ESCUELA SUPERIOR POLITECNICA DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION

" CONTROL POR COMPUTADORA DE UN VEHICULO PROTOTIPO A TRAVES DE SEÑALES DE RADIO "

TESIS DE GRADO

Previa a la obtención del Título de

INGENIERO EN COMPUTACION

Presentada por

IVONNE HACAY-CHANG LEON

GUAYAQUIL - ECUADOR

1995

A todas aquellas personas que colaboraron en la realización de este trabajo, y que confiaron, me apoyaron e incentivaron en todo momento. Mil Gracias.

A MI PADRE

A MI MADRE

A MIS HERMANOS

A TI

DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

(Reglamento de Exámenes y Títulos profesionales de la ESPOL).

Ivonne Hacay-Chang León

RESUMEN

Para el presente proyecto se aplican técnicas que proporcionan una herramienta útil para trabajos futuros en los cuales se requiera establecer la comunicación entre una computadora y un dispositivo externo sin enlace directo, además de poder controlar gráficamente las operaciones de dicho dispositivo. Para ésto, se plantea el control por computadora de un vehículo prototipo a través de señales de radio y cuyos movimientos son monitoreados por pantalla.

El equipo utilizado es un computador personal compatible con IBM, operando bajo el Sistema Operativo DOS, el cual se comunica con el vehículo prototipo a través de una interfaz digital/analógica conectada al control remoto del vehículo que genera las señales de radio respectivas.

Para monitorear las operaciones del vehículo prototipo se aplican técnicas avanzadas de programación, graficación y animación por computadora utilizando un lenguaje de programación orientado a objeto.

Con el fin de aplicar las técnicas desarrolladas, se propone un proyecto que consiste en controlar un vehículo prototipo que se encuentra en un laberinto, del cual deberá salir ya sea en forma manual, es decir guiado por el operador, o en forma automática ejecutando el algoritmo correspondiente. Los movimientos del vehículo observados en la pantalla del computador serán transmitidos al vehículo prototipo por medio de señales de radio control.

INDICE GENERAL

Pág.

INDICE DE FIGURAS

INTRODUCCION

El presente proyecto consiste en el control por computadora de un vehículo prototipo a través de señales de radio y cuyos movimientos son monitoreados por pantalla.

Básicamente, se requiere del desarrollo e implantación de dos módulos: un Módulo de Software, constituido por programas desarrollados en un lenguaje de programación orientado a objeto (LPOO por sus siglas en inglés), que permitan controlar y visualizar el vehículo prototipo; y, un Módulo de Electrónica, constituido por un circuito de transmisión de señales de radio desde el computador hasta el vehículo prototipo.

Para la implantación de los módulos desarrollados, se propone el caso de un vehículo encerrado en un laberinto. El vehículo deberá hallar la salida, ya sea guiado por el operador o aplicando el algoritmo planteado. Todos los movimientos del vehículo que se observen en la pantalla del computador serán transmitidos al vehículo prototipo a través de señales de radio.

Para el Módulo de Software se emplea preferiblemente un lenguaje de programación orientado a objeto, que permite aplicar técnicas avanzadas de programación, graficación, animación y comunicación. Para el Módulo de Electrónica se aplican técnicas básicas de electrónica y comunicación, las cuales permiten crear la interfaz digital/analógica y establecer la comunicación entre la computadora, la interfaz y el control remoto del vehículo prototipo.

Este documento se inicia con un análisis detallado del proyecto, describiéndolo y especificando sus objetivos, requerimientos y estructura.

Luego se procede a analizar el Módulo de Software, presentando la herramienta utilizada para su desarrollo, los módulos que lo componen, y las técnicas aplicadas tanto de programación, graficación y animación.

Como siguiente punto, se presenta un análisis del Módulo de Electrónica indicando el equipo utilizado, las condiciones de implantación, los circuitos desarrollados y las conexiones entre los diferentes elementos.

Finalmente, se plantea y analiza el proyecto propuesto para aplicar las técnicas desarrolladas.

CAPITULO I

ANALISIS DEL PROYECTO

Con el fin de analizar detalladamente el proyecto que se plantea, es necesario conocerlo a fondo, lo cual se logra a través de su definición, determinación de su alcance de desarrollo e implantación, sus objetivos planteados, la metodología aplicada y los requerimientos de equipo. Además, al determinar la estructura del proyecto se podrá identificar cada uno de los módulos que lo componen.

1.1 DEFINICION Y ALCANCE DEL PROYECTO

Se desea controlar por computadora los movimientos de un vehículo prototipo a través de señales de radio que son generadas desde su control remoto, el cual estará conectado a una interfaz digital/analógica, y que a su vez estará conectada a la computadora que enviará las señales digitales.

Los movimientos del vehículo prototipo podrán ser monitoreados desde la pantalla del computador, para lo cual se deberá desarrollar una librería de procedimientos

gráficos que permitan la visualización del vehículo con sus movimientos básicos de traslación y rotación (en dos dimensiones).

Los movimientos del vehículo prototipo dependerán únicamente de las señales enviadas desde el computador.

La aplicación podrá ser instalada en cualquier computador personal compatible con IBM con monitor a color.

Una vez creada la interfaz digital/analógica y desarrollados los procedimientos gráficos y de control del vehículo prototipo, se desarrollará una aplicación para el sistema.

1.2 OBJETIVOS

Los objetivos planteados para el presente proyecto son los siguientes:

1. Diseño e implantación de un Módulo de Software constituido por los programas que permitan el control y visualización del vehículo prototipo.

2. Diseño e implantación de un Módulo de Electrónica constituido por un circuito de transmisión de señales desde el computador hasta el vehículo prototipo.

3. Aplicación de la técnica de Programación Orientada a Objeto, optimizando el desarrollo, implantación y ejecución de los códigos que constituyen el Módulo de Software.

4. Diseño e implantación de una aplicación del sistema controlador.

1.3 REQUERIMIENTOS

A continuación se detallan las características del equipo básico que se requiere para instalación y ejecución del presente proyecto:

- Una computadora compatible con IBM con los siguientes elementos:
  - Monitor a color
  - 4Mb. de memoria RAM (mínimo)
  - Procesador 80386 SX en adelante
  - Disco Duro opcional

- Un vehículo prototipo a control remoto que ejecute los movimientos de traslación (adelante y atrás) y de rotación (derecha o izquierda ya sea hacia adelante o hacia atrás).

- Una interfaz digital/analógica que tome las señales digitales que salen de la computadora a través del puerto paralelo y las transforme en señales analógicas, para luego transmitirlas al control remoto del vehículo prototipo, el cual las recepta como señales de radio.

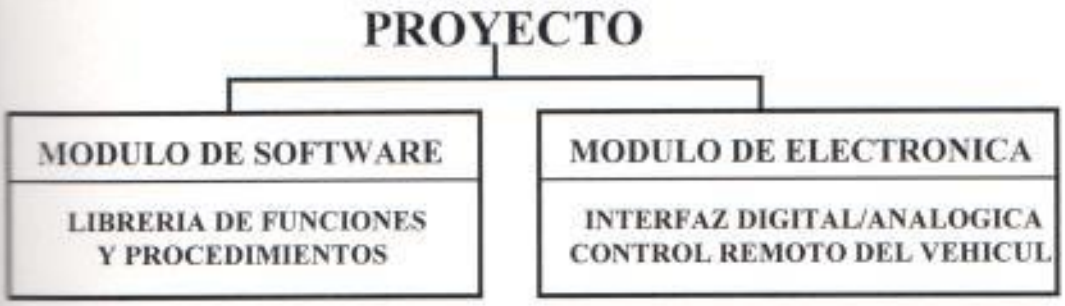
Externamente el equipo se conecta de la siguiente manera:

Fig. 1.1. Conexión del equipo

1.4 ESTRUCTURA DEL PROYECTO

De acuerdo a la descripción del proyecto planteado, se pueden identificar básicamente dos módulos: el Módulo de Software constituido por los procedimientos y funciones que permiten visualizar y controlar los movimientos en pantalla del vehículo, y el Módulo de Electrónica que permite el control físico del vehículo prototipo.

Fig. 1.2. Diagrama de bloques del proyecto



MODULO DE SOFTWARE.-

Conformado por una librería de funciones y procedimientos gráficos desarrollados en un lenguaje de programación orientado a objeto (LPOO) que permita optimizar la manipulación de los elementos que componen el proyecto, ya sea para controlar visualmente el vehículo en la pantalla, como para generar las señales que indiquen los movimientos que son transmitidos al vehículo prototipo.

MODULO DE ELECTRONICA.-

Constituido por la interfaz digital/analógica y el control remoto del vehículo, donde la interfaz digital/analógica toma las señales digitales enviadas por el puerto paralelo de la computadora y las transforma en señales analógicas que transmite al control remoto, el cual genera las señales de radio que se transmiten al vehículo prototipo.

En los capítulos II y III se detallarán cada uno de los módulos en mención.

CAPITULO II

MODULO DE SOFTWARE

El Módulo de Software está constituido por todos los programas creados para el proyecto, que en este caso han sido desarrollados con el lenguaje de programación C++. Este capítulo ha sido estructurado considerando tres aspectos básicos: la programación orientada a objeto, la graficación, la animación y el control del vehículo prototipo.

2.1 ANALISIS DE TECNICAS APLICADAS

Con el fin de efectuar un análisis más detallado de las técnicas aplicadas en el Módulo de Software, se han identificado cuatro aspectos diferentes, entorno a los cuales han sido desarrollados los programas que lo constituyen. Estos son:

- PROGRAMACION
- GRAFICACION
- ANIMACION
- CONTROL

A continuación se procederá a analizar cada uno de estos aspectos.

2.1.1 TECNICAS DE PROGRAMACION

Para desarrollar el módulo de software se utilizó el lenguaje de programación C++ por ser uno de los más poderosos y flexibles tanto para el diseño como para la implantación; y que además, permite obtener códigos estructurados, compactos, rápidos, portables y ejecutables, optimizando las operaciones que se requieren efectuar en el proyecto.

C++ es considerado un lenguaje de programación de alto nivel puesto que ofrece las facilidades de desarrollo y mantenimiento, y a su vez brinda muchos de los beneficios de los lenguajes de programación de nivel medio y bajo, ya que la sintaxis y la estructura de las funciones estándares de C++ permiten operar con los registros internos de la computadora.

El programador tiene la flexibilidad de optimizar la estructura de sus códigos de acuerdo a los requerimientos de su aplicación. Así pues, en caso de que la prioridad sea la velocidad de ejecución del programa, entonces en las principales operaciones pueden aplicarse las características de programación de bajo nivel; en cambio, si el desarrollo y el mantenimiento de la codificación son más importantes, entonces se aplicarán las características de programación de alto nivel.

Una de las características que más anima a trabajar con C++ es su característica de ser un lenguaje modular, es decir, que secciones específicas de código diseñadas para ejecutar una tarea específica, no se encuentran en el programa principal, sino que están dentro de sus propias funciones.

2.1.1.1 VENTAJAS DE LA PROGRAMACION MODULAR

1) Se eliminan las redundancias de código. Es mucho más eficiente producir una función que ejecute una tarea específica que adicionar varias líneas de código cada vez que se las requiera.

2) El código modular es fácil de leer e interpretar. Se puede interpretar fácilmente la operación de la función dándole un nombre que represente la tarea que realiza.

3) El código modular es fácil de mantener puesto que es fácil de leer e interpretar.

2.1.1.2 PROGRAMACION ORIENTADA A OBJETO

Otra ventaja de trabajar con C++ es que es un lenguaje orientado a objeto, es decir que nos permite trabajar con objetos, que no son más que una nueva clase de estructura que puede contener tanto datos como funciones.

La Programación Orientada a Objeto (OOP - por sus iniciales en inglés) es una filosofía de diseño que permite al programador clasificar y generalizar objetos. Por ejemplo, se sabe que un objeto proporciona una gran cantidad de información, y si se pidiera encontrar una pelota en una habitación, ciertas imagenes aparecen en nuestra mente, tales que todas las pelotas son redondas y de aproximadamente un pie de diámetro. Por lo tanto, se ha podido clasificar una variedad de pelotas bajo esta

descripción, y solamente conociendo esta información se tiene una buena oportunidad para encontrar la pelota.

La programación orientada a objeto, además de ser una programación modular, presenta varias ventajas, y se caracteriza principalmente por la aplicación de:

- ENCAPSULACION
- HERENCIA
- POLIMORFISMO

con las cuales es posible reducir la complejidad de aplicaciones extensas, además de desarrollar códigos modulares y que tengan mayor facilidad de reutilización y mantenimiento.

a) ENCAPSULACION

Encapsulación es la práctica del uso de clases para encadenar datos y funciones en una sola estructura.

En la programación tradicional de C, los datos generalmente eran guardados en estructuras, y las funciones eran creadas en forma separada para manipular los datos. Un ejemplo de este estilo se muestra a continuación:

```
struct datos
{
```

```
        int a;
        int b;
        int c;
};
void manipulacion_de_datos (int x, int y , int z)
{
datos.a = datos.a + x;
datos.b = datos.b + y;
datos.c = datos.c + z;
}
```

Tanto la estructura *datos* como la función *manipulacion_de_datos* deben constar en un programa fuente, compilados en forma separada y tratados como un módulo. El problema con este método es que no pueden ser usados los datos y la función en conjunto, sino que los datos debe ser accesado sin usar la función descrita.

La encapsulación resuelve este problema ya que permite combinar tanto datos como funciones en una sola estructura llamada **"clase"**. Los datos son llamados **"miembros datos"** y las funciones **"miembros funciones"**.

Un ejemplo de clase se indica a continuación:

```
class Circulo {
private:
        int centro_x;
        int centro_y;
```

```
        int radio;
public:
        Circulo (void);
        void DibujoCirculo (void);
};
Circulo::Circulo (void)
{
        centro_x = 100;
        centro_y = 100;
        radio = 20;
}
void Circulo::DibujoCirculo (void)
{
        circle (centro_x, centro_y, radio);
}
main( )
{
        int graphdriver = DETECT, graphmode;
        Circulo micirculo;
        initgraph (&graphdriver, &graphmode, "..\\bgi");
        micirculo.DibujoCirculo( );
        getch( );
        return 0;
}
```

donde los miembros datos son *centro_x*, *centro_y*, y *radio*; y los miembro función son *Circulo* y *DibujoCirculo*.

Definiendo la clase *Circulo*, las propiedades del objeto no pueden ser accesadas directamente desde el exterior, sino sólo a través de las funciones de *Circulo* y *DibujoCirculo*. Es decir, las características del objeto pueden ser invocadas únicamente enviando información al objeto, donde los detalles de implantación no son visibles o accesibles desde el exterior.

b) HERENCIA

Herencia es la habilidad de crear una clase con las propiedades y características de otra clase. Es decir, que es posible tomar un código que ya existe y reutilizarlo desarrollando una extensión del mismo.

Tomando el ejemplo que sigue, es posible identificar la propiedad de herencia que posee el C++.

```
#include <iostream.h>
class Perro
{
        public:
        void patas (void);
        void cabeza (void);
        void cola (void);
}
```

```
class Perro_Pequeño : public Perro
{
    public:
    void pequeño (void);
    void liviano (void);
};
class Chihuahua : public Perro_Pequeño
{
    public:
    void nombre (void);
    void patas_cortas (void);
    void pelo_corto (void);
};
void Perro::patas (void)
{
    cout << "Tengo 4 patas..";
}
void Perro::cabeza (void)
{
    cout << "Tengo una cabeza...\n";
}
void Perro::cola (void)
{
    cout << "Tengo una cola...\n";
}
void Perro_Pequeño::pequeño (void)
```

```
{
        cout << "Soy pequeño...\n";
}
void Perro_Pequeño::liviano (void)
{
        cout << "Soy liviano...\n";
}
void Chihuahua::nombre (void)
{
        cout << "Soy un perro Chihuahua...\n";
}
void Chihuahua::patas_cortas (void)
{
        cout << "Tengo patas cortas...\n";
}
void Chihuahua::pelo_corto (void)
{
        cout << "Tengo el pelo corto...\n";
}
main ()
{
        Chihuahua.miperro;
        miperro.nombre;
        miperro.patas;
        miperro.cabeza;
        miperro.cola
```

```
        miperro.pequeño;
        miperro.liviano;
        miperro.patas_cortas;
        miperro.pelo_corto;
        return 0;
}
```

donde la salida del programa será:

Soy un perro Chihuahua...
Tengo 4 patas...
Tengo una cabeza...
Tengo una cola...
Soy pequeño...
Soy liviano...
Tengo patas cortas...
Tengo el pelo corto...

Como se puede observar, se ha creado la "**clase base**" llamada *Perro*, pues es a partir de ella que se definen las "**clases derivadas**": *Perro_Pequeño* y *Chihuahua*. Todos los **miembros funciones** de *Perro* han sido heredados a la clase *Perro_Pequeño*, la cual a su vez hereda todos sus **miembros funciones** a la clase *Chihuahua*. Luego, al definir *miperro* en el programa principal que es de clase *Chihuahua*, éste tiene acceso a todos los miembros públicos tanto de *Perro*, *Perro_Pequeño* y *Chihuahua*.

Adicionalmente, pueden crearse otras clases que herenden las funciones de las clases ya definidas, por ejemplo *Pequines* puede tomar las funciones de Perro y Perro_Pequeño, Collie puede tomar las funciones de Perro, etc.; definiendo para cada una sus propiedades específicas.

Esta es una simple ilustración de herencia, a través de la cual es posible generalizar datos y propiedades, proporcionando una mayor eficiencia en la programación y reduciedo las redundancias en el código.

c) POLIMORFISMO

Polimorfismo en C++ es la habilidad de crear varias versiones de una misma función u operador. La librería de funciones de C++ contiene varias funciones que pueden trabajar con varios tipos de datos. Por ejemplo, si tenemos varias funciones, todas ellas con el mismo nombre, pero con diferentes parámetros:

int cuadrado (int valor);
float cuadrado (float valor);
double cuadrado (double valor);

Todas las funciones se llaman *cuadrado*, sin embargo, cada función está diseñada para aceptar y retornar un tipo de dato particular. En el caso de lenguaje C, solamente se puede tener una función con un nombre; en cambio, C++ acepta declarar varias funciones con el mismo nombre, pero que difieren en su argumento y la información que retornan, C++ es capaz de reconocer y aplicar la versión correcta de la función

invocada. En este caso, si la función *cuadrado* es llamada utilizando un valor entero como parámetro, entonces el valor retornado será un entero, si se la invoca utilizando un valor tipo float o double, entonces la función retornará un valor tipo float o double, respectivamente.

2.1.1.3 APLICACION EN EL PROYECTO

Para el desarrollo del Software del proyecto de tesis se han utilizado las técnicas de programación estructurada y modular proporcionada por la aplicación de las características que ofrece el lenguaje C++ como lenguaje orientado a objeto.

La ENCAPSULACION fue una de las características de C++ más utilizadas a través de la definición de varias clases que contienen tanto datos como funciones en su estructura.

En los puntos 2.2 y 2.3 se detallan cada una de las clases definidas. Cabe indicar que estas clases han sido agrupadas en librerías con el objetivo de lograr una mejor estructuración y modularización de la codificación, permitiendo una mejor interacción entre los procesos que se ejecutan en el Módulo de Software.

La HERENCIA ha sido aplicada en el proyecto para el diseño de la aplicación, en la cual se requiere la visualización de un laberinto, y partiendo del hecho de que un laberinto está constituido por paredes y éstas a su vez están constituidas por bloques, entonces se definió como **clase base** la clase *Bloque*, a partir de la cual se crean las **clases derivadas** que son la clase *Pared* y que a su vez da origen a la clase *Laberinto*.

El POLIMORFISMO no ha sido necesario aplicarlo en desarrollo de las funciones y procedimientos del presente proyecto. Unicamente existe una versión tanto de las funciones y procedimientos generales, como los declarados como miembros de las clases.

2.1.2 TECNICAS DE GRAFICACION

El Módulo de Software está orientado básicamente al ambiente gráfico gracias al cual es posible visualizar el objeto que se desea controlar, además de crear el entorno que comprende la aplicación. Los objetos han sido creados a través de la aplicación de funciones propias de C++ que pertenecen a su librería gráfica, aprovechando al máximo las herramientas que proporciona dicho lenguaje.

Todos los mensajes, menús y objetos están desarrollados bajo una plataforma gráfica, permitiendo una mayor flexibilidad en la manipulación de formas, diseños y colores, además de facilitar la interacción entre todos los objetos en pantalla.

Todas las imagenes que se observan en el proyecto han sido dibujadas en dos dimensiones, para lo cual se trabajará con un sistema de coordenadas bidimensional (eje X y eje Y).

Con el fin de comprender mejor la plataforma gráfica aplicada en el proyecto, se requiere hacer un análisis previo del sistema de coordenadas aplicado.

2.1.2.1 SISTEMA DE COORDENADAS

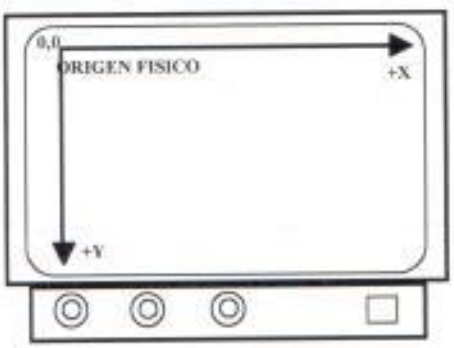
Las computadoras personales tienen básicamente dos modos de operación de video. La primera es el modo texto, donde la pantalla está dividida en celdas de caracteres, generalmente 80 celdas de ancho y 25 celdas de alto. El segundo modo de operación es el modo gráfico, en la cual la pantalla está dividida en píxeles, donde un pixel no es más que la parte más pequeña de una pantalla.

Cuando se trabaja en modo gráfico, C++ reconoce dos sistemas de coordenadas: las físicas y las visuales.

a) SISTEMA DE COORDENADAS FISICAS

Las dimensiones del sistema de coordenadas físicas está determinado por: el hardware, la configuración de la pantalla, y el modo de video. Su origen (0,0) está localizado en la esquina superior izquierda de la pantalla.

Fig. 2.1 Sistema de Coordenadas Físicas

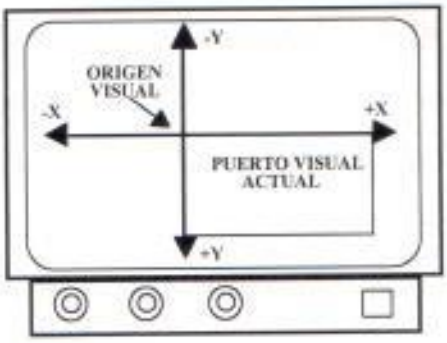


El eje positivo de las X se extiende hacia la derecha de la pantalla, mientras que el eje positivo de las Y se extiende hacia abajo de la pantalla. El máximo valor de X es determinado por el número de pixeles en la dirección horizontal de la pantalla, y el máximo valor de Y es determinado por el número de pixeles en la dirección vertical de la pantalla. Estos valores varían de acuerdo al modo de video utilizado; por ejemplo, si se aplica el modo de video EGAHI, existen 640 pixeles en la dirección horizontal y 350 en la dirección vertical.

b) SISTEMA DE COORDENADAS VISUAL

El sistema de coordenadas visual está basado en el puerto visual actual. Un puerto visual puede describirse como una región rectangular en la pantalla, donde todos los gráficos aparecen en una posición relativa al puerto visual. Por omisión, el sistema de coordenadas visual es el sistema de coordenadas físicas.

Fig. 2.2. Sistema de Coordenadas Visual



La diferencia entre los dos sistemas de coordenadas es que el sistema de coordenadas visual puede moverse de acuerdo al puerto visual definido. Cuando se define un puerto visual, entonces el origen del sistema de coordenadas visual es ubicado en la esquina superior izquierda del **puerto visual**.

El puerto visual se define de acuerdo a las coordenadas físicas. Por tanto el sistema de coordenadas físicas se utiliza sólo como una referencia para la definición del puerto visual.

El sistema de coordenas visual tiene la misma resolución que el sistema de coordenadas físicas. Por tanto, si tenemos 640x350 pixeles disponibles en pantalla, éstos son válidos para ambos sistemas de coordenadas, pero al definir un nuevo puerto visual, los pixeles toman otros valores relativos a su posición de origen.

2.1.2.2 RELACION DE ASPECTO

La manera más sencilla de crear gráficos en la pantalla es dibujar la figura en un papel, marcar los puntos relevantes y luego transferir estas coordenadas al programa gráfico. Sin embargo existe un problema, ya que generalmente se aplica papel cuadriculado y los pixeles no son cuadrados (los modos VGA son la excepción). Por tanto, al correr la imagen, ésta aparece distorcionada. Con el fin de crear la imagen deseada es necesario considerar la relación de aspecto de la configuración de video.
La relación de aspecto de la configuración de video es la relación entre el número de píxeles en la línea vertical de la pantalla y el númerode píxeles en la línea horizontal con la misma longitud. Por ejemplo, si la relación de aspecto es 1 entonces una línea

vertical de 20 píxeles tendrá la misma longitud que una línea horizontal de 20 píxeles. Por lo tanto, si se desea que la imagen no aparezca distorcionada, deberá multiplicarse la dimensión en *y* por la relación de aspecto. Se puede utilizar la siguiente fórmula para determinar la relación de aspecto:

$$\text{RELACION DE ASPECTO} = \frac{\text{ANCHO DE PANTALLA}}{\text{ALTURA DE PANTALLA}} \times \frac{\text{NUMERO DE PIXELES EN Y}}{\text{NUMERO DE PIXELES EN X}}$$

2.1.2.3 APLICACION EN EL PROYECTO

Para el desarrollo de las imagenes mostradas en el proyecto se necesitan aplicar ampliamente los conceptos de Sistema de Coordenadas y Relación de Aspecto, con los cuales es posible obtener imagenes que simulan la realidad de una manera bastante clara y sencilla.

