A Simulation Framework for UAV Sensor Fusion

Enrique Martí, Jesús García, and Jose Manuel Molina

Group of Applied Artificial Intelligence, Universidad Carlos III de Madrid, Av. de la Universidad Carlos III, 22, 28270 Colmenarejo, Madrid (Spain)
emarti@inf.uc3m.es, jgherrer@inf.uc3m.es, molina@ia.uc3m.es

Abstract. The design of complex fusion systems requires experimental analysis, following the classical structure of experiment design, data acquisition, experiment execution and analysis of the obtained results. We present here a framework with simulation capabilities for sensor fusion in aerial vehicles. Thanks to its abstraction level it only requires a few high level properties for defining a whole experiment. Its modular design offers flexibility and makes easy to complete its functionality. Finally, it includes a set of tools for fast development and more accurate analysis of the experimental results.

Keywords: sensor fusion, simulation framework, unmanned air vehicle.

1 Introduction

The research of fusion solutions to real-world complex problems is a time costly process, plagued by accessory tasks which demand a great effort to be done. The market offers powerful tools for accelerating some of the more generic parts, as data analysis or visualization. Nonetheless, as we focus in a more reduced and specialized field, it is common to find that one has to perform the expensive task of implementing its own tools.

Counting with an effective piece of software really makes a difference. Apart from the time saving, having a good toolbox for data representation/visualization can suppose detecting an otherwise ignored problem, or knowing how to improve the analyzed algorithms. This paper presents a generic framework for experimentation on unmanned air vehicles (UAV) sensor fusion. Bearing in mind the way in which such a tool is used, the whole system has been implemented in MATLAB™ to make it flexible, easily modifiable, as well as speeding up data visualization [1].

With illustrative purposes, this document shows its application to the multisensory navigation subsystem of a vehicle that is performing maneuvers related with air traffic management (ATM)[2]. Nonetheless, it can manage any other type of flight trajectories and even other kind of vehicles (such as maritime or terrestrial).

The structure of the simulator will be reviewed, with special attention to the design and functioning details of each module. We will also present our simulation and experimentation methodology, illustrating the process with some of the figures generated for results analysis and validation.

2 Simulator Architecture

The simulator is composed by three modules for data generation (see Fig. 1), plus an additional module for fusion algorithms and another one for performance evaluation. Data generation begins creating the specification of a vehicle trajectory. The resulting data is then feed to the aerodynamic model, generating the flight simulation. That process results in a set of values related with the dynamics of the UAV (such as position, attitude or accelerations), that can be easily used to synthesize realistic sensor measurements.

The fusion module takes the outputs of the selected sensors and processes them sequentially using the desired technique among the available library of implemented algorithms: (Extended) Kalman Filters, Particle Filters, etc.
Agustín.

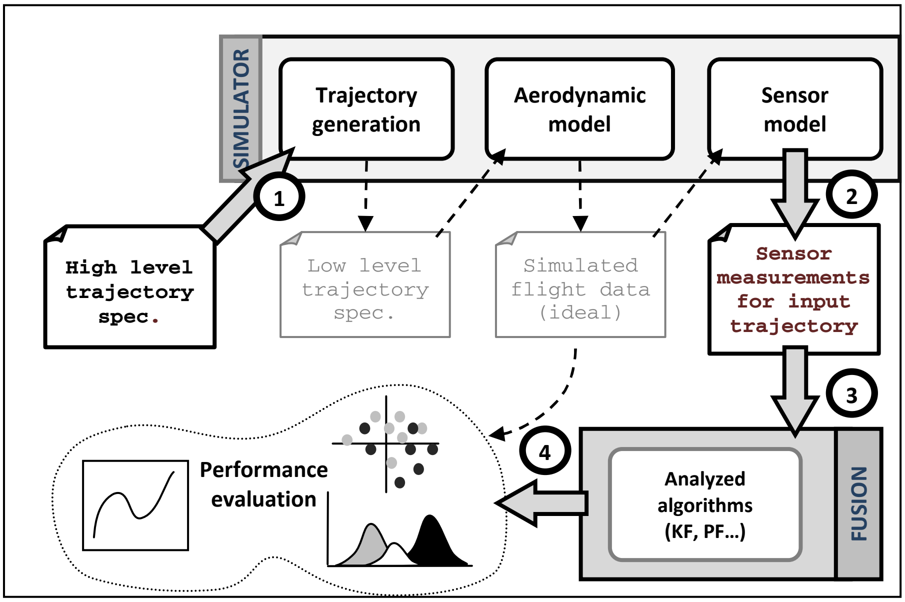

SIMULATOR
Trajectory generation
Aerodynamic model
Sensor model
1
2
High level trajectory spec.
Low level trajectory spec.
Simulated flight data (ideal)
Sensor measurements for input trajectory
3
4
Performance evaluation
Analyzed algorithms (KF, PF...)
FUSION

Fig. 1. Framework schematic view. Specification for a desired trajectory (1), sensors measurement models (2), fusion process (3), performance evaluation (4).

