CAPÍTULO 16

QSEARCH: BÚSQUEDA OPTIMIZADA APLICADA A LA CALIBRACIÓN DE CONTROLADORES PID EN UN CUADRICÓPTERO AUTÓNOMO

O. HIGUERA, C. AGÜERO, J. M. CAÑAS y S. MILLÁN

Grupo de Robótica, Universidad Rey Juan Carlos
{ohiguera, caguero,jmplaza}@gsyc.urjc.es

En este capítulo se presenta un algoritmo de búsqueda optimizada aplicado a la calibración de los bucles de control PID de un UAV (*Unmanned Aerial Vehicle*) de cuatro motores. El objetivo es encontrar una parametrización PID que ofrezca estabilidad de vuelo con el mínimo número de casos de prueba.

1 Introducción

Cuando se desea resolver un problema de optimización del cual no se tiene una descripción matemática, o aún teniéndola no es posible aplicar (o no son efectivos) los métodos analíticos de resolución conocidos, se puede recurrir a los métodos de búsqueda optimizada.

Los algoritmos de búsqueda optimizada son métodos computacionales que intentan optimizar un problema a base de ir eligiendo posibles soluciones dentro del espacio de búsqueda del problema en concreto, y apoyándose en una función objetivo que indica cómo de buena es la solución seleccionada. Hasta ahí la descripción encaja con cualquier algoritmo de búsqueda usual, el secreto de los algoritmos de búsqueda optimizada es que consiguen explorar el espacio de búsqueda eficientemente, de modo que encuentren una solución al problema antes de lo que lo haría un método aleatorio.

El caso de la parametrización de controladores de lazo cerrado PID es un caso concreto de problema de optimización. El objetivo del problema es elegir los valores óptimos de las variables que parametrizan el controlador, *Kp* (Proporcional), *Ti* (Integral) y *Td* (Derivativo). Para la correcta calibración de un PID normalmente es necesario contar con un modelo de simulación del proceso que permita escoger los valores basándose en el modelo dinámico. Sin embargo, en ocasiones no se puede contar con dicho modelo (ya sea por complejidad o limitaciones en recursos). En estos casos se puede optar por realizar una búsqueda optimizada.

Para ello se define el espacio de soluciones que constituye el espacio de búsqueda, en este caso es un espacio tridimensional formado por todos los posibles valores de *Kp*, *Ti* y *Td*. A la hora de realizar la búsqueda se cuenta con la gran restricción del coste de calcular la calidad de la solución seleccionada, ya que la función de calidad no es instantánea. Para calcular la calidad de una solución es necesario realizar una medida del comportamiento del modelo que puede incluir diferentes factores, tales como tiempo de reacción, oscilación, precisión y muchos otros indicadores que deben ser medidos con el sistema en marcha durante un tiempo determinado.

Por esa razón el objetivo de este algoritmo de búsqueda optimizada no es encontrar la mejor solución, o una solución con un grado de resolución dado, sino encontrar una solución suficientemente buena con el mínimo número de candidatos seleccionados. En el caso de la estabilización de un UAV de cuatro motores la solución debe permitir una salida estable, mínima oscilación y mínimo tiempo de reacción ya que este control es crítico. Para ello es primordial una gran eficiencia en la exploración del espacio de soluciones y un alto nivel de convergencia del algoritmo de búsqueda, dándole menor valor a otros factores como gran resolución de la solución, o conocimiento de los diferentes máximos/mínimos locales, etc.

2 Trabajos previos en búsqueda optimizada

Cuando surgió el problema de la optimización de los parámetros PID, se realizó un estudio sobre los diferentes métodos de optimización existentes para examinar qué métodos podían ser aplicados y si eran prácticos en el caso concreto de nuestro problema.

2.1 Métodos Numéricos

Existen diferentes tipos de algoritmos de optimización (Rardin, 1997), (Arora, 1989). Cuando el problema en cuestión puede ser modelado matemáticamente pero la resolución analítica es complicada se pueden usar métodos de optimización numéricos, basados en la aproximación usando diferentes vertientes, tales como el método de Newton, el método del Gradiente, o diferentes combinaciones que aprovechan las virtudes de ambos métodos.

Estos métodos necesitan que los problemas sean derivables, ya que precisan del cálculo de Gradientes y Hessianas para conocer hacia dónde buscar. Otros métodos especulan sobre las propiedades de concavidad/convexidad del problema para poder atacarlo.

Sin embargo, cuando el problema no es derivable, o no es posible conocer las propiedades de optimalidad del mismo es necesario recurrir a métodos que acotan los límites de búsqueda, pero son en su mayoría poco eficientes y muy costosos computacionalmente. Cuando ni siquiera es posible contar con un modelo matemático del problema este tipo de métodos se descarta.

