UNIVERSIDAD POLITÉCNICA DE PUEBLA

PROGRAMA ACADÉMICO DE
INGENIERÍA EN INFORMÁTICA

Evasión de Obstáculos a través de Sensores de Proximidad

ALUMNO:
Amador Flores Flores

Reporte Técnico PII-26-08-09

COMITÉ EVALUADOR
Dr. Antonio Benítez Ruíz (*Asesor*)
Dr. Pedro Vargas García
Dra. María Auxilio Medina Nieto

PROFESORA DE PROYECTO DE INVESTIGACIÓN II

Dra. María Auxilio Medina Nieto

Agosto 2009

Índice de Figuras

Índice de Tablas

Resumen

Este proyecto de investigación describe la programación de un PIC que almacena comportamientos básicos de evasión de obstáculos para un robot tipo vehículo a través de la entrada de señales enviadas por un sensor ultrasónico. El programa para el PIC está codificado en ensamblador. Así mismo, se propone el diseño de una tarjeta de control para realizar la comunicación entre PIC – Sensor y PIC – Tarjeta de control de motores tomando en cuenta los elementos necesarios para su realización. El proyecto incluye una documentación detallada del funcionamiento del PIC 16f877A y del sensor SRF05.

CAPÍTULO 1. Planteamiento del problema

1.1 Introducción

La robótica se ha convertido en una de las áreas más sobresalientes dentro de la automatización de los procesos, para así lograr la mayor productividad posible con la mínima intervención del hombre. La mayoría de los sistemas robotizados operan en fábricas, donde el espacio de trabajo ha sido ideado para adecuarse al robot [Ángel S. 2005].

Actualmente, los robots tipo vehículo han tomado gran importancia tanto para el transporte de personas y carga en general, como dentro de aplicaciones de exploración de ambientes. Estos desarrollos han llevado a la necesidad de crear vehículos autónomos capaces de conducirse por carretera sin conductor [Aníbal O. 2001]

El presente proyecto utiliza un robot tipo vehículo que será dotado con sensores de proximidad de tipo ultrasónico. El principal objetivo de dotar a un robot con este tipo de sensores es poder detectar los objetos que lo rodean para identificar sus posiciones dentro de un ambiente controlado, además, el robot actuará como consecuencia de la detección de estos objetos utilizando una rutina de evasión de obstáculos.

1.2 Objetivo general

Dotar a un vehículo todo terreno con la capacidad de evadir obstáculos a través del diseño de un sistema que recupere y procese las señales provenientes de un sensor de proximidad y ejecute rutinas de comportamiento desde un PIC.

1.3 Objetivos específicos

- Entender y describir el funcionamiento del PIC para implementar rutinas de evasión de obstáculos que se puedan mandar a ejecutar desde él.
- Entender y describir el funcionamiento del sensor de proximidad SFR05 para diseñar una tarjeta de control para la recuperación de señales desde el sensor y su interfaz con el PIC.
- Diseñar interfaz entre el PIC y la tarjeta de control de motores para el envío de señales.

1.4 Justificación

Este trabajo es sólo una parte de un proyecto a mayor escala que pretende desarrollar un robot tipo vehículo. Así el proyecto titulado "Evasión de obstáculos a través de sensores de proximidad" combina elementos de hardware como el diseño de la tarjeta de control para dichos sensores con programas de software que permitirán controlar estos dispositivos. Con esto, se propone un proyecto que genera una estrecha relación entre la parte física y la programación de un comportamiento lógico. Estos conocimientos son indispensables dentro de la formación de un ingeniero en informática, debido a que los requerimientos de las grandes industrias se basan actualmente en la automatización de los procesos a través del manejo de robots o de dispositivos mecánico-electrónicos que deben de realizar tareas específicas.

Los robots tipo vehículo tienen gran impacto dentro del campo de la investigación, un claro ejemplo son las exploraciones de lugares que el hombre intenta conocer y que por diferentes circunstancias no puede hacerlo. Todo esto conlleva a la creación de robots de exploración mas sofisticados que puedan detectar los objetos que existen en el ambiente que los rodea para utilizar esa información de diferentes maneras. La integración de sensores de proximidad en el robot de este proyecto, permitirá obtener información relacionada con posibles obstáculos dentro del ambiente de trabajo; ésta es una de las tareas más importantes dentro del campo de la exploración.

1.4 Cronograma

Las actividades principales del proyecto se ilustran en la Tabla 1 y 2.

Tabla 1. Actividades principales para Proyecto de Investigación 1.

	Enero				*Febrero*				*Marzo*			*Abril*				
Actividad	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4
Analizar el proyecto	X	X														
Presentar la propuesta de protocolo			X	X	X											
Buscar documentación del sensor						X	X	X								
Comprender el funcionamiento del PIC									X	X	X					
Estudiar los algoritmos propuestos por Braitenberg												X	X			
Entender cómo funciona la recuperación de señales del sensor														X	X	X

Tabla 2. Actividades principales para Proyecto de Investigación 2.

	Mayo				*Junio*				*Julio*				*Agosto*			
Actividad	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4	S 1	S 2	S 3	S 4
Seleccionar los pines de E/S del PIC	X	X														
Analizar el funcionamiento del MPLAB IDE			X	X												
Implementar un programa en ensamblador con comportamientos de evasión de obstáculos					X	X	X									
Probar el programa en ensamblador y verificar las E/S								X	X	X	X					
Proponer un diseño de un a tarjera de control para montar el PIC												X	X	X		
Proponer diseño de una interfaz PIC – Motores y PIC - Sensor														X	X	X

1.6 Recursos

Recursos de hardware:

- Computadora con sistema operativo Windows XP,
- Procesador Pentium 4
- 500 Megabytes en memoria RAM
- Robot tipo vehículo
- Programador de PIC modelo MPLAB ID

Recursos de software:

- Compilador C versión 5.02
- Sensor de proximidad ultrasónico MaxSonar-EZ1
- Compilador MPLAB versión 8.3
- Herramienta de diseño de circuitos Proteus versión 7.0

1.7 Alcances y limitaciones

Dentro de los alcances se espera lo siguiente:

- ✓ Tener un vehículo dotado con sensores que le permitan evadir obstáculos
- ✓ Crear rutinas de software para el control del envío y recepción de señales
- ✓ Tener la rutina de evasión de obstáculos en un PIC
- ✓ Diseñar y construir una interfaz entre el PIC y la tarjeta controladora de motores
- ✓ Crear la documentación técnica del proyecto “Evasión de obstáculos a través de sensores de proximidad”

En cuanto a las limitaciones, están las siguientes:

- No encontrar documentación suficiente referida al tipo de sensor escogido
- Existe una amplia gama de PICs en el mercado por lo que sólo se ha contemplado estudiar un modelo de PIC específico.
- El robot tendrá un buen desempeño sólo en ambientes con ausencia de ruido.

CAPÍTULO 2. Marco teórico

2.1 Robótica reactiva

En la década de los 80 surgió una nueva corriente dentro de la robótica denominada robótica reactiva. Esta corriente fue encabezada por Brooks [Rodney 1991]. El enfoque de Brooks consistía en intentar descomponer el razonamiento humano en partes más sencillas con la idea de poder realizar un estudio más minucioso de estas partes y comprenderlas. De esta forma, algún día la integración de estos elementos aislados podría llegar a generar una máquina inteligente.

Por otro lado, Brooks también pensaba que jamás se podría conseguir crear una máquina con este procedimiento y lo más indicado sería comenzar por desarrollar comportamientos más sencillos, por ejemplo, los relacionados con los insectos. De esta forma, se podría ir avanzando según los resultados que se encontraran y por los conocimientos nuevos acerca del pensamiento.

Todo esto conllevo a la propuesta de una arquitectura de la robótica reactiva. Se denomina reactiva porque el robot no tiene ningún modelo del entorno, sino que reacciona ante los estímulos de éste. La descomposición de las tareas se realiza en tareas más simples que se ejecutan en paralelo, buscando ciertas características del entorno y actuando cuando éstas aparecen. De este modo, las distintas sub tareas compiten por el control de los actuadores del robot.

La arquitectura reactiva propone los siguientes puntos:

- ✓ La tarea se descompone en pequeñas sub tareas ó comportamientos
- ✓ Los comportamientos se procesan en paralelo
- ✓ Los comportamientos compiten por el control de los actuadores
- ✓ Existe un arbitraje que controla esta competencia
- ✓ En muchos casos, los comportamientos se pueden implementar con facilidad como autómatas de estados finitos

2.2 Sensor ultrasónico SFR04

Para poder obtener información del medio que rodea al robot, es necesario utilizar algún tipo de dispositivo. En este caso se escogió el sensor ultrasónico SFR04. Una de las principales ventajas que poseen los sensores ultrasónicos contra los infrarrojos (sensores de luz) es que los últimos no funcionan bien en lugares donde se encuentra presente la luz solar. Debido a esto, los sensores infrarrojos pueden generar una información incorrecta y por lo tanto, no se obtendrían datos reales del mundo exterior.

El sensor ultrasónico SFR05 es una versión mejorada del modelo SFR04, es altamente compatible con este último. Algunas de las ventajas más sobresalientes es que ha incrementado el rango de detección, ya que en el modelo anterior percibía objetos localizados en un máximo de tres metros y el modelo actual alcanza un rango de cuatro metros. Tiene dos modos de configuración: la primera es configurar dos pines, uno para el envío de la señal ultrasónica y otra para la recepción de dicha señal enviada; el segundo modo consiste en utilizar un solo pin tanto para el envío como para la recepción [Robot 2009]. En la Figura 1 se puede apreciar la vista frontal como la vista trasera del sensor SRF05.

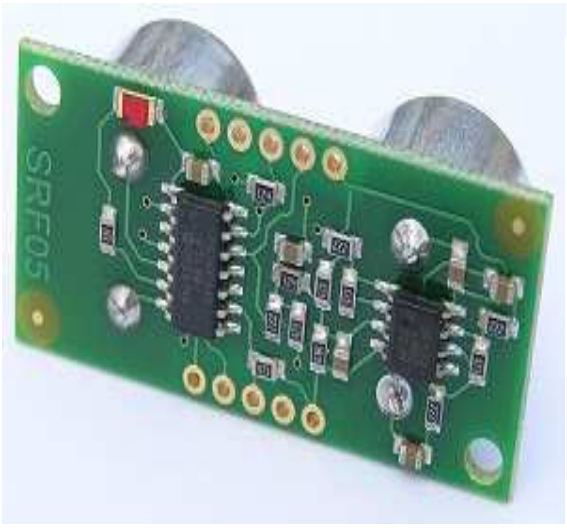

Figura 1. Vista frontal y trasera del sensor ultrasónico SFR05

Para la programación se utilizará el modo 1, en el cual se separa tanto el envío como la recepción de la señal. Para lograr esta configuración, es necesario dejar desconectado el modo pin del sensor. La conexión completa se muestra en la Figura 2.

Figura 2. Esquema de Conexión para configurar el sensor en modo 1

La Figura 3 muestra una gráfica para poder entender mejor la forma en que se comporta el sensor SFR05 y el tiempo que tarda en reaccionar ante un objeto.

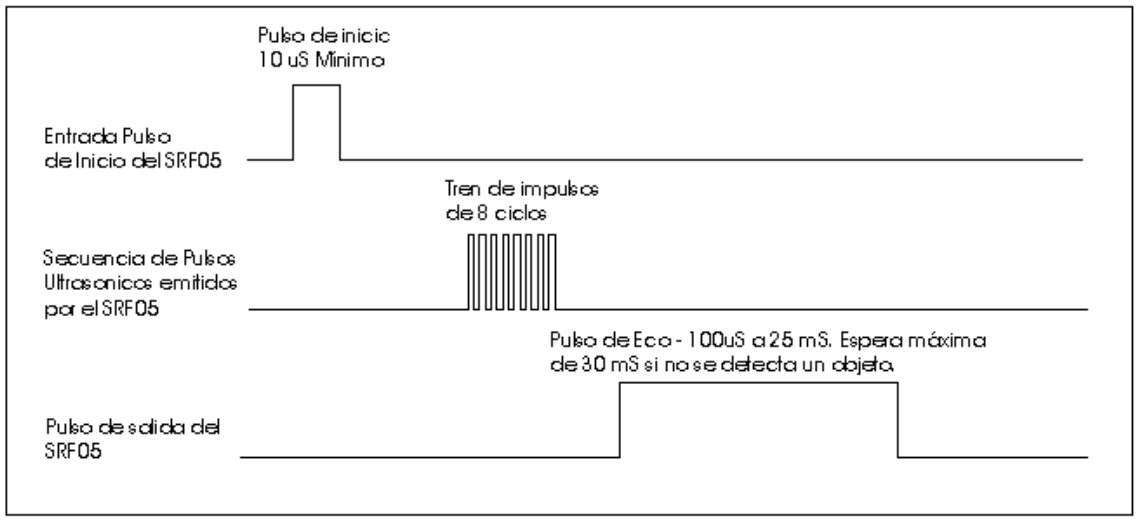

Figura 3. Gráfica que muestra los tiempos de reacción del sensor SRF05.

La Figura 3 muestra una gráfica que indica que para poder comenzar a detectar un objeto, lo primero que se debe hacer es generar un pulso de entrada de al menos 10 microsegundos. Este pulso indicará al sensor que debe comenzar su funcionamiento. Una vez que se ha generado el pulso de entrada, el sensor SRF05 disparará una ráfaga de 8 ciclos de ultrasónico a 40khz. En el momento en que el sensor escucha un eco se activa el pin, esto significa que el pin de eco también ha generado un pulso.

El pulso de salida del sensor SFR05 tiene una espera máxima de detección de 30 milisegundos, una vez pasado ese tiempo, el sensor da por echo que no se ha detectado el objeto y debe de generarse otro pulso de entrada hacia el sensor. Este procedimiento se repite para poder obtener los valores deseados [Robot 2009].

