ANEXOS

Módulo de posicionamiento y búsqueda basado en GPS y redes ad-hoc

Anexo: Unidad de procesamiento y puertos de expansión del módulo KYNEO V0.2

A continuación, se describe microcontrolador y los puertos de expansión integrados en el módulo KYNEO V0.2.

- Unidad de Procesamiento

El módulo KINEO V0.2 incluye un microcontrolador ATmega1284P con un rendimiento de hasta 1 MIPS/MHz que soporta velocidades de reloj de hasta 20 MHz (en este caso trabajará a 16 MHz). Se trata de un microcontrolador de Atmel AVR de 8-bit que dispone de 128 kB de memoria flash, 16 kB de memoria SRAM y 4 kB de memoria EEPROM, y hasta 32 pines de entrada y salida de propósito general.

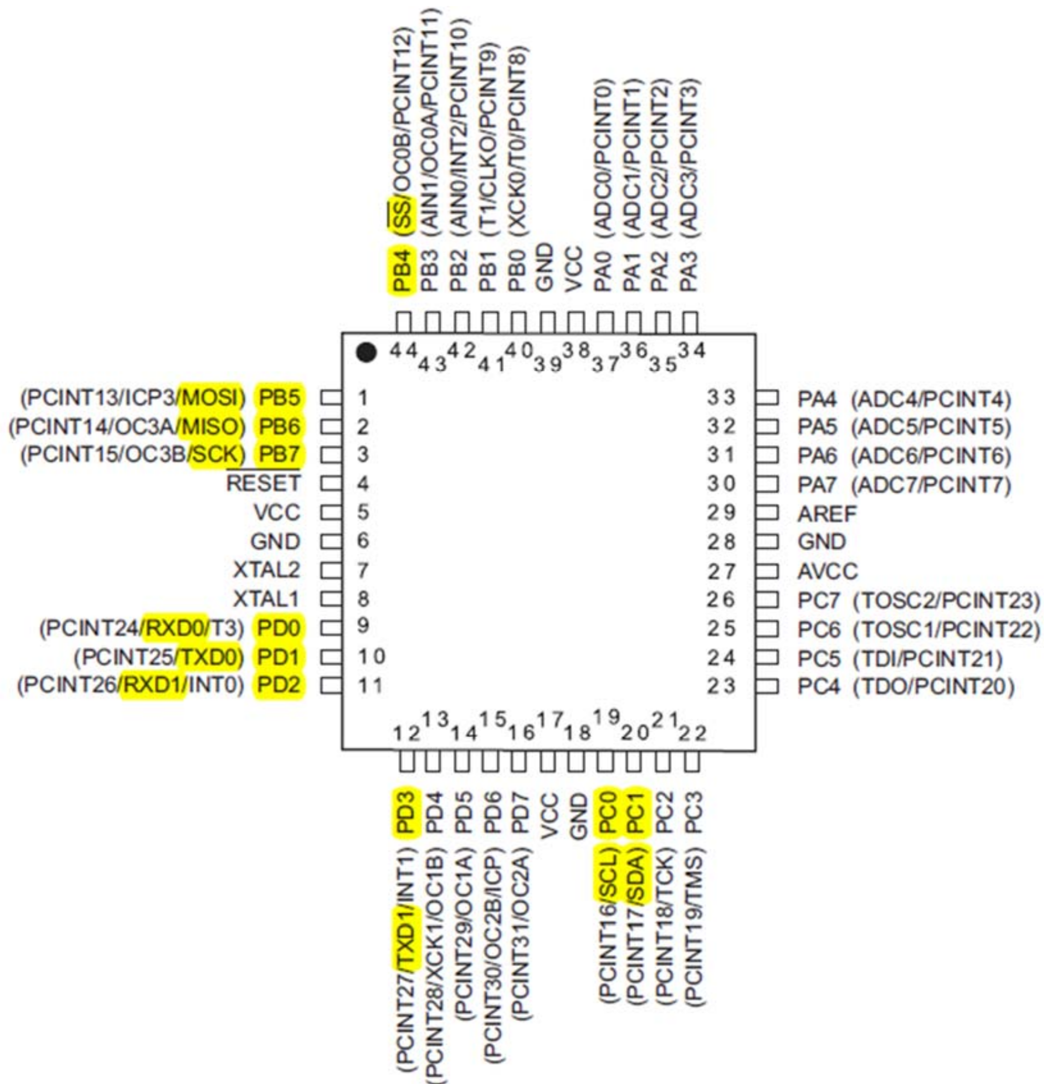

Figura 1. Configuración de los pines del microcontrolador ATmega1284P [7]

- Puertos de Expansión

En ambos laterales del módulo, se hayan dos puertos de expansión que permiten conectar otros dispositivos mediante los interfaces de comunicación UART, SPI e I2C. La conexión de estas líneas se muestra en la Tablas 8 y 9.

P1	Nomenclatura	Función/es	Utilización
1	PB3 / AIN1 / OC0A / PCINT11	Línea digital. Modulación por ancho de pulso.	Disponible
2	PD4 / OC1B / XCK1 / PCINT28	Línea digital. Modulación por ancho de pulso.	Disponible
3	PC0 / SCL / PCINT16	Línea digital. Reloj de interfaz I2C.	I2C
4	PC1 / SDA / PCINT17	Línea digital. Bus de datos de interfaz I2C.	I2C
5	PC2 / TCK / PCINT18	Línea digital. Reloj de temporizador.	Disponible
6	PC3 / TMS / PCINT19	Línea digital.	Disponible
7	PC4 / TDO / PCINT20	Línea digital.	Disponible
8	PC5 / TDI / PCINT21	Línea digital.	Disponible
9	PD0 / T3 / RXD0 / PCINT24	Línea digital. Línea de recepción UART0.	UART0
10	PD1 / TXD0 / PCINT25	Línea digital. Línea de transmisión UART0.	UART0
11	PD2 / INT0 / RXD1 / PCINT26	Línea digital. Línea de recepción UART1.	UART1
12	PD3 / INT1 / TXD1 / PCINT27	Línea digital. Línea de transmisión UART1.	UART1
13	+3V3	Tensión de alimentación 3.3 V regulada.	Alimentación

Tabla 1. Puerto de expansión P1

P1	Nomenclatura	Función/es	Utilización
1	PB4 / SS / OC0B / PCINT12	Línea digital. Slave Select por defecto (SPI)	SPI
2	PB5 / MOSI / ICP3 / PCINT13	Línea digital. Bus direccional MOSI (SPI)	SPI
3	PB6 / MISO / OC3A / PCINT14	Línea digital. Bus direccional MISO (SPI)	SPI
4	PB7 / SCK / OC3B / PCINT15	Línea digital. Reloj de interfaz SPI.	SPI
5	RESET	Reset del sistema, activo a nivel bajo.	RESET
6	PD6 / ICP / OC2B / PCINT30	Línea digital. Modulación por ancho de pulso.	Disponible
7	PD7 / OC2A / PCINT31	Línea digital. Modulación por ancho de pulso.	Disponible
8	PA1 / ADC1 / PCINT1	Entrada analógica. Interrupción cambio de entrada.	Disponible
9	PA2 / ADC2 / PCINT2	Entrada analógica. Interrupción cambio de entrada.	Disponible
10	PA3 / ADC3 / PCINT3	Entrada analógica. Interrupción cambio de entrada.	Disponible
11	AREF	Referencia de tensión para entradas analógicas.	Disponible
12	GND	Tensión de referencia / Masa del circuito.	Alimentación
13	+5V	Tensión de alimentación 5 V regulada.	Alimentación

Tabla 2. Puerto de expansión P2

En color oro se han recalcado los pines utilizados para la comunicación con los distintos módulos utilizados en la aplicación.

Anexo: Calibración del acelerómetro

A continuación, se lleva a cabo la calibración del giróscopo. Dicho procedimiento consiste en obtener el offset de medida del sensor (error medio).

Los valores de medida de la aceleración obtenidos para la calibración del acelerómetro se recogen en la Tabla 10.

Medida	X (m/s2)	Y (m/s2)	Z (m/s2)	Módulo
1	-0,30625	0,87568	-9,32148	9,3675287
2	-0,17226	1,00967	-9,31670	9,3728334
3	-0,15791	0,77998	-9,28320	9,3172478
4	0,00000	0,89004	-9,34063	9,38293379
5	-0,27275	0,88525	-9,32627	9,37215942
6	-0,11484	0,91396	-9,29277	9,33831575
7	-0,15313	0,97617	-9,34541	9,39750197
8	0,05264	0,86132	-9,36455	9,40422455
9	0,03350	0,80390	-9,27363	9,30847174
10	-0,10018	0,77998	-9,35020	9,38320681

Tabla 3. Medidas de la aceleración

Los valores de error medio obtenidos se recogen en la Tabla 11.

	X (m/s2)	Y (m/s2)	Z (m/s2)
Error medio	0,1191183	-0.877595	-0,4785163

Tabla 4. Error medio de los valores de aceleración

Tras aplicar los valores de corrección tomados del error medio, las medidas obtenidas quedan tal y como se muestra en la Tabla 12.

Medida	X (m/s2)	Y (m/s2)	Z (m/s2)	Módulo	Error (m/s2)
1	-0,18713	-0,00192	-9,80000	9,80178	-0,00178
2	-0,05314	0,13208	-9,79522	9,79625	0,00375
3	-0,03879	-0,09762	-9,76172	9,76228	0,03772
4	0,11912	0,01245	-9,81914	9,81987	-0,01987
5	-0,15363	0,00765	-9,80479	9,80599	-0,00599
6	0,00428	0,03637	-9,77129	9,77136	0,02864
7	-0,03401	0,09858	-9,82393	9,82448	-0,02448
8	0,17175	-0,01628	-9,84307	9,84458	-0,04458
9	0,15261	-0,07370	-9,75215	9,75362	0,04638
10	0,01894	-0,09762	-9,82871	9,82922	-0,02922
				Error medio	-0,00094

Tabla 5. Medidas de la aceleración tras aplicar la corrección

Anexo: Calibración del giróscopo

A continuación, se lleva a cabo la calibración del giróscopo. Dicho procedimiento consiste en obtener el offset de medida del sensor (error medio).

Los valores de medida de velocidad angular obtenidos para la calibración del giróscopo se recogen en la Tabla 13.

Medida	X (º/s)	Y (º/s)	Z (º/s)	Modulo
1	-1,7099	-0,9924	73,4962	73,5228
2	-1,1603	-0,0611	73,5267	73,5359
3	-1,1756	-0,1374	72,7939	72,8035
4	-1,0534	-0,3511	73,2824	73,2909
5	-1,0382	-0,1069	73,3435	73,3509
6	-1,0387	0,1221	72,7481	72,7556
7	-1,3435	-0,1832	72,3664	72,3791
8	-1,0382	-0,1985	72,5802	72,5878
9	-0,3359	0,3511	72,0611	72,0627
10	-1,2672	-0,0153	71,7099	71,7211

Tabla 6. Medidas de la velocidad angular

Los valores de error medio obtenidos se recogen en la Tabla 14.

	X (º/s)	Y (º/s)	Z (º/s)
Error medio	1,1161	0.1573	-72,7908

Tabla 7. Error medio de los valores de velocidad angular

Tras aplicar los valores de corrección tomados del error medio, las medidas obtenidas quedan tal y como se muestra en la Tabla 15.

Modulo	Medida	X (º/s)	Y (º/s)	Z (º/s)	Módulo	Error (º/s)
73,5228	1	-0,5938	-0,8351	0,7054	1,2440	1,2440
73,5359	2	-0,0442	0,0962	0,7359	0,7435	0,7435
72,8035	3	-0,0595	0,0198	0,0031	0,0628	0,0628
73,2909	4	0,0626	-0,1939	0,4916	0,5322	0,5322
73,3509	5	0,0779	0,0504	0,5527	0,5604	0,5604
72,7556	6	0,0774	0,2794	-0,0427	0,2930	0,2930
72,3791	7	-0,2274	-0,0260	-0,4244	0,4822	0,4822
72,5878	8	0,0779	-0,0412	-0,2107	0,2284	0,2284
72,0627	9	0,7802	0,5084	-0,7298	1,1831	1,1831
71,7211	10	-0,1511	0,1420	-1,0809	1,1006	1,1006
					Error medio	0,6430

Tabla 8. Medidas de la velocidad angular tras aplicar la corrección

Anexo: Calibración del magnetómetro

A continuación, se lleva a cabo la calibración del magnetómetro. Dicho procedimiento consiste en obtener el offset de medida del sensor (Error medio).

Los valores de medida del campo magnético en las direcciones X e Y, tangentes a la superficie terrestre, se recogen en la Tabla 16.

Orientación norte (º)	X (Ga)	Y (Ga)	Modulo
0	-0,187	-0,397	0,439
45	0,109	-0,236	0,260
90	0,249	0,093	0,266
135	0,116	0,518	0,530
180	-0,204	0,598	0,632
225	-0,614	0,476	0,777
270	-0,661	0,050	0,663
315	-0,479	-0,292	0,561
180	-0,188	0,579	0,609

Tabla 9. Medidas del campo magnético

En la Figura 40 se representa en los ejes X e Y las medidas tomadas.

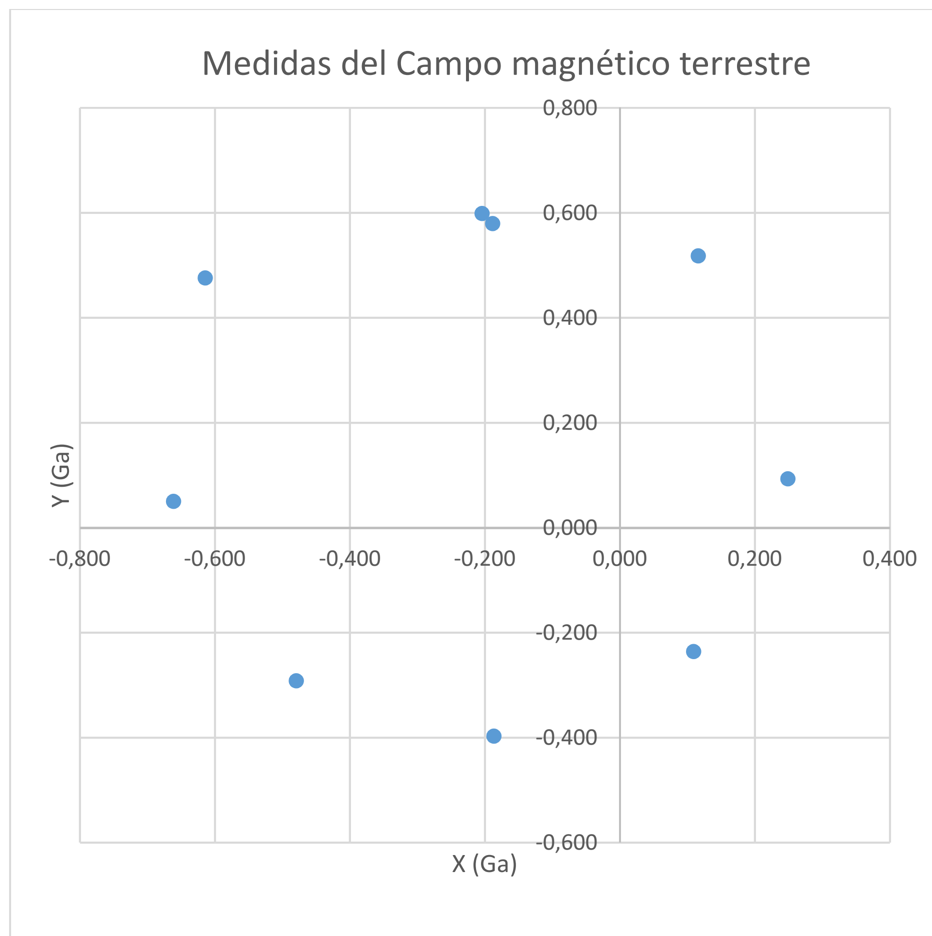

Figura 2. Representación X – Y de las medidas de campo magnético

Los valores de error medio obtenidos se recogen en la Tabla 17.

	X (Ga)	Y (Ga)
Error medio	0,206055	-0,09082

Tabla 10. Error medio de las medidas de campo magnético

Tras aplicar los valores de corrección tomados de la desviación media, las medidas obtenidas quedan tal y como se muestra en la Tabla 18.

Orientación norte (º)	X (Ga)	Y (Ga)	Modulo	Argumento (rad)	Normalización (º)	Error (º)
0	0,020	-0,488	0,489	-1,531	2,288	2,288
45	0,315	-0,327	0,454	-0,804	43,953	1,047
90	0,455	0,002	0,455	0,004	90,246	0,246
135	0,322	0,427	0,535	0,924	142,943	7,943
180	0,002	0,507	0,507	1,567	179,782	0,218
225	-0,408	0,385	0,561	0,756	226,691	1,691
270	-0,455	-0,041	0,457	-0,090	275,150	5,150
315	-0,273	-0,383	0,470	-0,951	324,464	9,464
180	0,018	0,488	0,489	1,535	182,059	2,059
					Error medio	1,773

Tabla 11. Medidas del campo magnético tras aplicar la corrección

En la Figura 41 se representa en los ejes X e Y los valores obtenidos tras la corrección.

