Model-based System Health Management and Contingency Planning for Autonomous UAS

Johann Schumann*
SGT, NASA Ames Research Center, Moffett Field, CA 94035, USA

Nagabhushan Mahadevan†
Vanderbilt Univ., Nashville, TN 37212, USA

Adam Sweet‡
NASA Ames Research Center, Moffett Field, CA 94035, USA

Anupa Bajwa§ and Michael Lowry¶
NASA Ames Research Center, Moffett Field, CA 94035, USA

Gabor Karsai‖
Vanderbilt Univ., Nashville, TN 37212, USA

Safe autonomous operations of an Unmanned Aerial System (UAS) requires that the UAS can react to unforeseen circumstances, for example, after a failure has occurred. In this paper, we describe a model-based run-time architecture for autonomous on-board diagnosis, system health management, and contingency management. This architecture is being instantiated on top of NASA's Core Flight System (cFS/cFE) as a major component of the on-board Autonomous Operating System (AOS). We will describe our diagnosis and monitoring components, which continuously provide system health status.

Automated reasoning with constraint satisfaction form the core of our decision-making component, which assesses the current situation, aids in failure disambiguation, and constructs a contingency plan to mitigate the failure(s) and allow for a safe end of the mission. We will illustrate our contingency management system with two case studies, one for a fixed-wing aircraft in simulation, and one for an autonomous DJI S1000+ octo-copter.

I. Nomenclature

AOS	=	Autonomous Operating System
ATC	=	Air Traffic Control
cFS/cFE	=	NASA Core Flight System/Executive
DM	=	Decision Maker
DR	=	Diagnostic Reasoner
FML	=	Fault Modeling Language
LC	=	Limit checker
UAS	=	Unmanned Aerial System
R2U2	=	Realizable, Responsive, Unobtrusive Unit
R5 Edge	=	NASA Langley Edge T240 Model

*SGT Chief Scientist Computational Science, MS 269/3
†Systems Architect, Institute for Software Integrated Systems, Vanderbilt University, 1025 16th Ave S, Nashville, TN 37212
‡Computer Engineer, MS 269/2
§Computer Engineer, M/S 269-3
¶Senior Scientist for Reliable Software Engineering, MS 269/2
‖Professor, Vanderbilt University, 1025 16th Ave S, Nashville, TN 37212

II. Introduction

SAFE autonomous operations of an Unmanned Aerial System (UAS) requires that the UAS is, during flight, aware of its health status and environment, and can react accordingly. On a manned aircraft, the pilot is in charge of detecting and identifying faults as well as trying to find their root causes. Only with a detailed knowledge about the current state of the aircraft, the pilot can plan any contingency activity that can mitigate the failures or provide means to safely end the flight by, for example, landing at a nearby airport. Fault identification and planning for a contingency requires that the pilot has substantial knowledge and experience, and that he is able to plan ahead and to mentally play through several alternatives to address the current situation.

An autonomous UAS must essentially perform the same actions in real time, without the help of a ground station or a human in the loop. Modern UASs have numerous sensors on-board, which can be used to monitor the current system state and to diagnose faults. The determination of the root cause can require substantial reasoning but still might yield ambiguous results. For example, an inconsistent altimeter reading might be caused by a failure in the barometric altimeter, the GPS system, or the LIDAR system. An easy test to find out, which of the sensors cause the problem is to perform a short climb and monitor the individual sensor readings during that climb. Such an active diagnosis procedure must be planned and executed by the UAS in order to try to disambiguate the failure signature.

Then, proper contingency planning can take place. Based upon knowledge of the failure(s) and the resulting changed capabilities of the UAS, the on-board software must device a plan that leads to a safe conclusion of the mission. Actions, which can be taken include, for example, shorten the mission, fly to a nearby emergency airport, or land at the nearest safe location. This planning also requires prediction of the aircraft state.

In this paper, we describe a model-based architecture for autonomous on-board system health management and contingency management. This model-based architecture has been instantiated as a major component of AOS (Autonomous Operating System [1]), which is based upon NASA's open source Core Flight System (cFS/cFE) [2]. We will describe the overall architecture, the diagnostics and monitoring components and then focus on the "Decision Maker", which performs contingency planning. Our components are model-based and we will discuss the underlying structure of the relevant models.

We will illustrate the on-board health management and contingency planning capabilities with the help of two case studies: actuator failure for a fixed-wing aircraft in simulation, and a LIDAR-failure scenario, which was test-flown on an autonomous DJI S1000+ octo-copter.

The rest of this paper is structured as follows: in Section III we will give a brief description of the NASA Core Flight System software, which provides the infrastructure for Autonomous Operating System (AOS). Section IV discusses our architecture for model-based health management and contingency planning, which are core capabilities of AOS. Section V describes the on-board diagnosis (DR) and health management (R2U2) systems; Section VI our model capturing and generation framework. Section VII focuses on contingency planning. We illustrate our system with two case studies in Section VIII). Section IX discusses related work and Section X finally concludes.

