ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

DISEÑO E IMPLEMENTACIÓN DE UN DRIVER PARA RECEPTOR GPS CON ESPECIFICACIONES EN TIEMPO REAL

(Design and implementation of a driver for GPS reciever with real time specification)

Para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Autor: Luis Alberto Riancho Martín

Marzo – 2013

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Luis Alberto Riancho Martín
Director del PFC: Tomás Fernández Ibañez

Título: "Diseño e implementación de un driver para receptor GPS con especificaciones en tiempo real "
Title: " Design and implementation of a driver for GPS reciever with real time specification"

Presentado a examen el día: 18 de Marzo de 2013

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Valle López, Luis
Secretario (Apellidos, Nombre): Fernández Ibáñez, Tomás
Vocal (Apellidos, Nombre): Tazón Puente, Antonio

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente Fdo.: El Secretario

Fdo.: El Vocal Fdo.: El Director del PFC

Vº Bº del Subdirector Proyecto Fin de Carrera Nº

ÍNDICE

1 INTRODUCCIÓN

Todo este trabajo ha sido realizado por el autor de este documento en el Centro Tecnológico de Componentes (CTC) durante un período de prácticas en el verano de 2011.

Los objetivos establecidos durante ese período fueron:

- Adquisición de conocimientos sobre el funcionamiento del sistema GPS
- Refuerzo de conocimientos sobre programación en lenguaje C
- Realización de documentación técnica
- Diseño del driver para el receptor GPS con especificaciones en tiempo real
- Diseño del driver para facilitar el manejo de la mesa de rotación para la colocación de las antenas receptoras

Este documento intentará reflejar la base teórica que se ha seguido a la hora de realizar el controlador del receptor GPS de Septentrio, explicando tanto teoría de GPS como la teoría de programación seguida para el propio controlador. Así mismo, quedará reflejado el diseño e implementación del driver tanto para el propio receptor GPS como para la mesa de rotación utilizada como banco de pruebas para las antenas receptoras.

1.1 Idea principal del sistema GPS

La idea principal del cálculo de la posición mediante el rastreo de satélites GPS, es que, a partir de la posición conocida de 3 satélites (calculada a partir de las efemérides enviadas por cada satélite) y las distancias de cada uno de ellos al receptor (que se obtendrán gracias al tiempo de propagación que existe entre el satélite y el receptor, multiplicándolo por la velocidad de la luz). Con esos datos se crean 3 esferas, cada una centrada en un satélite y de radio la distancia al receptor, obteniendo la posición como la intersección de las 3 esferas.

Sin embargo, debido a diferentes efectos que sufren las señales enviadas desde el satélite hasta que llegan al receptor (Ionosfera, Troposfera, fallos de sincronización entre el reloj del satélite y el del receptor, efectos multipath…), el resultado obtenido no es exacto, por lo que han de aplicarse una serie de correcciones en el receptor que aporten una mayor validez a la posición obtenida.

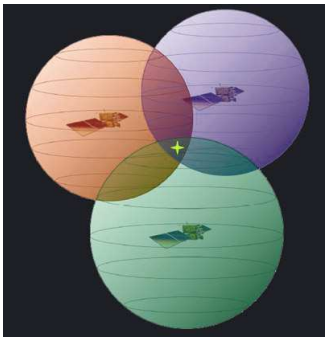

Figura 1. Determinación de la posición a partir de 3 satélites

1.2 Bloques que conforman el sistema GPS

El sistema GPS está formado por 3 grandes bloques, el segmento espacial, el segmento de control y el segmento de usuario.

1.2.1 Segmento espacial

Esta parte sigue las instrucciones proporcionadas por el segmento de control, y ha de proporcionar una referencia de tiempo atómico, además de generar las señales de RF pseudoaleatorias y almacenar y reenviar el mensaje de navegación.

Así mismo, el segmento espacial consta de: la constelación, los satélites y la señal GPS.

La constelación

Consta de al menos 24 satélites, distribuidos en 6 planos orbitales con una inclinación de 55 grados respecto al ecuador. Las órbitas son elípticas, pero prácticamente pueden aproximarse a órbitas circulares debido a su baja excentricidad. El período orbital es de aproximadamente 12 horas, lo que puede resultar bastante ventajoso a la hora de realizar medidas durante varios días a la misma hora y en situaciones similares para posteriormente compararlas.

La distribución mencionada permite que siempre haya, al menos, 4 satélites visibles desde cualquier punto del planeta, siempre y cuando el ángulo de elevación sea superior a 15 grados.

Los satélites

Los satélites han de ser capaces de mantenerse continuamente en órbita, comunicarse con el segmento de control y emitir las señales pertinentes a los receptores. Una parte crítica de los satélites es el reloj interno del satélite, el cuál ha de tener una estabilidad enorme. Por ello se usan relojes atómicos, como los de Cesio, que se desvían un segundo cada 300.000 años.

Además cada satélite se puede identificar de diferentes maneras: por su posición en el plano orbital, por el número de catalogación de la NASA, por el número internacional de identificación, por el código PRN (código pseudoaleatorio de ruido) o por el número de secuencia de lanzamiento (SVN).

La señal GPS

Cada satélite transmite en dos frecuencias (L1 y L2), derivadas de la frecuencia fundamental de su reloj interno, $f_0 = 10.23MHz$, siendo:

$$L_1 = 154 \cdot 10.23MHz = 1575.42MHz$$
$$L_2 = 120 \cdot 10.23MHz = 1227.60MHz$$

Gracias a que el satélite emita en esas dos frecuencias, permite al receptor cancelar uno de los errores principales, la refracción debida a la Ionosfera.

Ya sea en una portadora, o en ambas, se modulan los siguientes códigos PRN y mensajes:

- Coarse/Acquisition code (C/A), o también código civil. Secuencia repetida cada 1 ms, a una velocidad de 1Mbps. Esta secuencia se modula sólo en L1.

- Precision Code (P), es un código reservado para el uso militar o para civiles autorizados, y se modula tanto en L1 como en L2.

- Mensaje de navegación, que se modula sobre ambas portadoras a 50bps. Contiene información de las efemérides del satélite, así como otras informaciones importantes para correcciones que han de ser tenidas en cuenta en el receptor, etc.

Además, para restringir el uso en caso de amenaza a al información de los satélites, se creó una técnica de degradación del reloj del satélite y de las efemérides, la S/A (Selective Availability), que puede desembocar en un aumento del error en la posición que puede alcanzar los 100 metros. Es por ello que otros países o comunidades hayan creado sus propios sistemas de navegación, como el GLONASS ruso.

1.2.2 El segmento de control

Es el bloque encargado del funcionamiento del sistema GPS. Sus principales funciones son el control y el mantenimiento de la constelación de satélites, predecir las efemérides y el comportamiento de los relojes de los satélites, mantener la escala de tiempos GPS mediante relojes atómicos y actualizar el mensaje de navegación de cada satélite de manera periódica.

También se encarga de activar, si fuera necesario, la técnica S/A. El segmento de control está formado por 5 estaciones de seguimiento o de monitorización, situadas en Hawaii, Colorado Springs, Isla de Ascensión, Diego García y en la Isla de Kwajalein, además de una estación central o maestra situada también en Colorado Springs. Además se cuenta con 3 antenas de transmisión a los satélites en Ascensión, Diego García y Kwajalein.

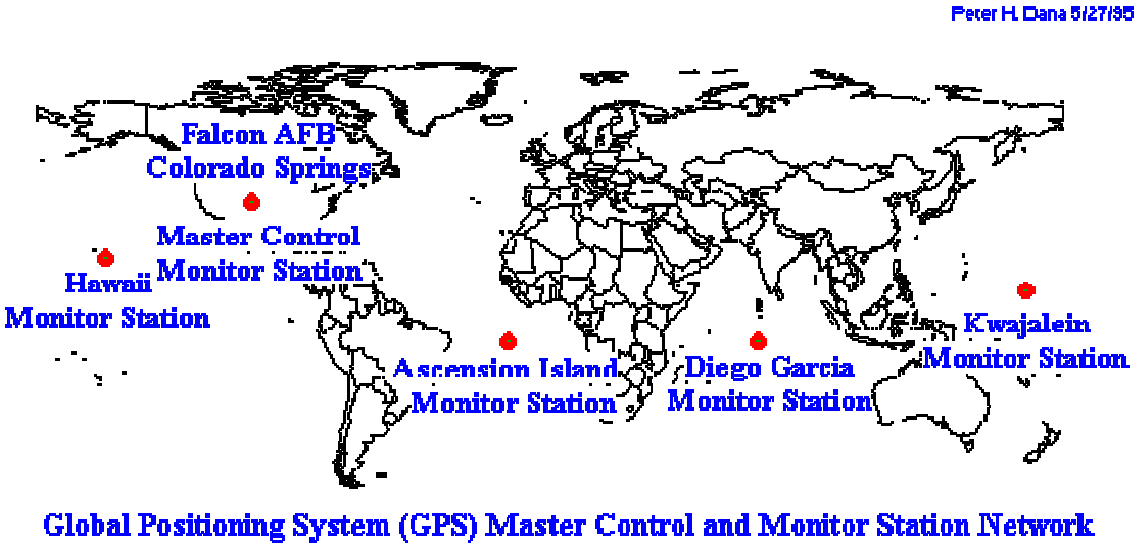

Figura 2. Distribución de las estaciones en la Tierra.

Su funcionamiento es el siguiente, las estaciones de monitorización reciben la información de los satélites que tienen a la vista, y se la reenvían a la estación maestra. Ésta procesa la información y estima una serie de parámetros concernientes a la órbita de los satélites o a

los errores en el reloj de los satélites, entre otras cosas. Las correcciones calculadas son enviadas a las antenas de transmisión y a su vez se reenvían a los satélites en banda S (de 2 a 4 GHz) para que éstos corrijan sus efemérides y demás parámetros. Este "refresco" de información puede ser realizado hasta 3 veces al día (cada 8 horas), aunque lo normal es realizarlo cada 24 horas.

1.2.3 El segmento de usuario

Este segmento lo conforman los receptores GPS. Su principal función es la de recibir la señal procedente de los satélites, determinar las pseudodistancias y resolver las ecuaciones de navegación para obtener sus coordenadas y proporcionar un tiempo muy preciso.

El mensaje de navegación

Cada satélite, como ya se ha mencionado, recibe información procedente de tierra con correcciones sobre sus parámetros orbitales y otros estados temporales. Esta información es reenviada por el satélite a los usuarios a través del mensaje de navegación.

El mensaje de navegación consta de 25 tramas, y se modula sobre las portadoras L1 y L2 a 50 bps. Cada una de las 25 tramas consta, a su vez, de 5 subtramas, formadas por 10 palabras de 30 bits. De manera que cada una de las subtramas se tarda en enviar 6 segundos, cada trama 30 segundos y, finalmente, el mensaje de navegación se envía cada 12.5 minutos.

Cada trama empieza con la palabra de telemetría (TLM), necesaria para la sincronización, seguida de la palabra de transferencia (HOW) encargada de una rápida conmutación del código C/A al código P.

1.3 Tiempo

Como ya se ha comentado, el tiempo es una cuestión fundamental para el cálculo correcto de la posición, por ello es muy importante establecer una referencia temporal periódica. En este caso existen varias referencias periódicas (fenómenos de rotación de la Tierra, mecánica celeste o transiciones entre niveles de energía de los osciladores atómicos), lo que desemboca en diferentes sistemas de tiempo.

Los más importantes son:

Fenómeno periódico	**Tiempo**
Rotación de la Tierra	Tiempo Universal (UT0, UT1, UT2)
	Tiempo sidéreo
Revolución de la Tierra	Tiempo Dinámico Terrestre (TDT)
	Tiempo Dinámico Baricéntrico (BDT)

Osciladores atómicos	Tiempo Atómico Internacional (IAT)
	Tiempo Universal Coordinado (UTC)
	Tiempo GPS (GPST)

Figura 3. Referencias temporales

1.4 Sistemas de referencia

Al igual que la referencia temporal, es muy importante establecer un sistema de referencia adecuado para establecer un correcto posicionamiento. Hay varios sistemas de referencia (el CIS, CTS...) pero nos centraremos en el sistema WGS-84 (World Geodetic System), que es un sistema desarrollado por el Departamento de Defensa de los EEUU y que desde 1987 es utilizado por GPS.

1.4.1 El sistema WGS-84

Este sistema está asociado a un elipsoide de parámetros:

Semieje mayor de la elipse	a	6378.137 Km
Semieje menor de la elipse	b	6356.752 Km
Factor de achatamiento	f	1/298.257223563
Velocidad angular Tierra	ω_E	$7292115 \cdot 10^{-11}$ rad/s
Constante de gravitación	μ	$3986005 \cdot 10^{8}\ m^3/s^2$

Siendo las coordenadas (λ,φ,h), las correspondientes a la longitud y latitud elipsoidales y la altura sobre el elipsoide respectivamente.

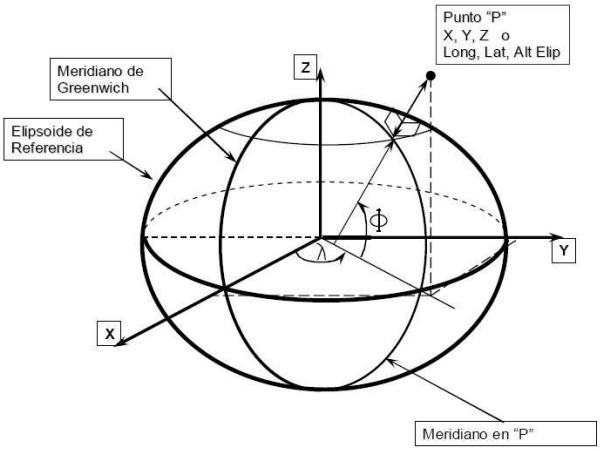

Figura 3. Sistema WGS-84

1.4.2 Cálculo de la pseudodistancia

El observable básico en GPS es el retardo (dT), es decir, el tiempo que tarda en viajar la señal desde que es emitida desde la antena del satélite hasta que es recibida por la antena del receptor. Si escalamos este valor con la velocidad de la luz (c), obtendremos la distancia aparente o pseudodistancia entre el satélite y el receptor.

$$D = c \cdot dT$$

Para calcular el término dT, hemos de correlar el código, ya sea el P o el C/A recibido del satélite con una réplica generada en el receptor, de tal manera que la réplica tiene que desplazarse un intervalo de tiempo (correspondiente a dT) hasta conseguir la máxima correlación.

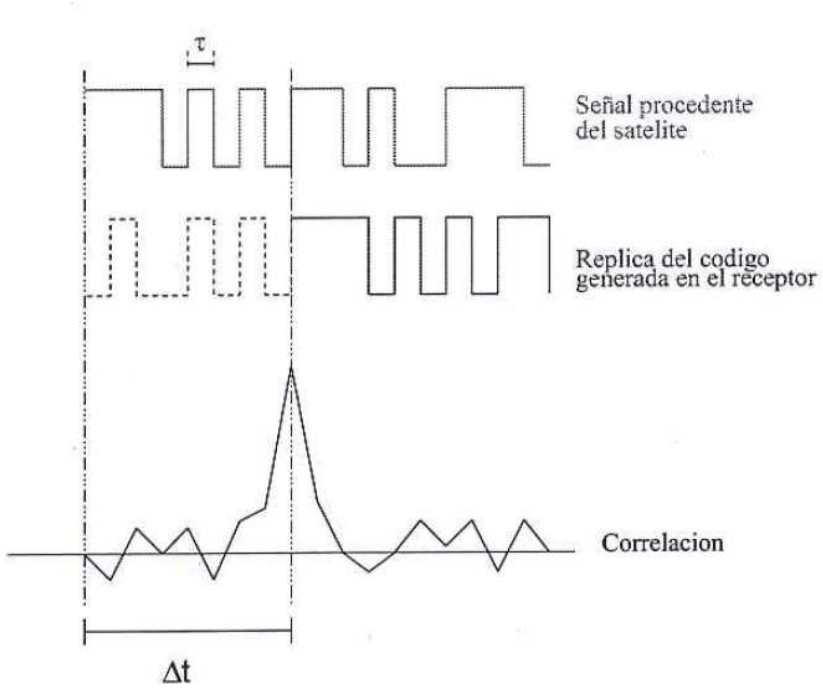

Figura 4. Correlación de la señal recibida con la réplica.

