ANEXOS

A PERSONAS DE LA TERCERA EDAD AL 2007, POR EDADES, EN PERU

Los porcentajes mostrados son respecto al total de población del Perú al 2007.

EDAD	CANTIDAD	%
65 años	155,075	0.57
66 años	107,085	0.39
67 años	124,991	0.46
68 años	108,316	0.40
69 años	83,835	0.31
70 años	130,702	0.48
71 años	67,429	0.25
72 años	95,874	0.35
73 años	80,297	0.29
74 años	78,696	0.29
75 años	95,144	0.35
76 años	65,196	0.24
77 años	67,547	0.25
78 años	69,590	0.25
79 años	46,522	0.17
80 años	70,032	0.26
81 años	31,224	0.11
82 años	39,661	0.14
83 años	32,010	0.12
84 años	30,709	0.11
85 años	36,223	0.13
86 años	24,842	0.09
87 años	25,044	0.09
88 años	16,989	0.06
89 años	14,400	0.05
90 años	16,658	0.06
91 años	6,288	0.02
92 años	7,503	0.03
93 años	5,988	0.02
94 años	4,894	0.02
95 años	6,076	0.02
96 años	3,807	0.01
97 años	3,202	0.01
98 años	12,838	0.05
Totales	**1’764,687**	**6**

