UDK 681.51, 004.05

Safety Analysis of Longitunal Motion Controllers during Climb Flight

Baar T. (Hochschule für Technik und Wirtschaft (HTW) Berlin, Department of Engineering I)

Schulte H. (Hochschule für Technik und Wirtschaft (HTW) Berlin, Department of Engineering I)

During the climb flight of big passenger planes, the pilot directly adjusts the pitch elevator and the plane reacts on this by changing its pitch angle. However, if the pitch angle becomes too large, the plane is in danger of an airflow disruption on the wings, which can cause the plane to crash. In order to prevent this, modern planes take advantage of control software to limit the pitch angle. However, if the software is poorly designed and if system designers have forgotten that sensors might yield wrong data, the software might cause the pitch angle to become negative, so that the plane loses height and can - eventually - crash.

In this paper, we investigate on a model for a Boeing passenger plane how the control software could look like. Based on our model described in Matlab/Simulink®, it is easy to see based on simulation that the plane loses height when the sensor for the pitch angle provides wrong data. For the opposite case of a correctly functioning sensor, our simulation does not indicate any problems. This simulation, however, is not a guarantee that the control is indeed safe. For this reason, we translated the Matlab/Simulink®-model of the controler into a hybrid program in order to make this system amenable to formal verification using the theorem prover KeYmaera.

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

1. Flight Control Model of Longitudinal Motion

For a complete description of the aircraft motion in the three dimensional space six variables are needed that denote the degrees of freedom of a rigid body. The aircraft motion is calculable by six nonlinear coupled 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 derive a third order state space model with the state vector

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

to describe the longitudinal motion [4], [5]. The state vector contains (1) the angle of attack α, pitch rate q, and pitch angle θ (cmp. Fig. 1).

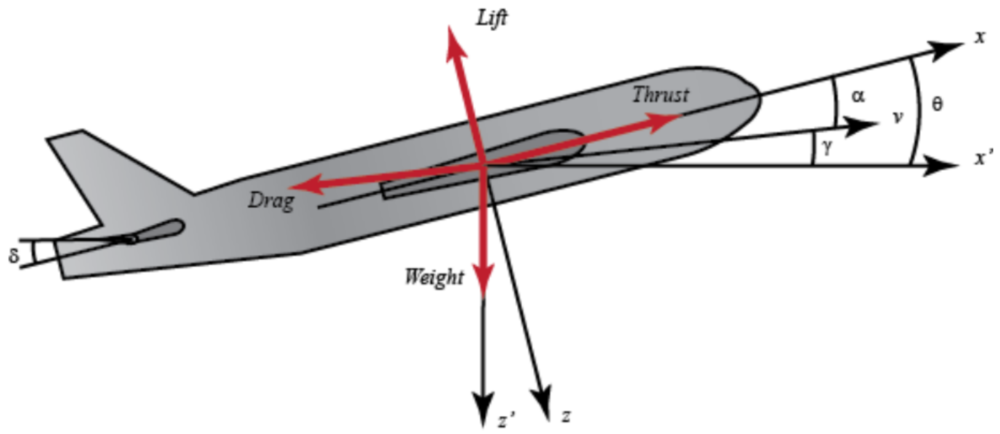


Fig. 1. Important parameters of Flight Model
(Source: *http://ctms.engin.umich.edu/CTMS/index.php?example=AircraftPitch &section=SystemModeling*)

Based on the assumption that the aircraft is in steady-cruise at constant altitude and velocity and that a change in the elevator deflection angle δ as controllable system input will not change the aircraft speed the longitudinal equations of motion for the aircraft in state space form $\dot{x} = f(x, u)$ with the state vector (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 aircraft, C_W denotes the coefficient of weight, C_M denotes coefficient of pitch moment, and I_{yy} as 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 aircraft 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 $f(x, u)$

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

can be described as follows

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

1.1. Control loop designed in Matlab/Simulink®

For the flight model introduced above, we have developed using Matlab/Simulink® a series of controllers those 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 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} to prevent airflow disruption. 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 plane.

