Procesador de datos de sensores GNSS

Protocolos NMEA-0183/UBX 8 – Formato GPX

Algoritmos y Programación I (95.11/75.02)

16 de noviembre de 2018

1. Objetivo

El objetivo del presente trabajo es la integración de los conocimientos adquiridos hasta el momento en el curso, a través del diseño y desarrollo de una aplicación que permita procesar los datos provenientes de un sensor de un sistema global de navegación por satélites, GNSS por sus siglas en inglés.

2. Alcance

Mediante el presente TP se busca que el/la estudiante aplique conocimientos sobre los siguientes temas:

- Directivas al preprocesador C
- Programas en modo consola
- Tipos enumerativos
- Funciones
- Salida de datos con formato
- Modularización
- Arreglos/Vectores
- Memoria dinámica
- Arg. en línea de comandos
- Estructuras
- Operadores de Bits
- Archivos
- Punteros a función
- Modularización (bis)
- Tipos de Dato Abstracto, particularmente contenedores

Fecha de entrega: 6 de diciembre de 2018

3. Introducción

En el trabajo anterior implementaron un procesador de sentencias GGA del protocolo NMEA-0183 y las imprimieron en formato GPX. Se asume que entienden qué es el protocolo NMEA-0183 y el formato GPX, pero en caso contrario, pueden repasarlo en el enunciado del trabajo práctico anterior [**2018c1tp1**].

Este trabajo práctico extenderá la funcionalidad del trabajo anterior, agregando el procesamiento de nuevas sentencias y un nuevo protocolo, más reciente que el NMEA[1], el protocolo *UBX*[2]. La aplicación desarrollada deberá ser capaz de interpretar datos provenientes tanto de un sensor que entregue los datos utilizando el protocolo NMEA-0183, como de uno que entregue los datos utilizando el protocolo UBX. Estos datos deberán ser impresos en formato GPX[3].

3.1. Protocolo UBX

El protocolo UBX fue (y es actualmente) desarrollado por la empresa u-blox[1], de origen suizo. Este protocolo es un protocolo binario, por lo que los datos pueden procesarse a mayor velocidad que en un formato compuesto por caracteres ASCII y puede transportar la misma información en menor cantidad de bytes (algo muy positivo ;)). La hoja técnica del protocolo se encuentra en inglés y puede obtenerse del siguiente enlace (para un receptor en particular que utiliza dicho protocolo): https://www.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_(UBX-13003221)_Public.pdf.

El *frame* o estructura del mensaje se muestra a continuación, y corresponde a la sección 33.2 de la hoja técnica[2].

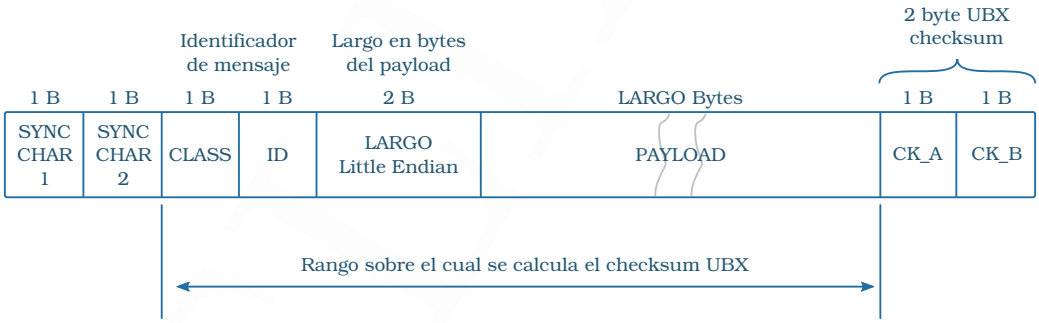

Figura 1: Estructura del *frame* UBX

- Todos los frames comienzan con un preámbulo de 2 bytes que consiste en 2 caracteres de sincronismo: `0xB5 0x62`. Esos 2 bytes son equivalentes al signo $ en la sentencia NMEA.
- A continuación, siguen 2 bytes que identifican el mensaje. El primero identifica una clase y el segundo el mensaje dentro de la clase.
- Le siguen 2 bytes en los que se almacena el **largo** del *payload* en *little endian*.
- Luego se encuentra el *payload*, compuesto por **largo** bytes.