III. Background

A. NASA Core Flight System

The NASA core Flight System (cFS) [2] is a platform and framework that has been developed at the Goddard Space Flight Center. Its component-based design, layered software and dynamic runtime environment has been used in a number of successful space missions. This Open-source software* has been certified for man-rated space applications. The cFS architecture simplifies the flight software development process by providing the underlying infrastructure and hosting a runtime environment for mission-specific applications, so-called "lollipops". cFS can be executed on, among other target platforms, Linux or VxWorks.†

B. The Autonomy Operating System (AOS)

The Autonomy Operating system (AOS) is a software system that enables core capabilities for the autonomous operations for an unmanned aircraft [1]. It is based on the NASA cFS system and features applications or "apps" for the execution of flight plans, natural-language communication with Air traffic Control, Diagnostics, Prognostics, and contingency planning.

*https://cfs.gsfc.nasa.gov/

†https://www.windriver.com/products/vxworks/

The AOS architecture shown in Figure 1 illustrates the structure of the system. The underlying cFS system provides a "software bus," a publish-subscribe architecture, which is used by numerous applications ("apps" or "lollipops") to communicate with each other. Apps can be activated on regular schedule or can be event driven. Numerous apps are provided by the cFS software (shown as circles in Figure 1). A specific app, the "MAV-bridge" is in charge of communicating with the low level flight control software to obtain sensor and aircraft status information. In our case, we use a slightly modified version of the open-source ArduCopter software,[‡] which is running on a Pixhawk hardware.[§] It is in charge of keeping the UAS controlled in the air, executing simple sequences of way-points, and reading and processing numerous sensor values for navigation and aircraft status. The Pixhawk, mainly designed for RC operation of model aircraft directly interfaces with sensors and controls the motors of the aircraft.

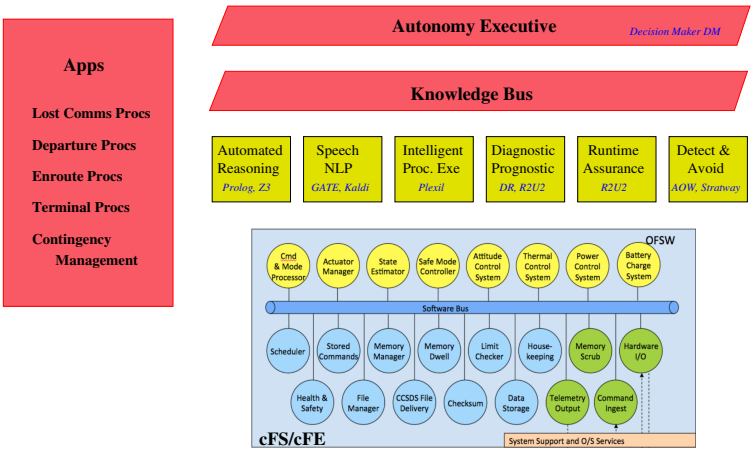


Fig. 1 High-level architecture of AOS based on cFS/cFE. cFS architecture diagram from [3]

The AOS applications (shown as yellow boxes) provide specific capabilities and infrastructure that enable autonomous UAS operations: Automated Reasoning, based upon an underlying Prolog or Z3 reasoner, is used for navigation, contingency planning, and interaction with Air Traffic Control (ATC). Spoken ATC commands are processed by the Natural Language Processing (NLP) unit [4]. Plexil is an event-driven planner [5] that has been customized to execute flight plans for nominal and off-nominal operations, as well as procedures, which require ATC interaction.

The applications for diagnosis, prognostics, and runtime assurance are based on the Diagnostic Reasoner (DR) and the R2U2 components, which will be described in detail in the following sections. The AOS apps exchange information using the Knowledge Bus. It is also used by the Decision Maker "DM" (Section VII), which, as a part of the Autonomy Executive, provides contingency planning capabilities. With this infrastructure, apps and flight procedures for lost communication situations, for departure, en-route, and approaches, as well as for contingency management can be realized.

IV. Model-based Architecture for Diagnosis and Contingency Planning

AOS is separating the capability of health and contingency management into two modules: diagnosis and decision-making. In the case of an adverse event or failure, diagnosis is performed first, to determine what failure has occurred. Decision-making is done next, to improve the resolution of the fault diagnosis and to determine the impact of the failure followed by what actions might need to be done to recover from the failure and fly the aircraft to safety. The selected actions are then executed.