1.5 Retardos que afectan al cálculo de la pseudodistancia:

1.5.1 Retardo troposférico

A la frecuencia en que trabajan las señales de GPS, la Troposfera se comporta como un medio no dispersivo, es decir, que el efecto que provoca es independiente de la frecuencia a la que se trabaja. Puede modelarse de forma aproximada, casi al 90%, lo que permite aportar una buena corrección al cálculo final.

1.5.2 Retardo ionosférico

La Ionosfera es la zona de la atmósfera que se extiende desde 60 Km hasta más de 2000 Km de altura. En ese tramo, las señales electromagnéticas entran en contacto con los electrones libres presentes, sufriendo retrasos y adelantos continuamente que provocan errores en la determinación del tiempo total. Dado que la Ionosfera es un medio dispersivo, y por lo tanto, depende de la frecuencia de trabajo, el uso de 2 frecuencias diferentes (L1 y L2) nos permite corregir los retardos sufridos en esta capa. Si el receptor sólo dispone de

una sola frecuencia de trabajo, entonces han de utilizarse modelos de predicción, como el modelo Klobuchar.

1.5.3 Retardos instrumentales

Dependiendo de la calidad de los equipos utilizados en la comunicación (antenas, cables, filtros…) obtendremos mayor o menor retardo adicional. Evidentemente, cuanto mejores sean nuestros equipos, menos retardo instrumental tendremos.

1.5.4 Efectos multicamino (multipath)

Las señales pueden llegar a las antenas de dos maneras, directamente desde el satélite sin ser interferidas por ningún objeto o por medio de reflexiones en estructuras reflectantes. Ello provoca que la misma señal provenga de diferentes fuentes a diferentes tiempos, provocando errores en la medida de la fase y en la de los códigos.

El método más eficiente para evitar este efecto es contar con antenas que rechacen señales provenientes de determinadas direcciones, así como colocar la propia antena en una zona alejada de elementos reflectantes.

1.5.5 Ruido

Incluye el ruido de medida de la pseudodistancia y todos los efectos que no pueden ser modelados.

1.6 Método de cálculo de los parámetros de orientación basado en GPS y ayudado por sensores inerciales de bajo coste.

1.6.1 El método

Primero comentaremos el método basado sólo en la ayuda GPS, que se basa en el concepto de diferencia de fases.

Concretamente, este método está “testeado” para 4 antenas, aunque se modificará para hacerlo con 3.

Cada satélite modula en fase dos señales, una que es el código C/A y la otra que son los datos de navegación del satélite, en una portadora de 1575.42 MHz (L1). Cada satélite presenta un código C/A propio, lo que permitirá separar las señales de los diferentes satélites.

El método utiliza una frecuencia de referencia, obtenida a partir del oscilador que controla el receptor. El propio receptor correla la señal recibida con los diferentes códigos de los satélites hasta obtener el correcto.

Cuando la señal llega a la antena, habrá recorrido una distancia D, correspondiente a un número n de longitudes de onda, llamado ambigüedad, además de una porción de longitud de onda, es decir, el desfase, que estará comprendido entre 0 y 2π y el cuál podemos medir.

La diferencia de fase se puede calcular como:

$$\Delta\varphi = \frac{2\pi}{\lambda} l \cos(\theta)$$

En términos de longitud:

$$\Delta\rho = l\cos(\theta)$$

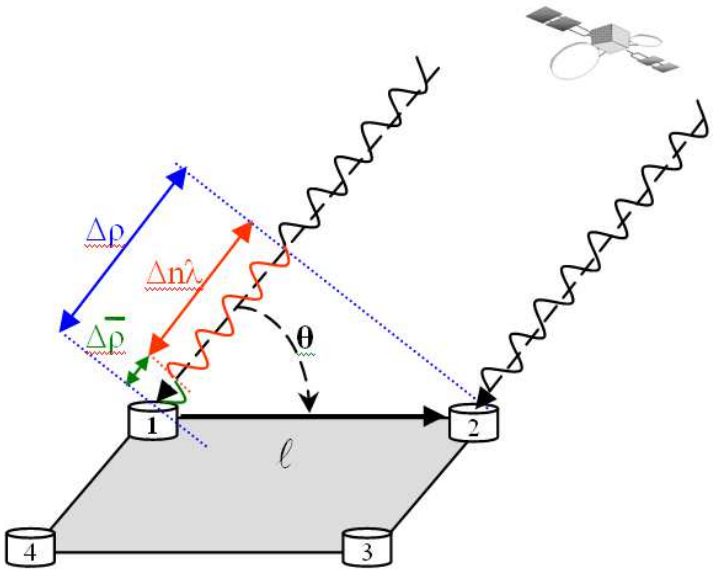

Figura 5. Diferencia de fase entre dos receptores

$\Delta\overline{\rho}$ será la fase de la señal que podemos medir, mientras que n, será el parámetro que tendremos que estimar para finalmente encontrar la solución correspondiente a $\Delta\rho$.

Para estimar n, es necesario calcular las coordenadas de las líneas base en el sistema local.

Después, los parámetros "attitude" del vehículo pueden ser estimados mediante la rotación del cuerpo respecto al sistema local.

Sin embargo se nos presenta una dificultad a la hora de estimar las ambigüedades, la presencia de errores de fase en la medida, debido a diferentes factores, como pueden ser los inducidos por la Ionosfera y la Troposfera, los debidos al reloj del receptor o incluso los provocados por las efemérides del satélite. Para reducir dicho error, realizaremos una diferencia de fases para eliminar algunos de esos errores, ya que se considerarán iguales y al hacer la diferencia se eliminarán.

Teniendo finalmente:

$$\nabla\Delta\varphi = \nabla\Delta\overline{\rho} + \lambda\nabla\Delta n + \nabla\Delta d\varphi$$

Dónde sólo queda el error debido a multipath, ruido térmico y los errores intercanal del receptor.

Ahora el objetivo es conseguir, en el menor tiempo posible y con la mínima carga computacional, la solución correcta de las ambigüedades.

Además, el método de resolución ha de cumplir una serie de características, que son las siguientes:

1. Conseguir resultados en un solo período de tiempo determinado (Single Epoch)

2. Selección consistente de las ambigüedades correctas

3. Habilidad para desechar las ambigüedades incorrectas

4. Poder partir de unos parámetros de "attitude" desconocidos

El método está basado en una ortogonalización Gram-Schmidt, que consigue reducir en una dimensión (de 2 a 3) el espacio de búsqueda de la solución, lo que minimiza mucho la cantidad de posibles resultados.

Como ventajas del método tenemos que:

1. Está basado en una técnica de doble diferencia

2. Es muy eficiente computacionalmente

3. Realiza el cálculo en un solo período determinado

4. Puede trabajar sin conocer los parámetros iniciales de "attitude"

Sin embargo su principal desventaja es que fue diseñado para trabajar con satélites LEO, de órbitas bajas.

Por ello se contará con la ayuda de sensores inerciales o giróscopos de bajo coste, que nos ayudarán a obtener una solución mucho más fiable.

Partimos con la ventaja de que conocemos las distancias de las líneas base entre antenas, con lo que podemos calcular el error que se produce, en términos de distancias, como:

$$\varepsilon_i = \left| \|b_i\|^2 - \ell_i^2 \right| \qquad i = 1..3$$

Del número total de posibles soluciones, se escogerán las N que tengan mínimo error. Las cuáles se combinarán generando una lista de N^3 posibles soluciones.

Esa lista se organizará usando la dependencia entre líneas base ya conocida de la siguiente manera:

$$\varepsilon_{TOT} = \sum_{i=1}^{3} \left| \|b_i\|^2 - \|I_i\|^2 \right| + \sum_{\substack{i=1 \\ j=1}}^{3} \left| \|\Delta b_{ij}\|^2 - \|\Delta I_{ij}\|^2 \right|$$

Que, en ausencia de errores ni ruido, nos aporta la solución correcta al problema. Sin embargo, dado que en la práctica las condiciones no son ideales, la salida del método nos dará una lista de posibles soluciones ordenadas por probabilidad decreciente.

Para mejorar el método se incluyó el uso de los sensores inerciales, que ayudan en gran medida a establecer la solución correcta. Además contamos con un filtro de Kalman, que, a partir de 2 entradas, una que tomará por "buena" y otra que tomará como estimada, ofrece a la salida los parámetros de "heading", "pitch" y "roll" que requerimos para determinar la estima de las líneas base de las antenas.

Al filtro Kalman le introduciremos como entrada "buena" las estimas generadas a partir del GPS y como entrada estimada las calculadas por los sensores inerciales. Por último, el espacio de búsqueda quedará reducido por medio de un umbral adaptativo que depende de

factores como el tiempo de integración desde la última medida recibida, la dinámica actual del vehículo, etc.

1.6.2 Los Cuaterniones

Haremos una mención especial a los cuaterniones por su utilidad de cara a las rotaciones necesarias para calcular los parámetros de orientación del vehículo.

Podría describirse a los cuaterniones como una extensión a los números reales, similar a los números imaginarios, ya que se incluyen unidades imaginarias. En este caso cuenta con una parte real (o escalar) y tres extensiones imaginarias i, j y k. Luego se puede hablar de que los cuaterniones manejan 4 dimensiones. Todo ello es muy útil para realizar rotaciones, de un sistema de referencia a otro y, además, evita una serie de singularidades que pueden ocurrir en el caso de utilizar otro sistema como el de los ángulos de Euler, además de reducir bastante la carga computacional asociada a los cálculos de dichas rotaciones.

Definiremos un cuaternión genérico de la siguiente forma:

$$q = a + a_1 i + a_2 j + a_3 k$$

Todo punto del espacio de 3 dimensiones puede representarse como un cuaternión de parte real 0.

Rotaciones con cuaterniones:

Una de las características más importantes de los cuaterniones es la sencillez que aporta a la realización de rotaciones. De manera que a1, a2 y a3 serán las componentes de cualquier eje arbitrario y a será el ángulo de rotación deseado.

Las diferentes operaciones que hay que realizar para el cálculo de dichas rotaciones son transparentes desde nuestro punto de vista, ya que podemos aprovechar diferentes librerías existentes que ofrecen a las salidas que nos interesan.

2 PROGRAMACIÓN DEL DRIVER

Antes de comenzar con el driver en sí, vamos a explicar una serie de aspectos a tener en cuenta a la hora de programar que resultarán críticos para un correcto funcionamiento, tanto en términos de tiempo como de eficiencia.

2.1 Aspectos generales de programación en tiempo real

Para una correcta programación, hemos de tener en cuenta los siguientes conceptos, que trataremos de evitar:

- "Responsivenes": No satisfacer las espectativas del usuario
- "Reliability": Programa que presenta fallos y es de difícil depuración
- "Cost": El coste es difícil de evaluar y más caro de lo esperado
- "Modificability": Los programas son productos muy rígidos y difíciles de mantener
- "Timeless": Se requiero para la ejecución del programa más tiempo del previsto
- "Transoportability": La migración a diferentes plataformas puede generar problemas
- "Efficiency": El programa sólo usa una parte de la capacidad del hardware.

Distinguiremos 3 tipos de programación: secuencial, concurrente y en tiempo real.

2.2 Programación secuencial

Es el tipo de programación más sencillo de controlar, ya que las sentencias ejecutadas se ejecutan según el orden en el que han sido programadas, es decir, tiene una única línea de flujo de control. El tiempo que tarde en ejecutarse cada sentencia no influirá en el resultado final, cosa que no ocurrirá en el resto de casos. Es muy fácil de verificar ya que las sentencias se ejecutan en orden y se puede comprobar que todas aporten el resultado correcto correspondiente.

2.3 Programación concurrente

En este caso, los programas tienen varias líneas de control establecidas. El programa se compone de un conjunto de procesos que colaboran y compiten entre sí. El resultado ha de ser independiente de los tiempos de ejecución de las sentencias utilizadas. Además, el orden de ejecución de las sentencias no es estricto, si no que se irán utilizando diferentes métodos de planificación que actualicen el orden de ejecución estableciendo una serie de prioridades, diferentes según el método utilizado.

2.4 Programación en tiempo real

Aquí, el orden de ejecución de las tareas dependerá de 2 aspectos, el orden establecido por las líneas de flujo de control y los eventos externos. El cumplimiento de los plazos temporales es una parte de la especificación funcional. Además, puede haber dos tipos de sistemas en tiempo real, los "Hard Real Time" en los que el incumplimiento del plazo temporal es un fallo irrecuperable, y los "Soft Real Time" en los que los requerimientos temporales se cumplen en promedio.

2.5 Recursos para la programación concurrente:

Se puede definir un programa concurrente como un conjunto de threads o hilos de control, en los que cada uno de esos threads ejecuta una única actividad secuencial y además, cada uno de ellos lo hace en un procesador virtual independiente. Dichos threads intercambian entre sí mensajes de información y también de sincronización. También hay que tener en cuenta que, crear, instanciar y destruir un proceso implica un conjunto de tareas bastante complejas y también costosas, y además, el mantener dicho proceso en funcionamiento exige un uso de recursos que tendrá que ser gestionado.

2.5.1 Procesos estáticos

Son aquellos procesos instanciados al comienzo del programa, y son creados por el programador explícitamente.

2.5.2 Procesos dinámicos

Son creados durante la ejecución del programa y en función de los datos. A su vez, pueden ser sistemas cerrados o abiertos. En los sistemas cerrados se conocen los procesos que se crean en su totalidad, mientras que en los sistemas abiertos, se crean procesos de forma impredecible.

2.5.3 Estructuras de procesos

Podemos distinguir entre dos tipos de procesos en cuanto a su estructura, los de estructura plana y los de estructura jerarquizada. En cuanto a los primeros tenemos que todos los procesos son equivalentes, y además evolucionan de manera totalmente independiente. Los segundos, mantienen una dependencia entre procesos, de manera que, un proceso no termina hasta que los otros procesos que dependen de él no lo hacen.

2.5.4 Inicialización de un proceso

La inicialización de un proceso consiste en transferirle la información necesaria para caracetizarlo. Para ello existen diferentes formas:

- Unix: Cada proceso es inicialmente una réplica del proceso “padre”
- Ada / Java: Se le pasa al proceso unos parámetros de inicialización en su creación
- Tras la creación del proceso, el hijo se comunica con su padre para recibir la información de inicialización

2.5.5 Finalización de un proceso

También es importante acabar los procesos y también hay diferentes formas de que eso ocurra:

- Finalizado correctamente (éxito)
- Excepción no atendida (fallo)
- Sentencia “sel terminate” (fallo)
- Abortado por otro proceso (fallo)
- Finalización coordinada (éxito)
- No termina nunca

2.6 Políticas de planificación

Como ya se ha comentado anteriormente, es necesario seguir una serie de políticas para planificar el acceso de los procesos a los recursos. Por ello se crean una serie de criterios para seleccionar el proceso (activo) que pasará a tomar el control del recurso (o recursos) en cuestión. Además, dichas políticas de planificación no deben influir en el funcionamiento del programa, aunque sí en el tiempo de ejecución, por ello es crítico seleccionar la política que mejor se ajuste a nuestro programa para que se ejecute en el menor tiempo posible.

