Procedia Computer Science

Volume XXX, 2015, Pages 1–10

ICCS 2015. International Conference On Computational Science

Dynamic Data-Driven Avionics Systems: Inferring Failure Modes from Data Streams

Shigeru Imai, Alessandro Galli, and Carlos A. Varela

Rensselaer Polytechnic Institute, Troy, NY 12180, USA
{imais,gallia}@.rpi.edu, cvarela@cs.rpi.edu

Abstract

Dynamic Data-Driven Avionics Systems (DDDAS) embody ideas from the Dynamic Data-Driven Application Systems paradigm by creating a data-driven feedback loop that analyzes spatio-temporal data streams coming from aircraft sensors and instruments, looks for errors in the data signaling potential failure modes, and corrects for erroneous data when possible. In case of emergency, DDDAS need to provide enough information about the failure to pilots to support their decision making in real-time. We have developed the PILOTS system, which supports data-error tolerant spatio-temporal stream processing, as an initial step to realize the concept of DDDAS. In this paper, we apply the PILOTS system to actual data from the Tuninter 1153 (TU1153) flight accident in August 2005, where the installation of an incorrect fuel sensor led to a fatal accident. The underweight condition suggesting an incorrect fuel indication for TU1153 is successfully detected with 100% accuracy during cruise flight phases. Adding logical redundancy to avionics through a dynamic data-driven approach can significantly improve the safety of flight.

Keywords: programming models, spatio-temporal data, data streaming

1 Introduction

Dynamic Data-Driven Avionics Systems (DDDAS) based on the concept of *Dynamic Data-Driven Application Systems* [1] use sensor data in real-time to enrich computational models in order to more accurately predict aircraft performance under partial failures. DDDAS can therefore support better-informed decision making for pilots and can also be applicable to autonomous unmanned air and space vehicles. As a first step towards realizing DDDAS, we have developed a system called *PILOTS* (**P**rogramm**I**ng **L**anguage for spati**O**-**T**emporal **S**treaming applications) [2, 3, 4] that enables specifying error detection and data correction in spatio-temporal data streams using a highly-declarative programming language. Figure 1 shows a conceptual view of DDDAS. Upon a request from the Avionics Application, the Pre-Processor takes raw data streams from aircraft sensors and then produces homogeneous and corrected streams. Following data pre-processing, the Avionics Application can constantly compute its

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

desired output with the corrected data. Since the Avionics Application controls the aircraft ultimately based on the raw data streams from the sensors, we can see that understanding data errors to detect potential sensor and instrument failures can significantly augment the envelope of operations of autonomous flight systems.

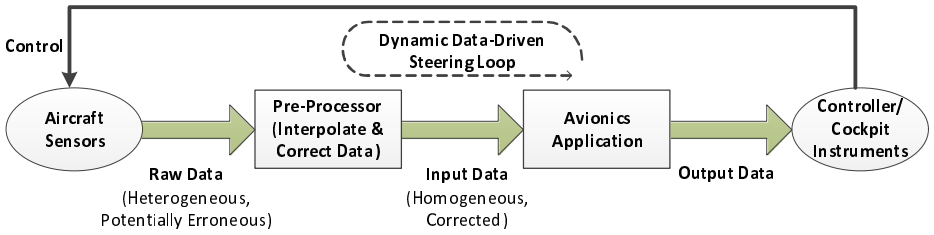


Figure 1: Conceptual view of Dynamic Data-Driven Avionics Systems

PILOTS has been successfully applied to detect the airspeed sensors failure which occurred in the Air France flight 447 accident from its recovered black box data [5]. In this paper, we analyze the TU1153 flight [6], where the installation of an incorrect fuel sensor led to a fatal accident. In the AF447 case, we used the relationship between airspeed, ground speed, and wind speed data to infer the pitot tubes failure. In the TU1153 case, we use the relationship between fuel weight, and aircraft performance (airspeed) data.

The rest of the paper is organized as follows. Section 2 describes *error signature*-based error detection and correction methods as well as the PILOTS programming language and the architecture of its runtime system. Section 3 discusses the design of error signatures and results of error detection performance for the TU1153 flight data, and Section 4 describes related work. Finally, we briefly describe future research directions for real-time error-tolerant stream processing and conclude the paper in Section 5.