[1] u-blox: www.u-blox.com

- El frame finaliza con 2 caracteres de *checksum*, `CK_A` y `CK_B`, que forman el checksum de 16 bits.

3.1.1. Cálculo del checksum

Sea `buffer` un arreglo con `N` bytes. Sean `ck_a` y `ck_b` dos variables de **8 bits**, sin signo. El algoritmo es el siguiente:

```
Datos de entrada: buffer, N
Datos de salida: ck_a, ck_b
ubx_checksum():
    ck_a <- 0
    ck_b <- 0
    Para cada byte en buffer, hacer:
        ck_a <- ck_a + byte
        ck_b <- ck_b + ck_a
```

4. Desarrollo

En este trabajo se pide que implementar un programa ejecutable por línea de comandos que lea de un archivo de entrada una secuencia de datos provenientes de un sensor e imprima por un flujo de salida el contenido que se genera para un archivo GPX.

Para realizar esta tarea, el programa deberá ir analizando los datos recibidos, interpretar aquellos que tienen información útil, procesarlos e imprimir en el formato especificado.

El programa deberá ser capaz de almacenar una cierta cantidad de mensajes en una lista de mensajes en espera a ser procesados.

El programa deberá contener un ciclo de procesamiento donde se almacenan los mensajes y luego se imprime una cantidad de mensajes a calcular por el programa.

4.1. Procesamiento NMEA-0183

Además de las sentencias `GGA`, en este trabajo se procesarán las sentencias `ZDA` y `RMC`. Estas son sentencias que traen información de tiempo y servirán para dar una referencia a los mensajes que se procesan con las sentencias `GGA`.

4.1.1. Sentencia RMC

Contiene la mínima cantidad de información recomendada para hacer navegación. Para ver el formato de esta sentencia, tomemos una línea del archivo de ejemplo:

```
$GPRMC,hhmmss.sss,S,ddmm.mmm,C,dddmm.mmm,C,dddddd.d,ddd.d,ddmmyy,mmm.m,m*cc
$GPRMC,112836.854,A,1955.118,N,07742.541,W,198121.7,078.3,300918,000.0,W*52

Donde:
   $            Caracter inicial de TODAS las sentencias
```

```
   GP           ID del talker (GPS)
   RMC          Tipo de la sentencia (Mínimo Recomendado C)
   112836.854   Horario del fix (11:28:36.854 UTC): hhmmss.sss
   A            Status A=active or V=Void
   1955.118     Latitud (19 deg 55.118'): ddmm.mmm
   N            Indicador de latitud Norte o Sur (N): S/N
   07742.541    Longitud (77 deg 42.541'): dddmm.mmm
   W            Indicador de longitud Este u Oeste (W): E/W
   198121.7     Velocidad sobre el suelo, en nudos
   078.3        Ángulo de seguimiento en grados
   300918       Fecha - 30 de septiembre de 2018
   000.0,W      Desviación magnética
   *52          suma de verificación (siempre comienza con *)
```

Estos datos deben cargarse en una estructura diseñada para tal fin.

4.1.2. Sentencia ZDA

Contiene información sobre la fecha.

```
$GPZDA,hhmmss.ss,dd,mm,yyyy,xx,yy
$GPZDA,003912.00,13,11,2018,00,00*66

Donde:
   $            Caracter inicial de TODAS las sentencias
   GP           ID del talker (GPS)
   ZDA          Tipo de la sentencia (fecha)
   003912.00    Horario (00:39:12.00 UTC): hhmmss.sss
   13,11,2018   Fecha (13/11/2018)
   00           Zona horaria (horas de diferencia con UTC)
   00           Minutos de diferencia de la zona horaria
   *66          suma de verificación
```

Estos datos deben cargarse en una estructura diseñada para tal fin.

4.2. Procesamiento UBX

En el formato binario de UBX se procesarán los mensajes de tipo `UBX-TIM-TOS`, `UBX-NAV-POSLLH` y `UBX-NAV-PVT`. Estos mensajes dan información sobre el tiempo y el posicionamiento. Los mensajes a procesar son *similares* a los correspondientes en NMEA, y al igual que en dicho caso, deben volcarse los datos del mensaje a una estructura definida a tal efecto.

Ante cualquier duda en la estructura del mensaje, referirse a la figura 1.

