ISSN 3045-5693

JCAAC

Journal of Computational Astronomy & Astronomical Computing

Revista de astronomía computacional y cálculo astronómico

Nº 4. December 2025

In this issue:

Edited by ***Federación de Asociaciones Astronómicas de España*** in Madrid (c/Serrano, 117)

The Journal of Computational Astronomy & Astronomical Computing is an effort by the FAAE - Grupo de Cálculo Astronómico (GCA) to encourage the use of software tools and the development of codes and algorithms for astronomical applications within the framework of amateur astronomy, as well as to connect the amateur community with the professional astronomy community and promote cross-collaboration and ProAm projects between both groups.

Sociedad Española de Astronomía

Federación de Asociaciones Astronómicas de España

This work is licensed under a Creative Commons Attribution 4.0 License

Journal of Computational Astronomy and Astronomical Computing
An online publication of Federación de Asociaciones Astronómicas de España, Grupo de Cálculo Astronómico
Editorial Board: Enrique Velasco, Sergio Díaz, Joaquín Álvaro, César González, José Luis Navarro, Enrique Sánchez, Roberto Ruiz, Ramón Naves y Montse Campàs.
Advisory Board: Luis Mederos and Matilde Fernández.
https://federacionastronomica.es/index.php/recursos-astronomicos/journal-of-ca-ac

JCAAC

EDITORIAL

Nuestra revista llega a su cuarto número, y a su primer año de vida. Durante este año hemos publicado artículos sobre diversas facetas de la astronomía amateur, con el objetivo de servir a la comunidad y llenar un hueco, que creemos existe, entre los astrónomos amateur avanzados interesados en desarrollos de software y hardware, así como en el análisis de los resultados de observaciones. La acogida de la publicación ha sido excelente, si nos atenemos al número de lecturas recogido en nuestro servidor. Sin embargo, sería deseable una mayor participación en el envío de trabajos: somos conscientes de la existencia de una comunidad amateur interesada en estos aspectos y que realiza desarrollos notables en este área. Pedimos pues, a aquellas personas interesadas, que envíen sus contribuciones, o que animen a compañeros de su entorno a que lo hagan. Asimismo, nos gustaría satisfacer una de los objetivos de esta publicación, la de servir de enlace entre la comunidad amateur y la investigación profesional puntera que se realiza en España. Es por ello que animamos también a la comunidad de la astronomía profesional española a que participe y utilice este medio como vehículo para difundir sus actividades investigadoras.

En este número de JCAAC continuamos con una serie de artículos enmarcados en las secciones específicas abiertas en los números anteriores. La serie sobre el *Observatorio Virtual* ofrece una nueva entrega, que vuelve a centrarse en la caracterización de cúmulos abiertos, pero utilizando esta vez la herramienta `VOSA`. La sección sobre *Astronomical Computing* trata del importante problema de cómo generar efemérides precisas para el Sol y la Luna, tema básico sobre el cual gira gran parte de la programación de eventos astronómicos de interés. Finalmente, la sección *Software for Photometry & Astrometry* continúa con la aplicación del software `Tycho`, esta vez al problema de cómo confirmar la astrometría de un NEO y los pasos a seguir para enviar un informe al *Minor Planet Center*. En este número no ha sido posible incluir una nueva entrega (la tercera) de la sección sobre *Programación de Dispositivos Astronómicos*; esperamos hacerlo en el próximo.

Además de las secciones anteriores, presentamos en este número un interesante artículo sobre la implementación y análisis de un dispositivo para la generación de una base de tiempos precisa y de bajo coste basada en un receptor GPS. Se trata de un tema central en aplicaciones tales como el registro de ocultaciones lunares, asteroidales o de otro tipo, en las que el control del tiempo UTC exige precisiones de pocos milisegundos o menos. El artículo es claro y extenso, y se encuentra perfectamente documentado. Animamos a los lectores interesados en estas observaciones a realizar esta implementación, en caso de que no dispongan de una en su propio equipo, o de considerar la mejora de la instalación ya existente si lo consideran oporturno.

Our journal reaches its fourth issue and its first year of existence. During this year we have published articles on various facets of amateur astronomy, with the aim of serving the community and filling a gap that we believe exists between advanced amateur astronomers interested in software and hardware developments, as well as in the analysis of observational results. The reception of the publication has been excellent, judging by the number of reads recorded on our server. However, a larger participation in the submission of works would be desirable: we are aware of the existence of an amateur community interested in these aspects and that produces remarkable developments in this area. We therefore encourage those of you who are interested to send their contributions, or to encourage colleagues around them to do so. Likewise, we would like to fulfill one of the objectives of this publication: to serve as a link between the amateur community and the cutting-edge professional research being carried out in Spain. For this reason, we also encourage the Spanish professional astronomy community to participate and use this medium as a vehicle to disseminate their research activities.

In this issue of JCAAC we continue with a series of articles framed within the specific sections opened in previous issues. The series on the *Virtual Observatory* offers a new installment, once again focusing on the characterization of open clusters, but this time using the VOSA tool. The section on *Astronomical Computing* addresses the important problem of how to generate accuratee ephemerides for the Sun and the Moon, a fundamental topic underpinning much of the tasks in the computation of interesting astronomical events. Finally, the section on *Software for Photometry & Astrometry* continues with the application of the `Tycho` software, this time to the problem of how to confirm the astrometry of a NEO and the steps to follow to submit a report to the Minor Planet Center. In this issue it was not possible to include a new installment (the third) of the section on *Programming Astronomical Devices*; we hope to do so in the next one.

In addition to the previous sections, we present in this issue an interesting article on the implementation and analysis of a device for generating an accurate, low-cost time base based on a GPS receiver. This is a central topic in applications such as the recording of lunar, asteroidal, or other types of occultations, in which controlling UTC time requires accuracies of a few milliseconds or less. The article is clear and extensive, and is perfectly documented. We encourage readers interested in these observations to carry out this implementation if they do not have such a device in their own equipment, or to consider improving their existing setup if they consider it appropriate.

E. Velasco, December 2025

JCAAC

Journal of Computational Astronomy and Astronomical Computing

An online publication of Federación de Asociaciones Astronómicas de España, Grupo de Cálculo Astronómico

Editorial Board: Enrique Velasco, Sergio Díaz, Joaquín Álvaro, César González, José Luis Navarro, Enrique Sánchez, Roberto Ruiz, Ramón Naves y Montse Campàs.

Advisory Board: Luis Mederos and Matilde Fernández.

`https://federacionastronomica.es/index.php/recursos-astronomicos/journal-of-ca-ac`

CONTENTS

All contents of this publication are licensed under a Creative Commons Attribution 4.0 License
Published codes can be used and distributed under a Gnu gpl license

With the sponsorship of Sociedad Española de Astronomía (SEA), Grupo ProAm

JCAAC (2025), **4**, 5–31

JCAAC

HARDWARE

Base de tiempos precisa basada en GPS USB

Sergio Díaz Ruiz[1]

[1]Asociación Astronomía Sevilla, Spain. E-mail: sergio.diaz.ruiz@gmail.com.

Keywords: GPS, PPS, NTP, USB, gpsd, chrony, ntpd

© Este artículo está protegido bajo una licencia Creative Commons Attribution 4.0 License

Resumen

La observación de ocultaciones estelares, campo en el que la astronomía amateur supone una colaboración clave, requiere una resolución temporal del orden del milisegundo, siendo la señal GPS la mejor referencia temporal disponible. Este trabajo aborda una implementación práctica y económica que utiliza un módulo receptor GPS de bajo coste conectado mediante un conversor USB-serie, transportando la señal Pulse-Per-Second (PPS) a través de una de las líneas de control del interfaz serie. Esta solución, que ha sido recurrentemente descartada debido a las dudas en torno al impacto en la precisión temporal del transporte USB, se documenta aquí detalladamente, incluyendo la configuración hardware y software para entornos Linux (`gpsd` y `chrony`) y Windows (Meinberg NTP). Se destaca la importancia de un proceso de calibración de desfase, crucial para corregir las latencias inherentes a las capas de software intermedias. Los resultados de las pruebas confirman la viabilidad de la solución, demostrando que, tras la corrección del desfase, se logra una precisión por debajo del milisegundo en ambos sistemas.

Abstract

The observation of stellar occultations, a field in which amateur astronomy plays a key collaborative role, requires temporal resolution on the order of milliseconds, with the GPS signal being the best available time reference. This work addresses a practical and economical implementation that uses a low-cost GPS receiver module connected via a USB-to-serial converter, transporting the Pulse-Per-Second (PPS) signal through one of the control lines of the serial interface. This solution, which has been repeatedly dismissed due to doubts about the impact of USB transport on temporal accuracy, is documented here in detail, including the hardware and software configuration for Linux (`gpsd` and `chrony`) and Windows (Meinberg NTP) environments. The importance of an offset calibration process, crucial for correcting the latencies inherent in intermediate software layers, is highlighted. Test results confirm the viability of the solution, demonstrating that, after offset correction, sub-millisecond accuracy is achieved on both systems.

1. Introducción

La ciencia ciudadana posee una larga tradición en las ciencias naturales, siendo la práctica habitual hasta la especialización de las disciplinas científicas a finales del siglo XVIII. En astronomía, la aportación de los astrónomos amateur, especialmente en el campo de la observación de ocultaciones estelares [2], inicialmente por la Luna y más recientemente por asteroides y otros cuerpos del sistema solar, es especialmente destacable, dada la importancia que tiene contar con múltiples observaciones del mismo fenómeno, realizadas desde distintos puntos de la geografía terrestre [3].

Estas observaciones permiten conocer la forma de los cuerpos (y si presentan anillos) [3], estudiar su atmósfera [4] o determinar de forma precisa su posición, información que resulta crítica para misiones como, p.ej., NASA Lucy [5].

Para aportar datos de calidad a una campaña de ocultación de la IOTA (*International Occultation Timing Association*), es necesario que los tiempos de desaparición y reaparición se proporcionen con precisión suficiente, actualmente del orden de decenas de milisegundos o incluso mejor. Para ello es necesario disponer de una base de tiempos basada en GPS [3] para sincronizar el reloj del equipo de captura. En este contexto, nos referiremos al *equipo de captura* como el ordenador, normalmente portátil, con el que se realiza la captura de vídeo a alta tasa de fotogramas o *frames* usando una cámara CMOS o de vídeo analógico y su correspondiente dispositivo digitalizador.

El presente trabajo propone una solución adicional a las ya usadas habitualmente en este ámbito. Consiste en conectar un módulo receptor GPS de bajo coste con interfaz serie UART y señal PPS a través de un conversor USB-serie, transportando la señal PPS a través de una de las líneas de control. Se proporcionan los pasos necesarios tanto para su implementación hardware como la instalación y configuración del software necesario tanto en entornos Linux como Windows. Seguidamente se detalla el procedimiento de calibración del desfase y se analizan los resultados de las pruebas realizadas para evaluar la precisión de la solución.

Los scripts desarrollados para analizar el ajuste necesario para la calibración se encuentran disponibles en el repositorio GitHub [1].

2. Sincronización de tiempo

La sincronización de reloj es un problema de ingeniería enfocado en la coordinación de los relojes de distintos sistemas. En el ámbito que nos ocupa, no sólo es relevante que los relojes de los observadores estén coordinados entre sí, sino además que lo estén respecto de una referencia estándar absoluta como es UTC (*Coordinated Universal Time*), que permita relacionar las observaciones realizadas con las efemérides de los cuerpos observados.

En el ámbito de las redes informáticas, la sincronización de tiempo se implementa mediante protocolos de comunicaciones especialmente diseñados para tal fin, tales como NTP y PTP. NTP (*Network Time Protocol*) está ampliamente extendido, contando con implementaciones para esencialmente todos los sistemas operativos existentes. Habitualmente se implementa bajo una arquitectura cliente-servidor, de forma que varios servidores actúan como referencia de tiempo que cada cliente consulta para derivar una estimación de la hora, habitualmente con una exactitud del orden de milisegundos, más que suficiente para muchas aplicaciones.

PTP (*Precision Time Protocol*) busca una mayor exactitud, del orden de nanosegundos o incluso mejor, necesaria para aplicaciones críticas (sistemas financieros, automatización industrial, operación de redes 5G, proyectos científicos, etc.), mediante el uso de marcas de tiempo insertadas por hardware directamente en las tarjetas de red (NIC, *Network Interface Card*), que además incorporan osciladores de alta precisión. Actualmente, esta funcionalidad sólo está implementada en tarjetas de red para servidores y estaciones de trabajo, no estando presente en las tarjetas de red usadas en los equipos de captura, por lo que no es viable en nuestro caso.

En el ámbito que nos ocupa, que conlleva la realización de observaciones en campo a lo largo de la banda de sombra de la ocultación, en general no tendremos asegurada la conectividad de red. Por ello, la solución más sencilla y económica es usar un receptor GPS, que además presenta mayor exactitud temporal que la conseguida generalmente con NTP. Sin embargo, tanto para la gestión de la sincronización horaria en el ordenador, como para el proceso de calibración de desfase que debemos realizar sobre la hora obtenida por el GPS y que describimos más adelante en este artículo, necesitamos tener desplegado un servicio de hora basado en NTP en el equipo.

2.1. Conceptos fundamentales de NTP

NTP define tanto un protocolo de comunicaciones, orientado a capturar marcas de tiempo tanto del cliente como del servidor, como los algoritmos necesarios para derivar los parámetros de ajuste del reloj local del cliente para su sincronización con el del servidor.

El intercambio de mensajes NTP entre cliente y servidor, así como el fundamento básico para derivar a partir de los mismos tanto el desfase (*offset*) entre sus relojes como el retardo de ida y vuelta (*round-trip delay* o simplemente *delay*) debido a la red, se resume a continuación de manera muy simplificada [6][7][8].

El cliente envía un mensaje NTP insertando la marca de tiempo (*timestamp*) correspondiente al instante de envío, t_0, basada en su reloj local, figura 1. El mensaje se recibe en el servidor en el instante t_1 según su propio reloj. En el caso ideal en que ambos relojes ya estuviesen sincronizados, la diferencia $t_1 - t_0$ correspondería a la latencia introducida por la red de comunicaciones, λ. Sin embargo, en la práctica, ambos relojes presentarán un desfase θ, de modo que $t_1 - t_0 = \lambda + \theta$. Aquí, θ se considera positivo si el reloj del servidor adelanta al del cliente y negativo en caso contrario. Tras procesar el mensaje, el servidor responde enviando un nuevo mensaje, en el que inserta el *timestamp* t_2, que el cliente recibirá en t_3. Suponiendo que la red presenta simetría, es decir, la latencia de red para este camino de vuelta es igual que para el de ida, y que el desfase entre los relojes se mantiene constante durante este intercambio de mensajes, tendremos que $t_3 - t_2 = \lambda - \theta$, ya que ahora el desfase afecta en sentido opuesto.

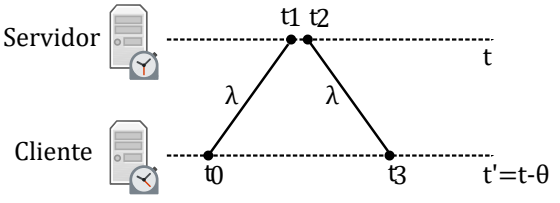

Figura 1. *Esquema de envío de mensajes NTP entre cliente y servidor.*

Sumando ambas ecuaciones obtenemos la latencia de red, $2\lambda = (t_1 - t_0) + (t_3 - t_2)$. Restándolas podemos aislar el desfase, $2\theta = (t_1 - t_0) - (t_3 - t_2)$. Dado que, en lugar de la latencia λ, habitualmente se suele usar el retardo de ida y vuelta (*round-trip delay*), δ, que en caso simétrico es $\delta = 2\lambda$, quedaría:

$$\delta = (t_1 - t_0) + (t_3 - t_2)$$
$$\theta = \frac{(t_1 - t_0) - (t_3 - t_2)}{2}$$

El cliente mide estos parámetros por **sondeo periódico** sobre todos los servidores que tenga configurados. Se aplica un algoritmo de **selección** que descarta las fuentes que no se consideran lo suficientemente fiables, *falsetickers*. Sobre las fuentes supervivientes, *truechimers*, o el subconjunto más representativo de las mismas, se usa un algoritmo de **combinación** que trata de obtener la mejor estimación del desfase a aplicar sobre el reloj local. Este ajuste generalmente se realiza de forma gradual, proceso que se conoce como **disciplinar** el reloj. Todo el mecanismo descrito conforma un bucle de control retroalimentado que idealmente debe converger, coordinando el reloj del cliente con los servidores.

Por tanto, en un equipo dotado de un servicio NTP, cabe distinguir entre los siguientes tipos de relojes:

- **Reloj NTP**: reloj virtual, interno al servicio NTP, basado en el reloj del sistema y las fuentes externas de tiempo configuradas. Mantiene la mejor estimación posible de la hora real.

- **Reloj del sistema**: reloj software mantenido por el núcleo (*kernel*) del sistema operativo, en base al oscilador del sistema. Es el reloj que "ven" las aplicaciones. En equipos que dispongan del servicio NTP, este reloj es ajustado gradualmente ("*disciplinado*") a partir de la hora del reloj NTP.
- **Reloj en tiempo real (RTC)**: reloj de hardware que mantiene la hora cuando el ordenador está apagado. Permite inicializar el reloj del sistema en el arranque.

NTP define un sistema jerarquizado de fuentes de tiempo. Cada nivel se conoce con el nombre de *stratum*. El ***stratum* 0** corresponde a fuentes de tiempo tales como relojes atómicos, receptores GPS, relojes de radio, etc. Este tipo de fuente también recibe el nombre de **reloj de referencia**, *refclock*. Los servidores NTP conectados directamente a los relojes de referencia pertenecen al ***stratum* 1**. Los servidores NTP que se sincronizan a través de la red con servidores NTP *stratum* 1 pertenecen al ***stratum* 2**, y así sucesivamente.

En este punto conviene aclarar que un ordenador que tenga desplegado un servicio NTP no tiene por qué desempeñar el rol de servidor NTP. Es decir, este servicio permite al ordenador también funcionar simplemente como cliente NTP.

2.2. Implementaciones de NTP

Actualmente existen varias implementaciones de NTP. La más destacable, por ser la desarrollada por la *Network Time Foundation*, entidad a cargo de las revisiones del propio protocolo, se denomina **ntpd**[1], siendo la implementación de referencia. Está disponible virtualmente en todos los sistemas operativos de uso general. En particular, es la única disponible en Windows, de la mano del fabricante *Meinberg*, que se encarga de compilar los fuentes y de liberar un instalador [22].

Otra implementación, más reciente y de uso muy extendido, es **chrony**, que incorpora algunas mejoras respecto de la implementación de referencia, si bien está centrada principalmente en sistemas Linux. A continuación se resumen algunas de sus características principales:

- chrony se adapta con mayor agilidad a cambios rápidos de frecuencia de reloj que ntpd [9].
- Al estar continuamente monitorizando la deriva de la frecuencia de reloj del sistema respecto del reloj NTP (*frequency tracking*) [21], chrony puede extrapolar las correcciones necesarias en entornos donde el acceso a la referencia de tiempo es intermitente. En estos casos ntpd no resulta efectivo ya que requiere un sondeo regular, y una vez recuperada la conectividad con la referencia, el ajuste de hora puede llevar bastante tiempo [9] .
- De la misma forma, chrony también mide la deriva del reloj RTC, por lo que tras el arranque del sistema, la hora se puede ajustar en base a una versión corregida del reloj RTC [21][9].
- El algorimo de combinación de chrony tiene en cuenta no sólo las medidas de desfase de las fuentes *truechimers*, sino también las medidas de frecuencia del reloj de sistema [21]. La disciplina de reloj se aplica independiemente tanto en fase como en frecuencia [9].

En el ámbito de este estudio, donde habitualmente el equipo de captura no está siempre encendido y con conexión a la referencia horaria, estas características hacen que chrony sea especialmente interesante, ya que tanto el ajuste de deriva del reloj RTC durante el arranque como el ajuste de deriva del reloj del sistema permiten una puesta en marcha de la sincronización de forma más ágil así como mayor estabilidad en caso de pérdida de conectividad de red o señal GPS. Sin embargo, como se ha mencionado anteriormente, sólo es aplicable a Linux. En el caso de Windows, la única opción es recurrir a la implementación de ntpd de Meinberg.

[1] Si bien el proyecto se denomina `ntp`, en este documento nos referiremos a esta implementación con el nombre del programa, `ntpd`, para evitar confusión con el nombre del protocolo.

3. Sistemas de navegación por satélite

El Sistema Global de Navegación por Satélite (GNSS, *Global Navigation Satellite System*) es el término genérico para tecnologías que usan redes de satélites para determinar la posición, velocidad y tiempo en cualquier lugar del mundo [10]. Aunque en el uso popular los conocemos como "GPS", esta denominación corresponde específicamente al GNSS desplegado por EE.UU., siendo también GNSSs los sistemas Galileo (Europa), GLONASS (Rusia) y BeiDou (China). Aquí usaremos "GPS" de forma genérica para referirnos a cualquier GNSS.

En nuestro ámbito es de gran interés, no sólo por informar sobre la posición con una incertidumbre de tan sólo 2-10 metros, algo de gran interés para la observación de ocultaciones, sino por proporcionar un reloj de referencia UTC de gran precisión, del orden de decenas de nanosegundos.

3.1. Conceptos fundamentales

La radionavegación por satélite se basa en el cálculo de una posición sobre la superficie terrestre midiendo las distancias de un mínimo de tres satélites de posición conocida ("2D fix"). Un cuarto satélite aporta información suficiente para determinar también la altitud ("3D fix"). El receptor GPS capta los mensajes de sincronización emitidos por los satélites, que contiene la posición del satélite y la marca de tiempo en que fue enviado [10].

La precisión de la posición depende de la exactitud de la información de tiempo. Solo los relojes atómicos proveen la precisión requerida, del orden de nanosegundos, por ello son los que se usan a bordo de los satélites GPS. El receptor compara la hora codificada en la transmisión con el instante de la recepción medido por un reloj interno, de forma que se mide el *tiempo de vuelo* de la señal desde el satélite [10].

3.2. Implementaciones hardware para posicionamiento

Los receptores GPS presentan dos rangos de exactitud: el estándar, 2-10 metros, y el de alta precisión, 1-2 centímetros. Aunque existen módulos especializados en sincronización horaria, que emplean tecnologías como SBAS (Satellite-Based Augmentation System) para compensar el retardo ionosférico (usando estaciones en tierra para monitorizarlo) [11], no se consideran adecuados para este caso de uso, ya que estos módulos son en torno a diez veces más caros y la precisión adicional que ofrecen no sería aprovechada por los servicios basados en NTP.

El receptor GPS seleccionado para este proyecto es un módulo de bajo coste, Beitian BE-280, basado en chip u-blox M10050, y dispone de antena integrada, presentando un formato muy compacto (28 x 28 x 8.4 mm), figura 2. Es compatible con GPS, Galileo y BeiDou (BDS). Tiene una precisión de hasta 2 metros en posicionamiento y de 30 ns en tiempo [16].

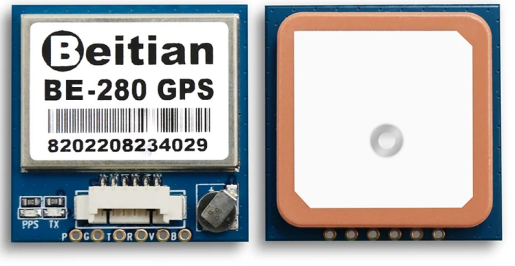

Figura 2. *Módulo Beitian BE-280 usado en las pruebas.*

3.3. Interfaz de comunicaciones

Los receptores GPS suelen integrar un interfaz UART (*Universal Asynchronous Receiver Transmitter*) de tipo TTL para comunicarse con un ordenador. Los mensajes se envían secuencialmente bit a bit, a una velocidad configurable (expresada en bits por segundo, *bps*), agrupados por bytes. A cada byte le precede una marca de inicio, y le sucede una marca de fin. "TTL" hace referencia a los niveles de tensión usados para codificar los bits, en este caso 0 y 5 voltios. El interfaz dispone de líneas de comunicación separadas para recepción y transmisión de datos, denominadas RX y TX. Estrictamente, el receptor GPS sólo necesitaría la línea de transmisión, sin embargo, es frecuente usar ambas ya que eso permite al ordenador enviar comandos al receptor GPS, generalmente para ajustar la configuración del mismo.

También son frecuentes interfaces I^2C (*Inter-Integrated Circuit*), SPI (*Serial Periphere Interface*) e incluso CAN (*Controller Area Network*), si bien estos están orientados a sistemas empotrados e industriales.

Los mensajes enviados por el receptor GPS suelen estar en formato NMEA 0183. Se trata de un protocolo que define mensajes en texto plano, de entre los que cabe destacar GPRMC (*GPS - Recommended Minimum Specific Data*), que incorpora la información sobre posición, velocidad y tiempo. Algunos fabricantes incorporan protocolos propietarios, como es el caso de u-blox, que define el protocolo binario UBX. Al ser binario, permite mayor densidad de información: el equivalente a GPRMC sería UBX-NAV-PVT, que ofrece además una estimación del error de la geolocalización, el vector de velocidad y su incertidumbre, etc.

En cualquier caso, de cara a ofrecer una referencia horaria, se plantea el problema de que la marca horaria enviada en el mensaje al ordenador tarda un cierto tiempo en recibirse, que además depende también del tiempo que tarde el receptor en calcular la solución de geolocalización y tiempo. Esto añade un retardo considerable y lo que es peor, una incertidumbre respecto a qué instante, medido con el reloj de sistema, correspondía a la marca temporal recibida en el mensaje.

Por este motivo, los receptores GPS suelen incorporar una línea de salida adicional, denominada *Pulse-Per-Second*, PPS ó 1PPS, por la que envían un pulso de 100 ms (típ.) cuyo flanco ascendente coincide con el inicio de cada segundo. De esta forma, el ordenador puede capturar la marca de tiempo de su reloj de sistema cuando llega este flanco, y etiquetarla con la hora del GPS cuando llegue el mensaje GPRMC (o similar).

Existen en el mercado receptores GPS con interfaz USB, sin embargo, la señal PPS no está disponible de ninguna forma, por lo que no son recomendables para este caso de uso.

3.4. Software para comunicación con receptores GPS

`ntpd` ya incluye un driver NMEA, por lo que no es necesario software adicional. En el caso de Linux, además tenemos la opción de usar el servicio `gpsd`, que además de NMEA también soporta UBX. De hecho, `chrony` no dispone de drivers sino que se integra con `gpsd` (a través de un *socket* local) para el caso en que queremos usar relojes GPS como referencia.

La ventaja en este caso es que `gpsd` también gestiona la geolocalización, de forma que cualquier aplicación puede consultarla a través de este servicio en lugar de comunicarse directamente con el receptor GPS. Esto no es posible tan fácilmente en el caso de `ntpd`: al tener capturado el puerto de comunicación con el receptor GPS, la única forma de acceder a la geolocalización es usar el comando `ntpq -c clockvar`, cuya salida incluye el último mensaje NMEA recibido.

