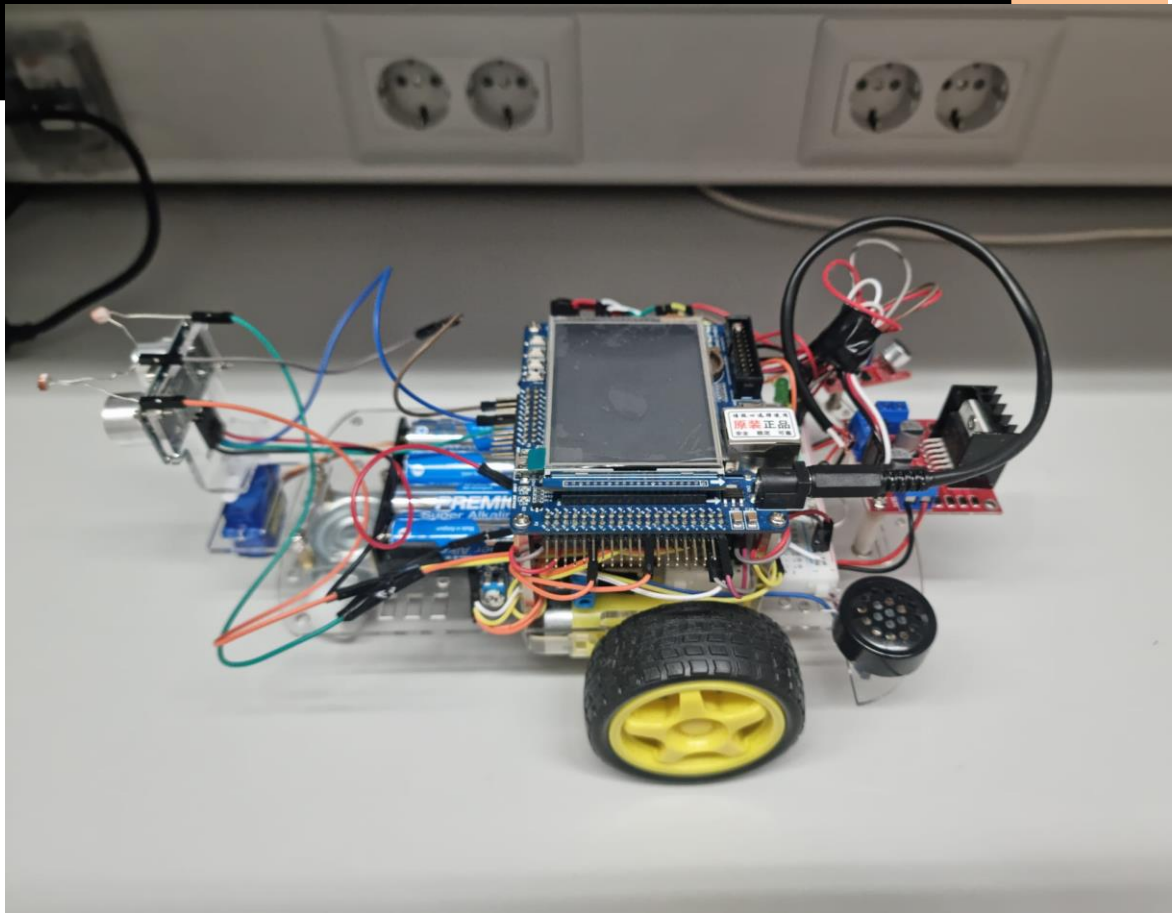


2022

# DISEÑO DE UN ROBOT CON TRACCION DIFERENCIAL



Álvaro Martín Fernández

SISTEMAS ELECTRONICOS DIFITALES

AVANZADOS

Unversidad de Álcala

1-6-2022



## Índice

### Contenido

<b>INTRODUCCIÓN .....</b>	<b>5</b>
<b>PROYECTO.....</b>	<b>5</b>
<b>1. Descripción del Hardware.....</b>	<b>6</b>
1.1.    LPC1768-Mini DK2 Development board +2.8" TFT LCD <sup>1</sup> .....	6
1.2.    Motores. ....	7
1.3.    ServoMotor.....	7
1.4.    Puente en H .....	7
1.5.    Sensor Ultrasonidos (Periférico Distancia). ....	8
1.6.    ULINK2/ME Cortex Debugger .....	8
1.7.    Portapilas y pilas .....	9
1.8.    Coche. ....	9
1.9.    Altavoz .....	10
1.10.   Micrófono .....	10
1.11.   Sensores de luz LDR .....	10
1.12.   Bluetooth/USB .....	11
1.13.   Nunchuck .....	11
1.14.   Cables.....	12
1.15.   Conexion Pines.....	13
1.16.   Diagrama de conexion del Sistema.....	15
<b>2. Descripción del Software. ....</b>	<b>16</b>
2.1.    Main. ....	16
2.2.    PWM (Motores y Servo). ....	17
2.3.    TP_Simple. ....	19
2.4.    Nunchuck. ....	31
2.5.    ADC. ....	34
2.6.    DAC. ....	36
2.7.    Interrupciones.....	36
2.8.    Bluetooth. ....	37
2.9.    HTTP_CGI.C.....	40
2.10.   WatchDog .....	42
2.11.   Statechart Menú (MAQUINA DE ESTADOS).....	43
2.12.   Timer .....	46

<b>3. <i>Análisis de ejecutibilidad del sistema.</i></b>	<b>53</b>
<b>4. <i>Manual de usuario.</i></b>	<b>55</b>
4.1 Introducción.....	55
4.2 Menú principal.....	55
4.3 Modo Manual. ....	56
4.4 Modo Automatico.....	57
4.5 Modo Depuracion.....	58
4.6 Modo Online. ....	59
4.7 Conexion Bluetooth/USB. ....	62
<b>ANEXO.....</b>	<b>64</b>
I. Código fuente del Proyecto.....	64
II. Hojas de datos de los chip utilizados.....	96

## INTRODUCCIÓN

El proyecto desarrollado consiste en el diseño de un sistema que se basara en el diseño de un robot con tracción diferencial que se puede mover mediante el panel táctil de la placa, mediante el mando nunchuck de la Wii, mediante conexión remota a través de una comucacion seria atraves de Bluetooth/usb y a través de TCP/IP servidor Web. El coche también tendrá un periférico de distancia, un sensor de luminosidad,ademas reproducirá y grabara audio y tendrá un modo de movimiento automático.

## PROYECTO

El sistema tendrá tres modos de funcionamiento con las siguientes especificaciones.

a. Modo Manual:

- (1) El movimiento del robot sera controlado con el mando Nunchuck de la Wii con el joystick.
- (1) El movimiento del servomotor se controlara con inclinaciones la del mando Nunchuck.
- (2) Si detecta un objeto delante del robot, el robot se detendrá y emitirá un pitido discontinuonde un segundo de periodo.
- (3) Al pulsar el boton C del mando Nunchuck, el robot grabara audio y al pulsar el boton Z del mando lo reproducira.
- (4) El robot pasara al modo Automatico cuando se pulsen los dos botones del mando de la Wii de forma simultanea durante menos de 2 segundos o los pulsadores Key1 y Key2 de la tarjeta, y pasara al modo Depuracion cuando se pulsen los dos botones durante mas de 2 segundos o los pulsadores Key1 y Key2 de la tarjeta.
- (5) El display visualizara la informacion recibida por los sensores y la velocidad de los motores.

b. Modo Automático:

- (1) El robot se movera aleatoriamente buscando una fuente de luz lo suficientemente fuerte interesa ser seguida. Cuando se detecte la fuente de luz, el robot la seguira.
- (2) Si en el recorrido el robot detecta un objeto frente a el, intentara evitarlo para mantenerse siguiendo la luz y emitira un pitido discontinuo de un segundo de periodo.
- (3) Si el robot deja de detectar la luz, intentara buscarla con dos barridos de 180o del servomotor y si no la encuentra, se iniciara el movimiento aleatorio de navegacion.
- (4) El robot pasara a modo Manual al pulsar el boton C del mando Nunchuck durante mas de 2 segundos.
- (5) El robot pasara a modo Configuracion cuando se pulsen los dos pulsadores Key1 y Key2 de la tarjeta durante mas de 2 segundos.

c. Modo Depuración:

- (1) Utilizando el display táctil se podrá controlar el movimiento de cada motor por separado (motores derecho e izquierdo y el servomotor), visualizar las medidas de los sensores, y las variables que proporciona el mando Nunchuck.
- (2) También se visualizará la dirección IP de la tarjeta.
- (3) Al pulsar Key 1, grabará audio durante un segundo y al pulsar Key 2, reproducirá el audio grabado.
- (4) El robot pasará al modo Automático cuando se pulsen simultáneamente durante menos de 2 segundos los pulsadores Key1 y Key2 de la tarjeta, y pasará al modo Manual cuando se pulsen los pulsadores Key1 y Key2 de la tarjeta durante más de 2 segundos.

## 1. Descripción del Hardware.

A continuación, se comenta la descripción Hardware del proyecto y se mostrará un esquema de conexiones

### 1.1. LPC1768-Mini DK2 Development board +2.8" TFT LCD<sup>1</sup>.

La LCD Interface utilizada se trata de una interfaz en paralelo de 16 bits al ser más rápida que la interfaz SPI serie también disponible para esta tarjeta de desarrollo.

Las características principales del sistema son:

- 32 niveles de prioridad para interrupciones programables.
- Tiene un total de 100 pines, incluyendo los pines de 5V y masa.
- 8 timers disponibles; de los cuales 4 son timers generales, 2 son PWM y los dos restantes son SysTick y RIT timer.
- Dispone de 4 entradas de interrupción externas específicas.
- Posee tres tipos de osciladores: un oscilador RC interno (IRC) el cual funciona por defecto a 4MHz; un oscilador principal a 12 MHz en este caso; y un oscilador RTC de 32,768KHz.
- Dispone de un ADC con 12 bits, de tipo aproximaciones sucesivas. Además la frecuencia de muestreo máxima es de 200KHz.
- Posee puerto Ethernet incorporado.



Figura 1. LPC1768-MiniDK2+TFT LCD con pines macho

### 1.2. Motores.

Se hace uso de dos motores, los cuales son utilizados para desplazar las dos ruedas del coche.



Figura 2. Motor

### 1.3. ServoMotor.

Hacemos uso de un servomotor para poder mover los sensores de ultrasonidos y de luz para poder hacer barridos y detectar objetos o bien luz.

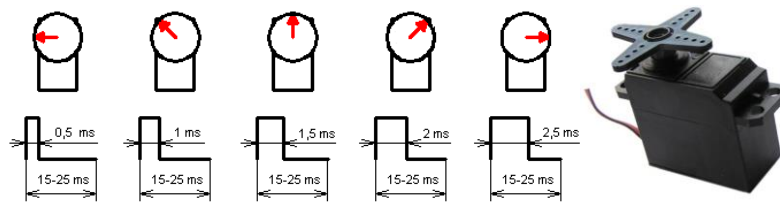


Figura 3. Servomotor

### 1.4. Puente en H

El puente en H externo que se utiliza para controlar los dos servomotores es un dispositivo que da la posibilidad de controlar dos motores en ambas direcciones cada uno. Nosotros realizaremos el método de funcionamiento de conexión a 6 hilos.

#### Conexión de los motores a 6 hilos



Figura 4 Dispositivo Puente en H

### 1.5. Sensor Ultrasonidos (Periférico Distancia).

Para la recepción de la distancia con objetos hacemos uso de un sensor de ultrasonidos, este sensor tendrá la funcionalidad que el coche se para antes de que se colisione con cualquier obstáculo. Donde según las especificaciones la distancia será la duración del pulso en microsegundos entre 58.

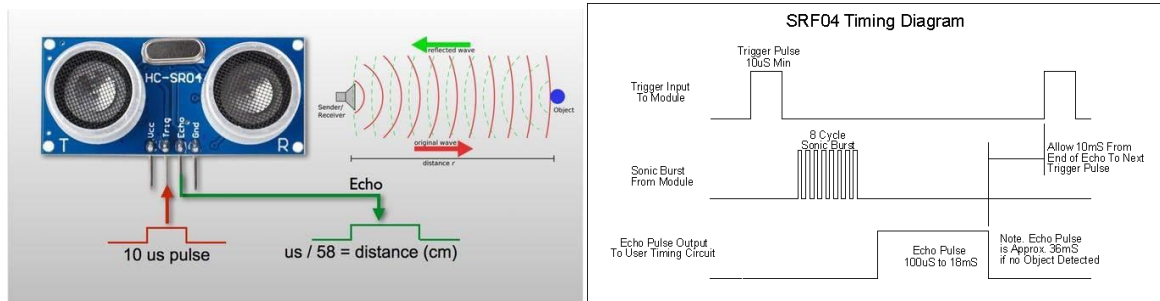


Figura 5 Sensor de ultrasonidos.

### 1.6. ULINK2/ME Cortex Debugger

Este dispositivo hace que la depuración la depuración del software. Permite ejecutar el programa paso a paso y así detectar el error de una manera sencilla y con la posibilidad de ver más detalles y la ejecución en la placa. Se conecta a la placa con un bus y a la computadora a través de un USB.

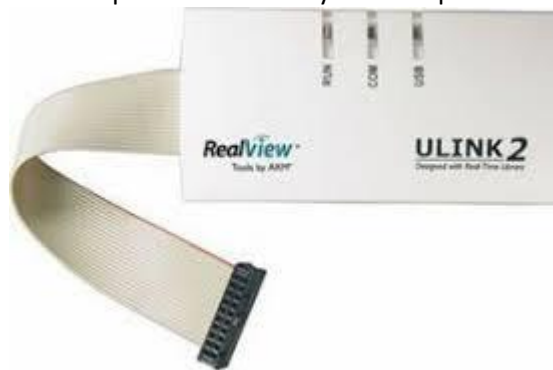


Figura 6. ULINK2



### 1.7. Portapilas y pilas

Para la alimentación de los motores se ha usado un portapilas formado por 4 pilas a 1,5V y para la alimentación de la Mini DK2 hemos utilizado dos pilas de 3,6V.

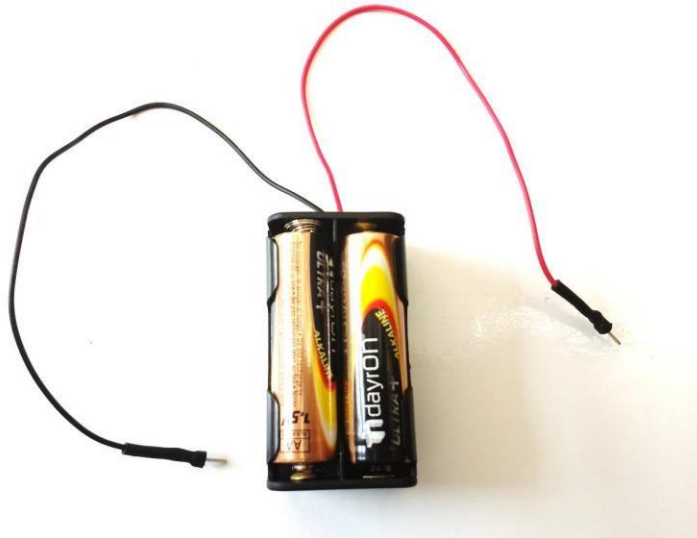


Figura 7. Portapilas y pilas

### 1.8. Coche.

El coche será el esqueleto sobre el cual el sistema está montado.



Figura 8. Coche

### 1.9. Altavoz

Se hará uso de un altavoz para poder reproducir el audio que ha sido almacenado y procesado previamente o bien reproducir un mensaje de alarma también haremos uso de un modulo amplificador de audio LM386.4

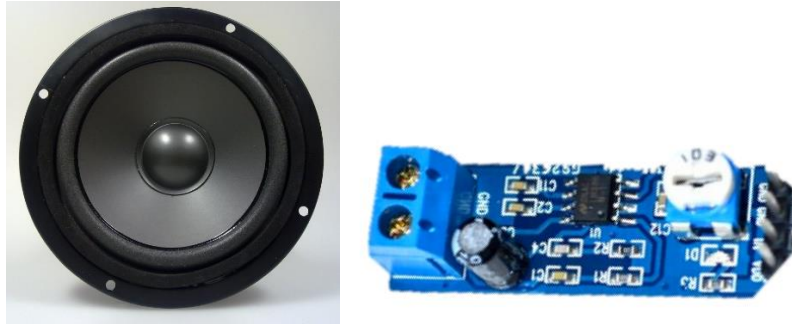


Figura 9. Altavoz y Amplificador de audio

### 1.10. Micrófono

Se hará uso de un micrófono para poder grabar muestras de sonido para reproducirlas posteriormente. También haremos uso de dos condensadores y una resistencia para eliminar un poco el ruido.

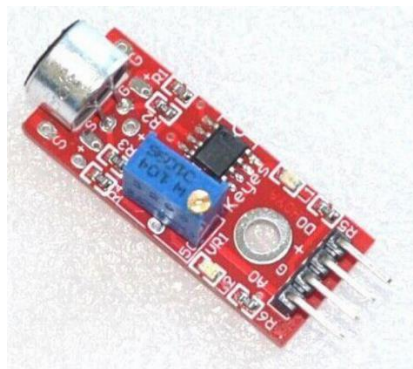


Figura 10. Micrófono

### 1.11. Sensores de luz LDR

Hacemos uso sensores de luz para que el coche se mueva de manera autónoma en el modo automatico siguiendo siempre la fuente de luz.

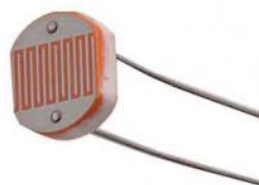


Figura 10. Sensor de Luz(LDR)

### 1.12. Bluetooth/USB

Hacemos uso receptor Bluetooth o bien de un USB que irá conectado a nuestro sistema de manera que este se pueda controlar de manera remota mediante una conexión asíncrona y a través de una aplicación de terminal Bluetooth o bien de la aplicación Termite.

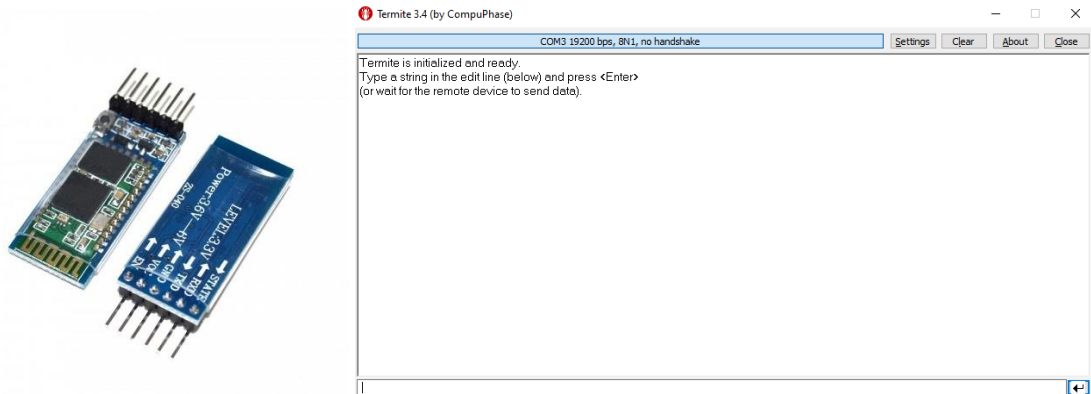


Figura 12. Receptor Bluetooth HC06 y aplicación Termite.

### 1.13. Nunchuck

También haremos uso de un receptor Nunchuck el cual nos permitirá mover el motor y el servo en el modo manual. Este receptor formará un módulo que irá conectado a la placa y mandará las solitudes que han sido enviadas mediante el mando nunchuck.

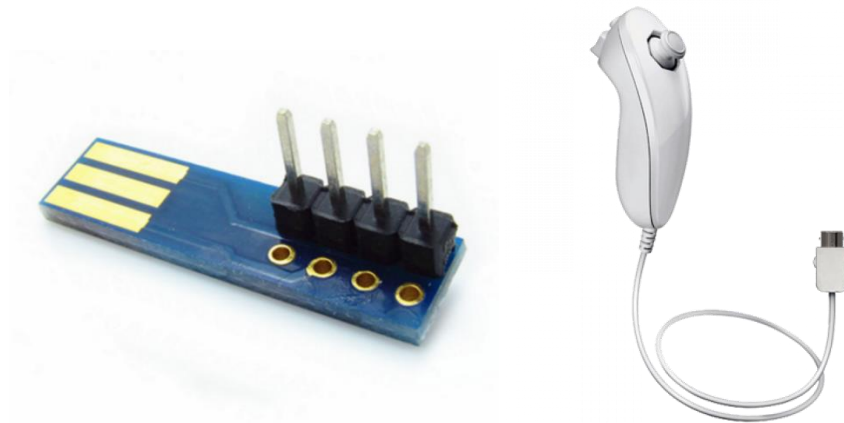


Figura 13. Receptor Nunchuck y Mando Nunchuck

### 1.14. Cables

Haremos uso de un cable Ethernet para realizar una conexión al servidor Web que contiene nuestra placa integrada, también haremos uso de jumper para la conexión de los componentes con la placa y por último haremos utilización de un cable USB para la conexión serie asincrónica o bien como método alternativo de alimentación de la placa.



Figura 14. Cable Ethernet



Figura 15. Jumper hembra-hembra y jumper macho-hembra



Figura 16. Cable USB-mini USB

### 1.15. Conexión Pines.

A continuación detallaremos la conexión de los pines con los diferentes módulos de nuestro Sistema.

#### 1.15.1 Motores, Puente en H y Servomotor.

Los pines usados por los motores serán dos para las señales PWM y otros dos para definir el puerto de salida.

Motor Der(ENA)	P3.25	PWM1.2
Motor Izq (ENB)	P3.26	PWM1.3
IN1	P1.20	PUENTE EN H
IN2	P1.21	PUENTE EN H
IN3	P1.24	PUENTE EN H
IN4	P1.25	PUENTE EN H
SERVOMOTOR	P1.23	PWM1.4

#### 1.15.2 Pantalla táctil.

Los pines usados por la LCD no van unidos con ningún jumper a ningún otro punto, no obstante es importante señalar cuáles son estos pines ya que si alguno de éstos es usado para otro fin el funcionamiento del sistema se verá afectado.

En la Tabla se detallan los pines usados por el TFT táctil.

P0.6	P1.27
P0.7	P1.28
P0.8	P1.29
P0.9	P2.0
P0.15	P2.1
P0.16	P2.2
P0.17	P2.3
P0.18	P2.4
P0.19	P2.5
P0.20	P2.6
P0.21	P2.7
P0.22	P2.8
P1.26	P2.13

Es importante recordar que ninguno de los pines listados en la Tabla se debe usar para cualquier función del sistema ya que éstos son usados por el display táctil.

#### 1.15.3 Altavoz

Para el altavoz haremos uso de un pin para el DAC.

AOUT	P0.26
------	-------

#### 1.15.4 Micrófono

Para el micrófono haremos uso de un pin para hacer uso del ADC.

AD0.2	P0.25
-------	-------

### 1.15.5 LDRs

Para los LDRs haremos uso de dos pines para hacer uso del ADC.

AD0.1	P0.24
AD0.5	P1.31

### 1.15.6 Sensor Ultrasonidos

Para el sensor de ultrasonidos al igual que el micrófono vamos a hacer uso de dos pines un en modo MAT para enviar un trigger y otro en modo CAP para capturar el eco.

MAT3.1	P0.11
CAP3.0	P0.23

### 1.15.7 Bluetooth o conexión USB

Para el uso del receptor de bluetooth hacemos uso de dos pines uno será la transmisión TXD0 y el otro para la recepción RXD0, el otro método que podremos utilizar será una conexión mediante usb donde tiene integrado un chip que se encarga de realizar la comunicación serie mediante el UART

TXD0	P0.2
RXD0	P0.3

### 1.15.8 Nunchuck

Para el uso del receptor del mando nunchuck hacemos uso de dos pines uno para el SDA y otro para el SCL.

SDA	P0.0
SCL	P0.1

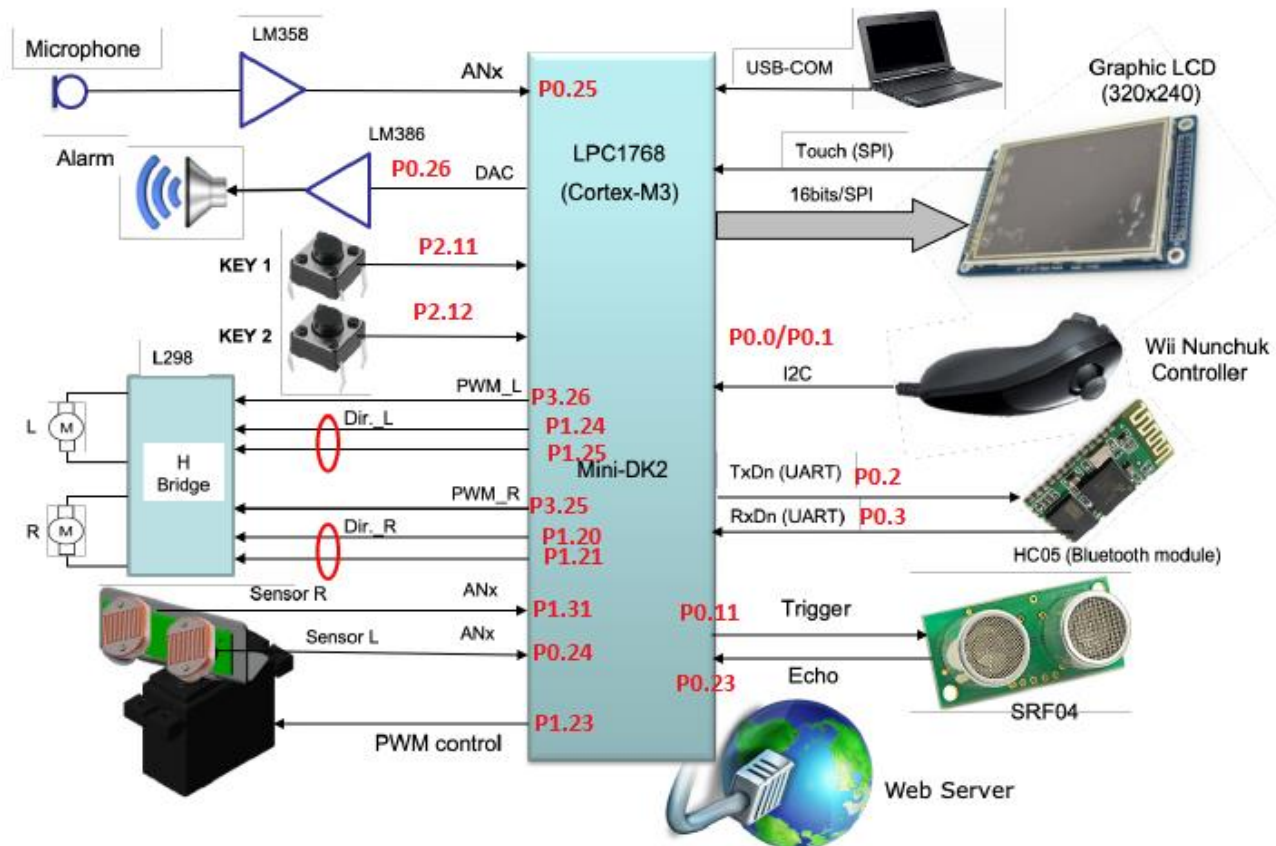
### 1.15.9 Ethernet.

La conexión con TCP/IP se realiza a través del módulo Ethernet de la tarjeta. Su unión se realiza con un cable Ethernet como se ha comentado anteriormente. No obstante, como en el caso del display TFT, existen unos determinados pines que son usados aunque no estén conectados físicamente. En la Tabla se detallan los pines que no se pueden usar para otra función.

P1.0	P1.10
P1.1	P1.14
P1.4	P1.15
P1.8	P1.16
P1.9	P1.17

### 1.16. Diagrama de conexión del Sistema.

A continuación representaremos el diagrama del nuestro Sistema conjunto.



## 2. Descripción del Software.

A continuación, vamos a la función que tendrá el código de nuestro proyecto. Comentaremos cada parte del proyecto por separado y terminaremos juntando todo en un mismo proyecto.

En primer lugar describiremos lo que hace nuestro código y para finalizar mostraremos este código.

Para la realización del presente proyecto se han usado los siguientes programas:

- Keil  $\mu$ Vision 4  
Software para la gestión de proyectos, edición de código fuente, depuración de programas y simulación completa.
- Termite  
Aplicación para windows para poder controlar el coche de manera remota mediante USB

El programa se divide en los siguientes módulos:

### 2.1. Main.

Esta función es la función principal de cualquier programa. En este archivo se encuentran la definición de las variables globales declaradas en los siguientes módulos como son la velocidad de los motores la posición del servo, la estructuras de las pantallas de los menus principales y el estado en el que los encontramos.

```
extern uint8_t Estado;
extern struct t_screenZone Manual;
extern struct t_screenZone Automatico;
extern struct t_screenZone Depuracion;
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;
```

También tendremos las funciones init y timer\_poll y la variable externa U32 CheckMedia del modulo del servidor web.

La funcionalidad de la función main es iniciar todos los módulos externos y sus configuraciones mediante las siguientes funciones.

```
LCD_Initializtion();
LCD_Clear(Black);
TP_Init();
Estado= 0;
// TouchPanel_Calibrate();
configPWM();
PuenteH ();
Nunchuck_Init();
init_ADC_LDRs();
init_TIMERS();
init_DAC();
init_EINT();
NVIC_SetPriorityGrouping(2);
uart0_init(19200);
Nunchuck_Init();
WDT_init();
init ();
```



Por ultimo la función Main contara con una función While (1) la cual ira actualizando la maquina de estados (StateChar) y la maquina de estados de bluetooth/USB, también actualizara la velocidad de los motores, la posición del servo, el reset del WatchDog y las funciones del servidor Web.

```
while(1)
{
    MaquinaEstados();
    MaquinaEstadosBluetooth();
    MotorDerecha(VelDer);
    MotorIzquierda(VelIzq);
    setServo(Posserv);
    timer_poll ();
    main_TcpNet ();
    WDT_Feed();
}
```

## 2.2. PWM (Motores y Servo).

Para que sea posible el movimiento de los dos motores son necesarias dos señales PWM. Éstas son generadas en la tarjeta de desarrollo utilizada. En nuestra practica haremos uso del PWM1.2 y PWM1.3.

También haremos uso de otra señal PWM para controlar la posición del servomotor en nuestra practica haremos uso del PWM1.4

En el presente apartado se tratará tanto la generación de la señal PWM para los motores y el servo como el uso del puente en H para controlar a los motores.

