Resumen

Este proyecto consiste en diseñar el sistema electrónico para el control de un vehículo de tres ruedas mediante microcontroladores tipo PIC; el control sobre dicho sistema será ejercido por el usuario mediante señales *bluetooth* enviadas por un dispositivo *Android.* El vehículo además contará con una pantalla LCD por la que presentará información sobre la dirección y velocidad de movimiento del sistema. Para la alimentación del vehículo se usarán 6 pilas AA de 1.6 V destinadas a los motores y una pila de 9V con un regulador de tensión a 5V para la alimentación del resto del sistema.

En esta memoria que describe el proyecto realizado, se comentarán uno a uno todos los componentes electrónicos usados, dando una descripción detalla de su propósito y de su funcionamiento, seguidamente se indicará el software que se ha utilizado y sus características. A continuación se narrará paso por paso la programación y configuración que se lleva a cabo en los microcontroladores para cada uno de los subsistemas diseñados. Para el control desde el dispositivo *Android* se diseñará una aplicación móvil de la cual se explicará la creación de su interfaz gráfica y del diseño del programa de esta.

Se dará una explicación global del funcionamiento del sistema de control, además de las interacciones entre subsistemas y cómo afectan unos a otros. Los problemas que han surgido por la implantación de un nuevo subsistema no tenido en cuenta en un principio.

Todo se escribirá de manera que esta memoria pueda servir en un futuro como una posible referencia para aquellos estudiantes que deseen inicializarse en el mundo de la electrónica.

Satisfactoriamente se cumplieron los objetivos planteados de diseñar y construir el sistema electrónico para el control de un vehículo de tres ruedas controlado por un dispositivo *Android* mediante *bluetooth*. Además se llegó a la conclusión de que cuando se diseña un sistema a medida para cumplir unas especificaciones concretas y se le integran nuevos subsistemas no considerados en un inicio, la interacción entre estos es más compleja y requiere por lo tanto de nuevas soluciones.

Sumario

RESUMEN 1

SUMARIO 3

1. GLOSARIO 7

2. LISTA DE FIGURAS 9

3. LISTA DE TABLAS 11

4. PREFACIO 13

4.1. Origen del proyecto 13

4.2. Requerimientos previos 13

5. INTRODUCCIÓN 14

5.1. Objetivos del proyecto 14

5.2. Alcance del proyecto 14

6. HARDWARE 15

6.1. Microcontroladores PIC de Microchip 15

6.1.1. PIC12F509 15

6.1.2. PIC16F690 16

6.2. PICKit 2 17

6.3. Placa *protoboard* 18

6.4. Otros componentes electrónicos 18

6.4.1. Cables 18

6.4.2. Resistores 18

6.4.3. Resistores variables 19

6.4.4. Condensadores 19

6.4.5. Interruptores 19

6.4.6. Regulador de Tensión 19

6.5. Pantalla LCD 20

6.6. Módulo *bluetooth* HC-05 21

6.7. Motor BDC 21

6.8. Controlador: L293D 23

6.9. Sensor de distancia por ultrasonidos 24

7. SOFTWARE PARA EL MICROCONTROLADOR 26

7.1. MPLABX 26
7.2. HI-TECH C 27

8. PROGRAMACIÓN DEL MICROCONTROLADOR ____ 28
8.1. Programación LCD 28
8.2. Programación del motor 30
8.3. Programación módulo ADC 31
8.4. Programación módulo EUSART 33
8.5. Programación del sensor de ultrasonidos 36

9. PROGRAMACIÓN DE LA APLICACIÓN MÓVIL ____ 37
9.1. MIT App Inventor 37
9.2. Interfaz gráfica 37
9.3. Programación en Bloques 39

10. SISTEMA DE CONTROL DEL COCHE ELÉCTRICO ____ 42
10.1. Funcionamiento general 42
10.2. Recepción de señales mediante módulo bluetooth 44
10.3. Control de motores 45
10.4. Pantalla LCD 47
10.5. Lectura del estado de baterías 49
10.6. Detección de obstáculos 51
10.7. Alimentación del coche eléctrico 52

11. ESTRUCTURA DEL COCHE ____ 54
11.1. Composición física del coche 54
11.2. Problema de desplazamiento en trayectorias rectas 54

12. PRESUPUESTO ECONÓMICO ____ 57

13. IMPACTO MEDIOAMBIENTAL ____ 60

CONCLUSIONES ____ 61

AGRADECIMIENTOS ____ 62

BIBLIOGRAFÍA ____ 63
Referencias bibliográficas 63
Bibliografía de las figuras 65

ANEXOS (VOLUMEN SEPARADO) ____ 1

I. Tabla de comandos de control 1
II. Tabla de caracteres estándares LCD 2
III. Ejemplo código programación LCD 3
IV. Ejemplo código carácter personalizado en LCD 4
V. Ejemplo código señal PWM para control de motores 5
VI. Ejemplo inicialización de módulo ADC 6
VII. Ejemplo inicialización de módulo EUSART 6
VIII. Ejemplo de código detección de obstáculos. 7
IX. Código utilizado en el microcontrolador derecho 8
X. Código utilizado en el microcontrolador izquierdo 23

1. Glosario

I/O*: Input*/*Output*, de entrada y salida.

LED: *Light-emitting diode,* diodo emisor de luz.

Reset: Reiniciar.

Pinout: Asignación de patillaje o disposición de los pines.

Placa protoard: Placa de pruebas.

Motor BDC: *Brushed direct current, m*otor de corriente continua con escobillas.

PWM: *Pulse-width modulation,* modulación por ancho de pulsos.

LCD: *Liquid crystal display,* pantalla de cristal líquido.

A/D: Conversión analógica-digital.

ADC: *Analog-to-digital converter*, convertidor analógica-digital

EUSART: *Enhanced universal synchronous asynchronous receiver transmitter,* transmisor-receptor asíncrono universal mejorado.

RSR: *Receive shift register*, registro de desplazamiento receptor**.**

FIFO: *First in, first out,* primero en entrar, primero en salir.

Overflow: Inundación. Se produce cuando un temporizador rebasa su valor máximo, es decir 255, y por lo tanto empezará a contar desde 0 nuevamente.

Bluetooth: Es una especificación industrial para Redes Inalámbricas de Área Personal que posibilita la transmisión de voz y datos entre diferentes dispositivos

True: Verdadero.

False: Falso.

2. Lista de Figuras

Figura 1. Microcontrolador Microchip. 15

Figura 2. Pinout de PIC12F509. 15

Figura 3. Pinout de PIC16F690. 16

Figura 4. Programador PICkit 2. 17

Figura 5. Pinout del PICkit 2. 17

Figura 6. Placa protoboard. 18

Figura 7. Resistencia. 18

Figura 8. Resistencia variable. 19

Figura 9. Condensador. 19

Figura 10. Regulador de tensión. 19

Figura 11. Pinout de pantalla LCD. 20

Figura 12. Pantalla LCD. 20

Figura 13. Placa bluetooth. 21

Figura 14. Componentes básicos en un motor BDC. 21

Figura 15. Motores utilizados en el coche eléctrico. 22

Figura 16.Placa con controlador L293D. 23

Figura 17. Circuito de L293D en la placa. 23

Figura 18. Sensor de ultrasonidos. 24

Figura 19. Señales enviadas y recibidas por el sensor de ultrasonidos. 25

Figura 20. Interfaz de MPLABX. 26

Figura 21. Circuito entre pantalla LDC y PIC16F690. 28

Figura 22. Ejemplo de carácter personalizado. 30

Figura 23. Señal modulada en función del Timer2. 30

Figura 24. Diagrama de bloques de ADC 32

Figura 25. Resultado de conversión 10 bit A/D. 33

Figura 26. Diagrama de bloques de recepción EUSART 34

Figura 27. Interfaz gráfica de la aplicación móvil. 38

Figura 28. Bloque de programación cliente bluetooth. 39

Figura 29. Bloques de programación de los elementos de dirección. 39

Figura 30. Bloques de programación de los elementos de velocidad y el sensor. 40

Figura 31. Bloques de programación de los deslizadores compensación y grado de giro. 41

Figura 32. Funcionamiento interno de los microcontroladores izquierdo y derecho. 42

Figura 33. Circuito del módulo bluetooth a los microcontroladores. 44

Figura 34. Conexión de los motores, placa L293D y microcontroladores. 47

Figura 35. Conexiones entre microcontroladores y pantalla LCD. 48

Figura 36. Divisor de tensión. 49

Figura 37. Descarga de una pila AA. 50

Figura 38. Circuito del divisor de tensión a los microcontroladores. 50

Figura 39. Circuito del sensor de ultrasonidos a los microcontroladores. 51

Figura 40. Alimentación utilizada en el vehículo. 52

Figura 41. Ruedas y bastidor utilizados. 54

Figura 42. Representación 2D simplificada del coche eléctrico. 54

Figura 43. Posición del CIR en un desplazamiento recto. 55

Figura 44. Posición del CIR en un giro. 55

3. Lista de Tablas

Tabla 1. Fórmulas de tasas de baudio. 35

Tabla 2. Variables de estado. 43

Tabla 3. Tensión en bornes de los motores para ejecutar una orden determinada. 45

Tabla 4. Ciclos de trabajo utilizados según orden y velocidad. 45

Tabla 5. Ciclos de trabajo utilizados según orden y velocidad. 46

Tabla 6. Elementos necesarios para la fabricación del coche eléctrico. 57

Tabla 7. Herramientas utilizadas para la programación y manufactura del coche eléctrico. 58

Tabla 8. Mano de obra Ingeniero Junior. 58

Tabla 9. Mano de obra Ingeniero. 58

Tabla 10. Coste total del proyecto. 59

4. Prefacio

4.1. Origen del proyecto

El origen de este proyecto surge de la inquietud personal del autor por profundizar sus conocimientos en una especialización de la ingeniería tan extensa como es la electrónica. El interés desarrollado primeramente en las clases de esta asignatura, que finalmente resultaron insuficientes para satisfacer la curiosidad del autor dieron lugar a una disposición por incrementar los conocimientos en esta rama de la física e ingeniería. En la búsqueda de un proyecto que sirviera para ampliar estos conocimientos, y gracias a la propuesta de la que luego sería la tutora de este proyecto, se decidió llevar a cabo un tema que tocase diferentes áreas.

4.2. Requerimientos previos

Para la realización de este proyecto los conocimientos previos necesarios, puesto que se tocan temas diversos, incluyen desde nociones de electrónica digital que se han visto a lo largo de la carrera, hasta conocimientos de programación en distintos ámbitos requeridos para entender C y programar en *App Inventor.*

En lo que respecta al material, al ser un proyecto con un componente práctico importante, se necesitó tener desde el principio a disposición diferentes componentes electrónicos básicos, ya sea para la manufactura del vehículo o para diferentes pruebas con fines experimentales y de aprendizaje.

Para el software que se ha de usar para la programación de los microcontroladores que incorpora el vehículo se utilizó software gratuito, como el MPLABX y el compilador HI-TECH C.

5. Introducción

5.1. Objetivos del proyecto

El objetivo principal del proyecto es diseñar el sistema electrónico para el control basado en microcontrolador de un vehículo sencillo de 3 ruedas. La comunicación entre usuario y vehículo se realizará mediante tecnología *bluetooth* desde un teléfono móvil Android.

Además se busca crear con la redacción de este documento una guía clara y concisa para todos aquellos estudiantes interesados en iniciarse en la electrónica.

5.2. Alcance del proyecto

Este proyecto, abarca diferentes ámbitos y ramas de la ingeniería y tiene un alcance sobre temas de amplio espectro; desde la programación y configuración de microcontroladores, el uso de una pantalla LCD, la detección de obstáculos mediante un sensor de ultrasonidos, el control de los motores y de todo el sistema a través de una aplicación móvil por señales *bluetooth*.

6. Hardware

En este capítulo se presentan los dispositivos y módulos básicos electrónicos utilizados para la realización del proyecto.

6.1. Microcontroladores PIC de Microchip

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria de programa. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida. **[1]**

Figura 1. Microcontrolador Microchip.

6.1.1. PIC12F509

El primer microcontrolador que se utilizó a modo de ensayo para la futura realización del proyecto fue el PIC12F509 debido a su simplicidad era el ideal para comenzar a familiarizarse con ellos. Son microcontroladores de bajo costo, y pequeños de tamaño físico, perfectos para aplicaciones donde el espacio es limitado y tienen tan solo ocho terminales. De sus características más importantes destacar que cuenta con, temporizador interno y un oscilador interno de 4MHz, entre otras cosas.

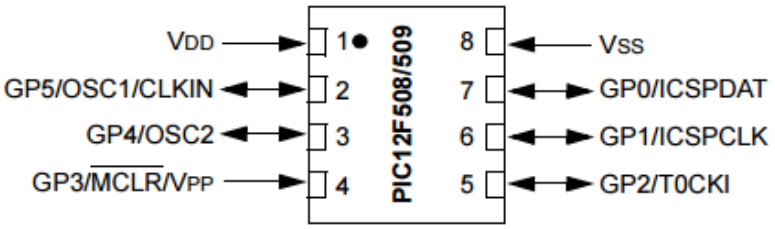

Figura 2. Pinout de PIC12F509.

Como se puede apreciar en la figura 2, los pin uno, ocho y cuatro solamente pueden funcionar como entrada o señales de alimentación, mientras que el resto de pines pueden ser programados tanto en entrada como en salida, es decir como un pin bidireccional I/O.

La entrada $\mathbf{V_{DD}}$ es por donde se alimentará el chip con una tensión de 5V (valor compatible con sus especificaciones), la entrada $\mathbf{V_{SS}}$ será la conexión a tierra por donde se cerrará el circuito.

Cuando los pines se configuran según su función **GPX**, se comportan como puertos bidireccionales de entrada y/o salida, que se pueden programar para que adopten el valor que interese según la aplicación: 1 (voltaje de alimentación $\mathbf{V_{DD}}$, 5V) ó 0 (voltaje de tierra $\mathbf{V_{SS}}$, 0V).

Las funciones de los pines siete y ocho, **ICSPDAT** e **ICSPCLK** respectivamente, se utilizan en la programación del microcontrolador que tiene la finalidad de grabar en la memoria de programa del chip el programa diseñado.

La función **TOCKI** del pin cinco se utiliza cuando se quiere aumentar el temporizador del reloj a un tiempo determinado determinada.

En el pin dos la función **CLKIN** se utiliza cuando no se quiere trabajar con el reloj interno y se prefiere utilizar uno externo.

El pin cuatro tendrá tres posibles funciones; **GP3** se puede programar para que sea un pin de entrada, **MCLR** para forzar un *reset* en el microcontrolador y $\mathbf{V_{PP}}$ que se utiliza durante la programación del microcontrolador.

6.1.2. PIC16F690

Este segundo microcontrolador, que es un poco más complejo que el PIC12F509, es el que se utilizará en el vehículo de tres ruedas que se construirá debido a sus prestaciones y bajo costo. Entre las diferencias más importantes entre estos dos microcontroladores, cabe destacar que el PIC16F690 cuenta con veinte pines en vez de ocho, tiene tres temporizadores internos, además de ser capaz de trabajar con señales analógicas. **[2]**

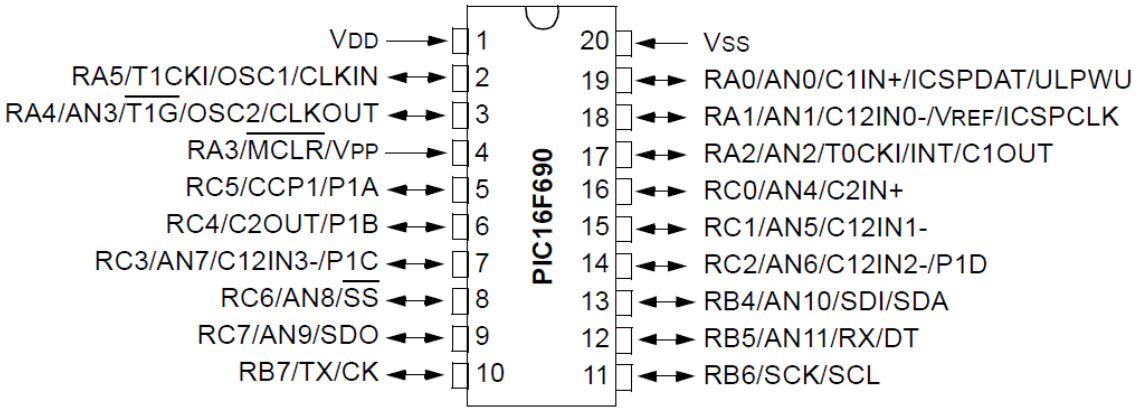

Figura 3. Pinout de PIC16F690.

Como se puede ver en la figura 3, muchas de las funciones que se describieron anteriormente se repiten en el PIC16F690, si bien pueden encontrarse en otro pin o

compartirlo con nuevas funciones.

De las nuevas funciones que nos encontramos hay que destacar las de **ANX** que permiten utilizar esos pines como unas entradas analógicas.

Los pines 5, 6, 7 y 14 tienen disponibles las funciones **P1A**, **P1B**, **P1C**, **P1D** que corresponden a las salidas del módulo PWM, cuando éste está activado, en el que se puede generar una señal de frecuencia y ciclo de trabajo (Duty Cycle) variable.

El pin 10 con función **TX,** y el pin 12 con función **RX,** es por donde el módulo **EUSART** enviará y recibirá la información digital en formato serie respectivamente. Esta funcionalidad será aprovechada para enviar o recibir información desde el módulo *bluetooth*.

6.2. PICKit 2

PICkit es una familia de programadores para microcontroladores PIC hechos por Microchip Technology. Se usan para programar y depurar microcontroladores, y para programar memorias EEPROM. Será el que transmitirá el código del programa desde el PC hasta el microcontrolador. **[3]**

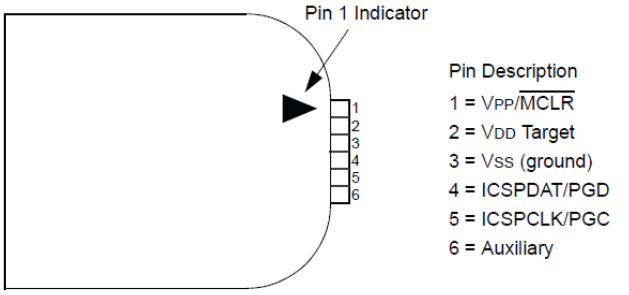

Figura 5. Pinout del PICkit 2.

Figura 4. Programador PICkit 2.

Como se observa en la figura 5, el PICkit cuenta con seis salidas que para poder programar el microcontrolador, se deberán conectar en su correspondiente pin.

La salida $\mathbf{V_{PP}}$ es por donde sale el voltaje de programación, cuando se aplica al dispositivo entrará en modo programación. Las salidas $\mathbf{V_{DD}}$ y $\mathbf{V_{SS}}$ son la tensión de alimentación y la conexión a tierra, respectivamente.

Por último el terminal **ICSPCL** es una señal de reloj unidireccional síncrono y en serie del programador al microcontrolador y el terminal **ICSPDAT** es un canal bidireccional síncrono en serie por donde circula la información. La salida seis no será utilizada.

6.3. Placa *protoboard*

Un *protoboard* es un tablero con orificios conectados eléctricamente entre sí, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado de circuitos electrónicos y sistemas similares. Está hecho de dos materiales, un aislante, generalmente un plástico, y un conductor que conecta los diversos orificios entre sí. Uno de sus usos principales es la creación y comprobación de prototipos de circuitos electrónicos antes de llegar a la impresión mecánica del circuito en sistemas de producción comercial. **[4]**

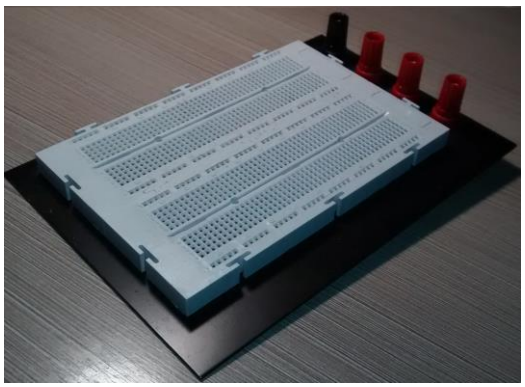

Figura 6. Placa protoboard.

Será sobre esta placa junto con los componentes electrónicos necesarios donde se realizarán la mayoría de ensayos y pruebas.

6.4. Otros componentes electrónicos

6.4.1. Cables

Conductor, generalmente de cobre recubierto de un material aislante o protector, que se utilizará para realizar las conexiones entre los patrones de línea del protoboard o entre el PICkit y el protoboard. **[5]**

6.4.2. Resistores

Se denomina resistor al componente electrónico diseñado para introducir una resistencia eléctrica determinada entre dos puntos de un circuito eléctrico. Se usan normalmente para reducir el flujo de corriente de un circuito, y al mismo tiempo, bajar el voltaje dentro del circuito. Está formado por un material de carbón y otros elementos resistivos para disminuir la corriente que pasa. **[6]**

Figura 7. Resistencia.

6.4.3. Resistores variables

Figura 8. Resistencia variable.

Es un tipo de resistor que es capaz de la en la regulación de la intensidad de corriente a través de la carga, de forma que se controla la cantidad de energía que fluye hacia la misma. **[7]**

6.4.4. Condensadores

Figura 9. Condensador.

Un condensador eléctrico es un dispositivo pasivo, capaz de almacenar energía sustentando un campo eléctrico. Está formado por un par de superficies conductoras en forma de láminas en situación de influencia total, todas las líneas de campo eléctrico que parten de una van a parar a la otra, separadas por un material dieléctrico o por el vacío. Las placas, sometidas a una diferencia de potencial, adquieren una determinada carga eléctrica, positiva en una de ellas y negativa en la otra, siendo nula la variación de carga total. Una vez cargado ya se tiene entre las dos placas una o tensión, y esta carga será soltada cuando se conecte un receptor de salida. **[8]**

6.4.5. Interruptores

Un interruptor eléctrico es un dispositivo que permite desviar o interrumpir el curso de una corriente eléctrica. Está formado por dos contactos de metal inoxidable y el actuante. Los contactos, se unen mediante un actuante para permitir que la corriente circule. El actuante es la parte móvil que en una de sus posiciones hace presión sobre los contactos para mantenerlos unidos. **[9]**

6.4.6. Regulador de Tensión

Figura 10. Regulador de tensión.

Un regulador de tensión es un dispositivo electrónico diseñado para mantener un nivel de tensión constante en su terminal de salida. El regulador utilizado en este proyecto, de salida regulada a 5V, cuenta con 3 pines, una entrada, una salida y la tierra o referencia. Por la entrada se introducirá el voltaje alto que como máximo podrá ser de 35V (en nuestro caso será 9V por la batería utilizada), la tierra o referencia es una conexión a tierra y la salida es por donde saldrá una tensión constante de 5V. **[10]**

6.5. Pantalla LCD

Una pantalla de cristal líquido o LCD, *Liquid Crystal Display,* es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora.

