UNIVERSIDAD DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

Aceleración de algoritmos intervalares de ajuste automático del lazo en QFT

D. Isaac Martínez Forte

2022

Departamento de Informática y Sistemas

Facultad de Informática

Universidad de Murcia

Aceleración de algoritmos intervalares de ajuste automático del lazo en QFT

Tesis doctoral

Isaac Martínez Forte
Ingeniero Informático

Director: Joaquín Cervera López

Murcia, marzo 2022

Aceleración de algoritmos intervalares de ajuste automático del lazo en QFT

Tesis doctoral
Universidad de Murcia

Murcia, marzo 2022

La composición del texto ha sido realizada usando LaTeX

Isaac Martínez Forte
Ingeniero Informático

Director: Joaquín Cervera López
Doctor en Informática

Facultad de Informática
Departamento de Informática y Sistemas

Universidad de Murcia
Campus de Espinardo
30100 Murcia

Teléfono: (+34) 968398334
Fax: (+34) 968364151
Correo electrónico: isaac.martinez@um.es


QUIERO HACER LA TESIS
ES COMPLICADO...
POR FAAAAVOOOOOOOOOOOOR
VALE
¿POR QUÉ ESTO NO ESTÁ EXPLICADO?
¿!
PORQUE ES MUY OBVIO Y NO HACE FALTA EXPLICARLO
AH...
DESPUÉS DE MUCHO TRABAJO REALIZADO...
... ISAAC EMPEZÓ A PERDER LA ESPERANZA
NO ME SALE
NO SIRVO PARA ESTO
HAY DEMASIADOS ERRORES
ES TODO UN DESASTRE
NUNCA VOY A SACARLO ADELANTE
SIN EMBARGO, AL AÑO SIGUIENTE, RECARGADO DE ENERGÍAS...
DOCTORADO 2.0.
¡VOY A INTENTARLO!
TRAS MESES Y MESES DE TRABAJO, ERRORES, ACIERTOS, NUEVAS IDEAS..
ARTÍCULO
¡HEMOS ACEPTADO TU ARTÍCULO EN LA REVISTA!!
yuhuuu
Y, POR FIN, UN ÚLTIMO EMPUJÓN...
Tesis Doctoral
ACELERACIÓN DE ALGORITMOS INTERVALARES DE AJUSTE AUTOMÁTICO DEL LAZO EN QFT
Autor: Isaac Martínez Forte
Tutor: Joaquín Cervera López

Dedicada a mis padres,
José Rafael y Josefa,
y a mi hermana,
Irene

Resumen

En este trabajo se aborda la aceleración de algoritmos del tipo búsqueda global intervalar que tienen por objetivo la resolución, de forma automática, del problema del ajuste del lazo en QFT. Esta técnica de diseño de controladores robustos en el dominio de la frecuencia se plantea como una sucesión de etapas. En cada una de ellas se genera la información necesaria para la siguiente, culminando todo este proceso en la fase más importante, el ajuste del lazo, donde se genera el controlador.

Disponer de algoritmos que puedan resolver de forma automática, sin intervención del usuario, el ajuste del lazo en QFT no es una tarea sencilla. Dado que constituye un problema de optimización no lineal y no convexo, las soluciones algorítmicas que se le pueden aplicar tienen dificultades inherentes de difícil solución.

A lo largo de la historia se han planteado diferentes propuestas para dar respuesta a esta necesidad, pero hasta ahora no se ha conseguido ninguna solución satisfactoria. Dada la complejidad del proceso, se ha afrontado el problema desde diversas ópticas. Una aproximación interesante son los algoritmos de búsqueda global intervalar. Este tipo de propuestas buscan asegurar el óptimo en la solución utilizando aritmética intervalar, pero tienen la problemática de que, al ser una búsqueda global, tienen un coste computacional muy alto, y con ello, lleva a una curva exponencial del tiempo de ejecución conforme el tamaño del problema aumenta.

Este trabajo se ha centrado en analizar con detenimiento los algoritmos de este tipo, focalizando el esfuerzo en comprender donde existen más posibilidades de mejora. Una vez concluida esta tarea, se ha trabajado en el planteamiento de alternativas que mejoren las propuestas existentes, culminando en una serie de novedosas estrategias que tienen por objetivo reducir el coste computacional y el tiempo de ejecución.

Un primer grupo de estrategias tienen la finalidad de acotar el espacio de búsqueda del algoritmo, y su objetivo es detectar tanto el subrango viable como inviable de cada una de las variables del controlador, utilizando la fase y la magnitud en el plano de Nichols. De esta forma, por un lado se con-

sigue eliminar los subrangos identificados como inviables, y por otro lado, encontrar soluciones en el subrango viable. Dentro de este mismo grupo, se plantea una estrategia que busca encontrar soluciones al problema de forma rápida, su funcionamiento es similar a las acotaciones descritas, pero tiene la capacidad de encontrar mejores soluciones, que podrán ser utilizadas para realizar podas en nodos poco prometedores.

Un segundo grupo tienen un carácter más transversal. La primera de ellas contiene varias opciones para realizar la bisección del nodo. Como opción principal se propone una bisección avanzada que utiliza la información generada del nodo actual para tomar la mejor decisión posible. Complementado a ésta, se plantean tres opciones más básicas, realizar la bisección por la variable que más influye en el área, por la que más influye en la magnitud o por la que más influye en la fase, todas ellas en el plano de Nichols, según convenga en cada momento. A continuación, se plantea una segunda estrategia que está centrada en adaptar el funcionamiento del algoritmo a la situación en la que se encuentre a lo largo de su ejecución, es decir, modifica el comportamiento del resto de estrategias y las adapta a las circunstancias concretas del algoritmo en ese momento consiguiendo, de esta forma, mejorar las prestaciones de todas las estrategias desarrolladas.

Para terminar, esta tesis propone un nuevo algoritmo de ajuste automático del lazo en QFT del tipo búsqueda global intervalar que engloba todas estas estrategias propuestas. Dicho algoritmo logra reducir el coste computacional de manera importante con respecto a las soluciones previas y, de esta forma, consigue una mejora en el tiempo de ejecución de dos órdenes de magnitud para controladores típicos.

Abstract

This paper deals with the acceleration of interval analysis global search algorithms whose goal is to automatically solve the loop shaping problem in QFT. This design technique for robust controllers in the frequency domain is proposed as a succession of stages. Each step generates information for the following stage. The last stage, the most important one, is the loop shaping, where the controller is generated.

Having algorithms which can automatically (without user intervention) solve the QFT loop shaping problem isn't an easy task. Given that it's a non-linear and non-convex optimization problem, the algorithmic solutions that can be applied pose several difficulties.

Throughout time, different proposals have been made to solve the problem but, until now, no satisfactory solutions has been achieved. Given the complexity of the process, the problem has been faced from various perspectives. An interesting approach is interval global search algorithms. This type of proposals try to ensure an optimum solution using interval arithmetic, but they have the problem that, being a global search, they have a very high computational cost, and because of that, an exponential curve of the execution time grows exponentially with the size of the problem.

This work has focused on carefully analyzing this kind of algorithms and the effort has been centered on understanding where more possibilities for improvement were. Once this task has been completed, the work has been focused on the proposal of alternatives that improve the existing algorithms and it culminates in a series of novel strategies that aim to reduce the computational cost and the execution time.

A first group of strategies has the purpose of delimiting the algorithm's search space and its objective is to detect both the feasible and unfeasible subrange of each controller's variable, using the phase and magnitude in the Nichols plane. In this way, on the one hand, it's possible to eliminate the subranges identified as unfeasible, and on the other hand, to find solutions in the viable subrange. Within this same group, it's proposed a strategy which seeks to quickly find solutions to the problem and whose performance is si-

milar to the bounding described although it has the ability to find better solutions, which can be used to prune unpromising nodes.

A second group has a more transversal disposition. The first one contains several options to perform the bisection of the node. As a main option, the proposal is an advanced bisection which uses the information generated from the current node to make the best decision. Three more basic options are proposed to complement the previous one, to carry out the bisection by the most influential variable on the area, by the most influential one on the magnitude or by the most influential one on the phase, all of them in the Nichols plane, as appropiate depending on each moment's conditions. Following, it's proposed a second strategy that aims to adapt the algorithm behaviour to every situation along the execution, in other words, it modifies the behavior of the rest of strategies and adapts them to each specific circumstance of the algorithm at any time. Thereby it improves the performance of every developed strategies.

Finally, this thesis proposes a new interval global search QFT Automatic Loop Shaping algorithm which involves all these strategies. This algorithm achieves a significant reduction of the computational cost with respect to previous solutions and, in this way, it achieves a two orders of magnitude improvement in execution time for typical controllers.

Agradecimientos

Después de tantos años, tantas alegrías y disgustos, tantas risas y llantos, estoy escribiendo la última página de esta tesis. No tengo claro cómo me siento, creo que emocionado, pero también agotado física y mentalmente. Ha sido un largo camino lleno de muchos baches, algunos de ellos muy profundos en los que siempre hubo una persona que tiró de mí cuando yo no pude. Más allá de ser mi tutor, y de todo el esfuerzo académico que ha realizado, lo realmente importante ha sido el apoyo personal que me ha dado. Sin él, esta tesis no habría sido posible, gracias Quino (Joaquín Cervera), has sido mucho más que un tutor.

¿Cuantos años llevo estudiando?, realmente toda mi vida, y durante todo este aprendizaje he tenido muchos fallos y errores, pero siempre tuve a mi familia ahí para apoyarme. Se esforzaron mucho por mí cuando en cada trimestre del instituto no hacía más que recoger suspensos. Me apoyaron cuando mi primer año de carrera fue desastroso y estuve a punto abandonar. Fini, Joserafa e Irene, gracias por tanto, llegar hasta aquí ha sido gracias a vosotros.

La muerte es parte de la vida, y cuando llegue la hora, lo único que quedará de nosotros será el recuerdo en nuestros seres queridos. Ella aprendió a escribir con una rama en la tierra, sin medios, sin nada, sólo con la determinación de anhelar saber leer y escribir, de no querer el futuro que la sociedad patriarcal tenía preparado para ella. Has sido un ejemplo de vida, gracias abuela.

Sólo me has conocido estresado, agobiado y en una demoledora carrera sin fin por terminar esta tesis, con todo lo que ello supone. Aún así, has estado a mi lado, me has apoyado en todos los momentos, me has sostenido cuando yo ya no era capaz, hemos llorado y reído, has hecho esta tesis tuya y la has vivido junto a mí, con todo lo que eso conlleva. Has sido el viento que me ha empujado a seguir hacia delante en mis días más amargos, has sido el faro que me ha guiado para no caer en la desesperación, has sido todo lo que he necesitado para sobrevivir a estos últimos meses, gracias Carmen.

A veces, hay espacios en los que te sientes un poco solo, que no cono-

ces a otras personas, que no tienes con quien compartir tus preocupaciones. María, Mario, gracias por ser mis compañeros de doctorado, por compartir las conversaciones que no podía compartir con casi nadie, por montar una delegación. Pero sobretodo, gracias por las noches de risa, vinos y juegos de mesa.

Luego están los que siempre han estado, los de toda la vida, llevamos tantos años juntos que ya ni me acuerdo. Sé que he estado ausente, que los kilómetros de distancia se han hecho largos, que he ido al pueblo menos de lo que debería, gracias por ser inquebrantables.

Quizás esta frase no suene tan bien como las que dicen en las películas, pero *el yoga ha salvado mi espalda*, puede parecer poco, pero tantas y tantas horas sentado en una silla no han sido fáciles. Gracias María, sin ti, seguramente ahora mismo ya no tendría cuello ni espalda.

En fin, esto se termina. Ha llegado el momento de escribir la última línea. Tantos años después, esta historia está apunto de acabar, y con final feliz, que no estaba tan claro. Esto sólo ha sido un pequeño resumen de todas las personas que han pasado por mi vida, pero hay muchas más, gracias a todas ellas por haber estado, por haberse interesado y por sus ánimos. Se cierra un capítulo de mi vida muy importante, no sé lo que me deparará el futuro, no sé si podré cumplir mis sueños, pero estoy contento, esta tesis está terminada, hoy voy a llorar por ella una última vez, pero por fin, será de emoción y felicidad.

Isaac Martínez Forte

Notaciones

- *LTI:* Sistemas lineal e invariante en el tiempo, como su propio nombre indica, cumple las propiedades de linealidad e invarianza en el tiempo, como por ejemplo, el principio de superposición.
- k_{hf}: ganancia de alta frecuencia de un lazo $L(s)$,

$$k_{hf} = \lim_{s \to \infty} s^{n_{pe}} L(s) \tag{1}$$

  donde n_{pe} es el exceso de polos sobre ceros.
- *Diagrama de Nichols* o *plano de Nichols*: forma de representación del plano complejo en forma polar, donde la fase en grados se representa en el eje de abscisas y la magnitud en decibelios se representa en el eje de ordenadas.
- *Fronteras multivaluadas*: Son fronteras que representadas en el plano de Nichols donde para una fase concreta tiene más de una solución.
- *Círculo-M*: Representa una frontera en el plano de Nichols donde los puntos p cumplen la siguiente ecuación

$$\left| \frac{p}{1+p} \right| = \lambda$$

  para un cierto λ. Se suelen utilizar típicamente para definir fronteras de estabilidad robusta.
- *ILP:* Programación lineal entera (*Integer Linear Programming*). Son problemas de optimización lineal donde todas o parte de las variables pertenecen a los números enteros.
- *CACSD:* Siglas de diseño de sistemas de control asistido por computador (*computer-aided control systems design*). Son aplicaciones que ayudan al experto a diseñar sistemas de control.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Un sistema puede definirse como un conjunto de partes o elementos organizados que se coordina para lograr un objetivo. Este sistema puede tener variables de entrada que condicionan la actuación del sistema, y también es posible, que generen una respuesta de salida en forma de señales (1.1).

Los sistemas pueden clasificarse de diferentes formas, algunas de las más comunes se mencionan a continuación.

La primera clasificación diferencia entre sistemas dinámicos y sistemas estáticos, los dinámicos evolucionan con el tiempo y, por tanto, tienen un estado que define su comportamiento ante la entrada en un momento dado. En cambio, los sistemas estáticos no varían su comportamiento con el tiempo.

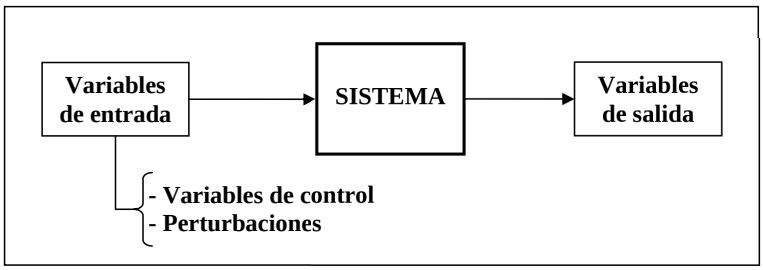

Figura 1.1: Sistema representado esquemáticamente.

La segunda de las clasificaciones es según el número de señales de salida y de entrada que tengan, pueden ser sistemas de una sola señal de entrada y una de salida, conocidos como *SISO*, o múltiples señales de entrada y de salida, conocidos como *MIMO*.

Dentro de los sistemas dinámicos, existen los sistemas *Lineales e invariantes en el tiempo* (LTI), que como su propio nombre indica, cumplen las

propiedades de linealidad e invarianza en el tiempo. Un sistema se define como lineal (L) si cumple el principio de superposición, que contiene las propiedades de proporcionalidad y aditividad. La propiedad de proporcionalidad significa que si la entrada de un sistema es multiplicada por un factor, la salida del sistema también se verá afectada en la misma proporción por dicho factor. Por otro lado, que un sistema sea aditivo conlleva que si la entrada es el resultado de la suma de dos entradas, la salida resultante será la suma de las salidas que producirían cada una de esas entradas individualmente. Este tipo de sistemas se pueden representar con una función de transferencia, la cual se define para un sistema LTI, como la *transformada de Laplace* de la respuesta a un impulso. Es decir, si el sistema tiene como entrada una *función delta de Dirac* o equivalente, la función de transferencia representa el cociente de la *transformada de Laplace* de la salida y de la entrada, dando por supuesto que las condiciones iniciales son nulas.

En ingeniería de control, para que un sistema dinámico tenga el comportamiento deseado, se le acopla otro sistema, denominado de control, que modificará su funcionamiento para adecuarlo a las restricciones que se le quieran imponer. Al sistema original se le denominará planta, y al sistema de control, que puede estar conformado por uno o más sistemas dinámicos, se les llamará controladores. Las dos partes estarán representadas por funciones de transferencia.

Existen diferentes esquemas de control para sistemas dinámicos, siendo el más básico el *lazo abierto* (1.2). Otro más complejo, y utilizado en esta tesis, es el *lazo cerrado* (1.3).

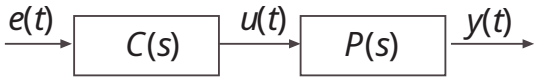

Figura 1.2: Esquema de sistema de control abierto.

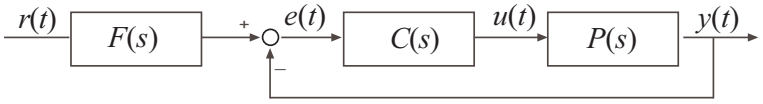

Figura 1.3: Esquema de sistema de control cerrado.

En el contexto de un sistema de control, la señal de entrada $e(t)$ y la señal de salida $y(t)$ se interpretan como una función en el tiempo de una variable del dominio de los reales, $u : t \longrightarrow \mathbb{R}$ que representa una magnitud física continua y su evolución en el tiempo.

La característica principal de los sistemas de *lazo abierto* es que la señal de entrada $e(t)$ no tiene relación con la señal de salida $y(t)$, por tanto, la

señal de entrada no recibe ningún tipo de corrección propiciada por la señal de salida. Eso obliga a que se conozca perfectamente, y sin incertidumbre, el sistema que se quiere controlar.

La función de transferencia de un *lazo abierto* es:

$$L(s) = C(s)P(s) \tag{1.1}$$

donde $C(s)$ es el controlador, $P(s)$ es el sistema a controlar y $L(s)$ es el sistema de control completo resultante.

Por otro lado, en un sistema de control de *lazo cerrado* que sigue un esquema con realimentación, la señal de entrada $e(t)$ depende de la señal de salida $y(t)$, por tanto, la señal de entrada se puede modificar con información de la señal de salida. De esta forma, se permite corregir la incertidumbre de la planta, o los posibles errores que puedan producirse en el tiempo y no estén controlados previamente.

La función de transferencia de un *lazo cerrado* es:

$$T(s) = \frac{L(s)}{1 + L(s)} F(s) \tag{1.2}$$

donde $F(s)$ es el prefiltro/precompensador en el sistema de control, $T(s)$ es el sistema controlador y $L(s)$ es el sistema de control.

Para que un sistema de control sea considerado robusto, debe cumplir con las especificaciones impuestas en el diseño a pesar de la incertidumbre y las perturbaciones sobre la planta.

En las siguientes secciones, se realiza un repaso a los principales conceptos utilizados en esta tesis, en primer lugar, se encuentra una introducción a QFT (*Quantitative feedback theory*) donde se explica su funcionamiento, su buen desempeño para obtener controladores robustos y se completa con un ejemplo donde se muestran todas las etapas de su desarrollo.

A continuación, se realiza una introducción a la aritmética intervalar, herramienta fundamental usada en los algoritmos propuestos que permite, entre otras cosas, usar rangos para representar la incertidumbre de la planta y la variabilidad del controlador sin perder precisión.

Para terminar, se introduce el método algorítmico de *Ramificación y poda* (*Branch and Bound*),

1.1. Introducción a QFT

QFT es una técnica de control robusto que trabaja en el dominio de la frecuencia, se centra específicamente en cuantificar de forma precisa la relación que existe entre la incertidumbre y el esfuerzo de control necesario para cumplir las especificaciones impuestas. Las restricciones se deben cumplir a pesar de la incertidumbre y las perturbaciones de la planta, por tanto, se focaliza en encontrar el controlador óptimo que permita cumplir con las especificaciones de diseño con el menor esfuerzo de control posible.

En el libro *Network Analysis and Feedback Amplifier Design* (Bode [1945]) del año 1945, H.W. Bode introdujo los conceptos básicos del análisis de sistemas, de la realimentación (*lazo cerrado* (1.2)) y del diseño de compensadores en el dominio de la frecuencia.

Posteriormente, en 1959, Isaac M. Horowitz publicó un artículo titulado *Fundamental theory of automatic linear feedback control systems* (Horowitz [1959]) en el cual se introdujo el concepto de realimentación cuantitativa, complementando las ideas de *Bode*. De esta forma, añadió la idea fundamental de que, cuando se implementa un sistema de control realimentado, la cuantificación es importante con respecto a las especificaciones requeridas y la incertidumbre de la planta. Con ello, inició la teoría del control cuantitativo. En 1963, ampliaría estos conceptos en el libro *Synthesis of Feedback Systems* (Horowitz [1963]), donde se aportaron interesantes ideas como el control basado en dos grados de libertad, entre otras.

Ya en 1972 Horowitz y Sidi publicaron en libro *Synthesis of feedback systems with large plant ignorance for prescribed time-domain tolerances* (Horowitz y Sidi [1972]) donde se acuña el término *Teoría de la realimentación cuantitativa* (QFT) como método para para paliar los efectos de la incertidumbre en la planta. La incertidumbre forma parte de los sistemas de control robusto, puede estar presente debido a la incapacidad de modelar mejor un sistema o por variaciones no previstas de los parámetros que pueden producirse en un proceso real. De esta forma, en el que se incluye a priori un modelo de incertidumbre al proceso de diseño, se garantiza que el sistema de control que se obtenga minimice sus efectos y satisfaga las especificaciones de diseño impuestas a la planta, siempre teniendo en cuenta que existen restricciones en el esfuerzo de control.

Por tanto, es fundamental modelar la incertidumbre de forma correcta para conseguir representarla adecuadamente en el modelo del sistema, y así diseñar un controlador que reduzca sus efectos en lo máximo posible.

A continuación, se definen dos de las formas más comunes para representar la incertidumbre en el caso de los sistemas LTI dados por una función de

transferencia. Para una incertidumbre aditiva:

$$P_a = \{P_0(s) + D(s), D\,\epsilon\,\mathbf{D}\} \tag{1.3}$$

y para la incertidumbre multiplicativa:

$$P_m = \{P_0(s)\,Q(s), Q\,\epsilon\,\mathbf{Q}\} \tag{1.4}$$

donde los conjuntos D y Q representan la incertidumbre.

Si se está utilizando el dominio de la frecuencia, esta es la forma natural para representar las plantas modeladas, donde se tienen unas frecuencias de diseño concretas y para cada una de ellas se representa la respuesta de P_a y P_m mediante una plantilla. Se puede observar en la figura (1.4).

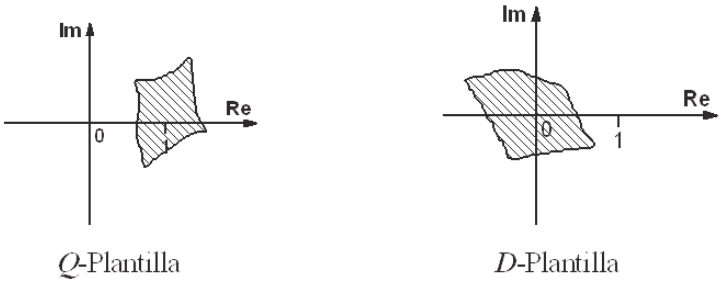

Figura 1.4: Representación en forma de plantilla de la incertidumbre aditiva y multiplicativa de una planta incierta.

Las diferentes formas que pueden tener las plantillas representadas en la figura (1.4) dependerán del tipo de incertidumbre que tenga la planta, esta puede ser de diferentes formas. Por ejemplo, si es incertidumbre no estructurada estas tendrán forma de círculos, rectángulos, etc, siguiendo una norma. En cambio, cuando estás plantillas se obtengan a partir de las variaciones de parámetros de una función de transferencia (coeficientes de polinomios o polos y ceros), la incertidumbre será paramétrica.

Por otro lado, existen formas más estructuradas de incertidumbre que las paramétricas, por ejemplo, cuando una planta es el resultado de la interconexión de subsistemas que a su vez también tienen incertidumbre, esta incertidumbre podrá ser paramétrica o no.

En las técnicas de diseño de sistemas de control robusto en el dominio de la frecuencia, de forma común, se suelen encontrar las siguiente etapas (Bailey et al. [1994]):

1. Un sistema para representar la planta y su incertidumbre asociada.

2. Implementación de unas restricciones de diseño.

3. Ajuste de una función racional que respete las restricciones de diseño impuestas.

Todo el diseño de las diferentes etapas en QFT se basa en una serie de datos que introduce el usuario y que se enumeran a continuación:

- Planta e incertidumbre: El primer dato que requiere QFT es definir cual es la planta que se quiere controlar, que además, llevará asociada una incertidumbre. De la planta se elegirá un nominal, que se utilizará en los cálculos posteriores, dicha elección indicará cual es la planta principal y la diferenciará de la incertidumbre.

- Frecuencias de diseño: Todas las etapas de QFT realizan los cálculos basados en unas frecuencias de diseño concretas, dado que, por coste computacional, las operaciones no se pueden realizar sobre un conjunto de frecuencias demasiado amplio. Por tanto, éstas deben de ser elegidas estratégicamente para que interaccionen adecuadamente con las restricciones de diseño.

- Restricciones de diseño: Describen cuál es el comportamiento mínimo admisible que debe respetar el sistema controlado.

Las etapas que se definen en (Bailey et al. [1994]) tienen su equivalencia en las etapas de las que consta el proceso de QFT. Todos estos pasos se realizan tradicionalmente en el plano de Nichols, donde se representa la ganancia y la fase del lazo abierto.

La etapa número uno en QFT, que tiene su equivalencia con la etapa (1), es el cálculo de plantillas o *templates*, estas representan la planta y la incertidumbre que tengan asociada. Esta etapa es importante, dado que, si se modela adecuadamente la incertidumbre, el controlador resultante, abarcará un mayor número de variaciones de la planta nominal.

A continuación, en la etapa 2, y una vez que se han obtenido las plantillas para el modelo de la planta $P(j\omega)$, junto con las especificaciones de diseño impuestas, tanto frecuenciales como temporales, se combinan con el fin de obtener un conjunto de curvas $B(j\omega_i)$ (una por cada frecuencia de diseño (ω_i)) que son regiones aceptables para la ganancia y la fase nominales de la función de transferencia en lazo abierto. Las fronteras de estas regiones, conocidas en la literatura sobre el tema como *boundaries*, son las restricciones de diseño, que es equivalente a la etapa (2).

La siguiente etapa, la número 3, que tiene su equivalencia con (3), es la más importante del diseño en QFT. Es es el *ajuste del lazo* o *loop shaping*, consiste en diseñar una función de transferencia nominal de lazo abierto $L_0(j\omega) = C(j\omega)P_0(j\omega)$ que cumpla con dichas restricciones de diseño para

cualquier posible planta dada por la incertidumbre. Existen en la literatura diferentes aproximaciones para abordar este apartado, desde técnicas basadas en prueba y error, que con ayuda de un sistema informático, pero de forma manual, consiguen llegar a una solución, hasta técnicas basadas en optimización. Hasta ahora, las técnicas que resuelven esta etapa de forma automática y sin intervención del usuario presentan diferentes problemas que todavía están sin resolver. Posteriormente, en la sección (2) se tratarán detalladamente.

El proceso de QFT se complementa con otras etapas de introducción de datos necesarias para poder realizar las etapas principales ya descritas. A continuación, se muestran estas etapas secundarias:

- Modelado de la planta y su incertidumbre.
- Elección de un conjunto de frecuencias de diseño (ω).
- Definición de las restricciones de diseño o especificaciones.
- Síntesis del pre-filtro $F(j\omega)$ (opcional).
- Validación del diseño.

A continuación, se ejemplifican las etapas más importantes de QFT a partir de un caso práctico. Para ello, se utiliza el ejemplo número dos del *Toolbox de QFT de Matlab* (Borghesani et al. [2003]).

1.1.1. Modelado de la planta y su incertidumbre

En primer lugar se debe modelar la planta junto con la incertidumbre que tenga asociada. En el ejemplo que se propone, la planta sería la siguiente:

$$P(s) = k\frac{1}{(s+a)(s+b)} \tag{1.5}$$

y la incertidumbre asociada sería la siguiente:

$k \in [1, 10]$

$a \in [1, 5]$

$b \in [20, 30]$

En QFT se permite describir el modelo dinámico de la planta a controlar desde diferentes perspectivas. Por ejemplo, desde datos experimentales recogidos directamente de una planta real, mediante datos de respuesta frecuencial, funciones de transferencia lineales, invariantes (LTI) o variantes (LTV) en el tiempo, desde ecuaciones en el espacio de estados o mediante

ecuaciones no lineales. Todas estas formas pueden tener incertidumbre no paramétrica o paramétrica, la cual, a su vez puede ser, entre otras opciones, intervalar, afín o probabilística. Con estas variantes que ofrece QFT, permite una amplia flexibilidad para definir la planta y la incertidumbre de la misma. A pesar de toda esta flexibilidad que ofrece QFT para poder describir la incertidumbre de la planta, la forma paramétrica es la más precisa, y por ello es la elegida para este ejemplo.

1.1.2. Elección de un conjunto de frecuencias de diseño

En esta etapa se eligen las frecuencias de diseño con las que se va a realizar todo el proceso. Suelen ser pocas, pero se eligen estratégicamente para que sean representativas, dado que, posteriormente el diseño se deberá validar con un conjunto de frecuencias más amplio y que siga respetando las especificaciones de diseño. Las formas comunes de introducir las frecuencias suelen ser, por un lado, utilizar un rango de inicio y fin, que puede tener espaciado lineal o logarítmico, y por otro lado, introducir frecuencias de forma manual.

Para el ejemplo, las frecuencias de diseño elegidas son las siguientes:

$$\Omega = \{0.1, 5, 10, 50, 100\}\ rad/s$$

1.1.3. Cálculo de las plantillas

Las plantillas, también conocidas como *templates*, modelizan toda la información disponible sobre la planta, tanto la propia planta nominal como su incertidumbre. Es fundamental el cálculo de unas plantillas adecuadas de la planta y, principalmente, el cómo se representa la incertidumbre en ella. Lo adecuado serían unas plantillas lo más cercanas a la realidad posibles e incluyendo todos los factores que pudieran afectar a la planta. La incertidumbre en las plantillas se puede representar de distintas formas, por un lado se tiene la de más alto nivel, que es la incertidumbre paramétrica, siendo normalmente una buena práctica elegir parámetros que tengan un significado físico. Por otro lado, el nivel más bajo de incertidumbre corresponde a la no estructurada, éstas suelen seguir un modelo nominal en el cual sus variaciones no tienen información de fase, sólo de magnitud, y suelen expresarse a partir de una norma.

El cálculo de plantillas no está resuelto de forma completa para incertidumbre paramétrica, es un tema en investigación, pero existen diversas técnicas, que en general funcionan de forma adecuada para la mayoría de problemas prácticos y permiten calcular, para diferentes niveles de estructura en los parámetros, las plantillas de la planta.

Para el ejemplo de esta sección, en la figura (1.5) se puede observar el resultado del cálculo de plantillas para la planta, junto con su incertidumbre.

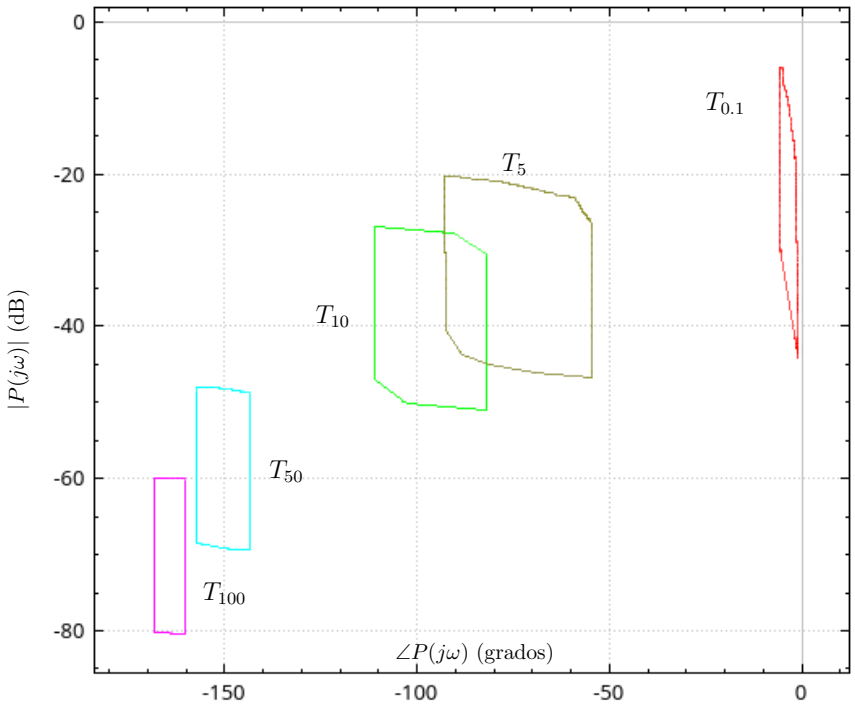

Figura 1.5: Representación en forma de plantilla de la incertidumbre de una planta incierta.

1.1.4. Definición de las restricciones de diseño

En QFT, las especificaciones describen las cotas admisibles sobre la respuesta en frecuencia de las funciones de transferencia de lazo cerrado.

Las restricciones que usualmente se consideran en QFT son las siguientes:

(1) Seguimiento:

$$\alpha(\omega) \leq \left| \frac{L(j\omega)}{1+L(j\omega)} F(j\omega) \right| \leq \beta(\omega) \ \forall \omega > 0 \tag{1.6}$$

(2) Estabilidad:

$$\left| \frac{L(j\omega)}{1+L(j\omega)} \right| \leq \lambda \ \forall \omega > 0 \tag{1.7}$$

(3) Eliminación de ruido del sensor:

$$\left| \frac{L(j\omega)}{1+L(j\omega)} \right| \leq \delta_n(\omega) \ \forall \omega > 0 \tag{1.8}$$

(4) Eliminación de perturbaciones en la salida de la planta:

$$\left|\frac{1}{1+L(j\omega)}\right| \leq \delta_{po}(\omega)\ \forall\omega > 0 \tag{1.9}$$

(5) Eliminación de perturbaciones en la entrada de la planta:

$$\left|\frac{P(j\omega)}{1+L(j\omega)}\right| \leq \delta_{pi}(\omega)\ \forall\omega > 0 \tag{1.10}$$

(6) Esfuerzo de control:

$$\left|\frac{C(j\omega)}{1+L(j\omega)}\right| \leq \delta_{ce}(\omega)\ \forall\omega > 0 \tag{1.11}$$

Se puede observar como la especificación de estabilidad robusta se da en términos de una limitación sobre la magnitud del lazo cerrado, lo que conlleva una especificación simultánea, y por tanto ligada, sobre el margen de fase y el margen de ganancia. Esta limitación conjunta suele recibir el nombre, en el plano de Nichols, de círculo-M.

Para el ejemplo específico que se está tratando, las especificaciones dadas son las siguientes:

(1) Seguimiento:

$$\frac{0.6584(s+30)}{s^2+4s+19.752} \leq \left|\frac{L(j\omega)}{1+L(j\omega)}F(j\omega)\right| \leq \frac{120}{s^3+17s^2+828s+120} \tag{1.12}$$

(2) Estabilidad:

$$\left|\frac{L(j\omega)}{1+L(j\omega)}\right| \leq 1.2 \tag{1.13}$$

Además, está implícita la restricción de estabilidad robusta, que debe respetar el círculo-M en el plano de Nichols.

1.1.5. Cálculo de las fronteras

A continuación, una vez calculadas las plantillas y definidas las restricciones de diseño, el siguiente paso es el cálculo de las fronteras o *boundaries*. El objetivo es el traspaso de dichas especificaciones, descritas en el apartado (1.1.4), y definidas en el dominio de la frecuencia, a regiones del plano de Nichols donde deberá mantenerse la función $L_0(j\omega)$. Las regiones resultantes, que definen los espacios prohibidos y permitidos en el plano de Nichols pueden ser de diferentes tipos. Por ejemplo, pueden ser conexas o no, o pueden

ser cerradas, y que delimiten un espacio concreto dentro del plano, o pueden ser abiertas, y por tanto definen una región, inferior o superior, prohibida. Un cálculo preciso es importante, dado que, van a determinar el coste de la realimentación en términos de ganancia y ancho de banda del controlador.

Dado que el cálculo de fronteras no es un proceso trivial, el uso de algoritmos que proporcionen un método automático es muy adecuado, sobre todo, en aquellos casos que se salen de la norma y tienen una forma más compleja, por ejemplo, las multivaluadas. De forma general, una frontera delimitará una región, normalmente conexa, del plano de Nichols, y que se expresa como una función $M = F(\phi)$. Una frontera es multivaluada cuando la función F es multivaluada, por ejemplo, cuando en el plano de Nichols, para una frecuencia concreta, existe más de un valor de magnitud. En el caso de las fronteras de estabilidad, al ser cerradas, típicamente son, al menos, bivaluadas y las fronteras de seguimiento, que normalmente son abiertas, en muchos casos pueden ser multivaluadas.

Tradicionalmente, estas fronteras eran calculadas manualmente con manipulaciones gráficas sobre el plano de Nichols, pero enseguida empezaron a surgir herramientas informáticas para el diseño asistido. Uno de los primeros artículos donde se aborda el estudio del cálculo de fronteras es (Bailey et al. [1988]). En dicho artículo, además, se dan algoritmos para el cálculo de las especificaciones de estabilidad y seguimiento, y se aborda el problema de las fronteras multivaluadas, aunque el método que aporta no es fácilmente implementable en la práctica. Años más tarde, surgirían nuevos trabajos, por ejemplo, en (Chait y Yaniv [1993]) se propone un algoritmo basado en la resolución de inecuaciones cuadráticas. Posteriormente, en los estudios de (Brown y Peterson [1991]) y (Fadali y LaForge [1996]) se plantean algoritmos para calcular las fronteras. Pero estos algoritmos tienen algunas restricciones y no son generalizables para todos los casos, la principal de las restricciones es en el cálculo de fronteras multivaluadas. También, en (Moreno et al. [2006]) se plantea un algoritmo que resuelve, por fin, el cálculo de fronteras multivaluadas. Está basado en los estudios del concepto de sección de cruces propuesto por (Bailey et al. [1988]).

En el ejemplo que se está tratando en esta sección se obtiene como resultado inicial la figura (1.6).

Una vez disponibles las fronteras de forma definitiva se continua con el siguiente paso de ajuste del lazo.

1.1.6. Ajuste del lazo

A continuación, viene la etapa más complicada de resolver de QFT, es el ajuste del lazo y síntesis del controlador, que se define como el diseño de

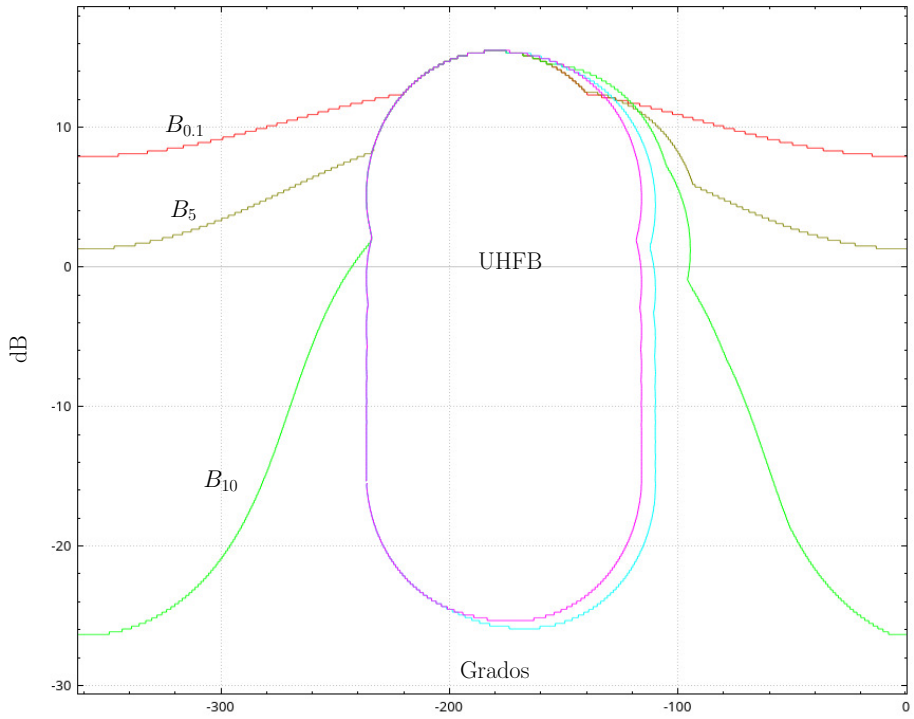

Figura 1.6: Representación de fronteras.

la función de ganancia del lazo nominal ($L_0(s)$). El cálculo del controlador consiste en ajustar la función $L_0(j\omega)$ en el plano de Nichols con el objetivo de que se cumplan las restricciones de diseño minimizando la ganancia de alta frecuencia (k_{hf}). Esto se traduce en que, para cada frecuencia de diseño (ω), de las elegidas en la sección (1.1.2), el lazo nominal ($L_0(j\omega)$) no debe violar las restricciones correspondientes, es decir, debe estar en una región permitida por las fronteras.

De forma general, esta etapa puede describirse como un problema de optimización no lineal con restricciones no convexas (Bailey et al. [1994]), lo cual conlleva que la obtención de la solución sea un problema complejo de resolver. Se han planteado diferentes algoritmos para algunos casos generalistas, por ejemplo, el de una planta racional incierta que contiene ceros de forma estricta únicamente en el semiplano izquierdo (Brown y Peterson [1991]). En esta situación se garantiza la solución, es decir, la existencia de controlador para las especificaciones de estabilidad, seguimiento y rechazo de perturbaciones. En (Blondel [1994]), se muestra cómo la dificultad de resolución del problema es alta, incluso para el caso (general) de estabilización de un conjunto de plantas y se llega a la conclusión de que es un problema *intratable*.