Fuente: INEI Sistema de consulta de datos vía web

B CODIGO FUENTE – RUTINAS DE BAJO NIVEL DE ACCESO AL BUS I^2C

Este programa, escrito en lenguaje C usando el compilador WINAVR, contiene rutinas elementales que interactúan directamente con el bus I^2C para diversos propósitos: configuración, lectura y escritura.

```
#include        "twi.h"
/*////////////////////////////////////////////////////////////////////////////////////////////
RUTINAS BASICAS PARA EL PUERTO I2C - MODO MASTER
        - Configuración de velocidad
        - Comando START
        - Comando START wait
        - Comando REP-START
        - Comando STOP
        - Comando WRITE
        - Comando READack
        - Comando READnak
void                    twimaster_init(void);
unsigned char           twimaster_start(unsigned char address);
void                    twimaster_start_wait(unsigned char address);
unsigned char           twimaster_rep_start(unsigned char address);
void                    twimaster_stop(void);
unsigned char           twimaster_write( unsigned char data);
unsigned char           twimaster_readAck(void);
unsigned char           twimaster_readNak(void);
*////////////////////////////////////////////////////////////////////////////////////////////
//INICIALIZACION DE PUERTO SERIE I2C
//Frecuencia de 100 KHz
void twimaster_init(void)
        {
                TWBR  = 0x32;         //TWI Bit Rate Register = 32
                TWSR  = 0x00;         //TWPS en TWSR se fija en 00        (PS = 1)
        }
/*************************************************************************************************
  Envía al bus condición de START, junto con dirección de esclavo y sentido de transferencia (R/W)
  Entrada:      address (dirección del esclavo + bit R/"W)
  return:       0 = Dispositivo accesible, 1= Falla en acceso al dispositivo
```

```
*************************************************************************************************/
unsigned char twimaster_start(unsigned char address)
{
   uint8_t   twst;
   // Envía condición de START
   TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

   // Espera hasta que se complete la transmisión
   while (!(TWCR & (1<<TWINT)));

   // Verifica valor del registro de estado TWI (enmascara bits PS - pre-escalador)
   twst = TW_STATUS & 0xF8;
   if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

   // Envía dirección del dispositivo
   TWDR = address;
   TWCR = (1<<TWINT) | (1<<TWEN);

   // Espera hasta que la transmisión se complete y se reciba un ACK/NACK del esclavo
   while(!(TWCR & (1<<TWINT)));

   // Verifica valor del registro de estado TWSR (enmascara bits PS - pre-escalador)
   twst = TW_STATUS & 0xF8;
   if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;
   return 0;

}/* twimaster_start */

/*****************************************************************************
 Envía condición de START, dirección del esclavo y sentido de transferencia (R/W).
 Si el dispositivo esta ocupado, usa ack polling para esperar hasta que el dispositivo se libre
  Entrada:   Dirección y sentido de transferencia del dispositivo I2C
*****************************************************************************/
void twimaster_start_wait(unsigned char address)
{
   uint8_t   twst;
   while ( 1 )
```

```
    {
        // Envía condición de START
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // Espera hasta que la transmisión se complete
        while(!(TWCR & (1<<TWINT)));

        // Verifica valor de registro de estado de TWI. Enmascara bits de PS
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

        // Envía dirección del dispositivo
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // Espera hasta que la transmisión se complete
        while(!(TWCR & (1<<TWINT)));

        // Verifica valor de registro de estado de TWI. Enmascara bits de PS
        twst = TW_STATUS & 0xF8;
        if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
        {
 /* Dispositivo ocupado, envía condición de STOP para terminar escritura*/
            TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

            // Espera hasta que la condición de STOP sea ejecutad y el bus quede libre
            while(TWCR & (1<<TWSTO));
            continue;
        }
        //if( twst != TW_MT_SLA_ACK) return 1;
        break;
     }
}/* twimaster_start_wait */
```

```
/*****************************************************************************
 Envía un START repetido, también dirección y sentido de transferencia
Entrada:   Dirección y sentido de transferencia del dispositivo I2C
Retorna:  0 Dispositivo accesible
          1 Falla en acceder al dispositivo
*****************************************************************************/
unsigned char twimaster_rep_start(unsigned char address)
{
   return twimaster_start( address );
}/* twimaster_rep_start */
/*****************************************************************************
 Termina la transferencia de datos y libera el bus I2C
*****************************************************************************/
void twimaster_stop(void)
{
   /* Envía condición de STOP */
   TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

   // Espera se complete la ejecución del STOP y libera el bus
   while(TWCR & (1<<TWSTO));
}/* twimaster_stop */

/*****************************************************************************
  Envía un byte por el bus I2C
  Entrada:    Byte a ser transferido
  Retorno:       0 Escritura OK
                 1 Escritura FAIL
*****************************************************************************/
unsigned char twimaster_write( unsigned char data )
{
   uint8_t   twst;
   // Envía dato al dispositivo previamente direccionado
   TWDR = data;
   TWCR = (1<<TWINT) | (1<<TWEN);
   //Espera se complete la transmisión
   while(!(TWCR & (1<<TWINT)));
```

```
    // Verifica valor de registro de estado de TWI. Enmascara bits de PS
    twst = TW_STATUS & 0xF8;
    if( twst != TW_MT_DATA_ACK) return 1;
    return 0;
}/* twimaster_write */

/*************************************************************************
 Lee un byte de dispositivo I2C, solicita más datos del dispositivo
Retorna:  Byte leido del dispositivo I2C
*************************************************************************/
unsigned char twimaster_readAck(void)
{
    TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCR & (1<<TWINT)));
    return TWDR;
}/* twimaster_readAck */

/*************************************************************************
 Lee un byte de dispositivo I2C, la lectura es seguida por una condición de STOP
Retorna: Byte leído del dispositivo I2C
*************************************************************************/
unsigned char twimaster_readNak(void)
{
    TWCR = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR & (1<<TWINT)));
    return TWDR;
}/* twimaster_readNak */
```

