POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Model-Based Design of a Generic Automatic Flight Control System for Helicopter Flight Simulation Training Devices

Executive Summary - Laurea Magistrale in Aeronautical Engineering

Author: Francesco Paolo De Simone

Advisor: Prof. Marco Lovera

Co-Advisor: Eng. Francesco Borgatelli

Academic Year: 2021 - 2022

1 Introduction

By the end of 2039, the civil aviation industry will need to recruit roughly 2.4 million new pilots and technicians to cope with the increasing demand for air travel [2]. Flight simulators can play a crucial role in addressing this shortage by preparing pilots for the complexities of flying, providing a safe and cost-effective means of training.

TXT e-solutions S.p.A., a provider of engineering software solutions, has been a leading company in this field and has shown interest in using *Model-Based Design* to replace traditional *C++* hand-coding approach in the development of flight simulators.

The current objective, as established by the company, is to develop a modular and customizable autopilot architecture using the *Model-Based Design* approach. The intended outcome is the creation of a single, generic helicopter autopilot that can be adapted to various helicopter types by simply modifying configurations and parameters.

A proof-of-concept for low-level functionalities of a basic generic helicopter autopilot was previously developed in another work [5]. Instead, the present research aims to analyze and improve this model in order to demonstrate that the *Model-Based Design* approach is still valid, even when addressing more realistic and intricate autopilot architectures.

As a result, to deal with the increasing complexity of the new autopilot, it has been necessary to reconstruct the *Simulink* model from scratch since the initial configuration did not allow for sufficient modularity and generality to accommodate the high-level functionalities introduced by the upper modes and their control logics. A customizable auto-tuning process has also been developed to optimize the control systems gains of the assembled model of the generic autopilot. Finally, the results have been validated by comparing them with the ones achievable with a twin-engine light utility helicopter *Full Flight Simulator level D* (highest flight simulator certification recognition according to *EASA*).

2 AFCS Structure

In its most generic definition, an AFCS comprises of various hardware components such as sensors, actuators, computers, as well as software elements like control laws and logic. The sensors collect extensive data from both the helicopter and the surrounding environment and use these measurements to feed the control systems of the autopilot. Based on this data, the control logic and laws determine the most

suitable control inputs and direct them to the actuators, which adjust the flight controls of the helicopter accordingly. The main objective of an AFCS is to ensure the safe and efficient stabilization and control of an aircraft, while simultaneously minimizing the workload of the pilot.
The software components modeled in the current autopilot are presented below, considering the following AFCS framework:

- within the AFCS inner loop, the *lower modes* are responsible for controlling the pitch, roll, and yaw angular rates and attitude of the helicopter to enhance its maneuverability and stability.
  The *Stability Augmentation System* addresses and resolves the inherent instability of conventional helicopters by utilizing *Proportional* (or *Proportional-Integral*) controllers.
  On the other hand, the *Attitude Hold System* allows for the selection and automatic maintenance of a specific attitude. This system includes a logic module that sets the reference attitude based on the pilot's actions on the *force trim* and *beep trim* controls located on the cyclic and collective grips. The other module of this system computes the control action using PID controllers. In general, the *lower modes* provide control in the pitch, roll, and yaw channels of the AFCS.

- in the outer AFCS loop, the *upper modes* enable the control and hold of "external" conditions [4] by means of PID controllers. The most important among these systems are the *Indicated Airspeed Hold* (IAS), the *Barometric Altitude Hold* (ALT), *Radar Height Hold* (RHT), the *Heading Hold* (HDG) and the *Hover Hold* (HOV).
  The *upper modes* that control the pitch, roll, or yaw axes rely on the proper functioning of the ATT to perform their task successfully. However, this is not the case for the collective modes.
  The *autopilot control logics* are responsible for transitioning between modes on different axes, as well as for the manual and automatic engagement and disengagement decisions of these modes.
  The pilot can access these modes primarily through switches and buttons on the *Autopilot Control Panel* (APCP). Adjustments in the reference datum of a specific *upper mode* can be made using the *force trim* and *beep trim* controls located on the stick grips.

