Self-Healing Data Streams Using Multiple Models of Analytical Redundancy

Shigeru Imai, Frederick Hole, and Carlos A. Varela
Department of Computer Science, *Rensselaer Polytechnic Institute*
shigeru.imai1@gmail.com, holef@rpi.edu, cvarela@cs.rpi.edu

***Abstract*—We have created a highly declarative programming language called PILOTS that enables error detection and estimation of correct data streams based on analytical redundancy (*i.e.*, algebraic relationship between data streams). Data scientists are able to express their analytical redundancy models with the domain specific grammar of PILOTS and test their models with erroneous data streams. PILOTS has the ability to express a single analytical redundancy, and it has been successfully applied to data from aircraft accidents such as Air France flight 447 and Tuninter flight 1153 where only one simultaneous sensor type failure was observed. In this work, we extend PILOTS to support multiple models of analytical redundancy and improve situational awareness for multiple simultaneous sensor type failures. Motivated by the two recent accidents involving the Boeing 737 Max 8, which was potentially caused by a faulty angle of attack sensor, we focus on recovering angle of attack data streams under multiple sensor type failure scenarios. The simulation results show that multiple models of analytical redundancy enable us to detect failure modes that are not detectable with a single model.**

***Index Terms*—fault-tolerance; data streams; analytical redundancy**

I. Introduction

Two recent airplane accidents, Lion Air flight 610 on October 29, 2018 and Ethiopian Airlines flight 302 on March 10, 2019, involved fatal crashes of the Boeing 737 Max 8. According to a preliminary report on the Lion Air accident [1], [2], the pilots experienced inconsistent altitude and airspeed data readings due to erroneous angle of attack data on the day of the flight. It is speculated that erroneous angle of attack data caused the Max 8's Maneuvering Characteristics Augmentation System (MCAS) to malfunction. The pilots experienced repeated automatic nose down trim despite the fact that they manually commanded nose up trim to avoid stalling. The causes of these two accidents are still under active investigation; however, if the 737 Max 8 aircraft was aware of the erroneous angle of attack data and had the ability to automatically recover the correct data, it could have reduced the risk of accidents significantly.

In modern aircraft systems, triplex sensors are used to provide a fault tolerant signal fusion scheme called *signal consolidation* [3]. One way to implement the signal consolidation is to take a weighted sum of the values from three sensors. However, when all three sensors are failed, this scheme cannot produce reliable data. It indeed happened to Air France flight 447: When the aircraft went into a thunderstorm, all the pitot tubes iced and produced abnormally low values [4]. One way to overcome the limitation of this physical redundancy is to adopt *analytical redundancy* [5], [6], which is an algebraic relationship between multiple data streams. Even if all the sensors of type X fail, we may be able to estimate the true value of X using a mathematical relationship f with other sensor streams Y and Z (*i.e.*, $\hat{X} = f(Y, Z)$).

We have created a highly declarative programming language called PILOTS [7], [8][1] that enables error detection and estimation of correct data streams based on analytical redundancy. PILOTS implements the concept of Dynamic Data-Driven Application Systems [9], [10], [11], [12]. Data scientists are able to express their error detection and data estimation models with the domain specific grammar of PILOTS and test their models with erroneous data streams. PILOTS has the ability to express a single model of analytical redundancy, and it has been successfully applied to aircraft accidents such as Air France flight 447 [13] and Tuninter flight 1153 [14] where only one simultaneous sensor type failure was observed. Assuming there is no error on airspeed, we can recover angle of attack data from airspeed using an algebraic relationship based on the lift coefficient equation; however, there is no guarantee that airspeed is always correct.

In this work, we enhance PILOTS to support multiple models of analytical redundancy and improve situational awareness for multiple simultaneous sensor type failures. In particular, we consider situations where any combinations of three types of sensors, GPS, pitot-static system, and angle of attack sensor, can fail. Combining 1) an analytical redundancy model on ground speed, airspeed, and wind speed, and 2) another analytical redundancy model on airspeed and angle of attack, we estimate the true failure modes of sensors through PILOTS programs. To evaluate the mode estimation accuracy, we produce test data streams using the X-Plane flight simulator [15] and simulate multiple sensor type failure scenarios. The simulation results show that multiple models of analytical redundancy enable us to detect failure modes that are not detectable just by using a single model.

The rest of the paper is organized as follows. We first describe an overview of our PILOTS programming language and its error detection method in Section II. In Section III, we show analytical redundancy models on 1) speed data and 2) airspeed and angle of attack data, and show how we

[1]PILOTS is an open-source software and downloadable at: http://wcl.cs.rpi.edu/pilots/.

implement these models separately on two PILOTS programs using existing PILOTS capabilities. In Section IV, we enhance PILOTS with the support for multiple models of analytical redundancy and show three PILOTS program options to implement these models. In Section V, we evaluate the proposed PILOTS programs with multiple sensor type failure scenarios. We present related work in Section VI and conclude the paper in Section VII.

II. PILOTS: A Programming Language for Self-Healing Data Streams

PILOTS is a highly-declarative, domain-specific programming language with self-healing capability [7]. It is designed to be used for data streaming applications in avionics. PILOTS application programs must contain inputs and outputs sections. The inputs section specifies the incoming data streams and how data is to be interpolated and/or extrapolated from incomplete data, typically using declarative geometric criteria (*e.g.*, closest, interpolate, euclidean keywords). 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 error streams to be produced by the application and to be analyzed by the runtime system to recognize known *error signatures* [13] as described in the signatures section. If a detected error is recoverable, output values are computed from corrected input data using estimation formulas under the estimate clause.

A. Example PILOTS Program

Fig. 1 shows a primitive PILOTS program called Twice. As its name suggests, it takes two input streams $a(t)$ and $b(t)$, where b is supposed to be twice as large as a. Both a and b are expected to increase by one for a and two for b every second (*i.e.*, $a(t) = t + k$ and $b(t) = 2t + k$, where t is time and k is a constant). Thus, the error is zero in the Normal mode. Suppose a is failed and keeps producing the last observed value, the error keeps increasing with the slope of 2. Similarly, when b is failed and keeps producing the last observed value, the error keeps decreasing with the slope of -2. We can express these behaviors as error signatures: $e = 2t + k$ and $e = -2t + k$ respectively for A failure and B failure. Once we compute error in e as specified in the errors section, we compute the relative distances from e to each signature and estimate the most likely error mode. Once a mode is determined, the original data is estimated by the application model as shown in the estimate clauses: $a = b/2$ for A failure and $b = 2a$ for B failure.

```
program Twice;
    inputs
        a (t) using closest(t);
        b (t) using closest(t);
    outputs
        o: (b - 2*a) at every 1 sec;
    errors
        e: b - 2*a;
    signatures
       s0: e =  0          "Normal";
       s1: e =  2*t + k    "A failure"
           estimate a = b / 2;
       s2: e = -2*t + k    "B failure"
           estimate b = 2*a;
end
```

Fig. 1. Twice PILOTS program.

