

APRENDIZAJE DE COMPORTAMIENTOS COMPLEJOS EN ROBOTS CUADRUPEDOS

Juan Cristóbal Zagal, Javier Ruiz-del-Solar

Dept. de Ing. Eléctrica, Casilla 412-3, Universidad de Chile

Email: jzagal@ing.uchile.cl, jruizd@ing.uchile.cl

RESUMEN

En este artículo se presenta una metodología para la generación automática de comportamientos robóticos complejos en robots cuadrúpedos. En general el aprendizaje de un comportamiento corresponde a la búsqueda de soluciones en el espacio de parámetros que definen a un controlador robótico. El tamaño de este espacio está directamente relacionado con la complejidad del comportamiento a desarrollar. Probar una solución perteneciente a este espacio equivale a evaluar el comportamiento del robot cuando éste interactúa con su medio usando la función de control asociada. Para implementar esta tarea, en este trabajo se utiliza aprendizaje en ambientes simulados y aprendizaje en ambientes reales, ambos combinados de acuerdo al paradigma Back to Reality (BTR), recientemente propuesto por los autores. La principal ventaja de utilizar simulaciones computacionales es la velocidad y paralelismo en la búsqueda de soluciones. Dada la diferencia insoslayable entre la realidad y la simulación, el enfoque propuesto pretende encontrar un punto de operación del simulador tal que se minimicen las diferencias entre los comportamientos ejecutados en la realidad y en la simulación, desde el punto de vista de la evaluación de estos.

1. Introducción

1.1. Robótica Evolutiva

La robótica evolutiva [13] corresponde a una metodología emergente cuyo objetivo es el diseño y la generación de robots en forma automática. En ella, los robots se consideran como agentes autónomos artificiales capaces de desarrollar sus habilidades o comportamientos, al interactuar estrechamente con su entorno. Dicho de otro modo, se asume que los robots pueden aprender a partir de su propia experiencia. Las principales herramientas de modelación y aprendizaje empleadas en robótica evolutiva, son la computación evolutiva, las redes neuronales y, en general todos los métodos de aprendizaje de máquinas. La robótica evolutiva ha demostrado ser exitosa en la generación de controladores robóticos reactivos simples [10,8] y en la adaptación de sistemas perceptuales [3,12,24]. No obstante existen pocos trabajos en la literatura

sobre la generación de comportamientos complejos, algunos ejemplos son [14,2].

Usualmente durante la ejecución de un comportamiento robótico se utiliza una función de *fitness* para medir el grado en que cierta tarea se satisface, o en otras palabras el grado de adaptación del robot a su medio. Uno de los principales factores a considerar en la selección de la función de *fitness* a utilizar es la cantidad y calidad de información que el robot es capaz de extraer de su entorno, producto de su interacción con éste (experiencia). Sin embargo, el proceso de adquirir información a través de interacciones con el medio requiere de tiempo e implica necesariamente el desgaste de la plataforma robótica empleada. Una alternativa consiste en simular la interacción entre el robot y el ambiente, explorando rápidamente, y en paralelo, el espacio de parámetros que definen a un controlador. De esta manera es posible dejar la variable temporal como una función de la capacidad computacional disponible. La principal limitante impuesta por esta filosofía de trabajo está dada por la diferencia

fundamental e insoslayable existente entre la realidad y la simulación (*reality-gap*).

La literatura presenta diversos experimentos en los cuales se emplean simuladores para generar comportamientos robóticos simples [10,13]. Sin embargo, no existen muchos ejemplos de la generación de comportamientos complejos en un entorno simulado, que puedan luego ser transferidos exitosamente a un entorno real. La simulación puede desarrollarse en distintos niveles, como por ejemplo, la simulación de procesos de alto nivel, donde la información sensor motriz es simplificada. Sin embargo, una representación más completa se logra al generar simulaciones que incluyan interacciones de bajo nivel, teniendo en cuenta la física de sensores y actuadores, i.e. la interacción de bajo nivel entre el robot y el entorno. Esto requiere usualmente de modelos matemáticos complejos, cuya solución posee un elevado costo computacional. Consideramos que un requisito fundamental para la generación de comportamientos complejos de bajo nivel en un ambiente simulado, es que el simulador posea una representación completa de las interacciones entre el robot y su entorno. Los procesos de alto nivel se obtienen luego como resultado de los procesos de bajo nivel. No obstante definir hasta que punto el simulador es fiel respecto de la realidad no es una tarea sencilla. Actualmente generar un simulador representativo de la realidad, que considere la dinámica de las interacciones entre los objetos a un bajo nivel, es una tarea factible, gracias a los avances en el poder computacional y el desarrollo de software especializados. Sin embargo, consideramos que la generación de un simulador realista no solo es materia de modelamiento y diseño, sino también de un proceso de adaptación, tal como se propone en [22].

1.2. *Back to Reality*

Los experimentos expuestos en este trabajo se basan en el paradigma *Back to Reality* de robótica evolutiva, recientemente propuesto por los autores [22]. La principal idea de este enfoque es permitir a los robots interactuar en su entorno real y en una simulación de éste. Una medida del *fitness* asociado a cada comportamiento se determina luego de la ejecución del comportamiento en el

ambiente real y el simulado. La diferencia existente entre el *fitness* obtenido en el ambiente real y el *fitness* resultante de la ejecución del mismo comportamiento en el ambiente simulado, es empleada como medida de la discrepancia existente entre ambos ambientes y es minimizada por medio de un proceso de adaptación evolutiva del simulador. De esta manera la ejecución del comportamiento robótico permite adaptar paulatinamente el simulador, el cual equivale a una representación computacional del entorno del robot. Esta adaptación es función de la tarea que debe desempeñar el robot. Luego, distintas representaciones o simulaciones del mundo surgen para distintas tareas o comportamientos a realizar. Si bien los modelos empleados por el simulador pueden no representar fielmente a los procesos físicos reales, *Back to Reality* permite encontrar un punto de operación para la ejecución de la tarea específica a desarrollar. De esta manera se consigue liberar al proceso, no solo del sesgo introducido en el diseño del controlador, sino también del sesgo introducido en la calibración del simulador.

Tal como se muestra en la figura 1, los parámetros de la simulación se ajustan continuamente, estrechando la diferencia que existe entre la realidad y la simulación respecto del *fitness* asociado al comportamiento específico a desarrollar. Dado que la adaptación del simulador depende del comportamiento en ejecución, éste define la manera en la cual el mundo es muestreado u observado. Durante el proceso de adaptación existen tres etapas de aprendizaje; primero, el aprendizaje del controlador robótico en el ambiente simulado; segundo, el aprendizaje del controlador robótico en la realidad; y finalmente el aprendizaje del simulador a partir de las discrepancias entre el *fitness* obtenido en la realidad y en la simulación. Cuando el robot aprende en la realidad, el controlador se encuentra *estructuralmente acoplado* con la realidad. Cuando éste aprende en la simulación, el controlador está *estructuralmente acoplado* con el simulador. Dados estos dos acoplamientos del controlador surge la necesidad de concebir al robot como la unión de su controlador, su hardware (cuerpo, sensores, actuadores y electrónica interna) y su simulador.

