International Journal of Geo-Information

MDPI

Article

ASTROLABE: A Rigorous, Geodetic-Oriented Data Model for Trajectory Determination Systems

José A. Navarro [1,*], M. Eulàlia Parés [1,*], Ismael Colomina [2] and Marta Blázquez [2]

[1] Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Parc Mediterrani de la Tecnologia, Av. Carl Friedrich Gauss 7, Building B4, 08860 Castelldefels, Spain

[2] GeoNumerics S. L., Parc Mediterrani de la Tecnologia, Av. Carl Friedrich Gauss 11, 08860 Castelldefels, Spain; ismael.colomina@geonumerics.com (I.C.); marta.blazquez@geonumerics.com (M.B.)

* Correspondence: jose.navarro@cttc.es (J.A.N.); eulalia.pares@cttc.es (M.E.P.); Tel.: +34-93-645-29-00 (J.A.N. & M.E.P.)

Academic Editor: Wolfgang Kainz
Received: 20 December 2016; Accepted: 25 March 2017; Published: 28 March 2017

Abstract: The constant irruption of new sensors is a challenge for software systems that do not rely on generic data models able to manage change or innovation. Several data modeling standards exist. Some of these address the problem from a generic perspective but are far too complex for the kind of applications targeted by this work, while others focus strictly on specific kinds of sensors. These approaches pose a problem for the maintainability of software systems dealing with sensor data. This work presents ASTROLABE, a generic and extensible data model specifically devised for trajectory determination systems working with sensors whose error distributions may be fully modeled using means and covariance matrices. A data model relying on four fundamental entities (observation, state, instrument, mathematical model) and related metadata is described; two compliant specifications (for file storage and network communications) are presented; a portable C++ library implementing these specifications is also briefly introduced. ASTROLABE, integrated in CTTC's trajectory determination system NAVEGA, has been extensively used since 2009 in research and production (real-life) projects, coping successfully with a significant variety of sensors. Such experience helped to improve the data model and validate its suitability for the target problem. The authors are considering putting ASTROLABE in the public domain.

Keywords: trajectory determination systems; data model; genericity; extensibility

1. Introduction

In the context of this paper, Trajectory Determination Systems (TDS) [1] are software components that compute *trajectories* using a variety of sensor measurements as input. The word trajectory usually refers to the path of a flying object. In this article, we will first generalize its use to the path of any moving object through the 3D space as a function of time. In addition, we further generalize it to paths of entities—objects or mathematical abstractions—whose coordinates change with time in some *navigation*, *state* or *parameter* space N; e.g., the actual 3D path plus the velocity and attitude angles of a 3D rigid body. Clearly, this concept is closely related to the mathematical one of a stochastic process since we describe the coordinates of the object as a time-dependent N-valued random variable. More to the point, a TDS estimates optimal—in some sense—trajectories from sensor measurements and mathematical models. A TDS may be either a real-time or post-processing tool.

Today, the applications of trajectory determination are becoming increasingly relevant and demanding: unmanned aircraft navigation [2], precision agriculture [3,4], indoor mapping, indoor positioning [5], robotics—including autonomous vehicle driving— [6] are blooming examples among many others. Moreover, the trend is not bounded by the traditional relative small size of the

professional markets since accurate trajectory determination is also an enabler of mass market applications [7]. In general, these applications cannot be served by mono-sensor systems due to the complexity and variety of the navigation environments. In parallel, sensing technology—including satellite navigation technology—is evolving fast and contributing new sensors either in their very concept or in their features (size, weight, error properties and other) [8,9]. To leverage their potential for trajectory determination in the context of extremely competitive markets, the measurements of new sensors shall be modelled and integrated in TDS as quickly as possible.

The challenges discussed above are not only scientific, but also software engineering ones, both at internal computational and external interface levels. The described situation, large variety and rapid evolution of input data, is not particular to TDS but a general condition of modern software systems. In fact, coping with change and evolution of software systems is an old challenge of software engineering. Many software development methods like Extreme Programming [10] or, more in general, the family of agile software development methods [11] were, to a significant extent, motivated and influenced by the need to cope with unforeseeable change and rapid evolution. Many programming languages were also shaped by these challenges, one of the most popular being object-oriented languages [12].

One solution to the problem is to develop systems—TDS in particular—that are easily extensible. Over the past years we have developed a number of TDS [13–15] that achieved extensibility through genericity. For this purpose, we designed internal estimation engines and external interface engines, for a wide class of data and mathematical models in a way that dealing with new data types or new models neither requires to modify the estimation engines nor the input/output ones [14,15], following a plug-and-play paradigm. As opposed to the simultaneous approach in [13,14], ASTROLABE targets sequential estimation and is based on seed concepts anticipated in [15]. More specifically, under "wide class" we understand an abstraction that supports, in principle, all existing and future sensor models and data for TDS. A similar approach is reported in [16] for a "Reconfigurable Integration Filter Engine" (RIFE). While [16] describes a generic TDS, in ASTROLABE we concentrate on a generic interface independent of the internal navigation engine and target all types of trajectories. A related article [17] mentions the Unified Aiding Information Drives (UAID) generic sensor interface for RIFE. However, in [16] the use of a new sensor requires that the sensor type is available in a system library whereas we pursue a different level of abstraction where new types of sensors can be also integrated in a plug-and-play fashion.

We note that there are other approaches to the extensibility of TDS. One possibility is to develop, for each sensor type, a "maximalist" model and interface with the hope that it is general enough. Another approach is to use sensor replacement models (RPM) [18]. This technique is popular in small-scale photogrammetry and remote sensing. It is based on general analytical formulations, usually unrelated to the physics of the sensor, like rational functions (algebraic fractions where both the numerator and the denominator are polynomials). Its benefits notwithstanding, a RPM is nothing else than a model that works for a wide variety of imaging sensors. A true generic interface shall allow for a seamless use of any type of model and entirely leave to the decision of the sensor specialists which particular model shall be used.

As with software engineering, extensibility is an old issue in geomatics with some early contributions that had influenced our work: in [19] early attempts to classify geodetic entities under a common umbrella were made; in [20] a pioneering generic adjustment engine was presented; in [21] the advantages of the object-oriented approach to generic adjustment systems were discussed; and in [22,23] the underlying general structure of network adjustment systems was analysed. Though loosely related to our problem, we acknowledge the research on ISO standards for geomatics as reported in [24] that have the potential to generate generic interfaces for TDS. These are contributions from geomatics relevant to TDS as they equally apply to real-time and post-processing tools.

Generally speaking and surprisingly to some extent, the literature on navigation and orientation, either in broad general surveys like [25] or even for specific generic systems like [26,27] or [28] do not

tackle generic interfaces. It has been only recently, that the design of plug-and-play systems has made apparent the need for generic interfaces; automation—e.g., in robotics—and harmonisation—e.g., large organisations—are drivers of this. Thus, as already referenced, [17] mentions the UAID generic sensor interface for their universal plug-and-play navigation system. [29] refers to the Defense Advanced Research Projects Agency (DARPA) All-Source Positioning and Navigation (ASPN) programme and its set of generic interface documents known as the "ASPN Interface Control Documents (ICDs)". ASPN was launched by DARPA in November 2010 with the aim of developing low cost sensor fusion technologies and achieving a plug-and-play architecture; i.e., that sensors can be added, removed, or replaced on-the-fly and that their measurements can be included in the input data streams with minimal delay [30]. The DARPA initiative is an independent confirmation of the relevance of the topic of this article.

To at least alleviate this problem, this research work proposes a data abstraction. The identification of the common traits defining the essence of the information provided by sensors leads to a data model general enough to support such diversity, that is, able to represent any kind of sensor whose error distributions may be fully modelled by means of a mean and a covariance matrix (see Section 2) including metadata. Data (values) by themselves are meaningless; a rigorous data model must take care of including the metadata that unequivocally characterize data if it means to be complete. Ignoring metadata has been a constant source of problems in the past—and it still is [31]. Unfortunately, applications in production environments regularly tend to ignore metadata for the sake of performance (The absence of metadata implies a series of assumptions on default values concerning data, as the physical units or the expected reference frame and coordinate system used, among others. As a consequence, it should not be necessary to perform conversion tasks (units, coordinate systems) on input data, thus reducing the computational effort required to provide a solution and, consequently, improving performance.).

Standards covering sensor and observation data models already exist [32,33] and efforts to bring those closer to geomatics also exist [24]. Although these data models would have been powerful (and general and extensible) enough for the purposes of our work, we focus, however, *in a much narrower field of applications* than those targeted by the aforementioned standards; a suitable data model for TDSs needs to specify a fewer number of characteristics to be sufficient, which leads to a more concise specification. In short, ASTROLABE may be seen as a subset of [32,33]. Additionally, terseness is an advantage from several standpoints, as for instance: (1) the amount of information to store when a TDS is working in real time is noticeably reduced, thus decreasing the time needed to save data; (2) transmitting such data through a network connection reduces both bandwidth requirements and transfer time; (3) the burden related to the preparation and management of data and metadata is significantly reduced in actual production environments and (4) the implementation of an Application Programming Interface (API) managing such data model is much simpler and compact.

On the other side, standards designed to represent data from specific types of sensors exist and are widely used. A well known example is the Receiver INdependent EXchange (RINEX) format [34], specifically designed for GNSS range measurements, and its relatives, the Antenna Exchange (ANTEX) format, the IONosphere map EXchange (IONEX) format and the Standard Product 3 (SP3) orbit format. We note that each of these formats are defined independently and that the inclusion of new GNSS systems requires modification and recompilation of the IO software in most cases. Another popular example is the LASer file format (LAS) [35] for point clouds derived from laser scanning. The main advantage of such kinds of standards, being tailored and optimized for specific sensors, is, at the same time, the reason to avoid them; one of the goals of the research work presented in this paper is to devise a generic data model able to manage diversity and evolution.

This paper presents ASTROLABE, developed since 2009 at the former Institute of Geomatics (IG) and since 2014 at the Geomatics Division of the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) with the cooperation of GeoNumerics. Its goal is to create a generic, extensible, complete and efficient data model suitable especifically for TDSs. This work is inspired in the generic data models

and interface specifications of two generic and extensible network adjustment tools, GeoTeX [13] and GENA [14]. At the time when the decision to develop the generic standard was made, to the best knowledge of the authors, there was nothing comparable, academic or industrial. It is true that the standards defined in [32,33] already existed but, as stated above, these were too ambitious for our purposes.

ASTROLABE is the generic interface of CTTC's NAVEGA [15] and GeoNumerics' NEXA trajectory determination systems. In the rest of the article, for the sake of clarity and readability, we will use NAVEGA examples and refer to NAVEGA functions whenever helpful.

This work is structured as follows: Section 2 identifies and introduces the data entities involved in the process of trajectory estimation; these data entities are the pillars on which the ASTROLABE data model relies. Section 3 describes such data model in full detail from the conceptual standpoint, including data and metadata. On Section 4 the ASTROLABE data model is translated into a formal specification comprising a file and a network interfaces, which are briefly described. An almost complete C++ library implementing these two interfaces is briefly discussed in Section 5. Some considerations on the kind of sensors (and related observations) that may be modeled using ASTROLABE are included in Section 6. To finish, conclusions are presented in Section 7. The interested reader may find a more detailed description of the ASTROLABE file interface in Appendix A. An example describing ASTROLABE data and metadata used in a real-life project (GAL) may be found in Appendix B.

2. The Foundations of the Data Model

Shortly stated, the goal of a TDS is to derive trajectories out of sensor data, whatever these are, be it in post-processing or real-time mode.

As any other kind of application, a TDS transforms input data into some kind of result using specialized logic appropriate for the problem to solve. In the particular case of TDSs, these use *observations* coming from *instruments* (sensors) as input, and apply a series of *mathematical models* specific for the intervening observations and instruments to compute the *states*. The computation of these values in consecutive points of time deliver a time-ordered series of states, the trajectory.