B. Error Detection by Error Signatures

An *error signature* is a constrained mathematical function pattern that is used to capture the characteristics of an error function $e(t)$ [2]. Looking at the example used in Fig. 1, the error signature s1: $e = 2t + k$ represents a set of all linear functions with slope of 2. Other examples of error signatures include:

1) $e = 2t + k, k < 5$,
2) $e = k, 0 < k, k < 10$.

The first example is a set of all linear functions with slope of 2 and y-intercept less than 5. The second one represents a range of constants $k \in (5, 10)$.

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:

$$\delta_i(S_i, t) = \min_{g(t) \in S_i} \int_{t-\omega}^{t} |e(t) - g(t)| dt, \tag{1}$$

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:

$$l_i(t) = \begin{cases} 1, & \text{if } \delta_i(t) = 0 \\ \frac{\min\{\delta_0(t), \ldots, \delta_n(t)\}}{\delta_i(t)}, & \text{otherwise.} \end{cases} \tag{2}$$

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.

III. Self-Healing Data Streams with Single Model of Analytical Redundancy

In this section, we focus on two examples, 1) speed data and 2) angle of attack data, to illustrate self-healing data streams with a *single* analytical redundancy model. For each example, we first show analytical redundancy that exists in the data streams, describe how to detect errors under some

[2]For the detailed definition, see [7]

assumptions on how sensors fail, and show a PILOTS program implemented with the analytical redundancy model.

A. Self-Healing Speed Data

1) Analytical Redundancy: We consider a physics-based relationship that exists between airspeed $\vec{v}_a$, ground speed $\vec{v}_g$, and wind speed $\vec{v}_w$. The airspeed is the speed of an aircraft relative to the air mass, and the ground speed is the speed of an aircraft relative to the ground. When there is wind, the air mass is affected by the wind, and the ground speed $\vec{v}_g$ can be determined by the vector sum of the airspeed $\vec{v}_a$ and wind speed $\vec{v}_w$ as follows:

$$\vec{v}_g = \vec{v}_a + \vec{v}_w. \tag{3}$$

Using trigonometry, the following relationship on the magnitude of ground speed holds:

$$\begin{aligned}\hat{v}_g(\vec{v}_a, \vec{v}_w) &= \hat{v}_g(v_a, \alpha_a, v_w, \alpha_w) \\ &= \sqrt{v_a^2 + v_w^2 + 2v_a v_w \cos(\alpha_w - \alpha_a)},\end{aligned} \tag{4}$$

where $\hat{v}_g$ is the estimated ground speed computed from airspeed v_a and its direction α_a, and wind speed v_w and its direction α_w. As long as all the speed data are consistent (*i.e.*, have no error), the difference between v_g and $\hat{v}_g$ should be zero. We define the difference by the following error e_1 and use it to monitor whether speed data are consistent:

$$e_1 = v_g - \hat{v}_g(v_a, \alpha_a, v_w, \alpha_w). \tag{5}$$

2) Error Detection: The advances of ADS-B [16] has enabled an aircraft to obtain weather information from nearby peer aircrafts and also from ground controllers. Given multiple weather data from different sources, we can eliminate erroneous weather data using a simple technique such as majority vote. Thus, we assume there is no error on wind data and only consider the following four error modes:

- Normal: no error on all data.
- Pitot tube failure: error on airspeed v_a.
- GPS failure: error on ground speed v_g.
- Pitot tube and GPS failures: error on v_a and v_g.

We parameterize how these failures occur as follows:

- Wind speed is usually very small compared to airspeed: $v_w = av_a$, where a is the small wind to airspeed ratio. This means that ground speed is approximately same as airspeed: $v_g \approx v_a$.
- Pitot tube produces airspeed that is proportional to how much the pitot tube is cleared. Failed airspeed $v_a^{\text{fail}} = bv_a$, where $b \in [b_l, b_h], 0 \leqslant b_l \leqslant b_h \leqslant 1$. $b = 0$ represents a fully clogged pitot tube, while $b = 1$ represents a fully clear pitot tube.
- When GPS fails, it produces zero ground speed: $v_g^{\text{fail}} = 0$.

To identify the above error modes under the assumptions on how pitot tube and GPS fail, we have previously designed error signatures for speed data in [13]. We show them in Table I.

TABLE I
ERROR SIGNATURES FOR SPEED DATA [13].

Mode	Error signature
Normal	$e_1 \in [-av_a, av_a]$
Pitot tube failure	$e_1 \in [(1-a-b_h)v_a, (1-\lvert a-b_l\rvert)v_a]$
GPS failure	$e_1 \in [-(a+1)v_a, -\lvert a-1\rvert v_a]$
Pitot and GPS failures	$e_1 \in [-(a+b_h)v_a, -\lvert a-b_l\rvert v_a]$

3) Error Recovery: In case of the GPS failure mode, we can estimate ground speed by Eq. 4. In case of the Pitot tube failure mode, we can estimate airspeed by the following equation:

$$\hat{v}_a(\vec{v}_g, \vec{v}_w) = \sqrt{v_g^2 + v_w^2 - 2v_g v_w \cos(\alpha_g - \alpha_w)}. \tag{6}$$

If the Pitot and GPS failures mode is detected, we cannot recover either ground speed or airspeed as we do not have enough redundancy between speed data.

4) PILOTS Program: In SpeedCheck program in Fig. 2, the error signatures defined in Table I are translated into code under the signatures section. Note that we plug in $v_a = 110$ knots (denoted V_CRUISE in the program), $a = 0.1$, $b_l = 0.2$, and $b_h = 0.33$ when translating the signatures into the program. In case of the pitot tube failure or GPS failure, we estimate the true airspeed or ground speed using Eqs. (6) or (4) as shown under the estimate clauses. The mode variable in the outputs section is reserved to store the estimated mode.

```
program SpeedCheck;
  /* va: airspeed,     aa: airspeed angle,
     vw: wind speed,   aw: wind speed angle,
     vg: ground speed, ag: ground angle */
  inputs
    va, vg, vw (t) using closest(t);
    aa, ag, aw (t) using closest(t);
  constants /* For Cessna 172 SP */
    V_CRUISE    =   110;
    NORMAL_L    =  -0.1 * V_CRUISE;
    NORMAL_H    =   0.1 * V_CRUISE;
    PITOT_L     =  0.57 * V_CRUISE;
    PITOT_H     =   0.9 * V_CRUISE;
    GPS_L       =  -1.1 * V_CRUISE;
    GPS_H       =  -0.9 * V_CRUISE;
    GPS_PITOT_L = -0.43 * V_CRUISE;
    GPS_PITOT_H =  -0.1 * V_CRUISE;
  outputs
    va, vg, mode at every 1 sec;
  errors
    e1: vg - sqrt(va^2 + vw^2 +
                  2*va*vw*cos((PI/180)*(aw-aa)));
  signatures
    s0: e1 = k, NORMAL_L < k, k < NORMAL_H "Normal";
    s1: e1 = k, PITOT_L  < k, k < PITOT_H
      "Pitot tube failure"
      estimate va = sqrt(vg^2 + vw^2 -
        2*vg*vw*cos((PI/180)*(ag-aw)));
    s2: e1 = k, GPS_L    < k, k < GPS_H "GPS failure"
      estimate vg = sqrt(va^2 + vw^2 +
        2*va*vw*cos((PI/180)*(aw-aa)));
    s3: e1 = k, GPS_PITOT_L < k, k < GPS_PITOT_H
      "GPS + Pitot tube failure";
end;
```

Fig. 2. SpeedCheck PILOTS program.

