Trabajo fin de Máster
Máster en Ingeniería Industrial

Control predictivo basado en partículas y simulaciones de Montecarlo

Autor: Alfonso Daniel Carnerero Panduro

Tutor: Daniel Rodríguez Ramírez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo fin de Máster
Máster en Ingeniería Industrial

Control predictivo basado en partículas y simulaciones de Montecarlo

Autor:
Alfonso Daniel Carnerero Panduro

Tutor:
Daniel Rodríguez Ramírez
Profesor Titular de Universidad

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo fin de Máster: Control predictivo basado en partículas y simulaciones de Montecarlo

Autor: Alfonso Daniel Carnerero Panduro
Tutor: Daniel Rodríguez Ramírez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Vistas las críticas recibidas por algunos sectores acerca de la extensión de los agradecimientos en el anterior Trabajo Fin de Grado, me veo en la obligación de extenderme un par de líneas (aunque no tengo la intención de quemar todos los cartuchos aún).

Me gustaría dirigirme a todos aquellos que han hecho posible esto (quien tenga oídos para oír, que oiga) cada uno a su manera. Algunos me dieron apoyo o consejos, otras personas me han dado algo que llevarme a la boca durante mi existencia, etc. No voy a poner nombres porque aún todavía me queda un pasito más por dar y, como ya he dicho, no pretendo hacer la sección de agradecimientos definitiva hasta que culmine mi etapa estudiantil...

Así que, después de esta parrafada sin ningún sentido más que añadir un poco de volumen a la anterior sección sosa y vacía de hace 2 años, ¡os agradezco por todos estos años y los que quedan!

Sevilla, 26 de Junio de 2019

Resumen

El Control Predictivo basado en Modelo es, desde su aparición, una de las estrategias de control avanzado más populares y con mayor relevancia en la industria en la actualidad. Continuamente se publican nuevos resultados y avances, consiguiendo cada vez un mejor rendimiento del controlador o proporcionando garantías que antes no existían.

Por esta razón, enfocamos el presente trabajo en el estudio de una posible estrategia de Control Predictivo basado en Métodos de Montecarlo o filtros de partículas, la cual tiene el potencial de hacer frente a diferentes tipos de perturbaciones presentes en los sistemas a controlar.

En los algoritmos desarrollados a lo largo del proyecto aparecen los conceptos de partícula y escenario, los cuáles tienen una gran importancia en estas páginas. Cada uno de ellos hace referencia a una característica concreta de los algoritmos y de los problemas a resolver. A modo de resumen:

- Las partículas hacen referencia a una solución potencial del problema de optimización. Es decir, se utilizan para resolver un problema matemático con un método alternativo a otros más convencionales como pueden ser los gradenciales.
- Los escenarios son una realización hipotética y posible de las perturbaciones durante la operación del sistema. Por tanto, el controlador tiene en cuenta algunos de los escenarios posibles para tomar las decisiones respecto al sistema incierto.

Los algoritmos se prueban en distintos sistemas con características diferentes así como a diferentes tipos de perturbación, con el objetivo de evaluar la bondad de estos. En el capítulo 5 se presentan todos los ensayos realizados así como los resultados obtenidos.

Índice Abreviado

Índice

1 Introducción

Este documento resulta una continuación natural del trabajo fin de grado ”Procesamiento masivamente paralelo en control predictivo basado en datos”. En aquel trabajo, se hizo una introducción a la programación paralela y al uso básico de las librerías de CUDA proporcionadas por NVIDIA. Como resultado, se programó un algoritmo paralelizado de control predictivo basado completamente en datos.

Ahora bien, en el trabajo que presentamos actualmente proponemos la evaluación de otros algoritmos susceptibles de ser paralelizados nuevamente para el ámbito del control predictivo, aunque en esta ocasión nos centraremos en la vertiente ”clásica” basada en modelo, en lugar de basarlo en datos.

Sin embargo, se pretende hacer especial hincapié en el rechazo de las perturbaciones (de diversa índole como se verá posteriormente) en los sistemas propuestos como casos de estudio. Más concretamente, los problemas que se plantean a lo largo de este proyecto son los siguientes:

- Controlar un sistema no lineal a partir de un modelo lineal y de una serie de tolerancias en las matrices del sistema que tratan de reflejar la dinámica total del sistema. Es decir, representar un modelo no lineal a partir de modelos lineales para todo el rango de operación.
- Controlar un sistema lineal de primer orden con una única entrada y salida con una ganancia variable dentro de un determinado rango. Es decir, un sistema que posee uno de los coeficientes desconocido dentro de un rango que sí es conocido.
- Controlar un sistema lineal multivariable con perturbaciones aditivas no acotadas en uno de los estados. Afectar a uno de los estados en lugar de a las salidas implicará que la perturbación se integrará en la evolución del mismo, añadiendo mayor dificultad al problema de control.
- Controlar un sistema lineal sujeto a incertidumbres de tipo politópico en una de las matrices del sistema. Se trata de un sistema lineal variante en el tiempo (LTV) donde en cada instante el valor de la matriz A cambia, permaneciendo siempre dentro de un determinado conjunto convexo.

Durante el transcurso del trabajo se explicarán cada uno de los algoritmos implementados y se expondrán los resultados que caracterizan a cada uno de ellos, comparándolos en todo momento con el MPC lineal, el cual consideraremos como el caso base.

El trabajo a partir de ahora se estructura de la siguiente manera:

- Control Predictivo basado en Modelo. Se hará una breve introducción histórica, se plantearán sus ventajas e inconvenientes y algunos detalles matemáticos de implementación, tales como las funciones de coste más comunes, etc.
- En el siguiente capítulo, se profundiza un poco más en la planta de los cuatro tanques, que es el esqueleto principal alrededor del cual se ha construido el trabajo y se plantean los controladores siguientes. Eso no significa que no existan ejemplos con otros sistemas distintos en el trabajo. Al contrario, aparecen 3 ejemplos más, pero no necesitan un análisis en profundidad al ser sistemas lineales.
- Posteriormente, se explica el algoritmo inicial y cada una de las modificaciones que han ido surgiendo durante el desarrollo del proyecto. Estos cambios tienen por objetivo mejorar el rendimiento del algoritmo en algún aspecto concreto de su funcionamiento.

- El siguiente capítulo aglutina todos los resultados obtenidos en el trabajo para cada uno de los ejemplos expuestos anteriormente.
- Para terminar, se plantean las conclusiones del trabajo a partir de los resultados obtenidos, así como las perspectivas de futuro en esta línea de investigación.

1.1 Tarjetas gráficas y CUDA

Una GPU (Graphics Processor Unit) no es más que un coprocesador encargado de aligerar la carga de la CPU en algunas tareas determinadas. Habitualmente, se encarga de mostrar los gráficos en pantalla. Antes de pasar a estudiar su estructura ser interesante esbozar su evolución histórica.

1.1.1 Historia

En un principio, la única manera de mejorar el rendimiento de un ordenador estaba completamente orientado en la mejora de su procesador central. De esta manera, los fabricantes fueron incrementando la velocidad de sus procesadores a través de mejoras en los relojes internos de estos. En poco tiempo, pasaron de funcionar de unas frecuencias como 1MHz a otras en torno a a 1 GHz. Esto suponía mejorar la velocidad 1000 veces. Pero, debido a diversas limitaciones que fueron presentándose, los fabricantes e investigadores tuvieron que empezar a buscar otras posibilidades.

Concurrentemente, estaba el terreno de los supercomputadores. Estos se estaban beneficiando de las mejoras de los ordenadores personales pero también empezaron a desarrollar tecnología que terminó resultando de gran interés general. En resumen, consistía en poner a trabajar varios procesadores en paralelo. De esta manera, las CPUs empezaron a a incluir varios procesadores independientes, cosa bastante habitual a día de hoy.

¿Y las GPUs? Pues bien, estas empezaron a desarrollarse a principios de la decáda de los 80 como consecuencia del éxito de sistemas operativos con interfaz gráfica. En aquel entonces, los usuarios empezaron a pedir dispositivos que mejoraran el funcionamiento del sistema a la hora de presentar los gráficos en pantalla.

De igual manera que ocurrió con las CPUs, dos mercados independientes empezaron a trabajar en las GPUs. Por ello, en el ámbito de la computación profesional, la compañía Silicon Graphics empezó a popularizar el uso de gráficos en 3D para aplicaciones muy diversas. De esta manera, terminaron liberando su interfaz de programación y sus librerías (OpenGL) en el año 1992, pretendiendo que se convirtiera en el estándar de creación de gráficos tridimensionales.

Como era de esperar, los consumidores empezaron a demandar esta tecnología, gracias principalmente al éxito de algunos videjuegos inmersivos en primera persona como Doom o Quake. A partir de aquí, varias empresas como NVIDIA, ATI Technologies o 3dfx empezaron a desarrollar procesadores gráficos más asequibles con el objetivo de satisfacer este sector del mercado.

En paralelo a este desarrollo de las GPUs, algunos investigadores empezaron a estudiar la posibilidad de usarlas para otros objetivos. Sin embargo, la única manera de acceder a las GPUs por aquel entonces era a través de su API (Application Programming Interface). Sin embargo, esta programación era tan complicada y dependiente del modelo concreto de la tarjeta que no logró mucho éxito dentro del sector.

Todo esto cambió cuando NVIDIA presentó la arquitectura CUDA (Compute Unified Device Architecture), la cual alivia gran parte de las limitaciones antes expuestas en el terreno de la computación de propósito general.

1.1.2 Arquitectura CUDA

Existen un buen número de problemas que pueden beneficiarse de la inmensa cantidad de núcleos que poseen las tarjetas gráficas, mejorando su rendimiento de manera exponencial haciendo uso de la computación paralela. Aún así, existen alguna deficiencias como por ejemplo:

- En la ejecución de código en la GPU existe siempre un tiempo implícito relacionado con el paso de memoria entre la CPU y GPU.
- Los núcleos CUDA deben ejecutar todos el mismo código. Es decir, no son independientes.
- La sincronización entre ellos es muy restrictiva.

El flujo de información de una ejecución en CUDA viene representado en la figura 1.1.

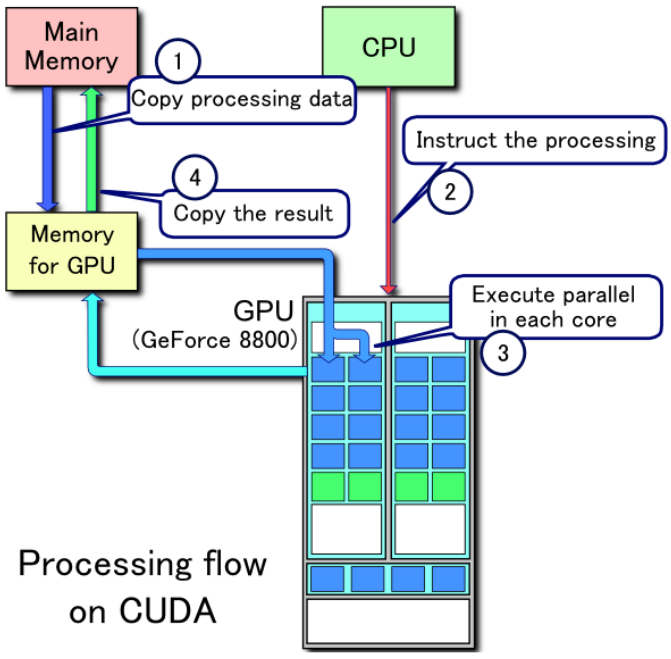

Figura 1.1 Flujo de información en CUDA.

Es fácil ver cómo es necesario mover mucha información durante la ejecución de un programa. Los tiempos que son necesarios para este intercambio son denominados overheads y son decisivos a la hora de elegir usar una GPU para una determinada aplicación. Por esta razón, la mejora en velocidad surge en problemas de gran escala. Mientras que el tiempo de computación para problemas de pequeño tamaño se mantiene igual, llegando a empeorar en algunos casos.

Por otro lado, es imprescindible que el problema que queramos resolver sea paralelizable. Para aclarar estos conceptos, presentamos los dos ejemplos siguientes:

- En las aplicaciones para las que fueran concebidas las GPUs, las operaciones a realizar sobre cada pixel son completamente independientes. Además, no importa de el orden de ejecución.
- En contraposición, para la resolución de un sistema de ecuaciones diferenciales es necesario conocer la información de los instantes anteriores para poder computar el instante actual. Es decir, debe iterarse en orden y éste no puede alterarse. En cambio, sí que podría resolverse el mismo sistema de ecuaciones para diferentes condiciones iniciales en paralelo puesto que cada problema sería independiente.

Es decir, la GPU nos permite dos aplicaciones de carácter general bien diferenciadas:

- Resolver problemas implícitamente paralelizables.
- Resolver múltiples problemas no paralelizables.

1.1.3 Estructura de la memoria

Los hilos son la unidad indivisible de CUDA, es decir, que cada uno se ejecutará de manera independiente en un núcleo de la GPU. Estos hilos, a su vez, se agrupan en bloques. Esto no es más que una manera de organizar los hilos tanto a nivel hardware como software. Esta estructura nos permite asignar índices en 3 dimensiones tanto a los hilos como a los bloques. De esta manera, podemos identificar completamente a cualquier hilo perteneciente al conjunto.

2 Control Predictivo basado en Modelo

2.1 Introducción

El Control Predictivo aparece durante la década de los 70 cuando se empiezan a usar modelos de sistemas para predecir las salidas futuras de manera explícita teniendo en cuenta diversos factores. Principalmente, como es de esperar, el valor de las acciones de control. Además, permitía fácilmente tener en consideración restricciones de operación, lo cual resultaba mucho más complicado para otras estrategias de control. De esta manera, se resolvía un problema de optimización que calculaba la secuencia de acciones óptimas para llevar el sistema hasta una determinada referencia deseada. Cabe destacar que la mayoría de estas formulaciones estaban basadas en la heurística y buscaban aprovecharse de los computadores de la época.

Rápidamente, El Control Predictivo basado en Modelo (*Model Predictive Control*) ganó popularidad principalmente en las industrias químicas debido a la sencillez de uso del algoritmo e identificación del sistema.

Sin embargo, el Control Predictivo no es una estrategia de control concreta, sino que se asemeja más bien a un conjunto de ideas comunes con muchas ramificaciones y diferencias posibles. Las ideas generales que conforman el esqueleto de un algoritmo MPC son las siguientes:

- Predicción de la salida del sistema para los instantes futuros hasta un determinado horizonte de predicción. En nuestro caso, usaremos un modelo linealizado en torno a un punto de operación y posteriormente discretizado.
- Cálculo de las acciones de control para un determinado horizonte de control minimizando una determinada función de costes. Esta función de coste puede tener una forma cualquiera, aunque sabremos que tendrá términos que penalicen el error en seguimiento, el esfuerzo de las acciones de control y otros similares.
- Estrategia de horizonte deslizante. Es decir, aunque en cada instante de muestreo se calcula una secuencia de acciones de control completa, sólo se aplica al sistema la primera de ellas. Esto proporcionará *realimentación* a nuestro controlador.

Muchos algoritmos se diferencian en los modelos empleados y en las funciones que se desean minimizar. Sin embargo, también pueden diferenciarse, por ejemplo, en la metodología de cálculo que planteen. Por ejemplo, para el trabajo que se está presentando, se plantea un algoritmo que calcula la solución del problema de manera distinta (además de tener otras consideraciones, por supuesto).

2.2 Ventajas e inconvenientes

Una vez expresadas las ideas generales del Control Predictivo, podemos presentar algunas de sus ventajas, entre las que podemos destacar:

- Presenta unos conceptos fácilmente comprensibles e intuitivos que permiten una fácil comprensión incluso a personal que no se encuentra muy familiarizado con esta disciplina.

- Permite el control de sistemas muy diversos, desde aquellos que presentan una dinámica muy simple a otros con gran complejidad, incluyendo sistemas de fase no mínima, inestables, etc. De hecho, es fácil darse cuenta por la propia definición de los conceptos que el algoritmo posee una compensación intrínseca al retardo.

- Resulta de gran sencillez el trato del caso multivariable, debido a que simplemente supone la adición de variables de decisión adicionales al problema de optimización.

- Añadir restricciones de operación al controlador es muy sencillo, puesto que simplemente son restricciones a un problema de optimización numérica.

- Permite tener en cuenta en el control las referencias para los instantes futuros, lo cual es interesante en algunos campos como la robótica.

Sin embargo, también posee varios inconvenientes, tales como:

- La carga de cálculo necesaria para la resolución del problema de optimización en cada instante, lo cual puede provocar en algunos casos extremos, que no se pueda cumplir el tiempo de muestreo especificado.

- La necesidad de tener un modelo que represente fielmente la dinámica del sistema, puesto que nuestras predicciones las haremos a partir de él.

2.3 Problema de optimización

Suponiendo una definición del sistema en espacio de estados, en cada instante de muestreo, se plantea el problema de optimización siguiente:

$$\begin{aligned} &Min_u \quad V_N(x(0),u) \\ &s.a. \\ &\qquad x(k+1) = f(x(k),u(k)) \\ &\qquad x \in X \quad u \in U \end{aligned}$$

Donde V es la función de costes, la primera restricción hace referencia a que las predicciones evolucionen respecto al modelo y las últimas indican que tanto x como u deben permanecer contenidas en unos determinados conjuntos X y U, lo que indican restricciones de operación. Este problema puede ser reescrito de muchas maneras. Por ejemplo, para el caso de un sistema lineal donde únicamente existan restricciones de desigualdad quedará expresado de la siguiente manera:

$$\begin{aligned} &Min_u \quad V_N(x(0),u) \\ &s.a. \\ &\qquad x(k+1) = Ax(k) + Bu(k) \\ &\qquad x_{min} < x < x_{max} \\ &\qquad u_{min} < u < u_{max} \end{aligned}$$

Donde ahora el sistema está descrito por un conjunto de ecuaciones diferenciales lineales y los valores de x y u están acotados superior e inferiormente. Resuelto este problema, se aplica la primera acción de control y el resto se desechan (conforme a la estrategia de horizonte deslizante). Véase figura 2.1.

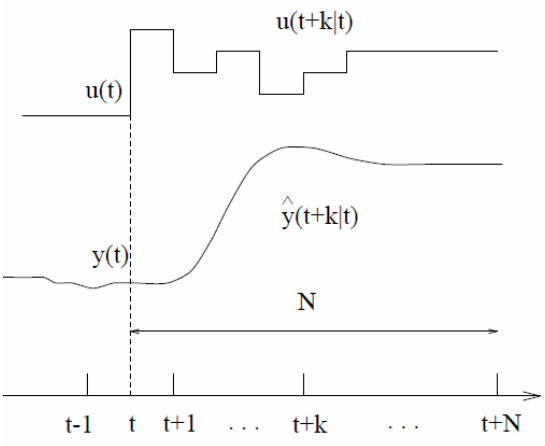

Figura 2.1 Estrategia de los controladores predictivos.

$$\hat{y}(t+k\,|\,t) \quad \text{Para } k = 1\ldots N \tag{2.1}$$
$$u(t+k\,|\,t) \quad \text{Para } k = 0\ldots N-1 \tag{2.2}$$

Donde $(t+k\,|\,t)$ implica los valores en el instante $t+k$ calculados en el instante t. Es muy común el uso de funciones de coste como la siguiente:

$$V_N(x(0),u) = \sum_{k=0}^{N-1} l(x(k),u(k)) + V_f(x(N)) \tag{2.3}$$

Siendo l y V_f el coste de etapa y el coste terminal respectivamente. Los cuales se pueden definir de la siguiente manera:

$$l(x(k),u(k)) = x(k)'Qx(k) + u(k)'Ru(k) \tag{2.4}$$
$$V_f(x(N)) = x(N)'Q_f x(N) \tag{2.5}$$

El coste de etapa y el coste terminal resultan de vital importancia a la hora de demostrar la estabilidad de un controlador predictivo, por lo que deben escogerse teniendo en cuenta una serie de factores para poder garantizar que se poseen determinadas propiedades. Sin embargo, es importante tener en cuenta un detalle antes de continuar.

2.4 Traslación de un sistema lineal

En las expresiones anteriores el objetivo del controlador es llevar el sistema desde un estado arbitrario $x(0)$ hasta el origen. Es decir, en caso de desear estabilizar el sistema en un estado distinto es necesario realizar un cambio de coordenadas de la manera:

$$\hat{x} = x - x_s \tag{2.6}$$
$$\hat{u} = u - u_s \tag{2.7}$$

Siendo x_s el estado estacionario del sistema para un conjunto de entradas u_s. Por tanto, el controlador tiende al origen respecto las coordenadas relativas y al punto deseado respecto las coordenadas absolutas. A continuación se demuestra que la definición del sistema no varía respecto a este cambio de coordenadas.

2.4.1 Caso continuo

Sea un sistema en espacio de estados definido por las ecuaciones siguientes:

$$\dot{x} = Ax + Bu \tag{2.8}$$

Se cumple para un punto estacionario (por definición) la siguiente relación:

$$Ax_s + Bu_s = 0 \tag{2.9}$$

Teniendo en cuenta que $\dot{\hat{x}} = \dot{x}$ y sustituyendo:

$$\dot{\hat{x}} = A(\hat{x} + x_s) + B(\hat{u} + u_s)$$
$$\dot{\hat{x}} = A\hat{x} + B\hat{u} + Ax_s + Bu_s$$

Aplicando la relación 2.9 obtenemos:

$$\dot{\hat{x}} = A\hat{x} + B\hat{u} \tag{2.10}$$

Por lo que el mismo sistema de ecuaciones define el sistema con el cambio de coordenadas definido en 2.6 y 2.7.

2.4.2 Caso discreto

El razonamiento es similar. El sistema queda definido por las siguientes ecuaciones:

$$x(k+1) = Ax(k) + Bu(k) \tag{2.11}$$

En este caso, para un punto estacionario se cumple:

$$x_s = Ax_s + Bu_s \tag{2.12}$$

Por lo que sustituyendo 2.6 y 2.7 en 2.11, se obtiene:

$$\hat{x}(k+1) + x_s = A(\hat{x}(k) + x_s) + B(\hat{u}(k) + u_s)$$
$$\hat{x}(k+1) = A\hat{x}(k) + B\hat{u}(k) + Ax_s + Bu_s - x_s$$

Teniendo en cuenta la relación 2.12:

$$\hat{x}(k+1) = A\hat{x}(k) + B\hat{u}(k) \tag{2.13}$$

Por lo que el sistema discretizado también admite traslaciones. Es importante tener en cuenta que esto se cumple para ambos casos en el supuesto de que el sistema sea lineal.

2.5 Horizontes de Control y Predicción diferentes

Como se especificó anteriormente, el Control Predictivo corresponde a una serie de ideas comunes con gran cantidad de posibles variantes o adiciones. Por poner un ejemplo, es posible diferenciar un horizonte de predicción y otro de control. Definamos el problema asociado de la siguiente manera:

$$\begin{aligned} &Min_u \quad V_{N_c,N_p}(x(0),u) \\ &s.a. \\ &\qquad \dot{x} = f(x,u) \\ &\qquad x \in X \quad u \in U \end{aligned}$$

Considerando ahora que N_c es el horizonte de control y N_p el horizonte de predicción. Cumpliéndose siempre que $N_p > N_c$. Convirtiéndose la función 2.14 en:

$$V_{N_c,N_p}(x(0),u) = \sum_{k=0}^{N_c-1} l(x(k),u(k)) + \sum_{k=N_c}^{N_p-1} l(x(k),\kappa_f(x(k))) + V_f(x(N_p)) \tag{2.14}$$

Donde $\kappa_f(x)$ es una ley de control calculada fuera de línea. De esta manera, es fácil ver que sólo unas cuantas acciones de control son calculadas al comienzo mientras que el resto son evaluadas con una función definida

anteriormente. Esto puede presentar diversas ventajas, como la reducción de la carga computacional para un mismo horizonte de predicción, el aumento del dominio de atracción, etc.

2.6 Estabilidad de controladores predictivos

Las definiciones que haremos a partir de ahora estarán referidas al origen. Para el caso en que se trate de un punto diferente será necesario realizar el cambio de coordenadas descrito en las expresiones 2.6 y 2.7.