4. Bases de tiempo para ocultaciones

Actualmente, todas las soluciones para implementar una base de tiempos de precisión suficiente para la observaciones de ocultaciones usan señales GPS. Se pueden resumir en las siguientes [3]:

- Cámara CMOS con obturador global y GPS integrado. Probablemente la solución idónea: los *frames* capturados directamente se etiquetan con las marcas de tiempo obtenidas por GPS, todo ello en la propia cámara.
- Insertador de tiempo en vídeo. Usado con cámaras de salida de vídeo analógica para sobreimpresionar la marca de tiempo, obtenido mediante GPS, sobre cada *frame* de vídeo, antes de ser procesado por la tarjeta capturadora para digitalizarlo.
- Servidor dedicado NTP basado en GPS. Muy extendida en estos últimos años, consiste en usar un SBC (*Single Board Computer*), p.ej. Raspberry Pi, al que se conecta un receptor GPS con UART y PPS, y en el que se despliega un servicio NTP[2]. Es sencilla de implementar y de bajo coste, y obtiene una precisión por debajo del milisegundo [12]. El equipo de captura debe configurarse para usar este servidor como referencia principal, y ambos deben estar conectados preferiblemente por ethernet.
- *Flasher*. Dispositivo que emite destellos regulares antes y después del evento a observar, en instantes conocidos. El dispositivo se instala físicamente de manera que los destellos se registran en la cámara que captura el evento, y son analizados posteriormente mediante software para asociar a cada *frame* su marca de tiempo, interpolada a partir de aquellas en las que está presente el destello. Dentro de esta categoría existe una aplicación para Android, denominada *Occult Flash Tag*, que emite los destellos con el LED del *smartphone* (aunque usa NTP para la sincronización horaria).

5. Descripción de la solución propuesta

La solución que aquí se propone, que no pretende sustituir a las ya mencionadas, es la de conectar un módulo receptor GPS de bajo coste, con interfaz UART, usando un conversor a USB y transportando la señal PPS a través de una de las líneas de control. No se trata de una propuesta original, ya que ha sido implementada en unos pocos productos comerciales, pero que actualmente resultan muy difíciles de adquirir. Esta solución se ha descartado de forma recurrente en parte por este motivo, pero sobre todo, por las dudas que surgen al considerar que el transporte USB pueda afectar negativamente a su precisión.

La aportación del presente trabajo es compilar ordenadamente y de forma contrastada todos los pasos necesarios para su implementación, y especialmente, mostrar los resultados de las pruebas realizadas para determinar la precisión alcanzable.

Los siguientes apartados detallan el **montaje de hardware** realizado y la **configuración base** para probar el correcto funcionamiento del dispositivo y del software, tanto en Linux como en Windows. Los pasos necesarios para la **calibración de la fuente GPS** son los siguientes:

1. Medir el desfase instrumental que presenta el fuente de tiempo GPS desde la perspectiva del servicio NTP. Este desfase se debe principalmente a las latencias de todas las capas de software intermedias. Para realizar esta medición usaremos múltiples fuentes NTP stratum 1 para construir una referencia lo más fidedigna posible. En este sentido es importante que la conexión a Internet que usemos sea lo más estable posible, por lo que es recomendable disponer de una línea de fibra óptica y conectar el equipo de captura mediante ethernet al router, evitando usar Wi-Fi.

[2]Raspberry Pi 5 añade soporte hardware para marcaje de tiempo, lo que permite desplegar un servicio PTP. No obstante, la NIC no dispone de entrada PPS dedicada por lo que la señal PPS proveniente del receptor GPS debe ir conectada a un pin de entrada/salida de propósito general (GPIO) y ser gestionada por software. A pesar de ello, experimentos recientes [13] sugieren que los resultados pueden ser mejores que con NTP. En cualquier caso, las NIC de equipos portátiles no soportan marcaje de tiempo por hardware y el cliente PTP de Windows y su interacción con los drivers todavía necesita madurar [15].

2. Una vez obtenida una estimación de dicho desfase, aplicaremos una corrección del mismo en la configuración del servicio NTP. Gracias a ello, sólo quedará una fluctuación residual o *jitter*, que nos indicará la precisión alcanzable.

La estimación del desfase se realiza con un script que analiza los logs de estadísticas generados por el servicio NTP. Se encuentra disponible en el repositorio asociado al artículo [1].

5.1. Montaje hardware

La conexión con el ordenador se ha realizado con un módulo USB-serie del fabricante FTDI, basado en el chip FT232RL[3]. Se trata de un módulo de precio muy reducido y fácilmente localizable en el mercado. Por otra parte, este módulo está plenamente soportado en Linux y Windows (aunque en este último caso requiere instalar manualmente los drivers VCP disponibles en la web del fabricante[4]). Sin embargo, el motivo principal para seleccionar este fabricante es que, según la documentación oficial [17], la latencia del driver es mínima cuando se detecta un cambio en las líneas de control. Esto sugiere que quizá sea posible transportar la señal PPS a través de USB sin añadir excesiva incertidumbre[5].

En particular, conectaremos PPS con la línea de control DCD (*Data Carrier Detect*), ya que es la habitualmente usada con `gpsd` y el driver NMEA de `ntpd`. Para la recepción de mensajes conectaremos la línea TX del módulo BE-280 con RX de FT232RL, e incluiremos también la conexión recíproca (RX BE-280 con TX FT232RL) para tener la posibilidad de controlar el módulo GPS. Por último, y dado que el consumo de éste es muy bajo, podemos alimentarlo directamente desde el módulo FT232RL, asegurándonos que el selector de tensión (en caso de existir) esté en la posición 5V. El conexionado físico, que requiere soldar varios cables, se muestra en la figura 3.

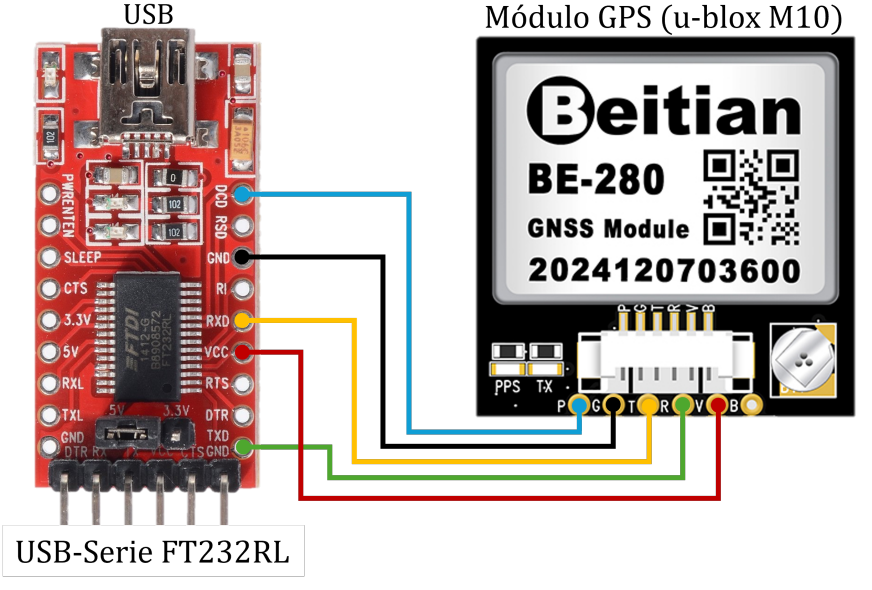

Figura 3. *Esquema de conexionado.*

[3]La medición de desfase se ha probado también con un módulo basado en el chip FT232H, que implementa USB 2.0 Hi-Speed, si bien los resultados han sido similares

[4]https://ftdichip.com/drivers/vcp-drivers/

[5]Existe cierto debate respecto a la conveniencia de cambiar la configuración del *latency timer* del *driver* FTDI, reduciendo su valor (por defecto, 16 ms) al mínimo (1 ms). Este temporizador está ligado al buffer de recepción, y sirve para poner un límite a la espera de la llegada de bytes a través del puerto serie. Dado que no se usa en el caso de las líneas de control, no tiene efecto en la solución propuesta.

Una cuestión a tener en cuenta es que USB 3 produce emisiones de radio en el rango de frecuencias que se usan para la recepción de señales GPS [18]. Por ello es necesario alejar en lo posible el módulo, que lleva la antena integrada, del equipo de captura y de la cámara.

Para proteger el conjunto es conveniente preparar una carcasa apta para intemperie, teniendo en cuenta que la antena debe quedar orientada hacia el cielo. La figura 4 muestra una carcasa impresa en 3D atada a una pata del telescopio, alejando el receptor de posibles interferencias. La parte que queda horizontal contiene el módulo GPS con la antena orientada hacia arriba. El diseño está disponible en el repositorio [1].

Figura 4. *Dispositivo alojado en carcasa impresa en 3D.*

5.2. Implementación en Linux

Las instrucciones de instalación y configuración que se desarrollan en esta sección han sido probadas en una instalación de Ubuntu 22.04 LTS y posteriormente validadas sobre Ubuntu 24.04 LTS.

5.2.1. Configuración base con `gpsd` y `chrony`

En primer lugar prepararemos la configuración de `gpsd` [19] y validaremos que se comunica correctamente con el hardware. Instalamos los paquetes de software necesarios y preparamos el usuario de sistema para el servicio:

```
$ sudo apt install gpsd gpsd-clients pps-tools
$ sudo dpkg-reconfigure gpsd
Creating/updating gpsd user account...
gpsd.socket is a disabled or a static unit not running, not starting it.
gpsd.service is a disabled or a static unit not running, not starting it.
gpsd.socket is a disabled or a static unit not running, not starting it.
```

Al conectar el módulo GPS, podemos comprobar a qué nodo de dispositivo se le asocia en el sistema:

```
$ sudo dmesg
[...]
... usb 1-5: FTDI USB Serial Device converter now attached to ttyUSB0
[...]
```

En este caso es `/dev/ttyUSB0`. En este punto podemos comprobar si `gpsd` reconoce y se comunica con el dispositivo, lanzando el servicio manualmente en modo de depuración:

```
$ sudo gpsd -D 5 -N -n /dev/ttyUSB0
gpsd:INFO: launching (Version 3.25, revision 3.25)
gpsd:INFO: starting uid 0, gid 0
gpsd:INFO: Command line: gpsd -D 4 -N -n /dev/ttyUSB0
gpsd:INFO: listening on port gpsd
gpsd:PROG: NTP:SHM: shmat(32769,0,0) succeeded, unit 0
gpsd:PROG: NTP:SHM: shmat(32770,0,0) succeeded, unit 1
gpsd:PROG: NTP:SHM: shmat(32772,0,0) succeeded, unit 2
gpsd:PROG: NTP:SHM: shmat(32774,0,0) succeeded, unit 3
gpsd:PROG: NTP:SHM: shmat(32775,0,0) succeeded, unit 4
gpsd:PROG: NTP:SHM: shmat(32776,0,0) succeeded, unit 5
gpsd:PROG: NTP:SHM: shmat(32777,0,0) succeeded, unit 6
gpsd:PROG: NTP:SHM: shmat(32779,0,0) succeeded, unit 7
gpsd:PROG: NTP:SHM: shmat(32780,0,0) succeeded, unit 8
gpsd:PROG: NTP:SHM: shmat(32781,0,0) succeeded, unit 9
gpsd:PROG: NTP:SHM: shmat(32782,0,0) succeeded, unit 10
gpsd:PROG: NTP:SHM: shmat(32784,0,0) succeeded, unit 11
gpsd:PROG: successfully connected to the DBUS system bus
gpsd:PROG: SHM: shmget(0x47505344, 29240, 0666) for SHM export succeeded
gpsd:PROG: SHM: shmat() for SHM export succeeded, segment 32785
gpsd:INFO: stashing device /dev/ttyUSB0 at slot 0
gpsd:PROG: CORE: open_device(/dev/ttyUSB0) fd -1
gpsd:PROG: CORE: gpsd_activate(/dev/ttyUSB0, 2) fd -1
gpsd:PROG: CORE: no /etc/gpsd/device-hook present, skipped running ACTIVATE hook.
    No such file or directory(2)
gpsd:PROG: CORE: gpsd_open(/dev/ttyUSB0) fd -1
gpsd:PROG: SER: gpsd_serial_open(/dev/ttyUSB0) sourcetype 3 fd -1
gpsd:INFO: SER: opening GPS data source type 3 at '/dev/ttyUSB0'
gpsd:INFO: SER: fd 7 current speed 38400, 8N1
gpsd:PROG: CORE: Probing "Garmin USB binary" driver...
[...]
gpsd:PROG: CORE: selecting u-blox driver...
gpsd:INFO: CORE: /dev/ttyUSB0 identified as type u-blox, 1 sec
gpsd:INFO: CORE: /dev/ttyUSB0 38400bps
[...]
```

Este comando genera mucha información por lo que debemos cancelarlo tras pocos segundos con `Ctrl+C` para poder analizar la salida. Tras la inicialización del servicio, `gpsd` trata de determinar el tipo de receptor GPS y la velocidad del puerto serie. En este caso determina que se trata de un módulo u-blox (`gpsd:INFO: CORE: /dev/ttyUSB0 identified as type u-blox`) que está enviando mensajes a 38400 bps (`gpsd:INFO: CORE: /dev/ttyUSB0 38400bps`).

Es importante que confirmar en este punto que se está usando *kernel PPS* (KPPS), es decir, que el seguimiento de la señal PPS se está realizando mediante un módulo de *kernel* del sistema operativo, lo que proporciona, en condiciones normales, menor latencia y *jitter*. Para ello debemos localizar, en la salida del comando anterior, el siguiente mensaje:

```
[...]
gpsd:INFO: KPPS:/dev/ttyUSB0 kernel PPS will be used
[...]
```

En caso contrario, podemos investigar si la señal PPS está llegando por otra línea de control y por ello no es detectada por `gpsd`. En este caso, `ppscheck` avisa de que `/dev/ttyUSB0` no es un dispositivo KPPS, ya que se trata de un puerto serie virtual, y crea el dispositivo KPPS `/dev/pps0` asociado a éste. En la salida del comando debemos ver que aparece, cada segundo, los eventos de `clear` y `assert`, es decir, los cambios de nivel de la señal PPS, a través de la línea D**CD**, `TIOCM_CD`.

```
$ sudo ppscheck /dev/ttyUSB0
WARNING: time_pps_create(/dev/ttyUSB0)) failed: Operation not supported(95)
WARRING: /dev/ttyUSB0 does not appear to be a KPPS device
INFO: matching /dev/pps0 opened

# Src  Seconds             Signals

  KPPS 0.000000000  assert 0
  KPPS 1759603810.001042068 clear  1
  TTY  1759603810.001276456
  KPPS 1759603810.101000282 assert 1
  TTY  1759603810.101121817 TIOCM_CD

  KPPS 1759603811.001014980 clear  2
  TTY  1759603811.001167710
  KPPS 1759603811.100972784 assert 2
  TTY  1759603811.101108597 TIOCM_CD
[...]
```

A continuación debemos crear el fichero de configuración para `gpsd` a partir de la información obtenida, `/etc/default/gpsd`:

```
# Devices gpsd should collect to at boot time.
# They need to be read/writeable, either by user gpsd or the group dialout.
DEVICES="/dev/ttyUSB0"

# Other options you want to pass to gpsd
GPSD_OPTIONS="-n -s38400"

# Automatically hot add/remove USB GPS devices via gpsdctl
USBAUTO="false"
```

Seguidamente arrancamos el servicio:

```
$ sudo systemctl start gpsd
```

Y comprobamos que está recibiendo correctamente la solución de posición, velocidad y hora, véase la figura 5.

```
$ cgps -s -u m  # para salir, pulsar q
```

`gpsd` debe haber podido crear el dispositivo PPS, `/dev/pps0`. Podemos comprobarlo con `ppstest`:

```
$ sudo ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1759606048.101202976, sequence: 9 - clear 1759606049.001205442,
    sequence: 10
source 0 - assert 1759606049.101212416, sequence: 10 - clear 1759606049.001205442,
    sequence: 10
source 0 - assert 1759606049.101212416, sequence: 10 - clear 1759606050.001215413,
    sequence: 11
[...]
```

A continuación instalamos `chrony` y generamos los ficheros de configuración:

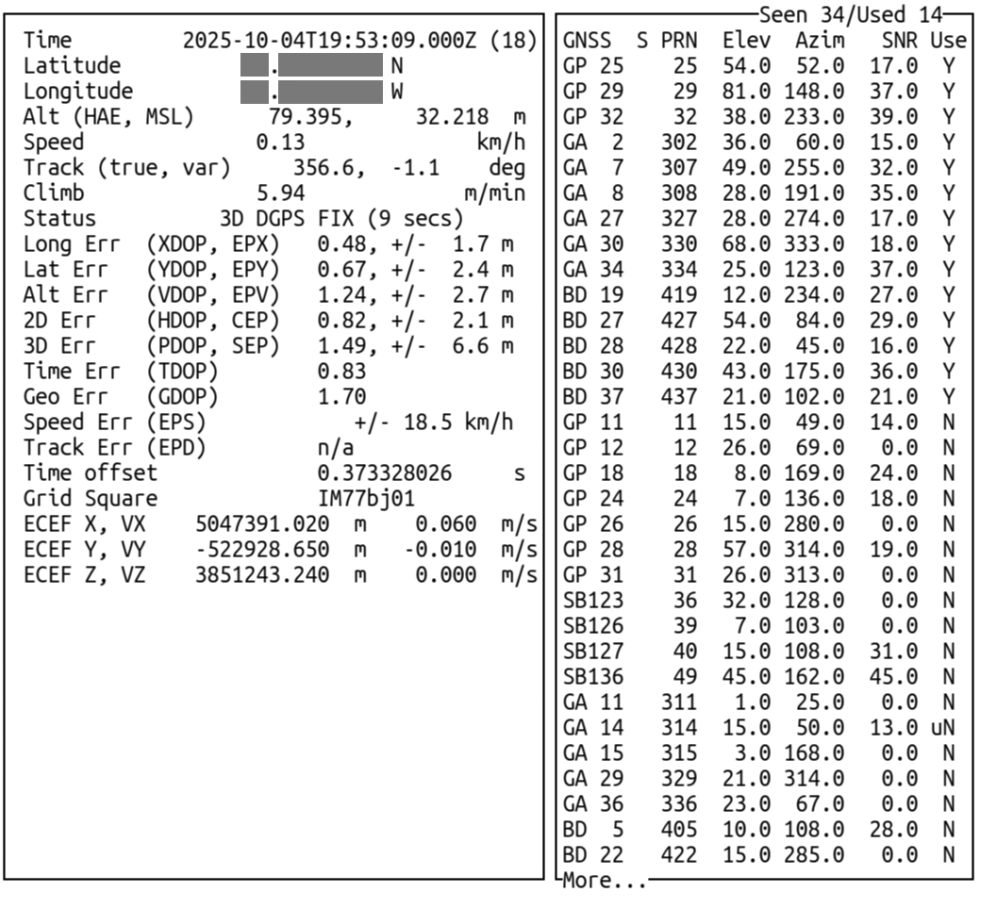

Figura 5. *Información mostrada por el comando* `cgps`.

```
$ sudo apt install chrony

$ echo "server hora.roa.es iburst prefer" | sudo tee
    /etc/chrony/sources.d/st1.sources
$ echo "refclock SOCK /var/run/chrony.ttyUSB0.sock refid GPS" | sudo tee
    /etc/chrony/conf.d/01-gpsd.conf
$ echo "log tracking statistics measurements" | sudo tee
    /etc/chrony/conf.d/02-log.conf
$ echo "logbanner 0" | sudo tee -a /etc/chrony/conf.d/02-log.conf
```

El fichero `st1.sources` contendrá las fuentes NTP, por ahora solo el servidor de ROA (Real Observatorio de la Armada), `hora.roa.es`. En `01-gpsd.conf` estamos indicando que la interacción con `gpsd` se realizará mediante el socket `/var/run/chrony.ttyUSB0.sock`, y que el reloj de referencia asociado tendrá como identificador `GPS`. En `02-log.conf` indicamos que se deben generar distintos tipos de logs, entre ellos el de estadísticas, que nos permitirá estudiar el desfase de la fuente GPS; la directiva `logbanner 0` desactiva la generación repetitiva de los rótulos de las columnas de datos en los ficheros de log, que entorpecería su análisis automatizado.

Reiniciamos los servicios, primero `chrony` para que lea los ficheros de configuración y genere el socket, y a continuación `gpsd` para que lo detecte y haga uso del mismo.

```
$ sudo systemctl daemon-reload # necesario tras modificar ficheros de
    configuracion de chrony
$ sudo systemctl restart chronyd
$ sudo systemctl restart gpsd
```

Al iniciar chrony, éste registra si ha forzado un ajuste "de un salto" al reloj del sistema, p. ej.:

```
$ sudo systemctl status chrony
[...]
...: System clock wrong by -2.308569 seconds
...: System clock was stepped by -2.308569 seconds
```

Para comprobar el estado de chrony podemos usar el cliente chronyc [20], añadiendo los argumentos `tracking`, `sources` y/o `sourcestats` para obtener información detallada sobre la fuente seleccionada como primaria, el resumen de las fuentes y estadísticas sobre las mismas. El parámetro `-v` muestra la leyenda de las columnas para estas dos últimas opciones.

```
$ chronyc -n -m tracking 'sources -v' 'sourcestats -v'
Reference ID  : 47505300 (GPS)
Stratum       : 1
Ref time (UTC) : Sat Oct 04 20:18:34 2025
System time   : 0.000000075 seconds fast of NTP time
Last offset   : -0.000000140 seconds
RMS offset    : 0.000000281 seconds
Frequency     : 18.246 ppm slow
Residual freq : -0.000 ppm
Skew          : 0.005 ppm
Root delay    : 0.000000001 seconds
Root dispersion : 0.000016294 seconds
Update interval : 16.0 seconds
Leap status   : Normal

  .-- Source mode '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current best, '+' = combined, '-' = not combined,
| /           'x' = may be in error, '~' = too variable, '?' = unusable.
||                                                 .- xxxx [ yyyy ] +/- zzzz
||     Reachability register (octal) -.     |      |  xxxx = adjusted offset,
||     Log2(Polling interval) --.  |        |      |  yyyy = measured offset,
||                              \     |     |      |  zzzz = estimated error.
||                               |    |     |       \
MS Name/IP address     Stratum Poll Reach LastRx Last sample
===============================================================================
#* GPS                     0   4   377    13   -883ns[-1021ns] +/- 1638ns
^- 150.214.94.5             1   7   377    97   +382us[ +382us] +/- 17ms

                          .- Number of sample points in measurement set.
                         /    .- Number of residual runs with same sign.
                        |    /    .- Length of measurement set (time).
                        |   |    /      .- Est. clock freq error (ppm).
                        |   |   |      /          .- Est. error in freq.
                        |   |   |     |          /       .- Est. offset.
                        |   |   |     |         |        |   On the -.
                        |   |   |     |         |        |   samples. \
                        |   |   |     |         |        |            |
Name/IP Address         NP  NR  Span  Frequency  Freq Skew  Offset  Std Dev
==============================================================================
GPS                     46  21   721     -0.000     0.005    -0ns  2456ns
150.214.94.5             9   4  1032     -0.074     0.571  +640us   116us
```

Vemos que se ha seleccionado la fuente GPS, de tipo `local clock`, como principal (`current best`), mientras que el servidor de ROA se queda fuera del algoritmo de combinación (`not combined`), ya que comparativamente presenta peores estadísticas de desfase (`offset`) e incertidumbre. Destaca el bajo valor de incertidumbre (`Std Dev`, *Standard Deviation*) obtenido, de tan sólo 2456 ns.

No es necesario configurar el socket `chrony.ttyUSB0.sock` explícitamente en `gpsd`, pero para que funcione la comunicación con `chrony` es necesario asegurarse de que éste se ha iniciado antes que `gpsd`, tal como hemos hecho anteriormente de forma manual. El orden de arranque de los servicios se establece como una regla `After` en `gpsd`, que definimos en el fichero `/etc/systemd/system/gpsd.service.d/gpsd-after-chronyd.conf`:

```
[Unit]
After=chronyd.service
```

Finalmente, solo quedaría habilitar el arranque automático de estos servicios:

```
$ sudo systemctl enable chrony
$ sudo systemctl enable gpsd
```

5.2.2. Medición del desfase de la fuente GPS PPS

Modificamos la definición del reloj de referencia GPS para que únicamente sea monitorizado, sin que `chrony` pueda seleccionarlo, editando `/etc/chrony/conf.d/01-gpsd.conf`:

```
refclock SOCK /var/run/chrony.ttyUSB0.sock refid GPS noselect
```

Añadimos un conjunto de servidores NTP stratum 1 que conjuntamente servirán para contruir la hora NTP de referencia sobre la cual medir el desfase de la fuente GPS. En particular, para la prueba realizada, `/etc/chrony/sources.d/st1.sources` se configuró con el siguiente contenido:

```
server hora.roa.es iburst maxpoll 7
server minuto.roa.es iburst maxpoll 7
server hora.cica.es iburst maxpoll 7
server ntps1-0.cs.tu-berlin.de iburst maxpoll 7
server rustime01.rus.uni-stuttgart.de iburst maxpoll 7
server rustime02.rus.uni-stuttgart.de iburst maxpoll 7
server ntp.vsl.nl iburst maxpoll 7
server ntp.fizyka.umk.pl iburst maxpoll 7
server tempus1.gum.gov.pl iburst maxpoll 7
server tempus2.gum.gov.pl iburst maxpoll 7
server time1.stupi.se iburst maxpoll 7
server ntp1.inrim.it iburst maxpoll 7
server ntp1.oma.be iburst maxpoll 7
server time.esa.int iburst maxpoll 7
server time1.google.com iburst maxpoll 7
server time2.google.com iburst maxpoll 7
server time.windows.com iburst maxpoll 7
server time1.apple.com iburst maxpoll 7
server time2.apple.com iburst maxpoll 7
server gbg1.ntp.se iburst maxpoll 7
server mmo1.ntp.se iburst maxpoll 7
server sth1.ntp.se iburst maxpoll 7
server ntp.time.nl iburst maxpoll 7
```

Para que tengan efecto los cambios:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart chronyd
```

El conjunto de servidores NTP elegidos puede tener bastante repercusión en la medición. En general es conveniente seleccionar servidores que tengan conectividad de latencia baja y a ser posible, estable (propia de proveedores de servicio). Por otra parte, debe monitorizarse el número de servidores seleccionados por `chrony`, mediante el comando `chronyc -n sources -v`: si se seleccionase un número

bajo de servidores, la medición podría quedar sesgada. Las fuentes seleccionadas aparecen con el símbolo '+', incluyendo también la principal, '*'.

Los ficheros de log de estadísticas generados por `chrony`, `/var/log/chrony/statistics.log*`, cuyo formato se describe con detalle en [21], sirven de base para el análisis de offset de la fuente GPS. Es conveniente acumular datos durante varios días si es posible para que las estadísticas tengan el menor sesgo posible.

Este análisis se ha implementado con un script Python [1] que obtiene el valor medio de desfase, $\overline{\theta}$, y su desviación estándar, σ, como medida de su dispersión o fluctuación. También se aplica *sigma clipping* (con factor 2.5 sobre la desviación estándar) para que los resultados sean robustos ante la presencia de *outliers*, obteniendo los estadísticos $\overline{\theta}_{CLIP}$ y σ_{CLIP}.

La gráfica de la figura 6 muestra la evolución del desfase durante el intervalo de análisis (más de 8 días) y las estadísticas resultantes. El dato que nos interesa en este paso es el valor de **0.000855 s** para el desfase, que corregiremos en el apartado siguiente. Es notable que su desviación estándar es de tan sólo 94 μs (70 μs en el caso de *sigma clipping*), lo que nos indica que el valor de desfase obtenido tiene poca incertidumbre.

5.2.3. Corrección del desfase de la fuente GPS PPS

Eliminamos `noselect` y añadimos el valor de desfase obtenido durante el análisis al argumento `offset`, así como la opción `prefer` en `/etc/chrony/conf.d/01-gpsd.conf`:

```
refclock SOCK /var/run/chrony.ttyUSB0.sock refid GPS offset 0.000855 prefer
```

Aplicamos los cambios:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart chronyd
```