The most common methods used to determine trajectories in TDSs are sequential least-squares estimators. Kalman filters [36] are the best known among them. Although these filters do not specifically require Gaussian distributions for errors, they yield the exact conditional probability estimate when this happens. Further, to fix the ideas, for the data model described hereafter, a Gaussian distribution of the observation error will be assumed. Nevertheless, the data model can be used for non-Gaussian situations. In fact, the proposed data model supports a broad range of error probability distribution functions, namely those for which the first and second moments exist.

Four relevant entities arise from the previous description: states (a.k.a. parameters), observations (a.k.a. measurements), instruments and mathematical models. The data model is precisely based on them. In our context, it is convenient to add a "new" one, namely, the *observation equation*, intended to identify a particular combination of observations, states, instruments and mathematical models.

Below, a short formal description of these entities is provided to help to focus the discussion.

- **(Observable and) Observation.** An observable is a numerical property of a physical system that can be determined by a sequence of physical or mathematical operations. Technically, it is a random variable. An observation or measurement is one of the values that an observable or random variable may take; i.e., it is a random sample of a random variable. (For example, the various repeated measurements between A and B of a distance meter instrument—the measured distances—are observations and the abstract concept of distance between A and B is the observable).

  It is worth mentioning that for ASTROLABE, and this is a language abuse, an observation is, in fact, a vector of measurements for a set of observables (an array of observations as defined just above).

In the context of TDSs, typical observables are acceleration, temperature, range or angular speed. Observations for these observables might be -1.3 m/s^2, $+23.556°$, +12,832.34 m and -27.42 rad/s respectively.

- **State.** Similar to observations, a state is a time-dependent random vector whose expectation and covariance have to be estimated from known observations, instruments and mathematical models. Note that in simultaneous, least-squares estimation the states are called parameters.
  Typical states in TDS environments are position or attitude random vectors.
- **Instrument.** An instrument (or sensor) is a device used to measure some observable, thus delivering observations. From the data abstraction point of view, it is an entity which contains the constants that characterizes an actual instrument.
  Within the context of TDSs, inertial measuring units, gyroscopes or GNSS receivers are examples, among others, of instruments.
- **(Mathematical) Model.** A mathematical description of a system or process. In the context of ASTROLABE, it is a stochastic equation or a stochastic differential equation; traditionally, this has also been described as a functional model plus an stochastic model.

The last data entity in ASTROLABE's data model is the observation equation which, in our context, is defined as:

- **Observation equation.** A materialization of a given mathematical model—stochastic or stochastic differential equation— for a particular set of measurements, instruments or states.

The last part of the former sentence introduces a new facet in the definition of observation equations. From the processing standpoint these may be viewed as *commands* or *triggers* that tell the trajectory determination software that the right moment to derive a new state has come (and that the information to use is the one specified by such equation). A series of observations, all of them including measurements reported by—probably—different, cooperating sensors, may be held in stock until the best moment to derive the new state comes. This moment is reported by the observation equation. This is specially important when working with systems that hybridize a set of sensors to compute trajectories; some usual examples combine, for instance, GNSS receivers, inertial measuring units and magnetometers for such purpose. The use of the simultaneous data reported by these sensors noticeably increases the quality of the solution (see [37]).

Figure 1 depicts graphically an hypothetical observation equation. This example assumes that there exist j different kinds of models, l kinds of observations, o different kinds of states and z kinds of instruments. The observation equation in the figure relates the model whose identifier is h, two observations with identifiers 1 and k respectively, two states, whose identifiers are m and o and, finally, a single instrument with identifier y.

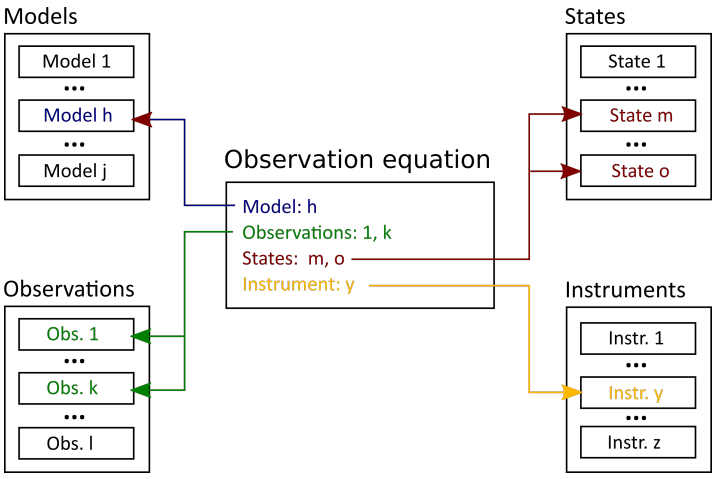

Figure 1. Example of an observation equation.

3. The Data Model in Detail

The sections to follow will describe in detail the observation, state, instrument and observation equation entities just introduced. Such a description takes place from the data and metadata standpoints; it is the result of the abstraction process that leads to the ASTROLABE data model. However, before such a description may be detailed, it is necessary to discuss how data and metadata are identified and cross-referenced. Section 3.1 will introduce the [meta]data identification mechanism used in ASTROLABE. Sections 3.2–3.5 will then describe both data and metadata.

3.1. On Data Identification and Cross-Referencing: Types, Identifiers and Instance Identifiers

ASTROLABE relies on a hierarchical, three-level mechanism to unequivocally identify and cross-reference data. Metadata employ two of these levels, namely types and identifiers, while data complete the identification triplet incorporating instance identifiers.

A type is a code tagging a unique kind of object (be it an observation, state, instrument or model) with a specific, and also unique, set of properties. An example of such object would be a certain kind of instrument, e.g., a temperature-compensated barometer. A unique type code would be assigned to this kind of barometer. Should a different kind of barometer exist, as for instance, one whose measurements were not compensated by means of temperature, a new, different type code should be used. Examples of these type codes could be `barometer_t_compensated` and `barometer_basic` respectively.

There may exist, nonetheless, many brands and models of barometers that use temperature to correct their measurements. All these would be incarnations of `barometer_t_compensated` (instrument) objects, that, in spite of sharing a common type, might behave differently—e.g., being more or less accurate or delivering data using different physical units. In other words, even though all the temperature-compensated barometers may be described by a common set of properties, the actual values of these properties may differ.

The identifier is used to take these differences into account. An identifier is a unique code used to tell apart different incarnations of objects (temperature-compensated barometers in the example above) that, in spite of being characterized by the same set of properties have different values for these.

A tuple composed by a type and an identifier unequivocally identifies any kind of object in ASTROLABE metadata. Figure A2, lines 01–29, shows the full XML specification of temperature-compensated barometer observations using the ASTROLABE file interface (Although this section takes care of describing a data model that may be implemented in many different ways, examples are presented using the XML syntax defined by ASTROLABE to materialize its file interface. Such file interface is detailed in Section 4.1.). Note the type/identifier codes in lines 02 and 04 of this example.

Actual data records use the aforementioned identifier to characterize themselves. That is, these records include the identifier as an extra field pointing to the metadata that will serve to describe them. Figure 2 depicts the type/identifier hierarchy and shows as well how actual data use the identifier code to state what kind of information is stored in data records.

The identification schema just depicted is not complete yet from the data (not metadata) standpoint, since it does not take into account that multiple, identical instruments might be used simultaneously when collecting data. That is, data originating from two or more identical instruments—sharing the same type and identifier codes—may be present in a dataset. To solve this problem, the third element in ASTROLABE's data identification schema is used. It is the instance identifier.

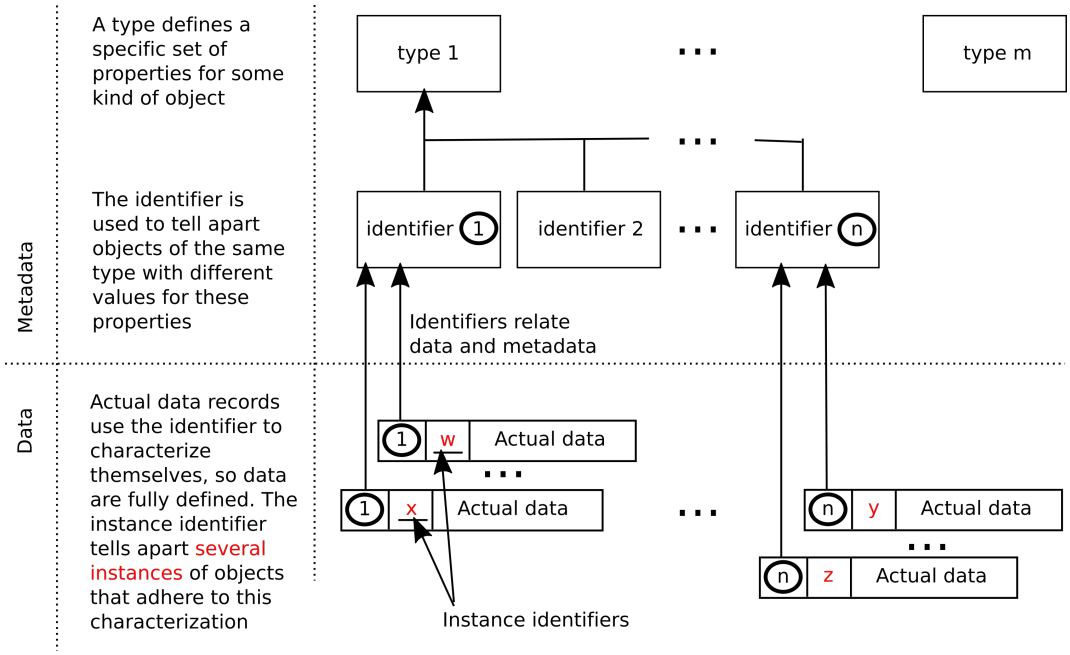

Figure 2. Types, identifier and instance identifier. Cross-referencing from actual data.

The instance identifier is a code used to distinguish between several instances of objects whose types and codes are identical. It purpose is to allow multiplicity.

Thus, actual data records use a combination of an identifier plus an instance identifier to fully characterize themselves. The identifier points to metadata (and therefore, to the exhaustive description of the information the data record contains) while the instance identifier is used to discriminate between several instances of identical data sources. Figure 2 depicts graphically this situation. Figure 3 shows several actual barometer data records in ASTROLABE XML format including the same metadata identifier (baro1, see Figure A2) but two different instance identifiers (1 and 2). That is, data from two identical barometers have been collected at times 124.88 to 124.92 (two readings from barometer 1, only one from barometer 2).

Note that the values of the instance identifiers may be chosen arbitrarily, providing these are different for every object instance they represent.

```
<l id="baro1" n="1"> 124.88 23.44 1023.442 0.3 </l>
<l id="baro1" n="2"> 124.90 23.45 1023.450 0.3 </l>
<l id="baro1" n="1"> 124.92 23.45 1023.443 0.3 </l>
```

Figure 3. Data: identifier and instance identifiers in barometer data.

3.2. Data: Observations

Observations are one of the main pivots the ASTROLABE data model turns around; not in vain, observations are the primary source of data used to derive results, either in TDS or in many other kind of software tools. In fact, many of these tools consider observations as the unique source of information—often forgetting metadata completely. Therefore, finding a powerful abstraction for observations is of capital importance for a data model that means to be generic and extensible.

The type of information provided by sensors is—apparently—very different. It is evident that a barometer will not deliver the same kind of data as with a spectrometer, a gravimeter or a densitometer, to mention some. Moreover, and now moving to the software realm, the algorithms used to deliver useful results out of sensor readings are different too. Taking into account, however, how information

is processed is not the task of a data model; it must deal only with those aspects that are exclusively intrinsic to data.

