UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de herramientas de optimización bayesiana para sintonizado experimental de controladores en procesos industriales

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: Marqués Iglesias, Lucas

Tutor/a: Sala Piqueras, Antonio

CURSO ACADÉMICO: 2023/2024

RESUMEN

El presente TFM aborda la temática de sintonizado experimental de procesos en los que es alta la duración o el coste de recursos a emplear en cada ensayo para validar prestaciones. Para minimizar el número de ensayos, se realiza una optimización fuera de línea de una función subrogada (de adquisición) basada en un modelo estadístico (proceso gaussiano) que interpola las muestras experimentales obtenidas en los supuestamente costosos ensayos. El TFM desarrollará herramientas de optimización bayesiana en Matlab y las aplicará al sintonizado óptimo de controladores de una serie de procesos industriales, validando la propuesta metodológica mediante simulaciones exhaustivas.

Palabras Clave: Control de procesos, Optimización experimental, procesos estocásticos, control PID, simulación, optimización bayesiana.

RESUM

El present TFM aborda la temàtica de la sintonització experimental de processos en els quals és alta la duració o el cost de recursos a emprar en cada assaig per a validar prestacions. Per a minimitzar el nombre d'assajos, es realitza una optimització fora de línia d'una funció substitutiva (d'adquisició) basada en un model estadístic (procés gaussià) que interpola les mostres experimentals obtingudes en els suposadament costosos assajos. El TFM desenvoluparà eines d'optimització bayesiana en Matlab i les aplicarà a la sintonització òptima de controladors d'una sèrie de processos industrials, validant la proposta metodològica mitjançant simulacions exhaustives.

Paraules Clau: Control de processos, optimització experimental, processos estocàstics, control PID i simulació, optimització bayesiana.

ABSTRACT

This M.Sc.Thesis addresses the issue of experimental tuning of processes in which the duration, or the cost of resources to be used in each test to validate performance, is high. To minimize the number of trials, an offline optimization of a surrogate (acquisition) function is performed based on a statistical model (Gaussian process) that interpolates the experimental samples obtained in the supposedly expensive trials. The M.Sc.Thesis will develop Bayesian optimization tools in Matlab and apply them to the optimal tuning of controllers of a series of industrial processes, validating the proposal in exhaustive simulations.

Keywords: Process control, experimental optimization, stochastic processes, PID control, simulation, Bayesian optimization

ÍNDICE

DOCUMENTOS CONTENIDOS EN EL TFM

- Memoria
- Presupuesto
- Anexos

ÍNDICE DE LA MEMORIA

ÍNDICE DEL PRESUPUESTO

ÍNDICE DE LOS ANEXOS

ÍNDICE DE LAS FIGURAS DE LA MEMORIA

ÍNDICE DE LAS FIGURAS DE LOS ANEXOS

ÍNDICE DE LAS TABLAS DE LA MEMORIA

ÍNDICE DE LAS TABLAS DEL PRESUPUESTO

ÍNDICE DE LAS TABLAS DEL ANEXO

MEMORIA

1. INTRODUCCIÓN

1.1. CONTEXTO ACTUAL

En los últimos años, la optimización de procesos industriales ha cobrado una relevancia cada vez mayor debido a la necesidad de maximizar la eficiencia y reducir los costos operativos. En un entorno globalizado y altamente competitivo, las empresas se ven obligadas a mejorar continuamente sus procesos para mantenerse a la vanguardia. Además, la creciente presión por lograr la sostenibilidad y reducir la huella de carbono ha llevado a que las estrategias de optimización no solo se centren en la rentabilidad económica, sino también en su impacto ambiental [1].

El avance de las tecnologías digitales, como la inteligencia artificial, el aprendizaje automático y la automatización avanzada, ha transformado por completo la manera en que las industrias gestionan y optimizan sus procesos. Estas herramientas permiten realizar análisis en tiempo real de las variables críticas del sistema y ajustarlas sobre la marcha, mejorando significativamente el rendimiento, reduciendo desperdicios y minimizando errores [2]. En este contexto, la optimización bayesiana ha destacado como una herramienta eficaz para optimizar procesos donde la evaluación de diferentes configuraciones resulta costosa y el tiempo es un factor limitante.

Un ejemplo típico de este tipo de problemas complejos es el control del péndulo invertido, una configuración clásica en la teoría de control que se utiliza como banco de pruebas para nuevas estrategias de optimización y control. Este sistema es altamente no lineal y tiene una fuerte tendencia a la inestabilidad, lo que lo convierte en un desafío ideal para explorar técnicas avanzadas de optimización, especialmente cuando los recursos para llevar a cabo experimentos son limitados o costosos. Dada su complejidad, surge la necesidad de implementar métodos que permitan ajustar los parámetros de los controladores de manera eficiente, minimizando el número de experimentos necesarios para validar el rendimiento.

En este contexto, la optimización bayesiana se presenta como una herramienta clave, ya que permite realizar ajustes finos en sistemas donde las evaluaciones son costosas o difíciles de realizar [3]. Al utilizar modelos probabilísticos, esta técnica ayuda a explorar las mejores soluciones reduciendo el número de pruebas físicas necesarias.

Este Trabajo Fin de Máster (TFM) se centra en desarrollar e implementar técnicas avanzadas de optimización bayesiana para el ajuste de controladores en sistemas industriales complejos, utilizando el péndulo invertido como caso de estudio. El objetivo principal es reducir significativamente el número de experimentos necesarios para validar los controladores, lo que mejorará tanto la eficiencia del proceso como su aplicabilidad en otros sistemas industriales donde los recursos son limitados o costosos.

1.2. OBJETIVOS

1. Desarrollar un sistema de control óptimo para un proceso inestable como el péndulo invertido.

Este objetivo se centra en maximizar la estabilidad y el rendimiento del sistema de control, enfrentando el desafío de un proceso inestable como el péndulo invertido. Se busca garantizar la eficiencia del sistema bajo diferentes condiciones operativas y mejorar la respuesta ante perturbaciones externas.

2. Validar la optimización bayesiana de manera offline mediante simulaciones, como método para optimizar el controlador inicial desarrollado.

Este objetivo busca demostrar que la optimización bayesiana offline es efectiva para ajustar el controlador inicial ya desarrollado. A través de simulaciones, se pretende comprobar que el proceso de optimización funciona correctamente, permitiendo ajustar los parámetros del sistema de manera precisa

3. Optimizar el controlador en el entorno operativo mediante optimización bayesiana en tiempo real.

El objetivo es implementar el controlador previamente validado en simulación dentro del proceso real y, a partir de ahí, utilizar la optimización bayesiana en tiempo real para ajustar y mejorar los parámetros del controlador bajo condiciones operativas reales. Esto permitirá garantizar que el sistema sea robusto y eficiente frente a las variaciones que puedan surgir durante su funcionamiento.

4. Desarrollar un enfoque de optimización adaptable a otros sistemas industriales

Finalmente, se busca crear una metodología de optimización lo suficientemente flexible como para ser aplicable a otros sistemas industriales. Aunque el péndulo invertido es el caso de estudio, este enfoque debe poder adaptarse a otros procesos de control que requieran optimización con un número limitado de experimentos, asegurando así su viabilidad en distintas aplicaciones industriales.

2. FUNDAMENTOS Y ESTRATEGIA DE IMPLEMENTACIÓN

2.1. ESTRATEGIA GLOBAL DE CONTROL ÓPTIMO

El objetivo principal de la Estrategia Global de Control Óptimo es crear un sistema de control que maximice la eficiencia de un proceso. Para lograrlo, el controlador debe minimizar la función de costo del proceso. Esta estrategia es fundamental para lograr alcanzar los objetivos deseados cumpliendo con las restricciones y respetando las propiedades del sistema.

2.1.1. OBJETIVO Y CONTEXTO

En muchos sistemas dinámicos, el objetivo principal es gestionar el funcionamiento del sistema para minimizar el valor de una función de costo relacionada con su rendimiento. Esta función puede estar asociada a diversos factores, como el seguimiento de la referencia, la capacidad del sistema para manejar perturbaciones, el valor de la acción de control o el tiempo de respuesta.

Este proyecto se enfoca en la optimización de la norma L^2, que representa la integral del cuadrado de la señal de error. En este caso, la norma L^2 se utiliza para controlar el seguimiento de las referencias, midiendo las diferencias entre el valor deseado en cada salida y su valor real, y para minimizar el valor de la acción de control. Dentro de esta estrategia, la norma L^2 es fundamental, ya que permite medir el comportamiento energético de las señales de error en el sistema. Al minimizar esta norma, se busca no solo asegurar que el sistema responda de manera eficiente y efectiva a las señales de referencia, sino también reducir los errores tanto en magnitud como en duración, resultando en un sistema más preciso y eficiente en el uso de recursos como la energía.

2.1.2. ENFOQUE METODOLÓGICO

El enfoque metodológico para alcanzar un control óptimo global se desglosa en las siguientes etapas clave, diseñadas para asegurar la precisión, eficacia y adaptabilidad del sistema de control:

1. Modelado del Sistema.

   El primer paso es crear un modelo matemático que describa con precisión el sistema a controlar. Dependiendo de las características del proceso, este modelo puede ser lineal o no lineal. Durante este proceso, es importante identificar las variables clave como las entradas, salidas y las dinámicas que rigen el comportamiento del sistema.

2. Definición de la Función de Costo.

   Se establece una función de costo que permita medir qué tan bien se está comportando el sistema respecto a lo que se espera de él. Esta función será la referencia que guiará el diseño del controlador para optimizar el rendimiento del sistema.

3. Configuración del Problema de Optimización

   Aquí se define el problema que se quiere resolver, buscando minimizar la función de costo considerando las restricciones y dinámicas del sistema. Este paso asegura que el controlador busque el equilibrio adecuado entre el rendimiento y las limitaciones del entorno.

4. Métodos de Control

Se seleccionan los métodos de control más adecuados, basándonos en las necesidades del sistema y las metas establecidas. Esto puede incluir técnicas clásicas de control o métodos más avanzados, dependiendo de las exigencias del proyecto.

5. Implementación y Simulación

Las estrategias de control diseñadas se prueban mediante simulaciones. Estas pruebas buscan recrear diversos escenarios operativos para verificar si el controlador funciona de manera eficiente en distintas condiciones.

6. Evaluación de Resultados en Simulación

Luego de las simulaciones, se analizan los resultados obtenidos. Se revisan métricas clave para evaluar el desempeño del controlador y determinar qué áreas necesitan ajustes o mejoras para seguir afinando la estrategia.

7. Validación con el Proceso Real

Se lleva a cabo la implementación y validación del controlador optimizado en el sistema real. Se monitoriza su comportamiento para asegurarse de que el rendimiento en condiciones reales sea acorde con lo obtenido en simulación.

8. Ajuste Fino y Optimización Experimental

Durante las pruebas reales, se ajustan los parámetros del controlador de manera experimental para refinar su rendimiento. El objetivo es asegurar que el sistema funcione de manera óptima frente a variaciones no modeladas en la simulación.

9. Validación de la Configuración Final y Entrega a Cliente

Después de finalizar los ajustes y optimización, se realiza una validación completa del sistema. Una vez validado, el controlador se entrega para su uso en producción, garantizando que cumple con todas las especificaciones de rendimiento.

2.1.3. APLICACIONES PRÁCTICAS

Este enfoque es aplicable a diversos sectores industriales dada su robustez teórica y su versatilidad. Algunas de las áreas donde esta metodología puede ser de utilidad son las siguientes:

1. Automatización Industrial

En la automatización industrial, el control óptimo es fundamental para desarrollar soluciones más eficientes y reducir los costos operativos. Este enfoque permite a las industrias minimizar el desperdicio de recursos al optimizar su uso, al mismo tiempo que mejora la calidad del producto final. Además, establece sistemas de control adaptables a las variaciones, lo cual es esencial para mejorar la eficiencia en procesos que presentan variabilidad en el tiempo.

2. Robótica

En el ámbito de la robótica, los sistemas de control óptimo son fundamentales para el desarrollo de robots inteligentes y adaptativos. La capacidad de ajustar los parámetros al

instante es clave para su operación en entornos impredecibles. Además, estos sistemas permiten a los robots ejecutar tareas complejas con precisión y fiabilidad.

3. Transporte Inteligente

El control óptimo se está utilizando ampliamente en el sector del transporte inteligente, donde la coordinación instantánea basada en datos actualizados en tiempo real, junto con un uso eficiente de las infraestructuras, ha permitido desarrollar sistemas más seguros y eficientes, además de contribuir a la reducción del tráfico.

4. Energía y Gestión de Redes Eléctricas

En el sector energético, el control óptimo es clave para la gestión eficiente de redes eléctricas, especialmente en la integración de energía renovable. Permite equilibrar en tiempo real la oferta y la demanda, asegurando la estabilidad de la red a la vez que se optimiza el uso de los recursos. Esto es clave en el momento actual en el cual hay una gran variabilidad en la generación de la energía.

En resumen, el control óptimo es fundamental para reducir costos operativos y garantizar que los sistemas de control se adapten de manera efectiva a los cambios del entorno. Este método fomenta la innovación y asegura el éxito de diversas aplicaciones industriales, resaltando su adaptabilidad e importancia en el contexto tecnológico actual.

2.2. CONTROLADOR INICIAL

El Control Basado en Modelos es una estrategia muy empleada en el desarrollo de sistemas de control avanzados, donde se utiliza un modelo matemático para representar el proceso real. Este enfoque facilita el diseño de controladores que no solo aseguren un funcionamiento preciso del proceso, sino que también ofrezcan robustez y versatilidad.

Al usar un modelo matemático que simula con precisión cómo se comporta el sistema, se puede prever cómo reaccionará ante cambios y perturbaciones. Esto permite ajustar los controladores para que trabajen de manera eficiente, incluso en situaciones inesperadas o complicadas. Además, este enfoque facilita hacer simulaciones previas, lo que ayuda a optimizar el diseño y reducir riesgos antes de llevar el sistema de control al proceso real. Esto resulta especialmente importante en sistemas complejos, donde los controladores deben ser adaptables y garantizar la estabilidad del sistema.

En esta sección, vamos a centrarnos en dos métodos de control que se aplicarán más adelante para resolver el problema planteado: el control LQR (Linear Quadratic Regulator) y el μ-Synthesis (Musyn). Ambos son buenos ejemplos de cómo un enfoque basado en modelos puede mejorar la precisión, estabilidad y eficiencia de un sistema de control.

2.2.1. CONTROL LQR

El Control LQR es un método utilizado para diseñar controladores que minimicen una función de costo cuadrática. Esta función de costo penaliza tanto las desviaciones del estado del sistema respecto a su punto de referencia como el esfuerzo necesario para corregir dichas desviaciones. El objetivo del

LQR es lograr un equilibrio entre el rendimiento del sistema y la acción de control, asegurando una respuesta rápida y estable.

Una de las principales ventajas del LQR es su capacidad para manejar múltiples variables de estado simultáneamente, lo que lo hace ideal para sistemas con muchas variables en juego. Además, permite ajustar los pesos en la función de costo, pudiéndose priorizar ciertos aspectos del sistema sobre otros. Por ejemplo, en un sistema de control de temperatura industrial, se podría dar mayor importancia a la precisión en el mantenimiento de la temperatura deseada que a la velocidad de respuesta del sistema.

El LQR es eficiente a nivel de cálculo, lo que facilita su uso en tiempo real, incluso en sistemas con recursos limitados. Al estar basado en fórmulas matemáticas, ofrece una solución que es fácil de predecir y evaluar, lo cual es clave en aplicaciones donde la seguridad es lo más importante.

2.2.2. MUSYN (M-SYNTHESIS)

La Sintonización Musyn (μ-Synthesis) es un método de control diseñado para sistemas con mucha incertidumbre o que presentan no linealidades. A diferencia del LQR, que se enfoca en sistemas lineales, Musyn es más adecuada para situaciones en las que el modelo del sistema no es completamente conocido o donde pueden ocurrir cambios con el tiempo. Musyn ayuda a diseñar controladores robustos que mantienen un buen rendimiento incluso cuando el sistema enfrenta incertidumbres o variaciones en su comportamiento.

Musyn es especialmente útil en escenarios donde no se puede ignorar la incertidumbre y donde es importante que el sistema funcione correctamente en bajo diferentes condiciones. En lugar de optimizar para un conjunto fijo de parámetros, Musyn se enfoca en minimizar el impacto de las peores perturbaciones posibles. Esto hace de ella una herramienta muy valiosa en el sector industrial, donde los sistemas deben ser confiables y estables, incluso en las peores condiciones.

El diseño de controladores mediante Musyn es un proceso iterativo, lo que significa que se ajustan los parámetros varias veces hasta que el controlador esté listo para manejar las incertidumbres de la mejor manera posible. Este proceso está respaldado por herramientas de software que ayudan a afinar el controlador, asegurando que el sistema opere de manera segura y eficiente, incluso en situaciones imprevistas. Como resultado se obtiene un sistema de control robusto, que es capaz de rendir de manera óptima incluso cuando se enfrenta a condiciones difíciles.

2.3. CONTROL DE SISTEMAS DINÁMICOS

El control de sistemas dinámicos es una parte muy importante de la ingeniería de control, dedicada a manejar cómo se comportan sistemas que cambian con el tiempo. Estos sistemas pueden ser más o menos simples, lo que los caracteriza es su capacidad para responder a diferentes entradas o perturbaciones. El objetivo es que dichos sistemas operen como se desee, a pesar de cualquier cambio ya sea externo interno.

En un sistema dinámico, las variables del sistema, como la posición, velocidad, temperatura o presión, varían con el tiempo. La dificultad en el control de estos sistemas radica en que muchas veces son no lineales o son inestables. Aunque un sistema pueda describirse y modelarse de manera relativamente sencilla mediante ecuaciones matemáticas, lograr un control efectivo requiere una estrategia cuidadosa que tenga en cuenta tanto la estabilidad como la capacidad de respuesta.

Los sistemas dinámicos como se ha comentado anteriormente están presentes en casi todas las áreas de la ingeniería y la tecnología. Para controlar estos sistemas, es necesario diseñar controladores que puedan prever cómo se comportará el sistema en el futuro y aplicar las correcciones necesarias antes de que ocurran cambios importantes. Para esto es necesaria una retroalimentación constante y la capacidad del sistema de control de anticiparse a posibles cambios en el entorno o en las condiciones en que opera el sistema.

Caso del Péndulo Invertido

Uno de los ejemplos más conocidos en el control de sistemas dinámicos es el péndulo invertido. Este es un sistema inestable en el que el pivote está por encima del centro de masas del péndulo. Lo que quiere decir que si no se controla de manera adecuada este caerá por el simple efecto de la fuerza de la gravedad, alejándose así de su posición vertical. Mantener el péndulo por lo tanto en esa posición requiere de un método de control preciso, esto convierte este caso en uno ideal para probar y perfeccionar diferentes técnicas de control.

Además, el problema del péndulo invertido no es algo únicamente teórico; es una representación simplificada de muchos sistemas más complejos que deben mantenerse estables a pesar de su tendencia natural a la inestabilidad. Los vehículos autónomos enfrentan desafíos similares, ya que mientras operan deben mantenerse dinámicamente equilibrados.

Controlar el péndulo invertido implica aplicar fuerzas que contrarresten cualquier movimiento que lo aleje de su posición vertical. Esto requiere medir constantemente su posición y velocidad, y ajustar el control en tiempo real para mantenerlo estable. Técnicas como el control LQR y el μ-Synthesis (Musyn) son particularmente útiles en este contexto. Mientras que el LQR se centra en minimizar una función de costo cuadrática, Musyn es más robusto frente a incertidumbres y variaciones en el sistema.

Al analizar y comparar estos métodos a través de simulaciones y pruebas, podremos determinar cuál ofrece el mejor rendimiento para mantener la estabilidad del péndulo bajo diferentes condiciones. Este análisis no solo permitirá comprender mejor las técnicas de control empleadas, sino que también proporcionará una base sólida para aplicarlas en otros sistemas dinámicos complejos que requieran un control preciso y robusto.

2.4. Optimización Bayesiana

La optimización bayesiana es una estrategia que se ha destacado en varios campos industriales, ya que permite minimizar el número de evaluaciones necesarias para encontrar el valor óptimo de una función objetivo. En muchos escenarios industriales, cada prueba puede resultar extremadamente costosa en términos de tiempo, recursos o incluso riesgos operativos. Este método se basa en construir un modelo probabilístico de la función objetivo, y luego utilizar ese modelo para seleccionar de manera eficiente los puntos donde realizar futuras evaluaciones. Esta capacidad de reducir el número de pruebas necesarias es la razón por la que la optimización bayesiana ha ganado popularidad en entornos donde las evaluaciones tienen un coste elevado [4].

Una de las características más importantes de la optimización bayesiana es que no requiere suposiciones fuertes sobre la función objetivo; es decir, no necesita que la función sea lineal o derivable, ni que sea sencilla de modelar matemáticamente. En cambio, el modelo se construye mediante un enfoque iterativo que utiliza los datos recopilados de evaluaciones anteriores para mejorar continuamente sus predicciones. Esta metodología es particularmente útil cuando se trata con espacios de búsqueda amplios o funciones que presentan múltiples óptimos locales. Gracias a su estructura, la optimización bayesiana es capaz de ajustarse y refinar las predicciones, identificando los puntos más prometedores en los que realizar futuras evaluaciones.

2.4.1. Principios Básicos

La optimización bayesiana se basa en un proceso gaussiano que actúa como un modelo probabilístico de la función objetivo. Un proceso gaussiano permite predecir el valor esperado de la función en cualquier punto del espacio de búsqueda, junto con una estimación de la incertidumbre asociada a esa predicción. Esta capacidad de modelar la incertidumbre es una de las claves del éxito de la optimización bayesiana [5].

En cada iteración, se actualiza el proceso gaussiano con nuevos puntos evaluados, lo que reduce la incertidumbre en esas áreas del espacio de búsqueda. El objetivo es equilibrar la exploración y la explotación: por un lado, la exploración implica investigar áreas del espacio de búsqueda donde el modelo tiene mayor incertidumbre, mientras que la explotación se centra en áreas que ya han mostrado ser prometedoras para encontrar un óptimo. Este equilibrio es crucial para que la optimización bayesiana sea eficiente, maximizando la información obtenida de cada evaluación.

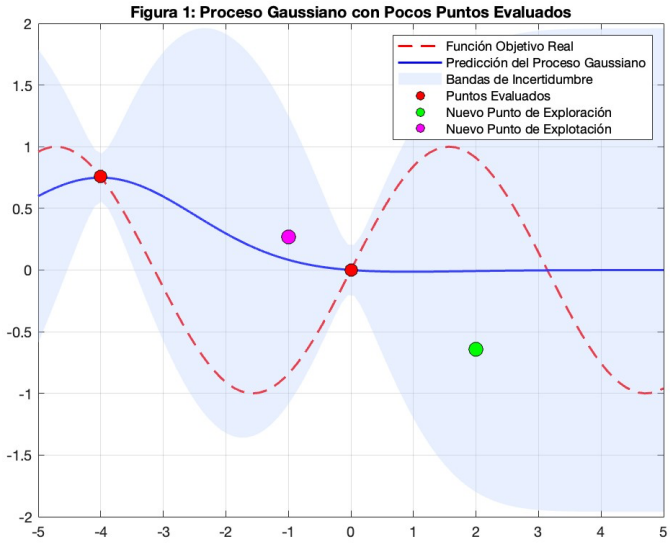


Figura 1. Optimización Bayesiana: Exploración vs Explotación

El modelo gaussiano se ajusta en cada iteración, y a medida que se obtienen más datos, la predicción se vuelve más precisa. Esto significa que las áreas con mayor incertidumbre tienden a disminuir a medida que se evalúan nuevos puntos, y el modelo se concentra en explotar las áreas que son más prometedoras. Este ciclo de predicción, evaluación y ajuste es lo que permite que la

optimización bayesiana converja al óptimo global con menos evaluaciones que otros métodos de optimización como el descenso de gradiente.

En la optimización bayesiana, este ciclo se repite hasta que se ha alcanzado un número predeterminado de iteraciones, o hasta que el algoritmo deja de encontrar mejoras significativas en la función objetivo. Este proceso es particularmente útil en escenarios donde las evaluaciones son costosas o difíciles de realizar, ya que el método minimiza el número de pruebas necesarias para encontrar una solución óptima.

2.4.2. PROCESO GAUSSIANO (GP)

El proceso gaussiano es un modelo probabilístico flexible y no paramétrico que se adapta bien a una amplia variedad de problemas. Al modelar las correlaciones entre diferentes puntos en el espacio de búsqueda, un proceso gaussiano no solo puede predecir valores en puntos conocidos, sino también estimar el comportamiento de la función objetivo en puntos no evaluados. Esta capacidad para modelar tanto el valor esperado como la incertidumbre lo convierte en una herramienta extremadamente poderosa para la optimización bayesiana.

Para un conjunto de puntos evaluados X y sus correspondientes valores observados Y, la predicción en un nuevo punto X_* se realiza a partir de la distribución condicional $p(Y_* \mid X_*, X, Y)$. La media posterior μ y la varianza posterior Σ se calculan mediante las siguientes ecuaciones:

$$\mu = K(X_*, X)[K(X, X) + \sigma^2 I]^{-1} Y$$

$$\Sigma = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma^2 I]^{-1} K(X, X_*)$$

Donde, $K(X, X)$ es la matriz de covarianza entre los puntos evaluados y σ^2 es la varianza del ruido en las observaciones.

En cada iteración, el proceso gaussiano se actualiza con nuevos puntos evaluados, lo que reduce la incertidumbre en esas áreas del espacio de búsqueda. El objetivo es equilibrar la exploración y la explotación: la exploración implica investigar áreas del espacio con mayor incertidumbre, mientras que la explotación se centra en áreas ya prometedoras para encontrar un óptimo local o global [6].

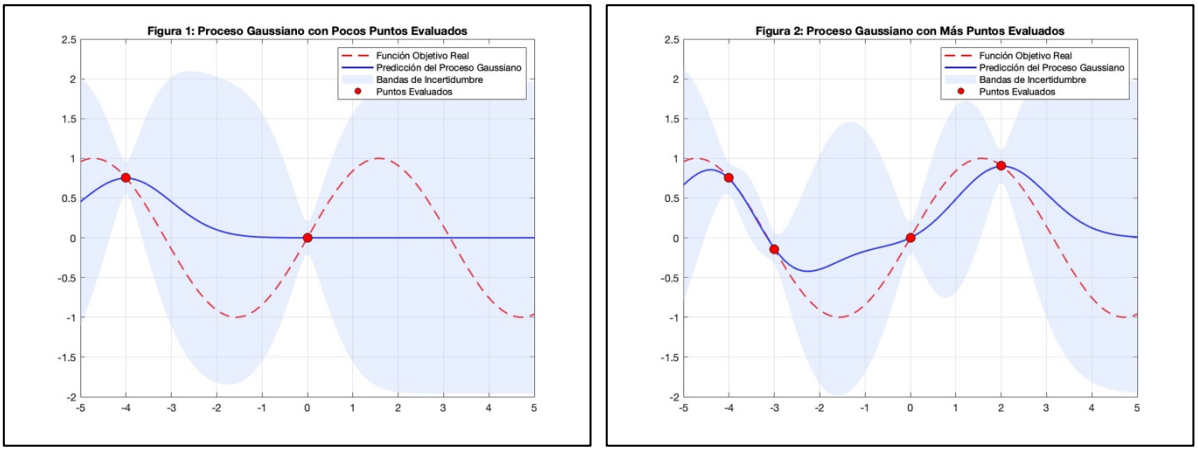


Figura 2. Proceso Gaussiano: Evolución de la Predicción y la Incertidumbre con Incremento de Puntos Evaluados

2.4.3. FUNCIÓN DE ADQUISICIÓN

La función de adquisición es uno de los componentes más importantes de la optimización bayesiana. Se utiliza para determinar en qué puntos del espacio de búsqueda se debe realizar la siguiente evaluación. Al utilizar la información proporcionada por el proceso gaussiano, la función de adquisición ayuda a equilibrar la exploración y la explotación [7].

Existen varias funciones de adquisición que se utilizan comúnmente, y cada una de ellas tiene diferentes formas de equilibrar la exploración y la explotación:

- **Expected Improvement (EI)**: Esta función selecciona puntos basándose en la mejora esperada sobre el mejor valor conocido. EI es particularmente eficaz en la explotación de áreas que ya han mostrado buenos resultados, maximizando el beneficio esperado de cada evaluación. Su expresión es la siguiente:

$$EI(x) = (Y_{best} - \mu(x)) \cdot \Phi(Z) + \sigma(x) \cdot \phi(Z)$$

$$Z = \frac{Y_{best} - \mu(x)}{\sigma(x)}$$

  Donde Φ y ϕ son la función de distribución acumulada y la función de densidad de una normal estándar, respectivamente. Aquí, $\mu(x)$ es la predicción media de la función objetivo en el punto x, $\sigma(x)$ es la incertidumbre o desviación estándar asociada con esa predicción, y Y_{best} es el mejor valor conocido hasta el momento.

- **Probability of Improvement (PI)**: PI selecciona puntos que tienen una alta probabilidad de mejorar el mejor valor conocido, priorizando la explotación sobre la exploración. Esta función tiende a centrarse en áreas que ya han sido bien exploradas, lo que la hace adecuada cuando se quiere maximizar el valor en un espacio de búsqueda limitado. Su expresión es la siguiente:

$$PI(x) = \Phi\left(\frac{Ybest - \mu(x)}{\sigma(x)}\right)$$

- **Upper Confidence Bound (UCB)**: Combina el valor esperado con la incertidumbre asociada, lo que permite una mayor exploración en áreas con alta incertidumbre. UCB es útil cuando se necesita equilibrar de manera eficiente la exploración de nuevas áreas y la explotación de las áreas más prometedoras. Su expresión es la siguiente:

$$UCB(x) = \mu(x) + \kappa \cdot \sigma(x)$$

  Donde κ es un parámetro que controla el equilibrio entre exploración y explotación.

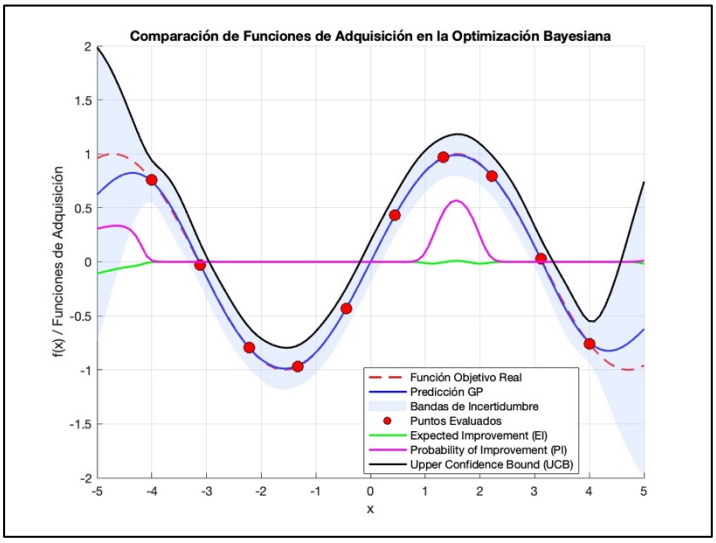


Figura 3. Comparación de Funciones de Adquisición (EI, PI y UCB) en la Optimización Bayesiana

Elegir la función de adquisición adecuada y ajustarla correctamente es esencial para maximizar la eficiencia de la optimización bayesiana. Una función de adquisición bien ajustada permite que el algoritmo converja al óptimo global con el menor número de evaluaciones posibles, lo que es clave en situaciones donde el coste de cada prueba es elevado.

2.5. Restricciones de Seguridad en la Optimización

La optimización de procesos industriales, aunque eficaz, puede implicar riesgos considerables si no se manejan adecuadamente las restricciones de seguridad. En el contexto de la ingeniería de control, estas restricciones son fundamentales para garantizar que el sistema funcione dentro de límites seguros, evitando daños materiales y humanos.

Cuando se implementa un proceso de optimización, el objetivo principal suele ser mejorar la eficiencia o el rendimiento de un sistema. Sin embargo, en muchos casos, este enfoque pasa por alto ciertos límites que deben respetarse si no se desea poner en riesgo la estabilidad del sistema o incluso causar fallos catastróficos. En este contexto es donde entran en juego las restricciones de seguridad, que se encargan de asegurar que el sistema siempre opere dentro de márgenes seguros, independientemente de los parámetros optimizados [8].

Las restricciones de seguridad pueden adoptar diversas formas, dependiendo del tipo de sistema que se esté optimizando. Pueden incluir límites en las variables de estado, así como restricciones en las entradas al sistema. También pueden estar relacionadas con la estabilidad del sistema, asegurando que las modificaciones en los parámetros de control no produzcan inestabilidades o un comportamiento impredecible del sistema.

Incorporar estas restricciones en el proceso de optimización no es fácil; para ello, se requiere un análisis cuidadoso que permita definir qué límites son críticos y cómo deben implementarse en los algoritmos de optimización. Es aquí donde enfoques como la optimización con restricciones de seguridad, o "Safe Optimization", juegan un papel crucial. Este enfoque garantiza que, durante el

proceso de optimización, no se realicen pruebas que podrían comprometer la seguridad del sistema [9].

La técnica que más ha destacado en este campo ha sido SafeOpt. Se trata de una técnica que permite explorar de manera segura, asegurando que el algoritmo utilizado para llevar a cabo la optimización del sistema no lo lleve a estados inseguros. A medida que se explora en el espacio de búsqueda, SafeOpt asegura que el sistema permanezca dentro de los límites de seguridad previamente definidos [10].

La aplicación de SafeOpt en la optimización de sistemas como el péndulo invertido, por ejemplo, puede ser clave a la hora de evitar que, durante el proceso de ajuste de los controladores, el péndulo se desestabilice o se mueva fuera de su rango seguro de operación. Esto no solo protege el sistema, sino que también, al mejorar la seguridad en el entorno, los ingenieros pueden enfocar sus esfuerzos en lograr los objetivos establecidos sin tener que preocuparse por su propia seguridad.

En conclusión, la integración de restricciones de seguridad en los procesos de optimización es crucial en el desarrollo de sistemas robustos y seguros. Aunque la optimización busca maximizar el rendimiento, nunca debe hacerse a expensas de la seguridad [11].

2.6. Desarrollo y Ejecución del Proyecto

Basándose en la Estrategia Global de Control Óptimo, el proyecto incluye los siguientes pasos:

1. Desarrollo de Herramientas de Optimización:

   Se desarrollan dos programas en Matlab. OptBayes se encarga de la optimización bayesiana de un proceso industrial específico, mientras que SafeOptBayes incorpora restricciones de seguridad para garantizar que el proceso optimizado cumpla con las normativas de seguridad establecidas.