En el proyecto se aplica graficación de tres objetos principales: del vehículo prototipo, del laberinto, y de letras especiales diseñadas para mostrar textos que resalten como títulos. A continuación se presenta el detalle de la graficación de cada uno de los objetos mencionados.

a) GRAFICACION DEL VEHICULO PROTOTIPO

Con el fin de poder monitorear los movimientos del vehículo prototipo, se ha tomado un plano de planta del mismo, es decir que en la pantalla se puede observar una vista superior del vehículo.

Previa la graficación del vehículo se requiere definir un puerto visual cuya escala respete las dimensiones reales del vehículo prototipo, para lo cual se han considerado los siguientes puntos:

- Las dimensiones reales del vehículo son de 15 x 30 cm..

- La aplicación se ejecuta utilizando un monitor a color VGA con dimensiones de 640 x 480 pixeles.

- Se debe tomar un factor de conversión que permita que la imagen del vehículo diseñado no aparezca distorcionada en la pantalla, para lo cual se establece una relación de:

1 cm. = 2 pixeles

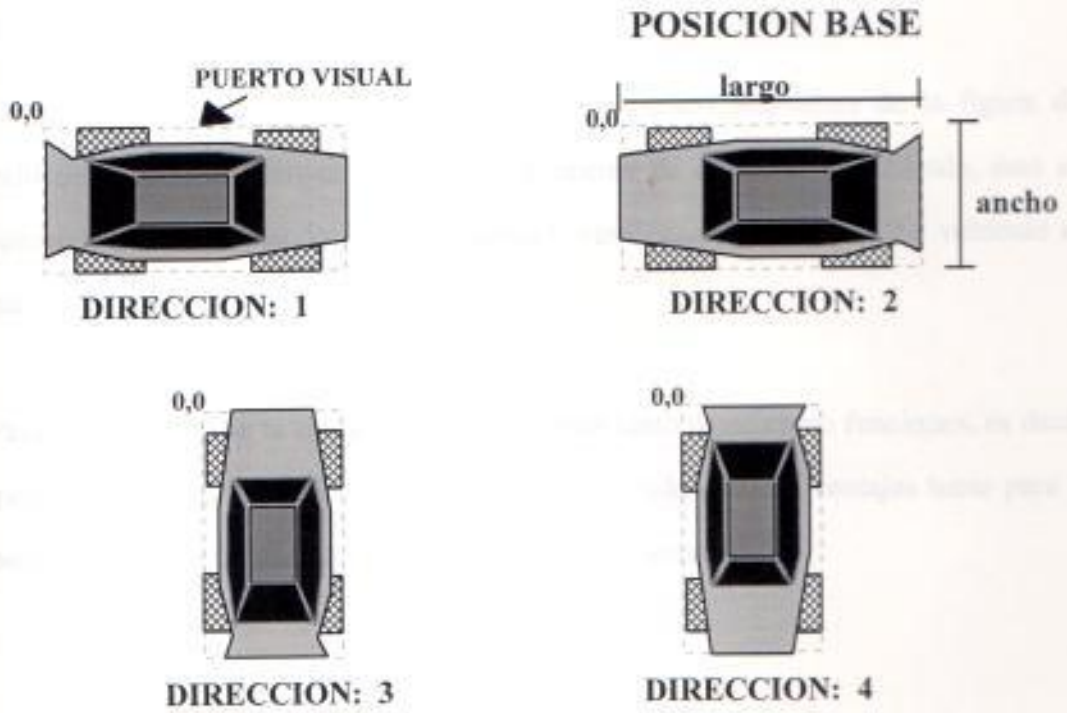
El resultado de estas consideraciones es un puerto visual para el vehículo de **30 x 60 pixeles**. Este puerto visual permite la manipulación de la imagen del vehículo al momento de controlar los movimientos del mismo.

La figura del vehículo prototipo puede tener cuatro direcciones respecto a la pantalla del computador, las cuales han sido numeradas con el fin de poderlas identificar (ver Fig. 2.3.).

POSICION	SIMBOLO	DIRECCION
Vista a la derecha	→	**1**
Vista a la izquierda	←	**2**
Vista hacia arriba	↑	**3**
Vista hacia abajo	↓	**4**

El vehículo ha sido diseñado para una posición inicial con vista hacia la izquierda, es decir en la Dirección 2., y de acuerdo a los requerimientos de ejecución del proyecto, el vehículo aparece en la posición respectiva. En todos los casos se requiere la definición del puerto visual, donde se observa la posición del origen (0,0) en la esquina superior izquierda, la cual se tomará como referencia en todos los casos. En la posición base se indican también los conceptos de ancho y largo del vehículo prototipo.

Fig. 2.3. Vista Superior del Vehículo Prototipo en todas sus Direcciones



Para la graficación del vehículo prototipo se ha utilizado una clase llamada *Carro* que tiene un miembro función llamada *Inicializa* que define un arreglo de tipo *pointtype* (propio de C++) cuya estructura está constituida por dos variables enteras *X* y *Y*, que servirán para determinar las coordenadas de cada uno de los puntos relevantes del vehículo, tomando como referencia el origen del puerto visual definido. Cada punto relevante del vehículo corresponde a un pixel. La dimensión de este arreglo es igual al número de puntos relevantes que constituyen la figura del vehículo prototipo.

Luego, para la graficación se toman todos los puntos relevantes y se aplican las funciones de C++ que permiten unir los puntos correspondientes, formando polígonos y líneas que van a dar forma al vehículo. Finalmente, se aplican las funciones que

permiten colorear las figuras obteniendo la imagen deseada. Esto se efectúa con el miembro función de *Carro* llamada *Grafico*.

La aplicación de un arreglo que almacene los puntos relevantes de la figura del vehículo brinda una mayor facilidad al momento de controlar el vehículo, ésto es, sensando paredes, rotando en las esquinas y verificando la posición del vehículo en pantalla.

Para la definición de la clase *Carro* se utilizaron tanto datos como funciones, es decir, que se ha aplicado la ENCAPSULACION ofreciendo grandes ventajas tanto para la programación como para la manipulación de la información.

b) GRAFICACION DEL LABERINTO

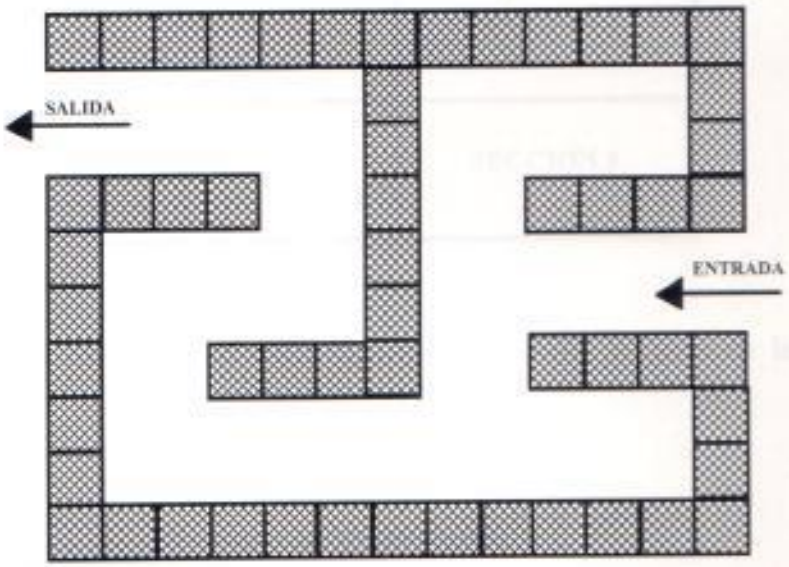
Para la graficación del laberinto se trabaja con el sistema de coordenadas físicas. Para este caso, se ha aplicado la característica de HERENCIA que proporciona el C++, partiendo de la idea que un laberinto está constituido por paredes y estas a su vez por bloques.

Inicialmente, se ha definido un clase base llamada *Bloque*, la cual hereda todos sus miembros públicos a la clase *Pared*, y ésta a su vez hereda todos sus miembros públicos a la clase *Laberinto*. En la Fig. 2.4. se observa un ejemplo de un bloque y cómo la unión de los mismos crean paredes horizontales y verticales. En el caso de las paredes, se indica el **punto origen**, con coordenadas en X y Y, que es el que se toma como referencia para posicionar la pared en la pantalla.

Fig. 2.4. Gráficos de Bloque, Pared Horizontal,.Pared Vertical

La clase *Bloque* es la que toma todas las características del bloque: forma, tamaño y color. La clase Pared es la que determina la forma de agrupamiento de los bloques, de tal manera que formen paredes horizontales o verticales (en relación a la pantalla del computador). Y por último, la clase Laberinto es la que proporciona el diseño del laberinto. En la Fig. 2.5. se presenta un ejemplo de una porción de laberinto.

Fig. 2.5. Ejemplo General de un Laberinto



Se han diseñado cuatro diferentes laberintos, donde cada laberinto tiene el tamaño de 2x3 secciones, cada una de ellas del tamaño visual de una pantalla (ver Fig. 2.6.). El diseño de estos laberintos es estático, es decir que se encuentran predefinidos y no es posible variarlos entre una ejecución y otra. El tamaño de bloque aplicado ha sido de 30 pixeles, con el fin de poder lograr un efecto visual más claro y preciso en pantalla.

Fig. 2.6. Estructura General de un Laberinto

SECCION 0	**SECCION 1**
SECCION 2	**SECCION 3**
SECCION 4	**SECCION 5**

Para obtener la estructura de cada sección de laberinto se parte de las siguientes premisas:

a) Cada bloque tiene el tamaño de 30x30 pixeles.

b) Cada pared tiene un tamaño de 4 bloques, donde uno de ellos se utiliza de enlace con la siguiente pared.

c) Se trabaja con un monitor a color VGA donde cada pantalla tiene un tamaño de 640x480 pixeles:

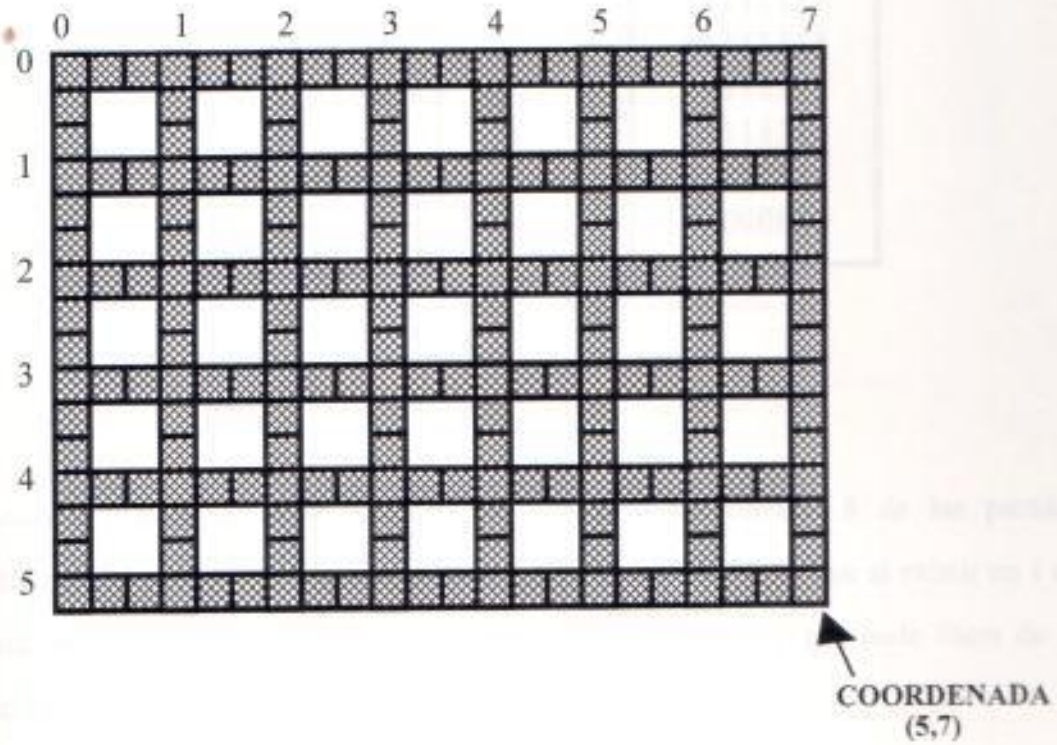
Efectuando los cálculos:

$$640 / 30 \times 3 = 7.11$$

$$480 / 30 \times 3 = 5.33$$

Resultan un máximo de 7 paredes horizontales por 5 paredes verticales por cada sección, donde existirán 6 orígenes para paredes horizontales y 8 orígenes para paredes verticales (ver Fig. 2.7.). A estos orígenes se los identifica como coordenadas del laberinto.

Fig. 2.7. Estructura básica de una Sección de Laberinto



Cada coordenada donde empieza una pared, ya sea horizontal o vertical, está codificada con el fin de identificar la existencia o no de una pared. Un 1 indica la existencia de una pared y un 0 la no existencia. Por tanto, al tomar estos datos y almacenarlos en una matriz de 6x8 para cada sección. resulta inicialmente la codificación que se muestra en la Fig. 2.8..

Fig. 2.8. Codificación Inicial de una Sección de Laberinto

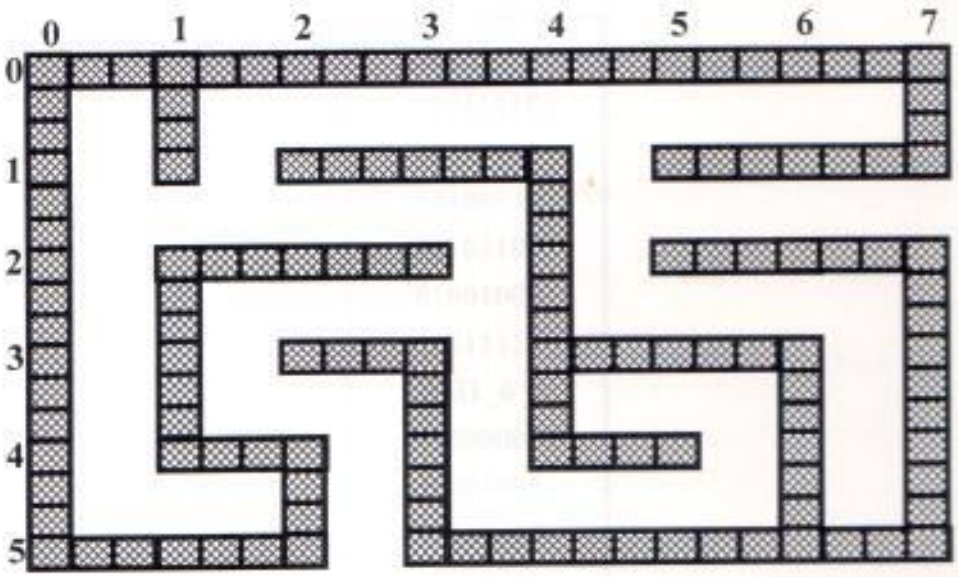
PAREDES HORIZONTALES
11111110
11111110
11111110
11111110
11111110
11111110

PAREDES VERTICALES
11111111
11111111
11111111
11111111
11111111
00000000

Como se puede observar, siempre en este caso la columna 8 de las paredes horizontales será 0, y la fila 6 de las paredes verticales será 0, ya que al existir un 1 en estas posiciones se obtendrían paredes fuera de la sección y por ende fuera de la pantalla del computador.

Al transformar un 1 por un 0 se estará eliminando una pared, y si se reemplaza un 0 por un 1 se estará colocando una pared. A continuación se muestra la sección de un laberinto y su codificación correspondiente:

Fig. 2.9 Ejemplo de una Sección de Laberinto y su Codificación correspondiente.



PAREDES HORIZONTALES
11111110
00110110
01100110
00101100
01001000
11011110

PAREDES VERTICALES
11000001
10001000
11001001
11011011
10110011
00000000

Luego, la unión de 6 de estas secciones de laberinto constituyen un Laberinto completo en el proyecto.

La codificación de los laberintos está almacenada en un archivo tipo texto llamado ***DATOS.TXT*** de la siguiente manera (ver fig. 2.10.).

Fig. 2.10. Codificación del Laberinto 1, Sección 0

```
LAB1_0_H
11111110
00110110
01100110
00101100
01001000
11011110
LAB1_0_V
11000001
10001000
11001001
11011011
10110011
00000000
```

Cada laberinto está identificado por un número de 1 a 4, cada sección está identificada por un número de 0 a 6, y existen dos matrices de 6x8 por cada sección de laberinto, una para las paredes verticales y otra para las paredes horizontales, y donde cada matriz está precedida por un título identificador, por ejemplo: LAB1_0_H indica que la matriz a continuación corresponde al Laberinto 1, sección 0, paredes horizontales, LAB1_0_V indica que la matriz siguiente corresponde al Laberinto 1, sección 0, paredes verticales, y así sucesivamente hasta completar toda la codificación.

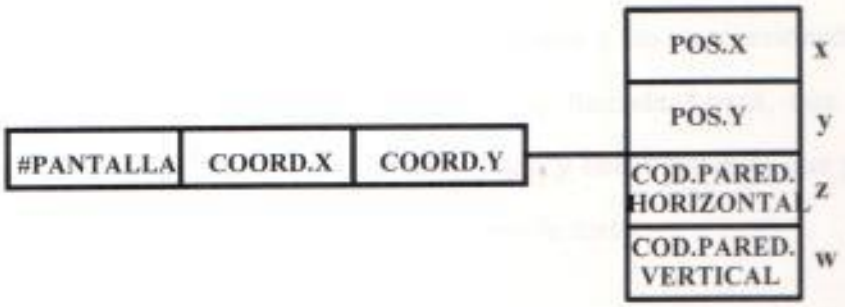
Para la graficación del laberinto, inicialmente se define una estructura llamada PTS constituida por los siguientes elementos:

```
struct PTS
{
        int x, y;
        char z, w;
}
```

donde los enteros *X* y *Y* almacenarán las posiciones en los ejes X y Y respectivamente de los orígenes de las paredes, y los caracteres Z y W almacenarán la codificación de 0 o 1 indicando indicando si existe o no una pared horizontal (Z=1 o Z=0) o una pared vertical (W=1 o W=0). Los valores de X y Y se calculan de acuerdo al tamaño de bloque utilizado, en este caso 30 pixeles.

Luego, se define un arreglo *Laber* de 6x10x10 de tipo *PTS*, el cual almacenará los datos necesarios para graficar un laberinto. El primer parámetro del arreglo *Laber* indicará el número de sección (0 - 5), el siguiente parámetro indica la coordenada en X del laberinto (0 - 9), y el tercer parámetro indica la coordenada en Y del laberinto (0 - 9) (ver Fig. 2.11.). Ya que la estructura de un arreglo es estática, se ha definido un rango de 0 a 9 coordenadas como máximo en el eje de las X y en el eje de las Y. Luego, al momento de almacenar los datos del laberinto, sólo se ocuparán los registros necesarios.

Fig. 2.11 Estructura de un Registro del Arreglo Laber



De acuerdo al número de laberinto escogido, se accesa al archivo DATOS.TXT, posicionando el puntero de lectura al inicio de la información del laberinto correspondiente. Los datos son leídos línea por línea y archivados en el arreglo en los campos respectivos. Luego, de acuerdo a la información almacenada en el arreglo se grafica el laberinto.

Aplicando la HERENCIA se ha logrado que el programa principal interactue únicamente con la clase *Laberinto*, eliminando cualquier tipo de acción que el programa principal pudiera efectuar sobre las clases *Bloque* y *Pared*, las cuales han heredado todas sus funciones miembros a la clase *Laberinto*.

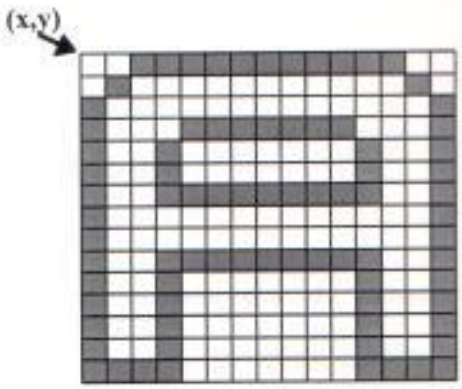
La aplicación de una codificación para la graficación de los laberintos facilita e incrementa la eficiencia en la construcción y diseño de los mismos; sin embargo, existe la limitante de poder crear laberintos más extensos, debido a que la clase *Laberinto* está diseñada para trabajar sólo bajo las premisas mencionadas.

c) GRAFICACION DE LETRAS ESPECIALES

Con el fin de obtener una clase de letras diferentes a las proporcionadas por las librerías propias de C++, se ha creado una clase llamada *Letras*, que brinda un conjunto de letras (más anchas y de tamaño, fondo y color variable) que pueden ser utilizadas para mostrar títulos o cualquier otro tipo de texto.

Para la graficación de estas letras especiales se ha empleado una técnica de diseño utilizando una matriz de puntos de 15x15 pixeles, en la cual fueron dibujadas las figuras correspondientes a cada una de las letras del abecedario (ver Fig. 2.12.)

Fig. 2.12. Modelo de la Letra A



Para identificar la posición de cada punto, se toma como referencia la posición (X,Y) ubicada en la esquina superior izquierda.

Puesto que la figura de la letra se encuentra siempre en la misma dirección (no rota), entonces se ha utilizado un método más sencillo que el aplicado para graficar el vehículo prototipo en el que fue necesario determinar y almacenar las posiciones de

los puntos relevantes tanto en el eje X como en el eje Y. En este caso, se utiliza un arreglo unidimensional de enteros llamado *let_fract* de dimension 15, donde se almacena el valor correspondiente a la posición de cada celda, y este valor se aplicará para ambos ejes. Luego, la posición correspondiente a la celda en la esquina inferior derecha será (X+*let_fract*[14], Y+*let_fract*[14]).

Para variar la posición de la letra en la pantalla, basta con modificar los valores de X y Y. También es posible variar la escala de las letras, para lo cual se deben multiplicar los valores almacenados en *let_fract* por una escala (valor entero).

Finalmente, para graficar la letra, se aplican las funciones propias de C++ que permiten unir estos puntos(celdas) entre sí dando forma a la figura de la letra, y se desea es posible adicionarle un color de fondo.

2.1.3 TECNICAS DE ANIMACION

Animación es el arte de hacer que una serie de imagenes mostradas en pantalla parezcan que tuvieran vida o movimiento. Esta manipulación de las imagenes se logra a través de la apropiada aplicación de transformaciones geométricas al sistema de coordenadas utilizado.

Básicamente existen dos tipos de animación: total y parcial. Con la animación total se utiliza toda la pantalla para crear una animación y con la animación parcial se utiliza sólo una sección de la pantalla.

Para crear una animación, ya sea total o parcial, se pueden aplicar tres transformaciones principales: traslación, escalamiento y rotación. Adicionalmente, también es posible la aplicación del arrastre de imagenes.

A continuación se presenta un detalle de cada una de estas transformaciones.

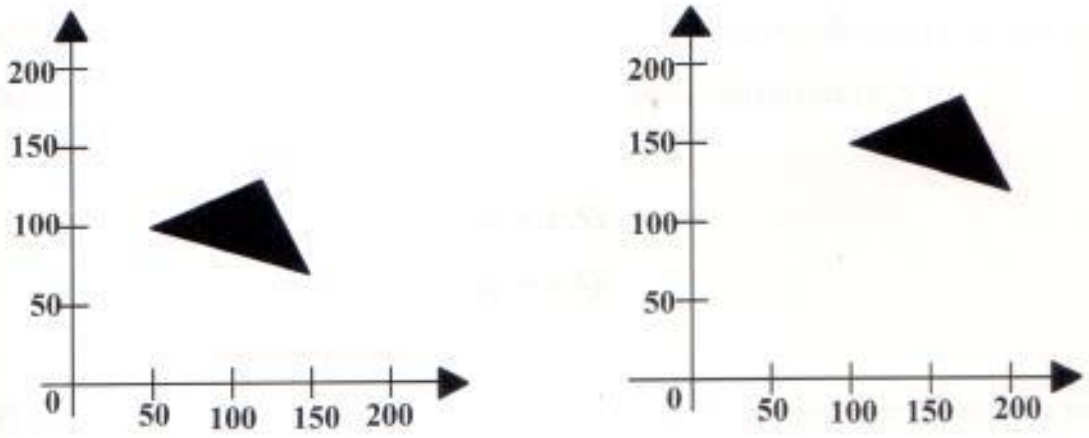
2.1.3.1 TRASLACION

La traslación es el movimiento en línea recta de un objeto de una posición a otra. Se puede trasladar un punto de coordenada *(x, y)* a otra *(x', y')* adicionándole una "distancia de traslación", Tx y Ty para las coordenadas originales *x* y *y* respectivamente:

$$x' = x + Tx$$
$$y' = y + Ty$$

Un polígono puede ser trasladado adicionándole una distancia de traslación a cada uno sus vértices, y luego redibujándolo. La Fig. 2.13. muestra la traslación de un polígono, al cual se le ha aplicada una distancia de traslación de $Tx = 50$, $Ty = 50$.

Fig. 2.13. Traslación de un polígono



Los objetos que presentan curvas pueden ser trasladados al cambiar las coordenadas del objeto, es decir, que por ejemplo, para el caso de un círculo o una elipse se trasladan las coordenadas del centro y se redibuja la figura.

Las distancias de traslación pueden ser especificadas en cualquier número real (positivo, negativo o cero), donde el movimiento se ejecuta en dirección acorde con el sistema de coordenadas indicado en el punto 2.1.2.. Si el objeto es trasladado a coordenadas que exceden las coordenadas máximas de la pantalla, entonces se producirá una distorción de la figura, donde parte de ella quedará fuera de pantalla, o en todo caso se retornará un error.

2.1.3.2 ESCALAMIENTO

Una transformación para alterar el tamaño de la figura es conocida como escalamiento. Esta operación consiste en multiplicar cada coordenada (x, y) por un factor de escalamiento Sx y Sy produciendo una nueva coordenada (x', y'):

$$x' = x\,Sx$$
$$y' = y\,Sy$$

El valor de *Sx* produce un escalamiento en el eje de las X, mientras que *Sy* produce un escalamiento en el eje de las Y.