In spite of the aforementioned dissimilarities in observations, a common structure, their essence, may be identified in all of them. The observations that may be modelled by ASTROLABE consists of a set of expectations (The word expectation in the context of this paper stands for *the best estimate of the expectation of the random variable associated to the measurement*.) (the measurements reported by the device) and an assessment of its quality (the covariance matrix (This is so because ASTROLABE assumes observations with error distributions fully defined by the first and second moments; therefore, an observation always consist of an array of expectations plus a covariance matrix.)). No matter what the observation is, these elements will always be present (Some sensors may not provide actual values for the covariance matrices that should accompany every measurement. In such situations the nominal quality values reported by the manufacturer may be used. Such default values may be specified by means of metadata. For instance, the example in Figure A2 (line 18) provides a default value (1 hPa) for the standard deviation component of the covariance matrix by means of the <c> tag.).

Of course, the number of values present in each kind of observations will vary; a 3-axial accelerometer will report three readings (acceleration around the three axes) while a thermometer will provide just one (the temperature). Even when considering sensors of the same type providing the same number of readings, the units used to deliver such values may vary: m/s^2 or cm/s^2 in the case of accelerometers, for example.

These differences are, however, accidental and may easily modeled through the proper use of metadata (see Section 3.5). What remains is the fact that every observation will be composed of a set of expectations and the related covariance matrix.

Therefore, the first approach to define the common structure for observations would state that it is composed of two elements:

observation: (expectations, covariance matrix)

It is necessary to be able to tell apart different kinds of observations; otherwise, it would be impossible to interpret the semantics of these measurements: observations from a distancemeter would be indistinguishable from those coming from an odometer. To do so, the identifier and instance identifiers defined in Section 3.1 are introduced:

observation: (**identifier**, **instance identifier**, expectations, covariance matrix) (Note that the **boldfaced** words in definitions are used to denote changes with respect to previous ones.)

The unique identifier will point to the appropriate metadata, where aspects as the dimension of the vector of expectations or the units used—as well as other accidental properties—will be specified. The instance identifier, as stated in Section 3.1, will serve to tell apart observations originating from different instances of equivalent data sources (as for instance, two identical barometers).

Note that the need to identify different kinds of observations responds to the (geodetic) principle of heterogeneity (Heterogeneity refers to the use of different types of observables for the determination of the same set or subset of parameters or states. Better systematic error correction is also the goal of heterogeneity.) on which the ASTROLABE data model relies. The need to identify several instances of the same kind of observation responds to the (geodetic) principle of redundancy (Redundancy refers to the repetition of measurements, apparently superfluous and unneeded, with the objective of mitigating the random dispersion of the observables (random variables) being measured (samples of the observable). Better outlier detection is also the goal of redundancy.).

Time is a **requirement** in TDSs because of the nature of their purpose. Observations must therefore be time-tagged to register the moment when the sensor delivered its readings.

observation: (identifier, instance identifier, **time**, expectations, covariance matrix)

Sometimes, it is interesting to keep track of extra data that, although not an intrinsic component of an observation, helps to complement it. This auxiliary data may be of any kind. For instance, an accelerometer delivering acceleration measurements may be seriously affected by temperature. This observable (temperature) is not, from the conceptual standpoint, part of the acceleration observation; however, ignoring it may lead to incorrect results when used by a TDS. On the contrary, extending the observation with this information may help the TDS to, for instance, apply a calibration method to correct the data coming from the accelerometer.

The former is just an example to illustrate why auxiliary information should be a companion of pure observation data. Auxiliary values are called *tags* in the context of ASTROLABE. The number of tags is arbitrary; other accelerometers, for instance, might be affected by other kinds of observables. Other sensors may need not auxiliary values at all.

Therefore, the ASTROLABE data model makes tags an integral part of its observation data entity; the number and properties of tags—that may be zero—for each kind of observation is defined by means of metadata (see <t_spec>, Figure A2, lines 20–28).

observation: (identifier, instance identifier, time, **tags**, expectations, covariance matrix)

Leaving aside metadata, and from the conceptual—ASTROLABE's, at least—standpoint, there is nothing else that an observation should include. There are, however, two more items that the current observation model has adopted: an extra event tag and an activation flag.

The typical dataset used in trajectory determination (either a file or a network stream, for instance) will include *at least* two kinds of data entities: observations and observation equations. Since both data entities will be merged in datasets, a mechanism to tell apart these entities is necessary. A simple marker identifying the kind of entity it precedes will suffice for that purpose. ASTROLABE names this marker as event tag.

The final element composing the observation model is provided just for (processing) convenience purposes: it is the activation flag. When processing data, some observations may be detected as wrong ones (for instance, because a magnetometer starts producing invalid readings when approaching a powerful source of electromagnetic interferences, as power lines). If these observations are not removed from the computation of the output trajectory the results will be distorted. Of course, it is possible to eliminate such observations from the input dataset thus removing their harmful effects; however, this distorts how data was captured, since the original observation will no longer be available in the input dataset.

The solution to this—apparent—problem is to provide with the aforementioned activation flag. An observation set to "inactive" by the human operator must be ignored by any TDS just as it would have never existed.

This leads to the final step in the definition of the observation entity data model as seen by ASTROLABE:

observation: (**event tag**, **activation flag**, identifier, instance identifier, time, tags, expectations, covariance matrix)

Table 1 summarizes the discussion above.

Table 1. The observation entity data model.

Item	Description
Event tag	Identifies the kind of data entity (observation).
Activation flag	Enables or disables the observation for processing purposes.
Identifier	Defines the kind of observation (points to metadata).
Instance identifier	Tells apart multiple occurrences of identical data sources.
Time	Time stamp; time when the measurements were made.
Tags	Data that is not part of the observation but complements it.
Expectations	The observed measurements.
Covariance matrix	Estimated error of the expectations.

Figure 4 shows two examples of observations using the ASTROLABE XML syntax. The metadata describing these of observations may be found in Figure A2. Section 3.5 describes metadata in detail.

It is possible to know that these are observations because of the <l> tag opening (and closing) data records. This XML tag corresponds to the event tag described above. The active flag is represented as the "s" (status) attribute, which may take two values: "a" for active or "r" meaning removed or inactive. Both observations include their identifiers (pointing to the metadata that characterize them) in the "id" attribute. The first observation states that its identifier is "baro1" while the second one identifies itself as "imu1". To link these observations to specific instances of sensors the instance identifier, represented by the "n" attribute, is used, whose values, for the "baro1" and "imu1" observations are, respectively, 32 and 41.

The next field is the time stamp (124.88 in both observations). In the case of the "baro1/32" observation, one tag (auxiliary value) has been included (23.44); this is so because the sensor related to this observation is a temperature-compensated barometer; the tag corresponds to the temperature reading at the moment when atmospheric pressure was measured. There are no tags for the "imu1/41" observation.

Then, the measurements themselves are included. In the case of "obs1/32" a single value (pressure, 1023.44) is provided. The "imu1/41" observation, on the contrary, provides the readings for three angular velocities and accelerations (0.01 0.02 0.015 0.32 0.43 9.95).

Lines 2 and 5 provide the covariance matrices (a single standard deviation in the case of "obs1/32") for these observations respectively.

See Figure A1 for a complete ASTROLABE XML example showing both observations and observation equations.

```
1 <l s="a" id="baro1" n="32"> 124.88 23.44 1023.44
2                                        0.3                          </l>
3
4 <l s="r" id="imu1"  n="41"> 124.88       0.01 0.02 0.015 0.32 0.43 9.95
5                                        1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
```

Figure 4. Data: examples of observations in XML syntax.

3.3. Data: States, Instruments and Mathematical Models

States and instrument data may be modelled using exactly the same structure just discussed in Section 3.2, at least in the context of ASTROLABE and its goals (In a trajectory determination and, in general, in a parameter or state estimation "ecosystem" the role of measurements, states and instruments uses to commute. In other words and more specifically, an instrument—i.e., its calibration parameters, time varying or not—could be the result of a trajectory determination exercise. However, in a next software run, an instrument calibration set of states can be seen as an instrument constant. Or for example, a GNSS-derived position can be a state of a GNSS trajectory determination based

on GNSS measurements. However, in a next INS/GNSS loosely-coupled trajectory determination run, those position states become the position observations. In ASTROLABE a trade off between the engineering model and the mathematical model is made. We could refer to the measurements, states and instruments as known or unknown stochastic processes. However, by telling apart the measurements, from the states, from the instruments we facilitate the use of the standard and the modelling process. This is why, mathematically and structurally, observations, states and instruments are alike.). The following are the definitions for state and instrument data (which have been provided for the sake of completeness, since these are exactly the same as the definition of an observation:)

state: (event tag, activation flag, identifier, instance identifier, time, tags, expectations, covariance matrix)

instrument: (event tag, activation flag, identifier, instance identifier, time, tags, expectations, covariance matrix)

In the case of observations and states, the coincidence of their definitions should not surprise the reader; in fact, when working in post-processing environments, TDSs use to compute trajectories in three steps, the so-called forward, backwards and smoothing ones. The output (trajectories, made of states) produced by the two first steps become the input of the third one. *The role of states is thus changed to observations at that point.*

Observations and states may easily be told apart by the context in which these both entities appear. Observations will always be included in input files/network streams. States are always the output of TDSs, so these will always be found in output files/network streams. For an example of states written in the ASTROLABE XML syntax, the reader may refer to Figure 4, which, in fact, depicts observations.

Instruments, on the other side, are not random variables. That is, time does not affect instrument data, which is considered constant information. From the structural standpoint, instrument data is, once more, a set of values and their quality information (that is, the expectations and covariance matrices found in observations and states). Note that, in the case of instrument (constant) data, the covariance is considered by ASTROLABE as a mere indication of the quality of the list of values.

Since instruments must also be identified, the three level hierarchy used in observations and states is adopted here. Tags are provided for the same reason as with observation data; the difference, in the case of instrument constants, is that these tags are purely informative, and contain their values at the moment the aforementioned constants were computed—that is, when the instrument was calibrated.

The use of time stamps and activation flags as an integral part of the instrument data needs some justification. Although purely informative, time stamps are used to keep track of the time when the instrument constants were calculated. This is very important in the case of unstable instruments that must be recalibrated often; an explicit calibration time helps to avoid problems due to the use of outdated information. Note, finally, that the time stamp must use the same reference frame and coordinate system that those used in observation records.

The presence of the activation flag might seem controversial: if observations from some kind of instrument are included in a dataset, the constants defining the instrument itself should be included as well. On the contrary, if the observations contains no data from some kind of instrument, the inclusion of the instrument data themselves would be superfluous. Therefore, the possibility of activating or deactivating instruments seems useless. This is true; however, this flag is included for convenience, purely practical reasons: it allows storing the constants of several instruments in the same single file. Depending on the sensors (and therefore, observations) used to compute a particular trajectory, the instruments available in such file may be activated or deactivated correspondingly, and the unique instruments file reused easily.

Typical examples of instruments constants would be the focal length of a camera or the position of the center of a GNSS antenna.

The following are two instrument data records that adhere to the definition above but include no tags. The instrument is described in the metadata file fragment shown in Figure A3, more specifically in lines 138–171.

<l id="p0_h0" n="51"> 124.88 1024.01 0.0 </l>
<l id="p0_h0" n="52"> 124.88 1023.99 0.0 </l>

These records show the expected pressure readings (1024.01 and 1023.99 mBar, see the <units> tag in Figure A3, line 155) for two (instance identifiers 51 and 52 respectively) identical (id = "p0_h0" in both cases) barometers, while their heights are 0.0 meters (the units are defined in line 166 of Figure A3). The value of the time tag (informational only) is 124.88 in both records. No covariance matrices have been included; should these be necessary (just for informational purposes only) then default values for such matrices may be retrieved from the corresponding metadata (<c> tags in lines 156 and 167 in Figure A3).

Concerning mathematical models, there are no data for these; in fact, ASTROLABE models are mathematical equations implemented by TDSs. The only information related to mathematical models that is included in the data model is their metadata (see Section 3.5 and Figure A4).

3.4. Data: Observation Equations

As already discussed in Section 2 and shown in Figure 1, the observation equation relates all the intervening data entities needed to derive a new output state (models, observations, states and instruments).