4.2.1. Mensaje UBX-NAV-PVT

Tabla 1: Estructura del mensaje `UBX-NAV-PVT`

SYNC	*Clase*	*ID*	*Largo (bytes)*	*Payload*	*Checksum*
`0xB5 0x62`	`0x01`	`0x07`	92	ver abajo	ck_a ck_b

Payload:

Byte	*Fmt.*	*Scale*	*Nombre*	*Unidad*	*Descripción*
0	U4	–	iTOW	ms	Tiempo transcurrido desde el EPOCH GPS
4	U2	–	year	y	Año (UTC)
6	U1	–	mes	mes	Mes, rango 1..12 (UTC)
7	U1	–	dia	día	Día del mes, rango 1..31 (UTC)
8	U1	–	hora	hora	Hora del día rango 0..23 (UTC)
9	U1	–	min	min	Minuto de la hora, rango 0..59 (UTC)
10	U1	–	sec	s	Segundo del minuto, rango 0..60 (UTC)
11	X1	–	valid	—	Registro binario *validity flags* (ver abajo)
12	U1[8]	–	—	—	8 bytes a ignorar
20	U4	–	fix	—	Tipo de fix GNSS: 0: sin fix 1: sólo *dead-reckoning* 2: fix 2D 3: fix 3D 4: combinado GNSS + *dead-reckoning* 5: fix con tiempo únicamente
21	X1	–	flags	—	Registro binario *fix status* (ver abajo)
22	U1	–	—	—	1 byte a ignorar
23	U1	–	numSV	—	Cantidad de satélites usados
24	I4	1e-7	lon	deg	Longitud
28	I4	1e-7	lat	deg	Latitud
32	I4	–	altura	mm	Altura sobre el elipsoide
36	I4	–	hMSL	mm	Altura sobre el nivel del mar
40	U1[36]	–	—	—	36 bytes a ignorar
76	U2	0.01	pDOP	—	DOP de la posición
78	U1[6]	–	reserved	—	Reservado
84	U1[8]	–	—	—	8 bytes a ignorar

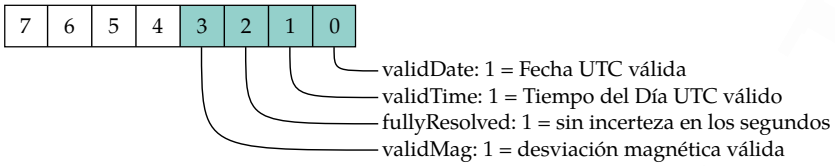

Figura 2: Campos del registro *valid* (byte 11 del payload)

Campos del registro *flags*: (byte 21 del payload) el bit 0 del registro *flags* se denomina *gnssFixOK* y tiene el valor 1 si el fix es válido.

4.2.2. Mensaje UBX-TIM-TOS

Tabla 2: Estructura del mensaje `UBX-TIM-TOS`

SYNC	*Clase*	*ID*	*Largo (bytes)*	*Payload*	*Checksum*
`0xB5 0x62`	`0x0D`	`0x12`	56	ver abajo	ck_a ck_b

Payload:

Byte	*Fmt.*	*Scale*	*Nombre*	*Unidad*	*Descripción*
0	U1	–	version	—	Versión del mensaje
1	U1	–	gnssId	—	GNSS utilizado
2	U1[6]	–	—	—	6 bytes a ignorar
8	U2	–	year	y	Año (UTC)
10	U1	–	mes	mes	Mes, rango 1..12 (UTC)
11	U1	–	dia	día	Día del mes, rango 1..31 (UTC)
12	U1	–	hora	hora	Hora del día rango 0..23 (UTC)
13	U1	–	min	min	Minuto de la hora, rango 0..59 (UTC)
14	U1	–	sec	s	Segundo del minuto, rango 0..60 (UTC)
15	U1[9]	–	—	—	9 bytes a ignorar
24	U4	–	week	—	número de semana GNSS
28	U4	–	TOW	seg	tiempo (segundos) transcurridos en la semana
32	U1[24]	–	—	—	24 bytes a ignorar

4.2.3. Mensaje UBX-NAV-POSLLH

Tabla 3: Estructura del mensaje `UBX-NAV-POSLLH`

SYNC	*Clase*	*ID*	*Largo (bytes)*	*Payload*	*Checksum*
`0xB5 0x62`	`0x01`	`0x02`	28	ver abajo	ck_a ck_b