2 Error-Tolerant Spatio-Temporal Data Streaming

2.1 Error Detection and Correction Methods

Error functions An error function is an arbitrary function that computes a numerical value from independently measured input data. It is used to examine the validity of redundant data. If the value of an error function is zero, we normally interpret it as no error in the given data.

The relationship between *ground speed* ($\overrightarrow{v_g}$), *airspeed* ($\overrightarrow{v_a}$), and *wind speed* ($\overrightarrow{v_w}$) shown in Equation 1 is visually depicted in Figure 2.

$$\overrightarrow{v_g} = \overrightarrow{v_a} + \overrightarrow{v_w}. \tag{1}$$

A vector $\overrightarrow{v}$ can be defined by a tuple (v, α), where v is the magnitude of $\overrightarrow{v}$ and α is the angle between $\overrightarrow{v}$ and a base vector. Following this expression, $\overrightarrow{v_g}, \overrightarrow{v_a}$, and $\overrightarrow{v_w}$ are defined as $(v_g, \alpha_g), (v_a, \alpha_a)$, and (v_w, α_w) respectively. We can compute $\overrightarrow{v_g}$ by applying trigonometry to $\triangle ABC$ and define an error function as the difference between measured v_g and computed v_g as follows:

$$e(\overrightarrow{v_g}, \overrightarrow{v_a}, \overrightarrow{v_w}) = |\overrightarrow{v_g} - (\overrightarrow{v_a} + \overrightarrow{v_w})| = v_g - \sqrt{v_a^2 + 2v_a v_w \cos(\alpha_a - \alpha_w) + v_w^2}. \tag{2}$$

The values of input data are assumed to be sampled periodically from corresponding spatio-temporal data streams. Thus, an error function e changes its value as time proceeds and can also be represented as $e(t)$.

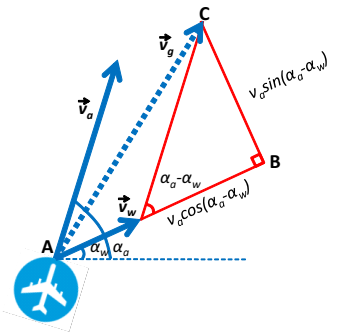


Figure 2: Trigonometry applied to the ground speed, airspeed, and wind speed.

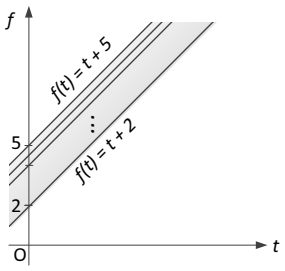


Figure 3: Error signature S_I with a linear function $f(t) = t + k, 2 \leqslant k \leqslant 5$.

Error signatures An *error signature* is a constrained mathematical function pattern that is used to capture the characteristics of an error function $e(t)$. Using a vector of constants $\bar{K} = \langle k_1, \ldots, k_m \rangle$, a function $f(t, \bar{K})$, and a set of constraint predicates $\bar{P} = \{p_1(\bar{K}), \ldots, p_l(\bar{K})\}$, the error signature $S(\bar{K}, f(t, \bar{K}), \bar{P}(\bar{K}))$ is defined as follows:

$$S(f(t, \bar{K}), \bar{P}(\bar{K})) \triangleq \{\ f \mid p_1(\bar{K}) \wedge \cdots \wedge p_l(\bar{K})\}. \tag{3}$$

For example, an interval error signature S_I can be defined as $S_I = S(f(t, \bar{K}), \bar{I}(\bar{K}, \bar{A}, \bar{B})) = \{\ f \mid\ a_1 \leqslant k_1 \leqslant b_1, \ldots, a_m \leqslant k_m \leqslant b_m\}$, where $\bar{A} = \langle a_1, \ldots, a_m \rangle$ and $\bar{B} = \langle b_1, \ldots, b_m \rangle$. When $f(t, \bar{K}) = t + k, \bar{K} = \langle k \rangle, \bar{A} = \langle 2 \rangle$, and $\bar{B} = \langle 5 \rangle$, the error signature S_I contains all linear functions with slope 1, and crossing the Y-axis at values $[2, 5]$ as shown in Figure 3.