2. Pruebas en Modelos Sencillos:

   Ambos programas se prueban en un modelo sencillo de control PID, lo que permite validar su eficacia antes de aplicarlos en un sistema industrial más complejo.

3. Diseño de Controladores

   Se desarrollan controladores utilizando un modelo linealizado del sistema, aplicando tanto el enfoque LQR como el método Musyn.

4. Implementación en modelo no lineal y evaluación para optimización:

   Los controladores se prueban en un modelo no lineal del sistema, evaluando su desempeño para determinar si presentan un comportamiento adecuado como base para la optimización bayesiana.

5. Aplicación de la optimización bayesiana y evaluación de los resultados

Se lleva a cabo un proceso de revisión continua, ajustando los controladores basados en el rendimiento observado y utilizando la optimización bayesiana para realizar mejoras iterativas que aseguran el cumplimiento de los objetivo.

3. DESARROLLO DE LAS HERRAMIENTAS

Para cumplir con los objetivos del proyecto, se decidió crear dos herramientas. La primera en ser desarrollada fue OptBayes, una herramienta diseñada para llevar a cabo la optimización bayesiana en cualquier proceso industrial. En OptBayes se permite al usuario ajustar ciertos parámetros iniciales, adaptando la herramienta a las necesidades específicas de su proceso. Una vez que OptBayes fue desarrollada y validada, se procedió a crear SafeOptBayes, una herramienta cuyo objetivo es realizar la optimización bayesiana de un proceso industrial, pero incorporando restricciones de seguridad que aseguren el funcionamiento seguro del sistema durante todo el proceso de optimización.

Debido a la inclusión de estas nuevas funcionalidades en SafeOptBayes, su funcionamiento varía respecto a OptBayes, lo que hizo necesario llevar a cabo una validación adicional. Esta validación fue esencial para asegurar que, a pesar de las modificaciones y nuevas características, SafeOptBayes operara de manera correcta y segura, cumpliendo con los requisitos establecidos para el proyecto.

Ambas herramientas fueron codificadas en Matlab, y para su validación y ajuste se hizo uso de Simulink, una herramienta integrada en el entorno de Matlab.

En los siguientes apartados se explicará el desarrollo y funcionamiento de las herramientas. Para evitar que el documento principal se vuelva demasiado extenso y pesado, se abordarán en detalle solo los aspectos más relevantes. La información adicional y los detalles técnicos más específicos se incluirán en el Anexo I.

3.1. OPTBAYES

La función *OptBayes* comienza generando una serie de muestras iniciales de forma aleatoria dentro del espacio de búsqueda definido por las variables de entrada. El número de muestras iniciales es determinado por el usuario a través del parámetro *MuestrasIniciales*. Estas muestras iniciales se evalúan utilizando la función objetivo, y los resultados obtenidos se almacenan para su uso posterior en la construcción del modelo probabilístico. Esta fase es importante, ya que estas muestras proporcionan la base sobre la cual se construirá posteriormente el modelo probabilístico.

La función de evaluación que se aplica en cada muestra se especifica por el usuario y se representa como un *function_handle*. Este handle apunta a la función objetivo, que devuelve un valor basado en las variables de entrada. Este valor es el que se trata de optimizar mediante el proceso bayesiano.

3.1.1 CONSTRUCCIÓN DEL MODELO PROBABILÍSTICO

Como se mencionó anteriormente, un elemento esencial en la optimización bayesiana es el desarrollo de un modelo probabilístico que no solo aproxime con precisión el valor de la función objetivo en los puntos ya evaluados, sino que también permita estimar de manera confiable el valor en los puntos no evaluados dentro del espacio de búsqueda. En este proyecto, se ha optado por utilizar un proceso gaussiano para construir dicho modelo probabilístico. Un proceso gaussiano es una herramienta matemática que permite modelar funciones complejas, utilizando las observaciones previas y suponiendo que estas siguen una distribución normal multivariante. Esta capacidad de estimar tanto el valor esperado como la incertidumbre en las predicciones para cada punto dentro del

espacio de búsqueda resulta muy útil a la hora de guiar la exploración hacia nuevas regiones durante la optimización.

Con las muestras iniciales obtenidas, se procede a construir el modelo probabilístico utilizando el Proceso Gaussiano (GP). Este modelo se define principalmente a través de un conjunto de hiperparámetros que incluyen la longitud de escala, la varianza (sigmaF) y el ruido (sigmaN). Estos hiperparámetros son muy importantes, ya que determinan cómo de suave es la función predicha y cuánta confianza se tiene en los valores observados, influyendo así en el comportamiento del GP.

Es importante destacar que el proceso gaussiano inicial se construye utilizando unos valores de hiperparámetros que son proporcionados por el usuario o establecidos de manera predeterminada. Estos valores iniciales no dependen de las muestras obtenidas, sino que sirven como un punto de partida para el modelo. A medida que se acumulan más datos, se vuelve necesario ajustar estos hiperparámetros para reflejar mejor el comportamiento real de la función objetivo en el espacio de búsqueda.

Actualización de los Hiperparámetros

Para el ajuste de los hiperparámetros que definen el proceso gaussiano, se utiliza la función *hiperparam_actualization*. Esta función recurre a *fmincon*, una herramienta de optimización de MATLAB, para encontrar la mejor combinación de estos parámetros, minimizando la función *nlml* (Negative Log Marginal Likelihood). La función *nlml* se encarga de evaluar qué tan bien se ajusta el modelo a los datos disponibles, dado un conjunto específico de hiperparámetros.

Esta es esencial en este proceso porque mide la calidad del modelo probabilístico. A continuación, se muestra el código de la función *nlml*:

```
function nll = nlml(logtheta, X, Y)
    % Parámetros de regularización para prevenir valores Inf
    epsilon = 1e-6;

    % Hiperparámetros
    lengthScale = logtheta(1);
    sigmaF = logtheta(2);
    sigmaN = logtheta(3);

    % Matriz de covarianza con regularización (jitter)
    K = matern52_kernel(X, X, lengthScale, sigmaF) + (sigmaN^2 + epsilon) * eye(size(X, 1));

    % Comprobar si K es definida positiva
    [L, p] = chol(K, 'lower');
    if p > 0
        % Si no es definida positiva, devolver un número grande
        nll = Inf;
        return;
    end

    alpha = L' \ (L \ Y);

    % Calcular la probabilidad marginal negativa logarítmica
    nll = 0.5 * Y' * alpha + sum(log(diag(L))) + 0.5 * size(X, 1) * log(2*pi);

    % Añadir un término de regularización para evitar valores Inf
    if isnan(nll) || isinf(nll)
        nll = Inf;
    end
end
```

Tabla 1. Función Negative Log Marginal Likelihood (nlml)

En este código, se construye la matriz de covarianza usando el Matern kernel 5/2. El Matern kernel es una función matemática que evalúa la similitud entre diferentes puntos del espacio de entrada. Es

clave en un proceso gaussiano porque define cómo de conectados están los valores entre sí en distintos puntos. Este kernel en particular ofrece un buen balance entre suavidad y la capacidad de capturar variaciones complejas en los datos, lo que lo convierte en una buena elección a la hora de modelar funciones complejas. Si se desea consultar el código completo de la función *matern52_kernel*, se encuentra en el Anexo I.

La descomposición de Cholesky se utiliza para descomponer esta matriz de covarianza, lo que permite calcular de forma estable el logaritmo de la verosimilitud. Además, se añade un término de regularización para manejar situaciones donde la verosimilitud podría generar valores infinitos o indefinidos, asegurando así que el modelo permanezca robusto y útil a lo largo del proceso de optimización.

Una vez que se han reunido suficientes datos, se comienza a ajustar estos hiperparámetros utilizando la función *fmincon* de MATLAB. Este ajuste se realiza de manera periódica a lo largo del proceso de optimización, lo que garantiza que el modelo GP se mantenga alineado con los datos más recientes y refleje de la mejor manera posible el comportamiento real de la función objetivo.

De manera paralela a la actualización de los hiperparámetros, también se realiza la actualización de la media de la función objetivo. Al inicio, esta media se fija en cero a través de un proceso de escalamiento, pero es necesario recalcularla periódicamente para que refleje con mayor precisión las condiciones actuales del proceso. Este ajuste continuo asegura que el modelo GP funcione de manera óptima durante todo el proceso de optimización.

La siguiente imagen muestra el bloque de código donde se realizan estas actualizaciones periódicas tanto de los hiperparámetros como de la media de la función objetivo, resaltando la importancia de ambos elementos en el proceso de optimización bayesiana:

```
% Ajustar el modelo de regresión gaussiana
if iter == (muestrasIni+1)
     ini_hiper = 1;
    gpModel = hiperparam_actualization(X, Y, ini_hiper, initialHiperparametros(1), ...
        initialHiperparametros(2), initialHiperparametros(3));
     ini_hiper = 0;
elseif (mod((iter-(muestrasIni+1)-hiper(1)), hiper(2)) == 0 ...
         && (iter-(muestrasIni+1)-hiper(1))/hiper(2) > 0)|| iter == (muestrasIni+1 + hiper(1))
     gpModel = hiperparam_actualization(X, Y, ini_hiper, initialHiperparametros(1), ...
         initialHiperparametros(2), initialHiperparametros(3));
    % Recalcular priormean
     priormean_value = recalculate_priormean(X, Y, gpModel.KernelInformation.KernelParameters(1) ...
        , gpModel.KernelInformation.KernelParameters(2));
     priormean = @(x) priormean_value * ones(size(x, 1), 1);  % Redefinir la función priormean
end
gpModel.X = X;
gpModel.Y = Y;
```

Tabla 2. Ajuste del modelo de regresión gaussiana (OptBayes)

Para la actualización de la media de la función objetivo, se utiliza la función *recalculate_priormean*. La clave de esta función es cómo gestiona las ponderaciones basadas en la covarianza entre los datos. Lo que hace es determinar cuánto "peso" debe tener cada punto del conjunto de datos en el cálculo de la media, en función de su relación con los otros puntos del conjunto. No todos los datos son igual de relevantes: algunos pueden estar en regiones del espacio de búsqueda más exploradas o en áreas donde la función objetivo cambia menos, y que, por lo tanto, su influencia en la media deba ser

ajustada. Si se desea acceder a la función *recalculate_priormean* completa esta estará incluida en el Anexo I.

```
        % Calcular las ponderaciones basadas en la covarianza
        weights = calculate_weights_based_on_covariance(X, lengthScale, sigmaF);

        % Calcular la media previa utilizando las ponderaciones de Kriging
        priormean_value = sum((K * alpha) .* weights);
```

Tabla 3. Función recalculate_priormean

Para llevar a cabo la actualización de la media de la función objetivo, la función clave es *calculate_weights_based_on_covariance*. Esta función es la encargada de calcular las ponderaciones que determinan cuánto debe influir cada punto del conjunto total en la estimación de la media.

```
function weights = calculate_weights_based_on_covariance(X, lengthScale, sigmaF)
    % Calcular la matriz de covarianza utilizando el kernel de Matérn 5/2
    K = matern52_kernel(X, X, lengthScale, sigmaF);

    % Calcular las sumas de cada fila de la matriz de covarianza
    row_sums = sum(K, 2);

    % Invertir las sumas para obtener las ponderaciones
    % Añadir un pequeño valor (1e-6) para evitar divisiones por cero
    weights = 1 ./ (row_sums + 1e-6);

    % Normalizar las ponderaciones para que sumen 1
    weights = weights / sum(weights);
end
```

Tabla 4. Función calculate_weights_based_on_covariance

La función *calculate_weights_based*_on_covariance asigna un peso a cada punto de datos, teniendo en cuenta cómo se relaciona con los demás puntos del conjunto. Para ello, utiliza la matriz de covarianza generada por el Matern kernel 5/2, que mide la similitud entre los puntos en el espacio de entrada.

Primero, calcula la matriz de covarianza, K, la cual muestra como de conectados están los puntos entre sí. Luego, se suma cada fila de la matriz, lo que indica el grado de conexión de cada punto con el resto. Esas sumas se invierten para calcular las ponderaciones: los puntos menos conectados obtienen un mayor peso, asegurando que su influencia refleje adecuadamente su relevancia en el modelo.

Por último, se normalizan estas ponderaciones para que sumen 1, lo que garantiza que la media calculada sea una combinación equilibrada de todos los puntos de datos. Este enfoque permite que incluso los puntos más "distantes" en el espacio de búsqueda tengan la influencia adecuada, mejorando así la precisión en el cálculo de la media de la función, lo que a su vez contribuye a que el modelo gaussiano se ajuste de manera más efectiva a los datos disponibles.

3.1.2. SELECCIÓN DEL PRÓXIMO PUNTO A EVALUAR

Para determinar cuál será el próximo punto a evaluar en el proceso de optimización bayesiana, después de haber construido el modelo gaussiano (GP) con los datos disponibles, se emplea la función *getNextSample*. Esta función es clave a la hora de guiar la exploración y explotación del espacio de búsqueda, dirigiendo el proceso hacia nuevas áreas o afinando aquellas ya exploradas, dependiendo de la estrategia de adquisición seleccionada.

Esta función se encarga de calcular cuál es el siguiente punto en el espacio de búsqueda que debería ser evaluado, utilizando las predicciones generadas por el modelo GP. La función toma como entrada el modelo GP actual, el tipo de función de adquisición a utilizar, y otros parámetros específicos como el número de puntos a evaluar o el nivel de exploración deseado.

El código comienza generando un conjunto de puntos candidatos (*X_new*) dentro del espacio de búsqueda, definidos por los rangos de las variables. Estos puntos se evalúan mediante el modelo GP para obtener las predicciones del valor objetivo (*Y_pred*) y su desviación estándar (*Y_pred_sd*).

```
% Definir un rango de valores de entrada para hacer predicciones
    X_new = zeros(num_points, numel(variables));
    for j = 1:numel(variables)
        range = variables{j}.Range;
        X_new(:, j) = unifrnd(range(1), range(2), num_points, 1);
    end

    % Obtener las predicciones del modelo en el nuevo conjunto de datos de entrada
    [Y_pred, Y_pred_sd] = predGP(gpModel, X_new);
```

Tabla 5. Función getNextSample: generación de puntos y evaluación

El siguiente paso es calcular el valor de la función de adquisición para cada punto candidato. Dependiendo del tipo de función de adquisición especificada por el usuario, se emplean diferentes estrategias:

```
% Calcular la función de adquisición según el tipo
Y_best = min(gpModel.Y);
switch type
    case 'EI'
        acquisitionValue = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
    case 'EI+'
        acquisitionValue = expected_improvement_plus(Y_best, Y_pred, Y_pred_sd ...
            , xi, explorationRatio, gpModel.sigmaN);
    case 'EIPS'
        acquisitionValue = expected_improvement_per_second(Y_best, Y_pred ...
            , Y_pred_sd, xi, evalTimes);
    case 'EIPS+'
        acquisitionValue = expected_improvement_per_second_plus(Y_best, Y_pred ...
            , Y_pred_sd, xi, evalTimes, explorationRatio, gpModel.sigmaN);
    case 'LCB'
        acquisitionValue = lower_confidence_bound(Y_pred, Y_pred_sd, xi);
    case 'PI'
        acquisitionValue = probability_of_improvement(Y_best, Y_pred, Y_pred_sd, xi);
    otherwise
        error('Tipo de función de adquisición no reconocido.');
end
```

Tabla 6. Función getNextSample implementación de diferentes funciones de adquisición

Una de las funciones de adquisición más utilizadas es Expected Improvement (EI). Esta función calcula la expectativa de mejora en comparación con el mejor valor observado hasta el momento, ajustando dicho valor mediante un factor xi, que regula el equilibrio entre exploración y explotación. El objetivo es seleccionar el punto que, en promedio, se espera que ofrezca la mayor mejora respecto al mínimo actual.

```
function EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi)
    Z = (Y_best*(1-xi) - Y_pred) ./ Y_pred_sd;
    EI = (Y_best*(1-xi) - Y_pred) .* normcdf(Z) + Y_pred_sd .* normpdf(Z);
end
```

Tabla 7. Función expected_improvement

En este código, Z indica cuántas desviaciones estándar hay entre el mejor valor conocido ajustado y el valor predicho para un punto candidato. La función normcdf se utiliza para calcular la probabilidad acumulada en una distribución normal, mientras que normpdf proporciona la densidad de probabilidad de esa misma distribución. Estas dos funciones se combinan para obtener el Expected Improvement (EI), que nos da una medida de cuánto podría mejorar el mejor resultado conocido al evaluar el nuevo punto.

Además del Expected Improvement clásico, también se ha desarrollado una variación denominada Expected Improvement Plus (EI+). Esta función incorpora un parámetro adicional, el exploration ratio, que ayuda a balancear mejor la explotación y la exploración durante la optimización.

```
function EI_plus = expected_improvement_plus(Y_best, Y_pred, Y_pred_sd, ...
    xi, explorationRatio, sigmaN)

    sigma_Q = sqrt(Y_pred_sd.^2 + sigmaN^2);
    overExploited = sigma_Q < explorationRatio * sigmaN;

    EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);

    % Incrementar EI en regiones sobreexplotadas
    EI_plus = EI;
    EI_plus(overExploited) = EI_plus(overExploited) * 10;
end
```

Tabla 8. Función expected_improvement_plus

El exploration ratio se utiliza para identificar áreas del espacio de búsqueda que pueden haber sido sobreexplotadas, es decir, regiones donde la incertidumbre (*sigma_Q*) es baja en relación con el ruido (sigmaN). Cuando se detectan tales áreas, el valor de EI se incrementa significativamente, incentivando al algoritmo a continuar explorando esas regiones a pesar de su bajo nivel de incertidumbre. De esta manera, se evita que el algoritmo descarte potenciales óptimos globales por haber explorado demasiado pronto ciertas áreas.

En el código de *EI_plus*, la función evalúa si *sigma_Q* (una medida de la incertidumbre combinada con el ruido) es menor que un umbral definido por la multiplicación del exploration ratio y *sigmaN*. Si es así, el valor de EI se multiplica por un factor determinado para incrementar la exploración en esa región. Esto permite al algoritmo ser más agresivo en la exploración de regiones que, de otro modo, podrían ser ignoradas, asegurando así una mejor cobertura del espacio de búsqueda.

Una vez calculados los valores de la función de adquisición para todos los puntos candidatos, la función *getNextSample* selecciona aquel punto que presenta el valor más alto. Y posteriormente para afinar aún más la búsqueda y acercarse al óptimo, se emplea la función *fmincon*. Esta herramienta realiza una optimización local alrededor del punto inicialmente seleccionado (*initialSample*), explorando el espacio de búsqueda en busca de un punto que ofrezca un mejor resultado dentro de los límites establecidos.

```
% Seleccionar el próximo punto a probar
[~, idx] = max(acquisitionValue);
initialSample = X_new(idx, :);

% Definir límites para fmincon
lb = cellfun(@(var) var.Range(1), variables);
ub = cellfun(@(var) var.Range(2), variables);

% Usar fmincon para buscar puntos cerca del óptimo estimado
options = optimoptions('fmincon', 'Display', 'off', 'MaxIterations', ...
```

```
        1000, 'MaxFunctionEvaluations', 3000, 'OptimalityTolerance', 1e-6);
    fun = @(X) optimization_function(X, Y_best, gpModel, type, xi, explorationRatio, evalTimes);
    [nextSample, ~] = fmincon(fun, initialSample, [], [], [], [], lb, ub, [], options);

    % Valor estimado en el próximo punto
    [estimatedValue, ~] = predGP(gpModel, nextSample);
```

Tabla 9. Función getNextSample: Selección y optimización del próximo punto de prueba en el proceso bayesiano

Finalmente, el punto seleccionado (*nextSample*) se devuelve para ser evaluado en la función objetivo. Este punto, determinado por el modelo GP, la función de adquisición y la optimización local con fmincon, es el que se considera que tiene el mayor potencial para mejorar el rendimiento del proceso o descubrir nuevas áreas prometedoras en el espacio de búsqueda.

Al combinar estas estrategias, se logra explorar el espacio de búsqueda de manera efectiva, ajustando al mismo tiempo los resultados de forma precisa a nivel local, lo que incrementa significativamente las posibilidades de encontrar el óptimo global real.

3.1.3. EVALUACIÓN Y ACTUALIZACIÓN

Una vez elegido el próximo punto a evaluar en el espacio de búsqueda, se procede a ejecutar la función objetivo en ese punto. Este paso es clave porque el valor que obtenemos nos proporciona nueva información sobre el comportamiento de la función objetivo en esa región. Esta información se añade a las muestras ya evaluadas, enriqueciendo el conjunto de datos con el que trabaja nuestro modelo de proceso gaussiano (GP).

```
    % Guardar el tiempo de evaluación y valor funcion objetivo
    tic;
    nuevoValor = funcion(table2struct(nuevaFila));
    evalTimes(iter) = toc;
```

Tabla 10. Función OptBayes registro del tiempo de evaluación

Cada vez que evaluamos la función objetivo en un nuevo punto, no solo obtenemos un valor que refleja cómo se comporta la función en ese lugar, sino que también registramos cuánto tiempo lleva realizar esa evaluación. Este detalle es especialmente importante porque algunas funciones de adquisición, como *Expected Improvement per Second*, consideran no solo el posible mejoramiento del valor objetivo, sino también el tiempo que se necesita para calcularlo. Así, si se desea se pueden priorizar los puntos en el espacio de búsqueda que, además de prometer una buena mejora, requieren menos tiempo de evaluación, optimizando así el uso de los recursos disponibles.

La incorporación de este nuevo punto permite que el modelo GP se ajuste de manera más precisa a la función objetivo, modelando mejor las características del problema. En función de política de actualización establecida (que puede ser después de un número determinado de iteraciones o en función de otros criterios definidos por el usuario), se ajustan los hiperparámetros del modelo GP. Este ajuste como se mencionó con anterioridad es clave para mantener la precisión del modelo a medida que se amplía el conjunto de datos. Por ejemplo, si se ha añadido un punto que revela una región del espacio de búsqueda con características inesperadas, el ajuste de los hiperparámetros permitirá que el GP se adapte y modele esta región de manera más precisa.

Este ciclo de evaluación y actualización es la parte más relevante de la optimización bayesiana: a medida que se obtienen más datos, el modelo se vuelve cada vez mejor, lo que hace que se mejore la selección de puntos en las siguientes iteraciones.

3.1.4. Representación Visual en el Proceso de Optimización

La visualización juega un papel fundamental en el proceso de optimización bayesiana, ya que nos permite seguir de cerca cómo va avanzando la optimización, entender mejor el comportamiento del modelo y analizar cómo las decisiones de exploración y explotación impactan en el resultado final. En este proyecto, se han creado gráficos específicos para comparar el mejor valor esperado con el valor mínimo obtenido y para observar cómo evolucionan la función objetivo y las variables a lo largo del proceso. Para la representación de dichos gráficos se ha implementado la función updateVisualization, la se encuentra en el Anexo I si se desea consultar.

Comparación entre el Mejor Valor Esperado y el Valor Mínimo Obtenido

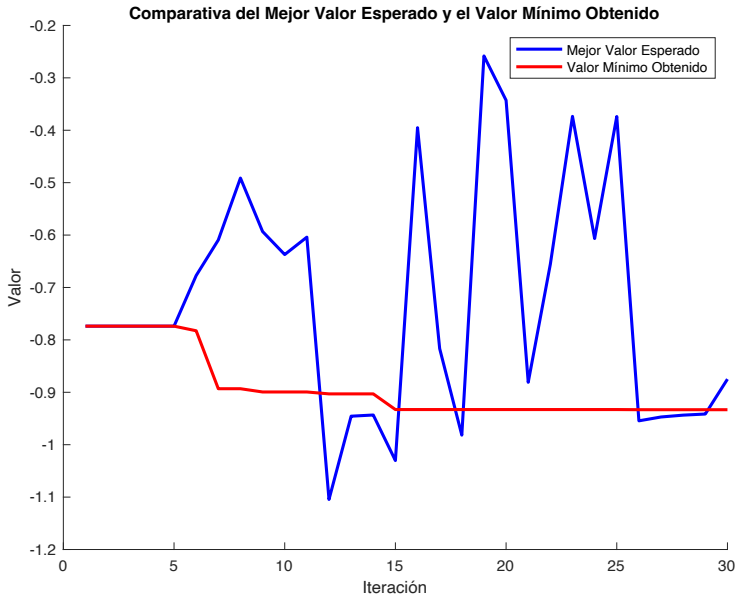


Figura 4. Evolución del mejor valor esperado y el valor mínimo obtenido durante el proceso de optimización (OptBayes)

Uno de los gráficos más relevantes que se han generado compara el mejor valor esperado, tal como lo predice el modelo gaussiano, con el valor mínimo que se ha obtenido realmente durante la optimización. Este gráfico es clave para evaluar si el modelo está haciendo un buen trabajo al identificar los puntos más prometedores y si está logrando mejorar los resultados con cada iteración.

- Mejor Valor Esperado (línea azul)

Esta línea muestra lo que el modelo predice como el mejor resultado posible en cada iteración, basándose en las observaciones anteriores. El "mejor valor esperado" corresponde al valor de la media $\mu(x)$ estimada por el modelo gaussiano en el punto más prometedor, de acuerdo con la función de adquisición seleccionada.

$$Mejor\ valor\ esperado = \mu(x_{\text{óptimo estimado}})$$

- Valor Mínimo Obtenido (línea roja)

Esta línea muestra el mejor resultado que se ha logrado realmente tras evaluar los puntos sugeridos por el modelo. Así, se puede ver si el modelo está logrando mejoras reales y cómo se ajusta a los datos obtenidos.

Evolución de la Función Objetivo y las Variables

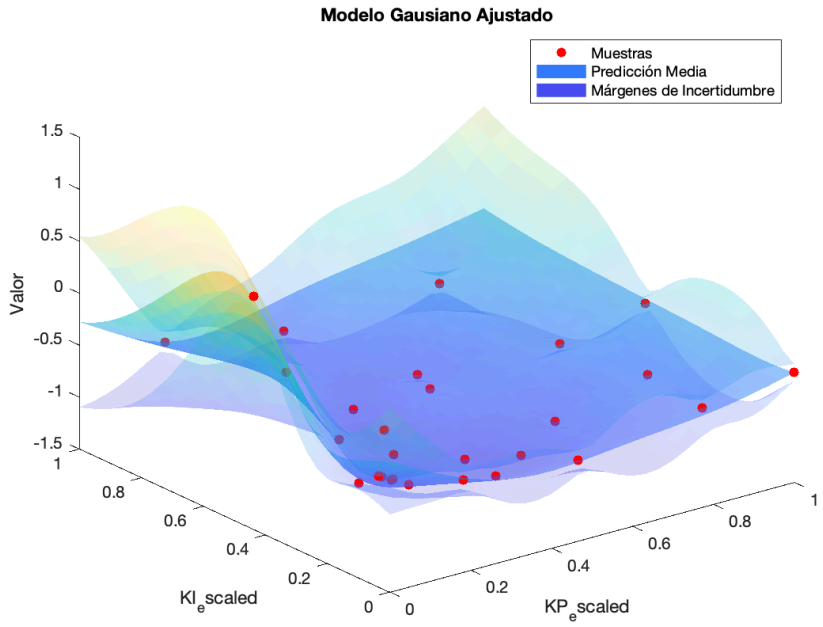


Figura 5. Modelo Gaussiano Ajustado que representa la predicción de la función objetivo con márgenes de incertidumbre (OptBayes)

A medida que avanza el proceso de optimización, se generan gráficos que muestran cómo cambian los valores de la función objetivo y de las variables de entrada en cada iteración. Estos gráficos son útiles para ver cómo se ajustan las variables para mejorar el resultado.

- Gráficos en 2D para un Solo Parámetro

Cuando se optimiza un solo parámetro, se usan gráficos de líneas que muestran cómo varía ese parámetro y su impacto en la función objetivo.

- Superficies en 3D para Dos Parámetros

Si se están optimizando dos parámetros, se crean gráficos en 3D que muestran la relación entre las variables y la función objetivo en el espacio de búsqueda.

Para tres parámetros o mas no se llevará a cabo esta visualización al no ser posible llevar a cabo representaciones en más de tres dimensiones.

Tabla Iterativa de Resultados

Iteracion	KP_escaled	KI_escaled	Objetivo	ValorDesescalado	lengthScale	sigmaF	sigmaN	Mejor
1	0.4953	0.5147	−0.7738	6.9048	NaN	NaN	NaN	Sí
2	0.7842	0.4734	−0.5924	7.6303	NaN	NaN	NaN	No
3	0.5147	0.5812	−0.7486	7.0054	NaN	NaN	NaN	No
4	0.4793	0.0181	−0.7620	6.9520	NaN	NaN	NaN	No
5	0.9889	0.4664	−0.4066	8.3735	NaN	NaN	NaN	No
6	0.5810	0.2259	−0.7826	6.8696	0.2500	1.0000	0.0100	Sí
7	0.3521	0.2172	−0.8932	6.4272	0.2500	1.0000	0.0100	Sí
8	0.2192	0.4488	−0.8814	6.4742	0.2500	1.0000	0.0100	No
9	0.1095	0.2409	−0.8994	6.4025	0.2500	1.0000	0.0100	Sí
10	0.2001	0.0240	−0.6689	7.3243	0.2500	1.0000	0.0100	No
11	0.0000	0.4392	0.7379	12.9518	0.2500	1.0000	0.0100	No
12	0.2589	0.3256	−0.9030	6.3878	0.2500	1.0000	0.0100	Sí
13	0.3252	0.5410	−0.8333	6.6669	0.2500	1.0000	0.0100	No
14	0.3151	0.4288	−0.8646	6.5416	0.2500	1.0000	0.0100	No
15	0.1939	0.1917	−0.9329	6.2683	0.2500	1.0000	0.0100	Sí
16	0.8018	0.0379	−0.6280	7.4879	0.2757	0.6951	0.0100	No
17	0.0699	0.1261	−0.6303	7.4787	0.2757	0.6951	0.0100	No
18	0.1755	0.2542	−0.9302	6.2790	0.2757	0.6951	0.0100	No
19	0.7725	0.8468	−0.5124	7.9502	0.2757	0.6951	0.0100	No
20	0.4346	0.9085	−0.7003	7.1989	0.2757	0.6951	0.0100	No
21	0.3451	0.1088	−0.8944	6.4224	0.2757	0.6951	0.0100	No
22	0.3251	0.7582	−0.7726	6.9094	0.2757	0.6951	0.0100	No
23	0.2117	1.0000	−0.6941	7.2235	0.2757	0.6951	0.0100	No
24	0.8219	0.2413	−0.6082	7.5671	0.2757	0.6951	0.0100	No
25	1.0000	0.0000	−0.4446	8.2218	0.2757	0.6951	0.0100	No
26	0.1868	0.2333	−0.9332	6.2672	0.3096	0.6264	0.0412	Sí
27	0.1825	0.2318	−0.9332	6.2671	0.3096	0.6264	0.0412	Sí
28	0.1814	0.2317	−0.9332	6.2672	0.3096	0.6264	0.0412	No
29	0.1810	0.2318	−0.9332	6.2673	0.3096	0.6264	0.0412	No
30	0.4426	0.1535	−0.8616	6.5537	0.3096	0.6264	0.0412	No

Figura 6. Resultados de las iteraciones del proceso de optimización bayesiana

Durante el proceso de optimización, se genera una tabla iterativa que recoge, en cada iteración, el valor de la función objetivo, los valores de las variables, y otros parámetros importantes como la longitud de escala (*lengthScale*), *sigmaF*, *sigmaN*, y si el punto es el mejor encontrado hasta el momento.

3.1.5. Llamada de la Función y Personalización por parte del Usuario

La función OptBayes permite al usuario realizar un proceso de optimización bayesiana personalizado, ofreciendo la posibilidad de ajustar una serie de parámetros clave. Algunos de estos parámetros son obligatorios y deben ser proporcionados por el usuario, mientras que otros son opcionales y, de no ser especificados, tomarán valores predeterminados.

Parámetros Obligatorios

1. Función Objetivo (fun)

   El primer y más importante argumento que el usuario debe proporcionar es la función objetivo que se desea optimizar. Esta función se pasa como un *function_handle* y debe aceptar las variables de entrada especificadas, devolviendo un valor que representa la calidad o el costo en ese punto particular del espacio de búsqueda. Sin una función objetivo bien definida, la optimización no puede llevarse a cabo.

   Para mejorar la eficacia del proceso de optimización, se recomienda escalar la función objetivo de tal manera que su media sea cercana a 0. Este escalado ayuda a que el modelo probabilístico (en este caso, un proceso gaussiano) se ajuste de manera más precisa a los datos observados, optimizando así los resultados obtenidos durante el proceso de optimización.

2. Variables de Entrada (variables)

El usuario también debe especificar las variables de entrada que se van a optimizar. Estas variables se definen dentro de un espacio de búsqueda específico, y el usuario debe controlar tanto el rango como el tipo de cada variable. Las variables de entrada son fundamentales para definir el dominio sobre el cual el proceso de optimización se llevará a cabo.

Parámetros Opcionales

Además de los parámetros obligatorios, el usuario puede ajustar varios parámetros opcionales para personalizar el proceso de optimización. Si estos no se especifican, OptBayes utilizará valores predeterminados:

1. Número de Evaluaciones de la Función Objetivo (ObjetivosEval)

   Define el número máximo de veces que la función objetivo será evaluada durante el proceso de optimización. Por defecto, este valor es 100, pero el usuario puede ajustarlo según la profundidad y duración deseada para la optimización.

2. Muestras Iniciales (MuestrasIniciales/MuestraInicialManual)

   Este parámetro define cómo se generan las muestras iniciales antes de comenzar la optimización bayesiana. Existen dos formas principales para definir estas muestras:

   - Generación aleatoria (MuestrasIniciales)

     Por defecto, el programa generará de forma aleatoria un conjunto de puntos iniciales dentro del rango especificado para cada variable. El número de puntos puede ajustarse con el parámetro MuestrasIniciales, que por defecto genera 5 muestras, aunque el usuario puede definir la cantidad deseada.

   - Provisión manual (MuestraInicialManual)

     Alternativamente, el usuario puede proporcionar una matriz con muestras iniciales específicas, donde cada fila representará un conjunto de valores de las variables. En este caso, se omite la generación aleatoria, y el número de muestras iniciales vendrá determinado por el número de filas de la matriz dada.

3. Método de Optimización (MetodoOptimizacion)

   El usuario puede seleccionar entre diferentes métodos de optimización, como Expected Improvement (EI) o variantes como EI+. El método predeterminado es EI. Este parámetro controla la función de adquisición utilizada para seleccionar el siguiente punto a evaluar.