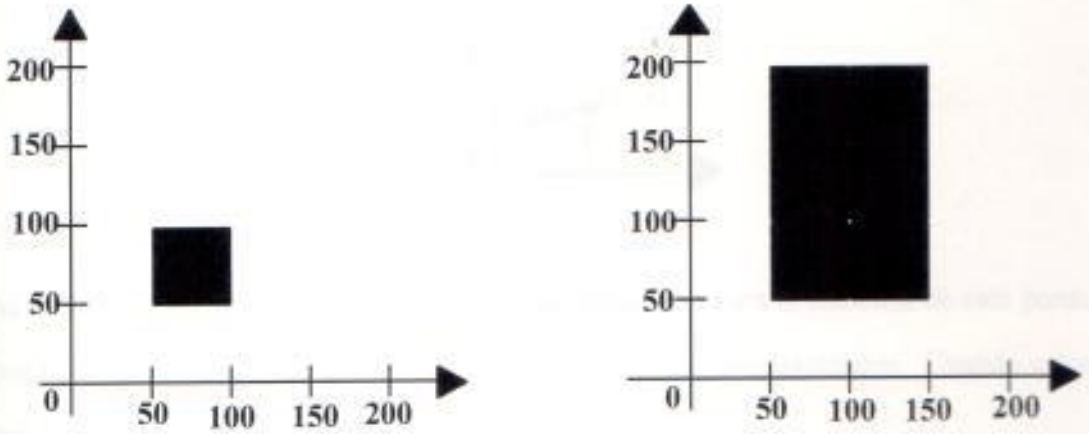
El efecto que producen los valores de Sx y Sy en la figura son los siguientes:

VALORES DE Sx Y Sy	TAMAÑO DE LA FIGURA
> 1	AUMENTA
< 1	DISMINUYE
= 1	MANTIENE

Si *Sx* y *Sy* presentan el mismo valor, entonces el escalimiento de la figura es uniforme.

En la Fig. 2.14. se muestra el escalimiento de un polígono, donde *Sx* = 2 y *Sy* = 3.

Fig. 2.14. Escalamiento de un polígono

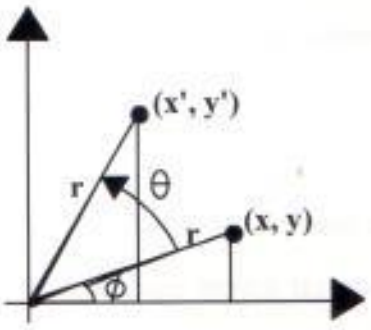


2.1.3.3 ROTACION

La transformación de la posición de un punto alrededor de un eje es llamado rotación. La especificación de este tipo de transformación se la hace con un "ángulo de rotación", el cual determina la cantidad de rotación para cada vértice del polígono.

La Fig. 2.15. muestra el desplazamiento de un punto desde la posición (x, y) a la posición (x', y'), creando un ángulo θ relativo a la posición de origen. En esta figura, el ángulo ϕ es la posición angular original del punto desde el eje horizontal.

Fig. 2.15. Rotación de un punto con eje en el origen



Se pueden determinar las ecuaciones de transformación para la rotación de este punto por la relación entre los lados de los triángulos y los ángulos asociados. Usando estos triángulos y las identidades trigonométricas básicas, resultan:

$$x' = r\,Cos\,(\phi+\theta) = r\,Cos\phi Cos\theta - r\,Sen\phi Sen\theta$$
$$y' = r\,Sen(\phi+\theta) = r\,Sen\phi Cos\theta + r\,Cos\phi Sen\theta$$

donde r es la distancia desde el punto hasta el origen. Luego, tomando las siguientes relaciones:

$$x = r\,Cos\phi$$
$$y = r\,Sen\phi$$

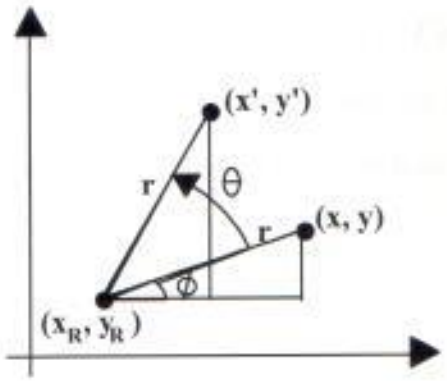
y reemplazándolas en las ecuaciones anteriores, se obtienen las siguientes ecuaciones:

$$x' = x\,Cos\theta - y\,Sen\theta$$
$$y' = y\,Cos\theta + x\,Sen\theta$$

En estas ecuaciones, un valor positivo para el ángulo θ indica un movimiento de rotación en sentido opuesto al movimiento de las manecillas del reloj, y un ángulo negativo indica una dirección opuesta.

Los objetos también pueden rotar alrededor de un punto arbitrario, es decir que el eje será un punto diferente al origen, el cual tendrá las coordenadas (x_R, y_R). A este punto se lo conoce como "punto pivote" (ver Fig. 2.16.).

Fig. 2.16. Rotación de un punto con eje en un punto pivote



Donde las ecuaciones de rotación también se transforman en:

$$x' = x_R + (x - x_R)\cos\theta - (y - y_R)\,Sen\theta$$
$$y' = y_R + (y - y_R)\cos\theta + (x - x_R)\,Sen\theta$$

El punto pivote puede estar ubicado en cualquier lugar, ésto es, si el punto pivote se encuentra dentro del contorno de la figura, entonces el efecto será el giro de la figura tantos grados como se le indique; pero si el punto pivote se encuentra fuera de la figura, entonces se producirá un desplazamiento circular de todos sus puntos alrededor del punto pivote.

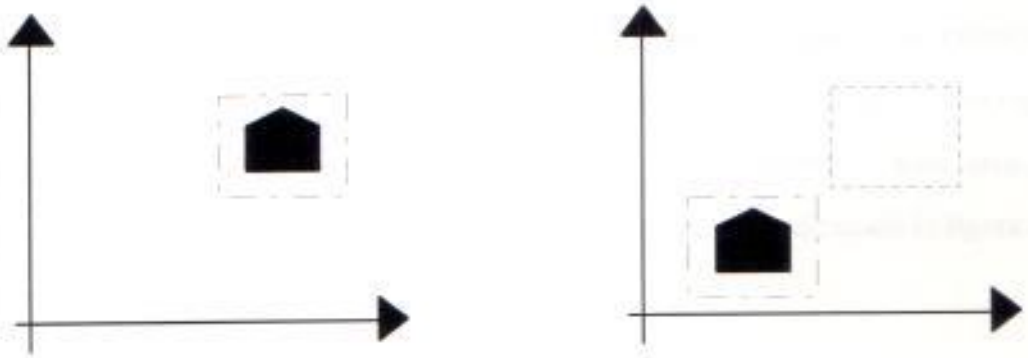
Ya que el movimiento de rotación involucra varias operaciones trigonométricas y aritméticas, el tiempo de procesamiento de la información resultante puede ser bastante elevado, especialmente en el caso de trabajar con muchos puntos que tienen que rotar varias posiciones repetidamente. Una manera de optimizar este procesamiento es considerando que si el ángulo de rotación es pequeño (menor a 10°), entonces se puede trabajar con aproximaciones, donde $Cos\theta$ es aproximadamente igual a 1 y $Sen\theta$ es aproximadamente igual al valor de θ en radianes. El error producido por estas aproximaciones es menor mientras más pequeño sea el ángulo de rotación.

2.1.3.4 ARRASTRE

Otro tipo de transformación de una figura es el arrastre, el cual permite almacenar la imagen de la figura en una variable de memoria, para luego hacerla aparecer en otra posición de la pantalla. Básicamente son pocas las operaciones matemáticas que se requieren efectuar, por tanto se logra una mayor eficiencia y rapidez en el movimento de la figura.

En la Fig. 2.17. se observa la traslación de un polígono aplicando arrastre. Inicialmente se marca un área rectangular alrededor del polígono, y luego, toda la imagen se copia en otra zona, para lo cual se lee la intensidad de los pixeles correspondientes al área rectangular almacenando esta información en una variable en memoria; y luego, toda la información es copiada en la nueva posición. La imagen original puede ser borrada llenando el área rectangular marcada con el color de fondo.

Fig. 2.17. Arrastre de un polígono



Generalmente, la aplicación del arrastre para simular el movimiento de una imagen es mucho más rápida en su ejecución que una traslación, donde es necesario efectuar operaciones con cada uno de los puntos que constituyen la imagen. Es por este motivo, que esta técnica es ampliamente utilizada en las animaciones.

2.1.3.5 APLICACION EN EL PROYECTO

En el presente proyecto se puede observar la animación de dos figuras: el vehículo y el laberinto. La animación del vehículo se traduce en los movimientos que éste posee

en la pantalla: traslación y rotación; estos movimientos los puede efectuar en cualquiera de las direcciones posibles: arriba, abajo, derecha e izquierda. La animación del laberinto se traduce en los cambios de pantalla que se ejecutan al momentos del paso del vehículo de una sección a otra del laberinto (ver Fig. 2.6.).

a) TRASLACION DEL VEHICULO

Para simular el movimiento de traslación del vehículo, ya sea hacia adelante o hacia atrás en cualquiera de sus direcciones (ver Fig. 2.3), se ha aplicado una animación parcial con una transformación de arrastre, donde se toma únicamente el área de la pantalla correspondiente al vehículo para simular el movimiento. Esta área de pantalla tiene el tamaño del puerto visual en el cual se encuentra dibujada la figura del vehículo prototipo.

Previo al movimiento de traslación, la sección de pantalla que contiene la imagen del vehículo es grabada en memoria, lo cual permite reutilizarla para visualizar dicha imagen en otra posición.

Cuando la imagen del vehículo aparece en una nueva posición, se ejecuta una función de "limpieza" de la zona donde estuvo anteriormente, que pinta esa zona con el color de fondo de la pantalla, y luego se vuelve a utilizar la imagen haciéndola aparecer unos pixeles más adelante o más atrás, de acuerdo a la dirección del vehículo y al movimiento requerido. Repitiendo continuamente esta secuencia de limpieza y mostreo de la imagen del vehículo se simula su traslado de un punto a otro en la pantalla; sin embargo, se provoca un parpadeo al observar el movimiento de la

imagen del vehículo, por el tiempo transcurrido entre la limpieza de la zona y la aparición de la imagen en la nueva posición.

Como se indicó anteriormente, el vehículo puede ser trasladado en dos sentidos, hacia adelante o hacia atrás, es decir retroceder, de acuerdo a la dirección en que se encuentre. Al igual que fueron codificadas las direcciones en que se encontraba el vehículo, este movimiento también ha sido codificado.

MOVIMIENTO	DIREC	SIMBOLO	SIMBOLO DE TRASLACION	CODIGO
Retroceso	1	→	←	0
	2	←	→	
	3	↑	↓	
	4	↓	↑	
Avance	1	→	→	1
	2	←	←	
	3	↑	↑	
	4	↓	↓	

En la Fig. 2.18. se ha aplicado la traslación a la imagen del vehículo prototipo con dirección = 2, haciéndolo "avanzar". En este caso con el fin de poder observar mejor el efecto, el avance ha sido bastante significativo (mayor al tamaño del vehículo); pero, en el proyecto el avance se ejecuta a razón de dos pixeles en la dirección

deseada para poder simular en forma más real un avance del vehículo. En caso de no realizarse la función de limpieza, se provocaría un montaje de las imagenes.

Fig. 2.18. Traslación de la imagen del vehículo prototipo

Es posible efectuar la traslación de la imagen del vehículo prototipo aplicando las operaciones propias de traslación, en vez de aplicarse arrastre de imagenes; pero en este caso el efecto de parpadeo es bastante evidente puesto que para cada movimiento se requieren efectuar varias operaciones para luego redibujar totalmente la figura.

Aplicando arrastre a la imagen del vehículo prototipo se agilizó el movimiento del vehículo y se disminuyó considerablemente el efecto de parpadeo de la imagen.

b) ROTACION DEL VEHICULO

Existen dos tipos de rotación de la imagen del vehículo prototipo: la que se realiza para ubicar la imagen en la dirección requerida sin efectuar ninguna traslación y la que se realiza ejecutando una traslación partiendo de cualquiera de las direcciones.

ROTACION SIN TRASLACION

La rotación sin traslación como se indicó se utiliza para posicionar la imagen del vehículo prototipo en la dirección requerida, partiendo de la posición base con dirección igual a 2.

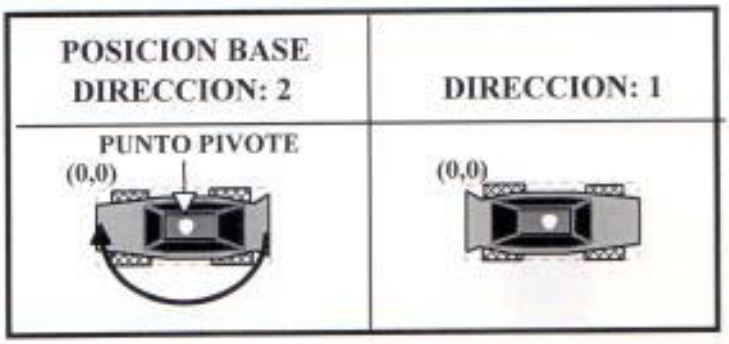
El ángulo de giro y la ubicación del punto pivote varían para cada caso, siempre y cuando el origen de la imagen del vehículo (0,0) no haya cambiado de posición.

A continuación se indican cada uno de los casos que se presentan para obtener las imagenes del vehículo en cada dirección a partir de la posición base.

Dirección 1.-

El punto pivote se ubica al centro de la imagen del vehículo, tanto a lo largo como a lo ancho del mismo, y se efectua un giro de 180°.

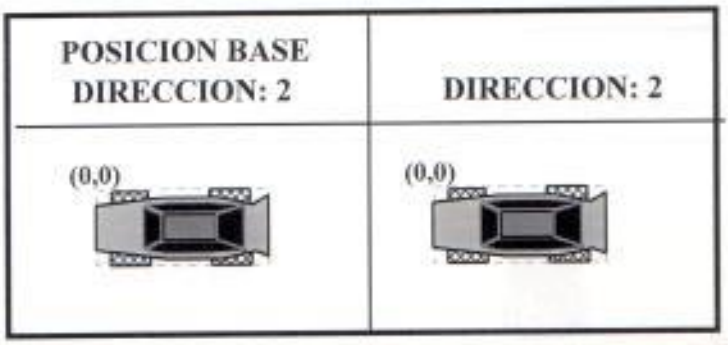
Fig. 2.19 Posicionamiento del vehículo en dirección 1



Dirección 2.-

En este caso no se requiere efectuar ningún movimiento puesto que se trata de la posición base que tiene dirección 2.

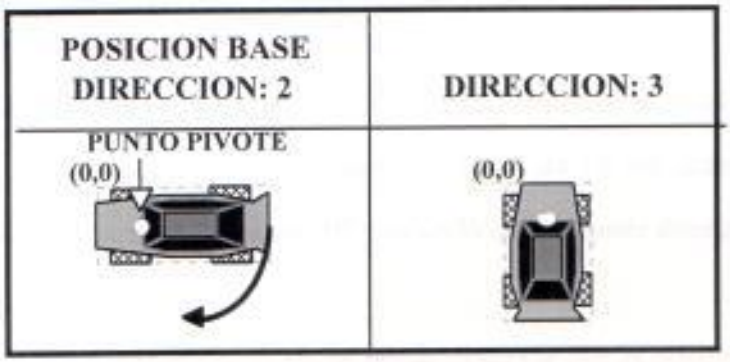
Fig. 2.20 Posicionamiento del vehículo en dirección 2



Dirección 3.-

El punto pivote se ubica al centro del ancho de la imagen del vehículo y a un cuarto del largo. El giro efectuado es de 90°.

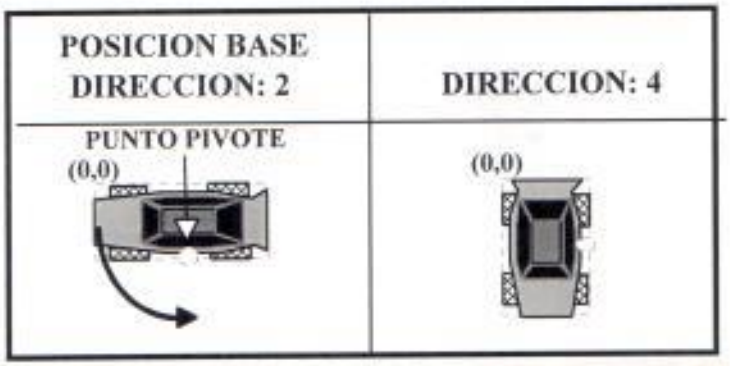
Fig. 2.21 Posicionamiento del vehículo en dirección 3



Dirección 4.-

El punto pivote se ubica en todo el ancho de la imagen del vehículo y a centro del largo. El giro efectuado es de 90°.

Fig. 2.22 Posicionamiento del vehículo en dirección 4

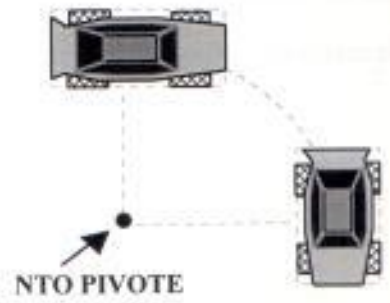


ROTACION CON TRASLACION

Para simular el movimiento de rotación del vehículo prototipo efectuando una traslación, se ha aplicado la rotación de la imagen del vehículo alrededor de un punto pivote ubicado hacia su derecha o izquierda a la altura de su centro. Los giros considerados son únicamente de hasta 90° partiendo de cualquier dirección.

En el ejemplo mostrado en la Fig. 2.19. se puede observar que la posición inicial de la imagen del vehículo prototipo es la dirección 1 y efectua una rotación o giro de 90° con una dirección final de 4. Además se observa que el punto pivote se encuentra ubicado fuera de la imagen del vehículo.

Fig. 2.23. Ejemplo de rotación del vehículo prototipo con traslación



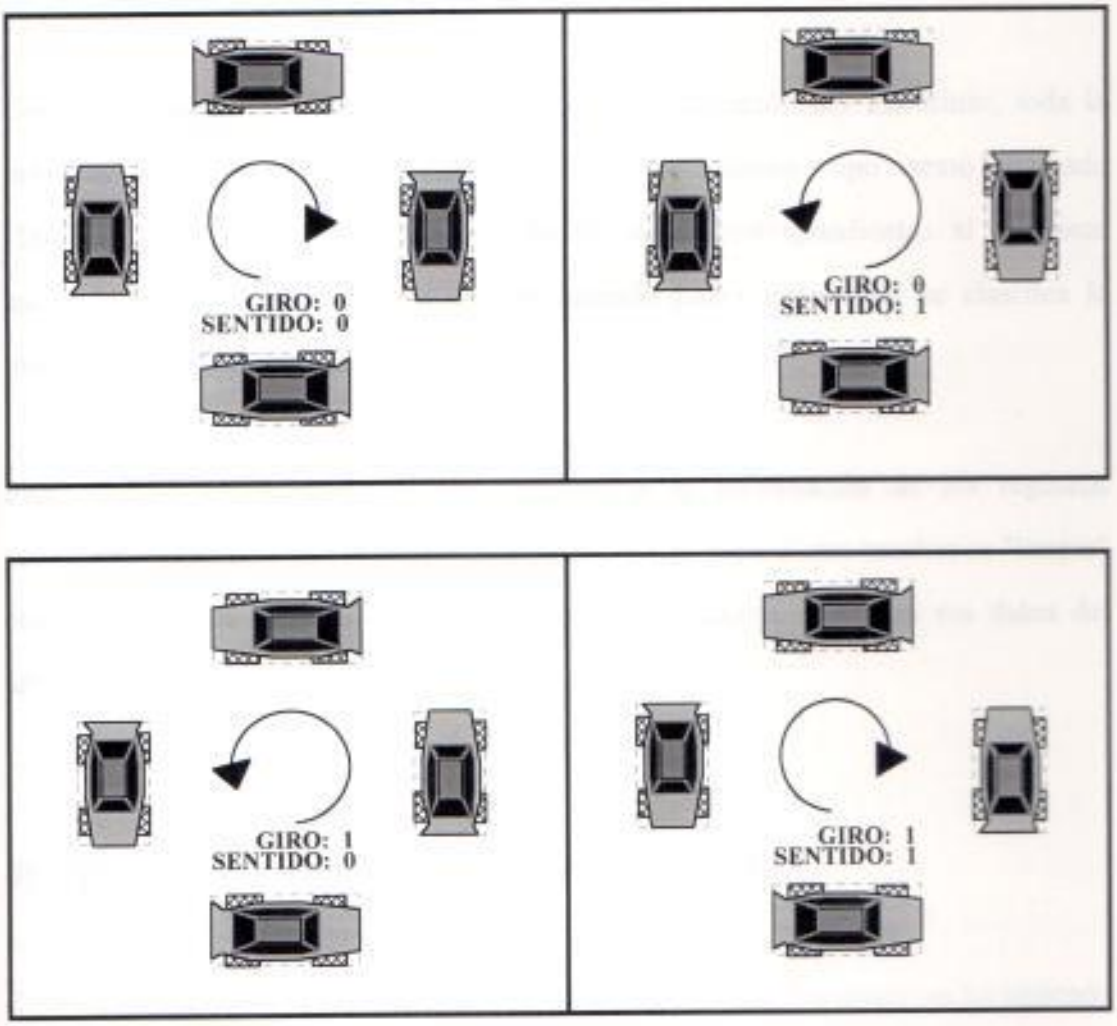
Como se indicó anteriormente, la rotación puede ser realizada en dos sentidos, siguiendo el movimiento de las manecillas del reloj y en sentido opuesto. En este caso los giros serán identificados de acuerdo a la posición del punto pivote, es decir hacia la derecha o hacia la izquierda.

Además, se debe considerar que el movimiento puede ser realizado hacia adelante o hacia atrás. Por tanto, resultan cuatro movimientos de rotación posibles para cada una de las direcciones en que se encuentre la imagen del vehículo prototipo:

GIRO	CODIGO	SENTIDO	CODIGO
Derecha	0	Adelante	0
Derecha	0	Atrás	1
Izquierda	1	Adelante	0
Izquierda	1	Atrás	1

En la Fig. 2.24. se pueden observar más claramente los movimientos de rotación que puede realizar el vehículo prototipo de acuerdo a la dirección en que se encuentre.

Fig. 2.24. Giros y Sentidos de Rotación



c) CAMBIO DE SECCION EN EL LABERINTO

Realmente el laberinto no presenta ningún tipo de movimiento (traslación, rotación, escalamiento o arrastre), pero el hecho de que su tamaño visual sea de seis pantallas

de computador (ver Fig. 2.6) significa que es necesario simular la conexión entre estas secciones al momento de pasar de una a otra.

Como se trató en el punto 2.1.2 respecto a la graficación del Laberinto, toda la información se encuentra almacenada en un archivo tipo texto llamado "DATOS.TXT" del cual se toman sólo los datos correspondientes al laberinto escogido que se trasladan a un arreglo llamado Laber en el cual se clasifica la información en sus registros

Para graficar el laberinto se toma únicamente la información de los registros correspondientes a la sección actual, y al momento de pasar a otra sección se "limpia" completamente la pantalla, se determina la nueva sección, se toman sus datos del arreglo y se grafican.

d) ESCALAMIENTO DE LAS LETRAS ESPECIALES

Como se trató en el punto 2.1.2, en la graficación de Letras Especiales se ha aplicado escalamiento que varía de acuerdo al factor (valor entero) indicado como parámetro de graficación.

2.1.4 TECNICAS DE CONTROL

A nivel de software se analizarán dos aspectos: el envío de señales desde el computador a la interfaz digital/analógica, y la coordinación de los movimientos reales del vehículo prototipo con los movimientos del vehículo en pantalla.

2.1.4.1 ENVIO DE SEÑALES DE CONTROL DESDE EL COMPUTADOR

Para efectuar el control de los movimientos del vehículo prototipo se requiere enviar señales a la interfaz digital/analógica que permitan identificar el movimiento que debe efectuar el vehículo. En este caso, se ha utilizado bytes que son enviados a través del puerto paralelo del computador.

Cada byte (compuesto por ocho bits) es un código que indica una acción diferente que debe ejecutar el vehículo prototipo, como se indica en la Fig. 2.25. Los bits de datos son transmitidos simultáneamente a través de las líneas de datos del puerto paralelo del computador D0 a D7 y se envían a la interfaz digital/analógica.

En el capítulo 3 se detalla la configuración de hardware utilizada, incluyendo la información correspondiente a la conexión entre el puerto paralelo y la interfaz digital/analógica.

Fig. 2.25 Códigos que se envían al puerto paralelo

ACCION	PUERTO PARALELO							
	D7	D6	D5	D4	D3	D2	D1	D0
Avance	0	0	0	1	0	0	0	0
Retroceso	0	0	1	0	0	0	0	0
Rotación adelante - derecha	0	0	1	1	0	0	0	0
Rotación atrás - derecha	0	1	0	0	0	0	0	0
Rotación adelante - izquierda	0	1	0	1	0	0	0	0
Rotación atrás - izquierda	0	1	1	0	0	0	0	0
Pare	0	1	1	1	0	0	0	0

2.1.4.2 COORDINACION DE MOVIMIENTOS DEL VEHICULO

En el instante que el vehículo en pantalla recibe la orden de traslación o rotación, la señal enviada por el computador debe mantenerse el tiempo suficiente para que el vehículo prototipo ejecute el movimiento respectivo, observándose una coordinación entre ambos movimientos: pantalla y realidad.

Para el movimiento de traslación se requiere considerar el factor de conversión establecido para el proyecto: 1cm=2pixeles para cubrir la distancia equivalente en pantalla y en la realidad.

Para el movimiento de rotación, dicho factor no se aplica. La señal que se envía debe mantenerse tiempo suficiente para que el vehículo realice el giro de 90°, 180° o 270°, sin importar el factor de conversión y ni cambios de escala.