Si no contamos con información suficiente como para modelizar el problema no queda más remedio que usar métodos de búsqueda optimizada, comúnmente llamados metaheurísticos (Blum, 2003), ya que consisten en seleccionar una serie de soluciones candidatas y aplicar sobre las mismas una función objetivo que ofrece información sobre la calidad de la solución. Entre estos métodos podemos encontrar variantes de filtros de partículas, algoritmos genéticos (Eiben, 1994), (Goldberg, 1989), el algoritmo PSO (*Particle Swarm Optimization*), u otros recientes algoritmos como el *Cuckoo Search* (CS).

2.2 Algoritmos basados en filtros de partículas

Los filtros de partículas se basan en el concepto de muestra estadística. Para explorar el problema utilizan una muestra. El conjunto muestral consiste en una población de partículas generadas que se repartirán de una manera pseudoaleatoria por todo el espacio de búsqueda. En lugar de aplicar la función objetivo sobre todo el conjunto de soluciones se aplica a cada una de las partículas (muestra). Una vez evaluada la población de partículas, se mueven o se generan nuevas partículas siguiendo una ley matemática de algún tipo, de manera que con las siguientes iteraciones el algoritmo converge en probabilidad al óptimo. Dicho de otra manera, al aumentar las ite-

raciones la probabilidad de contar con la solución óptima entre la población de partículas seleccionadas tiende a uno.

El caso de los algoritmos genéticos es un caso particular de un algoritmo basado en muestras. En este caso se hace una analogía entre el conjunto de soluciones (llamado fenotipo) y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena binaria (llamada cromosoma). En cada iteración los cromosomas son evaluados siguiendo una medida de aptitud. Las siguientes generaciones se forman usando una serie de operadores genéticos. Básicamente tras cada iteración se seleccionan los cromosomas que han obtenido mejor aptitud para realizar operaciones con ellos. Dos de los operadores genéticos más importantes son el sobrecruzamiento, que consiste en coger lo mejor de dos cromosomas y generar dos nuevos (mejores que los padres), y la mutación, que consiste en variar aleatoriamente algunos cromosomas de modo que se exploren partes del espacio de búsqueda aún no alcanzadas.

El algoritmo PSO (Kennedy, 1995), (Shi, 1998) (*Particle Swarm Optimization*) es otro caso particular de algoritmo de partículas en el cual a cada partícula se le otorga aleatoriamente un vector de movimiento. En la primera iteración las partículas se reparten uniformemente por todo el espacio de búsqueda, se evalúa la calidad de cada partícula y se elige la posición de la mejor partícula y la posición de toda la población (llamado enjambre/swarm). En la siguiente iteración cada partícula se moverá siguiendo su vector de movimiento, calculado en función de la posición anterior y la calidad de la partícula, pero teniendo en cuenta también la mejor solución encontrada tal y como se puede ver en la Ecuación (1). Una vez desplazadas se calculará de nuevo la calidad de cada partícula y se actualizará (si se da el caso) la posición de la mejor partícula.

$$v_i \leftarrow \omega \cdot v_i + \varphi_p \cdot r_p \cdot (p_i - x_i) + \varphi_p \cdot r_p \cdot (g - x_i) \qquad (1)$$

Donde ω, φ y r son constantes que parametrizan el algoritmo para que se ajuste a un espacio de soluciones concreto, x es la posición de la partícula, p es la calidad de la partícula y g es la calidad de la mejor partícula del enjambre.

El algoritmo PSO mueve las partículas teniendo en cuenta la mejor partícula encontrada de manera que toda la población tiene un factor de movimiento común, como si de un enjambre se tratara (de ahí el nombre).

2.3 *Cuckoo Search*

El algoritmo *Cuckoo Search* (Yang, 2009) (en adelante CS) es un nuevo tipo de algoritmo de muestras basado en la aplicación de metaheurísticas provenientes de la naturaleza. Como muchos otros algoritmos de optimización, las dos características principales de estas metaheurísticas son la diversificación y la intensificación. La intensificación se refiere a la capacidad de buscar siempre cerca de las mejores soluciones, mientras que la diversificación se refiere a la capacidad de explorar todo el espacio de búsqueda eficientemente.

Básicamente CS actúa esparciendo una población de partículas a lo largo del espacio de búsqueda de manera aleatoria, se califican las soluciones y se queda con las mejores soluciones, una parte de las peores soluciones son descartadas y sustituidas por nuevas soluciones que se generan y el resto de partículas se mueven siguiendo una distribución Lèvy.

Para el desarrollo de CS, sus autores se han fijado en el comportamiento de los cucos a la hora de anidar y en el vuelo característico denominado Lèvy Flights (Pavlyukevich, 2007) de algunos pájaros y moscas de la fruta a la hora de alimentarse. Los cucos son pájaros conocidos por su comportamiento parásito a la hora de anidar, depositan sus propios huevos en nidos ajenos de modo que sean otros pájaros los que se encarguen de cuidar y alimentar a sus polluelos. En ocasiones los huéspedes descubren que se trata de un intruso y echan al huevo intruso o abandonan el nido.

