STORM, THE NEW AIRBUS ROTORCRAFT SIMULATION TOOL BASED ON 60 YEARS OF CUMULATED EXPERIENCE IN DIGITAL FLIGHT PHYSICS

Didier Casolaro, didier.casolaro@airbus.com, Airbus Helicopters (FRANCE)

Aerodynamic Department, Airbus Helicopters

Abstract

The paper presents the new in-house rotorcraft aeromechanic simulation tool developed by Airbus. STORM (Simulation Tool for Overall Rotorcraft Modelling) gathers more than sixty years of experience in rotorcraft simulation and paves the way towards great perspectives thanks to its open architecture. STORM was developed to replace the former in-house aeromechanic simulation tool, HOST (Helicopter Overall Simulation Tool) developed in the nineties and widely used on engineering side by various disciplines, and also for building flight loops for real time training simulators.

From the requirements given by the community of users, a new architecture was envisaged based on HOST foundations. A first feasibility analysis was launched for one year before deciding to run the development of the future Airbus Helicopters aeromechanic simulation tool.

Key features of what STORM brings to rotorcraft aeromechanic simulation are presented, and the future capabilities are detailed.

NOTATION

AH	Airbus Helicopters
API	Application Programming Interface
CFD	Computational Fluid Dynamic
CLI	Command Line Interface
CPU	Central Processing Unit
DLR	*Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)*
eVTOL	*electric Vertical Takeoff and Landing Vehicle*
GANESH	Generalized Aeromechanics for Nested and Enhanced Simulations of Helicopters
GENSIM	GENeralized SIMulation
GHOST	Generalized HOST
GUI	Graphical User Interface
GUST	Graphical User Interface for STORM
HOST	Helicopter Overall Simulation Tool
MANO	MANoeuver Optimizer
MBS	Multi Body Simulation
NAMP	New AeroMechanic Platform
ONERA	Office National d'Etudes et de Recherches Aérospatiales (French Aerospace Center)
POST	STORM POST processing unit
PRE	STORM PRE processing unit
PYJAMA	Python Jettisoning Armament Monitoring Application
SDK	Software Development Kit
STORM	Simulation Tool for Overall Rotorcraft Modeling
UAV	Unmanned Aerial Vehicle
VAST	Versatile Aeromechanic Simulation Tool

1. INTRODUCTION

At the time Eurocopter was created in 1992 from Aerospatiale and Deutsche Aerospace, the idea of improving the existing aeromechanic simulation tools emerged both in France and in Germany. HOST was developed based on former existing tools on the French side whereas GENSIM was developed based on former existing German tools with one objective for each country: guarantee consistency of results for existing fleet.

Both HOST and GENSIM were developed with two different philosophies thus introducing some difficulties for exchanging data and applying the same methods for new developments.

On one side, HOST offered extended capabilities for simulating any kind of rotorcraft thanks to its modular architecture, but its usage was a bit too complex for people who were not sufficiently trained. More especially, it was nearly impossible to setup a script for running hundreds of simulations for covering the flight envelope unless you get sufficient IT skills. On the other side, GENSIM did not offer the possibility to simulate other rotorcrafts than helicopters but offered the possibility to define complex simulation process schemes thanks to its simple architecture for setting up the data.

In 2012, both HOST and GENSIM were tools based on outdated architecture and programming language and the need to develop a modern simulation tool based on the latest technologies arose. Having one single tool shared between France and Germany was also considered as a great opportunity towards more synergies between the two countries for future rotorcraft developments.

Finally, the decision for launching the development of a new aeromechanic platform (NAMP) was made in 2015, and the feasibility analysis started in 2016 for one year before the decision was made for running the STORM development mid of 2017.

2. SPECIFICATION

The chance of starting a new development with more than 30 years of experience with “combat” proven tools makes the definition of the needs easier to handle. Indeed, we could get a very accurate picture of the benefits and deficiencies for both HOST and GENSIM. A first obvious requirement immediately came out: the simulation capabilities of STORM shall not be less than what was offered by HOST and GENSIM. This first requirement was the keystone allowing to ensure the follow up of the existing fleet with STORM.

Then, came out the requirements related with the experience of the usage of HOST and GENSIM for decades. What is to be kept? What is to be improved?

