conexión con Simulink y la plenitud de sus funcionalidades ofrecidas. Como medio de aprendizaje se emplearán dichos *scripts* como referencia para entender el mecanismo de entrenamiento de las redes neuronales, la inclusión de un optimizador genético multi-objetivo, la función de evaluación y finalmente para obtener el conocimiento de cómo encaja cada parte.

3.2. Aprendizaje y análisis del API *S-Function* para Simulink ®

En este apartado se analizarán los diferentes enfoques que existen para introducir el código de la estrategia de control en una plataforma de simulación como Simulink. Matlab/Simulink

ya cuenta con un API (*Application Programming Interface*) llamadas S-Functions, o funciones especiales que permiten introducir rutinas de ejecución programadas por el usuario en el ciclo de simulación. Por lo tanto, este apartado se centra en aprender y realizar un previo desarrollo de acercamiento más simple, antes de abordar la estrategia completa.

3.3. Implementación de la estrategia de control inteligente sobre plataformas de simulación

Una vez que se tiene un buen entendimiento de las *S-Functions* se procederá a desarrollar iterativamente las diferentes funcionalidades de la estrategia de control predictivo no lineal multi-objetivo. Para ello, primero se actualizará el código del algoritmo genético propuesto (NSGA-II), a la última versión publicada por el autor K.Deb. Subsiguientemente se integrará en el esqueleto del S-Function y se desarrollarán los archivos correspondientes al modelo de predicción y el *Decision Maker*.

3.4. Estudio de alternativas para la ejecución en tiempo real

Un paso necesario para la madurez de la estrategia y su transición de ámbitos académicos a industriales es garantizar su ejecución en tiempo real. Igualmente, resulta también conveniente trabajar con MATLAB, ya que cuenta con herramientas automatizadas de compilación y generación de código para plataformas RT. Este último objetivo parcial tratará con el análisis de dichas vías y de la adecuación del código en C para permitir su compilación y ejecución en tiempo real sin errores.

4. Beneficios

El trabajo de implementación realizado actúa como base para futuros desarrollos en la línea de investigación; permitiendo realizar simulaciones para comprobar la eficiencia del algoritmo, así como generar código y ejecutables para plataformas de tiempo real como pueden ser los target *Speedgoat* o los PLC. No solo eso, sino que el análisis del estado del arte realizado ayuda a contemplar nuevas alternativas de desarrollo. Aunque existe un hueco más amplio que el habitual entre la investigación de técnicas de control predictivo en el ámbito académico y su implementación en la industria, debido a la naturaleza conservatoria de la industria de proceso y la reticencia ante lo nuevo, se esta presenciando una rápida adopción de estrategias novedosas.

Hoy en día las expectativas de la industria para el control predictivo han pasado de exigir un control óptimo para sistemas multivariables a hacer eso mismo pero con mínimos esfuerzos de configuración y mantenimiento. En la industria de proceso actual el control predictivo es imprescindible. No obstante, al mismo tiempo, los recursos de profesionales expertos para la puesta en marcha, supervisión y mantenimiento son cada vez más limitados. Por esta razón tanto los proveedores como los clientes buscan formas de mantener el rendimiento de los MPC con la mínima intervención manual.

4.1. Beneficios Técnicos

En la encuesta realizada por M.Forbes, R.S.Patwardhan, H.Hamadah y R. Bhushan Gopaluni en 2015 [23], se mencionan dos problemas para la puesta en marcha de plantas con MPC las cuales la línea de investigación de control inteligente podría aliviar.

El primero es el modelado del proceso; el éxito del control predictivo recae totalmente en un buen modelo de proceso. El modelo de la planta se construye a partir de los datos recogidos de un experimento de excitación, donde generalmente para obtener una buena caracterización, se lleva a la planta fuera de su rango normal de operación, frecuentemente resultando en un comportamiento no-lineal. En esta primera identificación normalmente no se da una disparidad entre modelo y planta ya que la puesta en marcha lo realiza un experto. No obstante, con el paso del tiempo y el desgaste de la planta este desajuste es inevitable, y lo tiene que realizar el operario de mantenimiento. Los paquetes software de identificación de proceso comerciales muchas veces sufren de una interfaz confusa con un flujo de trabajo arduo; esta carencia de claridad a menudo da lugar a errores en la configuración. De la misma manera, el software puede ofrecer mucha flexibilidad a cambio de complejidad; compromiso que en manos del departamento de operaciones, cuya responsabilidad y conocimiento se limita a operar el MPC, puede generar fallos.

El segundo problema es el de la no linealidad a lo largo de los puntos de operación. Como práctica usual en la industria, se emplea la planificación de ganancias, donde las operaciones

del proceso se dividen en un conjunto de regiones operativas basadas en los valores de una o más variables clave del proceso. Para cada región se identifican los parámetros del modelo y se ajusta el controlador adecuadamente. A medida que las operaciones del proceso cambian, desplazando el proceso de una región a otra, el MPC se actualiza para utilizar los parámetros adecuados para la nueva región.

Esta práctica es conservadora, ya que gran parte de la industria de procesos modelan sus plantas por medio de un POTM (*Primer Orden mas Tiempo Muerto*). De esta manera una región termina de ser caracterizada al haber identificado su ganancia, constante de tiempo y retardo. El empleo de este tipo de modelos está respaldada por años de experiencia e investigación, dotándolo de seguridad y garantías de estabilidad. Sin embargo, a menudo este encajonamiento del proceso en un modelo tan simple puede venir a cambio de carencias de rendimiento, al no poder el modelo representar precisamente la planta. En estos casos, si se precisa, el experto empleará un modelo de orden superior. Así, finalmente para configurar el control por planificación de ganancias habrá que obtener los parámetros del modelo para cada una de las regiones; labor que tendrá que repetir el operario en su mantenimiento.

La línea de investigación que engloba a la estrategia de control iMO-NMPC viene a estudiar la validez de las redes neuronales y un optimizador por algoritmos genéticos. Dentro de un rango suficientemente grande que encapsule el de operación, se puede asegurar la precisa caracterización del proceso mediante una red NARX. Queda demostrado que las redes NARX son más que capaces para proveer una intuitiva caracterización de modelos no-lineales frente los métodos clásicos [25]. Así pues, el empleo de las redes neuronales alivia los problemas mencionados; para el primer problema se puede argumentar que la tarea de entrenamiento de la red partiendo de los datos reales adquiridos de la planta, es un proceso totalmente automatizado, con un grado de abstracción suficiente para ser accesible para los operarios. Por otro lado, se pueden implementar técnicas de aprendizaje online para que la red se vaya ajustando a medida que se deteriora la instrumentación y el equipamiento de la planta, evitando así la re-identificación. En cuanto a los puntos de operación no-lineales, no haría falta realizar varias regiones de caracterización; habiendo realizado un entrenamiento y selección de red con garantías de cubrir las zonas de operación deseadas, no hay necesidad de compartimentar la región de operación. Se puede razonar que un inconveniente de utilizar una representación para toda la zona de operación es que desposee de la capacidad de modificar específicamente una región, tal y como se puede hacer en la planificación de ganancias cuando interesa aumentar el rendimiento de una zona insatisfactoria. Sin embargo, un contra-argumento es que al emplear una misma caracterización para toda la región no hay transiciones bruscas. Adicionalmente, la estrategia no sólo queda limitada a esta modelización sino que el método empleado para la predicción de la salida puede sustituirse por técnicas mencionadas anteriormente (SINDy, PCA…).

Es importante remarcar que las redes NARX funcionan como modelizado de caja negra; no hace falta tener ningún conocimiento interno del sistema para el modelado. Lo cual es una peculiaridad de importante utilidad, ya que esto permite elaborar la imagen del proceso, aún sin tener acceso a las variables internas. Aunque están registrados varias aplicaciones exitosas de MPC no lineal en la industria, la falta de modelos no lineales de alta fidelidad sigue siendo un desafío para la aplicación generalizada de esta tecnología [58].

Finalmente, queda por subrayar que la estrategia es más efectiva ante sistemas con dinámicas altamente no-lineales, donde se pretende conseguir la mejor secuencia de control para optimizar objetivos frecuentemente contrapuestos. Con el progresivo crecimiento de la capacidad de procesamiento, se abre la oportunidad de emplear modelos más precisos y aumentar el número de iteraciones y una población mayor, para una capacidad de optimización elevada. Como consecuencia, tanto los sistemas industriales, como de otros ámbitos, con las

características mencionadas serán beneficiadas enormemente.

4.2. Beneficios Económicos

Recientemente, ha habido una gran actividad de investigación sobre una idea novedosa, denominada EMPC (*Economic Model Predictive Control*). La idea central de esta técnica, que se menciona posteriormente en el estado del arte, es utilizar una única función objetivo MPC para controlar y optimizar las condiciones económicas [20]. El buen rendimiento de un proceso industrial depende de los aspectos económicos del mismo: rentabilidad, eficiencia, variabilidad, capacidad, sostenibilidad, etc. Como consecuencia de los continuos cambios en la economía del proceso (e.g variación de materias primas, altibajos en el precio de la energía, etc.), los objetivos y estrategias de las operaciones de proceso deben actualizarse con frecuencia para para tener en cuenta estos cambios. Tradicionalmente la optimización de aspectos económicos se ha gestionado por medio de una arquitectura multi-capa jerárquica. En la capa superior de control denominada RTO (*Real Time Optimization*) se definen las funciones de *coste de operación* y *beneficios de operación*. En esta capa de mayor abstracción se genera la consigna para la *capa de supervisión* donde reside el MPC. La capa de supervisión a su vez genera acciones de control para la *capa de regulación* que ya actúa directamente en las variables del proceso. Cada una de estas capas se extiende en escalas de tiempo diferentes; horas a días para el RTO y segundos a minutos para la capa de supervisión. Debido a que optimizar empleando un modelo preciso es importante para rendir satisfactoriamente, en la capa de RTO tradicionalmente se han utilizado sistemas no-lineales más complejos que en la capa de MPC donde es más habitual el empleo de modelos linealizados. Posibles discrepancias entre los modelos de las diferentes capas puede dar lugar a una situación donde el RTO genera consignas inalcanzables por el controlador de retroalimentación, a menudo provocando un desfase entre el estado estacionario del funcionamiento real y el deseado. Aunque el paradigma por jerarquía ha sido exitoso, hoy en día se presenta la creciente necesidad de operaciones dinámicas orientadas al mercado (*Smart manufacturing*) que incluyan una gestión de procesos más ágil y eficiente. Algunos de los objetivos y restricciones dependientes del tiempo, habitualmente contemplados, son el coste de la energía, la carestía o el precio de las materias primas y la demanda y el precio del producto. Para tener en cuenta directamente estas variables temporales, surge la necesidad para integrar estrechamente las capas mencionadas.

El cambio que propone EMPC al incluir aspectos económicos en la función de coste resuelve los problemas de discrepancia y lleva a trabajar con un modelo dinámico en la optimización. Es más, esta nueva función de coste económica puede no ser definida positiva con respecto a una referencia (estado estacionario) en específico. Además, puede suceder que el óptimo de la función de coste económica no se encuentre en un régimen de operación estacionario sino que esté en el estado transitorio, promoviendo controlar la planta dinámicamente. Al mismo hilo, en el artículo en cuestión [20] se mencionan 17 fuentes que respaldan la operación dinámica del proceso tanto en simulación, como experimentalmente, como medio fiable para obtener un rendimiento mejorado.

Para introducir la relación del iMO-NMPC al tema, se considera el ejemplo dado por M.Ellis, H.Durand y P.D.Christofides [20]; una entrada que proporciona calor a un reactor (e.g una camisa de vapor) para el control de la temperatura. Se puede apreciar de la anterior oración, que se da una situación de objetivos contrapuestos. Es necesario mantener una temperatura en específico para llevar a cabo una reacción exitosa, pero en cambio, proporcionar ese calor viene a cambio de un coste energético/económico; ya que una mayor temperatura requerida conlleva un mayor consumo de energía (combustible) para generar vapor, y esto va vinculado a un gasto. El optimizador de AG y el modelo de predicción NARX son adecuados para abordar

múltiples objetivos contrapuestos y un modelo capaz de representar la dinámica de la planta real precisamente. Es por ello que este primer paso de implementación es crucial para poder realizar pruebas de aspecto energético/económico.

5. Estado del arte

5.1. Control Predictivo

El control predictivo, junto con la teoría de control óptimo y control adaptativo conforman la base de la teoría de control contemporánea. Desde los primeros modelos comerciales como el DMC de Cutler y Ramaker de Shell Oil Co. y MAC/IDCOM (*Identification-Command*) basado en las ideas claves de Richalet et al, han sido técnicas apropiadas y ampliamente aceptadas por el sector petroquímico, principalmente por sus ventajas sobre los métodos de control reactivos (e.g el PID). Este permite un control proactivo anticipando referencias y perturbaciones futuras con una formulación explícita de las restricciones, a la vez que concede la oportunidad de trabajar con modelos no-lineales y/o multivariables directamente. La idea principal del MPC es obtener la acción de control resolviendo repetidamente, en cada instante de de muestreo, un problema de control óptimo de lazo abierto de horizonte finito utilizando el estado actual del sistema como estado inicial. El resultado de la optimización es una secuencia de control en lazo abierto cuya entrada se aplica para controlar el sistema. Gran parte de su éxito viene de la simplicidad del modelo de proceso empleado y la intuitiva idea detrás de la técnica; ya que la identificación de la respuesta impulso del sistema (MAC) o la respuesta escalón (DMC) es relativamente simple de obtener para los operarios.

Más adelante Clarke et al [9] desarrollan uno de los métodos más populares tanto en la industria como en el mundo académico: el GPC (*Generalized Predictive Control*). Tanto la formulación original como posteriores demuestran que GPC/LRPC (*Long Range Predictive Control*) es capaz de controlar establemente sistemas con tiempos muertos variables, sistemas inestables o inversamente-inestables, introduciendo un modelo CARIMA, horizonte de control, ponderación de la acción de control, un modelo auxiliar P y un observador/filtro polinómico T para mejorar la robustez y reducir el esfuerzo de control [8]. Aunque el control, computacionalmente, obligaba a tener un horizonte de control finito el cual no garantizaba estabilidad, en la mayoría de procesos permitía llegar a la estabilidad gracias a un horizonte suficientemente largo. Es necesario mencionar que no se pueden utilizar métodos de análisis convencionales como el criterio de estabilidad de *Nyquist* o el método de polo cero para analizar estos sistemas no lineales, ya que estos métodos están restringidos a sistemas lineales. Sin embargo para el año 2000, con una tardía aplicación de la teoría de estabilidad de Lyapunov se llega a un consenso sobre la estabilidad nominal del MPC tanto en sistemas lineales como en no lineales, por medio de métodos como la adición de un coste terminal/restricción o la extensión del horizonte. Para sistemas estables, el coste de un horizonte infinito de lazo abierto se puede expresar como el coste de un horizonte finito pero con una inclusión de un estado terminal de penalización, el cual debe calcularse mediante una solución a la ecuación de Lyapunov.

En el hilo del estudio de estabilidad, en [57] de G Prasad, GW Irwin, E Swidenbank, y BW Hogg, se plantea la formulación completa no-lineal en espacio de estados para una estrategia de control predictivo garantizando estabilidad nominal asintótica, teniendo en cuenta

perturbaciones medibles y estocásticas para una predicción a largo plazo. Se incluye un procedimiento de linearización sucesiva de las ecuaciones dinámicas para obtener un modelo lineal partiendo de uno no-lineal. En esta formulación se aplica una restricción terminal al final del horizonte de predicción de tal manera que las salidas alcanzan sus valores de estado estacionario asintóticamente. Así, convirtiendose en un problema de control LQ (*Linear Quadratic*), es decir un sistema representado con ecuaciones diferenciales lineales y un coste cuadrático, de horizonte infinito abordable por un controlador LQMPC.

En un mayor ámbito de las cosas, aparte de MPC para sistemas no lineales, David Q. Mayne en [52] (2014) realiza un estudio de los recientes desarrollos y tendencias en el control predictivo en los años hasta la fecha. En este estudio se revela la brecha que existe entre la literatura de MPC que trata con **sitemas *deterministas*** y **sistemas *inciertas/indeterminadas***. Se denomina habitualmente determinista a sistemas cuyos parámetros, aunque obtenidas experimentalmente se dan por conocidas e inequívocas. En el diseño del MPC robusto, el objetivo es concebir un controlador estable independientemente del punto de operación. En la literatura de control predictivo, la robustez del modelo se contempla considerando ciertas clases de incertidumbre. En [1] de Marco A. Rodrigues and Darci Odloak, se consideran relevantes estos tipos de sistemas con incertidumbre: **1)** *Sistemas multi-modelo*, donde las matrices reales (A, B, C) son desconocidas pero pertenecen a un conjunto conocido $\{A_j, B_j, C_j\}, j = 1, 2, 3...L$, donde cada uno corresponde a un punto de operación. **2)** *Sistemas politópicos*, donde se asume que las matrices (A, B, C) pertenecen en un conjunto politópico. Una subclase de este último son modelos cuya incertidumbre se concentra en la matriz de distribución de entradas.

En el apartado de ***sistemas deterministas*** de MPC, se deben mencionar los avances realizados en el estudio de la estabilidad inherente y robusta de MPC nominal, junto al rendimiento del mismo. Asimismo, se introduce un nuevo campo de estudio denominado **MPC Híbrido**; donde se contemplan tanto componentes discretos como válvulas, interruptores...Junto a componentes continuos descritos por ecuaciones diferenciales para representar el proceso. Adicionalmente, se desarrollan metodologías para estudiar MPC desde el aspecto económico (englobados por el término *Economic MPC*), así como el MPC como herramienta que genera la ley de control *online* para determinar soluciones explícitas *offline* de un problema óptimo en casos donde los estados iniciales no son fácilmente obtenibles, y se ve necesario realizar una primera ejecución para reconocer el estado.

En los recientes años, el DMPC (*Distributed Model Predictive Control*) ha recibido bastante atención. Para complejos sistemas como las redes de transporte, redes eléctricas o grandes procesos de producción, un control centralizado no ofrece una solución satisfactoria; esto es debido a que por la complejidad y gran extensión de estos sistemas, surgen problemas de comunicación, modelado y recolección de datos. Este ámbito, aunque sea reciente, ya cuenta con una gran bibliografía; signo de su importancia y necesidad en el sector industrial. Un libro de utilidad recomendado por Mayne es [40] de Negenborn y Maestre. Un artículo de los mismos autores ofreciendo una visión general se presenta en [69].

Dentro de MPC para sistemas deterministas, indagando en los componentes de la estrategia, el problema de optimización a resolver es la minimización de una función de coste $V_N(x, \mathbf{u})$ respecto a **u** (la acción de control), el estado y restricciones a cumplir $(g(x, \mathbf{u}) \leq 0)$. Si la ecuación de estado $(f(\cdot))$ es **lineal**, la función de coste es cuadrática $(l(\cdot))$ y el control, estado y restricciones son poliédricas; entonces el problema de optimización es estrictamente cuadrática convexa y se pueden calcular las soluciones globales efectivamente mediante algoritmos como el método de conjuntos activos, puntos interiores o proyección de gradiente. En cuanto a las técnicas empleadas para la optimización bajo restricciones **no lineales**, se emplea el SQP (*Sequential Quadratic Programming*); el cual bajo condiciones sin

restricciones, se reduce al método de Newton y en el caso de tener solo restricciones de igualdad, es equivalente a aplicar el método de Newton con las condiciones de Karush–Kuhn–Tucker.

La resolución *online* del problema de optimización MPC crece en complejidad cuando se trata un ***sistema con incertidumbre*** además de considerar restricciones. Realizar aproximaciones de procesos industriales reales, frecuentemente, con alguna no-linealidad, es una práctica común. En muchos casos, frente a la existencia de la saturación de actuadores en la planta y especificaciones de seguridad que se traducen en restricciones duras en variables de salida o estado, el problema de control se convierte en no lineal, aún cuando la dinámica básica de la planta es lineal. Tal práctica produce una idealización de la realidad, fundamentada en mediciones de parámetros tomados con la planta real. Sin embargo, hasta las mismas mediciones presentan incertidumbre debido a los errores aleatorios de los instrumentos y de la técnica de medición. Por lo tanto, esto afecta a la validez de las conclusiones obtenidas (optimalidad, robustez, estabilidad…) empleando los sistemas con incertidumbre en sus coeficientes.

El cálculo de la mejor acción de control, cuando el modelo de predicción es determinista es sencillo. Sin embargo, cuando el modelo predictivo es incierto (e.g estocástico, adverso/*adversarial*), se presenta la diferencia entre *open-loop* MPC y *closed-loop* MPC. En *open-loop* MPC, se considera una secuencia única y fija de acciones de control (y no un árbol de ramificaciones de la secuencia de control **u** como ejemplo). Esta secuencia construida con técnicas de optimización (e.g *SQP*), se da por suficientemente buena para hacer frente a todas las combinaciones posibles de indeterminaciones y perturbaciones. En otras palabras, cuando se construye el plan inicial, se ignora el hecho de que se volverá a planificar (con más información) en breve; Se planifica como si se estuviera comprometido con la secuencia calculada.

En cambio, en *closed-loop* MPC se tiene en cuenta el efecto del *feedback*; es decir, se tiene en cuenta el hecho de que se tendrá más información disponible antes de tomar decisiones futuras. Por esta razón, el controlador debe optimizar políticas de control, no sólo la secuencia de acciones de control. Iterar en políticas de control es a menudo impráctico y se recurre a iterar sobre secuencias.

Para aclarar la lógica de lo expuesto; el bucle de ejecución se basa en optimizar una secuencia de control en lazo abierto para una función de coste determinada en un horizonte fijo. Se acuña el término "lazo abierto" debido a que en el instante k mientras se calculan las predicciones $\hat{x}(k|k+j\cdot T_m) \quad \forall j = 1,2,3,..H_p$ no se tiene retroalimentación. Una vez determinada la secuencia, se aplica hasta una nueva actualización del estado, cuando se vuelve a optimizar de nuevo. Aunque esta estrategia supone que el modelo es exacto y que no hay perturbaciones externas, la optimización repetida proporciona un mecanismo de *feedback* que puede corregir los errores de modelado siempre que el bucle de control pueda ejecutarse a una velocidad suficientemente alta. Sin embargo, frecuentemente para sistemas de altas dimensiones y debido a las restricciones computacionales, la tasa de actualización alcanzable del bucle MPC es insuficiente para lidiar eficazmente con perturbaciones e incertidumbres. Ocurre lo mismo en sistemas donde la tasa de actualización de la salida/variables de estado es demasiado lenta por limitaciones técnicas y la predicción del controlador ($\widehat{\mathbf{x}}$) se desvía mucho del estado del sistema real ($\mathbf{x}$).

Un planteamiento ante esta problemática, es tomar las perturbaciones en cuenta de manera que el problema de control óptimo ($\mathbb{P}(x)$) se resuelve requiriendo que se satisfagan todas las restricciones para todas las posibles combinaciones de la secuencia de desviaciones/perturbaciones. Un inconveniente importante de este enfoque, es que el diámetro del "tubo" que recoge las trayectorias tomadas, la nominal junto con aquellas como consecuencia de todas las combinaciones de perturbaciones, puede volverse inabarcable.

Tube-based MPC,una técnica originada en el artículo de Bertsekas y Rhodes [10], es una estrategia en la rama del ámbito de MPC con incertidumbre. El enfoque surge de estudiar los problemas análogos de controlabilidad y seguimiento de sistemas deterministas, en los sistemas con incertidumbre. El objetivo es dirigir el estado del sistema a la referencia final bajo la combinación de todas las posibles perturbaciones y una asunción de "peor caso", y finalmente obtener el "mejor" rendimiento entre ellos. De manera similar se desea bajo las mismas condiciones mantener la trayectoria del estado en el tiempo, dentro de un "tubo"que contiene la trayectoria deseada bajo todas las posibles perturbaciones. Este enfoque aplicado al MPC fue aplicado por primera vez por Chisci et al/Mayne y Langson (2001) [7] [19]. Adicionalmente el término *Robust MPC*, hace referencia al campo del estudio de la robustez aplicada al control predictivo. Son de mención los artículos de Marruedo et al [41] y Pin et al [56].

Orientando el foco a la técnica a implementar en este trabajo, se profundiza subsiguientemente, el contexto del NMPC (*Nonlinear Model Predictive Control*). El interés hacia el NMPC ha ido creciendo, sobretodo a partir de los años 90, junto con el aumento de la necesidad de operar bajo unas especificaciones de rendimiento más estrictas y la necesidad de controlar óptimamente sistemas innovadores de dinámicas acopladas y no lineales. La pregunta clave en NMPC es si una estrategia de horizonte finito conduce a la estabilidad de lazo cerrado. En ausencia de perturbaciones, desajuste entre modelo y planta, y ante la posibilidad de poder computar una solución del problema de optimización para horizontes infinitos, no sería necesario incorporar ningún mecanismo de retroalimentación y serviría con realizar predicciones en lazo abierto partiendo desde $k = 0$ (en discreto) para todo $k \geq 0$. No obstante, en la realidad ninguna de las anteriores condiciones se cumple.

Idealmente se desea obtener una estrategia NMPC con garantías de estabilidad en lazo cerrado e independencia de los parámetros de rendimiento. Un enfoque encomiable que alcanza garantías de estabilidad es QIH-NMPC (*Quasi-Infinite Horizon NMPC*) de Chen y Allgöwer [5] (1998), posteriormente habiendo sido revisado y reformulado en discreto por Chinmay Rajhans et al [6] (2017). En esta estrategia el coste funcional de horizonte infinito, se separa en dos términos: 1) $[t, t+T_p]$ 2) $[t+T_p, \infty)$. El objetivo es aportar una aproximación de límite superior del segundo término mediante una restricción terminal $E(\bar{x}(t+T_p))$. Esto implica que el óptimo del problema de horizonte **finito** delimita el óptimo del problema de horizonte **infinito**.

En hilo a lo expuesto en el siguiente apartado, a diferencia de MPC linear, el problema NMPC supone obtener la solución de un programa no lineal, es decir la solución de un problema de optimización no lineal, no convexo, computacionalmente caro. Los métodos de descenso estándares (gradiente y conjugado, SQP, método de los puntos interiores...) al aplicarlos a un problema no convexo producen mínimos locales en lugar de globales. Por tanto, las conclusiones de estabilidad que dependen de la optimalidad global, no son derivables [53]. En el artículo de conferencia de Findeisen y Allgöwer [22] se recogen algunos métodos de optimización (entre ellos los mencionados en [53]): **1)** Ecuaciones diferenciales parciales de *Hamilton-Jacobi-Bellmann/programación dinámica* **2)**Ecuaciones diferenciales de *Euler-Lagrange*/cálculo de variaciones/principio de máxima **3)** Solución directa utilizando una parametrización finita del control y/o restricciones. En la práctica para una optimización *online* se ha empleado el último enfoque, generalmente por medio de *Sequential Quadratic Programming*. El SQP, un método determinista basado en gradiente, descompone un complejo problema no lineal en una serie de problemas de SQP fáciles de resolver. En este método se hace uso de la información proporcionada por las derivadas en la resolución de dichos problemas descompuestos. Sin embargo, resulta fácil que SQP se atasque en óptimos locales, por lo que puede considerarse como otra técnica de búsqueda local. En esa necesidad de búsqueda global del óptimo, entran los algoritmos metaheurísticos como los EA.

5.2. Optimización

Hoy en día, en el ámbito de la metaheurística inspirada por la naturaleza, existe una gran variedad de algoritmos basados en metáforas; comportamiento de insectos o aves, movimiento del agua, efecto de la gravedad,...Sin embargo los algoritmos evolutivos han sido históricamente el pilar principal: algoritmos genéticos, programación evolutiva, programación genética son algunos de los términos englobados. Por otro lado, algoritmos de inteligencia de enjambre, tales como ACO (*Ant Colony Optimization*), PSO (*Particle Swarm Optimization*) o GOA (*Grasshopper Optimization Algorithm*) y técnicas como *Simulated Annealing* o *Tabu Search* merecen ser mencionados por su mérito y su buen rendimiento en problemas específicos.

Tal y como se ha mencionado anteriormente los EAs engloban varias otras disciplinas. Dentro del campo de los algoritmos evolutivos, se puede realizar una clasificación entre GA (*Genetic Algorithms*), ES (*Evolutionary Strategies*) y EP (*Evolutionary Programming*) [Figura 1].

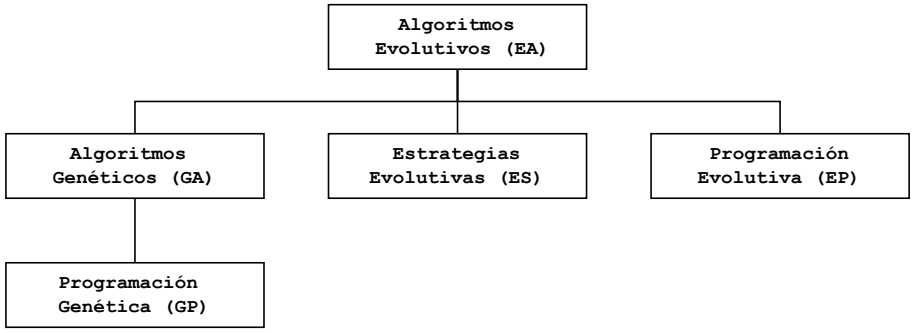


Figura 1: Clasificación de los algoritmos evolutivos

Aunque las corrientes fueron desarrolladas independientemente por pequeños grupos de investigación, a partir de los años 80 todos acogieron el término "algoritmo evolutivo" como agrupación y desde entonces la línea de diferenciación entre ellos se ha vuelto más difusa.

Los **algoritmos genéticos** se remontan a los trabajos de John Holland en los años 60. Clásicamente la representación de las soluciones se ha hecho mediante cadenas de bits, es decir, codificación binaria. Estas cadenas de bits serían los genomas de los individuos, por lo tanto los GAs operan a nivel de genotipo. Se emplean varios tipos de operadores genéticos dependiendo del problema en cuestión. En este caso la operación de "selección" se realiza de una manera no extintiva; todos los individuos de la población tienen posibilidad de generar descendencia. El operador genético principal que se encarga de explorar el espacio de búsqueda es el "crossover". La mutación es una operación de fondo cuyo objetivo es prevenir la estancación en mínimos locales y proveer diversidad.

Una corriente que surge de los algoritmos genéticos es la **programación genética**. Se emplea para la inducción de programas informáticos. Mediante la programación genética se describe el proceso de generación automática de programas informáticos. La programación genética evoluciona programas informáticos, tradicionalmente representados en memoria como estructuras de árbol. Los árboles pueden evaluarse fácilmente de forma recursiva. Cada nodo del árbol tiene una función de operador y cada nodo terminal tiene un operando, haciendo que las expresiones matemáticas sean fáciles de evolucionar y evaluar. Esta metodología favorece el uso de lenguajes de programación (como LISP) que intrinsicamente incorporan estructuras en árbol.

A diferencia de los algoritmos genéticos, que operan a nivel del genotipo, las **estrategias evolutivas** operan a nivel del fenotipo. Por tanto, las estrategias evolutivas tienen un mayor

grado de abstracción y suelen utilizarse para optimizar variables de decisión continuas. Los elementos de las soluciones se representan por medio de *arrays* de números reales. La selección es determinista y extintiva, lo que significa que los individuos con un *fitness* pobre no tienen ninguna posibilidad de producir descendencia. El operador principal de búsqueda a diferencia de los GAs es la mutación.

Finalmente, en cuanto a la **programación evolutiva** se ha de mencionar que se empleaba para generar máquinas de estado finito (*Finite State Machines*) artificialmente inteligentes. Estas máquinas después se utilizaban para la predicción de series temporales. En éste caso el operador de búsqueda es la mutación y el proceso de selección es estocástica y extintiva.

Al igual que el problema que se pretende resolver en la estrategia iMO-NMPC propuesta [68], muchos problemas de diseño, planificación y cálculo de la ingeniería se pueden plantear en términos de problemas de optimización con restricciones formulado de la siguiente manera:

$$\begin{aligned} &\text{Minimizar} \quad f(\mathbf{x}) \\ &\text{Sujeto a:} \begin{cases} g_j(\mathbf{x}) \leq 0, & j = 1, \ldots, m, \\ h_k(\mathbf{x}) \leq 0, & k = 1, \ldots, n, \\ \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u \end{cases} \\ &\mathbf{x} = (x_1, x_2, \ldots, x_D) \end{aligned}$$

Siendo $g_j(\mathbf{x})$ y m, la función y el número total de restricciones de desigualdad y $h_k(\mathbf{x})$ y n, las de las restricciones de igualdad. Asímismo $\mathbf{x}$, es un vector de D dimensiones acotado entre $\mathbf{x}^l$ (límite inferior) y $\mathbf{x}^u$(límite superior).

Dentro de los algoritmos evolutivos, específicamente para la optimización numérica se emplean los algoritmos genéticos; algoritmos de búsqueda estocástica basados en población. Las principales ventajas de este tipo de algoritmos de optimización frente a técnicas más clásicas son las siguientes: **1)** No requieren que la función objetivo sea continua o diferenciable. **2)** No requieren la evaluación de gradientes. **3)** Reacio al estancamiento en mínimos locales * (Aunque tienen un mejor comportamiento ante el estancamiento frente a técnicas tradicionales no está libre del problema).

A continuación se presenta un fragmento de pseudocódigo exponiendo la lógica del AG [54], siendo `P(t)` la población de individuos y `j` el número de iteración. Este simple algoritmo sirve para generalizar los pasos en la mayoría de variantes de AGs. En la llamada `evaluate` se evalúa la población a las funciones objetivo y restricciones definidas.

```
1  procedure genetic_algorithm
2  begin
3      j = 0
4      initialize P(t)
5      evaluate   P(t)
6      while (not termination_condition) do
7      begin
8          j = j + 1
9          select P(t) from P(t-1)
10         recombine P(t)
11         evaluate  P(t)
12     end
13 end
```

Código 5.1: pseudocódigo de un algoritmo genético simple

La mayoría de los algoritmos genéticos presentan las fases expuestas en el fragmento de pseudocódigo. **1)** Una inicialización, para generar la primera población, a menudo

aleatoriamente partiendo de una distribución probabilística. **2)** Una primera evaluación antes del bucle. **3)** El bucle principal donde se recorren las instrucciones de selección, recombinación y evaluación hasta que se llega a la condición de terminación, generalmente por número máximo de iteraciones, tiempo o una diferencia despreciable en el coste. Las diferentes técnicas difieren en la selección y la recombinación (que suele ser mediante operadores genéticos de mutación y *crossover*).[1]

Las primeras formulaciones de los algoritmos genéticos presentaban problemas de retrasos, estancamientos y falta de diversidad. Problemas que en posteriores trabajos se abordarían. Algunos de estos problemas son la **convergencia prematura** a óptimos no globales, la falta de capacidad para un **ajuste local**, o la incapacidad de operar en presencia de **restricciones** no triviales.

Tal y como se explica en [54] (1991) de Michalewicz y Janikow, en 1975 DeJong [17] investigó en cinco modificaciones al algoritmo básico para hacer frente a la convergencia prematura. Los modelos modificados en cuestión son las siguientes: *elitist model*, *expected value model*, *generalized crossover model*. Unos años más tarde, en 1981 Brindle [2] examinó cinco modificaciones adicionales: *deterministic sampling*, *remainder stochastic sampling without replacement*, *stochastic sampling without replacement*, *stochastic sampling with replacement* y *stochastic tournament*. Las modificaciones mencionadas sirvieron como base para desarrollos posteriores.

Adicionalmente los algoritmos genéticos presentan problemas a la hora de realizar **búsquedas locales**. En 1975 Holland sugería emplear los AGs como técnica de preprocesamiento para realizar una primera búsqueda antes de pasar la tarea a un motor basado en el conocimiento del dominio para realizar una búsqueda local, ya que en problemas donde era necesario mucha precisión (e.g más de 6 dígitos) el rendimiento del algoritmo básico era bastante pobre.

Dentro de los problemas mencionados, no obstante, el principal problema que se presentaba y es aún presente con las técnicas de hoy en día es la **gestión de restricciones**. El enfoque empleado para abordar las restricciones tradicionalmente ha sido la adición de una función de penalización. Davis L. (1987) [13] comentaba sobre el siguiente problema:" si se incorpora una penalización elevada a la rutina de evaluación y el dominio es uno en el que la producción de un individuo que viola la restricción es probable, se corre el riesgo de crear un algoritmo genético que pase la mayor parte de su tiempo evaluando individuos ilegales. Además, puede ocurrir que cuando se encuentra un individuo legal, éste expulse a los demás y que la población converja en él sin encontrar individuos mejores, ya que los caminos probables hacia otros individuos legales requieren la producción de individuos ilegales como estructuras intermedias.…"

Posteriormente, [11] de Coello (2002) se presenta un estudio de los diferentes enfoques de la gestión de restricciones con sus ventajas y desventajas. En la literatura generalmente, se clasifican estos métodos en 4 grupos: **1)** Las restricciones solo se utilizan para ver si un punto es factible o no. **2)** La suma de la violación de cada restricción se combina con la función objetivo. **3)** La violación de restricciones y la función objetivo se tratan y se optimizan separadamente. **4)** Las restricciones como la función objetivo se optimizan por medio de una técnica de optimización multi-objetivo. Debido a la simplicidad y facilidad de aplicación, se emplean métodos de penalización mayoritariamente. La idea detrás de este enfoque es la de transformar el problema de optimización restringido en uno sin restricciones, aumentando

[1] En este trabajo se emplean los términos coste y *fitness* para referirse al valor obtenido al evaluar una solución en una función de coste o una función de *fitness*. No obstante, se ha de tener en cuenta que no tienen el mismo significado; generalmente interesa minimizar la función de coste y maximizar la función de aptitud (*fitness*). El primer término proviene del ámbito del *Machine Learning* y el segundo del ámbito de la programación genética. En un problema de minimización la solución más apta (un *fitness* mayor) será la que menor coste evalúe.

la función objetivo por medio de un término de penalización; así, no se limita el espacio de búsqueda, de manera que los individuos que no violen ninguna restricción se evalúan a cero en el término de penalización y solo tienen en cuenta la función objetivo. De la misma manera, en [26] de Huang et al (2009) se propone un algoritmo genético con ordenación restringida (GACS) para problemas de optimización con restricciones, empleando una función de penalización dinámica y un método de clasificación basado en esta penalización dinámica y ordenación de *no-dominancia* (expuesto más adelante). Los resultados experimentales del artículo original y posteriores trabajos, que actuaron como precursos para futuras variantes de AGs, revelan que el algoritmo es robusto, eficaz y eficiente para problemas de optimización con restricciones.

Es de mencionar también, que una desventaja de los algoritmos de la época, era que tenían una gran dependencia y sensibilidad en la selección del vector de pesos a la hora de escalarizar problemas multi-objetivo para resolverlos con técnicas mono-objetivo desarrolladas hasta el momento. En 1994, N.Srinivas y K.Deb [64] proponen NSGA (*Nondominated Sorting Genetic Algorithm*), que vino a abordar este problema. En el mismo artículo realizan un estudio de las desventajas de las técnicas clásicas y proponen un nuevo método de optimización multi-objetivo basado en VEGA (*Vector Evaluated Genetic Algorithm*) de Schaffer (1984) (y la modificación de Goldberg (1989)). La optimización multi-objetivo en principio, es diferente a la optimización de un solo objetivo, en el sentido en que en el caso mono-objetivo normalmente existe una solución *mejor* que suele ser el mínimo o el máximo (según el problema). En un problema multi-objetivo existe un *conjunto* de mejores soluciones, llamadas soluciones *óptimas de Pareto* o *no-dominadas*. En estos casos en lugar de emplear un vector de pesos para asignar una importancia por objetivo, interesa mantener todo el conjunto de soluciones para tomar una decisión informada por medio de un *decision maker* externo, por ejemplo.

El concepto que se introduce con NSGA es el método de selección de *ranking* para preservar buenos individuos y subpoblaciones estables, manteniendo igual los habituales operadores de *crossover* y mutación. En el procedimiento, primero se identifican aquellos individuos *no-dominados* para crear un primer frente. A los individuos de este primer frente se les asigna un valor ficticio de *fitness* para dotarlos con el mismo potencial de reproducción. Después se produce un paso de compartición del valor de *fitness*. Esto se realiza dividiendo el *fitness* de cada individuo por una cantidad proporcional al número de soluciones alrededor, haciendo que coexistan múltiples puntos óptimos en la población. A continuación se realiza la misma operación con el resto de frentes *dominados*. Una vez terminado este proceso, se sigue con la etapa de reproducción; aquí, las soluciones *no-dominadas* tendrán más partes de copias de sí mismos permitiendo al algoritmo una rápida búsqueda y convergencia hacia las áreas *no-dominadas* del frente de Pareto. NSGA demostró poder mantener un potencial reproductivo estable y uniforme entre los individuos no dominados, lo cual era una gran desventaja de VEGA.

Subsiguientemente, en 2002, K.Deb, A.Pratap, S.Agarwal, y T.Meyarivan proponen **NSGA-II** [14], una importante mejora sobre el algoritmo original. El nuevo algoritmo intenta contrarrestar las principales críticas del NSGA original:

1. Alta complejidad computacional en la clasificación. El algoritmo tenía una elevada complejidad computacional de $O(MN^3)$, siendo M el número de objetivos y N el tamaño de la población.

2. Una falta de elitismo. Se demostró que el elitismo (la creación un grupo de élite de los mejores individuos de la población) acelera el proceso de convergencia significativamente y previene la pérdida de buenas soluciones una vez que se han encontrado. Una técnica que no estaba presente originalmente.

3. La necesidad de especificar el parámetro de compartición σ_{share}. El rendimiento del

NSGA original era dependiente de la elección de un buen parámetro de compartición. La nueva versión presenta una solución libre de parámetros de compartición para la preservación de diversidad.

Dentro del hilo de la problemática a resolver pero afinando hacia la técnica de interés en el TFM, NSGA-II presenta un método de gestión de restricciones, para abordar un apartado históricamente problemático. La solución se basa en la selección por torneo binario (*binary tournament selection*), donde se escogen dos individuos de la población y se enfrentan para obtener el mejor. Teniendo en cuenta las restricciones un individuo puede ser viable o inviable según si viola alguna restricción o no. Partiendo de estas opciones se define el operador de *dominación* para enfrentar a los individuos. Una solución i domina bajo restricción a otra solución j si:

1. La solución i es viable y j no.
2. Ambas soluciones j e i son inviables, pero la solución i tiene una menor violación global de restricciones.
3. Ambas soluciones j e i son viables, pero la solución i domina a la solución j.

Se debe destacar que un comportamiento favorable de este enfoque, es que las soluciones inviables que violan diferentes restricciones se clasifican como miembros del mismo frente *no dominado*. Así, una solución no factible, que viola una restricción marginalmente, se colocará en el mismo nivel *no-dominado* con otra solución que viola en mayor medida una restricción diferente. Esto provoca que el algoritmo deambule por la región de búsqueda inviable durante más generaciones antes de alcanzar la región viable, evitando una convergencia prematura y proporcionando diversidad.

El nuevo bucle principal que presento el NSGA-II, se expone a continuación. Inicialmente se genera una población aleatoria P_0, que es clasificado según la no dominación; a cada solución se le asigna un rango igual a su nivel de no dominancia (nivel 1, nivel 2…). En la primera generación, los operadores genéticos de selección por torneo binario, recombinación y mutación se emplean de manera habitual para generar una población de descendencia Q_0 de tamaño N. Debido a que para esta primera iteración aún no se ha conformado el grupo de élite, el procedimiento no es el mismo que para el resto de iteraciones. Los pasos para la generación t serían las siguientes: primero se forma una población combinada, $R_t = P_t \cup Q_t$, de tamaño $2N$. Esta población se clasifica basándose en el criterio de *no-dominación* y se forman los conjuntos $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$…. Cada uno de estos grupos esta compuesto por soluciones del mismo nivel de *no-dominancia*, es decir soluciones del mismo grupo no son ni mejores ni peores entre ellos. La suma de individuos de todos los grupos es superior al tamaño de la población N, por lo tanto es necesario realizar una purga de las peores soluciones para ajustarse al número máximo. Es en esta etapa, cuando se realiza la clasificación por distancia de aglomeración (*crowding distance*); esto es, la distancia media de dos puntos $(i-1, i+1)$ a cada lado del punto i, a lo largo de cada uno de los objetivos. Este valor $i_{distance}$, sirve para estimar el perímetro del cuboide formado utilizando los vecinos más cercanos como vértices. Un punto con una menor distancia, se entiende que esta en una zona más poblada que otro con una distancia mayor. A la hora de clasificar, se premian los puntos menos aglomerados para dotar a la población de mayor diversidad y exploración del frente. Habiendo realizado la clasificación se genera la población P_{t+1}, partiendo de los N primeras soluciones de la población anterior clasificada. La población P_{t+1} se utiliza para generar Q_{t+1} haciendo uso de los operadores de *crossover*, mutación y selección, y se procede con el bucle sucesivamente.