2.7 Interacción entre procesos

Los procesos interactúan entre sí aunque de diferentes maneras:

- Independientes entre sí: Sólo interfieren por compartir el procesador
- Cooperan entre sí: Uno genera una información o servicio que va a ser utilizado por el otro proceso
- Compiten entre sí: Requieren el uso de una serie de recursos comunes en régimen compartido

2.7.1 Problemas asociados a la interacción de procesos

Al interaccionar entre ellos, los procesos intercambian información, o incluso comparten variables, y es muy importante que dichas variables que son modificadas continuamente, puedan ser accedidas en el momento correcto para no producir resultados indeseados. Es ahí donde surge el concepto de sección crítica de un programa, que será aquella a la que sólo un proceso podrá acceder cada vez, y el resto tendrá que esperar (en un orden establecido por la política de planificación) a que el proceso que accedió a dicha variable termine de modificarla. Por ello también es crítica la correcta sincronización entre procesos. Por todo ello, se crearon una serie de primitivas para facilitar la interacción de procesos y hacerlo de fomrma sencilla y eficiente. Estas primitivas deben hacer posible la sincronización, la exclusión mutua y la espera limitada. Existen dos modelos, el de intercambio de mensajes y el de memoria compartida. Este último será el que utilizaremos y por lo tanto el que se comentará.

2.8 Problemas de los programas concurrentes

2.8.1 Propiedades de seguridad

Es importante (y trivial) que debe impedirse, en la medida que sea posible, que no se ejecuten partes que puedan inducir a errores. Por ello se tienen en cuenta los siguientes conceptos a utilizar:

- Sección crítica en caso de que haya varios procesos
- Procesos han de respetar los puntos de sincronismo
- Impedir que uno o varios procesos permanezcan a la espera de un evento que no ocurrirá (Interbloqueo)

2.8.2 Propiedades de vivacidad

También es muy importante que las sentencias que se ejecuten en el programa conduzcan inequívocamente al objetivo final que tenga nuestro programa, por tanto hay que impedir que se produzcan las siguientes situaciones:

- Bloqueos activos, es decir, que 2 procesos estén ejecutando sentencias que no hacen avanzar al programa
- Aplazamiento definitivo, es decir, que el programa se quede sin tiempo de procesador para avanzar.
- Interbloqueo

2.9 Memoria compartida

Ésta es la vía habitual para compartir variables globales o secciones de memoria compartida. Para ello disponemos de diferentes mecanismos, los semáforos, las secciones críticas y los monitores. Los semáforos se encargan de arbitrar el acceso a un recuros (nivel bajo de abstracción). Por su parte, las secciones críticas, permiten la ejecución de un bloque de sentencias de forma segura (nivel medio de abstracción). Mientras que, por último, los monitores gestionan los recursos que van a ser utilizados de manera concurrente (nivel alto de abstracción).

2.9.1 Semáforos

Dado que son un tipo de datos, se definen por:

- Conjunto de valores que se le pueden asignar
- Conjunto de operaciones que pueden realizar

Llevan asociados una lista de procesos en la que se incluyen los procesos suspendidos a la espera de un cambio en el estado del semáforo. En esta lista se establecerá un orden según la planificación escogida.

Hay dos tipos, binarios (que pueden valer 0 o 1) y contadores (que pueden valer cualquier número natural). Siempre que el valor de un semáforo sea 0, es que está cerrado. Mientras que si el valor es mayor que cero, implica que el semáforo está abierto.
Con los semáforos se pueden realizar operaciones seguras o no seguras. Las operaciones seguras son:

- Wait(p):
  - Semáforo abierto: Decrementa en 1 su valor
  - Semáforo cerrado: El thread que lo ejecutó se suspende y se pone en la lista de procesos del semáforo
- Signal(p):
  - Si hay procesos en la lista de procesos activa uno de ellos y ejecuta lo que sigue a la sentencia Wait(p)
  - Si no hay procesos en la lista de procesos, se incrementa el valor del contador en 1

Además también existe una operación no segura, initial(p,Valor_inicial), que significa asignar al semáforo “p” un “Valor_inicial”.

Las ventajas del uso de semáforos son las siguientes:

- Resuelven todos los problemas de concurrencia
- Son estructuras pasivas muy simples
- Son fáciles de comprender

- Son eficientes

Mientras que sus desventajas son:

- Son de muy bajo nivle, lo que implica que un pequeño fallo puede inducir a un bloqueo
- La gestión del semáforo ha de realizarse a lo largo de todo el código. Ello hace que la depuración de errores sea muy complicada.

2.9.2 Secciones críticas

Son bloques de código a los que el compilador introduce mecanismos de sincronicación para que su ejecución se realice mediante exclusión mutua, es decir, que sólo un proceso pueda ejecutar esa parte del código a la vez. Se definen para que las actualizaciones de los valores de una variable compartida se realicen de forma segura, y se asegure la no aparición deresultados indeaseados.

Su ventaja frente a los semáforos es que proporcionan un nivel de abstracción mucho mayor, por lo que son menos proclives al error. Sin embargo presentan problemas, ya que su mantenimiento es bastante complicado, la integridad de las variables puede quedar comprometida y no son fáciles de implementar.

2.9.3 Monitores

Los monitores consisten en encerrar recursos o variables compartidas como componentes internos o privados y ofrecen una interfaz de exclusión mutua garantizada.

Para declararlos, hemos de declarar las constantens, variables, procedimientos y funciones privadas del monitor. También los procedimientos y funciones que el monitor exporta. Además hay que inicializar el monitor, lo que implica inicializar las variables y estructuras internas.

Las estructuras internas de datos del monitor cuya finalidad es ser compartidas por varios procesos de forma concurrente, sólo pueden ser inicializadas, leídas y actualizadas por el código propio del monitor. Además, los únicos componentes “públicos” del monitor son los procedimientos y las funciones que fueron exportadas. Los monitores aseguran la exclusión mutua así como un fácil mantenimiento.

Sin embargo, aunque los monitores aseguran el acceso seguro a los recursos compartidos, no tiene capacidades de sincronización. Por ello surge el concepto de las “variables condition”.

2.9.3.1 Variables Condition

No toman un valor determinado, sino que tienen asociada una lista de procesos suspendidos. Con ellos se pueden realizar tres operaciones, delay (para suspender un proceso e incluirlo en la lista asociada), resume (para reactivar un proceso de una lista) y empty (si toma el valor True, es que la lista de procesos está vacía).

2.10 El estándar POSIX

Para poder entender el código del "driver" con el que se va a trabajar, primero hay que entender los diferentes aspectos del estándar que seguiremos, así como el sistema operativo en tiempo real MaRTE OS sobre el que se manejará la aplicación.

El estándar POSIX (Portable Operative System Interface) basado en UniX, surge como idea para resolver problemas de portabilidad que tenían lugar en utilidades o aplicaciones a nivel de código fuente.

A su vez, el sistema define:

- Funciones, tipos y constantes agrupadas en ficheros de cabeceras
- Un intérprete de comandos o "shell"
- Programas de utilidad

Por otra parte, por motivos prácticos, surgió también la necesidad de un sistema en tiempo real, y actualmente tenemos 4 perfiles de sistema diferentes, los sistemas en tiempo real mínimos, los controladores de tiempo real, los sistemas en tiempo real dedicado y los sistemas en tiempo real multipropósito.

Sistema de tiempo real mínimo:

- Son sistemas empotrados pequeños, sin MMU (Unidad de Gestión de Memoria), sin disco y sin terminal
- Un buen ejemplo puede ser un tostador

Controlador en tiempo real:

- Es un controlador de propósito especial, que no tiene Mmu pero sí disco con un sistema de ficheros simplificado
- Por ejemplo, un robot industrial

Sistema en tiempo real dedicado:

- Es un sistema empotrado grade y sin disco, pero con MMU, incluso puede llegar a tener un sistema de memoria secundaria tipo flash
- Se trata de un software bastante complejo, requiere protección de memoria y comunicaciones
- Por ejemplo, un avión o una célula de un sistema de telefonía móvil

Sistema en tiempo real multipropósito:

- Es un sistema en tiempo real grande, con todas las facilidades
- Por ejemplo, sistemas de control aéreo, sistemas de telemetría de F1

2.10.1 Generalidades

El estándar POSIX está constituido por un conjunto de funciones, tipos y constantes en C, agrupadas en diferentes ficheros de cabecera, como por ejemplo:

- <string.h> que contiene las operaciones con strings
- <time.h> que contiene operaciones de tiempo, relojes, temporizadores…
- <pthread.h> con todo lo relacionado con los threads

Los identificadores del estándar pueden seguir dos convenios diferentes:

- Identificadores heredados de Unix o de C
  - Tienen nombres cortos (kill (), malloc ())
- Identificadores introducidos durante el desarrollo del estándar:
  - Funciones de tipo servicio_accion ()
  - O del tipo servicio_objeto_accion ()
  - Tipos de datos, finalizados en "_t"
- Constantes, siempre en mayúsculas y comenzando por el nombre del servicio

2.10.2 Códigos de error

Las funciones POSIX informan de los errores que se puedan producir por medio de un código numérico. Además en la cabecera <errno.h> están definidas una serie de constantes que identifican los tipos de errores que se producen, como por ejemplo:

- EACCES: permiso denegado
- EINVAL: argumento inválido

2.10.3 Detección de errores

Existen dos formas de detección de errores, dependiendo de si la función en cuestión es "antigua" o "moderna", ya que, en el primero de los casos, que abarca aquellas funciones anteriores al estánder "threads", devuelven el valor "-1" en caso de error, y el código de error se encuentra en la variable global "errno". En el segundo caso, devuelven "0" si no se ha producido una situación de error, y en caso contrario, devuelven el código del error.
Por otra parte, es muy importante comprobar que, tras una llamada a una función POSIX, no se ha producido un error. Ya que si se produce y no es localizado de inmediato, se puede propagar y provocar otro error mayor y de difícil diagnosis.

* El chequeo de errores dificulta la lectura del código, por ello se pueden utilizar las macros del sistema operativo MaRTE OS, definidas en la cabecera "misc/error_checks.h". Algunos de los ejemplos son los siguientes:

- CHK: funciones que retornan 0 o el código del error
- CHKE: funciones que retornan -1 si hay error

Así mismo, existen otras macros que informan del error, pero no finalizan el programa, las mencionadas anteriormente sí lo hacen.

2.11 Sistema operativo MaRTE

Actualmente los sistemas en tiempo real necesitan un sistema operativo eficiente, dado que requieren programación concurrente, un sistema de ficheros… De hecho, el comportamiento temporal de un fichero depende del sistema operativo, algo crítico para los sistemas en tiempo real. Por todo ello, hemos seleccionado el sistema operativo MaRTE para el funcionamiento de nuestra aplicación.

2.11.1 Características principales

Se ajusta correctamente al perfil mínimo de un sistema operativo en tiempo real ya que puede trabajar con concurruencia a nivel de threads. Todos los servicios que ofrece tienen tiempos de respuesta acotados, así como la latencia de atención a interrupciones. Posee un único espacio de direcciones, que comparten el núcleo y la aplicación. El núcleo del sistema operativo es monolítico, lo que quiere decir que concentra todas las funcionalidades del sistema (planificación, sistema de archivos, gestión de memoria…) en un solo gran programa. Está escrito en lenguaje Ada, aunque también posee partes en lenguajes C y ensamablador. A su vez, permite ejecutar aplicaciones en C, C++ y Ada, y es posible portarlo a otras plataformas.

2.11.2 Entorno de desarrollo

Se compone de una serie de herramientas para editar, compilar, enlazar, depurar, cargar aplicaciones en sistemas empotrados…

2.12 Gestión de Threads

El uso de threads surge de la necesidad de reducir una serie de aspectos de los procesos como los tiempos de cambio de contexto, los tiempos de creación y destrucción, y la necesidad de un hardware especial como es el MMU.
Por ello POSIX define las interfaces para soportar múltiples threads en cada proceso, y mejorar, de esta manera, los aspectos mencionados anteriormente.
Concretamente, en un sistema mínimo como MaRTE, existe un único proceso

2.12.1 Conceptos básicos

Un thread se define como un flujo de control simple perteneciente a un proceso. Posee un tid, un identificador de thread, que sólo es válido para threads del mismo proceso. Cada thread contiene su propia política de gestión, así como los recursos del sistema necesarios. Además todos los threads de un mismo proceso comparten el espacio de direcciones.

Una posibilidad es la implementación multi-thread, que consta de un thread principal y varios threads más que finalizarán cuando lo haga el thread principal. Además, los threads tienen 2 posibles estados para controlar la devolución de recursos al sistema:

- Detached o independiente. En este caso, cuando el thread thermina, devuelve al sistema todos los recursos utilizados
- Joinable o sincronizado. Aquí, aunque el thread en cuestión termine, mantiene sus recursos. Sólo devolverá dichos recursos cuando otro thread ejecute la instrucción pthread_join().

2.12.2 Creación de threads

Antes de crear un thread, es necesario definir previamente sus atributos en un objeto especial, (pthread_join()).

Ese objeto de atributos es creado mediante "pthread_attr_init()", aunque puede ser también eliminado mediante "pthread_attr_destroy ()". Incluso puede ser modificado, pero nunca pueden ser modificados los atributos de un thread ya creado previamente. Estos atributos son:

- Tamaño del stack (opcional)
- Dirección del stack (opcional)
- Control de devolución de recursos (detach state)
- Atributos de planificación

En cuanto a la asignación de prioridades de los threads hay que destacar que, cuanto menor sea el valor asignado, mayor será la prioridad.

2.13 Gestión del tiempo

El tiempo, como ya hemos comentado en varias ocasiones, es un aspecto fundamental a tratar, ya que para nuestra finalidad, una mala gestión del mismo resultaría crítico.

2.13.1 Conceptos básicos

En un sistema POSIX existen varios relojes, el de sistema, el de tiempo real, el monótono e incluso otros definidos por la propia implementación.
También, la Época es otro concepto importante, que ya mencionamos en la parte de GPS, y que dio comienzo a las 00:00:00 horas del 1 de Enero de 1970, UTC (Tiempo Universal Coordinado). De tal manera que los relojes mencionados quedan definidos de la siguiente manera:

- Reloj de sistema:
  - Mide en segundos el tiempo transcurrido desde la Época
  - Es usado para marcar las horas de creación de los ficheros, etc.
  - Es global al sistema
- Reloj de tiempo real
  - Mide el tiempo transcurrido desde la Época
  - Se usa para timeouts y temporizadores
  - No tiene por qué coincidir con el reloj de sistema
  - Es global al sistema
  - Tiene una resolución máxima de 20 ms y mínima de 1 ns

2.13.2 Tipo Timespec

Se trata de un tipo que permite obtener el tiempo con gran resolución, por ejemplo en nanosegundos mediante la expresión que se puede ver a continuación:
time_t tv_nsec;
Además, el sistema operativo MaRTE, ofrece una cabecera <misc/timespec_operations.h> con operaciones para comparar, sumar, restar, multiplicar o dividir "timespecs".

2.13.3 Otras operaciones

También tenemos la opción de leer la hora a partir del reloj de sistema, mediante las funciones "time", "gettimeofday" o "ctime" entre otras. Incluso podemos establecer una alarma (1 por proceso) con la función "alarm" que envía SIGALARM transcurridos los segundos que hayamos especificado.

Además tenemos que para los sistemas en tiempo real, lo más recomendable es utilizar el reloj monótono, ya que su hora no puede ser cambiada y su origen de tiempos es indeterminado.
También hay que mencionar la posibilidad de suspender momentáneamente un thread los segundos que elijamos, mediante la función sleep.

2.13.4 Threads periódicos

Una opción de gran interés, es la posibilidad que tenemos de crear threads de forma periódica, algo muy común en los sistemas operativos en tiempo real y que, en nuestro caso resulta muy útil, ya que los datos que queremos extraer del receptor es necesario que sean transmitidos al PC de forma periódica. Una forma de realizarlo fácilmente es mediante un bucle while, que esté continuamente ejecutándose, y emplear, dentro del propio bucle, una función sleep que nos permita controlar el período de repetición de la tarea.

2.14 Exclusión mutua (mutex)

Éste será el mecanismo que emplearemos en la programación del driver para asegurar una correcta sincronización entre los diferentes threads/procesos.
Podemos definir el mutex como un objeto por medio del cual múltiples threads/procesos pueden acceder a una serie de recursos de forma exclusiva. Por ello diremos que cada mutex tiene un propietario.
El mutex tiene dos posibles estados, cerrado (lock) o abierto (unlock). Si se da el primer caso, y un thread/proceso quiere acceder a la sección crítica del programa, pasará a la cola de espera del mutex con el resto de threads/procesos, y no podrá acceder hasta que el thread/proceso que lo está ocupando en ese momento lo libere y el mutex pase a estado unlock. De la cola de threads/procesos pasará al mutex el de mayor prioridad, que a su vez dependerá de la política de planificación empleada.