En nuestro fichero tendremos las variables globales que indican la velocidad de los motores y la posición del servo .

Tendremos un función configPWM que configurara los motore y el servo.

```
/* *****
* Function Name : configPWM
* Description : Configura una señal PWM de 15ms de periodo por P1.20
* Input : None
* Output : None
* Return : None
* Attention : None
***** */
void configPWM(void) {

    LPC_PINCON->PINSEL7|=(3<<18); // P3.25 salida PWM (PWM1.2) y LED 1
    LPC_PINCON->PINSEL7|=(3<<20); // P3.26 salida PWM (PWM1.3) y LED 2
    LPC_PINCON->PINSEL3|=(2<<14); // P1.23 como salida PWM para servo (PWM1.4)
    LPC_SC->PCONP|=(1<<6);
    LPC_PWM1->MR0=Fpclk*Tpwm-1;
    LPC_PWM1->PCR|=(3<<10); //configurado el ENA2 (1.2),ENA3 (1.3),ENA4 (1.4)
    LPC_PWM1->PCR|=(1<<12);
    LPC_PWM1->MCR|=(1<<1);
    LPC_PWM1->TCR|=(1<<0) | (1<<3);
}
```

Para controlar el sentido de giro del servomotor hemos utilizado un puente en H, donde alimentamos ENA con la señal PWM de un motor y ENB con la señal PWM del otro motor. Dichas señales PWM indicaran la velocidad a la que se mueve el vehículo. Para las entradas IN1, IN2 utilizamos los puertos 1.20 y 1.21 para indicar el sentido de un motor, IN3, IN4 utilizamos los puertos 1.24 y 1.25 para el sentido del otro motor.

Esto lo podemos ver en las funciones MotorIzquierda y MotorDerecha y la función PuenteH.

```

/*****
* Function Name   : PuenteH
* Description     : Configura Señales del puente en en H y los LED
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PuenteH (void)
{
    LPC_GPIO1->FIODIR |= (1<<20);
    LPC_GPIO1->FIODIR |= (1<<21);
    LPC_GPIO1->FIODIR |= (1<<24);
    LPC_GPIO1->FIODIR |= (1<<25);
}

/*****
* Function Name   : MotorDerecha
* Description     : Genera el nivel alto con un PWC que representa la velocidad de las ruedas
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void MotorDerecha(int8_t VelDerecha) // LED1 para la rueda Izquierda
{
    LPC_PWM1->MR2=((Fpclk*Tpwm-1)*abs(VelDerecha)/100); // TH PWM1.2
    LPC_PWM1->LER|=(1<<2)|(1<<0); //le pasamos un 3 porque usamos el MR2 y MR0
    Dato_motorDer();
    if (VelDerecha > 0){
        LPC_GPIO1->FIOCLR |= (1<<20); // Low el pin P1.20
        LPC_GPIO1->FIOSET |= (1<<21); // Hight el pin P1.21
    }
    else{
        LPC_GPIO1->FIOSET |= (1<<20); // Hight el pin P1.20
        LPC_GPIO1->FIOCLR |= (1<<21); // Low el pin P1.21
    }
}

/*****
* Function Name   : MotorIzquierda
* Description     : Genera el nivel alto con un PWC que representa la velocidad de las ruedas
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void MotorIzquierda(int8_t VelIzquierda) // LED2 para la rueda izquierda
{
    LPC_PWM1->MR3=((Fpclk*Tpwm-1)*abs(VelIzquierda)/100); // TH PWM1.3
    LPC_PWM1->LER|=(1<<3)|(1<<0); //le pasamos un 4 porque usamos el MR3 y MR0
    Dato_motorIzq();
    if (VelIzquierda > 0){
        LPC_GPIO1->FIOCLR |= (1<<25); // Low el pin P1.24
        LPC_GPIO1->FIOSET |= (1<<24); // Hight el pin P1.25
    }
    else{
        LPC_GPIO1->FIOSET |= (1<<25); // Hight el pin P1.24
        LPC_GPIO1->FIOCLR |= (1<<24); // Low el pin P1.25
    }
}

```

También tendremos la función `setServo` que actualizará la posición del servomotor.

```

/*****
* Function Name   : setServo
* Description     : Actualiza el valor de la señal PWM
* Input          : grados - Debe tomar valores de -90 a 90
* Output         : None
* Return         : None
* Attention      : None
*****/
void setServo(int8_t grados) {
    LPC_PWM1->MR4=(Fpclk*1.5e-3 + Fpclk*1e-3*grados/90); // TH
    LPC_PWM1->LER|=(1<<4) | (1<<0);
    Dato_motorServo();
}

```

Como podemos observar las tres funciones contienen una función `Dato_X()` estas funciones servirán para actualizar en pantalla los valores de los motores y la posición del servo.

### 2.3. TP\_Simple.

En este módulo contendremos todas las funciones con las que pintamos la pantalla y definimos las zonas para la pantalla.

Como variables tendremos declaradas las variables de las velocidades de los motores y las posición del servo, la variable de distancia del sensor de ultrasonidos y la variable de estado. También tenemos las variables del mando nunchuck. Las variables de los Sensores LDR, las del servidor web para poder imprimir la ip y un conjunto de variables auxiliares.

```

#define MY_IP localm[NETIF_ETH].IpAdr
extern LOCALM localm[];

/* Variable que contiene el dato del programa */
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;
extern int distancia;
extern uint8_t Estado;

//Variables mando nunchuck
extern int8_t x; // coordenada x
extern int8_t y; // coordenada y
extern int8_t acelerometro_x;
extern int8_t acelerometro_y;
extern int8_t acelerometro_z;
extern bool flagContadorC;
extern bool flagContadorZ;

//Variables sensores
uint32_t SLuz1 = 0;
uint32_t SLuz2 = 0;
uint32_t SPromedio = 0;
//Variables auxiliares
uint8_t messageText[25+1] = {"Esto es una prueba 1000"};
int8_t aux1,aux2,aux3, aux4, aux5,aux6,aux7,aux8,aux9,aux10,aux11,aux12,aux13,aux14;
bool dep_nunchuck= true;

```

También como hemos indicado definimos las siguientes zonas de la pantalla para los modos manual, automático y Depuración

```
/* Definición de las diferentes zonas de la pantalla */
//Modo MAnual & Automático
struct t_screenZone Cabecera = { 10, 20, 220, 40, 0}; /*Mensaje "Bienvenida "*/
struct t_screenZone subCabecera = { 10, 60, 220, 40, 0}; /*Elija el modo*/
struct t_screenZone Manual = { 10, 100, 220, 40, 0}; /*Modo Manual*/
struct t_screenZone Automatico = { 10, 140, 220, 40, 0}; /*Modo Automatico*/
struct t_screenZone Depuracion = { 10, 180, 220, 40, 0}; /*Modo Depuracion*/
struct t_screenZone Infor = { 10, 60, 220, 30, 0}; /*Informacion*/
struct t_screenZone VDer = { 10, 90, 150, 30, 0}; /*Vel Derecha*/
struct t_screenZone DatDer = { 160, 90, 70, 30, 0}; /*Datos Vel Derecha*/
struct t_screenZone VIZq = { 10, 120, 150, 30, 0}; /*Vel Izquierda */
struct t_screenZone DatIzq = { 160, 120, 70, 30, 0}; /*Datos Vel Izquierda */
struct t_screenZone PServ = { 10, 150, 150, 30, 0}; /*Posicion Servo */
struct t_screenZone DatServ = { 160, 150, 70, 30, 0}; /*Datos posicion servo */
struct t_screenZone Dist = { 10, 180, 150, 30, 0}; /*Umbral Distancia */
struct t_screenZone DatDist = { 160, 180, 70, 30, 0}; /*Datos Umbral Distancia */
struct t_screenZone Luz1 = { 10, 210, 150, 30, 0}; /*Intensidad Luz sensor 1 */
struct t_screenZone DatLuz1 = { 160, 210, 70, 30, 0}; /*Datos Intensidad Luz sensor 1 */
struct t_screenZone Luz2 = { 10, 240, 150, 30, 0}; /*Intensidad Luz sensor 2 */
struct t_screenZone DatLuz2 = { 160, 240, 70, 30, 0}; /*Datos Intensidad Luz sensor 2 */
struct t_screenZone Promedio = { 10, 270, 150, 30, 0}; /*Promedio Intensidad Luz */
struct t_screenZone DatPromedio = { 160, 270, 70, 30, 0}; /*Datos Promedio */

//zona modo Depuracion
struct t_screenZone DVDer = { 10, 90, 90, 30, 0}; /*Vel Derecha*/
struct t_screenZone DDatDer = { 100, 90, 40, 30, 0}; /*Datos Vel Derecha*/
struct t_screenZone zone_1 = { 140, 90, 45, 30, 0}; /*Incrementar Vel Derecha*/
struct t_screenZone zone_2 = { 185, 90, 45, 30, 0}; /*Decrementar Vel Derecha*/
struct t_screenZone DVIzq = { 10, 120, 90, 30, 0}; /*Vel Izquierda */
struct t_screenZone DDatIzq = { 100, 120, 40, 30, 0}; /*Datos Vel Izquierda */
struct t_screenZone zone_3 = { 140, 120, 45, 30, 0}; /*Incrementar Vel Izquierda*/
struct t_screenZone zone_4 = { 185, 120, 45, 30, 0}; /*Decrementar Vel Izquierda*/
struct t_screenZone DPServ = { 10, 150, 90, 30, 0}; /*Posicion Servo */
struct t_screenZone DDatServ = { 100, 150, 40, 30, 0}; /*Datos posicion servo */
struct t_screenZone zone_5 = { 140, 150, 45, 30, 0}; /*Mover serv Derecha*/
struct t_screenZone zone_6 = { 185, 150, 45, 30, 0}; /*Mover servo Izquierda*/
struct t_screenZone DDist = { 10, 180, 170, 30, 0}; /*Umbral Distancia */
struct t_screenZone DDatDist = { 180, 180, 50, 30, 0}; /*Datos Umbral Distancia */
struct t_screenZone DLuz1 = { 10, 210, 170, 30, 0}; /*Intensidad Luz sensor 1 */
struct t_screenZone DDatLuz1 = { 180, 210, 50, 30, 0}; /*Datos Intensidad Luz sensor 1 */
struct t_screenZone DLuz2 = { 10, 240, 170, 30, 0}; /*Intensidad Luz sensor 2 */
struct t_screenZone DDatLuz2 = { 180, 240, 50, 30, 0}; /*Datos Intensidad Luz sensor 2 */
struct t_screenZone DPromedio = { 10, 270, 170, 30, 0}; /*Promedio Intensidad Luz */
struct t_screenZone DDatPromedio = { 180, 270, 50, 30, 0}; /*Datos Promedio */
struct t_screenZone IP = { 10, 60, 70, 30, 0}; /*Ip */
struct t_screenZone DatIP = { 80, 60, 150, 30, 0}; /*Valor IP */
```

También tenemos las funciones que pintan cuadrados, el signo menos y el signo mas.

```
/******
* Function Name : squareButton
* Description : Dibuja un cuadrado en las coordenadas especificadas colocando
* un texto en el centro del recuadro
* Input : zone: zone struct
* text: texto a representar en el cuadro
* textColor: color del texto
* lineColor: color de la línea
* Output : None
* Return : None
* Attention : None
*****/
void squareButton(struct t_screenZone* zone, char * text, uint16_t textColor, uint16_t lineColor)
{
    LCD_DrawLine( zone->x, zone->y, zone->x + zone->size_x, zone->y, lineColor);
    LCD_DrawLine( zone->x, zone->y, zone->x, zone->y + zone->size_y, lineColor);
    LCD_DrawLine( zone->x, zone->y + zone->size_y, zone->x + zone->size_x, zone->y + zone->size_y, lineColor);
    LCD_DrawLine( zone->x + zone->size_x, zone->y, zone->x + zone->size_x, zone->y + zone->size_y, lineColor);
    GUI_Text(zone->x + zone->size_x/2 - (strlen(text)/2)*8, zone->y + zone->size_y/2 - 8,
            (uint8_t*) text, textColor, Black);
}
```

```

/*****
* Function Name   : drawMinus
* Description     : Draw a minus sign in the center of the zone
* Input          : zone: zone struct
*                : lineColor
* Output         : None
* Return         : None
* Attention      : None
*****/
void drawMinus(struct t_screenZone* zone, uint16_t lineColor)
{
    LCD_DrawLine( zone->x + 5 , zone->y + zone->size_y/2 - 1,
                  zone->x + zone->size_x-5, zone->y + zone->size_y/2 - 1,
                  lineColor);
    LCD_DrawLine( zone->x + 5 , zone->y + zone->size_y/2,
                  zone->x + zone->size_x-5, zone->y + zone->size_y/2,
                  lineColor);
    LCD_DrawLine( zone->x + 5 , zone->y + zone->size_y/2 + 1,
                  zone->x + zone->size_x-5, zone->y + zone->size_y/2 + 1,
                  lineColor);
}
/*****
* Function Name   : drawMinus
* Description     : Draw a minus sign in the center of the zone
* Input          : zone: zone struct
*                : lineColor
* Output         : None
* Return         : None
* Attention      : None
*****/
void drawAdd(struct t_screenZone* zone, uint16_t lineColor)
{
    drawMinus(zone, lineColor);

    LCD_DrawLine( zone->x + zone->size_x/2 - 1, zone->y + 5 ,
                  zone->x + zone->size_x/2 - 1, zone->y + zone->size_y - 5,
                  lineColor);
    LCD_DrawLine( zone->x + zone->size_x/2 , zone->y + 5 ,
                  zone->x + zone->size_x/2 , zone->y + zone->size_y - 5,
                  lineColor);
    LCD_DrawLine( zone->x + zone->size_x/2 + 1, zone->y + 5 ,
                  zone->x + zone->size_x/2 + 1, zone->y + zone->size_y - 5,
                  lineColor);
}

```

En este modulo también tenemos las funciones que pintan la Pantalla de inicio, Pantalla de modo manual, Pantalla del modo automático, Pantalla del modo Depuración y Pantalla del modo Depuración que muestra los valores del Nunchuck.

```

/*****
* Function Name   : PantallaInicio
* Description     : Visualiza el menu de inicio
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PantallaInicio(void)
{
    squareButton(&Cabecera, "BIENVENIDO USUARIO", White, Blue);
    squareButton(&subCabecera, "Eliga el modo:", White, Blue);
    squareButton(&Manual, "Manual", White, Blue);
    squareButton(&Automatico, "Automatico", White, Blue);
    squareButton(&Depuracion, "Depuracion", White, Blue);
}

```



```

/*****
* Function Name   : PantallaManual
* Description     : Visualiza la informacion a mostra en el modo manual
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PantallaManual(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(&Cabecera, "MODO MANUAL", White, Blue);
    squareButton(&Infor, "Datos:", White, Blue);
    squareButton(&VDer, "Vel Derecha", White, Blue);
    squareButton(&DatDer, " ", White, Blue);
    squareButton(&VIzq, "Vel Izquierda", White, Blue);
    squareButton(&DatIzq, " ", White, Blue);
    squareButton(&PServ, "Pos. Servo", White, Blue);
    squareButton(&DatServ, " ", White, Blue);
    squareButton(&Dist, "Umb.Distancia", White, Blue);
    squareButton(&DatDist, " ", White, Blue);
    squareButton(&Luz1, "Intensidad Luz1", White, Blue);
    squareButton(&DatLuz1, " ", White, Blue);
    squareButton(&Luz2, "Intensidad Luz2", White, Blue);
    squareButton(&DatLuz2, " ", White, Blue);
    squareButton(&Promedio, "Prom. Intensidad", White, Blue);
    squareButton(&DatPromedio, " ", White, Blue);
    //Dibujamos los Valores de los parametros
    sprintf(texto1,"%2d", VelDer);
    sprintf(texto2,"%2d", VelIzq);
    sprintf(texto3,"%2d", Posserv);
    sprintf(texto4,"%2d", distancia);
    sprintf(texto5,"%2d", SLuz1);
    sprintf(texto6,"%2d", SLuz2);
    SPromedio= (SLuz1+SLuz2)/2;
    sprintf(texto7,"%2d", SPromedio);

    GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
        (uint8_t*) texto1, White, Black);
    GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
        (uint8_t*) texto2, White, Black);
    GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
        (uint8_t*) texto3, White, Black);
    GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
        (uint8_t*) texto4, White, Black);
    GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
    GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
    GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
        (uint8_t*) texto7, White, Black);
}

/*****
* Function Name   : PantallaAutomatico
* Description     : Visualiza la informacion a mostra en el modo automatico
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PantallaAutomatico(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(&Cabecera, "MODO AUTOMATICO", White, Blue);
    squareButton(&Infor, "Datos:", White, Blue);
    squareButton(&VDer, "Vel Derecha", White, Blue);
    squareButton(&DatDer, " ", White, Blue);
    squareButton(&VIzq, "Vel Izquierda", White, Blue);
    squareButton(&DatIzq, " ", White, Blue);
    squareButton(&PServ, "Pos. Servo", White, Blue);
    squareButton(&DatServ, " ", White, Blue);
    squareButton(&Dist, "Umb.Distancia", White, Blue);
    squareButton(&DatDist, " ", White, Blue);
    squareButton(&Luz1, "Intensidad Luz1", White, Blue);
    squareButton(&DatLuz1, " ", White, Blue);
    squareButton(&Luz2, "Intensidad Luz2", White, Blue);
    squareButton(&DatLuz2, " ", White, Blue);
    squareButton(&Promedio, "Prom. Intensidad", White, Blue);
    squareButton(&DatPromedio, " ", White, Blue);
    //Dibujamos los Valores de los parametros
    sprintf(texto1,"%2d", VelDer);
    sprintf(texto2,"%2d", VelIzq);
    sprintf(texto3,"%2d", Posserv);
    sprintf(texto4,"%2d", distancia);
    sprintf(texto5,"%2d", SLuz1);
    sprintf(texto6,"%2d", SLuz2);
    SPromedio= (SLuz1+SLuz2)/2;
    sprintf(texto7,"%2d", SPromedio);

    GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
        (uint8_t*) texto1, White, Black);
    GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
        (uint8_t*) texto2, White, Black);
    GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
        (uint8_t*) texto3, White, Black);
    GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
        (uint8_t*) texto4, White, Black);
    GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
    GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
    GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
        (uint8_t*) texto7, White, Black);
}

```

```

/*****
* Function Name : PantallaDepuracion
* Description : Visualiza la informacion a mostra en el modo depuracion
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PantallaDepuracion(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(sCabecera, "MODULO DEPURACION", White, Blue);
    squareButton(sIP, "Dir. IP:", White, Blue);
    squareButton(sDatIP, " ", White, Blue);
    squareButton(sDVDer, "V.Derecha", White, Blue);
    squareButton(sDDatDer, " ", White, Blue);
    squareButton(sDVIsq, "V.Isquierda", White, Blue);
    squareButton(sDDatIsq, " ", White, Blue);
    squareButton(sDPServ, "Pos.Servo", White, Blue);
    squareButton(sDDatServ, " ", White, Blue);
    squareButton(sDDist, "Umb.Distancia", White, Blue);
    squareButton(sDDatDist, " ", White, Blue);
    squareButton(sDLus1, "Intensidad Lus1", White, Blue);
    squareButton(sDDatLus1, " ", White, Blue);
    squareButton(sDLus2, "Intensidad Lus2", White, Blue);
    squareButton(sDDatLus2, " ", White, Blue);
    squareButton(sDPromedio, "Promedio Int.", White, Blue);
    squareButton(sDDatPromedio, " ", White, Blue);
    squareButton(sone_7, " ", White, Blue);
    drawAdd(sone_1, White);
    drawAdd(sone_3, White);
    drawAdd(sone_5, White);
    drawMinus(sone_2, White);
    drawMinus(sone_4, White);
    drawMinus(sone_6, White);
    //Dibujamos los Valores de los parametros
    sprintf(texto1, "%2d", VelDer);
    sprintf(texto2, "%2d", VelIsq);
    sprintf(texto3, "%2d", PosServ);
    sprintf(texto4, "%2d", distancia);
    sprintf(texto5, "%2d", SLus1);
    sprintf(texto6, "%2d", SLus2);
    SPromedio = (SLus1 + SLus2) / 2;
    sprintf(texto7, "%2d", SPromedio);

    GUI_Text(DDatDer.x + DDatDer.size_x/2 - (strlen(texto1)/2)*8, DDatDer.y + DDatDer.size_y/2 - 8,
        (uint8_t*) texto1, White, Black);
    GUI_Text(DDatIsq.x + DDatIsq.size_x/2 - (strlen(texto2)/2)*8, DDatIsq.y + DDatIsq.size_y/2 - 8,
        (uint8_t*) texto2, White, Black);
    GUI_Text(DDatServ.x + DDatServ.size_x/2 - (strlen(texto3)/2)*8, DDatServ.y + DDatServ.size_y/2 - 8,
        (uint8_t*) texto3, White, Black);
    GUI_Text(DDatDist.x + DDatDist.size_x/2 - (strlen(texto4)/2)*8, DDatDist.y + DDatDist.size_y/2 - 8,
        (uint8_t*) texto4, White, Black);
    GUI_Text(DDatLus1.x + DDatLus1.size_x/2 - (strlen(texto5)/2)*8, DDatLus1.y + DDatLus1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
    GUI_Text(DDatLus2.x + DDatLus2.size_x/2 - (strlen(texto6)/2)*8, DDatLus2.y + DDatLus2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
    GUI_Text(DDatPromedio.x + DDatPromedio.size_x/2 - (strlen(texto7)/2)*8, DDatPromedio.y + DDatPromedio.size_y/2 - 8,
        (uint8_t*) texto7, White, Black);
}

```

```

/*****
* Function Name : PantallaDepuracion_Nunchuck
* Description : Visualiza la informacion de los valores del Nunchuck
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PantallaDepuracion_Nunchuck(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(sCabecera, "MODO DEPURACION", White, Blue);
    squareButton(sIP, "Dir. IP:", White, Blue);
    squareButton(sDatIP, " ", White, Blue);
    squareButton(sVDer, "Eje X", White, Blue);
    squareButton(sDatDer, " ", White, Blue);
    squareButton(sVIsq, "Eje Y", White, Blue);
    squareButton(sDatIsq, " ", White, Blue);
    squareButton(sPServ, "Acelerometro X", White, Blue);
    squareButton(sDatServ, " ", White, Blue);
    squareButton(sDist, "Acelerometro Y", White, Blue);
    squareButton(sDatDist, " ", White, Blue);
    squareButton(sLus1, "Acelerometro Z", White, Blue);
    squareButton(sDatLus1, " ", White, Blue);
    squareButton(sLus2, "Boton C", White, Blue);
    squareButton(sDatLus2, " ", White, Blue);
    squareButton(sPromedio, "Boton Z", White, Blue);
    squareButton(sDatPromedio, " ", White, Blue);

    //Dibujamos los Valores de los parametros
    sprintf(texto1, "%4d", x);
    sprintf(texto2, "%4d", y);
    sprintf(texto3, "%4d", acelerometro_x);
    sprintf(texto4, "%4d", acelerometro_y);
    sprintf(texto5, "%4d", acelerometro_z);
    sprintf(texto6, "%4d", flagContadorC);
    sprintf(texto7, "%4d", flagContadorZ);

    GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
        (uint8_t*) texto1, White, Black);
    GUI_Text(DatIsq.x + DatIsq.size_x/2 - (strlen(texto2)/2)*8, DatIsq.y + DatIsq.size_y/2 - 8,
        (uint8_t*) texto2, White, Black);
    GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
        (uint8_t*) texto3, White, Black);
    GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DDatDist.size_y/2 - 8,
        (uint8_t*) texto4, White, Black);
    GUI_Text(DatLus1.x + DatLus1.size_x/2 - (strlen(texto5)/2)*8, DatLus1.y + DatLus1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
    GUI_Text(DatLus2.x + DatLus2.size_x/2 - (strlen(texto6)/2)*8, DatLus2.y + DatLus2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
    GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
        (uint8_t*) texto7, White, Black);
}

```



En este modulo también tendremos una función que chequea si se presiona una zona de la pantalla.

```

/*****
* Function Name   : checkTouchPanel
* Description     : Lee el TouchPanel y almacena las coordenadas si detecta pulsación
* Input          : None
* Output         : Modifica processedTouchPanel
*                0 = si no se detecta pulsación
*                1 = si se detecta pulsación
*                En este caso se actualizan las coordenadas en la estructura display
* Return         : None
* Attention      : None
*****/
void checkTouchPanel(void)
{
    Coordinate* coord;

    coord = Read_Ads7846();

    if (coord > 0) {
        getDisplayPoint(&display, coord, &matrix );
        processedTouchPanel = 1;
    }
    else
    {
        processedTouchPanel = 0;

        // Esto es necesario hacerlo si hay dos zonas diferentes en
        // dos pantallas secuenciales que se solapan
        zone_1.processed = 1;
        zone_2.processed = 1;
        zone_3.processed = 1;
        zone_4.processed = 1;
        zone_5.processed = 1;
        zone_6.processed = 1;
        zone_7.processed = 1;
        Cabeecera.processed = 1;
        subCabeecera.processed = 1;
        Manual.processed = 1;
        Automatic.processed = 1;
        Depuration.processed = 1;
        Infor.processed = 1;
        VDer.processed = 1;
        DetDer.processed = 1;
        VInq.processed = 1;
        DetInq.processed = 1;
        PServ.processed = 1;
        DetServ.processed = 1;
        Dist.processed = 1;
        DetDist.processed = 1;
        Iux1.processed = 1;
        DetIux1.processed = 1;
        Iux2.processed = 1;
        DetIux2.processed = 1;
        Promedio.processed = 1;
        DetPromedio.processed = 1;
        DVDer.processed = 1;
        DDetDer.processed = 1;
        DVInq.processed = 1;
        DDetInq.processed = 1;
        DPServ.processed = 1;
        DDetServ.processed = 1;
        DDist.processed = 1;
        DDetDist.processed = 1;
        DIux1.processed = 1;
        DDetIux1.processed = 1;
        DIux2.processed = 1;
        DDetIux2.processed = 1;
        DPromedio.processed = 1;
        DDetPromedio.processed = 1;
        IP.processed = 1;
        DetIP.processed = 1;
    }
}

```

Este modulo también contendrá funciones que van pintando los datos de las variables de los motores, sensores y del mando nunchuck.

```

/*****
* Function Name   : Dato_motorDer
* Description     : Imprime el dato del motor derecho
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_motorDer(void)
{
    if (Estado== 3 || Estado==5)
    {
        if (aux1!=VelDer)
            squareButton(sDatDer, " ", White, Blue);
        sprintf(texto1,"%2d", VelDer);
        GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
            (uint8_t*) texto1, White, Black);
        aux1=VelDer;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if (aux1!=VelDer)
            squareButton(sDDatDer, " ", White, Blue);
        sprintf(texto1,"%2d", VelDer);
        GUI_Text(DDatDer.x + DDatDer.size_x/2 - (strlen(texto1)/2)*8, DDatDer.y + DDatDer.size_y/2 - 8,
            (uint8_t*) texto1, White, Black);
        aux1=VelDer;
    }
}
/*****
* Function Name   : Dato_motorIsq
* Description     : Imprime el dato del motor izquierdo
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_motorIsq(void)
{
    if (Estado== 3 || Estado==5)
    {
        if (aux2!=VelIsq)
            squareButton(sDatIsq, " ", White, Blue);
        sprintf(texto2,"%2d", VelIsq);
        GUI_Text(DatIsq.x + DatIsq.size_x/2 - (strlen(texto2)/2)*8, DatIsq.y + DatIsq.size_y/2 - 8,
            (uint8_t*) texto2, White, Black);
        aux2=VelIsq;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if (aux2!=VelIsq)
            squareButton(sDDatIsq, " ", White, Blue);
        sprintf(texto2,"%2d", VelIsq);
        GUI_Text(DDatIsq.x + DDatIsq.size_x/2 - (strlen(texto2)/2)*8, DDatIsq.y + DDatIsq.size_y/2 - 8,
            (uint8_t*) texto2, White, Black);
        aux2=VelIsq;
    }
}

```

```

/*****
* Function Name   : Dato_motorServo
* Description     : Imprime el dato del motor servo
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_motorServo(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux3!=Posserv)
            squareButton(&DatServ, "    ", White, Blue);
        sprintf(texto3,"%2d",Posserv);
        GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
            (uint8_t*) texto3, White, Black);
        aux3=Posserv;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux3!=Posserv)
            squareButton(&DDatServ, "    ", White, Blue);
        sprintf(texto3,"%2d", Posserv);
        GUI_Text(DDatServ.x + DDatServ.size_x/2 - (strlen(texto3)/2)*8, DDatServ.y + DDatServ.size_y/2 - 8,
            (uint8_t*) texto3, White, Black);

        aux3=Posserv;
    }
}

/*****
* Function Name   : Dato_Distanciadm
* Description     : Imprime el dato del sensor ultrasonidos
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_Distanciadm(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux4!=distancia)
            squareButton(&DatDist, "    ", White, Blue);
        if (distancia>400)
            sprintf(texto4,"max");
        else
            sprintf(texto4,"%4d",distancia);
        GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
            (uint8_t*) texto4, White, Black);
        aux4=distancia;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux4!=distancia)
            squareButton(&DDatDist, "    ", White, Blue);
        if (distancia>400)
            sprintf(texto4,"max");
        else
            sprintf(texto4,"%4d", distancia);
        GUI_Text(DDatDist.x + DDatDist.size_x/2 - (strlen(texto4)/2)*8, DDatDist.y + DDatDist.size_y/2 - 8,
            (uint8_t*) texto4, White, Black);

        aux4=distancia;
    }
}