Por otro lado, algunas moscas de la fruta y aves muestran un comportamiento en su vuelo que se ajusta a los denominados Lèvy Flights. Este tipo de vuelo se caracteriza por largos recorridos en línea recta seguidos de bruscos cambios de dirección con giros de 90 grados. La dirección del giro se distribuye uniformemente y el tamaño del incremento está distribuido de acuerdo a una distribución de Lèvy de la forma $y = x^{-a}$ donde $1<a<3$.

Para CS cada solución del espacio de búsqueda es un nido, de los cuáles se elegirán una población de nidos en los que depositar huevos, que se corresponden con las soluciones candidatas. La función objetivo pone nota a la calidad de cada nido, en cada iteración se descarta una fracción P_a de los peores nidos, y se seleccionan nuevos nidos a través de Lèvy Flights, de esta manera la población de nidos permanece constante.

La potencia del algoritmo se basa en usar vuelos Lèvy para explorar el espacio de soluciones y para añadir aleatoriedad a la selección de nuevas soluciones candidatas.

Aunque todos los algoritmos revisados generan soluciones muy buenas, ninguno asegura que pueda encontrar el máximo global en todos los casos. Además, aunque cuentan con estrategias de exploración del espacio de búsqueda eficientes, necesitan grandes cantidades de pruebas de soluciones candidatas, ya que al tratarse de algoritmos iterativos es necesario realizar varias iteraciones para que las partículas converjan. Si contamos con una población de partículas de tamaño *n* constante, necesitaremos de *n * Iteraciones* ejecuciones de la función de calidad, lo que en el caso que nos ocupa puede llegar a ser un número demasiado grande.

3 Descripción del algoritmo QSearch

El algoritmo QSearch parte del intento de usar el *Cuckoo Search* para el caso de la calibración de PIDs. A la hora de realizar las primeras pruebas constatamos que el número de ejecuciones de la función de calidad necesitadas por el *Cuckoo Search* suponía una limitación debido al tiempo consumido por cada ejecución.

Adicionalmente, a la hora de analizar los algoritmos existentes, todos ellos hacían hincapié en la precisión con la que se acercaban a los máximos teóricos en las pruebas que ejecutaban, pero a costa del número de ejecuciones de la función de calidad. Sin embargo, para nuestro objetivo no es tan importante la resolución de la solución como minimizar el número de ejecuciones necesarias para llegar a una resolución mínima.

Como ya se ha comentado anteriormente, un algoritmo de búsqueda optimizada debe cumplir dos características principales, la diversificación o exploración del espacio de búsqueda eficientemente, y la intensificación o la capacidad de aprovechar el principio de localidad espacial. Esta última se refiere a la característica de continuidad de las funciones sobre las que se aplica el algoritmo que hace que los máximos locales o el máximo global esté localizado espacialmente muy cerca de soluciones muy cercanas a dicho máximo, por continuidad.

Para realizar una exploración eficiente del espacio de soluciones, la idea de usar paseos aleatorios (*random walks*) como los obtenidos por Lèvy Flights nos pareció muy buena.

Un vuelo Lèvy (Fig. 1) es un paseo aleatorio que sigue una distribución de Lèvy, de la forma $y = x^{-a}$, como ya se ha mencionado. Los vuelos Lèvy son procesos de Markov, muy útiles para modelizar datos que exhiben el agrupamiento, y usados para modelar fenómenos naturales al azar. Se caracterizan por largos saltos seguidos de pequeños saltos con cambios

bruscos de dirección, y modelan muy bien comportamientos de búsqueda en el mundo animal. Tiburones, moscas, y aves presentan este tipo de patrón de búsqueda de alimentos cuando no consiguen buenos resultados con los patrones estándar.

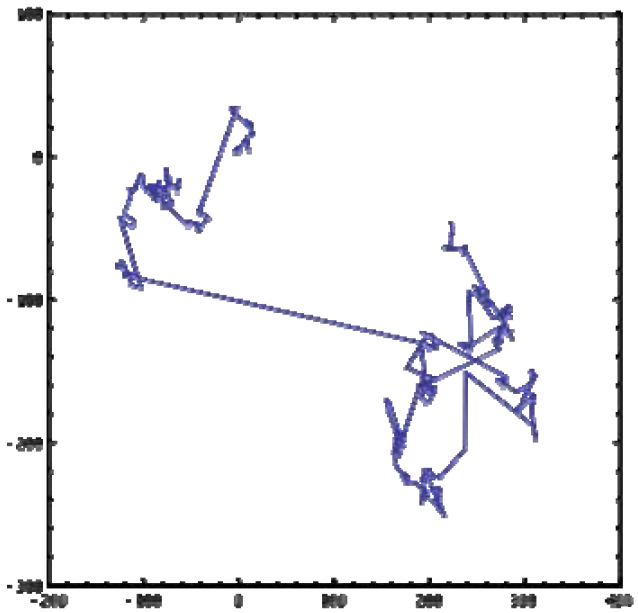


Fig. 1. Vuelo Lèvy convencional

QSearch utiliza algunas características del vuelo Lèvy, pero la longitud de los saltos utilizada sigue una distribución exponencial del tipo $y = x^a$, por lo que a la trayectoria la hemos llamado vuelo Lèvy invertido (a partir de ahora vuelo invertido).