La pantalla que se utilizará en nuestro vehículo será en concreto una QC1602A v2, la cual cuenta con dieciséis pines, es capaz de mostrar dieciséis caracteres en cada línea y también permite la programación de caracteres personalizados. Para su programación es necesario utilizar la tabla de comandos y para la escritura de caracteres la tabla de caracteres, adjuntas en el anexo. **[11]**

La función de cada uno de los pines puede verse reflejada en la figura 11.

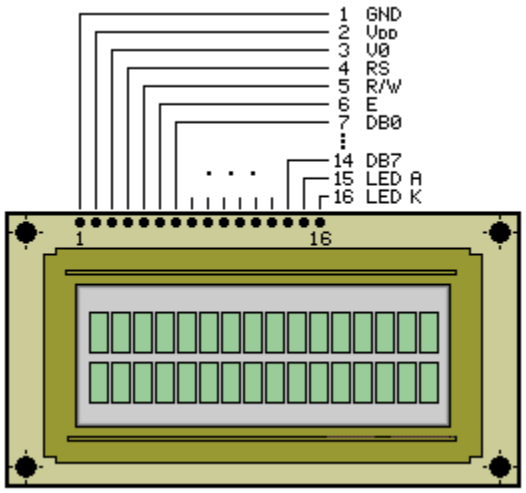

Figura 11. Pinout de pantalla LCD.

Figura 12. Pantalla LCD.

El pin **GND** y **VDD** son la conexión a tierra y la alimentación de 5V, respectivamente. El pin V0 se utiliza para controlar el contraste del visualizador, lo recomendable es conectarlo a una fuente de voltaje variable permitiendo variar el contraste.

El pin **RS** es el registro de control, cuando se encuentra en un valor bajo los bits transferido al visualizador son tratados como comandos mientras que si se le da un valor alto se puede enviar y recibir información de caracteres.

El pin **R/W** es la lectura/escritura, cuando tiene un valor bajo se usa para escribir comando o información de caracteres al módulo y un valor alto cuando se quiere leer información de caracteres o de estado.

El pin **E** es el encargado del envío de información de caracteres o comandos al módulo. Cuando se escribe hacia el visualizador la información se envía únicamente cuando esta señal pasa de un valor alto a otro bajo. En cambio, al leer del visualizador la información estará disponible después de una transición de valor bajo a alto de esta señal.

Los pines **DBX** forman el bus de datos que se utiliza para enviar los comandos o la información de caracteres al visualizador.

Por último, los pines **LEDA** y **LEDK** son el ánodo y el cátodo, respectivamente, del LED del fondo del visualizador.

6.6. Módulo *bluetooth* HC-05

En total tiene 6 pines pero únicamente se utilizarán 3 de ellos: el pin **TXD** por donde se enviarán las señales captadas con el módulo *bluetooth* hacia el exterior del módulo, el pin **5.0** que es la alimentación a 5V y el pin **GND** que es la conexión a tierra. Además también cuenta con un pin **RXD** es el que se tendría que utilizar si se quisiera enviar señales hacia el interior del módulo, el pin **3.3** una alimentación más baja y el pin KEY que se utiliza cuando se quiere configurarlo (como emisor o receptor, darle un nombre, etc).

Figura 13. Placa bluetooth.

El módulo *bluetooth* trabaja a unos 9600 baudios por segundo.

6.7. Motor BDC

Los dos motores que se utilizarán para desplazar el coche serán de corriente continua y con escobillas (Brushed DC motor). Este tipo de motor tiene un amplio rango de aplicaciones desde juguetes hasta botones para asientos de coche, ya que son económicos, fácilmente manejables y se encuentran en diferentes tamaños y formas.

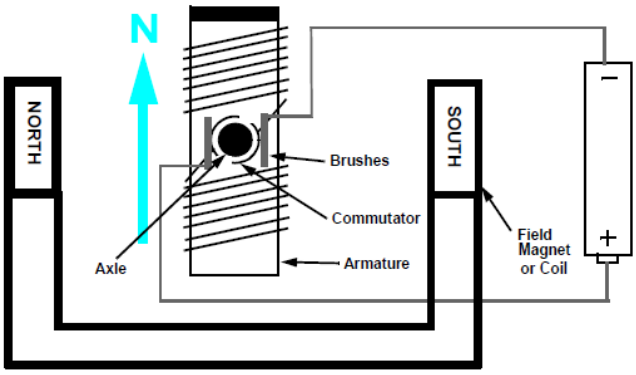

Figura 14. Componentes básicos en un motor BDC.

Estos motores cuentan con cuatro componentes básicos: un estator, un rotor, unas escobillas y un conmutador.

El estator generará un campo magnético estacional que rodeará al rotor, este campo será generado por unos imanes o bien por una bobina electromagnética.

El rotor o armadura estará constituido por una o más bobinas, que cuando se les de energía generarán otro campo magnético. Los polos magnéticos del rotor serán atraídos por los polos magnéticos opuestos del estator, provocando de esta forma que el rotor gire. Si la corriente que alimenta a las bobinas del rotor no cambiase no continuaría girando, ya que los polos del campo magnético del rotor una vez se han atraído a los polos opuestos del estator no se moverían. Al cambiar el sentido de la corriente que alimenta el rotor se cambian los polos del campo magnético del rotor, de manera que cuando los polos magnéticos del rotor se acercan a los polos opuestos del estator se cambian los polos del rotor continuando el giro. El cambio de los polos magnéticos en el rotor se llama conmutación.

Para lograr esta conmutación se hará de manera mecánica mediante las escobillas y el conmutador. Sobre el eje del motor habrá un recubrimiento segmentado de cobre que estará conectado con las bobinas del rotor llamado conmutador. El conmutador se encontrará en contacto con unas escobillas de carbono, las cuales estarán conectadas a una corriente eléctrica, por lo que a medida que el motor gira las escobillas se deslizan sobre el conmutador entrando en contacto con sus diferentes segmentos. De esta manera a medida que el rotor gira, las escobillas que alimentan a las bobinas del rotor, entran en contacto con diferentes partes del conmutador alternando la alimentación de las bobinas del rotor. **[12]**

Figura 15. Motores utilizados en el coche eléctrico.

Es importante destacar, que tanto las escobillas como el conmutador son partes del motor con tendencia al desgaste debido a su deslizamiento constante entre ellas.

Los motores que se utilizarán para este proyecto tienen una tensión de alimentación mínima de 6V y máxima de 9V.

6.8. Controlador: L293D

Figura 16.Placa con controlador L293D.

El controlador L293D es un cuádruple medio puente H de corriente alta, está diseñado para controlar cargas inductivas como relés, motores, solenoides y otros dispositivos de corriente alta y voltaje alto. Puede proveer de una señal bidireccional de hasta 600mA con voltajes de 4.5V a 36V.

Además, la placa usada en el proyecto, cuenta con 4 LEDs correspondientes a cada una de las salidas que se enciende o se apaga con mayor o menor intensidad en función de la corriente que circula por su salida.

También es importante destacar que cuenta internamente con diodos protectores encargados de proteger los motores de los picos de voltaje, que se producen en una carga inductiva cuando de manera repentina se reduce o remueve el voltaje. **[13]**

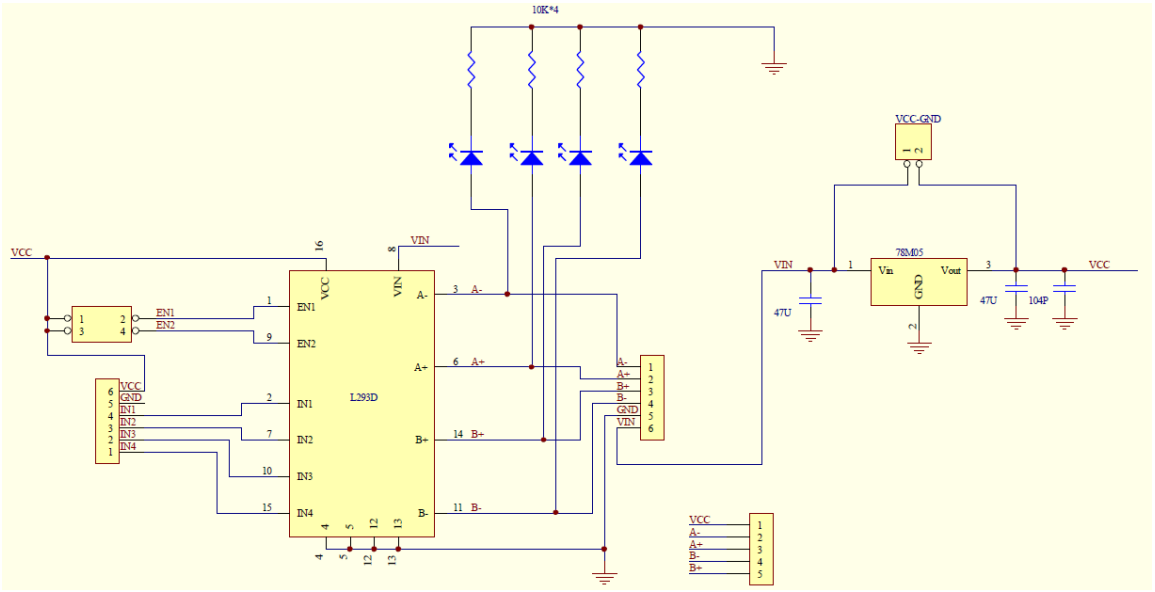

Figura 17. Circuito de L293D en la placa.

En este caso viene integrado ya dentro de una placa con las conexiones necesarias hechas, de tal manera que conectado nuestras entradas en las posiciones **INX** y sacándolas por **A±** y **B±** ya se tendrá la señal amplificada. En lo que respecta a la alimentación cuenta con las entradas **VCC** y **GND**, que son una entrada de 5V y la conexión a tierra respectivamente. Su función es la de alimentar la lógica interna del controlador por lo que solamente con esta ya sería capaz de funcionar. Pero los motores para que funcionen correctamente necesitarán al menos un voltaje de 6V o superior por lo que si únicamente se utiliza la alimentación anterior

resultaría insuficiente. Es por esto que también se cuenta con otra alimentación adicional exclusiva para los motores en las entradas **VIN** y **GND**, en que a la primera se le dará un voltaje de 6V o mayor y la otra es la tierra.

Este controlador permite controlar cuatro motores unidireccionalmente o dos motores de manera bidireccional, que es lo se utilizará en este caso para controlar los motores del coche eléctrico.

6.9. Sensor de distancia por ultrasonidos

El ultrasonido es una onda acústica cuya frecuencia está por encima del umbral de audición del oído humano. No se diferencia en propiedades físicas de las ondas acústicas convencionales solo en el hecho que los humanos no son capaces de oírlas, ya que se encuentran por encima de los 20 kHz.

Figura 18. Sensor de ultrasonidos.

El sensor con el que se trabajará cuenta con un transmisor de ultrasonidos, un receptor y un circuito de control. Su funcionamiento a grandes rasgos consiste en un pin **Trig** por el que se ha de enviar como mínimo un pulso de 10µs, para que el módulo se active y envíe automáticamente una ráfaga de 8 señales de ultrasonido a 40kHz. El módulo se encargará de captar y ampliar el eco de la señal si esta vuelve y transmitirlo mediante el pin **Echo**. Midiendo el ancho del pulso de la señal eco se puede calcular la distancia a la que se encuentra el obstáculo.

$$Distancia\,[cm] = \frac{Ancho\ del\ pulso\ [\mu s]}{58}$$

Ecuación 1. Cálculo de distancias.

Además cuenta con otros dos pines más el **Vcc** que es la alimentación a 5V y el pin **GND** que es la conexión a tierra. El alcance máximo del sensor es de 400 cm y el mínimo de 2cm. Para evitar que se superpongan el pulso de 10µs y la señal de eco se recomienda trabajar en ciclos de 60ms. El área del obstáculo tendrá que tener al menos 0.5m^2 y ser lo más liso posible de lo contrario afectará los resultados de la medida.

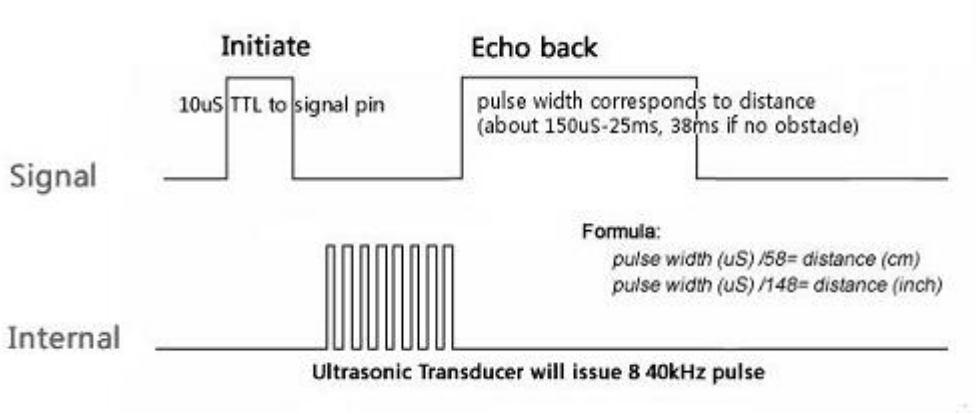

Figura 19. Señales enviadas y recibidas por el sensor de ultrasonidos.

7. Software para el microcontrolador

En este capítulo se describe el software utilizado para la programación del microcontrolador PIC.

7.1. MPLABX

MPLAB es un entorno de desarrollo integrado (IDE) gratuito, destinado a productos de la marca Microchip Technology. Este editor es modular, permite seleccionar los distintos microcontroladores soportados, además de permitir la grabación de estos circuitos integrados directamente al programador.

Es un programa que corre bajo Windows, Mac OS y Linux. Presenta las clásicas barras de programa, de menú, de herramientas de estado, etc. El ambiente MPLABX posee editor de texto, compilador y simulación.

A diferencia de otras versiones de MPLAB que fueron desarrolladas completamente interna, MPLABX está basada en el código abierto NetBeans IDE de Oracle. Siguiendo este camino ha permitido añadir muchas de las funciones más solicitadas de manera rápida y fácil, mientras a su vez dotando al programa de una arquitectura mucho más flexible para poder agregar nuevas funciones. **[15]**

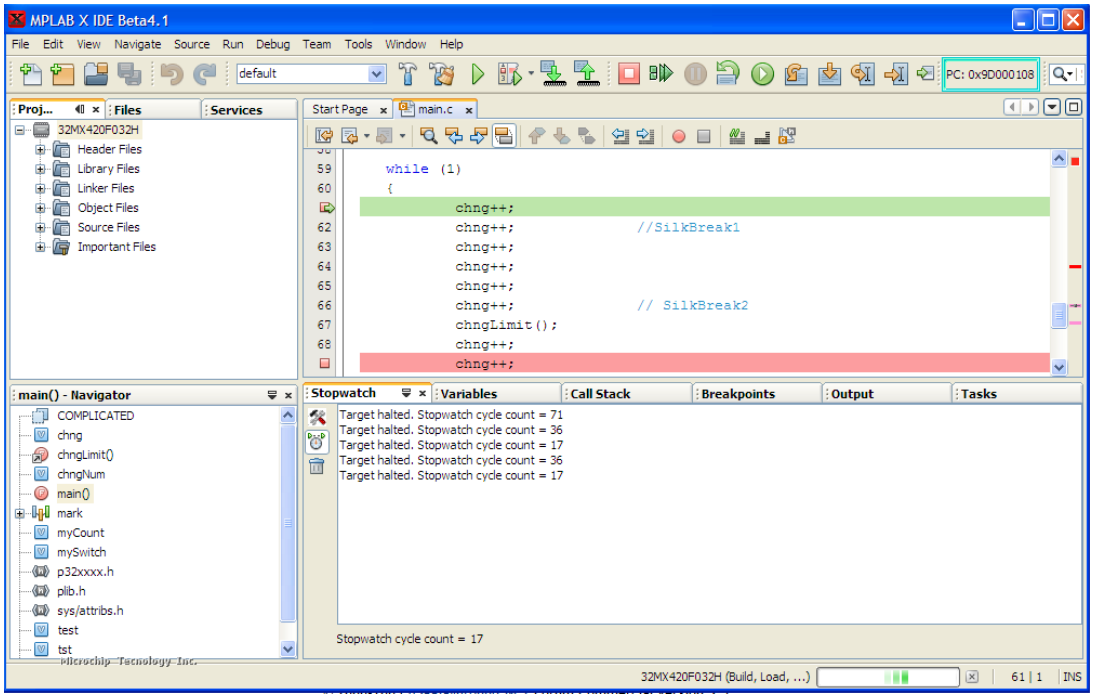

Figura 20. Interfaz de MPLABX.

7.2. HI-TECH C

HI-TECH C es un compilador gratuito perteneciente a Microchip Technology basado en el estándar ANSI C. Soportas todas las series de PIC10, PIC12 y PIC16 además de PIC14000 y los mejorados Mid-Range PIC de arquitectura MCU. Está disponible para diferentes sistemas operativos como Windows, Linux y Mac OS.

Además de ser una aplicación de consola independiente, es completamente compatible con la IDE MPLABX de Microchip Technology.

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación. En este caso traducirá del código escrito en C a un código que el microcontrolador sea capaz de entender en lenguaje ensamblador. **[16]**

8. Programación del microcontrolador

En este capítulo se describirá detalladamente y con ejemplos en el anexo la programación realizada de los diferentes módulos y sistemas utilizados en el microcontrolador PIC16F690.

8.1. Programación LCD

En lo que respecta al control de la placa LCD se utilizará el PIC16F690 interconectado como muestra la figura 21.

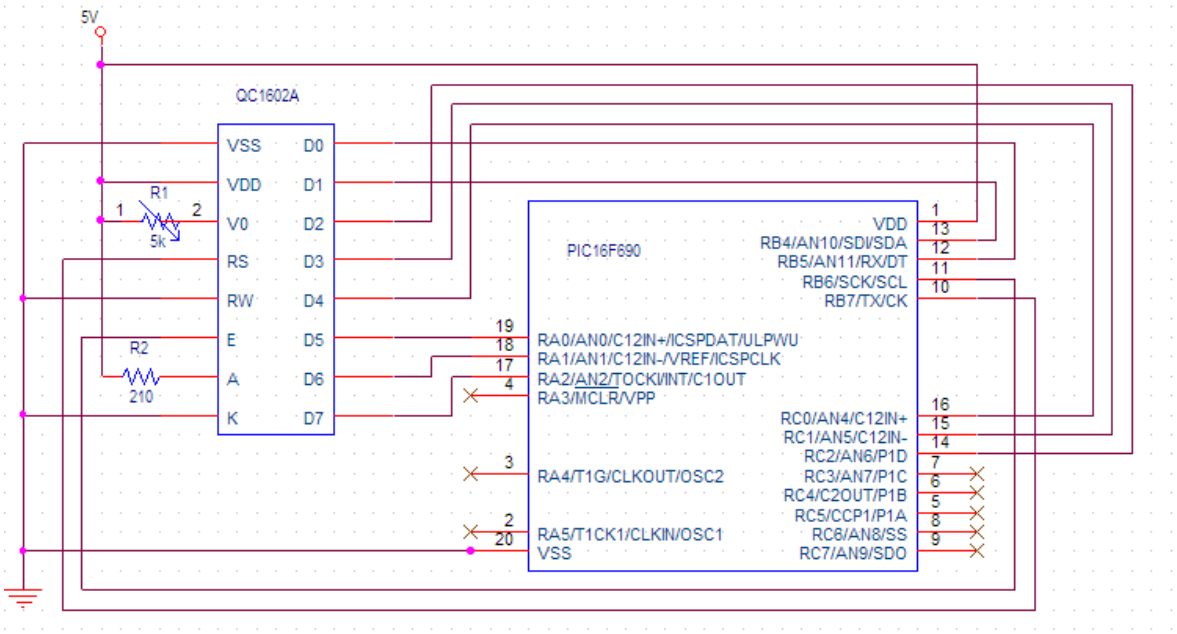

Figura 21. Circuito entre pantalla LDC y PIC16F690.

Para describir el funcionamiento y programación del LCD se utilizará como ejemplo el circuito 2, como se puede observar los pines **DBX** se han conectado con los pines del lado derecho del PIC16F690 por una cuestión de comodidad en la conexión, de la misma manera **RS** y **E** se han conectado a **RB7** y **RB6** respectivamente. Puesto que solo nos interesa enviar información al LCD (y no leer de la pantalla) se ha conectado el **RW** directamente a tierra. El pin **K** del cátodo del LED del visualizador del fondo también se conectará directamente a tierra. Entre la alimentación de 5V y el pin del ánodo **A** del LED del visualizador del fondo, se conectará una resistencia de 210 Ω para limitar la corriente que circule por dicho LED. De la misma manera, entre el pin **V0** y la alimentación se conectará una resistencia variable que nos permitirá controlar el contraste del LCD. **[17]**

Un problema a destacar en la programación del PIC16F690 en comparación con el PIC12F509 es la necesidad de seleccionar un tipo de modulo oscilador en concreto, debido a la gran variedad de diferentes tipos de relojes posibles. Finalmente se decidió trabajar con el oscilador de reloj **INTOSCIO** interno para liberar los pines correspondientes y, por lo tanto, con posibilidad de puertos en **RA4** y **RA5.** Ya que estos pines según el tipo de reloj que se seleccione puede que se necesiten como entrada o salida para una señal de reloj.

Otro posible problema es la configuración analógica de algunos pines a los que si no se les indica con los registros **ANSEL** y **ANSELH** con un 0, estos no actuaran de manera digital como puertos.

Algo a tener en cuenta siempre en los microcontroladores es que la memoria en la que se escribe el programa no borra el contenido anterior sino que lo sobrescribe. Esto puede provocar problemas si el programa no está bien cerrado, ya sea por ejemplo con un bucle infinito que continúe leyendo vestigios de un antiguo programa, provocando comportamientos extraños e imprevistos.

A la hora de programar nuestra placa LCD lo primero a hacer es configurarla mediante la tabla de comandos de control mostrada en Anexo I, enviando antes un 0 por **RS** para asegurarnos que estamos en modo comandos.

Por una cuestión de comodidad para no tener que trabajar con todas las salidas del microcontrolador relacionadas con los puertos **DBX** individualmente se ha creado una unión de todas estas llamadas RDATA. Después de darle a la unión RDATA el valor que nos interesa es necesario usar la función rdatasender() que iguala estos valores a sus puertos correspondientes, enviándolos hacia el LCD. Una vez enviados estos valores al LCD otra función, enable_up_low() que simplemente enviará una señal de valor alto a bajo para que la información de los puertos **DBX** sea procesada.

Cuando se ha configurado la pantalla como se desea, la segunda línea activada y el cursor en la posición deseada, se enviará por **RS** un 1 para entrar en modo escritura. En este modo para enviar una letra que se desee hay que seguir la tabla de caracteres estándares del Anexo II, que indica los valores que se han de enviar por **DBX** para que salga la letra. De la misma manera que antes se ha de seguir el proceso anterior, después de enviar la información a los puertos **DBX** con rdatasender() se usará la función enable_up_low() para que la información sea procesada y la letra salga por pantalla. Todo esto se puede ver en un programa en el Anexo III que retorna por la pantalla del LCD “HELLO WORLD”, programado de tal manera para que todo el proceso anteriormente descrito este lo más automatizado posible.