El ancho del pulso generado en el pin de eco es proporcional a la distancia a la que se encuentra el objeto, este dato es muy importante ya que a partir de él se puede calcular la distancia a la que se ha detectado el objeto. El cálculo de la distancia puede realzarse en cualquier unidad de medida (centímetros, pulgadas, yardas, etcétera).

La forma de calcular la distancia a la que se encuentra el objeto es registrando la duración que tuvo el pulso de entrada. En el caso de que el ancho del pulso se mida en microsegundos, se debe tomar el tiempo que tardó en generarse tal pulso y al resultado dividirlo entre 58 para obtener el equivalente en centímetros o entre 148 si es que se desea obtener la distancia equivalente en pulgadas.

Otro dato interesante es que el sensor SRF05 puede activarse cada 50mS o 20 veces por segundo. Se recomienda un retardo de 50 microsegundos antes de realizar la siguiente activación, incluso si el SRF05 detecta un objeto cerca y el pulso del eco es más corto. Con esto se garantiza que el eco generado por el sensor ha desaparecido totalmente y así se evita obtener una falsa señal de eco durante la siguiente medición [Robot 2009].

2.3 Programador de PIC MPLAB IDE

MPLAB-IDE es una plataforma de desarrollo integrada bajo Windows, la cual permite escribir programas para PICs (Controlador de Interfaz Periférico) en lenguaje ensamblador o C, crear proyectos, ensamblar, compilar, simular el programa y programar el componente [MPLAB 2009]. MPLAB-IDE permite editar un archivo, ensamblarlo y simularlo en pantalla, brindando también la facilidad de ejecutarlo paso a paso para depurarlo y verificar su funcionamiento, se pueden observar también los registros internos, la memoria RAM y/o EEPROM de usuario como la memoria de programa, según se ejecuten las instrucciones. El entorno que utiliza es similar al de un emulador.

Componente de MPLAB-IDE [MPLAB 2009]:

- *Editor*.- Permite escribir y editar programas u otros archivos de texto
- *Project manager*.- Organiza los distintos archivos relacionados con un programa en un proyecto. Además, se puede crear un proyecto, editar y simular un programa. A lo sumo, brinda la facilidad de crear archivos, objetos y bajar archivos hacia emuladores (MPLAB-ICE) o simuladores de hardware (SIMICE)
- *Simulador*.- Simula eventos discretos para programas con puntos de ejecución, permite examinar/modificar registros, observar variables, tiempos y simular estímulos externos
- *Ensamblador***.-** Genera archivos que pueden ser descargados (almacenados) en un PIC
- *Ligador*.- Brinda la ventaja de unir varios archivos objetos en uno solo, generados por el ensamblador o compiladores C como MPAB-C18 o compiladores de terceros

- *Programador*.- Trabaja con programadores como los siguientes: PICSTART Plus, MPLAB ICD 2, MPLAB PM 3 y PRO MATE II

2.4 Descripción del PIC 16F877A

Para la programación de los comportamientos, se escogió el PIC 16F877A, el cual es un modelo creado por Microchips. Este modelo posee varias características que hacen que sea una buena opción a elegir para la implementación de programas. A continuación se enlistan las principales características con que cuenta el PIC 16F877A:

- Procesador de arquitectura RISC avanzada
- Juego de 35 instrucciones con 14 bits de longitud, todas ellas se ejecutan en un ciclo de instrucción, a excepción de las de salto que tardan 2 ciclos.
- Frecuencia de 20 Mhz
- Hasta 8K palabras de 14 bits para la memoria de código, tipo flash
- Hasta 368 bytes de memoria de datos RAM
- Hasta 256 bytes de memoria de datos EEPROM
- Hasta 14 fuentes de interrupción internas y externas
- Pila con 8 niveles
- Modos de direccionamiento directo, indirecto y relativo
- *Watchdog Timer* (WDT)
- Código de protección programable
- Modo *sleep* de bajo consumo
- Programación serie en circuito con 2 patitas
- Voltaje de alimentación comprendido entre 2 y 5.5 voltios
- Bajo consumo (menos de 2 mA a 5 V y 5 Mhz).

Físicamente, el PIC seleccionado cuenta con 40 pines, algunos de éstos fungen como pines de entrada o salida digital, otros como entrada o salida analógica, otros más son los encargados de alimentar de corriente a este dispositivo. Estos pines se describen brevemente en la Tabla 1. La Figura 4 muestra la distribución de los pines y sus correspondientes nombres.

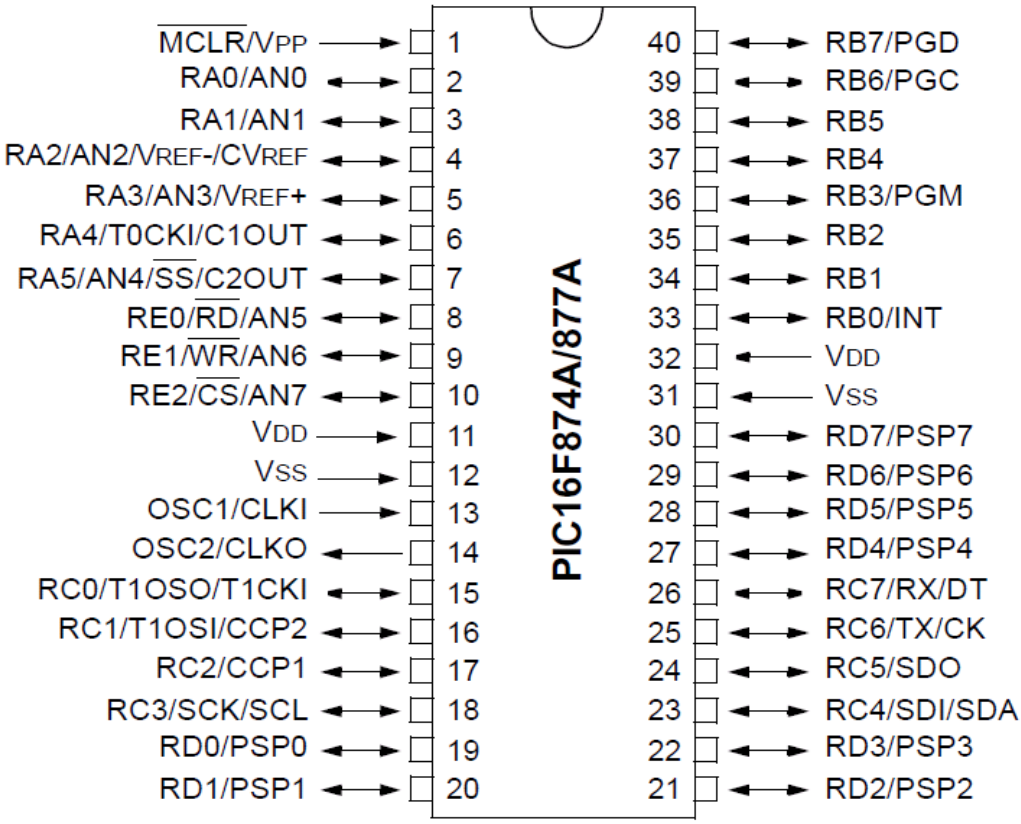

Figura 4. Distribución de los pines del PIC 16F877A

Tabla 1. Descripción de los pines del PIC 16F877A

Pin	Descripción
OSC1/CLKIN(9)	Entrada para el oscilador o cristal externo.
OSC2/ CLKOUT (10)	Salida del oscilador. Este pin debe conectarse al cristal o resonador. En caso de usar una red RC, este pin se puede usar como tren de pulsos o reloj cuya frecuencia es 1/4 de OSC1
MCLR/ VPP/THV(1)	Este pin es el reset del micro controlador, también se usa como entrada o pulso de grabación al momento de programar el dispositivo.
RA0/AN0(2) - RA3/AN3/VREF+(5)	Puede actuar como línea digital de E/S o como entrada analógica del convertidor AD (canal 0, 2 y 3)

RA4/T0CKI (6)	Línea digital de E/S o entrada del reloj del *Timer* 0. Salida con colector abierto
RA5/SS#/AN4(7)	Línea digital de E/S, entrada analógica o selección como esclavo de la puerta serie síncrona.
RB0 - RB7	Puerto B del pin 0 al 7 bidireccional.
RC0/T1OSO/T1CKI(11), RC1/T1OSI/CCP2(12)	Línea digital de E/S o salida del oscilador del *Timer* 1 o como entrada de reloj del *Timer* 1
RC2/CCP1(13)	E/S digital. También puede actuar como entrada captura 1,/salida comparación 1/ salida de PWM 1
RC3/SCK/SCL(14)	E/S digital o entrada de reloj serie síncrona /salida de los módulos SP1 e I2C.
RC4/SDI/SDA(15)	E/S digital o entrada de datos en modo SPI o I/O datos en modo I2C
RC5/SDO(16)	E/S digital o salida digital en modo SPI
RC6/TX/CK(17)	E/S digital o patita de transmisión de USART asíncrono
RC7/RX/DT(18)	E/S digital o receptor del USART asíncrono
RD0/PSP0- RD7/PSP7 (19-22, 27-30)	Las ocho patitas de esta puerta pueden actuar como E/S digitales o como líneas para la transferencia de información en la comunicación de la puerta paralela esclava.
RE0/RD#/AN5 - RE2/CS#/AN7	E/S digital o señal de lectura para la puerta paralela esclava o entrada analógica canal 5, 6 y 7
VSS(8,19)	Tierra.
VDD(20,32)	Fuente (5V).

El PIC16F877A contiene 5 puertos que pueden ser configurados como entrada o salida digitales (A, B, C, D, E). El puerto A contiene 6 bits (RA0-5). El puerto B (RB0-7), el puerto C (RC0-7) y el puerto D (RD0-7) tienen cada uno 8 líneas. El puerto E sólo cuenta con 3 líneas (RE0-2).

Es preciso mencionar que este PIC cuenta con un conjunto de 35 instrucciones para la programación. La razón para que sólo se utilicen 35 instrucciones en la programación es

que los PIC están basados en la arquitectura RISC, lo cuál significa que sus instrucciones están bastante optimizadas para así obtener una mayor velocidad de trabajo, una arquitectura más simple y un código más compacto. Las instrucciones con las que cuenta el PIC son muy parecidas a las utilizadas en el lenguaje ensamblador, este hecho hace que sea un poco más sencillo identificar la operación que se realiza e implementar un algoritmo propio de una forma más fácil.

A manera de resumen, y con la finalidad de dejar claro lo que hace cada una de las instrucciones del set que ofrece el PIC16F877A, la sintaxis que debe cumplir, el tiempo en ciclos de reloj que tarda en ejecutarse y los registros de estado afectados, se describen las instrucciones soportadas por el PIC en la Tabla 2.

Tabla 2. Conjunto de instrucciones del PIC 16F877A y su descripción

Nemónicos- Operandos		**Descripción**	**Ciclos**	**Estados afectados**
Instrucciones orientadas a registros				
ADDWF	f, d	Suma W y f	1	C, DC, Z
ANDWF	f, d	And de W con f	1	Z
CLRF	f	Borrar a f	1	Z
CLRW	-	Borrar a W	1	Z
COMF	f, d	Complementa f	1	Z
DECF	f, d	Decremento a f	1	Z
DECFSZ	f, d	Decremento a f, salta si el resultado es 0	1(2)	-
INCF	f, d	Incrementa a f	1	Z
INCFSZ	f, d	Incrementa a f, salta si el resultado es 0	1(2)	-
IORWF	f, d	Operación lógica O entre W y f	1	Z
MOVF	f, d	Mover el registro f	1	Z
MOVWF	f	Mover W a f	1	-
NOP	-	No operación	1	-
RLF	f, d	Rotar el registro f a la izquierda	1	C
RRF	f, d	Rotar el registro f a la derecha	1	C
SUBWF	f, d	Resta W de f	1	C, DC, Z
SWAPF	f, d	Intercambia los cuartetos de f	1	-
XORWF	f, d	OR Exclusivo de W con f	1	Z
Instrucciones orientadas a bits				
BCF	f, b	Pone a 0 el bit b de f	1	-
BSF	f, b	Pone a 1 el bit b de f	1	-

BTFSC	f, b	Chaca el bit b de f y salta si es 0	1(2)	-
BTFSS	f, b	Chaca el bit b de f y salta si es 1	1(2)	-
Instrucciones orientadas a constantes y operaciones de control				
ADDLW	k	Suma constante y W	1	C,DC,Z
ANDLW	k	AND entre constante y W	1	Z
CALL	k	Llama a una subrutina	2	-
CLRWDT	-	Borra el *Watchdog Timer*	1	TO.PD
GOTO	k	Salto incondicional	2	-
IORLW	k	Operación lógica O entre constante y W	1	Z
MOVLW	k	Mueve la constante a W	1	-
RETFIE	-	Regresa de la interrupción	2	-
RETLW	k	Regresa de subrutina y carga k en W	2	-
RETURN	-	Regresa de la subrutina	2	-
SLEEP	-	Entra en estado de reposo	1	TO.PD
SUBLW	k	Resta W de constante k	1	C,DC,Z
XORLW	k	Operación logica O exclusivo entre W y constante k	1	Z

Nota: Definiciones de las literales empleadas:
W: Registro de trabajo del procesador
f: Cualquier registro
k: Una constante o registro

2.5 Comportamientos propuestos por Braitenberg

En 1984, Valentino Braitenberg, uno de los pioneros de la cibernética y entonces director del Max Planck Institute for Biological Cybernetics (Tübingen, Alemania), publicó el libro titulado *"Vehicles: Experiments in Synthetic Psychology"* [Braitenberg 1984]. Este libro propone la construcción de una serie de robots tipo vehículo que se comportan de una manera inteligente y sofisticada con el propósito de colaborar con la investigación psicológica. También en este libro, Braitenberg describe varios experimentos en los que se construyen pequeños vehículos, de complejidad gradualmente creciente, con componentes mecánicos y eléctricos sencillos. Cada una de estos vehículos imita un comportamiento inteligente, al cual se le asocia un nombre. En total son 6 comportamientos: 1) tímido, 2) indeciso, 3) paranoico, 4) obstinado, 5) conductor y 6) atractivo y repulsivo (perseguidor y perseguido).

