Automating the Handling Qualities Predictions of the Flying-V

By

Adam Świderski
Track: Control and Operations

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended on 08.03.2023

Supervisors:
Ir. O. Stroosma, TU Delft
Prof. Dr. Ir. M. Mulder, TU Delft

TU Delft

Automating the Handling Qualities Predictions of the Flying-V

Adam Świderski *
Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands

As the Flying-V goes through subsequent design iterations, each new model requires dedicated handling qualities research. A part of that work, namely obtaining the handling qualities predictions, is highly repetitive and can be automated. To address that issue, a Python library for automated handling qualities prediction evaluations of the Flying-V was developed. The library's functionalities cover model implementation, trim and linearization, model reductions, identification of the modal parameters, and finally, the evaluation of the requirements. The library takes a semi 'black-box' approach toward the model structure, relying on defining it through a set of functions instead of parameters. With that, it can accommodate various sources and formulations of the model. The library was successfully validated on two different aircraft models. Part of the validation was replicating some of the previous Flying-V handling qualities research data. It is expected that the use of the library will bring time savings to future handling qualities research of the Flying-V, as well as reduce the risk of mistakes in the process.

I. Introduction

Flying-V [1–3] is a new flying wing concept aircraft currently developed at TU Delft. It is a tailless, V-shaped aircraft dedicated to long-haul operations – seating between 293 and 361 passengers and with a design range between 11,200 km to 15,400 km (based on the family design by Oosterom and Vos [3]). In the Flying-V, the cabin, cargo holds, and fuel tanks are contained within one highly-swept wing structure. Initial designs claim that the Flying-V will have a better lift-to-drag ratio and smaller fuel consumption compared to state-of-the-art aircraft of that size [1–3] and can be seen as a new step towards more sustainable aviation.

One of the areas of research within the Flying-V is studying its handling qualities. Those are the assessments of how easy the aircraft is to fly and how well the pilot can perform specific tasks with it [4–6]. Good handling qualities are critical for every aircraft, but special attention is devoted to Flying-V due to its unusual shape. Handling qualities can be measured in two ways: either by performing test flights and gathering pilot opinions [4] (a subjective assessment) or by using handling qualities predictions [7, 8] (an objective assessment) – various formulae and criteria which, using the aircraft data, try to estimate the handling qualities level, as it would be perceived by the pilot.

In recent years, several research experiments into the handling qualities of the Flying-V were conducted [9–13], which addressed different iterations of the aircraft or flight conditions. The goal of these efforts was, among others, to provide feedback to the designers of the Flying-V, which would influence the subsequent iterations of the aircraft. The works covered the preliminary offline analysis of the model, arriving at handling qualities predictions. Some also included piloted flight simulation experiments to obtain pilot ratings [11–13]. The Flying-V can be expected to go through many design iterations in the future, each of which will require its own handling qualities research. This brings the question of whether the process can be automated, as so far, the works have been done independently. Therefore, this study was commenced into exactly which parts of this process can be automated and how.

By reviewing the previous works and the steps taken in each of them, it was established that the preparation and execution of piloted experiments in the simulator would be very difficult to automate in a way that would bring meaningful time savings. This is because those steps either cannot physically be automated (collecting pilot opinions), require creative input (designing a task to test a specific hypothesis), or expert knowledge (tuning the motion cueing systems). On the other hand, offline analysis, i.e., evaluating handling qualities predictions, could potentially be highly automated. This is because these steps are highly algorithmic, their results are very predictable, and they require almost no decision-making.

Therefore, a Python library was developed to cover all operations that have to be made to obtain the handling qualities predictions starting from the aircraft model. The goal is to provide a flexible framework for future Flying-V handling qualities research, which would accommodate potentially different types of aircraft models, contribute to substantial time savings, and minimize the risk of implementation mistakes. Handling qualities research is part of the

*MSc. student, Control & Simulation section, Faculty of Aerospace Engineering.

aircraft design lifecycle [14], so fast and effective handling qualities predictions should positively impact the entire project.

The aim of this paper is to describe the design of the library and how each step necessary to arrive at handling qualities predictions can be automated. It highlights the model implementation framework – its capabilities and assumptions. For this library to be a useful tool in future research, it needs to be thoroughly tested in order to ensure that it operates properly. Therefore, the later part of the paper also outlines the extent of the validation efforts taken during this project.

II. Background

The purpose of this section is not only to provide the relevant background to the topics addressed in this paper but also present information that informed the design decisions of the library. Firstly, a brief introduction to handling qualities will be presented, followed by a review of some of the software tools for preliminary analysis of aircraft designs. Finally, the recent handling qualities research into the Flying-V will be discussed.

A. Handling qualities

In general, *handling qualities* can be defined as characteristics of an aircraft that influence the ease and safety of flying. The specific definitions vary slightly between authors. Firstly, different elements are considered to be contributing to handling qualities. Ashkenas [5] includes "dynamic and static properties", Phillips [6] similarly refers to "stability and control characteristics", but Cooper and Harper [4] also include other factors, such as "the cockpit interface (e.g., displays, controls), the aircraft environment (e.g., weather conditions, visibility, turbulence) and stress". The impact of handling qualities is also understood a bit differently between authors. For Cooper and Harper, it is to "perform the tasks required in support of an aircraft role", for Ashkenas to " fully exploit its performance and other potential in a variety of mission and roles", and for Phillips handling qualities have "important bearing on the safety of flight and other pilots' impressions of the ease of flying an airplane in steady flight". In the context of this work, which focuses on handling qualities predictions, the source of handling qualities will be limited to stability and control parameters. The interpretation of the impact of the handling qualities while using the predictions is not up to their users. Instead, it is decided by whoever designs a particular prediction criterion.

Pilot opinions are the basis of handling qualities evaluations. They can be just simple comments. Hodgkinson [15, p.4] claims that "[t]he best way to collect a pilot's opinion is to ask him or her". They can also be quantified, like in the ubiquitous Cooper-Harper rating scale [4], which helps the pilot to translate their experience into a 10-point scale. Of course, conducting piloted experiments can be cost and time-consuming, so naturally, one would like to be able to *predict* the handling qualities from the aircraft model. This led to establishing the handling qualities *requirements* or *predictions*, arguably the most popular collection of which being the military specification for "Flying Qualities of Piloted Aircraft", also known as MIL-STD-1797 or MIL-HDBK-1797 [7]. There are also other sources on handling qualities requirements, like ESDU 92006 [8], published by ESDU, a British engineering advisory organization, or the book "Aircraft Handling Qualities" by John Hodgkinson [15], which provides an exhaustive overview of aircraft stability, handling qualities and their assessments. However, MIL-STD-1797 is by far the most comprehensive one, supplemented with supporting research data, and almost all of the requirements in the other two sources are also present in MIL-STD-1797.

The great majority of the handling qualities requirements are intended for linear aircraft models. The most commonly used ones specifically address modal parameters of the aircraft, e.g., the damping ratio of the phugoid mode, by providing allowable ranges of the parameters or functions thereof. With some exceptions, the output of the requirement is the handling qualities level. Level 1 means satisfactory handling qualities, Level 2 – acceptable, and Level 3 – controllable. It is also possible to score no level, indicating an uncontrollable aircraft. The requirements will often have different limits based on the flight phase and aircraft class (Flying-V is a Class III aircraft – "Large, heavy. low-to-medium maneuverability" [7])

There is not much mention of handling qualities in the civil requirements. For example, Aircraft Certification Specification CS25 [16] only mentions it briefly, including non-specific qualitative terms like "Any short period oscillation, not including combined lateral directional oscillations ... must be heavily damped with the primary controls" (CS 25.181 Dynamic Stability [16]).

B. Existing tools for evaluations of preliminary aircraft designs

Being able to accurately predict aircraft characteristics, such as handling qualities, in the earliest stages of the design is critical for effective and fast development. Rizzi [14] claims that "80% of the life-cycle cost of an aircraft is incurred by decisions taken during the conceptual design phase" and highlights that the mistakes made during that stage will be very costly to fix in the future. This is why many software solutions were developed to evaluate certain aspects of preliminary aircraft designs. A recent article by Marco et al. [17] provides an overview of the most popular products. Reviewing these programs, as well as other available examples, it is clear that a bigger focus is put on aircraft performance than on handling qualities. Nevertheless, the latter are addressed in several instances, and the most relevant of those will be discussed in this subsection.

CONDUIT (Control Designer's Unified Interface) [18] is a tool for aircraft and rotorcraft analysis focused on evaluating flight control system designs and providing optimization capabilities. The user implements their aircraft as a Simulink model, and the CONDUIT can attach to the model and evaluate a set of characteristics and requirements. Those requirements can also serve as constraints in the design tuning. The user specifies which parameters of their model are tunable, and the CONDUIT optimizes them by minimizing either the actuator energy or feedback-loop crossover frequency. For handling qualities requirements, the CONDUIT uses the MIL-STD-1797A.

HAREM (HAndling Qualities Research and Evaluation using MATLAB) [19] developed at DLR is a MATLAB-based command-line tool that automates evaluation of the relevant handling qualities requirements from MIL-STD-1797A. The need for automation arises from the fact that to evaluate an aircraft "within the entire flight envelope often several thousands of configurations ... have to be investigated" [19]. The program evaluates handling qualities and outputs the data in the form of plots, tables, or binary files. HAREM also contains a database of various aircraft data that can be used with the tool.

MAPET (Model based Aircraft Performance Evaluation Tool) [20] is a performance assessment tool developed in MATLAB/Simulink. It bases its results on trim calculations for ranges of mass, altitudes, and speeds. It then uses analytical relations (often called "handbook methods") to estimate values related to, among others, flight endurance and range, climb, descent, and level turn. Its next iteration was the MAPET II [21], which focuses on assessing the take-off and landing performances. In this instance, the authors recognize that, since those are not steady flight phases, trim calculations cannot be used. Instead, the relevant maneuvers are simulated in MATLAB/Simulink.

MITRA [22] automates the evaluation of handling qualities and performance parameters based on simulation data. It serves as an interface to the Simulink simulation, setting up and running scenarios and then analyzing the obtained data. MITRA, among others, evaluates take-off, landing, and climb performance, many handling qualities criteria (from MIL-STD-1797), as well as reference speeds.

SimSAC (Simulating Aircraft Stability And Control Characteristics for Use in Conceptual Design) project was commenced to improve the ways the aircraft designs are evaluated at their conceptual phases [14]. One of the outcomes of this project was the development of the CAESIOM Tool. It covers a wide area of aircraft design activities, which are realized using various modules, including aircraft geometry, aerodynamic identification, flight control design, and stability and control. One of those modules is the SDSA (Simulation and Dynamic Stability Analysis) tool [23]. The program performs stability analysis using both the eigenmodes of the linearized aircraft model and the time histories of the non-linear 6DOF simulation. Furthermore, the tool features actuator models, a human pilot model (for pilot-in-the-loop simulations), and a simple stability augmentation which can be used with a Linear Quadratic Regulator (LQR). In the SDSA, the aircraft model is defined through a file containing aerodynamic derivatives.

The desire to obtain handling qualities in the early design stages is also present in rotorcraft development. Examples of that can be the tools developed by Lawrence et al. [24] or Zanoni et al. [25]. They both use the existing NDARC (NASA Design and Analysis of Rotorcraft) software tool and integrate it with handling qualities predictions: Lawrence et al. utilize the aforementioned CONDUIT, while Zanoni et al. implement the evaluations themselves.