La novedosa implementación basado en la dominancia de Pareto se ha mantenido relevante a través de los años, habiendo evolucionado con muchas nuevas variantes [21] [66] [28] que han intentado reducir su complejidad temporal o mejorar su convergencia al verdadero frente de Pareto.

Más recientemente, un fuerte contendiente que ha surgido entre los algoritmos evolutivos es el DE (*Differential Evolution*), que desde su génesis ha demostrado ser una de las mejores opciones en optimización de problemas de parámetros reales. Obteniendo su lugar en el podio del ICEO (*International Contest on Evolutionary Optimization*) y CEC(*Conference on Evolutionary Computation*), en varios años sucesivos [12], sólamente por debajo de estrategias evolutivas como CMA-ES (*restart Covariance Matrix Adaptation ES*) o variantes como MOEA/D(*Multiobjective Evolutionary Algorithm based on Decomposition*), que se expone más adelante. Este enfoque, a diferencia de los EAs tradicionales no requiere de una distribución de probabilidad diferente para crear su descendencia. Los variantes del algoritmo DE, perturban los parámetros de la población actual con la diferencia escalada entre dos miembros distintos, aleatoriamente seleccionados de la misma población.

En varios artículos comparativos se estudian los algoritmos evolutivos DE, PSO, GA [29] y SEA (*Simple Evolutionary Algorithm*) [70]. En [70] de Vesterstrøm y Thomsen publicado en 2004, ya se demuestra la superioridad de DE en problemas numéricos de parámetros reales habiendo realizado un *benchmark* en 34 problemas. Solamente en 2 problemas con ruido demostró el SEA superioridad; habiendo convergido antes y siendo el único que encuentra el óptimo. En cuanto a la velocidad de convergencia, PSO demuestra ser el más rápido y SEA el más lento. En [29] de Kachitvichyanukul (2012) se comparán también PSO, DE y GA en problemas de optimización combinatoria. LLegando a la siguiente tabla:

	GA	**PSO**	**DE**
Necesitan *ranking* de soluciones	Si	No	No
Influencia del tamaño de población en el tiempo	Exponencial	Lineal	Lineal
Influencia de la mejor solución en el resto de la población	Medio	Más	Menos
El *fitness* promedio no empeora	Falso	Falso	Verdadero
Tendencia para una convergencia prematura	Medio	Alto	Bajo
Continuidad (densidad) del espacio de búsqueda	Menos	Más	Más
Capacidad de llegar a una buena solución sin búsqueda local	Menos	Más	Más
La subagrupación homogénea mejora la convergencia	Si	Si	NA

Tabla 1: Comparación cualitativa de GA,PSO,DE [29]

Algunas de las razones mencionadas en [12], que hacen DE tan atractivo son: **1)** Comparado con la mayoría de algoritmos evolutivos tiene una lógica simple y directa de implementar; Se ha de mencionar que PSO también es sencillo de implementar, sin embargo DE tiene un rendimiento muy superior a cualquier variación de PSO. **2)** Presenta un buen rendimiento en problemas uni-modal, multimodal, con funciones objetivo separables y no separables. **3)** El número de parámetros de control del algoritmo son pocos, y el efecto de cada uno esta bien estudiado. **4)** Finalmente la complejidad de espacio de DE es inferior a la mayoría del resto de algoritmos que compiten en optimización de parámetros reales. Esta característica permite ampliar el empleo de DE a problemas de optimización costosos y de gran escala.

No obstante, DE tiene inconvenientes que han de ser mencionados. En [59] se muestra que DE tiene dificultades considerables en funciones que no son linealmente separables, en estos casos es superado en rendimiento por algoritmos ya mencionados como el CMA-ES. En [65] Sutton et al conjeturan que la estrategia de mutación carece de la presión de selección suficiente a la hora de designar vectores *target* (receptor) y vectores *donor*(donante) para poder

extraer información suficiente en la función no separable. Por otro lado, Langdon y Poli [31] encontraron que para algunas morfologías de funciones objetivo, el algoritmo DE se atascaba en óptimos locales mientras que PSO siempre era capaz de obtener el óptimo dentro de un tiempo límite máximo.

Aunque los algoritmos genéticos sean más adecuados para optimización discreta, esto no los hace descartables para optimización continua. Tal y como se ha mencionado antes, NSGA-II presentado por Deb et al en 2002 [14], superaba el rendimiento de otros dos MOEA contemporáneos, PAES (*Pareto-archived evolution strategy*) y SPEA (*Strengh-Pareto EA*). NSGA-II demostró ser superior en rendimiento a los otros algoritmos evolutivos multiobjetivo, con excepción de un problema altamente epistático, donde PAES fue capaz de llegar más cerca del frente de Pareto. En los MOEAs, la utilidad de cada individuo de la población está determinada por su dominancia de Pareto en relación a otras soluciones. Utilizar solo esta técnica puede desalentar la búsqueda de diversidad. El PAES mantiene la diversidad entre soluciones controlando la aglomeración de las soluciones en un número determinista y preestablecida de celdas de igual tamaño en el espacio de búsqueda, en comparación al mecanismo de diversidad libre parámetros de NSGA-II. Por ello, el NSGA-II podría decirse que es uno de los MOEAs más populares basados en la dominancia de Pareto.

Más recientemente, En [73] Q.Zhang et al (2007) presentan un novedoso MOEA basado en descomposición. El cual descompone un problema de optimización multiobjetivo en un número de subproblemas de optimización escalar y los optimiza simultáneamente. Cada subproblema se optimiza utilizando únicamente utilizando la información de sus varios subproblemas vecinos, lo que hace que MOEA/D tenga una menor complejidad computacional que MOGLS y NSGA-II y sea más fácil de paralelizar. Se espera que la solución óptima de cada subproblema sea una solución óptima de Pareto para el MOP (*Multiobjective Optimization Problem*) considerado. La resolución de un conjunto de subproblemas bien seleccionados puede producir una buena aproximación al frente de Pareto (PS (*Pareto Set*)/ PF (*Pareto Front*)). Obtener una precisa representación del frente de Pareto resulta exhaustivo, sino imposible. Generalmente, para el *decision maker* no es conveniente tener un conjunto indebidamente grande y por tanto interesa manejar un número limitado de puntos del PF que sirvan de buena caracterización del mismo.

Como bien se sabe, es posible convertir un problema de optimización multi-objetivo, en un problema mono-objetivo/escalar mediante la agregación de las funciones objetivos. Esta metodología ha demostrado ser válida para obtener la solución óptima bajo condiciones no estrictas. Entre las técnicas más populares de agregación están la suma ponderada, el enfoque de Tchebycheff y la intersección de límites (*Boundary Intersection*); todas ellas expuestas en el artículo. En MOEA/D, se generan N subproblemas escalares asignando diferentes pesos para la agregación de los múltiples objetivos del problema. Por medio de esta descomposición se alivia el problema de asignación de *fitness*, problema que ha dado lugar a varios enfoques entre los MOEAs como los ya mencionados VEGA, PAES, SPEA-II o el NSGA-II. La idea detrás de este enfoque es que para cada punto óptimo $x*$ de Pareto existe un vector de pesos λ tal que $x*$ también es la solución óptima del subproblema escalar. Así, cada solución óptima del subproblema descompuesto escalar es una solución óptima del problema general de Pareto. En el artículo de Q.Zhang et al se emplea como ejemplo el enfoque de Tchebycheff, que a diferencia de la suma ponderada es capaz de tratar con PFs no convexas, definiendo de la siguiente manera los subproblemas:

$$\text{minimizar} \quad g^{te}(x|\lambda^j, z^*) = \max_{1\leq i\leq m} \{\lambda_i^j \cdot |f_i(x) - z_i^*|\}$$

Siendo $f_i(x)$ la evaluación de cada objetivo a minimizar, z_i^* la mínima evaluación de esos objetivos encontrados hasta la iteración del momento, λ^j el vector de pesos del subproblema j distribuidos uniformemente y m el número total de objetivos. De esta manera, al emplear el

máximo de todos los objetivos (en lugar de la suma ponderada), se penaliza y se minimizará primero en la dirección del más costoso. Se debe tener en cuenta que si g^{te} es continuo en λ, la solución óptima de $g^{te}(x|\lambda^i, z^*)$ debería ser similar a aquella de $g^{te}(x|\lambda^j, z^*)$, siempre y cuando λ^i y λ^j estén cerca. Esta es una característica clave y una de las principales motivaciones de MOEA/D, ya que al compartir información entre las T soluciones vecinas, si en algún subproblema se ha dado con una buena solución, esta dirige y ayuda a las demás.

Hasta el momento, la mayoría de MOEAs del estado del arte no implicaban técnicas de descomposición; cada individuo de la población estaba asociada con el problema en conjunto, y no un problema de optimización escalar en específico. La estrategia MOEA/D aporta las siguientes características: 1) Ofrece una simple y eficiente manera de introducir diferentes enfoques de descomposición desarrolladas en la comunidad de programación matemática. 2) Debido a que MOEA/D optimiza sobre N problemas de optimización escalares, en lugar del conjunto entero, problemas como la asignación de *fitness* y el mantenimiento de la diversidad son más fáciles de tratar. 3) MOEA/D tiene una complejidad computacional inferior en cada generación que NSGA-II y MOGLS (*Multiobjective Genetic Local Search*). Con el enfoque de descompisición de Tchebycheff tiene un rendimiento similar a NSGA-II en problemas continuos. Sin embargo, utilizando técnicas de descomposición avanzadas rinde mucho mejor que NSGA-II en problemas continuos de 3 objetivos. 4) Las técnicas de normalización de objetivos son fáciles de incorporar en cada subproblema para tratar con objetivos de escalas dispares. 5) Adicionalmente, permite incorporar naturalmente técnicas de optimización escalares desarrolladas en la comunidad de programación matemática.

Como investigaciones posteriores, en [39] Hui Li y Q.Zhang comparan el rendimiento de MOEA/D con NSGA-II, ambos con operadores de reproducción diferenciales, demostrando que MOEA/D-DE aunque esta sujeto a la cercanía de las soluciones en el espacio de decisiones, supera significativamente los resultados de NSGA-II ante conjuntos de Pareto con geometrías complejas. Posteriormente en [74], también se compara MOAE/D-DE (basado en operadores diferenciales y polinómicos) y MOAE/D-MG (*Multivariate Gaussian*) (basado en un operador de reproducción de descendencia con distribución Gausiana). El estudio comparativo muestra que el generador de descendencia es una herramienta prometedora para tratar con MOPs continuos.

Ante la necesidad de incluir un mayor número de objetivos para abarcar los problemas de control crecientemente estrictos y complicados, incrementa la necesidad de mejorar los algoritmos EMO (*Evolutionary Multiobjective Optimization*), que se habían probado para no más de 2 o 3 objetivos. K.Deb y Himanshu Jain (2014) parten del marco de trabajo del NSGA-II y presentan un algoritmo evolutivo pluriobjetivo utilizando una clasificación *no-dominada* basada en puntos de referencia (o **NSGA-III**)[16] y lo comparan con dos versiones del algoritmo MOEA/D. En este primer artículo de los dos publicados sobre la nueva técnica, se expone la nueva mejora frente a un MOP de hasta 15 objetivos **sin restricciones**. Asimismo en el segundo artículo [27], se presenta el análisis **bajo restricciones**.

Algunos de los problemas de los EMOs mencionados en el artículo son los siguientes: **1)** La proporción de soluciones no dominadas en un conjunto de de vectores objetivo elegidos al azar se hace exponencialmente grande con un número de objetivos tan elevado. Dado que las soluciones no dominadas ocupan la mayor parte de los puestos de la población, cualquier EMO con mecanismos de preservación de la élite se enfrenta a la dificultad de acomodar un número loable de nuevas soluciones en la población. Esto ralentiza el proceso de búsqueda considerablemente. **2)** La ejecución de operadores de preservación de diversidad como el *crowding distance* o el *clustering operator* se vuelve computacionalmente caro. **3)** La fase de recombinación puede ser ineficiente. En un problema pluriobjetivo, debido al espacio multidimensional es frecuente que dos soluciones escogidas sean distantes en espacio y por lo

tanto su descendencia también quedará lejos de sus progenitores. **4)** La visualización de un frente de grandes dimensiones se convierte en una tarea ardua, lo que dificulta la toma de decisiones y evaluación de rendimiento.

Para aliviar estos problemas proponen dos ideas para algoritmos evolutivos multi-objetivo. La primera es el uso de un *principio especial de dominación*; tal como la dominación ϵ. La dominación ϵ hará que todos los puntos dentro de una distancia ϵ de puntos óptimos de *Pareto* se consideren ϵ-dominados, y por tanto el proceso generará un número finito de puntos *no-dominados* como objeto. La segunda es el uso de una *búsqueda múltiple predefinida*. Esta búsqueda preestablecida se podría implementar como direcciones predefinidas abarcando todo el frente de Pareto o partiendo de puntos de referencia predeterminados. En NSGA-III se reemplaza el operador de distancia de aglomeración por las siguientes técnicas: **1)** Clasificación de la población en niveles *no-dominados*. **2)** Determinación de los puntos de referencia en un hiperplano. **3)** Normalización adaptativa de los miembros de la población. **4)** Operación de asociación. **5)** Operación de preservación de nichos.

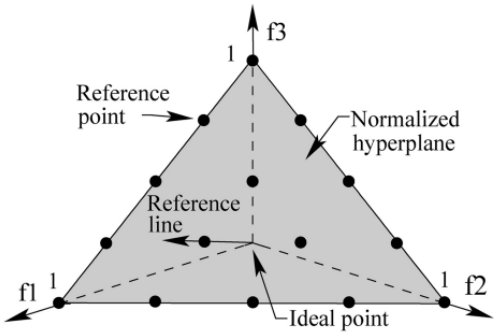


Figura 2: NSGA-III: Representación ejemplo de una búsqueda en un espacio de 3 objetivos con puntos y líneas de referencia [16]

Habiendo expuesto desde algunos de los AGs más clásicos hasta dos de los más populares actualmente, se da una visión del papel que han tenido y tendrán los AGs a lo largo de la historia. Así, mostrando la estrecha relación de las técnicas de control y problemas de ingeniería con los EAs y su mutuo apoyo en su correspondiente desarrollo.

5.3. Modelado de Sistemas

Un modelo físico se construye a partir de un profundo conocimiento del comportamiento interno del sistema. Se realizan descripciones físicas detalladas de a nivel de los subsistemas, y a continuación se unen en un modelo macroscópico cuya dinámica finalmente se representa con numerosas ecuaciones diferenciales parciales lineales/no-lineales. Estos modelos construidos en base a mediciones físicas y conocimiento de sus ecuaciones gobernantes, aunque preferidos en la industria por su fiabilidad, son muy caros de realizar ya que dependen de gran cantidad de datos empíricos (e.g fricción en ruedas, viscosidades de fluidos ...). Adicionalmente son difíciles de utilizar en el control en tiempo real. Por ello, estos modelos de caja blanca/*White-box* no suelen ser asequibles.

Por otro lado, los modelos de caja negra/*Black-box*, tales como las redes neuronales, resultan muy atractivos cuando un modelo físico es demasiado caro de desarrollar. En los recientes años ha crecido el interés por los modelos *Black-box* debido al asombroso rendimiento de LMN (*Local Model Networks*) y GPM (*Gaussian Process Models*). Los modelos de caja negra describen el comportamiento de entrada-salida de un sistema y se ajustan directamente

de los datos experimentales. Es por ello que la calidad de estos modelos es fuertemente dependiente de la información recopilada y *data set* empleado para su entrenamiento. Los modelos de caja negra son sencillos de utilizar y pueden aplicarse a los cálculos en tiempo real, ya que aunque obvian sutilezas del comportamiento completo y están limitadas a un rango de utilización específico, normalmente requieren menos gasto computacional. En modelización de series temporales, para el problema de la estrategia que se pretende abordar, una red NARX es un modelo autorregresivo no lineal con entradas exógenas. Este tipo de modelo es capaz de relacionar el valor de la variable de interés en este instante, con valores pasados de la misma serie y valores pasados de una serie determinada externamente.

$$y_k = F(y_{k-1}, y_{k-2}, y_{k-3}, \ldots, u_k, u_{k-1}, u_{k-2}) + \epsilon_k \tag{5.1}$$

Dentro de los múltiples usos de las redes NARX (predictor, filtro ...) la más significativa es el modelado de sistemas no lineales. Si durante el entrenamiento de la red la salida real esta disponible, se puede emplear la arquitectura *Series-Parallel* mostrada en la figura 3, en lugar de realimentar la predicción de la red como entrada.

Las redes NARX presentan la ventaja de que pueden modelar sistemas no lineales y adicionalmente, en comparación a las redes NNFIR (*Neural Network Finite Imput Response*), no solo incluyen la entrada, si no que emplean salidas del sistema. Tal y como se ha mencionado, en el caso de que se cuente con la salida real del sistema (variable de interés) en ese instante, emplean la arquitectura *Series-Parallel*. No obstante, si se realimenta la misma predicción de la red, se trata de una arquitectura *Parallel* o una red NNOE (*Neural Network Output Error*). La ventaja de esta última estructura es que no es necesario medir la salida real del sistema, la cual debiera ser medida con suficiente precisión para obtener un buen resultado. Sin embargo, un inconveniente que actúa en su contra es que puede presentar inestabilidad ya que esta en un lazo cerrado.[62]

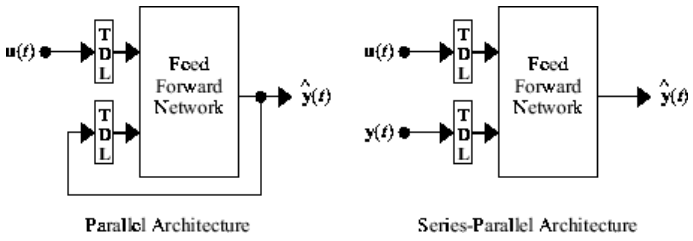


Figura 3: Arquitectura NARX paralelo y serie-paralelo [42], [55]

Un diagrama más detallado de la arquitectura paralela (Figura 4) muestra una red neuronal *feedforward* de dos capas:

Entre las técnicas más antiguas (1887) para representar sistemas no lineales están las series *Volterra*. Las series *Volterra* al igual que las series de *Taylor* sirven para representar el

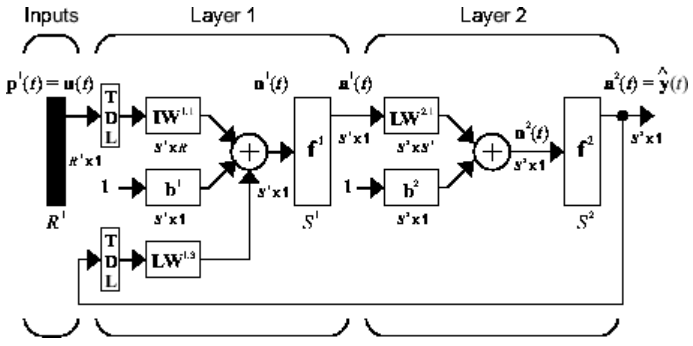


Figura 4: NARX Paralelo de dos capas [42]

comportamiento no lineal. No obstante, difieren de las series de *Taylor* en el sentido de que son capaces de retener la "memoria" del sistema. Las series de Taylor pueden utilizarse para aproximar la respuesta de un sistema no lineal a una entrada determinada, solo si la salida de este sistema depende estrictamente de la entrada en ese instante concreto. En las series de Volterra, en cambio, la salida del sistema depende de la entrada de todos los instantes pasados. Esto permite captar el efecto de "memoria" de dispositivos como condensadores e inductores. En tiempo discreto, la salida del sistema $y(n)$ viene dada por la siguiente expresión, siendo $x(n-\tau_j)$ la entrada en los diferentes instantes:

$$y(n) = h_0 + \sum_{p=1}^{P} \sum_{\tau_1=a}^{b} \cdots \sum_{\tau_p=a}^{b} h_p(\tau_1, \ldots, \tau_p) \prod_{j=1}^{p} x(n-\tau_j) \tag{5.2}$$

Los coeficientes, $h_p(\tau_1, \ldots, \tau_p)$, denominados kernels de *Volterra*, serán cero para todo τ_p negativo, debido a la condición de causalidad. Teóricamente aunque los sumatorios son infinitos, prácticamente se limita a un número finito concluyendo en una suma *truncada* o *doblemente finita*. La estimación de los coeficientes de los kernels de *Volterra* es muy complicado y existen varios métodos para ello. Entre ellos: método de Correlación cruzada, Multi-Varianza, redes *Feedforward*, Algoritmo ortogonal exacto, regresión lineal, el método Kernel y muestreo diferencial [71].

Otras alternativas clásicas de modelado son los modelos *Wiener-Hammerstein*, cuyo advenimiento proviene de el empleo de modelos *Volterra* de complejidad reducida. Estos modelos se componen de una conexión en cascada de dos bloques dinámicos lineales separados por un bloque que representa una no linealidad de tipo estático como se observa en la figura 5. De la misma manera, los modelos *Hammerstein-Wiener* son una conexión de dos no linealidades estáticas separadas por un modelo lineal (invariante en el tiempo). Las no linealidades se pueden clasificar en estáticas, como ya se ha mencionado, y dinámicas. Por un lado, las de tipo estático afectan a la ganancia estática del sistema y por otro lado, las de tipo dinámico afectan al comportamiento dinámico o transitorio del sistema. Estas no linealidades estáticas son frecuentes en los sistemas reales y es por esta razón que el enfoque por bloques posee un gran potencial para modelar estos sistemas. Los bloques lineales generalmente se representan como modelos de transferencia de pulso, modelos de respuesta a impulso o modelos de espacio de estados, mientras que los bloques de no linealidad suelen parametrizarse con polinomios, funciones a trozos, *splines*, redes neuronales, funciones base o *wavelets*. Cuando los bloques lineales y no lineales se representan de alguna de las maneras mencionadas se clasifican como modelos paramétricos. Los modelos Wiener-Hammerstein (mayormente empleados) son flexibles y ampliamente documentados, no obstante plantean un problema de identificación. La literatura sobre cómo estimar el modelo de Wiener-Hammerstein (y los casos especiales de Hammerstein o Wiener) es muy extensa; sin embargo, se puede afirmar que los principales algoritmos de identificación se basan en BLA (*Best Linear Approximation*) y ϵ-*approximation*. En la encuesta realizada por Maarten Schoukens y Koen Tiels [61] se da una visión general de los tipos de modelos no lineales que pueden ser identificados con aproximaciones lineales y a su vez los diferentes algoritmos empleados para ello en el pasado.

Una ventaja de estos modelos, es que una vez perfilada la no linealidad, se puede cancelar o minimizar su efecto en el control. Al evitar la no linealidad estática que pueda presentar el sistema, su control se simplifica enormemente, y así, se posibilita el empleo de técnicas clásicas de diseño, así como teorías de estabilidad y robustez de manera acotada.

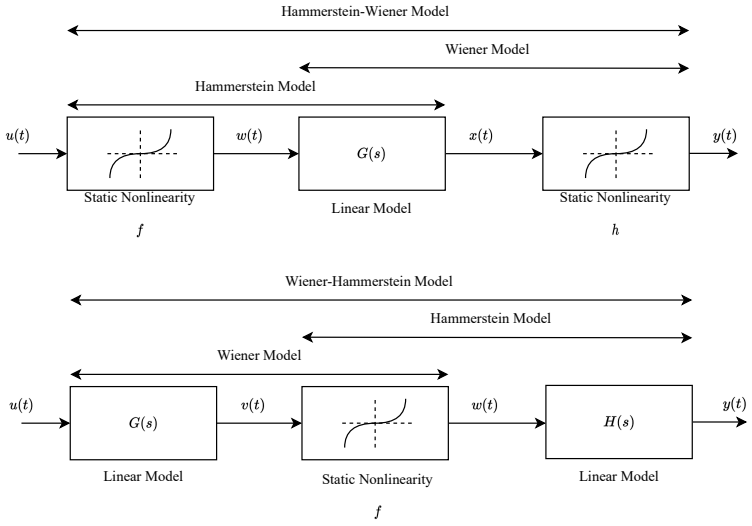


Figura 5: modelo Hammerstein-Wiener y viceversa

Existen otras alternativas intermedias que también se benefician de cantidades masivas de datos. Estos métodos podrían clasificarse como modelos de caja gris/*Gray-box*. Denominados así por ser modelos matemáticos/estadísticos que combinan una estructura teórica parcial a completar con datos experimentales para su culminación. Dentro de esta clasificación se podrían encontrar métodos como el veterano DMD (*Dynamic Mode Decomposition*)[60] y SINDy (*Sparse Identification of Non-linear Dynamics*)[3] que evolucionaron del PCA (*Principal Component Analysis*) de K.Pearson. Entre los mencionados vale la pena destacar el reciente SINDy.

En el caso de SINDy, la suposición de este algoritmo es que sólo hay unos pocos términos importantes que gobiernan la dinámica de cualquier planta; en la práctica acceder a abundantes cantidades de datos es fácil, no obstante, la cantidad de posibles ecuaciones que gobiernan los sistemas de los cuales se extraen los datos, son escasas. En dicho algoritmo, se hace uso de *machine-learning* junto con técnicas de regresión dispersas para obtener modelos con parsimonia que equilibran precisión y complejidad para evitar problemas de *overfitting*.

Teniendo un sistema dinámico en la siguiente forma:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)).$$

$\mathbf{x}(t) \in \mathbb{R}$ representa el estado del sistema en cualquier instante, y la función $\mathbf{f}$ representa las ecuaciones de movimiento de dicho sistema. Refiriendo a lo expuesto en el párrafo anterior de otra manera, dicha función $\mathbf{f}$ solo puede tomar una cantidad limitada de formas y consistirá de pocos términos de todo el espacio de funciones posibles.

Para determinar la función $\mathbf{f}$ de los datos, se recoge un historial temporal del estado $\mathbf{x}(t)$ y en caso de que sea posible, se mide su derivada $\dot{\mathbf{x}}(t)$, de la otra manera se aproxima la $\dot{\mathbf{x}}(t)$ numéricamente. Procediendo de dicha información, se construyen las matrices $\mathbf{X}$ y $\dot{\mathbf{X}}$, que tendrán como columnas los diferentes estados (n) medidos del sistema y las muestras en el mismo instante de tiempo (m), por cada fila.

Partiendo de ahí, se construye de la matriz $\Theta(\mathbf{X})$, consistiendo de posibles candidatos de funciones no lineales aplicadas a la matriz estado $\mathbf{X}$:

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | \\ 1 & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \dots & sin(\mathbf{X}) & cos(\mathbf{X}) & \dots \\ | & | & | & | & & | & | \end{bmatrix}$$

En resumen, debido a que la mayoría de las veces no se tiene acceso a medir la derivada real, ésta se aproxima numéricamente. A causa de esto y el ruido de la medición, las matrices $\mathbf{X}$ y $\dot{\mathbf{X}}$ quedan contaminadas y por ello se plantea el problema de regresión dispersa con un término adicional de ruido:

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi + \eta\mathbf{Z}$$
$$\Xi = \begin{bmatrix} \xi_1 & \xi_2 & \dots & \xi_n \end{bmatrix}$$

$\mathbf{Z}$ se modela como una matriz de entradas gaussianas independientes e idénticamente distribuidas con media cero, y η representa la magnitud del ruido. El objetivo es encontrar los coeficientes de Ξ que determinarán qué no linealidades se encuentran activas en la planta real.

Esta técnica ofrece una gran ventaja frente a métodos de *machine-learning* como las redes neuronales, que requieren de grandes cantidades de datos, donde su funcionamiento interno es poco interpretable, y no son capaces de generalizar mucho más allá que su rango de entrenamiento. Dentro de un límite de datos bajo, SINDy es capaz de extraer los rasgos generales del sistema, permitiendo una mayor generalización.

En un ejemplo de aplicación de la técnica, en el artículo de E.Kaiser, J.N. Kutz y S.L. Brunton [30], se demuestra que el marco SINDY-MPC resultante, tiene un mayor rendimiento, requiere menos datos, es más eficiente computacionalmente y resiste mejor el ruido que los modelos neuronales. Lo que lo hace viable para el entrenamiento *online* y ejecución ante cambios rápidos del sistema.

Todas las técnicas de modelado vienen a demostrar que no hay un claro ganador en todos los apartados, y que cada método tiene algún concepto diferente, digno de estudio y análisis.

6. Análisis de riesgos

Aunque el trabajo en cuestión, no es un proyecto de ingeniería grande, existen factores que pueden poner en riesgo la efectuación del TFM en su totalidad. Es una práctica conservadora realizar una clasificación de los diferentes riesgos acorde con su relevancia para, de esta manera, elegir de forma óptima en qué riesgos se va a concentrar mayor esfuerzo para evitarlos, reducir la probabilidad de que sucedan o que su impacto sea menor.

Debido a que el alcance de este trabajo solo se extiende hasta el desarrollo del código de la estrategia de control inteligente para una plataforma de simulación y su posterior comprobación en *hardware* de tiempo real, se tendrán en cuenta los riesgos que incumban hasta ese punto.

Una herramienta básica para este cometido es la matriz de probabilidad/impacto. Gracias a esta matriz se puede apreciar de manera clara aquellos riesgos que ponderen mayormente en probabilidad e impacto. Se contemplan los siguientes riesgos que se plasman en la tabla 2.

- **A. Falta de stock en hardware de TR o retrasos en la entrega**. Para comprobar el funcionamiento del código y caracterizar su comportamiento temporal es necesario contar con *hardware* que permita una ejecución en tiempo real. Entre las posibles plataformas se contemplan los *Speedgoat* y los PLC compatibles con MATLAB/Simulink ®. Puede suceder que a la hora de hacer el pedido los proveedores tengan falta de stock o que habiendo realizado la compra el producto llegué con retraso. Ambos de estos casos causarían que se retrasase esta fase y que no se pudiera terminar el trabajo a tiempo.
- **B. Fallos de software o peculiaridades en el ordenador donde se desarrolla el código**. Otro posible riesgo es que en el ordenador donde se haya programado y ejecutado el código, exista alguna modificación de MATLAB, diferencia de software o fallo en el sistema operativo. Esto puede haber ocasionado que el desarrollador escriba un programa mancillado (e.g diferentes zonas de memoria, accesos modificados, etc.)que cause que al portarlo a otros ordenadores no se obtenga el mismo resultado o no funcione el programa.
- **C. Duplicidad de una investigación original**. La probabilidad que otra entidad haya investigado la misma estrategia es escasa. Aún así no se puede descartar dicha posibilidad, ya que significaría que se ha realizado trabajo en vano.
- **D. Riesgo de plagio** No solo tiene riesgo la duplicidad de la investigación sin ser consciente de otras entidades; existe la posibilidad que otros organismos privados plagien el trabajo con el objetivo de desarrollar productos o servicios para su propio beneficio.

Al igual que en un proyecto de ingeniería real se propone un plan de contingencia. Los riesgos identificados pueden tratarse de las siguientes maneras: 1) Evitar el riesgo; la solución más económica y que menor impacto supone. 2) Aceptar el riesgo; si no es posible evitar el riesgo, se debe tomar consciencia de ello y no ignorarlo para que no cause problemas a

		Impacto		
		BAJO	MEDIO	ALTO
Probabilidad	BAJO			**C,D**
	MEDIO		**B**	
	ALTO		**A**	

Tabla 2: Matriz Probabilidad/Impacto

largo plazo. 3) Controlar el riesgo; se destinan recursos para solucionar o minimizar el impacto que los riesgos puedan causar. 4) Transferir riesgos; para aquellos riesgos relativos a campos de conocimiento ajenos de la entidad investigadora, se puede transferir la responsabilidad de solucionarlo a otra entidad u organización.

Aunque no todos los riesgos listados tienen la misma importancia, es conveniente trazar el plan de contingencia para todos.

- **A. Falta de stock en hardware de TR o retrasos en la entrega**. Este problema queda fuera de las manos de la empresa. Una forma de evitar o minimizar el impacto que pueda tener es realizar un sondeo de diferentes proveedores y realizar el pedido con suficiente antelación. Así en caso de que el proveedor seleccionado este sin stock se tendrán suministradores alternativos. En caso de que exista un retraso (cuestión de semanas), haber realizado el pedido un par de meses antes bastaría para que cuando se empiece la fase de pruebas, ya se tenga el *hardware*.
- **B. Fallos de software o peculiaridades en el ordenador donde se desarrolla el código**. Una forma de evitar esta problemática, es realizar periódicamente ejecuciones en un conjunto alternativo de ordenadores y modificar el código pertinente para asegurarse de un funcionamiento uniforme y parejo.
- **C. Duplicidad de una investigación original**. La forma para evitar duplicidad es realizar un extensivo estudio del arte en la temática para encontrar similitudes con otros autores y cerciorarse de la originalidad del trabajo.
- **D. Riesgo de plagio**. Para prevenir los plagios se trabaja en repositorios privados de Github y el avance del trabajo solo se comparte con el grupo de investigación y personal relacionado.

7. Descripción de la solución

Se expone en este apartado los pasos seguidos en el desarrollo hasta la implementación en C MEX S-Function de la estrategia de control iMO-NMPC. Las secciones en las que se ha dividido coinciden con los objetivos parciales expuestos en el apartado de *Objetivos y Alcance*.

7.1. Estudio del estado del arte y de la estrategia iMO-NMPC mediante scripts de MATLAB ®

La primera tarea para abordar el trabajo propuesto, es adquirir conocimiento sobre el tema para que después, uno pueda realizar decisiones informadas y pueda depurar adecuadamente los problemas con el código. Tal y como ya se ha expuesto, gran parte del esfuerzo en este apartado ha sido para el estudio del estado del arte. No obstante, en esta sección se expondrá la labor realizada por medio de los scripts.

Gracias al trabajo realizado por Juanjo Valera, Eloy Irigoyen y Kerman Viena, ya se cuenta con scripts que replican el funcionamiento de la estrategia empleando tanto modelos matemáticos como modelos neuronales (NARX). No obstante, el rendimiento temporal empleando la red neuronal como modelo de predicción, habiendo sido entrenada con los datos para un sistema no lineal de *benchmark*, era muy pobre; es decir, la duración del cómputo de la simulación era mucho más prolongada que empleando el modelo matemático. Adicionalmente, debido a la naturaleza robusta de la red neuronal auto-regresiva con entradas exógenas había confusión sobre la validez de la predicción y el orden de los datos de entrada al mismo; ya que la red puede producir resultados similares con e.g $u(k), y(k-1)$ que con $u(k-1), y(k-1)$ como entradas.

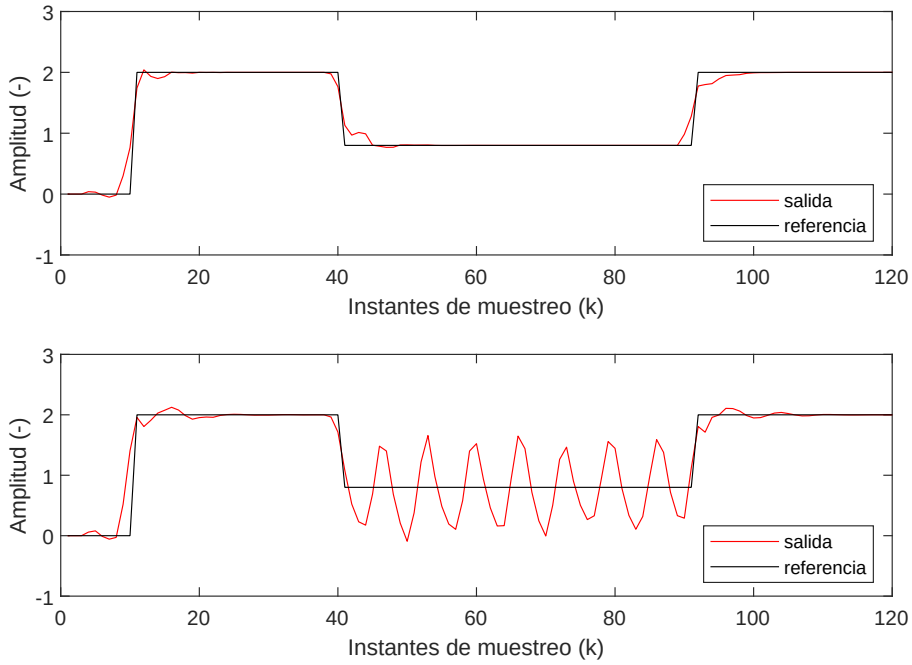


Figura 6: Simulación comparada de la misma red neuronal con **[**$u(k-1)$,$y(k-1)$**] y [**$u(k)$,$y(k-1)$**]**

Como se puede apreciar en la figura 6, el comportamiento del sistema es similar cuando la consigna esta en cero o dos, sin embargo, la red que emplea $[u(k),y(k-1)]$ empieza a oscilar fuertemente alrededor de uno. Esto viene a demostrar la importancia de barrer diferentes puntos de consigna para cerciorarse de una buena predicción, ya que si solo se hubieran contemplado referencias cerca de dos, no se apreciaría ningún comportamiento erróneo de la red.

Para afrontar las mejoras al *script*, primero es necesario presentar la estructura general y sus pasos (figura 7). El *script* principal es `ControlPredictivo_01.m`, del cual se realiza la llamada al optimizador multi-objetivo de algoritmos genéticos. En el paso de ejecución por este archivo se realizan las siguientes tareas: primero se declaran e inicializan las variables globales (como los horizontes, el vector de salidas, el vector de acciones de control, el orden del sistema, el tiempo de muestreo, el modelo de predicción a emplear…) que deben ser visibles desde la función de evaluación. El empleo de variables globales no es una práctica que recomiende MATLAB; aunque da una rápida solución a la necesidad, son ineficientes y dificultan el diagnóstico de errores. Una posible solución a esto, es recoger las variables que requieren ser visibles desde el archivo donde esta la función de evaluación, en una estructura para así pasarla como argumento. Después, se establece la referencia deseada y los parámetros de configuración para el `@gamultiobj`, tales como el tamaño de la población, el número máximo de iteraciones, el rango de las acciones de control y las funciones para las operaciones de *crossover* y mutación. Finalmente como parámetro para el optimizador, se asigna la función de *fitness*, `@EvalObj`.

El archivo `EvalObj_01` contiene la función mencionada. Aquí se toman las variables ya definidas para inicializar dos variables temporales, `yp` e `up`. Estas se inicializarán siguiendo el esquema de la figura 8; almacenando salidas y acciones de control pasadas hasta un número total igual que el orden del sistema en las primeras celdas, y las acciones de control (cromosomas del individuo) generadas por el optimizador en las siguientes celdas para el caso de `up`.

Obsérvese que el número de cromosomas coincide con la cantidad de instantes en el horizonte de control (H_c) y el valor de la acción de control del último se repite un número de instantes necesario hasta llegar al horizonte de predicción (H_p). Para el caso de `yp`, las celdas remarcadas con color rojo serían el resultado del modelo empleado. Por lo tanto, se realizarían las predicciones de H+1 instantes siendo k, el actual.

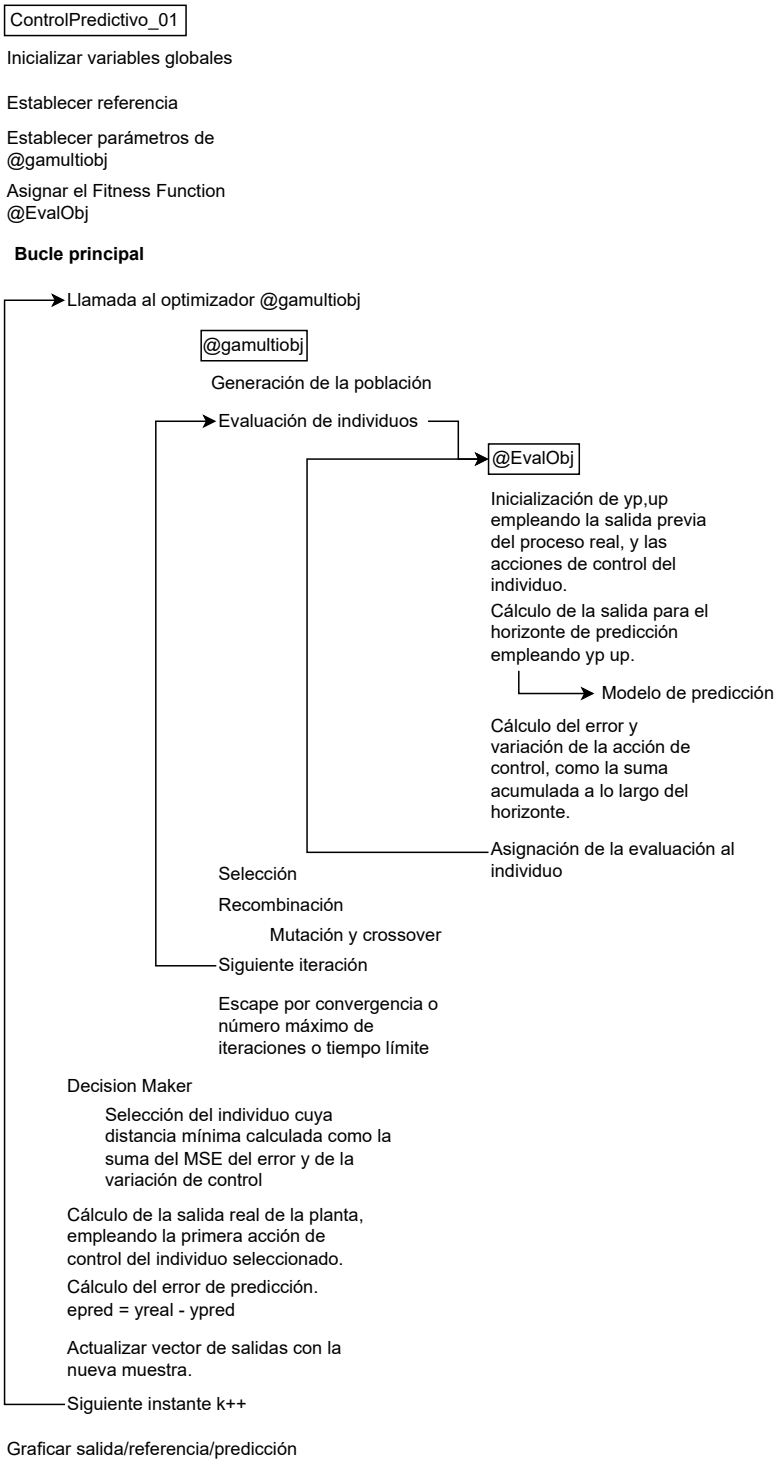


Figura 7: Pasos esquematizados del script de control predictivo inteligente

Una vez asignado la función de *fitness*, comienza el bucle principal. Los pasos resumidos (figura 7) que se repiten en el bucle son los siguientes: **1)** Llamada al optimizador multi-objetivo para generar una población de secuencias de acciones de control. **2)** Selección de la mejor secuencia según los criterios definidos en el *Decision Maker* (e.g distancia mínima al origen de coordenadas en el espacio de *fitness*/coste. **3)** Cálculo de la salida "real" de la planta (por medio representaciones matemáticas de la misma). **4)** Cálculo del error de predicción y el guardado de la salida y la señal de control aplicada.

