IE-01 (REFERENCIA)

Codificación de autómatas en Arduino.

Mario Alfredo Ibarra Carrillo, **Elizabeth Fonseca Chávez***

RESUMEN

El presente artículo muestra tres estrategias que facilitan la codificación de los diagramas de estados a lenguaje C, lenguaje con el cual se programa Arduino (una plataforma de hardware libre basada en el microcontrolador Atmega328P).

La primera estrategia, algo ineficiente pero simple de implementar, consiste en preguntar mediante la función IF, ¿En cuál estado se encuentra Arduino? y ¿Cuáles son las entradas? Así entonces, se generan las salidas y un nuevo estado. La segunda Estrategia se basa en tablas de transición; en este caso, un estado y un grupo de entradas hacen un índice a una fila en la tabla. En esta fila se encuentran el grupo de salidas y el nuevo estado. La tercera estrategia es mediante funciones booleanas, las cuales son generadas por métodos de minimización.

La aplicación realizada para cada estrategia es con el modelado de una línea del metro urbano. Cada tren es simulado con un único vagón (motoreductores, puente H, Arduino, modem inalámbrico y sensores ópticos) que siguen una línea. En el modelo, los vagones enfrentan problemas de colisión y retardos.

El artículo innova en el ensamble de todos los elementos de hardware contra los algoritmos de codificación presentados como estrategias en este artículo, así como da una pauta de aprendizaje a los profesores y alumnos sobre el uso de la teoría de autómatas con la plataforma Arduino. No se mencionan temas similares en la literatura arbitrada ni de manera informal.

Los seguidores de línea se han visto encasillados siempre en su papel de, precisamente, seguir una línea y sin ninguna aplicación práctica. En este trabajo se espera haber mostrado un vistazo a un universo más amplio.

A futuro, las cátedras de Diseño de Sistemas Digitales, ya podrán adecuarse mejor para cada área específica de ingeniería eléctrica gracias a la eficiencia de la plataforma Arduino para lograr resultados rápidos en armado de simuladores.

Palabras claves: Diagramas de Estado, Arduino, Xbee, Autómatas.

ABSTRACT

This paper presents three strategies that facilitate the encoding of state diagrams to C language, language that is programmed Arduino (open hardware platform based on ATmega328P microcontroller).

The first strategy, somewhat inefficient but simple to implement, is to ask using the IF function, which one state is Arduino and what are the inputs? And then the outputs and a new state are generated. The second strategy is based on transition tables, in this case, a state and a group of entries made to a row index in the table. In this row are output group and the new state. The third strategy is using boolean functions, which are generated by minimization methods.

The application is made for each strategy with modeling urban metro line. Each Train is simulated with a single wagon (motors, H Bridge, Arduino, wireless modem and optical sensors) that follow a line. In the model, the cars have collision problems and delays.

Article innovation in the assembly of all the hardware elements coding algorithms and strategies presented in this article, as well as a pattern of learning gives teachers and students on the use of automata theory with the Arduino platform. No mention of similar issues in refereed literature or informally.

Online followers have been typecast forever in his role precisely follow a line without any practical application. This work is expected to have shown a look at a broader universe.

In the future, the classes of Digital System Design may be better suited for each specific area of electrical engineering thanks to the efficiency of the Arduino platform to achieve rapid results in armed simulators.

1.0 INTRODUCCIÓN

Actualmente, se observa la utilización del sistema Arduino para muy diversas aplicaciones, la facilidad de trabajo y el bajo costo de este microcontrolador ha motivado la impartición de prácticas con alumnos de secundaria, Rodríguez [1], Web Arduino [2]. En la universidad, también se comienza a marcar una pauta, dado que el alumno fácilmente instala, configura y trabaja con esta plataforma en el desarrollo de proyectos. Por otro lado tenemos el hueco enorme en el cambio de punto de vista para la enseñanza/aprendizaje. Existen temas de Arduino para aplicaciones en Electrónica, Westerfield [3], así como sitios web para ingenieros y científicos aficionados al Arduino, Web SPECTRUM -IEEE [14]. En cuanto al tema de autómatas con Arduino no se tiene información publicada en literatura arbitrada.

Además, tenemos otro sistema llamado Xbee, WEB digi[4] que nos permite fácilmente realizar redes inalámbricas. No se tiene literatura arbitrada de xbee/Arduino aplicado con autómatas. En este articulo, nosotros aplicamos ambas tecnologías formalmente para implementarse con el tema de teoría de autómatas, y tenemos

* UNAM, Facultad de ingeniería, Telecomunicaciones y Computación , buzonliz@yahoo.com
UNAM, Facultad de ingeniería, Telecomunicaciones , maixx@yahoo.com.mx

la oportunidad de probar esta teoría con el armado del sistema, mediante ejemplos prácticos dado tres estrategias.

A continuación, se dará una breve explicación de que es Arduino, Xbee y la teoría de autómatas, como punto de partida; para que después comencemos a proponer las estrategias de prueba y así presentar los resultados.

1.1 ¿Qué es Arduino?

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador ATMEL y un entorno de desarrollo, Arduino[7], ambos diseñados para facilitar el uso de la electrónica en proyectos multidisciplinarios.

El hardware consiste en una placa impresa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8. El IDE de Arduino se denomina "Sketch", la sintaxis del lenguaje de programación Arduino es una versión simplificada de C/C++. Obsérvese en figura 1.

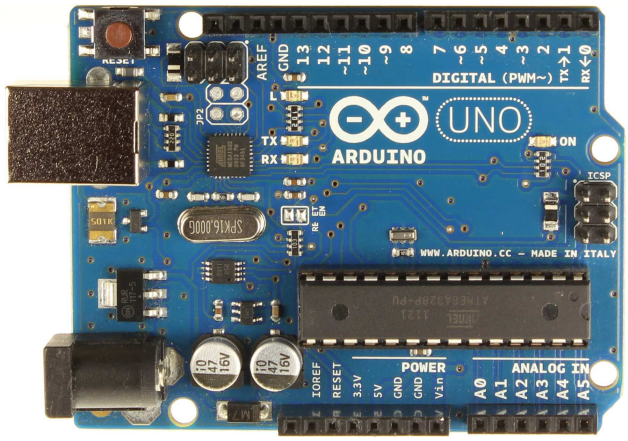

Figura 1. Visualización de Arduino UNO.