The presented examples highlight the desire to evaluate the handling qualities predictions for early aircraft (and rotorcraft) designs. The software tools can either be a part of a larger design pipeline (e.g., SDSA[23]) or a standalone tool (e.g., MITRA[22]). While the specifics of model formulation were not always disclosed in the referenced papers, one could observe different approaches to that subject. In some cases, the model structure was constrained, and the user has only to provide a set of parameters (e.g., SDSA), or the user has to implement the entire model by themselves (e.g., CONDUIT[18]). The source of handling qualities requirements, if referenced, was always the MIL-STD-1797. Lastly, the discussed examples showcase the close coupling between handling qualities evaluations and the flight control system design.

C. The Flying-V

The design was first proposed by Justus Benad in 2015 as a result of his thesis project at Airbus in Hamburg [1]. The initial design claimed a 10% higher lift-to-drag ratio and 2% lower empty weight over the state-of-the-art Airbus A350-900. The work on the project subsequently continued at TU Delft. In 2017 Faggiano et al. [2] proposed an improved design. Their CFD (Computational Fluid Dynamics) analyses showed a 25% improvement in the lift-to-drag ratio over a reference aircraft (based on NASA Common Research Model). In 2022, a conceptual design of the aircraft family was published [3], which includes three variants with different passenger capacities and ranges. Due to its lack of a tail, the Flying-V does not have a classical flight control configuration. Instead, the pitch and roll authority is shared between the elevons located at the trailing edges. The yaw control is achieved through rudders located on the winglets.

In 2019, Cappuyns [9] performed a stability and control analysis on that design and found unstable Dutch Roll and limited lateral-directional controllability with one engine inoperative, and proposed further improvements to the design. Overeem et al. [10] combined the full-scale Flying-V model, obtained using the Vortex Lattice Method, with results of wind tunnel tests of the small-scale model and analyzed handling qualities requirements on that model. The short-period characteristics were good, but on the approach the phugoid and Dutch Roll became unstable. Furthermore, they found that the aircraft becomes statically unstable for large angles of attack.

Most recently, several piloted (in a ground simulator) handling qualities experiments of Cappuyns' design were conducted. Vugts et al. [11] analyzed the longitudinal channel in cruise. A new controlled allocation scheme was proposed to alleviate the non-minimum phase behavior of the flight path angle response to stick step input, which stemmed from a preliminary offline analysis. The pilots deemed the handling qualities with respect to the pitch angle response (while performing a pitch angle tracking task) good. They did not notice the non-minimum phase behavior while performing the flight path tracking task but still performed better under the new control allocation.

Joosten et al. [12] looked at the lateral-directional handling qualities and found insufficient control authority for slow speeds, which was confirmed in a piloted experiment. Besides, the Dutch Roll was found to be unstable for all analyzed flight conditions, although controllable for some (higher) speeds. To address these issues, a generalized-inverse control allocation and a stability augmentation system were implemented. Those were shown to have a positive effect on handling qualities, although the problem with reaching actuator limits persisted, and some indications of pilot-induced oscillations were noticed, which were attributed to actuator rate limits.

Torelli et al. [13] analyzed the longitudinal handling qualities at approach and found an overdamped short period that led to the aircraft's sluggish response, which was both yielded by the preliminary results and indicated in the pilot's comments. Despite that, both the handling qualities predictions and pilot opinion ratings showed acceptable handling qualities. A pitch rate controller was implemented and tuned based on the handling qualities requirements, and improvement was noticed by all pilots. In this experiment as well, problems with control saturation were observed.

Many steps were recurring in the previous works regarding the handling qualities predictions. The aircraft model first needed to be implemented. The authors' starting points were most often the stability and control derivatives, which had to be turned into force and moment equations and supplemented with kinematics, and sometimes actuator dynamics, to form a system of (non-linear) differential equations that describe the aircraft's motion. Next, a trim point had to be found – firstly, to investigate the aircraft's configuration (angle of attack, control surface deflections, etc.) for certain flight phases, and secondly, to be able to linearize the system. A linear system was used to evaluate some of the handling qualities requirements and, in some cases, for flight control design. The handling qualities requirements came predominantly from MIL-STD-1797. While the research into longitudinal handling qualities featured some non-modal requirements, such as bandwidth or Gibson dropback criteria, the lateral-directional part relied solely on modal requirements. The handling qualities research was also accompanied by attempts to fix the poor handling qualities with a flight control system.

III. Design

A. Introduction

This section presents an overview of the design of the library, outlining how the evaluation of the handling qualities predictions of the Flying-V can be automated. However, before specific solutions are discussed, two important philosophies that drove the design have to be introduced:

1) Reduced assumptions about the aircraft model structure – The library will not assume that a pre-determined set of stability and control derivatives will represent the model. This implies no explicit, analytical solutions to the equations of motion nor their linearizations. Furthermore, the handling qualities criteria will be evaluated

based on the numerical linearization of the black-box model and not on specific flight dynamics parameters. The library was designed as a stand-alone tool. That does not preclude future integration with other software but avoids basing the design on a specific output format of a different tool, which, if replaced, might require redesigning this library.

2) Modularity – Instead of a single program/function running the entire evaluation, the library is designed as a set of modules. This affords greater flexibility, covers more use cases, and facilitates easier expansion.

A review of the previous handling qualities research into the Flying-V and the insight from similar software solutions allowed to establish the desired functionalities of the library. They cover all steps that need to be taken to obtain the handling qualities predictions, starting from the non-linear aircraft model. Those functionalities are:

- providing a framework in which the users can implement their aircraft model,
- trimming the aircraft model,
- linearizing the aircraft model,
- obtaining modal parameters required for most of the handling qualities criteria from the linear aircraft model,
- evaluating the selected criteria,
- attempting to improve the undesirable handling qualities criteria with a Flight Control System (FCS).

The presented functionalities and their division at the same time define the library's modules. A description of each of them and how each step is automated will be shown in the rest of this section.

The library implementation heavily relies on the python-control library [26]. The library is used to create and handle non-linear and linear systems and perform related operations, such as finding equilibrium points, linearization, or simulating time responses. The library also utilizes commonly used scientific packages such as NumPy[27] and SciPy[28].

B. Model implementation

The model implementation framework follows a semi 'black-box' approach toward the model formulation. The top-level structure of the model is defined, and the user implements the aircraft through a set of functions that describe its specific elements (e.g., control allocation).

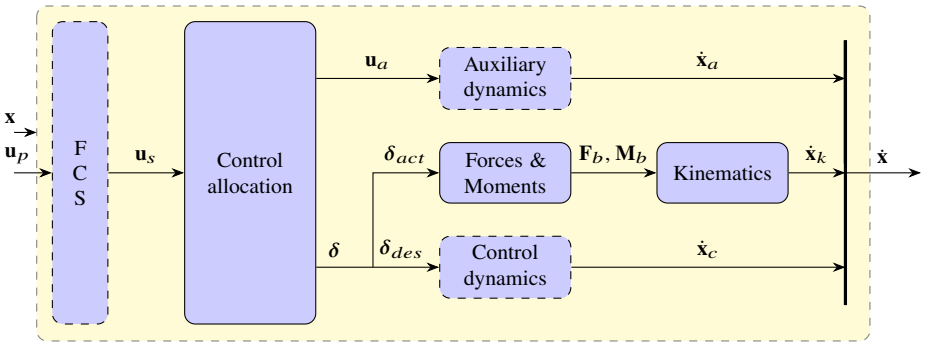


Fig. 1 Right-hand-side function structure for the Flying-V model. Dashed functions are optional. The default implementation of the Kinematics function is provided by the library and does not have to be specified by the user. The state vector x is available for all the functions inside.

The main part of the aircraft model is its right-hand-side (RHS) function of a system of first-order non-linear differential equations. This function can be seen in Figure 1. It is composed of smaller functions that describe elements of the model. Those functions are provided by the user, and the library manages the data flow between them. The library supplies the kinematic equations, so the user does not have to provide them (although they can if they wish). The built-in equations work under the assumption of a rigid aircraft with constant mass and flat and non-rotating Earth – sufficient for handling qualities research.

The inputs to the system are the "pilot inputs", which are two throttle positions (or thrust) and three stick/pedals controls for roll, pitch, and yaw. The state vector is comprised of three parts: a kinematic state vector, an auxiliary state vector containing additional states introduced by the user, and a control state vector containing control surface deflections and thrust when control dynamics are used.

The kinematic state vector has 12 states: three positions (North, East, altitude), three Euler angles (roll, pitch, yaw), three velocity components, and three body rates. It is the most common set of kinematic states used in aircraft modeling, and there is hardly any variation except for velocity representation. Aircraft's velocity can be represented in two ways: either through total velocity V, angle of attack α and sideslip angle β – the triad is sometimes called the *aerodynamic velocity* – or through u, v, w *body velocities*. The model structure currently does not include any wind input. Therefore, those representations are equivalent. Because the two are commonly used, the user will be able to use either of them.

Following the order of operation in the RHS function (Figure 1), firstly, the user may define a *Flight Control System* function if they wish to evaluate a closed-loop system. If this function is skipped, then the pilot inputs are fed directly to the *Control Allocation* function. This function is mandatory because the Flying-V has a set of elevons that control both the pitch and the roll, as well as two rudders. Another obligatory function to implement is the *Forces and Moments* function. This is the core function of the model, which calculates the force and moment vectors in the body frame based on the current state and inputs. The aircraft model should be explicit, meaning that the forces and moments must be expressed exclusively in terms of inputs and states.

The user may choose to include the dynamics of the actuators and thrust in the model. If they do so, they must provide the *Control Dynamics* function, which returns the derivatives of the control states. If the control dynamics are disabled, then the control deflections are not states, and the outputs of *Control Allocation* are directly fed to the *Forces and Moments* function. On the other hand, when the control dynamics are enabled, then the *Control Allocation* function outputs desired deflections, which serve as an input to the *Control Dynamics* function. In this situation, the *Forces and Moments* function obtains the actuator deflections and thrust directly from the control state vector.

Finally, if the user wishes to include any additional states, they can do so by providing the *Auxiliary Dynamics* function, which calculates the derivatives of these states. Auxiliary dynamics can cover additional aircraft states but can also accommodate any dynamics of the Flight Control System, such as integrator or filter states.

Despite dispatching the overwhelming majority of the model description to the user functions, the library's model structure still requires a couple of parameters. Those are non-arbitrary quantities like mass and tensor of inertia (required for evaluation of kinematic equations of motion) or general model information, such as the number of control surfaces, choice of velocity representation, etc. The user can also pass actuator deflection and thrust limits. While these are not really used in the handling qualities requirements, they can be used to assess the validity of the trim points.

The library implements the aircraft model as a `NonlinearIOSystem` * from the python-control library, so all functions and methods designed to work with that class should also work with the aircraft model.

As discussed in the Background section, existing tools employ a range of approaches to model structure. On one end of the spectrum, the library may force a very specific model structure and ask the user to pass a predefined set of parameters, for example, stability and control derivatives. On the other end of that spectrum, the user may be provided with a blank canvas and asked to implement the entire aircraft model. The proposed model framework aims to exploit the benefits of both approaches. By deconstructing it into smaller elements and providing a top-level structure, the challenge of creating an aircraft model is simplified for the user. At the same time, the user still has the flexibility to use various forms of model representation, such as lookup tables, polynomial models, or multivariate splines.