Dado que el tiempo de parada es muy breve, `chrony` tarda pocos minutos en volver a converger. La salida típica obtenida al consultar el estado las fuentes muestra que GPS es la fuente principal, con *jitter* de aproximadamente 1.5 μs, inferior al de las fuentes NTP en varios órdenes de magnitud, por lo que el algoritmo de combinación no las considera:

```
$ chronyc -n sources
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
#* GPS                           0   4   377    18   +187ns[ +443ns] +/- 1468ns
^- 194.58.203.20                 1   6   377    37   -141us[ -140us] +/- 31ms
^- 150.214.5.121                 1   6   377    36    +43us[ +44us] +/- 15ms
^- 150.214.94.5                  1   6   377    42   +705us[ +706us] +/- 17ms
^- 150.214.94.10                 1   6   377    36   +752us[ +752us] +/- 17ms
^- 194.58.204.20                 1   6   377    37   -210us[ -210us] +/- 29ms
^- 158.75.5.245                  1   6   377    40    +90us[ +91us] +/- 30ms
^- 94.198.159.10                 1   6   377    35  +1550us[+1550us] +/- 21ms
^- 31.223.173.226                1   6   377    40   +458us[ +459us] +/- 18ms
^- 193.204.114.232               1   6   377    38  -3080us[-3079us] +/- 20ms
^- 193.190.230.105               2   6   377    39  -2457us[-2456us] +/- 48ms
^- 130.149.17.21                 1   6   346   168  +2117us[+2117us] +/- 30ms
^- 129.69.253.1                  1   6   377    41  +1371us[+1372us] +/- 22ms
^- 129.69.253.17                 1   6   377    34   +631us[ +632us] +/- 23ms
^- 194.58.202.20                 1   6   377    36  -1017us[-1016us] +/- 30ms
^- 194.146.251.100               1   6   377    37  -1756us[-1756us] +/- 31ms
^- 194.146.251.101               1   6   377    35  -1182us[-1181us] +/- 32ms
^- 192.171.1.150                 1   6   377    35  +1304us[+1305us] +/- 21ms
^- 51.145.123.29                 3   6   377    36  +1855us[+1856us] +/- 34ms
^- 216.239.35.0                  1   6   377    36  +1240us[+1241us] +/- 15ms
^- 192.36.143.150                1   6   377    35  +1118us[+1119us] +/- 31ms
^- 216.239.35.4                  1   6   377    37   +461us[ +462us] +/- 14ms
```

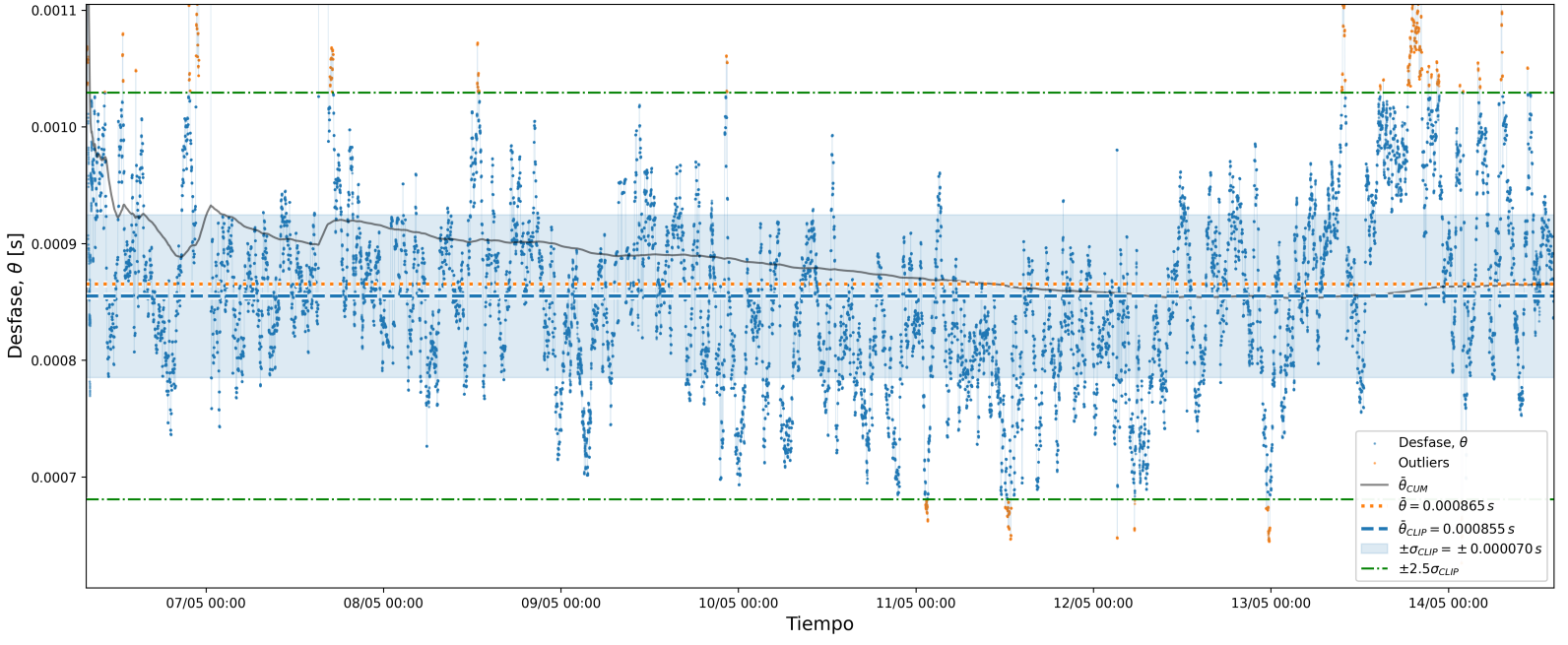

Figura 6. *Evolución del desfase de la fuente GPS durante más de 8 días, medido en base a 21 fuentes NTP stratum 1. Prueba realizada en entorno Linux, con* `gpsd` *y* `chrony`*.*

En cualquier caso, para comprobar si el desfase queda debidamente corregido y obtener una medición fiable del error, debemos debemos dejar transcurrir varias horas antes de realizar un nuevo análisis con el script. Durante este lapso, se puede realizar la prueba de captura descrita para establecer si tiene algún impacto en el *jitter*.

5.3. Implementación en Windows

Para poder usar la señal PPS en Windows es necesario recurrir a la implementación de NTP realizada por el fabricante Meinberg. El software se descarga desde la sección 'NTP for current Windows versions' de su portal web [22]. La instalación es guiada y basta con dejar todas las opciones por defecto, salvo en el paso de creación del fichero de configuración, donde seleccionaremos 'Spain' en el campo 'Want to use predefined public NTP servers', para poder hacer una prueba inicial. El instalador finaliza arrancando el servicio NTP.

Para monitorizar de forma cómoda el servicio, podemos usar la herramienta 'NTP Time Server Monitor' (figura 7), también de Meinberg, disponible en su portal web [23]. Una vez instalada, es necesario ejecutarla *como administrador* si queremos tener control para parar o reiniciar el servicio, cambiar la configuración, acceder al log de eventos, etc.

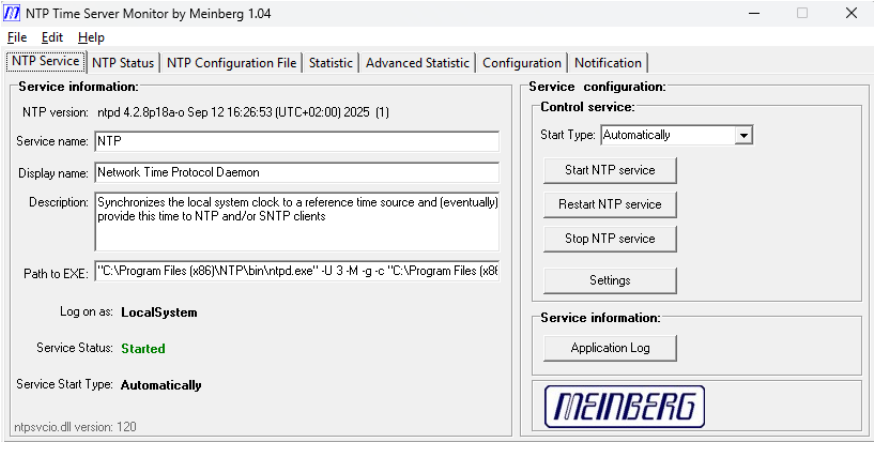

Figura 7. *NTP Time Server Monitor.*

5.3.1. Configuración en Windows con Meinberg NTP

Antes de configurar la fuente GPS debemos habilitar el *driver* PPS [24]. Para ello, debemos ejecutar la herramienta 'Editor de registro' de Windows (`regedit.exe`). En ella, navegamos hasta la entrada:

```
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTP
```

y creamos una entrada de tipo 'cadena múltiple' (`REG_MULTI_SZ`) con clave `PPSProviders` y valor `loopback-ppsapi-provider.dll` (guiones medios, no bajos). Esta nueva configuración requiere reiniciar el servicio NTP para que tenga efecto, cosa que podemos hacer desde 'NTP Time Server Monitor', pestaña 'NTP Service', botón 'Restart NTP Service'. En esta misma pestaña, pulsamos 'Application Log', que creará una nueva pestaña 'NTP Event Log'. En ella debe aparecer el evento:

```
time_pps_create: loaded
    'C:\Program Files (x86)\NTP\bin\loopback-ppsapi-provider.dll'
```

que nos confirma que el *driver* PPS se ha cargado correctamente. Este *driver* se ejecuta en espacio de usuario, con menor nivel de privilegio y prioridad que los procesos de kernel del sistema operativo, lo cual puede afectar en el *jitter* observado [24].

Para configurar la fuente GPS necesitamos saber en primer lugar qué identificador tiene el dispositivo serie asociado y a qué velocidad está configurado. El primer dato se puede obtener del 'Administrador de dispositivos', desplegando la sección 'Puertos (COM & LPT)', donde debe aparecer alguna entrada 'USB Serial Port', indicando el identificador de puerto entre paréntesis, p.ej. 'USB Serial Port (COM12)'. Para conocer la velocidad de transmisión, tendremos que recurrir a la información del fabricante del módulo. En el caso de módulos u-blox como el usado para las pruebas, otra opción es recurrir al software 'u-center 2' [25]: en la sección 'Data sources', podemos configurar la conexión con el módulo pulsando el botón '+' y seleccionando el puerto (COM12 en este caso) y dejando la opción 'Enable autobauding'. Al conectar, podremos ver la velocidad con la que ha conseguido comunicarse con el módulo, figura 8.

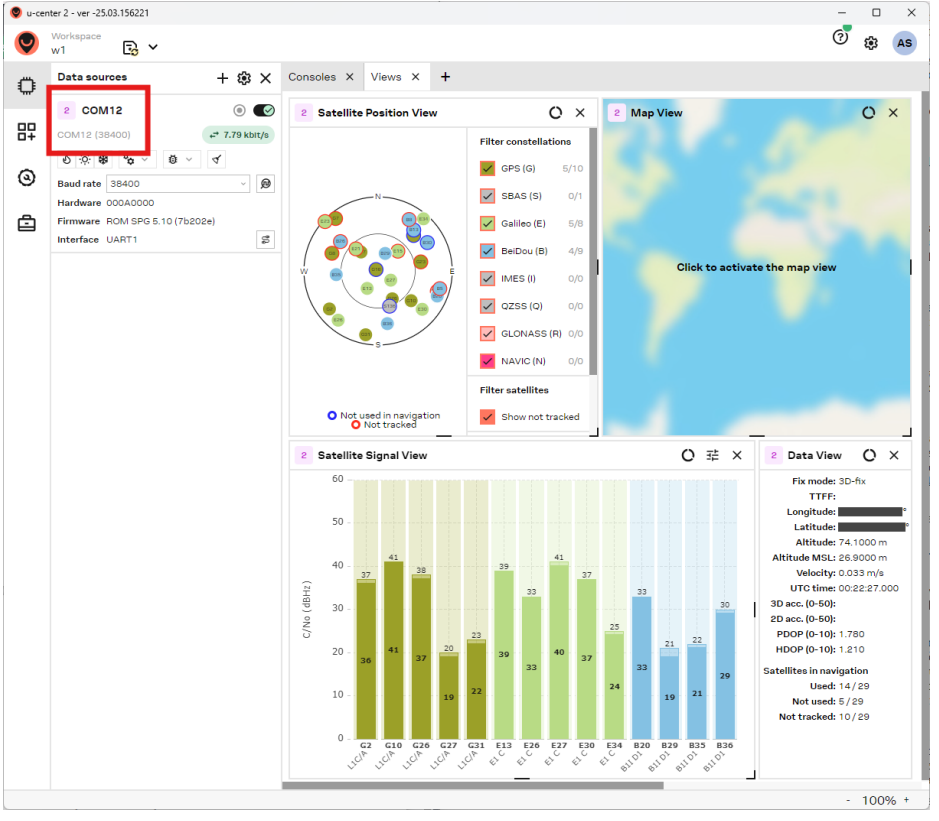

Figura 8. *Uso de u-center 2 para obtener el puerto y velocidad del receptor GPS (solo para dispositivos u-blox).*

A partir de esta información, puerto y velocidad, vamos a contruir los valores, un tanto crípticos, que deberán tener los parámetros de configuración del reloj basado en GPS. En primer lugar, en `ntpd`, el reloj GPS se declara con la directiva `server`, como si se tratase de una fuente NTP, pero usando una dirección IP ficticia basada en el número de puerto: `127.127.20.x`, donde el octeto `20` especifica el driver NMEA [26], y `x` corresponde al número de puerto, es decir, `12` para COM12. El siguiente parámetro a indicar es `mode`, que se construye sumando los códigos correspondientes a las opciones que

queremos habilitar: en nuestro caso, `49`, como suma de `1`, que indica que se procesarán los mensajes NMEA de tipo GPRMC (*Recommended Minimum Data*), y de `48`, que indica que la velocidad es de 38400 bps. Los valores disponibles están documentados en la referencia del driver NMEA [26]. Usaremos un sondeo de 16 ($= 2^4$) segundos, que indicaremos con el valor `4` en los parámetros `minpoll` y `maxpoll`. Para especificar que debe asociarse a esta fuente una señal PPS hay que añadir una directiva `fudge` asociada a la misma dirección IP ficticia, en la que activaremos el parámetro `flag1` (*enable PPS signal processing*) y que más adelante nos permitirá compensar el desfase medido a través del parámetro `offset`. En definitiva, las líneas que debemos añadir al fichero de configuración ya existente, `C:\Program Files (x86)\NTP\etc\ntp.conf`, son las siguientes:

```
# GPS PPS via COM12
#   mode 49 = 1 + 48 = process $GPRMC + linespeed 38400 bps
server 127.127.20.12 mode 49 minpoll 4 maxpoll 4
#   flag1 = PPS, time1 = offset calibration
fudge 127.127.20.12 time1 0.0 flag1 1
```

El fichero puede editarse desde la pestaña 'NTP Configuration File' de la herramienta 'NTP Time Server Monitor', que al salvar nos dará opción a reiniciar el servicio para aplicar los cambios. Hay que recordar que para otro puerto y/o velocidad es necesario revisar la IP ficticia y el valor de `mode` según lo indicado.

Tras el reinicio del servicio, podemos recurrir de nuevo a la herramienta de monitorización, pestaña 'NTP Status', para observar el listado de fuentes y su stratum, frecuencia de sondeo, disponibilidad, etc. En este punto, nos interesa especialmente consultar el estado de la fuentes, en particular de la fuente GPS, que se codifica un carácter: '`o`' nos indicaría que esta fuente está seleccionada y tiene asociada una señal PPS (figura 9), mientras que '`*`' nos indicaría que la señal PPS no se detecta. El significado de los códigos se puede consultar pulsando el botón 'Legend'.

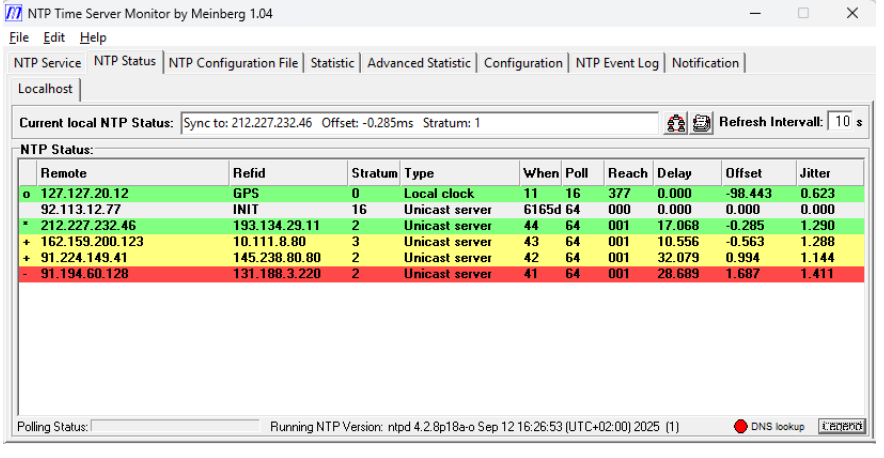

Figura 9. *Fuente GPS seleccionada, verificando que se detecta la señal PPS.*

Es muy posible que la fuente GPS se seleccione por un momento, mostrando el indicador PPS '`o`' (*PPS source of server is selected for synchronization*) y por tanto confirmando que el montaje hardware y la configuración es correcta, pero que unos minutos después quede marcada como *falseticker* (figura 10), '`x`', ya que, al no haber corregido todavía su desfase, la hora que reporta estará en desacuerdo con el resto de fuentes. En cualquier caso, si tenemos dudas, podemos repetir la prueba eliminando todas las fuentes NTP del fichero de configuración (líneas `server` generadas durante la instalación de NTP), dejando solo la fuente GPS.

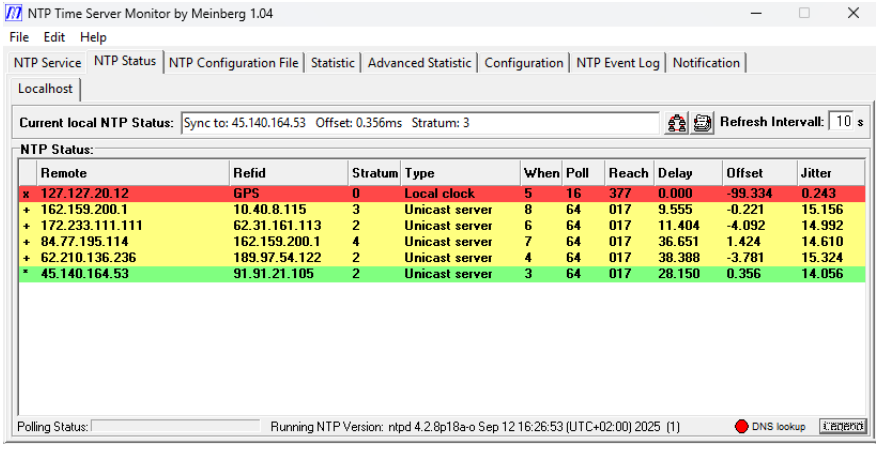

Figura 10. *Fuente GPS marcada como falseticker.*

También es posible consultar el estado de las fuentes con el comando `ntpq -n -p`.

5.3.2. Medición del desfase de la fuente GPS PPS

Al igual que con `chrony`, para la medición del desfase añadimos el parámetro `noselect` a la línea que define la fuente GPS:

```
server 127.127.20.12 mode 49 minpoll 4 maxpoll 4 noselect
```

Igualmente, añadimos un conjunto de servidores NTP stratum 1 para realizar la monitorización. En este caso, indicamos de forma explícita que el algoritmo de selección acepte hasta 20 servidores (`maxclock 20`) y nos aseguramos de habilitar los logs `peerstats` (entre otros), que nos servirán para analizar el desfase.

```
# Servidores stratum 1
server hora.roa.es iburst maxpoll 7
server minuto.roa.es iburst maxpoll 7
server hora.cica.es iburst maxpoll 7
server ntps1-0.cs.tu-berlin.de iburst maxpoll 7
server rustime01.rus.uni-stuttgart.de iburst maxpoll 7
server rustime02.rus.uni-stuttgart.de iburst maxpoll 7
server ntp.vsl.nl iburst maxpoll 7
server ntp.fizyka.umk.pl iburst maxpoll 7
server tempus1.gum.gov.pl iburst maxpoll 7
server tempus2.gum.gov.pl iburst maxpoll 7
server time1.stupi.se iburst maxpoll 7
server ntp1.inrim.it iburst maxpoll 7
server ntp1.oma.be iburst maxpoll 7
server time.esa.int iburst maxpoll 7
server time1.google.com iburst maxpoll 7
server time2.google.com iburst maxpoll 7
server time.windows.com iburst maxpoll 7
server time1.apple.com iburst maxpoll 7
server time2.apple.com iburst maxpoll 7
server gbg1.ntp.se iburst maxpoll 7
server mmo1.ntp.se iburst maxpoll 7
server sth1.ntp.se iburst maxpoll 7
server ntp.time.nl iburst maxpoll 7

# Permitir seleccion de hasta 20 servidores
tos maxclock 20
```

```
# Configuracion de logs
enable stats
statsdir "C:\Program Files (x86)\NTP\etc\"
statistics peerstats loopstats
filegen peerstats file peerstats type day enable
```

Tras reiniciar el servicio, veremos que la fuente GPS queda descartada (sin código de estado), tal como hemos configurado. El servicio `ntpd` tiene una convergencia más lenta que `chrony` por lo que es posible que observemos valores de desfase altos o muy variables que pueden tardar horas en estabilizarse. Por lo tanto, al igual que con `chrony`, también es recomendable realizar la medición durante varios días.

En este caso se ha observado que la fuente principal cambia con frecuencia, y a consecuencia de ello, también cambia frecuentemente el conjunto de fuentes seleccionadas, a diferencia del comportamiento de `chrony`.

El análisis del desfase se realiza con un script similar al descrito para `chrony`, adaptándolo al formato de log usado por `ntpd`.

La gráfica de la figura 11 muestra la evolución del desfase durante el intervalo de análisis (más de 8 días) y las estadísticas resultantes. El dato que nos interesa en este paso es valor de **-0.100183 s** para el desfase, que corregiremos en el apartado siguiente.

La gráfica muestra mayor variabilidad en comparación con la obtenida en Linux: nótese la diferencia en la escala vertical de las gráficas, 0.5 ms para la de Linux frente a 5 ms para la de Windows. Aunque puede afectar el hecho de que el *driver* PPS trabaje en espacio de usuario, también hay que considerar que el continuo cambio de selección de fuentes puede haber contribuido a incrementar la desviación estándar observada, que alcanza el valor de 0.84 ms en el caso de *sigma clipping*. En este punto lo importante es que el valor de desfase obtenido sea representativo, y estos datos nos indican que debe serlo con bastante fiabilidad en un margen inferior a ± 1 ms. Una vez configurada la fuente GPS como principal, la selección será estable, y el *jitter* quedará afectado principalmente por las capas de transmisión vía USB y de software (*drivers* PPS y NMEA, y `ntpd`) y la posible carga de CPU del equipo.
Puede resultar sorprendente que los valores de desfase obtenidos sean tan diferentes para Linux y Windows, a pesar de usar el mismo hardware. Hay que tener en cuenta que no son comparables ya que en el primer caso, el reloj de referencia GPS que ve `chrony` se corresponde con el ofrecido por `gpsd`, que se encarga de toda comunicación con el receptor GPS y su sincronización con PPS, mientras que en el segundo caso, es `ntpd` quien se encarga de sincronizar PPS con los mensajes que le llegan desde el receptor GPS a través del driver NMEA.

5.3.3. Corrección del desfase de la fuente GPS PPS

La configuración final se obtiene realizando los siguientes cambios:

- sustituyendo `noselect` por `prefer`; esto de manera efectiva deshabilita el algoritmo de combinación de NTP, de tal forma que los valores de desfase y su error asociado serán los de la fuente GPS, que sistemáticamente serán mejores que los de las fuentes NTP.
- especificando el valor de desfase en segundos, en este caso 0.100183, en el parámetro `time1` (*PPS time offset calibration factor*) [26]. Obsérvese el cambio de signo respecto al valor obtenido en el apartado anterior.

Quedando, por tanto, de la siguiente manera:

```
# GPS PPS via COM12
#   mode 49 = 1 + 48 = process $GPRMC + linespeed 38400 bps
server 127.127.20.12 mode 49 minpoll 4 maxpoll 4 prefer
#   flag1 = PPS, time1 = offset calibration
fudge 127.127.20.12 time1 +0.100183 flag1 1
```

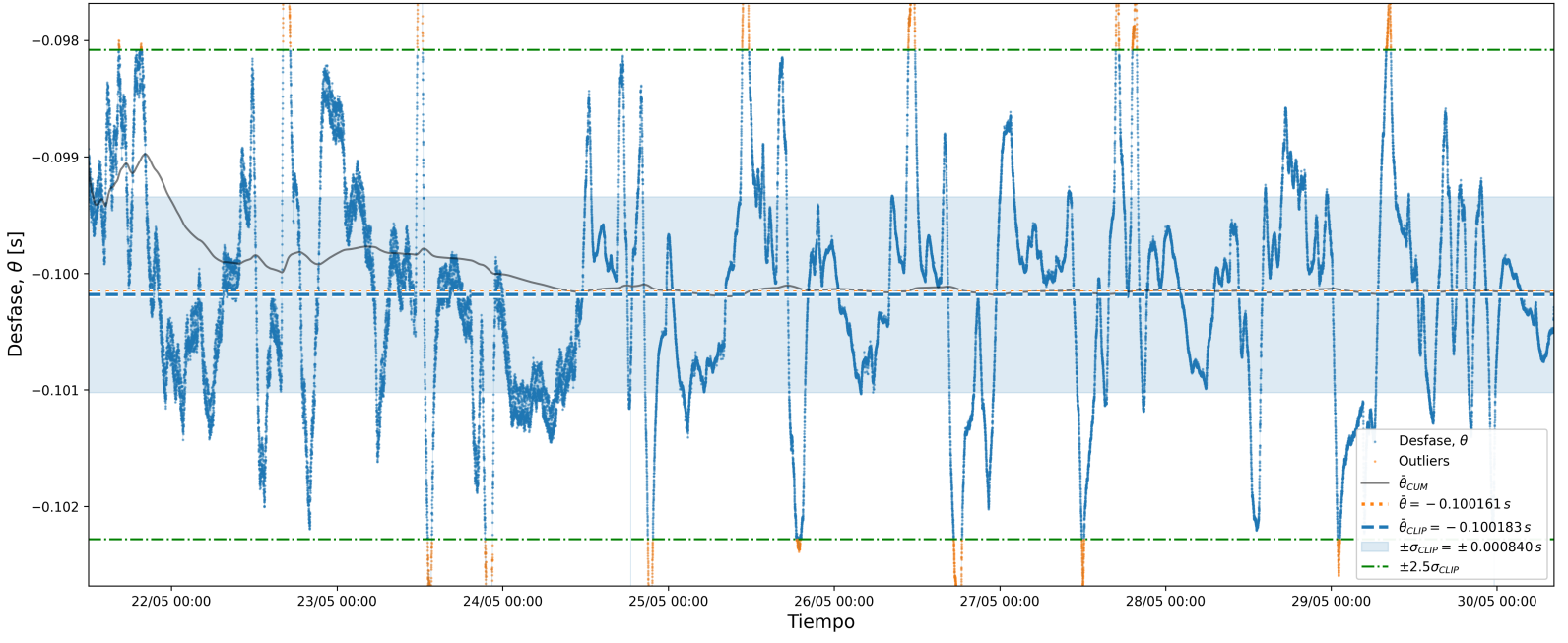

Figura 11. *Evolución del desfase de la fuente GPS durante más de 8 días, medido en base a 21 fuentes NTP stratum 1. Prueba realizada en entorno Windows, con Meinberg NTP.*

Puede ser interesante reducir el valor de `minpoll` y `maxpoll` a 3, de manera que el sondeo al GPS se realice cada $2^3 = 8$ segundos, si bien en las pruebas realizadas no se han obtenido diferencias significativas.