B. Self-Healing Angle of Attack Data

1) Analytical Redundancy: The lift force L is defined as

$$L = C_\ell \frac{\rho v_a^2}{2} S, \tag{7}$$

where C_ℓ is the lift coefficient, ρ is the air density, v_a is the airspeed, and S is the wing surface area. Solving Eq. (7) for v_a, we get

$$v_a = \sqrt{\frac{2L}{C_\ell \cdot \rho S}}. \tag{8}$$

Using the fact that C_ℓ can also be modeled as a function of angle of attack α, we define analytical redundancy between airspeed v_a and angle of attack α. Depending on to what extent we use a data-driven approach, we consider the following three models.

Model 1 *Thin air foil theory*: According to the thin airfoil theory [17], coefficient of lift is linearly approximated by $C_\ell = 2\pi\alpha + c_{\ell_0}$, where c_{ℓ_0} is the lift coefficient solely determined by the shape of the wing when $\alpha = 0$. By plugging in this formula to Eq. (8), Model 1 is defined as

$$\hat{v}_a(\alpha) = \sqrt{\frac{2L}{(2\pi\alpha + c_{\ell_0})\rho S}}. \tag{9}$$

Model 2 *Linear least square approximation of coefficient of lift*: The coefficient of lift C_ℓ can be approximated by a linear function of angle of attack α until α hits the stall angle. After learning a linear function by minimizing square error with respect to training samples of coefficient of lift, Model 2 is defined as

$$\hat{v}_a(\alpha) = \sqrt{\frac{2L}{(k_1\alpha + k_2)\rho S}}, \tag{10}$$

where k_1 and k_2 are constants obtained as the result of linear least square.

Model 3 *Non-linear least square approximation of airspeed*: Given a set of training data for airspeed v_a and angle of attack α, we can directly approximate the relationship between v_a and α. Keeping the general form of Eq. (8), Model 3 is defined as

$$\hat{v}_a(\alpha) = \sqrt{\frac{k_1}{k_2\alpha + k_3}}, \tag{11}$$

where k_1, k_2, and k_3 are constants obtained as the result of non-linear least square approximation.

We define error e_2 as the difference between the monitored airspeed v_a and the airspeed estimated from angle of attack $\hat{v}_a(\alpha)$ as follows.

$$e_2(\alpha) = v_a - \hat{v}_a(\alpha). \tag{12}$$

2) Error Detection & Recovery: To detect errors on angle of attack, we check whether e_2 is within a certain range $[r_l, r_h]$. Assuming there is no error on airspeed, if e_2 is out of this range, we detect it as an error on angle of attack. When we estimate the true angle of attack value, we use analytical redundancy between airspeed and angle of attack. For example, we can derive the estimated true value of angle of attack $\hat{\alpha}$ for Model 1 as follows:

$$\hat{\alpha}(v_a) = \frac{L}{\pi\rho v_a^2 S} - \frac{c_{\ell 0}}{2\pi}. \tag{13}$$

3) PILOTS Program: In AoaCheck program in Fig. 3, we check if the computed error e_2 is within $\pm 10\%$ of the cruise airspeed. If the error is outside the range, we estimate the true angle of attack using Eq. (13) as shown under the estimate clause.

```
program AoaCheck;
  /* va: airspeed, aoa: angle of attack */
  inputs
    va, aoa (t) using closest(t);
  constants /* For Cessna 172 SP */
    MPS2KNOT = 1.94384;
    G        = 9.81;
    L        = 1156.6 * G;
    RHO      = 16.2;
    S        = 1.225;
    CL0      = 0.2279;
    V_CRUISE = 110;
    NORMAL_L = -0.10 * V_CRUISE;
    NORMAL_H =  0.10 * V_CRUISE;
  outputs
    va, aoa, mode at every 1 sec;
  errors
    e2: va - MPS2KNOT*
             sqrt(2*L/(2*PI*(PI/180)*aoa + CL0)*S*RHO);
  signatures
    s0: e2 = k, NORMAL_L < k, k < NORMAL_H  "Normal";
    s1: e2 = k, k < NORMAL_L, NORMAL_H < k
      "AoA sensor failure"
      estimate aoa = L/(PI*RHO*va^2*S) - CL0/(2*PI);
end;
```

Fig. 3. AoaCheck PILOTS program for Model 1.

IV. Self-Healing Data Streams with Multiple Models of Analytical Redundancy

The AoaCheck program presented in Section III assumes airspeed is always correct. However, pitot tube sensor failures can occur at any time, and thus estimated modes from the AoaCheck program are not totally reliable. In this section, we enhance PILOTS to support multiple models of analytical redundancy, aiming to improve situational awareness upon failures of sensors of multiple types. Here is a summary of the new features for the PILOTS programming language to support multiple models of analytical redundancy.

- Support for multiple error values (*i.e.*, analytical redundancy models) under the errors section
- Enhancement of the modes section to support general boolean expressions[3]
- Scoped naming for output variables from child programs (*e.g.*, variable x from program A can be referred to as A.x in the parent program)
- Reserved mode variable to store the estimated mode

[3]The modes section was first introduced in [8].

A. Error Detection

Fig. 4 shows the relationship between the two error functions, e_1 and e_2, which we have shown in Eqs. (5) and (12), respectively. We expect one or more sensors of the GPS, pitot tube, and angle of attack sensor fail at any time whereas we assume other speed data v_w, α_a, α_w do not fail. Since v_a is

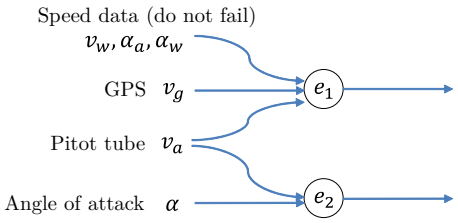

Fig. 4. Relationship between two error functions.

used by both error functions, when a pitot tube failure occurs, both e_1 and e_2 are expected to be non-zero. On the other hand, when a pitot tube failure does not occur, e_2 solely depends on α, and it allows us to determine whether there is an error on α. This is an improvement over using the single AoaCheck program, which makes mode detection decisions without the ability to know whether the pitot tube may have failed.

