Programación de Robots utilizando Hoja de Cálculo

[1]Sergio Alberino, [1]Fernando A. Sampieri , [1]Pablo Daniel Folino, [1,2] Juan Carlos Gómez, [1,3]Claudio Verrastro

[1]Grupo de Inteligencia Artificial y Robótica, UTN-FRBA, Buenos Aires, Argentina
[2]Instituto Nacional de Tecnología Industrial (INTI).
[3]CNEA, Centro Atómico Ezeiza

fasampieri@yahoo.com.ar; alberino@gmail.com; pfolino@hotmail.com; juanca@inti.gov.ar; cverra@cae.cnea.gov.ar

***Resumen*- El presente trabajo aborda la programación de robots con una solución del tipo *Máquina de Estados Finita* y ofrece un método para su implementación independiente de la complejidad de la tarea a realizar, utilizando directamente una planilla de cálculo, lo cual facilita su desarrollo y minimiza los errores al no tener que modificar el código por cada cambio introducido.**

Se consideran dos variantes de este método, utilizando un archivo de cabecera o *header* o con una tabla dinámica actualizada por medio de comunicación serie.

Este método es simple, utiliza pocos recursos computacionales y puede resultar especialmente útil para la programación dinámica de robots que participan en distintas competencias y para fines educativos.

I. Introducción

Los desarrollos de robótica, en particular los que poseen algún tipo de inteligencia, deben implementar mecanismos para obtener una respuesta ante estímulos o predicciones. Existen diversas técnicas de programación según la aplicación para la que se encuentra destinado el robot, y en general, aumentan su complejidad con el número de "entradas" del sistema, ya que cuentan con variedad de sensores que los relacionan con el medio en el que se desenvuelven y la tarea que deben realizar.

Una solución posible a esta problemática consiste en el uso de una Máquina de Estados Finita o *Finite State Machine* (FSM) [1] [2], que representa el problema como una sucesión finita de estados, donde el estado próximo y las salidas solo dependen de las entradas y del estado actual.

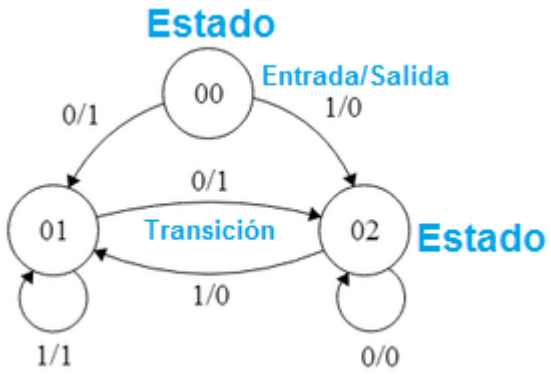

Figura N°1: Ejemplo de Diagrama de transición de Estados

Las Máquinas de Estado (MdE) son utilizadas ampliamente como rutinas para reconocimiento de secuencias de datos, para interpretación de tramas de comunicación serie, y para el ordenamiento de tareas ante distintas condiciones medidas. Son básicamente estructuras de programa cuyo comportamiento está determinado por el "estado" en el que se encuentran y por variables (generalmente Booleanas) de entrada, ofreciendo para cada caso una salida y un "estado próximo" (que puede tratarse del mismo estado en el que se encuentra).

El uso de MdE es sencillo, implica un bajo consumo de procesamiento, y resulta flexible para el agregado de nuevas condiciones y estados. Sin embargo, su implementación y mantenimiento resultan complejos ya que el comportamiento se establece en el código fuente.

Existen numerosas herramientas para la utilización de MdE, de las cuales muchas tienen como objetivo la simulación y verificación (*VERILOG* [3], *SYNOPSYS* [4], *Veracity* [5], *Algorithm State Machine (ASM)* [6]), y si bien pueden ser utilizadas como complemento, no permiten realizar programación.