6. Resultados

Presentamos a continuación los resultados obtenidos tras monitorizar el desfase residual y su desviación estándar, tanto en Linux como en Windows, una vez compensado el desfase estimado en apartados anteriores.

Durante el intervalo de análisis se ha realizado una prueba de carga consistente en una captura de una ROI (*Region of Interest*) de 480 x 480 píxeles para conseguir una tasa de 100 fps (*frames per second*), y comprobar si provoca fluctuaciones en el desfase.

Respecto a la configuración definitiva del servicio NTP, `chrony` o `ntpd`, es conveniente indicar que no es necesario mantener una lista tan exhaustiva de servidores NTP. Basta con mantener unos pocos, y no necesariamente stratum 1, simplemente por disponer de fuentes alternativas en caso de que usemos el equipo de captura sin el receptor GPS.

6.1. Resultados en Linux

Las especificaciones del equipo de captura usado son: Intel Celeron J3455 @ 1.50GHz 4 cores (Q3'16), 4GB RAM, SSD SATA.

Los resultados obtenidos se resumen en la figura 12, que conserva la misma escala vertical que la gráfica de medición de desfase, figura 6, para facilitar su comparación.

Se observa que, tras la corrección de desfase, la media del residuo es inferior a la precisión de medida. Su desviación estándar es de tan sólo 4 μs, lo que implica una estabilidad excelente para tratarse de una fuente de tiempo transportada a través de USB.

En términos absolutos, se ha apreciado una desviación de hasta 52 μs, que se produce bajo las condiciones de la prueba de captura y posterior copia del fichero de vídeo a través de red (ethernet). En cualquier caso, este resultado indica una precisión bastante por debajo del milisegundo.

6.2. Resultados en Windows

Las especificaciones del equipo de captura usado son: Intel i7-8750H @ 2.2GHz 6 cores (Q2'18), 32 GB RAM, SSD NVMe.

Los resultados obtenidos se resumen en la figura 13, que conserva la misma escala vertical que la gráfica de medición de desfase, figura 11, para facilitar su comparación. Nótese que dicha escala es diez veces superior a la de los resultados en Linux, 12.

Se observa que, tras la corrección de desfase, la media del residuo es muy pequeña, del orden de microsegundos, o incluso mejor en el caso de sigma-clipping. Su desviación estándar es de 80 μs, lo que supone una estabilidad bastante aceptable.

En términos absolutos, se ha observado una desviación de hasta 371 μs, que no se ha podido trazar hasta ninguna actividad concreta en el sistema. Durante las pruebas de captura no se aprecia ningún impacto obvio en el desfase. Se realizaron dos pruebas seguidas para confirmar este aspecto. En cualquier caso, a pesar de que estos resultados no son tan buenos como los obtenidos con Linux, la precisión alcanzada también está holgadamente por debajo del milisegundo.

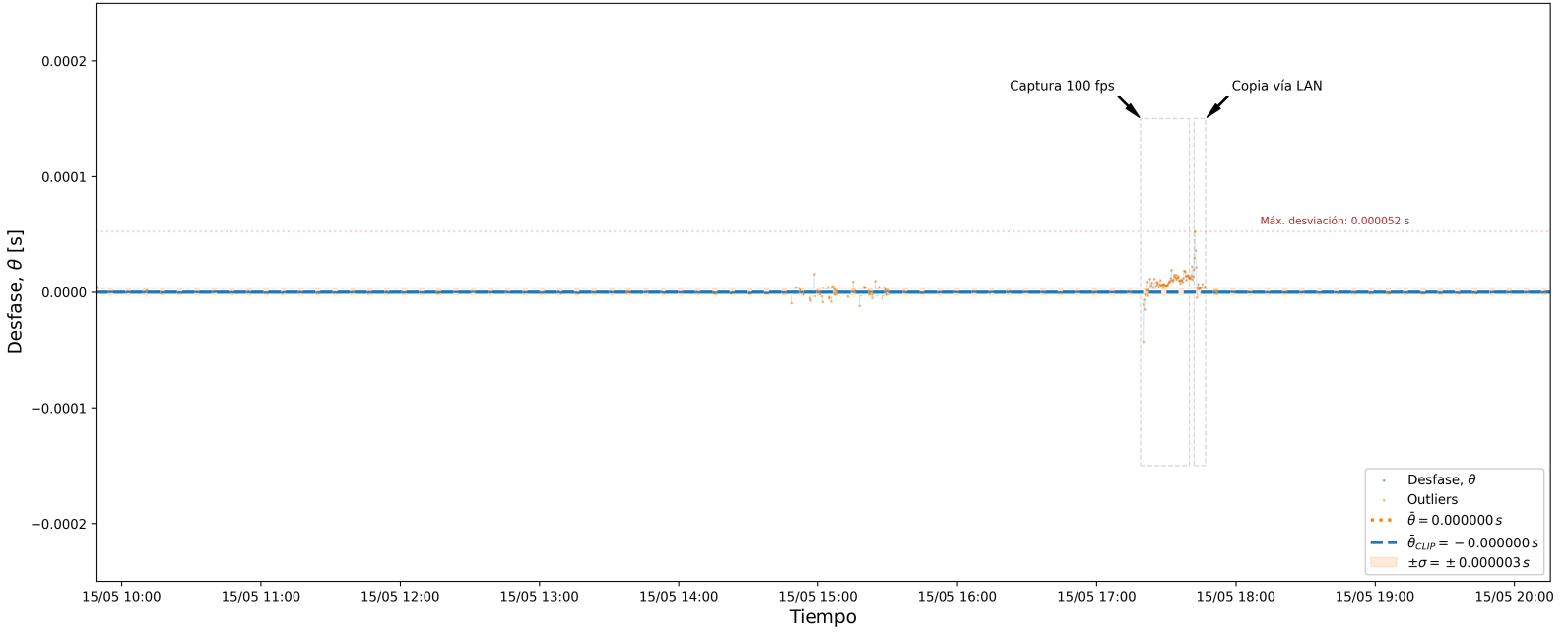

Figura 12. *Resultados obtenidos en la prueba realizada en entorno Linux, con* `gpsd` *y* `chrony`*.*

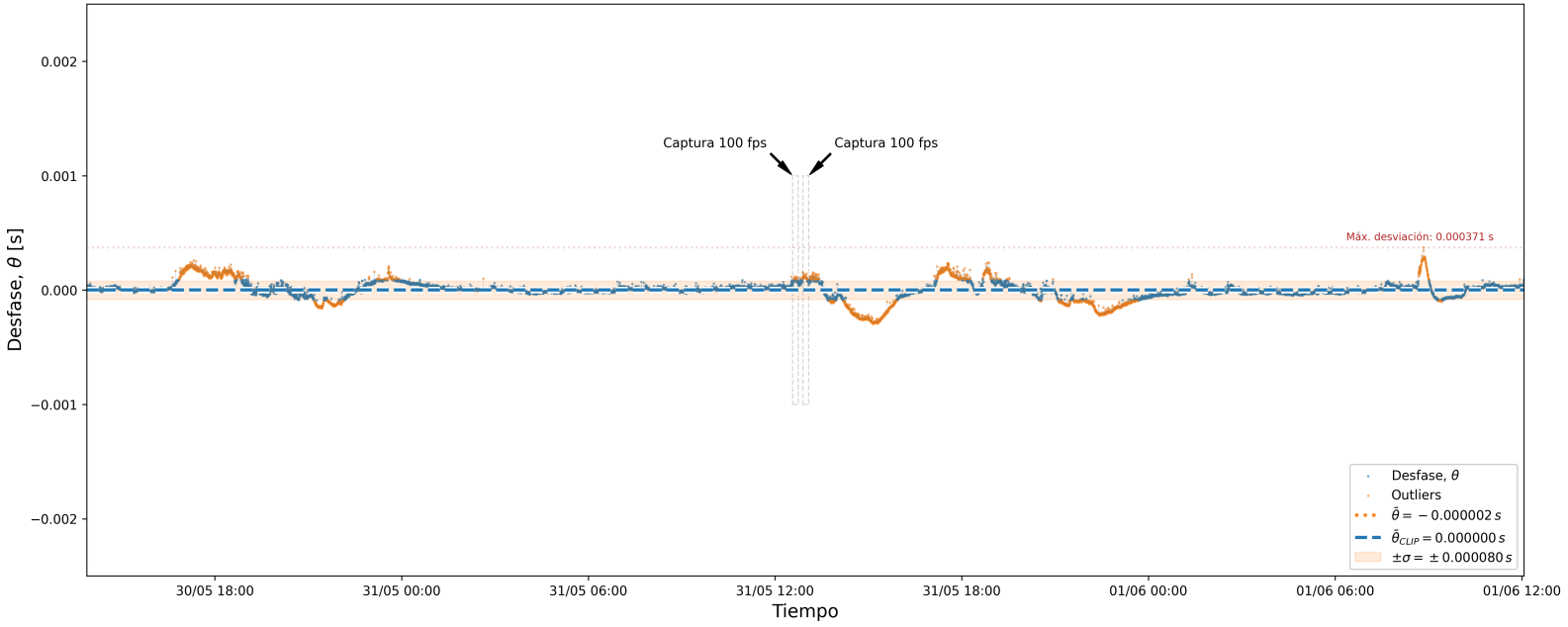

Figura 13. *Resultados obtenidos en la prueba realizada en entorno Windows, con Meinberg NTP.*

7. Trabajo futuro

Dado que nuestro objetivo final es obtener marcas de tiempo con la mejor precisión que podamos alcanzar en nuestra captura de los fenómenos a observar, debemos considerar también el factor que constituye el software de captura de vídeo, entre los que cabe destacar Sharpcap, Firecapture o AstroDMX Capture.

En un próximo número de esta revista se complementará el presente estudio con un análisis de la latencia instrumental involucrada en este proceso. El experimento propuesto consiste en capturar con una cámara CMOS los destellos del LED PPS incorporado en el receptor GPS, lo que proporcionará una vía alternativa, fotométrica, para determinar los instantes que corresponden al inicio de cada segundo UTC. El estudio desarrollará en particular la problemática asociada a las cámaras con *rolling shutter*.

8. Conclusiones

La solución propuesta para obtener una base de tiempos precisa mediante un módulo GPS a través de USB, según las pruebas realizadas, es capaz de ofrecer la exactitud requerida para las observaciones de ocultaciones estelares, al ser la precisión obtenida inferior al milisegundo. Los factores críticos para lograr esta precisión se resumen en:

- Usar de manera efectiva la señal PPS, que en este caso implica asegurarse de transportarla a través de USB y de configurar el software para asignar adecuadamente a qué instante de tiempo corresponden las marcas temporales que envía el módulo GPS;
- Usar un dispositivo USB-serie cuyo *driver* presente mínima latencia ante cambios en las líneas de control, ya que a través de una de ellas, DCD, se envían los pulsos PPS;
- Calibrar correctamente el desfase asociado a la referencia GPS en el servicio NTP, para lo cual hay que medir su evolución en base a fuentes NTP fiables, y hacer posteriormente un análisis de los datos obtenidos.

En el presente artículo se tratan detalladamente todos los pasos para la implementación de esta solución, incluyendo estos aspectos.

Debe considerarse, no obstante, que el montaje del hardware requiere trabajar con herramientas de electrónica para hacer soldadura. Por otra parte, la medición del desfase puede requerir varios días de captura de datos para obtener valores significativos.

Una cuestión no analizada en este estudio es el impacto que podría tener la activación del modo de ahorro de energía del ordenador portátil sobre la precisión de la base de tiempos.

Referencias

[1] Repositorio de código en GitHub. https://github.com/JCAAC-FAAE

[2] Beisker et al. Contributions of 'citizen science' to occultation astronomy. https://www.researchgate.net/publication/389394183_Contributions_of_'citizen_science'_to_occultation_astronomy. Philosophical Transactions A, The Royal Society. Febrero, 2025.

[3] George Viscome. Occultation Observing and Recording Primer (v5.1.3). https://occultations.org/documents/OccultationObservingPrimer.pdf. International Occultation Timing Association (IOTA). 2024.

[4] NASA Science Directorate. Uranus Stellar Occultation Campaign 2025. https://science.larc.nasa.gov/uranus2025/. Consultada en octubre de 2025.

[5] David D. Turnell. NASA Lucy Mission. Chasing an Asteroid's Shadow. https://lucy.swri.edu/2021/03/17/Occultations.html. Marzo, 2021.

[6] Wikipedia. Network Time Protocol. https://en.wikipedia.org/wiki/Network_Time_Protocol. Consultada en octubre de 2025.
[7] Kevin Sookocheff. How Does NTP Work? https://sookocheff.com/post/time/how-does-ntp-work/. Noviembre, 2021.
[8] Network Time Foundation. How NTP Works. https://www.ntp.org/documentation/4.2.8-series/warp/. Noviembre, 2022.
[9] Comparison of NTP implementations. Chrony Documentation. https://chrony-project.org/comparison.html. Consultada en octubre de 2025.
[10] Wikipedia. Sistema global de navegación por satélite. https://en.wikipedia.org/wiki/Satellite_navigation. Consultada en octubre de 2025.
[11] Wikipedia. SBAS. https://es.wikipedia.org/wiki/SBAS
[12] Austin Pivarnik. Microsecond accurate NTP with a Raspberry Pi and PPS GPS. https://austinsnerdythings.com/2021/04/19/microsecond-accurate-ntp-with-a-raspberry-pi-and-pps-gps/. Abril 2021.
[13] Austin Pivarnik. Revisiting Microsecond Accurate NTP for Raspberry Pi with GPS PPS in 2025. https://austinsnerdythings.com/2025/02/14/revisiting-microsecond-accurate-ntp-for-raspberry-pi-with-gps-pps-in-2025/. Febrero 2025.
[14] James Clark. PTP Client Hardware. https://satpulse.net/hardware/clients.html. Consultada en octubre de 2025.
[15] James Clark. Windows PTP Client. https://satpulse.net/howtos/ptp-windows.html. Consultada en octubre de 2025.
[16] Módulo Beitian BE-280 basado en u-blox M10050. https://store.beitian.com/products/beitian-gps-module-with-antenna-ubx-m10050-gnss-chip-ultra-low-power-gnss-receiver-for-track-be-180
[17] FTDI. AN232B-04 Data Throughput, Latency and Handshaking. https://ftdichip.com/wp-content/uploads/2020/08/AN232B-04_DataLatencyFlow.pdf. 2006.
[18] Intel. USB 3.0 Radio Frequency Interference Impact on 2.4 GHz Wireless Devices. https://www.usb.org/sites/default/files/327216.pdf. Abril 2012.
[19] Gary E. Miller, Eric S. Raymond. GPSD Time Service HOWTO. https://gpsd.gitlab.io/gpsd/gpsd-time-service-howto.html. Consultada en octubre de 2025.
[20] chronyc(1) Manual Page. https://chrony-project.org/doc/4.0/chronyc.html. Consultada en octubre de 2025.
[21] chrony.conf(5) Manual Page. https://chrony-project.org/doc/4.8/chrony.conf.html. Consultada en octubre de 2025.
[22] Meinberg NTP Download. https://www.meinbergglobal.com/english/sw/ntp.htm. Consultada en octubre de 2025.
[23] Meinberg NTP Time Server Monitor. https://www.meinbergglobal.com/english/sw/ntp-server-monitor.htm. Consultada en octubre de 2025.
[24] Meinberg Knowledge Base. NTP for Windows: Using PPS Signals on Windows. https://kb.meinbergglobal.com/kb/time_sync/ntp/ntp_for_windows/using_pps_signals_on_windows. Consultada en octubre de 2025.
[25] u-blox u-center 2. https://www.u-blox.com/en/u-center-2. Consultada en octubre de 2025.
[26] Meinberg Knowledge Base. NTP for Windows: Generic NMEA GPS Receiver. https://www.meinbergglobal.com/download/ntp/docs/html/drivers/driver20.html. Consultada en octubre de 2025.

JCAAC (2025), **4**, 33–43

JCAAC

SECTION: HERRAMIENTAS DEL OBSERVATORIO VIRTUAL

Caracterización de cúmulos abiertos - VOSA

Joaquín Álvaro Contreras[1]

[1]FAAE, Madrid, Spain. E-mail: jalvaro@citelan.es.

Keywords: cúmulos estelares, herramientas VO, TOPCAT, Aladin, observatorio virtual, Gaia, VOSA

© Este artículo está protegido bajo una licencia Creative Commons Attribution 4.0 License

Resumen

En artículos anteriores de la serie dedicada a herramientas del Observatorio Virtual (VO) aplicadas a la caracterización de cúmulos estelares abiertos (OCs) hemos utilizado aplicaciones como Clusterix 2.0 [1] y TOPCAT [2]. Con estas herramientas hemos conseguido una buena determinación de los componentes de un OC, y con ello una estimación razonable del número de objetos que lo integran, así como su distancia media y dinámica en movientos propios. Pero es deseable obtener otra información importante como la edad del cúmulo y una valoración de la masa total al menos. Para este propósito utilizaremos otra herramienta del Observatorio Virtual, en este caso VOSA [3].

Abstract

In previous articles of this series on Virtual Observatory (VO) tools applied to the characterization of open clusters (OCs), we made use of applications such as Clusterix 2.0 [1] and TOPCAT [2]. These tools allowed us to reliably identify OC members and, consequently, to obtain reasonable estimates of the number of objects they contain, their mean distance, and their proper-motion kinematics. Nevertheless, it is also desirable to derive other key parameters, such as the cluster's mean age and at least an estimate of its total mass. To this end, we turn to another VO tool, namely VOSA [3].

1. Introducción

El trabajo desarrollado hasta aquí, tomando como ejemplo práctico el cúmulo estelar NGC 2682 (M67), nos ha permitido una estimación del número de objetos del cúmulo identificados individualmente, así como sus respectivas propiedades astrométricas y también una referencia fotométrica ('Gmag' de GAIA/EDR3). Ahora haremos uso de VOSA [3] para ampliar la información relativa a cada uno de los componentes de NGC 2682 y tener así una caracterización más completa del cúmulo.

VOSA (Virtual Observatory SED Analyzer) es una aplicación web desarrollada por el Centro de Astrobiología (CAB, CSIC-INTA) y el Spanish Virtual Observatory (SVO) [4] diseñada para ayudar a construir, analizar y ajustar *Distribuciones Espectrales de Energía* (SEDs, por sus siglas en inglés) de objetos astronómicos usando datos tanto propios como procedentes de servicios del Observatorio Virtual (VO).

A partir del nombre de las estrellas a estudiar y/o de sus coordenadas, VOSA puede consultar múltiples catálogos fotométricos en el enterno VO (Gaia, 2MASS, WISE, Pan-STARRS, SDSS, ...), lo que permite obtener la distribución espectral de energía (SED) de estos objetos. A partir de ahí se hace posible la comparación y ajuste de cada estrella con los diferentes modelos teóricos (BT-Setti, Kurucz, NextGen, PHOENIX, etc.). Integrando la SED de mejor ajuste con los modelos, VOSA calcula luminosidades e isócronas evolutivas para determinar masas y edades.

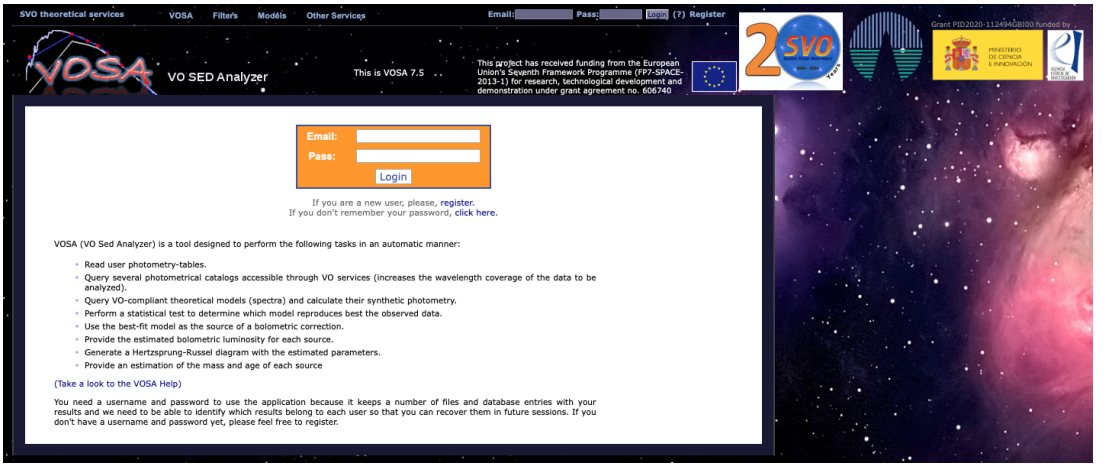

Figura 1. *Página de acceso a la aplicación - http://svo2.cab.inta-csic.es/svo/theory/vosa/index.php.*

2. Trabajando con VOSA

El acceso a VOSA es gratuito, pero requiere crear una cuenta de usuario y utilizar las claves asignadas para hacer uso de la herramienta. Esto tiene una ventaja importante: los datos y tareas realizadas quedan almacenados en la cuenta correspondiente mientras el propio usuario no decida borrarlos y esto permite volver a dichas tareas y/o modificarlas tantas veces como sea necesario.

Asumiendo que esto se ha hecho y que hemos accedido a la aplicación debidamente identificados, empezaremos por importar a VOSA nuestro *set* de estrellas.

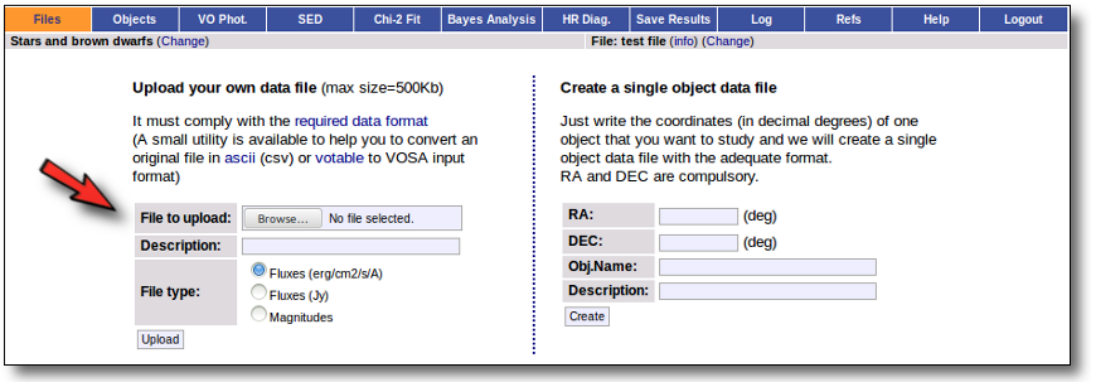

Figura 2. *Página inicial - importar datos propios / obtener datos de un objeto a partir del VO.*

Utilizaremos el conjunto que se dio por válido al final del trabajo anterior con Clusterix/TOPCAT. Pero aquí hay que tener en cuenta un par de consideraciones:

1. el formato de los datos/fichero a importar.
2. el tamaño del fichero y/o el número de registros máximo admitido.

En cuanto al formato hay información suficiente en el *help* de VOSA http://svo2.cab.inta-csic.es/svo/theory/vosa/helpw4.php?otype=star&action=help&what=format, pero podemos simplificar diciendo que basta un fichero .CSV con los campos separados por un espacio (" ") y los valores numéricos

con un punto (.) como separador decimal. Los campos no informados se dejarán con tres guiones (- - -).

En lo relativo al tamaño del fichero, como se indica en la misma utilidad de '*upload*', está restringido a 50 MB, (esto da para mucho ...), pero hay también un valor máximo en cuanto al número de filas que está limitado a 1 000. Si fuera necesario trabajar con un número mayor de objetos, se puede hacer, pero hay que contactar con el administrador del sistema para que lo autorice.

Si echamos mano de la tabla que dimos por buena en el artículo anterior (**M67v2**), vemos que ésta contiene 1 194 estrellas. Con objeto de evitar el límite de 1 000 filas, que nos obligaría a solicitar la excepción a esta restricción, y dado que se trata de un ejemplo/práctica con VOSA, haremos una nueva selección de objetos modificando uno de los filtros empleados en el trabajo anterior de manera que el tamaño de la muestra sea ligeramente inferior. En concreto, aplicaremos el filtro:

$$\text{Expression} : M67v2 \;\&\&\; ePLL/PLL <= 0.06. \tag{2.1}$$

De esta manera reducimos el error relativo en el parámetro '*parallax*' del 10%, empleado entonces, al 6%, con lo que nuestra muestra pasará a ser de 972 estrellas. Podemos emplear de nuevo TOPCAT para hacer esto.

Finalmente, el fichero a importar a VOSA tendrá la forma:

```
STAR_NO RAJ2000 DECJ2000 --- ---
9000 132.77497 9.273203 --- ---
11551 132.823916 9.675918 --- ---
86713 132.999679 12.083028 --- ---
103402 132.074298 12.369349 --- ---
88986 132.622547 12.317817 --- ---
40562 132.428658 11.719099 --- ---
85727 132.742274 11.872869 --- ---
86689 133.136241 12.106948 --- ---
87333 132.48592 11.948121 --- ---
130797 132.043857 14.48867 --- ---
88940 132.741523 12.308915 --- ---
```

Figura 3. *Primeras filas del fichero - formato texto asci (.CSV), con 'espacio' como delimitador y '.' como separador decimal.*

Como puede comprobarse sólo hemos incluido en este fichero el nombre (o número) de la estrella y las coordenadas (ascensión recta y declinación) con objeto de que VOSA pueda utilizar estas '*coordenadas de usuario*' si no hubiera concordancia del *nombre* con el de los catálogos consultados. El resto de campos se ignoran, ('- - -' '- - -'), y no es necesario añadir más que un par de estos campos *vacíos*.

Hecho esto, seleccionamos el archivo y lo subimos a VOSA. Quedará en nuestro repositorio y, antes de empezar a trabajar con él, lo seleccionamos y tendremos varias opciones disponibles: '*Save*', '*Show Objects*', ..., '*Delete*'. Cada vez que se entre en VOSA este fichero formará parte de nuestro particular repositorio. Habrá que seleccionarlo y se mostrará una breve información del mismo, así como las acciones ya realizadas sobre él en VOSA en sesiones previas.

2.1. Servicios de VOSA

El hilo de tareas posibles, una vez cargado el fichero sobre el que se quiere trabajar, es sencillo y está bien ordenado siguiendo la secuencia marcada por el menú superior de la aplicación:

2.1.1. Objects

La primera de las acciones a realizar es definir la astrometría de las estrellas objeto de estudio. En concreto aquí es necesario conocer las coordenadas ecuatoriales y la distancia de cada objeto. Estos son

Figura 4. *Menú principal de la aplicación.*

datos que ya teníamos después del trabajo hecho con TOPCAT previamente y que podríamos haber incluido en el fichero importado, pero que también VOSA puede resolver.

Si pedimos a VOSA que resuelva las coordenadas de las estrellas, utilizará el servicio '*Sesame*' consultando diversas bases de datos (*Simbad, NED y VizieR*) y recuperarlas de ahí a partir del *nombre* de cada objeto. En nuestro ejemplo no hay concordancia de este nombre con los contemplados en los catálogos referidos, por lo que ya hemos tenido la precaución de añadir estas coordenadas en el fichero importado y VOSA las utilizará como *datos de usuario* válidos para la identificación de cada objeto en las sucesivas tareas.

	Final		User Data			Sesame	
Object	RA (deg)	DEC (deg)	RA (deg)	DEC (deg)		RA (deg)	DEC (deg)
102283	134.428656	13.755897	134.428656	13.755897	☑	---	---
102802	132.111048	12.107285	132.111048	12.107285	☑	---	---
102804	132.122303	12.109806	132.122303	12.109806	☑	---	---
102869	132.232864	12.161909	132.232864	12.161909	☑	---	---
102959	132.087474	12.235831	132.087474	12.235831	☑	---	---

Figura 5. *Coordenadas* 'User Data' *utilizadas por VOSA en adelante. El servicio* 'Sesame' *no ha identificado los objetos por el* nombre *en los catálogos consultados.*

También '*las distancias*' (en *parsec*) podríamos haberlas incluido en el conjunto de datos importado a VOSA, pero dejaremos que sea VOSA quien resuelva ya este dato.

