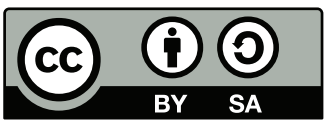
Universidad Complutense de Madrid

Manual de Paparazzi

UCM237

13 de mayo de 2022



Índice general

Índice de figuras

Esto es un simple guía burros para paparazzi, dado que entender lo que dice es complicadillo. En principio, podemos dividirlo en capítulos, y emplear para cada uno un archivo nuevo.

De todos modos, lo importante es ir escribiendo y luego vemos como organizarlo.

Antes de comenzar, indicar que a lo largo de todo este documento vamos a trabajar con Linux, más concretamente con el sistema operativo Ubuntu en su versión 20.04. Este es el único entorno que va a soportar PaparazziUAV.

El equipo que desarrolla PaparazziUAV cuenta con una documentación bastante detallada que puede llegar a ser de gran utilidad para incorporarse al entorno: https://paparazzi-uav.readthedocs.io/en/stable/index.html. Para instalar PaparazziUAV recomendamos al usuario seguir directamente la entrada *installation* (https://paparazzi-uav.readthedocs.io/en/stable/installation/index_installation.html) y leer el *README.md* principal del proyecto (https://github.com/UCM-237/paparazzi).

Capítulo 1

Paparazzi Center

Una vez realizada la instalación, podremos ejecutar el binario precompilado *paparazzi* que se encuentra en la carpeta raíz de Paparazzi. Entonces se inicializará todo el ecosistema y se ejecutará Paparazzi Center, el panel de control principal de PaparazziUAV [Figura 1.1]. En él nos encontraremos inicialmente con un proyecto (A/C) ejemplo. Dentro de cada proyecto podremos introducir varios **archivos de configuración .xml** (en el capítulo sobre el directorio *conf* explicaremos más detalladamente para qué sirve cada uno):

1. **Airframe**: va a contener toda la información sobre el hardware y firmware de nuestro vehículo. https://wiki.paparazziuav.org/wiki/Airframe_Configuration.
2. **Flight Plan**: definirá la geolocalización, los bloques y los waypoints que aparecerán en el modo navegación.
3. **Radio**: ajustará las ganancias y la funcionalidad de los canales del radio control. Paparazzi ya trae la configuración de varios modelos, si nuestro controlador no se encuentra entre ellos deberemos de generar su .xml teniendo en cuenta sus especificaciones.
4. **Telemetry**: definirá los distintos modos de telemetría del vehículo y los mensajes que se enviarán en cada uno de ellos.

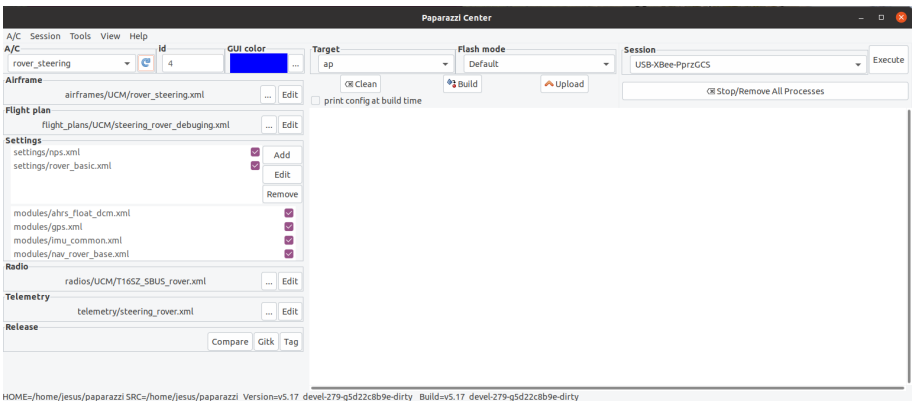


Figura 1.1: Interfaz de Paparazzi Center, el panel de control principal de PaparazziUAV.

Para compilar ...

Herramientas y sesiones ...

Capítulo 2

Ground control station

Capítulo 3

El directorio conf

Capítulo 4

El directorio sw

Capítulo 5

Un ejemplo completo: Rover Steering

Capítulo 6

Módulo de comunicación serie

El módulo de comunicación serie permite que Paparazzi establezca comunicación bidireccional con otro dispositivo a través de uno de los puertos serie. Con este módulo implementaremos el protocolo de comunicaciones entre paparazzi y la Raspberry (o el companion computer) correspondiente.

6.1. Protocolo de comunicación

La comunicación entre Paparazzi y el (companion Computer) CC será bidireccional. Todos los mensajes llevarán una marca de tiempo. Los mensajes establecidos son los siguientes:

1. Mensajes desde Paparazzi al CC
   - *a*) Mensaje con los datos de telemetría.
   - *b*) Mensaje indicando al CC que puede proceder a realizar las medidas.
   - *c*) Mensaje solicitando al CC la posición de la sonda.
2. Mensajes desde el CC al Paparazzi
   - *a*) Mensaje de respuesta a la solicitud de medida.
   - *b*) Mensaje periódico con la profundidad de la sonda (durante la medida).
   - *c*) Mensaje de finalización de medida.
   - *d*) Mensaje de respuesta a la solicitud de posición de la sonda con la posición.