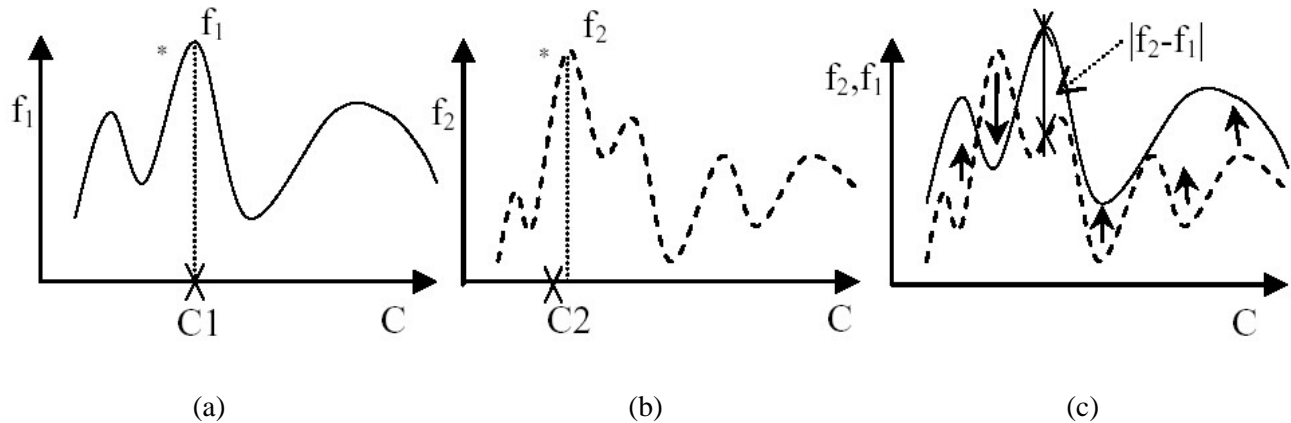


Figura 1: Ilustración de la manera en que la diferencia existente entre la realidad y la simulación es reducida mediante la aplicación del paradigma *Back to Reality*. En (a)/(b) se muestra la curva de *fitness* f_1/f_2 sobre el espacio de parámetros que definen el controlador en la realidad/simulación; el valor máximo del *fitness* f_1^*/f_2^* corresponde al controlador C_1/C_2 . En (c) se ilustra como la diferencia entre ambas curvas se reduce durante el proceso evolutivo; la curva de *fitness* del simulador, definida por los parámetros del simulador, converge hacia la curva de *fitness* real, empleando la diferencia de *fitness* real versus *fitness* obtenido en simulación como variable de ajuste.

1.3. Pegarle a la Pelota Un Comportamiento Complejo Relevante en RoboCup

La RoboCup es la federación mundial de fútbol robótico (www.robocup.org). Una de las ligas más destacadas dentro de RoboCup es la de robots cuadrúpedos Sony AIBO, en la cual equipos de cuatro robots cuadrúpedos se enfrentan. Lograr que un robot cuadrúpedo golpee efectivamente a la pelota con potencia, rapidez de reacción y precisión, son requerimientos fundamentales para cualquier equipo de fútbol robótico. Hasta el momento todos los equipos de la liga de robots cuadrúpedos de RoboCup emplean aproximadamente el mismo método para golpear la pelota: “primero auto-localizar al robot con respecto a la pelota y la posición final que ésta debe alcanzar, luego realizar alguno de los golpes previamente diseñados para el robot”. Nosotros consideramos que este enfoque es muy restringido debido a que: (1) requiere que la pelota esté posicionada respecto del robot en un reducido conjunto discreto de posiciones, de esta manera se debe invertir tiempo valioso en reposicionar al robot; (2) existe solo un conjunto discreto de direcciones objetivo a escoger al momento de lanzar la pelota (los jugadores de fútbol humanos son capaces de golpear el balón en forma libre, i.e. desde casi cualquier posición relativa y con cualquier dirección objetivo). Resulta interesante

mencionar que dentro de la liga de cuadrúpedos de RoboCup podemos notar una relación directa entre la cantidad de maneras en que un equipo puede golpear la pelota y su rendimiento al jugar fútbol.

Existen algunos requerimientos básicos que deben satisfacerse antes de pensar en aplicar comportamientos no restringidos para golpear la pelota. Primero, los robots deben ser capaces de estimar con precisión la posición de la pelota en el instante de impacto; y segundo, deben ser capaces de controlar con precisión sus extremidades. Nosotros consideramos que estos requerimientos son actualmente satisfechos, por lo menos por la mitad de los equipos que compiten en la liga. Por lo tanto es un buen momento para explorar este nuevo enfoque. Aún más, esta habilidad será de importancia fundamental al jugar fútbol con robots humanoides. Es de esperar que en un futuro cercano veamos comportamientos veloces y efectivos en robots capaces de desplazarse y golpear la pelota simultáneamente, incluso con capacidad para golpearla mientras ésta se encuentra en el aire, tal como los jugadores humanos pueden hacerlo.

Otro aspecto importante que queremos abordar en este trabajo es el relacionado con el aprendizaje. Los jugadores humanos adquieren habilidades sensor-motrices de bajo nivel, a través de extensos períodos de entrenamiento que pueden

durar años. Por lo mismo, creemos que este tipo de comportamientos reactivos complejos debe ser el producto de una estrecha interacción entre el robot y el medio ambiente. Golpear adecuadamente la pelota utilizando robots cuadrúpedos parece ser una tarea que requiere de mucho entrenamiento y por tanto mucho tiempo. Nosotros proponemos emplear UCHILSIM, un sofisticado simulador de robots, en conjunto con el enfoque *Back to Reality* para combinar eficientemente el aprendizaje en ambientes reales y ambientes virtuales. Nuestro objetivo es producir un comportamiento que permita golpear efectivamente la pelota desde cualquier posición relativa inicial y con cualquier dirección objetivo final que el robot pueda alcanzar sin la necesidad de reposicionarse.

1.4. Estructura del Artículo

Este artículo está organizado de la siguiente manera. En la sección 2 se presenta el simulador UCHILSIM. En la sección 3 se presenta la metodología empleada para aprender a pegar a la pelota con robots AIBO. En la sección 4 se presentan experimentos de validación. Finalmente, en la sección 5 se presentan las conclusiones de este trabajo.

2. EL Simulador UCHILSIM

UCHILSIM es un simulador de ambientes robóticos diseñado para la liga de robots cuadrúpedos de RoboCup. Está diseñado sobre la base de dos motores de software de procesamiento; uno a cargo de reproducir la dinámica de cuerpos rígidos articulados requerida para la simulación, y el otro, a cargo de generar una representación gráfica adecuada de los objetos dentro del escenario de juego.

El simulador también incluye una variedad de funcionalidades de interfase que permiten conectar el núcleo del sistema con sistemas de aprendizaje y con los módulos de control UChile1, el cual corresponde a un paquete de software que permite

controlar a un equipo de robots cuadrúpedos, gobernando los servomotores, procesando y analizando los datos de las cámaras, estimando la localización de los jugadores y en definitiva controlando a todos los robots durante un partido de fútbol robótico [16]. El simulador incluye también una completa interfase gráfica (ver figura 2).

