aerospace

MDPI

Article

Plant Model-Based Fault Detection during Aircraft Takeoff Using Non-Deterministic Finite-State Automata

Ferdinand Settele *,†, Alexander Weber † and Alexander Knoll

Department of Mechanical, Automotive and Aeronautical Engineering, University of Applied Sciences Munich, 80335 Munich, Germany; weber13@hm.edu (A.W.); alexander.knoll@hm.edu (A.K.)

* Correspondence: ferdinand.settele@hm.edu

† These authors contributed equally to this work.

Received: 8 March 2020; Accepted: 29 July 2020; Published: 31 July 2020

Abstract: In this note, the application of a plant model-based fault detection method for nonlinear control systems on aircraft takeoff is introduced. This method utilizes non-deterministic finite-state automata, which approximate the fault-free dynamics of the plant. The aforementioned automaton is computed in a preliminary step while during evolution of the plant the automaton is continually evaluated to detect discrepancies between the actual and the nominal dynamics. In this way the fault detection module itself can be implemented on simpler hardware on board of the plant. Moreover, an implementation technique is presented that allows the use of the proposed fault detection method when the plant dynamics is given only by means of a graphical programming script. The great potential and practicality of the used method are demonstrated on a simulated takeoff manoeuvre of a battery-electrically driven aircraft.

Keywords: model-based fault detection; finite-state automata; battery-electrically driven aircraft

1. Introduction

Fault detection is the first and most important component of the well-known scheme of Fault Detection, Isolation and Accomodation [1,2]. Approaches of fault detection can be model-free (based on simple "if-then" blocks) or model-based (e.g., based on models of signals, the plant dynamics itself or knowledge-based). There is a wide variety of techniques for fault diagnosis. Rather simple limit checking methods are applied in process automation systems [3]. For affine nonlinear systems the work [4] shows a differential geometry approach using residual generators. Non-standard fault detection methods include, for example, the use of artificial neural networks [5] or discrete event systems [6–15].

Roughly speaking, the present work takes up ideas relying on discrete event systems, yet differs in various aspects. To be specific, this work reformulates some known concepts of model-based fault detection. A substantial difference of this note to existing results is the utilization of a non-deterministic finite-state automaton (FSA) and the particular application on an aircraft takeoff fault detection scenario. The FSA is used to detect discrepancies ("residuals") between the model-predicted and the actual behaviour of the dynamical system under consideration. It approximates the fault-free dynamics in a discrete, non-deterministic form following the methodology of References [16,17]. The non-determinism allows to account for unknown parameters or (regular) disturbances to the plant dynamics.

Classically, this automaton can be obtained fully automated and computationally efficient under certain continuity assumptions on the plant dynamics. For the purpose of fault detection, we propose

a technique for generating an "empirical" approximation in case when only a simulation model, for example, in form of a graphical programming script, is available. Clearly, this is not an exceptional case in industrial engineering environments.

Though our method can be formally used in a wide range of situations (from plants with known dynamics to the aforementioned case of simulation models) it is not the purpose of this paper to compare our method in each application case with well-established fault detection techniques. The goal of this work is to present a fully automated and easy-to-implement technique and we are going to highlight the case of simulation models.

The differences to existing works are various. Firstly, "empirical" approximations have been proposed previously, yet we extend this idea to generate approximations given by non-deterministic finite-state automata rather than directed graphs (deterministic automata) [18]. As for fault-detection based on discrete event systems [6–15], References [6–9] build up automata by hand on the basis of discrete events. The authors of References [10–12] model the used discrete automaton based on the continuous dynamics of the plant under consideration, as we do. In Reference [15] the original system is mapped to a simplified system and conditions are given under which it is sufficient to perform fault-detection on the simplified model only. The authors of References [13,14] add focus on stochastic automata. In contrast, our approach relies on a fully automated sampling-based methodology involving a non-determinism already mentioned before. The latter has been proven to be applicable in real applications in the context of symbolic controller synthesis, for example, Reference [19].

To demonstrate the practicality this work investigates the application of the novel fault detection method to aeronautics. It is applied to the acceleration phase during aircraft takeoff using a detailed propulsion model of a battery-electrically driven aircraft from Reference [20]. Our motivation origins from the fact that a huge amount of accidents in aviation is related to the first phases of flight. For example, between 1999 and 2002 about 25 % of all accidents of airliners happened during takeoff and climb phase [21,22]. The accident of a Boeing 737–800 in Goa is a recent example, where an insufficient stabilization phase for the two engines caused a runway excursion to the side [23]. To minimize risks during acceleration phase, system states have to be monitored carefully by the pilot. Automated fault detection, supporting pilot assistance systems can reduce workload for the pilot, risk potential can be identified earlier and therefore accidents can be avoided.

Several work concentrates on fault detection during the takeoff. References [24,25] show a takeoff monitoring concept, which contains two segments: In the first (pre-takeoff) segment, scheduled performance data is generated, which is compared with measured data in the second (real-time) segment. In contrast to the approach we will present, data measurement is done manually and specific to the problem at hand. In References [26,27] a so-called Flight Safety Assessment and Management is investigated which uses the theory of discrete event systems.

The rest of the paper is organized as follows. First, Section 2 explains the presented fault detection method. It includes details to the concept of discrete approximations of plant dynamics. Section 3 discusses the application of the main contributions to aeronautics in simulation. Section 4 contains a short conclusion and outlook.

2. Modification of Common Methodology

Before presenting the details of the novel method, the basic notation and terminology used subsequently is introduced.

The set of real numbers, non-negative real numbers, integers and non-negative integers is denoted by $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{Z}$ and $\mathbb{Z}_+$, respectively. For a set A we denote by $\mathcal{P}(A)$ the power set of A. $A \setminus B$ stands for the set difference. A *cover* of a set A is a subset of $\mathcal{P}(A)$ such that the union of its elements is equal to A. All elements of the *cover* represent A in turn. For non-empty sets A and B we denote by B^A the set of all maps $A \to B$. If $A \subseteq \mathbb{R}$ then an element of B^A is called *signal* with *time range* A. A set-valued map $f\colon A \to B$ is called *strict* if $\emptyset \neq f(a)$ for all $a \in A$.

2.1. Systems and Faults

In this paper we consider continuous-time continuous-state control systems governed by a differential inclusion of the form

$$\dot{x}(t) \in F(x(t), u(t)), \tag{1}$$

where $x\colon \mathbb{R} \to X$ and $u\colon \mathbb{R} \to U$ is the state and input signal, respectively, and where $X \subseteq \mathbb{R}^n$, $U \subseteq \mathbb{R}^m$. The map $F\colon X \times U \to \mathcal{P}(X)$ in (1) is set-valued and strict. Disturbances or unknown parameters can be formalized by means of the non-determinism induced by F to the dynamics. For example, a parameter-depend control system governed by

$$\dot{x}(t) = f(x(t), u(t), p(t)), \tag{2}$$

where $p\colon \mathbb{R} \to P$, $P \subseteq \mathbb{R}^l$ is a non-empty set of parameters and $f\colon X \times U \times P \to X$ is an ordinary map, follows dynamics (1) by setting $F = f(\cdot, \cdot, P)$.