Mode likelihood vectors Given a vector of error signatures $\langle S_0, \ldots, S_n \rangle$, we calculate $\delta_i(S_i, t)$, the distance between the measured error function $e(t)$ and each error signature S_i by:

$$\delta_i(S_i, t) = \min_{g(t) \in S_i} \int_{t-\omega}^{t} |e(t) - g(t)| dt. \tag{4}$$

where ω is the window size. Note that our convention is to capture "normal" conditions as signature S_0. The smaller the distance δ_i, the closer the raw data is to the theoretical signature S_i. We define the *mode likelihood vector* as $L(t) = \langle l_0(t), l_1(t), \ldots, l_n(t) \rangle$ where each $l_i(t)$ is:

$$l_i(t) = \begin{cases} 1, & \text{if } \delta_i(t) = 0 \\ \frac{\min\{\delta_0(t), \ldots, \delta_n(t)\}}{\delta_i(t)}, & \text{otherwise.} \end{cases} \tag{5}$$

Mode estimation Using the mode likelihood vector, the final mode output is estimated as follows. Observe that for each $l_i \in L, 0 < l_i \leqslant 1$ where l_i represents the ratio of the likelihood of signature S_i being matched with respect to the likelihood of the best signature. Because of the way $L(t)$ is created, the largest element l_j will always be equal to 1. Given a threshold $\tau \in (0, 1)$, we check for one likely candidate l_j that is sufficiently more likely than its successor l_k by ensuring that $l_k \leqslant \tau$. Thus, we determine j to be the correct mode by choosing the most likely error signature S_j. If $j = 0$ then the system is in *normal mode*. If $l_k > \tau$, then regardless of the value of k, *unknown error* mode (-1) is assumed.

Error correction Whether or not a known error mode i is recoverable is problem dependent. If there is a mathematical relationship between an erroneous value and other independently measured values, the erroneous value can be replaced by a new value computed from the

other independently measured values. In the case of the speed example used in Equations 1 and 2, if the ground speed v_g is detected as erroneous, its corrected value $\hat{v_g}$ can be computed by the airspeed and wind speed as $\hat{v_g} = \sqrt{v_a^2 + 2v_a v_w \cos(\alpha_a - \alpha_w) + v_w^2}$.

2.2 Spatio-Temporal Data Stream Processing System

2.2.1 System Architecture

Figure 4 shows the architecture of the PILOTS runtime system, which implements the error detection and correction methods described in Section 2.1. It consists of three parts: the *Data Selection*, the *Error Analyzer*, and the *Application Model* modules. The Application Model obtains homogeneous data streams $(d'_1, d'_2, \ldots, d'_N)$ from the Data Selection module, and then it generates outputs $(o_1, o_2, \ldots, o_M)$ and data errors $(e_1, e_2, \ldots, e_L)$. The Data Selection module takes heterogeneous incoming data streams $(d_1, d_2, \ldots, d_N)$ as inputs. Since this runtime is assumed to be working on moving objects, the Data Selection module is aware of the current location and time. Thus, it returns appropriate values to the Application Model by selecting or interpolating data in time and location, depending on the data selection method specified in the PILOTS program. The Error Analyzer collects the latest ω error values from the Application Model and keeps analyzing errors based on the error signatures. If it detects a recoverable error, then it replaces an erroneous input with the corrected one by applying a corresponding error correction equation. The Application Model computes the outputs based on the corrected inputs produced from the Error Analyzer.

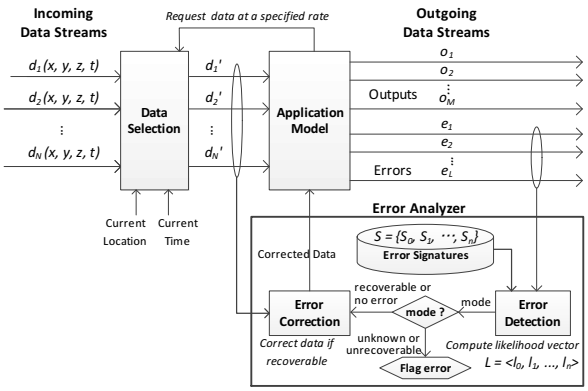


Figure 4: Architecture of the PILOTS runtime system.