C. Trim and linearization

To trim the aircraft, the library uses the `find_eqpt` † function. The function finds an equilibrium point of a system based on which inputs/outputs/states are free and which are fixed and to what values. Based on user input, the library generates the equilibrium condition specification that is later passed to the `find_eqpt`, together with an aircraft model.

The library implements the wings-level flight condition, which is a flight with zero roll and sideslip angles and constant vertical speed. The flight condition is described by the desired altitude and speed and optionally desired vertical speed or a flight path angle γ for climb and descent. Although not relevant for the handling qualities evaluations, the trim module also implements the Steady Heading Sideslip flight condition to demonstrate the library's trim capabilities. Furthermore, the library provides a framework through which users can define their own flight condition categories.

An important feature of the trim function is that it finds the trim point in terms of the pilot input, not in terms of the individual actuator deflections. There are two reasons for that. Firstly, the trim condition is constructed in a way that the optimization problem is well-constrained, meaning that the number of degrees of freedom equals the number of constraints. Using actuator deflections instead of pilot input would make the problem under-determined, meaning it has infinitely many solutions. Adjusting the number of constraints to match the new number of degrees of freedom would

*`https://python-control.readthedocs.io/en/latest/generated/control.NonlinearIOSystem.html`

†`https://python-control.readthedocs.io/en/latest/generated/control.find_eqpt.html`

Table 1 Summary of available system reductions

Name	States (using aerodynamic velocity)	States (using body velocities)	Inputs
Longitudinal	θ, V, α, q	θ, u, w, q	pitch stick
Short-period	α, q	w, q	pitch stick
Lateral-directional	ϕ, β, p, r	ϕ, v, p, r	roll stick and yaw pedals
Directional	β, r	v, r	yaw pedals

be impractical as the number of control surfaces may change in the future, and the entire trim problem would have to be reworked. Secondly, defining the trim point in terms of actuator deflections treats each of them independently, bypassing the control allocation, which constrains the deflections to each other. This means that the trim point will most likely have a control surface configuration that violates the control allocation.

The library also performs the numerical linearization of the entire model, which means that the dynamics, kinematics, control allocation, and flight control system get linearized into a single state-space system. The linearization can be done with respect to either velocity representation: V, α, β, or u, v, w. Part of the linearization module are also functions to generate reduced linear models, often used in handling qualities evaluations (see Table 1).

D. Eigenmode identification

As most of the handling qualities requirements rely on the modal parameters of the linear aircraft model, obtaining them is a crucial step in the process. It is assumed, and was shown in previous research, that the Flying-V exhibits the classic five eigenmodes: short period (can be overdamped[11, 13]), phugoid, roll mode, spiral mode, and the Dutch Roll mode. The modal parameters in question include time constants for first-order modes, and natural frequency and damping ratio for second-order modes. Those can be directly calculated from the eigenvalues of the linearized system's **A** matrix. In the identification process, those eigenvalues have to be matched to specific eigenmodes.

Finding the appropriate eigenvalue from (at least) 12 values is challenging. However, a good approximation of these eigenvalues can be found in the reduced systems. A 4-state reduced system for longitudinal movements, with $[\theta, V, \alpha, q]$ states (or $[\theta, u, w, q]$, depending on the velocity representation) will contain the approximate phugoid and short period modes – 4 eigenvalues for two second-order modes. Similarly, with lateral-directional movements, a different 4-state reduction with $[\phi, \beta, p, r]$ states (or $[\phi, v, p, r]$) will contain approximate spiral, roll, and Dutch Roll modes – 4 eigenvalues for two first-order modes and one second-order mode). Both systems will have eigenvalues only related to these modes, which makes the identification easier.

In matching the eigenvalues of the reduced longitudinal system to the longitudinal eigenmodes, the following is assumed:

- the phugoid is an oscillatory mode ($\zeta < 1$, produces two complex poles),
- the short period may be overdamped (two poles, but they do not have to be complex),
- the phugoid has a smaller natural frequency than the short period.

In matching the eigenvalues of the reduced lateral-directional system to the lateral-directional eigenmodes, the following is assumed:

- the Dutch Roll is an oscillatory mode ($\zeta < 1$, produces two complex poles),
- the roll mode has a smaller time constant than the spiral mode.

If the system meets these assumptions, the reduced systems' eigenvalues are guaranteed to be unambiguously matched to the appropriate eigenmodes. Preliminary analysis showed that those reduced systems eigenvalues approximate the full-system eigenvalues very well unless auxiliary states are used. In cases where the auxiliary states influence the kinematic states (e.g., flight control system states), those states also affect the eigenmodes. Simple reductions ignore that influence. To address that, first, eigenvalues of the reduced systems are obtained and matched to appropriate eigenmodes using the assumptions above. Next, the reduced systems are augmented to include the auxiliary states, and the eigenvalues of those systems are obtained. Finally, the eigenvalues connected to the eigenmodes are updated with the augmented systems' eigenvalues. The updates are done by finding a value in the augmented system eigenmodes that is closest to the current value connected with a given mode. The updates are done separately for longitudinal and

Table 2 Handling qualities requirements available in the library

Name	Source	Description
Phugoid modal	MIL-STD, ESDU, Hodgkinson	Limits on the minimum damping ratio of the phugoid mode
CAP	MIL-STD, ESDU, Hodgkinson	Allowable ranges for the Command Anticipation Parameter (CAP). CAP $\approx g\omega_{sp}T_{\theta 2}/V$. For more, see: [29] and [7, p. 176–177]
Short period damping	MIL-STD, ESDU, Hodgkinson	Allowable ranges for the short period damping ratio
Gibson dropback	MIL-STD, Hodgkinson	Not a full requirement. After the step input is removed, the ratio between attitude dropback and steady-state pitch rate (DB/q_{ss}) should be between 0 and 0.25.
Short period step response	MIL-STD	Allowable ranges for the transient peak ratio, effective time delay, and effective rise time in the pitch rate step response (with speed constant)
Spiral modal	MIL-STD, ESDU	Minimum time to double amplitude of the spiral mode
Roll modal	MIL-STD, ESDU	Maximum time constant of the roll mode.
Dutch Roll modal	MIL-STD, ESDU, Hodgkinson	Allowable limits for the natural frequency, damping ratio, and their product, of the Dutch Roll mode.
Phi-to-beta	MIL-STD, ESDU, Hodgkinson	The value of $(\phi/\beta)_d$ – a modal response ratio of roll angle to sideslip angle at the Dutch Roll. It is not constrained by any requirement on itself. However, it can influence allowable limits in the Dutch Roll modal requirement.

lateral-directional cases.

Lastly, this module also provides utilities that obtain the modal parameters from the eigenvalues. For the first-order modes, those parameters are time constant and time to double amplitude for unstable modes. For second-order modes, those parameters are natural frequency, damping ratio, and time to double amplitude for unstable modes.

E. Handling qualities requirements evaluation

Table 2 presents a summary of handling qualities requirements available in the library. The source of these requirements are MIL-STD-1797 [7], ESDU 92006 [8], and "Aircraft handling qualities" by John Hodgkinson [15]. However, as mentioned in section II, those contain mostly identical requirements.

The modal requirements (including Command Anticipation Parameter (CAP) and short-period damping) require comparing modal parameters (already obtained), or simple functions thereof, with allowable ranges and deducing the final level. The CAP also requires $T_{\theta 2}$ – a high-frequency zero in the pitch attitude to pitch input transfer function. In the library, the parameters are obtained from the short-period reduced system transfer function, which is defined to have the following form:

$$\frac{q}{u_p} = \frac{k_q(1 + T_{\theta 2}s)}{s^2 + 2\zeta_{sp}\omega_{sp}s + \omega_{sp}^2} \tag{1}$$

The Gibson dropback criterion is evaluated from the simulated response of a reduced longitudinal system with constant speed. The pitch input is held at 1 for the first half of the simulation (20 seconds) and 0 for the rest. The simulation time is long enough to allow the aircraft to settle (in pitch rate) after both input changes. The dropback is measured as the change in the pitch angle θ from the point the step input is taken off to the steady-state value thereafter. An example of such a response is presented in Figure 4.

The pitch rate step response requirement imposes allowable ranges for three parameters of the response. The first is the transient peak ratio, which is a ratio of the first peak (q_1) and first dip (q_2) with respect to the steady state value (q_{ss}). It also requires calculating the effective time delay – the value at which the maximum slope of the step response

Table 3 Combined short-period requirement limits for the Level 1 handling qualities

Flight phase category	min CAP	max CAP	min ω_{sp}	min ζ_{sp}	max ζ_{sp}
A	0.28	3.6	1.0	0.35	1.3
B	0.085	3.6	—	0.3	2.0
C	0.16	3.6	0.7	0.35	1.3

Table 4 Dutch Roll requirement limits for the Level 1 handling qualities

Flight phase category	min ζ_{dr}	min $\zeta_{dr}\omega_{dr}$	min ω_{dr}
A	0.19	0.35	0.4
B	0.08	0.15	0.4
C	0.08	0.10	0.4

intersects the time axis. Lastly needed is the effective rise time – a difference between when the step response crosses the eventual steady-state value and the effective time delay. An example of such a response is presented in Figure 3.

The $(\phi/\beta)_d$ modal response ratio describes the relation of the roll angle response to the sideslip angle response in the Dutch Roll mode. It is a complex value that includes amplitude and phase relation between the two states. Besides influencing the Dutch Roll modal requirement, its value gives a deeper insight into the mode's behavior, other than its natural frequency and damping. The ratio is obtained using a method described by McRuer et al. [30, p.72–74] with state space matrices.

F. Stability augmentation system tuning

The purpose of the automated stability augmentation system tuning in the library is not to supply the aircraft with a full and robust flight control system but to provide additional insight into the handling qualities of the Flying-V. If the gains required to improve the ratings to Level 1 are very large, that may indicate that the issue cannot be addressed with a flight control system, and a redesign may be required. On the other hand, if the gains are small, that may indicate that a flight control system can be sufficient to address the issues. This module should also be thought of as a proof-of-concept demonstration of how can the automated flight control system tuning be integrated into handling qualities evaluation.

The library can tune two controllers: a pitch rate controller that improves the short period response using the α and q feedback, and a yaw damper that improves the Dutch roll behavior using the ϕ, β, p, q feedback. The gains are obtained using the pole placement method [31] (implemented in SciPy library [28]). The pole locations are optimized to minimize the sum of gains (their absolute value) while still providing Level 1 handling qualities in relevant requirements: CAP and short period damping (see limits in Table 3) for the pitch rate controller and Dutch Roll modal requirement (see limits in Table 4) for the yaw damper. For the yaw damper, the feedback had to be made from 4 states because the Dutch Roll, even approximated, could not be found in smaller reduced systems (with fewer states), making the tuning through pole placement impossible. Since feeding back four states places four poles, that means that also the spiral and roll mode poles are placed. Currently, those poles are being placed in the same location as the open-loop system.

IV. Validation

During testing and validation, two aircraft models were implemented using the library: an F-16 model [32], created at the University of Minnesota, and the Flying-V model used by Joosten et al. [12] and Torelli et al. [13].