Al ser hardware libre (open-hardware), tanto su aplicación, rediseño y divulgación son libres, Arduino[7]. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

Habiendo instalado el IDE de Arduino, lo segundo que debe hacerse es conectar su sistema Arduino por medio del USB. Entonces, el sistema operativo buscará el software controlador para Arduino. Este controlador se encuentra los archivos de la distribución del Sketch. Una vez instalado el controlador se monitorea el administrador de dispositivos, que está dentro de panel de control/sistema, para detectar el puerto "com" asociado a Arduino. En el Sketch se debe palomear el puerto serie que se detecto como puerto "com".

Un programa típico consiste de dos funciones; setup() y loop(). En la función setup() se declara cuales de las agujas de la placa Arduino serán salidas y cuales serán entradas. También se configuran el convertidor análogo a digital y la velocidad de transferencia del puerto serie. En la función loop(), se escribe el proceso que se desea realizar con los datos digitales o el proceso matemático con los datos analógicos.

1.2 ¿Qué es XBEE?

Xbee es el nombre de una marca de radios digitales producida por "Digi international", Digi[4]. Los radios Xbee fueron introducidos bajo la marca MaxStream en 2005. Estos dispositivos operaban bajo es estándar 802.15.4-2033, lo que permitía una comunicación punto a punto y punto a multipunto. La tasa de bits era de hasta 250kbps.

Los módulos poseen una serie de entradas y salidas analógicas y digitales que pueden ser utilizadas en cualquier aplicación que implique transferencia inalámbrica de datos digitales a grandes tasas de transferencia de bits. La figura 2 ilustra a un Xbee serie 2 con antena de CHIP.

Figura 2. Visualización de Xbee S2.

1.3 Estándar IEEE 802.15.4 y ZigBee

Con el estándar de baja transmisión en redes inalámbricas para áreas personales (LR-WPAN)], nace el estándar que ahora se conoce como el 802.15.4[11]. El estándar IEEE 802.15.4 soporta múltiples topologías para su conexión en red, entre ellas la topología tipo estrella y la topología punto a punto (peer-to-peer).

El estándar ZigBee[8] amplía el estándar IEEE 802.15.4 aportando una capa de red (NWK) que gestiona las tareas de enrutado y de mantenimiento de los nodos de la red; y un entorno de aplicación que proporciona una subcapa de aplicación (APS) que establece una interfaz para la capa de red, y los objetos de los dispositivos tanto de ZigBee como del diseñador. Los estándares IEEE 802.15.4 y ZigBee se complementan proporcionando una pila completa de protocolos que permiten la comunicación entre dispositivos de una forma simple. Observemos las posibles redes en la figura 3: Estrella (Star), árbol (Tree) y Malla (Mesh).

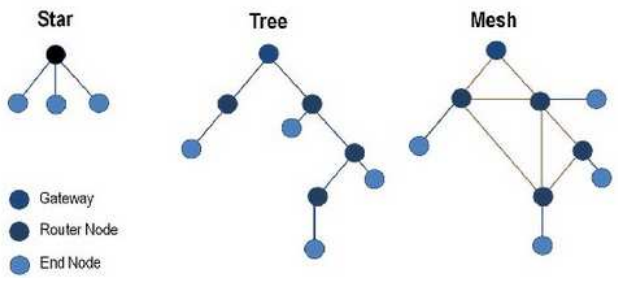

Figura 3. Tipos de redes ZigBee[11]

En la figura 4 se muestra el Xbee montado en una placa llamada "Shield Xbee", la que a su vez está montada sobre la placa de Arduino; este esquema se utilizará en el armado del sistema general.

Figura 4. Visualización de Xbee/ARDUINO.

1.4 ¿Qué es Teoría de Autómatas? Autómata de Mealy

En muchas de la bibliografías que tratan de autómatas éstos quedan definidos como entidades, las cuales, dado su estado actual y una excitación, generarán una respuesta, la que se conoce como estado siguiente, Mano[9] y Brown[10]. Para el caso que compete, los autómatas que se tratarán tendrán las siguientes dos características:

- Un conjunto finito de estados, razón por la cual se llaman finitos.
- Ante una palabra binaria de entrada, el autómata transitará a uno y sólo un estado. Razón por la cual se llaman determinísticos.

1.4.1 Definición de Autómata Finito Determinístico de Mealy:

Los autómatas que se tratarán en el presente texto tienen un comportamiento característico: comienzan en un estado que se llama "inicial", continúan por un conjunto de estados en los cuales transitan por un tiempo corto, estados de transición, y luego llegan a un conjunto de estados por los cuales transitan el resto de su existencia. A este último conjunto de estados se les conoce como finales. La ecuación 1 ilustra este concepto. Así entonces hablando con Matemáticas, una Autómata Finito Determinístico o AFD se define como el conjunto siguiente:

$$AFD=\{Q,S,G,\delta,\{q_i\},F\} \quad (1)$$

Donde:

$Q=\{q_0, q_1, \cdots q_R\}$ Es el conjunto de estados

$S=\{0,1\}$ Es el alfabeto de entrada.

$G=\{0,1\}$ Es el alfabeto de salida

$\delta: Q\times S^N \rightarrow Q\times G^M$ es la función de transición.

$\{q_i\}$ Es el estado inicial

F Es el conjunto de estados finales

Así entonces, considerando la función de transición de la ecuación (1), dado el estado actual y una palabra de entrada de bits (es el número de entradas digitales), se transita a otro estado generando una palabra de bits (es el número de salidas digitales). Tómese en cuenta que las entradas digitales están conectadas a sensores en tanto que las salidas digitales están conectadas a actuadores.

1.4.2 Elementos del diagrama de estados para el autómata de Mealy

El diagrama de estados será la representación gráfica de la función de transición. Los elementos necesarios para su construcción se listan a continuación y pueden verse en la figura 5.