The first attempt to define the data model for observation equations directly mirrors this definition:

observation equation: (model identifier, list of observation instance identifiers, list of state instance identifiers, list of instrument instance identifiers)

Note that the identifiers used to refer to states, observations and instruments are instance identifiers (see Section 3.1). This means that if data coming from different instances of identical sensors are available, it is possible to refer individually to each of these sensors in the observation equation.

For instance, assuming an observation equation relating one model to two observations and one state whose respective identifiers were compute_position (the model), GPS and IMU (the observations) and position (the state), the observation equation would roughly look like this:

(compute_position, GPS + IMU, position, -) (The dash at the end of the equation stands for "no instrument identifiers". This example equation includes no instruments.)

If two identical GPS receivers were used to collect data, using directly the metadata identifiers to characterize the observation equation records would not leave room for multiplicity. On the contrary, the use of instance identifiers solves this problem. Assuming that the instance identifiers are 20 and 21 (GPS receivers 1 and 2) and 30 (IMU) and 10 (state), the following observation equations would involve, respectively, the first and second GPS receiver:

(compute_position, 20 + 30, 10, -)
(compute_position, 21 + 30, 10, -)

As seen in the example above, the cardinality of the different lists of instance identifiers may be zero in some cases. The model identifier must, however, be always present.

Extra items must be added, as it happens to observation, states and instruments themselves. An event tag, an activation flag and a time stamp must be added to complete the data model for this entity. Their purpose is described in Section 3.2—although that section talks about observations, the explanations given there may be applied here so these are not repeated again. In this context, however, the activation flag may be interpreted as a quick and practical way to avoid the (erroneous) computation of states when a significant number of the observations involved are clearly wrong.

observation equation: (**event tag**, **activation flag**, **time**, model identifier, list of observation instance identifiers, list of state instance identifiers, list of instrument instance identifiers)

Table 2 briefly describes all the elements integrating an observation equation.

Table 2. The observation equation entity data model.

Item	Description
Event tag	Identifies the kind of data entity (observation equation).
Activation flag	Enables or disables the observation equation for processing purposes.
Time	Time stamp for the equation.
Model identifier	Points to the model to use to deliver new states.
Observation instance ids.	List of observations involved in the computation of new states.
State instance ids.	List of states involved in the computation of new states.
Instrument instance ids.	List of instruments involved in the computation of new states.

The explicit inclusion of the model identifier opens the way to the use of different kinds of models in structurally identical observation equations. Revisiting the example above, the model used to derive the new state may be changed while involving exactly the same lists of observations and states:

$$\begin{gathered}(\text{compute\_position\_}\mathbf{method\_1}, 20 + 30, 10, \text{-}) \\ (\text{compute\_position\_}\mathbf{method\_2}, 20 + 30, 10, \text{-}) \\ \ldots \\ (\text{compute\_position\_}\mathbf{method\_x}, 20 + 30, 10, \text{-})\end{gathered}$$

One positive consequence of this feature is that researchers may tests new algorithms simply changing the model identifier used to derive new states in observation equations.

Figure 5 includes three examples of observation equations in ASTROLABE XML syntax. All these may be easily identified by the opening tag <o> which represents the observation equation event tag. The activation flag is materialized by means of the "s" attribute (whose values, "a" and "r" stand for "active" and "inactive" or "removed" respectively). Note that the equation at line 3 has no explicit activation tag: it is assumed active by default.

The observation equation identifiers (attribute "id") point to the respective models involved in the equation, namely "pva1_d", "imu1_bias_d" and "height_update". After these, the time tag (124.88 in all cases) and the lists of observation, state and instrument instance identifiers are included. The metadata for these equations may be found in the example in Figure A4 using the model identifiers (id) shown in Figure 5. (ASTROLABE metadata are discussed in Section 3.5). Such metadata state that, for example, model "height_update" points to *one* observation whose identifier is "baro1", *two* states ("baro1" and "pva1") and just *one* instrument, ("p0_h0"). This implies that a total of *four* instance identifiers must be present in the observation equation, as shown in line 3 of Figure 5: instance identifier 27 for the observation, 34 and 32 for the states and 51 for the instrument.

See Figure A1 for a complete ASTROLABE XML example showing both observations and observation equations.

```
1 <o s="a" id="pva1_d"       > 124.88 27 11 41    </o>
2 <o s="r" id="imu1_bias_d"  > 124.88 11    17    </o>
3 <o       id="height_update"> 124.88 27 34 32 51 </o>
```

Figure 5. Data: examples of observation equations in XML syntax.

3.5. Metadata

Sections 3.2–3.4 describe the structure of the different data entities involved in ASTROLABE's data model. For this definition to be rigorous, some additional aspects related to these entities must be specified. These aspects constitute the metadata.

Observations and states share the same metadata structure, that is, the same kind of facets are specified. For this reason, these will be described below within a common frame. Instruments and models, on the contrary, are a case apart, so their metadata will be specified separately.

The main items constituting the metadata for **observations and states** are the following:

- **Type and identifier.** See Section 3.1 for a detailed explanation on type and identifier codes.
- **Toolbox.** Name of the software module—typically, a Dynamic Link Library (DLL, Windows environment) or Shared Library (Linux environment)—including an implementation of the logic related to the observation or state. The way this name is used by the underlying operating system to find the library is not defined by ASTROLABE.
- **Dimension.** Number of elements in the observation/states expectations vector.
- **Referencing.** The reference frame plus coordinate system to which data refer to is specified here. Alternatively, it is possible to define a coordinate reference frame instead.
- **Units.** Specifies the units of the individual elements of the expectations vector for observations or states.
- **Covariance matrix.** Default covariance matrix for the expectations vector for observations and states. The units of the covariance matrix are the same as those provided for the expectations vector, even in the default case. ASTROLABE accepts "reduced" or "full" covariance matrices. A reduced covariance matrix consist of just standard deviations—assuming zeros for correlation values. This is an optional field.
- **Scale factors.** This optional field contains a list of positive scale factors for the standard deviations included in the covariance matrix. The number of elements in this list equals the dimension of the expectations vector. Scale factors values of 1 are assumed when this field is not present.
- **Tags (auxiliary values) characterization.** Since observations or states may be optionally accompanied by tags, (Section 3.2), it is necessary to characterize these. The way to do it is to define:
  - The number or tags related to the observation or state (**dimension.**)
  - For each of these tags, provide the **referencing** data (again, reference frame plus coordinate system or coordinate reference frame), as well as the **units** in use.

Figure A2 shows how three different observations are defined (lines 1–29, 30–53 and 54–77), using the ASTROLABE XML syntax. Note the XML tags used to describe the metadata elements: <l_spec> starts the definition of an observation; <type> is used to input the observation's type code. The identifier is specified by means of the <id> tag—which is embedded in a higher level structure named <lineage>. Toolbox codes are specified by means of the <toolbox> tag, while reference frames and coordinate systems are described by means of the (<ref>, <ref_frame_VC>, <coor_system_VC>) triplet. For units, the <units> tag is used, <c> is for covariance matrices and <s> is for their scale factors. The tags are characterized by means of the <t_spec> XML tag, which in turn use other ones already defined.

Figure A3 shows the specification of three states (lines 78–94, 95–119 and 120–137). (This figure also depicts the specification of an instrument, and it will be discussed below.) Note that, as stated before, observation and states are equivalent from the structural standpoint, so the XML tags used to describe these are almost the same. The only difference is the XML tag <p_spec> (opening the definition of a state).

Instruments are characterized as follows:

- **Type, identifier, toolbox.** Same meanings as for observation/states metadata. See Section 3.1 for details on type and identifier codes.
- **List of instrument constants.** Includes the number of constants defined (the **dimension**) and the description of each of these, including:

- **Type.** Either scalar or matrix.
- **Referencing.** Again, either the combination of a reference frame plus a coordinate system or a single reference coordinate frame may be used.
- **Units.** Specifies the units of the instrument constant.
- **Covariance.** Covariance matrix for the instrument constant. Note that covariances in instrument constants are merely indication of the quality of these. This field is optional.
- **Scale factor.** Optional. Scale factor for the standard deviations of the covariance matrix. The value of the scale factor is assumed to be 1 when this field is not present.

Figure A3 includes the definition of an instrument (see lines 138–171). Most of the XML tags used in this example should now be obvious (as <id>, <toolbox>, <units>, <c> or <s>, already commented when describing observation and state metadata). Instruments are described by means of the <i_spec> tag. Once more, their types are input by means of the <type> tag. <c_list> is used to describe the list of constants; each constant is specified by means of the <item> tag, where units, covariances, scale factor and referencing information may be detailed.

The following are the most important fields constituting the metadata characterizing a **model**.

- **Type, identifier and toolbox.** Same meanings as for observation/state/instrument metadata. See Section 3.1 for details on type and identifier codes.
- **List of observations.** The set of identifiers of the observations involved in the model.
- **List of states.** The set of identifiers of the states involved in the model. The role played by each state is also specified. These roles may be either free (the TDS will be responsible for estimating its value) or constant (an input, immutable value will be provided for the state).
- **List of instruments.** The set of identifiers of the instruments involved in the model. This list is optional, since some models may work without the need of instrument constants.
- **List of sub-models.** A model may rely in other models to perform its task. This list includes their identifiers, which is optional.

In Figure A4 three models are defined (each one starting with an <m_spec> tag). The lists of observations and states are input by means of the <l_list> and <p_list> tags respectively. When present—these are optional—instrument and sub-model lists are specified by means of the <i_list> and <sub-m_list> tags. All lists are made of items (XML tag <item> in all cases) providing the identifiers (tag: <id>) of the involved observations, states, instruments or sub-models. State items also describe their role by means of the <role> tag.

It is worth mentioning that the identifiers of observations, states and parameters referenced in these lists correspond to those found in the examples included in Figures A2 and A3. These, together with Figure A4 make a complete example of ASTROLABE metadata.

Note that metadata for models describe the model structure and *not* the logic needed to derive new states out of the involved inputs—this is the task of TDSs themselves. The model type and identifier are the pointers which such software must use to ascertain both the specific mathematical model and the kind of data intervening in the computation of new states.

4. The ASTROLABE Specification

The data model described in Section 3 has been materialized in two different ways: the ASTROLABE file and network interface specifications. Sections 4.1 and 4.2 describe, respectively, the fundamental characteristics of these. The full ASTROLABE specification may be found in [38].

4.1. The File Interface

The ASTROLABE file specification completely implements the data/metadata model described in Sections 3.2–3.5 by means of disk files. These include text and binary formats.

Text files (either data or metadata ones) are written in XML. This language was selected by several reasons: it can be used to describe information accurately and unambiguously; it is based on a proven

standard and uses simple text files that make it durable; it is free of legal constraints or threats; XML may be manipulated programmatically and validated against grammar definition files (schema files). The early availability (as far as 1999) of an open source library for Java/C++ able to parse and validate XML files—Apache Xerces/Xerces-C, see [39]—as well as the involvement of some members of the team in other projects that already used XML-based languages [14], also helped to select XML for ASTROLABE text files.

Table 3 shows the most important files making the ASTROLABE file interface. Among these, the most representative are (1) the observation-events—for observation, state and instrument data—and (2) navigation metadata files, which are described in detail in Appendix A.

Table 3. Summary of the different types of files included in the file interface.

File Type	Description
Observation-events *	Observations (input) or states (output).
Instrument data *	Instruments constant data.
Correlation matrices *	Correlation matrices for (1) trajectory observations; (2) trajectory states and (3) trajectory residuals. One file per type of correlation matrix.
Navigation metadata	Metadata information for observations, states, instruments and models.
Directory	File listing all other files involved in the computation of the output trajectory, as the observation-events or metadata files. May be seen as a "project definition" file.
Process options	Optional, free format. This file contains the list of options controlling the behaviour of the TDS producing the output trajectory. Its goal is to keep a record about how the trajectory was generated.
Process log	Optional, free format. List of messages issued by the TDS when computing the trajectory. For tracking purposes.

* Contain only data, not headers. Use external ASTROLABE header files. See Appendix A.