```

```

/*****
* Function Name   : Dato_sensorLus1
* Description     : Imprime el dato del sensor lus1
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_sensorLus1(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux5!=SLus1)
            squareButton(sDatLus1, "    ", White, Blue);
        sprintf(texto5,"%3d",SLus1);
        GUI_Text(DatLus1.x + DatLus1.size_x/2 - (strlen(texto5)/2)*8, DatLus1.y + DatLus1.size_y/2 - 8,
            (uint8_t*) texto5, White, Black);
        aux5=SLus1;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux5!=SLus1)
            squareButton(sDDatLus1, "    ", White, Blue);
        sprintf(texto5,"%3d", SLus1);
        GUI_Text(DDatLus1.x + DDatLus1.size_x/2 - (strlen(texto5)/2)*8, DDatLus1.y + DDatLus1.size_y/2 - 8,
            (uint8_t*) texto5, White, Black);

        aux5=SLus1;
    }
}

/*****
* Function Name   : Dato_sensorLus2
* Description     : Imprime el dato de lus 2
* Input          :
* Output         :
* Return         :
* Attention      : None
*****/
void Dato_sensorLus2(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux6!=SLus2)
            squareButton(sDatLus2, "    ", White, Blue);
        sprintf(texto6,"%3d",SLus2);
        GUI_Text(DatLus2.x + DatLus2.size_x/2 - (strlen(texto6)/2)*8, DatLus2.y + DatLus2.size_y/2 - 8,
            (uint8_t*) texto6, White, Black);
        aux6=SLus2;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux6!=SLus2)
            squareButton(sDDatLus2, "    ", White, Blue);
        sprintf(texto6,"%3d", SLus2);
        GUI_Text(DDatLus2.x + DDatLus2.size_x/2 - (strlen(texto6)/2)*8, DDatLus2.y + DDatLus2.size_y/2 - 8,
            (uint8_t*) texto6, White, Black);

        aux6=SLus2;
    }
}
}

```

```

/*****
* Function Name : Dato_promedioSensores
* Description : Imprime el dato promedio de los sensores de lus
* Input :
* Output :
* Return :
* Attention : None
*****/
void Dato_promedioSensores(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux7!=SPromedio)
            squareButton(sDatPromedio, " ", White, Blue);
        sprintf(texto7,"%3d",SPromedio);
        GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
            (uint8_t*) texto7, White, Black);
        aux7=SPromedio;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux7!=SPromedio)
            squareButton(sDDatPromedio, " ", White, Blue);
        sprintf(texto7,"%3d", SPromedio);
        GUI_Text(DDatPromedio.x + DDatPromedio.size_x/2 - (strlen(texto7)/2)*8, DDatPromedio.y + DDatPromedio.size_y/2 - 8,
            (uint8_t*) texto7, White, Black);

        aux7=SPromedio;
    }
}

/*****
* Function Name : screenMessageIP
* Description : Visualiza la pantalla de mensajes
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void screenMessageIP(void)
{
    sprintf((char *)messageText, " %d.%d.%d.%d ", MY_IP[0], MY_IP[1],
                                                MY_IP[2], MY_IP[3]);
    squareButton(sDatIP, (char*)messageText, Red , Blue);
}

```

También tenemos otra función que actualiza los datos faltantes en el modo depuración que sería los del mando nunchuck y también sirve para incrementar y decrementar los valores de los motores y del servo.

```

/*****
 * Function Name : modoDepuracion
 * Description : Incrementa valores servo en pantalla
 * Input :
 * Output :
 * Return :
 * Attention : None
 *****/
void modoDepuracion(void)
{
    //Cambio para visualizar datos del nunchuck
    if(sonePressed($Cabecera))
    {
        Estado= 6;
    }
    //Actualizamos datos del nunchuck
    if(dep_nunchuck== true)
    {
        if(aux8!=x)
        {
            squareButton($DatDer, " ", White, Blue);
            sprintf(texto1,"%4d",x);
            GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
                    (uint8_t*) texto1, White, Black);
            aux8=x;
        }

        if(aux9!=y)
        {
            squareButton($DatIsq, " ", White, Blue);
            sprintf(texto2,"%4d",y);
            GUI_Text(DatIsq.x + DatIsq.size_x/2 - (strlen(texto2)/2)*8, DatIsq.y + DatIsq.size_y/2 - 8,
                    (uint8_t*) texto2, White, Black);
            aux9=y;
        }

        if(aux10!=acelerometro_x)
        {
            squareButton($DatServ, " ", White, Blue);
            sprintf(texto3,"%4d",acelerometro_x);
            GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
                    (uint8_t*) texto3, White, Black);
            aux10=acelerometro_x;
        }

        if(aux11!=acelerometro_y)
        {
            squareButton($DatDist, " ", White, Blue);
            sprintf(texto4,"%4d",acelerometro_y);
            GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
                    (uint8_t*) texto4, White, Black);
            aux11=acelerometro_y;
        }

        if(aux12!=acelerometro_s)
        {
            squareButton($DatLus1, " ", White, Blue);
            sprintf(texto5,"%4d",acelerometro_s);
            GUI_Text(DatLus1.x + DatLus1.size_x/2 - (strlen(texto5)/2)*8, DatLus1.y + DatLus1.size_y/2 - 8,
                    (uint8_t*) texto5, White, Black);
            aux12=acelerometro_s;
        }

        if(aux13!=flagContadorC)
        {
            squareButton($DatLus2, " ", White, Blue);
            sprintf(texto6,"%4d",flagContadorC);
            GUI_Text(DatLus2.x + DatLus2.size_x/2 - (strlen(texto6)/2)*8, DatLus2.y + DatLus2.size_y/2 - 8,
                    (uint8_t*) texto6, White, Black);
            aux13=flagContadorC;
        }

        if(aux14!=flagContadorZ)
        {
            squareButton($DatPromedio, " ", White, Blue);
            sprintf(texto7,"%4d",flagContadorZ);
            GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
                    (uint8_t*) texto7, White, Black);
            aux14=flagContadorZ;
        }
    }
    else
    {
        if (sonePressed($sone_1))
            VelDer++;
        if (sonePressed($sone_3))
            VelIsq++;
        if (sonePressed($sone_5))
            Posserv++;
        if (sonePressed($sone_2))
            VelDer--;
        if (sonePressed($sone_4))
            VelIsq--;
        if (sonePressed($sone_6))
            Posserv--;
        Dato_motorDer();
        Dato_motorIsq();
        Dato_motorServo();
    }
}

```

Por ultimo tenemos la función que detecta si se esta pulsando la pantalla.

```

/*****
* Function Name : sonePressed
* Description : Detecta si se ha producido una pulsación en una sone contreta
* Input : sone: Estructura con la información de la sone
* Output : Modifica sone->pressed
*          0 - si no se detecta pulsación en la sone
*          1 - si se detecta pulsación en la sone
* Return : 0 - si no se detecta pulsación en la sone
*          1 - si se detecta pulsación en la sone
* Attention : None
*****/
int8_t sonePressed(struct t_screenZone* sone)
{
    if (pressedTouchPanel == 1) {
        if ((display.x > sone->x) && (display.x < sone->x + sone->size_x) &&
            (display.y > sone->y) && (display.y < sone->y + sone->size_y))
        {
            sone->pressed = 1;
            return 1;
        }
    }

    sone->pressed = 0;
    return 0;
}

```

## 2.4. Nunchuck.

Como se indica en la descripción utilizaremos el mando Nunchuck para mover el vehículo en el modo manual. Para este modulo de nuestro utilizaremos las variables globales utilizadas previamente que indican la velocidad de los motores y la posición del servo, también declararemos las variables que nos proporcionan las lecturas del mando y dos variables eje x y eje y que utilizaremos para poder calibrar las lecturas del joystick del mando.

```

// Variable de velocidad de la ruedas
extern uint8_t VelDer;
extern uint8_t VelIzq;
extern int8_t Posserv;
int ejex, ejey;

//Variable de datos del mando
int8_t x; // coordenada x
int8_t y; // coordenada y
int8_t acelerometro_x, acelerometro_y, acelerometro_z;
int8_t resto;
extern bool flagContadorC, flagContadorZ, flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores;
extern int contadorZ, contadorC, contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioMas2segBotones, flagCambioMenos2segBotones;
extern int contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioPulsandoC;
extern int contadorCambioPulsandoC;
extern bool modobluetooth;

```

En nuestro modulo tenemos una función Nunchuck\_Init() que hará uso de las funciones proporcionadas por las bibliotecas de comunicación I2C que nos ha sido proporcionada y utilizaremos para realizar la comunicacion entre el mando y nuestro sistema.

```
void Nunchuck_Init() {
    I2Cdelay();
    I2Cdelay();

    // Enviar commando 0x52 para configurar el Nunchuck
    I2CSendAddr(0x52,0);      // I2C Address del Nunchuck, WRITE
    I2CSendByte(0xF0);        // Access Config Command
    I2CSendByte(0x55);        // Continuous Conversion
    I2CSendStop();

    I2Cdelay();              // Para propósitos de simulacion
    I2Cdelay();

    // Enviar commando 0xFB para arrancar conversion del Nunchuck
    I2CSendAddr(0x52,0);      // I2C Address del Nunchuck, WRITE
    I2CSendByte(0xFB);        // Start Conversion
    I2CSendByte(0x00);
    I2CSendStop();
}
```

Tambien tenemos la funcion Nunchuck\_read () que hará uso de las funciones proporcionadas por las bibliotecas de comunicación I2C que nos ha sido proporcionada y utilizaremos para obtener las lecturas del mando.

```
void Nunchuck_read() {

    // Enviar commando 0x52 para leer Nunchuck
    I2CSendAddr(0x52,0);      // I2C Address del Nunchuck, WRITE
    I2CSendByte(0x0);         // Read
    I2CSendStop();

    I2Cdelay();

    // Leer datos mando
    I2CSendAddr(0x52,1);      // I2C Address, READ
    x = I2CGetByte(0);        // Read MSB Byte, ACK : Esto es la coordenada x
    y = I2CGetByte(0);
    acelerometro_x = I2CGetByte(0);
    acelerometro_y = I2CGetByte(0);
    acelerometro_z = I2CGetByte(0);
    resto = I2CGetByte(1); //NACK

    if ((resto & 00000001) == 0x0) { // Boton Z
        flagContadorZ = true;
    }
    else {
        flagContadorZ = false;
    }
    if ((resto & 0x02) == 0x0) { // Boton C
        flagContadorC = true;
    }
    else {
        flagContadorC = false;
    }

    I2CSendStop();
}
```



También haremos uso de una función `Movimiento_Nunchuck` la cual utilizaremos para procesar los valores obtenidos y mover los motores del coche. De manera similar tenemos la función `Servo_Nunchuck` que utilizaremos para procesar los valores obtenidos y mover el servo en función de la inclinación del mando.

```

/*****
* Function Name   : MovimientoNunchuck
* Description     : Saca los valores recogidos por el Nunchuck para controlar el robot
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void Movimiento_Nunchuck()
{
    if(modobluetooth==false)
    {
        if(y<0)
        {
            ejey= (128+y)*100/127;
        }
        else if( y>= 0)
        {
            ejey= (y-128)*100/128;
        }

        if(x<0)
        {
            ejex= (x+129)*100/128;
        }
        else if( x>= 0)
        {
            ejex= (x-127)*100/127;
        }

        if (ejey==0 && ejex==0)
        {
            VelDer = 0;
            VelIzq= 0;
        }
        else if(ejey!=0 &&ejex==0)
        {
            VelDer=ejey;
            VelIzq=ejey;
        }
        else if(ejey==0 &&ejex!=0)
        {
            VelDer=-ejex;
            VelIzq=ejex;
        }
        else if(ejey>0 &&ejex!=0)
        {
            VelDer=ejey/2-ejex/2;
            VelIzq=ejey/2+ejex/2;
        }
        else if(ejey<0 &&ejex!=0)
        {
            VelDer=ejey/2+ejex/2;
            VelIzq=ejey/2-ejex/2;
        }
    }
}

```

```

/*****
* Function Name   : Servo_Nunchuck
* Description    : Mueve el servo con la inclinacion del mando
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void Servo_Nunchuck()
{
    if (acelerometro_x>0)
        Posserv=(127-acelerometro_x)*90/62;

    else
        Posserv=-(128+acelerometro_x)*90/63;
}

```

## 2.5. ADC.

En nuestro sistema haremos uso del ADC para tomar muestras del sensor de Luz y para poder realizar la grabación de audios. En este modulo contamos con la inicialización el ADC para la lectura sensores LDRs utilizando AD0.1 y AD0.5 y también tenemos otra función de inicialización en la que utilizamos el AD0.2 para grabar audios.

En este modulo tendremos declaradas unas variables int para almacenar la toma de las muestras ADC mediante los canales 1,2 y 5, la variables externas utilizadas para almacenar los valores leídos de los LDRs y la muestras de audio, tanto el mensaje de alarma como el audio grabado y una variable auxiliar que utilizamos para saber si estamos reproduciendo la alarma, el audio grabado , grabando o bien hemos finalizado.

```

#define N_muestras 15872

uint32_t canal_1, canal_2, canal_5;
extern uint32_t SLuz1,SLuz2,SPromedio;
uint8_t muestras[15872];

uint8_t audio;
int ALARMA = 1;
int GRABANDO=2;
int GRABADO=3;
int FIN =4;

void init_ADC_LDRs(void)
{
    LPC_SC->PCONP|= (1<<12);           // Power ON
    LPC_PINCON->PINSEL1|= (1<<16);      // ( AD0.1) P0.24 //LUZ1(izquierda)
    LPC_PINCON->PINSEL3|= (3<<30);      // ( AD0.5) P1.31 //LUZ2(derecha)
    LPC_PINCON->PINMODE1|= (2<<16);     // Deshabilita pullup/pulldown
    LPC_PINCON->PINMODE3|= (2<<30);     // Deshabilita pullup/pulldown
    LPC_SC->PCLKSELO|= (0<<8);          // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
    LPC_ADC->ADCR|=(1<<5)|(1<<1)        // canales 0 y 1
    LPC_ADC->ADCR|=(1<<8)                // CLKDIV=1 (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz) MAXIMA FRECUENCIA
    LPC_ADC->ADCR|=(1<<21);              // PDN=1
    LPC_ADC->ADINTEN|=(1<<5);            // Hab. interrupción fin de conversión del ÚLTIMO canal(canal 5)
    NVIC_EnableIRQ(ADC_IRQn);          //Habilita la interrupcion
    NVIC_SetPriority(ADC_IRQn,0);       //Prioridad
}

```

Nuestro modulo contiene la función init\_ADC\_LDRs que inicializa el ADC para capturar los valores de los LDRs.

Nuestro modulo contiene la función `init_ADC_grabar` que inicializa el ADC para capturar los valores del micrófono y almacenar un audio.

```
void init_ADC_grabar(void)
{
    LPC_SC->PCONP|= (1<<12);           // Power ON
    LPC_PINCON->PINSEL1|= (1<<18);      // ADC input= 0.25 (AD0.2)
    LPC_PINCON->PINMODE1|= (2<<18);     // Deshabilita pullup/pulldown*
    LPC_SC->PCLKSEL0|= (0<<8);          // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
    LPC_ADC->ADCR|= (1<<2) |            // Canal 2
                    (1<<8) |            // CLKDIV=1 (Fclk_ADC=25Mhz / (1+1)= 12.5Mhz)   MAXIMA FRECUENCIA
                    (1<<21) |           // PDN=1
                    (1<<24);            // Inicio de conversión con el Match 1 del Timer 0
    LPC_ADC->ADINTEN |= (1<<2);          // Hab. interrupción fin de conversión Canal 2
    NVIC_EnableIRQ(ADC_IRQn);           //Habilita la interrupcion
    NVIC_SetPriority(ADC_IRQn,0);
}

```

También tenemos la función que se ejecuta cuando se produce la interrupción que varia en función de si la variable audio esta en estado GRABANDO o bien toma los valores de los sensores LDRs.

```
void ADC_IRQHandler(void)
{
    static uint16_t indice_muestra;
    if (audio == GRABANDO) {
        canal_2= ((LPC_ADC->ADDR2>>8)&0xFF); // se borra automat. el flag DONE al leer ADGDR
        muestras[indice_muestra++] = canal_2; //almacenamos la muestra tomada
        LPC_ADC->ADCR|= (1<<24); // START = ON
        NVIC_DisableIRQ(ADC_IRQn);
        if (indice_muestra==N_muestras-1)
        {
            indice_muestra=0; // Reiniciamos el indice de muestras
            LPC_ADC->ADCR= (0<<24); // Paramos ADC
            init_ADC_LDRs(); // Para iniciar la configuracion del timer para las LDRs
            LPC_TIM2->TCR=0x01; // Activa timer de LDRs
            audio = FIN;
        }
    }
    else{
        LPC_ADC->ADCR&=~(1<<16);
        canal_1= ((LPC_ADC->ADDR1 >>4)&0xFFFF); // flag DONE se borra automat. al leer ADDR1
        canal_5= ((LPC_ADC->ADDR5 >>4)&0xFFFF); // flag DONE se borra automat. al leer ADDR0
        SLuz1 = ((canal_1/100)*2.5); // 12 bits -> 4096
        SLuz2 = ((canal_5/100)*2.5); // 12 bits -> 4096

        SPromedio = (SLuz1 + SLuz2)/2;
    }
}

```

Por ultimo este modulo tenemos una función que utilizamos para inicializar el TIMER1 en función de si estamos grabando o reproduciendo el audio y también es el encargado de inicializar el ADC cuando queremos grabar un audio.

```

/*****
* Function Name : Audio
* Description : En funcion de si queremos reproducir o grabar inicializa el timer 1 y el ADC
* Input : audioElegido: ALARMA - Reproduce Alarma
          GRABANDO - Graba audio
          GRABADO - Reproduce audio grabado
* Output : None
* Return : None
* Attention : None
*****/
void Audio(int Opcion) {
    if (Opcion == ALARMA) {
        audio = ALARMA;
        init_TIMER1_reproducir();
        LPC_TIM1->TCR=0x01;
    }
    if (Opcion == GRABANDO) {
        audio = GRABANDO;
        init_TIMER1_grabar();
        LPC_TIM1->TCR=0x01;
        init_ADC_grabar();
        LPC_TIM2->TCR = 0x02; //Paramos el timer2 para que no haya modo burst y que solo se convierta el canal de grabacion
    }
    if (Opcion == GRABADO) {
        audio = GRABADO;
        init_TIMER1_reproducir();
        LPC_TIM1->TCR=0x01;
    }
}

```

## 2.6. DAC.

Este modulo es utilizado para inicializar el modulo DAC y poder reproducir mediante TIMERS los audios previamente grabado o bien la alarma.

```
void init_DAC(void)
{
    LPC_PINCON->PINSEL1|= (2<<20);    // DAC output = P0.26 (AOUT)
    LPC_PINCON->PINMODE1|= (2<<20);    // Deshabilita pullup/pulldown
    LPC_DAC->DACCTRL=0;
}
```

## 2.7. Interrupciones.

Para que nuestro sistema tenga la funcionalidad de poder grabar y reproducir audio en el modo depuración mediante la pulsación de Key1 y Key2 haremos uso de dos interrupciones externas por flanco de subida, una para grabar audio y otro para reproducirlo.

En este modulo utilizaremos variables externas del estado para saber el modo de trabajo operativo, variables que nos indican el estado de los pulsadores para saber si esta solo un pulsador o bien están los dos pulsados y las variables de decisión de si grabamos o reproducimos audio.

```
extern uint8_t Estado;
extern int contadorKey1;
extern int contadorKey2;
extern bool flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores, flagContadorKey1, flagContadorKey2;

extern int ALARMA;
extern int GRABANDO;
extern int GRABADO;
```

Nuestro modulo contara con una función de inicialización de las interrupciones donde elegimos los pines 2.11 y 2.12 que corresponden con los pulsadores.

```
void init_EINT(void)
{
    // Configuración interrupciones externas
    LPC_PINCON->PINSEL4|=(1<<22);    // P2.11 es entrada interrup. EXT 1 (pulsador key1 en mini-dk2)
    LPC_PINCON->PINSEL4|=(1<<24);    // P2.12 es entrada interrup. EXT 2 (pulsador key2 en Mini-DK2)
    LPC_SC->EXTMODE|=(1<<1)|(1<<2);  // Por Flanco,
    LPC_SC->EXTPOLAR|=(1<<1)|(1<<2);  // de subida
    NVIC_SetPriority(EINT2_IRQn, 6);  // Menor prioritaria!!! ; sin CMSIS: NVIC->IP[18]=(4<<3);
    NVIC_EnableIRQ(EINT2_IRQn);       // sin CMSIS: NVIC->ISER[0]=(1<<18);
    NVIC_SetPriority(EINT1_IRQn, 6);
    NVIC_EnableIRQ(EINT1_IRQn);
}
```

Por otro lado tenemos las funciones de cada interrupción donde verificamos que estamos en el modo de depuración y que solo este pulsado uno de los pulsadores y no los dos ya que al pulsar los dos no se graba audio sino que cambiamos de modo.

```
/* *****
 * Function Name : EINT1 y EINT2
 * Description : Para grabacion y reproduccion de audio
 * Input : None
 * Output : None
 * Return : None
 * Attention : None
 * ***** */
void EINT1_IRQHandler() // PARA GRABAR
{
    LPC_SC->EXTINT=(1<<1); // Borrar flag Externa 1
    if (Estado == 7 && flagCambioMas2segPulsadores== false && flagCambioMenos2segPulsadores== false && flagContadorKey2==false){
        Audio(GRABANDO); //Para que el timer grabe
    }
}

void EINT2_IRQHandler() // PARA REPRODUCIR
{
    LPC_SC->EXTINT=(1<<2); // Borrar flag Externa 2
    if (Estado == 7 && flagCambioMas2segPulsadores== false && flagCambioMenos2segPulsadores== false && flagContadorKey1==false){
        Audio(GRABADO); //Para que el timer reproduzca
    }
}
```

## 2.8. Bluetooth.

Este modulo es utilizado para realizar una comunicación asíncrona que puede ser realizada mediante un modulo bluetooth o bien mediante un cable USB. Este modulo tendrá dos funcionalidades, la primera será mover el coche donde el coche pasara a modo manual, deshabilitaremos las lecturas del mando nunchuck y podremos mover el coche con comandos y la segunda funcionalidad será la de solicitar medidas devolviendo los datos de dichas medidas y el tiempo en milisegundos entre medidas.

En este modulo utilizaremos como variables externas las variables que contienen los datos de la velocidades de los motores, posición del servo y los datos de los sensores, además también utilizaremos la variable estado, una variable que indica el tiempo y unas variables externas proporcionadas por las bibliotecas de comunicación uart que nos han sido proporcionadas.

```
char buffer[30]; // Buffer de recepción de 30 caracteres --> Utilizada en la ISR de recepcion de la uart
char buffer_tx[30]; // Buffer de recepción de 30 caracteres
char *ptr_rx; // puntero de recepción
char *ptr; // puntero de recepción
char rx_completa; // Flag de recepción de cadena que se activa a "1" al recibir la tecla return CR(ASCII=13)
char *ptr_tx; // puntero de transmisión
char tx_completa; // Flag de transmisión de cadena que se activa al transmitir el caracter null (fin de cadena)
char fin=0;
long auxbth=0;
int auxmuestras = 0;
int estado_uart =1;
uint8_t orden_preparada = 0;

//variables externas
extern uint8_t Estado;
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;
extern bool Giro90;
extern bool Giromenos90;
extern uint32_t SLuz1,SLuz2,SPromedio;
extern int distancia;
extern int bluetooth;
extern int tiempo;
int milisegundos= 0;
bool modobluetooth= false;
```

Nuestro modulo contara con una única función que será una maquina de estados en la cual transmitimos mensaje y recibimos mensajes.

```
/*Funcion encargada de ejecutar la maquina de estados para el envio de mensajes*/
void MaquinaEstadosBluetooth(){
    switch(estado_uart){
        case 1: //envio del mensaje inicial
            ptr_rx=buffer;
            tx_cadena_UART0("Bienvenido al modo Bluetooth/USB indique su nombre\n\r");
            estado_uart = 2; //una vez que se ha indicado la trama a transmitir cambiamos de estado para que no la vuelva a repetir
            break;

        case 2:
            if(tx_completa == 1) //TRANSMISION DE LAS CADENAS(ESTADO SOLO PARA LA PRINCIPAL)
                estado_uart = 3; //en caso de que se haya transmitido toda empieza la recepcion
            break;

        case 3: //RECEPCION DE LAS CADENAS(ESTADO SOLO PARA LA PRINCIPAL)
            if(rx_completa == 0) //hasta que no se reciba todo nos mantenemos en este estado
                estado_uart = 3;

            if(rx_completa == 1){ //una vez que se reibe todo procedemos al estado general transmite-envia con ordenes en donde actuamos sobre el HW
                rx_completa = 0; //borramos flag
                estado_uart =4; //estado general
            }
            break;

        case 4: //PREPARAMOS LOS MENSAJES DE TRANSMISION GENERICOS

            if(orden_preparada == 0 ){
                tx_cadena_UART0("Elige modo:\n\r MOVER \n\r VISUALIZAR VALORES \n\r FIN \n\r");
                orden_preparada = 1;
            }

            estado_uart = 5;
            break;
```

```

case 5: // TRANSMISION DE MENSAJES GENERICOS
    if(tx_completa == 1 && fin == 0 )
        estado_uart = 6; //una vez transmitido todo pasamos a la recepcion
    else if(tx_completa == 1 && fin ==1)
        estado_uart = 0; //ninguno
    break;

case 6: //RECEPCION MENSAJES GENERICOS
    if(rx_completa == 0){
        estado_uart = 6; //se mantiene hasta que se completa
    }
    else if (rx_completa ==1){
        estado_uart = 4; //volvemos a preparar los mensajes para luego transmitirlos
        orden_preparada = 0;
        rx_completa = 0; //borramos flag

        //Segun lo recibido se actua sobre el HW
        if (strcmp (buffer, "MOVER\r") == 0) {
            estado_uart=7; //Realizamos ordenes
            Estado = 2;
            modobluetooth=true;
        }
        else if (strcmp (buffer, "VISUALIZAR VALORES\r") == 0)
            estado_uart=9; //Elegimos valores a visualizar
        else if (strcmp (buffer, "FIN\r") == 0){
            fin = 1;
            tx_cadena_UART0("FIN DEL PROGRAMA\n\r");
            estado_uart = 5;
        }
        else{ //en caso de ser un comando no reconocido
            tx_cadena_UART0("COMANDO ERRONEO\n\r");
            estado_uart = 5;
        }
    }
    break;

case 7: //ELIGE MOVIMIENTO DEL COCHE
    tx_cadena_UART0("Elige modo:\n\r ADELANTE \n\r ATRAS \n\r DERECHA \n\r IZQUIERDA \n\r VOLVER AL MENU PREVIO\n\r ");
    Posserv=0;
    estado_uart = 8;
    break;

case 8://PROCESAMOR ORDEN DE MOVIMIENTO
    Posserv=0;
    if(rx_completa == 0){
        estado_uart = 8; //se mantiene hasta que se completa
    }
    else if (rx_completa ==1){
        estado_uart = 7; //volvemos a preparar los mensajes para luego transmitirlos
        orden_preparada = 0;
        rx_completa = 0; //borramos flag

        //Segun lo recibido se actua sobre el HW
        if (strcmp (buffer, "ADELANTE\r") == 0)
        {
            VelDer=75;
            VelIzg=75;
            estado_uart=7; //Realizamos ordenes
        }
        else if (strcmp (buffer, "ATRAS\r") == 0)
        {
            VelDer=-75;
            VelIzg=-75;
            estado_uart=7; //Realizamos ordenes
        }
        else if (strcmp (buffer, "DERECHA\r") == 0){
            estado_uart = 7;
            Giromenos90 =true;
        }
        else if (strcmp (buffer, "IZQUIERDA\r") == 0){
            estado_uart = 7;
            Giro90 =true;
        }
        else if (strcmp (buffer, "VOLVER AL MENU PREVIO\r") == 0){
            estado_uart = 4;
            modobluetooth=false;
        }
        else{ //en caso de ser un comando no reconocido
            tx_cadena_UART0("COMANDO ERRONEO\n\r");
            estado_uart = 7;
        }
    }
    break;

case 9: //ELIGE DATO A VISUALIZAR
    if(tx_completa == 0){
        estado_uart = 9; //se mantiene hasta que se completa
    }
    else if (tx_completa ==1){
        tx_cadena_UART0("INDIQUE EL NUMERO DE MUESTRAS A TOMAR:\n\r ");
        estado_uart = 10;
    }
    break;

```

```

case 10://PROCESADOR ORDEN DE MOVIMIENTO
if(tx_completa == 0){
    estado_uart = 10; //se mantiene hasta que se completa
}
else if (tx_completa ==1){
    estado_uart = 9; //volvemos a preparar los mensajes para luego transmitirlos
    orden_preparada = 0;
    rx_completa = 0; //borramos flag
    //Segun lo recibido se actua sobre el HW
    auxbth =strtol(buffer,&ptr,10);

    if (auxbth > 0 && auxbth <100){
        sprintf(buffer_tx, "TOMA DE MUESTRAS:\n\r");
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        estado_uart = 11;
        auxmuestras= 8;
        tiempo =0;
    }
    else{ //en caso de ser un comando no reconocido
        tx_cadena_UART0("COMANDO ERRONEO\n\r");
        estado_uart = 9;
        tiempo =0;
    }
}
break;
case 11: //imprimimos muestras
if(tx_completa == 0){
    estado_uart = 11; //se mantiene hasta que se completa
}
else if (tx_completa ==1){
    if (auxmuestras==8)
    {
        sprintf(buffer_tx, "Motor Derecha: %3d \n\r", VelDer);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==7)
    {
        sprintf(buffer_tx, "Motor Izquierda: %3d \n\r", VelIzq);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==6)
    {
        sprintf(buffer_tx, "Posicion Servo: %3d °\n\r", Posserv);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==5)
    {
        sprintf(buffer_tx, "Sensor Luz1: %3d \n\r", SLuz1);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==4)
    {
        sprintf(buffer_tx, "Sensor Luz2: %3d \n\r", SLuz2);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==3)
    {
        sprintf(buffer_tx, "Promedio: %3d \n\r", SPromedio);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
    else if (auxmuestras==2)
    {
        sprintf(buffer_tx, "Distancia: %3d cm \n\r", distancia);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
    }
    else if (auxmuestras==1)
    {
        milisegundos=tiempo*10;
        auxbth--;
        sprintf(buffer_tx, "Tiempo de toma muestras:%d milisegundos \n\r",milisegundos);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras=8;
    }
    if(auxbth>0)
        estado_uart = 11;
    else
        estado_uart = 9;
}
break;
}
}

```

## 2.9. HTTP\_CGI.C

El Sistema aparte de tener acceso remoto mediante el modulo bluetooth también tendrá acceso remoto mediante un servidor web con el que cuenta nuestro sistema. Para este modulo haremos uso de las bibliotecas y archivos de configuración donde solo realizaremos la edición en el archivo Net\_config.c en el cual realizaremos la asignación Ip de nuestro servidor en nuestro caso será la 192.168.0.125 y los archivos index.cgi y HTTP\_CGI.c que es la encargada del funcionamiento del modulo.