Para el ejemplo estudiado en esta sección, en la figura (1.7) se puede observar el resultado obtenido, siendo controlador que resolvería el ajuste del

lazo de forma óptima, dentro de la estructura proporcionada al algoritmo, el siguiente:

$$C_{QFTex1}(s) = 16.36\frac{(s+96.90)(s+93.81)}{(s+344.40)}$$

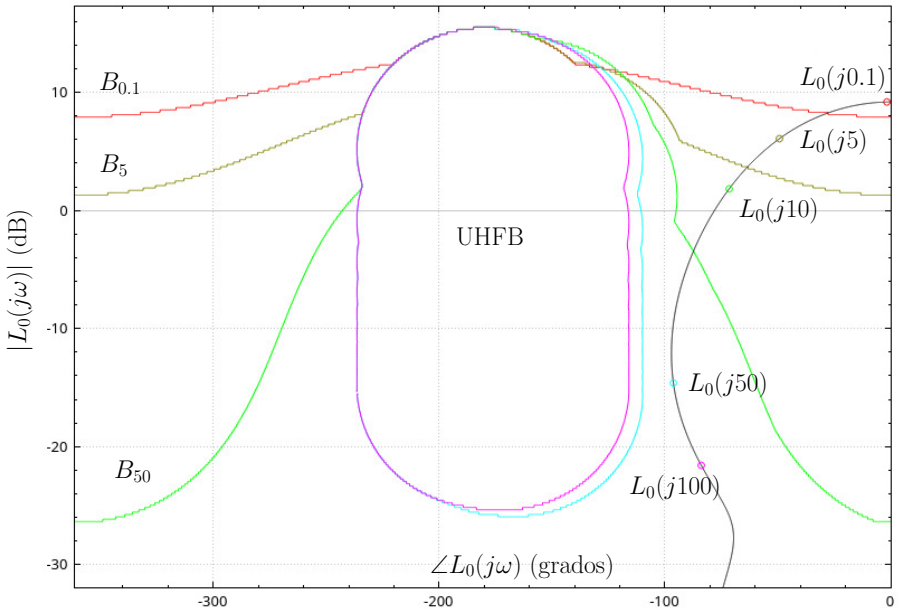

Figura 1.7: Representación de $L_0(s)$.

En la sección de *antecedentes* (2), serán extensamente explicadas las diferentes aproximaciones que han tratado de dar solución a este problema.

1.2. Introducción a Aritmética Intervalar

Si se desea representar, desde el punto de vista computacional, un subconjunto de la recta real que tiene un extremo de inicio y otro de fin, la única forma que existe para dicha representación que garantiza todas las posibilidades, es utilizar un intervalo. Dado que, cualquier otra representación, por ejemplo, una discretización del subconjunto, obligaría a tomar unos valores concretos que no representarían al subconjunto en su totalidad, independientemente de cuán elevado sea el número de valores utilizado para discretizar. Por ello, utilizar intervalos para este tipo de representaciones en la implementación de algoritmos, es la única forma de garantizar la consecución del resultado óptimo. En otras representaciones, al no ser exhaustivas y completas, puede ser que no se encuentre el óptimo aunque exista, dando lugar a un resultado inconsistente.

Varios autores (Dwyer [1951], Sunaga [2009]) han trabajado en cómo minimizar los errores de redondeo de la aritmética computacional. Desde el punto de vista teórico, la aritmética de números reales es completamente precisa y no existen problemas de errores cuando se trabaja con ella, pero, cuando se trata de problemas reales y además, utilizando computadores para resolver dichos problemas, los inconvenientes aparecen. Cuestiones como el límite de los buffer de representación numérica, la pérdida de precisión en algunas operaciones o, la ya comentada, discretización de rangos, suponen un problema real a tener en cuenta. Valga como ejemplo el problema planteado por Rust en (Loh y Walster [2002]).

Como se ha indicado, la utilización de intervalos puede ser muy beneficiosa en ámbitos concretos, pero tiene la problemática asociada de que es necesario realizar operaciones con intervalos. Es decir, al igual que existe una aritmética clásica asociada al conjunto de números reales, es necesario disponer de una aritmética intervalar que permita realizar estas operaciones. Esta aritmética intervalar no estaba desarrollada como la aritmética real clásica y suponía una limitación a la hora de trabajar con intervalos. A continuación, se introducen los conceptos y resultados básicos de aritmética intervalar (Moore [1962], Moore [1966] y Moore y Bierbaum [1979]) que se usan en este trabajo.

1.2.1. Definiciones

A continuación se muestran las definiciones principales en las que se basa la aritmética intervalar.

Definición 1 *Un intervalo cerrado* $[a, b]$ *se define como el conjunto de los números reales dados por*

$$[a, b] = \{x | a \leq x \leq b\}$$

se denotarán a los intervalos con letras mayúsculas y al conjunto de los intervalos como $\mathbb{I}$, siendo X ese intervalo. Entonces los extremos derecho e izquierdo del intervalo se denotarán como $\underline{X}$ y $\overline{X}$, respectivamente, por lo tanto $X = [\underline{X}, \overline{X}]$.

Definición 2 *Se nombrará intervalo degenerado cuando* $\underline{X} = \overline{X}$*, el número real* x *es equivalente a un intervalo de ancho cero, entonces*

$$X = [\underline{X}, \overline{X}] = [x, x]$$

Definición 3 *Sean dos intervalos* X *e* Y*, tal que, su intersección* $(X \cap Y)$ *se define como*

$$X \cap Y = \{z : z \in X \wedge z \in Y\} = [máx\{\underline{X}, \underline{Y}\}, mín\{\overline{X}, \overline{Y}\}]$$

Si se da el caso que los intervalos X e Y no comparten ningún punto z en común, es decir, $\overline{Y} < \underline{X}$ o $\overline{X} > \underline{Y}$, esto tiene como consecuencia que el intervalo resultante será el conjunto vacío $\varnothing$.

Definición 4 *Sean dos intervalos X e Y, tal que, su unión ($X \cup Y$) se define como*

$$X \cup Y = \{z : z \in X \vee z \in Y\} = [mín\{\underline{X}, \underline{Y}\}, máx\{\overline{X}, \overline{Y}\}]$$

cuando al menos existe un punto z compartido entre los dos intervalos.

En general, la unión de intervalos no tiene por qué dar como resultado un solo intervalo, podrían ser más. Para evitar este tipo de situaciones se define la *envolvente convexa*:

Definición 5 *Sean dos intervalos X e Y, tal que, su envolvente convexa ($X \underline{\cup} Y$) se define como*

$$X \underline{\cup} Y = [mín\{\underline{X}, \underline{Y}\}, máx\{\overline{X}, \overline{Y}\}]$$

Como resultado siempre se obtendrá un solo intervalo, además siempre se tendrá que

$$X \cup Y \subseteq X \underline{\cup} Y$$

1.2.2. Relaciones de orden entre intervalos

La cuestión del orden entre intervalos no es un tema sencillo, dado que no siempre tiene solución. En múltiples ocasiones, un intervalo X no es ni mayor, ni menor ni igual que un intervalo Y. A continuación, se definen las reglas que definen dichos órdenes.

Definición 6 *Sean dos intervalos X e Y, para que X sea mayor que Y debe cumplirse que*

$$\underline{X} > \overline{Y}$$

Por el contrario:

Definición 7 *Sean dos intervalos X e Y, para que X sea menor que Y debe cumplirse que*

$$\overline{X} < \underline{Y}$$

Para la igualdad:

Definición 8 *Sean dos intervalos X e Y, tal que, para que X sea igual que Y debe cumplirse que*

$$\overline{X} = \overline{Y} \; y \; \underline{X} = \underline{Y}$$

En cualquier otro caso, por ejemplo, cuando dos intervalos se solapen parcialmente, no existe orden entre ellos.

1.2.3. Operaciones intervalares aritméticas

La aritmética intervalar define una serie de operaciones aritméticas básicas para poder operar con intervalos. En esta sección se van a definir algunas de las operaciones más importantes utilizadas en esta tesis. Para mayor información se puede consultar la página web del software que se ha utilizado en la implementación de los algoritmos, la librería *C-XSC*, desarrollada por el grupo *Scientific Computing / Software Engineering* del departamento *School of Mathematics and Natural Sciences* de la *University of Wuppertal* [1] (Klatte et al. [1993] y Hammer et al. [1995]).

Suma de intervalos:

Dado dos intervalos X e Y, la suma de los mismos sería

$$X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}].$$

Resta de intervalos:

Dado dos intervalos X e Y, la resta de los mismos sería

$$X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}].$$

Multiplicación de intervalos:

Dado dos intervalos X e Y, la multiplicación de los mismos sería

$$X \cdot Y = [\text{mín } S, \text{máx } S], \text{ donde } S = [\underline{X} \cdot \underline{Y}, \underline{X} \cdot \overline{Y}, \underline{Y} \cdot \overline{X}, \overline{X} \cdot \overline{Y}].$$

División de intervalos:

La división de intervalos se define como un caso particular de la multiplicación, entonces, dados dos intervalos X e Y, la división de los mismos sería

[1] Página web: http://www2.math.uni-wuppertal.de/org/WRST/xsc-frame/index.html a fecha 13/05/2021

$$X/Y = X \cdot \frac{1}{Y} = [\underline{X}, \overline{X}] \cdot \left[\frac{1}{\overline{Y}}, \frac{1}{\underline{Y}}\right], \text{ donde } 0 \notin Y.$$

Como se puede observar en la definición, el intervalo que está dividiendo no puede contener el 0, lo cual puede ser una limitación importante en algunas situaciones. Existe una *aritmética intervalar extendida* que da solución a este problema (Kahan [1968], Novoa [1993] y Ratz [1996]).

1.2.4. Vectores de intervalos (Cajas)

Un vector de intervalos, es una tupla de intervalos ordenados de n dimensiones.

$$X = (X_1, \cdots, X_n) \in \mathbb{I}^n.$$

Por otro lado si $x = (x_1, ..., x_n)$ se define como un vector de reales y $x = (X_1, \cdots, X_n)$, entonces

$$x \in X \; si \; x_i \in X_i \forall_i = 1, \cdots, n.$$

Las operaciones sobre vectores de intervalos son similares a las de los intervalos individuales. A continuación, se definen varios operadores de los utilizados en esta tesis.

Supremo de un vector intervalos:

El supremo de un intervalo y de un vector de intervalos X se define como

$$sup(X) = \overline{X} \text{ y } sup(x) = (\overline{X}_1, \cdots, \overline{X}_n).$$

Ínfimo de un vector intervalos:

El ínfimo de un intervalo y de un vector de intervalos X se define como

$$inf(X) = X \text{ y } inf(x) = (\underline{X}_1, \cdots, \underline{X}_n).$$

Posición en un vector de intervalos:

El elemento i-ésimo en un vector de intervalos X se define como

$$x[i] = X_i$$

Operador sustitución:

Devuelve el mismo intervalo sustituyendo el elemento j por el intervalo I, se define como

$$subs(x, j, I) = (X_1, \cdots, X_{j-1}, I, X_{j+1}, \cdots, X_n).$$

1.2.5. Extensión natural a intervalos

Sea función intervalar f como $f : \mathbb{R}^n \to \mathbb{R}$, donde el rango real de la función f sobre el intervalo X es:

$$f(X) = \{f(x) : x \in X\}$$

El cálculo de f es muy variable en dificultad, dependiendo de la operación concreta que se quiera resolver, por ejemplo, para las funciones monótonas, como pueden ser, $exp(x)$ o $log(x)$, el simple cálculo de aplicar las funciones reales sobre sus extremos dará el resultado correcto. Pero existen otras muchas situaciones, donde las funciones no son monótonas y, por tanto, obtener el resultado es mucho más complejo.

Definición 9 *Se define $f(X)$ como el rango de una función f en el intervalo X. Entonces, una función de intervalos $F : \mathbb{I} \to \mathbb{I}$ es una función de inclusión si y, solo si, $f(X) \subseteq F(X) = [\underline{F}(X), \overline{F}(X)]$.*

Definición 10 *Se define una función de inclusión isótona F si para cualquier intervalo $X \subseteq Y$, se tiene que $F(X) \subseteq F(Y)$.*

La aritmética intervalar permite la obtención de una función F de inclusión isótona y que se define como el *Teorema fundamental del análisis de intervalos.*

Propiedad 1 *Teorema fundamental del análisis de intervalos. Se establece que, si F es una extensión natural a intervalos, denotada como F_{NIE}, donde $f : \mathbb{R}^n \to \mathbb{R}$, y*

$$rango(f, \mathbf{x}) \subseteq F(\mathbf{x})$$

*entonces F es una función de inclusión isótona (*Moore *[1966]).*

Siendo *NIE* una extensión de intervalo isotónico de inclusión, mediante la aplicación directa de (1), para cualquier $f : \mathbb{R}^n \to \mathbb{R}$,

$$rango(f, \mathbf{x}) \subseteq F^{NIE}(\mathbf{x})$$

La Extensión natural a intervalos es la función de aritmética intervalar que se utiliza en los algoritmos explicados en esta tesis.

1.3. Introducción a búsqueda global intervalar

El método de diseño de algoritmos llamado *Ramificación y poda* (en la literatura conocidos como *Branch and bound*), es una variante del método de *Bracktracking*. A este método se le dio nombre por primera vez en 1963, en el artículo que planteaba un nuevo algoritmo para el clásico problema del viajante de comercio (Little et al. [1963]). Su aplicación más habitual es para resolver problemas de optimización.

El método fue propuesto inicialmente, aunque no se le dio ningún nombre concreto, para problemas de programación lineal entera (ILP) (Land y Doig [1960]). Posteriormente, en 1966, se consiguió generalizar un método que fuera independiente y que pudiera abordar una gran variedad de problemas.

La técnica de *Ramificación y poda* se suele plantear como un árbol de soluciones que el algoritmo va expandiendo, al igual que hace el método de *Backtracking*, donde cada rama evoluciona en una posible solución posterior a la actual. Las principales características que introduce este método con respecto a los anteriores, es que, como su propio nombre indica, ofrece herramientas para ramificar y podar las ramas de dicho árbol.

En un algoritmo de Ramificación y poda, un nodo representa, en un momento concreto, una porción del espacio de parámetros. El denominado, *nodo actual* es el nodo seleccionado por el algoritmo para trabajar con él. Dicho nodo, entre otras posibles opciones, será particionado, con el objetivo de dividir el espacio de búsqueda en porciones más pequeñas. Esta forma de desarrollar el espacio de búsqueda se llama árbol y, cada una de sus ramas, que pueden estar más o menos expandidas, son caminos hacia posibles futuras soluciones situadas en las hojas del mismo al final de la rama. Por ello, al espacio de búsqueda del algoritmo, también se le suele denominar comúnmente árbol de búsqueda.

A continuación, se muestran las partes clave del método de *Ramificación y poda*:

- **División**: Su objetivo es particionar un nodo X en un número finito de subnodos X_i de tal forma que satisfaga que $X = \cup X_i$, es decir, entre todos los hijos generados, deben contener el mismo espacio de parámetros que el nodo padre. Existen múltiples formas de realizar este particionado. Por ejemplo, la elección del número de subnodos resultantes, donde una opción común suele ser la bisección, pero también existe la multisección. Otra elección que se puede hacer es la orientación del particionado.

- **Acotación**: Calcula cotas (superior y/o inferior) del valor del rango de la función objetivo para cada nodo, de esta forma se utiliza para acotar el *nodo actual* del algoritmo y reducir el espacio de búsqueda correspondiente.

- **Selección**: Determina cuál es el siguiente nodo a ser procesado. De todos los nodos posibles que el árbol de búsqueda tenga desplegados, el algoritmo debe elegir cual es el siguiente con el que va a trabajar. Esta selección es fundamental, dado que, es la forma principal de ramificación del método. Elegir un nodo que construya una rama prometedora y que se acerque o consiga la solución de forma eficaz garantiza unos tiempos de ejecución más rápidos.

- **Eliminación**: Establece cuándo un nodo no es viable para encontrar solución y es eliminado del árbol de búsqueda. También conocido como poda, es una parte fundamental del método, dado que permite eliminar las ramas por las cuales el algoritmo no va encontrar nada prometedor. Esta situación se puede dar cuando un nodo no contiene solución o cuando se tiene la certeza de que existen mejores soluciones que las que ofrece el nodo.

- **Finalización**: Determina cuándo un nodo es solución, es decir, pertenece al conjunto final de soluciones.

A continuación, se muestra el pseudocódigo del algoritmo clásico de *Ramificación y poda*:

Algoritmo 1.1: Pseudocódigo algoritmo Ramificación y Poda

```
1  Inicializar lista de nodos vivos Λ := S
2  Inicializar lista de conjunto final Θ := ∅
3  Inicializar cota superior ḡ* = +∞
4
5  mientras (λ ≠ ∅) hacer
6
7    Seleccionar nodo v de λ // Regla de selección
8    Actualizar f̄ * si F̄(x) < f̄*, x ∈ v
9    Particionar v generando v_i subnodos con i = 1 ··· k // Regla de división
10
11   Desde i = 1 hasta k
12     Evaluar v_i        // Regla de acotación
13     Si v_i puede ser eliminado // Regla de eliminación
14       Descartar
15     Sino
16       Si v_i cumple el // Regla de finalización
17       criterio de finalizaci ón
18         Insertar v_i en ω
19       Sino
20         Insertar v_i en λ
21
22
23 retornar ω
```

Capítulo 2

Antecedentes

En esta sección se hace un resumen histórico de los métodos de resolución automática del lazo que se han propuesto a lo largo del tiempo en QFT. Tradicionalmente, la resolución de esta etapa se realizaba de forma manual, con lo cual, la resolución del problema dependía exclusivamente de la experiencia y habilidad de la persona que lo realizara. Posteriormente, se introdujo el uso de herramientas CACSD, que básicamente es software que ayuda al experto de control a resolver el problema (por ejemplo el toolbox de QFT diseñado para Matlab (Borghesani et al. [2003])). Este tipo de software tiene mejoras relevantes con respecto al método manual, aunque sigue sin ser automático y requiere por tanto, la intervención fundamental de un experto.

2.1. Introducción

Como se comentó anteriormente, el ajuste automático del lazo en QFT es un problema de optimización no lineal y no convexo, y ello conlleva que es un problema de difícil solución. Por eso, a lo largo de los años se han realizado diferentes propuestas. En la introducción (1.1), se ha mencionado la opción de simplificar el problema para transformarlo en otro distinto más sencillo de resolver, el cual, sí que pueda afrontarse con métodos algorítmicos tradicionales. Pero esta forma de proceder, es una forma conservadora de afrontar el ajuste del lazo, dado que no se está resolviendo el problema original y, por tanto, de forma general, no se obtendría la solución óptima. Para aplicar este método, es necesario elegir un equilibrio entre el grado de simplificación del problema (cuanto más simple, más sencillo de resolver, pero más alejado del problema real y de la solución óptima) y, por otro lado, la dificultad de resolución (cuanto más cercano al problema original, mejor será la solución y más cercana al óptimo, pero más difícil será de resolverlo).

Otra opción es estudiar estructuras concretas para usar en el controla-

dor, que tengan un cierto grado de libertad, y que puedan ser adecuadas para el problema concreto a resolver. Esta solución limita mucho las opciones del controlador y es posible que no consiga alcanzar el óptimo que con otra estructura sí que podría conseguirse. Otra alternativa disponible son los algoritmos evolutivos, que son capaces de abordar problemas de optimización no lineal y no convexa y, aunque tienen ciertas ventajas, también tienen inconvenientes importantes, como por ejemplo, que no garantizan el óptimo global, dado que puede finalizar en un óptimo local, o también, los grandes recursos computacionales que requieren.

Para terminar, el último de los métodos para resolver el ajuste automático del lazo, son los algoritmos de optimización global intervalar, que se basan en el método clásico de *Ramificación y poda* junto con aritmética intervalar. Es el único método automático que asegura el óptimo global, pero a cambio, tiene unos costes computacionales potencialmente altos.

2.2. Algoritmos que aproximan el problema por simplificación

Esta serie de algoritmos realizan aproximaciones sucesivas que tienen como objetivo ir dando una mejor solución en cada una de las iteraciones. Su principal inconveniente es que no suelen alcanzar el óptimo, y su aproximación más cercana como mucho llega a ser un pseudóptimo.

2.2.1. Algoritmo Gera y Horowitz

El primero de los algoritmos propuestos (Gera y Horowitz [1980]), se basa en la repetición iterativa por parte del algoritmo. En cada una de las iteraciones se aproxima a los *boundaries* en forma de línea recta. Este algoritmo se basa en el documento (Horowitz [1973]) donde se demostraba que el resultado óptimo para el ajuste del lazo se sitúa, para cada frecuencia de diseño ω, justo sobre la frontera $\psi(\omega)$, siendo ésta la unión de todas las restricciones impuestas para la frecuencia ω.

El funcionamiento del procedimiento se basa en la parametrización de cada una de las fronteras como una recta tangente, es decir, la sustitución por aproximación de las fronteras reales $\psi(\omega)$, por fronteras rectilíneas $Slt(\omega)$. Una vez obtenidas estas fronteras rectilíneas, se calcula, para cada una de ellas, el punto de la frontera sobre el cual se situaría el lazo óptimo para cada frecuencia (ω) correspondiente, $Lopt(\omega)$. Una vez realizada esta primera iteración, para cada frecuencia (ω) en la cual $Lopt(\omega)$ no se sitúe sobre

$\psi(\omega)$, se inicia una nueva iteración por cada frecuencia en la que se desplaza $Slt(\omega)$ hasta que coincida $Lopt(\omega)$ con $\psi(\omega)$. Es posible que sea necesario añadir más frecuencias de diseño para poder seguir iterando con el algoritmo.

El lazo resultante, si se toman las decisiones adecuadas, debe corresponder al óptimo del problema original, es decir, sin linealizar las *fronteras*. Pero el resultado se da en forma de aproximación numérica que deberá ser tratada para expresarse en forma racional. El funcionamiento de este algoritmo estaba pensado para realizarse de forma manual y está muy basado en decisiones intuitivas del diseñador, por tanto, si las decisiones no se toman de forma adecuada el resultado no sería el óptimo (Yaniv y Nagurka [2004b]). Además, la convergencia al óptimo no está garantizada (Chait et al. [1999]). El procedimiento fue automatizado por (Ballance y Gawthrop [1991]), donde adicionalmente, se introdujeron algunas mejoras como la simplificación de las iteraciones.

2.2.2. Algoritmo Thompson y Nwokah

El siguiente de los algoritmos, es el planteado por Thompson y Nwokah (Thomson [1990]), el cual está basado en la simplificación de las plantillas de incertidumbre de la planta, que transforman las plantillas originales, que tienen formas arbitrarias, en rectángulos que las envuelven en el plano de Nichols.

Con esta simplificación, el ajuste del lazo original se transforma en un problema considerablemente más sencillo de resolver. Donde la violación de las fronteras se limita a la comprobación en un análisis de casos, de las 4 esquinas del rectángulo de la frontera simplificada, así como, un punto adicional por cada lado. De esta forma el problema original de optimización no lineal y no convexa se transforma en un problema no lineal convexo, que se resuelve mediante un algoritmo de optimización no lineal basado en gradiente analítico. Posteriormente, en (Thompson y Nwokah [1994]), se presenta una solución con aproximaciones circulares de las fronteras.

El problema asociado a este algoritmo es que encuentra soluciones conservadoras que no son óptimas, dado que al simplificar el problema, la solución se aporta sobre el problema simplificado, no sobre el original. Y ésta, estará más alejada del óptimo conforme la envolvente calculada tenga un área significativamente mayor que las fronteras originales.

2.2.3. Algoritmo Bryant y Halikias

Para terminar los algoritmos de este tipo, la propuesta de (Bryant y Halikias [1962]), aplica una simplificación del problema aproximando las fronteras mediante inecuaciones lineales para luego resolver mediante un algoritmo de programación dinámica. Este algoritmo tiene problemas como que el resultado que ofrece es conservador y no óptimo al tratarse de una simplificación, además sólo es adecuado para sistemas de fase no mínima y no asegura la estabilidad del lazo cerrado.

2.3. Algoritmos que limitan la estructura del controlador

Los siguientes algoritmos abordan el problema completo de optimización no lineal y no conexa pero, para ello, limitan la estructura del controlador resultante, y así, simplifican el problema a resolver.

Por tanto, aunque resuelven el problema original y completo, el resultado no tiene por qué ser el óptimo. Dado que al limitar la estructura del controlador, el algoritmo sólo puede optimizar hasta donde la estructura le permite.

2.3.1. Algoritmo Chait et al.

En (Chait et al. [1999]) se propone un algoritmo de programación lineal que aborda el problema original con una transformación no simplificadora, que pasa de no lineal y no convexo, a lineal y convexo. Para ello, las restricciones de lazo abierto de la función de transferencia L, se transforman en restricciones de la función de lazo cerrado $T = \frac{L}{1+L}$.

Esta transformación sin simplificar el problema original se basa en limitar la estructura del controlador, en este caso, prefijando los polos del mismo. Por tanto, el algoritmo optimizador sólo podrá optimizar los ceros del controlador para una estructura dada. Esta limitación es problemática, dado que en general, la solución que se obtenga como resultado estará lejos del óptimo.

2.3.2. Algoritmo Nandakumar et al.

El siguiente método (Nandakumar et al. [2002]) propone una serie de estructuras concretas que permiten obtener el controlador óptimo para una planta incierta. El objetivo es conseguir, a partir de una estructura concreta de controlador, el resultado óptimo permitido por esa estructura. Al

ser estructuras prefijadas, el resultado no tiene por qué ser el óptimo global.

Las estructuras propuestas son las siguientes:

- Un PID con la siguiente función:

$$K(s) = k_p + k_d s + \frac{k_i}{s}$$

- Una red de adelanto/atraso de primer orden con la siguiente función:

$$K(s) = \frac{k(s+b)}{s+a}$$

- Un sistema de segundo orden con polos y ceros complejos con la siguiente función:

$$k(s) = \frac{k}{s^2 + s\zeta\omega_n s + w_n^2}$$

Para las tres estructuras descritas se propone que, cada una de ellas tenga tres parámetros libres. Y así, determinar el lazo óptimo para una planta incierta, salvo un factor de escala, utilizando la especificación de la fase del lazo para dos frecuencias ω_i y ω_j. Estas dos frecuencias serían elegidas de, entre las frecuencias de diseño, para generar un muestreo discreto de las posibles fases del lazo. Por tanto, el método planteado, más que un algoritmo de optimización, se podría considerar un algoritmo que genera un conjunto denso de controladores, de entre los que elije al mejor candidato obtenido.

Como se puede observar, no se realiza ninguna simplificación del problema original. Sólo se aborda su solución con una serie de estructuras concretas, que además, son relativamente simples y con pocos grados de libertad. Por tanto, las soluciones obtenidas por esta aproximación estarán alejadas del óptimo. Los propios autores reconocen este problema y añaden que se podrían concatenar varias de estas estructuras, pero no plantean una forma concreta de hacerlo de forma conjunta, y se limitan a explicar su aplicación para cada una de ellas por separado. Por contra, y así mismo se indica en el trabajo, la composición de lazos óptimos, no asegura como resultado un lazo óptimo compuesto. Anteriormente a este trabajo, en (Zolotas y Halikias [1999]), se presentan resultados previos basados únicamente en el controlador PID.

2.3.3. Algoritmo Fransson et al.

El siguiente de los algoritmos (Fransson et al. [2002]), tiene un funcionamiento similar al expuesto en el punto anterior, dado que utiliza un controlador PID de estructura fija para obtener la solución. Dicha estructura tiene la siguiente forma:

$$K_{PID}(s) = k_1 \frac{1 + 2\zeta\tau s + (\tau s)^2}{s\left(1 + s\frac{\tau}{\beta}\right)}$$

Al usar una estructura de control fija, lo habitual es que la solución obtenida quede lejos del óptimo de la planta dado que está limitado a la mejor solución posible que ofrezca esa estructura. Por otro lado, a diferencia de algoritmos anteriores, éste sí que se puede definir como un verdadero proceso de optimización porque, utilizando técnicas algorítmicas que combinan búsquedas locales y globales, tiene como objetivo minimizar k_{hf}. Para terminar, esta propuesta aportó como novedad que es necesario considerar un balance adecuado entre las prestaciones obtenidas a baja, media y alta frecuencia.

2.3.4. Algoritmo Yaniv y Nagurka

El último de los algoritmos de este tipo es (Yaniv y Nagurka [2004b]), que se basa en la generación mediante algoritmo de un conjunto denso de controladores para luego elegir la mejor propuesta. Plantea una solución no iterativa con una estructura de controlador de dos parámetros, inicialmente propuesta en (Yaniv y Nagurka [2003]) para controladores PI, y ampliada para cualquier tipo de controlador de dos parámetros en (Yaniv y Nagurka [2004a]).

Para construir el controlador, se parte de una definición de lazo basada en las plantas $P_1(s)$ y $P_2(s)$, que pertenecen al conjunto de pares de plantas $P_1(s), P_2(s)$ y que, utilizando la definición de lazo, adopta la forma

$$L(s) = a(P_1(s) + bP_s(s))$$

donde a y b son dos números escalares que sirven como par de parámetros del controlador. Se pueden utilizar diferentes formulaciones de $P_1(s)$ y $P_2(s)$ que combinadas obtendrán diferentes estructuras de controladores que seguirán definiéndose por dos parámetros.

Por ejemplo:

- $P_1 = P$ y $\frac{P_2}{P_1} = s$ es equivalente a un controlador PD,

$$G(s) = a(s + bs)$$

- $P_1 = \frac{P}{s}$ y $\frac{P_2}{P_1} = 1$ es equivalente a un controlador PI,

$$G(s) = a\frac{1}{1 + bs}$$

- $P_1 = \frac{P}{1+\frac{s}{p}}$ y $\frac{P_2}{P_1} = s$ es equivalente a un controlador PD aumentado con un filtro paso bajo,

$$G(s) = a\frac{1 + bs}{2 + \frac{s}{p}}$$

El funcionamiento del algoritmo es el siguiente: se determina cual es la región del plano bidimensional, dado que son dos parámetros (a, b), donde están representados los controladores admisibles. La solución óptima estará en la frontera de esta región, con lo cual, es necesario calcular la frontera y, adicionalmente, el punto donde se minimice el producto de $a \cdot b$ estará el resultado óptimo.

El método permite ampliar el número de parámetros realizando una serie de procedimientos adicionales, por ejemplo, para la siguiente propuesta con 3 parámetros

$$G(s) = \frac{a(1 + bs)}{1 + \frac{s}{c}}$$

se pueden seguir los siguientes pasos: en primer lugar, se fija el polo c, esto simplifica el problema a encontrar la frontera de los pares (a, b) para un controlador de dos parámetros y se realiza aplicando el algoritmo original. A continuación, se calcula la frontera para los mencionados parámetros y se elije el mejor resultado que minimice la k_{hf}. Se continúa repitiendo los pasos 1 y 2 para un rango razonable de valores de c. Y para terminar, con los resultados obtenidos con el añadido del nuevo parámetro, se vuelve a elegir un nuevo controlador que minimice la k_{hf}. Que en este caso sería el que obtenga como resultado el menor producto de $a \cdot b \cdot c$.

Como se puede observar, la ampliación de los parámetros del controlador se acaba reduciendo a una búsqueda exhaustiva. Es cierto que esta búsqueda se reduce a dos parámetros, dado que el algoritmo resuelve los dos primeros. Pero, para el resto de parámetros adicionales, no se produciría una diferencia significativa con respecto a un algoritmo de fuerza bruta y, por tanto, su tiempo de ejecución acabaría en una curva de tipo exponencial con respecto al número de parámetros.

2.4. Algoritmos evolutivos

Los algoritmos que se muestran a continuación son del tipo evolutivo. Basan su funcionamiento en simular la evolución de una población de individuos en el tiempo como si fuera una evolución biológica. Cruzando las poblaciones y avanzando las que sean más aptas para encontrar la solución. Este tipo de algoritmos se pueden aplicar a cualquier problema de optimización, incluido el ajuste automático de lazo, siendo éste, como ya se ha indicado, un problema de optimización no lineal y no convexo, permitiendo abordar el problema original sin necesidad de simplificación. Por contra, este tipo de algoritmos tienen como problemática el alto consumo computacional que requieren para llegar a una solución, y que no aseguran el óptimo global, dado que podrían retornar como resultado un óptimo local.

2.4.1. Algoritmo Chen et al.

El primero de los métodos de este tipo es (Chen et al. [1998]), que propone un algoritmo genético donde se hace un diseño directo del controlador. En el que la estructura del controlador es elegida por el diseñador y se optimizan los coeficientes de su función de transferencia. Por otro lado, la estabilidad nominal del lazo se resuelve de forma numérica, mediante el cálculo de las raíces del polinomio. Y la violación de las fronteras se evalúa de forma gráfica, estando éstas representadas por un conjunto discreto de puntos en el plano de Nichols y utilizando interpolación entre cada uno de los puntos consecutivos.

La función objetivo se define de la siguiente forma

$$J = J_{hg} + \sum_{i=1}^{h} \gamma_i J_{bi} + \gamma_0 J_{sta}$$

donde γ_i, $i = 0, 1, \cdots, h$, se utilizan como ponderadores de la función objetivo, con la finalidad de penalizar lo suficiente la violación de las fronteras para que predomine la minimización de k_{hf}.

2.4.2. Algoritmo Raimúndez

El segundo de los algoritmos dentro de esta sección es (Raimúndez [1997]), ampliado en (Raimúndez et al. [2001]), en el cual se utiliza un método evolutivo para afrontar el problema completo de optimización no lineal y no convexo.

Este algoritmo se enfoca en optimizar las localizaciones de los polos y ceros del controlador. De esta forma, según indican los autores, se evitan las que potencialmente pueden ser grandes diferencias de escala en los coeficientes de la función de transferencia en los controladores de alto orden. Por otro lado, cuando es el polo el parámetro que participa directamente en la optimización, es más sencilla de evaluar la estabilidad.

En el caso de un controlador inestable, se asigna automáticamente como $f = \infty$ para que, de esta forma, este controlador quede en la práctica eliminado de las posibilidades. Para el resto de casos, la función objetivo que utiliza este algoritmo es la siguiente:

$$f = k_1\omega_c + k_2\mathcal{B}(G_c(\pi, j\omega)G_p(\pi, j\omega)) + k_3\mathcal{C}(G_c(\pi, j\omega)G_p(\pi, j\omega))$$

donde k_1, k_2 y k_3 son constantes de rango, ω_c es la frecuencia de cruce y π es el conjunto de parámetros a optimizar. Además, las funciones $\mathcal{B}(\ldots)$ y $\mathcal{C}(\ldots)$ son las encargadas de penalizar el coste de la violación de fronteras y el coste de la realimentación respectivamente.

En este algoritmo, a diferencia de (Chen et al. [1998]), el objetivo no es minimizar directamente k_{hf}. Si no que, se minimiza de forma indirecta mediante la minimización conjunta de la frecuencia de cruce (ω_{cg}) y la definición y minimización de $\mathcal{C}(\ldots)$.

2.5. Algoritmos que usan búsqueda global intervalar

Los algoritmos de ajuste automático del lazo que usan búsqueda global intervalar, aplican un método de Ramificación y poda (1.3) clásico, junto con Aritmética intervalar(1.2) para definir el espacio de búsqueda del algoritmo.

El método de Ramificación y poda, considera un conjunto discreto y finito de nodos vivos donde se representa el espacio de parámetros y que define el espacio de búsqueda del algoritmo. Si fuera un enfoque no basado en intervalos sería necesario discretizar, de alguna forma representativa, dicho espacio de parámetros, perdiendo con ello precisión y posibles soluciones. Al usar intervalos, el conjunto es continuo e infinito, por tanto, todas las soluciones están contenidas y el algoritmo siempre encontrará la óptima.

La principal problemática asociada a los algoritmos de búsqueda exhaustiva es el alto coste computacional que tienen y, por lo tanto, el tiempo de ejecución termina dibujando una curva exponencial con respecto al tamaño del problema. Para solucionar este problema, se han propuesto diferentes herramientas cuyo fin principal ha sido el reducir el espacio de búsqueda del

algoritmo, consiguiendo así disminuir el tiempo de ejecución.

El primero de los algoritmos que abordan el ajuste automático del lazo utilizando este método es el algoritmo NT (Nataraj y Tharewal [2002], Nataraj y Tharewal [2003] y Nataraj y Tharewal [2004]). En él, se propone un diseño de algoritmo que da solución al problema donde se utiliza aritmética intervalar. Además, para mejorar el tiempo de ejecución, se propone un método para acotar el espacio de parámetros centrado en la variable que representa la ganancia de alta frecuencia, y que tiene por objetivo eliminar las partes del rango de esa variable que violan las restricciones de diseño.

La problemática que tiene este algoritmo es que, al centrarse sólo en la ganancia de alta frecuencia y no plantear ninguna propuesta para los polos y ceros del controlador, sólo se aborda una parte concreta del espacio de parámetros, dejando el resto intacto. Por tanto, la mejora que pueda aportar esta solución está limitada por la influencia que tenga la ganancia de alta frecuencia en la incertidumbre del controlador.

El segundo de los algoritmos, NK (S. V. Paluri y Kubal [2007]), utiliza como base el algoritmo anterior y añade nuevas propuestas para acotar la parte inviable de todas las variables del controlador. Se basa en el procedimiento creado para reducir la ganancia de alta frecuencia y propone dos métodos que se aplican tanto en los polos como en los ceros del controlador.

Además, propone utilizar un algoritmo de búsqueda local, que trate de encontrar alguna solución de forma rápida al problema, y así poder utilizar dicha solución para realizar podas en el árbol de búsqueda. Este algoritmo de búsqueda local es muy improbable que encuentre la solución óptima, pero sí que es posible que encuentre un óptimo local, y con ello, poder descartar todo el espacio de parámetros que sea menos prometedor (poda).

El siguiente de los métodos propuestos (Nataraj y Deshpande [2008]), se basa en el mismo método ya comentado, pero añade nuevas funcionalidades para recortar el espacio de parámetros, e inaugura un nuevo tipo de algoritmos que utilizan técnicas de propagación de restricciones (*Constraint Satisfaction Techniques* en la literatura sobre el tema).

Este tipo de técnicas tiene como objetivo la propagación al espacio de parámetros de las restricciones de diseño impuestas, es decir, cumplir las restricciones de las fronteras. Estas técnicas se basan en el traslado de las restricciones de diseño a inecuaciones cuadráticas que luego se aplicarán para cada una de las variables del controlador. Con su resolución se sabrá qué parte de cada una de estas variables cumple con dichas restricciones y cuál

no, pudiendo eliminar la parte inviable para, de esta forma, reducir el espacio de búsqueda.

Posteriormente en (Rambabu Kalla y P. S. V. Nataraj [2010]), se amplió este algoritmo para que se pudiera aplicar a controladores fraccionales y, en (Patil y Nataraj [2012]) se extendió para sistemas multivariables.

El siguiente capítulo se extiende, de forma detallada, la explicación sobre este tipo de algoritmos de búsqueda global intervalar.

Capítulo 3

Algoritmos ALS intervalares

En esta sección se va a entrar en detalle en los algoritmos de ajuste automático del lazo que utilizan la búsqueda global intervalar para encontrar la solución. Como ya se ha mencionado en la sección (2.5), este tipo de algoritmos usan aritmética intervalar para representar la incertidumbre en la planta a controlar, así como para representar el rango de las variables (polos, ceros y ganancia de alta frecuencia) del controlador. Están basados en el método de resolución de algoritmos de Ramificación y Poda

Como ya se ha comentado, este tipo de algoritmos, al ser una búsqueda global, ofrecen la seguridad de encontrar la solución óptima al problema que se esté planteando. Además, al usar intervalos para representar el rango de las distintas variables, ya sea la incertidumbre de la planta o la variabilidad del controlador, permiten no tener que discretizar dichos rangos, y así, no perder precisión con las operaciones. Pero tienen un problema fundamental, y es el asociado a las búsquedas exhaustivas, que son, por definición, muy lentas en tiempo de ejecución y que definen una curva exponencial conforme crece el tamaño del problema.

Por ello, los diferentes algoritmos propuestos en esta sección, tratan de aportar diferentes mejoras y variaciones del algoritmo base para conseguir reducir el coste computacional, generalmente, tratando de reducir el espacio de búsqueda.

En las siguientes subsecciones se usará la notación introducida en el apartado (1.2).

3.1. Algoritmo Nataraj y Tharewal (NT)

En (Nataraj y Tharewal [2002], Nataraj y Tharewal [2003] y Nataraj y Tharewal [2004]) se plantea un algoritmo de ajuste automático del lazo que usa aritmética intervalar tanto para representar la incertidumbre de la planta, como la variabilidad del controlador. El algoritmo es una búsqueda exhaustiva en el espacio de parámetros del controlador, que se denomina *caja* y se denota como $\mathbf{x} \in \mathbb{R}^n$, siendo n el número de parámetros a optimizar.

La idea básica es usar la propiedad isotónica de inclusión de la aritmética intervalar (1.2.5), conocida como Extensión natural de intervalos, que permite realizar operaciones aritméticas considerando los rangos de las variables del controlador como intervalos y, de esta forma, poder calcular el resultado del controlador incierto. Asimismo, una vez calculado el resultado, se obtiene un número complejo intervalar que podrá ser proyectado en el plano de Nichols. En dicha proyección, podrá ser comparado con las fronteras para comprobar que respeta las restricciones de diseño. El *Teorema fundamental del análisis de intervalos* (1) nos asegura que la relación que se establezca en el plano de Nichols entre la caja proyectada y las fronteras, viable, ambigua o inviable (ver figura (3.1)), será la misma que la caja original y las restricciones de diseño impuestas. Eso sí, es necesario tener en cuenta que aplicar esta operación implica cierto grado de conservadurismo dada la pérdida de precisión que supone.

Como requisitos del algoritmo se asumen una serie de características iniciales. En primer lugar, el algoritmo se plantea para una planta incierta concreta P, con un conjunto de frecuencias de diseño elegidas Ω, un conjunto de especificaciones S_ω, y una estructura de controlador $(n_{rz}, n_{cz}, n_{rp}, n_{cp})$.

A continuación, el algoritmo requiere tener calculados tanto las plantillas (T_ω) como las fronteras (B_ω) correspondientes que son los datos base que utilizará. Seguidamente, para poder iniciar el procedimiento, es necesario plantear una estructura de controlador concreta, así como el espacio de parámetros inicial del mismo, que se traduce en indicar los intervalos correspondientes en cada variable del controlador a optimizar.