1. Los estados se representan con circunferencias etiquetadas con el nombre del estado o bien, con su código numérico.

- El estado inicial tiene una flecha que le apunta y viene de la nada

- La transición a otro estado se representa con flechas. La flecha tiene una doble etiqueta:
  - La palabra de entrada de N bits: $s_{N-1}\cdots,s_1,s_0$ formada a partir de alfabeto de entrada.
  - La palabra de salida de M bits $g_{M-1}\cdots,g_1,g_0$ formada a partir de alfabeto de salida.
- Los estados finales se representan con doble circunferencia

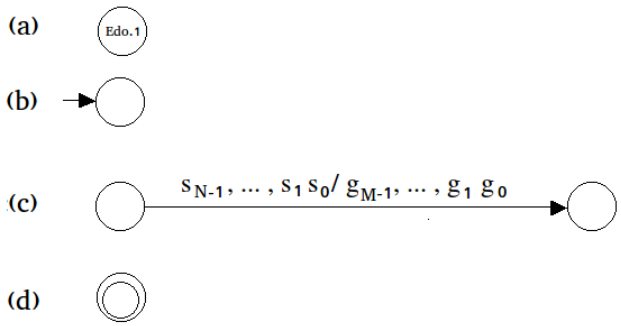

Figura 5. Elementos de un diagrama de estado.

Ahora después de recordar o conocer lo mínimo básico de Arduino, Xbee y teoría de autómatas, seguimos con el ensamble de hardware y software.

2 ARMADO DEL HARDWARE.

Problema general: Se pretende el modelado de una línea del metro urbano. En el modelo, los vagones enfrentan problemas de colisión y retardos en las estaciones por el intercambio de pasajeros. Así entonces se requiere de lo siguientes elementos:

- **Arduino Rover**: Los trenes son simulados con vehículos de un sólo vagón ("Arduino Rover": motoreductores, puente H, Arduino, xbee y sensores ópticos).
- **Las vías** son modeladas con un circuito de cinta negra.
- **Sistema de información**. La central que controla el sistema es un Arduino conectado a los vagones mediante una red Xbee.

2.1 El "Arduino Rover"

La figura 6 muestra una vista superior del "Arduino Rover". En esta vista pueden identificarse los siguientes subsistemas:

- Subsistema de potencia
- Subsistema de sensores
- Subsistema de actuadores
- Subsistema de inteligencia: Arduino UNO
- Subsistema de comunicación

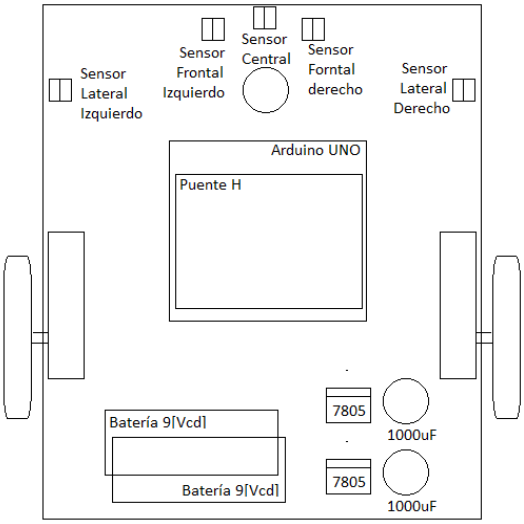

Figura 6. Prototipo del "Arduino Rover": Esquema.

Se comenzará comentando el subsistema de inteligencia. Éste se encuentra formado en su totalidad por el Arduino UNO, el cual tomará decisiones en función de las entradas. La figura 7 ilustra un Arduino UNO cuyas terminales de contacto se han modificado y que se llamarán "agujas".

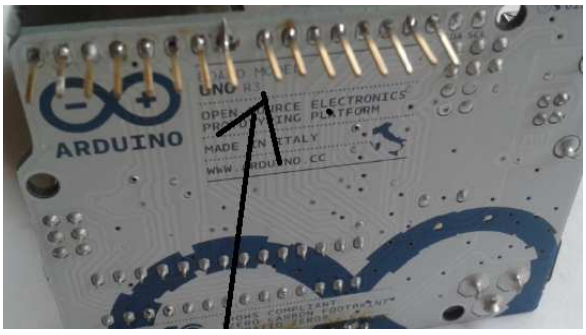

Figura 7. Agujas de Arduino

El subsistema de potencia consiste de dos baterías de nueve voltios regulados a cinco. Una de las baterías alimenta al Arduino UNO y la segunda batería alimenta al subsistema de actuadores.

El subsistema de sensores es un conjunto de cinco parejas emisor-sensor del tipo infrarrojo. La figura 8 resalta las posiciones de los sensores. Los sensores se conectan a los terminales del Arduino UNO como sigue:

- Aguja 12: Sensor lateral derecho
- Aguja 11: Sensor frontal derecho
- Aguja 10: Sensor frontal central (para detectar colisiones con obstáculos frontales)
- Aguja 9: Sensor frontal izquierdo
- Aguja 8: Sensor lateral izquierdo

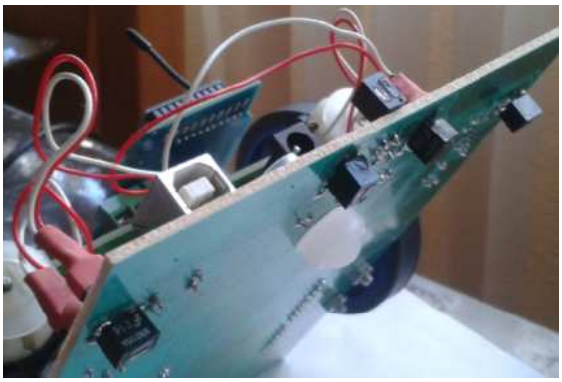

Figura 8. Parejas emisor-sensor en el carrito "Rover".

Los sensores están alambrados de forma que al detectar una superficie blanca devolverán un "uno" lógico. En tanto, que al detectar una superficie negra o vacía, devolverán un "cero" lógico.

El subsistema de actuadores consta de dos motores de CD los cuales son controlados por el Arduino UNO, mediante dos puentes H. La forma en que se conectan los actuadores a los pines (o agujas) del Arduino UNO es la siguiente:

- aguja 4: Terminal 1 del motor izquierdo
- aguja 5: terminal 2 del motor izquierdo
- aguja 6: Terminal 1 del motor derecho
- aguja 7: terminal 2 del motor derecho

Finalmente, el subsistema de comunicación lo integra el "Arduino Xbee Shield junto con su Xbee S2 con antena de alambre. La figura 9 ilustra al "Arduino Rover" completo.