En este proyecto, sólo uno de los comportamientos podrá ser implantado en el robot debido a que tal comportamiento se analizó y se encontró como el más adecuado para simular que el robot tipo vehículo está evadiendo un obstáculo. En principio, se considerará el comportamiento paranoico y se dará una breve descripción en la siguiente sección.

2.5.1 Comportamiento paranoico

Un robot con comportamiento paranoico se modela como se muestra en la Figura 5. Cuando el vehículo paranoico entra en la zona de oscuridad, modifica su trayectoria y gira a la izquierda o a la derecha con el objeto de volver a la zona iluminada.

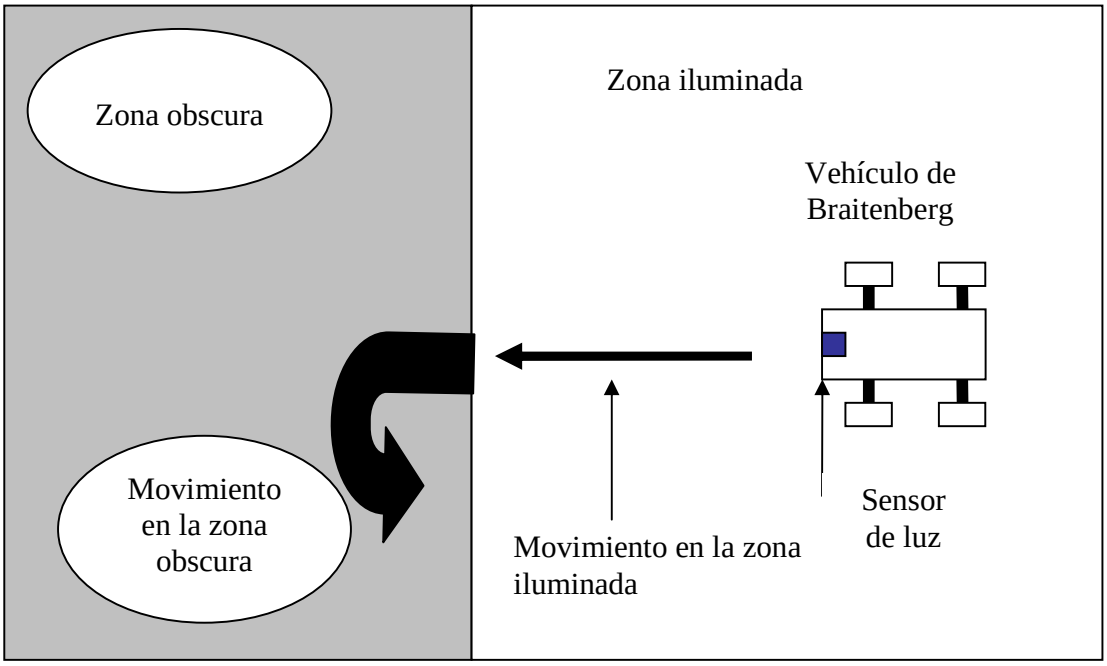

Figura 5. Comportamiento de vehículo paranoico [Braitenberg 1984].

CAPÍTULO 3. Diseño

3.1 Diseño de la tarjeta de control del PIC

Uno de los objetivos del proyecto es el diseño de la creación de una tarjeta de control para el robot tipo vehículo, para lo cual es necesario decir como estará conformada y cuales serán los componentes requeridos para poderla diseñar. En la Figura 6 se muestra una propuesta de diseño en forma de diagrama a bloques.

Tal y como se a mencionado en la justificación, este trabajo es una etapa más contemplada dentro de un proyecto más grande. La tarjeta de control de motores fue implementada en la primera parte del proyecto de investigación por un alumno de esta universidad [Alí 2009]. Con esto, es preciso destacar que el diseño en esta parte del proyecto contempla esa tarjeta de control de motores que actualmente se encuentra funcional.

Tarjeta de control

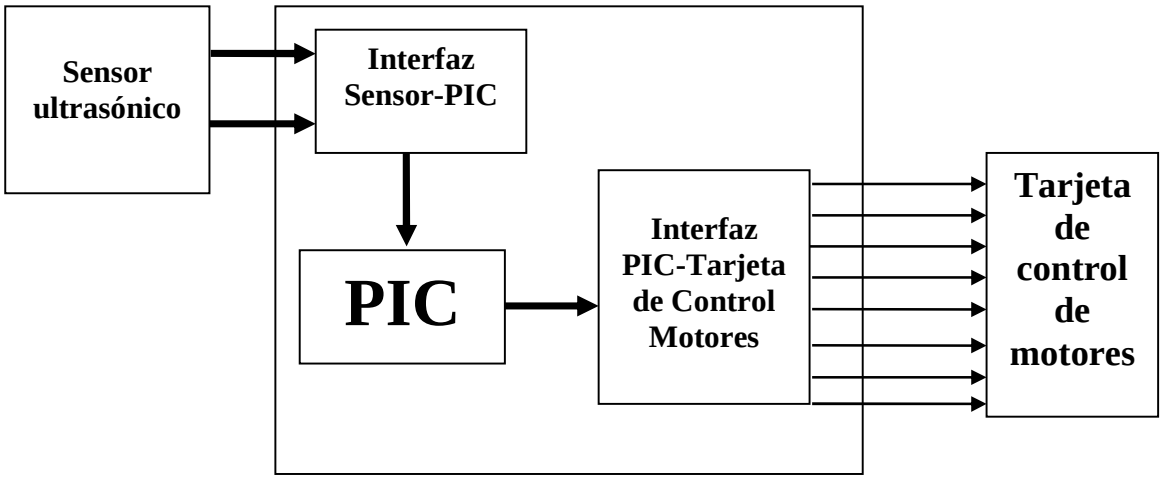

Figura 6. Diagrama a bloques de la tarjeta de control

Los elementos importantes que se han de incluir dentro de la tarjeta de control son: sensor ultrasónico, interfaz sensor-PIC, PIC, interfaz PIC-tarjeta de control de motores y tarjeta de control de motores. A continuación se describen estos elementos.

3.1.1 Sensor ultrasónico

Se utilizará un sensor ultrasónico, el cual, en caso de encontrar algún objeto dentro del mundo exterior, mandará una señal. Las señales que recibirá el sensor son digitales, por lo cual se debe de configurar al PIC con entradas y salidas digitales. Se tiene un diseño que contempla una interfaz para realizar la comunicación entre el sensor y el PIC, para que este último pueda interpretar las señales provenientes del sensor y responder con una rutina de atención a cada uno de los eventos que se diseñen. La forma en que se va a realizar la conexión del sensor se muestra en la Figura 7.

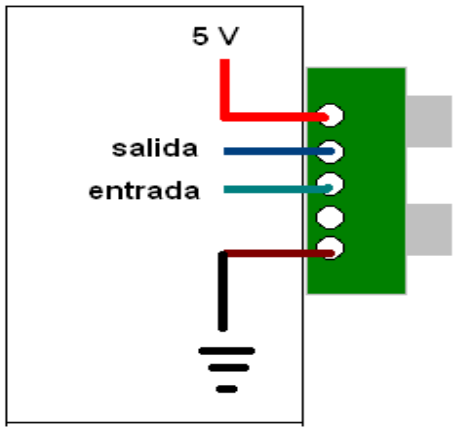

Figura 7. Conexión física del sensor SRF05

3.1.2 Interfaz sensor-PIC

La interfaz sensor-PCI será un circuito o un conjunto de circuitos integrados que se utilizarán para comunicar al sensor con el PIC (interfaz), esto en caso de que las señales que provienen del sensor no sean compatibles con las entradas que puede interpretar el PIC. Con esto se garantiza que las señales recibidas por el PIC son las indicadas. La propuesta de diseño se esquematiza en la Figura 8.

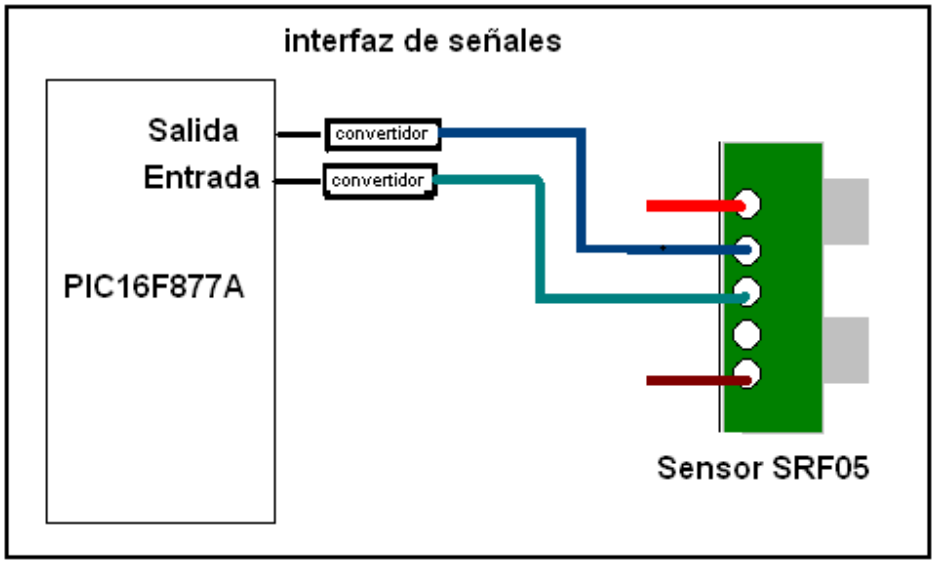

Figura 8. Interfaz entre el sensor ultrasónico y el PIC

3.1.3 PIC

Este proyecto utiliza el PIC 16F877A. Este microcontrolador es fabricado por la compañía Microchip. El modelo 16F877A posee varias características que hacen de este micro controlador un dispositivo muy versátil, eficiente y práctico para ser empleado en esta aplicación.

Es necesario destacar que la programación del PIC tiene como prioridad eliminar la dependencia que existe entre el robot y la CPU. Esta dependencia es a causa de que las rutinas o comportamientos que ejecuta el robot se encuentran almacenadas en una PC y son enviadas a través del puerto paralelo [Alí 2009]. Una vez que los programas se encuentren dentro del PIC, éste se encargará de seleccionar la acción correspondiente y mandará las señales a la tarjeta de control de actuadores para que se ejecuten sin la necesidad de hacer más conexiones a la CPU.

El PIC de la tarjeta de control no contendrá un único programa a ejecutar, sino un conjunto de programas o subrutinas. El programa principal será el encargado de la evaluación de la señal obtenida del sensor y de esto dependerá la selección de la subrutina a ejecutar. Este

conjunto de subrutinas pueden contemplar acciones como detener el vehículo, girarlo a la derecha, girarlo a la izquierda o ponerlo en reversa.

Cabe destacar que la tarjeta de control de motores recibe una secuencia de instrucciones en formato de 8 bits. Así mismo, se ha mencionado en el apartado 2.2 del capítulo del marco teórico, que el sensor cuenta con dos pines, uno que será el pin de entrada y otro que fungirá como pin de salida. Estos datos dan la pauta para concluir que se necesita un puerto para configurar entradas y otro para generar las salidas, en este caso, se propone al puerto A como entrada y salida y al puerto B como salida.

Las entradas y salidas del PIC son una parte imprescindible en todo sistema, ya que si no se identifican correctamente, puede que el sistema falle o no genere los resultados deseados.

Da manera gráfica, la forma en que quedará conectado el PIC para las E/S se muestra en la Figura 9.

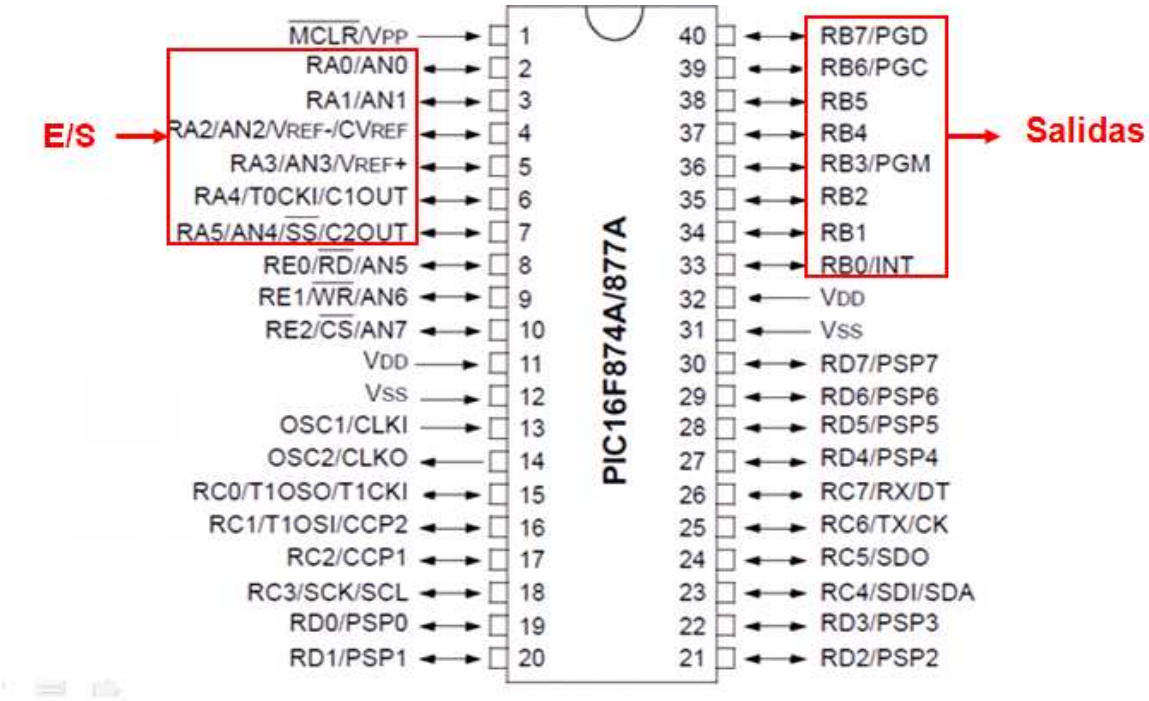

Figura 9. Puertos del PIC 16F877A que serán configurados como E/S

3.1.4 Interfaz PIC-tarjeta de control de motores

La interfaz PIC-tarjeta de control de motores será un circuito o un conjunto de circuitos integrados que se utilizará en el momento de transmitir las señales del PIC a la tarjeta de control de motores, ya que este último recibe entradas en formato de ocho bits.

