Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática industrial 14 (2017) 193–204

www.elsevier.es/RIAI

Heurísticas para el Ajuste de Algoritmos de Control de Plataformas Robóticas de Movimiento en Simuladores

Sergio Casas, Cristina Portalés*, José V. Riera, Marcos Fernández

Instituto de Robótica y Tecnologías de la Información y la Comunicación, Universitat de València, C/ Catedrático José Beltrán 2, 46980, Paterna, Valencia, España.

Resumen

Diversos tipos de plataformas robóticas son empleadas habitualmente para la generación de claves gravito-inerciales en simuladores. Además del control de los actuadores, dichas plataformas deben ejecutar complejos algoritmos de control conocidos como algoritmos de *washout*, que deben ser ajustados para que el movimiento generado sea similar al simulado. El ajuste de dichos algoritmos es complejo por el elevado número de parámetros que poseen. Además, dicho ajuste se ha venido realizando tradicionalmente de modo manual mediante evaluaciones subjetivas. En este trabajo, los autores proponen un método automático de ajuste basado en optimización heurística, métricas objetivas, y simulación de la plataforma robótica para conseguir realizar el ajuste de manera más rápida. Se valida la corrección de las soluciones, y se comparan diversas técnicas de optimización, para concluir que la técnica más apropiada es la de los algoritmos genéticos.

Palabras Clave:
Plataformas de movimiento, heurísticas, simuladores, algoritmos de control, ajuste de parámetros, robótica, optimización.

1. Introducción

El objetivo de todo simulador de vehículos es proporcionar al usuario una sensación de pertenencia a un entorno virtual alternativo. Para ello, se deben estimular, del modo más eficaz posible, las claves sensoriales que hacen posible que el usuario sienta como cierta esa pertenencia al mundo virtual. Aunque la mayoría de simuladores suelen centrarse en las claves visuales y sonoras, existen otras que también deben ser tenidas en cuenta. Entre las claves extra-audiovisuales más importantes se encuentran las claves gravito-inerciales, relacionadas con la percepción del movimiento. Este tipo de claves se estimulan habitualmente en simuladores de vehículos mediante la construcción de plataformas robóticas de movimiento, sobre las que se suele situar al usuario de la simulación. Estas plataformas están dotadas de actuadores que permiten desplazarlas y orientarlas, pero siempre dentro de unos límites (Figura 1). Para el control de los movimientos de la plataforma se diseñan unos algoritmos de control específicos conocidos como algoritmos de generación de claves gravito-inerciales (en inglés *Motion Cueing/Drive Algorithms – MCA/MDA*) (Schmidt and Conrad 1969), también conocidos como algoritmos de *washout*. Estos algoritmos toman como entrada el estado físico del vehículo simulado y generan como salida la pose deseada para la plataforma robótica en forma de grados de libertad (GdL) traslacionales y rotacionales, que es transformada en consignas para el control de los actuadores (Garrett and Best 2010).

En un nivel de abstracción inferior se situarían los algoritmos de control de los propios actuadores, cuya solución no se analiza en este trabajo. La Figura 2 muestra el esquema completo de generación de claves gravito-inerciales con manipuladores robóticos. El sistema de simulación debe implementar un módulo físico, que puede simular diferentes tipos de vehículos (Slob 2008, Cossalter, Lot et al. 2011, Casas, Rueda et al. 2012) y que interactúa con el piloto a través de una reproducción de los mandos del vehículo. Éste módulo calcula un estado físico simulado, que es el que se desea generar sobre la plataforma robótica. Ésta, mediante un algoritmo de control (MCA) generará los comandos necesarios para que la plataforma se mueva provocando sobre el usuario un estado físico similar al simulado por el módulo físico. Para ello, genera la pose deseada para la plataforma de movimiento y, tras comprobar que no se exceden los límites físicos, se envían los comandos deseados para los actuadores, empleando las ecuaciones de la cinemática inversa del manipulador. Opcionalmente, aunque no es lo habitual, se puede calcular, mediante la cinemática directa, el estado físico real de la plataforma y comprobar que es el deseado o utilizarlo como entrada de algoritmo MCA.

Aunque este tipo de claves ha sido incluido en simuladores desde hace más de 50 años, se ha avanzado menos en este campo

* Autor en correspondencia.
Correos electrónicos: sergio.casas@uv.es (Sergio Casas), cristina.portales@uv.es (Cristina Portalés), jorielo@robotica.uv.es (José V. Riera), marcos.fernandez@uv.es (Marcos Fernández),
URL: artec.uv.es (Sergio Casas)

http://dx.doi.org/10.1016/j.riai.2016.09.011
Disponible en www.sciencedirect.com

que en otros aspectos de la simulación, como el de la generación de claves audiovisuales.

Figura 1: Plataforma robótica de movimiento. Fuente: *Cobra Simulation*®.

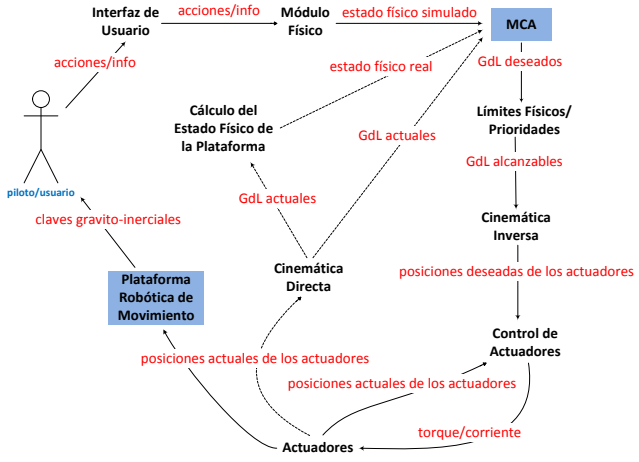

Figura 2: Esquema de generación de claves gravito-inerciales en simuladores.

Los autores identifican tres razones fundamentales para ello. La primera es la naturaleza del problema. Es un problema complejo, cuya solución depende de múltiples factores, entre ellos factores humanos difíciles de medir (y todavía parcialmente desconocidos) relacionados con la percepción del movimiento. La segunda es la falta de un criterio de evaluación para poder comparar diferentes soluciones. Y la tercera, la multitud de parámetros que se deben ajustar en los distintos algoritmos de *washout* para conseguir que se comporten de la forma esperada (Grant and Reid 1997).

La evaluación de este tipo de algoritmos está íntimamente relacionada con el ajuste de parámetros, ya que éstos se deben ajustar en base a un criterio de evaluación, y a su vez, el resultado de la evaluación depende en gran medida del ajuste realizado.

1.1. Algoritmos de Washout/MCA

El objetivo de estos algoritmos es generar las claves gravito-inerciales necesarias para que el piloto sienta mayor inmersión en el manejo del vehículo simulado. Para ello, los algoritmos MCA deben proporcionar siempre la pose deseada para la plataforma de movimiento. Esta pose se especifica siempre en forma de 6 GdL, aunque si la plataforma tiene menos de 6 GdL, algunos no se emplearán. Las entradas del simulador, sin embargo, pueden variar de un algoritmo a otro, aunque casi siempre se corresponden con la aceleración lineal (o fuerza específica) y la velocidad angular del vehículo simulado. Estas magnitudes son las que es capaz de sensorizar el sistema vestibular humano, por lo que es lógico emplearlas como entradas. Opcionalmente se pueden utilizar otras, como la posición o la orientación del vehículo simulado, aunque no es habitual. El objetivo fundamental de cualquier algoritmo de *washout* es reproducir el movimiento que está experimentando el vehículo simulado, intentando al mismo tiempo que la plataforma robótica no alcance sus límites físicos (porque si lo hace, el control deberá detenerla). Lógicamente, ambos objetivos son contrapuestos, por lo que rara vez es posible reproducir el movimiento original.

Existen numerosos tipos de algoritmos de *washout* que se aproximan al problema de manera diferente (Stahl, Abdulsamad et al. 2014) aunque todos ellos hacen uso de una serie de principios básicos comunes. La primera idea es reproducir los mismos movimientos que el vehículo virtual, pero a menor escala, reduciendo su intensidad en un factor prefijado. La segunda idea, más sofisticada, es la aplicación de filtros pasa-alta a las entradas del algoritmo (normalmente aceleraciones y velocidades angulares) que eliminen los movimientos de baja frecuencia que son los responsables de los desplazamientos largos y sostenidos (Reymond and Kemeny 2000). Al eliminar las aceleraciones o velocidades de baja frecuencia, perdemos fidelidad en la reproducción del movimiento, pero nos aseguramos que la plataforma de movimiento vuelva eventualmente a su posición neutral. La tercera idea consiste en aprovechar la ilusión somatográvica (MacNeilage, Banks et al. 2007), para hacer percibir al cerebro las aceleraciones sostenidas (eliminadas por la aplicación de filtros pasa-alta) mediante pequeñas inclinaciones que modifican la dirección relativa del vector gravedad con respecto al sistema de referencia del aparato vestibular. Esta idea se implementa aplicando filtros pasa-baja a las aceleraciones lineales para inclinar ligera y lentamente la plataforma en los ejes lateral y longitudinal, en función del valor de estas aceleraciones filtradas. A esta técnica se le conoce como *tilt coordination* (coordinación angular). Para una explicación más detallada de estas técnicas y de algunos de los tipos de algoritmos MCA más utilizados, el lector puede consultar (Nahon and Reid 1990).