The ground truth modes and corresponding detectable modes by the SpeedCheck and AoaCheck programs are summarized in Table II. When there is no error on the pitot tube, the associated ground truth modes, $0, 1, 4$, and 5, are uniquely identifiable by the combination of SpeedCheck and AoaCheck modes, $(0,0), (0,1), (2,0)$, and $(2,1)$. However, when there is an error on the pitot tube, modes 2 and 3 are not separable by SpeedCheck and AoaCheck. That is because depending on the value of e_2 used in AoaCheck, SpeedCheck and AoaCheck can produce the same combination of modes $(1,1)$ for both modes 2 and 3. For the same reason, modes 6 and 7 are not separable too.

TABLE II
GROUND TRUTH MODES FOR GPS, PITOT TUBE, AND ANGLE OF ATTACK (AOA) SENSORS (0: NOT FAILED, 1: FAILED) AND RESPECTIVE DETECTABLE MODES BY SpeedCheck AND AoaCheck PROGRAMS.

Ground truth				Detectable modes	
Mode	GPS	Pitot tube	AoA	SpeedCheck	AoaCheck
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	1	1
3	0	1	1	1	0,1,-1
4	1	0	0	2	0
5	1	0	1	2	1
6	1	1	0	3	1
7	1	1	1	3	0,1,-1

B. PILOTS Program Options

Using the approach described in Section IV-A, we implement PILOTS programs to detect one or more combination of GPS, pitot tube, and angle of attack sensor failures. We consider three implementation options as shown in Fig. 5. AoaSpeedCheck1 directly receives all the input data and detects error modes by computing e_1 and e_2 internally. AoaSpeedCheck2 detects error modes based on the outputs from SpeedCheck and AoaCheck. AoaSpeedCheck3 detects error modes based on the outputs from SpeedCheck and angle of attack data.

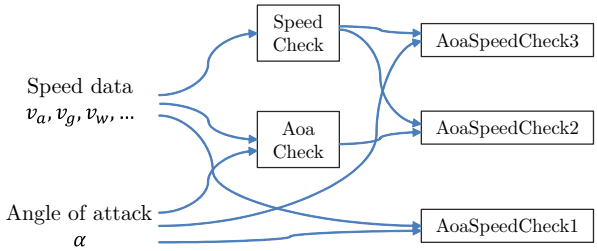

Fig. 5. Options for PILOTS programs to detect one or more combination of GPS, pitot tube, and angle of attack sensor failures.

Table III shows the detectable modes by the three AoaSpeedCheck programs. The meaning of modes 0-5 are the same as ground truth modes in Table II. Unlike AoaSpeedCheck1 and AoaSpeedCheck2, AoaSpeedCheck3 can detect modes 2 and 3 due to its sequential approach, which will be explained in Section IV-B3. Since none of the three programs can detect modes 6 and 7, they are not shown in Table III. The '?' mark used in modes 8-13 denotes unknown failure state: We do not know if the corresponding sensor has failed or not. Due to the reasons described in Section IV-A, in some cases PILOTS programs cannot determine the state of sensors, with only two redundancy models.

TABLE III
DETECTABLE MODES BY AoaSpeedCheck PROGRAMS (0: NOT FAILED, 1: FAILED, ?: UNKNOWN).

Mode	GPS	Pitot tube	AoA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
8	0	0	?
9	0	1	?
10	1	0	?
11	1	1	?
12	?	?	0
13	?	?	1

1) AoaSpeedCheck1*:* Fig. 6 shows the AoaSpeedCheck1 PILOTS program. Under the modes section, there are multiple conditions to detect error modes. Unlike the signature section we have used in SpeedCheck and AoaCheck, it behaves like a switch-case statement: Conditions are evaluated one by one from the top (*i.e.*, m0), and as soon as a condition is satisfied, its associated mode is chosen. When the mode is chosen, the integer part of mode identifier is set in the reserved mode variable (*e.g.*, if m9 is chosen, 9 is set in mode). If none of the conditions are satisfied, unknown (-1) is set in the mode variable. These conditions under the mode section are

a product of the conditions used in SpeedCheck with e_1 and AoaCheck with e_2 except for m9 and m11, where we do not need to check e_2 since a pitot tube failure is already estimated with e_1.

```
program AoaSpeedCheck1;
  inputs
    va, vg, vw, aa, ag, aw, aoa (t) using closest(t);
  constants
    V_CRUISE          =    110;
    SPEED_NORMAL_L    =   -0.1 * V_CRUISE;
    SPEED_NORMAL_H    =   0.33 * V_CRUISE;
    SPEED_PITOT_L     =   0.34 * V_CRUISE;
    SPEED_PITOT_H     =  16.10 * V_CRUISE;
    SPEED_GPS_L       = -13.83 * V_CRUISE;
    SPEED_GPS_H       =  -0.67 * V_CRUISE;
    SPEED_GPS_PITOT_L =  -0.66 * V_CRUISE;
    SPEED_GPS_PITOT_H =   -0.1 * V_CRUISE;
    AOA_NORMAL        =   0.10 * V_CRUISE;
  outputs
    va, vg, aoa, mode at every 1 sec;
  errors
    e1: vg - sqrt(va^2 + vw^2 +
                  2*va*vw*cos((PI/180)*(aw-aa)));
    e2: va -
        MPS2KNOT*sqrt(2*L/(2*PI*aoa + CL0)*S*RHO);
  modes
    m0: SPEED_NORMAL_L < e1 and e1 < SPEED_NORMAL_H
      and abs(e2) < AOA_NORMAL    "Normal";
    m1: SPEED_NORMAL_L < e1 and e1 < SPEED_NORMAL_H
      and AOA_NORMAL <= abs(e2)   "AoA sensor failure"
      estimate aoa = L/(PI*RHO*va^2*S) - CL0/(2*PI);
    m9: SPEED_PITOT_L < e1 and e1 < SPEED_PITOT_H
      "Pitot tube + (AoA sensor) failure"
      estimate va = sqrt(vg^2 + vw^2 -
        2*vg*vw*cos((PI/180)*(ag-aw)))
      estimate aoa = L/(PI*RHO*va^2*S) - CL0/(2*PI);
    m4: SPEED_GPS_L < e1 and e1 < SPEED_GPS_H
      and abs(e2) < AOA_NORMAL    "GPS failure"
      estimate vg = sqrt(va^2 + vw^2 +
        2*va*vw*cos((PI/180)*(aw-aa)));
    m5: SPEED_GPS_L < e1 and e1 < SPEED_GPS_H
      and AOA_NORMAL <= abs(e2)
      "GPS failure + AoA sensor failure"
      estimate vg = sqrt(va^2 + vw^2 +
        2*va*vw*cos((PI/180)*(aw-aa)))
      estimate aoa = L/(PI*RHO*va^2*S) - CL0/(2*PI);
   m11: SPEED_GPS_PITOT_L < e1
      and e1 < SPEED_GPS_PITOT_H
      "GPS + Pitot tube + (AoA sensor) failure";
end;
```

Fig. 6. AoaSpeedCheck1 PILOTS program.