A continuación, en la Figura 10 se muestra la forma en la que se realizará la comunicación entre el PIC y la tarjeta de control de motores.

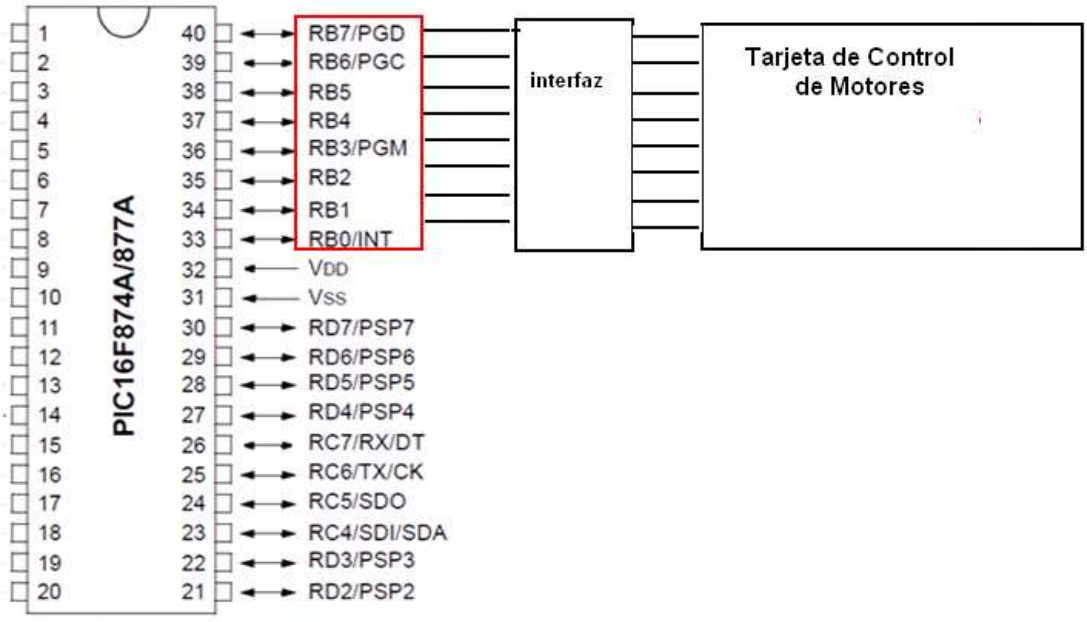

Figura 10. Diseño de comunicación entre el PIC y la tarjeta de control de motores

3.1.5 Tarjeta de control de motores

La tarjeta de control de motores ya se encuentra implementada dentro del robot, recibe señales provenientes de un programa almacenado en una CPU en formato de 8 bits. Esas señales son transferidas vía puerto paralelo. Así mismo, esta tarjeta ya puede ejecutar algunas rutinas implementadas por un programa creado en C++, el cual se encuentra almacenado en una PC y se comunica con ella a través del puerto paralelo. Lo que resta es convertir las señales del PIC al formato que entiende la tarjeta, de esta parte de encargará la

interfaz PIC-Tarjeta de Control de Motores. La descripción de esta parte ya se encuentra implementada en el proyecto de investigación anterior a esta fase [Alí 2009].

3.2 Acoplamiento

Una vez que se han conocido los elementos necesarios para el diseño de la tarjeta, se procede a crear un esquema que permita visualizar de forma general la ubicación de cada componente. Este esquema se puede apreciar en la Figura 11.

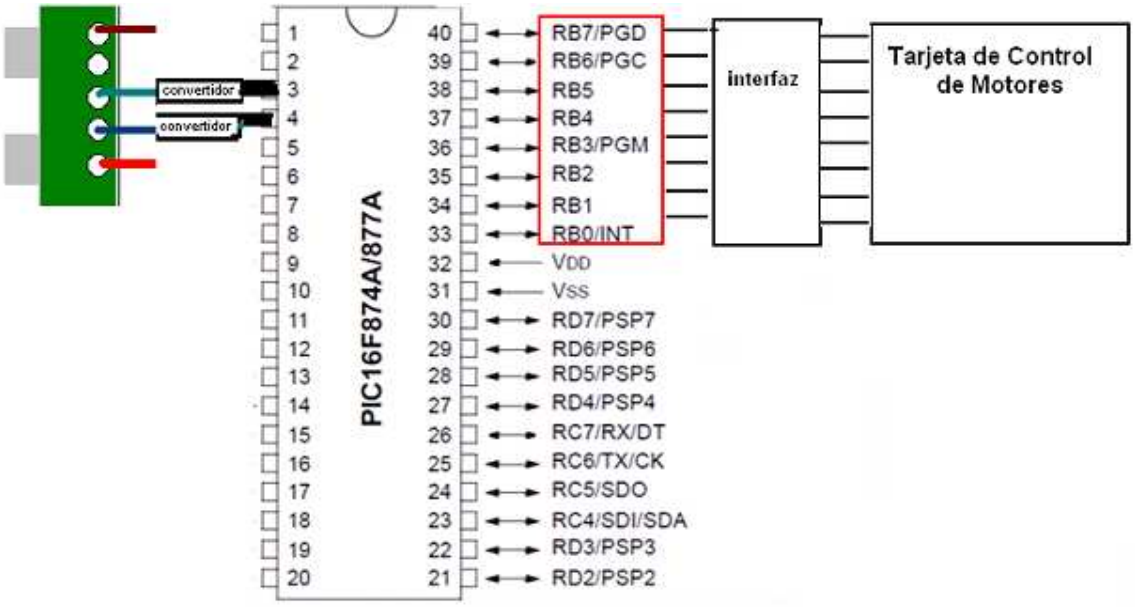

Figura 11. Diseño y conexiones físicas entre el PIC el sensor y la tarjeta de control de motores.

CAPÍTULO 4. Implementación y Prueba

4.1 Programación del PIC

Como parte de los objetivos específicos se encuentra la implementación de rutinas de evasión de obstáculos para el robot móvil. Estas rutinas deberán de ser ejecutadas desde un PIC. Para la programación del PIC se toma como punto de partida un programa en lenguaje C en el que ya se han implementado algunos movimientos básicos. Lo que se pretende hacer es traducir ese código a un lenguaje que pueda comprender el PIC (lenguaje ensamblador). El código a traducir se anexa en el apéndice A. Se utiliza la herramienta MPLAB descrita en el apartado 2.3 de este documento para implementar el código encargado de enviar y recibir señales para generar el o los comportamientos deseados.

4.1.1 Selección y configuración de entradas y salidas del PIC

El primer paso a seguir en el desarrollo del programa es identificar tanto las entradas como las salidas que tendrá el PIC. Como entrada, el PIC debe recibir las señales de un sensor y a partir de las mismas ejecutar una rutina específica. Por otro lado, las salidas deberán ser enviadas a la tarjeta de control de motores en un formato en el que la tarjeta pueda comprender.

El PIC recibirá dos señales, la primera que será un cero en el caso de que el sensor no encuentre nada a su paso y la segunda señal será 1 cuando encuentre un objeto. Se pueden escoger cualquiera de los puertos disponibles por el PIC y configurarlos como entrada y salida dependiendo de las necesidades del problema. En este proyecto se ha escogido al puerto A para ser configurado como entrada digital. Traduciendo esto a lenguaje ensamblador quedaría de la siguiente manera:

```
movlw b'00000111'                    ; PORTA como entrada
```

Donde el valor de ***b'00000111'*** depende del número de entradas que se deseen tener, en este ejemplo se toman tres bits como entrada que son los bits bajos puestos a uno.

Para el caso de las salidas, la entrada de la tarjeta de control de motores (actuadores) tiene un formato para la señal de 8 bits, por lo que se debe de escoger un puerto del PIC que sea capaz de generar como salida 8 bits. En este caso se tomará al puerto B y se configurará como salida digital. Para lograr esto, lo único que se debe de hacer es poner a cero todos los valores de *TRISB* como sigue:

```
clrf TRISB                           ; PORTB como salida
```

El código completo para la configuración de entrada y salida de los puertos A y B se muestra en la Figura 12.

```
1   CONF_ADCON1 equ b'00000110'      ; Configuración PORTA E/S digital
2   bsf STATUS,RP0
3   bcf STATUS,RP1
4   movlw CONF_ADCON1                ; Configurar el PORTA como digital
5   movwf ADCON1
6   movlw b'00000111'                ; PORTA como entrada
7   movwf TRISA
8       clrf TRISB                   ; PORTB como salida
```

Figura 12. Configuración de los puertos A y B como E/S digital

Una vez que se sabe lo que debe recibir el PIC (señal proveniente del sensor), es necesario tomar los valores del puerto A y generar un comportamiento.

4.1.2 Algoritmo propuesto para el PIC

De manera general, el PIC debe tener los comportamientos que se describen en el siguiente algoritmo:

1.- Al recibir en el puerto A un cero, se entiende que no se ha detectado ningún objeto cercano y por consiguiente el sensor debe enviar una señal a los actuadores que indica que debe de seguir avanzando.

2.- Al recibir en el puerto A un uno, se considerará que se ha detectado un objeto cercano y que el vehículo debe evadir tal objeto. Para lograr esto, se siguen las instrucciones siguientes:

   a. Detener el movimiento del vehículo
   b. El vehículo debe retroceder un poco
   c. El vehículo debe girar a la derecha
   d. El vehículo debe seguir avanzando

Se cuenta con una versión en ensamblador que simula el algoritmo descrito en esta sección. Ese programa se corre en el MPLAB y el código completo se encuentra en el Apéndice B.

Es necesario determinar cada cuánto tiempo se van a verificar los valores de entrada del puerto A, así como el tiempo que deberá tomarse como parámetro para enviar cualesquiera dos secuencias de bits a los motores. Esta subrutina se puede implementar creando dos ciclos anidados que terminen en un número de iteraciones determinadas, el código en ensamblador quedaría como se muestra en la Figura 13.

```
;.....  subrutina  delay  .....
Delay
        movlw   0x50
        movwf   CCP2CON  ; set outer delay loop
DelayOuter
        movlw   0xFF
        movwf   CCP1CON    ; set inner delay loop
DelayInner
        decfsz  CCP1CON
        goto    DelayInner
        decfsz  CCP2CON
        goto    DelayOuter
        return
;.....  Fin subbrutina delay  .....
```

Figura 13. Subrutina formada por dos ciclos anidados que dura un tiempo determinado

La rutina mostrada en la Figura13 funciona de manera correcta y genera un tiempo determinado, sin embargo, no hay garantía de que el tiempo que tarda en recorrer el ciclo es el que nosotros deseamos, ya que el tiempo de ejecución de esta rutina puede variar bastante y se tendrían que calcular los tiempos a prueba y error hasta alcanzar una aproximación del tiempo requerido.

Se ha sustituido la subrutina de retardo mostrada en la Figura 13 por una rutina que implementa un módulo propio del PIC, este módulo se le conoce como *Timer0*. La rutina implementada con *Timer0* se explicará en la siguiente sección.

4.1.3 Implementación de rutina de retardo utilizando el módulo Timer0

Una de las formas de solucionar el problema que se generaba con los ciclos anidados para generar un retardo de tiempo es utilizar un módulo del PIC llamado *Timer0***.** Este módulo puede ser configurado de dos formas, la primera es como un contador de eventos y la segunda es como un temporizador usado para generar periodos de tiempo, que es lo que se debe conocer para implementar retardos de tiempo. Esta y otras configuraciones se realizan a través del registro OPTION_REG del PIC.

El número de cuentas que debe realizar el Timer0 se carga a través del registro TMR0 el cual puede ejecutar hasta 256 cuentas debido a que es un registro de 8 bits. Este registro puede ser leído para conocer el valor actual y también puede ser escrito para iniciarlo en el valor que se desee.

El registro OPTION_REG está formado por 8 bits, cada uno de estos bits tiene una función específica habilitando ó deshabilitando una propiedad del módulo *Timer0.* En la Figura 14 se aprecia el orden en que se encuentran ubicados estos bits y sus respectivos nombres.

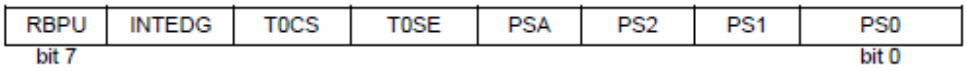

Figura 14. Orden y nombres de los bits del Registro OPTION_REG.