1.2. Evaluación los Algoritmos de Washout/MCA

Dado que por las restricciones físicas de los actuadores (tanto de espacio como de fuerza), no es posible generar un movimiento idéntico al original, se define la fidelidad (Sinacori 1977) gravito-inercial de un simulador como la capacidad de éste para generar las sensaciones de movimiento que se percibirían en una situación real con un vehículo. La evaluación de un algoritmo de *washout* es, por tanto, un método para obtener un valor de fidelidad para el simulador. No existen demasiadas referencias en la literatura sobre la evaluación de este tipo de algoritmos, y lo habitual es que el criterio de evaluación sea simplemente la opinión de uno o varios pilotos. Desde un punto de vista taxonómico, la evaluación se puede hacer básicamente de dos maneras: objetiva (cuantitativa) o subjetivamente (cualitativamente). La evaluación objetiva puede, además, ser directa o indirecta.

En la evaluación objetiva directa lo que se evalúa son ciertas magnitudes físicas que se corresponden con la ejecución del

algoritmo. Es decir, se evalúa cómo son las salidas del algoritmo con respecto a las que deberían ser idealmente.

En la evaluación objetiva indirecta no se evalúan las salidas del algoritmo, sino que se evalúa cómo los parámetros de éste influyen en el desarrollo, por parte del usuario, de tareas ejecutadas sobre el simulador.

El último tipo de evaluación (la más habitual) es la evaluación subjetiva. En este tipo de evaluación lo que se hace es someter a los pilotos a diversos algoritmos MCA o diversas variaciones de los mismos y preguntar a los pilotos acerca sus sensaciones sobre el simulador. Existen múltiples ejemplos de evaluación subjetiva (Reid and Nahon 1986, Gutridge 2004, Go, Bürki-Cohen et al. 2003) ya que ha venido siendo la manera tradicional de evaluar este tipo de problemas, aunque hay pocos estudios formales sobre ello, a excepción hecha de (Grant 1996, Grant and Reid 1997). La evaluación subjetiva tiene la ventaja de que es la forma natural de evaluar el problema. Aunque también presenta desventajas importantes: es un método muy lento; existen variaciones significativas entre los diferentes individuos; dada la naturaleza de las personas, la evaluación no garantiza la repetibilidad de las evaluaciones, aunque se den las mismas circunstancias.

1.3. Ajuste de Parámetros en los Algoritmos de Washout/MCA

Dado que una plataforma robótica de movimiento no siempre va a poder generar el movimiento que se correspondería con el estado físico del vehículo, el diseñador del sistema debe controlar qué parte del movimiento elimina y qué parte intenta reproducir, cosa que, de una u otra forma, acaba traduciéndose en parámetros del algoritmo MCA. Aunque con diferentes valores y significados, todos los algoritmos de este tipo tienen ciertos parámetros (a veces decenas de ellos) que controlan cuál es la cantidad y tipo de información que desechamos para que no se alcancen los límites de los actuadores.

Los parámetros concretos dependen del tipo de algoritmo de *washout* utilizado, aunque habitualmente están relacionados con los escalados/ganancias aplicados a las entradas, con los filtros de frecuencia pasa-alta o pasa-baja aplicados a las señales en los diferentes bloques de procesamiento del algoritmo, y con los límites de inclinación establecidos para aplicar la técnica de coordinación angular.

Dado que este trabajo no se limita a ningún algoritmo de *washout* concreto, no entraremos a detallar los valores y significados de todos estos parámetros porque dependen del tipo e implementación escogida, aunque podemos dar unas ideas generales que ayuden a entender su propósito. Habitualmente, a mayor ganancia en las entradas, mayor fidelidad, pero mayor riesgo de alcanzar los límites de la plataforma. En cuanto a los filtros pasa-alta, a mayor laxitud (frecuencias de corte más baja) mayor fidelidad, pero también mayor riesgo de alcanzar los límites físicos del manipulador robótico. Al contrario, si el filtro es muy restrictivo, se introducirán retrasos e inconsistencias notables en la generación de los movimientos, aunque la probabilidad de alcanzar los límites de la plataforma será menor. En los filtros pasa-baja, a mayor frecuencia de corte mayor capacidad de simular aceleraciones sostenidas, pero mayor probabilidad de que el efecto sea desambiguado por el cerebro. Algo similar ocurre con los límites de inclinación de la técnica de coordinación angular: a mayor límite, más capacidad de simulación, pero mayor interferencia con otros movimientos y mayor probabilidad de que el piloto perciba incorrectamente el movimiento.

El ajuste de parámetros en los algoritmos de *washout* ha venido realizándose tradicionalmente mediante pruebas sucesivas con pilotos y/o expertos en el tipo de vehículo simulado. El proceso es tedioso, ya que normalmente las opiniones de los pilotos son difíciles de traducir en cambios de parámetros. En (Reid and Nahon 1986) se puede consultar un ejemplo de este tipo de aproximaciones para el ajuste del algoritmo de *washout* más conocido y utilizado: el algoritmo *clásico* (Reid and Nahon 1985, Nahon and Reid 1990). Sólo para el ajuste de parámetros, Reid y Nahon dedican un volumen completo de su trabajo, lo cual da idea de la magnitud del problema.

El ajuste de los parámetros se realiza habitualmente con el método *pilot-in-the-loop*. Esta técnica requiere un piloto y un experto en el algoritmo. Las impresiones del piloto son pasadas al experto, que va realizando cambios en el algoritmo hasta que el experto considera que se obtiene el mejor conjunto de parámetros a juicio del piloto (Colombet, Dagdelen et al. 2008). Esta técnica presenta un gran número de problemas. El primero es que, muchas veces, los pilotos sólo son capaces de recordar unos pocos detalles de las últimas pruebas realizadas, por lo que su capacidad para decidir si la prueba es la mejor hasta el momento, se ve muy reducida. El segundo problema es la falta de un criterio para decidir cuándo terminar el proceso. ¿Cómo sabemos que el ajuste es lo suficientemente bueno? Normalmente, por desgracia, este criterio suele ser cuando el piloto y el experto están ya suficientemente cansados como para no ser capaces de distinguir ningún camino de mejora. Lo cual nos lleva al tercer problema. Aunque es lógico que la evaluación se base en la percepción de los pilotos, no es lógico que le añadamos al proceso la subjetividad de las decisiones del experto, que puede decidir ir por un camino o por otro sin otro criterio que el de su intuición. Además, el experto, debe poseer un conocimiento demasiado alto sobre el efecto de numerosos parámetros en un sistema complejo. Pero quizás el mayor problema es el tiempo que deben dedicarle el piloto y el experto al proceso. En ocasiones hacen falta días de pruebas para conseguir un resultado aceptable. El último problema es, evidentemente, que el criterio del piloto es subjetivo y, por tanto, puede ser muy diferente del de otros compañeros, por lo que un cambio de usuario puede implicar tener que volver a repetir todo el proceso.

2. Objetivos y Planteamiento del Problema

Dados los problemas que presenta el ajuste de parámetros de los algoritmos de *washout* y la lentitud de la evaluación de los mismos, el objetivo de este trabajo es presentar una alternativa al método tradicional subjetivo de pruebas sucesivas para el ajuste de los parámetros de este tipo de algoritmos de control de plataformas robóticas de movimiento en simuladores. Nuestra propuesta es simular la plataforma de movimiento (para no usar la real y acelerar el proceso) y medir sobre esta plataforma simulada las salidas (el estado físico) que generaría. A partir de esta información, proponemos medir una serie de indicadores objetivos sobre el movimiento generado (comparándolo con el que debería generarse) y proceder a un ajuste automático de los parámetros de los algoritmos MCA mediante técnicas de optimización. Es importante resaltar que este trabajo no se limita a ningún algoritmo de *washout* concreto, ya que intenta ser neutral en cuanto a la elección del MCA para poder ser aplicado al mayor número de entornos y aplicaciones posible. Posteriormente se validará su uso con el algoritmo MCA clásico, pero sin que esto limite su aplicación a otros algoritmos de *washout*.

2.1. Simulación de la Plataforma de Movimiento

Una plataforma de movimiento es un sistema autónomo de generación de movimiento accionado de forma electro-mecánica. Normalmente las plataformas de movimiento son manipuladores robóticos paralelos compuestos de motores eléctricos, piezas mecánicas rígidas y articulaciones como rótulas, uniones Cardan o bisagras (Merlet 2006). El diseño más popular es el de Gough-Stewart (Stewart 1965, Korobeynikov and Turlapov 2005).

A pesar de que su uso es generalizado, no es fácil generar las señales apropiadas para estos manipuladores de modo que obtengamos el comportamiento deseado (Grant 1996). Se requiere un amplio conjunto de pruebas para cada plataforma de movimiento y para cada uso específico. Este proceso es costoso y algunas de estas pruebas pueden realizar movimientos severos y potencialmente dañinos que pueden averiar el manipulador robótico, y lo que es más importante, pueden causar daños a humanos en caso de fallo de *software* o *hardware*.

Para resolver estos problemas, se puede utilizar un simulador de plataforma de movimiento. Este simulador realizaría una emulación computarizada de un manipulador robótico real. De esta forma podrán hacerse múltiples pruebas sin dañar el *hardware* de la plataforma, y garantizando la seguridad de los probadores humanos. Además, dado que es una simulación, las tareas realizadas por la plataforma virtual pueden ejecutarse más rápido que las tareas reales en la plataforma de movimiento real, con el consiguiente ahorro de tiempo.