6.1.1. Configuración del puerto serie

La configuración del puerto serie será la estándar con velocidad de 9600 baudios, tabla 7.1.

6.1.2. Comunicación desde paparazzi

Paparazzi enviará tres mensajes diferentes al CC. Uno de ellos periódico, el mensaje de la telemetría y el resto puntualmente.

Todos los mensajes comienzan con un byte que contiene el código ASCII de la letra **P** indicando que dicho mensaje procede de paparazzi. Todos los mensajes terminan con 2 bytes que corresponden al checksum.

Cuadro 6.1: Configuración del puerto serie

Parámetro	**Valor**
Velocidad	9600 (baudios)
Paridad	ninguna
Bits	8
Bits de parada	1
Control de flujo	ninguno

Las cantidades numéricas enteras sin signo se codifican usando el formato Little Endian (el byte más significativo ocupa la posición de menor índice en el mensaje). Por ejemplo si queremos enviar el número decimal 72 como hexadecimal de 2 bytes lo codificaremos como: $\{0x00, 0x48\}$.

Las cantidades numéricas con signo se codifican de igual manera que las enteras sin signo pero el byte de menor índice se usa para indicar el signo: $0x01$ si el signo es negativo y $0x00$ si es positivo. Por ejemplo si usamos 3 bytes para codificar el número -72 lo codificaremos como $\{0x01, 0x00, 0x48\}$ y el número 72 será $\{0x00, 0x00, 0x48\}$.

Mensaje de telemetría

El mensaje de telemetría es un mensaje periódico que envía paparazzi al CC con los datos de telemetría: GPS y medida de Sonar. Tiene una longitud de 25 bytes y la estructura en bytes especificada en la tabla 7.2.

Cuadro 6.2: Estructura del mensaje de telemetría

Posición	**Valor**	**Tipo**	**Número de bytes**
0	Byte de sincronía "P"	ASCII	1
1	Tipo de mensaje "T"	ASCII	1
$2-3$	Marca de tiempo (s)	Entero sin signo	2
$4-8$	Longitud $(grad \cdot 10^7)$	Entero con signo	5
$9-13$	Latitud $(grad \cdot 10^7)$	Entero con signo	5
$14-17$	Altitud (mm)	Entero sin signo	4
$18-21$	Distancia sonar (mm)	Entero sin signo	4
22	Confianza sonar $(\%)$	Entero sin signo	1
$23-24$	Checksum	Entero sin signo	2

En la tabla 7.3 puede verse un ejemplo de uno de estos mensajes.

Mensaje indicando al CC que puede proceder a realizar las medidas

El mensaje de inicio de medidas es un mensaje puntual que envía paparazzi al CC indicando que puede comenzar a descender la sona para tomar medidas. Incluye los datos de telemetría: GPS y medida de Sonar. Tiene una longitud de 25 bytes y la estructura en bytes especificada en la tabla 7.4.

En la tabla 7.5 puede verse un ejemplo de un mensaje de inicio de medida.

Mensaje solicitando al CC la posición de la sonda

El mensaje de solicitud de posición de la sonda es un mensaje puntual que envía paparazzi al CC solicitando la posición de la sonda. Tiene una longitud de 6 bytes y la estructura en bytes

Cuadro 6.3: Ejemplo de un mensaje de telemetría

Byte	**Valor (en hexadecimal)**	**Valor**	**Significado**
0	$0x50$	"P"	Byte sincronía
1	$0x54$	"T"	Tipo mensaje
2	$0x00$	72 segundos	Tiempo
3	$0x48$		
4	$0x01$	-37260579 $(grad \cdot 10^7)$	Longitud
5	$0x02$		
6	$0x38$		
7	$0x8D$		
8	$0x23$		
9	$0x00$	404506308 $(grad \cdot 10^7)$	Latitud
10	$0x18$		
11	$0x1C$		
12	$0x46$		
13	$0xC4$		
14	$0x00$	702385 (mm)	Altitud
15	$0x0A$		
16	$0xB7$		
17	$0xB1$		
18	$0x00$	8219 (mm)	Distancia sonar
19	$0x00$		
20	$0x20$		
21	$0x1B$		
22	$0x64$	100 %	Confianza
23	$0x05$	6405	checksum
24	$0x19$		

Cuadro 6.4: Estructura del mensaje de inicio de medida

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “P”	ASCII	1
1	Tipo de mensaje “M”	ASCII	1
$2-3$	Marca de tiempo (s)	Entero sin signo	2
$4-8$	Longitud $(grad \cdot 10^7)$	Entero con signo	5
$9-13$	Latitud $(grad \cdot 10^7)$	Entero con signo	5
$14-17$	Altitud (mm)	Entero sin signo	4
$18-21$	Distancia sonar (mm)	Entero sin signo	4
22	Confianza sonar (%)	Entero sin signo	1
$23-24$	Checksum	Entero sin signo	2

especificada en la tabla 6.6.