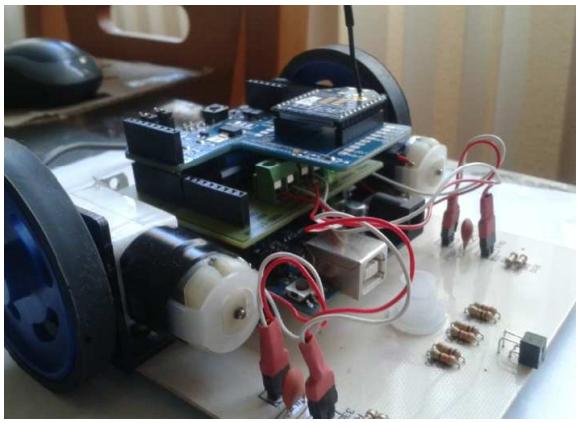

Figura 9. Prototipo Carrito Arduino Rover, puente H, Arduino y Xbee conectado a los motoreductores.

2.2 Construcción de la vía

Se debe armar una pista que representara nuestra ruta a seguir, esta se hizo con una superficie blanca y una línea negra formando un circuito, figura 10. Nótese que unas cintas transversales indican las estaciones de intercambio de pasajeros.

Figura 10. Pista

2.3 Protocolo de configuración de redes ZIGBEE. Digi Documentacion[5]

Cada Xbee viene con un "Número de Serie". De forma habitual, se separan en dos partes: **Alta (SH) y Baja (SL)**. Por ejemplo:

NS(Numero de serie): 13A200 4081B766
SL: 4081B766
SH: 13A200

Para enviar datos a otro XBEE se llena los datos de "Destino" que también se separa en parte baja y parte alta.

Requerimientos Hardware: Arduino y Xbee Shield
Requerimientos software: IDE Arduino y X-CTU (software de digi, sólo funciona en Windows)

1. Conectar el sistema completo: ARDUINO/XBEE SHIELD
2. Verificar el número de puerto que se detectó.
3. **Utilizar el Arduino sólo como interfase USB**, **para configurar los XBEE.** Para esto debemos **cargar** un programa con las funciones vacías: void setup() {} void loop() {}.
4. **Configurar XBEE. Deseamos un Coordinador y un Ruter**. Tipo estrella. Enviar/Recibir el coordinador y ruters a todos (difusión). Abrir software X-CTU, ver figura 11.

    a. **Oprimir botón: Test/Query**. Pestaña PC Settings. Información obtenida: número de serie de Xbee y tipo de modem detectado.
    b. **Oprimir botón: Read.** Pestaña Modem Configuración. Información a modificar: escoger en "Funtion Set" que un Xbee es tipo Coordinador y el otro sea tipo Ruter o usuario Final tipo AT.
    c. Para que sea **difusión (uno a todos)** se coloca en los valores de destino LOW=FFFF (DL) y HIGH= 0(DH)
    d. Palomear la casilla:"Always Update Firmware", siempre solo al inicio.
    e. Escribir en Xbee esta configuración, oprimir botón "WRITE".

Problemas: Si al momento de oprimir READ existe un problema, se debe pasar a pestaña "Modem Configuration" y escoger el modem dado primero, luego Funcion Set y grabar esta configuración sin actualizar firmware. Oprimir WRITE. Después volver a leer (Read).

3. ESTRATEGIA 1: CODIFICACIÓN DE AUTÓMATAS EN LENGUAJE C MEDIANTE FUNCIONES "IF"

3.1 Como se codifica

Una forma simple, aunque no muy eficiente de codificar un diagrama de estados en lenguaje C es usando funciones "if" en las cuales se pregunta por el estado actual y las entradas. Por cada evento que obliga al autómata a transitar se pregunta por todos los posibles estados y todas las posibles entradas. La figura 12 ilustra el formato general del código.

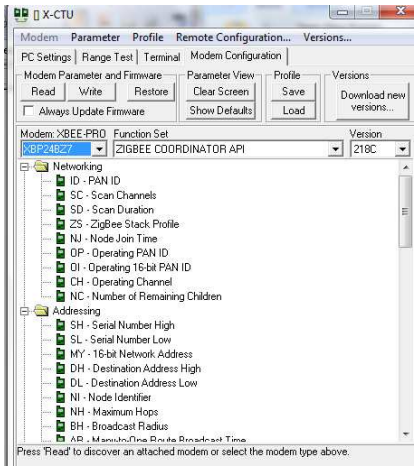

Figura 11. Visualización de X-CTU

Estando el autómata en un estado dado, no importará el valor de una o varias entradas. Tal situación implica que esta entrada digital no se tomará en cuenta en la lista de condiciones para transitar a otro estado. Una entrada no importa se denota escribiendo una "X" en la tabla de transiciones, véase Mano[9].

3.2 Conclusiones

Al respecto del comportamiento del código, pueden notarse dos aspectos:

1. El cálculo de la salida y del nuevo estado tomará mucho más tiempo en aquellas funciones "if" que están al final de la codificación. Esto provoca que el Arduino Rover tenga un comportamiento cuando viaja en un sentido sobre la pista y tenga otro comportamiento cuando viaja en sentido contrario.
2. El tamaño del código es considerable.

4. ESTRATEGIA 2: CODIFICACIÓN DE LA TABLA DE TRANSICIONES EN LENGUAJE C

Una tabla de transiciones es la representación tabular del Autómata Finito, a este respeto es necesario imponer algunas restricciones en cuanto a la elaboración del diagrama de estados:

1. Los estados se codifican numéricamente desde cero.
2. No hay saltos en la numeración de los estados.

Hecho ya el diagrama de estados, con sus restricciones antes mencionadas, lo que sigue es generar la tabla de transiciones tal como se muestra en la tabla 1.

4.1 La tabla de transiciones

En la tabla 1, bajo el símbolo Q del estado actual, puede notarse la etiqueta $q_{MS} \cdots q_0$ que se refiere a la codificación binaria del estado, desde el bit menos significativo q_0 al bit más significativo q_{MS} . Así también, bajo el símbolo S , la etiqueta $s_{N-1} \cdots s_0$ representa la palabra de entrada con una codificación de N bits. Finalmente, bajo el símbolo G , la etiqueta $g_{M-1} \cdots g_0$ representa la palabra binaria de salida con una la codificación de M bits.

```
#define edo0 <valor numérico>
#define edo1 <valor numérico>
...

 byte Q = <estado inicial>;

//Variables que representan las entradas digitales
 byte sN_1 ... ,s1, s0 ;

void loop ()
{
  s0 = digitalRead(pin?);
  sN_1=digitalRead(pin?);

  if  (sN_1= 0|1) && ( ... ) && (s0 = 0|1) && (Q = <estado>)
  {
     //Salidas de transición al siguiente estado
     digitalWrite(pin? , 0|1);
     digitalWrite(pin? , 0|1);

     //Estado siguiente
     Q = <estado>

    //Retardo
    delay(<tiempo>);
  }

 else   if (sN_1= 0|1) && ( ... ) && (s0 = 0|1) && (Q = <estado>)
 {
     //Salidas
    digitalWrite(pin? , 0|1);
    digitalWrite(pin? , 0|1);

    //Estado siguiente
    Q = <estado siguiente>

    //Retardo
     delay(<tiempo>);

  }
   else if    ...
}
```