2.2.2 PILOTS Programming Language

PILOTS is a programming language specifically designed for analyzing spatio-temporal data streams, which potentially contain erroneous data. Compiled PILOTS programs are designed to run on the runtime system described in Section 2.2.1. Using PILOTS, developers can easily program an application that handles spatio-temporal data streams by writing a high-level (declarative) program specification.

PILOTS application programs must contain `inputs` and `outputs` sections. The `inputs` section specifies the incoming data streams and how data is to be extrapolated from incomplete

data, typically using declarative geometric criteria (*e.g.*, `closest`, `interpolate`, `euclidean` keywords). Note that these extrapolations are performed based on the current location and time of the PILOTS system. The `outputs` section specifies outgoing data streams to be produced by the application, as a function of the input streams with a given frequency. `errors` and `signatures` sections are optional and can be used to detect errors. Similar to the `outputs` section, the `errors` section specifies an error stream to be produced by the application and to be analyzed by the runtime system to recognize known error patterns as described in the `signatures` section. If a detected error is recoverable, output values are computed from corrected input data using correction formulas under the `correct` section (See Figure 7 for an actual PILOTS program example).

3 Analyzing Tuninter 1153 Flight Data

Tuninter 1153 flight (TU1153) was a flight from Bari, Italy to Djerba, Tunisia on August 6th, 2005. About an hour after departure, the ATR 72 aircraft ditched into the Mediterranean Sea due to exhaustion of its fuel, killing 16 of 39 people on board (see Figure 5 for the altitude transition from the accident report [6]). The accident was caused by the installation of an incorrect fuel quantity indicator. That is, a fuel quantity indicator for the smaller ATR 42 was mistakenly installed, reporting 2150 kg more fuel than actually available. We call this incorrect indicated weight *fictitious weight* following the accident report.

How could DDDAS have prevented this accident from happening? If all other conditions are the same between two flights except for the weight, the one with lighter weight would have a higher airspeed. If we could compute expected airspeed from the monitored weight, we can compare the expected and monitored airspeed to detect if there is a discrepancy between the two. Once the system detects the discrepancy, it can warn pilots in an early stage of the flight to prevent the accident. Using this idea, we design a set of error signatures and evaluate it with actual data recorded in the flight data recorder (black box) during the TU1153 flight.

3.1 Error Signatures Design

In the ATR 72 flight crew operating manual [7], there are tables for pilots to estimate cruise airspeed (knots) under certain conditions of engine power settings, temperature difference to the International Standard Atmosphere (°C), flight level (feet), and weight (kg) of the aircraft, which are denoted by v_a, t_Δ, h, and w respectively. According to the accident report, $t_\Delta = +10$ is a reasonable approximation at the time of the accident. By using polynomial regression, we

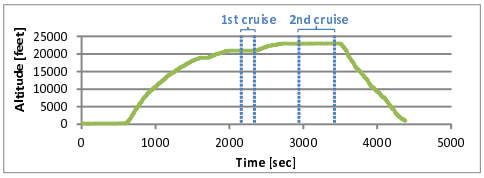


Figure 5: Altitude transition of the TU1153 flight.

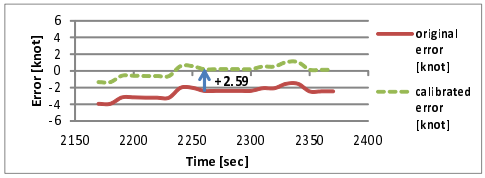


Figure 6: Calibration of the error function by the first cruise phase.

can derive a formula for the estimated airspeed $\hat{v_a}$ as follows:

$$\hat{v_a} = \mathbf{z}^{\mathrm{T}} \cdot \mathbf{k} = \begin{bmatrix} 1 \\ w \\ h \\ w \cdot h \\ w^2 \\ h^2 \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} 6.4869\mathrm{E}+01 \\ 1.4316\mathrm{E}-02 \\ 6.6730\mathrm{E}-03 \\ -3.7716\mathrm{E}-07 \\ -2.4208\mathrm{E}-07 \\ -1.1730\mathrm{E}-07 \end{bmatrix} \tag{6}$$

