Simulación de un Robot Bidimensional Simple mediante una interfaz en Scheme

Este trabajo presenta los pormenores de un proyecto mediante el cual se pretende introducir amigablemente a la robótica a los alumnos de nuevo ingreso de la carrera de Ciencias de la Computación de una universidad norteamericana, mediante el uso de un simulador gráfico de un robot bidimensional simple. Se discutirán algunos de los aspectos matemáticos y de programación que surgieron durante el desarrollo de este programa, así como su inesperado uso como herramienta auxiliar en la enseñanza de Inteligencia Artificial y Programación Funcional, aprovechando el hecho de que fue desarrollado en Scheme.

Palabras Clave : Robótica, Scheme, Programación Funcional, Graficación, Computadoras en la Educación.

Introducción

Desde hace algún tiempo, un profesor del Departamento de Computación de la Universidad Tulane, (New Orleans, LA), en la que uno de los autores de este trabajo cursa estudios de doctorado, dio a conocer su preocupación ante el poco interés mostrado por los estudiantes hacia la robótica, área en la que él se especializa. Al parecer, dicho comportamiento se debe principalmente al hecho de que de forma tradicional suele asociarse a la robótica con dispositivos electro-mecánicos, y por ello los que tienen inclinación por ella piensan que las carreras de Ingeniería Mecánica o Eléctrica son las más apropiadas para dar cabida a tal disciplina. Sin embargo, esta suposición pasa por alto el hecho de que existen muchas sub-áreas dentro de la robótica que requieren de expertos en Ciencias de la Computación (e.g., la planeación de movimientos, el aprendizaje, etc.)

El origen de este problema se encuentra normalmente en el currículum de las carreras de Ciencias de la Computación, en los que en rara ocasión suelen incluirse cursos introductorios a la Robótica, debido principalmente a las mismas razones antes expuestas.

Ante esta situación, el citado profesor decidió desarrollar un pequeño proyecto mediante el cual se diseñaría un simulador gráfico de un robot bidimensional simple. Tal simulador serviría para introducir a los alumnos de primer ingreso al campo de la robótica de una manera muy amigable, mediante el planteamiento de sencillos problemas de planeación de movimientos que pondrían a prueba sus conocimientos de geometría plana y trigonometría. Adicionalmente, se decidió diseñar el simulador en un lenguaje funcional (Scheme, en nuestro caso) a fin de que los estudiantes desarrollaran, de paso, buenos hábitos de programación. Uno de los autores de este trabajo fungió precisamente como responsable de este proyecto, el cual se ha llevado a cabo satisfactoriamente, tras varios meses de arduo trabajo.

Actualmente, el sistema se encuentra funcionando, aunque to-

davía es muy pronto para poder manifestar si los resultados obtenidos han sido los que se esperaban, pero ya nos encontramos en una segunda etapa -no anticipada- del proyecto, en la que el simulador se ha empezado a probar en México, como auxiliar en la enseñanza de programación funcional en los cursos de Inteligencia Artificial, para lo cual el otro autor de este trabajo, ha introducido valiosas modificaciones al sistema.

Modelo Geométrico Básico del Robot

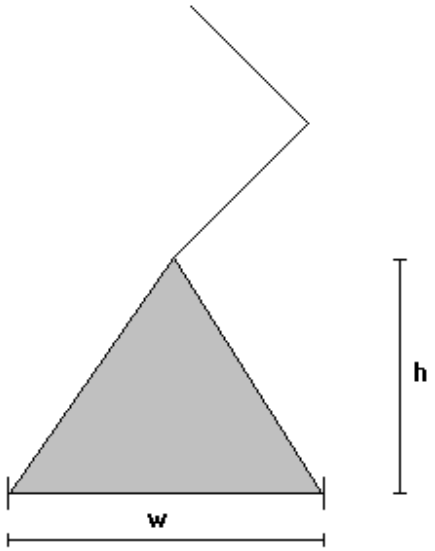

Figura 1 : Descripción geométrica del robot usado para el simulador. Los valores de **w** (ancho) y **h** (alto) del cuerpo, así como las orientaciones de los 2 extremos de su brazo se mantienen ocultos en el sistema.