- Ability to simulate any kind of rotorcraft: this requirement is directly inherited from HOST capabilities and also connected with the actual landscape of rotorcraft development with new concepts such as the compound rotorcraft (RACER), UAV and eVTOL for Urban Air Mobility (CityAirbus).
- Disconnect the graphical user interface from the computation kernel: this requirement is also inherited from HOST and induced by the choice which was made in the 90’s to integrate a GUI using a graphical library based on technologies developed at a time when the mouse did not yet exist! The obsolescence of this graphical library made HOST maintenance very complex and the user friendliness of the user interface was no longer up to current standards.
- Preserve real time capabilities: HOST was used so far for producing real time flight loops to be used for test benches and also for training simulators. This key capability shall be kept.
- Scripting capabilities: due to its architecture, GENSIM offered the possibility to use Python script for easily setting up the input data file and running complex simulation schemes. This capability came out with HOST in 2015 with a demonstrator which allowed running simulation from a proprietary scripting language offering less capabilities than Python scripting.
- Easy physical model development: implementing any improvement to the physics in HOST or GENSIM was difficult and therefore a limiting factor for improving the simulation fidelity. These two tools were developed in FORTRAN 77 which is no longer mastered by young engineers. Developing new physical models in HOST was something supposed to be easy in the 90’s for people having been trained: the modular approach of HOST provides that the integration of new physical models is safe but requires advanced knowledge of the object oriented interface structure without object oriented language capabilities. In the case of GENSIM, implementing additional features was not easier since it required the overall knowledge of the entire source code structure. For years, the development of new advanced physical models was reduced, because of the emergence of coupling capabilities with CFD, and also because the time required to develop such models was no more compatible with industrial needs. On the other side young engineers are hired with skills on modern scientific tools such

as Python: the best solution for developing synergies around the improvement of the simulation fidelity with new physical models may be to allow to use Python language for coding these models.

- Easier coupling with external tools: both HOST and GENSIM have got the capability to exchange data with external tools with lots of coding effort even though coupling between HOST and CFD could be achieved thanks to GHOST which relies on GANESH. STORM shall be able to extend coupling capabilities with an integrated generalized interface allowing to minimize the coding effort when coupling.
- Finally, the idea of starting STORM development came out at the same time DLR started VAST development. Even though these two tools are not based on the same architecture, some specific features had to be considered for minimizing the effort when physical models had to be developed for both STORM and VAST.

3. ARCHITECTURE AND TECHNOLOGIES

The decision was made to start STORM development from HOST since its internal architecture allowed to comply with the more complex requirement asking for being able to handle any kind of rotorcraft configuration. For keeping real time capabilities, the choice was made to preserve a compiled programming language for all the features needed for real time simulation. In order to comply the need of having scripting capabilities and improving the interfaces for easier coupling capabilities, the choice was made to use a Python wrapping.

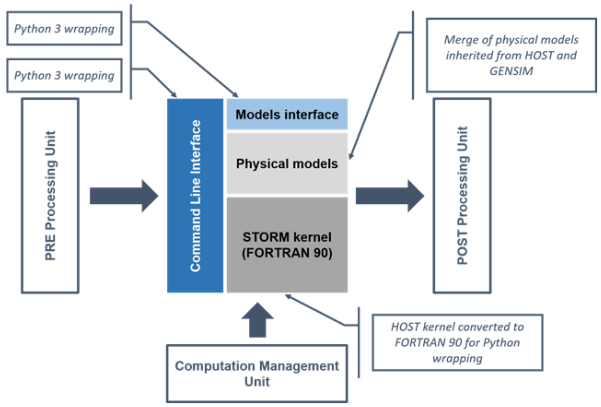

Fig 1 - STORM Architecture overview

In order to ease the wrapping with Python and also prepare the compiled part of the source code to a more object oriented approach, the entire source code inherited from HOST was converted from FORTRAN 77 into FORTRAN 90.

The deeper functionalities of the kernel were not modified and the kernel subroutines ensuring the communication with the physical models were adapted.