En ambos casos, traslación y rotación es necesario aplicar un retardo en el envío de la señal, el cual se obtiene mediante pruebas de muestreo. Este retardo varía de acuerdo a la configuración del equipo utilizado, pues dependiendo del mismo, los cálculos y procesos, se ejecutan con mayor o menor rapidez.

Este tiempo de retardo está compuesto por dos tiempo:

$$T_{RETARDO} = T_{RETARDO}1 + T_{RETARDO}2$$

donde:

$T_{RETARDO}1$ Tiempo de retardo propio del computador para ejecutar los procesos de traslación y rotación. Depende de la configuración del equipo utilizado. Permite que el vehículo en pantalla ejecute los movimientos a la misma velocidad que en la realidad.

$T_{RETARDO}2$ Tiempo de retardo adicional que permite que señal de control del vehículo prototipo se mantenga el tiempo suficiente para cubrir la distancia real.

Para determinar el $T_{RETARDO}$ que se necesita en el movimiento de traslación se consideran las siguientes premisas:

a) El vehículo prototipo no posee velocidad inicial (Vo = 0).

b) El movimiento del vehículo prototipo depende de una señal constante, es decir que su aceleración es nula (a=0). Aplicando las Leyes de la Cinemática:

$$d = vt \Rightarrow v = \frac{d}{t}$$

donde:
- d Distancia recorrida por el vehículo prototipo
- v Velocidad del vehículo prototipo
- t Tiempo de traslación del vehículo prototipo

Luego, de los hechos reales se realiza un muestreo obteniendo los siguientes datos:

DISTANCIA (cm)	TIEMPO (seg.)	VELOCIDAD (cm/seg)
250	2.29	109
250	2.53	99
300	2.66	113
300	2.72	110
350	2.84	123
350	2.96	118
400	3.25	123
400	3.25	123
450	3.71	121
450	3.69	122
500	3.78	132

500	3.75	133
Velocidad promedio (cm/seg)		**64,417**

Aplicando el factor de conversión, la velocidad del vehículo en pantalla debe ser de:

Velocidad promedio (pixeles/seg)	**238**

Por ejemplo, si el vehículo recorre en pantalla 270 pixeles, es decir 135cm.:

$$t = \frac{d}{v} = \frac{135\ cm}{119\ \frac{cm}{seg}} = 1.13\ seg$$

Se establece que el vehículo debe recibir una señal desde el computador que lo mantenga en movimiento durante 1.13seg., sin importar la configuración del mismo, para lo cual se le aplicará el $T_{RETARDO}$ correspondiente, que se lo obtiene mediante el muestreo con varios retardos hasta seleccionar el más certero.

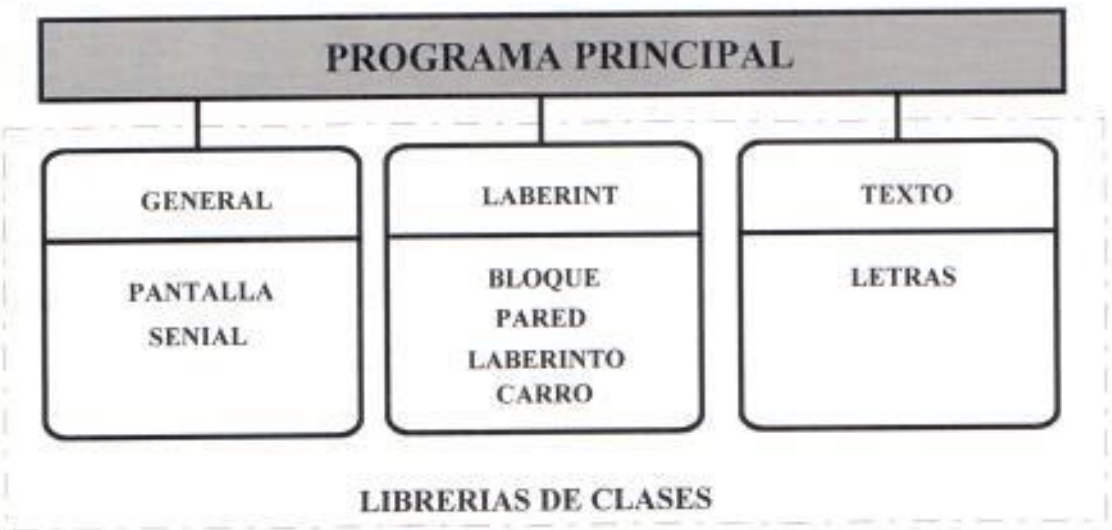
2.2 ESTRUCTURA DEL MODULO DE SOFTWARE

Como se indicó anteriormente, para el desarrollo del módulo de software se ha aplicado la programación orientada a objeto para la cual se han definido varias clases. Con el fin de estructurar el módulo se han agrupado las clases en librerías de acuerdo a sus características.

La aplicación del sistema de control se la realiza a través de un programa principal por medio del cual se accesa a las librerías de clases.

En la Fig. 2.25 es observan cada uno de los elementos que constituyen el Módulo de Software.

Fig. 2.26 Estructura del Módulo de Software



2.3 DESCRIPCION DE LOS ELEMENTOS DEL MODULO DE SOFTWARE

A continuación se describen de manera general cada uno de los elementos que constituyen el Módulo de Software, indicando su función y elementos que lo constituyen. Estos elementos serán descritos más detalladamente en el capítulo IV.

PROGRAMA PRINCIPAL.-

Contiene el cuerpo del sistema, que en el caso de este proyecto corresponde a la aplicación propuesta para el sistema de control. Es el que hace el llamado a las librerías propias de C++, incluyendo la librería de funciones gráficas. Además, es a través del programa principal que se accesa a las librerías creadas por el usuario que en este caso son: GENERAL, LABERINT y TEXTO.

LIBRERIA DE CLASES. GENERAL.-

Contiene las clases definidas con carácter de general puesto que se utilizan de manera independiente a la aplicación. La función de cada una es:

- Clase **PANTALLA**.- Definir el ambiente gráfico con el que se va a trabajar, identificando sus parámetros y determinando la pantalla de fondo.

- Clase **SENIAL**.- Establecer y enviar códigos de control al puerto paralelo del computador.

LIBRERIA DE CLASES. LABERINT.-

Contiene todas las clases creadas de manera exclusiva para el proyecto. Todos los gráficos utilizados son bidimensionales. La función de cada clase es la siguiente:

- Clase **BLOQUE**.- Definir y graficar un bloque de acuerdo al tamaño indicado.

- Clase **PARED**.- Creada a partir de la clase BLOQUE, hereda todos sus miembros. Tiene por función definir paredes (conjunto de bloques) horizontales y verticales, y graficarlas a través de la clase BLOQUE.

- Clase **LABERINTO**.- Creada a partir de la clase PARED, hereda todos sus miembros. Su función es definir laberintos tomando la información del archivo de texto "DATOS.TXT" para ser graficados a través de la clase PARED.

- Clase **CARRO**.- Tiene por función definir, graficar y controlar los movimientos de la imagen de un vehículo.

LIBRERIA DE CLASES. TEXTO.-

Ha sido creada para la definición y graficación de todo tipo de texto que no pueda ser obtenido a través de las librerías propias de C++. Para el presente proyecto contiene únicamente la clase LETRAS.

- Clase **LETRAS**.- Permite la definición y graficación de todas las letras del abecedario, aplicando una técnica avanzada de graficación con la que se obtiene un tipo de letra parametrizable en color, tamaño y fondo.

2.4 REQUERIMIENTOS DE OPERACION

El Módulo de Software ha sido diseñado para operar básicamente en una computadora compatible con IBM. La plataforma de operación será el Sistema Operativo DOS o compatible.

El equipo deberá tener por lo menos un procesador 80386 y contar con 2Mb. RAM; sin embargo, la respuesta del sistema es bastante lenta; por lo tanto se recomienda utilizar como mínimo un equipo con un procesador 80486 con 4Mb. RAM.

El sistema puede operar en un monitor monocromático, pero el uso de un monitor a color permite simular con mayor precisión los movimientos del vehículo prototipo, aún más si se trata de un monitor SVGA.

El número y complejidad de las operaciones que se realizan durante la ejecución del sistema no exigen la utilización de un coprocesador matemático.

Además, con el fin de poder enviar las señales de control hacia la interfaz digital/analógica, se requiere un puerto paralelo libre con salida de 25 pines.

CAPITULO III

MODULO DE ELECTRONICA

El Módulo de Electrónica está compuesto por todo el equipo requerido para el desarrollo del proyecto, desde el computador hasta la interfaz digital/analógica que se conecta al control remoto del vehículo prototipo. El presente capítulo muestra las técnicas aplicadas, los elementos utilizados y los diagramas de los circuitos de control.

3.1 ANALISIS DE TECNICAS APLICADAS

El objetivo básico del Módulo de Electrónica es el de conectar el computador con el control remoto del vehículo prototipo para adaptar las señales de comunicación que permitan generar las señales de radio que controlan el vehículo prototipo. Para tal efecto, ha sido necesario el diseño de una interfaz digital/analógica entre ambos dispositivos: computador y control remoto.

A continuación se procederá a analizar más detalladamente los dispositivos y elementos que han sido utilizados en el Módulo de Electrónica.

PUERTO PARALELO DEL COMPUTADOR

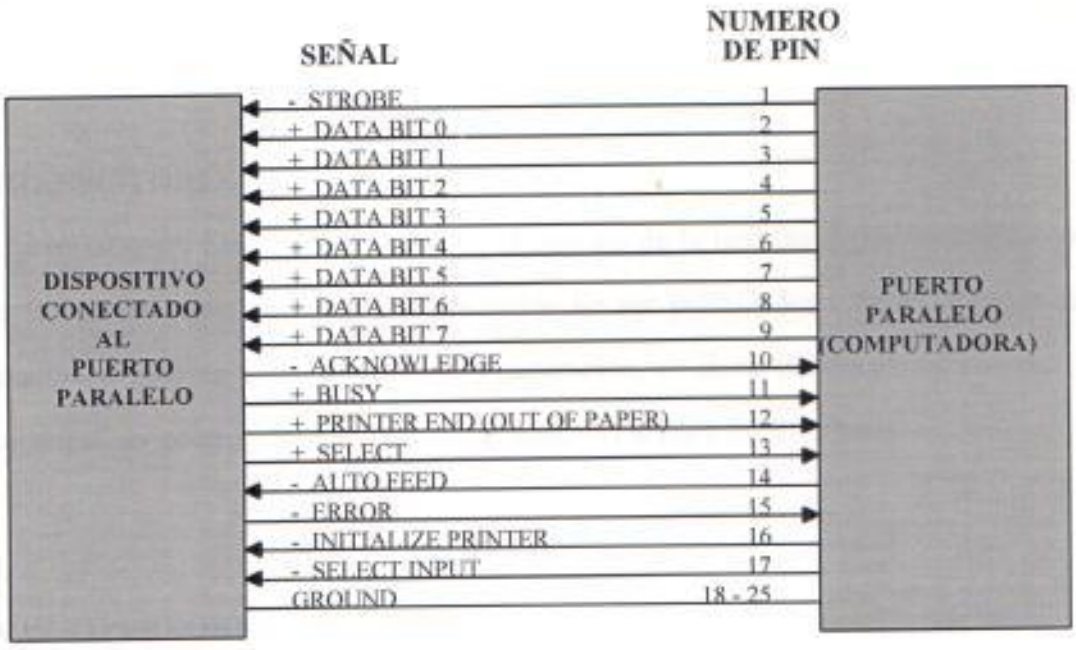
Para el envío de señales desde el computador a la interfaz digital/analógica se ha utilizado el puerto paralelo, puesto que éste presenta ocho líneas para enviar todos los bits de un byte de datos simultáneamente, permitiendo optimizar el control y comunicación entre los dispositivos. Además, esta interfaz es rápida y usualmente es reservada para impresoras y comunicaciones, como en el presente caso.

El puerto paralelo de un computador personal es unidireccional, tal que los datos viajan solo en una vía; ésto es, la información generada por el computador fluye al puerto paralelo y de allí al dispositivo conectado al él. Los bytes de información se envían de uno en uno, donde cada byte indica una acción a realizar por el vehículo prototipo, la cual se refleja en pantalla.

Es importante resaltar en este caso, que el cable utilizado para conectar la interfaz digital/analógica al computador no debe ser muy extenso, puesto que se podrían generar errores, salvo el caso que se amplifique la señal.

En la fig. 3.1 se puede observar la conexión requerida para poder enviar la información a través del puerto paralelo. Los pines 2 al 9 son utilizados para transmitir los ocho bits del byte que contiene la señal del movimiento que se desea realizar. Las equivalencias de las señales transmitidas y los movimientos correspondientes se pueden observar en el capítulo 2.1.4..

Fig. 3.1. Especificaciones del conector de 25 pines del puerto paralelo



BUFFERS

Los buffers están dispuestos en el circuito para acondicionar la señal TTL 5V a un nivel apropiado de voltaje para operar otros componentes electrónicos, por ejemplo transistores. Al mismo tiempo, estos dispositivos mantienen la señal recibida por un lapso hasta que recepta una nueva señal.

DIODOS LEDs

Los diodos LEDs se han dispuesto en el circuito con el fin de poder detectar las señales que son enviadas desde el computador a la interfaz digital/analógica e identificar los movimientos que debe ejecutar el vehículo prototipo de acuerdo a un código preestablecido en el punto 2.1.4. Además, a través de ellos se puede conocer

el estado de las señales que entran a los multiplexores puesto que los LEDs están conectados a los mismos puntos.

TRANSISTORES

Los transistores han sido utilizados en el circuito de la interfaz digital/analógica con el fin de conducir las señales al momento de ser polarizados. Estos dispositivos funcionan solo en estados de corte y saturación, es decir conducen de colector a emisor al ser polarizados con un voltaje positivo mayor a 0.7V en base.

MULTIPLEXORES (MUX)

Los multiplexores son dispositivos lógicos utilizados para tomar varias señales (que son las que llevan la información de control de los movimientos del vehiculo prototipo), y convertirlas en una sola señal que servirá para activar otros dispositivos del circuito de la interfaz digital/analógica. Cada multiplexor decodifica una de las señales de control (avance, retroceso, giro a la derecha, giro a la izquierda) de acuerdo a las señales obtenidas por la interfaz digital/analógica que proviene del computador.

RELES

Los relés utilizados en el presente proyecto son bobinas con dos contactos que se activan al momento de recibir una señal de control. Su función es aislar el circuito de decodificación con el circuito de control remoto

3.2 DIAGRAMA DE BLOQUES DEL SISTEMA DE CONTROL

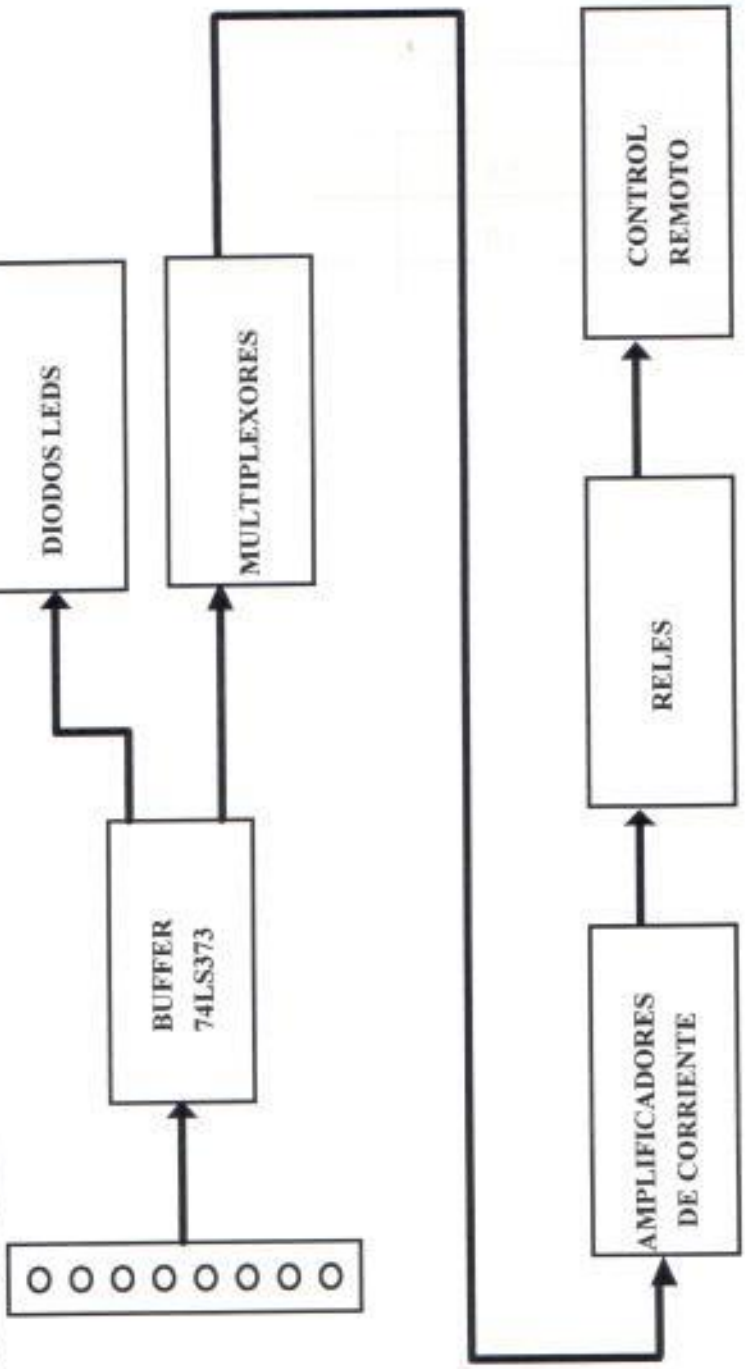


Fig. 3.2. Diagrama de bloques del Sistema de Control

3.3 EQUIPO UTILIZADO

A continuación se presenta el equipo utilizado para el diseño e implantación del Módulo de Electrónica:

COMPONENTE	# DE IC	UNIDAD
Resistencia (15 KΩ)	R1	11
Resistencia (470 Ω)	R2	8
Resistencia (390 Ω)	R3	4
Resistencia (68 Ω)	R4	4
Puertas AND	74LS08	1
Buffer	74LS373	1
Buffer	4050B	2
Buffer	74LS244	1
Multiplexor 1-16	74150	4
Inversor	74LS04	1
Transistor NPN equivalente tipo 2N222		4
Relé	A8709	4
Led's		8

Además, también se requieren los siguientes elementos:

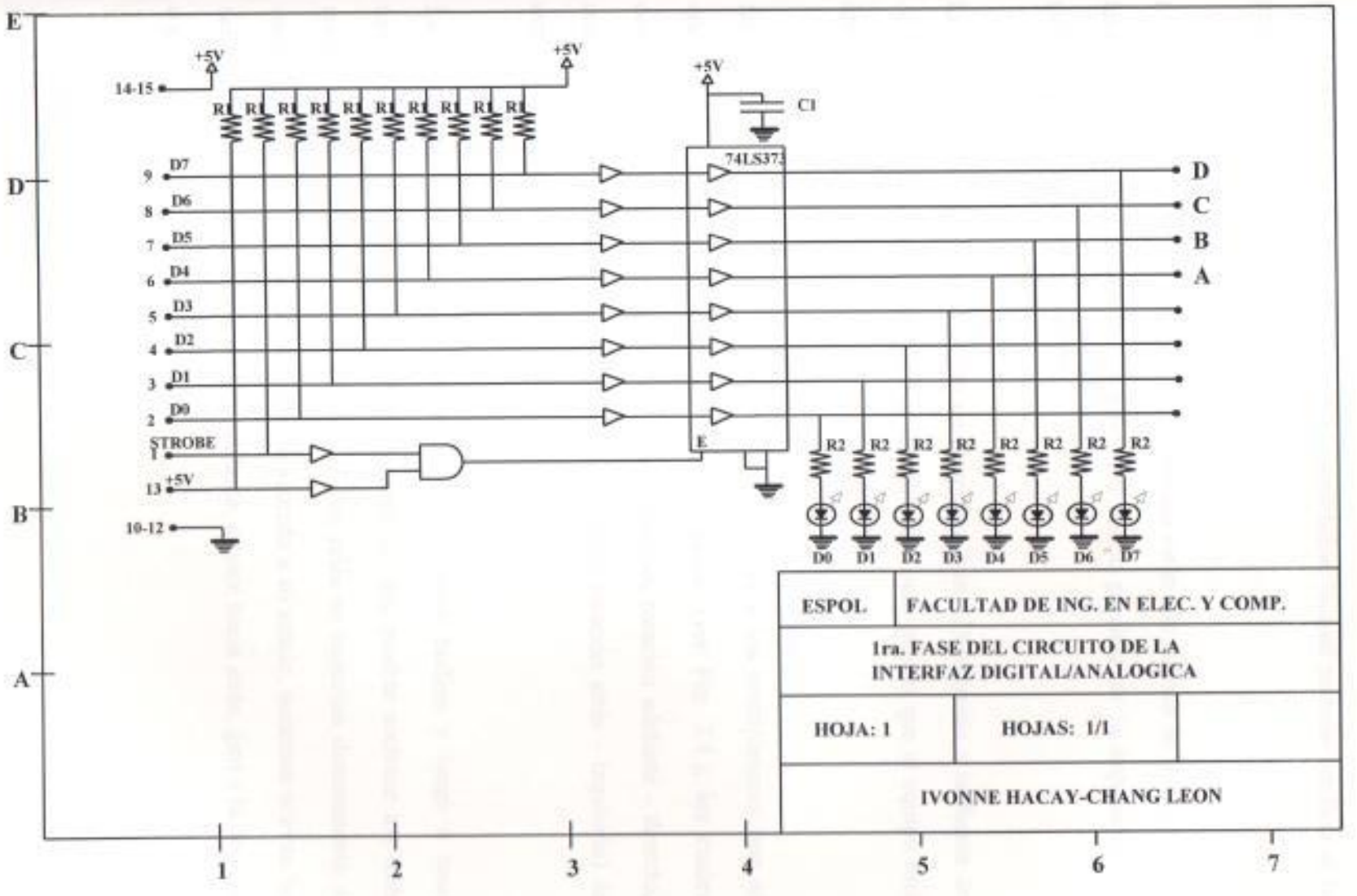
- Protoboard HB-1210
- Protoboard PB-103
- Cable de 25 hilos
- Fuente de Poder 5V
- Multímetro

3.4 CIRCUITOS DE CONTROL

A continuación se detalla el funcionamiento del circuito de la interfaz digital/analógica.

La Fig. 3.3. muestra el esquema de la primera fase de la interfaz digital/analógica, la cual capta las señales que envía el computador al control remoto del vehículo prototipo. El circuito usa todas las ocho líneas de salida de datos desde el puerto paralelo del computador personal para programar un 74LS373 a través de un 4050B.

Los buffer son el punto focal de la interfaz. Cuando la entrada de habilitación del 74LS373 está en baja, se cargan los datos desde los pines de entrada; y cuando está en alta, la información es enviada a la salida. Esta entrada de habilitación es controlada por la línea de salida STROBE (pin 1) del puerto paralelo.


E
D
C
B
A
1
2
3
4
5
6
7
+5V
14-15
R1
C1
74LS373
D7
D6
D5
D4
D3
D2
D1
D0
STROBE
10-12
E
R2
D
C
B
A
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
1ra. FASE DEL CIRCUITO DE LA
INTERFAZ DIGITAL/ANALOGICA
HOJA: 1
HOJAS: 1/1
IVONNE HACAY-CHANG LEON

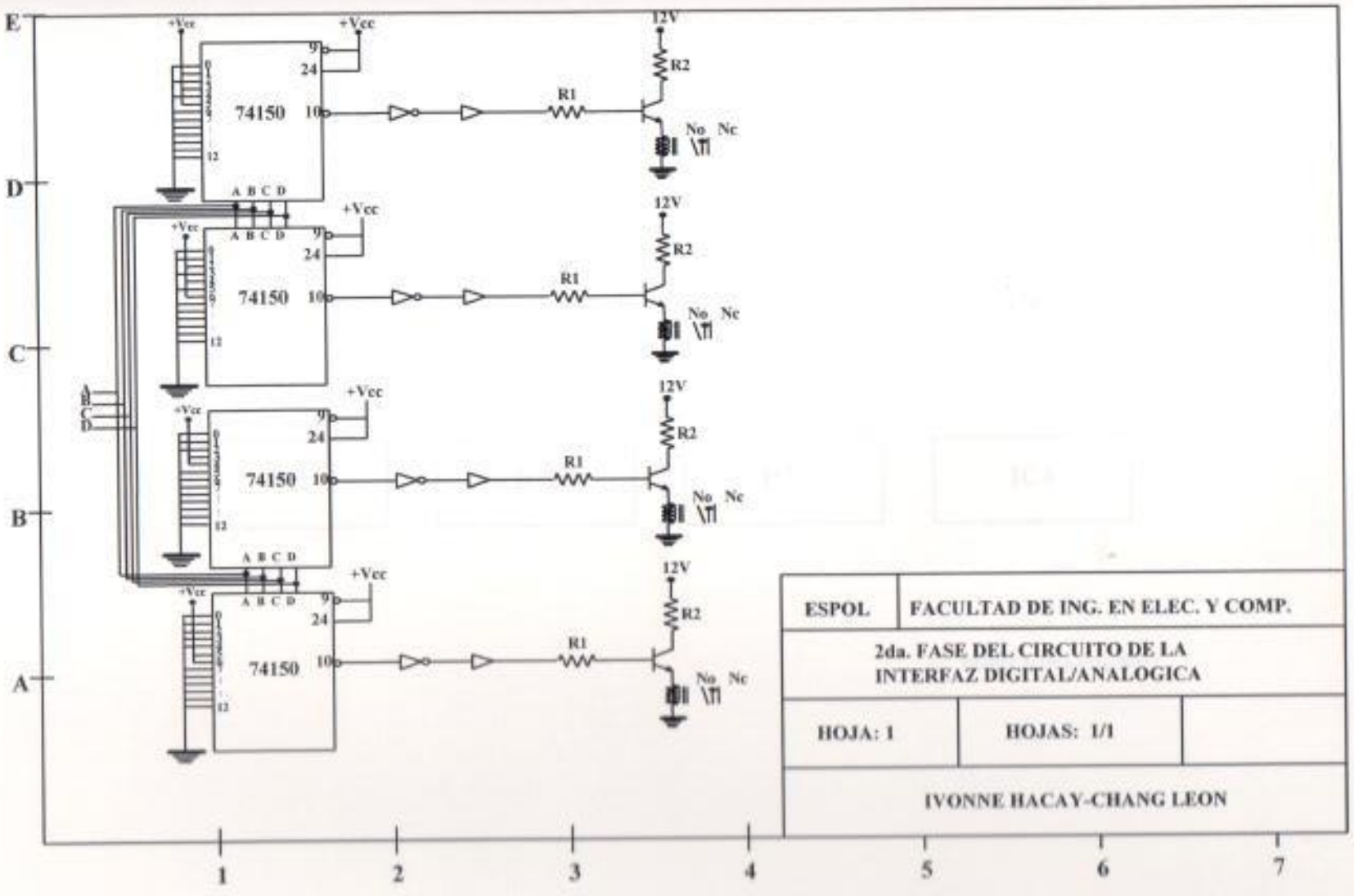
Hay un LED conectado a cada salida del controlador, lo cual permite verificar si la programación está correcta.