C CODIGO FUENTE – RUTINAS DE BAJO NIVEL DE ACCESO AL BUS I²C EN MODO MAESTRO

Este programa, escrito en lenguaje C usando el compilador WINAVR, contiene rutinas elementales que interactúan directamente con el bus I²C para diversos propósitos: configuración, lectura y escritura.

```
#include        "twimaster.h"
/************************************************************************
   Inicializa la interface I2C.
************************************************************************/
void i2c_init()
{
   twimaster_init();
}
/************************************************************************
   Realiza lectura de un byte de la dirección I2C especificada
      dev_addr = Dirección del dispositivo I2C
      addr = Dirección de lectura del registro
************************************************************************/
unsigned char i2c_read(unsigned char dev_addr, unsigned char addr)
{
   unsigned char status;
   // Envía START y fija modo de escritura
   if ((status = twimaster_start(dev_addr + I2C_WRITE)) != 0)
   {
      /* Falla en envío de condición de START, Posible ausencia de dispositivo*/
      twimaster_stop();
   }
   else
   {
      twimaster_write(addr);                    // Escribe dirección del registro
      twimaster_rep_start(dev_addr+I2C_READ);   // Fija dirección del dispositivo y
modo lectura
      status = twimaster_readNak();             // Lee un byte
      twimaster_stop();
   }
   return status;
```

```
}
/*************************************************************************
   Realiza lectura de mútiples bytes del bus I2C
      dev_addr    = Dirección del dispositivo I2C
      addr        = Dirección de inicio del registro leido
      data        = Arreglo del arreglo de bytes que contendrán los datos a leer
      len         = Longitud o cantidad de datos a leer
   Returna 0 si todo OK, 1 para falla
*************************************************************************/
unsigned char i2c_multi_read(unsigned char dev_addr, unsigned char addr, unsigned char
*data, unsigned char len)
{
   unsigned char status, i;
   // Envía START y fija modo escritura
   if ((status = twimaster_start(dev_addr + I2C_WRITE)) != 0)
   {
      /* Falla en envío de START, posible dispositivo ausente */
      twimaster_stop();
   }
   else
   {
      twimaster_write(addr);                          // Escribe dirección del registro
      twimaster_rep_start(dev_addr+I2C_READ);   // Fija dirección del dispositivo y modo de
lectura
      // Reconoce con ACK todas las lecturas excepto la última que será con NAK.
      for (i=0; i<len-1; i++)
      {
         *data++ = twimaster_readAck();
      }
      *data++ = twimaster_readNak();
      twimaster_stop();
   }
   return status;
}
```

```
/*****************************************************************************
   Realiza escritura de un byte del bus I2C.
      dev_addr    = Dirección del dispositivo I2C
      addr        = Dirección del registro de escritura
      data        = Dato a ser escrito
*****************************************************************************/
unsigned char i2c_write(unsigned char dev_addr, unsigned char addr, unsigned char data)
{
   unsigned char ret = 0;
   // Envía START y modo de escritura
   if ((ret = twimaster_start(dev_addr+I2C_WRITE)) != 0)
   {
      /* Falla en enviar condición de START */
      twimaster_stop();
   }
   else
   {
      twimaster_write(addr);
      twimaster_write(data);
      twimaster_stop();
   }
   return ret;
}
```

D CODIGO FUENTE – LIBRERIA DE ACCESO AL SENSOR ADXL345

Este programa, escrito en lenguaje C usando el compilador WINAVR, contiene rutinas elementales que interactúan directamente con el bus I^2C para diversos

```
#include "adxl345.h"
#include "i2c.h"
/*************************************************************************
   Inicializa el sensor i-MEMS ADXL345
*************************************************************************/
void adxl_init()
{   i2c_init();
   i2c_write((ADXL345_ADDR<<1), ADXL345_POWER_CTL, 0x08 );
   // Habilita medición     POWER_CTL = 0000 1000 = 0x08 - (1<<ADXL345_MEASURE)
}
/*************************************************************************
   Realiza lectura de un byte del sensor ADXL345
*************************************************************************/
unsigned char adxl_read(unsigned char addr)
{  unsigned char dev_addr = ADXL345_ADDR << 1;
   return i2c_read(dev_addr, addr);
}
/*************************************************************************
   Realiza lectura de múltiples bytes del sensor ADXL345
*************************************************************************/
unsigned char adxl_multi_read(unsigned char addr, unsigned char *data, unsigned char len)
{  unsigned char dev_addr = ADXL345_ADDR << 1;
   return i2c_multi_read(dev_addr, ADXL345_DATAX0, data, len);
}
/*************************************************************************
   Realiza escritura de un byte en el sensor ADXL345
*************************************************************************/
unsigned char adxl_write(unsigned char addr, unsigned char data)
{  unsigned char dev_addr = ADXL345_ADDR << 1;
   return i2c_write(dev_addr, addr, data);
}
```