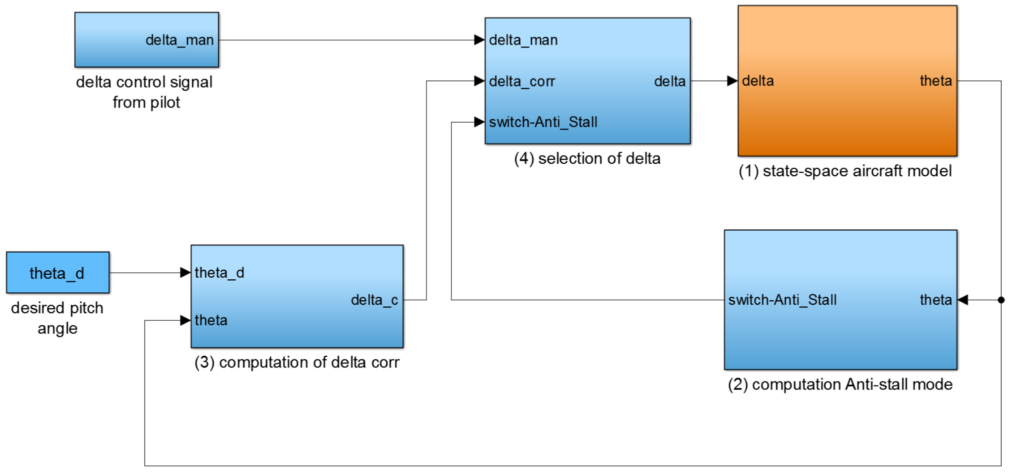


Fig. 2. Simple P-Controller designed in Matlab/Simulink®

A very simple version of the Matlab/Simulink®-controller is shown in Fig. 2 and consists of four main parts: (1) state space aircraft model, (2) computation of anti-stall mode, (3) computation of δ_{corr} (4) selection of δ from δ_{man}, δ_{corr} based on anti-stall mode.

1.2. System analysis in Matlab/Simulink®

The standard technique to analyze systems is by simulation, which is well-supported by Matlab/Simulink®.

1.2.1. Assuming correct sensor measuring for θ

The pitch angle θ is one of the outputs of the plant model and the input for the control loop. Based on θ, the input δ (pitch elevator angle) is computed for the next cycle.

Assuming that the angle θ is correcly measured by sensors, the simulation of the system does not show any situation in which θ becomes negative. This alone is not yet a guarantee, that this never happens but it is already a good starting point for formal verification of the system's safety (cmp. Sect. 2).

1.2.2. Assuming incorrect sensor measuring for θ

When building safety critical systems, engineers should always take into account that sensors might provided wrong data. We have modeled in a second Matlab/Simulink®-model a faulty sensor just by substituting the output θ of the plant model by $\theta + \theta_{offset}$. When simulating this second model, it can be immediately seen, that θ becomes soon negative, i.e. the plane can lose height.

2. Logical Analysis of Flight Control Models

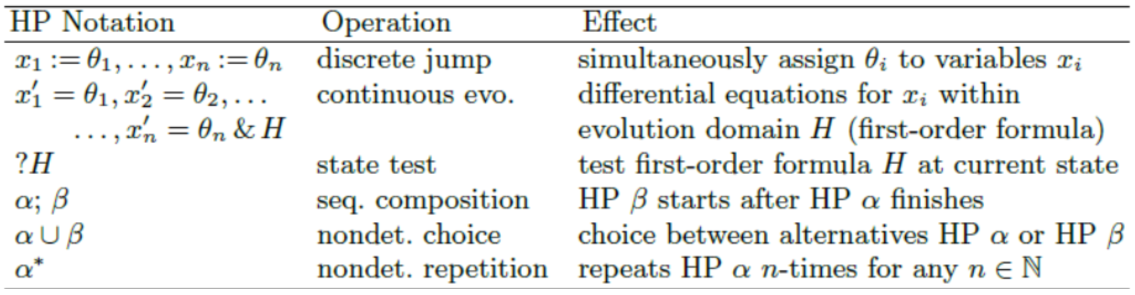
As detailed in the previous section, the Matlab/Simulink® toolkit is able to simulate the modeled system and it is easy to see, that the plane might lose height when sensors for measuring θ provide wrong data. However, for the opposite case of having a (presumably) correct system, simulation is not a sufficient technique in order to prove that the system behaves correctly under all possible circumstances. In our case, the correct behaviour means that θ remains always positive (recall that our system models the phase of a 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 [2]. The notion of HP is supported by the theorem prover KeYmaera, which enables the user to formally verify safety properties of hybrid systems [3].