El botón '*Distances*' nos lleva a la utilidad correspondiente (ver Figura 6), donde podemos asignar un margen de tolerancia en torno a las posiciones para la identificación de los objetos. Por defecto esta tolerancia es de ±5 *arcsec*, pero puede ajustarse de otro modo si fuera necesario. Al solicitar la búsqueda se inicia el proceso.

Algo a tener en cuenta es que algunas de las tareas en VOSA pueden tardar un cierto tiempo, pero todas son tareas *asíncronas* por lo que podemos abandonar la aplicación si fuera necesario y tendremos los resultados completados al volver a ella.

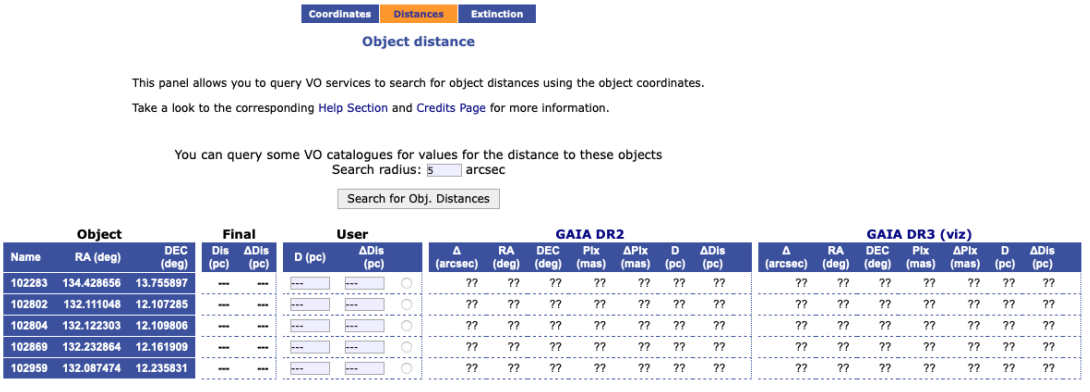

Figura 6. *Determinación de distancias.*

Una vez resuelto el proceso, VOSA nos presenta una nueva pantalla, (Figura 7), en la que vamos a realizar tres acciones:

1. seleccionar qué valores (distancias) se quieren utilizar -en este caso '*GAIA DR3 (viz)*'-.
2. proceder a los cambios.
3. salvar los datos.

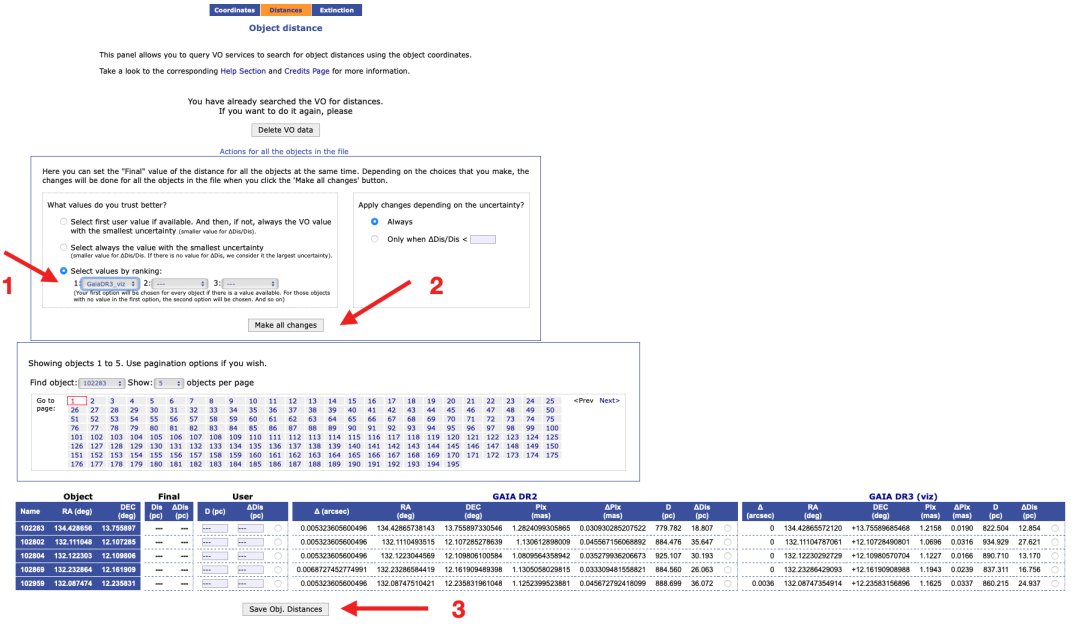

Figura 7. *Determinación de distancias - Proceso concluido - Guardar datos.*

2.1.2. Fotometría y Distribución Espectral de Energía

Ahora ya es el momento de recuperar datos fotométricos de nuestras estrellas con los que componer después sus respectivas distribuciones espectrales. Para ello vamos a la pestaña '*Build SEDs*'. Aquí aparecen marcados por defecto todos los catálogos disponibles, pero sólo vamos a hacer uso de algunos de ellos con objeto de no hacer de éste un proceso excesivamente pesado.

Desmarcamos todos y procedemos después a seleccionar sólo algunos: **2MASS, WISE,** en el infrarojo y **APASS 9, Gaia DR3** en el óptico. Y se ejecuta la consulta.

Con esto ya tenemos caracterizada cada estrella también en sus valores fotométricos, en unidades de *flujo*, (figura 8). Pueden analizarse una a una con objeto de estimar si presentan *exceso* en el infrarojo o en el ultravioleta/azul, pero dado que nuestro propósito no se centra en objetos particulares sino en el conjunto global del cúmulo podemos ignorar estos detalles.

2.1.3. Ajuste con modelos teóricos estelares

El siguiente paso es proceder al mejor ajuste '*Chi-square*' con los modelos teóricos estelares para cada objeto. También en este caso seleccionaremos sólo alguno de los modelos disponibles. En concreto '*Kurucz ODFNEW/NOVER, alpha: 0.0*' y acotamos, en la siguiente pantalla, valores para los parámetros:

1. teff: 3 500 - 20 000 k
2. logg: 4.0 - 5.0 dex

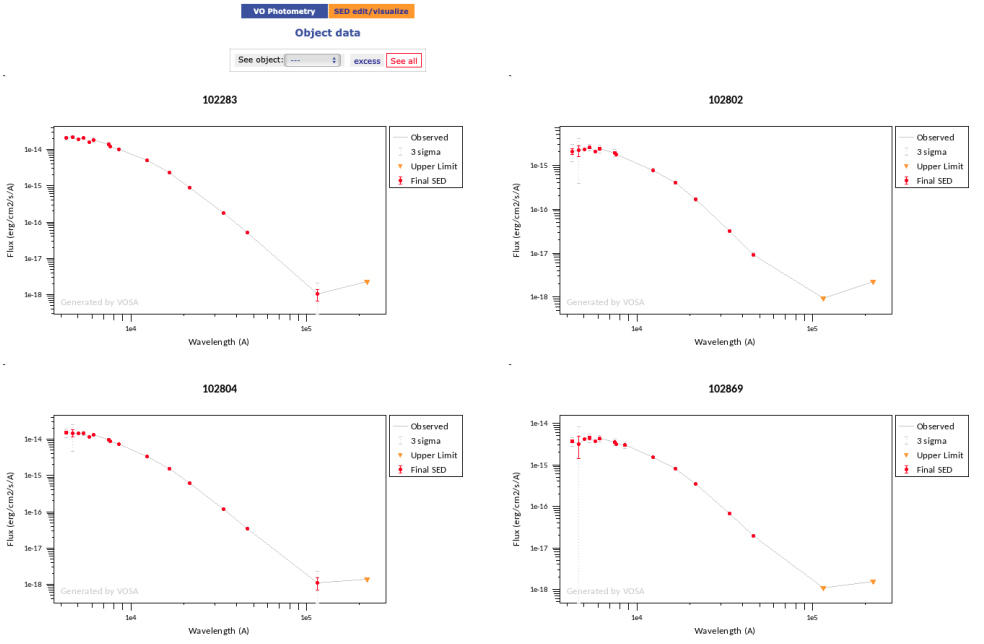

Figura 8. *Distribución espectral de los objetos estudiados.*

3. M/H = -0.5 - 0.5.

El resultado de esta última etapa es muy similar al presentado en la figura 8, pero aquí (figura 9) ya aparecen caracterizadas las estrellas según el ajuste con el modelo empleado.

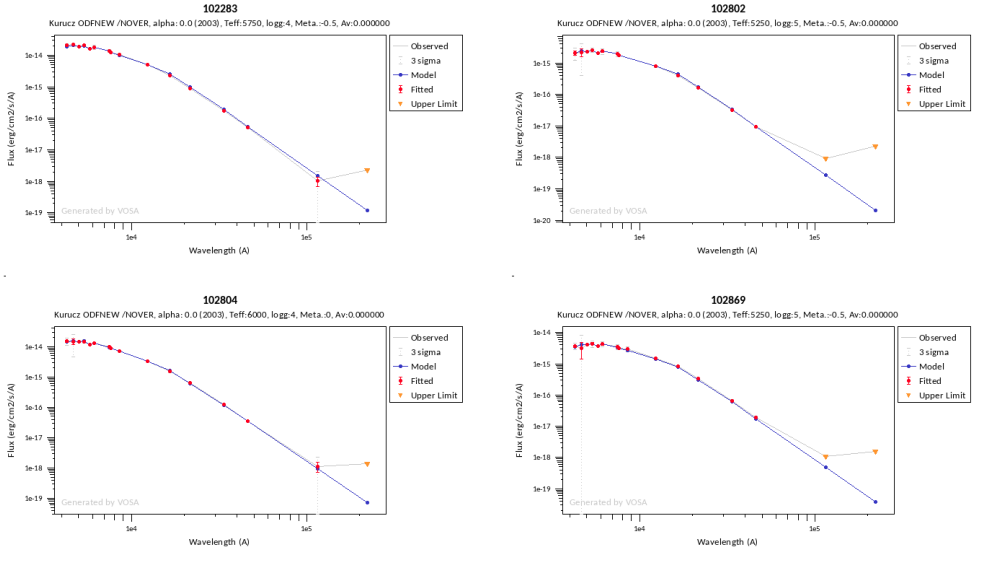

Figura 9. *Caracterización estelar por ajuste con los modelos teóricos* 'Kurucz ODFNEW/NOVER'.

2.1.4. El diagrama HR

VOSA ya está en condiciones de construir el diagrama H-R de los objetos que componen el cúmulo estimando *masas* para los objetos y definiendo isócronas que nos darán también una edad estimada. Con el modelo *Kurucz* se utilizan isócronas *Siess*. El resultado puede verse en la figura 10.

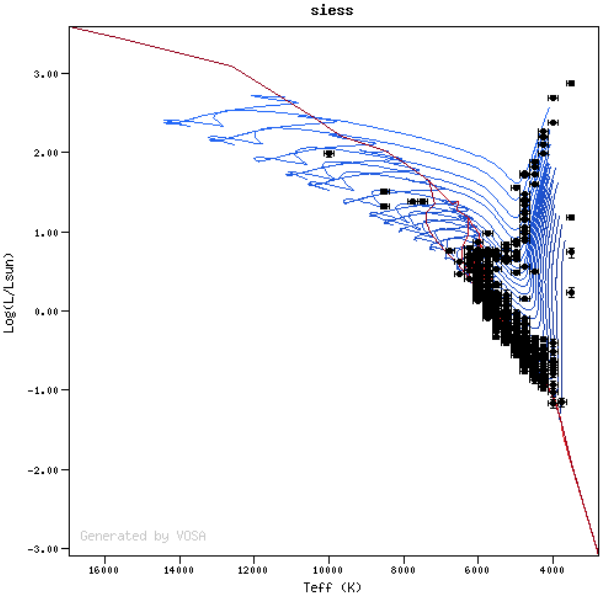

Figura 10. *Diagrama H-R con isócronas* Siess *del modelo* 'Kurucz ODFNEW/NOVER'.

VOSA puede exportar datos al ecosistema del Observatio Virtual. Si hemos abierto la aplicación TOPCAT, (o Aladin, por ejemplo), la tabla (figura 11) podrá ser enviada a estas aplicaciones.

Objects

Object	Model	T_{eff}		LogL		Age			Mass		
78216	siess	4750	(4625,4875)	-0.6458	(-0.6756,-0.6180)	---	---	[4]	---	---	[4]
78308	siess	5750	(5625,5875)	0.1785	(0.1623,0.1941)	5.0000	..(---,5.1367)		1.0989	(---,1.2000)	[2]
78703	siess	5000	(4875,5125)	-0.3644	(-0.3996,-0.3318)	---	---	[4]	0.8771	(---,0.9004)	
78773	siess	6000	(5875,6125)	0.2335	(0.2169,0.2495)	---	---	[4]	1.2012	(---,1.2012)	[2]
78779	siess	4500	(4375,4625)	-0.5919	(-0.6265,-0.5599)	---	---	[4]	0.8003	(---,0.8568)	[1]
78817	siess	3500	(3500,3625)	2.8695	(2.8422,2.8952)	---	---	[4]	---	---	[4]
78823	siess	6000	(5875,6125)	0.4998	(0.4831,0.5159)	3.7055	..(3.0002,4.4799)		1.3026	(1.2925,1.3217)	
78836	siess	4500	(4375,4625)	-0.7286	(-0.7820,-0.6810)	---	---	[4]	---	---	[4]
78878	siess	5750	(5625,5875)	0.1046	(0.0838,0.1244)	2.0253	..(---,5.9878)		1.1001	(---,1.1029)	[2]
78881	siess	5750	(5625,5875)	0.0652	(0.0448,0.0846)	---	---	[4]	1.1044	(---,1.1044)	[2]

Figura 11. *Tabla final con datos como temperatura efectiva, luminosidad, edad y masa.*

2.1.5. Resultados

Finalmente, la pestaña '*Results*' nos permite guardar todas las tablas (o selección) y gráficos generados durante el trabajo/proyecto realizado, incluido el '*Activity Log*' donde queda constancia de las acciones realizadas y parámetros aplicados en las mismas. También presenta una reseña completa de las referencias y agradecimientos a considerar en publicaciones relativas al trabajo efectuado.

El proyecto/fichero sigue disponible en la cuenta de usuario, hasta que éste lo borre, pudiendo volver al mismo tantas veces como sea necesario.

3. De nuevo en TOPCAT

Puede ser útil recurrir de nuevo a TOPCAT para hacer uso de algunas de sus facilidades con los datos obtenidos de VOSA. Para esto, como ya se ha comentado, es necesario abrir la aplicación de TOPCAT. Hecho esto, en VOSA aparecerá la opción '*Send results table to SAMP Hub*'. Con la tabla ya en TOPCAT podemos hacer un resumen adicional:

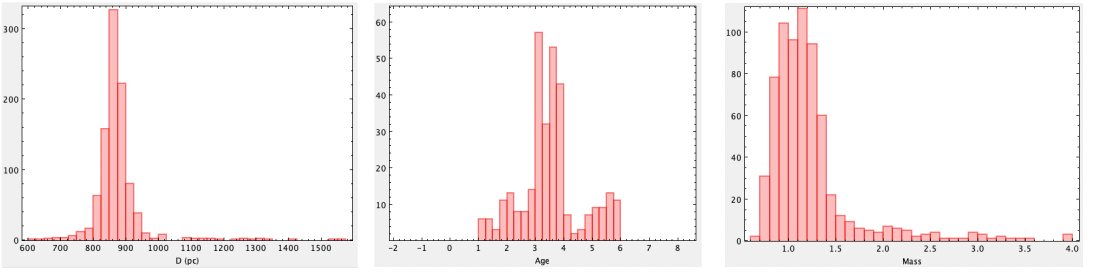

Figura 12. *Distribuciones de distancias (en parsec), edades (en miles de millones de años) y masas (en masas solares).*

La estimación de *distancia* a M67, realizada por VOSA a partir del *match* por coordenadas, es similar a la establecida en el artículo anterior: 873.09 *pc* ± 70.69 *pc*. Respecto a la '*edad*', el ajuste según el modelo *Kurucz/siess* sólo ha determinado un valor posible para 315 objetos con un valor medio de 3.53 *Gyears* ± 1.08 *Gyears*. Resulta evidente una dispersión importante. Lo mismo ocurre con la estimación de masas, donde las estrellas valoradas han sido 680 del total (972): masa media = 1.22 m_{sun} ± 0.47 m_{sun}.

4. Conclusiones

El propósito de estos trabajos ha sido fundamentalmente introducirnos en el uso de algunas herramientas del Observatorio Virtual de la mano de un caso práctico pero sin entrar en detalles de mayor rigor ... Si se persiguen objetivos de más calado, sería conveniente depurar datos espúrios y explorar mejores ajustes con los modelos aplicados.

Por ejemplo, si se añade el modelo BT-Setti en el análisis SED, (más adecuado para estrellas más frías), del conjunto de 972 objetos analizados para NGC 2682, 253 presentan un mejor ajuste con el modelo *Kurucz/Siess* y 719 lo hacen con el modelo *Bt-Setti*. Figuras 13 y 14.
Se propone aquí repetir las tareas realizadas en estos tres artículos de la serie con otro cúmulo estelar. Por ejemplo con *Melotte 22*, también conocido como M45 o *Pleiades*, figura 14. También en este caso se ha ensayado el ajuste SED con los modelos *Kurucz/siess* y *BT-Setti* con isócronas *BHAC15*. En este caso, de 658 estrellas, el modelo *Kurucz* se aplica a 142 objetos y 513 se ajustan mejor con BT-Setti.

M45 es un cúmulo '*jóven*'. Las isócronas *BHAC15* dan una edad media de 250 millones de años, pero con una dispersión/incertidumbre elevada. Mientras que el modelo *Kurucz* las sitúa en 925 millones de años, pero también con unos valores de alta dispersión.

Choose the isochrone/track collections to use for the HR diagram

719 objects (?) have been fitted using the **BT-Settl (CIFIST)** model.

Available ranges of values for this fit model:
- Teff: 2200 - 6400 (K)
- Lbol: 0.0689481 - 491.159 (Lsun)

Use: BHAC15 isochrones and tracks for the HR diagram.

If you want to use other isochrones/tracks for some values of Teff or Lbol, click to add more options

---

253 objects (?) have been fitted using the **Kurucz ODFNEW /NOVER, alpha: 0.0 (2003)** model.

Available ranges of values for this fit model:
- Teff: 3500 - 11000 (K)
- Lbol: 0.0699182 - 740.453 (Lsun)

Use: Siess isochrones and tracks for the HR diagram.

If you want to use other isochrones/tracks for some values of Teff or Lbol, click to add more options

Figura 13. *Ajuste y parámetros con los modelos BT-Setti y Kurucz..*

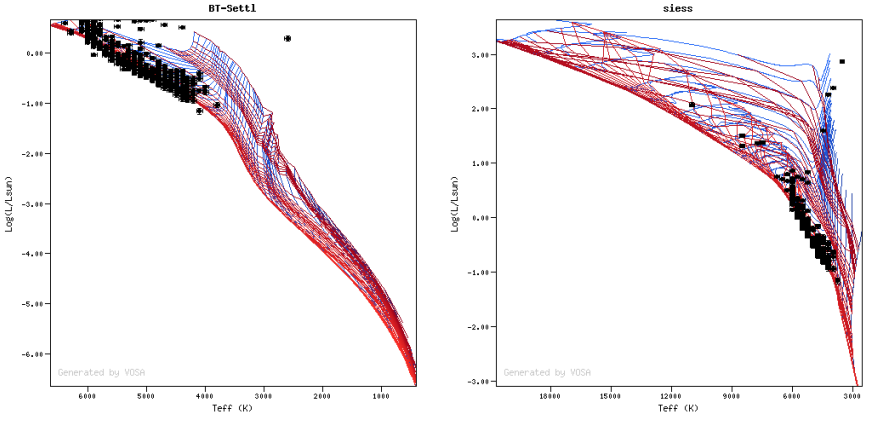

Figura 14. *NGC 2682 (M67). Véase que las escalas en los ejes de ambas gráficas no son iguales. El modelo BT-Setti con isócronas Bt-Settl da mayor resolución para estrellas menos luminosas y más frías..*

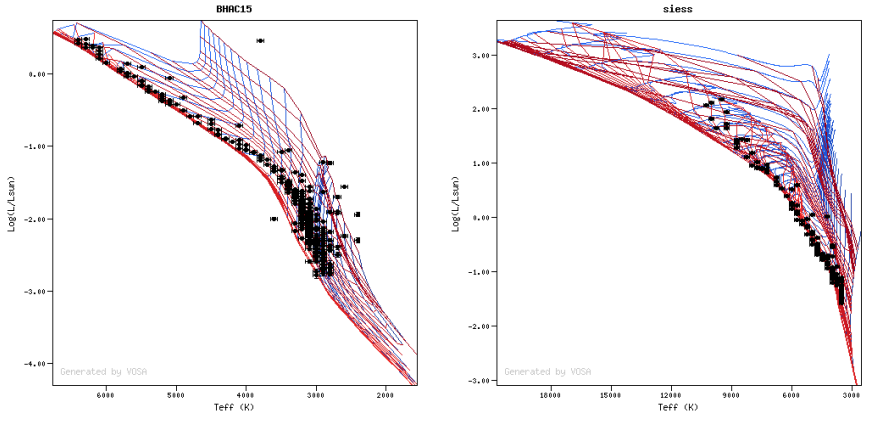

Figura 15. *Melotte 22 (M45). También en este caso el modelo BT-Setti con isócronas BHAC15 ofrece una mayor resolución para las estrellas menos calientes..*

Agradecimientos

VOSA. This publication makes use of VOSA, developed under the Spanish Virtual Observatory https://svo.cab.inta-csic.es project funded by MCIN/AEI/10.13039/501100011033/ through grant PID2020-112949GB-I00. VOSA has been partially updated by using funding from the European Union's Horizon 2020 Research and Innovation Programme, under Grant Agreement nº 776403 (EXOPLANETS-A)
http://svo2.cab.inta-csic.es/svo/theory/vosa/index.php

TOPCAT. This research has made use of "TOPCAT VO"
Author: Mark Taylor, Astrophysics Group, Physics Department, University of Bristol
Email: m.b.taylor@bristol.ac.uk
TOPCAT WWW page: http://www.starlink.ac.uk/topcat/
https://ui.adsabs.harvard.edu/abs/2005ASPC..347...29T/abstract

Spanish Virtual Observatory. This research has made use of the Spanish Virtual Observatory https://svo.cab.inta-csic.es project funded by MCIN/AEI/10.13039/501100011033/ through grant PID2020-112949GB-I00.

Otros

Dereddening. For dereddening the SEDs we make use of the extinction law by Fitzpatrick (1999) improved by Indebetouw et al (2005) in the infrared.
Fitzpatrick, E.; 1999, PASP, 111, 63
Indebetouw et al, 2005, ApJ 619, 931
VO photometry
2MASS All-Sky Point Source Catalog
Skrutskie et al, 2006, AJ, 131, 1163S
Ochsenbein et al 2000, A&AS 143, 221
Acknowledgement :
This publication makes use of data products from the Two Micron All Sky Survey, which is a joint project of the University of Massachusetts and the Infrared Processing and Analysis Center/California Institute of Technology, funded by the National Aeronautics and Space Administration and the National Science Foundation.
WISE
Acknowledgement :
This publication makes use of data products from the Wide-field Infrared Survey Explorer, which is a joint project of the University of California, Los Angeles, and the Jet Propulsion Laboratory/California Institute of Technology, funded by the National Aeronautics and Space Administration.
Ochsenbein et al 2000, A&AS 143, 221
Wright et al 2010, AJ 140, 1868W
APASS 9
Evans et al. 2002, A&A 395, 347E
Ochsenbein et al 2000, A&AS 143, 221
Gaia DR3 (CDS)
Gaia DR3 Documentation
Gaia DR3 Credit and citation instructions
Ochsenbein et al 2000, A&AS 143, 221
Gaia Collaboration et al. 2016, A&A 595, 1G
Acknowledgement :
This work has made use of data from the European Space Agency (ESA) mission Gaia (https://www.cosmos.esa.int/gaia), processed by the Gaia Data Processing and Analysis Consortium (DPAC, https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been

provided by national institutions, in particular the institutions participating in the Gaia Multilateral Agreement.
VO Distance
GAIA DR3 (vizier)
Gaia DR3 Documentation
Gaia DR3 Credit and citation instructions
Ochsenbein et al 2000, A&AS 143, 221
Gaia Collaboration et al. 2016, A&A 595, 1G
Acknowledgement :
This work has made use of data from the European Space Agency (ESA) mission Gaia (https://www.cosmos.esa.int/gaia), processed by the Gaia Data Processing and Analysis Consortium (DPAC, https://www.cosmos.esa.int/web/gaia/dpac/consortium). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the Gaia Multilateral Agreement.
Model Fit: theoretical spectra
Kurucz ODFNEW /NOVER, alpha: 0.0 (2003)
Castelli and Kurucz 2003, IAUS 210, A20
Castelli and Kurucz Atlas
Castelli ATLAS9 grids web page.
HR diagram: Isochrones and Evolutionary Tracks
Siess Isochrones and Evolutionary Tracks
Siess et al 2000, A&A, 358, 593S

References

[1] Clusterix 2.0 - http://clusterix.cab.inta-csic.es/clusterix/
[2] TOPCAT: Tool for OPerations on Catalogues And Tables - https://www.star.bris.ac.uk/~mbt/topcat/
[3] VO SED Analyzer - VOSA - http://svo2.cab.inta-csic.es/theory/vosa/index.php
Bayo, A., Rodrigo, C., Barrado y Navascués, D., Solano, E., Gutiérrez, R., Morales-Calderón, M., Allard, F. 2008, A&A 492,277B.
[4] Spanish Virtual Observatory https://svo.cab.inta-csic.es. Escuelas y tutoriales: https://svo.cab.inta-csic.es/docs/index.php?pagename=Meetings

JCAAC (2025), **4**, 45–60

JCAAC

SECTION: ASTRONOMICAL COMPUTING

Efemérides precisas del Sol y la Luna

Tomás Alonso Albi[1]

[1]Astrónomo del Observatorio Astronómico Nacional - Instituto Geográfico Nacional, Spain. E-mail: talonsoalbi@gmail.com.

Keywords: programación, programming, efemérides, ephemerides, cálculo astronómico, astronomical computing, Sol, Luna

© Este artículo está protegido bajo una licencia Creative Commons Attribution 4.0 License
Este artículo adjunta un *software* accesible en `https://github.com/JCAAC-FAAE`

Resumen

En este número presentamos algoritmos para el cálculo preciso de la posición del Sol y la Luna para un observador en la Tierra, utilizando el procedimiento de reducción explicado en el número anterior. Para alcanzar la mejor consistencia posible, respecto de la última integración numérica disponible del JPL, se han modificado algoritmos clásicos, de manera notable en el caso de la Luna. Las discrepancias máximas que se obtienen son de 2" para la posición geocéntrica del Sol, y en torno a 15" para la de la Luna, durante al menos 4000 años alrededor del año 2000. También se presentan cambios que se han introducido en ficheros publicados con anterioridad, para corregir algunos problemas.

Abstract

In this article we will see how to compute the accurate positions of the Sun and the Moon for any observer on Earth, using the reduction procedures presented in a previous article in this Section. To achieve the maximum consistency, with respect to the latest JPL numerical integration, some classic algorithms has been modified, especially for the Moon. The maximum discrepancies found are of 2" for the geocentric posicion of the Sun, and around 15" for that of the Moon, during at least 4000 years around the year 2000. Some corrections to files published previously are also presented, solving different problems identified.