4. Actualización de Hiperparámetros (ActualizacionHiperparametros)

   Este parámetro permite definir cuándo y con qué frecuencia se actualizan los hiperparámetros del modelo GP. Si no se especifica, los valores predeterminados controlarán esta actualización.

5. Exploration Ratio (ExplorationRatio)

   Cuando se utiliza el método EI+, este parámetro permite ajustar la cantidad de exploración frente a la explotación. El valor predeterminado es 0.1, pero puede ser ajustado por el usuario para modificar el comportamiento del algoritmo.

6. Xi (Xi)

   Utilizado en la función de Expected Improvement, este parámetro ajusta el equilibrio entre exploración y explotación. El valor predeterminado es 0.01, pero el usuario puede modificarlo para influir en la agresividad del algoritmo.

7. Valores Iniciales de los Hiperparámetros (InitialHiperparametros)

   El usuario tiene la opción de proporcionar valores iniciales para los hiperparámetros del proceso gaussiano, como la longitud de escala, sigmaF y sigmaN. Si no se especifican, se utilizarán valores base predeterminados para iniciar el modelo.

Ejemplo de Llamada

Un ejemplo de cómo el usuario podría llamar a la función OptBayes podría verse de la siguiente manera:

```
% Definir el espacio de búsqueda de parámetros
KP_escaled = optimizableVariable('KP_escaled',[0,1],'Type','real');
KI_escaled = optimizableVariable('KI_escaled',[0,1],'Type','real');
variables = {KP_escaled, KI_escaled};


% Definir la función objetivo
fun = @(vars) sim_PI(vars.KP_escaled, vars.KI_escaled);
resultados = optbayes(fun, variables, 'ObjetivosEval', 30, 'MuestrasIniciales', 5,...
                        'MetodoOptimizacion', 'EI+', 'ActualizacionHiperparametros',[5, 10], ...
                        'Xi',0.01,'ExplorationRatio',0.2,'InitialHiperparametros',[0.25, 1, 0.01]);
```

Figura 7. Ejemplo de llamada de la función OptBayes

En este ejemplo, el usuario ha configurado una optimización bayesiana que se detendrá después de 30 evaluaciones de la función objetivo, comenzará con 5 muestras iniciales, y utilizará la variante EI+ para la selección del próximo punto. Además, el usuario ha especificado que los hiperparámetros del modelo gaussiano se actualicen de manera específica: la primera actualización se realizará tras 5 iteraciones desde las muestras iniciales, y posteriormente se actualizarán periódicamente cada 10 iteraciones respecto la última actualización. Esto se define a través del parámetro ActualizacionHiperparametros como [5, 10]. De esta manera, se asegura que el modelo gaussiano se ajuste adecuadamente a medida que se obtiene más información durante el proceso de optimización.

El usuario también ha ajustado otros parámetros como el Xi para controlar el equilibrio entre exploración y explotación, y el ExplorationRatio para determinar cuánto se debe explorar el espacio de búsqueda. Además, se han proporcionado valores iniciales para los hiperparámetros del modelo a través de InitialHiperparametros.

Como se ha comentado anteriormente cualquier parámetro que no se especifique en la llamada a la función tomará los valores predeterminados establecidos por la función OptBayes. Esto permite un ajuste fino por parte del usuario cuando sea necesario, mientras que el resto del proceso se maneja automáticamente.

3.2. SAFEOPTBAYES

SafeOptBayes es una extensión del enfoque tradicional de OptBayes, especialmente diseñada para situaciones en las que es crucial que las soluciones no solo busquen el mejor resultado, sino que también cumplan con ciertas restricciones de seguridad. Aunque el proceso básico sigue siendo el mismo que en OptBayes, se han realizado algunas modificaciones clave para garantizar que cada solución propuesta sea segura.

Una de las diferencias principales es la incorporación de un segundo modelo probabilístico, que se encarga de manejar las restricciones de seguridad. Esto implica que tanto los hiperparámetros como la función de adquisición se ajustan teniendo en cuenta estos criterios adicionales. A pesar de estas diferencias, tanto la visualización de los resultados como el proceso de evaluación y actualización se llevan a cabo de la misma manera que en OptBayes. Esto significa que, una vez que se integran los puntos seguros y optimizados, el proceso de análisis y ajuste del modelo sigue siendo el mismo que se ha descrito en OptBayes.

3.2.1. ACTUALIZACIÓN DE LOS HIPERPARÁMETROS Y LA MEDIA

Una de las diferencias más importantes en SafeOptBayes es que, dado que trabajamos con dos funciones diferentes (la función objetivo y la función de restricción), es necesario mantener y actualizar dos modelos de procesos gaussianos (GP) por separado. Cada uno de estos modelos tiene sus propios hiperparámetros, como la longitud de escala, sigmaF, y sigmaN, que deben ser ajustados de forma independiente.

Esto significa que en SafeOptBayes, durante las iteraciones clave, se realiza una doble actualización de los hiperparámetros y de la media de las predicciones, una para cada modelo GP. Esta actualización dual es clave para asegurar que ambos modelos reflejen de manera precisa las características tanto del objetivo a optimizar como de las restricciones de seguridad que deben cumplirse.

```
    % Ajustar el modelo de regresión gausiana
    if iter == (muestrasIni+1)
        ini_hiper = 1;
        gpModel = hiperparam_actualization(X, Y, ini_hiper, initialHiperparametros(1) ...
            , initialHiperparametros(2), initialHiperparametros(3));
        gpModelRes = hiperparam_actualization(X, YRes, ini_hiper, initialSafeHiperparametros(1) ...
            , initialSafeHiperparametros(2), initialSafeHiperparametros(3));
        ini_hiper = 0;
    elseif (mod((iter-(muestrasIni+1)-hiper(1)), hiper(2)) == 0 && ...
             (iter-(muestrasIni+1)-hiper(1))/hiper(2) > 0)|| iter == (muestrasIni+1 + hiper(1))
        gpModel = hiperparam_actualization(X, Y, ini_hiper, initialHiperparametros(1), ...
             initialHiperparametros(2), initialHiperparametros(3));
        gpModelRes = hiperparam_actualization(X, YRes, ini_hiper, initialSafeHiperparametros(1), ...
            initialSafeHiperparametros(2), initialSafeHiperparametros(3));
        % Recalcular priormean para objetivo
        priormean_value = recalculate_priormean(X, Y, gpModel.KernelInformation.KernelParameters(1), ...
            gpModel.KernelInformation.KernelParameters(2));
        priormean = @(x) priormean_value * ones(size(x, 1), 1);  % Redefinir la función priormean
        % Recalcular priormean para restricción
        priormean_valueRes              =              recalculate_priormean(X,              YRes,
gpModelRes.KernelInformation.KernelParameters(1), ...
            gpModelRes.KernelInformation.KernelParameters(2));
        priormeanRes = @(x) priormean_valueRes * ones(size(x, 1), 1);  % Redefinir la función priormean
    end
    gpModel.X = X;
    gpModel.Y = Y;
    gpModelRes.X = X;
    gpModelRes.Y = YRes;
```

Tabla 11. Ajuste del modelo de regresión gaussiana (SafeOptBayes)

Este código muestra cómo se maneja la actualización dual tanto para los hiperparámetros como para la media. Aquí, *gpModel* se refiere al modelo GP para la función objetivo, mientras que *gpModelRes* se refiere al modelo GP para la restricción de seguridad. Cada modelo se actualiza de forma independiente para reflejar de la mejor manera posible los datos más recientes obtenidos en la optimización.

3.2.2. SELECCIÓN DEL PRÓXIMO PUNTO A EVALUAR

Otra diferencia clave en SafeOptBayes es cómo se selecciona el próximo punto a evaluar en el espacio de búsqueda. En OptBayes, la selección se basa únicamente en la función objetivo. Sin embargo, en SafeOptBayes, se debe considerar tanto la función objetivo como la restricción de seguridad. Esto significa que cualquier nuevo punto sugerido no solo debe prometer mejorar el objetivo, sino que también debe cumplir con la restricción de seguridad.

La función que maneja esto en SafeOptBayes es más compleja, ya que integra información de ambos modelos GP. Se calcula un valor de adquisición que no solo maximiza la mejora esperada en la función objetivo, sino que también asegura que el punto cumpla con la restricción de seguridad. De esta forma, el modelo prioriza puntos en el espacio de búsqueda que son seguros y, al mismo tiempo, ofrecen una alta probabilidad de mejorar el objetivo.

Aquí la función llamada para determinar el próximo punto a evaluar es *getNextSafeSample*, una versión de la función *getNextSample* que tiene en cuenta la restricción de seguridad. Esta toma en cuenta ambos modelos GP (el de la función objetivo *gpModel* y el de la restricción *gpModelRes*). El objetivo de esta función es garantizar que el próximo punto a evaluar no solo maximice la función objetivo, sino que también cumpla con las restricciones de seguridad definidas.

Cálculo del Safe Vector

Una de las primeras diferencias entre ambas funciones es el cálculo del *safe_vector*. Este vector se utiliza para evaluar si los puntos candidatos cumplen con la restricción de seguridad antes de proceder con la evaluación de su valor de adquisición.

```
% Obtener las predicciones del modelo en el nuevo conjunto de datos de entrada
[Y_pred, Y_pred_sd] = predGP(gpModel, X_new);
[YRes_pred, YRes_pred_sd] = predGP(gpModelRes, X_new);
safe_vector = YRes_pred + 2 * YRes_pred_sd - safeValue;
```

Tabla 12. Función getNextSafeSample: Predicción del modelo gaussiano y cálculo del vector de seguridad

En este fragmento de código, se calcula el *safe_vector* utilizando las predicciones del modelo GP para la restricción (*gpModelRes*). Se suman dos veces la desviación estándar de las predicciones a los valores predichos (*YRes_pred* + 2 * *YRes_pred_sd*), y luego se resta el valor de seguridad (*safeValue*). Esto ayuda a identificar los puntos que no cumplen con la restricción, permitiendo que sean descartados de la optimización.

Exclusión de Puntos No Seguros

Una vez calculado el *safe_vector*, la función getNextSafeSample procede a excluir aquellos puntos que no cumplen con la restricción de seguridad. Esto se logra ajustando el valor de adquisición de estos puntos a menos infinito, asegurando que no serán seleccionados en la optimización.

```
    if strcmp(safeMethod, 'Upper')
            for i = 1:num_points
                if safe_vector(i) >= 0
                    acquisitionValue(i) = -inf;
                end
            end
        elseif strcmp(safeMethod, 'Lower')
            for i = 1:num_points
                if safe_vector(i) <= 0
                    acquisitionValue(i) = -inf;
                end
            end
        end
```

Tabla 13. Función getNextSafeSample: Aplicación de método de seguridad para selección de puntos en el proceso de optimización

Este fragmento de código muestra cómo se manejan los puntos que no cumplen con la restricción de seguridad. Dependiendo del método de seguridad (*Upper* o *Lower*), se determina si un punto debe ser considerado seguro o no. El usuario debe especificar si la restricción de seguridad corresponde a un límite superior (*Upper*), lo que significa que los valores predichos deben estar por debajo de ese límite, o a un límite inferior (*Lower*), donde los valores predichos deben estar por encima. Si un punto no cumple con estas condiciones de seguridad, se excluye de la selección estableciendo su valor de adquisición como inválido, asegurando así que solo los puntos que respetan la restricción sean considerados en la optimización.

Inclusión de las Restricciones de Seguridad en fmincon

Finalmente, una diferencia importante al utilizar *getNextSafeSample* es la inclusión de restricciones de seguridad dentro del proceso de optimización local que realiza fmincon. Esto garantiza que cualquier optimización adicional alrededor del punto inicialmente seleccionado también respete las restricciones de seguridad.

```
    % Definir las restricciones de seguridad para fmincon
    nonlcon = @(X) safety_constraints(X, gpModelRes, safeValue, safeMethod);

    % Crear los límites inferiores y superiores
    lb = cellfun(@(var) var.Range(1), variables);
    ub = cellfun(@(var) var.Range(2), variables);

    % Usar fmincon para buscar puntos cerca del óptimo estimado
    options = optimoptions('fmincon', 'Display', 'off', 'MaxIterations', 1000, ...
        'MaxFunctionEvaluations', 3000, 'OptimalityTolerance', 1e-6);
     [nextSample, ~] = fmincon(fun, initialSample, [], [], [], [], lb, ub, nonlcon, options);
```

Tabla 14. Función getNextSafeSample: Inclusión de restricciones de seguridad en el proceso de optimización mediante fmincon

En este código, se define una función de restricciones (*nonlcon*) que se encarga de verificar que el punto optimizado por fmincon cumpla con las condiciones de seguridad impuestas por el modelo GP de la restricción (*gpModelRes*).

3.2.3. Llamada de la Función y Personalización por parte del Usuario

Al utilizar SafeOptBayes en lugar de OptBayes, el usuario debe proporcionar información adicional que asegure el cumplimiento de las restricciones de seguridad, además de la optimización de la función objetivo.

Parámetros Obligatorios en SafeOptBayes:

1. Función Objetivo y Restricción Combinadas

   A diferencia de OptBayes, en SafeOptBayes se requiere una única función que devuelva dos salidas: la primera corresponde al valor de la función objetivo y la segunda a la función de restricción. Esta función combinada (fun) debe aceptar las variables de entrada especificadas y devolver un valor que representa la calidad del punto en el espacio de búsqueda (función objetivo) y otro valor que indique si ese punto cumple con las restricciones de seguridad.

2. Variables de Entrada

   Similar a OptBayes, el usuario debe definir las variables de entrada que se van a optimizar. Estas variables se especifican dentro de un espacio de búsqueda particular y son esenciales para establecer el dominio sobre el cual se realizará la optimización.

Parámetros Opcionales Específicos de SafeOptBayes:

Los parámetros opcionales en SafeOptBayes incluyen todos los que ya existen en OptBayes, además de algunos específicos para la gestión de restricciones de seguridad:

1. Método de Seguridad (SafeMethod)

   El usuario puede especificar si la restricción de seguridad es un límite superior ('Upper') o inferior ('Lower'). Aunque este parámetro no es obligatorio, permite al usuario definir cómo se interpretan los resultados de la función de restricción y asegura que solo los puntos que cumplan con la restricción sean considerados como seguros. Si no se especifica, se tomará un valor predeterminado.

2. Valores Iniciales para los Hiperparámetros de Seguridad (InitialSafeHiperparametros)

   Similar a los valores iniciales para los hiperparámetros del modelo objetivo, pero aplicados al modelo de la restricción de seguridad. Estos valores determinan la suavidad y confianza del modelo gaussiano asociado con la restricción. Si no se especifican, se utilizarán valores base predeterminados.

3. Valor de Seguridad (SafeValue)

   Este parámetro opcional define el umbral de seguridad que debe respetarse en el proceso de optimización. Es el valor con el que se evalúa si un punto cumple o no con la restricción. Si no se especifica, se utilizará un valor predeterminado.

Ejemplo de Llamada a SafeOptBayes:

```
% Definir el espacio de búsqueda de parámetros
KP_escaled = optimizableVariable('KP_escaled',[0,1],'Type','real');
KI_escaled = optimizableVariable('KI_escaled',[0,1],'Type','real');
variables = {KP_escaled, KI_escaled};
```

```
% Definir la función objetivo
fun = @(vars) Safe_sim_PI(vars.KP_escaled, vars.KI_escaled);


resultados = safeoptbayes(fun, variables, 'ObjetivosEval', 50, 'MuestrasIniciales', 5,...
                          'MetodoOptimizacion', 'EI+', 'ActualizacionHiperparametros',[10, 10], ...
                          'Xi',0.01,'ExplorationRatio',0.2,'SafeValue',0.5,'MetodoSafe','Upper', ...
                          'InitialSafeHiperparametros',[0.25, 1, 0.1], ...
                          'InitialHiperparametros',[0.25, 1, 0.1]);
```

Tabla 15. Ejemplo de llamada de la función SafeOptBayes

En el ejemplo de llamada a SafeOptBayes, se han añadido parámetros específicos que permiten manejar las restricciones de seguridad, algo que no está presente en OptBayes. En este caso, se ha establecido *SafeValue* en 0.5, indicando el umbral de seguridad que debe respetarse. El *MetodoSafe* se ha configurado como 'Upper', lo que implica que el valor límite de seguridad actúa como un límite superior. Además, se han definido los valores iniciales para los hiperparámetros del modelo de seguridad mediante *InitialSafeHiperparametros*, con [0.25, 1, 0.1].

3.3. VALIDACIÓN

Para validar el correcto funcionamiento de las funciones de optimización implementadas, OptBayes y SafeOptBayes, se ha llevado a cabo una serie de pruebas utilizando un proceso de control simple. Este proceso está definido por la función de transferencia $\frac{3}{s+1}$ que representa un sistema de primer orden comúnmente utilizado en la ingeniería de control.

El objetivo de estas pruebas es optimizar los parámetros del controlador PI que regula este sistema, con el fin de minimizar el error cuadrático entre la salida del sistema y una referencia deseada. En este proceso, no solo se considera el error cuadrático, sino también el esfuerzo de control, que se penaliza aplicando un factor multiplicativo de 1/10 a la acción de control. Así, el algoritmo no solo busca minimizar el error, sino que también busca la opción que para minimizar el error en el seguimiento de la referencia necesite un aporte de energía menor.

El modelo de Simulink correspondiente se muestra a continuación. Si se desea ver a mayor tamaño o con más detalle, está incluido en el Anexo V adjunto al final del documento.

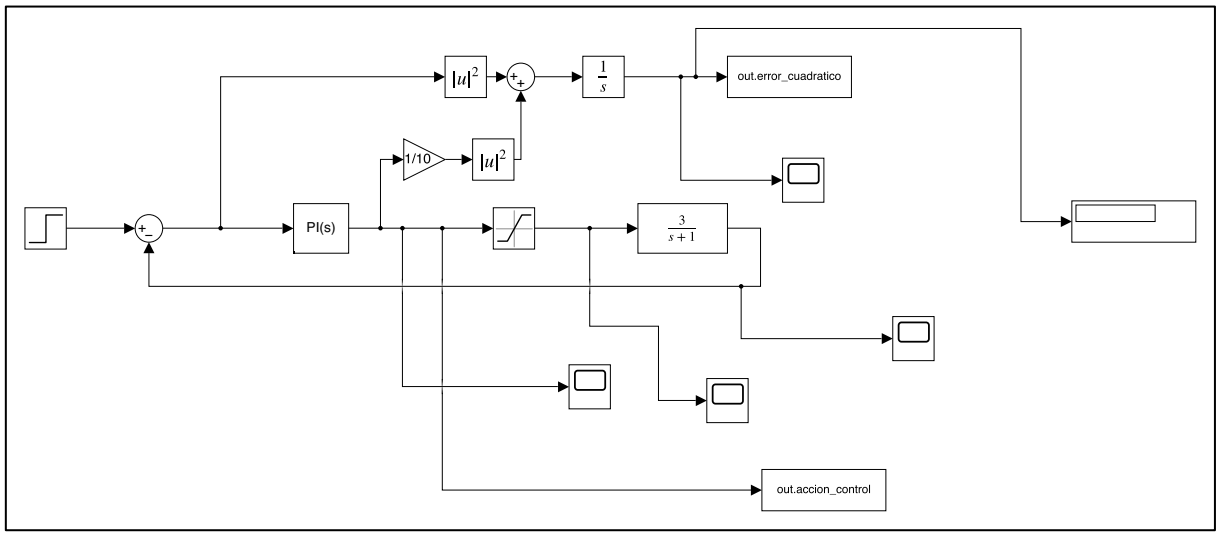


Figura 8. Modelo de Simulink para la prueba del controlador PI de la validación

Para determinar la validez de SafeOptBayes en la gestión de restricciones de seguridad, se introdujo la condición adicional de que la acción de control no debía superar un valor máximo determinado. Se decidió poner esta restricción porque en muchos procesos es importante que la acción de control no supere ciertos valores ya que sino esto podría producir daños en los actuadores del proceso y hacer que el sistema terminara funcionando peor.

Aunque se realizaron pruebas con controladores PID, las pruebas finales se centraron en controladores PI para poder llevar a cabo una mejor visualización de los resultados, si se hacían con un PID al haber tres variables a controlar no se podía llevar a cabo la visualización 3D de la optimización.

La validación se enfoca en cómo cada función de optimización maneja la mejora de la función objetivo y el cumplimiento de las restricciones de seguridad, ofreciendo así una evaluación completa del funcionamiento de OptBayes y SafeOptBayes en un entorno práctico.

Si se desea profundizar en el procedimiento completo que se llevó a cabo durante la validación de OptBayes y SafeOptBayes, incluyendo detalles sobre las pruebas realizadas y los resultados obtenidos, se encuentra disponible en el Anexo II, donde se documenta detalladamente el proceso y las conclusiones derivadas de cada optimización realizada.

4. CONTROL ÓPTIMO DE UN PÉNDULO INVERTIDO

El péndulo invertido es un sistema que pese a parecer muy simple no lo es y presenta un desafío en el campo del control. No es fácil mantener en equilibrio vertical una barra sobre un carro en movimiento dada la inestabilidad del sistema. Esto lo convierte en un modelo más que adecuado para poner a prueba diversas técnicas de control.

La principal razón por lo que este sistema es más que interesante es porque refleja situaciones reales que enfrentan los ingenieros en diferentes campos de la industria como la automoción, la robótica o la aviación.

La razón por la que este sistema es tan interesante es porque refleja situaciones reales que enfrentan los ingenieros en industrias como la robótica, la automoción y la aviación. Por ejemplo, estabilizar drones durante maniobras complejas o mantener el equilibrio de robots humanoides mientras caminan, son problemas que comparten las mismas dificultades que el control de un péndulo invertido.

Además, este tipo de sistema requiere una intervención constante para mantenerse en equilibrio, lo que significa que cualquier técnica de control aplicada debe ser tanto rápida como precisa. Esto hace que el péndulo invertido sea un excelente banco de pruebas para nuevas metodologías, como las técnicas de optimización que se explorarán en este trabajo.

En resumen, el péndulo invertido es un caso de estudio valioso porque permite demostrar cómo las técnicas avanzadas de control y optimización pueden mejorar la eficiencia y precisión en situaciones donde mantener la estabilidad es clave.

4.1. FUNDAMENTOS TÉCNICOS DEL PÉNDULO INVERTIDO

El péndulo invertido es un sistema que, debido a su inestabilidad inherente, necesita un control preciso para mantener la barra en posición vertical. Este sistema consta de una barra rígida montada sobre un carro que se desplaza a lo largo de una pista, la cual puede estar inclinada. La barra está conectada al carro mediante un pivote, lo que permite que gire libremente. Sin un control adecuado, la gravedad haría que la barra cayera, por lo que el objetivo es ajustar el movimiento del carro para mantener la barra en equilibrio vertical.

Descripción del Sistema

El sistema del péndulo invertido incluye los siguientes componentes principales:

1. La Barra

   Es una pieza rígida que está unida al carro a través de una articulación que le permite rotar. Cuando se inclina, la gravedad la hace caer, mientras que la inercia resiste los cambios en su movimiento, manteniendo su trayectoria a menos que el control del carro la corrija.

2. El Carro

Se desplaza a lo largo de una pista horizontal, y su movimiento es controlado por un motor. La fuerza que el motor aplica al carro es la que permite contrarrestar la inclinación de la barra. Para mantener la barra en equilibrio, es necesario ajustar continuamente el movimiento del carro.

3. Sensores y Actuadores

El sistema cuenta con varios sensores que proporcionan la información necesaria para mantener el péndulo en equilibrio. Estos sensores miden principalmente el ángulo de inclinación de la barra, que permite determinar su desviación respecto a la vertical, y la posición del carro en la pista. Aunque la velocidad angular y la velocidad del carro no se controlan se monitorizarán también ya que pueden aportar información que ayude a ajustar el control

El motor, como único actuador del sistema, es responsable de generar las fuerzas necesarias para mover el carro y mantener la barra en equilibrio. Estas fuerzas se ajustan en función de las señales de control que se generan a partir de las mediciones realizadas por los sensores.

Dinámica del Sistema

La dinámica del péndulo invertido está determinada por las siguientes ecuaciones diferenciales, que describen el comportamiento tanto de la barra como del carro en el sistema:

- Ecuación del Movimiento del Carro:

$$\left(M_c + m_p\right)\dot{r} - m_p l cos(\theta - \alpha)\ddot{\theta} + m_p l sin(\theta - \alpha)\dot{\theta}^2 + \left(M_c + m_p\right)gsin(\alpha) + b\dot{r} + F_s sign(\dot{r}) = au$$

Donde:

- M_c es la masa del carro.
- m es la masa de la barra (péndulo).
- l es la longitud de la barra.
- θ es el ángulo de la barra con respecto a la vertical.
- α es la inclinación de la pista.
- b es el coeficiente de fricción del carro.
- F_s es la fuerza de fricción estática.
- u es la señal de control aplicada.
- a es un factor de escalado para la señal de control.

2. Ecuación del Movimiento Angular de la Barra:

$$\left(J_p + m_p l^2\right)\ddot{\theta} - m_p l cos(\theta - \alpha)\ddot{r} - m_p f l sin\theta + c\dot{\theta} = \tau$$

Donde:

- J_p es el momento de inercia de la barra.
- g es la aceleración debida a la gravedad.
- c es el coeficiente de amortiguamiento angular.
- τ es el par de torsión aplicado.

Estas ecuaciones representan las fuerzas y momentos que actúan sobre el sistema, controlando tanto el movimiento del carro como la rotación de la barra [12].

Simplificación de Perturbaciones

Para facilitar el análisis y diseño del controlador, se decidió simplificar algunas de las perturbaciones en el modelo, aplicando las siguientes consideraciones:

- Inclinación de la Pista (α), Torque Externo (τ) y Fuerza de Fricción (F_s): Se consideran nulas. Esto implica que estas perturbaciones externas no se incorporan en las ecuaciones del modelo.

Estas simplificaciones permiten enfocar el diseño del controlador sin introducir la complejidad adicional que estas perturbaciones traerían al modelo.

Consideraciones de Estabilidad

Dado que el péndulo invertido es un sistema muy inestable, cualquier pequeña perturbación puede hacer que la barra caiga si no se controla correctamente. Por eso, la estabilidad del sistema es crucial. El controlador debe ser lo suficientemente rápido para corregir cualquier desviación antes de que se vuelva inmanejable.

El diseño del sistema de control también tiene que considerar factores como la fricción en las articulaciones, las limitaciones del motor y el ruido en las mediciones de los sensores. Estos elementos, junto con las incertidumbres en las masas, inercias y la dinámica del actuador, pueden introducir complicaciones en el sistema, lo que exige un diseño robusto para asegurar un buen rendimiento bajo todas las condiciones operativas. Por esta razón, se ha optado por desarrollar un controlador PID utilizando musyn, un método que tiene en cuenta estas variaciones y permite diseñar un controlador más robusto frente a las incertidumbres del sistema.

Este análisis técnico del péndulo invertido, junto con las ecuaciones que describen su comportamiento, ofrece una base sólida para desarrollar controladores avanzados que puedan manejar las dinámicas complejas y las perturbaciones externas, asegurando que el sistema funcione de manera óptima.

4.2. TÉCNICAS DE CONTROL

En este apartado, se explorarán las técnicas de control aplicadas al péndulo invertido, específicamente el Regulador Cuadrático Lineal (LQR) y la mu-síntesis (musyn). Mientras que el LQR proporciona un enfoque eficaz para estabilizar sistemas linealizados, musyn ofrece la ventaja de

gestionar incertidumbres, lo que lo hace más adecuado para aplicaciones donde los parámetros del sistema pueden variar. A su vez, se evaluará la viabilidad de utilizar estas técnicas como base para realizar la posterior optimización mediante las herramientas OptBayes y SafeOptBayes.

El objetivo final es controlar el proceso utilizando dos controladores PD configurados en paralelo: uno encargado de la posición del carro y otro dedicado al ángulo del péndulo. En esta configuración, ambos controladores actúan simultáneamente, pero de manera independiente, sobre la señal de control final.

4.2.1. LINEALIZACIÓN DEL MODELO DEL PÉNDULO INVERTIDO

La linealización es un paso principal para simplificar el análisis y diseño de controladores, permitiendo así el uso de técnicas de control lineal, como son LQR y musyn. En el caso del péndulo invertido, se asume que las no linealidades son pequeñas y que pueden despreciarse alrededor del punto de equilibrio. Para este proceso, se tienen en cuenta las siguientes consideraciones:

1. Desprecio de No Linealidades:

   Se asume que $\sin(\theta) = \theta$ y $\cos(\theta) = 1$. Estas aproximaciones son válidas para pequeñas oscilaciones del péndulo alrededor de la posición vertical.

2. Incorporación de Incertidumbre:

   Para reflejar las posibles variaciones en la dinámica real del sistema, se han modelado incertidumbres en ciertos parámetros:

   - Incertidumbre en masas e inercias: Se incorpora un 3% de incertidumbre en las masas (M_c, m_p) y en el momento de inercia (J_p)
   - Modelo de Actuador Dinámico**:** Sustituir el actuador ideal que produce una fuerza $F = a \cdot u$ por un actuador dinámico $F(s) = a \cdot 1/(0.02 \cdot s + 1) \cdot u$, con incertidumbre modelada con un filtro de primer orden de modo que tenga un error de un 3% a baja frecuencia, de aproximadamente un 10% a 50 rad/s, y de un 100% a 500 rad/s, y que a partir de esa frecuencia supere el 100% de error
   - Incertidumbre en la Viscosidad: Se añade un 10% de incertidumbre en el coeficiente de viscosidad b.

Modelo en Espacio de Estados

Con las simplificaciones mencionadas anteriormente, se construye el modelo en espacio de estados del péndulo invertido. Este enfoque permite representar las dinámicas del sistema facilitando el análisis y el diseño de los controladores.

En este modelo, los estados del sistema serán los siguientes por orden:

- Posición del Carro (r)
- Ángulo de la Barra (θ)
- Velocidad del Carro ($\dot{r}$)
- Velocidad Angular de la Barra ($\dot{\theta}$)

Las salidas del sistema, las variables que se controlan directamente serán las correspondientes a los dos primeros estados, $r\ y\ \theta$.

Matrices del Modelo en Espacio de Estados

Las ecuaciones de estado se expresan mediante un sistema de matrices compuesto por las matrices A, B, C y D. Estas dependen de los parámetros físicos del sistema, como las masas, las inercias, y los coeficientes de fricción y viscosidad y permiten describir cómo los estados del sistema evolucionan con el tiempo bajo la influencia de las entradas de control.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 2.805 & -5.256 & 0 \\ 0 & 18.55 & -7.728 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 3.754 \\ 5.52 \end{bmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

El sistema en espacio de estados se forma con estas matrices, permitiendo un análisis completo del comportamiento del péndulo invertido y la aplicación de técnicas de control.

Consideraciones para el Controlador LQR

Aunque el modelo en espacio de estados incluye incertidumbre en los parámetros, el Controlador LQR no puede manejar estas incertidumbres. Por lo tanto, para diseñar este controlador, se utilizan los valores nominales del sistema, que representan el modelo sin considerar variaciones en los parámetros.

4.2.2. Diseño de Controladores con Musyn

Musyn es una técnica avanzada que se usa principalmente para diseñar controladores robustos, especialmente en sistemas con incertidumbres significativas. Aunque normalmente se aplica en el diseño de controladores más complejos, musyn también puede ser útil para ajustar y optimizar controladores más simples como los PID, garantizando que funcionen bien incluso cuando las condiciones del sistema cambian.

El primer paso para el diseño de los controladores es modelar las incertidumbres del sistema que se comentaron anteriormente. Para modelar las incertidumbres de las masas, inercias y de la viscosidad, se utiliza el comando *ureal* de MATLAB, que permite definir parámetros inciertos con un valor nominal y un porcentaje de variación, reflejando las posibles fluctuaciones en el sistema real.

Después de modelar las incertidumbres en las masas, inercias y viscosidad, el siguiente paso es incorporar la dinámica del actuador al sistema, considerando también la incertidumbre en su

comportamiento. Para ello, se utiliza el comando *ultidyn* para definir una incertidumbre dinámica (deltaact), y *makeweight* para modelar cómo esta incertidumbre afecta al actuador a diferentes frecuencias.

```
deltaact=ultidyn('deltaact',[1 1]);
Wdelta = makeweight(0.03,500,2);
sigma(Wdelta), grid on
act = 1/(0.02*s+1)*(1+Wdelta*deltaact);
P = sys*act;
```

Tabla 16. Definición y Ajuste de la Dinámica del Actuador con Incertidumbre

Se define el actuador dinámico con un filtro de primer orden, representado por $\frac{1}{0.02 \cdot s+1}$, al que se añade la incertidumbre modelada por *Wdelta*. Esta incertidumbre se introduce multiplicando el sistema base por el modelo dinámico del actuador, resultando en la integración del actuador con el sistema, lo que significa que el sistema *sys* es afectado por el actuador y su incertidumbre.

Este proceso permite que el diseño del controlador considere la incertidumbre en la respuesta del actuador, lo que es importante si se desea garantizar que el controlador diseñado sea robusto frente a variaciones en la dinámica del actuador cuando se aplique en un sistema real.

Después de incorporar la dinámica del actuador y modelar la incertidumbre, el siguiente paso es establecer los objetivos de control para la posición del carro. En este caso, queremos que el carro siga una referencia con un error de posición que no supere el 2.5% en frecuencias bajas. Además, buscamos que el pico de resonancia en la sensibilidad, que refleja la respuesta del sistema a las entradas de referencia, se mantenga por debajo de 4 unidades. Esto es crucial para evitar inestabilidad o una respuesta demasiado oscilatoria.

Para alcanzar estos objetivos, definimos un ancho de banda objetivo para el sistema, lo que determina la rapidez con la que responde a los cambios en la referencia. En el código, hemos fijado este ancho de banda en 0.45 rad/s, indicando que, a partir de esta frecuencia, el rendimiento del sistema podría empezar a disminuir.

Además del filtro aplicado al error de la posición, se han configurado también filtros específicos para el ángulo del péndulo y la acción de control, con el objetivo de ajustar de manera precisa la respuesta del sistema a estos parámetros clave.

- Para el ángulo del péndulo, se ha establecido $W_{ang} = 0.01$. Este valor refleja la importancia de mantener el ángulo dentro de los límites deseados, permitiendo una respuesta controlada sin que el sistema reaccione de manera demasiado agresiva.
- En cuanto a la acción de control, se ha utilizado $W_u = 0.2$, lo que ayuda a regular la magnitud de la señal de control. Este filtro penaliza esfuerzos de control excesivamente altos, asegurando que la respuesta del sistema sea estable y controlada sin generar movimientos bruscos.