Además no se trata de un filtro de partículas, puesto que deseamos minimizar el número de ejecuciones, tan solo existirá una única partícula que se moverá siguiendo un vuelo invertido de longitud limitada. Para que sea efectivo, cada nuevo vuelo debe iniciarse desde la posición de la mejor solución encontrada hasta el momento, entonces buscará cerca de esta posición, y si no consigue mejorar el resultado realizará algunos saltos muy lejos, se alejará exponencialmente con cambios de dirección de 90 grados y sentidos aleatorios hasta encontrar una mejor solución o llegar al límite de saltos.

La dirección a tomar por el vuelo cambia en cada salto, siendo siempre normal al vector que caracterizó el salto anterior. Cada vez que un nuevo salto encuentra una solución que mejora a la actual el vuelo se reinicia, de modo que se vuelve a intentar buscar cerca de dicho punto.

$$StepLength = \alpha \cdot i^{\delta} \tag{2}$$

La longitud del salto viene dada por la Ecuación (2), donde $\alpha > 0$ es una constante de proporcionalidad asociada a las dimensiones del espacio de soluciones y a la resolución que es capaz de alcanzar la búsqueda. δ tendrá un valor entre 1 y 3 siendo la constante que define cómo de grande será el siguiente salto respecto al anterior, parametrizando cómo de rápido crece la longitud de los saltos de un vuelo. Finalmente, *i* es la variable que indica la iteración del vuelo en la que se encuentra el algoritmo, comenzará siendo 1 y se incrementará con cada salto, al reiniciar el vuelo volverá a tomar valor 1. La longitud del salto sigue una distribución con varianza infinita, donde la distancia del origen de un vuelo al azar tiende a una distribución estable, además el escalamiento de los saltos da al vuelo una propiedad de escala invariante.

Una vez definida la longitud del salto se calcula la dirección del mismo. Obtenemos un nuevo vector unitario que nos defina la dirección a seguir, este vector lo llamaremos U_{CN} puesto que viene dado por el punto *C* (*Current*/Actual) que es el punto de origen del salto y el punto *N* (*New*/Nuevo) que es el nuevo punto correspondiente a la nueva solución a probar (será sometido a la función de calidad). Para ello partimos del vector *OC* que se corresponde con el vector que definió el salto anterior y que está definido por el punto *O* (*Old*/Antiguo) que es el punto de origen del salto anterior, y el punto *C* (*Current*) que es el punto de origen del salto actual (y se corresponde con la última solución probada, o destino del último salto realizado).

Como U_{CN} debe ser perpendicular a *OC*, para conseguir un vector normal a *OC* tan solo se debe seleccionar cualquier vector del espacio y realizar el producto cruzado de este vector cualquiera con *OC*. Para seleccionar este vector para el producto cruzado podemos elegir el vector compuesto por el punto *C* (*Current*/Actual) y un punto seleccionado de manera aleatoria *R* (*Random*/Aleatorio), de esta manera introducimos el factor aleatorio que dirigirá U_{CN} .

Una vez seleccionado *R* de manera aleatoria (eligiendo coordenadas que se encuentren dentro del espacio de soluciones), componemos el vector *CR* y realizamos el producto cruzado *OC x CR* y conseguimos un vector normal (perpendicular) a *OC* (y a *CR*) pero no unitario, tan solo debemos convertir dicho vector en un vector unitario y obtendremos U_{CN} .

$$X_{i+1} = X_i + StepLength \cdot U_{CN} \tag{3}$$

La nueva posición del salto viene dado por la Ecuación (3). En la primera iteración aún no existe ningún vector *OC*, por lo que basta con elegir un punto origen al azar y otro punto destino también al azar de modo que podamos construir un primer vector unitario U_{CN} que nos dé la dirección y sentido del primer salto, entonces el punto origen (*C*) se convertirá en *O* y el punto destino (*N*) se convertirá en *C*, y a partir de ahí ya se pueden calcular el resto de saltos tal y como se ha indicado.

Después de calcular un nuevo salto se obtiene un punto nuevo *N* (*New*/Nuevo) que se corresponde con una solución a probar, entonces se calcula la calidad de la solución con la función de Fitness especificada. Si la calidad del nuevo punto no mejora a la de la mejor solución encontrada hasta el momento se sigue con el vuelo y se calcula un nuevo salto y un nuevo punto *N*, para ello el actual punto *C* pasa a ser el punto *O* en la nueva iteración, de la misma manera el punto *N* pasa a ser el nuevo punto *C* y el nuevo punto *N* a probar se calcula siguiendo el algoritmo, realizando un salto desde el punto *C* tal y como se ha definido.