La idea principal detrás de este simulador es sustituir completamente a la plataforma real, de forma que podamos usar la plataforma virtual en las mismas circunstancias y con los mismos componentes externos que pueda tener la plataforma real, pero de forma más segura y rápida. Esto significa que la plataforma de movimiento simulada debe recibir exactamente las mismas entradas y proporcionar, al menos, las mismas salidas que la plataforma de movimiento real. Por tanto, las entradas de la plataforma de movimiento virtual serán los ángulos deseados para los motores (ya que nos centramos en motores rotacionales), al igual que sucede en la plataforma real. Con respecto a las salidas, la plataforma de movimiento virtual debe proporcionar, al menos, los ángulos actuales de los motores y puede proporcionar opcionalmente el estado actual (en forma de GdL) de la plataforma de movimiento. Dado que no es el objetivo de este trabajo, para una descripción detallada del simulador, el lector puede consultar (Casas, Olanda et al. 2012, Casas, Alcaraz et al. 2014).

2.2. Sistema de Evaluación Objetiva

Puesto que la evaluación subjetiva es muy lenta, y no permite que realicemos un ajuste automático de los parámetros del algoritmo de *washout*, proponemos, en este trabajo, un sistema de evaluación objetiva de este tipo de algoritmos de control.

Como el cuerpo humano es sensible a la fuerza específica y a la velocidad angular, lo lógico es emplear métricas que analicen las 6 señales (F_x, F_y, F_z, ω_x, ω_y, ω_z) correspondientes a esas magnitudes físicas. El objetivo es comparar esas 6 señales físicas generadas por la plataforma de movimiento, con respecto a los valores (de esas mismas señales) que debería haber generado (idealmente) el algoritmo de *washout* al mover la plataforma de movimiento. El proceso se resume en la Figura 3: el módulo físico genera el estado físico del vehículo simulado y éste es comparado con el estado físico real en la posición del piloto, que se calcula aplicando el MCA sobre la plataforma a partir del citado estado físico simulado. Sin embargo, dado que el problema es realmente subjetivo, debemos diseñar métricas o indicadores que estén correlacionados con las opiniones de los pilotos. Las métricas propuestas están detalladas en (Casas, Coma et al. 2015), aunque los acrónimos están en inglés. Éstas son: error cuadrático medio (ECM), error absoluto medio (EAM), error cuadrático medio normalizado (ECMN), error absoluto medio normalizado (EAMN), escalado medio (EM), coeficiente de correlación de Pearson normalizado (CCPN) y retraso estimado (RE). Dado que no es el objetivo de este trabajo incidir en este aspecto, para una descripción detallada de estas métricas y su validación subjetiva, el lector puede consultar dicha referencia.

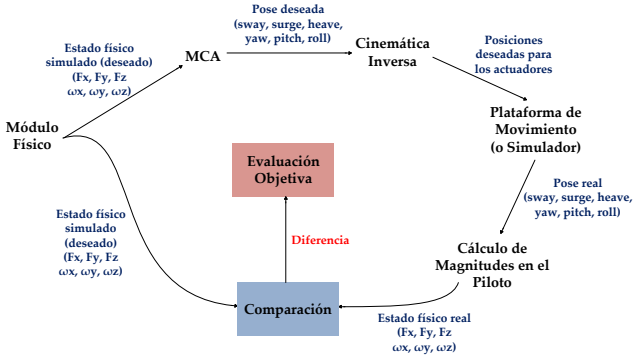

Figura 3: Esquema de evaluación propuesto para algoritmos de *washout*.

2.3. El Ajuste de Parámetros como Problema de Optimización

Puesto que disponemos de una métrica objetiva para evaluar algoritmos MCA, y una herramienta capaz de generar movimientos simulados sobre plataformas de movimiento, incluso más rápido que en tiempo real, podemos aprovechar dichas métricas y esta herramienta para probar sobre ella múltiples instancias de algoritmos MCA y evaluarlas de forma automática.

Por desgracia, dado que todo algoritmo de *washout* presenta una elevada cantidad de parámetros y éstos presentan rangos de variación grandes, el espacio de parámetros de un algoritmo de este tipo puede ser enorme. En el algoritmo clásico, por ejemplo, el número es superior a 30. ¿Cómo podemos hacer para encontrar en ese inmenso espacio el mejor conjunto de valores?

En algoritmia, el problema de encontrar la mejor solución, respecto a un determinado criterio, dentro de un conjunto de valores recibe el nombre de problema de optimización. Dependiendo del tamaño del conjunto, el problema se puede abordar de varias maneras. Si el conjunto es finito, el problema pasa a llamarse optimización combinatoria, ya que se reduce a encontrar la combinación óptima dentro de ese conjunto finito de combinaciones. Si es infinito, se llama problema de optimización continuo, y normalmente se resuelve discretizando el conjunto infinito y aplicando métodos de optimización combinatoria. Si además de finito, el conjunto no es muy grande, se puede plantear una búsqueda exhaustiva.

Normalmente, la búsqueda exhaustiva es computacionalmente intratable por su coste. Esto hace que debamos plantearnos otras alternativas, aunque proporcionen soluciones subóptimas. Entre los algoritmos de tipo subóptimo encontramos los algoritmos voraces (Ferri, Albert et al. 1998) y las heurísticas (Edelkamp and Schroedl 2011). Los algoritmos heurísticos de optimización son útiles en determinadas circunstancias (Díaz, Glover et al. 1996):

cuando la solución exacta no existe o es muy costosa de encontrar; cuando alcanzar la solución óptima no es estrictamente necesario y podemos conformarnos con soluciones cercanas, que podríamos llamar satisfactorias o subóptimas; cuando las funciones de evaluación que nos indican cómo de buena es una solución no son del todo fiables; cuando tenemos fuertes limitaciones temporales. Todas las anteriores circunstancias se dan en el problema que estamos intentando resolver, especialmente la no necesidad de la solución perfecta y la no fiabilidad de la función de evaluación, ya que tiene un substrato subjetivo importante. Por tanto, el uso de heurísticas está plenamente justificado para este problema.

En las siguientes secciones vamos a ver cómo podemos aplicar algoritmos de optimización para resolver el problema de asignar valores óptimos a los parámetros de los algoritmos de *washout*.

2.4. Formalización del Problema

Antes de intentar resolver el problema, vamos a intentar describirlo y formalizarlo de la mejor manera posible. Dispondremos de una instancia de un algoritmo de *washout*, que denominaremos *w* y que pertenecerá al conjunto de algoritmos de este tipo, llamado *W*. Cada algoritmo del conjunto *W* dispone de una serie de parámetros (en principio distintos en número, forma y semántica para cada algoritmo), cuyos valores modifican sustancialmente el comportamiento del mismo. Supondremos que todos los parámetros de todos los algoritmos del conjunto *W* son de tipo real (es lo habitual, y si son enteros o booleanos se pueden codificar fácilmente como reales), y supondremos que el número de parámetros del algoritmo *w*, que queremos optimizar, es *n*. Cada *t*-upla (de longitud *n*) de valores de parámetros hará que el algoritmo funcione de una manera distinta. Para evaluar cómo funciona dicho algoritmo con cada *t*-upla de valores, dispondremos de una función de evaluación *f*, que nos dirá para cada algoritmo y *t*-upla de parámetros si éste se comporta bien o no. Esta evaluación vendrá descrita en forma de número real positivo, de tal modo que a mayor valor de evaluación peor valoración. Las métricas descritas en el apartado 2.1, y calculadas según la Figura 3, son ejemplos de funciones de evaluación que cumplen ese requisito y que podríamos emplear.

El objetivo del algoritmo de optimización es encontrar una *t-upla p* de parámetros que minimice la función de evaluación para el algoritmo *w*. Como los parámetros casi siempre se asignan dentro de un cierto rango de valores razonable, podemos restringir el problema a buscar dentro de esos rangos. La especificación del algoritmo de optimización, restringiendo los valores de los parámetros a un cierto rango, es la siguiente (seguiremos el esquema y notación de Ferri (Ferri, Albert et al. 1998)):

Algoritmo **OptimizacionMCA**

Entradas:
w : W // algoritmo de washout (MCA)
n : N // número de parámetros modificables del algoritmo w
f : W, R^n -------> R // función de evaluación
li : vector [1..n] de R // límite inferior del rango de los parámetros
ls : vector [1..n] de R // límite superior del rango de los parámetros

Salidas:
p : vector [1..n] de R // t-upla óptima de parámetros
m : R // valor de evaluación de la t-upla óptima

Precondición:
$n > 0$
$\forall$ i=1..n : li[i] < ls[i]
$\forall x \in R^n, \forall w \in W$: f(w, x) ≥ 1

Postcondición:
$\forall x \in R^n : f(w, x) > m \mid w \in W,\ m = f(w, p),\ li[i] \le p[i] \le ls[i]$
con i= 1..n

Fin de OptimizacionMCA

Evidentemente, la evaluación del algoritmo de *washout* dependerá de sobre qué plataforma se ejecute, qué simulador genere los datos del estado físico y muchos otros factores. Es por ello que la palabra optimización hay que entenderla en el contexto adecuado. Se trata de optimizar el algoritmo de control para la función de evaluación elegida, la plataforma, el tipo de simulador y el vehículo empleado.

