Моделирование и анализ информационных систем. Т. 26, № 4 (2019), с. 488–501
Modeling and Analysis of Information Systems. Vol. 26, No 4 (2019), pp. 488–501

DOI: 10.18255/1818-1015-2019-4-488-501

UDC 004.052

Safety Analysis of Longitudinal Motion Controllers during Climb Flight

Baar T., Schulte H.

Received September 30, 2019
Revised November 19, 2019
Accepted November 27, 2019

Abstract. During the climb flight of big passenger airplanes, the airplane's vertical movement, i.e. its pitch angle, results from the elevator deflection angle chosen by the pilot. If the pitch angle becomes too large, the airplane is in danger of an airflow disruption at the wings, which can cause the airplane to crash. In some airplanes, the pilot is assisted by a software whose task is to prevent airflow disruptions. When the pitch angle becomes greater than a certain threshold, the software overrides the pilot's decisions with respect to the elevator deflection angle and enforces presumably safe values. While the assistance software can help to prevent human failures, the software itself is also prone to errors and is - generally - a risk to be assessed carefully. For example, if software designers have forgotten that sensors might yield wrong data, the software might cause the pitch angle to become negative. Consequently, the airplane loses height and can - eventually - crash.

In this paper, we provide an executable model written in Matlab/Simulink® for the control system of a passenger airplane. Our model takes also into account the software assisting the pilot to prevent airflow disruptions. When simulating the climb flight using our model, it is easy to see that the airplane might lose height in case the data provided by the pitch angle sensor are wrong. For the opposite case of correct sensor data, the simulation suggests that the control system works correctly and is able to prevent airflow disruptions effectively.

The simulation, however, is not a guarantee for the control system to be safe. For this reason, we translate the Matlab/Simulink®-model into a hybrid program (HP), i.e. into the input syntax of the theorem prover KeYmaera. This paves the way to formally verify safety properties of control systems modelled in Matlab/Simulink®. As an additional contribution of this paper, we discuss the current limitations of our transformation. For example, it turns out that simple proportional (P) controllers can be easily represented by HP programs, but more advanced PD (proportional-derivative) or PID (proportional-integral-derivative) controllers can be represented as HP programs only in exceptional cases.

Keywords: Cyber-Physical System (CPS), Formal Safety Analysis, Hybrid Automaton

For citation: Baar T., Schulte H., "Safety Analysis of Longitudinal Motion Controllers during Climb Flight", *Modeling and Analysis of Information Systems*, **26**:4 (2019), 488–501.

On the authors:
Thomas Baar, orcid.org/0000-0002-8443-1558,
Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany,
Campus Wilhelminenhof, Wilhelminenhofstraße 75A, 12459 Berlin, e-mail: thomas.baar@htw-berlin.de

Horst Schulte, orcid.org/0000-0001-5851-3616,
Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany,
Campus Wilhelminenhof, Wilhelminenhofstraße 75A, 12459 Berlin, e-mail: horst.schulte@htw-berlin.de

1. Flight Control Model of Longitudinal Motion

For a complete description of the airplane motion in the three dimensional space, six variables are needed that denote the degrees of freedom of a rigid body [1]. The airplane motion is calculable by six nonlinear ordinary differential equations (ODEs) of these variables. However, under certain assumptions, the ODEs can be decoupled and linearized into longitudinal and lateral equations. It is common practice to describe the longitudinal motion by a third order state space model [1], [2]:

$$x = [\,\alpha \;\; q \;\; \theta\,]^T \tag{1}$$

The state vector (1) contains the *angle of attack* α, *pitch rate* q, and *pitch angle* θ (cmp. Figure 1).

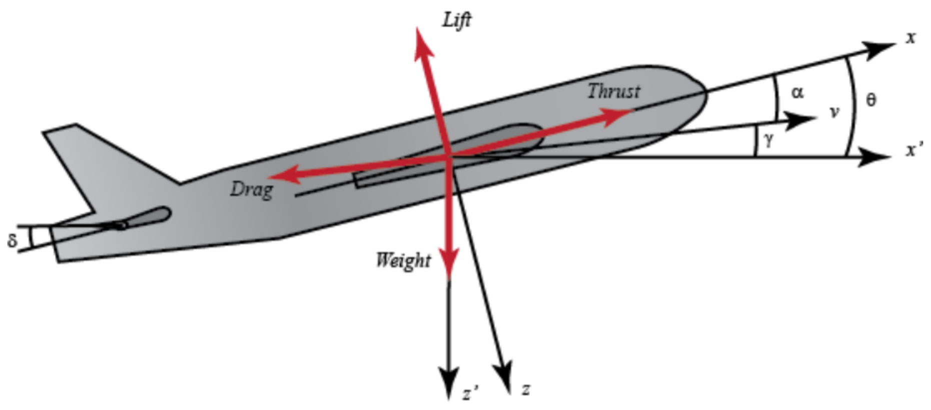

Fig 1. Important parameters of flight model
(Source: *http://ctms.engin.umich.edu/CTMS/index.php?example=AircraftPitch &section=SystemModeling*)