The method that we are going to use formally applies to sampled versions of the previous dynamics. In fact, dynamics (1) transform through sampling to a discrete-time continuous-state difference inclusion

$$x(k+1) \in G(x(k), u(k)) \tag{3}$$

with the state and input signals x and u, respectively. Here, the time range of x and u is $\mathbb{Z}_+$ rather than $\mathbb{R}$ with discrete-time variable k. We assume the map $G\colon X \times U \to \mathcal{P}(X)$ to be strict throughout, thus a successor state always exists. $G(x_0, v)$ is defined as the image of the point x_0 under the flow of the continuous-time dynamics (1) for input signal v at a predefined *sampling period* $\tau > 0$ [17] (Sec. VIII).

The notion of a *fault (in the dynamics)* is conveniently modeled by a change from G in (3) to some other function so that $x(k+1) \notin G(x(k), u(k))$ for at least one $k \in \mathbb{Z}_+$.

For the description of the presented method we first formalize the notion of system and its behaviour [16,17].

Definition 1. *A **system** is a triple (X, U, G) where X and U are non-empty sets and $G\colon X \times U \to \mathcal{P}(X)$ is strict. The components of the triple are called **state space**, **input space** and **transition function**, respectively. A system whose state space equals $\mathbb{R}^n$ for some $n \geq 1$ is called **plant**.*

Definition 2. *For a system $S = (X, U, G)$ the **behaviour** of S is the set*

$$\mathcal{B}(S) := \{(u, x) \in (U \times X)^{\mathbb{Z}_+} \mid \forall_{k \in \mathbb{Z}_+} : (3) \text{ holds}\}.$$

In words, the behaviour of a system consists of those pairs of input and output signal that are generated by the system according to (3). See Figure 1.

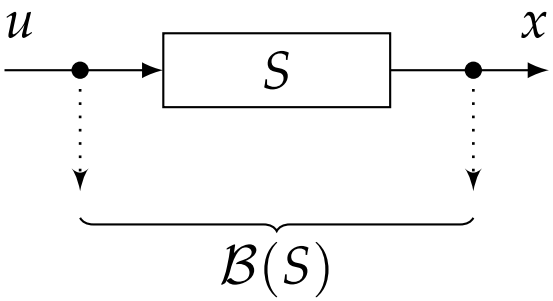

Figure 1. Illustration of the behaviour of a system S.

2.2. Fault Detection Method

Below, we introduce *discrete approximations* [28], also known as *discrete abstractions*, with which the presented method detects faults. We will use the notion from Reference [17] in a simplified and modified form that is suitable for our purposes.

The key property of a discrete approximation is that it quantizes the continuous state space of a plant into a discrete set and that it "approximates" the behaviour of the plant in an appropriate sense. We continue these descriptive explanations after the precise definition.

Definition 3. *Let $S = (X, U, G)$ be a plant, let X' and U' be a cover of X and U, respectively, by non-empty sets. A system $S' = (X', U', G')$ is called* ***discrete approximation*** *of S if*

$$\Omega_2 \cap G(\Omega_1, \Upsilon) \neq \emptyset \;\Rightarrow\; \Omega_2 \in G'(\Omega_1, \Upsilon) \tag{4}$$

holds whenever $\Omega_1, \Omega_2 \in X'$, $\Upsilon \in U'$. Elements of X' and U' are called ***cells*** *and* ***input symbols****, respectively.*

Property (4) is crucial for the "approximation" of the plant behaviour: If for two cells $\Omega_1, \Omega_2 \in X'$ and an input symbol $\Upsilon \in U'$ it holds $\Omega_2 \notin G'(\Omega_1, \Upsilon)$ then the contrapositive of (4) implies that the plant does not produce signals $(u, x) \in \mathcal{B}(S)$ such that

$$(u(k), x(k), x(k+1)) \in \Upsilon \times \Omega_1 \times \Omega_2 \tag{5}$$

for some $k \in \mathbb{Z}_+$. In turn, the observation of a signal possessing the property (5) gives evidence that a fault in the dynamics is present.

The latter remark, which is a rigorous mathematical statement, results in a straightforward algorithm for fault detection as follows. Firstly, the components of the algorithm are, besides the discrete approximation (X', U', G') of the plant (X, U, G), the two quantizers ("analog-to-digital converters")

$$Q_{\text{state}} \colon X \to \mathcal{P}(X') \;\text{ and }\; Q_{\text{input}} \colon U \to \mathcal{P}(U'), \tag{6}$$

which are induced by the covers of X and U, respectively. (I.e., $\Omega \in Q_{\text{state}}(a)$ if and only if $a \in \Omega$, and analogously for Q_{input}). Then, for every point in time $k \in \mathbb{Z}_+$ the values

$$u(k),\; x(k) \text{ and } x(k+1)$$

are measured and converted to

$$\Upsilon(k),\; \Omega(k) \text{ and } \Omega(k+1),$$

respectively, by means of the quantizers. More concretely, $\Upsilon(k) \in Q_{\text{input}}(u(k))$ and $\Omega(s) \in Q_{\text{state}}(x(s))$ for $s \in \{k, k+1\}$ is fulfilled. (If the image of a quantizer contains more than one element then an element may be picked randomly for the conversion.) Finally, the test

$$\Omega(k+1) \in G'(\Omega(k), \Upsilon(k)) \tag{7}$$

is performed. A fault is signaled if the latter set membership relation is violated. The algorithm is illustrated in Figure 2.

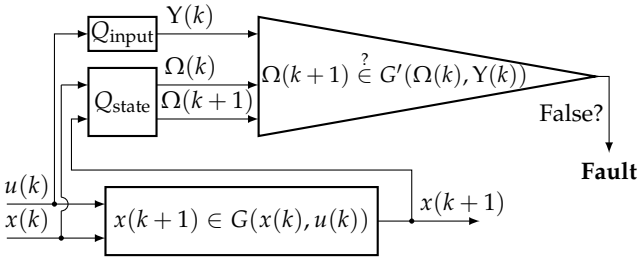

Figure 2. Illustration of the fault detection algorithm applied to a plant (X, U, G) with discrete approximation (X', U', G') and quantizers (6).