- The *functionalities* also operate within the inner (or outer) loop of the AFCS. These systems offer various types of automatic controls, which aim to improve the controllability, stability, and maneuverability of the helicopter. The unique feature of these systems is that they only provide automatic control actions in specific flight conditions that are characteristic of the selected *functionality*. Among these systems there is the *Turn Coordinator* (TC), which aims to ensure *coordinated turns* by utilizing a PID controller that annihilates lateral accelerations during specific flight operations.

3 AFCS Modeling

To achieve high levels of modularity, generality, and automation in the development of the AFCS, *Simulink* offers several solutions, with the most significant ones in this work being:

1. *Custom Libraries* used to store multiples modular custom blocks of the AFCS which are reused several times throughout the assembly of the final model.

2. *Block Masks* are essential to achieve high levels of system generality. They are custom GUIs that can be added to a subsystem made up of different blocks, allowing end-users to customize the structure, behavior, and parameters within the masked block. Proper utilization of the *Initialization* interface within the *Mask Editor* enables upstream customization and recycling of the same masked block for multiple uses. However, for complex and hierarchical structures like the one present in this AFCS, it is necessary to define a clear and unambiguous notation for managing the multiple functions of a single custom block.

3. *Stateflow Charts* which provide a graphical programming environment based on finite state machines, essential for the definition of generic control logic of the AFCS modes. *Stateflow blocks* can also be employed for the modeling and execution of custom *transfer functions* by means of personalized numerical integration algorithms [5].

4. *Code Generation Tools* which are basically the primary reason for switching from traditional hands-on programming to *Model-Based Design*. The *Simulink* AFCS model under development must be suitable for being exportable in *C++* by means of *Embedded Coder*. However, code compiling errors arise if the *simulation timestep* is defined as a *tunable* parameter [5].
That's the reason why most of the standard *Simulink* blocks concerning control system development (e.g. *PID controllers* and *transfer functions*) must be recreated from scratch: the issue is that they employ the "non-tunable" *simulation timestep* to perform integrations. Therefore, a practical solution involves, as already said, the use of *Stateflow blocks* to initialize the structure of the *transfer function* (example application in Fig.1) and the employment of *MATLAB Functions* to implement a custom numerical integration scheme such as *Runge-Kutta 4 algorithm* (Fig.2). To maintain the generality of the custom block created, it is necessary to initialize the structure of the *transfer function coefficients* and its corresponding *state-space* in the *Stateflow block*, as shown in Fig.1. By doing so, the tunability of the custom block is preserved, as coefficients (in Fig.1, *kind* and *Ti*) are directly extracted from the values specified in the block's mask. This approach enables the creation of custom *PID controllers* of various types (P-PI-PD-PID), which are essential components of the current AFCS control systems.

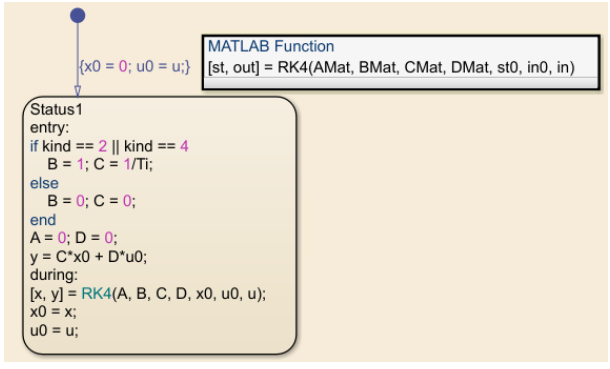

Figure 1: Initialization of the Integrator Block of the Custom PID Controller

```
function [st, out] = RK4(AMat, BMat, CMat, DMat, st0, in0, in)

inmed = (in0 + in)/2; % mid-steptime input

k1 = AMat*st0 + BMat*in0;
k2 = AMat*(st0 + k1*dt/2) + BMat*inmed;
k3 = AMat*(st0 + k2*dt/2) + BMat*inmed;
k4 = AMat*(st0 + k3*dt) + BMat*in;

st = st0 + dt/6*(k1 + 2*k2 + 2*k3 + k4); % state approximation
out = CMat*st + DMat*in; % output evaluation

end
```