The F-16 is a nonlinear model, which includes kinematic equations of motion, and comes with a set of MATLAB and Simulink files that add actuator dynamics, as well as trim and linearize the model. This makes it a good model to test the respective elements of the developed library. The F-16 software package contains two versions of the F-16 model: a low-fidelity and a high-fidelity one. Parts of the model (dynamics and kinematics) were implemented in the C programming language, while the rest (mainly actuator dynamics) is in the Simulink model. In order to implement the F-16 in the library, the C source file was changed slightly to return forces and moments instead of state derivatives, then compiled as a shared library and wrapped in a short Python code. The actuator dynamics and control allocation were

Table 5 Summary of validation sources/methods

	Model Implementation	Kinematics	Trim	Linearization	Eigenmode Identification	HQ requirements	SAS tuning
F-16 [32]	✓	✓	✓	✓			
Joosten et al. [12]	✓	✓	✓	✓	✓		
Torelli et al. [13]	✓	✓	✓				
unit-tests					✓	✓	
manual testing						✓	✓

```
-------Dutch Roll modal requirement-------
Natural frequency: 0.512
Natural frequency level: 1
Damping ratio: 0.023
Damping ratio level: 2
omega*zeta product: 0.012
omega*zeta product level: 3
Final level: 3
```

Fig. 2 Example output (verbose) of the Dutch Roll modal requirement

also re-implemented in Python, based on the Simulink model. The high-fidelity model has an additional control surface – the leading edge flap. This control surface is not controlled by the pilot and is instead a function of other state variables. Furthermore, the controller driving its deflection introduces an auxiliary state which allows showcasing the auxiliary dynamics part of the library's aircraft model.

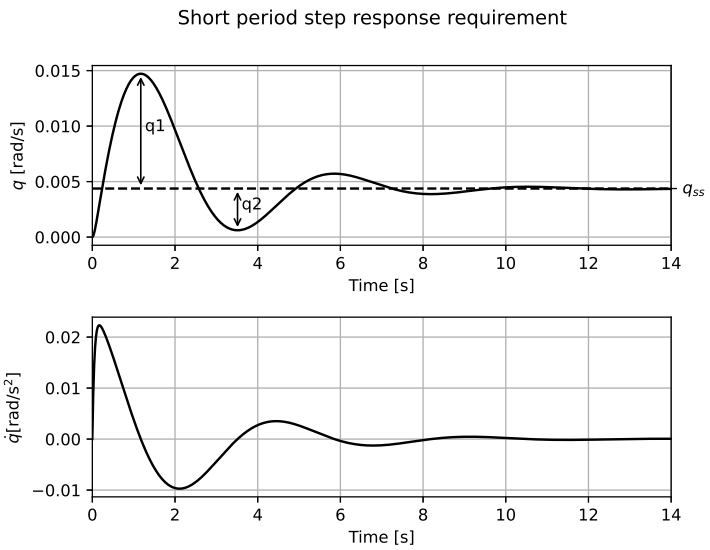
On the other hand, the Flying-V model comes in the form of a JSON (JavaScript Object Notation – a text-based data container format) file containing look-up tables for stability and control derivatives for different speeds and center-of-gravity locations. Joosten et al. [12] and Torelli et al. [13] implemented the aircraft in MATLAB and provided their own engine models, control allocation, and flight control systems, which were not part of the main model. Their implementation was replicated in Python, using the library's framework.

As mentioned earlier, the F-16 software package evaluates the kinematic equations of motion and can trim and linearize the aircraft. Therefore, those results can be compared with ones generated by the library. Likewise, Joosten et al. [12] and Torelli et al. [13], during their work on the Flying-V model, performed the same steps, as well as identified the eigenmodes of the linearized systems. The library was used to recreate those results for validation purposes.

The provided model structure was shown to function properly. Both F-16 models (low-fidelity and high-fidelity), as well as multiple configurations of the Flying-V (with different control allocation schemes, with or without the flight control systems), were successfully implemented and could be handled by the library. Furthermore, it was demonstrated that the library correctly evaluates the kinematic equations of motion.

The library also correctly replicated the trim points for the F-16 and the Flying-V. For the F-16 high-fidelity model, trim convergence was found to be sensitive to initial conditions. The library also correctly replicated the linearized matrices, although small discrepancies were found due to different linearization algorithms. Furthermore, the library correctly recreated the model parameters obtained by Joosten et al. [12].

In some cases, the validation could be better performed using automatic tests independent of any aircraft model. An example of that may be those handling qualities requirements that require checking the allowable limits for specific parameters like, for instance, the Dutch Roll modal requirement (Figure 2). In these instances, a set of input values covering every possible case was created, together with expected outputs (handling qualities level). Requirements that rely on time responses, and in which the library has to identify specific parameters from these responses, required manual validation. Those requirements were run on several configurations of the Flying-V model, and their results were carefully compared with plots of time responses. That allowed to assess whether the library correctly identified, e.g., the




```
---Pitch rate step response requirement---
Effective time delay: 0.039
Effective time delay level: 1
Effective raise time: 0.196
Effective raise time level: 1
Transient peak ratio:  0.363
Transient peak ratio level: 2
Final level: 2
```

(a) Time response **(b) (Verbose) console output**

Fig. 3 Short period step response requirement example

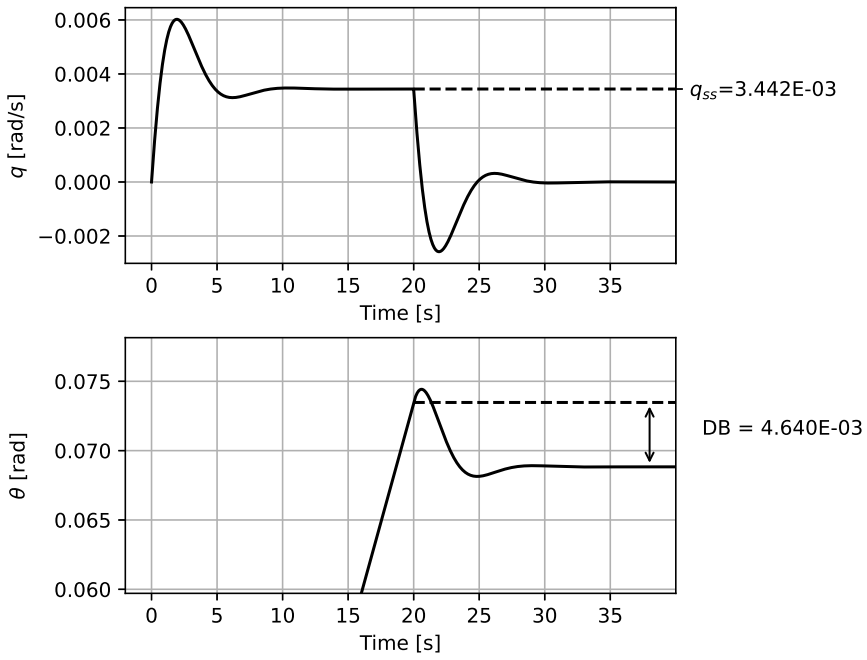


Fig. 4 Example of Gibson dropback criterion evaluation. The scale in the bottom plot is adjusted to better show the dropback.

response peaks or the steady-steady values. Examples of such requirements can be seen in Figure 3 and in Figure 4. Overall, all requirements were successfully validated.

The automatic tuning of the stability augmentation system was run on several configurations of the Flying-V and yielded results indicating correct behavior. First of all, the handling qualities of interest were improved to Level 1 in all cases. The most limiting parameter always landed just on the edge of Level 1, suggesting that the gains were optimized. Additionally, the relative magnitude of gains in the yaw damper resembled that obtained by Joosten et al. [12], who implemented a similar flight control system.

A summary of which method was used to validate which library module is presented in Table 5. To sum up, all modules of the library were successfully validated, and no major redesigns were required.

Finally, it is difficult to rigorously quantify the time savings that the library brings. However, it can be estimated that in the previous works [9–13] obtaining the handling qualities predictions could have taken at least a month of work. For the author, it took a day to re-implement the Flying-V model used by Joosten et al. [12] in the library. Furthermore, during validation and comparison between the library results and the reference Flying-V works, several programming errors in the latter were discovered (inconsequential to the outcome of the research).

V. Discussion

The validation confirmed the correct functioning of all of the library's modules. Some of the results prompted minor redesigns of the library. However, no fundamental changes were required. The library is ready to be used in the next Flying-V handling qualities research, where it should bring substantial time savings and reduce the risk of programming errors.

Successful implementation of both aircraft models showcases the expected robustness of the library with respect to the model structure, as the two aircraft models are represented in very different ways. The F-16 was an almost fully implemented aircraft model with force equations, written in another programming language. On the other hand, the Flying-V model consisted of raw data that had to be turned into appropriate equations. Although a small effort had to be made to implement them, it is argued that the benefit of flexibility outweighs that. Furthermore, if the same format of the Flying-V model will be utilized in the future, then that implementation code can be reused, thus requiring almost no additional work.

During validation, some errors in the previous work were revealed. Although they were not consequential, that perfectly showcases why such a library can minimize the risks of programming mistakes in subsequent research, as the library was carefully validated.

Many future handling qualities experiments will include piloted flight simulation experiments. While the library does not cover this aspect, it should be very helpful in the process. Usually, it is impossible to test all available aircraft configurations and flight control system designs in a simulator due to limited time and resources. Therefore, the configurations to be tested in the simulator must be carefully picked. The library will allow getting the preliminary analysis done quickly, perhaps test multiple control allocation schemes and flight control system solutions, and nominate the most promising ones for the simulator tests.

It is expected that the library will be further developed. The model structure can be expanded to accommodate even more complex models, and new handling qualities requirements can be implemented. The core functionalities of the library also allow for expansion outside of the current scope. The previous Flying-V handling qualities research often included some performance-related analysis, such as investigating trim points at various flight conditions. Since most of the steps required for that analysis are already covered by the library, expanding toward more performance-oriented analysis should not be an issue.

The library was shown to be robust to user model specification but should also be robust to potential changes in the library's model structure (Figure 1). For all of the handling qualities-related activities, a linearized model is used. The linearization is performed numerically and is agnostic to the internal structure of the non-linear model. For the non-linear operations, it should be reminded that the library's model implements the python-control's `NonlinearIOSystem`, which allows using that library's functionalities on the Flying-V model, regardless of its internal structure.

VI. Conclusion

The developed library for automated handling qualities evaluation of the Flying-V should speed up the subsequent handling qualities research to a large degree and decrease the risk of mistakes. The library provides a flexible model implementation framework, which can accommodate differently formulated aircraft models. This was demonstrated by successfully implementing two dissimilar aircraft models – the well-established F-16 model from the University of Minnesota and the Flying-V model, which was used in previous research. The library trims and linearizes the aircraft model, identifies the modal parameters of the aircraft, and evaluates the requirements – either from the modal parameters or from the time response of the linearized models. Furthermore, the library tunes the stability augmentation system that improves undesirable handling qualities. The library was successfully validated against the earlier Flying-V research and the F-16 model, as well as using automatic and manual testing.

References

[1] Benad, J., "The Flying V - A new Aircraft Configuration for Commercial Passenger Transport," *Deutscher Luft- und Raumfahrtkongress*, 2015. https://doi.org/10.25967/370094.

[2] Faggiano, F., Vos, R., Baan, M., and Dijk, R. V., "Aerodynamic Design of a Flying V Aircraft," *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017. https://doi.org/10.2514/6.2017-3589.

[3] Oosterom, W., and Vos, R., "Conceptual Design of a Flying-V Aircraft Family," *AIAA AVIATION 2022 Forum*, 2022. https://doi.org/10.2514/6.2022-3200.

[4] Cooper, G. E., and Harper, R. P., "The use of pilot rating in the evaluation of aircraft handling qualities," Report TN-D-5153, NASA, 1969. URL https://ntrs.nasa.gov/citations/19690013177.