El principal objetivo de diseño de UCHILSIM es proveer un ambiente virtual que permita a robots virtuales adquirir comportamientos al aprender en un medio representativo de la realidad. La idea es que las mismas funciones de control puedan luego transferirse al robot real. El objetivo de largo alcance del simulador es permitir la generación de comportamientos complejos para jugadores de fútbol robótico, al combinar eficientemente aprendizaje en realidad y en simulaciones [25].

La mayor parte de los modelos de dinámica de cuerpo rígido en UCHILSIM son computados empleando la librería *Open Dynamics Engine* (ODE) [17], la cual corresponde a una librería de software para la simulación de la dinámica de cuerpos sólidos rígidos articulados en ambientes virtuales. ODE es rápida, flexible y robusta; permite definir una variedad de juntas o articulaciones entre cuerpos y puntos de contacto bajo fricción. En ODE las ecuaciones de movimiento se resuelven mediante un modelo de multiplicadores de Lagrange [17]. Se emplea un modelo de contacto y fricción de Coulomb el cual se resuelve por medio de un método de Dantzing LCP [17]. En UCHILSIM se presta especial atención al modelamiento de los motores servo y de la pelota.

La representación gráfica de los objetos en UCHILSIM se obtiene mediante la librería gráfica *Open Graphical Library* (OpenGL) [18]. El modelo CAD de los robots AIBO se generó a partir de un modelo provisto por Sony [19].



Figura 2: Ilustración del software UCHILSIM, en su interfaz de usuario pueden apreciarse distintas opciones de visualización.

2.1 Arquitectura Básica

En la figura 3 se ilustra la arquitectura básica de UCHILSIM. Puede observarse como las principales funcionalidades del simulador se conectan con una variedad de aplicaciones. En el núcleo del simulador el motor dinámico coopera estrechamente con el motor gráfico generando una representación fiel de cada objeto. Por medio de una interfase de aprendizaje un conjunto de parámetros se intercambia con algoritmos de aprendizaje que se ejecutan de manera independiente al simulador. Estos parámetros se emplean usualmente para definir las variables de control del simulador, que se encuentran bajo adaptación durante el proceso de aprendizaje. La interfase de usuario permite variar, durante la operación del sistema, distintas variables de la simulación, tales como la manera en que los objetos son desplegados en la pantalla, así como también la manipulación externa de objetos dentro de la simulación. Una interfaz de aplicación permite ejecutar el paquete UChile1 dentro de la simulación. Actualmente se encuentra en desarrollo una interfaz de aplicación para integrar el sistema Open-r [19]. Esto permitirá compilar bajo el ambiente de simulación cualquier código Open-r. Consideramos que este tipo de herramienta será muy relevante para el desarrollo de la liga de cuadrúpedos de RoboCup puesto que permitirá simular en forma realista competencias

entre cualquier par de equipos de la liga de cuadrúpedos.

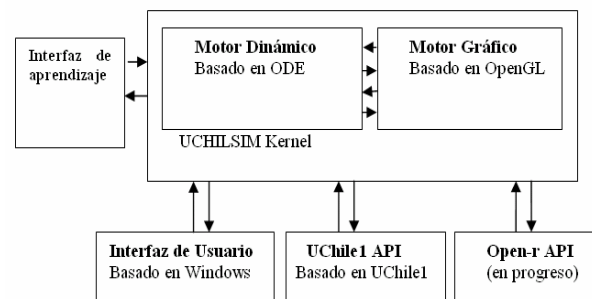


Figura 3: Ilustración de la arquitectura UCHILSIM. Puede observarse como el sistema está organizado como un conjunto de interfaces al rededor de un núcleo que contiene los motores gráficos y dinámicos.

2.2 Motor Dinámico

En UCHILSIM todos los elementos del robot AIBO y del campo de juego se modelan mediante cuerpos rígidos articulados, tales como paralelepípedos o esferas, que se conectan por medio de diferentes tipos de juntas.

Una junta corresponde a una restricción dinámica impuesta entre dos cuerpos de manera tal que ellos pueden tener solo ciertas posiciones relativas. En el caso del robot AIBO, se emplean modelos de junta universal para definir la relación entre el torso del AIBO y los muslos, i.e.

la rotación es restringida a dos grados de libertad. Junturas de tipo bisagra simple se emplean para definir la relación entre los muslos y las canillas del robot, i.e. la rotación se restringe a un grado de libertad. Modelos de junta fija se emplean para definir la relación entre las canillas y las pezuñas, en este caso solo se admite una dirección de deformación, de manera tal que el modelo representa fielmente la rotación de las pezuñas respecto a las canillas. Además pequeñas esferas se han agregado al final de cada pierna por medio de juntas fijas, las cuales se emplean para imitar el efecto de los topes de las piernas del robot.

La distribución de masa de cada cuerpo rígido se especifica en forma de tensores de inercia. Para esta implementación asumimos una distribución uniforme de la masa de cada cuerpo rígido. Se realizaron estimaciones de la masa de cada cuerpo con un dispositivo de medición.

La detección de colisiones se lleva a cabo mediante un modelo simple de esferas y paralelepípedos, o mediante el empleo de una versión simplificada de la malla de polígonos que se emplea para la representación gráfica de los cuerpos sólidos. Este último enfoque consume bastante tiempo de procesador, pero es más preciso. Hemos obtenido buenos resultados en

todos los experimentos empleando sólo el modelo de esferas y paralelepípedos. En la figura 4 se muestra un diagrama de las figuras geométricas que se ajustan a cada cuerpo rígido para desarrollar detección de colisiones. También se muestra la localización de los servomotores que se incluyen en nuestro modelo. Estos aplican torque sobre las juntas de acuerdo a la señal de salida de un controlador PID que recibe referencias angulares, entregadas por el módulo de actuación del paquete de control UChile1.

En cada paso de simulación las ecuaciones de dinámica son computadas y un nuevo estado es computado para cada cuerpo rígido. Este estado es definido por las velocidades, aceleraciones, velocidades angulares, etc. de cada cuerpo. Luego se realiza la detección de colisiones y las fuerzas resultantes se aplican sobre los cuerpos correspondientes, transmitiendo el efecto de las colisiones a lo largo de todo el cuerpo del robot o elementos en juego. En este caso se presta especial atención a los parámetros de fricción empleados para calcular las fuerzas de reacción entre las articulaciones del robot y la alfombra. Estos parámetros se encuentran bajo adaptación automática en cada experimento desarrollado.