Figura 12. Fracción de código. Estrategia 1

Tabla 1. Fracción de código. Estrategia

Edo. actual Q		Entrada S		Edo. Sig. Q_{Nuevo}		Salidas G	
q_{MS}	$\ldots q_0$	s_{N-1}	$\ldots s_0$	q_{MS}	$\ldots q_0$	g_{M-1}	$\ldots g_0$
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				

4.2 Llenado de la tabla de transiciones

Para llenar la tabla la tabla de transiciones se deben considerar las siguientes restricciones:

1. Las columnas “Estado actual” y “Entrada” forman, las dos juntas, un código numérico llamado el “índice” de la tabla.
2. Los índices deben estar en orden. Esto puede verse en la misma tabla 1.
3. El índice máximo de la tabla queda limitado por el máximo estado.

Consideradas las restricciones se procede al llenado de la tabla, estado por estado, entrada por entrada.

4.3 Codificación de la tabla de transiciones

Para codificar la tabla de transiciones se debe considerar lo siguiente:

- Las columnas “estado actual” y “entrada” son el índice de la tabla. Así que solamente se define una variable del tipo entero de 8 bits: nótese que este tipo de dato limita el número de filas de la tabla a 256 elementos.
  byte QS;

- La columna de ” estado siguiente” es un arreglo con los códigos numéricos de los estados siguientes.
  byte Q [] = { ... };

- La columna “salidas” es un arreglo con los códigos numéricos de las salidas.
  byte G [] = { ... } ;

El cálculo del índice QS se genera a partir del nuevo estado Q_{nuevo} y de la palabra de entrada $s_{N-1}\cdots s_0$. Esto es, se copia el nuevo estado en QS y los bits de las señales de entrada se van entrando por la derecha de QS . Esto se ilustra a continuación

```
QS=Q[Qs];
QS=(QS<<1) | digitalRead( );
...
QS=(QS<<1) | digitalRead( );
```

La palabra de entrada quedó definida como $s_{N-1}\cdots, s_1, s_0$. Cada símbolo tienen una asociación con un terminal del Arduino, esta asociación puede definirse a través de constantes simbólicas como sigue:

```
#define s0 2        //Se asocia el pin 2 con s0
#define s1 3        //Se asocia el pin 3 con s1
. . .
```

La palabra de salida quedó definida como. Cada símbolo tienen una asociación con un terminal del Arduino, esta asociación puede definirse a través de constantes simbólicas como sigue:

```
#define g0 4        //Se asocia el pin 4 con g0
#define g1 5        //Se asocia el pin 5 con g1
. . .
```

Finalmente, la figura 13 ilustra el formato para codificar una tabla de transiciones.

4.4 Conclusión

Al respecto del comportamiento del código, pueden notarse dos aspectos:

1. El cálculo de la salida y del nuevo estado tomará el mismo tiempo para cualquier situación de entrada. Esto hace que el Arduino Rover tenga el mismo comportamiento para cualquier sentido que siga en la pista.
2. El tamaño del código de la figura 13 tiene menos líneas que el código de la figura 12.

```
//Asociación de la palabra de entrada con los terminales de Arduino
#define s0 2   //Se asocia el pin 2 con s0
#define s1 3   //Se asocia el pin 3 con s1
...

//Asociación de la palabra de salida con los terminales de Arduino
#define g0 4   //Se asocia el pin 4 con g0
#define g1 5   //Se asocia el pin 5 con g1
...
#define g<M-1> 7  //Se asocia el pin 5 con g1

//Variables globales para la tabla de transiciones
byte QS=<estado inicial>;
byte wG; //palabra de salida
byte Q[ ] = { ... } ;
byte G[ ] = { ... } ;

void setup()
{
        //ENTRADAS
        pinMode(s0,INPUT);
        pinMode(s1,INPUT);
        ...

        //SALIDAS
        pinMode(g0,OUTPUT);
        pinMode(g1,OUTPUT);
        ...
}

void loop()
{
  //Formando el índice mediante las entradas
  QS=(QS <<1) | digitalRead ( SN-1);
   ...
   QS=(QS <<1) | digitalRead ( S0);

   //Evaluando las salidas
   wG = G [QS];
   digitalWrite ( g0 , 1 & wG);
   digitalWrite ( g1 , 2 & wG);
   digitalWrite ( g2 , 4 & wG);
   ...
   digitalWrite ( g<M-1> , 2^M & wG );

  //Formando el índice a partir del estado siguiente
  QS = Q[QS];
}
```

Figura 13. Formato para codificar la tabla de transiciones.

5. ESTRATEGIA 3: CODIFICACIÓN DE UNA AFD MEDIANTE ECUACIONES LÓGICAS

Una forma simple de programar un AFD en lenguaje estructurado es mediante el uso de ecuaciones lógicas. Las ecuaciones lógicas son la forma en la cual los diseñadores de sistemas digitales implementan circuitos lógicos y para el caso representan una solución intermedia entre la codificación de la tabla y el uso de funciones "IF". Las ecuaciones lógicas requieren más tiempo de cálculo que la tabla de transiciones pero, para cualquier señal de entrada, el tiempo de cálculo de la salida y del nuevo estado es constante.

5.1 La tabla de transiciones

Hecho ya el diagrama de estados, con sus restricciones antes mencionadas, lo que sigue es generar la tabla de transiciones tal como se muestra en la tabla 1. Esta tabla corresponde bien con el diseño de circuitos secuenciales utilizando flip-flops tipo D.