[5] Ashkenas, I. L., "Twenty-five years of handling qualities research," *Journal of Aircraft*, Vol. 21, No. 5, 1984, pp. 289–301. https://doi.org/10.2514/3.44963.

[6] Phillips, W. H., "Flying qualities from early airplanes to the Space Shuttle," *Journal of Guidance, Control, and Dynamics*, Vol. 12, No. 4, 1989, pp. 449–459. https://doi.org/10.2514/3.20432.

[7] US Department of Defence, "MIL-HDBK-1797. Flying Qualities of Piloted Aircraft," , 1997.

[8] Engineering Sciences Data Unit, "ESDU 92006. A background to the handling qualities of aircraft," , 1992.

[9] Cappuyns, T., "Handling Qualities of a Flying V Configuration," Unpublished MSc thesis, TU Delft, 2019.

[10] Overeem, S. v., Wang, X., and Kampen, E.-J. V., "Modelling and Handling Quality Assessment of the Flying-V Aircraft," *AIAA Scitech Forum*, 2022. https://doi.org/10.2514/6.2022-1429.

[11] Vugts, G., Stroosma, O., Vos, R., and Mulder, M., "Simulator Evaluation of Flightpath-oriented Control Allocation for the Flying-V," *AIAA SCITECH 2023 Forum*, 2023. https://doi.org/10.2514/6.2023-2508.

[12] Joosten, S., Stroosma, O., Vos, R., and Mulder, M., *Simulator Assessment of the Lateral-Directional Handling Qualities of the Flying-V*, 2023. https://doi.org/10.2514/6.2023-0906.

[13] Torelli, R., Stroosma, O., Vos, R., and Mulder, M., "Piloted Simulator Evaluation of Low-Speed Handling Qualities of the Flying-V," *AIAA SCITECH 2023 Forum*, 2023. https://doi.org/10.2514/6.2023-0907.

[14] Rizzi, A., "Modeling and Simulating Aircraft Stability & Control - SimSAC Project," *AIAA Atmospheric Flight Mechanics Conference*, 2010. https://doi.org/10.2514/6.2010-8238.

[15] Hodgkinson, J. M. S., *Aircraft handling qualities*, AIAA education series, American Institute of Aeronautics and Astronautics; Blackwell Science, 1999.

[16] European Union Aviation Safety Agency, "Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes (CS-25)," , 2021. URL https://www.easa.europa.eu/document-library/certification-specifications/cs-25-amendment-27.

[17] Marco, A. D., Trifari, V., Nicolosi, F., and Ruocco, M., "A Simulation-Based Performance Analysis Tool for Aircraft Design Workflows," *Aerospace*, Vol. 7, No. 11, 2020, p. 155. https://doi.org/10.3390/aerospace7110155.

[18] Tischler, M., Colbourne, J., Morel, M., Biezad, D., Levine, W., Moldoveanu, V., Tischler, M., Colbourne, J., Morel, M., Biezad, D., Levine, W., and Moldoveanu, V., "CONDUIT - A new multidisciplinary integration environment for flight control development," 1997. https://doi.org/10.2514/6.1997-3773.

[19] Duus, G., and Duda, H., "HAREM - HAndling Qualities Research and Evaluation using Matlab," *IEEE International Symposium on Computer Aided Control System Design*, 1999, pp. 428–432. https://doi.org/10.1109/CACSD.1999.808686.

[20] Ohme, P., and Raab, C., "A Model-Based Approach to Aircraft Performance Assessment," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, 2008. https://doi.org/10.2514/6.2008-6873.

[21] Ohme, P., "A Model-Based Approach to Aircraft Takeoff and Landing Performance Assessment," *AIAA Atmospheric Flight Mechanics Conference*, 2009. https://doi.org/10.2514/6.2009-6154.

[22] Krishnamurthy, V., and Luckner, R., "Automated Evaluation of Handling Qualities and Performance for Preliminary Aircraft Design using Flight Simulation Models," *Deutscher Luft- und Raumfahrtkongress*, German Aerospace Society, 2014. URL https://publikationen.dglr.de/?tx_dglrpublications_pi1%5bdocument_id%5d=340087.

[23] Goetzendorf-Grabowski, T., Mieszalski, D., and Marcinkiewicz, E., "Stability Analysis in Conceptual Design Using SDSA Tool," *AIAA Atmospheric Flight Mechanics Conference*, 2010. https://doi.org/10.2514/6.2010-8242.

[24] Lawrence, B., Theodore, C., Johnson, W., and Berger, T., "A handling qualities analysis tool for rotorcraft conceptual designs," *The Aeronautical Journal*, Vol. 122, No. 1252, 2018, pp. 960–987.

[25] Zanoni, A., Gerosa, G., Di Lallo, L., and Masarati, P., "Handling Qualities in Rotorcraft Conceptual Design," *Aerotecnica Missili & Spazio*, Vol. 101, No. 1, 2022, pp. 95–108.

[26] Fuller, S., Greiner, B., Moore, J., Murray, R., van Paassen, R., and Yorke, R., "The Python Control Systems Library (python-control)," *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 4875–4881. https://doi.org/10.1109/CDC45484.2021.9683368.

[27] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., "Array programming with NumPy," *Nature*, Vol. 585, No. 7825, 2020, pp. 357–362. https://doi.org/10.1038/s41586-020-2649-2.

[28] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, Vol. 17, 2020, pp. 261–272. https://doi.org/10.1038/s41592-019-0686-2.

[29] Bihrle Jr., W., "A Handling Qualities Theory for Precise Flight Path Control," Report TR-65-198, AFFDL, 1966.

[30] McRuer, D., Ashkenas, I., and Graham, D., *Aircraft Dynamics and Automatic Control*, Princeton University Press, 1973.

[31] Kautsky, J., Nichols, N. K., and van Dooren, P., "Robust pole assignment in linear state feedback," *International Journal of Control*, Vol. 41, No. 5, 1985, pp. 1129–1155. https://doi.org/10.1080/0020718508961188.

[32] Russell, R. S., *Non-linear F-16 Simulation using Simulink and Matlab*, University of Minnesota, 2003. URL https://dept.aem.umn.edu/~balas/darpa_sec/software/F16Manual.pdf.

[33] Świderski, A., "Preliminary Thesis Report: Pipeline for automated handling qualities evaluations of the Flying-V," Unpublished preliminary thesis, TU Delft, 2022.

A. Answering the Research Questions

A. Introduction

In the preliminary thesis [33], after scoping the problem, the research objective was formulated, which specified the desired functionalities of the library. This was followed by an introduction of the research questions, which asked how those functionalities would be automated. In the second part of the preliminary thesis, a design proposal, which answered (as hypotheses) those questions, was presented. After that, a second group of research questions was introduced, this time pertaining to the specific design proposal. The aim of these questions was, amongst others, to guide the validation of the library. Those questions were as follows (cited verbatim):

- "Does the proposed aircraft model structure accommodate an arbitrary number of control surfaces, auxiliary states, and any control allocation scheme?"
- "Does the pipeline[‡] correctly implement the kinematic equations of motion?"
- "Does the pipeline automatically and correctly trim the aircraft for the "straight-flight" flight condition?"
- "Does the pipeline automatically and correctly linearize the aircraft around a given trim point?"
- "Does the proposed method correctly identify eigenmodes from the state-space model?"
- "Does the pipeline correctly evaluate the handling qualities predictions?"
- "Does the proposed stability augmentation system, implemented and automatically tuned as described in Section 4.6 improve the CAP and the Dutch Roll requirements?"
- "How does the stability augmentation system, implemented and automatically tuned as described in Section 4.6 influence the remaining handling qualities requirements?"

In this section, the activities undertaken to validate the library will be presented, as well as their results. Their description will be organized by the aforementioned research questions. Note: in some cases, the research questions will be rephrased for clarity.

As mentioned in section IV, the F-16 model [32] and the Flying-V model were used in the validation. The goal was to replicate the data that can be generated from the F-16 software package (trim and linearization) and some of the results of Joosten et al. [12] and Torelli et al. [13] with the library. If the results match, that gives high confidence in the library's correct operation and, at the same time, validates the reference data. To compare the results, NumPy's function `assert_all_close`[§] was used. It uses two parameters that will also be used to quantify the accuracy of the match: relative tolerance (rtol) and absolute tolerance (atol). The assertion passes when the differences are smaller than $\text{atol} + \text{rtol} \cdot (x)$, where x is the reference value. The absolute tolerance is useful in situations where minuscule numerical artifacts are created, which can generate large relative errors. Absolute and relative tolerances of the match will be used in tables throughout this section. Those are not the measurements of the match, per se, but rather the most conservative settings that still allow the assertion to pass. Note that if, for a specific model, several data points are used, atol and rtol are set for the worst match.

Not all parts of the library were validated by comparing them to reference data. In some cases, there was no data to which to compare, for example, with some handling qualities requirements or stability augmentation system tuning. In other cases, like the modal handling qualities requirements, using different methods was more effective.

B. Research questions and answers

Does the proposed aircraft model structure accommodate an arbitrary number of control surfaces, auxiliary states, and any control allocation scheme?

The F-16 high-fidelity model contains a leading edge flap. This control surface is not controlled by the pilot, instead is a function of the angle of attack, as well as static and dynamic pressure. Additionally, there is an integration step involved. That means that compared to the low-fidelity model, the high-fidelity model has an additional control surface and auxiliary state.

Therefore, the F-16 model has 3 or 4 control surfaces. The Flying-V has 6 control surfaces. Furthermore, during validation, 3 different control allocation schemes for the latter were implemented – two from Joosten et al. [12] and one from Torelli et al. [13]. Besides that, the stability augmentation system designed by Joosten et al. [12], which introduces an auxiliary state (the washout filter state), was also implemented.

[‡] In the design proposal, the software was referred to as 'pipeline', not 'library'.

[§] `https://numpy.org/doc/stable/reference/generated/numpy.testing.assert_allclose.html`

Table 6 Summary of trim results comparisons

Model	Flight Condition	Number of datapoints	rtol	atol
F-16 low-fidelity	straight level	1	10^{-7}	10^{-9}
F-16 high-fidelity	straight level	1	10^{-3}	0.03
Flying-V (Joosten et al. [12])	straight level	27	10^{-5}	10^{-6}
Flying-V (Torelli et al. [13])	straight level	15	10^{-7}	10^{-9}
Flying-V (Joosten et al. [12])	steady-heading-sideslip	23	10^{-5}	10^{-6}
Flying-V (Torelli et al. [13])	descent (-3 deg)	13	$5 \cdot 10^{-4}$	10^{-6}

At no point during subsequent testing did the library have any issues with handling these models and their configurations. These models have different sources for force equations, various numbers of control surfaces, various control allocation schemes, and some introduced auxiliary states. Therefore it can be concluded that **the proposed aircraft model structure accommodates an arbitrary number of control surfaces, auxiliary states, and any control allocation scheme.**

Does the library correctly implement the kinematic equations of motion?

Both the F-16 model as well as the work of Joosten et al. [12] contain the 6DOF equations of motion under the same assumptions. Results from these two were compared with the results obtained from the library for a random input and state vector. In both instances, the results matched, meaning that the **the library correctly implements the kinematic equations of motion.**

Does the library automatically and correctly trim the aircraft for the "straight-flight" flight condition?