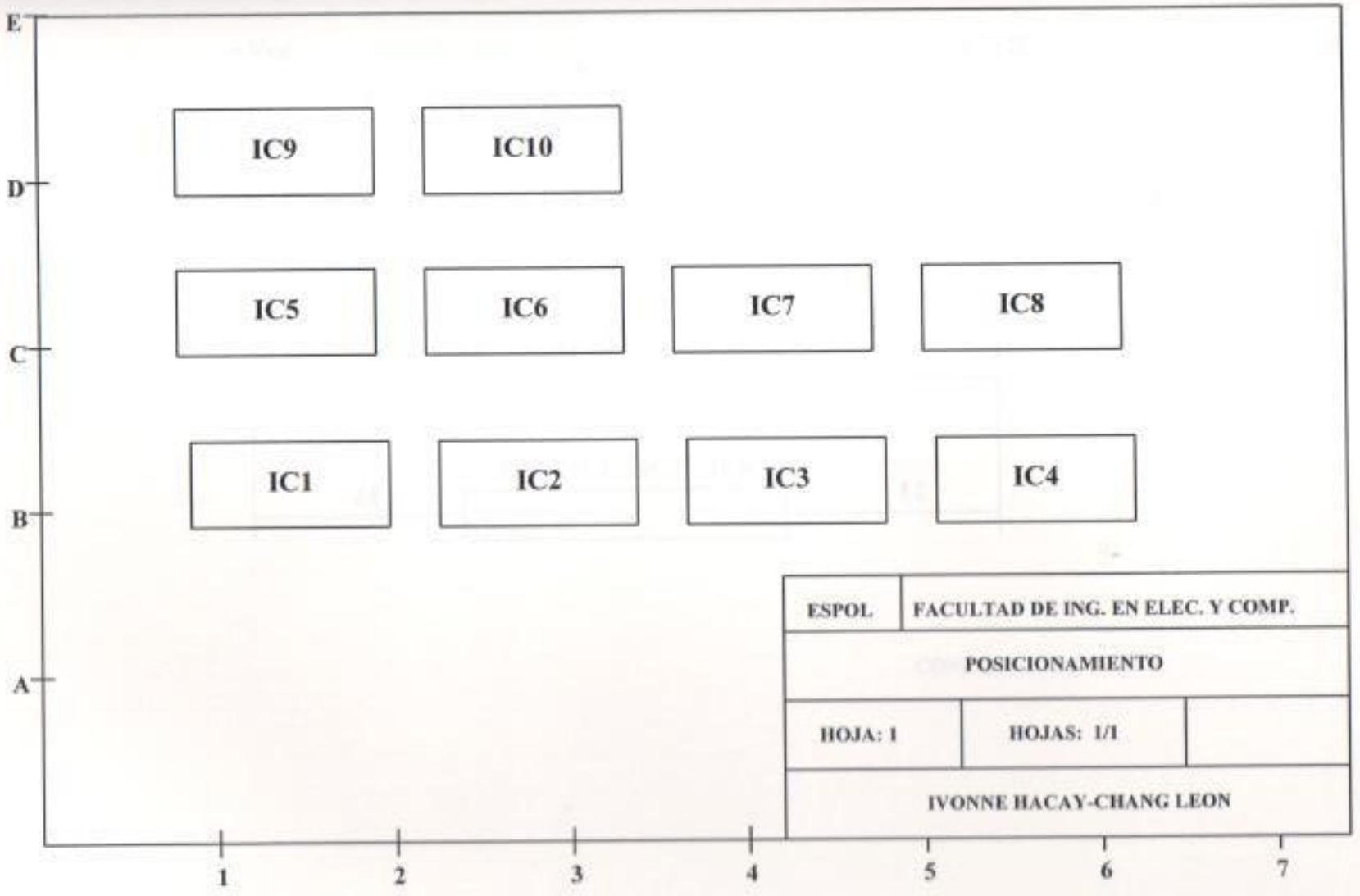
Los niveles de voltaje TTL con los que se trabaja están por encima de 5 voltios, por tanto los voltajes que salen de los pines del puerto paralelo de la impresora son suficientes para manejar los buffers.

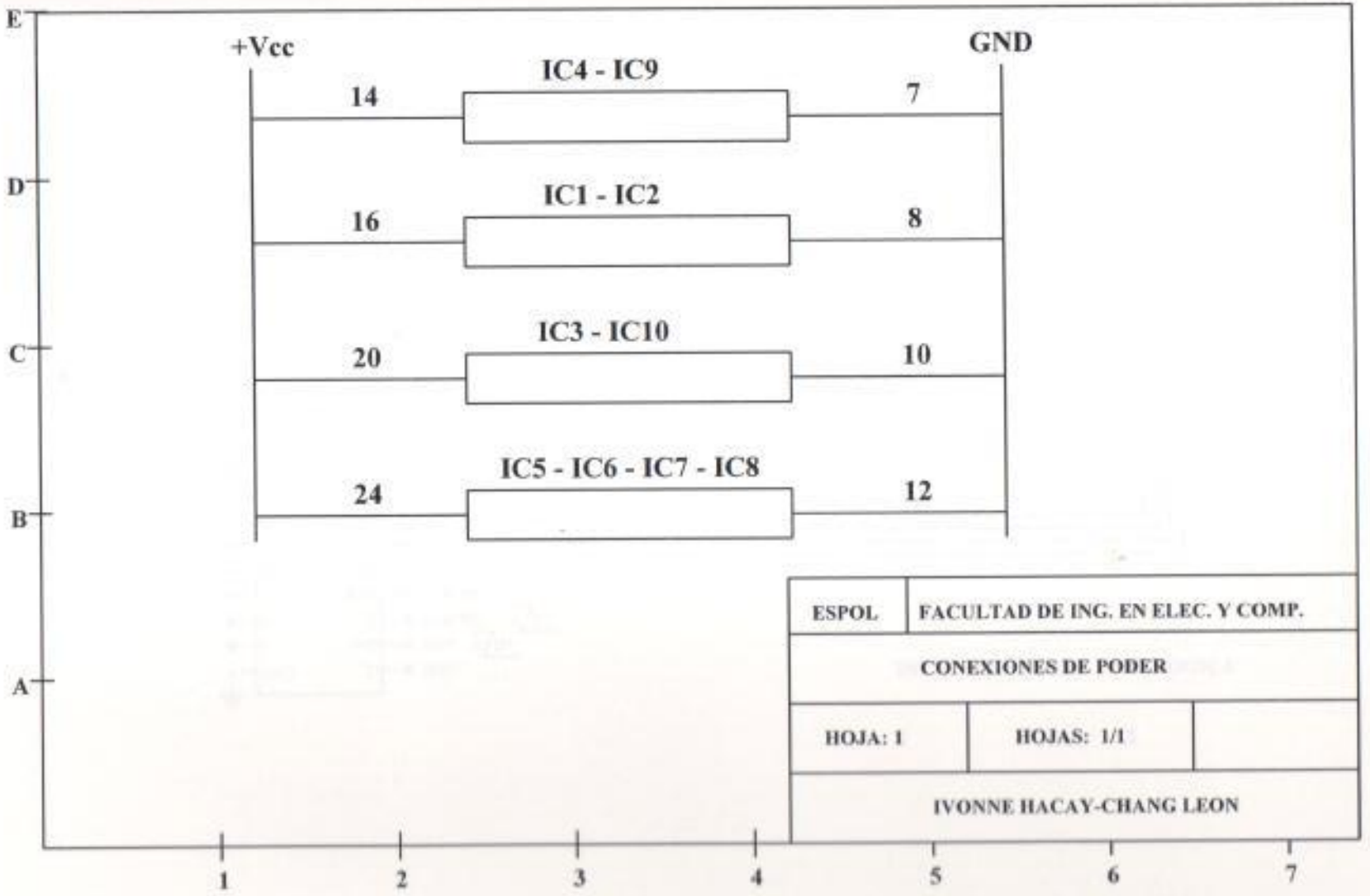
La entrada de SELECT en el puerto de impresora (pin 13) es unido a la fuente de +5V, sirviendo como una señal para la computadora para indicar que el equipo está encendido.

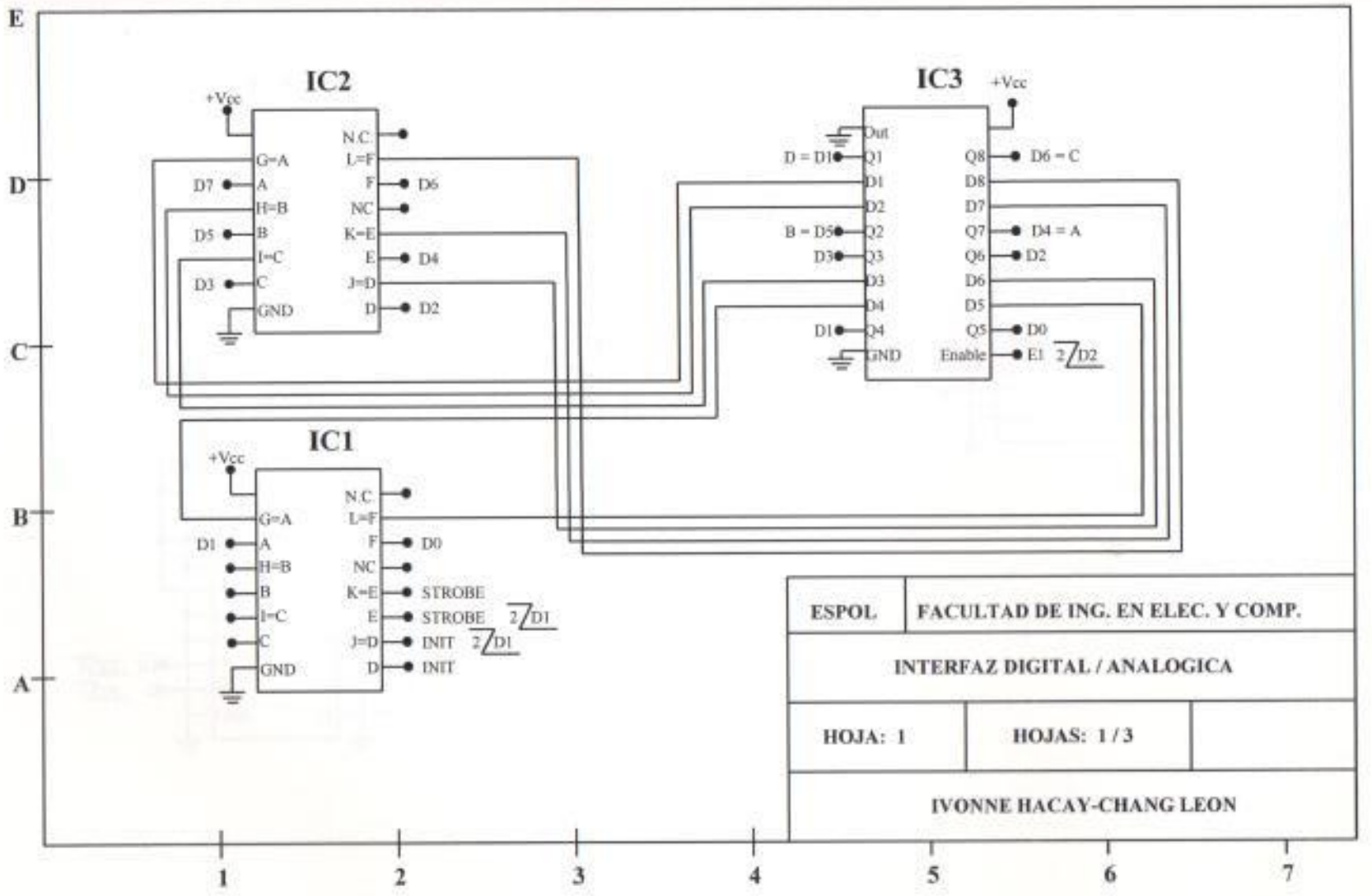
De este circuito se generan las señales que ingresan a los multiplexores que se encuentran ubicados en la segunda fase del circuito (ver Fig. 3.4.), los cuales decodifican las señales de control (avance, retroceso, rotación adelante - derecha, rotación atrás - derecha, rotación adelante-izquierda, rotación atrás - izquierda) de acuerdo a un código preestablecido (ver Fig. 2.25.).

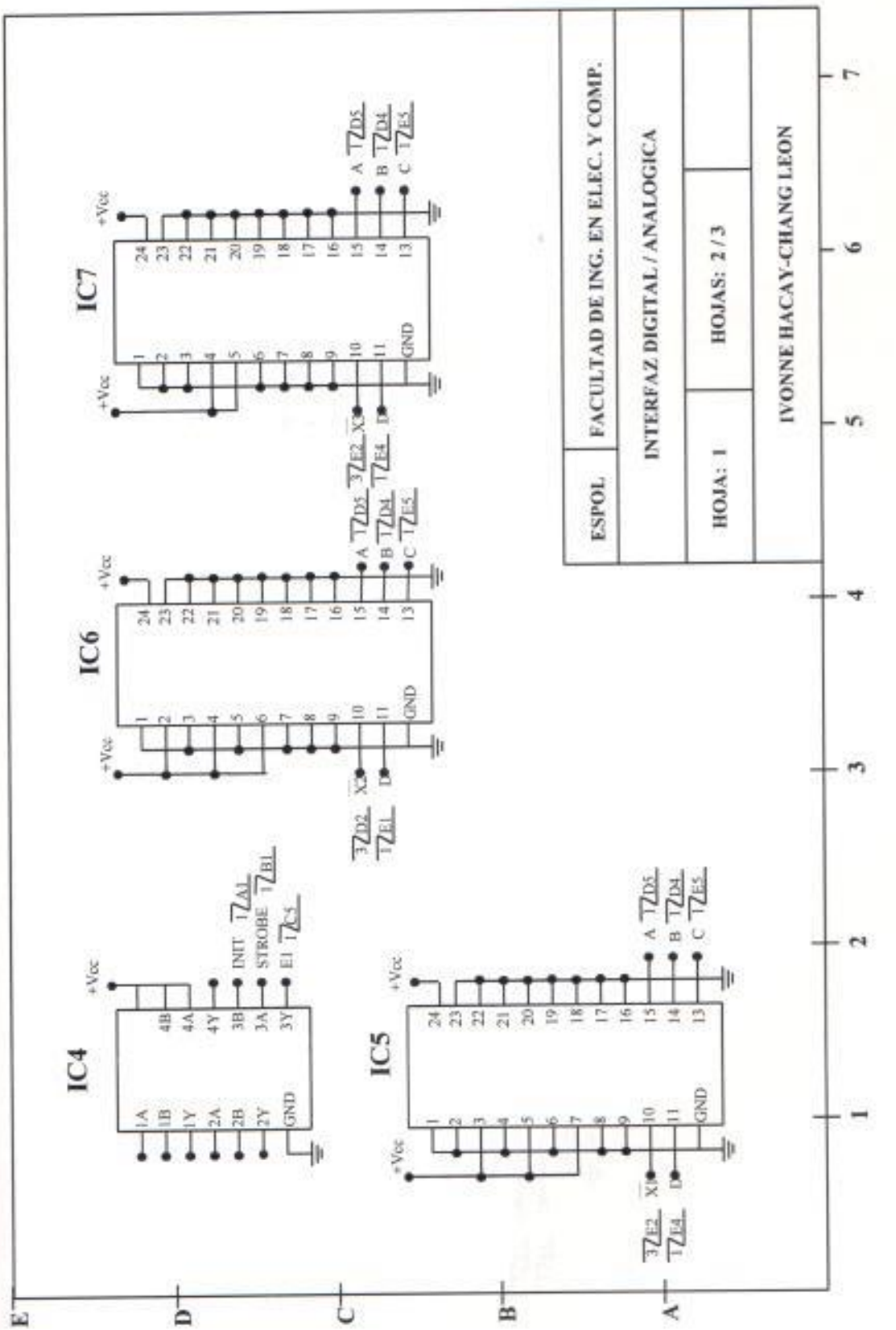
La señal generada por los multiplexores entra a unos buffers y luego a unos transistores que amplifican la señal, de manera que sea posible accionar los relés respectivos al momento que se saturan. Estos relés se conectan directamente al control remoto del vehículo prototipo y de acuerdo a su estado, accionan activan las señales de movimientos: avance hacia adelante, avance hacia atrás, giro a la derecha o giro a la izquierda.

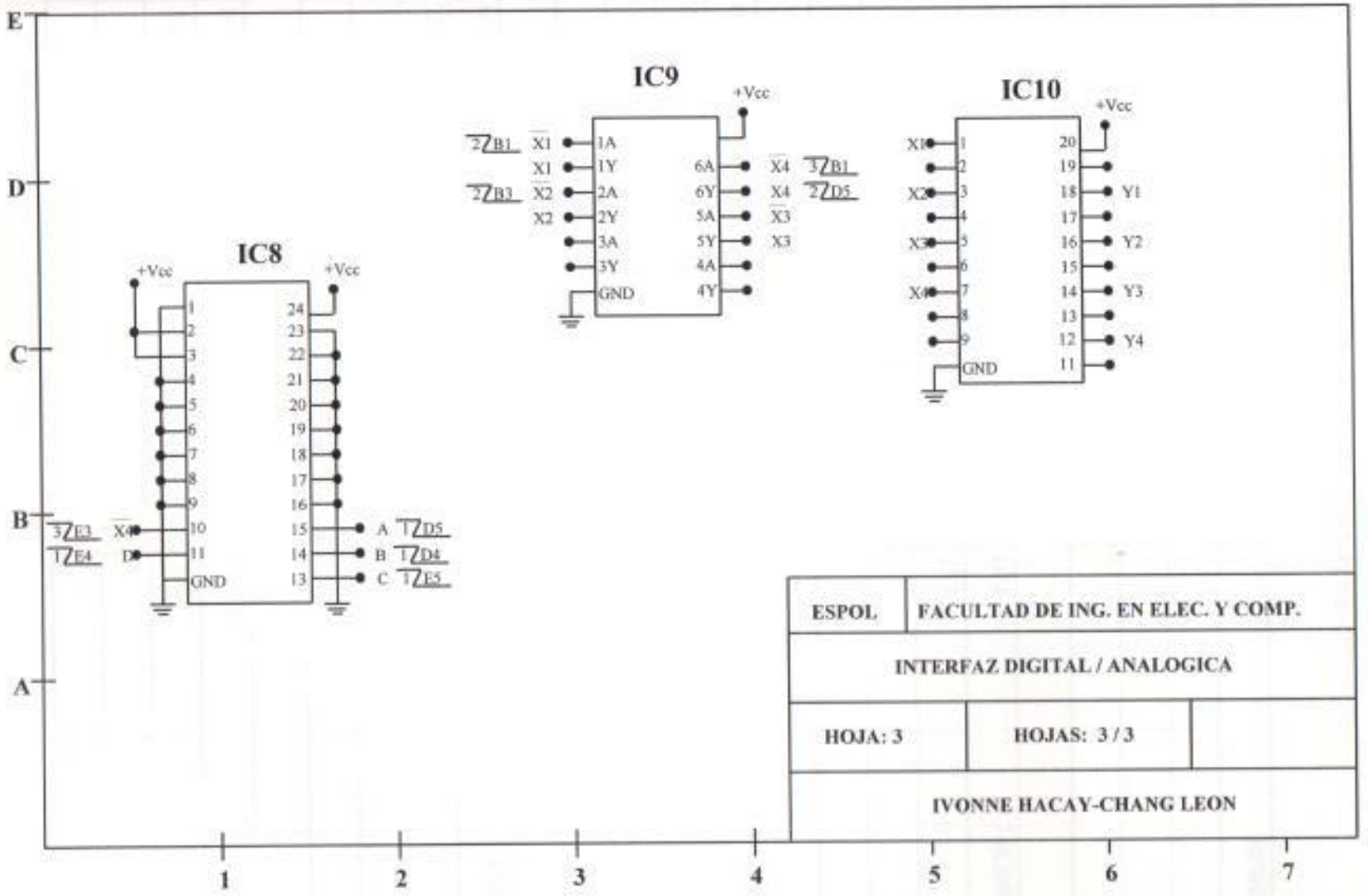

E
D
C
B
A
1
2
3
4
5
6
7
+Vcc
74150
A B C D
R1
R2
12V
No
Nc
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
2da. FASE DEL CIRCUITO DE LA INTERFAZ DIGITAL/ANALOGICA
HOJA: 1
HOJAS: 1/1
IVONNE HACAY-CHANG LEON


E
D
C
B
A
IC9
IC10
IC5
IC6
IC7
IC8
IC1
IC2
IC3
IC4
1
2
3
4
5
6
7
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
POSICIONAMIENTO
HOJA: 1
HOJAS: 1/1
IVONNE HACAY-CHANG LEON


E
D
C
B
A
+Vcc
GND
IC4 - IC9
14
7
IC1 - IC2
16
8
IC3 - IC10
20
10
IC5 - IC6 - IC7 - IC8
24
12
1
2
3
4
5
6
7
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
CONEXIONES DE PODER
HOJA: 1
HOJAS: 1/1
IVONNE HACAY-CHANG LEON


E
D
C
B
A
1
2
3
4
5
6
7
IC2
+Vcc
G=A
D7
A
H=B
D5
B
I=C
D3
C
GND
N.C.
L=F
F
D6
NC
K=E
E
D4
J=D
D
D2
IC3
+Vcc
Out
D = D1
Q1
D1
D2
B = D5
Q2
D3
Q3
D3
D4
D1
Q4
GND
Q8
D6 = C
D8
D7
Q7
D4 = A
Q6
D2
D6
D5
Q5
D0
Enable
E1
2
D2
IC1
+Vcc
G=A
D1
A
H=B
B
I=C
C
GND
N.C.
L=F
F
D0
NC
K=E
STROBE
E
STROBE
2
D1
J=D
INIT
2
D1
D
INIT
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
INTERFAZ DIGITAL / ANALOGICA
HOJA: 1
HOJAS: 1 / 3
IVONNE HACAY-CHANG LEON


IC4
IC5
IC6
IC7
+Vcc
GND
INIT
STROBE
E1
A
B
C
D
E
1
2
3
4
5
6
7
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
INTERFAZ DIGITAL / ANALOGICA
HOJA: 1
HOJAS: 2 / 3
IVONNE HACAY-CHANG LEON


E
D
C
B
A
1
2
3
4
5
6
7
IC8
+Vcc
+Vcc
GND
3/E3
7/E4
A 7/D5
B 7/D4
C 7/E5
IC9
+Vcc
2/B1
2/B3
3/B1
2/D5
1A
1Y
2A
2Y
3A
3Y
GND
6A
6Y
5A
5Y
4A
4Y
IC10
+Vcc
GND
Y1
Y2
Y3
Y4
ESPOL
FACULTAD DE ING. EN ELEC. Y COMP.
INTERFAZ DIGITAL / ANALOGICA
HOJA: 3
HOJAS: 3 / 3
IVONNE HACAY-CHANG LEON

LISTA DE STUFF SHEET

IC1	4050B
1B1	

IC2	4050B
1E1	

IC3	74LS373
1E4	

IC4	74LS08
2E1	

IC5	74150
2C1	

IC6	74150
2E3	

IC7	74150
2E5	

IC8	74150
3D1	

IC9	74LS04
3E3	

IC10	74LS244
3E5	

3.5 CONDICIONES DE IMPLANTACION

Las señales conectadas al computador a través del puerto paralelo deben tener un estado inicial previo a la recepción de los códigos de datos y son las siguientes:

SEÑAL	ESTADO
STROBE	-
DATA BIT 1	+
DATA BIT 2	+
DATA BIT 3	+
DATA BIT 4	+
DATA BIT 5	+
DATA BIT 6	+
DATA BIT 7	+
ACKNOWLEDGE	-
BUSY	-
PRINTER END	-
SELECT	+
AUTOFEED	+
ERROR	+
INITIALIZE PRINTER	+
SELECT INPUT	-
OTHERS	TIERRA

Luego, para poner en funcionamiento el equipo se deben ejecutar los siguientes pasos:

1. Conectar la interfaz digital/analógica al puerto paralelo del computador.

2. Encender el computador.

3. Energizar la fuente de poder, la cual debe generar por lo menos 5V. También es posible utilizar en su lugar un juego de baterías que provean de 6V.

4. Energizar los protoboards.

5. Verificar previamente que el vehículo prototipo tenga instaladas las baterías necesarias, así como también su control remoto. Encender el vehículo prototipo y verificar la antena del control remoto.

6. Ejecutar el programa desarrollado en el Módulo de Software.

CAPITULO IV

APLICACION DEL SISTEMA DE CONTROL

Una vez analizado el sistema de control del vehículo prototipo y las técnicas aplicadas para la visualización del mismo, se propone una aplicación del sistema ubicando un vehículo dentro de un laberinto donde deberá encontrar la salida. En el presente capítulo se analiza el algoritmo planteado, la integración de la aplicación y las pruebas efectuadas.