En lo que respecta a la creación de caracteres personalizados se puede ver en la tabla de caracteres estándares del Anexo II que se cuenta con 8 espacios de memoria para guardar los caracteres personalizados que se desee. Es importante destacar que una vez que el LCD pierde la corriente se perderá lo guardado en estos espacios de memoria.

Desde el modo comando se selecciona el comando correspondiente a programación de CGRAM, después estando en modo escritura se pasa a programar una a una las 8 filas que tiene cada casilla. Estas filas se programan simplemente usando los 5 bits menos significativos del RDATA que cada uno representa un punto en la fila del carácter, escribiendo un 1 en un bit se encenderá aquel punto en el carácter. En la figura 22 se puede ver un ejemplo.

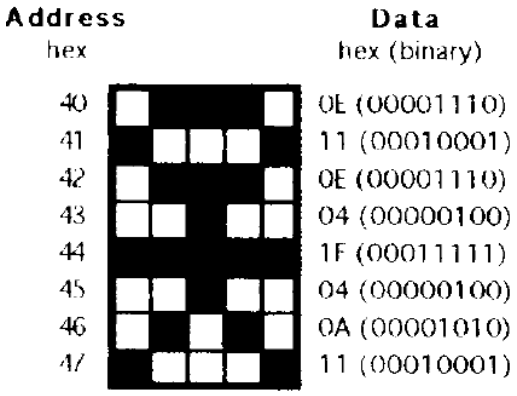

Figura 22. Ejemplo de carácter personalizado.

Una vez terminado con esto es importante que el cursor vuelva a su posición inicial. Para sacarlo por pantalla se escribirá como si fuera un carácter más. Un buen ejemplo de este proceso se puede ver en el código del Anexo IV.

8.2. Programación del motor

Para controlar los motores se utilizará el módulo PWM, *Pulse-Width Modulated signal* del PIC16F690, que permite la generación de una señal de frecuencia y ciclo de trabajo variable. Está señal modulada permitirá variar el voltaje medio que se envía, de este modo se podrá controlar la potencia que llega al motor y en consecuencia la velocidad a la que se mueve éste. Además también permitirá elegir el sentido en el que la corriente circula, seleccionando el sentido de rotación del motor.

El funcionamiento interno del módulo PWM consiste en que el microcontrolador en cuanto empieza a enviar un 1 activa el Timer2 que en el momento en el que llegue al valor determinado por **CCPR1L:CCP1CON<5:4>** dejará de enviar un 1 y empezará a enviar un 0. Por último hasta que el temporizador no llegue al valor determinado por el

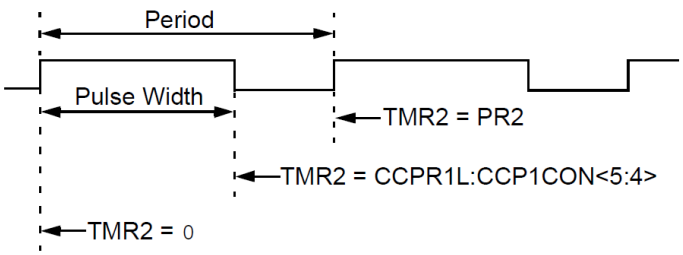

Figura 23. Señal modulada en función del Timer2.

registro **PR2** no volverá a enviar un 1. En cuanto llegue, el temporizador se reseteará empezando todo el ciclo anteriormente descrito otra vez. Todo esto se puede observar en la figura 23.

El Timer2 de 8 bits estará concatenado con 2 bits de su pre escala de esta manera se consigue un temporizador de 10 bits y es por eso que se puede comparar con **CCPR1L:CCP1CON<5:4>** que también tiene una longitud de 10 bits. En cambio para hacer la comparación con el registro **PR2** que es de 8 bits solo se comparará con Timer2. La razón por la que se realiza la comparación del ancho del pulso con 10 bits es para aumentar la resolución del ciclo de trabajo.

El tiempo del periodo de la señal modulada vendrá determinado por la ecuación 2, **PR2** siendo el registro que determina el momento en que se deja de enviar un 0, **Tosc** siendo el periodo de oscilación de nuestro reloj interno que por defecto es de 0.25µs, y la pre escala seleccionada para el Timer2 a la que se accede mediante el registro **T2CON.**

$$Period = [(PR2 + 1)] \times 4 \times TOSC \times TMR2Prescaler$$

Ecuación 2. Cálculo del periodo de PWM.

El ciclo de trabajo se determinará mediante la ecuación de 3, mediante el registro **CCPR1L** y los dos bits de **CCP1CON<5:4>,** el **Tosc** y la pre escala.

$$DC = CCPR1L:CCP1CON<5:4> \times TOSC \times TMR2Prescaler$$

Ecuación 3. Cálculo del ciclo de trabajo.

El microcontrolador cuenta con cuatro posibles salidas moduladas **P1A**, **P1B**, **P1C** y **P1D** que dependerán de la configuración que se les dé dentro del registro **CCP1CON** con los bits **P1M<1:0>,** ya sea salida única, puente H hacia delante, puente H hacia atrás o medio puente en H. Estas salidas moduladas para que funcionen habrán de estar configurados sus respectivos puertos como salidas. **[18]**

En el código del Anexo V se puede ver un ejemplo de código que utilizará la configuración de puente H hacia delante con un ciclo de trabajo del 0%.

8.3. Programación módulo ADC

El conversor analógico-digital, ADC (Analog to Digital Converter), permite la conversión de

una señal analógica a la representación binaria de 10 bits de esta señal. El funcionamiento de este módulo consiste en captar la señal analógica en el pin de entrada y compararla con el voltaje de referencia obteniendo su representación binaria. La conversión únicamente empezará cuando al bit **GO/DONE** se iguale a 0, tras producirse la conversión se actualizarán los registros **ADRESH** y **ADRESL** con el resultado de la conversión. Todo esto se puede observar en la figura 24.

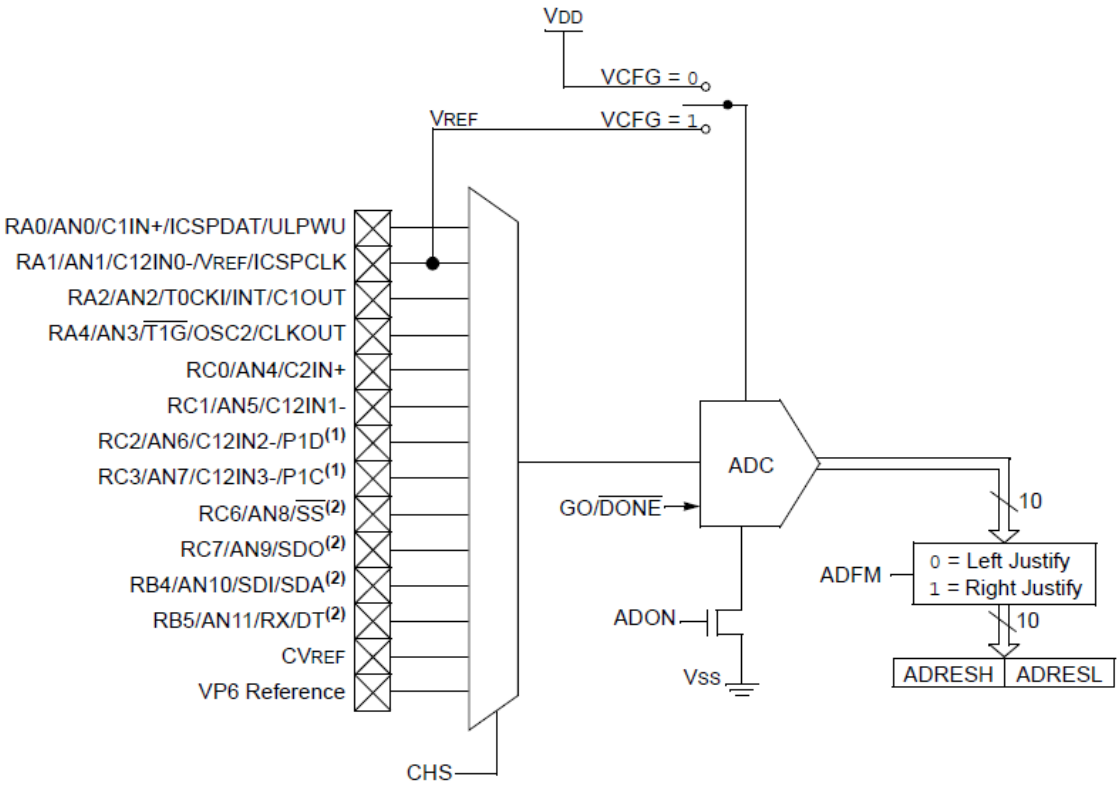

Figura 24. Diagrama de bloques de ADC

En el registro **ADCON0** se encuentran una serie de bits que controlan la programación del módulo ADC. Con el bit **ADCON** igual 1 se activa el módulo ADC. El software nos permite seleccionar mediante el bit **VCFG** el voltaje de referencia con el que trabajará el módulo ADC, se puede utilizar el voltaje de alimentación u otro aplicado al pin de referencia externa.

Es comparando el voltaje de la señal de entrada analógico con el de referencia, que el módulo realiza la conversión a binario. Por ejemplo, si el voltaje de referencia son 5V y la señal analógica de entrada son 3V. Los 5V serán el techo de la conversión binaria, ya que equivaldrán a 1111111111 que en decimal es igual a 1023. Siendo este el techo por una simple regla de tres se puede deducir que el valor de esta señal de entrada en decimal será de 624 y por lo tanto en binario de 1001110000.

Para seleccionar el pin de entrada de la señal analógica se usarán los 4 bits de **CHS<3:0>**. Además es necesario elegir el tipo de justificado que se desea mediante el bit **ADFM** que si se iguala a 0 se justifica a izquierda mientras que si se iguala a 1 se justifica a derechas, como se puede ver claramente en la figura 25 a continuación.

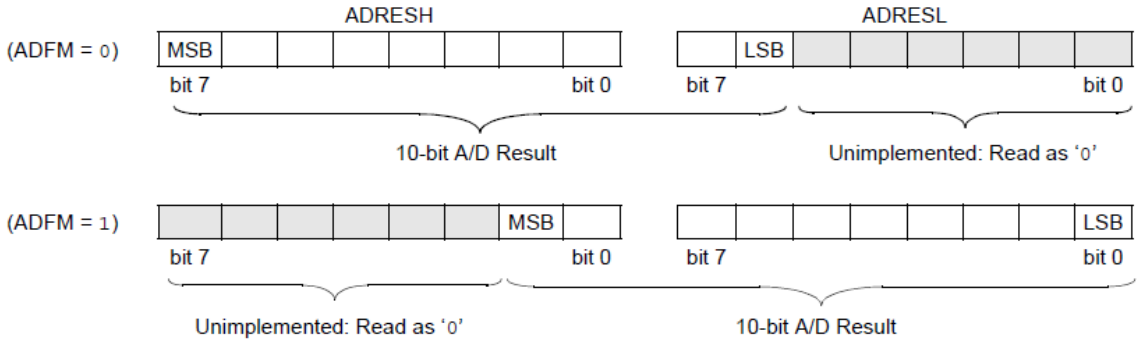

Figura 25. Resultado de conversión 10 bit A/D.

Por último se seleccionará el reloj de conversión ADC que es el tiempo necesario en completar la conversión de un bit, siguiendo las especificaciones indicadas por el fabricante para un **Fosc** de 4MHz corresponde un reloj de conversión de **Fosc/16**. Para seleccionarlo se usarán los bits **ADCS<2:0>** del registro **ADCON1**. **[19]**

En el Anexo VI se dispone de un ejemplo de configuración del módulo ADC.

8.4. Programación módulo EUSART

Para transmitir la señal del teléfono móvil al microcontrolador se utilizará la señal *bluetooth* enviada de este, por otro lado, para que el sistema diseñado basado en microcontrolador sea capaz de interpretarla necesitará contar con un módulo *bluetooth*. A través del módulo *bluetooth* se captará la información que envía el móvil y se redirigirá hacia el microcontrolador el cual la recibirá e interpretará mediante el módulo EUSART.

El módulo EUSART, acrónimo de *Enhanced Universal Synchronous Asynchronous Receiver Transmitter*, es un sistema de comunicaciones periféricas bidireccional que trabaja en serie. Cuenta con los generadores de relojes, registros de desplazamiento y búferes de datos necesarios para realizar el envío o recepción de datos de manera independiente de un programa en ejecución.

La información se envía y recibe siguiendo el formato estándar NRZ, *non-return-to-zero*. Se implementa utilizando dos niveles, uno de voltaje alto que representa un bit de valor 1 y otro

de voltaje bajo que representa un bit de valor 0. El significado del no retorno a cero se refiere al hecho que los bits que se transmiten de manera consecutiva con el mismo valor no es necesario que se retorne a una situación neutral entre cada bit que se envíe. Cada carácter que se envía cuenta con un bit que indica el comienzo seguido de ocho bits que contienen la información y terminado con un bit que indica el final. Los bits de comienzo son siempre 0 y los bits de fin son siempre 1. Cada bit transmitido persistirá por un periodo de 1/(tasa de baudios) Además cuenta con una tasa generadora de baudios que se utiliza para generar frecuencias estándar de baudios a partir del oscilador del sistema.

La información en EUSART se envía y recibe de manera independiente, por lo que el transmisor y receptor funcionan son completamente independientes entre sí.

Para captar las señales del módulo *bluetooth* se utilizará el sistema de recepción asíncrono que se describe en el figura 26.

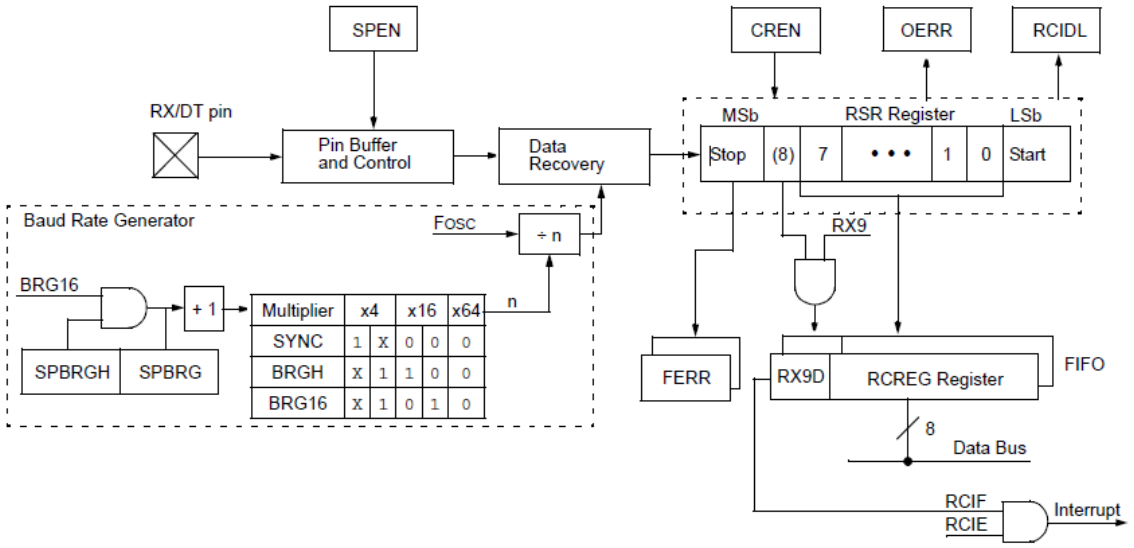

Figura 26. Diagrama de bloques de recepción EUSART

El BRG, *Baud Rate Generator* es un temporizador de 8 o 18 bits que se encarga de dar soporte a las operaciones síncronas y asíncronas del EUSART. Se utiliza para generar las tasas de baudios tanto del transmisor como del receptor EUSART, no tiene ningún otro tipo de función.

La información es recibida por el pin RX que la direcciona hacia el bloque de recuperación de datos, el cual es en realidad un registro de desplazamiento de alta velocidad que trabaja unas 16 veces la tasa de baudios. En el bloque de recuperación de datos inicia la recepción de caracteres con el flanco de bajada del primer bit. El primer bit que ha de ser siempre 0 ha de durar un tiempo de bit y medio para verificar que es el bit inicial del carácter, de lo contrario

el circuito de recuperación de datos lo descartará y seguirá buscando el primer bit del carácter. Cuando encuentre el primer bit proseguirá contando un tiempo de un bit para detectar el próximo bit en la serie, se leerá su valor 0 o 1 y enviara hacia el RSR, *Receive Shift Register,* que opera a la tasa de bits. Una vez que se han enviado todos los bits de datos al RSR, el bit final se envía el cual ha de ser siempre 1. Finalmente el carácter será enviado a la memoria búfer FIFO, *First-In-First-Out,* que es capaz de almacenar hasta dos caracteres. El primer carácter de FIFO será transferido en el momento que se lea el registro **RCREG**.

Lo primero que hay que hacer para programar el receptor del módulo EUSART es seleccionar la tasa de baudios deseada, esto se hará mediante diferentes opciones. El bit **BRG16** del registro **BAUDCTL** con un valor de 0 para seleccionar el modo 8 bits, con el bit **BRGH** del registro **TXSTA** con un valor de 1 se elige una velocidad alta en modo asíncrono y finalmente con los dos registros **SPBRGH** y **SPBRG** se determina el periodo del tiempo libre del temporizador de la tasa de baudios. Además se ha de tener en cuenta también la frecuencia interna de funcionamiento del reloj del microcontrolador. Finalmente para calcular la tasa de baudios se puede utilizar la tabla que se presenta a continuación.

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n+1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH, SPBRG register pair

Tabla 1. Fórmulas de tasas de baudio.

En nuestro caso sería un valor de 9600 que viene determinada por el módulo *bluetooth* utilizado ya que trabaja con esta tasa.

A continuación dando al bit **SPEN** del registro **RCSTA** un valor de 1 se habilita el funcionamiento de los pines 10 y 12 como **TX** y **RX** respectivamente. El bit **SYNC** de **TXSTA** se ha de igualar a 0 para seleccionar el modo asíncrono. Para desactivar el noveno bit hay que dar un valor de 0 al bit **RX9** y para activar la recepción dar al bit **CREN** un valor de 1, ambos corresponden al registro **RCSTA**.

En lo que respecta a las interrupciones si se desea activarlas hay que acceder al registro **INTCON,** poner el bit **GIE** a 1 para activar las interrupciones y el bit **PEIE** también a 1 para activar las interrupciones de los periféricos. Para terminar dando a **RCIE** un valor de 1 del

registro **PIE1** se activan las interrupciones del módulo EUSART cuando se produce una recepción. **[20]**

En el Anexo VII se puede observar un ejemplo de configuración de módulo EUSART.

8.5. Programación del sensor de ultrasonidos

La primera medida a tomar para la programación del sonar es la de enviar un pulso de 10µs para empezar el ciclo del sonar. Este pulso se enviará sencillamente poniendo uno de los puertos de salida a un valor de 1 durante el tiempo necesario y después poniéndolo en 0. Una vez hecho esto, el sensor de ultrasonidos enviará unas ocho señales a 40kHz y medirá la duración del pulso del eco. Por lo tanto, en el momento en que por el pin **Echo** del sensor pasa de un valor de 0 a 1 se iniciará un temporizador hasta que pase de 1 a 0 momento en que se detendrá el temporizador. Con este tiempo y utilizando la ecuación 1 se podrá calcular la distancia a la que se encuentra el obstáculo.

La distancia de seguridad necesaria para que el coche eléctrico sea capaz de detenerse sin colisionar con el obstáculo independiente de su velocidad, se ha determinado experimentalmente y es de unos 30cm. Siguiendo la ecuación 1 (apartado 6.9) esto nos da un tiempo de 1740µs, es decir que cuando el eco tarde un tiempo mayor al indicado anteriormente no se habrá de producir ninguna alteración en la marcha del coche.

Para contar el tiempo que tarda el pulso en volver se utilizará el Timer0 de 8 bits que viene integrado con el microcontrolador. Sabiendo que el reloj interno trabaja a 4MHz cada uno de los 255 tics del temporizador será 1µs, por lo tanto si queremos abarcar un tiempo de 1740µs hará falta utilizar la pre escala 1:8. En resumen cada uno de los tics del temporizador serán 8µs, por lo que se comprueba que cuando si el t ha medido un pulso de 218 el tiempo transcurrido habrá sido de 1744µs. En definitiva cualquier valor del temporizador inferior a 218 indicará que se ha de detener la marcha del coche. Cabe tener cuidado con el hecho de que una vez que el temporizador ha superado el 255 comenzará de 0, con tal de evitar mediciones erróneas se habrá de añadir la condición de que no se produzca *overflow* en el temporizador. Esto se podrá hacer utilizando la bandera **T0IF** del registro **INTCON**, la cual adopta un valor de 1 cuando se produce un *overflow* en el Timer0. El *overflow* se produce cuando un temporizador rebasa su valor máximo, es decir 255, y por lo tanto empezará a contar desde 0 nuevamente.

En el Anexo VIII se puede observar un ejemplo de programa para el sonar de ultrasonidos.

9. Programación de la Aplicación Móvil

En este capítulo se explica cómo se realizó la programación de la aplicación móvil de controlar el vehículo.

9.1. MIT App Inventor

Para crear la aplicación móvil que se encargará de enviar las señales de comando mediante *bluetooth* al coche eléctrico. Se utilizará *MIT App Inventor* que es una herramienta de programación basada en bloques que permite a cualquiera, incluso principiantes, comenzar a programar construyendo aplicaciones completamente funcionales para *Android*. Los recién llegados a *App Inventor* pueden tener creada su primera aplicación en menos de una hora, y pueden programar complejas aplicaciones en un tiempo significativamente menor en comparación con las basadas en los lenguajes tradicionales.

La creación de la aplicación en *App Inventor* consta de dos partes, la interfaz gráfica de la App y la programación en bloques de ésta. **[21]**

9.2. Interfaz gráfica

Cuando se construye la interfaz es el momento en que se determina como se transmitirá la información y, por lo tanto, como funcionarán los comandos sobre el coche eléctrico. Para confeccionar la se han utilizado principalmente botones y deslizadores. Se ha mejorado su aspecto estético incorporando unos botones personalizados, unas flechas para las direcciones y un fondo de temática electrónica.

Uno de los problemas de esta aplicación es el complicado posicionamiento de los diferentes elementos en la posición deseada en la pantalla. La utilización de esta aplicación en la creación de la interfaz no es para nada intuitivo.

A continuación en la figura 27 se puede apreciar la interfaz gráfica que dispondrá la aplicación móvil, que contará tanto con botones como con barras deslizadores.

Dentro de la categoría **Opciones,** se tendrán dos botones: **Sensor** que activa y desactiva el detector de obstáculos y **Bluetooth** que conecta el móvil con el módulo *bluetooth* del coche eléctrico.