2.3. Computing Discrete Approximations

Consequently, the presented fault detection method requires a discrete approximation of the plant to monitor.

2.3.1. Computational Method

Methods to compute discrete approximations (for the various existing notions of discrete approximations) have been developed since several years, for example, References [17,19,29]. Typically, the computation of a discrete approximation (X', U', G') for a given plant (X, U, G) is done in three steps [29], which are outlined subsequently.

Firstly, assuming for simplicity that U is finite, $U' = U$ is set and a *finite* cover X' of X is chosen. The cover typically contains two distinguished subsets $\bar{X}'$ and $X' \setminus \bar{X}'$, which cover the "operating range" of the plant and its complement ("overflows"), respectively. For example, in Figure 3 the cover is formed by the shown squares and outer enclosing rectangles. Secondly, for every cell $\Omega \in \bar{X}'$ the set $G(\Omega, Y)$ is overapproximated. Specifically, a set $\widehat{G}(\Omega, Y)$ that satisfies $G(\Omega, Y) \subseteq \widehat{G}(\Omega, Y)$ is calculated. Here, a proper superset is typically needed; the reachable set $G(\Omega, Y)$ is usually not explicitly representable, whereas an explicitly representable superset might be found. The overapproximation property is also enough in order to establish (4) in the following last step: The automaton is obtained, that is, G' is defined, by establishing (4) through intersection tests

$$\Omega_2 \cap \widehat{G}(\Omega_1, Y) \tag{8}$$

for every $\Omega_1 \in \bar{X}'$, $\Omega_2 \in X'$ and $Y \in U'$. To be specific, if (8) is non-empty then $\Omega_2 \in G'(\Omega_1, Y)$. The trivial choice $G'(\Omega, Y) = X'$ is made for $\Omega \in X' \setminus \bar{X}'$ ("overflow" cell).

2.3.2. Implementation

In order to allow for efficient computation, the cover in the first step of the scheme is formed by translated copies of a closed hyper-rectangle and a few unbounded hyper-rectangles (aforementioned overflow cells) to fulfill the cover property. When choosing the overapproximating superset $\widehat{G}(\Omega_1, Y)$ in (8) also as a hyper-rectangle then (8) turns into an intersection test between hyper-rectangles, which can be efficiently evaluated. So the difficult part when implementing the scheme described in Section 2.3.1 for control systems with continuous-time dynamics (1) is to establish a computationally efficient overapproximation method by hyper-rectangles.

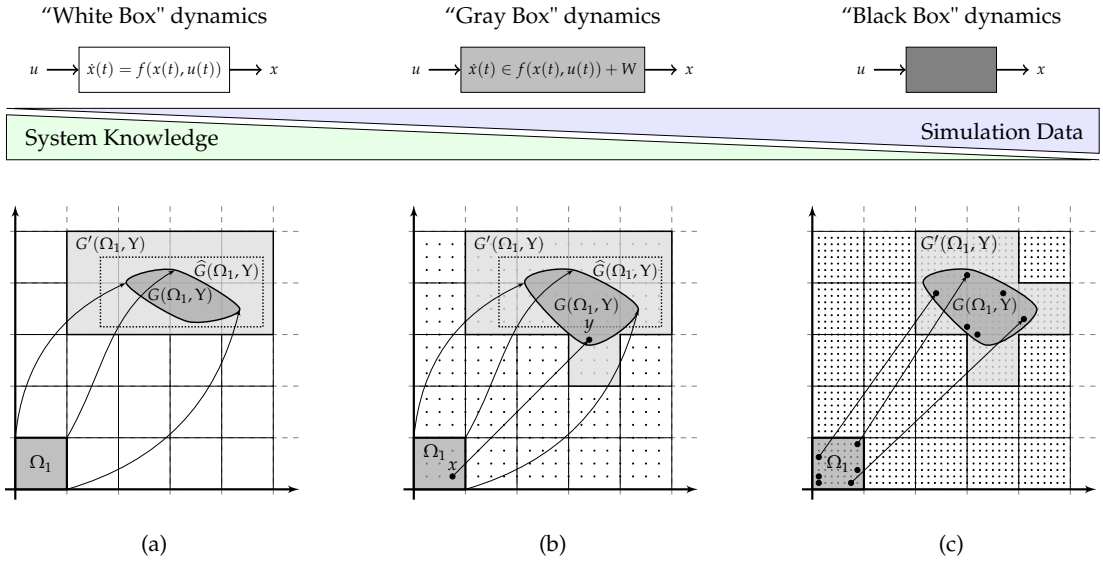

Figure 3. Illustration of the computation of a discrete approximation for a 2-dimensional plant in three situations: (**a**) dynamics are fully known; (**b**) dynamics are known except for the set of disturbances W; (**c**) dynamics are given by means of a simulation model. In the illustrations, for the cell Ω_1 and input symbol $Y \in U'$ the image of the transition function $G'(\Omega_1, Y)$ is determined in (**a**) by a rectangle $\widehat{G}(\Omega_1, Y)$ overapproximating the reachable set $G(\Omega_1, Y)$, in (**b**) as in (**a**) but with additional test points (e.g., x is mapped to y) to take into account the unknown disturbances, in (**c**) by test points only.

2.4. Theory Versus Reality

Such an overapproximating method exists [30] and can be efficiently applied [17,19]. However, one prerequisite is that F in (1) fulfills some smoothness conditions. More generally, the assumption that F in (1) is given as a mathematical formula constitutes a rather ideal situation ("white box" model, see Figure 3) which in applications is rarely present. In contrast, common engineering practice for modelling complex control systems (like aircraft) is the assembling of the dynamics from various subsystems to one huge "simulation model". Quite common is the use of graphical programming tools for it [31]. In this way, however, the function in (1) is hidden behind the assembled interconnection of the subsystems. It is typically unpractical to extract the mathematical formula for further processing, for example, for computing discrete approximations. Therefore, it suggests itself that, with abandoning formal guarantees from theory, empirical techniques may allow to generate an "empirical" discrete approximation from the simulation model. The details to this idea are given below.