Figure 2: Runge-Kutta 4 algorithm

Using the above-mentioned means, a personal *Custom Library* has been created containing all the modular custom blocks required to construct a complete generic autopilot. Indeed, every of these blocks is equipped with a mask which allows to specify customisable parameters in input.
Among these, two blocks are the real core of this generic autopilot, as they allow, adapting their mask for the specific customisation, to retrieve the structure of the ATT for pitch, roll and yaw axis as well as the structure of every modeled *upper mode*, namely the IAS, HDG, ALT (both on pitch and collective axes), RHT and HOV (both on pitch and roll axes). Those blocks are:

1. the *Mode Setpoint Block*, which allows to evaluate, based on *force trim* and *beep trim* functioning, the reference values assigned to the above-mentioned modes at each *simulation timestep*.

2. the *Mode δCommand/δSetpoint block*, which is used for two distinct and independent functions throughout the AFCS development.
The first possible task achieved by this block is the *δCommand* evaluation, which is necessary for the ATT and for the collective modes to provide the necessary control action to their *series actuators*.
The second one instead, is the *δSetpoint*

function, which is necessary for the *upper modes* on the pitch, roll and yaw axes to assess the reference attitude variation to provide to the ATT.

Along with these principal custom blocks, other relevant ones are the *Modes Logics Stateflow*, which account for engagement, disengagement and transitions of *upper modes*, the *PID Controller (2DOF) Block*, the *Clamping Anti-Windup Block*, the *SAS Block*, for the development of *Stability Augmentation System* on the pitch, roll and yaw axes, etc.
Therefore, by correctly parameterizing these blocks, it is possible to obtain the complete AFCS *Simulink* model. Figure 3 displays the highest hierarchical layer of this model, in which the AFCS overall structure is encircled in red, while other components such as the *state-space* representation of the helicopter dynamics are utilized for tuning purposes.