E CODIGO FUENTE – PROGRAMA DE ADQUISICION DE DATOS

Este programa, que corre en el microcontrolador ATmega88L, tiene como propósito principal leer datos del sensor, almacenarlos en memoria y enviarlos vía puerto RS-232 hacia la computadora personal para efectos de visualización de formas de onda de aceleración. Hace uso de todas las rutinas y librerías de los anexos B, C y D.

```
/*****************************************************************************************************************************************
 PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU -  MAESTRIA EN INGENIERIA BIOMEDICA
 TESIS
 DISEÑO DE UN SISTEMA INALAMBRICO DE DETECCION DE CAIDAS APLICADO A PERSONAS DE LA TERCERA EDAD BASADO EN ACELEROMETRO Y TELEFONO MOVIL
 AUTOR: Ing. Edgard Oporto Diaz -  LIMA - PERU - 2010
*****************************************************************************************************************************************/
//ARCHIVOS INCLUIDOS
#include                <avr/io.h>              //Llama al avr/iom8.h
#include                <avr_232.h>             //Definición de interrupcones para usar con AVR
#include                <interrupt.h>           //Definición de interrupcones para usar con AVR
#include                <adxl345.h>             //Rutinas personalizadas para interactuar con el ADXL345

//CONSTANTES
const unsigned char     ACELERA = 0xE5;         //ID del sensor
#define                 ACELERA_02 0x07         //ID del autor, para pruebas

//VARIABLES GLOBALES
unsigned char           *data = 0x100;          //Puntero a buffer para recibir datos leídos del sensor
unsigned char           data1;
unsigned char           data2 = 0;

//DECLARACION DE FUNCIONES
//void                  adxl_init();
//unsigned char         adxl_read(unsigned char addr);
//unsigned char         adxl_multi_read(unsigned char addr, unsigned char *data, unsigned char len);
//unsigned char         adxl_write(unsigned char addr, unsigned char data);
//void                  USART_init(void);
//void                  USART_Tx(unsigned char data);
  void                  Error(void);

//DECLARACION DE RUTINAS DE SERVICIO DE INTERRUPCION
ISR(INT0_vect);
ISR(INT1_vect);
ISR(USART_RXC_vect);

/*****************************************************************************************************************************************
//FUNCION PRINCIPAL
*****************************************************************************************************************************************/
void                    main(void)
        {
                        unsigned char data;
                        DDRB=0x00;          //Configura sentido PORTB
```

```
                        PORTB=0xFF;     //Todos como ENTRADAS (sin Hi-Z)
                        DDRC=0x00;      //Configura sentido PORTC
                        PORTC=0xFF;     //Todos como entradas (sin Hi-Z)

                        DDRD=0xFF;      //Configura sentido PORTD
                        PORTD=0x00;     //Todos como salidas, en valor HIGH
                        USART_init();   //Inicializa el puerto serial RS-232
                        adxl_init();    //Inicializa el puerto I2C. Velocidad de 100KHz, MEASURE habilitada

                        adxl_write(0x31,0x0B);      //Configura el sensor: 0000 1011, 12+1 bits mode, +- 16g range
                        adxl_write(0x38,0x0F);      //Deshabilita FIFO 0000 1111
                        adxl_write(0x2C,0x0A);      //RATE CODE = 100 Hz

//                      asm ("SEI");                //Habilitación general de interrupciones
//                      PORTD=PINB;
//                      data=PINB;
                        while   (1)
                        {
                        data=adxl_read(0x00);       //Lee dato del sensor (ACELERACION X, byte menos significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial
                        data=adxl_read(0x32);       //Lee dato del sensor (ACELERACION X, byte menos significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial

                        data=adxl_read(0x33);       //Lee dato del sensor (ACELERACION X, byte más significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial
                        data=adxl_read(0x34);       //Lee dato del sensor (ACELERACION Y, byte menos significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial

                        data=adxl_read(0x35);       //Lee dato del sensor (ACELERACION Y, byte más significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial
                        data=adxl_read(0x36);       //Lee dato del sensor (ACELERACION Z, byte menos significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial
                        data=adxl_read(0x37);       //Lee dato del sensor (ACELERACION Z, byte más significativo)
                        USART_Tx(data);             //Lo envía por el puerto serial
                        }
        }
/*****************************************************************************
RUTINAS DE SERVICIO DE INTERRUPCION
*****************************************************************************/
ISR(INT0_vect)
{
   //Lectura de 6 bytes del sensor
   adxl_multi_read(ADXL345_DATAX0, data, 6);
}
/*****************************************************************************
RUTINAS DE MANEJO DE ERRORES
*****************************************************************************/
//I2C - Error: no se logra enviar START
void Error(void)
        {
        }
```