Nuestro robot se modeló con un cuerpo triangular plano, el cual puede moverse mediante 2 ruedas, ubicadas en su parte inferior, y que cuenta en su parte superior con un brazo de 2 grados de libertad, tal y como se muestra en la Figura 1. El brazo se muestra más detalladamente en la Figura 2.

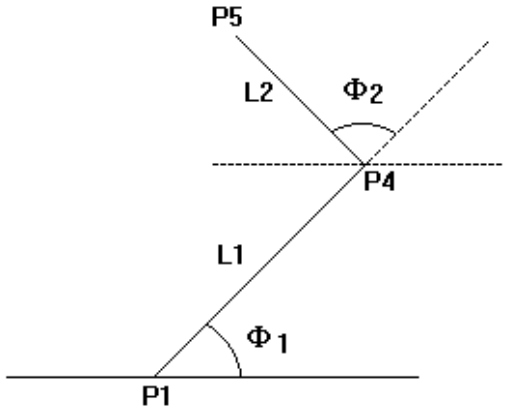

Figura 2 : Detalle del brazo del robot. Φ_1 y Φ_2 son las orientaciones respectivas de sus 2 extremos. **L1** y **L2** son las longitudes de las 2 partes del brazo. Toda esta información permanece oculta dentro del sistema.

Las características principales del sistema son las siguientes:

- El robot puede mover su cuerpo sólo a través de las 2 ruedas que posee en su parte inferior. Sin embargo, para mover las ruedas, el usuario debe proporcionar un ángulo (en grados) que se considera en el plano perpendicular al cuerpo del robot, como se indica en la Figura 3. De tal forma, cada rueda puede moverse independientemente a través de una única función, y sólo si movemos ambas lograremos desplazar al robot.

- El espacio donde el robot está ubicado se encuentra poblado de obstáculos y objetos. Los primeros deben evitarse, y cualquier contacto de una parte del robot con ellos es reportada por el sistema, anulándose como consecuencia el movimiento efectuado,

y haciendo que el robot se regrese a su posición anterior. Con los objetos, sin embargo, es posible usar 2 funciones: *sujetar* y *soltar*. La primera permite que el objeto quede adherido al imán imaginario que el robot porta en el extremo de su brazo, y con ello puede moverlo hacia el lugar que se desee. La segunda función simplemente hace que el robot suelte el objeto y que continúe su movimiento libre.

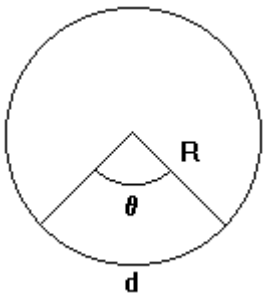

Figura 3 : Modelado de las ruedas del robot. El ángulo **θ** se maneja en grados, y es proporcionado por el usuario. Los valores de **R** (radio) y **d** (arco correspondiente al ángulo dado) se mantienen ocultos en el sistema.

- El sistema se encuentra *encapsulado* en su mayor parte, y al usuario sólo se le proporcionan ciertos parámetros tales como: un ángulo de referencia del cuerpo del robot, sus coordenadas y la posición de una o dos guías que se encuentran en el espacio del robot, y que servirán para orientarnos. Con esta información, el estudiante deberá ser capaz de escribir programas que resuelvan los retos planteados por el instructor.

- Los brazos pueden moverse en base a ángulos proporcionados por el usuario. Sin embargo, cualquier movimiento de la parte inferior del brazo implica uno de la parte superior.

Movimiento del Brazo del Robot

Si observamos la Figura 2, podemos derivar las ecuaciones básicas que modelan el brazo del robot. $\mathbf{P_1}$ es el punto superior del triángulo, y sus coordenadas (x_1, y_1) están ocultas para el usuario. Lo que queremos calcular son las coordenadas de $\mathbf{P_4}$ y $\mathbf{P_5}$. Las ecuaciones necesarias son las siguientes:

$$x_4 = x_1 + L_1 \text{ Cos } \phi_1$$
$$y_4 = y_1 + L_1 \text{ Sen } \phi_1$$
$$\Omega = 180 - (\phi_1 + \phi_2)$$
$$x_5 = x_4 - L_2 \text{ Cos } \Omega$$
$$y_5 = y_4 + L_2 \text{ Sen } \Omega$$

De estas ecuaciones puede verse cómo el movimiento de cualquiera de las 2 partes del brazo del robot está en función de los valores de ϕ_1 y ϕ_2. Cualquier modificación al primero de estos 2 ángulos, implica el movimiento de la segunda parte del brazo, si bien el valor relativo de ϕ_2 debe mantenerse. Por su parte, si se mueve únicamente la segunda parte del brazo, el valor de ϕ_1 permanece inalterado.