4.1 PROPUESTA DEL ALGORITMO A RESOLVER

Considerar el caso de un vehículo prototipo cuyos movimientos se controlan a través de señales de radio y que se encuentra dentro de un laberinto del cual tiene que salir.

Se desea controlar el vehículo a través de una computadora compatible con IBM, diseñando una interfaz digital/analógica que genere las señales de radio y donde la comunicación sea unidireccional (de la computadora al vehículo).

Además se quiere visualizar en pantalla al vehículo dentro del laberinto y hacerle seguimiento a todos sus movimientos.

Deberán existir cuatro diferentes opciones de laberinto (en la computadora), cuyos tamaños a escala sean iguales a seis pantallas de computadora. Sus vías tendrán un sólo carril, tal que el vehículo esté imposibilitado de variar de dirección sin salirse de la vía.

El laberinto podrá ser resuelto ya sea en forma manual o automática. En forma manual, será el operador de la computadora quien guíe al vehículo hasta la salida; luego de lo cual, se le tomará su tiempo de resolución y se mostrará una lista de los mejores tiempos de resolución para cada laberinto. En forma automática, el vehículo será el que encuentre la salida aplicando un algoritmo de resolución.

4.2 ANALISIS Y DISEÑO DEL ALGORITMO DE SOLUCION DEL PROBLEMA PLANTEADO

4.2.1 ANTECEDENTES

Se dispone de un sistema de control por computadora de un vehículo prototipo a través de señales de radio, cuyos movimientos pueden ser monitoreados en la pantalla, el cual está compuesto por un Módulo de Software y un Módulo de Electrónica, presentados en los capítulos II y III respectivamente.

A través del Módulo de Software es posible:

- Visualizar al vehículo prototipo y controlar en pantalla los movimientos de su imagen.

- Generar y graficar un laberinto partiendo de una codificación.
- Generar una codificación que se envía al puerto paralelo.

A través del Módulo de Electrónica se cuenta con:

- Un circuito de una interfaz digital/analógica que toma información del puerto paralelo y la transforma en niveles de voltaje.
- Un circuito conectado a la interfaz digital/analógica que toma la información generada por los niveles de voltaje y la traduce en señales de radio que envía al vehículo prototipo.

4.2.2 ANALISIS

Para poder resolver el algoritmo planteado y partiendo del Sistema de Control por Computadora del Vehículo Prototipo se seguirán los pasos que se presentan a continuación:

a) Diseño de los laberintos
b) Codificación de los laberintos
c) Ingreso de la codificación al archivo DATOS.TXT
d) Posicionamiento del vehículo en el laberinto
e) Condición de movimientos

4.2.2.1 DISEÑO DE LOS LABERINTOS

Para diseñar los laberintos es necesario tomar en cuenta las siguientes consideraciones:

- Cada laberinto estará compuesto de 6 secciones, cada una de ellas del tamaño de una pantalla del computador (ver Fig. 2.6.).

- De acuerdo al Módulo de Software desarrollado, se utilizarán bloques de 30x30 pixeles y paredes de 4 bloques; por tanto, cada sección tendrá un máximo de 7 paredes horizontales por 5 paredes verticales (ver Fig. 2.7.).

- Cada sección tendrá por lo menos una vía de conexión con otra sección, y en una de ellas debe existir una salida del laberinto.

- Las vías de conexión deben permitir ubicar al vehículo sin que exista ninguna pared que obstruya su circulación inicial y/o que exista el mínimo de espacio libre para que pueda efectuar un giro. El espacio libre debe ser mayor al ancho del vehículo. En la Fig. 4..1. se muestra un diseño correcto de las vías de conexión en una sección de laberinto donde por cualquiera de las dos vías el vehículo tiene libre circulación, en cambio en la Fig. 4.2. se observa en cada vía que frente al vehículo existe una pared, donde el espacio entre ellos es menor al ancho del vehículo, lo cual le impide realizar un giro y continuar su circulación.

Fig. 4.1. Diseño Correcto de las Vías de Conexión en una Sección de Laberinto

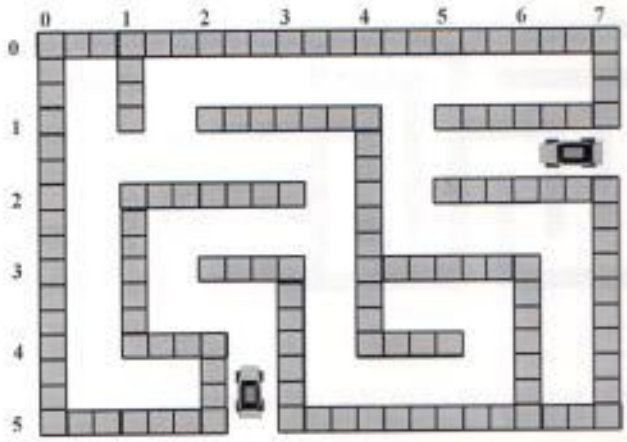
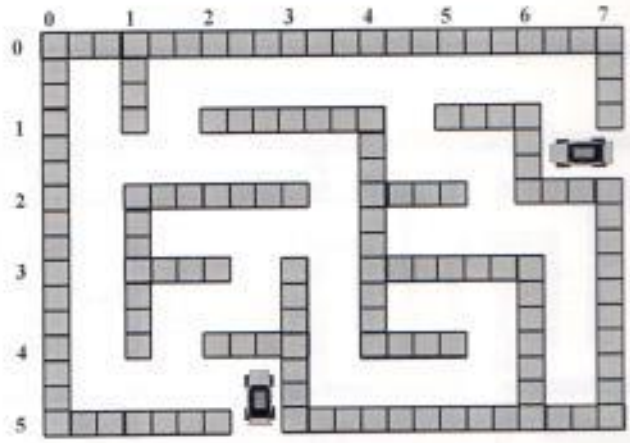
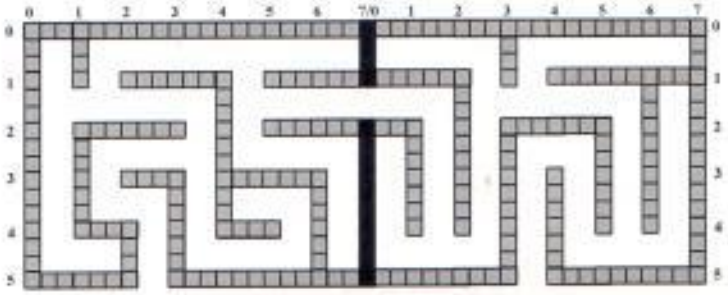


Fig. 4.2. Diseño Incorrecto de las Vías de Conexión en una Sección de Laberinto



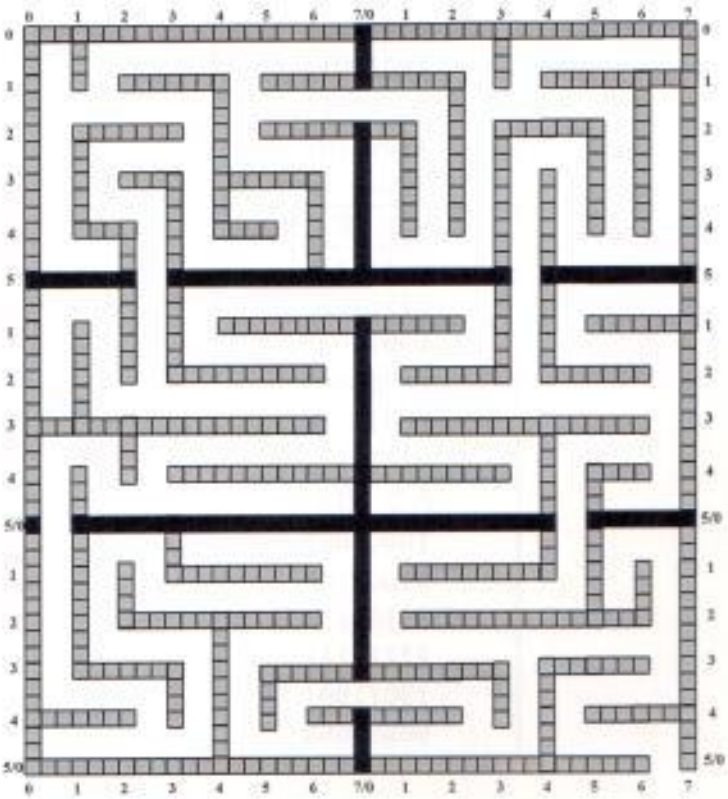
- El diseño de las paredes horizontales y verticales que tienen conexión con otra sección deberán tener el mismo diseño. Por ejemplo, asumiendo que la sección de laberinto que se muestra en la Fig. 4.1. corresponde a la sección 0, entonces el diseño de las paredes verticales de la columna 7 deberá ser el mismo para las paredes verticales de la columna 0 de la sección 1 (ver Fig. 4.3.).

Fig. 4.3. Diseño de dos Secciones de Laberinto y su Conexión



En la Fig. 4.4 se muestra el diseño de un laberinto completo, donde se observa la aplicación de todas las consideraciones mencionadas. Las paredes de conexión se indican en un color más oscuro.

Fig. 4.4. Diseño de un Laberinto



4.2.2.2 CODIFICACION DE LOS LABERINTOS

Una vez diseñado el laberinto, se procede a codificarlo siguiendo el esquema presentado en el punto 2.1.2 respecto a su graficación, por tanto, codificando el laberinto de la Fig. 4.4. resulta:

Fig. 4.5. Codificación del Laberinto de la Fig. 4.4.

LAB1_0_H
11111110
00110110
01100110
00101100
01001000
11011110
LAB1_0_V
11000001
10001000
11001001
11011011
10110011
00000000
LAB1_1_H
11111110
11001110
10011000
00000000
00000000
11101110
LAB1_1_V
10010001
00100011
11110111
11111111
10011001
00000000

Fig. 4.5. Codificación del Laberinto de la Fig. 4.4. (continuación)

LAB1_2_H
11011110
00001110
00011100
11111100
00011110
01111110
LAB1_2_V
10110000
11110001
11000001
10100001
11000001
00000000
LAB1_3_H
11101110
11000110
01101100
01111100
11100100
11110110
LAB1_3_V
00011001
10011001
10000001
10001001
10001101
00000000
LAB1_4_H
01111110
00011100
00111100
01100110
11000010
11111111

Fig. 4.5. Codificación del Laberinto de la Fig. 4.4. (continuación)

LAB1_4_V
11010001
11100001
11001001
10011100
10001001
00000000
LAB1_5_H
11110110
01110000
01111100
11101100
11000110
11111100
LAB1_5_V
10001101
10000111
10000001
00011001
10001001
00000000

4.2.2.3 INGRESO DE LA CODIFICACION AL ARCHIVO DATOS.TXT

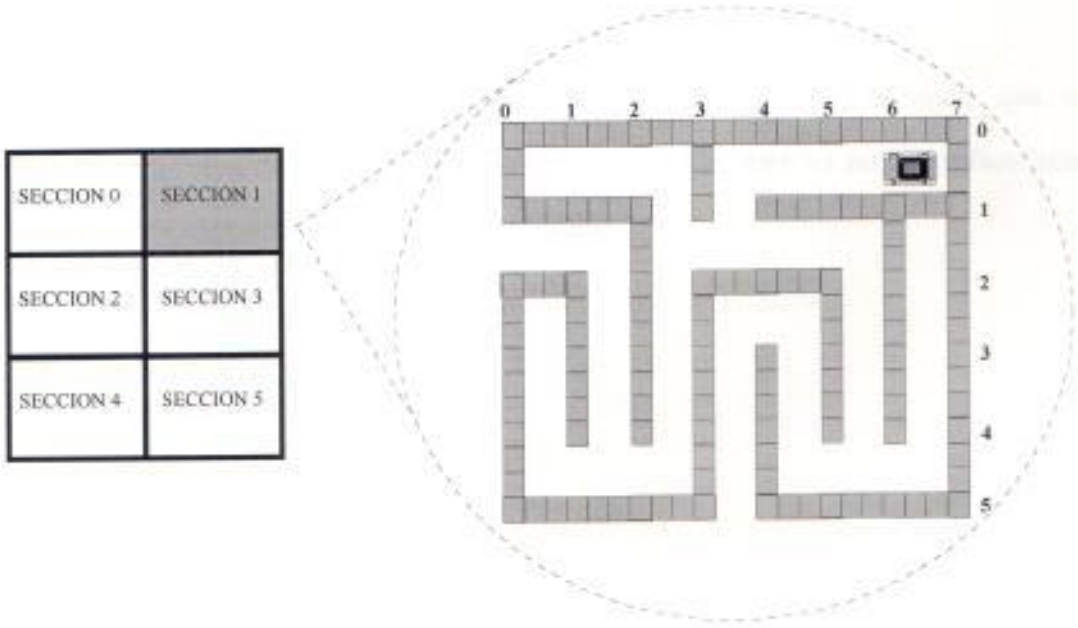
La codificación que se presenta en la Fig. 4.5. corresponde al Laberinto 1 y debe ser tipeada, sin dejar líneas libres, en un archivo tipo texto llamado DATOS.TXT. La codificación de los demás laberintos que se requieren diseñar para el proyecto será ingresada a continuación de la del Laberinto 1.

4.2.2.4 POSICIONAMIENTO DEL VEHICULO EN EL LABERINTO

Habiendo diseñado y codificado el laberinto, se procede a determinar la posición donde se ubicará al vehículo en el laberinto, la está sujeta a las siguientes consideraciones:

- El vehículo puede ser ubicado en cualquiera de las secciones del laberinto.
- El vehículo tiene que ser posicionado al centro del carril respecto a sus paredes laterales.
- En caso que el primer movimiento sea un giro, el vehiculo debe tener el espacio suficiente para poder realizarlo.

Fig. 4.6. Ejemplo de Posicionamiento del Vehiculo en el Laberinto



4.2.2.5 CONDICION DE MOVIMIENTOS

Los movimientos del vehículo, en cualquiera de los dos tipos de resolución: manual o automática, está sujeta a algunas consideraciones como son:

- El avance o retroceso del vehículo se efectua hasta que éste se topa con una pared o un cruce de vías. El vehículo se detiene unos pixeles (valor establecido) antes de llegar al tope o al cruce, de tal manera que pueda contar con el espacio suficiente para poder efectuar un giro.

- Si el vehículo se encuentra con una vía sin salida, entonces tendrá que retroceder para tomar una nueva ruta.

- Cuando el vehículo pasa de una sección a otra, la nueva sección es graficada en pantalla de manera automática, y el vehículo ubicado al inicio de esa pantalla en la misma dirección con la que ingresó a esa sección.

- Para indentificar que el vehículo ha resuelto el laberinto, es decir que se encuentra fuera de él, es que al pasar a otra sección, ésta ya no se grafica, sino que automáticamente se termina el programa.

4.2.3 TIPOS DE RESOLUCION

Como se indicó previamente, el laberinto debe ser resuelto por el vehículo en dos modalidades: manual o automática. En ambos casos, se requieren aplicar todas las consideraciones mencionadas en el análisis.

4.2.3.1 RESOLUCION MANUAL

En la resolución manual, es el operador del computador quien dirigirá al vehículo hasta encontrar la salida del laberinto seleccionado, para ésto se requiere asignar una tecla o una combinación de teclas a cada uno de los posibles movimientos que pueda efectuar el vehículo prototipo, en cada una de las direcciones en que se encuentre.

Con el objetivo de controlar los movimientos del vehículo de manera más clara y sencilla, únicamente se han utilizado las cuatro teclas de cursor, y adicionalemente la tecla Ctrl (Control) para lograr un mayor número de combinaciones. En la Fig. 4.7. se indican las teclas seleccionadas para simular los movimientos del vehículo prototipo en cualquiera de las direcciones posibles; además, se indica el código ASCII que le corresponde puesto que a través de este valor se podrá identificar la tecla o combinación de teclas presionada.

Fig. 4.7. Código ASCII de las teclas de control

TECLA	CODIGO ASCII
→	77
←	75
↑	72
↓	80
Ctrl + →	116
Ctrl + ←	115
Ctrl + ↑	141
Ctrl + ↓	145

Los movimientos del vehículo son identificados ubicándose en la posición de conductor, siendo indiferente la dirección que tenga el vehículo, es decir que al indicar un giro hacia la derecha, ya sea hacia adelante o hacia atrás, se refiere a la derecha del conductor, de igual manera se identifican los movimientos hacia la izquierda.

Las teclas asignadas a los movimientos del vehículo varían de acuerdo a la dirección que presente su imagen. A continuación se muestran cuatro tablas que indican la acción que representa cada tecla o combinación de tecla.

DIRECCION 1.-

ACCION	TECLA
Avance	→
Retroceso	←
Rotación adelante - derecha	↓
Rotación atrás - derecha	Ctrl + ↓
Rotación adelante - izquierda	↑
Rotación atrás - izquierda	Ctrl + ↑

DIRECCION 2.-

ACCION	TECLA
Avance	←
Retroceso	→
Rotación adelante - derecha	↑
Rotación atrás - derecha	Ctrl + ↑
Rotación adelante - izquierda	↓
Rotación atrás - izquierda	Ctrl + ↓

DIRECCION 3.-

ACCION	TECLA
Avance	↑
Retroceso	↓
Rotación adelante - derecha	→
Rotación atrás - derecha	Ctrl + →
Rotación adelante - izquierda	←
Rotación atrás - izquierda	Ctrl + ←

DIRECCION 4.-

ACCION	TECLA
Avance	↓
Retroceso	↑
Rotación adelante - derecha	←
Rotación atrás - derecha	Ctrl + ←
Rotación adelante - izquierda	→
Rotación atrás - izquierda	Ctrl + →

Para este tipo de resolución se lleva un control del tiempo que se tarda el operador en resolver el laberinto. Al finalizar la resolución manual se muestra en pantalla su tiempo (en segundos), y se lo compara con el mejor tiempo registrado para resolver el laberinto que escogió. En caso de que su tiempo sea menor entonces se le pide ingresar su nombre y se lo registra en la tabla de mejores tiempos. Finalmente se muestra la tabla de los mejores tiempos de resolución registrados para cada uno de los laberintos. Inicialmente los tiempos registrados son de 9999 segundos (ver Fig. 4.8.).

Fig. 4.8. Ejemplo de la Tabla de los Mejores Tiempos de Resolución

NOMBRE	TIEMPO	LAB.
OPERADOR1	9999	1
OPERADOR2	9999	2
OPERADOR3	9999	3
OPERADOR4	9999	4

4.2.3.2 RESOLUCION AUTOMATICA

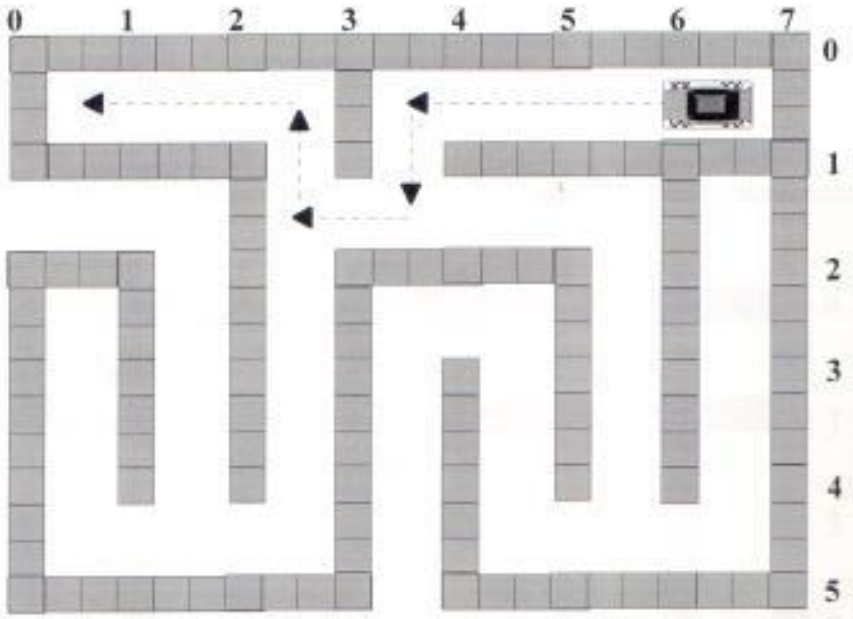
Para el tipo de resolución automática el vehículo debe aplicar un algoritmo que le permita encontrar la salida por sí mismo; es decir, que el operador del computador no interviene en el control de sus movimientos.

La idea básica para resolver el laberinto es que el vehículo siempre tienda a avanzar, por tanto sus movimientos principales serán: Avance, Rotación adelante - derecha y Rotación adelante - izquierda, asignándoles una prioridad a cada uno de ellos:

PRIORIDAD	MOVIMIENTO
1	**Avance**
2	**Rotación adelante - derecha**
3	**Rotación adelante - izquierda**

Esto significa que ante la posibilidad de circulación del vehículo, primeramente éste deberá intentar avanzar, en caso de no poder hacerlo, intentará efectuar un giro hacia la derecha y por último si no fueron posibles ninguno de los dos movimientos anteriores, intentará girar hacia la izquierda. Luego, se evalúan nuevamente cada una de las opciones mencionadas y se continua hasta salir del laberinto, o hasta llegar a una vía sin salida (ver Fig. 4.9).

Fig. 4.9. Secuencia de movimientos de avance del vehículo

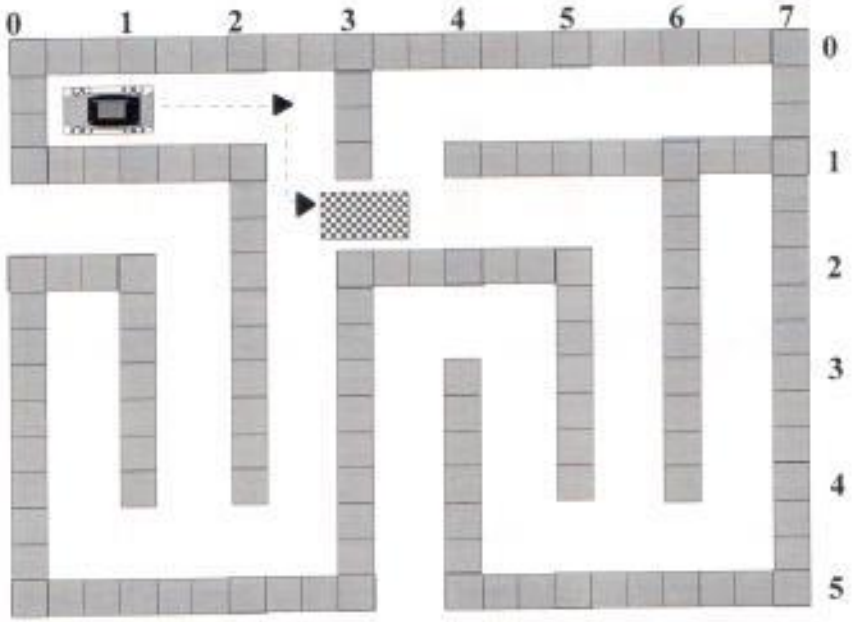


En caso de que el vehículo se encuentre en una vía sin salida, es decir que no pueda realizar ninguno de los tres movimientos anteriormente mencionados, entonces deberá ejecutar un retroceso en todos sus movimentos hasta llegar a un punto en que se le presentó una doble o triple opción de movimiento. Cada movimiento debe revertirse con su opuesto de acuerdo a la siguiente tabla:

ACCION	ACCION OPUESTA
Avance	**Retroceso**
Rotación adelante - derecha	**Rotación atrás - derecha**
Rotación adelante - izquierda	**Rotación atrás - izquierda**

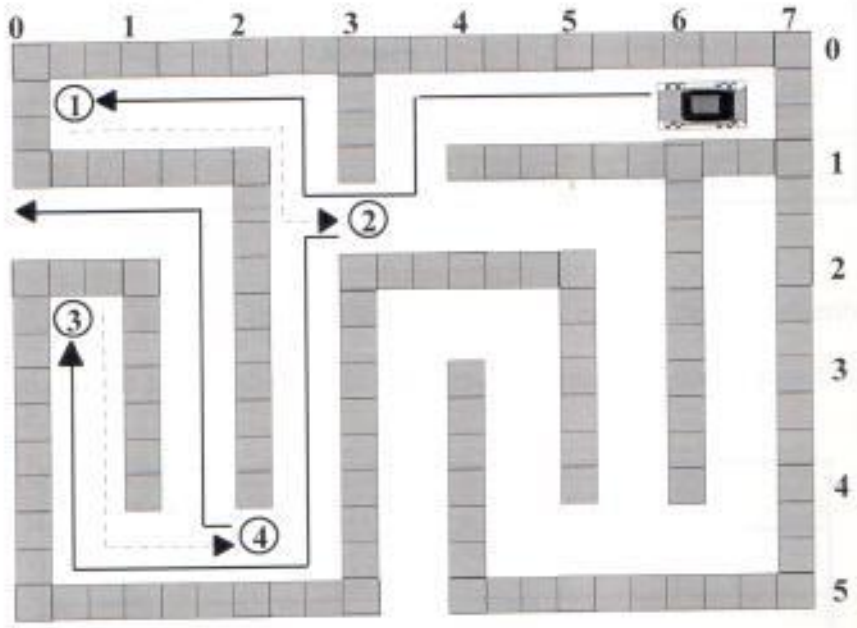
Tomando el ejemplo de la Fig. 4.9, el retroceso se ilustra en la Fig. 4.10. en donde se observa que el vehículo se detiene ante la última posibilidad que tuvo de escoger un movimiento.

Fig. 4.10. Secuencia de movimientos de retroceso del vehículo



El siguiente paso es reanudar los movimientos de avance tomando una vía diferente a la tomada en la primera ocasión. En la Fig. 4.11. se ilustran los movimientos del vehículo desde su punto de partida, hasta que pasa a otra sección del laberinto. Con líneas continuas se identifican los movimientos de avance y con líneas punteadas los movimientos de retroceso. Cada punto de parada que debe realizar el vehículo ha sido numerado indicando la secuencia de las paradas.