Figure 2 shows the flow of information. Sensor and status data from the UAS are transmitted by the ArduCopter flight software into the AOS system. These data are used to perform diagnosis and to monitor the UAS system. The model-based diagnosis apps DR and LC (Section V.A) perform perform fault detection and isolation; R2U2 dynamically monitors sensors and software and performs prognostics (Section!V.B). The current health status of the UAS, which is updated at 1Hz is then handed over to the DM (Decision Maker, Section VII) component. Based upon the system health

[‡] `http://ardupilot.org/copter/`

[§] `pixhawk.org`

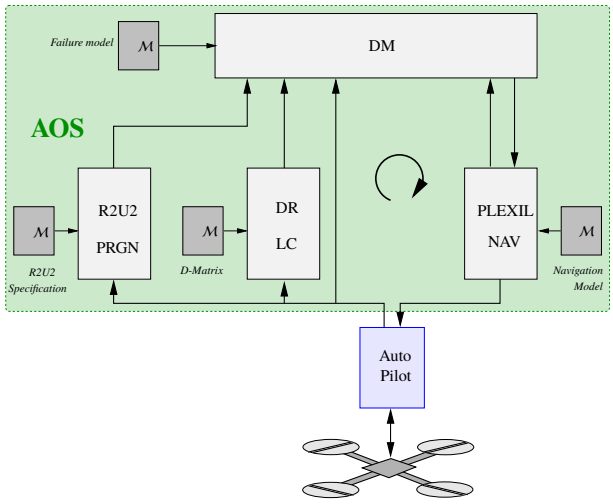


Fig. 2 High-level architecture of our model-based Health management and contingency planning system

status and the current flight plan, the DM performs logic-based search to find (a) active diagnostic procedures to improve the diagnostic resolution (if necessary) and (b) flight plans which can be safely executed under the current circumstances. If necessary, such contingency plans might contain emergency actions like, for example, cutting short the flight, diversion to a nearby airport for emergency landing, or an immediate ditch by activating an on-board parachute. Finally, if necessary, the generated active diagnosis or contingency plan is sent to the plan execution module PLEXIL/NAV, which is in charge of commanding or navigating the UAS and interacting with Air Traffic Control. This component emits the sequence of commands or waypoints that the UAS will follow.

All components of the health management and contingency planning system are model-based. The individual components access the necessary models (shown in Figure 2 as gray boxes marked $\mathcal{M}$), which are diagnostics and prognostics models of the UAS and its subcomponents, failure propagation and fault impact models and models about improving diagnostic resolution, navigation, flight procedures, and interaction with ATC.

V. Monitoring and Diagnosis

A. Diagnostic Reasoner (DR)

The Diagnostic Reasoner (DR) is a cFS component (“app”) developed for the AOS system. It monitors and diagnoses the vehicle on which AOS is running. Based on the sensor information from the vehicle, DR performs fault detection, and if an anomaly is found, it will perform fault isolation to identify the failure mode. If it is not possible to isolate the fault to a single failure mode, the diagnosis result is an ambiguity group of several potential failure modes. The diagnosis is published over the cFS software bus, where it can be used by the other components of the health management system. The architecture and information flow of DR is shown in Figure 3. In this diagram, the Limit Checker (LC) is a standard cFS application, which receives (analog) sensor data and continuously checks that data values do not exceed predetermined thresholds, called watch points. DR uses LC to process the UAS sensor and status data, and reads the LC watchpoint results. It then uses the Test Result Calculator to make the simple conversion from watchpoint results into the diagnostic test results that DR uses.

For diagnosis, DR uses a dependency matrix (D-matrix) approach [6]. This efficient model-based approach for diagnosis allows DR to determine the state of a system component as “GOOD”,“BAD”,“SUSPECT”, or “UNKNOWN” based upon a number of discrete test results. The D-matrix relates the diagnostic tests for a system to the failure modes of that system. It is represented as a matrix with the diagnostic tests along the top row and failure modes along the left column. If a particular test can detect a particular failure mode, the corresponding entry inside the D-matrix is 1 or ’true’; if not the entry is 0 or ’false’. The D-matrix approach was chosen to relate to work being done under other projects related to verifying D-matrices.

The D-matrix solving algorithm implemented in DR takes the test results as input, and returns which failure modes

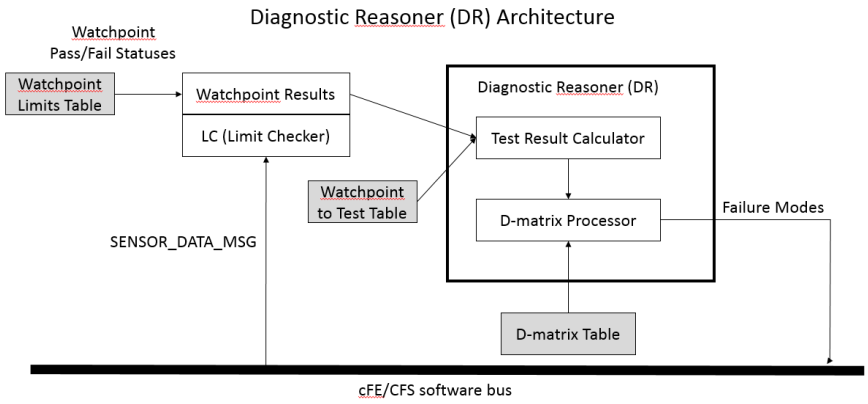


Fig. 3 Architecture of DR

may have occurred. Listing 1 shows the individual steps of the algorithm, which are illustrated in Figure 4.