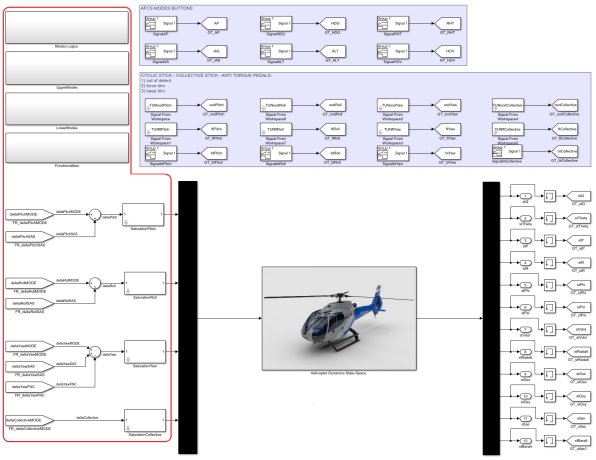

Figure 3: AFCS Tuning Model

4 AFCS Tuning

In traditional programming, tuning a control system can be a challenging and time consuming task. On the other hand, in *Model-Based Design*, the user hold a mathematical representation of the plant, derived from first principles or by data-driven techniques. Therefore, *Model-based tuning* can be applied, allowing for more efficient and automatic techniques to tune a control system.
Simulink offers numerous interactive interfaces to tune automatically a control system such as the *Control System Tuner*; however, due to the usage of custom PID controllers, direct tuning in *Simulink* is not feasible as numerous limitations arise. Since at the command-line those limitations fade away, an external autotuning *Matlab* script is generated with the only purpose to find the optimal gains of the AFCS *Simulink* model control systems.
Also in this case, being the aim of the present work the development of a generic AFCS, it is essential that the defined tuning process also allows for customisation in terms of requirements and objectives, which must be defined according to the specific helicopter and the performance of the control system required.
The autotuning process defined relies on the *Matlab* function *systune*. This command tunes the controller parameters optimizing the H_∞ norm across a closed-loop system; in practice, this function applies the structured H_∞ synthesis, which in contrast with traditional H_∞ synthesis allows to define a fixed-structure on the controller, which in the present AFCS are mostly 2DOF PID controllers. In addition, *systune* enables to define the number and configuration of feedback loops, as well as it allows the specification of the parameterisation of each tunable component, and the incorporation of multiple requirements on distinct closed-loop *transfer functions* [1].
To assess the reliability of the generic autotuning process defined in the *Matlab* script and to test the tuning workflow, a twin-engine light utility helicopter's linear dynamics model is used as a case study.
The linear model, which is provided by *TXT e-solutions*, is in a trim condition that corresponds to a steady level flight configuration at a *Barometric Altitude* of 5926 $[ft]$ and *Indicated Airspeed* of 68 $[kt]$. The matrix dimensions and properties are presented below:

$$\begin{cases} \dot{x} = Ax + Bu, & A \in \mathbb{R}^{39\times39}, \quad B \in \mathbb{R}^{39\times4} \\ y = Cx + Du, & C \in \mathbb{R}^{12\times39}, \quad D \in \mathbb{R}^{12\times4} \end{cases}$$

Matrix Inputs		
Input Name	**UoM**	**Description**
Longitudinal Cyclic	[-]	-1= Fully Forward; +1=Fully Backward
Lateral Cyclic	[-]	-1= Fully Left; +1=Fully Right
Pedals	[-]	-1= Fully Left; +1=Fully Right
Collective	[-]	-1= Fully Down; +1=Fully Up

Figure 4: Linearized System Inputs

The AFCS is set up in a multiloop feedback configuration, where each mode has its own tunable

Matrix Outputs			
Parameter Name	**Symbol**	**UoM**	**Reference Frame**
Pitch Rate	q	[rad/s]	body
Pitch Angle	θ	[rad]	inertial
Roll Rate	p	[rad/s]	body
Yaw Rate	r	[rad/s]	body
Bank Angle	ϕ	[rad]	inertial
Heading Angle	ψ	[rad]	inertial
CG Y-Acceleration	A_y	[G]	body
Radar Height	$Radalt$	[m]	
Longitudinal Groundspeed	GS_x	[m/s]	inertial
Lateral Groundspeed	GS_y	[m/s]	inertial
Indicated Airspeed	Ias	[m/s]	
Barometric Altitude	$Baralt$	[m]	

Figure 5: Linearized System Outputs

controller gains. However, the tuning process cannot be completed in a single optimization phase, and multiple tuning optimizations must be set and run independently. It is important to note that the tuning processes that can be defined depend greatly on the operating point at which the linear model was created, as control logics may allow or disallow the engagement of a mode and hence the tuning of its control system.

For the case study, several tuning processes were created, starting obviously with the tuning of the *lower modes* and then proceeding to the *upper modes*' control systems. Various tuning goals were implemented to meet the specific requirements of each mode's task. However, three broad types of tuning processes were employed:

- The first loop to be tuned is the most internal one and the only one that can operate when both helicopter autopilots fail. This process involves tuning the SAS on the pitch, roll, and yaw axes. The optimization process is enforced by ensuring that the closed-loop dynamics become stable, as the open-loop system is *unstable*. This is achieved through the use of the *Matlab* command *TuningGoal.Poles*. The result of this tuning process is presented in Fig. 6.