1. Introducción

En este número nos centraremos en obtener posiciones precisas del Sol y la Luna para un observador en la Tierra, necesarias para múltiples propósitos como el cálculos de eclipses, entre otros, que iremos viendo más adelante. Estos cálculos no son triviales, puesto que la posición del Sol se ve afectada por el propio movimiento de la Tierra alrededor del centro de masas del sistema que forma conjuntamente con la Luna, mientras que el movimiento de la Luna se ve afectado por la Tierra y otros cuerpos. La Luna ejerce un efecto sobre las mareas terrestres, que también afecta a su movimiento.

Trataremos entonces sobre las posiciones del Sol y la Luna, y veremos que con algoritmos no excesivamente extensos es posible lograr resultados que difieren en pocos segundos de arco respecto de la última integración numérica del JPL. Esto puede conseguirse mediante una ligera modificación o actualización de algoritmos clásicos. Posteriormente veremos cómo calcular también las posiciones precisas de los planetas y otros cuerpos menores, con una precisión similar, y seguiremos con el cálculo de eclipses, justo antes de que lleguen los eclipses de Sol visibles desde España, a partir de agosto de 2026.

Como siempre, el código presentado en esta ocasión puede también encontrarse en el repositorio de GitHub [1] correspondiente a esta sección, el cual será útil para los lectores que quieran utilizarlo tal

cual, o bien reescribirlo en su lenguaje de programación favorito. Es importante comentar que recientemente se han introducido correcciones en el código de cuatro ficheros publicados en números anteriores: *CoordinateSystem*, *EarthAngles*, *EphemReduction*, y *Constant*. En el primero se ha corregido la función *rotate*, que devolvía valores incorrectos al rotar las componentes de velocidad, si se incluían en la entrada. En el segundo se ha corregido un error en la función que calcula la nutación, se ha reemplazado la formulación de la IAU (1976) de la oblicuidad media por la de Laskar (1986), que ya estaba presente pero comentada en el código, y se ha forzado el valor TT-UT1 devuelto en la función *TTminusUT1* al valor 69.2s, cuando la fecha de entrada corresponde a un año entre el 2018 y 2030. Como se explicó en su momento, la diferencia TT-UT1 ha dejado en los últimos años de aumentar al ritmo previsto a largo plazo por el efecto de las mareas. Podría deberse a un cambio en la rotación del núcleo terrestre, o al deshielo de los polos y glaciares, pero el hecho es que la diferencia respecto al valor esperado alcanza ya los 7s, motivo por el que se ha actualizado el código presente en el repositorio. En el tercer fichero se ha establecido un valor mínimo para la distancia entre el objeto calculado y el centro de la Tierra, para el caso en que el objetivo sea el propio centro de la Tierra. En el cuarto fichero se ha retocado ligeramente el valor de la Unidad Astronómica, para hacerla consistente con la integración DE440, sin que haya consecuencias prácticas en los cálculos. Se recomienda revisar estos cambios e introducirlos en cualquier código que los lectores estén desarrollando a partir de ellos.

2. Efemérides precisas del Sol

El algoritmo que se propone en esta sección para calcular la posición del Sol procede de la publicación *Planetary Programs and Tables* [2], de 1986. Se trata de un ajuste a la integración DE200 durante miles de años, entre el 4000 a.C. y el 6000 d.C, que permite obtener la posición eclíptica geocéntrica del Sol para el equinoccio medio de la fecha, que es justo lo que necesitamos para aplicar el procedimiento de reducción de coordenadas explicado en el número anterior [3]. El error máximo de este código es de unos dos segundos de arco en longitud eclíptica. Además, al considerar que la latitud eclíptica del Sol es siempre cero, se introduce un error en latitud que puede llegar en casos extremos al segundo de arco.

En el código propuesto aquí se ha introducido el principal término correctivo en latitud eclíptica, debido al movimiento conjunto de la Tierra y la Luna. Este término es muy fácil de entender: dado que la Tierra gira alrededor del centro de masas que conforma con la Luna, y que ésta gira alrededor de la Tierra con una inclinación de 5 grados respecto de la eclíptica, cuando la Luna está en su máxima latitud eclíptica, la Tierra debe estar en la posición simétrica, con mínima latitud eclíptica. En esta situación, la posición geocéntrica del Sol tendrá su latitud eclíptica máxima, correspondiente por este término a unos 0.6", equivalente a un movimiento de 450 km en la posición de la Tierra, pequeño dado que es mucho más masiva que la Luna, pero considerable. Este término se ha introducido en un método aparte (*getSolarGeocenticEclipticLatitude*) para clarificar que se trata de una corrección al algoritmo original. Con este término no se reduce el error máximo del algoritmo, pero sí se reduce el error habitual que se puede esperar, a valores típicos por debajo del segundo de arco.

Además de esta corrección, la línea 66 presenta una pequeña corrección adicional en longitud eclíptica, descrita con detalle por Meeus [6] e implementada aquí de manera simplificada, para transformar las coordenadas del sistema de referencia utilizado por los autores en [2], denominado equinoccio dinámico J2000 (el mismo que en la teoría analítica VSOP87, desarrollada por los mismos autores para representar las posiciones de los planetas), al sistema de referencia J2000 estándard. Es poco relevante y se incluye por consistencia. Los tests sugieren que reduce las discrepancias con integraciones posteriores.

Pero la corrección más relevante se encuentra en la línea siguiente, la 67, en la que se introduce una corrección totalmente fuera del algoritmo original, con el objetivo de reproducir mejor los resultados devueltos por el servidor Horizons. Esta corrección se comenta con más detalle en la última sección. Se trata de usar el día Juliano en Tiempo Terrestre para aplicar una corrección empírica obtenida mediante

la comparación directa con Horizons. Con esto el error máximo de 2" ocurre en muy raras ocasiones.

En diferentes obras se pueden encontrar algoritmos parecidos, con los cuales sería en principio posible lograr precisiones ligeramente mejores a base de introducir más términos, tanto en longitud eclíptica como en latitud. Las pruebas realizadas dieron como resultado que el código propuesto es posiblemente de los más eficientes que se han desarrollado, pues proporciona las posiciones que necesitamos para la reducción, y contiene un número de términos suficientemente bajo como para presentarlo aquí, con unos 51 en total para las tres coordenadas de longitud, latitud, y distancia. Además, está actualizado para reproducir los resultados de Horizons durante milenios.

El código utiliza el mecanismo de inheritancia que se explicó en el número anterior [3], con el objetivo de reducir la repetición de código. El cálculo de la posición tiene lugar en el método *getBodyPosition*, en el que se usan los términos introducidos en *sunElements* y se llama al método para obtener la latitud eclíptica y corregir la longitud. Las coordenadas devueltas son luego utilizadas en la reducción. Existe otro método llamado *getGeometricEclipticPositionEquinoxOfDate*, el cual devuelve posiciones y velocidades cartesianas geométricas, en vez de coordenadas esféricas corregidas por aberración. Lo utilizaremos en el próximo número para obtener las posiciones de los planetas y poder hacer correcciones de aberración.

El código se completa con el método *getEquinoxesAndSolstices*, el cual es una demostración del uso de un algoritmo que podríamos llamar de fuerza bruta, para obtener una configuración específica, en este caso el instante en que la ascensión recta del Sol (o bien, opcionalmente, siendo algo más rigurosos, su longitud eclíptica, usando la línea comentada) alcanza valores de 0, 90, 180, y 270 grados, correspondientes a los instantes de los equinoccios y solsticios. Dado que la Tierra gira alrededor del Sol aproximadamente un grado por día, o 3600" por cada 86400s de tiempo, con el error del algoritmo los instantes devueltos estarán mal por unos 24s por cada segundo de arco de error, que es el error más habitual. La gran velocidad de los procesadores actuales hace conveniente el uso de este tipo de algoritmos, mucho más sencillos de comprender y limitados por la precisión del algoritmo principal, que otros más complejos desarrollados en el pasado por las limitaciones de velocidad. En un futuro veremos algoritmos de este tipo, aunque más elaborados, para el cálculo de eclipses.

```
1  package journal;
2
3  /**
4   * A class to compute the ephemerides of the Sun. This class uses the code inside
        {@linkplain EphemReduction} by inheritance.
5   */
6  public class EphemSun extends EphemReduction {
7
8      public EphemSun(double jd_utc, double lon, double lat, double alt, TWILIGHT tw,
           TWILIGHT_MODE twm, int tz) {
9          super(jd_utc, lon, lat, alt, tw, twm, tz);
10     }
11
12     // Sun data from "Planetary Programs and Tables" by Pierre Bretagnon and
           Jean-Louis Simon, Willman-Bell, 1986
13     private static double[][] sunElements = {
14         new double[] { 403406.0, 0.0, 4.721964, 1.621043 }, new double[] { 195207.0,
               -97597.0, 5.937458, 62830.348067 },
15         new double[] { 119433.0, -59715.0, 1.115589, 62830.821524 }, new double[] {
               112392.0, -56188.0, 5.781616, 62829.634302 },
16         new double[] { 3891.0, -1556.0, 5.5474, 125660.5691 }, new double[] { 2819.0,
               -1126.0, 1.512, 125660.9845 },
17         new double[] { 1721.0, -861.0, 4.1897, 62832.4766 }, new double[] { 0.0,
               941.0, 1.163, .813 },
18         new double[] { 660.0, -264.0, 5.415, 125659.31 }, new double[] { 350.0,
               -163.0, 4.315, 57533.85 },
19         new double[] { 334.0, 0.0, 4.553, -33.931 }, new double[] { 314.0, 309.0,
               5.198, 777137.715 },
```

```
        new double[] { 268.0, -158.0, 5.989, 78604.191 }, new double[] { 242.0, 0.0,
            2.911, 5.412 },
        new double[] { 234.0, -54.0, 1.423, 39302.098 }, new double[] { 158.0, 0.0,
            .061, -34.861 },
        new double[] { 132.0, -93.0, 2.317, 115067.698 }, new double[] { 129.0, -20.0,
            3.193, 15774.337 },
        new double[] { 114.0, 0.0, 2.828, 5296.67 }, new double[] { 99.0, -47.0, .52,
            58849.27 },
        new double[] { 93.0, 0.0, 4.65, 5296.11 }, new double[] { 86.0, 0.0, 4.35,
            -3980.7 },
        new double[] { 78.0, -33.0, 2.75, 52237.69 }, new double[] { 72.0, -32.0, 4.5,
            55076.47 },
        new double[] { 68.0, 0.0, 3.23, 261.08 }, new double[] { 64.0, -10.0, 1.22,
            15773.85 },
        new double[] { 46.0, -16.0, .14, 188491.03 }, new double[] { 38.0, 0.0, 3.44,
            -7756.55 },
        new double[] { 37.0, 0.0, 4.37, 264.89 }, new double[] { 32.0, -24.0, 1.14,
            117906.27 },
        new double[] { 29.0, -13.0, 2.84, 55075.75 }, new double[] { 28.0, 0.0, 5.96,
            -7961.39 },
        new double[] { 27.0, -9.0, 5.09, 188489.81 }, new double[] { 27.0, 0.0, 1.72,
            2132.19 },
        new double[] { 25.0, -17.0, 2.56, 109771.03 }, new double[] { 24.0, -11.0,
            1.92, 54868.56 },
        new double[] { 21.0, 0.0, .09, 25443.93 }, new double[] { 21.0, 31.0, 5.98,
            -55731.43 },
        new double[] { 20.0, -10.0, 4.03, 60697.74 }, new double[] { 18.0, 0.0, 4.27,
            2132.79 },
        new double[] { 17.0, -12.0, .79, 109771.63 }, new double[] { 14.0, 0.0, 4.24,
            -7752.82 },
        new double[] { 13.0, -5.0, 2.01, 188491.91 }, new double[] { 13.0, 0.0, 2.65,
            207.81 },
        new double[] { 13.0, 0.0, 4.98, 29424.63 }, new double[] { 12.0, 0.0, .93,
            -7.99 },
        new double[] { 10.0, 0.0, 2.21, 46941.14 }, new double[] { 10.0, 0.0, 3.59,
            -68.29 },
        new double[] { 10.0, 0.0, 1.5, 21463.25 }, new double[] { 10.0, -9.0, 2.55,
            157208.4 }
    };

    /**
     * Geocentric position of the Sun. Mean equinox and ecliptic of date. Expansion is
        from "Planetary Programs and Tables",
     * by Pierre Bretagnon and Jean-Louis Simon, Willman-Bell, 1986. The expansion is
        valid from 4000 B.C. to 8000 A.D, but
     * has been modified to reproduce the DE440 integration. The ecliptic latitude is
        not supposed to be 0 as in the previous
     * reference, but computed approximately from the first term of the VSOP solution
        in Jean Meeus's Astronomical Algorithms.
     * Peak errors are below 2".
     * @return Array with ecliptic longitude, latitude (0), distance, and angular
        radius of the Sun.
     */
    @Override
    public double[] getBodyPosition() {
        double t = EarthAngles.toCenturiesRespectJ2000(jd_UT, true) / 100.0;
        double L = 0.0, R = 0.0;

        for (int i = 0; i < sunElements.length; i++) {
            double variable = sunElements[i][2] + sunElements[i][3] * t;
            double u = Util.normalizeRadians(variable);
            double sU = Math.sin(u);
            double cU = Math.cos(u);
            L = L + sunElements[i][0] * sU;
            R = R + sunElements[i][1] * cU;
```

```
61          }
62
63          double tmp = Util.normalizeRadians(62833.196168 * t);
64          L = Util.normalizeRadians(4.9353929 + tmp + L * 1E-7);
65          R = 1.0001026 + R * 1E-7;
66          L -= 0.09 * Constant.ARCSEC_TO_RAD; // Basic transformation from mean
                dynamical equinox to FK5 J2000, as explained by Meeus
67          L += getCorrectionToEclipticLongitude(jd_UT + EarthAngles.TTminusUT1(jd_UT) /
                Constant.SECONDS_PER_DAY);
68
69              // Compute aberration, substracted later
70          double aberration = (993 - 17 * Math.cos(3.10 + 62830.14 * t)) * 1E-7;
71
72          return new double[] { L - aberration, getSolarGeocenticEclipticLatitude(t *
                10), R, Math.atan(695700.0 / (R * Constant.AU)) };
73      }
74
75      /**
76       * Geometric rectangular position of the Sun, mean ecliptic of date.
77       * @param jd Julian day (TT).
78       * @return x, y, z, vx, vy, vz, in AU and AU/d.
79       */
80      public static double[] getGeometricEclipticPositionEquinoxOfDate(double jd) {
81          double t = (jd - Constant.J2000) / 3652500.0;
82          double L = 0.0, R = 0.0, DL = 0.0, DR = 0.0;
83
84          for (int i = 0; i < sunElements.length; i++) {
85              double variable = sunElements[i][2] + sunElements[i][3] * t;
86              double u = Util.normalizeRadians(variable);
87              double sU = Math.sin(u);
88              double cU = Math.cos(u);
89              L = L + sunElements[i][0] * sU;
90              R = R + sunElements[i][1] * cU;
91              DL = DL + sunElements[i][0] * sunElements[i][3] * cU;
92              DR = DR - sunElements[i][1] * sunElements[i][3] * sU;
93          }
94
95          double tmp = Util.normalizeRadians(62833.196168 * t);
96          L = Util.normalizeRadians(4.9353929 + tmp + L * 1E-7);
97          R = 1.0001026 + R * 1E-7;
98          DL = (62833.196168 + DL * 1E-7) / 3652500.0;
99          DR = (DR * 1E-7) / 3652500.0;
100         L -= 0.09 * Constant.ARCSEC_TO_RAD; // Basic transformation from mean
                dynamical equinox to FK5 J2000, as explained by Meeus
101         double b = getSolarGeocenticEclipticLatitude(t * 10);
102         L += getCorrectionToEclipticLongitude(jd);
103
104         // For rectangular coordinates
105         double x = R * Math.cos(L) * Math.cos(b);
106         double y = R * Math.sin(L) * Math.cos(b);
107         double z = R * Math.sin(b);
108         double vx = DR * Math.cos(L) - DL * y;
109         double vy = DR * Math.sin(L) + DL * x;
110         double vz = 0.0;
111
112         return new double[] { x, y, z, vx, vy, vz};
113     }
114
115     private static double getSolarGeocenticEclipticLatitude(double t) {
116         // First 0-order B term from Meeus's Astronomical Algorithms, Appendix II, due
                to the Earth position respect E-M barycenter when the Moon has
117         // a high geocentric ecliptic latitude. Computed with the mean distance of
                moon from its ascending node. At most 0.6"
118         return -280E-8 * Math.cos(3.199 + 84334.662 * t); // t in Julian millenia
119     }
```

```
120
121     private static double getCorrectionToEclipticLongitude(double jd) {
122         return -(9.40 - 0.0000090 * jd + 291E-14 * jd * jd - 321E-21 * jd * jd * jd) *
                Constant.ARCSEC_TO_RAD; // To fit DE441 (Horizons)
123     }
124
125     /**
126      * Returns the dates of the official (geocentric) equinoxes and solstices.
127      * @return Dates of equinoxes and solstices, error always below 50s.
128      */
129     public double[] getEquinoxesAndSolstices() {
130         double jdOld = jd_UT;
131         double[] out = new double[4];
132
133         double prec = 0.1 / Constant.SECONDS_PER_DAY; // Output precision 0.1s,
                accuracy around 1 minute
134         JulianDay julDay = new JulianDay(jd_UT);
135         int year = julDay.year;
136
137         int[] months = new int[] {3, 9, 6, 12};
138         double[] refs = new double[] {0, Math.PI, Constant.PI_OVER_TWO, 3 *
                Constant.PI_OVER_TWO}; // Will search for these RAs
139         for (int i=0; i<4; i++) {
140             julDay = new JulianDay(year, months[i], 18);
141             double jd = julDay.getJulianDay();
142             setUTDate(jd);
143             double min = -1, minT = -1;
144             double stepDays = 0.25, lastRaDif = -1;
145             while (true) {
146                 EphemData data = doCalc(getBodyPosition(), true);
147                 double lon = data.rightAscension; // Using RA, replace with next lines
                        to use ecliptic longitude
148 //double lon = Util.normalizeRadians(CoordinateSystem.cartesianToSpherical
149 //(CoordinateSystem.equatorialToEcliptic(CoordinateSystem.sphericalToCartesian
150 //(data.rightAscension, data.declination), jd))[0]);
151                 double raDif = Math.abs(refs[i] - lon);
152                 if (raDif > Math.PI) raDif = Constant.TWO_PI - raDif;
153                 if (raDif < min || min == -1) {
154                     min = raDif;
155                     minT = jd;
156                 }
157                 if (raDif > lastRaDif && lastRaDif >= 0) {
158                     if (Math.abs(stepDays) < prec) {
159                         out[i] = minT;
160                         break;
161                     }
162                     stepDays = -stepDays / 2;
163                 }
164                 lastRaDif = raDif;
165                 jd += stepDays;
166                 setUTDate(jd);
167             }
168         }
169         setUTDate(jdOld);
170         return out;
171     }
172
173     /**
174      * Test program
175      * @param args Not used
176      */
177     public static void main(String[] args) {
178         // Prepare input data
179         int year = 2020, month = 6, day = 9, h = 18, m = 0, s = 0;
180         JulianDay jday = new JulianDay(year, month, day);
```

```
        jday.setDayFraction((h + m / 60.0 + s / 3600.0) / 24.0);

        double jd_utc = jday.getJulianDay();
        double lon = -4; // degrees
        double lat = 40;
        double alt = 0; // m
        int tz = 3; // h
        TWILIGHT tw = TWILIGHT.HORIZON_34arcmin;
        TWILIGHT_MODE twm = TWILIGHT_MODE.TODAY_UT;

        // Compute the ephemerides data
        EphemSun sunEph = new EphemSun(jd_utc, lon, lat, alt, tw, twm, tz);
        EphemData sunData = EphemReduction.getEphemeris(sunEph);

        // Report
        System.out.println("Sun");
        System.out.println(sunData.toString());

        double[] eqxSols = sunEph.getEquinoxesAndSolstices();
        System.out.println("Sprint equinox: " + EphemData.getDateAsString(eqxSols[0]));
        System.out.println("Autumn equinox: " + EphemData.getDateAsString(eqxSols[1]));
        System.out.println("Summer solstice: " +
            EphemData.getDateAsString(eqxSols[2]));
        System.out.println("Winter solstice: " +
            EphemData.getDateAsString(eqxSols[3]));

/*
Sun
 Az:      285.78955°
 El:      17.4235°
 Dist:    1.0152142 au
 RA:      78.40913°
 DEC:     23.0075°
 Ill:     100.0%
 ang.R:   0.26245394°
 Rise:    2020/06/09 04:46:56 UT
 Set:     2020/06/09 19:43:59 UT
 Transit: 2020/06/09 12:15:21 UT (elev. 72.99474°)

Sprint equinox: 2020/03/20 03:49:45 UT
Autumn equinox: 2020/09/22 13:30:43 UT
Summer solstice: 2020/06/20 21:43:33 UT
Winter solstice: 2020/12/21 10:02:36 UT

Horizons: 2020-Jun-09 18:01:09.200 * 78.40923 23.00749 285.788988 17.372201
    1.01521568029101 69.184687
 */
    }
}
```

Al final del código aparecen los resultados esperados para el ejemplo propuesto en el método *main*, junto con los resultados devueltos por el servidor Horizons. La diferencia es de unos 0.3" en ascensión recta. Recordar que la diferencia TT-UT1 se estableció en nuestros programas en 69.2s, fija durante algunos años, mientras que Horizons utiliza para esta fecha un valor muy similar de 69.18s.

3. La posición precisa de la Luna

El algoritmo propuesto está basado en la obra de Peter Duffet-Smith [4], aunque con modificaciones muy notables para actualizarlo debidamente, y así obtener posiciones consistentes con la última integración numérica del JPL. Este algoritmo utiliza una serie de términos trigonométricos, que conforman una teoría analítica del movimiento de la Luna, que pretende describir interacciones que de otra manera sólo

podrían calcularse mediante la integración numérica de los cuerpos del Sistema Solar. Si es debidamente implementada, una teoría analítica puede describir el movimiento de los astros durante milenios, pero para ello podría requerir de miles de términos trigonométricos para describir las interacciones de menor importancia. En la bibliografía se pueden encontrar otros algoritmos que hemos descartado por ser más laboriosos, como la expansión ELP2000 [5], que es utilizada por Jean Meeus [6] en su libro de manera simplificada. En lugar de utilizar métodos más elaborados, aquí se propone la implementación de [4], pero notablemente modificada para, de algún modo, ajustarla a la integración DE440, dentro de un margen de pocos segundos de arco, durante bastantes milenios. De esta manera obtenemos posiciones más precisas en el pasado que con los otros algoritmos, que están desarrollados para reproducir los resultados de integraciones más antiguas en un arco de tiempo mucho más limitado. Esto también es aplicable a los algoritmos *Standards of Fundamental Astronomy*, o SOFA [7]. El programa sugerido en SOFA para la posición de la Luna no está modificado para mantener la precisión en tiempos remotos, mientras que el del Sol, aunque es más preciso durante 1000 años alrededor de la época actual, también es mucho más extenso. Las modificaciones que veremos no aumentan significativamente la longitud del código.

Para esta adaptación, los valores de diferentes parámetros del Sol y la Luna de la teoría original han sido reemplazados por los valores calculados por S. L. Moshier [8], cuyas expansiones para obtener los valores medios de la longitud o latitud eclípticas de la Luna, entre otras, son válidas durante milenios, y fueron obtenidas de forma consistente con la integración DE405, bastante más evolucionada que la DE200, y relativamente cercana y consistente con DE440. Con eso mejoramos ya la teoría original cuando nos movemos a fechas lejanas. Pero también es necesario introducir términos trigonométricos adicionales en cada coordenada (ver líneas 106-117, 139-141, 176-180), para lograr afinar los resultados, como se describirá con más detalle en la sección siguiente. Estos términos incluyen una corrección secular o parabólica, debido a una estimación imprecisa de la aceleración secular de la Luna en las teorías implementadas décadas atrás, y unos términos trigonométricos adicionales, que logran ajustar los resultados a las integraciones numéricas del JPL. Las correcciones respecto de la integración DE431 fueron originalmente presentadas en un blog del autor algunos años atrás, y se han actualizado aquí para el DE440. Hasta donde sabemos, ni éste ni ningún otro código parecido ha sido nunca publicado en una revista.

El código se completa con métodos para obtener la edad de la Luna, a partir de la diferencia entre las longitudes eclípticas del Sol y la Luna, y las fases lunares, en donde se recurre de nuevo a un algoritmo de fuerza bruta. La precisión en las fases lunares es mejor que el minuto también, dado que, aunque la posición de la Luna pueda alcanzar picos de error 10 veces mayores que en el caso del Sol, se compensa porque la Luna se mueve en el cielo 10 veces más rápido. El repositorio [1] incluye un método adicional que permite obtener las libraciones y el resto de ángulos que definen la orientación del disco lunar para una fecha dada. Estos valores son útiles para dibujar el aspecto aparente del disco para un observador. En el futuro es posible que veamos cómo hacer esto, tanto para la Luna como para el resto de cuerpos.

