On software architecture concepts for a unified, generic and extensible trajectory determination system

M. Eulàlia Parés[1], Ismael Colomina[2],
[1]*CTTC, Spain*
[2]*GeoNumerics, Spain*

BIOGRAPHIES

M. Eulàlia Parés graduated in Mathematics at the University of Barcelona, holds an MSc in Meteorology and Climatology from the University of Barcelona and an MSc in Airborne Photogrammetry and Remote Sensing from the Institute of Geomatics. She is a researcher of the Geomatics division at the CTTC and a PhD candidate.

Dr. Ismael Colomina is GeoNumerics' director. He holds a PhD in Mathematics from the University of Barcelona. Dr. Colomina is a co-chair of the ICWG III/I "Sensor modelling for Integrated Navigation and Orientation" of the ISPRS and a member of the GPS World Editorial Board.

ABSTRACT

In this paper, we will describe the architecture of an innovative generic trajectory determination system. The architecture describes a software (SW) platform for the optimal determination of trajectories or paths of stochastic dynamical systems driven by observations –or measurements– and their associated dynamic or static models.

The proposed architecture has been already implemented in the NAVEGA SW. NAVEGA has evolved from an INS/GPS trajectory determination programme into the above more general concept to accommodate the various instrument and sensor configurations of modern navigation and orientation systems. Thus, NAVEGA can be "configured" for any sensor navigation combination, as, for example, navigation systems based only in GNSS, classical hybrid INS/GNSS systems, INS/GNSS systems augmented with other ancillary navigation sensors, INS/GNSS/visual aiding systems with multiple IMU or multiple GNSS receivers, to mention a few examples.

INTRODUCTION

During the last decade, with the popularization of smartphone devices, navigation and orientation have become an indispensable technology for most of the developed countries citizens. This way, the technology becomes not only a pre-requisite for enabling advanced, professional applications as those found in the remote sensing realm but also for the daily needs of every citizen. Navigation is applied in so disparate applications as pedestrian localization, airborne and marine bathymetry, railway track geometry surveying, motorcycle and car racing trajectory analysis, stabilization of moving video cameras in sport events, precise agriculture, archaeology, journalism and airborne photogrammetry and remote sensing. This heterogeneous demand translates in higher requirements to navigation software systems. As stated in [1] and [2], technology has to deal with new sensors –like plenoptic or photon-mixing cameras–, new performances –like the inertial sensors found in smartphones– and new environments –like indoor or urban canyons–.

An overview on current available solutions reveals that the market for traditional surveying applications is dominated by Applanix (Canada), OXTS (UK), IGI GmbH (Germany) and NovAtel (Canada). The first three companies provide integrated systems, including both HW and SW components that may not be acquired separately. The software components are, moreover, monolithic [real-time] navigators or [post-mission] orientators. NovAtel provides with integrated HW and SW solutions as well (the SPAN family), but also offers independent SW systems (GrafNav and Inertial Explorer). Yet, even in this case, these SW components are just navigators/orientators. The market segment covering new devices like smartphones, car driving assistance or pedestrian navigation is currently dominated by Garmin (Switzerland), Tom Tom (Netherlands), and Magellan (Japan). Once again, these are closed systems that do not allow the integration of new sensors. The consolidation and evolution of smartphones paves the way to a new navigation environment. Handheld devices provide the HW and the SW APIs while SW developers provide tools specifically developed for this kind of systems. The efforts to develop such a wide range of tools (from professional, to car-oriented or smartphones devices) may be measured in thousands of person/months for each company, and will keep being like that unless a common platform is developed for all of them. Beyond the traditional market, it is possible to find now open source

tools like RTKlib and GPStoolkit library, able to provide good solutions for a single specific sensor, the GNSS receiver.