En la siguiente categoría **Dirección,** se pueden observar 4 flechas y una señal de stop. Las flechas que apuntan hacia el norte y sur son las encargadas de determinar el sentido de la marcha del coche, ya sea hacia delante o hacia atrás. Por otro lado las flechas que apuntan hacia este y oeste se encargan de determinar un giro a la izquierda o derecha. Finalmente el botón de **Stop** detendrá el coche eléctrico en el lugar donde se encuentre.

Es importante destacar que si se pulsa alguna de las flechas de giro sin haber determinado un sentido previamente, el coche por defecto efectuará un giro hacia delante.

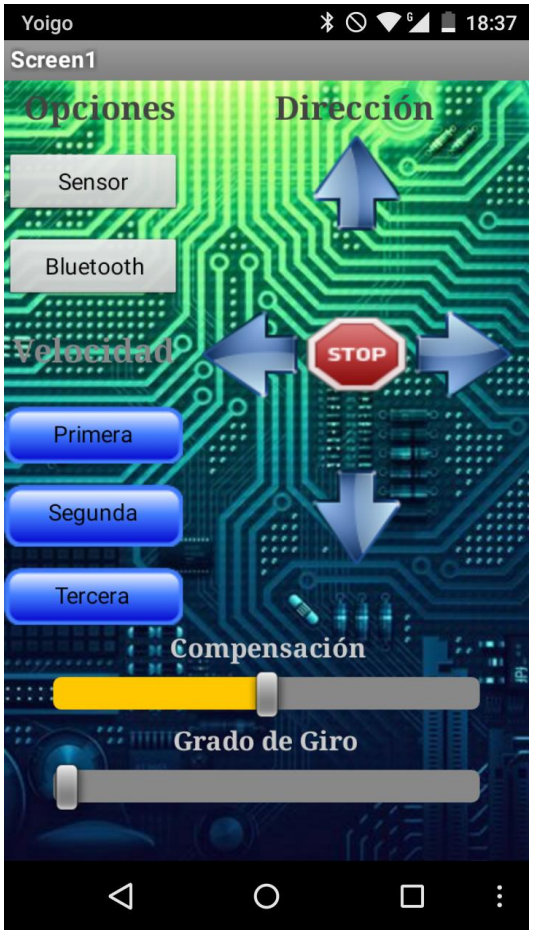

Figura 27. Interfaz gráfica de la aplicación móvil.

En la categoría de **Velocidad** únicamente hay 3 botones cada uno con velocidad mayor que la del anterior. **Primera** será la velocidad más lenta, **Segunda** será la velocidad Intermedia y **Tercera** la velocidad mayor. Cabe mencionar que después de pulsar **Stop** automáticamente se seleccionará la **Primera**.

Por último quedan los deslizadores de **Grado de Giro** y de **Compensación**.

El primero es el que determinará el tipo de curva que trazará el coche parar girar tanto a derecha o a izquierda. El deslizador en este caso sólo tiene 3 posiciones, a medida que se mueva más hacia la derecha pasará a la siguiente posición y la curva que trace será más abierta.

El segundo deslizador actúa únicamente sobre el motor izquierdo. Tiene 20 posiciones y empieza en su punto medio. A medida que se deslice más hacia la izquierda se irá incrementando la velocidad a la que se mueve el motor izquierdo y, en cambio, si se desliza a la derecha se irá reduciendo la velocidad del motor izquierdo. El objetivo de este deslizador es el de efectuar correcciones sobre el rumbo del coche cuando se quiere desplazar en línea recta, por lo tanto solo tendrá efecto cuando se esté desplazando hacia delante o hacia

atrás.

9.3. Programación en Bloques

La programación en bloques de la aplicación está íntimamente relacionada con la interfaz que se ha creado anteriormente, ya que se tendrán que utilizar los elementos determinados anteriormente en la programación en bloques.

El primer bloque de programación a realizar es el encargado de conectar el móvil con el módulo Bluetooth del coche. Por eso como se puede observar en la figura 28 al pulsar el botón **Bluetooth** el cliente se conectará con la dirección "00:14:02:10:51:48" que es el número MAC del módulo Bluetooth que se utiliza. La dirección MAC es un identificador de 48 bits (6 bloques hexadecimales) que corresponde de forma única a una tarjeta o dispositivo de red. **[22]**

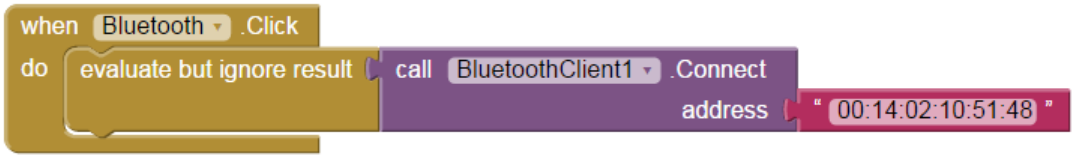

Figura 28. Bloque de programación cliente bluetooth.

Seguidamente se continuará por la programación de los elementos relacionados con la dirección del coche, en el que sencillamente cada comando tendrá relacionado un número determinado. De esta forma, cada orden que se envíe tendrá un número único e inconfundible. Como se puede ver en la figura 29 cada vez que se pulse alguno de los botones de dirección creados en la interfaz el cliente *bluetooth* enviará un número que identificará esa orden.

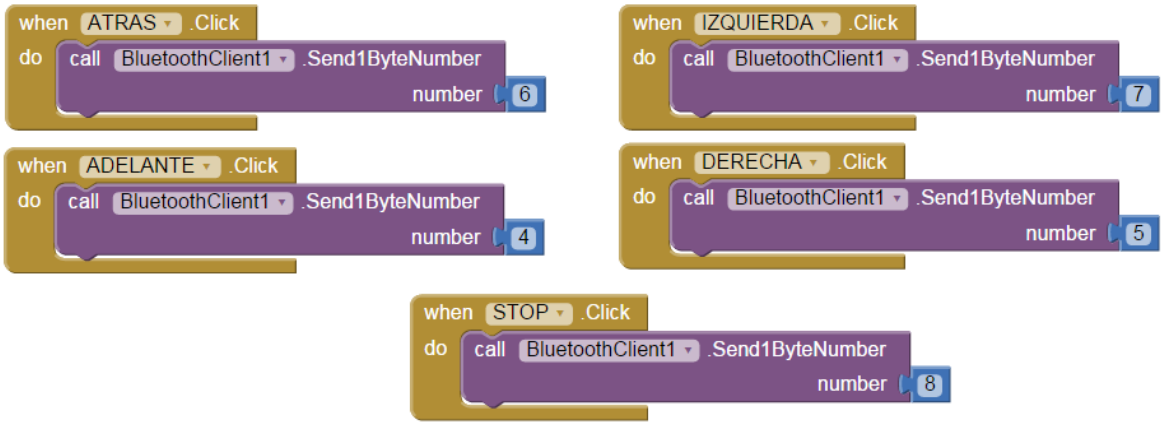

Figura 29. Bloques de programación de los elementos de dirección.

Lo mismo que se ha explicado para los elementos de dirección es aplicable para los elementos relacionados con la velocidad y el botón encargado del sensor, como se puede apreciar en la figura 30.

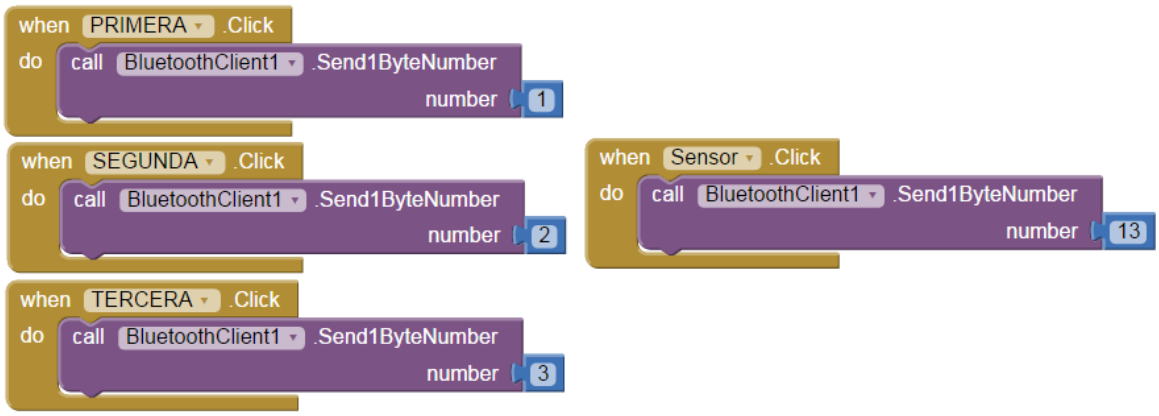

Figura 30. Bloques de programación de los elementos de velocidad y el sensor.

Finalmente se realizará la programación en bloque de los deslizadores que funcionaran de manera similar. Debido a que el microcontrolador a veces no era capaz de procesar todas las señales que le enviaba el módulo *bluetooth* mediante los deslizadores es necesario limitar el número de señales que se envían. Para hacer esto se ha utilizado un reloj en la aplicación que limita al tiempo mínimo entre señales de los deslizadores a unos 500ms. Además también se necesitará una variable bandera y una variable para almacenar el valor de la posición del deslizador.

Como se puede observar en la figura 31, el bloque reloj es el encargado de que únicamente se puedan enviar señales cada 500ms pero como solamente estamos interesados en que estas señales se envíen si el deslizador ha sido manipulado por el usuario previamente se necesita la variable bandera. Por defecto la bandera vendrá inicializada como *false*, pero cuando se modifique la posición del deslizador esta bandera adoptará un valor de *true*, además se obtendrá y redondeará el valor de la posición del deslizador. Ahora que la bandera está definida como *true* se puede acceder al bloque del reloj ya que se cumple la condición de entrada, por lo tanto el cliente *bluetooth* enviará el valor de la posición del deslizador y cambiará el valor de la bandera a *false.*

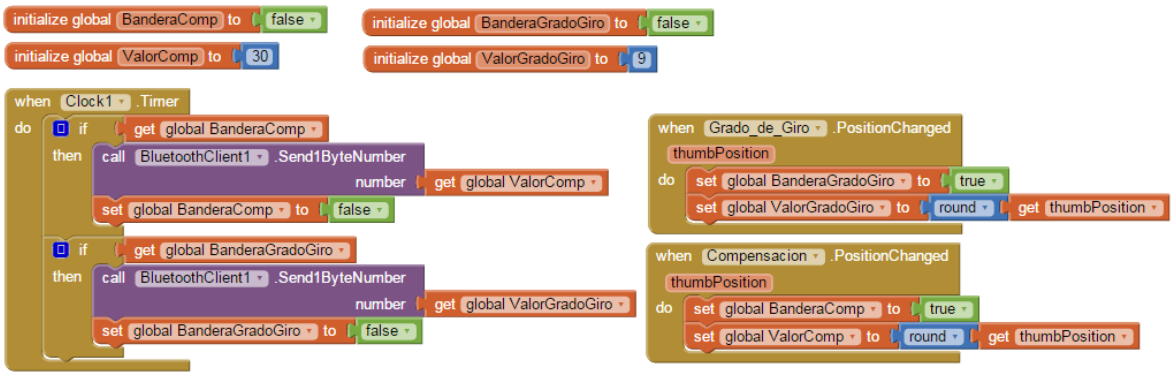

Figura 31. Bloques de programación de los deslizadores compensación y grado de giro.

Como se puede observar, gracias a *App Inventor* una persona sin ningún tipo de conocimiento en informática es capaz de programar y diseñar una aplicación que utiliza un cliente *bluetooth* para comunicarse de una manera muy sencilla. En cambio para hacer esto con un lenguaje de programación convencional se hubiera necesitado unos conocimientos importantes de informática.

10. Sistema de control del coche eléctrico

En este capítulo se hace una explicación detallada de cómo funcionan e interaccionan entre si diferentes sistemas y módulos que conforman el sistema de control del coche eléctrico.

10.1. Funcionamiento general

El funcionamiento del sistema de control en líneas generales se puede seguir mediante la figura 32 y explicado a continuación. Por disponer el PIC16F690 de sólo un módulo PWM, se han utilizado dos; uno para cada motor. El código utilizado para la programación del microprocesador izquierdo se encuentra disponible en el Anexo X y el del derecho en el Anexo IX.

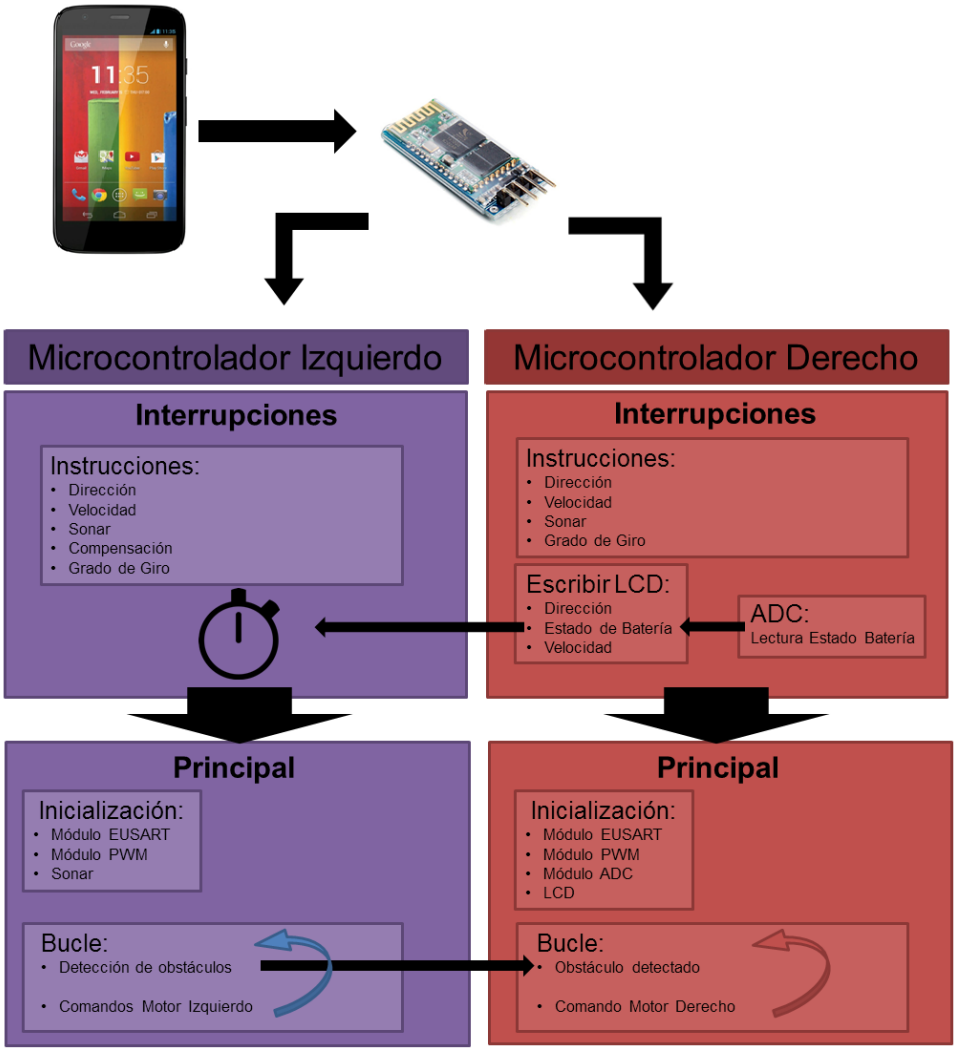

Figura 32. Funcionamiento interno de los microcontroladores izquierdo y derecho.

Una vez que las señales enviadas por el móvil han sido captadas por el módulo *bluetooth* y reenviadas hacia los microcontroladores, estos utilizando sus respectivos módulos EUSART interpretarán la señal. En todo momento en que se termine de interpretar una señal en el módulo EUSART se producirá automáticamente una interrupción en el programa de los microcontroladores. Al producirse una interrupción el microcontrolador abandonará la función principal que este ejecutando en ese momento para pasarse a la función interrupción. Es decir, siempre que se ejecuta una orden desde la aplicación móvil se producirá por lo tanto una interrupción en los microcontroladores. Como se ha explicado anteriormente cada botón o posición del deslizador de la interfaz está relacionado con un valor determinado, una orden en concreto. Según la orden que se ejecute desde la aplicación móvil el valor enviado será distinto, según el valor enviado las instrucciones dentro de la función interrupción alteraran una variable de estado u otra.

Las variables de estado son un conjunto de variables globales que mediante la modificación de sus valores se encargan de controlar el funcionamiento interno y externo del coche. Una variable adoptará un estado u otro según la función que tenga que cumplir, por ejemplo la variable **Velocidad** puede adoptar tres estados distintos: **Primera**, **Segunda** y **Tercera**. En la tabla 2 se pueden observar todas las variables que se utilizan con sus correspondientes posibles estados. La única manera de modificar el estado de una de estas variables es mediante un comando ejecutado a través de la aplicación móvil. De esta manera si desde la aplicación se selecciona el botón correspondiente a **Segunda,** se estará alterando el valor de la variable **Velocidad** cambiándolo por el nuevo estado seleccionado. Al alterar el estado de esta variable por consecuencia también se está alterando el funcionamiento que tendrá el coche. En definitiva, es la alteración de las variables de estado mediante la aplicación móvil lo que ejecuta las órdenes del usuario sobre el coche eléctrico.

Variables	Estados
Sentido	Adelante
	Atrás
	Nointroducido
Dirección	Izquierda
	Derecha
	Priodidad
	Nointroducido
Velocidad	Primera
	Segunda
	Tercera
EstadoSensor	Activado
	Desactiva
GradoGiro	GradoUno
	GradoDos
	GradoTres
EstadoBateria	Cargada
	Semicargada
	Descargada

Tabla 2. Variables de estado.

Una vez que el microcontrolador ha terminado la ejecución de la función interrupción volverá la función principal, en el punto en que la había abandonado anteriormente cuando se produjo la interrupción. Al dar corriente a los microcontroladores lo primero que harán estos es ejecutar la función principal, que es donde se inicializan los módulos EUSART, PWM,

ADC y se configuran las opciones necesarias para usar el LCD y el Sonar. Inicializado todo esto se pasará a entrar en un bucle infinito, que es donde se produce la ejecución de órdenes. En la función interrupción se produce el cambio de las variables de estados, pero es dentro del bucle infinito de la función principal donde se ejecutan las órdenes. Al alterar las variables de estado las condiciones de las funciones dentro del bucle infinito cambian, por lo tanto cambiando la ejecución de las órdenes.

10.2. Recepción de señales mediante módulo bluetooth

Se conectará la salida **TXD** del módulo *bluetooth* a las entradas **RX** de los dos microcontroladores, posteriormente los módulos EUSART traducirán estas señales y los microcontroladores actuarán de la manera que les corresponda. Todo esto se puede apreciar en la figura 33.

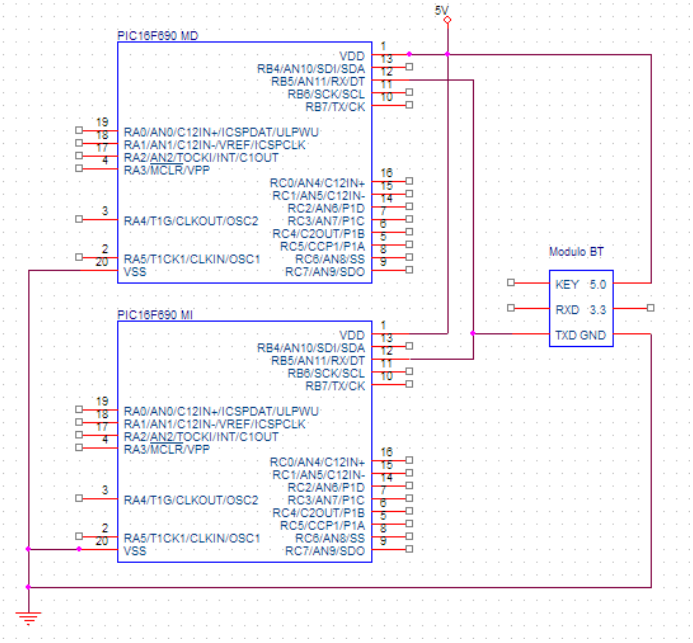

Figura 33. Circuito del módulo bluetooth a los microcontroladores.

A medida que se aumentaba el número de sistemas internos integrados en los microcontroladores, se observó que estos no eran siempre capaces de procesar todas estas señales. Por este motivo no solo se optó por establecer unos intervalos de 400ms dentro de los bucles infinitos, sino que también se limitó la cantidad de señales que era posible enviar desde la aplicación móvil.

10.3. Control de motores

Para controlar dos motores de manera independiente bidireccionalmente es necesario un microcontrolador PIC16F690 por motor. Esto se debe a que como se explicó en el apartado 8.2 si bien se pueden tener dos salidas moduladas y dos puertos, estas salidas moduladas tendrán valores complementarios. Es decir, si en una salida su ciclo de trabajo es de 40% en la otra el ciclo será del 60% y como para regular el grado de giro será necesario utilizar ciclos de trabajos distintos en cada motor, se optó por trabajar con dos microcontroladores.

En la tabla 3 se observa el tipo de tensión que se ha de aplicar en los bornes de cada motor para que se efectúe la orden dada. **P1C** es un puerto que puede adoptar un únicamente dos valores 0 o 1. **P1A(PWM)** es la salida modulada que según el ciclo de trabajo tendrá un valor del 0% al 100%, aunque en la tabla solo se ven sus valores máximos y mínimos.

	Motor Izquierdo		Motor Derecho	
	P1C	P1A(PWM)	P1C	P1A(PWM)
Adelante	1	0	1	0
Giro Derecha Adelante	1	0	0	0
Giro Izquierda Adelante	0	0	1	0
Atrás	0	1	0	1
Giro Derecha Atrás	0	1	0	0
Giro Izquierda Atrás	0	0	0	1

Tabla 3. Tensión en bornes de los motores para ejecutar una orden determinada.

En la fila de comando **Adelante** la mayor velocidad será cuando el ciclo de trabajo de las salidas **P1A(PWM)** sea menor, al aumentar el ciclo de trabajo se reducirá la velocidad ya que se reduce la diferencia de potencial entre bornes del motor. En cambio para el comando **Atrás** esto se produce de manera inversa, cuando el ciclo de trabajo en **P1A(PWM)** sea mayor la tensión entre bornes del motor será mayor y al reducir el ciclo de trabajo se reducirá la tensión. En la tabla 4 se pueden observar los ciclos de trabajo utilizados en los comandos **Adelante** y **Atrás**.

	Primera	Segunda	Tercera
Adelante	45	30	10
Atrás	55	70	90

Tabla 4. Ciclos de trabajo utilizados según orden y velocidad.