Como los parámetros son de tipo real, el número de valores posible para cada uno de los parámetros es infinito. Incluso restringiéndolos a un cierto rango de valores, el espacio de posibles soluciones sigue siendo infinito, por lo que su exploración exhaustiva es imposible. Por ello, debemos encontrar alguna manera inteligente de explorar el espacio de soluciones R^n.

Una alternativa consiste en discretizar los rangos de valores. Con ello conseguimos convertir el problema en un problema de optimización combinatoria. Aunque de esta manera el problema es más fácil de resolver, hacer una búsqueda exhaustiva de este espacio finito discretizado sigue siendo computacionalmente muy costoso, dado que habitualmente el espacio de soluciones es muy grande, a no ser que la discretización sea muy burda o el número de parámetros, muy pequeño.

El problema fundamental radica en que la consulta de la función de evaluación puede llegar a ser muy costosa, ya que lo habitual es testear los algoritmos MCA sobre una cierta ejecución más o menos larga del simulador. En un simulador de conducción correspondería, por ejemplo, a una vuelta del circuito, lo cual cuesta del orden de minutos, dependiendo del vehículo y el circuito. Dado que disponemos de un simulador de plataforma, podemos acelerar este proceso, pero si tenemos que hacer millones de evaluaciones (puesto que el número de parámetros es alto y el número de combinaciones, enorme), el problema es igualmente inabordable a nivel computacional.

Las soluciones voraces tampoco parecen aplicables porque no se conoce la forma del espacio de soluciones y no es factible que decisiones locales irreversibles nos lleven a soluciones globalmente óptimas. Por ello, debemos buscar soluciones heurísticas subóptimas que intenten minimizar el valor de la función de evaluación.

3. Esquemas Algorítmicos Propuestos

3.1. Búsqueda Exhaustiva Discretizada (BED)

La estrategia de búsqueda exhaustiva discretizada consistiría en recorrer todo el espacio de soluciones posible (previamente discretizado con un cierto nivel de detalle) para ver cuál es la mejor solución. Esto se corresponde con una estrategia de tipo *backtracking* (Ferri, Albert et al. 1998) sin poda, el coste temporal del cual es exponencial con el número de parámetros. De hecho, este tipo de problemas es NP-difícil (Kelley 1995). Aunque no es la mejor de las estrategias para recorrer un espacio tan grande, nos sirve como referencia y solución fácilmente validable para comparar el rendimiento de otras soluciones. Su implementación es relativamente sencilla y sólo posee un parámetro: el número de divisiones en que se trocean los rangos de los parámetros para discretizar el espacio continuo.

3.2. Monte-Carlo (MC)

Los algoritmos de Monte-Carlo son algoritmos de tipo probabilístico no determinista (Hammersley and Handscomb 1964). Su nombre indica su modo de funcionamiento, ya que funcionan seleccionando aleatoriamente valores para las posibles soluciones, para posteriormente hacer algún cálculo determinista sobre ellas. El esquema varía en función del tipo de problema.

En el caso que nos ocupa, la aplicación de este esquema intentaría evitar hacer una búsqueda exhaustiva del espacio de soluciones escogiendo elementos aleatorios de dicho espacio y evaluándolos. Si hacemos suficientes pruebas aleatorias, lo más probable es que recubramos de un modo bastante amplio, uniforme, y con un espaciado más o menos constante, el espacio de soluciones. Como la ejecución debe acabar en un tiempo finito, se debe poner un límite al número de configuraciones aleatorias que prueba el algoritmo, o un tiempo máximo. Una posible aproximación del método de Monte-Carlo a la optimización de parámetros en algoritmos MCA podría ser la siguiente:

Algoritmo **MonteCarloMCA**

Entradas:
w : W // algoritmo de washout (MCA)
n : N // número de parámetros modificables del algoritmo w
f : W, R^n -------> R // función de evaluación
li : vector [1..n] de R // límite inferior del rango de los parámetros
ls : vector [1..n] de R // límite superior del rango de los parámetros

Parámetros:
maxIters : N
maxTiempo : R

Auxiliar:
i : N
aleat : vector [1..n] de R
eval, tiempo : R

Salidas:
p : vector [1..n] de R // t-upla óptima de parámetros
m : R // valor de evaluación de la t-upla óptima

Algoritmo:

```
i = 0;
m = ∞;
tiempo = 0.0;
ponerRelojEnMarcha();

mientras ( (i < maxIters) ^ (tiempo < maxTiempo) ) hacer
{
   aleat = generarValoresAleatoriosEnRangos(n, li, ls);
   eval = f(w, aleat);

   si (eval < m) entonces
   {
      m = eval;
      p = aleat;
   }

   i = i +1;
   tiempo = consultarReloj();
}
```

Fin de MonteCarloMCA

El algoritmo de Monte-Carlo representa una mejora sobre la búsqueda exhaustiva ordenada ya que, como no conocemos qué forma tienen los valores de evaluación del espacio de soluciones, nada nos garantiza que recorrerlo en un cierto orden sea bueno. De hecho, lo más probable, es que si lo recorremos de modo aleatorio probemos soluciones muy distintas entre sí y alcancemos valores buenos (que no óptimos) más rápidamente, por lo que, en general, debe ser capaz de encontrar mejores soluciones que la búsqueda exhaustiva, dado el mismo tiempo de ejecución. Sin embargo, este método no nos evita tener que hacer muchas pruebas hasta encontrar un conjunto de parámetros lo suficientemente bueno. La condición de finalización es un tiempo máximo de búsqueda, puesto que es el criterio más lógico para este problema.

3.3. Recocido Simulado (RS)

Dado que el método de Monte-Carlo sigue ofreciendo poca inteligencia en la búsqueda de la solución óptima, el siguiente paso es idear una búsqueda más guiada. Aunque los valores de evaluación del espacio de soluciones no forman, a priori, ninguna forma conocida que podamos explorar analíticamente, es de esperar que cumplan el principio de localidad, dado que los algoritmos de *washout* funcionan habitualmente con filtros y elementos lineales. Es decir, dos *t*-uplas de parámetros que difieran poco entre sí (cuya distancia entre ellas sea pequeña) deberían inducir valores similares de la función de evaluación. Esto hace que los valores de evaluación del espacio de soluciones no tengan forma aleatoria, sino que presenten una forma más o menos continua con valles y montañas (aunque en muchas dimensiones).

Aprovechando esta idea, podemos desarrollar heurísticas para guiar el proceso de búsqueda. Existen múltiples heurísticas para la búsqueda de soluciones óptimas (Díaz, Glover et al. 1996, Alba and et al. 2009). De entre ellas las que creemos que se adaptan mejor a este tipo de problema son el Recocido Simulado y los algoritmos genéticos.

El Recocido Simulado (Simulated Annealing - SA) es una técnica heurística para buscar un mínimo (o máximo) global dentro de un espacio de búsqueda grande. El algoritmo está inspirado en el proceso de recocido del acero. La técnica de trabajo del acero consiste en calentarlo rápidamente para que sus átomos adquieran energía y se desplacen respecto de sus posiciones iniciales (que corresponden con mínimos locales de energía), para luego enfriarlo lentamente. El enfriamiento lento provoca que la probabilidad de formar configuraciones de energía más estables que la inicial aumente, hasta que en algún momento se alcance la configuración de mínima energía posible (lo que correspondería con un mínimo global) (Bertsimas and Tsitsiklis 1993, Díaz, Glover et al. 1996). Una posible aplicación de esta idea al ajuste de parámetros en MCA podría ser la siguiente:

Algoritmo **RecocidoSimuladoMCA**

Entradas:
w : W // algoritmo de washout (MCA)
n : N // número de parámetros modificables del algoritmo w
f : W, R^n -------> R // función de evaluación
li : vector [1..n] de R // límite inferior del rango de los parámetros
ls : vector [1..n] de R // límite superior del rango de los parámetros

Parámetros:
maxIters, maxReps : N
maxTiempo, tempInicial, tempFinal, K, tasaModificacion : R

Auxiliar:
sol1, sol2 : vector [1..n] de R
eval1, eval2, temp, delta, tiempo, aux, exp : R
i, numReps : N

Salidas:
p : vector [1..n] de R // t-upla óptima de parámetros
m : R // valor de evaluación de la t-upla óptima

Algoritmo:

```
i = 0;
numReps = 0;
temp = tempInicial;
```

```
tiempo = 0.0;
ponerRelojEnMarcha();
sol1 = generarValoresAleatoriosEnRangos(n, li, ls);
eval1 = f(w, sol1);
m = eval1;
p = sol1;

hacer
{
   sol2 = generarSolucionCercana(n, li, ls, sol1, tasaModificacion);
   eval2 = f(w, sol2);

   si (eval2 < m) entonces
   {
      m = eval2;
      p = sol2;
      numReps = 0;
   }
   si no
   {
      numReps = numReps + 1;
      si (numReps ≥ maxReps) entonces
      {
         sol1 = p;
         eval1 = m;
         numReps = 0;
      }
   }

   delta = eval1 – eval2;
   aux = (-delta*K)/temp;

   si (aux ≥ 0.0) entonces
      exp = 1.0;
   si no
      exp =e^aux;

   si (exp > generarValorAleatorioEnRango(0.0, 1.0)) entonces
   {
      sol1 = sol2;
      eval1 = eval2;
   }

   i = i +1;
   temp = ((tempInicial – tempFinal)*(maxIters - i))/maxIters;
   tiempo = consultarReloj();
}
mientras ( (i < maxIters) ^ (tiempo < maxTiempo) )

Fin de RecocidoSimuladoMCA
```

3.4. Algoritmo Genético o Darwiniano (AG)

La siguiente técnica heurística que vamos a adaptar a nuestro problema ha demostrado ser una de las más exitosas en la búsqueda de soluciones óptimas en espacios de soluciones muy grandes. Un algoritmo genético, evolutivo o Darwiniano (Holland 1992, Díaz, Glover et al. 1996) replica el funcionamiento evolutivo de los ecosistemas naturales codificando las soluciones como genes de una cadena de ADN, para encontrar una solución óptima (en realidad subóptima, porque es una heurística) a un problema. Aunque el término genético es el más empleado, el término evolutivo o incluso Darwiniano reflejan mucho mejor el funcionamiento del algoritmo. Dado que esta técnica es habitualmente conocida, para una descripción más detallada de este tipo de algoritmos, el lector puede consultar (Díaz, Glover et al. 1996).