A continuación, se muestra el pseudocódigo del algoritmo (3.1) dividido en secciones:

Algoritmo 3.1: Pseudocódigo del algoritmo *Nataraj-Tharewal (NT)*

1
2 ***Entradas***: B_Ω y $\mathbf{x}$, caja inicial de búsqueda
3
4 ***Salidas***: $\mathbf{z}$, caja con los parámetros del controlador calculados en sus valores óptimos, o mensaje de que "No existe solución válida en x".
5

❶ ***A***. Comprobar factibilidad de la caja inicial de búsqueda:

$\mathbf{z} \leftarrow \mathbf{x}$ (inicializar caja actual con la caja inicial)

$\mathbf{R}_{\mathbf{z}\Omega} \leftarrow (\text{mag}(\mathbf{z},\omega), \text{ang}(\mathbf{z},\omega)), \forall\omega \in \Omega$

$flag_{\mathbf{z}} \leftarrow \text{FT}(\mathbf{R}_{\mathbf{z}\Omega}, \text{B}_\Omega)$

Si $flag_{\mathbf{z}}$ = ***inviable***, *imprimir* "No existe solución válida en $\mathbf{z}$" y *salir*

❷ ***B***. Inicializar la lista ***NL***:

$z \leftarrow \inf(\mathbf{z}[1])$ (z inicialización)

$NL \leftarrow \{(\mathbf{z}, z, flag_{\mathbf{z}})\}$.

❸ ***C***. ***Si*** es solución $\boldsymbol{z}$, *imprimir* "La caja óptima de parámetros del controlador es z" y *salir*

❹ ***D***. Biseccionar $\mathbf{z}$ por la dirección más larga de la caja en el espacio de parámetros en dos subcajas $\mathbf{v}^1$ y $\mathbf{v}^2$ tal que $\mathbf{z} = \mathbf{v}^1 \cup \mathbf{v}^2$

❺ ***E***. Realizar prueba de factibilidad a las nuevas subcajas y descartar las que sean ***inviables***, i.e., para $l = 1, 2$:

$\mathbf{R}_{\mathbf{v}^l\Omega} \leftarrow (\text{mag}(\mathbf{v}^l,\omega), \text{ang}(\mathbf{v}^l,\omega)), \forall\omega \in \Omega$

$flag_{\mathbf{v}^l} \leftarrow \text{FT}(\mathbf{R}_{\mathbf{v}^l\Omega}, \text{B}_\Omega)$

$v^l \leftarrow \inf(\mathbf{v}^l[1])$

Formar tripleta $(\mathbf{v}^l, v^l, flag_{\mathbf{v}^l})$

Si $flag_{\mathbf{v}^l}$ = ***inviable***, descartar $(\mathbf{v}^l, v^l, flag^l_{\mathbf{v}})$

❻ ***F***. Acotación y recorte de la ganancia de alta frecuencia

Para $l = 1, 2 : \mathbf{v}^l \leftarrow CP(R_{v'\Omega}, B_\Omega, \Omega, \mathbf{v}^l)$

❼ ***G***. Ordenar lista ***NL***

$NL \leftarrow NL - \{(\mathbf{z}, z, flag_{\mathbf{z}})\}$

Incorporar las tripletas restantes del paso ***F***.(e) a ***NL***.

Si NL es vacía, entonces no existe solución válida en $\mathbf{x}$, *imprimir* "No existe solución válida en $\mathbf{x}$" y *salir*

Ordenar ***NL*** de modo que los segundos miembros de las tripletas en ***NL*** sean crecientes

Denotar el primer elemento de la lista ***NL*** como $(\mathbf{z}, z, flag_{\mathbf{z}})$

❽ ***H***. Ir a la etapa ***C***

Se procede a explicar cada una de las secciones del algoritmo.

3.1.1. Comprobar factibilidad de la caja

Corresponde a las secciones (❶ y ❺) del algoritmo. Es una de las partes principales donde el procedimiento comprueba si la caja respeta las especificaciones de diseño o no, y de qué forma lo hace. Esta comprobación se realiza en el plano de Nichols, para ello, en primer lugar es necesario proyectar la caja de parámetros del controlador en dicho plano mediante la utilización de NIE. El procedimiento se realiza calculando $mag(\mathbf{x},\omega)$ y $ang(\mathbf{x},\omega)$ siendo estas

funciones operaciones intervalares definidas como $|L_0(j\omega, x)|$ y $\angle L_0(j\omega, x)$. La caja resultante se proyecta en el plano de Nichols con la magnitud en decibelios y la fase en grados.

A continuación, para cada frecuencia de diseño (ω) se dibujan, también en el mismo plano, las fronteras previamente calculadas y se comparan con cada una de las cajas proyectadas para cada frecuencia. En la figura (3.1) se pueden observar distintas posibilidades.

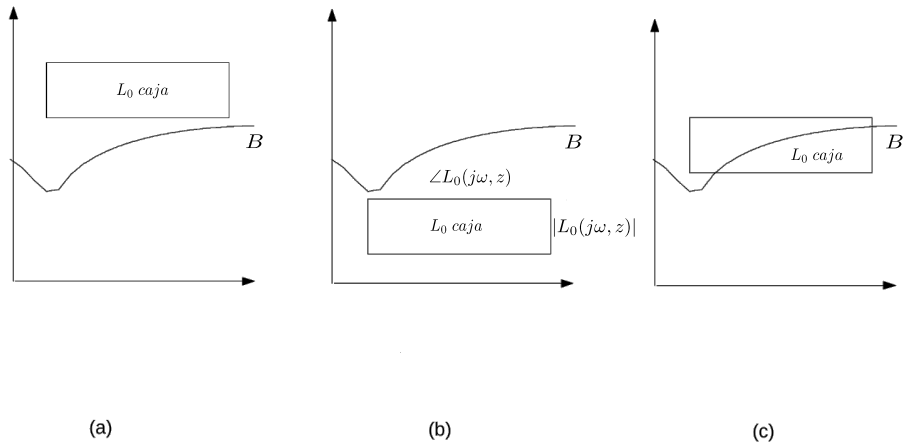

Figura 3.1: Opciones de interacción entre la caja y la frontera en el plano de Nichols.

Para este ejemplo, la frontera B_i hace viable el lado superior e inviable el lado inferior del plano de Nichols. Entonces, tal y como se observa en la figura, se pueden producir tres situaciones:

- En la figura (a) la caja proyectada cumple con las especificaciones de diseño, por lo tanto, esta caja se marca como tal y podrá tomarse como una posible solución al problema. Este resultado se denominará como viable.
- En la figura (b) la caja proyectada interseca con las fronteras y no se puede determinar qué partes respetan las restricciones de diseño y cuáles no. Este tipo de caja se denomina ambigua.
- En la figura (c) la caja proyectada no cumple con las especificaciones de diseño, es decir, no respeta las fronteras. Si este es el resultado, aunque sea únicamente para una de las frecuencias de diseño, el controlador es descartado como no válido. Este resultado se denominará como inviable.

Una vez que ha terminado la comprobación de la factibilidad de la caja de proyección con respecto a las especificaciones de diseño, se genera una tripleta de resultados con el siguiente contenido

$$(\mathbf{z}, z, flag_z)$$

donde **z** es el nodo analizado, z es el valor mínimo de la ganancia de alta frecuencia de ese nodo y $flag_z$ es el resultado de la comprobación de factibilidad de la caja, que puede tomar los valores de *viable*, *ambigua* o *inviable*. Si fuera este último caso, la tripleta se eliminaría.

Una vez creada la tripleta, se retornan los resultados.

3.1.2. Inicializar lista de nodos vivos

Corresponde al apartado (❷) del algoritmo. La lista de nodos vivos almacena el espacio de búsqueda del algoritmo. Son todos los nodos que, o bien son soluciones, o pueden contener soluciones. Esta lista se crea y se inicializa con el controlador propuesto por el usuario y, a partir de ahí, se va completando con los nodos generados por el propio algoritmo. Si esta lista se queda vacía antes de que se encuentre una solución, quiere decir que la estructura de controlador inicial propuesto no es adecuada para cumplir las restricciones de diseño.

3.1.3. Función solución

Corresponde con el apartado (❸) del algoritmo. Esta función se encarga de determinar cuando una caja **z** es solución para la planta propuesta. Para ello requiere por parte del usuario incluir un valor épsilon (ϵ) que indique la precisión que se requiere de la solución. Este valor indicará a partir de qué tamaño de caja proyectada (ancho y largo) se puede considerar un resultado válido. Como este valor es ajustable, el usuario puede decidir qué precisión requiere para solución.

De forma general, se considerará solución cuando la caja sea viable y menor que el valor ϵ introducido por el usuario. En un algoritmo de Ramificación y Poda clásico se debería explorar todo el espacio de búsqueda y encontrar todas las soluciones para poder quedarse con la mejor. Pero en este caso, por la propia ordenación de la lista de nodos vivos, la primera caja que sea solución será la óptima.

Por otro lado, existe una circunstancia que puede hacer que el algoritmo se bloquee y no encuentre nunca solución, y es una caja que, aunque cumpla las restricciones de tamaño ϵ, sea ambigua por quedarse siempre encima de una de las fronteras. Es decir, que por mucho que se biseccionen los nodos resultantes nunca serían viables, no encontrando nunca la solución y quedándose en un bloqueo infinito. Por este motivo, cuando una caja es ambigua y cumple con el tamaño exigido por ϵ, se extraen los valores de

los rangos de las variables del controlador que lo hacen viable, siguiendo lo mostrado en la imagen (3.3) y se termina el algoritmo.

3.1.4. Bisección de la caja

Una vez analizada la caja y, si esta es ambigua, el siguiente paso es el particionamiento de la misma, que corresponde con el apartado (❹) del algoritmo. Existen distintas formas de realizar esta operación. La regla general sería que particionar una caja X debe dar como resultado un número finito de subcajas X^i tal que se satisfaga la condición de $X = \cup X^i$.

En el particionamiento de una caja se toman dos decisiones principales. La primera de ellas es el número de subcajas a generar y la segunda es la dirección o coordenada (pueden ser una o varias) por la cual se quiere dividir la caja. En el caso de este algoritmo, se elige para particionar la variable de mayor tamaño del controlador y la división en dos subcajas denominándose de forma común como bisección.

3.1.5. Acotación y recorte del nodo

El siguiente paso corresponde con la etapa (❻). Una vez realizado el análisis del nodo para determinar su relación con las fronteras, si el nodo es inviable, se elimina, y si es viable se marca como solución. A continuación, en estos dos casos, se retorna al algoritmo principal y se selecciona un nuevo nodo. En el caso de que el nodo sea ambiguo, se continua con el procedimiento que se explica a continuación.

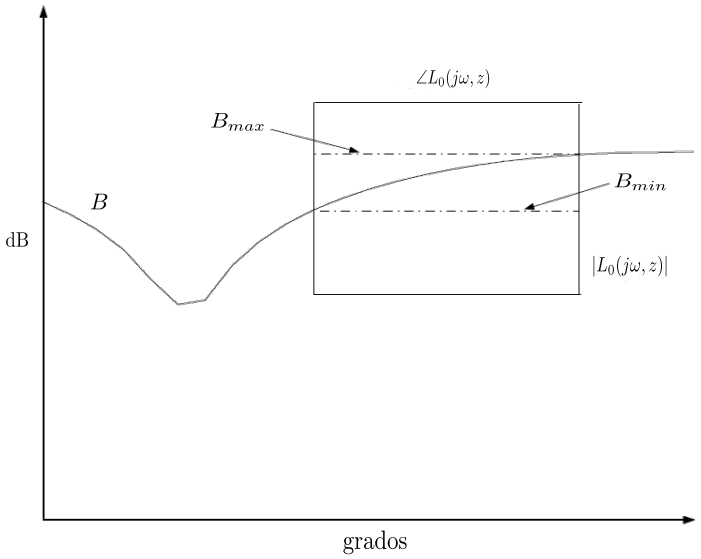

Figura 3.2: Análisis de una caja ambigua.

Cuando una caja es ambigua, significa que se ha situado sobre la frontera y no se puede determinar si esa caja contiene soluciones o viola las restricciones de diseño. Una frontera es sólo un cambio entre un espacio prohibido y un espacio permitido en el plano de Nichols, por tanto, una caja proyectada situada sobre ella, presumiblemente, tendrá una porción en la parte viable y otra porción en la parte inviable, solo que con las técnicas aplicadas hasta ahora no se pueden identificar estas porciones.

En la figura (3.2) se puede observar una caja ambigua proyectada sobre una frontera B en el plano de Nichols. Si la parte superior de la frontera fuera viable y la inferior inviable, el sitio indicado como B_{max} indicaría el punto donde la caja ambigua pasa a ser viable, y el sitio señalado por B_{min} donde pasaría a ser inviable. El objetivo de esta función es conseguir determinar, usando el valor B_{min}, qué parte de la ganancia de alta frecuencia es inviable con respecto a las fronteras para así poder eliminar esa porción cuando la caja sea ambigua. No es un procedimiento sencillo, dado que la proyección al plano de Nichols no es reversible. Esto quiere decir que no existe un procedimiento directo para relacionar el punto indicado por B_{min} en la figura (3.2) con ninguna de las variables del controlador. La variable B_{min} se define como el valor mínimo en magnitud de la frontera dentro de la caja proyectada y la variable B_{max}, como el valor máximo en magnitud de la frontera dentro de la caja proyectada.

El procedimiento que se plantea utiliza la monotonía que adquiere la caja proyectada en el plano de Nichols para que, de esta manera, y utilizando la implicación de cada variable del controlador en la forma de la caja proyectada, junto con la intersección con las fronteras, se pueda obtener qué parte de la ganancia de alta frecuencia es inviable. Es un método indirecto que se realiza utilizando el resto de variables del controlador, por tanto, aunque exista una parte inviable de la ganancia de alta frecuencia para la caja actual, el procedimiento no tiene por qué dar resultado, dado que dependerá de como sea la interacción con el resto de variables del controlador y de la forma que tenga la propia frontera.

En la figura (3.3) se puede observar una caja proyectada de un controlador que contiene una ganancia de alta frecuencia, un polo y un cero. Todas ellas variables con rangos. De esta manera, se puede atisbar cómo se implican en la forma de la caja los distintos términos del controlador. Por ejemplo, la ganancia de alta frecuencia incrementa la magnitud de la caja pero no modifica la fase. Por otro lado, los polos y los ceros se implican tanto en magnitud como en fase. De la imagen se puede deducir como cuanto mayor sea el valor del cero, la caja vira hacia la derecha y hacia arriba, en cambio, el polo se comporta de forma inversa, cuanto mayor sea su valor, la caja vira hacía la izquierda y hacia abajo.

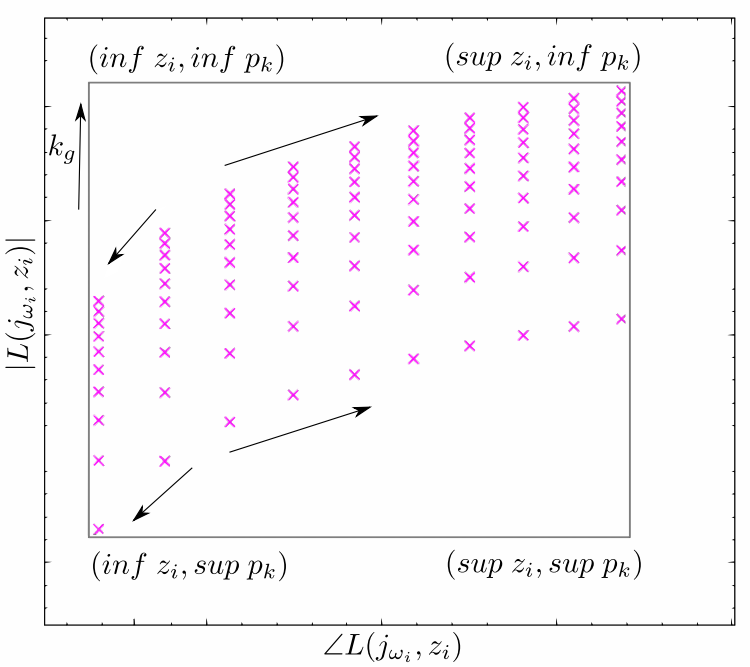

Figura 3.3: Implicación de las distintas variables del controlador en la caja proyectada.

Para poder realizar este cálculo, si la frontera define como viable el lado superior en el plano de Nichols, se ponen el resto de variables, polos y ceros, en el valor de su rango que maximiza el resultado en la caja de proyección. Tal y como se puede observar en la imagen (3.3) este sería el mínimo para los polos y el máximo para los ceros.

El algoritmo para realizar este cálculo (3.2) se muestra a continuación:

Algoritmo 3.2: Pseudocódigo del algoritmo que acota la k_{hf}

Entradas: Caja de parámetros $\mathbf{z} = \{k, z_k\}$, frecuencia de diseño ω, resultado de aplicar la extensión natural a intervalos a $|L_0(k, z_k, \omega)|$ y el valor B_{min} correspondiente con la frecuencia de diseño.

Salidas: Intervalo de la ganancia de alta frecuencia ($\boldsymbol{k}$).

$k_{min} = \underline{k}$

Calcular $L_{0_{kmin}} = \overline{|L_0(k_{min}, z_k, \omega)|}$

Calcular $k_B = \underline{k} + (B_{min} - L_{0_{kmin}})$

$k = k \cap [k_{min}, k_B]$

Retornar k

De esta forma, se obtendría el nuevo intervalo para la ganancia de alta frecuencia donde se ha eliminado su parte inviable, y dejando la parte res-

tante de la caja como ambigua.

Este procedimiento se realiza para cada una de las frecuencias de diseño. Si una parte del rango de la ganancia de alta frecuencia se determina inviable para alguna de dichas frecuencias, automáticamente se podría eliminar, dado que, si es inviable para una frecuencia de diseño, lo es para todas.

3.1.6. Ordenación de la lista NL y selección del nodo

Una vez biseccionada la caja y comprobada la factibilidad de los subnodos generados, para cada uno de ellos, si no son inviables, se incorporan a la lista de nodos vivos del algoritmo, que corresponde con el apartado (❼). La ordenación de la lista es un paso fundamental del procedimiento, dado que el nodo que quede en primera posición de la misma será el siguiente seleccionado. Los criterios utilizados para esta selección serán los que definan cómo ramifica el algoritmo y, por ende, cómo selecciona las ramas del árbol que se explorarán primero. El objetivo final deberá ser la priorización de las ramas más prometedoras para, de esta forma, encontrar antes la solución.

Para este algoritmo, las tripletas insertadas deben de estar ordenadas por la ganancia de alta frecuencia de forma creciente, es decir, las de menor valor en primer lugar. Como el objetivo del algoritmo es obtener el controlador con la menor ganancia de alta frecuencia que cumpla las restricciones de diseño, realizar esta ordenación asegura que se traten primero los nodos que tengan el valor más bajo para la misma, dejando para más tarde los nodos con valores superiores.

3.2. Algoritmo Nataraj y Kubal (NK)

Este algoritmo, propuesto en (Nataraj y Kubal [2006]), se basa en el algoritmo descrito en (3.1) y aporta varias modificaciones al algoritmo base con el fin de mejorar los tiempos de ejecución. Tal y como se describe en el algoritmo original, se proporciona un método basado en Ramificación y Poda utilizando aritmética intervalar. Esto se traduce en realizar una búsqueda global en el espacio de parámetros propuesto para el controlador. Este tipo de algoritmos son lentos por definición y se emplean diversas técnicas con el fin de mejorar su rendimiento.

En esta propuesta de algoritmo, se presentan dos mejoras con respecto al original. En primer lugar, se propone utilizar un método de búsqueda local para conseguir encontrar de forma rápida soluciones que permitan descartar nodos menos prometedores. Y en segundo lugar, propone extender la acotación y el recorte del nodo (3.1.5), realizado en el algoritmo inicial solo en la

ganancia de alta frecuencia, a los polos y ceros del controlador.

A continuación, se muestra el pseudocódigo (3.3) del algoritmo mostrando únicamente las diferencias con el algoritmo original:

Algoritmo 3.3: Pseudocódigo del algoritmo *Nataraj-Kubal (NK)*

Entradas: B_Ω y $\mathbf{x}$, caja inicial de búsqueda.

Salidas: $\mathbf{z}$, caja con los parámetros del controlador calculados en sus valores óptimos, o mensaje de que "No existe solución válida en x".

...

C—bis. Optimización Local

$z' \leftarrow LO(R_{z\Omega}, B_\Omega, \Omega, \mathbf{z})$

G. Acotación y recorte de los polos y ceros del controlador

Para $l = 1, 2 : \mathbf{v}^l \leftarrow QS(R_{v'\Omega}, B_\Omega, \Omega, \mathbf{v}^l)$

G—bis. ***Descartar*** si no mejora solución

Para $l = 1, 2 :$

Si $z' < \mathbf{v}^l$, descartar $\mathbf{v}^l$.

...

I. Ir a la etapa ***C***.

A continuación, se procede a explicar cada una de las partes del algoritmo.

3.2.1. Optimización local

Correspondiente a las secciones (❶) (donde se ejecuta la función) y (❸) (donde se aplica el resultado). Este procedimiento de búsqueda local pretende encontrar una solución de forma rápida con la finalidad de descartar los nodos menos prometedores y así eliminar parte del espacio de búsqueda. Este tipo de algoritmos no están pensados para encontrar la solución óptima, generalmente difícil y lenta de encontrar, sino que optan por encontrar de forma rápida una solución pseudóptima que sea útil para cierto cometido, en este caso, eliminar espacio de búsqueda.

3.2.2. Acotación y recorte de polos y ceros

Correspondiente a la sección (❷) del algoritmo. Sustituye la misma sección G del algoritmo base para extender la funcionalidad originalmente planteada. Basándose en los mismos principios de monotonía en el plano de Nichols e implicación de las variables del controlador en la caja proyectada explicados en (3.1.5). En este algoritmo se propone extender la misma funcionalidad a todos los polos y ceros del controlador. En la figura (3.4), se

puede observar un ejemplo donde la caja proyectada $R_{0.1z_1}$ puede ser acotada por este método, dado que, tiene una subcaja inviable en magnitud que se puede identificar. En cambio, las cajas proyectadas $R_{0.1z_2}$ y R_{100z_3}, no sería posible realizar ningún tipo de acotación, dado que su parte inviable esta en la fase y no en la magnitud, con respecto a la frontera.

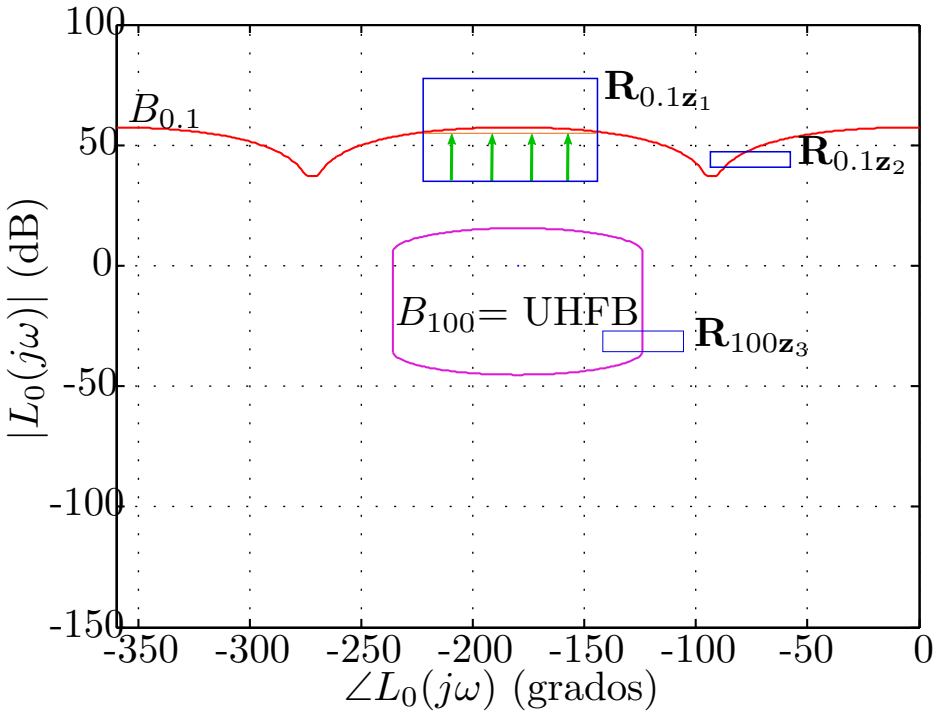

Figura 3.4: Acotación del subrango inviable de las variables del controlador.

El método se aplica para cada frecuencia de diseño y para cada variable del controlador. Para que se pueda aplicar, se necesitan unos datos de entrada, que se generan cuando se comparan las fronteras con cada caja proyectada en el plano de Nichols. De ésta se extraen los datos B_{min} y B_{max} que corresponderán al mínimo y máximo en magnitud de la frontera dentro de la caja proyectada, en la imagen (3.2) se puede observar.

Posteriormente, estos datos serán utilizados en las ecuaciones correspondientes (3.3, 3.4 y 3.5). Estas ecuaciones tomarán valores distintos dependiendo del semiplano que la frontera haga viable, es decir, si fuera una frontera abierta e hiciera su semiplano superior viable, se utilizarán unos datos; si fuera al revés, se utilizarían los datos contrarios.

Por ejemplo, si se tuviera una frontera abierta que hiciera inviable su semiplano inferior, entonces se utilizaría el valor B_{min}, dado que este método va orientado a estudiar la parte inviable de cada variable del controlador. Así mismo, tal y como se plantea en el algoritmo original, participan el resto

de variables del controlador, es decir, todas menos la que se está calculando, y estas deben tomar el valor del rango que maximice el resultado de dicho controlador. Esto se traduce en que la ganancia de alta frecuencia tomaría su valor máximo al igual que los ceros, pero los polos tomarían su valor mínimo. De esta forma, una vez conformada la ecuación se podría resolver.

Se debe tener en cuenta, que también existe el caso contrario, es decir, que una frontera haga inviable su semiplano superior. En este caso se tomaría el valor B_{max}, dado que la parte inviable estaría en el lado superior. A continuación, se tomarían los valores de las variables que minimizaran el resultado del controlador. Concretamente, sería el valor mínimo de la ganancia de alta frecuencia y de los ceros, y el valor máximo del rango de los polos. Como se puede observar, serían los valores contrarios.

A continuación, se muestran y desarrollan cómo serían las ecuaciones para despejar la ganancia de alta frecuencia, un polo y un cero del controlador. En el caso que se está estudiando, para ejemplificar las ecuaciones, se ha elegido el caso donde la frontera hace a su semiplano inferior inviable. En el caso contrario, habría que proceder de la forma que se ha explicado, utilizando los valores inversos, tanto en las variables del controlador, como los valores máximo o mínimo de la frontera dentro de la caja proyectada.

Seguidamente, se muestra cómo sería el desarrollo de la ecuación para la ganancia de alta frecuencia. En esencia, el mismo procedimiento que en (3.1.5) pero cambiando la forma de calcularse. Se parte de un lazo abierto ejemplo, como el siguiente:

$$L(j\omega) = C(j\omega, x) \cdot p_0 \tag{3.1}$$

donde p_0 es la planta nominal y $C(j\omega, x)$ es la estructura del controlador, que se define como

$$C(j\omega, x) = k\frac{\prod_{i=1}^{n_{rz}}(j\omega + z_i)}{\prod_{j=1}^{n_{rp}}(j\omega + p_j)} \tag{3.2}$$

con

$$x = (k, z_1, \dots z_{n_{rz}}, p_1, \dots p_{n_{rp}})$$

y donde $n_{rz} = \lfloor \frac{n-1}{2} \rfloor$ y $n_{rp} = \lceil \frac{n-1}{2} \rceil$

A continuación, para realizar el procedimiento, la magnitud de la ecuación del lazo abierto (3.1) se concreta para un ω determinado y se iguala con su correspondiente $|B_{min}|$. Los ceros del controlador se sustituyen por el valor superior de su correspondiente rango y los polo se sustituyen por el valor inferior de su correspondiente rango. A su vez, la ganancia de alta frecuencia

denominada como k, se sustituye por k' y se convierte en la variable incógnita a calcular de la ecuación. Por tanto, la ecuación a resolver quedaría de la siguiente forma:

$$|B_{min}| = \left| k' \frac{\prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i})}{\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)} p_0(j\omega) \right|$$

donde el número de polos y ceros es indeterminado y dependerá del problema concreto que se esté resolviendo.

La ecuación definitiva quedaría así:

$$k' = \frac{|B_{min}|}{\frac{\left|\prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i})\right|}{\left|\prod_{j=1}^{n_{rp}} (j\omega + \underline{p_j})\right|} |p_0(j\omega)|} \tag{3.3}$$

Donde el resultado k' sería el punto donde el rango de la ganancia de alta frecuencia pasa de inviable a ambiguo. Por tanto, todo el valor inferior se podría eliminar ($[\underline{k}, k']$), siendo el nuevo rango $k = [k', \overline{k}]$.

De forma similar, partiendo de la misma estructura inicial de lazo abierto, se pueden realizar las operaciones para calcular un cero del controlador. (3.1). El desarrollo es equivalente, el cambio reside en que uno de los ceros del controlador, el que se quiere calcular, se tratará de forma especial, dado que su valor no será sustituido por el máximo de su rango, sino que se convertirá en una variable a despejar. En cambio, el valor de la ganancia de alta frecuencia, de forma similar al resto de variables del controlador, se sustituirá por el valor de su rango que maximice el resultado, en este caso el valor máximo. Se usa z_1 a modo de ejemplo, pero podría ser cualquiera de los ceros del controlador.

$$|B_{min}| = \left| \frac{\overline{k}(j\omega + z_1') \prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i})}{\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)} p_0(j\omega) \right|$$

$$|B_{min}| = \frac{\overline{k}\,|(j\omega + z_1')|\,\left|\prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i})\right|}{\left|\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)\right|} |p_0(j\omega)|$$

$$|(j\omega + z_1')| = \frac{|B_{min}| \left|\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)\right|}{\overline{k}\,\left|\prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i})\right| |p_0(j\omega)|}$$

$$\sqrt{\omega^2 + z_1'^2} = \frac{|B_{min}| \left|\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)\right|}{\overline{k}\,\left|\prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i})\right| |p_0(j\omega)|}$$

$$\omega^2 + z_1'^2 = \left(\frac{|B_{min}| \left|\prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j}\right)\right|}{\overline{k}\,\left|\prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i})\right| |p_0(j\omega)|} \right)^2$$

$$z_1'^2 = \left(\frac{|B_{min}| \left| \prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|}{\overline{k} \left| \prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i}) \right| |p_0(j\omega)|} \right)^2 - \omega^2$$

La ecuación definitiva quedaría de la siguiente forma:

$$z_1' = \pm \sqrt{\left(\frac{|B_{min}| \left| \prod_{j=1}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|}{\overline{k} \left| \prod_{i=2}^{n_{rz}} (j\omega + \overline{z_i}) \right| |p_0(j\omega)|} \right)^2 - \omega^2} \tag{3.4}$$

Si el resultado z_1', estuviera dentro del rango, ya sea el positivo o el negativo, sería el punto del rango donde, para este cero, su valor pasaría de inviable a ambiguo. Este mismo procedimiento se repetiría para cada uno de los ceros del controlador de forma independiente. Por tanto, para todos los ceros del controlador, se aplicaría $z_n = [z_n', \overline{z_n}]$.

Para terminar, se muestra el procedimiento para calcular los polos del controlador. El procedimiento es muy similar al mostrado para los ceros, sólo cambiando el objetivo a despejar. Se parte de la misma ecuación para el lazo abierto (3.1) y se usa p_1 como ejemplo, pero se aplicaría a todos los polos.

$$|B_{min}| = \left| \frac{\overline{k} \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i})}{(j\omega + p_1') \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right)} p_0(j\omega) \right|$$

$$|B_{min}| = \frac{\overline{k} \left| \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i}) \right|}{|(j\omega + p_1')| \left| \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|} |p_0(j\omega)|$$

$$|(j\omega + p_1')| = \frac{\overline{k} \left| \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i}) \right|}{|B_{min}| \left| \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|} |p_0(j\omega)|$$

$$\sqrt{\omega^2 + p_1'^2} = \frac{\overline{k} \left| \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i}) \right|}{|B_{min}| \left| \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|} |p_0(j\omega)|$$

$$\omega^2 + p_1'^2 = \left(\frac{\overline{k} \left| \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i}) \right|}{|B_{min}| \left| \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|} |p_0(j\omega)| \right)^2$$

$$p_1'^2 = \left(\frac{\overline{k} \left| \prod_{i=1}^{n_{rz}} (j\omega + \overline{z_i}) \right|}{|B_{min}| \left| \prod_{j=2}^{n_{rp}} \left(j\omega + \underline{p_j} \right) \right|} |p_0(j\omega)| \right)^2 - \omega^2$$

La ecuación definitiva para despejar un polo del controlador:

$$p'_1 = \pm\sqrt{\left(\frac{\overline{k}\,|\prod_{i=1}^{n_{rz}}(j\omega+\overline{z_i})|}{|B_{min}|\left|\prod_{j=2}^{n_{rp}}\left(j\omega+\underline{p_j}\right)\right|}\,|p_0(j\omega)|\right)^2-\omega^2} \tag{3.5}$$

Si el resultado p'_1 estuviera dentro del rango, ya sea el positivo o el negativo, es el punto del rango de este polo en concreto donde pasa de inviable a ambiguo, por tanto, para todos los polos del controlador se aplicaría $p_n = [p'_n, \overline{p_n}]$.

El cálculo de estos valores se realiza para cada variable y para cada frecuencia de diseño, por tanto, para cada uno de los términos del controlador se dispondrá de varios valores. Uno por cada una de las mencionadas frecuencias de diseño. El resultado final será el valor que quite mayor proporción de rango de cada variable. Podría darse el caso, en las tres ecuaciones, de que no se obtuviera ningún resultado dentro del rango de la variable que se está calculando. Si esto sucede, es que no hay posibilidad de acotación para esa variable.

Este procedimiento se realiza para cada uno de los subnodos resultantes de la bisección del nodo original y justo antes de insertarse a la lista de nodos vivos. De esta forma, se reduce el espacio de parámetros de búsqueda y, al poder recortar la ganancia de alta frecuencia, que es la función objetivo, altera el posible orden en la lista de nodos vivos.

Una vez explicadas las ecuaciones matemáticas que se emplean para realizar la acotación y recorte del nodo, se procede a mostrar el pseudocódigo de la función que las aplica. En primer lugar recibe como parámetros de entrada los siguientes datos:

- $B_{nm\Omega}$ son los valores mínimos en magnitud de las fronteras dentro de cada caja proyectada[1].
- $\mathbf{\Omega}$ es el conjunto de frecuencias de diseño.
- $\mathbf{z}$ es la caja de parámetros.

Algoritmo 3.4: Pseudocódigo de la función *Quick Solution (QS)*

```
1  Entradas: B_nmΩ, Ω, z
2  Salidas: z, versión reducida de la caja original z.
3
4  Para ω ∈ Ω
5    Para j ∈ {1,...,n}
6      y ← z // z es copiado en y
7      Para λ ∈ {1,...,n} − {j} // En este para, y[λ] recibe el valor r de z[λ] que maximiza la
           contribución de x(λ) a |L_0(jω, x)|.
```

[1]Estos valores serán utilizados por las ecuaciones correspondientes para calcular los subrangos inviables en cada una de las frecuencias.

```
8       Si 1 ≤ λ ≤ 1 + n_rz + 2n_cz // ganancia (k) o cero
9         r ← sup(z[λ])
10      Sino                       // polo
11        r ← inf(z[λ])
12      Finsi
13      y ← subs(y, λ, [r, r]) // intervalo y[λ] ← r
14    Finpara
15    S ← mayor subintervalo de y[j] tal que subs(y, j, S) es inviable para ω
16    iz ← subs(z, j, S)
17    z ← z − iz
18  Finpara
19 Finpara
```

Capítulo 4

Diferentes estrategias para acelerar los algoritmos ALS Intervalares

En la sección (3) se han presentado una serie de algoritmos basados en una búsqueda global intervalar. Este tipo de algoritmos han supuesto un gran avance en el cálculo automático del lazo en QFT dado que son los primeros que aportan un método que obtiene como resultado la solución óptima. Dichos algoritmos, por sus propias características (1.3), tienen un coste computacional alto porque se basan en realizar una búsqueda exhaustiva en todo el espacio de parámetros propuesto. Por tanto, conforme el problema aumenta de tamaño, lo hace el espacio de búsqueda y el tiempo de ejecución se convierte en una curva exponencial, quedando limitados el uso de este tipo de algoritmos a tamaños donde el número de variables no sea muy grande.

En el algoritmo original, NT (3.1), se reconoce este problema y se proponen medidas que permiten recortar el espacio de búsqueda (3.1.5). Pero al centrarse sólo en una de las variables del controlador, que en este caso es la ganancia de alta frecuencia, su impacto es limitado.

Por otro lado, como se ha comentado en la sección (3.2), el algoritmo NK es una evolución del algoritmo NT que introduce nuevas características que mejoran el rendimiento. En primer lugar, se introduce una búsqueda local (3.2.1) con la finalidad de encontrar soluciones rápidas que permitan realizar podas en las ramas del árbol. Y en segundo lugar, se extiende la acotación de la parte inviable de la ganancia de alta frecuencia a todas las variables del controlador (polos y ceros) (3.2.2) con el objetivo de ampliar el impacto de esta característica.

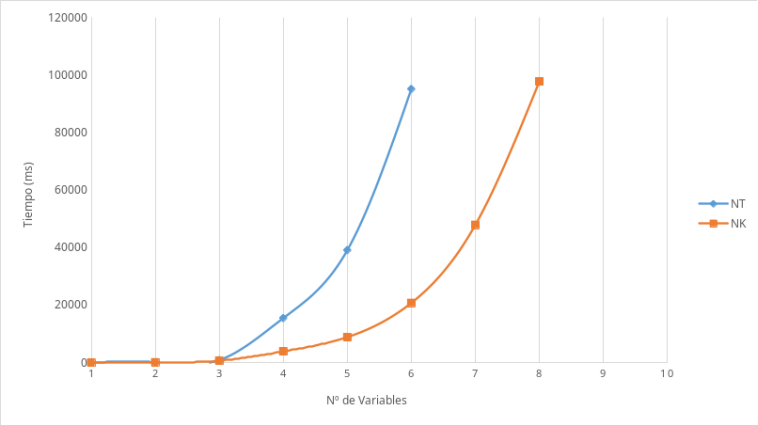

Figura 4.1: Tiempos de ejecución para los algoritmos NT y NK.

En la figura (4.1) se pueden observar los tiempos obtenidos[1] para los algoritmos NT y NK con el ejemplo número 2 del *Toolbox de QFT de Matlab* (Borghesani et al. [2003]). Se puede comprobar como a partir de cierto número de variables, los dos algoritmos disparan su tiempo de ejecución hasta convertirse en una curva exponencial, a pesar de las mejoras aportadas por el algoritmo NK con respecto al NT. Esta es un cuestión importante, dado que limita mucho a qué tipo de problemas pueden hacer frente estos procedimientos, dado que se tienen que ceñir a un número limitado de variables en el controlador.

Por ello, el objetivo de esta tesis ha sido realizar nuevas propuestas con la idea fundamental de crear un nuevo algoritmo que reduzca el tiempo de ejecución obtenido por las soluciones vigentes hasta el momento. Algunas de estas estrategias expanden varias de las funcionalidades ya implementadas en los algoritmos NT y NK, en cambio, otras proponen ideas novedosas que actúan en múltiples ámbitos hasta crear un algoritmo completamente nuevo (Martínez-Forte y Cervera [2021]). A continuación, se explican en detalle todas estas nuevas características.

4.1. Acotación del espacio de búsqueda

Las estrategias que se van a exponer en esta sección desarrollan nuevos procedimientos para acotar el espacio de búsqueda del algoritmo y, de esta forma, reducir el tiempo de ejecución. Todas ellas, aunque tengan características propias, tienen una forma similar de proceder y, por ello, se agrupan en esta sección.

[1] Para más detalles consultar el capítulo (6)

4.1.1. Información subcaja viable

El algoritmo original NT (3.1), ampliado en el algoritmo NK (3.2), se centra en explorar cómo localizar, y posteriormente eliminar, la subcaja inviable de las distintas variables del controlador. Esta propuesta es muy interesante para poder eliminar toda esta porción del rango de las distintas variables que no contienen soluciones. Pero esta idea tiene la problemática de que sólo se centra en reducir el espacio de búsqueda que incumple las restricciones de diseño, dejando intacto el resto del espacio de parámetros. De esta forma, aunque se consigue reducir el espacio de búsqueda, la implicación de esta reducción será limitada en el tiempo de ejecución, dado que, el resto del espacio del parámetros seguirá intacto. Esto se puede observar en la figura (4.1) donde, aunque se marcan diferencias entre el algoritmo NT y NK, el tamaño del problema solo puede crecer unas pocas variables antes de que la curva de tiempo de ejecución se vuelva exponencial.

Tal y como se indica en (3.1.1), la caja de parámetros, una vez comprobada su factibilidad, puede estar en tres situaciones:

- La primera de ellas es cuando la caja cumple las restricciones de diseño y todos sus elementos son soluciones, llamada subcaja *viable*.
- Una segunda situación es cuando la caja se denomina *ambigua*. Esto quiere decir que se sitúa sobre la frontera y es imposible determinar si contiene soluciones o, por el contrario, no cumple con las restricciones de diseño.
- Para terminar, estaría la subcaja *inviable*, que es eliminada por no cumplir con las restricciones de diseño.

La estrategia planteada en esta sección, para reducir el espacio de búsqueda, consiste en no solo usar información sobre violación de restricciones (subcajas inviables), como ya hacen los algoritmos NT y NK, sino también implicar información procedente de las subcajas viables. Por ello, el objetivo de esta nueva estrategia será encontrar las partes que contienen soluciones de cada una de las variables del controlador, de forma semejante a como se realiza para las partes inviables. En la figura (4.2), se puede observar como para la caja proyectada $R_{0.1z_1}$, se puede realizar la acotación del espacio viable utilizando la magnitud. En cambio, paras las cajas proyectadas $R_{0.1z_2}$ y R_{100z_3}, no sería posible, dado que su subcajas viables están en fase, no en magnitud, con respecto a la frontera.