The kernel is entirely coded in FORTRAN 90. It manages the physical models and integrates the core solver. The physical models have to be considered as standalone pieces of source code that are receiving inputs and producing outputs. Inputs and outputs are formatted with the physical model interface and the kernel is in charge of building trees for calling the models in the appropriate order. With this kind of architecture, the coding language used for developing a physical model is entirely free, as soon as this coding language is able to handle the model interface written in Python. For sure, if the model shall be used for real time simulation, the coding language shall be a compiled one, and preferably FORTRAN 90 in order to preserver computation time requirements. Finally, an additional Python layer comes on top for allowing the user to manipulate the simulation functions and also the models from the CLI. Around this part of the tool in charge of simulation, a GUI has been developed for easing the life of users:

- a pre-processing unit which mainly manipulates the models interfaces for building the connections,
- a computation management unit, GUST, which is in charge of manipulating mainly the CLI and dedicated to users who do not have Python skills for building advanced simulation schemes, and
- a post processing unit which allows anyone displaying plots, producing elaborated parameters or any kind of output analysis required by each discipline without having to produce any line of source code and also without having to use any third party software.

3.1 Physical model development

With such an architecture, physical models do not need to be coded using FORTRAN language unless this model is intended to be used for real time simulation. Hence, anyone having some basic Python skills can develop its own physical model thanks to the advanced API providing features allowing to easily manipulate the required interface with the kernel. In addition, while applying as set of coding rules, this Python model can be converted into FORTRAN for a real time usage with a minimum effort.

The basic principle remains the same as the one used for HOST. STORM kernel manages the configuration of the object to be simulated (most of the time, a rotorcraft) while building a tree structure

of all the physical models to be called. The physical models are exchanging data with each other thanks to dedicated connection points (so called links in STORM nomenclature) which are managed by the kernel. A physical model is to be considered as a standalone part of code which ignores to which models it is connected to: everything is mastered by the kernel which has the task to exchange data accordingly at the appropriate time. For achieving this, the model interface (see Fig 2) was inherited from the one used in HOST, and modernized thanks to the Python wrapping.

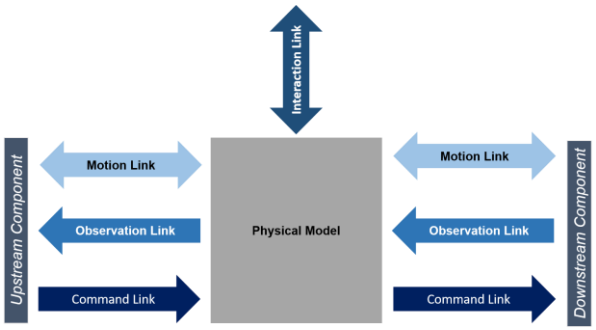

Fig 2 - Physical Model Interface

The kernel is in charge of building a first tree structure for transferring the motion from the upstream element to the downstream element in the kinematic path thanks to the motion link. The same link object is used in the load path for transferring the loads from the downstream elements to the upstream element. Another object named “Observation link” is in charge of transferring parameters produced by the model, when they are properly defined (eg after the load path is applied on this model). In addition another tree structure in managed by the kernel for sending commands from the upstream component to the downstream component. Finally, the interaction link is in charge of exchanging data needed for managing aerodynamic interactions.

3.2 Coupling with external tools

This model interface allows an easier coupling with external tools in a more generalized way than the one used with HOST and GENSIM.

Coupling with CFD consists in overwriting some data computed with an external tool: in this case we are just correcting the output of the equivalent physical model in STORM.

The current model interface makes possible to connect an external tool providing part of the physics which is not available in the current physical models. As for example users keen to have more advanced physics in the swashplate may replace the integrated swashplate model with another one modeled with a MBS tool. For that purpose they will just have to use the model interface for enabling the dialog between this tool and STORM kernel.

Other types of coupling with third party S/W may consist in correcting the outputs provided by an internal model. This is the case, for example when coupling with CFD: the induced velocities and the blade loads produced by STORM are corrected with the ones given by the CFD computation: STORM is still in charge of mastering the trim state of the rotorcraft.

4. DEVELOPMENT PROCESS

4.1 From HOST to STORM

The decision made to start the development from an existing tool used for decades gave us the opportunity to work in a secured environment allowing to check at any time that there was not any regression while switching from HOST to STORM. A non-regression data base of more than 300 test cases was used for making sure that the results obtained with STORM were the same as the ones produced by HOST. The major difficulty the development team had to face was to update STORM with the latest improvement brought into HOST while STORM was being developed. Indeed, while waiting for STORM to be released, and at least for the development time period which lasted 40 months, HOST was still maintained and so was constantly evolving.

