Optimización de los parámetros de movimiento de un robot cuadrúpedo mediante computación evolutiva y aprendizaje automático

J. I. Alonso
Grupo SIMD/I^{3}A
DSI - UCLM
juanignacio.alonso@uclm.es

J. A. Gámez
Grupo SIMD/I^{3}A
DSI - UCLM
jgamez@dsi.uclm.es

I. García-Varea
Grupo SIMD/I^{3}A
DSI - UCLM
ivarea@dsi.uclm.es

J. Martínez
Grupo SIMD/I^{3}A
DSI - UCLM
jesus_martinez@dsi.uclm.es

Resumen

En un robot móvil con patas, la velocidad viene determinada por una serie de parámetros que determinan la forma de caminar y que es necesario adaptar a las condiciones del entorno. Ajustar estos parámetros no es un problema trivial y requiere de mucho tiempo incluso para un experto. Por esto, en los últimos años se han desarrollado técnicas para el aprendizaje automático de estos parámetros. A lo largo del artículo se presentará una propuesta para resolver el problema del aprendizaje automático de estos parámetros en un robot AIBO ERS-7. Como parte de esta propuesta se incluye el uso de un algoritmo genético junto a técnicas de minería de datos que ayudan a reducir considerablemente el tiempo necesario para el aprendizaje.

1. Introducción

Para un robot móvil, la velocidad a la que puede desplazarse de un lugar a otro es una de las características más importantes. Esta velocidad está determinada en gran medida por la superficie sobre la que se desplaza. Esto hace que una determinada forma de andar sea buena para una superficie determinada, pero no lo sea en otra.

En el problema que se presenta en este artículo, se utilizará un robot AIBO ERS-7. Este robot se encuentra dentro de los robots móviles de 4 patas y su movimiento viene determinado por una serie de parámetros asignados a los grados de libertad de las articulaciones. En los últimos años se han desarrollado técnicas para el aprendizaje automático de estos parámetros[5] debido principalmente a las siguientes razones:

1. El tiempo y el conocimiento experto (cinemática) necesario para realizar una configuración manual.
2. Las configuraciones de parámetros aprendidos de forma automática han conseguido mejoras sustanciales en la velocidad de movimiento de los robots respecto a las configuraciones manuales.

La desarrollos sobre el robot AIBO generalmente están relacionados con la competición Robocup [1]. Esta competición se celebra anualmente y tiene una categoría en que se utiliza este robot como plataforma común para todos los equipos. La parte principal de la competición es un campeonato de fútbol en el que se enfrentan equipos de 4 robots. Los trabajos realizados sobre optimización de movimiento se centran en conseguir una buena velocidad durante el partido [4, 6], por lo que es imprescindible que el tiempo de configuración sea bajo para poder realizar el aprendizaje sobre el campo de juego antes de la competición.

El objetivo de este trabajo es desarrollar un método automático para la optimización de los parámetros del movimiento que se pueda ejecutar en un tiempo razonablemente bajo. Para ello, en este trabajo usamos un al-

[1] www.robocup.com

goritmo genético en combinación con técnicas de minería de datos que permiten establecer un modelo de movimiento y acelerar la evaluación de las distintas configuraciones a testear. Esta técnica nos permite no sólo obtener un movimiento que mejora la velocidad inicial, sino que además es lo suficientemente rápida como para realizar la adaptación a unas nuevas condiciones del entorno en un tiempo razonable de manera automática.

El resto del trabajo se estructura en las siguientes secciones. En la sección 2 se explica el módulo de movimiento usado y los parámetros que contiene. En la sección 3 se explican los aspectos generales del algoritmo genético. En la sección 4 se profundiza en la función de fitness y se explican las mejoras realizadas. En la sección 5 se explica las necesidades de la reevaluación de los individuos para minimizar el efecto del ruido. En la sección 6 se explica el uso de algoritmos de aprendizaje automático para reducir el tiempo de ejecución. En la sección 7 se muestran y analizan las pruebas y los resultados obtenidos. Por último, en la sección 8 se muestran las conclusiones.