El significado se cada uno de los bits se menciona a continuación:

Bit 7 **RBPU**

Bit 6 **INTEDG**

Bit 5 **T0CS**: Bit selector de fuente para el TMR0

1 = reloj externo, pin RA4/T0CKI

0 = reloj interno (CLKOUT)

Bit 4 **T0SE**: Bit selector de flanco

1 = Incrementa en flanco de bajada en pin T0CKI

0 = Incrementa en flanco de subida en pin T0CKI

Bit 3 **PSA**: Bits de asignación del *prescaler*

1 = *Prescaler* es asignado al WATCHDOG

0 = *Prescaler* es asignado al módulo Timer0

Bit 2-0 **PS2**:**PS0**: Bits selectores relación de trabajo

Uno de los bits más interesantes a configurar es el bit (**PSA**) el cuál es un módulo que se comparte tanto para el módulo WATCHDOG como para el Timer0. El módulo *prescaler* funciona como una división de frecuencia. Lo cuál significa que si se activa el bit de

asignación del *prescaler* se tendrá una duración de cuenta o tiempo resultante de multiplicar el valor del módulo (ya sea WATCHDOG ó Timer0) por el valor del *prescaler*. El valor del *prescaler* se asigna a través de los bits **PS0, PS1 Y PS2** y la relación que existe al ingresar los valores se muestra el la Tabla 3.

Tabla 3. Relación entre el valor del *prescaler* y los valores de TMR0 y WDT.

Valor de los bits	Relación TMR0	Relación WDT
000	1:2	1:1
001	1:4	1:2
010	1:5	1:4
011	1:16	1:5
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:128	1:128

Para comprender mejor su funcionamiento, a continuación se describe un ejemplo: Supóngase que se tiene al módulo timer0 funcionando como un reloj, se habilita el *prescaler* con una relación 1:16 (la combinación brindada en PS0-PS2 es de “011”), esto quiere decir que por cada pulso que genere el TMR0 se generarán 16 pulsos más, pero el TMR0 solo incrementará en uno, al final el tiempo generado será una relación entre el valor del TMR0 que multiplica al valor del *prescaler* para así obtener la duración final.

Otro dato importante es que el tiempo de retardo que se debe asignar al TMR0 no es de manera directa, sino que se aplica realizando una operación de complemento a 2, lo cual significa que si se requiere realizar una cuenta de 2, se debe realizar el complemento a 2 de 256, que en este caso daría 254. Este valor es el que se debe cargar al registro TMR0 para así obtener el número de cuentas exacto.

Para poder saber si el registro TMR0 ha alcanzado el número de cuentas que se ha solicitado, se tiene al bit T0IF del registro INTCON. El bit TOIF genera una señal de interrupción cuando se desborda el registro TMR0, el desborde de este registro significa que a pasado de 0xFF a 0x00.

De forma general, existe una fórmula que describe los valores que deben ser asignados tanto al TMR0 como al *prescaler* para poder obtener el tiempo de retardo deseado. Los elementos necesarios para obtener esta fórmula son:

Retardo = 4*Tosc*TMR0*Prescaler.

donde:

- 4*Tosc = un ciclo de instrucción (donde Tosc es el inverso de la frecuencia del clock que usa el PIC)
- TMR0 = un numero X en complemento a 2 que se asigna al TMR0.
- *Preescaler* = valor del *prescaler* que puede ser ejecutar una cuenta entre 2 y 256 aproximadamente por cada pulso o cuenta del TMR0.

Ahora, como se tiene un clock o cristal de 4MHz conectado al PIC se tendría 1 microsegundo por cada cuenta del TMR0 con lo cuál se simplifica la fórmula a:

Retardo = TMR0*Prescaler

Teniendo más claro como funciona el módulo timer0, así como los registros necesarios para su funcionamiento y configuración; ahora se puede implementar una subrutina que utilice un tiempo determinado como retardo, en este caso se tiene la implementación de una subrutina que al ser llamada genera un tiempo de espera de 10 milisegundos. El código para generar este tiempo de espera se muestra en la **Figura 15.**

```
;.....      subrutina delay  con Timer0 10 microSec      .....
_retardo
            movlw b'11000000'          ;Configuración del modulo TMR0
            banksel OPTION_REG
            movwf OPTION_REG           ;Preescaler = 2
            banksel INTCON
            clrf INTCON                ;Deshabilitar interrupciones
            movlw d'249'               ;Cargar el valor de TMR0 para 5 cuentas
            movwf TMR0                 ;(Complemento)
_espera
            btfss INTCON.T0IF          ;Esperar desborde del TMR0
            goto _espera
            return                                  ;retorno de call
;.....
```

Figura 15: Subrutina que genera un tiempo de espera de 10 microsegundos.

4.2 Pruebas de Software

Dentro de las pruebas de software se contemplan dos tipos, las primeras fueron realizadas dentro de la herramienta de programación MPLAB, el cual, como se ha descrito en capítulos anteriores, sirve para crear proyectos, compilarlos y descargarlos al PIC. Por otro lado, la herramienta Proteus es más un estilo de diseño de circuitos en la cual se carga el programa al PIC y se muestran los archivos de forma gráfica.

4.2.1 Prueba de comportamientos en el MPLAB

Se creó un programa en esta herramienta cuyo código fuente se encuentra indexado en el Apéndice B. La herramienta MPLAB permitió escoger el tipo de componente que se iba a programar, que en este caso fue el PIC16f877A. Ésta herramienta posee varia ventajas, por ejemplo, se puede visualizar tanto los registros propios del PIC, como las variables que uno mismo declare, permite conocer el valor de los registros, depurar el programa

La Figura 16 muestra el ambiente de trabajo de la herramienta de programación.

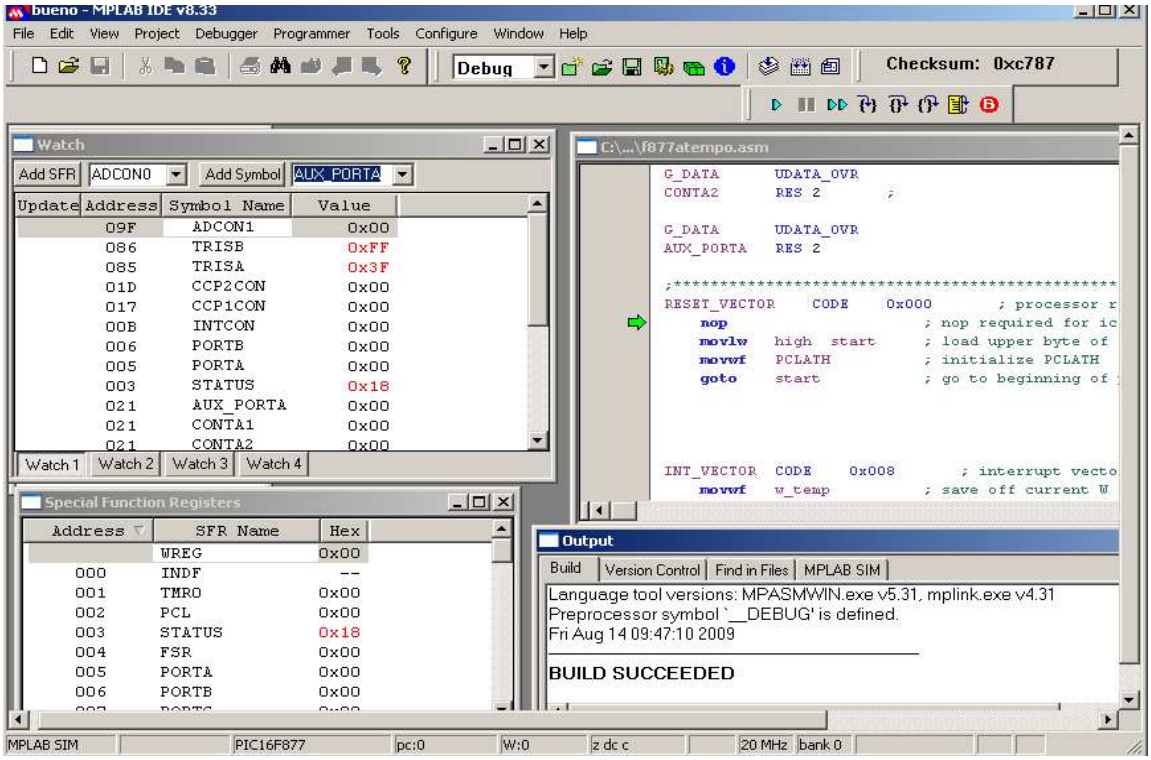

Figura 16: Ambiente de trabajo de MPLAB

La herramienta MPLAB es un IDE de programación y como tal, brinda una gran cantidad de funcionalidades que hacen que la interacción entre el programador y la aplicación sea más amigable.

En la Figura 16 se apreciar claramente como la herramienta permite mostrar varias ventanas a la vez, conociendo así partes que nos interesan. Por ejemplo, dentro de la ventana principal en la parte superior derecha, se muestra el código fuente, el cual se puede editar fácilmente (copiar, pegar, etcétera.).

En la parte superior izquierda (Figura 16), se encuentra una venta que muestra los registros tanto del Propio PIC, como de los que se han declarado por el Programador, sus direcciones físicas de memoria y los valores que tienen actualmente. Estos registros pueden ser agregados o eliminados de la ventana con el fin de hacer más personalizada la vista. Esta ventana es muy importante, ya que permite apreciar como se van modificando los registros en el momento que se realiza la ejecución del programa.

En la parte inferior izquierda (Figura 16), se encuentra otra ventana que a mi punto de vista es importante agregar, ya que permite mostrar de manera rápida y sencilla todos los registros especiales del PIC que se esta utilizando.

Para realizar la simulación del programa realizado, se utilizó un módulo llamado MPLAB SIM el cual agrega opciones como: correr el programa a pasos, simularlo, limpiar los registros de memoria, entre otros. La Figura 17 muestra el lugar donde se debe seleccionar esta opción.

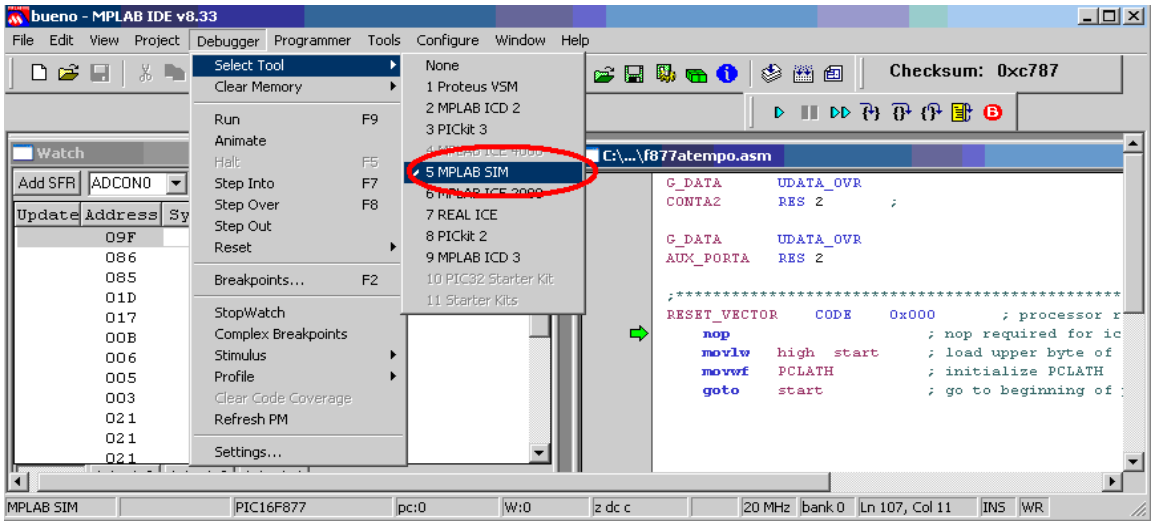

Figura 17: Selección de MPLAB SIM para poder simular el programa

Con el fin de probar una secuencia de entradas al PIC y conocer el comportamiento de los registros, se utilizó una opción que permite ingresar una secuencia de datos en formato de bits por un puerto específico. Par poder ingresar a este modo de configuración se tiene que elegir la opción de stimulus como se muestra en la Figura 18.

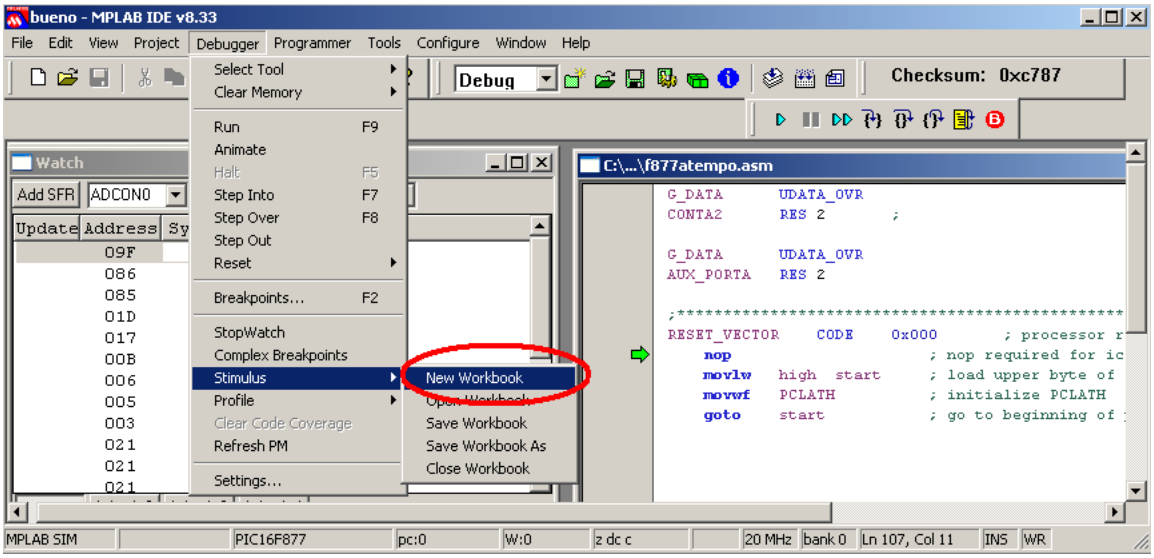

Figura 18: Selección de simulación con la opción stimulus.