Pero detectar y trabajar con las subcajas viables tiene algunas particularidades distintas respecto a trabajar con las subcajas inviables. El primer cambio fundamental es que, cuando se trabaja con el objetivo de detectar subcajas inviables, como estas no cumplen con las restricciones de diseño,

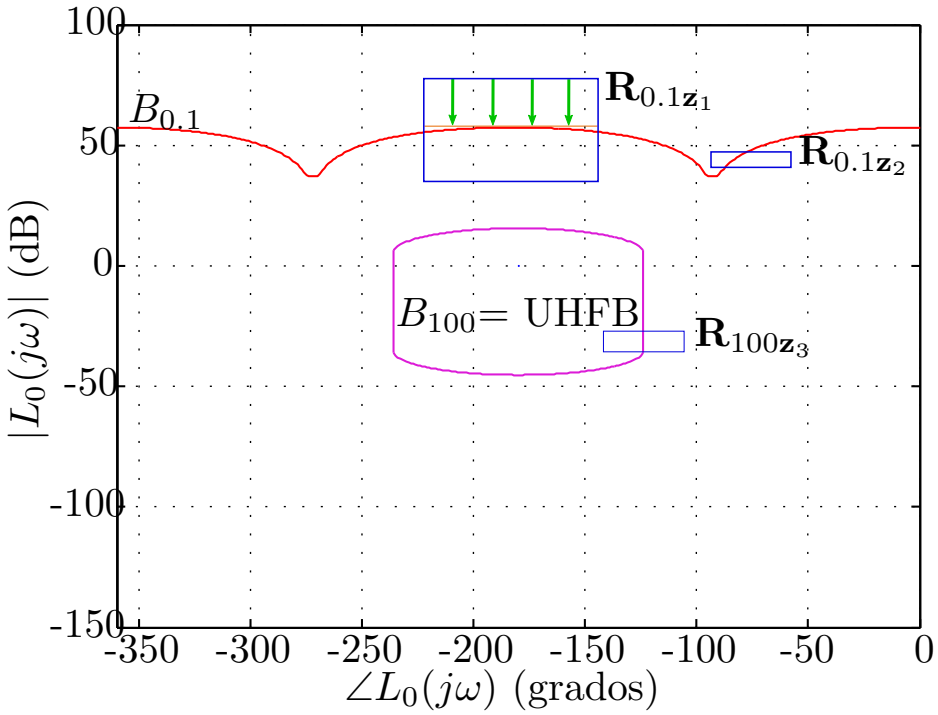

Figura 4.2: Acotación del subrango viable de las variables del controlador utilizando la magnitud.

pueden ser descartadas de forma automática porque no contienen soluciones. En cambio, cuando se detecta una subcaja viable, lo que indica es que se ha encontrado una solución, por tanto, no solo no se debe descartar, sino que se puede utilizar para acotar el espacio de búsqueda allí donde los nodos sean menos prometedores.

Una segunda diferencia importante radica en que, cuando se detecta una de estas situaciones en una subcaja inviable para una frecuencia de diseño concreta, automáticamente se puede descartar para todas las frecuencias de diseño. En cambio, para que una subcaja pueda ser marcada como viable, debe ser solución para todas las frecuencias de diseño. Por ello, cuando en una variable se estudia qué partes del rango son viables para cada una de las frecuencias de diseño, el resultado que satisfaga las restricciones para todas las frecuencias será la intersección de cada una de las soluciones individuales.

Los distintos subrangos detectados como viables para cada variable, irán desde un extremo del rango hasta el punto donde el rango se convierte en ambiguo. Dichos subrangos estarán solapados entre ellos en el extremo del rango pero, para cada frecuencia de diseño, terminarán en un punto distinto. Esto quiere decir que, cuando se realice la intersección de dichos subrangos, se seleccionará el que sea común a todas las frecuencias, es decir, el que menor tamaño tenga, por este motivo, se puede hablar de inclusión de rangos. El

subrango que sea viable solo para una frecuencia de diseño, estará incluido en el resto de subrangos. El que sea viable para dos frecuencias, estará incluido en el resto de subrangos menos en uno, y así consecutivamente.

El concepto base para poder detectar la subcaja viable es el mismo que el explicado en la sección (3.2.2). Se basa en la idea equivalente de implicación de los polos, ceros y ganancia de alta frecuencia en la caja proyectada. Por ello, las ecuaciones empleadas son las mismas que las utilizadas en el mencionado procedimiento. Dichas ecuaciones para cada tipo de elemento del controlador son las siguientes: para la ganancia de alta frecuencia (3.3), para los ceros (3.4) y para los polos (3.5); pero se utiliza el valor opuesto de las variables del controlador, es decir, si la frontera hace viable su parte superior del plano de Nichols, las variables de las ecuaciones se sitúan en el valor de su rango que maximiza el resultado de la función de transferencia del controlador. Justo lo contrario que lo utilizado para detectar la subcaja inviable. Si la frontera hiciera viable su parte inferior en el plano de Nichols serían los valores inversos y las variables se situarían en el valor del rango que minimiza el resultado del controlador. Así como, de una forma equiparable, en vez de utilizar el valor mínimo de la frontera dentro de la caja proyectada en el plano de Nichols, se utiliza el valor máximo, denominado como B_{max}.

Para ejemplificar las ecuaciones, se empleará el mismo caso que el utilizado en la explicación del algoritmo NK (3.2.2). En dicho ejemplo, la frontera es viable por parte superior e inviable en su parte inferior. No se entrará en el detalle del desarrollo de las ecuaciones, dado que son equivalente a las ya explicadas en la sección referida.

- Para la ganancia de alta frecuencia:

$$k' = \frac{|B_{min}|}{\frac{\left|\prod_{i=1}^{n_{rz}}(j\omega+\underline{z_i})\right|}{\left|\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right|}\,|p_0(\omega)|} \tag{4.1}$$

- Para un cero del controlador:

$$z_1' = \pm\sqrt{\left(\frac{|B_{min}|\left|\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right|}{\underline{k}\left|\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})\right||p_0(\omega)|}\right)^2-\omega^2} \tag{4.2}$$

- Para un polo del controlador:

$$p_1' = \pm\sqrt{\left(\frac{\underline{k}\left|\prod_{i=1}^{n_{rz}}(j\omega+\underline{z_i})\right|}{|B_{min}|\left|\prod_{j=2}^{n_{rp}}(j\omega+\overline{p_j})\right|}|p_0(\omega)|\right)^2-\omega^2} \tag{4.3}$$

A continuación, una vez aplicadas las ecuaciones correspondientes para cada una de las variables y para cada una de las frecuencias de diseño, se determinará como viable la porción del rango de cada variable que sea viable para todas las frecuencias de diseño. Estos subrangos serán detraídos de la caja de parámetros, en la cual solo quedarán las partes ambiguas de las variables.

Este procedimiento no siempre obtiene solución, dado que no siempre será posible detectar la subcaja viable. Dependerá de las circunstancias concretas que se estén tratando. Por ejemplo, podría darse la situación de que la forma de la caja proyectada en el plano de Nichols, junto con la forma que tenga la frontera en los lugares en los que intersecan, diera como resultado que no es posible detectar una subcaja viable. Por tanto, aunque se pudieran calcular las ecuaciones correspondientes, éstas obtendrían un valor fuera del rango de la variable, lo que quiere decir que no existe ninguna porción de ese rango que sea viable con la información disponible en ese momento.

4.1.2. Información de la fase

Una caja proyectada en el plano de Nichols es básicamente un rectángulo donde el eje de abscisas representa la fase en grados, de 0 a -360, y el eje de ordenadas representa la magnitud en decibelios. En secciones anteriores se han explicado varios procedimientos para realizar acotaciones en el espacio de parámetros de la caja y así reducir el tiempo de ejecución (3.1, 3.2 y 4.1.1). Todos estos procedimientos tienen la característica común de que trabajan con la magnitud del plano de Nichols para realizar las operaciones, ya sea para la parte viable como para la inviable. Pero hasta ahora no se ha realizado ninguna propuesta que utilice la fase para realizar, de una forma similar a cuando se utiliza la magnitud, la acotación en el espacio de parámetros.

Esta propuesta se basa en trasladar lo aplicado para la magnitud y realizar un procedimiento similar utilizando la fase. Dicho procedimiento puede aplicarse tanto para detectar subcajas viables como subcajas inviables, de forma similar a lo explicado en la sección anterior. En primer lugar, es necesario disponer del valor máximo y mínimo, en fase, de la frontera dentro de la caja proyectada en el plano de Nichols. A este valor se le denominará C_{min} y C_{max}, según corresponda al mínimo y al máximo.

A continuación, para cada una de las variables del controlador, polos y ceros (excluyendo la ganancia de alta frecuencia, dado que no tiene componente de fase), y cada una de las frecuencias de diseño, se calcula la parte viable e inviable de cada una de ellas en fase. El procedimiento es similar al ya explicado para las partes viable e inviable de la magnitud, pero en vez de despejar utilizando el módulo se despeja utilizando el ángulo. Para continuar, se

desarrollan las funciones, siendo estas dependientes de qué porción de plano haga viable o inviable la caja según el tipo de frontera, de forma similar a como se actúa cuando se utiliza la magnitud. En el caso de la magnitud, que corresponde al eje de ordenadas, se caracteriza como lado superior o inferior, donde el lado superior podía ser viable y el inferior inviable, o viceversa. En el caso de la fase, que corresponde al eje de abscisas, se caracteriza como lado izquierdo y derecho.

En la figura (4.3), se puede observar como para las cajas proyectadas $R_{0.1z_2}$ y R_{100z_3} se podría realizar la acotación de la subcaja, tanto viable como inviable, dado que las dos están en fase con respecto a las fronteras. En cambio, para la caja proyectada $R_{0.1z_1}$, utilizando esta estrategia, no se podría realizar ninguna acotación, dado que la subcaja viable está en magnitud con respecto a la frontera.

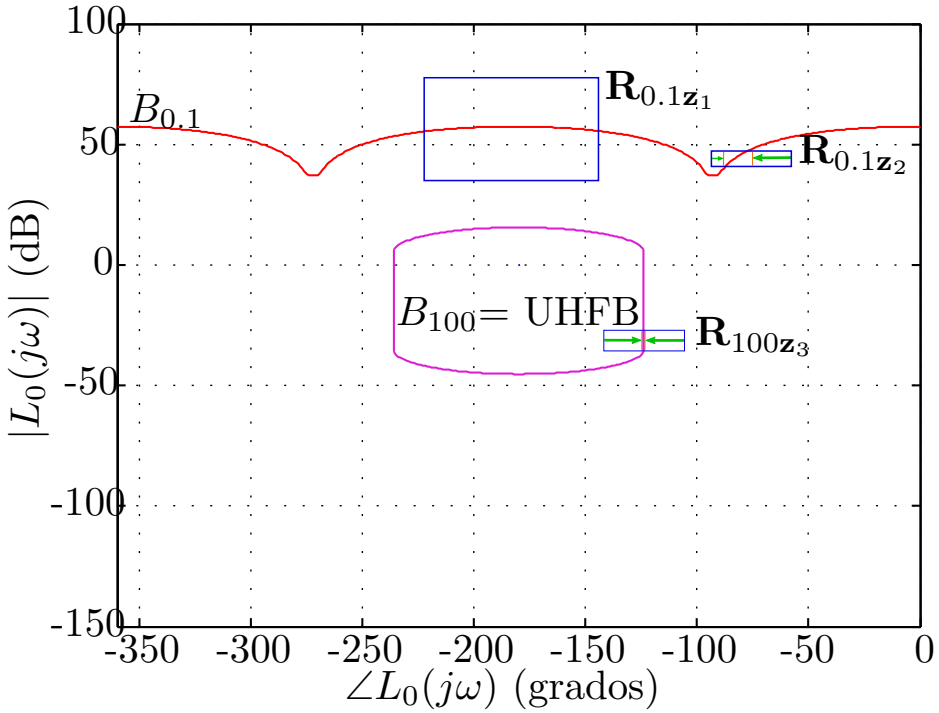

Figura 4.3: Acotación del subrango viable e inviable de las variables del controlador utilizando la fase.

De una forma similar a como se calcula utilizando la magnitud, para cada una de las variables, se puede calcular para su lado superior y su lado inferior del rango, dónde se situaría el punto donde la variable pasa a ser de inviable a ambigua, y de ambigua a viable. Como ya se ha explicado, el lado que corresponda a viable o a inviable dependerá de qué porción del plano de Nichols haga viable o inviable la frontera.

A continuación, se muestran las ecuaciones para realizar la mencionada acotación del espacio de parámetros utilizando la fase, tanto para ceros como para polos. Estas ecuaciones se van a ejemplificar empleando el mismo caso utilizado en (3.2.2), donde la frontera hace viable la porción derecha del plano e inviable su porción izquierda. Se parte del lazo abierto siguiente:

$$L(j\omega) = C(s,x) \cdot p_0 \tag{4.4}$$

donde p_0 es la planta nominal y $C(s,x)$ es la estructura del controlador, que se define como

$$C(s,x) = k\frac{\prod_{i=1}^{n_{rz}}(s+z_i)}{\prod_{j=1}^{n_{rp}}(s+p_j)} \tag{4.5}$$

con

$$x = (k, z_1, \dots z_{n_{rz}}, p_1, \dots p_{n_{rp}})$$

y donde $n_{rz} = \lfloor \frac{n-1}{2} \rfloor$ y $n_{rp} = \lceil \frac{n-1}{2} \rceil$.

A continuación, en la ecuación del lazo abierto los ceros y los polos tomarán el valor que minimice el resultado del controlador. Esto se traduce en que los ceros tomarán el valor mínimo de su rango y los polos el máximo. El siguiente paso es calcular el ángulo de todo el lazo abierto e igualarlo con C_{min}. Para terminar, z_i', es el cero que se quiere calcular.

$$C_{min} = \angle\left(\frac{\underline{k}(j\omega + z_1')\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})}{\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})}p_0(\omega)\right)$$

Las operaciones se realizan en coordenadas polares, por tanto, las multiplicaciones se transforman en sumas y las divisiones en restas. Así mismo, la ganancia de alta frecuencia desaparece al no tener componente de fase.

$$C_{min} = \angle(j\omega + z_1') + \angle\left(\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right)$$

$$C_{min} = atan\left(\frac{\omega}{z_1'}\right) + \angle\left(\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right)$$

$$-atan\left(\frac{\omega}{z_1'}\right) = -C_{min} + \angle\left(\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right)$$

$$\frac{\omega}{z_1'} = tan\left(C_{min} - \angle\left(\prod_{i=2}^{n_{rz}}(j\omega+\underline{z_i})\right) - \angle(p_0(\omega)) + \angle\left(\prod_{j=1}^{n_{rp}}(j\omega+\overline{p_j})\right)\right)$$

La ecuación definitiva sería la siguiente:

$$z_1' = \frac{\omega}{tan\left(C_{min} - \angle\left(\prod_{i=2}^{n_{rz}}(j\omega + \underline{z_i})\right) - \angle(p_0(\omega)) + \angle\left(\prod_{j=1}^{n_{rp}}(j\omega + \overline{p_j})\right)\right)} \tag{4.6}$$

De esta forma se podría despejar cada uno de los ceros del controlador.

A continuación, se replica el mismo proceso para despejar un polo:

$$C_{min} = \angle\left(\frac{\underline{k}\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})}{(j\omega + p_1)\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})}p_0(\omega)\right)$$

$$C_{min} = \angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(j\omega + p_1'\right) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right)$$

$$C_{min} = -atan\left(\frac{\omega}{p_1'}\right) + \angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right)$$

$$atan\left(\frac{\omega}{p_1'}\right) = \angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right) - C_{min}$$

$$\frac{\omega}{p_1'} = tan\left(\angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right) - C_{min}\right)$$

$$\frac{1}{p_1'} = \frac{tan\left(\angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right) - C_{min}\right)}{\omega}$$

$$p_1' = \frac{\omega}{tan\left(\angle\left(\prod_{i=1}^{n_{rz}}(j\omega + \underline{z_i})\right) + \angle(p_0(\omega)) - \angle\left(\prod_{j=2}^{n_{rp}}(j\omega + \overline{p_j})\right) - C_{min}\right)} \tag{4.7}$$

Este procedimiento se aplicaría para cada uno de los polos del controlador y para cada frecuencia de diseño. Al igual que en el caso de la magnitud, no tiene por qué obtenerse un resultado válido. Como se ha explicado anteriormente, si la variable que se está despejando no contiene soluciones, la ecuación obtendrá un resultado fuera del rango de dicha variable.

4.1.3. Estructura de la acotación del espacio de búsqueda

En este apartado se organizan las estrategias relacionadas con la acotación del espacio de parámetros, con el objetivo de aportar claridad dado que, en capítulos posteriores, serán utilizadas de forma conjunta. Las estrategias concernientes a esta sección se puede dividir en cuatro fundamentales:

- Acotación de la parte inviable del espacio de búsqueda utilizando la magnitud (3.2.2). Estrategia propuesta en el algoritmo NK y que fija las bases para poder realizar este tipo de acotaciones en todas las variables del controlador.
- Acotación de la parte inviable del espacio de búsqueda utilizando la fase (4.1.2). Este apartado se refiere a cuando se aplica el procedimiento centrado en la parte inviable del espacio de parámetros.
- Acotación de la parte viable del espacio de búsqueda utilizando la magnitud (4.1.1). Este apartado se centra en aplicar la acotación a la parte viable del espacio de parámetros.
- Acotación de la parte viable del espacio de búsqueda utilizando la fase (4.1.2 y 4.1.1). En este caso se aplican dos de las estrategias presentadas en este capítulo y que permiten realizar la acotación del espacio de búsqueda combinando las estrategias de acotación utilizando la fase y la parte viable de las variables del controlador.

De esta forma, se clasifican las distintas estrategias de acotación que serán utilizadas en capítulos posteriores, tanto en el capítulo (5) donde se explica el algoritmo propuesto, como en el capítulo (6), donde se desarrollan los casos prácticos en detalle para todas la estrategias.

4.2. Bisección del nodo

Tal y como se explica en la sección (3.1.4), la subdivisión del nodo es una parte fundamental del algoritmo, dado que es el mecanismo por el cual se ramifica el espacio de búsqueda. Realizar una ramificación adecuada ayudará a expandir las ramas más prometedoras del algoritmo en primer lugar. En el algoritmo original (NK y NT) esta bisección se realiza por la variable del controlador que tenga mayor diferencia entre los extremos de su rango. Como idea general es adecuada, pero la intersección entre las fronteras y las cajas de proyección puede tener una casuística muy variada, y ceñirse siempre a la misma forma de proceder, limita las posibilidades del algoritmo.

En esta sección se van a analizar diferentes formas de realizar la bisección. Algunas tendrán un propósito más general de reducir el tamaño de los

subnodos resultantes de forma homogénea, otras harán incidencia en reducir el tamaño en fase o en magnitud. Y para terminar, se plantea un sistema que utiliza la información disponible del nodo hasta ese momento para decidir por donde realizar la bisección.

Estas formas de bisección se utilizarán conforme indique la estrategia *Etapas de ejecución e historia del nodo* (4.4) según convenga en cada momento de la ejecución del algoritmo.

4.2.1. Bisección utilizando el tamaño en área

Esta forma de realizar la bisección es la más genérica de todas. Dado que, para seleccionar la variable por la cual se realiza la bisección, se usa como criterio para elegir la que más aporta al área de la caja de proyección en el plano de Nichols. De esta forma, la bisección siempre se hará por la variable que reduce más el tamaño de la proyección de los subnodos generados. Siendo ésta una opción que aporta un buen resultado de forma general, pero que no tiene en cuenta las diferentes casuísticas que puedan afectar a la intersección de la caja proyectada con las fronteras.

4.2.2. Bisección utilizando el tamaño en magnitud

En esta segunda forma de realizar la bisección se optará por seleccionar la variable que más aporte a la caja proyectada en el plano de Nichols en magnitud. De esta forma, las proyecciones de los dos subnodos generados verán reducido su tamaño especialmente en esa dirección. Esta opción podrá aplicarse cuando interese reducir el tamaño en magnitud de la caja proyectada, que puede ser en diversas situaciones. Por ejemplo, cuando por la forma en la que intersecan las fronteras con las cajas proyectadas, reducir la magnitud favorece para aplicar acotaciones del espacio de búsqueda.

4.2.3. Bisección utilizando el tamaño en fase

Opción muy similar a la explicada en la sección anterior (4.2.2),que se focaliza en seleccionar la variable que más favorece la reducción de la fase de la caja proyectada en los subnodos generados. Al igual que en el caso anterior, puede ser beneficioso utilizarla cuando reducir la fase favorezca la aplicación de otras técnicas para mejorar el rendimiento del algoritmo.

4.2.4. Bisección en árbol

Cuando se analiza la subcaja viable de la caja de proyección, para poder identificar qué parte de dicha caja contiene soluciones, tal y como se ha explicado en la sección (4.1.1), el resultado será la intersección de todos los

valores calculados de cada frecuencia de diseño. Dado que, para que una subcaja pueda considerarse solución, lo debe ser para todas las frecuencias. Por tanto, el resto de valores que son solución solo para unas frecuencias concretas, y que permitirían identificar una parte mayor del espacio de parámetros, no se utilizan en el mencionado procedimiento.

Los valores calculados para cada una de las frecuencias de diseño se pueden ordenar de la siguiente forma. La subcaja factible para todas las frecuencias de diseño es la que se utiliza para identificar las soluciones en (4.1.1). La siguiente subcaja lo será para todas las frecuencias menos una, el siguiente para todas menos dos, y así sucesivamente. Conforme la subcaja seleccionada sea viable para menos frecuencias, mayor porcentaje del rango de las variables podrá ser identificado como viable para esas frecuencias concretas para las que sea viable.

Estos puntos calculados para cada una de las variables del controlador y para cada una de las frecuencias podrían utilizarse para realizar la bisección. De esta forma, de los dos subnodos generados, uno de ellos sería ambiguo para todas las frecuencias de diseño y el otro sería viable para algunas de ellas, dependiendo del punto que se eligiera. A partir de esta idea se pueden plantear diferentes formas de proceder, por ejemplo, realizar múltiples bisecciones con la finalidad de obtener varias cajas viables para una o varias frecuencias de diseño. Estas cajas se marcarían como viables para esas frecuencias concretas y ya no sería necesario realizar operaciones sobre ellas, ahorrando tiempo de cómputo posterior.

Por otro lado, también permitiría priorizar las cajas que son viables para frecuencias de diseño concretas, concentrando las cajas ambiguas en otras frecuencias que permitan realizar acotaciones en el espacio de parámetros de forma más sencilla o con mayor efectividad.

En general, esta opción de bisección permite decidir estrategias con más información y posibilidades que las formas de bisección planteadas anteriormente y así tomar decisiones en distintas fases del algoritmo según convenga.

Para ejemplificar el funcionamiento de esta estrategia se parte de una caja $\mathbf{z}$ con un controlador que tiene dos variables, la ganancia de alta frecuencia y un polo. Ya se les ha aplicado la estrategia de (4.1.1), por tanto, las subcajas viables ya han sido identificadas y no están presentes, la subcaja restante es ambigua y en la caja actual solo quedan la subcajas que son viables para algunas frecuencias, pero no para todas.

Se parte del siguiente controlador ejemplo:

$$C(s) = \frac{k}{(s+a)},$$

donde

$k \in [1, 10]$

$a \in [20, 100]$

Y las frecuencias de diseño son $\Omega = \{0.1, 0.5, 1\}\ rad/s$.

Se ha detectado que para cada frecuencia de diseño las siguientes subcajas viables:

- Para la frecuencia 0.1 las subcajas factibles son $k'_1 \in [7, 10]$ y $a'_1 \in [80, 100]$
- Para la frecuencia 0.5 las subcajas factibles son $k'_2 \in [9, 10]$ y $a'_2 \in [60, 100]$
- Para la frecuencia 1 no existe ninguna subcaja viable.

El procedimiento calcula para que variable y para que frecuencia de diseño la subcaja factible es porcentualmente mayor. Por ejemplo:

- Para la variable k y la frecuencia 0.1 la subcaja factible es el 33.33 % de la caja.
- Para la variable a y la frecuencia 0.5 la subcaja factible es el 40 % de la caja.

Por tanto, el procedimiento decidirá que la bisección se realiza por la variable a y el punto 60, quedando como resultado lo siguiente:

- $\mathbf{z_1} = (k, a'_1)$ donde $k = [1, \ldots, 10]$ y $a'_1 = [20, \ldots, 60]$.
- $\mathbf{z_2} = (k, a'_2)$ donde $k = [1, \ldots, 10]$ y $a'_2 = [60, \ldots, 100]$.

donde se cumple que

$$\mathbf{z} = \mathbf{z_1} \cup \mathbf{z_2}$$

y obteniendo como resultado que:

- $\mathbf{z_1}$ es una caja ambigua que deberá se analizada posteriormente.
- $\mathbf{z_2}$ es una caja ambigua, pero es viable para la frecuencia 0.5. Esta información podrá ser usada tanto en esta caja como en sus descendientes.

De esta forma se consigue utilizar la información disponible del nodo para realizar la bisección del nodo y generar sus hijos.

4.3. Búsqueda mejor ganancia de alta frecuencia

El procedimiento explicado en (4.1.1) tiene como objetivo encontrar los rangos de las variables del controlador que sean viables para cada frecuencia de diseño. En este procedimiento, en el caso de que la parte viable de una frontera esté en el lado superior en el plano de Nichols, las variables del controlador se sitúan en el valor que minimiza el resultado del mismo. En este caso, los polos en su valor máximo y los ceros, junto con la ganancia de alta frecuencia, en su valor mínimo. De esta forma, al situarse los polos, ceros y ganancia de alta frecuencia en el valor que minimiza el resultado del controlador, mientras se busca la subcaja viable, el valor que se obtenga para la ganancia de alta frecuencia será viable independientemente de dónde estén situadas el resto de variables. Es decir, que como el resto de variables están en la peor situación posible para obtener soluciones, las zonas viables que se obtengan serán viables para todo el rango del resto de variables. Si el resto de variables del controlador movieran sus valores a otras zonas de sus rangos, la ganancia de alta frecuencia detectada como viable lo seguiría siendo. En todo caso, podría mejorar el resultado pero nunca empeorarlo.

Esta forma de proceder asegura que el rango detectado como viable lo sea siempre independientemente del valor que tomen el resto de variables del controlador. Esta opción es conservadora, dado que, este procedimiento tiene por objetivo encontrar las zonas viables de todas las variables del controlador y así acotar el espacio de búsqueda a las zonas ambiguas.

Tal y como se explicó en la sección (1.3), una de las características principales de este tipo de algoritmos es podar las ramas menos prometedoras del espacio de búsqueda. La forma principal de realizar esta poda es encontrar soluciones que permitan descartar todas las ramas que no puedan mejorar dichas soluciones. De esta forma, se pueden eliminar estas ramas evitando expandirlas y sin tener que realizar la búsqueda exhaustiva por ellas, reduciendo así el tiempo de ejecución.

El procedimiento explicado en (4.1.1) encuentra soluciones para cada una de las variables, también la ganancia de alta frecuencia, y se puede utilizar con la finalidad de realizar estas podas. Pero trata por igual a todas las variables del controlador, sin tener en cuenta que la función objetivo del algoritmo es encontrar la ganancia de alta frecuencia más baja que cumpla las restricciones de diseño. Por tanto, dada esta función objetivo, en esta estrategia se plantea una forma de proceder especial para esta variable concreta.

El procedimiento funciona utilizando la fórmula (4.1) para despejar la ganancia de alta frecuencia, pero cambiando el valor de los polos al mínimo y el

de los ceros a su máximo. Se puede consultar la ecuación definitiva en (4.8). De esta forma, el valor que se podría obtener como punto donde el rango de la ganancia de alta frecuencia pasa de ser ambiguo a ser viable, puede ser potencialmente mucho menor. Esta solución sólo sería un punto concreto de cada variable del controlador. Por tanto, otros valores de dentro del rango de los polos y los ceros podrían hacer que el resultado obtenido no fuera solución. Pero esto no es problema porque cumple el objetivo de encontrar la mínima ganancia de alta frecuencia para la que se encuentra una solución válida, independientemente del valor que tuvieran el resto de variables.

$$k' = \frac{|B_{min}|}{\frac{\left|\prod_{i=1}^{n_{rz}}(j\omega+\overline{z_i})\right|}{\left|\prod_{j=1}^{n_{rp}}(j\omega+\underline{p_j})\right|}\,|p_0(\omega)|} \tag{4.8}$$

En la figura (4.4) se puede observar un ejemplo proyectado en el plano de Nichols para el siguiente controlador:

$$C(s) = \frac{k}{s+a}$$

donde

- $k = [1 \cdots 10]$
- $a = [1 \cdots 10]$

donde la línea roja bordeada por B_{min} y B_{max} es la frontera que interseca con la caja de proyección. Ademas, se puede observar para las variables del controlador, qué valores tomarían cada una de ellas a lo largo de la imagen. De esta forma, se puede examinar la implicación del polo y de la ganancia de alta frecuencia en la caja proyectada. Si se quisiera calcular la parte viable de la ganancia de alta frecuencia con el método propuesto en (4.1.1) se utilizaría el valor del polo en su máximo, $a = 10$, dado que es cuando se minimiza el resultado del controlador. Como se puede observar en la propia figura por la curva introducida por el polo (línea discontinua), en el lado derecho donde $a = 10$, el valor de la frontera coincidente con B_{max}, es superior al rango de k. Por tanto, para este ejemplo concreto no se podría encontrar ninguna parte de la ganancia de alta frecuencia que fuera viable.

En cambio, tal y como se ha propuesto en esta sección, si situamos el valor del polo en su mínimo, $a = 1$, entonces el valor de la frontera en el lazo izquierdo de la gráfica, donde $a = 1$, coincidiría con $k = 2.5$; siendo el resto del rango, $k = [2.5 \ldots 10]$ viable y, por tanto, conteniendo soluciones.

La solución obtenida con $k = 2.5$ se podrá usar para realizar podas en otros nodos cuyos valores k_{hf} sean superiores. Pero, como ya se ha explicado

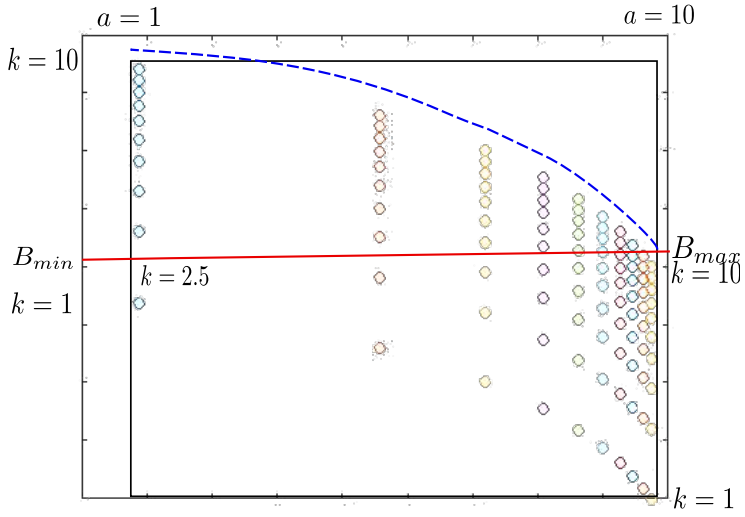

Figura 4.4: Ejemplo de búsqueda de mejor ganancia de alta frecuencia.

anteriormente, esta solución sólo es válida con $a = 1$. Del propio ejemplo se desprende que, si varía el valor de a, ese valor dejaría de cumplir las restricciones de diseño. En el ejemplo ilustrado, donde solo interviene otra variable, que es un polo, se puede observar claramente como éste es el mejor valor conseguible para la ganancia de alta frecuencia. Este ejemplo es muy sencillo de entender visualmente dado que, al contener solo dos variables el controlador, se puede plasmar de forma fácil en un gráfico. Pero en un caso general con múltiples variables, otras combinaciones de parámetros podrían dar un resultado mejor que el obtenido con esta estrategia. Por tanto, la solución obtenida por este procedimiento nunca podrá ser valorada como un óptimo global. Ademas, en este ejemplo sólo se está teniendo en cuenta una frecuencia de diseño. Por tanto, en un problema estándar se deberán realizar las operaciones y obtener la intersección de los subrangos de k_{hf} obtenidos como viables para cada una de ellas y, de esta forma, será viable para todas.

4.4. Etapas de ejecución e historia del nodo

En los algoritmos planteados en la sección (3), desde que se inician hasta que se terminan, siempre aplican los mismos procedimientos de manera sistemática. Independientemente del estado en el cual se encuentre la ejecución.

Después de estudiar a fondo esta forma de proceder, se ha llegado a la conclusión de que algunos procedimientos base del algoritmo, como son la bisección o algunas de las mejoras, como es la acotación del espacio de búsqueda, podrían funcionar de forma más efectiva si variaran su comportamiento a lo largo de la ejecución del algoritmo.

Se propone dividir la ejecución del procedimiento en tres etapas principales, donde en cada una de las etapas podrá variar el funcionamiento de una o varias funcionalidades.

A continuación, se entra en detalle de cada una de las etapas.

4.4.1. Etapa inicial

La etapa inicial se establece con el comienzo del algoritmo hasta que la caja de proyección deja de ocupar todo el ancho en frecuencia en el plano de Nichols. Es común que, cuando se inicia el algoritmo, si se han establecido rangos grandes para las variables del controlador, la caja proyectada en el plano de Nichols ocupe todo el ancho en frecuencia, de -360 a 0 grados. Esta situación puede darse incluso aunque se ya hayan realizado bisecciones en el nodo.

Para este tipo de situaciones, donde la caja proyectada tiene un tamaño tan grande, el interés principal es reducir su extensión de forma rápida sin centrarse en ninguna orientación concreta del plano. Para este fin, la acotación del espacio de búsqueda en magnitud puede ser muy útil dada la distancia del borde de las cajas de proyección con las fronteras, pudiendo detectar grandes partes inviables y viables de las variables del controlador. En cambio, la acotación del espacio de búsqueda utilizando la fase no tiene sentido si existen fronteras abiertas, dado que al ocupar la caja proyectada todo el ancho en frecuencia, este procedimiento no se puede realizar. Por tanto, su aplicación no obtendría ningún resultado y, en cambio, sí que consumiría tiempo de ejecución.

Por otro lado, la bisección se realiza utilizando el área (4.2.1) dado que el interés principal es reducir el tamaño de la caja de forma genérica, no especificando o detallando que se prefiere la reducción en fase, en magnitud u otra forma más avanzada de bisección.

La búsqueda de mejor ganancia también está activada con el objetivo de encontrar soluciones que permitan podar el espacio de búsqueda.

4.4.2. Etapa intermedia

La etapa intermedia se inicia después de la etapa inicial, cuando la caja proyectada ya no ocupa todo el ancho en fase del plano de Nichols. Esta etapa se caracteriza por tener cajas proyectadas de tamaño mediano donde la acotación del espacio de búsqueda, tanto en fase como en magnitud, puede dar buenos resultados, aunque siempre dependiendo de la casuística concreta

del problema.

Por otro lado, la bisección se realiza por el procedimiento del árbol (4.2.4), dado que el objetivo en esta fase ya no es solo reducir el tamaño de la caja proyectada de forma genérica, sino que se busca realizar la bisección de forma más inteligente aprovechando las características adicionales que aporta este tipo de bisección.

La búsqueda de mejor ganancia adquiere importancia por su capacidad de podar nodos que no mejoran las soluciones ya encontradas, pudiendo ser una herramienta fundamental a la hora de reducir el espacio de búsqueda.

4.4.3. Etapa final

Esta etapa se inicia al acabar la etapa intermedia. Se establece cuando el procedimiento de acotación del espacio de búsqueda tanto en magnitud como en fase ya no obtiene resultados. Esta situación se da cuando la caja proyectada se hace pequeña con respecto a la frontera y la intersección con la misma se aproxima a una línea recta. Esto quiere decir que no hay parte ambigua en la caja proyectada y, por tanto, con realizar una última detección de la parte viable e inviable puede desactivarse esta característica. La desactivación de esta funcionalidad puede darse en fase y/o magnitud. No es necesario que suceda en los dos tipos de acotación a la vez. También se desactivará la *Búsqueda de mejor ganancia*, dado que al igual que los procedimientos mencionados, no producirán resultados y consumiría tiempo de cómputo.

Para terminar, la bisección se realizará por fase o magnitud (4.2.3 y 4.2.2), dependiendo en cada momento de cuál de las dos tiene mayor ancho en el plano de Nichols, dado que en esta fase final del algoritmo, interesa reducir el tamaño de las cajas de forma similar en las dos coordenadas para que ninguno de los lados de la caja proyectada se quede con un tamaño significativamente mayor que el otro, puesto que la función solución del algoritmo (3.1.3) fija que terminará cuando se cumpla la condición de tamaño fijado por ϵ tanto en fase como en magnitud. Por ello, una reducción homogénea de las dos coordenadas acelerará el fin del algoritmo.

4.4.4. Historia del nodo

La estrategia planteada en (4.4) de variar el comportamiento del algoritmo y las funcionalidades que se aplican a lo largo de la ejecución del mismo, tiene una dificultad inherente a este tipo de algoritmos. Al ir expandiendo las ramas del árbol de parámetros cada nodo puede estar en una etapa distinta, sobretodo porque unas ramas se expanden más que otras. Por tanto,

es difícil definir unas etapas del algoritmo de forma general para todos los nodos. Con el objetivo de solucionar este problema, cada nodo guardará, aparte del momento actual, los datos que se estimen relevantes del pasado del mismo. Gracias a esto, el algoritmo se situará en la etapa adecuada con respecto al nodo actual que esté en ejecución.

Cada nodo contiene información. Por un lado, guarda de los datos básicos de la función de transferencia del controlador, si es viable o ambiguo (si fuera inviable ya se habría eliminado) y la mejor ganancia de alta frecuencia del mismo para usarse como índice de la lista de nodos vivos (3.1.6). Además, añade a esta información, la etapa actual del nodo, que se irá actualizando conforme se den las condiciones explicadas para cada una de las etapas. Por otro lado, guarda también las frecuencias para las cuales el nodo es viable, ya sea propiamente calculado por la función de comprobar factibilidad (3.1.1) o determinado por la bisección en árbol (4.2.4).

De esta forma cada nodo podrá guardar su información asociada y el algoritmo tomará las decisiones adecuadas conforme a la información que le proporcione.

4.5. Detección de la violación en Nyquist

La detección de la violación de las fronteras por parte de las cajas proyectadas para cada frecuencia de diseño se ha realizado tradicionalmente en el plano de Nichols. Este plano se compone de fase en grados y de magnitud en decibelios, por tanto, las operaciones con números complejos intervalares que componen el controlador deben trasladarse desde coordenadas cartesianas a coordenadas polares. Cuando se realiza este cambio con números complejos intervalares se obtiene una representación conservadora y, de esta forma, la caja proyectada en el plano de Nichols es, generalmente, más conservadora que si se comparara con las coordenadas cartesianas.

Como solución a esta problemática se propuso realizar la comprobación de factibilidad utilizando coordenadas cartesianas. En lo que se refiere al cálculo del controlador por la planta nominal utilizando la *Natural Interval Extension* (1.2.5), ya se realizan estas operaciones en coordenadas cartesianas, y solo al final del procedimiento se transforman a coordenadas polares para usarse en el plano de Nichols. Por tanto, para implementar este procedimiento, sólo es necesario no realizar este último paso. Por otro lado, las fronteras tradicionalmente se han calculado en magnitud y fase, por tanto, es necesario realizar el traspaso de coordenadas a forma cartesiana. Una vez que se disponga de los datos en forma adecuada sólo sería necesario realizar el procedimiento de comparación entre las fronteras y la caja proyectada.

Esta estrategia se implementó y se añadió como opción al algoritmo, pero los resultados que se obtuvieron no mejoraron el tiempo de ejecución del algoritmo, todo lo contrario, incluso lo empeoraron. El motivo principal de este resultado fue que, la mayor dificultad para realizar las operaciones en en el plano de Nyquist, no compensaba la potencial mejora que aportaba realizar la comprobación de factibilidad utilizando coordenadas cartesianas y, por tanto, esta estrategia no ha sido incorporada al algoritmo propuesto en (5).

4.6. Acotación del espacio de búsqueda en Nyquist

Tal y como se ha propuesto en la sección anterior (4.5) se puede realizar la comprobación de factibilidad utilizando coordenadas cartesianas. Se propuso aplicar esta misma idea a la acotación del espacio de búsqueda (3.2.2) en la parte del plano inviable utilizando la magnitud. Hasta ahora se habían utilizado coordenadas polares en el plano de Nichols en decibelios y grados. La idea principal es que al realizar el cambio de coordenadas de cartesiano a polar se pierde precisión en la caja proyectada, por tanto, si se puede realizar los recortes sin necesidad de realizar este cambio de coordenadas, el resultado será más preciso y el recorte podrá ser mayor.

Este sistema adolece de un problema principal y es que, las coordenadas cartesianas como resultado del cálculo del lazo abierto, no son monótonas. En una primera idea se intentó caracterizar el resultado del cálculo en una monotonía por partes, dependiente de las variables del controlador, que pudiera ser manejada adecuadamente para obtener un resultado satisfactorio; pero después de investigar más profundamente y apoyado en casos prácticos, se llegó a la conclusión de que la identificación de la monotonía por partes dependía de la combinatoria del número de variables del controlador y que, por tanto, las operaciones a realizar para poder identificar la monotonía por partes se complicaban tanto que resultaba inviable aplicarlas en el algoritmo. Finalmente se descartó usar este procedimiento.

Capítulo 5

Algoritmo propuesto

El objetivo fundamental de esta tesis es, tomando como base los algoritmos NT (3.1) y NK (3.2), proponer una nuevo algoritmo que integre y coordine todas las estrategias propuestas en el capítulo (4). Tiene como finalidad mejorar el coste computacional de las propuestas previas y, derivado de ello, mejorar el tiempo de ejecución, permitiendo así aumentar lo máximo posible el tamaño del problema.

A continuación, se estructuran todas las estrategias propuestas con el objetivo de clasificarlas para su posterior inclusión en el algoritmo:

1. **Acotación del espacio de búsqueda:** Estrategias destinadas a reducir el espacio de búsqueda, resumidas en la sección (4.1.3). Pueden categorizarse del siguiente modo:

   a) **Acotación del espacio de búsqueda en el subrango inviable:** Esta estrategia tiene como objetivo detectar para cada variable del controlador, qué parte de su rango es inviable para cada frecuencia de diseño. De esta manera, sabiendo que viola las restricciones de diseño, se puede eliminar dicha parte. Esta estrategia a su vez se subdivide en dos procedimientos:

   - **Acotación del espacio de búsqueda en el subrango inviable utilizando la magnitud:** Este primer procedimiento es el propuesto por el algoritmo NK (3.2.2) que, utilizando la magnitud en el plano de Nichols, tiene como objetivo detectar cuál es la parte inviable de cada una de las variables del controlador.
   - **Acotación del espacio de búsqueda en el subrango inviable utilizando la fase:** Este procedimiento, detallado en (4.1.2) tiene el mismo objetivo que el anterior, pero en este caso, utilizando la fase en el diagrama de Nichols.

b) **Acotación del espacio de búsqueda en el subrango viable:** Esta estrategia tiene por objetivo detectar, para cada frecuencia de diseño, y para cada variable del controlador, qué parte de las mismas sólo contiene soluciones. De esta forma, una vez realizada la intersección entre todas las porciones de rangos calculadas para cada frecuencia de diseño, se puede determinar qué parte del espacio de parámetros sólo contiene soluciones. Esta estrategia, de forma similar a la anterior, se compone de dos procedimientos:

- **Acotación del espacio de búsqueda en el subrango viable utilizando la magnitud:** Este procedimiento, descrito en (4.1.1), propone detectar las partes viables de cada variable del controlador utilizando la magnitud en el diagrama de Nichols.
- **Acotación del espacio de búsqueda en el subrango viable utilizando la fase:** Este procedimiento aplica dos de las estrategias descritas en (4.1.1 y 4.1.2) con el objetivo de conseguir encontrar la parte viable de cada una de las frecuencias de diseño del controlador utilizando la fase en el diagrama de Nichols.