4.2 GENSIM models integration

Since GENSIM and HOST did not share the same architecture, the integration of physical models available in GENSIM but not in STORM requested additional work for adapting the source code to the desired architecture. Also, testing the proper integration of GENSIM models inside STORM was not as obvious as HOST based models. As soon as STORM and GENSIM were not sharing the same mathematical integration scheme, a perfect match between the results of STORM and GENSIM could not be expected. First we had to implement additional features allowing to isolate the physical model to be implemented, and then engineering judgment was needed to state about the validity of the result.

The integration of GENSIM models was also the opportunity to integrate functions and options existing in GENSIM but not in HOST as required by the users: all what could be done with HOST and GENSIM must be done with STORM.

5. THE POWER OF SCRIPTING

The CLI provided by STORM allows to manipulate the basic simulation functions: trim computation, time domain simulation, linearization, etc…, but not only. The CLI can also be used for manipulating the input data without having to edit the input files. The initial purpose of this feature was to allow the user

to run sensitivity analysis on some input parameters without having to edit the input data files. Finally this feature is also considered as a way to ensure a perfect traceability of the computation: the input data files remains the reference and all the modified parameters can be clearly identified in the script.

The extended capabilities given by the various packages available with Python and coupled with the advanced features of STORM CLI allowed to generate advanced tools dedicated to complex tasks which required lots of operations from the user in the past. The daily life of aeromechanic engineers is simplified and they can keep concentrated on their major activity: analysis of the simulation results and understanding of the physics, the usage of the simulation tool has now become transparent.

6. GRAPHICAL USER INTERFACES

6.1 Pre Processing

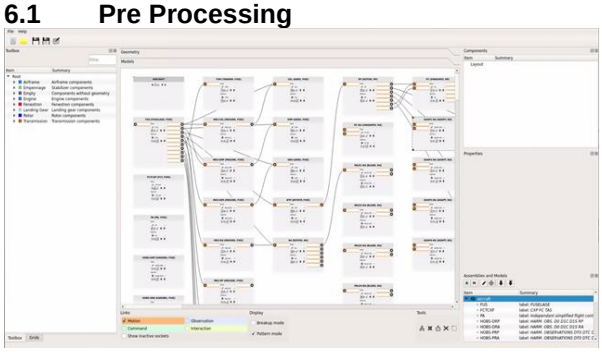

Fig 3 - STORM PRE processing unit (Link Motion Display)

The experience acquired with more than 25 years in using a text editor as the primary GUI for generating input data for aeromechanic simulation made us believe that having a dedicated tool allowing to setup a rotorcraft model was needed: it allows avoiding input mistakes, it prevents the user from using intensively copy and paste which is most of the time the source of these mistakes, and finally it is a way to setup a complex configuration without having to spend hours in reading the input data file documentation.

Considering the example of a simplified helicopter model (see Fig 4), the user will have to build a set of text files to detail how the various components are connected all together and what are the data to be used for simulating the physic for each of them.

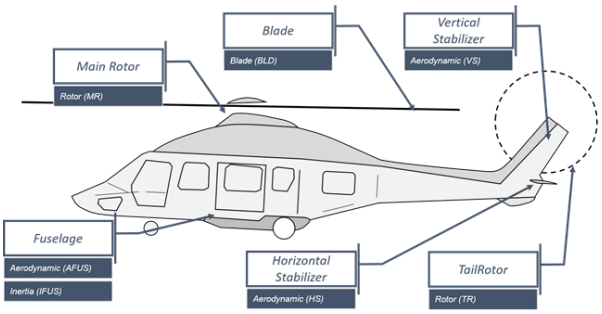

Fig 4 - Basic Helicopter Model

In the past, with HOST, building such files required advanced knowledges for each physical models and most of the time, users were starting from existing files built for a previous helicopter. Building an eight rotors UAV configuration, for example, was not an easy tasks with a basic text editor and required the support from HOST experts. One of the objectives of STORM development was to allow any user to quickly set-up a complex rotorcraft configuration without having to read the entire documentation. This is achieved with the development of the PRE Processing Unit (see Fig 4). The pre-processing unit allows generating templates dedicated to a rotorcraft configuration (helicopter, combined, tilt, multi rotor, etc…). Templates are models without data. In other words, the most difficult task consisting in the models connection is integrated in the template and the user has just to provide the data for each model.

Also the PRE Processing Unit gives the opportunity to visualize the aircraft in a 3D view allowing to make sure that the positioning of each component is correct.

In addition, specific editors are proposed for feeding the data for each physical model. These editors are in charge of checking the validity of each data given by the user and are providing specific features that help in understanding what is needed by the model.