Algorithm 1 DR diagnosis algorithm

```
Initialize the array of failure modes to "UNKNOWN".
for all tests in array of tests do
    check, which failure modes are implicated by that test (aka, the row has a '1' in the D-matrix).
    for all implicated failure modes do
        determine the new failure mode value based on value of the test result.
        if Testresult = "PASS" then
            set that failure mode to "GOOD", regardless of the current failure mode value.
        else if Testresult = "FAIL" then
            if failure mode has not already been marked "GOOD" (by an earlier test) then
                set that failure mode to "SUSPECT".
            end if
        end if
    end for
end for
for all failure mode FM in array of failure modes do
    if value of failure mode = "SUSPECT" then                ▷ check if it may be marked as "BAD"
        if ∃ "FAIL"-ed test T that implicates FM, and all other FMs associated with T are marked "GOOD" then
            mark this failure mode "BAD"
        end if
        if the "FAIL"-ed test only implicates this failure mode then
            this is a trivial case of the preceding rule and the failure mode may be marked "BAD".
        end if
    end if
end for
```

B. Online Monitoring System (R2U2)

R2U2 (Realizable, Responsive, Unobtrusive Unit) [7] is a framework and tool for the continuous monitoring of safety-critical and embedded cyber-physical systems. R2U2 combines past-time and future-time Metric Temporal Logic, probabilistic reasoning with Bayesian networks, and model-based prognostics.

Like the other components of AOS, R2U2 is implemented as a cFS app and activated at a regular rate of up to 10Hz. Figure 5 shows its architecture. R2U2 subscribes to numerous messages of the software bus that carry sensor data, current aircraft status, and the future flight plan. These messages originate from the on-board autopilot, the Plexil

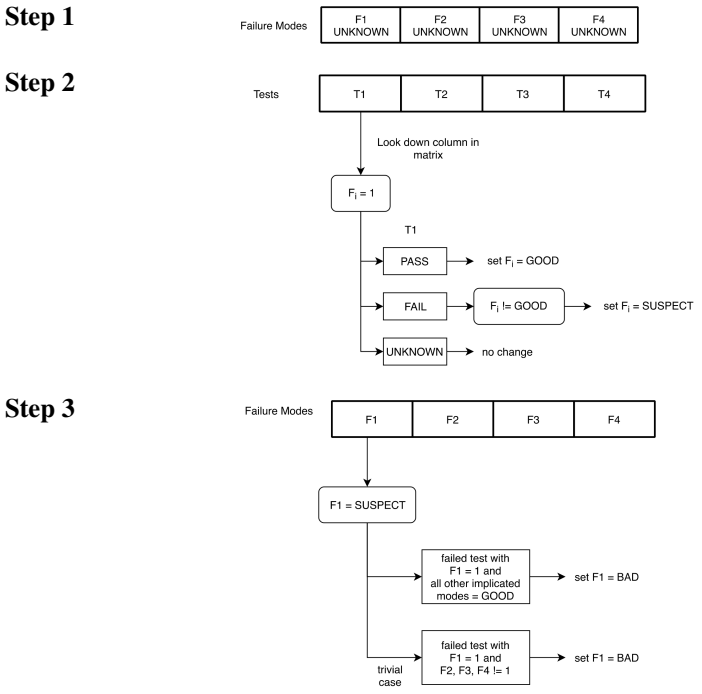


Fig. 4 Individual steps of the DR diagnostic algorithm

.

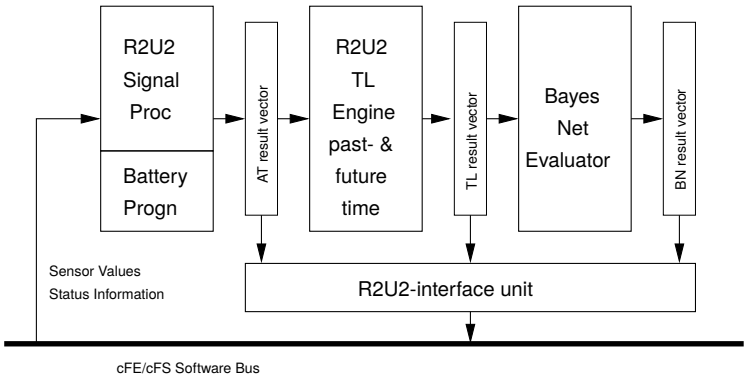


Fig. 5 The R2U2 monitoring system

planner, as well as from other AOS apps. Using a set of customizable filters, operators, and discretizers, R2U2 produces Boolean values, which are then processed by the R2U2 temporal engine [8, 9].