2.14.1 Atributos de inicialización:

- Pshared: indica si el mutex es o no compartido entre procesos
- Protocol: indica el protocolo utilizado (herencia o no de prioridad, protección de prioridad…)
- Priocelling: indica el techo de prioridad

Todos ellos se almacenan en un objeto de atributos, “pthread_mutexattr_t”. Concretamente en el sistema operativo MaRTE, no tenemos herencia de prioridad y el techo de prioridad es 0.

3 EL RECEPTOR POLARX2E

Por medio de este capítulo se pretende explicar y analizar el funcionamiento del receptor GNSS PolaRx2e de Septentrio, la aplicación RxControl realizada por el fabricante y el "driver" creado para el funcionamiento del receptor configurado con 3 antenas en tiempo real.

3.1 Principales características

De entre todas las características que el receptor PolaRx2e aporta destacaremos las que más nos interesan: proporciona rastreo de alta sensibilidad en doble frecuencia de las señales GPS (hasta 19 dB-Hz), el rastreo de la señal L2C, el seguimiento simultáneo de hasta tres antenas para la determinación de la orientación en 3D, el seguimiento simultáneo de hasta seis satélites SBAS; seguimiento de los mensajes de navegación decodificados, cálculo de la orientación 3D completo con tres antenas y un compacto y robusto formato de datos SBF unido a un conjunto completo de comandos del usuario.

*Nota: Para un correcto funcionamiento del receptor ha de ser alimentado con 5.20 V y 3.86 A. Tras ser encendido, hay que esperar un tiempo hasta que el propio receptor se inicialice correctamente antes de enviarle la configuración por el puerto serie. Sabremos que el receptor se ha inicializado correctamente cuando esté consumiendo unos 1.05 Amperios si no hay antenas conectadas, y unos 1.12 si éstas están conectadas.

3.2 Configuración del receptor

En nuestro caso disponemos de 3 antenas, por lo tanto, utilizaremos los 3 conectores de los que dispone el receptor, siendo uno de ellos para la antena que actuará como principal, y los otros 2 restantes para las antenas auxiliares.
Cada canal lógico podrá ser configurado para rastrear las señales a partir de 1, 2 o 3 antenas simultáneamente.
Las posibles configuraciones serán las siguientes:

- SingleFreq: la antena principal rastreará la señal L1 del GPS
- HeadingSF: las antenas principal y Aux1 rastrearán la señal L1 del GPS
- AttitudeSF: las 3 antenas rastrearán la señal L1 del GPS
- DualFreq: la antena principal rastreará las señales L1 y L2 del GPS
- HeadingDF: la antena principal rastreará las señales L1 y L2 del GPS mientras que la Aux1 rastreará la señal L1 del GPS
- **Attitude DF**: la antena principal rastreará las señales L1 y L2 del GPS y las antenas auxiliares la L1.
- HeadingAIIDF: las señales L1 y L2 del GPS serán rastreadas por la principal y la Aux1
- SBAS: SBAS en la antena principal
- HeadingSB: SBAS en la principal y la Aux1
- AttitudeSB: SBAS en las 3 antenas

Dependiendo de la configuración escogida, los canales lógicos serán mapeados en canales físicos de manera diferente. Dado que el número de canales físicos es fijo, el número de canales lógicos depende da la configuración.

Para cambiar la configuración, utilizar el comando SetChannelConfiguration (SCC).
En nuestro caso seleccionaremos la configuración “Attitude DF”.

3.2.1 Configuración de 3 antenas

La antena principal trabajará en DF (Dual Frequency) y las auxiliares en SF (Single Frequency), y lo configuraremos mediante el comando: SetChannelConfiguration (SCC) de la siguiente manera:

SetChannelConfiguration AttitudeDF auto <CR>

Este comando creará 9 canales lógicos, y en este caso no se reservan canales para el rastreo SBAS.
Para ello deberíamos ejecutar lo siguiente:

SetChannelConfiguration AttitudeSB 1 AttitudeDF auto <CR>

De esta manera se crearán 9 canales lógicos para AttitdeDF y uno para SBAS.

Configurar el Rx para calcular las posiciones de las antenas auxiliares:

Al usar 3 antenas y trabajar en DF, es recomendable usar el comando SetElevationMask (SEM) y de esta manera rastrear más satélites. Por ejemplo:

SetElevationMask 10 <CR> 10 sería un valor mínimo.

Otra opción que tenemos es que podemos sacar las posiciones de las antenas auxiliares a partir del Bloque SBF, para ello usaremos el comando SSO:

SetSBFOutput curr AuxPos <CR> Con ello mostramos el bloque a la salida.

Pasos a seguir:

1. Conectar las 3 antenas (Asegurarse de que no están alineadas)
2. Configurar el canal
3. (Opcional) Configurar la posición de las antenas. Con el comando (SAL):

SetAntennaLocation manual 1 5.214 10.505 0

Este comando indica: configuración manual, antena auxiliar 1, la posición según X, según Y y finalmente según Z.

4. También se puede obtener la información de la posición a partir de la salida SBF en ángulos de Euler o Cuaterniones. P.e. usando el comando SSO:

SetSBFOutput curr AttEuler <CR>

3.3 SBF: Septenario Binary Format

La información llega en bloques de datos binarios. Los diferentes tipos son:

- Bloques de medidas
- Bloques de mensaje de GPS
- Bloques de mensaje de SBAS
- Bloques de PVT
- Bloques de posición
- Bloques de información de tiempo
- Bloques de estado del receptor
- Bloques de comando de usuario

El tamaño del bloque varía entre 8 y 4096 bytes, siempre siendo múltiplo de 4. En caso de no serlo, se añadirán bytes de relleno (padding bytes) para asegurar la multiplicidad mencionada. Además cada bloque quedará dividido en diferentes partes, cada una con diferente tamaño. En las tablas del manual se muestran de la siguiente manera:

- u1 – entero sin signo de 1 byte
- u2 – entero sin signo de 2 bytes
- u4 – entero sin signo de 4 bytes
- i1 – entero con signo de 1 byte
- i2 – entero con signo de 2 bytes
- i4 – entero con signo de 4 bytes
- f4 – Coma flotante de IEEE de 4 bytes
- f8 – Coma flotante de IEEE de 8 bytes
- c1[x] – String de "x" caracteres en ASCII, relleno de 0's si fuese necesario.

*** Los datos están almacenados en Little-Endian, es decir, el bit menos significativo es el primer bit enviado.

3.3.1 Cabecera de los bloques SBF:

La cabecera tendrá un tamaño de 8 bytes distribuidos de la siguiente manera:

<-2bytes->	<-2bytes->	<-2bytes->	<-2bytes->
SYNC	CRC	ID	LENGTH
<-----	8 bytes		----->

- **SYNC** -c1(2)- 2 bytes que contienen los caracteres "$@". Sirven para controlar el inicio de una trama y también para la sincronización.

- **CRC** –u2- 16 bits de CRC de todos los bytes del bloque SBF, desde el campo de ID al final de la trama. Sigue el polinomio CRC siguiente: $x^{16} + x^{12} + x^5 + x^0$
- **ID** –u2- Usado para identificar el tipo de bloque. Puede ir desde 5889 hasta 6015, o desde 128 a 767 si se trata de un comando de usuario.
- **LENGTH** – u2 Longitud total de la trama incluyendo la cabecera. Va siempre desde 8 hasta 4096 y siempre es múltiplo de 4.

Además podemos encontrarnos con subtramas dentro de la propia trama principal, que aportarán información adicional.

El cálculo de los bits de relleno variará según la trama, ya que el tamaño total es diferente para cada una de ellas y por lo tanto también lo serán los bytes de relleno que deben ser desechados.

3.3.2 TOW y WNc:

TOW: Time of Week (en ms), se toma desde el comienzo de la semana actual del GPS.
WNc: Número de la semana actual asociado.

Dependen del tipo de bloque, es decir, en cada bloque la información puede ser totalmente diferente.

3.3.3 Do-not-use Value:

Hay ocasiones en las que no se dispone de suficiente información (o incluso ninguna) y hay ciertos campos que contienen éste valor DNU, que variará dependiendo de la trama.

3.3.4 Habilitar a la salida un bloque SBF:

En general no es posible activar un único bloque en particular, sino que hay que activar la categoría a la que pertenece el bloque. Si se da el caso de que el bloque pertenece a varias categorías, bastará con activar una de ellas.

3.3.5 Algoritmo de decodificación de las tramas SBF:

Sabemos que cada trama comienza con los caracteres “$@”, pero no obstante puede darse el caso de que esa misma secuencia tenga lugar dentro de la propia trama de nuevo, por ello, hay que comprobar que ese sea el inicio o no de la trama. Así mismo hay que comprobar el CRC y demás aspectos de la trama. El algoritmo recomendado por el fabricante es el siguiente:

1. Esperar a que en la trama aparezcan los caracteres “$@”.
2. Leer los siguientes 2 bytes. Han de ser CRC, y el valor ha de ser almacenado para su uso en el futuro
3. Leer los 2 siguientes bytes. Deben ser la ID de la trama, por lo tanto, han de estar comprendidos entre 5889 y 6015 o 128 y 767 si son comandos de usuario. En caso contrario, volver al paso 1.
4. Leer los siguientes 2 bytes. Han de corresponderse con la longitud de la trama, es decir, deben pertenecer al rango 8 – 4096. En caso de que no sea así, regresar al paso 1.
5. Leer los siguientes Length-8 bytes y guardarlos en un buffer. Calcular el CRC y compararlo con el obtenido en el punto 2. Si no es igual, regresar al punto 1. En caso correcto, la trama es buena.
6. Comprobar que la ID es de nuestro interés. Si es así, decodificar la trama.
7. Volver al punto 1 y buscar que de nuevo aparezcan los caracteres “$@” desde el último byte leído.

3.3.6 El Ancho de Banda requerido:

El ancho de banda depende del tamaño de la trama y del intervalo de tiempo en que hay que ofrecer datos a la salida. A continuación se muestran los diferentes anchos de banda dependiendo de la trama:

Identificador de trama	**Nombre de la trama**	**Ancho de Banda a 10 Hz (en bytes por segundo)**	**Descripción**
5944	GenMeasEpoch	6080.0	Depende del número de satélites rastreados. El valor 6080 se corresponde con la velocidad requerida para transmitir un paquete GenMeasEpoch cada 0.1s si en la antena principal se están rastreando 12 satélites GPS y 3 SBAS.
5889	MeasEpoch	8560.0	Depende del número de satélites rastreados. El valor 8560 se corresponde con la velocidad requerida para transmitir un paquete MeasEpoch cada 0.1s si en la antena principal se están rastreando 12 satélites GPS y 3 SBAS.
5890	ShortMeasEpoch	3760.0	Depende del número de satélites rastreados. El valor 6080 se corresponde con la velocidad requerida para transmitir un paquete ShortMeasEpoch cada 0.1s si en la antena principal se están rastreando 12 satélites GPS y 3 SBAS.
5922	EndOfMeas	160.0	
5903	PVTCartesian	680.0	
5904	PVTGeodetic	720.0	
5905	PosCovCartesian	560.0	
5906	PosCovGeodetic	560.0	
5907	VelCovCartesian	560.0	
5908	VelCovGeodetic	560.0	
5909	XDOP	320.0	
5910	PVTResiduals	1360.0	Para 10 satélites
5915	RAIMStatistics	1600.0	Para 10 satélites
5935	GEOCorrections	5760.0	Para 10 satélites
5938	AttitudeEuler	440.0	
5939	AttitudeCovEuler	400.0	
5940	AttitudeQuat	480.0	
5941	AttitudeCovQuat	560.0	
5942	AuxAntPositions	1180.0	Para 2 antenas auxiliares
5943	EndOfAttitude	160.0	

3.4 Tipos de tramas SBF

A continuación se explicarán los diferentes tipos de tramas que se envían del receptor al PC. La estructura será la siguiente:
Nombre de la trama ID Tipo de trama
No se mostrarán las estructuras de cada trama en particular, ya que se encuentran en el manual, y en puntos posteriores se especificará qué partes de cada trama serán útiles para los cálculos de la orientación.

3.4.1 Tramas de medidas:

MeasEpoch ID 5889 (Meas)

Esta trama contiene la fase de la portadora, observaciones Doppler y medidas de los indicadores de calidad de todos los satélites rastreados para un período en particular.

Además contiene N subtramas tipo “ChannelData” que contienen, a su vez, todas las medidas en DF tanto de la antena como del satélite.
Si disponemos de múltiples antenas, hay unos bytes en el campo Flag de la trama que indicarán la ID de la antena.

Output rate: se definirá con los comandos SetMeasInterval (SMI) y SetOutputDecimation (SOD).

ShortMeasEpoch ID 5890 (ShortMeas)

Es una alternativa al MeasEpoch que minimiza el tamaño del fichero log. Almacena la fase de la portadora. Es una buena alternativa para almacenar medidas durante un largo período de tiempo. No aporta información acerca de las observaciones Doppler, pero hay que tener en cuenta que pueden obtenerse derivando la fase de la portadora.
Al igual que “MeasEpoch” contiene la información de “N” ChannelData, y en la parte Flag de la trama aparece la ID de la antena a la que pertenece la información.

Output rate: se definirá con los comandos SetMeasInterval (SMI) y SetOutputDecimation (SOD).

GenMeasEpoch ID 5944 (Genmeas, Rinex)

Es parecido a MeasEpoch pero reducido. Para su decodificación tiene su propio algoritmo.

Output rate: se definirá con los comandos SetMeasInterval (SMI) y SetOutputDecimation (SOD).

EndOfMeas ID 5922 (EndOfMeas)

Es el último bloque de medida enviado para un TOW y WNc determinado. Indica que no se han realizado más medidas en dicho intervalo.

Output rate: Si está activo, esta trama se envía después de que se hayan enviado el resto de tramas de medida.

3.4.2 Tramas GPS:

GPSNav ID 5891 (Nav, Rinex)

Trama que contiene los datos de la navegación codificada para un satélite GPS. Estos datos son transmitidos en subtramas (de la 1 a la 3) del mensaje de navegación del satélite.

Output rate: si está activado, esta trama es generada, al menos, una vez cada vez que el receptor adquiere nueva información del satélite GPS.

GPSAlm ID 5892 (Alm)

Contiene los datos decodificados del almanaque (datos de un determinado período de tiempo) de un determinado satélite GPS.
La información viaja en las subtramas 4 y 5 del mensaje de navegación del satélite.

Output rate: Al menos una vez siempre que el GPS envía datos del almanaque.

GPSIon ID 5893 (IonUtc, Rinex)

Trama que contiene los datos decodificados de la Ionosfera (coeficientes Klobuchar). Son transportados en las subtramas 4 en la página 18 del mensaje de navegación (mirar).

Output rate: cada 12.5 minutos.

GPSUtc ID 5894 (IonUtc, Rinex)
Trama que contiene los datos decodificados del UTC. Igual que antes son transportados en las subtramas 4 de la página 18 del mensaje de navegación.

Output rate: cada 12.5 minutos.

GPSRaw ID 5895 (MeoRaw)

Contiene 300 bits de una subtrama GPS, que consisten en 10 tramas de 30 bits, almacenadas en 10 enteros de 32 bits dentro de la propia trama. Los bits sobrantes son utilizados de relleno.

Output rate: Típicamente, cada 6 segundos.

CNAVRaw ID 5947

Igual que la trama anterior, sin embargo, los datos son de L2C.