Como se ha hablado anteriormente cuando el coche eléctrico se desplaza hacia adelante o hacia atrás debido a la configuración física de sus ruedas es posible que no logre hacerlo en una línea completamente recta. Por este motivo se ha dotado a la interface un deslizador que actúa sobre el motor izquierdo, de manera que en el caso que se desvíe de la trayectoria

deseada el usuario sea capaz de corregirlo. La compensación como se ve en el diagrama 1 como sólo se produce en el motor izquierdo estará presente únicamente dentro de la función interrupción del microcontrolador izquierdo. Su funcionamiento consiste que en cualquiera de las velocidades que se encuentre el coche cuando se desplaza con dirección hacia delante o marcha atrás, se puede aumentar o disminuir el ciclo de trabajo del motor izquierdo. Aumentado o disminuyendo la velocidad de este motor a voluntad del usuario.

En lo que respecta a los giros uno de los motores tendrá que tener una tensión entre bornes mayor que el otro para que se produzca el giro. En cuanto mayor sea la diferencia de tensión entre motores más cerrado será el giro mientras que si la diferencia es muy pequeña se producirá un giro muy abierto. En los giros uno de los motores siempre tendrá la misma diferencia de tensión entre bornes y utilizará los mismos ciclos de trabajo que se indican en la tabla 4. El otro motor tendrá una diferencia de tensiones entre bornes variables según el grado de giro que se le indique. En la tabla 5 están los ciclos de trabajo utilizados en los grados de giro según la velocidad seleccionada, como se puede observar en el **Giro Atrás** los ciclos de trabajo en cada velocidad son siempre menores a los que se ven en la tabla 4 en el sentido **Atrás**. El motor exterior que traza las curvas siempre ha de tener una tensión entre bornes mayor que la del motor interior, por ejemplo en un **Giro Atrás** de **Grado Uno** en la **Primera** velocidad se usa un ciclo de trabajo de 25 para la rueda interior y uno de 55 para la rueda exterior. Lo mismo se sigue para un **Giro Adelante** la diferencia es que no se ve tan claro en estas tablas ya que el puerto **P1C** tiene un valor de 1, por lo que la tensión en bornes del motor es 100 menos el ciclo de trabajo de la salida modulada.

		Primera	Segunda	Tercera
Giro Adelante	Grado Uno	0	0	0
	Grado Dos	75	60	40
	Grado Tres	55	40	20
Giro Atrás	Grado Uno	0	0	0
	Grado Dos	25	40	60
	Grado Tres	45	60	80

Tabla 5. Ciclos de trabajo utilizados según orden y velocidad.

Un problema que presenta el uso de ciclos de trabajos para determinar las velocidades es que estas dependerán directamente de la tensión de las pilas. Si las pilas son nuevas estarán completamente cargadas por lo tanto se moverá más rápido, por el otro lado si las pilas están gastadas las velocidades serán menores.

En la figura 34 se aprecian los dos microcontroladores que se utilizan, el de arriba es el encargado del control del motor derecho y el de abajo es el encargado del motor izquierdo las salidas. La programación de estos microcontroladores será la de salida única, es decir

que **P1A** será la salida modulada y **P1B**, **P1C**, **P1D** serán puertos.

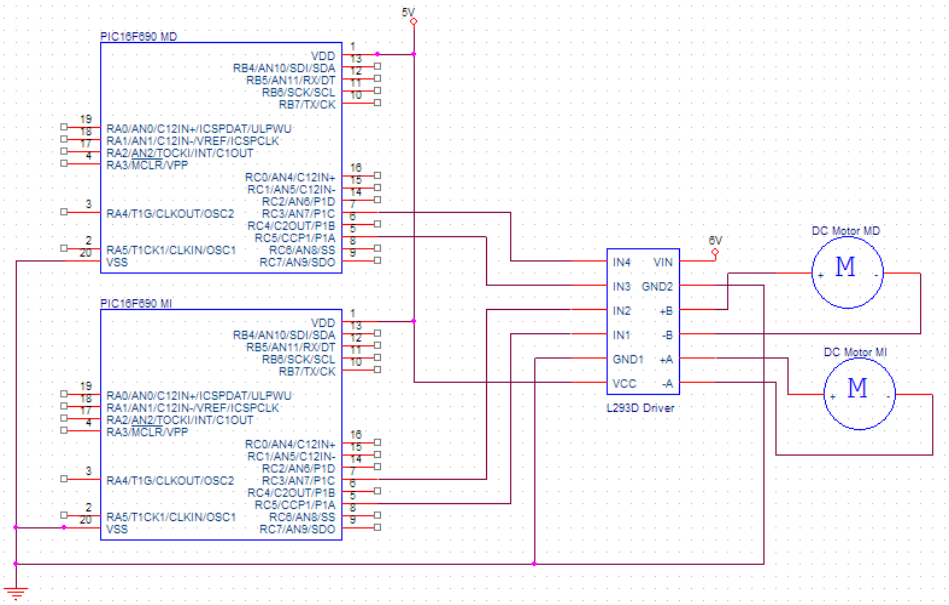

Figura 34. Conexión de los motores, placa L293D y microcontroladores.

Los pines de **P1A** y **P1C** del motor derecho se conectarán a las entradas **IN4** e **IN3** y que su señal saldrá amplificada por **+B** y **–B**. Para el motor izquierdo será similar solo que se conectará en **IN2** e **IN1** y saldrá por **+A** y **–A**. Hay que recordar que por la entrada **VCC** es la alimentación lógica del driver L293D mientras que por **VIN** es la alimentación de elevación de tensión. Finalmente se conectan los bornes de cada motor a sus correspondientes salidas amplificadas.

10.4. Pantalla LCD

El microcontrolador encargado controlar la placa LCD será el microcontrolador derecho. Los conexiones como se puede apreciar en la figura 34 han variado en comparación con las que se usaron en el ejemplo del apartado 8.1, esto se debe a que algunas de los pines que se usaron en un principio se tuvieron que cambiar ya que se tenían que usar para otros fines.

La escritura de los caracteres por pantalla como se puede ver en el diagrama 1 no se hará dentro de la función principal sino dentro de la función interrupciones. El motivo de esto es que de realizarse la escritura dentro de la función principal, que es un bucle infinito que funciona a intervalos de 400ms. Por esto se vería constantemente por la pantalla del LCD la actualización de los caracteres que puede resultar incómodo y poco estético. Poner la escritura en la función interrupción tiene más lógica ya que los mensajes del LCD solo cambian cuando se alteran las variables de estado y siempre que se altere una se accede

automáticamente a la función interrupción.

El mensaje que se escribirá por la pantalla dependerá de las variables de estado de **Sentido**, **Dirección**, **Velocidad** y **Estado de Batería**. En la primera línea de la pantalla del LCD se escribirá la dirección que esté tomando el coche eléctrico en ese momento, que puede ser: “Avanzando”, “Marcha Atrás”, “Girar Derecha” o “Girar Izquierda” y si se pulsa el botón de **Stop** el mensaje será “Parada”. En la segunda línea lo que aparecerá será la velocidad a la que se esté moviendo en ese momento, ya sea: “Primera”, “Segunda” o “Tercera”. Por último en la esquina inferior derecha de la pantalla LCD aparecerá un símbolo con el estado de las pilas, que tiene tres posibles estados: “Cargada”, “Descargada” o “Semidescargada”.

Los símbolos de las pilas no vienen dentro de los caracteres estándares con los que trabaja la pantalla, es por esto que se han de programar en la inicialización del microcontrolador derecho.

Para leer el nivel de tensión de las pilas se utiliza el módulo ADC, se realiza una lectura en cada interrupción del estado actual de las pilas.

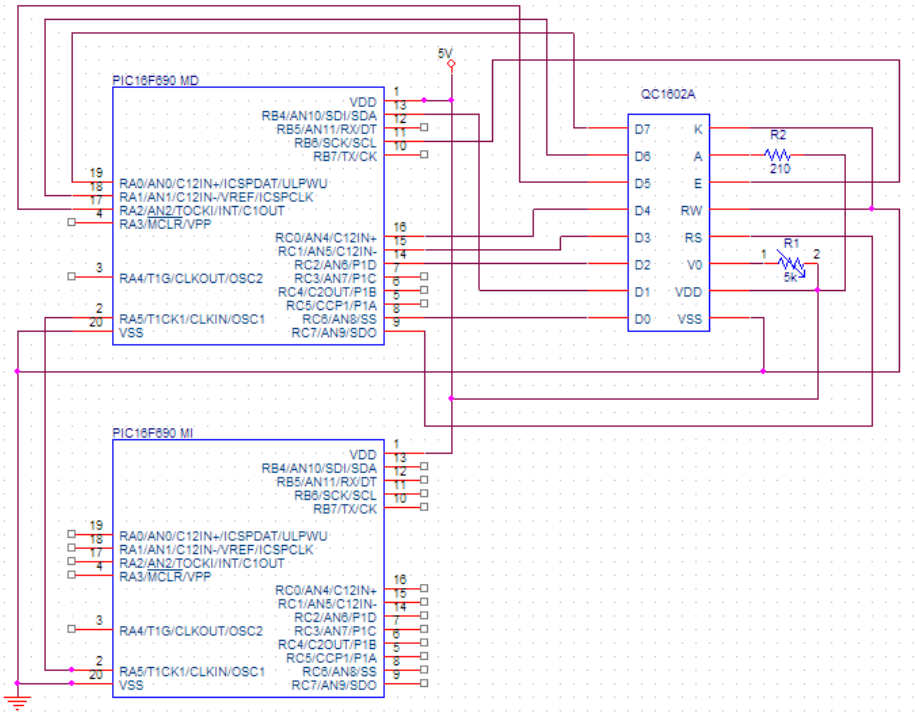

Figura 35. Conexiones entre microcontroladores y pantalla LCD.

Uno de los problemas que acarrea el que uno de los microcontroladores esté a cargo del LCD y el otro no, es que la escritura en el LCD retrasa al microcontrolador derecho según la longitud del mensaje que se imprima por pantalla. Si bien este retraso solo es de entre

100ms y 200ms, es lo suficiente como para que alterar el comportamiento de los motores de manera no deseada. Ya que el microcontrolador izquierdo se adelantaba al derecho, el motor izquierdo empezaba a funcionar antes que el derecho. Para evitar esto al microcontrolador izquierdo se le obliga a esperar una señal antes de poder salir de la función interrupción, esta señal la enviará el microcontrolador derecho en el momento que haya terminado de escribir el mensaje en la pantalla LCD. El canal de sincronización se puede observar en la figura 35 como sale del puerto **RA5** de salida en el microcontrolador derecho y llegado al puerto de entrada **RA5** del microcontrolador izquierdo. De esta manera se logra sincronizar el comportamiento de los dos microcontroladores.

10.5. Lectura del estado de baterías

El módulo ADC es el encargado de leer el nivel de tensión en la alimentación de elevación de los motores, pero como ya se explicó en el apartado 8.3 para poder hacer la comparación es necesario tener una tensión de referencia. Por lo tanto se utilizará la tensión de alimentación de los microcontroladores de 5V como referencia y la tensión de elevación se medirá. Por la entrada analógica **AN3** del pin 3 del microcontrolador derecho entrará el voltaje de la alimentación de elevación pero puesto que la tensión de referencia es de 5V la tensión que se quiere medir no puede ser mayor a esta. Es por este motivo que se utilizará un divisor de tensión para reducir el voltaje de alimentación a la mitad y así poder medir sus variaciones con el módulo ADC. Un divisor de tensión es una configuración de circuito eléctrico que reparte la tensión de una fuente, se puede ver la figura 36 y la ecuación 4 que calcula el valor de la tensión de salida. **[23]**

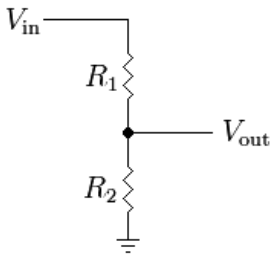

Figura 36. Divisor de tensión.

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

Ecuación 4. Cálculo de tensión de salida en divisor de tensión.

Finalmente para indicar con que voltaje se ha dejar de indicar que una pila está cargada y pase a estar semidescargada y lo mismo para el paso semidescargada a descargada. Se ha usado el grafico de la figura 37, en el que se puede ver la descarga de una pila con respecto al tiempo, considerando que se usan pilas de la marca Energizer: Una pila cargada estará entre 1.6V-1.4V, una pila semidescargada entre 1.4V y 1.2V y una descargada de uno 1.2V-0V.

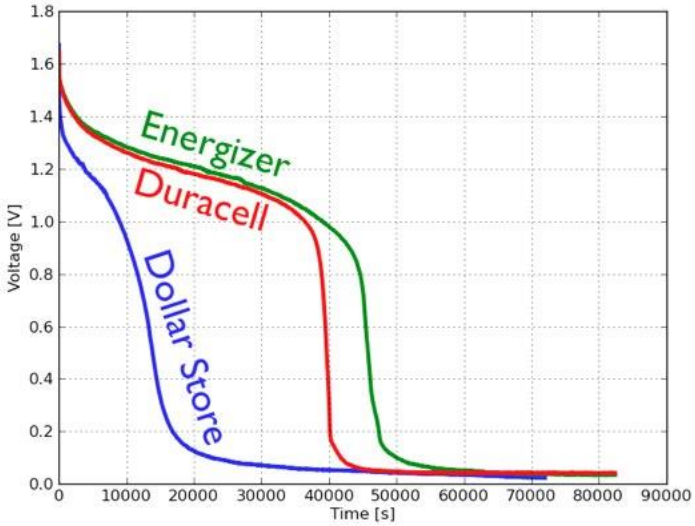

Figura 37. Descarga de una pila AA.

Para terminar se puede ver en la figura 38 la configuración necesaria para conectar la alimentación de elevación a la entrada analógica **AN3**.

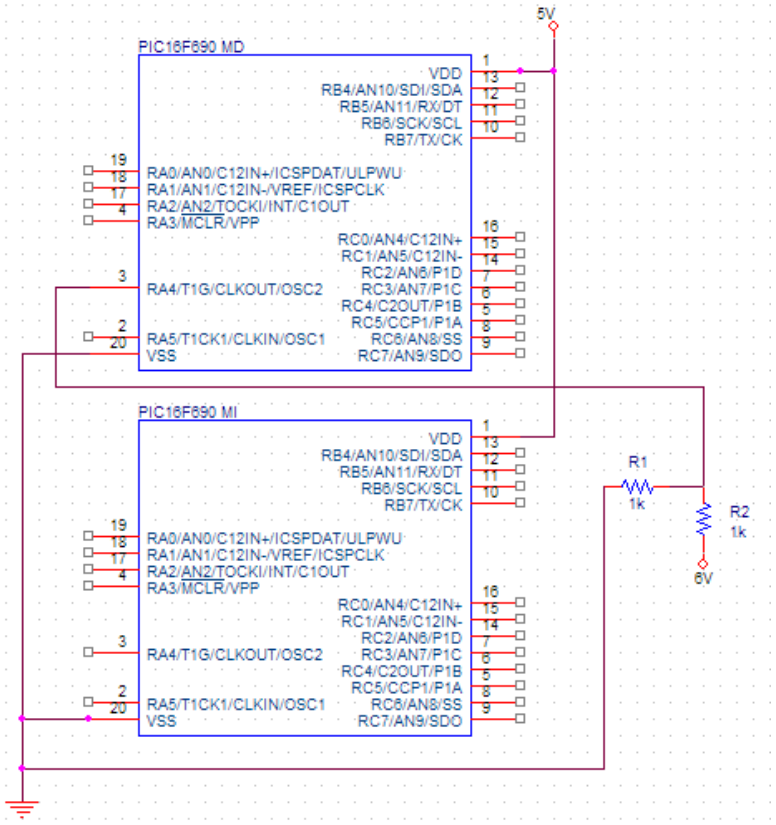

Figura 38. Circuito del divisor de tensión a los microcontroladores.

10.6. Detección de obstáculos

El microcontrolador izquierdo es el encargado de la detección de obstáculos, mediante el puerto **RA2** se enviará el pulso de 10µs al pin **Trig** que activará el sensor de ultrasonidos. El sensor enviará 8 señales de 40kHz y esperará a captar su eco, si se capta un eco será por el pin **Echo** del sensor que se envié la respuesta, este pin está conectado al puerto **RC4** encargado de captar la señal. En la figura 39 se puede ver lo descrito anteriormente. Internamente el microcontrolador medirá el ancho del pulso del eco y calculará con esto la distancia a la que se encuentra el obstáculo, si es menor a 30cm se producirá una parada automática en el coche eléctrico.

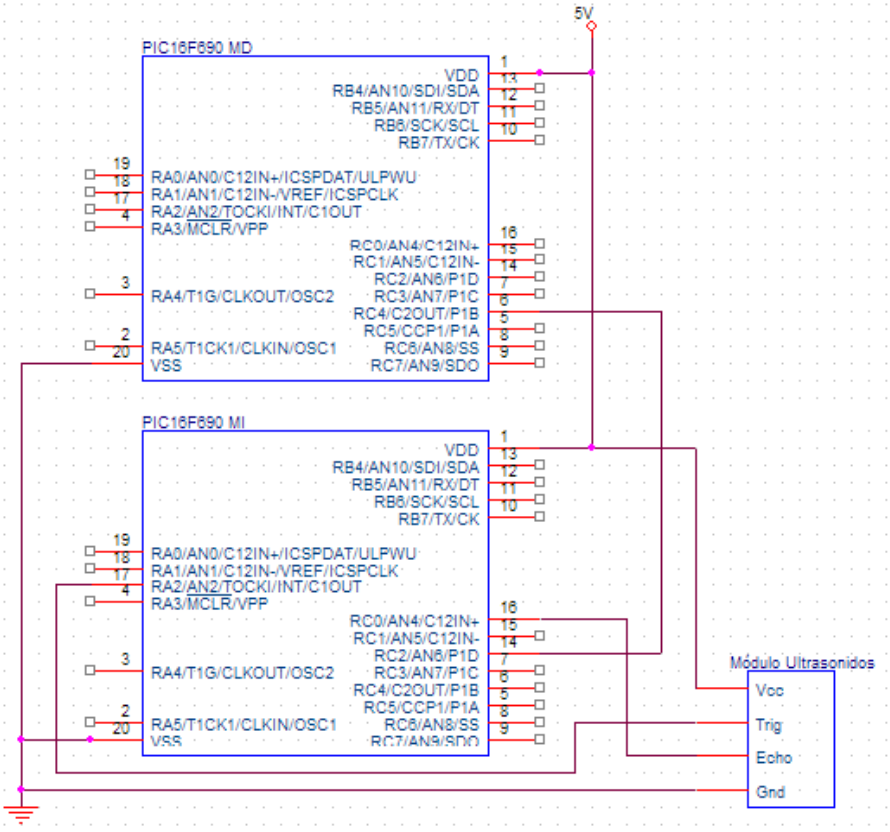

Figura 39. Circuito del sensor de ultrasonidos a los microcontroladores.

Puesto que es el microcontrolador izquierdo que en caso de que detecte un obstáculo es necesario transmitirlo al microcontrolador derecho. Para hacer esto se utiliza otro canal de comunicación entre microcontroladores en este caso saldrá del puerto **RC2** del microcontrolador izquierdo y llegada al puerto **RC4** del microcontrolador derecho. Cuando se produce una detección el microcontrolador izquierdo se lo comunica al microcontrolador derecho. El microcontrolador derecho escribe por pantalla un mensaje indicando que se ha encontrado un obstáculo, que dirá sencillamente “Obstaculo”. Además para que no se

desincronicen los microcontroladores se utilizará el canal de sincronización para que el microcontrolador izquierdo espere al derecho mientras escribe el mensaje por el LDC. Este mensaje será el único que se escriba fuera de la función interrupción.

La detección de obstáculos es un proceso repetitivo que se ha de realizar de manera constante ya que el entorno del coche cambia continuamente. Debido a esto su lugar ideal es dentro del bucle infinito de la función principal que trabaja a intervalos de 400ms.

Uno de los problemas que presentaba el detector de obstáculos es que, tras haberse producido una parada provocada por la detección de un obstáculo el propio detector impedía al usuario continuar operando con el coche eléctrico. Puesto que al enviar cualquier tipo de orden el detector, al seguir encontrándose a una distancia menor a 30cm del obstáculo, cancelaba cualquier tipo de orden enviada por el usuario. Por este motivo en la interfaz gráfica de la aplicación móvil se ha incluido el botón **Sensor** encargado de activar y desactivar el detector de obstáculos. Cuando se pulsa este botón lo que se está haciendo es alterar el valor de la variable de estado, **Estado de Sensor**. En consecuencia tras producirse una parada por detección de obstáculo el detector se desactivará automáticamente, evitando de esta manera que interfiera con las posteriores órdenes del usuario. Será el usuario quien a voluntad mediante el botón **Sensor** decidirá cuándo volver a activar el detector de obstáculos.

10.7. Alimentación del coche eléctrico

Se utilizarán 6 pilas doble AA conectadas en serie, de esta manera se conseguirá una tensión de 9V. En la figura 40 se puede ver como esta tensión de 9V será la tensión de elevación que se utilizará en los motores, por lo que se conectará directamente la entrada **VIN** de la placa del driver L293D. También será la que se conecte al divisor de tensión y que termine en la entrada analógica del microcontrolador derecho **AN3** para medir su nivel de tensión.

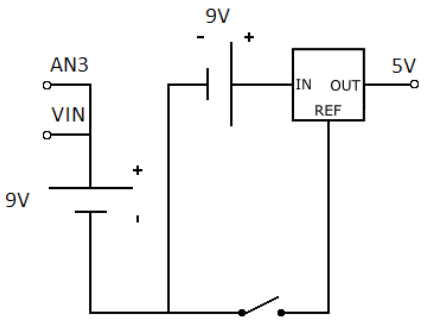

Figura 40. Alimentación utilizada en el vehículo.

Tanto los microcontroladores, la pantalla LCD, el módulo *bluetooth*, el sensor de ultrasonidos necesitan una tensión de 5V para funcionar es por esto que se utilizará un regulador de tensión para reducir la tensión de una pila de 9V a 5V. Conectando el positivo de la pila a la entrada **IN** del regulador y por la salida **OUT** saldrán los 5V que alimentarán a toda la lógica del coche.

El motivo por el que no se utiliza únicamente una fuente de alimentación de 9V para todo el conjunto en vez de tener una pila de 9V y 6 pilas de 1,6V, se debe a que como se explicó antes al aumentar las velocidades también se está en el consumo de tensión por los motores. Cuando se aumenta la tensión por los motores a veces crea problemas con la alimentación del resto del sistema, es decir al regulador de tensión no llega voltaje suficiente y su salida el voltaje es menor a 5V. Un voltaje menor a 5V en la alimentación de los microcontroladores provoca diversos problemas, la captación las ordenes enviadas, la pantalla LCD no tiene luz suficiente. Es por esto que se optó por mantener las fuentes de alimentación independientes entre sí, para asegurarse siempre que tanto los microcontroladores, la pantalla LDC, el módulo *bluetooth* y el sensor de ultrasonidos estén alimentados adecuadamente.

Finalmente para poder apagar y encender el coche se ha instalado un interruptor entre el negativo de las pilas y la tierra del resto del coche eléctrico.