F CODIGO FUENTE – PROGRAMA DE PRUEBAS SIN PACIENTE USANDO MATLAB

Este programa se emplea para realizar pruebas de caídas sin paciente bajo diversos escenarios. En vez de adultos mayores, las pruebas se realizan con una persona adulta (cuando sea posible) y en otros casos mediante un fantoma (en las caídas que implican más riesgo). Los datos sensados son visualizados mediante una computadora personal.

```
%RECEPCION DE DATOS POR EL PUERTO SERIAL EN FORMATO BINARIO
clc;
clf;
%Crear un objeto puerto serie obj_Serial_Port asociado con un puerto serial
p_serial = input('Ingrese el número de puerto serial COM: ','s')
%String
Numero_muestras = input('Ingrese el número de muestras a leer: ')
%Numérico
t1=clock;
obj_Serial_Port = serial(['COM', p_serial]);
set(obj_Serial_Port,'timeout', 30);   %30 segundos, tiempo de espera por
datos

%Configura propiedades – buffer de entrada para recibir bytes
%Configure the baud rate to the highest value
%Desactiva la opción de solicitud para enviar
obj_Serial_Port.InputBufferSize = 50000;
obj_Serial_Port.BaudRate = 9600;
set(obj_Serial_Port,'requesttosend','off');

%Conexión al dispositivo
fopen(obj_Serial_Port);

%Espera hasta que todos los datos sean enviados al buffer de entrada, luego
%los transfiere los datos a Matlab como enteros de 8 bits sin signo.
%out = fread(obj_Serial_Port,obj_Serial_Port.BytesAvailable,'uint8');

out = fread(obj_Serial_Port,Numero_muestras,'uint8');

fclose(obj_Serial_Port);
delete(obj_Serial_Port);
clear obj_Serial_Port;

%Construcción de las matrices de aceleración por ejes
%Acc_x
%Acc_x1
%Acc_x0
%Acc_y
%Acc_y1
%Acc_y0
%Acc_z
%Acc_z1
%Acc_z0
%Acc_resultante
%Matriz de ID (solo para verificación)
%ID_Matrix
```

```
%Detecta el primer ID para sincronizar los datos
Cantidad_datos = Numero_muestras;
i=1;
Pos_First_ID=1;

while out(i,1)~= 229
    i=i+1;
    Pos_First_ID=i;
end

Cantidad_datos = Cantidad_datos - Pos_First_ID - 1;
Cantidad_datos = Cantidad_datos - rem(Cantidad_datos,7);

Total_Datos_eje = Cantidad_datos / 7;

Acc_x  =zeros(Total_Datos_eje,1);
Acc_x0 =zeros(Total_Datos_eje,1,'int8');
Acc_x1 =zeros(Total_Datos_eje,1,'int8');
Acc_y  =zeros(Total_Datos_eje,1);
Acc_y0 =zeros(Total_Datos_eje,1,'int8');
Acc_y1 =zeros(Total_Datos_eje,1,'int8');
Acc_z  =zeros(Total_Datos_eje,1);
Acc_z0 =zeros(Total_Datos_eje,1,'int8');
Acc_z1 =zeros(Total_Datos_eje,1,'int8');
Acc_resultante =zeros(Total_Datos_eje,1);
%Matriz de ID (solo para verificación)
ID_Matrix     =zeros(Total_Datos_eje,1);