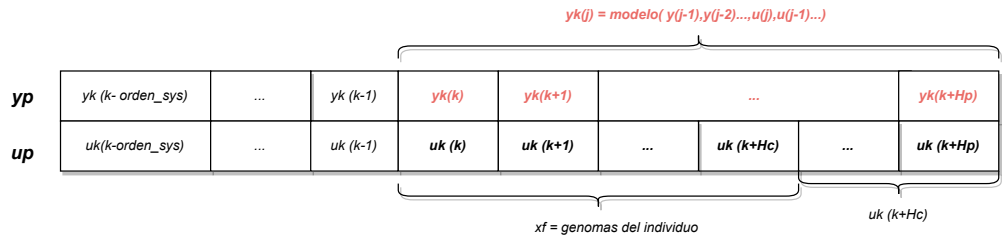


Figura 8: Vectores `yp` y `up`

Como se puede observar la mayoría de la lógica sucede en la llamada al algoritmo evolutivo multi-objetivo. Esta etapa se repite en cada instante de muestreo. Se nombran a continuación las acciones realizadas dentro: primero de todo, se genera una población inicializando aleatoriamente los valores de los cromosomas de cada individuo entre el límite inferior y superior de la acción de control; datos proporcionados anteriormente en la configuración. A continuación se realiza una primera evaluación por medio de la función `EvalObj` definida por el usuario. Los objetivos definidos por norma, en caso de SISO, son la minimización del error $(e(k) = r(k) - y(k))$ y la variación de la acción de control (Δu).

Como ya se ha mencionado, dentro de la llamada a la función de evaluación, se realiza la declaración e inicialización de los vectores `yp` y `up`. En esta sub-rutina, el modelo a emplear para el cálculo de la predicción puede ser uno matemático, es decir una ecuación en diferencias de su dinámica o se puede utilizar un modelo neuronal entrenado para determinado rango de operación. Por medio de una secuencia `for` se rellenan las celdas correspondientes a los instantes desde k hasta H_p del vector `yp`. Se emplean las salidas ya calculadas como retroalimentación al modelo para generar las predicciones de los siguientes instantes del horizonte. De esta manera se puede argumentar que se trabaja en lazo abierto sin información real de cómo respondería la planta. Por último se realiza la suma acumulada del error de seguimiento y la variación de la acción de control, tal y como se presenta en la ingenua implementación del siguiente fragmento de script:

```
%------------------------------------------------------------------------
%        Primer Objetivo. Error trayectoria  mnimo en Horizonte:
%------------------------------------------------------------------------

ref_h = ref(k:k+H-1); % referencia en el horizonte
sum2 = 0;

for i=1:H
    sum2 = sum2 + (ref_h(i)-yp(orden_sys+i))^2;
end

yf(1) = sum2;

%------------------------------------------------------------------------
%          Segundo Objetivo. minimizacion incremento accion de control
%------------------------------------------------------------------------

sum4 = 0;

for i=orden_sys+1:H+orden_sys
    sum4 = sum4 + (up(i)-up(i-1))^2;
end
```

Código 7.1: Fragmento de script: Evaluación de objetivos

Téngase en cuenta de que la suma acumulada es sobre H_p instantes y no $H_p + 1$, y que es

importante discernir los índices en los que empiezan y terminan dichos bucles `for`. Teniendo en cuenta el aspecto de los vectores `yp` y `up`, se debe observar que el índice `orden_sys + 1` corresponde al instante k, es decir los valores `yk(k)` y `uk(k)` (en MATLAB los índices empiezan desde 1).

En cuanto a la variación de la acción de control, se observa que el bucle `''for''` empieza desde el índice `orden_sys + 1` y termina en `orden_sys + H`, haciendo coincidir: `up(i-1) = up(orden_sys) = uk(k-1)`.

Esta es una de las correcciones sobre el código original (7.2), donde se puede ver que dada la estructura anteriormente mencionada de `yp`, al restar `refp(i) - yp(i)` se estaría realizando la operación `ref(k) - yk(k-orden_sys)`.

```
%-------------------------------------------------------------------------
%        Primer Objetivo. Error trayectoria mínimo en Horizonte:
%-------------------------------------------------------------------------

refp = ref(k:k+H);
sum2 = 0;

for i=1:H
    sum2 = sum2 + (refp(i)-yp(i))^2;
end

yf(1) = sum2;
```

Código 7.2: Fragmento de script original: Evaluación de objetivos

Siguiendo con los pasos del *script*, finalmente se asignan estos errores cuadráticos acumulados a cada individuo, por medio de la salida de la función `@EvalObj`. Habiendo recorrido toda la población y evaluado todos los individuos, se continua con las fases del algoritmo genético multi-objetivo propietario de MATLAB, seleccionando los mejores individuos y recombinando para generar la población para la siguiente iteración. Una vez que se llega al número máximo de iteraciones, al tiempo límite de cómputo o a un decremento insignificante en el coste, se finaliza el algoritmo. Así, se vuelve al bucle principal de `ControlPredictivo_01`, donde un *Decision Maker* seleccionará los mejores individuos para cada objetivo compuesto, basándose en criterios definidos por el programador; en este caso, el mínimo de las sumas al cuadrado del error cuadrático y la variación de control cuadrática. Adicionalmente, en el caso de ejemplo mostrado, al tratarse de SISO, solo se desea escoger un individuo. Pero si se deseara controlar un sistema MIMO con varias acciones de control que optimizan objetivos diferentes, se podrían escoger múltiples individuos, cada uno siendo el mejor en los objetivos de interés.

Después, se calcula la salida "real" de la planta; en el caso del script se emplea un modelo matemático como su representación. Aplicando la primera acción de control de la secuencia seleccionada del mejor individuo, se obtiene `yk`, la salida de la planta. En un escenario real, esto se obtendría de mediciones empíricas provenientes de la instrumentación instalada en el proceso, sin embargo en este tipo de script la planta se representa con ecuaciones en diferencias. Finalmente se calcula el error de predicción, se guardan los vectores de salidas y control actualizadas y se pasa al siguiente instante.

Sobre este script se realizan dos mejoras fundamentales: 1) Vectorización de la función de evaluación para algoritmo genético. 2) Reprogramación del *feedforward* de la red NARX para mejorar su rendimiento temporal como computacional.

Como ya se ha mencionado, anteriormente, se empleaba la llamada `net()` de MATLAB para realizar el *feedforward* de la red, llamada que consumía demasiado tiempo del límite asignado que tenía el algoritmo genético, posibilitando solamente unas pocas iteraciones en

comparación con las realizadas en cada instante por en el caso del modelo matemático. Junto a la labor realizada por los compañeros del grupo de investigación se consigue acelerar este proceso, disminuyendo el tiempo necesario en un orden de magnitud. Resulta que era altamente ineficiente hacer una llamada `net()` por cada instante del horizonte de predicción, por cada individuo y por cada iteración. Ya que esta función estaba cargando varios parámetros cada vez y realizando comprobaciones innecesarias, en lugar de cargarlos una vez y solo realizar multiplicaciones matriciales.

Se presentan a continuación, las ecuaciones de dos sistemas que se han utilizado como *benchmark*, también empleados en anteriores trabajos [35] y [36]. Dichos sistemas presentan dinámicas no lineales, tal y como se puede apreciar por sus ecuaciones en diferencias, y sirven de comparativa de la estrategia de control propuesta frente a las técnicas de control clásicas.

$$y_1(k+1) = \frac{1{,}5 \cdot y_1(k) \cdot y_1(k-1)}{1 + y_1(k)^2 + y_1(k-1)^2} + 0{,}7 \sin\left[0{,}5\left(y_1(k) + y_1(k-1)\right)\right] \cdot$$
$$\cdot \cos\left[0{,}5\left(y_1(k) + y_1(k-1)\right)\right] + 1{,}2u_1(k) \tag{7.1}$$

$$y_5(k+1) = u_5(k)^3 + \frac{y_5(k)}{1 + y_5(k)^2} \tag{7.2}$$

Del mismo modo, en el afán de mejorar el coste computacional del script, se realiza una vectorización de la función de evaluación, tanto cuando se emplea el modelo matemático como cuando se utiliza una red NARX. Incluyendo el fragmento 7.3 como nuevo ajuste para los parámetros del `@gamultiobj` se activa la entrada y salida vectorizada del algoritmo genético.

```
options = optimoptions(@gamultiobj,'PlotFcns',[],...);
options = optimoptions(options,'UseVectorized', true);
```

Código 7.3: Activación de vectorización del `@gamultiobj`

En el script 10.1 [1], se muestra un caso de codificación fija del control de dos sistemas conjuntos (los sistemas no lineales SNL1(7.2), SNL5((7.1))), con 4 objetivos de optimización empleando el modelo matemático de predicción. Con esta notación, se pretende hacer uso de la afinidad y el funcionamiento interno que tiene MATLAB para la gestión matricial.

De esta manera, `@gamultiobj` en lugar de entregar un individuo a la función de evaluación cada vez, entrega una matriz de tamaño `[numP x Hc]` (a cada fila le corresponde un individuo y a cada columna un instante de predicción), donde se recoge toda la población. A su vez, para realizar las predicciones, se calcula la columna de celdas que corresponde a `yk(k+1)` para todos los individuos empleando las entradas `yk(k)` y `uk(k)` (figura 9). En el fragmento de script 7.4 se muestra dicho bucle `''for''`.

Obsérvese que SNL5 y SNL1 emplean diferente cantidad de salidas pasadas, esto es debido a que el orden del sistema de SNL5 es de 2 y el de SNL1 de 1. Además se puede ver la adición de `epred(1)` y `epred(2)`, los cuales se corresponden con los errores de predicción de ambos sistemas. Este último término, es una forma de modelizar las perturbaciones y desvíos para poder rechazarlos por medio de una compensación del error que se supone constante en todo el horizonte de predicción. El efecto del mecanismo se muestra más adelante en la figura 13.

```
for i=(orden_sys+1):(orden_sys+1+H)
    yp_1(:,i) = SNL5_fun([yp_1(:,i-1) yp_1(:,i-2)],up_1(:,i-1)) + epred(1).*
    ones(numP,1);
    yp_2(:,i) = SNL1_fun(yp_2(:,i-1),up_2(:,i-1)) + epred(2)*ones(numP,1);
end
```

Código 7.4: bucle for, llamada a las ecuaciones en diferencia de SNL1 y SNL5

[1] Las diferentes versiones de los scripts mencionados en esta sección se pueden encontrar en el repositorio de github : `https://github.com/GICI-UPV-EHU/practicasCI`

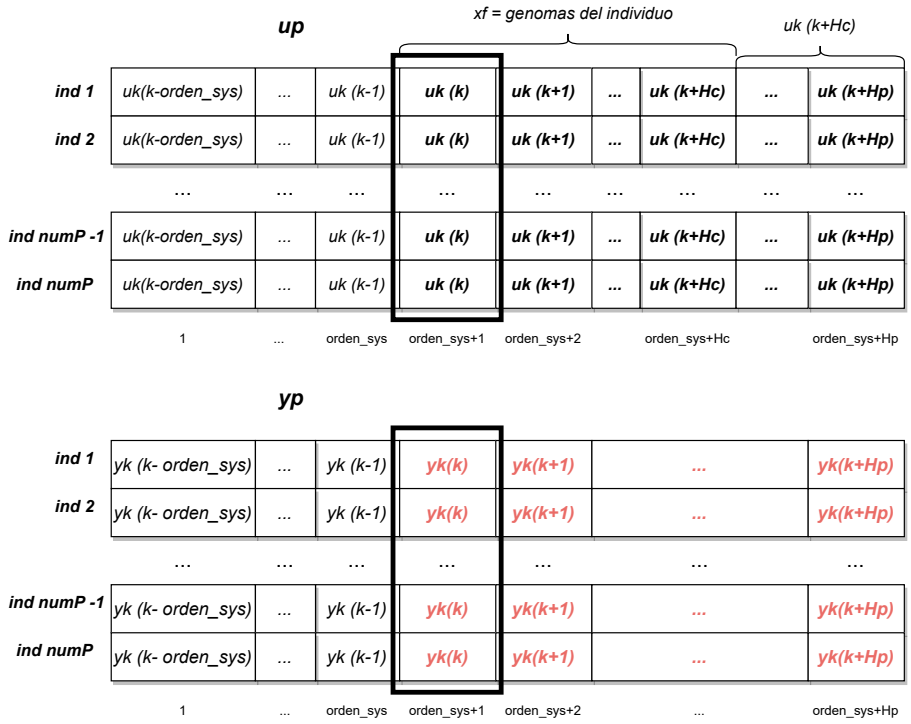


Figura 9: Matrices yp y up para toda la población

Aunque en este caso se esta empleando la misma ecuación matemática tanto para la planta "real" como para el modelo de predicción, se pueden introducir variaciones porcentuales de los coeficientes de la ecuación como parámetros adicionales a la función, tal y como se puede apreciar en 7.5. Como nota al margen, el fragmento a continuación es una de las múltiples posibles formas de implementar la inclusión de variaciones entre modelo y planta.

Idealmente este enfoque es válido, sin embargo en la realidad rara vez se tienen las ecuaciones dinámicas de la planta; esto se debe al gran coste y complejidad que conlleva obtener las ecuaciones dinámicas de la planta. Es más, en las contadas ocasiones en las que se cuenta con las ecuaciones, se da un compromiso entre la simplicidad del modelo y su fidelidad. Por ello, los modelos de *Caja Negra* como las redes NARX resultan apropiadas para un rápido modelizado pudiendo cubrir varios niveles de complejidad sin mayor esfuerzo.

```
function yk = SNL5_fun(yk_p,u_p,optional_varA)
if nargin > 2
    varA =optional_varA;
else
    varA = [0 0 0 0];
end
a1=1.5;
a2=0.7;
a3=0.5;
a4=1.2;
b1=a1 + (varA(1)*a1/100);
b2=a2 + (varA(2)*a2/100);
b3=a3 + (varA(3)*a3/100);
b4=a4 + (varA(4)*a4/100);

yk = (b1.*yk_p(:,1).*yk_p(:,2))./(1+yk_p(:,1).^2+yk_p(:,2).^2)+...
    b2.*sin(0.5.*(yk_p(:,1)+yk_p(:,2))).*...
    cos(b3.*(yk_p(:,1)+yk_p(:,2)))+b4.*u_p(:,1);
end
```

Código 7.5: Ecuación en diferencias del SNL5

En la figura 10 se puede observar el efecto que tiene una variación del 20 % en todos los coeficientes de la ecuación del SNL5. Igualmente se observa que la salida real siempre sigue a la referencia con indiferencia de las discrepancias que pueda haber entre el modelo de predicción utilizado y la planta, gracias a la compensación del error `epred`.

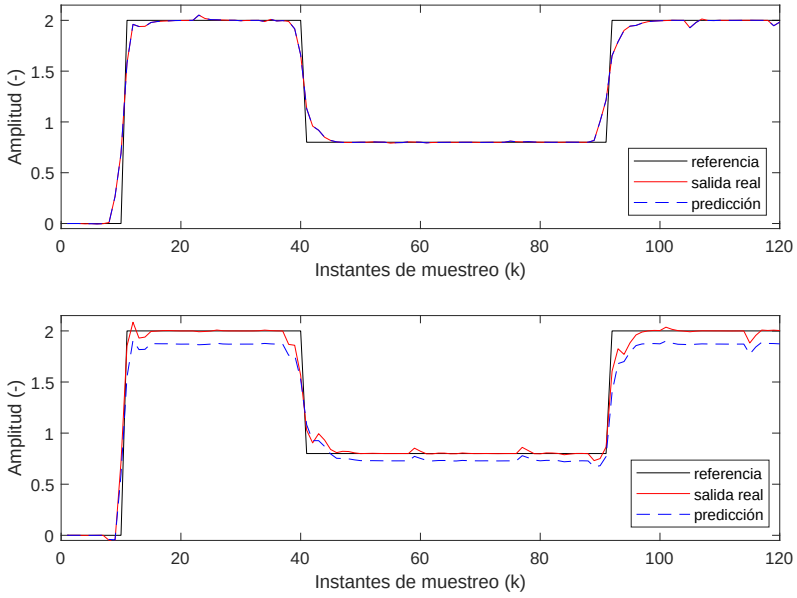


Figura 10: Efecto de una variación de 20 % en todos los coeficientes de la ecuación para la planta SNL5.

Concluyendo con en el script 10.1, se muestra a continuación (figura 11) la evolución de la salida para un horizonte de control y predicción de 4 instantes de muestreo, 200 iteraciones máximas, 200 individuos y la misma ponderación para los objetivos de error de seguimiento y variación de la acción de control.

Se obtiene un resultado satisfactorio para ambos sistemas; con una respuesta más suavizada para el SNL5 y pequeños picos para el SNL1. Este comportamiento se corresponde con las dinámicas que se podrían esperar de cada sistema; el SNL5 teniendo una respuesta oscilante pero menos brusca y en el caso del SNL1, los pequeños picos pueden ser el resultado de una iteración donde la acción de control esta cerca del óptimo pero debido a que `u(k)` esta elevado al cubo, el efecto de esa pequeña diferencia se ve agravada.

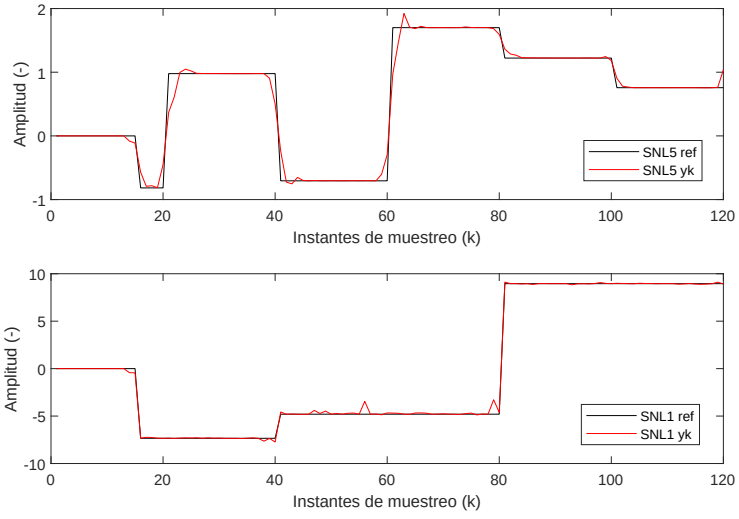


Figura 11: Evolución de la salida para los sistemas SNL5 y SNL1 empleando el modelo matemático para la predicción

El siguiente paso lógico es la inclusión de las redes NARX. Se ha ido aumentando la complejidad del modelo en incrementos antes de llegar a una red NARX MIMO que englobe ambos sistemas. Primero, gracias al trabajo del personal del grupo de investigación, se han obtenido dos redes NARX correspondientes a SNL1 y SNL5, habiendo realizado un barrido de parámetros para obtener las redes con mejor rendimiento y menor complejidad. Estas redes de entrenamiento supervisado, se han obtenido utilizando un rango de acción de control de [-4,4] y las salidas correspondientes a las mismas como *training input*. El algoritmo de *backpropagation* utilizado ha sido Levenberg-Marquardt. Un comentario al margen es que con el algoritmo Levenberg-Marquardt ("trainlm") aunque con un mejor rendimiento en CPU, no se puede hacer un entrenamiento por GPU, ya que esta no soporta el uso de Jacobianos. Si se quisiera entrenar por GPU sería necesario recurrir a *Stochastic Gradient Descent* ("trainscg"). No obstante, aunque es un factor importante a considerar, para redes tan pequeñas como las seleccionadas en la Tabla 3, la diferencia en el tiempo de entrenamiento es despreciable como para inclinarse por un entrenamiento en GPU.

Sistema	**Retardos en u**	**Retardos en y**	**Neuronas por capa**
SNL1	0:1	1:2	[20]
SNL5	0:1	1:2	[20]
SNL1_SNL5	0:1	1:2	[20]

Tabla 3: Redes NARX seleccionadas para SNL1 y SNL5

Estas redes a continuación se estudian dentro del conjunto del script iMO-NMPC, como modelos de predicción. Primero individualmente como sistemas SISO a controlar y posteriormente MIMO. Como paso antes de emplear una red entrenada conjuntamente para representar SNL1 y SNL5 a la vez, se hace uso de las redes individuales para generar un sistema MIMO **desacoplado no lineal** .

La función de evaluación de esta versión se presenta en el *script* 10.2. Esta función es una extensión a MIMO 2x2, combinando el modelo de predicción neuronal desarrollado para ambos sistemas SNL1 y SNL5. Esta versión, sin ser la final, sirve como aprendizaje y demostración del método de *feedforward* manual sin hacer uso de la llamada `net()` de MATLAB. Como se puede apreciar por los retardos seleccionados de los vectores de salida (`yp_net_1 yp_net_2`) y acción de control (`up_net_1 up_net_2`), estas coinciden con los de las redes.

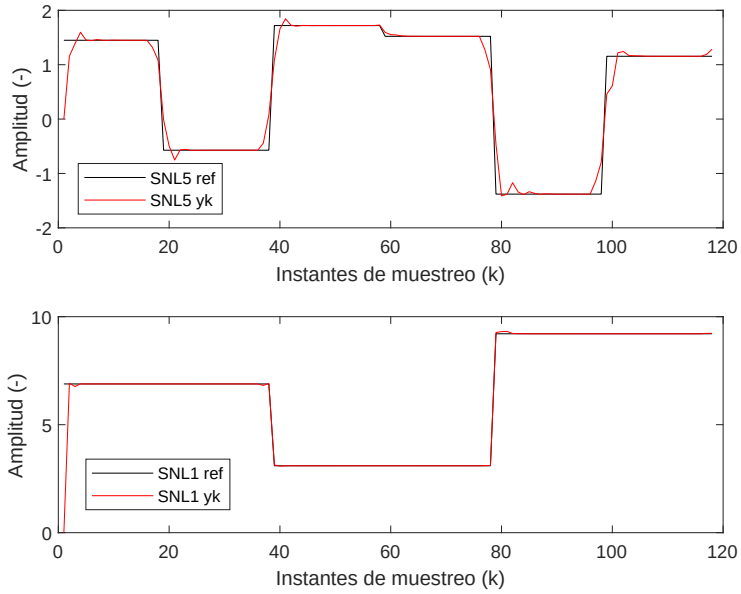


Figura 12: Evolución de la salida para los sistemas SNL5 y SNL1 empleando el modelo NARX conjunto para la predicción

Habiendo comprobado el buen funcionamiento de la combinación de las redes individuales de SNL1 y SNL5, se procede a incorporar la red SNL1_SNL5 entrenada conjuntamente para representar ambos. La modificación de la función de evaluación para éste último caso es el script 10.3. Finalmente el comportamiento del conjunto se expone en la figura 12 para 200 individuos, 200 iteraciones máximas, 4 instantes de muestreo para ambos horizonte de control y predicción, y misma ponderación para los dos objetivos. Con la gráfica presente (figura 12), se puede razonar que se tiene un comportamiento satisfactorio; similar al matemático para el rango de trabajo acotado.

Se incluye también las evoluciones con los mismos parámetros representando una situación de rechazo de perturbaciones, para una perturbación de 0.5 en distintos instantes y una consigna en 0 y 1 (figuras 13, 14). Se puede observar que las perturbaciones en un sistema no tienen efecto en la respuesta de la otra, esto se debe a que no existe ningún acoplamiento entre los dos. Un estudio ulterior con sistemas acoplados resultaría interesante para validar la eficacia de la estrategia ante sistemas no lineales, multivariables y acoplados.

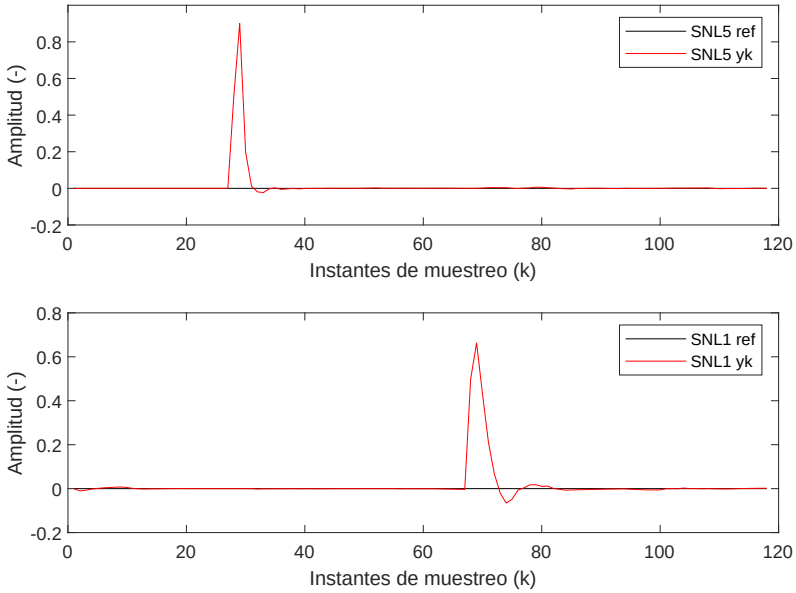


Figura 13: Evolución de las salidas en el rechazo de perturbación de 0.5, referencia en 0.

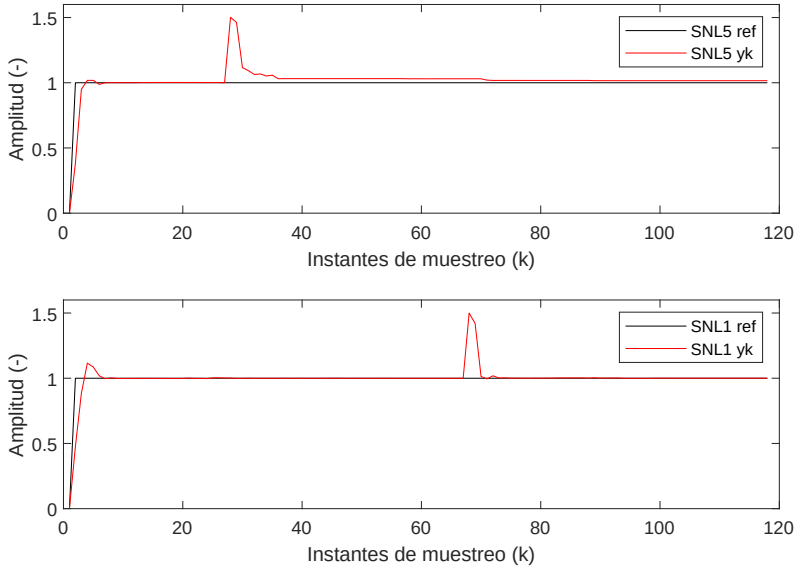


Figura 14: Evolución de las salidas en el rechazo de perturbación de 0.5, referencia en 1.

Finalmente, se introduce un método de inicialización de la población basado en la solución obtenida en el instante anterior. Es lógico pensar que la secuencia de acciones de control del horizonte propuestas en un instante `k` desplazada en el tiempo, será similar a la propuesta en el instante `k+1`. En un esfuerzo de reutilizar el cómputo realizado en un instante `k` para `k+1` y facilitar la convergencia hacia la nueva secuencia de control, en lugar de inicializar la población aleatoriamente, se emplea como semilla el conjunto de los mejores individuos de la población anterior. El algoritmo `@gamultiobj` devuelve por defecto un 35 % de los mejores individuos de la población para el frente de Pareto; por lo tanto para una población de 100 se escogerán 35 soluciones. En la figura 15 se expone gráficamente la operación.

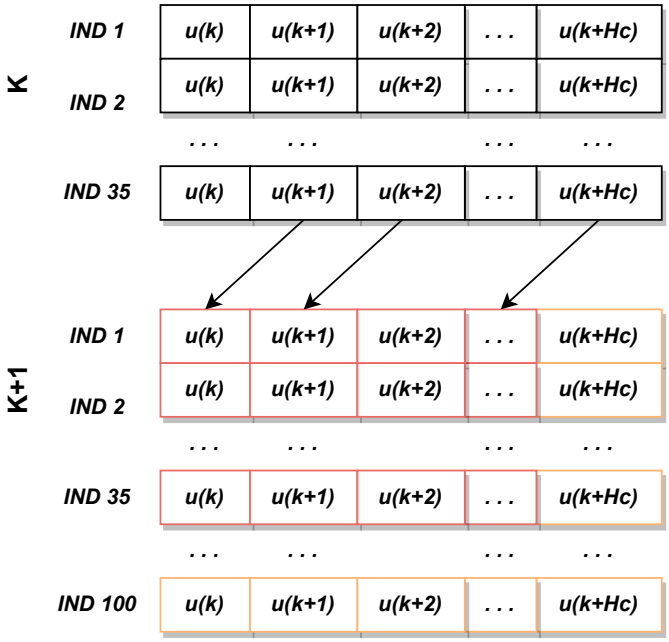


Figura 15: Inicialización de la población `k+1`

El valor del instante `k+Hc` se inicializa aleatoriamente, ya que ese instante del horizonte corresponde a un valor de la consigna que no se ha contemplado en el horizonte del instante `k`. En caso de que de `k` a `k+1` no se haya introducido ninguna perturbación, la secuencia de acciones de control óptima permanecerá igual, y en caso contrario este método no introduce ninguna inconveniencia para la iteración. Por otro lado al suponer un 35 % de la población tampoco se da una pérdida de la diversidad de soluciones, para que se de una convergencia prematura. En cada instante de muestreo al principio de cada iteración se evalúa la siguiente instrucción para dicha inicialización:

```
1    options = optimoptions(options,'InitialPopulation',u_prev_pop);
```

Con este último paso se pausa el trabajo en script. Sin embargo, como ya se ha mencionado, un desarrollo empleando un sistema MIMO **acoplado no lineal** sería de vital importancia para asentar la efectividad de la estrategia, queda por seleccionar y desarrollar el modelo de tal sistema.

7.2. Aprendizaje y Análisis del API S-Function para Simulink ®

Se ha optado por MATLAB®/Simulink® para el desarrollo y simulación de el sistema de control propuesto, al ser el software estándar de facto de computación numérica y prototipado de algoritmos. Simulink, herramienta de modelización visual por bloques, cuenta con bloques de llamada a *S-Functions*, que permiten al usuario programar rutinas personalizadas para su ejecución en conjunción al resto de bloques propietarios. Una *S-Function* es una descripción de lenguaje para un bloque de Simulink escrito en MATLAB®, C, C++ o Fortran. A diferencia de los *S-Functions* escritos en MATLAB® (que son interpretadas), las escritas en C,C++ o Fortran (existe soporte para programar en *Ada* también) se compilan como archivos MEX (*Matlab Executable*) empleando el compilador `'mex'`. Al igual que otros archivos MEX, las *S-Functions* son subrutinas enlazadas dinámicamente que el motor de ejecución de MATLAB® puede cargar y ejecutar automáticamente [51]. Para el desarrollo de estas funciones es necesario seguir una sintaxis especificada por la API de *S-Functions*. Por medio de dicha API, se interactua con el motor de Simulink de igual manera que lo hace con el resto de bloques propietarios.

Se ha considerado emplear esta funcionalidad para llevar a cabo la tarea de implementar la estrategia, no solo porque otorga al usuario una mayor flexibilidad a la hora de programar, sino porque bajo condiciones específicas, permite generar el código de la simulación para su posterior ejecución en plataformas de tiempo real.

Según el lenguaje seleccionado para desarrollar la función especial, Mathworks los clasifica en tres grupos:

1. *Level 1 MATLAB S-Functions*. Una manera muy simple y limitada de programar *S-Functions* mediante *Flags* utilizando lenguaje de MATLAB. Pronto será obsoleto y desechado.
2. *Level 2 MATLAB S-Functions*. Proporciona acceso a un conjunto más amplio de la API de las *S-Functions* y admite generación de código. En esta opción también se emplea el lenguaje de programación de MATLAB. Para habilitar la generación de código, el usuario debe proveer un archivo `.tlc` (*Target Language Compiler*), donde se especifican las pautas de generación [43].
3. *MEX S-Functions*. Permite implementar algoritmos en formato C MEX *S-Functions* o programar *wrappers* como llamada a código ya existente escrito en *C,C++* o *Fortran*. En este caso, proveer un archivo `.tlc` para la generación de código es opcional.

Adicionalmente existen medios para ayudar al usuario desarrollar funciones e incorporar código en C/C++ ya existente:

- S-Function Builder, que proporciona una interfaz gráfica de usuario, sin contemplar los detalles de la sintaxis de la API y permite además incorporar código en C/C++ ya escrito.
- Legacy Code Tool. Esta herrramienta de manera similar a la anterior permite desarrollar *S-Functions* sin atender al API, pero esta vez por medio de comandos de MATLAB.

Para una primera introducción al API se decide empezar desarrollando pequeñas funciones empleando *Level 2 MATLAB S-Functions*, ya que el lenguaje utilizado coincide con el que se viene utilizando para los scripts. La estructura del código y las llamadas a los procedimientos en *Level 2* estan relacionados de manera directa a como se hace en *MEX S-Functions*, por

Level-2 MATLAB Method	**Equivalent C MEX Method**
Setup	mdlInitializeSizes
CheckParameters	mdlCheckParameters
Derivatives	mdlDerivatives
InitializeConditions	mdlInitializeConditions
Outputs	mdlOutputs
Start	mdlStart
Update	mdlUpdate
Terminate	mdlTerminate
...	...

Tabla 4: Nomenclatura de métodos *callback* en Level 2 y C MEX *S-Functions*

lo tanto, queda confirmada la garantía de que la comprensión interiorizada para *Level-2* se traduzca a *C MEX*.

En el script 10.4 se presenta un algoritmo de un controlador PID paralelo filtrado, con los parámetros `Kc, Ki, Kd, N` (Coeficiente del filtro derivativo) entregados por el usuario. [2]

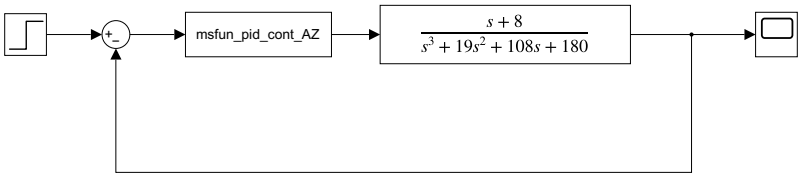


Figura 16: Esquema de bloques en Simulink junto a un *Level-2 S-Function* de PID

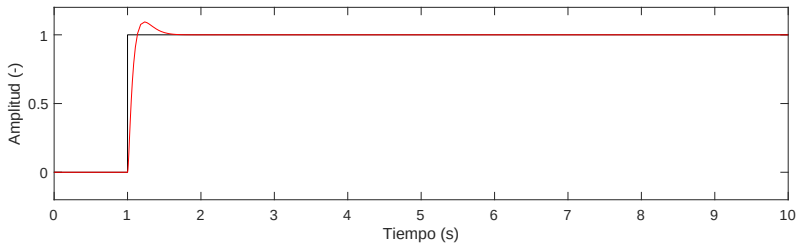


Figura 17: Respuesta del sistema; Kc = 205.77, Ki = 712.03, Kd = 12.67, N = 100

En la figura 16 se muestra la adición de un bloque *S-Function* que realizará las llamadas a las funciones programadas por el usuario. Seguido, en la siguiente figura 17, se tiene la evolución de un sistema elegido arbitrariamente ante un control PID, cuyos parámetros se han seleccionado para un control rápido y robusto. Asimismo, habiendo desarrollado el mismo código (Archivo 10.5) para *C MEX S-Function* se obtiene una evolución idéntica.

La interacción del motor de Simulink con *C MEX S-Functions* se muestra en las siguientes figuras (18,19) y toda su documentación se encuentra en la página oficial de MathWorks [48]. Cabe destacar que aunque el motor de Simulink realiza ciclos a través de todos los métodos de *callback* en la figura, sólo `mdlInitializeSizes, mdlInitializeSampleTimes, mdlOutputs, mdlTerminate` son llamadas obligatorias a definir. Por norma general, se aloja la memoria necesaria y se declararán las variables requeridas en la simulación en `mdlInitializeSizes` o `mdlStart`, estas a su vez, en la inicialización, recurren a `mdlCheckParameters` para la verificación de parámetros con los criterios definidos por el usuario.

[2] Todo el código desarrollado en el proceso de aprendizaje de los *S-Functions*, tanto en lenguaje MATLAB como en C se encuentra en el repositorio de github `https://github.com/GICI-UPV-EHU/S-Function_`

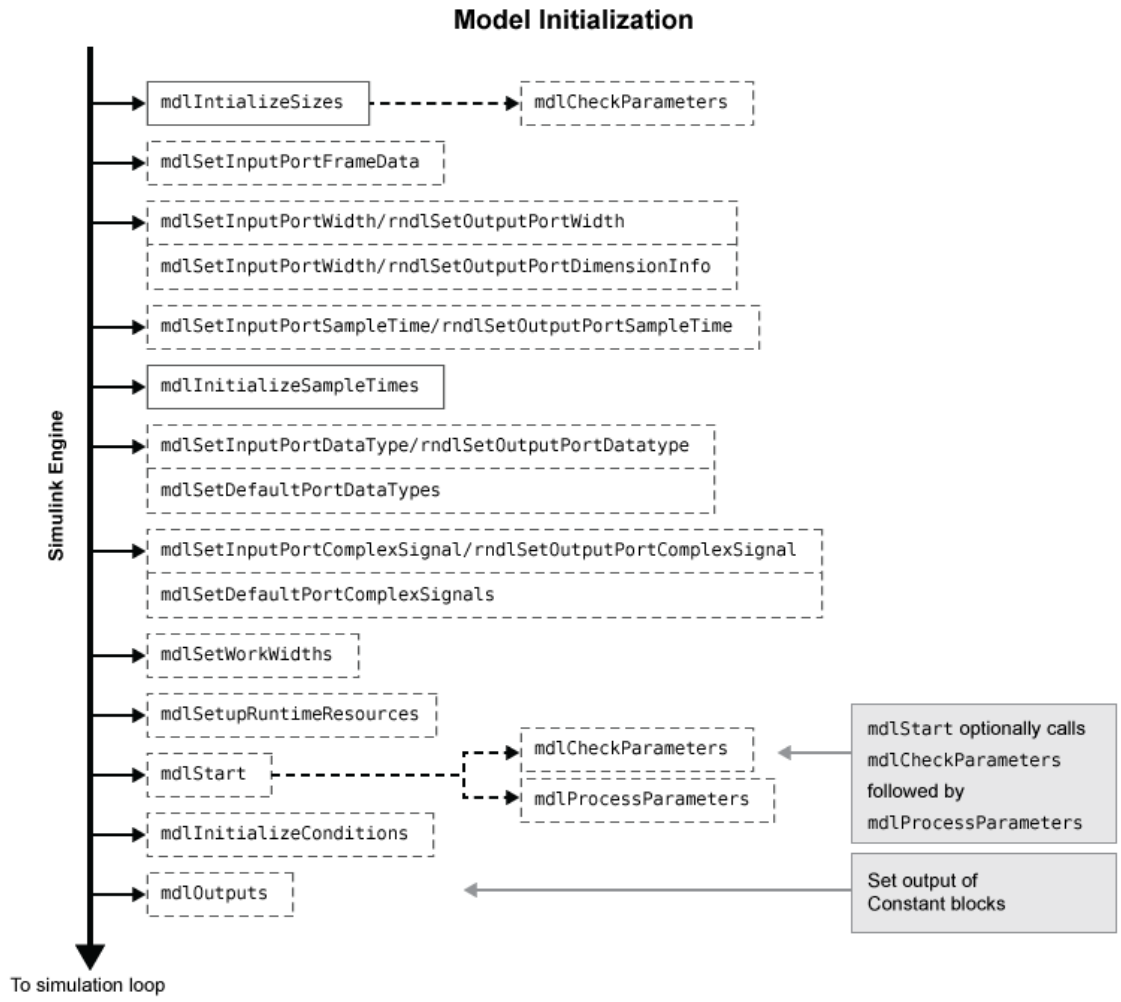


Figura 18: Inicialización de modelo [48]

Finalizada la inicialización del sistema y la propagación de señales, se procede al bucle principal donde primero se calculan las derivadas (`mdlDerivatives`) e integrales, en caso de que el sistema cuente con estados continuos; se comprueba el paso por cero, la variación de parámetros y se actualizan las salidas (`mdlOutputs`). Si se trata con sistemas discretos, los cálculos correspondientes se realizarían en la llamada `mdlUpdate` antes de cerrar el bucle. Terminado el tiempo de simulación, se llama a `mdlTerminate`, donde el usuario programará el desalojamiento de memoria y la configuración de recursos necesaria para cesar la simulación finalmente.

Un desarrollo que vale la pena destacar, el cual ha servido de puente hacia la estrategia completa iMO-NMPC, es un control predictivo, DMC (*Dynamic Matrix Control*) para sistemas lineales (Programa 10.6). Para una formulación de DMC donde no se aplican restricciones, se puede obtener la secuencia $\Delta\mathbf{u}^+$ analíticamente [4].

$$\widehat{\mathbf{y}} = \mathbf{G}\Delta\mathbf{u}^+ + \mathbf{f}$$
$$\mathbf{f} = \mathbf{F}\Delta u^- + \mathbf{y}(\mathbf{k})$$

Con las siguientes dimensiones (siendo N_t el límite de truncamiento de la secuencia de ponderación): `G:[hp x hc], F:[hp x (Nt - 1)], delta_u-:[(Nt -1) x 1], delta_u+:[hc x 1], y:[hp x 1]`

`FirstSteps/tree/examples`.