Una vez más, el método *main* presente al final del listado contiene un ejemplo con la salida que cabe esperar de la ejecución del programa, para los datos de entrada que aparecen.

```
1  package journal;
2
3  /**
4   * A class to compute the ephemerides of the Moon. This class uses the code inside
        {@linkplain EphemReduction} by inheritance.
5   */
6  public class EphemMoon extends EphemReduction {
7
8      /** The set of phases to compute the moon phases */
9      public enum MOONPHASE {
10         /** New Moon phase */
11         NEW_MOON ("New Moon:     ", 0),
```

```
12          /** Crescent quarter phase */
13          CRESCENT_QUARTER ("Crescent quarter:", 0.25),
14          /** Full Moon phase */
15          FULL_MOON ("Full Moon:   ", 0.5),
16          /** Descent quarter phase */
17          DESCENT_QUARTER ("Descent quarter: ", 0.75);
18
19          /** Phase name */
20          public String phaseName;
21          /** Phase value */
22          public double phase;
23
24          private MOONPHASE(String name, double ph) {
25              phaseName = name;
26              phase = ph;
27          }
28      }
29
30      public EphemMoon(double jd_utc, double lon, double lat, double alt, TWILIGHT tw,
            TWILIGHT_MODE twm, int tz) {
31          super(jd_utc, lon, lat, alt, tw, twm, tz);
32      }
33
34      /**
35       * Geocentric position of the Moon. Mean equinox and ecliptic of date.
36       * Based on the Petter Duffet program (and mean elements from S. L. Moshier),
37       * modified to fit DE441 during millenia.
38       * @return Array with ecliptic longitude, latitude (0), distance, and angular
             radius of the Sun
39       */
40      @Override
41      protected double[] getBodyPosition() {
42          double t = EarthAngles.toCenturiesRespectJ2000(jd_UT, true);
43
44          // These expansions up to t^7 for the mean elements are taken from S. L.
                Moshier
45          /* Mean elongation of moon = D */
46          double x = (1.6029616009939659e+09 * t + 1.0722612202445078e+06);
47          x += (((((-3.207663637426e-013 * t + 2.555243317839e-011) * t +
                2.560078201452e-009) * t - 3.702060118571e-005) * t +
                6.9492746836058421e-03) * t /* D, t^3 */
48                      - 6.7352202374457519e+00) * t * t; /* D, t^2 */
49          double phase = Util.normalizeRadians(Constant.ARCSEC_TO_RAD * x);
50
51          /* Mean distance of moon from its ascending node = F */
52          x = (1.7395272628437717e+09 * t + 3.3577951412884740e+05);
53          x += (((((4.474984866301e-013 * t + 4.189032191814e-011) * t -
                2.790392351314e-009) * t - 2.165750777942e-006) * t -
                7.5311878482337989e-04) * t /* F, t^3 */
54                      - 1.3117809789650071e+01) * t * t; /* F, t^2 */
55          double node = Util.normalizeRadians(Constant.ARCSEC_TO_RAD * x);
56
57          /* Mean anomaly of sun = l' (J. Laskar) */
58          x = (1.2959658102304320e+08 * t + 1.2871027407441526e+06);
59          x += (((((((1.62e-20 * t - 1.0390e-17) * t - 3.83508e-15) * t + 4.237343e-13)
                * t + 8.8555011e-11) * t - 4.77258489e-8) * t - 1.1297037031e-5) * t +
                8.7473717367324703e-05) * t - 5.5281306421783094e-01) * t * t;
60          double sanomaly = Util.normalizeRadians(Constant.ARCSEC_TO_RAD * x);
61
62          /* Mean anomaly of moon = l */
63          x = (1.7179159228846793e+09 * t + 4.8586817465825332e+05);
64          x += (((((-1.755312760154e-012 * t + 3.452144225877e-011) * t -
                2.506365935364e-008) * t - 2.536291235258e-004) * t +
                5.2099641302735818e-02) * t /* l, t^3 */
65                      + 3.1501359071894147e+01) * t * t; /* l, t^2 */
```

```
66          double anomaly = Util.normalizeRadians(Constant.ARCSEC_TO_RAD * x);
67
68          /* Mean longitude of moon, re mean ecliptic and equinox of date = L */
69          x = (1.7325643720442266e+09 * t + 7.859398092105242 0e+05);
70          x += (((((7.200592540556e-014 * t + 2.235210987108e-010) * t -
                1.024222633731e-008) * t - 6.073960534117e-005) * t +
                6.9017248528380490e-03) * t /* L, t^3 */
71                      - 5.6550460027471399e+00) * t * t; /* L, t^2 */
72          double l = Util.normalizeRadians(Constant.ARCSEC_TO_RAD * x) *
                Constant.RAD_TO_DEG;
73
74          // Now longitude, with the three main correcting terms of evection,
75          // variation, and equation of year, plus other terms (error<0.01 deg)
76          // P. Duffet's Moon program taken as reference for the periodic terms
77          double E = 1.0 - (.002495 + 7.52E-06 * (t + 1.0)) * (t + 1.0), E2 = E * E;
78          double td = t + 1, td2 = t * t;
79          double M6 = td * Constant.JULIAN_DAYS_PER_CENTURY * 360.0 / 6.798363307E3;
80          double NA = Util.normalizeRadians((2.59183275E2 - M6 + (2.078E-3 + 2.2E-6 *
                td) * td2) * Constant.DEG_TO_RAD);
81          double C = NA + Constant.DEG_TO_RAD * (275.05 - 2.3 * td);
82
83          l += 6.28875 * Math.sin(anomaly) + 1.274018 * Math.sin(2 * phase - anomaly) +
                .658309 * Math.sin(2 * phase);
84          l += 0.213616 * Math.sin(2 * anomaly) - E * .185596 * Math.sin(sanomaly) -
                0.114336 * Math.sin(2 * node);
85          l += .058793 * Math.sin(2 * phase - 2 * anomaly) + .057212 * E * Math.sin(2 *
                phase - anomaly - sanomaly) + .05332 * Math.sin(2 * phase + anomaly);
86          l += .045874 * E * Math.sin(2 * phase - sanomaly) + .041024 * E *
                Math.sin(anomaly - sanomaly) - .034718 * Math.sin(phase) - E * .030465 *
                Math.sin(sanomaly + anomaly);
87          l += .015326 * Math.sin(2 * (phase - node)) - .012528 * Math.sin(2 * node +
                anomaly) - .01098 * Math.sin(2 * node - anomaly) + .010674 * Math.sin(4 *
                phase - anomaly);
88          l += .010034 * Math.sin(3 * anomaly) + .008548 * Math.sin(4 * phase - 2 *
                anomaly);
89          l += -E * .00791 * Math.sin(sanomaly - anomaly + 2 * phase) - E * .006783 *
                Math.sin(2 * phase + sanomaly) + .005162 * Math.sin(anomaly - phase) + E *
                .005 * Math.sin(sanomaly + phase);
90          l += .003862 * Math.sin(4 * phase) + E * .004049 * Math.sin(anomaly - sanomaly
                + 2 * phase) + .003996 * Math.sin(2 * (anomaly + phase)) + .003665 *
                Math.sin(2 * phase - 3 * anomaly);
91          l += E * 2.695E-3 * Math.sin(2 * anomaly - sanomaly) + 2.602E-3 *
                Math.sin(anomaly - 2*(node+phase));
92          l += E * 2.396E-3 * Math.sin(2*(phase - anomaly) - sanomaly) - 2.349E-3 *
                Math.sin(anomaly+phase);
93          l += E * E * 2.249E-3 * Math.sin(2*(phase-sanomaly)) - E * 2.125E-3 *
                Math.sin(2*anomaly+sanomaly);
94          l += -E * E * 2.079E-3 * Math.sin(2*sanomaly) + E * E * 2.059E-3 *
                Math.sin(2*(phase-sanomaly)-anomaly);
95          l += -1.773E-3 * Math.sin(anomaly+2*(phase-node)) - 1.595E-3 *
                Math.sin(2*(node+phase));
96          l += E * 1.22E-3 * Math.sin(4*phase-sanomaly-anomaly) - 1.11E-3 *
                Math.sin(2*(anomaly+node));
97          l += 8.92E-4 * Math.sin(anomaly - 3 * phase) - E * 8.11E-4 * Math.sin(sanomaly
                + anomaly + 2 * phase);
98          l += E * 7.61E-4 * Math.sin(4 * phase - sanomaly - 2 * anomaly);
99          l += E2 * 7.04E-4 * Math.sin(anomaly - 2 * (sanomaly + phase));
100         l += E * 6.93E-4 * Math.sin(sanomaly - 2 * (anomaly - phase));
101         l += E * 5.98E-4 * Math.sin(2 * (phase - node) - sanomaly);
102         l += 5.5E-4 * Math.sin(anomaly + 4 * phase) + 5.38E-4 * Math.sin(4 * anomaly);
103         l += E * 5.21E-4 * Math.sin(4 * phase - sanomaly) + 4.86E-4 * Math.sin(2 *
                anomaly - phase);
104         l += E2 * 7.17E-4 * Math.sin(anomaly - 2 * sanomaly);
105
106         // Ecliptic longitude: additional terms to fit DE404
```

```
107         l += 14.1983 * Math.cos(2.3232 * t + 0.4964) / 3600.0;
108         double ph = 0.5582 + 0.11 * t;
109         l += 7.2167 * Math.cos(33.8624 * t - ph) / 3600.0;
110         // Additional terms to fit DE431
111         l += -0.0759777 * Math.pow(t - 1.541336, 2.0) / 3600.0;
112         // Previous term can be replaced by this one for a better fit between years
                -2000 to 6000 (only for DE431)
113         //l += (-0.0001617 * t * t * t - 0.075925 * t * t + 0.3932 * t - 0.4375) /
                3600.0;
114         l += -0.01 * t * t * Math.cos(anomaly) / 3600.0;
115         // Fit DE440
116         l -= (0.88 - 0.156 * t - 0.00584 * t * t - 0.003128 * t * t * t) / 3600.0;
117         l += 0.25 * t * Math.cos(anomaly) / 3600.0;
118
119         double longitude = l * Constant.DEG_TO_RAD;
120
121         // Now Moon parallax
122         double p = .950724 + .051818 * Math.cos(anomaly) + .009531 * Math.cos(2 *
                phase - anomaly);
123         p += .007843 * Math.cos(2 * phase) + .002824 * Math.cos(2 * anomaly);
124         p += 0.000857 * Math.cos(2 * phase + anomaly) + E * .000533 * Math.cos(2 *
                phase - sanomaly);
125         p += E * .000401 * Math.cos(2 * phase - anomaly - sanomaly) + E * .00032 *
                Math.cos(anomaly - sanomaly) - .000271 * Math.cos(phase);
126         p += -E * .000264 * Math.cos(sanomaly + anomaly) - .000198 * Math.cos(2 * node
                - anomaly);
127         p += 1.73E-4 * Math.cos(3 * anomaly) + 1.67E-4 * Math.cos(4*phase-anomaly);
128         p += -E * 1.11E-4 * Math.cos(sanomaly) + 1.03E-4 * Math.cos(4 * phase - 2 *
                anomaly);
129         p += -8.4E-5 * Math.cos(2 * anomaly - 2 * phase) - E * 8.3E-5 * Math.cos(2 *
                phase + sanomaly);
130         p += 7.9E-5 * Math.cos(2 * phase + 2 * anomaly) + 7.2E-5 * Math.cos(4 * phase);
131         p += E * 6.4E-5 * Math.cos(2 * phase - sanomaly + anomaly)
132                     - E * 6.3E-5 * Math.cos(2 * phase + sanomaly - anomaly);
133         p += E * 4.1E-5 * Math.cos(sanomaly + phase) + E * 3.5E-5 * Math.cos(2 *
                anomaly - sanomaly);
134         p += -3.3E-5 * Math.cos(3 * anomaly - 2 * phase) - 3E-5 * Math.cos(anomaly +
                phase);
135         p += -2.9E-5 * Math.cos(2 * (node - phase)) - E * 2.9E-5 * Math.cos(2 *
                anomaly + sanomaly);
136         p += E2 * 2.6E-5 * Math.cos(2 * (phase - sanomaly)) - 2.3E-5 * Math.cos(2 *
                (node - phase) + anomaly);
137         p += E * 1.9E-5 * Math.cos(4 * phase - sanomaly - anomaly);
138
139               // Parallax: additional terms to fit DE431+
140         double pc = (20 - 1799 * t) * Constant.DEG_TO_RAD;
141         p += 0.0000925 * Math.cos(pc);
142
143         // So Moon distance in Earth radii is, more or less,
144         double distance = 1.0 / Math.sin(p * Constant.DEG_TO_RAD);
145
146         // Ecliptic latitude with nodal phase (error<0.01 deg)
147         l = 5.128189 * Math.sin(node) + 0.280606 * Math.sin(node + anomaly) + 0.277693
                * Math.sin(anomaly - node);
148         l += .173238 * Math.sin(2 * phase - node) + .055413 * Math.sin(2 * phase +
                node - anomaly);
149         l += .046272 * Math.sin(2 * phase - node - anomaly) + .032573 * Math.sin(2 *
                phase + node);
150         l += .017198 * Math.sin(2 * anomaly + node) + .009267 * Math.sin(2 * phase +
                anomaly - node);
151         l += .008823 * Math.sin(2 * anomaly - node) + E * .008247 * Math.sin(2 * phase
                - sanomaly - node) + .004323 * Math.sin(2 * (phase - anomaly) - node);
152         l += .0042 * Math.sin(2 * phase + node + anomaly) + E * .003372 *
                Math.sin(node - sanomaly - 2 * phase);
153         l += E * 2.472E-3 * Math.sin(2 * phase + node - sanomaly - anomaly);
```

```
154         l += E * 2.222E-3 * Math.sin(2 * phase + node - sanomaly);
155         l += E * 2.072E-3 * Math.sin(2 * phase - node - sanomaly - anomaly);
156         l += E * 1.877E-3 * Math.sin(node - sanomaly + anomaly) + 1.828E-3 *
                Math.sin(4 * phase - node - anomaly);
157         l += -E * 1.803E-3 * Math.sin(node + sanomaly) - 1.75E-3 * Math.sin(3 * node);
158         l += E * 1.57E-3 * Math.sin(anomaly - sanomaly - node) - 1.487E-3 *
                Math.sin(node + phase);
159         l += -E * 1.481E-3 * Math.sin(node + sanomaly + anomaly) + E * 1.417E-3 *
                Math.sin(node - sanomaly - anomaly);
160         l += E * 1.35E-3 * Math.sin(node - sanomaly) + 1.33E-3 * Math.sin(node -
                phase);
161         l += 1.106E-3 * Math.sin(node + 3 * anomaly) + 1.02E-3 * Math.sin(4 * phase -
                node);
162         l += 8.33E-4 * Math.sin(node + 4 * phase - anomaly) + 7.81E-4 *
                Math.sin(anomaly - 3 * node);
163         l += 6.7E-4 * Math.sin(node + 4 * phase - 2 * anomaly) + 6.06E-4 * Math.sin(2
                * phase - 3 * node);
164         l += 5.97E-4 * Math.sin(2 * (phase + anomaly) - node);
165         l += E * 4.92E-4 * Math.sin(2 * phase + anomaly - sanomaly - node)
166                         + 4.5E-4 * Math.sin(2 * (anomaly - phase) - node);
167         l += 4.39E-4 * Math.sin(3 * anomaly - node) + 4.23E-4 * Math.sin(node + 2 *
                (phase + anomaly));
168         l += 4.22E-4 * Math.sin(2 * phase - node - 3 * anomaly)
169                         - E * 3.67E-4 * Math.sin(sanomaly + node + 2 * phase - anomaly);
170         l += -E * 3.53E-4 * Math.sin(sanomaly + node + 2 * phase) + 3.31E-4 *
                Math.sin(node + 4 * phase);
171         l += E * 3.17E-4 * Math.sin(2 * phase + node - sanomaly + anomaly);
172         l += E2 * 3.06E-4 * Math.sin(2 * (phase - sanomaly) - node) - 2.83E-4 *
                Math.sin(anomaly + 3 * node);
173         double W1 = 4.664E-4 * Math.cos(NA);
174         double W2 = 7.54E-5 * Math.cos(C);
175
176         // Ecliptic latitude: additional terms to fit DE431
177         double M1 = (124.90 - 1934.134 * t + 0.002063 * t * t) * Constant.DEG_TO_RAD;
178         l += -8 * Math.sin(M1) * Math.cos(node) / 3600.0; // For DE431
179         l += -0.007 * t * t * Math.cos(node) / 3600.0; // For DE431
180         l += 0.48 * t * Math.cos(node) / 3600.0; // For DE440
181
182         double latitude = l * Constant.DEG_TO_RAD * (1.0 - W1 - W2);
183
184         double earthRadius = 6378.1366;
185         double moonRadius = 1737.4;
186         return new double[] {longitude, latitude, distance * earthRadius / Constant.AU,
187                 Math.atan(moonRadius / (distance * earthRadius)), phase};
188     }
189
190     /**
191      * Returns the moon age
192      * @param sun Ephemerides of the Sun, or null for a moon age with less precision
193      * @return Age in days
194      */
195     public double getMoonAge(EphemData sun) {
196         double[] moon = getBodyPosition();
197         double Psin = 29.530588853;
198         if (sun != null) {
199             // Get Moon age, more accurate than 'phase' but we need the Sun position
200             return Util.normalizeRadians(moon[0] - sun.eclipticLongitude) * Psin /
                    Constant.TWO_PI;
201         } else {
202             // Use the phase variable as estimate, less accurate but this is used only
                    when we don't need an accurate value
203             return moon[4] * Psin / Constant.TWO_PI;
204         }
205     }
206
```

```
207         /**
208      * Returns the instant of a given moon phase.
209      * @param phase The phase.
210      * @return The instant of that phase, accuracy around 1 minute or better.
211      */
212     public double getMoonPhaseTime(MOONPHASE phase) {
213         double accuracy = 10 / (30 * Constant.SECONDS_PER_DAY); // 10s / lunar cycle
                length in s => 10s accuracy
214         double refPhase = phase.phase;
215         double oldJD = jd_UT;
216         EphemSun sunEph = new EphemSun(jd_UT, obsLon * Constant.RAD_TO_DEG, obsLat *
                Constant.RAD_TO_DEG, obsAlt,
217                 twilight, twilightMode, timeZone);
218         while (true) {
219             double[] sun = sunEph.getBodyPosition();
220             double[] moon = getBodyPosition();
221             double age = Util.normalizeRadians((moon[0] - sun[0])) / Constant.TWO_PI -
                    refPhase;
222             if (age < 0) age += 1;
223             if (age < accuracy || age > 1 - accuracy) break;
224             if (age < 0.5) {
225                 jd_UT -= age;
226             } else {
227                 jd_UT += 1-age;
228             }
229             sunEph.setUTDate(jd_UT);
230             setUTDate(jd_UT);
231         }
232         double out = jd_UT;
233         setUTDate(oldJD);
234         return out;
235     }
236
237     /**
238      * Test program
239      * @param args Not used
240      */
241     public static void main(String[] args) {
242         // Prepare input data
243         int year = 2020, month = 6, day = 9, h = 18, m = 0, s = 0;
244         JulianDay jday = new JulianDay(year, month, day);
245         jday.setDayFraction((h + m / 60.0 + s / 3600.0) / 24.0);
246
247         double jd_utc = jday.getJulianDay();
248         double lon = -4; // degrees
249         double lat = 40;
250         double alt = 0; // m
251         int tz = 3; // h
252         TWILIGHT tw = TWILIGHT.HORIZON_34arcmin;
253         TWILIGHT_MODE twm = TWILIGHT_MODE.TODAY_UT;
254
255         // Compute the ephemerides data
256         EphemMoon moonEph = new EphemMoon(jd_utc, lon, lat, alt, tw, twm, tz);
257         EphemData moonData = EphemReduction.getEphemeris(moonEph);
258
259         EphemSun sunEph = new EphemSun(jd_utc, lon, lat, alt, tw, twm, tz);
260         EphemData sunData = EphemReduction.getEphemeris(sunEph);
261         moonData.setIlluminationPhase(sunData);
262
263         // Report
264         System.out.println("Moon");
265         System.out.println(moonData.toString());
266         System.out.println(" Moon age: " + (float) moonEph.getMoonAge(sunData) + "
                days");
267         System.out.println();
```

```
268
269         System.out.println("Closest Moon phases:");
270         System.out.println(" New moon:    " +
                EphemData.getDateAsString(moonEph.getMoonPhaseTime(MOONPHASE.NEW_MOON)));
271         System.out.println(" Crescent quarter: " +
                EphemData.getDateAsString(moonEph.getMoonPhaseTime(MOONPHASE.CRESCENT_QUARTER)));
272         System.out.println(" Full moon:   " +
                EphemData.getDateAsString(moonEph.getMoonPhaseTime(MOONPHASE.FULL_MOON)));
273         System.out.println(" Descent quarter: " +
                EphemData.getDateAsString(moonEph.getMoonPhaseTime(MOONPHASE.DESCENT_QUARTER)));
274 /*
275 Moon
276  Az:      64.830414°
277  El:      -57.52784°
278  Dist:    0.0026317919 au
279  RA:      313.06357°
280  DEC:     -21.55362°
281  Ill:     82.01859%
282  ang.R:   0.25632995°
283  Rise:    2020/06/09 23:17:41 UT
284  Set:     2020/06/09 08:10:47 UT
285  Transit: 2020/06/09 03:21:08 UT (elev. 26.579206°)
286
287  Moon age: 18.860058 days
288
289 Closest Moon phases:
290  New moon:       2020/06/21 06:41:23 UT
291  Crescent quarter: 2020/05/30 03:30:03 UT
292  Full moon:      2020/06/05 19:12:33 UT
293  Descent quarter: 2020/06/13 06:23:07 UT
294  */
295     }
296 }
```

4. Comparación con la integración numérica DE440 (Horizons)

La Fig. 1 muestra las discrepancias entre la posición del Sol devuelta por el programa y la proporcionada por el servidor Horizons. Se han considerado tres mil puntos espaciados mil días entre sí, aproximadamente entre los años -2100 al 6100. A la izquierda, las posiciones corresponden al algoritmo original, sin hacer correcciones en la longitud, y manteniendo la latitud en el valor cero. A la derecha aparece el resultado de aplicar las correcciones propuestas en ambas coordenadas, con lo que la discrepancia se mantiene por debajo de los 2" durante al menos cuatro milenios alrededor del año 2000. El valor medio del error en longitud se mantiene en torno a 0.7", mientras que en latitud desaparecen los picos superiores a 1", y los errores pasan a valores habituales de 0.6" o menos. Estas correcciones propuestas representan la diferencia en la Tierra entre ajustar las efemérides DE200 del JPL, ya muy obsoletas, y las DE440. El algoritmo propuesto puede seguir utilizándose para fechas fuera del rango mostrado. Por ejemplo, para el año 4000 a.C. o el 8000 d.C. las discrepancias pueden superar los 3", pero sólo en casos excepcionales.

El mismo procedimiento se muestra para la Luna en la Fig. 2. En el código aparecen las correcciones necesarias para cada coordenada y para diferentes integraciones del JPL, como la DE404, DE431, y DE440, de manera incremental. Los errores se reducen drásticamente al aplicar las correcciones necesarias para cada una, y con todas ellas los picos de discrepancia respecto de la DE440 se reducen por debajo de los 10" (equivalente a 20 km), salvo para más de dos milenios de distancia respecto del año 2000. Se trata de un resultado excelente para un algoritmo compacto, lo que nos permitirá obtener resultados precisos incluso para cálculos de eclipses en épocas remotas. Los residuos aún muestran cierta periodicidad en épocas remotas, lo que indica que con un poco más de trabajo los residuos podrían

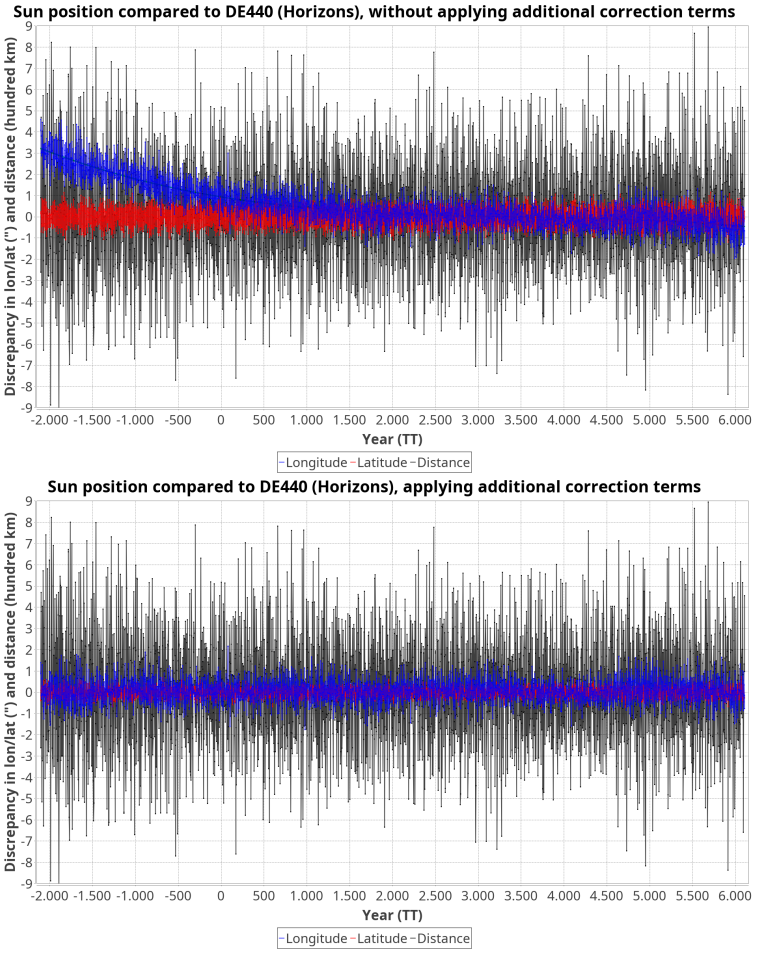

Figura 1. *Diferencia entre las posiciones del Sol calculadas con el programa propuesto y el servidor Horizons. Arriba: sin incluir las correcciones adicionales propuestas. Abajo: tras incluir las correcciones propuestas para la longitud y latitud eclípticas.*

reducirse a la mitad en los extremos del gráfico, pero el resultado ya es muy bueno. Para el año 3000 a.C., o el 7000 d.C., las discrepancias pueden superar los 25", y 1000 años más lejos los 50", especialmente en latitud eclíptica.

References

[1] Repositorio de código en GitHub: https://github.com/JCAAC-FAAE
[2] Planetary Programs and Tables, Pierre Bretagnon and Jean-Louis Simon, Willman-Bell, 1986.
[3] Astronomical computing: Cálculo de efemérides, T. Alonso Albi, JCAAC **3**, 65 (2025).
[4] Astronomy with your Personal Computer, Peter Duffet-Smith, Cambridge University Press, 1985.
[5] ELP 2000-85: a semi-analytical lunar ephemeris adequate for historical times, M. Chapront-Touze y J. Chapront, 1988. https://articles.adsabs.harvard.edu/pdf/1988A%26A...190..342C
[6] Astronomical Algorithms, Jean Meeus, editorial Atlantic Books.

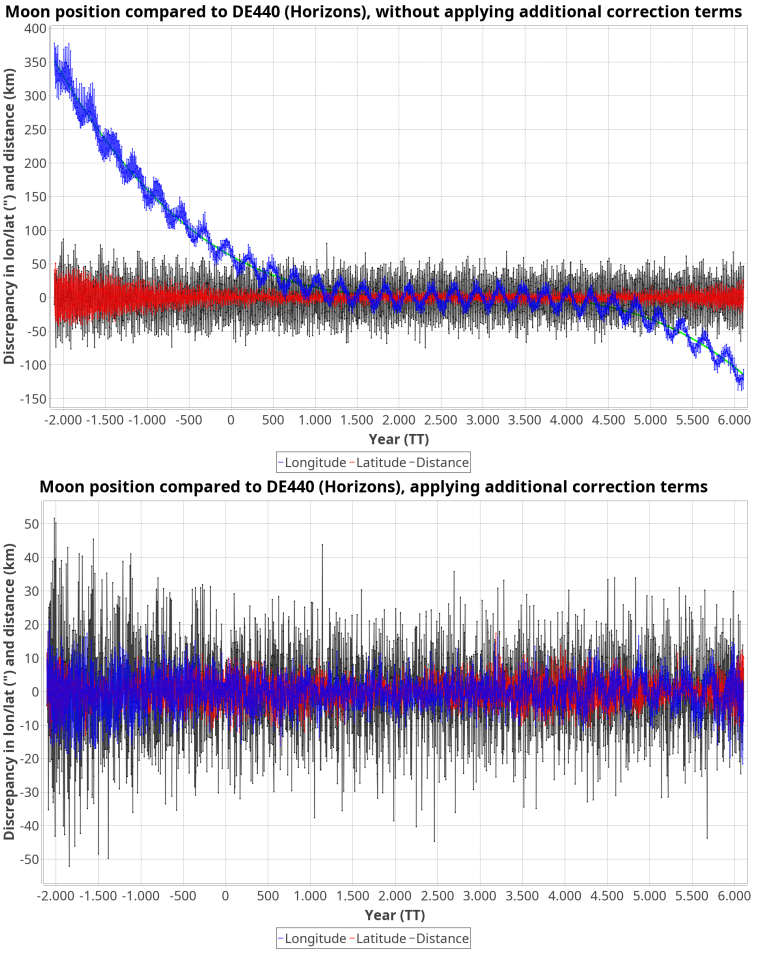

Figura 2. *Diferencia entre las posiciones de la Luna calculadas con el programa propuesto y el servidor Horizons. Arriba: sin incluir las correcciones adicionales propuestas. Abajo: tras incluir las correcciones propuestas para la distancia, y la longitud y latitud eclípticas.*

[7] Algoritmos de SOFA para el cálculo de la posición de la Luna:
http://iausofa.org/2021_0512_C/sofa/moon98.c
y para el Sol:
http://www.iausofa.org/2023_1011_F/sofa/epv00.for
[8] Algoritmos de S. L. Moshier: https://www.moshier.net/

JCAAC (2025), **4**, 61–67

SECTION: SOFTWARE FOR PHOTOMETRY & ASTROMETRY

Astrometría de confirmación de NEOs

Ramón Naves[1] and Montse Campàs[2]

[1]Obs. Montcabrer - MPC 213, Cabrils–Barcelona, Spain. E-mail: ramonnavesnogues@gmail.com.
[2]Obs. Montcabrer - MPC 213, Cabrils–Barcelona, Spain. E-mail: mcampast@gmail.com.