11. Estructura del coche

En este capítulo se hace una descripción de los elementos que componen el coche, de su funcionamiento y de posibles problemas que pueden dar.

11.1. Composición física del coche

La estructura del coche que se utilizará es la de un bastidor de plástico donde se atornillaran el motor izquierdo y motor derecho, además en la punta contará con una rueda giratoria también atornillada. La figura 42 es una representación del coche en 2D con una vista en planta, las dos ruedas traseras estarán siempre fijas en la misma dirección y la rueda delantera giratoria puede adoptar cualquier dirección. Sobre el bastidor es donde se apoyaran el porta pilas y la placa de baquelita con los componentes soldados en ella, como se puede ver en la imagen 41.

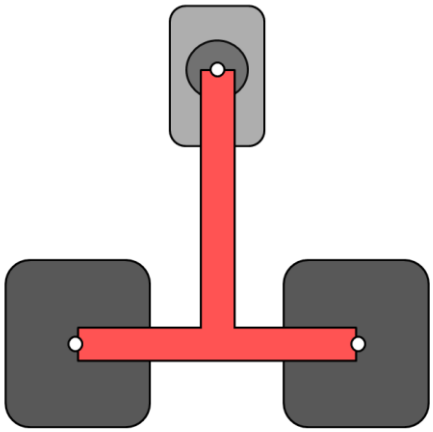

Figura 42. Representación 2D simplificada del coche eléctrico.

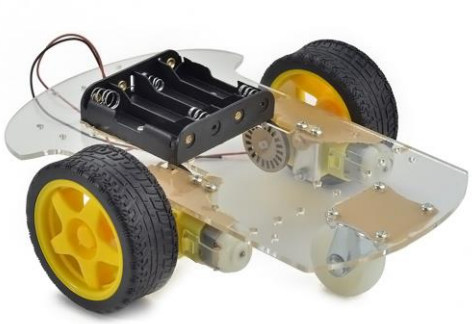

Figura 41. Ruedas y bastidor utilizados.

11.2. Problema de desplazamiento en trayectorias rectas

Uno de los primeros problemas que se observó desde el principio es que cuando se indicaba al coche eléctrico para que se desplazara hacia delante o atrás en línea recta, en la mayoría de los casos se producía un giro a derecha o izquierda. Algunas veces giraba durante unos segundos para después enderezarse, en otros casos nunca dejaba de girar y a veces de

desplazaba en línea recta pero al final se terminaba produciendo un giro. En definitiva los desplazamientos en línea recta, tanto adelante como hacia atrás eran completamente impredecibles. Los giros en cambio se producían sin ningún tipo de problema.

La primera opción que se exploró fue la de considerar que los motores utilizados estaban sujetos a una gran variabilidad y a pesar de recibir la misma tensión estarían retornando velocidades distintas. Si los motores tuvieran velocidades distintas esto provocaría que se produjese un giro en consecuencia.

Por otro lado también se estudió que el problema viniera del uso de una rueda giratoria en la parte delantera del coche. Idealmente la rueda giratoria se ha de adaptar a la trayectoria que las ruedas motores le indiquen. Si las dos ruedas traseras tienen la misma velocidad la delantera tiene que adoptar una dirección paralela a estas y si una de las ruedas tiene mayor velocidad que la otra la rueda delantera tiene que girar.

La mejor manera de explicar esto es mediante el estudio de la posición del CIR del coche. El centro instantáneo de rotación se define como el punto del solido o de su prolongación en el que la velocidad instantánea del cuerpo es nula. Si el cuerpo realiza una rotación pura alrededor de un punto, dicho punto es el CIR. El CIR se obtiene como la intersección de las normales a las velocidades de dos puntos cualesquiera de un sólido plano. **[24]**

En la figura 44 se puede ver como con la rueda giratoria inclinada con respecto a las ruedas traseras el CIR se encontrará sobre una proyección del eje de las ruedas traseras. En cambio en la figura 43 cuando lo que se desea es que el coche no gire, sino que se traslade el CIR se encontrará en el infinito.

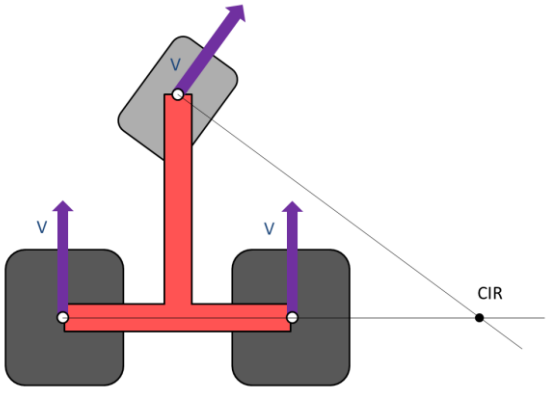

Figura 44. Posición del CIR en un giro.

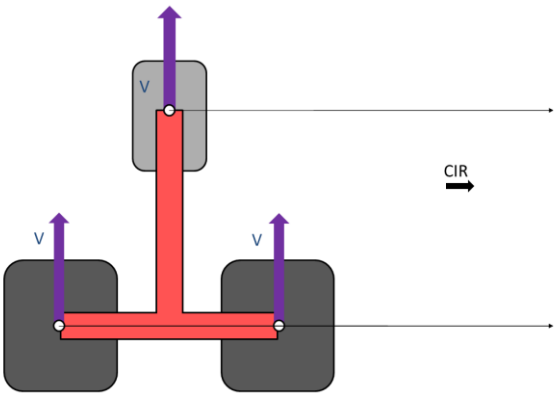

Figura 43. Posición del CIR en un desplazamiento recto.

Idealmente para que se produzca lo anteriormente descrito y que la rueda giratoria se encuentre en paralelo con las traseras es muy complicado, la más leve desviación en la rueda delantera producirá un giro.

Otro de los aspectos contemplados fue la posibilidad de que las ruedas traseras no estuvieran perfectamente alineadas y paralelas entre sí, lo cual también tendría efectos contraproducentes en la trayectoria.

Para encontrar la causa del problema se decidió hacer una comprobación empírica. La rueda con eje giratorio se cambió por una rueda con un eje fijo paralelo a las ruedas traseras y se dio la misma tensión a los dos motores. Así si la causa de los giros no deseados es la diferencia de velocidad entre motores los giros se continuarían produciendo por otra parte si la causa de los giros es la rueda giratoria tendría que desplazarse en línea recta.

Finalmente se observó como el coche se desplazaba prácticamente en línea recta, aunque sí que se producían muy pequeñas desviaciones. La conclusión fue muy clara: los motores si bien retornaban velocidades distintas no lo suficiente como para alterar la trayectoria de manera tan importante, el problema venia dado por el uso de una rueda giratoria.

Para solucionar a este problema se valoraron diferentes alternativas, la primera de ellas era la de cambiar la rueda giratoria por dos ruedas motoras más. El problema de esta solución es que al girar se produciría mucho deslizamiento. Otra de las posibles opciones era de controlar con otro motor el giro de la rueda delantera pero no se contaba con el material. Al final se optó por agrega el deslizador de **Compensación** para que el usuario corrija manualmente la trayectoria.

12. Presupuesto económico

Para realizar el presupuesto se han considerado tres tipos de costes, los costes por los componentes electrónicos utilizados para la fabricación del vehículo, los costes de las herramientas que se utilizan para la programación de los microcontroladores y manufactura del vehículo y por último el coste asociado a la mano de obra del proyecto.

Se distinguirán dos tipos de mano de obra, la del ingeniero junior que cobrará unos 15€/h y la del ingeniero que cobrará 50€/h, cada uno tendrá asociadas tareas diferentes con tiempos de trabajo distintos. El ingeniero junior se encargará de la mayor parte del trabajo mientras que el ingeniero lo supervisarla y guiará.

Componentes para la fabricación	Precio/unidad [€/u]	Cantidad [u]	Coste [€]
Placa de baquelita	1,52	1	1,52
Estaño	120euros/Kg	30g	3,6
Cable	0,1 euro/m	0,5m	0,05
Zócalos	0,22	3	0,66
Microprocesador PIC16F690	1,95	2	3,9
Sensor ultrasonidos	2,1	1	2,1
Pantalla LCD	4,75	1	4,75
Modulo *bluetooth*	6,1	1	6,1
Placa con L293D	2,3	1	2,3
Resistencia de 1kΩ	0,02	2	0,04
Resistencia de 220Ω	0,02	1	0,02
Interruptor	0,1	1	0,1
Reductor de tensión	0,75	1	0,75
Portapilas x6	1,6	1	1,6
Bastidor + 2 Motores DC + Rueda giratoria	13,5	1	13,5
Pila AA Energizer	2,05	6	12,3
	Subtotal Componentes Fabricación		**53,29**

Tabla 6. Elementos necesarios para la fabricación del coche eléctrico.

En la tabla 6 se observan todos los elementos necesarios para la construcción del vehículo con sus cantidades y correspondiente coste. Se ha de tener en cuenta que como se utilizan componentes electrónicos que únicamente se venden en paquetes de ciertas cantidades para saber el precio unidad se ha hecho una estimación.

Componentes para la programación y manufactura	Precio/unidad [€/u]	Cantidad [u]	Coste [€]	Vida útil	Tiempo amortización	Coste Amortización [€]
Ordenador	1100	1	1100	8 años	4 meses	45,83
KitPIC2	35	1	35	5 años	4 meses	2,33
Soldador	100	1	100	10 años	4 meses	3,33
Placa protoboard	45	1	45	5 años	4 meses	2,97
MPLABX	0	1	0	1 año	4 meses	0
HI-TECH C	0	1	0	1 año	4 meses	0
			Subtotal Componentes Programación			**54,46**

Tabla 7. Herramientas utilizadas para la programación y manufactura del coche eléctrico.

En la tabla 7 se tienen el precio por unidad de los herramientas utilizadas para la programación y manufactura. El software que se ha empleado, tanto el entorno de desarrollo como el compilador son gratuitos por lo que no acarrearan ningún tipo de coste. Se considerado que el tiempo de duración del proyecto son 4 meses, con esto y una estimación de la vida útil se puede calcular el coste de amortización de cada herramienta.

Ingeniero Junior	Tiempo [h]	Coste [€]
Búsqueda bibliográfica y lectura	40	600
Diseño del sistema de control	100	1500
Comprobación del funcionamiento	80	1200
Correcciones sobre el diseño	30	450
Redacción de la memoria	50	750
Subtotal Ingeniero Junior	300	**4500**

Tabla 8. Mano de obra Ingeniero Junior.

El ingeniero junior será el que más trabajo tenga que hacer, como se indica en la tabla 8 primero tendrá que llevar a cabo una búsqueda bibliográfica y lectura para adquirir los conocimientos necesarios para efectuar el diseño. Una vez ha diseñado el sistema tendrá que proseguir que asegurarse que funcione correctamente y aplicar las correcciones convenientes sobre él. Finalmente habrá de redactar todo esta experiencia en un detallado informe.

Ingeniero	Tiempo [h]	Coste [€]
Pruebas de material	10	500
Búsqueda de información	5	250
Consultas	20	1000
Subtotal Ingeniero	35	**1750**

Tabla 9. Mano de obra Ingeniero.

Por otro lado el Ingeniero se involucrará únicamente en tareas de supervisión, consulta o guía. Se cerciorará que el material que utilice el Ingeniero junior sea el adecuado y no se estropee, le dará la guía necesaria en aquellos temas que no tenga experiencia y le ayudará con cualquier tipo de pregunta o problema que tenga. En la tabla 9 se pueden apreciar todas estas tareas.

	Coste [€]
Subtotal Componentes Fabricación	53,29
Subtotal Herramientas de Programación y Manufactura	54,46
Subtotal Ingeniero Junior	4500
Subtotal Ingeniero	1750
Coste Total	**6357,75**

Tabla 10. Coste total del proyecto.

Finalmente en la tabla 10 se encuentra el coste total del proyecto que es la suma de todos los costes anteriormente descritos. El coste de los componentes de fabricación y las herramientas de programación y manufactura es muy bajo en comparación la mano de obra utilizada. El coste total del proyecto será de 6357,75 €.

13. Impacto medioambiental

Todos los componentes utilizados en este trabajo siguen la directiva RoHS, Restriction of Hazardous Substances, fue adoptada por la Unión Europea en Febrero de 2003. Se encarga de regular las sustancias peligrosas en aparatos eléctricos y electrónicos, estas sustancias son el plomo, mercurio, cadmio, cromo VI y los retardantes para plásticos PBB y PBDE.

Una vez que se haya llegado al final de la vida útil del vehículo y se desee desecharlo, se tendrá que realizar el desmantelamiento en un sitio especializado. Esto se hará acorde con la directiva WEEE, *Waste Electrical and Electronic Equipment*, que entro en vigor en agosto de 2015. Esta directiva promueve el reciclaje, reutilización y recuperación de los residuos eléctrico y electrónicos, su objetivo es el de resolver el problema de las grandes cantidades de chatarra electrónica. **[25]**

Conclusiones

Si se miran los objetivos planteados para este proyecto que eran el de diseñar el sistema electrónico basado en microcontrolador para el control de un vehículo se puede concluir que se ha cumplido de manera satisfactoria. Se ha logrado un completo control direccional y de velocidad, visualizando esto por una pantalla LCD de manera escrita la dirección y velocidad del momento. Además se ha dotado de un sensor de ultrasonidos en el frente del vehículo para provocar una detención automática en caso de que se encuentre un obstáculo cerca. Todo se controla mediante un dispositivo *Android* y la emisión de una señal captada por un módulo *bluetooth* incorporado al vehículo.

Uno de los fenómenos observados es que a medida que se iban incrementando la cantidad de subsistemas dentro del microcontrolador surgen nuevas complicaciones. La convivencia entre subsistemas generaba inconvenientes que requerían de la implementación de nuevas soluciones tanto en el funcionamiento interno del microcontrolador como sobre el dispositivo *Android*.

El proceso programación de microcontroladores si bien en un principio puede resultar complejo y confuso, una vez que se han aprendido las bases de su funcionamiento su programación y configuración siempre siguen el mismo proceso facilitando de manera notable su uso.

Como se había comentado en el apartado 11.2 la estructura escogida no era la más óptima para realizar desplazamientos rectos por eso como recomendación se propone la sustitución de la rueda delantera por otra rueda que se pueda controlar su giro.

Agradecimientos

Me gustaría agradecer a mi tutora Rosa por toda la paciencia que ha tenido conmigo, siempre dispuesta a ayudarme con cualquier tipo de duda que me surgiera o cuando por mi inexperiencia quemase algún dispositivo accidentalmente. Además también a mi hermano Jordán por ayudarme siempre que lo necesitaba en la programación en C y darme su opinión siempre que necesitase otro punto de vista.

Bibliografía

Referencias bibliográficas

[1] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Microcontrolador*, [en línea], http://es.wikipedia.org/wiki/Microcontrolador [Febrero 2015]

[2] MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet,* TABLE 5: PIC16F690 Pin Summary, [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf, [Febrero 2015]

[3] MICROCHIP, *PICkit 2 Programmer/Debugger User's Guide,* Chapter 1. PICkit 2 Programmer/Debugger Overview, [en línea], http://ww1.microchip.com/downloads/en/DeviceDoc/51553E.pdf, [Febrero 2015]

[4] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Placa de pruebas*, [en línea], http://es.wikipedia.org/wiki/Placa_de_pruebas, [Febrero 2015]

[5] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Cable,* [en línea], https://es.wikipedia.org/wiki/Cable, [Febrero 2015]

[6] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Resistor*, [en línea], https://es.wikipedia.org/wiki/Resistor, [Febrero 2015]

[7] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Reostato*, [en línea], http://es.wikipedia.org/wiki/Reostato. FOROS DE ELECTRONICA, *El Resistor,* [en línea], http://www.forosdeelectronica.com/tutoriales/resistencia.htm, [Febrero 2015]

[8] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Condensador eléctrico*, [en línea], https://es.wikipedia.org/wiki/Condensador_el%C3%A9ctrico. AREA TECNOLOGIA, *¿Cómo almacena la Carga el Condensador?,* [en línea] , http://www.areatecnologia.com/electricidad/condensador.html, [Febrero 2015].

[9] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Interruptor* , [en línea], http://es.wikipedia.org/wiki/Interruptor, [Marzo 2015]

[10] TEXAS INSTRUMENT, *Description/ordering information*, [en línea], https://www.sparkfun.com/datasheets/Components/LM7805.pdf , [Marzo 2015]

[11] DATASHEET QC1602A, *Dimensión outline* [en línea], https://www.mpja.com/download/qc1602a.pdf. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Pantalla de cristal líquido,* [en línea], http://es.wikipedia.org/wiki/Pantalla_de_cristal_l%C3%ADquido, [Marzo 2015]

[12] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Motor de corriente continua* , [en línea] https://es.wikipedia.org/wiki/Motor_de_corriente_continua. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Brushed DC motor*, [en línea] , https://en.wikipedia.org/wiki/Brushed_DC_electric_motor, [Marzo 2015]

[13] TEXAS INSTRUMENT, *L293, L293D Quadruple half-h drivers*, Description/ordering information, [en línea] http://www.ti.com/lit/ds/symlink/l293.pdf. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Flyback diode*, [en línea],https://en.wikipedia.org/wiki/Flyback_diode, [Abril 2015]

[14] ELECFREAKS, *Datasheet Ultrasonic Ranging Module HC - SR04* [en línea], http://www.micropik.com/PDF/HCSR04.pdf, [Abril 2015]

[15] MICROCHIP, *MPLAB® X Integrated Development Environment* , [en línea], http://www.microchip.com/pagehandler/en-us/family/mplabx/. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *MPLAB*, [en línea], http://es.wikipedia.org/wiki/MPLAB, [Abril 2015]

[16] MICROCHIP, *HI-TECH for PIC10/12/16 User's Guide* [en línea], http://ww1.microchip.com/downloads/en/DeviceDoc/HTC_PIC_manual.pdf. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Compilador,* [en línea], http://es.wikipedia.org/wiki/Compilador, [Abril 2015]

[17] JULYAN ILET*, How to use intelligent LCD's*, Part One, [en línea], http://www.datsi.fi.upm.es/docencia/Micro_C/lcd/HowtouseLCD1.pdf. JULYAN ILET*, How to use intelligent LCD's*, Part two, [en línea] http://www.datsi.fi.upm.es/docencia/Micro_C/lcd/HowtouseLCD2.pdf, [Abril 2015]

[18] MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 11.0 Enhanced Capture/Compare/PWM module, [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf, [Junio 2015]

[19] MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 9.0 Analog-to-digital Converter(ADC) Module, [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf, [Junio 2015]

[20] MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 12.0 Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART), [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf, [Junio 2015]

[21] MIT, *App Inventor*, [en línea], http://appinventor.mit.edu/explore/about-us.html [Junio 2015]

[22] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Dirección MAC*, [en línea], https://es.wikipedia.org/wiki/Direcci%C3%B3n_MAC, [Junio 2015]

[23] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Divisor de tensión*, [en línea], https://es.wikipedia.org/wiki/Divisor_de_tensi%C3%B3n, [Junio 2015]

[24] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Centro y eje instantáneo de rotación*, [en línea], https://es.wikipedia.org/wiki/Centro_y_eje_instant%C3%A1neo_de_rotaci%C3%B3n [Junio 2015]

[25] WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Restriction of Hazardous Substances*, [en línea], https://es.wikipedia.org/wiki/RoHS. WIKIPEDIA: THE FREE ENCYCLOPEDIA, *Waste Electrical and Electronic Equipment Directive*, [en línea], https://en.wikipedia.org/wiki/Waste_Electrical_and_Electronic_Equipment_Directive, [Junio 2015]

Bibliografía de las figuras

Figura 1. ELECTROSOME, [en línea], https://electrosome.com/getting-started-pic-ccs-c/ .

Figura 2. MICROCHIP, *PIC12F508/509/16F505 Data Sheet,* Pin Diagrams, [en línea], http://www.farnell.com/datasheets/720391.pdf.

Figura 3. MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet,* Pin Diagrams**,** [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf.

Figura 4. MICROCHIP, *PICkit 2 Programmer/Debugger User's Guide,* PICkit 2 PROGRAMMER CONNECTOR PINOUT, [en línea], http://ww1.microchip.com/downloads/en/DeviceDoc/51553E.pdf.

Figura 5. VOTI, [en línea], http://www.voti.nl/pickit2faq/index_1.html.

Figura 6. PALACIOS M., [Fuente propia].

Figura 7. PIDELECTRONICS, [en línea], http://pidelectronics.com/resistencia-22-kohm-1-4watts.html.

Figura 8. ALIEXPRESS, [en línea], http://es.aliexpress.com/c-variable-resistor-100k.html.

Figura 9. TECNOSTORE, [en línea], http://www.tiendaelectronica.com.ve/condensadores/15-condensador-electrolitico-1uf-50v.html.

Figura 10. REPARATUOCIO, [en línea], http://www.reparatuocio.com/regulador-de-tension-9v-1a-t0220-utc-7809-p-1281.html.

Figura 11. ABC-RC, [en línea], http://abc-rc.pl/QC1602A.

Figura 12. ABC-RC, [en línea], http://abc-rc.pl/QC1602A.

Figura 13. MERCADOLIBRE, [en línea], http://articulo.mercadolibre.com.mx/MLM-495499632-modulo-bluetooth-para-arduino-pic-arduino-raspberry-robotica-_JM.

Figura 15. MERCADOLIBRE, [en línea], http://produto.mercadolivre.com.br/MLB-663962595-2-pcs-x-motor-dc-3-6v-caixa-de-reduco-roda-arduino-_JM#redirectedFromParent.

Figura 16. UJUNGSOLDER, [en línea], http://www.ujungsolder.com/motor-drive-expansion-board-l293d.html.

Figura 18. ANDTECHNOLOGIES, [en línea], https://andtechnologies.in/ultrasonic-ranging-module-hc-sr04-hc-sr04.html.

Figura 19. WIKI ITEAD, [en línea], http://wiki.iteadstudio.com/Ultrasonic_Ranging_Module_HC-SR04.

Figura 20. MICROCHIP, [en línea], https://microchip.wikidot.com/mplabx:stopwatch.

Figura 21. PALACIOS M., OrCAD, [Fuente propia].

Figura 22. JULYAN ILET, *How to use intelligent LCD's*, Part One, [en línea], http://www.datsi.fi.upm.es/docencia/Micro_C/lcd/HowtouseLCD1.pdf.

Figura 23. MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 11.0 Enhanced Capture/Compare/PWM module, CCP PWM OUTPUT , [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf.

Figura 24. MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 9.0 Analog-to-digital Converter(ADC) Module, ADC BLOCK DIAGRAM, [en línea], http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf.

Figura 25. MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 9.0 Analog-to-digital Converter(ADC) Module, 10-BIT A/D CONVERSION RESULT FORMAT, [en línea], http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf.

Figura 26. MICROCHIP, *PIC16F631/677/685/687/689/690 Datasheet*, 12.0 Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART), EUSART RECEIVE BLOCK DIAGRAM, [en línea] http://ww1.microchip.com/downloads/en/DeviceDoc/41262c.pdf.