In this paper we will describe the architecture of an innovative generic, modular and extensible trajectory determination system based on architectural principles successfully applied by the photogrammetric community [3],[4],[5]. These principles rely on the abstraction of the navigation algorithms in a way that including new sensors to the system implies only the development of a toolbox. The architecture proposed hereafter describe a SW platform for the optimal determination of trajectories or paths of stochastic dynamical systems driven by observations –or measurements– and their associated dynamic or static models. The reader will find firstly the requirements of the future Bayesian-based navigation systems, and a brief snapshot of the navigation algorithm structure. Later on, we present the principles that, from our point of view, should rule the next generation of navigation system architectures. Finally, we present a particular implementation of these architectural principles together with a short summary of its performance.

NEXT GENERATION TRAJECTORY DETERMINATION SYSTEMS REQUIREMENTS

Although the main requirements (quality and robustness) of navigation systems have not varied significantly since the appearance of the first tools, the technology available in the market having to comply with these requirements has. On the other hand, the availability of cheaper micro-processor and the miniaturization of technology introduce new requirements to the system (performance, integrability and extensibility.) Finally, the popularization of location-based applications has had also a great impact on navigation requirements evolution (usability):

Quality (precision/accuracy). These terms refer to the quality of the solution. Back in the early times of navigation systems, high quality solutions were expected – in direct correspondence to the high quality of the measurement systems used by those. Due to the irruption of COTS systems in recent years, the quality of the solutions must rely more on algorithms and less in the characteristics of sensors. Thus, more and more sophisticated inference algorithms are needed to obtain similar results with cheaper sensors.

Robustness. This requirement is about the system resilience to the presence of outliers, the lack of information or bad initialization. New users or non-specialized on the field expect for solutions valid in any environment, and will not be satisfied with system that behave incorrectly due to issues that they are not even aware of, such as multipath, magnetic fields or satellite occlusions. New systems should consider all this scenarios and their related problematics.

Performance. The owners of handheld devices demand navigation systems able to position them in the most singular places and always in real time. This implies to use a wide range of sensors to be able to provide a continuous solution but also algorithms with a low computational cost able to deliver results in short time. Even when dealing with the most sophisticated sensors, users expect to have a solution, computed by a micro-processor, in real time.

Integrability. New users also expect that location-based applications have access to location service providers. The new navigation systems should be easily accessible from external users in order to be easily integrated in non-geo applications. This translates in clear and manageable interfaces.

Extensibility. The last requirement, related to the continuous irruption of new technologies, refers to the capability of the system to integrate new sensors without increasing exponentially the development effort. Either for professional applications or for mass-market ones, more and/or new sensors are used every year. New navigation systems should be able to cope with that, focusing only on sensor modelling and not in the estimation algorithms.

Usability. The users of navigation system have evolved from airplane or boat crews and photogrammetry specialists to millions of citizens requesting for geo-located services in their daily routines. These new users have never received (and will never receive) training on navigation techniques, so the interfaces of navigation systems should be clear and intuitive. For professional users and researchers, interfaces should allow the modification of states controlling the behavior of the software as well as a deep interaction with the application. To sum up, the system should be able to deal with a wide range of users and, at the same time, guarantee the quality of the solution and its performance.

STATE-SPACE APPROACH FOR TRAJECTORY DETERMINATION

We will call state-space approach (SSA), the method and principles of solving the trajectory determination problem by Bayesian Filtering. The optimal Bayesian sequential structure is a recursive loop in time where prediction steps are followed by an updating step and so on. The prediction step aims at computing states expected values and errors trough the knowledge of the last state expected values and errors. Updating step aims at computing this information by joining the predicted states with external measurements. For linear problems and measurements/states Gaussian distributed, this recursive state algorithm is the standard Kalman filter [6]. For Gaussian distributed data but (nearly) non-lineal systems, almost optimal solutions are the Extended Kalman filter or the Sigma-Point Kalman Filters. For hardly non-linear

and non Gaussian systems, the almost optimal solutions are obtained through Particle Filters [7]. The architecture presented in this paper applies for any Bayesian estimation procedure; we do not focus in any particular implementation.