The library was used to trim the two F-16 models and several Flying-V configurations. Table 6 presents a summary of trim results comparison against the F-16 software package and the Flying-V research data. The worst match of trim results happened for the F-16 high-fidelity model. The tolerance is lowered mainly because of small non-zero aileron and rudder deflections in the trim points in both the library and F-16 software. Those, of course, for that flight condition should be both zero. In section IV, it was mentioned that the trim routine was sensitive to the initial condition for the high-fidelity model. That specifically concerned the initial angle-of-attack value. However, for the most intuitive value of $\alpha = 0$, all trim problems (including the Flying-V) converged. Because the reference Flying-V work contained calculations for the many configurations of the aircraft, that allowed comparing results from a multitude of data points. For the Joosten et al. [12] implementation, the aircraft was also trimmed for two different control allocation schemes as well as the stability augmentation system.

All the trim points compared during validation matched. Furthermore, the `find_eqpt` function used in the trim routine returns, amongst others, information from the solver if the optimization was converged successfully. That was the case for all the test runs. Furthermore, for a couple of aircraft configurations, a simple non-linear simulation was run, with a trim point (state and input) as initial conditions, to verify that the aircraft is indeed in equilibrium. That was also confirmed. All that allows concluding that the library **correctly trims the aircraft for the "straight-flight" flight condition.** Although not asked in the original research questions, the library also correctly trims the aircraft for the steady-heading-sideslip.

Does the library automatically and correctly linearize the aircraft around a given trim point?

Table 7 presents a summary of linearization results comparison conducted during validation. During testing, it was discovered that the type of linearization algorithm has a non-negligible influence on the results of the linearization. Initially, the linearization function in python-control was used. It used the perturbation method with a forward difference, which was a source of some inaccuracies. Therefore, the linearization method was re-implemented using central difference perturbation. This improved the accuracy of the match for both the F-16 and Flying-V data. Furthermore, initial comparison with the work of Joosten et al. [12] yielded noticeable differences, which were traced to two programming bugs in their source code. After fixing them, the errors dropped by one order of magnitude. There may still be errors in the reference code. Otherwise, the current discrepancies are most likely due to different linearization

Table 7 Summary of linearization results comparisons

Model	Flight Condition	Number of datapoints	rtol	atol
F-16 low-fidelity	straight level	1	10^{-5}	10^{-5}
F-16 high-fidelity	straight level	1	10^{-5}	10^{-5}
Flying-V (Joosten et al. [12])	straight level	27	10^{-6}	10^{-2}

approaches – Joosten et al. linearized the forces equations numerically and kinematics analytically. Regardless, given the close match on multiple points and various models, it can be concluded that the **library correctly linearizes the aircraft around a given trim point.**

Does the library correctly identify eigenmodes from the state-space model?

To validate the eigenmode identification, the results from the Joosten et al. [12] were used. For 27 different aircraft configurations, the results matched very closely (tested on rtol = 10^{-9} and atol = 0), allowing to conclude that **the library correctly identifies eigenmodes from the state-space model.** The validation highlighted a need to match the reduced-system eigenmodes to the systems augmented by auxiliary variables. Some of the tested configurations contained the yaw damper, one of its elements being the washout filter on the yaw rate. That meant that the FCS's output and, as a result, the aircraft yaw response was a function of the auxiliary filter states. The classic 4-state reduced system ignores that variable, and the results were as if there was no washout filter. Matching the reduced system's eigenvalues to the augmented system's eigenvalues allowed for achieving the correct results. The possibility of matching the approximated eigenvalues to the full system was explored during development. However, it became apparent that that runs a risk of mixing up the longitudinal and lateral-directional eigenvalues.

Does the library correctly evaluate the specified handling qualities predictions?

The validation of the handling qualities predictions was performed in several ways. For those requirements specifying allowable ranges of parameters, automatic testing was performed. The tests were designed to cover each possible case. Table 8 present which requirements were tested that way, together with the number of required tests. Simple requirements, like the phugoid modal requirement, needed a small number of points, while the more complex ones, like the Dutch Roll modal, required more.

The remaining requirements were tested manually. The short-period step response and Gibson criteria were validated by looking at the time responses and seeing if the obtained results matched the plots, as the values used in the criteria can be easily read from the plots. An example of the Gibson criterion can be seen in Figure 4. It is unclear what aircraft model the Gibson dropback criterion requires. However, in the initial Flying-V tests using a 4-state reduced longitudinal system, the phugoid response was so profound that it completely obscured the desired dropback observation, as evident in Figure 5. Therefore, it was decided to use a 2-state short-period reduced system for that requirement.

The $(\phi/\beta)_d$ modal ratio is calculated in the library from the state space matrices but can also be estimated from time histories. The second method is less accurate but can be used to validate the library results. An example of such a response is presented in Figure 6. Because the modal ratio in question is specifically in the Dutch Roll mode, the response must contain only that mode. To do that, the initial condition was set to the Dutch Roll eigenvector. The phase of $(\phi/\beta)_d$ was obtained by comparing the time separation between peaks in the two responses, and the results matched the ones obtained by the library with errors smaller than 0.1 degrees. The magnitude of the $(\phi/\beta)_d$ was estimated by dividing the first peak values of each response. That method is not accurate for a better damped Dutch Roll, as the peaks do not appear at the same time, and the amplitude quickly decreases with time. However, for a poorly damped Dutch Roll, as presented in Figure 6, the estimation is accurate.

To summarize, all implemented handling qualities requirements were successfully tested, and correct results were demonstrated. Therefore, it can be concluded that **the library correctly evaluates the specified handling qualities predictions.**

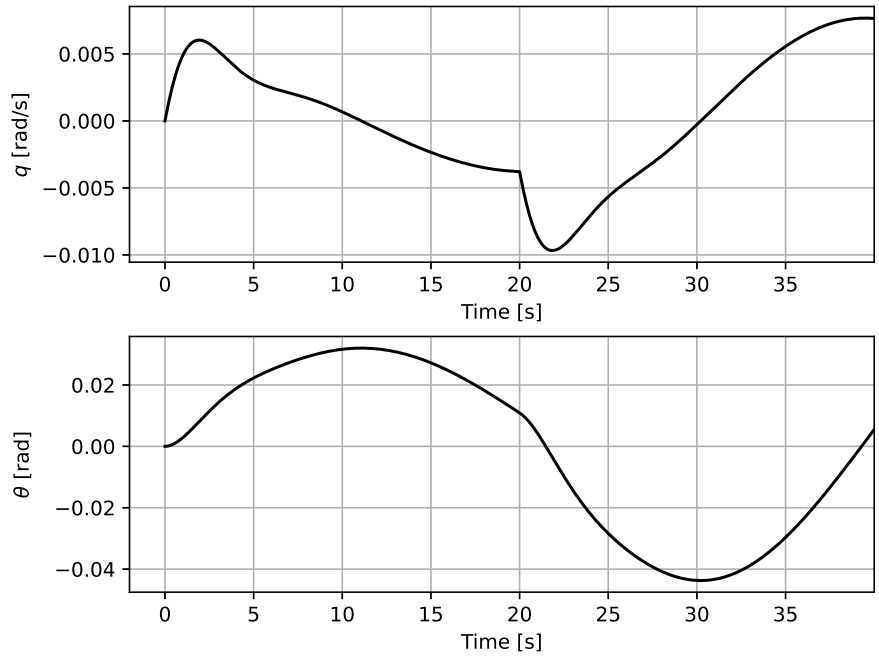


Fig. 5 Aircraft response to Gibson dropback simulation with non-constant speed

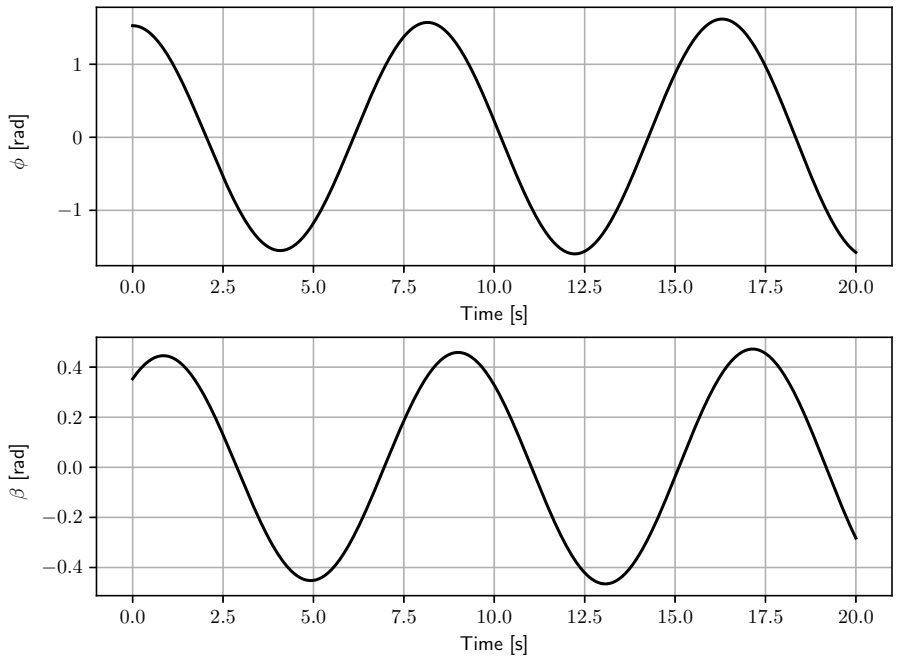


Fig. 6 The time histories of the roll and sideslip angles in the free response in the Dutch Roll. The $(\phi/\beta)_d$ obtained by the library: $2.735 + 2.087i$. $(\phi/\beta)_d$ from the time histories: $2.727 + 2.080i$

Table 8 Summary of handling qualities requirements validation

Requirement	Validation type	number of tests
phugoid modal	unit tests	6
CAP	unit tests	42
short period damping	unit tests	23
short period step	manual testing	9
Gibson Dropback	manual testing	9
roll mode	unit tests	8
spiral mode	unit tests	31
Dutch Roll	unit tests	52
phi-to-beta	manual testing	27

Does the automatically tuned stability augmentation system improve CAP, short period, and the Dutch Roll requirements? How does automatically tuned stability influence the remaining handling qualities requirements?

The automatic tuning of the stability augmentation system was run on several aircraft configurations (those used by Joosten et al. [12]). After the gains were tuned, the stability augmentation system was implemented on the non-linear aircraft model and linearized again to compare the bare airframe and closed-loop systems. Table 9 presented selected results for the pitch rate controller and Table 10 presents selected results for the yaw damper. In the pitch rate controller, it can be seen that the short-period damping and CAP requirements are improved. Furthermore, closed-loop systems scored just on the edge of the limiting requirements. For example, in the first two examples, the short period damping ratio ζ_{sp} is placed at 0.3, which is the bottom limit for Level 1. This shows that the gains were optimized. An interesting observation can also be made that the unsatisfactory short-period damping is improved using the angle-of-attack feedback, and the unsatisfactory CAP (which is a function of short-period frequency) is improved by the pitch rate feedback. For the yaw damper, similar observations can be made. The tuning successfully improves the Dutch Roll requirement. The poles are again placed on the edge of the requirement limits – in this case, the limiting requirement is the product of the damping ratio and natural frequency (0.15 for cruise and 0.1 for takeoff and approach).