2.6.1 Definiciones iniciales

Las consideraciones que se harán estarán basadas en los teoremas de Lyapunov, definidos en términos de funciones de clase K, K_∞ y KL:

- Una función $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ es de tipo K si es continua, $\alpha(0) = 0$ y es monótona creciente.
- Una función α es una función K_∞ si es de tipo K y $\alpha(s) = \infty$ cuando $s \to \infty$.
- Una función $\beta : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ es una función KL si para $t \in \mathbb{R}^+$, $\beta(\cdot,t)$ es K y para $s \in \mathbb{R}^+$, $\beta(s,t) \to 0$ cuando $t \to \infty$.

De igual manera diremos que un determinado conjunto $\mathbb{X}$ es un invariante positivo sí y sólo sí:

- Para todo $x(0) \in \mathbb{X}$, $x(k) \in \mathbb{X}$ para todo $k \geq 0$.
- Para todo $x(0) \in \mathbb{X}$ $x(k+1) = f(x(k)) \in \mathbb{X}$ para todo $k \geq 0$.

Además, supondremos que se trata de un conjunto cerrado que garantiza el cumplimiento de las restricciones de operación. Por tanto, suponiendo que $\mathbb{X}$ es un invariante positivo de $x^+ = f(x,u)$, una función $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ es considerada una función de Lyapunov en $\mathbb{X}$ para el sistema $f(\cdot)$ si existen tres funciones α_1 , α_2 *y* $\alpha_3 \in K$ tales que:

$$V(x) \geq \alpha_1(\|x\|) \quad \forall x \in \mathbb{X} \tag{2.15}$$

$$V(x) \leq \alpha_2(\|x\|) \quad \forall x \in \Omega \tag{2.16}$$

$$V(F(x)) - V(x) \leq -\alpha_3(\|x\|) \quad \forall x \in \mathbb{X} \tag{2.17}$$

Siendo Ω un subconjunto de $\mathbb{X}$.

Estos conceptos matemáticos pueden presentarse excesivamente abstractos, por lo que resulta interesante relacionarlos con otros que resulten más visuales.

- Podemos ver en la figura 2.2 como una función K no es más que una función monovariable cualquiera monótona creciente. Para el caso de K_∞ se va un poco más allá y se fuerza que la función tienda al infinito. Por último, una función es KL cuando para el caso en que la primera de sus variables se mantiene constante es una función K, mientras que al contrario se trata de una función decreciente que tiende a 0.
- Un invariante positivo no es más que una "zona" en la que una vez entra una trayectoria, esta no sale.
- Las restricciones de α_1 y α_2 implican que la funcion $V(x)$ se encuentra acotada tanto inferior como superiormente. Por otro lado, la última restricción impone que la función $V(x)$ debe decrecer más deprisa que otra determinada función α_3.

2.6.2 Suposiciones de partida

Para probar el conjunto de condiciones anterior en el caso de la función V_N^0 (donde el superíndice cero indica que se trata del óptimo), es necesario hacer previamente una serie de suposiciones[1]:

[1] Cabe destacar que para las demostraciones posteriores se definirá el modelo del sistema por la expresión $x^+ = f(x,u)$, lo cual es idéntico a $x(k+1) = f(x(k),u(k))$.

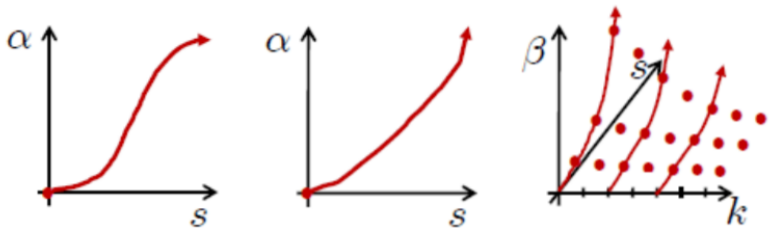

Figura 2.2 Funciones de tipo K ,K_∞ y KL [3].

- Para todo $x \in \mathbb{X}_f$, existe una u (tal que $u \in U$) de manera que se cumple:

$$f(x,u) \in \mathbb{X}_f \tag{2.18}$$
$$V_f(f(x,u)) - V_f(x) \leq -l(x,u) \tag{2.19}$$

- Existen funciones K α_1 y α_f tal que:

$$l(x,u) \geq \alpha_1(|x|) \quad \forall x \in \mathbb{X}_N, \forall u \in U \tag{2.20}$$
$$V_f(x) \leq \alpha_f(|x|) \quad \forall x \in \mathbb{X}_f \tag{2.21}$$

2.6.3 Demostración

Una vez dicho esto, pasamos a la demostración. Esta se dividirá en 3 partes. Una por cada una de las restricciones impuestas. La función elegida para ser función de Lyapunov será $V(x) = V_N^0(x)$, donde $V_N^0(x)$ es la función objetivo con los valores de u óptimos. Es decir, el mínimo valor que puede alcanzar $V_N(x)$.

- **Límite inferior**. La cual se corresponde con la ecuación 2.15. Esta puede obtenerse fácilmente ya que por la propia definición de V_N^0 se tiene:

$$V_N^0 \geq l(x,\, \kappa_N(x)) \quad \forall x \in \mathbb{X}_N$$

  Y gracias a la suposición 2.20 se cumple perfectamente. Además, es posible satisfacer esta suposición sin muchas complicaciones, por ejemplo, eligiendo la función de la forma 2.4 y eligiendo las matrices Q y R de manera que sean definidas positivas

- **Decrecimiento de** V_N^0. La cual se corresponde con la ecuación 2.17. Hacemos el salto a la tercera debido a que los resultados aquí obtenidos serán necesarios para la 2.16. Definamos V_N^0 de manera que:

$$V_N^0 = V_N(x, u^0(x))$$

  Donde:

$$u^0(x) = (u^0(0;x), u^0(1;x), \ldots, u^0(N-1;x))$$

  Es una secuencia de control minimizante. La secuencia de estados resultante de aplicar dichas acciones de control será:

$$x^0(x) = (x^0(0;x), x^0(1;x), \ldots, x^0(N;x))$$

  Donde $x^0(0;x) = x$ y $x^0(1;x) = x^+$. Es decir, el estado sucesor de x es $x^+ = f(x, \kappa_N(x)) = x^0(1;x)$. De igual manera:

$$V_N^0(x^+) = V_N^0(x^+, u^0(x^+))$$

  Y también:

$$u^0(x^+) = (u^0(0;x^+), u^0(1;x^+), \ldots, u^0(N-1;x^+))$$

Nuestra meta más inmediata consiste en comparar las funciones $V_N^0(x)$ y $V_N^0(x^+)$ con el objetivo de estudiar su comportamiento. Por tanto, elijamos una determinada $\check{u}$ factible para poder comparar $V_N^0(x)$ con $V_N^0(x^+,\check{u})$:

$$\check{u} = (u^0(1;x),...,u^0(N-1;x),u)$$

Donde aún es necesario elegir el valor u. La evolución del estado $\check{x}$ gracias a la secuencia $\check{u}$ será:

$$\check{x} = (x^0(1;x),x^0(2;x),...,x^0(N;x),f(x^0(N;x),u))$$

Es fácil comprobar que las secuencias $\check{u}$ y $u(\cdot)$ poseen ambas los términos $1,2,...,N-1$ (pero no 0 y N). Esto ocurre de igual manera con las secuencias x^0 y $\check{x}$. Por tanto:

$$V_N(x^+,\check{u}) = \sum_{j=1}^{N-1} l(x^0(j;x),u^0(j;x)) + l(x^0(N;x)) + V_f(f(x^0(N;x),u))$$

$$V_N^0(x) = l(x,\kappa_N(x)) + \sum_{j=1}^{N-1} l(x^0(j;x),u^0(j;x)) + V_f(x^0(N;x))$$

De manera que:

$$\sum_{j=1}^{N-1} l(x^0(j;x),u^0(j;x)) = V_N^0(x) - l(x,\kappa_N(x)) - V_f(x^0(N;x))$$

Sustituyendo en las expresiones anteriores:

$$V_N(x^+,\check{u}) = V_N^0(x) - l(x,\kappa_N(x)) - V_f(x^0(N;x)) + l(x^0(N;x)) + V_f(f(x^0(N;x),u))$$

Gracias a la suposición 2.19, tenemos que:

$$l(x^0(N;x)) + V_f(f(x^0(N;x),u)) - V_f(x^0(N;x)) \leq 0$$

Por lo que llegamos a:

$$V_N(x^+,\check{u}) - V_N^0(x) \leq -l(x,\kappa_N(x))$$

Debido a que $V_N^0(x^+) \leq V_N(x^+,\check{u})$, se llega a la siguiente conclusión:

$$V_N^0(f(x,\kappa_N(x))) - V_N^0(x) \leq -l(x,\kappa_N(x)) \tag{2.22}$$

- **Límite superior**. El cual se corresponde con la ecuación 2.16. En primer lugar, se propone la demostración de la siguiente afirmación:

$$V_{j+1}^0 \leq V_j^0 \quad \forall x \in X_j \tag{2.23}$$

Para ello, partamos del caso más sencillo, correspondiente a V_0^0 y V_1^0. Ambos costes quedan definidos de la siguiente manera:

$$V_0^0 = V_f(x)$$
$$V_1^0 = V_f(f(x,u)) + l(x,u)$$

Restando ambas:

$$V_1^0 - V_0^0 = V_f(f(x,u)) - V_f(x) + l(x,u)$$

Gracias a la suposición 2.19, llegamos a que:

$$V_1^0 - V_0^0 \leq 0$$

Para el caso de $V_{j+1}^0 \leq V_j^0$:

$$V_j^0 = \sum_{i=0}^{j-1} l(x^0(i,x),u^0(i,x)) + V_f(x^0(j;x))$$

Restando de igual manera que hicimos anteriormente nos queda:

$$V_{j+1}^0 - V_j^0 = l(x^0(j,x),u^0(j,x)) - V_f(x^0(j;x)) + V_f(x^0(j+1;x))$$

Por lo que aplicando las suposiciones de igual manera que en el caso anterior obtenemos:

$$V_{j+1}^0 - V_j^0 \leq 0$$

De los resultados anteriores es fácil darse cuenta de que $V_j^0(x) \leq V_f(x)$ para todo $x \in X_f$. Por tanto, gracias a la suposición 2.21, existe límite superior para V_N^0 en X_f.

Por tanto, hemos demostrado la verificación de las desigualdades con unas suposiciones de partida fácilmente alcanzables para el caso de la función $V_N^0(x)$. Resumiendo, el resultado es el siguiente:

$$V_N^0(x) \geq l(x,u) \geq \alpha_1(\|x\|) \quad \forall x \in X_N \tag{2.24}$$

$$V_N^0(x) \leq V_f(x,u) \leq \alpha_2(\|x\|) \quad \forall x \in X_f \tag{2.25}$$

$$V_N^0(f(x,\kappa_N(x))) - V_N^0(x) \leq -l(x,\kappa_N(x)) \leq -\alpha_1(\|x\|) \quad \forall x \in X_N \tag{2.26}$$

Donde X_N es el dominio de atracción y X_f es conocida como la región terminal. Es interesante saber que los puntos en el interior del dominio de atracción son aquellos en los que el problema de optimización es factible. Estos resultados son válidos en el caso de que exista un conjunto U acotado y el modelo de predicción del sistema sea idéntico al real.

2.6.4 Estabilidad KL

Es posible a su vez avanzar en algunos casos hasta la definición de estabilidad KL AS si existe una función que satisfaga la desigualdad siguiente:

$$\|x\| \leq \beta(\|x(0)\|,k) \quad \forall x \in \mathbb{X}$$

Esta noción de estabilidad resulta más ventajosa que la anterior por diversas razones, entre ellas podríamos destacar el hecho de que proporciona una cota a la evolución del estado. Sin embargo, son necesarias unas condiciones más restrictivas para poder garantizarla. En concreto, necesitamos que:

- α_1 sea una función de tipo K_∞.
- $V(x)$ esté acotada localmente en $\mathbb{X}$.

Es fácil garantizar la primera de ellas con una elección adecuada de $l(x,u)$ y α_1. Para la segunda, será necesario asumir que $x \in X$ y $u \in U$. Es decir, que tanto x como u pertenecen a un conjunto acotado convexo. De igual manera, asumiremos que $f(x,u)$, $l(x,u)$ y $V_f(x)$ son funciones convexas. Cumpliéndose estas suposiciones, podemos garantizar que la función V_N^0 estará acotada en X_N.

3 Descripción del Sistema

Una vez definidas las estrategias de control a usar en el ámbito de este proyecto, es necesario a su vez definir el banco de pruebas (*benchmark*) donde se va a comprobar la validez de la propuesta definida en el capítulo 5 en comparación con el controlador MPC ”clásico”.

Como se ha especificado en el capítulo 2 a modo de introducción al Control Predictivo, es necesario definir algún modelo matemático que proporcione suficiente información acerca de la dinámica del sistema a controlar. Llegados a este punto, resulta necesario definir con anterioridad **qué** sistema se pretende controlar. El sistema que trataremos en mayor profundidad se describe a continuación.

3.1 Elección del sistema

De igual manera que se hizo en trabajos anteriores, se elige la planta de los 4 tanques por las peculiaridades de su dinámica. Este sistema fue presentado por K.H. Johansson en [4] y resultó bastante aceptado por la comunidad científica como *benchmark* de algoritmos de control por sus características internas. Un esquema de la planta puede visualizarse en la figura 3.1.

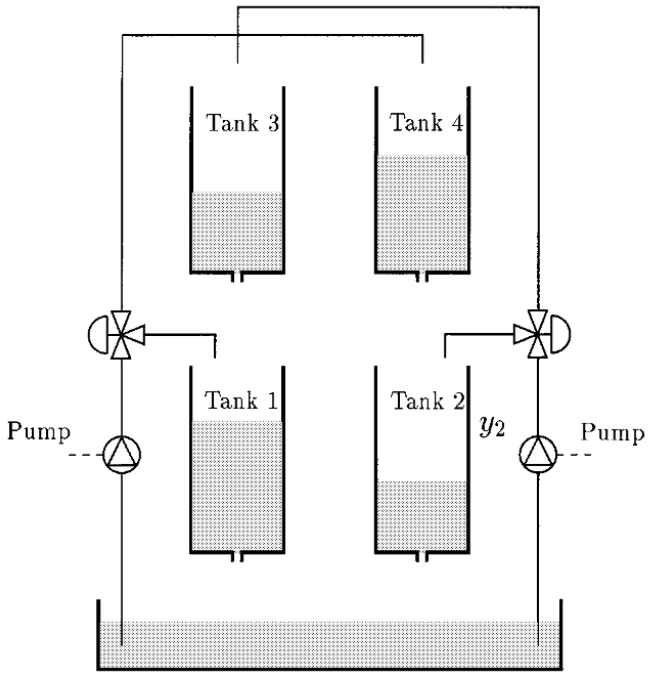

Figura 3.1 Representación gráfica de la planta de 4 tanques.

Se trata por tanto de dos tanques que descargan por gravedad en otros dos inferiores. Además, hay dos bombas en la parte más baja de la instalación que proporcionan los caudales desde un depósito inferior. Cabe destacar que la totalidad de estos caudales no se dirige a los tanques superiores sino que existen unas válvulas de 3 vías que dividen el flujo entre los superiores e inferiores. De hecho, el sistema presenta ceros de fase no mínimo en función de los valores de las aperturas de estas válvulas. Puede verse así mismo que se trata de un problema donde las entradas se encuentran acopladas.

Por tanto, el control quedará definido como el problema de mantener la altura de los tanques inferiores en unos determinadas niveles a los que denominamos *referencia*. Puesto que únicamente existen 2 bombas en la parte inferior, es decir, 2 entradas, no es posible controlar la altura de todos los tanques simultáneamente.

3.2 Descripción matemática

La descripción matemática del sistema viene descrita por el siguiente conjunto de ecuaciones diferenciales no lineales:

$$A\frac{dh_1}{dt} = -a_1\sqrt{2gh_1} + a_3\sqrt{2gh_3} + \gamma_a\frac{q_a}{3600} \tag{3.1}$$

$$A\frac{dh_2}{dt} = -a_2\sqrt{2gh_2} + a_4\sqrt{2gh_4} + \gamma_b\frac{q_b}{3600} \tag{3.2}$$

$$A\frac{dh_3}{dt} = -a_3\sqrt{2gh_3} + (1-\gamma_b)\frac{q_b}{3600} \tag{3.3}$$

$$A\frac{dh_4}{dt} = -a_4\sqrt{2gh_4} + (1-\gamma_a)\frac{q_a}{3600} \tag{3.4}$$

Donde A corresponde a la sección de los tanques en m^2, a_j corresponde a los orificios de descarga del tanque j, g la gravedad, h_j la altura de la columna de agua en el tanque j, γ_a y γ_b las aperturas de las válvulas de 3 vías y por último, q_a y q_b las entradas del sistema (caudales en $\frac{m^3}{s}$).

Es fácil ver en las ecuaciones cómo afecta el valor de las aperturas de las válvulas a los caudales y cómo se encuentran acoplados entre ellos.

Una vez se ha definido el sistema a controlar, necesitamos establecer el modelo que vamos a utilizar y cómo se identifica dicho modelo. En nuestro caso, hemos optado por una definición en espacio de estados de la forma:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \tag{3.5}$$

$$y(t) = C(t)x(t) + D(t)u(t) \tag{3.6}$$

Lo cual se simplifica de la siguiente manera para sistemas invariantes en el tiempo:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.7}$$

$$y(t) = Cx(t) + Du(t) \tag{3.8}$$

Donde x representa el vector de estados, $\dot{x}$ la derivada del estado, u las entradas, y las salidas y A,B,C,D son las matrices que definen el sistema. Sin embargo, las ecuaciones que acabamos de presentar no se corresponden con las de un sistema de dinámica no lineal. Es decir, será necesario linealizar previamente en torno a un punto de funcionamiento determinado. Definiendo las variables incrementales siguientes:

$$\Delta h_j = h_j - \overline{h}_j \qquad \forall j = 1...4 \tag{3.9}$$

$$\Delta u_a = u_a - \overline{u}_a \tag{3.10}$$

$$\Delta u_b = u_b - \overline{u}_b \tag{3.11}$$

Y linealizando en torno al punto de operación $\overline{h}$ las ecuaciones anteriores quedan de la siguiente manera:

$$\Delta\dot{h_1} = -\frac{a_1}{A}\sqrt{\frac{g}{2\overline{h}_1}}\Delta h_1 + \frac{a_3}{A}\sqrt{\frac{g}{2\overline{h}_3}}\Delta h_3 + \gamma_a\frac{\Delta q_a}{3600A} \tag{3.12}$$

$$\Delta\dot{h_2} = -\frac{a_2}{A}\sqrt{\frac{g}{2\overline{h}_2}}\Delta h_2 + \frac{a_4}{A}\sqrt{\frac{g}{2\overline{h}_4}}\Delta h_4 + \gamma_b\frac{\Delta q_b}{3600A} \tag{3.13}$$

$$\Delta\dot{h_3} = -\frac{a_3}{A}\sqrt{\frac{g}{2\overline{h}_3}}\Delta h_3 + (1-\gamma_b)\frac{\Delta q_b}{3600A} \tag{3.14}$$

$$\Delta\dot{h_4} = -\frac{a_4}{A}\sqrt{\frac{g}{2\overline{h}_4}}\Delta h_4 + (1-\gamma_a)\frac{\Delta q_a}{3600A} \tag{3.15}$$

Expresadas matricialmente:

$$A = \begin{bmatrix} -\frac{a_1}{A}\sqrt{\frac{g}{2\overline{h}_1}} & 0 & \frac{a_3}{A}\sqrt{\frac{g}{2\overline{h}_3}} & 0 \\ 0 & -\frac{a_2}{A}\sqrt{\frac{g}{2\overline{h}_2}} & 0 & \frac{a_4}{A}\sqrt{\frac{g}{2\overline{h}_4}} \\ 0 & 0 & -\frac{a_3}{A}\sqrt{\frac{g}{2\overline{h}_3}} & 0 \\ 0 & 0 & 0 & -\frac{a_4}{A}\sqrt{\frac{g}{2\overline{h}_4}} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{\gamma_a}{3600A} & 0 \\ 0 & \frac{\gamma_b}{3600A} \\ 0 & \frac{(1-\gamma_b)}{3600A} \\ \frac{(1-\gamma_a)}{3600A} & 0 \end{bmatrix}$$

Sin embargo, en Control Predictivo es más habitual el uso de modelos discretos, trabajando con un determinado horizonte de predicción N que define un número finito de instantes de muestreo. Por tanto, la forma deseada para el modelo es la siguiente:

$$x(k+1) = Ax(k) + Bu(k) \tag{3.16}$$

$$y(k) = Cx(k) + Du(k) \tag{3.17}$$

El cual se puede calcular fácilmente a partir de las matrices anteriores una vez definido el tiempo de muestreo T y asumiendo un mantenedor de orden cero:

$$x((k+1)T) = G(T)x(kT) + H(T)u(kT) \tag{3.18}$$

$$y(kT) = Cx(kT) + Du(kT) \tag{3.19}$$

Donde:

$$G(T) = e^{AT}$$

$$H(T) = (\int_0^T e^{A\lambda}d\lambda)B$$

4 Sequential Monte Carlo MPC

Los métodos secuenciales de Monte Carlo (también conocidos habitualmente como filtros de partículas) son un tipo de algoritmo aleatorio usado generalmente para resolver problemas de filtrado pertenecientes a diversos ámbitos como el procesamiento de señales, la estimación del estado de robots, etc.

El objetivo de este tipo de problemas consiste en la estimación del estado interno de un sistema dinámico a partir de observaciones parciales, teniendo en cuenta la presencia de perturbaciones aleatorias tanto en los sistemas de medida como en el propio sistema. Para ello, se calcula la distribución probabilidad condicionada de los sucesos aleatorios correspondientes (que se consideran procesos Markovianos), a partir de la cual puede elegirse la solución que presenta la mayor verosimilitud.

4.1 Definiciones preliminares

Dicho esto, resulta de interés definir formalmente algunos de los conceptos esbozados anteriormente:

- Un proceso Markoviano es un tipo de proceso estocástico discreto en el que la probabilidad de que ocurra un determinado evento depende únicamente del evento anterior. Es decir, definimos un determinado vector aleatorio ξ donde en cada instante k tendrá asociado un valor ξ_k y tendrá asociada una observación ς_k. Por tanto, ξ es un proceso Markovianos si cumple que:

$$p(\xi_k \mid \xi_1, \xi_2, \dots, \xi_{k-1}) = p(\xi_k \mid \xi_{k-1}) \tag{4.1}$$
$$p(\varsigma_k \mid \xi_k) = f(\xi_k) \tag{4.2}$$

  Es decir, la probabilidad de la observación $p(\varsigma_k \mid \xi_k)$ depende únicamente de ξ_k.

- La probabilidad condicionada de un suceso se define como la posibilidad de que ocurra un determinado suceso A dado que sucede otro suceso B. Esta se puede calcular mediante la expresión:

$$p(A \mid B) = \frac{p(A \bigcap B)}{p(B)} \tag{4.3}$$

  O bien mediante el teorema de Bayes:

$$p(A \mid B) = \frac{p(B \mid A)}{p(B)} \tag{4.4}$$

A modo de resumen, buscamos hallar la estimación óptima de ξ_k a partir de las observaciones actuales ς_k.

4.2 Uso del algoritmo

Sin embargo, el uso que queremos hacer del algoritmo se desvía un poco de su concepción original. En nuestro caso, el estado oculto ξ_k pasa a ser la secuencia de acciones de control u^* mientras que las observaciones ς_k

se corresponden con el coste $V_{N_c,N_p}(x_k,u_k)$ junto a las predicciones $\hat{x}_{k:k+N_p}$. Es necesario escoger un número de partículas L suficientemente grande para aproximar de manera adecuada la distribución de probabilidad de ξ_k. Esto dependerá en gran medida de la dimensión del problema, junto a otros factores. Para un estudio en mayor profundidad acerca del crecimiento del problema puede consultarse [7].