NEXT GENERATION SYSTEMS ARCHITECTURE PRICIPLES

In order to achieve the current requirements of extensibility and modularity we have revisited the previous algorithm, which has gone through a process of abstraction and generalization. As stated in [5], simple and extensible software design requires correct abstraction levels. Abstraction is the process of expressing a quality or characteristic apart from any specific object or instance. Thus, we have identified the navigation software characteristics apart from its actual implementations. According to [5] *"insufficient or needless abstraction leads to complex systems, wrong abstraction leads to non extensible systems but correct abstract models are, therefore, the key to simple and extensible systems."* Because of this abstraction exercise on navigation systems, we have arrived to some principles that will drive the definition of our new architecture:

Separation between estimation and modelling. Bayesian estimation processes ingredients are data (measurements/states), the relations between data and the specific algorithm used to estimate the states [8]. Current navigation systems implementations do not take into account this process conceptual segmentation, leading to very efficient but hardly reusable SW tools. We propose to define an architecture where **estimation** algorithms (procedure**)** and **modelling** (data and its relation) are implemented as separated components. The separation of the "numerical cruncher" and the equations allow a quick extension of the software when new sensors appear. Roughly speaking, "configuring" the system implies "loading" components related to the specific navigation sensors and setting a number of mission related states. Still roughly speaking, "configuring" the system for a specific instrument has no effect on the computational kernel even if new sensors come into play. The actual implementation of this concept could take benefit from dynamic libraries. While the common elements of the state-space approach should be implemented directly in the "number crunching" kernel, the uncommon ones should be materialized in dynamic libraries. An implementation like this one, has a lot of benefits such as minimizing the required amount of RAM in run time, since only the needed sensors are loaded; minimizing the development time, since the inclusion of new sensors imply the development of relatively small fragments of code and, finally, minimizing the time needed to train developers, since these should only have a deep knowledge about the sensor but not about Bayesian filters. Furthermore quality performance and robustness will not only not decrease but can even increase since more efforts can be placed on the development of the "number crunching" kernel.

Sensor/measurements abstract reference model. Focusing on the information related to the navigation system, we distinguish four elements: (input) measurements, (input) auxiliary instrument constant values, (output) states and (equations) models relating all those elements. Available technology allows us to have several sensors providing different kind of observations which purpose is to estimate same states type (e.g. magnetometers and star trackers both aims to determine orientation in space). Technology evolution, provide users with same principle sensors but a wide range of quality performances, the purpose of all this sensors, however, is to estimate same states type (e.g. navigation-grade to MEMS inertial sensors for position and orientation estimation). The modelling of the information component in four categories (observation/measurement, state, model and instrument) avoids code repetition and allows the use of object-oriented modelling, its inheritance, encapsulation and polymorphism mechanisms [4], [5]. A clear stateization of measurements, states, equations and auxiliary values elements is a powerful mechanism that allows the simplification of the system and consequently will help developers to implement quickly new sensor measurement and state models in the SW.

Computational strategy object. Navigation solution performance depends not only on the equipment but also on the environmental conditions [1][2]. Some years ago, this last factor was not taken into account in navigation systems, mainly focused on airborne environments. Nowadays, since the number of non-airborne users is growing exponentially, such systems should include a mechanism able to deal not only with sensors but also with scenarios, a context awareness *strategy object*. Users should be able to inform the system about its environmental context through an options file. In situations where the user is not able to provide with this kind of information, the strategy object should be capable to determine such context by itself (e.g. the qualitative analysis of the data provided by inertial sensors mounted in a car should allow the system to determine this environment by itself). Strategy should also be aware of the environment to be able to add to the system, if needed, contextual information (e.g. vertical velocity restrictions for terrestrial platforms or sudden direction changes restrictions in fixed-wing airplanes.) Strategy must also decide at any moment of the process if the system is able to provide the required solution out of the available information and, if it is not possible, decide how to proceed. If the object detects that the provided information is not enough to compute a solution, it must

be able to choose between delivering a partial solution or warning the user that the system is not able to provide an acceptable one and that more information is required. Once again, the software must reflect the conceptual difference of this object and the estimation and modelling objects; thus, the strategy object should not be integrated in the "numerical cruncher", but implemented as a separate component. An approach like this will allow the use of the same sensor models for several platforms and environments without having to implement them repeatedly. This implementation have multiple benefits like minimizing the system RAM requirements and simplifying the development process, since developers can tune its strategies independently of the estimation process.