El funcionamiento del algoritmo genético se puede adaptar a nuestro problema de la siguiente manera. Consideremos como posibles individuos cada *t*-upla *x* de valores para los parámetros de nuestro algoritmo de *washout w*. Codificaremos cada individuo como una secuencia ordenada de los valores de los parámetros: *x[1] x[2] … x[i] … x[n]*. Como función de *fitness* nos sirve nuestra función de evaluación *f*, que coge un algoritmo *w* y una asignación de parámetros, y lo evalúa.

Partiremos inicialmente de una población *P* de individuos (posibles soluciones) aleatorios. Evaluaremos todos los elementos de *P* y los ordenaremos en un vector según el valor creciente de dicha evaluación. Por efecto de la selección natural sólo sobrevivirá un porcentaje de ellos. El resto, morirá. Esto se traduce en que borraremos la parte final del vector de individuos ordenados.

Una vez el proceso de selección natural ha eliminado a los individuos menos aptos (las soluciones peores), los supervivientes se reproducirán dando lugar a nuevas soluciones (individuos). Dichos individuos deben contener material genético de ambos padres (parte de ambas soluciones). Además, en este punto se pueden producir pequeñas mutaciones que hacen que los hijos reciban los genes de los padres mejorados o empeorados (depende de la mutación). Las mutaciones pueden hacer al individuo más apto (mejor solución) o menos apto (peor solución).

Existen varias maneras de hacer la recombinación genética. La reproducción puede ser sexual (se asigna sexo a los padres) o asexual. Para nuestro problema, consideramos que lo más adecuado era una reproducción asexual, con selección aleatoria de padres y recombinación mediante mitosis (que la solución hija contenga la mitad de material genético de cada padre). Para ello, lo que haremos es escoger, cada vez, aleatoriamente, dos individuos para procrear. Haremos este proceso tantas veces como individuos murieran en la selección natural anterior, para reponer la población.

Para la recombinación de genes en la procreación, la propuesta es escoger de los *n* parámetros, aleatoria y equiprobablemente, o bien el valor que portaba el primer padre para ese parámetro o bien el que portaba el segundo padre. En promedio, cada hijo obtendrá la mitad de valores de cada padre.

Una vez combinados los genes de ambos padres para formar un nuevo individuo, pueden producirse mutaciones. En nuestro caso, la mutación (si ocurre) consistirá en cambiar aleatoriamente el valor de cada uno de los *n* parámetros. Se recorren todos y para cada uno, con cierta probabilidad se decide si hay mutación o no. Si se da la mutación, se generará un valor completamente nuevo para dicho parámetro dentro de su rango válido. Con ello, tenemos una nueva solución creada a partir de dos soluciones de la generación anterior pero ligeramente variada.

Posteriormente, todos los individuos padres mueren por vejez y los hijos constituyen la nueva generación, que es de esperar que contenga mejores soluciones que la anterior, ya que, en la anterior generación, las peores murieron por selección natural.

Una modificación (que no ocurre en la naturaleza) es la de añadir la posibilidad de que unos pocos individuos, con valores altos de la función de *fitness*, sobrevivan de generación en generación y nunca mueran. A esto se le llama elitismo. En nuestro caso, el elitismo se consideró una buena aportación, puesto que la evaluación del algoritmo *w* es muy costosa y si encontramos una muy buena solución (parametrización de *w*), esta no debería ser morir (ser descartada) incluso aunque tenga muchos hijos.

Algoritmo **DarwinianoMCA**

Entradas:
w : W // algoritmo de washout (MCA)
n : N // número de parámetros modificables del algoritmo w
f : W, R^n -------> R // función de evaluación
li : vector [1..n] de R // límite inferior del rango de los parámetros
ls : vector [1..n] de R // límite superior del rango de los parámetros

Parámetros:
tamPoblacion, maxIters : N
tasaMortalidad, tasaElitismo, probMutacion, maxTiempo: R

Auxiliar:
poblacion, padres : lista de {params: vector [1..n] de R, eval: R}
p1, p2, hijo : vector [1..n] de R
i, j, numMuertes, numVivos: N
tiempo, primero: R

Salidas:
p : vector [1..n] de R // t-upla óptima de parámetros
m : R // valor de evaluación de la t-upla óptima

Algoritmo:

```
i = 0;
m = ∞;
tiempo = 0.0;
ponerRelojEnMarcha();
poblacion = crearPoblacionAleatoria(tamPoblacion, li, ls);

hacer
{
    i = i + 1;
   // evaluar población
   desde (j=1 hasta tamPoblacion) hacer
      poblacion[j].eval =  f(w, poblacion[j].params);

   ordenarPorValorDeEvaluacion(poblacion);
   primero = poblacion[1].eval;

   si (primero < m) entonces
   {
      m = primero;
      p = poblacion[1].params;
    }

   // selección natural
   numMuertes = tamPoblacion*tasaMortalidad;
   eliminarUltimos(poblacion, numMuertes);
   padres = poblacion;

   // elitismo
   numVivos = tamPoblacion*tasaElitismo;
   numMuertes = tamaño(poblacion) – numVivos;
   eliminarUltimos(poblacion, numMuertes);

   // reproducción
   hacer
   {
      p1 = elejirPadreAleatoriamente(padres);
      p2 = elejirPadreAleatoriamente(padres);
      hijo = mitosis(p1, p2);

      // mutación
      desde (j=1 hasta n) hacer
         generarMutacion(hijo[j], probMutacion);

      añadir(poblacion, hijo);
   }
   mientras (tamaño(poblacion) < tamPoblacion)

   tiempo = consultarReloj();
}
mientras ( (i < maxIters) ^ (tiempo < maxTiempo) )

Fin de DarwinianoMCA
```

4. Evaluación de las Soluciones

4.1. Corrección de los Algoritmos

Antes de analizar el rendimiento de los algoritmos de optimización presentados, primero debemos comprobar que funcionan correctamente, es decir, que efectivamente encuentran el valor mínimo dentro de un determinado espacio de posibles soluciones. Dado que algunos de los algoritmos presentados son heurísticas, tiene poco sentido buscar una demostración formal de que encuentran el mínimo global, ya que sólo una búsqueda exhaustiva de grano muy fino puede garantizar eso. Nos conformaremos, pues, con una demostración empírica que muestre que son subóptimos.

Para ello, preparamos una sencilla prueba en la que, empleando el algoritmo de *washout* clásico, comprobamos si los distintos algoritmos de optimización eran capaces de dar como solución los valores esperados. Como el objetivo es simplemente comprobar la corrección de los algoritmos de optimización, utilizamos un modelo de plataforma de 3 GdL. El algoritmo clásico se puede implementar de diversas formas. Se emplea en este trabajo la versión de Reid y Nahon, puesto que es la más extendida y ampliamente utilizada (Reid and Nahon 1986). Dicha implementación, así como la semántica y funcionamiento de sus parámetros se detalla también en (Casas, Coma et al. 2015).

Tomamos como entrada del algoritmo clásico una señal sinusoidal para la fuerza específica en el eje Y de 10 segundos de duración, 1 Hz de frecuencia y 3 m/s^2 de pico, que vendría a ser la aceleración típica de un turismo. El resto de entradas del algoritmo clásico se fijaron a 0 (excepto la fuerza específica en el eje Z que en ausencia de aceleración queda en 9.8 m/s^2). Usando el indicador CCPN – porque es el que presenta mayor correlación con las opiniones de los pilotos – y permitiendo variar sólo 2 parámetros del algoritmo clásico: el límite de inclinación en grados del canal de coordinación angular entre 0 y 20º, y la frecuencia de corte del filtro pasa-baja del canal de coordinación angular en el eje Y, entre 0 y 10 Hz, los algoritmos de optimización propuestos deberían devolver valores cercanos a 10 Hz y a 20º, ya que, a mayor inclinación y mayor laxitud del filtro, el funcionamiento del algoritmo clásico debe ser, en este caso particular, mejor. El resto de parámetros del algoritmo clásico fueron fijados a valores considerados neutros (3 Hz para el resto de frecuencias de corte, 10 º/s para los límites de velocidad de inclinación, 10º para el resto de límites de inclinación, y 1 para el resto de parámetros) excepto el límite en la velocidad de inclinación en el eje Y, que fue eliminado para que este parámetro no limitara el resultado final y éste fuera más previsible. Al no haber más señales variables que la del eje Y, el indicador se calculó, en este caso, sólo para la fuerza específica en el eje Y.