Otras herramientas ofrecen una interfaz gráfica, que permite trabajar con diagramas de transición de estados (Fig.1) como puede ser *FSM-Design* [7], cuya función es la creación de un código intermedio a ser utilizado por otra aplicación de conversión, la que finalmente portará la MdE a las aplicaciones antes mencionadas.

Si bien son conocidas aplicaciones capaces de compilar una MdE, corresponden, típicamente, a sistemas propietarios o requieren del conocimiento de herramientas adicionales como ser *UML* (o un lenguaje de programación propietario) y algún formato destino de la compilación. Como ejemplos se pueden mencionar las aplicaciones *Raphsody* [8] y *Gentleware* [9]. Adicionalmente se puede utilizar un traductor [10] para generar código en C++, pero para ser utilizado con el formato *Quantum Event* [11], lo que requiere de conocimiento específico.

También se encuentran herramientas orientadas al desarrollo de hardware, y sus salidas permiten una fácil integración con componentes físicos, como por ejemplo *Sequence Invariant State Machine Compiler* [12], que está orientada al desarrollo de VLSI y no es adaptable para usos generales. Si bien la tabla de configuración es similar a la herramienta presentada en este documento, los resultados de salida están orientados a controlar cada bit por separado del registro de estado. Otra alternativa es el lenguaje *SMALL* [13], que es un lenguaje escrito y se orienta al desarrollo de hardware, realizado con compuertas.

Como referencias comerciales se puede citar el producto *Lego Mindstorm* [14], que ofrece un método grafico y didáctico para el diseño de máquinas de estado, utilizando el lenguaje *MIT Programmable Brick*, pero la salida ofrece un código para ser cargado únicamente en los módulos de hardware propios de Lego.

Es de particular interés el desarrollo de una aplicación para crear máquinas de estados en juegos llamada *BrainFrame* [15], cuyo enfoque principal esta en formulación de los comportamientos de inteligencia artificial (IA), y para esto se ofrece una interfaz gráfica cuya salida será utilizada por un modulo propio en run-time. Nuevamente es un caso de difícil integración con proyectos personales y requiere el conocimiento del manejo del programa de modelado.

El relevamiento de las herramientas basadas en Máquinas de Estados demuestra la amplitud y diversidad de sus aplicaciones, pero también la dificultad para la adaptación de las mismas a proyectos particulares dado que la mayoría corresponde a soluciones "propietarias" de gran especificidad, o que implican el conocimiento de pseudolenguajes de programación.

II. DESARROLLO

Se planteó como objetivo el desarrollo de un método que permitiera aprovechar las virtudes de las Máquinas de Estado, ofreciendo a su vez una forma sencilla para su configuración.

El sistema propuesto consta de tres partes:

- Una Máquina de Estados implementada como proceso principal en el microcontrolador del robot.
- Una planilla de cálculo donde se consignan los estados y las condiciones de entrada y salida deseadas.
- Una aplicación informática que analiza y completa la información contenida en la hoja de cálculo y la transfiere al software del robot.

La estructura que define los estados junto a sus entradas, salidas y estados futuros se guarda en la Planilla de cálculo, la Aplicación Informática convierte el archivo al formato que puede ser interpretado por la Máquina de Estados implementada en el Microcontrolador del robot.

A. Máquina de Estados (MdE)

La MdE se implementa en el Microcontrolador utilizando muy pocos recursos. Se trata de una rutina que explora los estados agrupados y, si los valores de los sensores coinciden con los previstos en la tabla, actualiza las salidas y el estado siguiente.

Los valores que necesita la MdE podrán estar definidos en un archivo de cabecera (o Header) o en una tabla dinámica definida al comienzo del programa principal.

Las características del código del motor de la Máquina de Estados dependerán del tipo de microcontrolador utilizado para cada proyecto y de su lenguaje de programación, pero deberá ejecutar una rutina como la descripta en la Fig.2.