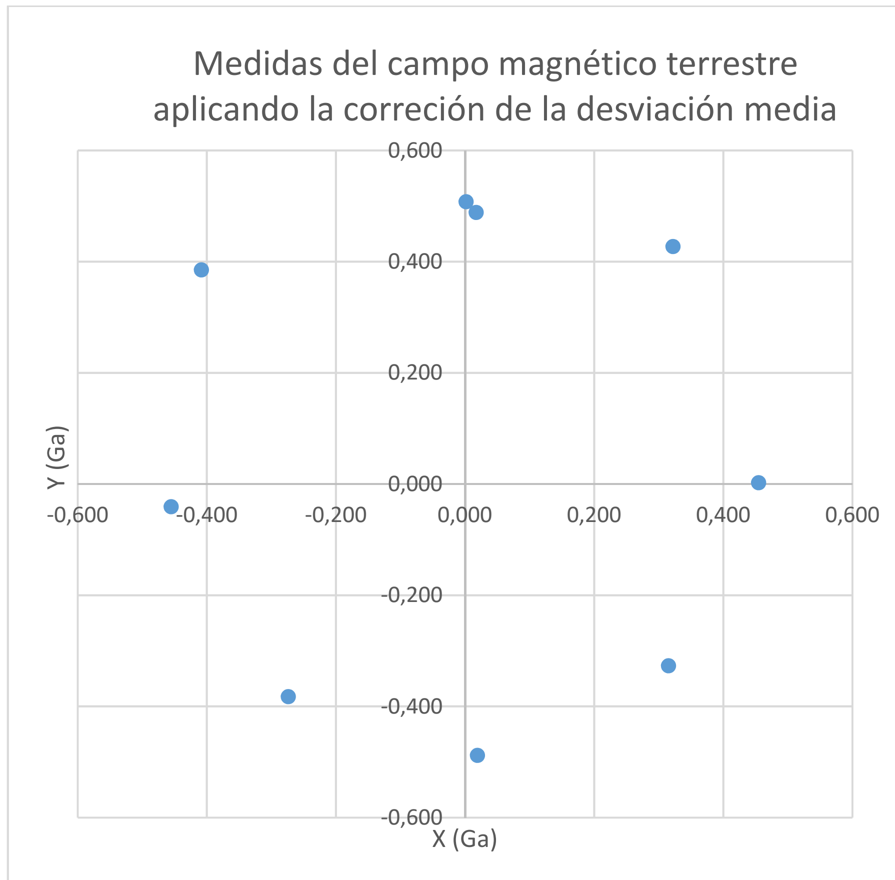

Figura 3. Representación X-Y de las medidas del campo magnético tras la corrección

Como el módulo del campo magnético no es relevante para la aplicación, se trabaja con valores unitarios, eliminando así la posible influencia de los valores extremos de X e Y sobre el argumento, quedando así la representación gráfica de la Figura 42.

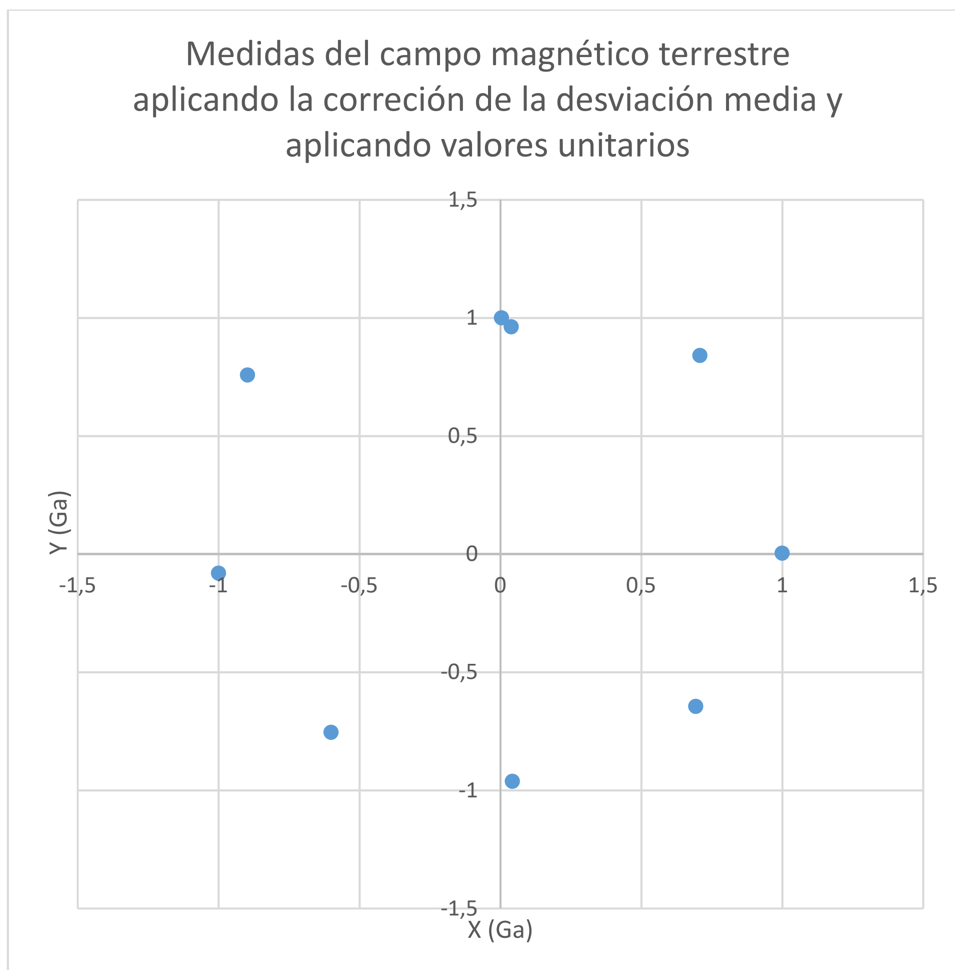

Figura 4. Representación X-Y de las medidas de campo magnético tras la corrección y con valores unitarios

Anexo: Planificación

El conjunto de tareas que conforman la aplicación se recogen en la Tabla 19.

Tarea	C	D	T	S
RXBee	<0.005 ms	0.102 ms	-	0.102 ms
RGPS	<0.005 ms	0.102 ms	-	0.102 ms
Alarma	0.46 ms	20 ms	20 ms	-
Orient	0.87 ms	50 ms	50 ms	
TXBee	14.25 ms	500 ms	500 ms	-
TBt	14.96 ms	250	250	-

Tabla 12. Conjunto de tareas que conforman la aplicación

El conjunto de tareas a planificar se recoge en la Tabla 20.

Tarea	C	D	T
Alarma	0.46 ms	20 ms	20 ms
Orient	0.87 ms	20 ms	20 ms
TXBee	14.25 ms	500 ms	500 ms
TBt	14.96 ms	250 ms	250 ms

Tabla 13. Conjunto de tareas a planificar

Se comienza seleccionando la duración del hiperperiodo M:

$$M = mcm\ (20, 20, 250,500) = 500$$

A continuación, se comprueba qué duración del marco (m) cumple las condiciones necesarias para que todas las tareas cumplan plazos y no se produzca desbordamiento:

$$m \ \leq \min(D_i)\ , m \geq \max(C_i)\ , \exists k\colon M = k * m\ \rightarrow m = 20$$

Se elige m = 20, ya que es marco de mayor tamaño posible, lo que evitará en mayor medida realizar la partición de alguna de las tareas.

Se comprueba que para m = 20 se cumpla la última condición:

$$Alarma\colon 20 + \big(20 - mcd(20{,}20)\big) = 20\ \leq 20$$

$$Orient\colon 20 + \big(20 - mcd(20{,}20)\big) = 20 \leq 20$$

$$TXBee\colon 20 + \big(20 - mcd(20{,}250)\big) = 30 \leq 250$$

$$TBt\colon 20 + \big(20 - mcd(20{,}500)\big) = 20 \leq 500$$

Finalmente, el ejecutivo cíclico consta de un marco principal de 500 ms y 25 marcos secundarios de 20 ms. La planificación queda de esta forma:

Marco 1 → Ejecuta: Alarma, Orient y TXBee. La suma de tiempo más el tiempo de cómputo de las tareas RXBee y RGPS es:

$$0.46 + 0.87 + 14.25 + 2 * 0.005 * \frac{20}{0.102} = 17.54\ s < m = 20\ ms$$

Marco 2 → Ejecuta: Alarma, Orient y TBt. La suma de tiempo más el tiempo de cómputo de las tareas RXBee y RGPS es:

$$0.46 + 0.87 + 14.96 + 2 * 0.005 * \frac{20}{0.102} = 18.25\, ms < m = 20\, ms$$

Marco 3 → Ejecuta: Alarma y Orient. La suma de tiempo más el tiempo de cómputo de las tareas RXBee y RGPS es:

$$0.46 + 0.87 + 2 * 0.005 * \frac{20}{0.102} = 6.23\, ms < m = 20\, ms$$

[...]

Marco 13 → Ejecuta: Alarma, Orient y TBt. La suma de tiempo más el tiempo de cómputo de las tareas RXBee y RGPS es:

$$0.46 + 0.87 + 14.96 + 2 * 0.005 * \frac{20}{0.102} = 18.25\, ms < m = 20\, ms$$

[...]

Marco 25 → Ejecuta: Alarma, Orient y TXBee. La suma de tiempo más el tiempo de cómputo de las tareas RXBee y RGPS es:

$$0.46 + 0.87 + 14.25 + 2 * 0.005 * \frac{20}{0.102} = 20.48\, ms < m = 50\, ms$$

Como se puede observar, en ninguno de los marcos se produce ningún desbordamiento, por lo que el ejecutivo cíclico funcionará de forma correcta.

Anexo: Comunicación I^2C

Las reglas de comunicación I2C utilizadas para la lectura y escritura de los registros de los diferentes sensores que componen la unidad MARG se describen en las siguientes imágenes.

Las reglas para la lectura de datos se presentan en las Figuras 43 y 44:

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Figura 5. Reglas de comunicación I2C para la lectura de un registro [11]

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Figura 6. Reglas de comunicación I2C para la lectura de varios registros consecutivos [11]

Las reglas para la escritura de datos se presentan en las Figuras 45 y 46:

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Figura 7. Reglas de comunicación I2C para la escritura de un registro [11]

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Figura 8. Reglas de comunicación I2C para la escritura de varios registros consecutivos [11]

Las abreviaturas utilizadas en las figuras anteriores tienen el siguiente significado:

S → Comando START de comienzo de la comunicación

P → Comando STOP de finalización de la comunicación

AD + W → Dirección del esclavo + Comando de escritura WRITE

AD + R → Dirección del esclavo + Comando de lectura READ

RA → Puntero a la dirección de memoria del esclavo RA

DATA → Dato enviado por el maestro o dato enviado por el esclavo

ACK → Comando o dato recibido

NACK → Comando o dato recibido y finalización de la lectura o de la escritura

Anexo: Fichero SCI.c

Código implementado en el fichero SCI.c:

```
/****************************************************************************/
/*                         Used modules                                     */
/****************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

#include "sci.h"

/****************************************************************************/
/*                         Local variables                                  */
/****************************************************************************/

/****************************************************************************/
/*                         Prototypes of local functions                    */
/****************************************************************************/

static void (*scia_tx_callback)(void);  // Función que hace de puntero a un
argumento de SCIa_Init().
static void (*scib_tx_callback)(void);  // Función que hace de puntero a un
argumento de SCIb_Init().
static void (*scia_rx_callback)(unsigned char);  // Función que hace de puntero a
un argumento de SCIa_Init();
static void (*scib_rx_callback)(unsigned char);  // Función que hace de puntero a
un argumento de SCIb_Init();

static void SCIARX_ISR (void) ; // Función ejecutable en la interrupción del SCIa
para la recepción de un dato.
static void SCIBRX_ISR (void) ; // Función ejecutable en la interrupción del SCIb
para la recepción de un dato.
static void SCIATX_ISR (void) ; // Función ejecutable en la interrupción del SCIa
para la trasmisión de un dato.
static void SCIBTX_ISR (void) ; // Función ejecutable en la interrupción del SCIb
para la trasmisión de un dato.

/****************************************************************************/
/*                         Prototypes of exported functions                 */
/****************************************************************************/

void SCIa_Init(void (*tx_int)(void), void (*rx_int)(unsigned char)); // Función
de inicialización del UART0.
void SCIb_Init(void (*tx_int)(void), void (*rx_int)(unsigned char)); // Función
de inicialización del UART1.
void SCIa_PutChar(unsigned char Send_Data); // Función que trasmite un char por
el UART0.
void SCIa_TXINT(TX_STATUS st); // Función que habilita la interrupción  trasmitir
un dato por la UART0.
void SCIb_PutChar(unsigned char Send_Data); // Función que trasmite un char por
el UART1.
void SCIb_TXINT(TX_STATUS st); // Función que habilita la interrupción al
trasmitir un dato por la UART1.

/****************************************************************************/
/*                         Exported functions                               */
/****************************************************************************/

// Inicialización de la SCIa
```

```
void SCIa_Init(void (*tx_int)(void), void (*rx_int)(unsigned char))
  {
      /* Para una frecuencia de oscilación (fosc) de 8MHz y
        una velocidad de trasmisión (BR) de 9600 bps obtenemos
        un BRR de 51 (BR=fosc/16(BRR+1)) */

      const unsigned int BRR = 103 ;
      UBRR0 = BRR ;

   /* SCI - USART Control and Status Register B
            RXEN - Habilitación de la recepción
                    TXEN - Habilitación de la trasmisión
                    RXCIE - Habilitación de la interrupción de recepción
                    TXCIE - Habilitación de la interrupción de trasmisión */

      UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(0<<TXCIE0);

   /* SCI - USART Control and Status Register C
            USBS - Un bit de stop
                    UCSZ - 8 bits para el dato */

      UCSR0C = (0<<USBS0)|(3<<UCSZ00) ;

      // Asignamos las funciones del argumento
      scia_tx_callback=tx_int;
      scia_rx_callback=rx_int;

   return ;
}

// Inicialización de la SCIb
void SCIb_Init(void (*tx_int)(void), void (*rx_int)(unsigned char))
  {
      /* Para una frecuencia de oscilación (fosc) de 8MHz y
        una velocidad de trasmisión (BR) de 9600 bps obtenemos
        un BRR de 51 (BR=fosc/16(BRR+1)) */

   const unsigned int BRR = 103 ;

   UBRR1 = BRR ;

   /* SCI - USART Control and Status Register A */

   /* SCI - USART Control and Status Register B
            RXEN - Habilitación de la recepción
                    TXEN - Habilitación de la trasmisión
                    RXCIE - Habilitación de la interrupción de recepción
                    TXCIE - Habilitación de la interrupción de trasmisión */

      UCSR1B = (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1)|(0<<TXCIE1);

   /* SCI - USART Control and Status Register C
            USBS - one stop bit
                    UCSZ - 8 bits data size */

      UCSR1C = (0<<USBS1)|(3<<UCSZ10) ;

      // Asignamos las funciones del argumento
      scib_tx_callback=tx_int;
      scib_rx_callback=rx_int;

   return ;
```

```
}

// Enviar dato por la UART0
void SCIa_PutChar(unsigned char Send_Data)
{
  while(!(UCSR0A & (1<<UDRE0))); // Esperamos a que el registro haya enviado
  UDR0 = Send_Data;              // Escribimos el dato en el registro de envio

  return ;
}

// Habilitación de la interrupción de trasmisión de la UART0
void SCIa_TXINT(TX_STATUS st) {
	if (st == ENABLED) UCSR0B |= (1<<TXCIE0) ; // Interrupción deshabilitada
	else UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(0<<TXCIE0) ; //
Interrupción habilitada
}

// Enviar dato por la UART1
void SCIb_PutChar(unsigned char Send_Data)
{
  while(!(UCSR1A & (1<<UDRE1))); // Esperamos a que el registro haya enviado
  UDR1 = Send_Data;              // Escribimos el dato en el registro de envio

  return ;
}

// Habilitación de la interrupción al enviar por UART1
void SCIb_TXINT(TX_STATUS st) {
	if (st == ENABLED) UCSR1B |= (1<<TXCIE1) ; // Interrupción deshabilitada
	else UCSR1B &= ~(0<<TXCIE1) ;  // Interrupción habilitada
}


/*****************************************************************************/
/*                        Local functions                                    */
/*****************************************************************************/

// Interrupción de recepción del SCIa
 ISR (USART0_RX_vect)
 {
	SCIARX_ISR () ;
 }

 static void SCIARX_ISR (void)
 {
	unsigned char SCI0_Data ;

	SCI0_Data = UDR0 ; // Reset del flag de la interrupción

	if(scia_rx_callback != 0)
	(*scia_rx_callback)(SCI0_Data); // Ejecutamos la función asignada en la
inicialización

 }

// Interrupción de trasmisión del SCIa
 ISR (USART0_TX_vect)
 {
	SCIATX_ISR () ;
 }
```

```
 static void SCIATX_ISR (void)
 {

        if(scia_tx_callback != 0)
        (*scia_tx_callback)(); // Ejecutamos la función asignada en la
inicialización

 }

// Interrupción de recepción del SCIb
 ISR (USART1_RX_vect)
 {
        SCIBRX_ISR () ;
 }

 static void SCIBRX_ISR (void)
 {
        unsigned char SCI0_Data ;

        SCI0_Data = UDR1 ; // Reset del flag de la interrupción

        if(scib_rx_callback != 0)
        (*scib_rx_callback)(SCI0_Data); // Ejecutamos la función asignada en la
interrupción

 }

// Interrupción de trasmisión del SCIb
 ISR (USART1_TX_vect)
 {
        SCIBTX_ISR () ;
 }


 static void SCIBTX_ISR (void)
 {

        if(scib_tx_callback != 0)
        (*scib_tx_callback)(); // Ejecutamos la función asignada en la
interrupción

 }
```