Generic/adaptable user interface. The definition of the interface (be it file or network) of modern navigation systems shall be funded in the fact that the interface should not complicate the extensibility of the system. Therefore, the definition of such interfaces should take into account future extensions of the system. An incorrect interface definition leads to an under-utilization of the capabilities of the system and to an increase on development efforts. Currently, when integrating new sensors into a navigation system, not only the source code must be modified but also the user interface definition and layer. As the eyes are the mirror of the soul, the interface should be the mirror of the system. For this reason, we propose that the definition of the interface take into account all the abstraction processes presented in previous premises. Analogously to what has been discussed until this moment, we will separate the user interface –control of the estimation process— and the data interface –generic description of the rules to provide and read data from the system. [4]— Nowadays, trajectory SW can be used by such different users like geophysics, photogrammetrists, wedding recorders or pedestrian shoppers. We can classify trajectory determination SW users in three groups: lab, workshop, factory [9]. By lab users we refer to the ones that intend to develop new models for new sensors. They need to have access to all SW configuration states in order to find the best configuration to solve a specific problem. On the other side, a civilian user that intends to use the navigation system to go for a dinner in a restaurant is not worried about what are the sensors used by the navigation system or how these have to be calibrated. Between both users we can find the workshop ones, typically professional navigation system users that want to get the best available performance of the system. A successful SW should be useful for all of them. It must be able to deal with several dialects. This translates into the fact that system should be completely configurable (for lab users) but should allow external interfaces with restricted configuration access (for factory users). Concerning data interface, it should reflect the abstraction exercise performed in previous sections and its foundations should be the sensor, measurement, state and model concepts. With a definition like that, the long-term development costs will be reduced since the interface, once defined, will be suitable to incorporate any new sensor

Table 1 summarize the contribution of each principle to the fulfilment of the system requirements, through the traceability matrix confronting requirements and principles. Note that the performance requirement is not specifically considered by any principle. Since we are proposing an object-oriented implementation it could be stated that performance will suffer a heavy with respect to the use of sequential programming approaches. This is true but it can be demonstrated that the penalty to pay for the use of an object-oriented approach is almost negligible. Furthermore, the strength of new computer processors makes this overhead less significant when compared to the overall system performance.

	Estimation vs modelling	**Sensors abstract modelling**	**Adaptable user interface**	**Strategy object**
Precision/accuracy	**X**	**X**		**X**
Robustness	**X**			**X**
Performance				
Integrability	**X**	**X**	**X**	
Extensibility	**X**	**X**	**X**	**X**
Usability			**X**	**X**

Table 1: Requirements vs architecture principles traceability matrix

PROPOSED ARCHITECTURE DEFINITION

Based on the design principles presented in the above section hereafter we propose an architecture that complies with that design.

System components and activity diagram

Our system is based on four components (Figure 1): computation kernel (or "numerical cruncher"), trajectory (this component is the one includes modelling and supervisor), interfaces and a driver to control all the process. These components should be implemented using both static and dynamic libraries -when operative system allows it. Static libraries should be used to implement the procedure ("number crunching" kernel) while the modelling/strategy should be resort to dynamic libraries - thus, only the classes related to the requested sensors will be loaded in run time. To extend the system, incorporating new sensors, is as simple as writing new classes – that should be included in new dynamic libraries - adhering to the API defined by the system.