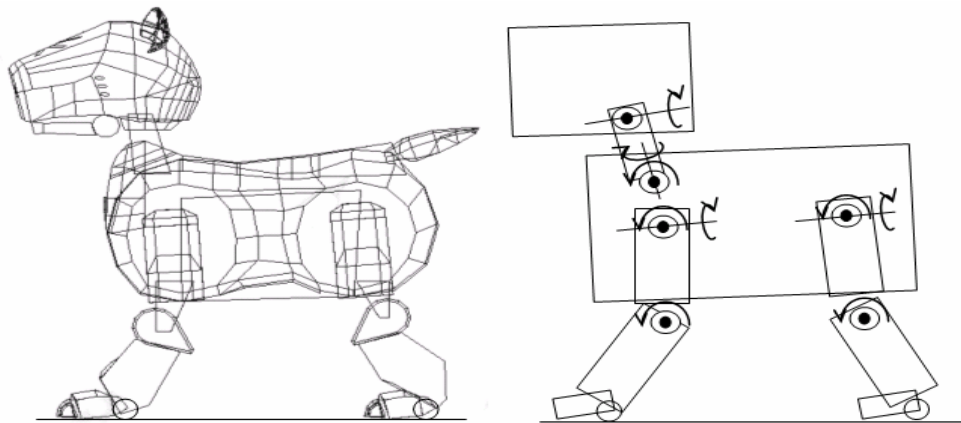


Figura 4: Modelos de detección de colisiones que se emplean en forma alternativa por el simulador UCHILSIM. La figura de la izquierda corresponde a la malla de polígonos simplificada del modelo gráfico. La figura en el lado derecho corresponde al modelo generado con el conjunto de paralelepípedos y esferas. El empleo de este último modelo permite realizar modelos dinámicos precisos mientras se mantiene la simulación en tiempo real. La figura también muestra la posición de los servomotores que se emplean para modelar el robot.

2.3 Motor Gráfico

Al final de cada paso de simulación el motor dinámico se encarga de calcular las correspondientes matrices de posición y rotación de cada cuerpo en el espacio de la simulación. Entonces, para cada cuerpo rígido, los correspondientes objetos gráficos son desplegados en pantalla siguiendo un proceso de *rendering* (*clipping*, eliminación de caras ocultas, etc.). Esto se realiza al cargar eficientemente los correspondientes datos gráficos.

El motor gráfico se encarga asimismo de generar la imagen que se obtiene desde la cámara que el robot AIBO posee. Esto es bastante relevante para generar experimentos con los sistemas de visión. En particular, empleando este sistema, produciremos una extensión del trabajo presentado en [24]. Hasta el momento no hemos concentrado nuestros esfuerzos en producir imágenes extremadamente realistas, pero estimamos que este proceso es bastante simple. En particular incorporaremos algunas de las transformaciones que fueron propuestas en [1].

Mediante el empleo de un software de modelamiento CAD hemos reconstruido el ambiente de juego de los robots. El proceso de importar datos gráficos hacia código C++ ha sido muy costoso y por tanto nos encontramos actualmente trabajando en una herramienta que permite cargar directamente dentro de nuestro simulador objetos en formato LightWave [6] o VRML [20].

2.4 Interfase de Usuario

La interfase de usuario del simulador actualmente entrega diversas funcionalidades, entre las que destacan el permitir reubicar objetos, tales como robots y pelotas, en cualquier instante de la simulación. Permite mover objetos en forma arbitraria, mientras ellos se encuentran en movimiento. Esto es particularmente útil para realizar experimentos donde exista interacción entre el robot y los usuarios. También permite modificar en línea distintos parámetros del

controlador UChile1. La interfase permite modificar distintas opciones de *renderizado* y visualización, tales como representación de malla de alambres, representación por *bounding boxes*, representación de objetos por vértices, etc. La interfase permite asimismo manipular las imágenes que son capturadas con la cámara del robot AIBO. Es posible exportar dichas imágenes en forma de archivos, o estas pueden transmitirse hacia sistemas de aprendizaje externos. La interfaz permite cargar y grabar archivos de configuración que también definen las condiciones de un determinado juego. También permite el manejo eficiente de ventanas dentro de la pantalla.

2.5 Interfase de aprendizaje

UCHILSIM permite variar los parámetros que definen la simulación y el sistema de control empleado en el robot durante la ejecución del programa. Esto es particularmente atractivo puesto que el programa no requiere de reinicializaciones para la ejecución de algoritmos de aprendizaje externos. El simulador está diseñado para comunicarse con otros programas vía TCP/IP. Todas estas consideraciones responden a la necesidad de emplear el simulador en conexión con el software de aprendizaje en el contexto del enfoque *Back to Reality*.

2.6 UChile1 y Open-r API

Todo el paquete de software UChile1, que permite a un equipo de robots cuadrúpedos AIBO jugar fútbol autónomamente, puede compilarse bajo el sistema de simulación UCHILSIM. En general esto implica la realización de varias modificaciones en nuestro código. El simulador se ha transformado en una importante herramienta para revisar el desempeño de nuestro software. Actualmente se está trabajando en la generación de una interfaz de aplicación que permita la compatibilidad entre cualquier código existente en la liga y nuestro simulador.

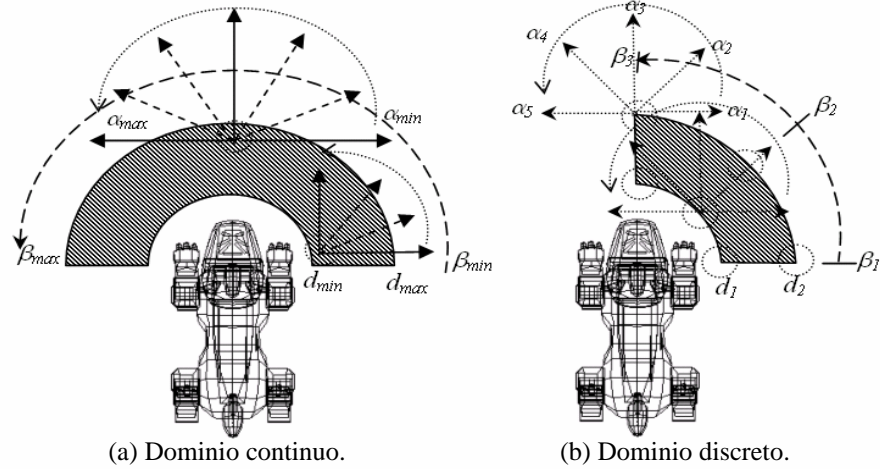


Figura 5: Ilustración del dominio del comportamiento definido por un conjunto de diferentes posiciones iniciales de la pelota y direcciones objetivo. Estas se muestran primero en (a) para el comportamiento continuo a ser generado y en (b) para el conjunto discreto considerado para aprendizaje.

3 Aprendiendo a pegarle a la pelota

Nuestro objetivo es desarrollar mediante aprendizaje un comportamiento que permita al robot AIBO golpear la pelota en forma óptima introduciendo un mínimo de sesgo en el diseño. Tal como se muestra en la figura 5 (a), el comportamiento deseado debe operar sobre un continuo de posiciones accesibles de la pelota y un continuo de direcciones objetivo. Nosotros definimos el dominio del comportamiento como el espacio continuo de tres dimensiones compuesto por la posición de partida de la pelota a distancia d [14cm→18cm], medida desde el cuello del robot, el ángulo relativo al robot \mathbf{b} [0°→180°], medido desde el lado derecho del robot, y la dirección objetivo de lanzamiento definida por el ángulo \mathbf{a} [0°→180°], medida desde el lado derecho del robot.