Anexo: Fichero SCI.h

Código implementado en el fichero SCI.h:

```
#ifndef _SCI_H
#define _SCI_H

/******************************************************************************/
/*                        Typedefs and structures                             */
/******************************************************************************/

typedef enum {ENABLED, DISABLED} TX_STATUS ; // Activar o desactivar las
interrupciones de trasmisión

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

void SCIa_Init(void (*tx_int)(void), void (*rx_int)(unsigned char)); // Función
de inicialización del UART0.
void SCIb_Init(void (*tx_int)(void), void (*rx_int)(unsigned char)); // Función
de inicialización del UART1.
void SCIa_PutChar(unsigned char Send_Data); // Función que trasmite un char por
el UART0.
void SCIa_TXINT(TX_STATUS st); // Función que habilita la interrupción  trasmitir
un dato por la UART0.
void SCIb_PutChar(unsigned char Send_Data); // Función que trasmite un char por
el UART1.
void SCIb_TXINT(TX_STATUS st); // Función que habilita la interrupción al
trasmitir un dato por la UART1.


#endif
```

Anexo: Fichero SPI.c

Código implementado en el fichero SPI.c:

```
/***************************************************************************/
/*                          Used modules                                   */
/***************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "SPI.h"

/***************************************************************************/
/*                          Local variables                                */
/***************************************************************************/

/***************************************************************************/
/*                          Prototypes of local functions                  */
/***************************************************************************/

static void (*spi_tx_callback)(unsigned char); // Función que hace de puntero a
un argumento
static void (*spi_rx_callback)(unsigned char); // Función que hace de puntero a
un argumento

static void SPITRX_ISR (void); // Función ejecutable en la interrupción del SPI

/***************************************************************************/
/*                          Prototypes of exported functions               */
/***************************************************************************/

void SPI_MasterTransmit(char cData); // Trasmisión de un byte por el SPI
void SPI_MasterInit(void (*tx_int)(unsigned char),void (*rx_int)(unsigned char));
// Inicialización del SPI
void SPI_SPIE(TRX_STATUS st); // Función que habilita / deshabilita la
interrupción SPI

/***************************************************************************/
/*                          exported functions                             */
/***************************************************************************/

// Inicialización del SPI
void SPI_MasterInit(void (*tx_int)(unsigned char), void (*rx_int)(unsigned char))
{

       /* Establecemos como salida el puerto MOSI y SCK y como entradas el MISO y
SS */
       DDRB = (1<<DDB5)|(1<<DDB7);
       PORTB = (1<<PORTB4); // Conectado con resistencia de pull-up para evitar
valor 0 y entrada en modo esclavo


       /* SPCR: SPI Control Register
              SPIE >> habilitación de la interrupción
              SPE  >> habilitación de la comunicación SPI
              DORD >> inicio de la comunicación a partir del LSB o el MSB
              MSTR >> Selección de la función de Master
              CPOL >> polaridad del reloj (pulsos de subida)
              CPHA >> Envio de datos con el flanco de subida.
              SPR0/1 >> Selección de la frecuencia de reloj. // 14204 bps - 15625
       */
```

```
	SPCR =
(1<<SPIE)|(1<<SPE)|(1<<MSTR)|(1<<SPR1)|(1<<SPR0)|(0<<CPOL)|(0<<CPHA)|(0<<DORD);

	/* SPSR: SPI Status Register
		SPIF >> Flag de finalización de la trasmisión (Flag de
interrupción)
		SPI2X >> Bit para completar la selección de la frecuencia de reloj.
	*/

	SPSR = (0<<SPI2X);

	// Asignamos las funciones del argumento
	spi_tx_callback = tx_int;
	spi_rx_callback = rx_int;

}

// Trasmisión de un byte a través del SPI
void SPI_MasterTransmit(char cData)
{
	/* Comenzamos la trasmisión con la escritura del dato a trasmitir */
	SPDR = cData;
	/* Esperamos a la finalización de la tramisión */
	//while(!(SPSR & (1<<SPIF))); //eliminado al usar interrupciones

}

// Habilitación / Deshabilitación de la interrupción
void SPI_SPIE(TRX_STATUS st)
{
	if (st == ENABLEDSPI) SPCR |= (1<<SPIE); // Interrupción habilitada
	else SPCR &= ~(0<<SPIE); // Interrupción deshabilitada
}


/******************************************************************************/
/*                        local functions
*/
/******************************************************************************/

// Interrupción de fin de trasmisión del SPI
ISR (SPI_STC_vect)
{
	SPITRX_ISR ();
}

// Función que ejecutamos en la interrupción tras finalizar la trasmisión SPI
static void SPITRX_ISR(void)
{
	unsigned char SPI_Data;

	SPI_Data = SPDR;

	if(spi_rx_callback !=0)
	(*spi_rx_callback)(SPI_Data); // Ejecutamos la función asignada para la
interrupción (Recepción)

	if(spi_tx_callback != 0)
	(*spi_tx_callback)(SPI_Data); // Ejecutamos la función asignada para la
interrupción (Trasmisión)
}
```

Anexo: Fichero SPI.h

Código implementado en el fichero SPI.h:

```
#ifndef SPI_H_
#define SPI_H_

/******************************************************************************/
/*                        Typedefs and structures                             */
/******************************************************************************/

typedef enum {ENABLEDSPI, DISABLEDSPI} TRX_STATUS ; // Variable que utilizamos
para habilitar o deshabilatar las interrupciones

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

void SPI_MasterTransmit(char cData); // Tramisión de un byte a través del SPI
void SPI_MasterInit(void (*tx_int)(unsigned char), void (*rx_int)(unsigned
char)); // Inicialización del SPI
void SPI_SPIE(TRX_STATUS st); // Función que habilita / deshabilita la
interrupción SPI

#endif /* SPI_H_ */
```

Anexo: Fichero I2C.c

Código implementado en el fichero I2C.c:

```
/*****************************************************************************/
/*                         Used modules                                      */
/*****************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <compat/twi.h>

#include "I2C.h"

/*****************************************************************************/
/*                         Local variables                                   */
/*****************************************************************************/
#define MAX_TRIES 50 // Número máximo de intento para establecer la comunicación

#define I2C_START 0 // Indicativo para la trasmisión del comando START
#define I2C_DATA_ACK  1 // Indicativo para el comienzo de la trasmisión/lectura
de un dato
#define I2C_DATA_NACK 2 // Indicativo para la lectura del ultimo dato
#define I2C_STOP  3 // Indicativo para la trasmisión del comando STOP

static MAGNETOMETER_Data Measure_Magnetometer;
static ACCELEROMETER_Data Measure_Accelerometer;
static GYROSCOPE_Data Measure_Gyroscope;

static int Magnetometer_Measure_Trasmited;
static int Gyroscope_Measure_Trasmited;
static int Accelerometer_Measure_Trasmited;

/*****************************************************************************/
/*                         Prototypes of local functions                     */
/*****************************************************************************/

static unsigned char i2c_transmit(unsigned char type); // Set de los bits
necesarós para la trasmisión
static void Accelerometer_Gyroscope_Init (void); // Inicialización del los
registros del MPU6050
static void Magnetometer_Init (void); // Inicialización de los registros del
magnetómetro
static int i2c_write_byte(unsigned int i2c_register, char data, I2C_SENSOR
sensor); // Escritura de un byte en un registro de los sensores
static int i2c_read (double *valueX, double *valueY, double *valueZ, I2C_SENSOR
sensor); // Lectura de 16 bits de las medidas en los ejes X, Y y Z de los
sensores
static int i2c_read_byte (int* data, unsigned int i2c_register, I2C_SENSOR
sensor);
static void Magnetometer_Init (void); // Inicialización Magnetómetro
static void Accelerometer_Gyroscope_Init (void); // Inicialización acelerómetro y
giróscopo

/*****************************************************************************/
/*                         Prototypes of exported functions                  */
/*****************************************************************************/

void I2C_Init(void); // Inicialización de la comunicación 2-wire y de los
sensores
GYROSCOPE_Data Get_Data_Gyroscope (void); // Lectura de las medidas
```

```
MAGNETOMETER_Data Get_Data_Magnetometer (void); // Lectura de las medidas
ACCELEROMETER_Data Get_Data_Accelerometer (void); // Lectura de las medidas
int Received_Measure_Magnetometer (void); // Informa sobre la correcta lectura de
las medidas
int Received_Measure_Gyroscope (void); // Informa sobre la correcta lectura de
las medidas
int Received_Measure_Accelerometer (void); // Informa sobre la correcta lectura
de las medidas
void Get_Measure_Magnetometer (void); // Obtiene las medidas
void Get_Measure_Accelerometer (void); // Obtiene las medidas
void Get_Measure_Gyroscope (void); // Obtiene las medidas

/*****************************************************************************/
/*                        exported functions
*/
/*****************************************************************************/

// Inicialización de la comunicación 2-wire y de los sensores
void I2C_Init(void)
{

	/*TWBR: Bit rate register
		SCL frequency = (CPU Clock frequency)/(16+2*TWBR*4^(TWPS))
		Con TWPS = 1 y TWBR = 3 >>> SCLf = 200 kHz
	*/

	TWBR = 0b00000011; // 3

	/* TWSR: TWI Status Register
		TWPS >> Prescaler (TWPS0 y TWPS1)
	*/

	TWSR = (0<<TWPS1)|(0<<TWPS0);

	/* TWCR: TWI Control Register
		TWEA >> Activamos/Desacitvamos de la comunicación
		TWEN >> Habilitación de la comunicación
		TWIE >> Habilitación de la interrupción
	*/

	TWCR = (0<<TWEA)|(1<<TWEN)|(0<<TWIE);

	// Inicialización de los sensores

	Accelerometer_Gyroscope_Init();
	Magnetometer_Init();

}

// Lectura de las medidas
void Get_Measure_Magnetometer (void){
	if(i2c_read(&Measure_Magnetometer.X, &Measure_Magnetometer.Z,
&Measure_Magnetometer.Y, MAGNETOMETER)){
		Magnetometer_Measure_Trasmited = 1;
		} else {
		Magnetometer_Measure_Trasmited = 0;
	}
}

// Lectura de las medidas
void Get_Measure_Accelerometer (void){
```

```
	if(i2c_read(&Measure_Accelerometer.X, &Measure_Accelerometer.Y, 
&Measure_Accelerometer.Z, ACCELEROMETER)){
		Accelerometer_Measure_Trasmited = 1;
		} else {
		Accelerometer_Measure_Trasmited = 0;
	}
}

// Lectura de las medidas
void Get_Measure_Gyroscope (void){
	if(i2c_read(&Measure_Gyroscope.X, &Measure_Gyroscope.Y, 
&Measure_Gyroscope.Z, GYROSCOPE)){
		Gyroscope_Measure_Trasmited = 1;
		} else {
		Gyroscope_Measure_Trasmited = 0;
	}
}

MAGNETOMETER_Data Get_Data_Magnetometer (void){
	return Measure_Magnetometer;
}

ACCELEROMETER_Data Get_Data_Accelerometer (void){
	return Measure_Accelerometer;
}

GYROSCOPE_Data Get_Data_Gyroscope (void){
	return Measure_Gyroscope;
}

int Received_Measure_Magnetometer (void){
	return Magnetometer_Measure_Trasmited;
}

int Received_Measure_Gyroscope (void){
	return Gyroscope_Measure_Trasmited;
}

int Received_Measure_Accelerometer (void){
	return Accelerometer_Measure_Trasmited;
}

/******************************************************************************/
/*                        local functions                                     
*/
/******************************************************************************/

// Escritura de un byte en un registro de los sensores
static int i2c_write_byte(unsigned int i2c_register, char data, I2C_SENSOR 
sensor)
{
	unsigned char n = 0;
	unsigned char twi_status;
	unsigned int i2c_address = 0;
	char r_val = -1;

	// Selección de la dirección del sensor
	if (sensor == MAGNETOMETER){
		i2c_address = 0b00011110;
	} else {
		i2c_address = 0b01101000;
	}
```

```
	// Inicio de la comunicación:

	// Maestro: START >> ___ >> Sensor Address + W >> ___ >> Register >> ___ 
>> DATA >> ___ >> STOP // fin
	// Esclavo: _____ >> ACK >> __________________ >> ACK >> ________ >> ACK 
>> ____ >> ACK >> ____ // fin

	i2c_retry:

	// Realización máxima de 50 intentos de comunicación
	if (n++ >= MAX_TRIES) return r_val;

	// Trasmitimos la condición de inicio
	twi_status = i2c_transmit(I2C_START);
	// Comprobamos el estado de la comunicación
	if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
	if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto 
i2c_quit;

	// Enviamos la condición de escritura junto con la dirección del esclavo 
(SLA_W)
	TWDR = (i2c_address << 1 | TW_WRITE);
	// Trasmitimos el dato
	twi_status = i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if ((twi_status == TW_MT_SLA_NACK) || (twi_status == TW_MT_ARB_LOST)) goto 
i2c_retry;
	if (twi_status != TW_MT_SLA_ACK) goto i2c_quit;

	// Enviamos un puntero al registro en el que deseamos escribir
	TWDR = i2c_register;
	// Trasmitimos el dato
	twi_status = i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if (twi_status != TW_MT_DATA_ACK) goto i2c_quit;

	// Enviamos el dato que deseamos escribir
	TWDR = data;
	// Trasmitimos el dato
	twi_status = i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if (twi_status != TW_MT_DATA_ACK) goto i2c_quit;

	// I2C trasmisión correcta
	r_val=1;

	i2c_quit:

	// Trasmitimos la condición de stop
	twi_status=i2c_transmit(I2C_STOP);
	return r_val;
}

// Lectura de 16 bits de las medidas en los ejes X, Y y Z de los sensores
static int i2c_read (double *valueX1, double *valueX2, double *valueX3, 
I2C_SENSOR sensor)
{
	unsigned char n = 0;
	unsigned char twi_status;
	unsigned int i2c_address = 0;
	unsigned int i2c_register = 0;
```

```
	char r_val = -1;

	// Selección de la dirección del sensor y del registro que almacena las 
medidas
	if (sensor == MAGNETOMETER){
		i2c_address = 0b00011110;
		i2c_register = 0x03;
		} else {
		i2c_address = 0b01101000;
		if (sensor == ACCELEROMETER){
			i2c_register = 0b00111011;
		} else {
			i2c_register = 0b01000011;
		}
	}

	// Ejemplo de la comunicación:
	// Maestro: START >> ___ >> "Sensor Address + W >> ___ >> Register >> ___ 
>> START >> ___" >> ...
	// Esclavo: _____ >> ACK >> "__________________ >> ACK >> ________ >> ACK 
>> _____ >> ACK" >> ...

	// Maestro: ... >> SensorAddress + R >> ___ >> ____ >> ACK >> [...] >> 
NACK >> STOP // fin
	// Esclavo: ... >> _________________ >> ACK >> DATA >> ___ >> [...] >> 
____ >> ____ // fin

	// Inicio de la comunicación:
	i2c_retry:

	// Establecemos un número máximo de intentos de trasmisión
	if (n++ >= MAX_TRIES) return r_val;

	// Trasmitimos la condición de START
	twi_status=i2c_transmit(I2C_START);
	// Comprobamos el estado de la trasmisión
	if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
	if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto 
i2c_quit;

	// Enviamos la condición de escritura junto con la dirección del esclavo 
(SLA_W)
	TWDR =  i2c_address << 1 | TW_WRITE;
	// Trasmisión del dato
	twi_status=i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if ((twi_status == TW_MT_SLA_NACK) || (twi_status == TW_MT_ARB_LOST)) goto 
i2c_retry;
	if (twi_status != TW_MT_SLA_ACK) goto i2c_quit;

	// Enviamos un puntero al registro que deseamos leer
	TWDR = i2c_register;
	// Trasmitimos el dato
	twi_status = i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if (twi_status != TW_MT_DATA_ACK) goto i2c_quit;

	twi_status=i2c_transmit(I2C_STOP);

	// Trasmitimos la condición START
	twi_status=i2c_transmit(I2C_START);
	// Comprobamos el estado de la trasmisión
```

```
	if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
	if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto
i2c_quit;

	// Enviamos el comando de lectura junto con la cireccón del esclavo
(SLA_R)
	TWDR =i2c_address << 1 | TW_READ;
	// Trasmitimos el dato
	twi_status=i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if ((twi_status == TW_MR_SLA_NACK) || (twi_status == TW_MR_ARB_LOST)) goto
i2c_retry;
	if (twi_status != TW_MR_SLA_ACK) goto i2c_quit;

	// Lectura de dato (MSB X1)
	twi_status=i2c_transmit(I2C_DATA_ACK);
	if ((twi_status != TW_MR_DATA_ACK)) goto i2c_quit;
	// Almacenamiento del dato
	*valueX1=(double)(TWDR<<8);

	// Lectura del dato (LSB X1)
	twi_status=i2c_transmit(I2C_DATA_ACK);
	if ((twi_status != TW_MR_DATA_ACK)) goto i2c_quit;
	// Almacenamiento del dato
	*valueX1=*valueX1+(double)(TWDR);//dataX1=TWDR+dataX1;

	// Lectura del dato (MSB X2)
	twi_status=i2c_transmit(I2C_DATA_ACK);
	if ((twi_status != TW_MR_DATA_ACK)) goto i2c_quit;
	// Almacenamiento del dato
	//dataX2=TWDR;
	//dataX2=dataX2<<8;

	*valueX2=(double)(TWDR<<8);
	// Lectura del dato (LSB X2)
	twi_status=i2c_transmit(I2C_DATA_ACK);
	if ((twi_status != TW_MR_DATA_ACK)) goto i2c_quit;
	// Almacenamiento de dato
	*valueX2=*valueX2+(double)(TWDR);

	// Lectura del dato (MSB X3)
	twi_status=i2c_transmit(I2C_DATA_ACK);
	if ((twi_status != TW_MR_DATA_ACK)) goto i2c_quit;
	// Almacenamiento del dato
	*valueX3=(double)(TWDR<<8);
	// Lectura del dato (LSB X3)
	twi_status=i2c_transmit(I2C_DATA_NACK);
	if ((twi_status != TW_MR_DATA_NACK)) goto i2c_quit;
	// Almacenamiento del dato
	//dataX3=TWDR+dataX3;
	*valueX3=*valueX3+(double)(TWDR);
	// Trasmisión correcta
	r_val=1;

	// Transformación del valor obtenido del ADC a unidades del SI
	if(sensor==MAGNETOMETER){
		(*valueX1)=*valueX1/1024+0.2;
		(*valueX2)=*valueX2/1024;
		(*valueX3)=*valueX3/1024-0.09;
	} else if(sensor==ACCELEROMETER){
		(*valueX1)=*valueX1*9.8/2048+0.1191183;
		(*valueX2)=*valueX2*9.8/2048-0.4785163;
```

```
		(*valueX3)=*valueX3*9.8/2048-0.877595;
	} else {
		(*valueX1)=*valueX1/65.5+1.06256;
		(*valueX2)=*valueX2/65.5+0.1573;
		(*valueX3)=*valueX3/65.5-72.7908;
	}

	i2c_quit:
	// Envio del comando STOP
	twi_status=i2c_transmit(I2C_STOP);

	// Devolvemos si la trasmisión a sido correcta
	return r_val;
}

// Lectura de un byte del uno de los registros de un sensor
static int i2c_read_byte (int* data, unsigned int i2c_register, I2C_SENSOR
sensor)
{
	unsigned char n = 0;
	unsigned char twi_status;
	unsigned int i2c_address = 0;//0b00011110;
	char r_val = -1;

	// Selección de la dirección del sensor y del registro que almacena las
medidas
	if (sensor == MAGNETOMETER){
		i2c_address = 0b00011110;
		} else {
		i2c_address = 0b01101000;
	}

	// Inicio de la comunicación:

	// Maestro: START >> ___ >> "Sensor Address + W >> ___ >> Register >> ___
>> START >> ___" >> ...
	// Esclavo: _____ >> ACK >> "__________________ >> ACK >> ________ >> ACK
>> _____ >> ACK" >> ...

	// Maestro: ... >> SensorAddress + R >> ___ >> ____ >> NACK >> STOP // fin
	// Esclavo: ... >> _________________ >> ACK >> DATA >> ____ >> ____ // fin

	i2c_retry:

	// Establecemos un número máximo de intentos de trasmisión
	if (n++ >= MAX_TRIES) return r_val;

	// Trasmitimos la condición de START
	twi_status=i2c_transmit(I2C_START);
	// Comprobamos el estado de la trasmisión
	if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
	if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto
i2c_quit;

	// Enviamos la condición de escritura junto con la dirección del esclavo
(SLA_W)
	TWDR =  i2c_address << 1 | TW_WRITE;
	// Trasmisión del dato
	twi_status=i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if ((twi_status == TW_MT_SLA_NACK) || (twi_status == TW_MT_ARB_LOST)) goto
i2c_retry;
```

```
	if (twi_status != TW_MT_SLA_ACK) goto i2c_quit;

	// Enviamos un puntero al registro que deseamos leer
	TWDR = i2c_register;
	// Trasmitimos el dato
	twi_status = i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if (twi_status != TW_MT_DATA_ACK) goto i2c_quit;

	twi_status=i2c_transmit(I2C_STOP);

	// Trasmitimos la condición START
	twi_status=i2c_transmit(I2C_START);
	// Comprobamos el estado de la trasmisión
	if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
	if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto
i2c_quit;

	// Enviamos el comando de lectura junto con la cirecc�n del esclavo
(SLA_R)
	TWDR =i2c_address << 1 | TW_READ;
	// Trasmitimos el dato
	twi_status=i2c_transmit(I2C_DATA_ACK);
	// Comprobamos el estado de la comunicación
	if ((twi_status == TW_MR_SLA_NACK) || (twi_status == TW_MR_ARB_LOST)) goto
i2c_retry;
	if (twi_status != TW_MR_SLA_ACK) goto i2c_quit;

	// Lectura de dato (MSB X1)
	twi_status=i2c_transmit(I2C_DATA_NACK);
	if ((twi_status != TW_MR_DATA_NACK)) goto i2c_quit;
	// Almacenamiento del dato
	*data=TWDR;

	// Trasmisión correcta
	r_val=1;

	i2c_quit:
	// Envio del comando STOP
	twi_status=i2c_transmit(I2C_STOP);

	// Devolvemos si la trasmisión a sido correcta
	return r_val;
}

// Inicio de la trasmisión de un dato
static unsigned char i2c_transmit(unsigned char type) {
	switch(type) {
		case I2C_START:    // Enviamos la condición de START
		TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
		break;
		case I2C_DATA_ACK:     // Envio/Lectura de un dato;
		TWCR = (1 << TWINT) | (1 << TWEN) | (1<<TWEA);
		break;
		case I2C_DATA_NACK:     // Lectura del último dato;
		TWCR = (1 << TWINT) | (1 << TWEN);
		break;
		case I2C_STOP:     // Enviamos la condición de STOP
		TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
		return 0;
	}
	// Esperamos a que se confirme la trasmisión a través del flag TWINT
```

```
	while (!(TWCR & (1 << TWINT)));
	// Devuelve es estado de la trasmisión (TWSR), eliminando a través de la
mascara el registro del prescaler.
	return (TWSR & 0xF8);
}

// Inicialización del acelerómetro/giróscopo
static void Accelerometer_Gyroscope_Init (void){

	// Registro 25: Sample Rate Divider (0x19) (Giroscopo) >> 0x19
	// Sample Rate = Gyroscope Output Rate/(1+SMLRT_DIV) >> 0x07 >> 1 kHz
(Acelerometro 1kHz no problemas)

	i2c_write_byte(0x19, 0x07, ACCELEROMETER);

	// Registro 26: Configuration >> 0x1A
	// FSYNC disable and Gyroscope Output Rate 8kHz >> 0x00000000

	i2c_write_byte(0x1A, 0x00, ACCELEROMETER);

	// Registro 27: Gyroscope Configuration >> 0x1B
	// No testeamos y rango de +/-  1000 º/s >> 0b00010000

	i2c_write_byte(0x1B, 0x10, ACCELEROMETER);

	// Registro 28: Accelerometer Configuration >> 0x1C
	// No testeamos y rango de +/- 16g >> 0b00011000

	i2c_write_byte(0x1C, 0x18, ACCELEROMETER);

	// FIFO Enable >> 0x23 >> 0b00000000

	i2c_write_byte(0x23, 0x00, ACCELEROMETER);

	// Bypass mode >> 0x37
	// Able bypass mode>> 0b00000010;

	i2c_write_byte(0x37, 0x02, ACCELEROMETER);

	// User Control >> 0x6A
	// I2C, conexión auxiliar con principal,
	// desactivar FIFO>> 0b00000000;

	i2c_write_byte(0x6A, 0x00, ACCELEROMETER);

	// User Control >> 0x6B
	// Clock source to gyro reference X
	// desactivar FIFO>> 0b00000001;

	i2c_write_byte(0x6B, 0x00, ACCELEROMETER);

}

// Inicialización del magnetometro
static void Magnetometer_Init (void){

	// Configuration Register A >> 0x00
	// Actualización del valor y medidas por defecto>> 0b00011000

	i2c_write_byte(0x00, 0x18, MAGNETOMETER);

	// Configuration Register B >> 0x01
```

```
        // Campo terrestre 0.25-0.65 Ga >> +/- 0.9 Ga >> 0b00000000

        i2c_write_byte(0x01, 0x00, MAGNETOMETER);

        // Mode Register >> 0x02
        // Actualización de la medida continua >> 0x00

        i2c_write_byte(0x02, 0x00, MAGNETOMETER);

}
```