Besides the aforementioned case of a "white box" model we further distinguish two more situations. First, the plant dynamics are given by

$$\dot{x}(t) \in f(x(t), u(t)) + W \tag{9}$$

where f is known and possesses the previous continuity properties and where $W \subseteq \mathbb{R}^n$ is unknown ("gray box" model), but is encoded in the simulation model and $0 \in W$. (Works like References [17,32,33] model uncertainties or disturbances by W in (9)). In this case, a discrete approximation (X', U', G') for the nominal dynamics, that is, for the plant (X, U, G) obtained from sampling (9) for $W = \{0\}$, is computed in a first step in the convential way. In a second step, transitions are supplemented to the obtained automaton by evaluating the simulation model for a chosen set of test points in X and U. More concretely, for every $(\Omega, Y) \in X' \times U'$ two sets of test points $x_i \in \Omega$, $u_j \in Y$ are chosen, where $(i, j) \in I \times J$ and I and J are finite index sets. Next, the images $y_{i,j}$ of x_i under u_j obtained from evaluating the simulation model are computed. Then, using the quantizers (6) the corresponding triple $T = (Y, \Omega_1, \Omega_2)$ satisfying $(u_j, x_i, y_{i,j}) \in T$ is determined. If $\Omega_2 \notin G'(\Omega_1, Y)$ then G' is redefined

so that $\Omega_2 \in G'(\Omega_1, \mathrm{Y})$. In this way, transitions due to the initially neglected set W are added to the automaton. Hence, it approximates more accurately the actual dynamics (9) of the plant. See Figure 3b.

In the last situation ("black box" model), the simulation model is exclusively used to build up the automaton. As before, for every $(\Omega, \mathrm{Y}) \in X' \times U'$ two sets of test points $x_i \in \Omega$, $u_j \in \mathrm{Y}$ are chosen, where $(i, j) \in I \times J$ (I and J as before) and the images $y_{i,j}$ of x_i under u_j obtained from evaluating the simulation model are computed. Finally, the automaton is defined by virtue of

$$G'(\Omega, \mathrm{Y}) = \{Q_{\mathrm{state}}(y_{i,j}) \mid (i, j) \in I \times J\}. \tag{10}$$

See Figure 3c. In the case that the model additionally depends on parameters an additional set of test points $d_k \in P$, $k \in K$ is chosen, where K is a finite index set. Then, the images $y_{i,j,k}$ of x_i under u_j for parameter d_k are computed and the elements $Q_{\mathrm{state}}(y_{i,j,k})$ for all $(i, j, k) \in I \times J \times K$ are used in the right hand side of (10) accordingly.

Finally, a remark on input inaccuracies is given. Formally, the control input $u(t)$ in (1) is assumed to be constant during the sampling period. In reality, those control values may vary. These disturbances can be compensated by simply letting neighboring cells of U' be strictly overlapping by an amount accounting for the expected input inaccuracies. The previously described computational techniques remain unchanged in this case.

3. Application: Takeoff Monitoring

To investigate the presented fault detection method in application, we consider a model of a small battery-electrically driven aircraft accelerating on runway until V_1-speed. At this pre-calculated speed pilots must decide to continue takeoff and lift off or to start braking to abort takeoff. In case of continuation the pilot commands rotation at speed $V_R \geq V_1$, that is, initiates the lift-off (cf. Figure 4).

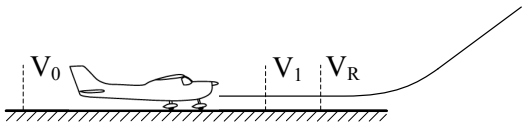

Figure 4. Takeoff Scheme.

We will consider the "black box" situation described in Section 2.4. Nevertheless, for convenience the dynamics of the aircraft are outlined in the first part of this section. Three illustrative fault scenarios will be considered. In addition, this section includes remarks on implementation and finally the experimental evaluation of the presented fault detection algorithm on the chosen example is presented.

3.1. Regular Relations during Takeoff

The dynamic model of the battery-electrically driven airplane aims at a detailed energy efficiency modelling of the involved components (Propeller, electric motor including motor controller and battery). Figure 5 shows the components of the propulsion model.

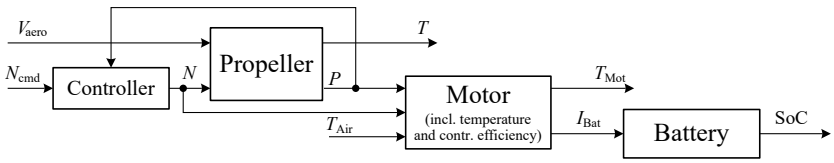

Figure 5. Overview of propulsion components modelling.

Inputs of the system are the aerodynamic velocity V_{aero}, the commanded rotational speed of the propeller N_{cmd} and the air temperature T_{Air} , which is assumed to be constant. Outputs of the system are the thrust T, produced by the propeller, the motor temperature T_{Mot} and the battery state of charge SoC.

The propeller, which produces a thrust $T \in \mathbb{R}$, is modelled by means of lookup tables of dimensionless thrust and power coefficients $C_T(V_{\text{aero}}, N)$ and $C_P(V_{\text{aero}}, N)$. With these, provided thrust and required power are calculated depending on the aerodynamic velocity V_{aero} and rotational speed N [20]. The motor (including motor controlling) translates the mechanical power P, required by the propeller, into electrical power, provided by the battery, containing motor and controller efficiency $\eta_{\text{Mot+Cont}}(N, P)$ [20]:

$$P = U_{\text{Bat}}\ I_{\text{Bat}}\ \eta_{\text{Mot+Inv}}.$$

The battery voltage U_{Bat} is assumed to be constant during acceleration phase. Increasing of the motor temperature T_{Mot} is modelled by

$$\dot{T}_{\text{Mot}} = \frac{dT_{\text{Mot}}}{dt} = \left(P\left(1 - \eta_{\text{Mot+Cont}}\right) - \dot{Q}_{\text{cool}}\right)\frac{1}{k_{T_{\text{Mot}}}} \tag{11}$$

with the thermal conductivity of the motor windings. The cooling flux $\dot{Q} = \alpha_{\text{cool}} \cdot S_{\text{cool}}\left(T_{\text{Mot}} - T_{\text{Air}}\right)$ is determined with heat transfer coefficient α_{cool}, the cooling surface S_{cool}, the thermal conductivity $k_{T_{\text{Mot}}}$ and the air temperature T_{Air}, where all terms except the motor temperature T_{Mot} are assumed to be constant during the acceleration phase of the takeoff. Furthermore, the motor dynamics contain an internal saturation, which may reduce the resulting propeller RPM with respect to limitations $I_{\text{Mot}}(t) \leq \overline{I}$, $P(t) \leq \overline{P}$ of the motor. Here, I_{Mot} and P denotes the motor current and the motor power, respectively, and some bounding constants $\overline{I} \in \mathbb{R}$ and $\overline{P} \in \mathbb{R}$. The resulting propeller RPM $N(t)$ equals

$$\begin{cases} N_{\text{cmd}} & \text{if } I_{\text{Mot}}(N_{\text{cmd}}, t) < \overline{I} \wedge P(N_{\text{cmd}}, t) < \overline{P} \\ N(I_{\text{Mot,max}}) & \text{if } I_{\text{Mot}}(N_{\text{cmd}}, t) \geq \overline{I} \wedge P(N_{\text{cmd}}, t) < \overline{P} \\ N(P_{\text{max}}) & \text{otherwise .} \end{cases}$$

Finally, the decrease of the state of charge of the battery SoC is calculated with the nominal capacity C_{nom} of the battery:

$$\dot{SoC} = \frac{d\text{SoC}}{dt} = \frac{I_{\text{Bat}}}{C_{\text{nom}}}. \tag{12}$$

For more detailed descriptions of the components the reader is referred to Reference [20].