2.1 Trajectory Generation

This module is composed by several scripts and functions. They are focused on generating the input for the aerodynamic simulation from a high level specification of the trajectory.

The framework is provided with a set of functions for simple maneuvers such as straight flight at constant speed or with longitudinal accelerations/decelerations, or turns around a single axis of the vehicle local coordinate system. These basic pieces can be combined and concatenated then to generate more complex trajectories. In the case of ATM trajectories, we have created scripts for typical scenarios as the racetrack (performed during the waiting time before landing of an aircraft in order to fit with the time scheduled. They have the shape of a hippodrome, a rectangle with two semicircles attached to its shorter sides (**Fig. 2**).

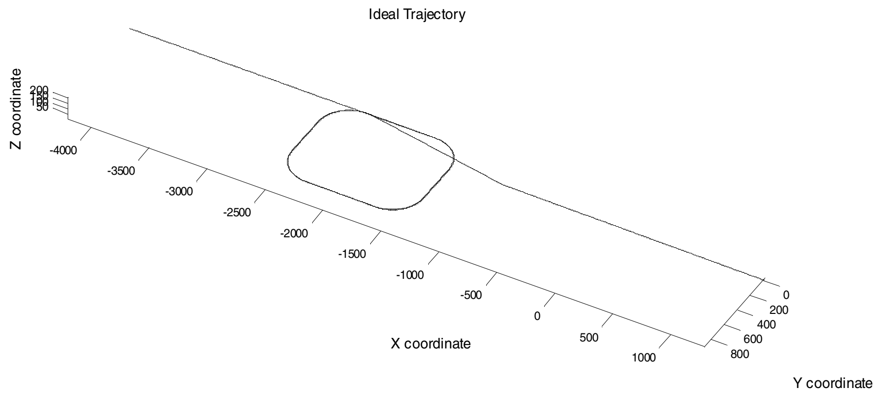


Fig. 2. 3D view of simulated racetrack+landing trajectory

The output of the system consists in six arrays of pairs instant-value. The three first arrays contain the forces in the body-fixed frame of reference of the vehicle (F_x, F_y, F_z), which determine the translation. The three remaining are the moments (M_{xx}, M_{yy}, M_{zz}) in the same frame, which determine the vehicle rotations.

2.2 Aerodynamic Simulation

The following step is the simulation of the vehicle dynamics. The selected model in our case has been, for the sake of simplicity, a rigid body with six degrees of freedom (6DoF in advance). Our reference implementation uses the aerospace MATLAB^{TM} Aerosim Aeronautical Simulation Block Set(1)[3][4], that provides a complete set of tools for rapid development of detailed 6DoF nonlinear generic aerial vehicle models and also a graphical view to check the behavior of system under test.

This generic motion model can be substituted by a more detailed scheme. The only requirement for the replacement is to generate all the real data needed to synthesize the measurements on the next phase. In our example we store the position, speed, attitude (in quaternion and Euler angles), accelerations and angular rates of the body. This information is enough for simulating all the common sensors.

As **Fig. 3**, the 6DoF dynamic model integrated both ideal segments to compose the simulated trajectory, and the simulated noisy sensor data (in its lower part). This is especially useful for online simulations, but is not recommended for experiments because the result will be sensitive to the particular generation of random noise injected in the data.

The "real" data of the flight is stored in separate files for later use. This is useful for creating persistent datasets and for saving computation time, because the simulation of a flight is usually a costly process implying complex numerical operations. Even in the case of a standard 6DoF system, it is required to solve some differential equations at each time step –and typical time step resolution is in the order of a few milliseconds.