Payload:

Byte offset	*Formato*	*Escalado*	*Nombre*	*Unidad*	*Descripción*
0	U4	–	iTOW	ms	Tiempo transcurrido desde el EPOCH GPS
4	I4	1e-7	lon	deg	Longitud
8	I4	1e-7	lat	deg	Latitud
12	I4	–	altura	mm	Altura sobre el elipsoide
16	I4	–	hMSL	mm	Altura sobre el nivel del mar
20	U4	–	hAcc	mm	Precisión horizontal estimada
24	U4	–	vAcc	mm	Precisión vertical estimada

4.3. Funcionamiento del programa

El programa a desarrollar debe leer mensajes del receptor GNSS, a través de un archivo ya que no se dispone de las herramientas para implementarlo con un sensor real. Los mensajes ZDA, en el caso de NMEA, y UBX-TIM-TOS, en el caso de UBX, se utilizarán para actualizar la fecha almacenada, para ser utilizada en caso de que algún mensaje la requiera. Los mensajes GGA, en el caso de NMEA, y UBX-NAV-PVT, en el caso de UBX, se utilizarán para obtener datos de posicionamiento. Estos últimos deben descartarse si, siendo válido el mensaje, el fix no lo es (la calidad del fix es inválida en NMEA o el bit *gnssFixOK* del registro *flags* del mensaje UBX-NAV-PVT no es 1).

Estos datos serán almacenados en un contenedor de tipo lista. La lista siempre contendrá como primer elemento el mensaje más antiguo.

Luego de cada carga de mensaje en la lista, se analizará si se procesan datos de la lista o no. Para tal fin, se generá un número aleatorio entero y ese número será utilizado como cantidad de mensajes a procesar. Al procesar los mensajes, se generarán los *Track Points* correspondientes al formato GPX.

4.4. Logs

El programa deberá utilizar un archivo de logs donde se almacenarán incidentes en el programa. Sin embargo los incidentes en general no serán terminantes y el programa deberá continuar. A continuación se dan algunos ejemplos de incidentes con diferentes niveles de “log”.

ERROR Son errores graves que ocurren durante la ejecución. Algunos ejemplos son:

- No se pudo abrir un archivo,
- Un checksum no concuerda,
- Un mensaje UBX debería tener un cierto largo y tiene otro,
- Un mensaje NMEA estaba mal formado, etc.

WARN Son avisos que ocurren durante la ejecución. Algunos ejemplos son:

- No se reconoce un ID (habrá muchos no reconocidos)
- Un mensaje contiene un fix inválido,
- Se descarta un mensaje por lista llena, etc.

DEBUG Son mensajes de información sobre lo que está haciendo el programa. Por ejemplo, buscando los 2 bytes de sincronismo, detecta un determinado ID, recolectó un mensaje, cargó un mensaje en la lista, imprimió un mensaje, etc.

5. Ejecutable

A continuación se describen los argumentos que debe recibir el programa. Esta descripción sirve para armar la ayuda que debe imprimir el programa.

-h, --help

Muestra una ayuda

-n `nombre`, **--name** `nombre`

Indica el *metadato* nombre (*name*).

-p `protocolo`, **--protocol** `protocolo`

Indica el protocolo a leer. `protocolo` puede tomar los valores `nmea` o `ubx`.

Opcional: además de las opciones `nmea` y `ubx`, agregar una tercera opción, `auto`. En esta tercera opción, el protocolo debe ser detectado por la aplicación en función de los mensajes que lee.

-i `archivo`, **--infile** `archivo`

Indica el nombre del archivo a utilizar como entrada de datos.

Opcional: utilice `-` como nombre de archivo para indicar la lectura desde `stdin`. Si no se ingresa el argumento, fundamente el accionar de su programa.

-o `archivo`, **--outfile** `archivo`

Indica el nombre del archivo a utilizar para el archivo gpx.

Opcional: utilice `-` como nombre de archivo para indicar la impresión de datos en `stdout`. Si no se ingresa el argumento, fundamente el accionar de su programa.

-l `archivo`, **--logfile** `archivo`

Indica el nombre del archivo a utilizar para el archivo log.