Using Equation 6, the error function is defined as the difference between the monitored (v_a) and estimated ($\hat{v_a}$) airspeed:

$$e(v_a, w, h) \quad = \quad v_a - \hat{v_a} \tag{7}$$

Figure 6 shows how the error function behaves for the first cruise phase from 2170 to 2370 seconds of the flight (see Figure 5 for the two cruise phases observed in the flight) using actual weight. As we can see from the original error plot, error values range from around -4 to -1.5 knots. This is expected because there are variables that are not considered in Equation 6 such as angle of attack, center of gravity, and aircraft tear and wear. To improve the overall fitness of the error function to the TU1153 data, we can "calibrate" the error function by adding a constant. In Equation 8, CP_1 is a set of data points during the first cruise phase and N is the number of data points in CP_1 (*i.e.*, $N = |CP_1|$). This equation is meant to find a constant that minimizes the squared difference between the error function and the constant itself. By subtracting the value of $K_{calib} = -2.59$ from Equation 7, we get a new calibrated error function $e_{calib} = e - K_{calib}$ as shown in the dotted-line of Figure 6.

$$K_{calib} = \underset{k}{\operatorname{argmin}} \frac{1}{N} \sum_{i \in CP_1}^{N} \parallel e_i - k \parallel^2 = \frac{1}{N} \sum_{i \in CP_1}^{N} e_i \tag{8}$$

Since the error function is adjusted to be zero after calibration, naturally we use the error value of zero with small margins to identify a normal condition. For underweight conditions, we set a constraint that will identify 10% discrepancy in weight regardless of the value of τ. Since the 10% weight difference leads to a 4.69 knots difference in airspeed using Equation 6 (computed by averaging over $w = 13000 \sim 22000$ kg at $h = 23000$ feet), we use 4.69 as the boundary for the underweight conditions. Note that from Equation 6, the estimated airspeed $\hat{v_a}$ monotonically decreases as the weight w increases for fixed h and t_Δ. Since cruise flights keep the same altitude, the error should be positive in underweight conditions as occurred in the TU1153 flight. In summary, we derive the error signature set shown in Table 1.

Table 1: Error signatures for the Tuninter 1153 flight.

Mode	Error Signature	
	Function	Constraints
Normal	$e = k$	$-2 < k < 2$
Underweight	$e = k$	$4.69 < k$

Note that this error signature set is not generally applicable, but specifically adjusted for the first cruise phase of the TU1153 flight. In a real-world application, we would use more data sets to derive a general error signature set for ATR 72 aircraft; however, we do not have access to actual ATR 72 flight data including airspeed, weight, altitude, and temperature. Therefore, we create a model for the cruise phases, adjust it to the first cruise phase, and evaluate it with the second cruise phase.

3.2 PILOTS Program

A PILOTS program called `WeightCheck` implementing the error signature set of Table 1 is presented in Figure 7. This program tries to detect an underweight condition by comparing the monitored airspeed and estimated airspeed computed from the weight and altitude. Once the program detects the underweight condition by the error signature `S1`, it estimates the corrected weight $\hat{w}$ by Equation 9, assuming airspeed, altitude, and temperature difference are all correct. This equation is obtained by solving Equation 6 for w after the calibration (*i.e.*, adding K_{calib} to the right-hand side of the equation).