Based on the assumption that the airplane is in steady-cruise at constant velocity, the longitudinal equations of motion for the airplane in state space form $\dot{x} = f(x, u)$ with the state vector given in (1) and the input $u := \delta$ can be written as

$$\begin{aligned}
\dot{\alpha} &= \mu\Omega\sigma\big[-(C_L + C_D)\alpha + \frac{1}{(\mu - C_L)}q - (C_W \sin\gamma)\theta + C_L\big] \\
\dot{q} &= \frac{\mu\Omega}{2I_{yy}}\Big(\big[C_M - \eta(C_L + C_D)\big]\alpha + \big[C_M + \sigma C_M(1 - \mu C_L)\big]q + (\eta C_W \sin\gamma)\delta\Big) \\
\dot{\theta} &= \Omega q
\end{aligned} \tag{2}$$

where

$$\Omega = \frac{2U}{\bar{c}}, \qquad \mu = \frac{\rho S\bar{c}}{4m}, \qquad \sigma = \frac{1}{1 + \mu C_L}, \qquad \eta = \mu\sigma C_M, \tag{3}$$

with the equilibrium flight speed U and γ as the flight path angle. The parameter ρ denotes the density of air, S denotes the platform area of the wing, $\bar{c}$ denotes the average chord length and m denotes the mass of the airplane, C_W denotes the coefficient of weight, C_M denotes coefficient of pitch moment, and I_{yy} denotes the normalized moment of inertia. The aerodynamic coefficients of thrust, drag and lift are C_T, C_D, C_L. Based on the above assumptions, the dynamics of the airplane around a stationary operating point $p_c = (\alpha_c, q_c, \theta_c, \delta_c)$ for an equilibrium flight speed is obtained by Taylor linearization of $f(x,u)$

$$A = \frac{\partial f}{\partial x}|_{p_c} = \begin{pmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{pmatrix}, \qquad B = \frac{\partial f}{\partial u}|_{p_c} = \begin{pmatrix} 0.232 \\ 0.0203 \\ 0 \end{pmatrix} \tag{4}$$

and can be described as follows

$$\begin{aligned} \dot{\alpha} &= -0.313\,\alpha && +56.7\,q && +0.232\,\delta \\ \dot{q} &= -0.0139\,\alpha && -0.426\,q && +0.0203\,\delta \\ \dot{\theta} &= && \;\;56.7\,q && \end{aligned} \tag{5}$$

2. Control loop designed in Matlab/Simulink®

2.1. Description of the designed structure

For the flight model introduced above, we have developed a series of controllers using Matlab/Simulink® whose aim is to keep the pitch angle θ below a maximum value θ_{max} to prevent airflow disruption at the wings. We have selected the period of a climb flight and assume, that the pilot selects a constant deflection angle δ_{man} as manual input, what might cause pitch angle θ to increase. If θ becomes greater than an upper bound, the anti-stall mode is activated and the controller computes a corrective δ_{corr}. When – as a consequence – θ falls again below a lower bound (due to hysteresis, the lower bound is slightly different from upper bound), the anti-stall mode is switched off again and the pilot's δ_{man} become again the input for the airplane. An overview of the entire control system is shown in Figure 2.

Our closed loop control system follows the classical approach and can be split into a plant model and a controller system. The plant model is depicted in Figure 2 by the orange block (1) with input *delta* and outputs q and *theta*. This block realizes the linearized airplane model specified by equation Eq.(5).

The controller system consists of a total of three blocks (cmp. Figure 2) including the following functions: (2) Computation of the anti-stall mode, (3) computation of the delta correction value δ_{corr}, and (4) selection of the delta value δ as actuating input of the airplane model block (1).

The decision making in block (2) is based on a comparison between the currently measured pitch angle θ and the maximum value θ_{max}. To avoid chattering, a distinction is made between a lower $\theta_{max,low}$ and an upper threshold $\theta_{max,up}$. Thereby the anti-stall mode is activated if $\theta > \theta_{max,up}$ is valid and is deactivated again if $\theta < \theta_{max,low}$ is fulfilled.

Block (3) realizes the computation of the delta correction value δ_{corr} by applying the simple proportional (P) control law

$$\delta_{corr} = k_p\, e \tag{6}$$

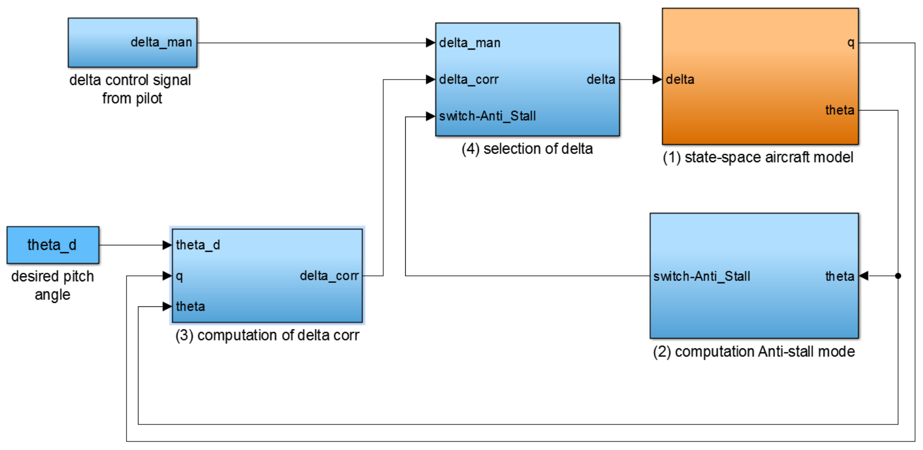

Fig 2. Overview of the entire control system designed in Matlab/Simulink®

where $e = \theta_d - \theta$ denotes the control error and $k_p > 0$ denotes the real-valued controller gain.