Todas las pruebas se realizaron con un PC Intel Core 2 Quad Q8400 a 2.66 GHz, con 4 Gb de RAM y sistema operativo Windows 7. El simulador de plataforma, ajustado para correr sobre este equipo, permite alcanzar factores de aceleración del tiempo real bastante superiores a 1, lo cual permite ejecutar muchas más pruebas de las que se podrían realizar con una plataforma real. Los resultados obtenidos para cada uno de los algoritmos de optimización, con un tiempo de búsqueda de 500 segundos, se pueden observar en la Tabla 1.

Tabla 1: Validación de los algoritmos de optimización – 500 s

Algoritmo	Valor para		Mejor Indicador
	Frecuencia de Corte (Hz)	Límite de Inclinación (º)	
BED	9.99999	16.79999	1.06981
MC	9.98340	17.71427	1.06986
RS	9.99999	16.94035	1.06980
AG	9.97510	16.95966	1.06980

Como se puede observar, todos los algoritmos encuentran como valores óptimos números en torno a 10 Hz para la frecuencia de corte y en torno a 20º para el valor del límite de inclinación (valores cercanos al máximo de los intervalos de búsqueda proporcionados). Los valores obtenidos son los esperados, ya que

a mayor inclinación y mayor frecuencia de corte la simulación del movimiento, en este caso concreto, será más fidedigna. Como todos los algoritmos coinciden en sugerir los valores esperados y proporcionan valores similares para el indicador escogido, inferimos que el funcionamiento de los mismos es el correcto. Es reseñable que los valores óptimos no tienen por qué llegar al límite superior del intervalo porque, por ejemplo, para la inclinación, dado que la señal de entrada es sólo de 3 m/s^2, pasado un cierto valor, no se necesita más inclinación. Además, para esta prueba de corrección no se realizó un ajuste detallado de los parámetros de los algoritmos de optimización, eligiéndolos simplemente para que no cayeran en mínimos locales, sin preocuparnos de optimizarlos al máximo. Esto refuerza aún más su corrección, ya que, sin preocuparnos de ajustar sus parámetros, todos funcionaron de la forma esperada.

4.2. Estudio de Rendimiento

Búsqueda Exhaustiva Discretizada (BED)

El algoritmo de búsqueda ordenada o búsqueda exhaustiva discretizada es el más fácil de analizar. Si el número de parámetros variables del algoritmo de *washout* no es muy grande, este algoritmo es capaz de discretizar el espacio de soluciones total con un grano no muy grueso que, sin embargo, no requiere un tiempo de exploración muy elevado, por lo que es capaz de encontrar valores relativamente cercanos al óptimo. Dado que el único parámetro que posee este algoritmo es el número de divisiones de cada intervalo de parámetros del algoritmo de *washout*, a mayor número de divisiones, mayor coste tiene explorar todo el espacio. El coste computacional temporal es del orden de d^n, siendo n el número de parámetros y d, el número de divisiones. Este coste hace que este algoritmo sea, en la práctica, bastante poco útil, ya que, a poco que aumente n, el número de divisiones ha de hacerse tan pequeño para que el algoritmo termine en un tiempo razonable, que la exploración del espacio de soluciones es muy burda, lo que hace muy improbable que encuentre valores cercanos al óptimo.

La Tabla 2 resume el comportamiento del algoritmo para distintas combinaciones de tiempo y número de parámetros variables del algoritmo de *washout*. El número de divisiones se ajustó en cada prueba según el número de parámetros variables del MCA (a mayor, menor número de divisiones) y el tiempo disponible para su ejecución (a mayor, más divisiones) de forma que la finalización del algoritmo de optimización no fuera por tiempo máximo sino por fin de la búsqueda, aunque a partir de 8 parámetros el algoritmo no es capaz de terminar la búsqueda ni tan siquiera con 2 divisiones. En todas las pruebas se empleó el algoritmo clásico, el simulador de plataforma con el mismo computador que en el aparatado anterior, la misma plataforma de 3 GdL, y señales extraídas de un Formula 3 en base a una vuelta en Montmeló con el simulador *rFactor*. En este caso el indicador fue el CCPN, pero teniendo en cuenta los 6 GdL, y son los valores de este indicador los que se muestran en la Tabla 2.

Se puede observar que, cuando el número de parámetros es alto, incrementar mucho el tiempo de búsqueda prácticamente no mejora en nada el rendimiento del algoritmo, ya que en estos casos el problema es que no es capaz de explorar ni una pequeña parte del espacio de soluciones y el algoritmo acaba por superar el tiempo máximo, no por haber completado el rastreo completo del espacio de soluciones. Sin embargo, con 2 ó 4 parámetros, el algoritmo sí es capaz de realizar un barrido completo del espacio de soluciones en algunos casos con cierta solvencia y puede ser el más apropiado. Es por eso que, con este algoritmo de optimización, se da la curiosa circunstancia de obtener peores indicadores empleando un mayor número de parámetros variables del algoritmo MCA, ya que el efecto del aumento de parámetros no puede ser absorbido por la estrategia de búsqueda del algoritmo de optimización. Por ello, si las soluciones óptimas están al final del recorrido del espacio de búsqueda, este algoritmo fracasa estrepitosamente.

Tabla 2: Análisis del algoritmo de búsqueda exhaustiva

Tiempo	Número de parámetros			
(s)	2	4	8	18
100	1.28257644	1.30962312	1.32078445	1.47133744
200	1.28128842	1.30745662	1.31809866	1.45918723
300	1.28043871	1.30654892	1.31786573	1.43871265
500	1.27977326	1.30123901	1.31762381	1.42732394
1000	1.27528065	1.29993782	1.31744329	1.42689221
10000	1.27462733	1.29892326	1.31721387	1.42639518

Método de Monte-Carlo (MC)

Este método tiene la ventaja de no tener ningún parámetro que configurar, a excepción de la condición de terminación, que puede ser por número de iteraciones, por tiempo máximo o por convergencia entre soluciones, aunque lo lógico es poner sencillamente un límite de tiempo, por lo que la condición de terminación nunca la consideramos como un parámetro del algoritmo de optimización.

El método puede funcionar peor que la búsqueda exhaustiva cuando el número de parámetros es muy pequeño, aunque resulta una mejora sustancial de ésta cuando el número de parámetros comienza a aumentar. Sin embargo, a medida que este número aumenta, el algoritmo se pierde en un espacio de soluciones muy grande y empieza a ser muy improbable que encuentre una solución buena entre tantas posibilidades, ya que funciona de modo aleatorio. Precisamente por ser aleatorio, además, posee una cierta variabilidad, ya que puede encontrar una buena solución en poco tiempo de igual forma que puede transcurrir largo tiempo sin encontrar ninguna solución decente. En este último aspecto, el método anterior le aventaja, ya que aquél es completamente determinista.

La Tabla 3 ilustra el funcionamiento de este algoritmo de optimización. Al ser un algoritmo probabilístico, los resultados mostrados corresponden al indicador CCPN calculado como media de 20 pruebas de ejecución del mismo, en las mismas condiciones que la prueba realizada para el algoritmo anterior.

Tabla 3: Análisis del algoritmo de Monte-Carlo

Tiempo	Número de parámetros			
(s)	2	4	8	18
100	1.29799002	1.29566218	1.29468882	1.29676592
200	1.29602827	1.29438092	1.29435082	1.29454982
300	1.29589211	1.29248535	1.29128071	1.29129324
500	1.29496012	1.29012938	1.29000123	1.28917016
1000	1.29012397	1.28949325	1.28993732	1.28822884
10000	1.28588841	1.28812993	1.28927344	1.28713194
Desviación estándar máxima $\sigma_{máx} = 0.005213$				

Como vemos, el algoritmo se maneja relativamente bien con un número pequeño/medio de parámetros. Con 2 parámetros llega a

dar peores valores que la búsqueda ordenada, pero a partir de 4 parámetros la situación se invierte y los indicadores obtenidos son bastante mejores que los que se obtienen con el anterior algoritmo. Aunque las mejoras pueden parecer pequeñas en términos absolutos, una mejora de unas décimas en el indicador objetivo puede reflejar un cambio sustancial en la percepción, y pueden suponer un gran cambio en términos relativos.

Sin embargo, cuando el número de parámetros aumenta, el espacio de búsqueda es tan grande que el algoritmo no es capaz de encontrar grandes mejoras con tiempos de ejecución superiores y su rendimiento es prácticamente constante. La variabilidad, si bien existe, no es demasiado elevada, ya que la desviación estándar máxima de toda la serie se sitúa en 0.005213, lo cual en términos relativos representa en torno al 4% de los valores medios obtenidos.

Recocido Simulado (RS)

Este algoritmo es una mejora del algoritmo de Monte-Carlo ya que a diferencia del anterior no se basa solamente en el azar. En las pruebas realizadas (en las mismas condiciones que los anteriores algoritmos) se observa que es capaz de proporcionar mejores valores que el algoritmo de Monte-Carlo (ver Tabla 4). Los valores presentados son también valores medios de 20 pruebas.