5.2 La codificación de la tabla en fórmulas lógicas

Para codificar la tabla de transiciones se debe considerar que las salidas y el estado siguiente es un conjunto de funciones de las entradas y del estado actual, es decir, las salidas se calculan como:

$$g_0 = f_0(\ldots, q_1, q_0, \ldots, s_1, s_0)$$
$$g_1 = f_1(\ldots, q_1, q_0, \ldots, s_1, s_0) \quad (1)$$

Y el nuevo estado se calcula como:

$$q_0 = h_0(\ldots, q_1, q_0, \ldots, s_1, s_0)$$
$$q_1 = h_1(\ldots, q_1, q_0, \ldots, s_1, s_0) \quad (5,2)$$

Las fórmulas lógicas suelen pasar por un proceso de minimización en el cual se reduce el número de operaciones lógicas y también se reduce el número de términos. Estos procesos se conocen como "mapas de karnaugh".

5.3 La codificación del autómata

El esquema general de codificación por medio de ecuaciones lógicas se muestra en la figura 14. Al respecto debe observarse que hace falta definir constantes que de nombre a las agujas del Arduino UNO.

```
//VARIABLES PARA LA TABLA DE TRANSICIONES
        //El estado
        byte q0,q1,...;

        //Las entradas
        byte s0,s1,...;

        //Las salidas
        byte g0,g1,...;

void setup() {
 //ENTRADAS
 pinMode( <aguja> , INPUT);
 pinMode( <aguja> , INPUT);

 //SALIDAS
 pinMode( <aguja> , OUTPUT);
 pinMode( <aguja> , OUTPUT);
 pinMode( <aguja> , OUTPUT);
 pinMode( <aguja> , OUTPUT);
}

void loop() {
        //EL ESTADO INICIAL
        q0=...;
        q1=...;
        ...

        //LAS ENTRADAS
        s0 = digitalRead( <aguja> );
        s1 = digitalRead( <aguja> );
        ...

        //LAS SALIDAS
        g0 = f0 ( ...q0,q1, ...s1,s0);
        g1 = f1 ( ...q0,q1, ...s1,s0);
        ...

        //EL NUEVO ESTADO
        q0 = h0 ( ...q0,q1, ...s1,s0);
        q1 = h1 ( ...q0,q1, ...s1,s0);
        ...

        //ENVIANDO LAS SEÑALES DE SALIDA
        digitalWrite ( <aguja> ,g0);
        digitalWrite ( <aguja>,g1);
        ...

}
```

Figura 14. Formato para codificar la tabla de transiciones mediante ecuaciones lógicas.

5.4 Conclusiones de estrategia

1. El código ocupa menos líneas que el de la primera estrategia.
2. No se requiere memoria RAM para almacenar la tabla de transiciones.
3. Es mucho más rápido que usar funciones IF y no tan rápido como direccionar líneas en una tabla de transiciones.

6. SIMULADOR METRO.

Para este proyecto, se usará la plataforma del "Arduino Rover", mencionada en una sección anterior. Una vez que se conoce el funcionamiento del "Arduino Rover" se procede al diseño del autómata y posteriormente a la codificación del autómata mediante saltos en la tabla de transiciones.

Se elije la segunda estrategia de codificación debido a que es posible codificar la llamada a funciones en cada transición.

6.1 El enunciado del problema. Simulador de metro

Como ya se mencionó en una sección anterior, se pretende un simulador del sistema de transporte ferroviario urbano (metro). Este simulador consta de los siguientes elementos:

- Trenes: simulados con seguidores de línea llamados "Arduino Rover.
- La vía férrea: simulada con un circuito de cinta negra sobre fondo blanco.
- Central de control: simulada con un Arduino que concede permisos de marcha a cada tren.
- Un sistema de comunicación: simulado con una red inalámbrica cuyos nodos son módulos Xbee.

6.2 Configurar la red inalámbrica y probar.

Los dispositivos Xbee, se configuraron para trabajar en modo "difusión", lo que permite conformar una red punto-multipunto: ya comentado en párrafos anteriores.

Así entonces, la red de comunicación queda como sigue:

- Un Xbee se encarga de conformar la red, según lo indica el protocolo 8025.15.4.
- Tres Xbee, montados sobre los Arduino rover.
- Un Xbee hace las veces de estación central.

6.3 La estación central

La estación central, es decir, aquella que concede los permisos de marcha a los trenes se simulará con un Arduino UNO. La figura 15 muestra la conformación de la estación central:

- Arduino UNO: es el cerebro de la central y se encarga de conceder los permisos de marcha.
- Una tarjeta Isis servirá como indicador del estado de cada tren. Cada tren solo tiene dos estados: marcha o parada.
- Un Xbee usado para comunicación remota con los trenes.

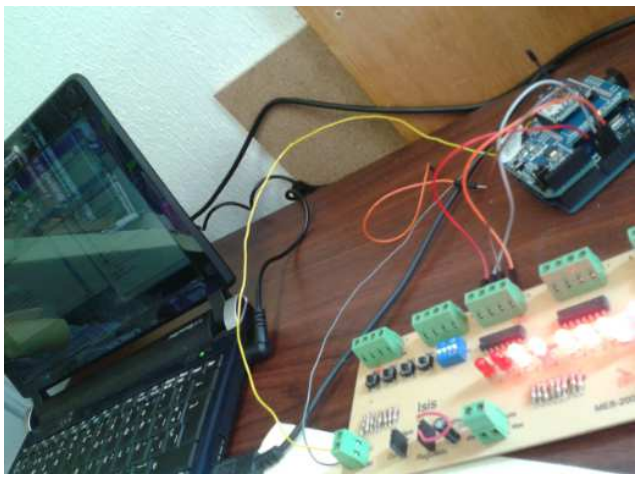

Figura 15. Sistema de prueba de red inalámbrica. La computadora es para cargar el código de la central.

6.4 Los trenes

Como ya se mencionó, cada tren es simulado con un Arduino Rover, el cual está conformado de un chasís que soporta dos motores de CD, el Arduino UNO, un puente H y un Xbee. La programación del cerebro del Arduino Rover se explicará en dos partes. La primera parte consiste solamente en el algoritmo seguidor de línea y la parte dos consiste del diagrama de estados que considera el arribo a la estación y la detección de obstáculos en la vía.

6.5 Comportamiento de seguidor de línea

Como se indicó en el párrafo anterior, por cuestiones didácticas, primero se describe la parte del comportamiento del tren referida al seguidor de línea. La figura 16.a ilustra las suposiciones que hechas respecto a la interacción entre la vía y los sensores frontales del tren.