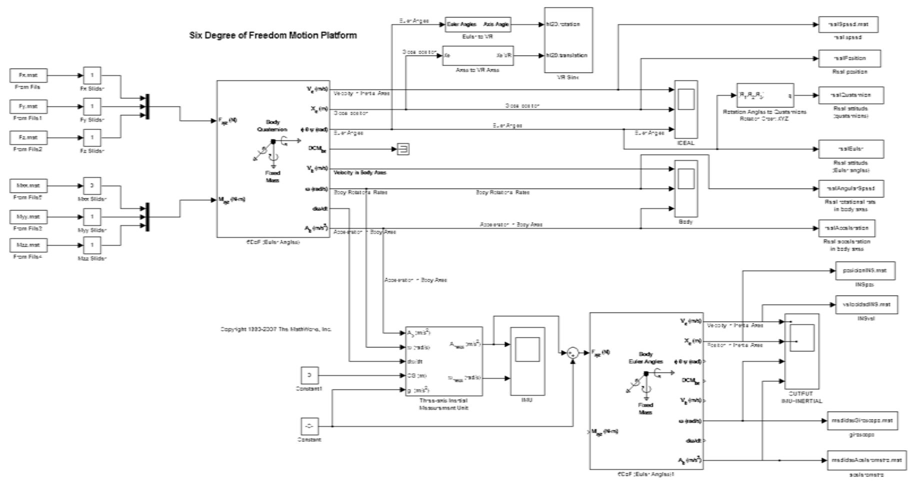


Fig. 3. Simulation of ideal trajectory and IMU measures with MATLAB Aerospace blockset

2.3 Generating Realistic Sensor Data

It is commonly known that sensors do not provide perfect information because they suffer from different effects such as inappropriate calibration, time drifts or interferences from external entities. Instead, their measures of real magnitudes are usually altered with an added random noise, and systematic effects as biases.

Sensor measures can be generated from “ideal” data quite straightforwardly, because all the aforementioned effects can be subsequently incorporated through simple operations –for instance, addition or matrix multiplication.

We can find an example in **Fig. 4**, where the ideal flight altitude (ascending line of crosses in the bottom part) is taken as starting point for simulating the measure of a barometric altimeter. The process consists in three consecutive steps:

- Simulate noise, by perturbing each value with a random sample drawn from a Gaussian distribution. The result is shown in hollow circles, also in the bottom.
- Add a constant value to mimic the altimeter bias. The bias is an effect mainly caused by differences between the sea level atmospheric pressure assumed by the device and its real value. The hollow triangles in the upper part are the result.
- The last effect to be simulated is the quantization step. Altimeters based in a barometer do not provide continuous measures: output values are quantized (here, the step is 50 meters). The final measures are the solid squares.

Thus, the reason for separating flight simulation from sensor measurement synthesis is quite simple: it simplifies the production of several generations of measurements for the same trajectory, and swapping among different sensor models.

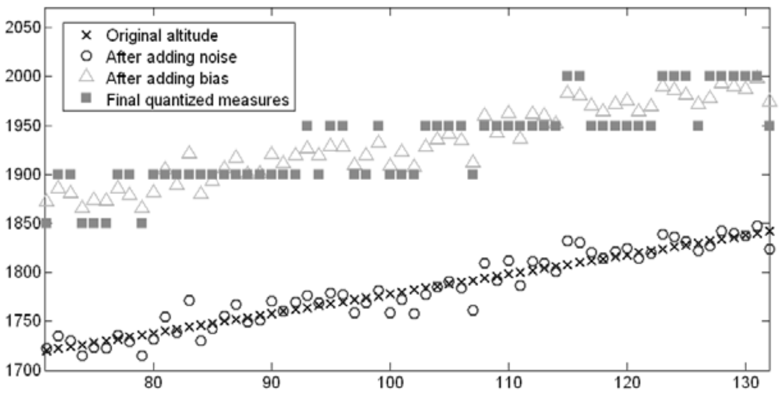


Fig. 4. Example of measurement generation using the model of a barometric altimeter. The starting point is the original ideal flight altitude.

The implemented framework applies the noise model of each sensor to every available sample of unaltered data, disregarding its temporal resolution. The produced values are individually marked with a timestamp, resulting in a very dense (and unrealistic) set of measures. This will allow later selection and tuning of the sensor update ratio, and the emulation of more advanced effects such as measurement loss.

Each sensor model is implemented as a separate function. It receives the ideal flight data and the parameters of the measurement model, and returns the simulated values.

3 Sensor Fusion

After generating all the necessary data, experiments can be performed. The first step consists on defining the set of available sensors and their features (including noise model and update ratio), the fusion architecture and the concrete algorithms to be used.

Once the architecture is defined [5], the great majority of the experiments can be run using a fixed scheme. All our experiments have been implemented over a few script templates. The next step is to configure the fusion algorithms. One of the available tracking techniques has to be selected and configured to make use of the selected inputs. The result at each time step is registered together with its timestamp.