4.2. The Network Interface

The second materialization of the ASTROLABE data model is the network interface, which relies on TCP/IP (Transmission Control Protocol/Internet Protocol) sockets [40]. Typically, it will be used in real-time environments to communicate two processes, one of them sending data (the producer) and the other one receiving them (the consumer).

The ASTROLABE network interface is far more reduced than its file counterpart. In fact, only the observation (state) and observation equations data entities have been materialized up to the moment. Therefore, two processes communicating via this interface must previously agree on all the facets characterizing data (that is, on their metadata), since there are no mechanisms to transmit (receive) these.

Only three messages exist to exchange observation-events data:

- Data:
  - Observation.
  - Observation equation.
- Commands:
  - End of transmission (end of data).

The structure of the two data messages (observation, observation equation) faithfully mirrors the data model defined in Sections 3.2 and 3.4, summarized also in Tables 1 and 2. Minor changes have been applied to make possible the transmission and reception of data, as the addition of the dimensions of some of the components of the observation or observation equation messages (for instance, to indicate how many elements are included in a list of instrument identifiers) so the right amount of data is transferred. Nonetheless, these changes are implementation-related and in no way change the rationale behind the data model, so these will not be described here for the sake of simplicity.

The unique command message, *End of transmission*, is used by a producer process to tell its consumer counterpart that the transmission of a dataset has finished. It is defined as follows:

```
end_of_transmission: (message tag)
```

The unique component of this command is a tag to distinguish this message from others being transmitted between the two endpoints. This tag is the equivalent of the event tag described in Sections 3.2–3.4.

Since the current version of the network interface only allows sending an unique dataset (observation-events data), this command would not be necessary; closing the socket connection would have been enough to signal the end of the transmission process. However, it is foreseen that additional kinds of datasets (as instruments or output parameters) will be transmitted by future versions of the network interface. The availability of an *End of transmission* command will then allow the transmission of several datasets using the same *open* socket connection; there will be no need to close and open connections repeatdly to send different datasets. Sending *End of transmission* messages between these will be enough.

5. The ASTROLABE C++ Library

The CTTC has implemented a portable object oriented C++ library including reader and writer (file interface) and sender/receiver (network interface) classes. It offers all the necessary tools to process ASTROLABE data and metadata.

Specific classes have been provided to manage each available data format (as for instance, text or binary data files, see Section 4.1), so a client software module may use the specific implementation for each of these formats if desired. However, the use of external ASTROLABE header files describing how and where data are to be found, opened the way to a couple of generic classes (a reader or receiver, a writer or sender) able to manage all the available formats in a completely transparent way.

For instance, the generic reader class uses the information found in the ASTROLABE header file to ascertain how and where actual data are stored. Once this is determined, it instantiates (transparently) the appropriate specific reader class. The advantage of using such generic reader lies in the fact that client modules do not need to be aware of the details concerning how data are stored; thus, code is simpler and valid for any available format.

Another feature only available in the file interface of the ASTROLABE C++ library is the ability to read files either in forward or backward directions. This feature has been included since TDSs working in post-processing mode usually generate trajectories in three steps: (1) computing the output in forward direction (that is, from its beginning to its end); then (2) in backwards direction and; finally (3) filtering the two previous results to obtain the final output. Processing backwards implies the need to read data in backward direction, so adding such kind of feature to the library facilitates the development of client software modules.

Performance is also an issue that has been addressed in this library. All readers and writers use a technique known as "buffered reading (writing)" to speed up the process. It is worth to mention that NAVEGA doubled its performance when the ASTROLABE library was used to substitute the former readers and writers that did not implement this technique—no other changes were made to the tool. This is a noticeable improvement, especially when post-processing long datasets covering several hours of data that might take up to an hour or more of elapsed time to process.

The network interface always transmits binary data for efficiency reasons. However, the representation (and interpretation) of binary information may differ depending on the architecture of the processor managing it, more precisely when data are transferred from one computer to another via a network connection. To avoid such problems, the ASTROLABE library encodes and decodes all data using the XDR (eXternal Data Representation) standard [41] thus overcoming this issue. This encoding/decoding processing is performed transparently.

6. Use Cases. Sensors Supported. Sensors Not Supported

The ASTROLABE data model has been put to the test in real use cases along the years, facing different situations involving several projects as well as a variety of sensors. Such exposition to real-life problems made the original data model evolve to what it is now. Obviously, not all the concepts described in this paper were implemented from the very beginning, as for instance, the proper handling of metadata.

Up to the moment, the tandem NAVEGA/ASTROLABE has been able to deal with several kinds of sensors and their related observations. Table 4 summarizes a subset of the projects where NAVEGA has been used, including the sensors and related observation types that were involved in these projects. The interested reader will find more details about one of these projects, GAL, in Appendix B.

In general, ASTROLABE will be able to incorporate any kind of sensor providing observations whose error distributions are fully defined by its first and second moments (see Section 2). This is so because the structure (model) of an ASTROLABE observation (see Section 3.2) includes an expectations array plus a covariance matrix, which is all that an observation with such kind of error distribution needs to be properly modeled.

Table 4. Sensors already tested in ASTROLABE.

Project	Purpose	Sensors (Observations)
ENCORE (see [42,43])	Precise positioning for surveying applications using multi-frequency GNSS data.	GNSS receiver (GNSS raw data).
IADIRA (see [44])	Validation of the close + tight coupling concept.	IMU (accelerations, angular velocities). GNSS receiver (GNSS raw data).
CLOSE-SEARCH (see [45])	Robust navigation for UAVs (real time position and attitude computation using close-coupling).	IMU (accelerations, angular velocities). GNSS receiver (GNSS raw data). EGNOS signals (EGNOS corrections). Magnetometer (Magnetic field). Barometer (pressure).
GAL (see [46])	Airborne gravimetry.	Multiple IMUs (accelerations, angular velocities). GNSS receiver (x, y, z).
ATENEA (see [47])	Mobile mapping for urban cartography.	IMU (accelerations, angular velocities.) GNSS receiver (x, y, z). Odometer (time tagged distances, angular velocities). Optical camera (coordinates of tie points). LiDAR (ranges, angles).

This is true at least from the data standpoint, that is, ASTROLABE's. Obviously, and assuming that it is possible to model some kind of observation using ASTROLABE's approach, it will also be necessary to devise the proper mathematical model(s) for such observation to be useful from the TDS software's point of view. In short, it is necessary to provide with the appropriate mathematical machinery to transform the input observation into output states. This said, it must be clarified that this is not a problem of the ASTROLABE data model at all, since it takes care of data, not algorithms.

The other side of the coin is that observations with error distributions that cannot be fully defined by its first two moments may not be modeled using ASTROLABE. For example, map-matching contexts are usually affected by this kind of problems. See for instance [48] for a description of a situation where the constraints set by the environment (walls) make observations exhibit other types of error distributions.

7. Conclusions and Outlook

The ASTROLABE data model, file and network specifications and successive implementations of the C++ library have been put to the test for several years now. This includes a variety of European projects where different kinds of sensors had to be incorporated (see Table 4 for details). Being conceived as a generic and extensible system, the addition of such new sensors posed no unsurmountable problems for the devised abstraction. Needless to say, this continuous exposition to real-life conditions served to improve the data model progressively. It is possible to state, that ASTROLABE has been, at least up to the moment, able to manage change, evolution and innovation in the field it is specifically targeted at: data representation in the context of TDSs dealing with observations whose error distributions are fully charaterized by the first and second moments. This is of special importance, since the decision of developing a specific data model instead of using existing, mature ones as those described in [32,33] supposed a non-negligible risk.

The ASTROLABE data model and its two specifications (file and network) opened the path to the development of a C++ library exposing a very terse API and a high level of abstraction. Third party software modules accessing ASTROLABE data by means of this library may use very high level code that is independent on details like how and where data are stored (or transmitted).

The development of the ASTROLABE data model, specifications and library went hand in hand with the implementation of CTTC's TDS, NAVEGA. NAVEGA was designed using the same principles on which ASTROLABE rely, that is, it is a generic and extensible software tool mirroring internally the concepts present in the data model. This correspondence notably eased its development and, as expected, reduced to a minimum the maintenance tasks required to cope with change. Other TDSs may also benefit from the characteristics of the ASTROLABE specification by incorporating the library and its features related to data input/output and metadata management.

ASTROLABE is not yet complete. Although both the specification and the C++ library are very close to their final versions, some work remains to be done; for instance, no proper metadata to define the coordinate reference frame for time stamps exist yet. The foreseen work for the coming months will serve to address these minor issues.

The CTTC is considering putting the ASTROLABE specification as well as the portable C++ library in the public domain as soon as the work on these is finished. For more details, contact the authors.

Acknowledgments: The research reported in this paper started around eight years ago at the former Institute of Geomatics and has been funded by means of several European projects. We would like to highlight IADIRA, ATENEA, ENCORE, CLOSE-SEARCH and GAL. The authors would also like to thank Deimos for their support. Last but not least, the authors would like to thank Eduard Angelats and Pere Molina for their time, patience and helpful comments.

Author Contributions: The data model described in this work was devised mainly by M. Eulàlia Parés and Ismael Colomina. José A. Navarro contributed to some parts of the data model, implemented the ASTROLABE C++ library and wrote the majority of this paper. Finally, Marta Bláquez contributed to set the foundations of the data model abstraction.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. The File Interface: The Observation-Events and Navigation Metadata Files

This appendix relies on a full example to describe the observation-events and navigation metadata files, the pillars of the ASTROLABE file interface.

The **observation-events** (*obs-e*) file is used to store observations (when used as input) or trajectories (states, on output) since these two data entities are structurally equivalent.

When used as an input file, it contain a series of time-sorted observations and observation equations records—as defined in Sections 3.2 and 3.3. When it represents an output trajectory, only states—also time-sorted—are included.

Figure A1 depicts a complete (although short) observations-event file in text (XML) format used to store observations and observation equations.

```
01     <l id="baro1"        n="32"> 124.88 23.44 1023.44
02                                              0.3                                  </l>
03     <l id="imu1"         n="41"> 124.88       0.01 0.02 0.015 0.32 0.43 9.95
04                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
05     <l id="imu1_bias_pn" n="17"> 124.88       0.0  0.0  0.0   0.0  0.0  0.0  </l>
06     <o id="pva1_d"             > 124.88       27   11   41                   </o>
07     <o id="imu1_bias_d"        > 124.88       11        17                   </o>
08     <o id="height_update"      > 124.88       27   34   32    51             </o>
09
10     <l id="baro1"        n="33"> 124.90 23.45 1023.45
11                                              0.3                                  </l>
12     <l id="imu1"         n="41"> 124.90       0.01 0.01 0.014 0.33 0.44 9.95
13                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
14     <l id="imu1_bias_pn" n="17"> 124.90       0.0  0.0  0.0   0.0  0.0  0.0  </l>
15     <o id="pva1_d"             > 124.90       27   11   41                   </o>
16     <o id="imu1_bias_d"        > 124.90       11        17                   </o>
17     <o id="height_update"      > 124.90       27   35   33    52             </o>
18
19     <l id="baro1"        n="32"> 124.92 23.45 1023.45
20                                              0.3                                  </l>
21     <l id="imu1"         n="41"> 124.92       0.01 0.01 0.013 0.30 0.42 9.95
22                                              1e-3 1e-3 1e-3  1e-2 1e-2 1e-2 </l>
23     <l id="imu1_bias_pn" n="17"> 124.92       0.0  0.0  0.0   0.0  0.0  0.0  </l>
24     <o id="pva1_d"             > 124.92       27   11   41                   </o>
25     <o id="imu1_bias_d"        > 124.92       11        17                   </o>
26     <o id="height_update"      > 124.92       27   34   32    51             </o>
```

Figure A1. An observations-event file in text (XML) format.

In this example all the elements described in Sections 3.2 and 3.3 are shown. In every data record, the opening XML tag (either of the lowercase letters "l" or "o") represents the record event tag used to tell apart observations (tag "<l>") from observation equations (tag: "<o>"). Identifiers are input by means of the attribute "id"; the activation flags appear as the attribute "s". The instance identifier (in the case of observations only) is written using the "n" attribute.