Figura 27. PALACIOS M., Impresión de pantalla, [Fuente propia].

Figura 28. PALACIOS M., App Inventor, [Fuente propia].

Figura 29. PALACIOS M., App Inventor, [Fuente propia].

Figura 30. PALACIOS M., App Inventor, [Fuente propia].

Figura 31. PALACIOS M., App Inventor, [Fuente propia].

Figura 32. PALACIOS M., Power Point, [Fuente propia].

Figura 33. PALACIOS M., OrCAD, [Fuente propia].

Figura 34. PALACIOS M., OrCAD, [Fuente propia].

Figura 35. PALACIOS M., OrCAD, [Fuente propia].

Figura 36. PALACIOS M., [Fuente propia].

Figura 37. WIRED, *ARE EXPENSIVE BATTERIES WORTH THE EXTRA COST?*,[en línea,] http://www.wired.com/2012/01/are-expensive-batteries-worth-the-extra-cost/.

Figura 38. PALACIOS M., OrCAD, [Fuente propia].

Figura 39. PALACIOS M., OrCAD, [Fuente propia].

Figura 40. PALACIOS M., [Fuente propia].

Figura 41. DEALEXTREME, [en línea],http://www.dx.com/p/smart-robot-car-chassis-kit-for-arduino-transparent-yellow-152984#.VZSPfvmqqko.

Figura 42. PALACIOS M., Power Point, [Fuente propia].

Figura 43. PALACIOS M., Power Point, [Fuente propia].

Figura 44. PALACIOS M., Power Point, [Fuente propia].

Anexos (volumen separado)

I. Tabla de comandos de control

Table 2. The Command Control Codes.

Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move

R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*

x = Don't care * = Initialisation settings

II. Tabla de caracteres estándares LCD

Table 3. Standard l.c.d. character table.

III. Ejemplo código programación LCD

```
#include <htc.h>
#include <stdint.h>
__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO

#define _XTAL_FREQ  4000000
#define ENABLE RB6
#define MODE RC7

void rdatasender()
{
    RC6 = RDATA.RC6;
    RB4 = RDATA.RB4;
    RC2 = RDATA.RC2;
    RC1 = RDATA.RC1;
    RC0 = RDATA.RC0;
    RA2 = RDATA.RA2;
    RA1 = RDATA.RA1;
    RA0 = RDATA.RA0;
}

void trisdsender()
{
    TRISC6 = TRISD.TRISC6;
    TRISB4 = TRISD.TRISB4;
    TRISC2 = TRISD.TRISC2;
    TRISC1 = TRISD.TRISC1;
    TRISC0 = TRISD.TRISC0;
    TRISA2 = TRISD.TRISA2;
    TRISA1 = TRISD.TRISA1;
    TRISA0 = TRISD.TRISA0;
}


void enable_up_low() //SEÑAL HIGH TO LOW
{
    ENABLE = 1;
    __delay_ms(15);
    ENABLE = 0;
    __delay_ms(15);
}

void printToLCD(char* text)
{
    MODE = 1; //MODO ESCRITURA

    unsigned int i = 0;
    while(text[i] != '\0')
    {
        RDATA.port = text[i];
        rdatasender();
        enable_up_low();
```

```
        ++i;
    }
}

void EnviarMensajeLCD()
{

    ANSEL=0; //PINS EN MODO DIGITAL
    ANSELH=0;

    TRISC7 = 0; // HABILITAMOS LAS SALIDAS DEL MODE Y ENABLE
    TRISB6 = 0;

    TRISD.port=0; // HABILITAMOS LAS SALIDAS DE LOS PUERTOS DATA
    trisdsender();

    MODE = 0; // MODO COMANDOS

    RDATA.port = 0b00111000 ;//ACTIVAR SEGUNDA LINEA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00001100 ;//CONFIGURAR CURSOR
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00000001 ; //CLEAR DISPLAY
    rdatasender();
    enable_up_low();

    MODE = 1; //MODO ESCRITURA

    printToLCD("HOLA MUNDO")

    for(;;);
    }
```

IV. Ejemplo código carácter personalizado en LCD

```
#include <htc.h>
#include <stdint.h>

__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO


void escribirsimbolo()
{
    MODE = 0;

    RDATA.port = 0b01000000 ;//SET CGRAM
    rdatasender();
    enable_up_low();

    MODE=1;

    RDATA.port = 0b00001110;//PROGRAMAR CASILLA
    rdatasender();
```

```
    enable_up_low();

    RDATA.port = 0b00010001;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00001110 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00000100 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00011111 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00000100 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00001010 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    RDATA.port = 0b00010001 ;//PROGRAMAR CASILLA
    rdatasender();
    enable_up_low();

    MODE = 0;

    RDATA.port = 0b00000010 ; //DISPLAY HOME
    rdatasender();
    enable_up_low();

}
```

V. Ejemplo código señal PWM para control de motores

```
#include <htc.h>
#include <stdint.h>

__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO


void InicializarMotores()
{
    TRISC = 0;            //PUERTOS C COMO OUTPUT

    PR2  =    0b11111111; //255

    T2CON =   0b00000110; //TIMER2 ENCENDIDO Y PRE ESCALA 16

    CCP1CON = 0b01001100;
              // --          PUENTE H HACIA DELANTE
```

ETSEIB

```
    CCPR1L =  0b00000000;  // CICLO DE TRABAJO = 0
    DC1B0= 0;
    DC1B1 = 0;

    for(;;);
}
```

VI. Ejemplo inicialización de módulo ADC

```
#include <htc.h>
#include <stdint.h>
__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO

void InicializarADC()
{
    TRISA4=1; //PIN RA4 COMO INPUT
    ANS3 = 1; //PIN RA4 COMO ANALOGICO

    TRISA5=0;

    ADCON0 = 0b10001101;
          //   -          JUSTIFICADO A DERECHAS, ADFM = 1.
          //    -         REFERENCIA DE VOLTAJE DE ALIMENTACION VDD = 0.
          //     ----     SELECCION AN3
          //         -    GO/DONE = 0 NO HAY QUE EMPEZAR LA CONVERSION AUN
          //          -   ADCON = 1 MODULO ADC ACTIVADO
    ADCON1 = 0b01010000;
          //    ---       SELECCION DE CLOCK FOSC/16. RESTO DE BITS SIN
IMPLEMENTAR
}
```

VII. Ejemplo inicialización de módulo EUSART

```
#include <htc.h>
#include <stdint.h>
__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO

void InicializacionEUSART()
{
    ANSEL = 0; // TODOS PUERTOS I/O DIGITALES
    ANSELH = 0;

    TXSTA = 0b00000100;
    //          -           TXEN = 0 TRANSMIT DISABLE
    //           -          SYNC = 0 ASINCRONO
    //             -        BRGH = 1 VELOCIDAD ALTA


    SPBRG = 0b00011001;     //Numero 25
    BAUDCTL = 0b00000000;
    //              -       BRG16 = 0; MODO DE 8 BITS
```

```
    RCSTA = 0b10110000;
    //        -             SPEN = 1 SERIAL PORT ENABLE TX Y RX
    //         -            RX9 DESACTIVADO EL NOVENO BIT
    //           -          CREN = 1 ACTIVADO RECIBIDOR


    INTCON = 0b11000000;
    //         -            GIE = 1 PERMITIR TODAS LAS INTERRUPCIONES
    //          -           PEIE = 1 PERMITIR TODAS LAS INTERRUPCIONES
EXTERNAS
    PIE1 = 0b00100000;
    //         -            RCIE = 1 ACTIVAR LAS INTERRUPCIONES CUANDO
EUSART RECIBE

}
```

VIII. Ejemplo de código detección de obstáculos.

```
#include <htc.h>
#include <stdint.h>
__CONFIG(MCLRE_ON & CP_OFF & WDTE_OFF & FOSC_INTRCIO); // PROTECCIÓN DEL
CODIGO DESACTIVADA, WATCHDOG DESACTIVADO Y SELECCION DE RELOJ INTERNO

#define TRIGGER  RA2
#define ECHO  RC0

void DetectarObstaculo()
{
    TRISA2= 0; //TRIGGER COMO OUTPUT
    uint8_t Lectura;

    OBSTACULO = NODETECTADO;


    OPTION_REG = 0b11010010; //PROGRAMACIÓN DEL TEMPORIZADOR
                  // -       TIMER MODE
                    // -     PRESCALA EN EL TIMER
                      //--- ESCALA 8

    TRIGGER=1;
    __delay_us(10); //ENVIO DEL PULSO
    TRIGGER=0;

    while (ECHO == 0);// LOW TO HIGH
    T0IF=0;             // FLAG OVERFLOW A 0
    TMR0=0;             // TIMER0 A 0
    while (ECHO == 1);// HIGH TO LOW
    Lectura = TMR0;    // LECTURA DEL CRONÓMETOR

    if (Lectura < 218)// SI OBSTACULO A MENOS DE 30CM
    {
        if (T0IF==0) // SI NO SE PRODUCE OVERFLOW
        {
        OBSTACULO = DETECTADO;
        }
    }
}
}
```

IX. Código utilizado en el microcontrolador derecho

Archivo: MotorDerechoPrincial.c

```
#include <htc.h>
#include <stdint.h>
#include "MD.h"

__CONFIG(MCLRE_OFF & CP_OFF & WDTE_OFF & FOSC_INTRCIO & PWRTE_OFF &
IESO_OFF & FCMEN_OFF & BOREN_OFF );

void interrupt ISR()
{
    if (RCIF==1)
    {
        Lectura = RCREG; //RCIF = 0
        AVISO = ESPERAR;
        if (Lectura==BOTON_ADELANTE) // BOTON ADELANTE SELECCIONADO
        {
            SENTIDO = ADELANTE;
            DIRECCION = PRIORIDAD;
        }
        else if (Lectura==BOTON_ATRAS) // BOTON ATRAS SELECCIONADO
        {
            SENTIDO = ATRAS;
            DIRECCION = PRIORIDAD;
        }
        else if (Lectura==BOTON_DERECHA) // BOTON GIRO DERECHA
SELECCIONADO
        {
            DIRECCION = DERECHA;
        }
        else if (Lectura==BOTON_IZQUIERDA) // BOTON GIRO IZQUIERDA
SELECCIONADO
        {

            DIRECCION = IZQUIERDA;
        }
        else if (Lectura==BOTON_STOP) // BOTON STOP SELECCIONADO
        {
            SENTIDO = NOINTRODUCIDO;
            DIRECCION = NOINTRODUCIDO;
            VELOCIDAD = NOINTRODUCIDO;
        }
        else if (Lectura==BOTON_PRIMERA) // BOTON PRIMERA VELOCIDAD
SELECCIONADO
        {
            VELOCIDAD = PRIMERA;
        }
        else if (Lectura==BOTON_SEGUNDA) // BOTON SEGUNDA VELOCIDAD
SELECCIONADO
        {
```

```
            VELOCIDAD = SEGUNDA;

        }
        else if (Lectura==BOTON_TERCERA) // BOTON TERCERA VELOCIDAD
SELECCIONADO
        {
            VELOCIDAD = TERCERA;
        }
        else if (Lectura==BOTON_SENSOR) // BOTON TERCERA SENSOR
SELECCIONADO
        {
            ESTADOSENSOR = ACTIVADO;
        }
        else if (BuscarGiro(Lectura)) // SLIDER POSICION 1 SELECCIONADO
        {
            GRADOGIRO = Lectura;
        }
    }
    LeerEstadoBateria();

    MostrarDireccion();
    MostrarVelocidad();
    MostrarEstadoBateria();

    AVISO = NOESPERAR;
}


void main()
{
    __delay_ms(2000);

    AVISO = ESPERAR;

    Inicializacion();

    InicializarLCD();

    SimbolosEstadoBateria();

    IncializarMotores();

    InicializarNivelBateria();

    AVISO = NOESPERAR;

        for (;;)
    {
        __delay_ms(200);

        if (ESTADOSENSOR == DESACTIVADO)
        {
        __delay_ms(100);
        }
        else if (ESTADOSENSOR == ACTIVADO)
        {
        ObstaculoDetectado();
        }
        __delay_ms(100);
```

```
            if (DIRECCION == IZQUIERDA && (SENTIDO == ADELANTE || SENTIDO
== NOINTRODUCIDO) )
            {
               GirarIzq(ADELANTE);
            }
            else if (DIRECCION == IZQUIERDA && SENTIDO == ATRAS )
            {
               GirarIzq(ATRAS);
            }
            else if (DIRECCION == DERECHA && ( SENTIDO == ADELANTE ||
SENTIDO == NOINTRODUCIDO) )
            {
              GirarDer(ADELANTE);
            }
            else if (DIRECCION == DERECHA && SENTIDO == ATRAS )
            {
              GirarDer(ATRAS);
            }
            else if (SENTIDO == ADELANTE)
            {
               Avanzar(ADELANTE);
            }
            else if (SENTIDO == ATRAS)
            {
               Avanzar(ATRAS);
            }
            else if (SENTIDO == NOINTRODUCIDO && DIRECCION ==
NOINTRODUCIDO && VELOCIDAD == NOINTRODUCIDO)
            {
                Stop();
                if (MENSAJEOBSTACULO == NOENVIADO)
                {
                        clear_display();
                        printToLCD("Obstaculo\0");
                        AVISO = NOESPERAR;
                        MENSAJEOBSTACULO = ENVIADO;
                }
            }
    }
}
```

Archivo MD.c

```
#include "MD.h"
void Inicializacion()
{
    ANSEL = 0; // TODOS PUERTOS I/O DIGITALES
    ANSELH = 0;
    TRISA5 = 0;//AVISO COMO OUTPUT
    TRISB = 0b01111111;
    TXSTA = 0b00000100;
//              -         TXEN = 0 TRANSMIT DISABLE
//               -        SYNC = 0 ASINCRONO
//                 -      BRGH = 1 VELOCIDAD ALTA
    SPBRG = 0b00011001; //Numero 25
    BAUDCTL = 0b00000000;
    //              -         BRG16 = 0; MODO DE 8 BITS
    RCSTA = 0b10110000;
```

```
    //         -                SPEN = 1 SERIAL PORT ENABLE TX Y RX
    //          -               RX9 DESACTIVADO EL NOVENO BIT
    //            -             CREN = 1 ACTIVADO RECIBIDOR
    INTCON = 0b11000000;
    //          -            GIE = 1 PERMITIR TODAS LAS INTERRUPCIONES
    //           -           PEIE = 1 PERMITIR TODAS LAS INTERRUPCIONES
EXTERNAS
    PIE1 = 0b00100000;
    //          -              RCIE = 1 ACTIVAR LAS INTERRUPCIONES CUANDO
EUSART RECIBE
}
void IncializarMotores()
{
    PR2  =    0b11111111; //255
    TRISC5=0;
    TRISC3=0;
    T2CON =   0b00000110; //Timer 2 encendido y PREESCALA 16
    CCP1M0 = 0; //PWM MODO, todos valores alto en activo
    CCP1M1 = 0;
    CCP1M2 = 1;
    CCP1M3 = 1;
    //MODO SINGLE OUTPUT
    P1M1=0; P1M0 = 0; // A PWM, B , C Y D PUERTOS
}
void InicializarNivelBateria()
{
    TRISA4=1; //PIN RA4 COMO INPUT
    ANS3 = 1; //PIN RA4 COMO ANALOGICO
    TRISA5=0;
    ADCON0 = 0b10001101;
          //   -         JUSTIFICADO A DERECHAS, ADFM = 1.
          //    -        REFERENCIA DE VOLTAJE DE ALIMENTACION VDD = 0.
          //     ----    SELECCION AN3
          //         -   GO/DONE = 0 NO HAY QUE EMPEZAR LA CONVERSION AUN
          //          -  ADCON = 1 MODULO ADC ACTIVADO
    ADCON1 = 0b01010000;
          //    ---      SELECCION DE CLOCK FOSC/16. RESTO DE BITS SIN
IMPLEMENTAR
}
void InicializarLCD()
{
    ANSEL=0; //PINS EN MODO DIGITAL
    ANSELH=0;
    TRISC7 = 0; // HABILITAMOS LAS SALIDAS DEL MODE Y ENABLE
    TRISB6 = 0;
    TRISD.port=0; // HABILITAMOS LAS SALIDAS DE LOS PUERTOS DATA
    trisdsender();
    MODE = 0; // MODO COMANDOS
    RDATA.port = 0b00111000 ;//ACTIVAR SEGUNDA LINEA
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00001100 ;//CONFIGURAR CURSOR
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00000001 ; //CLEAR DISPLAY
    rdatasender();
    enable_up_low();
}
void clear_display()
```

```
{
    MODE = 0; //MODO COMANDOS
    RDATA.port = 0b00000001 ; //CLEAR DISPLAY
    rdatasender();
    enable_up_low();
}
void segunda_linea()
{
    MODE = 0; //MODO COMANDOS
    RDATA.port = 0b11000000 ; //POSICIONAR CURSOR
    rdatasender();
    enable_up_low();
}
void segunda_linea_final()
{
    MODE = 0; //MODO COMANDOS
    RDATA.port = 0b11001111 ; //POSICIONAR CURSOR
    rdatasender();
    enable_up_low();
}
void enable_up_low()
{
    ENABLE = 1;
    __delay_ms(10);
    ENABLE = 0;
    __delay_ms(10);
}

void SeleccionarMarcha(int SEN)
{
  if ((SEN == ADELANTE))
  {
    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        DutyCycle(45);
    }
    else if ((VELOCIDAD == SEGUNDA ))
    {
        DutyCycle(30);
    }
    else if ((VELOCIDAD == TERCERA ))
    {
        DutyCycle(10);
    }
  }

  else  if ((SEN == ATRAS))
  {
      if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        DutyCycle(55);
    }
    else if ((VELOCIDAD == SEGUNDA ))
    {
        DutyCycle(70);
    }
    else if ((VELOCIDAD == TERCERA ))
    {
```

```
        DutyCycle(90);
    }
  }
}
void SeleccionarSentido(int SEN)
{
    if ((SEN == ADELANTE ) || (SEN == NOINTRODUCIDO))
        {
        P1C = 1;
        }
    else if (SEN == ATRAS)
        {
        P1C = 0;
        }
}
void MotorDerecha_ON(int SEN)
{
    SeleccionarSentido(SEN);

    SeleccionarMarcha(SEN);

}
void MotorDerecha_ON_Giro(int SEN)
{
  SeleccionarSentido(SEN);

  if ((SEN == ADELANTE))
  {
    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO)
)//100-45=55
            {
                P1C = 0; //Giro con rueda quieta

                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100- 75 = 25
            {
                DutyCycle(75);

            }
        else if (GRADOGIRO == GRADOTRES)//100 - 55 = 45
            {
                DutyCycle(55);
            }

    }
    else if ((VELOCIDAD == SEGUNDA ))//100 - 30 = 70
    {

        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta

                DutyCycle(0);

            }
        else if (GRADOGIRO == GRADODOS)// 100 - 60 = 40
```

```
            {
                //Giro con mitad del DC
                DutyCycle(60);
            }

        else if (GRADOGIRO == GRADOTRES)//100 - 40 = 60
            {
                DutyCycle(40);
              }
    }
    else if ((VELOCIDAD == TERCERA ))//100 - 10 = 90
    {

        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100 -40 = 60
            {
                //Giro con mitad del DC
                DutyCycle(40);
            }

        else if (GRADOGIRO == GRADOTRES)// 100 - 20= 80
            {
                DutyCycle(20);
            }
    }
}

else  if ((SEN == ATRAS)) // 55
{

    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta

                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100- 75 = 25
            {
                DutyCycle(25);

            }
        else if (GRADOGIRO == GRADOTRES)//100 - 55 = 45
            {
                DutyCycle(45);
            }

    }
    else if ((VELOCIDAD == SEGUNDA ))//70
    {

        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
```

```
                DutyCycle(0);

            }
        else if (GRADOGIRO == GRADODOS)// 100 - 60 = 40
            {

                DutyCycle(40);
            }

        else if (GRADOGIRO == GRADOTRES)//100 - 40 = 60
            {
                DutyCycle(60);
            }
    }
    else if ((VELOCIDAD == TERCERA ))//90
    {

        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta

                DutyCycle(0);

            }
        else if (GRADOGIRO == GRADODOS) // 60
            {

                DutyCycle(60);
            }

        else if (GRADOGIRO == GRADOTRES)// 80
            {
                DutyCycle(80);
              }
    }
}
}
void MotorDerecha_OFF()
{
    P1C = 0;
    CCPR1L =  0b00000000; //0%
    DC1B1 = 0;
    DC1B0= 0;

}
//COMANDOS DE DIRECCION
void GirarIzq(int SEN)
{   //MOTOR IZQUIERDA APAGADO
    //MOTOR DRECHA ENCENDENDIDO
    MotorDerecha_ON(SEN);
}
void GirarDer(int SEN)
{   //MOTOR IZQUIERDA APAGADO
    //MOTOR DERECHA ENCENDENDIDO
    MotorDerecha_ON_Giro(SEN);
}
void Avanzar(int SEN)
{   //MOTOR IZQUIERDA ENCENDIDO
    //MOTOR DERECHA ENCENDIDO
```

```
    MotorDerecha_ON(SEN);
}
void Stop()
{
      MotorDerecha_OFF();
}
//LCD
void rdatasender()
{
    RC6 = RDATA.RC6;
    RB4 = RDATA.RB4;
    RC2 = RDATA.RC2;
    RC1 = RDATA.RC1;
    RC0 = RDATA.RC0;
    RA2 = RDATA.RA2;
    RA1 = RDATA.RA1;
    RA0 = RDATA.RA0;
}
void trisdsender()
{
    TRISC6 = TRISD.TRISC6;
    TRISB4 = TRISD.TRISB4;
    TRISC2 = TRISD.TRISC2;
    TRISC1 = TRISD.TRISC1;
    TRISC0 = TRISD.TRISC0;
    TRISA2 = TRISD.TRISA2;
    TRISA1 = TRISD.TRISA1;
    TRISA0 = TRISD.TRISA0;
}
void printToLCD(char* text)
{
    MODE = 1; //MODO ESCRITURA

    unsigned int i = 0;
    while(text[i] != '\0')
    {
        RDATA.port = text[i];
        rdatasender();
        enable_up_low();

        ++i;
    }
}
void MostrarVelocidad()
{
    segunda_linea();
    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
    printToLCD("Primera\0");
    }
    else if ((VELOCIDAD == SEGUNDA ))
    {
    printToLCD("Segunda\0");
    }
    else if ((VELOCIDAD == TERCERA ))
    {
    printToLCD("Tercera\0");
    }
```

```
}
void MostrarDireccion()
{

    clear_display();

    if (DIRECCION == IZQUIERDA  )
    {
        printToLCD("Girar Izquierda\0");
    }
    else if (DIRECCION == DERECHA)
    {
        printToLCD("Girar Derecha\0");
    }
    else if (SENTIDO == ADELANTE)
    {
        printToLCD("Avanzando\0");
    }
    else if (SENTIDO == ATRAS)
    {
        printToLCD("Marcha Atras\0");
    }
    else if (SENTIDO == NOINTRODUCIDO && DIRECCION == NOINTRODUCIDO)
    {
        printToLCD("Parada\0");
    }
}
void MostrarEstadoBateria()
{
    MODE = 0;
    segunda_linea_final();
    MODE = 1;
    if (ESTADO == CARGADA)
    {
    RDATA.port = 0b00000000;//Bateria Cargada
    }
    else if (ESTADO == SEMIDESCARGADA)
    {
    RDATA.port = 0b00000001;//Bateria Semidescargada
    }
    else if (ESTADO == DESCARGADA)
    {
    RDATA.port = 0b00000010;//Bateria Descargada
    }
    rdatasender();
    enable_up_low();
}
void SimbolosEstadoBateria()
{
    RDATA.port = 0b01000000 ;//SET CGRAM
    rdatasender();
    enable_up_low();
    MODE=1;
    //Bateria Cargada
    RDATA.port = 0b00001110;//PROGRAMAR FILA 1
    rdatasender();
```

```
enable_up_low();
RDATA.port = 0b00011111;//PROGRAMAR FILA 2
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 3
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 4
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 5
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 6
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 7
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 8
rdatasender();
enable_up_low();
//Bateria Semidescargada
RDATA.port = 0b00001110;//PROGRAMAR FILA 1
rdatasender();
enable_up_low();
RDATA.port = 0b0011011;//PROGRAMAR FILA 2
rdatasender();
enable_up_low();
RDATA.port = 0b00010001 ;//PROGRAMAR FILA 3
rdatasender();
enable_up_low();
RDATA.port = 0b00010001 ;//PROGRAMAR FILA 4
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 5
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 6
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 7
rdatasender();
enable_up_low();
RDATA.port = 0b00011111 ;//PROGRAMAR FILA 8
rdatasender();
enable_up_low();
//Bateria Descargada
RDATA.port = 0b00001110;//PROGRAMAR FILA 1
rdatasender();
enable_up_low();
RDATA.port = 0b0011011;//PROGRAMAR FILA 2
rdatasender();
enable_up_low();
RDATA.port = 0b00010001 ;//PROGRAMAR FILA 3
rdatasender();
enable_up_low();
RDATA.port = 0b00010001 ;//PROGRAMAR FILA 4
```

```
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00010001 ;//PROGRAMAR FILA 5
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00010001 ;//PROGRAMAR FILA 6
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00010001 ;//PROGRAMAR FILA 7
    rdatasender();
    enable_up_low();
    RDATA.port = 0b00011111 ;//PROGRAMAR FILA 8
    rdatasender();
    enable_up_low();
    MODE=0;
}
void LeerEstadoBateria()
{
    uint16_t Lectura;
    GO=1; //EMPEZAR CONVERSION
    //LECTURA DEL RESULTADO
    while (GO==1);//CUANDO GO SEA 0 HAREMOS LA LECTURA
    {
        Lectura = 0;
        Lectura = ADRESH & 0b00000011;
        Lectura = Lectura << 8;
        Lectura = Lectura + ADRESL;
        if (Lectura > 600)
        {
            ESTADO = CARGADA;
        }
        else if (Lectura < 600 && Lectura > 250 )
        {
            ESTADO = SEMIDESCARGADA;
        }
        else if (Lectura > 250)
        {
            ESTADO = DESCARGADA;
        }
    }
}
void DutyCycle(int DC)
{
    uint16_t PW;
    PW  =  DC*10.24 - 1;
    if (DC*10.24 < 1)
    {
        PW = 0;
    }
    DC1B0  = PW & 0b0000000000000001;
    PW = PW>>1;
    DC1B1  = PW & 0b0000000000000001;
    PW = PW>>1;
    CCPR1L = PW;
}

void ObstaculoDetectado()
{
```

```
    OPTION_REG = 0b11010110;
                  // - TIMER MODE
                    // - PRESCALA EN EL TIMER
                      //--- ESCALA 128
     __delay_ms(31);
    TMR0 = 0;
    while (TMR0<234) // 30ms => 30000/128 = 234
    {
        if (OBSTACULO == DETECTADO)
        {
        SENTIDO = NOINTRODUCIDO;
        DIRECCION = NOINTRODUCIDO;
        VELOCIDAD = NOINTRODUCIDO;

        MENSAJEOBSTACULO = NOENVIADO;
        ESTADOSENSOR = DESACTIVADO;
        AVISO = ESPERAR;
        break;
        }
    }
}
int BuscarGiro(int Lect)
{
unsigned int i = 0;

    while (i<=4)
    {
    if (SLIDERGIRO[i]==Lect)
        {
        RC1=1;
        return TRUE;

        }
    i++;
    }
return FALSE;
}
```