Por tanto, vamos a usar el algoritmo para resolver un determinado problema de optimización de igual manera que podrían usarse otro tipo de técnicas: solvers, algoritmos genéticos, etc. Cabe destacar que la solución encontrada por el algoritmo probablemente no sea la óptima, pero se encontrará en la vecindad de esta con una tolerancia ε suficientemente pequeña. Definiendo $\breve{u}$ como la solución proporcionada por el algoritmo:

$$\breve{u} \pm \varepsilon = u^* \tag{4.5}$$

En función de diversos parámetros (entre los cuáles el más destacable sería el número de iteraciones), se puede reducir más o menos la brecha existente con la solución óptima. La idea de usar métodos de Montecarlo

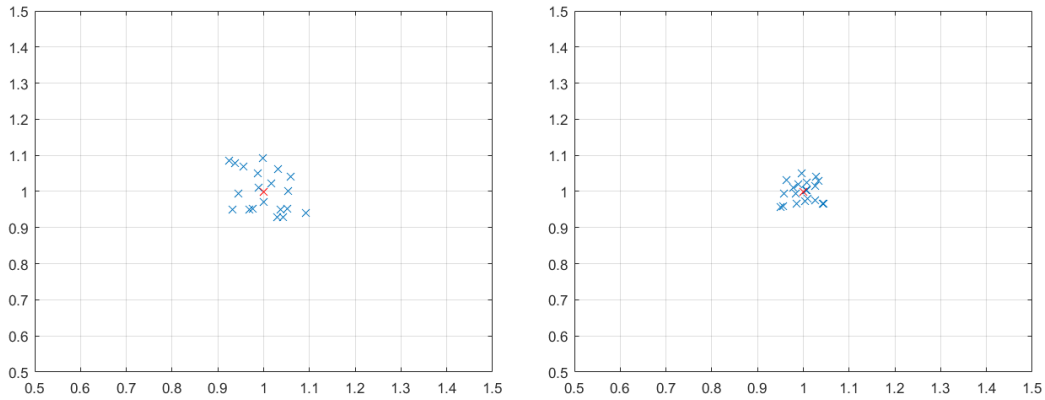

Figura 4.1 Población de partículas entorno al óptimo para diferente número de iteraciones. Puede verse como el paso de las iteraciones tiene un efecto beneficioso para la convergencia.

para resolver problemas de Control Predictivo ha aparecido anteriormente en otras publicaciones como [6] en el contexto de la generación de trayectorias de aviones en proceso de acercamiento a un aeropuerto. Sin embargo, el problema a resolver no será estrictamente similar al expuesto en capítulos anteriores sino que recibirá una cierta modificación con el objetivo de mejorar el funcionamiento del sistema.

4.3 Descripción del algoritmo

Existen varios conceptos fundamentales que es necesario aclarar y comprender para avanzar a través de las páginas siguientes:

- Una *partícula* representa una solución candidata al problema MPC planteado. Es decir, cada partícula posee una secuencia de acciones de control para todo el horizonte de predicción N_p. Conforme aumenta la complejidad del problema (por ejemplo al aumentar el horizonte de control N_c), el número de partículas necesarias para obtener una solución relativamente buena también lo hará. Además de la secuencia de acciones de control, cada partícula tiene definidas otras variables como su peso (indica la bondad de la partícula respecto al resto) y las trayectorias predichas. A partir de ahora, el número de partículas se definirá con la letra L.
- Un *escenario* corresponde a una realización posible de la incertidumbre que afecta a la dinámica del sistema a controlar. Es fácilmente comprobable que un modelo lineal no es capaz de representar fielmente la dinámica completa de un proceso no lineal. En todo caso, es posible expresar con mayor o menor fidelidad una región de este en torno el cual el sistema es linealizado. Por esta razón, suponemos que los elementos de las matrices A y B del sistema están sujetos a incertidumbres de valor acotado. Por

tanto, los escenarios serán generados en el subespacio formado por los coeficientes y sus respectivas tolerancias.

- El *remuestreo* es una técnica que permite la generación de nuevas partículas en aquellas zonas que presentan un mejor desempeño en la iteración anterior, tal como puede verse en la figura 4.2 y 4.3. El método utilizado es el de Kitagawa [11], más concretamente, el estratificado. Cabe destacar que las nuevas partículas no se generan únicamente alrededor de la partícula con mejor desempeño, sino que de manera progresiva se generan aglutinaciones de partículas en las zonas de mejor desempeño. Esto quiere decir que existen menos posibilidades de estancarse en un mínimo local en relación al caso en que únicamente eligiéramos la mejor de entre todas.

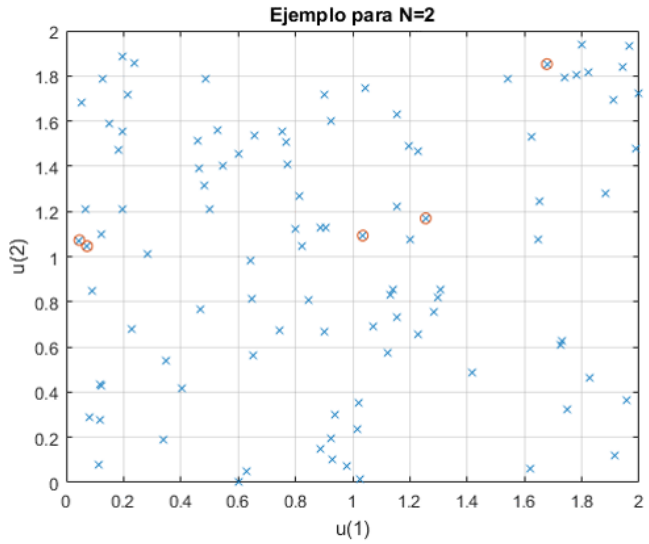

Figura 4.2 Generación aleatoria de partículas en R^2 . Se redondean a modo de ejemplo aquellas que presentan mejor desempeño.

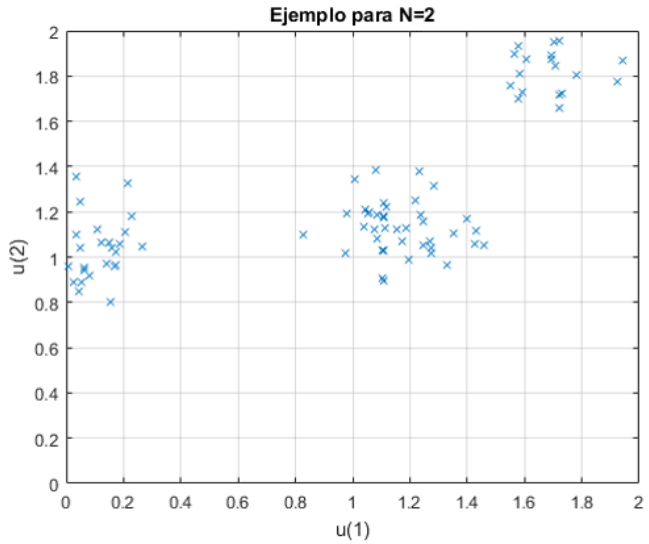

Figura 4.3 Generación de las partículas después de una iteración. Puede verse como las nuevas partículas se han generado en el entorno que presentaba mejor desempeño. A su vez, se han perturbado dentro de unos márgenes razonables para explorar nuevas soluciones.

- En cuanto a la *perturbación*, esta mueve las partículas en el espacio local de búsqueda con el objetivo de encontrar nuevas soluciones. Cada partícula es perturbada de manera independiente y aleatoria.

En realidad no es más que ruido blanco aditivo que va decreciendo conforme avanzan las iteraciones, de manera que se va convergiendo poco a poco a la solución final. Las acciones de control de una determinada partícula j se pueden escribir como el vector siguiente.

$$u^j = \begin{bmatrix} u_k^j \\ u_{k+1}^j \\ \vdots \\ u_{k+N_p-1}^j \end{bmatrix}$$

Por lo que la perturbación de una partícula j en la iteración z se puede escribir de la siguiente manera:

$$u_z^j = u_{z-1}^j + \frac{\check{r}}{e_z} \tag{4.6}$$

Siendo $\check{r}$ una variable aleatoria que toma valores de una distribución normal y e_z queda definido de la manera siguiente:

$$e_z = \Gamma z \tag{4.7}$$

Siendo Γ una constante a la que es necesario asignar un valor y que será representativa de la agresividad de la perturbación. En iteraciones posteriores del algoritmo, dejaremos de considerar que z esté multiplicada por una constante.

- Cada partícula tiene asociado un *peso* que representa el desempeño de sus acciones de control. Una partícula con un peso muy cercano a 1 significa que tiene un gran desempeño. Por el contrario, una partícula con un peso muy cercano a cero tiene un desempeño muy malo e interesa que no vuelva a aparecer en las posteriores iteraciones. Por esta razón, en el remuestreo, se tiende a generar las nuevas partículas alrededor de aquellas que poseían un peso superior.

4.3.1 Descripción detallada

El algoritmo desarrollado queda resumido de la siguiente manera:

Algoritmo 1: Algoritmo de Montecarlo secuencial.

Datos: L (número de partículas), N_p , N_c (horizontes de predicción y control), $V(x,u)$ (función de coste).

```
Generar de manera aleatoria las acciones de control de cada partícula;
Inicializar los pesos de las partículas a 1/L;
repetir
    para todo escenario hacer
        Predecir la evolución del estado de cada partícula a lo largo del horizonte de predicción;
        Calcular los costes y el desempeño normalizado de cada partícula;
        Actualizar los pesos;
    fin
    Remuestreo y perturbación de las partículas (acciones de control);
    Reinicio de los pesos de las partículas a 1/L;
hasta que se cumpla la condición de parada;
Escoger la partícula que presenta un mejor desempeño;
```

Dicho esto, pasamos a explicar en profundidad cada una de las etapas:

1. Para cada partícula j, se genera la secuencia de acciones de control u^j de dimensión N_p donde los primeros términos N_c son generados de manera aleatoria en los límites de actuación mientras que los restantes $(N_p - N_c)$ siguen una ley de control $\kappa_f(x)$ definida fuera de línea.
2. El peso de cada partícula, representado por W^j, es iniciado a un mismo valor que el resto de manera que la suma de todos ellos es la unidad.
3. Aquí comienza el bucle en z, donde cada iteración culmina con el remuestreo y perturbación de la población. La idea es que durante las primeras iteraciones se explore lo máximo el espacio de soluciones

con una resolución baja. Conforme avanzan las iteraciones se pretende ir afinando cada vez más hasta que se llega a una solución suficientemente buena.

4. En este bucle anidado se realizan las operaciones correspondientes a cada escenario de ruido. Para ello, un escenario es generado dentro de los límites establecidos por las incertidumbres de las matrices del sistema lineal. Cabe destacar que al hacer esto de manera aleatoria, es posible estar generando escenarios con cambios muy diversos de manera simultánea: movimiento de ceros y polos, cambios en la ganancia, etc. Para generar escenarios donde no aparezcan todos estos cambios al mismo tiempo, debería seguirse un procedimiento más heurístico.

5. Para cada escenario de ruido y partícula, se predice la evolución del sistema conforme a la secuencia de control generada en la línea 1.

6. A partir de las predicciones hechas en la línea 5, puede calcularse el coste $V^j(x,u)$ asociado a cada partícula **y** escenario de ruido. Una vez hecho esto, es posible normalizar los costes anteriores de manera que su valor esté comprendido entre cero y uno. Esta normalización sigue la expresión siguiente:

$$\hat{V}^j = \frac{V^j - V_{min}}{V_{max} - V_{min}} \tag{4.8}$$

Donde V_{min} y V_{max} corresponden al menor y mayor coste de la población respectivamente. Es decir, a la partícula con menor coste se le asigna un coste nulo y a la que tiene el mayor coste se le asigna la unidad. Por otro lado, el desempeño tiene la tendencia contraria. Deseamos que valga uno para la mejor partícula y cero para la peor. Esto nos lleva a la siguiente transformación:

$$P^j = 1 - \hat{V}^j \tag{4.9}$$

Donde P^j representa el desempeño de la partícula j. Es fácil darse cuenta de que es posible sustituir la expresión 4.8 en 4.9 dando como resultado:

$$P^j = \frac{V_{max} - V^j}{V_{max} - V_{min}} \tag{4.10}$$

Esta última ecuación nos permite calcular el desempeño sin necesidad de pasar por la ecuación 4.8.

7. El siguiente paso corresponde a la actualización de los pesos asociados a cada partícula. La manera más sencilla de realizar esta actualización es la siguiente:

$$\tilde{W}^j = P^j \cdot W^j \tag{4.11}$$

Una vez actualizados los pesos con los desempeños correspondientes, estos son normalizados acorde a la siguiente expresión:

$$W^j = \frac{\tilde{W}^j}{\sum_{j=1}^{L} \tilde{W}^j} \tag{4.12}$$

Por lo que la suma de todos ellos vuelve a ser uno al comienzo de cada iteración. El hecho de que todos sumen uno resulta de especial importancia a la hora del remuestreo y tiene connotaciones estadísticas respecto a las funciones de probabilidad.

8. Después de haber hecho las operaciones anteriores para todos los escenarios, tenemos unos pesos que guardan información sobre el desempeño de cada partícula en cada uno de los escenarios propuestos.

9. Es necesario seguir varios pasos para realizar el remuestreo estratificado de Kitagawa:

$$\alpha^j = \sum_{k=0}^{j} \bar{W}^k \tag{4.13}$$

$$v^j = \frac{j-1}{L} + U \tag{4.14}$$

Si nos paramos un poco en la definición del algoritmo y sus expresiones nos daremos cuenta de por qué las partículas con mejor desempeño son remuestreadas con mayor probabilidad que el resto. Es fácil

Algoritmo 2: Remuestreo estratificado de Kitagawa

Datos: W (vector de pesos)

1 Reordenar el vector W en orden ascendente. El resultado viene expresado por $\bar{W}$;
2 Se calcula el acumulado de $\bar{W}$. Este se almacena en un vector α (Ecuación 4.13);
3 Se define un número aleatorio $U \in (0, \frac{1}{L})$;
4 **para todo** j **hacer**
5 Se define el vector v a partir de la ecuación 4.14 ;
6 Se busca el i que satisface $\alpha^i < v^j < \alpha^{i+1}$;
7 Las nuevas partículas quedarán determinadas de manera que $u^j_{z+1} = \bar{u}^i_z$;
8 **fin**

darse cuenta de que una partícula con buen desempeño tendrá un peso grande en comparación al resto. Esto implica que en la distribución acumulada α tendrá una "banda" muy ancha para la componente de esa partícula en concreto, por lo que muchos valores del vector v cumplirán la relación de la línea 6, por lo que serán remuestreadas en esa zona. Una vez realizado el remuestreo, se pasa a perturbar las

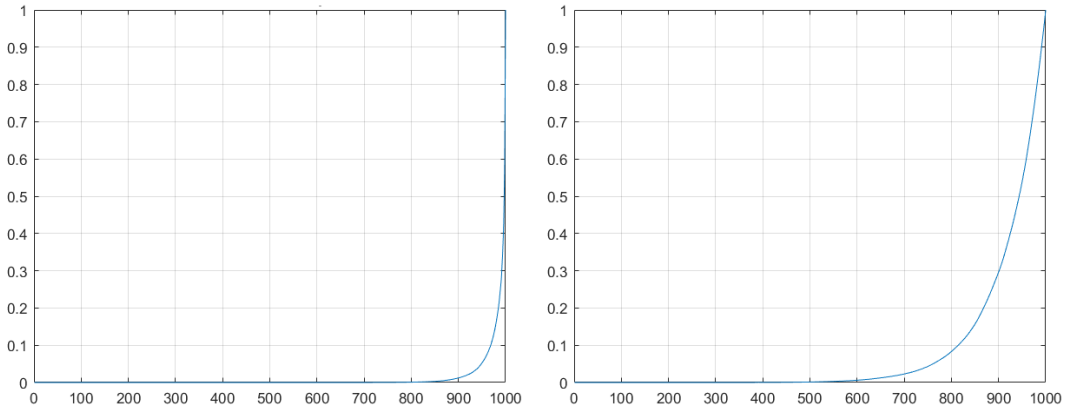

Figura 4.4 Ejemplos de funciones de distribución acumulada. En la primera, sólo unas pocas partículas tienen un desempeño aceptable. Por otro lado, en la segunda esta situación se encuentra un poco más equilibrada.

partículas.

10. Al remuestrear y perturbar las partículas, estas han cambiado su valor (podría decirse realmente que son nuevas), por lo que los pesos ya no son indicativos de su desempeño en cada uno de los escenarios. Por esta razón es necesario reiniciarlas nuevamente.

4.4 Algoritmo con trasplante

Una pequeña adición al algoritmo básico presentado es la idea del "trasplante". Esta consiste en considerar en el siguiente instante de muestreo $k+1$ algunas de las mejores partículas del instante k. Como se explicó en el capítulo 2, en el problema de Control Predictivo se calcula una secuencia completa de acciones de control pero sólo la primera de ellas es introducida. Sin embargo, parece lógico mantener algunas de las partículas anteriores desplazando la secuencia anterior en un instante y añadiendo un último término igual que aquel que era el último termino en el instante anterior. Es decir:

$$u^l_{k+1} = \begin{bmatrix} u^b(k+1\,|\,k) \\ \vdots \\ u^b(k+N\,|\,k) \\ u^b(k+N\,|\,k) \end{bmatrix}$$

Donde l representa un subconjunto del conjunto total de partículas L sobre las que se realiza el trasplante y b otro subconjunto de L que corresponde con las mejores partículas del instante k. Cabe destacar que las partículas trasplantadas pueden ser objetivo de remuestreos, pero no son modificadas.

4.4.1 Algoritmo adaptativo con trasplante

De igual manera, se plantea otra variación adicional donde la expresión de e_z se va a cambiar para que tenga una sinergia mayor con el transplante.

$$e_z = \frac{z}{\delta + \|x_k - x_k^{ref}\|} \tag{4.15}$$

Donde δ es una constante de valor muy pequeño cuyo objetivo es evitar que cuando la función tenga que evaluarse en el caso de $x = x_{ref}$ exista una indeterminación. Debe de ser de valor muy pequeño puesto que de lo contrario perturbaría en exceso la función. Resulta de interés estudiar el comportamiento de esta función para unos determinados conjuntos de valores:

- Cuando el estado actual sea muy parecido a la referencia el valor de la función vendrá dado por δ, lo cual implica un valor de e_z grande y por tanto, una perturbación pequeña. Esto tiene sentido puesto que cuando nos encontramos muy cerca de la referencia no necesitamos cambiar en gran medida la solución del instante anterior trasplantada.
- En el caso de que la diferencia sea grande, la función tomará un valor pequeño, el cual implica perturbaciones grandes. Al igual que razonábamos anteriormente, esto tiene sentido debido a que en el caso de estar muy lejos de la referencia conviene explorar lo máximo posible el espacio de soluciones.

Por tanto, el algoritmo 1 se verá modificado de la siguiente manera:

Algoritmo 3: Algoritmo de Montecarlo secuencial transplantado.

Datos: L (número de partículas), N_p , N_c (horizontes de predicción y control), $V(x,u)$ (función de coste), L_t (número de partículas a transplantar).

```
Transplantar un número de partículas L_t que presentaron mejor desempeño en el instante anterior;
Generar de manera aleatoria las acciones de control de las nuevas partículas (no pertenecientes a L_t);
Inicializar los pesos de las partículas a 1/L;
repetir
    para todo escenario hacer
        Predecir la evolución del estado de cada partícula a lo largo del horizonte de predicción;
        Calcular los costes y el desempeño normalizado de cada partícula;
        Actualizar los pesos;
    fin
    Remuestreo teniendo en cuenta todas las partículas;
    Perturbación de las partículas no pertenecientes a L_t;
    Reinicio de los pesos de las partículas a 1/L;
hasta que se cumpla la condición de parada;
Escoger la partícula que presenta un mejor desempeño;
```

4.5 Modificación del algoritmo

Sin embargo, también se propone una modificación del algoritmo a la vista de los primeros resultados obtenidos, los cuáles se presentan en el capítulo 5.

Aunque ahora mismo suponga un adelanto respecto al capítulo en cuestión, basta con saber que el funcionamiento del SMC será prácticamente idéntico al del controlador MPC lineal. Suponemos que esto se debe a que la ponderación de todos los escenarios es la misma, por lo que al final, el resultado es la media de todos

ellos, lo que nos lleva al sistema de partida.

Para tratar de solucionar este inconveniente, se proponen los siguientes cambios:

- Los *escenarios* no serán generados de manera aleatoria. En su lugar, se realizarán una serie de ensayos en la planta para diferentes puntos de operación, de manera que se muestree de manera suficiente el subespacio correspondiente a las acciones de control. Esto conlleva la identificación de un determinado número de modelos lineales que tendremos almacenados a modo de "base de datos".
- Aparece una *función de pertenencia* para evaluar la ponderación de cada uno de los escenarios. Este tipo de razonamiento tiene cierta similitud con algunos conceptos de lógica borrosa [8]. A modo de resumen, la función de pertenencia nos indicará cómo de "cerca" se encuentra nuestra referencia actual en relación a los diferentes modelos linealizados presentes en la base de datos. De esta manera, podremos dar mayor peso a los que se encuentren en el entorno más cercano. La función de pertenencia a utilizar es de tipo gaussiano, la cual es muy común en la literatura de modelado y control borroso [9]. La definición de esta función viene dada por la ecuación siguiente:

$$\mu(x) = e^{-\frac{(c-x)^2}{2\sigma^2}} \tag{4.16}$$

Donde c y σ son similares a los presentes a una distribución normal (centro y desviación típica). En la

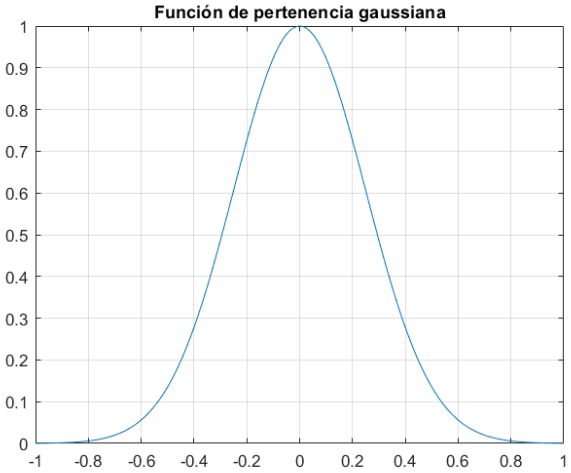

Figura 4.5 Función de pertenencia gaussiana para el caso concreto de $\sigma = 0.25$ y $c = 0$.

figura 4.4 puede verse un ejemplo de este tipo de función. Es fácil ver que adquiere el valor uno cuando x coincide con el centro, muy cercanos a uno cuando se encuentra en el entorno cercano al centro y decae rápidamente conforme nos alejamos del centro. En nuestro caso, los valores alimentados a la función serán la distancia a las referencias, por lo que el centro pasa a ser nulo y sólo es necesario definir una única función de pertenencia, quedando de la siguiente manera:

$$\mu(\phi_i^k) = e^{-\frac{\phi_i^{k\,2}}{2\sigma^2}} \tag{4.17}$$

$$\phi_i^k = \| x_k^{ref} - x_i^{BD} \| \tag{4.18}$$

Siendo x_k^{ref} la referencia actual del controlador y x_i^{BD} el punto de funcionamiento del modelo linealizado i-ésimo.

- Como consecuencia de la aparición de la función de pertenencia, la *actualización de los pesos* debe de ser modificada. Suponiendo que hemos computado los valores de $\mu(\phi_i)\,\forall\, i \in BD$, los pesos se podrían actualizar siguiendo la expresión siguiente:

$$\tilde{W}^j = W^j + P^j \cdot \phi_i^k \tag{4.19}$$

Puede observarse como la función de pertenencia afecta a todas las partículas por igual. Esto tiene sentido debido a que este término depende únicamente de la referencia actual y del escenario que se esté calculando. A su vez, puede demostrarse que en caso de que el desempeño de todas las partículas sea el mismo para un mismo escenario con respecto al peso acumulado, este se mantendrá tal y como está.

$$W^j = \frac{\tilde{W}^j}{\sum_{j=1}^{L} \tilde{W}^j} = \frac{W^j + P^j \cdot \phi_i^k}{\sum_{j=1}^{L} W^j + P^j \cdot \phi_i^k}$$

Puesto que $W^j = P^j$:

$$W^j = \frac{W^j + W^j \cdot \phi_i^k}{\sum_{j=1}^{L} W^j + W^j \cdot \phi_i^k} = \frac{(1+\phi_i^k)W^j}{(1+\phi_i^k)\sum_{j=1}^{L} W^j} = \frac{W^j}{\sum_{j=1}^{L} W^j}$$

Como W^j se corresponde con los pesos acumulados sin modificar, se cumple que $\sum_{j=1}^{L} W^j$ y por tanto $W^j = W^j$. Sin embargo, esta definición de los pesos no resulta especialmente buena porque, en realidad, nos resulta mucho más favorable que los pesos cambien continuamente aunque su desempeño sea el mismo. Supongamos un caso sencillo en que tenemos partículas con un peso alto y otras con un peso bajo. Realmente, nos interesa que aquellas que tengan un peso alto tengan posteriormente uno todavía más alto mientras que las que tuvieran uno bajo sea todavía más bajo, puesto que esto ayudará a remuestrear las partículas más convenientes. Puede verse en la definición original 4.11, como se realiza esta actualización. Por tanto, el método propuesto viene dado por la expresión:

$$\tilde{W}^j = W^j + (P^j)^{n_{esc}} \cdot \phi_i^k \tag{4.20}$$

Siendo n_{esc} el número total de escenarios. De esta manera, la definición es más parecida a la original y habrá un orden magnitud de diferencia mucho más acusado entre las partículas.