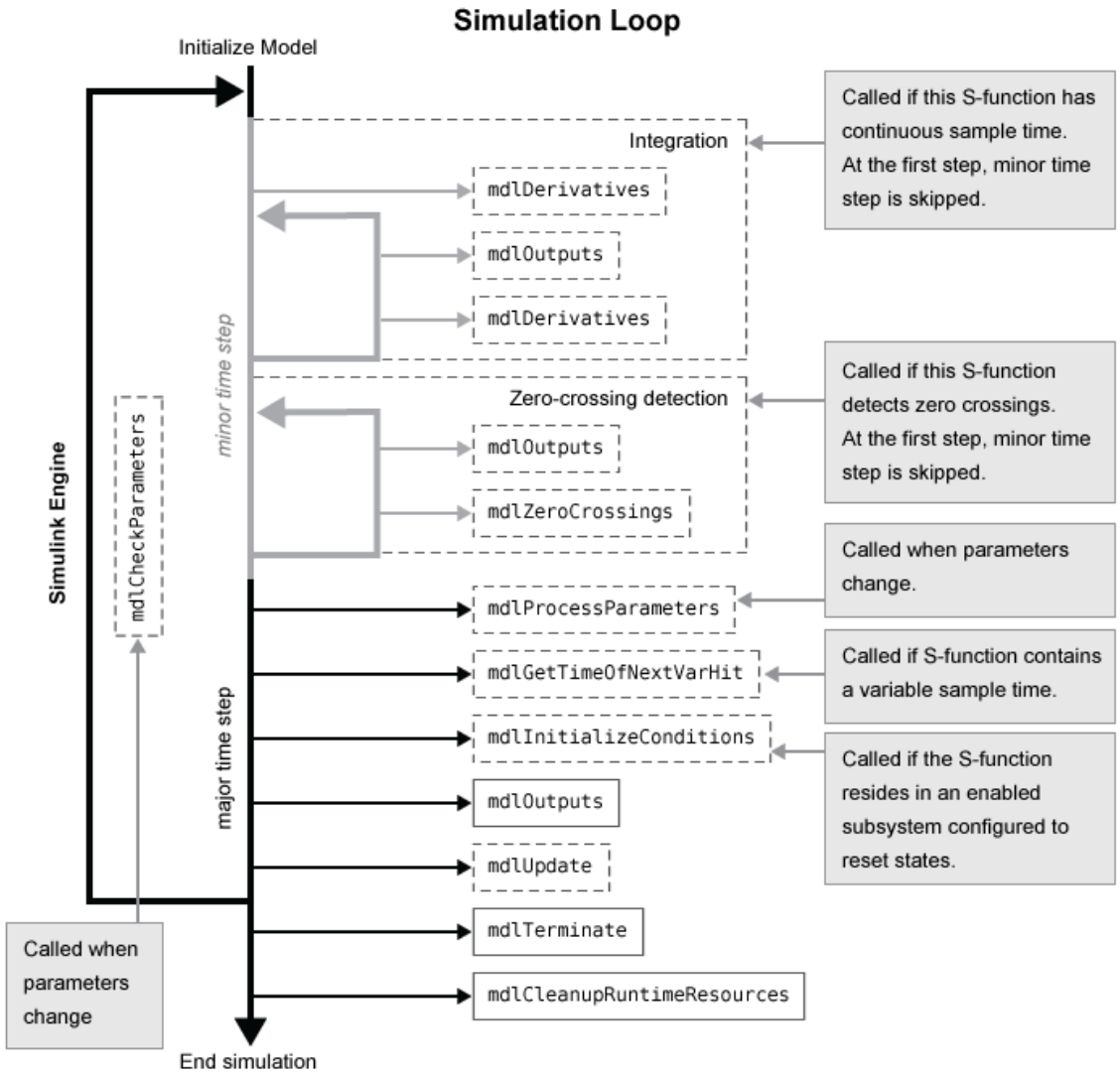


Figura 19: Bucle de simulación [48]

Siendo las matrices $\mathbf{G}$ y $\mathbf{F}$:

$$\mathbf{G} = \begin{bmatrix} g_1 & 0 & \dots & 0 \\ g_2 & g_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{h_c} & g_{h_c-1} & \dots & g_1 \\ \vdots & \vdots & \ddots & \vdots \\ g_{h_p} & g_{h_p-1} & \dots & g_{h_p-h_c+1} \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} g_2 - g_1 & \dots & g_{h_p} - g_{h_p-1} & \dots & g_{N_t-1} - g_{N_t-2} & g_{N_t} - g_{N_t-1} \\ g_3 - g_1 & \dots & g_{h_p+1} - g_{h_p-1} & \dots & g_{N_t} - g_{N_t-2} & g_{N_t} - g_{N_t-1} \\ \vdots & \dots & \vdots & \ddots & \vdots & \vdots \\ g_{h_p} - g_1 & \dots & g_{N_t} - g_{h_p-1} & \dots & g_{N_t} - g_{N_t-2} & g_{N_t} - g_{N_t-1} \\ g_{h_p+1} - g_1 & \dots & g_{N_t} - g_{h_p-1} & \dots & g_{N_t} - g_{N_t-2} & g_{N_t} - g_{N_t-1} \end{bmatrix}$$

Expresando vectorialmente la función de coste de QDMC y ponderando tanto el error como el coste energético:

$$J = (\widehat{\mathbf{y}} - \mathbf{r})^T (\widehat{\mathbf{y}} - \mathbf{r}) + \Delta u^{+T} \lambda \Delta u^+$$

Sustituyendo el modelo de predicción en la función de coste, derivando e igualando a cero para el óptimo, se puede obtener la expresión para la secuencia de acciones de control futuras:

$$\Delta \mathbf{u}^{+} = (\mathbf{G}^{T}\mathbf{G} + \lambda \mathbf{I})^{-1}\mathbf{G}^{T}(\mathbf{r} - \mathbf{f}) = \mathbf{K}(\mathbf{r} - \mathbf{f})$$
$$\Delta u(k) = u(k) - u(k-1) = [\mathbf{K}]_1(\mathbf{r} - \mathbf{f})$$
$$u(k) = [\mathbf{K}]_1(\mathbf{r} - \mathbf{f}) + u(k-1)$$

Multiplicando la primera fila de la matriz $\mathbf{K}$ por la diferencia entre la referencia y la evolución de la respuesta libre, se obtiene la primera variación de la acción de control a aplicar. Estas ecuaciones son las reflejadas en el programa 10.6. Para la manipulación vectorial y operaciones matriciales se desarrolló una pequeña librería de funciones en C[3]. En cuanto a los parámetros de entrada al *S-Function*, estas son `Tm` tiempo de muestreo, `HC` horizonte de control, `HP` horizonte de predicción, `g_i` secuencia de ponderación ante entrada escalón del sistema, `ref` la referencia a seguir y `LAMBDA` el cociente de ponderación de objetivos.

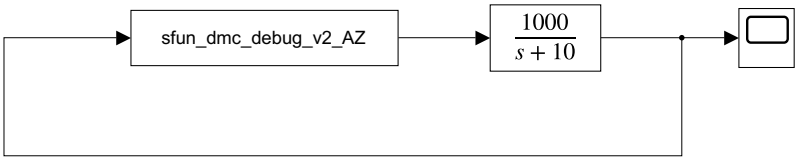


Figura 20: Diagrama de bloques Simulink del control de un sistema incorporando *S-Function* DMC

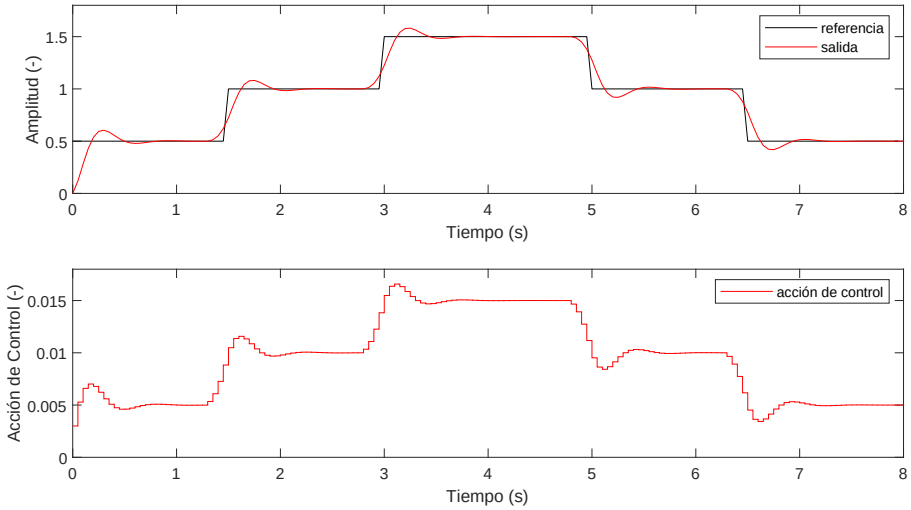


Figura 21: Evolución de la salida y la acción de control del control DMC. Tm=0.05, Hc=5, Hp=5, λ=0.6

7.3. Implementación de la estrategia de control inteligente sobre la plataforma de simulación

Habiendo aprendido las bases del API de programación de *S-Functions*, se aborda el desarrollo del código de la estrategia iMO-NMPC. Se parte del código original en C de Kalyanmoy Deb [15] para el algoritmo genético (NSGA-II) de la estrategia. Para el momento en el que se propone la estrategia y se desarrolla, ya existen algoritmos genéticos que afirman tener un rendimiento superior (tales como NSGA-III y MOEA/D mencionados). No obstante, ninguno tiene un recorrido tan largo y documentado como el NSGA-II. Por otro lado, debido

[3] https://github.com/txetxedeletxe/picoblas.git

a que en el estudio con scripts de MATLAB se ha hecho uso de `@gamultiobj`, que es una variante del algoritmo de K.Deb, el comportamiento de la estrategía con un algoritmo más novedoso diferiría demasiado con lo estudiado para poder proveer una comparación válida.

Anteriormente, para [36], en el grupo de investigación ya se desarrollaron *S-Functions* (permitiendo una compilación de código sobre plataformas industriales) para implementar una técnica de control predictivo basado en *soft computing* en un PC industrial. Sin embargo, aunque en esta implementación también se utiliza el algoritmo genético NSGA-II, se estaba empleando una revisión de código por K.Deb obsoleta. Por un lado, en un esfuerzo de modularización del código y reformulación de la lógica interna para resolver algunos errores de código y por otro lado, debido a cambios en el API y la conveniencia de emplear una revisión actualizada del NSGA-II se decide reprogramar la estrategia de control. La última versión del código del GA de K.Deb es la 1.1.6 del 8 de Julio de 2011.

A continuación se presenta el fragmento de código principal del algoritmo genético 7.7. Como práctica necesaria en C, al inicio y al final se observan sentencias de alojamiento y desalojamiento de memoria. Como se puede apreciar existen tres alojamientos para tres poblaciones distintas: `parent_pop, child_pop, mixed_pop`. Dichas poblaciones se corresponden a las expuestas en el funcionamiento del algoritmo; en cada iteración partiendo de la población progenitora se genera la población de descendencia empleando operadores de mutación y *crossover*. Posteriormente, para mantener el elitismo y no perder los mejores individuos de la población progenitora se combinan ambos para generar la población de mezcla (que como se puede apreciar es del doble de tamaño), donde se evaluarán y clasificarán de nuevo los individuos progenitores y descendientes. De esta manera, habiendo clasificado todas las soluciones se escogen todas las superiores hasta llegar al límite de individuos de la nueva población progenitora en la siguiente iteración.

Tales poblaciones son de tipo `population` que albergan un elemento de tipo `individual`. Estos tipos son estructuras con los siguientes elementos:

```
typedef struct {
    int rank;
    double constr_violation;
    double *xreal;
    int **gene;
    double *xbin;
    double *obj;
    double *constr;
    double crowd_dist;
} individual;

typedef struct {
    individual *ind;
} population;
```

Código 7.6: Tipos population e individual

Se explica a continuación el significado de cada elemento de la estructura. El elemento `rank` sirve para asignar en él, el rango que ocupa el individuo en la clasificación; el elemento `constr_violation`, aunque autoexplicativo, guarda la medida en la que se ha violado la restricción establecida, como otro criterio de clasificación; adicionalmente, el elemento `crowd_dist` guarda el valor de aglomeración de cada individuo, que se emplea como medida de la singularidad de la solución; `xreal` es el puntero que apunta a la zona de memoria donde se guardarán los cromosomas[4] si se trabaja con números reales; `xbin` es el puntero que apunta a la zona de memoria donde se guardará el valor decodificado del vector de genes (en unos

[4]Un cromosoma es la estructura más condensada de una molécula de ADN con proteínas. El genoma (material genético) de un individuo se caracteriza por una serie de cromosomas (e.g los humanos tienen 46

y ceros) cuando se trabaja con variables binarias, guardado en la zona de memoria apuntada por el puntero doble gene. Finalmente los elementos obj y constr son punteros que apuntan a la zona de memoria donde se guarda el resultado de cada objetivo y restricción después de aplicar la función de evaluación a la solución del individuo.

La llamada allocate_memory_pop(parent_pop,popsize) se encarga del alojamiento de memoria de la población. Internamente asigna un puntero a un elemento "ind" (pop ->ind) y llama a la función allocate_memory_ind(&(pop->ind[i])) hasta el tamaño de la población asignado.

La instrucción randomize() es parte de la lógica de la generación de números aleatorios partiendo de la semilla que se provee al algoritmo en los parámetros. Habiendo puesto en marcha la generación de números aleatorios, se inicializa la población progenitora, rellenando los cromosomas de los individuos con variables reales o binarias.

```
1     allocate_memory_pop(parent_pop, popsize);
2     allocate_memory_pop(child_pop, popsize);
3     allocate_memory_pop(mixed_pop, 2 * popsize);
4     randomize();
5
6     initialize_pop(parent_pop);
7     decode_pop(parent_pop);
8     evaluate_pop(parent_pop); // nobj and ncon must match define test_problem()
      -
9     assign_rank_and_crowding_distance(parent_pop);
10
11    for (int n = 2; n <= ngen; n++){
12          selection(parent_pop, child_pop);
13          mutation_pop(child_pop);
14          decode_pop(child_pop);
15          evaluate_pop(child_pop); // apply fitness function
16          merge(parent_pop, child_pop, mixed_pop);
17          fill_nondominated_sort(mixed_pop, parent_pop); //gen new pop
18    }
19    if (nreal!=0){
20        free (min_realvar);
21        free (max_realvar);
22    }
23    if (nbin!=0){
24        free (min_binvar);
25        free (max_binvar);
26        free (nbits);
27    }
28    deallocate_memory_pop (parent_pop, popsize);
29    deallocate_memory_pop (child_pop, popsize);
30    deallocate_memory_pop (mixed_pop, 2*popsize);
31    free (parent_pop);
32    free (child_pop);
33    free (mixed_pop);
```

Código 7.7: Pasos NSGA-II en C

La primera iteración difiere de las iteraciones del bucle principal, aunque si coincide en las instrucciones decode_pop y evaluate_pop, donde se decodifica la secuencia de unos y ceros en caso de trabajar con variables binarias y se evalúan las soluciones propuestas ante una función objetivo definida por el usuario. Finalmente, en esta primera iteración debido a que no se tiene más individuos que el límite de la población, basta con asignar el rango, el índice de aglomeración a cada individuo y clasificarlos. En las iteraciones del bucle principal, en lugar

cromosomas en su genoma). En cambio, un gen es una sección del cromosoma, es una unidad más básica en la representación del material genético de un individuo.

de inicializar la población aleatoriamente, ya se puede partir de una población progenitora y aplicarle operadores genéticos de selección y mutación para generar la descendencia. Esta población descendiente se evalúa y se asignan los correspondientes valores a los elementos de cada individuo mencionados anteriormente. Para concluir la iteración se combinan ambas poblaciones para extraer las mejores soluciones.

Antes de terminar la ejecución se desaloja la memoria reservada para evitar fugas, y con esto se concluye la visión de conjunto del código NSGA-II. El código de K.Deb en su extensión total se puede encontrar en [15] y su implementación el repositorio `sfun_imo_nmpc` de GICI [72].

Se muestra en el código 10.8 el *S-Function* principal desarrollado. La lógica de la estrategia se ha dividido en llamadas `mdlInitializeSizes`, `mdlStart`, `mdlOutputs`, `mdlUpdate` y `mdlTerminate`. En la llamada `mdlInitializeSizes` se aloja la memoria para las variables del problema y se inicializan dichas variables declaradas; en `mdlStart` se aloja la memoria para las poblaciones, se arranca la generación de números aleatorios y se llama a la inicialización del modelo de predicción. Se actualizan las salidas con la acción de control seleccionada en `mdlOutputs` y se realizan los cálculos para la acción de control en el siguiente instante en `mdlUpdate`.

La estructura del código fuente empleado (omitiendo los archivos *header*) con la adición de los nuevos archivos desarrollados queda de la siguiente manera Figura 22:

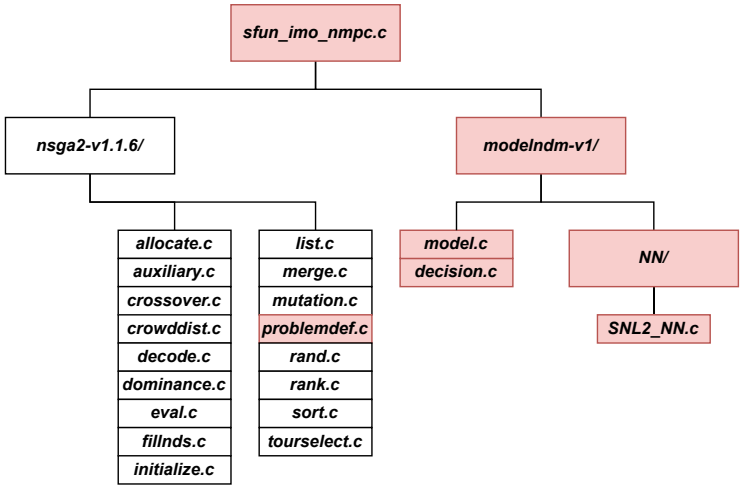


Figura 22: Esquema representando los archivos del código fuente con las adiciones resaltadas en rojo.

Se muestran en rojo los archivos nuevos desarrollados, exceptuando el caso del archivo `problemdef.c` que estaba presente en el código del algoritmo genético, al cual se le han añadido funciones adicionales para integrarlo con el modelo de predicción. Como se puede ver en el esquema, se ha añadido un nuevo directorio `modelndm-v1/` (*Model and Decision Maker version 1*) que recoge los archivos de código para la lógica del modelo de predicción y la toma de decisiones en la acción de control. Asimismo, incluye también otro directorio `NN/` que recoge la lógica para aquellos modelos neuronales. De momento, solo contiene un modelo neuronal entrenado para el sistema no lineal 2, para un rango óptimo de trabajo [-0.5,0.5] en la acción de control.

En la siguiente figura 23, se muestra en un gráfico combinado la interacción del motor de simulink y el código correspondiente a cada llamada de forma reducida. Los fragmentos mostrados en la figura corresponden a la versión SISO antes de realizar la extensión de la estrategia a MIMO.

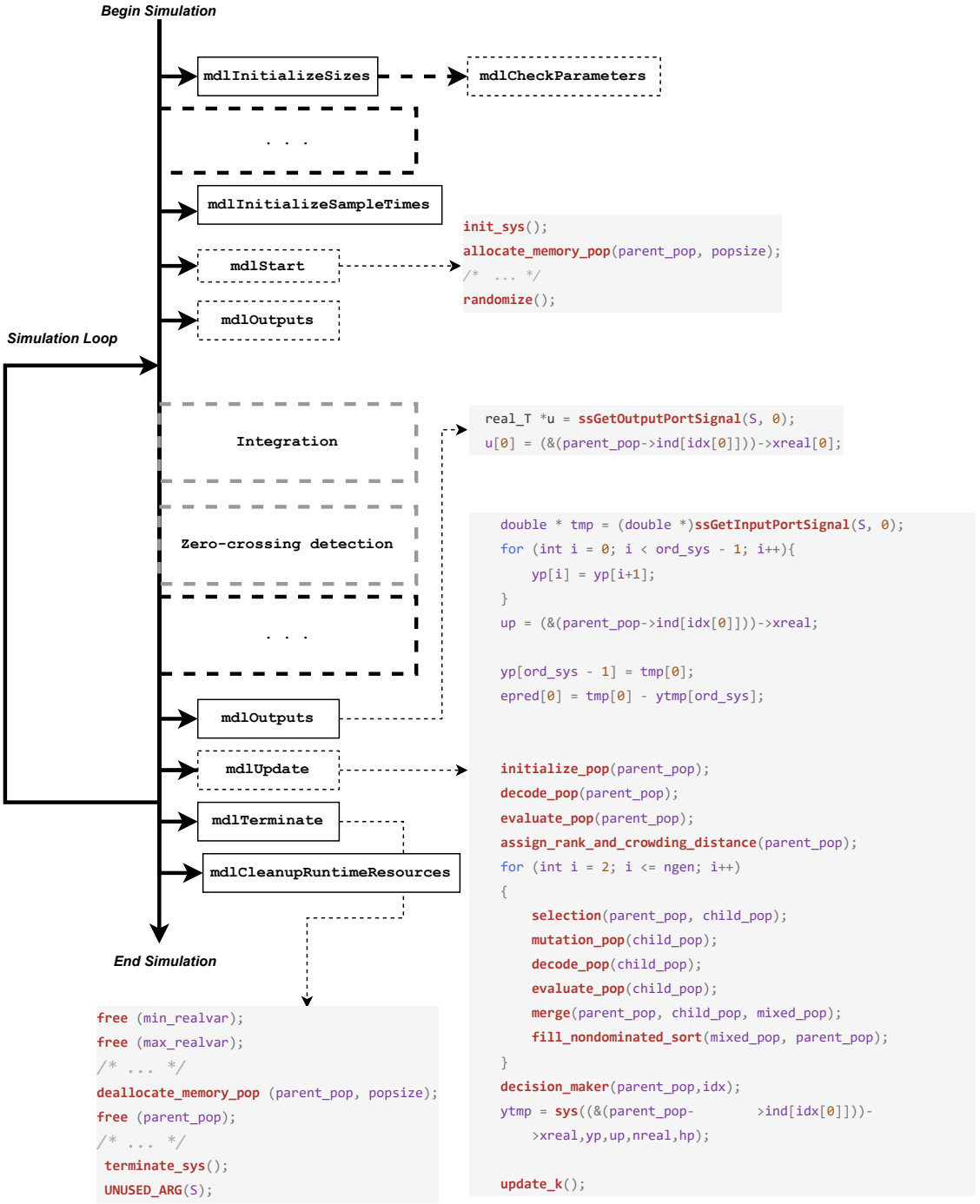


Figura 23: Código correspondiente de `sfun_imo_nmpc` a cada llamada en los pasos de simulación de Simulink.

En la llamada `mdlUpdate` ocurre gran parte del cómputo del algoritmo. Primero se guarda en una variable temporal la señal del puerto de entrada que corresponde a la salida del sistema en ese instante. Después se desplazan los contenidos del vector yp que contiene salidas pasadas para hacer sitio para el nuevo dato. De la misma manera se asigna en up la zona de memoria donde esta guardado la secuencia de acciones de control calculadas en el instante anterior. Se calcula el error de predicción como la diferencia entre el valor de la salida actual y la predicha para el mismo instante. Se suma este error a la salida predicha como compensación bajo la suposición de que esta diferencia se mantendrá constante durante todo el horizonte de predicción. Esta práctica, que provee un mecanismo para el rechazo de perturbaciones y

la disparidad entre modelo y planta, es una de múltiples técnicas empleadas en el ámbito del control predictivo. Siguiendo con la ejecución de instrucciones en el apartado `mdlUpdate`, se procede con el algoritmo genético para generar una plétora de soluciones no-dominantes entre sí.

En este apartado, la diferencia esta en la llamada a la función `evaluate_pop(child pop)`. Esta a su vez, llama a otra más para cada individuo de la población, `evaluate_ind()`. Al evaluar los individuos se emplea `test_problem()`, en el cual el usuario define el problema con el que se va probar la secuencia del individuo. En el caso actual para un problema con dos objetivos, el coste energético (variación de la acción de control) y el error de seguimiento, se tiene la siguiente definición:

```
#ifdef mse2

double * refx;
double * yp;
double * up;
int hp, k, ord_sys;

void test_problem (double *xreal, double *xbin, int **gene, double *obj, double
     *constr)
{
    double * yt = sys(xreal,yp,up,nreal,hp);
    double sum_e = 0.0;
    double sum_u = pow(xreal[0] - up[0], 2.0);

    for(int i = 1; i < hp+1; i++){
        sum_e += pow(refx[k+i] - (yt[i+ord_sys-1] + epred[0]),2.0);
    }
    for (int i = 0; i < nreal - 1; i++){
            sum_u += pow(xreal[i+1] - xreal[i],2.0);
    }
    obj[0] = sum_e;
    obj[1] = sum_u;

    free(yt); // cut memory leak.
    return;
}
#endif
```

Código 7.8: test_problem

Como se puede apreciar mediante la función `sys()` se obtiene el vector de salidas para el horizonte de predicción. Este vector se comparará con la referencia para obtener el error de seguimiento. El error de seguimiento que se guarda en el elemento `obj[0]` del individuo, es el sumatorio del error al cuadrado en todo el horizonte de predicción. Igualmente, el coste energético, se guarda en el elemento `obj[1]` como segundo objetivo, y es la agregación de la diferencia entre acciones de control en todo el horizonte de control.

La función `sys()` toma una definición distinta aunque con los mismos parámetros de salida y entrada, según el modelo de predicción que se haya especificado en las directivas de preprocesamiento en C. Por ejemplo, en el caso de que se defina el sistema no lineal 1, la definición es la siguiente:

```
#ifdef SNL1

// ...

double * sys(double * u, double * yp, double * up, int hc, int hp){
    int ord_u = 1;
    double * yx = (double *)malloc((hp+ord_sys)*sizeof(double));
    double * ux = (double *)malloc((hp+ord_u)*sizeof(double));

    for(int i = 0; i < hp+ord_u; i++){
        if(i < ord_u){
            ux[i] = up[i];
        }else if(i < ord_u+hc){
            ux[i] = u[i - ord_u];
        }else {
            ux[i] = u[hc - 1];
        }
    }

    yx[0] = yp[0];
    for(int i = ord_sys - 1 ; i < hp+ord_sys - 1 ; i++){
        yx[i+1] = (ux[i+1]*ux[i+1]*ux[i+1] + yx[i]/(1 + yx[i]*yx[i]));
    }
    free(ux);
    return yx;
}
#endif
```

Código 7.9: función `sys()`

El parámetro de salida en todos los casos es un puntero de tipo `double` que apunta a una zona en memoria que contiene las predicciones del modelo SNL1 para todo el horizonte de predicción. En este caso, como se puede apreciar en el bucle `for` se emplea la ecuación matemática para rellenar `yx`. Las variables `ord_sys` y `ord_u` hacen referencia al número de valores pasados que hace falta en `y` y `u` correspondientemente.

Para la simulación del programa, es necesario un ejecutable de extensión `.mex` o `.mexw64` que se generará al compilar el código. La compilación del código fuente y la asignación de valores a los parámetros se realiza por medio del script de MATLAB 10.9.

Se expone a continuación el esquema de bloques en Simulink incorporando el *S-Function* y habiendo seleccionado el SNL1 para su control. Adicionalmente se muestran los sistemas SNL2 y SNL5 que debido a la intercambiabilidad del sistema se pueden introducir de fácil manera en el bucle de control. Aunque no es la práctica usual, la señal de referencia no se introduce como entrada al bloque *S-Function* sino como parámetro desde el editor de *MATLAB Workspace*. El sistema SNL2 (también perteneciente al grupo de sistemas no lineales del banco de pruebas) no expuesto anteriormente tiene la siguiente ecuación en diferencias:

$$y_2(k+1) = 0{,}8 \cdot u_2(k)^3 + y_2(k) \cdot y_2(k-1)/(1+y_2(k)^2); \tag{7.3}$$

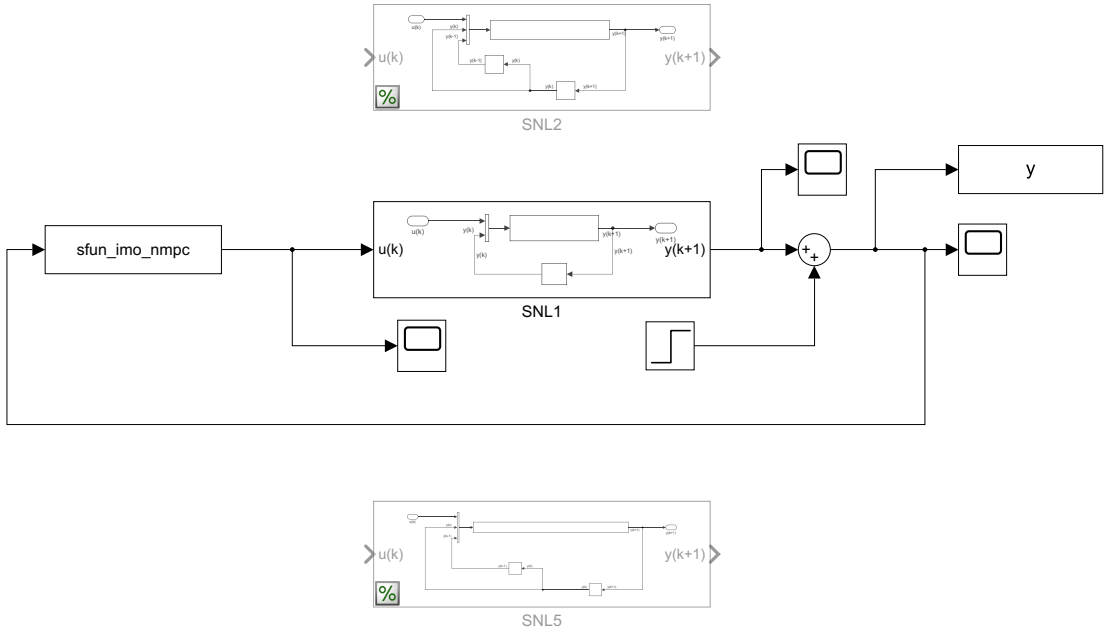


Figura 24: Diagrama de bloques Simulink del control del sistema no lineal 1 mediante el bloque *S-Function* iMO-NMPC

Se han realizado varias simulaciones intercambiando el sistema no lineal a controlar, el horizonte de control, el horizonte de predicción e introduciendo perturbaciones. Los parámetros del algoritmo genético se exponen a continuación; el tamaño de la población y número de iteraciones es 200; el número de objetivos es 2, el error de seguimiento y variación de la acción de control; se presentan las simulaciones de los diferentes sistemas partiendo de un horizonte de control y predicción de 2 hasta llegar a 6 en incrementos de 2. El máximo y mínimo para las variables reales es de [-4,4]. Los parámetros `pcross_real`, `pmut_real` corresponden a las probabilidades de *crossover* y mutación. De la misma manera, los parámetros `eta_c` y `eta_m` corresponden a los índices de distribución de probabilidad.

```
%% NSGA-II parameter definition
seed =      0.254;
popsize =   200;
ngen =      200;
nobj =      2;
ncon =      0;
% real variables/chromosomes
hc    =     2;
hp    =    2;
nout =      1;
nreal =     hc*nout; % hc
max_min_realvar = repmat([-4 4],[1 nreal]);
pcross_real =   0.65;
pmut_real =     0.5;
eta_c =         15;
eta_m =         25;
% binary variables/chromosomes
nbin =          0;
nbits =         0;
max_min_binvar = 0;
pcross_bin =    0;
pmut_bin =      0;

Tm = 0.1; % sfunction sampling time as well as simulation step time.
tf = 10;  % final time
```

Código 7.10: Lista de parámetros del NSGA-II para simulación (script 10.9)