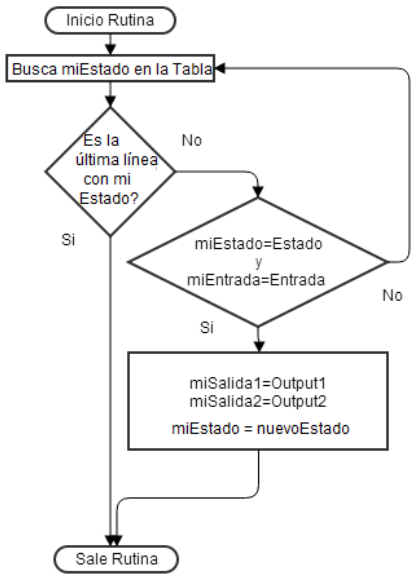

Figura N°2: Diagrama de flujo MdE

B. Hoja de Cálculo

La planilla de cálculo contiene en cada línea las distintas combinaciones de variables de entrada y de estado, y las correspondientes salidas.

Se utilizó una planilla confeccionada en *MS Excel*, de la Suite ofimática *Microsoft Office*, por tratarse de una herramienta ampliamente difundida en todo tipo de aplicaciones, sin que esto represente mayor dificultad para portarlo a otra Suite de oficina que no sea de licencia propietaria, como puede ser el programa *Calc* de *Openoffice* .

La planilla debe conservar un determinado formato para que pueda ser procesada por la aplicación informática:

La primera y segunda líneas corresponden a encabezado y a los nombres de columnas.

Las primeras columnas corresponden a la Matriz de Entrada:

- La primera columna corresponde al ESTADO de la MdE
- Las columnas siguientes (32 máx.) corresponden al valor de las distintas entradas. Podrán tomar valores: "1" para activo, "0" para inactivo, "x" para indiferente o "don´t care".

Las últimas 3 columnas corresponden a la Matriz de Salida. Su orden es fijo y contempla el siguiente formato:

- Columna Output1, corresponde al valor de la primera salida.
- Columna Output2, corresponde al valor de la segunda

salida.

- Columna Estadon+1, contiene el valor del Estado al que debe pasar la MdE.

Un ejemplo de los valores que se consignan en la Hoja de Cálculo se puede apreciar en la Fig.3.

	A	B	C	D	E	F	G	H	I	J
1	**Estado**	**Input0**	**Input1**	**Input2**	**Input3**	**Input4**	**Input5**	**Out1**	**Out2**	**E+1**
2	Estado	LLAVE	IE	II	CC	DI	DE	M1	M2	E+1
3	0	X	0	0	0	0	0	111	222	0
4	0	0	0	0	0	0	1	170	55	1
5	0	0	0	0	0	1	0	100	11	1
6	0	0	0	0	0	1	1	22	126	1
7	1	0	0	0	1	0	0	155	120	2
8	1	0	0	0	1	1	X	170	55	2
9	2	0	0	1	0	0	0	22	126	0
10	2	0	0	1	1	0	0	155	120	0
11	2	0	1	0	0	0	0	170	55	1
12	2	0	1	1	0	0	0	100	11	1
13	2	0	1	1	1	1	0	22	126	1

Figura N°3: Ejemplo de Hoja de Cálculo.

El valor "x", indiferente o "*don´t care*", permite definir máscaras de funcionamiento que simplifican la construcción de la tabla de Estados y su interpretación.

No es necesario contar con un estado para todas las combinaciones.

Si todas las líneas utilizan el mismo estado (actual y próximo) los datos indicados en la Planilla funcionan simplemente como tabla de verdad.

C. Aplicación Informática (AI)

La aplicación (Fig.4) presenta una ventana de interfaz con el usuario para la importación de los datos contenidos en la Planilla de Cálculo. Se utilizó el lenguaje Visual Basic para su programación y se presenta como archivo ejecutable. Utiliza *librerías* de acceso a los archivos de *MS Office*, para poder importar los datos de una planilla de cálculos de esa Suite.

Las tareas que realiza son:

- Importación de datos de la planilla de cálculo correspondiente.
- Verificación de rango de los valores importados (valores dentro de rango paralas columnas de Estado y Estado+1, valores "0", "1" y "X" para las entradas, 0-255 para las salidas Output1 y Output2).
- Expansión de la tabla para los Estados "X" (indiferente o "don´t Care")
- Conformación de la tabla completa.
- Ordenamiento según Estado y Entradas.