6.2 Simulation process management

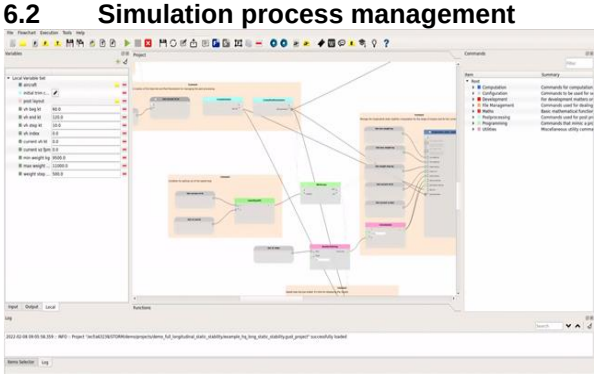

Fig 5 – GUST screen shot

In order to allow users who are not familiar with Python scripting and who would nevertheless like

to run advanced simulation scenario, a dedicated Graphical User Interface called GUST was developed. This tool allows manipulating the main functions of the API and the main data without having to produce any line of Python source code: this is the so called “Visual Script” as defined in GUST documentation. A specific interface was also developed for allowing users who are mastering Python scripting to produce advanced functions that can be used with this GUI. The main principle of this tool consist in manipulating boxes that represents API functions (or a more complex set of actions defined from a Python script) and data for building a logic flow and a data flow (see Fig 5). GUST can also generate Python scripts so that users can learn about the API without having to dive deeply into the online documentation.

6.2.1 Variables

Various types of variable are available corresponding to the type of data needed for the simulation. These type can be of single type such as a numeric value (float or integer), or of complex type such as a trim law or a control law. Each type of variable comes with a specific editor.

6.2.2 Integrated commands

Basic functionalities provided by STORM API are interfaced with GUST. Additional features such as loop management, conditions management and basic math operations are also provided. From this basic commands, any user can build a program without having to produce any line of code.

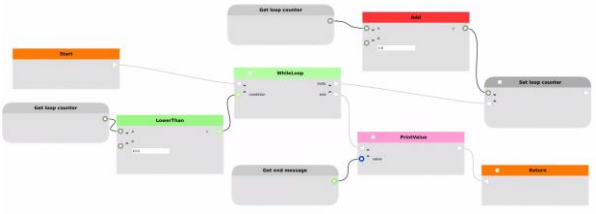

Fig 6 - GUST: While loop example

In the example depicted in Fig 6, we can observe how to deal with the while loop command. This example is just increasing a counter from 1 up to 10 and displays a message in the console when we exit the loop. Most of the time, commands have one input flow and one output flow, acting as basic functions. In the particular case of loop commands or conditions commands, there is one input flow and two output flows: one in case of the condition is met and another one in case of the conditions is not met.

Logic flow connection point is symbolized with triangle whereas data flow connection is symbolized with disc (the disc color is associated with the type of variable).

Fig 7 - GUST: While Loop Command

Fig 7 gives more details about the particular example of the while loop command with one upstream input flow and one input variable (the condition to exit the loop), and two downstream output flows: the body, what is to be executed when the condition is not met (you are inside the loop), and the end, what is to be done when the exit condition is met.

6.2.3 Building functions

6.2.3.1 From GUST

It is possible to build functions while defining input and outputs they shall produce. GUST functions can be saved, and can be managed within a dedicated browser. This kind of feature was a key point for making the visual script as efficient as a programming language.

Fig 8 - GUST function example

As for example, while computing cruise, climb or descent trim condition, you may have to switch from a trim law where the bank attitude is set free (low airspeed) to a trim law where the bank attitude is imposed (high speed). We can envisage building a function that uses the appropriate trim law according to the input speed. Once this function is designed, it can be re-used by the entire user

community and so building visual script will be faster.

In the example depicted in Fig 8, a more powerful function for computing trim is proposed: this function requires the definition of the trim conditions to be computed (speed in the horizontal plane and vertical speed) and also a speed threshold to switch from low airspeed trim law to a high airspeed trim law. It is also proposed as an output parameter to have a flag raised to warn the user if the low airspeed trim law (imposed drift in horizontal plane) was used.

6.2.3.2 From a Python script

As mentioned previously, it is also possible to create a function box from a Python script with a minimum of coding rules that are just consisting in defining inputs and outputs so that GUST can understand how to display this box.