Anexo: Fichero I2C.h

Código implementado en el fichero I2C.h:

```
#ifndef I2C_H_
#define I2C_H_

/******************************************************************************/
/*                        Typedefs and structures                            */
/******************************************************************************/

	// Estructura para nombrar los sensores
	typedef enum {ACCELEROMETER, GYROSCOPE, MAGNETOMETER} I2C_SENSOR ;

	// Estructura para almacenar las medidas del magnetómetro
	typedef struct {
		double X;
		double Y;
		double Z;
	} MAGNETOMETER_Data;

	// Estructura para almacenar las medidas del acelerómetro
	typedef struct {
		double X;
		double Y;
		double Z;
	} ACCELEROMETER_Data;

	// Estructura para almacenar las medidas de giróscopo
	typedef struct {
		double X;
		double Y;
		double Z;
	} GYROSCOPE_Data;

/******************************************************************************/
/*                        Exported functions                                 */
/******************************************************************************/

	void I2C_Init(void); // Inicialización de la comunicación 2-wire y de los
sensores

	// Lectura de las medidas
	MAGNETOMETER_Data Get_Data_Magnetometer (void);
	ACCELEROMETER_Data Get_Data_Accelerometer (void);
	GYROSCOPE_Data Get_Data_Gyroscope (void);

	// Comprobación de la lectura correcta de valores
	int Received_Measure_Magnetometer (void);
	int Received_Measure_Gyroscope (void);
	int Received_Measure_Accelerometer (void);

	// Obtención de las medidas
	void Get_Measure_Magnetometer (void);
	void Get_Measure_Accelerometer (void);
	void Get_Measure_Gyroscope (void);

#endif /* I2C_H_ */
```

Anexo: Fichero CLOCk.c

Código implementado en el fichero CLOCK.c:

```
/******************************************************************************/
/*                          Used modules                                      */
/******************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include "CLOCK.h"

/******************************************************************************/
/*                          Local variables                                   */
/******************************************************************************/

static unsigned char top; // Indica el máximo valor del contador

volatile static unsigned long tick_counter;
static unsigned long Timer=0, Timer2=0;
static char TO=0, TO2 = 0;
static char Active_Timer=0, Active_Timer2 = 0;

/******************************************************************************/
/*                          Prototypes of local functions                     */
/******************************************************************************/

static void clock (void); // Función de incremento del reloj durante la
interrupción
static unsigned char Get_Time (void); // Función para la lectura del contador

/******************************************************************************/
/*                          Prototypes of exported functions                  */
/******************************************************************************/

void Init_clock (void); // Inicialización del reloj
void Reset_clock (void); // Puesta a cero del reloj
void Start_clock (void); // Activación del contador
void Stop_clock (void); // Desactivación del tontador
unsigned long Get_tick_counter (void); // Devuelve el valor del reloj
void Set_Timer (unsigned long T_ms); // Activación temporización
char Time_Out (void); // Comprobación de la finalización de la temporización
void Remove_Timer (void); // Desactivación de la temporización
void Set_Timer2 (unsigned long T_ms); // Activación temporización 2
char Time_Out2 (void); // Comprobación de la finalización de la temporización 2
void Remove_Timer2 (void); // Desactivación de la temporización
void delay_Until (unsigned long t); // Espera de t milisegundos

/******************************************************************************/
/*                          Exported functions                                */
/******************************************************************************/

// Inicialización del reloj
void Init_clock (void){
	Reset_clock(); // Inicialización del reloj (contador detenido y puesto a
cero)
	Start_clock(); // Comienza a incrementar el contador
}
```

```
// Inicialización del reloj
void Reset_clock (void){
	/* TCCR0A, Registro de control A:
			CO01A -> Modo de operación normal
			WGM0 -> Operación CTC comparar el máximo con el registro
OCR0A
	*/
	TCCR0A =
(0<<COM0A1)|(0<<COM0A0)|(0<<COM0B1)|(0<<COM0B0)|(1<<WGM11)|(0<<WGM10);

	/* TCCR0B, Registro de control B:
			WGM1 -> Operación CTC comparar el máximo con el registro
OCR0A
			CS0 -> Prescaler: 64 (1ms) / 8 (us)
	*/
	TCCR0B =(0<<WGM12)|(0<<CS02)|(1<<CS01)|(1<<CS00);

	/* OCR0A, Registro de comparación A:
			Comparación con el registro TCNT0:
				f_tcnt0 = fclock/prescaler
				top = 0.0001[s]/(1/ftcnt0) = 250 (1ms) / 2 (1us)

	*/
	top = 250;
	OCR0A = top;

	/* TIMSK0, Registro de interrupción:
			OCIE0A -> Flag de interrupción para la comparación
			TOIE0 -> Flag de interrupción para el desvordamiento
	*/
	TIMSK0 = (1<<OCIE0A)|(0<<TOIE0);


}

// Activación del contador
void Start_clock (void){
	/* TCCR0B, Registro de control B:
			CS0 -> Prescaler: 64 (1ms)(011) / 8 (us)(010)
	*/
	TCCR0B =(0<<CS02)|(1<<CS01)|(1<<CS00); // Iniciamos con un prescaler de 64
}

// Desactivación del contador
void Stop_clock (void){
	TCCR0B =(0<<CS02)|(0<<CS01)|(0<<CS00); // Detenemos el reloj
}

// Devuelve el valor del reloj
unsigned long Get_tick_counter (void){
	return tick_counter;
}

// Funciones para las temporizaciones:
void Set_Timer (unsigned long T_ms){
	Timer = Get_tick_counter()+T_ms;
	TO=0;
	Active_Timer = 1;
}
```

```
void Set_Timer2 (unsigned long T_ms){
	Timer2 = Get_tick_counter()+T_ms;
	TO2=0;
	Active_Timer2 = 1;
}
char Time_Out (void){
	return TO;
}
char Time_Out2 (void){
	return TO2;
}
void Remove_Timer (void){
	Active_Timer = 0;
	TO = 0;
}
void Remove_Timer2 (void){
	Active_Timer2 = 0;
	TO2 = 0;
}
void delay_Until (unsigned long t){
	while (t!=tick_counter){

	}
}

/*****************************************************************************/
/*                         Local functions                                   */
/*****************************************************************************/

//Función para la lectura del contador de pulsos del reloj del microprocesador
static unsigned char Get_Time (void){
	unsigned char Aux_TC;
	//unsigned char Aux_SREG;

	//Aux_SREG = SREG; // Almacenamos el flag global de interrupción

	cli(); // Desactivamos las interrupciones
	Aux_TC=TCNT0;
	sei(); // Activamos las interrupciones

	//SREG = Aux_SREG;

	return Aux_TC;
}

// Incremento del reloj
static void clock (void){
	tick_counter ++;
	if (Active_Timer && (Timer<tick_counter)){
		TO=1;
		Active_Timer = 0;
	}
	if (Active_Timer2 && (Timer2<tick_counter)){
		TO2=1;
		Active_Timer2 = 0;
	}
}
// Interrupción del timer
ISR(TIMER0_COMPA_vect){
	TIFR0 |= (1<<OCF0A) ; // Reseteamos el flag
	clock(); // Incrementamos el contador y comprobamos las temporizaciones
}
```

Anexo: Fichero CLOCK.h

Código implementado en el fichero CLOCLK.h:

```
#ifndef CLOCK_H_
#define CLOCK_H_

/*****************************************************************************/
/*                         Exported functions                                */
/*****************************************************************************/

void Init_clock (void); // Inicialización del reloj
void Reset_clock (void); // Puesta a cero del reloj
void Start_clock (void); // Activación del contador
void Stop_clock (void); // Desactivación del tontador
unsigned long Get_tick_counter (void); // Devuelve el valor del reloj
void Set_Timer (unsigned long T_ms); // Activación temporización
char Time_Out (void); // Comprobación de la finalización de la temporización
void Remove_Timer (void); // Desactivación de la temporización
void Set_Timer2 (unsigned long T_ms); // Activación temporización 2
char Time_Out2 (void); // Comprobación de la finalización de la temporización 2
void Remove_Timer2 (void); // Desactivación de la temporización
void delay_Until (unsigned long t); // Espera de t milisegundos

#endif /* CLOCK_H_ */
```