En la tabla 6.7 puede verse un ejemplo de un mensaje de solicitud de posición de la sonda.

6.1.3. Comunicación desde el companion computer

El CC enviará a paparazzi cuatro mensajes diferentes. Uno de esos mensajes es periódico y se enviará cada segundo mientras la sonda está midiendo. El resto son mensajes puntuales.

Todos los mensajes comienzan con un byte que contiene el código ASCII de la letra **R** indicando que dicho mensaje procede de la Raspberry. Todos los mensajes terminan con 2 bytes que corresponden al checksum.

Las cantidades numéricas se codifican de la misma manera que en el caso de la comunicación desde paparazzi al CC.

Mensaje de respuesta a la solicitud de medida

El mensaje de respuesta a la solicitud de medida se enviará por parte del CC a paparazzi como respuesta al mensaje de solicitud de medida.

Tiene una longitud de 6 bytes y la estructura en bytes especificada en la tabla 6.8.

En la tabla 6.9 puede verse un ejemplo de un mensaje de respuesta a la solicitud de medida.

Mensaje periódico con la profundidad de la sonda (durante la medida)

El mensaje periódico con la profundidad de la sonda se enviará por parte del CC a paparazzi con una periodicidad de $1s$ mientras la sonda está midiendo.

Tiene una longitud de 8 bytes y la estructura en bytes especificada en la tabla 6.10.

En la tabla 6.11 puede verse un ejemplo de un mensaje de profundidad de la sonda.

Mensaje de finalización de medida

El mensaje de finalización de la medida se enviará por parte del CC a paparazzi al terminar el procedimiento de medida de la sonda.

Tiene una longitud de 8 bytes y la estructura en bytes especificada en la tabla 6.12.

En la tabla 6.13 puede verse un ejemplo de un mensaje de finalización de medida.

Cuadro 6.5: Ejemplo de un mensaje de inicio de medida

Byte	**Valor (en hexadecimal)**	**Valor**	**Significado**
0	$0x50$	"P"	Byte sincronía
1	$0x4D$	"M"	Tipo mensaje
2	$0x00$	72 segundos	Tiempo
3	$0x48$		
4	$0x01$	-37260579 $(grad \cdot 10^7)$	Longitud
5	$0x02$		
6	$0x38$		
7	$0x8D$		
8	$0x23$		
9	$0x00$	404506308 $(grad \cdot 10^7)$	Latitud
10	$0x18$		
11	$0x1C$		
12	$0x46$		
13	$0xC4$		
14	$0x00$	702385 (mm)	Altitud
15	$0x0A$		
16	$0xB7$		
17	$0xB1$		
18	$0x00$	8219 (mm)	Distancia sonar
19	$0x00$		
20	$0x20$		
21	$0x1B$		
22	$0x64$	100 %	Confianza
23	$0x05$	1311	checksum
24	$0x1F$		

Cuadro 6.6: Estructura del mensaje de solicitud de posición de la sonda

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “P”	ASCII	1
1	Tipo de mensaje “S”	ASCII	1
2 – 3	Marca de tiempo (s)	Entero sin signo	2
4 – 5	Checksum	Entero sin signo	2

Cuadro 6.7: Ejemplo de solicitud de posición de la sonda

Byte	Valor (en hexadecimal)	Valor	Significado
0	$0x50$	“P”	Byte sincronía
1	$0x53$	“S”	Tipo mensaje
2 3	$0x00$ $0x48$	72 segundos	Tiempo
4 5	$0x00$ $0xEB$	235	checksum

Mensaje de respuesta a la solicitud de posición de la sonda con la posición

El mensaje de respuesta a la solicitud de posición de la sonda se enviará por parte del CC a paparazzi como respuesta al mensaje de solicitud de posición por parte de paparazzi.

Tiene una longitud de 8 bytes y la estructura en bytes especificada en la tabla 6.14.

En la tabla 6.15 puede verse un ejemplo de un mensaje de posición de la sonda.

6.2. Archivo de configuración

El archivo de configuración de este módulo es *conf/modules/serial_com.xml*.