Una vez realizado esto se selecciona el puerto en el que se va a trabajar y los valores que debe de tener. En este caso, se programó una secuencia de entrada para el Puerto A con los valores en el bit 0, 1 y 2, como lo muestra la Figura 19.

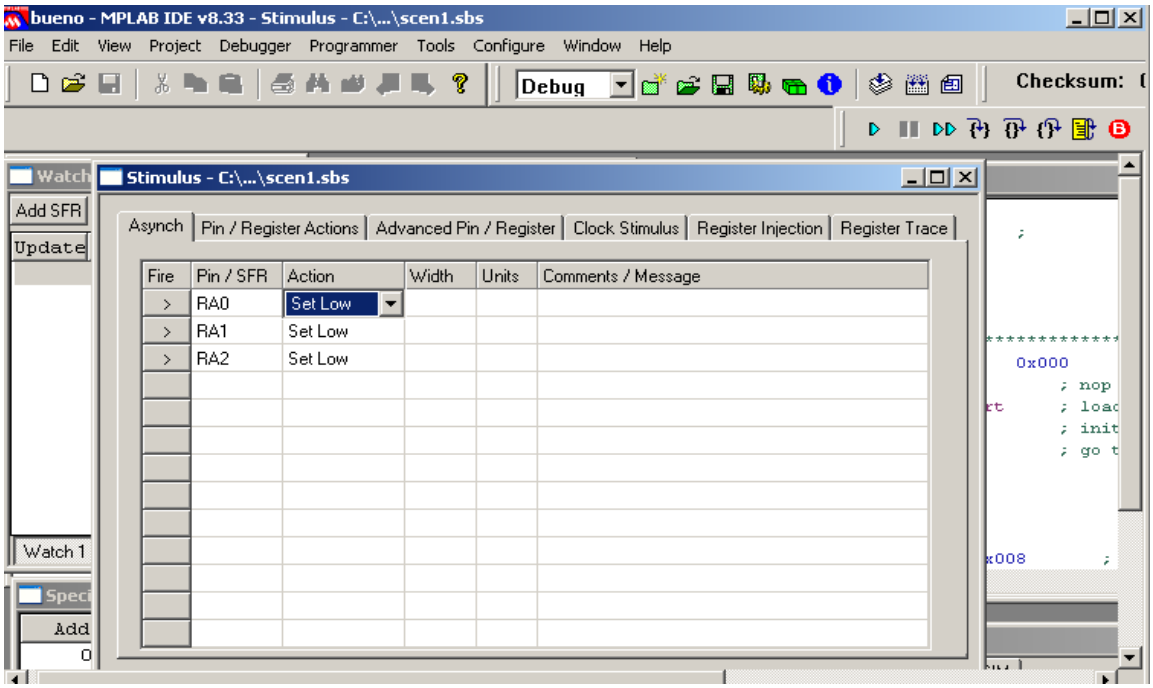

Figura 19: Configuración de los valores de entrada con stimulus

Se utiliza esta propiedad (simulación con stimulus) ya que los puertos que se configuran con entrada no pueden ser modificados de forma directa.

Todas las propiedades descritas en esta sección, permitieron que se tenga un código que funciona correctamente con las entradas y salidas propuestas. Se observó la ejecución de los comportamientos corriendo de manera deseada y que se esperaban logra en un principio.

4.2.2 Prueba de comportamientos en el Proteus

En esta sección se describirá una herramienta más para las pruebas de software llamada Proteus, esta herramienta esta disponible en la página oficial de microchips.

El manejo de esta herramienta es principalmente, para probar que las entradas y salidas, que se tienen configuradas dentro del programa desarrollado en MPLAB, estén ejecutándose correctamente, pero esta vez en un ambiente más gráfico.

El Proteus brinda la posibilidad de agregar diferentes dispositivos al proyecto ya sean PICs, displays, convertidores, así como materiales adicionales: resistencia, capacitares, inductores, entre otros.

La principal ventaja de esta herramienta es que se pueden simular proyectos muy complejos que primero requieren de un diseño experimental antes de poderlos crear físicamente, con esto se evitan los riesgos de perder un componente ocasionado por alguna conexiones errónea.

El Proteus es una herramienta de trabajo muy amigable, ya que muestra los componentes más comunes que se pueden agregar en un determinado momento al proyecto. En la Figura 20 se puede apreciar el ambiente de trabajo que brinda el Proteus.

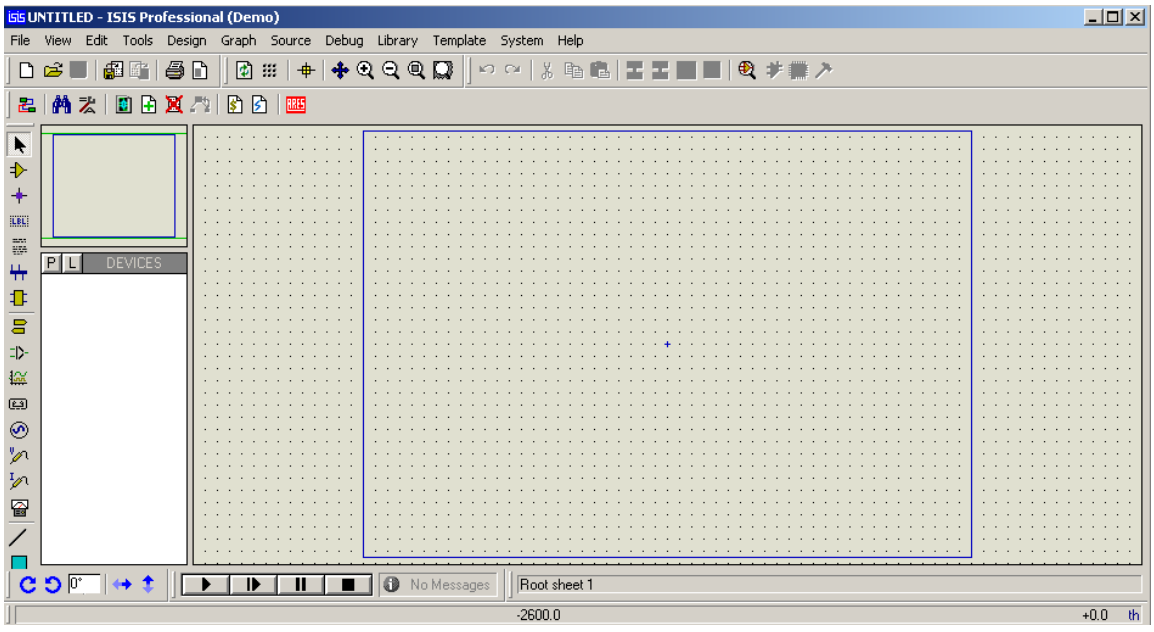

Figura 20: Ambiente de trabajo de la herramienta Proteus.

Ahora, para poder realizar la simulación, como en el MPLAB, se necesita seleccionar el dispositivo, esto se realiza como se muestra en la Figura 21.

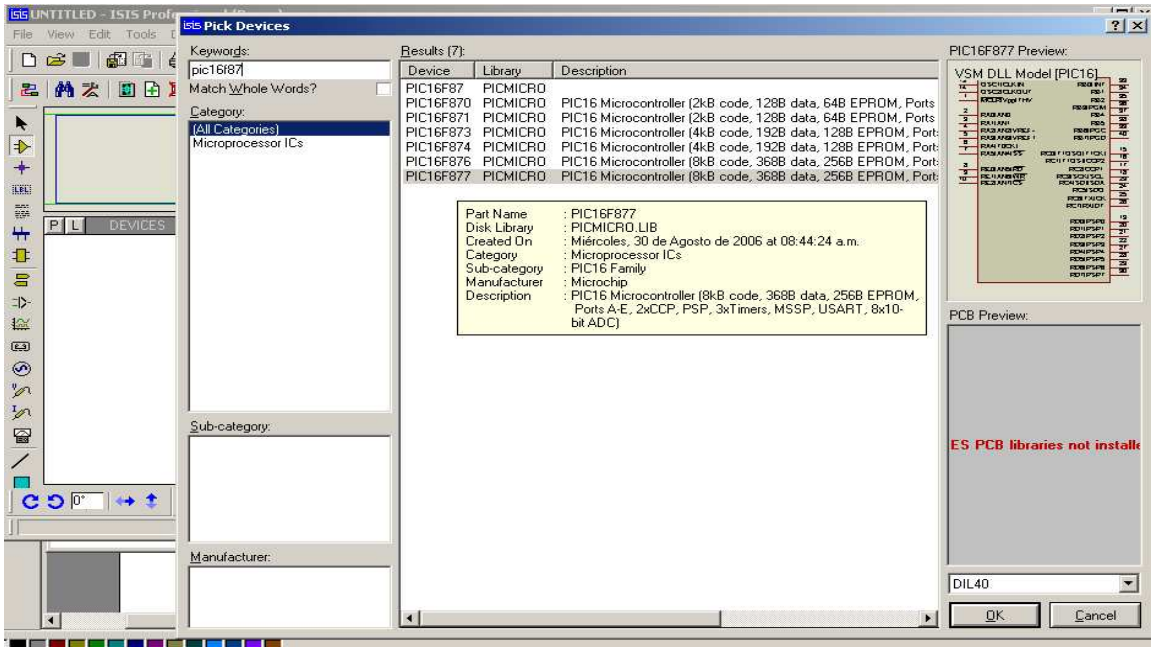

Figura 21: Selección del PIC a simular con el Proteus

Con esto se logra agregar el dispositivo al proyecto principal.

Una vez que se seleccionó el dispositivo, se procede a cargar el programa al PIC seleccionado, esto se realiza dando doble clic al dispositivo, es en ese momento cuando se muestra otra nueva ventana (Figura 22), en la cual se le debe de indicar el programa a ejecutar.

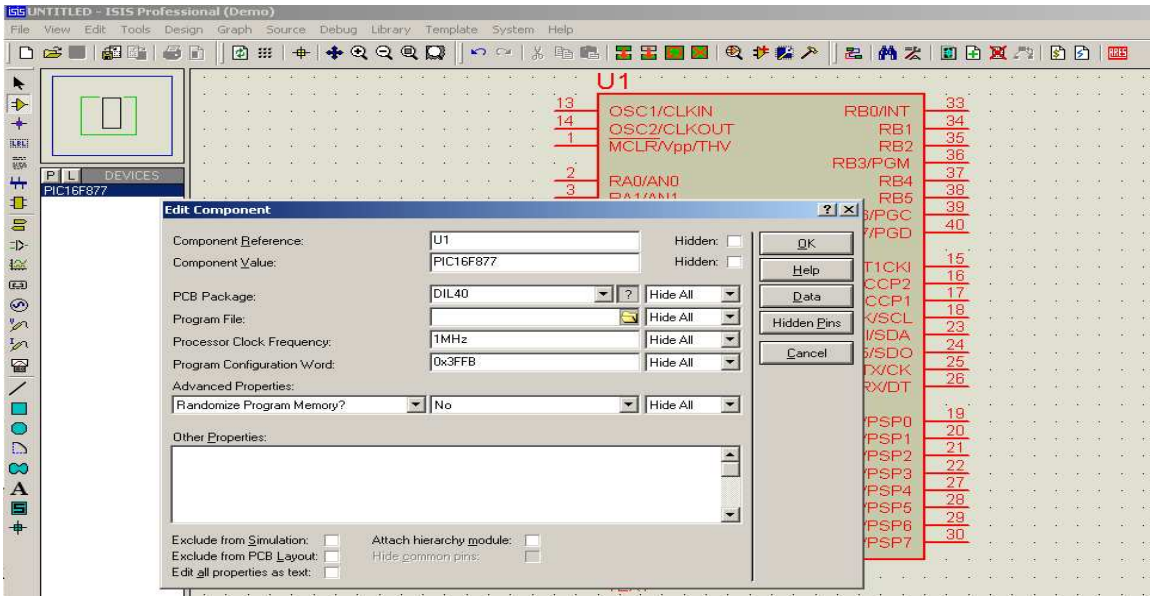

Figura 22: Selección del programa a cargar en el Dispositivo

Se selecciona en programa, que en este caso debe de ser con extensión “*.hex” o “*.cof”.

Es preciso aclarar que para probar el programa creado en MPLAB, no basta con descargar el programa al PIC en la herramienta Proteus. Sino que debe incluir partes físicas como resistencias, LEDs y un elemento muy importante que es el oscilador o cristal de cuarzo, el cual es encargado de dar la frecuencia de reloj con la que va a trabajar el PIC.