Output rate: cada 12.5 minutos

3.4.3 Tramas SBAS:

Los algoritmos para realizar los cálculos de la posición correcta del satélite, la hora y el alcance no aparecen en el manual, aparecen en el estándar RTCA/DO-229.
Para usar la ayuda SBAS, deben usarse las tramas GeoCorrections, que posteriormente se explicarán. Los 250 bits de "raw" del mensaje SBAS están contenidos en la trama GeoRaw.
En los mensajes SBAS la información temporal en el TOW y en el WNc viene dada desde el último bit enviado. Para hallar el tiempo en el que se envía el primer bit, hay que restar 1 segundo al TOW. El receptor PolaRx recibe continuamente información SBAS de todos los satélites que estén siendo rastreados en ese momento, pero la decodificación del mensaje se realiza sólo por el PRN que está siendo usado para realizar las correcciones en dicho instante. De tal manera que las tramas SBAS están disponibles sólo para dicho PRN, y sólo sí el modo de posicionamiento SBAS está activo. (SetPVTMode – SPM). Para la trama GeoRaw no se aplica todo lo anteriormente dicho.

GEOMT00 ID 5925 (GeoInfo)

Se envía para indicar que un mensaje SBAS vacío de tipo 0 ha sido recibido.
En la fase de test puede estar compuesto de los mismos contenidos que un mensaje tipo 2.
Tras la recepción de un mensaje de tipo 0, el receptor comprueba si el mensaje está vacío (todo 0's) o si el contenido es tipo 2. En el primer caso, es generado un bloque GEOMT00 mientras que en el segundo caso se genera un bloque tipo GeoFastCorr.

Output rate: si está activado, esta trama se genera cada vez que un mensaje tipo MT00 es recibido desde el satélite SBAS.

GEOPRNMask ID 5926 GeoInfo

Contiene la máscara PRN decodificada transmitida en el mensaje SBAS tipo 1.

Output rate: Cada vez que se recibe MT01 desde el satélite SBAS.

GEOFastCorr ID 5927 GeoInfo

Esta trama contiene las correcciones rápidas decodificadas transmitidas en los mensajes SBAS tipo 2, 3, 4, 5, 24 y 0 (si se crea como tipo 2). Contiene subtramas tipo FastCorr.

Output rate: cada vez que se genera un MT02, MT03, MT04, MT05, MT24 o un MT00 que va como tipo 2.

GEOIntegrity ID 5928 GeoInfo

Contiene la información decodificada de la integridad transmitida en un mensaje SBAS tipo 6.

Output rate: Cada vez que se envía un mensaje tipo MT06.

GEOFastCorrDegr ID 5929 GeoInfo

Esta trama contiene los factores rápidos de corrección de degradación decodificados de un mensaje tipo 7.

Output rate: Cada vez que se mande un mensaje MT07.

GEONav ID 5896 (Nav, Rinex, GeoInfo)

Contiene los datos de navegación decodificados transmitidos en un mensaje SBAS tipo 9.

Output rate: Cada vez que se manda un mensaje tipo MT09

GEODegrFactors ID 5930 GeoInfo

Contiene los factores de degradación decodificados en un mensaje tipo 10.

Output rate: Cada vez que se genere un mensaje tipo MT10.

GEONetworkTime ID 5918 GeoInfo

Contiene los parámetros del offset del tiempo de red decodificados en un mensaje tipo 12.

Output rate: Cada vez que llega un MT12.

GEOAlm ID 5897 Alm, GeoInfo

Esta trama contiene la información del almanaque decodificada en un mensaje tipo 17.

Output rate: Se genera cada vez que hay un MT17. Además, cada 3 almanaques generados, se genera otro adicional.

GEOIGPMask ID 5931 GeoInfo
Contiene la máscara de la red de puntos decodificada de la Ionosfera en un mensaje del tipo MT18.

Output rate: Cada vez que se recibe un MT18.

GEOLongTermCorr ID 5932 GeoInfo

Contiene los términos largos de corrección decodificados de los mensajes de tipo 24 y 25. Contiene subtramas del tipo IDC.

Output rate: Cada vez que llegan o un MT24 o un MT25

GEOIonoDelay ID 5933 GeoInfo

Contiene los la información decodificada de los retrasos debidos a la acción de la Ionosfera. Viene en mensajes tipo 26.

Output rate: Se genera cada vez que se recibe un MT26

GEOServiceLevel: ID 5917 GeoInfo

Contiene el nivel de servicio decodificado de un satélite SBAS geostacionario en mensajes de tipo 27. Además contiene subtramas del tipo ServiceRegion.

Output rate: Cada vez que se recibe un mensaje MT27.

GEOClockEphCovMatriz ID 5934 GeoInfo

Contiene la matriz de Cholesky de covarianza con las efemérides temporales decodificadas en mensajes del tipo 28. Además contiene tramas de tipo CovMatrix.

Output rate: Cada vez que se recibe un MT28.

GEORaw ID 5898 GEORaw

Contiene 250 bits de "raw" de un mensaje de navegación de satélite geostacionario SBAS. Los 250 bits están almacenados en 8 enteros de 32 bits sucesivos. De tal manera que los 6 restantes que quedan sin usar son rellenados por 0's.

Output rate: Cada vez que se recibe un mensaje de un satélite SBAS y si la paridad es correcta. Típicamente 1 vez por segundo.

3.4.4 Tramas PVT:

Las tramas PVT se mandan en orden, no obstante, pueden colarse otro tipo de tramas en medio, como por ejemplo tramas NMEA.

PVTCartesian ID 5903

Esta trama contiene la velocidad (Vx, Vy, Vz) y la posición (x, y, z) en el momento especificado por TOW y WNc. Las coordenadas vienen dadas en los ejes cartesianos del elipsoide WGS-84. Para calcular el tiempo que ha tardado en llegar el paquete, hay que restar el tiempo del receptor y el tiempo en el que ha sido enviada la trama.

Output rate: Se definirá con SetPVTInterval y SetOutputDecimation.

PVTGeodetic ID 5904 PVTGeo

Contiene la posición en coordenadas geodéticas a la hora especificada por TOW y WNc.

Output rate: Se definirá con SetPVTInterval y SetOutputDecimation.

PosCovCartesian ID 5905 CovCar

Contiene los elementos de una matriz simétrica de varianzas y covarianzas expresadas según las coordenadas cartesianas. Los elementos de la diagonal principal dan una idea de la precisión de los parámetros estimados.
El resto de elementos fuera de la diagonal dan la correlación de las estimas.

El PolaRx lleva implementado un modelo de error estocástico para cada una de las medidas.
Hay ocasiones en las que pueden darse valores DNU dentro de la matriz. Si la posición viene dada en 2D, si se trabaja en Base Mode o si se trabaja en RTK.

*Para calcular la correlación:

$$Rxy = \frac{\sigma_{xy}}{\sqrt{\sigma_x^2 \sigma_y^2}}$$ (0 incorrelación – 1 correlación máxima)

Output rate: Se genera junto a la trama PVTCartesian

PosCovGeodetic ID 5906 CovGeo

Similar a PosCovCartesian, sólo que ahora la información está en coordenadas geodéticas.

Output rate: se genera cada vez que se genera el paquete PVTGeodetic.

VelCovCartesian ID 5907 CovCar

Similar a las 2 anteriores, sin embargo, en este caso, la información es de velocidad y en coordenadas cartesianas.

Output rate: cada vez que se genera una trama PVTCartesian

VelCovGeodetic ID 5908 CovGeo

Contiene la misma información que la trama anterior pero en coordenadas geodéticas.

Output rate: Cada vez que se genera un PVTGeodetic

DOP ID 5909 DOP

Contiene los valores “Dilution of Precision” y los niveles de protección de los SBAS (PDOP, TDOP, HDOP, VDOP). Se usan para interpretar los actuales valores de la geometría de la constelación, además se sabe que el máximo valor para PDOP es 6.

Output rate: Cada vez que una nueva posición está disponible, se puede además, especificar una velocidad nominal con SetPVTInterval y SetOutputDecimation.

PVTResiduals ID 5910 PVTRes

Contiene los residuos de las medidas respecto a la posición calculada. Los residuos son la diferencia entre la medida y lo que se espera entre el receptor y el satélite.
Pueden servir para hacerse una idea de los errores no modelados, como por ejemplo el ruido del receptor o los efectos multicamino, es decir, para monotorizar la calidad de la posición calculada. Contiene subtramas de tipo ChannelData.
*Bajo circunstancias normales los residuos suelen "mentir" si no conocen la varianza a priori.

Output rate: Se define con SetPVTInterval y con SetOutputDecimation.

RAIMStatistics ID 5915 RAIM

Contiene las estadísticas de integridad que son calculadas por el algoritmo RAIM del PolaRx. Nos da una idea de la integridad, además si el test de integridad resulta exitoso permite al usuario ser más riguroso en cuanto al criterio de integridad, mediante los XERL (External Reliability Levels). Contiene además subtramas de tipo RAIMSatData.

Output rate: Con SetPVTInterval y SetOutputDecimation

GeoCorrections ID 5935 GeoCorr

Contiene las correcciones SBAS que el receptor ha aplicado para el cálculo de PVT. Compensan errores debidos tanto a la órbita del satélite, el reloj del satélite y los retrasos provocados por la Ionosfera. Contiene subtramas tipo SatCorr.

Output rate: Se controla con SetPVTInterval y con SetOutputDecimation.

Baseline ID 5950 Baseline

Contiene la posición relativa del receptor respecto a la estación base, sólo si hay posicionamiento DGPS o RTK.

Output rate: Con SetPVTInterval y SetOutputDecimation (página 208)

EndOfPVT ID 5921 EndOfEpoch

Trama de sincronización que indica que se ha mandado el último bloque de tipo PVT para un determinado período.

Output rate: Esta trama se envía una vez hayan sido enviadas todas las tramas PVT de un período de tiempo determinado.

3.4.5 Tramas Attitude:

Al igual que las tramas PVT, se envían por orden (AttitudeEuler primero y EndOfAttitude al final). Así mismo el orden está garantizado, sin embargo pueden "colarse" tramas de otro tipo, como por ejemplo NMEAS, entre ellas.

AttitudeEuler ID 5938 AttEuler

Contiene los ángulos (elevación, giro y orientación) a un tiempo específico indicado por TOW y por WNc, según el elipsoide WGS-84. Para mejorar la calidad de los datos hemos de utilizar la matriz de covarianza que se mostrará posteriormente.

Output rate: Se gestionará con SetPVTInterval y con SetOutputDecimation

AttitudeCovEuler ID 5939 CovEuler

Matriz simétrica de varianza-covarianza de los ángulos de posicionamiento. Además dependiendo del modo en el que se esté trabajando puede darse el caso de que haya valores del tipo DNU.

Output rate: Se maneja con SetPVTInterval y con SetOutputDecimation.

AttitudeQuat ID 5940 AttQuat

Contiene los 4 cuaterniones q1, q2, q3 y q4 a la hora especificada por TOW y WNc.

Output rate: Se gestionará con SetPVTInterval y con SetOutputDecimation

AttitudeCovQuat ID 5941 CovQuat

Es una matriz 4x4 simétrica con varianzas y covarianzas. Puede haber valores DNU.

Output rate: Se gestionará con SetPVTInterval y con SetOutputDecimation

AuxAntPossitions ID 5942 AuxPos

Contiene las posiciones y las velocidades relativas de las antenas auxiliares con respecto a la principal. Vienen dadas en coordenadas ENU (East, North, Up). Además tiene subtramas de tipo AuxAntPosition para dar las posiciones de las antenas auxiliares.

Output rate: Se gestionará con SetPVTInterval y con SetOutputDecimation

EndOfAttitude ID 5943 EndOfEpoch

Es un bloque de sincronización que es siempre la última trama de las tipo Attitude en un determinado período de tiempo dado.

Output rate: Este bloque aparece cuando se han mandado el resto de tramas del tipo Attitude para un período de tiempo establecido.

3.4.6 Tramas de información temporal:

RecieverTime ID 5914 Status

Aporta el tiempo actual con un segundo de resolución tanto en la escala temporal GPS como en la UTC. La precisión depende del nivel de sincronización del receptor con la hora del sistema GPS. Las correcciones se envían desde el GPS cada 12.5 minutos.

Output rate: 1 vez por segundo.

XPPSOffset ID 5911 PPS

Contiene el offset entre el verdadero pulso XPPS y el puslo a la salida del receptor.

Output rate: Después de cada pulso xPPS.

ExtEvent ID 5924 ExtEvent

Contiene el valor de la transición del voltaje entre los pins de entrada del TimeRx.

Output rate: Siempre y cuando se detecte una transición del voltaje entre los pi9ns de entrada del TimeRx.

3.4.7 Tramas de corrección diferencial:

DiffCorrIn ID 5919 DiffCorr

Contiene los datos RTCM y CMR.

Output rate: Cada vez que se recibe un mensaje de tipo RTCM o CMR por el PolaRx.

BaseStation ID 5949 BaseInfo

Contiene las coordenadas de la estación base a la que está conectado el receptor.

Output rate: cada vez que se genera una trama DifCorr.

BaseLink ID 5984 BaseInfo

Contiene las estadísticas del número de bytes y mensajes recibidos y aceptados por una corriente DiffCorr.

Output rate: 1 vez por segundo

3.4.8 Tramas de estado:

TrackingStatus ID 5912 Status

Describe el estado del rastreo de los canales del receptor activos. De tal manera que los canales activos serán aquellos que le han sido asignados a un satélite en concreto. Además tendrá tantas subtramas como canales activos.

Cada subtrama contiene la siguiente información:

- Número del canal lógico

- ID del satélite asignado al canal
- Estado del canal
- Información de la salud del satélite
- Acimut del satélite
- Elevación
- Dirección

El número de canales se puede controlar con SetChannelConfiguration. Si un canal no está activo, no estará en la trama. Tenemos la opción de deshabilitar todos los satélites si fuera necesario, mediante el siguiente comando: SetSatelliteTracking all of. Contiene subtramas de tipo ChannelData

Output rate: cada 2 segundos

RecieverStatus ID 5913 RxInfo, Status, Rinex

Contiene información general del estado del receptor.

Output rate: Cada 2 segundos.

RecieverSetup ID 5902 RxInfo, Rinex

Contiene información de la configuración del receptor.

Output rate: Se produce cada vez que se da uno de los 3 siguientes casos:
- Cada 10 minutos
- Cuando se emite SetAntennaParameters, SetMarkerParameters o SetObserverParameters
- Como se haya especificado en SetSBFOutput

3.4.9 Otras tramas:

Comment ID 5936 RxInfo

Contiene un String con el comentario descrito por SetSBFcoMment.
Output rate: Cada vez que se usa el comando.

3.4.10 Tramas de comandos de usuario:

[] ID 128 – 767 Comments

En este tipo de tramas se envían los comandos que el usuario quiere que realice el receptor, así como las respuestas del receptor a las órdenes que se le envíen.

3.5 Tramas NMEA

Una alternativa que puede facilitar la lectura de la información, es trabajar con las tramas en formato NMEA, ya que la información viene directamente en formato ASCII. En nuestro caso

no utilizaremos este tipo de tramas, ya que para el "driver" que programaremos, las tramas SBF se ajustan mejor a nuestras necesidades. Las tramas tienen la siguiente estructura:

Símbolo	Descripción
"$"	Comienzo de la trama
GPccc	Campo de dirección, que comienza con GP y continua con el identificador de la trama
","	Delimitador de campo
c-c	Bloque de datos de la trama
"*"	Delimitador del Checksum
hh	Campo de Checksum. Está en suma módulo 2 (XOR) de todos los bytes de la trama, excluyendo el "$" y el propio checksum.
<CR><LF>	Final de la trama

Hay 14 identificadores de trama diferentes, pero sólamente los 3 siguientes pueden aportar algunos datos que puedan resultarnos de interés.

GPVTG Course Over Ground and Ground Speed

Contiene el rumbo y la velocidad relativa a la tierra.

GPZDA Time & Date

Contiene el día, el mes y el año, así como la hora en formato UTC.

PSSN, HRP Septentrio Propietary Sentence – Heading, Roll, Pitch Information

Contiene la información de Attitude.