The remaining values, located between <l> or <o> opening and closing tags, correspond to the rest of items in the record. To ascertain their meaning, the related metadata are required. These are the time stamp, the tags (auxiliary values, if any), as well as the expectation vector and its related covariance matrix—in the case of observations—or the list of instance identifiers needed by observation equations.

The metadata defining the observation and observation equations included in this example may be found in Figures A2–A4. Line number 03 of Figure A1 contains an observation (<l> tag) data record whose identifier (<id>) is "imu1". The example in Figure A2 defines the metadata for this observation in lines 030–053. More specifically, line 033 includes the identifier sought, "imu1". The dimension of the observation array is six (line 037 in the metadata) so it is correct to have exactly six values (0.01 0.02 0.015 0.32 0.43 9.95) in the observation data record after the first one (124.88) which is the time stamp. Although metadata provide a default covariance matrix in lines 050–051 (in fact, only standard deviations in this case), the data record includes its own values for this matrix in line 04. Lines 047–048 in the metadata file also state that the units for the measurements are rad/s, rad/s, rad/s, m/s^2, m/s^2 and m/s^2.

```
001 <l_spec s="a">
002   <type> baro </type>
003   <lineage>
004     <id>   baro1 </id>
005     <name> Baro measurements </name>
006   </lineage>
007   <toolbox> BAROMETER </toolbox>
008   <dimension> 1 </dimension>
009   <ref>
010     <ref_frame_VC>
011       QNH
012     </ref_frame_VC>
013     <coor_system_VC>
014       cartesian
015     </coor_system_VC>
016   </ref>
017   <units> hPa </units>
018   <c> 1 </c>
019   <s> 1 </s>
020   <t_spec>
021     <dimension> 1 </dimension>
022     <ref>
023       <coor_ref_frame_VC>
024         Celsius
025       </coor_ref_frame_VC>
026     </ref>
027     <units> °C </units>
028   </t_spec>
029 </l_spec>
```

```
030 <l_spec s="a">
031   <type> imu </type>
032   <lineage>
033     <id>   imu1 </id>
034     <name> IMU measurements </name>
035   </lineage>
036   <toolbox> INS </toolbox>
037   <dimension> 6 </dimension>
038   <ref>
039     <ref_frame_VC>
040       Bfrd
041     </ref_frame_VC>
042     <coor_system_VC>
043       cartesian
044     </coor_system_VC>
045   </ref>
046   <units>
047     rad/s, rad/s, rad/s,
048     m/s^2, m/s^2, m/s^2
049   </units>
050   <c> 5e-2 5e-2 5e-2
051       1e-1 1e-1 1e-1 </c>
052   <s> 1 </s>
053 </l_spec>
```

```
054 <l_spec s="a">
055   <type> RW_6_PN </type>
056   <lineage>
057     <id>   imu1_bias_pn </id>
058     <name> IMU measurements </name>
059   </lineage>
060   <toolbox> INS </toolbox>
061   <dimension> 6 </dimension>
062   <ref>
063     <ref_frame_VC>
064       Bfrd
065     </ref_frame_VC>
066     <coor_system_VC>
067       cartesian
068     </coor_system_VC>
069   </ref>
070   <units>
071     rad/s, rad/s, rad/s,
072     m/s^2, m/s^2, m/s^2
073   </units>
074   <c> 5e-2 5e-2 5e-2
075       1e-1 1e-1 1e-1 </c>
076   <s> 1 </s>
077 </l_spec>
```

Figure A2. Metadata: defining observations.

```
<p_spec s="a">
 <type> IMU_bias </type>
 <lineage>
   <id> imu1_bias</id>
   <name> Calibration biases (IMU) </name>
 </lineage>
 <toolbox> INS </toolbox>
 <dimension> 6 </dimension>
 <ref>
   <ref_frame_VC> Bfrd </ref_frame_VC>
   <coor_system_VC>
     cartesian
   </coor_system_VC>
 </ref>
 <units> rad/s, rad/s, rad/s,
         m/s^2, m/s^2, m/s^2 </units>
</p_spec>
```

```
<p_spec s="a">
  <type> baro_cal </type>
  <lineage>
    <id> baro1_cal </id>
    <name>
      Calibration values (barometer)
    </name>
  </lineage>
  <toolbox> BAROMETER </toolbox>
  <dimension> 1 </dimension>
  <ref>
    <ref_frame_VC> QNH </ref_frame_VC>
    <coor_system_VC>
      cartesian
    </coor_system_VC>
  </ref>
 <units> hPa </units>
</p_spec>
```

```
<p_spec s="a">
  <type> pva </type>
  <lineage>
    <id> pva1 </id>
    <name>
      Position, velocity and
      attitude in WGS84
    </name>
  </lineage>
  <toolbox> INS </toolbox>
  <dimension> 9 </dimension>
  <ref>
    <ref_frame_VC> WGS84 </ref_frame_VC>
    <coor_system_VC>
      geodetic, geodetic, geodetic,
      Lned, Lned, Lned,
      Bfrd-Lned, Bfrd-Lned, Bfrd-Lned
    </coor_system_VC>
  </ref>
  <units>
    rad, rad, m,
    m/s, m/s, m/s,
    rad, rad, rad
  </units>
</p_spec>
```

```
<i_spec s="a">
  <type> baro_p0_h0 </type>
  <lineage>
    <id> p0_h0 </id>
    <name>Initial pressure+altitude</name>
  </lineage>
  <toolbox> BAROMETER </toolbox>
  <c_list>
    <dimension> 2 </dimension>
    <item n="1">
      <type> scalar </type>
      <ref>
        <ref_frame_VC> QNH </ref_frame_VC>
        <coor_system_VC>
          pressure
        </coor_system_VC>
      </ref>
      <units> mBar </units>
      <c> 1.2 </c>
      <s> 1 </s>
    </item>
    <item n="2">
      <type> scalar </type>
      <ref>
        <coor_ref_frame_VC>
          WGS84-ellipsoidal-height
        </coor_ref_frame_VC>
      </ref>
      <units> m </units>
      <c> 0.15 </c>
      <s> 1 </s>
    </item>
  </c_list>
</i_spec>
```

Figure A3. Metadata: defining states and instruments.

```
<m_spec s="a">
  <type> local_mec_eq_bias_d </type>
  <lineage>
    <id>   pva1_d  </id>
    <name> Mechanization equations </name>
  </lineage>
  <toolbox> INS </toolbox>
  <l_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> imu1 </id>
    </item>
  </l_list>
  <p_list>
    <dimension> 2 </dimension>
    <item n="1">
      <id> pva1 </id>
      <role> free </role>
    </item>
    <item n="2">
      <id> imu1_bias </id>
      <role> free </role>
    </item>
  </p_list>
  <sub-m_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> WGS84 </id>
    </item>
  </sub-m_list>
</m_spec>
```

```
<m_spec s="a">
  <type> RW_6_d </type>
  <lineage>
    <id> imu1_bias_d </id>
    <name>IMU biases as random walk</name>
  </lineage>
  <toolbox> INS </toolbox>
  <l_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> imu1_bias_pn </id>
    </item>
  </l_list>
  <p_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> imu1_bias </id>
      <role> free </role>
    </item>
  </p_list>
</m_spec>
```

```
<m_spec s="a">
  <type> PVA_baro_U </type>
  <lineage>
    <id> height_update  </id>
    <name> Update of height </name>
  </lineage>
  <toolbox> BAROMETER </toolbox>
  <l_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> baro1 </id>
    </item>
  </l_list>
  <p_list>
    <dimension> 2 </dimension>
    <item n="1">
      <id> pva1 </id>
      <role> free </role>
    </item>
    <item n="2">
      <id> baro1_cal </id>
      <role> free </role>
    </item>
  </p_list>
  <i_list>
    <dimension> 1 </dimension>
    <item n="1">
      <id> p0_h0 </id>
    </item>
  </i_list>
</m_spec>
```

Figure A4. Metadata: defining models.

The observation equation (tag <o>) in line 06 of this example includes the model whose identifier (<id>) is "pva1_d". This model is described in the metadata shown in Figure A4, lines 172–202. The descriptive text in the <name> field states that this model implements the so-called "mechanization equations". Looking at the <dimension> tags in lines 180 and 186 it should be clear that this equation relates 1 observation (whose identifier is "imu1", line 182) and 2 states ("pva1" and "imu1_bias", lines 188 and 192). The list of sub-models (lines 196–201 in this metadata file) states than an additional sub-model will be required ("WGS84", line 199). Matching the list of instance identifiers included in the observation equation data record in Figure A1, line 06 and the aforementioned metadata leads to the following correspondences: 27 is the instance identifier of an observation whose identifier is "imu1"; 11 and 41 are instance identifiers of states whose identifiers are "pva1" and "imu1_bias" respectively.

Note that data coming from two identical barometers has been acquired (Figure A1, lines 01, 10 and 19). This is made evident by the different instance identifiers used in these three observations (attribute "n" is 32 in lines 01 and 19 but it is 33 in line 10).

As shown above, all the information in the data file is fully characterized by the metadata files. Cross-referencing data and metadata files is made by means of the identifiers (see Section 3.1).

An observation-events file used to represent **output trajectories** (series of states) adheres to the description above. The only difference with actual observation data files is that *no* observation equations are included—there is no need to compute new states since all these have already been obtained.

Finally, observation-events are pseudo-XML files, since no opening or closing XML headers or footers are present. Only <l> (and eventually <o>) records are included. Headers are stored in *external ASTROLABE header files*.

Figure A5 shows two examples of ASTROLABE header files; the first one points to the text observations-events data file just described, while the second one shows how an ASTROLABE data stream is characterized.

No matter what the case is, an ASTROLABE header file will always include the following items:

- The kind of data being stored or transmitted (as an observation-events file or an instruments file). This is specified by means of the tag "type"—see lines 08 and 20 in Figure A5.
- Where actual (observations or states) data are stored or transmitted. When using the file interface, data may be stored either in text or binary files. This is specified by means of the attribute "format" included in the header's <data> tag (line 08 in Figure A5). In the case of the network interface, data are sent or received through TCP/IP sockets (see Section 4.2 for details). Line 20 shows how the "format" tag changes to state that a "socket" is used.
- The parameters used to characterize the data channel. In the case of files, a file name is used (line 09 in Figure A5) while an optional server plus a port number is required in the case of network streams (line 21 in the same example).

There are two reasons to separate headers from actual data:

- Uniformity. All kinds of data materializations (binary or text files, network streams) are characterized in the same way through the use of a separate header. Data (in whatever form) always contain only <l> and <o> records, and no more.
- Data segmentation. Separating headers from actual data is useful if (data) files are to be split into chunks for security reasons; a unique header may be therefore attached to a *series* of successive data files representing a single data acquisition campaign. Should a failure in the acquisition system occur, only the last file would be lost, thus minimizing data loss. In fact, the ASTROLABE data model allows for file segmentation for this specific reason. Data segmentation implies, of course, a naming scheme for the different data chunks thus generated. ASTROLABE defines such naming convention, although it is not described here.

The selection of the different data formats available in ASTROLABE will obviously depend on the user's criteria and constraints. Text, for instance, is very useful in post-processing tasks, where humans

may edit easily readable files. Binary versions of these, on the contrary, reduce significantly the amount of storage needed to keep big amounts of data but are difficult to handle by human personnel. A well known example of the text/binary implementations tandem are RINEX [34] and BINEX [49] (BINary EXchange format) respectively. Sockets will be a must in the case of distributed processing. This is the reason why the ASTROLABE specifications provide with different specifications, trying to cover a wide range of situations.