Para poder dar la funcionalidad deseada al sistema en este modulo utilizaremos las variables externas que contienen los datos de lectura de los motores, el servo, los sensores y el estado en el que nos encontramos. También utilizaremos como variables externas una serie de variable proporcionadas por las bibliotecas.

```
// VARIABLES NUESTRO PROGRAMA
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;
extern int distancia;
extern uint8_t Estado;
extern uint32_t SLuz1,SLuz2,SPromedio;

/* http_demo.c */
extern U16 AD_in (U32 ch);
extern U8 get_button (void);

/* at_System.c */
extern LOCALM localm[];
#define LocM localm[NETIF_ETH]

/* Net_Config.c */
extern struct tcp_cfg tcp_config;
extern struct http_cfg http_config;
#define tcp_NumSocks tcp_config.NumSocks
#define tcp_socket tcp_config.Scb
#define http_EnAuth http_config.EnAuth
#define http_auth_passw http_config.Passw

extern BOOL LEDrun;
extern void LED_out (U32 val);
extern BOOL LCDupdate;
extern U8 lcd_text[2][16+1];
int auxhttp;
/* Local variables. */
static char const state[][9] = {
    "FREE",
    "CLOSED",
    "LISTEN",
    "SYN_REC",
    "SYN_SENT",
    "FINW1",
    "FINW2",
    "CLOSING",
    "LAST_ACK",
    "TWAIT",
    "CONNECT"};

/* My structure of CGI status U32 variable. This variable is private for */
/* each HTTP Session and is not altered by HTTP Server. It is only set to */
/* zero when the cgi_func() is called for the first time. */
typedef struct {
    U16 xcnt;
    U16 unused;
} MY_BUF;
#define MYBUF(p) ((MY_BUF *)p)
```



En este modulo tendremos las funciones de configuración cgi\_process\_var que es la función encargada de actualizar valores en nuestro sistema en función de lecturas en la web cuando utilizamos el método GET en el servidor HTTP.

```
/*----- cgi_process_var -----*/

void cgi_process_var (U8 *qs) {
    /* This function is called by HTTP server to process the Query_String */
    /* for the CGI Form GET method. It is called on SUBMIT from the browser. */
    /* The Query_String is SPACE terminated. */
    U8 *var;
    var = (U8 *)alloc_mem (40);

    do {
        /* Loop through all the parameters. */
        qs = http_get_var (qs, var, 40);
        /* Check the returned string, 'qs' now points to the next. */
        if (var[0] != 0) {
            /* Returned string is non 0-length. */
            if (str_scomp (var, "modo=manual") == __TRUE) {
                Estado=3;
            }
            if (str_scomp (var, "modo=automatico") == __TRUE) {
                Estado=5;
            }
            else if (str_scomp (var, "modo=depuracion") == __TRUE) {
                Estado=7;
            }

            if(str_scomp(var,"VelDer")==__TRUE){
                sscanf((const char*)&var[7], "%d", &auxhttp);
                VelDer =auxhttp;
            }

            else if(str_scomp(var,"VelIzq")==__TRUE){
                sscanf((const char*)&var[7], "%d", &auxhttp);
                VelIzq =auxhttp;
            }

            else if(str_scomp(var,"Posserv")==__TRUE){
                sscanf((const char*)&var[7], "%d", &auxhttp);
                Posserv =auxhttp;
            }
        }
    }while (qs);

    free_mem ((OS_FRAME *)var);
}

```

Y también haremos uso de función cgi\_func que utilizaremos para mostrar los datos que deseamos en el servidor web en nuestros caso los varoles de las variables y el estado en el que estamos.

```
/*----- cgi_func -----*/

3 void cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
    /* This function is called by HTTP server script interpreter to make a */
    /* formatted output for 'stdout'. It returns the number of bytes written */
    /* to the output buffer. Hi-bit of return value (len is or-ed with 0x8000) */
    /* is a repeat flag for the system script interpreter. If this bit is set */
    /* to 1, the system will call the 'cgi_func()' again for the same script */
    /* line with parameter 'pcgi' pointing to a 4-byte buffer. This buffer */
    /* can be used for storing different status variables for this function. */
    /* It is set to 0 by HTTP Server on first call and is not altered by */
    /* HTTP server for repeated calls. This function should NEVER write more */
    /* than 'buflen' bytes to the buffer. */
    /* Parameters: */
    /* env - environment variable string */
    /* buf - HTTP transmit buffer */
    /* buflen - length of this buffer (500-1400 bytes - depends on MSS) */
    /* pcgi - pointer to session local buffer used for repeated loops */
    /* This is a U32 variable - size is 4 bytes. Value is: */
    /* - on 1st call = 0 */
    /* - 2nd call = as set by this function on first call */
    U32 len = 0;

    3 switch (env[0]) {
        /* Analyze the environment string. It is the script 'c' line starting */
        /* at position 2. What you write to the script file is returned here. */
        case 'l':
            3 switch (env[2]) {
                case 'a':
                    /* Write the local IP address. The format string is included */
                    /* in environment string of the script line. */
                    len = sprintf((char *)buf, (const char *)env[4], (Estado==2)? "Manual": (Estado==3)? "Manual": (Estado==4)? "Automatico": (Estado==5)? "Automatico": (Estado==6)? "Depuracion": (Estado==7)? "Depuracion": "Espera");
                    break;
                case 'b':
                    /* Write the local IP address. The format string is included */
                    /* in environment string of the script line. */
                    len = sprintf((char *)buf, (const char *)env[4], VelDer);
                    break;
                case 'c':
                    /* Write local Net mask. */
                    len = sprintf((char *)buf, (const char *)env[4], VelIzq);
                    break;
                case 'd':
                    /* Write the local IP address. The format string is included */
                    /* in environment string of the script line. */
                    len = sprintf((char *)buf, (const char *)env[4], Posserv);
                    break;
                case 'f':
                    /* Write local Net mask. */
                    len = sprintf((char *)buf, (const char *)env[4], SLazl);
                    break;
            }
    }
}

```

```

case 'g':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], S1uz2);
    break;
case 'h':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], SFromedio);
    break;
case 'i':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], distancia);
    break;
case 'j':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], (Estado==3)? "checked='checked'":"" );
    break;
case 'k':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], (Estado==5)? "checked='checked'":"" );
    break;
case 'l':
    /* Write local Net mask. */
    len = sprintf((char *)buf, (const char *)&env[4], (Estado==7)? "checked='checked'":"" );
    break;
case 'm':
    len=sprintf((char *)buf, (const char *)&env[4], "<form action='/index.cgi' method='get'>
    <input style='text-align: center;' name='VelDer' size='3' type='text' value=' '> <input type='submit' value='Actualizar'></form>";
    break;
case 'n':
    len=sprintf((char *)buf, (const char *)&env[4], (Estado==7)? "<form action='/index.cgi' method='get'>
    <input style='text-align: center;' name='VelIzq' size='3' type='text' value=' '> <input type='submit' value='Actualizar'></form>";
    break;
case 'o':
    len=sprintf((char *)buf, (const char *)&env[4], (Estado==7)? "<form action='/index.cgi' method='get'>
    <input style='text-align: center;' name='Posserv' size='3' type='text' value=' '> <input type='submit' value='Actualizar'></form>";
    break;
default:
    break;
}
break;
default:
    break;
}
return ((UI6)len);
}

```

## 2.10.WatchDog

Hacemos uso de un modulo WatchDog para evitar las esperas activas no controladas, este modulo tienen dos funciones principales.

La primera es la función de inicialización del WatchDog:

```

void WDT_init(void)
{
    LPC_WDT->WDTC= F_wdclk*8; // Timeout=8seg. //valor de recarga
    LPC_WDT->WDCLKSEL=0x01; // Clock=PCLK
    LPC_WDT->WDMOD=0x03; // Enable y Reset si vence el tiempo
    LPC_WDT->WDFEED=0xAA;
    LPC_WDT->WDFEED=0x55;
}

```

La segunda función será la función de reinicio del WatchDog.

```

void WDT_Feed(void)
{
    LPC_WDT->WDFEED=0xAA;
    LPC_WDT->WDFEED=0x55;
}

```

### 2.11. Statechart Menú (MAQUINA DE ESTADOS)

En este apartado explicaremos el funcionamiento del menú principal y de la maquina de estados.

En este modulo tendremos definidos todos los estados tanto el estado de inicio como en de espera,tambien definiremos todas las variables externan que vamos a necesitar tanto sensores como posición de motores , como variables auxiliares.

```
/* Definición de los estados */
// Maquina de estados global
#define Inicio 0
#define Inicio_Espera 1
#define Modo_Manual 2
#define Modo_Manual_Espera 3
#define Modo_Automatico 4
#define Modo_Automatico_Espera 5
#define Modo_Depuracion 6
#define Modo_Depuracion_Espera 7
uint8_t Estado;
extern int umbralDistancia, distancia;
extern bool Barrido_Completo,Barrido;
extern int8_t Posserv;
//Variables Nunchuck

extern int8_t x; // coordenada x
extern int8_t y; // coordenada y
extern int8_t acelerometro_x, acelerometro_y, acelerometro_z;
extern int8_t resto;
extern bool flagContadorC, flagContadorZ, flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores;
extern int contadorZ, contadorC, contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioMas2segBotones, flagCambioMenos2segBotones;
extern int contadorKey1,contadorKey2;
extern int contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioPulsandoC;
extern int contadorCambioPulsandoC;
extern bool dep_nunchuck;

//Definicion de las pulsacion de pantalla
extern struct t_screenZone Manual;
extern struct t_screenZone Automatico;
extern struct t_screenZone Depuracion;
extern struct t_screenZone zone_1;
extern struct t_screenZone zone_2;
extern struct t_screenZone zone_3;
extern struct t_screenZone zone_4;
extern struct t_screenZone zone_5;
extern struct t_screenZone zone_6;
```

Este modulo esta compuesto por una función a la cual llamamos maquina de MaquinaEstados, dicha función empezara en el estado “Inicio” en el cual imprimirá la pantalla de inicio y pasara a un modo de espera esperando el estímulo para cambiar de modo.

```
case Inicio : // Inicializamos contadores y cargamos pantalla de inicio y pasamos a modo espera
    LCD_Clear(Black);
    PantallaInicio();
    Estado = Inicio_Espera;
    break;

case Inicio_Espera : //Elegimos modo en funcion de la zona que pulsemos
    if (zonePressed(&Manual)) {
        Estado = Modo_Manual;
    }
    if (zonePressed(&Automatico)) {
        Estado = Modo_Automatico;
    }
    if (zonePressed(&Depuracion)) {
        Estado = Modo_Depuracion;
    }
}
```

Dicha función tambien esta compuesta por los caso del estado “Manual “,”Automático y “Depuracion” en el cual tendremos un case para inicializar el estado y otro en el cual esperamos el estímulo para cambiar de estado, esto lo realizamos para no tener que imprimir la pantalla todo el rato. Cada vez que entramos en un estado nuevo borramos la pantalla anterior e imprimos la pantalla del nuevo estado, tambien reiniciaremos variables, contadores y pasaremos al modo espera del estado.

```

case Modo_Manual :
    Posserv=0; //iniciamos la posicion de servo
    setServo(Posserv);
    contadoresreset(); //Ponemos contadores de botones del mando nunchuck y pulsadores a 0
    LCD_Clear(Black); //Limpiamos pantalla
    PantallaManual(); //Imprimimos pantalla del modo manual
    Estado = Modo_Manual_Espera; //cambiamos a modo espera manual
    Barrido_Completo =false;
    Barrido= false;
    break;

case Modo_Automatico :
    Posserv=0; //iniciamos la posicion de servo
    setServo(Posserv);
    contadoresreset(); //Ponemos contadores de botones del mando nunchuck y pulsadores a 0
    LCD_Clear(Black); //Limpiamos pantalla
    PantallaAutomatico(); //Imprimimos pantalla del modo automatico
    Estado = Modo_Automatico_Espera; //cambiamos a modo espera automatico
    Barrido_Completo =false;
    Barrido= false;
    break;

case Modo_Depuracion :
    Posserv=0;
    setServo(Posserv);
    contadoresreset();
    LCD_Clear(Black);
    if(dep_nunchuck==false) //En funcion de la variable mostramos el modo depuracion con los datos de los sensores o bien con los valores de mando nunchuck
    {
        PantallaDepuracion_Nunchuck();
        dep_nunchuck=true;
        screenMessageIP();
    }
    else
    {
        PantallaDepuracion();
        dep_nunchuck=false;
        screenMessageIP();
    }
    Estado = Modo_Depuracion_Espera;
    Barrido_Completo =false;
    Barrido= false;

```

Como vemos el modo depuración tiene una condicio if esto se debe a que en el estado depuración tenemos dos sub estado en el cual visualizaremos 2 pantallas diferentes una que muestre los valores de los motores y sensores y otro que muestra las medidas del mando nunchuck.

Como hemos indicado dicha función tendrá también los modos espera en los cuales dichos modos junto con los timer determinan el funcionamiento de los modos de uso.

En el Modo\_Manual\_Espera si la distancia del sensor es superior a la umbral que hemos determinado el coche se moverá con valores que el nos devuelve el mando, por otro lado la posición del servo también varía en función de la inclinación del mando, también llamaremos una función que controla si están pulsado o no los pulsadores Key1 y Key2 y por último tendremos la condición de cambio de estado en función de los pulsadores y los botones del mando. El resto del funcionamiento del modo manual se realizará a través de los timer y las interrupciones externas.

```

case Modo_Manual_Espera:
    if( distancia> umbralDistancia)
        Movimiento_Nunchuck(); //Funcion movimiento ruedas con el mando.
    Servo_Nunchuck(); //Funcion de movimiento del servo en funcion de la inclinacion del mando.
    controladorPulsadores(); //Funcion que controla Pulsadores Key1 y Key2
    if (flagCambioMas2segPulsadores || flagCambioMas2segBotones) { //Cambiamos de modo en funcion de los pulsadores y botones
        Estado = Modo_Depuracion;
        Posserv=0;
    }
    if (flagCambioMenos2segPulsadores|| flagCambioMenos2segBotones) {
        Estado = Modo_Automatico;
    }
    break;

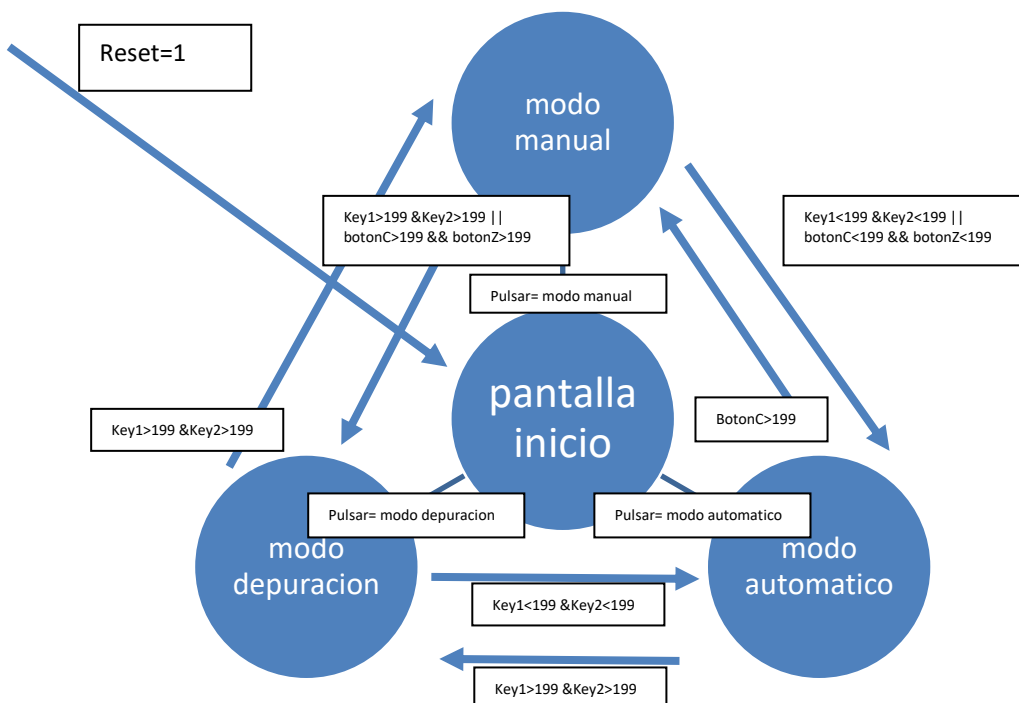
```

En el Modo\_Automatico\_Espera llamaremos una función que controla si están pulsado o no los pulsadores Key1 y Key2 y por ultimo tendremos la condición de cambio de estado en función de los pulsadores y los botones del mando. El resto del funcionamiento del modo automático se realizara con los timers.

```
case Modo_Automatico_Espera :
    controladorPulsadores();           //Funcion que controla Pulsadores Key1 y Key2
if (flagCambioMas2segPulsadores) { //cambiamos de modo en funcion de las keys y los pulsadores
    Estado = Modo_Depuracion;
    Posserv=0;
}
if(flagCambioPulsandoC)
    Estado = Modo_Manual;
break;
```

Por ultimo tendremos el Modo\_Depuracion\_Espera donde llamaremos a un función que si se toca la cabecera pasa de mostrar los valores que no se muestran en pantalla ocultando los que se muestran, tambien llamaremos a una función que controla si están pulsado o no los pulsadores Key1 y Key2 y por ultimo tendremos la condición de cambio de estado en función de los. El resto del funcionamiento del modo automático se realizara con los timers y la interrupciones externas eint1 y eint2 para grabar y reproducir audio.

```
case Modo_Depuracion_Espera:
    controladorPulsadores(); //controla los pulsadores
    modoDepuracion();        //funcion que cambia los valores a visualizar cada vez que tocamos la cabecera
    if (flagCambioMas2segPulsadores) { // cambiamos de modo en funcion de los pulsadores.
        Estado = Modo_Manual;
    }
    if (flagCambioMenos2segPulsadores) {
        Estado = Modo_Automatico;
    }
break;
default:
    break;
```



## 2.12. Timer

En este modulo tendremos las funciones Timer se podría decir que es el modelo mas importante ya que contiene el funcionamiento de la toma de datos de los sensores y del comportamiento del robot en función del modo en el que nos encontramos.

En este modulo definimos frecuencias que utilizaremos para el funcionamiento de los timers, declararemos las variables externas que utilizamos para los motores, el servo y el estado, tambien declaramos las variables para los sensores, contadores y variables auxiliares que utilizaremos para el funcionamiento del modulo.

```
//defenimos frecuencias
#define Th_trigger (10e-6*Fpclk_TIMx)
#define T_trigger (20e-3*Fpclk_TIMx)
#define Fcclk 100e6
#define Fpclk_TIMx (Fcclk/4)           // En los defines con operaciones: recomendable (...)
#define F_muestreo 10                  // Fs=10Hz (Cada 100 ms se toma una muestra del canal 0 y 2 )
#define F_out 8000
#define Fpclk 25e6
#define N_muestras 15872               // igual a la longitud del array generado en Matlab

//defenimos variables externas
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;

int N, distancia;
int tiempo=0;
extern bool modobluetooth;
bool barridoluz =false;

extern uint8_t Estado;
extern uint32_t SLuz1,SLuz2,SPromedio;
int32_t valorluz =40;
bool Luz= false;

//variables interruptores
bool flagContadorKey1=false,flagContadorKey2=false, flagCambioMas2segPulsadores=false, flagCambioMenos2segPulsadores=false;
int contadorKey1=0, contadorKey2=0;

//Variables Botones mando wii
bool flagContadorC=false, flagContadorZ=false,flagCambioMas2segBotones=false, flagCambioMenos2segBotones=false;
int contadorC=0, contadorZ=0;
bool flagCambioPulsandoC=false;
extern uint8_t muestras[];
extern uint8_t audio;
extern int ALARMA;
extern int GRABANDO;
extern int GRABADO;
extern int FIN;

//Variables para marcar lecturas y movimientos
int Lectura_nunchuck = 0;
int GiroDer =0;
int GiroIzq=0;
bool Barrido = false;
bool Barrido_Completo=false;
bool Giro90= false;
bool Giromenos90= false;
int aux_barrido = 0;
int aux_Giro=0;
int aux=0;
int grados=0;
int umbralDistancia= 27;
```

El TIMERO esta configurado para que produzca una interrupción cada 10 milisegundos.

```
void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1);           // Activa Timer0
    LPC_TIM0->PR = 24;                 // Para que cada tick sea de 1 Mhz (Fpclk/PR+1)
    LPC_TIM0->MCR = 0x3;               // Interrupt on Match 0 and Reset TC
    LPC_TIM0->MR0 = 10000;              // Periodo de interrupcion de TODOS los contadores 10ms
    LPC_TIM0->EMR = 1<<0 | 1<<5;       // Match por MR0 y set con 10 en EMC0 (bits 4 y 5)
    LPC_TIM0->TCR = 0x01;               // Arranca timer
    NVIC_SetPriority(TIMERO_IRQn,4);    // Prioridad por encima del priority grouping
    NVIC_EnableIRQ(TIMERO_IRQn);
}
```

El TIMER0 lo utilizaremos para tomar una lectura del mando nunchuck cada un tiempo definido, se encargara de realizar el barrido del servomotor del modo automático, tambien definirá los giros de 90º a izquierda o derecha,tambien se encargara de contar el tiempo pulsado de los key1, key2 y los botones del mando nunchuck y las acciones que debemos tomar en función del tiempo pulsado.

```
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR |= (1<<1);           // Borrar flag interrupción
    Lectura_nunchuck++;
    aux_barrido++;
    aux_Giro++;
    tiempo++;
    //LECTURA DEL MANDO NUNCHUCK
    if(Lectura_nunchuck == 19 && modobluetooth==false)
    {
        // Para que lea el Nunchuck cada segundo
        Lectura_nunchuck = 0;
        Nunchuck_read();
    }
    else if(Lectura_nunchuck == 19 && modobluetooth==true)
    {
        Lectura_nunchuck = 0;
    }
    //BARRIDOS DEL SERVOMOTOR
    if(aux_barrido == 49 && Barrido== true)
    {
        Posserv += 10;
        setServo(Posserv);
        aux_barrido=0;
        if (SPromedio>valorluz)
        {
            aux_barrido=0;
            grados=Posserv;
            Posserv=0;
            setServo(Posserv);
            Barrido=false;
            barridoluz =true;
        }
        if(Posserv==90&&aux<1)
        {
            Posserv=-90;
            setServo(Posserv);
            aux++;
        }
        else if(Posserv==90 && aux==1) //Los dos barridos han sido completados
        {
            Posserv=0;
            setServo(Posserv);
            Barrido_Completo= true;
            Barrido=false;
        }
    }
    else if(Barrido==false)
    {
        aux_barrido=0;
    }
    //FUNCION DE GIROS
    if(Giro90==true && Giromenos90==false )
    {
        VelDer=70;
        VelIzq= 0;
    }
    else if(Giromenos90==true && Giro90==false)
    {
        VelDer= 0;
        VelIzq= 70;
    }
    else if(Giromenos90==false && Giro90==false)
    {
        aux_Giro=0;
    }
    if(aux_Giro == 60 && (Giro90==true ||Giromenos90 ==true))
    {
        VelDer=0;
        VelIzq=0;
        Giro90=false;
        Giromenos90= false;
    }
    if(aux_Giro>60)
    {
        aux_Giro=0;
    }
}
```

```
//ESTADO DE LOS PULSADORES Y BOTONES
if (flagContadorKey1)
{
    contadorKey1++;
}
else
{
    if (contadorKey1>199 && contadorKey2>199)
    {
        flagCambioMas2segPulsadores= true;
        contadorKey1=0;
    }
    else if(contadorKey1>0 && contadorKey2>0)
    {
        flagCambioMenos2segPulsadores= true;
        contadorKey1=0;
    }
    else
        contadorKey1=0;
}

if (flagContadorKey2)
    contadorKey2++;
else
{
    if (contadorKey1>199 && contadorKey2>199)
    {
        flagCambioMas2segPulsadores= true;
        contadorKey2=0;
    }
    else if(contadorKey1>0 && contadorKey2>0)
    {
        flagCambioMenos2segPulsadores= true;
        contadorKey2=0;
    }
    else
        contadorKey2=0;
}

if (flagContadorC)
    contadorC++;
else
{
    if (contadorC>199 && contadorZ>199)
    {
        flagCambioMas2segBotones= true;
        contadorC=0;
        contadorZ=0;;
    }
    else if(contadorC>0 && contadorZ>0)
    {
        flagCambioMenos2segBotones= true;
        contadorC=0;
        contadorZ=0;
    }
    else if(contadorC>199 && contadorZ==0)
    {
        flagCambioPulsandoC= true;
        contadorC=0;
    }
    else if(contadorC>0 &&contadorZ==0)
    {
        contadorC=0;
        if(Estado==3)
            Audio(GRABANDO);
    }
}

if (flagContadorZ)
    contadorZ++;
else
{
    if (contadorC>199 && contadorZ>199)
    {
        flagCambioMas2segBotones= true;
        contadorZ=0;
        contadorC=0;
    }
    else if(contadorC>0 && contadorZ>0)
    {
        flagCambioMenos2segBotones= true;
        contadorZ=0;
        contadorC=0;
    }
    else if(contadorZ>0 && contadorC==0)
    {
        contadorZ=0;
        if (Estado==3)
            Audio(GRABADO);
    }
}
}
```



El TIMER1 esta configurado para utilizarse solamente cuandose vaya a grabar o reproducir audios, el resto del tiempo no esta activo este TIMER interrump cada 125 microsegundos.

```
void init_TIMER1(void)
{
    LPC_SC->PCONP|= (1<<2);           // Activa timer 1
    LPC_TIM1->PR = 0;                   // Para que cada tick sea de 25 Mhz (Fpclk/PR+1)
    LPC_TIM1->MCR = 0x03;               // Reset TC on Match, and Interrupt!
    LPC_TIM1->MR0 = Fpclk_TIMx/F_out-1; // MR1=25e6/(8KHz)-1 -----> Interrumpe cada 125 us
    LPC_TIM1->TCR = 0x02;               // Timer STOP y RESET
    NVIC_EnableIRQ(TIM1_IRQn);
    NVIC_SetPriority(TIM1_IRQn,4);      // Prioridad por encima del priority grouping
}                                       // No hace falta EHR ni asignar un pin porque no queremos ningun match, solo reseteo
```

El TIMER1 reproducira la alarma la alarma almacenada en memoria, reproducirá el audio grabado previamente o bien grabara un audio, todo esto en función de la variable semáforo audio.

```
void TIM1_IRQHandler(void)
{
    //Relativas al audio

    static uint16_t indice_muestra;
    LPC_TIM1->IR|= (1<<0);             // Borrar flag
    if (audio == ALARMA) {
        LPC_DAC->DACR= muestra[indice_muestra++]<<8; // 8 bits!!!!
    }
    else if(audio == GRABADO)
    {
        LPC_DAC->DACR= muestras[indice_muestra++]<<8; // 8 bits!!!!
    }
    else if(audio==GRABANDO)
    {
        LPC_ADC->ADINTEN=(1<<2); // Hab. interrupción fin de conversión del canal 0
        NVIC_EnableIRQ(ADC_IRQn);
    }
    if(indice_muestra==(N_muestras-1))
    {
        indice_muestra=0;
        LPC_TIM1->TCR=0x02; //Stop Timer and reset, DAC= 0V.
        LPC_DAC->DACR=0;    // 0 V /33
        init_ADC_LDRs();
        audio= FIN;
    }
    else if( indice_muestra==(N_muestras/2)-1 && audio ==ALARMA)
    {
        indice_muestra=0;
        LPC_TIM1->TCR=0x02; //Stop Timer and reset, DAC= 0V.
        LPC_DAC->DACR=0;    // 0 V /33
        init_ADC_LDRs();
        audio =FIN;
    }
}
```

El TIMER2 sera el encargado de tomar las muestras de los LDRs y de controlar el movimiento del modo automático en función de las muestras de este TIMER interrump cada 125 microsegundos.

```
// TIMER 2 PARA LDRs Y MOVIMIENTO MODO AUTOMATICO EN FUNCION DE LOS LDRS
void init_TIMER2(void)
{
    LPC_SC->PCONP|= (1<<22);           // Alimento el Timer2 , está desactivado de forma predeterminada
    LPC_PINCON->PINSEL9|= (1<<25);     // MAT2.0 para PWM servo (P4.28)
    LPC_TIM2->MCR = 0x18;               // Interrupt on Match 1 and Reset TC (bit 4 y 5 activos)
    LPC_TIM2->MR1 = (Fpclk/F_muestreo)-1; // Periodo de muestreo de TODAS las entradas!!!!
    LPC_TIM2->EMR = 0x00C2;             // MODO TOGGLE en EMCL
    LPC_TIM2->TCR = 0x01;               // Arranca timer
    NVIC_EnableIRQ(TIM2_IRQn);
    NVIC_SetPriority(TIM2_IRQn,4);      // Prioridad por encima del priority grouping
}                                       // No hace falta asignar un pin porque no queremos ningun match, solo interrump
```

El TIMER2 tomara muestras de los sensores LDRs mediante el ADC, llamara a las funciones que pintaras los valores obtenidos de dichos sensores y además será la encargada de la toma de decisiones del movimiento del modo autónomo en función de los valores de los LDR's y de la distancia.

```
void TIMER2_IRQHandler(void)
{
    LPC_TIM2->IR |= (1<<1); // Borrar flag interrupción
    LPC_ADC->ADCR |= (1<<16); // BURST=1 --> Cada 65TclkADC se toma una muestra de cada canal comenzando
                                // desde el más bajo (bit LSB de CR[0..7])

    Dato_sensorLuz1();
    Dato_sensorLuz2();
    Dato_promedioSensores();

    //Movimiento automatico modo automatico
    if (Estado==5 && distancia>umbralDistancia && Giro90== false && Giromenos90== false)
    {
        if (SPromedio>valorluz && barridoluz==false) //Si el coche detecta LUZ
        {
            Posserv=0;
            setServo(Posserv);
            grados=0;
            Barrido_Completo=false;
            Barrido= false;
            if (SLuz1>SLuz2) //Si la el el sensor IZq recibe mas luz nos movemos a la izquierda
            {
                VelIzq= 60;
                VelDer = 75;
            }
            else if (SLuz1<SLuz2) //Si la el el sensor Der recibe mas luz nos movemos a la Derecha
            {
                VelIzq = 75;
                VelDer = 60;
            }
            else //Si reciben la misma Luz nos movemos a recto
            {
                VelDer =75;
                VelIzq=75;
            }
        }
        else //Si no detecta luz
        {
            if (Barrido_Completo==false&& Barrido== false && grados ==0)
            {
                Posserv=-90;
                setServo(Posserv);
                Barrido = true;
            }
            else if (Barrido_Completo== true)
            {
                VelDer=70;
                VelIzq=70;
            }
            else if (grados<0)
            {
                VelIzq= 60;
                VelDer = 75;
                barridoluz=false;
            },
            else if (grados<0)
            {
                VelIzq= 60;
                VelDer = 75;
                barridoluz=false;
            }
            else if (grados>0)
            {
                VelIzq= 75;
                VelDer = 60;
                barridoluz=false;
            }
        }
    }
    else if (distancia<umbralDistancia&& Estado ==5)
    {
        Barrido =false;
        Barrido_Completo= false;

        if (SLuz1>SLuz2) //Si la el el sensor IZq recibe mas luz nos movemos a la izquierda
        {
            Giro90= true; //Giro Izquierda 90°
        }
        else if (SLuz2>SLuz1) //Si la el el sensor IZq recibe mas luz nos movemos a la izquierda
        {
            Giromenos90= true; //Giro Derecha 90°
        }
        else
        {
            Giro90= true;
        }
    }
}
```