NEGATIVOS_x=0; NEGATIVOS_y=0; NEGATIVOS_z=0;

for i=1:Total_Datos_eje

    ID_Matrix(i) =  out(Pos_First_ID + 7*(i-1) + 0,1);

    Acc_x0    =     out(Pos_First_ID + 7*(i-1) + 1,1);
    Acc_x1    =     out(Pos_First_ID + 7*(i-1) + 2,1);
    if bitget(Acc_x1,8)==1
        Acc_x0    = bitcmp(Acc_x0,8) + 1;  %Complemento a 2 si es negativo)
        Acc_x1    = bitcmp(Acc_x1,8)    ;  %Complemento a 2 si es negativo)
        Acc_x(i)  = -(256*Acc_x1 + Acc_x0)*3.9*0.001;
NEGATIVOS_x=NEGATIVOS_x+1;
    else
    Acc_x(i)  =     (256*Acc_x1 + Acc_x0)*3.9*0.001;
    end

    Acc_y0   =     out(Pos_First_ID + 7*(i-1) + 3,1);
    Acc_y1   =     out(Pos_First_ID + 7*(i-1) + 4,1);
    if bitget(Acc_y1,8)==1
        Acc_y0    = bitcmp(Acc_y0,8) + 1;  %Complemento a 2 si es negativo)
        Acc_y1    = bitcmp(Acc_y1,8)    ;  %Complemento a 2 si es negativo)
        Acc_y(i)  = -(256*Acc_y1 + Acc_y0)*3.9*0.001;
NEGATIVOS_y=NEGATIVOS_y+1;
    else
    Acc_y(i)  =     (256*Acc_y1 + Acc_y0)*3.9*0.001;
    end

    Acc_z0    =     out(Pos_First_ID + 7*(i-1) + 5,1);
    Acc_z1    =     out(Pos_First_ID + 7*(i-1) + 6,1);
    if bitget(Acc_z1,8)==1
```

```
        Acc_z0      = bitcmp(Acc_z0,8) + 1;%Complemento a 2 si es negativo)
        Acc_z1      = bitcmp(Acc_z1,8)    ;%Complemento a 2 si es negativo)
        Acc_z(i)    = -(256*Acc_z1 + Acc_z0)*3.9*0.001;
NEGATIVOS_z=NEGATIVOS_z+1;
    else
    Acc_z(i)  =     (256*Acc_z1 + Acc_z0)*3.9*0.001;
    end

    Acc_resultante(i) = Acc_x(i)*Acc_x(i) + Acc_y(i)+Acc_x(i) +
Acc_z(i)*Acc_z(i);
    Acc_resultante(i) = sqrt(Acc_resultante(i));
end

elapsed_time = etime(clock,t1)

t=0:0.01:0.01*(Total_Datos_eje-1);

figure (1)
plot(t,Acc_x,'b')
ylim([-16 16])
title('ACELERACION EJE X')
xlabel('Tiempo (s)')
ylabel('Aceleración en g (m/s)')

figure (2)
plot(t,Acc_y,'r')
ylim([-16 16])
title('ACELERACION EJE Y')
xlabel('Tiempo (s)')
ylabel('Aceleración en g (m/s)')

figure (3)
plot(t,Acc_z,'g')
ylim([-16 16])
title('ACELERACION EJE Z')
xlabel('Tiempo (s)')
ylabel('Aceleración en g (m/s)')

figure (4)
plot(t,Acc_x,'b');            hold on; %legend('Aceleración X'); hold
plot(t,Acc_y,'r');            hold on; %legend('Aceleración Y'); hold
plot(t,Acc_z,'g');            hold on; %legend('Aceleración Z'); hold
plot(t,Acc_resultante,'k');   hold on;

legend('Acc_X', 'Acc_Y', 'Acc_Z', '|AccXYZ|'); hold on
ylim([-16 16])
title('ACELERACION - EJES X Y Z')
xlabel('Tiempo (s)')
ylabel('g (m/s)')
```