$$\begin{aligned} \hat{w} &= \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \\ \text{where} \quad a &= \mathbf{k}(5),\ b = \mathbf{k}(2) + \mathbf{k}(4) \cdot h, \\ c &= \mathbf{k}(1) + \mathbf{k}(3) + \mathbf{k}(6) \cdot h^2 - v_a + K_{calib}. \end{aligned} \tag{9}$$

```
program WeightCheck;
  /* v_a: airspeed, w: weight, h: altitude */
  inputs
    v_a, w, h(t) using closest(t);

  outputs
    corrected_w: w at every 10 sec;

  errors
    e: v_a - (6.4869E+01 + 1.4316E-02 * w + 6.6730E-03 * h + (-3.7716E-07) * w * h +
              (-2.4208E-07) * w * w + (-1.1730E-07) * h * h) + 2.59;

  signatures
    S0(K): e = K, -2 < K, K < 2    "Normal";
    S1(K): e = K, 4.69 < K         "Underweight";

  correct
    S1: w = 3.34523E-12 * (sqrt(1.09278E+22 * h * h + (-1.65342E+27) * h +
                           (-3.69137E+29) * v_a + 1.01119E+32) - 2.32868E+11 * h +
                           8.83906E+15);
end
```

Figure 7: A declarative specification of the `WeightCheck` PILOTS program.

3.3 Evaluation

Flight Data: The airspeed, fictitious fuel weight, and altitude of the aircraft are collected from Attachment H of the accident report [6]. Since we need the total weight of the aircraft for our model to work, we compute it by adding the fuel weight to the zero fuel weight of the aircraft. As a result, before the departure, the fictitious weight of the aircraft is 19420 kg whereas the real weight is 17270 kg.

Evaluation Criteria: We evaluate the performance of error detection based on *accuracy* and *response time*, which are defined as follows:

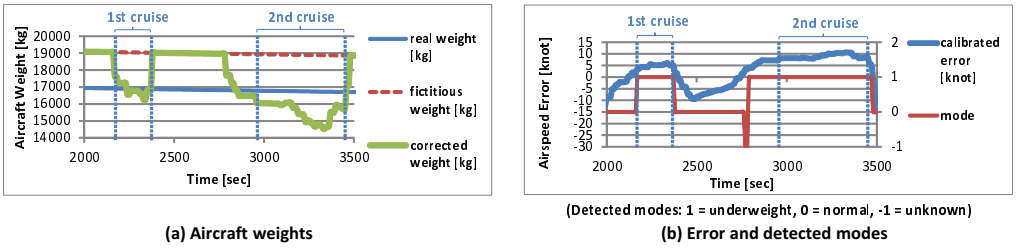
- **Accuracy**: This metric is used to evaluate how accurately the algorithm determines the true mode. Assuming the true mode transition $m(t)$ is known for $t = 0, 1, 2, \ldots, T$, let $m'(t)$ for $t = 0, 1, 2, \ldots, T$ be the mode determined by the error detection algorithm. We define $accuracy(m, m') = \frac{1}{T}\sum_{t=0}^{T} p(t)$, where $p(t) = 1$ if $m(t) = m'(t)$ and $p(t) = 0$ otherwise.

- **Maximum/Minimum/Average Response Time**: This metric is used to evaluate how quickly the algorithm reacts to mode changes. Let a tuple (t_i, m_i) represent a mode change point, where the mode changes to m_i at time t_i. Let $M = \{(t_1, m_1), (t_2, m_2), \ldots, (t_N, m_N)\}$ and $M' = \{(t'_1, m'_1), (t'_2, m'_2), \ldots, (t'_{N'}, m'_{N'})\}$ be the sets of true mode changes and detected mode changes respectively. For each $i = 1 \ldots N$, we can find the smallest t'_j such that $(t_i \leqslant t'_j) \wedge (m_i = m'_j)$; if not found, let t'_j be t_{i+1}. The response time r_i for the true mode m_i is given by $t'_j - t_i$. We define the maximum, minimum, and average response times by $\max_{1 \leqslant i \leqslant N} r_i$, $\min_{1 \leqslant i \leqslant N} r_i$, and $\frac{1}{N} \sum_{i=1}^{N} r_i$ respectively.

Experimental Settings: We run the `WeightCheck` PILOTS program in Figure 7 for 1500 seconds, which is from 2000 to 3500 seconds after the departure including the first (2170–2370 seconds) and the second cruise phase (2960–3450 seconds), to see how the error signature set works for these two cruise phases. On the other hand, we evaluate only the second cruise phase since the error signature set is adjusted by the first cruise phase. The true mode changes for the second cruise phase is defined as $M = \{(2960, 1)\}$. The accuracy and average response time are investigated for window sizes $\omega \in \{1, 2, 4, 8, 16\}$ and threshold $\tau \in \{0.2, 0.4, 0.6, 0.8\}$.