Typical health properties concern the battery. For example, if we encounter a higher than usual battery current for more than 5 consecutive seconds, we expect that the aircraft is climbing. This is reflected by a positive vertical velocity v_z that persists for at least 3 seconds:

$$\square_{[10s]}(I_{batt} > 65A) \rightarrow \Diamond_{[5s]}\square_{[3s]}(v_z > 3\text{ft/s})$$

Similarly, after a change of the target heading of the UAS, we expect that the UAS heading is aligned with the new heading within 5 seconds. Short glitches should be ignored. This property only needs to hold while the UAS is in the auto mode, following a flight-plan:

$$mode\_auto \rightarrow \Diamond_{[2s]} hdg\_achieved \vee \Diamond_{[3s]} nav\_bearing\_changed$$

More properties and examples are discussed in [10].

R2U2 can perform real-time Bayesian reasoning [8], using the results of temporal formals as inputs to support root cause analysis by, for example, estimating the likelihood of a specific sensor failure. A Kalman-filter based battery prognostics module [9] provides up-to-date estimates of state-of-charge of the battery and remaining useful life. R2U2 publishes its monitoring and reasoning results on the cFS software bus.

VI. System Modeling and D-matrix Generation

AOS leverages an open source meta-programmable, domain-specific, and collaborative modeling platform, called WebGME [11], to create a domain specific modeling language, the Fault Modeling Language (FML). Our FML captures (a) the system architecture of the aircraft, (b) a functional decomposition model, (c) a fault propagation model, models for (d) fault impact, (e) diagnosis refinement procedures and, (f) fault masking and/ or recovery plans.

FML uses the SysML [12] style Block Diagram models to capture the components (blocks), their interfaces (ports) and their interaction (wiring between the ports). The SysML internal block diagram models are extended to capture the fault model within the component and its propagation across the system using the concepts of Timed Failure Propagation Graphs (TFPG). For the definition and use of TFPG for diagnosis see [13, 14]. Table 1 lists the different kinds of models and their description.

Table 1 FML Models

Model	Description
System Model	Captures the architecture model as SysML block diagram. Components/ subsystems are modeled using blocks. Component interfaces are represented by ports. Wiring between the ports captures the interaction between the components.
Functional Decomposition Models	Breakdown of desired system function from high level functions to lower level function(s). Lowest level functions are mapped to components implementing them.
Fault Propagation Models	Modeled as an extension to the SysML internal block diagram model. The fault model captures the fault sources (Faults), deviations from nominal or expected behavior (Anomalies), observable anomalies (Tests), functionality degradation (Effects). The edges represent cause-effect relationship and the fault propagation. Labeled edges from and to the ports capture fault propagation across component boundaries.
Fault Impact Models	Captures impact of fault or fault ambiguity groups on system functions as changes to safe/ permissible operating conditions in terms of modifications to system variable ranges and system mode(s).
Refinement Models	Captures the fault ambiguity groups that cannot be resolved based on the existing Tests. It prescribes active diagnosis procedures that can be executed to trigger additional tests which may be used to prune the fault ambiguity sets. It also captures the valid operating conditions and modes in which each active diagnosis procedures may be executed.
Recovery Models	Captures one or more recovery procedures that can be executed so that the system can recover from certain fault(s) or arrest propagation effects. The model captures the mode changes and changes to operating range of system variables when the recovery procedure is executed.

FML allows cross-links or cross-references across the different model types to provide context to each model. For instance, the lowest level functions in the functional decomposition model are related to blocks/ components in the system model, which implement the associated function(s). Likewise, the fault propagation model references functions in the "Effect" nodes to capture functional degradation due to the presence and propagation of one or more faults. The diagnostic refinement model captures the fault ambiguity set and the corresponding active diagnosis procedures (and the impacted tests) that could help in isolating the fault source. The fault ambiguity set references the faults in the system model that could not be disambiguated based on the operational tests. Each active diagnosis procedure cross-references the tests in the system model that would be activated by the procedure. Table 2 provides a list of cross references that can be used to integrate the models in FML.