G CODIGO FUENTE – PROGRAMA DE CONFIGURACION Y PRUEBAS DE ENVIO DE SMS CON EL MODULO BLUETOOTH EZURIO BTM402

Este programa es, en realidad, una secuencia de comandos AT enviados al módulo a través de su puerto serial para propósitos de configuración y verificación de su funcionamiento. Los comandos enviados son los siguientes:

AT	→	*Conecta con el modulo*
*AT&F**	→	*Pone todos los registros internos por defecto*
ATS512=4	→	*Activa visibilidad y conexión*
ATS555=4	→	*Activa visibilidad y conexión después del PowerUp*
ATS554=0	→	*Para que active el S555 después del PowerUp*
ATS0=1	→	*Configura el Auto-Answer para que sea automático*
ATS536=1	→	*Para que acepte los comandos AT remotamente.*
ATS610=$7FFF	→	*Configura los puertos GPIO como salidas.*
AT+BTN=”string”	→	*Nombre del módulo. p.e: “RADIO”*
AT+BTK=”pin”	→	*Pin contraseña del dispositivo p.e: “1234”*
AT&W	→	*Guarda a memoria no volátil los registros modificados*
AT+BTN?	→	*Comprueba configuración, pregunta nombre dispositivo*

Cabe mencionar que todo comando debe finalizar con un <CR> (RETORNO DE CARRO) para que sea reconocido por el módulo de radio.

H LAYOUTs DE CIRCUITOS IMPRESOS

VERSION DE PRUEBA 01

Las características de esta primera tarjeta, mostrada en la Figura H.1, son las siguientes:

a) Material: fibra de vidrio
b) Doble cara
c) Máscara antisoldante
d) Distribución de componentes
e) Dimensiones: 4 cm x 4 cm
f) Se empleó uC Atmel ATmega8L tipo DIP y sensor de aceleración ADXL345 SMD 14-LGA.
g) No usa batería, emplea alimentación conectado a la red comercial.

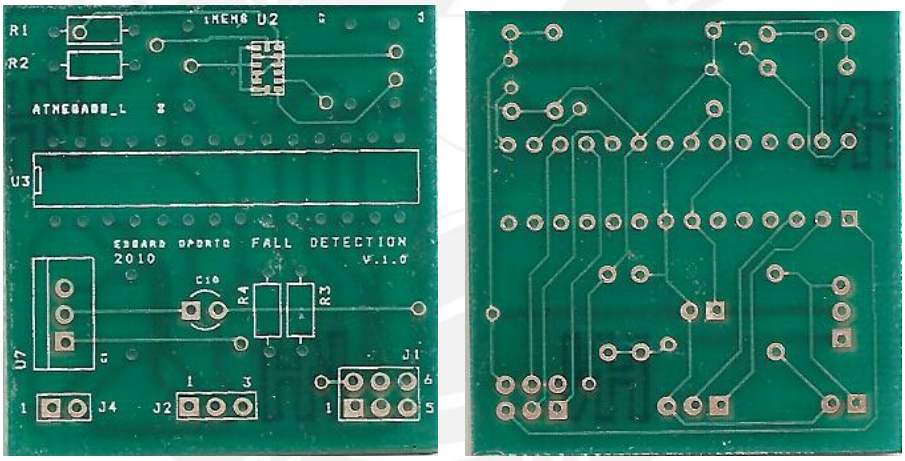

Figura H.1– Circuitos impresos del circuito de aplicación, versión 01
(Fuente: Elaboración propia)

VERSION DE PRUEBA 02

Las características de esta segunda tarjeta, mostrada en la Figura H.2, son las siguientes:

a) Similar a la primera, pero con agujeros metalizados
b) Dimensiones: 6.5 cm x 7 cm
c) Se empleó uC Atmel ATmega8L tipo DIP y sensor de aceleración ADXL345 SMD 14-LGA y módulo de radio Bluetooth Ezurio BTM402.

d) Emplea conector HIROSE de 40 pines y montaje superficial para el módulo de radio.