2. **Bisección del nodo:** Esta estrategia propone varias formas de biseccionar el nodo (4.2). Cada una de ellas centrada en una necesidad distinta para el momento en el que se ejecuten. Está relacionada con la estrategia *Etapas del nodo* (4.4). Dicha estrategia, que tiene como fundamento variar el funcionamiento del algoritmo a lo largo de su ejecución, selecciona las mejores estrategias según el momento en que se encuentre:

   - **Bisección utilizando el área:** Este procedimiento se describe en (4.2.1). Tiene como base de funcionamiento realizar la bisección por la variable del controlador que más reduzca el área de la proyección en el plano de Nichols de los subnodos generados.
   - **Bisección utilizando la magnitud:** Este procedimiento descrito en (4.2.2) tiene por objetivo realizar la bisección del nodo por la variable del controlador que más reduce su tamaño en magnitud de la proyección en el plano de Nichols. Esta forma de proceder es adecuada cuando la casuística concreta necesite reducir la magnitud de las cajas proyectadas.
   - **Bisección utilizando la fase:** Equivalente al procedimiento anterior pero con el objetivo de reducir la fase de la caja proyectada en el plano de Nichols. Se describe en (4.2.3).
   - **Bisección en árbol:** Este procedimiento, descrito en (4.2.4), utiliza los datos calculados en la estrategia (4.1.1) para realizar la

bisección en el lugar exacto donde la variable del controlador pasa de ser ambigua a ser viable para una, o varias, frecuencias de diseño. De este modo, se obtienen dos subnodos, uno de ellos es viable a ciertas frecuencias, siendo esta información utilizada, en este mismo nodo y en su descendencia, para ahorrar cálculos innecesarios. Esta forma de proceder tiene como objetivo conseguir que el nodo sea viable para el mayor número posible de frecuencias de diseño y así poder centrar el esfuerzo en las restantes frecuencias donde todavía sea ambiguo.

3. **Búsqueda mejor ganancia de alta frecuencia:** Esta estrategia se describe en (4.3) y funciona encontrando soluciones rápidas al problema donde la ganancia de alta frecuencia sea lo menor posible. Esto permite podar o recortar en etapas tempranas nodos del espacio de búsqueda con el consiguiente ahorro sustancial de tiempo.

4. **Etapas de ejecución e historia del nodo**: Esta estrategia desarrollada en (4.4) propone que, dado que la relación entre la caja de proyección y las fronteras varía cualitativamente a lo largo de la ejecución del algoritmo, es una opción prometedora modificar el funcionamiento del procedimiento y adaptarlo a la situación en la que el nodo esté en cada momento. Para ello, dado que cada nodo es independiente y tendrá un desarrollo diferente al resto dependiendo principalmente de cuándo se haya expandido la rama del árbol de búsqueda donde se sitúe, es necesario guardar la historia del nodo. Una vez que se conoce con detalle el momento concreto del nodo, se pueden tomar diferentes decisiones. Por ejemplo, elegir una forma u otra de bisección según sea más conveniente, o desactivar las estrategias de acotación del espacio de búsqueda cuando ya no tenga sentido ejecutarlas.

Una vez listadas todas las estrategias (4) que se van a incorporar al algoritmo, es importante indicar que, para facilitar la legibilidad y claridad de las mismas, varias de estas estrategias serán utilizadas mediante la definición de una serie de funciones que las implementan con la intención de incorporarlas posteriormente al algoritmo general mediante la invocación de dichas funciones. Otras estrategias, dada su naturaleza más transversal al algoritmo, serán implementadas directamente en el esquema general del mismo.

Todas las estrategias propuestas, junto con el algoritmo general, van a ser descritas en pseudocódigo. Esto permitirá obviar los detalles específicos de la implementación pero tendrá el suficiente grado de detalle como para llevar a la práctica todas las ideas expuestas. Si se quiere visualizar en detalle del código implementado, el capítulo (7) describe detenidamente el software desarrollado, así como el acceso de forma libre al mismo, bajo licencia GNU.

A continuación, se explica y se muestra el pseudocódigo de cada una de las funciones en las que se divide el algoritmo para terminar con el esquema general del mismo.

5.1. Acotación del espacio de búsqueda

En este primer apartado, que corresponde a las estrategias listadas en el punto (1), se van a describir las funciones asociadas a la acotación del espacio de búsqueda. Dicho apartado se va dividir en dos partes, correspondiente cada una de ellas al tratamiento de un subrango de cada variable del controlador. Por un lado, se va a afrontar la acotación del subrango inviable. Por otro lado y en una segunda función, se va a definir la acotación del subrango viable, utilizando ambas partes los cálculos en fase y en magnitud en el plano de Nichols.

Las fronteras en el plano de Nichols pueden ser de dos tipos. En primer lugar están las abiertas, que tienen como característica principal que ocupan todo el ancho en fase, además pueden ser multivaluadas, pero nunca definirán un espacio cerrado. Este tipo de fronteras, al tener una predominancia de variación en fase en el plano de Nichols, la acotación que hacen utilizando la magnitud suele obtener buenos resultados. Eso no quita que, en algunas zonas particulares, la frontera pueda mostrar comportamientos de variación en magnitud que permitan aplicar la acotación utilizando la fase.

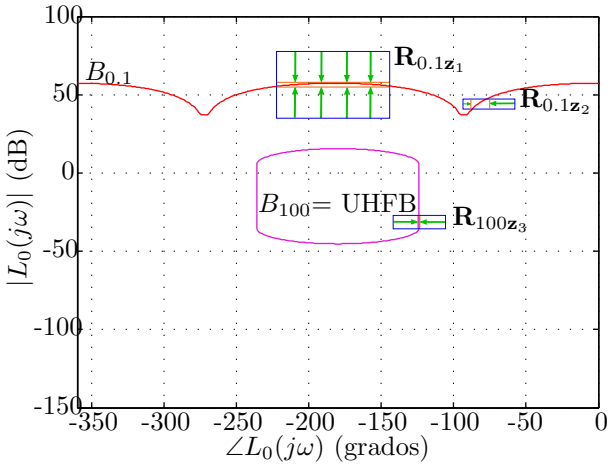

Figura 5.1: Posibilidad de acotación del espacio de búsqueda en fase y magnitud en el plano de Nichols.

En segundo lugar, existen las fronteras cerradas, que encierran parte del espacio del plano de Nichols. En este tipo de fronteras, la acotación del espacio de búsqueda se puede realizar de forma equivalente, tanto en fase como en magnitud. La efectividad de cada uno de estos métodos dependerá de cada

problema concreto y de como varíe en fase y en magnitud cada una de las fronteras.

En la figura (5.1) se pueden observar los dos tipos de fronteras señalados, el $B_{0.1}$ corresponde a una frontera abierta, y el B_{100} a una frontera cerrada. A su vez, en las cajas proyectadas $\mathbf{R}_{0.1\mathbf{z}_1}$, $\mathbf{R}_{0.1\mathbf{z}_2}$ y $\mathbf{R}_{100\mathbf{z}_3}$, se puede observar, indicado con flechas, qué subcajas serían viables e inviables, tanto en fase como en magnitud, y que pueden ser detectadas por este procedimiento. A su vez, se puede advertir que, tanto en las fronteras abiertas como en las cerradas, existe la posibilidad de acotar el espacio de búsqueda utilizando tanto la fase como la magnitud.

A continuación, se detallan las funciones para realizar la acotación del espacio de búsqueda tanto en el subrango inviable como en el subrango viable, siendo cada una de ellas un procedimiento independiente.

5.1.1. Acotación del espacio de búsqueda, subrango inviable

Como se ha indicado, esta primera función está centrada en la acotación del subrango inviable utilizando la fase y magnitud en el plano de Nichols. Con el objetivo, de conseguir identificar, para cada variable del controlador cual es su parte inviable (1a), y de esta forma poder descartar dicha parte, reduciendo así el espacio de búsqueda del algoritmo.

La función implementada incluye la función QS (3.4) presentada en el algoritmo NT (3.2.2) y centrada en acotar el espacio de búsqueda utilizando la magnitud completa con la presentada en la estrategia (4.1.2) para realizar esta misma acotación utilizando también la fase.

Es necesario tener en cuenta que, para simplificar la exposición del pseudocódigo, se describe solo el caso en que las fronteras abiertas delimitan por su mitad superior la parte viable y por su mitad inferior la parte inviable, y las fronteras cerradas tienen en su interior la parte inviable y en su exterior la parte viable. Si bien, el algoritmo implementado contempla todos los casos posibles, para llevar a cabo esta tarea, la función de factibilidad (3.1.1), reporta estas características propias de las fronteras para que el algoritmo adapte su funcionamiento, invirtiendo las funciones *superior* (sup) e *inferior* (inf), cuando sea necesario.

La función recibe como parámetros de entrada los siguientes datos:

- $B_{nf\Omega}$ son los valores mínimos en fase de las fronteras dentro de cada caja proyectada.

- $B_{nm\Omega}$ son los valores mínimos en magnitud de las fronteras dentro de cada caja proyectada[1].
- $\mathbf{\Omega}$ es el conjunto de frecuencias de diseño.
- $\mathbf{z}$ es la caja de parámetros.

A continuación, se describe el funcionamiento del algoritmo propuesto (5.1). Los apartados se han identificado mediante números para facilitar su explicación:

- En primer lugar se llama a la función *QS*, que se encarga de calcular la acotación utilizando la magnitud (Apartado ❶).
- Se continúa con la acotación utilizando la fase. Para ello, se recorren las frecuencias de diseño, llevando a cabo los siguientes pasos para cada una. (Apartado ❷).
- Si $\mathbf{z}$ es viable para ω salta a la siguiente caja, dado que este procedimiento sólo funciona con $\mathbf{z}$ que sean ambiguas para la frecuencia de diseño seleccionada (Apartado ❸).
- Se recorren las variables del controlador, exceptuando la ganancia de alta frecuencia (que no tiene componente de fase) y para cada una de estas variables se calcula la ecuación correspondiente desarrollada (4.1.2) (Apartado ❹).
- Para ello, es necesario seleccionar, para el resto de variables que no se están despejando, qué valores deben tomar de sus rangos para maximizar el resultado del controlador (Apartado ❺).
- Con este objetivo, para cada cero del controlador se seleccionará el valor superior de su rango y para cada polo del controlador, el valor inferior de su rango (Apartado ❻).
- A continuación, se aplica la ecuación correspondiente. Si j es un polo, se utiliza (4.7); si es un cero, se utiliza (4.6). Y se calcula el subintervalo inviable para dicha variable (Apartado ❼).
- El subintervalo identificado como inviable se elimina de $\mathbf{z}$ (Apartado ❽).
- Estos pasos se repiten para el resto de variables del controlador, en la frecuencia de diseño actual. Y posteriormente, para el resto de frecuencias.

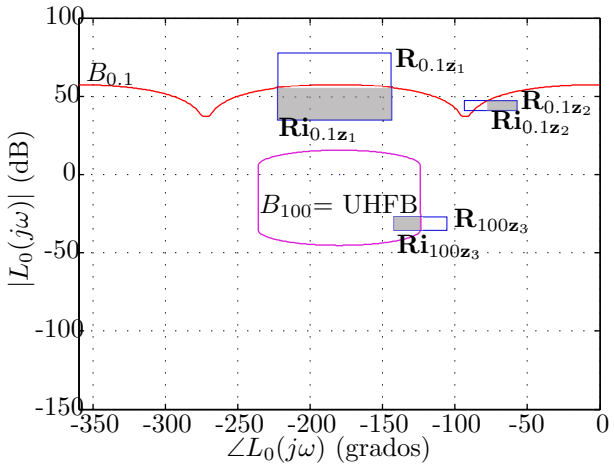

Figura 5.2: Acotación de la subcaja inviable utilizando la fase y la magnitud en el plano de Nichols.

Como se puede observar en el ejemplo de la figura (5.2), el objetivo de esta función es poder detectar, y posteriormente eliminar, todas las subcajas inviables de las cajas proyectadas en el plano de Nichols. Siendo éstas denominadas, en dicho ejemplo, como $\mathbf{Ri}_{0.1\mathbf{z}_1}$, $\mathbf{Ri}_{100\mathbf{z}_3}$ y $\mathbf{Ri}_{0.1\mathbf{z}_2}$. Como se ha definido en (1.2.5), cuando se utiliza la *Extensión natural a intervalos* y la caja se proyecta en el plano de Nichols, este procedimiento no es reversible. No existe correspondencia directa entre la proyección en el plano de Nichols y el espacio de parámetros. Por tanto, el objetivo de esta función será conseguir trasladar parte de la información de la proyección al espacio de parámetros con el objetivo de detectar la subcaja inviable en cada uno de los subrangos de las variables del controlador. La intención final es eliminar dicha parte y reducir así el espacio de búsqueda del algoritmo.

Algoritmo 5.1: Pseudocódigo del algoritmo *QSInv*

```
1  Entradas: B_nmΩ, B_nfΩ, Ω y z
2  Salidas: z
3
4  // APARTADO 1, QS es usada para reducir z usando la información de la magnitud.
❶ 5  z ← QS(B_nmΩ, Ω, z) // Función implementada en (3.4)
6
7  // APARTADO 2, se reduce z usando la información de fase
❷ 8  Para ω ∈ Ω,
❸ 9    Si z no es viable para ω
❹10      Para j ∈ {2,...,n} // se excluye la k porque no tiene influencia en la fase.
11        y ← z // z es copiado a y
❺12        Para λ ∈ {2,...,n} − {j} // En este bucle, y recibe el valor r en z[λ] que maximiza la
           contribución de x(λ) a ∠L_0(jω, x) ···
13          Si 2 ≤ j ≤ 1 + n_rz + 2n_cz // cero
14            r ← sup(z[λ])
❻15         Sino  // polo
16            r ← inf(z[λ])
17          Finsi
18          y ← subs(y, λ, [r, r])    // intervalo y[λ] ← r
```

[1]Estos valores serán utilizados por las ecuaciones correspondientes para calcular los subrangos inviables en cada una de las frecuencias.

```
19        Finpara
20
21        Si {j} es un cero
22          S ← utilizando (4.6) mayor subrango de y[j] tal que subs(y, j, S) es inviable en ω
23        Sino
24          S ← utilizando (4.7) mayor subrango de y[j] tal que subs(y, j, S) es inviable en ω
25        Finsi
26        S ← mayor subintervalo de y[j] tal que subs(y, j, S) es inviable para ω
27        iz ← subs(z, j, S)
28        z ← z − iz
29      Finpara
30    Finsi
31  Finpara
```

5.1.2. Acotación del espacio de búsqueda, subrango viable

La segunda función, que se describe a continuación, está centrada en acotar el espacio de búsqueda del algoritmo identificando el subrango viable de cada una de las variables del controlador, utilizando para ello la fase y la magnitud en el plano de Nichols (1*b*).

Para conseguir este objetivo, utiliza dos de las estrategias propuestas en el capítulo (4). La estrategia propuesta en (4.1.1), donde se explica la posibilidad de identificar las partes viables de cada una de las variables del controlador y la estrategia propuesta en (4.1.2), donde se desarrolla cómo añadir a este procedimiento la posibilidad de utilizar la fase en el plano de Nichols y no sólo la magnitud.

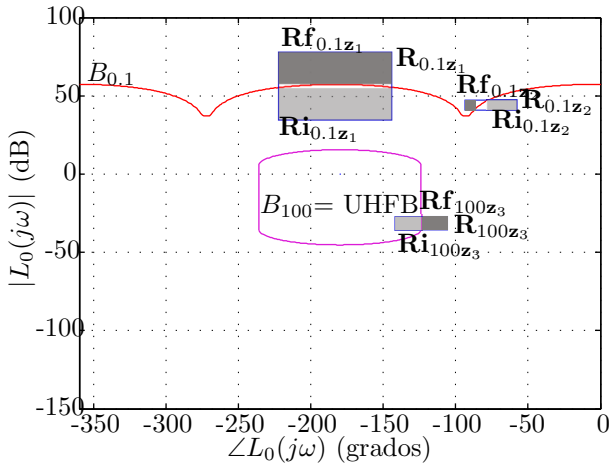

Figura 5.3: Acotación de la subcaja inviable utilizando la fase y la magnitud en el plano de Nichols.

Como se ha indicado en la función anterior, la utilización de la *Extensión natural a intervalos* no es reversible una vez que se proyecta la caja en el plano de Nichols y no hay correlación directa con el espacio de parámetros.

Por tanto, el objetivo es trasladar la parte detectada como viable a cada una de las variables del controlador. En el ejemplo de la figura (5.3), se pueden observar las subcajas $\mathbf{Rf}_{0.1\mathbf{z}_1}$, $\mathbf{Rf}_{0.1\mathbf{z}_2}$ y $\mathbf{Rf}_{100\mathbf{z}_3}$, identificadas como viables, junto con el resto de subcajas, identificadas como inviables por la función (5.1.1).

Como ya se indicó en el apartado (4.1.1), la identificación de partes viables es diferente a la identificación de partes inviables porque, para que un subrango de una variable pueda ser marcado como viable, lo debe de ser para todas las frecuencias de diseño, y así cumplir con las restricciones impuestas por todas las fronteras. Esto se traduce en que, una vez que se han detectado los subrangos viables para cada una de las frecuencias de diseño, el resultado viable para todas ellas será la intersección de los subrangos. El resto de valores viables que no son comunes a todas las frecuencias de diseño no se descartan, sino que se devuelven como resultado del procedimiento, dado que, posteriormente, podrían ser utilizados por la función (5.3).

Para terminar, es necesario comentar que el procedimiento implementado en (5.2) es complementario y más efectivo que éste para encontrar soluciones. Por tanto, en el esquema general del algoritmo se decidió que, si el procedimiento *Búsqueda de mejor ganancia* termina con éxito y obtiene resultado, se descarta ejecutar este apartado. El motivo de esta decisión se debe a que tratan de resolver la misma cuestión, es decir, encontrar soluciones al problema con el objetivo de realizar podas y acotaciones del espacio de parámetros. Por tanto, dado que solaparían funcionalidad y consumirían más tiempo de ejecución, no es interesante ejecutar los dos a la vez. Dado lo anterior, este procedimiento sólo se ejecutará cuando la *Búsqueda de mejor ganancia* no encuentre resultado.

Igual que en el apartado anterior, es necesario tener en cuenta que, para simplificar la exposición del pseudocódigo, se describe solo el caso en que las fronteras abiertas delimitan por su mitad superior la parte viable y por su mitad inferior la parte inviable, así como, las fronteras cerradas tienen en su interior la parte inviable y en su exterior la parte viable. Si bien el algoritmo implementado contempla todos los casos posibles, para llevar a cabo esta tarea, la función de factibilidad (3.1.1) reporta estas características propias de las fronteras para que el algoritmo adapte su funcionamiento, invirtiendo las funciones *superior* (sup) e *inferior* (inf), cuando sea necesario.

La función recibe como parámetros de entrada los siguientes datos:

- $B_{xf\Omega}$ que son los valores máximo en fase de las fronteras dentro de cada caja proyectada. $B_{xm\Omega}$ que son los valores máximos en magnitud de las fronteras dentro de cada caja proyectada[2].

[2]Estos valores serán utilizados por las ecuaciones correspondientes para calcular los

- $\mathbf{\Omega}$ es el conjunto de frecuencias de diseño.
- $\mathbf{z}$ es la caja de parámetros.

A continuación, se describe el funcionamiento del algoritmo propuesto (5.2), donde se ha identificado mediante números los apartados para facilitar su explicación:

- En primer lugar, se recorren todas las variables del controlador, para realizar operaciones sobre ellas (Apartado ❶).
- A continuación, se recorren todas las frecuencias de diseño, dado que las operaciones que se realizan para cada variables son dependientes de una frecuencia de diseño concreta (Apartado ❷).
- Si la caja de parámetros ($\mathbf{z}$) es viable para la frecuencia actual (ω), se salta a la siguiente frecuencia de diseño, dado que estas operaciones sólo se pueden aplicar sobre cajas ambiguas (Apartado ❸).
- El siguiente paso será seleccionar los valores de cada una de las variables del controlador que participan en la ecuación para calcular el valor de la variable actual (j). En este caso, para la ganancia de alta frecuencia y para los ceros se selecciona su valor mínimo, y para los polos su valor máximo; cada una de ellas dentro de su rango (Apartado ❹).
- Con los valores adecuados en cada una de las variables, se aplica la ecuación correspondiente según sea la variable a despejar (Apartado ❺).
- Una vez calculados los subrangos viables para cada variable y para cada frecuencia de diseño, se seleccionará la intersección de todos ellos en magnitud (Apartado ❻).
- Se continúa quitando de la caja actual el subrango detectado como viable en magnitud, para así dejar solo los subrangos ambiguos. Además, los subrangos viables se almacenan en un vector (Apartado ❼).
- Se realiza este mismo proceso para la fase. En primer lugar selecciona el mejor subrango detectado como viable y, a continuación, quita dicho subrango de la caja de parámetros (Apartado ❽).
- El siguiente paso será almacenar los subrangos viables restantes de la intersección (Apartado ❾).
- Para terminar, la función retornará los siguientes datos:

subrangos viables en cada una de las frecuencias.

- **z**: La caja de parámetros que recibía la función como entrada con los subrangos viables eliminados (Apartado ⑩).
- **UM**: Vector que contiene los subrangos viables para todas las frecuencias de diseño detectados en la acotación utilizando la magnitud.
- **UF**: Vector que contiene los subrangos viables para todas las frecuencias de diseño detectados en la acotación utilizando la fase.
- **MM**: Vector de vectores, uno por cada variable del controlador, que contiene los subrangos viables para solo alguna frecuencia de diseño en magnitud.
- **MF**: Vector de vectores, uno por cada variable del controlador, que contiene los subrangos viables para solo alguna frecuencia de diseño en fase.

Algoritmo 5.2: Pseudocódigo del algoritmo *QSFact*

```
1   Entradas: B_xmΩ, B_xfΩ, Ω y z
2   Salidas: z, UM, UF, MM y MF
3
❶ 4  Para j ∈ {1,...,n} // todas las variables del controlador
❷ 5    Para ω ∈ Ω,
❸ 6    Si z no es viable para ω
7        y ← z // z es copiado a y
❹ 8      Para λ ∈ {1,...,n} − {j} // En este bucle, y recibe el valor r en z[λ] que minimiza la
           contribución de x(λ) a ∠L_0(jω, x) y a |L_0(jω, x)|...
9          Si 2 ≤ j ≤ 1 + n_rz + 2n_cz // cero y ganancia de alta frecuencia
10           r ← inf(z[λ])
11         Sino                                  // polo
12           r ← sup(z[λ])
13         Finsi
14         y ← subs(y, λ, [r, r])    // intervalo y[λ] ← r
15       Finpara
❺ 16     Si {j} es la ganancia de alta frecuencia
17         S_1[ω] ← utilizando (4.1) mayor subrango de y[j] tal que subs(y, j, S_1[ω]) es viable en ω
18       Sino Si  {j} es un cero
19         S_1[ω] ← utilizando (4.2) mayor subrango de y[j] tal que subs(y, j, S_1[ω]) es viable en ω
20         S_2[ω] ← utilizando (4.6) mayor subrango de y[j] tal que subs(y, j, S_2[ω]) es viable en ω
21       Sino
22         S_1[ω] ← utilizando (4.3) mayor subrango de y[j] tal que subs(y, j, S_1[ω]) es viable en ω
23         S_2[ω] ← utilizando (4.7) mayor subrango de y[j] tal que subs(y, j, S_2[ω]) es viable en ω
24       Finsi
25     Finsi
26   Finpara
27
❻ 28 V_M ← min(S_1[1,...,n_ω]) // intersección magnitudes
29   eliminar(S_1, V_M)
30   mz ← subs(z, j, V_M)
31   UM[j] ← mz
❼ 32 y ← y − mz
33
❽ 34 V_F ← min(S_2[1,...,n_ω]) // intersección fases
35   eliminar(S_2, V_F)
36   fz ← subs(z, j, V_F)
37   UF[j] ← fz
38   y ← y − fz
39
❾ 40 MM[j] ← S_1
41   MF[j] ← S_2
```

```
42  Finpara
43
44  z ← y
```

5.2. Búsqueda de la mejor ganancia de alta frecuencia

La función que se plantea en esta sección implementa la estrategia de *Búsqueda de la mejor ganancia de alta frecuencia* (4.3). En líneas generales, este procedimiento tiene como objetivo encontrar la menor ganancia de alta frecuencia que es solución al problema que se esté tratando con la caja planteada. Esto se traduce en encontrar el subrango viable más grande de la ganancia de alta frecuencia que es solución para todas las frecuencias de diseño, estando el resto de variables en un valor determinado. Objetivo similar al explicado en (5.1.2), pero con la diferencia fundamental de que, en este procedimiento, se busca encontrar una solución puntual para unos valores de **z** concretos. De esta forma, el subrango viable encontrado para la ganancia de alta frecuencia no podrá ser utilizado para acotar el espacio de búsqueda, eliminando este subrango de la caja de parámetros. Pero sí que servirá para podar las ramas del espacio de parámetros que tengan soluciones menos prometedoras que la ya encontrada o para reducir las cajas que no puedan ser podadas.

Parece importante añadir, como ya se ha explicado en (5.1.2) que, al ser este procedimiento complementario a *Acotación de las variables del controlador, subrango viable*, no se ejecutan los dos a la vez. Concretamente éste tiene prioridad al poder encontrar soluciones con mejor ganancia de alta frecuencia.

La función recibe como parámetros de entrada los siguientes datos:

- $B_{nm\Omega}$ que son los valores mínimos en magnitud de las fronteras dentro de cada caja proyectada[3].
- $\mathbf{\Omega}$ es el conjunto de frecuencias de diseño.
- **z** que es la caja de parámetros.

En este punto es necesario reseñar que, al igual que en funciones anteriores, para simplificar la exposición del pseudocódigo, se describe solo el caso en que las fronteras abiertas delimitan por su mitad superior la parte viable y por su mitad inferior la parte inviable, y las fronteras cerradas tienen en su interior la parte inviable y en su exterior la parte viable. El algoritmo

[3]Estos valores serán utilizados por las ecuaciones correspondientes para calcular los subrangos viables en cada una de las frecuencias.

implementado contempla todos los casos posibles, por lo que se eligen esos valores del rango de las variables para utilizarlos en la ecuación. En el código implementado se tiene en cuenta este hecho y se contemplan todas las posibilidades.

A continuación, se describe el funcionamiento del algoritmo propuesto (5.3), donde se ha identificado mediante números los apartados para facilitar su explicación:

- En primer lugar, se seleccionan los valores adecuados del rango para cada una de las variables, exceptuando la ganancia de alta frecuencia dado que es el objetivo del cálculo. En este caso para los ceros se selecciona su valor máximo del rango y para los polos su valor mínimo (Apartado ❶).
- A continuación, se recorren las frecuencias de diseño. Si la caja **z** no es ambigua para alguna de estas frecuencias, se salta dicha frecuencia, y se pasa a la siguiente (Apartado ❷).
- El siguiente paso es aplicar la ecuación correspondiente para el cálculo de la ganancia de alta frecuencia (Apartado ❸).
- Una vez calculado el valor, si es que este existiera, se guarda el mayor encontrado hasta ese momento, dado que al estar tratando el subrango viable, se debe guardar la porción del rango que sea viable para todas las frecuencias de diseño (Apartado ❹).
- Para terminar, si se encuentra resultado, se devuelve una nueva caja de parámetros **v** con la mejor solución en ganancia de alta frecuencia encontrada (Apartado ❺).

Algoritmo 5.3: Pseudocódigo del algoritmo *MG*

```
1  Entradas: B_xmΩ, Ω y z
2  Salidas: v, solución encontrada o vacío
3
4  y ← z // z es copiado en y
❶ 5  Para λ ∈ {2,...,n} // se excluye la k por que es la variable que se va a calcular
6    Si 1 ≤ λ ≤ 2 + n_rz + 2n_cz // cero
7      r ← sup(z[λ])
8    Sino                        // polo
9      r ← inf(z[λ])
10   Finsi
11   y ← subs(y, λ, [r, r]) // intervalo y[λ] ← r
12 Finpara
13 k_min ← inf(z[1]),  k_max ← sup(z[1]),  k_f ← −∞
❷ 14 Para ω ∈ Ω
15   Si z no es viable para esta ω
❸ 16     k_ω ← utilizando (4.8) valor mínimo de z[1] si subs(y, 1, [k_f, k_max]) es viable para ω, o −∞ si
         no existe ningún valor.
❹ 17     k_f ← max(k_ω, k_f) // intersección
18   Finsi
19 Finpara
```

```
❺ 20  Si k_f = −∞ // no existe caja viable
  21    v ← ∅
  22  Sino
  23    v ← subs(z, 1, [k_f, k_max])
  24  Finsi
```

5.3. Bisección en árbol

Este procedimiento describe la estrategia de *Bisección por el método del árbol* (4.2.4) que tiene por objetivo realizar una bisección guiada por la información disponible del nodo a biseccionar. Como se ha explicado en la sección (5.1.2), el procedimiento que allí se describe, retorna para cada variable del controlador una serie de puntos del rango de dicha variable donde esta se hace viable para una o varias frecuencias de diseño, pero nunca para todas, porque entonces se consideraría solución y se trataría de forma distinta. La clave del funcionamiento de esta función es que, si se realiza la bisección del nodo por alguno de estos puntos concretos, uno de los subnodos resultantes será viable para alguna de las frecuencias de diseño. De esta forma, con la bisección se facilita no sólo profundizar en la rama del árbol del nodo, sino también obtener soluciones parciales para algunas de las frecuencias de diseño. Así, por un lado no sería necesario comprobar la factibilidad del nodo para estas frecuencias de diseño (lo que ahorra tiempo de ejecución) y se consigue que la exploración del árbol de búsqueda se haga de una forma más guiada por la información disponible, en comparación con, simplemente dividir por la mitad el rango de un parámetro, como se hace en 3.1. En el algoritmo principal (5.6), en el apartado *G*, se indica el lugar donde se ejecutaría esta función.

Es necesario reseñar en este punto, al igual que en funciones anteriores, que para simplificar la exposición del pseudocódigo, se describe solo el caso en que las fronteras abiertas delimitan por su mitad superior la parte viable, y por su mitad inferior la parte inviable, así como, las fronteras cerradas tienen en su interior la parte inviable y en su exterior la parte viable. Si bien, el algoritmo implementado contempla todos los casos posibles.

La función recibe como parámetros de entrada los siguientes datos:

- **z** es la caja de parámetros.
- **MM** son los valores viables de cada una de las variables, utilizando la magnitud, para alguna de las frecuencias de diseño.
- **MF** son los valores viables de cada una de las variables, utilizando la fase, para alguna de las frecuencias de diseño.

La primera parte del procedimiento se basa en encontrar, para los subrangos recibidos en los parámetros **MM** y **MF**, cuál de ellos es el más grande.

Esto quiere decir que se seleccionará el subrango que más porcentaje de rango de una variable concreta convierta en viable la caja para una frecuencia de diseño. En la segunda parte, una vez seleccionado el subrango, se realizará la bisección por el valor del mismo donde la caja pase de viable a ambigua.

A continuación, se describe el funcionamiento del algoritmo propuesto (5.4), donde se ha identificado mediante números los apartados para facilitar su explicación:

- En primer lugar, el procedimiento busca el menor valor de los guardados para la ganancia de alta frecuencia (Apartado ❶).
- A continuación, se realiza el mismo procedimiento para los ceros y los polos del controlador. En este caso, empleando también los valores generados utilizando la fase (Apartado ❷).
- El siguiente paso se trata de, una vez seleccionado el valor que más porcentaje de rango indica que es viable, realizar la bisección por ese punto (Apartado ❸).
- Para terminar, el algoritmo retorna las dos subcajas generadas. La primera de ellas que se desconoce su factibilidad y la segunda se sabe que será viable para, al menos, una frecuencia de diseño (Apartado ❹).

Algoritmo 5.4: Pseudocódigo del algoritmo *BA*

```
1   Entradas:z, MM y MF
2   Salidas: z_1 y z_2
3
4    // Se empieza por la ganancia de alta frecuencia
❶ 5  i ← min(MM[1][1,...,n]) // Se busca la variable más propicia en la ganancia de alta frecuencia.
6    var ← 1
7
8    // Continúa con los polos y los ceros
❷ 9  Para j ∈ {2,...,n}
10     m ← min(min(MM[j][1,...,n]), min(MF[j][1,...,n]))// Se busca la variable más propicia para
           los polos y ceros del controlador.
11     Si m < i
12       i = m
13       var ← j
14     Finsi
15   Finpara
16
❸ 17   [z_1, z_2] ← bisección por la variable var por el punto del rango indicado por i.
18
❹ 19   Se marca la caja z_2 como viable para la ω por la cual se haya hecho la partición.
```

5.4. Esquema general del algoritmo

En esta sección se explica el esquema general del algoritmo (5.4.9) donde se ensamblan todas las estrategias propuestas en el capítulo (4). Algunas de ellas se han implementando, por claridad, en funciones independientes explicadas en las secciones anteriores y serán llamadas desde el algoritmo

principal. El algoritmo está dividido en diferentes etapas para facilitar así la comprensión y claridad del código.

A continuación, para una primera explicación, se presenta el esquema general sin entrar en el detalle de cada una de las etapas:

Algoritmo 5.5: Secciones del algoritmo *Martínez-Cervera (MC)*

Entradas: B_Ω y **z**, caja inicial de búsqueda
Salidas: **z**, caja con los parámetros del controlador calculados en sus valores óptimos, o mensaje de que "No existe solución válida en x"

A. Comprobar factibilidad de la caja inicial de búsqueda

B. Inicializar la lista ***NL*** y variable de poda C

C. Poda y reducción de cajas usando la información de C

D. Comprobar factibilidad de la caja actual

E. Cambio de etapa

F. ***B***úsqueda de soluciones y reducción del nodo

G. Bisección de la caja

H. Procesar las cajas z^1, z^2, UM y UF, e insertar en ***NL***

I. Ir a la etapa ***C***

En los apartados siguientes se van a explicar cada una de las etapas del algoritmo entrando en el detalle de la implementación y funcionalidad de cada una de ellas.

5.4.1. Comprobar factibilidad de la caja

Corresponde con los apartados A (❶) y D (❹) del algoritmo y su tarea fundamental es comprobar la factibilidad de la caja de parámetros con respecto a las fronteras. El apartado A, además de la mencionada funcionalidad, al ejecutarse en la parte inicial del algoritmo, incluye algunas instrucciones más de inicialización.

La funcionalidad principal de esta etapa es realizar una llamada a la función de factibilidad (3.1.1) y, según el resultado, tomar las decisiones oportunas. En el caso del apartado A, al ser esta la caja inicial de búsqueda, si la función retorna que la caja es inviable, quiere decir que la propuesta de controlador, junto con su espacio de parámetros, no es válida para dar solución al problema y, por tanto, el algoritmo termina e informa de ello. Si no fuera este el caso, el algoritmo continúa e inicializa algunas variables antes de continuar con el siguiente apartado.

En el caso del apartado D, después de comprobar la factibilidad de la caja actual del algoritmo, si esta fuera inviable, se descartaría y el algoritmo

volvería a la etapa C para seleccionar una nueva caja de la lista de nodos vivos. En caso contrario, la ejecución continúa con la caja actual seleccionada.

5.4.2. Inicializar la lista de nodos vivos y la variable de poda

Esta sección, que corresponde con el apartado B (❷), se encarga de inicializar la lista de nodos vivos con la caja inicial de búsqueda proporcionada por el usuario e inicializar la variable de poda C a infinito (∞), dado que todavía no se ha encontrado ninguna solución válida al problema.

5.4.3. Poda y reducción de la caja usando la información de C

Corresponde con el apartado C (❸) del algoritmo. Es la sección que inicia el bucle del algoritmo y que solo terminará cuando no encuentre solución al problema, o bien, encuentre la solución óptima. Por claridad expositiva, en diferentes puntos del pseudocódigo se utilizan saltos incondicionales para retornar a este apartado C, a diferencia del código realmente implementado, en el cual se utiliza programación estructurada.

Este apartado comienza extrayendo de la lista de nodos vivos la caja que esté en su primera posición, la de menor ganancia de alta frecuencia, para convertirla en la *caja acual* de búsqueda (**z**). Si dicha lista estuviera vacía, el algoritmo terminaría porque no podría encontrar una solución viable con la caja inicial propuesta. A continuación, se comprueba si la caja seleccionada tiene mejor ganancia de alta frecuencia que la almacenada en la variable C, que recoge la mejor solución encontrada hasta el momento. Si fuera así, y la caja **z** fuera peor, se descartaría y se podaría el nodo, volviendo a iniciar la etapa C. Si no, la segunda opción es comprobar si se puede reducir el rango de la ganancia de alta frecuencia. Esta situación podría darse si alguna parte del rango de k superara a la variable de poda C. De esta forma, se podría reducir el espacio de búsqueda del algoritmo. Este tipo de podas se pueden realizar en la ganancia de alta frecuencia, dado que la función solución del algoritmo es encontrar la mejor ganancia de alta frecuencia que cumpla las restricciones de diseño. Por ello, cualquier solución encontrada que sea mejor que la actual, o parcialmente mejor, puede podar parte del espacio de búsqueda.

Para terminar, indicar que la variable de poda C es una caja de parámetros que representa la mejor solución encontrada hasta el momento.

5.4.4. Cambio de etapa

Corresponde con el apartado E (❺) del algoritmo. En esta sección se implementan los cambios de etapa propuestos en la estrategia (4.4). La variable que regula las etapas se inicializa en el apartado A y cada subnodo (hijo) hereda el valor de esa variable desde el padre cada vez que se realiza la bisección. En este apartado se analiza la situación del nodo actual y, si cumple las condiciones para un cambio de etapa, se lleva a cabo.

Para terminar, es importante reseñar que esta estrategia va complementada con la *historia del nodo*, la cual es imprescindible para disponer de la información necesaria para realizar los cambios de etapa. Esta estrategia es transversal en el algoritmo y se implementa a lo largo de todo el código.

5.4.5. Búsqueda de soluciones y reducción del nodo

Corresponde con el apartado F (❻) del algoritmo, y tiene por objetivo aplicar las funciones de acotación del espacio de búsqueda. Estas funciones se aplicarán si, para el *nodo actual* que se esté tratando, tiene activa dicha funcionalidad según indique la estrategia de *Etapas de ejecución* (4.4). Si no fuera así, y esta opción no estuviera activa, este apartado no se ejecutaría y se pasaría al siguiente.

En cambio, si el nodo tiene la funcionalidad activa, en primer lugar se tratará de ejecutar la función de *Búsqueda de mejor ganancia* (5.3) que, tal y como se indica en su sección (5.2), trata de encontrar soluciones donde la ganancia de alta frecuencia sea lo menor posible, coincidiendo este criterio con la función objetivo del algoritmo. Si este procedimiento encuentra alguna solución que sea mejor que la variable de poda C, se establece como nueva mejor solución encontrada hasta el momento.

A continuación, si el apartado anterior no encuentra ninguna solución, se ejecuta la función (5.2) que, tal y como se indica en (5.1.2), trata de encontrar la parte viable de cada una de las variables del controlador.

Para terminar, se ejecuta la función (5.1) que, según se indica en el apartado (5.1.1), trata de encontrar las partes inviables de cada una de las variables del controlador para que estas puedan ser eliminadas y reducir así el espacio de búsqueda.

5.4.6. Bisección de la caja

En esta sección, correspondiente al apartado G (❼) del algoritmo, es donde se procede con la bisección del nodo, aplicando alguno de los métodos en

la estrategia (4.2). Una de las opciones será la mencionada como *Bisección en árbol* (5.3) que, dada su superior complejidad programática comparada con el resto de estrategias de bisección, tiene una función implementada por separado (5.4).

El tipo de bisección que se ejecutará dependerá de la *etapa* (4.4) en la que se encuentre el *nodo actual*. En la etapa inicial se utiliza la bisección por área (4.2.1). En cambio, si la etapa actual es la intermedia, se ejecutará la bisección en forma de árbol (4.2.4). Para terminar, si la etapa actual es la final, dependiendo de si la caja proyectada tiene mayor tamaño en fase o en magnitud en el plano de Nichols, se ejecutará la bisección en fase (4.2.3) o la bisección en magnitud (4.2.2).

5.4.7. Procesamiento de las cajas e inserción en la lista de nodos vivos

Para continuar, el apartado *H* (❽) se encarga de procesar todas las cajas resultantes de los procesos anteriores que corresponden, por un lado a la bisección (5.4.6) y, por otro lado, a las cajas viables generadas por la función de *acotación del nodo utilizando la parte viable* (5.1.2) si es que se hubiera ejecutado y hubieran producido dichas cajas. Una vez procesadas, se incluirían en la lista de nodos vivos en el orden que les correspondiera.

5.4.8. Retorno al bucle principal

Para terminar, y correspondiente al apartado **I** (❾). Si el algoritmo ha llegado a esta etapa significa que todavía no ha encontrado solución y que siguen quedando nodos en la LNV por explorar, por tanto, retorna al apartado *C* para continuar con la ejecución.