Figure 6 shows the system model for the fixed wing aircraft R5 Edge, which will be discussed in detail in Section VIII.A. Here, we show the underlying fault propagation model of one of the components (the Left Aileron

Table 2 FML Model Cross Links

Model	Context	Description
Effect	Fault Propagation Model	Functions cross referenced in fault propagation model. This is used to capture the functional degradation due to the presence of one or more fault(s).
Implementation	Functional Decomposition Model	Reference to blocks in the system model that implement one or more function(s) in the functional decomposition model.
Ambiguity Set	Fault Impact, Refinement Model	Represents a set of faults which cannot be distinguished based on the triggering Tests.
Operational Impact	Fault Impact, Recovery Plan	Reference to system variables and changes in their operating range.
Requirement	Active Diagnosis Procedures, Recovery Plan	Conditions on system variables in order to execute active diagnosis procedures or recovery plans.
Mode Requirement	Active Diagnosis Procedures, Recovery plan	Conditions on system modes in order to execute active diagnosis procedures or recovery plans.
TestRefs	Active Diagnosis Procedures	Additional tests that can be evaluated when an active diagnosis procedure is executed.
Trigger Condition	Recovery Plan	A set of faults related to triggering a recovery plan. Any of the faults in the triggering condition may be handled using the recovery plan.
Mode Change	Recovery Plan	Mode change introduced by executing a recovery plan.

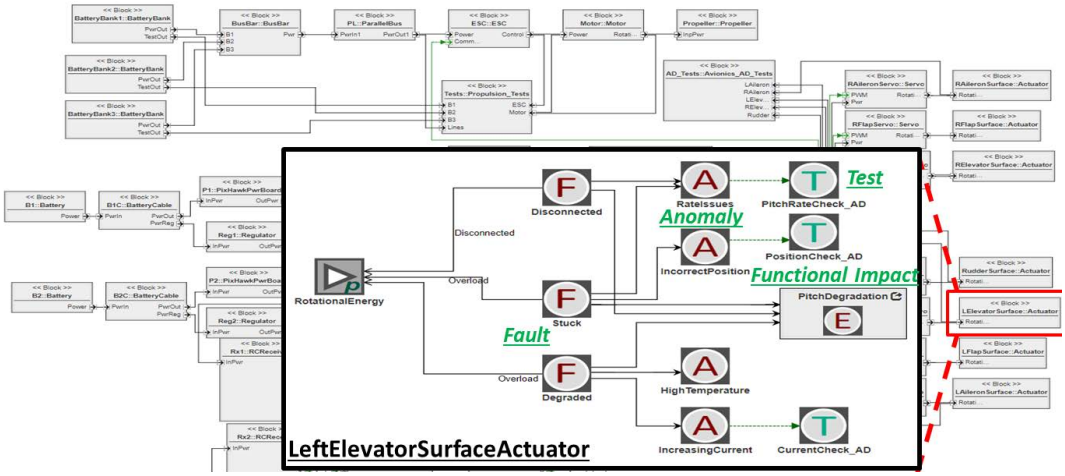


Fig. 6 R5: System model with fault propagation

Surface Actuator). It shows the anomalies resulting from different kinds of faults (stuck, degraded, and disconnected) and their impact on system functions (pitch degradation). Anomalies are connected to Tests that can be observed at runtime. Figure 7 shows the breakdown of the high-level functions into lower level functions and their association to implementing components in the system model.

The Fault Impact Model in Figure 8 captures the impact of a set of faults (ambiguity group) on desired system functions and the underlying operational variables, such as limits on the climb angle or drop in efficiency. Figure 8 lays out prescriptive active diagnosis procedures for different classes of fault ambiguity sets. While the model does not capture the procedural code for executing the active diagnosis procedure, it captures the operation constraints under

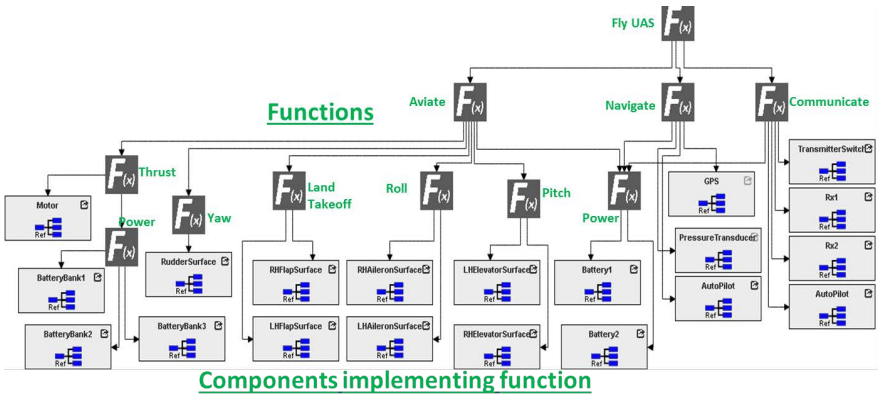


Fig. 7 R5: Functional Decomposition Model

which the procedures may be executed such as system mode, minimum altitude, ground speed, etc.