- La *selección de los escenarios* vendrá dada por los valores de la función de pertenencia para todas las posibilidades de la base de datos. De esta manera, en lugar de escogerlos de manera aleatoria de esta (lo cual puede llevar en algunos casos a la evaluación de escenarios sin ningún interés) se escogen los que presenten mayor pertenencia.

Por lo que el algoritmo 3 queda de la siguiente manera:

Algoritmo 4: Algoritmo de Montecarlo secuencial ponderado transplantado.

Datos: L (número de partículas), N_p , N_c (horizontes de predicción y control), $V(x,u)$ (función de coste), L_t (número de partículas a transplantar), BD (base de datos de sistemas linealizados).

Transplantar un número de partículas L_t que presentaron mejor desempeño en el instante anterior;
Generar de manera aleatoria las acciones de control de las nuevas partículas (no pertenecientes a L_t);
Inicializar los pesos de las partículas a $\frac{1}{L}$;
Evaluación de la función de pertenencia ϕ_i^k;
repetir
    para todo *escenario* **hacer**
        Predecir la evolución del estado de cada partícula a lo largo del horizonte de predicción;
        Calcular los costes y el desempeño normalizado de cada partícula teniendo en cuenta ϕ_i^k;
        Actualizar los pesos;
    fin
    Remuestreo teniendo en cuenta todas las partículas;
    Perturbación de las partículas no pertenecientes a L_t;
    Reinicio de los pesos de las partículas a $\frac{1}{L}$;
hasta que *se cumpla la condición de parada*;
Escoger la partícula que presenta un mejor desempeño;

4.6 Implementación

Las tareas que se realizan completamente en la GPU son las siguientes:

- Predicción del estado futuro a partir de la secuencia de acciones de control para cada partícula.
- Cálculo de los costes para cada partícula.
- Escalado de los costes y actualización de los pesos para cada partícula.

Es fácil ver en los esquemas de los algoritmos mostrados anteriormente que existen 2 bucles secuenciales principales en el controlador. Por un lado, está el bucle que controla el remuestreo y perturbación, identificado con la letra z. Por otro lado, están las iteraciones para cada escenario de ruido. Es decir, para una determinada iteración de z y escenario de ruido se calcula al mismo tiempo en la GPU la evolución del estado, etc. Puesto que los resultados de cada escenario son prácticamente independientes entre sí, podría plantearse una paralelización más exhaustiva.

Aún así, con la implementación realizada, los tiempos de computación se reducen drásticamente, lo cual ha sido imprescindible para poder ejecutar la enorme cantidad de ensayos que han sido necesarios, que de otra manera hubieran requerido un tiempo aún más prohibitivo.

4.6.1 Comparativas de tiempos

Se adjuntan a continuación tablas que reflejan los tiempos necesarios para calcular la acción de control para un instante de muestreo cambiando los diferentes parámetros existentes. En todas ellas, el caso inicial se calcula con el conjunto de parámetros $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$.

Tabla 4.1 Variación de tiempos respecto a z.

	CPU	GPU
$z = 25$	0.78182	0.60376
$z = 50$	1.60199	0.96286
$z = 100$	2.87489	1.66800
$z = 250$	7.15867	3.78506
$z = 500$	14.05404	7.51823

Es fácil darse cuenta de que los tiempos crecen más rápidamente para la programación completamente secuencial que para el caso paralelizado. En realidad, las mejoras de velocidad no resultan enormes puesto que lo que estamos haciendo es aumentar las iteraciones de un bucle que aparece en ambas versiones.

Tabla 4.2 Variación de tiempos respecto a n_{ruido}.

	CPU	GPU
$n_{ruido} = 25$	0.78182	0.60376
$n_{ruido} = 50$	1.31241	0.91742
$n_{ruido} = 100$	2.42014	1.48886
$n_{ruido} = 250$	5.79654	3.22587
$n_{ruido} = 500$	10.89002	6.04457

En este caso, el comportamiento se asemeja mucho al anterior. Las razones por las que esto ocurre son las mismas, en ambas versiones se itera para cada escenario de ruido. En caso de que se hubiera realizado una paralelización mayor, se habrían observado mayores mejorías en este apartado.

Tabla 4.3 Variación de tiempos respecto a L.

	CPU	GPU
$L = 2000$	0.78182	0.60376
$L = 5000$	2.38001	0.70453
$L = 10000$	6.63152	0.94174
$L = 50000$	93.78464	4.54952
$L = 100000$	294.89768	13.10022

Aquí, el comportamiento esperado se hace mucho más acusado que en los casos anteriores. Los tiempos de la CPU crecen de manera exponencial mientras que la GPU mantiene unos órdenes de magnitud aceptables. Por tanto, la implementación en paralelo permite un aumento de partículas mucho mayor que la versión secuencial.

Tabla 4.4 Variación de tiempos respecto a N_p.

	CPU	GPU
$N_p = 4$	0.78182	0.60376
$N_p = 10$	1.50954	0.72003
$N_p = 15$	1.94999	0.74087
$N_p = 20$	3.19954	0.80962
$N_p = 30$	4.91503	0.95644

La GPU mantiene unos tiempos prácticamente constantes para cambios considerables del horizonte de predicción. Para el caso de la CPU, este aumenta levemente.

5 Resultados

En este capítulo se irán exponiendo los resultados de los ensayos, los inconvenientes detectados y las razones que impulsan a modificar el algoritmo inicialmente expuesto hasta el final. Además, se irán modificando diferentes parámetros del algoritmo para estudiar los efectos que estos producen en la solución.

En el ensayo a realizar, la referencia se cambiará un total de 4 veces. Estas referencias son las que se muestran en la tabla siguiente:

Tabla 5.1 Referencias asignadas durante el ensayo.

h_1	h_2	h_3	h_4	q_a	q_b
0.5955	0.6616	0.5384	0.7682	1.7602	1.8072
0.8421	0.9356	0.7614	1.0863	2.0933	2.1491
0.4211	0.4678	0.3807	0.5432	1.4802	1.5197
0.6391	0.4601	0.8473	0.2649	1.0337	2.2671

A continuación se resumen las principales diferencias de los algoritmos a comparar:

- Algoritmo básico. Este es el algoritmo inicial presentado en el capítulo anterior proveniente de la formulación básica del problema SMC y de las publicaciones anteriormente citadas. La función de coste tendrá la siguiente forma:

$$V_{N_p}(x,u) = \sum_{k=1}^{Np} (x(k) - x_{ref}(k))'(x(k) - x_{ref}(k)) + \sum_{k=1}^{Np} \lambda (u(k) - u(k-1))'(u(k) - u(k-1)) \quad (5.1)$$

- Algoritmo con trasplante básico. En este caso, se incluyen las mejores partículas del instante anterior desplazadas en un instante de muestreo.
- Algoritmo adaptativo con trasplante. Al caso anterior se le añade la modificación de la expresión del e_z en función de la distancia a la referencia.
- Algoritmo adaptativo con trasplante incluyendo un término u_{ref}. La función de coste se ve modificada para tener en cuenta un término u_{ref} que sustituye al término de esfuerzo de control. Por tanto, la función de coste queda de la siguiente manera:

$$V_{N_p}(x,u) = \sum_{k=1}^{Np} (x(k) - x_{ref}(k))'(x(k) - x_{ref}(k)) + \sum_{k=1}^{Np} \lambda (u(k) - u_{ref}(k))'(u(k) - u_{ref}(k)) \quad (5.2)$$

- Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}. El último de los algoritmos propuestos aporta la ponderación de los escenarios, un cambio en el cálculo de los desempeños y en la obtención de los escenarios.
- MPC lineal incluyendo un término u_{ref}. Cuya función de coste será la correspondiente a la ecuación 5.2.

A lo largo de las simulaciones se observan principalmente dos problemas que vamos a intentar erradicar conforme avancemos:

- El error de seguimiento. Esto es debido a que estamos usando un modelo lineal para controlar un sistema no lineal, por lo que en el momento en el que nos alejamos del punto de linealización se producen discrepancias en el modelo. Cabe destacar que los escenarios de ruido los hemos comparado con las incertidumbres de los parámetros del modelo lineal con la idea de que al tenerlas en cuenta sea posible representar en todo el espacio el sistema de manera suficientemente buena.
- El rizado de las acciones de control calculadas. Los algoritmos propuestos tienen una gran componente aleatoria. Esto provoca que durante su funcionamiento (en especial durante el régimen permanente) exista cierto ruido en las acciones de control, mientras que lo ideal sería que fueran perfectamente suaves.

5.1 $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$

5.1.1 Algoritmo básico

En esta versión inicial se evidencian los problemas presentados anteriormente. Se puede observar tanto la existencia de cierto offset tanto como un gran rizado en las acciones de control.

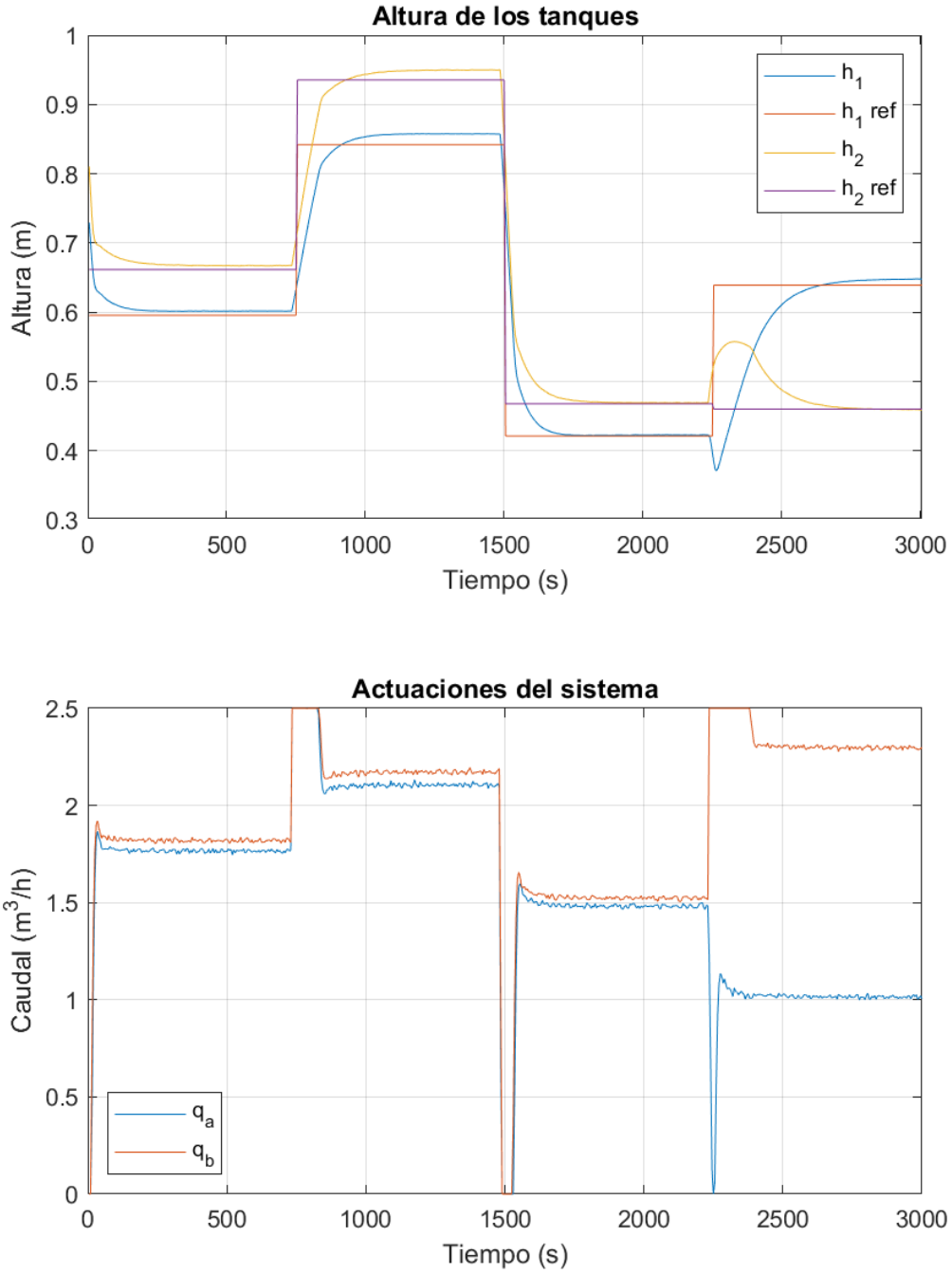

Figura 5.1 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el algoritmo básico.

5.1.2 Algoritmo con trasplante básico

Añadir el caso más básico de trasplante no tiene prácticamente efecto en el offset que aparece en régimen permanente. Por otro lado, las acciones de control parecen un poco más suaves, llegando en algunos de los puntos de funcionamiento a tener un comportamiento casi estacionario, aunque en la mayoría de las ocasiones siguen pareciendo bastante caóticas.

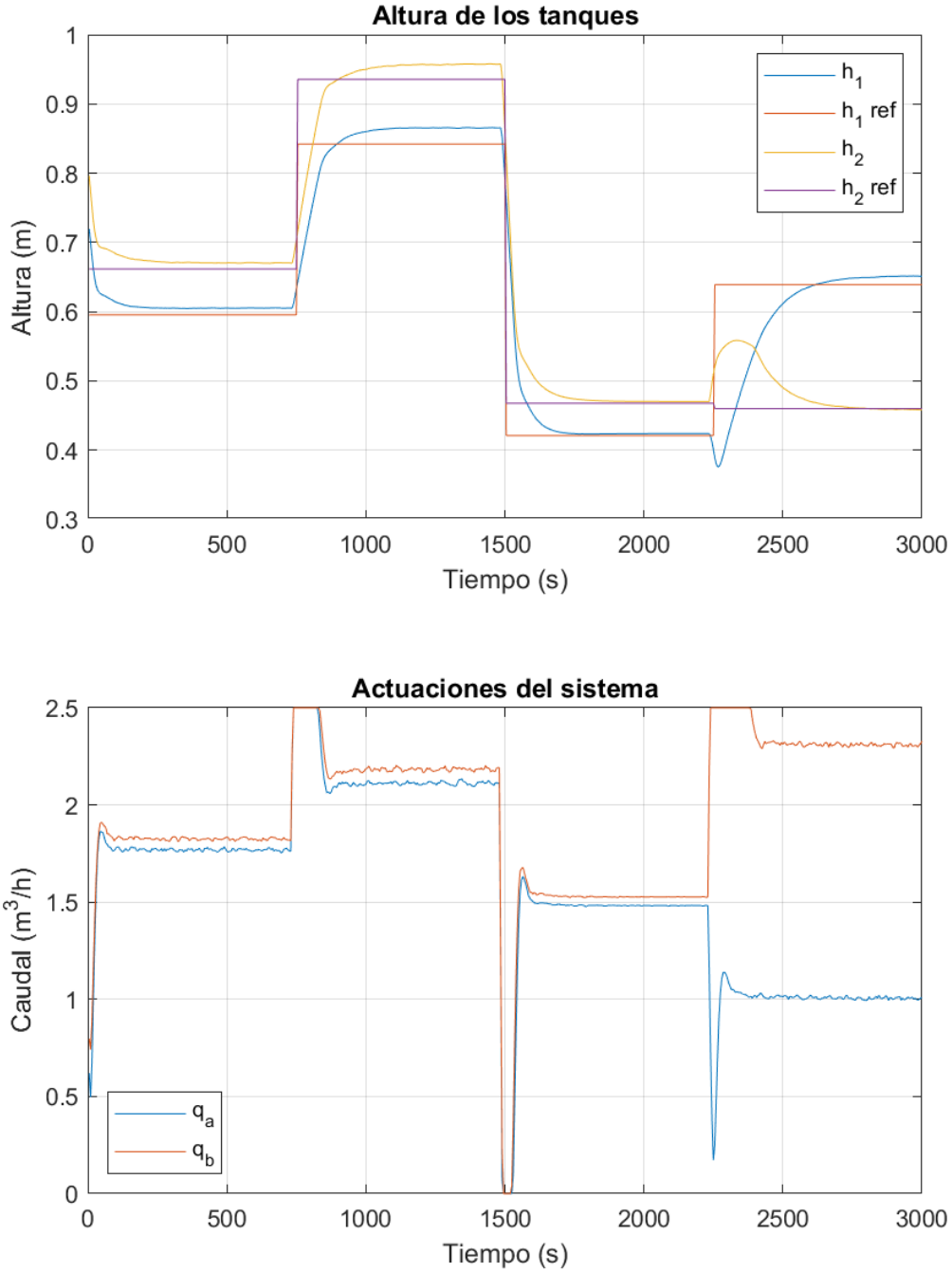

Figura 5.2 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante básico.

5.1.3 Algoritmo adaptativo con trasplante

De igual manera que ocurría en el caso anterior, el offset permanece prácticamente inalterable. Por otra parte, gracias a la adaptación de la amplitud de la perturbación en función de la distancia del estado actual a la referencia, se consigue suavizar todavía más las acciones de control.

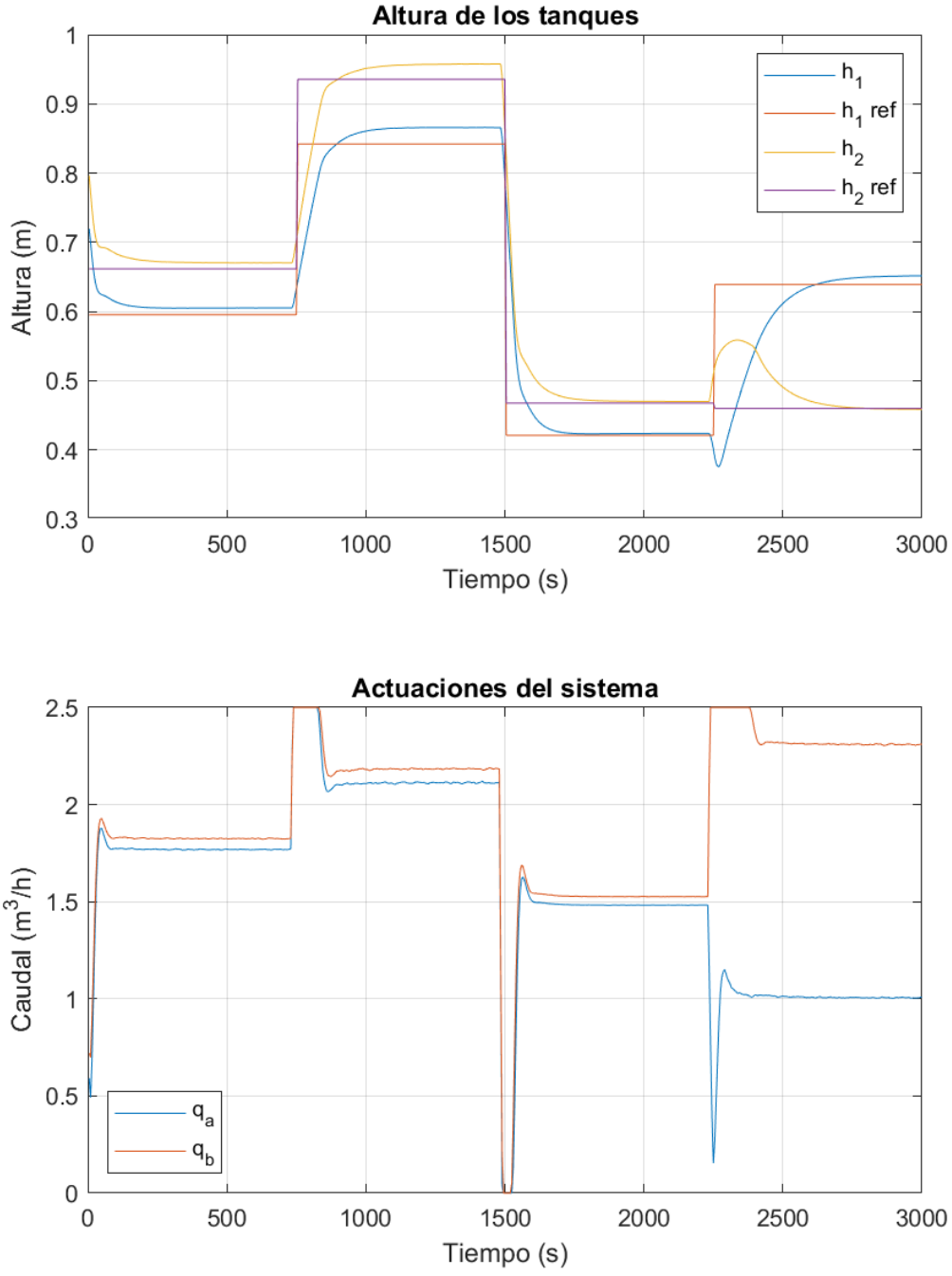

Figura 5.3 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo.

5.1.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

Para este caso, la convergencia es un poco más lenta debido a que el control se vuelve un poco más conservador. Por otro lado, el rizado en régimen permanente es prácticamente inexistente Puede observarse a su vez como las acciones de control no se estabilizan en su referencia asignada. Esto es principalmente debido a las discrepancias del modelo lineal respecto al sistema real.