The integration of GPS with inertial sensors attract a considerable number of research attention [6]. Plenty of classical and advanced techniques to increase the robustness have been applied, such as unscented Kalman, particle filters or soft computing paradigms [7][8][9]. There are many available algorithms available for their direct integration in Matlab. At this moment, our framework includes Kalman Filter, Extended Kalman Filter, Unscented Kalman Filter and Particle Filter [10][11]. As the references show, we have adapted existing libraries to work with our framework, such that its inclusion in the code consists in a few lines.

Once the whole trajectory has been filtered, we obtain a different interpretation of the flight trajectory. It can be directly compared with both real data and sensor measurements because the three sets provide values for the same sequence of time instants. Back to our illustrative example, we have performed several experiments of interest, as shown in the next subsections. A centralized processing architecture is the selected option for all the single-vehicle problems, given the coupling requirements to estimate sensor corrections together with trajectory parameters: a single algorithm of loosely coupled type will track the whole UAV state using the information from all the available sensors.

Typically, it is interesting to generate the measures in the same script the fusion is performed, because it allows to experiment also with sensor features, as the noise models to be used. One of the problems of dealing with noisy data is that a certain scenario can be particularly favorable (or unfavorable) for the applied algorithm, leading to non representative results. To avoid this, the whole trajectory is not filtered once per experiment. We will follow a Monte Carlo approach instead.

This means that for the same trajectory, several sets of measures will be generated. The noise used in the synthesis of each set of measures is, by definition, random and different on each generation, so the final set will be unique. The random number generator seed can be stored for assuring experiment reproducibility. With the simulated data sets, the Monte Carlo methodology allows for different experiments which can be run in order to perform rigorous statistical analysis on the output (root mean squared errors, t-test for performance comparison, integrity analysis, etc.).

4 Results Analysis and Validation

Evaluating the performance of a solution can be complex task. In order to facilitate it, our framework provides tools for supervising the process while it is executed, and to analyze the results one the trajectory has been filtered.

As an example of a tool of the first category, **Fig. 5** shows a 3D plot obtained for a Particle Filter (PF) during the tracking of a trajectory using a GPS, an accelerometer and a gyroscope as sensors. Each particle (the faded cloud in the right of the figure) is drawn as an arrow, with color intensities and sizes directly proportional to the weights of the particles. Note how only a few particles in the bottom are considered important after the last GPS measure is received (just over the X axis, near the -3590 meters mark), while the vast majority of the population is represented in a very light pale tone. The estimation of the filter is the wide arrow with triangles in the extremes.

Intermediate figures of this kind have multiple uses, such as visualizing with high detail on each step to diagnose the causes of a previously detected problem. For instance, we can supervise if the resampling stage is introducing enough variability in the population of particles.

The overall performance of a certain solution, however, can only be evaluated after the experiment is finished. MATLAB^{TM} makes very easy the calculation of different statistics and plotting the desired variables. The real dare is to select the appropriate quality indicators. Next figures are some examples obtained using our framework.

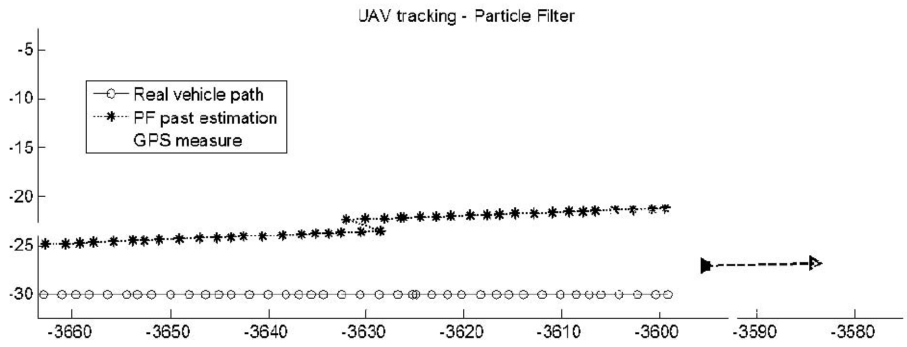


Fig. 5. Auxiliary plot to help visualizing current system state during fusion process