The threshold values for speed data used in AoaSpeedCheck1 are different from the values used in SpeedCheck. This is due to the following reasons. Even if the value of error is out of range of defined error signatures, the error signature-based mode estimation method in Section II-B chooses the closest error signature and its associated mode. The final estimated mode depends on the τ parameter: We choose the most closest signature only if the likelihood of the second closest signature is greater than τ. Unlike the signature-based mode estimation, the mode estimation introduced in this section is purely based on boolean expressions. Thus, if we directly apply the threshold values for SpeedCheck defined in Table I to the boolean expressions in AoaSpeecCheck1, we are expected to end up with many unknown modes. To adjust the difference between the two mode estimation methods, we expand the threshold values defined for SpeedCheck. Fig. 7 shows the boundaries of two error signatures S_i and S_j. We consider a case where error e is between the two error signatures, and let b_i and b_j be the closer boundaries of the two error signatures to e. The distances from e to S_i and S_j are $\delta_i = e - b_i$ and $\delta_j = b_j - e$. When $\delta_i < \delta_j$, the mode estimation method chooses mode i if the following condition is met: $l_j = \frac{\delta_i}{\delta_j} = \frac{e-b_i}{b_j-e} < \tau$. Solving this inequality for e, we get

$$e < \frac{b_i + \tau b_j}{1+\tau}. \tag{14}$$

Since $b_i < b_j$, the RHS of (14) is greater than b_i, and thus τ effectively expands the mode estimation boundary of signature S_i. Similarly, when $\delta_i > \delta_j$, $\frac{b_j+\tau b_i}{1+\tau} < e$ gives us the expanded boundary for signature S_j. Applying these boundary expansion operations to the error signatures in Fig. 2 with $\tau = 0.95$, we obtain threshold values for AoaSpeedCheck1 as shown in Fig. 6.

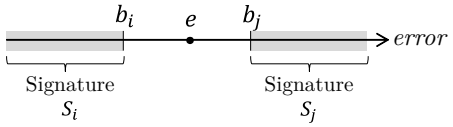

Fig. 7. Boundaries of two error signatures S_i and S_j.

2) AoaSpeedCheck2*:* Fig. 8 shows the AoaSpeedCheck2 PILOTS program. This program makes mode decisions based on the estimated modes received from SpeedCheck and AoaCheck. The received modes are referred to as SpeedCheck.mode and AoaCheck.mode for the SpeedCheck and AoaCheck programs, respectively. Compared to AoaSpeedCheck1, modes $8, 10, 12$, and 13 are newly created to handle an unknown mode (-1) sent from both child programs.

3) AoaSpeedCheck3*:* Fig. 9 shows the AoaSpeedCheck3 PILOTS program. This program takes an estimated mode in SpeedCheck.mode, airspeed in SpeedCheck.va, and ground speed in SpeedCheck.vg from the SpeedCheck program, and makes mode decisions with the value of e_2 computed from angle of attack data. This program is superior compared to AoaSpeedCheck1 and AoaSpeedCheck2 in terms of mode separation ability for modes 2 and 3. When a pitot tube failure is detected (*i.e.*, SpeedCheck.mode = 0), we can assume SpeedCheck already estimated a correct value in SpeedCheck.va. That allows us to compute e_2 with the correct value of airspeed, and thus we can separate modes 2 and 3. For the ground truth modes 6 and 7 when both GPS and pitot tube failure have occurred, SpeedCheck cannot estimate a correct airspeed value. Thus, we cannot separate these two modes and can only detect mode 11 same as AoaSpeedCheck1 and AoaSpeedCheck2.

V. Evaluation

We evaluate the AoaSpeedCheck programs that we have proposed in Section IV with test data streams produced with the X-Plane flight simulator [15]. We first describe the experimental settings, then compare the three redundancy models for angle of attack, and finally compare the three AoaSpeedCheck programs using the best performing angle of attack model.

```
program AoaSpeedCheck2;
  inputs
    SpeedCheck.va, SpeedCheck.vg, SpeedCheck.mode,
    AoaCheck.aoa, AoaCheck.mode (t) using closest(t);
  outputs
    SpeedCheck.va, SpeedCheck.vg, aoa, mode
      at every 1 sec;
  modes
    m0: SpeedCheck.mode == 0 and AoaCheck.mode == 0
      "Normal";
    m1: SpeedCheck.mode == 0 and AoaCheck.mode == 1
      "AoA sensor failure";
    m8: SpeedCheck.mode == 0 and AoaCheck.mode == -1
      "(AoA sensor) failure"
      estimate AoaCheck.aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
    m9: SpeedCheck.mode == 1
      "Pitot tube + (AoA sensor) failure"
      estimate AoaCheck.aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
    m4: SpeedCheck.mode == 2 and AoaCheck.mode == 0
      "GPS failure";
    m5: SpeedCheck.mode == 2 and AoaCheck.mode == 1
      "GPS + AoA sensor failure";
   m10: SpeedCheck.mode == 2 and AoaCheck.mode == -1
      "GPS + (AoA sensor) failure"
      estimate AoaCheck.aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
   m11: SpeedCheck.mode == 3
      "GPS + Pitot tube + (AoA sensor) failure";
   m12: SpeedCheck.mode == -1 and AoaCheck.mode == 0
      "(GPS) failure";
   m13: SpeedCheck.mode == -1 and AoaCheck.mode == 1
      "Unknown failure";
end;
```

Fig. 8. AoaSpeedCheck2 PILOTS program.

```
program AoaSpeedCheck3;
  inputs
    aoa,
    SpeedCheck.va, SpeedCheck.vg, SpeedCheck.mode (t)
      using closest(t);
  constants
    V_CRUISE   = 110;
    AOA_NORMAL = 0.10 * V_CRUISE;
  outputs
    SpeedCheck.va, SpeedCheck.vg, aoa, mode
      at every 1 sec;
  errors
    e2: SpeedCheck.va -
        MPS2KNOT*sqrt(2*L/(2*PI*aoa + CL0)*S*RHO);
  modes
    m0: SpeedCheck.mode == 0 and abs(e2) <  AOA_NORMAL
      "Normal";
    m1: SpeedCheck.mode == 0 and abs(e2) >= AOA_NORMAL
      "AoA sensor failure"
      estimate aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
    m2: SpeedCheck.mode == 1 and abs(e2) <  AOA_NORMAL
      "Pitot tube failure";
    m3: SpeedCheck.mode == 1 and abs(e2) >= AOA_NORMAL
      "Pitot tube + AoA sensor failure"
      estimate aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
    m4: SpeedCheck.mode == 2 and abs(e2) <  AOA_NORMAL
      "GPS failure";
    m5: SpeedCheck.mode == 2 and abs(e2) >= AOA_NORMAL
      "GPS + AoA sensor failure"
      estimate aoa
        = L/(PI*RHO*SpeedCheck.va^2*S) - CL0/(2*PI);
   m11: SpeedCheck.mode == 3
      "GPS + Pitot tube (+ AoA sensor) failure";
   m12: SpeedCheck.mode == -1 and abs(e2) <  AOA_NORMAL
      "(GPS) failure";
   m13: SpeedCheck.mode == -1 and abs(e2) >= AOA_NORMAL
      "Unknown failure";
end;
```

Fig. 9. AoaSpeedCheck3 PILOTS program with Model 1.