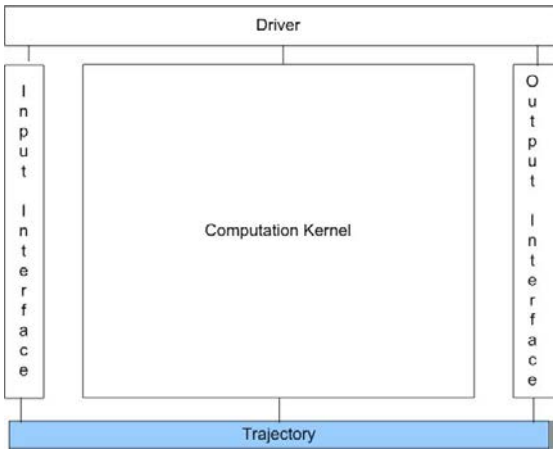

Figure 1: Architecture components diagram

The system inputs are not only sensor observations and generic context, as in common trajectory determination tools but also a list of requested states/states, a list of desired models/equations to compute these states and a specific computation context or strategy. All this information should be collected through the interface layer. The control component is in charge of loading all the requested objects (the aforementioned dynamic libraries.) The strategy object should analyse the feasibility of computing the target states from the set of given observations. If the computation is possible all the data should be sent to the estimation subcomponent to process it; otherwise a warning message should be sent to the user. There will be situations where only a subset of the desired states can be computed; should this happen, the strategy object should decide whether to compute what is possible to compute or to send a warning message. The output of the estimation subcomponent is delivered to the user through the interface component.

It is important to note here that the process is valid either for forward, backward and smoothing processes. Thus, a few (and simple) modification of both (1) the (unique) source code implementing this process and (2) the interface component, make possible to use the same algorithm for either real-time or post-processing work modes

Platform component

The platform component includes not only the driver subcomponent but also the two filtering steps subcomponents: the prediction component (e.g. stochastic dynamical systems solver for Kalman filter family estimators) and the update component (e.g. correction step for Kalman filter family estimators). The platform component is responsible for loading computation options and start the desired trajectory estimation processes (forward, backward and/or smoothing.) These processes mainly relies on the two numerical estimation subcomponents, prediction and correction.

The API must allow the inclusion of new numerical methods that may be of interest to the final user (extensibility criterion). In this way, both the prediction and the update components should be able to incorporate any related numerical method implementation. Through the interface layer, the user should be able to select among the available methods. Furthermore, it is important to remind that, as stated in previous sections, each of these implementation should be independent of the actual set of equations to solve; these should be designed to interface efficiently with the modelling objects (observations, equations, states and constant values) to retrieve values, to allocate and properly fill in vectors and matrices that will allow state estimation.

Modelling component

We propose a sensor/measurements model like the one presented in [5], where the fundamental reference model is based on four abstract categories (i.e. classes): instruments, observations, states and mathematical models. It is important to remark here that when we talk about observations and states we refer not to a single observation/state item but a group of them. Every new kind of sensor to be integrated in the system will probably collect measurements not available for other sensors. These measurements will be related to some unknown states, probably position or orientation. The relation between the observations and the states, maybe with the participation of the instrument auxiliary constants, is materialized by the mathematical models (stochastic equations). In order to extend the system, is necessary to identify the instrument and its characteristic values, to list its related measurements, to identify the states and the mathematical models that relate them. Once this is done, specific classes that inherit from the fundamental reference model can be reused or newly developed.

EXAMPLE – magnetometer modelling

- *Instrument.* The magnetic declination should be taken into account to correct the measurements. This value can be entered as a constant to the system through the instrument object. Also the boresight matrix between magnetometer and trajectory reference frames.
- *Observation.* Attitude values.
- *States.* Orientation values.
- *Model*. Rotation matrix that measurements of magnetometer and trajectory orientation. Please note that the model should take into account that trajectory and magnetometer reference frames could not be the same.

Strategy component
When designing the strategy component three issues arise: context awareness, context conditioning and computation feasibility.

The strategy component should be implemented, if possible, in classes stored in dynamic libraries. These classes are closely related to the sensors involved in a particular computation. All strategy specific implementation classes will be derived (inherit) from the same class, so all of them should have the interface. This component is aware of several strategy issues, highlighting context awareness determination, context conditioning and computation feasibility. In this way, several strategies (strategy classes) may be implemented and be able to deal, in different ways, with the same kind of data (e.g. IMU and GNSS), taking into account issues like the quality of the sensors, its performance or the environment.