Para limitar el número de saltos de un vuelo se añade una restricción adicional, de modo que si el vuelo invertido realiza β saltos sin encontrar una solución mejor que la existente, entonces se reinicia el vuelo de modo que se realiza un vuelo invertido nuevo. El parámetro β indica, por tanto, la longitud máxima de un vuelo, el sentido de este parámetro es evitar que se llegue a saltos en el vuelo de tal magnitud que se salgan del espacio de búsqueda y queden reducidos a los límites del mismo. Por lo que hay que parametrizar este parámetro junto con α y δ de modo que en β saltos con los valores seleccionados se pueda llegar de cualquier punto a cualquier otro punto en el espacio de soluciones, de esta manera aseguramos que la exploración del espacio de soluciones sea óptima.

Tabla 1. Pseudocódigo del algoritmo QSearch

```
Fitness Function f(x), x = (x_1, x_2, ..., x_d)^T ;
Init x^0 randomly;
Init F_g = min(f(x)) (best global solution);
t = 0; i = 1;
While (t < MaxIterations) or (Stop Criterion);
   Calculate fitness of x^t, F_{x_t} ;
   if ( F_{x_i} > F_g )
      Update F_g and Reinit the flight from x^t
   else
      if ( i > β )
         Reinit the Flight from F_g
      end;
      Update StepLength and calculate new U_CN
      t++; i++;
      x^t = x_{t-1} + StepLength · U_CN;
   end;
end while;
Process Result;
```

Como regla general, para tener una buena relación de número de pruebas a realizar y calidad de las soluciones, debería bastar con seleccionar β entre 6 y 8, de modo que en 6 a 8 saltos se abarque la máxima separación en el espacio de búsqueda. Los parámetros α y δ deben parametrizarse una vez definido β, α define la longitud mínima del salto, o la resolución que es capaz de alcanzar el algoritmo y δ la aceleración del vuelo.

4 Experimentos

4.1 Pruebas con función de Test

Para verificar el correcto comportamiento del algoritmo, antes de probarlo con el modelo, realizamos varias pruebas usando como funciones objetivo funciones conocidas, de modo que nos permitiese comparar los resultados obtenidos con los conseguidos con otros algoritmos.

Una de las funciones de prueba usada fue la función de Michaelwicz:

$$f(x,y) = -\sin(x)\sin^{2m}\left(\frac{x^2}{\pi}\right) - \sin(y)\sin^{2m}\left(\frac{2y^2}{\pi}\right) \quad (4)$$

En la Fig. 2 podemos ver el comportamiento de dicha función cuando $(x,y) \in [0,4]x[0,4]$. Podemos observar que la función tiene varios mínimos locales y un mínimo global situado en el punto (2.2031,1.5704) en la que la función toma el valor -1.8013.

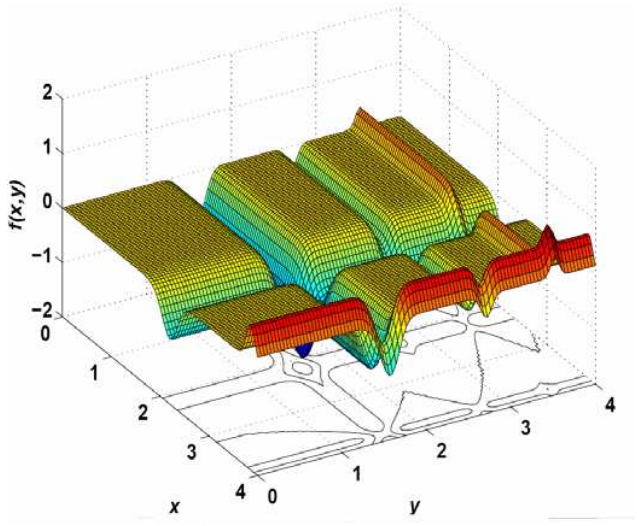


Fig. 2. Michaelwicz Landscape

En las siguientes gráficas vemos la trayectoria seguida por Qsearch con diferentes configuraciones. En la Fig. 3 vemos una configuración 3-6-3 ($\alpha - \beta - \delta$) y 100 iteraciones (ejecuciones de la función de calidad) con un resultado de -1.7836 en el punto (2.2354, 1.5736), bastante aceptable para solo 100 ejecuciones, con esa configuración no mejora la precisión significativamente con el número de iteraciones al tener el parámetro α un valor alto.

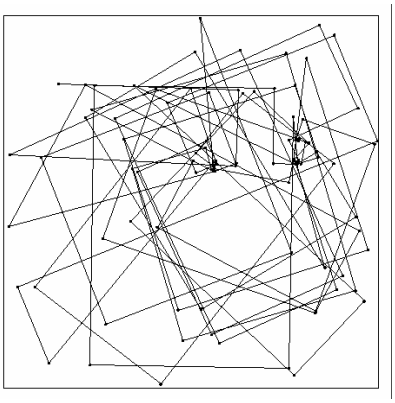


Fig. 3. Qsearch de Michaelwitz con 3-6-3 y 100 iteraciones (2.2354,1.5736) = -1.7836

En la Fig. 4 vemos una configuración 2-7-3 y 100 iteraciones obteniendo una mejora en la resolución de la solución encontrada.