In many cases, a system behaves smoother and has more performance if instead the simple P-control law the PD (proportional-derivative) or the PID (proportional-integral-derivative) control law is applied. While the implementation of the PD / PID control law within block (3) in Matlab/Simulink® can be done easily, the transformation of this new Matlab/Simulink®-model into a hybrid program (HP) as input for the theorem prover KeYmaera as described in Section 3. faces some serious problems. These problems are discussed in Section 4.

In block (4), the delivery of the actuating signal either from the manual demand of the pilot δ_{man} or from the anti-stall controller δ_{corr} takes place. The implementation of block (4) is illustrated in detail in Figure 3, where the switching between δ_{man} and δ_{corr} is realized by a crisp mode-dependent function.

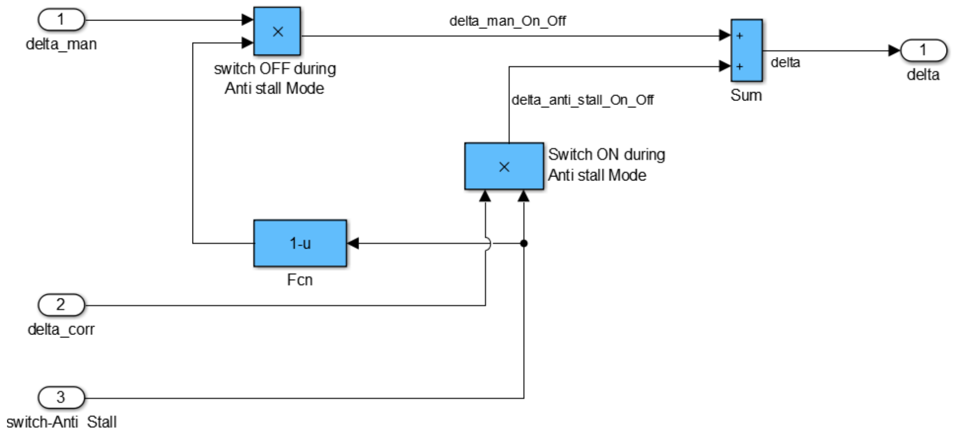

Fig 3. Control block (4) of Figure 2: Selection of the delta value by crisp function

2.2. System analysis by simulation

The standard technique to analyse Matlab/Simulink®-models is *simulation*. For a simulation, concrete values for constants (e.g. k_p) and for input parameters (δ_{man}, θ_d) are chosen and the system is executed. The system behaviour can be depicted by function graphs showing the values of system variables over time (cmp. Figure 4, Figure 5).

We actually simulate two versions of the system: the first version assumes correct measurement of the output θ of block (1), as described above and as depicted in Figure 2.

In the second simulation, we assume that output θ of block (1) is not measured correctly by sensors. This is modelled by applying an error function on θ before it becomes the input for block (2) and block (3). Applying error functions is a standard modelling technique in Matlab/Simulink®.

2.2.1. Assuming correct sensor measuring for θ

The pitch angle θ is one of the outputs of the plant model realized by block (1) and the input for the control loop consisting of blocks (2-4). Based on θ, the input δ (pitch elevator angle) is computed for the next cycle.

Assuming that the angle θ is correctly measured by sensors, the simulation of the system as shown in Figure 4 does not detect any point in time in which pitch angle θ becomes negative. Note that if pitch angle θ would become negative, the airplane would lose height and might eventually crash.

Observing only non-negative values for θ in a simulation is not a guarantee that θ never becomes negative, but it is already a good starting point for formal verification of the system's safety (cmp. Section 3.).