El TIMER3 esta configurado se utilizara solamente cuandose vaya a grabar o reproducir audios, el resto del tiempo no esta activo este TIMER interrupe cada 500 milisegundos o bien cada vez que tenemos un flanco en el CAP3.0

```
// TIMER 3 PARA ULTRASONIDO
void init_TIMER3(void)
{
    LPC_SC->PCONP|= 1<<23;           // Alimento el Timer 3, que esta desactivado de forma predeterminada.
    LPC_PINCON->PINSEL0|=(3<<22);    // MAT3.1 para el Trigger (P0.11)
    LPC_PINCON->PINSEL1|=(3<<14);    // CAP3.0 para el echo (P0.23)

    LPC_TIM3->PR = 24;
    LPC_TIM3->CCR = 0x6;             // Interrupcion en flanco de bajada

    LPC_TIM3->MCR |= (0x3<<0);       // GENERAR INTERRUPTIÓN Y RESETEAR CUANDO LLEGUE MRO

    LPC_TIM3->MR0 = T_trigger;
    LPC_TIM3->MR1 = Th_trigger;
    LPC_TIM3->EMR = 1<<1 | 1<<6;    // mas rápido que EMR |= 1<<1;

    NVIC_SetPriority(TIM3_IRQn,3);
    NVIC_EnableIRQ(TIM3_IRQn);

    LPC_TIM3->TCR = 1<<1;            // RESET TIM3
    LPC_TIM3->TCR = 1<<0;            // START TIM3.
}
```

En este TIMER capturaremos el tiempo en el que se produce un flanco de subida y luego volverá a interrumpir en el siguiente flanco que será el flanco de bajada así podremos capturar el ECHO y la distancia en centímetros dividiendo entre 58. Si la distancia es menor a la distancia umbral que hemos seleccionado y estamos en el modo manual o automático el coche sonara si además estamos en el modo manual además el vehículo se parara.

```
void TIM3_IRQHandler(void) {
    static uint32_t temp;
    if(LPC_TIM3->IR & (1 << 0)){
        LPC_TIM3->IR |= 1<<0;
        LPC_TIM3->EMR = 1<<6 | 1<<1; // mas rápido que EMR |= 1<<1;
    }

    if (LPC_TIM3->IR & (1 << 4)){ // si hay evento de CR0
        LPC_TIM3->IR |= 1<<4;

        if(LPC_TIM3->CCR == 5){
            temp = LPC_TIM3->CR0;
            LPC_TIM3->CCR = 0x6;
        }
        else{
            N = LPC_TIM3->CR0 - temp; // Tiempo en alto
            LPC_TIM3->CCR = 0x5;
        }
        if(N>58){
            distancia = N/58;
            Dato_Distanciadm();
            if (distancia < umbralDistancia && Barrido==false){
                if(Estado==3){
                    VelDer= 0;
                    VelIzq = 0;}
                if(Estado==3 || Estado==5)
                    Audio(ALARMA);
            }
        }
    }
}
```

En este modulo tambien tendremos una función que inicializa todos los timer menos el TIMER1 que solo lo utilizamos cuando sea necesario.

```
//INICIA TODOS LOS TIMERS MENOS EL TIMER1 QUE UTILIZAMOS PARA EL AUDIO Y SOLO LO INICIAREMOS CUANDO LO NECESITEMOS
void init_TIMERS(void)
{
    init_TIMER0();
    init_TIMER2();
    init_TIMER3();
}
```

Tambien tendremos una función que verifica si los pulsadores están pulsados o no.

```
//FUNCION QUE CONTROLA QUE LOS PULSADORES SEAN PULSADOS
void controladorPulsadores(void) {
    if ((LPC_GPIO2->FIOPIN & 1<<11) == 0x0000) { //Key1
        flagContadorKey1 = true;
    }
    else {
        flagContadorKey1 = false;
    }
    if ((LPC_GPIO2->FIOPIN & 1<<12) == 0x0000) { //Key2
        flagContadorKey2 = true;
    }
    else {
        flagContadorKey2 = false;
    }
}
```

Y por ultimo tendremos una función que reinicia los flags que utilizamos para los pulsadores y botones del mando.

```
//FUNCION QUE REINICIA CONTADORES
void contadoresreset(void)
{
    contadorKey1=0;
    contadorKey2=0;
    contadorC = 0;
    contadorZ = 0;
    flagCambioMas2segPulsadores=false;
    flagCambioMenos2segPulsadores=false;
    flagCambioMas2segBotones=false;
    flagCambioMenos2segBotones=false;
    flagCambioPulsandoC = false;
}
```

### 3. Análisis de ejecutibilidad del sistema.

No hay ninguna región crítica en el programa. Grupo de prioridad 2 por lo tanto no tendremos subprioridades.

TAREA	PRIORIDAD	C(us)	T(us)	D(ms)
ADC	0	0,71	125	125
UART	3	0,45	1146	1146
Timer3	3	2,77	100	100
Timer0	4	9608	10000	10000
Timer1	4	2,09	125	125
Timer2	4	8394	100000	100000

#### ADC:

El ADC tiene una frecuencia 12.5MHz por lo que de  $T=125\text{ us}$ , y tiene 6 instrucciones en el peor caso por lo que su C es de 0.71 us

#### Timer0

El Timer0 tiene un MR de 10000 tick y la frecuencia de cada tick es de 1MHz por lo que  $T=10000\text{us}=10\text{ ms}$ , y en el peor caso se ejecuta en 9608us por lo tanto su C sería 9608 us.

#### Timer1

El Timer1 tiene una frecuencia de 8000 hz para el altavoz, que reproduce el sonido. Por lo que tiene un periodo de  $T=0.125\text{ms}$ , y en el peor caso se ejecuta en 2,09 us por lo tanto su C sería 2,09 us.

#### Timer2

El Timer2 tiene un MR de 2500000 tick y la frecuencia de cada tick es de 25MHz por lo que  $T=100000\text{us}=100\text{ ms}$ , y en el peor caso se ejecuta en 8394 us por lo tanto su C sería 8394 us.

#### Timer3

El Timer3 tiene un MR de 500000 tick y la frecuencia de cada tick es de 1MHz pero tambien puede interrumpir cada flanco que tengamos por cap3.0 por lo que el periodo mínimo según el dispositivo será de 100us ya que es la distancia mínima que podremos tener entre flanco de subida y de bajada por lo que  $T=100\text{us}$  y en el peor caso se ejecuta en 2,77 us por lo tanto su C sería 2,77 us.

#### UART0

Esta controla el bluetooth por lo que deberá tener la mayor prioridad en este caso. Se ejecuta cada 8 bit de datos + 1 bit paridad + 1 bit start +1 bit de stop /9600 lo que da un period de 1146 us. Y su peor caso se ejecuta en 450ns por lo que su C será de 450 ns.

La tarea ADC puede interrumpir a todas las tareas. El timer 0, 1 y 2 tienen el mismo nivel de prioridad y subprioridad. El TIMER3 y UART tienen el mismo nivel de prioridad, menor nivel de prioridad que el ADC pero mas que el resto de los timers.

$$R_{ADC} = C_{ADC} = 1 < D_{ADC} = 125 \mu s$$

$$R_{UART} = C_{UART} + C_{ADC} \left[ \frac{R_{UART}}{T_{ADC}} \right]$$

$$W_0 = 2 \mu s; W_1 = 1 + 1 \frac{2}{125} = 2 \mu s < D_{UART} = 1146$$

$$R_{TIMER3} = C_{TIMER3} + C_{ADC} \left[ \frac{R_{TIMER3}}{T_{ADC}} \right]$$

$$W_0 = 3 + 1 = 4; W_1 = 3 + 1 \frac{5}{125} = 4 \mu s < W_0 < D_{TIMER3} = 1146$$

$$R_{TIMER0} = C_{TIMER0} + C_{ADC} \left[ \frac{R_{TIMER0}}{T_{ADC}} \right] + C_{UART} \left[ \frac{R_{TIMER0}}{T_{UART}} \right] + C_{TIMER3} \left[ \frac{R_{TIMER0}}{T_{TIMER3}} \right]$$

$$W_0 = 9608 + 1 + 1 + 3 = 9613; W_1 = 9608 + 1 \frac{9613}{125} + 1 \frac{9613}{1146} + 3 \frac{9613}{100} = 9982 \mu s;$$

$$W_2 = 9997 \mu s = W_3 < D_{TIMER0} = 10000 \mu s$$

$$R_{TIMER1} = C_{TIMER1} + C_{ADC} \left[ \frac{R_{TIMER1}}{T_{ADC}} \right] + C_{UART} \left[ \frac{R_{TIMER1}}{T_{UART}} \right] + C_{TIMER3} \left[ \frac{R_{TIMER1}}{T_{TIMER3}} \right]$$

$$W_0 = 3 + 1 + 1 + 3 = 8; W_1 = 3 + 1 \frac{8}{125} + 1 \frac{8}{1146} + 3 \frac{8}{100} = 8 \mu s;$$

$$W_1 < D_{TIMER1} = 125 \mu s$$

$$R_{TIMER2} = C_{TIMER2} + C_{ADC} \left[ \frac{R_{TIMER2}}{T_{ADC}} \right] + C_{UART} \left[ \frac{R_{TIMER2}}{T_{UART}} \right] + C_{TIMER3} \left[ \frac{R_{TIMER2}}{T_{TIMER3}} \right]$$

$$W_0 = 8394 + 1 + 1 + 3 = 8399; W_1 = 8394 + 1 \frac{8399}{125} + 1 \frac{8399}{1146} + 3 \frac{8399}{100} = 8721 \mu s;$$

$$W_2 = 8734 \mu s = W_3 < D_{TIMER2} = 100000 \mu s$$

El sistema es ejecutable debido a que todas las tareas tienen un tiempo de respuesta menor que el deadline.

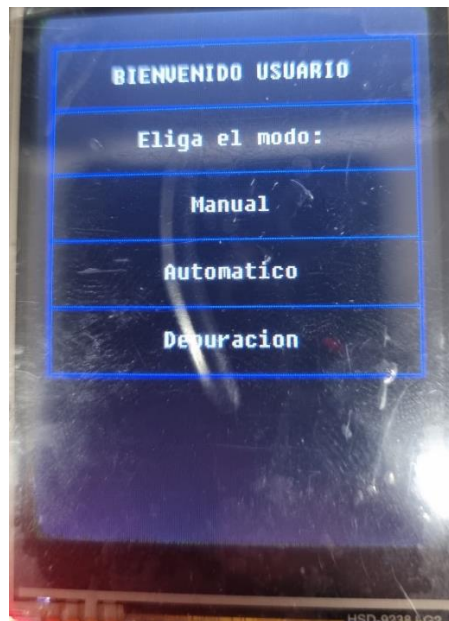
#### 4. Manual de usuario.

##### 4.1 Introducción.

En este apartado, se verá una breve explicación del funcionamiento de nuestro proyecto a nivel de usuario. Veremos los pasos que se deben seguir para poder ejecutar los 3 modos de uso Manual, Automático y Depuración, también explicaremos las conexiones serie asíncronas mediante Bluetooth/USB y a través del servidor web.

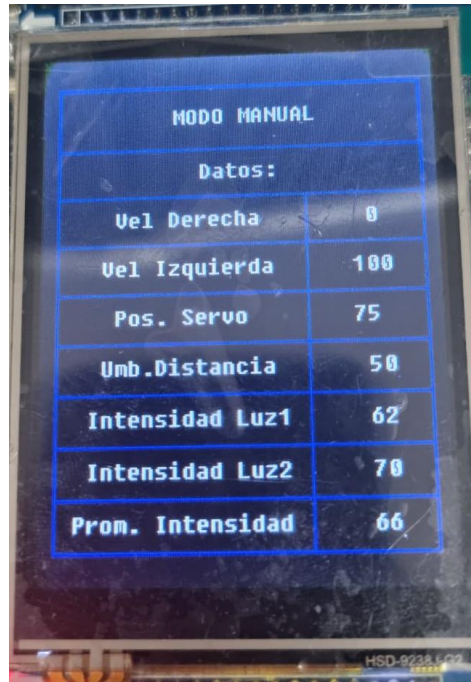
##### 4.2 Menú principal.

En primer lugar tras encender el LPC1768 o pulsar RESET, nos vamos a encontrar con un menú principal el cual tendrá las siguientes opciones.



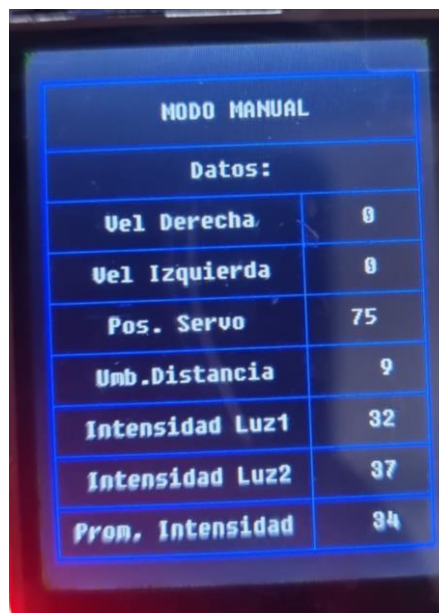
#### 4.3 Modo Manual.

En este modo el usuario tendrá la opción de controlar al coche manualmente a través del joystick del mando Nunchuck, también moveremos el servo en función de la inclinación del mando Nunchuck y para finalizar cuando detectemos un obstáculo el coche se detendrá.



MODO MANUAL	
Datos:	
Vel Derecha	0
Vel Izquierda	100
Pos. Servo	75
Umb. Distancia	50
Intensidad Luz1	62
Intensidad Luz2	70
Prom. Intensidad	66

Como podremos ver en la siguiente imagen al estar cerca de un obstáculo detectado por el sensor el vehículo se parará y reproducirá un audio alarma almacenado.



MODO MANUAL	
Datos:	
Vel Derecha	0
Vel Izquierda	0
Pos. Servo	75
Umb. Distancia	9
Intensidad Luz1	32
Intensidad Luz2	37
Prom. Intensidad	34

En este modo podremos grabar audio pulsando el botón C del mando y reproducirlo pulsando el botón Z del mando nunchuck. Además el robot pasará al modo Automático cuando se pulsen los dos botones del mando de la Wii de forma simultánea durante menos de 2 segundos o los pulsadores Key1 y Key2 de la tarjeta, y pasará al modo Depuración cuando se pulsen los dos botones durante más de 2 segundos o los pulsadores Key1 y Key2 de la tarjeta.



#### 4.4 Modo Automatico.

En este modo el coche se moverá de manera autónoma buscando una fuente de luz. Como podemos ver en las siguientes imágenes cuando la fuente de luz detectada es mas fuerte en un sensor el coche va orientandose hacia esa luz en caso de ser la misma el coche se desplaza recto.

MODO AUTOMATICO	
Datos:	
Vel Derecha	75
Vel Izquierda	75
Pos. Servo	0
Umb. Distancia	67
Intensidad Luz1	72
Intensidad Luz2	72
Prom. Intensidad	72

MODO AUTOMATICO	
Datos:	
Vel Derecha	60
Vel Izquierda	75
Pos. Servo	0
Umb. Distancia	68
Intensidad Luz1	65
Intensidad Luz2	67
Prom. Intensidad	66

MODO AUTOMATICO	
Datos:	
Vel Derecha	75
Vel Izquierda	60
Pos. Servo	0
Umb. Distancia	67
Intensidad Luz1	72
Intensidad Luz2	65
Prom. Intensidad	68

Por otro lado cuando el coche detecte un obstáculo lo intentara evitar girando 90º hacia el lado donde la luz es mas fuerte en caso de ser la fuente de la misma intensidad girara hacia la izquierda.

MODO AUTOMATICO	
Datos:	
Vel Derecha	70
Vel Izquierda	0
Pos. Servo	0
Umb. Distancia	3
Intensidad Luz1	70
Intensidad Luz2	57
Prom. Intensidad	63

MODO AUTOMATICO	
Datos:	
Vel Derecha	0
Vel Izquierda	70
Pos. Servo	0
Umb. Distancia	4
Intensidad Luz1	45
Intensidad Luz2	70
Prom. Intensidad	57

En este modo automático cuando la intensidad de luz promedio es menor que el que indicamos realizaremos dos barridos buscando una fuente, en el caso de encontrar la fuente nos desplazamos en ese sentido, sino encontramos nada nos desplazamos hacia delante.

Para finalizar el robot pasara a modo Manual al pulsar el boton C del mando Nunchuck durante mas de 2 segundos y pasara a modo Depuracion cuando se pulsen los dos pulsadores Key1 y Key2 de la tarjeta durante mas de 2 segundos.

#### 4.5 Modo Depuracion.

En este modo el coche será controlado mediante la pantalla táctil donde podremos modificar los valores de la velocidad de los motores o bien la posición del Servo, tambien se mostraran el valor del resto de sensores y la IP, si pulsamos en al pantalla táctil en la cabecera pasaremos a visualizar los valores que nos proporciona el mando nunchuck.

MODO DEPURACION			
Dir. IP:	192.168.0.125		
V.Derecha	75	+	—
V.Izquierd	75	+	—
Pos.Servo	-23	+	—
Umb.Distance		1217	
Intensidad Luz1		75	
Intensidad Luz2		70	
Promedio Int.		72	

MODO DEPURACION	
Dir. IP:	192.168.0.125
Eje X	-1
Eje Y	-1
Acelerometro X	-1
Acelerometro Y	-1
Acelerometro Z	-1
Boton C	0
Boton Z	0

Ademas al pulsar Key 1, grabara audio durante un segundo y al pulsar Key 2, reproducira el audio grabado. Para finalizar el robot pasara al modo Automatico cuando se pulsen simultaneamente durante menos de 2 segundos los pulsadores Key1 y Key2 de la tarjeta, y pasara al modo Manual cuando se pulsen los pulsadores Key1 y Key2 de la tarjeta durante mas de 2 segundos.

#### 4.6 Modo Online.

En este modo online a través del servidor web integrado en la Mini DK2, podremos visualizar el modo en el que nos encontramos, por otro lado también podremos visualizar los valores de la velocidad de los motores, la posición del servo y los datos de los sensores. Además cuando nos encontremos en el modo depuración tendremos la opción de actualizar los valores de los motores y del servo.

La primera imagen corresponde a cuando estamos en el menú principal donde podemos ver que el modo en el que estamos es de espera.

Control coche	
<b>Espera</b>	
Vel. Derecha:	0
Vel. Izquierda:	0
Posicion Servo:	0
Sensor Luz 1:	62
Sensor Luz 2:	62
Promedio:	62
Distancia:	61

**Selección de modo**

☐ Modo Manual
 ☐ Modo Automatico
 ☐ Modo Depuracion

Las imágenes de abajo corresponden a cuando estemos en el modo Manual.

Control coche	
<b>Manual</b>	
Vel. Derecha:	0
Vel. Izquierda:	100
Posicion Servo:	75
Sensor Luz 1:	52
Sensor Luz 2:	57
Promedio:	54
Distancia:	47

**Selección de modo**

☒ Modo Manual
 ☐ Modo Automatico
 ☐ Modo Depuracion

MODO MANUAL	
Datos:	
Vel Derecha	0
Vel Izquierda	100
Pos. Servo	75
Umb.Distancia	47
Intensidad Luz1	50
Intensidad Luz2	55
Prom. Intensidad	52

Las imágenes de abajo corresponden cuando nos encontramos en el modo Automático.

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 60

### DISEÑO DE UN ROBOT CON TRACIÓN DIFERENCIAL

Control Web

No es seguro | 192.168.0.125/index.cgi

Control coche
<b>Automatico</b>
Vel. Derecha: 60
Vel. Izquierda: 75
Posicion Servo: 0
Sensor Luz 1: 77
Sensor Luz 2: 80
Promedio: 78
Distancia: 63

**Selección de modo**

☐ Modo Manual
 ☒ Modo Automatico
 ☐ Modo Depuracion

MODO AUTOMATICO	
Datos:	
Vel Derecha	60
Vel Izquierda	75
Pos. Servo	0
Umb.Distancia	62
Intensidad Luz1	77
Intensidad Luz2	80
Prom. Intensidad	78

En las imágenes de abajo veremos el modo depuración y además como podemos actualizar valores y serán reflejados en nuestro sistema.

Control Web

No es seguro | 192.168.0.125/index.cgi

Control coche
<b>Depuracion</b>
Vel. Derecha: 60
Vel. Izquierda: 75
Posicion Servo: 0
Sensor Luz 1: 62
Sensor Luz 2: 62
Promedio: 62
Distancia: 69

**Selección de modo**

☐ Modo Manual
 ☐ Modo Automatico
 ☒ Modo Depuracion

MODO DEPURACION	
Dir. IP:	192.168.0.125
U.Derecha	60
U.Izquierd	75
Pos.Servo	0
Umb.Distancia	64
Intensidad Luz1	62
Intensidad Luz2	62
Promedio Int.	62

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 61

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

Control Web

Depuracion

Vel. Derecha: 75	Actualizar
Vel. Izquierda: 75	Actualizar
Posicion Servo: 75	Actualizar
Sensor Luz 1: 52	
Sensor Luz 2: 57	
Promedio: 54	
Distancia: 47	

Selección de modo

☐ Modo Manual
 ☐ Modo Automatico
 ☒ Modo Depuracion

Control Web

Depuracion

Vel. Derecha: 75	Actualizar
Vel. Izquierda: 0	Actualizar
Posicion Servo: 75	Actualizar
Sensor Luz 1: 52	
Sensor Luz 2: 57	
Promedio: 54	
Distancia: 47	

Selección de modo

☐ Modo Manual
 ☐ Modo Automatico
 ☒ Modo Depuracion

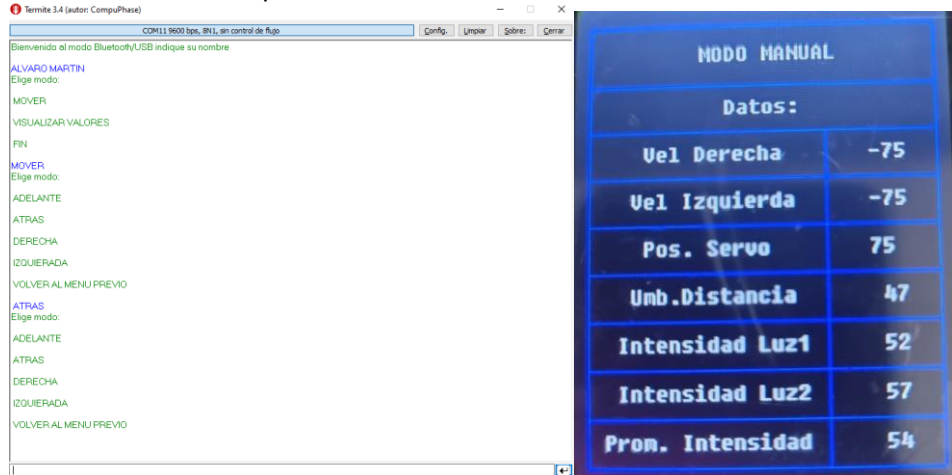
**NODO DEPURACION**

Dir. IP:	192.168.0.125	
V.Derecha	75	← →
V.Izquierd	0	← →
Pos.Servo	75	← →
Umb.Distancia	47	
Intensidad Luz1	52	
Intensidad Luz2	57	
Promedio Int.	54	

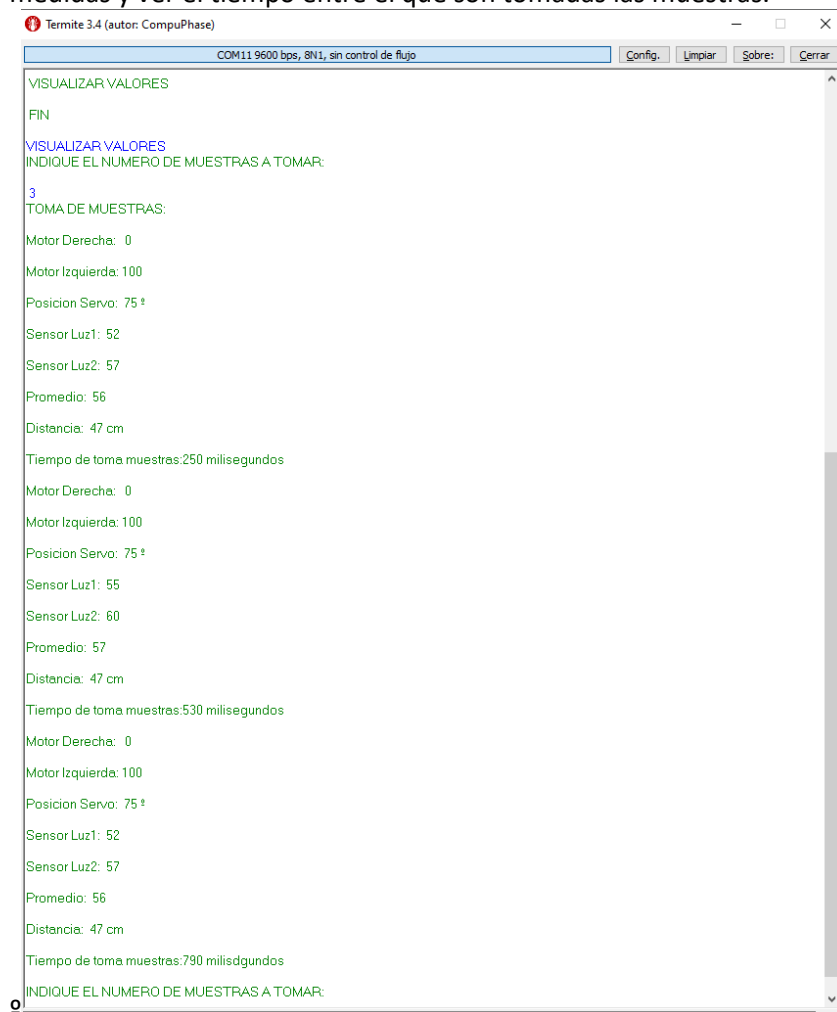


#### 4.7 Conexion Bluetooth/USB.

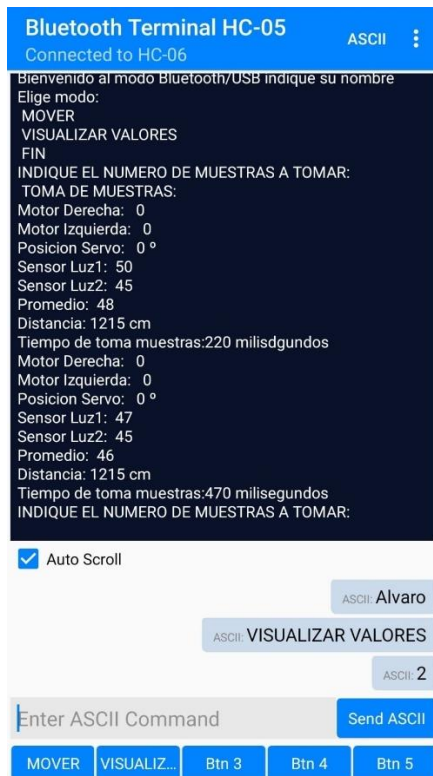
En el modo de conexion USB tendremos dos modos 2 funcionalidades la primera para mover el coche donde el coche se pondra en modo manual y desactivara las lecturas del nunchuck para poder moverse con los comandos que le demos exclusivamente.



Por otro lado tendremos la otra funcionalidad que sirve para visualizar un numero indicado de medidas y ver el tiempo entre el que son tomadas las muestras.



Por ultimo tambien tendremos este acceso mediante un modulo Bluetooth HC06 donde nos conectaremos mediante la aplicacion Bluetooth terminal de android y podremos realizar las mismas operaciones que con la conexion USB.