En todos los casos, el programa informa al usuario sobre la existencia de valores fuera de rango o de condiciones lógicas repetidas, deteniendo el proceso de compilación.

La aplicación permite optar entre dos formas para la transferencia de la información al microcontrolador.

- Un archivo de cabecera o Header "StateData.h" que se incluye a la programación del robot, al que accederá la rutina principal.
- Un conjunto de tramas (*frames*) enviadas por comunicación serie al microcontrolador, para la actualización de una tabla dinámica alojada en el mismo.

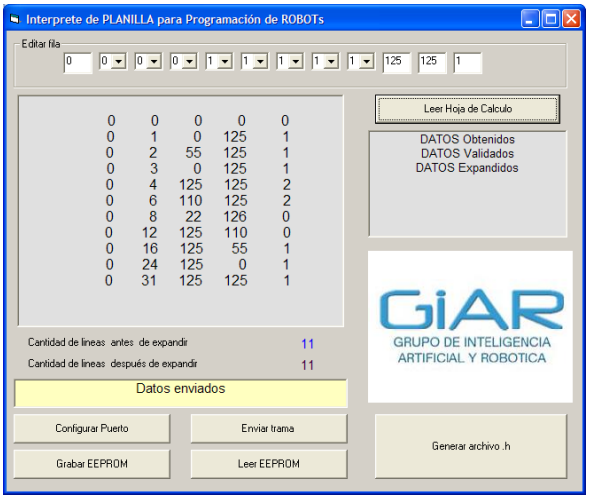

Figura N°4: Pantalla de comando de la AI

D. Funcionamiento: Método con Cabecera o Header

Una vez que la aplicación Informática compila la información contenida en la planilla de cálculo, vuelca el resultado en forma de un array de estructuras en un archivo de cabecera del lenguaje C, llamado "*StateData.h*" para ser incluido en el proyecto. Este archivo no contiene más que una traducción al lenguaje C de los diferentes estados y sus salidas. Adicionalmente una *librería* capaz de interpretar esta matriz ofrece una API al motor de la máquina de estados, para proveer la salida correspondiente a los estados particulares que se van sucediendo.

El Motor de la Máquina de Estado (MMdE) está desarrollado en una librería en lenguaje C, (archivo "*StateMachine.c*") y ofrece una función de acceso llamada *StateEngine*, que toma como parámetros la matriz de estados que contiene la especificación de cómo se comportará el sistema, y un array con los valores de las entradas. Los valores booleanos de entrada se compilan como valores binarios y se le asigna un peso potencia de 2 a cada entrada para codificarla en sistema decimal. Para obtener la información de salida se realiza una búsqueda binaria del valor codificado en decimal del estado, que garantiza una alta eficiencia aún en casos de gran cantidad de entradas.

La matriz de estados del sistema se encuentra en un archivo llamado "*StateData.h*" generado en forma automática por cada compilación de la lista de estados. Esta compuesta de entradas en forma de estructuras con los siguientes campos:

- unsigned long Conditions

En donde, “Conditions” es el valor codificado en decimal de las entradas discretizadas a 1 ó 0.

- unsigned int Estado

“Estado” es una variable de estado, en este caso utilizada como entrada del sistema, y permite lograr diferentes comportamientos para las mismas entradas en función de estados previos.

- OutputType Output

Output es la salida para cada estado, la cual esta definida por el tipo de datos OutputType, que es una nueva estructura con los siguientes campos:

unsigned int Output1
unsigned int Output2
unsigned int Estado

“Output1” y “Output2” son los valores de salida correspondientes a esa entrada, mientras que “Estado” es el valor que tomará la variable de estado, que será una entrada en el próximo ingreso al motor de búsqueda.