A. Experimental Settings

1) Test Data: Fig. 10 shows the test data streams. Test data streams were produced by a pilot using the X-Plane flight simulator with the Cessna 172 SP aircraft. We started capturing the data 160 seconds after the take-off and recorded ground speed, airspeed, and angle of attack for 240 seconds as drawn by solid lines in Fig. 10(a), Fig. 10(b), and Fig. 10(c), respectively.

To test the combinations of multiple sensor type failures, we artificially added data errors according to the ground truth modes that we defined in Fig. 10(d). These ground truth modes created sensor failures as shown in Table II. Data streams after failures are shown in dotted lines in Figs. 10(a)-(c). For GPS and pitot tube failures, we followed the assumptions that we made in Section III-A when we designed error signatures for speed data: For GPS failures, we dropped the ground speed to zero, whereas failed airspeed values kept 20% to 33% of their original values. For angle of attack sensor failures, we shifted the true angle of attack value $\alpha(t)$ by a parameter $\delta > 0$ as shown in Fig. 10(c).

2) Evaluation Metrics: We evaluate the mode estimation with the following two types of mode estimation accuracy metrics.

- *Complete-match mode estimation accuracy*: This metric quantifies how accurately the estimated modes estimate the ground truth modes. We call this metric *complete-match* since we do not reward modes unless their associated sensor failure states are completely matched. Let $\hat{\mu}(t)$ and $\mu(t)$ be the estimated and ground truth modes for time t, respectively, the complete-match mode estimation accuracy [%] is defined by:

$$\frac{100}{T}\sum_{t=1}^{T}\mathcal{I}\left[\hat{\mu}(t)=\mu(t)\right], \tag{15}$$

  where $\mathcal{I}[\cdot]$ is the indicator function that returns 1 if the argument is true and 0 otherwise and T is the total evaluation period in seconds (*i.e.*, $T = 240$).
- *Partial-match mode estimation accuracy*: As shown in Table III, AoaSpeedCheck programs are aware of sensor failure states through mode estimation. Unlike the complete-match mode estimation accuracy in Eq. (15) requires all the sensor failure states to be equal for each mode, this metric is designed to reward partial failure state matches. For example, when the estimated mode is 9 and the ground truth mode is 2, their sensor failure states are $[0, 1, ?]$ and $[0, 1, 0]$ as defined in Table III, respectively. We reward 2/3 points in this case since the sensor state estimation is correct for the first two. Let $s_i(\mu)$ be the i-th sensor failure state selector function for a mode μ, the partial-match mode accuracy [%] is defined

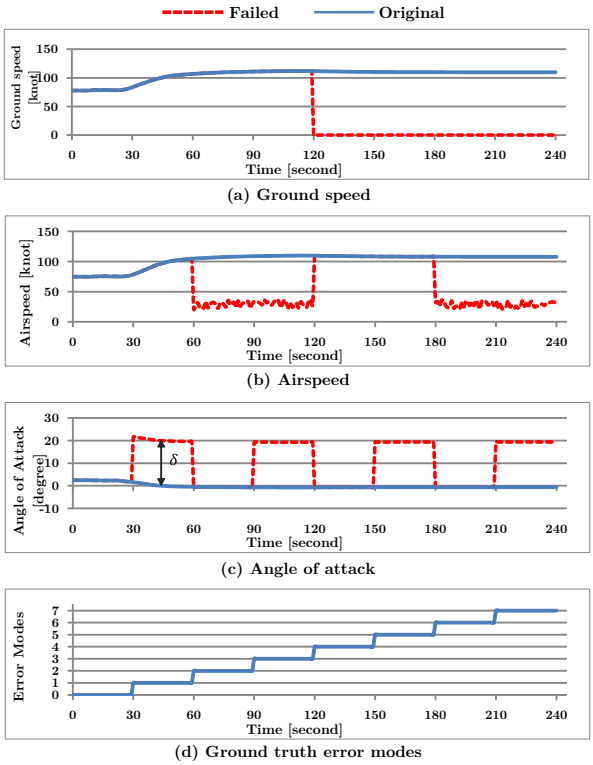

Fig. 10. Test data streams and ground truth error modes.

by:

$$\frac{100}{T}\sum_{t=1}^{T}\frac{1}{S}\sum_{i=1}^{S}\mathcal{I}\left[s_i(\hat{\mu}(t)) = s_i(\mu(t))\right], \qquad (16)$$

where S is the total number of sensor types (*i.e.*, $S = 3$ for GPS, pitot tube, and angle of attack).

B. Angle of Attack Failure Detection

We compare prediction accuracy of the AoaCheck program with the three redundancy models we developed in Eqs. (9)-(11). First, we trained Models 2 and 3 with a training data set that was recorded from a separate simulation flight with X-Plane. After training, we obtained $k_1 = 0.0694, k_2 = 0.3396$ for Model 2, and $k_1 = 2.900, k_2 = 0.0002, k_3 = 0.0011$ for Model 3. Fig. 11 shows a comparison of the three models after training and the monitored (training) data in dots. Fig. 11(a) shows estimated coefficient of lift using Models 1 and 2, and Fig. 11(b) shows estimated airspeed using Models 1-3. From Fig. 11(a), the linear model of Model 2 fits well with the monitored data; however, we see that the slope of Model 1 (0.1097) is larger than the slope of Model 2 (0.0694), and it makes Model 1 diverge from Model 2 as α increases. From Fig. 11(b), we notice that the difference in coefficient of lift between Model 1 and the other two models is magnified in the airspeed as the angle of attack approaches zero. Model 2 is slightly shifted upwards from Model 3, which closely follows the monitored data.

We evaluate AoaCheck with Models 1-3 with $\delta = \{5, 10, 20, 40\}$ to see the relationship between the model accuracy and the magnitude of error on angle of attack data. When we tested the AoaCheck program, we used the periods where there were no errors on airspeed in test data: 0-60s and 120-180s in Fig. 10. Fig. 12 shows the results of mode estimation accuracy for the three models. Model 3 is so accurate that it can pick up even a $\delta = 5$ degrees difference on angle of attack data with almost 100% accuracy, whereas Models 1 and 2 can barely detect error modes. Once δ gets to 20 degrees, Model 2 detects correct error modes almost 100% and Model 1 reaches 80% of mode estimation accuracy.