2. Módulo de movimiento

Para el movimiento del robot usamos el módulo de movimiento desarrollado por el equipo rUNSWift [2] para la competición RoboCup 2005. Este módulo es bastante flexible, permitiendo elegir entre varios tipos de movimientos, cada uno de ellos parametrizable. Nueve de los parámetros son comunes a todos los tipos de movimientos (ver Cuadro 1).

En las pruebas hemos usado dos de los tipos de movimientos definidos en el módulo. El primero de ellos, conocido como *NormalWalk*[3], es el movimiento más simple que posee la librería y consta únicamente de los parámetros comunes. Este movimiento es interesante porque fue el primero que se desarrolló y ha sido la base de desarrollo del resto. El otro movimiento usado, conocido como *SkellipticalWalk*[1], es el último desarrollado por el equipo y el que mejores prestaciones ha obtenido. Este movimiento consta de 22 parámetros propios, haciendo un total de 31 parámetros.

[2] http://www.cse.unsw.edu.au/ robocup/

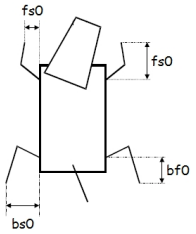

Figura 1: Posición inicial de las patas del AIBO

Los parámetros utilizados son todos de tipo numérico, siendo algunos de tipo real y otros de tipo entero. Para abordar el problema, los parámetros reales se han discretizado en valores enteros ya que experimentalmente se ha comprobado que esta pérdida de granularidad no tiene efecto sobre el fitness de los individuos.

3. Algoritmo genético

Una vez que un determinado esquema de movimiento está definido como la obtención de una configuración para los distintos parámetros que lo definen, es claro que la tarea puede plantearse como un problema de optimización combinatoria cuyo objetivo (en nuestro caso) es maximizar la velocidad del robot AIBO al caminar.

Para elegir un algoritmo de optimización combinatoria para el problema debe tenerse en cuenta que el proceso de evaluación es muy costoso, generalmente entre 20 y 30 segundos por configuración. Además la función de evaluación contiene ruido por lo que dos evaluaciones de la misma configuración generalmente no darán exáctamente el mismo valor.

En este artículo se ha usado un algoritmo genético generacional [8] haciendo uso de la librería LiO [7]. Las características generales de este algoritmo son:
a) La representación utilizada para los individuos consiste en un vector de enteros donde cada posición se corresponde con uno de los parámetros del movimiento,

Nombre	Descripción
PG	Tiempo para completar un paso en unidades de 0.008 s. P.ej. si PG=65 tardará 0.52 s en dar un paso
hF	Altura en mm de la parte delantera del cuerpo
hB	Altura en mm de la parte trasera del cuerpo
hdF	Altura en mm a la que levanta las patas delanteras durante un paso
hdB	Altura en mm a la que levanta las patas traseras durante un paso
ff0	Posición inicial de las patas delanteras (ver Figura 1)
fs0	Posición inicial de las patas delanteras (ver Figura 1)
bf0	Posición inicial de las patas traseras (ver Figura 1)
bs0	Posición inicial de las patas traseras (ver Figura 1)

Cuadro 1: Descripción de los parámetros comunes del movimiento

b) La población inicial se genera de forma aleatoria,
c) La selección de los padres genera $n/2$ parejas que incluyen a los n padres,
d) Se realiza cruce por un punto con una determinada probabilidad de cruce,
e) La mutación se realiza gen a gen y consiste en la generación de un valor aleatorio para ese gen,
f) La evaluación de los individuos se hace de forma paralela. En nuestro caso hemos usado 4 robots, aunque el algoritmo escala de forma natural a cualquier número de robots,
g) El reemplazo se hace de manera elitista seleccionando los k mejores individuos de la unión de los hijos y los padres y el resto se seleccionan de manera aleatoria entre aquellos individuos válidos (se consideran individuos no válidos aquellos que han superado el tiempo máximo de evaluación),
h) La condición de parada se ha fijado en el número de iteraciones.