Este archivo contiene:

1. La descripción de los parámetros que habrá que configurar: el puerto de conexión y la velocidad

```
<!DOCTYPE module SYSTEM "module.dtd">

<module name="serial_com" dir="com">
<doc>
<description>
Decoder for serial protocol

Data are extracted and sent from a serial link
</description>
<configure name="SERIAL_UART" value="UARTX" description="
    SERIAL on which other device is connected"/>
<configure name="SERIAL_BAUD" value="B9600" description="UART
    Baudrate, default to 9600"/>
</doc>
```

2. Los parámetros para el envío de los mensajes a la telemetría

Cuadro 6.8: Estructura del mensaje de respuesta a la solicitud de medida

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “R”	ASCII	1
1	Tipo de mensaje “O”	ASCII	1
2 – 3	Marca de tiempo (s)	Entero sin signo	2
4 – 5	Checksum	Entero sin signo	2

Cuadro 6.9: Ejemplo de respuesta a la solicitud de medida

Byte	Valor (en hexadecimal)	Valor	Significado
0	$0x52$	“R”	Byte sincronía
1	$0x4F$	“O”	Tipo mensaje
2 3	$0x00$ $0x48$	72 segundos	Tiempo
4 5	$0x00$ $0xE9$	233	checksum

```
<settings>
<dl_settings>
<dl_settings NAME="SR_Com">
<dl_setting MAX="1" MIN="0" STEP="1" VAR="serial_msg_setting"
    shortname="stream" module="modules/com/serial_com" values="
    FALSE|TRUE"/>
</dl_settings>
</dl_settings>
</settings>
```

3. Las dependencias. En este caso el módulo solo depende de *uart*

```
<dep>
<depends>uart</depends>
</dep>
```

4. Las funciones programadas en el módulo (ver 7.2) que permiten iniciar la tarea, reaccionar a eventos y enviar periódicamente los mensajes.

```
<header>
<file name="serial_com.h"/>
</header>
<init fun="serial_init()"/>
<periodic fun="serial_ping()" freq="10" autorun="TRUE"/>
<event fun="serial_event()"/>
```

5. Los parámetros por defecto para el *makefile*

```
<makefile>
```

Cuadro 6.10: Estructura del mensaje periódico con la profundidad de la sonda

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “R”	ASCII	1
1	Tipo de mensaje “M”	ASCII	1
2 – 3	Marca de tiempo (s)	Entero sin signo	2
4 – 5	Profundidad (mm)	Entero sin signo	2
6 – 7	Checksum	Entero sin signo	2

Cuadro 6.11: Ejemplo de mensaje de profundidad de la sonda

Byte	Valor (en hexadecimal)	Valor	Significado
0	$0x52$	“R”	Byte sincronía
1	$0x4D$	“M”	Tipo mensaje
2 3	$0x00$ $0x48$	72 segundos	Tiempo
4 5	$0x3D$ $0xCC$	15820 mm	Profundidad
6 7	$0x01$ $0xF0$	496	checksum

```
2          <configure name="SERIAL_UART" default = "UART5" case="upper|
               lower"/>
3          <configure name="SERIAL_BAUD" default="B9600"/>
4          <file name="serial_com.c"/>
5          <define name="USE_$(SERIAL_UART_UPPER)"/> <!-- for uart_arch--
               >
6          <define name="SERIAL_DEV" value="$(SERIAL_UART_LOWER)"/>
7          <define name="$(SERIAL_UART_UPPER)_BAUD" value="$(SERIAL_BAUD)
               "/>
8          </makefile>
9          </module>
```

6.3. Archivos de código

El código que gobierna el funcionamiento de la comunicación a través del puerto serie está en los archivos: *serial_com.c* y *serial_com.h*.

6.3.1. Estructuras

Se han creado dos estructuras para almacenar el mensaje que se envía y el que se recibe.

```
1  struct serial_send_t {
2
3  uint8_t msg_length;
4
5  uint8_t msgData[SERIAL_MAX_PAYLOAD];
6  uint8_t error_cnt;
7  uint8_t error_last;
```

Cuadro 6.12: Estructura del mensaje de finalización de medida

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “R”	ASCII	1
1	Tipo de mensaje “F”	ASCII	1
$2-3$	Marca de tiempo (s)	Entero sin signo	2
$4-5$	Profundidad (mm)	Entero sin signo	2
$6-7$	Checksum	Entero sin signo	2

Cuadro 6.13: Ejemplo de mensaje de finalización de medida

Byte	Valor (en hexadecimal)	Valor	Significado
0	$0x52$	“R”	Byte sincronía
1	$0x46$	“M”	Tipo mensaje
2 3	$0x00$ $0x48$	72 segundos	Tiempo
4 5	$0x00$ $0x00$	0 mm	Profundidad
6 7	$0x00$ $0xE0$	224	checksum

```
8
9
10 uint16_t time;
11
12 int32_t lon;
13 int32_t lat;
14 int32_t alt;
15
16 uint32_t distance;
17 uint8_t confidence;
18
19 uint16_t ck;
20
21 };
```

```
1  struct serial_parse_t {
2
3  uint8_t msg_id;
4
5  uint8_t msgData[SERIAL_MAX_MSG] __attribute__((aligned));
6  uint8_t status;
7
8  uint8_t error_cnt;
9  uint8_t error_last;
10
11 uint8_t payload_len;
12
13 uint16_t ck;
14 bool msg_available;
```