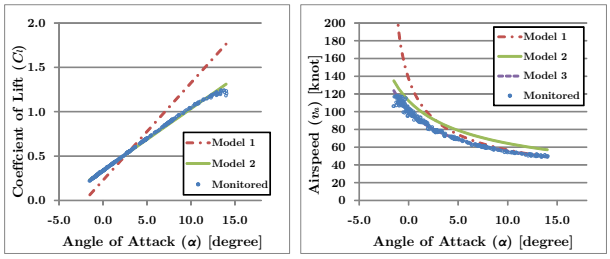

Fig. 11. Comparison of three airspeed models.

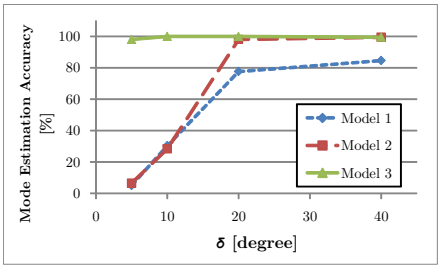

Fig. 12. Comparison of mode estimation accuracy for the AoaCheck PILOTS program with Models 1-3.

C. Multiple Sensor Type Failure Detection

Using Model 3 and $\delta = 20$ for the angle of attack test data, we compare mode estimation accuracy between the three AoaSpeedCheck programs. Figs. 13(a) and (b) show the mode estimation results produced by the SpeedCheck and AoaCheck programs, respectively. SpeedCheck correctly identified four modes (*i.e.*, 0: Normal, 1: Pitot tube failure, 2: GPS failure, and 3: Pitot tube and GPS failures). During the pitot tube failure periods, 60-120s and 180-240s, AoaCheck was not able to produce the correct modes due to its inability to recognize pitot tube failures. The zigzagging error modes during 90-120s and 210-240s were generated because the values of e_2 were around the border between normal and angle of attack sensor failure threshold values.

Figs. 14(a)-(c) show the mode estimation results produced by the AoaSpeedCheck1, AoaSpeedCheck2, and AoaSpeedCheck3, respectively. Fig. 15 shows the mode estimation results for the three AoaSpeedCheck programs. Both AoaSpeedCheck1 and AoaSpeedCheck2 managed to generate the same

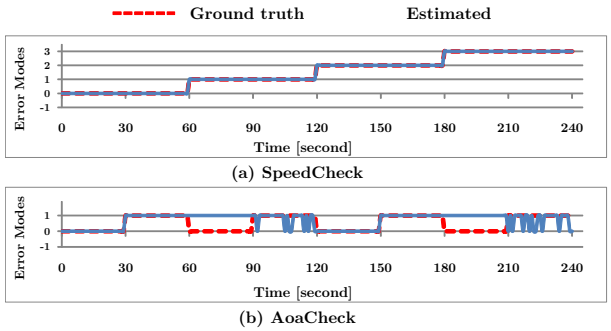

Fig. 13. Mode estimation results for SpeedCheck and AoaCheck programs.

mode estimation sequences, and we obtained the same mode estimation accuracy metrics: 50% for complete-match and 82% for partial-match. They both failed to correctly identify modes 2 and 3 during 60-120s and also modes 6 and 7 during 180-240s when the pitot tube sensor failure occured. On the other hand, AoaSpeedCheck3 correctly estimated modes 2 and 3 during 60-120s since it could identify the angle of attack sensor failure using corrected airspeed data provided by SpeedCheck. That led to a significant improvement on the mode estimation accuracy. For AoaSpeedCheck3, mode estimation accuracy metrics reached 75% for complete-match and 92% for partial-match. Since AoaSpeedCheck1 computes

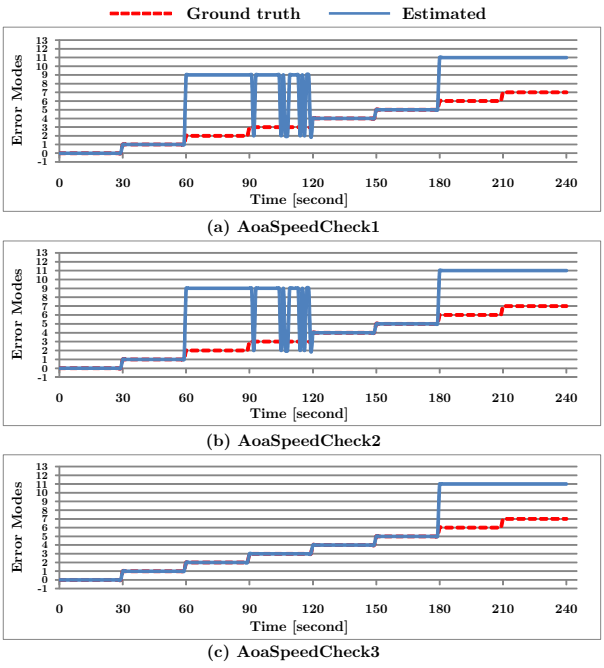

Fig. 14. Mode estimation results for AoaSpeedCheck programs.

e_2 internally, we have the same zigzag patterns observed in AoaCheck for 90-120s. For AoaSpeedCheck2, the same patterns were propagated for 90-120s through the estimated mode from AoaCheck.

Fig. 16 shows estimated data streams produced by AoaSpeedCheck3. Using the redundancy between data streams,

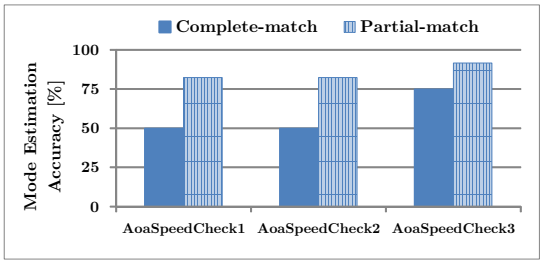

Fig. 15. Comparison of mode estimation accuracy for AoaSpeedCheck PILOTS programs.

AoaSpeedCheck3 was able to recover from sensor failures except for the period where both GPS and airspeed failed simultaneously (*i.e.*, 180-240s). Mean absolute error values between the original and estimated data streams up to 180s are 1.84 knots for ground speed, 2.53 knots for airspeed, and 0.21 degrees for angle of attack.