El resto de características del algoritmo se muestran en las siguientes secciones.

4. Función de fitness

El escenario donde se van a realizar las pruebas consiste en una alfombra cuadrada de 4 metros de lado. Los robots recorren la alfombra de un lado a otro entre dos cajas (situadas en lados opuestos de la alfombra) del mismo color que sirven para orientarse. En total hay 4 pares de cajas situadas en la alfombra, de manera que se pueda realizar la evaluación simultánea de hasta 4 robots. La función de fitness consiste en medir el tiempo que tardan los robots en ir de un lado a otro de la alfombra dada una configuración de parámetros. Esta función se muestra en la Ec.(1), donde P es la configuración de parámetros (vector de enteros) y t es el tiempo en segundos.

$$f(P) = t \quad (1)$$

Durante la evaluación el robot corrige su trayectoria en caso de que se desvie de su objetivo (la caja que tiene enfrente). Esto, además de asegurar que el robot llega correctamente a su objetivo, es una forma de penalizar aquellos movimientos que no son completamente rectilíneos (recorren más distancia). De esta forma el algoritmo es robusto frente a movimientos que no son completamente rectilíneos.

Otro problema que surge es que los robots no se paran siempre a la misma distancia de las cajas y, como consecuencia, la distancia recorrida es distinta en cada evaluación. Esto se debe a que desde que se le da la orden al robot de pararse hasta que realmente se para pasa un cierto tiempo. Para reducir este problema realizamos la siguiente transformación, consistente en maximizar la velocidad en lugar de minimizar el tiempo: se estima la distancia real recorrida a partir de la medición del sensor de infrarrojos que tiene el robot y de su propio tamaño (27 cm). Esta distancia se mide en centimetros y se calcula del siguiente modo (ver Figura 2):

$$d_{recorrida} = 400 - d_{IR1} - d_{IR2} - 2 * 27 \quad (2)$$

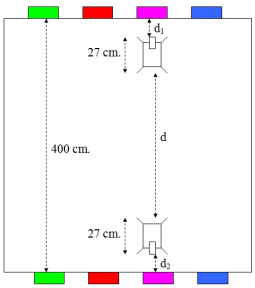

Figura 2: Estimación de la distancia real recorrida

Teniendo esto en cuenta, la nueva función de fitness mide la velocidad real en cm/s y se calcula como:

$$f(P) = \frac{d_{recorrida}}{t} \quad (3)$$

Se establece también un límite de tiempo (en nuestro caso 30 segundos) como tiempo máximo que puede tardar la evaluación de un individuo. Si se supera este tiempo se aborta la evaluación y al individuo se le asigna como fitness el valor -1.

5. Reevaluación de los individuos

El problema del ruido generalmente se ha tratado realizando varias mediciones, descartando alguna de ellas y haciendo la media de las restantes [4, 6]. El problema es que esto hace más lenta aún la ejecución del algoritmo. La solución que se ha adoptado es similar a la usada en [2] y consiste en realizar una sola medida para obtener el fitness de una solución y reevaluar la solución cada cierto número de iteraciones si todavía permanece en la población. Así se consigue evitar realizar un gran número de mediciones de soluciones que son poco prometedoras y por tanto descartadas de la población.

Concretamente, los individuos son reevaluados cada 2^{n-1} generaciones si el individuo todavía permanece en la población, siendo n el número de evaluaciones que ya se han hecho del individuo. Es decir, después de la primera evaluación, el individuo se reevaluará en la siguiente generación, luego 2 generaciones más tarde y así sucesivamente. Para obtener el fitness del individuo se hace la media de sus evaluaciones si lleva menos de 3 evaluaciones. En caso contrario, se descarta el valor más alejado de la media y se recalcula la media del resto, eliminando así posibles valores con mucho ruido.