Archivo: MD.h

```
/*
 * File:   MD.h
 * Author: Marcos
 *
 * Created on 31 de mayo de 2015, 16:46
 */

#ifndef MD_H
#define MD_H

#include <htc.h>
#include <stdint.h>
#include <pic16f690.h>

#define _XTAL_FREQ  4000000
#define P1C RC3

volatile unsigned int SENTIDO = 0; // 1 ADELANTE, 2 ATRAS, 0 NO
INTRODUCIDO
```

```
#define ADELANTE 1
#define ATRAS 2
#define NOINTRODUCIDO 0

volatile unsigned int DIRECCION = 0; // 1 IZQUIERDA, 2 DERECHA, 0 NO
INTRODUCIDO,4 PRIORIDAD

#define IZQUIERDA 1
#define DERECHA 2
#define PRIORIDAD 4

volatile unsigned int VELOCIDAD = 0;  // 1 PRIMERA, 2 SEGUNDA, 3 TERCERA

#define PRIMERA 1
#define SEGUNDA 2
#define TERCERA 3

volatile unsigned int GRADOGIRO = 9;  // 1,2,3,4 DE MENOS A MAS GIRO

#define GRADOUNO 9
#define GRADODOS 10
#define GRADOTRES 11

volatile unsigned int ESTADO = 0;  // 1 CARGADA, 2 SEMIDESCARGADA, 3
DESCARGADA

#define CARGADA 1
#define SEMIDESCARGADA 2
#define DESCARGADA 3

volatile unsigned int ESTADOSENSOR = 1;  // 1 ACTIVADO, 0 DESACTIVADO

#define ACTIVADO 1
#define DESACTIVADO 0

volatile unsigned int MENSAJEOBSTACULO = 1;  // 1 SI, 0 NO

#define ENVIADO 1
#define NOENVIADO 0
volatile uint8_t Lectura;


#define AVISO RA5
#define ESPERAR 0
#define NOESPERAR 1

#define OBSTACULO RC4
#define NODETECTADO 0
#define DETECTADO 1

#define BOTON_PRIMERA 1
#define BOTON_SEGUNDA 2
#define BOTON_TERCERA 3
#define BOTON_ADELANTE 4
#define BOTON_ATRAS 6
#define BOTON_DERECHA 5
#define BOTON_IZQUIERDA 7
#define BOTON_STOP 8

#define BOTON_SENSOR 13
```

```
int SLIDERGIRO[4] = {9,10,11};

#define TRUE 1
#define FALSE 0


void Inicializacion();

void IncializarMotores();

void SeleccionarMarcha(int SEN);

void SeleccionarSentido(int SEN);

void MotorDerecha_ON(int SEN);

void MotorDerecha_OFF();

void GirarIzq(int SEN);

void GirarDer(int SEN);

void Avanzar(int SEN);

void Stop();

void MotorDerecha_ON_Giro(int SEN);

void DutyCycle(int DC);

//LCD

#define ENABLE RB6
#define MODE RC7

union {
    uint8_t         port;
    struct {
    unsigned  RC6   : 1;
    unsigned  RB4   : 1;
    unsigned  RC2   : 1;
    unsigned  RC1   : 1;
    unsigned  RC0   : 1;
    unsigned  RA2   : 1;
    unsigned  RA1   : 1;
    unsigned  RA0   : 1;
    };
} RDATA;

union {
    uint8_t         port;
    struct {
    unsigned    TRISC6     : 1;
    unsigned    TRISB4     : 1;
    unsigned    TRISC2     : 1;
    unsigned    TRISC1     : 1;
    unsigned    TRISC0     : 1;
    unsigned    TRISA2     : 1;
    unsigned    TRISA1     : 1;
```

```
    unsigned    TRISA0      : 1;
    };
} TRISD;

void InicializarLCD();

void InicializarNivelBateria();

void SimbolosEstadoBateria();

void rdatasender();

void trisdsender();

void enable_up_low();

void clear_display();

void segunda_linea();

void segunda_linea_final();

void printToLCD(char* text);

void MostrarLCD(char text);

void LeerEstadoBateria();

void MostrarVelocidad();

void MostrarDireccion();

void MostrarEstadoBateria();

void ObstaculoDetectado();

int BuscarGiro(int Lect);

#endif  /* MD_H */
```

X. Código utilizado en el microcontrolador izquierdo

Archivo: MotorIzquierdoPrincipal.c

```
#include <htc.h>
#include <stdint.h>
#include "MI.h"

__CONFIG(MCLRE_OFF & CP_OFF & WDTE_OFF & FOSC_INTRCIO & PWRTE_OFF &
IESO_OFF & FCMEN_OFF & BOREN_OFF );

void interrupt ISR()
{
    if (RCIF==1)
    {
            Lectura = RCREG;
```

```
        if (Lectura==BOTON_ADELANTE) // BOTON ADELANTE SELECCIONADO
        {

            SENTIDO = ADELANTE;
            DIRECCION = PRIORIDAD;
        }

        else if (Lectura==BOTON_ATRAS) // BOTON ATRAS SELECCIONADO
        {
            SENTIDO = ATRAS;
            DIRECCION = PRIORIDAD;
        }
        else if (Lectura==BOTON_DERECHA) // BOTON GIRO DERECHA
SELECCIONADO
        {
            DIRECCION = DERECHA;
        }

        else if (Lectura==BOTON_IZQUIERDA) // BOTON GIRO IZQUIERDA
SELECCIONADO
        {
            DIRECCION = IZQUIERDA;
        }
        else if (Lectura==BOTON_STOP) // BOTON STOP SELECCIONADO
        {
            SENTIDO = NOINTRODUCIDO;
            DIRECCION = NOINTRODUCIDO;
            VELOCIDAD = NOINTRODUCIDO;
        }
        else if (Lectura==BOTON_PRIMERA) // BOTON PRIMERA VELOCIDAD
SELECCIONADO
        {
            VELOCIDAD = PRIMERA;
        }
        else if (Lectura==BOTON_SEGUNDA) // BOTON SEGUNDA VELOCIDAD
SELECCIONADO
        {
            VELOCIDAD = SEGUNDA;
        }
        else if (Lectura==BOTON_TERCERA) // BOTON TERCERA VELOCIDAD
SELECCIONADO
        {
            VELOCIDAD = TERCERA;
        }
        else if (Lectura==BOTON_SENSOR)
        {
        ESTADOSENSOR = 1;
        }
        else if (BuscarComp(Lectura))
        {
        Compensacion = Lectura - 30;
        }
        else if (BuscarGiro(Lectura)) // SLIDER POSICION 1 SELECCIONADO
        {
        GRADOGIRO = Lectura;
        }
        while ( AVISO == ESPERAR);
    }
}
```

```
void main()
{
    __delay_ms(2000);
    Inicializacion();
    IncializarMotores();
    InicilizarSonar();
    while ( AVISO == ESPERAR);
    for (;;)
    {
            __delay_ms(200);
            if (ESTADOSENSOR == 0)
            {
            __delay_ms(100);
            }
            else if (ESTADOSENSOR == 1)
            {
            DetectarObstaculo();
            }
            __delay_ms(100);
            if (DIRECCION == IZQUIERDA && (SENTIDO == ADELANTE || SENTIDO
== NOINTRODUCIDO) )
            {
               GirarIzq(ADELANTE);
            }
            else if (DIRECCION == IZQUIERDA && SENTIDO == ATRAS )
            {
               GirarIzq(ATRAS);
            }
            else if (DIRECCION == DERECHA && ( SENTIDO == ADELANTE ||
SENTIDO == NOINTRODUCIDO) )
            {
               GirarDer(ADELANTE);
            }
            else if (DIRECCION == DERECHA && SENTIDO == ATRAS )
            {
               GirarDer(ATRAS);
            }
            else if (SENTIDO == ADELANTE)
            {
               Avanzar(ADELANTE);
            }
            else if (SENTIDO == ATRAS)
            {
               Avanzar(ATRAS);
            }
            else if (SENTIDO == NOINTRODUCIDO && DIRECCION ==
NOINTRODUCIDO && VELOCIDAD == NOINTRODUCIDO)
            {
                Stop();
                if (OBSTACULO == DETECTADO)
                {
                    while ( AVISO == ESPERAR);
                }
            }
    }
```

```
}
```

Archivo: MI.c

```
#include "MI.h"

void Inicializacion()
{
    ANSEL = 0; // TODOS PUERTOS I/O DIGITALES
    ANSELH = 0;
    TRISB = 0b01111111;
    // BRGH = 1 VELOCIDAD ALTA, SYNC = 0 ASINCRONO
    TXSTA = 0b00000100;
    SPBRG = 0b00011001; //Numero 25
    BAUDCTL = 0b00000000;//BRG16 = 0; MODO DE 8 BITS
    PIE1 = 0b00100000; //RCIE = 1; ACTIVAR LOS EUSART RECIVER
INTERRUPCIONES
    //SPEN = 1;
    //RX9 DESACTIVADO EL NOVENO BIT
    //CREN ACTIVADO RECIVIDOR
    RCSTA = 0b10110000;
    INTCON = 0b11000000; //PERMITIR INTERRUPCIONES EXTERNAS
}

void IncializarMotores()
{
    PR2  =    0b11111111; //255
    TRISC5=0;
    TRISC3=0;
    T2CON =   0b00000110; //Timer 2 encendido y PREESCALA 16
    CCP1M0 = 0; //PWM MODO, todos valores alto en activo
    CCP1M1 = 0;
    CCP1M2 = 1;
    CCP1M3 = 1;
    //MODO HALF BRIDGE
    P1M1=1; P1M0 = 0; // A Y B PWM, C Y D PUERTOS
}
void SeleccionarSentido(int SEN)
{
    if ((SEN == ADELANTE ) || (SEN == NOINTRODUCIDO))
        {
        P1C = 1;
        }
    else if (SEN == ATRAS)
        {
        P1C = 0;
         }
}
void MotorIzquierda_ON(int SEN)
{
    SeleccionarSentido(SEN);
    SeleccionarMarcha(SEN);
}

void MotorIzquierda_OFF()
{
```

```
    P1C = 0;
    CCPR1L =  0b00000000; //0%
    DC1B1 = 0;
    DC1B0= 0;
}

void GirarIzq(int SEN)
{   //MOTOR IZQUIERDA APAGADO
    //MOTOR DRECHA ENCENDENDIDO
    MotorIzquierda_ON_Giro(SEN);
}

void GirarDer(int SEN)
{   //MOTOR IZQUIERDA APAGADO
    //MOTOR DRECHA ENCENDENDIDO
    Compensacion = 0;
    MotorIzquierda_ON(SEN);
}

void Avanzar(int SEN)
{   //MOTOR IZQUIERDA ENCENDIDO
    //MOTOR DRECHA ENCENDIDO
    MotorIzquierda_ON(SEN);
}

void Stop()
{
     MotorIzquierda_OFF();
}
void SeleccionarMarcha(int SEN)
{
  if ((SEN == ADELANTE)) // P1C = 1
  {
    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        DutyCycle(45-Compensacion);
    }
    else if ((VELOCIDAD == SEGUNDA ))
    {
        DutyCycle(30-Compensacion);
    }
    else if ((VELOCIDAD == TERCERA ))
    {
        DutyCycle(10-Compensacion);
    }
  }
  else  if ((SEN == ATRAS))
  {
      if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        DutyCycle(55+Compensacion);
    }
    else if ((VELOCIDAD == SEGUNDA ))
    {
        DutyCycle(70+Compensacion);
    }
    else if ((VELOCIDAD == TERCERA ))
    {
        DutyCycle(90+Compensacion);
```

```
    }
  }
}
void DutyCycle(int DC)
{
    uint16_t PW;
    PW  =  DC*10.24 - 1;
    if (DC*10.24 < 1)
    {
        PW = 0;
    }
    DC1B0  = PW & 0b0000000000000001;
    PW = PW>>1;
    DC1B1  = PW & 0b0000000000000001;
    PW = PW>>1;
    CCPR1L = PW;
}

void InicilizarSonar()
{
    TRISA2= 0; //TRIGGER COMO OUTPUT
    TRISC2=0;
    OPTION_REG = 0b11010010;
                  // - TIMER MODE
                    // - PRESCALA EN EL TIMER
                      //--- ESCALA 8
}
void EnviarPulso()
{
    TRIGGER=1;
    __delay_us(10);
    TRIGGER=0;
}

void DetectarObstaculo()
{
    uint8_t Lectura;
    OBSTACULO = NODETECTADO
    __delay_ms(30);
    EnviarPulso();
    while (ECHO == 0);// LOW TO HIGH
    T0IF=0;             // FLAG OVERFLOW A 0
    TMR0=0;             // TIMER0 A 0
    while (ECHO == 1);// HIGH TO LOW
    Lectura = TMR0;
    if (Lectura < 218)// OBSTACULO A 20CM
    {
        if (T0IF==0) //Si NO SE PRODUCE OVERFLOW
        {
        SENTIDO = NOINTRODUCIDO;
        DIRECCION = NOINTRODUCIDO;
        VELOCIDAD = NOINTRODUCIDO;
        ESTADOSENSOR = DESACTIVADO;
        OBSTACULO = DETECTADO;
        }

    }
    __delay_ms(30);
}
```

```
int BuscarComp(int Lect)
{
unsigned int i = 0;

    while (i<= 21)
    {
    if (SLIDERCOMPENSACION[i]==Lect)
        {
        return TRUE;
        }
    i++;
    }
return FALSE;
}

int BuscarGiro(int Lect)
{
unsigned int i = 0;

    while (i<=3)
    {
    if (SLIDERGIRO[i]==Lect)
        {
        return TRUE;
        }
    i++;
    }
return FALSE;
}

void MotorIzquierda_ON_Giro(int SEN)
{
  SeleccionarSentido(SEN);
  if ((SEN == ADELANTE))
  {
    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO)
)//100-45=55
            {
                P1C = 0;
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100- 75 = 25
            {
                DutyCycle(75);

            }
        else if (GRADOGIRO == GRADOTRES)//100 - 55 = 45
            {
                DutyCycle(55);
            }

    }
    else if ((VELOCIDAD == SEGUNDA ))//100 - 30 = 70
    {

        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
```

```
            {
                P1C = 0; //Giro con rueda quieta
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS)// 100 - 60 = 40
            {
                DutyCycle(60);
            }

        else if (GRADOGIRO == GRADOTRES)//100 - 40 = 60
            {
                DutyCycle(40);
             }
    }
    else if ((VELOCIDAD == TERCERA ))//100 - 10 = 90
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100 -40 = 60
            {
                DutyCycle(40);
            }

        else if (GRADOGIRO == GRADOTRES)// 100 - 20= 80
            {
                DutyCycle(20);
              }
    }
}

else  if ((SEN == ATRAS)) // 55
{

    if (((VELOCIDAD == PRIMERA ) || (VELOCIDAD == NOINTRODUCIDO)))
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) //100- 75 = 25
            {
                DutyCycle(25);
            }
        else if (GRADOGIRO == GRADOTRES)//100 - 55 = 45
            {
                DutyCycle(45);
            }

    }
    else if ((VELOCIDAD == SEGUNDA ))//70
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
```

```
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS)// 100 - 60 = 40
            {

                DutyCycle(40);
            }

        else if (GRADOGIRO == GRADOTRES)//100 - 40 = 60
            {
                DutyCycle(60);
              }
    }
    else if ((VELOCIDAD == TERCERA ))//90
    {
        if ( (GRADOGIRO == GRADOUNO) || (GRADOGIRO == NOINTRODUCIDO) )
            {
                P1C = 0; //Giro con rueda quieta
                DutyCycle(0);
            }
        else if (GRADOGIRO == GRADODOS) // 60
            {

                DutyCycle(60);
            }

        else if (GRADOGIRO == GRADOTRES)// 80
            {
                DutyCycle(80);
              }
    }
}
}
```

Archivo: MI.h

```
/*
 * File:   MD.h
 * Author: Marcos
 *
 * Created on 31 de mayo de 2015, 16:46
 */

#ifndef MD_H
#define MD_H

#include <htc.h>
#include <stdint.h>
#include <pic16f690.h>

#define _XTAL_FREQ  4000000
#define P1C RC3
#define TRIGGER  RA2
#define ECHO  RC0

volatile unsigned int SENTIDO = 0; // 1 ADELANTE, 2 ATRAS, 0 NO
INTRODUCIDO

#define ADELANTE 1
#define ATRAS 2
```

```
#define NOINTRODUCIDO 0

volatile unsigned int DIRECCION = 0; // 1 IZQUIERDA, 2 DERECHA, 0 NO
INTRODUCIDO,4 PRIORIDAD

#define IZQUIERDA 1
#define DERECHA 2
#define PRIORIDAD 4

volatile unsigned int VELOCIDAD = 0;  // 1 PRIMERA, 2 SEGUNDA, 3 TERCERA

#define PRIMERA 1
#define SEGUNDA 2
#define TERCERA 3

#define AVISO RA5
#define ESPERAR 0
#define NOESPERAR 1

#define OBSTACULO RC2
#define NODETECTADO 0
#define DETECTADO 1

volatile uint8_t Lectura;

#define BOTON_PRIMERA 1
#define BOTON_SEGUNDA 2
#define BOTON_TERCERA 3
#define BOTON_ADELANTE 4
#define BOTON_ATRAS 6
#define BOTON_DERECHA 5
#define BOTON_IZQUIERDA 7
#define BOTON_STOP 8
#define BOTON_SENSOR 13

volatile unsigned int ESTADOSENSOR = 1;  // 1 ACTIVADO, 0 DESACTIVADO

#define ACTIVADO 1
#define DESACTIVADO 0

volatile unsigned int Compensacion = 0;

int SLIDERCOMPENSACION[21] =
{20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40};

#define TRUE 1
#define FALSE 0

volatile unsigned int GRADOGIRO = 9;  // 1,2,3,4 DE MENOS A MAS GIRO

#define GRADOUNO 9
#define GRADODOS 10
#define GRADOTRES 11

int SLIDERGIRO[4] = {48,9,10,11};

void Inicializacion();

void IncializarMotores();

void SeleccionarMarcha(int SEN);
```

```
void SeleccionarSentido(int SEN);
void MotorDerecha_ON(int SEN);
void MotorDerecha_OFF();
void GirarIzq(int SEN);
void GirarDer(int SEN);
void Avanzar(int SEN);
void Stop();
void DutyCycle(int DC);
void InicilizarSonar();
void EnviarPulso();
void DetectarObstaculo();
int BuscarComp(int Lect);
int BuscarGiro(int Lect);
void MotorIzquierda_ON_Giro(int SEN);
#endif  /* MD_H */
```