Se implementó un diseño mostrado en la Figura 23, donde se agregan los elementos necesarios para la prueba y funcionamiento del Programa

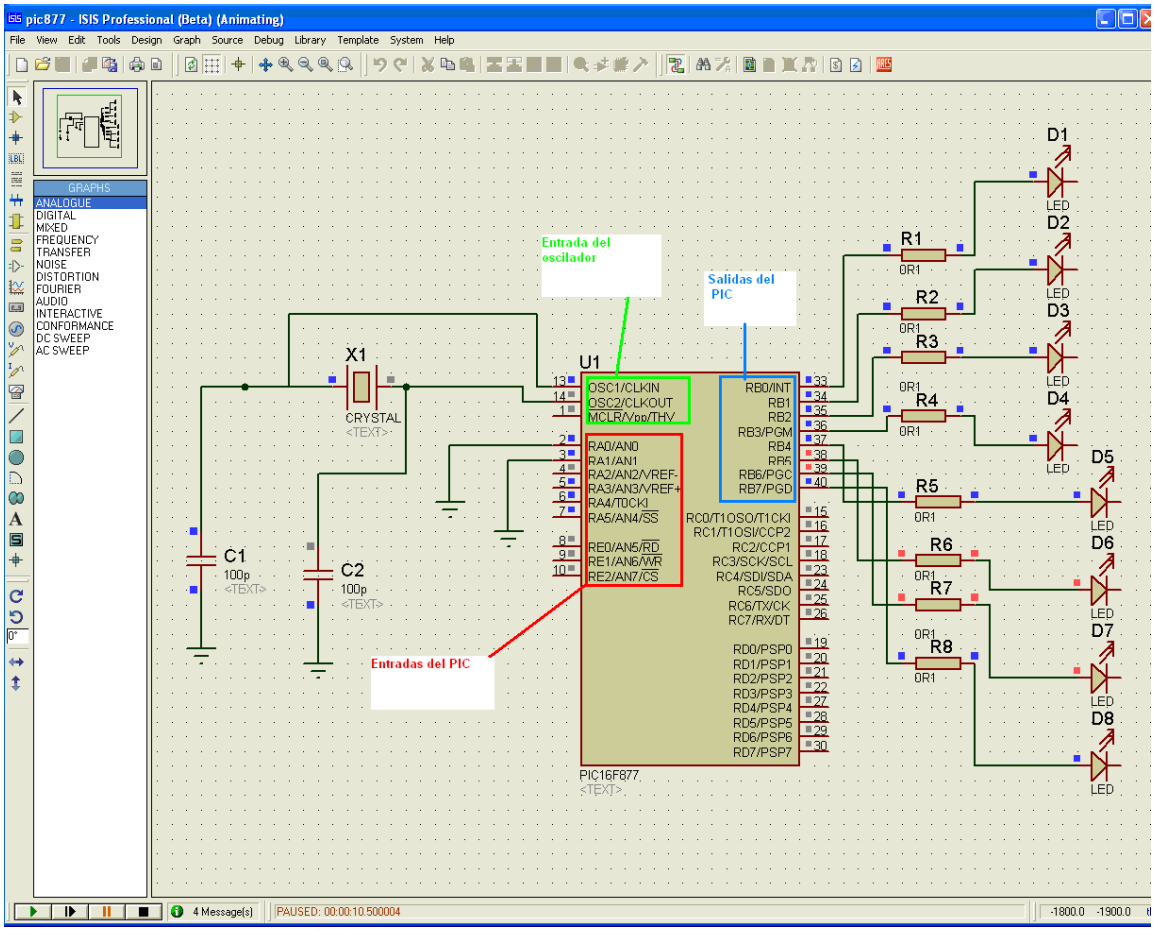

Figura 23: Diseño de Tarjeta en Proteus con resistencias, LEDs, capacitares y un oscilador.

Dentro de este diseño se han agregado resistencia de un valor de 4.3 Homs a las salidas del puerto B (recuadro azul de la Figura 23). Después de las resistencias se han agregado los LEDs los cuales se van a prender o apagar dependiendo del valor recibido.

Por otro lado, las entradas que pertenecen al puerto A (recuadro rojo de la Figura 23), están alimentadas con tierra y corriente, este cambio se realiza dependiendo del valor que se quiera ingresar al puerto, si se quiere ingresar un uno a un pin del puerto A, entonces ese pin se alimenta con 5 volts, en caso contrario se conecta a tierra.

También se ha conectado al PIC un oscilador de 4MHZ, el cual debe ser conectado en los pines *OSCI/CLKIN* Y *OSCI/CLKOUT* como se indica en la Figura 23 (recuadro verde de la Figura 23). El oscilador se encuentra conectado a dos capacitares de un valor de 100 pico faradios.

Las conexiones que se realizaron para este diseño se han tomado del manual de referencia de medio rango para el PIC16F877A [PIC 2009].

Este diseño fue probado y funcionó de manera correcta.

4.2 Pruebas de hardware

Dentro de las pruebas realizadas en el proyecto se encuentra el diseño físico de la tarjeta de control del PIC, el cual esta montado sobre un *Protoboard* y contiene las especificaciones del diseño creado en Proteus descrito en la sección 4.2.2 de este documento.

En la Figura 24 se muestra el Diseño físico implementado.

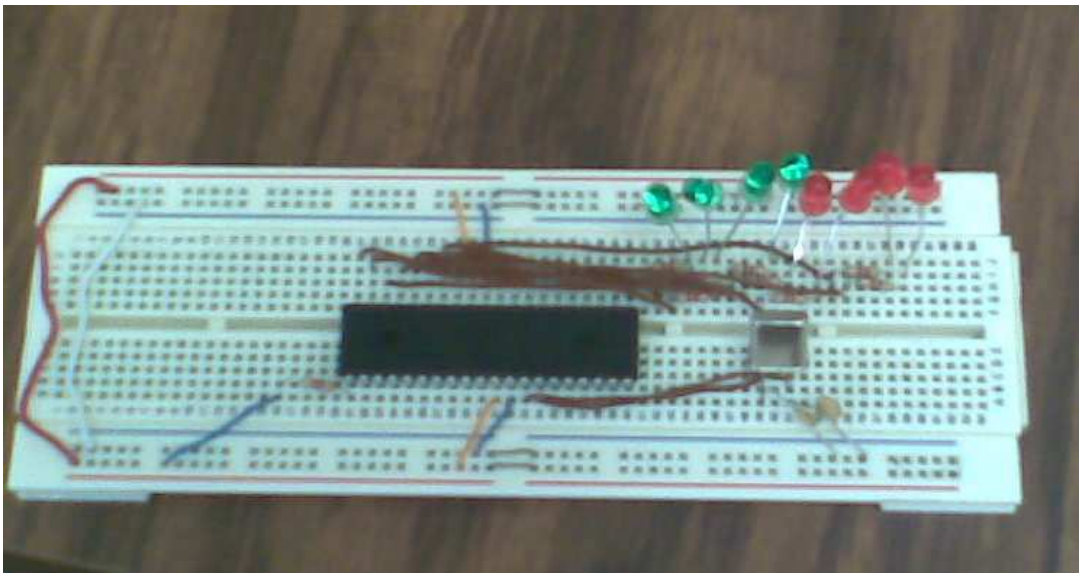

Figura 24: Diseño físico de la tarjeta de control del PIC

Se descargó el programa al PIC y se realizaron las pruebas pertinentes, sin embargo el PIC no funcionó correctamente.

En ese momento se contempló la posibilidad de que el PIC estuviera dañado. Para argumentar esto, se consiguió prestado un PIC del mismo modelo, se probó el programa y funcionó correctamente.

Actualmente se cuenta con un PIC modelo 16f877, para dar solución a la limitante que se encontró con el PIC anterior (PIC 16f877A), la propuesta es tratar de migrar el código que ya se tiene al nuevo PIC.

Cabe mencionar que la migración del código realizado para el PIC 16f877A al PIC 16f877 no debe tener mayor complicación, ya que comparten un conjunto de instrucciones similares.

CAPITULO 5. Resultados y Conclusiones

5.1 Resultados obtenidos en este proyecto

Se llegó a comprender el funcionamiento del PIC, con esto se a logrado crear un programa en lenguaje ensamblador que puede ser ejecutado dentro del PIC16f877A., tal programa fue escrito y probado con la herramienta MPLAB IDE y la herramienta de diseño Proteus y funciona correctamente.

Se implementó una rutina de retardo de tiempo a través de un módulo propio del PIC el cuál sirve para generar un retardo de tiempo entre el envío para secuencia de bits hacia tarjeta de control de motores.

Se tiene un diseño en la herramienta Proteus donde se pueden visualizar de manera grafica las salidas del PIC, así como los elementos necesarios para el diseño físico.

Se cuenta con el diseño físico de la tarjeta de control para el PIC montado en un *Protoboard.*

Los comportamientos que se programaron en el PIC son los descritos en el algoritmo de la sección 4.1.2 de este documento, también están basados en comportamientos básicos que se implementaron en C++, durante una fase anterior a este proyecto.

Se cuenta con la documentación detallada del funcionamiento del sensor de proximidad SRF05, la cual describe la forma en la cual se debe conectarse, así como la manera en la que se recuperaran las señales desde el mismo. De esta manera se logra calcular la distancia a la que se detecta un objeto.

Con la implementación de programas almacenados en un PIC se alcanzará una independencia mayor del vehiculo, ya que no dependerá de la conexión de un cable a una CPU que envíe las secuencias de comportamientos a ejecutar, sino que se realizaran allí mismo, en el vehículo.

5.2 Conclusiones del proyecto

Este proyecto es solo una fase dentro de un proyecto más ambicioso, que busca dotar con cierto nivel de inteligencia a un vehiculo para realizar tareas de exploración, reconstrucción de ambientes y planificación de trayectorias. Aunque aun se esta muy lejos de lograr ese objetivo general, con este proyecto de investigación se aporta el diseño para controlar la tarjeta de motores a través de un PIC utilizando un evento que se genera desde un sensor ultrasónico. Así, este proyecto contribuye en cierta medida con la construcción del robot, ya que a partir de aquí pueden desarrollarse otras etapas del mismo.

La realización de pruebas del sensor de proximidad ultrasónico no se pudo realizar completamente debido a la complejidad de tal prueba, ya que es necesario tener conocimientos más especializados sobre herramientas electrónicas, en este caso, el manejo a detalle de un osciloscopio.

Físicamente, la tarjeta de control donde es montado el PIC funciona correctamente, el único dispositivo que se encontró con fallos fue el propio PIC.

El desarrollo de este proyecto permitió adquirir un conjunto de conocimientos más firmes y fundamentados sobre el diseño de circuitos necesario para este tipo de aplicaciones, además, dio la oportunidad de implementar programas en un lenguaje de bajo nivel y cooperar en el desarrollo de un proyecto de mayor escala.

Referencias

[Alí 2009] Alí Martínez "Construcción de un Vehículo todo Terreno con Control de Señales para Movimientos Básicos"

[Aníbal O. 2001] Aníbal O. 2001 "Robótica*: Manipuladores y robots móviles"*, illustrated.

[Ángel S. 2005] Ángel S. 2005 Control Visual de Robots Paralelos: Análisis, Desarrollo y Aplicación a la Plataforma RoboTenis. Tesis Doctoral. Departamento/Escuela: Automática, Ingeniería Electrónica e Informática Industrial / E.T.S.I. Industriales (UPM)

[Braitemberg 1984] Braitemberg 1984, "*Vehicles: Experiments in Synthetic Psychology"*, Cambridge, Massachusetts

[MPLAB 2008] MPLAB Integrated Development Environment. Part Number: SW007002. ©2008 Microchip Technology Inc. Fecha de acceso: 25-febrero-2009. Disponible en:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

[PIC 2009] PIC 16f877A Documentación. ©2008 Microchip Technology Inc. Fecha de acceso: 20-abril-2009. Disponible en:

http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010242

[Robot 2009] Robot Electronics Fecha de acceso: 8-junio-2009. Disponible en http://www.robot-electronics.co.uk/htm/srf05tech.htm

[Robótica 2009] Robótica Fácil © Copyright 2002 - 2009 INTPLUS ® Fecha de acceso: 6-febrero-2009. Disponible en: http://www.SuperRobotica.com

[Rodney 1991] Rodney A. Brooks. "*Intelligence without representation"*. Artificial Intelligence 47, Elsvier Science Publisher, pp.139-159.

Lista de apéndices

Apéndice A

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include "io.h"
#include <dos.h>
using namespace std;
//---------------------------------------
void Horario(int port, int value)
{  PortWordOut(port, value);    }
//=======================================
int leido;
int menu(int);
int main(int argc, char *argv[]){
   int j;
   int value=1;
   byte lastvalue=0;
   unsigned port = 0x00000378;
   printf("previous value : %d",value);
   int l;   // inicializa controlador de puerto
   l = LoadIODLL();
   int dr = -10;
   dr = IsDriverInstalled(); // regresa no cero si pudo inicializar corrr.
   printf("Driver ? %d\n", dr);
   int opc,opclast,num=0;
   int valc[5], valcw[5];
   float grd, mts;

   valc[0]=3;  valc[1]=6;  valc[2]=12; valc[3]=9; valc[4]=3;
   valcw[0]=9; valcw[1]=12;valcw[2]=6; valcw[3]=3;valcw[4]=9;
   do  {
      opc=menu (num);
      switch(opc) {
                  case 1:
                  int   m1;
                  m1=1;
                    // motor superior torque a 2 bobinas giro a la izkierda
                  lastvalue=lastvalue & 240;
                  for (int i1=1; i1<=m1; i1++)                    {
                     value=0x3 | lastvalue;
                     Horario(port,value);
                     Sleep(500);
                     value=0x6 | lastvalue;
```

```
                    Horario(port,value);
                    Sleep(500);
                    value=0xC | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0x9 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                }
                lastvalue=value;
                break;
//PRIMEROS 4 BITS LADO DERECHO //AVANZAR A UN PASO
DERECHA=============================================================
============
          case 2:
                int  m2;
                m2=1;
                lastvalue=lastvalue & 240;
                for (int i2=1; i2<=m2; i2++){
                    value=0x9 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0xC | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0x6 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0x3 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                }
                 lastvalue=value;
                     break;
          case 3:
                int   m3;
                m3=1;
                  // MI (4 pasos 1), Der  4 pasos, torqu3 3 , 3TIEMPOS
                lastvalue=lastvalue & 240;
                value=0x0;
                Horario(port,value);
                Sleep(100);
                for (int i3=1; i3<=m3; i3++) {
                    value=0x3 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0x6 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                    value=0xC | lastvalue;
                    Horario(port,value);
```

```
                    Sleep(500);
                    value=0x9 | lastvalue;
                    Horario(port,value);
                    Sleep(500);
                }
                lastvalue=value;
                break;