- Once the gains for the SAS PI controllers are found, the next step involves the tuning of the ATT, which is the other *lower mode* of the the current AFCS. In this case, due to the case-study trim condition, this tuning process involves the ATT on the pitch and roll axes and the TC, which is functioning on the yaw axis.
  The requirements for the first two systems are expressed using the *Matlab* command

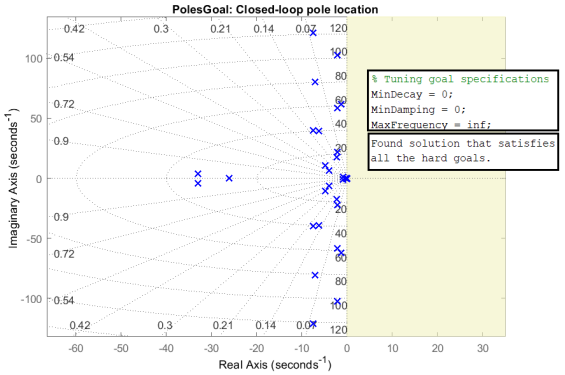

Figure 6: Tuning SAS: Closed-Loop Poles

TuningGoal.StepTracking, while for the third, *TuningGoal.StepRejection* is used, as the TC must be able to repel the lateral accelerations that develop during roll manoeuvres. Results for the ATT on the pitch axis and for TC are respectively shown in Fig. 7 and 8.

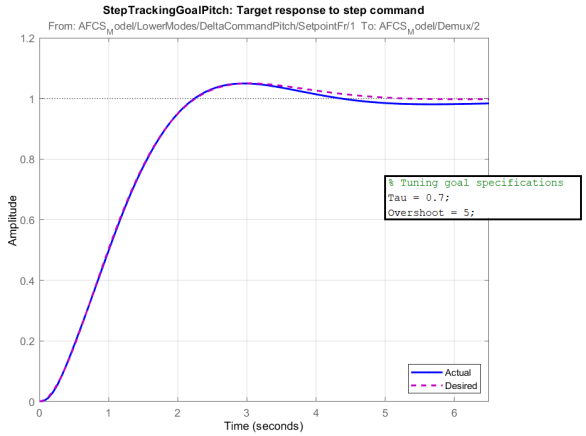

Figure 7: Tuning ATT Pitch, ATT Roll and TC: θ_{ref} vs θ

- Finally, the third and last category of tuning processes is focused on optimizing the gains of the *upper modes* controllers, which mostly rely on the ATT to function. Therefore, optimal gains from the previous tuning process are required to carry out these ones.
  However, not all the *upper modes* may be tuned due to the specific trim point of the case-study dynamics. Generally, one tuning process is necessary for each *upper mode*, while two separate tuning processes

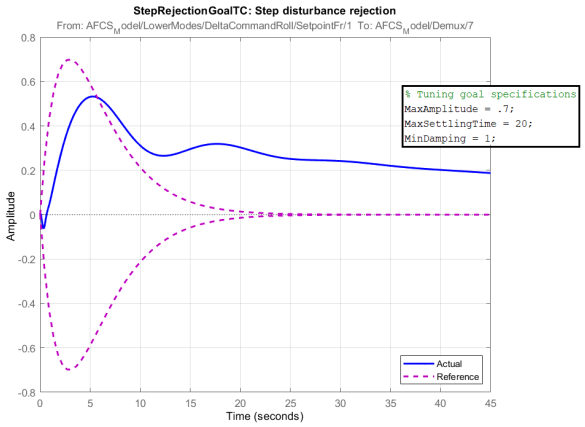

Figure 8: Tuning ATT Pitch, ATT Roll and TC: A_{yref} vs A_y

are required for modes that can operate on different axes (e.g. ALT).
Requirements, also in this case, are set using *TuningGoal.StepTracking*, as the task accomplished by an ATT is not different in practice from that of an *upper mode*: they both need to track a reference datum. Fig. 9 provides an example of such tuning processes.