6. Uso de una función secundaria de fitness

El algoritmo descrito hasta ahora consiste en una versión completa para resolver nuestro problema en la que además se trata la reducción del ruido de la función de evaluación. Sin embargo, este algoritmo es muy costoso, tanto en tiempo como en intervención humana (cambio de baterías, supervisión de las trayectorias, etc.). Para paliar aunque solo sea en parte estos problemas, proponemos introducir un modelo que permite estimar el valor de la función de fitness y, en ocasiones, usar el valor estimado en lugar del real. El modelo usado puede ser cualquier algoritmo de predición numérica y se obtiene mediante aprendizaje automático a partir de un conjunto de entrenamiento con resultados de ejecuciones anteriores. Los modelos usados han sido el algoritmo de clasificación por vecinos más cercanos (KNN) [9], árbol de modelos mediante el algoritmo de Weka M5P[11] y regresión lineal [10]. Para la implementación se ha utilizado la librería WEKA[3][11].

Definimos 3 rangos para la función de fitness (buena, regular y mala) a partir del valor de 2 valores límite A_1 y A_2 que se fijan previamente a la ejecución del algoritmo del siguiente modo:

$$\begin{aligned} Mala: &\quad fitness \leq A_1 \\ Regular: &\quad A_1 > fitness \leq A_2 \\ Buena: &\quad fitness > A_2 \end{aligned}$$

En las pruebas se ha utilizado como valor de A_2 un valor similar al obtenido por el movimiento sin optimizar (con los valores de los parámetros por defecto de cada tipo de

[3] http://www.cs.waikato.ac.nz/ml/weka/

movimiento) de manera que los individuos seleccionados como buenos sean superiores a este valor. El valor de A_1 se ha escogido experimentalmente observando el fitness de los individuos que a simple vista parecían poco prometedores.

Antes de evaluar un individuo se estima su fitness ($\hat{f}_e$) usando el modelo y se clasifica en uno de los 3 rangos anteriores. Se definen también 3 umbrales α_1, α_2 y α_3 ($0 \leq \alpha_i \leq 1$). Un individuo será evaluado si se cumple alguna de las siguientes condiciones ($U[0,1]$ es un valor aleatorio entre 0 y 1):

$$\hat{f}_e \in Buena \wedge U[0,1] \leq \alpha_1$$
$$\hat{f}_e \in Regular \wedge U[0,1] \leq \alpha_2$$
$$\hat{f}_e \in Mala \wedge U[0,1] \leq \alpha_3$$

En otro caso el fitness del individuo será el estimado por el modelo. El valor de los umbrales determina el porcentaje de individuos que no se van a evaluar y deben tenerse en cuenta algunas consideraciones:

- El valor de α_3 debería ser próximo a 0, de manera que no se pierda tiempo en evaluar individuos que seguramente tendrán un fitness bajo.
- El valor de α_2 debería ser un valor intermedio, en torno a 0.5, ya que los individuos estimados como regulares en ocasiones contendrán "material genético"útil.
- El valor de α_1 debería ser 1, ya que determina el número de individuos estimados como buenos que queremos realmente evaluar.

Cuando se evalua un individuo se dispone de su valor estimado y se puede comprobar si el modelo lo había clasificado en el rango correcto. Por tanto, se puede calcular la tasa de aciertos del modelo como el número de individuos correctamente clasificados entre el número total de individuos evaluados. Sólo se usará el modelo en aquellos casos en que la tasa de aciertos supera el umbral t_{min} ($0 \leq t_{min} \leq 1$). El valor de t_{min} debe ser un valor suficientemente alto (en las pruebas se ha utilizado 0.75) para que el modelo sólo se use si está funcionando realmente bien. Si durante la ejecución el modelo no está obteniendo una buena tasa de aciertos el algoritmo funcionará de manera identica a como lo haría sin usar esta mejora. El pseudo-código de la función evaluar es el que se muestra como Algoritmo 1.

```
Funcion Evaluar(Poblacion)
Si (tasaAciertos<Umbral)
     Para cada Individuo en Poblacion
          Evaluar(Individuo)
Si no
     Para cada Individuo en Poblacion
          Estimar(Individuo)
          Si Rango(Individuo) = Bueno
          Y Random(0,1) ≤ α1
               Evaluar(Individuo)
          Si Rango(Individuo) = Regular
          Y Random(0,1) ≤ α2
               Evaluar(Individuo)
          Si Rango(Individuo) = Malo
          Y Random(0,1) ≤ α3
               Evaluar(Individuo)
```