## ANEXO

### I. Código fuente del Proyecto.

#### a. *Main.c*

```
#include "PWM.h"
#include "tp_simple.h"
#include "StateChar.h"
#include "i2c_lpc17xx.h"
#include "Nunchuck.h"
#include "ADC.h"
#include "Timer.h"
#include "DAC.h"
#include "Interrupciones.h"
#include "uart.h"
#include "Bluetooth.h"
#include <RTL.h>
#include <Net_Config.h>
#include <serial.h>
#include "stdbool.h"
#include "WatchDog.h"

extern U32 CheckMedia (void);

/*----- init -----*/
/*****
 * Function Name : init
 * Description : Initialize every subsystem
 * Input : None
 * Output : None
 * Return : None
 * Attention : None
 *****/
static void init () {

    SER_Init ();
    init_TcpNet ();
    /* Setup and enable the SysTick timer for 100ms. */
    SysTick->LOAD = (SystemCoreClock / 10) - 1;
    SysTick->CTRL = 0x05;

}

/*----- timer_poll -----*/
/*****
 * Function Name : timer_poll
 * Description : Call timer_tick() if every 100ms (aprox)
 * Input : None
 * Output : None
 * Return : None
 * Attention : None
 *****/
static void timer_poll () {
    /* System tick timer running in poll mode */

    if (SysTick->CTRL & 0x10000) {
        /* Timer tick every 100 ms */
        timer_tick ();
    }
}

extern uint8_t Estado;
extern struct t_screenZone Manual;
extern struct t_screenZone Automatico;
extern struct t_screenZone Depuracion;
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;
int main(void)
{
    LCD_Initializtion();
    LCD_Clear(Black);
    TP_Init();
    Estado= 0;
    // TouchPanel_Calibrate();
    configPWM();
    PuenteH ();
    Nunchuck_Init();
    init_ADC_LDRs();
    init_TIMERS();
    init_DAC();
    init_EINT();
}
```



```
NVIC_SetPriorityGrouping(2);  
uart0_init(9600);  
Nunchuck_Init();  
WDT_init();  
init();  
while(1)  
{  
    MaquinaEstados();  
    MaquinaEstadosBluetooth();  
    MotorDerecha(VelDer);  
    MotorIzquierda(VelIzg);  
    setServo(Posserv);  
    timer_poll();  
    main_TcpNet();  
    WDT_Feed();  
}
```

### *b. PWM.c*

```
#include "PWM.h"
#include "tp_simple.h"

#define Fpclk 25e6 // Fcpu/4 (defecto después del reset)
#define Tpwmm 15e-3 // Perido de la señal PWM (15ms)

int8_t VelDer = 0;
int8_t VelIzq = 0;
int8_t Posserv = 0;

/*****
* Function Name : configPWM
* Description : Configura una señal PWM de 15ms de periodo por P1.20
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void configPWM(void) {

    LPC_PINCON->PINSEL7|=(3<<18); // P3.25 salida PWM (PWM1.2) y LED 1
    LPC_PINCON->PINSEL7|=(3<<20); // P3.26 salida PWM (PWM1.3) y Led 2
    LPC_PINCON->PINSEL3|=(2<<14); // P1.23 como salida PWM para servo (PWM1.4)
    LPC_SC->PCONP|=(1<<6);
    LPC_PWM1->MR0=Fpclk*Tpwmm-1;
    LPC_PWM1->PCR|=(3<<10); //configurado el ENA2 (1.2),ENA3 (1.3),ENA4 (1.4)
    LPC_PWM1->PCR|=(1<<12);
    LPC_PWM1->MCR|=(1<<1);
    LPC_PWM1->TCR|=(1<<0)|(1<<3);
}

/*****
* Function Name : MotorDerecha
* Description : Genera el nivel alto con un PWC que representa la velocidad de las ruedas
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void MotorDerecha(int8_t VelDerecha) // LED1 para la rueda Izquierda
{
    LPC_PWM1->MR2=((Fpclk*Tpwmm-1)*abs(VelDerecha)/100); // TH PWM1.2
    LPC_PWM1->LER|=(1<<2)|(1<<0); //le pasamos un 3 porque usamos el MR2 y MR0
    Dato_motorDer();
    if (VelDerecha > 0){
        LPC_GPIO1->FIOCLR |= (1<<20); // Low el pin P1.20
        LPC_GPIO1->FIOSET |= (1<<21); // Hight el pin P1.21
    }
    else{
        LPC_GPIO1->FIOSET |= (1<<20); // Hight el pin P1.20
        LPC_GPIO1->FIOCLR |= (1<<21); // Low el pin P1.21
    }
}

/*****
* Function Name : MotorIzquierda
* Description : Genera el nivel alto con un PWC que representa la velocidad de las ruedas
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void MotorIzquierda(int8_t VelIzquierda) // LED2 para la rueda izquierda
{
    LPC_PWM1->MR3=((Fpclk*Tpwmm-1)*abs(VelIzquierda)/100); // TH PWM1.3
    LPC_PWM1->LER|=(1<<3)|(1<<0); //le pasamos un 4 porque usamos el MR3 y MR0
    Dato_motorIzq();
    if (VelIzquierda > 0){
        LPC_GPIO1->FIOCLR |= (1<<25); // Low el pin P1.24
        LPC_GPIO1->FIOSET |= (1<<24); // Hight el pin P1.25
    }
    else{
        LPC_GPIO1->FIOSET |= (1<<25); // Hight el pin P1.24
        LPC_GPIO1->FIOCLR |= (1<<24); // Low1; el pin P1.25
    }
}

/*****
* Function Name : PuenteH
* Description : Configura Señales del puente en en H y los LED
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PuenteH (void)
{
    LPC_GPIO1->FIODIR |= (1<<20);
    LPC_GPIO1->FIODIR |= (1<<21);
    LPC_GPIO1->FIODIR |= (1<<24);
    LPC_GPIO1->FIODIR |= (1<<25);
}
```

```

/*****
* Function Name   : setServo
* Description    : Actualiza el valor de la señal PWM
* Input         : grados - Debe tomar valores de -90 a 90
* Output        : None
* Return        : None
* Attention      : None
*****/
void setServo(int8_t grados) {
    LPC_PWM1->MR4=(Fpclk*1.5e-3 + Fpclk*1e-3*grados/90); // TH
    LPC_PWM1->LER=(1<<4) | (1<<0);
    Dato_motorServo();
}

```

### c. TP\_simple.c

```

#include <tp_simple.h>
#include <stdbool.h>
#include "Net_Config.h"

#define MY_IP localm[NETIF_ETH].IpAddr
extern LOCALM localm[];

/* Variable que contiene el dato del programa */
extern int8_t VelDer;
extern int8_t VelIzg;
extern int8_t Posserv;
extern int distancia;
extern uint8_t Estado;

//Variables mando nunchuck
extern int8_t x;           // coordenada x
extern int8_t y;           // coordenada y
extern int8_t acelerometro_x;
extern int8_t acelerometro_y;
extern int8_t acelerometro_z;
extern bool flagContadorC;
extern bool flagContadorZ;

//Variables sensores
uint32_t SLuz1 = 0;
uint32_t SLuz2 = 0;
uint32_t SPromedio = 0;
//Variables auxiliares
uint8_t messageText[25+1] = {"Esto es una prueba 1000"};
int8_t aux1,aux2,aux3, aux4, aux5,aux6,aux7,aux8,aux9,aux10,aux11,aux12,aux13,aux14;
bool dep_nunchuck= true;

/* Definicion de las diferentes zonas de la pantalla */
//Modo Manual & Automatico
struct t_screenZone Cabecera = { 10, 20, 220, 40, 0}; /*Mensaje "Bienvenida" */
struct t_screenZone subCabecera = { 10, 60, 220, 40, 0}; /*Elija el modo*/
struct t_screenZone Manual = { 10, 100, 220, 40, 0}; /*Modo Manual*/
struct t_screenZone Automatico = { 10, 140, 220, 40, 0}; /*Modo Automatico*/
struct t_screenZone Depuracion = { 10, 180, 220, 40, 0}; /*Modo Depuracion*/
struct t_screenZone Infor = { 10, 60, 220, 30, 0}; /*Informacion*/
struct t_screenZone VDer = { 10, 90, 150, 30, 0}; /*Vel Derecha*/
struct t_screenZone DatDer = { 160, 90, 70, 30, 0}; /*Datos Vel Derecha*/
struct t_screenZone VIzq = { 10, 120, 150, 30, 0}; /*Vel Izquierda */
struct t_screenZone DatIzq = { 160, 120, 70, 30, 0}; /*Datos Vel Izquierda */
struct t_screenZone PServ = { 10, 150, 150, 30, 0}; /*Posicion Servo */
struct t_screenZone DatServ = { 160, 150, 70, 30, 0}; /*Datos posicion servo */
struct t_screenZone Dist = { 10, 180, 150, 30, 0}; /*Umbral Distancia */
struct t_screenZone DatDist = { 160, 180, 70, 30, 0}; /*Datos Umbral Distancia */
struct t_screenZone Luz1 = { 10, 210, 150, 30, 0}; /*Intensidad Luz sensor 1 */
struct t_screenZone DatLuz1 = { 160, 210, 70, 30, 0}; /*Datos Intensidad Luz sensor 1 */
struct t_screenZone Luz2 = { 10, 240, 150, 30, 0}; /*Intensidad Luz sensor 2 */
struct t_screenZone DatLuz2 = { 160, 240, 70, 30, 0}; /*Datos Intensidad Luz sensor 2 */
struct t_screenZone Promedio = { 10, 270, 150, 30, 0}; /*Promedio Intensidad Luz */
struct t_screenZone DatPromedio = { 160, 270, 70, 30, 0}; /*Datos Promedio */

//zona modo Depuracion
struct t_screenZone DVDer = { 10, 90, 90, 30, 0}; /*Vel Derecha*/
struct t_screenZone DDatDer = { 100, 90, 40, 30, 0}; /*Datos Vel Derecha*/
struct t_screenZone zone_1 = { 140, 90, 45, 30, 0}; /*Incrementar Vel Derecha*/
struct t_screenZone zone_2 = { 185, 90, 45, 30, 0}; /*Decrementar Vel Derecha*/
struct t_screenZone DVIZq = { 10, 120, 90, 30, 0}; /*Vel Izquierda */
struct t_screenZone DDatIzq = { 100, 120, 40, 30, 0}; /*Datos Vel Izquierda */
struct t_screenZone zone_3 = { 140, 120, 45, 30, 0}; /*Incrementar Vel Izquierda*/
struct t_screenZone zone_4 = { 185, 120, 45, 30, 0}; /*Decrementar Vel Izquierda*/
struct t_screenZone DPServ = { 10, 150, 90, 30, 0}; /*Posicion Servo */
struct t_screenZone DDatServ = { 100, 150, 40, 30, 0}; /*Datos posicion servo */
struct t_screenZone zone_5 = { 140, 150, 45, 30, 0}; /*Mover serv Derecha*/
struct t_screenZone zone_6 = { 185, 150, 45, 30, 0}; /*Mover servo Izquierda*/
struct t_screenZone DDist = { 10, 180, 170, 30, 0}; /*Umbral Distancia */
struct t_screenZone DDatDist = { 180, 180, 50, 30, 0}; /*Datos Umbral Distancia */
struct t_screenZone DLuz1 = { 10, 210, 170, 30, 0}; /*Intensidad Luz sensor 1 */
struct t_screenZone DDatLuz1 = { 180, 210, 50, 30, 0}; /*Datos Intensidad Luz sensor 1 */
struct t_screenZone DLuz2 = { 10, 240, 170, 30, 0}; /*Intensidad Luz sensor 2 */
struct t_screenZone DDatLuz2 = { 180, 240, 50, 30, 0}; /*Datos Intensidad Luz sensor 2 */
struct t_screenZone DPromedio = { 10, 270, 170, 30, 0}; /*Promedio Intensidad Luz */
struct t_screenZone DDatPromedio = { 180, 270, 50, 30, 0}; /*Datos Promedio */
struct t_screenZone IP = { 10, 60, 70, 30, 0}; /*Ip */
struct t_screenZone DatIP = { 80, 60, 150, 30, 0}; /*Valor IP */
struct t_screenZone zone_7 = { 140, 90, 90, 30, 0}; /*Mover servo Izquierda*/

/* Flag que indica si se detecta una pulsación válida */
uint8_t pressedTouchPanel = 0;

/* Variable temporal donde almacenar cadenas de caracteres */
char texto1[25];
char texto2[25];
char texto3[25];
char texto4[25];
char texto5[25];
char texto6[25];
char texto7[25];

```

```

/*****
* Function Name : squareButton
* Description : Dibuja un cuadrado en las coordenadas especificadas colocando
* un texto en el centro del recuadro
* Input : zone: zone struct
* text: texto a representar en el cuadro
* textColor: color del texto
* lineColor: color de la línea
* Output : None
* Return : None
* Attention : None
*****/
void squareButton(struct t_screenZone* zone, char * text, uint16_t textColor, uint16_t lineColor)
{
    LCD_DrawLine( zone->x, zone->y, zone->x + zone->size_x, zone->y, lineColor);
    LCD_DrawLine( zone->x, zone->y, zone->x, zone->y + zone->size_y, lineColor);
    LCD_DrawLine( zone->x, zone->y + zone->size_y, zone->x + zone->size_x, zone->y + zone->size_y, lineColor);
    LCD_DrawLine( zone->x + zone->size_x, zone->y, zone->x + zone->size_x, zone->y + zone->size_y, lineColor);
    GUI_Text(zone->x + zone->size_x/2 - (strlen(text)/2)*8, zone->y + zone->size_y/2 - 8,
            (uint8_t*) text, textColor, Black);
}

/*****
* Function Name : drawMinus
* Description : Draw a minus sign in the center of the zone
* Input : zone: zone struct
* lineColor
* Output : None
* Return : None
* Attention : None
*****/
void drawMinus(struct t_screenZone* zone, uint16_t lineColor)
{
    LCD_DrawLine( zone->x + 5, zone->y + zone->size_y/2 - 1,
        zone->x + zone->size_x-5, zone->y + zone->size_y/2 - 1,
        lineColor);
    LCD_DrawLine( zone->x + 5, zone->y + zone->size_y/2,
        zone->x + zone->size_x-5, zone->y + zone->size_y/2,
        lineColor);
    LCD_DrawLine( zone->x + 5, zone->y + zone->size_y/2 + 1,
        zone->x + zone->size_x-5, zone->y + zone->size_y/2 + 1,
        lineColor);
}

/*****
* Function Name : drawMinus
* Description : Draw a minus sign in the center of the zone
* Input : zone: zone struct
* lineColor
* Output : None
* Return : None
* Attention : None
*****/
void drawAdd(struct t_screenZone* zone, uint16_t lineColor)
{
    drawMinus(zone, lineColor);

    LCD_DrawLine( zone->x + zone->size_x/2 - 1, zone->y + 5,
        zone->x + zone->size_x/2 - 1, zone->y + zone->size_y - 5,
        lineColor);
    LCD_DrawLine( zone->x + zone->size_x/2, zone->y + 5,
        zone->x + zone->size_x/2, zone->y + zone->size_y - 5,
        lineColor);
    LCD_DrawLine( zone->x + zone->size_x/2 + 1, zone->y + 5,
        zone->x + zone->size_x/2 + 1, zone->y + zone->size_y - 5,
        lineColor);
}

/*****
* Function Name : PantallaInicio
* Description : Visualiza el menu de inicio
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PantallaInicio(void)
{
    squareButton(&Cabecera, "BIENVENIDO USUARIO", White, Blue);
    squareButton(&subCabecera, "Eliga el modo:", White, Blue);
    squareButton(&Manual, "Manual", White, Blue);
    squareButton(&Automatico, "Automatico", White, Blue);
    squareButton(&Depuracion, "Depuracion", White, Blue);
}

/*****
* Function Name : PantallaManual
* Description : Visualiza la informacion a mostra en el modo manual
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PantallaManual(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(&Cabecera, "MODO MANUAL", White, Blue);
    squareButton(&Infor, "Datos:", White, Blue);
    squareButton(&VDer, "Vel Derecha", White, Blue);
    squareButton(&DatDer, " ", White, Blue);
    squareButton(&VIZq, "Vel Izquierda", White, Blue);
    squareButton(&DatIzq, " ", White, Blue);
    squareButton(&PServ, "Pos. Servo", White, Blue);
    squareButton(&DatServ, " ", White, Blue);
    squareButton(&Dist, "Distancia", White, Blue);
}

```

# SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 70

## DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

squareButton(&DatDist, "      ", White, Blue);
squareButton(&Luz1, "Intensidad Luz1", White, Blue);
squareButton(&DatLuz1, "      ", White, Blue);
squareButton(&Luz2, "Intensidad Luz2", White, Blue);
squareButton(&DatLuz2, "      ", White, Blue);
squareButton(&Promedio, "Prom. Intensidad", White, Blue);
squareButton(&DatPromedio, "      ", White, Blue);
//Dibujamos los Valores de los parametros
sprintf(texto1,"%2d", VelDer);
sprintf(texto2,"%2d", VelIzq);
sprintf(texto3,"%2d", PosServ);
sprintf(texto4,"%2d", distancia);
sprintf(texto5,"%2d", SLuz1);
sprintf(texto6,"%2d", SLuz2);
SPromedio= (SLuz1+SLuz2)/2;
sprintf(texto7,"%2d", SPromedio);

GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
    (uint8_t*) texto1, White, Black);
    GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
    (uint8_t*) texto2, White, Black);
    GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
    (uint8_t*) texto3, White, Black);
    GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
    (uint8_t*) texto4, White, Black);
    GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
    (uint8_t*) texto5, White, Black);
    GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
    (uint8_t*) texto6, White, Black);
    GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 -
8,
    (uint8_t*) texto7, White, Black);
}
/*****
* Function Name   : PantallaAutomatico
* Description     : Visualiza la informacion a mostra en el modo automatico
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PantallaAutomatico(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(&Cabecera, "MODO AUTOMATICO", White, Blue);
    squareButton(&Infor, "Datos:", White, Blue);
    squareButton(&VDer, "Vel Derecha", White, Blue);
    squareButton(&DatDer, "      ", White, Blue);
    squareButton(&VIzq, "Vel Izquierda", White, Blue);
    squareButton(&DatIzq, "      ", White, Blue);
    squareButton(&PServ, "Pos. Servo", White, Blue);
    squareButton(&DatServ, "      ", White, Blue);
    squareButton(&Dist, "Distancia", White, Blue);
    squareButton(&DatDist, "      ", White, Blue);
    squareButton(&Luz1, "Intensidad Luz1", White, Blue);
    squareButton(&DatLuz1, "      ", White, Blue);
    squareButton(&Luz2, "Intensidad Luz2", White, Blue);
    squareButton(&DatLuz2, "      ", White, Blue);
    squareButton(&Promedio, "Prom. Intensidad", White, Blue);
    squareButton(&DatPromedio, "      ", White, Blue);
    //Dibujamos los Valores de los parametros
    sprintf(texto1,"%2d", VelDer);
    sprintf(texto2,"%2d", VelIzq);
    sprintf(texto3,"%2d", PosServ);
    sprintf(texto4,"%2d", distancia);
    sprintf(texto5,"%2d", SLuz1);
    sprintf(texto6,"%2d", SLuz2);
    SPromedio= (SLuz1+SLuz2)/2;
    sprintf(texto7,"%2d", SPromedio);

    GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
        (uint8_t*) texto1, White, Black);
        GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
        (uint8_t*) texto2, White, Black);
        GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
        (uint8_t*) texto3, White, Black);
        GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
        (uint8_t*) texto4, White, Black);
        GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
        GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
        GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 -
8,
        (uint8_t*) texto7, White, Black);
}
/*****
* Function Name   : PantallaDepuracion
* Description     : Visualiza la informacion a mostra en el modo depuracion
* Input          : None
* Output         : None
* Return         : None
* Attention      : None
*****/
void PantallaDepuracion(void)
{
    //Dibujamos Zona de pantalla.
    squareButton(&Cabecera, "MODO DEPURACION", White, Blue);
    squareButton(&IP, "Dir. IP:", White, Blue);
    squareButton(&DatIP, "      ", White, Blue);

```

```

squareButton(&DVDer, "V.Derecha", White, Blue);
squareButton(&DDatDer, " ", White, Blue);
squareButton(&DVIZq, "V.Izquierd", White, Blue);
squareButton(&DDatIzq, " ", White, Blue);
squareButton(&DPServ, "Pos.Servo", White, Blue);
squareButton(&DDatServ, " ", White, Blue);
squareButton(&DDist, "Distancia", White, Blue);
squareButton(&DDatDist, " ", White, Blue);
squareButton(&DLuz1, "Intensidad Luz1", White, Blue);
squareButton(&DDatLuz1, " ", White, Blue);
squareButton(&DLuz2, "Intensidad Luz2", White, Blue);
squareButton(&DDatLuz2, " ", White, Blue);
squareButton(&DPromedio, "Promedio Int.", White, Blue);
squareButton(&DDatPromedio, " ", White, Blue);
squareButton(&zzone_7, " ", White, Blue);
drawAdd(&zzone_1, White);
drawAdd(&zzone_3, White);
drawAdd(&zzone_5, White);
drawMinus(&zzone_2, White);
drawMinus(&zzone_4, White);
drawMinus(&zzone_6, White);
//Dibujamos los Valores de los parametros
sprintf(texto1, "%2d", VelDer);
sprintf(texto2, "%2d", VelIzq);
sprintf(texto3, "%2d", Posserv);
sprintf(texto4, "%2d", distancia);
sprintf(texto5, "%2d", SLuz1);
sprintf(texto6, "%2d", SLuz2);
SPromedio = (SLuz1 + SLuz2) / 2;
sprintf(texto7, "%2d", SPromedio);

GUI_Text(DDatDer.x + DDatDer.size_x/2 - (strlen(texto1)/2)*8, DDatDer.y + DDatDer.size_y/2 - 8,
(uint8_t*) texto1, White, Black);
GUI_Text(DDatIzq.x + DDatIzq.size_x/2 - (strlen(texto2)/2)*8, DDatIzq.y + DDatIzq.size_y/2 - 8,
(uint8_t*) texto2, White, Black);
GUI_Text(DDatServ.x + DDatServ.size_x/2 - (strlen(texto3)/2)*8, DDatServ.y + DDatServ.size_y/2 - 8,
(uint8_t*) texto3, White, Black);
GUI_Text(DDatDist.x + DDatDist.size_x/2 - (strlen(texto4)/2)*8, DDatDist.y + DDatDist.size_y/2 - 8,
(uint8_t*) texto4, White, Black);
GUI_Text(DDatLuz1.x + DDatLuz1.size_x/2 - (strlen(texto5)/2)*8, DDatLuz1.y + DDatLuz1.size_y/2 - 8,
(uint8_t*) texto5, White, Black);
GUI_Text(DDatLuz2.x + DDatLuz2.size_x/2 - (strlen(texto6)/2)*8, DDatLuz2.y + DDatLuz2.size_y/2 - 8,
(uint8_t*) texto6, White, Black);
GUI_Text(DDatPromedio.x + DDatPromedio.size_x/2 - (strlen(texto7)/2)*8, DDatPromedio.y +
DDatPromedio.size_y/2 - 8,
(uint8_t*) texto7, White, Black);
}
/*****
* Function Name : PantallaDepuracion_Nunchuck
* Description : Visualiza la informacion de los valores del NunChuck
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void PantallaDepuracion_Nunchuck(void)
{
//Dibujamos Zona de pantalla.
squareButton(&Cabecera, "MOD0 DEPURACION", White, Blue);
squareButton(&IP, "Dir. IP:", White, Blue);
squareButton(&DatIP, " ", White, Blue);
squareButton(&VDer, "Eje X", White, Blue);
squareButton(&DatDer, " ", White, Blue);
squareButton(&VIZq, "Eje Y", White, Blue);
squareButton(&DatIzq, " ", White, Blue);
squareButton(&PServ, "Acelerometro X", White, Blue);
squareButton(&DatServ, " ", White, Blue);
squareButton(&Dist, "Acelerometro Y", White, Blue);
squareButton(&DatDist, " ", White, Blue);
squareButton(&Luz1, "Acelerometro Z", White, Blue);
squareButton(&DatLuz1, " ", White, Blue);
squareButton(&Luz2, "Boton C", White, Blue);
squareButton(&DatLuz2, " ", White, Blue);
squareButton(&DPromedio, "Boton Z", White, Blue);
squareButton(&DatPromedio, " ", White, Blue);

//Dibujamos los Valores de los parametros
sprintf(texto1, "%4d", x);
sprintf(texto2, "%4d", y);
sprintf(texto3, "%4d", acelerometro_x);
sprintf(texto4, "%4d", acelerometro_y);
sprintf(texto5, "%4d", acelerometro_z);
sprintf(texto6, "%4d", flagContadorC);
sprintf(texto7, "%4d", flagContadorZ);

GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
(uint8_t*) texto1, White, Black);
GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
(uint8_t*) texto2, White, Black);
GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
(uint8_t*) texto3, White, Black);
GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DDatDist.size_y/2 - 8,
(uint8_t*) texto4, White, Black);
GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
(uint8_t*) texto5, White, Black);
GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
(uint8_t*) texto6, White, Black);
GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 -
8,
(uint8_t*) texto7, White, Black);
}
/*****
* Function Name : checkTouchPanel

```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 72

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

* Description      : Lee el TouchPanel y almacena las coordenadas si detecta pulsación
* Input           : None
* Output          : Modifica pressedTouchPanel
*                 0 - si no se detecta pulsación
*                 1 - si se detecta pulsación
*                 En este caso se actualizan las coordenadas en la estructura display
* Return          : None
* Attention       : None
*****/
void checkTouchPanel(void)
{
    Coordinate* coord;

    coord = Read_Ads7846();

    if (coord > 0) {
        getDisplayPoint(&display, coord, &matrix );
        pressedTouchPanel = 1;
    }
    else
    {
        pressedTouchPanel = 0;

        // Esto es necesario hacerlo si hay dos zonas diferentes en
        // dos pantallas secuenciales que se solapen
        zone_1.pressed = 1;
        zone_2.pressed = 1;
        zone_3.pressed = 1;
        zone_4.pressed = 1;
        zone_5.pressed = 1;
        zone_6.pressed = 1;
        zone_7.pressed = 1;
        Cabecera.pressed = 1;
        subCabecera.pressed = 1;
        Manual.pressed = 1;
        Automatico.pressed = 1;
        Depuracion.pressed = 1;

        Infor.pressed = 1;
        VDer.pressed = 1;
        DatDer.pressed = 1;
        VIzq.pressed = 1;
        DatIzq.pressed = 1;
        PServ.pressed = 1;
        DatServ.pressed = 1;
        Dist.pressed = 1;
        DatDist.pressed = 1;
        Luz1.pressed = 1;
        DatLuz1.pressed = 1;
        Luz2.pressed = 1;
        DatLuz2.pressed = 1;
        Promedio.pressed = 1;
        DatPromedio.pressed = 1;

        DVDer.pressed = 1;
        DDatDer.pressed = 1;
        DVizq.pressed = 1;
        DDatIzq.pressed = 1;
        DPServ.pressed = 1;
        DDatServ.pressed = 1;
        DDist.pressed = 1;
        DDatDist.pressed = 1;
        DLuz1.pressed = 1;
        DDatLuz1.pressed = 1;
        DLuz2.pressed = 1;
        DDatLuz2.pressed = 1;
        DPromedio.pressed = 1;
        DDatPromedio.pressed = 1;
        IP.pressed = 1;
        DatIP.pressed = 1;
    }
}
*****/
* Function Name    : Dato_motorDer
* Description      : Imprime el dato del motor derecho
* Input           :
* Output          :
* Return          :
* Attention       : None
*****/
void Dato_motorDer(void)
{
    if (Estado== 3 || Estado==5)
    {
        if (aux1!=VelDer)
            squareButton(&DatDer, "      ", White, Blue);
        sprintf(textol,"%2d", VelDer);
        GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(textol)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
        (uint8_t*) textol, White, Black);
        aux1=VelDer;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if (aux1!=VelDer)
            squareButton(&DDatDer, "      ", White, Blue);
        sprintf(textol,"%2d", VelDer);
        GUI_Text(DDatDer.x + DDatDer.size_x/2 - (strlen(textol)/2)*8, DDatDer.y + DDatDer.size_y/2 - 8,
        (uint8_t*) textol, White, Black);
        aux1=VelDer;
    }
}
*****/
* Function Name    : Dato_motorIzq
* Description      : Imprime el dato del motor derecho

```



```

* Input      :
* Output     :
* Return     :
* Attention  : None
*****/
void Dato_motorIzq(void)
{
    if (Estado== 3 || Estado==5)
    {
        if (aux2!=VelIzq)
            squareButton(&DatIzq, " ", White, Blue);
        sprintf(texto2,"%2d", VelIzq);
        GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
            (uint8_t*) texto2, White, Black);
        aux2=VelIzq;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if (aux2!=VelIzq)
            squareButton(&DDatIzq, " ", White, Blue);
        sprintf(texto2,"%2d", VelIzq);
        GUI_Text(DDatIzq.x + DDatIzq.size_x/2 - (strlen(texto2)/2)*8, DDatIzq.y + DDatIzq.size_y/2 - 8,
            (uint8_t*) texto2, White, Black);
        aux2=VelIzq;
    }
}
*****/
* Function Name : Dato motorServo
* Description   : Imprime el dato del motor servo
* Input        :
* Output       :
* Return       :
* Attention    : None
*****/
void Dato_motorServo(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux3!=Posserv)
            squareButton(&DatServ, " ", White, Blue);
        sprintf(texto3,"%2d",Posserv);
        GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
            (uint8_t*) texto3, White, Black);
        aux3=Posserv;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux3!=Posserv)
            squareButton(&DDatServ, " ", White, Blue);
        sprintf(texto3,"%2d", Posserv);
        GUI_Text(DDatServ.x + DDatServ.size_x/2 - (strlen(texto3)/2)*8, DDatServ.y + DDatServ.size_y/2 - 8,
            (uint8_t*) texto3, White, Black);
        aux3=Posserv;
    }
}
*****/
* Function Name : Dato Distanciadm
* Description   : Imprime el dato del sensor ultrasonidos
* Input        :
* Output       :
* Return       :
* Attention    : None
*****/
void Dato_Distanciadm(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux4!=distancia)
            squareButton(&DatDist, " ", White, Blue);
        if (distancia>400)
            sprintf(texto4,"max");
        else
            sprintf(texto4,"%4d",distancia);
        GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
            (uint8_t*) texto4, White, Black);
        aux4=distancia;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux4!=distancia)
            squareButton(&DDatDist, " ", White, Blue);
        if (distancia>400)
            sprintf(texto4,"max");
        else
            sprintf(texto4,"%4d", distancia);
        GUI_Text(DDatDist.x + DDatDist.size_x/2 - (strlen(texto4)/2)*8, DDatDist.y + DDatDist.size_y/2 - 8,
            (uint8_t*) texto4, White, Black);
        aux4=distancia;
    }
}
*****/
* Function Name : Dato sensorLuz1
* Description   : Imprime el dato del sensor luz1
* Input        :
* Output       :
* Return       :
* Attention    : None
*****/