Un ejemplo de la información creada en el archivo “StateData.h” es:

```
#include "State.h"
#define MAX_ENTRIES 4
#define MAX_CONDITIONS 13
const MatrixEntryType StateMatrix[MAX_ENTRIES] =
 {
     { 0, 0, { 100, 0, 0 } }
  ,  { 64, 0, { 10, 0, 0 } }
  ,  { 65, 0, { 10, 0, 1 } }
  ,  { 66, 0, { 10, 0, 10 } }
};
```

Para integrar la Máquina de Estados con el proyecto se deben incluir los siguientes archivos:

StateMachine.c
State.h
StateData.h

Luego, tan solo es necesario hacer una llamada, desde el bucle principal del programa, a un *Handle* llamado *StateEngine()* cuyos parámetros son:

- *StateMatrix* es simplemente la referencia a la matriz de estados, creada durante la compilación de la máquina de estados.
- *maxMatrixEntries* es la cantidad de entradas en dicha matriz, para tal fin se genera una constante definida (MAX_ENTRIES).
- estado es la entrada, consiste en el estado previo, resultante de la ejecución anterior de la MdE.
- *conditions* es un array que representa las entradas previamente discretizadas.
- *maxConditions* corresponde al tamaño del array de condiciones.

La función devuelve una estructura con un campo por cada salida, más el campo correspondiente al estado futuro (n+1), que será una entrada en el próximo ciclo.

Esta modalidad de MMdE y archivo header con los datos transferidos de la Planilla de Cálculo se implementó en un robot de tamaño reducido para múltiples aplicaciones denominado Mini Plataforma Inteligente (MiniPI, [16]), con en microcontrolador LPC2148, basado en un núcleo ARM 7 de 32 bits (Fig. N°5)

Este robot cuenta con sensores infrarrojos y de ultrasonido con los que obtiene gran información sobre su entorno. Se lo utilizó en competencias de *minisumo* y para resolución de *laberintos*.

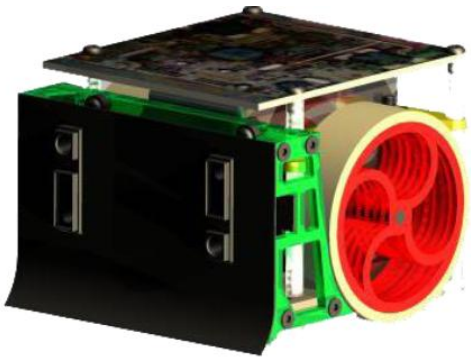

Figura N°5: Robot miniPI

Para reconocer y evitar obstáculos se confeccionó una Hoja de Cálculo considerando 12 entradas y 11 Estados posibles. Las salidas Output 1 y Output2 corresponden a Velocidad y Giro de los motores en configuración diferencial.

El uso de una MdE simplificó fuertemente la programación del microcontrolador y permitió darle diversas aplicaciones al mismo código fuente.

E. Funcionamiento: Método por comunicación Serie

Como otra opción de uso se consideró la posibilidad de contar con una tabla dinámica definida en el microcontrolador, que pudiera ser actualizada por medio de paquetes enviados por comunicación serie. El MMdE mantiene sus características, pero no es necesario conectar un programador al microcontrolador para cargar el archivo Header.

De esta manera es posible modificar el comportamiento del robot mientras el mismo se encuentra desarrollando sus tareas, y si se cuenta con una línea de comunicación inalámbrica, es posible realizar una “reprogramación” a distancia.

Esta opción es especialmente práctica cuando se experimenta con robots para los que se deben realizar ajustes ante su desenvolvimiento “en campo”.

Es necesario contar con una rutina de comunicación y en el método propuesto la tabla se envía "línea por línea" para poder llevar un control de errores a medida que se envían los datos. Una vez transferida la totalidad de la tabla se envía un comando para indicar que deben actualizarse los valores anteriores por los actuales y se define el estado que debe tomar la MdE.

La Tabla se puede actualizar continuamente mientras se realizan las pruebas, sin que se "reprograme" fisicamente el microcontrolador. Una vez que se determina un conjunto de valores que resulta óptimo para la tarea que debe realizar el robot, y a partir de un comando enviado vía serie, la tabla se almacena en memoria no volátil (E^2PROM) del micro para su uso continuo o posterior recuperación.