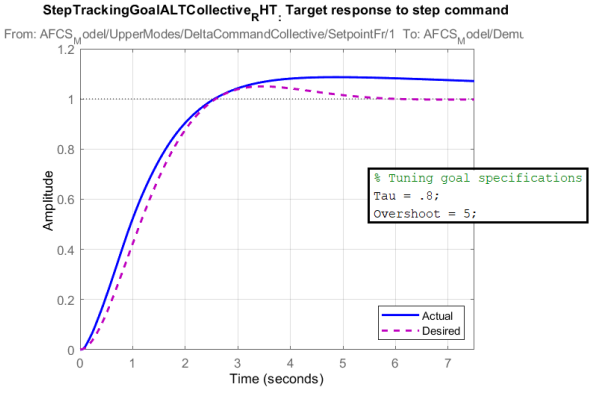

Figure 9: Tuning ALT Collective: $Baralt_{ref}$ vs $Baralt$

5 AFCS Validation

In the so-called *V-Model*, the missing passages to complete the *Simulink Model-Based Design* of the current AFCS are related to the *Code-Generation*, *Integration* and *Validation* phases. However, in the previous work [5], the proofs of concept of the first two phases have already met satisfying results, and since in the current work the design methodology and modeling philosophy have remained unchanged, those phases can be taken as granted. However, the *Validation* phase must be obviously carried out as the model has been completely re-built and re-organized from scratch with the addition of multiple new features.
Given the complexity of the model, three different verification tests have been employed to validate the entire autopilot modelling, both from the point of view of control logic and physical modelling, and from the point of view of tuning workflow effectiveness:

- the first test aims to validate the correct functionality of the *upper modes* control logics modeled as state machines in a *Stateflow block*.
  By means of *Signal Builder Blocks* have been generated test cases that cover all possible states and transitions of the state machine; later, the outcomes of these tests have been carefully verified in order to assess that the output signals matched the expected ones. An example of these tests outcomes is shown in fig. 10

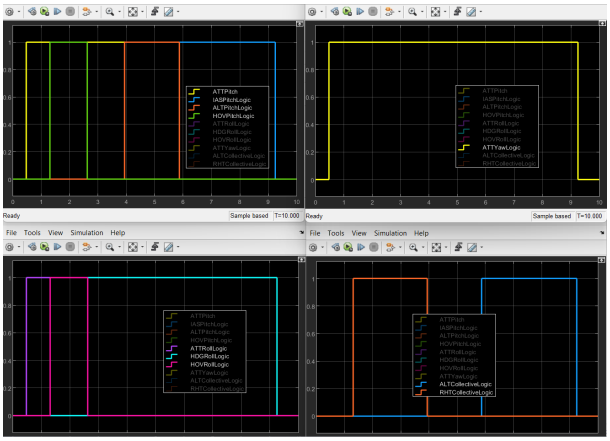

Figure 10: Engagement of Modes in Pitch, Roll, Yaw and Collective axis during the Simulation of a Control Logics Test

Numerous tests have been carried out varying numerous flight conditions and triggering the activation of different combination of *upper modes*.
Since at the end of the process each test was successful, the *upper modes* control logics are resulted robust enough to be validated.

- The second test was conducted to assess the proper functioning of the *Stability Augmentation Systems* alone.
  The tuning model depicted in figure 3 was modified to include pilot inputs in the form of longitudinal and lateral cyclic, pedals, and collective controls. The pilots inputs supplied to the *Simulink* model are the same gathered from real flight tests of the same helicopter used for the tuning of the current AFCS model. Those flight tests were used for certification purposes during the assessment of *EASA CS-FSTD(H) 2.c.(1)* test and *2.d.(1)(i)* test [3].
  Therefore, in *Simulink*, providing these pilot inputs with the only SAS active, was possible to demonstrate that the results obtained within this *Simulink* model comply with the tolerances imposed for these certification tests. Figure 11 shows one of the responses obtained by comparing flight test data and the *Simulink* AFCS.