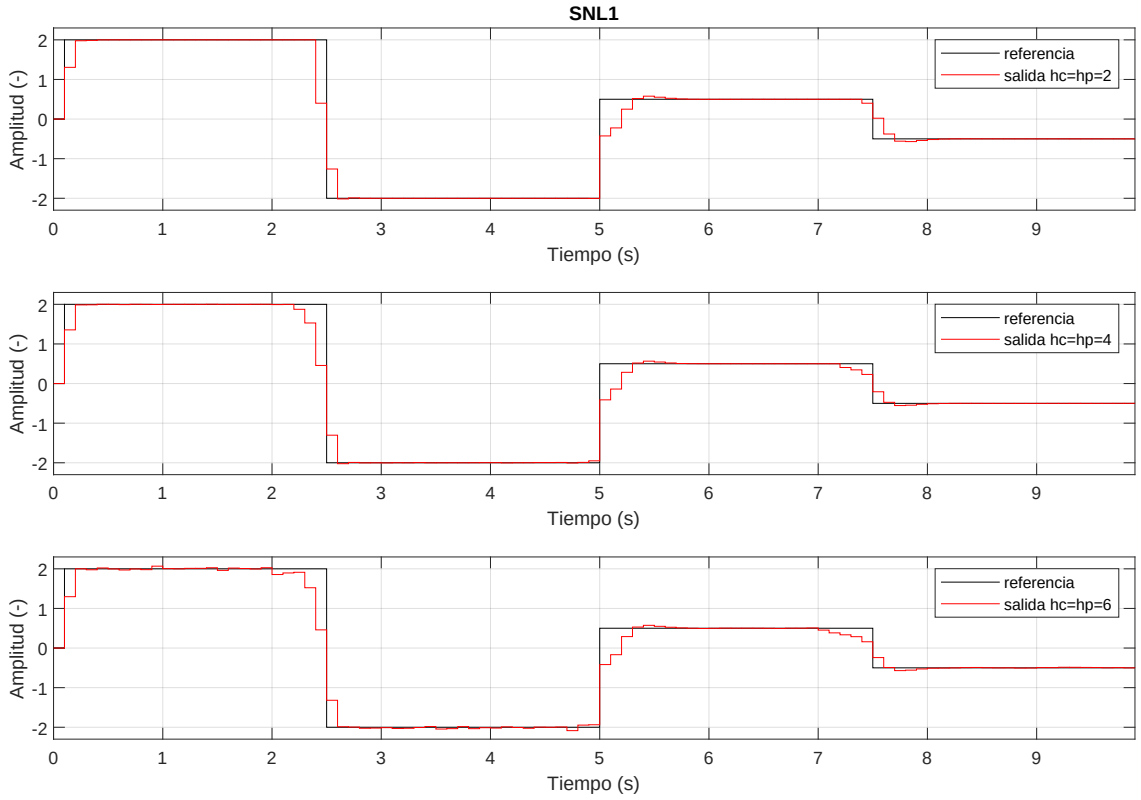


Figura 25: Evolución de la salida de SNL1 para hc=hp=2:2:6

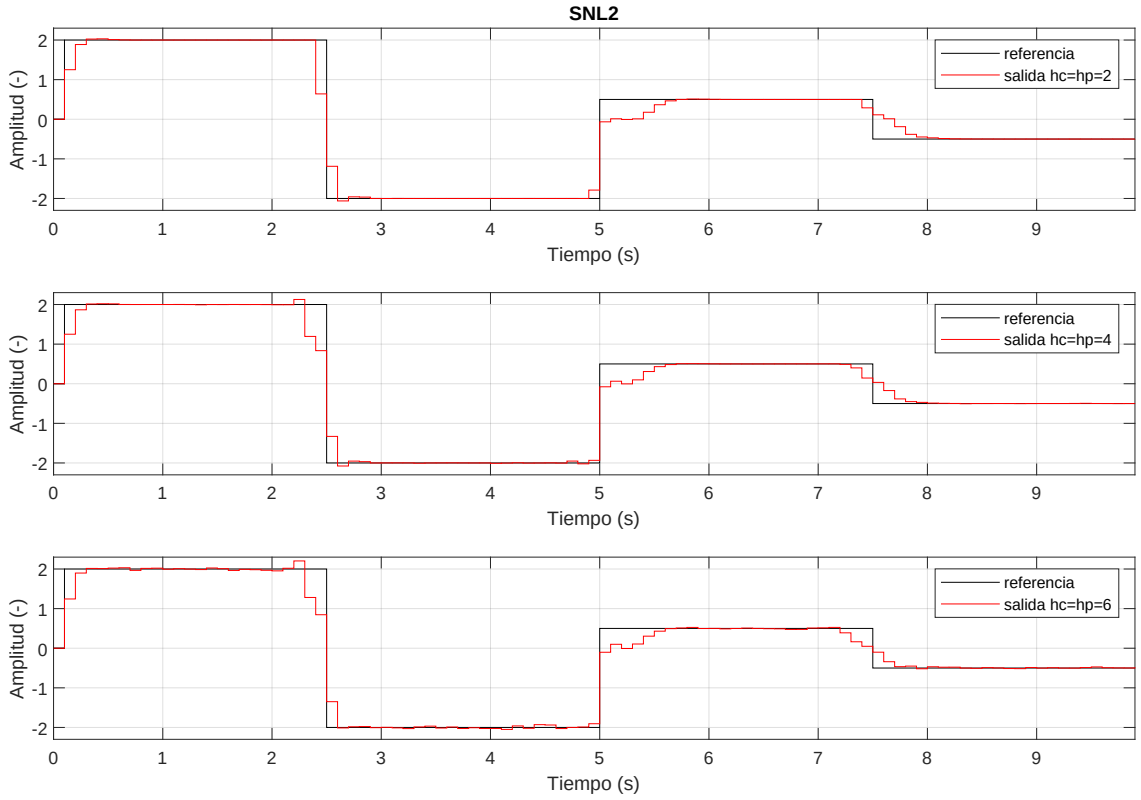


Figura 26: Evolución de la salida de SNL2 para hc=hp=2:2:6

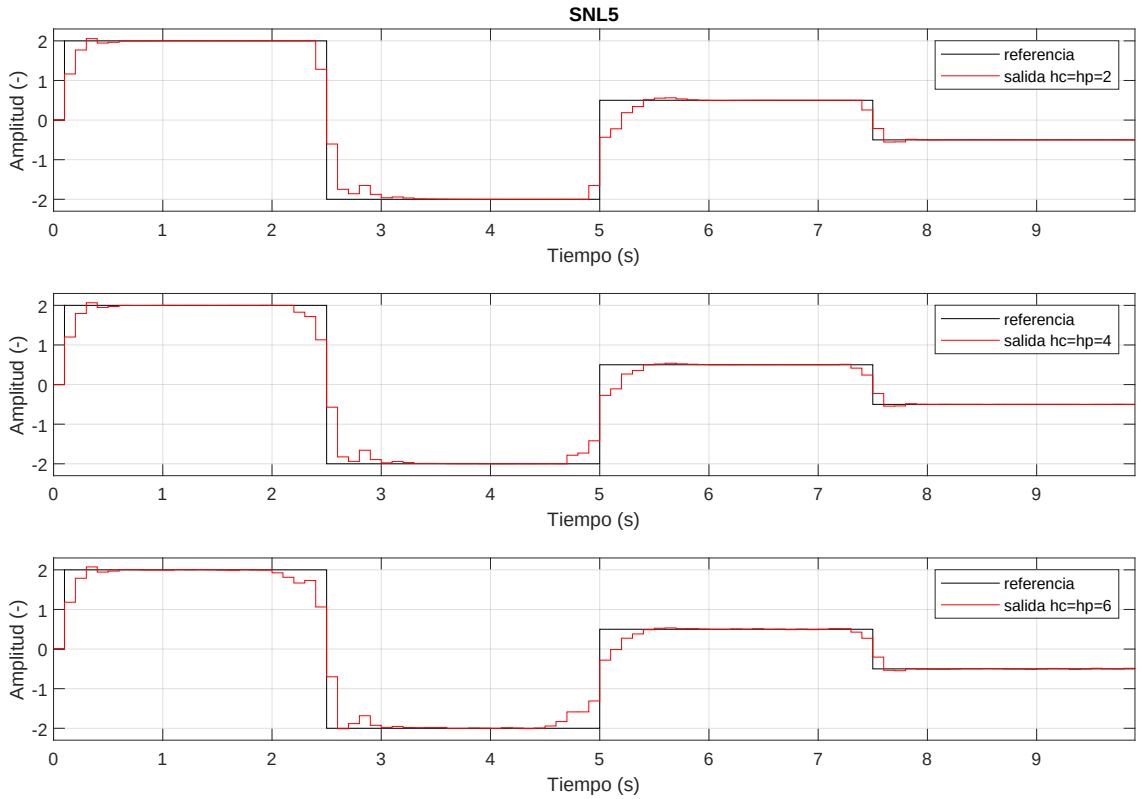


Figura 27: Evolución de la salida de SNL5 para hc=hp=2:2:6

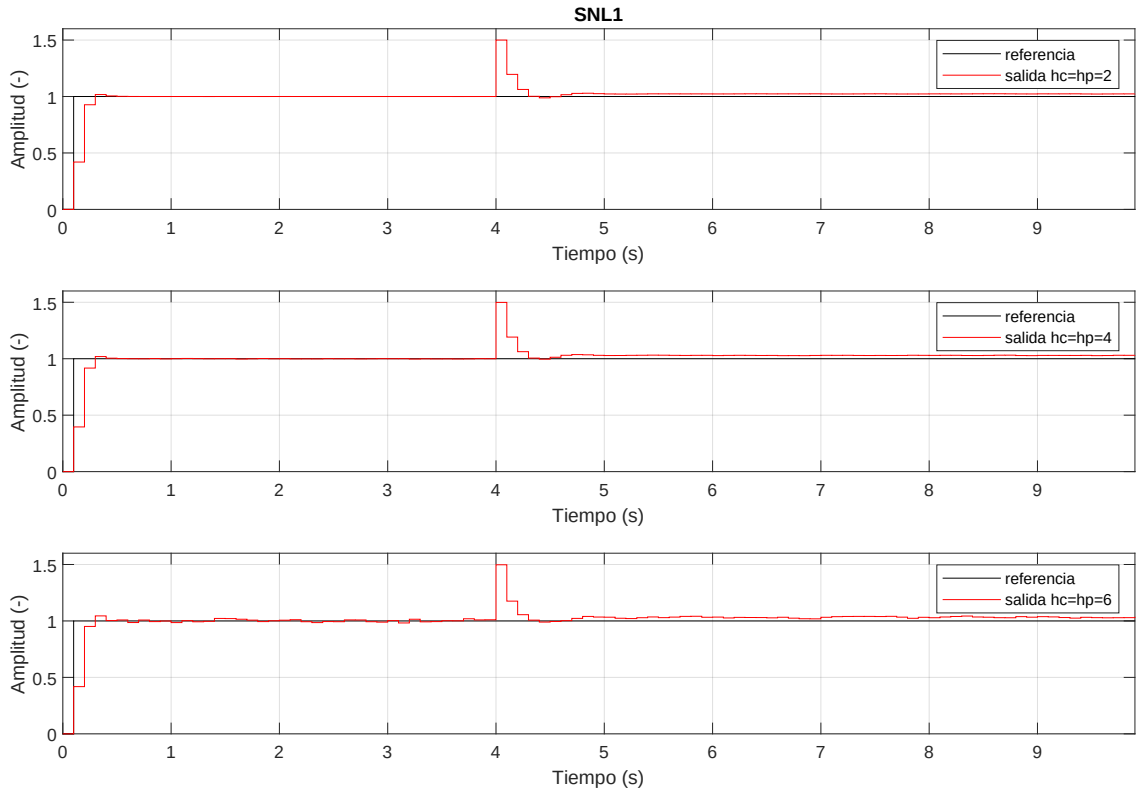


Figura 28: Evolución de la salida de SNL1 con perturbación para hc=hp=2:2:6

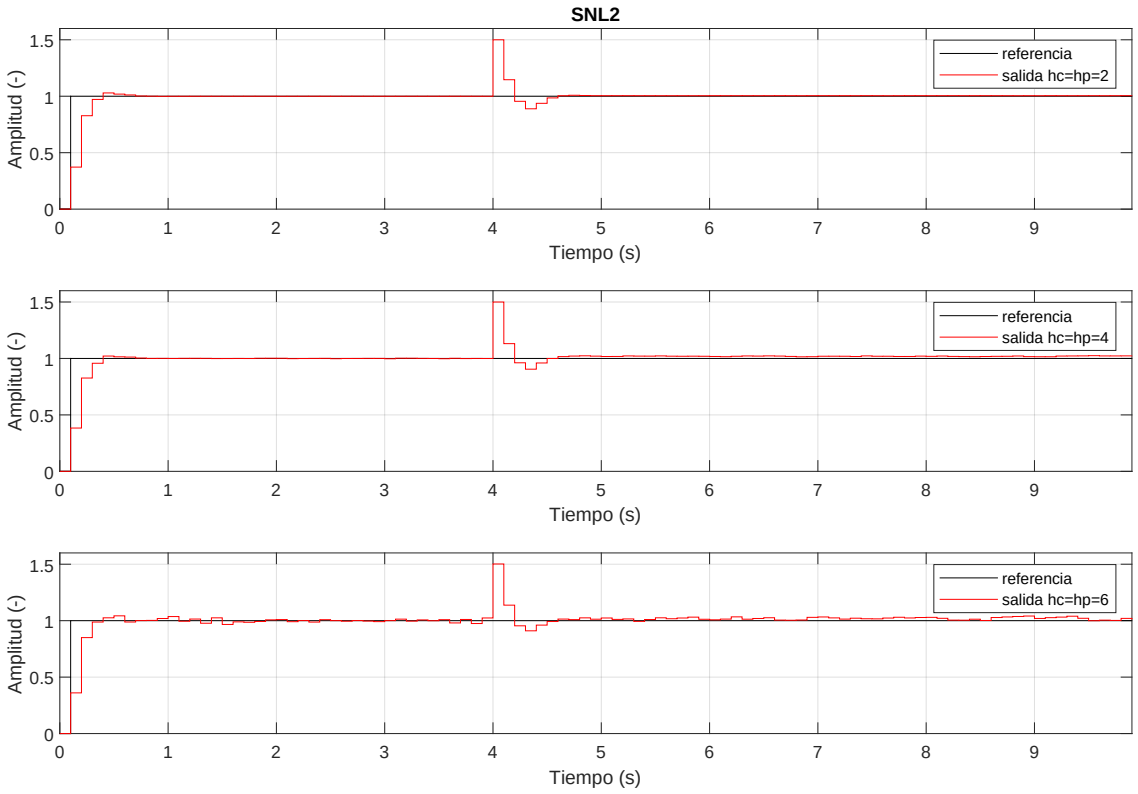


Figura 29: Evolución de la salida de SNL2 con perturbación para hc=hp=2:2:6

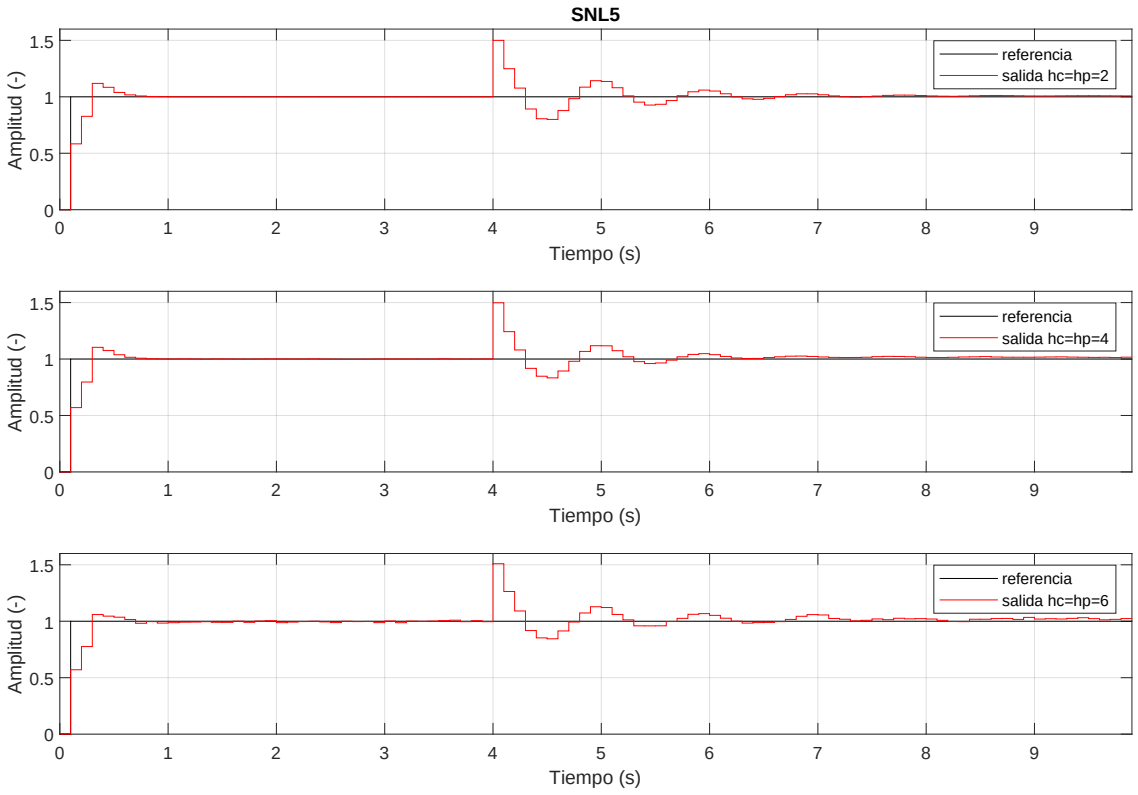


Figura 30: Evolución de la salida de SNL5 con perturbación para hc=hp=2:2:6

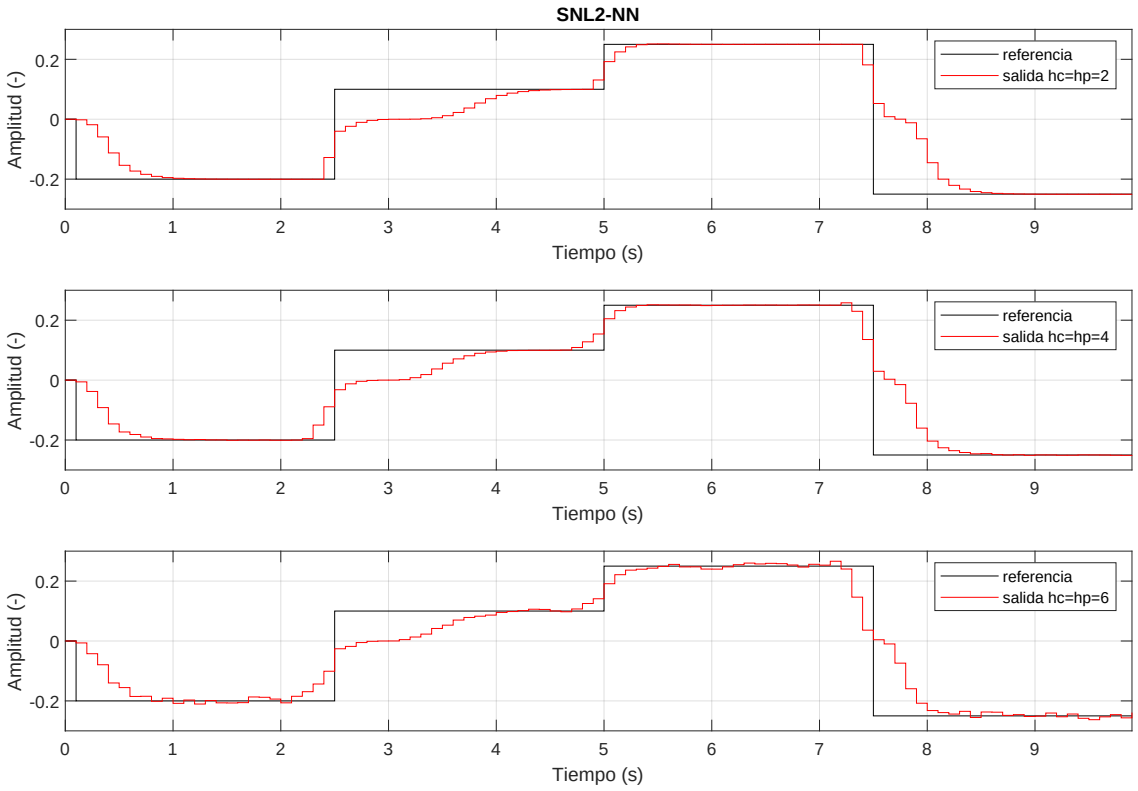


Figura 31: Evolución de la salida de SNL2-NN para hc=hp=2:2:6

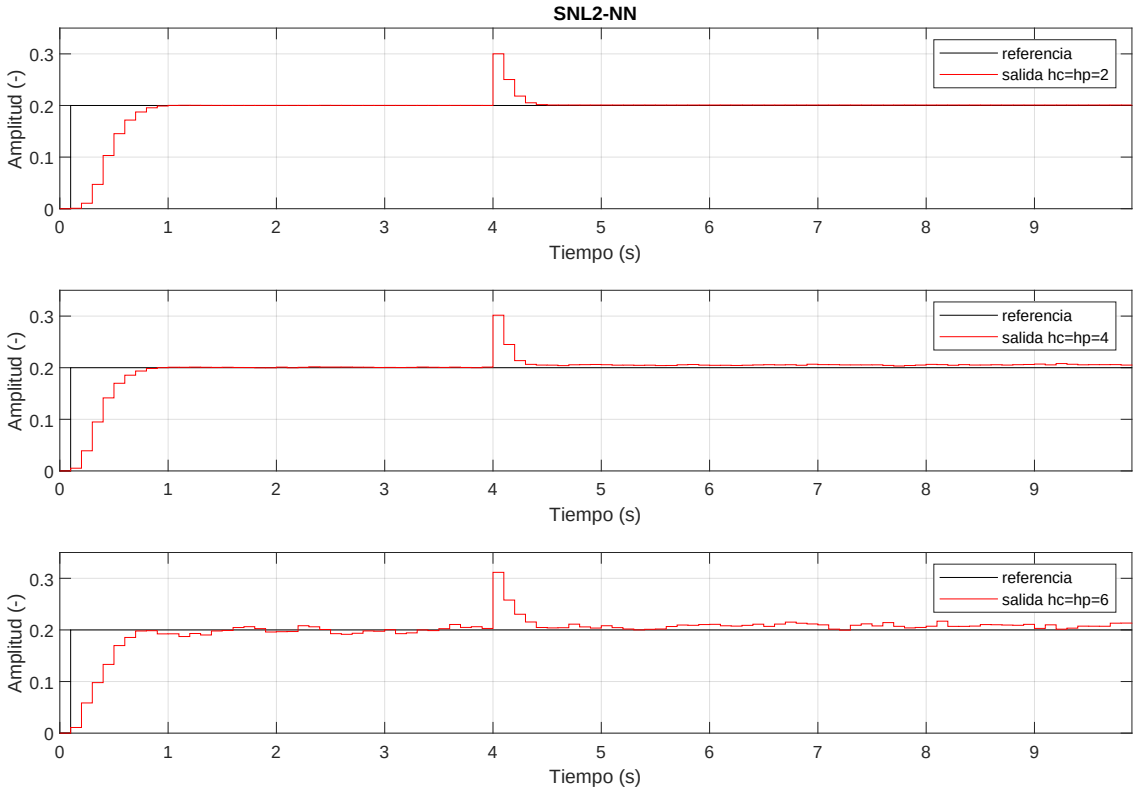


Figura 32: Evolución de la salida de SNL2-NN con perturbación para hc=hp=2:2:6

Exceptuando las dos últimas gráficas correspondientes a un modelo neuronal limitado, en el resto de casos con independencia del horizonte de control y predicción, la salida llega sin problemas a la referencia. La deterioración del seguimiento a medida que se va aumentando el horizonte se debe al incremento del coste computacional causado por el incremento de cromosomas por individuo. Es por eso que debido a que el número máximo de iteraciones para todos los rangos es el mismo, el resultado se va deteriorando para mayores horizontes ya que no se llega tan cerca del óptimo. Adicionalmente, para los casos de $H_c = H_p = 4$ y $H_c = H_p = 6$, se puede apreciar la antelación de la salida antes del cambio de referencia. El error cuadrático medio y error máximo absoluto de dichas simulaciones se encuentra en la tabla 12 del anexo I.

En la figura 31 se muestra la evolución del sistema SNL2 habiendo empleado un modelo de predicción neuronal. El comportamiento es pobre, aún teniendo en cuenta que se ha asignado una referencia para su rango de operación [-0.5,0.5], lo cual demuestra la importancia de la necesidad de un buen modelo de predicción. Esta primera red sirve como validación del método empleado para introducir redes entrenadas en MATLAB. Se escoge una arquitectura personalizada partiendo de la plantilla de redes neuronales de MATLAB por defecto. Esta arquitectura reducida carece de funciones de entrada y salida de normalización, esto se hace para facilitar la obtención de pesos directamente. Después, estos pesos se codifican de manera fija en el código en C dentro de la función `SNL2_ff()` (archivo `SNL2_NN.c` en Github) que solo realizará multiplicaciones matriciales y aplicará la función de activación *sigmoide*. Esta metodología aunque rudimentaria y preliminar sirve para validar la estrategia. Se anota el apartado de inclusión de modelos neuronales dentro de futuras mejoras en el código.

Las gráficas 28, 29, 30 muestran la evolución de los tres sistemas ante una perturbación de 0.5 en la salida. Se observa que se rechazan rápidamente las perturbaciones con excepción del sistema SNL5 que muestra bastantes oscilaciones, posiblemente a raíz de la naturaleza oscilante de su ecuación. En cuanto a la figura 32, se muestra también la evolución del sistema SNL2 empleando como modelo de predicción el modelo neuronal NARX. En este caso a excepción de $H_c = H_p = 6$, las simulaciones para el resto de horizontes rechazan la perturbación y siguen la referencia con un error mínimo. Igualmente, los errores cuadráticos medios y errores máximos absolutos en el caso con perturbaciones esta resumida en la tabla 13 del anexo I.

7.4. Extensión a MIMO

Las gráficas y los resultados expuestos hasta ahora corresponden al comportamiento del sistema ante la estrategia de control predictivo inteligente con un modelo de predicción matemático para SISO (a excepción de SNL2_NN que es neuronal). El siguiente paso lógico ha sido extender el código desarrollado para que soporte sistemas con múltiples entradas y salidas, es decir MIMO. Para ello se identifican las entradas y salidas que tendrá el programa para la interacción con Simulink. Se enumeran mediante el parámetro `nin` el número de entradas al bloque *S-Function*, estas serán las salidas ($y_1, y_2, ...$) que tendrá el sistema. El parámetro `nout`, a su vez, hace referencia al número de salidas del bloque que serán las variables de control ($u_1, u_2, ...$). Ambos de estos parámetros se asignan antes de la ejecución por medio del script de parámetros. Por lo tanto, para los sistemas MIMO 2x2 que se van a construir partiendo de los anteriores sistemas no lineales (SNL1,SNL2,SNL5...), `nin` será 2 y `nout` 2.

La estructura de datos que se ha empleado en MIMO para las soluciones propuestas, no cambia; sigue siendo un *array* unidimensional de tipo `double`, aunque en lugar de `hc` celdas ahora tendrá `nout*hc`. De esta manera, cada individuo contiene la secuencia de acciones a aplicar en cada variable de control para controlar el sistema. Para un ejemplo con **tres variables de control**, se expone a continuación el esquema de la población:

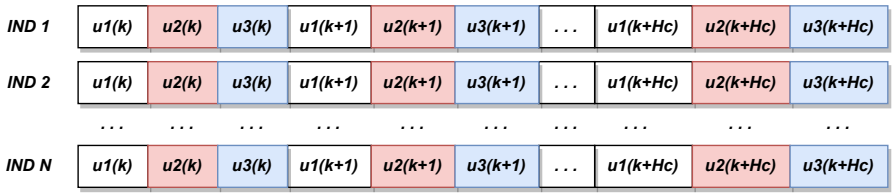


Figura 33: Disposición de las acciones de control por celdas e individuo en MIMO

Lo parte que se ha cambiado es el acceso a las celdas. Se expone el método en el siguiente fragmento de código en C:

```
for(int i = 0; i < rows; i++){
    for(int j = 0; j < columns; j++){
        printf(" array[i][j]: %lf ",array[i + rows*j]);
    }
}
```

En el caso de la figura 33 se tendrían tres filas y `hc` columnas por individuo almacenadas en un *array* unidimensional. Igualmente, el vector de salidas empleado con los modelos de predicción matemáticos multivariables también se accederá de la misma manera. Las nuevas combinaciones de modelos (e.g SNL1-SNL1, SNL1-SNL5 ...) se incorporan también en el archivo `model.c` y se selecciona uno u otro según las directivas de preprocesamiento al compilar.

Es necesario también modificar la función `test_problem()` que asigna el coste a cada solución, ahora para los modelos 2x2, asignará a cuatro objetivos (dos referentes al coste del error y otros dos al coste energético). Por la misma lógica, es necesario añadir una nueva definición de la función `decision_maker()` para tener en cuenta cuatro objetivos, y partiendo de ahí seleccionar dos individuos cuyas acciones de control minimicen mejor el coste del error y energía de cada uno de los sistemas no lineales que componen el MIMO 2x2. [5]

[5] El código que contiene los cambios realizados para la extensión a MIMO se encuentran en el *branch* `mimo` de repositorio de github: `https://github.com/GICI-UPV-EHU/sfun_imo_nmpc/tree/mimo`

El diagrama de bloques en simulink (para el SNL1-SNL5) es el siguiente, siendo el bloque central un subsistema que contiene la combinación de los sistemas no lineales a controlar:

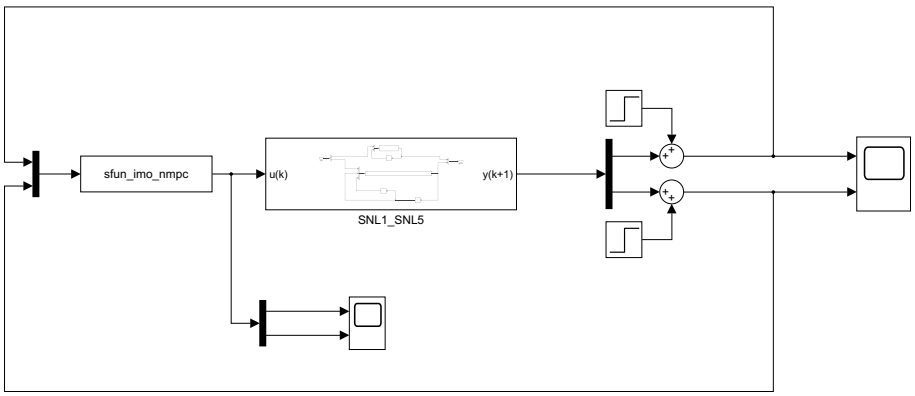


Figura 34: Diagrama de bloques Simulink del control del sistema no lineal combinado SNL1-SNL1

Se exponen a continuación las simulaciones de los sistemas combinados SNL1-SNL1 y SNL1-SNL5 para diferentes horizontes de control y predicción. Los parámetros empleados esta vez son los siguientes:

```
%% NSGA-II parameter definition
seed =      0.254;
popsize =   200;
ngen =      400;
nobj =      4;
ncon =      0;
% real variables/chromosomes
hc    =     2;
hp    =    2;
nout =      2;
nin  =      2;
nreal =     hc*nout; % hc
max_min_realvar = repmat([-4 4],[1 nreal]);
pcross_real =   0.65;
pmut_real =     0.5;
eta_c =         15;
eta_m =         25;
% binary variables/chromosomes
nbin =          0;
nbits =         0;
max_min_binvar = 0;
pcross_bin =    0;
pmut_bin =      0;

Tm = 0.1; % sfunction sampling time as well as simulation step time.
tf = 10;  % final time
```

Código 7.11: Lista de parámetros del NSGA-II para simulación en MIMO

El principal cambio que se puede observar es que en este caso se ha empleado un número de población de 200 y un número de iteraciones de 400. Esto se debe a que en este enfoque, para un sistema MIMO con las variables reales (`nreal = hc*nout`) siendo múltiplos del número de salidas del bloque, se requieren más iteraciones que en SISO, al empezar de base con un número mayor de cromosomas por individuo. En el caso particular para la primera simulación de `hc = 2` y `nout = 2`, el número de cromosomas por individuo es 4. Posteriormente, cuando `hc = 4`, `nreal` será 8 y cuando `hc = 6`, `nreal = 12`. Es por ello que para dotar al AG de suficiente cómputo con el objetivo de mantener un control mínimamente satisfactorio para horizontes cortos, se aumenta el número de iteraciones.

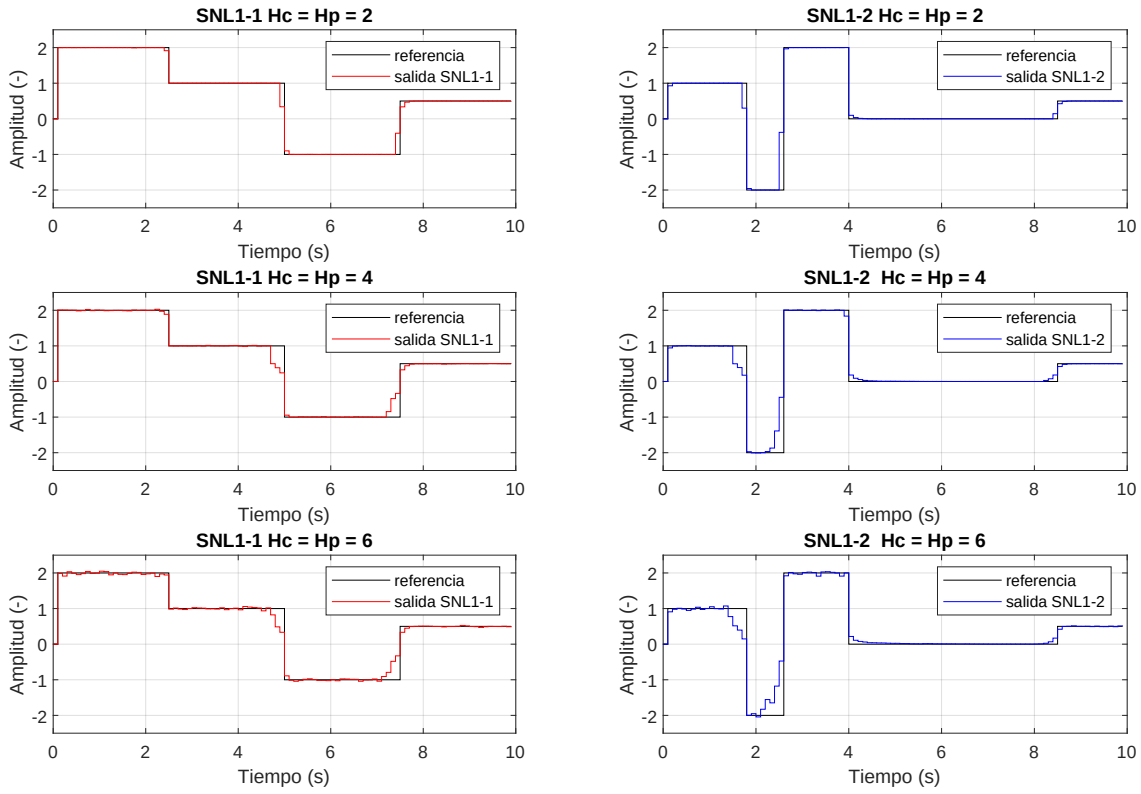


Figura 35: Evolución del sistema SNL1-SNL1 ante diferentes horizontes

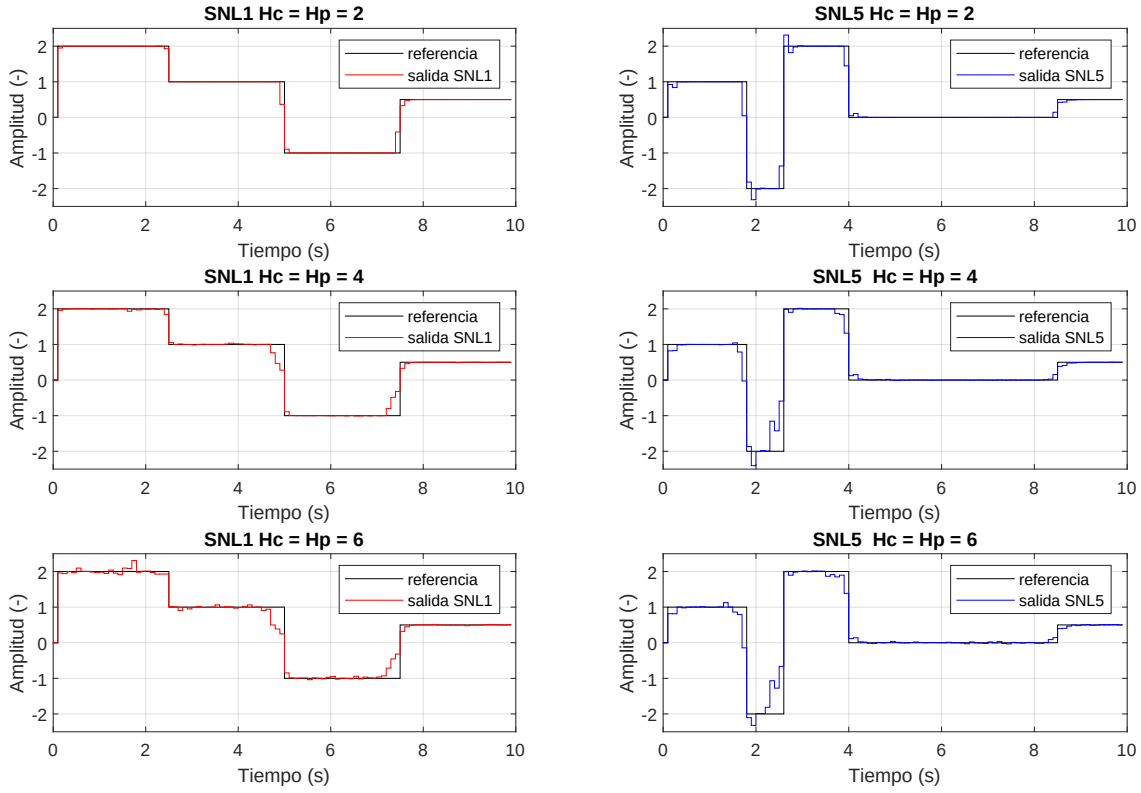


Figura 36: Evolución del sistema SNL1-SNL5 ante diferentes horizontes

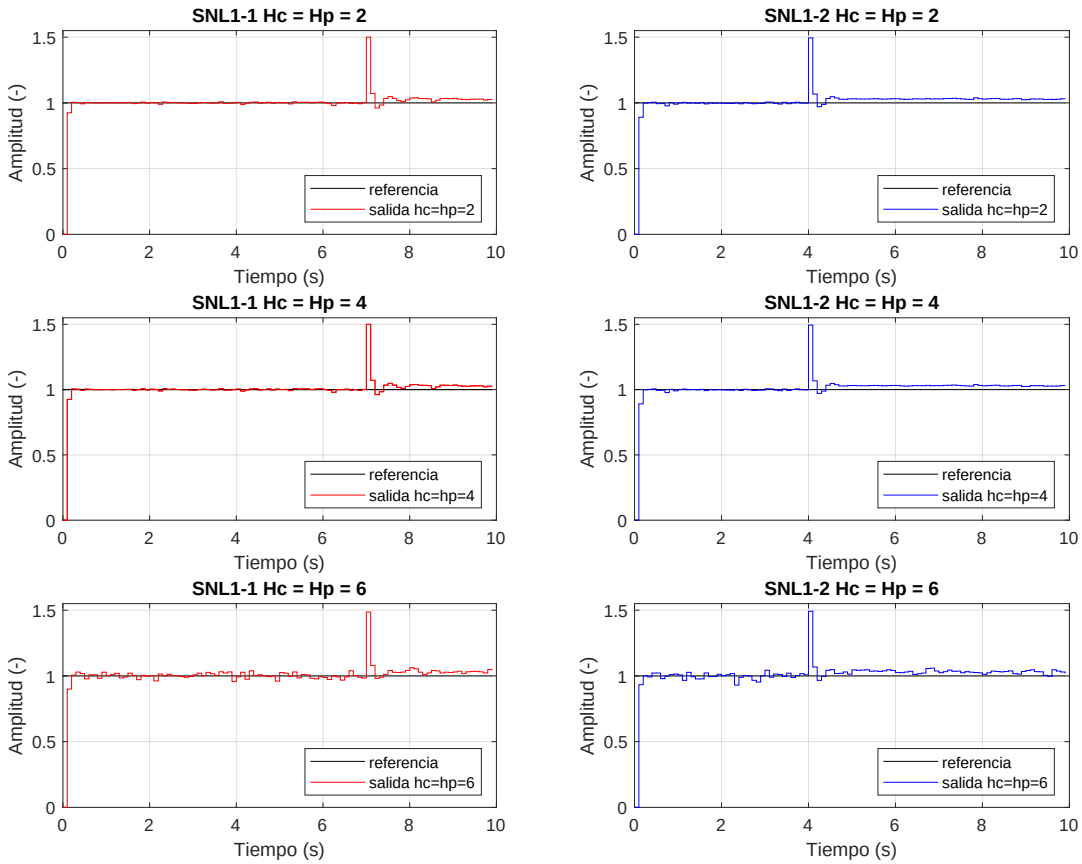


Figura 37: SNL1-SNL1 perturbación de 0.5 en diferentes instantes, hc=hp=2:4:6

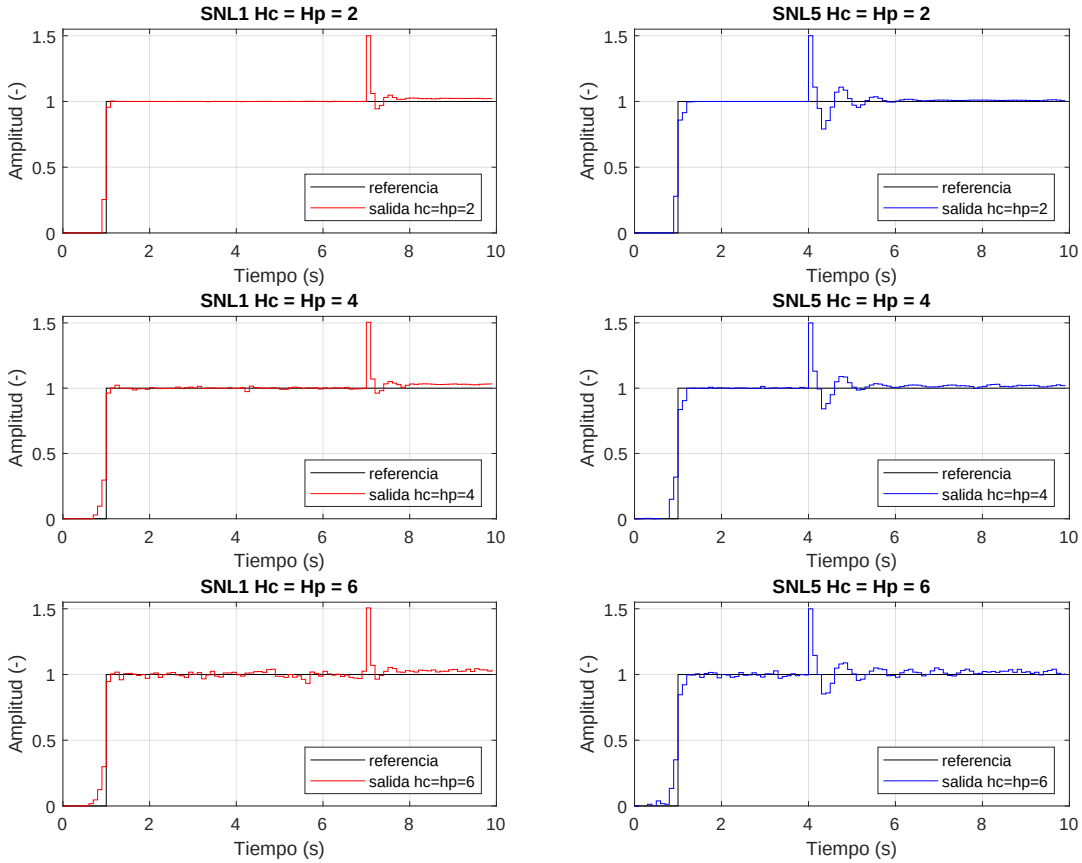


Figura 38: SNL1-SNL5 perturbación de 0.5 en diferentes instantes, hc=hp=2:4:6

Al igual que en las simulaciones de SISO, se puede apreciar un deterioro en la calidad de control al incrementar los instantes del horizonte de control, manteniendo constante el número de individuos de la población y las iteraciones. Asimismo, se puede observar la antelación presente de la salida, sobretodo para el horizonte de 6 instantes. Se presentan los errores cuadráticos medios y errores máximos absolutos en la tabla 14 del anexo I.

En cuanto a los gráficos mostrando el rechazo de una perturbación de 0.5, se da en todos el mismo fenómeno: hacia el final del rechazo de la perturbación permanece un error residual, que no permite a la salida alcanzar la referencia. Este error remanente, no obstante, desaparece al cambiar la ponderación de los objetivos.

En la manera que esta codificada,la toma de decisiones (*Decision Maker*) para este caso, la finalidad es obtener dos índices que corresponderán a los dos mejores individuos de la población que minimicen los costes definidos:

$$C_1 = \lambda_{11} \cdot \sum_{i=1}^{hp}((ref_1(k+i) - y_1(k+i)) + e_1)^2 + \lambda_{12} \cdot \sum_{j=1}^{hc} \Delta u_{1j}^2$$
$$C_2 = \lambda_{21} \cdot \sum_{i=1}^{hp}((ref_2(k+i) - y_2(k+i)) + e_2)^2 + \lambda_{22} \cdot \sum_{j=1}^{hc} \Delta u_{2j}^2$$

Para estas simulaciones la ponderación es la misma para todos los objetivos en el coste:

$$\lambda_{ij} = 1 \quad \forall i, j| \quad i \in [1, N_{cost}], j \in [1, N_{obj}]$$

Si no se penaliza el coste energético, es decir, la variación de la acción de control, para ninguno de los dos costes, en otras palabras, $\lambda_{12}, \lambda_{22} = 0$; se consigue eliminar el error residual al final del rechazo, tal y como se puede observar en la figura 39. De esta forma, se concluye que ese error residual es el resultante de una situación donde el incremento de la variación de la acción de control a cambio de una reducción de error no minimiza el coste.

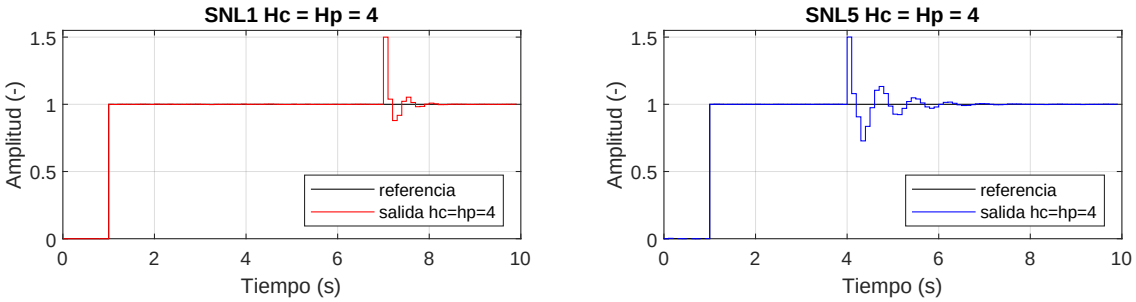


Figura 39: SNL1-SNL5 hc=hp=4, ponderación del coste energético nula

Finalmente, otro aspecto importante a comentar, es la diferencia en la rapidez del rechazo y las oscilaciones. Por un lado, en el primer sistema combinado SNL1-SNL1, el comportamiento ante perturbaciones es idéntico pero desplazado en el tiempo. Por otro lado, la evolución del rechazo en el SNL1 es mucho más rápida y amortiguada que el del SNL5, donde adicionalmente, para el caso de un horizonte de 6 instantes se aumentan aún más dichas oscilaciones. Como es de esperar, este comportamiento se corresponde con la naturaleza de cada uno de los sistemas. La tabla correspondiente a los errores cuadráticos medios y el máximo absoluto tanto para las simulaciones normales como para el ejemplo con ponderación del coste energético nulo es el número 15 del anexo I.

7.4.1. Adición de una temporización del algoritmo genético

Antes de seguir con estudio de alternativas para la ejecución en tiempo real, se implemento un ingenuo mecanismo de temporización para el AG. La lógica de este mecanismo es muy simple:

1. Definir una variable de tiempo límite y asignarle un valor de máximo tiempo permisible para el AG (`trigger`).
2. Antes de entrar al bucle de iteración, se guarda el valor del reloj en otra variable (`before`).
3. Una vez en el bucle se calcula la diferencia entre el tiempo en ese instante y la variable `before` para obtener el tiempo transcurrido (`difference`).
4. se realiza una conversión de unidades a milisegundos (`msec`) para compararlo con el tiempo límite.
5. Finalmente, si el tiempo transcurrido es superior al límite o el algoritmo genético ha alcanzado el máximo de generaciones, se sale del bucle.

A continuación se presenta el fragmento de código implementa dicha lógica:

```
#include "time.h"
 // ...
int n;
int msec = 0, trigger = 100; // 0.1 segundos
clock_t before;
    // ...
       before = clock();

      initialize_pop(parent_pop);
      decode_pop(parent_pop);
      evaluate_pop(parent_pop);
      assign_rank_and_crowding_distance(parent_pop);
      n = 2;
      do {
          selection(parent_pop, child_pop);
          mutation_pop(child_pop);
          decode_pop(child_pop);
          evaluate_pop(child_pop);
          merge(parent_pop, child_pop, mixed_pop);
          fill_nondominated_sort(mixed_pop, parent_pop);
          n++;
          clock_t difference = clock() - before;
          msec = difference*1000/CLOCKS_PER_SEC;
      } while((msec < trigger) || (n <= ngen));
      decision_maker(parent_pop,idx);
      // ...

```

Código 7.12: mecanismo de temporización en C

Esta implementación aunque simple, no funciona para simulaciones en modo externo. Como se expondrá en el siguiente capítulo, el modo externo se corresponde a una simulación en *Simulink Desktop Real-Time (Kernel Mode)* o *Simulink Real-Time*. Resulta que la librería `"time.h"`, empleada para las variables de tipo `clock_t`, la función `clock()` y macros como `CLOCKS_PER_SEC`, no es nativa en modo externo y requiere la entrega de sus binarios o archivo fuente para compilarlo, los cuales no se encontraron. Es cierto que *Simulink*, también cuenta con librerías en C para gestión del tiempo, pero debido al tiempo limitado y la poca documentación que se encontró, finalmente se dejó para desarrollos futuros.

7.5. Estudio de alternativas para la ejecución en tiempo real

Existen varias formas que ofrece MATLAB/Simulink®para ejecutar una simulación con garantías de tiempo real (o por lo menos *soft real time*). Entre las que se contemplan están:

- **Simulink Desktop Real-Time**: " Simulink Desktop Real-Time™ proporciona un kernel en tiempo real para ejecutar modelos de Simulink® en un ordenador portátil o de sobremesa con Windows® o macOS®. Incluye bloques de librerías que se conectan a determinados dispositivos de E/S. Puede crear un sistema en tiempo real en Simulink con su PC o Mac y conectarlo a dispositivos físicos. " [47].
- **Simulink Real-Time**: "Simulink Real-Time™ y Speedgoat permiten pasar de la simulación al prototipado rápido de sistemas de control (RCP) y las pruebas de hardware-in-the-loop (HiL) con un solo clic." - " Puede crear, controlar e instrumentar aplicaciones en tiempo real que se ejecuten en plataformas de destino en tiempo real Speedgoat, directamente desde el modelo de Simulink o con la API de MATLAB y App Designer. " [50] .
- **Simulink PLC Coder**: "Simulink PLC Coder™ genera texto estructurado IEC 61131-3 independiente del hardware y diagramas en escalera a partir de modelos de Simulink®, gráficas de Stateflow® y funciones de MATLAB®. El texto estructurado se genera en PLCopen XML y otros formatos de archivo soportados mediante entornos de desarrollo integrados (IDEs) de uso habitual, incluidos 3S-Smart Software Solutions CODESYS®, Rockwell Automation Studio 5000, Siemens TIA Portal y Omron® Sysmac® Studio." - "En consecuencia, puede compilar e implementar su aplicación en numerosos dispositivos de controlador lógico programable (PLC) y controlador de automatización programable (PAC)." [49].

Se decide por las dos primeras alternativas que resultan más accesibles. Asimismo, debido a que en el pasado ya se hicieron pruebas en la versión anterior de *Simulink Real-Time*, antes denominado *xPC Target*, ya se es familiar con el flujo de trabajo y la tarea de aprendizaje es menos ardua.

7.5.1. Pruebas en Simulink Desktop Real-Time

Tal y como se menciona en la página oficial, gracias a Simulink Desktop Real-Time el usuario podrá ejecutar modelos Simulink en un ordenador Windows® o macOS®. No obstante, debido a que ninguno de estos sistemas operativos posee un kernel de tiempo real, este método no puede ofrecer garantías de tiempo real al 100 %. Aún así, permite una forma fácil y rápida para comprobar el funcionamiento del programa desarrollado en el kernel externo sin necesidad de *hardware* adicional. Se expone a continuación los pasos seguidos para la configuración del entorno para una ejecución en modo Kernel.

Dentro de desplegable de aplicaciones (figura 40) se encuentra el modo de ejecución Simulink Desktop Real-Time al igual que el resto de alternativas ya mencionadas.

Una vez dentro del entorno existen dos modos (figura 41):

- **Connected IO Mode** [45]: Este modo de ejecución es una ampliación del modo normal para actualizar las salidas en tiempo real. Normalmente el algoritmo de simulación que no es de tiempo real se ejecuta completamente dentro de Simulink. La opción del *solver*

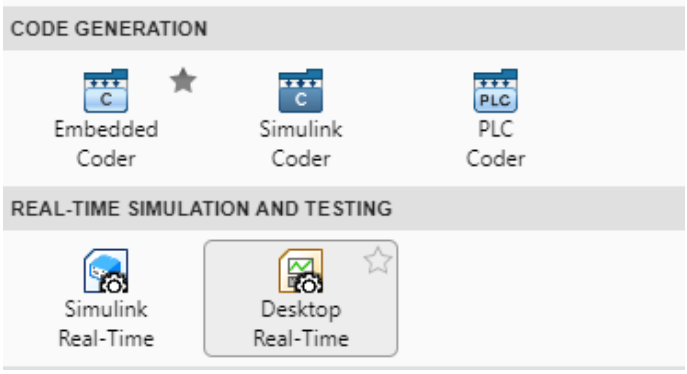


Figura 40: Desplegable de aplicaciones en Simulink

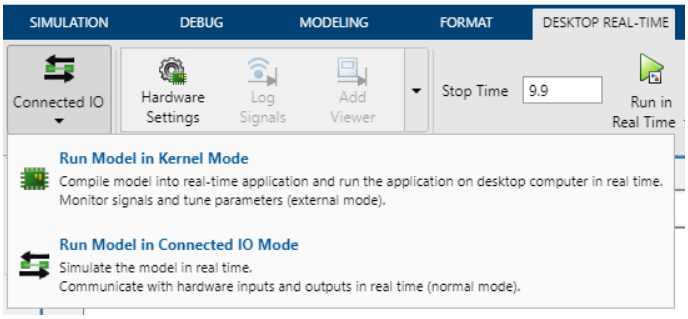


Figura 41: Desplegable de aplicaciones en Simulink

puede ser de paso fijo o paso variable, y el modelo se ejecuta en la medida posible en presencia del resto de procesos del sistema operativo que compiten por el cómputo. En el modo *Connected IO* se puede sincronizar el modelo Simulink con un reloj de tiempo real mediante los bloques E/S. En este modo, Simulink ejecuta el algoritmo de simulación, mientras que un proceso independiente del modo *Kernel* del sistema operativo ejecuta los controladores E/S. El proceso de Simulink y el proceso del modo *Kernel* se ejecutan en una *host machine* (máquina huésped) mediante una interfaz de memoria compartida para transferir datos de los parámetros.

En el modo de *Connected IO*, en cada intervalo de muestra, Simulink evalúa cada bloque en tiempo real. Simulink escribe los datos de entrada en un búfer que pasa al proceso de modo kernel. El proceso del modo kernel propaga los datos al hardware, donde se escribirán los datos de respuesta en otro búfer. En el próximo tic de tiempo, Simulink leerá los datos de respuesta y los propagará al resto del modelo.

- **Kernel Mode** [46]: El modo Run in Kernel es una alternativa de mayor rendimiento a la ejecución en tiempo real en el modo *Connected IO*. En el modo *Run in Kernel*, se utiliza Simulink ® Coder™ para vincular el código del algoritmo generado al código E/S generado por los bloques de E/S. El ejecutable resultante se ejecuta en el *Kernel* del sistema operativo del ordenador donde se esta desarrollando.

  El ejecutable del modo *Kernel* está completamente sincronizado con el reloj en tiempo real. La función del proceso Simulink se limita a leer y mostrar los resultados de la simulación devueltos por el ejecutable. Empleando los drivers de E/S para comunicarse con el hardware, la aplicación almacena los datos de salida en una memoria accesible por Simulink hasta que se llena el búfer de datos. Una vez, este búfer esta lleno, la aplicación de tiempo real sigue ejecutándose mientras que el proceso de Simulink se encarga de transferir los datos al entorno de MATLAB. Esta transferencia de datos es menos crítica que la manutención de actualizaciones deterministas en tiempo real dentro del intervalo de muestreo requerido.

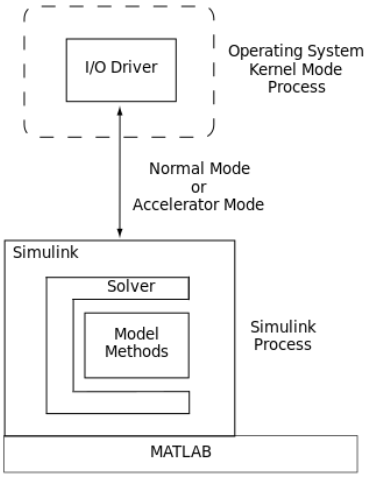


Figura 42: *Connected IO Mode* [45]

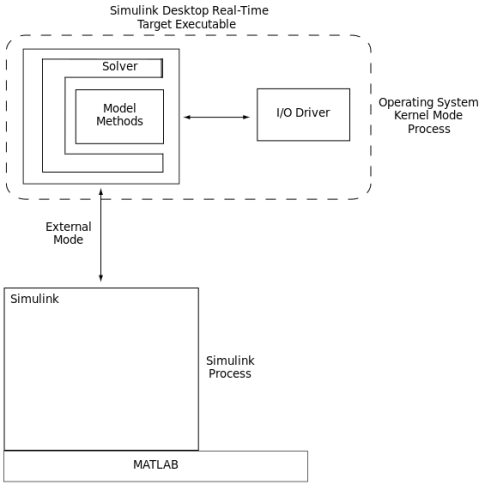


Figura 43: *Kernel Mode* [46]

Debido a que en el punto del desarrollo no hace falta dispositivos de entrada salida externos, las pruebas se centran en el comportamiento del modo *Kernel* donde el algoritmo desarrollado se ejecuta como proceso del *kernel* del sistema operativo.