Una vez determinados los filtros, se procede a calcular la Planta Generalizada Ponderada (GPW). Para ello, primero se obtiene la Planta Generalizada No Ponderada (GPNW), que se expresa matricialmente como:

$$GPNW = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -P(1) \\ 0 & 1 & -P(2) \end{bmatrix}$$

- P (1) corresponde a la posición real del carro r.
- P (2) corresponde al ángulo del péndulo θ que se mide directamente.

Entradas y Salidas del Sistema:

Entradas por orden:

- Referencia: El objetivo que deseamos que el sistema alcance, es decir, la posición deseada del carro.
- Ruido de Medida del Ángulo: Perturbaciones que afectan la medición del ángulo del péndulo.
- Acción de Control: La señal enviada al actuador para controlar el sistema.

Salidas por orden:

- Error: La diferencia entre la referencia y la posición real del carro. Este valor indica qué tan bien el sistema sigue la referencia.
- Ángulo del Péndulo: La medición del ángulo del péndulo, que es crucial para mantener el equilibrio.
- Acción de Control: La señal de control aplicada por el sistema.

Planta Generalizada Ponderada (GPW)

Para adaptar el diseño del controlador musyn a las necesidades del sistema, se aplican filtros ponderados a las señales relevantes. Esto nos lleva a obtener la Planta Generalizada Ponderada (GPW). La ponderación de las señales nos permite ajustar cómo priorizamos los errores, el esfuerzo de control y la sensibilidad del sistema al ruido, lo que optimiza el rendimiento del controlador en su conjunto.

$$GPW = \begin{bmatrix} W_{err} & 0 & 0 & 0 \\ 0 & W_u & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot GPNW \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & W_{ang} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Como se puede observar en la imagen superior as entradas y salidas que se conectan directamente al controlador no se ponderan. Esto se hace para preservar la integridad de esas señales y garantizar que las ponderaciones solo afecten la gestión del ruido y las perturbaciones, sin alterar la interacción del controlador con el sistema. Así, la GPW se configura para conseguir una respuesta óptima del sistema bajo las condiciones reales de operación.

Una vez obtenida la planta generalizada ponderada, se procede a determinar los controladores PD utilizando musyn. Para ello, se configuran los controladores PD en MATLAB, estableciendo las conexiones necesarias entre la planta y los controladores.

Para determinar los controladores PD con musyn, primero se definen los controladores PD usando *tunablePID* en MATLAB, uno para la posición del carro y otro para el ángulo. Luego, se configuran las entradas y salidas de la planta generalizada ponderada (GPW) y de los controladores, asegurando que las señales estén correctamente conectadas. Las salidas de los controladores PD se suman para generar la señal de control final. Usando la función *connect*, se interconectan todos los elementos del sistema.

Una vez que el sistema está conectado, se utiliza el comando musyn para ajustar los controladores PD de manera óptima. Para lograr esto, musyn realiza un proceso iterativo en el que evalúa y ajusta continuamente los parámetros de los controladores. Durante cada iteración, analiza cómo responde el sistema a las incertidumbres y verifica si se cumplen las especificaciones de rendimiento. Con cada ajuste, musyn va afinando los controladores, buscando un equilibrio entre estabilidad y rendimiento, asegurando que el sistema funcione de manera efectiva en todos los escenarios considerados.

Después de completar el proceso iterativo, se obtiene la siguiente configuración óptima para los controladores. El primer controlador es el diseñado para gestionar la posición del carro, mientras que el segundo se encarga del control del ángulo.

- Controlador 1:

$$K_p = -0.408;\ K_d = -2.39;\ T_f = 1.59$$

- Controlador 2:

$$K_p = -53.6;\ K_d = 83.9;\ T_f = 2.08$$

El siguiente paso es evaluar el rendimiento de los controladores optimizados en MATLAB, utilizando una respuesta al escalón para observar cómo se comporta el sistema en bucle cerrado. A continuación, se presentan las gráficas que muestran los resultados obtenidos.

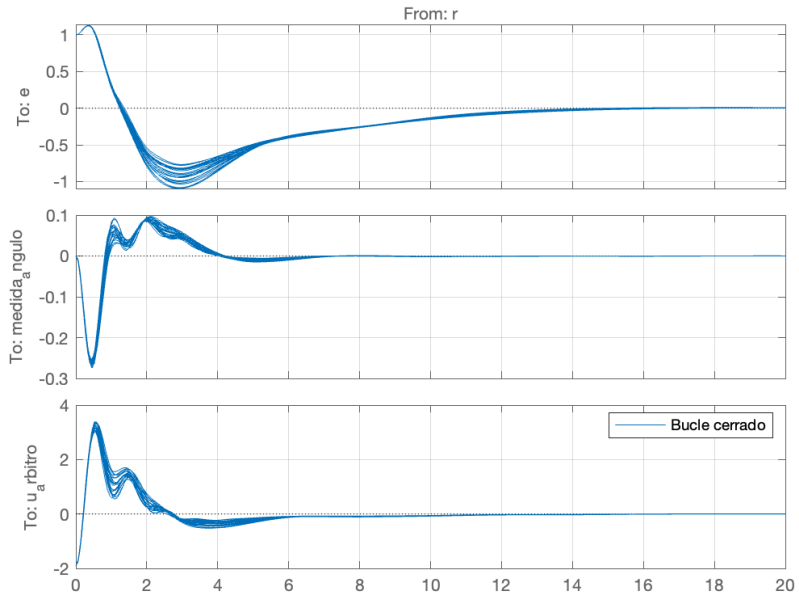


Figura 9. Respuesta en bucle cerrado del sistema con incertidumbre utilizando los controladores PD desarrollados mediante Musyn

Las gráficas muestran cómo las salidas del sistema, específicamente la posición del carro, el ángulo del péndulo y la acción de control, responden bajo la influencia de los controladores PD diseñado. Estos consiguen ajustar rápidamente la posición del carro, con algunas oscilaciones las cuales se estabilizan en un tiempo razonable al igual que el ángulo. Estos resultados indican que el sistema tiene un desempeño robusto con el modelo linealizado. El siguiente paso será probar el controlador en el sistema no lineal para evaluar su efectividad en condiciones más realistas.

Para probar el modelo no linealizado, se desarrolló el siguiente modelo en Simulink, al que se le añadió ruido de medida en los sensores, el cual se podrá observar con mayor detalle si se desea en Anexo V. Esto permite simular un entorno más realista y verificar la capacidad del controlador para mantener su rendimiento en condiciones similares a las de un sistema físico, donde las mediciones no son perfectas y pueden variar.

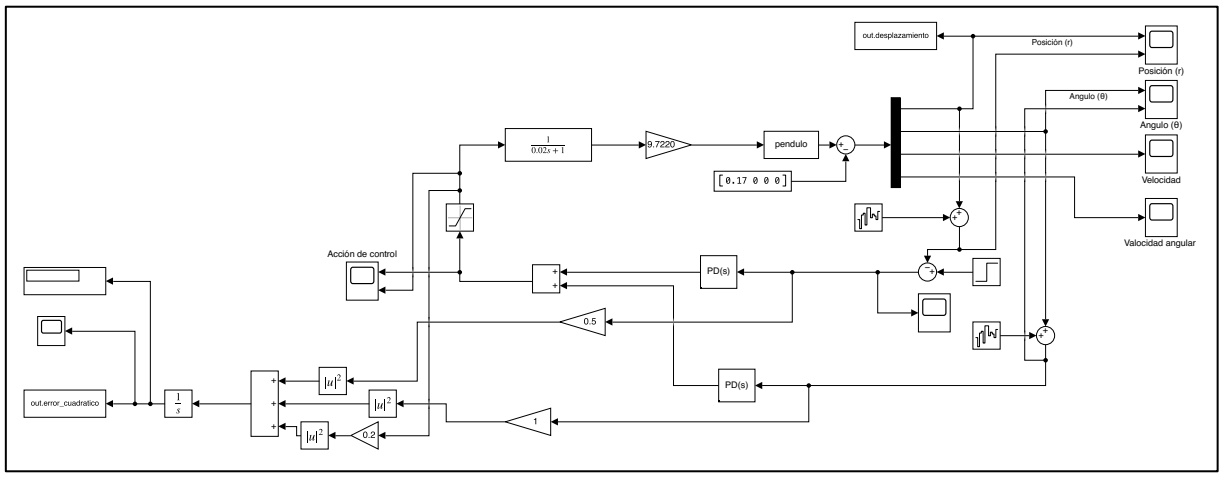
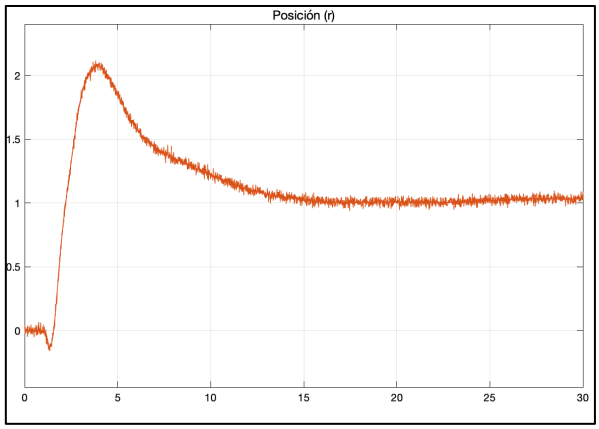
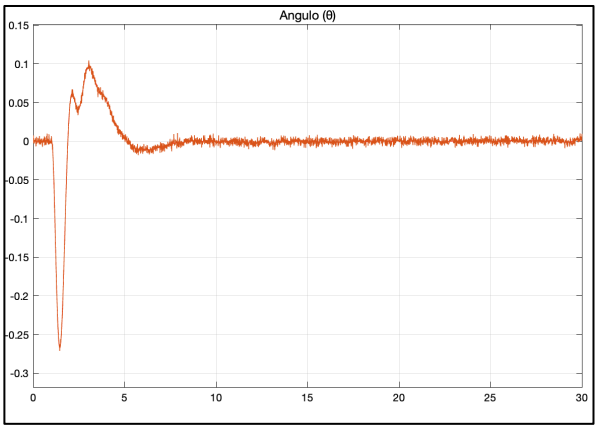


Figura 10. Modelo Simulink para probar y optimizar los controladores PD desarrollados con Musyn

A continuación, se muestran las gráficas de la posición del carro, el ángulo del péndulo y la acción de control:





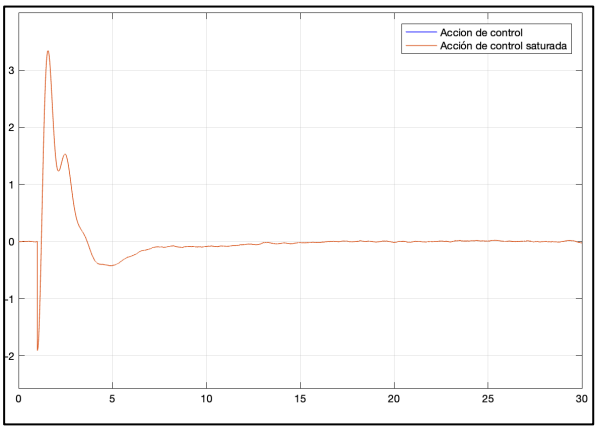


Figura 11. Respuestas del sistema y acción de control con los controladores PD desarrollados por Musyn

Los resultados obtenidos tras aplicar el controlador PD al modelo no linealizado con ruido de medida son muy similares a los obtenidos con el modelo linealizado. Aunque en ambos casos se observan oscilaciones iniciales, estas se estabilizan rápidamente, tanto en la posición del carro como en el ángulo del péndulo. Esto confirma que el controlador PD diseñado es robusto ante las incertidumbres del sistema y el ruido en las mediciones, sin que se noten grandes diferencias respecto al comportamiento del modelo linealizado.

En conjunto, estas gráficas demuestran que el controlador PD está manejando bien la estabilidad del sistema, ajustando tanto la posición del carro como el ángulo del péndulo de manera efectiva. Este resultado es un buen punto de partida para aplicar la optimización bayesiana, con la expectativa de que el sistema pueda alcanzar un rendimiento aún mejor.

4.2.3. Diseño del Controlador LQR

El diseño del controlador LQR se basa en la minimización de una función de costo que equilibra las desviaciones de los estados del sistema con el esfuerzo requerido para controlarlo. Esta función de costo se define a través de las matrices de ponderación Q y R.

La matriz Q pondera los estados del sistema, asignando mayor peso a aquellos que son más críticos para el control. En este caso, se decidió darle un peso mayor al ángulo del péndulo θ, ya que es muy importante mantenerlo en posición vertical.

Mientras que la matriz R penaliza el esfuerzo de control, es decir, la magnitud de la señal de control u. Un valor más alto de R implicará que el controlador será más conservador, limitando las acciones de control para evitar movimientos bruscos, mientras que un valor más bajo permite un control más agresivo.

Se probaron diferentes combinaciones de valores para Q y R hasta obtener una respuesta del sistema que mantuviera el equilibrio del péndulo de manera efectiva sin requerir un esfuerzo de control excesivo. La configuración final que se aplicó es la siguiente:

$$Q = [0.5, 1, 0, 0]$$

$$R = 0.2$$

Con esta configuración, se calculó la ganancia K del controlador LQR. Posteriormente, se evaluó el desempeño del controlador mediante simulaciones en MATLAB. La simulación se llevó a cabo utilizando el siguiente código, que muestra la respuesta del sistema en lazo cerrado:

```
% Crear sistema de lazo cerrado con la ganancia LQR calculada
A_cl = A - B * K;
sys_cl = ss(A_cl, B, [C;-K], [D;0]);


% Simulación de la respuesta inicial del sistema
[response, t] = initial(sys_cl, [1; 0; 0; 0]);
```

Tabla 17. ganancia calculada por el LQR

Gráficas de Respuesta del Sistema:

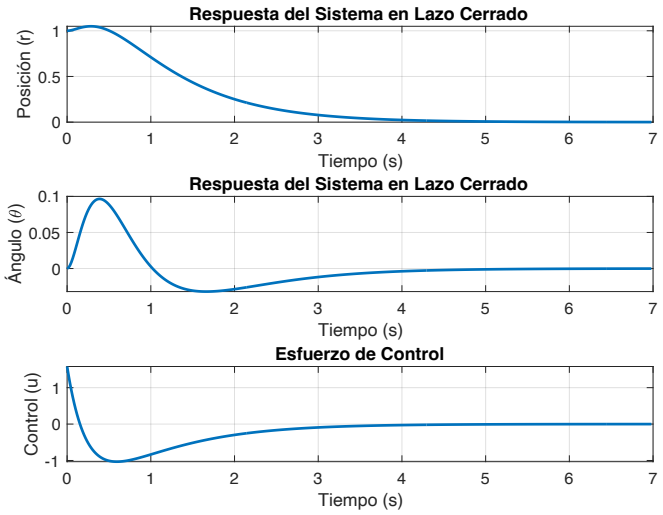


Figura 12. Respuesta del sistema con control LQR: Posición, Ángulo y Acción de Control

Las gráficas obtenidas muestran cómo el sistema reacciona ante una perturbación inicial, lo que permite evaluar tanto la estabilidad del sistema como la eficacia del controlador LQR para mantener el péndulo en equilibrio. Se puede ver que el controlador actúa de manera efectiva, estabilizando el sistema sin requerir un esfuerzo de control excesivo.

Tras confirmar que el controlador funcionaba bien con el modelo linealizado, se pasó a una validación más rigurosa utilizando el modelo no linealizado. Para esto, se implementó el modelo en Simulink, incorporando la complejidad de la dinámica no lineal del péndulo mediante una s-function. Esta validación permite comprobar que el controlador diseñado sea capaz de mantener la estabilidad del sistema real, el cual es más complejo que su versión linealizada.

Además, para acercar la simulación aún más a la realidad, se introdujeron incertidumbres en las mediciones y se añadió la dinámica del actuador. Adicionalmente, se estableció una saturación de la acción de control en un rango de -10 a 10 voltios para reflejar las limitaciones físicas del sistema. Estas modificaciones permitieron simular de manera más precisa las condiciones del sistema real, evaluando

cómo respondería el controlador ante las variaciones y asegurando que sea efectivo en un entorno más cercano al real. El modelo desarrollado en Simulink que incorpora estas características se muestra a continuación para una mejor comprensión de la estructura utilizada. Para verlo con mayor detalle, se encuentra disponible en el Anexo V.

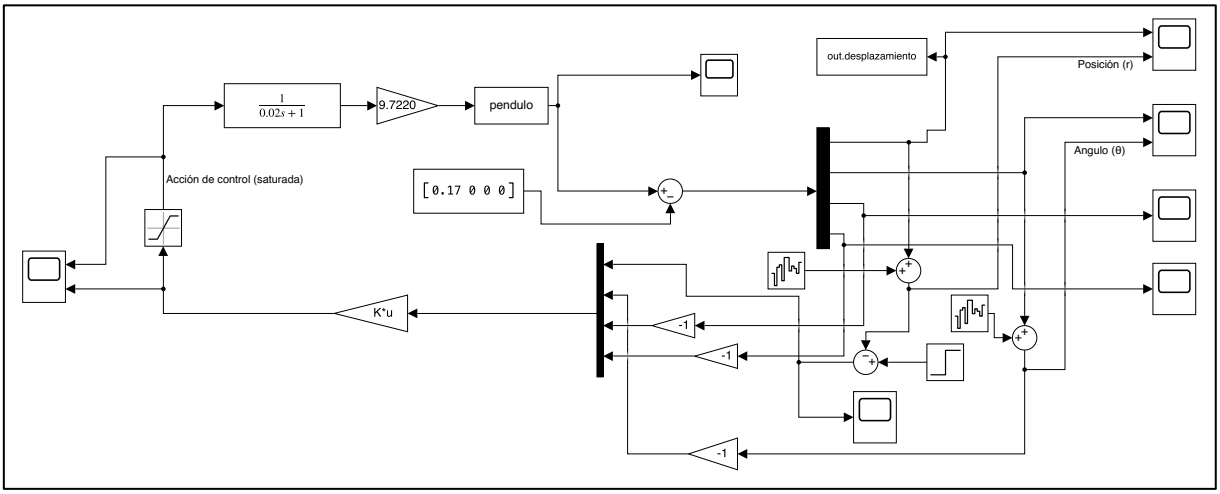


Figura 13. Modelo de Simulink para la prueba del controlador LQR

A continuación, se presentan las gráficas de las salidas resultantes al aplicar el controlador LQR, incluyendo la acción de control que muestra el esfuerzo necesario para estabilizar el sistema.

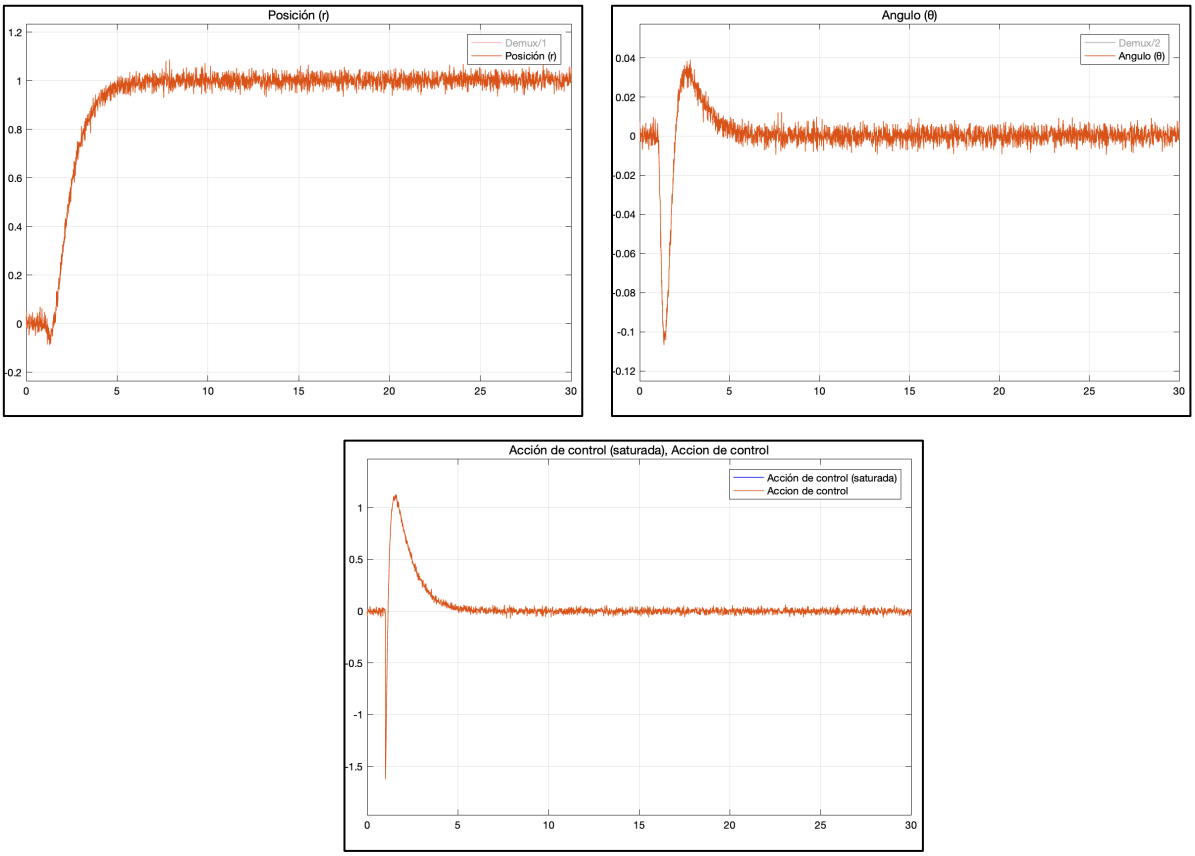


Figura 14. Respuestas del Sistema No Lineal con Control LQR y Ruido en la Medida: Posición, Ángulo y Acción de Control

Las gráficas muestran cómo las salidas del sistema (posición del carro y ángulo del péndulo) responden bajo la influencia del controlador LQR. Se puede observar cómo el controlador logra estabilizar tanto la posición como el ángulo en un tiempo razonable, a pesar de las incertidumbres y la dinámica añadida.

Controladores PD a partir de LQR

Para implementar los controladores PD derivados del diseño LQR, es importante tener en cuenta que, para controlar la posición del carro es necesario un PD con dos grados de libertad. Esto se debe a que el diseño LQR original tiene en cuenta tanto la posición como su derivada, lo que ayuda a mejorar el control y a mantener la estabilidad del sistema. Cuando intentamos implementar un controlador PD estándar solo con la referencia de la posición, se puede generar un desequilibrio inicial cuando se aplica un escalón, lo que provoca una sobreoscilación indeseada. Utilizar un PD de dos grados de libertad permite gestionar tanto la referencia de la posición como el feedback, evitando estas oscilaciones iniciales.

Por lo tanto, al derivar los controladores PD a partir del LQR, implementaremos dos controladores: uno para la posición del carro, que utilizará un PD con dos grados de libertad, y otro para el ángulo del péndulo, que será un PD estándar.

En la matriz de ganancia K obtenida, los elementos se corresponden de la siguiente manera:

$$K = [-1.58, 17.42, -3.95, 4.35]$$

- $K\ (1)$: Ganancia proporcional P1 para la posición del carro.
- $K\ (3)$: Ganancia derivativa D1 para la posición del carro.
- $-K\ (2)$: Ganancia proporcional P2 para el ángulo del péndulo.
- $-K\ (4)$: Ganancia derivativa D2 para el ángulo del péndulo.

Es necesario invertir las ganancias del controlador del ángulo dado que, en lugar de recibir el error con respecto al valor deseado (0), el controlador recibe directamente el valor absoluto del ángulo. Esto implica que, para corregir de manera adecuada las desviaciones, las ganancias proporcionales y derivativas deben invertirse, ya que el controlador espera operar sobre el error, no sobre el ángulo directamente.

Después de determinar los valores P y D para cada controlador, fue necesario ajustar el filtro de la derivada mediante un proceso de prueba y error. Esto se debe a que el modelo considera ruido en las mediciones y la dinámica del actuador. Si se utilizara un filtro de derivada demasiado alto, que podría ser aceptable en un sistema sin ruido, este amplificaría el ruido y afectaría negativamente el rendimiento del sistema. Además, cuando el filtro es demasiado alto, la acción de control entra en saturación de forma continua, lo que compromete la estabilidad del sistema.

Para realizar estas pruebas y ajustes, se utilizó el siguiente modelo de Simulink, el cual replica el comportamiento real del sistema, si se desea se podrá ver con mayor detalle en el Anexo V. Este modelo permitió observar la respuesta del sistema bajo diferentes configuraciones del filtro de la derivada, ayudando a encontrar el valor adecuado.

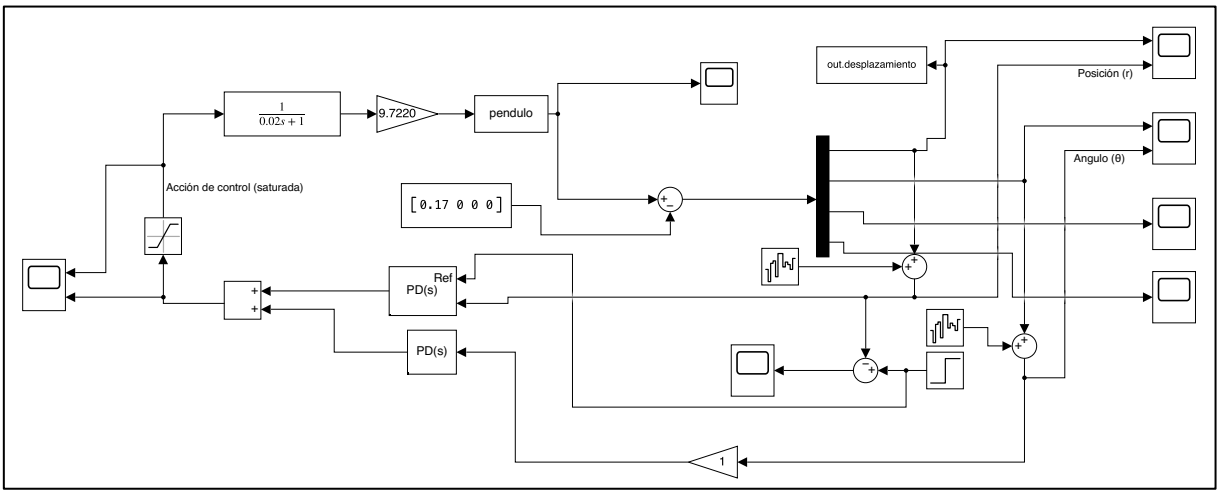


Figura 15. Modelo de Simulink para la prueba de controladores PD desarrollados a partir del LQR

Finalmente, se concluyó que un valor de 5 para el filtro de la derivada para ambos controladores ofrecía la mejor respuesta posible en términos de rendimiento y estabilidad, dadas las condiciones de ruido y la dinámica del actuador del sistema.

A continuación, se muestran las gráficas de la posición del carro, el ángulo del péndulo y la acción de control.

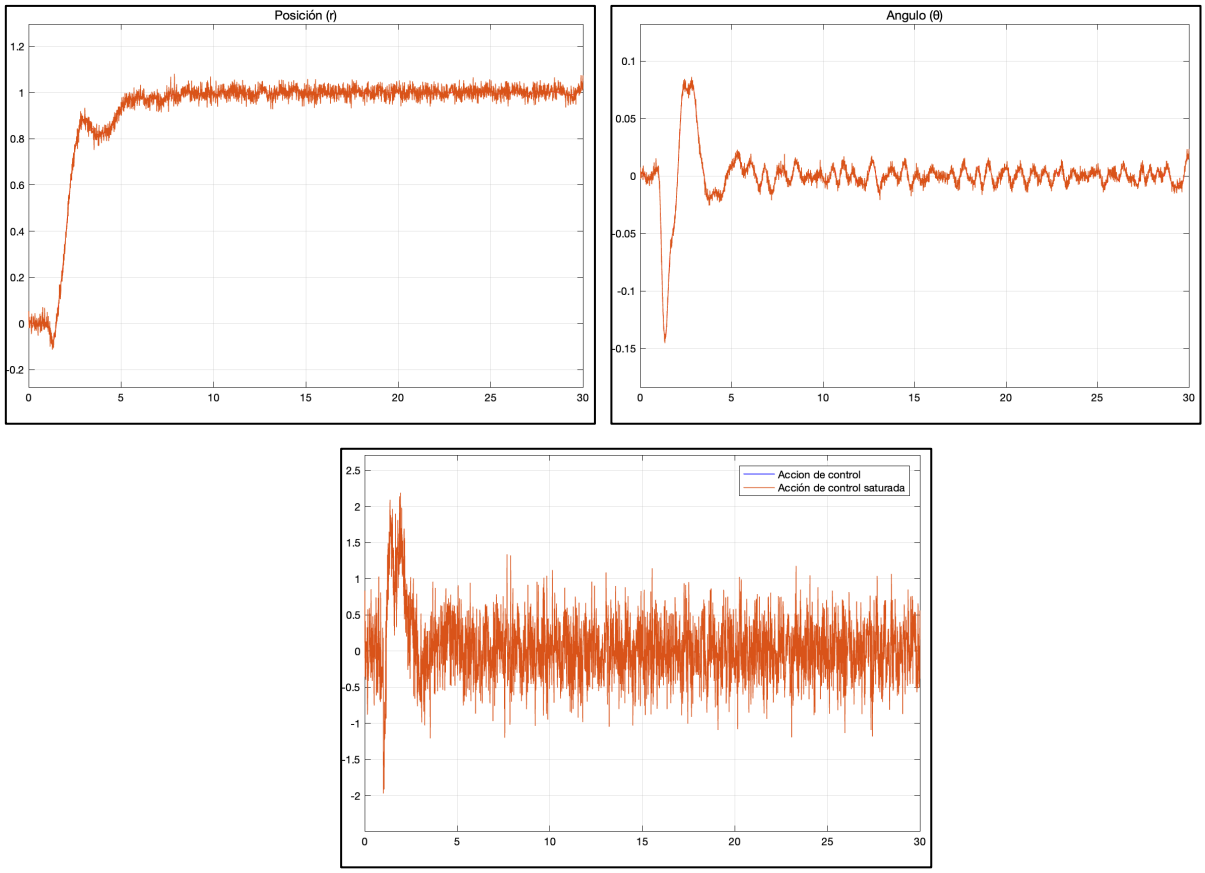


Figura 16. Respuestas con controladores PD desarrollados a partir del LQR: Posición, Ángulo y Acción de Control

Si las comparamos con las gráficas obtenidas con el controlador LQR, encontramos algunas similitudes, pero también diferencias claras, especialmente en la acción de control.

En cuanto a las salidas del sistema, ambos controladores logran estabilizar tanto la posición del carro como el ángulo del péndulo, alcanzando eficazmente las referencias deseadas. Sin embargo, la diferencia clave radica en que el LQR muestra un comportamiento más suave y estable. Una vez que las salidas alcanzan el valor esperado, las oscilaciones en el LQR son de mucha menor amplitud en comparación con las obtenidas con los controladores PD, lo que refleja un control más preciso y menos propenso a fluctuaciones.

La gran diferencia está en la acción de control:

- En el caso del LQR, tras el pico inicial para ajustar el sistema rápidamente, la señal de control se suaviza considerablemente y no presenta oscilaciones repetitivas, lo que muestra que el controlador apenas necesita hacer correcciones una vez que el sistema está equilibrado.
- En el caso de los controladores PD, aunque también se logra estabilizar el sistema, la acción de control oscila de forma continua a una frecuencia alta entre aproximadamente -1V y 1V. Estas oscilaciones constantes no superan los límites permitidos, pero su frecuencia elevada podría causar problemas en la práctica. Esto incluye un mayor desgaste del actuador y una posible disminución de la eficiencia a largo plazo.

Pasar de un controlador LQR a un PD en un sistema real se complica cuando entran en juego factores como el ruido en las mediciones y la dinámica del actuador. Aunque el uso de un controlador de segundo orden para la posición puede ayudar a mitigar ciertos problemas, el rendimiento general está lejos de alcanzar al que ofrece el controlador LQR. Las oscilaciones constantes que se observan en el PD, aunque dentro de un rango permitido, indican que, aunque la solución es funcional, no es tan estable ni precisa como la que se consigue con el LQR. Esto refleja lo complejo que es implementar un controlador PD en condiciones que incluyen ruido y dinámicas adicionales, y cómo esas condiciones afectan su efectividad.

Dado que los dos controladores PD no ofrecen resultados suficientemente prometedores y el modelo presenta ciertas incertidumbres, que el enfoque LQR no contempla en su desarrollo, continuar con esta opción para la optimización implicaría riesgos innecesarios. Estas incertidumbres, como el ruido en las mediciones y la dinámica del actuador, complican aún más la mejora de los controladores PD, lo que podría resultar en un esfuerzo poco eficaz y con un coste que no se quiere asumir.

Por todas estas razones, finalmente se decidió desestimar esta opción como base para llevar a cabo la optimización bayesiana.

4.3. Optimización de los Controladores con restricciones

Para la optimización de los controladores, se utilizarán como base los controladores desarrollados mediante el método musyn, ya que, como se explicó en el apartado anterior, los controladores

obtenidos con el enfoque LQR no ofrecían suficientes garantías para funcionar de manera robusta en condiciones reales.

En este proceso de optimización, se empleará la herramienta SafeOptBayes, que permitirá optimizar el rendimiento de los controladores musyn con una restricción adicional de seguridad. En este caso, la restricción de seguridad que se impondrá será que la sobreoscilación de la posición del carro no supere el 130%. Esta restricción tiene como objetivo asegurar que el sistema se mantenga dentro de límites seguros, evitando que el comportamiento se vuelva inestable o difícil de controlar, lo cual es fundamental en un proceso tan propenso a desequilibrios como lo es el del péndulo invertido.

DEFINICIÓN DE LA FUNCIÓN DE COSTO Y LOS RANGOS DE BÚSQUEDA

Para llevar a cabo la optimización, el primer paso será definir una función de costo que incluya tanto el error cuadrático de las salidas (posición y ángulo) como la acción de control. Las ponderaciones de esta función se basan en los valores que se utilizaron previamente en el diseño del controlador LQR, ya que se considera que estos son los más adecuados para encontrar un equilibrio entre el rendimiento del sistema y el esfuerzo de control. Las ponderaciones son las siguientes:

- 0.5 para la posición
- 1 para el ángulo
- 0.2 para la acción de control

El siguiente paso es definir los rangos de búsqueda de los parámetros a optimizar. Es esencial que estos rangos estén bien acotados para evitar que el proceso de optimización se vuelva demasiado complejo o que el algoritmo tarde en encontrar una solución válida. Si los rangos fueran demasiado amplios, el algoritmo tendría que explorar un espacio de búsqueda enorme, incrementando innecesariamente el tiempo de cálculo y complicando el proceso.