Fig. 4.11. Secuencia de movimientos del vehículo en una sección



Para controlar todos los movimientos que ejecuta el vehículo y para determinar los puntos de parada al momento que tiene que retroceder, se ha utilizado una estructura llamada *EST* compuesta por los siguientes elementos:

```
struct EST
{
        int tipmov, marca;
}
```

donde *tipmov* almacena un valor entero identificando el tipo de movimiento ejecutado por el vehículo de acuerdo a la siguiente tabla:

TIP_MOV	MOVIMIENTO
1	**Avance**
2	**Rotación adelante - derecha**
3	**Rotación adelante - izquierda**

y *marca* almacena un valor entero identificando si la opción de movimiento es única o existe alguna otra posibilidad. Este valor está determinado por la siguiente tabla:

MARCA	OPCION
0	**Unica**
1	**Cruce a la derecha libre**
2	**Cruce a la izquierda libre**

Luego, se define un arreglo llamado *Control* de tipo EST, compuesto por 80 registros (número máximo de movimientos de avance permitidos) donde cada registro presenta la siguiente estrucutura (ver Fig.4.12.).

Fig. 4.12 Estructura de un Registro del Arreglo Control

TIP_MOV	MARCA

Combinando los valores de *tip_mov* y *marca*, se pueden controlar los movimientos que realiza el vehículo. Así, analizando tres casos tenemos:

- Si *tip_mov* = 1 y *marca* = 0 significa que el vehículo ejecuta un avance sin opción de giro.
- Si *tip_mov* = 2 y *marca* = 2 significa que el vehículo rota hacia adelante a la derecha pero que tenía una opción de rotación hacia adelante izquierda que no toma.
- Si *tip_mov* = 1 y *marca* = 1 significa que el vehículo avanza hacia adelante pero que tenía una opción de rotación hacia adelante derecha que no toma.

Cuando el vehículo ya no puede seguir avanzando, debe retroceder cada uno de sus movimientos hasta que *marca* <> 0, incluyendo este último registro. De acuerdo a la información del último registro revertido, se ejecuta un movimiento en una dirección diferente a la tomada en la primera ocasión de acuerdo a la siguiente tabla:

REGISTRO CON MARCA <> 0		**NUEVO REGISTRO**	
TIP_MOV	**MARCA**	**TIP_MOV**	**MARCA**
1	**1**	**2**	**0**
1	**2**	**3**	**0**
2	**2**	**3**	**0**
3	**1**	**2**	**0**

De esta manera, el arreglo para resolver el laberinto de la Fig. 4.11. va creándose de acuerdo a las tablas de la Fig. 4.13, indicando paso a paso los cambios de movimientos que se realizan:

Fig. 4.13. Creación del arreglo *Control* para la Fig. 4.11.

Desde el inicio hasta la Parada ①		
No. REG.	**TIP_MOV**	**MARCA**
0	1	0
1	3	0
2	2	2
3	2	2
4	3	0
5	1	0

Desde ① hasta ②		
No. REG.	**TIP_MOV**	**MARCA**
0	1	0
1	3	0
2	2	2
3	2	2
4	3	0
5	1	0

Registros que se deben revertir

Fig. 4.13. Creación del arreglo *Control* para la Fig. 4.11. (continuación)

Desde ② hasta ③		
No. REG.	**TIP_MOV**	**MARCA**
0	1	0
1	3	0
2	2	2
3	3	0
4	1	0
5	2	0
6	1	1
7	2	0
8	1	0

Desde ③ hasta ④		
No. REG.	**TIP_MOV**	**MARCA**
0	1	0
1	3	0
2	2	2
3	3	0
4	1	0
5	2	0
6	1	1
7	2	0
8	1	0

Fig. 4.13. Creación del arreglo *Control* para la Fig. 4.11. (continuación)

Desde ④ hasta el paso a otra sección		
No. REG.	**TIP_MOV**	**MARCA**
0	1	0
1	3	0
2	2	2
3	3	0
4	1	0
5	2	0
6	2	0
7	1	0
8	3	0
9	1	0

Como puede observarse en la Fig. 4.13., se efectuaron en total 15 movimientos de avance; sin embargo, 6 de ellos tuvieron que revertirse, por tanto para resolver la sección de laberinto de la Fig. 4.11 se requieren 9 movimientos.

4.3 INTEGRACION DE LA APLICACION

La aplicación desarrollada ha sido denominada **"LABERINTO"** en referencia al proyecto planteado que consiste en implantar el Sistema de Control del Vehículo Prototipo por Computadora con un vehículo ubicado dentro de un laberinto, en el cual tiene que encontrar la salida ya sea en forma manual o automática. Una vez conocida la metodología de solución de la aplicación, se procede a integrar todos sus elementos que lo constituyen.

4.3.1 ESTRUCTURA DEL SISTEMA

El sistema de aplicación integrado consiste básicamente en un Módulo de Software y un Módulo de Electrónica.

El Módulo de Software está compuesto por el Programa Principal incluyendo sus procedimientos internos, las librerías y archivos propios de C++ y las librerías de clases desarrolladas para el control del vehículo y la visualización tanto del laberinto como del vehículo (ver Fig. 4.14.).

El Módulo de Electrónica está constituido por el computador, la interfaz digital/analógica conectada al computador a través de su puerto paralelo y el vehículo prototipo que recibe las señales enviadas por la interfaz digital/analógica, tal como se observa en la Fig.4.15..

Fig. 4.14. Integración del Módulo de Software

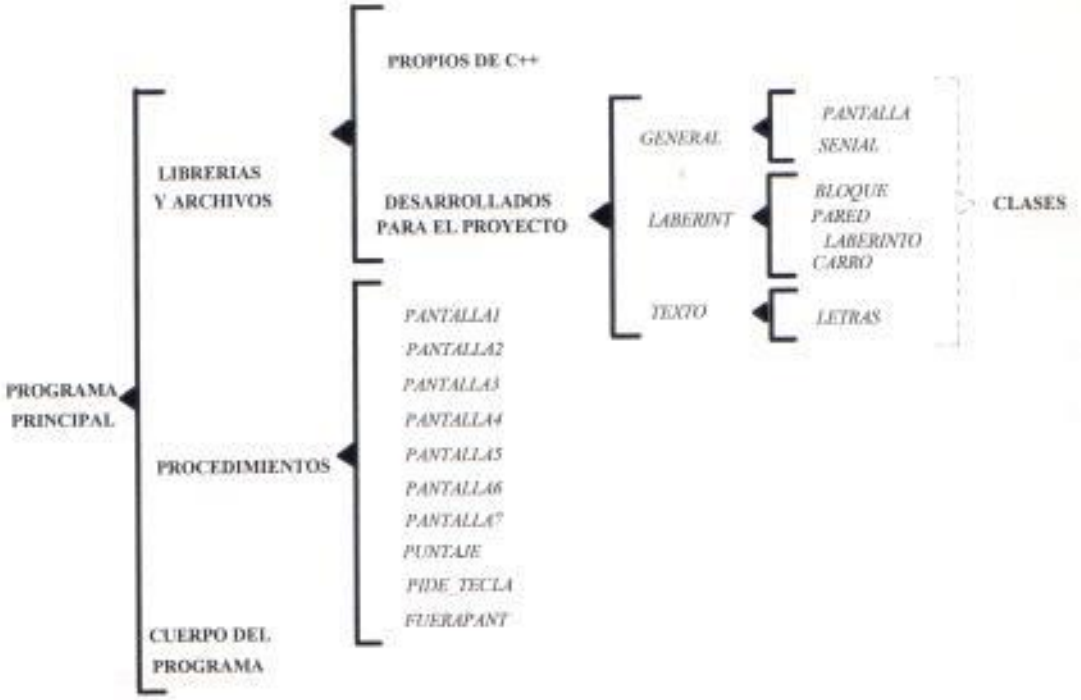


Fig. 4.15. Integración del Módulo de Electrónica

4.3.2 DESCRIPCION Y ANALISIS DE LOS MODULOS DE LA APLICACION

4.3.2.1 MODULO DE SOFTWARE

La metodología aplicada para desarrollar el Módulo de Software fue extensamente analizada en el capítulo II y en el presente capítulo se procede a describir y detallar los programas que constituyen la aplicación desarrollada, los cuales pueden ser agrupados en tres bloques: las librerías de clases, los procedimientos del programa principal y el programa principal.

La codificación de los programas ha sido incluida en el presente documento como un anexo que se encuentra al final del mismo.

LIBRERIAS DE CLASES

Para el diseño de este proyecto se ha aplicado la programación orientada a objeto, lo que implica el desarrollo de clases, las mismas que han sido agrupadas en tres librerías, cada una de ellas con sus características particulares: General, Laberint y Texto.

LIBRERIA GENERAL.-

La librería General contiene las clases que definen el ambiente general del sistema y que pueden ser implantadas en cualquier aplicación. Presenta dos clases: Pantalla y Senial.

Clase PANTALLA

Define el ambiente gráfico del sistema. Está compuesta por los siguientes miembros públicos:

pantinic.- Inicializa el ambiente gráfico.

pantbase.- Retorna un valor entero correspondiente al color de fondo de pantalla de acuerdo al tipo de monitor: 0 si es monocromático o 1 si es a color.

pantmaxx.- Retorna un valor entero correspondiente al número máximo de pixeles en el eje X.

pantmaxy.- Retorna un valor entero correspondiente al número máximo de pixeles en el eje Y.

pantaspt.- Retorna un valor double correspondiente a la razón de aspecto.

pantgraf.- Limpia la pantalla con el color de fondo correspondiente

Clase SENIAL

Determina y envía códigos al puerto paralelo del computador. Sus miembros públicos son:

Senial.- Determina los códigos a enviarse.

senenvio.- Envía códigos al puerto paralelo de acuerdo al movimiento realizado por el vehículo. Este valor ingresa a la función como un valor entero.

LIBRERIA LABERINT

Está compuesta por las clases diseñadas exclusivamente para el proyecto como Bloque, Pared, Laberinto y Carro.

Clase BLOQUE

Define y grafica un bloque bidimensional en la posición indicada. Está compuesta por los siguientes miembros públicos:

Bloque.- Determina los parámetros de definición del bloque.

bloqgraf.- Grafica un bloque del tamaño y la posición señalada.

Clase PARED

A partir de la clase Bloque define y grafica paredes horizontales y verticales. Sus miembros públicos son:

PHGraf.- Grafica una pared horizontal en la posición indicada a partir de la función bloqgraf de la clase Bloque.

PVGraf.- Grafica una pared vertical en la posición indicada a partir de la función bloqgraf de la clase Bloque.

Clase LABERINTO

A partir de la clase Pared define y grafica secciones de laberinto en la pantalla de acuerdo a la información que obtiene del archivo DATOS.TXT para diseñar el laberinto. Está compuesto por los siguientes miembros públicos:

Laberinto.- Determina los parámetros de definición del laberinto.

labinic.- Crea el arreglo que contiene la información del laberinto seleccionado tomando los datos del archivo DATOS.TXT..

labgraf.- Grafica una sección de laberinto en la pantalla a partir de las funciones PHGraf y PVGraf de la clase Pared.

Clase CARRO

Define y grafica el vehículo en pantalla, además de realizar el control de sus movimientos.

Sus miembros privados son:

limpia.- Limpia el área donde estuvo el vehículo.

rotación.- Efectua la rotación de todos los puntos que corresponden a las coordenadas del vehículo de acuerdo al ángulo y eje indicado.

pos_carro.- Indica la ubicación del vehículo en la pantalla de acuerdo a su dirección.

Sus miembros públicos son:

Carro.- Determina los parámetros de definición del vehículo.

inicializa.- Define las coordenadas que constituyen los vértices del vehículo.

grafico.- Grafica el vehículo en el puerto visual actual.

posiciona.- Ubica el vehículo en la dirección y posición indicada.

rota.- Efectua la rotación del vehículo en la dirección indicada: adelante derecha, adelante izquierda, atrás derecha o atrás izquierda y retorna la ubicación del vehículo en pantalla.

avanza.- Efectua el movimiento de traslación del vehículo en la dirección indicada: avance o retroceso y retorna la ubicación del vehículo en pantalla.

avan_espe.- Efectua el movimiento de traslación del vehículo a una posición que le permite efectuar un giro inmediato y retorna la ubicación del vehículo en pantalla.

puesto.- Sensa las paredes y determina si existe una pared adelante, a la izquierda y/o a la derecha del vehículo.

LIBRERIA TEXTO

Define y grafica símbolos y letras que no pueden ser creadas y graficadas con las librerías propias de C++. Está compuesta por la clase LETRAS.

Clase LETRAS

Define y grafica todas las letras del abecedario. Sus miembros públicos son:

Letras.- Define la estructura base para el diseño de las letras a la escala y factor señalado. Además, define el arreglo que identifica cada letra.

let_graf.- Grafica el texto ingresado en la posición señalada, y del color y patrón indicados.

PROCEDIMIENTOS DEL PROGRAMA PRINCIPAL

En el programa principal del proyecto se han definido algunos procedimentos que se utilizan para mostrar pantallas de presentación y de opciones, y otros para controlar la interacción entre el vehículo y el laberinto. Estos procedimientos son:

Pantalla1.- Pantalla de presentación que indica los datos de la Universidad y Facultad.

Pantalla2.- Pantalla de presentación que indica el nombre de la aplicación y autor.

Pantalla3.- Pantalla que indica los antecedentes de la aplicación y sus objetivos.

Pantalla4.- Pantalla para seleccionar la opción de resolución o de finalizar la aplicación.

Pantalla5.- Pantalla para seleccionar el laberinto que se utilizará.

Pantalla6.- Pantalla para indicar el tiempo que se tomó el operador para resolver el laberinto.

Pantalla7.- Pantalla para indicar que se ha finalizado el uso de la aplicación.

Puntaje.- Procedimiento para calcular el tiempo de resolución del laberinto.

pide_tecla.- Procedimiento que pide la digitación de una tecla y valida su ingreso.

FueraPant.- Procedimiento para determinar si el vehículo se encuentra fuera de la pantalla y verificar si terminó el laberinto o pasa a otra sección.

PROGRAMA PRINCIPAL.- TESIS

El programa principal es el que integra toda la aplicación y es desde el cual se invoca a todas las librerías y procedimientos desarrollados.

Básicamente el programa muestra unas pantallas de presentación y luego de seleccionar el tipo de resolución y laberinto a resolver se muestra en pantalla el vehículo y el laberinto. En caso de escoger una resolución manual, el vehículo no se moverá hasta que el operador digite alguna de las teclas de control. Para la resolución automática únicamente se requiere observar la trayectoria que va tomando el vehículo hasta que encuentra la salida por sí solo. Finalmente se brinda nuevamente la opción de escoger el tipo de resolución y algún otro laberinto para resolver. Para la resolución manual se calcula y compara el tiempo de resolución del laberinto y de acuerdo a su valor se lo registra o no en la tabla de mejores tiempos.

Cada movimiento que realiza el vehículo está codificado, y esta codificación es enviada al puerto paralelo para transmitir las señales al vehículo prototipo.

4.3.2.2 MODULO DE ELECTRONICA

El Módulo de Electrónica está compuesto únicamente el Módulo de Electrónica desarrollado para el Sistema de Control por Computadora del Vehículo Prototipo, es decir que no se requieren conexiones adicionales para poder operar con el vehículo una vez integrado a la aplicación.

Un mayor detalle del Módulo de Electrónica fue presentado en el Capítulo III, en el cual de describió y detalló cada uno de los elementos que lo componen, los circuitos que se desarrollaron, el equipo utilizado y las condiciones de operación.

4.3.3 OPERACION DEL SISTEMA

Para poder operar con el sistema desarrollado se requieren efectuar los siguientes pasos:

a) INSTALACION DEL COMPUTADOR

Implica la conexión básica del computador compuesto por el CPU, el monitor y el teclado. No se incluye impresora puesto que el puerto paralelo será utilizado para la conexión con la interfaz digital/analógica.

b) INSTALACION DE LA APLICACION EN EL COMPUTADOR

Consiste en crear un directorio en el computador donde sean almacenados todos los archivos requeridos para la ejecución del proyecto o en su defecto, se utiliza un disco que tenga almacenado dichos archivos.

Para poder ejecutar la aplicación se requieren los siguientes archivos:

- TESIS.EXE
- TESIS.OBJ
- DATOS.TXT

- PUNTOS.TXT
- LABERINT.H
- GENERAL.H
- TEXTO.H
- ARCHIVOS BGI Y CHR DE C++

c) CONEXION DE LA INTERFAZ DIGITAL/ANALOGICA AL COMPUTADOR

Se verifican todas las conexiones internas de la interfaz digital/analógica y luego se extiende el cable y se lo conecta al puerto paralelo del computador.

d) UBICACION Y ENCENDIDO DEL VEHICULO PROTOTIPO

Se verifica que el vehículo prototipo se encuentre cargado con las pilas respectivas, se lo enciende y se lo ubica en el laberinto en la posición determinada por el proyecto.

e) EJECUCION DE LA APLICACION

Para ejecutar la aplicación basta con ubicarse en el directorio donde se encuentren los archivos del proyecto y digitar: **TESIS**↵.

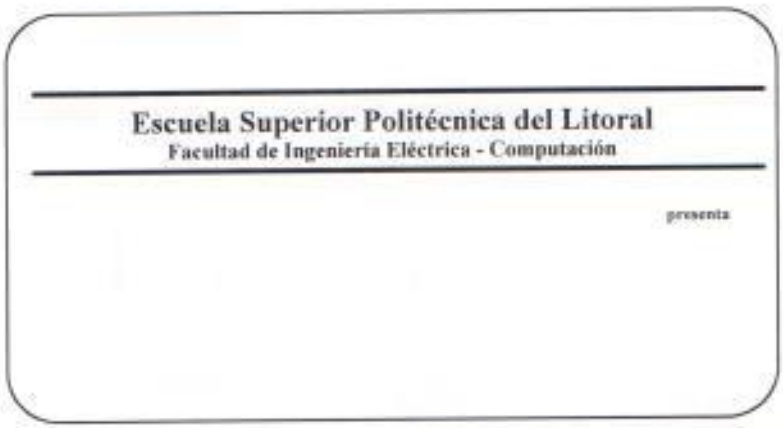
4.4 PRUEBAS Y AJUSTES

A continuación se describe una corrida de prueba de la aplicación desarrollada.

- Inicialmente se digita **TESIS <ENTER>**.

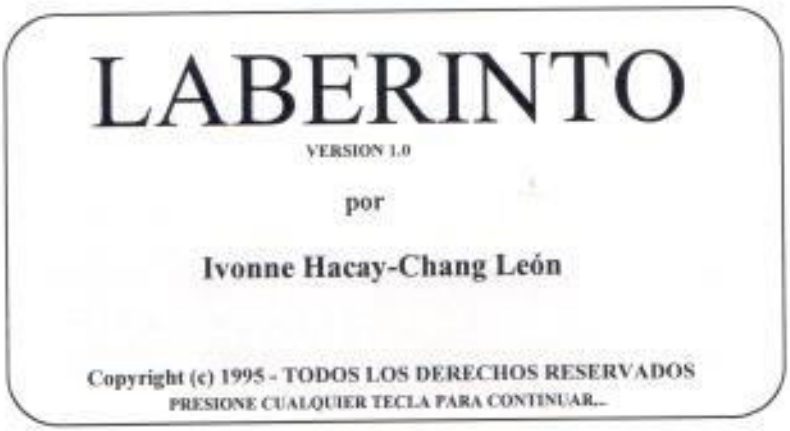
- Se muestra en pantalla una presentación con el nombre de la universidad y de la Facultad. Luego se presiona cualquier tecla para continuar.

Fig. 4.16. Pantalla de Presentación de la Aplicación



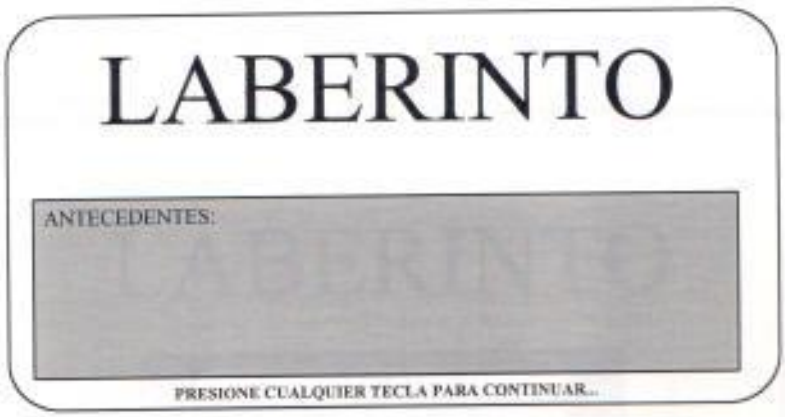
- Se muestra el título con el que identifica a la aplicación "LABERINTO" y el nombre de su autor. Se presiona cualquier tecla para continuar.

Fig. 4.17. Pantalla indicando Nombre y Autor del Proyecto



- Se describe la aplicación brevemente planteado sus antecedentes.

Fig. 4.18. Pantalla presentando los Antecedentes de la Aplicación



- Se plantean los objetivos del proyecto exponiendo los dos tipos de resolución del laberinto: manual y automático. Se presiona cualquier tecla para continuar.

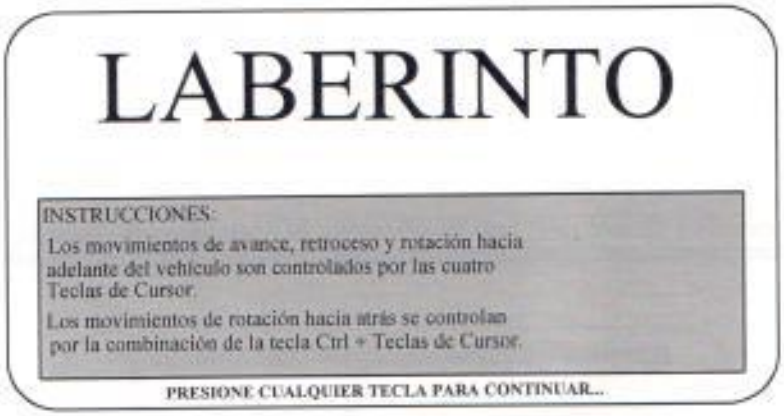
Fig. 4.19. Pantalla presentando los Objetivos y Tipos de Resolución

- Se muestra una pantalla de opciones con los dos tipos de resolución del laberinto: manual o automática, y la opción de salir del programa. Para moverse de una opción a otra se utilizan las flechas de cursor: ↑ o ↓, y se selecciona presionando <ENTER>.

Fig. 4.20. Pantalla presentando las Opciones de Resolución

- En caso de seleccionar la Resolución Manual, se muestra a continuación una pantalla con las instrucciones para utilizar las teclas de control.

Fig. 4.21. Pantalla de instrucciones para el uso de las Teclas de Control

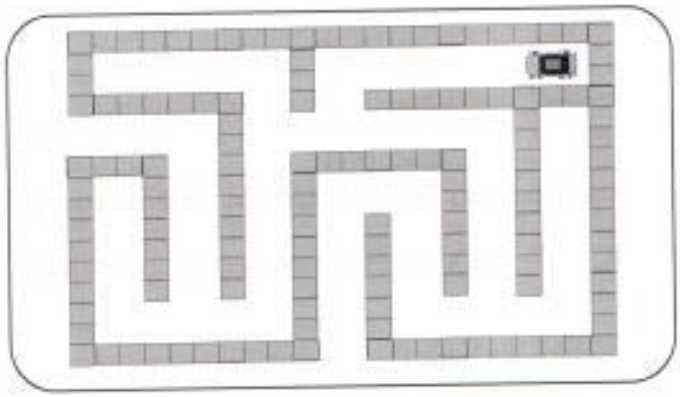


- Luego, se muestra una pantalla de opciones para escoger entre los cuatro laberintos diseñados para el proyecto o para escoger uno de ellos de manera aleatoria. Para moverse de una opción a otra se utilizan las flechas de cursor: ↑ o ↓, y se selecciona presionando <ENTER>.

Fig. 4.22. Pantalla presentando las Opciones de Laberinto

- Aparece el vehículo en el laberinto escogido. Si se seleccionó una Resolución Manual, el vehículo no se moverá hasta que el operador de la computadora no presione alguna de las teclas de control, y será él quien guíe al vehículo hasta encontrar la salida. Para una Resolución Automática, el vehículo empezará a moverse por sí solo aplicando el algoritmo de resolución del laberinto hasta encontrar la salida.

Fig. 4.23. Pantalla presentando el vehículo en una Sección de Laberinto



- Si la resolución fue manual, se calcula el tiempo (en segundos) que el operador se tardó en resolver el laberinto y que luego aparece en pantalla.

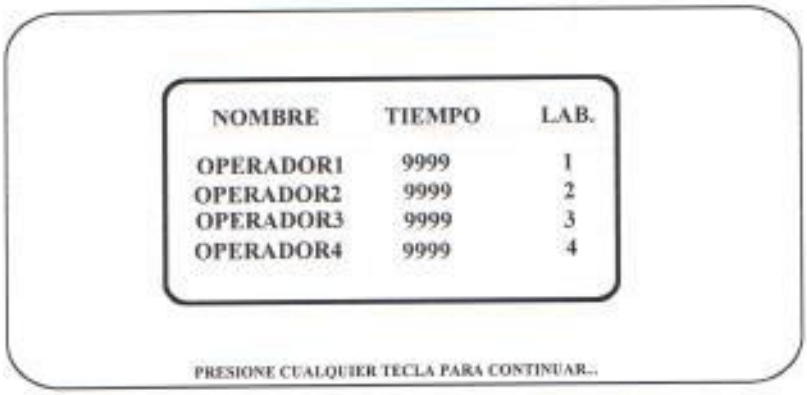
Fig. 4.24. Pantalla presentando el Tiempo de Resolución

Se compara el tiempo registrado por el operador con el que está registrado para el laberinto correspondiente, tal que, si su tiempo fue menor entonces aparece en pantalla una pequeña área solicitándole al operador que digite su nombre (máximo 10 caracteres). Presionando la tecla de <BACKSPACE> es posible retroceder para corregir cualquier caracter mal ingresado.