Para la configuración de la simulación se deben realizar los siguientes pasos:

1. Contar con el archivo `mexw64` (en caso de Windows) y añadir la carpeta donde se encuentra a los directorios de trabajo de MATLAB. Para ello, de igual manera que se ha especificado anteriormente, se emplea el *script* `compile.m` que llama a la función "`mex`" para generar dicho ejecutable.

2. Establecer el *solver* de Simulink a pasos fijos discretos (figura 44).

Figura 44: Paso 2: Establecer el solver de Simulink a pasos fijos discretos.

3. En la pestaña de *Simulation Target* de la configuración de la simulación, se incluyen los *headers* y los archivos fuente tal y como se muestra en la figura 45.

Figura 45: Paso 3: Añadir *headers* y archivos fuente.

4. En la pestaña *Code generation* seleccionar el *System target file* correspondiente a *Simulink Desktop Real-Time*, es decir, `sldrt.tlc` (figura 46).

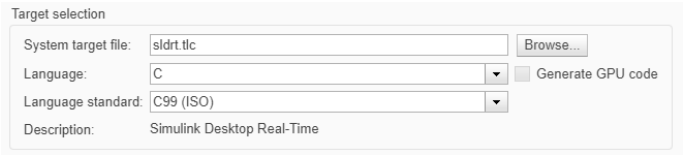


Figura 46: Paso 4: Seleccionar *System target file*.

5. Finalmente dentro de *Code generation* en la pestaña *Custom code* basta con marcar la casilla donde dice "*Use the same custom code settings as Simulation Target*" (Emplear mismos ajustes para el código personalizado que en el objetivo de simulación) (figura 47).

Figura 47: Paso 5: marcar la casilla en *Custom Code*.

Exceptuando un pequeño cambio, la ejecución del código fue correcta en las pruebas con los diferentes modelos, horizontes y resto de parámetros. La única modificación que se tuvo que realizar para que dejara saltar un error que impedía la ejecución, fue la inicialización de las variables `flag_real` y `flag_bin`. Según el orden cronológico expuesto en la documentación (figuras 18, 19), la función `mdlInitializeSizes()` opcionalmente llama a `mdlCheckParameters()`, fragmento donde se comprueban los parámetros y se inicializan ambas variables mencionadas anteriormente según lo especificado en el *script* `compile.m` que se ejecuta en antelación.

En la llamada `mdlInitializeSizes()`, como se muestra en el fragmento de código 7.13, se especifica dicha llamada en condición de que el archivo entero se compile como un archivo *MEX*. No obstante, aunque ejecutándolo en modo normal no saltaba errores, en modo *kernel* externo saltaba un error que bloqueaba el programa. Resulta que el fragmento de asignación de memoria para las variables reales (fragmento de código 7.14) no se ejecutaba ya que la variable `flag_real` no había sido inicializada; lo que causaba que en un paso posterior, al intentar leer en esa zona de memoria saltará un error.

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 21);  /* Number of expected parameters */
    #if defined(MATLAB_MEX_FILE)
        if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
            mdlCheckParameters(S);
            if (ssGetErrorStatus(S) != NULL){
                return;
            }
        }else {
            return; /* Parameter mismatch will be reported by Simulink */
        }
    #endif
// ...
```

Código 7.13: fragmento de código de la llamada a mdlCheckParameters dentro de mdlInitializeSizes

Al no producirse la llamada a `mdlCheckParameters` en el orden creído, no se llegaba a dar un valor a `flag_real` o `flag_bin`. Este cambio en el orden de la simulación esta documentada en la página de MathWorks [48]:

```
    if(flag_real){
      min_realvar = (double *)malloc(nreal*sizeof(double));
      max_realvar = (double *)malloc(nreal*sizeof(double));
      for(int i = 0; i < nreal; i++){
        min_realvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i];
        max_realvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i+1];
      }
      pcross_real = mxGetPr(PCROSS_REAL(S))[0];
      pmut_real   = mxGetPr(PMUT_REAL(S))[0];
    }
    //...
```

Código 7.14: Inicialización de las variables requeridas en caso de trabajar con números reales como cromosoma

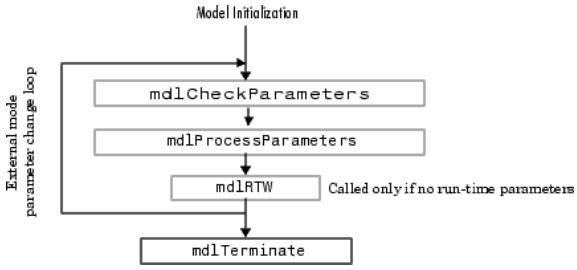


Figura 48: Estructura de llamada alternativa para el modo externo.

Habiendo revisado de nuevo la documentación, como se muestra en la figura 48, al ejecutarlo en modo externo, dicha llamada se realiza después de la inicialización del modelo; razón por la cual no se había asignado un valor a las variables y causaba que no se adjudicará la memoria correctamente.

7.5.2. Pruebas en Simulink Real-Time

Para concluir se procede con la simulación del algoritmo de control sobre una plataforma *Speedgoat*. La plataforma *Speedgoat Baseline-S* solicitada se trata de un ordenador industrial resistente con las siguientes características [63]:

- CPU: *Intel Celeron 2 GHz, 4 cores*.
- memoria RAM: 4GB
- unidad principal: 256 GB *SSD*.

Figura 49: *Baseline-S frente*

Figura 50: *Baseline-S trasera*

Figura 51: *Baseline-S lado*

Para poder interactuar con el ordenador se debe contar obviamente, con el *Add-on* de *Simulink Real-Time* y las librerías de *driver* `speedgoat` para el mismo. Para instalar la librería *Speedgoat* proporcionadas por el fabricante y que se descargan desde el portal de cliente de su página web, simplemente basta con navegar hasta el directorio raíz, ejecutar el *script* `speedgoat_setup.p` y seguir las instrucciones del asistente. Una vez terminada la instalación, se debe reiniciar MATLAB y con ello se termina de preparar la librería.

A continuación se debe configurar la conexión IP/Ethernet entre el *Speedgoat* y el ordenador de desarrollo. Para ello, se hace uso del puerto Ethernet con la indicación *Host Link* del *Speedgoat* y se conectan ambos ordenadores a la misma red privada. Según los pasos del fabricante, se debe asignar la siguiente dirección IP al ordenador de desarrollo (*Windows*):

- **Dirección IP**: 192.168.7.2
- **Máscara de subred**: 255.255.255.0

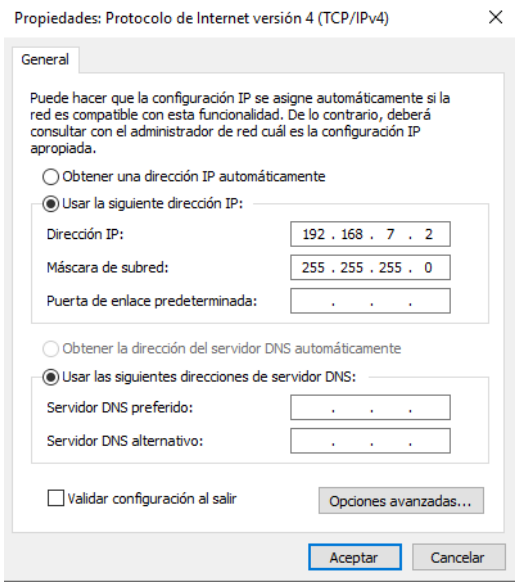


Figura 52: Configuración de ajustes de red en *Windows*

Por otro lado, para el *Speedgoat* la dirección IP *flasheada* por defecto es:

- **Dirección IP:** 192.168.7.5
- **Máscara de subred**: 255.255.255.0

Esta dirección se puede modificar cargando una versión modificada del *Simulink Real-Time Kernel* por defecto, o una vez establecida una primera conexión, asignando una nueva dirección al *target* desde *Simulink Real-Time Explorer*

El siguiente paso en la configuración de la conexión, es el ajuste en *Simulink Real-Time Explorer*, en el ordenador de desarrollo. En MATLAB, se puede escribir `slrtexplr` o `slrtExplorer` (en versiones actuales) para abrir la ventana de búsqueda de dispositivos *target* en red. En el caso que se ha llevado a cabo, bastaba con especificar la dirección IP; no obstante, es posible configurar varios parámetros y ajustes extra, como los puertos y enlaces si la conexión lo requiriese. Finalmente, se puede dar al botón de conectar para establecer la conexión con el ordenador industrial.

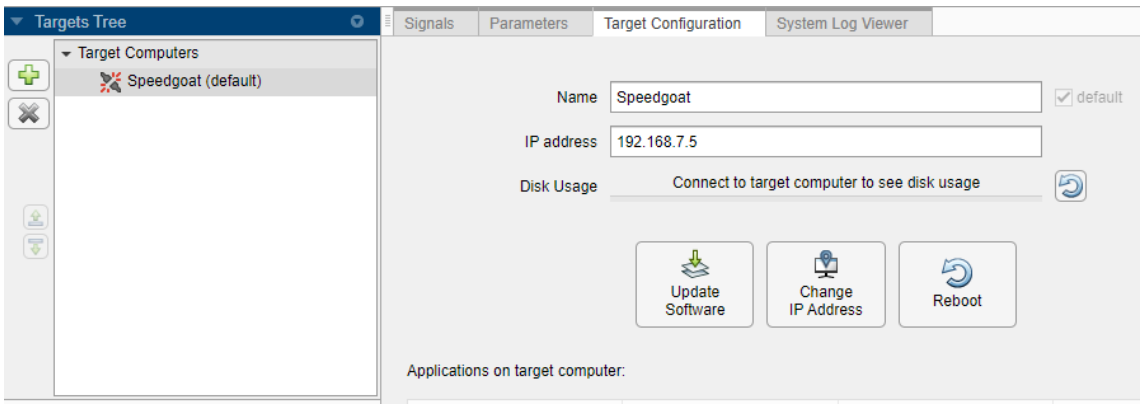


Figura 53: *Ajuste en Simulink Real-Time Explorer*

Se puede comprobar la conexión desde MATLAB con el comando `slrtpingtarget` o desde

el *Powershell* de *Windows* mediante: `ping 192.168.7.5`. Para asegurarse de que los bytes de datos llegan correctamente y que hay una latencia baja.

Habiendo establecido la conexión, se puede proceder con la parte de *Simulink Real-Time*. Dentro de *Simulink* se puede acceder a la pestaña de tiempo real y cerciorarse de un enlace correcto con el *Speedgoat*, gracias al icono arriba a la izquierda.

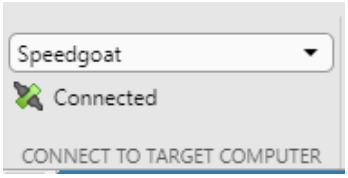


Figura 54: Señalización de la conexión

Un comentario a remarcar, es que antes de realizar un pequeño cambio de ajustes, saltaba un error al intentar montar el programa y ejecutarlo en el *target*:

```
1    The storage class of a parameter of a non-inlined S-function must be 'Auto'
     , or defined in the Embedded Coder Dictionary as a structured storage class
     . Parameter number 20 ('nin') in the non-inlined S-function 'SNL1_rt/S-
     Function' has an unsupported storage class.
```

Resulta que los parámetros de entrada de la *S-Function* no estaban especificados para ser 'perfilados'(*inlined*). Sobre esta cuestión, se debe cambiar la casilla de "*Default parameter behaviour*" en la pestaña de optimización, de "*Tunable*" a "*Inlined*" tal y como se muestra en la siguiente figura:

Figura 55: Modificación del comportamiento por defecto de los parámetros en la generación de código

Finalmente, tal y como se ha expuesto para la ejecución en modo *Kernel*, se debe seleccionar un *solver* de paso fijo y en este caso como no se tienen estados continuos, se ha escogido uno discreto. Habiendo compilado la función especial a un archivo `.mex` y asignando los parámetros de la entrada del bloque se puede iniciar la ejecución en el *Speedgoat*.

Sobre la ejecución, se ha de mencionar que tanto en SISO como en MIMO las simulaciones han sido un exitosas y sin ningún error de código, más allá de posibles sobrecargas del CPU debido a poblaciones y número de generaciones demasiado altas. En los siguientes párrafos se profundiza más sobre dichas simulaciones.

7.5.3. Resultados de ejecución en tiempo real

Debido a la reducida capacidad de cómputo del ordenador industrial, frente al ordenador de desarrollo que se había empleado, no se han podido emplear tantos individuos y tantas generaciones como en simulación. Si se comete el error de emplear un número demasiado elevado de individuos o generaciones en el *Speedgoat*, salta un error de sobrecarga; es decir, el TET (*Target Execution Time*/ Tiempo de ejecución del objetivo) es superior al tiempo de muestreo establecido.

Ante el posible riesgo de sobrecarga, se ha decidido comenzar con una simulación más simple, en SISO, para ir aumentando en complejidad hasta llegar a MIMO con perturbaciones. Se parte de un número de población y generaciones reducido (e.g 80 individuos y 100 generaciones) y 4 instantes de muestreo tanto para el horizonte de control como el de predicción.

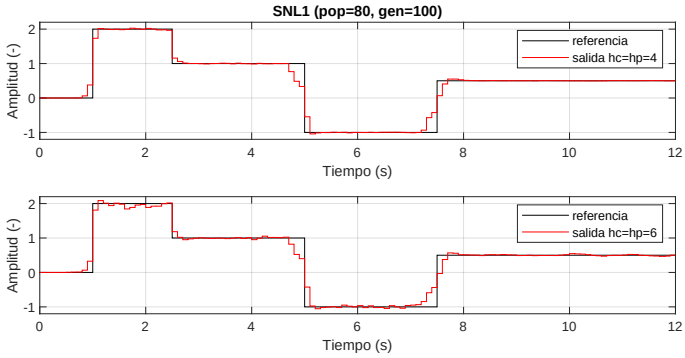


Figura 56: SNL1 en RT, hc=hp=4:6

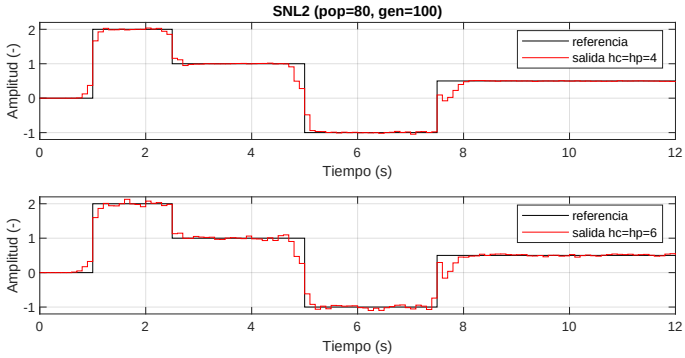


Figura 57: SNL2 en RT, hc=hp=4:6

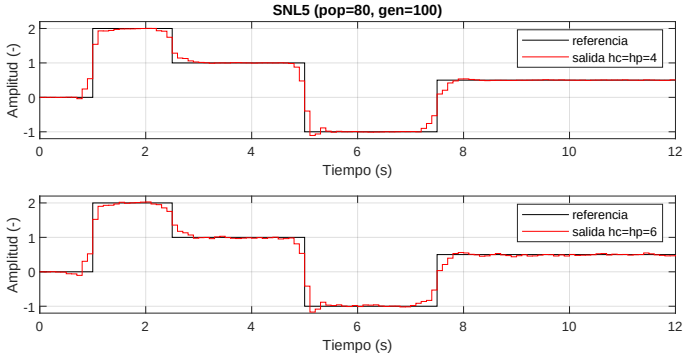


Figura 58: SNL5 en RT, hc=hp=4:6

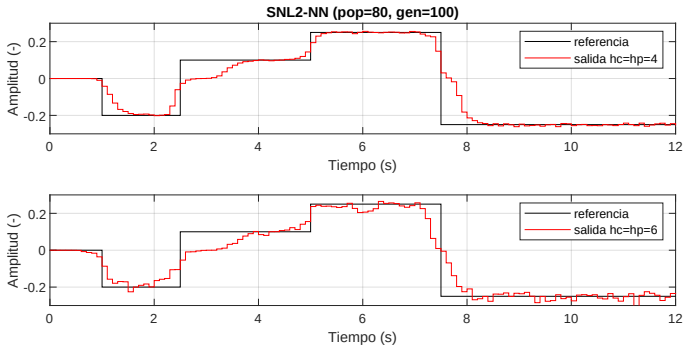


Figura 59: SNL2-NN en RT, hc=hp=4:6

Las figuras anteriores presentan las evoluciones de las salidas en simulación para los diferentes sistemas ante horizonte de control y predicción diferentes. Relacionando el hilo con el artículo de 2015 de M.Larrea, E.Larzabal, E.Irigoyen, J.J.Valera y M.Dendaluce [34], donde también se realizan pruebas del algoritmo predictivo inteligente sobre un controlador industrial, se emplea una población de 80 individuos con 100 iteraciones en cada muestreo como parámetros del optimizador genético. Aunque en el caso del artículo son 70 individuos con 100 iteraciones, debido a la limitación del código original de NSGA-II se requiere que se empleen múltiplos de 4 como tamaño de población.

Aunque con una población e iteraciones inferior a la habitual para simulaciones anteriores en *Simulink*, el resultado que se puede apreciar es satisfactorio. En la tabla 5 de abajo se exponen los errores cuadráticos medios y errores absolutos. Se puede observar que el *MSE* se mantiene por debajo de 0.02 para todos los sistemas con modelo matemático, y solo empeora ligeramente cuando se incrementa el horizonte. De la misma manera, el error máximo absoluto esta alrededor de 0.65, siendo el único sistema que se desvía el SNL2. Mirando el índice del instante donde ocurre este error, se descubre que se da justo cuando se da el mayor cambio de referencia (de 1 a -1), en el instante 5.

	SISO RT (pop = 80, gen = 100)		
	Hc = Hp	**MSE**	**MaxAbsE**
SNL1	4	0,01608	0,6652
	6	0,01711	0,6003
SNL2	4	0,01833	0,721
	6	0,01909	0,7339
SNL5	4	0,01656	0,6022
	6	0,01802	0,5839

	SISO RT (pop = 80, gen = 100)		
	Hc = Hp	**MSE**	**MaxAbsE**
SNL2_NN	4	0,00334	0,2804
	6	0,00329	0,2446

Tabla 5: Tablas de error cuadrático medio y absoluto para simulaciones en tiempo real SISO

Asimismo, los resultados de error con el modelo neuronal `SNL2_NN`, no son del todo desfavorables. Bien es verdad que el modelo neuronal empleado como predictor es pobre, dentro del rango de operación [-0.5, 0.5] su rendimiento es mejor, demostrando un reducido deterioro debido al incremento del horizonte bajo el mismo tiempo de cómputo.

Los resultados de seguimiento de referencia, sin perturbaciones, de ambos sistemas MIMO 2x2, se presentan en las figuras 60 y 61. El principal cambio que se observa es una pérdida de calidad; esto se debe a que manteniendo un tiempo de muestreo de 0.1s no se conseguía terminar la optimización dentro del límite y se requirió una reducción de población e iteraciones. En otras palabras, el TET (*Target Execution Time*) era superior a 0.1 segundos, y la combinación de cantidades máximas de individuos e iteraciones que se encontró para una ejecución fluida para todos los tamaños de horizontes hasta llegar a 6 instantes, fue 64 individuos y 90 iteraciones.

De igual manera que las simulaciones de una entrada/una salida, se tiene en la tabla 6, la

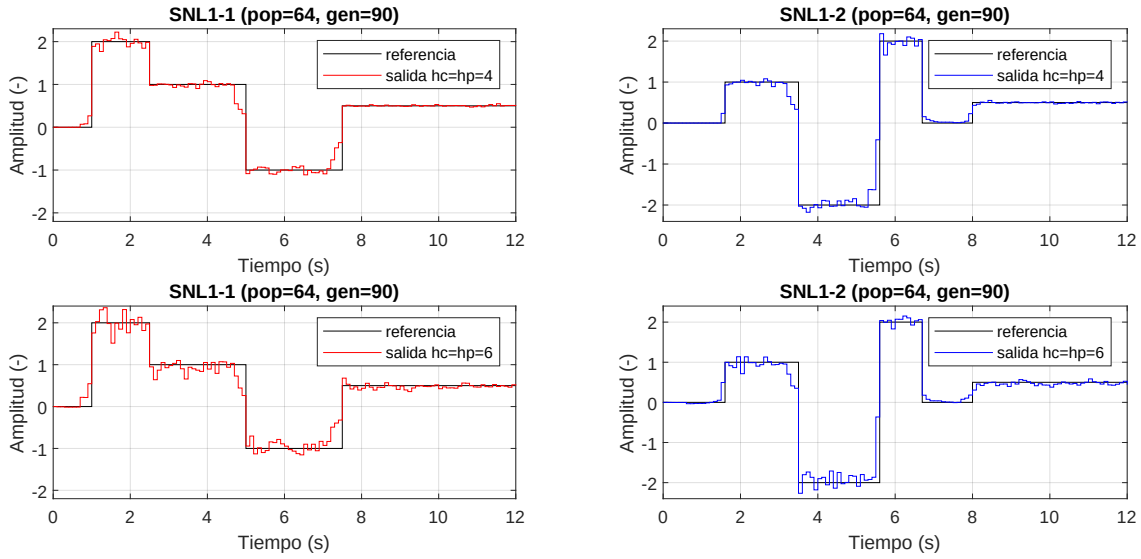


Figura 60: SNL1-SNL1 en RT, hc=hp=4:6

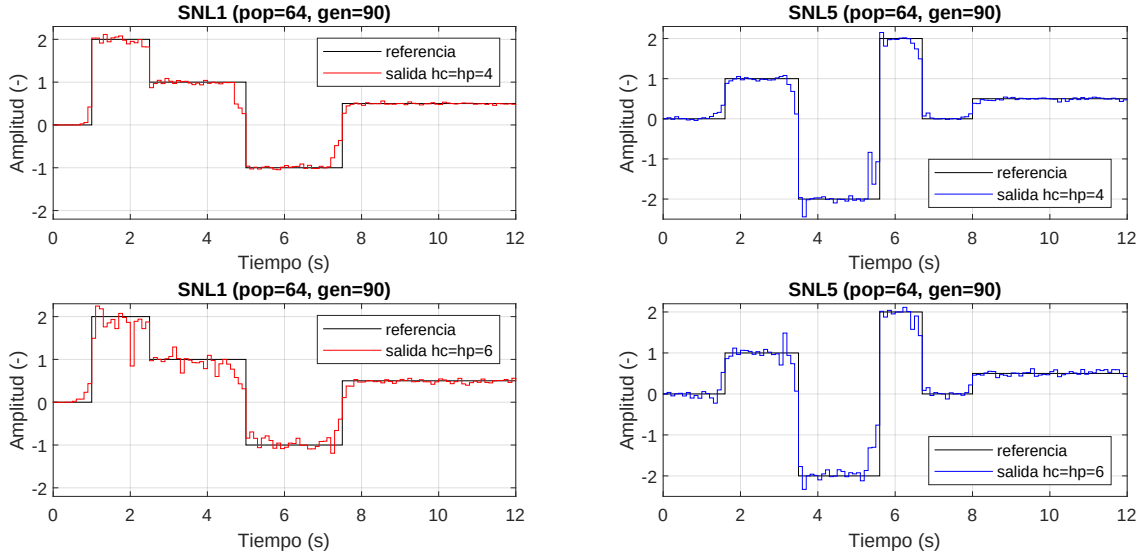


Figura 61: SNL1-SNL5 en RT, hc=hp=4:6

		MIMO RT (pop = 64, gen = 90)		
		Hc = Hp	**MSE**	**MaxAbsE**
SNL1-SNL1	SNL1 - 1	4	0,01767	0,6846
		6	0,03390	0,7365
	SNL1 - 2	4	0,03482	1,5916
		6	0,04017	1,6361
SNL1-SNL5	SNL1	4	0,01931	0,736
		6	0,03979	1,1548
	SNL5	4	0,03461	1,1641
		6	0,04355	1,2394

Tabla 6: Tabla de error cuadrático medio y absoluto para simulaciones en tiempo real MIMO

información sobre los errores de las simulaciones de entrada múltiple/salida múltiple. Aunque a simple vista se perciben mayores altibajos en las salidas del sistema a la hora de seguir la consigna, mirando las cifras, se puede argumentar que tampoco difieren demasiado de los resultados mostrados anteriormente en la tabla 5 para el caso del horizonte de 4.

En el caso de SNL1 en ambos sistemas, su *MSE* sigue manteniendose debajo de 0.02 y su error absoluto alrededor de 0.7, manteniendo la misma referencia. Para SNL1-2 y SNL5 se tiene una referencia distinta para poder diferenciarlos fácilmente y adicionalmente, demostrar que no se tiene ningún efecto de acoplamiento entre SNL1-1 y SNL1-2 ni tampoco entre SNL1 y SNL5.

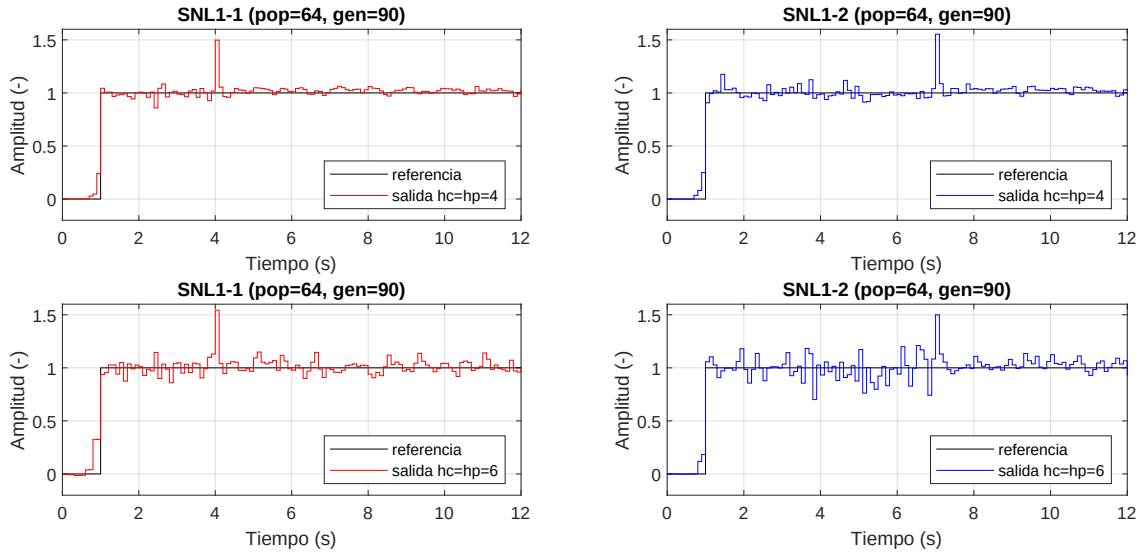


Figura 62: SNL1-SNL5 en RT ante una perturbación de 0.5, hc=hp=4:6

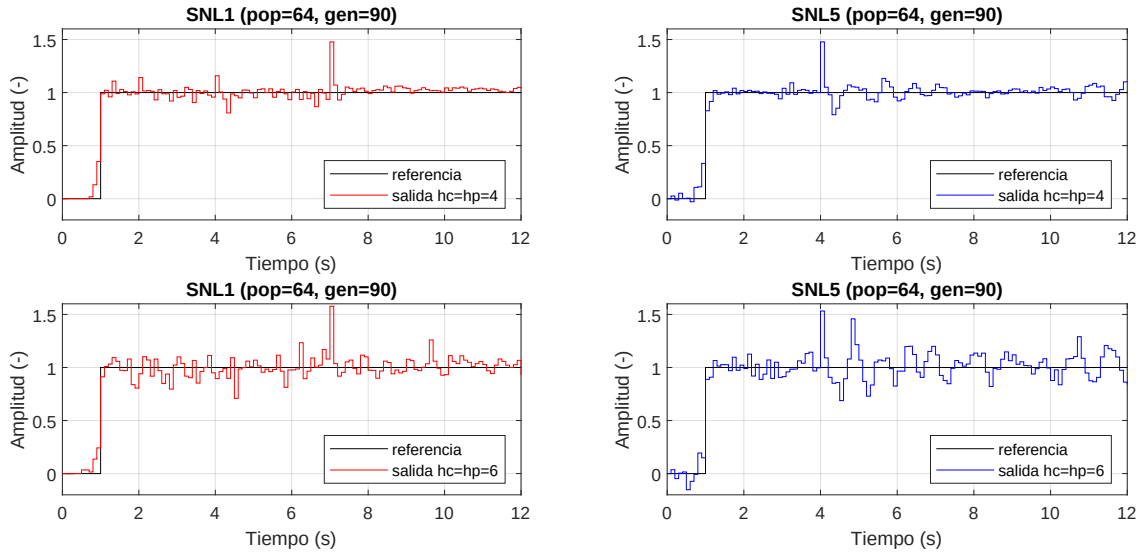


Figura 63: SNL1-SNL5 en RT ante una perturbación de 0.5, hc=hp=4:6

		MIMO + perturbación RT (pop = 64, gen = 90)		
		Hc = Hp	**MSE**	**MaxAbsE**
SNL1-SNL1	SNL1 - 1	4	0,0075	0,5429
		6	0,0036	0,4976
	SNL1 - 2	4	0,0102	0,4997
		6	0,0049	0,554
SNL1-SNL5	SNL1	4	0,0051	0,4776
		6	0,0099	0,5773
	SNL5	4	0,0055	0,4777
		6	0,0157	0,5322

Tabla 7: Tabla de error cuadrático medio y absoluto para simulaciones en tiempo real MIMO con perturbación

8. Planificación

En este capítulo se tratará la planificación de las tareas a lo largo del proyecto. Se han dividido los objetivos principales, en subtareas, para una mayor resolución del trabajo que se ha realizado en el tiempo, cronológicamente.

Código	Nombre	Duración
T.0	**Gestión del TFM**	
T.0.1	Preparación del TFM y planificación inicial	1 sem
T.0.2	**Seguimiento y revisión del estado del TFM**	
T.0.2.1	**Comité de seguimiento**	
T.1	**Estudio del estado del arte y estrategia iMO-NMPC mediante scripts de MATLAB**	
T.1.1	Recopilación de información y Estado del Arte	3;6 sem
T.1.2	Estudio y mejora scripts SISO	2 sem
T.1.3	Estudio y mejora scripts MIMO	2 sem
H.1.1	Script con modelo matemático vectorizado finalizado	
T.1.4	Entrenamiento e inclusión de redes neuronales	1 sem
T.1.5	Adaptación del script para redes neuronales y gestión de cromosomas	2 sem
H.1.2	Script con modelo neuronal vectorizado y gestión de cromosomas finalizado	
T.2	**Aprendizaje y Análisis del API S-Function para Simulink**	
T.2.1	Estudio del API Matlab Level-2 y desarrollo de programas ejemplo	1 sem
T.2.2	Estudio del API CMEX y desarrollo de programas ejemplo	2 sem
H.2.1	Documentación CMEX API	
T.3	**Implementación de la estrategia de control inteligente sobre plataformas de Simulación**	
T.3.1	Estudio del código NSGA-II en C	1 sem
T.3.2	Desarrollo de la estrategia iMO-NMPC SISO con modelo matemático	3 sem
T.3.3	Inclusión modelo neuronal en el control SISO.	1 sem
T.3.4	Desarrollo de mecanismo de rechazo de perturbaciones	1 sem
H.3.1	Bloque de programa iMO-NMPC SISO funcional	
T.3.5	Desarrollo de la estrategia iMO-NMPC MIMO con modelo matemático	2 sem
H.3.2	Bloque de programa iMO-NMPC MIMO funcional	
T.4	**Estudio de alternativas para la ejecución en tiempo real**	
T.4.1	Pruebas en Simulink Desktop Real Time y adaptación del código	0,5 sem
T.4.2	Pruebas en Simulink Real Time	0,5 sem
H.4	Prototipo funcional en plataformas de tiempo real	
T.5	**Cierre del Trabajo**	
T.5.1	Elaboración del documento TFM	1 sem
T.5.2	Revisión del Documento TFM	1 sem
H.5	TFG Entregado	
	TOTAL	24 sem

Tabla 8: Listado de tareas.

En esta proposición inicial, la duración total del trabajo es de 24 semanas que equivalen a 600h (24 créditos ECTS) con 25 horas semanales. Una particularidad de esta planificación es que la tarea T.1.2 "Estudio y mejora scripts SISO" no se empieza hasta haber realizado un estudio básico en la tarea T.1.1 " Recopilación de información y Estado del Arte" que se traduce en el trabajo de 3 semanas, dando por terminado el estado del arte en 6 semanas. Por otra parte se puede observar que se plantean que las tareas T.3.3 y T.3.4 se efectúen en paralelo en la misma semana ya que el bloque de software de uno no afecta al otro.

Se detallan a continuación cada una de las tareas:

1. **T.0 Gestión del TFM**: Esta tarea engloba toda la gestión y planificación realizada a lo largo del desarrollo del TFM.
2. T.0.1 Preparación del TFM y planificación inicial: Esta tarea trata los trámites iniciales del ingeniero junior; identificación, documentación y una primera planificación de las tareas.
3. **T.0.2 Seguimiento y revisión del estado del TFM**
4. T.0.2.1 Comité de seguimiento: esta tarea es representativa de las reuniones del ingeniero junior con los directores del trabajo para su revisión, orientación y dudas.
5. **T.1 Estudio del estado del arte y estrategia iMO-NMPC mediante scripts de MATLAB**
6. T.1.1 Recopilación de información y Estado del Arte: se recopila información sobre las diferentes partes y algoritmos que compone la estrategia y se redacta el apartado del estado del arte.
7. T.1.2 Estudio y mejora de scripts en SISO: Se parte de los algoritmos en script de MATLAB desarrollados por el grupo de investigación para familiarizarse con la lógica de la estrategia y realizar optimizaciones y pequeñas mejoras sobre el código.
8. T.1.3 Estudio y mejora de scripts en MIMO: Partiendo de los scripts monovariables se extiende la lógica del algoritmo para englobar sistemas multivariables.
9. T.1.4 Entrenamiento e inclusión de redes neuronales: En esta tarea, partiendo de los modelos matemáticos de los sistemas no lineales de *benchmark* se entrenan algunas redes neuronales para emplearlas como modelo de predicción.
10. T.1.5 Adaptación del script para redes neuronales y gestión de cromosomas. Se sustituye el método convencional de MATLAB para *feedforward* en las redes por un método manual y rápido. Adicionalmente se desarrolla un método de inicialización en para el algoritmo genético de manera que la siguiente población se le asignan los valores obtenidos en la anterior para ahorrar cómputo y acelerar la convergencia al óptimo.
11. T.2 **Aprendizaje y Análisis del API S-Functions para Simulink**
12. T.2.1 Estudio del API Matlab Level-2 y desarrollo de programas ejemplo: esta tarea trata de ganar experiencia y familiarizarse con el API que tiene MATLAB para integrar funciones especiales del usuario en Simulink. En este primer paso programando en lenguaje de MATLAB.
13. T.2.2 Estudio del API CMEX y desarrollo de programas ejemplo: esta tarea es la continuación de la anterior. No obstante, ahora se hace el salto del lenguaje de MATLAB a C. Dentro de esta tarea se desarrollan tanto S-Functions triviales como otras más complejas como el control DMC antes de abordar el desarrollo del código de la estrategia completa.
14. T.3 **Implementación de la estrategia de control inteligente sobre la plataforma de Simulación**
15. T.3.1 Estudio del código NSGA-II en C: Este primer estudio del código del algoritmo genético desarrollado por K.Deb es esencial para su incorporación en totalidad del código.

16. T.3.2 Desarrollo de la estrategia iMO-NMPC SISO con modelo matemático: Esta es la principal subtarea de este apartado. Inicialmente, para abordar el problema se trata con SISO y modelos de predicción matemáticos.

17. T.3.3 Inclusión del modelo neuronal en el control SISO. En esta subtarea se desarrolla una función que realizará el papel de modelo de predicción. La función se encarga del *feedforward* de una red entrenada (SNL2) para obtener la salida predicha.

18. T.3.4 Desarrollo del mecanismo de rechazo de perturbaciones. Esta tarea, junto a la tarea anterior completa el desarrollo de la estrategia; finalmente incorporando la lógica para calcular el error de predicción y teniéndola en cuenta para futuras predicciones, así, afianzando un mecanismo de rechazo de perturbaciones.

19. T.3.5 Desarrollo de la estrategia iMO-NMPC MIMO con modelo matemático. De la misma manera que se ha realizado en SISO, se extiende la lógica de la estrategia de control para englobar múltiples entradas y salidas.

20. **T.4 Estudio de alternativas para la ejecución en tiempo real**. Esta tarea trata de estudiar la viabilidad de los diferentes métodos que se ofrecen dentro y fuera de MATLAB para permitir una ejecución del código desarrollado en tiempo real.

21. T.4.1 Pruebas en Simulink Desktop Real Time y adaptación del código.

22. T.4.2 Pruebas en Simulink Real Time.

23. **T.5 Cierre del Trabajo**

24. T.5.1 Elaboración del documento TFM. Se valoran los resultados obtenidos y se realiza una recopilación de todos los pasos en el desarrollo para redactar la memoria final del trabajo.

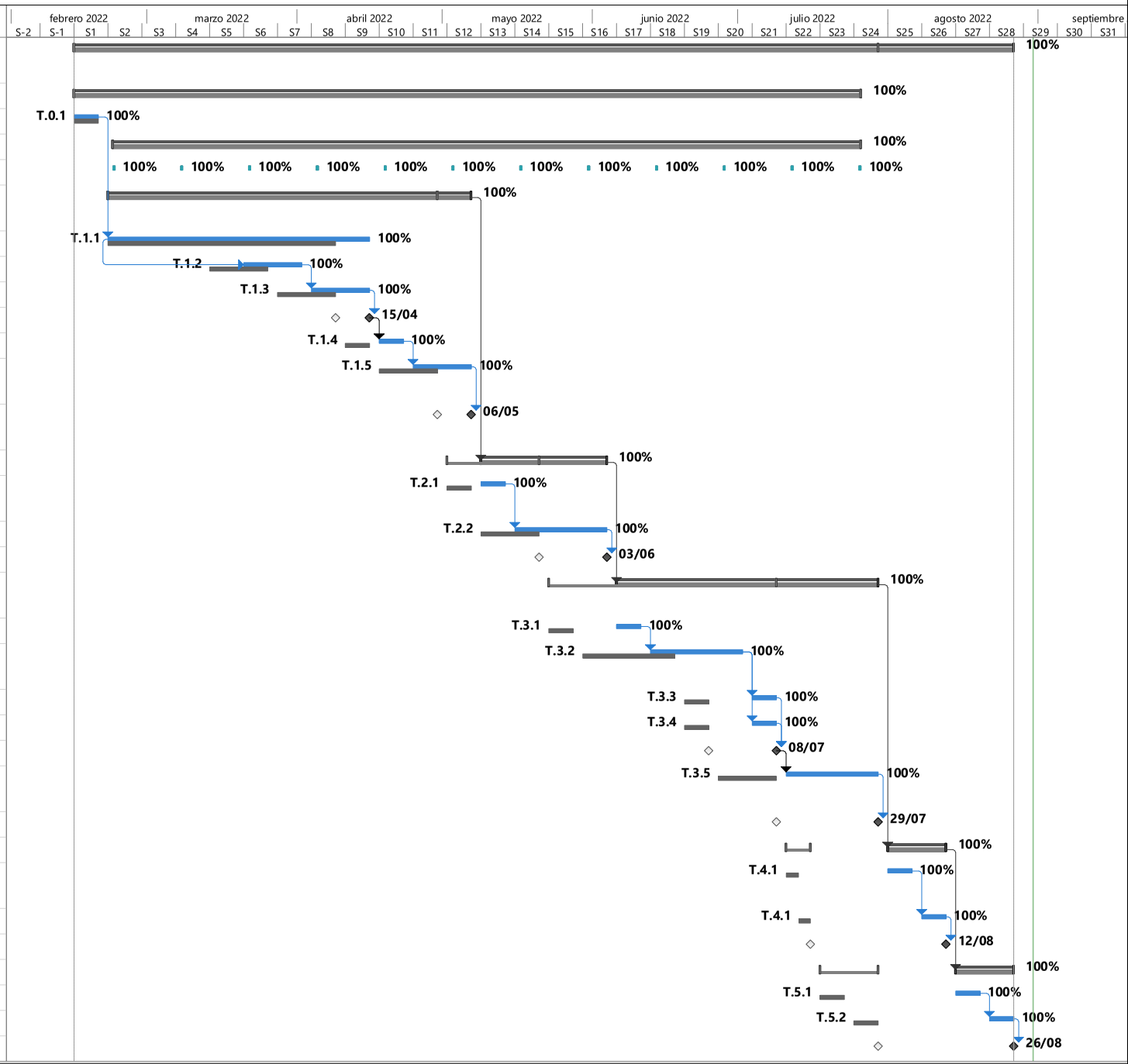
25. T.5.2 Revisión del Documento TFM. Finalmente se corrige y se modifica el documento generado gracias a las aportaciones de los directores del TFM y el ingeniero junior.

8.1. Seguimiento del Trabajo

En el diagrama gantt de la siguiente página se puede observar que existen diferencias entre la línea base y lo real temporalmente. Las principales diferencias se deben a lo siguiente: Inicialmente la recopilación básica de información de tarea del estado del arte se demoró más de lo previsto, es por eso que la tarea T.1.2 empieza una semana más tarde. En segundo lugar la tarea T.2.2 "Estudio del API CMEX y desarrollo de programas ejemplo" ha tomado más tiempo de lo previsto, con una duración real de 3 semanas. En tercer lugar, de la misma manera, el tiempo previsto para la tarea T.3.5 de modificación del código en C para MIMO se alarga una semana demás. Finalmente ambas tareas T.4.1 y T.4.1 pasan de una duración prevista de 0.5 semanas cada uno a 1 semana.

Concluyendo se tiene una duración final del trabajo de fin de máster de 28 semanas, con 700h en total (25 h por semana).

Id	Código	Nombre de tarea	Duración real
1		**IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL PREDICTIVO INTELIGENTE PARA COMPLEJOS PROCESOS MULTIVARIABLES**	**28 sem.**
2	**T.0**	**Gestión del TFM**	**23,24 sem.**
3	T.0.1	Preparación del TFG y Planificación Inicial	1 sem
4	**T.0.2**	**Seguimiento y Revisión del Estado del TFM**	**22,04 sem.**
5	**T.0.2.1**	**Comité de Seguimiento**	**22,04 sem.**
18	**T.1**	**Estudio del estado del arte y estrategia iMO-NMPC mediante scripts de MATLAB**	**11 sem.**
19	T.1.1	Recopilación de información y Estado del Arte	8 sem.
20	T.1.2	Estudio y mejora scripts SISO	2 sem.
21	T.1.3	Estudio y mejora scripts MIMO	2 sem.
22	H.1.1	H.1.1 - Script con modelo matemático vectorizado finalizado	0 sem.
23	T.1.4	Entrenamiento e inclusión de redes neuronales	1 sem
24	T.1.5	Adaptación del script para redes neuronales y gestión de cromosomas	2 sem.
25	H.1.2	H.1.2 - Script con modelo neuronal vectorizado y gestión de cromosomas finalizado	0 sem.
26	**T.2**	**Aprendizaje y Análisis del API S-Function para Simulink**	**4 sem.**
27	T.2.1	Estudio del API Matlab Level-2 y desarrollo de programas ejemplo	1 sem
28	T.2.2	Estudio del API CMEX y desarrollo de programas ejemplo	3 sem.
29	H.2	H.2 - Documentación CMEX API	0 sem.
30	**T.3**	**Implementación de la estrategia de control inteligente sobre plataformas de Simulación**	**8 sem.**
31	T.3.1	Estudio del código NSGA-II en C	1 sem
32	T.3.2	Desarrollo de la estrategia iMO-NMPC SISO con modelo matemático	3 sem.
33	T.3.3	Inclusión modelo neuronal en el control SISO	1 sem
34	T.3.4	Desarrollo de mecanismo de rechazo de perturbaciones	1 sem
35	H.3.1	H-3.1 - Bloque de programa iMO-NMPC SISO funcional	0 sem.
36	T.3.5	Desarrollo de la estrategia iMO-NMPC MIMO con modelo matemático	3 sem.
37	H.3.2	H.3.2 - Bloque de programa Imo-NMPC MIMO funcional	0 sem.
38	**T.4**	**Estudio de alternativas para la ejecución en tiempo real**	**2 sem.**
39	T.4.1	Pruebas en Simulink Desktop Real Time y adaptación del código	1 sem
40	T.4.1	Pruebas en Simulink Real Time	1 sem
41	H.4	H.4 - Prototipo funcional en plataformas de tiempo real	0 sem.
42	**T.5**	**Cierre del Trabajo**	**2 sem.**
43	T.5.1	Elaboración del Documento de TFM	1 sem
44	T.5.2	Revisión Documento TFM	1 sem
45	H.5	H.5 -TFM Entregado	0 sem.



Tareas críticas · División crítica · Progreso de tarea crítica · Tarea · División · Progreso de tarea · Tarea manual · solo el comienzo · solo fin · solo duración · Línea base · División de la línea base · Hito de línea base · Hito · Progreso del resumen · Resumen · Resumen manual · Resumen del proyecto · Tareas externas · Hito externo · Tarea inactiva · Hito inactivo · Resumen inactivo · Fecha límite · Resumen de línea base

Asier Zabaljauregi Lekerika

9. Descripción del presupuesto

En este capitulo se van a detallar los gastos y amortizaciones que ha supuesto el trabajo de fin de máster. Para ello se divide el coste total en partidas incluyendo horas internas, amortizaciones y gastos.

9.1. Tabla de cálculo de tasas unitarias

Para la tabla de tasas unitarias se incluyen las herramientas principales requeridas, tanto hardware como software, para llevar a cabo el trabajo de fin de máster. Se consideran 5 años de vida útil tanto para el conjunto del ordenador, como para el *Speedgoat*, debido a su uso continuado e intensivo y al rápido desarrollo de la tecnología que los dejara obsoletos.

En cuanto a elementos de software, se ha considerado una licencia anual para las herramientas de *MATLAB/Simulink* y una quinquenal para las licencias de ofimática de *Microsoft*. Esto se debe a que se considera más crítico mantener actualizadas las herramientas de desarrollo principales ante cualquier cambio. Los precios de las licencias de Simulink y Matlab se han obtenido de la página web oficial de MathWorks [44].

Concepto	**Precio Inicial**	**Cantidad**	**Vida útil (años)**	**horas/año**	**Tasa Amort. (€/h)**
Conjunto Ordenador	4000	1	5	1700	0,47
Software					
MATLAB	840	1	1	1700	0,49
Matlab Coder	2500	1	1	1700	1,47
Matlab Compiler	2100	1	1	1700	1,24
Control System Toolbox	480	1	1	1700	0,28
Deep Learning Toolbox	480	1	1	1700	0,28
Optimization Toolbox	480	1	1	1700	0,28
Simulink	1260	1	1	1700	0,74
Simulink Coder	1260	1	1	1700	0,74
Simulink Compiler	1920	1	1	1700	1,13
Simulink Control Design	520	1	1	1700	0,31
Simulink Desktop Real-Time	840	1	1	1700	0,49
Simulink Real-Time	1920	1	1	1700	1,13
Microsoft 365 Personal	69	1	1	1700	0,04
Microsoft Project Standard 2021	929	1	5	1700	0,11
Ordenador Speedgoat	6000	1	5	1700	0,71

Tabla 9: Tabla de cálculo de tasas unitarias

9.2. Presupuesto por partidas

Concepto	Unidad	NºUnidades	Tasa Unitaria	Coste
HORAS INTERNAS				**17.700,00 €**
Ingeniero Junior	h	600	20	12.000,00 €
Ingeniero Senior	h	60	45	2.700,00 €
Director de Proyecto	h	60	50	3.000,00 €
AMORTIZACIONES				**1.727,05 €**
Conjunto Ordenador	h	600	0,471	282,35 €
Ordenador Speedgoat	h	12,5	0,706	8,82 €
Software	h			
MATLAB	h	250	0,494	123,53 €
Matlab Coder	h	300	1,471	441,18 €
Matlab Compiler	h	300	1,235	370,59 €
Control System Toolbox	h	100	0,282	28,24 €
Deep Learning Toolbox	h	75	0,282	21,18 €
Optimization Toolbox	h	250	0,282	70,59 €
Simulink	h	300	0,741	222,35 €
Simulink Coder	h	12,5	0,741	9,26 €
Simulink Compiler	h	12,5	1,129	14,12 €
Simulink Control Design	h	350	0,306	107,06 €
Simulink Desktop Real-Time	h	12,5	0,494	6,18 €
Simulink Real-Time	h	12,5	1,129	14,12 €
Microsoft 365 Personal	h	50	0,041	2,03 €
Microsoft Project Standard 2021	h	50	0,109	5,46 €
COSTES DIRECTOS				**19.427,05 €**
Costes Indirectos	7 %			1.359,89 €
SUBTOTAL				**20.786,95 €**
Imprevistos	10 %			2.078,70 €
TOTAL				**22.865,67 €**

Tabla 10: Presupuesto preliminar dividido por partidas

Concepto	Unidad	NºUnidades	Tasa Unitaria	Coste
HORAS INTERNAS				**20.650,00 €**
Ingeniero Junior	h	700	20	14.000,00 €
Ingeniero Senior	h	70	45	3.150,00 €
Director de Proyecto	h	70	50	3.500,00 €
AMORTIZACIONES				**2.037,79 €**
Conjunto Ordenador	h	700	0,471	329,41 €
Ordenador Speedgoat	h	25	0,706	17,65 €
Software	h			
MATLAB	h	300	0,494	148,24 €
Matlab Coder	h	350	1,471	514,71 €
Matlab Compiler	h	350	1,235	432,35 €
Control System Toolbox	h	100	0,282	28,24 €
Deep Learning Toolbox	h	75	0,282	21,18 €
Optimization Toolbox	h	300	0,282	84,71 €
Simulink	h	350	0,741	259,41 €
Simulink Coder	h	25	0,741	18,53 €
Simulink Compiler	h	25	1,129	28,24 €
Simulink Control Design	h	350	0,306	107,06 €
Simulink Desktop Real-Time	h	25	0,494	12,35 €
Simulink Real-Time	h	25	1,129	28,24 €
Microsoft 365 Personal	h	50	0,041	2,03 €
Microsoft Project Standard 2021	h	50	0,109	5,46 €
COSTES DIRECTOS				**22.687,79 €**
Costes Indirectos	7 %			1.588,15 €
TOTAL				**24.275,93 €**

Tabla 11: Presupuesto de seguimiento dividido por partidas

Como se puede apreciar, del presupuesto preliminar a la final, existe una diferencia de 1500€, las cuales tampoco se llegaron a cubrir hasta con un 10 % de imprevistos. Las tareas de estudio del API y la modificación a MIMO tomaron más tiempo de lo previsto, incrementando por mucho las horas de ingeniería y amortización de los ordenadores y software. Percance que a lo mejor, se podría haber evitado con métodos de planificación ágiles.

10. Conclusiones y Líneas Futuras

Finalmente se ha obtenido una base de código actualizada, así como una prueba de trabajo del código tanto para sistemas de única entrada y salida como para sistemas de múltiples entradas y salidas, gracias a la novedosa implementación en MIMO. Adicionalmente se ha comprobado su validez en plataformas de tiempo real y se dejan documentadas las instrucciones para ello.

Dando una visión general, el trabajo abordado cubre varios ámbitos de conocimiento; desde un aspecto teórico como el control predictivo, redes neuronales y algoritmos genéticos, hasta aspectos más prácticos como el manejo de diferentes lenguajes de programación (MATLAB, C), entornos de desarrollo y *hardware*.

No obstante, la investigación en la estrategia esta lejos de estar finalizada. Como se ha mencionado en la descripción de la solución, es de importancia obtener un modelo MIMO acoplado, donde los cambios en las variables de control se afectan entre ellos y comprobar la eficacia de la estrategia. En lo referente a pruebas sobre distintos sistemas y modelos de predicción, es de interés también realizar ensayos en plantas MIMO, cuyo número de entradas y salidas sea alto, así como el número de objetivos a optimizar (e.g 10-20). Esto se debe a que como bien afirma el artículo del NSGA-III, el predecesor de éste, rinde escasamente en problemas de optimización con muchos objetivos. De igual manera, aunque el rendimiento del NSGA-II en simulación es robusto, se beneficiaría mayormente de una buena afinación de sus parámetros para cada tipo de planta a controlar. Este problema a su vez, lleva a otro camino en la investigación que es la búsqueda de métodos para una correcta puesta a punto de los parámetros del NSGA-II; según la literatura en la materia, técnicas como redes neuronales o lógica difusa pueden resultar de utilidad para esta meta-optimización de parámetros.

Un punto considerable para la validez del algoritmo de control sería comprobarlo sobre un sistema real; en un primer paso se podría simular empleando una planta con una gran verosimilitud a la real y un modelo de predicción basado en redes neuronales cuyo rango cubra el de operación, y después realizar el salto a la real empleando plataformas que permitan ejecución en tiempo real con entradas y salidas para interactuar con la planta. Los *scripts* y simulaciones sirven para comprobar la correcta lógica y funcionamiento del algoritmo, no obstante no cubren las incertidumbres e inconvenientes que puedan presentar los sistemas reales y el despliegue del control.

Mirando en un plazo más corto, queda por incorporar en el *C MEX S-Function*, la técnica de gestión de cromosomas desarrollada en *script* y mencionada en la sección 7.1. Asimismo, un estudio más profundo de los provechos que pueda traer la gestión de cromosomas en el apartado temporal y convergencia, queda por realizar. De la misma manera, se requiere la incorporación de un mecanismo de *time-out* o temporización al algoritmo de optimización sobre la plataforma de tiempo real. Por otro lado, en el apartado de modelización de sistemas, queda por efectuar un riguroso análisis de la validez de las redes neuronales NARX en la arquitectura "paralela", es decir, habiendo sido entrenada con sus propias predicciones como entrada, y después su

posterior comprobación en bucle abierto con horizontes cada vez mayores. Debido a que para grandes horizontes de predicción o intervalos grandes entre muestras de las salidas reales, el sistema de control se encuentra en bucle abierto, se pueden producir grandes discrepancias entre la predicción del modelo y la salida real. Es por ello que se considera importante tener un modelo que no se vaya a la deriva y se desvíe de la verdadera respuesta del sistema en lazo abierto. En la misma línea, es de importancia también obtener más datos estadísticos del error y varianza con esta implementación.; aunque durante el trabajo ya se han expuesto pequeñas muestras, no son más que ejemplos anecdóticos ya que solo muestran el rendimiento en una simulación. Habiendo empleado sistemas no lineales de *benchmark*, es fácil comparar su rendimiento con otras técnicas que también se han comprobado sobre los mismos sistemas.

En un aspecto más práctico, para permitir una verificación y experimentación más fluida sobre el código *S-Function*, convendría una refactorización del código para permitir una fácil intercambiabilidad de los componentes que lo conforman. Los *scripts* de MATLAB bastan para un rápido prototipado y comprobación de la lógica, no obstante, para realizar el salto a una simulación más completa y tener compatibilidad con plataformas externas, es necesaria una base de código compilable, como es el caso del *C MEX S-Function* y el resto de archivos fuente. En el mismo hilo, es necesaria desarrollar una solución accesible para la incorporación de redes neuronales NARX, entrenadas por medio de herramientas como *Deep Learning Toolbox*, u otras fuentes.

A largo plazo, sería de interés incorporar los algoritmos genéticos como NSGA-III y MOEA/D que han demostrado ser superiores en rendimiento que el GA de veinte años de antiguedad NSGA-II. No solo eso, si no que cualquier técnica de optimización dentro o fuera de la clasificación de "algoritmo genético" que haya demostrado una robusta eficacia es bienvenida para su incorporación en la estrategia de control conjunta.

Por último, pero no menos importante queda por comentar el reto de abordar los problemas de robustez y estabilidad que pueda plantear esta estrategia tan flexible como variable. Futuros trabajos pueden venir en la línea de investigación sobre el rol de las redes neuronales o modelos de *Caja Negra* como herramientas de predicción de sistemas no lineales (tanto ***determinados*** *como* ***indeterminados***) y su efecto en la estabilidad del control.

Bibliografía

[1] Marco A. Rodrigues and Darci Odloak. Mpc for stable linear systems with model uncertainty. *Automatica*, 39(4):569–583, 2003.

[2] A. Brindle. Genetic algorithms for function optimization. *PhD dissertation, University of Alberta, Edmonton.*, 1981.

[3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[4] E.F. Camacho, C. Bordons, and C.B. Alba. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2004.

[5] H. CHEN and F. ALLGöWER. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability**this paper was not presented at any ifac meeting. this paper was accepted for publication in revised form by associate editor w. bequette under the direction of editor prof. s. skogestad. *Automatica*, 34(10):1205–1217, 1998.

[6] Harish Pillai Chinmay Rajhans, Sachin C. Patwardhan. Discrete time formulation of quasi infinite horizon nonlinear model predictive control scheme with guaranteed stability. *IFAC-PapersOnLine*, 50(1):7181–7186, 2017. 20th IFAC World Congress.

[7] Luigi Chisci, John Rossiter, and G. Zappa. Systems with persistent disturbances: Predictive control with restricted constraints. *Automatica*, 37:1019–1028, 07 2001.

[8] D.W. Clarke and C. Mohtadi. Properties of generalized predictive control. *IFAC Proceedings Volumes*, 20(5, Part 10):65–76, 1987. 10th Triennial IFAC Congress on Automatic Control - 1987 Volume X, Munich, Germany, 27-31 July.

[9] D.W. Clarke, C. Mohtadi, and P.S. Tuffs. Generalized predictive control—part i. the basic algorithm. *Automatica*, 23(2):137–148, 1987.

[10] Mhhhmakchofi Cnocohocti and D. Bert. On the minimax reachability of target sets and target tubes. 07 2002.

[11] Carlos A Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11):1245–1287, 2002.

[12] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[13] L. Davis. Genetic algorithms and simulated annealing. 1987.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[15] Kalyanmoy Deb. Multi-objective NSGA-II code in C. `https://www.egr.msu.edu/~kdeb/codes.shtml`. [Online; accessed 5-August-2022].

[16] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[17] K.A DeJong. An analysis of the behaviour of a class ofgenetic adaptive systems. *PhD dissertation, University of Michigan*, 1975.

[18] Martin Dendaluce, JuanJose Valera, Vicente Gomez-Garay, Eloy Irigoyen, and Ekaitz Larzabal. Microcontroller implementation of a multi objective genetic algorithm for real-time intelligent control. In Alvaro Herrero, , and Emilio Corchado, editors, *International Joint Conference SOCO13*, volume 239 of *Advances in Intelligent Systems and Computing*, pages 71–80. Springer International Publishing, 2014.

[19] Alex dos Reis de Souza, Denis Efimov, Tarek Raïssi, and Xubin Ping. Robust output feedback model predictive control for constrained linear systems via interval observers. *Automatica*, 135:109951, 2022.

[20] Matthew Ellis, Helen Durand, and Panagiotis D. Christofides. A tutorial review of economic model predictive control methods. *Journal of Process Control*, 24(8):1156–1178, 2014. Economic nonlinear model predictive control.

[21] Hongbing Fang, Qian Wang, Yi-Cheng Tu, and Mark Horstemeyer. An efficient non-dominated sorting method for evolutionary algorithms. *Evolutionary computation*, 16:355–84, 02 2008.

[22] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. 01 2002.

[23] Michael Forbes, Rohit Patwardhan, Hamza Hamadah, and Bhushan Gopaluni. Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, 48:531–538, 12 2015.

[24] A. Gutierrez, E. Irigoyen, E. Larzabal, J. J. Valera, and M. Larrea. Primeros resultados de un control genético predictivo sobre maqueta de helicóptero (twinrotor), 2014.

[25] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[26] Zhangjun Huang, Chengen Wang, and Hong Tian. A genetic algorithm with constrained sorting method for constrained optimization problems. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 806–811, 2009.

[27] Himanshu Jain and Kalyanmoy Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation*, 18(4):602–622, 2014.

[28] Mikkel Jensen. Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms. *Evolutionary Computation, IEEE Transactions on*, 7:503 – 515, 11 2003.

[29] Voratas Kachitvichyanukul. Comparison of three evolutionary algorithms: Ga, pso, and de. *Industrial Engineering and Management Systems*, 12:215–223, 09 2012.

[30] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2219):20180335, nov 2018.

[31] W.B. Langdon and Riccardo Poli. Evolving problems to learn about particle swarm optimizers and other search algorithms. *Evolutionary Computation, IEEE Transactions on*, 11:561 – 578, 11 2007.

[32] M. Larrea, E. Irigoyen, V. Gomez, and F. Artaza. Nonlinear system control based on neural networks with adaptive predictive strategy. *Proceedings of the 2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, page 7 pp., 2010.

[33] M. Larrea, E. Irigoyen, and V. Gómez. Adding nonlinear system dynamics to levenberg-marquardt algorithm for neural network control. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6354 LNCS(PART 3):352–357, 2010.

[34] M. Larrea, E. Larzabal, E. Irigoyen, J.J. Valera, and M. Dendaluce. Implementation and testing of a soft computing based model predictive control on an industrial controller. *Journal of Applied Logic*, 13(2):114–125, 2015.

[35] Mikel Larrea, Eloy Irigoyen, and Vicente Gómez. Adding nonlinear system dynamics to levenberg-marquardt algorithm for neural network control. In *ICANN 2010*, volume 6354 of *Lecture Notes in Computer Science*, pages 352–357. 2010.

[36] Mikel Larrea, Ekaitz Larzabal, Eloy Irigoyen, Juan José Valera García, and Martin Dendaluce. Implementation and testing of a soft computing based model predictive control on an industrial controller. *J. Appl. Log.*, 13:114–125, 2015.

[37] Mikel Larrea Álava, Eloy Irigoyen Gordo, Vicente Gómez Garay, and Fernando Artaza Fano. *Diseño de un sistema de control inteligente basado en redes neuronales para una planta real multivariable*. 2009.

[38] E. Larzabal, J. A. Cubillos, M. Larrea, E. Irigoyen, and J. J. Valera. *Soft Computing Testing in Real Industrial Platforms for Process Intelligent Control*, volume 188 of *Advances in Intelligent Systems and Computing*, pages 221–230. 2013.

[39] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE Transactions on Evolutionary Computation*, 13(2):284–302, April 2009.

[40] José M Maestre, Rudy R Negenborn, et al. *Distributed model predictive control made easy*, volume 69. Springer, 2014.

[41] D.L. Marruedo, T. Alamo, and E.F. Camacho. Input-to-state stable mpc for constrained discrete-time nonlinear systems with bounded additive uncertainties. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 4, pages 4619–4624 vol.4, 2002.

[42] Mathworks. Design Time Series NARX Feedback Neural Networks. `https://es.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html`. [Online; accessed 5-August-2022].

[43] MathWorks. Generating Code from a Level-2 MATLAB S-Function. https://es.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html#f7-68222. [Online; accessed 5-August-2022].

[44] MathWorks. Pricing and Licensing. https://www.mathworks.com/pricing-licensing.html?prodcode=WT. [Online; accessed 5-August-2022].

[45] Mathworks. Real-Time Execution in Connected IO Mode. https://es.mathworks.com/help/sldrt/ug/simulink-real-time-connected-io-mode.html. [Online; accessed 5-August-2022].

[46] Mathworks. Real-Time Execution in Run in Kernel Mode. https://es.mathworks.com/help/sldrt/ug/simulink-run-in-kernel-mode.html. [Online; accessed 5-August-2022].

[47] Mathworks. Simulink Desktop Real-Time. https://es.mathworks.com/products/simulink-desktop-real-time.html. [Online; accessed 5-August-2022].

[48] MathWorks. Simulink Engine Interaction with C S-Functions. https://es.mathworks.com/help/simulink/sfg/how-the-simulink-engine-interacts-with-c-s-functions.html. [Online; accessed 5-August-2022].

[49] Mathworks. Simulink PLC Coder. https://es.mathworks.com/products/simulink-plc-coder.html?s_tid=srchtitle_PLC%20Code_1. [Online; accessed 5-August-2022].

[50] Mathworks. Simulink Real-Time. https://es.mathworks.com/products/simulink-real-time.html. [Online; accessed 5-August-2022].

[51] MathWorks. What is a S-Function. https://es.mathworks.com/help/simulink/sfg/what-is-an-s-function.html. [Online; accessed 5-August-2022].

[52] David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[53] D.Q. Mayne. Optimization in model based control1. *IFAC Proceedings Volumes*, 28(9):229–242, 1995. 4th IFAC Symposium on Dynamics and Control of Chemical Reactors, Distillation Columns and Batch Processes (DYCORD+ '95), Helsingør, Denmark, 7-9 June, 1995.

[54] Zbigniew Michalewicz and Cezary Z. Janikow. Genetic algorithms for numerical optimization. *Statistics and Computing*, 1(2):75–91, Dec 1991.

[55] K.S. Narendra and K. Parthasarathy. Learning automata approach to hierarchical multiobjective analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(1):263–272, Jan 1991.

[56] Gilberto Pin, Davide M. Raimondo, Lalo Magni, and Thomas Parisini. Robust model predictive control of nonlinear systems with bounded and state-dependent uncertainties. *IEEE Transactions on Automatic Control*, 54(7):1681–1687, 2009.

[57] G Prasad, GW Irwin, E Swidenbank, and BW Hogg. A complete formulation of non-linear model-based stable predictive control strategy. In *Unknown Host Publication*, pages 3458–3463. European Control Conference, September 2001. European Control Conference (ECC 01) ; Conference date: 01-09-2001.

[58] S. Joe Qin and Thomas A. Badgwell. An overview of nonlinear model predictive control applications. In *Nonlinear Predictive Control*, pages 369–392. Verlag, 2000.

[59] J. Ronkkonen, S. Kukkonen, and K.V. Price. Real-parameter optimization with differential evolution. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 506–513 Vol.1, 2005.

[60] Peter Schmid and Jörn Sesterhenn. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656, 11 2008.

[61] Maarten Schoukens and Koen Tiels. Identification of block-oriented nonlinear systems starting from linear approximations: A survey. *Autom.*, 85:272–292, 2017.

[62] J. Sjöberg, H. Hjalmarsson, and L. Ljung. Neural networks in system identification. *IFAC Proceedings Volumes*, 27(8):359–382, 1994. IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July.

[63] Speedgoat. Choice of Form Factors. `https://www.speedgoat.com/products-services/real-time-target-machines/baseline-real-time-target-machine/capabilities`. [Online; accessed 5-August-2022].

[64] N. Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, sep 1994.

[65] Andrew M. Sutton, Monte Lunacek, and L. Darrell Whitley. Differential evolution and non-separability: Using selective pressure to focus search. In *Proceedings of GECCO 2007*, Proceedings of GECCO 2007: Genetic and Evolutionary Computation Conference, pages 1428–1435, 2007. 9th Annual Genetic and Evolutionary Computation Conference, GECCO 2007 ; Conference date: 07-07-2007 Through 11-07-2007.

[66] Khoa Tran. An improved non-dominated sorting genetic algorithm-ii (ansga-ii) with adaptable parameters. *IJISTA*, 7:347–369, 01 2009.

[67] J. Valera, E. Irigoyen, V. Gomez-Garay, and F. Artaza. Aproximación al control predictivo no lineal y multiobjetivo usando técnicas de computación inteligente, 2011.

[68] Juan José Valera García, Vicente Gómez Garay, Eloy Irigoyen Gordo, Fernando Artaza Fano, and Mikel Larrea Sukia. Intelligent multi-objective nonlinear model predictive control (imo-nmpc): Towards the 'on-line' optimization of highly complex control problems. *Expert Systems with Applications*, 39(7):6527–6540, 2012.

[69] J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 1980–1987 Vol.2, 2004.

[70] J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 1980–1987 Vol.2, 2004.

[71] Wikipedia. Volterra Series. `https://en.wikipedia.org/wiki/Volterra_series`. [Online; accessed 5-August-2022].

[72] Asier Zabaljauregi. sfun_imo_nmpc. `https://github.com/GICI-UPV-EHU/sfun_imo_nmpc`. [Online; accessed 5-August-2022].

[73] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[74] Aimin Zhou, Qingfu Zhang, and Guixu Zhang. A multiobjective evolutionary algorithm based on decomposition and probability model. pages 1–8, 06 2012.

	SISO Sim (pop = 200, gen = 200) Hc = Hp	MSE	MaxAbsE
SNL1	2	0,0535	1,5999
	4	0,0501	1,543
	6	0,0526	1,5411
SNL2	2	0,0511	1,3601
	4	0,046	1,1648
	6	0,0444	1,1542
SNL5	2	0,0524	1,3958
	4	0,0542	1,4285
	6	0,0558	1,3

	SISO Sim (pop = 200, gen = 200) Hc = Hp	MSE	MaxAbsE
SNL2_NN	2	0,0058	0,3026
	4	0,0046	0,2794
	6	0,0043	0,2542

Tabla 12: Tabla de error cuadrático medio y máximo absoluto para simulaciones SISO

	SISO Sim + pert (pop = 200, gen = 200) Hc = Hp	MSE	MaxAbsE
SNL1	2	0,0066	0,508
	4	0,0071	0,6041
	6	0,0069	0,5818
SNL2	2	0,0072	0,6277
	4	0,0071	0,6164
	6	0,0075	0,6396
SNL5	2	0,0073	0,5
	4	0,0069	0,4983
	6	0,0072	0,5094

	SISO Sim + pert (pop = 200, gen = 200) Hc = Hp	MSE	MaxAbsE
SNL2_NN	2	0,0013	0,1992
	4	0,000937	0,1947
	6	0,000941	0,189

Tabla 13: Tabla de error cuadrático medio y máximo absoluto para simulaciones SISO con perturbación

		MIMO Sim (pop = 200, gen = 400) Hc = Hp	MSE	MaxAbsE
SNL1-SNL1	SNL1-1	2	0,0084	0,6616
		4	0,0199	0,758
		6	0,0162	0,6715
	SNL1-2	2	0,0316	1,6197
		4	0,0424	1,556
		6	0,0491	1,5262
SNL1-SNL5	SNL1	2	0,0082	0,6481
		4	0,0168	0,7108
		6	0,0231	0,752
	SNL5	2	0,0252	0,9743
		4	0,0541	1,4045
		6	0,0532	1,3352

Tabla 14: Tabla de error cuadrático medio y máximo absoluto para simulaciones MIMO en seguimiento

		MIMO Sim + pert (pop = 200, gen = 400) Hc = Hp	MSE	MaxAbsE
SNL1-SNL1	SNL1-1	2	0,0028	0,5007
		4	0,0029	0,4996
		6	0,0031	0,4856
	SNL1-2	2	0,0029	0,5
		4	0,0032	0,4943
		6	0,0033	0,4914
SNL1-SNL5	SNL1	2	0,0034	0,5004
		4	0,0039	0,5047
		6	0,0041	0,5074
	SNL5	2	0,0047	0,5
		4	0,0051	0,4997
		6	0,0065	0,4287

MIMO Sim + pert (ponderación nula coste energético)	
MSE	MaxAbsE
0,0028	0,4997
0,0042	0,4999

Tabla 15: Tabla de error cuadrático medio y máximo absoluto para simulaciones MIMO con perturbación