Para generar un golpe óptimo desde cada punto de este dominio consideramos un conjunto discreto de puntos para el aprendizaje, tal como se muestra en la figura 5 (b); consideramos dos distancias de la pelota $\{d_1, d_2\}$, tres ángulos relativos $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ y cinco ángulos objetivo $\{\mathbf{a}_1, \mathbf{a}_2, \alpha_3, \mathbf{a}_4, \mathbf{a}_5\}$, i.e. treinta puntos en total. Hemos seleccionado solo la mitad derecha del rango de la variable \mathbf{b} dado que esperamos que las soluciones sean simétricas al rededor de $\mathbf{b} = 90^\circ$. Para cada uno de estos treinta puntos, un golpe específico será generado por medio del

aprendizaje de un conjunto de parámetros que definen la secuencia de desplazamientos articulares del robot, dando lugar al golpe de la pelota.

El comportamiento continuo final, para un punto arbitrario, i.e. posición de la pelota $\{d, \mathbf{b}\}$ y ángulo objetivo $\{\mathbf{a}\}$, se obtendrá al interpolar, en el espacio de parámetros, las soluciones que se obtienen en alguno de los 30 puntos vecinos previamente definidos. En todos estos casos la idea es maximizar la distancia recorrida por la pelota, manteniendo la precisión en la dirección de disparo resultante.

3.1 Parametrización del golpe

La manera en la cual se parametriza el golpe de la pelota puede ser una importante fuente de sesgo introducida por el diseñador. Por lo tanto es necesario ser cuidadoso asegurando flexibilidad en el diseño. A continuación presentamos una parametrización que intenta servir como un compromiso entre simplicidad de aprendizaje y flexibilidad para permitir variadas soluciones. Bajo un punto de vista físico debemos tener en cuenta que una mayor cantidad de energía puede transmitirse a la pelota cuando todas las articulaciones del robot son libres de moverse. Por ejemplo, las piernas que soportan el cuerpo pueden desplazar el torso de manera tal que, su velocidad relativa al suelo, sea agregada a la

velocidad existente entre el extremo de la articulación que hace contacto efectivo con la pelota y el torso. Es importante también proveer libertad con respecto al lugar sobre la superficie de la pelota en el cual se aplican las fuerzas. Esta libertad puede permitir al robot descubrir, por ejemplo, que pegarle a la pelota en su porción superior es más efectivo que pegarle en su porción inferior (tal como ocurre en el juego de billar). Hay que notar que el simulador UCHILSIM se ajusta bien a este nivel de detalle.

El estado articular o la configuración angular de cada articulación está definida por la tripleta $\{\nu, \omega, \gamma\}$. La trayectoria de una articulación durante un golpe de pelota, puede describirse empleando cientos de puntos en el espacio. Sin embargo, nosotros estamos analizando solo el caso de golpes reactivos rápidos en los cuales no es necesario realizar un reposicionamiento del robot respecto de la pelota. Para este caso nosotros estimamos que es suficientemente adecuado el considerar solo cuatro puntos extremos del desplazamiento de cada articulación: (1) la configuración inicial de la articulación, (2) una configuración articular en la cual se acumula energía, (3) una configuración que define el punto de liberación de energía, i.e. el lugar donde se realiza contacto con la pelota, y (4) la configuración final de la articulación. En este caso consideramos la configuración inicial y final como iguales. Por lo tanto, para una configuración dada de posición de pelota y ángulo objetivo, un golpe de pelota corresponde a la ejecución de las siguientes cuatro etapas, que se ilustran en la figura 6:

1. Las articulaciones del robot se ubican en su posición inicial de equilibrio, la cual corresponde a la posición de equilibrio de la caminata de régimen permanente.
2. Se realiza una transición desde el punto de equilibrio inicial de cada articulación hacia su correspondiente punto de acumulación de energía $\{\nu_{i1}, \omega_{i1}, \gamma_{i1}\}$, con $i=[1, \dots, 4]$.
3. Se realiza otra transición para liberar la energía acumulada, en el instante de contacto con la pelota según la configuración descrita por $\{\nu_{i2}, \omega_{i2}, \gamma_{i2}\}$, con $i=[1, \dots, 4]$.

4. Finalmente las articulaciones del robot regresan a su configuración de equilibrio inicial.

En este trabajo deben obtenerse 25 parámetros por medio de aprendizaje, estos corresponden a los seis parámetros $\{\nu_{i1}, \omega_{i1}, \gamma_{i1}, \nu_{i2}, \omega_{i2}, \gamma_{i2}\}$ de cada una de las piernas del robot, y a un parámetro que controla la velocidad de las articulaciones, incorporando cierta cantidad de puntos dentro de la trayectoria que se entregan como referencias intermedias a los servomotores de cada articulación. Un número mayor de puntos de articulación hace más lento el movimiento de las piernas.

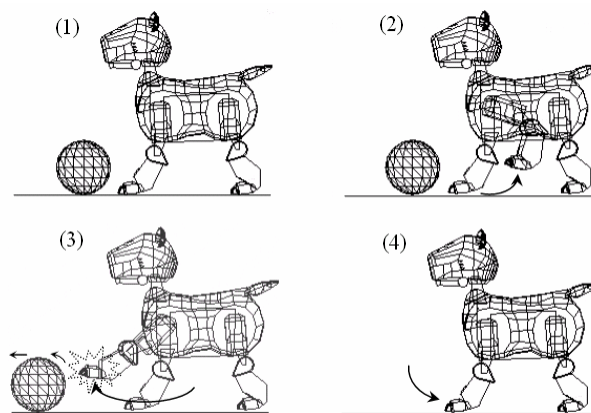


Figura 6: Ilustración de las configuraciones consecutivas de una de las articulaciones del robot durante la ejecución de un golpe. La articulación parte de una configuración de equilibrio (1), luego pasa a una configuración de acumulación de energía (2), entonces se dispara el golpe hacia la pelota (3) y finalmente el robot retorna a la posición de equilibrio inicial (4). Hay que notar que por simplicidad esta ilustración solo cubre la acción de una articulación. En este trabajo se considera el desplazamiento de las cuatro piernas.

3.2 Parámetros del Simulador

El simulador robótico resuelve ecuaciones de movimiento derivadas de un modelo de multiplicadores de Lagrange [17]. Hemos seleccionado un conjunto de 12 parámetros que determinan la configuración del simulador en términos de la dinámica del ambiente y variables del modelo del robot. Estos parámetros incluyen variables empleadas para resolver ecuaciones de dinámica de cuerpo rígido y las constantes que

definen el controlador PID empleado para modelar las articulaciones motorizadas del robot. Existen 4 parámetros para la distribución de masa del robot los que corresponden a la masa de la cabeza, la masa del cuello, la masa del torso, y la masa de las piernas; cuatro parámetros que describen el modelo dinámico: la constante de fricción, la constante de gravedad, el coeficiente de resbalamiento en dos direcciones diferentes; y finalmente cuatro parámetros que describen el modelo articular de las piernas: la constante proporcional del controlador PID, la constante integral del controlador PID, la constante diferencial del controlador PID y el máximo torque que el dispositivo puede aplicar.