Movimiento del Cuerpo del Robot

Consideremos la nomenclatura mostrada en la Figura 4. El ángulo **θ** se proporciona al usuario, y es el ángulo de orientación del lado izquierdo del cuerpo del robot con respecto a la horizontal. Dado que se conocen las coordenadas de $\mathbf{P_1}$, sólo necesitamos calcular las coordenadas de $\mathbf{P_2}$ y $\mathbf{P_3}$.

Las ecuaciones necesarias para modelar el movimiento del cuerpo del robot son las siguientes:

$x_2 = x_1 - c \cos \theta$
$y_2 = y_1 - c \operatorname{Sen} \theta$
$\rho = 180 - 2\beta$
$\beta = \operatorname{Tan}^{-1}(2h/w)$
$c = \frac{1}{2}\sqrt{w^2 + 4h^2}$
$\Omega = 2\beta - \theta$
$x_3 = x_1 + c \cos \Omega$
$y_3 = y_1 - c \operatorname{Sen} \Omega$

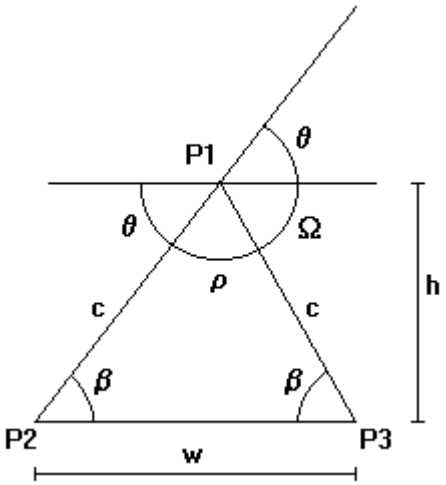

Figura 4 : Variables usadas para derivar las ecuaciones del movimiento del cuerpo del robot.

Movimiento de las Ruedas

Para poder derivar las ecuaciones del movimiento de las ruedas del robot, consideraremos que tienen una cierta dimensión **s**, y usaremos como puntos de referencia las coordenadas (x_2, y_2) y (x_3, y_3), tal y como se indica en la Figura 5.

Usando un poco de trigonometría, podemos construir fácilmente el triángulo mostrado en la Figura 6, el cual tiene como hipotenusa la mitad del tamaño de la rueda, vista ésta en un plano perpendicular a sus movimientos. De este triángulo podemos derivar las siguientes ecuaciones:

$\alpha = \operatorname{Sen}^{-1}\left(\frac{y_2 - y_3}{w}\right)$
$x_s = (s * \operatorname{Sen} \alpha)/2$
$y_s = (s * \cos \alpha)/2$

Las distancias $\mathbf{x_s}$ y $\mathbf{y_s}$ nos permitirán dibujar las ruedas del robot sin importar la posición del cuerpo del mismo.

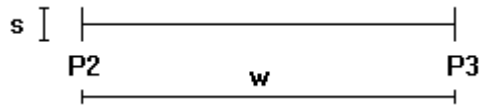

Figura 5 : Eje donde se encuentran colocadas las ruedas del robot. Los puntos $\mathbf{P_2}$ y $\mathbf{P_3}$ se usan como referencia para calcular sus posiciones relativas al cuerpo del mismo.

Para mover el cuerpo del robot, el usuario debe proporcionar un ángulo que indique cuánto se va a mover una rueda (i.e., el ángulo permitirá calcular el arco correspondiente, y ésto desplazará al robot). La Figura 3 muestra la forma en que modelaron las ruedas del robot. De ahí, resulta fácil derivar las ecuaciones:

$d = R\theta$
$R = s/2$

Dados estos valores, para poder calcular el movimiento del robot en su totalidad, tenemos que considerar los 3 casos siguientes:

1) <u>Sólo la rueda derecha se mueve</u> : En este caso, la Figura 7 muestra gráficamente lo que sucede. De ahí puede verse que:

$\phi = 2\ \text{Sen-1}(d_2/w)$

El nuevo valor de θ estará dado por:

$\theta' = \theta + \phi$

Estos valores se utilizarán para recalcular las coordenadas (x_1, y_1) y (x_3, y_3) con las fórmulas proporcionadas anteriormente.