Keywords: astrometría, fotometría, Tycho, NEOs

© Este artículo está protegido bajo una licencia Creative Commons Attribution 4.0 License

Resumen
En esta nueva entrega de la sección, continuamos con el estudio del programa *Tycho*, esta vez en lo que respecta a su aplicación a la astrometría de confirmación de asteroides de tipo NEO.

Abstract
In this new installment of the section, we continue with the study of the Tycho program, this time with regard to its application to the confirmation astrometry of NEO-type asteroids.

1. Introducción

En anteriores números de esta sección de la revista vimos cómo obtener astrometría de cometas y asteroides descubiertos con el programa `Tycho`. En este número usaremos la misma herramienta y veremos cómo proceder cuando se trata de confirmar objetos recién descubiertos que aparecen en las lista del Minor Planet Center (MPC), como son la página NEOCP y subpágina PCCP:

`https://www.minorplanetcenter.net/iau/NEO/toconfirm_tabular.html`

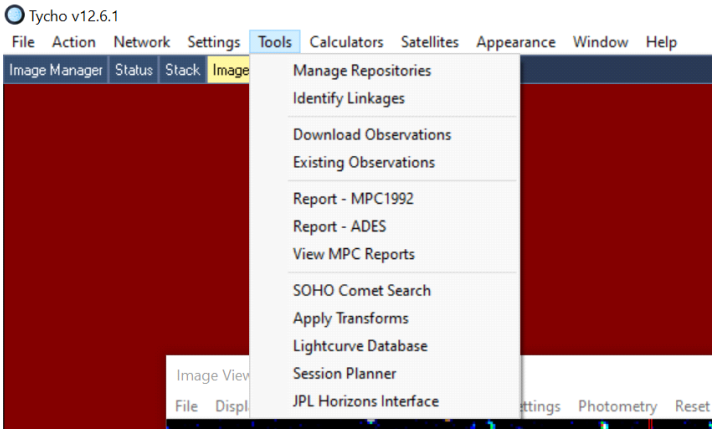

Figure 1. *Extracción de observaciones en el apartado Download Observations.*

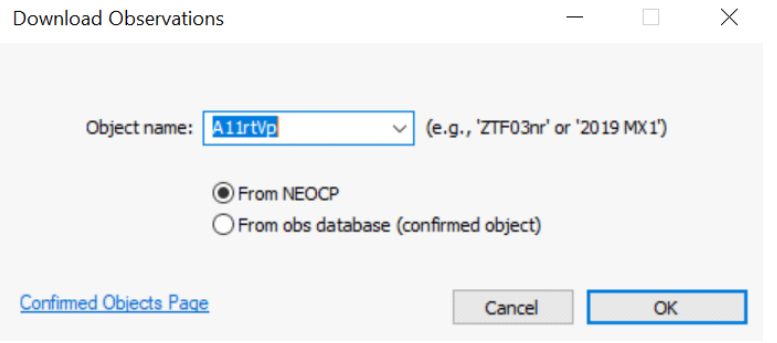

Figure 2. *Sección de Download Observations.*

2. Confirmación de NEOs con Tycho

Cuando queremos confirmar un NEO (asteroide cercano a la Tierra) o un posible cometa de la subpágina PCCP dentro de la página NEOCP, nos encontramos con el problema añadido que el programa Tycho no tiene dicho objeto en su base de datos, por lo que deberemos proceder de manera distinta a la habitual.

Primero procedemos como siempre, tomando la lista de imágenes donde suponemos que tenemos el objeto (cometa o asteroide a medir) a las cuales les habremos realizado el preanálisis con *expres mode* (ver números anteriores).

El segundo paso consistirá en ir al menú *Tools* de la barra superior y marcar *Download Observations*, Fig. 1. Nos aparece entonces un menú (Fig. 2) donde le indicamos el nombre provisional que nos aparece en la pagina del NEOCP, en este caso se trata de un posible cometa denominado A11rtVp.

En el menú debemos clicar la opción *From NEOCP*, aunque se trate como en este caso de un objeto de la subpágina PCCP. Nos aparece entonces una lista con todas las observaciones que se han enviado a la pagina del NEOCP, Fig. 3. Clicamos en *Compute Orbit* (abajo a la derecha) para calcular una órbita preliminar del asteroide, o posible cometa en este caso.

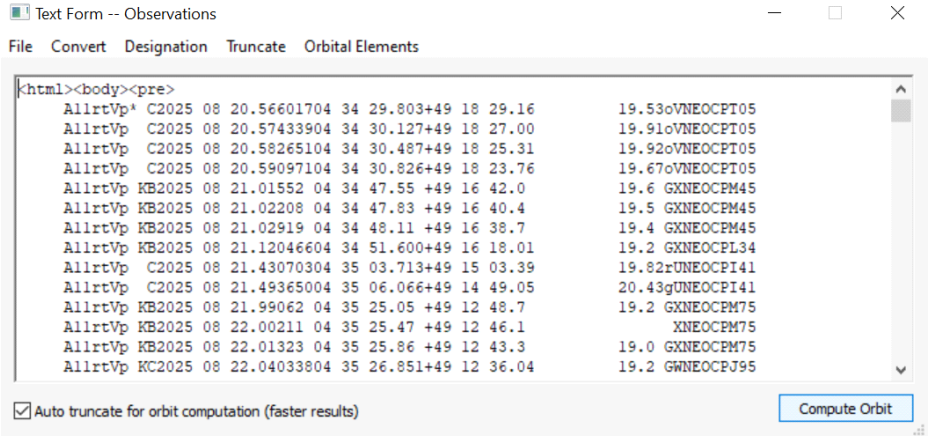

Figure 3. *Listado con las observaciones obtenidas.*

Nos aparecerá un nuevo cuadro con los parámetros orbitales del cometa o asteroide (Fig. 4, panel izquierdo). Clicamos en el menú superior *Ephemeris*, y añadiremos estas a la lista de imágenes clicando en *Attach to Dataset*, Fig. 4, panel derecho.

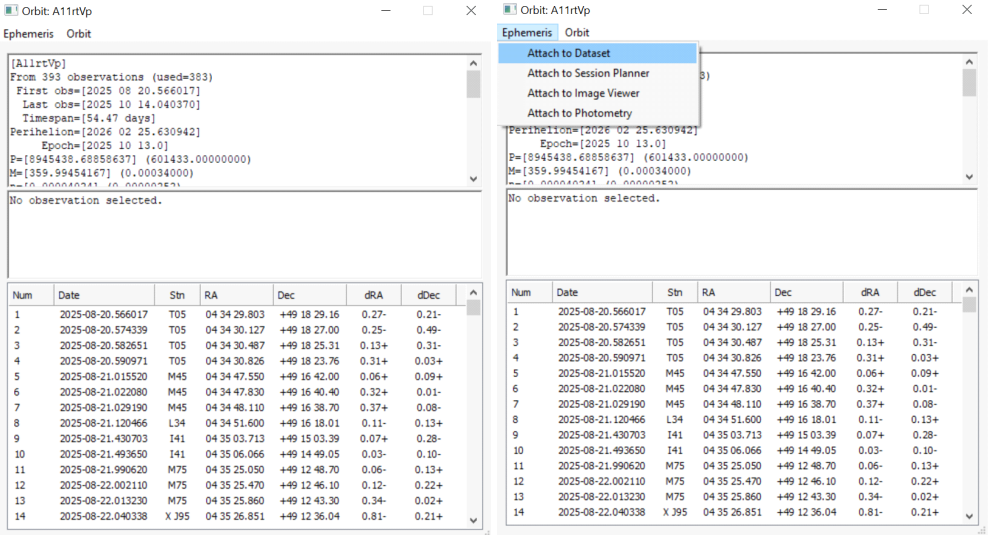

Figure 4. *Menú Orbit y selección de imágenes.*

Podemos ver que en el *Image Manager*, donde teníamos seleccionada la lista de imágenes, se nos muestra con nueva información, Fig. 5.

Importante: Nótese que en *EPH_IN_FOV* aparece escrito *Yes*. Esto simplemente implica que el objeto a medir esta dentro del campo de la imagen, lo cual por desgracia no sucede siempre.

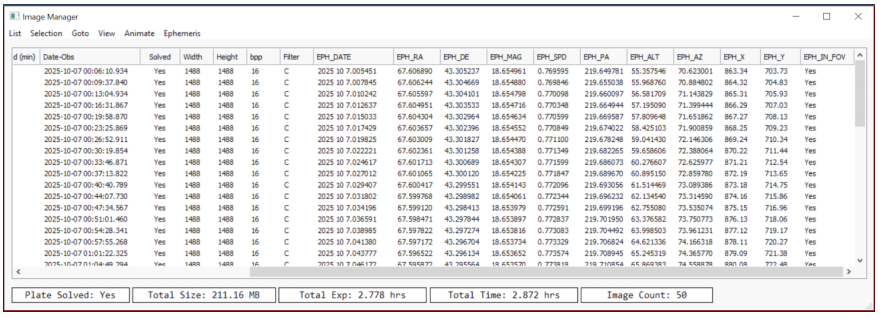

Figure 5. *Ventana del Image Manager.*

A continuación, en el menú *Selection*, clicamos *All*, Fig. 6. Veremos entonces que en el *Image Viewer* nos aparecerá, como por arte de magia, el objeto en cuestión.

En este caso, una vez escogida la opción deseada, *All, 1/2* o *2/2*, nos debería aparecer el objeto, más o menos puntual, en la ventana de imágenes *Image Viewer*, Fig. 7. Si es así podemos clicar en *Create Observation* (abajo a la derecha) para generar la primera observación astrométrica del objeto en cuestión.

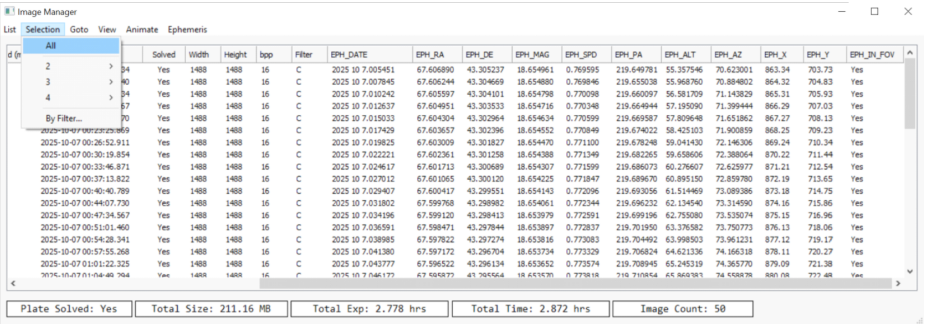

Figure 6. *Selección del objeto en el Image Manager.*

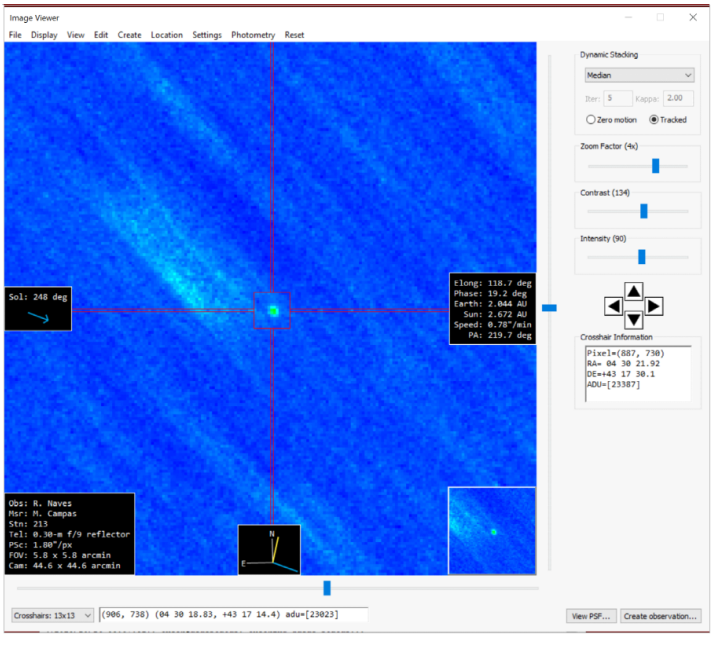

Figure 7. *Objeto seleccionado en la ventana del Image Viewer.*

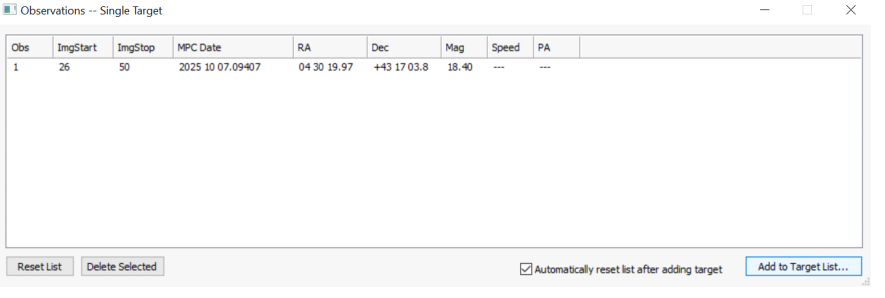

Figure 8. *Menú de observaciones.*

Nos aparecerá entonces el menú de la Fig. 8. Al clicar en *Add to Target List* nos aparecerá el menú mostrado en la Fig. 9, donde pondremos en la casilla inferior el nombre del objeto que figura en la página del NEOCP, en este caso A11rtVp, poniendo especial atención en las mayúsculas y minúsculas.

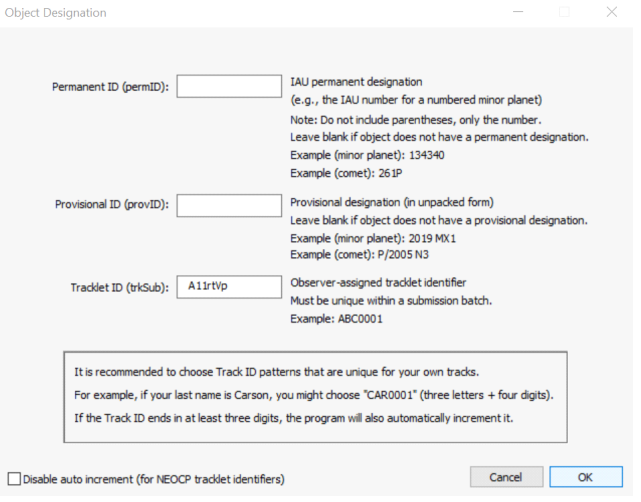

Figure 9. *Selección de nombre para el objeto.*

Otra posibilidad más directa consiste en clicar en la imagen, sobre el objeto a medir con el botón derecho del ratón, y seleccionar *Create Track - Current Position*, Fig. 10. Hay que tener cuidado ya que, dependiendo de lo poblado que esté el campo de estrellas, puede ser mejor usar una opción u otra, así como experimentar con las distintas opciones de apilado (*Dynamic Stacking*), usando *median* (pues minimiza muchos los trazos estelares).

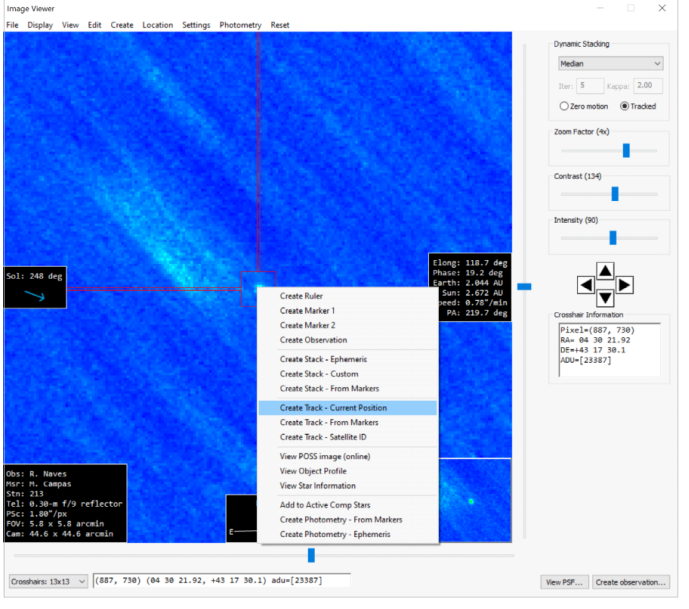

Figure 10. *Selección de Create Track - Current Position.*

Esto nos hace aparecer un menú que ya conocemos, como es el *Track Navigator*, donde procedemos como si de un cometa o asteroide normal se tratara, y clicamos *Verify Track*, Fig. 11.

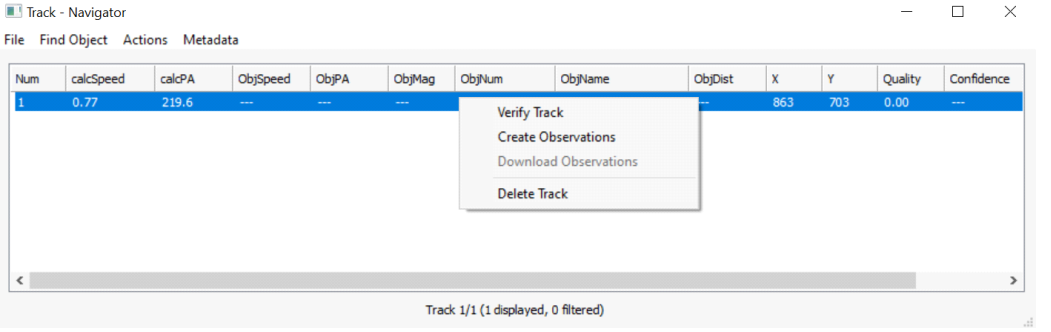

Figure 11. *Menú Track Navigator.*

Esta opción nos abre el menú habitual, Fig. 12, que ya conocemos y hemos explicado en artículos anteriores. Recomendamos usar la opción *3 stacks* y *3 observations*, con idea de obtener tres mediciones astrométricas procedentes cada una de ellas de un apilado independiente el uno del otro, sin imágenes en común.

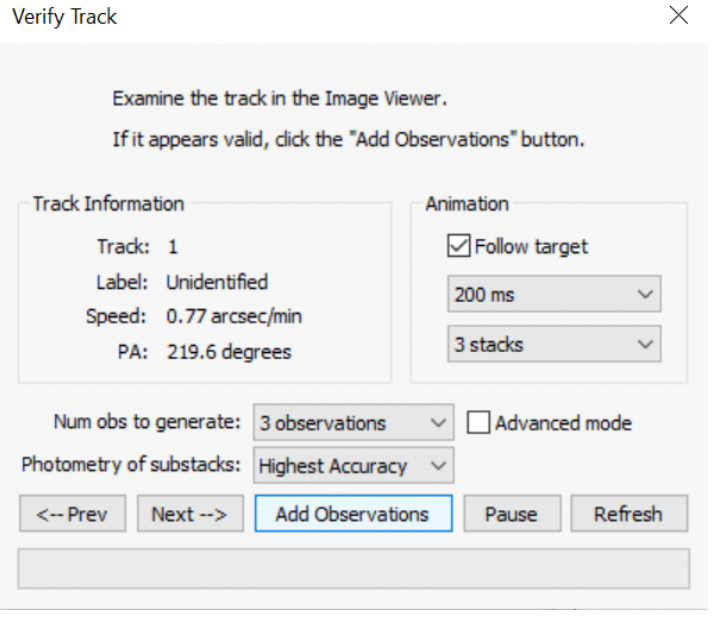

Figure 12. *Menú Verify Track.*

En este mismo menú clicamos ahora sobre *Add Observations*, y nos aparecerá una nueva ventana, Fig. 13, donde pondremos el nombre correcto del objeto en *Tracklet ID*. Esto nos generará tres medidas astrométricas del objeto en cuestión, Fig. 14.

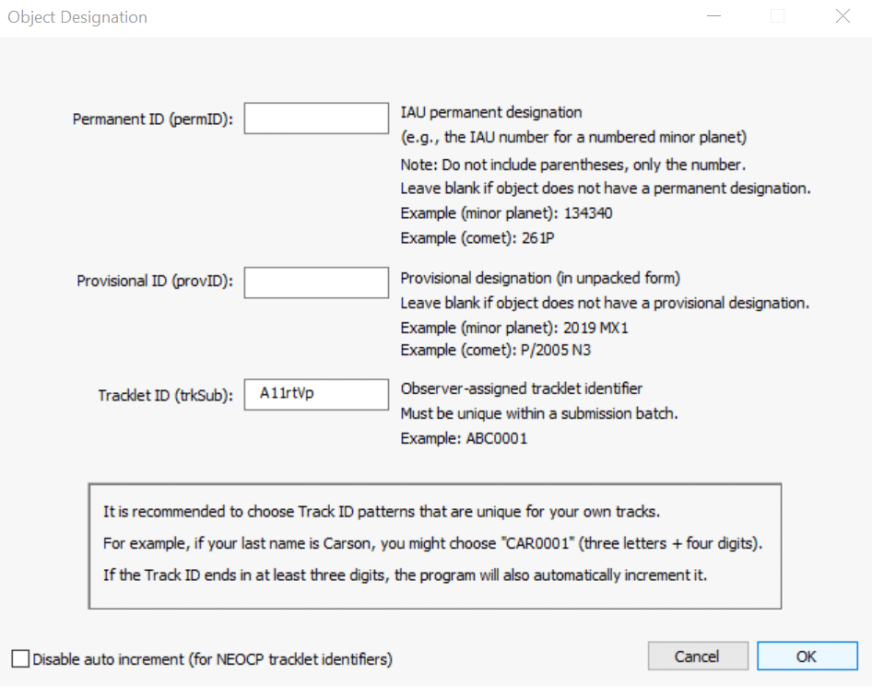

Figure 13. *Selección del nombre correcto para el objeto.*

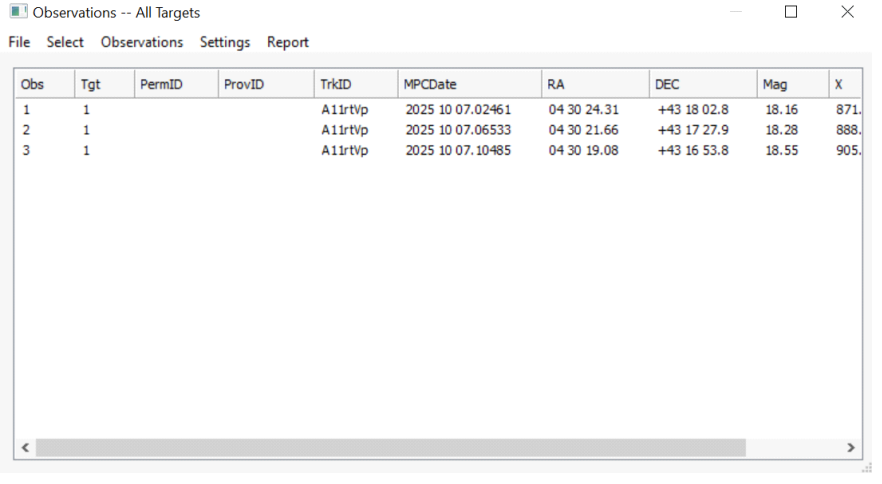

Figure 14. *Medidas astrométricas generadas para el objeto.*

A continuación clicaremos en *Generate ADES Report* (Fig. 15), y se nos abrirá una nueva ventana donde seleccionaremos *Send to MPC*, Fig. 16. En esta nueva ventana es imprescindible seleccionar la opción *PCCP* ó *NEOCP* (según sea el objeto que hayamos decidido medir) en el desplegable *Target Type*. Por último, clicando *Send*, el informe será enviado al MPC y habremos concluido nuestro trabajo.

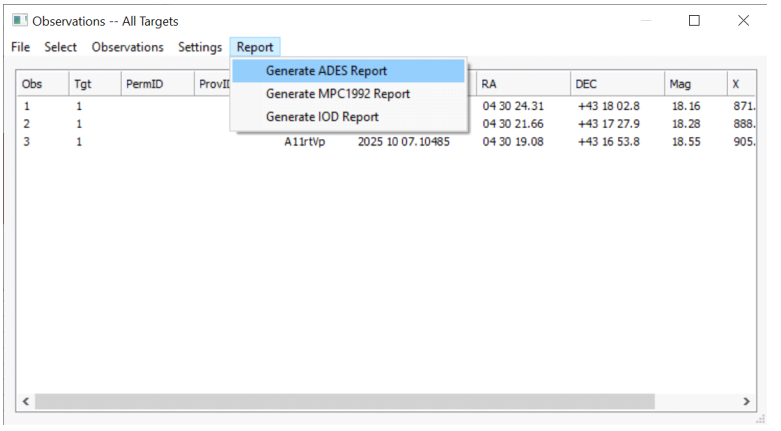

Figure 15. *Selección del nombre correcto para el objeto.*

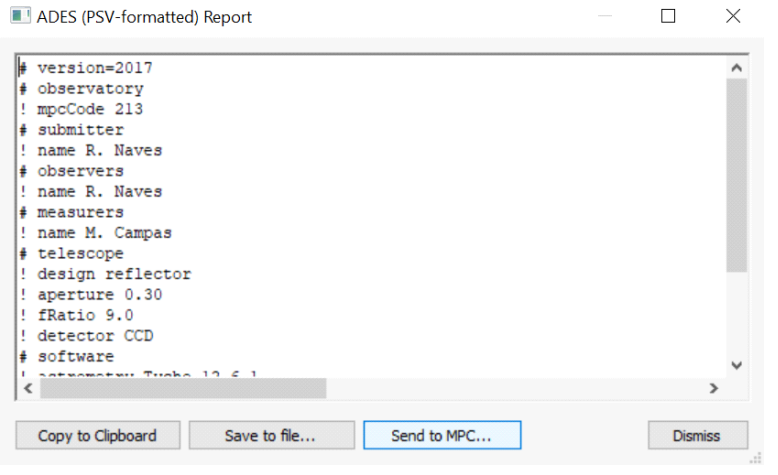

Figure 16. *Generación del informe.*

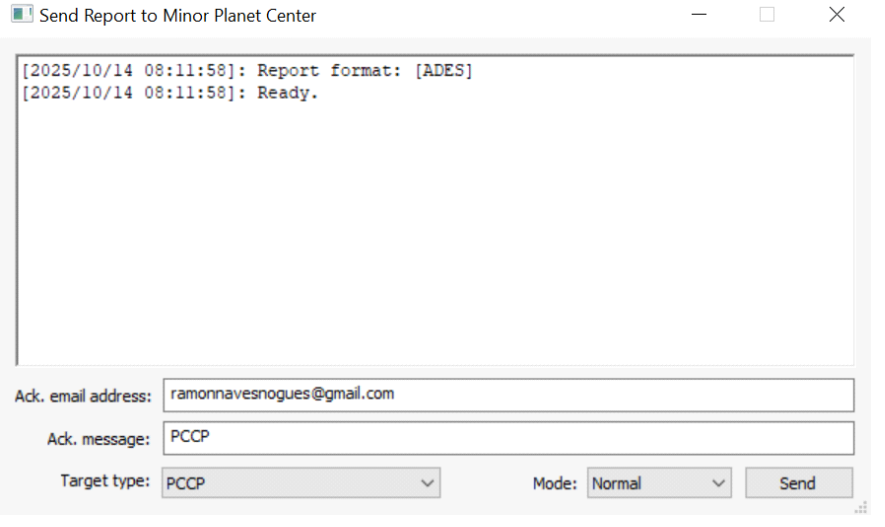

Figure 17. *Ventana para enviar el informe al MPC.*

JCAAC
Journal of Computational Astronomy & Astronomical Computing
Revista de astronomía computacional y cálculo astronómico