Tabla 4: Análisis del algoritmo de Recocido Simulado

Tiempo	Número de parámetros			
(s)	2	4	8	18
100	1.30541287	1.29723823	1.30198491	1.29666865
200	1.29790291	1.29438362	1.30095403	1.29123833
300	1.29422071	1.29283022	1.29574667	1.28672632
500	1.29112762	1.28908371	1.29312938	1.28232801
1000	1.28672892	1.28763512	1.28653482	1.27652782
10000	1.28274593	1.28517803	1.28174512	1.27323125
Desviación estándar máxima $\sigma_{máx} = 0.017432$				

En este caso, se observa que su rendimiento mejora cuando hay un número alto de parámetros y sobre todo a mayor tiempo disponible. En cualquier caso, la mejora obtenida es bastante menor de la esperada. Por el contrario, con un número pequeño de parámetros y/o poco tiempo de ejecución obtiene resultados a veces peores que el algoritmo de Monte-Carlo.

Se observa, además, que la variabilidad de los resultados es relativamente alta. Esto es debido a que este tipo de algoritmos es propenso a quedarse estancado en mínimos locales y tiene cierta dependencia del valor inicial. La desviación estándar máxima de la serie se sitúa en 0.017432, lo cual representa alrededor de un 13% sobre los valores medios. Si bien, este dato es bastante elevado, también es cierto que la variabilidad disminuye a medida que se aumenta el tiempo de ejecución.

Todas las pruebas se realizaron con una constante de temperatura de 1000, una temperatura inicial de 0 ºC, una temperatura final de 100 ºC, una ratio de modificación de 0.2 y 5 repeticiones como número máximo de repeticiones. El número de iteraciones se ajustó para que coincidiera aproximadamente con el tiempo de ejecución. Estos valores se ajustaron con pruebas previas, que, por brevedad, no incluimos. Es importante recalcar que, el hecho de tener varios parámetros que ajustar y que su rendimiento sea bastante variable, son dos desventajas a tener en cuenta.

Algoritmo Genético o Darwiniano (AG)

Este algoritmo representa una alternativa al Recocido Simulado (que no se comporta sustancialmente mejor que los anteriores). Como este esquema ha demostrado ser útil en búsquedas en espacios de soluciones grandes, su aplicación a este problema debería ser provechosa.

Todas las pruebas se realizaron con una probabilidad de mutación de 0.2, con un tamaño de población de 20 individuos, con un elitismo del 20% y un porcentaje de selección natural del 40%. Estos valores se ajustaron con pruebas previas que, por brevedad, no incluimos. Sin embargo, el algoritmo se comportó bien para variaciones amplias de sus propios parámetros. Es importante recalcar que, el hecho de que sea bastante insensible a sus propios parámetros, es una ventaja grande a tener en cuenta. La condición de terminación es por tiempo máximo de búsqueda, como en el resto de esquemas. Las pruebas se realizaron en las mismas condiciones que los anteriores algoritmos.

Como se puede observar en la Tabla 5, el algoritmo genético obtiene buenos resultados en todo el espectro de situaciones, siendo especialmente efectivo cuando el número de parámetros es alto. Aunque sus ventajas se ponen de manifiesto cuando el tiempo de ejecución es relativamente alto, ya que su estrategia de exploración es mejor que la de los anteriores algoritmos, también se comporta bien con tiempos de ejecución pequeños y con pocos parámetros, aunque en este último caso las ventajas de este algoritmo con respecto a otros, se diluyen. Tanto es así, que para 2 parámetros, el algoritmo de búsqueda ordenada se comporta ligeramente mejor que éste. En cualquier caso, el hecho de que se comporte bien en todas las situaciones hace que sea el más idóneo, en cuanto a rendimiento, de los analizados. Al ser una heurística preparada para funcionar en espacios de soluciones grandes, esto no debería sorprendernos.

Tabla 5: Análisis del algoritmo genético

Tiempo	Número de parámetros			
(s)	2	4	8	18
100	1.28283178	1.28691323	1.29114046	1.28719175
200	1.28192831	1.28347992	1.28836286	1.27546432
300	1.28012839	1.28038452	1.28152083	1.27098391
500	1.27906072	1.27898394	1.27349029	1.26312893
1000	1.27639782	1.27472465	1.26992023	1.25172981
10000	1.27556721	1.27012297	1.26502301	1.24171954
Desviación estándar máxima $\sigma_{máx} = 0.005028$				

Además, la variabilidad del algoritmo es más baja que en el caso del anterior algoritmo, ya que, aunque tiene una componente aleatoria, tiende a converger hacia soluciones óptimas globalmente. Esto hace que los mejores indicadores se obtengan, claramente, cuando el algoritmo MCA presenta un mayor número de parámetros variables.

La desviación estándar máxima es similar a la del algoritmo de Monte-Carlo, si bien es cierto que ésta disminuye bastante a medida que aumenta el tiempo de ejecución, lo cual es un dato positivo.

4.3. Estudio Comparativo de Rendimiento

Para comparar todos los anteriores algoritmos y estudiar qué diferencias puede haber entre la optimización con distintos

indicadores, decidimos realizar un estudio comparativo exhaustivo.

Para la comparación, empleando el algoritmo clásico variando 18 de sus parámetros, empleando el mismo computador que en anteriores pruebas, el simulador *Live for Speed*, con un *BMW FB02* en el circuito de *Blackwood* y tomando 10000 segundos como tiempo de búsqueda, estudiamos el rendimiento de los 4 algoritmos propuestos con 2 plataformas de movimiento distintas y los siguientes indicadores: correlación (CCPN), error absoluto medio normalizado (EAMN), escalado medio (EM), retraso estimado (RE) y una combinación aditiva equiponderada de escalado medio, error absoluto medio normalizado y correlación. Los resultados se pueden ver en las siguientes tablas. La Tabla 6 muestra los resultados con respecto a la plataforma de 3 GdL que hemos estado empleando en las anteriores pruebas. La Tabla 7 muestra los resultados para una plataforma de 6 GdL de características similares en cuanto a potencia, pero con 3 tipos de movimiento más (*surge*, *sway*, *yaw*). Las características de ambos dispositivos se pueden consultar en (Casas, Coma et al. 2015).

De este estudio se pueden deducir varias cosas. La primera es que el indicador elegido afecta obviamente al resultado obtenido, aunque no parece afectar al rendimiento de los algoritmos, que dependen más de su estrategia de búsqueda y de sus problemas intrínsecos que del indicador, aunque es obvio que un cambio de indicador puede cambiar la forma del espacio de soluciones y alterar ligeramente el rendimiento de los diferentes algoritmos de optimización.

Tabla 6: Rendimiento comparado con plataforma de 3 GdL

Alg.	Indicador				
	CCPN	EAMN	EM	RE	Combin.
BED	1.416395	1.177793	433.3949	2.108332	846.40723
MC	1.286132	1.168325	16.75626	1.163333	37.001137
RS	1.283231	1.169000	14.17908	1.158333	32.581150
AG	1.251236	1.165731	9.183581	1.105000	16.566420

Tabla 7: Rendimiento comparado con plataforma de 6 GdL

Alg.	Indicador				
	CCPN	EAMN	EM	RE	Combin.
BED	1.412334	1.178764	22.70326	1.188333	40.585899
MC	1.294718	1.171918	4.973333	1.153333	9.9357395
RS	1.296791	1.171614	4.253120	1.146667	8.0651226
AG	1.264514	1.167899	3.667997	1.078333	5.2865601

La segunda conclusión es que la diferencia entre 3 y 6 GdL no es muy grande en términos de correlación ni de diferencia de señales. De hecho, en algunas ocasiones saca mejor valoración la plataforma de 3 GdL. Sin embargo, el uso de una plataforma de 6 GdL da mejores resultados en cuanto al escalado medio de las señales. Esto es lógico si tenemos en cuenta que en el cálculo de los diferentes indicadores se incluyen todas las señales y en la plataforma de 3 GdL hay 1 GdL que no puede simularse de ninguna manera y 2 que están más limitados que en la de 6. El indicador combinado también incluye este efecto. El retraso también es ligeramente mejor con la de 6 GdL, pero habría que analizar con mucho cuidado si estas pequeñas mejoras que esta plataforma proporciona (con respecto a otras de menos GdL), compensan a nivel de inversión económica.

En cuanto a los algoritmos de optimización, el algoritmo genético es el que mejor funciona en casi todos los casos. El de búsqueda ordenada es el peor en todos los casos. El algoritmo de Monte-Carlo se comporta ligeramente peor que el de Recocido Simulado, pero, al no tener parámetros propios que ajustar para ser ejecutado, puede ser una alternativa. Hemos de recordar que esta prueba se realizó con 18 parámetros del algoritmo MCA, que no es la condición más adecuada para el algoritmo de Monte-Carlo. El algoritmo de Recocido Simulado tiene más parámetros propios que el genético, es mucho más sensible a ellos y, además, tiene un rendimiento peor que el Darwiniano.

5. Conclusiones y Trabajo Futuro

Del trabajo presentado en este artículo, pueden extraer varias conclusiones. En cuanto a la metodología de ajuste, las pruebas nos sugieren que el ajuste automático aquí presentado puede ser una alternativa respecto al ajuste tradicional y permite ahorrar tiempo, aunque no debemos plantearlo como un sustituto completo del ajuste subjetivo manual sino como un complemento, ya que todo depende del tiempo que dispongamos para el ajuste.