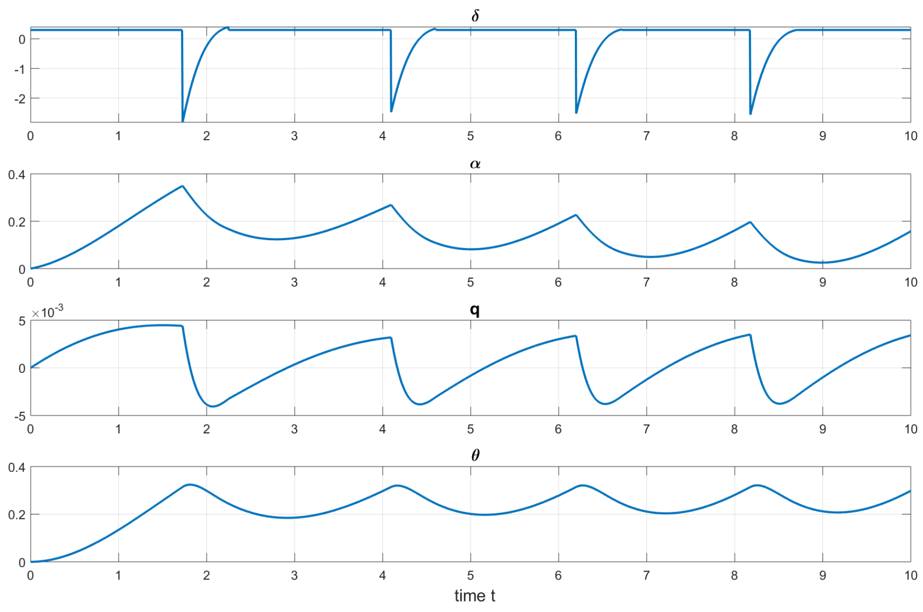

Fig 4. Simulation results for correct (fault free) sensor measuring of θ

2.2.2. Assuming incorrect sensor measuring for θ

When building safety critical systems, engineers should always take into account that sensors might provide wrong data. We have modelled in a second Matlab/Simulink®-model a faulty sensor just by substituting the output θ of the plant model in block (1) by $\theta + \theta_{\text{offset}}$, where θ_{offset} is a predefined constant. When simulating this second model, it can be immediately seen that θ becomes soon negative, i.e. the airplane can lose height. This simulation is shown in Figure 5.

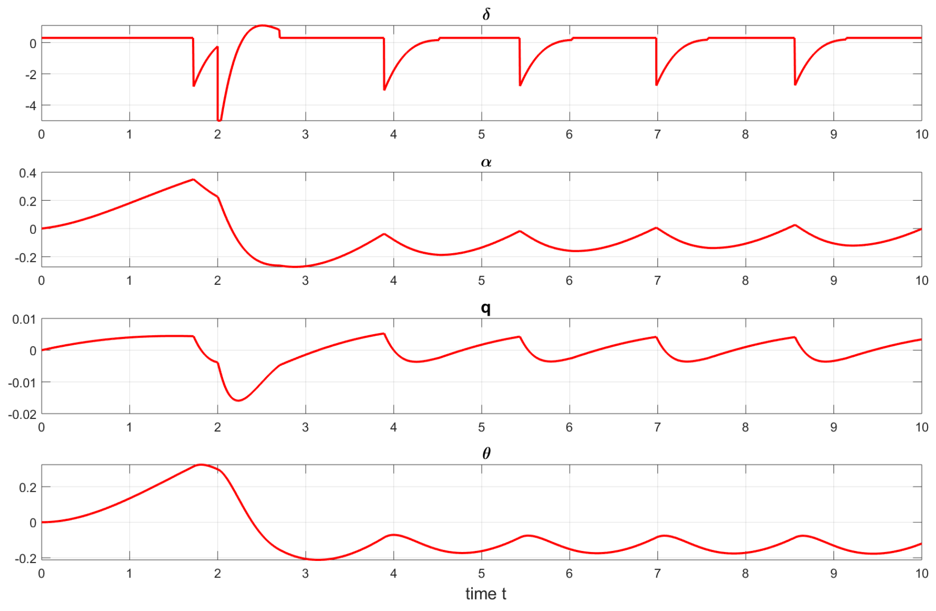

Fig 5. Simulation results for incorrect sensor measuring of θ

3. Logical Analysis of Flight Control Models

As detailed in the previous section, the Matlab/Simulink® toolkit is able to simulate the modelled system. As one can easily see, the airplane might lose height when sensors for measuring pitch angle θ provide wrong data. However, for the opposite case of having a (presumably) correct system, simulation is not a sufficient technique to ensure the correct system behaviour under all possible circumstances. In our case, the correct behaviour means that pitch angle θ remains always positive (recall that we model the phase of climb flight).

In this section, we present a translation of the Matlab/Simulink® model into a hybrid program (HP), a notion similar to well-known hybrid automata [7]. Since the notion of HP is supported by the theorem prover KeYmaera [4, 3], our transformation paves the way to formally verify safety properties of hybrid systems [6, 5].

3.1. KeYmaera

A proof task for KeYmaera has to be formulated in *differential dynamic logic* (DDL), which is an extension of classical *dynamic logic* (DL) [9].

Classical DL allows to prove pre-/post-condition contracts (known from Design-by-Contract) for programs written in a simple while-language, i.e. a programming language supporting the classical concepts of imperative programming, such as *assignment to a variable*, *sequential execution*, *iterative execution*, and *case distinction*. The programming language handled by the theorem prover KeYmaera - known as *Hybrid Program* (HP) - also supports to a certain degree nondeterministic execution, as realized by the statements *nondeterministic assignment* (`x = *`), *nondeterministic choice* ($\alpha \cup \beta$), and *nondeterministic iteration* ($(\alpha)*$).

Technically, classical DL is a modal logic with modalities *box* ($[\alpha]\psi$) and *diamond* ($< \alpha > \psi$), where α is a program and ψ a logical formula expressing a property on the state of the machine, on which the program is executed. The state of the machine is defined as the value vector of all program variables occurring in program α. Note that in HP all variables are of type Real and represent only one single scalar value. This is an important difference to Matlab/Simulink®, where a variable can be of type vector or matrix and can represent a list of scalar values or even a (numerically represented) function.

For the rest of the paper, only the box-modality is applied; the formula $[\alpha]\psi$ states that in each possible post-state after program α has terminated the formula ψ holds. Please note that termination of α is not claimed! Please further note that since α can contain nondeterministic statements the execution of α can indeed result in multiple post-states.

The main difference of DDL and DL is that the former supports *continuous evolution* statements. When a continuous evolution statement is executed, the variables $x_1, x_2, \ldots, x_n$ selected by the statement change their values synchronously according to coupled differential equations. The value change is restricted by an optional *evolution constraint* H. Table 1 summarizes the basic statements of KeYmaera's programming language HP.