5.4.9. Pseudocódigo

Algoritmo 5.6: Pseudocódigo algoritmo *Martínez-Cervera (MC)*

Entradas: B_Ω y $\mathbf{z}$, caja inicial de búsqueda
Salidas: $\mathbf{z}$, caja con los parámetros del controlador calculados en sus valores óptimos, o mensaje de que "No existe solución válida en x"

❶ ***A***. Comprobar factibilidad de la caja inicial de búsqueda:
$\mathbf{z} \leftarrow \mathbf{x}$ (Inicializar la *caja actual* con la caja inicial).
$\mathbf{R}_{\mathbf{z}\Omega} \leftarrow (mag(\mathbf{z},\omega), ang(\mathbf{z},\omega)), \forall\omega \in \Omega$
$[flag_z, B_{nm\Omega}, B_{nf\Omega}] \leftarrow FT(\mathbf{R}_{\mathbf{z}\Omega}, B_\Omega)$ // Llamada a la función de factibilidad (3.1.1)
Si $flag_z = inviable$, IMPRIMIR "No existe solución viable para $\mathbf{x}$" y SALIR
$\mathbf{ez} \leftarrow Inicial$ // Se marca como inicial la etapa de la caja.
$\mathbf{r} \leftarrow True$ // Los procedimientos de reducción de caja y búsqueda de soluciones están activados

❷ ***B***. Inicializar lista ***NL*** y variable de poda C
$NL \leftarrow \{(\mathbf{z}, z, flag_z, \mathbf{ez}, \mathbf{r})\}$ // Inicialización de la lista de nodos vivos
$C \leftarrow \infty$ // Inicialización de la variable de poda a infinito

❸ 16 ***C***. Poda y reducción de cajas usando la información de C:
17 ***Si NL*** es vacía, IMPRIMIR "No existe solución viable para $\mathbf{x}$" y SALIR
18 $\mathbf{z} \leftarrow NL[1]$ // Se recupera el primer nodo
19 ***Si*** $C < z$, // Se usa $<$ y no $\leq$ para evitar la poda de nodos viables
20 ***Descartar*** $\mathbf{z}$
21 Ir a la etapa ***C***
22 ***Sino*** //Reducir $\mathbf{z}$ usando la información de C
23 $K \leftarrow [z, C]$
24 $\mathbf{y} \leftarrow subs(\mathbf{z}, 1, K)$
25 $\{(\mathbf{z}, z, flag_z, \mathbf{ez}, \mathbf{r})\} \leftarrow \mathbf{y}$ // Actualizar $\mathbf{z}$ con la nueva caja reducida
26 ***Finsi***
27
❹ 28 ***D***. Comprobar factibilidad de la caja actual
29 $\mathbf{R}_{\mathbf{z}\Omega} \leftarrow (mag(\mathbf{z}, \omega), ang(\mathbf{z}, \omega)), \forall \omega \in \Omega$
30 $[flag_z, B_{nm\Omega}, B_{nf\Omega}] \leftarrow FT(\mathbf{R}_{\mathbf{z}\Omega}, B_\Omega)$ // Llamada a la función de factibilidad (3.1.1)
31 ***Si*** $flag_z = inviable$
32 ***Descartar*** $\mathbf{z}$ // Si es inviable la caja se descarta
33 Ir a la etapa ***C***.
34 ***Finsi***
35
❺ 36 ***E***. Cambio de etapa
37 ***Si*** $\mathbf{z}$ es menor en fase que el ancho del plano de Nichols
38 $\mathbf{ez} \leftarrow$ Intermedia
39 ***Sino Si*** $\mathbf{z} < (mag(\mathbf{z}, \omega)\ o\ \mathbf{z} < ang(\mathbf{z}, \omega)), \forall \omega \in \Omega$
40 $\mathbf{ez} \leftarrow$ Final
41 $\mathbf{r} \leftarrow false$ // Los procedimientos de reducción de caja y búsqueda de soluciones están desactivados
42 ***Finsi***
43
44
❻ 45 ***F***. ***B***úsqueda de soluciones y reducción del nodo
46 ***Si*** $\mathbf{r} = True$
47 $v \leftarrow MG(B_{xm\Omega}, \Omega, \mathbf{z})$ // Llamada a la función (5.3)
48 ***Si*** $v < C$ // Si la solución encontrada es mejor que la actual se sustituye
49 $C \leftarrow v$
50 ***Finsi***
51 ***Si*** $v = \varnothing$
52 $[\mathbf{z}, \mathbf{UM}, \mathbf{UF}, \mathbf{MM}, \mathbf{MF}] \leftarrow QSFact(B_{xm\Omega}, B_{xf\Omega}, \Omega, \mathbf{z})$ // Llamada a la función (5.2)
53 ***Finsi***
54
55 $\mathbf{z} \leftarrow QSInv(B_{nm\Omega}, B_{nf\Omega}, \Omega, \mathbf{z})$ // // Llamada a la función (5.1)
56 ***Finsi***
57
❼ 58 ***G***. Bisección de la caja
59 ***Si*** $\mathbf{ez} = Inicial$
60 $[z_1, z_2] \leftarrow$ bisección por la variable del controlador que más implicación tenga en el área.
61 ***Sino Si*** $\mathbf{ez} = Intermedia$
62 $[z_1, z_2] \leftarrow BA(\mathbf{z}, \mathbf{MM}, \mathbf{MF})$ // Llamada a la función Bisección árbol (5.4)
63 ***Sino***
64 ***Si*** diámetro$(mag(\mathbf{z}, \omega))$ > diámetro$(ang(\mathbf{z}, \omega))$
65 $[z_1, z_2] \leftarrow$ bisección por la variable del controlador con más contribución en magnitud
66 ***Sino***
67 $[z_1, z_2] \leftarrow$ bisección por la variable del controlador con más contribución en fase
68 ***Finsi***
69 ***Finsi***
70
❽ 71 ***H***. Procesar las cajas z^1, z^2, UM y UF, e insertar en ***NL***:
72 ***Para*** $j = 1, 2$:
73 ***Si*** $\mathbf{z^j} \neq \varnothing$,
74 $NL \leftarrow (\mathbf{z^j}, inf(\mathbf{z^j}[1], ambigua, \mathbf{ez}, \mathbf{r}))$
75 ***Finsi***
76 ***Finpara***
77 ***Para*** $j = UM[1, \ldots, n]$
78 ***Si*** $\mathbf{UM[j]} \neq \varnothing$,
79 $NL \leftarrow (\mathbf{UM[j]}, inf(\mathbf{UM[j]}[1], viable, \mathbf{ez}, \mathbf{r}))$
80 ***Finsi***
81 ***Finpara***

```
82    Para j = UF[1, ..., n]
83      Si UF[j] ≠ ∅,
84        NL ← (UF[j], inf(UF[j][1], viable, ez, r))
85      Finsi
86    Finpara
87
88    I. Ir a la etapa C
```

Capítulo 6

Casos prácticos

6.1. Introducción

En este capítulo, con el objetivo de demostrar el funcionamiento del algoritmo propuesto (5), se van a resolver dos casos prácticos diferentes. Las pruebas realizadas incluyen comparar su funcionamiento con los algoritmos NT (3.1) y NK (3.2) expuestos en el capítulo (3), junto con el algoritmo MC-prev, descrito en (Martínez-Forte y Cervera [2021]). Para realizar esta tarea, se van a tomar diferentes mediciones de tiempos (pruebas de rendimiento) utilizando diferentes tamaños de problema con el objetivo de comprobar cómo responden los algoritmos conforme crece el tamaño del problema (número de parámetros del controlador) de cada caso práctico.

Por otro lado, ya centrados en el algoritmo propuesto en esta tesis, se van a medir los tiempos de ejecución para cada una de las estrategias explicadas en el capítulo (4). Estas pruebas se realizarán con el objetivo de comprobar cómo impacta en el tiempo de ejecución cada una de las estrategias de forma individual (cuando no existan restricciones o dependencias entre ellas), así como diferentes combinaciones interesantes entre ellas, con la intención de realizar un análisis más exhaustivo del algoritmo propuesto. El objetivo final del capítulo es realizar una análisis pormenorizado del comportamiento, tanto del algoritmo completo, como de las estrategias individuales, utilizando para ello ejemplos clásicos de la literatura de QFT y comparando estos resultados con los algoritmos del mismo tipo predecesores.

Para cada uno de los casos prácticos, se calcularán todas las etapas del proceso de diseño de un controlador mediante QFT utilizando el software implementado explicado en el capítulo (7). De esta forma, se expondrá el desarrollo completo para resolver un problema aplicando QFT y utilizando el software desarrollado.

6.2. Descripción de las pruebas de rendimiento

Para cada uno de los casos prácticos que se desarrollan en las siguientes secciones, se harán dos comparaciones fundamentales a partir de las pruebas de rendimiento:

A) En primer lugar, el algoritmo planteado en esta tesis (5) se compara con los algoritmos previos NT (3.1), NK (3.2) y MC-prev (Martínez-Forte y Cervera [2021]). Para cada uno de los casos prácticos, se propondrá una estructura de controlador con polos y ceros reales que se inicializará con una sola variable y que irá incrementado el número de estas para cada prueba de rendimiento. Las mediciones terminarán para cada algoritmo cuando el tiempo de ejecución supere un minuto de duración. Se ha utilizado este límite buscando un equilibrio que permita un tamaño adecuado de los experimentos, suficiente para observar la diferencia entre algoritmos y, a la vez, permita que sean ágiles de realizar. De esta forma se podrá comprobar, para cada caso de prueba concreto, qué tamaño de problema, basado en el número de variables, es capaz de resolver cada algoritmo. Todas las pruebas ejecutadas resuelven el problema de optimización satisfactoriamente.

B) En segundo lugar, se va a estudiar cómo se comportan las diferentes estrategias propuestas en el Capítulo (4). Para ello, se han realizado diferentes pruebas y se han medido los tiempos de ejecución. Tanto de las diferentes estrategias de forma individual (siempre que las dependencias entre ellas lo permitan), como agrupadas por tipología. El algoritmo base utilizado, NK, es el propuesto en (3.2). A él se le añadirán las diferentes propuestas.

En el siguiente listado se muestran todas las estrategias individuales que se han utilizado para medir tiempos:

1. Acotación del espacio de búsqueda del subrango viable de las variables del controlador utilizando la magnitud (4.1.1).
2. Acotación del espacio de búsqueda del subrango inviable de las variables del controlador utilizando la fase (4.1.2).
3. Acotación del espacio de búsqueda del subrango viable de las variables del controlador utilizando la fase (4.1.1 y 4.1.2).
4. Búsqueda de mejor ganancia de alta frecuencia (4.3).
5. Bisección avanzada del nodo (4.2).
6. Etapas de ejecución e historia del nodo (4.4).

A continuación, se enumeran y describen las diferentes combinaciones de estrategias estudiadas:

1. Combina las estrategias (1 y 3) para medir de forma conjunta cómo afecta la acotación del espacio de búsqueda del subrango viable de las variables del controlador.

2. Implementa la estrategia (2) para medir de qué forma afecta la utilización de la fase en la acotación del espacio de búsqueda del subrango inviable de las variables del controlador.

3. Combina las estrategias (1, 2 y 3) midiendo así cómo afectan al tiempo de ejecución todas las estrategias de acotación del espacio de búsqueda de forma conjunta.

4. Mide el desempeño de la estrategia de *Búsqueda mejor ganancia* (4).

5. Combina las estrategias (1, 2 y 3, junto con 5), dado que esta última es dependiente de las anteriores para poder ejecutarse. Y tiene por objetivo medir el impacto en el tiempo de ejecución de activar esta estrategia.

6. Combina las estrategias (1, 2 y 5), con la idea de comprobar la diferencia de tiempo de ejecución de estas tres mejoras sin activar (3). Dado que esta última, al utilizar la fase, y teniendo el primero de los casos prácticos casi todas las fronteras abiertas, es interesante medir si realmente aporta a mejorar el tiempo de ejecución o no.

7. Combina las estrategias (1, 2, 4 y 5), con la idea de comprobar la diferencia de tiempo de ejecución de estas cuatro mejoras sin activar (3).

8. Combina las estrategias (1, 2, 3, 5 y 4), que son todas las estrategias desarrolladas menos (6), es decir, el algoritmo casi completo menos esta última funcionalidad indicada.

A continuación, enumeradas de la 9 a la 16, se repiten todas las combinaciones mencionadas anteriormente pero añadiendo la estrategia (6). Por tanto, la número 16 será el algoritmo completo. Se han planteado así las pruebas, dado que dicha estrategia tiene un funcionamiento global que afecta al funcionamiento base del algoritmo y a su organización. Por tanto, se quería estudiar cómo interaccionaba con el resto de estrategias que tienen un carácter local, es decir, actúan sólo en algunas partes concretas del algoritmo.

Siguiendo con lo explicado y, para que queden listadas para su posterior uso, se enumeran a continuación cada una de ellas:

9 Combina las estrategias (1, 3 y 6).

10 Combina las estrategias (2 y 6).

11 Combina las estrategias (1, 2, 3 y 6).

12 Combina las estrategias (4 y 6).

13 Combina las estrategias (1, 2, 3, 5 y 6).

14 Combina las estrategias (1, 2, 5 y 6).

15 Combina las estrategias (1, 2, 4, 5 y 6).

16 Combina las estrategias (1, 2, 3, 4, 5 y 6), que suponen el algoritmo completo.

Una vez presentadas la configuración de las distintas pruebas de rendimiento que se van a utilizar con los casos prácticos, se procede con las secciones correspondientes a cada uno de ellos.

6.3. Ejemplo de diseño 2 de Matlab QFT Toolbox

Este caso práctico es descrito en el Matlab® QFT Toolbox (Borghesani et al. [2003]) (de ahora en adelante QFT Toolbox para abreviar) como el ejemplo de diseño número 2. Es un problema fácil de resolver a mano, usando QFT Toolbox como herramienta de ayuda al diseño. Este caso práctico ilustra un problema clásico de restricción de seguimiento en QFT.

Este ejemplo ha sido también usado en diferentes publicaciones, por ejemplo, en (Chen et al. [1998], Raimúndez et al. [2001] y Nataraj y Tharewal [2004]).

6.3.1. Realización de las etapas de QFT

Modelado de la planta y su incertidumbre

A continuación se define la planta que se quiere controlar:

$$P(s) = \frac{ka}{s(s+a)}$$

Junto con la incertidumbre de dicha planta:

- $k \in [1, 10]$
- $a \in [1, 10]$

Definición de las restricciones de diseño

Se continúa con el siguiente paso, la definición de las restricciones de diseño, para el ejemplo que se está tratando son las siguientes:

(1) Seguimiento:

$$\alpha(\omega) \leq \left| F(j\omega) \frac{P(j\omega)C(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \leq \beta(\omega), \forall \omega \in \Omega \tag{6.1}$$

con

$$\alpha(\omega) = \left| \frac{0.6584(j\omega + 30)}{(j\omega)^2 + 4j\omega + 19.752} \right|$$

y

$$\beta(\omega) = \left| \frac{120}{(j\omega)^3 + 17(j\omega)^2 + 82j\omega + 120} \right|.$$

(2) Estabilidad:

$$\left| \frac{P(j\omega)C(j\omega)}{1 + P(j\omega)C(j\omega)} \right| \leq \gamma = 1.2, \ \forall P \in \mathrm{P}, \ \omega \in \mathbb{R}^+ \tag{6.2}$$

Elección de un conjunto de frecuencias de diseño

En la siguiente etapa, se elige el conjunto de frecuencias de diseño que se van a utilizar a lo largo de todas las etapas de QFT. En este caso, las seleccionadas son:

$$\Omega = \{0.1, 0.5, 1, 2, 15, 100\} \ rad/s$$

Cálculo de plantillas

Una vez descrita la planta, junto con su incertidumbre, y las frecuencias de diseño, se procede al cálculo de plantillas. Se creará una plantilla que refleja la planta nominal y su incertidumbre por cada frecuencia de diseño. En la figura (6.1) se puede observar el resultado.

Cálculo de fronteras

Con las restricciones de diseño, junto con las plantillas evaluadas en la etapa anterior, se calculan las fronteras, que definirán las regiones de aceptación-prohibición que deberá cumplir la planta junto con el controlador que se diseñe. A continuación, en la figura (6.2), se muestra el resultado obtenido.

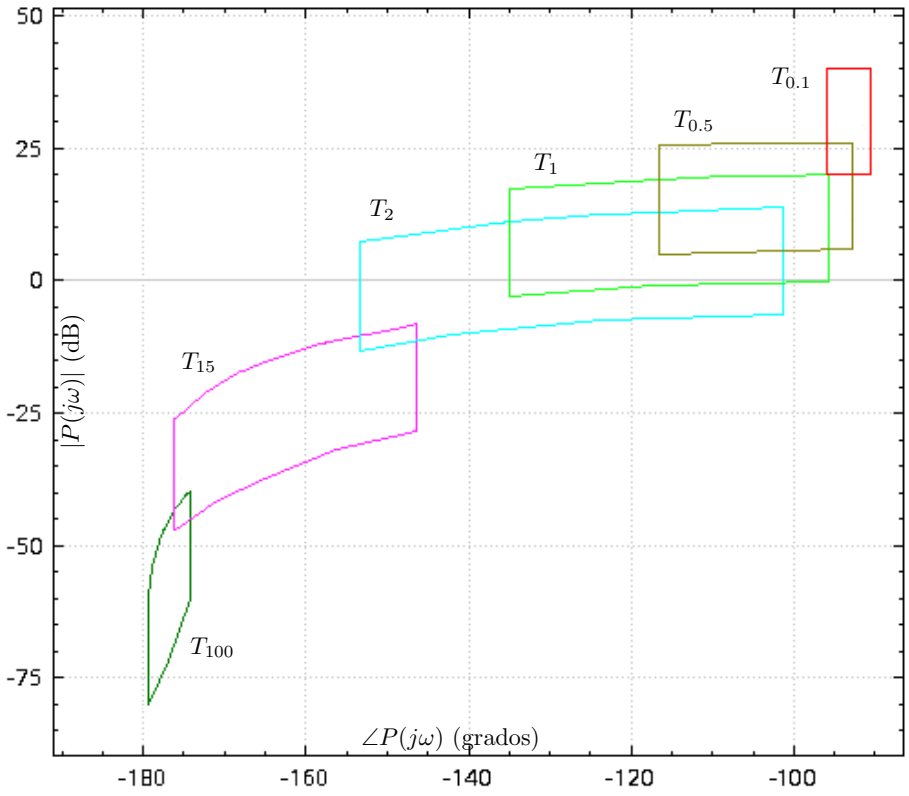

Figura 6.1: Plantillas calculadas del el caso práctico 1.

Ajuste del lazo

Una vez que se han calculado las fronteras, se dispone de toda la información necesaria para realizar el ajuste del lazo y, con ello, obtener el controlador óptimo, que respete la restricciones de diseño impuestas a la planta. A continuación, se muestra un ejemplo de controlador resultante con $n = 10$:

$$C_{QFTex2}(s) = 7 \cdot 10^6 \frac{(s+1.47)(s+1089)(s+1987.5)(s+3212.9)}{(s+575)(s+998)(s+1995)(s+2700)(s+3010)}$$

Para terminar se muestra el lazo resultante en el diagrama de Nichols junto con las fronteras en la figura (6.3).

6.3.2. Análisis de tiempos de ejecución

A) Comparación con otros algoritmos

En este apartado se van a detallar de forma pormenorizada los distintos tiempos de ejecución medidos en cada una de las pruebas realizadas en comparación con los algoritmos NT (3.1) y NK (3.2).

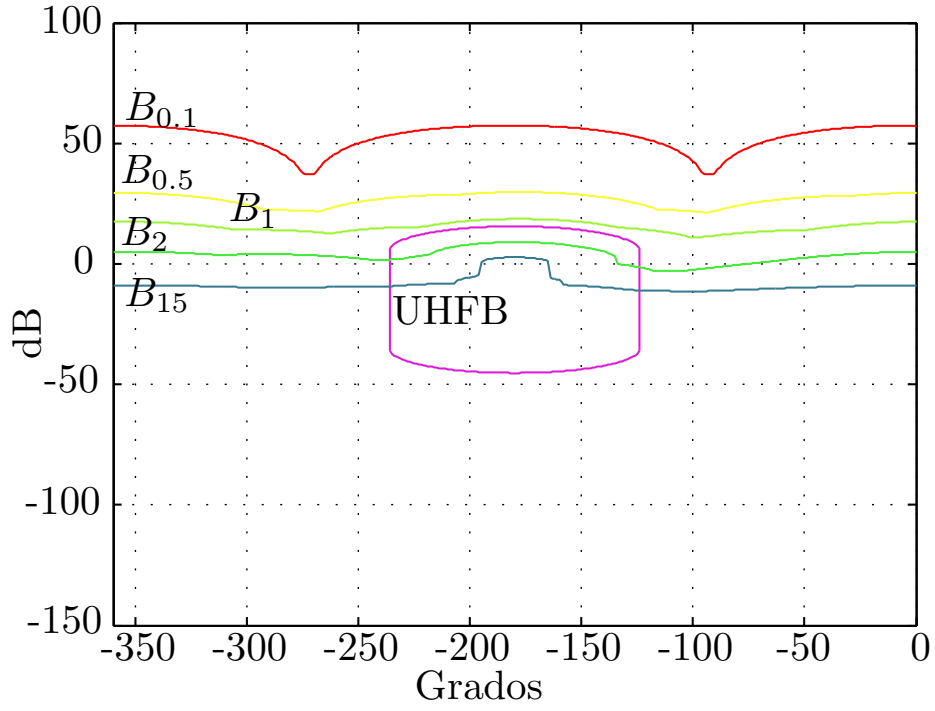

Figura 6.2: Problema diseño 2 de Matlab QFT Toolbox, fronteras B_ω.

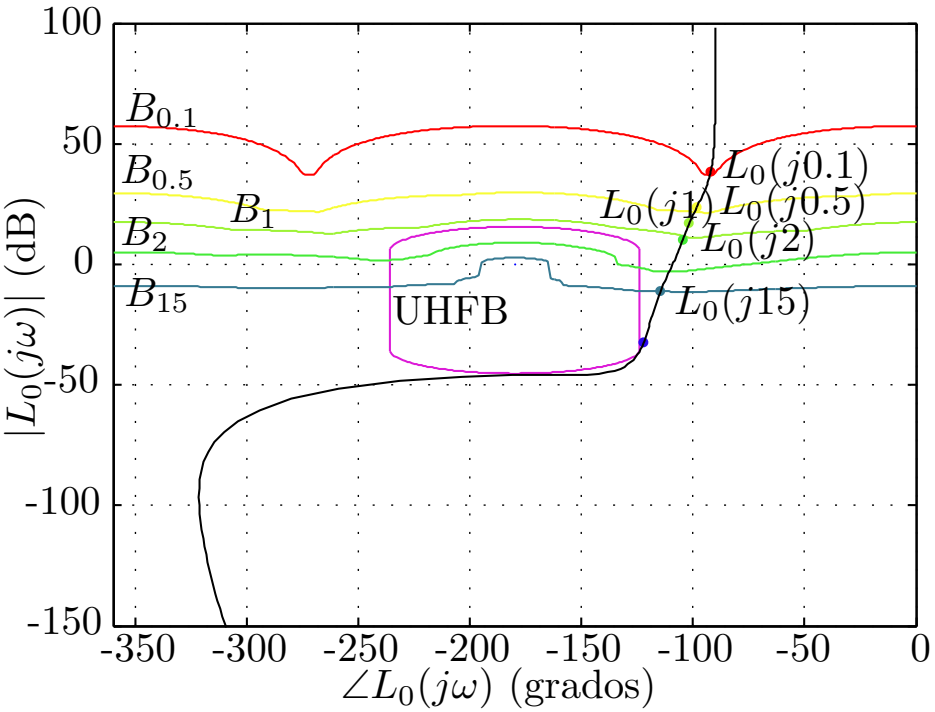

Figura 6.3: Problema diseño 2 de Matlab QFT Toolbox, fronteras B_ω, y ejemplo para $L_0(s)$ con solución obtenida con el algoritmo *MC*.

En primer lugar, en la tabla (6.1) se muestran los tiempos de ejecución

Tabla 6.1: Tiempo de ejecución (ms) para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 de Matlab QFT Toolbox.

Var \ Alg	NT	NK	MC-prev	MC
1	7	24	32	5
2	108	112	99	20
3	879	657	349	38
4	15 476	3897	2397	128
5	39 091	8880	2289	259
6	95 097	20 754	5009	157
7		47 890	12 098	538
8		97 745	24 789	368
9			50 490	890
10			101 087	1359
11				2515
12				9313
13				13 578
14				18 906
15				53 301
16				127 324

del algoritmo completo MC, que incluye todas las estrategias activadas, propuesto en (5) con los algoritmos NT (3.1), NK (3.2) y el algoritmo MC-prev, descrito en (Martínez-Forte y Cervera [2021]). Para complementar la información, en la figura (6.4) se muestran los mismos tiempos de ejecución en forma de gráfico de líneas para una mejor visualización de los mismos. Tal y como se indicó al inicio de este capítulo, no se han tomado la misma cantidad de mediciones para todos los algoritmos, sino que, se ha ajustado al tiempo indicado como límite para las pruebas.

Tabla 6.2: Coeficientes de la función de regresión para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 de Matlab QFT Toolbox.

Cof \ Alg	NT	NK	MC-prev	MC
α	2.01	14.22	24.82	5.52
β	1.94	1.18	0.86	0.59

Para cuantificar de forma precisa cuál es la diferencia entre los tiempos de ejecución de los distintos algoritmos, se muestran los coeficientes de la función de regresión exponencial $y(x) = \alpha + \beta^n$, calculando el valor de α y β para cada uno de de ellos. Los resultados se pueden visualizar en la tabla

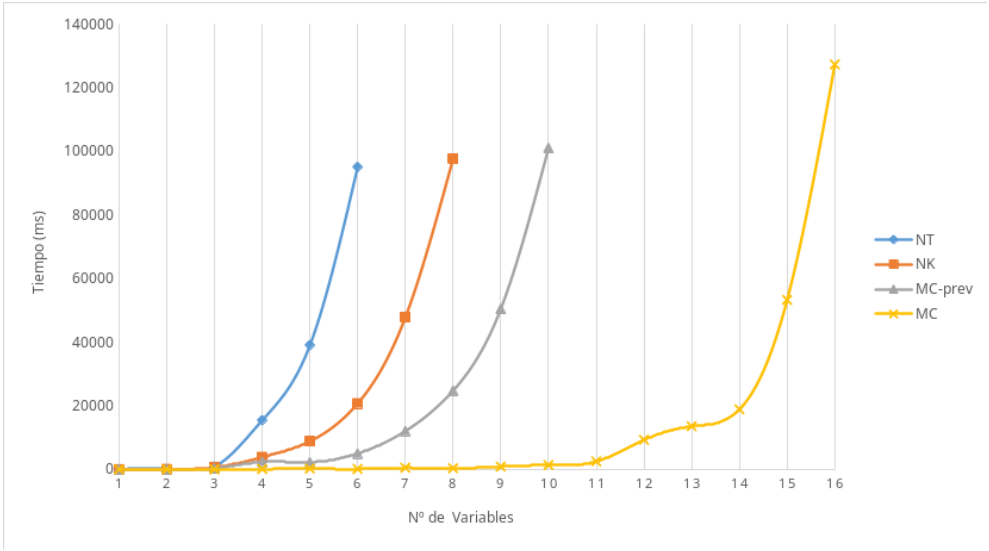

Figura 6.4: Tiempo de ejecución (ms) para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 del Matlab QFT Toolbox.

(6.2).

Como se puede observar en los datos mostrados, los tiempos de ejecución para el algoritmo propuesto mejoran de forma considerable los resultados obtenidos por los algoritmos previos similares. En la sección (6.5) se hará un análisis detallado de los resultados.

B) Comparación de diferentes estrategias

En este apartado se ven los resultados de cada una de las pruebas realizadas para cada una de las estrategias según la configuración expuesta al principio de este capítulo (6.2). Una vez conocidas las distintas combinaciones, los resultados se han divido en dos tablas para mayor comodidad de lectura. En la tabla (6.3) se muestran los resultados de tiempo obtenidos para las 8 primeras pruebas, y en la tabla (6.4) para las 8 pruebas restantes. Tal y como se indicó al inicio de este capítulo, no se han tomado la misma cantidad de mediciones para todas las combinaciones, sino que, se ha ajustado al tiempo indicado como límite para las pruebas.

La figura (6.5) muestra gráficamente los datos de las tablas (6.3 y 6.4).

6.4. Ejemplo de diseño ACC '90 benchmark

Este caso práctico es descrito de forma amplia en la literatura sobre QFT. Es un problema propuesto en (Wie y Bernstein [1990]) que se ha usado de forma recurrente para realizar ajustes del lazo con distintas técnicas, por

Tabla 6.3: Tiempo de ejecución (ms) para las 8 primeras combinaciones, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 de Matlab QFT Toolbox.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	4	3	5	3	5	4	5	4
2	15	20	20	21	28	23	21	16
3	35	31	36	61	37	58	34	54
4	293	1862	359	259	114	71	72	127
5	664	16 598	808	4827	378	158	185	418
6	519	44 847	664	10 723	132	466	459	138
7	667	111 345	821	35 218	1961	1318	1331	1636
8	1059		1321	100 989	1671	1367	1246	2213
9	2530		3045		3781	6249	5946	3144
10	5423		6694		6786	9377	5397	7234
11	22 195		19 457		12 315	24 587	13 655	11 931
12	40 520		29 005		25 854	36 868	32 216	20 453
13	77 685		50 457		39 894	52 640	52 906	30 992
14			88 214		77 485	89 389	85 573	48 059
15								90 729
16								

Tabla 6.4: Tiempo de ejecución (ms) para las 8 últimas combinaciones, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 de Matlab QFT Toolbox.

	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	3	3	5	5	3	4	4	5
2	25	22	24	20	12	17	18	20
3	36	65	37	45	36	32	41	38
4	87	149	108	138	58	77	72	128
5	241	2017	237	1982	129	132	185	259
6	111	5401	134	5809	134	100	102	157
7	124	18 917	137	21 482	152	88	79	538
8	182	56 129	222	74 241	200	201	265	368
9	410	122 438	433		452	1107	434	869
10	721		859		799	551	1512	1359
11	2875		2236		2776	1626	1533	2515
12	9132		74 625		9448	4978	3245	9313
13	15 488		16 575		16 563	9490	9966	13 578
14	36 545		27 966		27 591	14 348	14 041	18 906
15	81 893		79 956		89 284	20 408	22 145	52 488
16						82 091	75 971	127 324

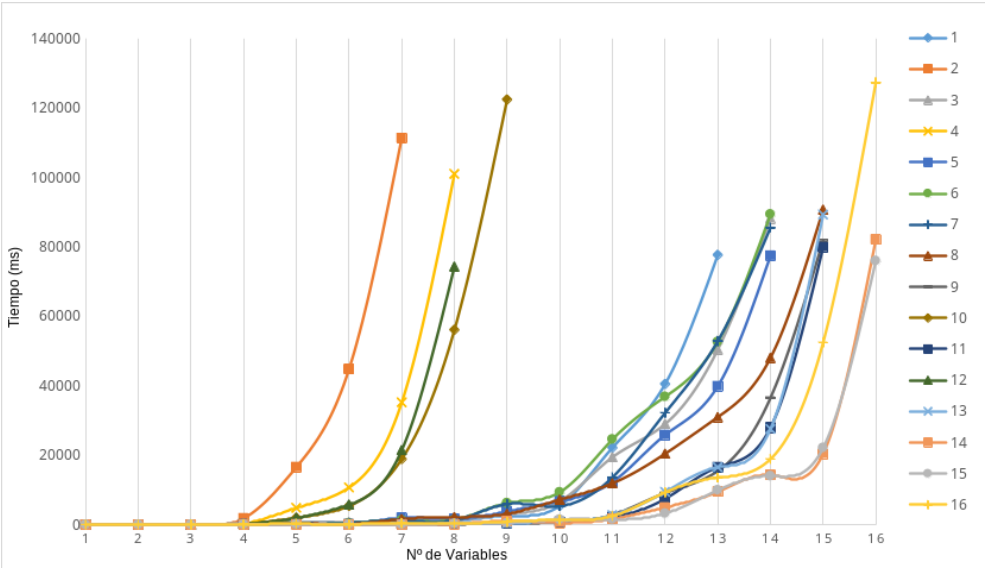

Figura 6.5: Tiempos de ejecución (ms) para cada combinación, con tamaño de problema $n = 1..16$, para el ejemplo de diseño 2 de Matlab QFT Toolbox.

ejemplo en (Horowitz [1993], Chiang y Safonov [2001] y Nataraj y Kubal [2006]).

6.4.1. Realización de las etapas de QFT

Modelado de la planta y su incertidumbre

A continuación, se define la planta que se quiere controlar:

$$P(s) = \frac{Y(s)}{U(s)} = \frac{e}{m_1 s^2 \left(m_2 s^2 + e \left(1 + \frac{m_1}{m_2} \right) \right)},$$

donde $m_1 = m_2 = 1$ y $e \in [0.5, 2]$.

Es un sistema masa-resorte no amortiguado (6.6). El objetivo de control es obtener una estabilidad robusta con un esfuerzo de control razonable. El tiempo de establecimiento de la respuesta al impulso debe de estar alrededor de $t_s = 15s$. La principal dificultad proviene del par de polos complejos no amortiguados en $s = \sqrt{-2e}$ porque conducen a un pico de resonancia de magnitud infinita en la frecuencia ω_p, dependiendo del valor de e. Para poder visualizar gráficamente $L_0(s)$ en un área finita del plano de Nichols, estos polos se considerarán ligeramente amortiguados.

$e_0 = 0.5$ se elije como valor nominal de e, lo que conduce a $\omega_p = 1\ rad/s$. Entonces, el conjunto de frecuencias de diseño Ω se elegirá con especial énfa-

sis entorno a esa frecuencia.

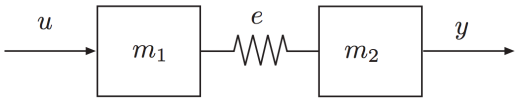

Figura 6.6: Problema ACC '90 benchmark

Definición de las restricciones de diseño

Se continúa con el siguiente paso, la definición de las restricciones de diseño, para el ejemplo que se está tratando son las siguientes:

(1) Estabilidad:

$$\left|\frac{P(j\omega)C(j\omega)}{1+P(j\omega)C(j\omega)}\right| \leq \gamma = 1.75,\ \forall P \in \mathrm{P},\ \omega \in \mathbb{R}^{+} \tag{6.3}$$

Elección de un conjunto de frecuencias de diseño

En la siguiente etapa, se elige el conjunto de frecuencias de diseño que se van a utilizar a lo largo de todas las etapas de QFT, como para este problema $\omega_p = 1\ rad/s$, las frecuencias se han elegido entorno a ella:

$$\Omega = \{0.1, 0.98, 0.99, 1, 2, 5, 7, 8.5, 10, 15, 20, 100\}\ rad/s$$

Cálculo de plantillas

Una vez descrita la planta, junto con su incertidumbre, y las frecuencias de diseño, se procede al cálculo de plantillas. Se creará una plantilla que refleja la planta nominal y su incertidumbre por cada frecuencia de diseño. En la figura (6.7) se puede observar el resultado.

Cálculo de fronteras

Con las restricciones de diseño, junto con las plantillas evaluadas en la etapa anterior, se calculan las fronteras, que definirán las regiones de aceptación-prohibición que deberá cumplir la planta junto con el controlador que se diseñe. A continuación, en la figura (6.8), se muestra el resultado obtenido.

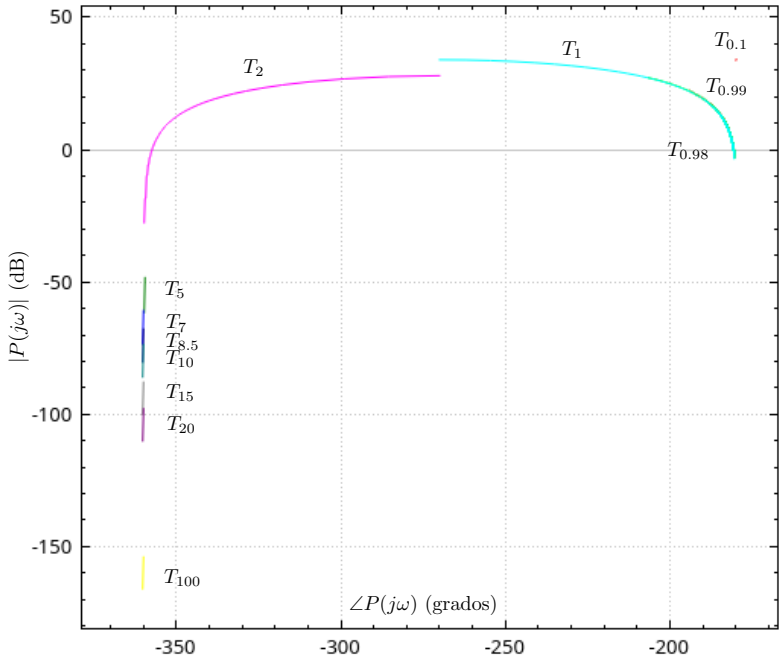

Figura 6.7: Plantillas calculadas para ACC '90 benchmark.

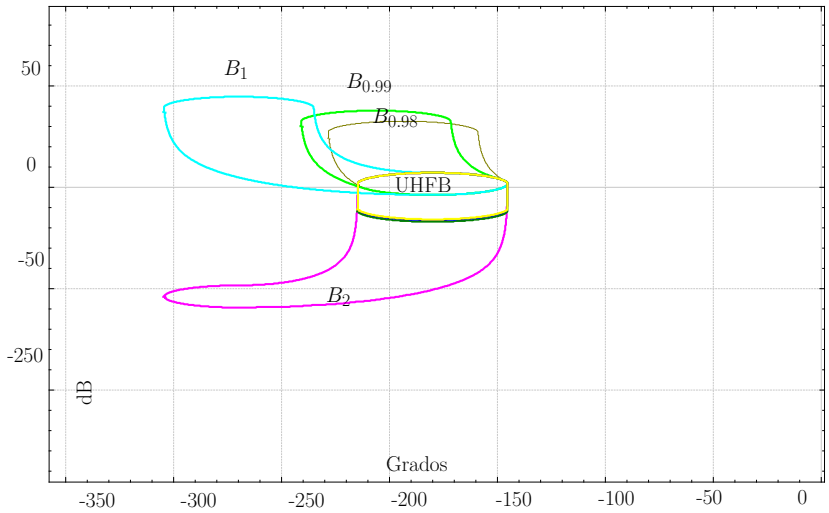

Figura 6.8: Problema ACC '90 benchmark, fronteras B_ω.

Ajuste del lazo

Una vez que se han calculado las fronteras, se dispone de toda la información necesaria para realizar el ajuste del lazo y, con ello, obtener el controlador óptimo que respete la restricciones de diseño impuestas a la planta. A continuación, se muestra un ejemplo de controlador resultante para $n = 12$:

$$C_{ACC90}(s) = 1.14{\cdot}10^7 \frac{(s+0.075)(s+0.35)(s+0.39)(s+25)(s+35)}{(s+7.2)(s+30)(s+39.8)(s+40)(s+95.9)(s+96.25)}$$

Para terminar se muestra el lazo resultante en el diagrama de Nichols junto con las fronteras en la figura (6.9).

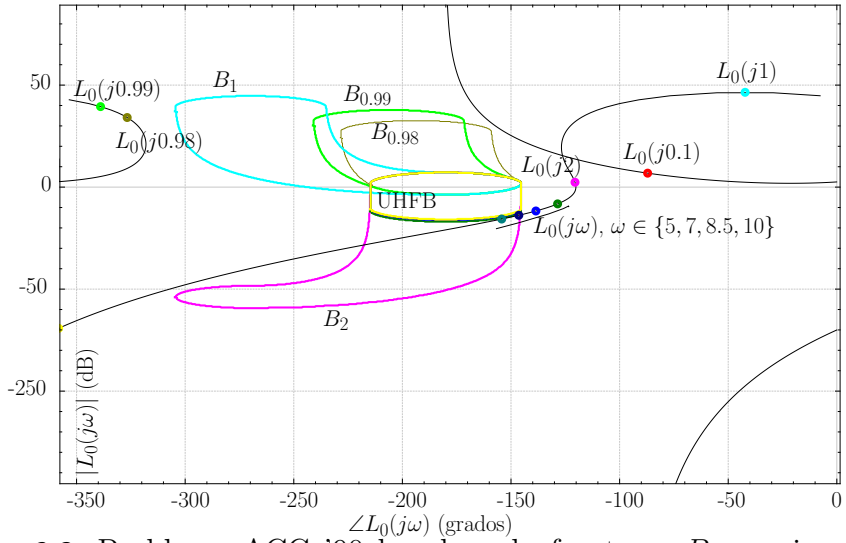

Figura 6.9: Problema ACC '90 benchmark, fronteras B_ω, y ejemplo para $L_0(s)$ con solución obtenida con el algoritmo *MC*.

6.4.2. Análisis de tiempos de ejecución

A) Comparación con otros algoritmos

En este apartado se van a detallar de forma pormenorizada los distintos tiempos de ejecución medidos en cada una de las pruebas realizadas en comparación con los algoritmos NT (3.1) y NK (3.2).

En primer lugar, en la tabla (6.5) se muestran los tiempos de ejecución del algoritmo completo MC, propuesto en (5) con los algoritmos NT (3.1), NK (3.2) y el algoritmo MC-prev (Martínez-Forte y Cervera [2021]). Para complementar la información, en la figura (6.10) se muestran los mismos tiempos de ejecución en forma de gráfico de líneas para facilitar una mejor visualización. Tal y como se indicó al inicio de este capítulo, no se han tomado la misma cantidad de mediciones para todos los algoritmos, sino que se ha ajustado al tiempo indicado como límite para las pruebas.

Para cuantificar de forma precisa cuál es la diferencia entre los tiempos de ejecución de los distintos algoritmos, se muestran los coeficientes de la función de regresión exponencial $y(x) = \alpha + \beta^n$, calculando el valor de α y β para cada uno de de ellos. Los resultados se pueden visualizar en la tabla (6.2).