Visual script was developed for allowing user who are not comfortable with Python scripting to use the maximum capabilities provided by STORM. In some cases, defining complex operations from GUST can become a bit fastidious and it would be more efficient to use directly Python script. On the other hand, this new function developed from a Python script shall also be available to users that are only using GUST.

Fig 9 - GUST function defined from a script

In the example given in Fig 9, we are defining a control law allowing to describe the recoil loads providing by a gun. This recoil load is defined from a sinusoidal signal which depends on the firing period, a time constant to be considered when we stop firing, the signal amplitude and also the gun position. While applying some basic coding rules aiming at defining the input and output parameter of this function accordingly with the type of variables managed by GUST, it is possible to have this function available for GUST users without having to release a new GUST version.

6.3 Post Processing

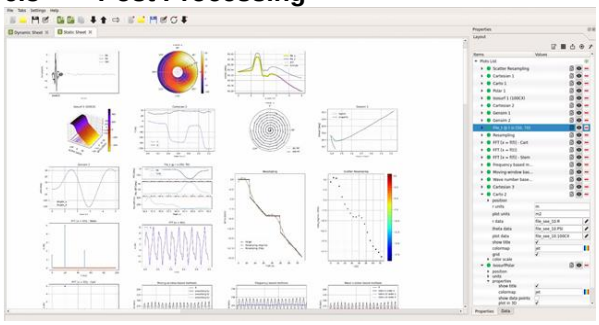

Fig 10 - STORM POST processing unit

One of the objectives of STORM development was to propose an integrated suite of tools allowing to master aeromechanic simulation from the building of the data package up to the simulation result analysis. For that purpose a dedicated tool for analyzing the simulation results was also developed. It should allow in a near future to ensure a full consistency of the entire simulation process including the post processing. The way data were post processed were mainly depending on the user skills with off the shelf post processing tools he could access: some were expert in Tecplot™ usage, some other were using Microsoft Excel™ or Python script with their preferred plot generator library… Furthermore, as these generic tools were not developed for rotorcraft simulation analysis, each user had to build on its specific procedures to match with his needs. Mastering the tool used for post processing simulation data is a way to improve our efficiency while providing users with dedicated features that are consistent with their expectations and also making sure that anyone can understand how the conclusions were built since everybody use the same tool (see Fig 10). The post processing analysis unit basic features can also be accessed from a dedicated Python API to allow users who are mastering Python scripting to build scripts that are managing the modification to be implemented in the reference input model, the simulation process, but also the production of plots and additional data dedicated to the final report.

Post processing consist in mainly two kinds of actions: data manipulation or data generation, and data display.

In order to keep consistent with user habits in post processing data with commercial tools, we decided to split the post processing tool in two sub-tools integrated in the same GUI.

6.3.1 Data manipulation and generation

One part of the tool is in charge of data manipulation and/or data generation. The user manipulates a so called Data Set that could be represented as a Microsoft Excel™ workbook. A Data Set is gathering several Data Actions that could be associated with a Microsoft Excel™ worksheet: a table. With an object oriented architecture, a Data Set is gathering several Data Actions aiming at producing a table of data and each Data Action is associated with a specific way of manipulating data or producing new data.

As for example, the first basic Data Action which was implemented was the one consisting in loading a data file. Quickly came the need of producing additional parameters from the ones obtained from a reading Data Action and so the Formula Data Action was developed…

In order to get the same level of flexibility user were used to get with their preferred commercial tool, we offered the possibility to create specific data actions without having to request anything to the development team. Once again, a minimum set of coding rules have to be considered so that the Data Action created by one user can be integrated without any effort in the post processing unit.

6.3.2 Display

The second part of the post processing unit consists in displaying data. The GUI in charge of displaying data only requires a Data Set as input. Hence, it is possible to save a layout in which the user specified the display parameter for each plot independently of the data it is connected to. It allows having a standardized set of layouts associated with each kind of study.

The plotting tool is based on matplotlib™ library which offers a large set of capabilities. The GUI was developed to allow anybody who is not familiar with this library to quickly set layouts.

The post processing unit allows to display parameters as a function of time in a dedicated view with the possibility to visualize the aircraft trajectory in 3D viewports (Fig 11).