Opcional: utilice - como nombre de archivo para indicar la impresión de logs en `stderr`. Si no se ingresa el argumento, fundamente el accionar de su programa.

-m `cantidad`, **--maxlen** `cantidad`

Indica la máxima cantidad de mensajes que se pueden almacenar en una lista.

6. Testing

6.1. Fugas de memoria

Las fugas de memoria de la aplicación pueden ser advertidas utilizando la aplicación *valgrind*. Para que la misma indique dónde se producen fugas de memoria, es necesario compilar nuestra aplicación en modo debug.

Una vez compilada la aplicación, *valgrind* se ejecuta de la siguiente manera:

```
valgrind ./mi_aplicacion
valgrind ./mi_aplicacion argumento1 argumento2 ...
```

Al ejecutar correctamente el comando **y** si el programa no tiene fallas, se ve una leyenda similar a la siguiente:

```
==23390== HEAP SUMMARY:
==23390==     in use at exit: 0 bytes in 0 blocks
==23390==   total heap usage: 89,408 allocs, 89,408 frees, 5,057,256
    bytes allocated
==23390==
==23390== All heap blocks were freed -- no leaks are possible
==23390==
==23390== For counts of detected and suppressed errors, rerun with:
   -v
==23390== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0
   from 0)
```

Ejemplo 1: Resultado de ejecución de valgrind sin fugas de memoria

Si en cambio el programa tiene fugas de memoria, el mensaje es:

```
==25854== HEAP SUMMARY:
==25854==     in use at exit: 456,722 bytes in 3,932 blocks
==25854==   total heap usage: 89,408 allocs, 85,476 frees, 5,057,256
    bytes allocated
==25854==
==25854== LEAK SUMMARY:
==25854==    definitely lost: 456,722 bytes in 3,932 blocks
==25854==    indirectly lost: 0 bytes in 0 blocks
==25854==      possibly lost: 0 bytes in 0 blocks
==25854==    still reachable: 0 bytes in 0 blocks
==25854==         suppressed: 0 bytes in 0 blocks
==25854== Rerun with --leak-check=full to see details of leaked
   memory
```

```
==25854==
==25854== For counts of detected and suppressed errors, rerun with:
    -v
==25854== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0
    from 0)
```

Ejemplo 2: Resultado de ejecución de valgrind con fugas de memoria

7. Restricciones

La realización de los programas pedidos está sujeta a las siguientes restricciones:

- Debe realizarse en grupos de **3 (tres)** integrantes.
- No está permitida la utilización de `scanf()`, `gets()`[2], `fflush(stdin)`[3],la biblioteca `conio.h`[4] (`#include <conio.h>`), etc.
- Debe recurrirse a la utilización de funciones mediante una adecuada parametrización.
- Deben utilizarse punteros a función para el procesamiento de los mensajes.
- Debe utilizarse TDA para las listas, por lo menos.
- No está permitido en absoluto tener hard-codings:

```
1 ...
2 char s[282];
3 ...
4 if (s[1] == 'G' && s[2] == 'G' && s[3] == 'A') /*¡hard-coded!*/
5     printf("%s","xxxxxxx");          /* ¡¡hard-coded!! */
6 ...
7 if (!strcmp(argv[1], "--nombre"))   /* ¡¡hard-coded!! */
```

  sino que debe recurrirse al uso de `ETIQUETAS`, `CONSTANTES SIMBÓLICAS`, `MACROS`, etc.

  Los ejemplos no son exhaustivos, sino que existen otros hard-codings y tampoco son aceptados.

- Hay ciertas cuestiones que no han sido especificadas intencionalmente en este Requerimiento, para darle al/la desarrollador/a la libertad de elegir implementaciones que, según su criterio, resulten más convenientes en determinadas situaciones. Por lo tanto, se debe explicitar cada una de las decisiones adoptadas, y el o los fundamentos considerados para las mismas.

[2] obsoleta en C99 [3], eliminada en C11 [4] por fallas de seguridad en su uso.
[3] comportamiento indefinido para flujos de entrada ([3],[4]). Definida en estándar POSIX.
[4] biblioteca no estándar, con diferentes implementaciones y licencias, y no siempre disponible.

8. Entrega

Fecha límite de entrega: 6 de diciembre de 2018 .