Algoritmo 1: Función evaluar

El modelo de movimiento planteado puede ser mejorado si se tienen en cuenta algunas consideraciones:

- La construcción del conjunto de entrenamiento se realiza mediante varias ejecuciones del algoritmo sobre la misma superficie (o una similar) por lo que generalmente este conjunto no contendrá muchas instancias.
- El aprendizaje del modelo a partir de ejecuciones anteriores puede resultar inapropiado si el espacio de soluciones explorado en la ejecución actual es muy distinto al explorado en otras ejecuciones.
- Si el entrenamiento se realiza con datos parecidos a los de la ejecución actual el modelo puede realizar una mejor estimación.
- Durante la ejecución del algoritmo, cada evaluación que se realiza puede servir co-

mo nuevo caso para la construcción del modelo.

Teniendo esto en cuenta es claro que se podrían utilizar las evaluaciones que se estan realizando para estimar las siguientes evaluaciones en una misma ejecución. En nuestro caso, se ha dejado como opción hacer uso del modelo tal y como se ha explicado hasta ahora o hacer uso de un *modelo incremental*. Los *modelos incrementales* consisten en reestimar el modelo en cada generación añadiéndole las evaluaciones de la generación anterior. Para realizar esto, es claro que serán preferibles aquellos modelos que puedan ser actualizados incrementalmente o cuyo tiempo de aprendizaje sea reducido.

En la Figura 3 se puede ver un esquema de la utilización de la función secundaria de fitness. Para calcular el fitness de una configuración de parámetros primero se estima su valor con la función $\hat{f}_e$. Sólo algunos de estas configuraciones serán evaluadas (siguiendo la linea punteada) obteniendo el fitness del individuo X. Además cada una de estas evaluaciones se añaden al conjunto de entrenamiento a partir del cual se aprende el modelo.

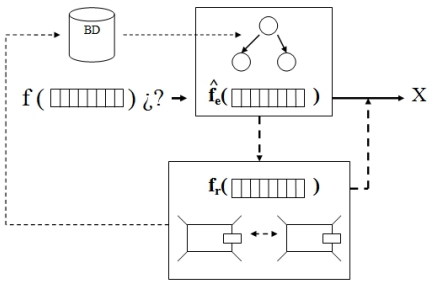

Figura 3: Esquema de utilización de la función secundaria de fitness

Teniendo en cuenta todas las características mostradas anteriormente, el pseudo-código del algoritmo genético diseñado es el que se muestra como Algoritmo 2.

```
Poblacion = GenerarIndividuos()
Evaluar(Poblacion)
Modelo = GenerarModelo()
Mientras (No Condición de parada)
    Padres = SeleccionarPadres(Poblacion)
    Hijos = Cruzar(Padres,pCruce)
    Hijos = Mutar(Hijos,pMutacion)
    Evaluar(Hijos)
    Poblacion = Reemplazar(Padres,Hijos)
    Reevaluar(Poblacion)
    Modelo = ReestimarModelo()
```

Algoritmo 2: Algoritmo genético

7. Pruebas realizadas y resultados obtenidos

Con el método de optimización descrito anteriormente se han realizado un conjunto de pruebas variando el modelo de aprendizaje usado en cada una de ellas. Los valores usados para el algoritmo genético en todas las pruebas se resumen en el Cuadro 2. Estos valores se han obtenido de forma experimental realizando varias ejecuciones del algoritmo que además sirvieron para formar los conjuntos de entrenamiento (2100 instancias para *NormalWalk* y 2300 para *SkellipticalWalk*).