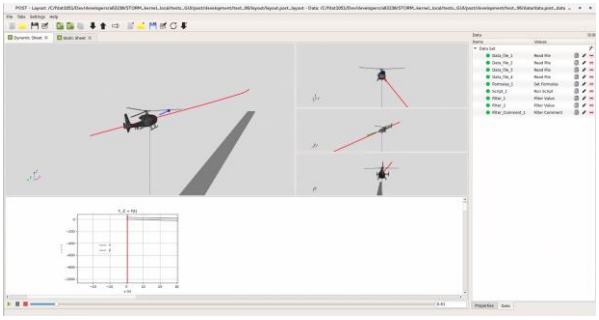

Fig 11 - STORM post processing: dynamic display

6.4 Specific applications

STORM was officially released in AH design office in January 2021. Many scripts dedicated to specific disciplines were quickly developed and some of them were entirely revisited by the development team to ease their usage and propose a standardized environment through GUIs.

6.4.1 MANO

Assessing structural load substantiation requires to apply a normalized longitudinal stick input to reach a load factor at a given airspeed which is consistent with the rotor envelope limits. In the past years, this process required the full attention of aerodynamic loads engineers applying HOST or GENSIM to iterate over the initial trim conditions and the shape of the longitudinal stick input so that the rotor envelope can be reached.

Thanks to the advanced CLI and also to the efficiency of available Python library, the process was entirely automatized with a dedicated algorithm combining a genetic algorithm and a gradient based method.

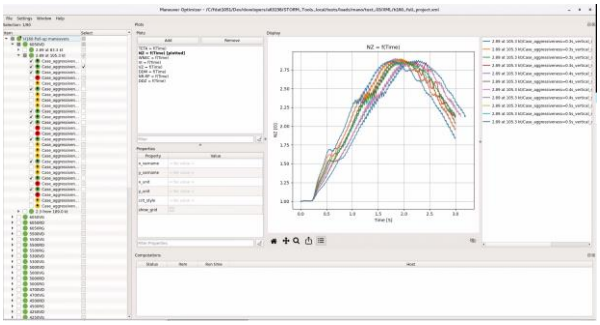

Fig 12 - MANO screen shot

This algorithm optimizes the longitudinal input amplitude and the initial trim speed for reaching the required vertical load factor at the required airspeed. The other parameters, such as the initial vertical speed, the input aggressiveness and the duration the longitudinal stick is maintain backward (or forward in case of push over) are sensitivity parameters managed by the genetic algorithm. Thanks to this approach, the algorithm is able to find several solutions for the same point on the rotor envelope. Each maneuver simulation can be

computed on one CPU that makes parallel computing very efficient.

As a consequence, a new problem came out: how to post-process this huge amount of data? How to help our engineers to efficiently analyze the simulation results and select the most appropriate solution according to their needs? For that purpose a dedicated GUI was created for setting up the study parameters, running and monitoring the computation on several CPUs, and post processing the results.

This tool was developed first for dealing with pull-up and push over maneuvers keeping in mind that additional types of maneuvers will have to be considered for satisfying the needs of end users.

Thanks to this new way of handling aeromechanic simulation, a load computation loop can be performed within few days instead of several weeks.

6.4.2 PYJAMA

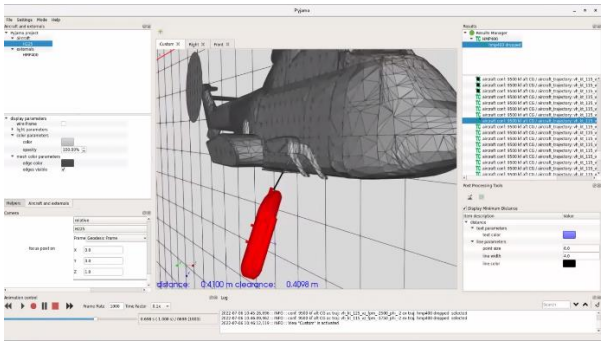

Fig 13 - PYJAMA Post Processing

PYJAMA is a good example showing that mastering graphic technology can help a lot in building efficient applications.

Armament jettisoning and firing requires first to compute the trajectory of one object from an aircraft state (attitudes, speeds, attitude rates) and considering a specific aerodynamic environment (influence of the aircraft fuselage, influence of the rotor flow, etc…). This part of the process can be easily ensured by a set of chained STORM simulations.

The second step of the process consists in computing the minimum distance between the jettisoned object and the aircraft through the time. This is ensured with the integrated function of the 3D library that allows computing the distance between two meshed objects within seconds using the capabilities of the graphics card.