The aircraft dynamics are given by a parametrized differential equation of the form (2), where $f\colon \mathbb{R}^3 \times \mathbb{R}^2 \times P \to \mathbb{R}^3$. The parameter vector $p = [\mu\ V_{\text{Wind}}]^\top$ in (2) includes the surface friction coefficient μ and the headwind velocity V_{Wind}. Regarding the later fault scenarios, the surface friction coefficient is realistically bounded by $\mu := [0.005, 0.04]$ [24,25] and the headwind amounts to $\text{V}_{\text{Wind}} := [0, 2]$ [m/s]. More realistic values of the friction coefficient can be determined by friction tests. It is assumed that real friction parameter values do not spread that wide like given in our paper. Furthermore, wind is assumed to hit the aircraft straightly from the front (Aircraft takeoff is typically performed with headwind best possible with regard to the runway direction).

The state vector is given by

$$x(t) = \begin{bmatrix} V(t) & T_{\text{Mot}}(t) & \text{SoC}(t) \end{bmatrix}^\top \tag{13}$$

where the components represent the aircraft kinematic velocity $V = V_{\text{aero}} + V_{\text{Wind}}$, the motor temperature T_{Mot} and the state of charge of the battery (SoC), respectively. The control vector is given by

$$u(t) = \begin{bmatrix} C_L(t) & N_{\text{cmd}}(t) \end{bmatrix}^\top \tag{14}$$

where C_L and N_{cmd}, respectively, is the commanded lift coefficient and commanded propeller RPM (rotational speed of the propeller).

Aerodynamic lift and drag forces L and D are computed using standard formulas, for example,

$$D = C_D \, \frac{\rho}{2} V_{\text{aero}}^2 \, S_{\text{ref}} \tag{15}$$

with drag coefficient C_D depending on C_L by a polar, constant air density ρ and reference surface of the wing S_{ref}. The kinematic acceleration $\dot{V}$ is given by

$$\dot{V} = \frac{T - D - \mu \, (mg - L)}{m} \tag{16}$$

with gravitational acceleration g, the mass m of the aircraft and the friction coefficient μ. Therefore, it is assumed that (1) thrust T and drag D are co-linear with $\dot{V}$, (2) lift L and weight force mg are co-linear among each other and perpendicular to thrust and drag, and (3) the runway is exactly horizontal. The dynamics for T_{Mot} and SoC taken from Reference [20].

3.2. Fault Modelling

Three different types of faults are modelled in the simulation of the takeoff run. For fault 1, which is an exceptionally increasing motor temperature due to an assumed error in the cooling system, factor $e_2 \in \mathbb{R}$ is added to the state derivative $\dot{T}_{\text{Mot}}$ from time t_2. This fault will cause an unusual faster increase of the motor temperature T_{Mot} with (11). Fault 2 is an increasing friction due to a problem with the tyres, so a factor $e_1 \in \mathbb{R}$ is multiplied to the value of the drag D in (15) beginning from time t_1 of the takeoff run simulation time t. As a consequence, the acceleration (16) will decrease. Fault 3 illustrates a short circuit in the powertrain, due to which the battery current I_{Bat} increases. This is realized by a factor $e_3 \in \mathbb{R}$, which is multiplied to I_{Bat} from time t_3. This fault will lead to an unusual steeper decrease of the battery state of charge SoC (12). See Table 1 for the values of the aforementioned constants.

Table 1. Numerical values and occurrence times of the faults.

Fault i	Increasing of ...	e_i	t_i [s]
1	motor temperature $\dot{T}_{\text{Mot}}$	2.5	6
2	drag D	2.5	7
3	battery current I_{Bat}	2	2

3.3. Implementation of the Takeoff Monitoring

The takeoff monitoring system is divided into two parts according to the presented fault detection method. In the first part, a discrete approximation for the model defined in Section 3.1 is computed. The second part contains the "real-time" algorithm for fault detection during takeoff.

3.3.1. Computing the Discrete Approximation

The discrete approximation $S' = (X', U', G')$ for the given plant (aircraft) is determined according to Section 2, where the "black box" model situation is assumed. A detailed description of the construction of S' follows. To begin with, all states, controls and parameter ranges are given in Table 2. The cover X' of $\mathbb{R}^3$ is formed by translated copies of $[0, 0.5] \times [0, 0.5] \times [0, 0.004]$ (covering the operating range $[0, 25] \times [35, 45] \times [0.94, 0.96]$) and some unbounded hyper-rectangles to cover $\mathbb{R}^3$. The cover U' of U is given by translated copies of $[0, 0.067] \times [0, 20]$. The shape of the cells and input symbols can be deduced from Table 2. The cover P' of the parameter range is given by the single hyper-rectangle $[0.005, 0.04] \times [0, 2]$.

Table 2. Ranges of the states, controls and the parameter assumed for the construction of the discrete approximation.

		Unit	Interval	Resolution
States				
Velocity	V	m/s	$[0, 25]$	0.5
Motor temp.	T_{Mot}	°C	$[35, 45]$	0.5
State of charge	SoC	–	$[0.94, 0.96]$	0.004
Controls				
Lift coefficient	C_L	–	$[0.9, 1.1]$	0.067
Propeller RPM	N_{cmd}	RPM	$[2400, 2600]$	20
Parameter				
Friction coeff.	μ	–	$[0.005, 0.04]$	0.035
Headwind	V_{Wind}	m/s	$[0, 2]$	2

To obtain the transition function G' of the discrete approximation S' the sampling time for evaluating the black box model is chosen as $\tau = 1\,\text{s}$ and about 1.67×10^8 test points are used. These points form a uniform grid on the Cartesian product of the six intervals defined in Table 2.

The calculation is performed using MATLAB [34]. The latter environment provides easy-to-use grid organization using the function `ndgrid` on the one hand. On the other hand, it provides user-friendly data accessing functions, for example, linear indexing by means of the function `sub2ind`. Furthermore, using the function `parfor` from the *Parallel Computing Toolbox* [34] the computation of the discrete approximation can be parallelized easily.

The computation of the discrete approximation takes 43 h using 6 threads at 3.7 GHz-CPUs. The required memory for the discrete approximation amounts to 330 MB, where indices are saved as 4-byte integers. Finally, approximately 500 MB are needed for the on board implementation of the introduced algorithm.