Cuadro 6.14: Estructura del mensaje de posición de la sonda

Posición	Valor	Tipo	Número de bytes
0	Byte de sincronía “R”	ASCII	1
1	Tipo de mensaje “P”	ASCII	1
2 – 3	Marca de tiempo (s)	Entero sin signo	2
4 – 5	Profundidad (mm)	Entero sin signo	2
6 – 7	Checksum	Entero sin signo	2

Cuadro 6.15: Ejemplo de mensaje de posición de la sonda

Byte	Valor (en hexadecimal)	Valor	Significado
0	$0x52$	“R”	Byte sincronía
1	$0x50$	“P”	Tipo mensaje
2 3	$0x00$ $0x48$	72 segundos	Tiempo
4 5	$0x12$ $0xA8$	4776 mm	Profundidad
6 7	$0x01$ $0xA4$	420	checksum

```
15
16
17 uint16_t time;
18 uint16_t depth;
19 };
```

6.3.2. Funciones auxiliares

También una serie de funciones auxiliares que facilitan la codificación y decodificación de los mensajes:

- Función que devuelve la longitud total del mensaje teniendo en cuenta los bytes de cabecera, los bytes del checksum y los bytes del contenido del mensaje.

```
1          static uint32_t msgLength(void);
```

- Función que devuelve el valor del *checksum* y lo escribe en el campo `ck` de la estructura del mensaje recibido `serial_parse_t`.

```
1          static uint32_t serial_calculateChecksum(void);
```

- Función que calcula el valor del *checksum* del mensaje que va a enviarse, lo codifica en 2 bytes y lo escribe en las posiciones correspondientes del mensaje de salida. Lo escribe también en el campo `ck` de la estructura `serial_send_t`.

```
1          void serial_calculateChecksumMsg(uint8_t *msg, int msgLength);
```

- Convierte una cadena de bytes de longitud `length` en un entero sin signo. Devuelve el entero sin signo. Considera codificación estilo *Little Endian*.

```
unsigned int serial_byteToint(uint8_t * bytes,int length);
```

- Codifica un entero sin signo en 2 bytes.Considera codificación estilo *Little Endian*.

```
void ito2h(int value, unsigned char* str);
```

- Codifica un entero con signo en una cadena de longitud `nbytes`. Si el tamaño del entero sobrepasa el número de bytes de la cadena no realiza el cálculo. El LSB corresponde al signo, si $LSB = 0x01$ el número es negativo y si $LSB = 0x00$ el número es postivo.

```
void itoh(int value, unsigned char* str, int nbytes);
```

6.3.3. Funciones de decodificación de los mensajes que se reciben

Estas funciones realizan la recepción, almacenamiento y decodificación de los mensajes que llegan a paparazzi.

- Esta función recibe uno a uno los bytes que se leen por el puerto serie y los va almacenando en el campo `msgData` de la estructura `serial_parse_t`. Comprueba que el byte inicial es correcto. Si se recibe algún byte que no concuerda con la estructura esperada del mensaje rellena el campo `error_last` con el error `SERIAL_ERROR_UNEXPECTED`. Cuando se han recibido todos los bytes que completan un mensaje se indica que hay un mensaje disponible para la decodificación poniendo el campo `available` de la estructura `serial_parse_t` a *true*.

```
static void serial_parse(uint8_t byte);
```

- Esta función decodifica el mensaje almacenado en el campo `msgData` de la estructura `serial_parse_t` rellenando adecuadamente los campos de datos `depth` y `time`. Para ello se ayuda de las dos funciones `message_parse` y `message_OK_pase` explicadas más adelante. Comprueba que el valor del *checksum* contenido en el mensaje coincide con la suma de los bytes del mensaje. De no ser así **sigue decodificando el mensaje** pero devuelve el error `SERIAL_BR_ERR_CHECKSUM` en el campo `error_last`.

```
void serial_read_message(void);
```

- Decodifica el mensaje `RO` alamacenando el valor del tiempo `time`.

```
static void message_OK_parse(void);
```

- Decodifica el resto de mensajes almacenando los valores `time` y `depth`.

```
static void message_parse(void);
```

6.3.4. Función para el envío de bytes por el puerto serie

La siguiente función es la encargada de enviar un mensaje byte a byte por el puerto serie. Necesita un puntero al array de bytes que se quieren enviar y la longitud del array.

```
1 static void serial_send_msg(uint8_t len, uint8_t *bytes);
```

Hay también una función que se invoca periódicamente (sección 7.2) que se encarga de enviar el mensaje seleccionado por el puerto serie. Para ello comprueba que se ha completado la recepción de un mensaje completo y selecciona con una estructura `switch-case` el mensaje que se va a enviar. Es la variable global `message_type` la que indica el mensaje seleccionado. Dicha variable puede tomar los siguientes valores: `TELEMETRY_SN` si se quiere enviar un mensaje de telemetría (ver 7.1), `SONDA_RQ` si se quiere solicitar la profundidad a la que se encuentra la sonda (ver 7.1) o `MEASURE_SN` si se quiere indicar al CC que comience a bajar la sonda para medir (ver 7.1). En el caso de que la variable `message_type` tenga un valor distinto de los tres anteriores se indicará con un error `SERIAL_ERR_UNEXPECTED`.