Anexo: Fichero GPS.c

Código implementado en el fichero GPS.c:

```
/****************************************************************************/
/*                          Used modules                                     */
/****************************************************************************/

#include "GPS.h"
#include "SCI.h"

#include <stdlib.h>
#include <math.h>

/****************************************************************************/
/*                          Typedefs and structures                          */
/****************************************************************************/

enum recogniced_NMEA_type {
  NONE, DATA
};

/****************************************************************************/
/*                          Local variables                                  */
/****************************************************************************/

// Funciones para almacenar e interpretar una trama NMEA
static char GPS_NMEA_received = 0 ;
static char NMEA_buffer [2][100] ;
static unsigned char NMEAB_Row, NMEAB_Index ;
static char st[8] ;
static char st2[8] ;
static char st3[8] ;
static char Error_GPS;


/****************************************************************************/
/*                          Local functions prototypes                       */
/****************************************************************************/

int Send_CHAR (char C) ; // Envia un byte a través del puerto SCIb (TX1)
static unsigned char Calculate_CHS (char *NMEA) ; // Calcula del CHS para el
envio de comandos al módulo GPS
static void Receive_NMEA (unsigned char C) ; // Almacena una trama GPGGA recibida
static char Move_to_comma (char i, NMEA_frame_type F) ; // Obtiene la posición
del caracter "," en un string
extern void DelayMs(volatile unsigned int Msec) ; // Realiza una espera en ms

/****************************************************************************/
/*                                        Exported functions prototypes
*/
/****************************************************************************/

int Init_GPS (void); // Inicialización del GPS y del SCIb
int Send_NMEA (char *NMEA) ; // Envio de una trama al módulo GPS
char NMEA_Received (void) ; // Comprueba la recepción de una nueva trama GPGGA
int Read_NMEA  (NMEA_frame_type NMEA_frame, char *length) ; // Lectura de la
trama GPGGA recibida
position Read_GPS (void); // Lectura de la posición
char Get_error (void); // Error en la medida de posición
```

```
position Modificate_Range(position pos); // Rango 90/-90 lat y 180/-180 long
char Is_GGA (NMEA_frame_type NMEA_frame) ; // Comprobamos si se trata de una
trama GPGGA
char Have_Position (NMEA_frame_type NMEA_frame) ; // Comprobamos si la trama
posee una posición distinta de cero
position Interpret_GGA (NMEA_frame_type NMEA_frame) ; // Obtención de las
coordenadas
Earth_float GGA_Get_Latitude (NMEA_frame_type NMEA_frame) ;  // Obtención de la
latitud en radianes
Earth_float GGA_Get_Longitude (NMEA_frame_type NMEA_frame) ; // Obtención de la
longitud en radianes
int GGA_Get_Altitude (NMEA_frame_type NMEA_frame) ;  // Obtención de la altitud
en metros


/******************************************************************************/
/*                         Exported functions                                 */
/******************************************************************************/

// Inicialización del Módulo del GPS y del SCIb
int Init_GPS (void) {

       char *SET_NMEA_OUTPUT ;

       // Inicialización puerto UART1
       SCIb_Init (0, Receive_NMEA) ;

       // Trasmisión trama GPGGA
       SET_NMEA_OUTPUT = "PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0" ;
    Send_NMEA (SET_NMEA_OUTPUT) ;

       // Frecuencia de trasmisión de 5 Hz
    SET_NMEA_OUTPUT = "PMTK220,200" ;
       Send_NMEA (SET_NMEA_OUTPUT) ;

       DelayMs (20) ;

       // Desavilitación de la velocidad del módulo
       SET_NMEA_OUTPUT = "PMTK397,0.0" ;
    Send_NMEA (SET_NMEA_OUTPUT) ;

    NMEAB_Row = 0 ;
       NMEAB_Index = 0 ;

    return 0 ;
}

// Envia una trama NMEA al módulo GNSS
int Send_NMEA (char *NMEA) {
       unsigned char CHS, i ;
       char C ;

       Send_CHAR ('$') ;
    for (i=0 ; i<100 ; i++) {
       if (NMEA[i] == '\0') break ;
       Send_CHAR (NMEA[i]) ;
    }
    if (i == 100) return -1 ;
    Send_CHAR ('*') ;
    CHS =  Calculate_CHS (NMEA) ;
    C = CHS >> 4 ;
    if (C < 10) Send_CHAR (C + 48) ;
```

```
    else Send_CHAR (C + 55) ;
    C = CHS&0x0F ;
    if (C < 10) Send_CHAR (C + 48) ;
    else Send_CHAR (C + 55) ;
    Send_CHAR ('\r') ;
    Send_CHAR ('\n') ;

        return 0 ;
}

// Devuelve si se ha recibido una trama NMEA
char NMEA_Received (void) {
        return GPS_NMEA_received ;
}

// Lectura de la trama NMEA recibida
int Read_NMEA  (NMEA_frame_type NMEA_frame, char *length) {

        unsigned char i ;

        //if (!GPS_NMEA_received) return -1 ;
        *length = 0 ;
        for (i=0; i<100; i++) {
                NMEA_frame[i] = NMEA_buffer [(NMEAB_Row+1)%2][i] ;
                if (NMEA_frame[i] == '\n') {
                        *length = i+1 ;
                        break ;
                }
        }
        GPS_NMEA_received = 0 ;
        return 0 ;
}

// Devuelve la posición obtenida de la trama GPGGA
position Read_GPS (void){

        NMEA_frame_type frame;
        char longitude;
        position pos, Aux;

        if(NMEA_Received()){
                Read_NMEA(frame, &longitude);
                Aux=Interpret_GGA(frame);
                pos=Modificate_Range(Aux);
        }

        if(!Have_Position(frame)){
                Error_GPS = 1;
        } else {
                Error_GPS = 0;
        }

        return pos;
}
```

```
// Normaliza los valores de posición
position Modificate_Range(position pos){

	position Aux;

	if(pos.Latitude>90){
		Aux.Latitude = pos.Latitude-360;
	} else if(pos.Latitude<-90){
		Aux.Latitude = pos.Latitude+360;
	} else {
		Aux.Latitude = pos.Latitude;
	}

	if(pos.Longitude>180){
		Aux.Longitude = pos.Longitude-360;
	} else if(pos.Longitude<-180){
		Aux.Longitude = pos.Longitude+360;
	} else {
		Aux.Longitude = pos.Longitude;
	}

	Aux.Altitude = pos.Altitude;

	return Aux;
}

// Devuelve si hay un error en la lectura de la trama GPGGA
char Get_error (void){
	return Error_GPS;
}

// Comprueba si es una trama GPGGA
char Is_GGA (NMEA_frame_type NMEA_frame) {
   if ((NMEA_frame[0] == '$') & (NMEA_frame[1] == 'G') & (NMEA_frame[2] == 'P') &
   (NMEA_frame[3] == 'G') & (NMEA_frame[4] == 'G') & (NMEA_frame[5] == 'A'))
return 1 ;
   else return 0 ;
}

// Comprueba si la trama recibida es correcta
char Have_Position (NMEA_frame_type NMEA_frame) {
	char result = 0 ;
	unsigned char Last_comma = 1 ;

	if (Is_GGA(NMEA_frame)) {
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
		if (NMEA_frame[Last_comma+1] != '0') result = 1 ;
	}
	return result ;
}
```

```
// Obtiene la latitud contenida en una trama GPGGA
Earth_float GGA_Get_Latitude (NMEA_frame_type NMEA_frame) {  // In radians

    Earth_float Degrees, Minutes, Minutes2;
    unsigned char Last_comma = 1 ;

    Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
    Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
    st[0] = NMEA_frame[Last_comma+1] ;
    st[1] = NMEA_frame[Last_comma+2] ;
    st[2] = '\0' ;
    Degrees = (float)atoi (st) ;
       st2[0] = NMEA_frame[Last_comma+3];
       st2[1] = NMEA_frame[Last_comma+4];
       st2[2] = '\0';
       Minutes = (float)atoi (st2) ;
       st3[0] = NMEA_frame[Last_comma+6];
       st3[1] = NMEA_frame[Last_comma+7];
       st3[2] = NMEA_frame[Last_comma+8];
       Minutes2 = (float) atoi(st3) ;

       Last_comma = Move_to_comma (Last_comma, NMEA_frame);
       if(NMEA_frame[Last_comma+1]==0x4E){ // Norte
              return (Degrees + Minutes/60.0 + Minutes2/1000.0/60.0);
       } else { // Sur
              return (-Degrees - (float)Minutes/60.0 - Minutes2/1000/60.0);
       }
}

// Obtiene la longitud contenida en una trama GPGGA
Earth_float GGA_Get_Longitude (NMEA_frame_type NMEA_frame) { // In radians

       Earth_float Degrees, Minutes, Minutes2 ;
       unsigned char Last_comma = 1 ;

       Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
       Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
       Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
       Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
       st[0] = NMEA_frame[Last_comma+1] ;
       st[1] = NMEA_frame[Last_comma+2] ;
       st[2] = NMEA_frame[Last_comma+3] ;
       st[3] = '\0' ;
       Degrees = (float)atoi (st) ;
       st2[0] = NMEA_frame[Last_comma+4];
       st2[1] = NMEA_frame[Last_comma+5];
       st2[2] = '\0';
       Minutes = (float)atoi (st2) ;
       st3[0] = NMEA_frame[Last_comma+7];
       st3[1] = NMEA_frame[Last_comma+8];
       st3[2] = NMEA_frame[Last_comma+9];
       st3[3] = '\0';
       Minutes2 = (float)atoi (st3) ;

       Last_comma = Move_to_comma (Last_comma, NMEA_frame);
       if(NMEA_frame[Last_comma+1]==0x57){
              return (-Degrees - Minutes/60.0 - Minutes2/1000.0/60.0) ;
       } else {
              return (Degrees + Minutes/60.0 + Minutes2/1000.0/60.0 ) ;
       }

}
```

```
// Obtiene la altitud contenida en una trama GPGGA
int GGA_Get_Altitude (NMEA_frame_type NMEA_frame) {  // In meters

	unsigned char Last_comma = 1 ;
	int Altitude;

	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	Last_comma = Move_to_comma (Last_comma, NMEA_frame) ;
	st[0] = NMEA_frame[Last_comma+1] ;
	st[1] = NMEA_frame[Last_comma+2] ;
	st[2] = NMEA_frame[Last_comma+3] ;
	st[3] = '\0' ;
	Altitude = atoi (st) ;

	return Altitude;
}

// Obtiene la posición contenida en una trama GPGGA
position Interpret_GGA (NMEA_frame_type NMEA_frame) {

	position ESF_position ;


	ESF_position.Latitude = GGA_Get_Latitude (NMEA_frame) ;
	ESF_position.Longitude = GGA_Get_Longitude (NMEA_frame) ;
	ESF_position.Altitude = GGA_Get_Altitude (NMEA_frame) ;

	return ESF_position ;
}

/**************************************************************************/
/*                          Local functions                               */
/**************************************************************************/

// Envía un char C por el puerto UART1
int Send_CHAR (char C) {
	SCIb_PutChar(C) ;
	return 0 ;
}

// Cálculo de las tramas de comunicación con el módulo GPS
static unsigned char Calculate_CHS (char *NMEA) {

  unsigned char CHS, i ;

  CHS = NMEA[0] ;
  for (i=1 ; i<100 ; i++) {
	if (NMEA[i] == '\0') return CHS ;
	CHS ^= NMEA[i] ;
  }
  return 0 ;
}
```

```c
// Comprueba si la trama recivida es GPGGA
void Receive_NMEA (unsigned char C) {

  static enum recogniced_NMEA_type recogniced_NMEA = NONE;

  switch (recogniced_NMEA) {
        case NONE: if (C == '$') {
                recogniced_NMEA = DATA ;
                NMEA_buffer [NMEAB_Row][NMEAB_Index] = C ; NMEAB_Index++ ;
                break ; }
        case DATA: NMEA_buffer [NMEAB_Row][NMEAB_Index] = C ; NMEAB_Index++ ;
              if (C == 0x0A) {
                    recogniced_NMEA = NONE ;
                    NMEAB_Index = 0 ;
                    GPS_NMEA_received = 1 ;
                    if(Is_GGA(NMEA_buffer [NMEAB_Row])){
                          NMEAB_Row = (NMEAB_Row + 1)%2 ;
                    }
              } else if (NMEAB_Index == 100) {
                    recogniced_NMEA = NONE ;
                    NMEAB_Index = 0 ;
              }
  }
  return ;
}

// Obtiene la posición en una trama del caracter ',' a partir de la posición i
static char Move_to_comma (char i, NMEA_frame_type F) {

   unsigned char j ;

   for (j=i+1; j < 100; j++) if (F[j] == ',') return j ;
   return 0 ;

}
```

Anexo: Fichero GPS.h

Código implementado en el fichero GPS.h:

```
#ifndef GPS_H_
#define GPS_H_

/******************************************************************************/
/*                        Typedefs and structures                             */
/******************************************************************************/

typedef char NMEA_frame_type[100] ;

typedef float Earth_float ;

// Estructura para almacenar la posición
typedef struct {
   float Longitude ;   // Grados
   float Latitude ;    // Grados
   float Altitude ;    // Metros
} position ;


/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

int Init_GPS (void); // Inicialización del GPS y del SCIb
int Send_NMEA (char *NMEA) ; // Envio de una trama al módulo GPS
char NMEA_Received (void) ; // Comprueba la recepción de una nueva trama GPGGA
int Read_NMEA  (NMEA_frame_type NMEA_frame, char *length) ; // Lectura de la
trama GPGGA recibida
position Read_GPS (void); // Lectura de la posición
char Get_error (void); // Error en la medida de posición
position Modificate_Range(position pos); // Rango 90/-90 lat y 180/-180 long
char Is_GGA (NMEA_frame_type NMEA_frame) ; // Comprobamos si se trata de una
trama GPGGA
char Have_Position (NMEA_frame_type NMEA_frame) ; // Comprobamos si la trama
posee una posición distinta de cero
position Interpret_GGA (NMEA_frame_type NMEA_frame) ; // Obtención de las
coordenadas
Earth_float GGA_Get_Latitude (NMEA_frame_type NMEA_frame) ;  // Obtención de la
latitud en radianes
Earth_float GGA_Get_Longitude (NMEA_frame_type NMEA_frame) ; // Obtención de la
longitud en radianes
int GGA_Get_Altitude (NMEA_frame_type NMEA_frame) ;  // Obtención de la altitud
en metros

#endif /* GPS_H_ */
```

Anexo: Fichero ALARMA.c

Código implementado en el fichero ALARMA.c:

```
/****************************************************************************/
/*                        Used modules                                      */
/****************************************************************************/

#include <stdlib.h>
#include <math.h>
#include <avr/io.h>
#include <compat/twi.h>
#include "CLOCK.h"
#include "I2C.h"
#include "ALARMA.h"
#include "BUFFER.h"
#include "GPS.h"

/****************************************************************************/
/*                        Local variables                                   */
/****************************************************************************/

static double aT;

position point1, point2;
float distance_1_2 = -2;
char time1=0;

int pasos = 1;

static char Alarma_Leve = 0;
static char Alarma_Golpe = 0;
static char Alarma_MedidaA = 0;
static char Alarma_Critica = 0;

static ACCELEROMETER_Data DataA;

/****************************************************************************/
/*                        Local functions prototypes                        */
/****************************************************************************/

static double total_aceleration (void); // Cálcula el módulo de la acelereción

/****************************************************************************/
/*                        Exported functions prototypes                     */
/****************************************************************************/

void Alarm (void); // Activa las alarmas
char Get_Alarma_Golpe (void); // Devuelve la activación de la alarma
char Get_Alarma_Leve (void); // Devuelve la activación de la alarma
char Get_Alarm_MedidaA (void); // Devuelve la activación de la alarma
char Get_Alarma_Critica (void); // Devuelve la activación de la alarma
void Reset_Alarma_Golpe (void); // Puesta a cero de la alarma
void Reset_Alarma_Critica (void); // Puesta a cero de la alarma
```

```
/******************************************************************************/
/*                        Local functions                                     */
/******************************************************************************/

static double total_aceleration (void){
       double total=0.0;

       Get_Measure_Accelerometer();
       DataA = Get_Data_Accelerometer();
       total = sqrt(DataA.X*DataA.X+DataA.Y*DataA.Y+DataA.Z*DataA.Z);

       return total;
}


/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

// Activa las alarmas
void Alarm (void){

       // Obtenemos la aceleración total
       aT = total_aceleration();

       // Comprobamos si hay un fallo en la recepción de los datos del
acelerometro
       if(!Received_Measure_Accelerometer()){
              Alarma_MedidaA = 1;
       } else {
              Alarma_MedidaA = 0;

              if((aT>12.5) && (Alarma_Golpe != 1)){
                     Alarma_Golpe = 1;
                     Remove_Timer();
                     Set_Timer(30000); //ms
              }

              if((aT<10.5)&&(aT>8.5)){
                     if(pasos == 1){
                            Set_Timer2(5000);
                            pasos = 0;
                     }
              } else {
                     Alarma_Leve = 0;
                     pasos = 1;
                     Remove_Timer2();
              }
       }

       if (Time_Out()){

              if(time1 ==0){
                     point1 = Read_GPS();
                     if(Get_error()){
                            time1 = 2;
                     } else {
                            time1 = 1;
                     }
                     Remove_Timer();
                     Set_Timer(30000);
```

```
			} else {
				point2 = Read_GPS();
				if ((Get_error())||(time1 ==2)){
					distance_1_2 = -1;
				} else {
					distance_1_2 = Distance(point2, point1);
				}
				if(((distance_1_2>-1)&&(distance_1_2<1))||(Alarma_Leve ==
1)){
					Alarma_Golpe = 1;
				} else {
					Alarma_Golpe = 0;
				}
				if((Alarma_Leve == 1)){
					Alarma_Critica = 1;
				}
				Remove_Timer();
			}
		}

		if (Time_Out2()){
			Alarma_Leve = 1;
			Remove_Timer2();
		}



}

// Devuelve la activación de la alarma
char Get_Alarma_Golpe (void){
	return Alarma_Golpe;
}

// Devuelve la activación de la alarma
char Get_Alarma_Leve (void){
	return Alarma_Leve;
}

// Devuelve la activación de la alarma
char Get_Alarm_MeasureA (void){
	return Alarma_MedidaA;
}

// Devuelve la activación de la alarma
char Get_Alarma_Critica (void){
	return Alarma_Critica;
}

// Puesta a cero de la alarma
void Reset_Alarma_Golpe (void){
	Alarma_Golpe = 0;
}

// Puesta a cero de la alarma
void Reset_Alarma_Critica (void){
	Alarma_Critica = 0;
}
```

Anexo: Fichero ALARMA.h

Código implementado en el fichero ALARMA.h:

```
#ifndef ALARMA_H_
#define ALARMA_H_

/******************************************************************************/
/*                        Used modules                                        */
/******************************************************************************/

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

void Alarm (void); // Activa las alarmas
char Get_Alarma_Golpe (void); // Devuelve la activación de la alarma
char Get_Alarma_Leve (void); // Devuelve la activación de la alarma
char Get_Alarm_MeasureA (void); // Devuelve la activación de la alarma
char Get_Alarma_Critica (void); // Devuelve la activación de la alarma
void Reset_Alarma_Golpe (void); // Puesta a cero de la alarma
void Reset_Alarma_Critica (void); // Puesta a cero de la alarma


#endif /* ALARMA_H_ */
```