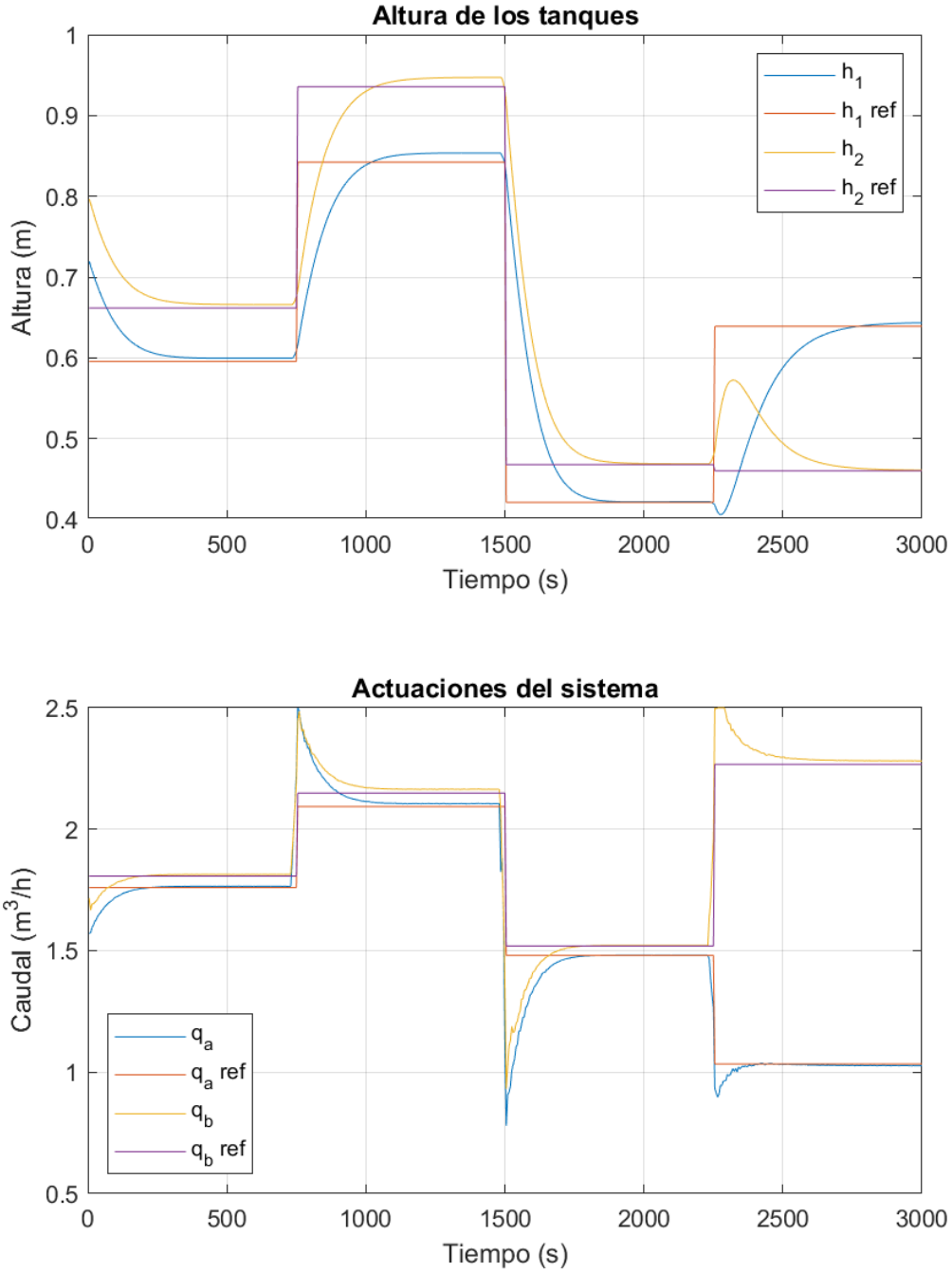

Figura 5.4 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.1.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

Los resultados resultan prometedores ya que sin necesidad de tener valores elevados en los parámetros (que llevarían a una mayor cantidad de iteraciones, partículas, etc. y por ende, mayor tiempo de computación) es capaz de controlar con un offset prácticamente nulo. En las acciones de control puede verse como estas sí convergen a los valores de referencia sin ningún rizado,

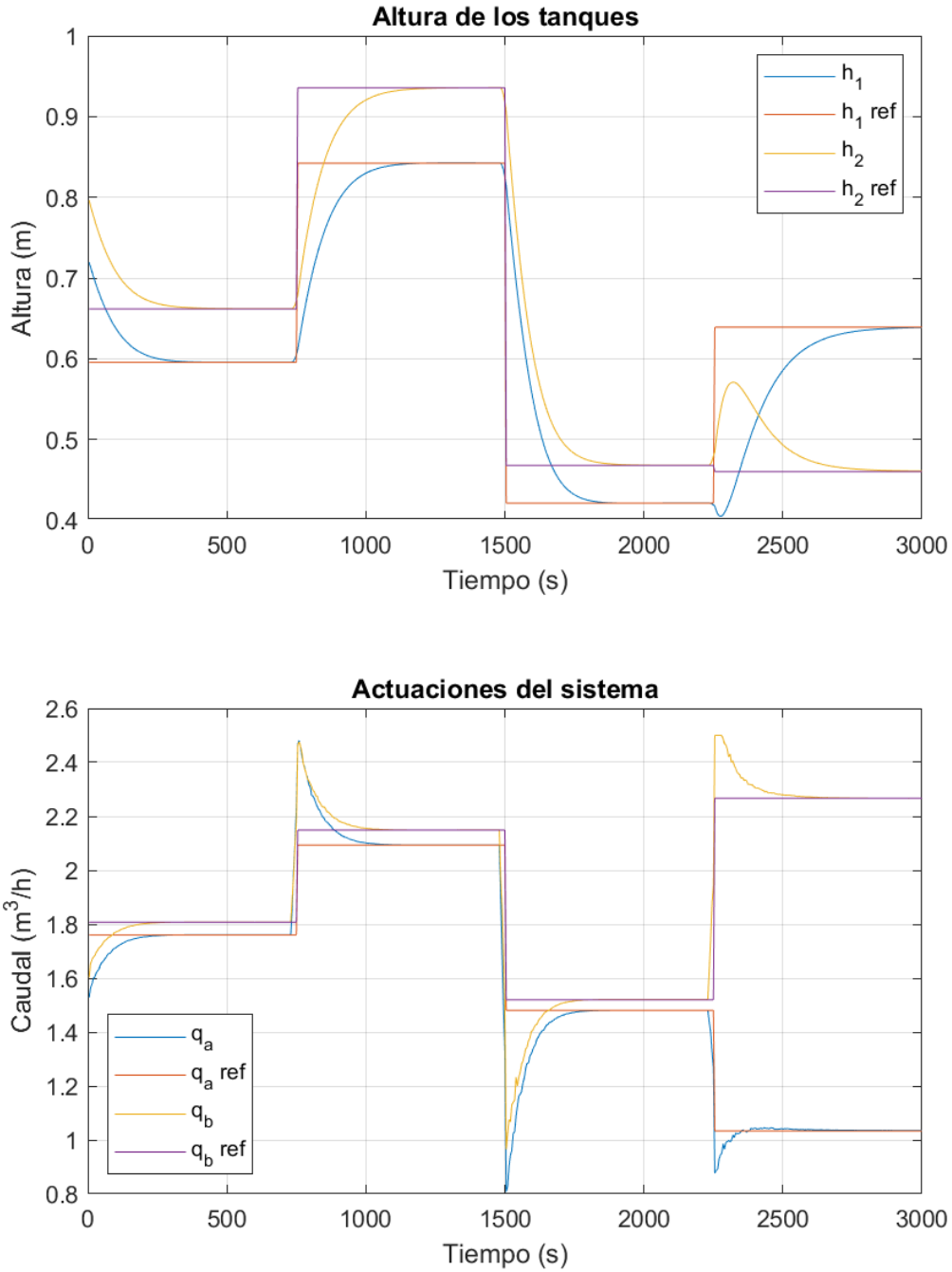

Figura 5.5 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

Por tanto, parece que los primeros algoritmos no son capaces de cumplir su cometido en este ejemplo, puesto que su comportamiento resulta muy parecido al del MPC siguiente. Sin embargo, la ponderación de los escenarios presenta un buen panorama de inicio.

5.1.6 MPC lineal

El funcionamiento es muy parecido al de los primeros algoritmos planteados. Sin embargo, es mejor que estos debido a que tiene unas acciones de control más suaves y tiene un tiempo de computación menor.

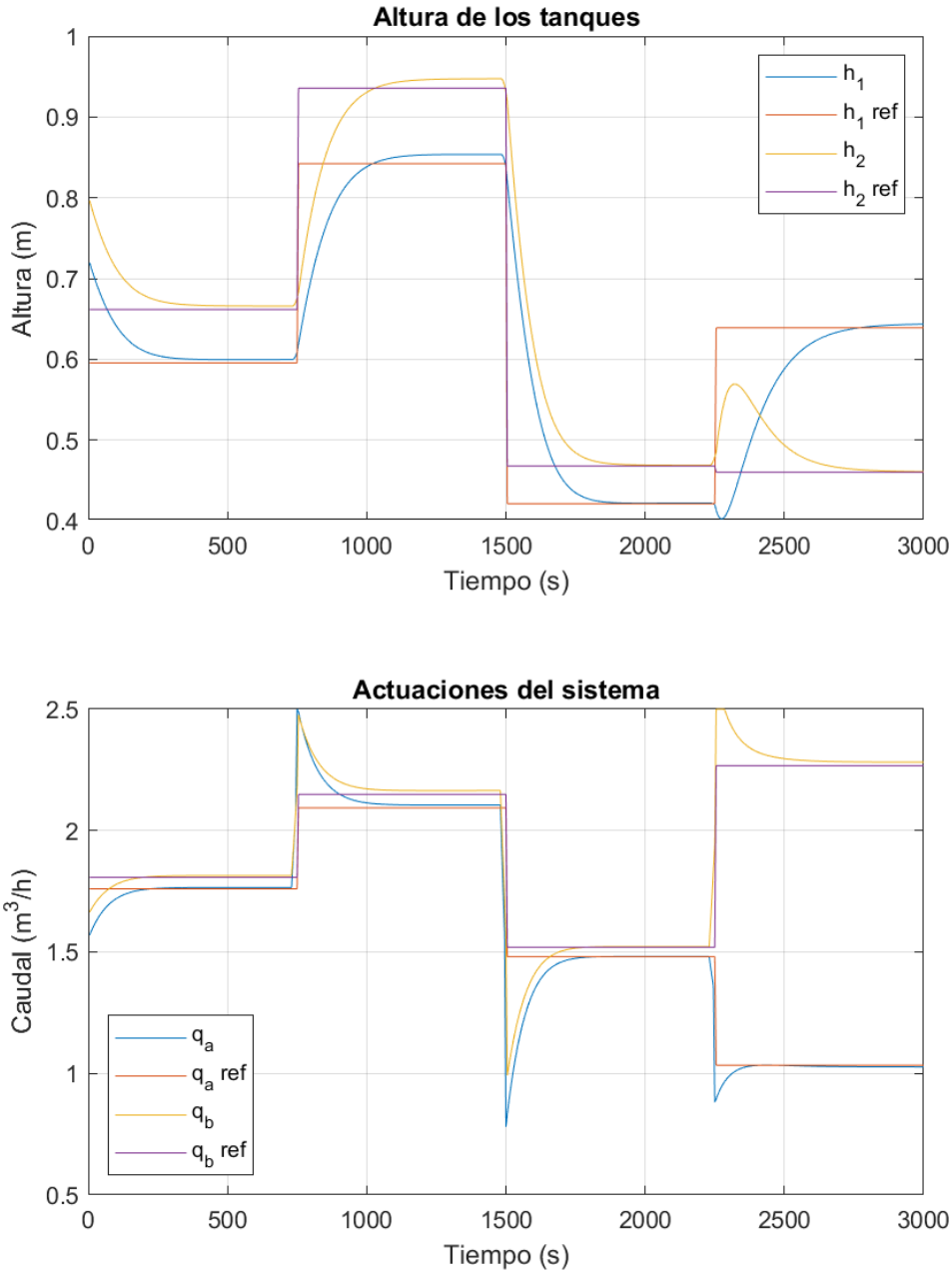

Figura 5.6 Resultado del ensayo para MPC lineal incluyendo un término u_{ref}.

5.2 $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$

A continuación, se amplía el horizonte de predicción para comprobar cómo se comporta el algoritmo.

5.2.1 Algoritmo básico

Aumentar el horizonte de predicción supone aumentar la compplejidad del problema. Si no se aumentan el número de partículas u otros parámetros, se obtienen señales muy ruidosas para el algoritmo básico.

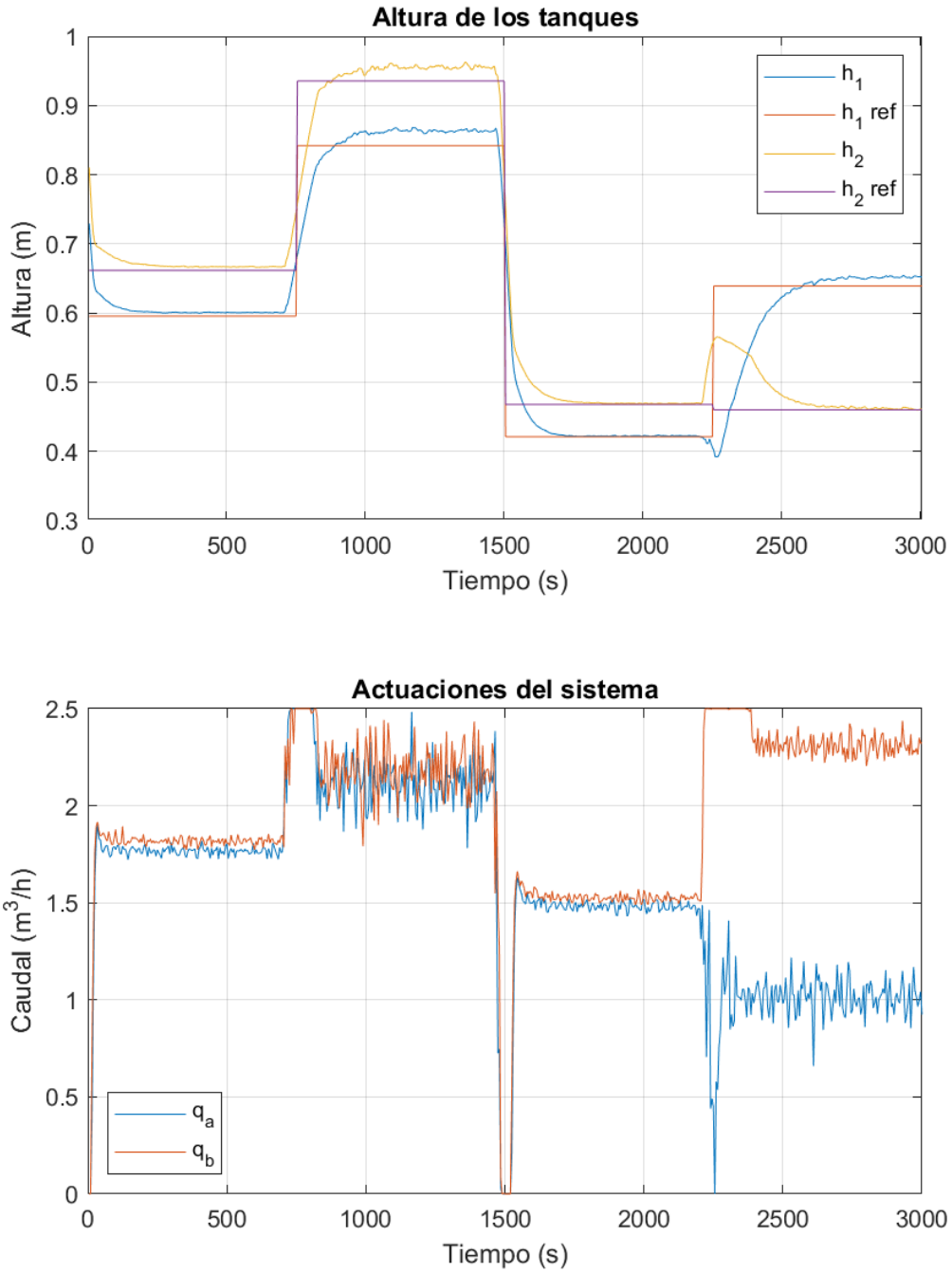

Figura 5.7 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$ con el algoritmo básico.

5.2.2 Algoritmo con trasplante básico

En este caso, el trasplante (aún en su faceta más básica) ayuda bastante a suavizar las acciones de control con respecto al algoritmo inicial. Sin embargo, sigue existiendo un rizado considerable.

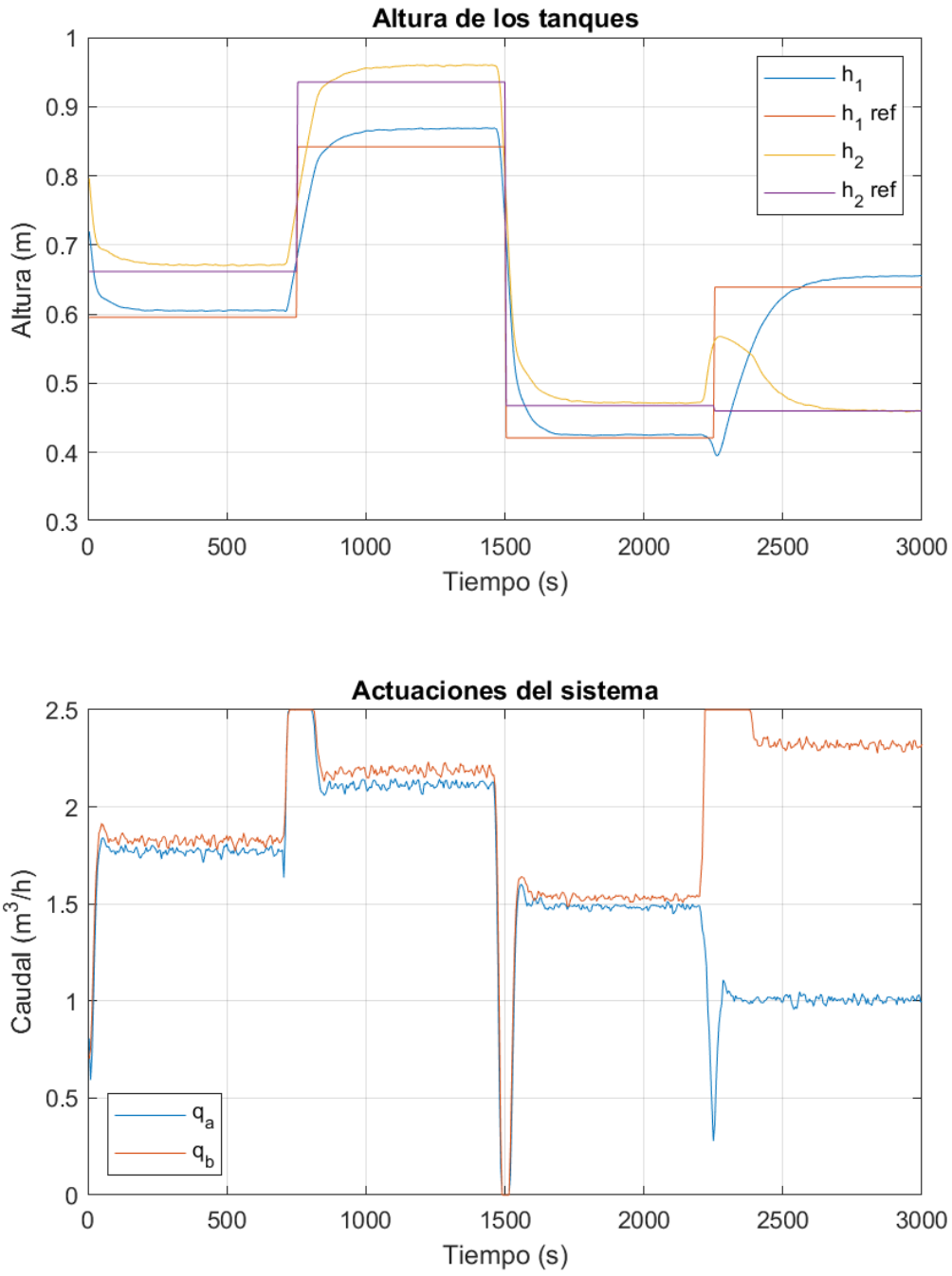

Figura 5.8 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$ con el trasplante básico.

5.2.3 Algoritmo adaptativo con trasplante

Añadir la adaptación al trasplante produce un efecto muy beneficioso en las acciones de control, donde el rizado prácticamente ha desaparecido.

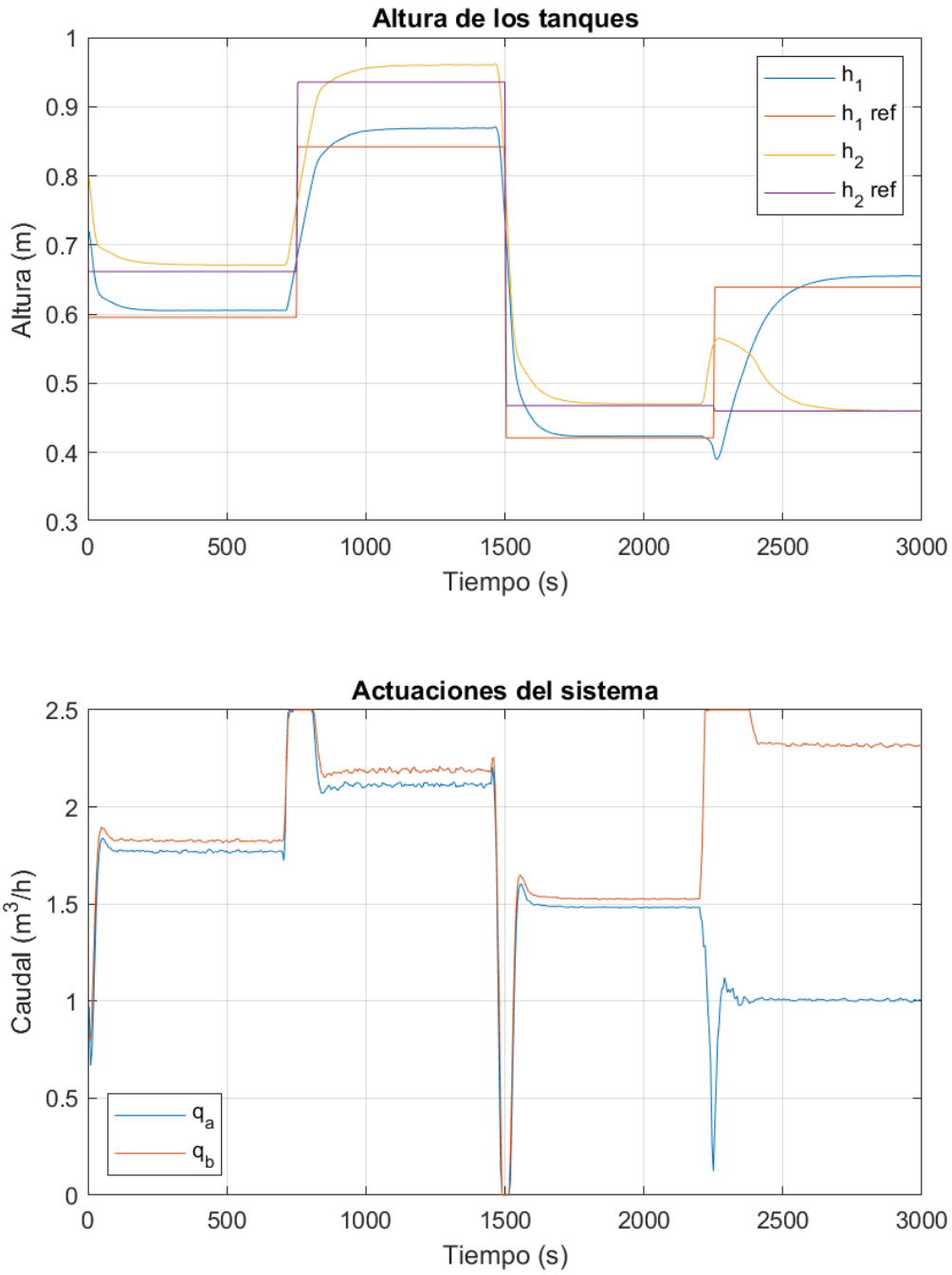

Figura 5.9 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$ con el trasplante adaptativo.

5.2.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

Al igual que para el caso anterior, añadir un término de referencia para las entradas reduce un poco el ruido que estas presentan aunque no lo elimina por completo.

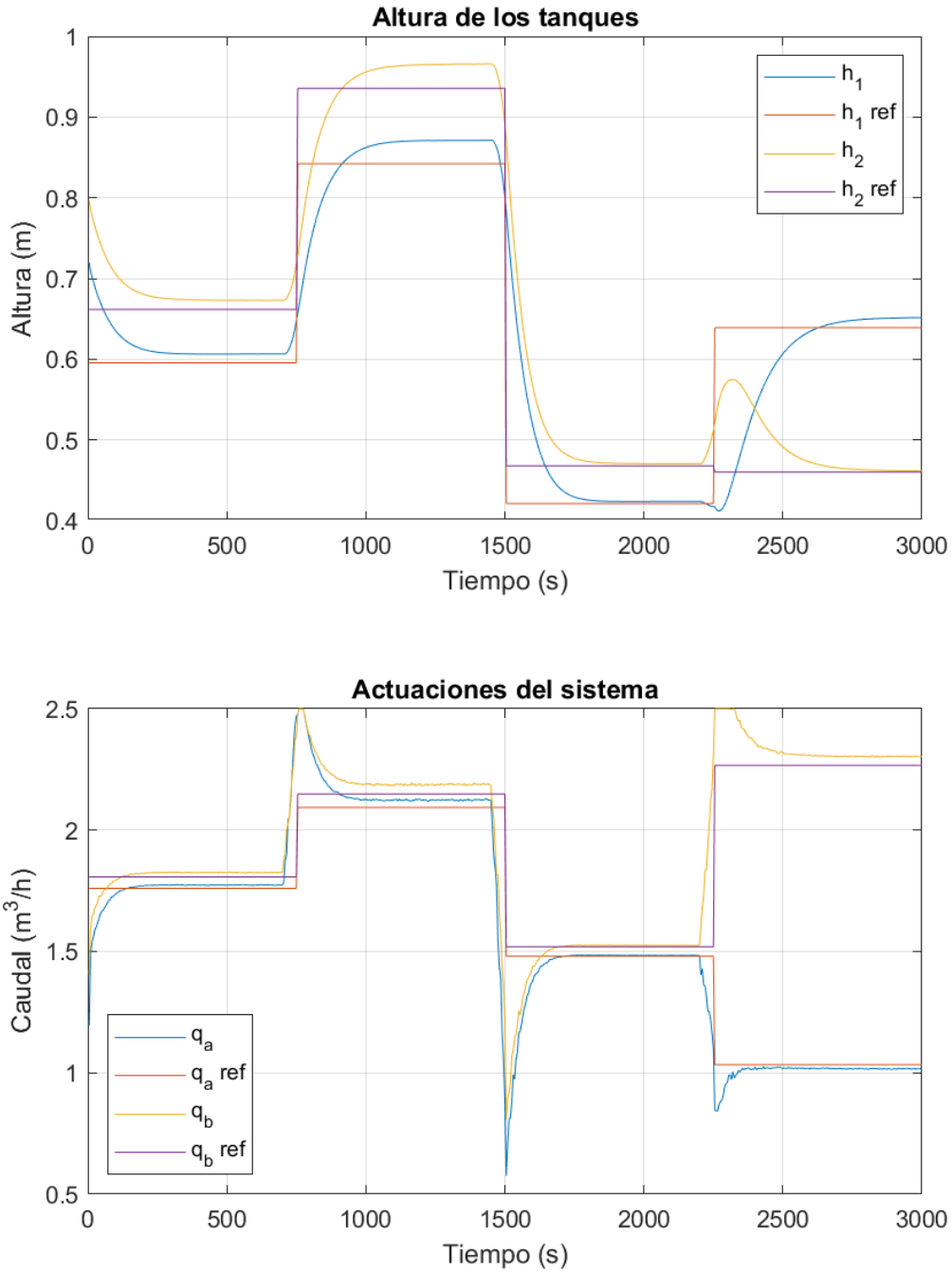

Figura 5.10 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.2.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

El algoritmo que incluye todas las mejoras sigue siendo el que destaca en su comportamiento. Puede verse como a pesar de haber aumentado en más del doble el horizonte de predicción, este es capaz de obtener soluciones suficientemente buenas para alcanzar la referencia sin ningún offset y con un rizado mínimo en las acciones de control.