Table 1. Statements of HP (adapted from [6])

HP Notation	Operation	Effect
$x_1 := \theta_1, \ldots, x_n := \theta_n$	discrete jump	simultaneously assign θ_i to variable x_i
$x := *$	nondet. jump	assign any value to variable x
$x'_1 := \theta_1, x'_2 := \theta_2, \ldots$ $\ldots, x'_n := \theta_n \;\&\; H$	continuous evo.	differential equations for x_i within evolution domain H (first-order formula)
$?H$	state test	test first-order formula H at current state
$\alpha; \beta$	seq. composition	HP β starts after HP α finishes
$\alpha \cup \beta$	nondet. choice	choice between alternatives HP α or HP β
$\alpha*$	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}_0$

For a detailed introduction to DDL, the reader is referred to [6]. In [8], the limitations of HP with respect to expressiveness and maintainability is investigated.

3.2. Flight model as KeYmaera-input

Table 2. AirplaneClimbControl (ACC)

$ACC \equiv$	$(ctrl; plant)*$	(7)
$ctrl \equiv$	$block_2; block_3; block_4$	(8)
$block_2 \equiv$	$(?\theta > \theta_{max,up}; switch_{AntiStall} := 1)\cup$	(9)
	$(?\theta < \theta_{max,low}; switch_{AntiStall} := 0)\cup$	(10)
	$(?(\theta \leq \theta_{max,up} \wedge \theta \geq \theta_{max,low}))$	(11)
$block_3 \equiv$	$\delta_{corr} = k_p * (\theta_d - \theta)$	(12)
$block_4 \equiv$	$\delta = switch_{AntiStall} * \delta_{corr} + \delta_{man} * (1 - switch_{AntiStall})$	(13)
$plant \equiv$	$t := 0;$	(14)
	$(t' = 1,$	(15)
	$\alpha' = -0.313 * \alpha + 56.7 * q + 0.232 * \delta,$	(16)
	$q' = -0.0139 * \alpha - 0.426 * q + 0.0203 * \delta,$	(17)
	$\theta' = 56.7 * q$	(18)
	$\& \; t <= \epsilon)$	(19)

The Matlab/Simulink®-model shown in Figure 2 can be translated into a hybrid program *ACC* as shown in Table 2. In line (7), the overall structure of *ACC* is shown as the nondeterministic iteration (operator $*$) over the sequential composition (operator ;) of subsystems *ctrl* and *plant*. The control part *ctrl* in line (8) is sequentially composed of $block_2 - block_4$, while these HP programs tightly correspond to the blocks (2), (3), (4) in the Matlab/Simulink®-model shown in Figure 2.

In $block_2$, the flag $switch_{AntiStall}$ is either switched on (line (9)) or switched off (line (10)) after comparing the current value of pitch angle θ with predefined threshold values $\theta_{max,up}, \theta_{max,low}$. The program structure of $block_2$ is basically an if-then-else and therefore we have to specify in line (11) that in all remaining cases the value of $switch_{AntiStall}$ remains the same.

In $block_3$, the correction value δ_{corr} is computed in terms of a simple proportional (P) controller (line (12)).

In $block_4$, the actual plant input δ is computed by switching between δ_{corr} and δ_{man} depending on the value of $switch_{AntiStall}$ (line (13)).

Subprogram *plant* is the sequential composition of the reset of auxiliary variable t to 0 (line (14)) and a continuous evolution statement (lines (15) - (19)). This continuous evolution statement encodes exactly the linearized airplane model specified in equation Eq.(5). The domain constraint $t < \epsilon$ (line (19)) together with the ODE $t' = 1$ (line (15)) has the effect that the system remains at longest time ϵ in the evolution state.

3.3. Proof task for correct behaviour

We can now formally formulate the safety property we would like to show for our hybrid program ACC:

$$\theta > 0 \to [ACC]\,\theta > 0 \tag{20}$$

In words, (20) reads as: whenever the system (including its controller) is started in a situation in which the pitch angle is positive, then after every control loop (which takes mostly time ϵ), the pitch angle remains positive.

Note that it is not the goal of this paper to actually establish a formal proof for (20) using the theorem prover KeYmaera. This task has been postponed to future work.

4. Lessons Learned

In Section 3.2. we presented based on an example a transformation of Matlab/Simulink®-models into input artefacts for the theorem prover KeYmaera, which are written in HP

Source and target notation of this transformation have a semantic gap that cannot always be bridged by a clever encoding in the target notation HP. The most striking difference are the allowed types for variables and the predefined operations on these types. In KeYmaera, every variable is of type *Real*, for which only common arithmetic operation such as $+\,,-\,,*\,,/$ and comparison operation $<\,,\leq\,,>\,,\geq\,,=$ are provided. In Matlab, variables can be of a numerical type or of an n-dimensional list (aka. vector, matrix, grid) or even of a function type. The list of operations supported by these types is much longer than in HP: arithmetic operation, trigonometric function, matrix multiplication, and many more.

Another important difference is that in HP the derivation operator ' can only be applied in a continuous evolution statement, whose ODEs are restricted to the form $x_i' = \theta_i$, while θ_i must not contain any derivative term. In Matlab, the derivation operator can be applied much more freely, e.g. in the plant model as well as in the controller system.