```

```

void Dato_sensorLuz1(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux5!=SLuz1)
            squareButton(&DatLuz1, " ", White, Blue);
        sprintf(texto5,"%3d",SLuz1);
        GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
            (uint8_t*) texto5, White, Black);
        aux5=SLuz1;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux5!=SLuz1)
            squareButton(&DDatLuz1, " ", White, Blue);
        sprintf(texto5,"%3d", SLuz1);
        GUI_Text(DDatLuz1.x + DDatLuz1.size_x/2 - (strlen(texto5)/2)*8, DDatLuz1.y + DDatLuz1.size_y/2 - 8,
            (uint8_t*) texto5, White, Black);

        aux5=SLuz1;
    }
}

/*****
* Function Name : Dato_sensorLuz2
* Description : Imprime el dato de luz 2
* Input :
* Output :
* Return :
* Attention : None
*****/
void Dato_sensorLuz2(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux6!=SLuz2)
            squareButton(&DatLuz2, " ", White, Blue);
        sprintf(texto6,"%3d",SLuz2);
        GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
            (uint8_t*) texto6, White, Black);
        aux6=SLuz2;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux6!=SLuz2)
            squareButton(&DDatLuz2, " ", White, Blue);
        sprintf(texto6,"%3d", SLuz2);
        GUI_Text(DDatLuz2.x + DDatLuz2.size_x/2 - (strlen(texto6)/2)*8, DDatLuz2.y + DDatLuz2.size_y/2 - 8,
            (uint8_t*) texto6, White, Black);

        aux6=SLuz2;
    }
}

/*****
* Function Name : Dato_promedioSensores
* Description : Imprime el dato promedio de los sensores de luz
* Input :
* Output :
* Return :
* Attention : None
*****/
void Dato_promedioSensores(void)
{
    if (Estado== 3 || Estado==5)
    {
        if(aux7!=SPromedio)
            squareButton(&DatPromedio, " ", White, Blue);
        sprintf(texto7,"%3d",SPromedio);
        GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
            (uint8_t*) texto7, White, Black);
        aux7=SPromedio;
    }
    else if(Estado==7 && dep_nunchuck == false)
    {
        if(aux7!=SPromedio)
            squareButton(&DDatPromedio, " ", White, Blue);
        sprintf(texto7,"%3d", SPromedio);
        GUI_Text(DDatPromedio.x + DDatPromedio.size_x/2 - (strlen(texto7)/2)*8, DDatPromedio.y +
            DDatPromedio.size_y/2 - 8,
            (uint8_t*) texto7, White, Black);

        aux7=SPromedio;
    }
}

/*****
* Function Name : screenMessageIP
* Description : Visualiza la pantalla de mensajes
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void screenMessageIP(void)
{
    sprintf((char *)messageText," %d.%d.%d.%d ", MY_IP[0], MY_IP[1],
        MY_IP[2], MY_IP[3]);
    squareButton(&DatIP, (char*)messageText, Red , Blue);
}

/*****
* Function Name : zonePressed

```

# SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 75

## DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

* Description      : Detecta si se ha producido una pulsación en una zona concreta
* Input           : zone: Estructura con la información de la zona
* Output          : Modifica zone->pressed
*                 0 - si no se detecta pulsación en la zona
*                 1 - si se detecta pulsación en la zona
* Return          : 0 - si no se detecta pulsación en la zona
*                 1 - si se detecta pulsación en la zona
* Attention       : None
*****/
int8_t zonePressed(struct t_screenZone* zone)
{
    if (pressedTouchPanel == 1) {

        if ((display.x > zone->x) && (display.x < zone->x + zone->size_x) &&
            (display.y > zone->y) && (display.y < zone->y + zone->size_y))

        {
            zone->pressed = 1;
            return 1;
        }

        zone->pressed = 0;
        return 0;
    }

    /*****
    * Function Name   : zoneNewPressed
    * Description     : Detecta si se ha producido el flanco de una nueva pulsación en
    *                 una zona concreta
    * Input          : zone: Estructura con la información de la zona
    * Output         : Modifica zone->pressed
    *                 0 - si no se detecta pulsación en la zona
    *                 1 - si se detecta pulsación en la zona
    * Return         : 0 - si no se detecta nueva pulsación en la zona
    *                 1 - si se detecta una nueva pulsación en la zona
    * Attention      : None
    *****/
    int8_t zoneNewPressed(struct t_screenZone* zone)
    {
        if (pressedTouchPanel == 1) {

            if ((display.x > zone->x) && (display.x < zone->x + zone->size_x) &&
                (display.y > zone->y) && (display.y < zone->y + zone->size_y))

            {
                if (zone->pressed == 0)
                {
                    zone->pressed = 1;
                    return 1;
                }

                return 0;
            }

            zone->pressed = 0;
            return 0;
        }

        /*****
        * Function Name   : modoDepuracion
        * Description     : Incrementa valores servo en pantalla
        * Input          :
        * Output         :
        * Return         :
        * Attention      : None
        *****/
        void modoDepuracion(void)
        {
            //Cambio para visualizar datos del nunchuck
            if(zonePressed(&Cabecera))
            {
                Estado= 6;
            }
            //Actualizamos datos del nunchuck
            if(dep_nunchuck== true)
            {
                if(aux8!=x)
                    squareButton(&DatDer, "      ", White, Blue);
                sprintf(texto1,"%4d",x);
                GUI_Text(DatDer.x + DatDer.size_x/2 - (strlen(texto1)/2)*8, DatDer.y + DatDer.size_y/2 - 8,
                    (uint8_t*) texto1, White, Black);
                aux8=x;

                if(aux9!=y)
                    squareButton(&DatIzq, "      ", White, Blue);
                sprintf(texto2,"%4d",y);
                GUI_Text(DatIzq.x + DatIzq.size_x/2 - (strlen(texto2)/2)*8, DatIzq.y + DatIzq.size_y/2 - 8,
                    (uint8_t*) texto2, White, Black);
                aux9=y;

                if(aux10!=acelerometro_x)
                    squareButton(&DatServ, "      ", White, Blue);
                sprintf(texto3,"%4d",acelerometro_x);
                GUI_Text(DatServ.x + DatServ.size_x/2 - (strlen(texto3)/2)*8, DatServ.y + DatServ.size_y/2 - 8,
                    (uint8_t*) texto3, White, Black);
                aux10=acelerometro_x;

                if(aux11!=acelerometro_y)
                    squareButton(&DatDist, "      ", White, Blue);
                sprintf(texto4,"%4d",acelerometro_y);
                GUI_Text(DatDist.x + DatDist.size_x/2 - (strlen(texto4)/2)*8, DatDist.y + DatDist.size_y/2 - 8,
                    (uint8_t*) texto4, White, Black);
                aux11=acelerometro_y;
            }
        }
    }
}

```

```

        if(aux12!=acelerometro_z)
            squareButton(&DatLuz1, "          ", White, Blue);
            sprintf(texto5,"%4d",acelerometro_z);
GUI_Text(DatLuz1.x + DatLuz1.size_x/2 - (strlen(texto5)/2)*8, DatLuz1.y + DatLuz1.size_y/2 - 8,
        (uint8_t*) texto5, White, Black);
        aux12=acelerometro_z;

        if(aux13!=flagContadorC)
            squareButton(&DatLuz2, "          ", White, Blue);
            sprintf(texto6,"%4d",flagContadorC);
GUI_Text(DatLuz2.x + DatLuz2.size_x/2 - (strlen(texto6)/2)*8, DatLuz2.y + DatLuz2.size_y/2 - 8,
        (uint8_t*) texto6, White, Black);
        aux13=flagContadorC;

        if(aux14!=flagContadorZ)
            squareButton(&DatPromedio, "          ", White, Blue);
            sprintf(texto7,"%4d",flagContadorZ);
GUI_Text(DatPromedio.x + DatPromedio.size_x/2 - (strlen(texto7)/2)*8, DatPromedio.y + DatPromedio.size_y/2 - 8,
        (uint8_t*) texto7, White, Black);
        aux14=flagContadorZ;
    }
    else
    {
        if (zonePressed(&zone_1))
            VelDer++;
        if (zonePressed(&zone_3))
            VelIzq++;
        if (zonePressed(&zone_5))
            Posserv++;
        if (zonePressed(&zone_2))
            VelDer--;
        if (zonePressed(&zone_4))
            VelIzq--;
        if (zonePressed(&zone_6))
            Posserv--;

        Dato_motorDer();
        Dato_motorIzq();
        Dato_motorServo();
    }
}

```

### d. Nunchuck.c

```
#include <LPC17xx.H>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "i2c_lpc17xx.h"
#include "Nunchuck.h"
#include "PWM.h"

// Variable de velocidad de la ruedas
extern uint8_t VelDer;
extern uint8_t VelIzq;
extern int8_t Posserv;
int ejex,ejey;

//Variable de datos del mando
int8_t x; // coordenada x
int8_t y; // coordenada y
int8_t acelerometro_x, acelerometro_y, acelerometro_z;
int8_t resto;
extern bool flagContadorC, flagContadorZ, flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores;
extern int contadorZ, contadorC, contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioMas2segBotones, flagCambioMenos2segBotones;
extern int contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioPulsandoC;
extern int contadorCambioPulsandoC;
extern bool modobluetooth;

//-----

void Nunchuck_Init(){
    I2Cdelay();
    I2Cdelay();

    // Enviar commando 0x52 para configurar el Nunchuck
    I2CSendAddr(0x52,0); // I2C Address del Nunchuck, WRITE
    I2CSendByte(0xF0); // Access Config Command
    I2CSendByte(0x55); // Continuous Conversion
    I2CSendStop();

    I2Cdelay(); // Para propósitos de
    simulacion
    I2Cdelay();

    // Enviar commando 0xFB para arrancar conversion del Nunchuck
    I2CSendAddr(0x52,0); // I2C Address del Nunchuck, WRITE
    I2CSendByte(0xFB); // Start Conversion
    I2CSendByte(0x00);
    I2CSendStop();
}

//-----

void Nunchuck_read(){
    // Enviar commando 0x52 para leer Nunchuck
    I2CSendAddr(0x52,0); // I2C Address del Nunchuck, WRITE
    I2CSendByte(0x0); // Read
    I2CSendStop();

    I2Cdelay();

    // Leer datos mando
    I2CSendAddr(0x52,1); // I2C Address, READ
    x = I2CGetByte(0); // Read MSB Byte, ACK : Esto es la coordenada x
    y = I2CGetByte(0);
    acelerometro_x = I2CGetByte(0);
    acelerometro_y = I2CGetByte(0);
    acelerometro_z = I2CGetByte(0);
    resto = I2CGetByte(1); //NACK

    if ((resto & 00000001) == 0x0) { // Boton Z
        flagContadorZ = true;
    }
    else {
        flagContadorZ = false;
    }
    if ((resto & 0x02) == 0x0) { // Boton C
        flagContadorC = true;
    }
    else {
        flagContadorC = false;
    }

    I2CSendStop();
}
```

```

/*****
* Function Name : MovimientoNunchuck
* Description : Saca los valores recogidos por el Nunchuck para controlar el robot
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void Movimiento_Nunchuck()
{
    if(modobluetooth==false)
    {
        if(y<0)
        {
            ejey= (128+y)*100/127;
        }
        else if( y>= 0)
        {
            ejey= (y-128)*100/128;
        }

        if(x<0)
        {
            ejex= (x+129)*100/128;
        }
        else if( x>= 0)
        {
            ejex= (x-127)*100/127;
        }

        if (ejey==0 && ejex==0)
        {
            VelDer = 0;
            VelIzq= 0;
        }
        else if(ejey!=0 &&ejex==0)
        {
            VelDer=ejey;
            VelIzq=ejey;
        }
        else if(ejey==0 &&ejex!=0)
        {
            VelDer=-ejex;
            VelIzq=ejex;
        }
        else if(ejey>0 &&ejex!=0)
        {
            VelDer=ejey/2-ejex/2;
            VelIzq=ejey/2+ejex/2;
        }
        else if(ejey<0 &&ejex!=0)
        {
            VelDer=ejey/2+ejex/2;
            VelIzq=ejey/2-ejex/2;
        }
    }
}

/*****
* Function Name : Servo_Nunchuck
* Description : Mueve el servo con la inclinacion del mando
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void Servo_Nunchuck()
{
    if (acelerometro_x>0)
        Posserv=(127-acelerometro_x)*90/62;

    else
        Posserv=-(128+acelerometro_x)*90/63;
}

```

### e. ADC.c

```
#include "ADC.h"
#include "Timer.h"

#define N_muestras 15872 // Igual a la longitud del array generado en Matlab

uint32_t canal_1, canal_2, canal_5;
extern uint32_t SLuz1, SLuz2, SPromedio;
uint8_t muestras[15872];

uint8_t audio;
int ALARMA = 1;
int GRABANDO = 2;
int GRABADO = 3;
int FIN = 4;

void init_ADC_LDRs(void)
{
    LPC_SC->PCONP |= (1<<12); // Power ON
    LPC_PINCON->PINSEL1 |= (1<<16); // ( AD0.1) P0.24 //LUZ1(izquierda)
    LPC_PINCON->PINSEL3 |= (3<<30); // ( AD0.5) P1.31)
    //LUZ2(derecha)
    LPC_PINCON->PINMODE1 |= (2<<16); // Deshabilita
    // Deshabilita pullup/pulldown
    LPC_PINCON->PINMODE3 |= (2<<30); // CCLK/4 (Fpcl después del reset) (100 Mhz/4 = 25Mhz)
    LPC_SC->PCLKSEL0 |= (0<<8); // canales 0 y 1
    LPC_ADC->ADCR |= (1<<5) | (1<<1) // CLKDIV=1 (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz) MAXIMA FRECUENCIA
    // PDN=1
    LPC_ADC->ADINTEN = (1<<5); // Hab. interrupción fin de conversión
    del_ÚLTIMO canal (canal 5)
    NVIC_EnableIRQ(ADC_IRQn);
    NVIC_SetPriority(ADC_IRQn, 0); //Prioridad
}

void init_ADC_grabar(void)
{
    LPC_SC->PCONP |= (1<<12); // Power ON
    LPC_PINCON->PINSEL1 |= (1<<18); // ADC input= P0.25 (AD0.2)
    LPC_PINCON->PINMODE1 |= (2<<18); // Deshabilita pullup/pulldown*
    LPC_SC->PCLKSEL0 |= (0<<8); // CCLK/4 (Fpcl después del reset) (100 Mhz/4 = 25Mhz)
    LPC_ADC->ADCR = (1<<2) | // Canal 2
    (1<<8) | // CLKDIV=1
    (1<<21) |
    (1<<24); //
    Fclk_ADC=25Mhz /(1+1)= 12.5Mhz) MAXIMA FRECUENCIA
    // PDN=1
    Inicio de conversión con el Match 1 del Timer 1
    LPC_ADC->ADINTEN = (1<<2); // Hab. interrupción fin de conversión Canal 2
    NVIC_EnableIRQ(ADC_IRQn);
    NVIC_SetPriority(ADC_IRQn, 0); //Habilita la interrupcion
}

void ADC_IRQHandler(void)
{
    static uint16_t indice_muestra;
    if (audio == GRABANDO) {
        canal_2 = ((LPC_ADC->ADDR2 >> 8) & 0xFF); // se borra automat. el flag DONE al leer ADCGDR
        muestras[indice_muestra++] = canal_2; //almacenamos la muestra tomada
        LPC_ADC->ADCR |= (1<<24); // START = ON
        NVIC_DisableIRQ(ADC_IRQn);
        if (indice_muestra == N_muestras-1) {
            indice_muestra = 0; // Reiniciamos el indice de muestras
            LPC_ADC->ADCR = (0<<24); // Paramos ADC
            init_ADC_LDRs(); // Para iniciar la configuracion del timer para las LDRs
            LPC_TIM2->TCR = 0x01; // Activa timer de LDRs
            audio = FIN;
        }
    }
    else {
        LPC_ADC->ADCR &= ~(1<<16);
        canal_1 = ((LPC_ADC->ADDR1 >> 4) & 0xFFFF); // flag DONE se borra automat. al leer ADDR1
        canal_5 = ((LPC_ADC->ADDR5 >> 4) & 0xFFFF); // flag DONE se borra automat. al leer ADDR0
        SLuz1 = ((canal_1/100)*2.5); // 12 bits -> 4096
        SLuz2 = ((canal_5/100)*2.5); // 12 bits -> 4096
        SPromedio = (SLuz1 + SLuz2)/2;
    }
}
```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 80

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

/*****
* Function Name   : Audio
* Description    : En funcion de si queremos reproducir o grabar inicializa el timer 1 y el ADC
* Input         : audioElegido:  ALARMA   - Reproduce Alarma
*               :               GRABANDO - Graba audio
*
*               GRABADO - Reproduce audio grabado
* Output        : None
* Return        : None
* Attention     : None
*****/
void Audio(int Opcion) {
    if (Opcion == ALARMA) {
        audio = ALARMA;
        init_TIMER1_reproducir();
        LPC_TIM1->TCR=0x01;
    }
    if (Opcion == GRABANDO) {
        audio = GRABANDO;
        init_TIMER1_grabar();
        LPC_TIM1->TCR=0x01;
        init_ADC_grabar();
        LPC_TIM2->TCR = 0x02; //Paramos el timer2 para que no haya modo burst y que solo se convierta el canal
    }
    de grabacion
    }
    if (Opcion == GRABADO) {
        audio = GRABADO;
        init_TIMER1_reproducir();
        LPC_TIM1->TCR=0x01;
    }
}

```



### f. DAC.c

```
#include <LPC17xx.H>
#include <stdio.h>
#include <string.h>
#include "DAC.h"

void init_DAC(void)
{
    LPC_PINCON->PINSEL1|= (2<<20);           // DAC output = P0.26 (AOUT)
    LPC_PINCON->PINMODE1|= (2<<20);           // Deshabilita pullup/pulldown
    LPC_DAC->DACCTRL=0;
}
```

### g. Interrupciones.c

```
#include <stdio.h>
#include <string.h>
#include <Math.h>
#include <LPC17xx.H>
#include "ADC.h"
#include "timer.h"
#include "Interrupciones.h"
#include "stdbool.h"
#include "tp_simple.h"

extern uint8_t Estado;
extern int contadorKey1;
extern int contadorKey2;
extern bool flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores, flagContadorKey1, flagContadorKey2;

extern int ALARMA;
extern int GRABANDO;
extern int GRABADO;

void init_EINT(void)
{
    // Configuración interrupciones externas
    LPC_PINCON->PINSEL4|=(1<<22);           // P2.11 es entrada interrup. EXT 1 (pulsador key1 en mini-dk2)
    LPC_PINCON->PINSEL4|=(1<<24);           // P2.12 es entrada interrup. EXT 2 (pulsador key2 en Mini-DK2)
    LPC_SC->EXTMODE|=(1<<1)|(1<<2);        // Por Flanco,
    LPC_SC->EXTPOLAR|=(1<<1)|(1<<2);        //de subida
    NVIC_SetPriority(EINT2_IRQn, 6);         // Menor prioritaria!!! ; sin CMSIS: NVIC->IP[18]=(4<<3);
    NVIC_EnableIRQ(EINT2_IRQn);             // sin CMSIS: NVIC->ISER[0]=(1<<18);
    NVIC_SetPriority(EINT1_IRQn, 6);
    NVIC_EnableIRQ(EINT1_IRQn);
}

/*****
* Function Name : EINT1 y EINT2
* Description : Para grabacion y reproduccion de audio
* Input : None
* Output : None
* Return : None
* Attention : None
*****/
void EINT1_IRQHandler() // PARA GRABAR
{
    LPC_SC->EXTINT=(1<<1); // Borrar flag Externa 1
    if (Estado == 7 && flagCambioMas2segPulsadores== false && flagCambioMenos2segPulsadores== false &&
    flagContadorKey2==false){
        Audio(GRABANDO); //Para que el timer grabe
    }
}

void EINT2_IRQHandler() // PARA REPRODUCIR
{
    LPC_SC->EXTINT=(1<<2); // Borrar flag Externa 2
    if (Estado == 7 && flagCambioMas2segPulsadores== false && flagCambioMenos2segPulsadores== false &&
    flagContadorKey1==false){
        Audio(GRABADO); //Para que el timer
        reproduzca
    }
}
```

## *h Bluetooth.c*

```
#include <LPC17xx.H>
#include "uart.h"
#include <stdio.h>
#include "string.h"
#include "stdbool.h"
#include "stdlib.h"

char buffer[30]; // Buffer de recepción de 30 caracteres --> Utilizada en la ISR de recepción de la uart
char buffer_tx[30]; // Buffer de recepción de 30 caracteres
char *ptr_rx; // puntero de recepción
char *ptr_tx; // puntero de recepción
char rx_completa; // Flag de recepción de cadena que se activa a "1" al recibir la tecla return CR(ASCII=13)
char *ptr_tx; // puntero de transmisión
char tx_completa; // Flag de transmisión de cadena que se activa al transmitir el caracter null (fin de cadena)
char fin=0;
long auxbth=0;
int auxmuestras = 0;
int estado_uart =1;
uint8_t orden_preparada = 0;

//variables externas
extern uint8_t Estado;
extern int8_t VelDer;
extern int8_t VelIzg;
extern int8_t Posserv;
extern bool Giro90;
extern bool Giromenos90;
extern uint32_t SLuz1,SLuz2,SPromedio;
extern int distancia;
extern int bluetooth;
extern int tiempo;
int milisegundos= 0;
bool modobluetooth= false;

/*Funcion encargada de ejecutar la maquina de estados para el envio de mensajes*/
void MaquinaEstadosBluetooth(){
    switch(estado_uart){
        case 1: //envio del mensaje inicial
            ptr_rx=buffer;
            tx_cadena_UART0("Bienvenido al modo Bluetooth/USB indique su nombre\n\r");
            estado_uart = 2; //una vez que se ha indicado la trama a transmitir cambiamos de estado para que no
            la vuelva a repetir
            break;

        case 2:
            if(tx_completa == 1) //TRANSMISION DE LAS CADENAS(ESTADO SOLO PARA LA PRINCIPAL)
                estado_uart = 3; //en caso de que se haya transmitido toda empieza la recepcion
            break;

        case 3: //RECEPCION DE LAS CADENAS(ESTADO SOLO PARA LA PRINCIPAL)
            if(rx_completa == 0) //hasta que no se reciba todo nos mantenemos en este estado
                estado_uart = 3;

            if(rx_completa == 1){ //una vez que se recibe todo procedemos al estado general transmite-envia con
            ordenes en donde actuamos sobre el HW
                rx_completa = 0; //borramos flag
                estado_uart =4; //estado general
            }
            break;

        case 4: //PREPARAMOS LOS MENSAJES DE TRANSMISION GENERICOS
            if(orden_preparada == 0){
                tx_cadena_UART0("Elige modo:\n\r MOVER \n\r VISUALIZAR VALORES \n\r FIN \n\r");
                orden_preparada = 1;
            }

            estado_uart = 5;
            break;

        case 5: // TRANSMISION DE MENSAJES GENERICOS
            if(tx_completa == 1 && fin == 0 )
                estado_uart = 6; //una vez transmitido todo pasamos a la recepcion
            else if(tx_completa == 1 && fin ==1)
                estado_uart = 0; //ninguno
            break;

        case 6: //RECEPCION MENSAJES GENERICOS
            if(rx_completa == 0){
                estado_uart = 6; //se mantiene hasta que se completa
            }
            else if (rx_completa ==1){
                estado_uart = 4; //volvemos a preparar los mensajes para luego transmitirlos
                orden_preparada = 0;
                rx_completa = 0; //borramos flag

                //Segun lo recibido se actua sobre el HW
                if (strcmp (buffer, "MOVER\n\r") == 0) {
                    estado_uart=7; //Realizamos ordenes
                }
            }
        }
    }
}
```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 83

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

        Estado = 2;
        modobluetooth =true;
    }
    else if (strcmp (buffer, "VISUALIZAR VALORES\r") == 0)
        estado_uart =9; //Elegimos valores a visualizar
    else if (strcmp (buffer, "FIN\r") == 0){
        fin = 1;
        tx_cadena_UART0("FIN DEL PROGRAMA\n\r");
        estado_uart = 5;
    }
    else{ //en caso de ser un comando no reconocido
        tx_cadena_UART0("COMANDO ERRONEO\n\r");
        estado_uart = 5;
    }
}
break;

case 7: //ELIGE MOVIMIENTO DEL COCHE
    tx_cadena_UART0("Elige modo:\n\r ADELANTE \n\r ATRAS \n\r DERECHA \n\r IZQUIERDA \n\r VOLVER AL
MENU PREVIO\n\r ");
    Posserv=0;
    estado_uart = 8;
    break;

case 8://PROCESAMOR ORDEN DE MOVIMIENTO
    Posserv=0;
    if(rx_completa == 0){
        estado_uart = 8; //se mantiene hasta que se completa
    }
    else if (rx_completa ==1){
        estado_uart = 7; //volvemos a preparar los mensajes para luego transmitirlos
        orden_preparada = 0;
        rx_completa = 0; //borramos flag

        //Segun lo recibido se actua sobre el HW
        if (strcmp (buffer, "ADELANTE\r") == 0)
        {
            VelDer=75;
            VelIzq=75;
            estado_uart=7; //Realizamos ordenes
        }
        else if (strcmp (buffer, "ATRAS\r") == 0)
        {
            VelDer=-75;
            VelIzq=-75;
            estado_uart=7; //Realizamos ordenes
        }
        else if (strcmp (buffer, "DERECHA\r") == 0){
            estado_uart = 7;
            Giromenos90 =true;
        }
        else if (strcmp (buffer, "IZQUIERDA\r") == 0){
            estado_uart = 7;
            Giro90 =true;
        }
        else if (strcmp (buffer, "VOLVER AL MENU PREVIO\r") == 0){
            estado_uart = 4;
            modobluetooth=false;
        }
        else{ //en caso de ser un comando no reconocido
            tx_cadena_UART0("COMANDO ERRONEO\n\r");
            estado_uart = 7;
        }
    }
    break;

case 9: //ELIGE DATO A VISUALIZAR
    if(tx_completa == 0){
        estado_uart = 9; //se mantiene hasta que se completa
    }
    else if (tx_completa ==1){
        tx_cadena_UART0("INDIQUE EL NUMERO DE MUESTRAS A TOMAR:\n\r ");
        estado_uart = 10;
    }
    break;

case 10://PROCESAMOR ORDEN DE MOVIMIENTO
    if(rx_completa == 0){
        estado_uart = 10; //se mantiene hasta que se completa
    }
    else if (rx_completa ==1){
        estado_uart = 9; //volvemos a preparar los mensajes para luego transmitirlos
        orden_preparada = 0;
        rx_completa = 0; //borramos flag
        //Segun lo recibido se actua sobre el HW
        auxbth =strtol(buffer,&ptr,10);

        if (auxbth > 0 && auxbth <100){
            sprintf(buffer_tx, "TOMA DE MUESTRAS:\n\r");
            tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
            estado_uart = 11;
            auxmuestras= 8;
            tiempo =0;
        }
        else{ //en caso de ser un comando no reconocido
            tx_cadena_UART0("COMANDO ERRONEO\n\r");
            estado_uart = 9;
            tiempo =0;
        }
    }
    break;
case 11: //imprimimos muestras
    if(tx_completa == 0){
        estado_uart = 11; //se mantiene hasta que se completa
    }
}

```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 84

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

else if (tx_completa ==1){
    if (auxmuestras==8)
    {
        sprintf(buffer_tx, "Motor Derecha: %3d \n\r", VelDer);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==7)
    {
        sprintf(buffer_tx, "Motor Izquierda: %3d \n\r", VelIzq);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==6)
    {
        sprintf(buffer_tx, "Posicion Servo: %3d °\n\r", Posserv);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==5)
    {
        sprintf(buffer_tx, "Sensor Luz1: %3d \n\r", SLuz1);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==4)
    {
        sprintf(buffer_tx, "Sensor Luz2: %3d \n\r", SLuz2);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==3)
    {
        sprintf(buffer_tx, "Promedio: %3d \n\r", SPromedio);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==2)
    {
        sprintf(buffer_tx, "Distancia: %3d cm \n\r", distancia);
        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras--;
    }
else if (auxmuestras==1)
    {
        milisegundos=tiempo*10;
        auxbth--;
        sprintf(buffer_tx, "Tiempo de toma muestras:%d milisegundos

\n\r",milisegundos);

        tx_cadena_UART0(buffer_tx); //enviamos la cadena para visualizarla
        auxmuestras=8;
    }
    if(auxbth>0)
        estado_uart = 11;
    else
        estado_uart = 9;
    break;
}
}
}

```

## *i HTTP\_CGI.c*

```

/*-----
 *      RL-ARM - TCPnet
 *-----
 *      Name:      HTTP_CGI.C
 *      Purpose:   HTTP Server CGI Module
 *      Rev.:      V4.22
 *-----
 *      This code is part of the RealView Run-Time Library.
 *      Copyright (c) 2004-2011 KEIL - An ARM Company. All rights reserved.
 *-----*/

#include <Net_Config.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "stdbool.h"
#include "stdlib.h"

/* -----
 * The HTTP server provides a small scripting language.
 *
 * The script language is simple and works as follows. Each script line starts
 * with a command character, either "i", "t", "c", "#" or ".".
 *
 * "i" - command tells the script interpreter to "include" a file from the
 *       virtual file system and output it to the web browser.
 *
 * "t" - command should be followed by a line of text that is to be output
 *       to the browser.
 *
 * "c" - command is used to call one of the C functions from the this file.
 *       It may be followed by the line of text. This text is passed to
 *       'cgi_func()' as a pointer to environment variable.
 *
 * "#" - command is a comment line and is ignored (the "#" denotes a comment)
 *
 * "." - denotes the last script line.
 * -----*/

// VARIABLES NUESTRO PROGRAMA
extern int8_t VclDer;
extern int8_t VclIzg;
extern int8_t Posserv;
extern int distancia;
extern uint8_t Estado;
extern uint32_t SLuz1,SLuz2,SPromedio;

/* http_demo.c */
extern U16 AD_in (U32 ch);
extern U8  get_button (void);

/* at System.c */
extern LOCALM localm[];
#define LocM localm[NETIF_ETH]

/* Net_Config.c */
extern struct tcp_cfg tcp_cfg;
extern struct http_cfg http_cfg;
#define tcp_NumSocks tcp_cfg.NumSocks
#define tcp_socket tcp_cfg.Scb
#define http_EnAuth http_cfg.EnAuth
#define http_auth_passw http_cfg.Passw

extern BOOL LEDrun;
extern void LED_out (U32 val);
extern BOOL LCDupdate;
extern U8 lcd_text[2][16+1];
int auxhttp;
/* Local variables. */
static char const state[][9] = {
    "FREE",
    "CLOSED",
    "LISTEN",
    "SYN_REC",
    "SYN_SENT",
    "FINW1",
    "FINW2",
    "CLOSING",
    "LAST_ACK",
    "TWAIT",
    "CONNECT"};

/* My structure of CGI status U32 variable. This variable is private for */
/* each HTTP Session and is not altered by HTTP Server. It is only set to */
/* zero when the cgi_func() is called for the first time. */
typedef struct {
    U16 xcnt;
    U16 unused;
} MY_BUF;
#define MYBUF(p) ((MY_BUF *)p)