Los rangos de búsqueda para los parámetros se han ajustado en torno a los valores base obtenidos durante el diseño inicial con musyn, permitiendo así explorar ajustes cercanos. Los rangos definidos son los siguientes:

- KP1: [-1.5, -0.1]
- KD1: [-4, -1]
- KP2: [-80, -50]
- KD2: [74, 94]
- N1: [0.4, 0.8]
- N2: [0.3, 0.75]

Resultados Iniciales

Para todo el proceso de optimización, se trabajará con un modelo que introduce variaciones en parámetros como las masas, inercias y el coeficiente de viscosidad. Estas modificaciones no solo permiten probar la robustez del controlador ante posibles incertidumbres del sistema, sino que también reflejan condiciones más realistas de operación. Como los controladores desarrollados

mediante musyn ya contemplan estas variaciones, se espera que el sistema mantenga su rendimiento robusto.

Antes de proceder con la optimización, es importante analizar los resultados iniciales obtenidos con este modelo modificado, ya que servirán como punto de comparación para evaluar las mejoras logradas. A continuación, se presentan las gráficas que muestran la posición del carro, el ángulo del péndulo y la acción de control en estas nuevas condiciones. En este escenario inicial, la función de costo toma un valor de 1.676.

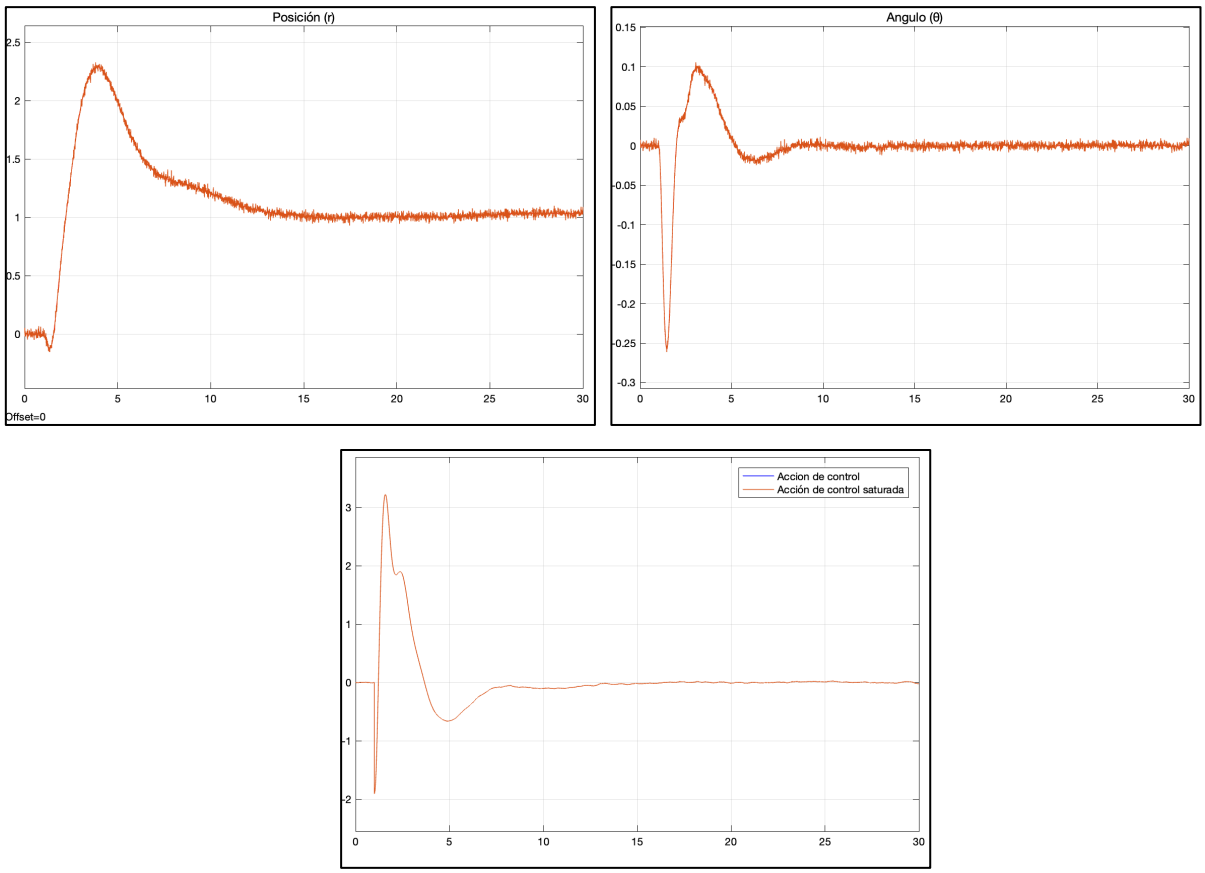


Figura 17. Resultados iniciales con incertidumbre en los parámetros del sistema: Respuesta de posición, ángulo del péndulo y acción de control antes de la optimización

Aunque los cambios respecto al modelo nominal son leves, estos reflejan cómo las variaciones en los parámetros físicos del sistema afectan el rendimiento. A pesar de ello, como se esperaba, el controlador *musyn* sigue mostrando un desempeño robusto, respondiendo eficazmente a las alteraciones. Esto confirma que será una base sólida para la optimización, tal y como se concluyó previamente en el análisis del sistema nominal.

Estrategias de optimización

1. Optimización global simultánea

En este enfoque, todas las variables se optimizan al mismo tiempo. Esto permite una visión más completa, ya que se evalúan todas las variables en una única fase. Aunque esta estrategia ofrece una

solución más global y detallada, tiene la desventaja de que el número de puntos a evaluar aumenta exponencialmente con cada nueva variable añadida, incrementando la complejidad y el tiempo de cálculo. Se realizarán 50 pruebas, tomando como muestra inicial el controlador base obtenido con musyn, asegurando así que se parte de una configuración ya funcional y robusta.

2. Optimización por fases

Este método divide el proceso de optimización en dos etapas. Primero, se optimizan las ganancias proporcionales (P) y derivativas (D), manteniendo fijos los valores de los filtros de derivada (N). En una segunda fase, una vez optimizadas las P y D, se ajustan los valores de los filtros N. Este enfoque es más eficiente, ya que reduce la cantidad de puntos a evaluar en cada fase. Se realizarán 30 pruebas para optimizar P y D, y luego se ajustarán los filtros N con 20 pruebas adicionales. Al igual que en el enfoque global, en este caso también se utilizará el controlador base obtenido con musyn como muestra inicial.

4.3.1. MÉTODO 1: OPTIMIZACIÓN GLOBAL SIMULTÁNEA

Como se explicó en el apartado anterior, en este enfoque se ajustan todas las variables del sistema en una única fase para evaluar la mejor configuración posible. A continuación, se muestran los resultados obtenidos al aplicar este método de optimización.

```
Optimization completed.
MaxObjectiveEvaluations of 50 reached.
Total function evaluations: 50
Best observed feasible point:
    KP1_escaled    KD1_escaled    KP2_escaled    KD2_escaled     N1_escaled     N2_escaled
         0.0000         0.0000         0.0000         0.0000         1.0000         0.8203
Observed objective function value = -1.31800
Descaled objective function value = 1.04610
LengthScale = 1.00000
SigmaF = 6.81250
SigmaN = 0.01000
```

Figura 18. Valor de los parámetros de los controladores tras la optimización global simultanea

Si se desescalan los valores de las variables que están escalados entre 0 y 1 se obtienen los siguientes valores para las variables controladas:

- $KP1 = -1.5$
- $KD1 = -4$
- $N1 = 0.8$
- $KP2 = -80$
- $KD2 = 74$
- $N2 = 0.66914$

Con esta configuración, se alcanzó un valor de la función de costo de 0.99697, como se muestra en la imagen X. A continuación, se presentan las gráficas que ilustran el comportamiento de las salidas del sistema y la acción de control tras la optimización.

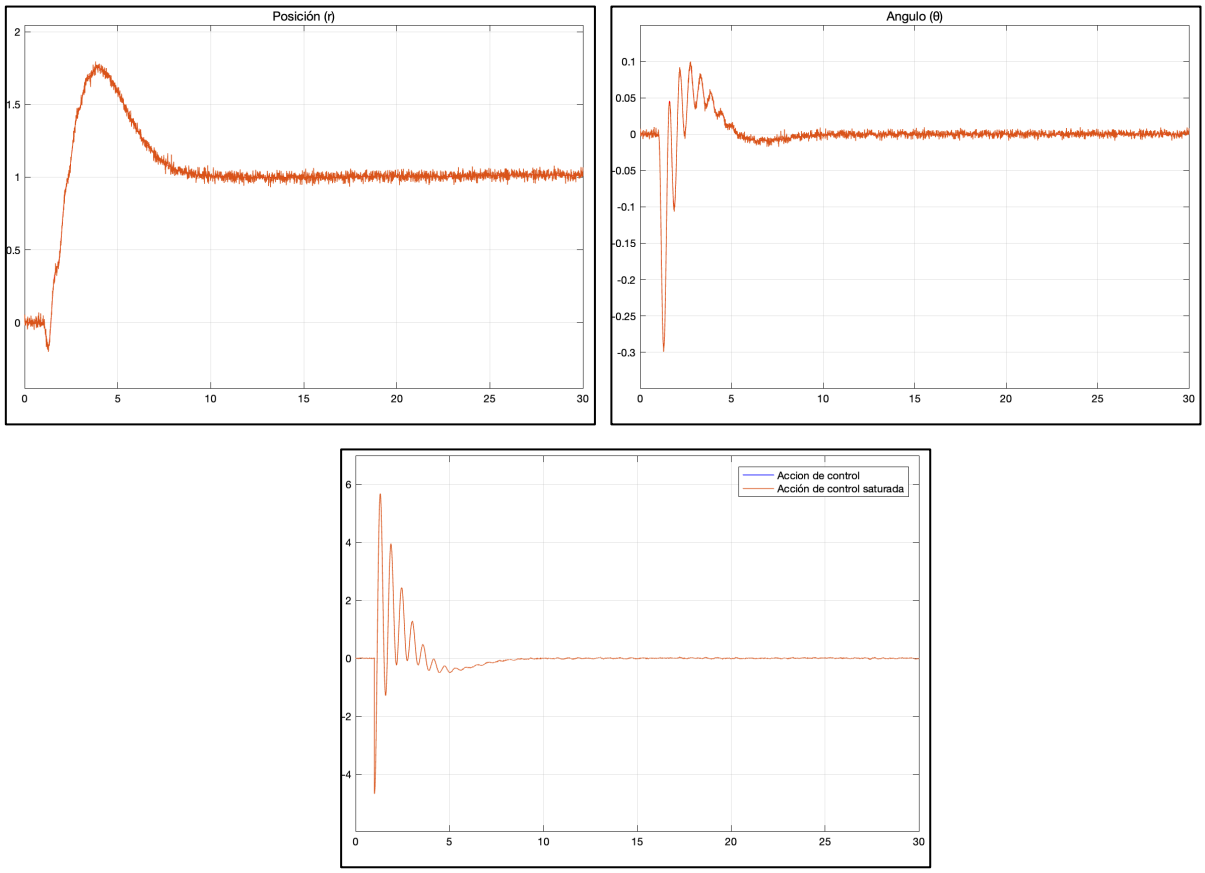


Figura 19. Respuesta de posición, ángulo del péndulo y acción de control tras la optimización global simultanea

4.3.2. MÉTODO 2: OPTIMIZACIÓN POR FASES

Aplicando este método en el cual se divide el proceso de optimización en dos etapas, con una etapa inicial para las ganancias proporcionales y derivativas y otra etapa para el filtro de la derivada se obtuvieron los siguiente resultados:

- Primera fase

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Best observed feasible point:
    KP1_escaled    KD1_escaled    KP2_escaled    KD2_escaled
         0.7498         0.0000         0.3075         1.0000
Observed objective function value = -1.33200
Descaled objective function value = 1.00400
LengthScale = 1.00000
SigmaF = 5.50260
SigmaN = 0.01000
```

Figura 20. Valores optimizados de los parámetros proporcionales (P) y derivativos (D) tras la optimización por fases

```
Optimization completed.
MaxObjectiveEvaluations of 20 reached.
Total function evaluations: 20
Best observed feasible point:
     N1_escaled     N2_escaled
         0.4496         0.4530
Observed objective function value = -1.33460
Descaled objective function value = 0.99620
LengthScale = 1.00000
SigmaF = 0.59177
SigmaN = 0.01000
```

Figura 21. Valores optimizados de los parámetros de los filtros derivativos (N) tras la optimización por fases

Si se desescalan los valores de las variables que están escalados entre 0 y 1 se obtienen los siguientes valores para las variables controladas:

- $KP1 = -0.67898$
- $KD1 = -4$
- $N1 = 0.62143$
- $KP2 = -80$
- $KD2 = 83.7057$
- $N2 = 0.62261$

Con esta configuración, se alcanzó un valor de la función de costo de 0.9, como se muestra en la Figura 21. A continuación, se presentan las gráficas que ilustran el comportamiento de las salidas del sistema y la acción de control tras la optimización.

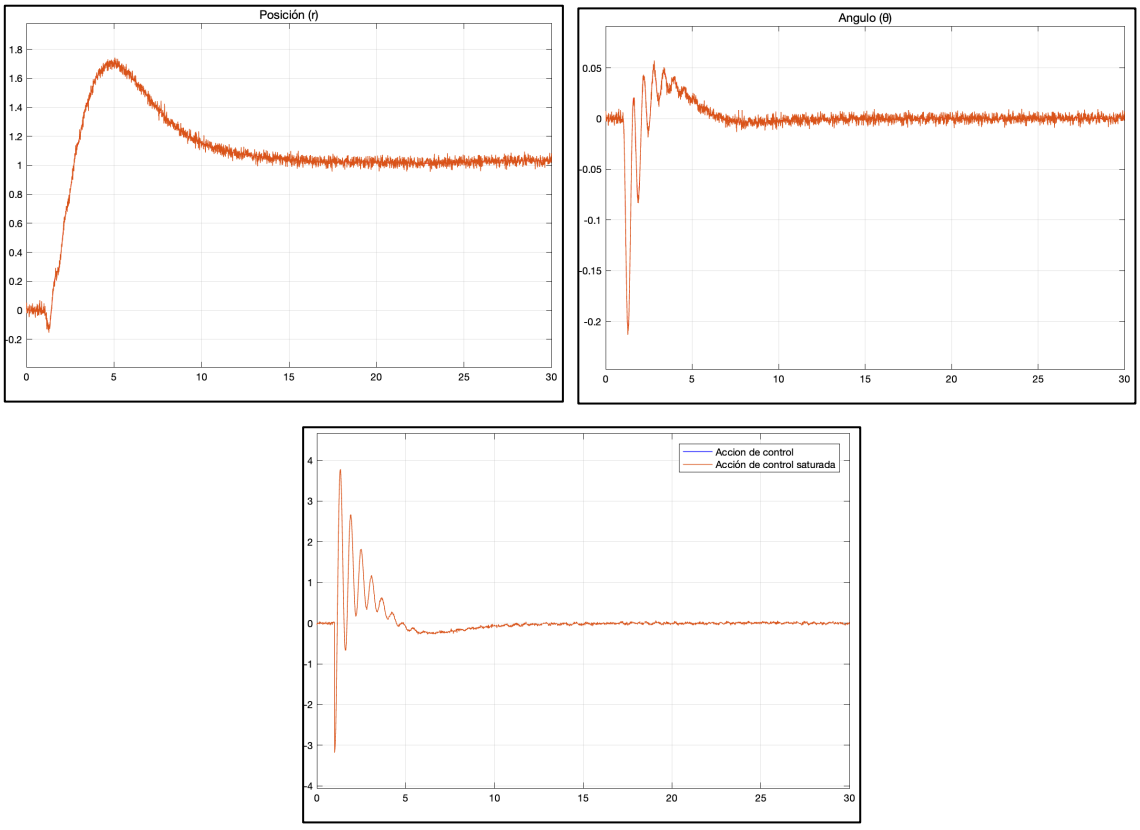


Figura 22. Respuesta de posición, ángulo del péndulo y acción de control tras la optimización por fases

4.3.3. COMPARACIÓN DE LOS RESULTADOS

Primero, es importante resaltar que ambos métodos de optimización empleados, tanto el global simultáneo como el de fases, han proporcionado resultados bastante similares en términos del comportamiento general del sistema. Ambos han mejorado significativamente la función de costo, reduciendo considerablemente la sobreoscilación en la posición del carro. Esto confirma la validez de ambas estrategias como enfoques viables para la optimización.

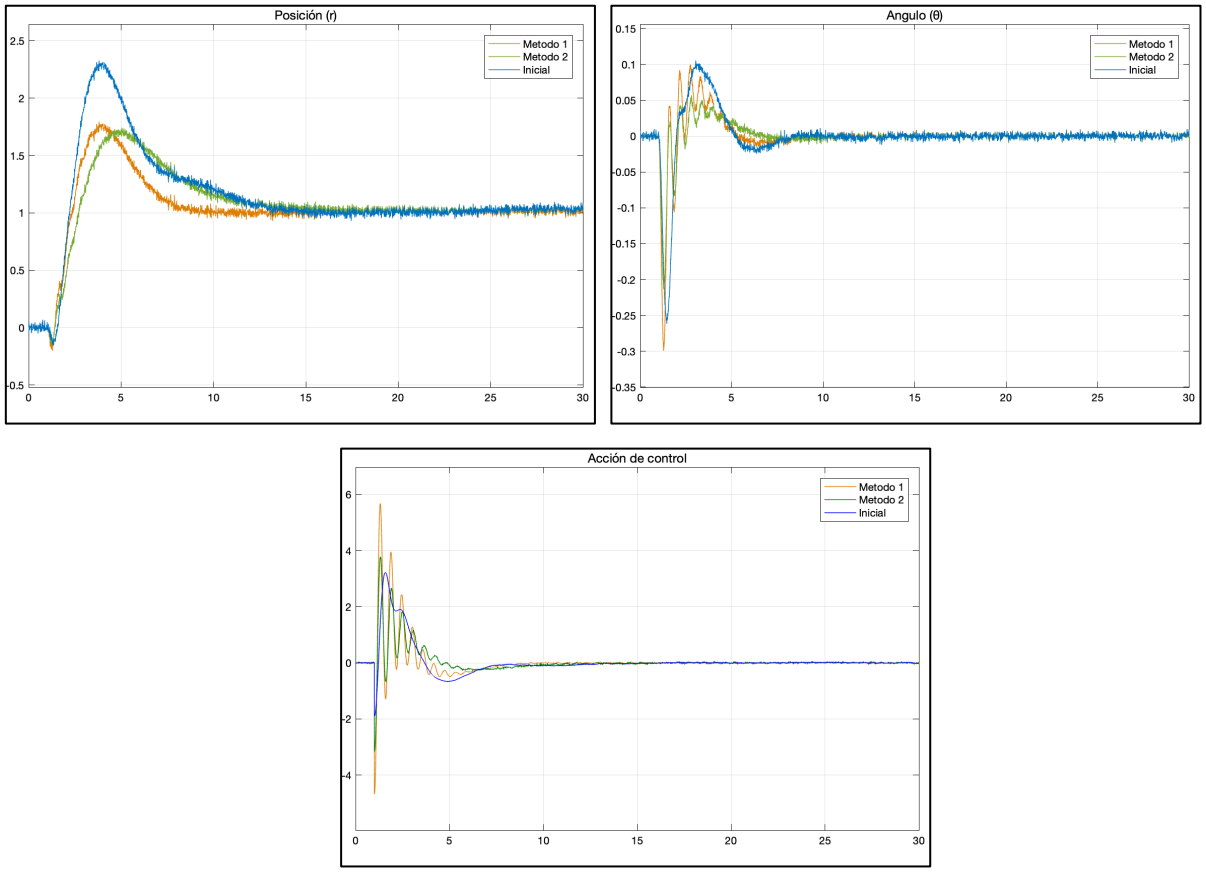


Figura 23. Comparación de la respuesta de posición, ángulo del péndulo y acción de control de la versión inicial con ambos métodos de optimización

Sin embargo, al analizar las salidas del sistema en mayor detalle, se observa que los dos métodos presentan un incremento en las oscilaciones iniciales, tanto en el ángulo del péndulo como en la acción de control. Aunque estas oscilaciones son más notorias tras la optimización, no comprometen la estabilidad del sistema. Estas oscilaciones iniciales más bruscas podrían deberse a las correcciones rápidas impuestas por los controladores optimizados.

Una posible solución para mitigar estas oscilaciones sería implementar filtros de alta frecuencia en la ponderación de las salidas dentro de la función de costo. Esto ayudaría al optimizador a suavizar las oscilaciones al atenuar las componentes de alta frecuencia, sin comprometer demasiado el rendimiento general. Sin embargo, ajustar estos filtros puede ser un desafío, ya que los operarios del sistema podrían encontrar difícil entender y manejar los ajustes adicionales que implican.

En cuanto a las diferencias, aunque ambos métodos logran una mejora notable en la sobreoscilación del carro, el método por fases parece alcanzar estos resultados con una mayor

eficiencia, empleando menos iteraciones para acercarse a la solución final. El método global simultáneo, por otro lado, tiende a explorar de manera más amplia el espacio de soluciones, lo que genera más variabilidad en las iteraciones iniciales. Esto se refleja claramente en las tablas de iteraciones llevadas a cabo en cada método, presentes en el Anexo IV.

En resumen, aunque los dos enfoques ofrecen soluciones válidas y efectivas, la optimización por fases parece lograr mejores resultados con mayor eficiencia, mientras que el enfoque global simultáneo explora un espacio más amplio antes de converger hacia soluciones cercanas al óptimo.

5. DISCUSIÓN Y CONCLUSIONES

A lo largo de este trabajo, se ha llevado a cabo la optimización de un sistema de control basado en un péndulo invertido, utilizando dos enfoques principales: la optimización global simultánea y la optimización por fases. Ambos métodos han mostrado resultados satisfactorios, aunque la optimización por fases ha sido más eficiente en cuanto al número de iteraciones necesarias para acercarse al óptimo.

El objetivo principal de este proyecto ha sido validar la optimización bayesiana offline en un entorno de simulación para optimizar un controlador inicial ya existente en un sistema inestable, ajustando los parámetros en función de las condiciones de operación. Las simulaciones han permitido refinar el comportamiento del sistema sin necesidad de realizar pruebas físicas, lo que ha supuesto un ahorro significativo de recursos. Los resultados obtenidos indican que esta metodología tiene un gran potencial para mejorar el rendimiento del sistema en aplicaciones futuras, reduciendo la cantidad de pruebas necesarias y optimizando tanto el tiempo como los costes asociados.

En cuanto a las limitaciones del estudio, es importante señalar que el número de iteraciones utilizado fue una decisión condicionada por los recursos y el tiempo disponibles. Durante el proceso de optimización, no se observó un estancamiento claro, lo que sugiere que con un mayor número de iteraciones se podrían haber alcanzado soluciones mejores. Este aspecto es clave para tener en cuenta en futuras aplicaciones, especialmente en entornos reales donde los costes asociados a cada prueba son elevados.

Con esto en mente, el siguiente paso será implementar el proceso de optimización bayesiana en tiempo real en el sistema físico. A diferencia del entorno simulado, en el proceso real el número de iteraciones no será fijo, sino que se ajustará dinámicamente en función del comportamiento del sistema en tiempo real. Esta capacidad de adaptación permitirá optimizar los parámetros del controlador de manera continua y eficiente, maximizando el rendimiento del sistema sin necesidad de un gran número de pruebas físicas. Además, la optimización en tiempo real ofrecerá la flexibilidad necesaria para adaptarse a las variaciones en las condiciones operativas, garantizando una mayor robustez y eficiencia en el rendimiento del sistema.

Por último, uno de los objetivos fundamentales de este proyecto ha sido desarrollar un enfoque de optimización que sea adaptable a otros sistemas industriales. Aunque el caso de estudio ha sido el péndulo invertido, la metodología desarrollada ha mostrado indicios de ser lo suficientemente robusta como para poder aplicarse en otros sistemas de control con retos similares. La flexibilidad del enfoque sugiere que podría ser implementado en una variedad de aplicaciones industriales que requieran optimización con un número limitado de pruebas

6. BIBLIOGRAFÍA/REFERENCIAS

[1] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. H. Gillula y C. J. Tomlin, «A General Safety Framework for Learning-based Control in Uncertain Robotic Systems,» *IEEE Transactions on Automatic Control,* 2019.

[2] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams y N. de Freitas, «Taking the Human Out of the Loop: A Review of Bayesian Optimization,» *Proceedings of the IEEE,* vol. Sección 4, 2016.

[3] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams y N. de Freitas, «Taking the Human Out of the Loop: A Review of Bayesian Optimization,» *Proceedings of the IEEE,* vol. Sección 3.1, 2016.

[4] E. Brochu, V. M. Cora y N. de Freitas, «A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning,» 2010.

[5] C. E. Rasmussen y C. K. I. Williams, Gaussian Processes for Machine Learning, vol. Capítulo 2: "Gaussian Process Regression", MIT Press, 2006.

[6] C. E. Rasmussen y C. K. I. Williams, Gaussian Processes for Machine Learning, vol. Sección 2: "Gaussian Processes for Optimization", MIT Press, 2006.

[7] D. R. Jones, M. Schonlau y W. J. Welch, «Efficient Global Optimization of Expensive Black-Box Functions,» *Journal of Global Optimization,* vol. Sección 3: "Expected Improvement Algorithm", 1998.

[8] Y. Sui, A. Gotovos, J. Burdick y A. Krause, «Safe Exploration for Optimization with Gaussian Processes,» de *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[9] F. Berkenkamp, A. Krause y A. P. Schoellig, «Safe Model-based Reinforcement Learning with Stability Guarantees,» de *Advances in Neural Information Processing Systems*, 2017, p. Capítulo 3: "Optimización segura y exploración de sistemas dinámicos".

[10] N. Chatzipanagiotis, S. Moustakidis y E. B. Kosmatopoulos, «Safe Optimization for Control Applications,» *IEEE Transactions on Control Systems Technology,* vol. Sección 2.4, 2017.

[11] N. Chatzipanagiotis, S. Moustakidis y E. B. Kosmatopoulos, «Safe Optimization for Control Applications,» *IEEE Transactions on Control Systems Technology,* p. Conclusiones, 2017.

[12] G. Medrano-Cersa, «Robust computer control of an inverted pendulum,» *IEEE,* 1999.

[13] A. S. Piqueras, Capitulo 6.6 y apendices A.2 y B. [En línea]. Available: HTTPS://PERSONALES.UPV.ES/ASALA/DOCENCIAONLINE/CURSOS/APUNTES.PDF .

PRESUPUESTO

1. OBJETIVO

El objetivo de este apartado es cuantificar los costes asociados al desarrollo y ejecución del proyecto presentado, centrándose en los siguientes elementos clave:

- **Coste de mano de obra:** Incluye el tiempo dedicado al desarrollo, implementación y optimización del sistema de control mediante técnicas de optimización bayesiana.
- **Coste de hardware:** Se refiere al uso del equipo necesario para llevar a cabo las simulaciones y pruebas del sistema.
- **Coste de software:** Considera las licencias de software utilizadas durante el desarrollo del proyecto.
- **Coste de Implementación:** Este coste incluye lo necesario para llevar a cabo la implementación de los modelos optimizados en un entorno real, así como las pruebas correspondientes.

En los siguientes apartados, se detalla el desglose de cada uno de estos costes, diferenciando entre los costes ya incurridos en el proyecto actual y los costes adicionales que implicarían futuras aplicaciones físicas del modelo.

2. COSTE DE MANO DE OBRA

Para la realización del presupuesto, se ha establecido un sueldo bruto de 2.500 euros mensuales para el ingeniero industrial encargado del proyecto en 2024. Además, la empresa deberá asumir los siguientes costes adicionales según la normativa vigente:

- Contingencias comunes: 23,60%
- Prestaciones por desempleo: 5,50%
- Formación profesional: 0,60%
- Fondo de Garantía Salarial (FOGASA): 0,20%
- IT/MS (accidentes de trabajo y enfermedades profesionales): 1,50%

Esto resulta en un coste mensual total de 3.285 euros para la empresa por cada mes trabajado por el ingeniero industrial.

Para llevar a cabo los cálculos del coste por mano de obra, se ha considerado una jornada laboral de 8 horas diarias y 40 horas semanales. En cuanto a los días festivos y las vacaciones, se han considerado 15 días festivos anuales y 22 días laborables de vacaciones, lo que equivale a un total de 7 semanas de descanso al año.

Las horas laborales anuales se obtienen restando las semanas no laborables de las 52 semanas del año, y multiplicando el resultado por las horas trabajadas por semana, obteniéndose un total de 1.800 horas.

Para calcular el coste por hora del ingeniero industrial, se ha tomado como base un coste anual bruto de 39.420 euros. Este valor incluye el sueldo base más las cotizaciones y gastos adicionales de la empresa. Finalmente se obtiene un coste de 22 €/h.

MOII = Mano de Obra Ingeniero Industrial

Tabla P 1. Coste de Mano de Obra

TAREAS	HORAS MOII	SALARIO	COSTE (€)
1. Planificación del proyecto	12	22	264
1.1. Reuniones para la planificación de plazos	2	22	44
1.2. Organización de tareas	5	22	110
1.3. Revisión de objetivos del proyecto	5	22	110
2. Investigación tecnologías aplicadas	66	22	1452
2.1. Análisis de bibliografía y estudios previos	66	22	1452

3. Desarrollo de herramientas de optimización	180	22	3960
3.1. Programación del modelo de simulación	70	22	1540
3.2. Implementación de la optimización bayesiana	60	22	1320
3.3. Pruebas preliminares de las herramientas	50	22	1100
4. Validación de resultados	55	22	1210
4.1. Análisis de los resultados obtenidos	20	22	440
4.2. Ajuste fino de parámetros	15	22	330
4.3. Comparación de controladores	20	22	440
5. Redacción del informe final	100	22	2200
5.1. Redacción de introducción y metodología	50	22	1100
5.2. Redacción de análisis de resultados	40	22	880
5.3. Redacción de conclusiones y recomendaciones	10	22	220
Total			9.086 €

3. COSTE HARDWARE

Para realizar este proyecto ha sido necesaria la utilización de un ordenador. El equipo utilizado por el ingeniero industrial no es de uso único para el proyecto, por lo que se calculará la amortización del mismo, teniendo en cuenta una vida útil del ordenador de 7 años. La duración del proyecto ha sido de 3 meses, lo que equivale a una amortización del 3,57% del coste total del equipo. En el coste total y la deprecación estará incluido el Impuesto Sobre el Valor Añadido (IVA).

Tabla P 2. Coste Hardware

Nº	PRODUCTO	CANTIDAD	COSTE	AMORTIZACIÓN	DEPRECIACIÓN
1	Ordenador Portátil MacBook Air M1	1u	1.044,00 €	3,57%	37,27
TOTAL HARDWARE					37,27 €

4. COSTE SOFTWARE

En este apartado se desarrollan los costes procedentes de las licencias de software necesarias para llevar a cabo el proyecto. Para este trabajo, se ha considerado la adquisición de las siguientes licencias:

- **Licencia de Microsoft Office 365 Empresa Premium**: Se ha calculado el coste proporcional a 3 meses de uso, teniendo en cuenta un precio aproximado anual de 150 €.
- **Licencia de MATLAB con Simulink y otros complementos**: Basado en un precio aproximado de 10.000 € anuales para empresas, también se ha calculado el coste proporcional a 3 meses de uso.

Tabla P 3. Coste Software

N.º	SOFTWARE	Coste anual	Porcentaje de uso	COSTE (€)
1	Licencia de Microsoft Office 365	150 €	25 %	37,5
2	Licencia de MATLAB con Simulink	10.000 €	25 %	2.500
TOTAL HARDWARE				2.537,50 €

5. COSTE DE IMPLEMENTACIÓN

Este apartado detalla los costes estimados para la posible implementación del modelo en pruebas físicas en un entorno real. Se considera la supervisión por parte de un ingeniero y la intervención de un operario especializado, que realizará los ajustes y las pruebas. Estos costes serían adicionales y opcionales, aplicándose solo en caso de que se decida ejecutar las pruebas en condiciones físicas reales.

Para calcular el coste del operario, se ha seguido el mismo procedimiento que para el ingeniero, tomando en cuenta un salario bruto de 1.600 euros mensuales.

Tabla P 4. Coste de Implementación

CATEGORÍA	HORAS	Salario	TOTAL (€)
Ingeniero de pruebas	100	22 €/hora	2.200,00
Operario especializado	100	14 €/hora	1.400
TOTAL COSTE IMPLEMENTACIÓN			3.600,00 €

6. PRESUPUESTO FINAL

En la siguiente tabla se muestra el presupuesto final del proyecto teniendo en cuenta el coste de mano de obra, del hardware, del software y de la implementación.

Tabla P 5. Presupuesto Final

DESCRIPCIÓN	COSTE (€)
COSTE DE MANO DE OBRA	9.086,00 €
COSTE HARDWARE	37,27 €
COSTE SOFTWARE	2.537,50 €
COSTE DE IMPLEMENTACIÓN	3.600 €
PRESUPUESTO DE EJECUCIÓN MATERIAL	15.260,77 €
10% GASTOS GENERALES	1.526,08 €
6% BENEFICIO INDUSTRIAL	915,65 €
SUMA	17.702,50 €
21%	3.717,53 €
PRESUPUESTO POR EJECUCIÓN POR CONTRATA	21.420,03 €

El coste final del presupuesto es de DIECISÉIS MIL TRESCIENTOS SESENTA Y SIETE CON SIETE EUROS.

ANEXOS

ANEXO I: FUNCIONES DE OPTBAYES Y SAFEOPTBAYES

Este anexo presenta las funciones desarrolladas para la optimización de controladores mediante los métodos OptBayes y SafeOptBayes. La optimización bayesiana (OptBayes) implementa un enfoque probabilístico para ajustar los parámetros del controlador con un número reducido de pruebas. Por su parte, SafeOptBayes, basado en OptBayes, introduce restricciones de seguridad que aseguran la estabilidad del sistema durante el proceso de optimización.

ANEXO I.1. FUNCIONES OPTBAYES

OptBayes es un conjunto de funciones que permite optimizar el sistema mediante la exploración de nuevas áreas de búsqueda y la explotación de regiones conocidas. La mayoría de las funciones de este bloque se utilizan también en SafeOptBayes.

1. Función de optimización

Tabla A 1. Función optbayes