Parámetro	Valor
Tamaño población	20
Probabilidad de cruce	0,6
Probabilidad de mutación	0,05
Nivel de elitismo (k)	2
Número de iteraciones	15
A_1/A_2 (*NormalWalk*)	15 / 25
A_1/A_2 (*SkellipticalWalk*)	18 / 28
$\alpha_1/\alpha_2/\alpha_3$	1 / 0.4 / 0.2
t_{min}	0,75

Cuadro 2: Valores comunes del algoritmo genético

Los resultados obtenidos por el algoritmo sin utilizar el modelo han sido de 29,1 cm/s (fitness del mejor individuo) para el movimiento *NormalWalk* y de 34,3 cm/s para el *SkellipticalWalk*. Las evaluaciones realizadas han sido de 412 y 497 respectivamente. Cada evaluación sobre los rotots, teniendo en cuenta que se realizan 4 de forma paralela, tarda en media unos 8 segundos en el caso del movimiento

NormalWalk y 7 segundos en el caso de *SkellipticalWalk*. El tiempo de ejecución es proporcional al número de evaluaciones realizadas y, en ambos caso, ha sido de aproximadamente 1 hora.

Para cada uno de los 2 tipos de movimientos se han realizado 6 pruebas más, utilizando los distintos modelos tanto en su versión incremental como no incremental. Además de las pruebas realizadas, a partir de los resultados de una ejecución se puede calcular la tasa de aciertos que hubiese tenido cualquier otro modelo en esa ejecución. En el Cuadro 3 los resultados en negrita son las tasas de acierto reales obtenidas por el algoritmo y el resto de datos que aparecen (a excepción de las filas fitness, evaluaciones y estimaciones) son las tasas de acieto calculadas con este procedimiento. Por ejemplo, en la prueba 2 que se utilizó M5P incremental la tasa de acierto fue de 0,6471 para *NormalWalk* y 0,8178 para *SkellipticalWalk*. Los demás valores que aparecen son las tasas de acierto calculadas para el resto de modelos.

El movimiento *NormalWalk*, usando los parámetros por defecto, obtiene, sobre la alfombra donde se realizaron las pruebas, una velocidad media de 24,2 cm/s. Con el método implementado se han alcanzado velocidades entre 27,8 cm/s y 30,9 cm/s lo que supone mejoras entre el 15 % y el 27 % respectivamente. En el caso del movimiento *SkellipticalWalk*, usando los parámetros por defecto se obtiene una velocidad media de 26,4 cm/s. En las pruebas se han obtenido velocidades entre 32,1 cm/s y 35,1 cm/s lo que supone mejoras entre el 22 % y el 33 % respectivamente. Estos resultados se resumen en el Cuadro 3 y en las Figuras 4 y 5.

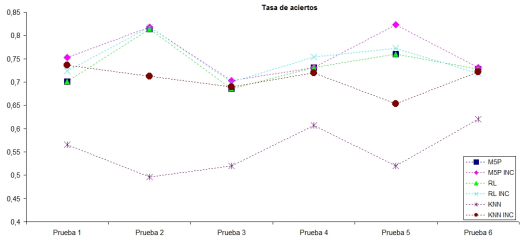

Figura 4: Tasas de acierto con *NormalWalk*

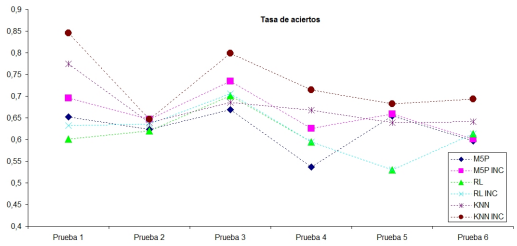

Figura 5: Tasas de acierto con *SkellipticalWalk*

El uso del modelo de movimiento en muchos casos no ha tenido ningún efecto sobre los resultados finales debido a que la tasa de aciertos mínima ha sido muy alta y el modelo no se ha llegado a usar. Sin embargo, en los casos en que el modelo ha funcionado bien se ha reducido considerablemente el número de evaluaciones manteniendo la misma efectividad en cuando al mejor individuo encontrado por el algoritmo. Por ejemplo, en el movimiento *SkelleptikalWalk* usando el algoritmo M5P se han estimado 147 de las 501 evaluaciones del algoritmo, lo que supone una reducción de casi el 30 % de las evaluaciones y, por tanto, del tiempo de ejecución.