Esta función construye en mensaje que se quiere enviar con los bytes adecuados e invoca a la función `serial_send_message`. Para ello obtiene la posición del GPS (en coordenadas geodésicas) invocando la función `stateGetPositionLla_i()`y los datos del sonar mediante la función `sonar_get()` y los codifica en bytes según el convenio

```
1 void serial_ping(void);
```

6.3.5. Función para la recepción de bytes por el puerto serie

La siguiente función se invoca cuando se produce un evento asociado al puerto serie. En ese caso se comprueba si hay un caracter disponible y de ser así se obtiene y se envía al decoficador de bytes `serial_parse`. Si tras decodificar ese byte el mensaje está completo se invoca a la función de decodificación del mensaje `serial_read_message`.

```
1  void serial_event(void);
```

6.3.6. Otras funciones

Finalmente tenemos otras dos funciones:

- Función para inicializar la comunicación serie. Esta función ha de invocarse en el archivo de configuración como función de `init` (ver 7.2).

```
1          void serial_init(void);
```

- Función para enviar los datos por telemtría. Esta función hace uso del mensaje `SERIAL_COM` de la telemetría para mostrar en el interfaz gráfico tanto los mensajes que se envían como los mensajes que se reciben.

```
1          static void send_telemetry(struct transport_tx *trans, struct
               link_device *dev);
```

Capítulo 7

Módulo de comunicación con el sonar de Bluerobotics

En este capítulo se describe el software desarrollado para el módulo de comunicación con el ping sonar de Bluerobotics.

7.1. Archivo de configuración

El archivo de configuración del sonar es `sonar_bluerobotics.xml`.
En este achivo xml se configura la ejecución del módulo.

- En primer lugar (tras la descripción) se configura el envío de los mensajes de telemetría.

```
<!DOCTYPE module SYSTEM "module.dtd">

<module name="sonar_bluerobotics" dir="sonar">
<doc>
<description>
Decoder for ping protocol from Blue

Data are extracted from a serial link
</description>
<configure name="SONAR_UART" value="UARTX" description="UART
    on which sonar is connected"/>
<configure name="SONAR_BAUD" value="B115200" description="UART
     Baudrate, default to 115200"/>
</doc>
<settings>
<dl_settings>
<dl_settings NAME="BR_Sonar">
<dl_setting MAX="1" MIN="0" STEP="1" VAR="sonar_stream_setting
    " shortname="stream" module="modules/sonar/
    sonar_bluerobotics" values="FALSE|TRUE"/>
</dl_settings>
</dl_settings>
</settings>
```

- Se establecen las dependencias. En este caso puesto que la comunicación con el sonar es vía puerto serie la única dependencia es con la biblioteca de la uart.

```
<dep>
<depends>uart</depends>
</dep>
```

- Se configura el fichero de cabecera y las funciones de la biblioteca del sonar que van a invocarse. Estas funciones (que se encuentran descritas en la sección 7.3) son: la función de inicialización, la función que se ejecuta periódicamente para enviar mensajes o decodificar mensajes si hay un mensaje completo y la función que se invoca cuando hay un evento en el puerto serie.

```
<header>
<file name="sonar_bluerobotics.h"/>
</header>
<init fun="sonar_init()"/>
<periodic fun="sonar_ping()" freq="5" autorun="TRUE"/>
<event fun="sonar_event()"/>
```

- Por último se configuran los parámetros del puerto para el makefile.

```
<makefile>
<configure name="SONAR_UART" default = "UART3" case="upper|
    lower"/>
<configure name="SONAR_BAUD" default="B115200"/>
<file name="sonar_bluerobotics.c"/>
<define name="USE_$(SONAR_UART_UPPER)"/> <!-- for uart_arch-->
<define name="SONAR_DEV" value="$(SONAR_UART_LOWER)"/>
<define name="$(SONAR_UART_UPPER)_BAUD" value="$(SONAR_BAUD)"/
    >
</makefile>
</module>
```

7.1.1. Configuración del puerto serie

En la tabla 7.1 se muestra la configuración del puerto serie.

Cuadro 7.1: Configuración del puerto serie

Parámetro	**Valor**
Velocidad	115200 (baudios)
Paridad	ninguna
Bits	8
Bits de parada	1
Control de flujo	ninguno

7.2. Formato de los mensajes implementados

7.2.1. Mensajes de paparazzi al sonar

Se han implementado los siguientes mensajes:

- Request protocol version. Solicita al sonar la versión de protocolo que está usando (tabla 7.2).
- Request simple distance. Solicita al sonar un mensaje con la distancia medida y la confianza (tabla 7.3).
- Request simple distance start. Solicita al sonar que envíe cada 250 (ms) el mensaje con la distancia simple (contiene distancia medida y confianza) (tabla 7.4).
- Request simple distance start. Solicita al sonar que cese de enviar el mensaje de distancia simple (tabla 7.5).
- Request speed of sound. Solicita al sonar que envíe el valor de la velocidad del sonido que está usando (tabla 7.6).
- Request device id. Solicita al sonar que envíe su identificador (tabla 7.7).