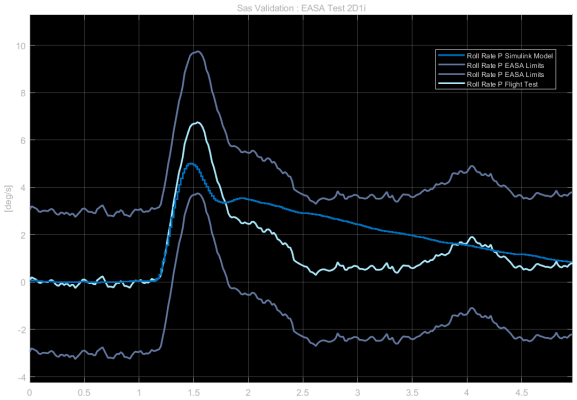

Figure 11: Test *2.d.(1)(i)*: Flight Test and Simulated Roll Rate Responses with SAS on

- the third and final validation analysis conducted is employed to prove that the tuned control systems of the ATT and of the *upper modes* are capable of producing similar responses to those achievable with a certified *Full Flight Simulator (FFS) level D* .
  For this phase, *TXT e-solutions* provided the possibility to log data directly from the existing FFS of the same helicopter used to tune the AFCS *Simulink* model.
  Therefore, on the FFS were generated multiple real-time simulations in operating flight conditions similar to the one used to obtain the linearized model. In each of those tests, a different mode is engaged via APCP and is subject to *beep trim* excitements.
  Once inputs and outputs data are logged from each FFS simulations, the same tests are built in the *Simulink* model and finally outputs from the custom AFCS and from the FFS are compared.
  An example of these results is depicted in figure 12: in this test the ALT mode rightly transitions its control action from the pitch to the collective axis and is still able to generate an effective and smooth following of the reference with a shorter *settling time* with respect to the FFS.

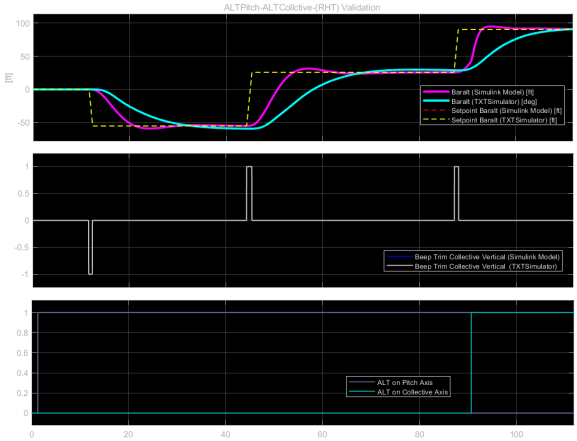

Figure 12: Validation of the ALT Collective and ALT Pitch

6 Conclusions and Future Developments

In conclusion, the aim of this work was to investigate the potential of the *Model-Based Design* approach in the creation of a generic helicopter automatic flight control system that guarantees modularity, customisation, tunability and exportability of the code.
At the end of this work, the *Model-Based Design* proved to be robust and appropriate enough to deal with complex autopilot structures equipped with *lower modes*, *upper modes*, *functionalities* and control logics.
Therefore, the generic autopilot architecture developed in this thesis can be used as a starting point for future research in this topic, and

the knowledge gained from this study can be applied to develop new AFCS *upper modes*, such as the *Vertical Speed Mode* (VS) or the *Go Around Mode* (GA), or most importantly to introduce a new category of modes, namely the *Flight Director Modes* such as *En-Route Navigation Mode* (NAV), *Localizer Mode* (LOC) and *Glideslope Mode* (GS).

7 Acknowledgments

The author expresses gratitude to all the individuals who provided support in various ways towards the completion of this work. Special thanks are extended to Prof. Marco Lovera, Eng. Francesco Borgatelli, and Eng. Gennaro Romagnoli for their significant contributions.

References

[1] P. Apkarian and D. Noll. *Nonsmooth H-infinity Synthesis*. IEEE Transactions on Automatic Control, Vol. 51, Number 1, 2006.

[2] Boeing. *Boeing Pilot & Technician Outlook 2020-2039*. 2020.

[3] EASA. *CS-FSTD(H) (Initial issue)*. 2020.

[4] E. Pallett. *Automatic Flight Control*. Blackwwell Publishing, 1993.

[5] G. Romagnoli. *Model-Based Design of a Generic Autopilot System for Helicopter Flight Simulators in Simulink*. 2021.