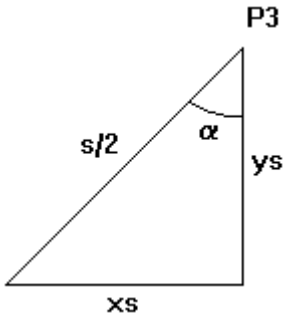

Figura 6 : Triángulo que se forma tomando como hipotenusa la mitad del valor de una de las ruedas del robot.

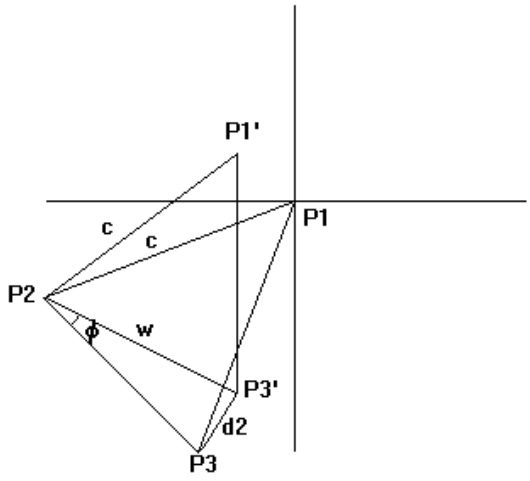

Figura 7 : Proyección del movimiento del cuerpo del robot en función del movimiento de la rueda derecha.

2) Sólo la rueda izquierda se mueve : En este caso, calculamos d_1 y ϕ. Estos valores nos permitirán recalcular las coordenadas (x_1, y_1) y (x_2, y_2).

3) Ambas ruedas se mueven : En este caso se combinan los efectos de los 2 puntos previos.

Chequeo de posibles colisiones

A fin de simplificar el chequeo de colisiones, se asume que todos los objetos y obstáculos que se definan estarán formados por segmentos rectilíneos. De tal forma, dado que el robot también se encuentra constituído por segmentos de las mismas características, el problema se reduce al chequeo de intersecciones entre segmentos rectilíneos, para lo cual usamos el algoritmo creado por Mukesh Prasad [1], el cual se reproduce a continuación:

Se asume que los 2 segmentos rectilíneos a considerarse están definidos por las coordenadas de sus extremos. Consideremos entonces que:

La línea $\mathbf{L_{12}}$ conecta los puntos (x_1, y_1) y (x_2, y_2).
La línea $\mathbf{L_{34}}$ conecta los puntos (x_3, y_3) y (x_4, y_4).

Hagamos que la ecuación de $\mathbf{L_{12}}$ sea $\mathbf{F(x,y) = 0}$, y que la de $\mathbf{L_{34}}$ sea $\mathbf{G(x,y) = 0}$. Entonces, el algoritmo consta de los siguientes pasos:

Paso 1 : De la ecuación de $\mathbf{L_{12}}$, se sustituyen $\mathbf{x_3}$, $\mathbf{y_3}$ por los valores de **x**, **y**, calculándose $\mathbf{r_3 = F(x_3, y_3)}$.
Paso 2 : Calcular $\mathbf{r_4 = F(x_4, y_4)}$.
Paso 3 : Si (i) $\mathbf{r_3}$ no es igual a cero, (ii) $\mathbf{r_4}$ no es igual a cero, y (iii) los signos de $\mathbf{r_3}$ y $\mathbf{r_4}$ son iguales (i.e., ambos son positivos, o ambos son negativos), las líneas no se intersectan. Terminar el algoritmo.
Paso 4 : De la ecuación de $\mathbf{L_{34}}$, calcular $\mathbf{r_1 = G(x_1, y_1)}$.
Paso 5 : Calcular $\mathbf{r_2 = G(x_2, y_2)}$.
Paso 6 : Si (i) $\mathbf{r_1}$ no es igual a cero, (ii) $\mathbf{r_2}$ no es igual a cero, y (iii) los signos de $\mathbf{r_1}$ y $\mathbf{r_2}$ son

iguales, las líneas no se intersectan. Terminar el algoritmo.
Paso 7 : Las líneas se intersectan (o son colineales). Notificar mediante un indicador.