Context awareness refers to the capability of the system to be aware of the environment and to add constrains to the system if required. The context awareness can be provided by the user or automatically detected by the SW itself. In the second case, a qualitative analysis of available data should be performed. Thus, for example if an IMU is involved in the computation and the signal output by the IMU matches (in a qualitative sense) the typical movement patterns related to the behaviour of human beings [10], a pedestrian environment can be assumed. This context awareness analysis should be done periodically to assure correct environmental assumptions during the processing of the whole trajectory.

Context conditioning refers to the capability of the system to include conditions to the system that depend on the environment. In our approach, instead of implementing condition equations, we add observation equations to the system. This reduces computation complexity and allow reusing all the implemented estimation algorithm without any additional changes. In order to do this, condition equations should be transformed to observation equations by assuming a reduced noise in the condition. The smaller the noise, the most restrictive the condition will be. For each environment, a set of additional condition equations are defined and added to the intrinsic sensor observation equations.

Last issue refers to *computation feasibility*. In order to decide if the system is able to provide a solution out of the available information we should check the degree of freedom of the problem and the update observations' matrix. If not enough observations are provided, the strategy object checks the suitability of performing a reduced adjustment with (pre-selected) fundamental states. If this is not possible, computation is finished and control is returned to the user. On the other hand, it could happen that there are enough measurements but the system is not robust enough to converge properly. This can be detected by means of covariance matrix analysis. Each strategy object, should include a set of relevant and non relevant states, with covariance thresholds to determine whether those states can be estimated or not.

EXAMPLE – Strategy for IMU/GNSS/MAGN

- *Context awareness.* IMU signal qualitative analysis. Fit with predefined patterns to deduce, pedestrian, terrestrial vehicle or fixed-wing airplane environment.
- *Context conditioning.* For pedestrian navigation include observation equations to limit maximum velocity. For terrestrial vehicles include observation equations to limit vertical and lateral velocities. For fixed-wing airplane include observation equations to avoid backward movement
- *Computation feasibility.* Initial state estimations will be derived from GNSS (position and velocity), and magnetometer (orientation). IMU calibration values will be derived form a combination of magnetometer data and IMU data. Position, velocity and attitude are the fundamental states; the system will not try to solve IMU calibration values until the standard deviations of the first ones are below predefined thresholds.

Interface component
The interface component based on three clearly distinguishable concepts: the Application Program Interface, the Data Interface and the User/Options Interface.

Application Program Interface design is totally driven by the system extensibility requirement. This translates into object oriented principles, inheritance capability and the use of dynamic libraries for the modelling and strategy components. Since the modelling and strategy component classes inherit from five basic classes (instrument, observation, state, model and strategy) the interfaces are the ones defined by these classes. New classes must completely adhere to this API. Regarding the platform interface with other systems, its design is as simple as possible. The tool should executed just with a "go" command and loading a configuration options file (explained at the end of this subsection).

Since the design principle of the *Data Interface* is based on is adaptability - to incorporate easily and painlessly new instruments, measurements, models or states - we have defined an interface (equivalent for network TCP/IP sockets and file interfaces) based on the generic description of observations and models. Each observation is described through a time tag, its value (expectation), its estimated error (standard deviation) and some auxiliary values that can be useful for computation purposes. When using a file interface – based on the XML standard - the

way to introduce these values in the SW is through the <l> tag shown below:

<l> obs_code obs_id time a1…an e1…en s1…sn </l>

where obs_code refers to the unique identifier for a specific observation type (for example, GNSS position) and obs_id refers to the sensor identifier. time refers to the acquired data time tag, {ai} refers to auxiliary values, {ei} refer to the acquired set of values and {si} refer to its standard deviation.

The observation equation to be used relates observations and states. Again, when using the XML file interface, the way to introduce these values in the SW is through a the <o> tag:

<o> model_code time par1..parn obs1…obsn</o>

where model_code refers to the unique identifier of the model class that relates a certain set of state types with observation types. Time refers to the time to perform the calculus, {pari} refers to the identifiers of the states to solve (the type of this state is known by the model class) and {obsi} refers to the identifiers of the observations to involve (again, the type of this observations is known by the model class). This data interface is common (uniform) for the two different data channels implemented by the system: (disk) files or TCP/IP sockets. This allows for remote *(in the cloud*) execution, being possible to capture data in whatever the place providing that a network connection is available. The advantages of this approach are manifold: perhaps the most important one is *ubiquity* (data may be captured everywhere); another advantage to consider is the reduction of the computational power needed by the processor in charge of computing the navigation solution, since it will not be responsible of managing the sensors acquiring data. For extended details on the interface definition, please contact the authors.