Por otro lado, los resultados obtenidos por los modelos varían mucho de un tipo de movimiento a otro. En el caso del movimiento *SkelleptikalWalk* los mejores resultados han sido los obtenidos por los métodos M5P y regresión lineal. Sin embargo, en el movimiento *NormalWalk* los mejores resultados son los obtenidos por el método KNN.

Respecto al uso del modelo incremental los resultados han sido muy satisfactorios, superando los resultados de las versiones no incrementales en prácticamente todas las pruebas.

8. Conclusiones

Se ha presentado un método para el aprendizaje automático de los parámetros del movimien-

NormalWalk / *SkellipticalWalk*	Prueba 1 (M5P)	Prueba 2 (M5P INC)	Prueba 3 (RL)	Prueba 4 (RL INC)	Prueba 5 (KNN)	Prueba 6 (KNN INC)
M5P	**0,6522/ 0,701**	0,6235/ 0,8136	0,6694/ 0,6857	0,5367/ 0,7313	0,6546/ 0,7603	0,5968/ 0,7278
M5P INC	0,6957/ 0,7524	**0,6471/ 0,8178**	0,7339/ 0,7033	0,6255/ 0,7313	0,6586/ 0,8233	0,6008/ 0,731
RL	0,6008/ 0,701	0,6196/ 0,8136	**0,7016/ 0,6857**	0,5946/ 0,7313	0,5301/ 0,7603	0,6129/ 0,7278
RL INC	0,6324/ 0,7235	0,6353/ 0,8178	0,7056/ 0,7	**0,5946/ 0,7537**	0,5301/ 0,7729	0,6129/ 0,7184
KNN	0,7747/ 0,5659	0,6392/ 0,4958	0,6855/ 0,52	0,668/ 0,607	**0,6386/ 0,5205**	0,6411/ 0,6203
KNN INC	0,8458/ 0,7363	0,6471/ 0,7119	0,7984/ 0,69	0,7143/ 0,7201	0,6827/ 0,653	**0,6935/ 0,7215**
Fitness	28,2/34,6	27,8/34,3	30,9/32,5	28,1/32,1	28,6/35,1	29,6/33,5
Evaluaciones	426/501	405/354	410/465	419/416	401/505	412/504
Estimaciones	0/0	6/147	12/67	0/94	0/0	0/0

Cuadro 3: Resultados de las pruebas

to en un robot AIBO-ERS7. Además, se han presentado soluciones para tratar el ruido y mejorar el tiempo de ejecución. La propuesta ha demostrado ser eficiente en cuanto a resultados y tiempo, obteniendo mejoras de hasta el 30 % en ambos factores.

La propuesta desarrollada es extensible a otras metaheurísticas, a otros robots y al aprendizaje de otros comportamientos. Además, el uso de técnicas de minería de datos como se ha visto en este trabajo puede ser utilizado en multitud de problemas en los que el aprendizaje y uso de un modelo sea más eficiente que la evaluación de la función objetivo.

Referencias

[1] W. Chen. Odometry calibration and gait optimisation. Technical report, The UNSW, CSE, 2005.

[2] S. Chernova and M. Veloso. An evolutionary approach to gait learning for four-legged robots, 2004.

[3] B. Hengst, D. Ibbotson, S. B. Pham, and C. Sammut. The UNSW united 2000 sony legged robot software system. Technical report, The UNSW, CSE, 2000.

[4] M. Kim and W. Uther. Automatic gait optimisation for quadruped robots, 2003.

[5] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion, 2004.

[6] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion, 2004.

[7] J. L. Mateo and L. de la Ossa. LiO: an easy and flexible library of metaheuristics. Technical report, DSI, EPSA, UCLM, 2006.

[8] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[9] B. Sierra. *Aprendizaje Automático: Conceptos básicos y avanzados*. Prentice Hall, 2006.

[10] C. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond.

[11] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.