El algoritmo utilizado para detectar una colisión con un obstáculo es, por tanto, el siguiente:

Paso 1 : Proporcionar un ángulo para mover el brazo del robot, o sus ruedas.
Paso 2 : Checar intersección entre cada parte del robot (i.e., los 3 lados del triángulo, las 2 partes del brazo y las ruedas) con cada uno de los segmentos rectilíneos que conforman un obstáculo.
Paso 3 : Si el movimiento intentado produce una colisión, entonces realizar un retroceso (*backtracking*), para hacer que el robot retorne a su posición previa.
Paso 4 : Si se proporciona un ángulo para mover los brazos, permitir su movimiento hasta el punto más cercano al obstáculo que no produzca colisión. Para determinar este punto, se subdivide el arco producido entre la posición inicial y final del robot en segmentos pequeños.
Paso 5 : Una vez determinada la nueva posición del robot (o su permanencia en la actual), se borra el gráfico, y se redibuja en la posición correspondiente.

Sujetar y Soltar un Objeto

Para poder sujetar un objeto (considerado a su vez como un polígono cerrado), tenemos que verificar primero si existe una intersección de la segunda parte del brazo del robot con cualquier lado del obstáculo (ver Figura 8). Si tal intersección no existe, esta función no tendrá efecto.

Cuando exista intersección, se establecerá un indicador y se almacenarán las pendientes del objeto de tal forma que puedan usarse para recalcular las coordenadas de los vértices del mismo dado un nuevo punto (x_5, y_5). Para tal fin, utilizamos 2 arreglos: uno contiene las diferencias horizontales (x_{i+1} - x_i) y el otro contiene las diferencias verticales (y_{i+1} - y_i). Adicionalmente, tenemos que almacenar las diferencias de (x_5, y_5) con (x_v, y_v) -coordenadas del punto donde los 2 segmentos rectilíneos se intersectan-. También, tenemos que mantener el índice correspondiente al vértice inicial del lado del obstáculo que se intersectó con el brazo.

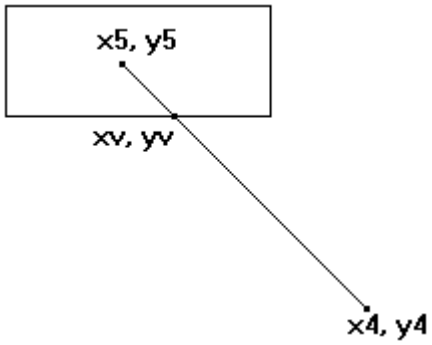

Figura 8 : Un caso en que es posible sujetar un objeto con la segunda parte del brazo del robot.

Para recalcular las coordenadas de los vértices del objeto, usamos el punto (x_5, y_5) como punto de partida. Posteriormente, utilizamos el índice almacenado previamente para reconstruir la línea con la que se intersectó el brazo. Dos ciclos más nos permiten reconstruir el obstáculo en su totalidad : uno va del *índice* al primer vértice, y el otro va del *índice+1* al último vértice.

En aquellos casos en que el brazo del robot se intersecte con más de un lado del obstáculo, se hacen chequeos adicionales para devolver la información correspondiente a sólo una de ellas.

Para soltar un objeto, simplemente se modifica un indicador global, y posteriormente cualquier movimiento del brazo del robot se hará de forma independiente al objeto.

Guías de Referencia

Puesto que la mayor parte de la información del robot se encuentra oculta al usuario, se requieren de ciertas ayudas que le permitan determinar, tras ciertos cálculos matemáticos, la posición del robot. Una de estas ayudas, son las guías de referencia, las cuales se colocan en ciertos puntos clave del espacio de movimiento del robot, y tienen como única función el devolver el ángulo que forman con respecto al punto (x_1, y_1) del cuerpo del robot, el cual puede determinarse usando la siguiente ecuación:

$$\lambda = \text{Tan}^{-1} \left(\frac{y_a - y_1}{x_a - x_1} \right)$$

($\mathbf{x_a}$, $\mathbf{y_a}$) son las coordenadas de la guía correspondiente. El valor que se proporciona al usuario se expresa en grados.