Let us illustrate these problems with a concrete example. In our Matlab/Simulink®-model presented above, the computation of δ_{corr} is done by applying the P control law $\delta_{corr} = k_p\, e$ (cmp. Eq. (6)). This law is transformed to $block_3$ with implementation $\delta_{corr} = k_p * (\theta_d - \theta)$ (comp. line (12) in Table 2).

Suppose, we would like to substitute in the Matlab/Simulink®-model the P by a PD control law, which would be implemented by

$$\delta_{corr} = k_p\, e + k_d\, \dot{e} \tag{21}$$

where $e = \theta_d - \theta$ denotes the control error and $k_p > 0$ and $k_d > 0$ denote real-valued controller gains. For such a system definition, our transformation would yield for $block_3$ the implementation

$$\delta_{corr} = k_p * (\theta_d - \theta) + k_d * (\theta_d - \theta)^{\cdot} \tag{22}$$

The problem now is that Eq. (22) cannot be directly transformed into a HP since the derivation operator ' must only occur in form $var' = \langle\exp\rangle$ within a continuous evolution statement and this form is not met by Eq. (22).

Fortunately, since θ_d is a predefined constant, we can now rewrite within Eq. (22) subterm $(\theta_d - \theta)^{\cdot}$ by $-\dot{\theta}$ and get

$$\delta_{corr} = k_p * (\theta_d - \theta) - k_d * \dot{\theta} \tag{23}$$

Now we remember the last line of Eq.(5) $\dot{\theta} = 56.7\, q$ and get finally

$$\delta_{corr} = k_p * (\theta_d - \theta) - k_d * 56.7 * q \tag{24}$$

If the PD control law is formulated in this form, it can be directly transformed into HP since not derivation operator is applied any longer.

5. Related Work

Safety analysis of flight control systems has a long tradition in the academic control community. Based on linear models which describe the dynamics of the aircraft for individual degrees of freedom, formal methods of control theory such as stability analysis, model-based fault detection and isolation (FDI) and fault tolerant control (FTC) are applied [16],[29]. Due to the nonlinear behaviour of aircraft dynamics, the assumption of an approximate description with linear models is only valid in a small working range. In the case of an fault, however, this range is left. For this reason, model classes such as quasi-LPV (linear parameter variable) [30] and Takagi-Sugeno systems have been investigated in recent years. These techniques allow to describe nonlinear dynamics in such a way that formal safety system analysis is feasible [31]. In order to be able to compare the different approaches in an objective way, benchmarks have been established in cooperation with industry [20], [17], [18]. For this purpose, generic flight models or models that describe common target aircraft are used.

For the safety analysis of a longitudinal motion controller, a logic based verification technique has been chosen in this paper. The core of our approach is based on differential dynamic logic (DDL) [3], for which the prover KeYmaera has been developed. Numerous case studies from different domains have been conducted, in which KeYmaera proved to be powerful enough to verify real-world cyber-physical systems [22], [23], [24].

An alternative logic-based verification backend would be HyComp [27], which is also able to verify hybrid automata. HyComp is an SMT-based model checker and expects a system description in terms of the HyDI symbolic language.

One of the bottlenecks for applying KeYmaera is the necessity to describe the system (both plant and controller) in form of a hybrid program, which is a rather archaic, text oriented notation. In this paper, we have provided – based on an example – a translation from the notation Matlab/Simulink® into a hybrid program. While this translation is for many constructs straightforward, special attention is needed to translate how the controller computes the corrective input for the plant. In practice, this is often done by applying a P-, PD-, or PID-controller law.

In [6], the integration of a PD-controller is illustrated in Example 9b (page 18), but the PD-controller becomes part of the plant and not of the controller. Thus, to our knowledge, it is still an unsolved problem how to model PD- and PID-controller in terms of hybrid programs.

Instead of using logic-based verification methods, tools such as SpaceEx [25] and Flow* [26] apply *reachability analysis* as verification method. The goal of reachability analysis is to obtain a verification whether a hybrid automaton never reaches an unsafe state configuration [11]. In contrast to logic analysis, the verification of the reachability for hybrid systems is based on the level set techniques [12], which determines an implicit representation of the boundary of this reachable set.

For systems with complex dynamics, some approximation methods are needed for reachability computations [13]: A group of methods seeks an efficient over-approximation of the reachable set for hybrid systems, whose continuous dynamics is defined by linear differential equations. Such systems can be implemented using tools like *d/dt* [14] or the MATLAB-based tool *CheckMate* [28]. Here, sets are represented as convex polyhedra. The propagation of these polyhedra under linear dynamics could result in over-approximations of nonlinear dynamics along each surface of the polyhedra.

Conclusion

In this paper, we have investigated safety critical software for controlling the flight of modern airplanes. Such control software is usually developed using tools such as Matlab/Simulink®. We present a possible controller for the computation of the pitch elevator angle, but this controller has been completely designed by ourselves. The sole purpose of our model is to provide an example at which quality assurance techniques can be applied and demonstrated.

For the controller of our example, we review two main safety properties: Does the controller effectively prevent airflow disruption, which is the main purpose of the controller. However, there is another safety property, which can be easily overlooked when airplane software is hastily developed: Is it possible that the controller software could cause the airplane to lose height, which - eventually - might cause the airplane to crash.