The **navigation metadata files** are the mechanism to store all the metadata (see Section 3.5) related to the several data entities included in the ASTROLABE data model. These are always text, XML-based files.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03   version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
04   <lineage>
05     <id> july_campaign_01 </id>
06   </lineage>
07   <data>
08     <device type = "obs-e_file" format="text_file">
09       july_campaign_data.obs-e
10     </device>
11   </data>
12 </astrolabe-header_file>

13 <?xml version="1.0" encoding="UTF-8"?>
14 <astrolabe-header_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
15   version="1.0" xsi:noNamespaceSchemaLocation="astrolabe-header_file.xsd">
16   <lineage>
17     <id> july_campaign_01 </id>
18   </lineage>
19   <data>
20     <device type = "obs-e_file" format="socket">
21       localhost:2000
22     </device>
23   </data>
24 </astrolabe-header_file>
```

Figure A5. Examples of ASTROLABE XML header files.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <nav_metadata_file xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03                   xsi:noNamespaceSchemaLocation="nav_metadata.xsd">

                ...

** </nav_metadata_file>
```

Figure A6. Example of a navigation metadata file's header and footer.

Figure A6 depicts the header and footer in a navigation metadata file. It has been included for the sake of completeness. This figure is incomplete. To obtain a fully fledged navigation metadata file, the contents of Figures A2–A4 should be inserted (no matter the order) in the place of the ellipsis.

Navigation metadata files always include header and footer XML tags, as opposite to data files, that use the separate ASTROLABE header file.

Note that:

- In lines 02 and ** (the last one) the tag to denote navigation metadata files may be found: <nav_metadata_file>.

- In line 03 the name of the validating schema used to check the correctness of the file is specified: nav_metadata.xsd.

To finish, the whole set of metadata characterizing a dataset may be included either in a single navigation metadata file or to be spread across several ones.

Appendix B. ASTROLABE in the GAL Project

This appendix describes how the ASTROLABE data model was used in the GAL project, one of the real-life use cases mentioned in Section 6.

The goal of GAL, or *GAlileo for Gravity*, is the study and development of a state-of-the-art methodology for the determination of precise and high-resolution gravity field models. The key issue is the precise Kinematic Airborne Gravimetry (KAG), with the joint use of most recent techniques and technologies, such as GPS/Galileo and strapdown Inertial Measurement Units (IMUs), and its further integration with the global models from the gravimetric mission GOCE.

The main sensors used in the GAL data collection campaigns were one geodetic-grade GNSS receiver (Javad TR-G3T), one navigation-grade IMU (iMAR iNAV FJI) and one high tactical-grade IMU (IGI IIe). Table A1 depicts the main characteristics of the IMUs. The Javad TR-G3T is able to receive GPS (L1/L2/L2C/L5), GLONASS(L1/L2) and Galileo (E1/E5A) signals.

Table A1. Characteristics of the IMUs used in the GAL project.

Characteristic	iMAR iNAV FJI	IGI IIe
Gyro Drift/Offset	0.01°/hr	0.03°/hr
Gyro Random Walk/Q	$<0.001^\circ/\sqrt{h}$	$<0.005^\circ/\sqrt{h}$
Acc. Drift/Offset	<100 µg	<300 µg
Acc. Random Walk/Q	$<8\ \mu g/\sqrt{Hz}$	n/a
Gyro Range	$\pm 450^\circ$/s	$\pm 610^\circ$/s
Acc. Range	±5 g (option 2/g)	±20 g
Gyro Drift/Offset	0.01°/hr	0.03°/hr
Gyro Random Walk/Q	$<0.001^\circ/\sqrt{h}$	$<0.005^\circ/\sqrt{h}$
Gyro Scale factor	<30 ppm/<10 ppm	n/a
Gyro Axis Misalignment	<100 µrad	n/a
Acc. Drift/Offset	<100 µg	<300 µg
Acc. Random Walk/Q	$<8\ \mu g/\sqrt{Hz}$	n/a
Acc. Scale factor	<100 ppm/<20 $\mu g/g^2$	n/a
Acc. Axis Misalignment	<100 µrad	n/a
Data output rate	1...1000 Hz, BW 400 Hz	128, 256, 400 Hz
Size	370 × 213 × 179 mm	126 × 146 × 98 mm
Weight	10.2 kg, approx.	2.2 kg, approx.

Two aerial campaigns took place, the first in Lleida and the second over the catalan Pyrenees (Spain). 5.5 and 1.7 h of data were collected respectively. Figure A7 shows the flight plans for both campaigns. The selected areas had to fulfill a series of requirements, the most important being that quality gravimetric ground control points had to exist for verification purposes.

Both IMUs deliver the values of three linear accelerations (a_x, a_y, a_z) in m/s^2 and three angular velocities $(\omega_x, \omega_y, \omega_z)$ in rad/s, at 400 Hz. The coordinate reference frame is "body inertial", that is, all measurements are referred to the three IMU's internal x, y and z axes. The set of properties characterizing these IMUs is the same. Therefore, and since there are no differences from this standpoint, the two IMUs may be modeled using the same type (See Section 3.1 for information about types, identifiers and instance identifiers.) of observation. Figure A8 shows the metadata for the FJI and IIe IMUs. Note that the same type is used, but the identifiers differ.

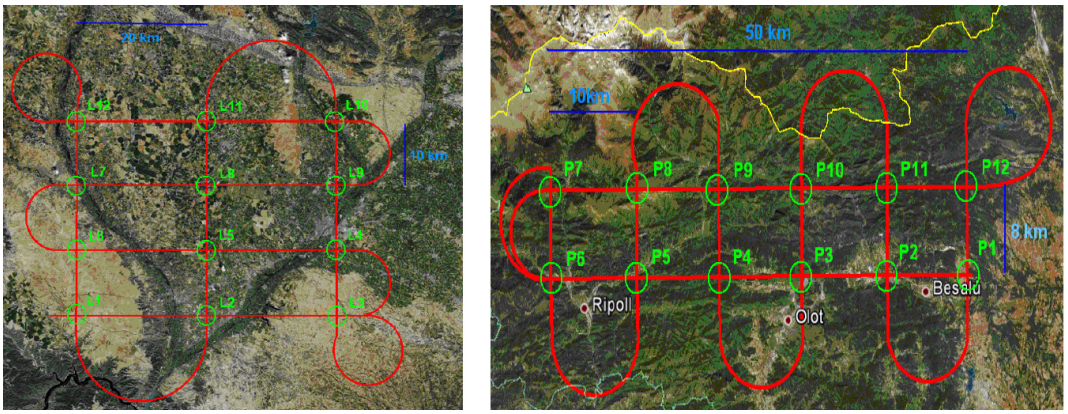

Figure A7. Fligh plans for the Lleida (**left**) and Pyrenees (**righ**) campaigns.

```
01 <l_spec>                                  18 <l_spec>
02   <type> IMU_a_w </type>                  19   <type> IMU_a_w </type>
03   <lineage>                               20   <lineage>
04     <id>   fji </id>                      21     <id>   IIe </id>
05     <name> IMU (a + w) -                  22     <name> IMU (a + w) -
06            iMAR iNAV FJI </name>          23            IGI IIe </name>
07   </lineage>                              24   </lineage>
08   <toolbox> INS </toolbox>                25   <toolbox> INS </toolbox>
09   <dimension> 6 </dimension>              26   <dimension> 6 </dimension>
10   <ref>                                   27   <ref>
11     <coor_ref_frame_VC>                   28     <coor_ref_frame_VC>
12       inertial                            39       inertial
13     </coor_ref_frame_VC>                  30     </coor_ref_frame_VC>
14   </ref>                                  31   </ref>
15   <units> m*s^-2, m*s^-2, m*s^-2,         32   <units> m*s^-2, m*s^-2, m*s^-2,
16           rad/s, rad/s, rad/s </units>    33           rad/s, rad/s, rad/s </units>
17 </l_spec>                                 34 </l_spec>
```

Figure A8. Metadata defining the IMU observations. **Left**: FJI; **right**: IIe.

The Javad GNSS receiver provides positions (x, y, z). Units are meters. The coordinate reference frame is ECEF (Earth-Centered-Earth-Fixed). Figure A9 shows the metadata defining the Javad observation.

The output trajectory is defined by a series of "position, velocity and attitude" (PVA) states, which consist of a position (longitude, latitude, height), a velocity (v_x, v_y, v_z) and three attitude angles (heading, pitch and roll). Such kind of state, with identifier `pva1`, has already been described in a former example (see Figure A3, lines 95–119).

Models are needed to transform observations into states. The most important ones used in the GAL project were (1) PVA prediction, hybridizing the observations originating from the two available IMUs and generating an output PVA state and (2) PVA update, using the GNSS receiver only and producing, again, an output PVA state.

The metadata defining the observation equations related to these two models may be found in Figure A10. Note how the references to the identifiers of the different observations and states involved in these equations clearly define what kind of data will be used or generated by the models. The first one, whose identifier is `pva_2IMU_1`, points to two IMU observations, namely `fji` (line 11) and `IIe` (line 14). The metadata for these two observations may be found in Figure A8. The second model, with identifier `pva_GNSS_1`, requires a single `javad` observation (line 35, Figure A10), which was defined

in Figure A9. Finally, both models point to the same output state, `pva1` (lines 20 and 41, Figure A10), defined in Figure A3, lines 95–119.

```
01 <l_spec>
02   <type> GNSS_xyz </type>
03   <lineage>
04     <id>   javad </id>
05     <name> GNSS receiver delivering
06            x, y and z </name>
07   </lineage>
08   <toolbox> INS </toolbox>
09   <dimension> 3 </dimension>
10   <ref>
11     <coor_ref_frame_VC>
12       ECEF
14     </coor_ref_frame_VC>
15   </ref>
16   <units> m, m, m </units>
17 </l_spec>
```

Figure A9. Metadata defining the Javad observations.

```
01 <m_spec>
02   <type> PVA_from_double_IMU </type>
03   <lineage>
04     <id> pva_2IMU_1 </id>
05     <name> Two IMUs to PVA </name>
06   </lineage>
07   <toolbox> INS </toolbox>
08   <l_list>
09     <dimension> 2 </dimension>
10     <item n="1">
11       <id> fji </id>
12     </item>
13     <item n="2">
14       <id> IIe </id>
15     </item>
16   </l_list>
17   <p_list>
18     <dimension> 1 </dimension>
19     <item n="1">
20       <id> pva1 </id>
21       <role> free </role>
22     </item>
23   </p_list>
24 </m_spec>
```

```
25 <m_spec>
26   <type> PVA_from_GNSS </type>
27   <lineage>
28     <id> pva_GNSS_1  </id>
29     <name> GNSS xyz to PVA </name>
30   </lineage>
31   <toolbox> INS </toolbox>
32   <l_list>
33     <dimension> 1 </dimension>
34     <item n="1">
35       <id> javad </id>
36     </item>
37   </l_list>
38   <p_list>
39     <dimension> 1 </dimension>
40     <item n="1">
41       <id> pva1 </id>
42       <role> free </role>
43     </item>
44   </p_list>
45 </m_spec>
```

Figure A10. Two models from the GAL project.

```
1 <l id="fji" n="1">   474631.000  0.048352247977806 -0.102871215632823   8.98256623212034
2                                 -0.024930613136148  0.050247523604562   0.006996777810447 </l>
3 <l id="lle" n="1">   474631.000 -0.23193359375     -0.18310546875     -10.6811523437
4                                  0.006484985351562  0.011825561523437 -0.010299682617187  </l>
5 <l id="javad" n="1"> 474631.000  4749387.498        228989.265         4237067.382        </l>
6 <o id=pva_2IMU_1>    474631.000 1 1 1 </o>
7 <o id=pva_GNSS_1>    474631.000 1 1   </o>
```

Figure A11. Actual data from the GAL project (a single epoch).

Having described the relevant metadata, it is possible to see—and understand—actual data from the GAL project. Figure A11 shows a single epoch (data sharing the same time tag) including an instance of every observation and observation equation discussed above.

Note the event tags "l" or "o" telling apart observations and equations, the different identifiers characterizing these (`fji`, `IIe`, `javad`, `pva_2IMU_1` and `pva_GNSS_1`), and the instance identifiers, which have the same value (1) in all cases. This is not a problem, though.