2.1. KeYmaera

A proof task for KeYmaera has to be formulated in differential dynamic logic (DDL), which is an extension of classical dynamic logic [1]. In short, classical dynamic logic is a modal logic with modalities *box* ($[\alpha]\psi$) and *diamond* ($< \alpha > \psi$). In the rest of the paper, only the box-modality is applied; the formula $[\alpha]\psi$ states that in each possible poststate after program α has terminated the formula ψ holds. Please note that termination of α is not claimed!

The main difference of DDL and DL is, that the former supports continuous state statements, in which variables changes its value automatically according to differential equations. This list of supported statements is summarized in the following table[1]. For a detailed introduction to DDL, the interested reader is referred to [3].



HP Notation	Operation	Effect
$x_1 := \theta_1, \ldots, x_n := \theta_n$	discrete jump	simultaneously assign θ_i to variables x_i
$x_1' = \theta_1, x_2' = \theta_2, \ldots$	continuous evo.	differential equations for x_i within
$\ldots, x_n' = \theta_n \,\&\, H$		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
α^*	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}$

2.2. Flight model as KeYmaera-Input

The Matlab/Simulink®-model shown in Fig. 2 can be translated into a hybrid program α as follows:

[1]The table has been taken from [3].

```
            {
                $if$        $theta > upper\_theta\_bound$  $//adjust\ the\ mode$
                $then$      $stallCtrl\_mode := 1$
                $else$      $if\ theta < lower\_theta\_bound$
                            $then\ stallCtrl\_mode := 0$
                            $else\ skip$
                            $endif$
                $endif;$
                $delta\_stall := (theta_d - theta) * k_p * stallCtrl\_mode;$
$\alpha =$          $delta := delta\_stall + delta\_manual * (1 - stallCtrl\_mode);$
                $t := 0;$
                {           $alpha' = -0.313 * alpha + 56.7 * q + 0.232 * delta,$
                            $q' = -0.0139 * alpha + -0.426 * q + 0.0203 * delta,$
                            $theta' = 56.7 * q,$
                            $t' = 1$
                            $\& \ t <= ep$
                }
            }*
```

Here, the control loop of the Matlab/Simulink®-model is encoded by an iteration (*) as the outermost operator of α. Within the iteration, we have basically a sequence ($cntrl; plant$), where *plant* is the continuous state statement $\{alpha' = \ldots \& t <= ep\}$ and *cntrl* is the sequence of statements before.

The statement *plant* directly corresponds to part (1) from Fig. 2. The statements forming *cntrl* realize the parts (2), (3), (4) from Fig. 2.

2.3. Proof task for correct behaviour

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

$$\theta > 0 \rightarrow [\alpha]\theta > 0 \qquad (4)$$

In words, (4) 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 ep), the pitch angle remains positive. Note, that the proof of this claim will rely on some other assumptions, e.g. $ep > 0$, which have been suppressed here for the sake of brevity.

3. Conclusion and Future Work

In this paper, we have investigate safety critical software for controlling the flight of modern aircrafts. 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. Its sole purpose is to provide an example at which quality assurance techniques can be applied.

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 plane software is hastily developed: Is it possible that the controller software could cause the plane to lose height, which - eventually - might cause the plane to crash.

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

For the case that the sensor works correctly, we propose to translate the Matlab/Simulink®-model into a hybrid program and to apply the theorem prover KeYmaera in order to ensure for all possible situations that the system works correctly (not only for the few situations captured by the simulation).

Unfortunately, establishing such a formal proof using KeYmaera is a non-trivial task, because techniques for traditional software verification (e.g. case distinction for if-then-else) have to be combined with mathematical analysis methods for ordinary differential equations (ODEs). We will continue to work to formulate the key arguments of the proof so, that they can be processed by KeYmaera easily.

References

1. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* Foundation of Computing. MIT Press, 2000.
2. Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE*

Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996, pages 278–292. IEEE Computer Society, 1996.

3. Jan-David Quesel, Stefan Mitsch, Sarah Loos, Nikos Aréchiga, and André Platzer. How to model and prove hybrid systems with KeYmaera: A tutorial on safety. *STTT*, 18(1):67–91, 2016.
4. Robert F. Stengel. *Flight Dynamics.* Princeton University Press, 2004.
5. Thomas R. Yechout. *Introduction to Aircraft Flight Mechanics.* American Institute of Aeronautics & Astronautics, 2003.