3.3 Búsqueda Genética

Se emplean algoritmos genéticos para explorar espacios paramétricos que definen el controlador y el simulador. En el caso del controlador la búsqueda se conduce hacia puntos en que se maximiza la función de *fitness* de este. En el caso del simulador la búsqueda es conducida con el objetivo de minimizar la diferencia entre el *fitness* obtenido en la ejecución de cada comportamiento en la simulación y en la realidad. Cada parámetro es representado por palabras de 8 bits. Por lo tanto se genera un cromosoma de largo 8x25 para determinar un controlador y un cromosoma de largo 8x12 para determinar el simulador. Se define un rango de variación para cada parámetro p_i , de tal manera que $p_i \in [p_{i_{\min}}, p_{i_{\max}}]$. Se emplea un algoritmo genético con selección proporcional al *fitness*, escalamiento lineal, esquema no elitista, *crossover* de dos puntos con probabilidad 0.75 de combinar dos individuos y mutación con probabilidad de mutación por bit de 0.015.

3.4 Búsqueda Genética

Para una posición dada de la pelota y un ángulo objetivo específico se desea encontrar un golpe que maximice el desplazamiento de la pelota y al mismo tiempo posea una elevada precisión en la dirección resultante. Derivar una buena función de *fitness* para este propósito puede parecer bastante simple, sin embargo, es necesario considerar ciertos aspectos relevantes. En (1) se presenta la primera función de *fitness* que

empleamos durante nuestros experimentos que corresponde a la distancia euclidiana medida entre la posición final de la pelota y el cuello del robot, multiplicada por el coseno del ángulo de error \mathbf{y} entre la dirección resultante y la dirección deseada. Esta función se maximiza cuando el error es cero.

$$fitness = d \cos(\mathbf{y}) \quad (1)$$

Esta función ha demostrado ser inútil para nuestros propósitos, puesto que la búsqueda genética se concentra en encontrar primero individuos que son capaces de disparar la pelota a gran distancia sin tomar en cuenta la precisión angular. La razón es que la función coseno es muy plana en torno a cero. Para resolver este problema hemos escogido una función exponencial tal como se muestra en (2). Empleando esta función hemos observado que la búsqueda genética se concentra primero en generar lanzamientos con un ángulo adecuado y luego se concentra en maximizar la distancia recorrida por la pelota.

$$fitness = d \cdot e^{-ky^2} \quad (2)$$

3.5 Combinando Soluciones en el Espacio Paramétrico

Se espera que un comportamiento diferente se obtenga para cada uno de los 30 puntos pertenecientes al dominio discreto de comportamientos. Estos se obtendrán a través de búsqueda genética sobre el espacio de 25 parámetros que definen cada controlador empleando el enfoque *Back to Reality*. Sin embargo, el comportamiento global deseado debe desarrollar un golpe adecuado para un punto arbitrario $\{d, \beta, \alpha\}$ perteneciente a un continuo de puntos. Para hacer esto empleamos interpolación tri-lineal sobre los parámetros obtenidos. La figura 7 (a) muestra una ilustración del la grilla tridimensional que define el dominio de comportamientos a generarse por medio del aprendizaje. Cada vértice de la grilla contiene los 25 parámetros del mejor comportamiento resultante. La figura 7(b) muestra la interpolación tri-lineal que se desarrolla sobre cada uno de los 25 parámetros con el objeto de derivar el correspondiente valor interpolado. Un requisito

para que este procedimiento tenga resultados satisfactorios es que las soluciones vecinas deben producir comportamientos similares. De otra manera los golpes resultantes interpolados no tendrían sentido. Nosotros medimos esta similitud como la distancia Euclidiana de los vectores paramétricos. Dado que esta métrica es muy sensible al efecto de un elemento en particular del vector, también consideramos emplear la distancia de Hamming en futuros experimentos.

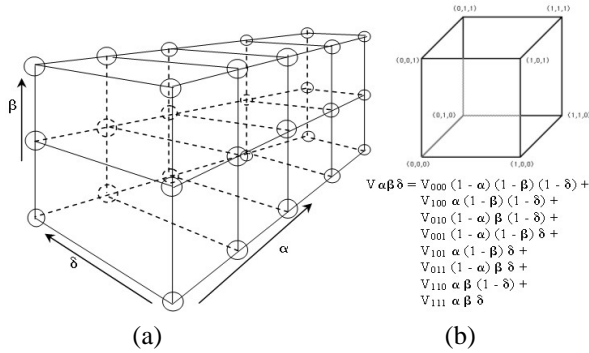


Figura 7: (a) La grilla paramétrica tridimensional del comportamiento. Cada vértice representa un conjunto de 25 parámetros que se obtienen vía búsqueda genética. (b) La expresión de la interpolación tri-lineal que se emplea para obtener los 5 parámetros en un punto arbitrario $\{d, b, a\}$ perteneciente al dominio del comportamiento.

4 Experimentos

En esta sección se presentan experimentos que fueron desarrollados con la metodología presentada en 4. Primero presentamos algunos ejemplos de la evolución del *fitness* y comportamientos resultantes de aprender un solo punto aislado dentro del dominio del comportamiento a generar. Finalmente presentamos distintos resultados en el aprendizaje de toda la grilla discreta y luego de la obtención de un comportamiento continuo global.

4.1 Aprendiendo Puntos Aislados

Para aprender un golpe en particular primero se genera en forma aleatoria una población de controladores de acuerdo a 3.3. Una solución conocida inicial fue empleada para definir el conjunto de 12 parámetros que definen el

simulador (esta solución se obtuvo de experimentos anteriores en el aprendizaje de caminatas [21]). Entonces el proceso continúa de la siguiente manera:

1. Cada controlador es probado en el simulador y el *fitness* es calculado de acuerdo a (2). El proceso evolutivo se desarrolla a lo largo de 50 generaciones y luego se detiene desplegando una alarma hacia el usuario.
2. Se requiere de intervención humana para probar en la realidad el conjunto de los 10 mejores individuos. Para esto se mide en forma manual el *fitness* correspondiente. Es necesario notar que este proceso puede ser fácilmente automatizado, sin embargo este no es el principal objetivo del presente trabajo.
3. Se realiza la evolución del simulador minimizando las diferencias promedio de *fitness* del controlador resultante obtenido en la realidad versus el *fitness* obtenido en la simulación. Para esto cada comportamiento resultante es ejecutado repetitivamente empleando diferentes parámetros en el simulador, los cuales se obtienen del proceso genético de búsqueda. Esta etapa se desarrolla en cada oportunidad a lo largo de 10 generaciones.
4. Una vez que los parámetros del simulador se han ajustado, todo el proceso de adaptación continúa volviendo al punto 1. Sin embargo las actualizaciones del simulador se ejecutan ahora solo cada 100 generaciones en vez de 50.

Este proceso se realiza para cada uno de los 30 puntos pertenecientes a la grilla del dominio del comportamiento a construir.