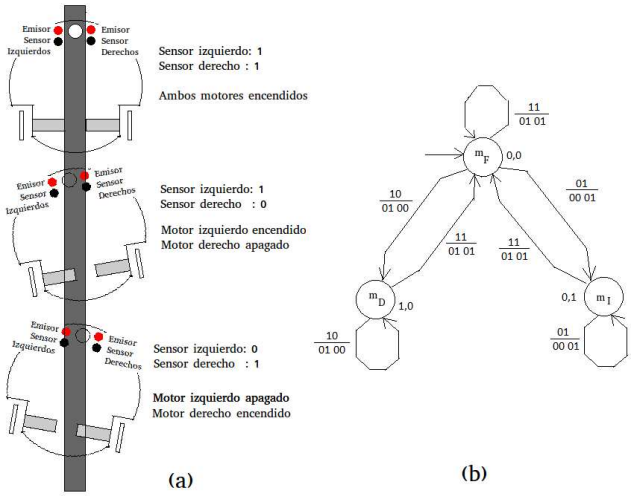

Figura 16. (a) Suposiciones para la generación del diagrama de estados. (b) Diagrama de estados: mF: movimiento al frente, mI: Movimiento a la izquierda, MD: movimiento a la derecha.

La asignación de nomenclatura a las agujas del Arduino UNO que llevarán las señales de entrada provistas por los sensores así como la asignación de nomenclatura a las agujas del Arduino UNO que llevarán las señales de salida es:

- La entrada del autómata es una palabra de dos bits formada por las señales de dos de los sensores frontales: $[s_{FI}, s_{FD}]$, donde:
  - s_{FI} Es la aguja del Arduino UNO conectada al sensor frontal izquierdo.
  - s_{FD} Es la aguja del Arduino UNO conectada al sensor frontal derecho
- La salida es una palabra de cuatro bits formada por las señales que activan a los motores: $[g_{MIA}, g_{MIF}, g_{MDA}, g_{MDF}]$ donde:
  - Motor derecho
    - g_{MDF} ES la aguja asociada al terminal del motor derecho que provoca su avance hacia el frente.
    - g_{MDA} ES la aguja asociada al terminal del motor derecho que provoca su avance hacia atrás.
  - Motor izquierdo
    - g_{MIF} ES la aguja asociada al terminal del motor izquierdo que provoca su avance hacia el frente.
    - g_{MIA} ES la aguja asociada al terminal del motor derecho que provoca su avance hacia atrás.

Considérese que para que el motor derecho avance, las señales lógicas en las respectivas agujas deben ser: $(g_{MDA}, g_{MDF})=(0,1)$ Una situación semejante ocurre con el motor izquierdo.

El diagrama de estados del autómata que controlará al "Arduino Rover", mostrado en la figura 16.b se construye a partir de las suposiciones mostradas en la figura 16a. El significado de los nombres de los estados es:

- m_F : movimiento al frente: ambos motores activados.
- m_I : movimiento a la izquierda: motor izquierdo apagado y motor derecho activo
- m_D : movimiento a la derecha: motor izquierdo activo y motor derecho apagado

La figura 17 muestra al seguidor de línea haciendo lo suyo en la vía aún sin estaciones.

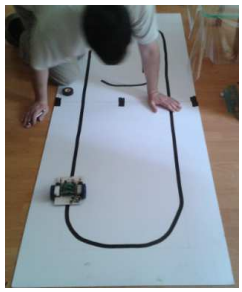

Figura 17. Armado de sistema ferroviario Metro

6.6 Comportamiento final del tren.

Los procesos generales involucrados en el control de los trenes son:

1. El tren llega a la estación y se detiene durante un tiempo x.
2. El tren hace una "Solicitud de marcha" a la central
3. La central le da permiso de partir.
4. Hay un obstáculo en el camino y el tren se detiene. El obstáculo es un tren en la estación.
5. Cuando el obstáculo se retira, el tren hace la "Solicitud de marcha" a la central.
6. La central le da permiso de partir.

El diagrama de estados correspondiente a este comportamiento se muestra en la figura 18.

6.7 El sistema en acción

La figura 19 es la toma del sistema de trenes, todos en rodamiento sobre la vía. Las estaciones de intercambio de pasajeros están marcas al lado de la vía: se han marcado cuatro estaciones. La figura 20 muestra algo recurrente en todo sistema de transporte embebido: el congestionamiento en un segmento de la vía. Finalmente observamos en la figura 21 otro punto de vista de este sistema férreo, más cercano.

Figura 19. Trenes en marcha sobre la vía.