//PRIMEROS 4 BITS LADO DERECHO //AVANZAR A UN PASO
DERECHA=============================================================
============
          case 4:
               int  m4;
               m4=1;
               lastvalue=lastvalue & 240;
               value=0x0;
               Horario(port,value);
               Sleep(100);
               for (int i4=1; i4<=m4; i4++){
                  value=0x9 | lastvalue;
                  Horario(port,value);
                  Sleep(500);
                  value=0xC | lastvalue;
                  Horario(port,value);
                  Sleep(500);
                  value=0x6 | lastvalue;
                  Horario(port,value);
                  Sleep(500);
                  value=0x3 | lastvalue;
                  Horario(port,value);
                  Sleep(500);
                }
                lastvalue=value;
                   break;
                         case 5:  //AVANZAR A UN PASO
TODO=================================================================
========
               int t5, g5, v5, s5, m5;
               lastvalue=lastvalue & 15;
               for (int i5=0; i5<=4.8; i5++)  {
                  value=0x30 | lastvalue;
                  Horario(port,value);
                  Sleep(100);
                  value=0x60 | lastvalue;
                  Horario(port,value);
                  Sleep(100);
                  value=0xC0 | lastvalue;
                  Horario(port,value);
                  Sleep(100);
                  value=0x90 | lastvalue;
                  Horario(port,value);
```

```
            Sleep(100);
          }
          lastvalue=value;
          break;
case 6: //RETROCEDER A UN PASO TODO====
        int t6, g6, v6, s6, m6;
        lastvalue=lastvalue & 15;
        for (int i6=0; i6<=4.8; i6++) {
          value=0x90 | lastvalue;
          Horario(port,value);
          Sleep(100);

          value=0xC0 | lastvalue;
          Horario(port,value);
          Sleep(100);

          value=0x60 | lastvalue;
          Horario(port,value);
          Sleep(100);

          value=0x30 | lastvalue;
          Horario(port,value);
          Sleep(100);
         }
         lastvalue=value;
             break;
 case 7:
        int   m7;
        m7=1;
          // motor superior torque a 2 bobinas giro a la izkierda
        for (int i7=1; i7<=m7; i7++) {
          value=0x0;
          Horario(port,value);
          Sleep(500);
          value=0x0;

          Horario(port,value);
          Sleep(500);
          value=0x0;

          Horario(port,value);
          Sleep(500);
          value=0x0;

          Horario(port,value);
          Sleep(500);
        }
        lastvalue=value;
        break;
case 8:
         break;
```

```
        }//fin del switch
        opclast=opc;
       if (opc==9){
              printf ("\nUsted a salido del programa\n");
                }
   }while (opc!=9);
   printf("Value %d sent to port number %h\n", value, port);
   system("PAUSE");
   return EXIT_SUCCESS;
}
int menu (int num)
{
   printf("Opciones para restablecer la direccion\n");
      printf("1.Restablece de derecha al centro\n");
      printf("2.Restablece de izquierda al centro\n\n");
      printf("Seleccione el movimiento que de desea hacer\n");
      printf("3.Girar a la izqhierda\n");
      printf("4.Girar a la derecha\n");
      printf("5.Avanzar\n");
      printf("6.Retrodecer\n");
      printf("7.liberara motores\n");
   printf("8.Ayuda\n");
      printf("9.Salir\n");
   leido=getch();
   num = int (leido);
   num=num-48;
   return (num);
}
```

Programa en lenguaje C++ con algunos comportamientos implementados

Apéndice B

```
        list            p=16f877a; list directive to define processor
        #include <p16f877a.inc>     ; processor specific variable definitions

        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _RC_OSC & _WRT_OFF & _LVP_ON &
_CPD_OFF
; '__CONFIG' directive is used to embed configuration data within .asm file.
; The labels following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.
;***** VARIABLE DEFINITIONS (examples)

; example of using Shared Uninitialized Data Section
INT_VAR         UDATA_SHR       0x71
w_temp          RES     1       ; variable used for context saving
status_temp     RES     1       ; variable used for context saving
pclath_temp     RES             1               ; variable used for context saving

; example of using Uninitialized Data Section
TEMP_VAR        UDATA   0x20            ; explicit address specified is not required
temp_count      RES     1               ; temporary variable (example)

; example of using Overlayed Uninitialized Data Section
; in this example both variables are assigned the same GPR location by linker
G_DATA          UDATA_OVR               ; explicit address can be specified
flag            RES     2               ; temporary variable (shared locations - G_DATA)

G_DATA          UDATA_OVR
count           RES     2               ; temporary variable (shared locations - G_DATA)


; CODIGO DE AMADOR.....................................
G_DATA          UDATA_OVR
CONTA1          RES     2               ;

G_DATA          UDATA_OVR
CONTA2          RES     2               ;

G_DATA          UDATA_OVR
AUX_PORTA       RES     2

;*********************************************************************
RESET_VECTOR    CODE    0x000           ; processor reset vector
        nop                                             ; nop required for icd
        movlw   high  start     ; load upper byte of 'start' label
        movwf   PCLATH                  ; initialize PCLATH
        goto    start                   ; go to beginning of program

INT_VECTOR      CODE    0x008           ; interrupt vector location
        movwf   w_temp                  ; save off current W register contents
        movf    STATUS,w                ; move status register into W register
        movwf   status_temp             ; save off contents of STATUS register
        movf    PCLATH,w                ; move pclath register into w register
        movwf   pclath_temp             ; save off contents of PCLATH register

; isr code can go here or be located as a call subroutine elsewhere

        movf    pclath_temp,w           ; retrieve copy of PCLATH register
        movwf   PCLATH                  ; restore pre-isr PCLATH register contents
        movf    status_temp,w           ; retrieve copy of STATUS register
        movwf   STATUS                  ; restore pre-isr STATUS register contents
        swapf   w_temp,f
        swapf   w_temp,w        ; restore pre-isr W register contents
        retfie                                  ; return from interrupt
;-----------------------------------------------------------------------------------------
MAIN    CODE
start
```

```
        CONF_ADCON1 equ b'00000110' ; Configuracion PORTA E/S digital
        nop                                     ; No operacion
        nop                                     ; No operacion
        bsf STATUS,RP0
        bcf STATUS,RP1
        movlw CONF_ADCON1               ; Configurar el PORTA como digital
        movwf ADCON1
        movlw b'00000111'       ; PORTA como entrada
        movwf TRISA
        clrf TRISB                              ; PORTB como salida
        bcf STATUS,RP0                  ; Ir banco 0
        bcf STATUS,RP1
;-------------------------------------------------------------------------------------
Principal                                       ; Programa principal que ejecuta todos los comportamientos
                                                        ;//////////////////////////////////////////////////////////////
                                                        ; // el CicloA permite que el robot avance hasta encontrar
                                                        ;       algun obstaculo, esto mientras el valor leido de la
entrafa A sea 0
                                                        ;//////////////////////////////////////////////////////////////

        ;=====================================================================
CicloA
          movfw PORTA                   ;almacena el valor de PORTA en el registro WREG
          movwf AUX_PORTA               ;almacena el valor de PORTA en una variable auxiliar
    movlw 0x00
          subwf AUX_PORTA,0     ; Cero en el puerto de entrada implica avanzar
          btfsc STATUS,Z        ; Si Z=1 Va a Avanzar
                call Avanzar
          btfsc STATUS,Z        ; Si Z=0 Va siguientes Rutinas
        goto CicloA
                                                        ;.....

                                                        ;//////////////////////////////////////////////////////////////
                                                        ; // el CicloB reacciona al valor de entrada 1, en caso de que
se
                                                        ; dertecte un objeto del lado izquierdo
                                                        ;//////////////////////////////////////////////////////////////

        ;=====================================================================
        movlw 0x01                              ; Comparar con 1
        subwf AUX_PORTA,0               ; ----------(Aux_portA - Wreg)
        btfsc STATUS,Z                  ; Si Z=1 Va a Restablece_derecha
                call Curso_Derecha

        movlw 0x02                              ; Comparar con 1
        subwf AUX_PORTA,0               ; ----------(Aux_portA - Wreg)
        btfsc STATUS,Z                  ; Si Z=1 Va a Restablece_derecha
                call Curso_Izquierda
        goto Principal

;*********************************************************************
;*****************************     subrutinas     ******************
;*********************************************************************
Curso_Derecha

        movlw 0x04              ;Se llama cinco veces a la rutina gira_der
        movwf INTCON
itera1
        decfsz INTCON
        call Retroceder
        movlw 0x01                      ; Comparar con 1
        subwf INTCON,0
        btfss STATUS,Z
        goto itera1

        call Derecha
        return

;.....................................................
Curso_Izquierda
```

```
        movlw 0x04              ;Se llama cinco veces a la rutina gira_der
        movwf INTCON
itera3
        decfsz INTCON
        call Retroceder
        movlw 0x01                      ; Comparar con 1
        subwf INTCON,0
        btfss STATUS,Z
        goto itera3

        call Izquierda
        return

;.....................................................................
Avanzar
        movlw 0x03              ;mueve 0x03 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0x60              ;mueve 0x06 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0xC0              ;mueve 0x0C al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0x90              ;mueve 0x09 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;manda a dormir un momento
        return                  ;regresa a la etiqueta Principal

;.....................................................................
Retroceder
        movlw 0x90              ;mueve 0x03 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0xC0              ;mueve 0x06 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0x60              ;mueve 0x0C al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;duerme un momento
        movlw 0x03              ;mueve 0x09 al registro W
        movwf PORTB             ; pone el valor de W para enviarlo por el PORTB
        call Delay          ;manda a dormir un momento
        return                  ;regresa a la etiqueta Principal

;.....................................................................
;  gira 50 veces
Derecha
        movlw 0x05              ;Se llama cinco veces a la rutina gira_der
        movwf INTCON
itera
        decfsz INTCON
        call gira_der
        movlw 0x01                      ; Comparar con 1
        subwf INTCON,0
        btfss STATUS,Z
        goto itera
   RETURN

;.....................................................
Izquierda
        movlw 0x05
        movwf INTCON
iterai
        decfsz INTCON
        call gira_izq
        movlw 0x01                      ; Comparar con 1
        subwf INTCON,0
        btfss STATUS,Z
```

```
            goto iterai
   RETURN

;.....................................................................
;----Gira_derecha; Comportamiento de giro
gira_der
            movlw 0x09                  ;mueve 0x03 al registro W
            movwf PORTB                 ; pone el valor de W para enviarlo por el PORTB
            call Delay              ;manda a dormir un momento
            nop
            movlw 0x0C
            movwf PORTB                 ; pone el valor de W para enviarlo por el PORTB
            call Delay              ;manda a dormir un momento
            movlw 0x06
            movwf PORTB                 ; pone el valor de W para enviarlo por el PORTB
            call Delay              ;manda a dormir un momento
            movlw 0x03
            movwf PORTB                 ; pone el valor de W para enviarlo por el PORTB
            call Delay              ;manda a dormir un momentoç
            return
;.....................................................................
;----Gira_izquierda; Comportamiento de giro
gira_izq
            movlw 0x03 ;mueve 0x03 al registro W
            movwf PORTB; pone el valor de W para enviarlo por el PORTB
            call Delay ;duerme un momento
            movlw 0x06 ;mueve 0x06 al registro W
            movwf PORTB; pone el valor de W para enviarlo por el PORTB
            call Delay ;duerme un momento
            movlw 0x0C ;mueve 0x0C al registro W
            movwf PORTB; pone el valor de W para enviarlo por el PORTB
            call Delay ;duerme un momento
            movlw 0x09 ;mueve 0x09 al registro W
            movwf PORTB; pone el valor de W para enviarlo por el PORTB
            call Delay ;manda a dormir un momento
            return
;.....................................................................
;.....  subrutine delay .....................
Delay
                        movlw    0xF0
                        movwf    CCP2CON   ; set outer delay loop
DelayOuter
                        movlw    0xFF
                        movwf    CCP1CON     ; set inner delay loop
DelayInner
                        decfsz   CCP1CON
                        goto     DelayInner
                        decfsz   CCP2CON
                        goto     DelayOuter
                        return
;.....................................................................

Restablece_derecha              ; Comportamiento de giro a la izquierda, motor superior torque a 2 bobinas
            movlw 0x03                          ;mueve 0x03 al registro W
            movwf PORTB                         ; pone el valor de W para enviarlo por el PORTB
            call Delay                      ;duerme un momento
            movlw 0x06                          ;mueve 0x06 al registro W
            movwf PORTB                         ; pone el valor de W para enviarlo por el PORTB
            call Delay                      ;duerme un momento
            movlw 0x0C                          ;mueve 0x0C al registro W
            movwf PORTB                         ; pone el valor de W para enviarlo por el PORTB
            call Delay                      ;duerme un momento
            movlw 0x09                          ;mueve 0x09 al registro W
            movwf PORTB                         ; pone el valor de W para enviarlo por el PORTB
            call Delay                      ;manda a dormir un momento
;           return                              ;regresa a la etiqueta Principal

;-------------------------------------------------------------------------------------------
Restablece_izquierda  ; avanza un paso a la derecha
            movlw 0x09 ;mueve 0x03 al registro W
```

```
        movwf PORTB; pone el valor de W para enviarlo por el PORTB
        call Delay ;manda a dormir un momento
        movwf 0x0C
        movwf PORTB; pone el valor de W para enviarlo por el PORTB
        call Delay ;manda a dormir un momento
        movwf 0x06
        movwf PORTB; pone el valor de W para enviarlo por el PORTB
        call Delay ;manda a dormir un momento
        movwf 0x03
        movwf PORTB; pone el valor de W para enviarlo por el PORTB
        call Delay ;manda a dormir un momento
        goto Principal;regresa a la etiqueta Princupal

;-------------------------------------------------------------------------------------------
LiberarMotores;pone valores a cero y los manda por el puerto B
        movlw 0x00
        movwf PORTB
        call Delay
        movwf 0x00
        movwf PORTB
        call Delay
        movwf 0x00
        movwf PORTB
        call Delay
        movwf 0x00
        movwf PORTB
        call Delay
        goto Principal;regresa a la etiqueta Princupal
;-------------------------------------------------------------------------------------------

        END                     ; directive 'end of program'
```