Sensores

El robot se encuentra dotado de un sensor en el punto $\mathbf{P_1}$, de coordenadas (x_1, y_1). Este sensor tiene un cierto rango el cual se define mediante un ángulo. Cada vez que se invoca el sensor, se calculará el ángulo entre cada uno de los vértices de los objetos y $\mathbf{P_1}$:

$$\text{angalf} = \text{Tan}^{-1} \left(\frac{y_i - y_1}{x_i - x_1} \right)$$

Aquí, $\mathbf{x_i}$ y $\mathbf{y_i}$ denotan un vértice del obstáculo.

El sensor regresará la distancia del punto $\mathbf{P_1}$ del robot al punto $\mathbf{P_i}$ del obstáculo, determinada mediante la ecuación:

$$\text{distancia} = \sqrt{(x_i - x_1)^2 + (y_i - y_1)^2}$$

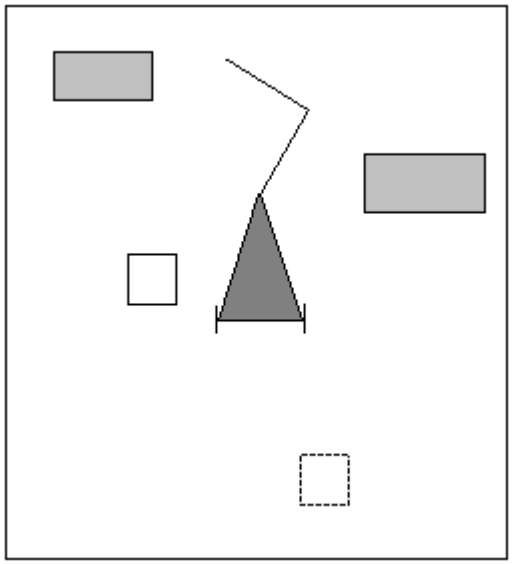

Figura 9 : Posible escenario del simulador a resolverse por el usuario. Los rectángulos rellenos representan obstáculos, y el cuadrado sin relleno es el objeto que deberá sujetarse y moverse a la posición marcada con líneas punteadas.

Interfaz con el usuario

El usuario tiene acceso sólo a un puñado de funciones del simulador, y las restantes se mantienen encapsuladas en el sistema.

Típicamente, se presenta al alumno un escenario en el que hay un problema a resolverse, como el

mostrado en la Figura 9, y en base a él, deberá elaborar un programa en Scheme que realice la tarea deseada haciendo uso de funciones como:

(**mueve-ruedas** (<val1> <val2>))
(**mueve-brazo** (<ang1> <ang2>))
(**sujeta** <obstáculo>)

Dada la poca información con que cuenta, el alumno deberá hacer uso de todo su ingenio (y sus conocimientos de matemáticas) para poder resolver el problema planteado.

Implementación

Se eligió al PC Scheme de Texas Instruments [2] como lenguaje de implementación, debido a su riqueza de funciones y fácil disponibilidad. Esta versión del popular intérprete cuenta, entre otras, con las siguientes características [9]:

- Cuenta con un editor que sigue la tradición de EMACS, y que facilita el chequeo de paréntesis balanceados.

- Tiene la capacidad de direccionar hasta 2 Mb de memoria expandida o extendida.

- Usa reglas de ámbito estático (o léxico), a diferencia del LISP que suele usar reglas de ámbito dinámico.

- Proporciona los tipos de datos que encontramos usualmente en la mayor parte de las implementaciones de LISP. Entre ellos se incluyen enteros con signo de 15 bits, enteros de precisión variable con hasta 9520 dígitos, números de coma flotante que siguen el formato IEEE, cadenas de caracteres de hasta 16K, vectores de hasta 10,921 elementos, y listas al estilo LISP.

- Soporta archivos de texto con acceso secuencial. También pueden pueden leerse archivos en modo binario, pero sólo accesando un caracter a la vez. Una seria limitación es que no se proporcionan archivos de acceso aleatorio.

- Cuenta con ventanas de texto fácilmente controlables por el lenguaje.

- Pueden invocarse archivos ejecutables .COM y .EXE que hayan sido creados en otros lenguajes, mediante llamadas al DOS.

- Soporta programación orientada a objetos mediante el SCOOPS (Scheme Object-Oriented Programming System), el cual a su vez está basado en LOOPS [10] y Flavors [11]. Este sistema de programación incluye, entre otras cosas, un sistema de herencia múltiple y dinámica.