Results: Figure 8 shows the transitions of (a) aircraft weights and (b) error and detected modes when $\omega = 1$ and $\tau = 0.8$. As shown in Figure 8(b), the PILOTS program correctly detects the underweight condition for the first cruise phase whereas it detects the underweight condition well before the second cruise phase starts. This is because the error goes beyond 4.69 (the boundary for the underweight condition) around 2770 seconds. From Figure 8(a), we can tell that corrected weight is reasonably close to the real weight during the first cruise phase, but is not very close during the second phase. That is, the differences between the corrected and real weights are at most -643 kg for the first cruise phase and -1935 kg for the second cruise phase. This corrected weight discrepancy between the two phases can be explained by inaccuracy of the airspeed estimation during the second cruise phase. This result reveals that our airspeed prediction model is not accurate enough to precisely estimate the airspeed from the weight, but nonetheless, it is able to detect the underweight condition.

When we ran the experiments for accuracy and response time, we noticed that the accuracy was 1.0 and response time was 0 second for all combinations of ω and τ. As explained above, the PILOTS program recognizes the correct mode (=1, underweight) for the entire second cruise phase, which makes the accuracy 1.0. Since there is a 190 seconds (=19 window periods) buffer between when the program starts recognizing the underweight condition at 2770 seconds and when the actual second cruise phase starts at 2960 seconds, changing the window size ω from 1 to 16 does not affect the response time at all. Assuming the ground truth mode for the non cruise phases is 0 (*i.e.*, normal mode), the accuracy goes down to 0.84 for all mode detection period (*i.e.*, 2000-3500 seconds).



(a) Aircraft weights

(b) Error and detected modes

Figure 8: Aircraft weights and detected modes for the TU1153 flight ($\omega = 1, \tau = 0.8$).

4 Related Work

There have been many systems that combine data stream processing and data base management, *i.e.*, Data Stream Management Systems (DSMS). PLACE [8] and Microsoft StreamInsight [9] are DSMS-based systems supporting spatio-temporal streams. These DSMS-based spatio-temporal stream management systems support general continuous queries for multiple moving objects such as "Find all the cars running within a diameter of X from a point Y in the past Z time". Unlike these DSMS-based systems which handle multiple spatio-temporal objects, a PILOTS program is assumed to be moving and tries to extrapolate data that is relevant to the current location and time. This approach narrows down the applicability of PILOTS; however, users can more easily design error signatures to correct data on the fly thanks to the declarative programming approach.

Fault detection, isolation, and reconfiguration (FDIR) has been actively studied in the control community [10]. Mission critical systems, such as nuclear power plants, flight control systems, and automotive systems, are main application systems of FDIR. FDIR systems 1) generate a set of values called *residuals* and determine if a fault has occurred based on residuals, 2) identify the type of the fault, and 3) reconfigure the system accordingly. To alleviate the effect of noise on residuals, robust residual generation techniques, such as a Kalman-filter based approach [11], have been used. Since the PILOTS system is a fault-tolerance system that endures data failures, it is expected PILOTS' framework has resemblance to FDIR systems. However, due to PILOTS' domain-specific programming language approach, PILOTS users have the ability to control error conditions more generally through error signatures, which is not part of the FDIR framework.

5 Conclusion and Future Work

Supplementing flight systems with error detection and data correction based on error signatures can add another layer of (logical) fault-tolerance and therefore make airplane flights safer. With a few tens of lines of PILOTS programs, a data scientist can test and validate their error detection models in the form of error signatures. In previous work [3], we showed our error signature-based approach was able to detect and correct the airspeed sensor failure in the Air France 447 flight accident. In this paper, an evaluation of the Tuninter 1153 flight accident illustrates the fuel quantity indicator failure could also have been detected.