For the pitch rate controller, the phugoid is only slightly changed in the closed loop. For the yaw damper, the roll and spiral poles remained unchanged since they were also placed, unlike the phugoid poles. The validation also has shown no cross-effects between the control channels, i.e., the pitch rate controller did not influence the lateral-directional movements, and the yaw damper did not influence the longitudinal movements. All that allows stating that **the stability augmentation system improves the relevant handling qualities requirements and has a negligible effect on other handling qualities requirements.**

C. Discussion

In short, it can be concluded that the library worked as designed. Some of the small issues detected during validation were immediately addressed, and the improved design worked as desired. Changes with respect to the proposed design include:

- The default python-control linearization method was overwritten to use central difference perturbation instead of the forward difference.
- The approximate eigenvalues from the reduced systems are not good approximations when auxiliary states are used. Therefore, matching them to augmented systems' eigenvalues is necessary. However, matching to a full system runs a risk of mixing up the eigenmodes.
- The Gibson criterion had to be evaluated on the reduced system with speed constant because the phugoid component made it hard to measure the dropback.
- The yaw damper design initially assumed only β and r feedback. That was based on the assumption that the reduced system with β and r would contain the Dutch Roll approximation. That assumption is false for the Flying-V, so the design was changed to 4-state feedback.

Table 9 Selected results for the pitch rate controller tuning. Requirements' abbreviations: PH = phugoid, SP-D = short period damping, CAP = Command Anticipation Parameter

CG option	Flight condition	Bare airframe							Gains		Closed loop						
		Modal Parameters				HQ Levels					Modal Parameters				HQ Levels		
		ω_{ph}	ζ_{ph}	ω_{sp}	ζ_{sp}	PH	SP-D	CAP	K_α	K_q	ω_{ph}	ζ_{ph}	ω_{sp}	ζ_{sp}	PH	SP-D	CAP
1	cruise	0.054	0.010	1.995	0.240	2	2	1	0.000	8.601	0.054	0.010	2.014	0.300	2	1	1
2	cruise	0.054	0.010	1.724	0.263	2	2	1	0.000	4.932	0.053	0.010	1.737	0.300	2	1	1
3	takeoff	0.097	0.004	0.982	0.711	2	1	2	6.194	0.000	0.101	0.001	1.029	0.683	2	1	1

Table 10 Selected results for the yaw damper tuning. Requirements' abbreviations: R = roll mode, S = spiral mode, DR = Dutch Roll

CG option	Flight condition	Bare airframe							Gains				Closed loop						
		Modal Parameters				HQ Levels							Modal Parameters				HQ Levels		
		T_r	T_s	ω_{dr}	ζ_{dr}	R	S	DR	K_ϕ	K_β	K_p	K_q	T_r	T_s	ω_{dr}	ζ_{dr}	R	S	DR
1	cruise	1.647	951.4	0.811	-0.011	2	1	None	4.192	0.000	16.53	-106.8	1.647	951.4	0.790	0.190	2	1	1
2	takeoff	1.051	-11903	0.803	-0.024	1	1	None	12.13	0.000	34.18	-125.6	1.051	-11903	0.771	0.130	1	1	1
3	approach	1.222	-63.97	0.898	-0.078	1	1	None	52.25	0.000	202.17	-353.8	1.222	-63.97	0.826	0.121	1	1	1

B. User Guide

A. Introduction

This is the user guide of the *fvlib* – a Python library for the automated handling qualities evaluation. The goal of this document is to describe the setup and provide a top-level overview of the library. For a specific description of functions and classes, refer to the source code documentation (the documentation is embedded in the code). To navigate the library source code, use the README.md files. Furthermore, you can look at the provided code examples. A good overview of the library's functionalities can also be found in the library's tests. Using *fvlib* requires basic knowledge of Python. Familiarity with the NumPy library will also be helpful. Furthermore, in many places, the library utilizes the python-control package, and it is therefore recommended to refer to its documentation as well[¶]. The library uses SI units (meters, Newtons, kilograms, radians, etc.).

B. Getting started

1. Installation

The library source code can be obtained from Ir. Olaf Stroosma [‖]. Alternatively, the repository is currently hosted at `https://github.com/adswid/FlyingV-Pipeline`. To get access, contact the author [**]. The library will be ready to use "as is" already. However, to run tests or code examples, two aircraft models: F-16 and Flying-V, have to be configured.

To use the F-16 model, you must build its nonlinear plant library. The Python implementation expects a shared library `libnlplant.*` (`.dll` for Windows and `.so` for Linux) in the `data\f16\build` directory. The `CMakeLists.txt` file is provided to build the plant using CMake [††]. The example build commands (Linux) should look as follows. In the `data\f16` directory:

```
mkdir build & cd build
cmake ..
make
```

To use the Flying-V model, you must manually add a few files. These are not a part of the repository because of data confidentiality. The files can be obtained from Ir. Olaf Stroosma [‖]. The folder `FlyingVData` contains the following items:

1) 3 JSON files: `FV_reset_xcg=*.HQ.json` – place them in the `data\flyingv` directory
2) `flyingv` directory – place it in the `tests\reference_data` folder

2. Repository structure

The top-level directory looks as follows:

- `data` – contains aircraft models used in code demos and tests
- `doc` – source files for autogenerated documentation
- `examples` – a collection of short and simple python scripts that demonstrate the library's functionalities
- `fvlib` – the actual library source
- `tests` – contains library tests and reference data

C. Running tests and demos

The code demos are simple stand-alone Python scripts. Run them in the IDE of your choice. Refer to the inline comments for an explanation of the steps and their results.

The tests were developed using pytest [‡‡] and were run from the PyCharm IDE. They can also be run from the command line, using `pytest` command in the repository's root directory, but the plots may not show. You can also run single test files (the SAS tuning tests take a long time).

[¶] `python-control.readthedocs.io`

[‖] O.Stroosma@tudelft.nl

[**] al.swiderski@gmail.com

[††] `https://cmake.org/`

[‡‡] `https://docs.pytest.org/`

```
import fvlib
def actuator_dynamics(model, t, x, u, params=None):
# Actuator dynamics function contents
# ...
    return control_derivatives

def forces_and_moments(model, t, x, u, params=None):
# Forces and moments function contents
# ...
    return forces, moments

def ca(model, t, x, u, params=None):
# Control allocation function contents
# ...
    return desired_controls

aircraft_spec = fvlib.AircraftSpec(mass=300000,
                                   tensor_of_inertia=[[1e7, 0, 0],[0, 1e6, 0],[0, 0, 1e7]]
                                   control_surfaces_count=6,
                                   control_dynamics=actuator_dynamics,
                                   auxiliary_states_count=0,
                                   forces_and_moments=forces_and_moments,
                                   control_allocation=ca,
                                   fcs=None,
                                   auxiliary_dynamics=None)
aircraft_model = fvlib.Model(aircraft_spec, use_vab=True)
```

Listing 1 Example of model implementation (does not represent any actual aircraft)

D. Creating an aircraft model

1. The fvlib.Model class

Your aircraft model will be represented by `fvlib.Model`. This class derives from the `control.NonlinearIOSystem`[§§]. This means that you can treat your aircraft model as an instance of that class and for example: calculate the derivatives of the state vector with the `dynamics` method or simulate time responses [¶¶]. The `fvlib.Model` adds a few additional elements to the `control.NonlinearIOSystem`, some of which will be later discussed. The source and complete documentation of the class is available in `fvlib\model.py`.

The most important part of the `fvlib.Model` is the `_model_rhs` – a main right-hand-side (RHS) function of the nonlinear system. This function manages the data flow between various user-defined and built-in parts of the model. The top-level overview of this function is presented in Figure 1 of the main paper.

2. The fvlib.AircraftSpec class

So how do you create and implement your own aircraft model? For that, we need to introduce the `fvlib.AircraftSpec` class. As mentioned in the main paper, the aircraft in the `fvlib` is defined by a set of functions describing elements of the model, together with a couple of parameters. `fvlib.AircraftSpec` is a structure to store these functions [***] and other parameters. The detailed description of the model elements you define in the `fvlib.AircraftSpec` is presented in subsection B.E, but for now, let us continue with model implementation.

[§§] `https://python-control.readthedocs.io/en/latest/generated/control.NonlinearIOSystem.html`

[¶¶] `https://python-control.readthedocs.io/en/latest/generated/control.input_output_response.html`

[***] In Python, functions are objects (unlike in, e.g., C++) and can be simply passed as arguments to function or stored as attributes in a class

3. Creating a fvlib.Model instance

Having created your model functions and parameters and filled the `fvlib.AircraftSpec` with them, you can now create your aircraft model. The `fvlib.AircraftSpec` is passed as the first argument in the `fvlib.Model` constructor, followed by other settings, such as the selection of velocity representation. This `fvlib.AircraftSpec` is then stored in the `fvlib.Model` instance and its elements are called by the class at appropriate moments. The simplified example of a model implementation is presented in Listing 1

E. Model definition specification (fvlib.AircraftSpec)

This section provides a complete description of (required and optional) fields of the `fvlib.AircraftSpec` structure, together with desired format or interface (for functions). If the field is marked as optional, it can be omitted. Otherwise, it must be specified. The fields can be divided into two parts: functions and parameters.

The following parameters can be set in the `fvlib.AircraftSpec`:

- `mass` – aircraft's mass in kg.
- `tensor_of_inertia` – tensor of inertia in $\mathrm{kg} \cdot \mathrm{m}^2$.
- `control_surfaces_count` – number of control surfaces.
- `auxiliary_states_count` (*optional, default: 0*) – number of auxiliary states.
- `thrust_limits` (*optional*) – minimum and maximum thrust as an array (*optional*). It is assumed that both engines have the same limits, so pass only one set of limits. Example: `[5000, 100000]`.
- `control_surfaces_limits` (*optional*) – minimum and maximum deflections per surface. Example: `[[-20,20], [-30,30], [-10,15]]`
- control surfaces limits policy (*optional, default: 'off'*) – choose 'soft' (exceeding limits is allowed, but warnings are raised), 'hard' (limits cannot be exceeded), or 'off' (limits are ignored).

Before the specification of the functions inside the `fvlib.AircraftSpec` will be presented, a class `fvlib.StateVector` has to be introduced. A state vector will be passed to every user function. The `fvlib.StateVector` subclasses `numpy.ndarray` (a typical NumPy array) to include convenience properties for easier access to the state's elements. For example, to get the roll rate p, you can call `state_vector[9]`, having to remember states' order, but you can also use `state_vector.body_rates[0]`. This is also helpful when trying to access control states. A second control state will not always be `state_vector[13]` because that depends on the number of auxiliary states, so it's easier to call `state_vector.control_states[1]`. Regardless, the `fvlib.StateVector` can still be treated as a regular array.

Having introduced the `fvlib.StateVector`, we can now discuss all the functions the user can provide in the `fvlib.AircraftSpec`. The presentation will roughly resemble the order of execution (some functions as evaluated in parallel). Use Figure 1 for reference.