Cuadro 7.2: Request protocol version

Byte	**Valor (hexadecimal)**	**Significado**
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x06$	6_L Message ID
5	$0x00$	0_H Message ID
6	$0x00$	0 Source ID
7	$0x00$	0 Device ID
8	$0x05$	5_L Requested Message ID
9	$0x00$	0_H Requested Message ID
10	$0xA1$	A1_L Checksum
11	$0x00$	0_H Checksum

7.2.2. Mensajes del sonar a paparazzi

Se ha implementado la decodificación de los siguientes mensajes:

- Distancia simple (tabla 7.8).
- Version de protocolo (tabla 7.9).
- Velocidad del sonido (tabla 7.10).
- Identificador del dispositivo (tabla 7.11).

Cuadro 7.3: Request distance simple

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x06$	6_L Message ID
5	$0x00$	0_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8	$0xBB$	BB_L Requested Message ID
9	$0x04$	04_H Requested Message ID
10	$0x5C$	5C_L Checksum
11	$0x01$	01_H Checksum

Cuadro 7.4: Request distance simple streaming start

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x78$	78_L Message ID
5	$0x05$	05_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8	$0xBB$	BB_L Requested Message ID
9	$0x04$	04_H Requested Message ID
10	$0xD3$	D3_L Checksum
11	$0x01$	01_H Checksum

7.3. Código

7.3.1. Estructuras

Se ha implementado la siguiente estructura para almacenar los datos que se reciben del sonar:

```
1
2  struct sonar_parse_t {
3  uint16_t payload_len;
4  uint16_t msg_id;
5  uint8_t src_id;
6  uint8_t dev_id;
7
8  uint8_t msgData[SONAR_MAX_PAYLOAD] __attribute__((aligned));
9  uint8_t status;
10
11 uint8_t error_cnt;
```

Cuadro 7.5: Request distance simple streaming stop

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x79$	79_L Message ID
5	$0x05$	05_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8	$0xBB$	BB_L Requested Message ID
9	$0x04$	04_H Requested Message ID
10	$0xD4$	D4_L Checksum
11	$0x01$	01_H Checksum

Cuadro 7.6: Request speed of sound

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x06$	06_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8	$0xB3$	B3_L Requested Message ID
9	$0x04$	04_H Requested Message ID
10	$0x54$	54_L Checksum
11	$0x01$	01_H Checksum

```
12 uint8_t error_last;
13
14 uint16_t ck;
15 bool msg_available;
16
17 uint8_t protocol_version;
18 uint8_t protocol_subversion;
19 uint8_t protocol_patch;
20
21 uint32_t sound_vel;
22 uint32_t distance;
23 uint8_t confidence;
24 };
```

Cuadro 7.7: Request device id

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x02$	2_L Payload length
3	$0x00$	0_H Payload length
4	$0x06$	06_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8	$0xB1$	B1_L Requested Message ID
9	$0x04$	04_H Requested Message ID
10	$0x52$	52_L Checksum
11	$0x01$	01_H Checksum

Cuadro 7.8: Distancia simple (simple distance)

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x05$	5_L Payload length
3	$0x00$	0_H Payload length
4	$0x04$	04_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
$8-11$		Distancia (mm)
12		Confianza (%)
13		_L Checksum
14		_H Checksum

7.3.2. Funciones auxiliares

Se han implementado las siguientes funciones auxiliares que serán necesarias para la codificación y decodificación de los mensajes:

- Función que calcula la longitud total del mensaje

```
1                  static uint32_t msgLength(void);
```

- Función que calcula el checksum del mensaje, lo retorna y lo almacena en el campo `ck` de la estructura `sonar_parse_t`.

```
1          static uint32_t calculateChecksum(void);
```

- Función que convierte en entero sin signo una cadena de bytes de longitud `length`.