4 LA APLICACIÓN RXCONTROL

4.1 Descripción de la aplicación:

RxControl es una aplicación GUI (Interfaz Gráfica de Usuario) que permite controlar el receptor PolaRx, registrar los datos y monitorizar la información que procesa el receptor.

4.2 Otras herramientas disponibles:

Además de la aplicación RxControl, disponemos de otras dos herramientas de utilidad, la herramienta "Data Link" y la herramienta "SBF Converter". La primera nos permite establecer conexiones entre varios puertos serie y/o los puertos TCP/IP, mientras que la segunda nos permite convertir los ficheros en formato SBF a formato Rinex, ASCII, GPX y KML.

4.3 Pasos para comenzar la conexión PC-Rx:

Los siguientes pasos son necesarios para el correcto funcionamiento de la aplicación.

1. Asegurarse de que el receptor está encendido y funcionando, y que está conectado a la antena.
2. Usar un cable null-modem serie para conectar los puertos serie del receptor y el PC.
3. Arrancar la aplicación RxControl.
4. Aparecerá una ventana en la que seleccionaremos "Serial connection | Create new | Next >"
5. Aparecerá otra ventana en la que configuraremos la conexión entre el puerto serie del receptor y el puerto serie del PC. (Puede guardarse la configuración y utilizarla en futuras sesiones)
6. Presionar "OK" y comenzar la conexión con el receptor PolaRx

Configuración de la posición de las antenas:

Puede realizarse de dos maneras, usando la pantalla "Attitude Settings" o mediante el comando SetAntennaLocation (SAL).

El receptor PolaRx calcula la posición del vehículo a partir de la información generada por el array de antenas que hay a nuestra disposición, que en nuestro caso serán tres, una principal y dos auxiliares (Aux1 y Aux2).

Hay que tener en cuenta que dependiendo de cómo estén colocadas las antenas, puede haber un "offset" del array de antenas respecto al eje primario del vehículo, que no permita calcular la posición directamente.
Es por ello que hemos de indicarle al programa las posiciones relativas de las antenas respecto al eje primario del vehículo.

4.4 Diferentes modos de establecer las posiciones de las antenas:

Modo automático: Con este modo, el usuario puede elegir un cálculo automático de las ubicaciones de las antenas auxiliares. En este caso, la tabla de posiciones de la antena permanece inhabilitada para el usuario. El modo se activará haciendo clic en el botón OK o el botón "Apply". En RxControl el modo automático se selecciona para todas las antenas al mismo tiempo y no se pueden seleccionar para antenas individuales (para ello debería usarse el comando SAL, SetAntennaLocation). En el modo automático, que es el modo predeterminado, el receptor calcula la ubicación por sí mismo, asumiendo que tanto la antena auxiliar 1 y la 2 se encuentran en el plano xy del vehículo y que la antena auxiliar 1 se encuentra en el eje y del vehículo.

Modo manual: El modo manual puede utilizarse en caso de que el usuario conozca las posiciones relativas de las antenas a partir de medidas con precisiones milimétricas.

Modo "Attitude Calibration Offset": Es posible que el receptor calcule las componentes de "Attitude" a partir de otras componentes de referencia conocidas. Para ello, el vehículo ha de colocarse en una posición de parámetros conocidos (que será la posición de referencia), por ejemplo con el "heading" alineado con el Norte, sin "pitch" ni "roll". El receptor calculará los parámetros de "Attitude" (current Attitude) con la fase de la portadora corregida, y tras ellos se introducirán los parámetros de "Attitude" de referencia. A partir de esos parámetros de referencia y el sistema local (en coordenadas geodéticas) dónde se han generado las orientaciones de las líneas base, el receptor calcula las posiciones de las antenas y emite un comando "SetAntennaLocation" para configurar cada posición cuando se pulse "OK" o "Apply" en el cuadro de diálogo. Tener cuenta que cualquier diferencia entre la "attitude" real y la introducida en el marco de referencia desembocará en más errores que se irán propagando en los cálculos.

4.5 Configurar las diferentes salidas de datos:

También podemos configurar las salidas que serán monitorizadas posteriormente. Pueden aparecer en dos formatos, NMEA y/o SBF.
Ambas salidas podrán ser configuradas en la pestaña "Settings | Output Settings".

*SBF es un formato que se maneja por categorías, es decir una trama SBF pertenece a una o varias categorías, que son las que manejaremos, no cada trama por separado.
** Rinex. Dentro del formato SBF podemos también manejar varias categorías a la vez, ésto se consigue mediante el "subformato" Rinex.

4.6 Conexión en cadena (Daisy Chain):

También tenemos la opción de "linkar" diferentes puertos y establecer una conexión entre ellos, ya sea unidireccional o bidireccional. Para ello no debe usarse el mismo puerto serie que se usa para el RxControl.

4.7 Comandos avanzados:

Mediante esta opción podemos asignar los satélites que deseemos, especificar el tipo de rastreo, etc.

4.8 Representación de la información:

Los datos recibidos pueden mostrarse tanto en una tabla (Channel Table) o gráficamente (diferentes gráficas que se comentarán a continuación).

Channel Table: En esta tabla se muestran todos los datos referidos al canal, desde el número PRN del satélite, la elevación, el acimut… hasta el número de antenas, etc.

Representaciones gráficas:

Sky plot: Gráfico que muestra tanto la elevación como el acimut, visto desde el hemisferio topocéntrico local.

Planimetric plot: Muestra gráficamente las coordenadas horizontales calculadas por el PolaRx.

Height plot: Nos muestra gráficamente la altura elipsoidal de la posición calculada por el PolaRx

Statistics plot: Nos muestra los errores en las medidas de los PVT (Position Velocity Time)

PVT status time plot: Esta gráfica tiene 2 modos posibles.
PVT mode: Aquí se nos mostrarán los diferentes modos PVT utilizados
PVT Errors: Que nos muestra los errores en el Rx

Attitude metrics screen: En este caso se nos mostrarán por pantalla los diferentes parámetros para calcular la posición, es decir, "heading" (orientación), "pitch" (elevación) y "Roll" (giro), y quedarán representados por medio de un avión que cambiará su posición en función de los diferentes parámetros medidos.

Attitude 3D plot: Simula un vehículo en 3D. En cuanto se recibe información, el vehículo se muestra en movimiento en pantalla, dependiendo de la orientación, elevación y giro. Las coordenadas utilizadas siguen el formato ENU (East North Up). Los valores positivos serán mostrados en rojo, mientras que los negativos se mostrarán en azul.
Para controlar la vista 3D usaremos el ratón, presionando la tecla "Alt" si queremos introducir zoom.
Las antenas auxiliares se muestran en rojo (Aux1) y verde (Aux2).

Attitude in time plot: Muestra tanto el "heading" como el "pitch" como el "roll" en función del tiempo. Admite hasta 3600 puntos, 1 hora trabajando a 1 Hz.

Carrier to noise Ratio plot: Es un ratio entre la potencia de las señales recibidas y el nivel de ruido ambiental. C/N_0. Sus unidades son los dB-Hz. Este valor se verá influido por los ángulos de elevación de los satélites y por las condiciones locales, como por ejemplo los efectos multicamino.
El receptor rastrea los códigos de las portadoras y para ello asigna una correlación física específica de los canales por cada GPS dependiendo de la configuración del canal.

Cada canal muestra el C/N_0 de los diferentes GPS's en varios colores.
Puede mostrar: 1 canal físico por cada canal lógico y 2 o más canales físicos por satélite.

* En las antenas auxiliares no se muestran las modulaciones P(Y) ni las L2C.

SBAS Ionospheric Delay plot: Representa el retraso debido a la Ionosfera. Ya que ésta induce a errores de fase en la portadora, lo cuál implica que las medidas han de ser modeladas por un algoritmo de navegación.
La información se muestra en un mapa del mundo dónde podemos ver los valores de los diferentes retrasos provocados por la Ionosfera en diferentes colores.

Residuals Plot: Los residuos son las diferencias entre los intervalos de código medidos y los calculados, teniendo en cuenta el modelado de errores. Cuanto más bajos sean, la posición obtenida será más precisa.

ENU plot: Sirve para comprobar el rendimiento de l algoritmo de navegación del receptor. Tiene 3 componentes, N,E y U (Norte geográfico, Este y Cénit, es decir la recta totalmente vertical al pulto).

4.9 Metricas en un sistema de SBA:

Hay 4 conceptos que deben quedar claros, que son: precisión, integridad, continuidad y disponibilidad.
Precisión: Hay que conseguir un error mínimo entre medidas reales y los cálculos.
Integridad: Capacidad de responder correctamente ante eventos adversos que puedan ocurrir.
Continuidad: Que el componente funcione correctamente y de manera contínua sin que se produzca una situación de alarma.
Disponibilidad: Se define como la porción de tiempo en la que el sistema proporcione posiciones fijas para los niveles específicos de precisión, integridad y continuidad.

4.10 Consola de experto:

La consola de experto está diseñada para que el usuario se familiarice con los comandos enviados al receptor.
Hay 4 paneles: panel de comandos (muestra la comunicación entre RxControl y el PolaRx), panel de sentencias NMEA (Muestra las sentencias NMEA enviadas por el Rx), panel de mensajes de Diffcorr (informa de las correciones realizadas en formato RTCM o CMR cuando el receptor está en modo Rover) y panel de monitor de la salida (permite ver al usuario el comportamiento del Rx mediante una representación en texto de su estado).
Cada panel dispone de un LED que parpadea en verde si se recibe un mensaje y estará en gris si no hay nada. Se pondrá naranja si se recibe información y el panel no está activo en ese momento, y permanecerá en amarillo por defecto.
Disponemos de un botón "Freeze" con el que conseguiremos que se dejen de mostrar los mensajes por pantalla, aunque se estén generando, no se mostrarán. Mediante la pestaña de "Log" podremos almacenar los datos en un fichero de log de extensión .cmd.
En el panel de comandos, los diferentes prompts nos indicarán los diferentes estados de la comunicación que establezcamos con el receptor:

‘<’ comando que enviamos
‘>’ información de la configuración actual del Rx
‘#’ información del estado actual del Rx
‘?’ error en el comando
‘&’ indica recepción de un mensaje en formato libre
‘%’ indica que no hay información de que haya configuración actualmente en el PolaRx.

Se puede crear un script para almacenar una serie de comandos consecutivos que vayamos a utilizar habitualmente.

Por su parte, **en el panel de sentencias NMEA**, se nos mostrarán directamente las tramas NMEA.

También disponemos del **panel de mensajes de Difcorr**, que es útil si el Rx trabaja en modo Rover, pero no es nuestro caso.

Por último tenemos el **panel de eventos externos**, en el que podremos ver el tipo de evento externo que se ha producido, la hora del receptor y la hora del GPS calculada del evento externo.

4.11 Driver de alto nivel

El esquema que se muestra a continuación puede servir de idea de cómo funciona el driver que hemos implementado.

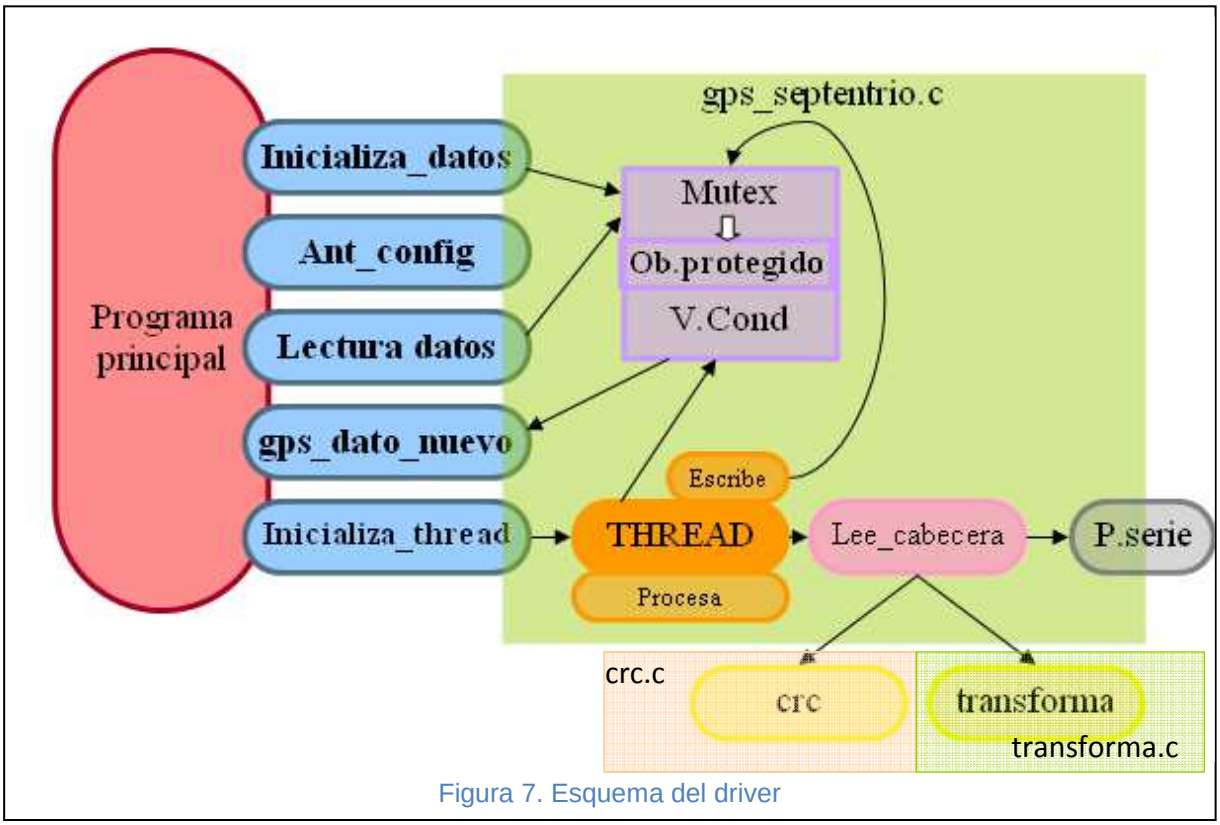

Figura 7. Esquema del driver

Lo primero que se hace desde el programa principal (Prueba) es inicializar los datos, es decir preparar el puerto serie para poder recibir los datos correctamente. Así mismo, se dispone de la opción de configurarlo para que lea ficheros ".SBF" (formato propietario del fabricante) y los procese. Tras haber configurado el puerto serie, lo siguiente es configurar correctamente el receptor, lo haremos enviando los comandos pertinentes a través del propio puerto serie. Posteriormente se inicializa el thread principal que se encargará de procesar los datos. Una vez procesada la trama, se escriben los datos que nos interesan en el objeto protegido, para que desde fuera podamos acceder a su lectura siempre y cuando haya un dato nuevo disponible. Para saber si hay o no un nuevo dato disponible, hemos creado una variable condicional, que siempre que haya un dato nuevo le será notificado al programa principal mediante "gps_hay_dato_nuevo" y procederá a la lectura del mismo. De esta manera la utilización de la CPU es más eficiente.