The simulations of our controller suggest, that both safety properties are met. However and not surprisingly, the controller can cause airplane crashes when the sensor measuring the pitch angle θ does not provide correct data.

For the case that the sensor works correctly, we have translated the Matlab/Simulink®-model successfully into a HP model. This paves the way to verify using the theorem prover KeYmaera that for all possible situations the system is safe (not only for the few situations captured by the simulation). Finding an actual proof for the described safety properties was not the goal of this paper.

As discussed in Section 4., a successful transformation is only possible if the source model meets some restrictions. Whenever functions are used in the source model for which there is no corresponding function in HP, the generated target model cannot be parsed by KeYmaera. Likewise, the derivation operator should be used in the source model only at locations, which are mapped to continuous evolutions statements in HP. As the example given in Section 4. illustrates, all other occurrences of the derivation operator have to be manually substituted by equivalent terms prior to transformation, but this seems to be not always possible.

References

[1] Stengel R. F., *Flight Dynamics*, Princeton University Press, 2004.

[2] Yechout T. R., *Introduction to Aircraft Flight Mechanics*, American Institute of Aeronautics & Astronautics, 2003.

[3] Platzer A., *Logical Foundations of Cyber-Physical Systems*, Springer, Heidelberg, 2018.

[4] Platzer A., *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer, Heidelberg, 2010.

[5] Platzer A., "Logic and Compositional Verification of Hybrid Systems (Invited Tutorial)", *In: Gopalakrishnan G., Qadeer S. (eds) Computer Aided Verification. CAV 2011. LNCS. Springer, Berlin, Heidelberg*, **6806** (2011), 28–43.

[6] Quesel J. D., Mitsch S., Loos S., Aréchiga N., Platzer A., "How to Model and Prove Hybrid Systems with KeYmaera: A Tutorial on Safety", *International Journal on Software Tools for Technology Transfer*, **18**:1 (2016), 67–91.

[7] Henzinger T. A., "The Theory of Hybrid Automata", *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA*, 1996, 278–292.

[8] Baar T., "A Metamodel-Based Approach for Adding Modularization to KeYmaera's Input-Syntax", *Proceedings, 11th A. P. Ershov Informatics Conference, Akademgorodok, Novosibirsk, Russia. 2019.*

[9] Harel D., Kozen D., Tiuryn J., *Dynamic Logic*, MIT Press Cambridge, 2000.

[10] Frehse G., Kekatos N., Nickovic D., Oehlerking J., Schuler S., Walsch A., Woehrle M., "A Toolchain for Verifying Safety Properties of Hybrid Automata via Pattern Templates", *Proceedings, Annual American Control Conference (ACC), Milwaukee, USA*, 2018, 2384–2391.

[11] Alur R., Courcoubetis C., Henzinger T. A., Ho P. H., "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems", *International Hybrid Systems Workshop. LNCS, Springer, Berlin, Heidelberg*, **736** (1991), 209–229.

[12] Osher S., Sethian J. A., "Fronts Propagating with Curvature-dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", *Journal of Computational Physics*, **79**:1 (1988), 12–49.

[13] Tomlin C. J., Mitchell I., Bayen A. M., Oishi M., "Computational Techniques for the Verification of Hybrid Systems", *Proceedings of the IEEE*, **91**:7 (2003), 986–1001.

[14] Asarin E., Bournez O., Dang T., Maler O., "Approximate Reachability Analysis of Piecewise-Linear Dynamical Systems", *In: Lynch N., Krogh B.H. (eds) Hybrid Systems: Computation and Control. HSCC 2000. LNCS, Springer, Berlin, Heidelberg*, **1790** (2000), 20–31.

[15] Clarke Jr. E. M., Grumberg O., Kroening D., Peled D., Veith H., *Model Checking (second edition)*, MIT Press, 2018.

[16] Chen J., Patton R. J., *Robust Model-Based Fault Diagnosis for Dynamic Systems*, Kluwer Academic Publishers Norwell, MA, USA, 1999.

[17] Goupil P., Marcos A., *Advanced Diagnosis for Sustainable Flight Guidance and Control: The European ADDSAFE Project*, SAE technical paper 2011-01-2804, 2011.

[18] Goupil P., Marcos A., "Industrial Benchmarking and Evaluation of ADDSAFE FDD Design", *In Proc. of 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, **45**:20 (2012), 1131–1136.

[19] Grenaille S., Henry D., Zolghadri A., "A method for designing fault diagnosis filters for LPV polytopic systems", *Journal of Control Science and Engineering*, 2008.

[20] Christopher E., Thomas L., Hafid S., "Fault Tolerant Flight Control: A Benchmark Challenge", *Lecture Notes in Control and Information Sciences*, **399** (2010).

[21] Witczak M., "Fault Diagnosis and Fault-Tolerant Control Strategies for Non-Linear Systems", *Lecture Notes in Electrical Engineering, Springer*, **266** (2014), 375–392.

[22] Mitsch S., Loos S. M., Platzer A., “Towards Formal Verification of Freeway Traffic Control”, *In Proc. of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems(ICCPS)*, 2012, 171–180.