Cuadro 7.9: Versión del protocolo (protocol version)

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x03$	3_L Payload length
3	$0x00$	0_H Payload length
4	$0x03$	03_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8		Versión
9		Subversión
10		Parche
11		_L Checksum
12		_H Checksum

Cuadro 7.10: Velocidad del sonido

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x04$	4_L Payload length
3	$0x00$	0_H Payload length
4	$0x02$	02_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
$8-11$		Velocidad sonido (mm/s)
12		_L Checksum
13		_H Checksum

```
1          unsigned int byteToint(uint8_t * bytes,int length);
```

7.3.3. Función `get`

Esta función retorna un puntero a la estructura `sonar_parse_t` que contiene los datos leidos del sonar. Esta función será la que invoquemos desde otros módulos cuando queramos leer las medidas del sonar.

```
1 struct sonar_parse_t * sonar_get(void);
```

7.3.4. Funciones para decodificar mensajes del sonar

- Esta función recibe un byte leido por el puerto serie y lo va almacenando en el campo `msgData` de la estructura `sonar_parse_t`. Comprueba que los dos primeros bytes del men-

Cuadro 7.11: Identificador del sonar

Byte	Valor (hexadecimal)	Significado
0	$0x42$	"B"
1	$0x52$	"R"
2	$0x01$	1_L Payload length
3	$0x00$	0_H Payload length
4	$0x05$	05s_L Message ID
5	$0x00$	00_H Message ID
6	$0x00$	0 Source ID
7	$0x01$	1 Device ID
8		ID
9		_L Checksum
10		_H Checksum

saje son correctos. En caso de no serlo o de recibir un valor inesperado almacena un error. Cuando ha recibido y almacenado un mensaje completo escribe el valor *true* en el campo `message_available` de la estructura para indicarlo.

```
1          static void sonar_parse(uint8_t byte);
```

- Esta función actúa cuando hay un mensaje completo disposible (*message_available* = *true*). Analiza el valor del campo `msg_id` y llama a la función que corresponda según el tipo de mensaje. Si el valor no coincide con ninguno de los valores predeterminados lo indica en el campo `error_last`. Esta función también comprueba que el valor del *checksum* del mensaje es correcto.

```
1          void sonar_read_message(void);
```

- Esta función decodifica el mensaje que devuelve el identificador del sonar. Almacena el identificador en el campo `dev_id`.

```
1          static void sonar_parse_device_id(void);
```

- Esta función decodifica el mensaje que devuelve la versión del protocolo usado. Almacena los resultados en los campos `protocol_version`, `protocol_subversion` y `protocol_patch`.

```
1          static void sonar_parse_firmware(void);
```

- Esta función se encarga de la decodificación del mensaje que devuelve el valor de la velocidad del sonido empleada. Almacena el valor obtenido en el campo `sound_vel`.

```
1          static void sonar_parse_sound_speed(void);
```

- Esta función decodifica el mensaje de distancia sencillo. Almacena la distancia medida en el campo `distance` y la confianza en el campo `confidence`.

```
1          static void sonar_parse_simple_distance(void);
```

7.3.5. Función para el envío de bytes por el puerto serie

Esta función recibe un puntero a un array que contiene los bytes que se quieren enviar y la longitud de dicho array. Se encarga de ir enviando uno a uno los bytes por el puerto serie.

```
1 static void sonar_send_msg(uint8_t len, uint8_t *bytes);
```

Hay también una función que se invoca periódicamente (sección 7.2) que se encarga de enviar el mensaje seleccionado por el puerto serie. Para ello comprueba que se ha completado la recepción de un mensaje completo y selecciona con una estructura `switch-case` el mensaje que se va a enviar. Es la variable global `modo` la que indica el mensaje seleccionado. Dicha variable puede tomar los siguientes valores: `BR_RQ_SONAR_START` si se quiere enviar un mensaje petición de envío en streaming del mensaje de distancia sencillo (ver 7.1), `BR_RQ_SONAR_STOP` si se quiere solicitar que se deje de enviar en streaming el mensaje de distancia sencillo (ver 7.1), `BR_RQ_FIRMWARE` si se quiere solicitar la versión del protocolo que emplea el sonar, `BR_RQ_SOUND_VEL` si se quiere solicitar el envío de la velocidad del sonido, `BR_RQ_DISTANCE_SIMPLE` si se quiere solicitar el envío de un mensaje con la distancia sencilla o `BR_RQ_DEVICE_ID` si se quiere solicitar el envío del identificador del sonar (ver 7.1). En el caso de que la variable `message_type` tenga un valor distinto de los tres anteriores se indicará con un error `SONAR_BR_ERR_UNEXPECTED`.

```
1 void serial_ping(void);
```

7.3.6. Función para la recepción de bytes por el puerto serie

Esta función se invoca cuando hay un evento asociado al puerto serie. Comprueba si hay un byte disponible y si es así lo recupera y se lo envía a la función `sonar_parse`. En el caso de que al recibir ese byte se complete un mensaje (el campo `available` es *true*) invoca a la función `sonar_read_message` para que empiece la decodificación.

```
1 void sonar_event(void);
```

7.3.7. Otras funciones

Finalmente tenemos otras dos funciones:

- Función para inicializar la comunicación serie. Esta función ha de invocarse en el archivo de configuración como función de `init` (ver 7.2).

```
1          void sonar_init(void);
```

- Función para enviar los datos por telemetría. Esta función hace uso del mensaje `BR_SONAR` de la telemetría para mostrar en el interfaz gráfico los mensajes que se reciben del sonar.

```
1          static void send_telemetry(struct transport_tx *trans, struct
               link_device *dev);
```