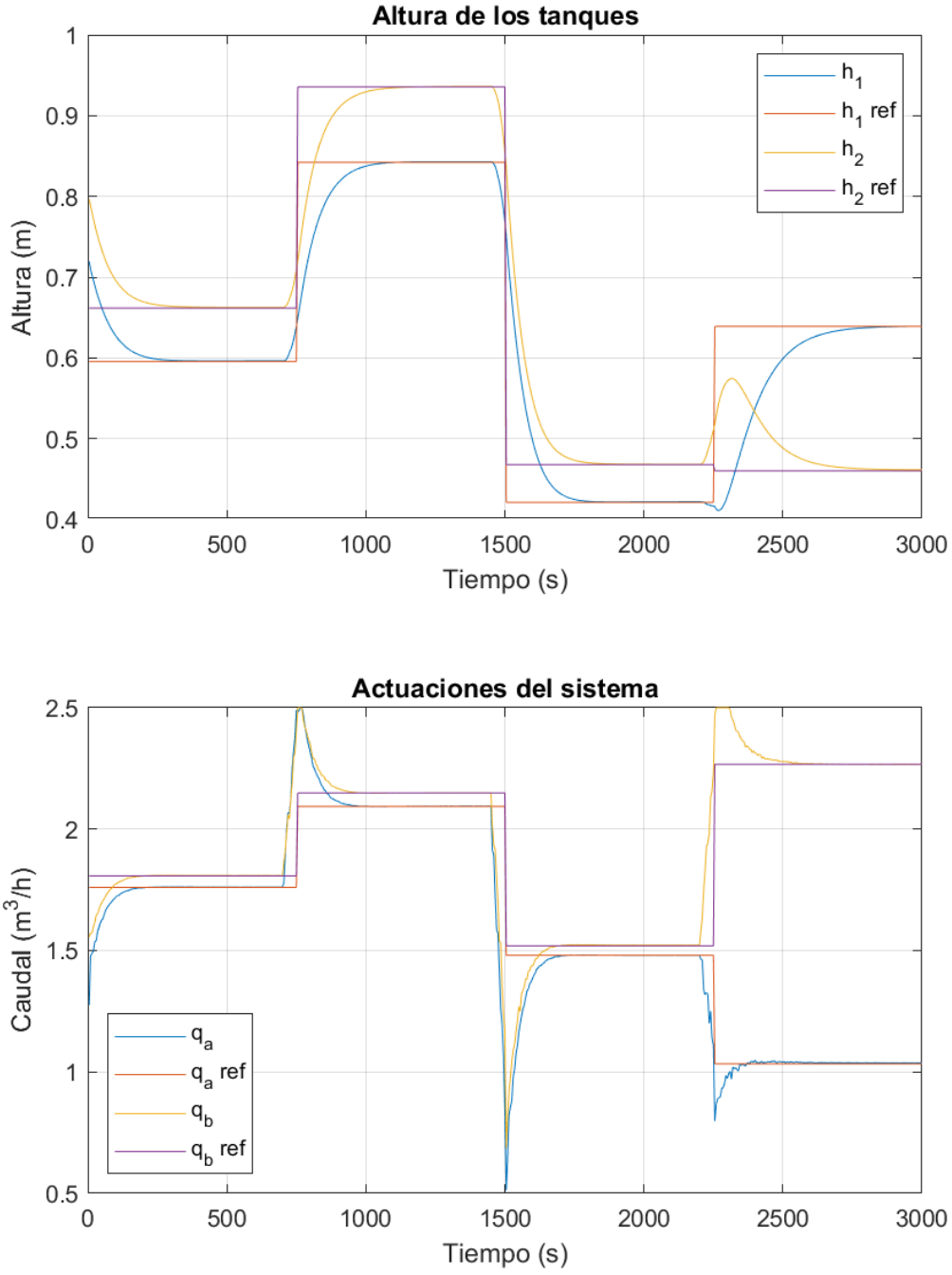

Figura 5.11 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 2000$, $N_p = 10$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

5.3 $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$

Puesto que a lo largo de los resultados siguientes los cambios en la evolución de las alturas no son apreciables, se omiten dichas gráficas, exponiéndose únicamente las correspondientes a las acciones de control.

5.3.1 Algoritmo básico

Comprobamos que tener gran cantidad de partículas no siempre es estrictamente necesario. Es fácil ver que no mejora enormemente a suavizar las acciones de control, por lo que concluímos que la cantidad de partículas necesarias está más relacionada con el tamaño del horizonte de predicción.

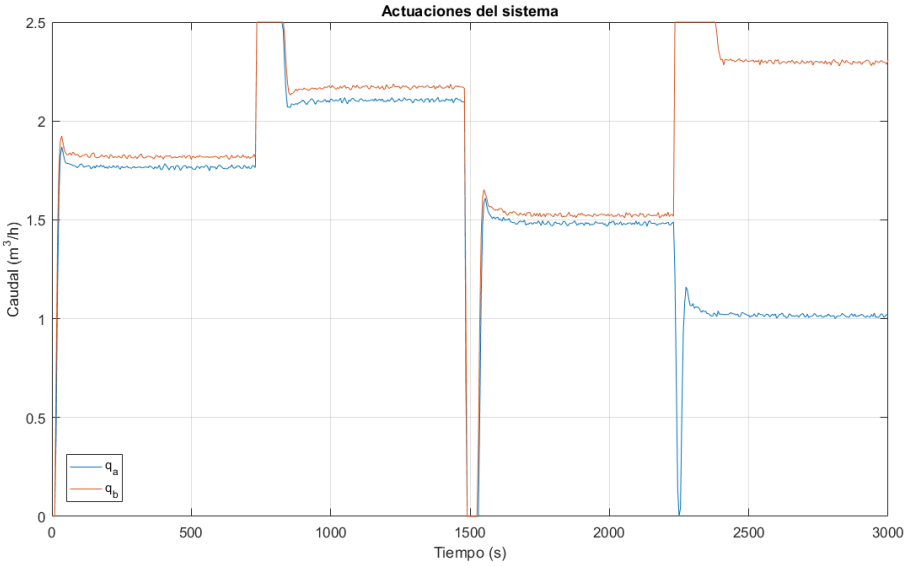

Figura 5.12 Acciones de control para $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$ con el algoritmo básico.

5.3.2 Algoritmo con trasplante básico

Lo mismo ocurre en este caso, donde el aumento del número de partículas en cinco veces no ha provocado una mejora sustancial.

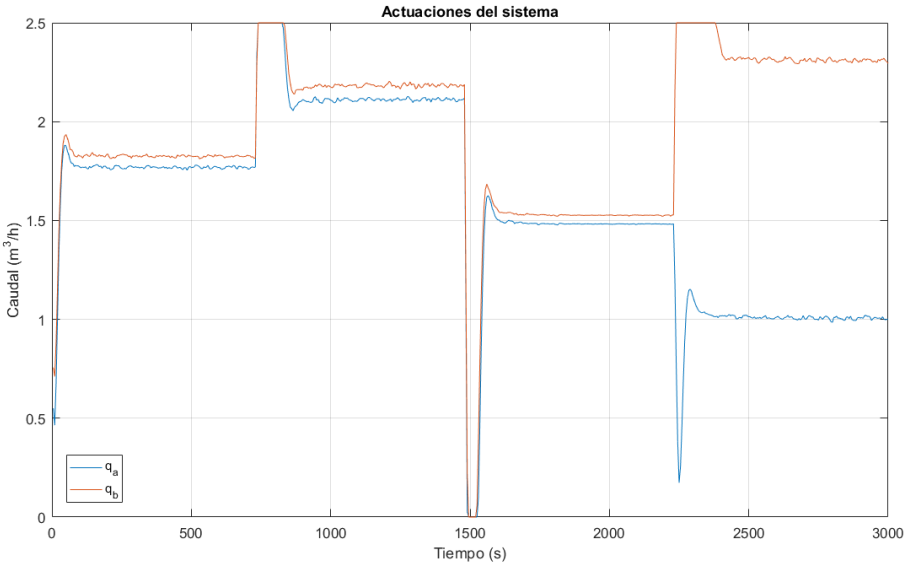

Figura 5.13 Acciones de control para $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$ con el trasplante básico.

5.3.3 Algoritmo adaptativo con trasplante

De igual manera que en los casos anteriores, no se aprecian cambios importantes en el rizado de las acciones de control.

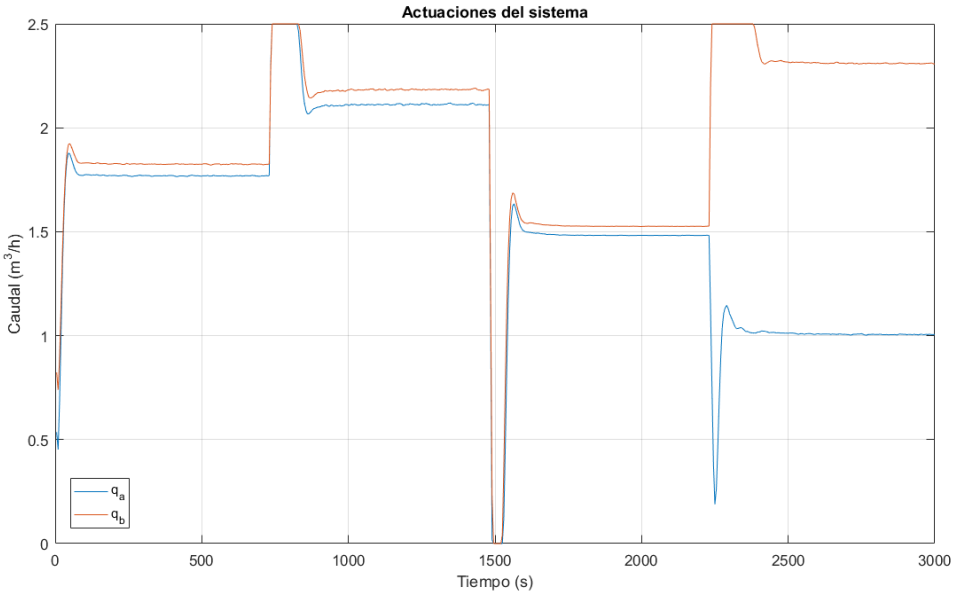

Figura 5.14 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$ con el trasplante adaptativo.

5.3.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

El comportamiento sigue siendo similar al expuesto anteriormente.

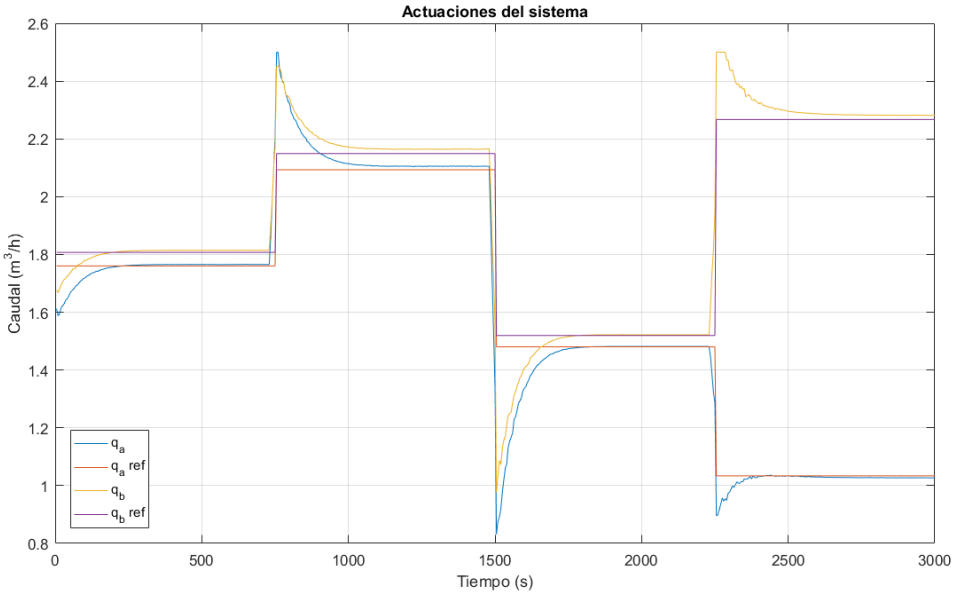

Figura 5.15 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.3.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

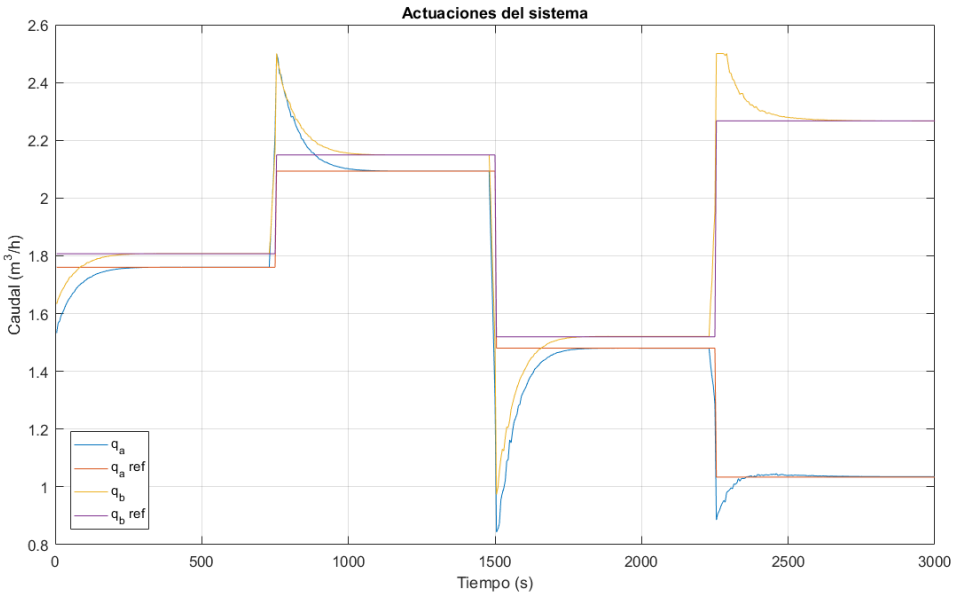

Figura 5.16 Resultado del ensayo para $z = 25$, $n_{ruido} = 25$, $L = 10000$, $N_p = 4$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

5.4 $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$

Se propone ahora volver a los parámetros iniciales cambiando únicamente el número de escenarios por iteración de z, los cuales se elevarán hasta 50. Puede apreciarse en las figuras siguientes que no se produce ningún cambio apreciable en las acciones de control para ninguno de los algoritmos presentados.

5.4.1 Algoritmo básico

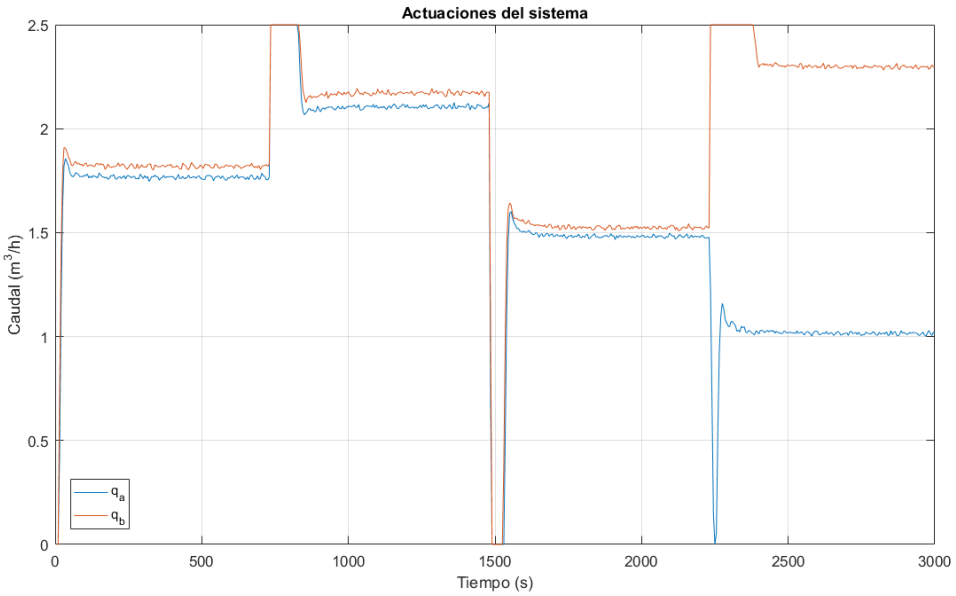

Figura 5.17 Acciones de control para $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$ con el algoritmo básico.

5.4.2 Algoritmo con trasplante básico

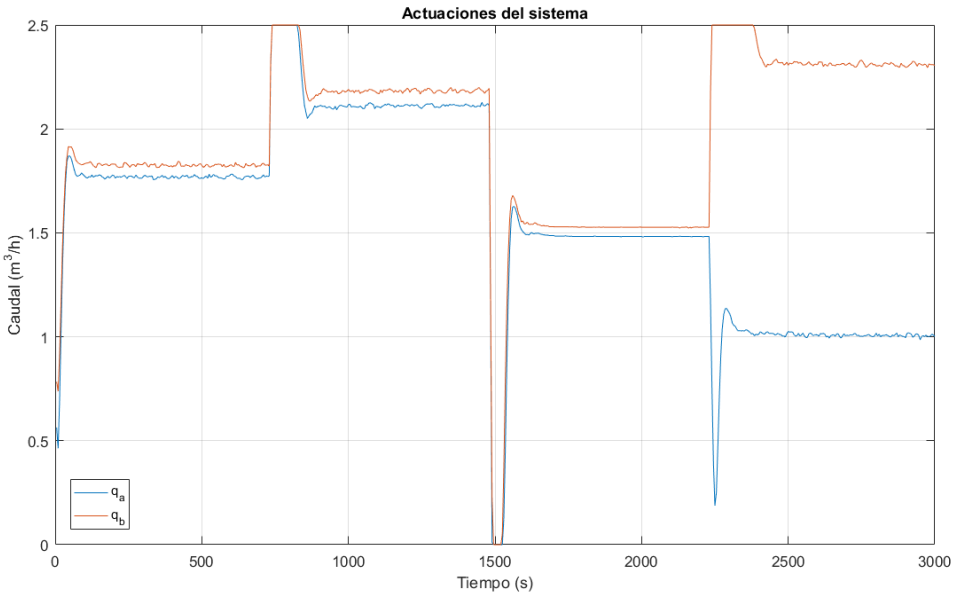

Figura 5.18 Acciones de control para $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$ con el trasplante básico.

5.4.3 Algoritmo adaptativo con trasplante

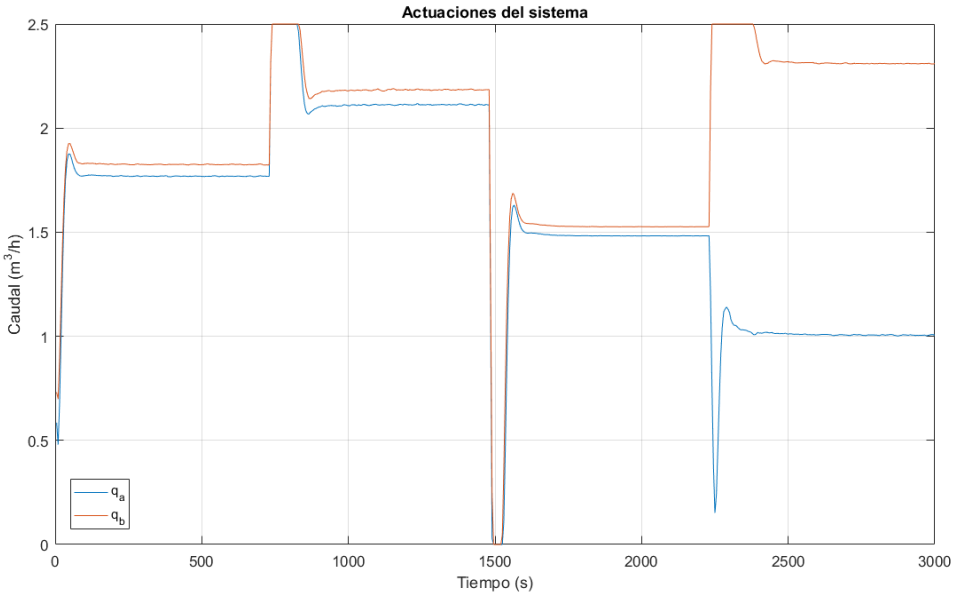

Figura 5.19 Resultado del ensayo para $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo.

5.4.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

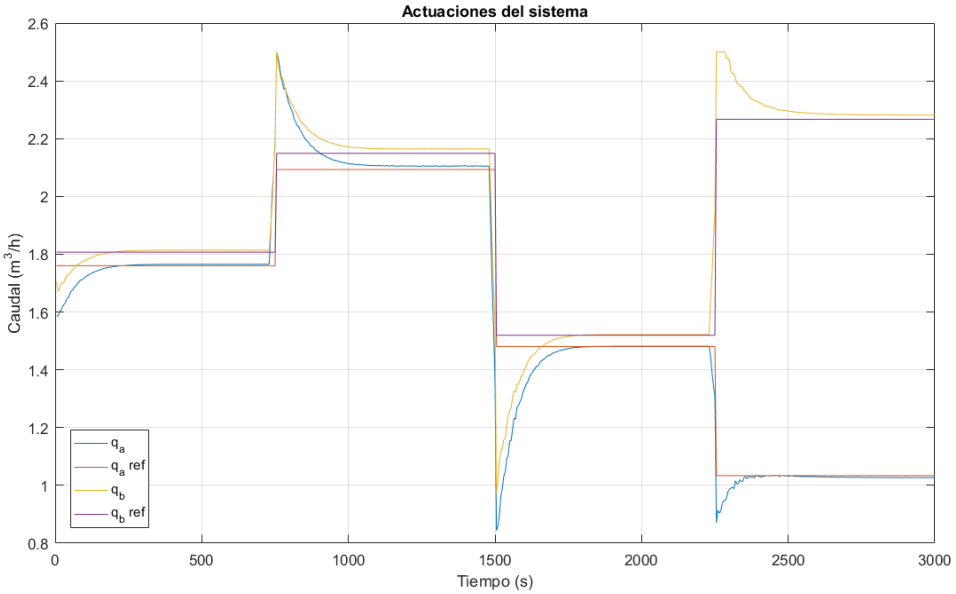

Figura 5.20 Resultado del ensayo para $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.4.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

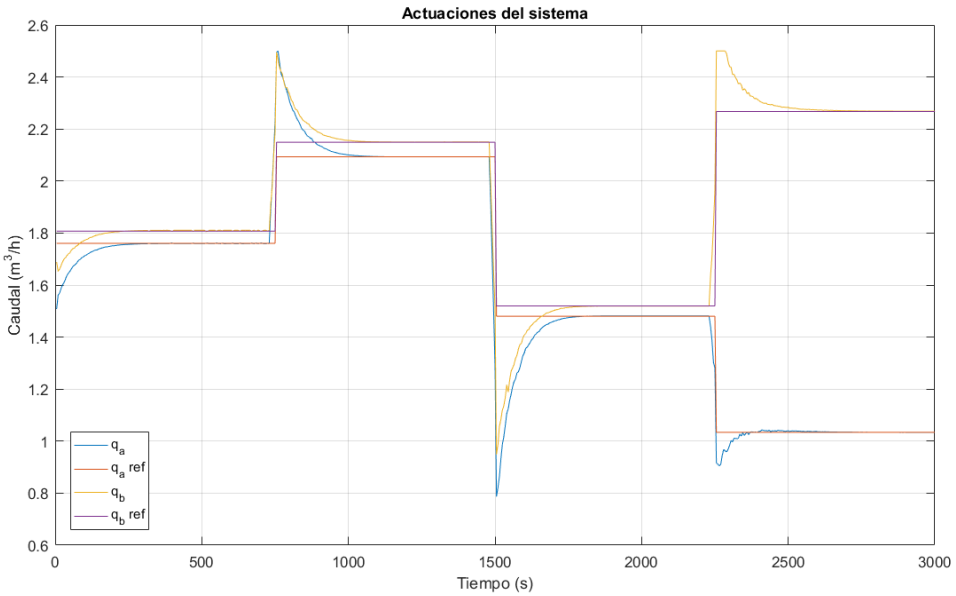

Figura 5.21 Resultado del ensayo para $z = 25$, $n_{ruido} = 50$, $L = 2000$, $N_p = 4$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

5.5 $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$

En este caso se propone el aumento de las iteraciones y del consiguiente remuestreo asociado, por lo que se espera alcanzar mayor precisión en las soluciones y por tanto, un menor rizado. Puede verse en las figuras siguientes que este se ve un poco reducido para todos los algoritmos propuestos.