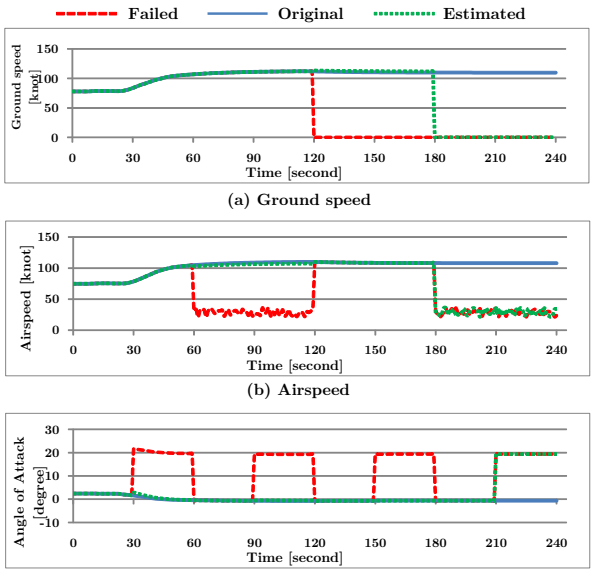

Fig. 16. Data streams estimated by AoaSpeedCheck3 with Model 3 ($\delta = 20$).

VI. Related Work

Fault detection, isolation, and reconfiguration (FDIR) has a long history of research in the control systems community [18]. 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 using analytical redundancy [5], [6] between sensor data and determine if a fault has occurred based on residuals, 2) identify the type of the fault, and 3) reconfigure the system accordingly. Commonly, the residual is a difference between a measured value and an estimated value (*i.e.*, error). PILOTS has a resemblance to FDIR systems: Error functions in PILOTS are equivalent to residuals in FDIR, error mode detection based on the likelihood vector in PILOTS is analogous to fault type identification in FDIR,

and data correction in PILOTS is a type of reconfiguration in FDIR. However, due to PILOTS' focus on domain-specific programming language approach, it allows users to isolate error conditions more generally through error signatures.

The angle of attack sensor is one of the important instruments to ensure flight safety and has been widely studied. Ossmann and Joos proposed a mechanism to detect and isolate erroneous angle of attack sensors using a combination of signal-based and model-based components [3]. While the signal-based component uses discrete Fourier transformation to detect unwanted oscillation in the signals, the model-based component uses a linear filter to generate a residual signal. Hardier et al. presented a method to estimate both angle of attack and airspeed simultaneously from aerodynamic coefficients [19]. They trained a neural network to learn a non-linear relationship between multiple variables and used it to estimate angle of attack and airspeed data. Both techniques are mathematically more complicated than the approach we take in this work; however, with the right level of abstraction, it is possible for PILOTS to accurately isolate sensor failures and estimate sensor values from redundant data streams for safer more robust flight systems.

VII. Conclusion and Future Work

In this work, we have presented enhancements to PILOTS to support multiple models of analytical redundancy and improved situational awareness for multiple simultaneous sensor type failures. In particular, we assume a situation where any combinations of three types of sensors, GPS, pitot tube, and angle of attack, can fail. Combining 1) an analytical redundancy model on airspeed, ground speed, and wind speed, and 2) another analytical redundancy model on airspeed and angle of attack, we estimated the true failure modes of sensors through PILOTS programs. To evaluate the mode estimation accuracy, we produced test data streams using the X-Plane flight simulator and simulated multiple sensor type failure scenarios. The simulation results show that multiple models of analytical redundancy enable us to detect failure modes that are not detectable just by using a single model.

As the number of sensors in aircraft increases, we need to discover failure models and also execute mode estimation process in a scalable manner. It is also important to pursuit high-level abstractions that will enable data scientists to more easily develop fault-tolerant applications for a massive number of sensor data streams. Currently no semantics is defined for the PILOTS programming language. Defining denotational semantics for PILOTS is a future research direction toward formally verifiable flight safety. Finally, uncertainty quantification [10] is another important future direction to associate confidence to mode estimation in support of decision making.

Acknowledgment

This research is partially supported by the Air Force Office of Scientific Research, Grant No. FA9550-19-1-0054.

References

[1] Indonesian National Transportation Safety Committee (NTSC), "Preliminary Aircraft Accident Investigation Report: Lion Air Flight 610," http://knkt.dephub.go.id/knkt/ntsc_aviation/baru/pre/2018/2018%20-%20035%20-%20PK-LQP%20Preliminary%20Report.pdf, 2018.

[2] Boeing, "Boeing Statement on Lion Air Flight 610 Preliminary Report," https://boeing.mediaroom.com/news-releases-statements?item=130336, 2018.

[3] D. Ossmann, "Enhanced detection and isolation of angle of attack sensor faults," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 1135.

[4] Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile, "Final Report: On the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro - Paris," https://www.bea.aero/docspa/2009/f-cp090601.en/pdf/f-cp090601.en.pdf, 2012.

[5] E. Chow and A. Willsky, "Analytical redundancy and the design of robust failure detection systems," *IEEE Transactions on automatic control*, vol. 29, no. 7, pp. 603–614, 1984.

[6] M.-O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès, "Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 5, pp. 2163–2177, 2004.

[7] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela, "Airplane flight safety using error-tolerant data stream processing," *IEEE Aerospace and Electronics Systems Magazine*, vol. 32, no. 4, pp. 4–17, 2017.

[8] S. Imai, S. Chen, W. Zhu, and C. A. Varela, "Dynamic data-driven learning for self-healing avionics," *Cluster Computing*, Nov 2017.

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

[10] 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*, June 2014, pp. 1182–1192.

[11] E. P. Blasch, D. A. Lambert, P. Valin, M. M. Kokar, J. Llinas, S. Das, C. Chong, and E. Shahbazian, "High level information fusion (hlif): survey of models, issues, and grand challenges," *IEEE Aerospace and Electronic Systems Magazine*, vol. 27, no. 9, pp. 4–20, 2012.

[12] J. T. Oden, E. E. Prudencio, and P. T. Bauman, "Virtual model validation of complex multiscale systems: Applications to nonlinear elastostatics," *Computer Methods in Applied Mechanics and Engineering*, vol. 266, pp. 162–184, 2013.

[13] 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)*, December 2013.

[14] S. Imai, A. Galli, and C. A. Varela, "Dynamic data-driven avionics systems: Inferring failure modes from data streams," in *Dynamic Data-Driven Application Systems (DDDAS 2015)*, June 2015.

[15] Laminar Research, "X-Plane 11 - The world's most advanced fligtht simulator." https://www.x-plane.com/, 2019.

[16] Federal Aviation Administration, "Automatic Dependent Surveillance-Broadcast (ADS-B)," https://www.faa.gov/nextgen/programs/adsb/, 2019.

[17] I. H. Abbott and A. E. Von Doenhoff, *Theory of Wing Sections: Including a Summary of Airfoil Data*. Courier Corporation, 1959.

[18] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *IEEE transactions on control systems technology*, vol. 18, no. 3, pp. 636–653, 2009.

[19] G. Hardier, C. Seren, P. Ezerzere, and G. Puyou, "Aerodynamic model inversion for virtual sensing of longitudinal flight parameters," in *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*. IEEE, 2013, pp. 140–145.