4.11.1 Tipos de datos que maneja el driver:

```
/*********************************************************************
 *                      CONTROLADOR de Receptor GPS Septentrio
 *                            tipos.h
 *********************************************************************
* Fecha ultima modificacion: 3 Agosto 2011
 *********************************************************************/


#ifndef _TIPOS_H
#define _TIPOS_H
#define True         ((Bool_t)1)
#define False        ((Bool_t)0)

/** Tipos de datos que manejan las tramas SBF --- PÚBLICOS**/


typedef float          F32_t;
typedef double         F64_t;
typedef signed char    I08_t;
typedef signed short   I16_t;
typedef signed long    I32_t;
typedef unsigned char  UI08_t;
typedef unsigned short UI16_t;
typedef unsigned long  UI32_t;
typedef int            Bool_t;
typedef signed char    Char_t;

/* Estructuras de los diferentes paquetes SBF --- PRIVADOS */

typedef struct
{

unsigned long TOW;
unsigned short WNc;
unsigned char N_SB;
unsigned char SB_LON;
double  PSEUDO1;
double  PSEUDO2;
unsigned int SAT[37];

}array_meas_t;

typedef struct
{

double  CACODE;
float   P1_CACODE;
float   P2_CACODE;

}array_sb_meas_t;

typedef struct
{

unsigned long   CACODE;
```

```
signed int  P1_CACODE;
signed int  P2_CACODE;
unsigned int    SVID;


}array_sb_smeas_t;


typedef struct
{

unsigned char   ERROR;
unsigned long TOW;
unsigned short WNc;
float   HEADING;
float   PITCH;
float   ROLL;

}array_attitude_t;

typedef struct
{

unsigned long TOW;
unsigned short WNc;
unsigned char ERROR;
unsigned char MODE;
float   POSX;
float   POSY;
float   POSZ;
float   VELX;
float   VELY;
float   VELZ;

}array_pvtcar_t;

typedef struct
{

unsigned long TOW;
unsigned short WNc;
unsigned char ERROR;
unsigned char MODE;
float   LAT;
float   LONG;
float   ALT;
float   VELN;
float   VELE;
float   VELU;

}array_pvtgeo_t;

typedef struct
{

//unsigned long TOW;
unsigned long TOW;
unsigned short WNc;
unsigned char N_ANT;
unsigned char SB_LON;
```

```
}array_auxant_t;

typedef struct
{
unsigned char ERROR[3];
unsigned char AMB[3];
unsigned char ID_ANT[3];
double  EAST[3];
double  NORTH[3];
double  UP[3];
double  EASTVEL[3];
double  NORTHVEL[3];
double  UPVEL[3];
//3 es el valor máximo de antenas auxiliares soportado
}array_sb_auxant_t;

/* Estructura en la que se almacena la cantidad de paquetes de cada tipo
que hay */

typedef struct
{

int cpvt;
int catt;
int cmeas;
int caux;
int cotros;
int ccomandos;
int error;


}array_cuenta_t;

/* Este tipo de dato sirver para informar al programa principal del tipo de
dato que tiene que manejar */

typedef enum
{ATTI,PVTC,PVTG,SMEAS,AUX
}tipo_dato_t;

/* Tipo de dato para la configuración del receptor */

typedef struct
{
bool P_ATT;
bool P_PVTcar;
bool P_PVTgeo;
bool P_AUX;
bool P_MEAS;
}paquetes_t;

// Estructuras para las posiciones de las antenas auxiliares


typedef struct
{
float Aux1_x;
float Aux1_y;
float Aux1_z;
```

```
}pos_aux1_t;

typedef struct
{
float Aux2_x;
float Aux2_y;
float Aux2_z;
}pos_aux2_t;


#endif
```

4.11.2 Parámetros configurables:

El driver, como veremos a continuación, ofrece un módulo de configuración, *config.h*, en el que podremos establecer una serie de parámetros, tales como la prioridad de nuestro thread, así como los paquetes que deseamos recibir por parte del receptor y la configuración de las antenas auxiliares, ya sea en modo automático o manual (tendremos que introducir nosotros las posiciones respecto de la antena principal). Para establecer la posición de las antenas hemos de seguir los pasos marcados en el apartado *3.2.1* de este informe. En nuestro caso, gráficamente, las antenas están configuradas de la siguiente manera:

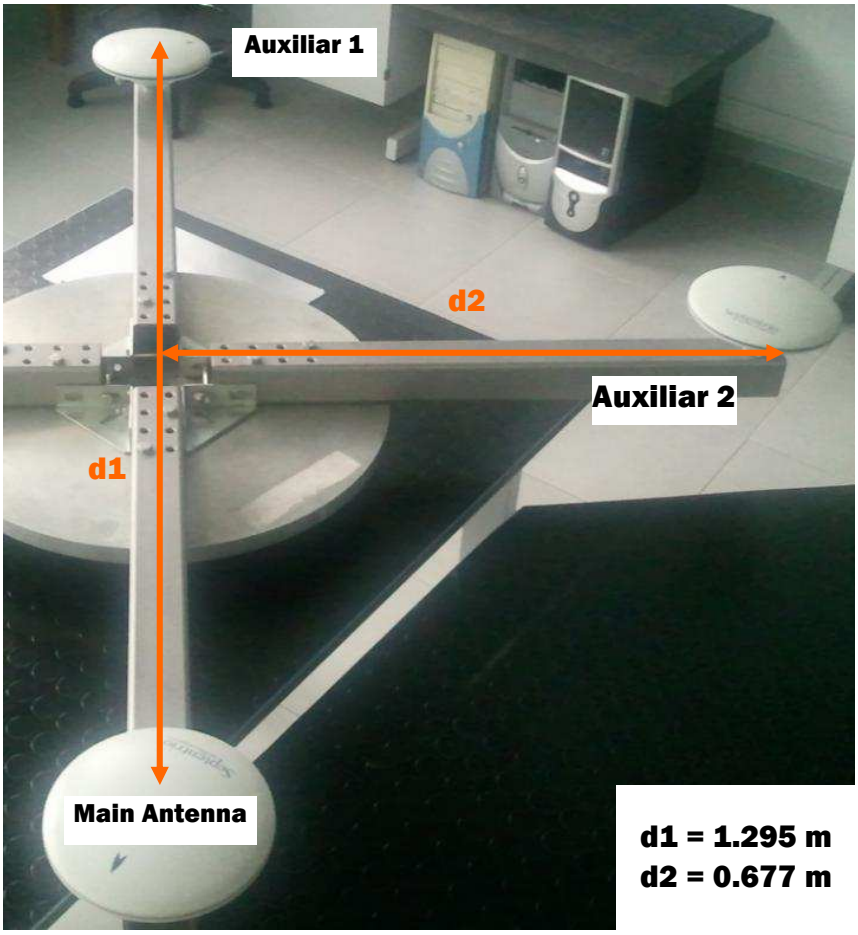

Figura 8. Configuración de las antenas

Con esos dos datos, se pueden sacar fácilmente los parámetros necesarios para la configuración de la posición, ya que tomaremos la antena principal como origen de coordenadas XYZ, estando la antena auxiliar 1 en el eje "*y*", a una distancia del origen de 1.295 metros (posición 0, 1.295, 0) mientras que la antena auxiliar 2 se encontrará en la posición (0.677, 0.677, 0).

Por último, se procederá a mostrar el código del módulo de configuración:

```
/*****************************************************************
 *                      CONTROLADOR de Receptor GPS Septentrio
                                    config.c
 *****************************************************************
* Fecha ultima modificacion: 2 Agosto 2011
 *****************************************************************/

// Parámetros configurables del funcionamiento del receptor

const int prio_gps_septentrio                 =  12;
const int techo_gps_septentrio                =  15;

//  Procesado de la configuracion de paquetes a recibir. Elegir true o
false dependiendo de las tramas que queramos recibir

paquetes_t paquete = {  .P_ATT = true,
            .P_PVTcar = true,
            .P_PVTgeo = true,
            .P_AUX = true,
            .P_MEAS = true,
};

//  Modo de establecer posici□e las antenas auxiliares (true manual, false
automático)

bool manual_mode=true;

//  Posiciones de las antenas auxiliares en caso de establcerse el modo
manual de configuración:

pos_aux1_t auxant1 = {     .Aux1_x=1.295,
                           .Aux1_y=0,
                           .Aux1_z=0,
};

 pos_aux2_t auxant1 = {     .Aux2_x=0.677,
                           .Aux2_y=0.677,
                           .Aux2_z=0,
};
```

4.11.3 Funciones del driver:

```
/*****************************************************************
*               SISTEMA DE CLCULO DE ORIENTACIM
*                          gps_septentrio.h
*****************************************************************
*
*****************************************************************
*
* Fecha ultima modificacion: 2 Agosto 2011
*
*/

#ifndef _GPS_SEPT_H
#define _GPS_SEPT_H



// FUNCIONES PÚBLICAS //

////////////////////////////////////////////////////////////////////
// Inicializacion software del modulo, que da valor inicial a las variables
// estáticas y configura el HW.

bool gps_septentrio_inicializa_datos();

//Función para comprobar la aparición de un dato nuevo a procesar
tipo_dato_t gps_hay_dato_nuevo();

//Función para configurar el receptor

bool gps_septentrio_ant_config();

////////////////////////////////////////////////////////////////////
// Inicialización software del modulo, que crea los thread de medida
// de los GPS . Debe invocarse después de la operación anterior
void gps_septentrio_inicializa_threads();

// Declaración las funciones de lectura

//gps_lee_attitude. Función lee los parámetros de orientación ubicados en
un objeto protegido por el mutex. Además dispone de una espera
condicionada, es decir, la lectura no se realizará hasta que disponga de un
dato nuevo.
void gps_lee_attitude(array_attitude_t *att);

//gps_lee_pvtcar. Función que lee los parámetros de posición y velocidad en
coordenadas cartesianas, ubicados en un objeto protegido por el mutex.
Además dispone de una espera condicionada, es decir, la lectura no se
realizará hasta que disponga de un dato nuevo.
void gps_lee_pvtcar(array_pvtcar_t *car);

//gps_lee_pvtgeo. Funciones lee los parámetros de posición y velocidad en
coordenadas geodéticas ubicados en un objeto protegido por el mutex. Además
dispone de una espera condicionada, es decir, la lectura no se realizará
hasta que disponga de un dato nuevo.
void gps_lee_pvtgeo(array_pvtgeo_t *geo);
```

```
//gps_lee_auxant. Función lee los parámetros de posición y velocidad de las
antenas auxiliares, ubicados en un objeto protegido por el mutex. Además
dispone de una espera condicionada, es decir, la lectura no se realizará
hasta que disponga de un dato nuevo.
void gps_lee_auxant(array_auxant_t *ant,array_sb_auxant_t *sb_auxant);

//gps_lee_smeas. Función que lee los parámetros de pseudodistancia y los
satélites activos, ubicados en un objeto protegido por el mutex. Además
dispone de una espera condicionada, es decir, la lectura no se realizará
hasta que disponga de un dato nuevo.
void gps_lee_smeas(array_meas_t *meas,array_sb_smeas_t *sb_smeas);

//gps_lee_cuenta_tramas. Función que lee el número de tramas procesadas,
ubicado en un objeto protegido por el mutex. Además dispone de una espera
condicionada, es decir, la lectura no se realizará hasta que disponga de un
dato nuevo.
void gps_lee_cuenta_tramas(array_cuenta_t *cuenta_tramas);

// FUNCIONES PRIVADAS
//
//gps_escribe_attitude. Función que escribe la información de orientación
en el objeto protegido.

void gps_escribe_attitude(array_attitude_t att)();

//gps_escribe_pvtcar. Función que escribe la información de la posición del
vehículo en coordenadas cartesianas en el objeto protegido.

void gps_escribe_pvtcar(array_pvtcar_t car)();

//gps_escribe_pvtcar. Función que escribe la información de la posición del
vehículo en coordenadas geodéticas en el objeto protegido.

void gps_escribe_pvtgeo(array_pvtgeo_t geo)();

//gps_escribe_auxant. Función que escribe la información de la posición y
de la velocidad de las antenas auxiliares respecto de la principal en el
objeto protegido.

void gps_escribe_auxant(array_auxant_t ant,array_sb_auxant_t sb_auxant)();

//gps_escribe_smeas. Función que escribe la información de los paquetes
SMEAS en el objeto protegido.

void gps_escribe_smeas(array_meas_t meas,array_sb_smeas_t sb_smeas)();

#endif // _SEPT_H
```

4.11.4 El thread principal:

El thread principal utiliza una función para detectar la aparición de una nueva trama (lee cabecera), buscando continuamente los 2 caracteres que significan comienzo de trama, “*$@*”, y comprobando posteriormente que tanto la ID de la trama como su longitud son correctas. Tras ello, se procede a realizar un cálculo del CRC y se compara con el recibido en dicha trama. Cuando se acaba de comprobar que la trama ha sido correctamente recibida, dicha función devuelve el valor “true” lo que le indica al thread principal que puede procesar la trama.

Para el cálculo del CRC se utiliza una función perteneciente a un modulo externo, crc.c, llamada "*ComputeCRC*". Además, hay que destacar, que para pasar de los datos recibidos que están en formato "*unsigned char*" a los diferentes formatos que nos interesan (*float*, *double*, *integer*…), disponemos de una serie de funciones que reciben la cantidad de caracteres correspondientes y devuelve el dato en el formato deseado. Todas ellas se encuentran en el módulo transforma.c.

A continuación se muestra un pseudocódigo para facilitar la comprensión del funcionamiento del thread principal:

```
Bucle infinito
    Si lee_cabecera
        Seleccionar ID_trama
            Caso ID_PVTcar
                Procesa_PVTcar
                Escribe_PVTcar
                Cuenta_tramas_PVTcar ++
                break

            Caso ID_PVTgeo
                Procesa_PVTgeo
                Escribe_PVTgeo
                Cuenta_tramas_PVTgeo ++
                break




            Caso ID_AUXant
                Procesa_ AUXant
                Escribe_ AUXant
                Cuenta_tramas_ AUXant ++
                break

            Caso ID_ATTEuler
                Procesa_ ATTEuler
                Escribe_ ATTEuler
                Cuenta_tramas_ ATTEuler ++
                break

            Caso ID_SMEAS
                Procesa_ SMEAS
                Escribe_ SMEAS
                Cuenta_tramas_ SMEAS ++
                break

            Caso ID_COMMAND
                Cuenta_tramas_ COMMAND ++
                break
```

```
                En otro caso
                    Cuenta_tramas_OTROS ++
                    break

Si no entonces
        Cuenta_tramas_ERROR ++
Fin si

Fin bucle infinito
```

4.11.5 Ejemplo de uso del driver:

Para que el driver funcione correctamente, debemos realizar un programa principal que se encargue de utilizar correctamente las funciones que hemos creado en el resto de programas. Por ello, hemos creado un fichero de prueba, que nos sirve para comprobar el correcto funcionamiento del driver realizado. A continuación se mostrará un pseudocódigo, que, con la ayuda del gráfico mostrado en el apartado 2.11, explicará el funcionamiento del fichero de prueba, y por tanto, del driver funcionando al completo:

```
Si configura el receptor correctamente
        Si la inicialización de datos fue correcta
                Inicializa thread
        Fin Si
Fin Si

Inicialización de la intefaz gráfica.

Bucle infinito

        Seleccionar gps_hay_dato_nuevo()
                Caso ID_PVTcar
                    lee_PVTcar
                    mostrar por pantalla
                    break

                Caso ID_PVTgeo
                    lee_PVTgeo
                    break




                Caso ID_AUXant
                    lee_ AUXant
                    calcula media posiciones antenas auxliares
```

mostrar por pantalla
break

Caso ID_ATTEuler
lee_ ATTEuler
mostrar por pantalla
break

Caso ID_SMEAS
lee_ SMEAS
mostrar por pantalla
break

En otro caso
break

Fin bucle infinito

Por pantalla podríamos ver una cosa así:

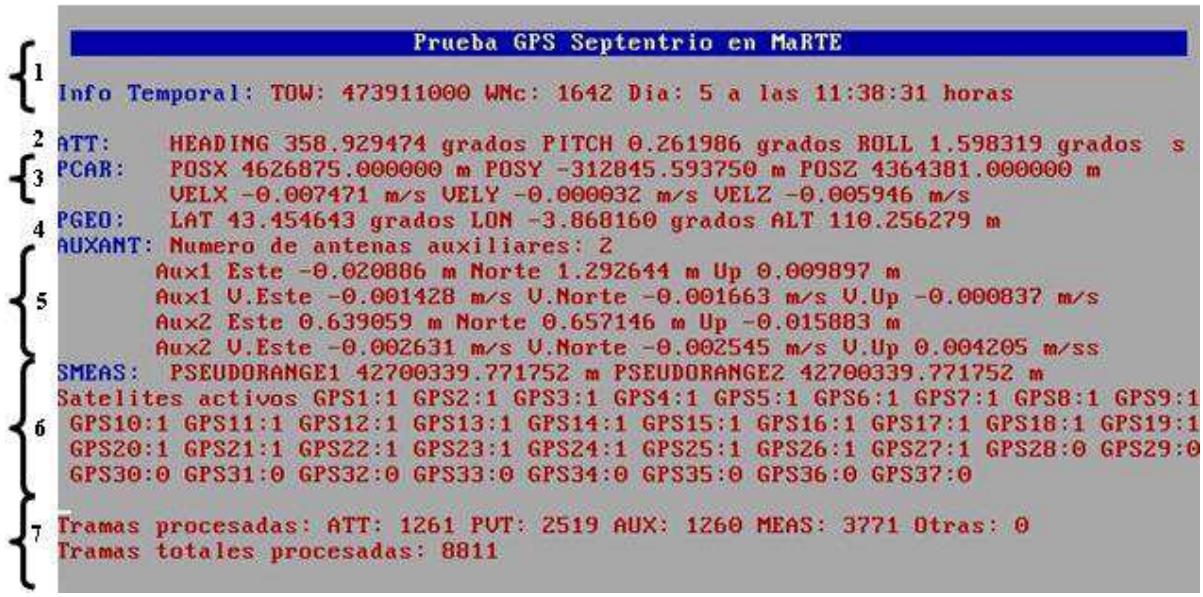

Figura 9. Pantallazo del programa en MaRTe

1. Cabecera del programa e información temporal: En la parte superior de la pantalla nos encontramos con la cabecera del interfaz del programa. A continuación nos encontramos con la información temporal, extraída del TOW y del WNc enviado por cada paquete SBF.