5.5.1 Algoritmo básico

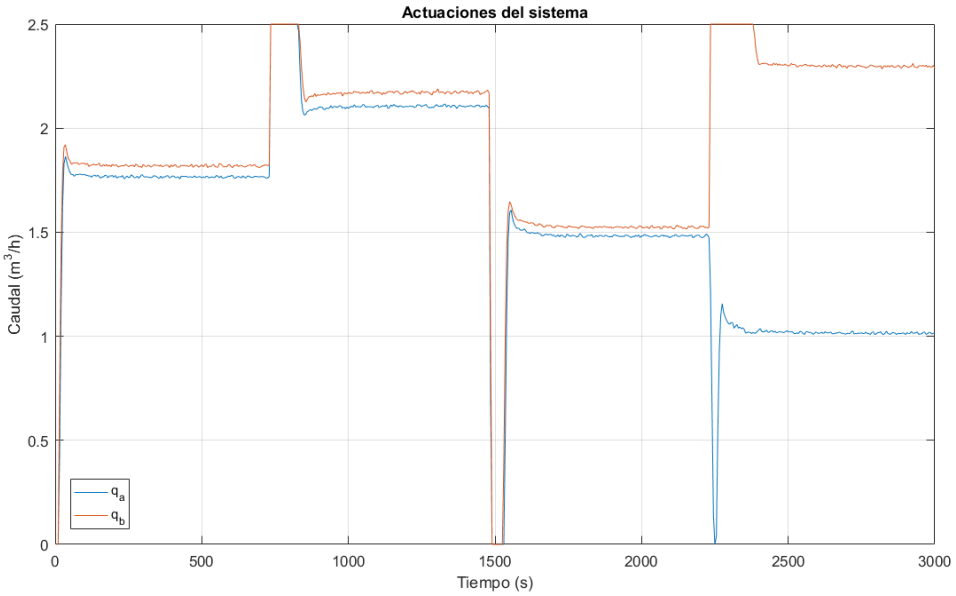

Figura 5.22 Acciones de control para $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el algoritmo básico.

5.5.2 Algoritmo con trasplante básico

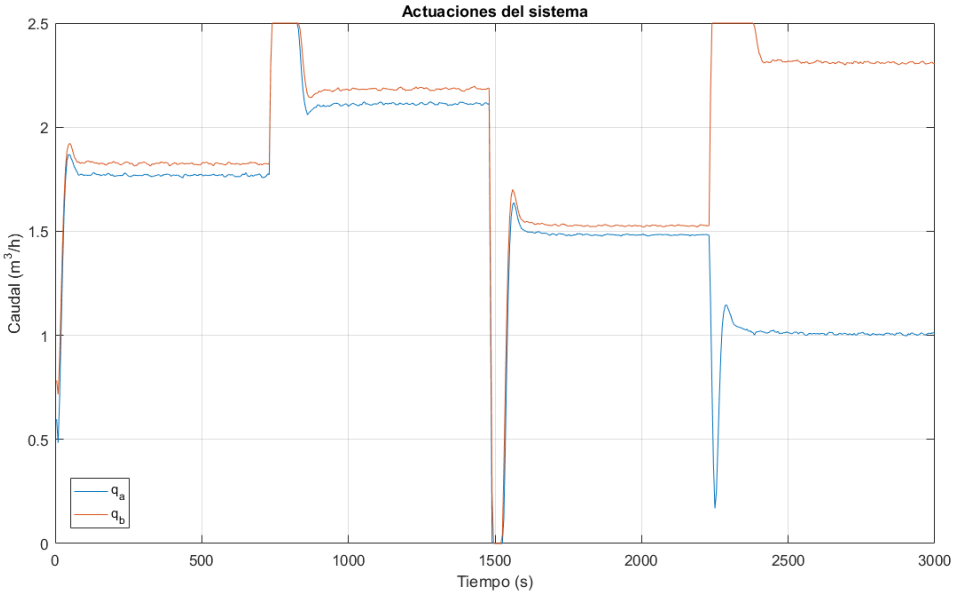

Figura 5.23 Acciones de control para $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante básico.

5.5.3 Algoritmo adaptativo con trasplante

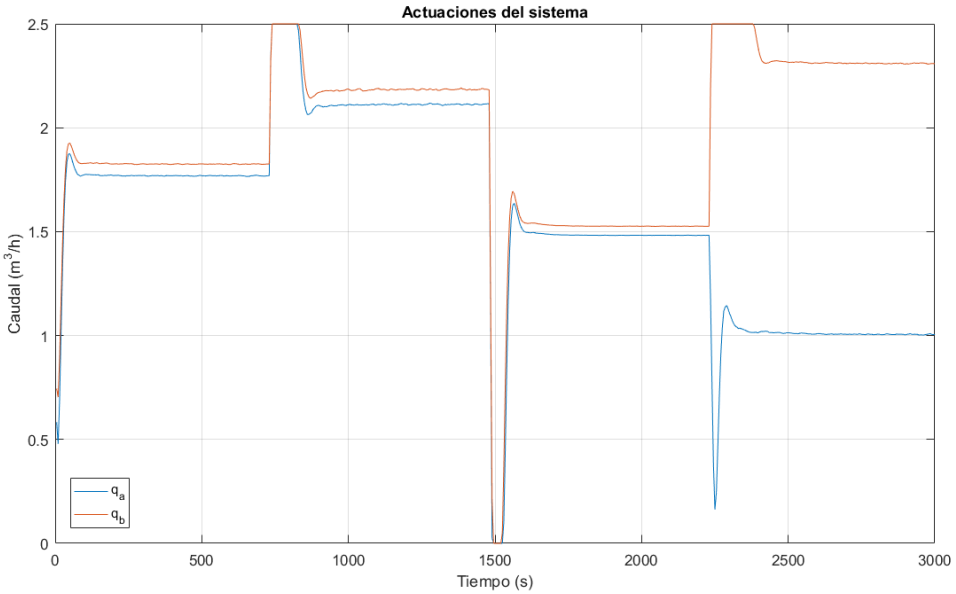

Figura 5.24 Resultado del ensayo para $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo.

5.5.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

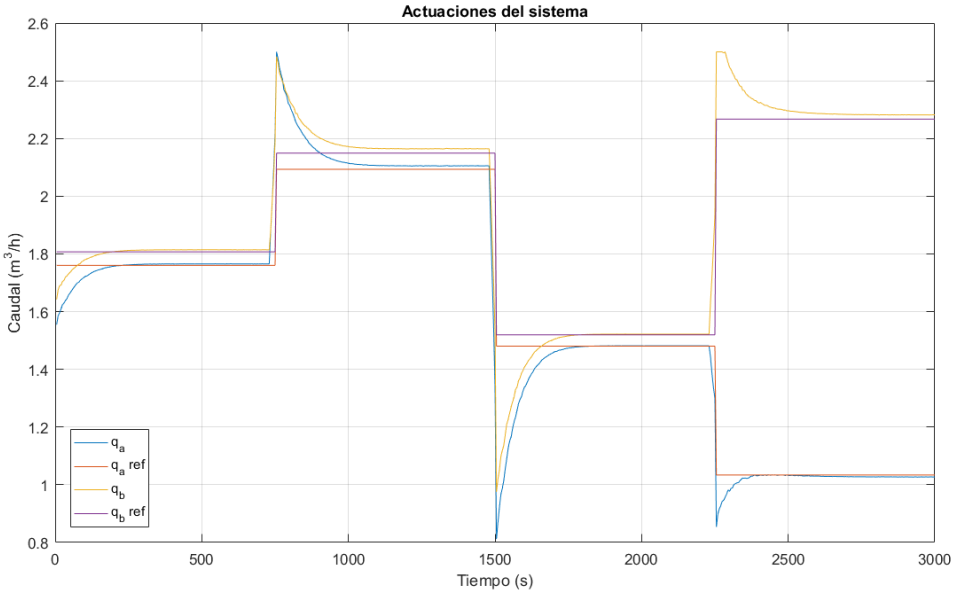

Figura 5.25 Resultado del ensayo para $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.5.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

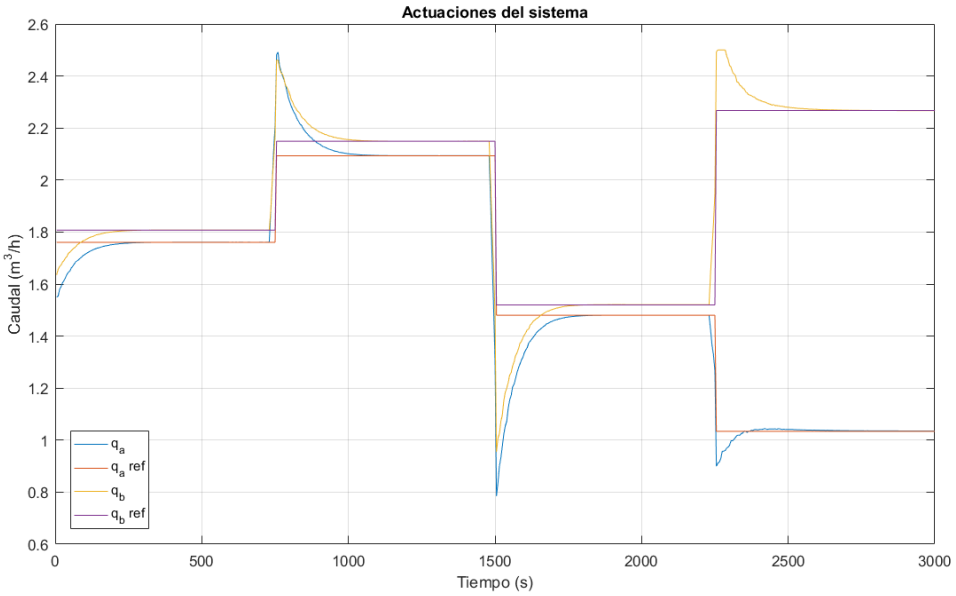

Figura 5.26 Resultado del ensayo para $z = 50$, $n_{ruido} = 25$, $L = 2000$, $N_p = 4$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

5.6 $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$

Aumentar todos los parámetros a excepción del horizonte de predicción debería proporcionar unas soluciones más suaves en teoría. Esto se demuestra en la práctica en las figuras siguientes.

5.6.1 Algoritmo básico

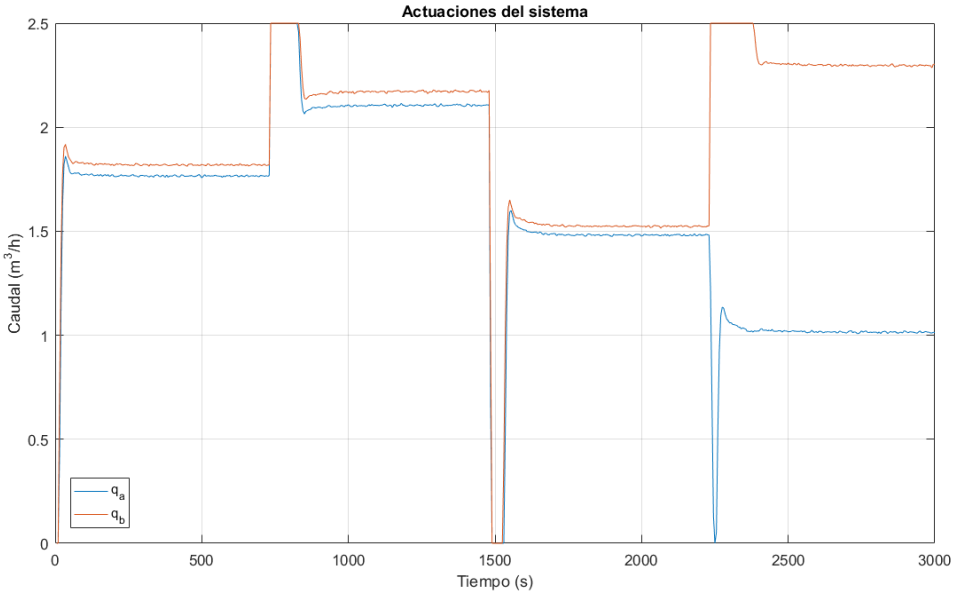

Figura 5.27 Acciones de control para $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$ con el algoritmo básico.

5.6.2 Algoritmo con trasplante básico

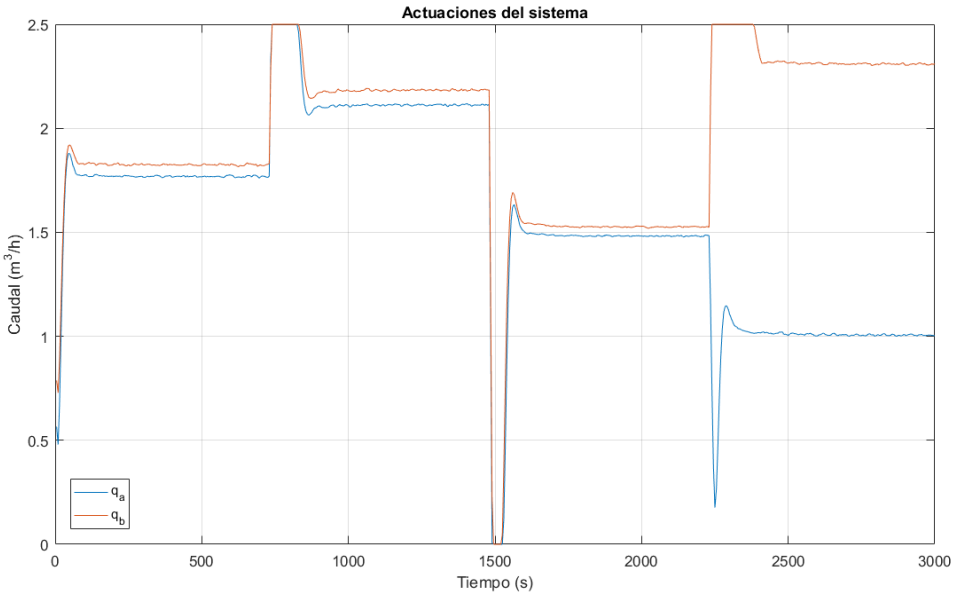

Figura 5.28 Acciones de control para $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$ con el trasplante básico.

5.6.3 Algoritmo adaptativo con trasplante

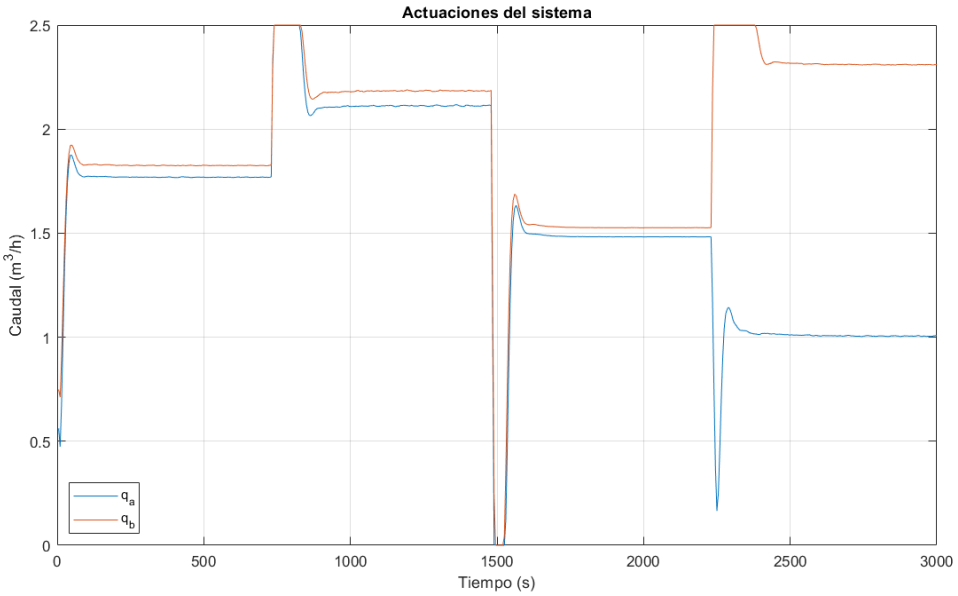

Figura 5.29 Resultado del ensayo para $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$ con el trasplante adaptativo.

5.6.4 Algoritmo adaptativo con trasplante incluyendo un término u_{ref}

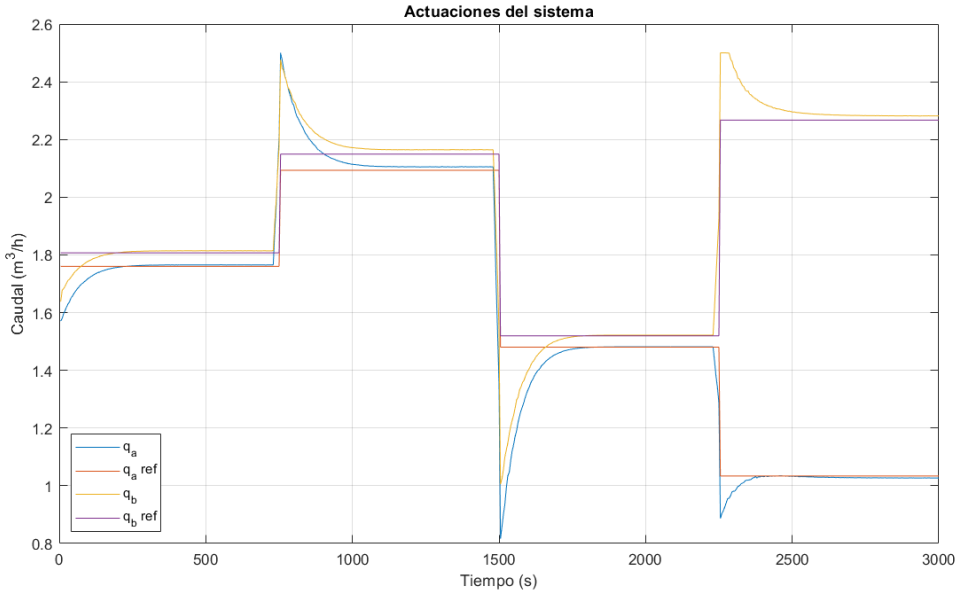

Figura 5.30 Resultado del ensayo para $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$ con el trasplante adaptativo incluyendo un término u_{ref}.

5.6.5 Algoritmo adaptativo con trasplante y ponderación de escenarios incluyendo un término u_{ref}

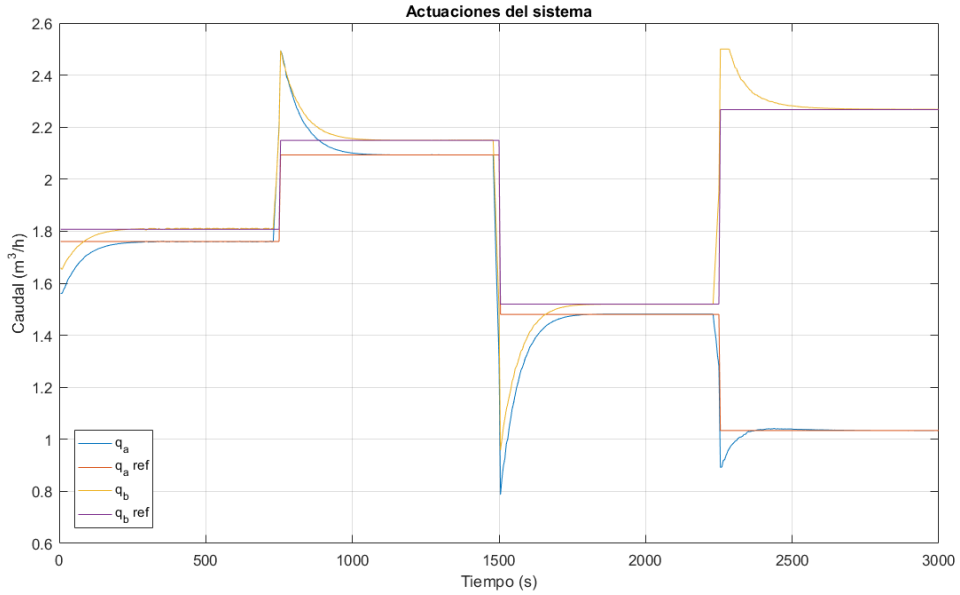

Figura 5.31 Resultado del ensayo para $z = 50$, $n_{ruido} = 50$, $L = 10000$, $N_p = 4$ con la ponderación de escenarios, trasplante adaptativo e incluyendo un término u_{ref}.

5.7 Sistema con ganancia variable

El único algoritmo de los anteriores que funciona notablemente bien para la planta de los cuatro tanques es el que tiene en cuenta la ponderación de los escenarios. Sin embargo, existen gran cantidad de diferencias entre éste último y la proposición inicial (como el propio concepto de escenario, por ejemplo), por lo que

resulta interesante estudiar el comportamiento de los algoritmos de partida en un sistema que presenta unas perturbaciones diferentes a las existentes en el caso anterior.

Para ello, se plantea el control de un sistema lineal dado por la siguiente expresión:

$$y_{k+1} = 0.9y_k + Ku_k$$

Donde K tiene un valor desconocido perteneciente al rango $\in [0.5\,, 1.5]$. De esta manera, para cada ensayo realizado, K tomará un valor dentro de ese rango según una distribución uniforme.

El MPC estará sintonizado para un valor de $K = 1$, el cual podría ser reconocido como la esperanza matemática de K. Por otro lado, el SMC tendrá en cuenta en sus "escenarios de ruido" diferentes valores posibles de K, extrayéndolos de la misma distribución uniforme.

Los ensayos consisten en llevar el sistema desde un determinado punto inicial aleatorio hasta una referencia también aleatoria (aunque dentro de unos límites). Por esta razón, los resultados presentados reflejan el error de seguimiento del sistema y no los valores de las salidas directamente (puesto que entonces no serían comparables). Dichos resultados se ven reflejados en la figura 5.32.