El proceso anteriormente descrito fue conducido a lo largo de 500 generaciones para cada punto perteneciente al dominio del comportamiento. En la figura 8 se presenta la curva de evolución del *fitness* para el punto $\{d=14cm, b=90^\circ, a=90^\circ\}$, i.e. donde la tarea consistía en lanzar la pelota cuando está ubicada a 14cm justo al frente del robot, con una dirección de destino frontal. Las líneas verticales en la figura representan las etapas de interrupción donde el simulador fue adaptado empleando como retroalimentación las discrepancias del *fitness*

obtenidas de evaluar los mejores 10 comportamientos en el ambiente real. Es posible apreciar como las actualizaciones en el proceso de simulación producen diferencias (saltos) en el *fitness* máximo y promedio a lo largo del proceso de adaptación del controlador. Hemos observado que al inicio del proceso los comportamientos resultantes en la simulación no podían transferirse directamente a la realidad, i.e. el *fitness* resultante es un resultado muy discrepante, sin embargo, al final del proceso evolutivo (mas allá de la generación 300), los comportamientos resultantes fueron casi directamente transferibles a la realidad.

Resultados similares se obtuvieron para los restantes 29 puntos pertenecientes al dominio del comportamiento. Sin embargo el aprendizaje no fue logrado exitosamente en algunos puntos como por ejemplo $\{d=14cm, b=0^\circ, a=180^\circ\}$. Ciertamente no es posible que el robot golpee la pelota mientras ésta se encuentra ubicada justo a su lado derecho con dirección opuesta. Algunos puntos cercanos al anterior tampoco obtuvieron buenos resultados durante el aprendizaje. Consideramos que éste corresponde a un problema de diseño que puede ser solucionado simplemente al restringir el dominio del comportamiento para ángulos mayores que $b=0^\circ$. Para los experimentos expuestos en este estudio hemos reemplazado los puntos vecinos al punto conflictivo por las soluciones más cercanas que si entregan resultados satisfactorios (solo fue necesario en 4 casos).

Los comportamientos resultantes son bastante interesantes, aún más si consideramos que estos fueron obtenidos desde una solución arbitraria. Por ejemplo el robot aprende que una manera de golpear la pelota con elevada potencia es saltar con el torso sobre ella. Varios comportamientos interesantes fueron obtenidos en los que puede verse claramente el concepto de “acumular energía”. Algunos de los comportamientos no se ven muy bien dado que el robot termina en posiciones bastante extrañas, no obstante son realmente efectivos. En la figura 9 se muestra un ejemplo en que se desarrolla un golpe aprendido en el ambiente simulado UCHILSIM.

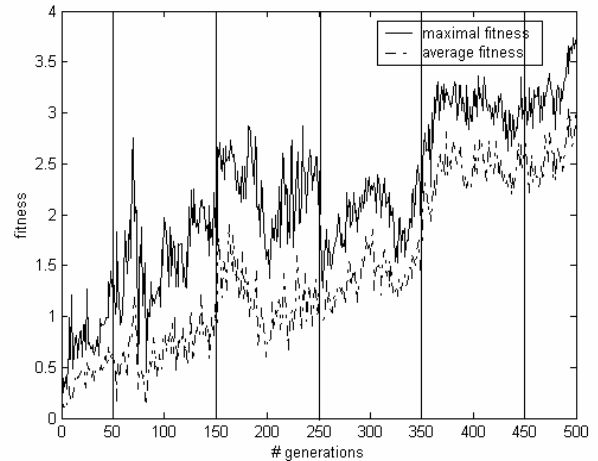


Figura 8: Curvas de evolución del *fitness* máximo y promedio resultantes de la adaptación del comportamiento generado para el punto $\{d=14cm, b=90^\circ, a=90^\circ\}$ perteneciente al dominio del comportamiento. Para este punto la tarea consiste en golpear la pelota cuando esta ubicada a 14cm justo al frente del cuello del robot con una dirección objetivo frontal. Las líneas verticales representan las etapas de interrupción donde el simulador es adaptado empleando como retroalimentación la diferencia de *fitness* resultante de evaluar comportamientos en la realidad y en la simulación. Es posible apreciar como ajustes en el simulador producen saltos en la curva a lo largo del proceso de adaptación.

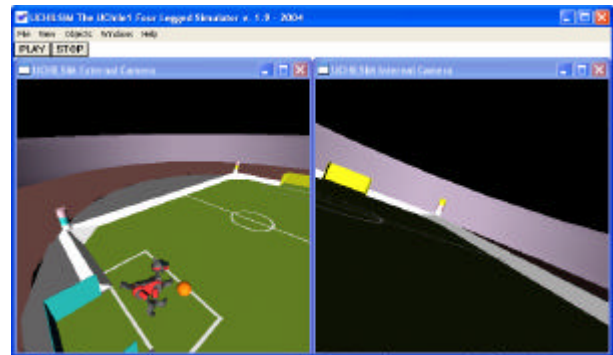


Figura 9: Ilustración del empleo del simulador UCHILSIM para aprender a pegarle a la pelota. Puede observarse en este caso un comportamiento curioso en el cual el robot primero empuja su cuerpo hacia adelante y luego golpea la pelota con su pierna derecha.

4.2 Aprendizaje de puntos combinados

A partir de la grilla que fue generada en la sección 4.1 es posible generar un comportamiento continuo global para cualquier punto arbitrario $\{d, \mathbf{b}, \mathbf{a}\}$, empleando interpolación tri-lineal tal como fue descrito en 3.5. Desafortunadamente no existe garantía respecto al buen desempeño de comportamientos resultantes de puntos diferentes de aquellos pertenecientes a la grilla. Esto se debe a la gran variedad de soluciones que fueron obtenidas. Por ejemplo dos puntos vecinos en la grilla producen un comportamiento similar, uno envía la pelota en la dirección 0° y el otro en la dirección 45° , sin embargo uno de ellos emplea la pierna izquierda mientras que el otro emplea la pierna derecha. La interpolación en el espacio paramétrico de estas soluciones da lugar a un comportamiento en el cual el robot simplemente se cae sin lograr contacto con la pelota. Bajo este escenario nosotros visualizamos 3 alternativas de solución: (1) seleccionar manualmente soluciones similares de la grilla sin tomar en cuenta su optimalidad; (2) conducir el proceso evolutivo de manera tal que el aprendizaje de cada nuevo punto se realice heredando elementos de puntos anteriores a través de la contaminación de individuos de la población; y (3) forzar la similitud entre los parámetros durante el proceso de aprendizaje mediante una redefinición de la función de fitness. Hemos producido un conjunto de buenas soluciones para un comportamiento continuo global empleando la segunda metodología. Partiendo de la solución presentada en 4.1 hemos contaminado a las poblaciones vecinas con un 80% de individuos pertenecientes a los puntos exitosos, y luego hemos repetido el proceso sobre el resto del dominio. En nuestra opinión este proceso no es eficiente, requiere de mucho tiempo y está en contra de la filosofía global de nuestro estudio. Sin embargo, funciona bien produciendo un comportamiento continuo para pegarle a la pelota.