Of course, the price to pay for using user-friendly programming environments is an increased runtime in comparison with compiled binaries. Nevertheless, we would like to point out that the computation of the discrete approximation is done once per plant, so a higher runtime does not limit applicability.

3.3.2. Simulation of the Takeoff Run

For a simulation of the takeoff run the initial state

$$x(0) = \begin{bmatrix} 0 & 35 & 0.95 \end{bmatrix}^\top$$

is assumed with the constant control $u(t) = u_0$ with

$$u_0 = \begin{bmatrix} 0.95 & 2500 \end{bmatrix}^\top$$

is applied. The friction parameter μ is assumed to be randomly distributed within its bounds given in Table 2 over the runway to allow an investigation of the robustness of the monitoring module with regard to deviation of parameters. For the simulation of fault 2 (drag increase, see Table 1), additionally wind was added to test the systems reliability in presence of an unknown parameter. Here, headwind is modelled in form of a sine function between values 0.1 and 1.9 m/s and a period time of about 15 s.

3.3.3. Implementation of the Fault Detection Algorithm

The algorithm in Figure 2 is implemented as follows. From time 0 s the state $x(t)$ and input $u(t)$ are measured and converted to $Q_{\text{state}}(x(t))$ and $Q_{\text{input}}(u(t))$, respectively, in a time raster of 0.05 s width and saved in a memory. The time raster is defined according to

$$t = k_1\tau + k_2 \cdot 0.05$$

with $k_1 \in \mathbb{N}$ and $k_2 \in \{0, \ldots, 19\}$. From time $t = \tau = 1\,\text{s}$, fault detection begins with evaluating (7) according to Figure 2. (Before the chosen sampling time τ fault detection is not feasible by concept since the state at time τ is located in the future.) The actual states $x(t)$ are converted similarly by (6) and the corresponding images at time $t - \tau$ are taken from memory and used in (7) (cf. Figure 2). Cells and input symbols older than $t - \tau$ are discarded. The program sequence for automated takeoff monitoring is depicted in Figure 6.

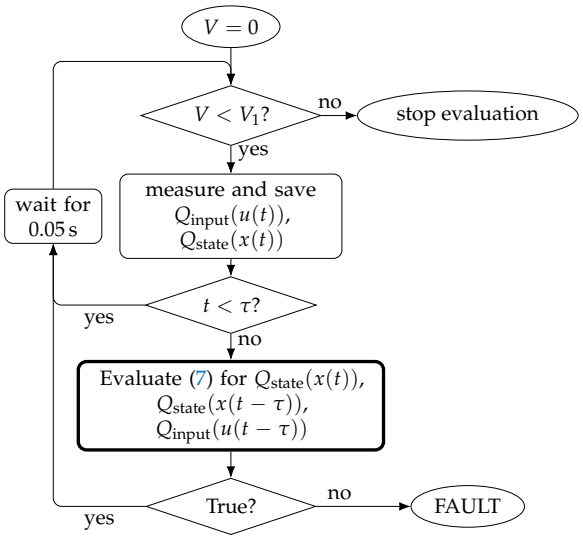

Figure 6. Program Scheme.

The core function (highlighted block) brings about a decision to continue or abort takeoff.

With the hardware described in Section 3.3 one automaton evaluation takes 0.034 ms. This comparison in the real-time module can be implemented as a simple index search. The MATLAB function is member can be used to determine whether the index $Q_{\text{state}}(x(t))$ is saved in the image of the transition function for the pair $(Q_{\text{state}}(x(t-\tau)), Q_{\text{input}}(u(t-\tau)))$ or not.

3.4. Experimental Evaluation

Below, the aforementioned three fault scenarios are simulated and discussed. The Figures 7–9 include the progression of the affected state (thick black line) together with the error flag of the fault detection program. The state progression without fault (gray dotted) is added to the plot as well as the bounds allowed by the discrete approximation (thin black stair lines).

State progression and the fault detection of **Fault 1** is shown in Figure 7. The constant e_2 leads to a steeper increasing of the motor temperature beginning at 6 s compared to the fault free progression (gray dotted line).

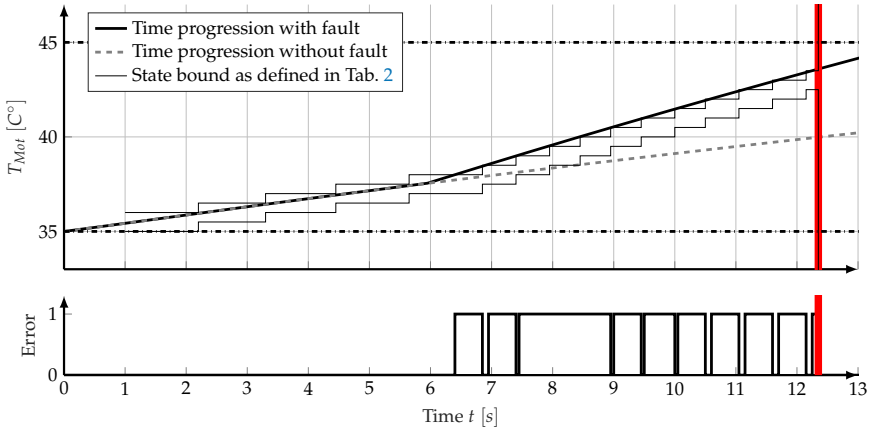

Figure 7. Fault 1.

Here, the fault can be detected very early and clear at about half a second after the fault appears. The fault detection is 'disabled' at about 12 s (red line) since the ground speed V exceeds the upper limit of the operating range (cf. Table 2). In other words, the conversion of the continuous state signal returns overflow cells. Furthermore, it should be noted that the state progress without fault (gray dotted line) does not run within the thresholds included by the FSA. The state progression and the fault detection logic are simulated with the active fault for Figure 7.