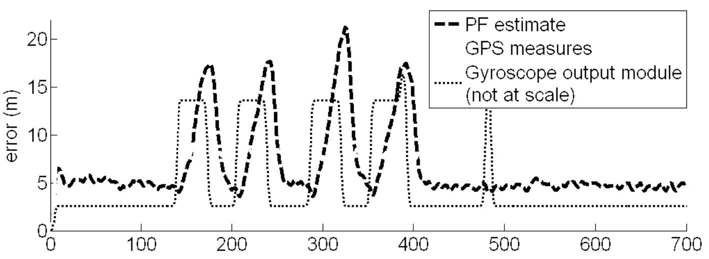
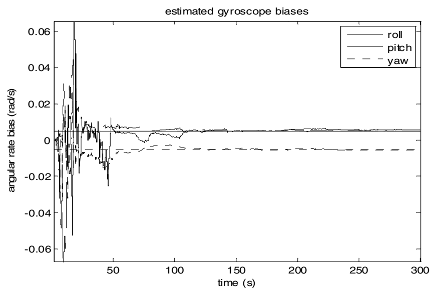


Fig. 6. Comparison between position estimation accuracy of a filtering algorithm against the raw GPS error. Gyroscope output shows context about turns and straight segments.



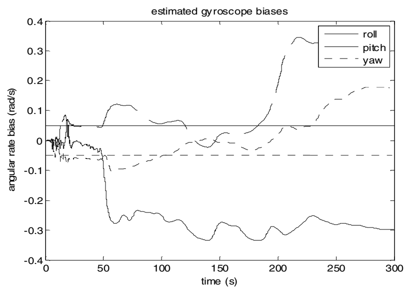


Fig. 7. Gyroscope bias estimation for an EKF-based solution of GPS+IMU fusion

Fig. 8. Unstable gyroscope bias estimation for an EKF-based solution of a GPS+IMU fusion

5 Conclusions

A framework for experimenting on sensor fusion has been presented in this document. Apart from detailing its structure, we have shown how it can be used for creating a whole experiment, starting with the generation of a simulated flight trajectory and ending with graphical descriptions of the results.

Using this software, we have reduced the implementation time of an already designed experiment to just a few minutes. Evaluating how a change in a variable affects the result is trivial, as well as changing the value of any set of configuration parameters. If required, the functionality can be completed by adding new algorithms, measure models or trajectory definitions.

Acknowledgements

This work was supported in part by Projects ATLANTIDA, CICYT TIN2008-06742-C02-02/TSI, CICYT TEC2008-06732-C02-02/TEC, SINPROB, CAM CONTEXTS S2009/TIC-1485 and DPS2008-07029-C02-02.

References

[1] Gade, K.: NAVLAB, a Generic Simulation and Post-processing Tool for Navigation. European Journal of Navigation 2(4), 51–59 (2004)

[2] Rodriguez, A.L., et al.: Real time sensor acquisition platform for experimental UAV research. In: IEEE/AIAA 28th DASC 2009, pp. 5.C.5-1–5.C.5-10 (October 2009)

[3] Aerospace Toolbox - MATLAB. The MathWorks, `http://www.mathworks.com/products/aerotb/` (Cited: 03 15, 2010)

[4] Kurnaz, S., Cetin, O., Kaynak, O.: Fuzzy Logic Based Approach to Design of Flight Control and Navigation Tasks for Autonomous Unmanned Aerial Vehicles. Journal of Intelligent and Robotic Systems 54(1-3), 229–244 (2009)

[5] García, J., et al.: Data fusion architectures for autonomous vehicles using heterogeneous sensors. In: 1st ESA NAVITEC. Noordwikj, Holland (December 2006)

[6] Wagner, J.F., Wienekeb, T.: Integrating satellite and inertial navigation—conventional and new fusion approaches. Control Engineering Practice 11(5), 543–550 (2003)

[7] van der Merwe, R., Wan, E., Julier, S.: Sigma Point Kalman Filters for Nonlinear Estimation and Sensor Fusion: Applications to Integrated Navigation. In: AIAA Guidance, Navigation and Controls Conference, Providence, USA (August 2004)

[8] Crassidis, J.: Sigma-Point Kalman Filtering for Integrated GPS and Inertial Navigation. IEEE Trans. on AES 42(2) (April 2006)

[9] Chiang, K.W., Huang, Y.W.: An intelligent navigator for seamless INS/GPS integrated land vehicle navigation applications. Applied Soft Computing 8(1), 722–733 (2008)

[10] Hartikainen, J., Sarkka, S.: Optimal filtering with Kalman filters and smoothers-a Manual for Matlab toolbox EKF/UKF (2007), `http://www.lce.hut.fi/research/mm/ekfukf/`

[11] Chen, L., et al.: PFLib - An Object Oriented MATLAB Toolbox for Particle Filtering. Department of Statistics - Colorado State University (2007), `http://www.stat.colostate.edu/~chihoon/paper-6567-25-revised.pdf` (Cited: 03 14, 2010)