The user/options interface is designed as a two-layer interface. The lower one is based on files and interacts directly with the platform component. On top of this one, a user oriented interface (generally a command line or a graphical user interface) is developed. This second interface allows the SW dealer to modify determined configuration states depending on the user needs. Thus, for example, in a lab environment, the GUI should allow the configuration of all the computation states, while, in a factory environment, these options should be restricted to the minimum. In this way the software can be used in lab/research/factory environments by just changing the most external layer, the GUI.

Architecture strengths and weakness

Since the proposed architecture fully relies on the principles presented in previous sections, we can state that it satisfies the requirements of the new generation of navigation systems concerning quality assessment, robustness, performance, extensibility, integrability and usability. In the next section we present an implementation of this architecture were we have been able to demonstrate empirically that these requirements are fulfilled.

As any other approach, the one proposed in this paper has some drawbacks. In this case, at the implementation level. For example, since we want to accommodate an undetermined number of sensors in each execution we have to take into account that the amount of memory used will change from execution to execution, so dynamic memory allocation should be used. Moreover, we can not fine-tune the Kalman filter to improve matrix management, since we do not know their dimensions a priori. Thus, special attention on the computational burden of the algorithms implemented must be paid.

A SAMPLE IMPLEMENTATION – NAVEGA

We have implemented the aforementioned architecture in a SW system called NAVEGA. It is coded in C++ and provides real time and post-processing trajectory estimation under Windows and Linux operating system running in different processor architectures like x86, LEON3 and ARM. Currently the system includes a family of Kalman filter algorithms and provides with Gaussian states, this is the output are expected values and covariance matrices. Furthermore, it also provides the user with residuals expected and standard deviation values, suitable for a posteriori trajectory analysis.

Until this moment, NAVEGA has been proved to be **extensible** since it has already been used to process data provided by several types of *sensors* like IMUS [11], GNSS [12], redundant IMUS [11], LIDAR [13], camera images [14] and odometers [15]. Since not all the implemented sensor models are available in commercial systems we validate NAVEGA solutions using indirect methods. Using NAVEGA as positioning provider to remote sensing platforms (like photogrammetry systems); we compare the NAVEGA georeferenced data with topographic measurements. Up to now, these comparisons have shown the success of NAVEGA, that is, the predicted state error estimation corresponds to actual state error, or, in other words, the expected **quality** is achieved. NAVEGA has also been successfully used for processing data collected in a wide range of *environments*: airplanes, helicopters [11] and terrestrial vehicles, either outdoors and indoors [15].

Regading **robustness** issues, the implemented analysis tools has been proven able to deal with odometers and camera outliers. When working with GNSS modelling, the system is able to detect outliers when these affect just one satellite. At this moment, more than one outlier cannot be properly detected.

NAVEGA has also been proven to be **integrable** since it has been used in several projects as the location provider

of different services like a mobile mapping system [16] or a GNSS receiver with tight coupling capabilities. [17]

With the previous examples we have been able to verify that NAVEGA's **performance** is suitable for a wide range of platforms. NAVEGA is able to process INS/GNSS solutions at 5000 Hz in Real Time when running on an Odroid-XU3 with a negligible latency.

Last but not least, due to its file interface the system is completely **usable** either for Kalman filter experts to unexperienced users, through several GUIs.