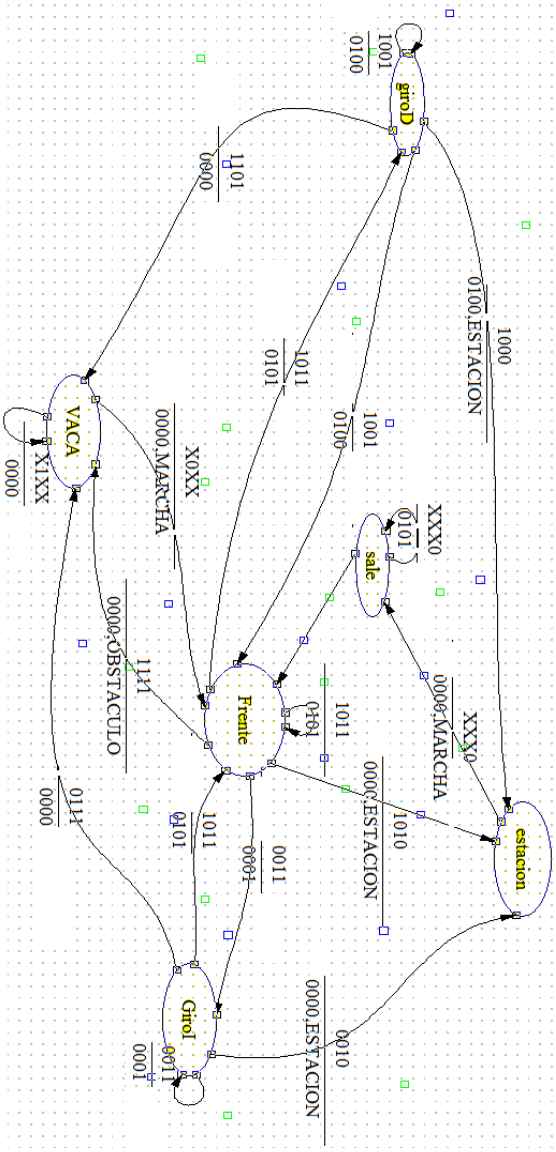

Figura 18. Diagrama de estado. Parte: Reporta estado y Solicitud de Marcha

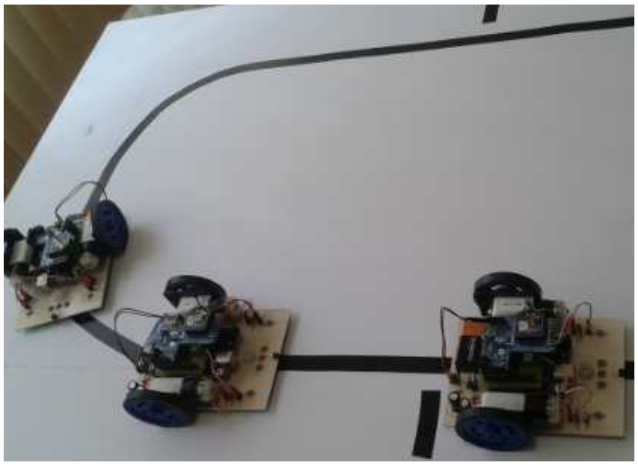

Figura 20. Conflicto de congestionamiento.

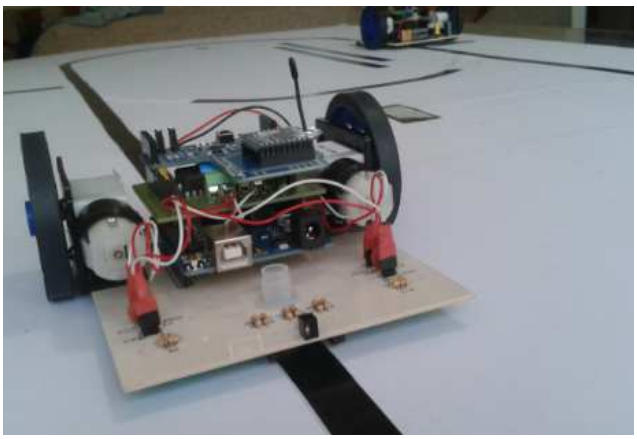

Figura 21. Acercamiento de Arduino Rover.

Conclusiones generales:

Los seguidores de línea se han visto encasillados siempre en su papel de, precisamente, seguir una línea y sin ninguna aplicación práctica. En este trabajo se espera haber mostrado un vistazo a un universo más amplio.

La teoría de autómatas ofrece un proceso intelectual fluido en cuanto a la solución de problemas; esto en comparación con el uso de diagramas de flujo. Desafortunadamente, los diagramas de estados tienden a dificultar la codificación a lenguaje de alto nivel. Por tanto se espera que las estrategias de codificación mostradas permitan la utilización más frecuente de los diagramas de estado.

Trabajar con Arduino fue simple, ahora la parte que nos ocupó tiempo y dinero, fue preparar el simulador hardware, organizar ideas para codificar el diagrama de estado con un resultado exitoso. La sensación visual de ver el resultado en forma tridimensional y con movimientos programados es muy motivadora.

Xbee proporciona una serie de protocolos de comunicación que no garantiza al cien por ciento las transferencias exitosas de datos. Durante los experimentos se notó que las órdenes de marcha enviadas por la central no llegaban completas a los Arduino Rover por lo que ocurrían trenes muertos en la vía. Esto se debe al hecho de trabajar en la banda de 2.4GHz, una banda congestionada por los sistemas de redes inalámbricas para computadoras.

Por otro lado, comenzar a trabajar con los Xbee resultó muy desgastante; entender los modos de operación y comportamientos de los protocolos requirió de un gran esfuerzo ya que se debían cambiar configuraciones quitando y recolocando puentes en las tarjetas. El costo también fue del tipo económico ya que los dispositivos Xbee son muy sensibles a errores de configuración, dando como consecuencia la inversión en más dispositivos.

Xbee es un mundo de protocolos y modos de operación. Lo que en el presente artículo se muestra es apenas un vistazo corto, más aún si se considera que no son los únicos módems digitales en el mercado.

Finalmente se espera que al abrir esta puerta de pruebas de hardware sea una motivación para el alumnado en este siglo 21.

8. BIBLIOGRAFÍA

[1] Marco Antonio Rodríguez Fernández, Computación Física en Secundaria, Primera Edición Editor Marf Boooks, España.

[2] WEB ARDUINO http://playground.arduino.cc/Es/Guias/

[3] Mike Westerfield. Iphone & iPad electronics Projects, editorial: O'Reilly Media.

[4] Web Digi de xbee: http://www.digi.com/

[5] Digi. Documentación, 2009: ftp://ftp1.digi.com/support/documentation/90000976_C.pdf

[6] Texas instruments, launchpad Stellaris: http://www.ti.com/ww/en/launchpad/tiva_c_head.html?DCMP=tivac-series&HQS=tivac-launchpad-b

[7] Web Arduino: http://www.arduino.cc/

[8] Zigbee http://www.zigbee.org

[9]Morris Mano. Diseño digitales. Pearson Educación, 2003, tema: diagramas de estados. Tercera edición. (Cualquier edición).

[10]Stephen Brown, Zvonko Vranesic, Fundamentos de lógica digital con diseño VHDL, segunda edición. McGraw Hill. De pág. 481 a pág. 574.

[11] Estándar 802.15.4 http://www.ieee802.org/15/pub/TG4.html

[12] manual DIGI XBEE ftp://ftp1.digi.com/support/documentation/90000976_C.pdf

[13] http://arduino.cc/en/Serial/Print

[14] http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

9. INFORMACIÓN ACADÉMICA

Elizabeth Fonseca Chávez. Ingeniera Mecánica Electricista egresada de la Facultad de Estudios Superiores Cuautitlán de la UNAM, Maestra en Ingeniería orientación Sistemas egresada de la División de Estudios de posgrado de la Facultad de Ingeniería de la UNAM. Actualmente profesora de asignatura del departamento de Computación y del departamento en Telecomunicaciones de la Facultad de Ingeniería de la UNAM.

Mario Alfredo Ibarra Carrillo, Ingeniero Mecánico Electricista egresado de la Facultad de Ingeniería UNAM, Maestro en Ingeniería orientación en procesamiento de señales egresado de la División de Estudios de posgrado de la Facultad de Ingeniería de la UNAM. Profesora asociado del departamento de Telecomunicaciones de la Facultad de Ingeniera UNAM.