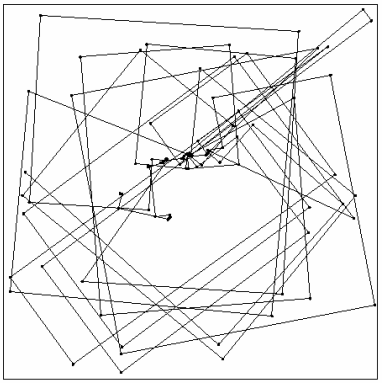


Fig. 4. Qsearch de Michaelwitz con 2-7-3 y 100 iteraciones (2.2079,1.5649) = -1.7995

En la Fig.5 y la Fig. 6 vemos cómo el comportamiento mejora sustancialmente al bajar α y aumentar las iteraciones, con 240 iteraciones el resultado es muy bueno obteniendo un mínimo de -1.80128 en el punto (2.2017, 1.5706).

Aunque la mejor relación de ejecuciones/resolución lo tenemos en la Fig. 6 con una configuración 0.5-6-3 y 200 iteraciones, obteniendo un resultado de -1.80107 en el punto (2.2022, 1.5731).

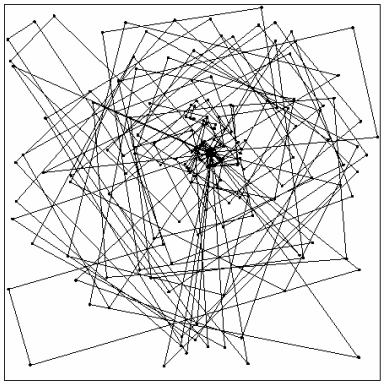
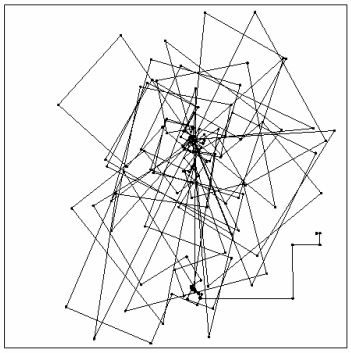


Fig. 5. Qsearch de Michaelwitz con 0.5-8-3 y 240 iteraciones
(2.2017, 1.5706) = - 1.80128



..

Fig. 6. Qsearch de Michaelwitz con 0.5-6-3 y 200 iteraciones
(2.2022, 1.5731) = - 1.80107

Tras los alentadores resultados de Qsearch aplicado a la función de prueba usada, pasamos a probar el algoritmo con el modelo QUAV (*Quadruple Unmaned Aerial Vehicle*). Para ello se usó un banco de pruebas diseñado para probar el controlador PID de cabeceo y alabeo (*pitch y roll*) del modelo. El banco está diseñado para minimizar el rozamiento y simular el comportamiento del modelo en vuelo, de modo que el propio banco no introduzca algún factor en el controlador.

Para el análisis de los resultados se introdujo en la interfaz de mando y control del proyecto una sección de calibración donde se muestran visualmente el desarrollo del algoritmo Qsearch, así como las gráficas y resultados de la función de calidad, que aunque no es objeto principal de este trabajo se resume a continuación.

La interfaz de mando y control (Fig. 7) cuenta con un grabador y reproductor de la prueba con el que se puede analizar detenidamente los resultados. El grabador almacena no solo los datos relacionados con el algoritmo, tales como parametrización, límites, valores iniciales, y tests realizados, sino también todos los valores de los sensores del modelo, para poder reproducir el comportamiento completo del modelo durante un test concreto.

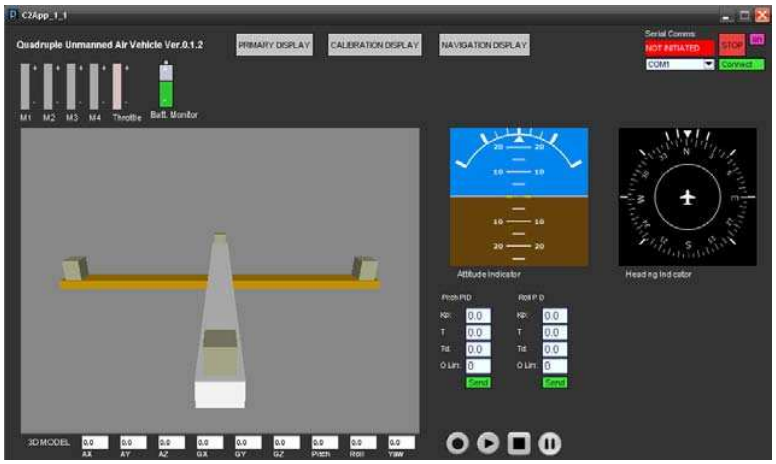


Fig. 7. Interfaz de Mando y control del proyecto QUAV

4.2 Función de calidad (*Fitness*)

La función de calidad (*fitness*) utilizada ha sido también implementada desde cero teniendo en cuenta varios factores que pasaremos a resumir.