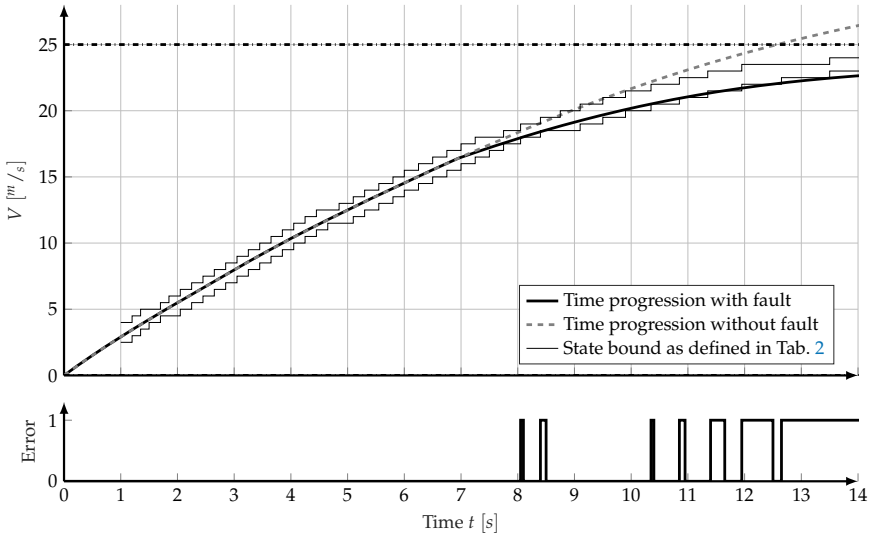

Figure 8. Fault 2.

Fault 2 scenario can be seen in Figure 8. The increased friction slows down the acceleration during takeoff. Consequently, the lines for state progression with and without fault diverge beginning at $t_1 = 6$ s. At 8.1 s a violation of property (7) occurs for the first time and the fault detection module signals a fault. The resolutions chosen to compute the discrete approximation and the wide ranges of both parameter μ and V_{Wind} do not allow an ealier detection of the fault. For the same reason an alternating 'true-false' fault signal with a gap around 9 and 10 s appears first. From 12.5 s a stable error is detected. A reason for this late fault detection is the rougher resolution of state V used in the discrete approximation in comparison to the state T_{Mot} (cf. Table 2).

As described before, a sinusoidal headwind is added to stress the presented method's reliability. Therefore, the phase shift of the sinusoidal headwind was varied manually, so that the duration between appearance and detection of the fault is nearly maximal. By using this adverse constellation of fault and environment, the reliability of the presented system can be tested.

State progression and the fault detection of **Fault 3** is shown in Figure 9. Increasing battery current leads to a steeper decrease of the state of charge in comparison to the non-fault progression beginning at 2 s.

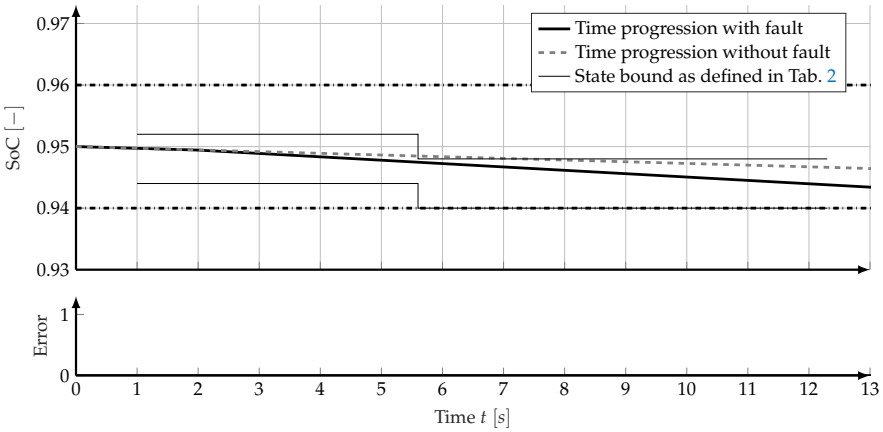

Figure 9. Fault 3.

Here, no fault can be detected in contrast to the cases of faults 1 and 2. Despite the fault, the SoC stays within the limits recorded in the discrete approximation. The reasons for this is the poor resolution of the cover of the discrete approximation in this component (cf. Table 2) and the comparatively 'slow' reacting state SoC. The scenario of fault 3 shows that the presented fault detection scheme has to be tested extensively for every application and simulation times of the discrete approximation are to be adjusted for proper work. Furthermore, the presented method can not guarantee complete reliability in case of uncertainties in model and measurement.

4. Discussion and Outlook

A plant model-based fault detection method was presented and applied to an aircraft takeoff scenario. Discrepancies between nominal and actual dynamics are detected utilizing a discrete approximation of the plant dynamics in form of a finite-state automaton. This discrete approximation is computed in a preliminary step. The fault detection module itself is less complex and can be executed on simple hardware.

To show an application of the presented fault detection method, faults during the takeoff run of a battery-electrically driven airplane have been simulated and detection capabilities have been analysed. The application example is based on a "black box" simulation model with which the discrete approximation is generated. It could be shown that with reasonable choices of the covers of the discrete approximation and the resolution of the test points the fault detection system performs reliably. Fault scenario 3 demonstrates that especially for 'slow reacting' states resolution must be enhanced to allow proper fault detection. Furthermore, it could be shown that the separation of the presented fault detection system in two segments simplifies its applicability. In spite of the long preliminary calculation period, the on board fault detection module provides real time capability. For applications with known dynamics (e.g., 'white box'), the presented method is perhaps not the most efficient moreover, but it can be utilized nevertheless.

The presented fault detection system offers further development potential. With small customization of the presented theory the system can be extended to include plant outputs or observable states in the fault monitoring instead of states. Therefore, more system information can be observed and the reliability of the system can be improved. For the demonstrated application, a plant output could be the resulting propeller RPM speed or the battery current. Moreover, the choice of the sampling time (denoted by τ in this note) should be investigated to reduce general computing costs on the one hand and to improve reliability of fault detection on the other hand. A decoupling of different cause-and-effect relationships between faults and states is explicitly not suggested. By using only one

FSA for the complete plant dynamics approximation, even faults with an unknown or unexpected cause-and-effect relationship can be detected. In time- and security critical processes, it is not necessary to know the exact cause of a failure. If any system parameter was not regular during the process, it should be aborted and the failure should be reconstructed under laboratory conditions.