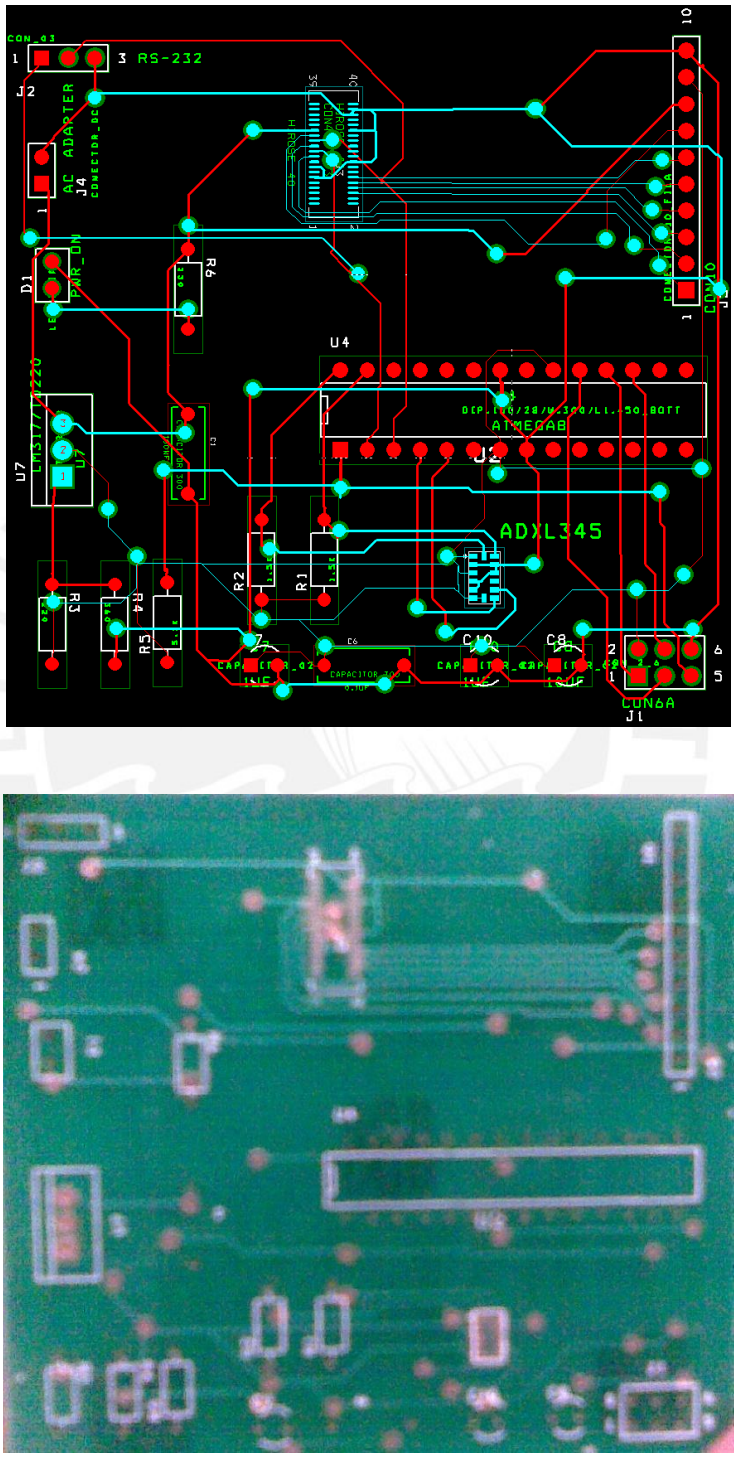

Figura H.2 – Diseño del circuito impreso de la segunda versión de la tarjeta. En la parte inferior se muestra la tarjeta aun por montar los dispositivos

(Fuente: Elaboración propia)

VERSION DEFINITIVA

Esta es la versión definitiva del circuito del proyecto. Todos los dispositivos que incorpora son de montaje superficial y se alimenta con batería de iones de Litio. Las características de esta versión definitiva de la, mostrada en la Figura H.3, son las siguientes:

a) Material: fibra de vidrio
b) Doble cara
c) Máscara antisoldate
d) Distribución de componentes
e) Agujeros metalizados
f) Dimensiones: 6 cm x 4 cm
g) Dispositivos y zócalos de montaje superficial
h) Empleo de batería de iones de Litio

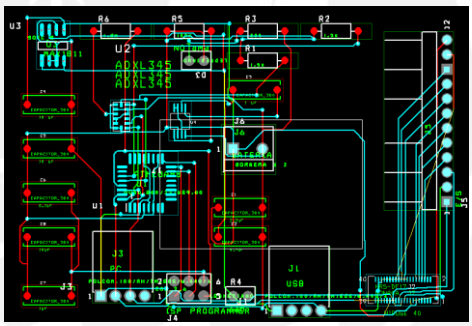

Figura H.3 – Diseño del circuito impreso de la versión definitiva de la tarjeta (Fuente: Elaboración propia)