Dicha función debe medir la estabilidad del modelo, entendiéndose por estabilidad la unión de varios aspectos. La entrada del PID será el ángulo calculado por el procesador embebido en el QUAV respecto a la horizontal, este ángulo es calculado a través de una matriz de cosenos directrices que toman de entrada los datos de aceleraciones y velocidades angulares obtenidos a través de los tres acelerómetros y giróscopos instalados en el modelo.

Se establecen unos límites de estabilidad, caracterizados por la constante MARE (Máximo Ángulo en Régimen de Estabilización), que situaremos en 5 grados, y que definen el error permitido dentro del régimen de estabilización. Mientras el ángulo (*AngAct*) esté en dichos márgenes consideraremos que el modelo se encuentra estabilizado (Ecuación (5)).

$$AngObj - MARE < AngAct < AngObj + MARE \tag{5}$$

Definiremos cuatro métricas que unidas conformarán la métrica de calidad. Dichas métricas son:

Métrica 1: Porcentaje de tiempo durante el cual el modelo ha estado dentro de los márgenes de estabilidad. Se debe maximizar.

Métrica 2: Se calcula para cada periodo de desestabilización la media de error en ángulo respecto al objetivo seleccionando el máximo de dichas medias. Hay que minimizar esta métrica.

Métrica 3: Máximo tiempo transcurrido desde que el modelo se ha desestabilizado hasta que ha vuelto a entrar en régimen de estabilización, mide el tiempo de reacción, se debe minimizar.

Métrica 4: Análoga a la métrica 2 pero minimizando el error en régimen de estabilización.

La función de calidad se define como la suma ponderada de las cuatro métricas, para ello se normalizan todas las métricas y se ponderan asignando costes a cada métrica, las métricas a maximizar se suman y las métricas a minimizar se restan como se puede ver en la Ecuación (6).

$$Fitness = P_1 \cdot Met1 - P_2 \cdot Met2 - P_3 \cdot Met3 - P_4 \cdot Met4 \tag{6}$$

4.3 Pruebas en banco

Para la realización de las pruebas se configuró una parametrización de Qsearch en la que se limitó el espacio de búsqueda de la siguiente manera:

$$(p,i,d) \in [\,200,600\,]x[\,1,200\,]x[\,1,50\,] \tag{7}$$

Aunque las dimensiones de cada parámetro están acotadas de modo que los rangos se limiten a valores comprensibles para un controlador PID y cada uno tiene una dimensión diferente, a la hora de realizar los vuelos se normalizan todos para que el espacio de soluciones tenga forma de cubo de 200x200x200. De manera que se simplifique la visualización de los datos.

Aunque la primera estimación para una calibración adecuada, habla de un número de iteraciones de entre 200 y 350, a la hora de escribir este capítulo tan solo se había podido realizar una calibración preliminar de 40 iteraciones. El tiempo medio de cada ejecución de la función de fitness es de 35 segundos con 5 segundos de reposo entre test y test.

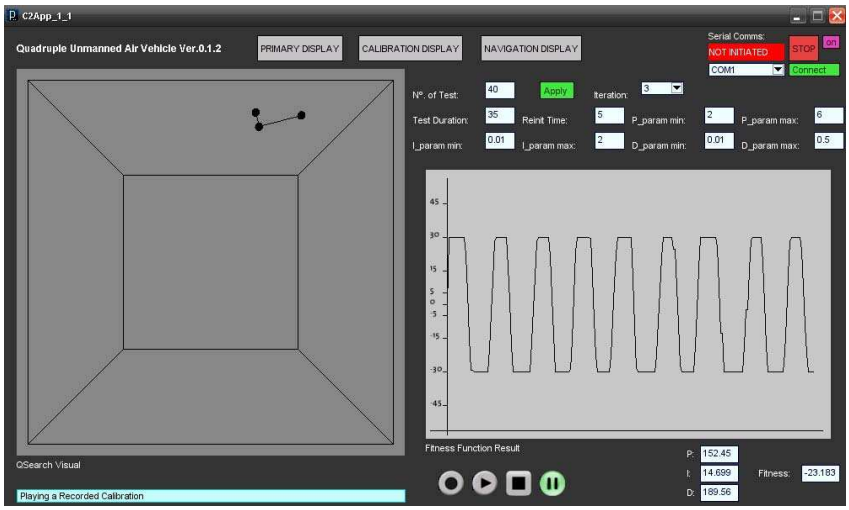


Fig. 8. QSearch aplicada a QUAV, tercera iteración de 40

En la Fig. 8 se puede ver la primera iteración y el resultado de la función de calidad. En la Fig. 9 se puede ver el resultado final del Qsearch y la gráfica correspondiente a la mejor solución encontrada.

El resultado de esta calibración ha sido de 27.490 (en una escala de -50 a 50), cuando la mejor calibración realizada a mano siguiendo los métodos de calibración manual encontrados no había sido mejor de 22.670. Por lo que podemos concluir que el algoritmo QSearch aplicado a la calibración del PID del QUAV es muy efectivo.