```
function yf = EvalObj_vec_AZ (xf)

global ref;
global uk;
global yk;

global k;
global H;
global Hc;

global orden_sys;
global epred;

[numP,~] = size(xf);

yp_1 = zeros(numP,H+orden_sys+1);
yp_2 = zeros(numP,H+orden_sys+1);

up_1 = zeros(numP,H+orden_sys);
up_2 = zeros(numP,H+orden_sys);

yp_1(:,1:orden_sys) = yk(1,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);
yp_2(:,1:orden_sys) = yk(2,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);

up_1(:,1:orden_sys) = uk(1,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);
up_2(:,1:orden_sys) = uk(2,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);

% | up(k) up(k+1) ... up(k+Hc) up(k+Hc) ... up(k+Hc)|

up_1(:,orden_sys+1:Hc+orden_sys) = xf(:,1:Hc);
up_1(:,orden_sys+Hc+1:orden_sys+H) = xf(:,Hc).*ones(numP,H-Hc);
up_2(:,orden_sys+1:Hc+orden_sys) = xf(:,Hc+1:end);
up_2(:,orden_sys+Hc+1:orden_sys+H) = xf(:,end).*ones(numP,H-Hc);

for i=(orden_sys+1):(orden_sys+1+H)
    yp_1(:,i) = SNL5_fun([yp_1(:,i-1) yp_1(:,i-2)],up_1(:,i-1)) + epred(1).*
    ones(numP,1);
    yp_2(:,i) = SNL1_fun(yp_2(:,i-1),up_2(:,i-1)) + epred(2)*ones(numP,1);
end

ref_h_1 = repmat(ref(1,k:k+H-1),[numP 1]);
ref_h_2 = repmat(ref(2,k:k+H-1),[numP 1]);

sum_e_1 = sum((ref_h_1 - yp_1(:,orden_sys+1:H+orden_sys)).^2,2);
sum_e_2 = sum((ref_h_2 - yp_2(:,orden_sys+1:H+orden_sys)).^2,2);

sum_u_1 = sum((up_1(:,orden_sys+1:end) - up_1(:,orden_sys:end-1)).^2,2);
sum_u_2 = sum((up_2(:,orden_sys+1:end) - up_2(:,orden_sys:end-1)).^2,2);


% yf(:,1) = sum_e_1 + sum_e_2;
% yf(:,2) = sum_u_1 + sum_u_2;
% 4 objetivos separados
yf(:,1) = sum_e_1;
yf(:,2) = sum_e_2;
yf(:,3) = sum_u_1;
yf(:,4) = sum_u_2;
```

Código 10.1: EvalObj_vec_AZ.m

```
function yf = EvalObj_vec_NN_AZ (xf)

global ref;
global uk;
global yk;

global k;
global H;
global Hc;

global orden_sys;
global epred;

global S1;
global S2;


% ------------------------------------------------------
% obtener numero de individuos entregados por el GA
[numP,~] = size(xf);
% generar vectores yp y up

% | yp(k-2) yp(k-1)  ...    ....   yp(k+H)  |

% | up(k-2) up(k-1)  ... up(k+H-1)|


yp_1 = zeros(numP,H+orden_sys+1);
yp_2 = zeros(numP,H+orden_sys+1);

up_1 = zeros(numP,H+orden_sys);
up_2 = zeros(numP,H+orden_sys);

yp_1(:,1:orden_sys) = yk(1,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);
yp_2(:,1:orden_sys) = yk(2,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);

up_1(:,1:orden_sys) = uk(1,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);
up_2(:,1:orden_sys) = uk(2,(k - orden_sys):(k - orden_sys + 1)).*ones(numP,
    orden_sys);

% | up(k) up(k+1) ... up(k+Hc) up(k+Hc) ... up(k+Hc)|

up_1(:,orden_sys+1:Hc+orden_sys) = xf(:,1:Hc);
up_1(:,orden_sys+Hc+1:orden_sys+H) = xf(:,Hc).*ones(numP,H-Hc);
up_2(:,orden_sys+1:Hc+orden_sys) = xf(:,Hc+1:end);
up_2(:,orden_sys+Hc+1:orden_sys+H) = xf(:,end).*ones(numP,H-Hc);

up_net_1 = 0;
up_net_2 = 0;

yp_net_1 = 0;
yp_net_2 = 0;

% ------------ prediccion modelo neuronal -----------

% up_net = (up - u_offset).*u_gain - 1;
% yp_net = (yp - y_offset).*y_gain - 1;


a1 = zeros(numP*S1.nnarq(1),H+1);
```

```
60 o1 = zeros(numP,H+1);
61
62 a2 = zeros(numP*S2.nnarq(1),H+1);
63 o2 = zeros(numP,H+1);
64
65 % normalizar up
66 up_net_1 = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,up_1,S1.i_offset(:,1)),S1.
       i_gain(:,1)),S1.i_min(:,1));
67 up_net_2 = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,up_2,S2.i_offset(:,1)),S2.
       i_gain(:,1)),S2.i_min(:,1));
68 for j = orden_sys+1:H+orden_sys
69     yp_net_1 = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,yp_1,S1.i_offset(:,2)),
       S1.i_gain(:,2)),S1.i_min(:,2));
70     yp_net_2 = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,yp_2,S2.i_offset(:,2)),
       S2.i_gain(:,2)),S2.i_min(:,2));
71     for i = 1:numP
72         a1((i-1)*S1.nnarq(1)+1:(i-1)*S1.nnarq(1)+S1.nnarq(1),j) = tansig(S1.I_W
       {1}*up_net_1(i,j:-1:j-1)' + S1.I_W{2}*yp_net_1(i,j-1:-1:j-2)' +S1.I_B);
73         a2((i-1)*S2.nnarq(1)+1:(i-1)*S2.nnarq(1)+S2.nnarq(1),j) = tansig(S2.I_W
       {1}*up_net_2(i,j:-1:j-1)' + S2.I_W{2}*yp_net_2(i,j-1:-1:j-2)' +S2.I_B);
74
75         o1(i,j) = purelin(S1.O_W{1}*a1((i-1)*S1.nnarq(1)+1:(i-1)*S1.nnarq(1)+S1
       .nnarq(1),j) + S1.O_B);
76         o2(i,j) = purelin(S2.O_W{1}*a2((i-1)*S2.nnarq(1)+1:(i-1)*S2.nnarq(1)+S2
       .nnarq(1),j) + S2.O_B);
77     end
78     o1 = bsxfun(@plus,bsxfun(@rdivide,bsxfun(@minus,o1,S1.o_min),S1.o_gain),S1.
       o_offset);
79     o2 = bsxfun(@plus,bsxfun(@rdivide,bsxfun(@minus,o2,S2.o_min),S2.o_gain),S2.
       o_offset);
80     yp_1(:,j) = o1(:,j) + epred(1,1);
81     yp_2(:,j) = o2(:,j) + epred(2,1);
82 end
83
84
85
86 ref_h_1 = repmat(ref(1,k:k+H-1),[numP 1]);
87 ref_h_2 = repmat(ref(2,k:k+H-1),[numP 1]);
88
89 % compute error and delta_u for  k : k + H
90 %
91 sum_e_1 = sum((ref_h_1 - yp_1(:,orden_sys+1:H+orden_sys)).^2,2);
92 sum_e_2 = sum((ref_h_2 - yp_2(:,orden_sys+1:H+orden_sys)).^2,2);
93
94 sum_u_1 = sum((up_1(:,orden_sys+1:end) - up_1(:,orden_sys:end-1)).^2,2);
95 sum_u_2 = sum((up_2(:,orden_sys+1:end) - up_2(:,orden_sys:end-1)).^2,2);
96
97 % 4 objetivos separados
98 yf(:,1) = sum_e_1;
99 yf(:,2) = sum_e_2;
100 yf(:,3) = sum_u_1;
101 yf(:,4) = sum_u_2;
```

Código 10.2: EvalObj_vec_NN_AZ.m

```
function yf = EvalObj_vec_NN_AZ (xf)

global ref;
global uk;
global yk;

global k;
global H;
global Hc;

global orden_sys;
global epred;

global S1;

% ---------------------------------------------------------
% obtener numero de individuos entregados por el GA
[numP,~] = size(xf);

up = zeros(S1.iport_sizes(1)*numP,H+orden_sys);
yp = zeros(S1.iport_sizes(2)*numP,H+orden_sys);

up(:,1:orden_sys) = repmat(uk(:,(k - orden_sys):(k - 1)),[numP 1]);
yp(:,1:orden_sys) = repmat(yk(:,(k - orden_sys):(k - 1)),[numP 1]);

% | up(k) up(k+1) ... up(k+Hc) up(k+Hc) ... up(k+Hc)|

up(:,orden_sys+1:orden_sys+Hc) = reshape(xf',Hc,[])';
up(:,orden_sys+Hc+1:orden_sys+H) = repmat(up(:,orden_sys+Hc),[1,H-Hc]);

% ------------ prediccion modelo neuronal -----------

% up_net = (up - u_offset).*u_gain - 1;
% yp_net = (yp - y_offset).*y_gain - 1;


a1 = zeros(numP*S1.nnarq(1),H);
o1 = zeros(numP*S1.oport_size,H);


% normalizar up
up_net = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,up,repmat(S1.i_offset(:,1),[
    numP 1])),repmat(S1.i_gain(:,1),[numP 1])),repmat(S1.i_min(:,1),[numP*2 1])
    );
for j = orden_sys+1:H+orden_sys
    yp_net = bsxfun(@plus,bsxfun(@times,bsxfun(@minus,yp,repmat(S1.i_offset
    (:,2),[numP 1])),repmat(S1.i_gain(:,2),[numP 1])),repmat(S1.i_min(:,2),[
    numP*2 1]));
    for i = 1:numP
        temp_u = [up_net(2*i-1:2*i,j); up_net(2*i-1:2*i,j-1)];
        temp_y = [yp_net(2*i-1:2*i,j-1); yp_net(2*i-1:2*i,j-2)];

        a1((i-1)*S1.nnarq(1)+1:(i-1)*S1.nnarq(1)+S1.nnarq(1),j) = tansig(S1.I_W
    {1}*temp_u + S1.I_W{2}*temp_y +S1.I_B);

        o1(2*i-1:2*i,j) = purelin(S1.O_W{1}*a1((i-1)*S1.nnarq(1)+1:(i-1)*S1.
    nnarq(1)+S1.nnarq(1),j) + S1.O_B);
    end
    o1 = bsxfun(@plus,bsxfun(@rdivide,bsxfun(@minus,o1,repmat(S1.o_min,[numP*2
    1])),repmat(S1.o_gain,[numP 1])),repmat(S1.o_offset,[numP 1]));
    yp(:,j) = o1(:,j) + repmat(epred,[numP 1]);
end

```

```
57
58 ref_h_1 = repmat(ref(1,k:k+H-1),[numP 1]);
59 ref_h_2 = repmat(ref(2,k:k+H-1),[numP 1]);
60
61 % compute error and delta_u for  k + 1 : k + H
62 %
63 sum_e_1 = sum((ref_h_1 - yp(1:2:end-1,orden_sys+1:H+orden_sys)).^2,2);
64 sum_e_2 = sum((ref_h_2 - yp(2:2:end,orden_sys+1:H+orden_sys)).^2,2);
65
66 sum_u_1 = sum((up(1:2:end-1,orden_sys+1:end) - up(1:2:end-1,orden_sys:end-1))
       .^2,2);
67 sum_u_2 = sum((up(2:2:end,orden_sys+1:end) - up(2:2:end,orden_sys:end-1)).^2,2)
       ;
68
69 %
70 % yf(:,1) = sum_e_1 + sum_e_2;
71 % yf(:,2) = sum_u_1 + sum_u_2;
72
73 % 4 objetivos separados
74 yf(:,1) = sum_e_1;
75 yf(:,2) = sum_e_2;
76 yf(:,3) = sum_u_1;
77 yf(:,4) = sum_u_2;
```

Código 10.3: EvalObj_vec_NN_combinada_AZ.m

```
1  function msfun_pid_cont_AZ(block)
2      % https://lost-contact.mit.edu/afs/inf.ed.ac.uk/group/teaching/matlab-help/
       R2016b/simulink/slref/pidcontroller.html
3      % Matlab parallel filtered PID controller implementation.
4      % Asier Zabaljauregi 2022
5
6      setup(block);
7
8      %% Setup del bloque
9      function setup(block)
10
11     % Parametrizar Kc, Ti , Td , N
12     block.NumDialogPrms = 4;
13
14     % Configurar número de puertos y propiedades
15     block.NumInputPorts = 1;
16     block.NumOutputPorts = 1;
17
18     block.SetPreCompInpPortInfoToDynamic;
19     block.SetPreCompOutPortInfoToDynamic;
20
21     block.InputPort(1).Dimensions = 1;
22     block.InputPort(1).DirectFeedthrough = true;
23
24     block.OutputPort(1).Dimensions = 1;
25
26     % Tiempo de muestreo continuo
27     block.SampleTimes = [0 0];
28
29     % Crear estado continuo
30     block.NumContStates = 2;
31
32     % Configurar "compliance" y métodos
33     block.SimStateCompliance = "DefaultSimState";
34
35     block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
36     block.RegBlockMethod("InitializeConditions",@InitConditions);
37     block.RegBlockMethod("CheckParameters",@CheckPar);
```

```
38      block.RegBlockMethod("Outputs",@Outputs);
39      block.RegBlockMethod("Derivatives",@Derivatives);
40
41      %% Definir métodos
42      function DoPostPropSetup(block)
43          %% Setup Dwork
44          block.NumDworks = 1;
45
46          block.Dwork(1).Name = ('params');
47          block.Dwork(1).Dimensions      = 4; %[Kc,Ki,Kd,N]
48          block.Dwork(1).DatatypeID      = 0;
49          block.Dwork(1).Complexity      = 'Real';
50          block.Dwork(1).UsedAsDiscState = false;
51
52      function InitConditions(block)
53          block.Dwork(1).Data = [block.DialogPrm(1).Data ...
54                                 block.DialogPrm(2).Data ...
55                                 block.DialogPrm(3).Data ...
56                                 block.DialogPrm(4).Data];
57
58          block.ContStates.Data = [0 0];
59
60
61      function CheckPar(block)
62          Ki = block.DialogPrm(2).Data;
63          if Ki < 0
64              error('Ki must be greater or equal to zero');
65          end
66
67      function Outputs(block)
68          Ke = block.Dwork(1).Data(1)*block.InputPort(1).Data; % Kc*E(s)
69          Ie = block.ContStates.Data(1); % Ki*E(s)/s
70          De = block.Dwork(1).Data(4)*(block.InputPort(1).Data*block.Dwork(1).
        Data(3) ...
71          - block.ContStates.Data(2)); % N*(E(s)*Kd - Yd(s)/s))
72
73          block.OutputPort(1).Data = Ke + Ie + De;
74
75      function Derivatives(block)
76          block.Derivatives.Data(1) = block.InputPort(1).Data*block.Dwork(1).Data
        (2); % E(s)*Ki
77          block.Derivatives.Data(2) = block.Dwork(1).Data(4)*(block.InputPort(1).
        Data*block.Dwork(1).Data(3) ...
78                                                              - block.ContStates.
        Data(2));
```

Código 10.4: msfun_pid_cont_AZ.m

```c
/*
* sfun_PID_simple_AZ: my trial at learning basic Level 2 CMEX file structure.
Implementation of a simple PID structure.
* 28/02/2022
*/

#define S_FUNCTION_NAME sfun_PID_simple_AZ
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define U(element) (*uPtrs[element])

#define KP_IDX 0
#define KP_PARAM(S) ssGetSFcnParam(S, KP_IDX)

#define KI_IDX 1
#define KI_PARAM(S) ssGetSFcnParam(S, KI_IDX)

#define KD_IDX 2
#define KD_PARAM(S) ssGetSFcnParam(S, KD_IDX)

#define N_IDX 3
#define N_PARAM(S) ssGetSFcnParam(S, N_IDX)

#define X0_IDX 4
#define X0_PARAM(S) ssGetSFcnParam(S, X0_IDX)