Recovery models (as shown in Figure 8) map the diagnosed faults (triggering fault) to the recovery plans that may be executed. Further, they list the requirement to execute the recovery plan in terms of mode and system operational variables. They also capture the resulting system mode once the recovery procedure is completed and any resulting operational constraints due to the recovery procedure. Figure 8 indicates that a "Disconnect" procedure can be used to deactivate an aileron surface that experiences a stuck or degraded fault and thereby conserve power as well as get a handle on the temperature rise. However, this could mean that the turn angle is limited for the rest of the flight. Emergency procedures (Figure 8) model the list of faults that necessitate such an action.

FML allows the users to compile the models into artifacts that can be used in the deployed system (see blocks marked $\mathcal{M}$ in Figure 2). FML generates a D-matrix based on the fault propagation model for each operational mode of the system. As described in Section V.A, a D-matrix is a matrix relating tests to failure modes. The D-matrix generated from this system is shown below (Figure 9a). Further at runtime, certain tests might be set to inactive or unknown state to further prune the fault impact matrix. FML generates AOS-compliant C/C++ code to be used by the DR module (Section V.A). It further generates CSV files and other artifacts to support debugging the model.

Apart from the D-matrix, the FML generates Prolog code (Figure 9b) to set up the interface between DR and DM. It generates Prolog rules that aid DM understand the DR output (the diagnostic hypothesis for the current set of faults). Furthermore, FML generates Prolog code to capture model information that allows DM to set up the functions and variables impacted, the possible active diagnosis procedures that could be executed to refine the diagnosis output, possible local recovery actions to arrest or mitigate a fault and resulting impact on the system performance. DM uses these at every stage to plan the next course of action, an active diagnosis procedure, a recovery procedure, or updates to the mission.

VII. Logic-based Decision-making for Contingency Management

On a high level, the contingency management system of AOS, the Decision Maker (DM) has to solve the following problem: can the UAS continue to execute its mission, given the current state of the aircraft and current failures? If a safe execution of the mission should not be possible, the DM has to find a suitable contingency plan, which can comprise an alternative ending of the mission, like landing at an emergency airport, pulling the parachute, or another emergency action. The DM will always attempt to device a contingency plan that (a) obeys all safety requirements, (b) can be executed safely, given the current and predicted state and capabilities of the UAS, and (c) is least intrusive with respect to the original mission.

For its operation, the DM relies on information about the current state of the UAS, its health, as well as the specifics of the mission. As shown in Figure 2 this information is provided by the diagnosis and monitoring systems DR and R2U2, as well as the PLEXIL planning system. Reasoning is performed with the help of the aircraft failure and performance model as presented in Section VI. Additional static information about the aircraft and the mission is obtained from an on-board database.

In essence, the input of the DM is comprised of the current flight plan (i.e., mission) as list of way-points $G = \langle w_1, \ldots, w_n \rangle$, where each waypoint is defined as a quadruple $w_i = [lat_i, lon_i, alt_i, spd_i]$ with latitude lat_i, longitude