Como se puede observar en los datos mostrados, los tiempos de ejecución para el algoritmo propuesto mejoran de forma considerable los resultados

Tabla 6.5: Tiempo de ejecución (ms) para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

Alg / Var	NT	NK	MC-prev	MC
1	31	24	14	16
2	264	34	158	47
3	468	37	158	71
4	9064	108	214	128
5	50 839	453	467	117
6	128 675	550	876	256
7		2785	1288	664
8		44 568	1346	943
9		97 890	4896	338
10			8547	1654
11			27 895	3857
12			64 730	9085
13			110 023	20 231
14				44 735
15				79 853
16				128 603

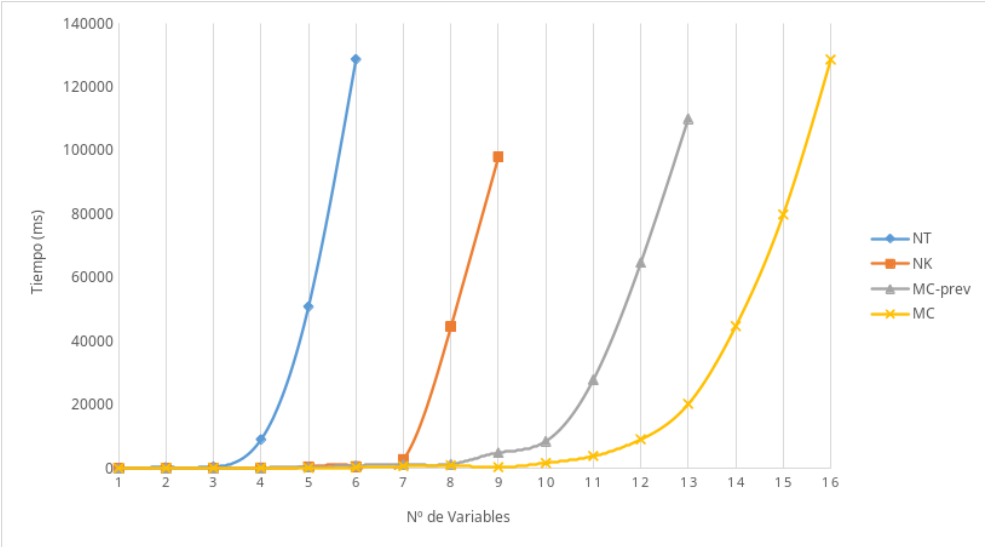

Figura 6.10: Tiempo de ejecución (ms) para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

obtenidos por los algoritmos previos similares. En la sección (6.5) se hará un análisis detallado de los resultados.

Tabla 6.6: Coeficientes de la función de regresión para cada algoritmo, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

Cof \ Alg	NT	NK	MC-prev	MC
α	5.88	2.65	16.15	8.17
β	1.72	1.08	0.65	0.58

B) Comparación de diferentes estrategias

En este apartado se ven los resultados de cada una de las pruebas realizadas para cada una de las estrategias según la configuración expuesta al principio de este capítulo (6.2). Una vez conocidas las distintas combinaciones, los resultados se han divido en dos tablas para mayor comodidad de lectura. En la tabla (6.7) se muestran los resultados de tiempo obtenidos para las 8 primeras pruebas y en la tabla (6.8) para las 8 pruebas restantes. Tal y como se indicó al inicio de este capítulo, no se han tomado la misma cantidad de mediciones para todas las combinaciones, sino que, se ha ajustado al tiempo indicado como límite para las pruebas.

Tabla 6.7: Tiempo de ejecución (ms) para las 8 primeras combinaciones, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	13	17	16	8	18	17	23	22
2	38	20	42	19	33	30	19	47
3	239	112	225	86	313	527	103	81
4	721	1257	660	101	967	445	115	168
5	992	3682	1128	1108	1448	351	571	388
6	2319	9112	4452	2183	3123	723	610	610
7	8884	18 762	8197	6784	4414	2988	1758	656
8	18 723	39 358	19 985	21 847	7963	5729	1296	701
9	47 955	88 480	35 495	55 370	11 352	11 618	5423	1721
10	87 623		70 921	93 868	27 107	20 076	9258	5686
11	141 749		122 416		60 934	35 528	12 763	10 082
12					112 867	58 723	34 972	19 313
13						91 670	66 540	51 296
14							125 854	86 520
15								132 952
16								

La figura (6.11) muestra gráficamente los datos de las tablas (6.7 y 6.8).

Tabla 6.8: Tiempo de ejecución (ms) para las 8 últimas combinaciones, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	15	14	14	15	14	12	17	16
2	15	8	24	16	18	12	35	47
3	121	104	231	87	239	113	52	71
4	342	908	546	72	526	152	82	80
5	577	1108	987	551	674	258	152	117
6	772	4925	1274	1831	980	506	276	256
7	1954	9328	2189	3680	1898	1515	610	664
8	4979	22 956	5108	9958	2009	2563	1052	943
9	10 008	61 428	10 087	25 783	2360	3564	762	338
10	24 176	112 008	30 762	54 629	3650	4860	2674	1654
11	46 762		80 006	134 267	12 158	15 923	10 453	3857
12	80 956		142 458		28 365	32 734	25 986	9085
13	142 479				56 954	60 067	53 749	20 231
14					92 307	94 692	89 967	44 735
15					132 907	135 900	131 872	79 853
16								128 603

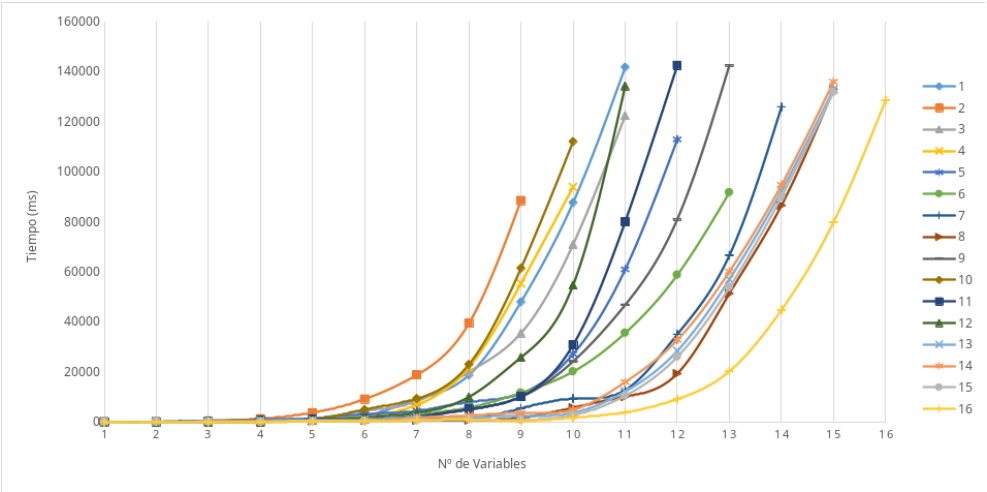

Figura 6.11: Tiempos de ejecución (ms) para cada combinación, con tamaño de problema $n = 1..16$, para el ejemplo de diseño ACC '90 benchmark.

6.5. Análisis de resultados

Una vez expuestos los datos de los dos casos prácticos, se llega a la conclusión de que el algoritmo MC (5), con respecto a los algoritmos NT (3.1), NK (3.2) y MC-prev (Martínez-Forte y Cervera [2021]), mejora los tiempos de ejecución de forma considerable. En la comparación de tiempos

que se realiza de estos algoritmos, se puede observar que el algoritmo MC consigue resolver el problema con hasta 16 variables antes de que el tiempo de ejecución sobrepase los dos minutos y medio , en comparación con el algoritmo NK, que consigue llegar hasta las 8 variables en el primer caso práctico y a 9 variables en el segundo. De esta forma se puede observar la mejoría de tiempo que aporta este nuevo algoritmo. Con respecto al resto de pruebas, en la sección (6.2) se detallan la configuración de las mismas y qué combinación de estrategias se ha realizado para cada una de ellas.

Centrados en el caso práctico 1, en la tabla (6.2) se pueden observar los coeficientes para la función de regresión exponencial de cada uno de los algoritmos. Para el algoritmo NK, su coeficiente β es 1.18, en cambio, para el algoritmo MC su coeficiente es 0.59, notándose de forma clara y cuantificada la mejora en el tiempo de ejecución.

A continuación, se comparan los tiempos de ejecución de las diferentes combinaciones de estrategias. Los resultados de la sección se pueden observar en (6.3, 6.4 y 6.5).

Objetivamos como la combinación número 16, que es el algoritmo al completo con todas las estrategias activadas, no es la que tiene el mejor tiempo de ejecución. Las pruebas 14 y 15, que se caracterizan por tener la acotación del espacio de búsqueda del algoritmo utilizando la fase en el subrango viable desactivado, obtienen mejores resultados. Esto se debe principalmente a dos cuestiones. La primera de ellas, como se ha explicado en (4.1.2), es la detección del subrango viable. Al tener que cumplirse esta condición para todas las frecuencias de diseño, es menos efectivo que cuando se trata de detectar el subrango inviable. La segunda se debe a emplear la acotación utilizando la fase. Ya que, dada la forma de las fronteras en este caso práctico (6.2), siendo todas abiertas menos la frontera de estabilidad y teniendo la mayoría muy poca variación en magnitud, dificulta realizar este tipo de acotación.

Por ello, los resultados de las mediciones de tiempo indican que desactivar esta estrategia mejora el rendimiento. Esto se debe a que el costo de realizar los cálculos necesarios y de aplicar dicha estrategia es mayor que la ganancia en tiempo de ejecución que pueda aportar. Este tipo de situaciones pueden ser muy comunes, dado que varias de las estrategias tienen una estrecha relación con cómo sea la interacción entre la caja proyectada y las fronteras en el plano de Nichols. Por lo que el resultado de aplicar dichas estrategias puede ser muy variable dependiendo del problema que se esté resolviendo.

Con respecto a la prueba nº 1, se puede observar que la estrategia de detección del subrango factible (4.1.1) de las variables del controlador es muy efectiva, consiguiendo por si sola llegar hasta las 13 variables. Siendo una

de las estrategias con mejor resultado al aplicarse de forma individual. Esta prueba incluye la detección usando la magnitud y la fase en el plano de Nichols. Tal y como se ha indicado en el párrafo anterior, la utilización de la fase no es muy efectiva para este caso práctico, por tanto, es la utilización de la magnitud la que proporciona la mejora de los tiempos de ejecución.

Del resto de estrategias implementadas, se puede observar como la *Búsqueda de mejor* k_{hf} (4.3) mejora el tiempo de ejecución de las pruebas, aunque tiene un impacto modesto. De hecho, en la prueba nº 5 se puede observar que si se activa de forma individual, el aporte es pequeño. De forma similar se comporta la estrategia *Bisección en forma de árbol* (4.2.4), pues tiene un impacto semejante. De estas dos estrategias se puede concluir que aportan una mejora en el tiempo de ejecución, pero su impacto es moderado para este ejemplo.

Con respecto a la estrategia de *Etapas del nodo* (4.4), como se puede observar en las pruebas de la 9 a la 16 y en comparación con las anteriores, su impacto es muy grande. Los tiempos de ejecución de todas las pruebas mejoran cuando esta estrategia está activa independientemente de como esté combinada con el resto de estrategias.

Del análisis de resultados de este caso práctico se puede concluir que todas las estrategias aportan mejoras en el tiempo de ejecución, salvo las que utilizan la fase en el plano de Nichols como base de su funcionamiento. Como ya se ha comentado, esta problemática surge por el tipo de fronteras que tiene este ejemplo. Lo que indica que el comportamiento de esta estrategia puede variar con otros casos prácticos que le fueran más favorables.

Con respecto al caso práctico 2, en la tabla (6.6) se pueden observar los coeficientes para la función de regresión exponencial de cada uno de los algoritmos. La diferencia entre el algoritmo NK, con coeficiente 1.08, con respecto al algoritmo MC, con coeficiente 0.58, es aproximadamente la mitad. Lo que cuantifica de forma clara la mejora en el tiempo de ejecución.

Para este caso práctico, el algoritmo completo (prueba nº 16) es el que tiene el mejor tiempo de ejecución, diferencia fundamental con respecto al caso práctico 1. Esto es debido a que, para este ejemplo, todas las fronteras son cerradas. De esta forma, cuando se aplica la acotación utilizando la fase en el plano de Nichols, su efectividad es similar a cuando se utiliza la magnitud.

Se puede observar como todas estas estrategias aportan de manera similar a la mejora del tiempo de ejecución, que es más parejo que en el caso práctico 1 y, conforme se van añadiendo más estrategias, éstas contribuyen a

mejorar el tiempo de ejecución de forma proporcional. Al igual que en el caso anterior, cabe destacar que la estrategia de *Etapas del nodo* (4.4), cuando está activa, mejora el rendimiento en todas las pruebas en las que participa. También es importante conocer que la *Búsqueda de mejor* k_{hf} (4.3) tiene un buen desempeño, incluso cuando está activa de forma individual, lo que contrasta con el caso práctico 1 donde tenía un impacto menor.

Resaltar que la estrategia de acotación de subrango factible de las variables (4.1.1) sigue siendo una de las estrategias principales. En este caso no destaca tanto con respecto a otras estrategias pero sigue siendo de las estrategias con mayor impacto en el tiempo de ejecución.

Como conclusión, se puede determinar que el algoritmo propuesto mejora de forma considerable los tiempos de ejecución en los dos casos prácticos. Con respecto al algoritmo NK, es capaz de resolver problemas con el doble de variables en el controlador en el caso práctico 1 y casi el doble en el caso práctico 2.

Con respecto a las distintas estrategias, en las pruebas individuales o en combinaciones, se ha podido observar que, dependiendo del tipo de problema a resolver, no todas las estrategias se comportan igual. Llegando a tener un impacto negativo en el tiempo de ejecución en los casos menos propicios. En el apartado de vías futuras (8.2), se plantean opciones para mejorar este comportamiento.

Capítulo 7

Software desarrollado

En este capítulo se describe el software implementado, durante un periodo que comienza antes de esta tesis doctoral. Este desarrollo empezó como alumno interno durante los estudios de ingeniería informática en la Universidad de Murcia, orientado a realizar el proyecto fin de carrera de dicha titulación. El objetivo de este proyecto fue crear un software libre que implementara algunas de las etapas de la técnica de control robusto QFT. Se planteó de esta forma porque todas las herramientas disponibles tenían licencia privativa y no eran un espacio abierto en el cual, cualquier investigador interesado, pudiera colaborar. Por esta limitación de licencia, los distintos desarrollos de software se limitaban a implementar los diseños propios de los desarrolladores y no estaban abiertos a las nuevas contribuciones más novedosas que se estaban realizando.

Por ejemplo, dos de los paquetes más utilizados, desarrollados como librería de Matlab® son el QFT Toolbox (Borghesani et al. [2003]) y QFTCT (Garcia-Sanz [2008-present]). En el caso de QFT Toolbox implementa el cálculo automático de fronteras pero no ofrece ningún algoritmo para las plantillas. Y en cuanto a QFTCT, sí que ofrece algoritmos para las etapas de cálculo de plantillas y de fronteras. Con respecto al ajuste del lazo, son lo que se conoce como un *software de ayuda a la decisión*, es decir, no implementan algoritmos que resuelvan el ajuste del lazo, si no, que ayudan al diseñador con herramientas informáticas que facilitan el trabajo. Este tipo de herramientas son útiles para realizar el proceso de QFT pero tienen una finalidad distinta al software planteado en este capítulo.

A esto se une que, los dos son software privativo. No están a disposición de los investigadores, ni para su uso de forma gratuita, ni tienen licencia que permita acceder al código fuente para realizar nuevos desarrollos. El objetivo cuando se empezó esta aplicación fue crear un software libre a disposición de

la comunidad para que cualquiera pudiera utilizarlo, y sobretodo, que fuera posible modificarlo y adaptarlo a las necesidades de cada investigador.

7.1. Introducción

El primer proyecto implementado, como proyecto fin de carrera (Martínez-Forte y Cervera [2014]), incluía las primeras etapas de QFT hasta el cálculo de fronteras (1.1). De las etapas implementadas, destacan dos por el número de horas de trabajo que conllevaron. La primera de ellas fue el cálculo de plantillas, que incluía dos algoritmos diferenciados, uno para el cálculo propiamente dicho de las plantillas y el segundo, para el cálculo del contorno de las mismas. La segunda etapa fue el cálculo de fronteras, que incluía diversos algoritmos accesorios y el principal, que soportaba cálculos de fronteras multivaluadas (Moreno et al. [2006]); además de otros accesorios para complementar al principal. Se decidió realizar la implementación usando como lenguaje C++ (ISO [2012]) y como librería gráfica QT (QT et al. [2022]).

El segundo proyecto, continuación del primero, fue el desarrollado como trabajo fin de máster (Martínez-Forte y Cervera [2014]) y consistió en la paralelización de los principales algoritmos implementados en el proyecto fin de carrera, tanto en CPU (multicore) como en GPU. Los algoritmos estudiados fueron 3, cálculo de plantillas, *e-hull* (algoritmo para calcular el contorno de las plantillas) y cálculo de fronteras. Las decisiones de diseño fueron implementar el paralelismo en CPU usando OpenMP (OpenMP Architecture Review Board [2011]) y, para la paralelización en GPU, se usó CUDA (NVIDIA et al. [2014]).

El proyecto se decidió continuar como tesis doctoral, focalizando la atención en la última de las etapas fundamentales de QFT, el ajuste del lazo. Para ello, se decidió partir de los únicos algoritmos publicados que permiten obtener la solución óptima, los explicados en (3). Como parte del estudio del estado del arte y base de la que partir, además de para poder realizar pruebas de rendimiento posteriores, se decidió implementar los algoritmos NT (3.1) y NK (3.2), junto con el algoritmo NR (Rambabu Kalla y P. S. V. Nataraj [2010]), aunque éste no está mantenido y tiene un código no funcional con los últimos desarrollos realizados. La decisión de no continuar con el mantenimiento del algoritmo NR se debe principalmente a que se desvía de la rama de estudio principal que se planteo para realizar la tesis doctoral y, por tanto, su interés disminuyó en el contexto de este trabajo.

Una vez con el estudio del arte realizado en profundidad, este proyecto se convirtió en una parte fundamental de la tesis doctoral, al servir como base

para realizar todas las implementaciones y las pruebas de todas las ideas que han ido surgiendo durante la realización de la misma. Disponer de todo el sistema de QFT implementado ayudó de forma notable a facilitar cualquier tipo de prueba práctica que fuera necesaria, dado que al tener toda la base desarrollada, todo el esfuerzo podía ser dirigido a las nuevas ideas.

El software incluye los dos algoritmos desarrollados durante la tesis (MC-prev y MC). El primero de ellos incluye algunas de las mejoras planteadas (Martínez-Forte y Cervera [2021]). El segundo de los algoritmos incluye todas las estrategias propuestas en (4). En el capítulo (5) se dan todos los detalles sobre los cálculos necesarios para implementar las estrategias propuestas en el capítulo (4).

Con estos algoritmos implementados, en el proyecto se incluyen actualmente todas las etapas principales de QFT terminadas y funcionales. Por ende, se puede usar tanto por un usuario interesado en generar un controlador óptimo para una planta utilizando QFT (ver 7.3), como por un investigador cuyo interés sea usar el proyecto como base para sus investigaciones; tanto como soporte para sus pruebas, como para implementar nuevas mejoras y algoritmos en el proyecto. La aplicación está disponible como software libre. Para más detalles consultar (7.2).

7.2. Repositorio y licencia

Dada la intención de publicar la aplicación como software libre, dos de las cuestiones más importantes eran qué licencia elegir y dónde publicar el código.

La primera de las opciones se resolvió eligiendo la licencia *GNU GENERAL PUBLIC LICENSE Version 3*[1], que permite su distribución como software libre. Además, quien quiera utilizar el código y publicarlo, debe hacerlo bajo las mismas condiciones.

En cuanto a la segunda cuestión, el proyecto se ha alojado durante su desarrollo en github, utilizado como control de versiones. Y es en este lugar desde donde se puede descargar todo el código junto con las instrucciones de instalación (`https://github.com/Isaac-Martinez-Forte/QFTbx`). El repositorio no incluye versiones estables ni publicadas, simplemente recoge el código a medida que se va desarrollando.

[1] `https://www.gnu.org/licenses/gpl-3.0.html`

7.3. Manual de usuario

Como se ha comentado al inicio de este capítulo, el software desarrollado ofrece la posibilidad de realizar todas las etapas importantes del proceso de QFT. Faltan algunas etapas opcionales por implementar, pero es perfectamente funcional para resolver un problema y obtener la solución óptima. En esa sección se desarrolla un manual completo de la aplicación, con el objetivo de facilitar su uso a cualquier usuario que estuviera interesado. Todo el proceso se va a realizar utilizando el ejemplo 1 del *QFT toolbox* (Borghesani et al. [2003]) para mostrar así su funcionamiento.

La planta ejemplo viene dada por:

$$P(s) = \frac{k}{(s+a)(s+b)} \tag{7.1}$$

Se define la incertidumbre de la planta como:

$k \in [1, 10]$ Nominal $= 1$

$a \in [1, 5]$ Nominal $= 5$

$b \in [20, 30]$ Nominal $= 30$

Las frecuencias de diseño elegidas en el toolbox son:

$$\Omega = \{0.1, 5, 10, 50, 100\}\ rad/s$$

7.3.1. Pantalla principal

La pantalla principal de la aplicación se plantea con un diseño muy sencillo que de acceso directo a las etapas del proceso de diseño QFT. Así como a algunas opciones extra para mostrar gráficos y guardar los datos introducidos y calculados. Para cada una de las etapas, al pulsar su botón correspondiente, se abrirá en una nueva ventana independiente, permitiendo así tener disponibles varias a la vez y poder consultar los datos calculados de varias etapas.

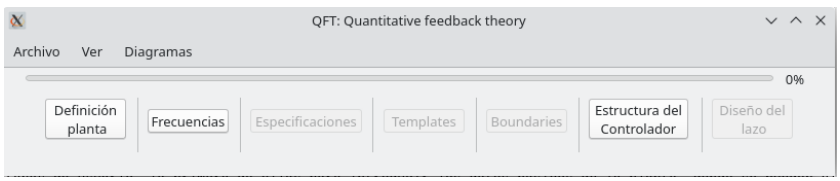

Figura 7.1: Pantalla principal de la aplicación.

7.3.2. Introducir la planta

Para esta etapa, el software dispone de dos ventanas de introducción de datos. En la primera pantalla (7.2), se elige la estructura de la planta a introducir de entre las opciones disponibles. Una vez elegida, es el momento de definir el numerador y denominador, que pueden ser tanto variables como valores concretos para los polos y los ceros. Si fueran variables, se deberán registrar con una letra. De esta forma, posteriormente se podrá indicar la incertidumbre de cada una de ellas. Para la ganancia y para el retardo, se deberá indicar cuál es su valor nominal. En la siguiente pantalla, si nos interesa, se definirá su incertidumbre asociada.

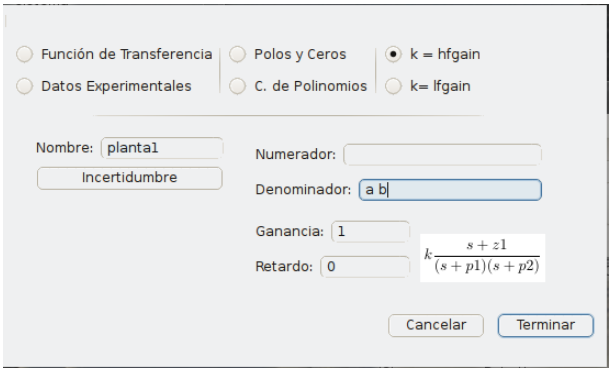

Figura 7.2: Pantalla de introducción de la planta en la aplicación.

Una vez cumplimentada, se pasará a una segunda pantalla (7.3). Que tiene como finalidad introducir la incertidumbre asociada de las variables definidas en la primera pantalla, incluida la ganancia y el retardo. Por ahora, sólo está implementada la opción de introducir el rango de las variables manualmente, pero en un futuro se podrán desarrollar otras opciones.

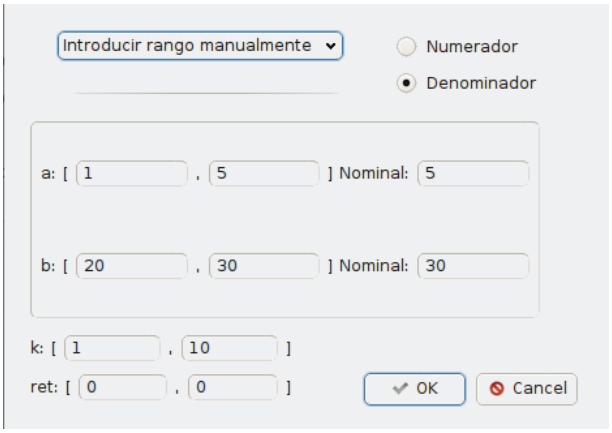

Figura 7.3: Pantalla de introducción de la incertidumbre de la planta en la aplicación.

7.3.3. Introducir las frecuencias de diseño

El siguiente paso es introducir las frecuencias de diseño que se van a utilizar durante todas las etapas del proceso de diseño QFT. Existen varias opciones disponibles: la primera opción es introducir los valores manualmente, como se muestra en la figura (7.4). La segunda y tercera opción dan la posibilidad de que los valores sean generados utilizando un espaciado lineal o logarítmico, como se muestra en la figura (7.5). Y para terminar, existe la opción de que los valores sean leídos desde un fichero.

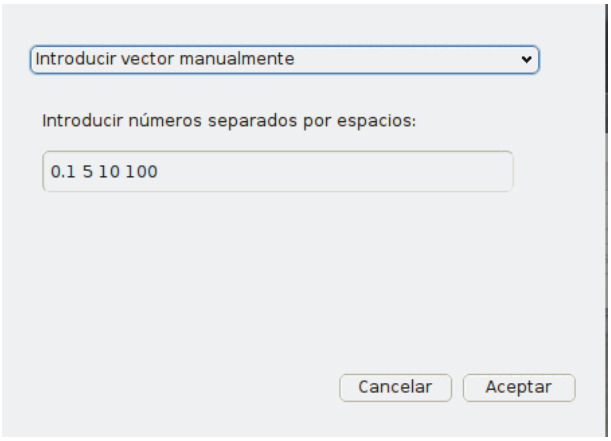

Figura 7.4: Pantalla de introducción manual de las frecuencias de diseño en la aplicación.

Figura 7.5: Pantalla de introducción con espaciado logarítmico de las frecuencias de diseño en la aplicación.

7.3.4. Cálculo de plantillas

La etapa del cálculo de plantillas se realiza por parte de dos algoritmos. Para iniciarlos, es necesario introducir unos datos básicos, tal y como se mues-

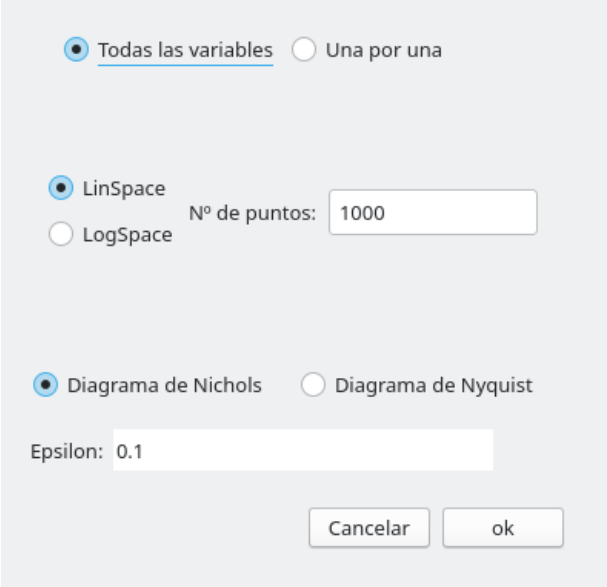

Figura 7.6: Pantalla de introducir datos para el algoritmo de cálculo de plantillas en la aplicación, parte 1.

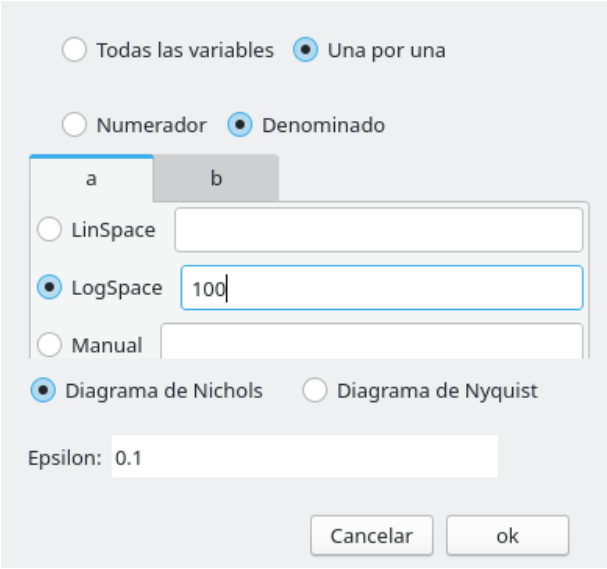

Figura 7.7: Pantalla de introducir datos para el algoritmo de cálculo de plantillas en la aplicación, parte 2.

tra en la figura (7.6). En primer lugar hay que definir el número de puntos en los que se va a discretizar cada variable. Se puede elegir un valor de forma global para todas ellas, tal y como se muestra en la figura (7.6). O se puede elegir un número de puntos distinto para cada variable de la planta, como se muestra en la figura (7.7). Para terminar será necesario indicar el tamaño que se requiere para ϵ (épsilon) parámetro necesario para el algoritmo *e-hull* para calcular el contorno de las plantillas.

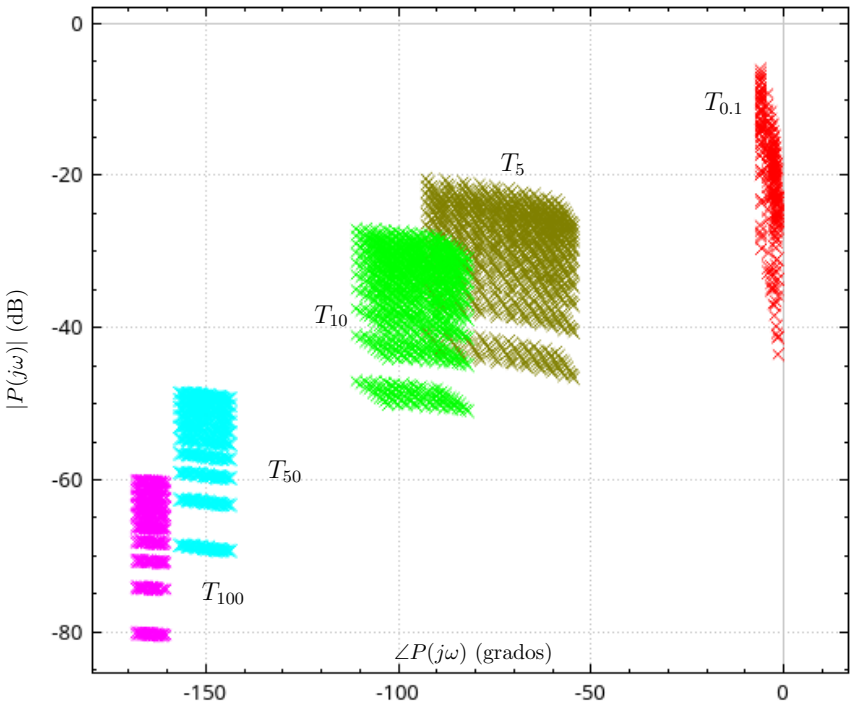

Figura 7.8: Diagrama de Nichols de las plantillas calculadas por la aplicación.

Una vez que los algoritmos han terminado, el resultado se puede mostrar para visualizar las plantillas completas en el diagrama de Nichols (7.8), o el contorno calculado de las plantillas, que se puede observar en la figura (7.9).

7.3.5. Introducir especificaciones de diseño

La siguiente etapa corresponde a la introducción de las especificaciones de diseño, donde se registrarán los datos necesarios, junto con las plantillas ya calculadas, para avanzar a la siguiente etapa. En la figura (7.10) se puede observar la pantalla para introducir las especificaciones de diseño.

En ella se pueden definir las distintas especificaciones de QFT (1.1.4), y a su vez, cada una de ellas, se pueden registrar de dos formas. Como primera opción, las que estén delimitadas por números, se pueden definir tanto en lineal como en decibelios. Como segunda opción, se pueden delimitar por funciones de transferencia. La primera opción se puede observar en la figura (figura 7.11) para una especificación de estabilidad. Y la segunda opción se puede observar en la figura (figura 7.12) para una especificación de seguimiento.

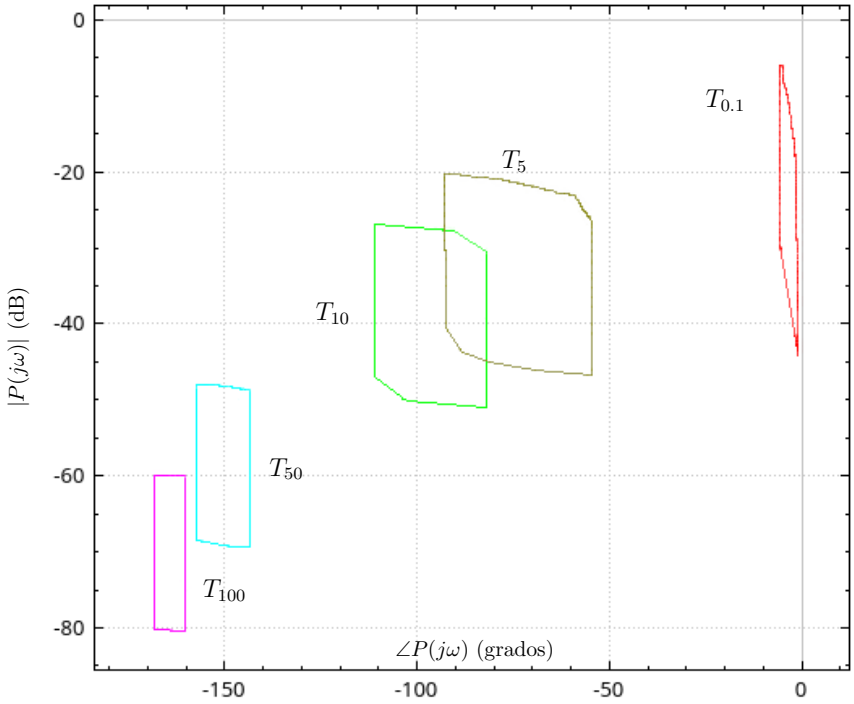

Figura 7.9: Diagrama de Nichols del contorno de las plantillas calculadas por la aplicación.

Figura 7.10: Pantalla de introducir datos de las especificaciones de diseño, parte 1.

Finalmente, se puede indicar, para cada una de las especificaciones, para qué frecuencias de diseño está activa dicha restricción, en el caso de que no sea para todas.

7.3.6. Cálculo de fronteras

El siguiente paso es el cálculo de fronteras. Donde se utilizarán los datos de las plantillas, junto con las especificaciones de diseño, para calcularlas. En primer lugar, es necesario introducir los datos básicos que necesitan los

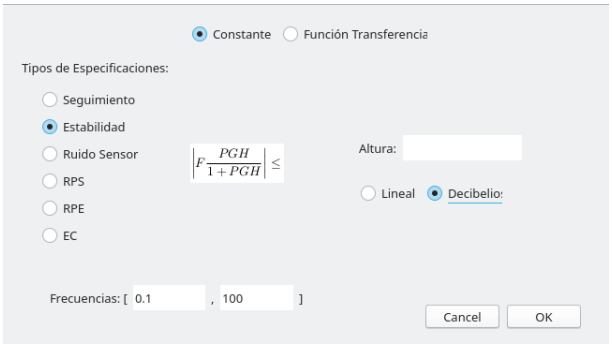

Figura 7.11: Pantalla de introducir datos de las especificaciones de diseño, parte 2.

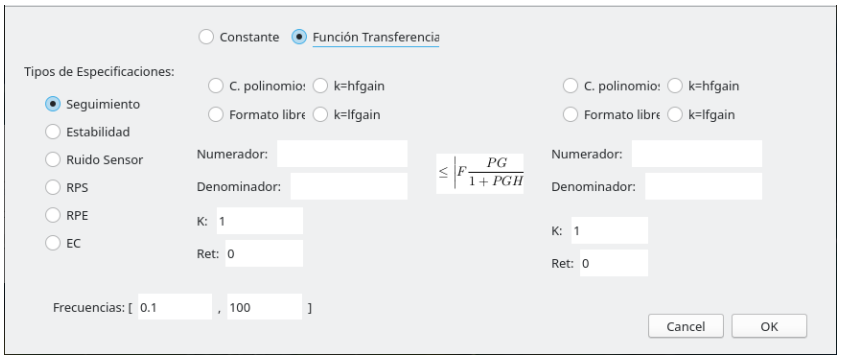

Figura 7.12: Pantalla de introducir datos de las especificaciones de diseño, parte 3.

diferentes algoritmos de esta etapa. Como se puede observar en la figura (7.13), inicialmente se proponen unos datos por defecto que son válidos para el algoritmo. Estos se pueden cambiar, por ejemplo, para indicar sobre qué fases y magnitudes del plano de Nichols se quieren realizar las operaciones, o qué precisión le requerimos al algoritmo. El siguiente de los datos a introducir es la altura de corte para la función *Contour*, necesaria para generar las fronteras bidimensionales en el plano de Nichols. También existe la opción de introducir un vector de alturas para que el algoritmo nos muestre el plano de Nichols con las diferente fronteras generadas para cada altura. Para terminar, como algunos datos generados por el algoritmo pueden tener el valor de infinito, y estos pueden ser exportados, por motivos de compatibilidad se le puede dar un valor concreto para que sustituya a infinito cuando éste se genere. Si no se necesita exportar los datos, este campo se puede dejar en blanco. En la figura (7.14), se puede observar el resultado una vez calculadas las fronteras.

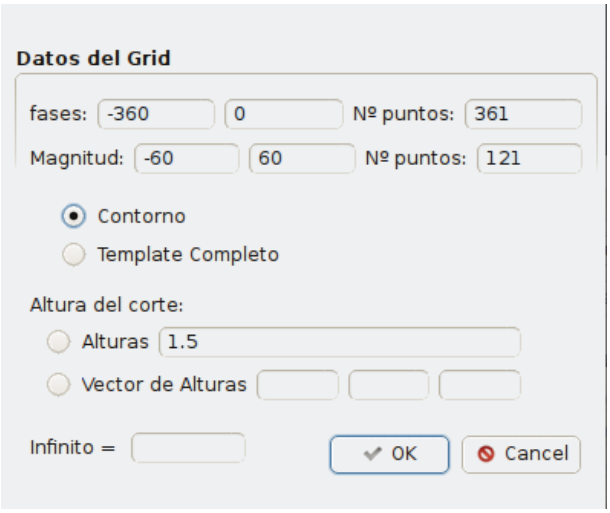

Figura 7.13: Pantalla de introducir datos para el algoritmo de cálculo de fronteras en la aplicación.

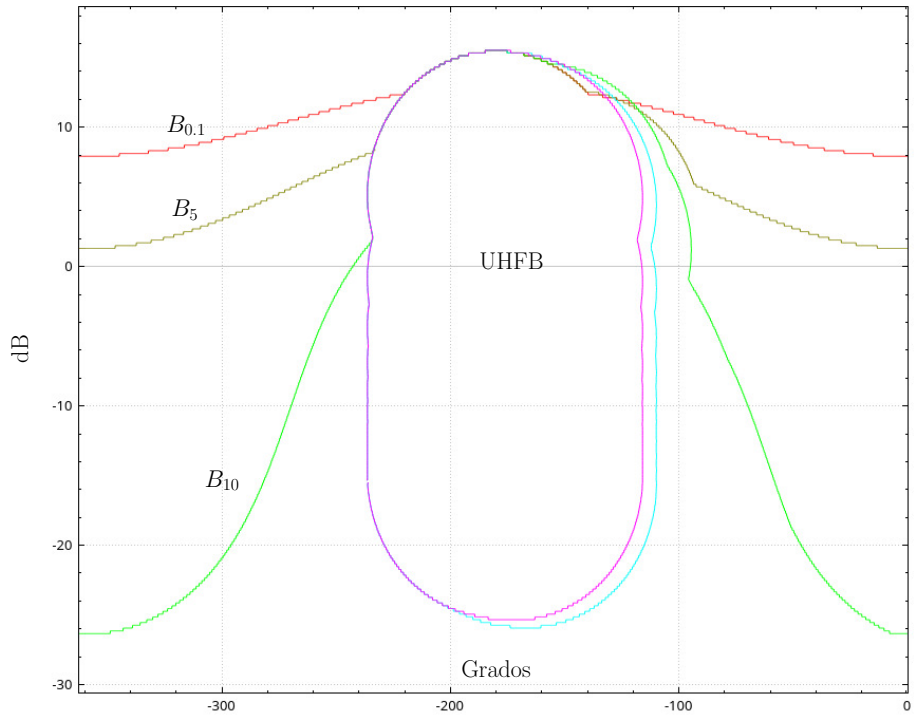

Figura 7.14: Diagrama de Nichols con las fronteras calculadas por la aplicación.

7.3.7. Ajuste del lazo

La última etapa implementada actualmente es el ajuste del lazo. La etapa principal de QFT donde se obtiene el controlador solución. En lo que se refiere a la ejecución de esta etapa, en primer lugar se inicia definiendo una estructura de controlador, tal y como se observa en la figura (7.15). Al igual

que en la etapa(7.3.2), consta de la misma pantalla y las mismas opciones.

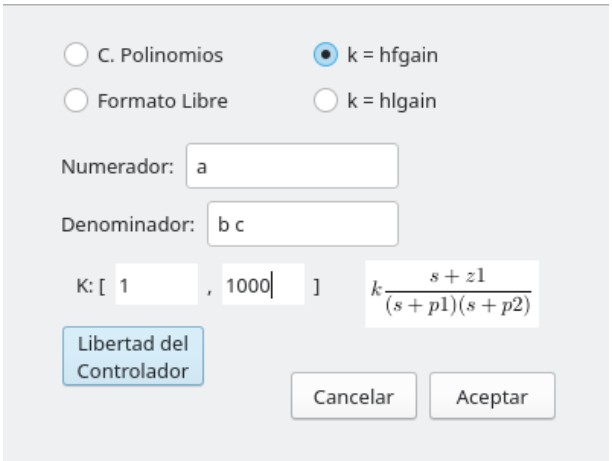

Figura 7.15: Pantalla de introducir los datos del controlador en la aplicación.

Una vez indicadas las variables del controlador, se deben introducir los rangos de dichas variables, tal y como se puede observar en la pantalla de la figura (7.16).

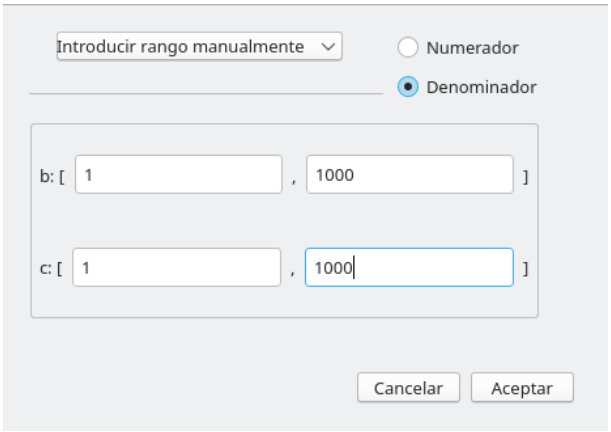

Figura 7.16: Pantalla de introducir los rangos de las variables del controlador en la aplicación.