Se definieron para esta funcionalidad cuatro comandos de configuración:

a) Enviar nueva línea de la Tabla
b) Actualizar tabla con los nuevos valores
c) Almacenar Set de datos Actual
d) Recuperar Set de datos

Por cada comando recibido el microcontrolador devuelve una trama de reconocimiento (*Acknowledge*) si los datos son válidos o un indicador de error (*Checksum* incorrecto, *valor fuera de rango* o *comando no soportado*).

Se implementó este sistema de MdE configurada por Planilla de Cálculo, con comunicación serie, en un robot VHS "seguidor de línea", que fue construido reutilizando tecnología de otro proyecto (Fig.N°6). Se basa en un microcontrolador AVR de la familia de *Atmel*, ATmega16, sobre una placa de desarrollo propio.

Figura N°6: Robot VHS

Las entradas se obtienen de la medición realizada sobre emisores-receptores infrarrojos CNY70, colocados en la base del robot. La salida la componen dos motores de CC con reducción mecánica, colocados en una configuración "diferencial". La programación se realizó en lenguaje C para AVR.

Para esta aplicación se utilizó una tabla dinámica definida al inicio del programa. Se implementó también una rutina de comunicaciones serie para el envío de comandos desde una PC.

Para la Planilla se consideraron 6 entradas y 4 Estados posibles (Avance, Giro Izquierda, Giro Derecha, línea perdida). Las salidas Output1 y Output2 representan directamente el valor de PWM para cada motor.

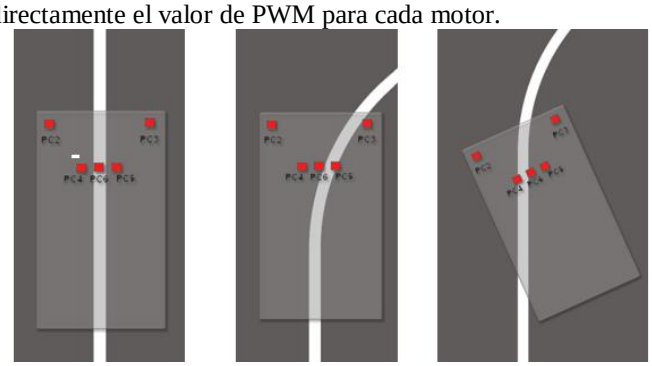

Figura N°7: plantilla de sensores

Para este caso en particular, el formato de "Tabla" para la programación resultó particularmente claro e intuitivo para la determinación de las acciones a realizar, ya que las entradas están representadas en el mismo orden en el que se encuentran físicamente distribuidas. Usando una plantilla de sensores (Fig.7) para el análisis de los distintos casos que podrían presentarse, se va completando la tabla con entradas y estados (como la de la Fig.3, donde se considera una columna para una entrada llamada *"llave"*, y los cinco sensores en orden denominados *"Izquierda Externo", "Izquierda Interno", "Centro", "Derecha Interno", "Derecha Extremo"*).

III. RESULTADOS

El método y sus variantes (Archivo de cabecera o comunicación serie) se ha probado con éxito en los dos prototipos robóticos descriptos, basados en diferentes microcontroladores y hardware.

IV. TRABAJOS A FUTURO

Como siguiente paso en el desarrollo se contempla la incorporación de una tercera salida (output3) asociada a una variable incluida en la matriz de entrada (Input0). El valor consignado en la tercera salida se utiliza como seteo de un temporizador (implementado como rutina de *timer* del microcontrolador) que modifica el valor de la entrada Input0 durante el tiempo correspondiente. La incorporación de una variable temporal asociada al sistema de Estados permitirá añadir mayores funcionalidades a este método.