Since instance identifiers are used to tell apart observations *whose identifier is the same*, they do not need to be different *when identifiers already differ*. In fact, the full identification of observations in the data realm is achieved by means of the union of identifiers plus instance identifiers: `fji` + 1, `IIe` + 1 and `javad` + 1 for the three observations in the example. These unions are unique, so there is no ambiguity.

In the case of observation equations (Figure A11, lines 6–7), it is the position of the instance identifiers that eliminates the ambiguity. For instance, in the equation in line 6 (identifier `pva_2IMU_1`) the first instance identifier points to `fji` observations, the second to `IIe` ones and the third to `pva1` states (see the corresponding metadata for the observation equation in Figure A10). So, in spite of the numerical coincidence, there is no doubt about what these instance identifiers are pointing to (`fji` + 1, `IIe` + 1 and `pva1`+1).

The value of the time tag is 474631.000 in all cases. Lines 1–2 and 3–4 in Figure A11 depict observations from the FJI and IIe IMUs respectively. These are structurally equivalent (both include three linear accelerations and angular velocities). Even though the units used are the same, the corresponding acceleration or angular velocity values do not match, not even approximately. This is because the IMUs were not mounted in the same position and therefore the axes pointed to different directions. Line 5 in Figure A11 depicts a Javad (GNSS) observation, including its three position components (x, y, z).

The covariance matrices have been removed in all cases to simplify the example—these may be omitted in ASTROLABE data and specified in the metadata defining the observations, if desired.

Finally, lines 6 and 7 in Figure A11 show an example of the two observation equations corresponding to the two models. As already stated, the first one (`pva_2IMU_1`) is used to predict a new PVA state using one observation from each IMU, while the second one (`pva_GNSS_1`), updates the PVA state using data originating from the Javad GNSS receiver only.

References

1. Titterton, D.H.; Weston, J.L. *Strapdown Inertial Navigation Technology*, 2nd ed.; The American Institute of Aeronautics and Astronautics: Reston, VA, USA; The Institution of Electrical Engineers: Herts, UK, 2004.
2. Colomina, I.; Molina, P. Unmanned Aerial Systems for Photogrammetry and Remote Sensing: A review. *ISPRS J. Photogramm. Remote Sens.* **2014**, *92*, 79–97.
3. Karpowicz, J. *Above the Field with UAVs in Precision Agriculture*; Commercial UAV Expo: Las Vegas, NV, USA, 2016.
4. European GNSS Agency (GSA). *GNSS Market Report*; European GNSS Agency: Prague, Czech Republic, 2015.

5. Zhu, L.; Yang, A.; Wu, D.; Liu, L. Survey of Indoor Positioning Technologies and Systems. In *Life System Modeling and Simulation*; Springer: Heidelberg, Germany, 2014; pp. 400–409.
6. Minh, V.T. Trajectory Generation for autonomous vehicles. In *Mechatronics 2013: Recent Technological and Scientific Advances*; Springer: Berlin, Germany, 2014; pp. 615–626.
7. Mongrédien, C.; Hide, C.; Fairhurst, P.; Ammann, D. Centimeter positioning for UAVs and mass market applications: UAVs, precision agriculture and robotic guidance require high accuracy at low cost. *GPS World* **2016**, *25*, 24–33.
8. Groves, P.D.; Wang, L.; Walter, D.; Martin, H.; Voutsis, K. Toward a Unified PNT—Part 1, Complexity and Context: Key Challenges of Multisensor Positioning. *GPS World* **2014**, *10*, 18–49.
9. Groves, P.D.; Wang, L.; Walter, D.; Martin, H.; Voutsis, K. Toward a Unified PNT — Part 2, Ambiguity and Environmental Data: Two Further Key Challenges of Multisensor Positioning. *GPS World* **2014**, *25*, 18–35.
10. Beck, K.; Andres, C. *Extreme Programming Explained: Embrace Change*, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA, 2004.
11. Martin, R.C. *Agile Software Development: Principles, Patterns, and Practices*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2003.
12. Meyer, B. *Object-Oriented Software Construction*, 2nd ed.; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1997.
13. Colomina, I.; Navarro, J.A.; Térmens, A. GeoTeX: A general point determination system. In Proceedings of the XVIIth International Congress of the ISPRS, Washington, DC, USA, 2–14 August 1992.
14. Colomina, I.; Blázquez, M.; Navarro, J.A.; Sastre, J. The need and keys for a new generation network adjustment software. In Proceedings of the ISPRS Congress, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Melbourne, Australia, 25 August–1 September 2012.
15. Parés, M.E.; Colomina, I. On software Architecture Concepts for a Unified, Generic and Extensible Trajectory Determination System. In Proceedings of the ION GNSS, Tampa, FL, USA, 8–12 September 2015.
16. Soloviev, A.; Yang, C. Reconfigurable Integration Filter Engine (RIFE) for Plug-and-Play Navigation. In Proceedings of the ION GNSS+, Nashville, TN, USA, 16–20 September 2013.
17. Soloviev, A.; Veth, M.; Yang, C.; Miller, M. Plug and play sensor fusion for navigation in GNSS-challenged environments. In Proceedings of the ION GNSS+ 2016, Portland, OR, USA, 12–13 September 2016.
18. Förstner, W.; Wrobel, B.; Paderes, F.; Craig, R.; Fraser, C.; Dolloff, J. Analytical Photogrammetric Operations. In *Manual of Photogrammetry*, 5th ed.; McGlone, C., Ed.; American Society for Photogrammetry and Remote Sensing: Bethesta, MA, USA, 2004; pp. 887–943.
19. Tscherning, C.C. Defining the basic entities in a geodetic data base. *Bull. Géod.* **1978**, *52*, 85–92.
20. Elassal, A.A. Generalized adjustment by least squares (GALS). *Photogramm. Eng. Remote Sens.* **1983**, *49*, 201–206.
21. Sarjakoski, T. Object-oriented approaches in the design of more capable (adjustment) systems. In Proceedings of the XVIth International Congress of the ISPRS, Kyoto, Japan, 1–10 July 1988.
22. Crippa, B.; de Haan, A.A.; Mussio, L. The formal structure of geodetic and photogrammetric observations. In Proceedings of the Tutorial on Mathematical Aspects of Data Analysis, ISPRS, Intercomission WG III/VI, Pisa, Italy, 1–2 June 1989.
23. Colomina, I. Discrete mathematical techniques in the analysis and adjustment of hybrid networks. In Proceedings of the XVIIth International Congress of the ISPRS, Washington, DC, USA, 2–14 August 1992.
24. Kresse, W.; Fadaie, K. *ISO Standards for Geographic Information*; Springer: Berlin/Heidelberg, Germany, 2004.
25. Hasan, A.M.; Samsudin, K.; Ramli, A.R.; Azmir, R.S.; Ismaeel, S.A. A Review of Navigation Systems (Integration and Algorithms). *Aust. J. Basic Appl. Sci.* **2009**, *3*, 943–959.
26. Gade, K. NAVLAB, a Generic Simulation and Post-processing Tool for Navigation. *Model. Identif. Control* **2005**, *26*, 135–150.
27. Martin, M.K. New low cost avionics with INS/GPS for a variety of vehicles. *IEEE Aerosp. Electron. Syst. Mag.* **1998**, *13*, 41–46.
28. Hagen, O.K. TerrLab—A generic simulation and post-processing tool for terrain referenced navigation. In Proceedings of the OCEANS 2006, Shanghai, China, 18–21 September 2006.
29. O'Leary, P.; Eliser, J.; Turbe, M.; Betts, K. Implementation of an Open Architecture for Plug-and-Play Navigation Software. In Proceedings of the Joint Navigation Conference, Dayton, OH, USA, 7–9 June 2016.
30. Elsner, D. Universal Plug-N-Play Sensor Integration for Advanced Navigation. Ph.D. Dissertation, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA, 2012.

31. Ashkenazi, V. Coordinate Systems: How to Get Your Position Very Precise and Completely Wrong. *J. Navig.* **1986**, *39*, 269–278.
32. OGC®. 07-000—OpenGIS® Sensor Model Language (SensorML) Implementation Specification. Available online: http://portal.opengeospatial.org/files/?artifact_id=21273 (accessed on 28 October 2015).
33. OGC®. 10-004r3—Geographic Information—Observations and Measurements. Available online: http://portal.opengeospatial.org/files/?artifact_id=41579 (accessed on 28 October 2015).
34. RINEX. The Receiver Independent Exchange Format. Version 3.02. Available online: ftp://igs.org/pub/data/format/rinex302.pdf (accessed on 28 October 2015).
35. LAS Specification. Version 1.3 R11. Available online: http://www.asprs.org/a/society/committees/standards/LAS_1_3_r11.pdf (accessed on 28 October 2015).
36. Grewal, M.; Andrews, P. *Kalman Filtering. Theory and Practice*; Prentice Hall: Englewood Cliffs, NJ, USA, 1993.
37. Groves, P.D. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, 2nd ed.; Artech House: Norwood, MA, USA, 2013.
38. Parés, M.E.; Navarro, J.A.; Colomina, I. *ASTROLABE. Interface Control Document*; Internal Unpublished Report, CTTC / GeoNumerics S.L.; Unpublished work, 2016.
39. Apache Software Foundation. Apache Xerces. Available online: http://xerces.apache.org/ (accessed on 30 May 2016).
40. Stevens, W.R.; Fenner, B.; Rudoff, A.M. *UNIX Network Programming*, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA, 2004.
41. RFC 4506—XDR: External Data Representation Standard. Available online: https://tools.ietf.org/pdf/rfc4506.pdf (accessed on 28 October 2015).
42. Silva, P.; Silva, J.; Peres, T.; Colomina, I.; Miranda, C.; Parés, M.E.; Andreotti, M.; Hill, C.; Galera, J.; Camargo, P.; et al. ENCORE: Enhanced Galileo code receiver for surveying applications. In Proceedings of the 24th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2011), Portland, ME, USA, 2011; pp. 3679–3689.
43. Colomina, I.; Miranda, C.; Parés, M.E.; Andreotti, M.; Hill, C.; da Silva, P.F.; Silva, J.S.; Parés, T.; Galera, M.J.F.; Camargo, P.O.; et al. Galileo's surveying potential. *GPS World* **2012**, *3*, 18–33.
44. Silva, P.; Silva, J.; Caramagno, A.; Wis, M.; Parés, M.E.; Colomina, I.; Fernández, J.; Diez, J.; Gabaglio, V. IADIRA: Inertial aided deeply integrated receiver architecture. In Proceedings of the 19th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2006), Fort Worth, TX, USA, 26–29 September 2006; pp. 2686–2694.
45. Molina, P.; Colomina, I.; Vitoria, T.; Silva, P.F.; Skaloud, J.; Kornus, W.; Prades, R.; Aguilera, C. Searching lost people with UAVs: The system and results of the CLOSE-SEARCH project. In Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Melbourne, Australia, 25 August–1 September 2012; Volume XXXIX-B1, pp. 299–306.
46. Skaloud, J.; Colomina, I.; Parés, M.E.; Blázquez, M.; Silva, J.; Chersich, M. Progress in airborne gravimetry by combining strapdown inertial and new satellite observations via dynamic networks. In Proceedings of the IUGG 2015 Conference, Prague, Czech Republic, 22 June–2 July 2015.
47. Fernández, A.; Diez, J.; de Castro, D.; Dovis, F.; Silva, P.; Friess, P.; Wis, M.; Colomina, I.; Lindenberger, J.; Fernández, I. ATENEA: Advanced techniques for deeply integrated GNSS/INS/LiDAR navigation. In Proceedings of the 24th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS), Portland, OR, USA, 19–23 September 2011; pp. 2395–2405.
48. Isaacs, J.T.; Irish, A.T.; Quitin, F.; Madhow, U.; Hespanham, J.P. Bayesian localization and mapping using GNSS SNR measurements. In Proceedings of the IEEE/ION PLANS 2014, Monterey, CA, USA, 5–8 May 2014.
49. BINEX. Binary Exchange Format. Available online: http://binex.unavco.org/binex.html (accessed on 28 October 2015).