Author Contributions: Conceptualization, A.W. and F.S.; methodology, A.W.; software, F.S.; validation, F.S.; investigation, A.W. and F.S.; resources, A.K; data curation, A.W. and F.S.; writing—original draft preparation, A.W. and F.S.; writing—review and editing, all authors; visualization, A.W. and F.S.; project administration, A.K.; funding acquisition, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie (project AURAIS) and Bundesministerium für Bildung und Forschung (project ARCUS).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Frank, P.M. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: A survey and some new results. *Automatica* **1990**, *26*, 459–474. [CrossRef]
2. Hwang, I.; Kim, S.; Kim, Y.; Seah, C.E. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Trans. Control Syst. Technol.* **2010**, *18*, 636–653. [CrossRef]
3. Isermann, R. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*; Springer Science & Business Media: Berlin, Germany, 2006.
4. De Persis, C.; Isidori, A. A geometric approach to nonlinear fault detection and isolation. *IEEE Trans. Autom. Control* **2001**, *46*, 853–865. [CrossRef]
5. Venkatasubramanian, V.; Chan, K. A neural network methodology for process fault diagnosis. *AIChE J.* **1989**, *35*, 1993–2002. [CrossRef]
6. Chand, S. Discrete-Event Based Monitoring and Diagnosis of Manufacturing Processes. In Proceedings of the American Control Conference, San Francisco, CA, USA, 2–4 June 1993; pp. 1508–1512. [CrossRef]
7. Bavishi, S.; Chong, E.K.P. Automated fault diagnosis using a discrete event systems framework. In Proceedings of the 9th IEEE International Symposium on Intelligent Control, Columbus, OH, USA, 16–18 August 1994; pp. 213–218. [CrossRef]
8. Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; Teneketzis, D. Failure diagnosis using discrete event models. In Proceedings of the 33rd IEEE Conference on Decision and Control, Lake Buena Vista, FL, USA, 14–16 December 1994; Volume 3, pp. 3110–3116. [CrossRef]
9. Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; Teneketzis, D.C. Failure diagnosis using discrete-event models. *IEEE Trans. Control Syst. Technol.* **1996**, *4*, 105–124. [CrossRef]
10. Ramkumar, K.; Philips, P.; Presig, H.A.; Ho, W.; Lim, K. Structured fault-detection and diagnosis using finite-state automaton. In Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, Aachen, Germany, 31 August–4 September 1998; Volume 3, pp. 1667–1672.
11. Ramkumar, K.; Druckenmuller, M.; Xi, Y.; Philips, P.; Presig, H.; Ho, W.; Lim, K. A fault-detection and diagnosis scheme by dynamic computation of finite-state automaton tables. In Proceedings of the 25th Annual Conference of the IEEE Industrial Electronics Society, San Jose, CA, USA, 29 November–3 December 1999; Volume 2, pp. 698–703.
12. Xi, Y.X.; Lim, K.W.; Ho, W.K.; Preisig, H.A. Fault diagnosis using dynamic finite-state automaton models. In Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society, Denver, CO, USA, 29 November–2 December 2001; Volume 1, pp. 484–489.
13. Lunze, J.; Schröder, J. State Observation and Diagnosis of Discrete-Event Systems Described by Stochastic Automata. *Discret. Event Dyn. Syst.* **2001**, *11*, 319–369. [CrossRef]
14. Lunze, J.; Schröder, J. Sensor and actuator fault diagnosis of systems with discrete inputs and outputs. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2004**, *34*, 1096–1107. [CrossRef] [PubMed]
15. Schmidt, K. Abstraction-based failure diagnosis for discrete event systems. *Syst. Control Lett.* **2010**, *59*, 42–47. [CrossRef]

16. Willems, J.C. Paradigms and puzzles in the theory of dynamical systems. *IEEE Trans. Autom. Control* **1991**, *36*, 259–294. [CrossRef]
17. Reissig, G.; Weber, A.; Rungger, M. Feedback Refinement Relations for the Synthesis of Symbolic Controllers. *IEEE Trans. Autom. Control* **2017**, *62*, 1781–1796. [CrossRef]
18. Junge, O.; Osinga, H.M. A set oriented approach to global optimal control. *ESAIM Control. Optim. Calc. Var.* **2004**, *10*, 259–270. [CrossRef]
19. Weber, A. Methoden zur Effizienzsteigerung Abstraktionsbasierter Reglerentwurfsverfahren. Ph.D. Thesis, Universität der Bundeswehr München, Verlag Dr. Hut, München, Germany, 2018.
20. Settele, F.; Bittner, M. Energy-optimal guidance of a battery-electrically driven airplane. *CEAS Aeronaut. J.* **2020**, *11*, 111–124. [CrossRef]
21. Ranter, H. Airliner Accident Statistics 2006. Aviation Safety Network. 2007. Available online: https://aviation-safety.net/pubs/asn/ASN_Airliner_Accident_Statistics_2006.pdf (accessed on 30 July 2019).
22. Flight Safety Foundation. Reducing the Risk of Runway Excursions—Report of the Runway Safety Initiative. 2009. Available online: https://www.skybrary.aero/bookshelf/books/900.pdf (accessed on 18 February 2019).
23. Flight International. Accident Reports Issued during the Second Half of 2018. 2019. Available online: https://finreader.flightglobal.com/publications-dist/1263/7943/2301/23183/article.html (accessed on 14 February 2019).
24. Srivatsan, R.; Downing, D.R.; Bryant, W.H. *Development of a Takeoff Performance Monitoring System*; Technical Memorandum; NASA: Hampton, VA, USA, 1986.
25. Srivatsan, R.; Downing, D.R.; Bryant, W.H. Development of a takeoff performance monitoring system. *J. Guid. Control Dyn.* **1987**, *10*, 433–440. [CrossRef]
26. Balachandran, S.; Atkins, E.M. An evaluation of flight safety assessment and management to avoid loss of control during takeoff. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, National Harbor, MD, USA, 13–17 January 2014; p. 0785.
27. Balachandran, S.; Atkins, E.M. Flight safety assessment and management for takeoff using deterministic Moore machines. *J. Aerosp. Inf. Syst.* **2015**, *12*, 599–615. [CrossRef]
28. Raisch, J.; O'Young, S.D. Discrete approximation and supervisory control of continuous systems. *IEEE Trans. Autom. Control* **1998**, *43*, 569–573. [CrossRef]
29. Reißig, G. Computing abstractions of nonlinear systems. *IEEE Trans. Autom. Control* **2011**, *56*, 2583–2598. [CrossRef]
30. Kapela, T.; Zgliczynski, P. A Lohner-type algorithm for control systems and ordinary differential inclusions. *Discret. Contin. Dyn. Syst. Ser. B* **2009**, *11*, 365. [CrossRef]
31. Breitenecker, F. Development of simulation software-from simple ode modelling to structural dynamic systems. In Proceedings of the 22nd European Conference on Modelling and Simulation (ECMS), Nicosia, Cyprus, 3–6 June 2008; pp. 20–37.
32. Weber, A.; Rungger, M.; Reissig, G. Optimized State Space Grids for Abstractions. *IEEE Trans. Autom. Control* **2017**, *62*, 5816–5821. [CrossRef]
33. Bai, Y.; Mallik, K.; Schmuck, A.; Zufferey, D.; Majumdar, R. Incremental Abstraction Computation for Symbolic Controller Synthesis in a Changing Environment. In Proceedings of the IEEE 58th Conference on Decision and Control (CDC), Nice, France, 11–13 December 2019; pp. 6261–6268.
34. *MATLAB Documentation;* MathWorks: Natick, MA, USA, 2019.