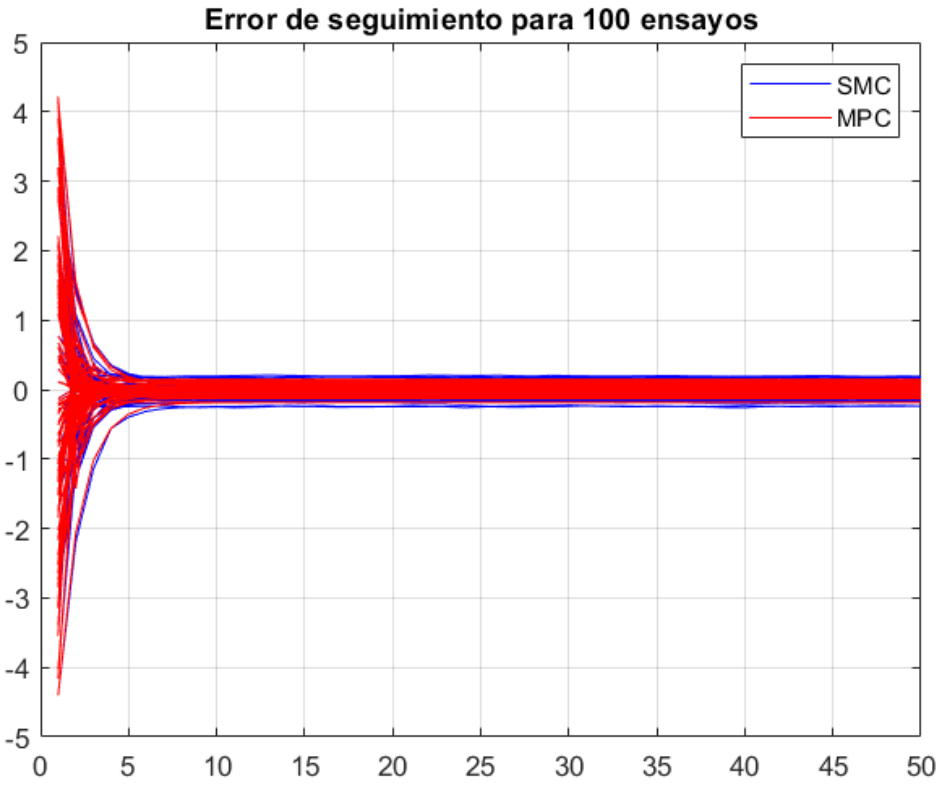

Figura 5.32 Resultado de los ensayos para el sistema con K variable.

Puede verse como en régimen permanente el SMC funciona claramente un poco peor que el MPC nominal. Es más, si observásemos las gráficas que se obtendrían de cada uno de los ensayos, podría verse como el SMC funciona como una versión aproximada y ruidosa del MPC. Es decir, el SMC está tendiendo a igualar la solución proporcionada por el MPC.

5.8 Sistema con perturbaciones aditivas

Se plantea el control de un determinado sistema definido por las matrices siguientes [10]:

$$A = \begin{bmatrix} 0.98676 & 0 \\ 0 & 0.98676 \end{bmatrix} \quad B = \begin{bmatrix} 0.011629 & -0.011444 \\ 0.014331 & -0.014517 \end{bmatrix} \quad E = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Con las siguientes restricciones:

$$A_u = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad b_u = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$A_u u = b_u$$

Quedando definido de la manera:

$$x(k+1) = Ax(k) + Bu(k) + Ew(k)$$

Siendo $w(k)$ una variable normal de media nula y desviación típica 0.025 para el experimento a realizar. Para una batería de 50 ensayos, la evolución del estado se presenta en la figura 5.33.

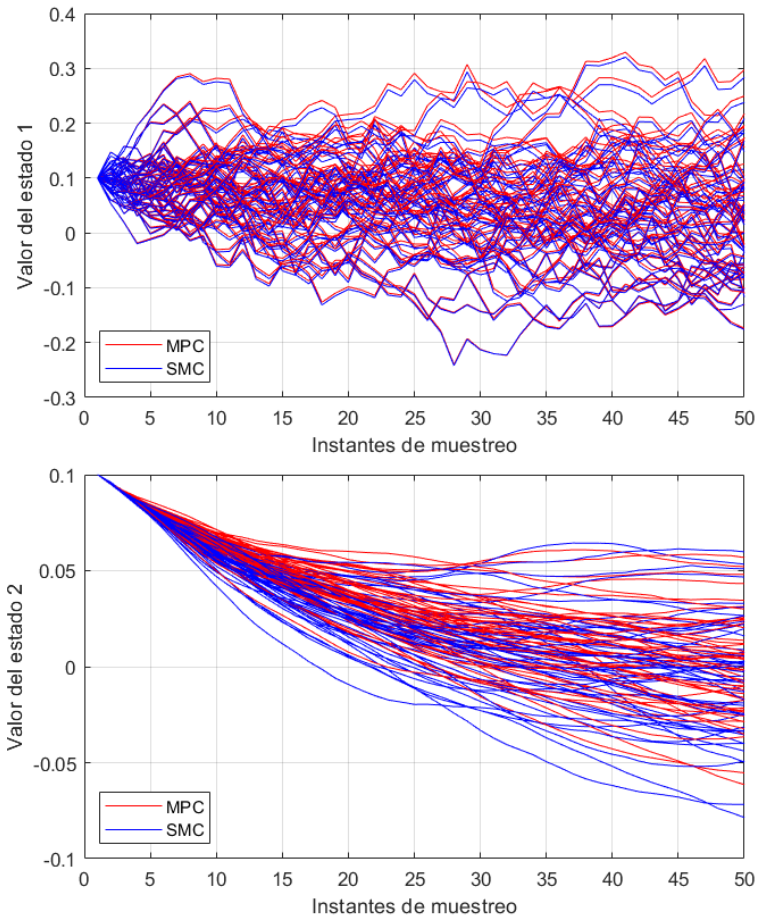

Figura 5.33 Evolución de los estados para 50 ensayos de un sistema con perturbaciones aditivas.

Sin embargo, en la figura anterior es difícil comparar ambas metodologías. Para ello, se presentan las medias instantáneas de los estados y de las entradas en las figuras 5.34 y 5.35.

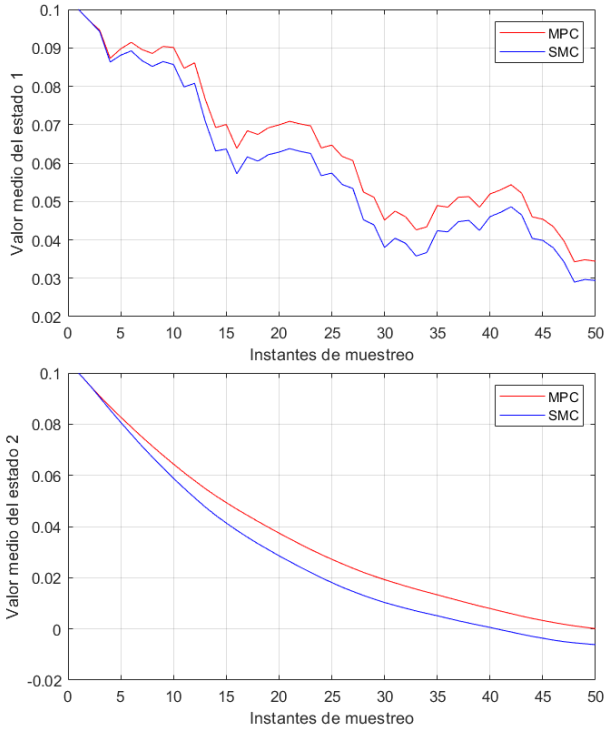

Figura 5.34 Media instantánea de los estados para 50 ensayos de un sistema con perturbaciones aditivas.

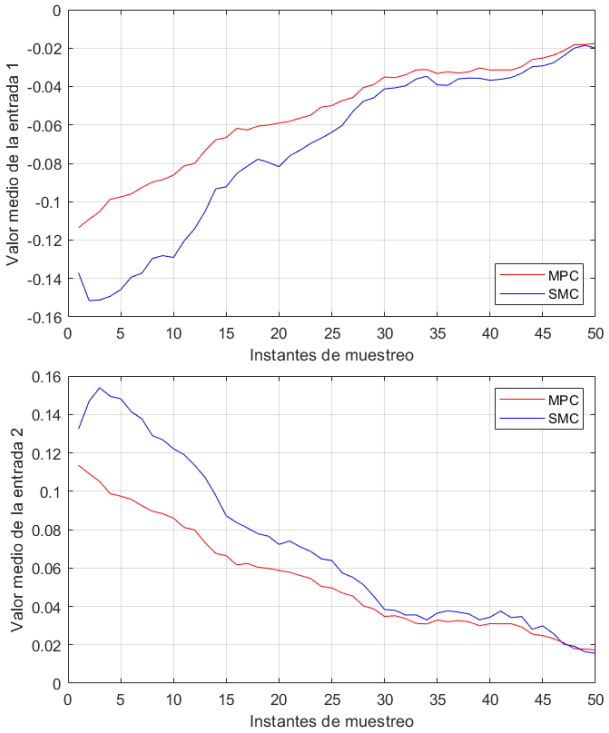

Figura 5.35 Media instantánea de las entradas para 50 ensayos de un sistema con perturbaciones aditivas.

Puede verse como ambos algoritmos tienen un comportamiento muy parecido, aunque el SMC parece más agresivo en sus acciones de control. Eso llevará a que en la función de coste resulte más penalizado en el término correspondiente a las entradas.

5.9 Sistemas politópicos

Los sistemas politópicos son un tipo de sistemas variantes en el tiempo. Es decir, se trata de un sistema lineal definido por matrices A y B que cambian en cada instante de muestreo (LTV). En este caso, la evolución de estas matrices está restringida por una combinación convexa de una serie de matrices $(A_1, B_1), (A_2, B_2) ... (A_L, B_L)$.

$$[A \ B] = \sum_{i=1}^{L} \lambda_i [A_i \ B_i] \tag{5.3}$$

$$\sum_{i=1}^{L} \lambda_i = 1 \tag{5.4}$$

$$\lambda_i \geq 0 \tag{5.5}$$

Por tanto, en cada instante de muestreo, se calcularán el conjunto $[A \ B]$ a partir de los valores de los λ_i correspondientes.

Para nuestros ensayos, el sistema objeto de estudio es el siguiente [12]:

$$A_1 = \begin{bmatrix} 0.9347 & 0.5194 \\ 0.3835 & 0.8310 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0.0591 & 0.2641 \\ 1.7971 & 0.8717 \end{bmatrix} \quad B = \begin{bmatrix} -1.4462 \\ -0.7012 \end{bmatrix}$$

En cada instante, se obtiene un λ_1 de una distribución triangular (véase figura 5.36), de manera que el otro λ_2 se calcula a partir de la restricción 5.4.

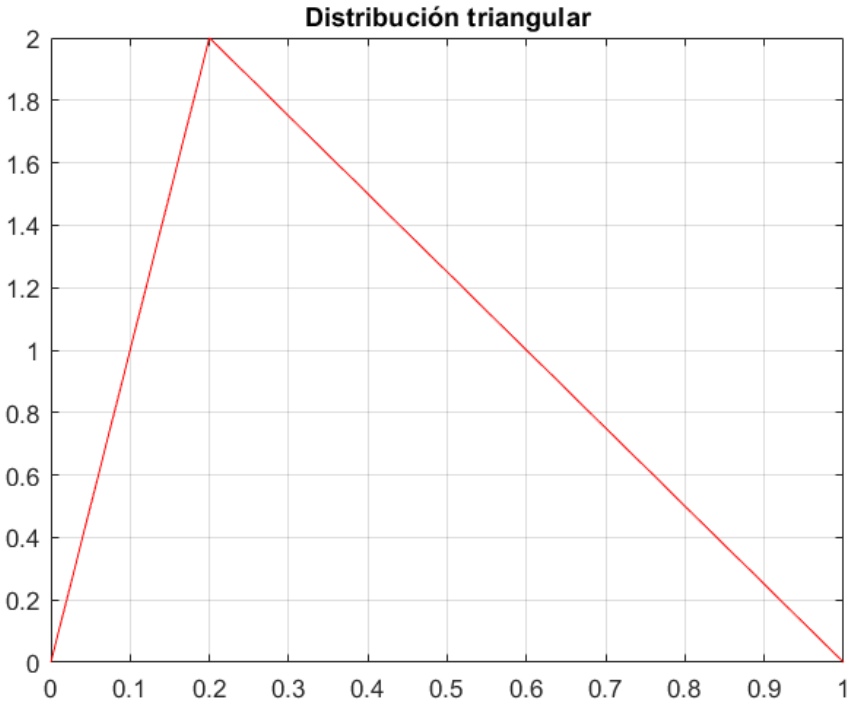

Figura 5.36 Distribución usada en los ensayos del sistema politópico.

Es fácil observar que la distribución no está centrada en el punto medio del rango (el cual se corresponde con la sintonización del controlador MPC).

Además, se añadirá una señal de ruido a la evolución del estado del sistema extraída de una distribución uniforme en el intervalo $[-0.05, 0.05]$.

Por otro lado, la función de coste incluye tanto el error en seguimiento como un término de esfuerzo de control ponderado.

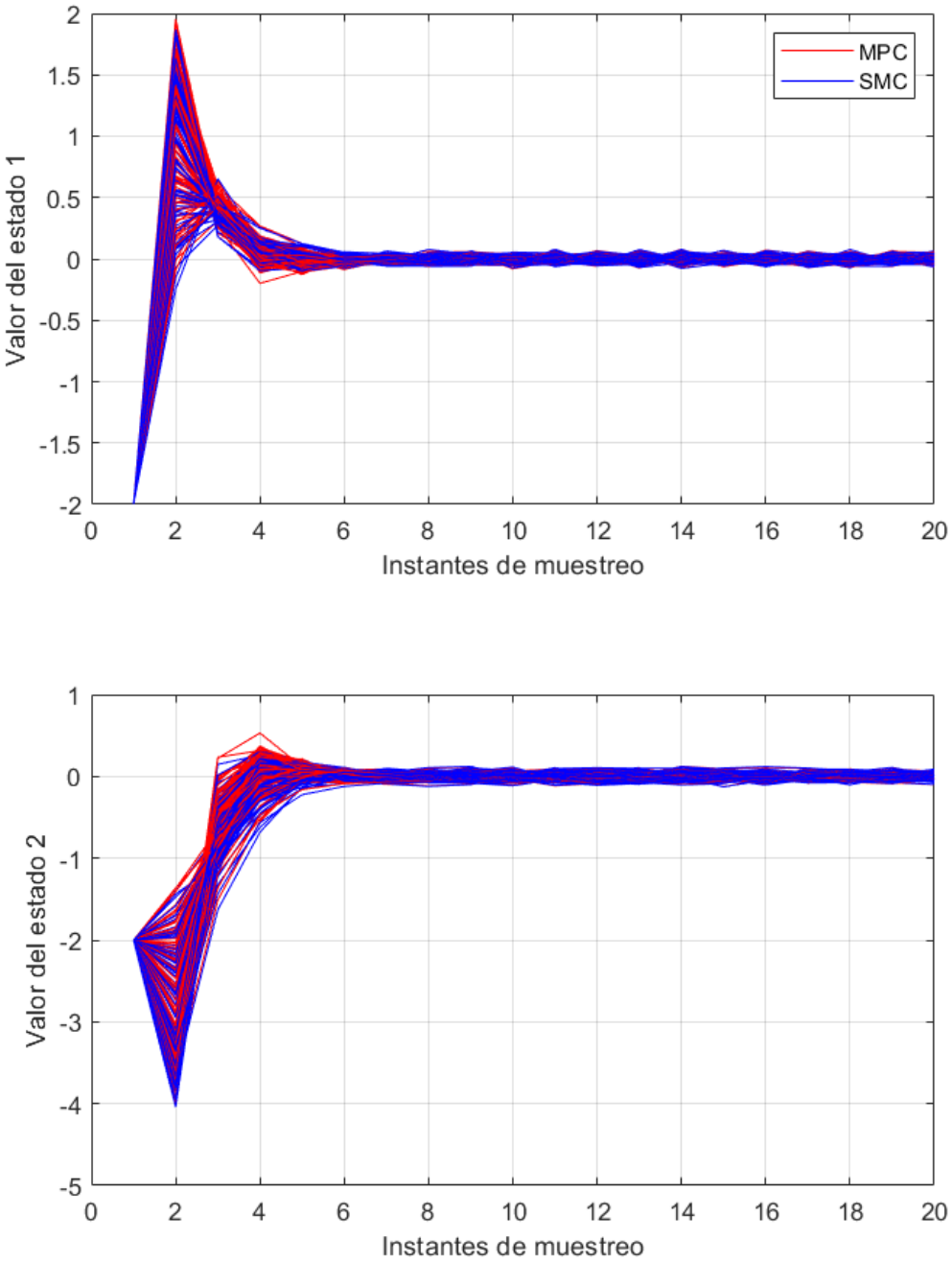

Figura 5.37 Evolución de los estados para los 100 ensayos.

Los resultados de la figura 5.37 no son especialmente claros. En algunas ocasiones parece que el MPC sobreoscila más que el el SMC, pero en otras ocasiones ocurre exactamente al contrario. Por esta razón, parece necesario también presentar otras gráficas que puedan aclarar la bondad de cada uno de los métodos.

Para ello, se calcula el coste en cada instante de muestreo (error de seguimiento y esfuerzo de control) y se le calculan diversos parámetros estádisticos (media, varianza) con los datos de todos los ensayos, con el fin de comparar los resultados para cada algoritmo en las figura 5.38.

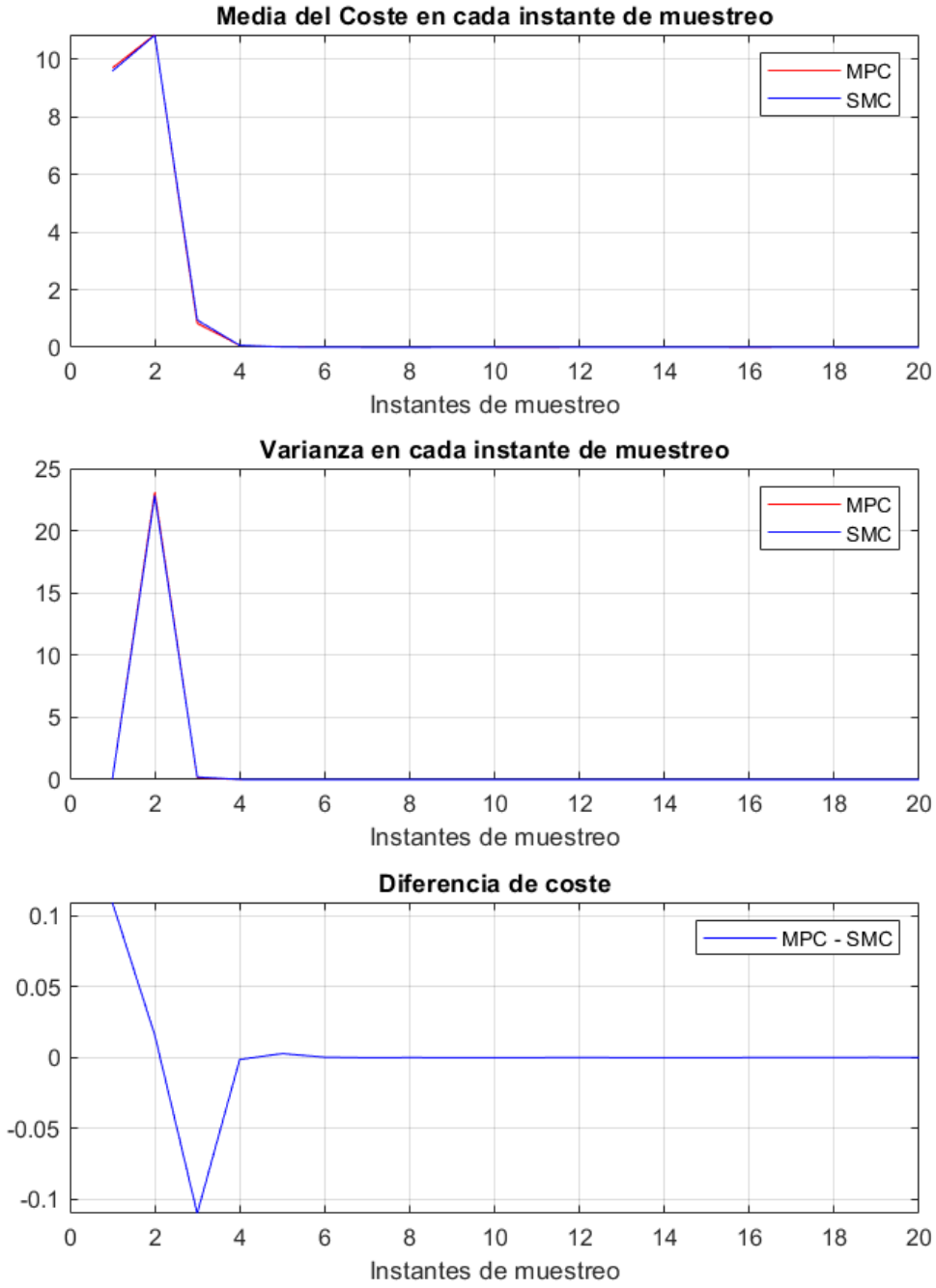

Figura 5.38 Evolución de los estados para los 100 ensayos.

Puede verse como ambos métodos difieren muy poco el uno del otro. En algunos instantes de muestreo, el SMC presenta un tenue mejor funcionamiento , y lo mismo podría decirse para el MPC, por lo que no puede llegarse realmente a una conclusión respecto de cuál resulta más ventajoso de los dos en este aspecto.

6 Conclusiones

En el presente trabajo se ha programado un algoritmo de Control Predictivo basado en técnicas de Montecarlo y se han generado diversas variantes originales con el objetivo de mejorar su rendimiento y funcionamiento las cuáles, efectivamente, fueron capaces de mejorar el comportamiento del algoritmo base presentado en la literatura.

A su vez, se han probado con problemas de control tales como la planta de los 4 tanques y otros presentes en diversas publicaciones especificadas en la bibliografía del proyecto. En el capítulo de resultados ha sido posible comprobar que los algoritmos realmente funcionan, aunque en algunas ocasiones no proporcionasen ninguna mejora respecto al MPC lineal que hemos considerado como caso base.

Esto no tiene por qué sorprendernos debido a que en la bibliografía proporcionada no se hacen comparaciones de funcionamiento de estos algoritmos con otros métodos alternativos, por lo que realmente no teníamos motivos para pensar que tuviese que existir una mejora.

Sin embargo, esto tampoco significa que el algoritmo tenga que ser desechado. Existen gran cantidad de sistemas cuyos modelos no pueden resolverse con los métodos tradicionales. Por esta razón, una de las líneas futuras que se espera continuar está enfocada en este tipo de sistemas.

Por otra parte, se ha propuesto una determinada paralelización del algoritmo, aunque esta podría llevarse a cabo de una manera diferente e incluso más exhaustiva. Podría hacerse uso de los métodos de paralelización dinámica para evitar el bucle secuencial de los escenarios de ruido, lo cual podría reducir todavía más los tiempos de computación.

Índice de Figuras

Bibliografía

[1] D. Rodríguez y C. Bordóns, *Apuntes de Ingeniería de Control*, Revisión del año 2007.

[2] James B. Rawlings, David Q. Mayne y Moritz M. Diehl, *Model Predictive Control: Theory, Computation, and Design, 2nd Edition.*

[3] Daniel Limón Marruedo, *Model Predictive Control for changing operating conditions.*

[4] K. H. Johansson, *"The quadruple-tank process"*, IEEE Transactions on Control Systems Technology, vol 8 (2000).

[5] A. D. Carnerero, *"Procesamiento masivamente paralelo en control predictivo basado en datos."* TFG , 2017.

[6] A. L. Visintini, W. Glover, J. Lygeros y J. Maciejowski, *"Monte Carlo Optimization for Conflict Resolution in Air Traffic Control"*, IEEE Transactions on intelligent transportation systems, Vol. 7, Nº 4 2006.

[7] J. P. Villiers, S. Godsill y S. Singh, Particle predictive control. Journal of Statistical Planning and Inference, 141:1753–1763, 2011.

[8] T. Takagi y M. Sugeno, Fuzzy Identification of Systems and Its Applications to Modeling and Control. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-15, No. 1, 1985.

[9] S. L. Chiu, Fuzzy Model Identification Based on Cluster Estimation. Journal of Intelligent and Fuzzy Systems, 1994.

[10] I. Batina, *"Model predictive control for stochastic systems by randomized algorithms."* Tesis doctoral, 2004.

[11] G. Kitagawa, *"Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models."*, Journal of Computational and Graphical Statistics, Volume 5, Number 1, 1996.

[12] M. V. Kothare, V. Balakrishnan y M. Morari, *"Robust Constrained Model Predictive Control using Linear Matrix Inequalities."*, Automatica, Volume 32, Number 10, 1996.