5 Conclusiones y proyecciones

Se ha presentado una método para que robots cuadrúpedos aprendan un comportamiento complejo consistente en pegarle a la pelota en forma reactiva. El aprendizaje se realiza

combinando experiencia real y experiencia virtual de acuerdo al paradigma *Back to Reality*. Los comportamientos resultantes son directamente transferibles desde el simulador al robot real.

En la presente metodología el sesgo del diseñador del controlador robótico se reduce al mínimo, manteniéndose solo cuando: (i) se diseña una parametrización para golpear la pelota y (ii) se seleccionan individuos para producir un comportamiento continuo global. El comportamiento continuo resultante permite golpear la pelota con potencia, con un período de reacción corto y con gran precisión. EL método empleado se basa en el uso de UCHILSIM, un simulador realista en la representación dinámica de objetos, y en el paradigma *Back to Reality* de robótica evolutiva. Es necesario destacar que este método ha permitido aprender a pegarle a la pelota desde cero. Desarrollando golpes que se desempeñan en forma similar a los golpes más poderosos que actualmente existen en la liga, por ejemplo, el golpe en el cual el robot deja caer el torso sobre la pelota. Todos los golpes se obtuvieron luego de solo 8 horas de simulaciones. Debemos notar que la alternativa a este enfoque es que un ingeniero diseñe el comportamiento durante un período de tiempo probablemente mayor. Actualmente estamos trabajando en un método que fuerza la similitud entre los comportamientos pertenecientes a la grilla. La función de fitness contiene un factor que depende de la distancia Euclidiana entre vectores de parámetros que definen puntos vecinos. La clave de este enfoque esta en desarrollar la búsqueda en paralelo. De esta manera es posible mantener las 8 horas de aprendizaje para todo el comportamiento global continuo (empleando varios computadores).

Agradecimientos

Esta investigación ha sido apoyada por el Departamento de Ingeniería Eléctrica de la Universidad de Chile y por el proyecto FONDECYT 1030500.

Referencias

1. Asanuma, K., Umeda, K., Ueda, R., Arai, T.: Development of a Simulator of Environment and Measurement for Autonomous Mobile Robots

- Considering Camera Characteristics. Proc. of Robot Soccer World Cup VII, Springer (2003).
2. Endo, K., Yamasaki, F., Maeno, T., Kitano, H.: Co-evolution of Morphology and Controller for Biped Humanoid Robot. In: LNAI Proceedings of RoboCup 2002 Robot Soccer World Cup VI, Springer, pp. 327-339 (2002).
 3. Cliff, D., Husbands, P., Harvey, I.: Evolving Visually Guided Robots. In: Meyer, J. A., Roitblat, H., Wilson, S. (eds.): Proceedings of the Second International Conference on Simulation of Adaptive Behaviour (SAB92). MIT Press Bradford Books, Cambridge, MA (1993) 374-383.
 4. Golubovic, D., Hu, H.: An Interactive Software Environment for Gait Generation and Control Design of Sony Legged Robots. In: LNAI Proceedings of RoboCup 2002 Robot Soccer World Cup VI, Springer, pp. 279-287 (2002).
 5. Google Source Directory Resource of Robotics Simulation Tools <http://directory.google.com/Top/Computers/Robotics/Software/Simulation/> (2004).
 6. LightWave Newtek Products Home Page <http://www.newtek.com/products/lightwave/index.php> (2004).
 7. Hardt, M., Stryk, O.: The Role of Motion Dynamics in the Design, Control and Stability of Bipedal and Quadrupedal Robots. In: LNAI Proceedings of RoboCup 2002 Robot Soccer World Cup VI, Springer, pp. 206-221 (2002).
 8. Hoffmann, F., Zagal, J.C.: Evolution of a Tactile Wall-Following Behavior in Real Time. In: Roy, R., Köppen, M., Ovaska, S., Huruhashi, T., Hoffmann, F. (eds.): Soft Computing and Industry, Recent Applications. Springer (2002) 747-755.
 9. Hornby, G.S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O.: Autonomous Evolution of Gaits with the Sony Quadruped Robot. In: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 1297-1304 (1999).
 10. Husbands, P., Harvey, I.: Evolution Versus Design: Controlling Autonomous Robots. In: Integrating Perception, Planning and Action: Proceedings of the Third Annual Conference on Artificial Intelligence, Simulation and Planning, IEEE Press, pp. 139-146 (1992).
 11. Ishimura, T., Kato, T., Oda, K., Ohashi, T.: An Open Robot Simulator Environment. Proc. of Robot Soccer World Cup VII, Proc. of Robot Soccer World Cup VII, Springer (2003).
 12. Johnson, M., Maes, P., Darrell, T.: Evolving Visual Routines. In: Brooks, R., Maes, P. (eds.): Artificial Life IV. MIT Press (1994) 198-209.
 13. Nolfi, S., Floreano, D.: Evolutionary Robotics – The Biology, Intelligence, and Technology of Self-Organizing Machines, In: Intelligent Robotics and Automation Agents. MIT Press (2000).
 14. Nolfi, S.: Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration., In: Gomi, T. (eds.): Evolutionary Robotics: From Intelligent Robots to Artificial Life, Ontario, Canada, AAI Books (1997).
 15. Roefer, T.: German Team RoboCup 2003 Technical Report, Available at <http://www.germanteam.de> (2003).
 16. Ruiz-del-Solar, J., Zagal, J.C., Guerrero, P., Vallejos, P., Middleton, C., Olivares, X.: UChile1 Team Description Paper, In: Proceedings of the 2003 RoboCup International Symposium, Springer, (2003).
 17. Smith, Open Dynamics Engine Library, ODE web site available at <http://opende.sourceforge.net> (2003) .
 18. The Open Graphics Library. Available at <http://www.opengl.org> (2004)
 19. The Open-r Software Development Kit. Available at <http://www.openr.org> (2003)
 20. Web 3D Consortium Home Page <http://www.web3d.org/>
 21. Zagal, J.C., Ruiz-del-Solar, J.: Back to Reality: Crossing the Reality Gap in Evolutionary Robotics: Part I: Theory. Submitted to IAV 2004, 5th IFAC Symposium on Intelligent Autonomous Vehicles (2004).
 22. Zagal, J.C., Ruiz-del-Solar, J., Vallejos, P.: Back to Reality: Crossing the Reality Gap in Evolutionary Robotics. Accepted in IAV 2004, 5th IFAC Symposium on Intelligent Autonomous Vehicles (2004).
 23. Zagal, J.C., Ruiz-del-Solar, J.: Learning to Kick the Ball Using Back to Reality. Submitted to The RoboCup 2004 International Symposium. (2004)
 24. Zagal, J.C., Guerrero, P., Palma, R., Ruiz-del-Solar, J.: Evolving Visual Object Recognition for Legged Robots, In: LNAI Proceedings of RoboCup 2003 Robot Soccer World Cup VII, Springer, (2003).
 25. Zagal, J.C., Ruiz-del-Solar, J.: UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. Submitted to the RoboCup 2004 International Symposium (2004).