[23] Jeannin J. B., Ghorbal K., Kouskoulas Y., Gardner R., Schmidt A., Zawadzki E., Platzer A., “A Formally Verified Hybrid System for the Next-Generation Airborne Collision Avoidance System”, *In Proc. of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015, 21–36.

[24] Platzer A., Quesel J. D., “European Train Control System: A Case Study in Formal Verification”, *In: Breitman K., Cavalcanti A. (eds) Formal Methods and Software Engineering. ICFEM 2009. LNCS, Springer*, **5885** (2009), 246–265.

[25] Frehse G., Le Guernic C., Donzé A.,Cotton S., Ray R., Lebeltel O., Ripado R., Girard A., Dang Th. Maler O., “SpaceEx: Scalable Verification of Hybrid Systems”, *In: Gopalakrishnan G., Qadeer S. (eds) Computer Aided Verification. CAV 2011. LNCS, Springer*, **6806** (2011), 379–395.

[26] Chen X., Ábrahám E., Sankaranarayanan S., “Flow*: An Analyzer for Non-linear Hybrid Systems”, *In: Sharygina N., Veith H. (eds) Computer Aided Verification. CAV 2013. LNCS, Springer*, **8044** (2013), 258–263.

[27] Cimatti A., Griggio A., Mover S., Tonetta S., “HyComp: An SMT-Based Model Checker for Hybrid Systems”, *In: Baier C., Tinelli C. (eds) Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, Springer*, **9035** (2015), 52–67.

[28] “Formal Verification of Hybrid Systems Using CheckMate: A Case Study”, *Proceedings of the 2000 American Control Conference*, **3** (2000), 1679–1683.

[29] Zolghadri A., “Advanced Model-Based FDIR Techniques for Aerospace Systems: Today Challenges and Opportunities”, *Progress in Aerospace Sciences, Elsevier*, **53** (2012), 18–29.

[30] Grenaille S., Henry D., Zolghadri A., “A Method for Designing Fault Diagnosis Filters for LPV Polytopic Systems”, *Journal of Control Science and Engineering*, 2008, 1–11.

[31] Witczak M., Dziekan L., Puig V., Korbicz J., “Design of a Fault-Tolerant Control Scheme for Takagi-Sugeno Fuzzy Systems”, *In Proc. 16th Mediterranean Conference on Control and Automation*, 2008, 280–285.

Баар Т., Шульте Х., "Анализ безопасности контроллеров продольного движения во время набора высоты", *Моделирование и анализ информационных систем*, **26**:4 (2019), 488–501.

DOI: 10.18255/1818-1015-2019-4-488-501

Аннотация. Во время набора высоты на больших пассажирских самолетах вертикальное движение самолета, то есть его угол наклона, зависит от угла отклонения руля высоты, выбранного пилотом. Если угол наклона становится слишком большим, самолет рискует нарушить воздушный поток на крыльях, что может привести к его падению. В некоторых самолетах пилоту помогает программное обеспечение, задачей которого является предотвращение нарушения воздушного потока. Когда угол наклона становится больше определенного порога, программное обеспечение отменяет решения пилота относительно угла отклонения руля высоты и обеспечивает предположительно безопасные значения. Хотя вспомогательное программное обеспечение может помочь предотвратить человеческие сбои, само программное обеспечение также подвержено ошибкам и, как правило, представляет собой риск для тщательной оценки. Например, если разработчики программного обеспечения забыли, что датчики могут давать неправильные данные, программное обеспечение может привести к тому, что угол наклона станет отрицательным. Следовательно, самолет теряет высоту и может – в конечном итоге – разбиться.
В этой статье мы представляем исполняемую модель, написанную на Matlab/Simulink® для системы управления пассажирским самолетом. Наша модель также учитывает программное обеспечение, помогающее пилоту предотвращать нарушение воздушного потока. При моделировании

набора высоты с использованием нашей модели легко увидеть, что самолет может потерять высоту, если данные, предоставленные датчиком угла наклона, неверны. Для противоположного случая правильных данных датчика, моделирование предполагает, что система управления работает правильно и способна эффективно предотвращать нарушение воздушного потока.
Однако симуляция не является гарантией безопасности системы управления. По этой причине мы переводим Matlab/Simulink®-модель в гибридную программу (HP), т. е. во входной синтаксис средства доказательства теорем KeYmaera. Это открывает путь для формальной проверки свойств безопасности систем управления, смоделированных в Matlab/Simulink®. В качестве дополнительного вклада в эту статью мы обсудим текущие ограничения нашей трансформации. Например, оказывается, что простые пропорциональные (P) контроллеры могут быть легко представлены программами HP, но более продвинутые контроллеры PD (пропорционально-производные) или PID (пропорционально-интегрально-производные) могут быть представлены как программы HP только в исключительных случаях.

Ключевые слова: киберфизическая система (CPS), анализ формальной безопасности, гибридный автомат

Об авторах:
Томас Баар, orcid.org/0000-0002-8443-1558,
Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany,
Кампус Wilhelminenhof, Wilhelminenhofstraße 75A, 12459 Берлин, e-mail: thomas.baar@htw-berlin.de

Хорст Шульте, orcid.org/0000-0001-5851-3616,
Hochschule für Technik und Wirtschaft (HTW) Berlin, Germany,
Кампус Wilhelminenhof, Wilhelminenhofstraße 75A, 12459 Берлин, e-mail: horst.schulte@htw-berlin.de