#define NPARAMS 5

#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
static void mdlCheckParameters(SimStruct *S)
{
    {
        int iParam = 0;
        int nParam = ssGetNumSFcnParams(S);
        for (iParam = 0; iParam < nParam; iParam++)
        {
            if (!mxIsDouble(ssGetSFcnParam(S, iParam)) ||
                mxGetNumberOfElements(ssGetSFcnParam(S, iParam)) != 1)
            {
                ssSetErrorStatus(S, "parameter to S-function must be a "
                                    "scalar");
                return;
            }
        }
    }
}
#endif

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NPARAMS); /* number of expected parameters */
#if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
    {
        mdlCheckParameters(S);
        if (ssGetErrorStatus(S) != NULL)
        {
            return;
        }
    }
```

```
64      else
65      {
66          return;
67      }
68  #endif
69      {
70          int iParam = 0;
71          int nParam = ssGetNumSFcnParams(S);
72
73          for (iParam = 0; iParam < nParam; iParam++)
74          {
75              ssSetSFcnParamTunable(S, iParam, SS_PRM_SIM_ONLY_TUNABLE);
76              /* pecify whether a user can change a dialog parameter during the
    simulation*/
77          }
78      }
79
80      ssSetNumContStates(S, 2);
81      ssSetNumDiscStates(S, 0);
82
83      if (!ssSetNumInputPorts(S, 1))
84          return; /* args: (1) Simstruct (2) number of inputs */
85      /* returns true if successful, in this case stops execution for non-
    positive inputs*/
86      ssSetInputPortWidth(S, 0, 1);
87      // ssSetInputPortRequiredContiguous(S,0,true); /* direct input signal
    access*/
88      ssSetInputPortDirectFeedThrough(S, 0, 1); /* args (1) Simstruct (2) number
    of the port (3) 1 yes, 0 no */
89
90      if (!ssSetNumOutputPorts(S, 1))
91          return;
92      /* returns true if successful, in this case stops execution for non-
    positive inputs*/
93      ssSetOutputPortWidth(S, 0, 1);
94
95      ssSetNumSampleTimes(S, 1);
96      /*set the number of sample times S has*/
97      ssSetNumDWork(S, 1);
98      ssSetDWorkWidth(S, 0, 4); // Kp, Ki, Kd, N
99      ssSetDWorkDataType(S, 0, SS_DOUBLE);
100
101     ssSetNumRWork(S, 0);          /*Number of elements in the floating-point
    work vector can be dinamically sized*/
102     ssSetNumIWork(S, 0);          /*specify the number of int_T work vector
    elements as  0*/
103     ssSetNumPWork(S, 0);          /*specify the number of pointer (void *) work
    vector elements as 0*/
104     ssSetNumModes(S, 0);          /* elements of the mode vector*/
105                                   /* this with zero-crossing detection allows
    for blocks with
106                                    * distinct operating modes, depending on the
    current
107                                    * value of the signals
108                                    */
109     ssSetNumNonsampledZCs(S, 0); /* Specify the number of states for which a
    block detects
110                                    * zero crossings that occur between sample
    point
111                                    */
112     /* Specify the operating point save/restore compliance to be same as a
113      * built-in block */
114     ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);
```

```
115     ssSetRuntimeThreadSafetyCompliance(S, RUNTIME_THREAD_SAFETY_COMPLIANCE_TRUE
        );
116     ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
117     /* There are several options to choose from depending on the function*/
118 }
119
120 static void mdlInitializeSampleTimes(SimStruct *S)
121 {
122     ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
123     /* [CONTINUOUS_SAMPLE_TIME, 0.0]
124      * [CONTINUOUS_SAMPLE_TIME, FIXED_IN_MINOR_STEP_OFFSET]
125      * [discrete_sample_time_period, offset]
126      * [VARIABLE_SAMPLE_TIME, 0.0]
127      *
128      * lowercase options are to be filled.
129      */
130     ssSetOffsetTime(S, 0, 0.0);
131     ssSetModelReferenceSampleTimeDefaultInheritance(S);
132     /* inherit its sample time from its parent model*/
133 }
134
135 #define MDL_INITIALIZE_CONDITIONS
136 #if defined(MDL_INITIALIZE_CONDITIONS)
137 static void mdlInitializeConditions(SimStruct *S)
138 {
139     real_T *params = (real_T *)ssGetDWork(S, 0);
140     params[0] = *mxGetPr(KP_PARAM(S));
141     params[1] = *mxGetPr(KI_PARAM(S));
142     params[2] = *mxGetPr(KD_PARAM(S));
143     params[3] = *mxGetPr(N_PARAM(S));
144
145     real_T x0_par = *mxGetPr(X0_PARAM(S));
146
147     size_t i;
148     real_T *x0 = ssGetContStates(S);
149     int_T nContStates = ssGetNumContStates(S);
150
151     for (i = 0; i < (size_t)nContStates; i++)
152     {
153         x0[i] = x0_par;
154     }
155 }
156 #endif
157
158 #undef MDL_START
159 #if defined(MDL_START)
160 static void mdlStart(Simstruct *S)
161 {
162 }
163 #endif
164
165 static void mdlOutputs(SimStruct *S, int_T tid)
166 {
167     real_T *y = ssGetOutputPortRealSignal(S, 0);
168     real_T *x = ssGetContStates(S);
169     real_T *params = (real_T *)ssGetDWork(S, 0);
170
171     real_T Kp = params[0];
172     real_T Ki = params[1];
173     real_T Kd = params[2];
174     real_T N = params[3];
175
176     InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S, 0);
```

```
177     y[0] = Ki * x[0] + Kp * U(0) + N * (Kd * U(0) - x[1]); /* assign to output
        n0 state n0 */
178 }
179
180 #undef MDL_UPDATE
181 #if defined(MDL_UPDATE)
182 static void mdlUpdate(SimStruct *S, int_T tid)
183 {
184 }
185 #endif
186
187 #define MDL_DERIVATIVES
188 #if defined(MDL_DERIVATIVES)
189 static void mdlDerivatives(SimStruct *S)
190 {
191     real_T *params = (real_T *)ssGetDWork(S, 0);
192     real_T Kd = params[2];
193     real_T N = params[3];
194     real_T *x = ssGetContStates(S);
195     real_T *dx = ssGetdX(S);
196     InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S, 0);
197     dx[0] = U(0);
198     dx[1] = N * (U(0) * Kd - x[1]);
199 }
200 #endif
201
202 static void mdlTerminate(SimStruct *S)
203 {
204 }
205
206 #ifdef MATLAB_MEX_FILE
207 #include "simulink.c"
208 #else
209 #include "cg_sfun.h"
210 #endif
```

Código 10.5: sfun_PID_simple_AZ.c

```c
/*
*  naive Dynamic Matrix Control implementation
*  for simulink s-functions.
*  28/03/2022.
*
*  Asier Zabaljauregi
*/

/*
*         +----------------+
*         |                |
*    y--->|      DMC       |------> u
*         |                |
*         +----------------+
*
*     >>mex sfun_dmc_debug_AZ.c picoblas.c
*     >>load_dmc_params
*/


#define S_FUNCTION_NAME sfun_dmc_debug_v2_AZ
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include "picoblas_matlab.h"

extern enum states state;

#define Y(element) (*uPtrs[element])

#define HP  mxGetPr(H_P_PARAM(S))[0]
#define HC  mxGetPr(H_C_PARAM(S))[0]


#define SAMPLE_T 0
#define SAMPLE_PARAM(S) ssGetSFcnParam(S,SAMPLE_T) // sample time

#define H_PRED 1
#define H_P_PARAM(S) ssGetSFcnParam(S,H_PRED)      // prediction horizon

#define H_CONTROL 2
#define H_C_PARAM(S) ssGetSFcnParam(S,H_CONTROL)   // control horizon

#define G_STEP 3
#define G_PARAM(S) ssGetSFcnParam(S,G_STEP)        // step ponderation sequence
#define G_N mxGetN(G_PARAM(S))                     // trucation index

#define REF 4
#define REF_PARAM(S) ssGetSFcnParam(S,REF)         // reference
#define R_N mxGetN(REF_PARAM(S))                   // reference size

#define LAMBDA 5
#define LAMBDA_PARAM(S) ssGetSFcnParam(S,LAMBDA)      // delta_u error
    ponderation

#define NPARAMS 6

#define IS_PARAM_DOUBLE(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
!mxIsEmpty(pVal) && !mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal)
    )

#define IS_PARAM_INTEGER(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
!mxIsEmpty(pVal) && !mxIsSparse(pVal) && !mxIsComplex(pVal))
```

```
62
63 #define OK_EMPTY_DOUBLE_PARAM(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal)
       &&\
64 !mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal))
65
66
67 #define MDL_CHECK_PARAMETERS
68 #if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
69 static void mdlCheckParameters(SimStruct *S)
70 {
71     {
72       if(!IS_PARAM_DOUBLE(SAMPLE_PARAM(S)) || !(mxGetPr(SAMPLE_PARAM(S))[0] >=
       0.0) ){
73         ssSetErrorStatus(S,"Sampling time must be a positive real number.");
74         return;
75       }
76     }
77
78     {
79       if(!IS_PARAM_INTEGER(H_P_PARAM(S)) || !(mxGetPr(H_P_PARAM(S))[0] >= 0.0)
       ){
80         ssSetErrorStatus(S,"Prediction horizon must be a positive integer.");
81         return;
82       }
83     }
84
85     {
86       if(!IS_PARAM_INTEGER(H_C_PARAM(S)) || !(mxGetPr(H_C_PARAM(S))[0] >= 0.0)
       || (mxGetPr(H_C_PARAM(S))[0] > mxGetPr(H_P_PARAM(S))[0])){
87         ssSetErrorStatus(S,"Control horizon must be a positive integer inferior
        to the prediction horizon.");
88         return;
89       }
90     }
91
92     {
93       if(!IS_PARAM_DOUBLE(G_PARAM(S)) || ! OK_EMPTY_DOUBLE_PARAM(G_PARAM(S))){
94         ssSetErrorStatus(S,"Step ponderation sequence must a non-empty double
       vector");
95         return;
96       }
97     }
98
99     {
100      if(!IS_PARAM_DOUBLE(REF_PARAM(S)) || ! OK_EMPTY_DOUBLE_PARAM(REF_PARAM(S)
       )){
101        ssSetErrorStatus(S,"Reference signal must a non-empty double vector");
102        return;
103      }
104    }
105
106    {
107      if(!IS_PARAM_DOUBLE(LAMBDA_PARAM(S)) || !(mxGetPr(LAMBDA_PARAM(S))[0] >=
       0.0) || !(mxGetPr(LAMBDA_PARAM(S))[0] <=1.0) ){
108        ssSetErrorStatus(S,"Lambda (Delta U ponderation value) must be a [0,1]
       range real number.");
109        return;
110      }
111    }
112
113 }
114 #endif
115
```

```
116 static void mdlInitializeSizes(SimStruct *S)
117 {
118     ssSetNumSFcnParams(S, NPARAMS);
119     #if defined(MATLAB_MEX_FILE)
120         if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
121             mdlCheckParameters(S);
122             if (ssGetErrorStatus(S) != NULL){
123                 return;
124             }
125         }else {
126             return; /* Parameter mismatch will be reported by Simulink */
127         }
128     #endif
129 
130     if (!ssSetNumInputPorts(S, 1)) return;
131     ssSetInputPortWidth(S, 0, 1);
132     ssSetInputPortDirectFeedThrough(S, 0, 1);
133 
134     if (!ssSetNumOutputPorts(S, 1)) return;
135     ssSetOutputPortWidth(S, 0, 1);
136 
137     ssSetNumSampleTimes(S, 1);
138 
139      ssSetNumRWork(S, 0);
140      ssSetNumIWork(S, 0);
141      ssSetNumPWork(S, 0);
142      ssSetNumModes(S, 0);
143 
144      ssSetNumDWork(S,6);
145         /*
146         *  ---------CHECK------------------
147         *   if necessary to free(id) before
148         *   allocating DWork
149         *
150         */
151 
152     //K matrix
153     ssSetDWorkWidth(S,0,HP*HC); // hc*hp one dimensional array
154     ssSetDWorkDataType(S,0,SS_DOUBLE);
155     ssSetDWorkName(S,0,"sfcnK");
156 
157     //Past Control Action
158     ssSetDWorkWidth(S,1,G_N - 1); // Nt - 1
159     ssSetDWorkDataType(S,1,SS_DOUBLE);
160     ssSetDWorkName(S,1,"sfcnControl_p");
161 
162     //Reference
163     ssSetDWorkWidth(S,2,R_N);
164     ssSetDWorkDataType(S,2,SS_DOUBLE);
165     ssSetDWorkName(S,2,"sfcnRefvals");
166 
167     // Future Control Action
168     ssSetDWorkWidth(S,3,HC); // hc
169     ssSetDWorkDataType(S,3,SS_DOUBLE);
170     ssSetDWorkName(S,3,"sfcnControl_f");
171 
172     // F matrix
173     ssSetDWorkWidth(S,4,HP*(G_N - 1)); // hc
174     ssSetDWorkDataType(S,4,SS_DOUBLE);
175     ssSetDWorkName(S,4,"sfcnF");
176 
177     // U last buffer
178     ssSetDWorkWidth(S,5,1); // hc
```

```
    ssSetDWorkDataType(S,5,SS_DOUBLE);
    ssSetDWorkName(S,5,"sfcnUlast");



    ssSetNumNonsampledZCs(S, 0);

    ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);
    ssSetRuntimeThreadSafetyCompliance(S, RUNTIME_THREAD_SAFETY_COMPLIANCE_TRUE
    );
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    real_T Ts = *mxGetPr(SAMPLE_PARAM(S));
    ssSetSampleTime(S, 0, Ts);
    ssSetOffsetTime(S, 0, 0.0);

}

#undef MDL_SET_WORK_WIDTHS   /* Change to #undef to remove function */
#if defined(MDL_SET_WORK_WIDTHS) && defined(MATLAB_MEX_FILE)
  static void mdlSetWorkWidths(SimStruct *S)
  {

  }
#endif

#undef MDL_INITIALIZE_CONDITIONS   /* Change to #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  static void mdlInitializeConditions(SimStruct *S)
  {

  }
#endif

#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  static void mdlStart(SimStruct *S)
  {
    const real_T * g_i = mxGetPr(G_PARAM(S));
    real_T * ref = mxGetPr(REF_PARAM(S));
    // Create G matrix
    real_T * G = mat_create(HP,HC);

    for (int_T i = 0; i < HP; i++){
        for (int_T j = 0; j < HC; j++){
            if (j > i) {
                G[i + j*(int)HP] = 0.0;
            } else {
                G[i + j*(int)HP] = g_i[i-j];
            }
        }
    }
    ssPrintf("G matrix: \n");
    for(int_T i = 0; i < HP; i++) {
      for(int_T j = 0; j < HC; j++){
         ssPrintf("%f ",G[i+j*(int)HP]);
      } ssPrintf("\n ");
    }
    // Create F matrix
    real_T * F = mat_create(HP, G_N -1);
```

```
241
242     for (int_T i = 0; i < HP; i++){
243         for (int_T j = 0; j < (G_N - 1); j++){
244             if((i+j) < G_N - 1){
245                 F[i + j*(int)HP] = g_i[j + i + 1] - g_i[j];
246             } else {
247                 F[i + j*(int)HP] = g_i[G_N - 1] - g_i[j];
248             }
249         }
250     }
251     ssPrintf("G_N is: %d \n",G_N);
252     ssPrintf("F matrix: \n");
253     for(int_T i = 0; i < HP; i++) {
254       for(int_T j = 0; j < G_N - 1; j++){
255          ssPrintf("%f ",F[i+j*(int)HP]);
256       } ssPrintf("\n ");
257     }
258
259     // Create K matrix
260     real_T * G_T = mat_transpose(G, HP, HC);
261     real_T * result = mat_mul(G_T,G,HC,HP,HC);
262
263     real_T * I = mat_create_identity(HC);
264     real_T max = mat_max(result,HC,HC);
265     mat_prod(I,mxGetPr(LAMBDA_PARAM(S))[0]*max,HC,HC);
266
267     mat_sum(result,I,HC,HC);
268     ssPrintf("Intermediate: \n");
269     for(int_T i = 0; i < HC; i++){
270       for(int_T j = 0; j < HC; j++){
271         ssPrintf("%f ",result[i+j*(int)HC]);
272       }
273       ssPrintf("\n");
274     }
275     mat_invert(result,HC);
276     real_T * K = mat_mul(result,G_T,HC,HC,HP);
277
278     ssPrintf("K matrix: \n");
279     for(int_T i = 0; i < HC; i++) {
280       for(int_T j = 0; j < HP; j++){
281          ssPrintf("%f ",K[i+j*(int)HC]);
282       } ssPrintf("\n ");
283     }
284     // Save K matrix to DWork
285     real_T * K_D = (real_T *) ssGetDWork(S,0);
286     mat_copy(K,K_D,HC,HP);
287     // Save F matrix to DWork
288     real_T * F_D = (real_T *) ssGetDWork(S,4);
289     //F_D = (real_T * ) F;
290     mat_copy(F,F_D,HP,G_N - 1);
291     // Save Reference to DWork
292     real_T * ref_d = (real_T *) ssGetDWork(S,2);
293     mat_copy(ref,ref_d,1,R_N);
294
295     ssPrintf("Saved reference: ");
296     for(int_T i = 0; i < R_N; i++) ssPrintf("%f ",ref_d[i]);
297
298     // Initialize Past Control actions to 0
299     real_T * U_p  = (real_T * ) ssGetDWork(S,3);
300
301     for(int_T i = 0; i < G_N - 1; i++){
302       U_p[i] = 0;
303     }
```

```
304
305     real_T * u_last = (real_T *) ssGetDWork(S,5);
306     u_last[0] = 0;
307   }
308 #endif
309
310 static void mdlOutputs(SimStruct *S, int_T tid)
311 {
312     InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S, 0);
313     real_T *u = ssGetOutputPortSignal(S,0);
314
315     UNUSED_ARG(tid);
316
317     real_T * K_D = (real_T *) ssGetDWork(S,0);
318     real_T * F_D = (real_T *) ssGetDWork(S,4);
319     real_T * u_last = (real_T *) ssGetDWork(S,5);
320     // create copy of transposed reference
321     real_T * ref = (real_T * )ssGetDWork(S,2);
322     real_T * ref_cp = mat_create_copy(ref,1,HP);
323     real_T * y_vect = mat_create(HP,1);
324
325     for(int_T i = 0; i < HP; i++){
326       y_vect[i] = 1;
327     }
328     mat_prod(y_vect,Y(0),HP,1);
329     //Fill the matrix
330     real_T * U_f = mat_transpose((real_T *) ssGetDWork(S,3),1,HC);
331     real_T * U_p = mat_transpose((real_T *) ssGetDWork(S,1),1,G_N - 1);
332     real_T * result = mat_mul(F_D,U_p,HP,G_N - 1,1); // F*U_p
333     mat_sum(result,y_vect,HP,1); // F*U_p + Y
334     mat_prod(result,-1,HP,1); // - F*U_p - Y
335     mat_sum(ref_cp,result,HP,1); // r - f = r - F*U_p - Y
336     U_f = mat_mul(K_D,ref_cp,HC,HP,1); // K*(r-f)
337     real_T tempU = U_f[0] + u_last[0];
338     u[0] = tempU;
339     u_last[0] = tempU; // u = delta_u+ -
340     // Shift U_p to the past roduce the updated value
341     for(int_T i = 0; i < (G_N - 2); i++){
342       U_p[i+1] = U_p[i];
343     }
344     // Shift reference values to the future
345     U_p[0] = U_f[0];
346     for( int_T i = 0; i < R_N - 1; i++){
347       ref[i] = ref[i+1];
348     }
349 }
350
351 #undef MDL_UPDATE  /* Change to #undef to remove function */
352 #if defined(MDL_UPDATE)
353   static void mdlUpdate(SimStruct *S, int_T tid)
354   {
355   }
356 #endif
357
358 #undef MDL_DERIVATIVES  /* Change to #undef to remove function */
359 #if defined(MDL_DERIVATIVES)
360   static void mdlDerivatives(SimStruct *S)
361   {
362   }
363 #endif
364
365 static void mdlTerminate(SimStruct *S)
366 {
```

```
367     UNUSED_ARG(S);
368 }
369
370 #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
371 #include "simulink.c"      /* MEX-file interface mechanism */
372 #else
373 #include "cg_sfun.h"       /* Code generation registration function */
374 #endif
```

Código 10.6: sfun_dmc_debug_v2_AZ.c

```
1  A = tf([1000],[1 10]);
2
3  g_i = step(A,0:0.05:1);
4  size(g_i)
5  g_i = g_i(2:11,:);
6  size(g_i)
7  g_i = g_i.';
8
9
10 ref = [0.5*ones(30,1); 1*ones(30,1); 1.5*ones(40,1);1*ones(30,1);0.5*ones(30,1)];
11 ref = ref.';
```

Código 10.7: load_params_dmc.m

```
1  /*
2   * sfuntmpl_basic.c: Basic 'C' template for a level 2 S-function.
3   *
4   * Copyright 1990-2018 The MathWorks, Inc.
5   */
6
7  /*
8  * sfun_imo_nmpc.c: S-function implementation of the iMO-NMPC control strategy
9  * Asier Zabaljauregi
10 * 23/05/2022
11 */
12 #define S_FUNCTION_NAME  sfun_imo_nmpc
13 #define S_FUNCTION_LEVEL 2
14
15 /*
16  * Need to include simstruc.h for the definition of the SimStruct and
17  * its associated macro definitions.
18  */
19 #include "simstruc.h"
20 //#include "matrix.h"
21 #include <math.h>
22 #include "global.h"
23 #include "rand.h"
24 #include "decision.h"
25 #include "model.h"
26 //#include "sfun_read_file.h"
27 #define TRUE  1
28 #define FALSE 0
29
30
31 #define SAMPLE_T_N 0
32 #define SAMPLE_T(S) ssGetSFcnParam(S,SAMPLE_T_N) // sample time
33
34 #define SEED_N             1// random seed (0,1)
35 #define SEED(S) ssGetSFcnParam(S,SEED_N)
36 #define POPSIZE_N          2// population size
37 #define POPSIZE(S) ssGetSFcnParam(S,POPSIZE_N)
38 #define NGEN_N             3// generation number
39 #define NGEN(S) ssGetSFcnParam(S,NGEN_N)
```

```
#define NOBJ_N             4// number of objectives
#define NOBJ(S) ssGetSFcnParam(S,NOBJ_N)
#define NCON_N             5// number of constraints
#define NCON(S) ssGetSFcnParam(S,NCON_N)
#define NREAL_N            6// number of real chromosomes
#define NREAL(S) ssGetSFcnParam(S,NREAL_N)
#define MAX_MIN_REALVAR_N  7
#define MAX_MIN_REALVAR(S) ssGetSFcnParam(S,MAX_MIN_REALVAR_N)
#define PCROSS_REAL_N      8//real crossover prob.
#define PCROSS_REAL(S) ssGetSFcnParam(S,PCROSS_REAL_N)
#define PMUT_REAL_N        9 //real mutation prob.
#define PMUT_REAL(S) ssGetSFcnParam(S,PMUT_REAL_N)
#define ETA_C_N           10// distribution index for crossover
#define ETA_C(S) ssGetSFcnParam(S,ETA_C_N)
#define ETA_M_N           11// distribution index for mutation
#define ETA_M(S) ssGetSFcnParam(S,ETA_M_N)
#define NBIN_N            12 // number of binary chromosomes
#define NBIN(S) ssGetSFcnParam(S,NBIN_N)
#define NBITS_N           13
#define NBITS(S) ssGetSFcnParam(S,NBITS_N)
#define MAX_MIN_BINVAR_N  14
#define MAX_MIN_BINVAR(S) ssGetSFcnParam(S,MAX_MIN_BINVAR_N)
#define PCROSS_BIN_N      15// bin crossover prob.
#define PCROSS_BIN(S) ssGetSFcnParam(S,PCROSS_BIN_N)
#define PMUT_BIN_N        16// bin mutation prob.
#define PMUT_BIN(S) ssGetSFcnParam(S,PMUT_BIN_N)

#define HP_N              17
#define HP(S) ssGetSFcnParam(S,HP_N)

#define REF_N             18// reference to follow
#define REF_PARAM(S) ssGetSFcnParam(S,REF_N)
#define R_N mxGetN(REF_PARAM(S))

#define NOUT_N             19// number of outputs
#define NOUT(S) ssGetSFcnParam(S,NOUT_N)

#define IS_PARAM_DOUBLE(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
!mxIsEmpty(pVal) && !mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal))

#define OK_EMPTY_DOUBLE_PARAM(pVal) (mxIsNumeric(pVal) && !mxIsLogical(pVal) &&\
!mxIsSparse(pVal) && !mxIsComplex(pVal) && mxIsDouble(pVal))



double seed;
int nreal;
int hp;
int nout;

int nbin;
int nobj;
int ncon;
int popsize;
double pcross_real;
double pcross_bin;
double pmut_real;
double pmut_bin;
double eta_c;
double eta_m;
int ngen;
int nbinmut;
int nrealmut;
```

```
int nbincross;
int nrealcross;
int *nbits;

int flag_real = TRUE;
double *min_realvar;
double *max_realvar;

int flag_bin = FALSE;
double *min_binvar;
double *max_binvar;
int bitlength;

population *parent_pop;
population *child_pop;
population *mixed_pop;

int * idx;

/* Error handling
 * --------------
 *
 * You should use the following technique to report errors encountered within
 * an S-function:
 *
 *       ssSetErrorStatus(S,"Error encountered due to ...");
 *       return;
 *
 * Note that the 2nd argument to ssSetErrorStatus must be persistent memory.
 * It cannot be a local variable. For example the following will cause
 * unpredictable errors:
 *
 *      mdlOutputs()
 *      {
 *         char msg[256];         {ILLEGAL: to fix use "static char msg[256];"}
 *         sprintf(msg,"Error due to %s", string);
 *         ssSetErrorStatus(S,msg);
 *         return;
 *      }
 *
 */

/*====================*
 * S-function methods *
 *====================*/

/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
static void mdlCheckParameters(SimStruct *S)
{
  if(IS_PARAM_DOUBLE(SAMPLE_T(S))){
   if(mxGetPr(SAMPLE_T(S))[0] <= 0.0){
     ssSetErrorStatus(S,"Sample Time must be greater than 0.");
   }
  } else ssSetErrorStatus(S,"Sample time must be a double.");
  double temp;
  if (IS_PARAM_DOUBLE(SEED(S))){
    if(mxGetPr(SEED(S))[0] <= 0.0 || mxGetPr(SEED(S))[0]>=1.0){
```

```
166       ssSetErrorStatus(S,"Entered seed value is wrong, seed value must be in (0,1) \n");
167     }
168   } else ssSetErrorStatus(S,"Error,Seed must be in (0,1) range.");
169 //
170   if(IS_PARAM_DOUBLE(POPSIZE(S)) && (modf(mxGetPr(POPSIZE(S))[0],&temp)== 0)){
171     if((int)mxGetPr(POPSIZE(S))[0] < 4.0 || ((int)mxGetPr(POPSIZE(S))[0]%4)!= 0){
172       ssSetErrorStatus(S,"Population size must be greater than 4 and multiple of 4.");
173     }
174   } else ssSetErrorStatus(S,"Population size must be an integer.");
175 
176   if(IS_PARAM_DOUBLE(NGEN(S)) && (modf(mxGetPr(NGEN(S))[0],&temp)== 0)){
177     if(mxGetPr(NGEN(S))[0] <= 2){
178       ssSetErrorStatus(S,"Number of generations must be greater than 2.");
179     }
180   } else ssSetErrorStatus(S,"Number of generations must be an integer.");
181 
182   if(!IS_PARAM_DOUBLE(NOBJ(S)) || !(modf(mxGetPr(NOBJ(S))[0],&temp)== 0) || (int)mxGetPr(
      NOBJ(S))[0] < 0){
183     ssSetErrorStatus(S,"Number of objectives must be a non-negative integer.");
184   }
185   if(!IS_PARAM_DOUBLE(NCON(S)) || !(modf(mxGetPr(NCON(S))[0],&temp)== 0) || !(int)mxGetPr(
      NCON(S))[0] < 0){
186     ssSetErrorStatus(S,"Number of constraints must be a non-negative integer.");
187   }
188   if(!IS_PARAM_DOUBLE(NREAL(S)) || !(modf(mxGetPr(NREAL(S))[0],&temp)== 0) || !(int)
      mxGetPr(NREAL(S))[0] < 0){
189     ssSetErrorStatus(S,"Number of real variables must be a non-negative integer.");
190   }
191   if(!IS_PARAM_DOUBLE(NOUT(S)) || !(modf(mxGetPr(NOUT(S))[0],&temp)== 0) || (mxGetPr(NOUT(
      S))[0] <= 0.0)){
192     ssSetErrorStatus(S,"Number of outputs must be a positive non-zero integer.");
193   }
194 
195   if(mxGetPr(NREAL(S))[0] != 0){
196     flag_real = TRUE;
197     if (IS_PARAM_DOUBLE(MAX_MIN_REALVAR(S)) && OK_EMPTY_DOUBLE_PARAM(MAX_MIN_REALVAR(S))){
198       if(mxGetN(MAX_MIN_REALVAR(S)) != 2*(int)mxGetPr(NREAL(S))[0]){
199         ssSetErrorStatus(S,"Specified number of real variables does not match given limit
      vector.");
200       }
201       for(int i = 0; i < 2*(int)mxGetPr(NREAL(S))[0] - 1; i = i + 2){
202         if(mxGetPr(MAX_MIN_REALVAR(S))[i] >= mxGetPr(MAX_MIN_REALVAR(S))[i+1]){
203           ssSetErrorStatus(S,"Error, incorrect format. Correct format [min1 max1 min2 max2
      ...].");
204         }
205       }
206     } else ssSetErrorStatus(S, "Maximum and minimum values must be given as a non-empty
      double vector.");
207 
208     if (!IS_PARAM_DOUBLE(PCROSS_REAL(S)) || mxGetPr(PCROSS_REAL(S))[0]>=1.0 || mxGetPr(
      PCROSS_REAL(S))[0] <= 0.0 ){
209       ssSetErrorStatus(S,"Probability of crossover must be in range (0,1).");
210     }
211     if (!IS_PARAM_DOUBLE(PMUT_REAL(S)) || mxGetPr(PMUT_REAL(S))[0]>=1.0 || mxGetPr(
      PMUT_REAL(S))[0]<= 0.0 ){
212       ssSetErrorStatus(S,"Probability of mutation must be in range (0,1).");
213     }
214   } else (flag_real = FALSE);
215 
216   if(IS_PARAM_DOUBLE(HP(S)) && (modf(mxGetPr(NREAL(S))[0],&temp)== 0) && !(mxGetPr(HP(S))
      [0] <= 0.0)){
217     if(mxGetPr(HP(S))[0] < mxGetPr(NREAL(S))[0]) ssSetErrorStatus(S,"Prediction horizon
      must be greater than control horizon.");
```

```
218
219   } else ssSetErrorStatus(S,"Prediction horizon must be a positive integer.");
220
221   if (!IS_PARAM_DOUBLE(ETA_C(S)) || mxGetPr(ETA_C(S))[0]>=20.0 || mxGetPr(ETA_C(S))[0]<=
        5.0 ){
222     ssSetErrorStatus(S,"Distribution index for crossover should be between (5,20).");
223   }
224   if (!IS_PARAM_DOUBLE(ETA_M(S)) || mxGetPr(ETA_M(S))[0]>=50.0 || mxGetPr(ETA_M(S))[0]<=
        5.0 ){
225     ssSetErrorStatus(S,"Distribution index for mutation should be between (5,50).");
226   }
227
228   if(!IS_PARAM_DOUBLE(NBIN(S)) || !(modf(mxGetPr(NBIN(S))[0],&temp)== 0) || (int)mxGetPr(
        NBIN(S))[0] < 0){
229     ssSetErrorStatus(S, "Number of binary variables must be a non-negative integer.");
230   }
231   if(mxGetPr(NBIN(S))[0] != 0){
232     flag_bin = TRUE;
233     if(IS_PARAM_DOUBLE(NBITS(S)) || (modf(mxGetPr(NBITS(S))[0],&temp)== 0) || !(int)
        mxGetPr(NBITS(S))[0] < 0){
234       for(int i = 0; i < 2*(int)mxGetPr(NBIN(S))[0] - 1; i++){
235         if(mxGetPr(NBITS(S))[i] < 1){
236           ssSetErrorStatus(S,"Number of bits for binary variable must be greater than 1.")
        ;
237         }
238       }
239     }
240     if (IS_PARAM_DOUBLE(MAX_MIN_BINVAR(S)) && OK_EMPTY_DOUBLE_PARAM(MAX_MIN_BINVAR(S))){
241       if (mxGetN(MAX_MIN_BINVAR(S)) != 2 * (int)mxGetPr(NBIN(S))[0]){
242         ssSetErrorStatus(S, "Specified number of binary variables does not match given
        limit vector.");
243       }
244       for (int i = 0; i < 2 * (int)mxGetPr(NBIN(S))[0] - 1; i = i + 2){
245         if (mxGetPr(MAX_MIN_BINVAR(S))[i] >= mxGetPr(MAX_MIN_BINVAR(S))[i + 1]){
246           ssSetErrorStatus(S, "Error, incorrect format. Correct format [min1 max1 min2
        max2 ...].");
247         }
248       }
249     }
250     else ssSetErrorStatus(S, "Maximum and minimum values must be given as a non-empty
        double vector.");
251
252     if (!IS_PARAM_DOUBLE(PCROSS_BIN(S)) || mxGetPr(PCROSS_BIN(S))[0]>=1.0 || mxGetPr(
        PCROSS_BIN(S))[0] <= 0.0 ){
253       ssSetErrorStatus(S,"Probability of crossover (binary) must be in range (0,1).");
254     }
255     if (!IS_PARAM_DOUBLE(PMUT_BIN(S)) || mxGetPr(PMUT_BIN(S))[0]>=1.0 || mxGetPr(PMUT_BIN(
        S))[0]<= 0.0 ){
256       ssSetErrorStatus(S,"Probability of mutation (binary) must be in range (0,1).");
257     }
258   } else (flag_bin = FALSE);
259
260   if((flag_bin == FALSE) && (flag_real == FALSE)){
261     ssSetErrorStatus(S, "Real and binary variables entered 0, exit simulation.")
262   }
263 }
264 #endif
265
266 static void mdlInitializeSizes(SimStruct *S)
267 {
268     ssSetNumSFcnParams(S, 20);  /* Number of expected parameters */
269     #if defined(MATLAB_MEX_FILE)
270         if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S)) {
```

```
            mdlCheckParameters(S);
            if (ssGetErrorStatus(S) != NULL){
                return;
            }
        }else {
            return; /* Parameter mismatch will be reported by Simulink */
        }
    #endif
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, true); /*direct input signal access*/
    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is used in either
     * the mdlOutputs or mdlGetTimeOfNextVarHit functions.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 0);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    seed    = mxGetPr(SEED(S))[0];
    popsize = mxGetPr(POPSIZE(S))[0];
    ngen    = mxGetPr(NGEN(S))[0];
    nobj    = (int) mxGetPr(NOBJ(S))[0];
    ncon    = (int) mxGetPr(NCON(S))[0];
    nreal   = (int) mxGetPr(NREAL(S))[0];
    hp      = (int) mxGetPr(HP(S))[0];
    nout    = (int) mxGetPr(NOUT(S))[0];
    if(flag_real){
      min_realvar = (double *)malloc(nreal*sizeof(double));
      max_realvar = (double *)malloc(nreal*sizeof(double));
      for(int i = 0; i < nreal; i++){
        min_realvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i];
        max_realvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i+1];
      }
      pcross_real = mxGetPr(PCROSS_REAL(S))[0];
      pmut_real   = mxGetPr(PMUT_REAL(S))[0];
    }
    eta_c       = mxGetPr(ETA_C(S))[0];
    eta_m       = mxGetPr(ETA_M(S))[0];
    nbin   = (int) mxGetPr(NBIN(S))[0];
    nbits  = (int *)malloc(nbin*sizeof(int));
    if(flag_bin){
    min_binvar = (double *)malloc(nbin*sizeof(double));
    max_binvar = (double *)malloc(nbin*sizeof(double));
    for(int i = 0; i < nbin; i++){
      nbits[i]      = mxGetPr(NBITS(S))[i];
      min_binvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i];
      max_binvar[i] = mxGetPr(MAX_MIN_REALVAR(S))[2*i+1];
    }
    pcross_bin  = mxGetPr(PCROSS_BIN(S))[0];
    pmut_bin    = mxGetPr(PMUT_BIN(S))[0];
```

```
    bitlength = 0;
      if ( nbin!= 0){
        for (int i = 0; i < nbin; i++)
        {
          bitlength += nbits[i];
        }
      }
    }
    nbinmut    =  0;
    nrealmut   =  0;
    nrealcross =  0;
    parent_pop = (population *)malloc(sizeof(population));
    child_pop  = (population *)malloc(sizeof(population));
    mixed_pop  = (population *)malloc(sizeof(population));

    idx        = (int *)malloc(nout*sizeof(int)); /*TODO malloc size depending on MIMO */
    idx[0]     = 0;

    assign_ref(mxGetPr(REF_PARAM(S)),R_N);
    /* Specify the operating point save/restore compliance to be same as a
     * built-in block */
    ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);

    ssSetRuntimeThreadSafetyCompliance(S, RUNTIME_THREAD_SAFETY_COMPLIANCE_TRUE);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}



/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, mxGetPr(SAMPLE_T(S))[0]);
    ssSetOffsetTime(S, 0, 0.0);
}



#define MDL_INITIALIZE_CONDITIONS   /* Change to #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  /* Function: mdlInitializeConditions ========================================
   * Abstract:
   *    In this function, you should initialize the continuous and discrete
   *    states for your S-function block.  The initial states are placed
   *    in the state vector, ssGetContStates(S) or ssGetRealDiscStates(S).
   *    You can also perform any other initialization activities that your
   *    S-function may require. Note, this routine will be called at the
   *    start of simulation and if it is present in an enabled subsystem
   *    configured to reset states, it will be call when the enabled subsystem
   *    restarts execution to reset the states.
   */
  static void mdlInitializeConditions(SimStruct *S)
  {
  }
#endif /* MDL_INITIALIZE_CONDITIONS */



```

```
#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart =======================================================
   * Abstract:
   *    This function is called once at start of model execution. If you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
    init_sys(); // initialize k,yp,refx of model
    allocate_memory_pop(parent_pop, popsize);
    allocate_memory_pop(child_pop, popsize);
    allocate_memory_pop(mixed_pop, 2 * popsize);
    randomize();
    (&(parent_pop->ind[idx[0]]))->xreal[0] = 0.0;
  }
#endif /*  MDL_START */



/* Function: mdlOutputs =======================================================
 * Abstract:
 *    In this function, you compute the outputs of your S-function
 *    block.
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
  real_T *u = ssGetOutputPortSignal(S, 0);
  u[0] = (&(parent_pop->ind[idx[0]]))->xreal[0];
}



#define MDL_UPDATE  /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate ======================================================
   * Abstract:
   *    This function is called once for every major integration time step.
   *    Discrete states are typically updated here, but this function is useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    double * tmp = (double *)ssGetInputPortSignal(S, 0);
    for (int i = 0; i < ord_sys - 1; i++){
        yp[i] = yp[i+1];
    }
    up = (&(parent_pop->ind[idx[0]]))->xreal;

    yp[ord_sys - 1] = tmp[0];
    epred[0] = tmp[0] - ytmp[ord_sys];


    initialize_pop(parent_pop);
    decode_pop(parent_pop);
    evaluate_pop(parent_pop); // nobj and ncon must match define test_problem()-
    assign_rank_and_crowding_distance(parent_pop);
    for (int i = 2; i <= ngen; i++)
    {
        selection(parent_pop, child_pop);
        mutation_pop(child_pop);
```

```
460         decode_pop(child_pop);
461         evaluate_pop(child_pop); // apply fitness function
462         merge(parent_pop, child_pop, mixed_pop);
463         fill_nondominated_sort(mixed_pop, parent_pop);// generate new parent_pop
464     }
465     decision_maker(parent_pop,idx);
466     ytmp = sys((&(parent_pop->ind[idx[0]]))->xreal,yp,up,nreal,hp);
467 
468     update_k();
469 }
470 #endif /* MDL_UPDATE */
471 
472 
473 
474 #define MDL_DERIVATIVES  /* Change to #undef to remove function */
475 #if defined(MDL_DERIVATIVES)
476   /* Function: mdlDerivatives =================================================
477    * Abstract:
478    *    In this function, you compute the S-function block's derivatives.
479    *    The derivatives are placed in the derivative vector, ssGetdX(S).
480    */
481   static void mdlDerivatives(SimStruct *S)
482   {
483   }
484 #endif /* MDL_DERIVATIVES */
485 
486 
487 
488 /* Function: mdlTerminate =====================================================
489  * Abstract:
490  *    In this function, you should perform any actions that are necessary
491  *    at the termination of a simulation.  For example, if memory was
492  *    allocated in mdlStart, this is the place to free it.
493  */
494 static void mdlTerminate(SimStruct *S)
495 {
496           if (nreal!=0)
497     {
498         free (min_realvar);
499         free (max_realvar);
500     }
501     if (nbin!=0)
502     {
503         free (min_binvar);
504         free (max_binvar);
505         free (nbits);
506     }
507     deallocate_memory_pop (parent_pop, popsize);
508     deallocate_memory_pop (child_pop, popsize);
509     deallocate_memory_pop (mixed_pop, 2*popsize);
510     free (parent_pop);
511     free (child_pop);
512     free (mixed_pop);
513     terminate_sys();
514     UNUSED_ARG(S);
515 }
516 
517 
518 /*=============================*
519  * Required S-function trailer *
520  *=============================*/
521 
522 #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
```

```
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Código 10.8: sfun_imo_nmpc.c

```

mex -outdir mex\ -IE:\c_ws\sfun_imo_nmpc\modelndm-v1 -IE:\c_ws\sfun_imo_nmpc\modelndm-v1\
    NN ...
     -IE:\c_ws\sfun_imo_nmpc\nsga2-v1.1.6 ... %modify include paths to match ones
    sfun_imo_nmpc.c nsga2-v1.1.6\allocate.c nsga2-v1.1.6\auxiliary.c ...
    nsga2-v1.1.6\crossover.c nsga2-v1.1.6\crowddist.c ...
    nsga2-v1.1.6\decode.c nsga2-v1.1.6\dominance.c nsga2-v1.1.6\eval.c ...
    nsga2-v1.1.6\fillnds.c nsga2-v1.1.6\initialize.c nsga2-v1.1.6\list.c ...
    nsga2-v1.1.6\merge.c nsga2-v1.1.6\mutation.c ...
    nsga2-v1.1.6\problemdef.c nsga2-v1.1.6\rand.c nsga2-v1.1.6\rank.c ...
    nsga2-v1.1.6\sort.c nsga2-v1.1.6\tourselect.c ... % upto here original files
    modelndm-v1\decision.c modelndm-v1\model.c  ... % added files
    modelndm-v1\NN\SNL2_NN.c
%% NSGA-II parameter definition
seed =      0.254;
popsize =   200;
ngen =      200;
nobj =      2;
ncon =      0;
% real variables/chromosomes
hc    =     4;
hp    =    6;
nout =      1;
nreal =     hc*nout; % hc
max_min_realvar = repmat([-4 4],[1 nreal]);
pcross_real =   0.65;
pmut_real =     0.5;
eta_c =         15;
eta_m =         25;
% binary variables/chromosomes
nbin =          0;
nbits =         0;
max_min_binvar = 0;
pcross_bin =    0;
pmut_bin =      0;

Tm = 0.1; % sfunction sampling time as well as simulation step time.
tf = 10;  % final time

%% Enter your desired reference here
ref = zeros(1,tf/Tm+hp);

% x = linspace(0,10,tf/Tm);
% ref = 3*sin(x);

ref(1,1) = 0;
ref(1,2:25) = -3;
ref(1,26:50) = 2;
ref(1,51:75) = -1;
ref(1,76:end) = 0.5;

% b = 0.5;
% for i = 1:2:length(ref)+2
%     ref(1,i:(i-1)+2) = i*b;
%     b = b/1.2;
% end
```

Código 10.9: compile.m