CONCLUSIONS AND FURTHER RESEARCH

We have designed a trajectory determination modular, generic and extensible system requiring only dynamic libraries to be able to incorporate new sensors. We believe that this approach responds to some of the challenges posed by the design of modern system. The design starting point is the abstraction of the traditional state-space approach navigation algorithm in two main components: estimation and modelling. This concept translates into a SW component that allows for fast and incremental modelling of multi-sensor navigation systems. We expect that the current tool will benefit both navigation and geomatic communities, particularly the Earth Observation (EO), mapping and surveying teams that perform primary data acquisition in kinematic mode be it from airborne, terrestrial and marine manned and unmanned vehicles.

Although a lot of effort has been devoted in the definition of this architecture and its implementation, more efforts need to be devoted to the implementation of the strategy object. At this moment we have clearly defined the principles on which this design is based and have implemented the context adaptation concept, but we have not analysed yet which is the best implementation for that object in terms of performance. Our future research will also be oriented to implement the context auto-identification capability.

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, GAL and GINSEC. The authors would also like to thank Deimos for their support. Last but not least, the authors would like to thank Dr. Marta Blàzquez, Mr Pere Molina and Dr. José Antonio Navarro for their time, patience and helpful comments.

REFERENCES

[1] Groves P. D, Wang L., Walter D., Martin H., and Voutsis K. "Toward a Unified PNT — Part 1, Complexity and Context: Key Challenges of Multisensor Positioning." GPS World, October 2014.

[2] Groves P. D, Wang L., Walter D., Martin H., and Voutsis K."Toward a Unified PNT — Part 2, Ambiguity and Environmental Data: Two Further Key Challenges of Multisensor Positioning." GPS World, October 2014.

[3] Elassal, A.A. "Generalized adjustment by least squares (GALS)." Photogrammetric Engineering and Remote Sensing, Vol.49, Issue 2. 1983.

[4] Colomina, I., Navarro, J.A., Térmens, A., "GeoTeX: a general point determination system". XVIIth International Congress of the ISPRS (International Society for Photogrammetry and Remote Sensing), 1992.8.2-14, Washington DC.

[5] Colomina I., Blázquez, M., Navarro J.A., Sastre J., "The need and keys for a new generation network adjustment software", International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2012.

[6] Kalman,R.E. "A new approach to linear filtering and prediction problems". Transactions of the ASME, Journal of Basic Engineering, Vol. 82, pp. 42-45. 1960.

[7] Ristic, Arulampelam, Gordon, 2004. "Beyond Kalman filter". Ed. Artech House.

[8] Schön, Thomas B., et al. "Sequential Monte Carlo Methods for System Identification." arXiv preprint arXiv:1503.06058. 2015.

[9] Navarro, J., "Object oriented technologies and beyond for software generation and integration in Geomatics." PhD. Thesis. November 1998.

[10] Molina, P. et al, "Qualitative Motion Analysis: INS/GNSS in Care-Giving Applications" GPS world, January 2011.

[11] Molina P., et al. "Drones to the rescue", InsideGNSS, pp. 36-47, July 2012.

[12] Silva P., Colomina,I. Miranda,C. Parés,M.E., "ENCORE: Enhanced Galileo Code Receiver for Surveying Applications", ION GNSS 2011.

[13] Montaño, J., et al. "Validation of inertial and imaging navigation techniques for space applications with UAVS". Proceeding of the DASIA 2015 conference. Barcelona 2015

[14] Angelats E., Molina,P., Parés,M.E. Colomina,I., "A parallax-based robust image matching for

improving multisensor navigation in GNSS-denied environments", in Proceedings of the ION GNSS+, 08-12 September 2014, Tampa, Florida (USA)

[15] Angelats,E. Parés,M.E. Colomina, I. "Methods, algorithms and tools for precise terrestrial navigation", In Proceedings of 9th International Geomatic Week, 15-17 March 2011, Barcelona (Spain).

[16] Fernandez, A., et al. "ATENEA: Advanced Techniques for Deeply Integrated GNSS/INS/LiDAR Navigation," Proceedings of the 24th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2011), Portland, OR, September 2011, pp. 2395-2405

[17] Silva, P.F. da, et al., "IADIRA: Inertial Aided Deeply Integrated Receiver Architecture," Proceedings of the 19th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2006), Fort Worth, TX, September 2006, pp. 2686-2694.