Anexo: Fichero BRUJULA.c

Código implementado en el fichero BRUJULA.c:

```
/*****************************************************************************/
/*                         Used modules                                      */
/*****************************************************************************/

#include <stdlib.h>
#include <math.h>
#include <avr/io.h>
#include <compat/twi.h>
#include "BRUJULA.h"
#include "XBEE.h"
#include "I2C.h"

/*****************************************************************************/
/*                         Local variables                                   */
/*****************************************************************************/

static float orientation;

static char Alarm_MeasureG = 0;
static char Alarm_MeasureM = 0;
static char Alarm_MeasureO = 0;

static GYROSCOPE_Data DataG;
static MAGNETOMETER_Data DataM;

/*****************************************************************************/
/*                         Local functions prototypes                        */
/*****************************************************************************/

static float norm (float X); // Normacilización de un ángulo X

/*****************************************************************************/
/*                         Exported functions prototypes                     */
/*****************************************************************************/

void Calculate_orientation (void); // Cálculo de la orientación
float Get_Orientation (void); // Devuelve la orientación
char Get_Error_MeasureG (void); // Devuelve un posible error
char Get_Error_MeasureM (void); // Devuelve un posible error
char Get_Error_MeasureB (void); // Devuelve un posible error

/*****************************************************************************/
/*                         Local functions                                   */
/*****************************************************************************/

// Normacilización de un ángulo X
static float norm (float X){
	float aux=X;

	if (X<0){
		aux = X + 360;
	} else if(X>360){
		aux = X - 360;
	}

	return aux;
}
```

```
/******************************************************************************/
/*                         Exported functions                                 */
/******************************************************************************/

// Cáculo de la orientación
void Calculate_orientation (void){

       float Modulo;

       Alarm_MeasureG = 0;
       Alarm_MeasureM = 0;
       Alarm_MeasureO = 0;

       Get_Measure_Magnetometer();
       Get_Measure_Gyroscope();

       if(!Received_Measure_Gyroscope()) Alarm_MeasureG = 1;
       if(!Received_Measure_Magnetometer()) Alarm_MeasureM = 1;

       if(Alarm_MeasureM || Alarm_MeasureG){
               Alarm_MeasureO = 1;
       }

       if(!(Alarm_MeasureM) || !(Alarm_MeasureG)){

              DataG = Get_Data_Gyroscope();
              DataM = Get_Data_Magnetometer();

              Modulo = sqrt(DataM.X*DataM.X+DataM.Y*DataM.Y);

              if ((DataG.X > 15) || (DataG.Y > 15) || (DataG.Z > 15) || (Modulo < 
0.0)){
                     Alarm_MeasureO = 1;
              } else {
                     orientation = norm((float)(atan2(DataM.Y/0.25, 
DataM.X/0.25))*180.0/3.1415);
                     orientation = norm(orientation+90.0);
              }
       }
}

// Devuelve la orientación
float Get_Orientation (void){
       return orientation;
}

// Devuelve un posible error
char Get_Error_MeasureG (void){
       return Alarm_MeasureG;
}

// Devuelve un posible error
char Get_Error_MeasureM (void){
       return Alarm_MeasureM;
}

// Devuelve un posible error
char Get_Error_MeasureB (void){
       return Alarm_MeasureO;
}
```

Anexo: Fichero BRUJULA.h

Código implementado en el fichero BRUJULA.h:

```
#ifndef BRUJULA_H_
#define BRUJULA_H_

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

void Calculate_orientation (void); // Cálculo de la orientación
float Get_Orientation (void); // Devuelve la orientación
char Get_Error_MeasureG (void); // Devuelve un posible error
char Get_Error_MeasureM (void); // Devuelve un posible error
char Get_Error_MeasureB (void); // Devuelve un posible error

#endif /* BRUJULA_H_ */
```

Anexo: Ficher XBEE.c

Código implementado en el fichero XBEE.c:

```
/*************************************************************************/
/*                          Used modules                                 */
/*************************************************************************/

#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "XBEE.h"
#include "SCI.h"
#include "GPS.h"
#include "BUFFER.h"
#include "GPS.h"
#include "BLUETOOTH.h"

/*************************************************************************/
/*                          Local variables                              */
/*************************************************************************/

static char RX_Buffer [180] ;
  // Circular receiver buffer of two frames
static unsigned char RXB_Index, RXB_Lenght ;
static char receiving_frame, RXB_Frame_Received ;

static char TX_Buffer [180] ;
  // Transmit buffer
static unsigned char TXB_Index, TXB_Length ;
static unsigned char TXB_Transmited = 1 ;

static enum {SINGLE, CONTINOUS} rx_mode ;
static char rx_end ;
static char B ;
static char Begin_Frame, End_Frame ;

/*************************************************************************/
/*                          Local functions prototype                    */
/*************************************************************************/

static void Get_Char (unsigned char C); // Función que almacena un char de la
trama recibida.
static void TXINT_Action (void); // Envia un char de la trama a trasmitir.
void DelayMs(volatile unsigned int Msec); // Función que realiza una espera con
unidad en ms.
static int Set_Command_Mode (void); // Activa la comunicación por comando con el
XBee
static int AT_comand (char comand[], char response[]); // Envia un comando y
espera la respuesta
static int Exit_Command_Mode (void); // Desactiva la comunicación por comandos
con el XBee
static int Get_RSSI_Sw (void) ; // Preguntar


/*************************************************************************/
/*                    Exported functions prototype                       */
/*************************************************************************/

int Init_XBee (void); // Inicializa el XBee
```

```
int Send_BYTE (char B); // Envia un char al XBee
char Receive_BYTE (void); // Recibe un char del XBee
int Send_Frame (unsigned char *data, char lenght); // Envia una trama al XBee
char Sent_Frame (void); // Comprueba que se ha enviado la trama
void Receive_Frames (char BF); // Recibe una trama del XBee
void Stop_Receive_Frames (void); // Detiene la recepción de tramas
int Read_Frame (char *data, char length); // Almacena el valor del buffer de
comunicación
char Received_Frame (void); // Comprueba si se ha recibido la trama
Information_XBee Generate_information_XBee (char num_Ident, char ErrorPos, char
AlarmaC, char AlarmaP, char AlarmaG, char AlarmaM, position pos); // Genera la
trama de la red
void Send_information (Information_XBee Inf); // Tramisión de la trama con
información en float e int
void Send_information_ASCII (Information_XBee Inf); // Trasmisión de la trama con
nomenclatura ASCII
unsigned char change_ASCII (char num); // Trasforma num en ASCII

/*****************************************************************************/
/*                          Local functions                                  */
/*****************************************************************************/

// Función que almacena un char de la trama recivida
static void Get_Char (unsigned char C) {
	if (rx_mode == SINGLE){ // Modo de operación SINGLE, un único char (usado
para la inicialización)
		B = C ;
		  rx_end = 1 ;
	}
	else {
		if (receiving_frame){ // Modo de operación continuo y hemos
recibido frame (usado para la comunicación)
			RX_Buffer [RXB_Index] = C ; // Almacenamos el char en el
buffer
			if (RXB_Index == (RXB_Lenght-1)) { // Comprobamos si nos
encontramos en el final de la trama
				// Indicamos que no recibimos el frame y apuntamos a
la otra posición del buffer.
				receiving_frame = 0 ;
				RXB_Frame_Received = 1 ;
				Save_Information(RX_Buffer);
			}
			RXB_Index ++ ;
			if (RXB_Index == 180) { // Comprobamos si la trama es
demasiado larga
				receiving_frame = 0 ;
			}
		}
		else if (C == Begin_Frame) { // Modo de operación continuo y no
hemos recibido un frame
			RXB_Index = 0 ;
			RX_Buffer [RXB_Index] = C ; // Almacena la primera posición
del buffer actual el comienzo del frame
			receiving_frame = 1 ; // Indicamos que hemos recibido un
frame
			RXB_Index ++ ;
		}
	}
  return ;
}
```

```
// Función enviar un char de la trama enviada
static void TXINT_Action (void) {
	SCIa_PutChar (TX_Buffer[TXB_Index]) ; // Enviamos el char por el SCIa
(USART0)
	TXB_Index ++ ;
	if (TXB_Index > (TXB_Length-1)) { // Comprobamos que llegamos al final de
la trama a enviar
		SCIa_TXINT (DISABLED) ; // Deshabilitamos las interrupciones
		TXB_Transmited = 1 ; // Indicamos que ya hemos enviado la trama
	}
}

// Función de espera n ms (Utilizado en la inicialización para esperar las
respuestas del Xbee)
void DelayMs(volatile unsigned int Msec)
{
	int i = 0 ;

	while( Msec-- )                                // 1ms loop at 16MHz CPUCLK
o 2ms loop at 8MHz CPUCLK
	{
		for (i=0; i<8000; i++)     asm(" nop ");
	}

} // end DelayMs()

// Función que active la comunicación por comandos
static int Set_Command_Mode (void) {
	DelayMs (2) ;
	Send_BYTE ('+') ;
	Send_BYTE ('+') ;
	Send_BYTE ('+') ;
	DelayMs (2) ;
	if (Receive_BYTE() != 'O') return -1 ;
	if (Receive_BYTE() != 'K') return -1 ;
	if (Receive_BYTE() != '\r') return -1 ;
	return 0 ;
}

// Función que envia un comando y obtiene la respuesta
static int AT_comand (char comand[], char response[]) {

	unsigned char i ;

	for (i=0; i<80; i++) {
		Send_BYTE (comand[i]) ;
		if (comand[i] == '\r') break ;
	}
	for (i=0; i<80; i++) {
		response [i] = Receive_BYTE () ;
		if (response [i] == '\r') break ;
	}
	if (response[i] != '\r') return -1 ;
	return 0 ;
}
```

```
// Finalización de la comunicación por comandos con el XBee
static int Exit_Command_Mode (void) {
	Send_BYTE ('A') ;
	Send_BYTE ('T') ;
	Send_BYTE ('C') ;
	Send_BYTE ('N') ;
	Send_BYTE (' ') ;
	Send_BYTE ('\r') ;
	if (Receive_BYTE() != 'O') return -1 ;
	if (Receive_BYTE() != 'K') return -1 ;
	if (Receive_BYTE() != '\r') return -1 ;
	return 0 ;
}

// Preguntar
static int Get_RSSI_Sw (void) {
	char st[6] = {0,0,0,0,0,0} ;

	Set_Command_Mode () ;
	if (AT_comand ("ATDB\r", st) < 0) return -1 ;
	Exit_Command_Mode () ;
	return atoi (st) ;
}



/*****************************************************************************/
/*                         Exported functions                                */
/*****************************************************************************/

// Inicialización del XBee.
int Init_XBee (void) {

	char st[8] ;

	SCIa_Init (TXINT_Action, Get_Char) ; // Inicializamos el SCIa (USART0)

    DelayMs (550) ; // Esperamos mas de un segundo sin enviar datos al XBee
	//Activamos el modo de comunicación por comandos
    Send_BYTE ('+') ;
    Send_BYTE ('+') ;
    Send_BYTE ('+') ;

	// Esperamos hasta obtener la respuesta de XBee
    if (Receive_BYTE() != 'O') return -1 ;
    if (Receive_BYTE() != 'K') return -1 ;
    if (Receive_BYTE() != '\r') return -1 ;

    AT_comand ("ATGT 2\r", st) ;                       // Guard Times = 2 ms
    AT_comand ("ATRO 0\r", st) ;                       // Packetization Timeout.
RO=0 to transmit characters immediately
    AT_comand ("ATP0 1\r", st) ;                       // RSSI PWM enabled
    AT_comand ("ATRP FF\r", st) ;                      // PWM output always on
    Exit_Command_Mode () ; // Finalización del modo de comunicación por comandos
para la inicialización.

    return 0 ;
}
```

```c
// Envia un char a través del RX0 del SCIa (USART0)
int Send_BYTE (char B) {
	SCIa_PutChar(B) ;

	return 0 ;
}

// Espera a la recepción de un BYTE
char Receive_BYTE (void) {
	rx_mode = SINGLE ;
	rx_end = 0 ;
	while (!rx_end) ;
	return B ;
}

// Enviamos la trama
int Send_Frame (unsigned char *data, char lenght) {

	TXB_Length = lenght ;
	//memcpy (TX_Buffer, data, TXB_Length) ;
	for(int i = 0; i<lenght ; i++){
		TX_Buffer[i]=data[i];
	}
	TXB_Index = 1 ;
	TXB_Transmited = 0 ;
	Send_BYTE(TX_Buffer[0]);
	SCIa_TXINT (ENABLED) ;

	DelayMs (1) ;
	return 0 ;
}

// Comprueba si hemos enviado la trama
char Sent_Frame (void) {
	return TXB_Transmited ;
}

// Recibe la trama indicandole el inicio y final de esta.
void Receive_Frames (char BF) {
	Begin_Frame = BF;
	RXB_Lenght = 13;
	rx_mode = CONTINOUS ;
      // BF = character that starts the frame
      // EF = character that ends the frame
}

// Interrumpe la recepción de una trama.
void Stop_Receive_Frames (void) {
	rx_mode = SINGLE ;
}

// Lee una trama recibida y la almacena en el buffer
int Read_Frame (char *data, char length) {

	unsigned char i ;

	if (!RXB_Frame_Received) return -1 ;
	for (i=0; i<80; i++) {
		data[i] = RX_Buffer [i] ;
		if (data[i] == End_Frame) break ;
	}
```

```
        length = i+1 ;
        RXB_Frame_Received = 0 ;
        return 0 ;
}

// Comprobamos si hemos recibido una trama
char Received_Frame (void) {
    // TRUE if a new frame has been received
        return RXB_Frame_Received ;
}

// Genera la información de la trama enviada
Information_XBee Generate_information_XBee (char num_Ident, char ErrorPos, char
AlarmaC, char AlarmaP, char AlarmaG, char AlarmaM, position pos){
        Information_XBee Data;
        char Aux=0;

        Aux = Aux | (num_Ident&(0b00000111));
        Aux = Aux | ((ErrorPos<<3)&(0b00001000));
        Aux = Aux | ((AlarmaC<<4)&(0b00010000));
        Aux = Aux | ((AlarmaP<<5)&(0b00100000));
        Aux = Aux | ((AlarmaG<<6)&(0b01000000));
        Aux = Aux | ((AlarmaM<<7)&(0b10000000));

        Data.Identification_Error_Alarm=Aux;
        Data.Longitude.number = pos.Longitude;
        Data.Latitude.number = pos.Latitude;
        Data.Altitude = pos.Altitude;

        return Data;
}

// Envia la trama con la información en formato float e int
void Send_information (Information_XBee Inf){

        // Información: $XB NºIdent,Error,Alarma(1) Lat(4) Long(4) Alt(4)

        unsigned char buffer[19];

        buffer[0]='$';
        buffer[1]='X';
        buffer[2]='B';

        buffer[3]=Inf.Identification_Error_Alarm;

        buffer[4]=Inf.Longitude.bytes[3];
        buffer[5]=Inf.Longitude.bytes[2];
        buffer[6]=Inf.Longitude.bytes[1];
        buffer[7]=Inf.Longitude.bytes[0];

        buffer[8]=Inf.Latitude.bytes[3];
        buffer[9]=Inf.Latitude.bytes[2];
        buffer[10]=Inf.Latitude.bytes[1];
        buffer[11]=Inf.Latitude.bytes[0];

        buffer[12]=Inf.Altitude>>8;
        buffer[13]=Inf.Altitude;


        Send_Frame(buffer, 14);
}
```

```c
// Envia la trama en nomenclatura ASCII
void Send_information_ASCII (Information_XBee Inf){
	unsigned char buffer[100];
	char leght = 35;
	char Aux = 0;
	int Aux2 = 0;
	float Aux3 = 0.0;

	buffer[0]='$';
	buffer[1]='X';
	buffer[2]='B';
	buffer[3]=',';

	Aux = Inf.Identification_Error_Alarm & 0b00000111;
	buffer[4]=change_ASCII(Aux);
	buffer[5]=',';

	if (Inf.Identification_Error_Alarm & 0b00001000){
		buffer[6]='1';
	} else {
		buffer[6]='0';
	}
	buffer[7]=',';

	if (Inf.Identification_Error_Alarm & 0b00010000){
		buffer[8]='1';
		} else {
		buffer[8]='0';
	}
	buffer[9]=',';

	if (Inf.Identification_Error_Alarm & 0b00100000){
		buffer[10]='1';
		} else {
		buffer[10]='0';
	}
	buffer[11]=',';

	Aux = Inf.Identification_Error_Alarm & 0b00000111;
	if (Inf.Identification_Error_Alarm & 0b01000000){
		buffer[12]='1';
		} else {
		buffer[12]='0';
	}
	buffer[13]=',';

	Aux = Inf.Identification_Error_Alarm & 0b00000111;
	if (Inf.Identification_Error_Alarm & 0b10000000){
		buffer[14]='1';
		} else {
		buffer[14]='0';
	}
	buffer[15]=',';

	if(Inf.Longitude.number>=0){
		buffer[16]='+';
		Aux3 = Inf.Longitude.number;
	} else {
		buffer[16]='-';
		Aux3 = -Inf.Longitude.number;
	}
```

```
Aux2 = (int)Aux3;
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[17]=change_ASCII(Aux);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[18]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[19]=change_ASCII(Aux);

buffer[20]='.';

Aux2 = ((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[21]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[22]=change_ASCII(Aux);

buffer[23]=',';

if(Inf.Latitude.number>=0){
       buffer[24]='+';
       Aux3 = Inf.Latitude.number;
       } else {
       buffer[24]='-';
       Aux3 = -Inf.Latitude.number;
}
Aux2 = (int)Aux3;
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[25]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[26]=change_ASCII(Aux);

buffer[27]='.';

Aux2=((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[28]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[29]=change_ASCII(Aux);

buffer[30]=',';

if(Inf.Altitude>=0){
       buffer[31]='+';
       Aux2 = Inf.Altitude;
       } else {
       buffer[31]='-';
       Aux2 = -Inf.Altitude;
}
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[32]=change_ASCII(Aux);
```

```
        Aux = Aux2/10;
        Aux2 = Aux2%10;
        buffer[33]=change_ASCII(Aux);
        Aux = Aux2/1;
        Aux2 = Aux2%1;
        buffer[34]=change_ASCII(Aux);

        Send_Frame(buffer, leght);

}

// Trasforma a ASCII
unsigned char change_ASCII (char num){
        unsigned char ASCII='0';

        if(num == 1){
                ASCII = '1';
        }
        if(num == 2){
                ASCII = '2';
        }
        if(num == 3){
                ASCII = '3';
        }
        if(num == 4){
                ASCII = '4';
        }
        if(num == 5){
                ASCII = '5';
        }
        if(num == 6){
                ASCII = '6';
        }
        if(num == 7){
                ASCII = '7';
        }
        if(num == 8){
                ASCII = '8';
        }
        if(num == 9){
                ASCII = '9';
        }

        return ASCII;
}
```