```
function [Data] = optbayes(funcion, variables, args)
    arguments
        funcion function_handle
        variables cell {mustBeNonempty}
        args.ObjetivosEval (1,1) double = 100
        args.MuestrasIniciales (1,1) double = 5
        args.MuestraInicialManual double = []  % Nuevo parámetro para muestras manuales
        args.MetodoOptimizacion char = 'EI'
        args.ActualizacionHiperparametros (1,2) double = [7, 10]
        args.ExplorationRatio (1,1) double = 0.1
        args.Xi (1,1) double = 0.01
        args.InitialHiperparametros (1,3) double = [0.25, 1, 0.01]
    end

    % Obtener los valores de los argumentos
    maxEval = args.ObjetivosEval;
    muestrasIni = args.MuestrasIniciales;
    type = args.MetodoOptimizacion;
    hiper = args.ActualizacionHiperparametros;
    explorationRatio = args.ExplorationRatio;
    xi = args.Xi;
    initialHiperparametros = args.InitialHiperparametros;

    % Mostrar los valores de los argumentos
    disp(['ObjetivosEval: ', num2str(maxEval)]);
    disp(['MuestrasIniciales: ', num2str(muestrasIni)]);
    disp(['MetodoOptimizacion: ', num2str(type)]);
    disp(['ActualizacionHiperparametros: ', num2str(hiper)]);
    disp(['ExplorationRatio: ', num2str(explorationRatio)]);
    disp(['Xi: ', num2str(xi)]);

    % Si se pasan muestras manuales, ajustar muestrasIni al número de filas de la matriz
    if ~isempty(args.MuestraInicialManual)
        muestrasIni = size(args.MuestraInicialManual, 1);
        disp('Usando muestras iniciales manuales.');
    else
        disp('Generando muestras iniciales aleatorias.');
    end

    % Generar muestras iniciales
```

```
        muestras = table;
        evalTimes = zeros(maxEval, 1); % Vector para guardar tiempos de evaluación
        bestExpectedValues = [];
        minObtainedValues = [];

        % Inicializar minPred para el mejor valor esperado
        minPred = inf;
        minValorObtenido = inf;
        bestParams = [];

        for i = 1:muestrasIni
            nuevaFila = table; % Crear una nueva fila en cada iteración

            if isempty(args.MuestraInicialManual)  % Si no hay muestras manuales, generar
aleatorias
                for j = 1:numel(variables)
                    nombreVar = variables{j}.Name;
                    valorVar = unifrnd(variables{j}.Range(1), variables{j}.Range(2));
                    nuevaFila.(nombreVar) = valorVar;
                end
            else  % Usar las muestras proporcionadas manualmente
                for j = 1:numel(variables)
                    nombreVar = variables{j}.Name;
                    nuevaFila.(nombreVar) = args.MuestraInicialManual(i, j);
                end
            end

            muestras = [muestras; nuevaFila]; % Agregar la nueva fila a la tabla

            % Evaluar la función objetivo y guardar el tiempo de evaluación
            tic;
            valor = funcion(table2struct(nuevaFila));
            evalTimes(i) = toc;

            Data(i).Variables = table2struct(nuevaFila);
            Data(i).Valor = valor;
            Data(i).ValorDesescalado = desnormalize_and_descale(valor, 5, 3, 0, 1);

            % Actualizar el mejor valor esperado
            if valor < minPred
                minPred = valor;
            end

            % Actualizar el mejor valor obtenido
            if valor < minValorObtenido
                minValorObtenido = valor;
                bestParams = table2struct(nuevaFila);
            end

            % Inicializar hiperparámetros para muestras iniciales (arbitrarios)
            Data(i).lengthScale = NaN;
            Data(i).sigmaF = NaN;
            Data(i).sigmaN = NaN;

            bestExpectedValues(end+1) = minPred; % Guardar el mejor valor esperado escalado
            minObtainedValues(end+1) = minValorObtenido; % Guardar el mejor valor obtenido
escalado
        end

        minVal = [inf, 0];
        for i = 1:height(muestras)
            if (Data(i).Valor <= minVal(1))
                minVal = [Data(i).Valor, i];
            end
        end

        % Inicializar tabla para el registro de iteraciones
        iteracion_log = table('Size', [0, numel(variables) + 7], ...
                              'VariableTypes', [repmat("double", 1, numel(variables) + 6),
"string"], ...
                              'VariableNames',   [{'Iteracion'},   variablesName(variables),
{'Objetivo', 'ValorDesescalado', 'lengthScale', 'sigmaF', 'sigmaN', 'Mejor'}]);
```

```
        % Imprimir encabezado de la tabla
        fprintf('%10s', 'Iteracion');
        for j = 1:numel(variables)
            fprintf('%15s', variables{j}.Name);
        end
        fprintf('%15s  %20s  %15s  %15s  %15s  %10s\n',  'Objetivo',  'ValorDesescalado',
'lengthScale', 'sigmaF', 'sigmaN', 'Mejor');

        % Evaluar las muestras iniciales
        for i = 1:muestrasIni
            esMejor = Data(i).Valor == minVal(1);
            esMejorStr = "No";
            if esMejor
                esMejorStr = "Sí";
            end
            nuevaIteracion     =     [i,     table2array(muestras(i,:)),     Data(i).Valor,
Data(i).ValorDesescalado, Data(i).lengthScale, Data(i).sigmaF, Data(i).sigmaN, esMejorStr];
            nuevaIteracionTable     =     array2table(nuevaIteracion,     'VariableNames',
iteracion_log.Properties.VariableNames);
            iteracion_log = [iteracion_log; nuevaIteracionTable];
            % Imprimir fila de la tabla
            fprintf('%10d', i);
            for j = 1:numel(variables)
                fprintf('%15.4f', muestras{i, j});
            end
            fprintf('%15.4f   %20.4f   %15.4f   %15.4f   %15.4f   %10s\n',   Data(i).Valor,
Data(i).ValorDesescalado, Data(i).lengthScale, Data(i).sigmaF, Data(i).sigmaN, esMejorStr);
        end

        for iter = muestrasIni+1:maxEval
            % Normalizar los datos
            X = struct2table([Data.Variables]);
            X = table2array(X);
            Y = [Data.Valor]';

            % Ajustar el modelo de regresión gausiana
            if iter == (muestrasIni+1)
                ini_hiper = 1;
                gpModel       =       hiperparam_actualization(X,       Y,       ini_hiper,
initialHiperparametros(1), ...
                    initialHiperparametros(2), initialHiperparametros(3));
                ini_hiper = 0;
            elseif (mod((iter-(muestrasIni+1)-hiper(1)), hiper(2)) == 0 ...
                    && (iter-(muestrasIni+1)-hiper(1))/hiper(2) > 0)|| iter == (muestrasIni+1
+ hiper(1))
                gpModel       =       hiperparam_actualization(X,       Y,       ini_hiper,
initialHiperparametros(1), ...
                    initialHiperparametros(2), initialHiperparametros(3));
                % Recalcular priormean
                priormean_value           =           recalculate_priormean(X,           Y,
gpModel.KernelInformation.KernelParameters(1) ...
                    , gpModel.KernelInformation.KernelParameters(2));
                priormean = @(x) priormean_value * ones(size(x, 1), 1);   % Redefinir la
función priormean
            end
            gpModel.X = X;
            gpModel.Y = Y;

            % Obtener la próxima muestra utilizando la incertidumbre del modelo de regresión
gausiana
            [nextSample, X_new, estimatedValue] = getNextSample(type, gpModel, variables,
1000, xi, evalTimes, explorationRatio);

            % Evaluar la función objetivo en la próxima muestra
            nuevaFila = array2table(nextSample, 'VariableNames', variablesName(variables));

            % Guardar el tiempo de evaluación y valor funcion objetivo
            tic;
            nuevoValor = funcion(table2struct(nuevaFila));
            evalTimes(iter) = toc;
```

```
            muestras = [muestras; nuevaFila];

            % Actualizar el mejor valor esperado
            minPred = estimatedValue;

            % Actualizar el mejor valor obtenido
            if nuevoValor < minValorObtenido
                minValorObtenido = nuevoValor;
                bestParams = table2struct(nuevaFila);
            end

            % Almacenar los resultados en la estructura de datos de salida
            Data(iter).Variables = table2struct(nuevaFila);
            Data(iter).Valor = nuevoValor;
            Data(iter).ValorDesescalado = desnormalize_and_descale(nuevoValor, 5, 3, 0, 1);
            Data(iter).lengthScale = gpModel.KernelInformation.KernelParameters(1);
            Data(iter).sigmaF = gpModel.KernelInformation.KernelParameters(2);
            Data(iter).sigmaN = gpModel.sigmaN;

            % Registro de iteraciones
            esMejor = Data(iter).Valor < minVal(1);
            esMejorStr = "No";
            if esMejor
                esMejorStr = "Sí";
            end
            nuevaIteracion     =     [iter,     table2array(nuevaFila),     Data(iter).Valor,
Data(iter).ValorDesescalado,  Data(iter).lengthScale,  Data(iter).sigmaF,  Data(iter).sigmaN,
esMejorStr];
            nuevaIteracionTable     =     array2table(nuevaIteracion,     'VariableNames',
iteracion_log.Properties.VariableNames);
            iteracion_log = [iteracion_log; nuevaIteracionTable];

            % Imprimir fila de la tabla
            fprintf('%10d', iter);
            for j = 1:numel(variables)
                fprintf('%15.4f', nextSample(j));
            end
            fprintf('%15.4f    %20.4f    %15.4f    %15.4f    %15.4f    %10s\n',    nuevoValor,
Data(iter).ValorDesescalado,  Data(iter).lengthScale,  Data(iter).sigmaF,  Data(iter).sigmaN,
esMejorStr);

            % Actualizar los valores comparativos
            bestExpectedValues(end+1) = minPred; % Guardar el mejor valor esperado en la
iteración
            minObtainedValues(end+1) = minValorObtenido; % Guardar el mejor valor obtenido

            if esMejor
                minVal = [Data(iter).Valor, iter];
            end

            % Mostrar la visualización de cada iteración
            updateVisualization(gpModel,     variables,     muestras,     Data,     X_new,
bestExpectedValues, minObtainedValues);
        end

        % Visualizar el modelo gausiano final
        updateVisualization(gpModel, variables, muestras, Data, X_new, bestExpectedValues,
minObtainedValues);

        % Mostrar el mejor resultado y los valores de las variables
        mejorIteracion = iteracion_log(minVal(2), :);
        fprintf('\nOptimization completed.\n');
        fprintf('MaxObjectiveEvaluations of %d reached.\n', maxEval);
        fprintf('Total function evaluations: %d\n', maxEval);
        fprintf('Best observed feasible point:\n');
        for j = 1:numel(variables)
            fprintf('%15s', variables{j}.Name);
        end
        fprintf('\n');
        for j = 1:numel(variables)
            fprintf('%15.4f', mejorIteracion{1, variables{j}.Name});
        end
```

```
        fprintf('\n');
        fprintf('Observed objective function value = %.5f\n', mejorIteracion.Objetivo);
        fprintf('Descaled       objective       function       value       =       %.5f\n',
mejorIteracion.ValorDesescalado);
        fprintf('LengthScale = %.5f\n', mejorIteracion.lengthScale);
        fprintf('SigmaF = %.5f\n', mejorIteracion.sigmaF);
        fprintf('SigmaN = %.5f\n', mejorIteracion.sigmaN);

        % Establecer los mejores parámetros en la función de simulación
        funcion(bestParams);
    end
```

2. Función para desnormalizar un valor

Tabla A 2. Función desnormalize_and_descale

```
    function value = desnormalize_and_descale(normalized_value, mean_value, ...
        std_value, mean_final, std_final)
        z = (normalized_value - mean_final) / std_final;
        value = z * std_value + mean_value;
    end
```

3. Función para la actualización de los hiperparámetros

Tabla A 3. Función hiperparam_actualization

```
    function  gpModel  =  hiperparam_actualization(X,  Y,  ini_hiper,  initialLengthScale,
initialSigmaF, initialSigmaN)
        % Inicializar hiperparámetros
        initialParams = [initialLengthScale; initialSigmaF; initialSigmaN];

        % Transformar los hiperparámetros iniciales a la escala logarítmica
        logInitialParams = log(initialParams);

        % Limites inferiores y superiores en logaritmo
        logLb = log([0.1, 0.1, 0.01]);
        logUb = log([1, 10, 1]);

        options = optimset('Display', 'off', 'MaxFunEvals', 10000, 'MaxIter', 10000);

        if ini_hiper
            logtheta = logInitialParams;
        else
            % Optimización en escala logarítmica
            [logtheta, fval, exitflag, output] = fmincon(@(logtheta) nlml(exp(logtheta), X,
Y), logInitialParams, [], [], [], [], logLb, logUb, [], options);
            if exitflag <= 0
                return
            end
        end

        % Transformar los parámetros optimizados de vuelta a la escala original
        theta = exp(logtheta);

        % Extraer hiperparámetros optimizados
        lengthScale = theta(1);
        sigmaF = theta(2);
        sigmaN = theta(3);

        % Guardar el modelo
        gpModel.X = X;
        gpModel.Y = Y;
```

```
        gpModel.KernelInformation.KernelParameters(1) = lengthScale;
        gpModel.KernelInformation.KernelParameters(2) = sigmaF;
        gpModel.sigmaN = sigmaN;
    end
```

4. Función Negative Log Marginal Likelihood

Tabla A 4. Función nlml

```
    function nll = nlml(logtheta, X, Y)
        % Parámetros de regularización para prevenir valores Inf
        epsilon = 1e-6;

        % Hiperparámetros
        lengthScale = logtheta(1);
        sigmaF = logtheta(2);
        sigmaN = logtheta(3);

        % Matriz de covarianza con regularización (jitter)
        K = matern52_kernel(X, X, lengthScale, sigmaF) + (sigmaN^2 + epsilon) * eye(size(X,
1));

        % Verificar si K es definida positiva
        [L, p] = chol(K, 'lower');
        if p > 0
            % Si no es definida positiva, devuelve un número grande
            nll = Inf;
            return;
        end

        alpha = L' \ (L \ Y);

        % Calcular el logaritmo de verosimilitud marginal negativa
        nll = 0.5 * Y' * alpha + sum(log(diag(L))) + 0.5 * size(X, 1) * log(2*pi);

        % Agregar término de regularización para evitar valores Inf
        if isnan(nll) || isinf(nll)
            nll = Inf;
        end
    end
```

5. Función para el cálculo del matern kernel

Tabla A 5. Función matern52_kernel

```
    function K = matern52_kernel(X1, X2, lengthScale, sigmaF)
        % Calcular la distancia euclidiana por pares
        dist = sqrt(sqdist(X1 / lengthScale, X2 / lengthScale));
        % Calcular el kernel Matérn 5/2
        K = sigmaF^2 * (1 + sqrt(5) * dist + 5/3 * dist.^2) .* exp(-sqrt(5) * dist);
    end
```

6. Función para recalcular la media predicha de la función objetivo

Tabla A 6. Función recalculate_priormean

```
    function priormean_value = recalculate_priormean(X, Y, lengthScale, sigmaF)
        n = size(X, 1);  % Número de puntos de entrenamiento

        % Calcular la matriz de covarianza K usando el kernel de Matérn 5/2
        K = matern52_kernel(X, X, lengthScale, sigmaF) + 1e-6 * eye(n);
```

```
        % Descomposición de Cholesky de la matriz de covarianza K
        L = chol(K, 'lower');

        % Resolver para alfa usando la descomposición de Cholesky
        alpha = L' \ (L \ Y);

        % Calcular las ponderaciones basadas en la covarianza
        weights = calculate_weights_based_on_covariance(X, lengthScale, sigmaF);

        % Calcular la media previa utilizando las ponderaciones de Kriging
        priormean_value = sum((K * alpha) .* weights);
    end
```

7. Función para determinar el peso de cada muestra a la hora de recalcular la media

Tabla A 7. Función calculate_weights_based_on_covariance

```
    function weights = calculate_weights_based_on_covariance(X, lengthScale, sigmaF)
        % Calcular la matriz de covarianza utilizando el kernel de Matérn 5/2
        K = matern52_kernel(X, X, lengthScale, sigmaF);

        % Calcular las sumas de cada fila de la matriz de covarianza
        row_sums = sum(K, 2);

        % Invertir las sumas para obtener las ponderaciones
        % Añadir un pequeño valor (1e-6) para evitar divisiones por cero
        weights = 1 ./ (row_sums + 1e-6);

        % Normalizar las ponderaciones para que sumen 1
        weights = weights / sum(weights);
    end
```

8. Función con la que se actualiza el valor de la media de la función objetivo

Tabla A 8. Función priormean

```
    function m = priormean(x)
        m = 0 * ones(size(x, 1), 1);
    end
```

9. Función para determinar el nuevo punto a evaluar

Tabla A 9. Función getNextSample

```
    function [nextSample, X_new, estimatedValue] = getNextSample(type, gpModel, variables,
num_points, xi, evalTimes, explorationRatio)
        % Definir un rango de valores de entrada para hacer predicciones
        X_new = zeros(num_points, numel(variables));
        for j = 1:numel(variables)
            range = variables{j}.Range;
            X_new(:, j) = unifrnd(range(1), range(2), num_points, 1);
        end

        % Obtener las predicciones del modelo en el nuevo conjunto de datos de entrada
        [Y_pred, Y_pred_sd] = predGP(gpModel, X_new);

        % Calcular la función de adquisición según el tipo
        Y_best = min(gpModel.Y);
        switch type
            case 'EI'
```

```
                acquisitionValue = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
            case 'EI+'
                acquisitionValue = expected_improvement_plus(Y_best, Y_pred, Y_pred_sd ...
                    , xi, explorationRatio, gpModel.sigmaN);
            case 'EIPS'
                acquisitionValue = expected_improvement_per_second(Y_best, Y_pred ...
                    , Y_pred_sd, xi, evalTimes);
            case 'EIPS+'
                acquisitionValue = expected_improvement_per_second_plus(Y_best, Y_pred ...
                    , Y_pred_sd, xi, evalTimes, explorationRatio, gpModel.sigmaN);
            case 'LCB'
                acquisitionValue = lower_confidence_bound(Y_pred, Y_pred_sd, xi);
            case 'PI'
                acquisitionValue  =  probability_of_improvement(Y_best,  Y_pred,  Y_pred_sd,
xi);
            otherwise
                error('Tipo de función de adquisición no reconocido.');
        end

        % Seleccionar el próximo punto a probar
        [~, idx] = max(acquisitionValue);
        initialSample = X_new(idx, :);

        % Definir límites para fmincon
        lb = cellfun(@(var) var.Range(1), variables);
        ub = cellfun(@(var) var.Range(2), variables);

        % Usar fmincon para buscar puntos cerca del óptimo estimado
        options = optimoptions('fmincon', 'Display', 'off', 'MaxIterations', ...
            1000, 'MaxFunctionEvaluations', 3000, 'OptimalityTolerance', 1e-6);
        fun  =  @(X)  optimization_function(X,  Y_best,  gpModel,  type,  xi,  explorationRatio,
evalTimes);
        [nextSample, ~] = fmincon(fun, initialSample, [], [], [], [], lb, ub, [], options);

        % Valor estimado en el próximo punto
        [estimatedValue, ~] = predGP(gpModel, nextSample);
    end
```

10. Función utilizada para realizar predicciones basadas en el modelo de proceso gaussiano, calculando la media y la incertidumbre para los puntos de entrada dados

Tabla A 10. Función predGP

```
    function [Y_pred, Y_pred_sd] = predGP(gpModel, X_new)
        % Recuperar el modelo
        X = gpModel.X;
        Y = gpModel.Y;
        lengthScale = gpModel.KernelInformation.KernelParameters(1);
        sigmaF = gpModel.KernelInformation.KernelParameters(2);
        sigmaN = gpModel.sigmaN;

        % Calcular la covarianza
        K = matern52_kernel(X, X, lengthScale, sigmaF) + (sigmaN^2 + 1e-6) * eye(size(X, 1));
        K_s = matern52_kernel(X, X_new, lengthScale, sigmaF);
        K_ss = matern52_kernel(X_new, X_new, lengthScale, sigmaF) + 1e-6 * eye(size(X_new,
1));

        % Verificar si K es positiva definida usando Cholesky
        [L, p] = chol(K, 'lower');
        if p > 0
            error('La matriz de covarianza no es positiva definida.');
        end

         % Resolver para alfa
        %alpha_1 = L' \ (L \ Y);
        alpha = L' \ (L \ (Y - priormean(X)));
```

```
        % Predicción de la media
        %Y_pred_1 = K_s' * alpha_1;
        Y_pred = priormean(X_new) + K_s' * alpha;

        % Predicción de la varianza
        v = L \ K_s;
        Y_pred_var = K_ss - v' * v;

        % Verificar si hay valores negativos en Y_pred_var
        if any(diag(Y_pred_var) < 0)
            warning('Se encontraron valores negativos en la matriz de varianza.');
        end

        % Predicción de la desviación estándar
        Y_pred_sd = sqrt(diag(Y_pred_var));
    end
```

11. Función para determinar el Expected Improvement

Tabla A 11. Función expected_improvement

```
    function EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi)
        Z = (Y_best*(1-xi) - Y_pred) ./ Y_pred_sd;
        EI = (Y_best*(1-xi) - Y_pred) .* normcdf(Z) + Y_pred_sd .* normpdf(Z);
    end
```

12. Función para determinar el Expected Improvement plus

Tabla A 12. Función expected_improvement_plus

```
    function EI_plus = expected_improvement_plus(Y_best, Y_pred, Y_pred_sd, ...
        xi, explorationRatio, sigmaN)

        sigma_Q = sqrt(Y_pred_sd.^2 + sigmaN^2);
        overExploited = sigma_Q < explorationRatio * sigmaN;

        EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);

        % Incrementar EI en regiones sobreexplotadas
        EI_plus = EI;
        EI_plus(overExploited) = EI_plus(overExploited) * 10;
    end
```

13. Función para determinar el Expected Improvement por segundo

Tabla A 13. Función expected_improvement_per_second

```
    function EIPS = expected_improvement_per_second(Y_best, Y_pred, Y_pred_sd, xi, evalTimes)
        EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
        EIPS = EI ./ evalTimes;
    end
```

14. Función para determinar el Expected Improvement plus por segundo

Tabla A 14. Función expected_improvement_per_second_plus

```
    function EIPS_plus = expected_improvement_per_second_plus(Y_best, Y_pred, Y_pred_sd, xi,
evalTimes, explorationRatio, sigmaN)
        sigma_Q = sqrt(Y_pred_sd.^2 + sigmaN^2);
        overExploited = sigma_Q < explorationRatio * sigmaN;

        EI = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
        EIPS = EI ./ evalTimes;

        % Incrementar EIPS en regiones sobreexplotadas
        EIPS_plus = EIPS;
        EIPS_plus(overExploited) = EIPS_plus(overExploited) * 10;
    end
```

15. Función para determinar el Lower Confidence Bound

Tabla A 15. Función lower_confidence_bound

```
    function LCB = lower_confidence_bound(Y_pred, Y_pred_sd, xi)
        LCB = Y_pred - (xi * Y_pred) .* Y_pred_sd;
    end
```

16. Función para determinar el Probability of Improvement

Tabla A 16. Función probability_of_improvement

```
    function PI = probability_of_improvement(Y_best, Y_pred, Y_pred_sd, xi)
        Z = (Y_best * (1 - xi) - Y_pred) ./ Y_pred_sd;
        PI = normcdf(Z);
    end
```

17. Función objetivo utilizada por el algoritmo de optimización fmincon en getNextSample

Tabla A 17. Función optimization_function

```
    function acquisitionValue = optimization_function(X, Y_best, gpModel, type, xi,
explorationRatio, evalTimes)
        [Y_pred, Y_pred_sd] = predGP(gpModel, X);
        switch type
            case 'EI'
                acquisitionValue = -expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
            case 'EI+'
                acquisitionValue = -expected_improvement_plus(Y_best, Y_pred, Y_pred_sd, xi,
explorationRatio, gpModel.sigmaN);
            case 'EIPS'
                acquisitionValue = -expected_improvement_per_second(Y_best, Y_pred,
Y_pred_sd, xi, evalTimes);
            case 'EIPS+'
                acquisitionValue = -expected_improvement_per_second_plus(Y_best, Y_pred,
Y_pred_sd, xi, evalTimes, explorationRatio, gpModel.sigmaN);
            case 'LCB'
                acquisitionValue = lower_confidence_bound(Y_pred, Y_pred_sd, xi);
            case 'PI'
                acquisitionValue = -probability_of_improvement(Y_best, Y_pred, Y_pred_sd,
xi);
            otherwise
                error('Tipo de función de adquisición no reconocido.');
        end
    end
```

18. Función que actualiza visualizaciones gráficas para comparar valores esperados y obtenidos, y ajusta el modelo gaussiano a las muestras, mostrando predicciones y márgenes de incertidumbre en visualizaciones 2D o 3D según el número de variables.

Tabla A 18. Función updateVisualization

```
    function    updateVisualization(gpModel,    variables,    muestras,    Data,    X_new,
bestExpectedValues, minObtainedValues)

        num_points = 1000;
        num_vars = numel(variables);

        % Comparativa de valores mínimos y mínimos esperados
        fig1 = findobj('Type', 'Figure', 'Name', 'Comparativa de Valores Mínimos');
        if isempty(fig1)
            fig1 = figure('Name', 'Comparativa de Valores Mínimos', 'NumberTitle', 'off');
        else
            figure(fig1);
        end
        clf; % Limpiar la figura
        hold on;
        plot(bestExpectedValues,  'b-',  'LineWidth',  2,  'DisplayName',  'Mejor  Valor
Esperado');
        plot(minObtainedValues,  'r-',  'LineWidth',  2,  'DisplayName',  'Valor  Mínimo
Obtenido');
        xlabel('Iteración');
        ylabel('Valor');
        title('Comparativa del Mejor Valor Esperado y el Valor Mínimo Obtenido');
        legend('show');
        hold off;
        drawnow;

        % Si hay más de dos variables, no graficar nada más
        if num_vars > 2
            return;
        end

        % Modelo Gaussiano Ajustado
        fig2 = findobj('Type', 'Figure', 'Name', 'Modelo Gaussiano Ajustado');
        if isempty(fig2)
            fig2 = figure('Name', 'Modelo Gaussiano Ajustado', 'NumberTitle', 'off');
        else
            figure(fig2);
        end
        clf; % Limpiar la figura
        hold on;

        % Dibujar las muestras actuales con sus valores medidos
        X_samples = table2array(muestras);
        Y_samples = [Data.Valor]';

        % Normalizar los puntos nuevos
        [Y_pred, Y_pred_sd] = predGP(gpModel, X_new);

        % Combinar puntos conocidos y nuevos
        X_combined = [X_samples; X_new];
        Y_combined = [Y_samples; Y_pred];
        Y_combined_sd = [zeros(size(Y_samples)); Y_pred_sd]; % Incertidumbre cero para puntos
conocidos

        if num_vars == 1
            scatter(X_samples, Y_samples, 'ro'); % Puntos medidos exactos

            % Dibujar el proceso esperado y márgenes de incertidumbre
            plot(X_combined, Y_combined, 'b-', 'LineWidth', 2);
```

```
            fill([X_combined;  flipud(X_combined)],  [Y_combined  -  2  *  Y_combined_sd;
flipud(Y_combined + 2 * Y_combined_sd)], 'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');
            xlabel(variables{1}.Name);
            ylabel('Valor');
            title('Modelo Gausiano Ajustado');
            legend('Muestras', 'Predicción Media', 'Márgenes de Incertidumbre');

        elseif num_vars == 2
            % Dibujar los puntos medidos exactos como círculos rojos vacíos
            scatter3(X_samples(:,1), X_samples(:,2), Y_samples, 'ro', 'filled');

            % Crear una malla para las predicciones
            [X1,  X2]  =  meshgrid(linspace(variables{1}.Range(1),  variables{1}.Range(2),
sqrt(num_points)),          linspace(variables{2}.Range(1),          variables{2}.Range(2),
sqrt(num_points)));
            X_new_grid = [X1(:), X2(:)];
            [Y_pred_grid, Y_pred_sd_grid] = predGP(gpModel, X_new_grid);
            Y_pred_grid = reshape(Y_pred_grid, size(X1));
            Y_pred_sd_grid = reshape(Y_pred_sd_grid, size(X1));

            % Superficie de predicción
            surf(X1, X2, Y_pred_grid, 'EdgeColor', 'none', 'FaceAlpha', 0.5); % Eje Z es la
predicción
            % Márgenes de incertidumbre
            surf(X1, X2, Y_pred_grid - 2 * Y_pred_sd_grid, 'FaceAlpha', 0.2, 'EdgeColor',
'none'); % Márgenes inferiores
            surf(X1, X2, Y_pred_grid + 2 * Y_pred_sd_grid, 'FaceAlpha', 0.2, 'EdgeColor',
'none'); % Márgenes superiores

            % Añadir puntos simulados como círculos rojos vacíos
            scatter3(X_samples(:,1), X_samples(:,2), Y_samples, 'ro', 'filled');

            xlabel(variables{1}.Name);
            ylabel(variables{2}.Name);
            zlabel('Valor'); % Eje Z para el valor
            title('Modelo Gausiano Ajustado');
            legend('Muestras', 'Predicción Media', 'Márgenes de Incertidumbre');
            view(3); % Establecer vista 3D
            rotate3d on; % Activar rotación 3D interactiva
        end

        hold off;
        drawnow;
    end
```

ANEXO I.2. FUNCIONES SAFEOPTBAYES

Esta sección incluye la función SafeOptBayes, que incorpora restricciones de seguridad en el proceso de optimización. El enfoque SafeOpt asegura que las evaluaciones no pongan en riesgo la estabilidad del sistema, explorando de manera segura dentro de los límites definidos.

1. Función de optimización con restricción de seguridad

Tabla A 19. Función safeoptbayes

```
    function [Data] = safeoptbayes(funcion, variables, args)
        arguments
            funcion function_handle
            variables cell {mustBeNonempty}
            args.ObjetivosEval (1,1) double = 100
            args.MuestrasIniciales (1,1) double = 5
            args.MuestraInicialManual double = []
            args.MetodoOptimizacion char = 'EI'
            args.MetodoSafe char = 'Upper'
            args.ActualizacionHiperparametros (1,2) double = [7, 10]
            args.ExplorationRatio (1,1) double = 0.1
```

```
            args.Xi (1,1) double = 0.01
            args.SafeValue (1,1) double = 1
            args.InitialHiperparametros (1,3) double = [0.25, 1, 0.01]
            args.InitialSafeHiperparametros (1,3) double = [0.25, 1, 0.01]
        end

        % Obtener los valores de los argumentos
        maxEval = args.ObjetivosEval;
        muestrasIni = args.MuestrasIniciales;
        muestrasManual = args.MuestraInicialManual;
        type = args.MetodoOptimizacion;
        hiper = args.ActualizacionHiperparametros;
        explorationRatio = args.ExplorationRatio;
        xi = args.Xi;
        safeValue = args.SafeValue;
        safeMethod = args.MetodoSafe;
        initialHiperparametros = args.InitialHiperparametros;
        initialSafeHiperparametros = args.InitialSafeHiperparametros;

        % Si el usuario proporciona muestras iniciales manuales, actualizar muestrasIni
        if ~isempty(muestrasManual)
            muestrasIni = size(muestrasManual, 1);   % Actualizar el número de muestras
iniciales con el número de filas de la matriz proporcionada
        end

        % Mostrar los valores de los argumentos
        disp(['ObjetivosEval: ', num2str(maxEval)]);
        disp(['MuestrasIniciales: ', num2str(muestrasIni)]);
        disp(['MetodoOptimizacion: ', num2str(type)]);
        disp(['ActualizacionHiperparametros: ', num2str(hiper)]);
        disp(['ExplorationRatio: ', num2str(explorationRatio)]);
        disp(['Xi: ', num2str(xi)]);
        disp(['SafeValue: ', num2str(safeValue)]);

        % Generar muestras iniciales
        muestras = table;
        evalTimes = zeros(maxEval, 1); % Vector para guardar tiempos de evaluación
        bestExpectedValues = [];
        minObtainedValues = [];

        % Inicializar minPred para el mejor valor esperado
        minPred = inf;
        minValorObtenido = inf;
        bestParams = [];

        for i = 1:muestrasIni
            nuevaFila = table; % Crear una nueva fila en cada iteración

            if isempty(muestrasManual)
                % Si no se proporciona una muestra manual, generar aleatoriamente
                for j = 1:numel(variables)
                    nombreVar = variables{j}.Name;
                    valorVar = unifrnd(variables{j}.Range(1), variables{j}.Range(2));
                    nuevaFila.(nombreVar) = valorVar;
                end
            else
                % Usar la muestra manual proporcionada
                for j = 1:numel(variables)
                    nombreVar = variables{j}.Name;
                    valorVar = muestrasManual(i, j);
                    nuevaFila.(nombreVar) = valorVar;
                end
            end

            muestras = [muestras; nuevaFila]; % Agregar la nueva fila a la tabla

            % Evaluar la función objetivo y guardar el tiempo de evaluación
            tic;
            [valor, valorRes] = funcion(table2struct(nuevaFila));
            evalTimes(i) = toc;

            Data(i).Variables = table2struct(nuevaFila);
```

```
            Data(i).Valor = valor;
            Data(i).ValorDesescalado = desnormalize_and_descale(valor,5, 3, 0, 1);
            Data(i).ValorRes = valorRes;

            % Actualizar el mejor valor esperado
            if valor < minPred
                minPred = valor;
            end

            % Actualizar el mejor valor obtenido
            if valor < minValorObtenido && valorRes <= safeValue
                minValorObtenido = valor;
                bestParams = table2struct(nuevaFila);
            end

            % Inicializar hiperparámetros para muestras iniciales (arbitrarios)
            Data(i).lengthScale = NaN;
            Data(i).sigmaF = NaN;
            Data(i).sigmaN = NaN;

            bestExpectedValues(end+1) = minPred; % Guardar el mejor valor esperado escalado
            minObtainedValues(end+1) = minValorObtenido; % Guardar el mejor valor obtenido
escalado
        end

        minVal = [inf, 0];
        for i = 1:height(muestras)
            if (Data(i).Valor < minVal(1)) && (Data(i).ValorRes <= safeValue)
                minVal = [Data(i).Valor, i];
            end
        end

        % Inicializar tabla para el registro de iteraciones
        iteracion_log = table('Size', [0, numel(variables) + 8], ...
                              'VariableTypes', [repmat("double", 1, numel(variables) + 7),
"string"], ...
                              'VariableNames',   [{'Iteracion'},   variablesName(variables),
{'Objetivo',   'ValorDesescalado','ValorRestriccion',   'lengthScale',   'sigmaF',   'sigmaN',
'Mejor'}]);