```

```

/*-----
 * HTTP Server Common Gateway Interface Functions
 *-----*/

/*----- cgi_process_var -----*/

void cgi_process_var (U8 *qs) {
    /* This function is called by HTTP server to process the Query_String */
    /* for the CGI Form GET method. It is called on SUBMIT from the browser. */
    /* The Query_String is SPACE terminated. */
    U8 *var;
    var = (U8 *)alloc_mem (40);

    do {
        /* Loop through all the parameters. */
        qs = http_get_var (qs, var, 40);
        /* Check the returned string, 'qs' now points to the next. */
        if (var[0] != 0) {
            /* Returned string is non 0-length. */
            if (str_scomp (var, "modo=manual") == __TRUE) {
                Estado=3;
            }
            if (str_scomp (var, "modo=automatico") == __TRUE) {
                Estado=5;
            }
            else if (str_scomp (var, "modo=depuracion") == __TRUE) {
                Estado=7;
            }

            if(str_scomp(var,"VelDer")==__TRUE){
                sscanf((const char*)&var[7], "%d", &auxhttp);
                VelDer =auxhttp;
            }

            else if(str_scomp(var,"VelIzq")==__TRUE){
                sscanf((const char*)&var[7], "%d", &auxhttp);
                VelIzq =auxhttp;
            }

            else if(str_scomp(var,"Posserv")==__TRUE){
                sscanf((const char*)&var[8], "%d", &auxhttp);
                Posserv =auxhttp;
            }

        }
    }while (qs);

    free_mem ((OS_FRAME *)var);
}

/*----- cgi_process_data -----*/

void cgi_process_data (U8 code, U8 *dat, U16 len) {
    /* This function is called by HTTP server to process the returned Data */
    /* for the CGI Form POST method. It is called on SUBMIT from the browser. */
    /* Parameters: */
    /* code - callback context code */
    /* 0 = www-url-encoded form data */
    /* 1 = filename for file upload (0-terminated string) */
    /* 2 = file upload raw data */
    /* 3 = end of file upload (file close requested) */
    /* 4 = any xml encoded POST data (single or last stream) */
    /* 5 = the same as 4, but with more xml data to follow */
    /* Use http_get_content_type() to check the content type */
    /* dat - pointer to POST received data */
    /* len - received data length */
    /*-----*/

    switch (code) {
        case 0:
            /* Url encoded form data received. */
            break;

        default:
            /* Ignore all other codes. */
            return;
    }
}

```

```

/*----- cgi_func -----*/

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
/* This function is called by HTTP server script interpreter to make a
/* formatted output for 'stdout'. It returns the number of bytes written
/* to the output buffer. Hi-bit of return value (len is or-ed with 0x8000)*/
/* is a repeat flag for the system script interpreter. If this bit is set
/* to 1, the system will call the 'cgi_func()' again for the same script
/* line with parameter 'pcgi' pointing to a 4-byte buffer. This buffer
/* can be used for storing different status variables for this function.
/* It is set to 0 by HTTP Server on first call and is not altered by
/* HTTP server for repeated calls. This function should NEVER write more
/* than 'buflen' bytes to the buffer.
/* Parameters:
/* env - environment variable string
/* buf - HTTP transmit buffer
/* buflen - length of this buffer (500-1400 bytes - depends on MSS)
/* pcgi - pointer to session local buffer used for repeated loops
/* This is a U32 variable - size is 4 bytes. Value is:
/* - on 1st call = 0
/* - 2nd call = as set by this function on first call
U32 len = 0;

switch (env[0]) {
/* Analyze the environment string. It is the script 'c' line starting
/* at position 2. What you write to the script file is returned here.
case '1' :
switch (env[2]) {
case 'a':
/* Write the local IP address. The format string is included
/* in environment string of the script line.
len = sprintf((char *)buf, (const char *)env[4], (Estado==2)? "Manual": (Estado==4)?
"Automatico": (Estado==5)? "Automatico": (Estado==6)? "Depuracion": (Estado==7)? "Depuración": "Espera");
break;
case 'b':
/* Write the local IP address. The format string is included
/* in environment string of the script line.
len = sprintf((char *)buf, (const char *)env[4], VelDer);
break;
case 'c':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], VelIzq);
break;
case 'd':
/* Write the local IP address. The format string is included
/* in environment string of the script line.
len = sprintf((char *)buf, (const char *)env[4], Posserv);
break;
case 'f':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], SLuz1);
break;
case 'g':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], SLuz2);
break;
case 'h':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], SPromedio);
break;
case 'i':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], distancia);
break;
case 'j':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], (Estado==3)? "checked='checked'":""");
break;
case 'k':
/* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], (Estado==5)? "checked='checked'":""");
break;
case 'l' /* Write local Net mask.
len = sprintf((char *)buf, (const char *)env[4], (Estado==7)? "checked='checked'":""");
break;
case 'm':
len=sprintf((char *)buf, (const char *)env[4], (Estado==7)? "<form action='/index.cgi'
method='get'><input style='text-align: center' name='VelDer' size='3' type='test' value=' '><input type='submit'
value='Actualizar'></form>":""");
break;
case 'n':
len=sprintf((char *)buf, (const char *)env[4], (Estado==7)? "<form action='/index.cgi'
method='get'><input style='text-align: center' name='VelIzq' size='3' type='test' value=' '><input type='submit'
value='Actualizar'></form>":""");
break;
case 'o':
len=sprintf((char *)buf, (const char *)env[4], (Estado==7)? "<form action='/index.cgi'
method='get'><input style='text-align: center' name='Posserv' size='3' type='test' value=' '>&ordm;C <input type='submit'
value='Actualizar'></form>":""");
break;
default:
break;
}
break;
default:
break;
}
return ((U16)len);
}
/*-----
* end of file
*-----*/

```

### j. WatchDog.c

```
#include <LPC17xx.H>
#include "WatchDog.h"

#define F_cpu 100e6 // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4 // Defecto despues del reset
#define F_wdclk F_pclk/4 // WDT clk

/*Funcion que ejecuta la secuencia de recarga del WDT*/
void WDT_Feed(void)
{
    LPC_WDT->WDFEED=0xAA;
    LPC_WDT->WDFEED=0x55;
}

void WDT_init(void)
{
    LPC_WDT->WDTTC= F_wdclk*8; // Timeout=8seg. //valor de recarga
    LPC_WDT->WDCLKSEL=0x01; // Clock=PCLK
    LPC_WDT->WDMOD=0x03; // Enable y Reset si vence el tiempo
    LPC_WDT->WDFEED=0xAA;
    LPC_WDT->WDFEED=0x55;
}
```

### k. StateChart.c

```
#include <tp.simple.h>
#include "Nunchuck.h"
#include "stdbool.h"
#include "timer.h"
#include "PWM.h"

/* Definición de los estados */
// Maquina de estados global
#define Inicio 0
#define Inicio_Espera 1
#define Modo_Manual 2
#define Modo_Manual_Espera 3
#define Modo_Automatico 4
#define Modo_Automatico_Espera 5
#define Modo_Depuracion 6
#define Modo_Depuracion_Espera 7
uint8_t Estado;
extern int umbralDistancia, distancia;
extern bool Barrido_Completo, Barrido;
extern int8_t Posserv;
//Variables Nunchuck

extern int8_t x; // coordenada x
extern int8_t y; // coordenada y
extern int8_t acelerometro_x, acelerometro_y, acelerometro_z;
extern int8_t resto;
extern bool flagContadorC, flagContadorZ, flagCambioMas2segPulsadores, flagCambioMenos2segPulsadores;
extern int contadorZ, contadorC, contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioMas2segBotones, flagCambioMenos2segBotones;
extern int contadorKey1, contadorKey2;
extern int contadorCambioMas2segPulsadores, contadorCambioMenos2segPulsadores;
extern bool flagCambioPulsandoC;
extern int contadorCambioPulsandoC;
extern bool dep_nunchuck;

//Definicion de las pulsacion de pantalla
extern struct t_screenZone Manual;
extern struct t_screenZone Automatico;
extern struct t_screenZone Depuracion;
extern struct t_screenZone zone_1;
extern struct t_screenZone zone_2;
extern struct t_screenZone zone_3;
extern struct t_screenZone zone_4;
extern struct t_screenZone zone_5;
extern struct t_screenZone zone_6;

void MaquinaEstados (void){
    checkTouchPanel();

    switch (Estado)
    {
        case Inicio : // Inicializamos contadores y cargamos pantalla de inicio y pasamos a modo espera
            LCD_Clear(Black);
            PantallaInicio();
            Estado = Inicio_Espera;
            break;

        case Inicio_Espera : //Elegimos modo en funcion de la zona que pulsemos
            if (zonePressed(&Manual)) {
                Estado = Modo_Manual;
            }
            if (zonePressed(&Automatico)) {
                Estado = Modo_Automatico;
            }
            if (zonePressed(&Depuracion)) {
                Estado = Modo_Depuracion;
            }
    }
}
```



```

    }
    break;

    case Modo_Manual :
        Posserv=0;
        //iniciamos la posicion de servo
        setServo(Posserv);
        contadoresreset(); //Ponemos contadores de botones del
mando nunchuck y pulsadores a 0
        LCD_Clear(Black); //Limpiamos pantalla
        PantallaManual(); //Imprimimos pantalla del modo manual
        Estado = Modo_Manual_Espera; //cambiamos a modo espera manual
        Barrido_Completo =false;
        Barrido= false;

    break;

    case Modo_Manual_Espera:
        if( distancia> umbralDistancia) //Funcion movimiento ruedas con el mando.
        Movimiento_Nunchuck(); //Funcion de movimiento del servo en funcion de
la inclinacion del mando.
        Servo_Nunchuck();
        controladorPulsadores(); //Funcion que controla Pulsadores Key1 y Key2
        if (flagCambioMas2segPulsadores || flagCambioMas2segBotones) { //Cambiamos de modo en
funcion de los pulsadores y votones
            Estado = Modo_Depuracion;
            Posserv=0;
        }
        if (flagCambioMenos2segPulsadores|| flagCambioMenos2segBotones) {
            Estado = Modo_Automatico;
        }

    break;

    case Modo_Automatico :
        Posserv=0;
        //iniciamos la posicion de servo
        setServo(Posserv);
        contadoresreset(); //Ponemos
contadores de botones del mando nunchuck y pulsadores a 0
        LCD_Clear(Black);
        //Limpiamos pantalla
        PantallaAutomatico(); //Imprimimos pantalla del
modo automatico
        Estado = Modo_Automatico_Espera; //cambiamos a modo espera automatico
        Barrido_Completo =false;
        Barrido= false;

    break;

    case Modo_Automatico_Espera :
        controladorPulsadores(); //Funcion que controla Pulsadores Key1
y Key2
        if (flagCambioMas2segPulsadores) { //cambiamos de modo en funcion de las keys y los pulsadores
            Estado = Modo_Depuracion;
            Posserv=0;
        }
        if(flagCambioPulsandoC)
            Estado = Modo_Manual;

    break;

    case Modo_Depuracion :
        Posserv=0;
        setServo(Posserv);
        contadoresreset();
        LCD_Clear(Black);
        if(dep_nunchuck==false) //En funcion de la variable mostramos el modo
depuracion con los datos de los sensores o bien con los valores de mando nunchuck
        {
            PantallaDepuracion_Nunchuck();
            dep_nunchuck=true;
            screenMessageIP();
        }
        else
        {
            PantallaDepuracion();
            dep_nunchuck=false;
            screenMessageIP();
        }
        Estado = Modo_Depuracion_Espera;
        Barrido_Completo =false;
        Barrido= false;

    break;

    case Modo_Depuracion_Espera:
        controladorPulsadores(); //controla los pulsadores
        modoDepuracion(); //funcion que cambia los valores a
visualizar cada vez que tocamos la cabecera
        if (flagCambioMas2segPulsadores) { // cambiamos de modo en funcion de los pulsadores.
            Estado = Modo_Manual;
        }
        if (flagCambioMenos2segPulsadores) {
            Estado = Modo_Automatico;
        }

    break;

    default:
        break;
}
}
}

```

## 1. Timer

```
#include <LPC17xx.H>
#include <glcd.h>
#include <TouchPanel.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Math.h>
#include "PWM.h"
#include "tp_simple.h"
#include "stdbool.h"
#include "ADC.h"
#include "Nunchuck.h"
#include "audio_muestras.h"
#include "uart.h"
//defenimos frecuencias
#define Th_trigger (10e-6*Fpclk_TIMx)
#define T_trigger (20e-3*Fpclk_TIMx)
#define Fcclk 100e6
#define Fpclk_TIMx (Fcclk/4) // En los defines con operaciones: recomendable (...)
#define F_muestreo 10 // Fs=10Hz (Cada 100 ms se toma una muestra del canal 0 y 2)
#define F_out 8000
#define Fpclk 25e6
#define N_muestras 15872 // igual a la longitud del array generado en Matlab

//defenimos variables externas
extern int8_t VelDer;
extern int8_t VelIzq;
extern int8_t Posserv;

int N, distancia;
int tiempo=0;
extern bool modobluetooth;
bool barridoluz =false;

extern uint8_t Estado;
extern uint32_t SLuz1,SLuz2,SPromedio;
int32_t valorluz =40;
bool Luz= false;

//variables interruptores
bool flagContadorKey1=false,flagContadorKey2=false, flagCambioMas2segPulsadores=false, flagCambioMenos2segPulsadores=false;
int contadorKey1=0, contadorKey2=0;

//Variables Botones mando wii
bool flagContadorC=false, flagContadorZ=false,flagCambioMas2segBotones=false, flagCambioMenos2segBotones=false;
int contadorC=0, contadorZ=0;
bool flagCambioPulsandoC=false;
extern uint8_t muestras[];
extern uint8_t audio;
extern int ALARMA;
extern int GRABANDO;
extern int GRABADO;
extern int FIN;

//Variables para marcar lecturas y movimientos
int Lectura_nunchuck = 0;
int GiroDer =0;
int GiroIzq=0;
bool Barrido = false;
bool Barrido_Completo=false;
bool Giro90= false;
bool Giromenos90= false;
int aux barrido = 0;
int aux Giro=0;
int aux=0;
int grados=0;
int umbralDistancia= 27;

// TIMER 0 PARA CONTADORES Y FUNCIONES CON CUENTA DE TIEMPOS MANDO NUNCHUCK, KEY1 Y KEY2, BARRIDO SERVO, GIROS DE COCHE
void init_TIMER0(void)
{
    LPC_SC->PCONP|=(1<<1); // Activa Timer0
    LPC_TIM0->PR = 24; // Para que cada tick sea de 1 Mhz (Fpclk/PR+1)
    LPC_TIM0->MCR = 0x3; // Interrupt on Match 0 and Reset TC
    LPC_TIM0->MR0 = 10000; // Periodo de interrupcion de TODOS los contadores 10ms
    LPC_TIM0->EMR = 1<<0 | 1<<5; // Match por MR0 y set con 10 en EMC0 (bits 4 y 5)
    LPC_TIM0->TCR = 0x01; // Arranca timer
    NVIC_SetPriority(TIMERO_IRQn,4); // Prioridad por encima del priority grouping
    NVIC_EnableIRQ(TIMERO_IRQn);
}

void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR|=(1<<1); // Borrar flag interrupción
    Lectura_nunchuck++;
    aux barrido++;
    aux Giro++;
    tiempo++;
    //LECTURA DEL MANDO NUNCHUCK
    if(Lectura_nunchuck == 19 && modobluetooth==false)
    {
        // Para que lea el Nunchuck cada segundo
        Lectura_nunchuck = 0;
        Nunchuck_read();
    }
    else if(Lectura_nunchuck == 19 && modobluetooth==true)
    {
    }
}
```

```

    Lectura_nunchuck = 0;
}
//BARRIDOS DEL SERVOMOTOR
if(aux_barrido == 49 && Barrido== true)
{
    Posserv += 10;
    setServo(Posserv);
    aux_barrido=0;
    if (SPromedio>valorluz)
    {
        aux_barrido=0;
        grados=Posserv;
        Posserv=0;
        setServo(Posserv);
        Barrido=false;
        barridoluz =true;
    }
    if(Posserv==90&&aux<1)
    {
        Posserv=-90;
        setServo(Posserv);
        aux++;
    }
    else if(Posserv==90 && aux==1) //Los dos barridos han sido completados
    {
        Posserv=0;
        setServo(Posserv);
        Barrido Completo= true;
        Barrido=false;
    }
}
else if(Barrido==false)
{
    aux_barrido=0;
}
//FUNCION DE GIROS
if(Giro90==true && Giromenos90==false )
{
    VelDer=70;
    VelIzq= 0;
}
else if(Giromenos90==true && Giro90==false)
{
    VelDer= 0;
    VelIzq= 70;
}
else if(Giromenos90==false && Giro90==false)
{
    aux_Giro=0;
}
if(aux_Giro == 60 &&(Giro90==true ||Giromenos90 ==true))
{
    VelDer=0;
    VelIzq=0;
    Giro90=false;
    Giromenos90= false;
}
if(aux_Giro>60)
{
    aux_Giro=0;
}
//ESTADO DE LOS PULSADORES Y BOTONES
if (flagContadorKey1)
{
    contadorKey1++;
}
else
{
    if (contadorKey1>199 && contadorKey2>199)
    {
        flagCambioMas2segPulsadores= true;
        contadorKey1=0;
    }
    else if(contadorKey1>0 && contadorKey2>0)
    {
        flagCambioMenos2segPulsadores= true;
        contadorKey1=0;
    }
    else
        contadorKey1=0;
}
if (flagContadorKey2)
    contadorKey2++;
else
{
    if (contadorKey1>199 && contadorKey2>199)
    {
        flagCambioMas2segPulsadores= true;
        contadorKey2=0;
    }
    else if(contadorKey1>0 && contadorKey2>0)
    {
        flagCambioMenos2segPulsadores= true;
        contadorKey2=0;
    }
    else
        contadorKey2=0;
}
if (flagContadorC)
    contadorC++;
else
{

```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 92

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

if (contadorC>199 && contadorZ>199)
{
    flagCambioMas2segBotones= true;
    contadorC=0;
    contadorZ=0;;
}
else if(contadorC>0 && contadorZ>0)
{
    flagCambioMenos2segBotones= true;
    contadorC=0;
    contadorZ=0;
}
else if(contadorC>199 && contadorZ==0)
{
    flagCambioPulsandoC= true;
    contadorC=0;
}
else if(contadorC>0 && contadorZ==0)
{
    contadorC=0;
    if(Estado==3)
        Audio(GRABANDO);
}
}

if (flagContadorZ)
    contadorZ++;
else
{
    if (contadorC>199 && contadorZ>199)
    {
        flagCambioMas2segBotones= true;
        contadorZ=0;
        contadorC=0;
    }
    else if(contadorC>0 && contadorZ>0)
    {
        flagCambioMenos2segBotones= true;
        contadorZ=0;
        contadorC=0;
    }
    else if(contadorZ>0 && contadorC==0)
    {
        contadorZ=0;
        if (Estado==3)
            Audio(GRABANDO);
    }
}
}

// TIMER 1 PARA TODO AUDIO.
/* Timer 1 en modo Output Compare (reset T0TC on Match 0)
Counter clk: 25 MHz          MAT1.0 : On match, salida de una muestra hacia el DAC */

void init_TIMER1(void)
{
    LPC_SC->PCONP|= (1<<2); // Activa timer 1
    LPC_TIM1->PR = 0; // Para que cada tick sea de 25 Mhz (Fpclk/PR+1)
    LPC_TIM1->MCR = 0x03; // Reset TC on Match, and Interrupt!
    LPC_TIM1->MR0 = Fpclk_TIMx/F_out-1; // MR1=25e6/(8KHz)-1 -----> Interrumpe cada 125 us
    LPC_TIM1->TCR = 0x02; // Timer STOP y RESET
    NVIC_EnableIRQ(TIM1_IRQn);
    NVIC_SetPriority(TIM1_IRQn,4); // Prioridad por encima del priority grouping
} // No hace falta EMR ni asignar un pin porque no queremos ningun match, solo reseteo

void TIM1_IRQHandler(void)
{
    //Relativas al audio

    static uint16_t indice_muestra;
    LPC_TIM1->IR|= (1<<0); // Borrar flag
    if (audio == ALARMA) {
        LPC_DAC->DACR= muestra[indice_muestra++]<<8; // 8 bits!!!!
    }
    else if(audio == GRABADO)
    {
        LPC_DAC->DACR= muestras[indice_muestra++]<<8; // 8 bits!!!!
    }
    else if(audio==GRABANDO)
    {
        LPC_ADC->ADINTEN=(1<<2); // Hab. interrupción fin de conversión del canal 0
        NVIC_EnableIRQ(ADC_IRQn);
    }
    if(indice_muestra==(N_muestras-1))
    {
        indice_muestra=0;
        LPC_TIM1->TCR=0x02; //Stop Timer and reset, DAC= 0V.
        LPC_DAC->DACR=0; // 0 V /33
        init_ADC_LDRs();
        audio= FIN;
    }

    else if( indice_muestra==(N_muestras/2)-1 && audio ==ALARMA)
    {
        indice_muestra=0;
        LPC_TIM1->TCR=0x02; //Stop Timer and reset, DAC= 0V.
        LPC_DAC->DACR=0; // 0 V /33
        init_ADC_LDRs();
        audio =FIN;
    }
}

// TIMER 2 PARA LDRs Y MOVIMIENTO MODO AUTOMATICO EN FUNCION DE LOS LDRs
void init_TIMER2(void)
{

```

## SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS 93

### DISEÑO DE UN ROBOT CON TRACCIÓN DIFERENCIAL

```

LPC_SC->PCONF|=(1<<22);           // Alimento el Timer2 , está desactivado de forma predeterminada
LPC_PINCON->PINSEL9|=(1<<25);     // MAT2.0 para PWM servo (P4.28)
LPC_TIM2->MCR = 0x18;              // Interrupt on Match 1 and Reset TC (bit 4 y 5 activos)
LPC_TIM2->MR1 = (Fpclk/F_muestreo)-1; // Periodo de muestreo de TODAS las entradas!!!!
LPC_TIM2->EMR = 0x00C2;            // MODO TOGGLE en EMC1
LPC_TIM2->TCR = 0x01;              // Arranca timer
NVIC_EnableIRQ(TIMER2_IRQn);
NVIC_SetPriority(TIMER2_IRQn,4);   // Prioridad por encima del priority grouping
}                                  // No hace falta asignar un pin porque no queremos ningun match, solo
interrumpe

void TIMER2_IRQHandler(void)
{
    LPC_TIM2->IR|=(1<<1);          // Borrar flag interrupción
    LPC_ADC->ADCR|=(1<<16);        // BURST=1 --> Cada 65TclkADC se toma una muestra de cada canal comenzando
                                   // desde el más bajo (bit LSB de CR[0..7])
    Dato_sensorLuz1();
    Dato_sensorLuz2();
    Dato_promedioSensores();

    //Movimiento automatico modo automatico
    if (Estado==5 && distancia>umbralDistancia && Giro90== false && Giromenos90== false)
    {
        if (SPromedio>valorluz && barridoluz==false) //Si el coche
        detecta LUZ
        {
            Posserv=0;
            setServo(Posserv);
            grados=0;
            Barrido_Completo=false;
            Barrido= false;
            if (SLuz1>SLuz2) //Si la
            el el sensor IZq recibe mas luz nos movemos a la izquierda
            {
                VelIzq= 60;
                VelDer = 75;
            }
            else if (SLuz1<SLuz2) //Si la el el sensor Der
            recibe mas luz nos movemos a la Derecha
            {
                VelIzq = 75;
                VelDer = 60;
            }
            else //Si reciben la misma Luz nos movemos a recto
            {
                VelDer =75;
                VelIzq=75;
            }
        }
        else //Si no detecta luz
        {
            if (Barrido_Completo==false&& Barrido== false && grados ==0)
            {
                Posserv=-90;
                setServo(Posserv);
                Barrido = true;
            }
            else if (Barrido_Completo== true)
            {
                VelDer=70;
                VelIzq=70;
            }
            else if (grados<0)
            {
                VelIzq= 60;
                VelDer = 75;
                barridoluz=false;
            }
            else if (grados>0)
            {
                VelIzq= 75;
                VelDer = 60;
                barridoluz=false;
            }
        }
    }
    else if (distancia<umbralDistancia&& Estado ==5)
    {
        Barrido =false;
        Barrido_Completo= false;
        if (SLuz1>SLuz2) //Si la el el
        sensor IZq recibe mas luz nos movemos a la izquierda
        {
            Giro90= true; //Giro
            Izquierda 90°
        }
        else if (SLuz2>SLuz1) //Si la
        el el sensor IZq recibe mas luz nos movemos a la izquierda
        {
            Giromenos90= true; //Giro Derecha 90°
        }
        else
        {
            Giro90= true;
        }
    }
}

// TIMER 3 PARA ULTRASONIDO

```

```

void init_TIMER3(void)
{
    LPC_SC->PCONP|= 1<<23;           // Alimento el Timer 3, que esta desactivado de forma predeterminada.
    LPC_PINCON->PINSEL0|= (3<<22);    // MAT3.1 para el Trigger (P0.11)
    LPC_PINCON->PINSEL1|= (3<<14);    // CAP3.0 para el echo (P0.23)

    LPC_TIM3->PR = 24;
    LPC_TIM3->CCR = 0x6;              // Interrupcion en flanco de bajada

    LPC_TIM3->MCR |= (0x3<<0);        // GENERAR INTERRUPCIÓN Y RESETEAR CUANDO LLEGUE MR0

    LPC_TIM3->MR0 = T_trigger;
    LPC_TIM3->MR1 = Th_trigger;
    LPC_TIM3->EMR = 1<<1 | 1<<6;     // mas rápido que EMR |= 1<<1;

    NVIC_SetPriority(TIM3_IRQn,3);
    NVIC_EnableIRQ(TIM3_IRQn);

    LPC_TIM3->TCR = 1<<1;             // RESET TIM3
    LPC_TIM3->TCR = 1<<0;             // START TIM3.
}

void TIMER3_IRQHandler(void) {
    static uint32_t temp;
    if(LPC_TIM3->IR & (1 << 0)){
        LPC_TIM3->IR |= 1<<0;
        LPC_TIM3->EMR = 1<<6 | 1<<1; // mas rápido que EMR |= 1<<1;
    }

    if (LPC_TIM3->IR & (1 << 4)){ // si hay evento de CR0
        LPC_TIM3->IR |= 1<<4;

        if(LPC_TIM3->CCR == 5){
            temp = LPC_TIM3->CR0;
            LPC_TIM3->CCR = 0x6;
        }
        else{
            N = LPC_TIM3->CR0 - temp;           // Tiempo en alto
            LPC_TIM3->CCR = 0x5;

        }
        if(N>58){
            distancia = N/58;
            Dato_Distanciacm();
            if (distancia < umbralDistancia && Barrido==false){
                if(Estado==3){
                    VelDer= 0;
                    VelIzq = 0;
                }
                if(Estado==3 || Estado==5)
                    Audio(ALARMA);
            }
        }
    }
}

//INICIA TODOS LOS TIMERS MENOS EL TIMER1 QUE UTILIZAMOS PARA EL AUDIO Y SOLO LO INICIAREMOS CUANDO LO NECESITEMOS
void init_TIMERS(void)
{
    init_TIMER0();
    init_TIMER2();
    init_TIMER3();
}

//FUNCION QUE CONTROLA QUE LOS PULSADORES SEAN PULSADOS
void controladorPulsadores(void) {
    if ((LPC_GPIO2->FIOPIN & 1<<11) == 0x0000) { //Key1
        flagContadorKey1 = true;
    }
    else {
        flagContadorKey1 = false;
    }
    if ((LPC_GPIO2->FIOPIN & 1<<12) == 0x0000) { //Key2
        flagContadorKey2 = true;
    }
    else {
        flagContadorKey2 = false;
    }
}

//FUNCION QUE REINICIA CONTADORES
void contadoresreset(void)
{
    contadorKey1=0;
    contadorKey2=0;
    contadorC = 0;
    contadorZ = 0;
    flagCambioMas2segPulsadores=false;
    flagCambioMenos2segPulsadores=false;
    flagCambioMas2segBotones=false;
    flagCambioMenos2segBotones=false;
    flagCambioPulsandoC = false;
}

```

*m. Index.cgi*

```

t  <!DOCTYPE html>
t  <html>
t  <head>
t    <meta content="text/html; charset=UTF-8" http-equiv="content-type">
t    <meta http-equiv="refresh" content="7; url=http://192.168.0.125/index.cgi">
t    <title>Control Web</title>
t  </head>
t  <table style="height:174px; width=700px; border-collapse: collapse; border="1">
t    <tbody>
t      <tr style="height:18px; width=700px;">
t        <td style="width: 100%; height: 18px; background-color: #2e6c80; text-align: center; colspan="3">
t          <span style="color: #000000;"><strong> Control coche<br/></strong></span></td>
t      </tr>
t      <tr style="height: 18px; width=700px;">
c 1 a      <td style="width: 100%; height: 18px; text-align: center;"><strong> %s <br/></strong></td>
t      <div style="font-weight:bold">
t      </div>
t      <tr style="height: 32px; width=700px;">
c 1 b      <td style="width: 400px; height: 100%; text-align: center;">
t        <p> Vel. Derecha: %d <br/></p></td>
t        <td style="width: 300px; height: 100%; text-align: center;">
t          %s
t        </td></tr>
t      <tr style="height: 32px; width=700px;">
c 1 c      <td style="width: 400px; height: 100%; text-align: center;">
t        <p> Vel. Izquierda: %d <br/></p></td>
c 1 n      <td style="width: 300px; height: 100%; text-align: center;">
t        %s
t      </td></tr>
t      <tr style="height: 32px; width=700px;">
c 1 d      <td style="width: 400px; height: 100%; text-align: center;">
t        <p> Posicion Servo: %d <br/></p></td>
c 1 o      <td style="width: 300px; height: 100%; text-align: center;">
t        %s
t      </td></tr>
t      <tr style="height: 32px; width=900px;">
c 1 f      <td style="width: 100%; height: 100%; text-align: center;">
t        <p> Sensor Luz 1: %d <br/></p>
t      </td></tr>
t      <td style="width: 100%; height: 100%; text-align: center;">
c 1 g      <p> Sensor Luz 2: %d <br/></p>
t      </td></tr>
t      <td style="width: 100%; height: 100%; text-align: center;">
c 1 h      <p> Promedio: %d <br/></p>
t      </td></tr>
t      <td style="width: 100%; height: 100%; text-align: center;">
c 1 i      <p> Distancia: %d <br/></p>
t      </td></tr>
t    </tbody>
t  </table>
t  <p><strong>Seleccion de modo</strong></p>
t  <form id="formulario">
t    <p>
c 1 j    <input %s name="manual" type="radio" value="manual" onClick="submiy();" /> Modo Manual
c 1 k    <input %s name="automatico" type="radio" value="automatico" onClick="submiy();" /> Modo Automatico
c 1 l    <input %s name="depuracion" type="radio" value="depuracion" onClick="submiy();" /> Modo Depuracion
t    </p></form>
t  </html>
.
```

## II. Hojas de datos de los chip utilizados.

Para la realización de la practica hemos usado una serie de documentación tanto del funcionamiento de la herramienta KEIL, como de los componentes y el microcontrolador que utilizamos, dicha documentación estará adjunta en nuestro proyecto.