A continuación, una vez definida la estructura del controlador, el siguiente paso es elegir el algoritmo a utilizar e introducir unos valores básicos, tal y como se puede observar en la figura (7.17).

Una vez terminada la configuración de la estructura del controlador y la elección del algoritmo por parte del usuario, se procederá a la ejecución del algoritmo elegido, si encuentra un resultado, se muestra una figura con el calculo realizado (7.18), así como el controlador resultante. Si no encontrara

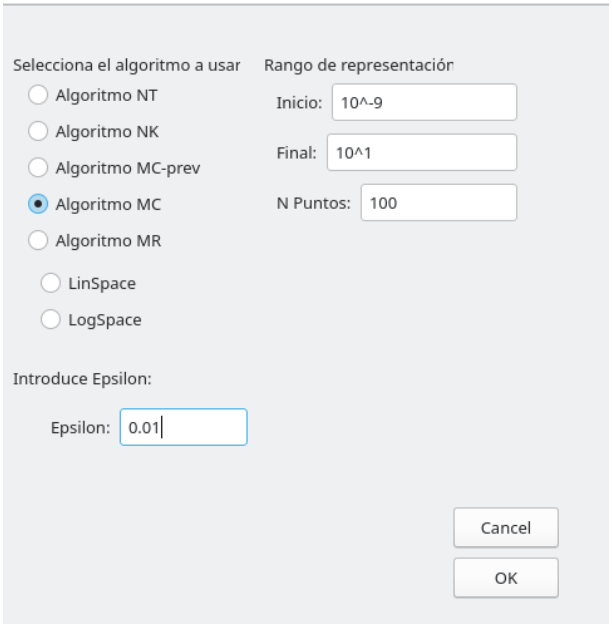

Figura 7.17: Pantalla de elección de algoritmo de ajuste del lazo en la aplicación.

resultado, mostraría un mensaje indicando el error encontrado.

Para este ejemplo el controlador resultante es el siguiente:

$$C_{QFTex1}(s) = 16.36\frac{(s+96.90)(s+93.81)}{(s+344.40)}$$

7.4. Estructura del proyecto

La aplicación está diseñada siguiendo técnicas estándar de desarrollo de software como UML (Rumbaugh et al. [2004] y Gamma et al. [1995]) que tienen asociado el uso de patrones de diseño, que simplifican la creación y gestión del código. Como base fundamental, el software está implementado usando el patrón modelo-vista-controlador, que propone la construcción del software separado en tres componentes distintos. Por un lado, estaría la vista, que es la parte del software encargada de la interacción con el usuario (u otros elementos clientes). Ésta puede ser una interfaz gráfica, una interfaz de comandos, etc, que se ponga a disposición de los clientes.

En segundo lugar, el componente modelo incluiría toda la lógica del negocio del software. De esta forma, todo el modelo del software quedaría agrupado en este componente. Para terminar, el patrón de diseño propone la construcción de un controlador que haga de interfaz entre el modelo y la vista. Con ello, toda la comunicación está centralizada en un único pun-

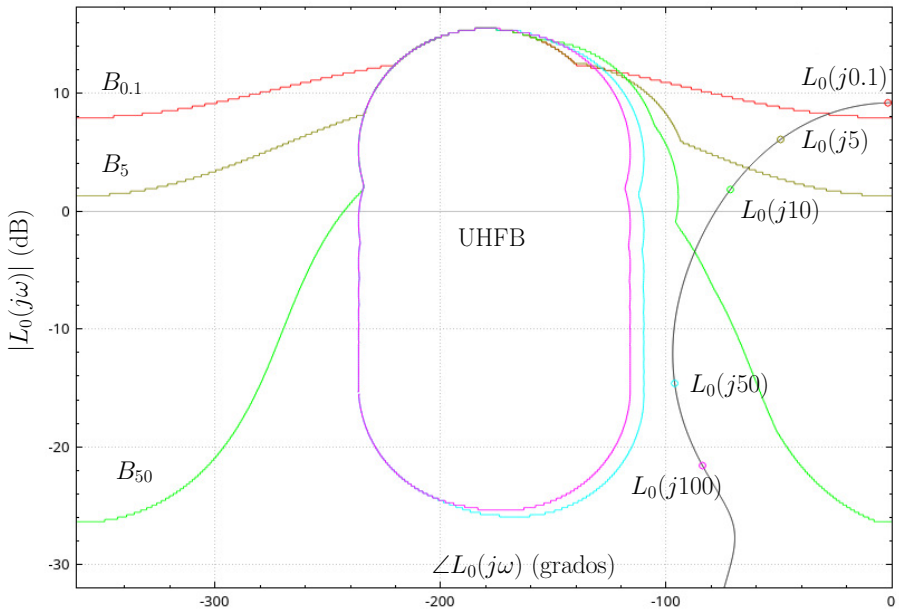

Figura 7.18: Diagrama de Nichols con $L_0(s)$ calculado por la aplicación.

to. Con lo que, el modelo y la vista quedarían desacoplados y su desarrollo puede producirse de forma independiente y autónoma. Además, este tipo de construcción permite que los dos componentes principales puedan ser intercambiables sin afectar al otro o añadir una nueva vista. Por ejemplo, una API REST (Richardson et al. [2013]), que también haga uso del modelo a través del controlador.

A lo largo de esta sección, se han utilizado conceptos relacionados con la ingeniería del software, como son UML y los patrones de diseño (Larman [2003]). A continuación, se introduce una notación básica para esta sección:

- *Patrón de diseño:* Técnicas estándar para la resolución de problemas comunes en el desarrollo de software.
- *Rol*: En el contexto de un sistema software, representa el comportamiento de un objeto con respecto al resto de objetos participantes.
- *Actor*: Representa el rol jugado con un usuario o cualquier otra entidad que interactúa en un sistema software.
- *Caso de uso*: Es una descripción de las distintas etapas o las actividades que deberán realizarse para llevar a cabo la conclusión de algún proceso. Los *usuarios* o *entidades* que participan en un caso de uso se denominan actores.
- *Diagrama conceptual*: Se encarga de definir qué elementos (entidades, objetos, áreas, departamentos, componentes, etc) pertenecen al sistema

que se está diseñando, así como las relaciones que se establecen entre ellos.

- *Diagrama de clases*: Describe la estructura estática de un sistema mostrando sus clases, atributos, funciones y las relaciones entre ellas.
- *Persistencia:* Acción de preservar la información de un objeto o sistema de forma permanente, y a su vez, también poder recuperar dicha información cuando sea necesaria.
- *Patrón adaptador:* Permite la colaboración entre dos interfaces software incompatibles entre ellas.
- *DAO*: *Data Access Object (DAO)* es un patrón de diseño de tipo adaptador que permite abstraer al sistema de una implementación de persistencia concreta.

En el la figura (7.19) se puede observar el diagrama de casos de uso, que representa toda la funcionalidad implementada en la aplicación. Como se puede observar, en el diagrama se representa a un solo actor, que es el usuario y que hace uso de todas las etapas de QFT que están implementadas en la aplicación.

Además, en la figura (7.20) se puede observar el diagrama conceptual del proyecto. Que refleja un diseño muy básico de como está construido el software. Al ser un diagrama conceptual no se entra en detalles de implementación. Esa información podrá consultarse en secciones posteriores.

7.4.1. Vista

La vista se diseñó e implementó como una interfaz gráfica basada en las etapas de QFT. Se inicia con una pantalla principal muy sencilla desde la cual acceder a cada una de dichas etapas. Como particularidad, indicar que las distintas pantallas de la aplicación propias de cada etapa están desacopladas de la pantalla principal y que permite abrir varias de ellas a la vez. De esta forma, se puede trabajar en las distintas etapas de forma simultánea y cualquier modificación de datos en una de las ventanas producirá una actualización de los mismos entre ellas, con los consecuentes recálculos de datos que sean necesarios, permitiendo así un funcionamiento más dinámico e interactivo. Para cada una de las etapas, la mecánica es muy similar. En primer lugar se muestra una pantalla de configuración e introducción de datos para que, al confirmar, y si corresponde, se ejecute el algoritmo respectivo. Al finalizar, se muestran los resultados con un gráfico en la pantalla.

En la figura (7.21) se puede observar el diagrama de clases de la interfaz.

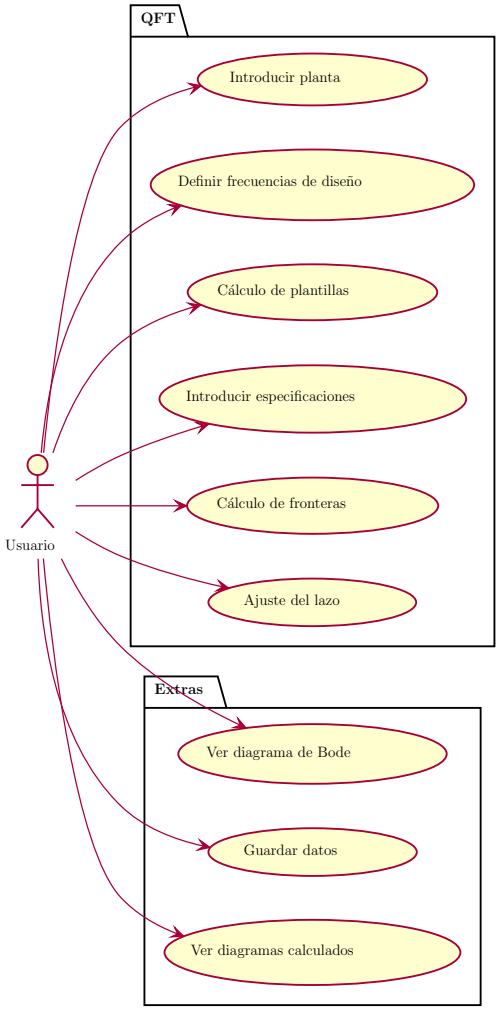

Figura 7.19: Diagrama de casos de uso del software.

Como ya se ha indicado anteriormente, la interfaz gráfica está programada usando la librería gráfica Qt sobre C++. Esta librería provee herramientas para crear gráficos de forma sencilla así como estructuras de almacenamiento de datos más completas que las clásicas de C++.

7.4.2. Modelo

El modelo del software reúne toda la funcionalidad implementada por el software. Ésta se divide principalmente en las distintas etapas de QFT, con algunos módulos accesorios. En la figura (7.22) se puede observar el diagrama de clases, en el cual, aparecen las principales entidades del modelo del software, así como sus principales relaciones. Por claridad, se han obviado multitud de clases y relaciones auxiliares.

A continuación, se procede a explicar cada uno de los módulos del modelo:

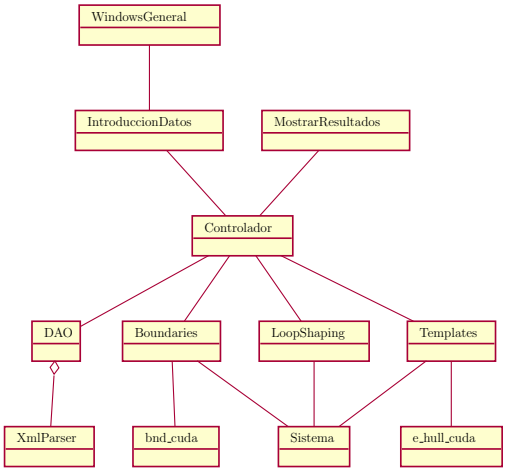

Figura 7.20: Diagrama conceptual del software.

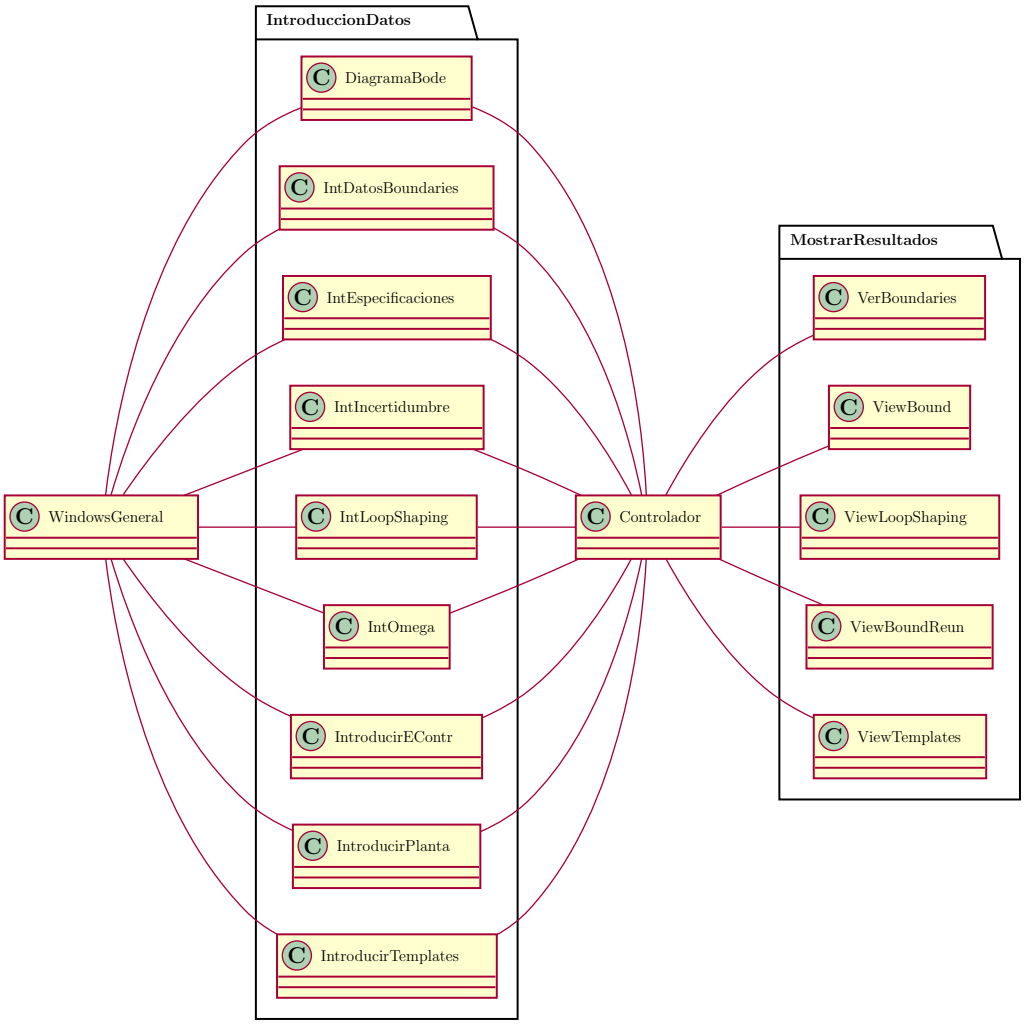

Figura 7.21: Diagrama de clases de la interfaz del software.

- **Boundaries:** (Moreno et al. [2006]) Es el módulo encargado de calcular las fronteras en QFT. Aparte del algoritmo principal, implementa dos algoritmos adicionales, el primero de ellos, *Contour* (Aramini [1980]), es necesario para completar el algoritmo principal y generar las fronteras en el diagrama de Nichols. El segundo de ellos, es el encargado de simplificar las fronteras creadas, reuniendo, para cada frecuencia de diseño, las distintas fronteras creadas en una sola. Esta forma de proceder facilita los cálculos posteriores.

- **Templates:** Módulo que agrupa los algoritmos necesarios para el cálculo de plantillas. El primero de ellos, calcula las plantillas completas realizando una combinatoria de los datos de la planta y su incertidumbre. El segundo de ellos, *e-hull* (Nordin [1993] usando como base la implementación de Montoya [1998]), resume estas plantillas mediante el cálculo de su contorno.

- **GPU:** Es el componente que incluye los algoritmos programados en *CUDA* para ejecutar en la GPU. Tiene implementados los siguientes procedimientos.

  - Algoritmo de cálculo de fronteras.
  - Algoritmo *e-hull*.

- **Sistema:** Es la estructura de datos encargada de representar tanto los datos como la funcionalidad asociada de las plantas y los controladores que se introduzcan en la aplicación. Está construida como una jerarquía de clases con herencia que representan las distintas formas que soporta la aplicación para introducir tanto plantas como controladores. Además, también soporta describir la incertidumbre de las plantas, así como los distintos rangos de las variables del controlador. Este sistema de herencia es flexible y permite añadir nuevas estructuras de definición de datos siempre que implementen las funciones cabecera que existen en la interfaz *Sistema*.

- **LoopShaping:** Módulo que agrupa los distintos algoritmos de ajuste del lazo implementados en la aplicación. Son los siguientes:

  - Algoritmo Nataraj y Tharewal (NT) (Nataraj y Tharewal [2002], Nataraj y Tharewal [2003] y Nataraj y Tharewal [2004]).
  - Algoritmo Nataraj y Kubal (NK) (Nataraj y Kubal [2006]).
  - Algoritmo Nataraj y Rambabu (NR) (Rambabu Kalla y P. S. V. Nataraj [2010]).
  - Algoritmo Martínez-Cervera previo (MC-prev) (Martínez-Forte y Cervera [2021]).

- Algoritmo Martínez-Cervera (MC), propuesto en esta tesis (5.6).

Complementando a los algoritmos principales, se añaden otros dos auxiliares, necesarios para los cálculos internos:

- Algoritmo que implementa la extensión natural a intervalos (1.2.5).
- Algoritmo que implementa el test de factibilidad de la caja con respecto a las fronteras (3.1.1).

Para terminar, se implementan dos estructuras de datos adicionales que son, la *Lista ordenada*, que es la lista de nodos vivos usados en el algoritmo, y la *Tripleta*, que representa a un nodo vivo, salvo para el algoritmo MC, que necesita guardar información extra de los nodos, y para ello utiliza un hijo de la clase anterior llamado *Tripleta2*.

- El siguiente de los módulos implementa el sistema DAO, que aplica dicho patrón para el sistema de guardado y lectura de datos. Es una especialización del patrón adaptador que permite abstraer el módulo de persistencia para no ligarlo a ningún tipo concreto de sistema. Para ello, implementa interfaces para cada una de las estructuras de datos a guardar que serán implementadas por adaptadores, que serán los encargados de desarrollar el sistema de persistencia concreto a utilizar.
- El último de los módulos complementa al patrón DAO y desarrolla el sistema concreto de persistencia. En este caso, estas dos clases implementan un parser a XML de las distintas estructuras de datos, con el objetivo de guardarlas en un fichero.

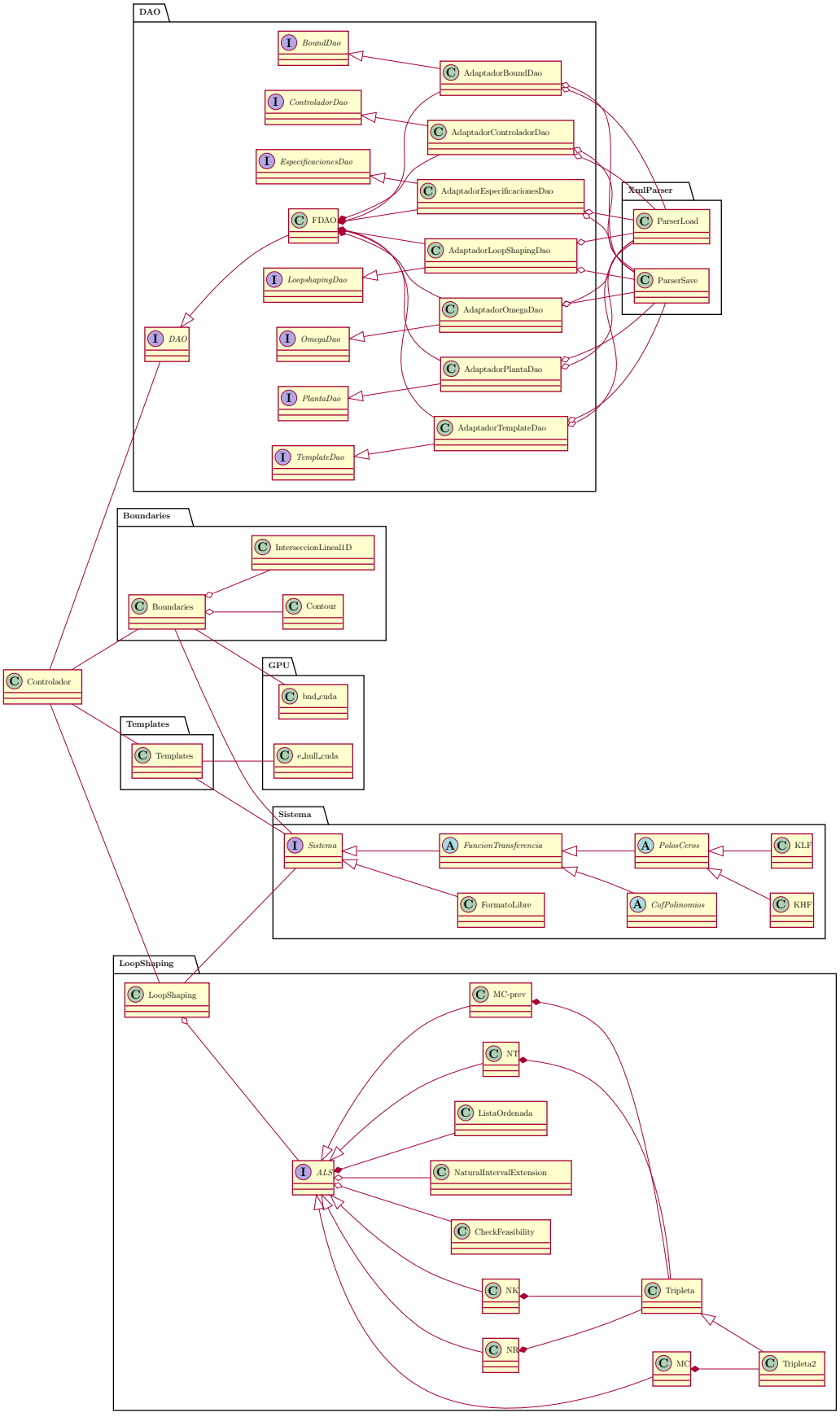

Figura 7.22: Diagrama de clases del modelo del software.

Capítulo 8

Conclusiones y vías futuras

8.1. Conclusiones

En este trabajo se ha abordado la resolución del problema del ajuste automático del lazo en QFT mediante un nuevo algoritmo de búsqueda global intervalar (ALS intervalar).

El algoritmo propuesto se basa en los algoritmos intervalares descritos en (3), capaces de resolver el problema original de forma exacta, pero con un alto coste computacional. Sobre esa base, se han propuesto diferentes estrategias (4) que tienen por objetivo conseguir un menor coste computacional y una mejora en el tiempo de ejecución. A modo de resumen, se listan todas la propuestas, implementadas y analizadas:

1. **Acotación del espacio de búsqueda:** Estrategias destinadas a reducir el espacio de búsqueda, resumidas en la sección (4.1.3). Pueden categorizarse del siguiente modo:

   a) **Acotación del espacio de búsqueda en el subrango inviable:** Esta estrategia tiene como objetivo detectar, para cada variable del controlador, qué parte de su rango es inviable para cada frecuencia de diseño, para así, sabiendo que viola las restricciones de diseño, poder eliminar dicha parte. Una primera propuesta en el algoritmo NK (3.2.2) realiza las operaciones utilizando la magnitud en el plano de Nichols. Esta funcionalidad es ampliada con una nueva estrategia que realiza esta misma operación utilizando también la fase; ampliando de esta forma, las opciones de acotación.

   b) **Acotación del espacio de búsqueda en el subrango viable:** Esta estrategia tiene por objetivo detectar para cada frecuencia

de diseño, y para cada variable del controlador, qué parte de las mismas contienen soluciones. Y con ello, poder podar todas las cajas que tienen soluciones peores. Esta estrategia, al igual que la anterior, se plantea para poder realizar los cálculos utilizando tanto la magnitud como la fase en el diagrama de Nichols.

2. **Bisección del nodo:** En esta estrategia se proponen varias formas de biseccionar el nodo (4.2), cada una de ellas centrada en una necesidad distinta para el momento en el que se ejecuten. Cabe reseñar que la más importante de las propuestas dentro de este apartado es la *bisección en árbol*. Este procedimiento, descrito en (4.2.4), utiliza los datos calculados en la estrategia (4.1.1) para realizar la bisección en el lugar exacto donde la variable del controlador pasa de ser ambigua a ser viable para una o varias frecuencias de diseño. De este modo se obtienen dos subnodos, siendo uno de ellos viable a ciertas frecuencias. Esta información puede ser utilizada en este mismo nodo y en su descendencia para ahorrar cálculos innecesarios además de para guiar la búsqueda. Esta forma de proceder tiene como objetivo conseguir que para el mayor número posible de frecuencias de diseño, el nodo sea viable. Y así, poder centrar el esfuerzo en las restantes frecuencias donde todavía sea ambiguo.

3. **Búsqueda mejor ganancia de alta frecuencia:** Esta estrategia se describe en (4.3) y funciona encontrando soluciones rápidas al problema donde la ganancia de alta frecuencia sea lo menor posible. Lo que permite podar o recortar en etapas tempranas nodos del espacio de búsqueda con el consiguiente ahorro sustancial de tiempo.

4. **Etapas de ejecución e historia del nodo**: Esta estrategia es desarrollada en (4.4) y propone que, dado que la relación entre la caja de proyección y las fronteras varía cualitativamente a lo largo de la ejecución del algoritmo, es una opción prometedora modificar el funcionamiento del procedimiento y adaptarlo a la situación en la que el nodo esté en cada momento. Para ello, es necesario guardar la historia del nodo, dado que cada nodo es independiente y tendrá un desarrollo diferente al resto, dependiendo principalmente de cuánto se haya expandido la rama del árbol de búsqueda donde se sitúe. Una vez que se conoce con detalle el momento concreto del nodo, se pueden tomar diferentes decisiones, por ejemplo, elegir una forma u otra de bisección según sea más conveniente, o desactivar las estrategias de acotación del espacio de búsqueda cuando ya no tenga sentido ejecutarlas.

Se ha propuesto e implementado un algoritmo (5) que incluye todas las estrategias enumeradas. Algunas de estas estrategias tienen un carácter más local, como por ejemplo, las herramientas para la acotación del espacio de búsqueda o la búsqueda de soluciones rápidas que permitan realizar podas

en las ramas menos prometedoras. Otras en cambio, tienen un carácter más transversal, como por ejemplo, la modificación del comportamiento del algoritmo conforme avanza su ejecución. Al ser estrategias de tan diverso funcionamiento, el algoritmo las coordina y resuelve las dependencias entre ellas, presentando de esta forma, una solución compacta que permite ponerlas en funcionamiento de forma sencilla.

Es necesario tener en cuenta que el algoritmo implementado tiene sus limitaciones, solo admite términos (ceros/polos) reales con ceros de fase mínimo y polos estables. Por otro lado, solo tiene implementada una función objetivo, que es la clásica en QFT de minimizar la ganancia de alta frecuencia.

El algoritmo desarrollado ha sido puesto a prueba mediante la resolución de dos casos prácticos (6) con el objetivo de medir el rendimiento del nuevo algoritmo con respecto a los anteriores. Se ha llegado a la conclusión de que la mejora en el tiempo de ejecución es importante, permitiendo aumentar el tamaño del problema de forma considerable mientras se mantiene el tiempo de ejecución por debajo de los dos minutos y medio. Además, en este mismo capítulo, se analizan las variaciones en el tiempo de ejecución producidas por cada estrategia de forma individual, así como en diversas combinaciones interesantes de las mismas. De esta forma, se puede conocer de forma pormenorizada cómo se comporta cada una de ellas.

También se ha desarrollado un software (7) que implementa todas las etapas principales de QFT. Faltan por implementar algunas accesorias, como por ejemplo, el ajuste del prefiltro. Para las etapas implementadas, permite dar solución a un problema aplicando diferentes algoritmos en cada una de dichas etapas. Para la etapa de ajuste del lazo, se han implementado los algoritmos ALS intervalares explicados en el capítulo (3), así como, dos versiones de los algoritmos desarrollados durante esta tesis. Una primera versión publicada (Martínez-Forte y Cervera [2021]) que incluye alguna de las estrategias propuestas y una segunda versión con el algoritmo completo propuesto en esta tesis.

En todos los algoritmos desarrollados en el software, tanto para el ajuste del lazo, como para el resto de etapas, se ha puesto el énfasis en conseguir que tuvieran tiempos de ejecución bajos con el objetivo de plantear una aplicación que permitiera la interactividad del usuario con las distintas etapas de QFT. De esta forma, si se varían los datos en cualquiera de las etapas, estos cambios se aplicarían de forma rápida al resto de etapas posteriores. Así se permite la realización de pruebas y cambios de forma rápida y ágil.

Este software se ha publicado bajo licencia libre GPLv3 y está a disposición en descarga libre en (`https://github.com/Isaac-Martinez-Forte/QFTbx`).

8.2. Vías futuras

A continuación, se enumeran diferentes posibles continuaciones del trabajo realizado:

- El algoritmo presentado está actualmente acotado al uso de polos y ceros reales, por tanto, sería importante ampliarlo para que permita utilizar estructuras con términos de segundo orden. Eso implica revisar varias de las estrategias propuestas y adaptarlas para esta característica.

- Una de las estrategias diseñadas (4.4) propone variar el funcionamiento del algoritmo a lo largo de su ejecución para adaptarse a las diferentes situaciones en las que se encuentra el nodo a lo largo del algoritmo. Una propuesta interesante es que el algoritmo adapte su funcionamiento según el tipo de problema que esté tratando de resolver. Varias de las estrategias basan su forma de proceder a la interacción de las fronteras y la caja proyectada en el plano de Nichols. Por tanto, analizar cómo es esta interacción y adaptarse a diferentes posibilidades puede dar buenos resultados.

- Actualmente, la ramificación se realiza por el nodo que tenga la ganancia de alta frecuencia más baja. Esta ramificación se convierte, en la práctica, en una búsqueda secuencial por dicha variable desde su valor más bajo hasta que se encuentra la solución óptima. Por tanto, incorporar otros sistemas inteligentes para la ramificación del árbol de búsqueda, por ejemplo, sistemas de ramificación basados en la información disponible, como el número de frecuencias para las que un nodo es viable, y que apliquen heurísticas para seleccionar el siguiente nodo a ser analizado.

- En algunos experimentos se ha comprobado, inicialmente y para algunos ejemplos concretos, que la extensión natural a intervalos produce cajas proyectadas de un tamaño mayor al estrictamente necesario. Sería importante profundizar en este estudio, y si fuera así, plantear soluciones para reducir al máximo dicho tamaño.

- Diversos algoritmos de la aplicación se han implementado utilizando programación paralela, pero la etapa principal, el ajuste del lazo, solo tiene implementación secuencial. Se plantea como mejora, utilizar técnicas de paralelismo, tanto en CPU como en GPU, para implementar una versión paralela del algoritmo de ajuste automático del lazo presentado en esta tesis.

- El software desarrollado incluye las principales etapas de QFT, pero sería interesante completarlo implementando el resto de fases accesorias, así como, realizar una revisión completa para la renovación del código antiguo.

- Actualmente, el algoritmo presentado tiene como única función objetivo minimizar la ganancia de alta frecuencia. El siguiente paso, sería generalizar el algoritmo diseñado junto las estrategias para que se puedan utilizar diferentes funciones objetivo en la optimización.

Bibliografía

M. J. Aramini. *Implementation of an Improved Contour Plotting Algorithm*. Tesis Doctoral Stevens Institute of Technology [1980].

F. Bailey, J. W. Helton y O. Merino. Alternative approaches in frequency domain design of single loop feedback systems with plant uncertainty. En *Proceedings of the American Control Conference* 345–349. [1994].

F. Bailey, D. Panzer y G. Gu [1988]. Two algorithms for frequency domain design of robust control systems. *International Journal of Robust and Nonlinear Control* **48**, nº 5: 1787–1806.

D. Ballance y P. J. Gawthrop. Control system design via a quantitative feedback approach. En *Proceedings of the IEEE Conference Control-91* 476–480. Heriot-Watt University, Edinburgh, U.K. [1991].

V. Blondel [1994]. *Simultaneous stabilization of linear systems*. Nº 191 en LNCIS series. Springer-Verlag.

H. Bode [1945]. *Network Analysis and Feedback Amplifier Design*. Van Nostrand, NY, USA.

C. Borghesani, Y. Chait y O. Yaniv [2003]. *The QFT Frequency Domain Control Design Toolbox For Use with MATLAB*. Terasoft, Inc. Último acceso (2/08/2013).

M. Brown y Y. R. Peterson [1991]. Synthesis of feedback systems with large plant ignorance for prescribed time-domain tolerances. *International Journal of Control* **16**, nº 2: 287–309.

G. Bryant y G. Halikias [1962]. Optimal loop shaping for systems with large parameter uncertainty via linear programming. *International Journal of Control* **3**: 557–568.

Y. Chait, Q. Chen y C. Hollot [1999]. Automatic loop-shaping of qft controllers via linear programming. *ASME J. Dynamic Systems, Measurement, and Control* **121**: 351–357.

Y. Chait y O. Yaniv [1993]. Multiple-input/single-output computer aided control design using the quantitative feedback theory. *International Journal of Robust and Nonlinear Control* **3**: 47–54.

W. Chen, D. Ballance y Y. Li. Automatic loop-shaping in qft using genetic algorithms. Inf. téc. Centre for Systems and Control, University of Glasgow Glasgow, UK [1998].

Y. Chiang y R. Safonov [2001]. *Robust Control Toolbox for use with MatLab*. The MathWorks, Inc., Natick, MA, USA.

P. S. Dwyer [1951]. *Computation with approximate numbers. Linear Computations*. New York, Wiley.

M. S. Fadali y L. E. LaForge [1996]. Algorithmic analysis of geometrically computed qft bounds. *IFAC Proceedings Volumes* **29**, nº 1: 3690–3695. 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July.

C. Fransson, B. Lennartson, T. Wik, K. Holmström, M. Saunders y P. Gutman. Global controller optimization using horowitz bounds. En *Proceedings of the FRAC 15th Triennial World Congress*. [2002].

E. Gamma, R. Helm, R. Johnson y J. Vlissides [1995]. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

M. Garcia-Sanz. The qft control toolbox (qftct) for matlab. [2008-present].

A. Gera y I. Horowitz [1980]. Optimization of the loop transfer function. *Int. J. of Control* **31**: 389–398.

R. Hammer, M. Hocks, U. Kulisch y D. Ratz [1995]. *C++ Toolbox for Verified Scientific Computing - Theory, Algorithms and Programs: Basic Numerical Problems*. Springer-Verlag Berlin Heidelberg.

I. Horowitz [1959]. Fundamental theory of automatic linear feedback control systems. *IRE Transactions on Automatic Control* **4**, nº 3: 5–19.

I. Horowitz [1973]. Optimum loop transfer function in single-loop minimum-phase feedback systems. *Int. J. of Control* **18**: 97–113.

I. Horowitz [1993]. *Quantitative Feedback Design Theory - QFT (Vol.1)*. QFT Press, Boulder, Colorado, USA.

I. M. Horowitz [1963]. *Synthesis of feedback systems*. Academic Press, New York.

I. M. Horowitz y M. Sidi [1972]. Synthesis of feedback systems with large plant ignorance for prescribed time-domain tolerances†. *International Journal of Control* **16**, nº 2: 287–309.

ISO [2012]. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland.

W. Kahan. A more complete interval analysis. [1968].

R. Klatte, U. Kulisch, A. Wiethoff, C. Lawo y M. Rauch [1993]. *C-XSC 2.0 a C++ Class Library for Extended Scientific Computing*. Springer-Verlag Berlin Heidelberg.

A. H. Land y A. G. Doig [1960]. An automatic method of solving discrete programming problems. *Econometrica* **28**, nº 3: pp. 497–520.

C. Larman [2003]. *UML y patrones: introduccioén al anaélisis y diseñño orientado a objetos y al proceso unificado*. Prentice-Hall, Madrid [etc 2ª ed., [uélt. reimp. 2003] edón.

J. D. C. Little, K. G. Murty, D. W. Sweeney y C. Karel [1963]. An algorithm for the traveling salesman problem. *Operations Research* **11**, nº 6: 972–989.

E. Loh y G. W. Walster [2002]. Rump's example revisited. *Reliable Computing* **8**, nº 3: 245–248.

I. Martínez-Forte y J. Cervera. *Paralelización de algoritmos QFT mediante OpenMP y CUDA*. Proyecto Fin de Carrera Facultad de Informática, Universidad de Murcia [2014].

I. Martínez-Forte y J. Cervera [2021]. Accelerated quantitative feedback theory interval automatic loop shaping algorithm. *International Journal of Robust and Nonlinear Control* **31**, nº 9: 4378–4396.

F. J. Montoya. *Diseño de sistemas de control no lineales mediante QFT: análisis computacional y desarrollo de una herramienta CACSD* . Tesis Doctoral [1998].

R. E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. Tesis Doctoral Department of Mathematics, Stanford University Stanford, CA, USA [1962]. Also published as Applied Mathematics and Statistics Laboratories Technical Report No. 25.

R. E. Moore [1966]. *Interval Analysis* 158. Prentice-Hall, New Jersey, (USA).

R. E. Moore y F. Bierbaum [1979]. *Methods and Applications of Interval Analysis (SIAM Studies in Applied and Numerical Mathematics) (Siam Studies in Applied Mathematics, 2.)*. Soc for Industrial & Applied Math.

J. C. Moreno, A. Baños y M. Berenguel [2006]. Improvements on the computation of boundaries in qft. *International Journal of Robust and Nonlinear Control* **16**, nº 12: 575–597.

R. Nandakumar, G. Halikias y A. Zolotas. An optimization algorithm for designing fixed-structure controllers using the qft method. En *Proceedings of the IEEE International Symposium on Computer Aided Control System Design* 157–162. Glasgow, Scotland, U.K. [2002].

P. Nataraj y M. M. Deshpande [2008]. Automated synthesis of fixed structure qft controller using interval constraint satisfaction techniques. *IFAC Proceedings Volumes* **41**, nº 2: 4976–4981. 17th IFAC World Congress.

P. Nataraj y N. Kubal [2006]. Automatic loop shaping in qft using hybrid optimisation and constraint propagation techniques. *International Journal of Robust and Nonlinear Control - Special Issue: Quantitative Feedback Theory. In memoriam of Isaac Horowitz.* **17**: 251–264.

P. S. V. Nataraj y S. Tharewal. Automatic loop-shaping in qft using interval global optimization. En *Proceedings of the International Conference on Multimedia and Design (ICMD)*. Mumbai, India [2002].

P. S. V. Nataraj y S. Tharewal. Design of optimal and sub-optimal qft controllers using interval analysis. En *Proceedings of the International Symposium on Process Systems Engineering and Control*. IIT Bombay, India, [2003].

P. S. V. Nataraj y S. Tharewal. An interval analysis algorithm for automated controller synthesis in qft designs. En *Proceedings of the NSF Workshop on Reliable Engineering Computing*. Savannah, Georgia, USA [2004].

M. Nordin. *Uncertain systems with backslash: modeling, identification and synthesis*. Proyecto Fin de Carrera Royal Institute of Technology [1993].

M. Novoa. Advances in the interval solution of algebraic systems. [1993].

NVIDIA, P. Vingelmann y F. H. Fitzek. Cuda, release: 10.2.89. [2014].

OpenMP Architecture Review Board. Openmp application program interface. Specification [2011].

M. D. Patil y P. Nataraj [2012]. Automated synthesis of multivariable qft controller using interval constraint satisfaction technique. *Journal of Process Control* **22**, nº 4: 751–765.

P. QT, H. Nord y E. Chambe-Eng. Qt5. https://www.qt.io/ [2022].

C. Raimúndez. *Estrategias Evolutivas y su Aplicación en la Síntesis de Controladores*. Tesis Doctoral E. T. S. de ingenieros industriales, Universidad de Vigo Vigo, España [1997].

C. Raimúndez, A. Baños y A. Barreiro. Qft controller synthesis using evolutive strategies. En *Proc. of the 5th International QFT Symposium on Quantitative Feedback Theory and Robust Frequency Domain Methods* 291–296. Pamplona, Spain [2001].

Rambabu Kalla y P. S. V. Nataraj. Synthesis of Fractional-order QFT Controllers using Interval Constraint Satisfaction Technique. Inf. téc. IIT Mumbai [2010].

D. Ratz. On extended interval arithmetic and inclusion isotonicity. Preprint [1996].

L. Richardson, M. Amundsen y S. Ruby [2013]. *RESTful Web APIs*. O'Reilly Media, Inc.

J. Rumbaugh, I. Jacobson y G. Booch [2004]. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education.

N. S. V. Paluri y N. Kubal [2007]. Automatic loop shaping in qft using hybrid optimization and constraint propagation techniques. *International Journal of Robust and Nonlinear Control* **17**, nº 2-3: 251–264.

T. Sunaga [2009]. Theory of an interval algebra and its application to numerical analysis. *Japan Journal of Industrial and Applied Mathematics* **26**, nº 2: 125–143.

D. F. Thompson y O. Nwokah [1994]. Analytic loop shaping methods in quantitative feedback theory. *Journal of Dynamic Systems, Measurement and Control* **116**: 169–177.

D. Thomson. *Optimal and Sub-Optimal Loop Shaping in Quantitative Feedback Theory*. Tesis Doctoral School of Mechanical Eng., Purdue University West Lafayette, IN, USA [1990].

B. Wie y D. Bernstein. A benchmark problem for robust control system design. En *Proceedings of the ACC*. ACC, San Diego, CA, USA. [1990].

O. Yaniv y M. Nagurka [2003]. Robust pi controller design satisfying sensitivity and uncertainty specifications. *IEEE Automatic Control* **48**, nº 11: 2069–2072.

O. Yaniv y M. Nagurka. Automatic loop shaping of structured controllers satisfying qft performance. Inf. téc. [2004a].

O. Yaniv y M. Nagurka [2004b]. Design of pid controllers satisfying gain margin and sensitivity constraints on a set of plants. *Automática* **40**, nº 1: 111–116.

A. Zolotas y G. Halikias [1999]. Optimal design of pid controllers using the qft method. *IEEE Proc.- Control Theory Appl.* **146**, nº 6: 585–589.