        % Imprimir encabezado de la tabla
        fprintf('%10s', 'Iteracion');
        for j = 1:numel(variables)
            fprintf('%15s', variables{j}.Name);
        end
        fprintf('%15s     %20s     %15s     %15s     %15s     %15s     %10s\n',     'Objetivo',
'ValorDesescalado','ValorRestriccion', 'lengthScale', 'sigmaF', 'sigmaN', 'Mejor');

        % Evaluar las muestras iniciales
        for i = 1:muestrasIni
            esMejor = Data(i).Valor <= minVal(1) && Data(i).ValorRes <= safeValue;
            esMejorStr = "No";
            if esMejor
                esMejorStr = "Sí";
            end
            nuevaIteracion     =     [i,     table2array(muestras(i,:)),     Data(i).Valor,
Data(i).ValorDesescalado,     Data(i).ValorRes,     Data(i).lengthScale,     Data(i).sigmaF,
Data(i).sigmaN, esMejorStr];
            nuevaIteracionTable     =     array2table(nuevaIteracion,     'VariableNames',
iteracion_log.Properties.VariableNames);
            iteracion_log = [iteracion_log; nuevaIteracionTable];
            % Imprimir fila de la tabla
            fprintf('%10d', i);
            for j = 1:numel(variables)
                fprintf('%15.4f', muestras{i, j});
            end
            fprintf('%15.4f  %20.4f  %15.4f  %15.4f  %15.4f  %15.4f  %10s\n',  Data(i).Valor,
Data(i).ValorDesescalado,     Data(i).ValorRes,     Data(i).lengthScale,     Data(i).sigmaF,
Data(i).sigmaN, esMejorStr);
        end

        for iter = muestrasIni+1:maxEval
```

```
            % Normalizar los datos
            X = struct2table([Data.Variables]);
            X = table2array(X);
            Y = [Data.Valor]';
            YRes = [Data.ValorRes]';

            % Ajustar el modelo de regresión gausiana
            if iter == (muestrasIni+1)
                ini_hiper = 1;
                gpModel       =       hiperparam_actualization(X,       Y,       ini_hiper,
initialHiperparametros(1) ...
                    , initialHiperparametros(2), initialHiperparametros(3));
                gpModelRes      =      hiperparam_actualization(X,      YRes,      ini_hiper,
initialSafeHiperparametros(1) ...
                    , initialSafeHiperparametros(2), initialSafeHiperparametros(3));
                ini_hiper = 0;
            elseif (mod((iter-(muestrasIni+1)-hiper(1)), hiper(2)) == 0 && ...
                    (iter-(muestrasIni+1)-hiper(1))/hiper(2) > 0)|| iter == (muestrasIni+1 +
hiper(1))
                gpModel       =       hiperparam_actualization(X,       Y,       ini_hiper,
initialHiperparametros(1), ...
                    initialHiperparametros(2), initialHiperparametros(3));
                gpModelRes      =      hiperparam_actualization(X,      YRes,      ini_hiper,
initialSafeHiperparametros(1), ...
                    initialSafeHiperparametros(2), initialSafeHiperparametros(3));
                % Recalcular priormean para objetivo
                priormean_value           =           recalculate_priormean(X,           Y,
gpModel.KernelInformation.KernelParameters(1), ...
                    gpModel.KernelInformation.KernelParameters(2));
                priormean = @(x) priormean_value * ones(size(x, 1), 1);   % Redefinir la
función priormean
                 % Recalcular priormean para restricción
                priormean_valueRes         =         recalculate_priormean(X,         YRes,
gpModelRes.KernelInformation.KernelParameters(1), ...
                    gpModelRes.KernelInformation.KernelParameters(2));
                priormeanRes = @(x) priormean_valueRes * ones(size(x, 1), 1);  % Redefinir
la función priormean
            end
            gpModel.X = X;
            gpModel.Y = Y;
            gpModelRes.X = X;
            gpModelRes.Y = YRes;

            % Obtener la próxima muestra utilizando la incertidumbre del modelo de regresión
gausiana
            [nextSample,      X_new,      estimatedValue]      =      getNextSafeSample(type,
gpModel,gpModelRes, variables, 1000, xi, evalTimes, explorationRatio, safeValue, safeMethod);

            % Evaluar la función objetivo en la próxima muestra
            nuevaFila = array2table(nextSample, 'VariableNames', variablesName(variables));

            % Guardar el tiempo de evaluación
            tic;
            [nuevoValor, nuevoValorRes] = funcion(table2struct(nuevaFila));
            evalTimes(iter) = toc;

            muestras = [muestras; nuevaFila];

            % Actualizar el mejor valor esperado
            minPred = estimatedValue;

            % Actualizar el mejor valor obtenido
            if nuevoValor < minValorObtenido && nuevoValorRes <= safeValue
                minValorObtenido = nuevoValor;
                bestParams = table2struct(nuevaFila);
            end

            % Almacenar los resultados en la estructura de datos de salida
            Data(iter).Variables = table2struct(nuevaFila);
            Data(iter).Valor = nuevoValor;
            Data(iter).ValorDesescalado = desnormalize_and_descale(nuevoValor, 5, 3, 0, 1);
            Data(iter).ValorRes = nuevoValorRes;
```

```
            Data(iter).lengthScale = gpModel.KernelInformation.KernelParameters(1);
            Data(iter).sigmaF = gpModel.KernelInformation.KernelParameters(2);
            Data(iter).sigmaN = gpModel.sigmaN;

            % Registro de iteraciones
            esMejor = Data(iter).Valor < minVal(1) && Data(iter).ValorRes <= safeValue;
            esMejorStr = "No";
            if esMejor
                esMejorStr = "Sí";
                minVal = [Data(iter).Valor, iter];
            end
            nuevaIteracion = [iter, nextSample, nuevoValor, Data(iter).ValorDesescalado,
nuevoValorRes, Data(iter).lengthScale, Data(iter).sigmaF, Data(iter).sigmaN, esMejorStr];
            nuevaIteracionTable = array2table(nuevaIteracion, 'VariableNames',
iteracion_log.Properties.VariableNames);
            iteracion_log = [iteracion_log; nuevaIteracionTable];

            % Imprimir fila de la tabla
            fprintf('%10d', iter);
            for j = 1:numel(variables)
                fprintf('%15.4f', nextSample(j));
            end
            fprintf('%15.4f  %20.4f  %15.4f  %15.4f  %15.4f  %15.4f  %10s\n', nuevoValor,
Data(iter).ValorDesescalado,  nuevoValorRes,  Data(iter).lengthScale,  Data(iter).sigmaF,
Data(iter).sigmaN, esMejorStr);

            % Actualizar los valores comparativos
            bestExpectedValues(end+1) = minPred; % Guardar el mejor valor esperado en la
iteración
            minObtainedValues(end+1) = minValorObtenido; % Guardar el mejor valor obtenido

            % Mostrar la visualización de cada iteración
            updateVisualization(gpModel, variables, muestras, Data, X_new,
bestExpectedValues, minObtainedValues);
        end

        % Visualizar el modelo gausiano final
        updateVisualization(gpModel, variables, muestras, Data, X_new, bestExpectedValues,
minObtainedValues);

        % Mostrar el mejor resultado y los valores de las variables
        mejorIteracion = iteracion_log(minVal(2), :);
        fprintf('\nOptimization completed.\n');
        fprintf('MaxObjectiveEvaluations of %d reached.\n', maxEval);
        fprintf('Total function evaluations: %d\n', maxEval);
        fprintf('Best observed feasible point:\n');
        for j = 1:numel(variables)
            fprintf('%15s', variables{j}.Name);
        end
        fprintf('\n');
        for j = 1:numel(variables)
            fprintf('%15.4f', mejorIteracion{1, variables{j}.Name});
        end
        fprintf('\n');
        fprintf('Observed objective function value = %.5f\n', mejorIteracion.Objetivo);
        fprintf('Descaled objective function value = %.5f\n',
mejorIteracion.ValorDesescalado);
        fprintf('LengthScale = %.5f\n', mejorIteracion.lengthScale);
        fprintf('SigmaF = %.5f\n', mejorIteracion.sigmaF);
        fprintf('SigmaN = %.5f\n', mejorIteracion.sigmaN);

        % Establecer los mejores parámetros en la función de simulación
        funcion(bestParams);
    end
```

2. Función para determinar el nuevo punto a evaluar teniendo en cuenta las restricciones de seguridad

Tabla A 20. Función getNextSafeSample

```
    function [nextSample, X_new, estimatedValue] = getNextSafeSample(type, gpModel,
gpModelRes, variables, num_points, xi, evalTimes, explorationRatio, safeValue, safeMethod)
        % Definir un rango de valores de entrada para hacer predicciones
        X_new = zeros(num_points, numel(variables));
        for j = 1:numel(variables)
            range = variables{j}.Range;
            X_new(:, j) = unifrnd(range(1), range(2), num_points, 1);
        end

        % Obtener las predicciones del modelo en el nuevo conjunto de datos de entrada
        [Y_pred, Y_pred_sd] = predGP(gpModel, X_new);
        [YRes_pred, YRes_pred_sd] = predGP(gpModelRes, X_new);
        safe_vector = YRes_pred + 2 * YRes_pred_sd - safeValue;

        % Calcular la función de adquisición según el tipo
        Y_best = min(gpModel.Y);
        switch type
            case 'EI'
                acquisitionValue = expected_improvement(Y_best, Y_pred, Y_pred_sd, xi);
            case 'EI+'
                acquisitionValue = expected_improvement_plus(Y_best, Y_pred, Y_pred_sd, xi,
explorationRatio, gpModel.sigmaN);
            case 'EIPS'
                acquisitionValue    =    expected_improvement_per_second(Y_best,    Y_pred,
Y_pred_sd, xi, evalTimes);
            case 'EIPS+'
                acquisitionValue  =  expected_improvement_per_second_plus(Y_best,  Y_pred,
Y_pred_sd, xi, evalTimes, explorationRatio, gpModel.sigmaN);
            case 'LCB'
                acquisitionValue = -lower_confidence_bound(Y_pred, Y_pred_sd, xi);
            case 'PI'
                acquisitionValue  =  probability_of_improvement(Y_best,  Y_pred,  Y_pred_sd,
xi);
            otherwise
                error('Tipo de función de adquisición no reconocido.');
        end

        if strcmp(safeMethod, 'Upper')
            for i = 1:num_points
                if safe_vector(i) >= 0
                    acquisitionValue(i) = -inf;
                end
            end
        elseif strcmp(safeMethod, 'Lower')
            for i = 1:num_points
                if safe_vector(i) <= 0
                    acquisitionValue(i) = -inf;
                end
            end
        end

        % Seleccionar el próximo punto a probar
        [~, idx] = max(acquisitionValue);
        initialSample = X_new(idx, :);

        % Definir la función objetivo para fmincon
        fun = @(X) optimization_function(X, Y_best, gpModel, type, xi, explorationRatio,
evalTimes);

        % Definir las restricciones de seguridad para fmincon
        nonlcon = @(X) safety_constraints(X, gpModelRes, safeValue, safeMethod);
```

```
        % Crear los límites inferiores y superiores
        lb = cellfun(@(var) var.Range(1), variables);
        ub = cellfun(@(var) var.Range(2), variables);

        % Usar fmincon para buscar puntos cerca del óptimo estimado
        options = optimoptions('fmincon', 'Display', 'off', 'MaxIterations', 1000, ...
            'MaxFunctionEvaluations', 3000, 'OptimalityTolerance', 1e-6);
        [nextSample, ~] = fmincon(fun, initialSample, [], [], [], [], lb, ub, ...
            nonlcon, options);

        % Valor estimado en el próximo punto
        [estimatedValue, ~] = predGP(gpModel, nextSample);
    end
```

3. Función define las restricciones de seguridad no lineales para la búsqueda final de getNextSafeSample

Tabla A 21. Función safety_constraints

```
    function [c, ceq] = safety_constraints(X, gpModelRes, safeValue, safeMethod)
        [YRes_pred, YRes_pred_sd] = predGP(gpModelRes, X);
        if strcmp(safeMethod, 'Upper')
            c = YRes_pred + 2 * YRes_pred_sd - safeValue;  % Restricción no lineal
        elseif strcmp(safeMethod, 'Lower')
            c = safeValue - (YRes_pred - 2 * YRes_pred_sd);  % Restricción no lineal
        else
            c = [];
        end
        ceq = [];  % No hay restricciones de igualdad
    end
```

ANEXO II: VALIDACIÓN DE OPTBAYES Y SAFEOPTBAYES

Este anexo está dedicado a la validación de los algoritmos de optimización OptBayes y SafeOptBayes desarrollados a lo largo del proyecto. A través de pruebas exhaustivas en un controlador PI, tanto con cómo sin restricciones de seguridad, se evaluó el rendimiento de los optimizadores, con el objetivo de determinar su capacidad para encontrar configuraciones óptimas de los parámetros de control con un número reducido de evaluaciones.

El contenido de este anexo incluye los resultados obtenidos en las pruebas realizadas, detallando tanto las configuraciones iniciales como las soluciones encontradas por cada método. Se incluyen también comparativas con la prueba maestra, que evalúa 2500 puntos en el espacio de búsqueda. Finalmente, las figuras y tablas incluidas proporcionan una visión visual del proceso de optimización y validación.

ANEXO II.1. PRUEBA MAESTRA DE VALIDACIÓN

Tras realizar las correcciones necesarias utilizando el proceso simple, se decidió llevar a cabo una validación más exhaustiva de las funciones OptBayes y SafeOptBayes, a través de lo que denominamos una "prueba maestra". Esta prueba tenía como objetivo asegurar que la versión final de las funciones devolvería la mejor opción o una muy próxima a esta.

Para ello, se evaluaron 2,500 puntos dentro del espacio de búsqueda de los parámetros del controlador PI, combinando diferentes valores de KP y KI. Básicamente, se exploraron casi todas las posibles combinaciones de estos parámetros dentro de un rango escalado para encontrar las mejores soluciones tanto con restricciones como sin ellas.

El procedimiento fue el siguiente: se crearon dos matrices, *Valor* y *ValorRes*, que registran, respectivamente, el rendimiento del controlador y el cumplimiento de la restricción de seguridad (que la acción de control no exceda un límite predefinido). Para cada una de las 2,500 combinaciones posibles de KP y KI, se calculó el valor de la función objetivo y de la restricción usando la función *Safe_sim_PI*, función que simula el modelo del proceso llevado a cabo en Simulink y devuelve el valor de la función objetivo y el valor máximo que toma la acción de control en dicha simulación.

Una vez completadas llenas dichas matrices, se procede a identificar el mejor resultado posible en términos de rendimiento puro, es decir, sin considerar la restricción de seguridad. A continuación, se filtraron los resultados para identificar la mejor solución que también respetaba la restricción de seguridad (ValorRes < 2). Este proceso permitió comparar directamente las soluciones óptimas teóricas con las obtenidas a través de las funciones de optimización desarrolladas.

Para visualizar y entender mejor los resultados, se generaron gráficos en 3D que muestran cómo varía el rendimiento en el espacio de búsqueda, junto con gráficos en 2D que resaltan la zona segura donde se cumplen las restricciones. Estas visualizaciones facilitan la comparación entre las soluciones óptimas teóricas y las proporcionadas por OptBayes y SafeOptBayes, permitiendo así validar si el funcionamiento de las funciones implementadas es adecuado.

Resultados de la Prueba Maestra

En la siguiente imagen se muestra la superficie del rendimiento del controlador PI (Valor) en función de las combinaciones de KP y KI. Esta gráfica en 3D permite observar cómo varía el rendimiento del sistema en el espacio de búsqueda. Las zonas más altas representan un peor rendimiento, mientras que las zonas más bajas indican las áreas donde el controlador opera de manera óptima.

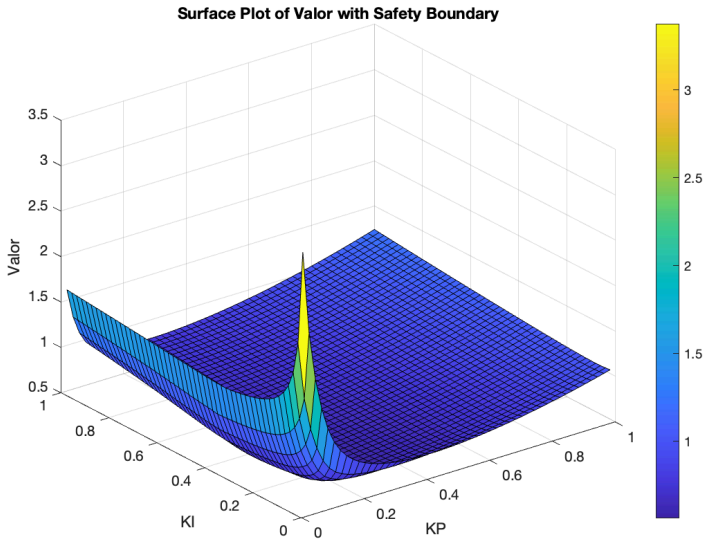


Figura A. 1. Superficie de rendimiento del controlador PI en función de KP y KI

En la imagen inferior se presenta una vista en 2D en planta, donde se destacan las áreas seguras donde se cumple la restricción de seguridad. El área sombreada en la imagen representa la región del espacio de búsqueda que respeta las condiciones de seguridad, proporcionando una visión clara de las zonas que son tanto seguras como eficientes en términos de rendimiento.

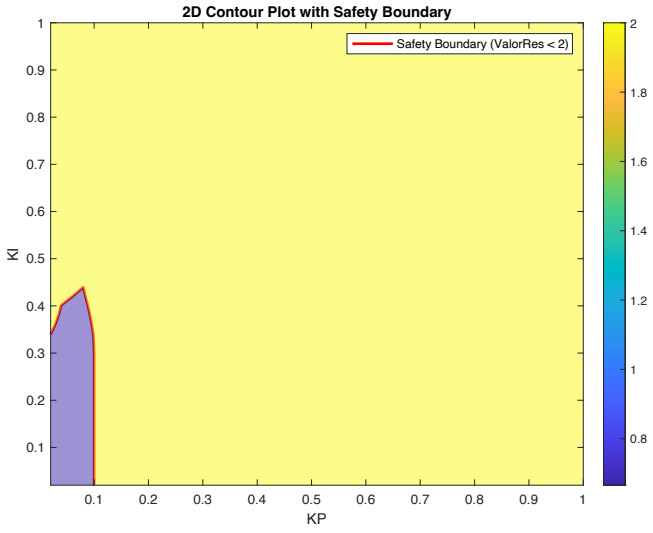


Figura A. 2. Mapa de calor del rendimiento del controlador PI con límite de seguridad (ValorRes < 2)

Después de analizar la vista en planta 2D, se observó que el área segura en el espacio de búsqueda era bastante reducida. Para explorar esta región con mayor detalle, se decidió realizar un análisis más concentrado. Se generaron otros 2500 puntos dentro de esta zona segura, utilizando una malla más fina que permitió obtener una visión más precisa de las posibles soluciones óptimas bajo las restricciones de seguridad.

El procedimiento para este análisis detallado fue el siguiente: se dividió la región segura en una malla más densa, escalando las variables KP y KI en un rango más pequeño, específicamente en [0, 0.1] para KP y [0, 0.5] para KI. Este enfoque permitió una mejor exploración de la zona segura y resultó en un gráfico con mucho más detalle, lo que facilitó la identificación de la mejor configuración del controlador que cumple con las restricciones impuestas.

A continuación, se presenta la imagen del gráfico resultante:

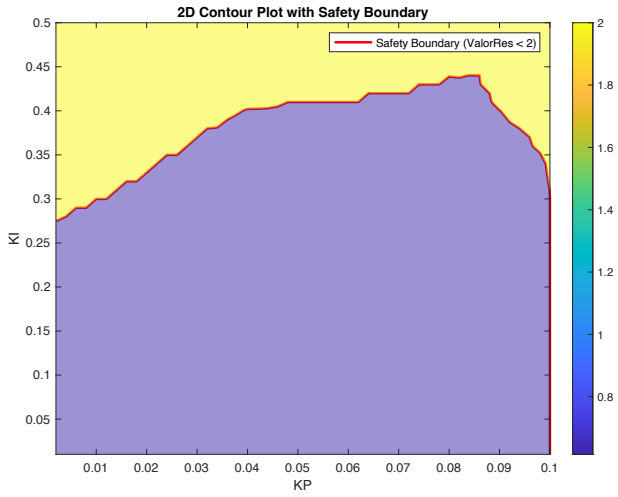


Figura A. 3. Mapa de calor detallado del rendimiento del controlador PI con límite de seguridad ajustado (ValorRes < 2)

Valores Óptimos Encontrados:

- Sin Restricciones

  El valor mínimo de la función objetivo sin tener en cuenta las restricciones de seguridad fue de 0.564889, obtenido en la posición (9, 10), es decir, con un KP = 1,8 y un KI = 2.

- Con Restricciones

  Cuando se aplicaron las restricciones de seguridad, el valor mínimo de la función objetivo fue de 0.615181, ubicado en la posición (49, 35), de la segunda prueba maestra en la que se redujo el área de búsqueda, lo que equivale a un KP = 0.098 y un KI = 3.5. El valor de la restricción, es decir, el valor máximo de la acción de control fue de 0.499111 en una escala entre cero y uno siendo el límite impuesto de 0.5.

Estos resultados muestran cómo la inclusión de restricciones impacta en la búsqueda de soluciones óptimas.

Resultados Obtenidos con OptBayes

Tras la implementación de OptBayes para optimizar el controlador PI, se realizaron varias pruebas para evaluar su rendimiento en la identificación de la mejor configuración de los parámetros del controlador. A continuación, se presentan los resultados obtenidos en una de estas pruebas, donde se optimizaron las variables KP y KI.

El proceso de optimización se realizó con las especificaciones detalladas en la siguiente imagen:

Tabla A 22. Configuración de la Optimización Bayesiana aplicada al controlador PI

```
    resultados = optbayes(fun, variables, 'ObjetivosEval', 30, 'MuestrasIniciales', 5,...
                          'MetodoOptimizacion', 'EI+', 'ActualizacionHiperparametros',[5, 10], ...
                          'Xi',0.01,'ExplorationRatio',0.2,'InitialHiperparametros',[0.25,     1,
0.01]);
```

Durante el proceso de optimización, se generó una visualización 3D en la que se pueden observar las muestras tomadas por el optimizador (puntos rojos), la predicción media del modelo gaussiano y los márgenes de incertidumbre. Esta visualización proporciona una idea clara de cómo el modelo fue ajustando las predicciones a lo largo de las iteraciones.

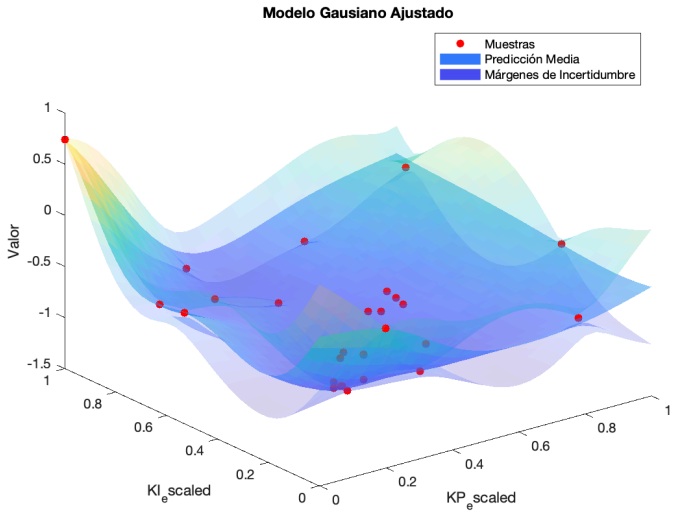


Figura A. 4. Modelo Gaussiano Ajustado con Muestras y Márgenes de Incertidumbre para la Optimización de KP y KI

Finalmente, el optimizador determinó que la mejor combinación de valores para las variables escaladas KP y KI fue la siguiente:

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Best observed feasible point:
     KP_escaled     KI_escaled
         0.1914         0.1940
Observed objective function value = -0.93305
Descaled objective function value = 0.56695
LengthScale = 0.37681
SigmaF = 0.55274
SigmaN = 0.01000
```

Figura A. 5. Resultados optimización PI con OptBayes

En esta solución se muestran los valores escalados de las variables de control KP y KI, ambos entre 0 y 1, junto con el valor escalado de la función objetivo, que se sitúa entre -1 y 1 (-0.93305). También se presenta el valor desescalado de la función objetivo, que en este caso es 0.56695. Además, se incluyen los valores de los hiperparámetros del modelo gaussiano que el optimizador utilizó para alcanzar esta solución.

Si desescalamos las variables de control a su rango original, obtenemos valores reales de 1.914 para KP y 1.94 para KI. Estos valores representan la configuración óptima del controlador PI en el espacio de búsqueda definido.

Es importante mencionar que el proceso de optimización no siempre produce los mismos resultados exactos en cada ejecución, ya que estos dependen de las muestras iniciales que se tomen al comienzo. Sin embargo, el optimizador suele llegar a soluciones muy parecidas, lo que refleja su consistencia y fiabilidad para encontrar configuraciones óptimas.

Resultados Obtenidos con SafeOptBayes

Para validar la capacidad de SafeOptBayes en la optimización de un controlador PI bajo restricciones de seguridad, se realizaron múltiples pruebas con el objetivo de identificar la configuración óptima de los parámetros KP y KI, asegurando al mismo tiempo que la acción de control no excediera un valor máximo permitido.

A continuación, se presentan los resultados obtenidos en una de estas pruebas, utilizando las especificaciones detalladas en la siguiente imagen:

Tabla A 23. Configuración de la Optimización Bayesiana con restricción de seguridad aplicada al controlador PI

```
    resultados = safeoptbayes(fun, variables, 'ObjetivosEval', 50, 'MuestrasIniciales', 5,...
                              'MetodoOptimizacion', 'EI+', 'ActualizacionHiperparametros',[10, 10],
...

'Xi',0.01,'ExplorationRatio',0.2,'SafeValue',0.5,'MetodoSafe','Upper', ...
                              'InitialSafeHiperparametros',[0.25, 1, 0.1], ...
                              'InitialHiperparametros',[0.25, 1, 0.1]);
```

El proceso de optimización generó una visualización 3D que muestra las muestras tomadas por el optimizador (puntos rojos), junto con la predicción media del modelo gaussiano y los márgenes de incertidumbre. Esta visualización no solo refleja cómo el modelo ajustó las predicciones, sino también cómo se integraron las restricciones de seguridad en el proceso.

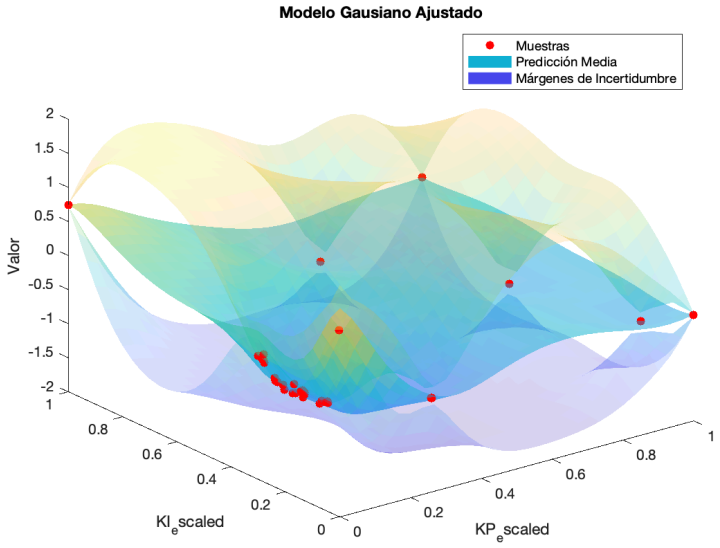


Figura A. 6. Modelo Gaussiano Ajustado con Muestras y Márgenes de Incertidumbre para la Optimización con restricción de seguridad

Finalmente, el optimizador determinó que la mejor combinación de valores para las variables escaladas KP y KI fue la siguiente:

```
Optimization completed.
MaxObjectiveEvaluations of 50 reached.
Total function evaluations: 50
Best observed feasible point:
     KP_escaled     KI_escaled
         0.1000         0.3032
Observed objective function value = -0.88638
Descaled objective function value = 0.61362
LengthScale = 0.10000
SigmaF = 0.58959
SigmaN = 0.01000
```

Figura A. 7. Resultados optimización PI con SafeOptBayes

En esta solución, se muestran los valores escalados de las variables de control KP y KI, ambos entre 0 y 1, junto con el valor escalado de la función objetivo, que se sitúa entre -1 y 1 (-0.88638). También se presenta el valor desescalado de la función objetivo, que en este caso es 0.61362. Además, se incluyen los valores de los hiperparámetros del modelo gaussiano que el optimizador utilizó para alcanzar esta solución.

Además de optimizar la función objetivo, SafeOptBayes también garantizó que la restricción de seguridad se cumpliera. El valor de la restricción en el punto óptimo fue 0.4999, lo que asegura que la acción de control no superó el umbral máximo permitido durante la optimización.

Al desescalar las variables de control a su rango original, obtenemos valores reales de 1 para KP y 3.032 para KI. Estos valores representan la configuración óptima del controlador PI dentro del espacio de búsqueda, considerando tanto la eficiencia del control como la seguridad del sistema.

Al igual que con OptBayes, es importante señalar que los resultados específicos pueden variar ligeramente entre ejecuciones debido a la selección aleatoria de muestras iniciales. Sin embargo,

SafeOptBayes demuestra una alta consistencia en la obtención de soluciones que no solo son óptimas, sino también seguras.

ANEXO II.1. EFICIENCIA Y VALIDACIÓN DE LOS OPTIMIZADORES

Si se comparan los resultados obtenidos con la prueba maestra y los optimizadores, queda claro que ambos logran soluciones prácticamente igual de efectivas. La gran diferencia radica en la eficiencia: mientras que la prueba maestra necesitó evaluar 2500 puntos en el espacio de búsqueda, OptBayes alcanzó las mejores configuraciones con solo 30 evaluaciones.

Para SafeOptBayes, se decidió configurar el número de iteraciones en 50, dado que para validar la restricción de seguridad se llevó a cabo una prueba maestra adicional de 2500 muestras, concentrada específicamente en la zona segura del espacio de búsqueda. Esta configuración permitió una comparación más justa y demostró que, SafeOptBayes fue capaz de encontrar soluciones equivalentes con un número de pruebas significativamente menor

Esto demuestra que OptBayes y SafeOptBayes no solo son precisos a la hora de encontrar soluciones óptimas, sino que además lo hacen de una forma mucho más rápida y eficiente. Esto es muy importante en situaciones donde el tiempo y los recursos son limitados. Gracias a OptBayes y SafeOptBayes, es posible llegar a soluciones óptimas con un numero de pruebas menor, lo que los convierte en herramientas muy útiles para la optimización en sistemas complejos y con restricciones.

ANEXO III: FUNCIONES PARA EL DISEÑO DE CONTROLADORES INICIALES (LQR Y MUSYN)

Este anexo recopila las funciones empleadas para el diseño de los controladores iniciales mediante los métodos LQR y MUSYN, esenciales para controlar el comportamiento del sistema en las primeras etapas del diseño.