Scalability becomes paramount as the number of data streams to be analyzed increases–*e.g.*, due to the increasing number of sensors in aircraft–and also, as we capture more complex aircraft failure models as error signatures and damaged aircraft performance profiles. Future work includes exploring distributed computing as a mechanism to scale the performance and quality (*e.g.*, see [12, 13]) of online (real-time) data analyses. It is also important to investigate high-level abstractions (*e.g.*, see [14]) that will enable data scientists and engineers to more easily develop concurrent software to analyze data, and that will facilitate distributed computing optimizations. Finally, uncertainty quantification [15, 16] is an important future direction to associate confidence to data and error estimations in support of decision making.

Acknowledgments This research is partially supported by the DDDAS program of the Air Force Office of Scientific Research, Grant No. FA9550-11-1-0332 and a Yamada Corporation Fellowship.

References

[1] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *Computational Science-ICCS 2004*, pp. 662–669, Springer, 2004.

[2] S. Imai and C. A. Varela, "A programming model for spatio-temporal data streaming applications," in *DDDAS 2012 Workshop at ICCS'12*, (Omaha, Nebraska), pp. 1139–1148, June 2012.

[3] R. S. Klockowski, S. Imai, C. Rice, and C. A. Varela, "Autonomous data error detection and recovery in streaming applications," in *DDDAS 2013 Workshop at ICCS'13*, pp. 2036–2045, May 2013.

[4] S. Imai, R. Klockowski, and C. A. Varela, "Self-healing spatio-temporal data streams using error signatures," in *2nd International Conference on Big Data Science and Engineering (BDSE 2013)*, (Sydney, Australia), December 2013.

[5] Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile, "Flight AF447 on 1st June 2009, A330-203, registered F-GZCP." `http://www.bea.aero/en/enquetes/flight.af.447/rapport.final.en.php`, 2012.

[6] ANSV - Agenzia Nazionale per la Sicurezza del Volo, "Final Report: Accident involving ATR 72 aircraft registration marks TL-LBB ditching off the coast of Capo Gallo (Palermo - Sicily)." `http://www.ansv.it/cgi-bin/eng/FINAL%20REPORT%20ATR%2072.pdf`, August 2005.

[7] ATR, *ATR72 - Flight Crew Operating Manual*. Aerei da Trasporto Regionale, July 1999.

[8] M. F. Mokbel, X. Xiong, W. G. Aref, and M. a. Hammad, "Continuous query processing of spatio-temporal data streams in PLACE," *GeoInformatica*, vol. 9, pp. 343–365, 2005.

[9] M. H. Ali, B. Chandramouli, B. S. Raman, and E. Katibah, "Spatio-temporal stream processing in Microsoft StreamInsight," *IEEE Data Eng. Bull.*, pp. 69–74, 2010.

[10] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE Trans. Control Systems Technology*, vol. 18, no. 3, pp. 636–653, 2010.

[11] T. E. Menke and P. S. Maybeck, "Sensor/actuator failure detection in the Vista F-16 by multiple model adaptive estimation," *IEEE Trans. Aerospace and Electronic Systems*, vol. 31, no. 4, pp. 1218–1229, 1995.

[12] K. E. Maghraoui, T. Desell, B. K. Szymanski, and C. A. Varela, "The Internet Operating System: Middleware for adaptive distributed computing," *International Journal of High Performance Computing Applications (IJHPCA)*, vol. 20, no. 4, pp. 467–480, 2006.

[13] S. Imai, T. Chestna, and C. A. Varela, "Elastic scalable cloud computing using application-level migration," in *5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2012)*, (Chicago, Illinois, USA), November 2012.

[14] C. A. Varela, *Programming Distributed Computing Systems: A Foundational Approach*. MIT Press, May 2013.

[15] E. Prudencio, P. Bauman, S. Williams, D. Faghihi, K. Ravi-Chandar, and J. Oden, "Real-time inference of stochastic damage in composite materials," *Composites Part B: Engineering*, vol. 67, pp. 209–219, 2014.

[16] D. Allaire, D. Kordonowy, M. Lecerf, L. Mainini, and K. Willcox, "Multifidelity DDDAS methods with application to a self-aware aerospace vehicle," in *DDDAS 2014 Workshop at ICCS'14*, pp. 1182–1192, June 2014.