Anexo: Fichero XBEE.h

Código implementado en el fichero XBEE.h:

```
#ifndef XBEE_H_
#define XBEE_H_

#include "GPS.h"

/******************************************************************************/
/*                         Typedefs and structures                            */
/******************************************************************************/

typedef union{
	float number;
	unsigned char bytes[4];
} Byte_Float;

typedef struct {
	char Identification_Error_Alarm ; // (0123:identificador, 4:Error GPS,
5:Alarma, 67:Reservado)
	Byte_Float Longitude ;   // en grados
	Byte_Float Latitude ;    // en grados
	int Altitude ;    // en metros
} Information_XBee ;

/******************************************************************************/
/*                         Exported functions                                 */
/******************************************************************************/

   int Init_XBee (void); // Inicializa el XBee
   int Send_BYTE (char B); // Envia un char al XBee
   char Receive_BYTE (void); // Recibe un char del XBee
   int Send_Frame (unsigned char *data, char lenght); // Envia una trama al XBee
   char Sent_Frame (void); // Comprueba que se ha enviado la trama
   void Receive_Frames (char BF); // Recibe una trama del XBee
   void Stop_Receive_Frames (void); // Detiene la recepción de tramas
   int Read_Frame (char *data, char length); // Almacena el valor del buffer de
comunicación
   char Received_Frame (void); // Comprueba si se ha recibido la trama
   Information_XBee Generate_information_XBee (char num_Ident, char ErrorPos,
char AlarmaC, char AlarmaP, char AlarmaG, char AlarmaM, position pos); // Genera
la trama de la red
   void Send_information (Information_XBee Inf); // Tramisión de la trama con
información en float e int
   void Send_information_ASCII (Information_XBee Inf); // Trasmisión de la trama
con nomenclatura ASCII
   unsigned char change_ASCII (char num); // Trasforma num en ASCII

#endif /* XBEE_H_ */
```

Anexo: Fichero BLUETOOTH.c

Código implementado en el fichero BLUETOOTH.c:

```
/******************************************************************************/
/*                        Used modules                                        */
/******************************************************************************/

#include <string.h>
#include <stdlib.h>

#include "BLUETOOTH.h"
#include "SPI.h"
#include "BRUJULA.h"
#include "GPS.h"
#include "BUFFER.h"
#include "XBEE.h"

/******************************************************************************/
/*                        Local variables                                     */
/******************************************************************************/

static char SPIR_Buffer [2][80]; // Almacenamos las tramas recibida a traves del
SPI
static char SPIR_Index = 0, SPIR_Row = 0, SPIR_Length = 0; // Nos desplazamos a
traves del buffer de recepción

static char SPIT_Buffer [80]; // Almacenamos la trama que trasmitimos a traves
del SPI
static char SPIT_Index = 0, SPIT_Length = 0; // Nos desplazamos a traves del
buffer de trasmisión

static char byte; // Almacenamos el byte recibido a traves del SPI en modo SINGLE

static enum {SINGLE, CONTINUOS} trx_mode; // Describe el modo de trasmisión (byte
/ trama)

static char end_trasmision=1; // Indica la finalización de la trasmisión de la
trama
static char end_reception = 1, first_received = 0; // Indica la finalización de
la recepción
static char end_byte=0; // Indica la finalización de la trasmisión del byte

Bluetooth_mode mode = ACTIVATE5;

/******************************************************************************/
/*                        Local functions prototype                           */
/******************************************************************************/

static void Get_Char(unsigned char C); // Función que almacena un byte individual
o un byte de una trama
static void Send_Char(unsigned char C); // Función que envia un byte individual
correspondiente a una trama

/******************************************************************************/
/*                    Exported functions prototype                            */
/******************************************************************************/

int Init_Bluetooth (void); // Inicializamos la comunicación bluetooth
int Send_byte (char B); // Enviamos un byte
int Send_frame (char *data, char length); // Enviamos una trama
```

```
char Send_Receive_byte (void); // Comprobamos si ha finalizado la trasmisión
char Sent_frame (void); // Comprobamos si ha finalizado la trasmisión
int Receive_frame (char BI, char length); // Lectura de una trama
char Received_frame (void); // Comprobamos si ha finalizado la recepción
int Get_Frame (char *data); // Almacenamos la trama recibida fuera del fichero
char Get_Byte (void); // Almacenamos el byte recibido fuera del fichero
Information_Bluetooth Generate_information_Bluetooth (char num_Ident, char
ErrorPos, char alarma1, char alarma2, char alarma3, char alarma4, char  ErrorG,
char ErrorM, char ErrorB, position pos, float orient); // Gneración de la trama
enviada
void Send_information_Bluetooth (Information_Bluetooth inf); // Envio de la trama
en formato float e int
void Change_Mode (char *data); // cambio de modo
Bluetooth_mode Get_Mode (void); // devuelve el modo de funcionamiento
void Send_information_Bluetooth_ASCII (Information_Bluetooth inf); // Envio de la
trama en nomenclatura ASCII

/******************************************************************************/
/*                          Local functions                                   */
/******************************************************************************/

// Almacenamos el byte individual o un byte de una trama
static void Get_Char (unsigned char C)
{
	if (trx_mode == SINGLE){
		byte = C;
		end_byte = 1;
	} else if(!end_reception){
		if (C == '&') first_received = 1;
		if (first_received){
			if(SPIT_Index != (SPIT_Length)){
				SPIR_Buffer[SPIR_Row][SPIR_Index] = C;
				SPIR_Index ++;
			} else {
				Change_Mode(SPIR_Buffer[SPIR_Row]);
				SPIR_Index = 0;
				SPIR_Row = (SPIR_Row+1)%2;
				end_reception = 1;
				first_received = 0;
			}
			if(end_trasmision){
				SPI_MasterTransmit (0);
			}
		} else {
			end_reception = 1;
			first_received = 0;
		}
	}
}
// Enviamos el siguiente byte de la trama a no ser que haya finalizado
static void Send_Char (unsigned char C)
{
	if (!end_trasmision && trx_mode==CONTINUOS && SPIT_Length>SPIT_Index){
		if(end_reception){
			if(C == (0x16)){
				end_trasmision=0;
				first_received = 1;
				SPIR_Buffer [SPIR_Row][SPIR_Index] = C;
				SPIT_Length = 3;
				SPIR_Index++;
			}
		}
```

```
		SPI_MasterTransmit (SPIT_Buffer[SPIT_Index]);
		SPIT_Index ++;
	} else {
		end_trasmision = 1;
		SPIT_Index = 0;
	}
}

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

// Inicialización de la comunicación bluetooth
int Init_Bluetooth (void)
{
	SPI_MasterInit(Get_Char, Send_Char);

	return 0;
}

// ENviamos un byte a través del SPI
int Send_byte (char B)
{
	end_byte = 0;
	trx_mode = SINGLE;
	SPI_MasterTransmit(B);


	return 0;
}

// Enviamos una trama a través del SPI
int Send_frame (char *data, char length)
{
	memcpy (SPIT_Buffer, data, length);
	SPIT_Length = length;
	end_trasmision = 0;

	if(end_reception==1){
		SPIT_Index = 1;
		trx_mode = CONTINUOS;
		SPI_MasterTransmit(data[0]);
	} else {
		SPIT_Index = 0;
	}
	return 0;
}

int Receive_frame (char BI, char length){

	SPIR_Length = length;
	end_reception =0;

	if(end_trasmision==1){
		trx_mode = CONTINUOS;
		SPI_MasterTransmit(0);
		SPIR_Index=0;
	}

	return 1;
}
```

```
// Comprobamos si ya ha sido trasmitido el byte.
char Send_Receive_byte (void)
{
	return end_byte;
}

// Comprobamos si la trama ya ha sido trasmitida
char Sent_frame (void)
{
	return end_trasmision;
}

char Received_frame (void)
{
	return end_reception;
}

// Almacenamos fuera del fichero la trama recibida
int Get_Frame (char *data)
{
	unsigned char i;

	if (!end_trasmision) return -1;
	for (i=0; i<80; i++){
		data[i] = SPIR_Buffer [(SPIR_Row+1)%2][i];
		if (i == SPIT_Length) break;
	}
	 end_trasmision = 0;
	 return 0;
}

// Almacenamos fuera del fichero el byte recibido
char Get_Byte (void)
{
	if(end_byte == 1){
		end_byte = 0;
		return byte;
	} else {
		return 10; // Modificar para avisar de que no se a recibido nada
	}
}

// Genera la trama enviada
Information_Bluetooth Generate_information_Bluetooth (char num_Ident, char
ErrorPos,char alarma1, char alarma2, char alarma3, char alarma4, char  ErrorG,
char ErrorM, char ErrorB, position pos, float orient){
	Information_Bluetooth inf;
	Informacion_Bluetooth_Alarma AuxPCBx;
	char Aux=0;
	char numErrores=0;

	Aux = Aux | (num_Ident&(0b00000111));
	Aux = Aux | ((ErrorPos<<3)&(0b00001000));
	if((alarma1==1)||(alarma2==1)||(alarma3==1)||(alarma4==1)){
		Aux = Aux | (0b00010000);
	} else {
		Aux = Aux | (0b00000000);
	}
	Aux = Aux | ((ErrorG<<5)&(0b00100000));
	Aux = Aux | ((ErrorM<<6)&(0b01000000));
	Aux = Aux | ((ErrorB<<7)&(0b10000000));
```

```
	inf.Identification_Error = Aux;
	inf.Longitude.number = pos.Longitude;
	inf.Latitude.number = pos.Latitude;
	inf.orientation.number = orient;
	inf.Altitude = pos.Altitude;

	for(char i=0; i<10; i++){
		AuxPCBx = Get_Information_PCBx(i);

		if((AuxPCBx.Identification_Alarm&0b10000000) ||
(AuxPCBx.Identification_Alarm&0b01000000) ||
(AuxPCBx.Identification_Alarm&0b00100000) ||
(AuxPCBx.Identification_Alarm&0b00010000) ||
(AuxPCBx.Identification_Alarm&0b00001000)){
			inf.Alarmas[numErrores]=AuxPCBx;
			numErrores ++;
		}
	}

	inf.numberE = numErrores;

	return inf;
}

// Envia la trama en formato float e int
void Send_information_Bluetooth (Information_Bluetooth inf){

	int i=0;
	int j=0;
	char buffer[15+10*13];

	buffer[0]='&';//0x16;
	buffer[1]='B';
	buffer[2]='T';

	buffer[3]=inf.Identification_Error;

	buffer[4]=inf.Longitude.bytes[3];
	buffer[5]=inf.Longitude.bytes[2];
	buffer[6]=inf.Longitude.bytes[1];
	buffer[7]=inf.Longitude.bytes[0];

	buffer[8]=inf.Latitude.bytes[3];
	buffer[9]=inf.Latitude.bytes[2];
	buffer[10]=inf.Latitude.bytes[1];
	buffer[11]=inf.Latitude.bytes[0];

	buffer[12]=inf.Altitude>>8;
	buffer[13]=inf.Altitude;

	buffer[14]=inf.numberE;

	j=17;
	if (inf.numberE!=0){
		for(i=0; i<inf.numberE; i++){
			buffer[i+j]=inf.Alarmas[i].Identification_Alarm;
			j++;

			buffer[i+j]=inf.Alarmas[i].distance.bytes[3];
			j++;
			buffer[i+j]=inf.Alarmas[i].distance.bytes[2];
			j++;
```

```
				buffer[i+j]=inf.Alarmas[i].distance.bytes[1];
				j++;
				buffer[i+j]=inf.Alarmas[i].distance.bytes[0];
				j++;

				buffer[i+j]=inf.Alarmas[i].direction.bytes[3];
				j++;
				buffer[i+j]=inf.Alarmas[i].direction.bytes[2];
				j++;
				buffer[i+j]=inf.Alarmas[i].direction.bytes[1];
				j++;
				buffer[i+j]=inf.Alarmas[i].direction.bytes[0];
				j++;

				buffer[i+j]=inf.Alarmas[i].direction_R_L;
				j++;

				buffer[i+j]=inf.Alarmas[i].Difference_Altitude>>8;
				j++;
				buffer[i+j]=inf.Alarmas[i].Difference_Altitude;
				j++;

				buffer[i+j]=inf.Alarmas[i].direction_up_down;
				j++;
			}
		}
		Send_frame(buffer, 15+15*inf.numberE);
}

// Envia la trama en nomenclatura ASCII
void Send_information_Bluetooth_ASCII (Information_Bluetooth inf){

		unsigned char buffer[500];
		char leght = 40+inf.numberE*39;
		char Aux = 0;
		int Aux2 = 0;
		float Aux3 = 0.0;
		int i=0;
		int j=0;

		buffer[0]='&';
		buffer[2]='B';
		buffer[3]='T';
		buffer[4]=',';

		Aux = inf.Identification_Error & 0b00000111;
		buffer[5]=change_ASCII(Aux);
		buffer[6]=',';

		if (inf.Identification_Error & 0b00001000){
			buffer[7]='1';
			} else {
			buffer[7]='0';
		}
		buffer[8]=',';

		if (inf.Identification_Error & 0b00010000){
			buffer[9]='1';
			} else {
			buffer[9]='0';
		}
		buffer[10]=',';
```

```
if (inf.Identification_Error & 0b00100000){
        buffer[11]='1';
        } else {
        buffer[11]='0';
}
buffer[12]=',';

if (inf.Identification_Error & 0b01000000){
        buffer[13]='1';
        } else {
        buffer[13]='0';
}
buffer[14]=',';

if (inf.Identification_Error & 0b10000000){
        buffer[15]='1';
        } else {
        buffer[15]='0';
}
buffer[16]=',';

if(inf.Longitude.number>=0){
        buffer[17]='+';
        Aux3 = inf.Longitude.number;
        } else {
        buffer[17]='-';
        Aux3 = -inf.Longitude.number;
}
Aux2 = (int)Aux3;
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[18]=change_ASCII(Aux);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[19]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[20]=change_ASCII(Aux);

buffer[21]='.';

Aux2 = ((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[22]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[23]=change_ASCII(Aux);

buffer[24]=',';

if(inf.Latitude.number>=0){
        buffer[25]='+';
        Aux3 = inf.Latitude.number;
        } else {
        buffer[25]='-';
        Aux3 = -inf.Latitude.number;
}
Aux2 = (int)Aux3;
Aux = Aux2/10;
Aux2 = Aux2%10;
```

```
buffer[26]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[27]=change_ASCII(Aux);

buffer[28]='.';

Aux2=((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[29]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[30]=change_ASCII(Aux);

buffer[31]=',';

if(inf.Altitude>=0){
       buffer[32]='+';
       Aux2 = inf.Altitude;
       } else {
       buffer[32]='-';
       Aux2 = -inf.Altitude;
}
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[33]=change_ASCII(Aux);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[34]=change_ASCII(Aux);
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[35]=change_ASCII(Aux);

buffer[36]=',';
buffer[37]= change_ASCII(inf.numberE);

j=38;
if (inf.numberE!=0){
       for(i=0; i<inf.numberE; i++){

              buffer[i+j]=',';
              j++;

              buffer[j+i]=11;
              j++;
              buffer[j+i]=13;
              j++;

              Aux = inf.Alarmas[i].Identification_Alarm & 0b00000111;
              buffer[j+i]=change_ASCII(Aux);
              j++;
              buffer[j+i]=',';
              j++;

              if (inf.Alarmas[i].Identification_Alarm & 0b00001000){
                     buffer[j+i]='1';
                     } else {
                     buffer[j+i]='0';
              }
              j++;
              buffer[j+i]=',';
```

```
j++;

if (inf.Alarmas[i].Identification_Alarm & 0b00010000){
        buffer[j+i]='1';
        } else {
        buffer[j+i]='0';
}
j++;
buffer[j+i]=',';
j++;

if (inf.Alarmas[i].Identification_Alarm & 0b00100000){
        buffer[j+i]='1';
        } else {
        buffer[j+i]='0';
}
j++;
buffer[j+i]=',';
j++;

if (inf.Alarmas[i].Identification_Alarm & 0b01000000){
        buffer[j+i]='1';
        } else {
        buffer[j+i]='0';
}
j++;
buffer[j+i]=',';
j++;

if (inf.Alarmas[i].Identification_Alarm & 0b10000000){
        buffer[j+i]='1';
        } else {
        buffer[j+i]='0';
}
j++;
buffer[j+i]=',';
j++;

if(inf.Alarmas[i].distance.number>=0.0){
        buffer[i+j]='+';
        Aux3 = inf.Alarmas[i].distance.number;
        } else {
        buffer[i+j]='-';
        Aux3 = -inf.Alarmas[i].distance.number;
}
j++;
Aux2 = (int)Aux3;
Aux = Aux2/10000;
Aux2 = Aux2%10000;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/1000;
Aux2 = Aux2%1000;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[i+j]=change_ASCII(Aux);
```

```
j++;
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[i+j]=change_ASCII(Aux);
j++;

buffer[i+j]='.';
j++;

Aux2 = ((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[i+j]=change_ASCII(Aux);
j++;

buffer[i+j]=',';
j++;

if(inf.Alarmas[i].direction.number>=0){
       buffer[i+j]='+';
       Aux3 = inf.Alarmas[i].direction.number;
       } else {
       buffer[i+j]='-';
       Aux3 = -inf.Alarmas[i].direction.number;
}
j++;
Aux2 = (int)Aux3;
Aux = Aux2/100;
Aux2 = Aux2%100;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[i+j]=change_ASCII(Aux);
j++;

buffer[i+j]='.';
j++;

Aux2 = ((int)(Aux3*100)%100);
Aux = Aux2/10;
Aux2 = Aux2%10;
buffer[i+j]=change_ASCII(Aux);
j++;
Aux = Aux2/1;
Aux2 = Aux2%1;
buffer[i+j]=change_ASCII(Aux);
j++;

buffer[i+j]=',';
j++;

buffer[i+j]=inf.Alarmas[i].direction_R_L;
j++;
```

```
                    buffer[i+j]=',';
                    j++;

                    if(inf.Alarmas[i].Difference_Altitude>=0){
                           buffer[i+j]='+';
                           Aux2 = inf.Alarmas[i].Difference_Altitude;
                           } else {
                           buffer[i+j]='-';
                           Aux2 = -inf.Alarmas[i].Difference_Altitude;
                    }
                    j++;
                    Aux = Aux2/100;
                    Aux2 = Aux2%100;
                    buffer[i+j]=change_ASCII(Aux);
                    j++;
                    Aux = Aux2/10;
                    Aux2 = Aux2%10;
                    buffer[i+j]=change_ASCII(Aux);
                    j++;
                    Aux = Aux2/1;
                    Aux2 = Aux2%1;
                    buffer[i+j]=change_ASCII(Aux);
                    j++;

                    buffer[i+j]=',';
                    j++;
                    buffer[i+j]=inf.Alarmas[i].direction_up_down;
                    j++;
             }
      }
      Send_frame(buffer, 38+42*inf.numberE);
}

// Cambia el modo de funcionamiento
void Change_Mode (char *data){
      if(data[1]==0){
             mode = DESACTIVATE;
      } else if(data[1]==2){
             mode=ACTIVATE1;
      } else if(data[1]==3){
             mode=ACTIVATE5;
      } else if(data[1]==4){
             mode=ACTIVATE10;
      } else {
             mode = RESET;
      }
}

// Devuelve el modo de funcionamiento
Bluetooth_mode Get_Mode(void){
      return mode;
}
```