ANEXO III.1. CONTROLADOR LQR

A continuación, se incluye la implementación del controlador LQR utilizado en la optimización del sistema:

Tabla A 24. Código implementado para determinar el controlador mediante LQR

```
% Parámetros del sistema
l = 0.5358;
g = 9.81;
a = 9.7220;
c = 0;
M_c = 2.39;
m_p = 0.94;
J_p = 0.0727;
b = 13.6111;

% Punto de equilibrio
r_eq = 0.17; % Valor de equilibrio para r
theta_eq = 0; % Valor de equilibrio para theta

% Matrices del sistema
ACEL = [(M_c + m_p), -(m_p * l);
        -(m_p * l), (J_p + m_p * l^2)];
VAR = [0, 0, -b, 0, a;
       0, (m_p * g * l), 0, -c, 0];
TMP = inv(ACEL) * VAR;

A = [0 0 1 0;
     0 0 0 1;
     TMP(:, 1:4)];

B = [0; 0; TMP(:, 5)];

C1 = [1 0 0 0]; % Para la primera salida (y1 = x1)
C2 = [0 1 0 0]; % Para la segunda salida (y2 = x2)

C = [1 0 0 0;
     0 1 0 0];
D = zeros(size(C, 1), size(B, 2));

sys = minreal(ss(A, B, C, D))
eig(A);

% Matrices de peso para LQR
Q = diag([0.5, 1, 0, 0]); % Ponderación en r y theta
R = 0.2; % Penalización en el esfuerzo de control

% Calcular la ganancia del controlador LQR
[K, S, P] = lqr(A, B, Q, R);

% Mostrar el valor de la ganancia K
disp('Ganancia K calculada por LQR:');
disp(K);
```

```
% Crear sistema de lazo cerrado con la ganancia LQR calculada
A_cl = A - B * K;
sys_cl = ss(A_cl, B, [C;-K], [D;0]);

% Simulación de la respuesta inicial del sistema
[response, t] = initial(sys_cl, [1; 0; 0; 0]);

eigenvalues = eig(A_cl)

figure;

% Graficar la posición del carro
subplot(3,1,1); % Crear un subgráfico para la posición
plot(t, response(:,1), 'LineWidth', 2); % Graficar la posición
grid on;
title('Respuesta del Sistema en Lazo Cerrado');
xlabel('Tiempo (s)');
ylabel('Posición (r)');
set(gca, 'FontSize', 12);

% Graficar el ángulo del péndulo
subplot(3,1,2); % Crear un subgráfico para el ángulo
plot(t, response(:,2), 'LineWidth', 2); % Graficar el ángulo
grid on;
title('Respuesta del Sistema en Lazo Cerrado');
xlabel('Tiempo (s)');
ylabel('Ángulo (\theta)');
set(gca, 'FontSize', 12);

% Graficar el esfuerzo de control
subplot(3,1,3); % Crear un subgráfico para el esfuerzo de control
plot(t, response(:,3), 'LineWidth', 2); % Graficar el esfuerzo de control
grid on;
title('Esfuerzo de Control');
xlabel('Tiempo (s)');
ylabel('Control (u)');
set(gca, 'FontSize', 12);
```

ANEXO III.1. CONTROLADOR MUSYN

El siguiente bloque contiene el código implementado para llevar a cabo el diseño de los controladores PD robustos utilizando MUSYN:

Tabla A 25. Código implementado para determinar los controladores PD mediante MUSYN

```
l = 0.5358;
g = 9.81;
a = 9.7220;
c = 0;

M_c = ureal('MasaCarro',2.39,'Perc',3);
m_p = ureal('MasaPendulo',0.94,'Perc',3);
J_p = ureal('InerciaPendulo',0.0727,'Perc',3);
b = ureal('CoefViscosidad',13.6111,'Perc',10);

MassMatrix = [(M_c+m_p), -(m_p*l); % buscar doc de matlab
        -(m_p*l), (J_p+m_p*l*l)];
VAR = [0, 0, -b, 0, a;
    0, (m_p*g*l),0, -c, 0];
TMP = inv(MassMatrix)*VAR
A=[0 0 1 0;
    0 0 0 1;
    TMP(:,1:4)];
B = [0; 0; TMP(:,5)];

%% Linealización
```

```
    C = [1 0 0 0;
        0 1 0 0];
    sys=minreal(ss(A,B,C,0))
    sys.NominalValue
    size(sys)
    sys.Uncertainty
    G = tf(sys);
    zpk(G)
    s = tf('s');

    deltaact=ultidyn('deltaact',[1 1]);
    Wdelta = makeweight(0.03,500,2);
    sigma(Wdelta), grid on
    act = 1/(0.02*s+1)*(1+Wdelta*deltaact);
    P = sys*act;

    sigma(act), grid on

    %Queremos que el control tenga un ancho de banda objetivo, lo más rápido posible
    w_objetivo=0.45;
    % A partir de una frecuencia de 0.45 rad/s dejarían de cumplirse las prestaciones
    % exigidas
    Werr=makeweight(40,w_objetivo,0.25);

    Wang = 1;
    Wu = 0.2;

    GPNW=[1 0 0;0 0 1;1 0 0;0 1 0]+[zeros(4,1) zeros(4,1) [-1 0;0 0;-1 0;0 1]*P]

    GPNW.InputName={'ref','ruidomedangulo','u'};
    GPNW.OutputName={'err','u_arbitro','err_copia','medida_angulo'};

    GPW=blkdiag(Werr,Wu,1,1)*GPNW*blkdiag(1,Wang,1)


    GPW.Uncertainty

    % Controladores PD
    PID1 = tunablePID('PID1', 'pd');
    PID2 = tunablePID('PID2', 'pd');

    % Definir los nombres de entradas y salidas de cada bloque
    GPW.InputName={'r','ruidomedangulo','u'};
    GPW.OutputName={'e','u_arbitro','err_copia','medida_angulo'};


    % Añadir un bloque de suma para restar el valor de equilibrio
    AddEquilibrio = sumblk('err_ajustado = err_copia - 0.17');

    % Conectar el error ajustado como entrada al PID1
    PID1.InputName = 'err_ajustado';  % Usar el error ajustado
    PID1.OutputName = 'u1';

    PID2.InputName = 'medida_angulo'; % Usar la salida y1 para el PD
    PID2.OutputName = 'u2';

    % Definir las conexiones de los sumadores
    Add = sumblk('u = u1 + u2');

    % Crear el sistema interconectado usando connect
    system_interconnected   =   connect(GPW,   PID1,   PID2,   Add,   AddEquilibrio,   {'r'},
{'e','u_arbitro'});
    %system_interconnected = connect(GPW, PID1, PID2, Add, {'r'}, {'e','u_arbitro'});

    % Opciones para musyn
    opt = musynOptions('MixedMu', 'on');

    % Ejecutar musyn
    [CL, WcGAM2] = musyn(system_interconnected, opt);
    WcGAM2
    %CLmusyn_PID = minreal(lft(GPW,CL));
```

```
    robstab(CL)
    wcgain(CL)
    robgain(CL,1)

    regePID1=pid(CL.Blocks.PID1)
    regePID2=pid(CL.Blocks.PID2)

    % Restar el valor de equilibrio del error de la posición (0.17)
    err_ajustado = sumblk('err_ajustado = err_copia - 0.17');

    % Define nombres de entrada/salida para GPNW y controladores PID resultantes
    regePID1.InputName = 'err_ajustado';  % El error ajustado será la nueva entrada
    regePID1.OutputName = 'u1';

    regePID2.InputName = 'medida_angulo';
    regePID2.OutputName = 'u2';

    % Definir nuevamente las conexiones de los sumadores
    Add = sumblk('u = u1 + u2');

    % Definir nombres de entradas/salidas de GPNW
    GPNW.InputName = {'r', 'ruidomedangulo', 'u'};
    GPNW.OutputName = {'e', 'u_arbitro', 'err_copia', 'medida_angulo'};

    % Conectar el sistema ajustado
    bcms   =   minreal(connect(GPNW,   regePID1,   regePID2,   err_ajustado,   Add,   {'r'},
{'e','medida_angulo', 'u_arbitro'}));

    % Simulación del sistema en bucle cerrado
    step(bcms, 20);
    grid on;
    title('Resp. Escalon SALIDA');
    legend('Bucle cerrado');
```

ANEXO IV: RESULTADOS DE LOS MÉTODOS DE OPTIMIZACIÓN

Este anexo recoge los resultados obtenidos a partir de dos estrategias diferentes de optimización utilizadas para mejorar el rendimiento del controlador.

ANEXO IV.1. MÉTODO 1: OPTIMIZACIÓN GLOBAL SIMULTÁNEA

A continuación, se presenta la tabla que detalla las iteraciones realizadas por el optimizador durante el proceso de optimización global simultánea. Al final de este apartado, también se incluye la configuración final de los controladores obtenida tras el proceso de optimización, donde se destacan los valores óptimos de los parámetros KP y KI.

Tabla A 26. Iteraciones Optimización Global Simultanea

ObjetivosEval: 50
MuestrasIniciales: 1
MetodoOptimizacion: EI+
ActualizacionHiperparametros: 15 10
ExplorationRatio: 1
Xi: 0.01
SafeValue: 1.3

Iteracion	KP1_escaled	KD1_escaled	KP2_escaled	KD2_escaled	N1_escaled	N2_escaled	Objetivo	ValorDesescalado	ValorRestriccion	lengthScale	sigmaF	sigmaN	Mejor
1	0.7800	0.4700	0.8800	0.4950	0.5723	0.4017	-1.1216	1.6352	1.2675	NaN	NaN	NaN	Si
2	0.0000	1.0000	0.0000	0.0000	0.0000	1.0000	-0.5836	3.2493	1.1602	0.2500	1.0000	0.1000	No
3	1.0000	0.0000	0.0000	1.0000	0.0000	1.0000	10.0000	35.0000	10.0000	0.2500	1.0000	0.1000	No
4	1.0000	1.0000	0.0000	0.0000	1.0000	1.0000	10.0000	35.0000	4.0167	0.2500	1.0000	0.1000	No
5	0.0000	1.0000	0.0000	1.0000	1.0000	0.0000	3.5400	15.6201	2.5185	0.2500	1.0000	0.1000	No
6	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000	10.0000	35.0000	5.6137	0.2500	1.0000	0.1000	No
7	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	-0.8861	2.3418	1.0535	0.2500	1.0000	0.1000	No
8	0.0000	0.0000	1.0000	1.0000	1.0000	1.0000	10.0000	35.0000	4.2639	0.2500	1.0000	0.1000	No
9	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	10.0000	35.0000	10.0000	0.2500	1.0000	0.1000	No
10	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000	10.0000	35.0000	10.0000	0.2500	1.0000	0.1000	No
11	0.0000	1.0000	1.0000	0.0000	1.0000	0.0000	10.0000	35.0000	10.0000	0.2500	1.0000	0.1000	No
12	1.0000	0.0000	1.0000	1.0000	0.0000	0.0000	3.8185	16.4556	2.2948	0.2500	1.0000	0.1000	No
13	0.0000	1.0000	1.0000	1.0000	0.0000	0.0000	10.0000	35.0000	10.0000	0.2500	1.0000	0.1000	No
14	1.0000	0.0000	0.0000	1.0000	1.0000	0.0000	-0.9579	2.1263	0.7359	0.2500	1.0000	0.1000	No
15	0.0000	0.0000	0.3874	1.0000	1.0000	0.0000	-1.1231	1.6306	1.1536	0.2500	1.0000	0.1000	Si
16	0.0000	0.0000	0.0000	0.0000	1.0000	1.0000	-1.2719	1.1842	0.7306	0.2500	1.0000	0.1000	Si
17	0.0088	0.0000	0.0000	0.0202	1.0000	0.0209	-0.9447	2.1660	1.0455	0.9354	7.1659	1.0000	No
18	0.8350	0.4126	0.8711	0.5426	0.6391	0.3696	-1.1082	1.6754	1.2335	0.9354	7.1659	1.0000	No
19	0.0000	0.0000	0.0000	0.0000	1.0000	0.9856	-1.2789	1.1634	0.7289	0.9354	7.1659	1.0000	Si
20	0.9819	0.0010	0.0275	0.9896	1.0000	0.0000	-1.0212	1.9364	0.7517	0.9354	7.1659	1.0000	No
21	0.0099	0.0000	0.0000	0.0228	1.0000	0.0237	-0.9518	2.1446	1.0447	0.9354	7.1659	1.0000	No
22	0.9802	0.0007	0.0316	0.9880	1.0000	0.0000	-1.0272	1.9183	0.7533	0.9354	7.1659	1.0000	No
23	0.9793	0.0005	0.0338	0.9873	1.0000	0.0000	-1.0302	1.9094	0.7541	0.9354	7.1659	1.0000	No
24	0.9787	0.0004	0.0351	0.9868	1.0000	0.0000	-1.0321	1.9037	0.7546	0.9354	7.1659	1.0000	No
25	0.0105	0.0000	0.0000	0.0242	1.0000	0.0249	-0.9548	2.1357	1.0443	0.9354	7.1659	1.0000	No
26	0.9783	0.0004	0.0361	0.9864	1.0000	0.0000	-1.0334	1.8997	0.7550	0.9354	7.1659	1.0000	No
27	0.8916	0.0000	0.0972	0.9636	1.0000	0.0000	-1.1812	1.4563	0.8086	1.0000	6.8125	0.0100	No
28	0.7835	0.4665	0.8795	0.4979	0.5766	0.3999	-1.1211	1.6367	1.2654	1.0000	6.8125	0.0100	No
29	0.7631	0.0000	0.1186	0.9525	1.0000	0.0000	-1.2389	1.2832	0.8633	1.0000	6.8125	0.0100	No
30	0.6374	0.0000	0.1117	0.9459	1.0000	0.0000	-1.2437	1.2688	0.9001	1.0000	6.8125	0.0100	No
31	0.0108	0.0000	0.3682	1.0000	1.0000	0.0000	-1.1316	1.6051	1.1395	1.0000	6.8125	0.0100	No
32	0.7859	0.4635	0.8787	0.4993	0.5793	0.3980	-1.1216	1.6351	1.2638	1.0000	6.8125	0.0100	No

33	0.0528	0.0000	0.3057	0.9892	1.0000	0.0000	-1.1551	1.5348	1.1077	1.0000	6.8125	0.0100	No
34	0.0000	0.0000	0.0000	0.0000	1.0000	0.8203	-1.3180	1.0461	0.7580	1.0000	6.8125	0.0100	Sí
35	0.7536	0.0000	0.1381	0.9126	1.0000	0.0000	-1.2417	1.2749	0.8721	1.0000	6.8125	0.0100	No
36	0.7904	0.4619	0.8770	0.5037	0.5839	0.3973	-1.1209	1.6372	1.2582	1.0000	6.8125	0.0100	No
37	0.0000	0.0000	0.0000	0.0000	0.9548	0.8887	-1.3132	1.0605	0.7446	1.0000	5.9197	0.0100	No
38	0.0314	0.0000	0.0000	0.0959	1.0000	0.1003	-1.0885	1.7344	1.0152	1.0000	5.9197	0.0100	No
39	0.7888	0.4619	0.8780	0.5023	0.5826	0.3973	-1.1209	1.6373	1.2608	1.0000	5.9197	0.0100	No
40	0.7974	0.4543	0.8748	0.5106	0.5933	0.3936	-1.1206	1.6381	1.2515	1.0000	5.9197	0.0100	No
41	0.8052	0.0000	0.1831	0.9571	1.0000	0.0000	-1.2401	1.2797	0.8628	1.0000	5.9197	0.0100	No
42	0.0000	0.0517	0.0000	0.0000	0.9511	0.9154	-1.3146	1.0562	0.7415	1.0000	5.9197	0.0100	No
43	0.7953	0.4561	0.8761	0.5082	0.5905	0.3943	-1.1204	1.6387	1.2548	1.0000	5.9197	0.0100	No
44	0.8305	0.4180	0.8713	0.5388	0.6337	0.3728	-1.1102	1.6694	1.2351	1.0000	5.9197	0.0100	No
45	0.7995	0.4520	0.8752	0.5119	0.5957	0.3921	-1.1197	1.6408	1.2517	1.0000	5.9197	0.0100	No
46	0.0000	0.1062	0.0000	0.0000	0.8526	1.0000	-1.3074	1.0779	0.7360	1.0000	5.9197	0.0100	No
47	0.8106	0.0000	0.3115	0.8981	1.0000	0.0000	-1.2468	1.2597	0.9034	1.0000	5.4801	0.0100	No
48	0.9089	0.0000	0.3409	0.9502	1.0000	0.0000	-1.1916	1.4252	0.8754	1.0000	5.4801	0.0100	No
49	0.8292	0.4195	0.8715	0.5377	0.6321	0.3737	-1.1107	1.6679	1.2358	1.0000	5.4801	0.0100	No
50	0.6241	0.0000	0.2339	0.8654	1.0000	0.0000	-1.2558	1.2327	0.9414	1.0000	5.4801	0.0100	No

Tabla A 27. Resultados obtenidos con la Optimización Global Simultanea

```
Optimization completed.
MaxObjectiveEvaluations of 50 reached.
Total function evaluations: 50
Best observed feasible point:
    KP1_escaled    KD1_escaled    KP2_escaled    KD2_escaled     N1_escaled     N2_escaled
         0.0000         0.0000         0.0000         0.0000         1.0000         0.8203
Observed objective function value = -1.31800
Descaled objective function value = 1.04610
LengthScale = 1.00000
SigmaF = 6.81250
SigmaN = 0.01000
```

ANEXO IV.1. MÉTODO 1: OPTIMIZACIÓN POR FASES

A continuación, se presentan las tablas que detallan las iteraciones realizadas por el optimizador durante el proceso de optimización por fases. En este enfoque, primero se optimizan las ganancias proporcionales (P) y derivativas (D), y luego se ajustan los valores de los filtros N. Al final de este apartado, también se incluye la configuración final de los controladores obtenida tras el proceso de optimización en cada fase, destacando los valores óptimos de los parámetros KP, KD y N.

Tabla A 28. Iteraciones primera fase de la Optimización por Fases

ObjetivosEval: 30
MuestrasIniciales: 5
MetodoOptimizacion: EI+
ActualizacionHiperparametros: 10 10
ExplorationRatio: 5
Xi: 0.01

Iteracion	KP1_escaled	KD1_escaled	KP2_escaled	KD2_escaled	Objetivo	ValorDesescalado	lengthScale	sigmaF	sigmaN	Mejor
1	0.1276	0.2666	0.4068	0.5079	12.0000	41.0000	NaN	NaN	NaN	No
2	0.7412	0.3191	0.6726	0.0282	12.0000	41.0000	NaN	NaN	NaN	No
3	0.9436	0.8722	0.3287	0.1993	-1.2417	1.2749	NaN	NaN	NaN	Sí
4	0.4789	0.7250	0.8112	0.2258	1.6996	10.0987	NaN	NaN	NaN	No
5	0.0342	0.5955	0.2598	0.1794	-0.5873	3.2380	NaN	NaN	NaN	No
6	0.9862	0.9780	0.2630	0.2254	-1.0431	1.8707	0.2500	1.0000	0.1000	No
7	0.0000	0.7218	0.1924	0.0690	-0.8239	2.5283	0.2500	1.0000	0.1000	No
8	0.9888	0.9880	0.3465	0.1476	-0.9929	2.0213	0.2500	1.0000	0.1000	No
9	0.8579	0.9162	0.2283	0.1568	-1.2345	1.2966	0.2500	1.0000	0.1000	No
10	0.8501	0.9363	0.3029	0.2608	-1.2081	1.3757	0.2500	1.0000	0.1000	No
11	0.9819	0.8320	0.2020	0.2037	-1.2285	1.3146	0.2500	1.0000	0.1000	No
12	0.9058	0.8756	0.2397	0.2343	-1.2442	1.2673	0.2500	1.0000	0.1000	Sí
13	0.9786	0.8751	0.3066	0.3436	-1.2057	1.3828	0.2500	1.0000	0.1000	No
14	0.8638	0.9033	0.1927	0.3639	-1.2282	1.3155	0.2500	1.0000	0.1000	No
15	0.8878	0.8894	0.0790	0.2836	-1.2427	1.2720	0.2500	1.0000	0.1000	No
16	0.0000	1.0000	0.7416	0.2669	12.0000	41.0000	1.0000	9.7515	0.0100	No
17	0.1729	0.6016	0.0000	0.0561	-0.9048	2.2855	1.0000	9.7515	0.0100	No
18	1.0000	1.0000	1.0000	1.0000	-0.2750	4.1750	1.0000	9.7515	0.0100	No
19	1.0000	1.0000	0.0000	1.0000	-0.4914	3.5258	1.0000	9.7515	0.0100	No
20	1.0000	1.0000	1.0000	0.3396	-0.6123	3.1631	1.0000	9.7515	0.0100	No
21	0.2042	0.8122	0.0000	0.3444	-1.0395	1.8816	1.0000	9.7515	0.0100	No
22	0.3741	0.7820	0.1928	0.1464	-1.0761	1.7717	1.0000	9.7515	0.0100	No
23	1.0000	1.0000	0.5559	0.8112	-0.6689	2.9933	1.0000	9.7515	0.0100	No
24	0.8385	0.9361	0.7094	0.4268	-1.0784	1.7649	1.0000	9.7515	0.0100	No
25	0.0000	0.6436	0.0000	0.1886	-0.8568	2.4295	1.0000	9.7515	0.0100	No
26	0.4428	1.0000	0.0000	1.0000	-0.8185	2.5446	0.9999	9.9992	0.0100	No
27	0.6099	1.0000	0.0000	0.6638	-0.9635	2.1096	0.9999	9.9992	0.0100	No
28	1.0000	0.5511	0.0000	1.0000	-0.1136	4.6592	0.9999	9.9992	0.0100	No
29	0.7951	1.0000	0.8445	0.0000	3.1010	14.3029	0.9999	9.9992	0.0100	No
30	1.0000	0.8063	1.0000	0.6468	-0.3651	3.9047	0.9999	9.9992	0.0100	No

Tabla A 29. Resultados obtenidos en la primera fase de la Optimización por Fases

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Best observed feasible point:
    KP1_escaled    KD1_escaled    KP2_escaled    KD2_escaled
         0.9058         0.8756         0.2397         0.2343
Observed objective function value = -1.24420
Descaled objective function value = 1.26730
LengthScale = 0.25000
SigmaF = 1.00000
SigmaN = 0.10000
```

Tabla A 30. Iteraciones segunda fase de la Optimización por Fases

```
ObjetivosEval: 20
MuestrasIniciales: 5
MetodoOptimizacion: EI+
ActualizacionHiperparametros: 10  10
ExplorationRatio: 0.5
Xi: 0.01
 Iteracion     N1_escaled     N2_escaled       Objetivo     ValorDesescalado     lengthScale          sigmaF          sigmaN      Mejor
         1         0.5702         0.7499        -1.2582               1.2254             NaN             NaN             NaN         Sí
         2         0.8745         0.0656        -1.1528               1.5415             NaN             NaN             NaN         No
         3         0.4354         0.1039        -1.2076               1.3771             NaN             NaN             NaN         No
         4         0.0842         0.5230        -1.2084               1.3749             NaN             NaN             NaN         No
         5         0.6034         0.0514        -1.1886               1.4341             NaN             NaN             NaN         No
         6         0.3830         0.5292        -1.2426               1.2721          0.2500          1.0000          0.1000         No
         7         0.5815         0.2886        -1.2231               1.3306          0.2500          1.0000          0.1000         No
         8         0.2686         0.3467        -1.2263               1.3210          0.2500          1.0000          0.1000         No
         9         0.5967         0.5692        -1.2478               1.2565          0.2500          1.0000          0.1000         No
        10         0.3883         0.7175        -1.2493               1.2521          0.2500          1.0000          0.1000         No
        11         0.7547         0.1973        -1.1946               1.4161          0.2500          1.0000          0.1000         No
        12         0.2235         0.5703        -1.2297               1.3109          0.2500          1.0000          0.1000         No
        13         0.4971         0.6519        -1.2526               1.2422          0.2500          1.0000          0.1000         No
        14         0.4228         0.2887        -1.2279               1.3164          0.2500          1.0000          0.1000         No
        15         0.7983         0.4583        -1.2240               1.3279          0.2500          1.0000          0.1000         No
        16         0.8908         0.9574        -1.2578               1.2266          1.0000          0.4320          0.0100         No
        17         0.7685         0.8519        -1.2593               1.2220          1.0000          0.4320          0.0100         Sí
        18         0.6608         1.0000        -1.2678               1.1967          1.0000          0.4320          0.0100         Sí
        19         0.7196         0.9746        -1.2667               1.1999          1.0000          0.4320          0.0100         No
        20         0.7048         0.9631        -1.2665               1.2005          1.0000          0.4320          0.0100         No
```

Tabla A 31. Resultados obtenidos en la segunda fase de la Optimización por Fases

```
Optimization completed.
MaxObjectiveEvaluations of 20 reached.
Total function evaluations: 20
Best observed feasible point:
     N1_escaled     N2_escaled
         0.6608         1.0000
Observed objective function value = -1.26780
Descaled objective function value = 1.19670
LengthScale = 1.00000
SigmaF = 0.43196
SigmaN = 0.01000
```

ANEXO V: MODELOS DE SIMULINK EMPLEADOS

En este anexo se incluyen los modelos de Simulink empleados durante el desarrollo del proyecto para la validación de los algoritmos de control y optimización.

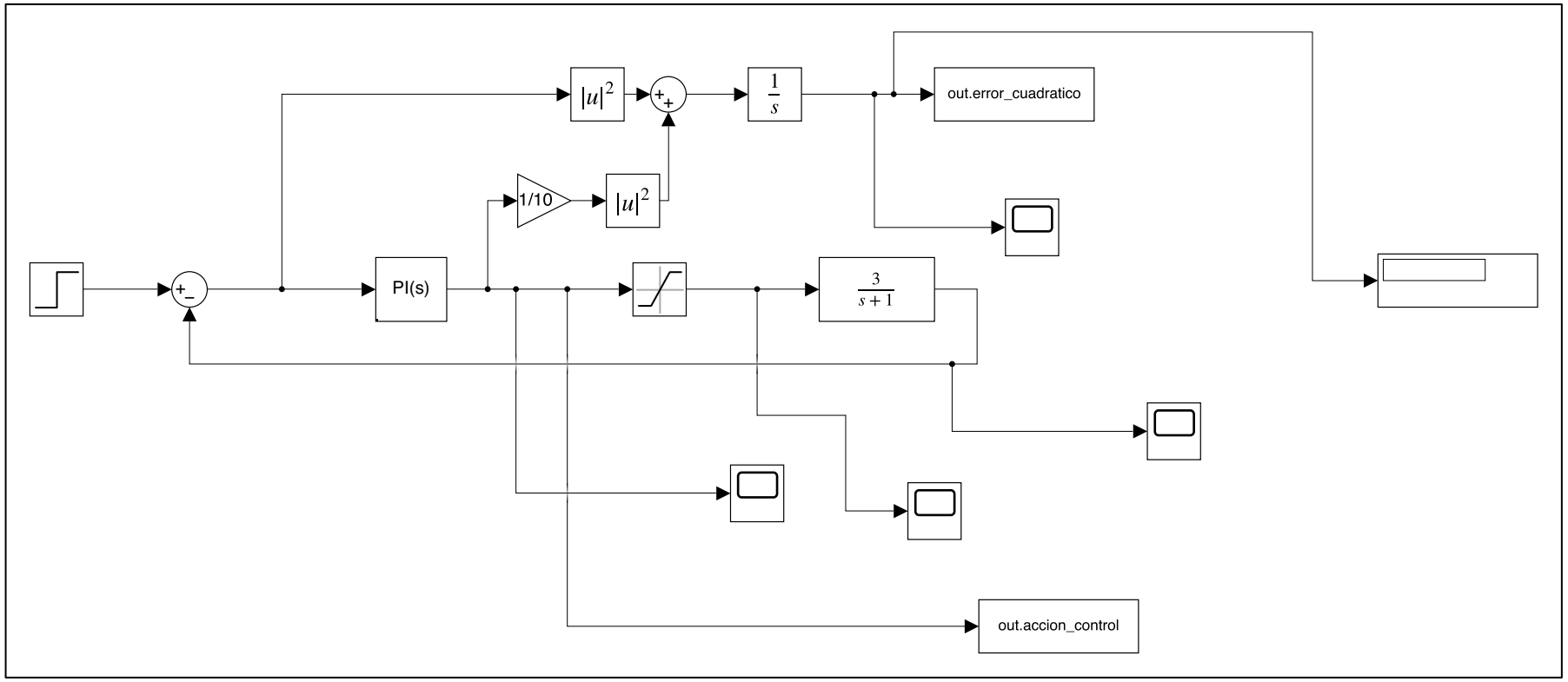


Figura A. 8. Modelo de Simulink para la prueba del controlador PI de la validación

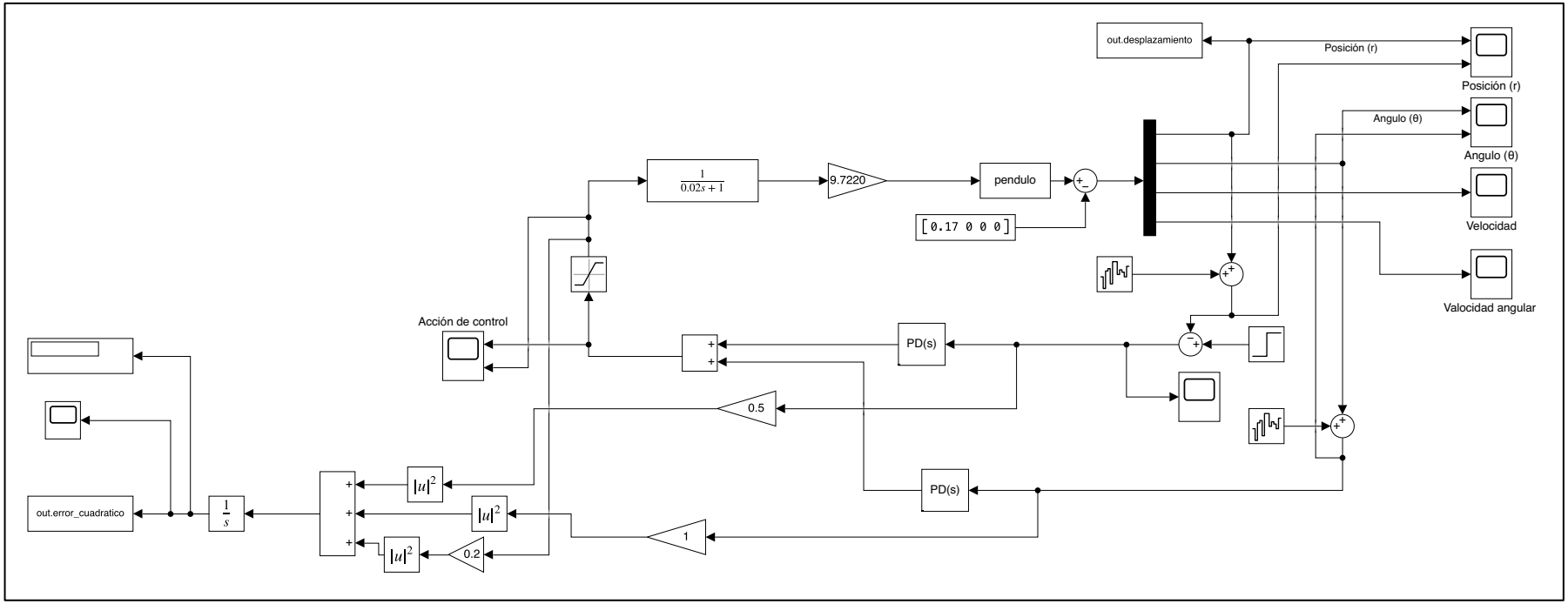


Figura A. 9. Modelo Simulink para probar y optimizar los controladores PD desarrollados con Musyn

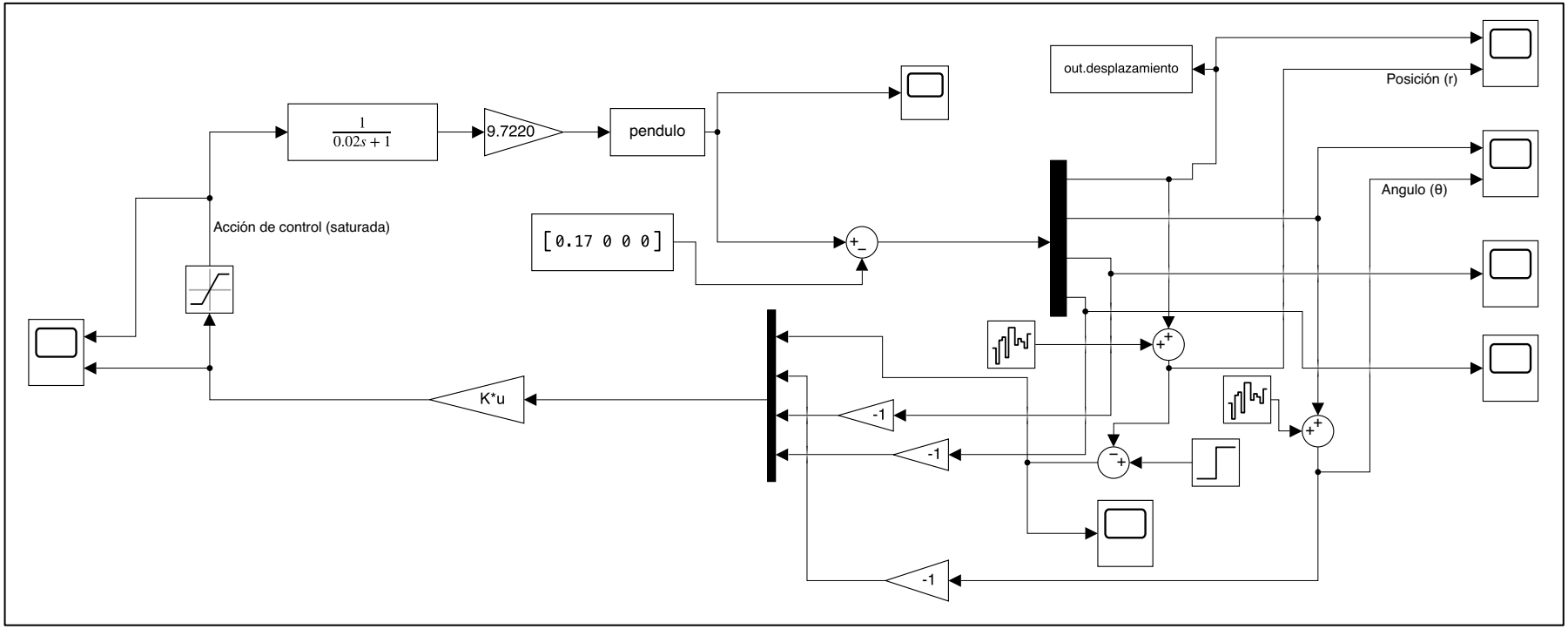


Figura A. 10. Modelo de Simulink para la prueba del controlador LQR

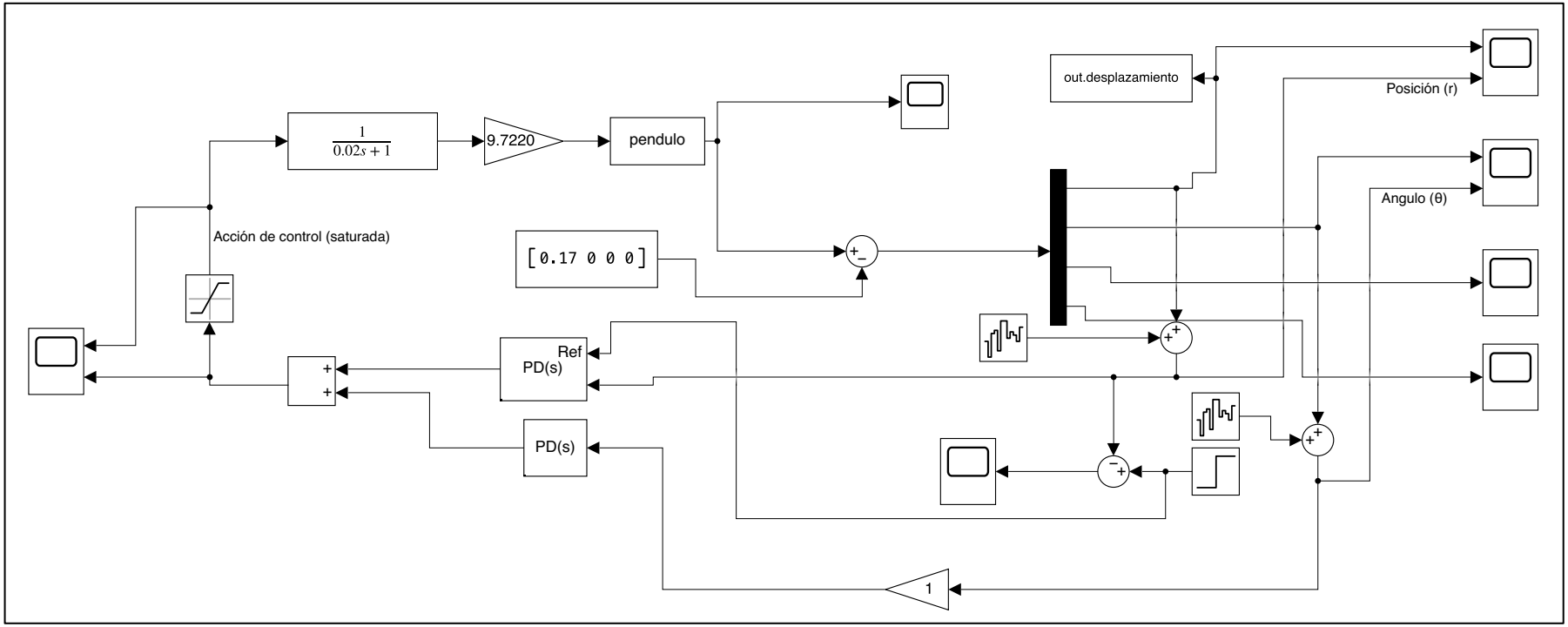


Figura A. 11. Modelo de Simulink para la prueba de controladores PD desarrollados a partir del LQR