- Aunque no documentada correctamente, hay soporte para multitareas.

- Facilidades de depuración mediante un manejador de errores y otras herramientas adicionales.

A todas estas características debe sumarse la facilidad de conseguir el intérprete de PC Scheme, pues éste se distribuye gratuitamente con la compra del texto de Texas Instruments [2]. Por todo ello, creemos que este producto no sólo es adecuado para este proyecto, sino también para otros de mayor envergadura. Lo que podría verse como un inconveniente serio para usar esta ver-

sión de Scheme, es el hecho de que no genera código ejecutable, sino código-p el cual no puede distribuirse de forma aislada, como pudiera desearse en caso de desarrollar una aplicación comercial. Sin embargo, si nos limitamos a proyectos de carácter académico, este problema puede no resultar tan grave.

Un Efecto Lateral Agradable

Aunque el simulador fue diseñado explícitamente para el área de robótica, existe el interés y la curiosidad por utilizarlo como auxiliar en la enseñanza de Inteligencia Artificial y programación funcional. Como se sabe, los ejemplos de programación que suelen usarse en un curso de LISP o Scheme suelen aburrir a alumnos de nuevo ingreso, porque los consideran inútiles. Sin embargo, si se utiliza un ambiente de programación como el proporcionado por este simulador, tal vez los alumnos muestren un mayor interés en asimilar los conceptos de un lenguaje de programación funcional tan simple y elegante como es Scheme.

Trabajo Futuro

Hay todavía muchas cosas proyectadas a futuro en torno a nuestro sistema, pues de demostrar su eficacia pretendemos elevar el grado de complejidad del robot y de los obstáculos y objetos de su entorno. Asimismo, pretendemos diseñar escenarios más complejos e interesantes.

También hay planes de realizar versiones del simulador en otros lenguajes (e.g. C y PROLOG), e incluso de usarlo en combinación con un robot auténtico.

Conclusiones

Hemos mostrado el diseño de un simulador gráfico de un robot que, pese a su sencillez, parece una buena alternativa para la enseñanza de los conceptos básicos de la robótica. Además, el hecho de haberse implementado en un lenguaje funcional lo hacen adecuado como vehículo para la enseñanza de LISP (o Scheme, según corresponda).

Creemos que los programas de este tipo motivarán más a los estudiantes para que se introduzcan a áreas que pudieran parecerles a simple vista poco atractivas, como la robótica y la programación funcional.

La presentación de escenarios con situaciones problemáticas que deberán ser resueltas por los alumnos, pretende poner a prueba sus conocimientos de programación y matemáticas, así como el desarrollo de virtudes que resultan muy deseables en los estudiantes de las carreras de computación.

Bibliografía

[1] Arvo, James (Editor). **Graphics Gems II**. Academic Press, Inc. 1991. pp. 7-8.

[2] Texas Instruments. **PC Scheme. User's Guide & Language Reference Manual**. Trade Edition. The MIT Press. 1990.

[3] Eisenberg, Michael. **Programming in Scheme**. The MIT Press. 1990. 304 p.

[4] Springer, George & Friedman, Daniel P. **Scheme and the Art of Programming**. The MIT Press. 1989. 594 p.

[5] Abelson, Harold; Sussman, Gerald Jay & Sussman, Julie. **Structure and Interpretation of Computer Programs**. The MIT Press. 1985. 542 p.

[6] Clinger, W., & Rees, J. (Editors). **The Revised³ Report on Scheme, a dialect of Lisp**. Memo 848a. MIT Artificial Intelligence Laboratory. 1986.

[7] Smith, Jerry D. **An Introduction to Scheme**. Prentice-Hall. 1988.

[8] Sussman, Gerald Jay & Steele, Jr., Guy Lewis. **Scheme : An Interpreter for Extended Lambda Calculus**. Memo 349. MIT Artificial Intelligence Laboratory. 1975.

[9] Wong, William G. **PC Scheme : A Lexical LISP**. *BYTE*. March 1987. pp. 223-6.

[10] Bobrow, D. G. and Stefik, M. J. **The LOOPS Manual**. Palo Alto, CA. Xerox Corporation. 1983.

[11] Weinreb, D.; Moon, D.; and Stallman, R. **Lisp Machine Manual**. Cambridge, MA. Massachusetts Institute of Technology. 1983.