La entrega en papel no es obligatoria, pero debe acordarse con quien corrija el trabajo si la exige o no. Sin embargo, es obligatorio realizar una entrega digital, a través del campus de la materia y por correo electrónico, de un único archivo cuyo nombre debe seguir el siguiente formato:

```
YYYYMMDD_apellido1-apellido2-apellido3-N.tar.gz
```

donde `YYYY` es el año (2018), `MM` el mes y `DD` el día en que uno de los integrantes sube el archivo, `apellido-1a3` son los apellidos de los integrantes ordenados **alfabéticamente**, `N` indica el número de vez que se envía el trabajo (`1`, `2`, etc.), y `.tar.gz` es la extensión, que no necesariamente es `.tar.gz`.

El archivo comprimido debe contener los siguientes elementos:

- La correspondiente documentación de desarrollo del TP (en formato pdf), siguiendo la numeración siguiente, incluyendo:
  1. Carátula del TP. Incluir una dirección de correo electrónico.
  2. Enunciado del TP.
  3. Estructura funcional de los programas desarrollados.
  4. Explicación de cada una de las alternativas consideradas y las estrategias adoptadas.
  5. Resultados de la ejecución (corridas) de los programas, captura de las pantallas, bajo condiciones normales e inesperadas de entrada.
  6. **Archivos de prueba utilizados.**
  7. Reseña sobre los problemas encontrados en el desarrollo de los programas y las soluciones implementadas para subsanarlos.
  8. Bibliografía (ver sección 9).
  9. Indicaciones sobre la compilación de lo entregado para generar la aplicación.
     NOTA: Si la compilación del código fuente presenta mensajes de aviso (warning), notas o errores, los mismos deben ser comentados en un apartado del informe.

  NOTA: El Informe deberá ser redactado en *correcto* idioma castellano.
- Códigos fuentes en formato de texto plano (.c y .h), *debidamente documentados*.

NOTA: Se debe generar y subir un único archivo (comprimido) con todos los elementos de la entrega digital. **NO usar RAR.** La compresión RAR no es un formato libre, en tanto sí se puede utilizar *ZIP*, *GUNZIP*, u otros (soportados, por ejemplo, por la aplicación de archivo *TAR*).

Si no se presenta cada uno de estos ítems, será rechazado el TP.

9. Bibliografía

Debe incluirse la referencia a toda bibliografía consultada para la realización del presente TP: libros, artículos, URLs, etc., citando:

- Denominación completa del material (Título, Autores, Edición, Volumen, etc.).
- Código ISBN si lo tiene.
- URL del sitio consultado. No poner Wikipedia.org o stackexchange.com, sino que debe incluírse un enlace al artículo, hilo, etc. consultado.

Utilizando LaTeX, la inclusión de citas/referencias es trivial. Los editores de texto gráficos de las suites de ofimática, como LibreOffice Write o MS Word, admiten plugins que facilitan la inclusión.

Ejemplo de referencias

[1] National Marine Electronics Association. *NMEA 0183 Interface Standard*. 0183. NMEA, 2008.

[2] u-blox. *u-blox 8 / u-blox M8 Receiver Description. Including Protocol Specification v15-20.30,22-23.01*. Manual UBX-13003221. u-blox, nov. de 2018, pág. 409. URL: https://www.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_(UBX-13003221)_Public.pdf.

[3] *GPS eXchange Format*. Open Standard/Public Domain 1.1. 2004. URL: http://www.topografix.com/gpx.asp.

[4] B.W. Kernighan y D.M. Ritchie. *The C Programming Language*. 2.ª ed. Prentice-Hall software series. Prentice Hall, 1988. ISBN: 9780131103627.

[5] P. Deitel y H. Deitel. *C How to Program*. 7.ª ed. Pearson Education, 2012. ISBN: 9780133061567.

[6] ISO/IEC. *Programming Languages – C*. ISO/IEC 9899:1999(E). ANSI, dic. de 1999, págs. 270-271.

[7] ISO/IEC. *Programming Languages – C*. INCITS/ISO/IEC 9899:2011. INCITS/ISO/IEC, 2012, pág. 305.

[8] Docentes 95.11/75.02 FI-UBA. *Procesador de datos de un sistema GNSS: Protocolo NMEA-0183 – Formato GPX*. CC BY-SA 4.0 1.0. 2018. URL: https://algoritmos9511.gitlab.io/tps/.