Anexo: Fichero BLUETOOTH.h

Código implementado en el fichero BLUETOOTH.h:

```
#ifndef BLUETOOTH_H_
#define BLUETOOTH_H_

#include "XBEE.h"

/****************************************************************************/
/*                        Typedefs and structures                          */
/****************************************************************************/

typedef enum {SEND, RECEIVED, SEND_RECEIVED} BLUETOOTH_STATUS ;
typedef enum {DESACTIVATE, ACTIVATE1, ACTIVATE5, ACTIVATE10, RESET}
Bluetooth_mode;

// Estructura para los módulos externos
typedef struct {
	char Identification_Alarm; //
	Byte_Float distance; // en metros
	Byte_Float direction; // en grados
	char direction_R_L;
	int Difference_Altitude;
	char direction_up_down;
} Informacion_Bluetooth_Alarma;

// Estructura para los módulo
typedef struct {
	char Identification_Error ; // (0123:identificador, 4:Error GPS, 5:Error
Brújula, 67:Reservado)
	Byte_Float Longitude ;   // en grados
	Byte_Float Latitude ;    // en grados
	Byte_Float orientation; // en grados
	int Altitude ;    // en metros
	char numberE;
	Informacion_Bluetooth_Alarma Alarmas[10];
} Information_Bluetooth ;

/****************************************************************************/
/*                        Exported functions                               */
/****************************************************************************/

int Init_Bluetooth (void); // Inicializamos la comunicación bluetooth
int Send_byte (char B); // Enviamos un byte
int Send_frame (char *data, char length); // Enviamos una trama
char Send_Receive_byte (void); // Comprobamos si ha finalizado la trasmisión
char Sent_frame (void); // Comprobamos si ha finalizado la trasmisión
int Receive_frame (char BI, char length); // Lectura de una trama
char Received_frame (void); // Comprobamos si ha finalizado la recepción
int Get_Frame (char *data); // Almacenamos la trama recibida fuera del fichero
char Get_Byte (void); // Almacenamos el byte recibido fuera del fichero
Information_Bluetooth Generate_information_Bluetooth (char num_Ident, char
ErrorPos, char alarma1, char alarma2, char alarma3, char alarma4, char  ErrorG,
char ErrorM, char ErrorB, position pos, float orient); // Generación de la trama
enviada
void Send_information_Bluetooth (Information_Bluetooth inf); // Envio de la trama
en formato float e int
void Change_Mode (char *data); // cambio de modo
Bluetooth_mode Get_Mode (void); // devuelve el modo de funcionamiento
```

```
void Send_information_Bluetooth_ASCII (Information_Bluetooth inf); // Envio de la
trama en nomenclatura ASCII

#endif /* BLUETOOTH_H_ */
```

Anexo: Fichero BUFFER.c

Código implementado en el fichero BUFFER.c:

```
/****************************************************************************/
/*                          Used modules                                    */
/****************************************************************************/

#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "XBEE.h"
#include "BLUETOOTH.h"
#include "GPS.h"
#include "BRUJULA.h"

/****************************************************************************/
/*                          Local variables                                 */
/****************************************************************************/

const double PI = 3.1415926535 ;
const Earth_float Earth_radius = 6378137.0 ;

static char numIdentification = 1;

static Informacion_Bluetooth_Alarma Information_PCBs_Bluetooth [10];
static Information_XBee Information_PCBs_XBee [10];

/****************************************************************************/
/*                          Prototypes of local functions                   */
/****************************************************************************/

float haversine (float x);
float Distance (const position P1, const position P2);
float normpi (float tita);
float Direction (const position P1, const position P2);

/****************************************************************************/
/*                          Prototypes of exported functions                */
/****************************************************************************/

void Save_Information (char *inf);
void Generate_information (void);
Informacion_Bluetooth_Alarma Get_Information_PCBx (char PCBx);

/****************************************************************************/
/*                          Local functions                                 */
/****************************************************************************/

float haversine (float x){
       return pow(sin(x/2),2) ;
}

float Distance (const position P1, const position P2){
       float h ;
       position X, Y;

       X.Latitude = P2.Latitude*3.1415/180.0;
       X.Longitude = P2.Longitude*3.1415/180.0;
       X.Altitude = 0;
```

```
	Y.Latitude = P1.Latitude*3.1415/180.0;
	Y.Longitude = P1.Longitude*3.1415/180.0;
	Y.Altitude = 0;

	h = haversine(Y.Latitude - X.Latitude) +
cos(Y.Latitude)*cos(X.Latitude)*haversine(Y.Longitude - X.Longitude) ;
	return 2*(Earth_radius + Y.Altitude)*asin(sqrt(h)) ;
}

float Direction (position P1, position P2){
	float d1, d2, nz, mz;
	position X, Y;

	X.Latitude = P2.Latitude*3.1415/180.0;
	X.Longitude = P2.Longitude*3.1415/180.0;
	X.Altitude = 0;

	Y.Latitude = P1.Latitude*3.1415/180.0;
	Y.Longitude = P1.Longitude*3.1415/180.0;
	Y.Altitude = 0;


	nz = Log(tan(Y.Latitude/2+PI/4)/tan(X.Latitude/2+PI/4));
	mz = fabs(X.Longitude-Y.Longitude);
	d1 = atan2(mz,nz)*180/PI;
	d1=360-d1;
	d2=d1-Get_Orientation();
	d2=normpi(d2);

	return d2;
}

float normpi (float tita){
	float tita_aux;

	if(tita<0){
		tita_aux = tita+360;
	} else if(tita>360){
		tita_aux = tita-360;
	} else {
		tita_aux = tita;
	}

	return tita_aux;
}

int Diference (const position P1, const position P2){
	int h;
	h=P1.Altitude-P2.Altitude;
	return h;
}

/******************************************************************************/
/*                          Exported functions                                */
/******************************************************************************/

void Save_Information (char *inf){
	char ident;
	ident= inf[2]&0b00000111;
	ident= ident;
	Information_XBee Aux;
```

```
	int auxA;

	if (ident>numIdentification){
		ident=ident-2;
	} else {
		ident = ident -1;
	}

	// Obtención del identificador
	Aux.Identification_Error_Alarm = inf[2];

	// Obtención de la long de la PCB emisora
	Aux.Longitude.bytes[3]=inf[3];
	Aux.Longitude.bytes[2]=inf[4];
	Aux.Longitude.bytes[1]=inf[5];
	Aux.Longitude.bytes[0]=inf[6];

	// Obtención de la lat de la PCB emisora
	Aux.Latitude.bytes[3]=inf[7];
	Aux.Latitude.bytes[2]=inf[8];
	Aux.Latitude.bytes[1]=inf[9];
	Aux.Latitude.bytes[0]=inf[10];

	// Obtención de la Alt de la PCB emisora
	auxA=inf[11];
	auxA=(auxA) <<8;
	Aux.Altitude= auxA + inf[12];

	// Almacenamos
	Information_PCBs_XBee[ident] = Aux;
}

void Generate_information (void){
	Informacion_Bluetooth_Alarma Aux;
	position aux1;
	position aux2;

	for (int i=0; i<10; i++){

		// Obtención del identificador
		Aux.Identification_Alarm =
Information_PCBs_XBee[i].Identification_Error_Alarm;

		if((Aux.Identification_Alarm&0b10000000) ||
(Aux.Identification_Alarm&0b00100000) || (Aux.Identification_Alarm&0b01000000) ||
(Aux.Identification_Alarm&0b00010000)){

			// Obtención de la long de la PCB emisora
			aux1.Longitude = Information_PCBs_XBee[i].Longitude.number;

			// Obtención de la lat de la PCB emisora
			aux1.Latitude = Information_PCBs_XBee[i].Latitude.number;

			// Obtención de la Alt de la PCB emisora
			aux1.Altitude = Information_PCBs_XBee[i].Altitude;

			// Obtención de la posición de la PCB receptora
			aux2 =  Read_GPS();

			// Obtención de la distancia
			Aux.distance.number = Distance(aux1,aux2);
```

```
				// Obtención de la dirección
				Aux.direction.number = Direction(aux1,aux2);

				if(Aux.direction.number>180){
					Aux.direction_R_L = 'R';
					Aux.direction.number = 360-Aux.direction.number;
				} else {
					Aux.direction_R_L = 'L';
				}

				// Obtención del desnivel
				Aux.Difference_Altitude = Diference(aux1,aux2);

				if(Aux.Difference_Altitude<0){
					Aux.direction_up_down = 'D';
					Aux.Difference_Altitude = -Aux.Difference_Altitude;
				} else {
					Aux.direction_up_down = 'U';
				}
			} else {
				// Obtención de la long de la PCB emisora
				aux1.Longitude = 0.0;

				// Obtención de la lat de la PCB emisora
				aux1.Latitude = 0.0;

				// Obtención de la Alt de la PCB emisora
				aux1.Altitude = 0;

				// Obtención de la distancia
				Aux.distance.number = 0.0;

				// Obtención de la dirección
				Aux.direction.number = 0.0;
				Aux.direction_up_down = 0;

				// Obtención del desnivel
				Aux.Difference_Altitude = 0;
				Aux.direction_up_down = 0;
			}

			// Almacenamos
			Information_PCBs_Bluetooth[i] = Aux;

		}
}

Informacion_Bluetooth_Alarma Get_Information_PCBx (char PCBx){
	return Information_PCBs_Bluetooth[PCBx];
}

void init_buffer (void){ // 50.2 m y 70.59º
	Information_XBee Aux;

	Aux.Identification_Error_Alarm = 0b00010010;
	Aux.Longitude.number = -0.89203;
	Aux.Latitude.number = 41.67456;
	Aux.Altitude = 218;

	Information_PCBs_XBee[0]=Aux;
}
```

```
void init_buffer2 (void){
	Information_XBee Aux;

	Aux.Identification_Error_Alarm = 0b00010011;
	Aux.Longitude.number = -0.89134;
	Aux.Latitude.number = 41.674;
	Aux.Altitude = 219;

	Information_PCBs_XBee[1]=Aux;

}
```

Anexo: Fichero BUFFER.h

Código implementado en el fichero BUFFER.h:

```
#ifndef BUFFER_H_
#define BUFFER_H_

#include "XBEE.h"
#include "BLUETOOTH.h"
#include "GPS.h"

/******************************************************************************/
/*                        Exported functions                                  */
/******************************************************************************/

void Save_Information (char *inf);
void Generate_information (void);
Informacion_Bluetooth_Alarma Get_Information_PCBx (char PCBx);

float haversine (float x);
float Distance (position P1, position P2);
float normpi (float tita);
float Direction (position P1,position P2);
void init_buffer (void);
void init_buffer2 (void);

#endif /* BUFFER_H_ */
```

Anexo: Fichero PRINCIPAL.c

Código implementado en el fichero PRINCIPAL.c:

```
/*****************************************************************************/
/*                        Used modules                                       */
/*****************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "XBEE.h"
#include "SCI.h"
#include "SPI.h"
#include "BLUETOOTH.h"
#include "I2C.h"
#include "GPS.h"
#include "CLOCK.h"
#include "ALARMA.h"
#include "BUFFER.h"
#include "BRUJULA.h"


/*****************************************************************************/
/*                        Global variables                                   */
/*****************************************************************************/

char identificationModule = 1;
unsigned long Siguiente=0;
int cont_marcos = 0;
int cont=0;
int cont_Bluetooth = 0;
int reset=0;
unsigned int marco = 1, num_marcos = 25;
unsigned char m = 25; /* interrupción 2.4 ms con 10us de tcomputo */

/*****************************************************************************/
/*                        Function Prototypes                                */
/*****************************************************************************/

void InitSystem(void);

void Send_XBee (void); // Tarea trasmisión al módulo XBee

void Send_Receive_Bluetooth (void); // Tarea trasmisión/recepción con el módulo
Bluetooth

void Calculation_Alarm (void); // Activación de las alarmas de detección de
accidente

void Calculation_Orientation (void); // Obtención de la orientación


/*****************************************************************************/
/*                        Main                                               */
/*****************************************************************************/
```

```
int main(void)
{
	Init_GPS ()  ;      // Inicialización del módulo GPS (SCIb)
	Init_XBee () ;      // Inicialización del módulo XBee (SCIa)
	Init_Bluetooth();    // Inicialización del módulo Bluetooth (SPI)
	I2C_Init();                 // Inicialización de la comunicación I2C y de
los correspondientes sensores
	Init_clock();       // Inicialización del reloj

	Siguiente = Get_tick_counter(); // Toma del instante de tiempo inicial

	InitSystem();       // Habilitación de las interrupciones

	while(1) {
		/* Ejecutivo cíclico */
		Siguiente = Siguiente + 500;

		switch (marco){
			case 1:
				Calculation_Alarm();
				Calculate_orientation();
				Send_XBee();
				marco = 2;
				break;
			case 2:
				Calculation_Alarm();
				Calculate_orientation();
				Send_Receive_Bluetooth();
				marco = 3;
				break;
			case 3:
				Calculation_Alarm();
				Calculate_orientation();
				if(cont_marcos==12) marco=2;
				if(cont_marcos==24) marco=1;
				break;
		}

		cont_marcos++;
		if(cont_marcos > 24) cont_marcos = 0;

		delay_Until(Siguiente);
	}
	return 0;
}


/*****************************************************************************/
/*                        Functions                                          */
/*****************************************************************************/

// Inicialización del sistema
void InitSystem(void)
{
	sei(); // Habilitamos las interrupciones del video.
	return ;
}
```

```
// Tarea trasmisión al módulo XBee
void Send_XBee (void){

       position pos;
       Information_XBee inf;

       pos=Read_GPS(); //  FUNCIONA
       inf=Generate_information_XBee(identificationModule/*identificador*/,
Get_error()/* Error GPS*/ ,Get_Alarma_Critica(), Get_Alarma_Leve(),
Get_Alarma_Golpe(), Get_Alarm_MeasureA(), pos/*posicion PCB*/);

       // Trasmisión de información en float e int
       Send_information(inf);

       // Trasmisión de información en nomenclatura ASCII
       //Send_information_ASCII(inf);


}

// Tarea recepción/trasmión a través del módulo Bluetooth
void Send_Receive_Bluetooth (void){

       position pos;
       float orientacion;
       Information_Bluetooth inf;

       if((Get_Mode() == DESACTIVATE) && (Get_Mode() == RESET)){
              cont = 0;
       } else if(Get_Mode() == ACTIVATE1){
              cont = 3;
       } else if(Get_Mode() == ACTIVATE5){
              cont = 19;
       } else if(Get_Mode() == ACTIVATE10){
              cont = 39;
       }

       Generate_information();

       if(Get_Mode() != DESACTIVATE && cont_Bluetooth > cont){

              pos=Read_GPS();
              orientacion = Get_Orientation();

       inf=Generate_information_Bluetooth(identificationModule/*identificador*/,
Get_error()/* Error GPS*/,Get_Alarma_Critica(), Get_Alarma_Leve(),
Get_Alarma_Golpe(), Get_Alarm_MeasureA(), Get_Error_MeasureG(),
Get_Error_MeasureM(), Get_Error_MeasureB(), pos/*posicion PCB*/, orientacion);

              // Trasmisión de la información en formato float e int
              Send_information_Bluetooth(inf);

              // Trasmisión de la información en nomenclatura ASCII
              //Send_information_Bluetooth_ASCII(inf);

              cont_Bluetooth = 0;
       } else {
              Receive_frame(0x16, 2); // Recepción de la información
       }
       cont_Bluetooth ++;
}
```

```c
// Tarea activación de detección de accidente
void Calculation_Alarm (void){ // Alarma de caida
	if((Get_Mode()==RESET) && (reset == 0)){
		reset = 1;
		Reset_Alarma_Critica();
		Reset_Alarma_Golpe();
	} else {
		reset = 0;
	}
	Alarm();
}

// Tarea obtención de la orientación
void Calculation_Orientation (void){
	Calculate_orientation();
}
```