Finally the last step relies on data post processing (Fig 13) in order to build the safe jettisoning envelope and understanding how the object behaves beside the aircraft. For this last part, commercial tools were used to display the 3D views and to produce movies.

The idea of producing an integrated tool allowing to manage the computation setup, the computation of the trajectory as well as the minimum distance and the post processing was launched when adapting the existing script for trajectory computation to STORM. For dealing with 3D views, the choice was made to use the VTK library, and, as for most of the 3D API, advanced functions could be used allowing to compute the distance between two meshed objects.

Now, the jettisoning process is standardized within a tool which is entirely mastered: from the pre-processing, to the post processing, the entire tool chain relies on in house developments providing features consistent with user requirements. This standard helps in more efficient exchanges from one user to another and through time. As for example, post processing the data could have been performed with existing tools available on the market, but in this case, each user would have selected the tool he is more comfortable with preventing another user to understand how the data were post processed if he does not master this tool.

7. FUTURE IMPROVEMENTS

7.1 Current status

STORM development was planned in two steps: the first one, which is now achieved, consisted in delivering to the user a stable version compliant with the primary requirement to provide the same physical modelling and computation features as HOST and GENSIM. Hence the first development stage was more user oriented than developer oriented. It was also a way to collect additional requests from the user so that they can be considered in the second development step the most efficient way.

The second step is more developer oriented and consists in setting up the new internal architecture for ensuring an easier code maintenance and support for the coming decades.

7.2 Basic cleaning

The current source code of STORM integrates a huge part of subroutines inherited from HOST that had to be revised. The ultimate goal would be that the “clever” part of STORM entirely written in FORTRAN 90 shall not have to deal with any file management, string management and any system call management: a scientific tool shall only have to deal with computation of physical effects while other tasks such as loading data, writing results or sending messages to users shall be performed outside of this core part. This will definitely cut any link between the FORTRAN part and the system

that would allow minimizing the side effects of operating system evolutions: FORTRAN takes in charge only the computation and other tasks are ensured by another programming language which may evolve with the new system technologies.

7.3 Towards the desired architecture

In parallel to the basic cleaning, the new architecture consisting in using a more object oriented approach at all levels is being started. The various “link” objects used for the model interface will be modernized in order to make the life easier for the users keen to develop physical models. Having a SDK developed for the users community within the coming years will contribute in the further improvement of simulation fidelity since it will allow anyone to propose his idea without excessive programming skills.

Also, revising the internal architecture of some complex models such as the Fenestron model or the Hub model, with a more object oriented approach will allow an easier maintenance and also ease the implementation of future improvements.

A great improvement inside a core model, the blade model, will be brought in the coming months. It will mainly consist in splitting the blade physical model into two sub-models: one in charge of computing the aerodynamic forces, the other one in charge of computing the motion of each blade element. The advantage of this kind of architecture is to allow an easier implementation of any kind of beam model for having soft blade capabilities. On one side, it will increase the setting up complexity of a rotorcraft model since the users will have to connect two models for creating a blade, but this problem can be overcome with a dedicated template for building a blade in the pre processing unit.

8. CONCLUSION

After more than 25 years of usage and based on the cumulated experience of more than 50 years in rotorcraft simulation, HOST and GENSIM gave birth to STORM.

This new aeromechanic simulation platform is based on modern technologies that are offering more user-friendliness on one side and an easier source code maintenance for the coming decades. In addition, these new technologies associated with a new architecture allow an easier implementation of additional physical models, and also easier and extended coupling capabilities with external tools.

The first official internal release of STORM was deployed in January 2021. Its entry into service was progressive, the time to train users, and also the time for users to accept a new tool. 18 months later, STORM is now used for all the new developments including City Airbus.

The novel perspectives offered by this new tool may allow increasing our prediction capabilities towards a more efficient design.

References

[1] **HOST, a General Helicopter Simulation Tool for Germany and France**, Bernard BENOIT, Wolfgang von GRÜNHAGEN, Konstantin KAMPA, Pierre-Marie BASSET, Bernard GIMONET; presented at the American Helicopter Society 56th Annual Forum, Virginia Beach, Virginia, May 2-4, 2000.

[2] **VTK, The Visualization Toolkit,** https://vtk.org

[3] **Qt, Cross-platform software development for embedded & desktop,** https://qt.io

[4] **PyQT,** https://riverbankcomputing.com/software/pyqt/intro