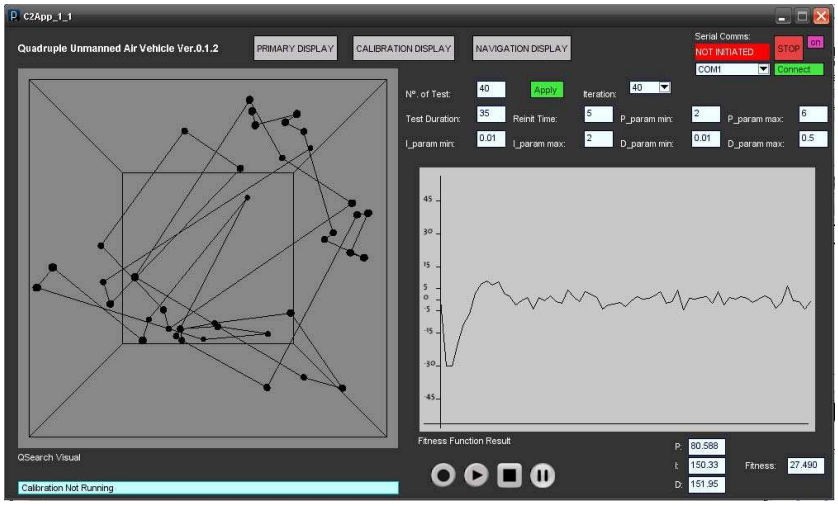


Fig. 9. Qsearch aplicada a QUAV, muestra el resultado tras 40 iteraciones

5 Conclusiones

La conclusión principal es que la aplicación de un algoritmo de búsqueda optimizada para la optimización de los parámetros de un controlador PID puede mejorar sensiblemente los resultados obtenidos con calibraciones manuales. Sobre todo puede ser de gran ayuda cuando no se dispone de un modelo matemático de comportamiento, ni de un simulador confiable del sistema.

En el caso particular del proyecto QUAV la calibración parcial llevada a cabo hasta la fecha ya ha logrado un resultado que ha mejorado la mejor calibración conseguida a mano. Si se tiene en cuenta que la calibración realizada ha sido limitada en 40 iteraciones, se puede suponer que al realizar una calibración de 250 iteraciones, mejorando la precisión del algorit-

mo bajando el valor del parámetro α, el resultado obtenido mejorará sustancialmente el actual, facilitando el desarrollo del resto de controladores que estarán por encima del sistema de estabilización, como los diferentes sistemas de control de altura, posición y navegación.

También se debe hacer constar que los resultados obtenidos hasta ahora son suficientemente buenos como para poder pasar a realizar pruebas más complejas como vuelos de test fuera del banco de calibración, y poder avanzar en módulos paralelos del proyecto.

Como algoritmo de búsqueda optimizado, los resultados obtenidos han sido muy positivos. Uno de los trabajos futuros consiste en un estudio comparativo más detallado sobre los resultados de Qsearch comparado con resultados al mismo nivel de otros algoritmos de búsqueda optimizada, como los expuestos en este capítulo.

Otro de los trabajos futuros es el estudio y análisis de un filtro de partículas más complejo que permita la ejecución de tantos QSearch como partículas en cada iteración, complementando los resultados individuales con los globales, permitiendo la implementación de un nuevo algoritmo de partículas basado en QSearch y que pueda ser comparado con el resto de algoritmos similares existentes.

Referencias

Arora, J. 1989. Introduction to Optimum Design, McGraw-Hill.

Blum, C. and Roli, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptural comparision. ACM Comput. Surv., 35: 268-308.

Eiben, A. E. *et al*. 1994. Genetic algorithms with multi-parent recombination. PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78-87.

Goldberg, David E. 1989. Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley. pp. 41.

Higuera R., O. 2010. QUAV: A Quadruple Unmanned Aerial Vehicle. Proyecto Fin de Carrera, Escuela Técnica Superior de Ingeniería Informá-

tica, Universidad Rey Juan Carlos, http://www.robotica-urjc.es/index.php/QUAV.

Kennedy, J., Eberhart, R. 1995. Particle Swarm Optimization. In Proceedings of IEEE International Conference on Neural Networks, IV. pp. 1942-1948.

Pavlyukevich, I. 2007. Lévy flights, non-local search and simulated annealing, J. Computational Physics, 226: 1830-1844.

Rardin, R. L. 1997. Optimization in Operations Research, Prentice Hall.

Shi, Y. and Eberhart, R.C. 1998. A modified particle swarm optimizer. In Proceedings of IEEE International Conference on Evolutionary Computation, pp. 69-73.

Yang, X.-S. and Deb, S. 2009. Cuckoo search via Lèvy flights. In Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), India. IEEE Publications, USA, pp. 210-214 .

Yang, X.-S., and Deb, S. 2010. Engineering Optimisation by Cuckoo Search, Int. J. Mathematical Modelling and Numerical Optimisation, 1(4): 330-343.