Se contempla además, la posibilidad de añadir una "paginación" en memoria E^2PROM para poder almacenar y recuperar varios sets de datos diferentes (en ese caso los comandos de *almacenar* y *recuperar set de datos* deberían indicar a cual página de memoria van referidos). Así, un mismo robot podría utilizar una única máquina de estado como rutina principal y desarrollar una tarea distinta con cada tabla en memoria, seleccionada por comando serie.

Se pretende también, desarrollar de una herramienta gráfica que permita la diagramación simple de máquinas de estado y su "traducción" al formato de planilla de cálculo, como una herramienta visual para la programación en los casos en los que ésta opción resulte más sencilla.

V. CONCLUSIONES

Se presentó un método de programación de robots usando una Máquina de Estados cuya configuración es fácilmente actualizable a partir de los datos consignados en una planilla de Cálculo convencional.

Esta modalidad permite gran flexibilidad a la hora de realizar ajustes "sobre la marcha" al comportamiento del robot.

De esta manera es posible lograr distintas funcionalidades utilizando un único código principal en el microcontrolador del robot.

La actualización por comunicación serie permite realizar ajustes en las estrategias de control sin necesidad de detener el funcionamiento del robot para realizar su reprogramación.

Aunque el uso de una MdE solo resulta conveniente cuando el comportamiento de un sistema puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones, existen gran número de problemáticas de control de robots que pueden solucionarse con este método.

Si bien la implementación de la MdE no es única, ya que depende de las características propias del microcontrolador donde se ejecuta, el método propone un modelo para su desarrollo que es adecuado para aplicaciones de complejidad media-baja, y resulta de gran simplicidad, permitiendo la programación del robot por parte de usuarios que no necesitan conocimientos de pseudolenguajes, a través de la hoja de cálculo. Esto hace que resulte especialmente útil en proyectos educativos o para resolver distintas consignas en certámenes y competencias de robótica.

REFERENCIAS

[1] Kohavi Zvi y Niraj K. Jha. *Switching and Finite Automata Theory*, Cambridge University Press Third Edition, Cap. 9 y 10. P. 265 a 330 ISBN-13: 521857481.

[2] Sterling R. Whitaker, Shamanna K. Manjunath y Gary K. Maki, *Sequence-Invariant State Machines*, 1991

[3] Verifying Logic, Verilog Resources, Consultada 9/5/13 (http://www.verilog.com).

[4] Himanshu Bhatnagar, Advanced ASIC Chip Synthesis Using Synopsys *Design Compiler, Physical Compiler and PrimeTime.*

[5] A. Cant, K. A. Eastaughffe and M. A. Ozols, "A tool for Practical Reasoning about State Machine Designs." 1996. *Australian Software Engineering Conference*, Proceedings of 1996.

[6] Clare, Christopher R. 1973. "*Designing logic systems using state machines.*" McGraw-Hill, P. 16-17

[7] Briining U. , G. Radke y J. Sladky, *State-Machine-Development-Tool for High-Level-Design Entry and Simulation.*

[8] IBM, "Raphsody, Model-Driven Development Environment", Consultada 13/5/13. (www.ibm.com/software/products/us/en/ratirhap)

[9] Gentleware AG, "Poseidon for UML",Consultada 9/5/13 (http://www.gentleware.com).

[10] Romaniuk P. , Translator of hierarchical state machine from UML statechart to the event processor pattern, 2007.

[11] "Quantum Event Processor/C++, Programmer's Manual", 2005, Consultada 9/5/13 (http://www.quantum-leaps.com).

[12] Buehler D. , S. Whitaker y J. Canaris, *Sequence Invariant State Machine Compiler.*

[13] Norvell Theodore S. , SMALL: A programming language for state machine design.

[14] Lego Mindstorm, Consultada 9/5/13 (http://mindstorms.lego.com).

[15]Fu Daniel and Ryan Houlette, Stottler Henke Associates, *Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games.*

[16] Alvarez, Alejandro y otros, *MiniPI: Plataforma Inteligente de Tamaño Reducido,* XV Reunión de Trabajo en Procesamiento de la Información y Control, 16 al 20 de septiembre de 2013.