2. Información de la orientación: A partir de los paquetes Attitude, podemos obtener los datos correspondientes a la orientación del vehículo, que es lo que se muestra en este apartado.

3. Información de la posición en coordenadas cartesianas: En este caso se utilizan los paquetes PVTCar para obtener la posición y velocidad del vehículo, en coordenadas cartesianas.

4. Información de la posición en coordenadas geodéticas: Mismo caso que el anterior, sin embargo el paquete procesado es el PVTGeo.

5. Información de las antenas auxiliares: En este apartado vemos el número de antenas auxiliares presentes en el sistema, así como sus posiciones y velocidades relativas a la antena principal. Esta interfaz está diseñada para mostrar, a lo sumo, la información de 2 antenas auxiliares dado que en nuestro sistema no utilizaremos más, aunque el driver es capaz de calcular el de todas las presentes.

6. Información de la pseudodistancia y de los satélites activos: A partir de la información de los paquetes SMEAS, el driver es capaz de determinar la pseudodistancia a cada uno de los satélites y cuáles de ellos están activos en cada instante de tiempo.

7. Por último se muestran número de tramas procesadas por tipo y en total.

* En caso de que se produzcan errores en el procesado de la información, tales como errores de CRC, la interfaz mostrará por pantalla un mensaje de advertencia si el porcentaje de acierto es inferior al 95%, si éste se mantiene por debajo de ese valor durante un cierto tiempo, el programa finalizará y será necesaria su reiniciación.

5 MESA DE ROTACIÓN HAAS TRT-210

En este último apartado se va a relatar el diseño e implementación de un driver específico para la mesa de rotación HAAS TRT-120, utilizada para realizar simulaciones de movimiento con las antenas del receptor GPS y comprobar así el correcto funcionamiento del driver en situaciones de movimiento.

5.1 Comunicación a través del puerto serie

El dispositivo dispone de una serie de parámetros configurables que han de modificarse a través del puerto serie o del panel frontal con el que cuenta el aparato.

No se va a ahondar en las diferentes funciones de cada parámetro, ya que tan sólo uno resulta de interés en nuestro caso. Concretamente nos referimos al parámetro 21, que tiene que estar a un valor diferente de cero.

Por otro lado, el dispositivo ha de encontrarse en el modo "RUN" para un correcto funcionamiento.

Todos los comandos que se envíen han de ir precedidos por el código del eje seleccionado (U, V, W, X, Y o Z).

En cuanto a la configuración del puerto serie, éstos son los parámetros que han de establecerse: 19200 bps, 7 bits de datos, paridad impar y 2 bits de stop. Así mismo la conexión se realizará sin Null-Módem.

Tras haber configurado correctamente el puerto serie, es momento de analizar los comandos que hay que enviar a la mesa. Todos ellos terminarán en "\r". A continuación se muestran los comandos a enviar y el correcto orden:

1. **UO**. Permite comenzar la comunicación mediante el puerto serie.
2. **UCX**. Establece que la posición en la que se encuentre la mesa en ese instante sea la de referencia cada vez que se le envía un comando de movimiento. La X será el eje a mover, en nuestro caso será el U.
3. **USXnn.nn**. Establece el paso que ha de moverse el eje X, el valor viene dado en nn.nn grados.
4. **UFXnn.nn**. Con este comando establecemos la velocidad del movimiento, en nn.nn grados por segundo.
5. **UGX.91**. Indica que el movimiento se realiza en modo incremental.
6. **UP**. Comando para recibir la posición actual de la mesa, así como si está en movimiento o parado. El comando de respuesta viene dado de la siguiente forma:
    a. **Xnnn.nnnR Bnnn.nnnR** si está en movimiento
    b. **Xnnn.nnn Bnnn.nnn** si está parado

5.2 El controlador

El driver ha sido diseñado en C y bajo sistema operativo Linux. El objetivo es conseguir enviar la configuración y las órdenes pertinentes a la mesa de rotación, a través del puerto serie, para que realice unos cambios de posición que determinaremos con el programa.

Para ello se ha hecho uso de 3 threads, con las siguientes funcionalidades:

- **Thread 1**: Encargado de la lectura de comandos enviados por teclado. Será el thread con menor prioridad de los 3.
- **Thread 2**: En este thread se trata de emular una máquina de estados, como la que se muestra a continuación:

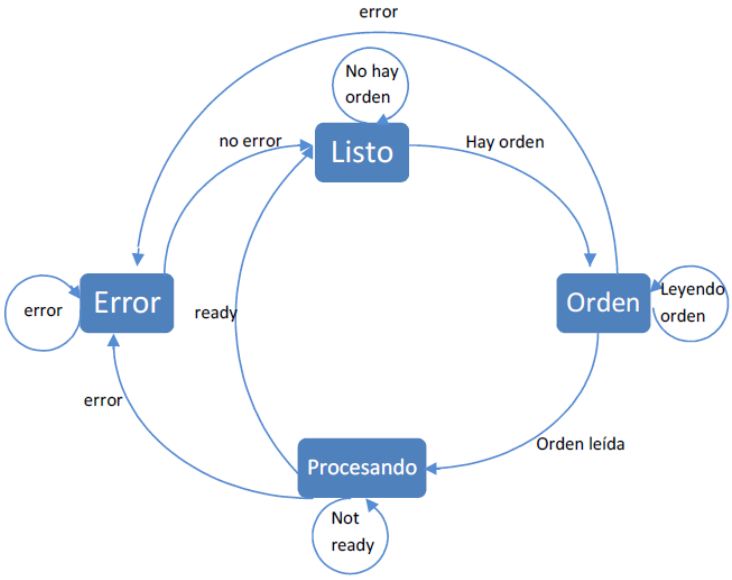

Figura 10. Máquina de estados del driver

De tal manera que, tendremos 4 posibles estados:

- Listo: La mesa está preparada para recibir un comando por teclado.

- Orden: La orden ha sido leída por teclado y será enviada a la mesa.

- Procesando: La orden ha sido enviada a la mesa y se está procesando.

- Error: Posibles situaciones de error.

Además, para ir "saltando" entre los estados, tenemos una serie de señales que indicarán si se puede pasar de un estado a otro. Son las siguientes:

- Hay orden: Si se ha leído una orden por teclado, se activa, para pasar del estado "**Listo**" al estado "**Orden**". Mientras permanezca desactivada, permanecerá en el estado "Listo" hasta que reciba dicha orden por teclado.

- Leyendo: Tras haber recibido la orden por teclado, se comprueba que sea correcta, y se envía a la mesa, pasando al estado "**Procesando**" si se desactiva. En caso de permanecer activada, se mantiene el estado "**Orden**" hasta que se envíe la orden a la mesa.

- Ready: Esta señal indicará a la máquina de estados que la orden ha sido procesada correctamente por la mesa, en caso de que la señal esté activa., pasando al estado "**Listo**". Si permanece desactivada, nos mantendremos en el estado "**Procesando**" hasta que la orden se procese por parte de la mesa.

- Error: Señal que nos indica si se ha producido un error, en tal caso se pasará automáticamente al estado de "**Error**", mientras que si se desactiva la señal, se pasa nuevamente al estado "**Listo**".

Por último, respecto a este thread, también indicar que es el encargado de enviar la orden deseada a través del puerto serie. Tiene una prioridad intermedia, entre el thread 1 y el 3.

- **Thread 3**: Este thread es el de mayor prioridad, ya que se encarga de "escuchar" continuamente del puerto serie. Procesa las tramas que recibe de la mesa y comprueba, a partir de esos datos, si la orden se ha procesado correctamente.

5.3 Tipos de datos

Tan sólo se han definido dos tipos de datos específicos para este controlador, un tipo de datos para los estados de la máquina de estados del "Thread 2" y otro para el "Switch" utilizado en la escucha del puerto serie del "Thread 3", tal y como se puede comprobar a continuación:

```
////////////////////////////////////////////////////////////////////////
///////////////            tipos.h        //////////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//////////////////    Posibilidades de la máquina de estados  ///////////
////////////////////////////////////////////////////////////////////////

typedef enum {LISTO,ORDEN,PROCESANDO,ERROR}estado_t;

////////////////////////////////////////////////////////////////////////
//////////////    Estados del switch de escucha del pto serie    //////////
////////////////////////////////////////////////////////////////////////

typedef enum {CABECERA,COLA,PARADO}leyendo_t;
```

5.4 Configuración del controlador

En este apartado tenemos varios parámetros que podemos configurar según nuestras necesidades. Primero encontramos las prioridades de los threads, de tal manera que tenga mayor prioridad el "Thread 3" y menor el "Thread 1", quedando en medio el "2". Otro parámetro que podemos variar es el número de movimientos que queremos introducir en el movimiento personalizado. El otro parámetro importante es el ángulo máximo (en valor absoluto) que queremos que gire nuestra mesa en el eje de elevación.

```
////////////////////////////////////////////////////////////////////////
//////////////////////////        config.c    //////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//////////////////////////    Prioridades de los threads  //////////////
////////////////////////////////////////////////////////////////////////

const int prio_thread_1 = 8;
const int prio_thread_2 = 10;
const int prio_thread_3 = 12;
const int techo_prio    = 15;

////////////////////////////////////////////////////////////////////////
//////////    Número de movimientos del movimiento personalizado ////////
////////////////////////////////////////////////////////////////////////

const int nmov          = 2;

////////////////////////////////////////////////////////////////////////
////////    Valor máximo del ángulo de elevación en valor absoluto ///////
////////////////////////////////////////////////////////////////////////

const float limite_A    =3;

int cfg_prio_thread_1() {return prio_thread_1;}
int cfg_prio_thread_2() {return prio_thread_2;}
int cfg_prio_thread_3() {return prio_thread_3;}
```

5.5 Programa principal

A continuación se van a comentar una serie de aspectos que pueden interesar al programador en el caso de querer modificar el programa, de manera que sepa a qué partes del mismo acceder sin comprometer su correcto funcionamiento.

5.5.1 Inicialización de la máquina

Una parte importante para el buen funcionamiento es inicializar la máquina correctamente. Por ello haremos un inciso y vamos a comentar cómo hemos de hacer dicha inicialización "manualmente" antes de ejecutar el programa.

1. Encender la máquina con el botón de "Emergency Stop" desactivado.
2. Pulsar la tecla verde "Cycle Start".
3. Mantener pulsado la tecla "Clear" hasta que el display muestre por pantalla el valor 000.000 en el eje "A".
4. Pulsar el botón "Display" tantas veces como sea necesario hasta pasar al eje "B".
5. Volver a mantener pulsado el botón "Clear" hasta que el display muestre por pantalla el valor 000.000 en el eje "B".
6. Ejecutar el programa.

Será el propio programa el que habilite la comunicación por el puerto serie mediante el comando "UO". Así mismo, hay una serie de comandos que pueden ser enviados para

devolver la máquina a su posición inicial, aunque recomendamos utilizar la comando “5” del programa, ya que podemos ajustar la velocidad de paso a nuestras necesidades, ya que utilizando otros comandos más específicos de la máquina, la velocidad es muy alta y eso puede provocar resultados indeseados.
En cualquier caso, si se desea establecer algún cambio en la inicialización de la máquina, se deben realizar modificaciones en la función “inicializa_maquina” del fichero “mesa.c”.

Para cambiar los límites tanto de movimientos personalizados como de los ángulos máximos de elevación, tal y como se ha comentado, han de modificarse los parámetros que se encuentran en el fichero “config.c”,

6 CONCLUSIONES

Tras los tres meses que conllevó el diseño y la implementación del driver del receptor GPS y de la mesa de rotación, los resultados obtenidos fueron muy satisfactorios.

En cuanto al driver del receptor, se consiguió realizar los cálculos necesarios y mostrar por pantalla los datos requeridos en tiempo real, gracias a su implementación en el sistema operativo MaRTe. Consiguiendo, de ésta manera, una mayor eficiencia a la hora de mostrar los datos que la establecida por el software de la casa.

Por otro lado el diseño del controlador para la mesa de rotación supuso un reto interesante, ya que, si bien el propio driver no fue demasiado complejo de diseñar comparado con el driver del receptor GPS, si lo fue la fase previa dedicada a entender el funcionamiento de la mesa. Esto se debió principalmente a que el manual de la mesa era bastante incompleto en algunas partes, sobre todo en lo referido a la comunicación por el puerto serie.

7 AGRADECIMIENTOS

A todos mis compañeros de trabajo durante mi período de prácticas en el Centro Tecnológico de Componentes, en especial a María Campo-Cossío, Raúl Arnau, Daniel Bolado y Alberto Puras, todos ellos integrantes del departamento Aeroespacial y que hicieron que mi estancia allí fuera muy agradable y productiva, tanto a nivel de aprendizaje personal como a nivel de introducción al mundo laboral.

También quisiera agradecer a Tomás Fernández Ibáñez, director de este proyecto y Subdirector Jefe de Estudios de Ingeniería de Telecomunicación y Grado en Ingeniería de Tecnologías de Telecomunicación, que ha mostrado gran interés en todo momento por el estado del proyecto y me ha facilitado la tramitación del mismo.

A mis compañeros de carrera, que me han ayudado en momentos complicados durante mi etapa universitaria.

No menos importantes han sido mis amigos, quienes aún cuando las cosas se tornaban difíciles siempre confiaron en mí.

Y por supuesto a mis padres, Luis Alberto y María Jesús, que siempre me han apoyado en todo y que sin ellos nunca hubiera llegado a estar dónde estoy ni a ser quién soy.

8 BIBLIOGRAFÍA

- **Presentación: REAL-TIME ATTITUDE DETERMINATION SYSTEM BASED ON GPS CARRIER PHASE MEASUREMENTS AND AIDED BY LOW-COST INERTIAL SENSORS FOR HIGH DYNAMIC APPLICATIONS. María Campo-Cossío. CTC**
- **Introducción a los Cuaterniones. Adriana Favieri, Universidad Tecnológica Nacional de Argentina**
- **Procesado de Datos GPS: código y fase. Algoritmos, Técnicas y Recetas. Grupo de Astronomía y Geomática (gAGE). M. Hernández-Pajares, J.M. Juan Zornoza, J. Sanz Subirana. gAGE-NAV S.L. Barcelona, Spain.**
- **Computación concurrente. J.M. Drake, M. Aldea, M. González**
- **Development of a Real-Time Attitude System Using a Quaternion Parameterization and Non-Dedicated GPS Receivers. John B. Schleppe. Unversity of Calgary, Canada.**
- **Septentrio PolaRx2/2e User Manual**
- **Septentrio RxControl User Manual**
- **Manual Mesa de Rotación HAAS TRT-120**