Fig. 4.25. Pantalla de Ingreso del Nombre del Operador

Y se muestra en pantalla una tabla donde se encuentran los mejores tiempos de resolución registrados para cada laberinto.

Fig. 4.26. Pantalla presentando los Mejores Tiempos de Resolución



- Luego, se muestra nuevamente la pantalla de opciones de la Fig. 4.20., brindando la oportunidad de continuar ejecutando la aplicación o de finalizar su ejecución.

- En caso de seleccionar la opción "**SALIR**" (ver Fig. 4.20), se muestra una pantalla de fin de programa, se presiona cualquier tecla y se termina la ejecución.

Fig. 4.27. Pantalla presentando Finalización de la Aplicación

CONCLUSIONES Y RECOMENDACIONES

1. Se ha desarrollado una aplicación utilizando un vehículo prototipo y una interfaz digital/analógica, mediante la implantación de un Módulo de Software y otro de Hardware, demostrándose su utilidad y facilidad de manejo.

2. El Módulo de Software ha sido desarrollado aplicando un lenguaje de programación orientado a objeto constatando sus ventajas, especialmente las referentes a encapsulación y herencia, que permitieron optimizar la codificación mejorando su estructura, modularidad, portabilidad y rapidez de ejecución.

3. Para la aplicación se crearon tres librerías: una con las funciones que controlan el ambiente gráfico, otra las clases que definen al vehículo y al laberinto, y otra con las letras especiales. Todas estas librerías pueden ser ampliadas y constituyen herramientas que puede ser utilizadas en proyectos futuros facilitando el manejo del entorno gráfico y el desarrollo de nuevas aplicaciones.

4. Mediante la identificación del teclado con los movimientos del vehículo en pantalla se logró coordinar sus desplazamientos y giros de acuerdo a la dirección en que se encontraba, sin embargo no es posible determinar la dirección real del vehículo respecto de su operador.

5. Debido al uso de motores DC en el vehículo prototipo, cuya velocidad no es constante, no es posible coordinar con exactitud los movimientos del vehículo en pantalla con la realidad. Además, el funcionamiento del vehículo prototipo

depende del uso de baterías y de acuerdo a su carga, el vehículo responde más rápido o no a las señales de radio que se le transmiten.

6. Ya que el vehículo prototipo no presenta ningún tipo de dispositivo para frenar, al momento de enviarle la señal de pare, éste continua realizando un desplazamiento adicional debido a su inercia. Este efecto se puede observar claramente al momento que el vehículo realiza un giro y éste no se ubica en la posición esperada, sino que ejecuta un avance hacia adelante una vez que termina de girar.

De la experiencia realizada se presentan las siguientes recomendaciones:

1. La coordinación del envío de señales del computador a un dispositivo externo vía radio es una técnica que va tomando fuerza en la actualidad, y debe continuarse investigando y reforzando ya que su aplicación reduce los costos de comunicación y facilita el acceso a lugares que anteriormente era prácticamente imposible de llegar.

2. La librería de clases para el laberinto está constituida de elementos que pueden servir para la creación de clases mucho más complejas, ampliando el paquete de herramientas de graficación.

3. El uso de sensores en el vehículo prototipo permitiría controlar mejor sus desplazamientos, tal que las señales no solo sean generadas por el computador y enviadas al vehiculo, sino que también sean receptadas por el computador y procesadas para determinar sus próximos movimientos.

4. Sería recomendable el uso de baterías de larga duración para el funcionamiento del vehículo prototipo que permitan obtener una carga continua estable por el mayor tiempo posible, tal que se pueda mejor la coordinación de movimientos con el vehículo en pantalla.

BIBLIOGRAFIA

1. HEARN D. Y BAKER P., Computer Graphics, Prentice Hall, Inc., New Yersey, 1986.

2. HOLZNER, S., Borland C++ Programming, Brady Books, New York, 1992.

3. McCORD, J., Borland C++ Programmer's guide to graphics, SAMS, Indiana, 1991

4. MUELLER, S., Upgrading and Repairing PCs, QUE Corporation, Indiana, 1992, 721 p.

5. Turbo C++ Library Reference, Borland International, California, 1990.

6. Turbo C++ Programmer's Guide, Borland International, California, 1990.

APENDICE

CODIFICACION DE LA APLICACION DESARROLLADA

LIBRERIA GENERAL

Clase Pantalla

Miembros Públicos:

void pantinic ( )

int pantbase ( )

int pantmaxx ( )

int pantmaxy ( )

double pantaspt ( )

void pantgraf ( )

void Pantalla - pantinic ( )

Inicio

Inicialización del modo gráfico

Si existe algún error entonces

Mostrar "Error en el Sistema Gráfico"

Fin Si

Fin

int Pantalla - pantbase ( )

Inicio

Si la pantalla es a color

retornar 1 *(Fondo de pantalla azul)*

sino

retornar 0 *(Fondo de pantalla negro)*

Fin Si

Fin

int Pantalla - pantmaxx ( )

Inicio

Retornar el número máximo de pixeles en el eje X

Fin

int Pantalla - pantmaxy ( )

Inicio

Retornar el número máximo de pixeles en el eje Y

Fin

double Pantalla - pantaspt ( )

Inicio

Cálculo de la razón de aspecto

Retornar razón de aspecto

Fin

void Pantalla - pantgraf ( )

Inicio

Limpieza de la pantalla

Pintar la pantalla del color de fondo *(1 o 0 dependiendo del monitor)*

Fin

Clase Senial

Miembros Privados:

char *senval[7]

Miembros Públicos:

Senial ( )

void senenvio (int)

Senial - Senial ( )

Inicio

Almaceno los códigos de control en el arreglo senval

Fin

void Senial - senenvio (int senmovi)

(senmovi es el tipo de movimiento del vehículo)

Inicio

De acuerdo al valor de senmovi se envía un código al puerto paralelo

Fin

LIBRERIA LABERINT

Clase Bloque

Miembros Públicos:

Variables de pantalla

Bloque ( )

int blq_tam

void blqgraf (int, int, int)

.

Bloque - Bloque ( )

Inicio

Asignación de valores a variables de pantalla

Fin

void blqgraf (int bloqtama, int blq_posx, int blq_posy)

(bloqtama indica el tamaño del bloque)

(blq_posx indica la coordenada en X del origen del bloque)

(blq_posy indica la coordenada en Y del origen del bloque)

Inicio

Determinación del patrón, tipo de línea y color a utilizar

Graficar el recuadro que simula el bloque

Fin

Clase Pared -Public Bloque

Miembros privados:

int blq_temp, blq_i

Miembros públicos:

void PHGraf (int, int, int)

void PVGraf (int, int, int)

void PHGraf (int blq_tam, int blq_posx, int blq_posy)

Inicio

blq_temp = blq_posx

Para blq_i = 0 mientras blq_i < 4 entonces i = i+1

bloqgraf (blq_tam, blq_temp, blq_posy)

blq_temp = blq_temp + blq_tam

Fin Para

Fin

void PVGraf (int blq_tam, int blq_posx, int blq_posy)

Inicio

blq_temp = blq_posy

Para blq_i = 0 mientras blq_i < 4 entonces i = i+1

bloqgraf (blq_tam, blq_posx, blq_temp)

blq_temp = blq_temp + blq_tam

Fin Para

Fin

Clase Laberinto - Public Pared

Miembros privados:

Definición de la estructura PTS {

int x, y; *(Coord. en X y Y del origen de una pared)*

int z,w *(z=1 sí existe pared horizontal, z=0 no existe)*

(w=1 sí existe pared vertical, w=0 no existe)

}

Definición del tipo PTS laber [6] [10] [10] *(6 No. secciones)*

(10 No. máx. de filas)

(10 No. máx. de columnas)

Laber L

int lab_i, lab_j

Miembros públicos:

Laberinto (int, int)

void labinic ( )

void labgraf (int)

Laberinto - Laberinto (int lab_bloq_tam, int lab_numero)

(lab_bloq_tam indica el tamaño del bloque)

(lab_numero indica el número del laberinto seleccionado)

Inicio

Definición de los valores de lab_blq, lab_col, lab_fil

(lab_blq es el tamaño del bloque)

(lab_col es el número máximo de columnas en una sección)

(lab_fil es el número máximio de filas en una sección)

Fin

void Laberinto - labinic ( )

Inicio

Limpieza del arreglo L

Apertura del archivo DATOS.TXT

Ubicación del puntero dentro del archivo DATOS.TXT al inicio de la inf. del laberinto seleccionado

Cargar la información del laberinto seleccionado al arreglo L

Cierre del archivo DATOS.TXT

Fin

void Laberinto - labgraf (int lab_pantalla)

Inicio

int labpan

labpan = lab_pantalla

Para lab_i=0 mientras lab_i < lab_fil entonces lab_i = lab_i + 1

Para lab_j=0 mientras lab_j < lab_col entonces lab_j = lab_j + 1

Si (L [labpan] [lab_i] [lab_j].z = "1" entonces

PHGraf (lab_blq, L [labpan] [lab_i] [lab_j].x, L [labpan] [lab_i] [lab_j].y)

Fin Si

Si (L [labpan] [lab_i] [lab_j].w = "1" entonces

PVGraf (lab_blq, L [labpan] [lab_i] [lab_j].x, L [labpan] [lab_i] [lab_j].y)

Fin Si

Fin Para

Fin Para

Fin

Clase Carro

Miembros privados:

pointype cr[58], llan_del[4], llan_tra[4], chasis[8], ventanas[10],
adorno[6], techo[4], part_tra[4]

Definición de la estructura coord_posic

float esq1X, esq1Y

float esq2X, esq2Y

void limpia ( )

void rotacion (double, int, int)

coord_posic pos_carro (int)

Miembros públicos:

Definición de la estructura coord_ubica

int direc, esqX, esqY *(Dirección, Coord.X y Y del origen)*

Definición de la estructura banderas

int der, izq, ade *(1 sí existe pared a la derecha, izquierda o adelante del vehículo, 0 no existe)*

Carro (int, int)

void inicializa ( )

void grafico ( )

void posiciona (int, int, int)

coord_ubica rota (int, int, int, int, int)

coord_ubica avanza (int, int, int, int)

coord_ubica avan_espe (int, int, int, int)

banderas puesto (int, int)

Carro - Carro (int car_largo, int car_ancho)

Inicio

LARGO = car_largo *(Largo del vehículo)*

ANCHO = car_ancho *(Ancho del vehículo)*

Asignación de valores a variables de pantalla

Fin

Carro - inicializa ( )

Inicio

Asignación de valores al arreglo cr *(Coord. para construir el vehículo)*

Fin

Carro - grafico ( )

Inicio

Asignación de todas las coordenadas en cr al arreglo respectivo:

llan_del, llan_tra, chasis, ventanas, adorno, techo, part_tra

Graficación del vehículo

Fin

Carro - limpia ( )

Inicio

Asignación de coordenadas a los arreglos llan_del, llan_tra, chasis

Rellenar el área formada por esos arreglos con el color de fondo

Fin

void Carro - rotación (double rot_angle, int rot_centx, int rot_ceny)

(rot_angle indica el ángulo de rotación)

(rot_centx y rot_centy indican las coord. del eje de rotación)

```
Inicio
    double angcos, angsin
    pointtype rottemp[58]
    Transformación de rot_angle de grados a radianes
    Si angcos es casi 0 entonces angcos = 0
    Si angsin es casi 0 entonces angsin = 0
    Para rot_i = 0 mientras rot_i < 58 entonces rot_i = rot_i + 1
        rottemp[rot_i].x = cr[rot_i].x
        rottemp[rot_i].y = cr[rot_i].y
        cr[rot_i].x =   (rottemp[rot_i].x * angcos)-(rottemp[rot_i].y *angsin) +
                        ((1-angcos) * rot_centx) + ((angsin * rot_centy))
        cr[rot_i].y =   (rottemp[rot_i].y * angcos)+(rottemp[rot_i].x*angsin) +
                        ((1-angcos) * rot_centy) - ((angsin * rot_centx))
    Fin Para
Fin
```

coord_posic Carro - pos_carro (int pos_car_direc)

(pos_car_direc indica la dirección del vehículo)

```
Inicio
    coor_posic posicion
    De acuerdo a la dirección del vehículo determinar los valores de
        posicion.esq1X, posicion.esq1Y, posicion.esq2X, posicion.esq2Y
Fin
```

void Carro - posiciona (int posic_dir, int posic_X, int posic_Y)

(posic_dir indica la dirección que se desea tenga vehículo)

(posic_X, posic_Y indican las coord. del origen del vehículo)

Inicio

int pos_grad

pointtype pos_a

De acuerdo a la dirección del vehículo calcular los valores de

pos_grad, pos_a.x, pos_a.y *(Indican el ángulo de giro y el eje para girar el vehículo a la dirección requerida)*

Definición el puerto visual del vehículo

inicializa ( )

rotacion (pos_grad, pos_a.x, pos_a.y)

Fin

coor_ubica Carro - rota (int rot_dir, int rot_giro, int rot_sentido, int rot_PosiniX, int rot_PosiniY)

(rot_dir indica la dirección del vehículo)

(rot_giro indica si el giro es hacia la derecha o izquierda)

(rot_sentido indica si el giro es para adelante o para atrás)

(rot_PosiniX, rot_PosiniY indican la coord. del origen del vehículo)

Inicio

De acuerdo a la dirección y giro se determina:

el eje de rotación del vehículo en rot_b y la prox. dirección en rot_prxdir

posiciona (rot_dir, rot_PosiniX, rot_PosiniY)

Si al rotar el vehículo está fuera de pantalla entonces

Retornar dirección y ubicación del vehículo en ubi_car

```
        Pedir una tecla
    Fin mientras
Fin

int pide_tecla ( )
Inicio
    Esperar hasta que el operador tipee una tecla
    Retornar el valor de la tecla
Fin

int FueraPant ( )
Inicio
    Determinación de la posición del vehículo en la pantalla
    Si el vehículo está fuera de pantalla entonces
        De acuerdo a la sección y dirección del vehículo se determina:
            Si el vehículo está fuera del laberinto entonces
                Retornar 1
            sino
                Indicar No. de nueva sección y posición
            Fin Si
        Retornar 2
    Fin Si
    Retornar 0
Fin
```

PROGRAMA PRINCIPAL

Llamado a librerías de C++:

conio.h, ctype.h, dos.h, graphics.h, stdio.h, stdlib.h, string.h, time.h

Llamado a librerías desarrolladas para la aplicación:

texto.h, general.h, laberint.h.

Inicio del Programa Principal

Emisión de sonido de inicio de programa

Apertura de Archivos datos_in y datos_out

Asignación de información de tiempos de resolución al arreglo calif

Pantalla pant; *(Definición de la pantalla)*

Letras titulo (60, 0) *(Definición del titulo)*

Inicializo pant y variables de pantalla

Pantalla1 ( )

Pantalla2 (titulo)

Pantalla3 ( )

tipres = Pantalla4 (titulo) *(Escojo tipo de resolución o "SALIR")*

Mientras (tipres <> 3) *(Repito mientras no escoja "SALIR")*

numlab = Pantalla5 ( ) *(Escojo laberinto)*

Si ya se seleccionó un laberinto entonces

Si laberinto es aleatorio entonces se selecciona entre 1 y 4

Senial misenial; *(Definición de la señal)*

Laberinto milaber (30, numlab-1) *(Definición del laberinto)*

Inicializar milaber

Carro car (60, 30) *(Definición del vehículo)*

```
    Fin Si
    Si rot_sentido = 1 entonces
        Si rot_giro = 0 entonces
            rot_giro = 1
        sino
            rot_giro = 0
        Fin Si
        Recalcular la próxima dirección
    Fin Si
    limpia ( )
    Para rota_i = 30 mientras rota_i < 100 entonces rota_i = rota_i + 30
        posiciona (rot_dir, rot_PosiniX, rot_PosiniY)
        Si rot_giro = 1 entonces
            rotacion (-rota_i, rot_b.x, rot_b.y)
        sino
            rotacion (rota_i, rot_b.x, rot_b.y)
        Fin Si
        Si rota_i < 90 entonces
            limpia ( )
        Fin Si
    Fin Para
    pos_car = pos_carro (rot_prxdir)
    Definición del puerto visual del vehículo
    Determinación de la dirección y ubicación del vehículo en ubi_car
    Retornar ubi_car
Fin
```

coord_ubica Carro - avanza (int ava_dir, int ava_sentido,

int ava_PosiniX, int ava_PosiniY)

(ava_dir indica la dirección del vehículo)

(ava_sentido indica si el avance es para adelante o para atrás)

(ava_PosiniX, ava_PosiniY indican la coord. del origen del vehículo)

Inicio

De acuerdo a la dirección se calcula la dirección de avance

ava_paso = 2 (Cada movimiento será de 2 pixeles)

Cálculo del punto tope (Punto máximo de acercamiento a una pared)

inicializa ( )

Tomar la imagen del vehículo

Repetir

Sensar las paredes de los lados

Si no existe una pared hacia la derecha o izquierda entonces

Salir del lazo

Fin Si

Ubicar la imagen del vehículo en la nueva posición

Actualizar la posición del vehículo dando un nuevo paso

Mientras que el vehículo no llegue al tope

Definición del puerto visual del vehículo

Determinación de la dirección y ubicación del vehículo en ava_ubi

Retornar ava_ubi

Fin

coord_ubica Carro - avan_espe (int ave_dir, int ave_sentido,

int ave_PosiniX, int ave_PosiniY)

(ave_dir indica la dirección del vehículo)

(ave_sentido indica si el avance es para adelante o para atrás)

(ave_PosiniX, ave_PosiniY indican la coord. del origen del vehículo)

Inicio

De acuerdo a la dirección se calcula la dirección de avance

ava_paso = 2 (Cada movimiento será de 2 pixeles)

inicializa ( )

Tomar la imagen del vehículo

Repetir

Sensar las paredes de los lados

Si no existe una pared hacia la derecha o izquierda entonces

Salir del lazo

Fin Si

Ubicar la imagen del vehículo en la nueva posición

Actualizar la posición del vehículo dando un nuevo paso

Mientras que el vehículo realice menos de 90 movimientos

Definición del puerto visual del vehículo

Determinación de la dirección y ubicación del vehículo en ava_ubi

Retornar ava_ubi

Fin

banderas Carro - puesto (int pue_dir, int pue_sentido)

(pue_dir indica la dirección del vehículo)

(pue_sentido indica si se sensa paredes adelante o atrás)

Inicio

banderas pue_ban

De acuerdo al sentido y dirección del vehículo, determinar si existe

una pared hacia la derecha, adelante o atrás en pue_ban

Retornar pue_ban

Fin

LIBRERIA TEXTO

Clase Letras

Miembros privados:

int let_fract[15]

chr let_arr[25]

Miembros públicos:

Letras (int, int)

void let_graf (int, int, char *let_txt, int, int)

Letras - Letras (int let_scale, int let_fact)

(let_scale indica la escala a aplicar)

(let_fact es un contador para la estructura de la letra)

Inicio

Construir la estructura de la letra de acuerdo a let_scale y let_fact

Asignar cada una de las letras del abecedario al arreglo let_arr

Fin

void Letras - let_graf (int let_xini, int let_yini, char *let_txt, int let_col, int let_pat)

(let_xini y let_yini indican la ubicación en X y Y del texto)

(let_txt es el texto a graficar)

(let_col es el color del texto)

(let_pal es el patrón del texto)

Inicio

Determinar color y tipo de línea a utilizar

Cálculo de la longitud del texto

Para let_i=1 mientras que let_i<long. del texto entonces let_i=let_i+1

Tomar la letra en la posición let_i del texto

Ubicarse en la posición indicada en la pantalla

Graficar la letra correspondiente

Rellenar el interior de la letra con el patrón indicado

Ubicarse en la posición de la siguiente letra

Fin Para

Fin

PROCEDIMIENTOS DEL PROGRAMA PRINCIPAL

void Pantalla1 ( )

Inicio

Mostrar en pantalla la presentación de la Universidad y Facultad

Fin

void Pantalla2 ( )

Inicio

Mostrar en pantalla el Título de la aplicación y autor

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

Pedir una tecla

Fin mientras

Fin

void Pantalla3 ( )

Inicio

Mostrar en pantalla el antecedente y objetivo de la aplicación

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

Pedir una tecla

Fin mientras

Fin

void Pantalla4 (Letras)

Inicio

Mostrar en pantalla tres opciones:

RESOLUCION MANUAL

RESOLUCION AUTOMATICA

SALIR

Repetir

Resaltar una opción

Pedir el ingreso de una tecla (Cursor hacia arriba o hacia abajo)

Si no es <Enter> entonces

Resaltar la nueva opción de acuerdo a la tecla ingresada

Fin Si

Mientras no se presione <Enter>

Si la resolución es manual

Mostrar en pantalla las intrucciones para el uso del teclado

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

Pedir una tecla

Fin mientras

Fin Si

Fin

void Pantalla5 ( )

Inicio

Mostrar en pantalla cinco opciones:

LABERINTO 1

LABERINTO 2

LABERINTO 3

LABERINTO 4

ALEATORIO

Repetir

Resaltar una opción

Pedir el ingreso de una tecla (Cursor hacia arriba o hacia abajo)

Si no es <Enter> o <Esc> entonces

Resaltar la nueva opción de acuerdo a la tecla ingresada

Fin Si

Mientras no se presione <Enter> o <Esc>

Si se presionó <Esc> entonces

Retornar 0

sino

Retornar No. laberinto o 5 para ALEATORIO

Fin Si

Fin

void Pantalla6 (char *xtiempo)

Inicio

Mostrar en pantalla el mensaje de haber resuelto el laberinto

Mostrar en pantalla el tiempo de resolución en segundos

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

Pedir una tecla

Fin mientras

Fin

void Pantalla7 (Letras)

Inicio

Mostrar en pantalla el mensaje de FIN de aplicación

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

Pedir una tecla

Fin mientras

Fin

void Puntaje (char *xtiempo, int xlaber)

Inicio

Si xtiempo < calif[xlaber-1].sc_tiem

(calif.sc_nombre indica el nombre del operador)

(calif.sc_tiem indica el tiempo de resolución)

(calif.sc_labe indica el No. de laberinto resuelto)

Pedir el ingreso de máximo 10 letras (nombre del operador)

Reemplazar en el arreglo el nuevo nombre y tiempo

Fin Si

Mostrar en pantalla los nombres de las personas con mejores tiempos

de resolución para cada laberinto

Mientras no se tipee cualquier tecla

Mostrar mensaje de ingreso de tecla

```
Incializar car
Determinar posición inicial del vehículo de acuerdo a numlab
Graficar pantalla
Graficar sección de laberinto
Posicionar vehículo
Graficar vehículo
Inicializar arreglo de control que registra movimiento y marca
Repetir
    Si la resolución es manual
        Enviar señal de paro
        Repetir
            pide_tecla
        Mientras sea una tecla válida
        Si (direc.=1 y tecla=77 o direc.=2 y tecla=75 o
            direc.=3 y tecla=72 o direc.=4 y tecla=80) entonces
            Enviar señal de avance
            Ejecutar Avance del vehículo
        Fin Si
        Si (direc.=1 y tecla=80 o direc.=2 y tecla=72 o
            direc.=3 y tecla=77 o direc.=4 y tecla=75) entonces
            Enviar señal de rotación adelante derecha
            Ejecutar Rotación Adelante Derecha del vehículo
        Fin Si
        Si (direc.=1 y tecla=72 o direc.=2 y tecla=80 o
            direc.=3 y tecla=75 o direc.=4 y tecla=77) entonces
            Enviar señal de rotación adelante izquierda
```

```
        Ejecutar Rotación Adelante Izquierda del vehículo
    Fin Si
    Si (direc.=1 y tecla=75 o direc.=2 y tecla=77 o
        direc.=3 y tecla=80 o direc.=4 y tecla=72) entonces
        Enviar señal de retroceso
        Ejecutar Retroceso del vehículo
    Fin Si
    Si (direc.=1 y tecla=145 o direc.=2 y tecla=141 o
        direc.=3 y tecla=116 o direc.=4 y tecla=115) entonces
        Enviar señal de rotación atrás derecha
        Ejecutar Rotación Atrás Derecha del vehículo
    Fin Si
    Si (direc.=1 y tecla=141 o direc.=2 y tecla=145 o
        direc.=3 y tecla=115 o direc.=4 y tecla=116) entonces
        Enviar señal de rotación atrás izquierda
        Ejecutar Rotación Atrás Izquierda del vehículo
    Fin Si
sino
    Sensar paredes
    Si no existe apared adelante o derecha o izquierda entonces
        Si no existe pared adelante entonces
            Ejecutar Avance del vehículo
            Registrar movimiento = 1
            Si no existe pared derecha registro marca = 1
            Si no existe pared izquierda registro marca = 2
        sino
```

```
                Si no existe pared derecha o izquierda
                    Si no existe pared derecha
                        Ejecutar Rotación Adelante Derecha
                        Registrar movimiento = 2
                    sino
                        Ejecutar Rotación Adelante Izquierda
                        Registrar movimiento = 3
                    Fin Si
                    Si no existe pared derecha e izquierda
                        Registrar marca = 2
                    Fin Si
                Fin Si
            Fin Si
        sino
            Mientras marca <> 0
                Revertir el movimiento registrado
                Registrar marca = 0 y movimiento = 0
            Fin Mientras
            Ejecutar movimiento en otro sentido al tomado al inicio
        Fin Si
    Fin Si
    Verificar si se resolvió el laberinto
    Si no se lo ha resuelto
        Graficar pantalla
        Graficar nueva sección de laberinto
        Posicionar vehículo
```

```
                    Graficar vehículo
                Fin Si
            Hasta que no salga del laberinto
            Si el tipo de resolución fue manual
                Calcular tiempo de resolución
                Pantalla6 (tiempo)
                Puntaje (tiempo, numlab)
            Fin Si
        Fin Si
        tipres = Pantalla4 (titulo)
    Fin Mientras
    Pantalla7
    Cerrar archivos de tiempos de resolución
    Eliminar archivos temporales
Fin del Programa Principal
```