En cuanto a los métodos de optimización, la principal conclusión es que el algoritmo genético es el que mejor funciona con espacios de soluciones muy grandes. El mayor problema de este algoritmo es que tiene varios parámetros propios, y aunque no son muchos y sus valores pueden fluctuar en rangos más o menos grandes sin que el algoritmo deje de funcionar bien, sería más interesante que no tuviéramos que ajustar ningún parámetro, ya que eso es precisamente lo que queremos evitar de los algoritmos MCA. En cualquier caso, es el que demuestra una mayor inteligencia a la hora de buscar soluciones, y es capaz de encontrar soluciones bastante buenas en muy pocas iteraciones, cosa que en otros algoritmos no siempre sucede. Su uso compensa cuando hay muchos parámetros que ajustar.

El resto de algoritmos presentan algunos problemas. El Recocido Simulado tiene un rendimiento no demasiado brillante con este problema. Es claramente peor que el algoritmo genético y no tiene muchas ventajas adicionales respecto a éste, ya que presenta varios parámetros propios que configurar y su rendimiento es el menos estable de todos. La explicación a su no buen rendimiento puede estar en que este método es muy propenso a caer en mínimos locales (cristalización prematura) si los parámetros de enfriamiento y temperatura no se ajustan con mucho acierto, cosa que no es sencilla y de tener que hacerlo desvirtuaría el objetivo pretendido, ya que el objetivo es realizar un ajuste lo más desatendido posible y con este método se deben ajustar varios parámetros del algoritmo de optimización, siendo, además, bastante sensible a ellos, cosa que no ocurre en tan gran medida con otros algoritmos evaluados.

El algoritmo de Monte-Carlo puede ser utilizado y tiene la ventaja de no tener parámetros. Aunque funciona, puede tardar demasiado tiempo en ofrecer una solución suficientemente buena y, a diferencia de otros algoritmos, la solución no se va refinando asintóticamente, sino que es puramente aleatoria.

Respecto a los indicadores, hemos podido comprobar que no afectan a los métodos de optimización, cosa que tampoco ocurre con un cambio de plataforma. Además, las diferencias entre los indicadores calculados para las dos plataformas analizadas son más pequeñas de lo que podría pensarse en un principio, ya que resulta más importante que la plataforma sea capaz de generar los movimientos de forma rápida y con una extensión razonable, que tener muchos GdL tan poco extensos que sean prácticamente inservibles. La conclusión más importante es, pues, que la rapidez de movimiento es más importante que el número de GdL.

En cuanto al trabajo futuro, en esta línea, podemos plantear muchas propuestas, como analizar detalladamente si el ajuste automático puede sustituir completamente al ajuste manual subjetivo, y en qué condiciones. También sería interesante analizar otro tipo de indicadores, otro tipo de plataformas robóticas, otro tipo de algoritmos de optimización, o realizar un estudio detallado de la influencia de los parámetros de estos algoritmos sobre el resultado final. También sería interesante probar el esquema propuesto con otros algoritmos de control/*washout* distintos al algoritmo clásico o incluso empleando otro tipo de simuladores. Por último, podría ser interesante también hacer una evaluación objetiva, pero perceptual. Es decir, evaluar la validez perceptual (Reymond and Kemeny 2000) del movimiento generado en lugar de la validez física. Esto implica generar modelos objetivos de percepción, lo cual complica el estudio aún más, pero es una línea de trabajo interesante porque estos manipuladores robóticos se construyen para generar claves gravito-inerciales perceptuales con el objetivo de que sean percibidas como correctas por los usuarios.

English Summary

Heuristics for solving the parameter tuning problem in motion cueing algorithms.

Abstract

Motion cueing algorithms are commonly used in vehicle simulators to control robotic motion platforms. These algorithms usually have a significant number of parameters that need to be tuned. This process has been traditionally performed in a pilot-in-the-loop subjective manner. The authors propose a simulation-based objective and automatic method using heuristic optimization. Several schemes are proposed, assessed and compared in this paper, showing that a genetic algorithm is the one that suits best this problem.

Keywords:

Motion platforms, heuristics, simulation, motion cueing algorithms, tuning, robotics, optimization.

Referencias

Alba, E., Blum, C., Asasi, P., Leon, C., Gomez, J. A., 2009. Optimization techniques for solving complex problems. New Jersey, NJ, USA, Wiley.

Bertsimas, D., Tsitsiklis, J., 1993. Simulated Annealing. Statistical Science 9 (1), 10–15.

Casas, S., Alcaraz, J. M., Olanda, R., Coma, I., Fernández, M., 2014. Towards an extensible simulator of real motion platforms. Simulation Modelling Practice and Theory 45 (0), 50–61.

Casas, S., Coma, I., Riera, J. V., Fernández, M., 2015. Motion-Cuing Algorithms: Characterization of Users' Perception. Human Factors: The Journal of the Human Factors and Ergonomics Society 57 (1), 144–162.

Casas, S., Olanda, R., Fernandez, M., Riera, J. V., 2012. A faster than real-time simulator of motion platforms. CMMSE, Murcia, Spain.

Casas, S., Rueda, S., Riera, J. V., Fernández, M., 2012. On the Real-time Physics Simulation of a Speed-boat Motion. GRAPP/IVAPP.

Colombet, F., Dagdelen, M., Reymond, G., Pere, C., Merienne, F., Kemeny, A., 2008. Motion Cueing: what's the impact on the driver's behaviour? Driving Simulator Conference, Monte-Carlo, Monaco.

Cossalter, V., Lot, R., Massaro, M., Sartori, R., 2011. Development and Validation of an Advanced Motorcycle Riding Simulator. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering 225 (6), 705–720.

Díaz, A., Glover, F., Ghaziri, H. M., González, J. L., Laguna, M., Moscato, P., Tseng, F. T., 1996. Optimización Heurística y Redes Neuronales. Madrid, Paraninfo.

Edelkamp, S., Schroedl, S., 2011. Heuristic Search: Theory and Applications. Waltham, MA, USA, Morgan Kaufman - Elsevier.

Ferri, F. J., Albert, J. V., Martín, G., 1998. Introducció a l'Anàlisi i Disseny d'Algorismes. Valencia, Spain, Publicacions de la Universitat de València.

Garrett, N. J. I., Best, M. C., 2010. Driving simulator motion cueing algorithms – a survey of the state of the art. Proceedings of the 10th International Symposium on Advanced Vehicle Control (AVEC), Loughborough, UK.

Go, T. H., Bürki-Cohen, J., Chung, W. H., Schroeder, J. A., Saillant, G., Jacobs, S., Longridge, T., 2003. The Effects of Enhanced Hexapod Motion on Airline Pilot Recurring Training and Evaluation AIAA Modeling and Simulation Technologies Conference and Exhibit, Austin, TX, USA.

Grant, P. R., 1996. The Development of a Tuning Paradigm for Flight Simulator Motion Drive Algorithms. PhD, University of Toronto.

Grant, P. R., Reid, L. D., 1997. Motion Washout Filter Tuning: Rules and Requirements. Journal of Aircraft 34 (2), 145–151.

Gutridge, J., 2004. Three Degree-of-Freedom Simulator Motion Cueing Using Classical Washout Filters and Acceleration Feedback. Master Thesis, Virginia Polytechnic Institute & State University.

Hammersley, J. M., Handscomb, D. C., 1964. Monte Carlo Methods. London, UK & New York, NY, USA.

Holland, J. H., 1992. Genetic Algorithms. Scientific American 267, 66–72.

Kelley, D., 1995. Automata and Formal Languages: An Introduction. New Jersey, NJ, USA, Prentice Hall.

Korobeynikov, A. V., Turlapov, V. E., 2005. Modeling and Evaluating of the Stewart Platform. International Conference Graphicon.

MacNeilage, P. R., Banks, M. S., Berger, D. R., Bulthoff, H. H., 2007. A Bayesian model of the disambiguation of gravitoinertial force by visual cues. Experimental Brain Research, 179 (2), 263-290.

Merlet, J. P., 2006. Parallel robots. The Netherlands, Springer.

Nahon, M. A., Reid, L. D., 1990. Simulator motion-drive algorithms - A designer's perspective. Journal of Guidance, Control, and Dynamics 13 (2), 356–362.

Reid, L. D., Nahon, M. A., 1985. Flight Simulation Motion-Base Drive Algorithms: Part 1 - Developing and Testing the Equations. University of Toronto, UTIAS. 296.

Reid, L. D., Nahon, M. A., 1986. Flight Simulation Motion-Base Drive Algorithms: Part 2- Selecting the System Parameters. University of Toronto, UTIAS. 307.

Reid, L. D., Nahon, M. A., 1986. Flight Simulation Motion-Base Drive Algorithms: Part 3 - Pilot Evaluations. University of Toronto, UTIAS.

Reymond, G., Kemeny, A., 2000. Motion Cueing in the Renault Driving Simulator. Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility 34, 249–259.

Schmidt, S. F., Conrad, B., 1969. The Calculation of Motion Drive Signals for Piloted Flight Simulators. Palo Alto, CA, USA, NASA. 69–17.

Sinacori, J. B., 1977. The Determination of Some Requirements for a Helicopter Flight Research Simulation Facility. CA, USA, Moffet Field. 7805.

Slob, J. J., 2008. State-of-the-Art Driving Simulators, a Literature Survey. Eindhoven, The Netherlands, Eindhoven University of Technology.

Stewart, D., 1965. A Platform with six degrees of freedom.

Stahl, K., Abdulsamad, G., Leimbach, K., Vershinin, Y. A., 2014. State of the Art and Simulation of Motion Cueing Algorithms for a Six Degree of Freedom Driving Simulator. Paper presented at the 17th International Conference on Intelligent Transportation Systems (ITSC).