- `fcs` (*optional*) – flight control system user function.
  Arguments:
  - `model` – the aircraft model (`fvlib.AircraftSpec`)
  - `t` – time (`float`)
  - `x` – state vector (`fvlib.StateVector`)
  - `u` – pilot input (`array`). Order: throttle left (1), throttle right (2), roll, pitch, yaw.
  - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns: augmented input (`array`). The output of this function is directly passed to the control allocation function, and therefore there is no convention on its structure. It can also contain additional variables (e.g., inputs to the auxiliary dynamics that must be forwarded through the control allocation).
- `control_allocation` – translates system inputs into thrust and control surface deflections.
  Arguments:
  - `model` – the aircraft model (`fvlib.AircraftSpec`)
  - `t` – time (`float`)
  - `x` – state vector (`fvlib.StateVector`)
  - `u` – pilot input (see above for order) if `fcs` is not implemented, otherwise `fcs` output
  - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns:
  - thrust and control surface deflections (in a single `array`). The order of control surface deflection is arbitrary. However, **the thrust values have to be placed before the control surface deflections**. The order must match the order of deflection limits and will also be reflected in the control state

vector if used. If the control dynamics are enabled, those values will be interpreted as *desired* thrust and deflections. If the control dynamics are disabled, those values will be interpreted as *actual* thrust and deflections.
    - input to auxiliary dynamics (*optional*)
- `forces_and_moments` – calculates forces and moments acting on the aircraft.
  Arguments:
    - `model` – the aircraft model (`fvlib.AircraftSpec`)
    - `t` – time (`float`)
    - `x` – state vector (`fvlib.StateVector`)
    - `u` – thrust and control surface deflections (`array`)
    - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns:
    - forces in body frame (`array`)
    - moments in body frame (`array`)
- `kinematics` (*optional*) – calculates derivatives of the kinematic states. The library provides a default implementation of this function. It is not recommended to overwrite it unless necessary.
  Arguments:
    - `model` – the aircraft model (`fvlib.AircraftSpec`)
    - `t` – time (`float`)
    - `x` – state vector (`fvlib.StateVector`)
    - `u` – thrust and control surface deflections (`array`)
    - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns: derivatives of kinematic states. Maintain the kinematic states order defined in `fvlib.StateVector`. Depending on the value of `model.use_vab`, return derivatives of aerodynamic velocity or body velocities.
- `control_dynamics` (*optional*) – calculates derivatives of the control states.
  Arguments:
    - `model` – the aircraft model (`fvlib.AircraftSpec`)
    - `t` – time (`float`)
    - `x` – state vector (`fvlib.StateVector`)
    - `u` – desired thrust and control surface deflections (`array`)
    - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns: Derivatives of the control states
- `auxiliary_dynamics` (*optional*) – calculates derivatives of the auxiliary states.
  Arguments:
    - `model` – the aircraft model (`fvlib.AircraftSpec`)
    - `t` – time (`float`)
    - `x` – state vector (`fvlib.StateVector`)
    - `u` – inputs to the auxiliary dynamics (`array`) – the second return value of the `control_allocation`. If `control_allocation` does not return a second value, an empty array will be passed as input
    - `params` – dictionary with additional parameters that can be passed during the RHS function.

  Returns: Derivatives of the control states

Lastly, it is possible to implement your model functions as methods inside the class so that they can share common data or for easier parametrization (see the implementation of the Flying-V in `data\flyingv\flyingv.py`). In that case, remember to add `self` as the first argument in each function.

F. Using the library

1. Aircraft model

Some of the functionalities of the `fvlib.Model` were already introduced. This subsection presents some additional elements.

We have discussed the RHS function of the model, but you can also call the output function: either by `output` (method of the `control.NonlinearIOSystem`) or `model_output_fcn` (method of the `fvlib.Model`). The output vector contains the kinematic states, auxiliary states, and control surface deflections. When the control dynamics are

enabled, the control surface deflections are states. However, when control dynamics are disabled, control surface deflections are neither states nor inputs. Therefore, they would be inaccessible to the user if not included in the output vector.

You can also add an additional output equation using the `add_output_eqn` method. This could be used for trimming (details in the Trim section of this guide) or to obtain variables that are not state variables, like the flight path angle γ. The new output equation should have the following signature:

```
eqn(aircraft: Model, t, x: StateVector, u, params) -> float
```

2. Automatic evaluation of handling qualities requirements

The fastest way to simply obtain the handling qualities predictions is to call the `fvlib.auto_eval`. This function will perform all necessary steps and print the results. Apart from passing your aircraft model, you need to provide the altitude, speed, and flight phase category. The example call looks like this:

```
fvlib.auto_eval(your_aircraft_model, altitude=0, speed=70, category='B')
```

3. Trim

To trim the aircraft, use the `fvlib.trim.trim` function. To trim your aircraft, pass the aircraft model (`fvlib.Model`), name of the flight condition (or custom trim class – see below), and keyword arguments with parameters of the trim condition. The example call looks like this:

```
x_eq, u_eq, y_eq, result = fvlib.trim.trim(aircraft_model,
                                           flight_condition='wings-level',
                                           altitude=2500,
                                           speed=150
                                           vertical_speed=-2)
```

For the wings-level flight condition, you have to pass altitude and speed. If you want to trim in steady climb or descent, include parameter `gamma` or `vertical speed`. The function returns trim values and trim results. You can also trim for the flight condition `steady-heading-sideslip`. That requires additional argument `beta` (sideslip angle).

Because the convergence of the trim may sometimes depend on the initial conditions, the user can pass them in the trim function as well (as keyword arguments). For the initial state vector, use `x0`, and for the initial input vector, use `u0`.

The library currently supports two flight conditions: 'wings-level' and 'steady-heading-sideslip'. If you wish to use different flight conditions, you can create a new class derived from `trim.TrimCondition`. In creating the class, you must implement two methods: `pre` and `post`. The first method is called before the trim optimization and takes the same arguments as the main `trim` function. In the `pre`, you should fill all of the trim specification constraints that will be passed to the `control.find_eqpt` function (see `TrimCondition.__init__` for the fields that you can fill and the `control.find_eqpt`[†††] documentation on how the trim condition is specified). The `post` function is called after the trim point has been found and can be used to do additional processing. When you have created your class, pass it as a `flight_condition` field in the `trim` function. Important: Pass the class, not the class instance! For examples of how the trim conditions are implemented, see `trim.py` and the implementation of `WingsLevel` and `SteadyHeadingSideslip` classes. To see how a custom trim condition can be created, refer to `WingsLevelF16` in `data\f16\f16.py`. Note that in that case, the new trim condition derives from `WingsLevel` class, not a `TrimCondition`, which allows reusing more of the existing code (do not forget about the `super()` call).

4. Linearization

To linearize the aircraft, call the `linearize` method of your aircraft model. Pass the linearization point's state and input. The example call looks like this:

```
aircraft_model.linearize(x_eq, u_eq)
```

[†††] https://python-control.readthedocs.io/en/0.9.3.post2/generated/control.find_eqpt.html

The linearize function returns the `control.LinearIOSystem`[‡‡‡]. Having obtained the linear system, you can use the library to get reduced systems. To do that, call the `fvlib.linear.reduce` function and specify the reduction type. You can also decide if the auxiliary or control states should be included or whether to have control surface deflections and thrust in the output equation. Refer to the code documentation for details.

5. Eigenmode Identification

To identify the eigenmodes, you need to provide the linearized **A** matrix, indicate whether the aerodynamic velocity is used, and if the system should match the approximated eigenvalues to the augmented system ones. The example call would be like this:

```
eigenmodes = fvlib.eigenmode.identify(a_matrix, use_vab=True,
                                      match_augmented_system = True)
```

The function returns an `Eigenmodes` object, which contains information about the five eigenmodes. Each field can be either of type `FirstOrderEigenmode` or `SecondOrderEigenmode`.

You can use the first and second-order eigenmode classes to create your own instances. You can define the first-order eigenmode by a pole value or a time constant. For example:

```
my_first_order_mode = eigenmode.FirstOrder(pole = -2)
```

will give the same results as:

```
my_first_order_mode = eigenmode.FirstOrder(time_constant = 0.5)
```

For a second order eigenmodes you create it using poles:

```
my_second_order_mode eigenmode.SecondOrder(-3 + 4j, -3 - 4j)
```

or using parameters:

```
my_second_order_mode = eigenmode.SecondOrder(omega=5, zeta=3/5)
```

6. HQ requirements

To run all of the HQ requirements, call the `fvlib.hqreq.run_all` function. The difference with the `fvlib.auto_eval` function is that it requires a linear system. The example call to the function looks like this:

```
fvlib.hqreq.run_all(linsys, speed, category, use_vab=True,
                    auxiliary_states_count=0)
```

You can also run each requirement individually. Refer to the code documentation for the required parameters. When the requirement yields a handling qualities level, that is what is returned by the function. In other cases, relevant values (like phi-to-beta) are returned. In all requirements, you can set the `verbose` argument for the function to print the intermediate and final results of the evaluation. In the short period step response and the Gibson criterion, you can also set the `plot` option to obtain the relevant time histories.

7. SAS tuning

To tune the stability augmentation system, call the `fvlib.sas.tune` function, and select the control channel ('lat', 'lon', or 'latlon') to tune. You also need to provide a trim point for which the gains should be tuned and the flight phase category. An example call looks like this:

```
fvlib.sas.tune('lat', aircraft_model, x_eq, u_eq, 'B', verbose=True)
```

The function returns the aircraft model (`fvlib.Model` type) with the flight control system implemented.

[‡‡‡] https://python-control.readthedocs.io/en/latest/generated/control.LinearIOSystem.html

C. Recommendations for future work

This section contains suggestions for the future development of the library. They include improvements to the original design, as well as expansion of the functionalities within the original scope and outside of it. Those recommendations are sorted by importance, starting with the most pressing actions/improvements and finishing with long-term expansions.

Use the library

The main recommendation for the future development of the library is to, first and foremost: use it, for instance, in the next Flying-V handling qualities research. This research project focused on scoping the desired functionalities of such a library, implementing them, and then validating the final product. However, the library was not yet used for any specific handling qualities research. Using it in practice will provide valuable insight into its potential shortcomings. It will also better highlight where the upgrades or expansions are much needed. And undoubtedly, it will uncover some bugs that were not caught in the testing phase.

Linear (state-space) model support

`fvlib.Model` implements and expands the `control.NonlinearIOSystem`. The same should be done for `control.LinearIOSystem`, which is used numerous times in later stages of the library. The `fvlib.Model` contains useful attributes like the number of auxiliary variables or control surfaces. Those parameters are also needed in functions that use linear models and must be passed separately. Including them in a single linear model class would simplify the function interfaces and improve the user experience.

Improve handling of the flight control system states

It is not uncommon that the flight control system introduces some states. It happens when the FCS has, for instance, integrators or filters. In the current model structure, these states are treated as auxiliary states. Therefore, their dynamics have to be handled in another function. Furthermore, the user has to pass the input to those dynamics through the Control Allocation function. This is an inconvenience for the user, and it is suggested to improve the model structure so that everything FCS-related is handled in one function.

Performance related requirements

As mentioned earlier, the library does not include performance-related requirements. However, many of the elements required for such analysis are already a part of the library. Given that such requirements were of interest in previous Flying-V handling qualities research, their inclusion in the library would be desirable. Furthermore, it is believed that the designed model framework, together with trim and linearization routines, constitutes a solid backbone upon which the library can be developed outside of the scope of handling qualities requirements to include other aspects of the analysis of preliminary aircraft designs.

Introduce auxiliary inputs

Auxiliary inputs were planned for the current version but were removed due to time constraints and limited utility in the first iteration of the library. However, as the Flying-V models would get more advanced, being able to introduce additional inputs, like, for example, flight spoiler lever position, would be appreciated by the user.

Integrate an FCS tuning system

The stability augmentation system tuning was introduced into the library as a proof-of-concept to demonstrate how the Flight Control System design/tuning can be integrated into the handling qualities evaluation process. It was also shown that discussions about the FCS are an integral part of the handling qualities evaluation process. However, it is not recommended to develop the module substantially. Instead, those efforts should be devoted to integrating an existing FCS tuning system or cooperation with Flying-V research that focuses purely on flight control system design.