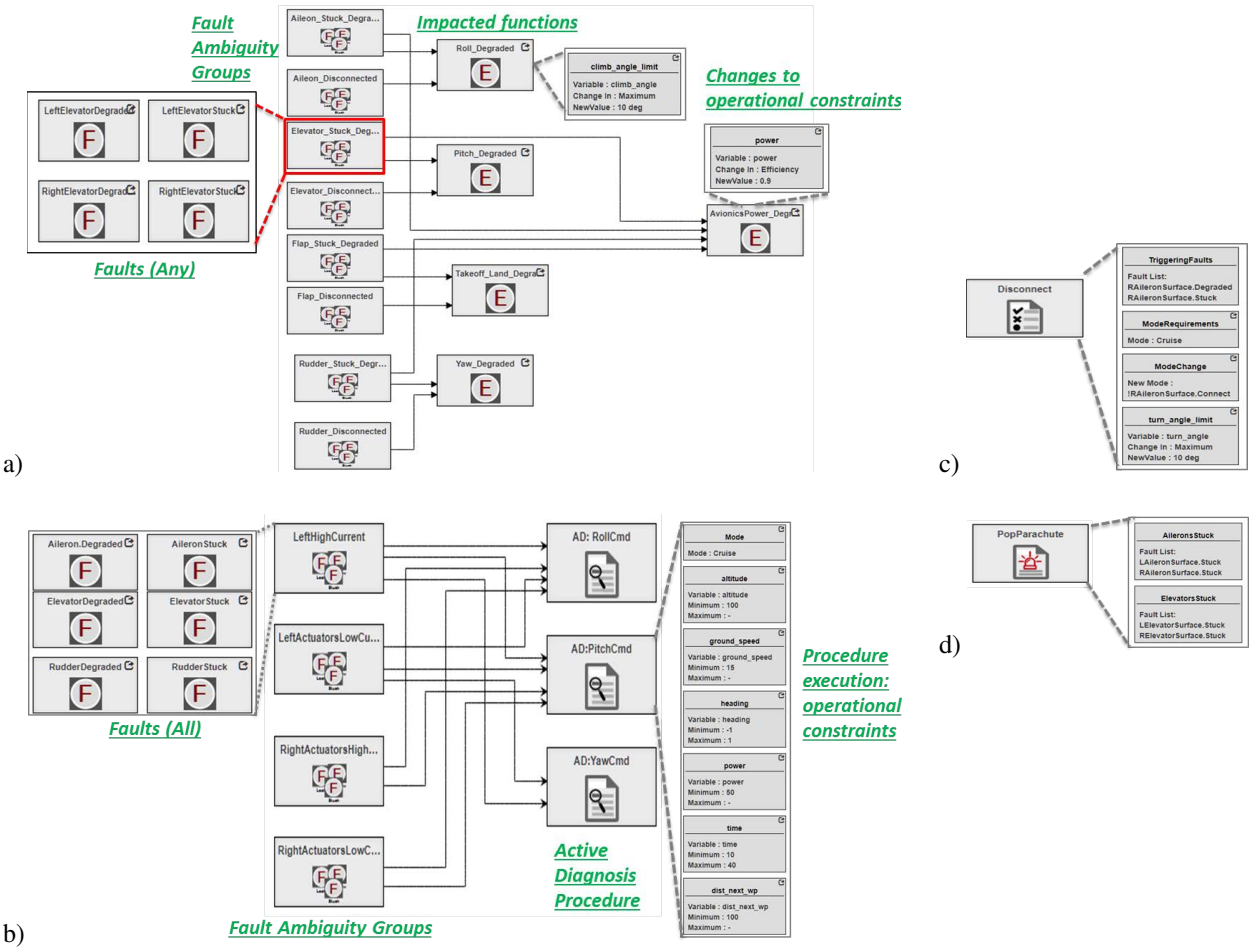


Fig. 8 a) R5: Fault Impact Model. b) R5: Diagnostic Refinement Model. c) R5: Recovery Procedure. d) R5: Emergency Procedure

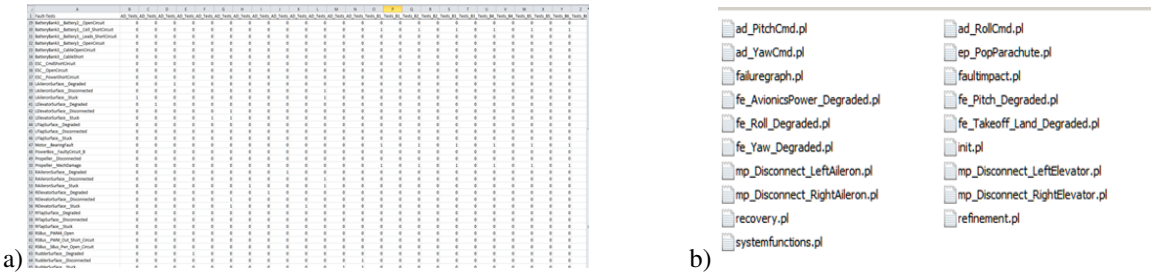


Fig. 9 a) FML: Output DMatrix. b) FML: Output Prolog files

lon_i, altitude alt_i, and target speed spd_i. In addition the current aircraft state $X = [lat, lon, alt, spd, U\_batt]$ and the current failure modes F are provided.

Then, the DM (Decision Maker) performs two tasks: failure disambiguation and contingency planning. The DM app is implemented in SWI Prolog and scheduled by the cFS to be executed every 2 seconds in our system configuration. DM has been implemented in SWI Prolog[¶] which provides powerful automated reasoning, backtracking search and constraint-logic programming.

[¶] `swiprolog.org`

Whenever the failure modes F contain a failure mode ambiguity group A, there is not enough initial diagnostic information to isolate the failure to a single component or subsystem. For example, an "inconsistent altitude reading" corresponds to an ambiguity group with three elements, because the barometric altimeter, the GPS, or the on-board LIDAR system might be the cause of the inconsistency. In many cases, a proper contingency management requires that the actual root cause is determined, which requires a failure mode disambiguation. In order to accomplish this, the DM triggers the execution of a dynamic test, called "active diagnosis". When the active diagnosis procedure is run, additional sensor readings allow the diagnosis system to uniquely identify the failure. In our example, an active diagnosis procedure would command the UAS to climb for a short amount of time. During this commanded climb, all measurements concerning the on-board altimeters (Baro, GPS, LIDAR) are carefully monitored and discrepancies are used to determine the root cause. If, for example, the LIDAR reading remains constant whereas barometric altitude and GPS altitude increase during the climb, it can be reasoned that the LIDAR altimeter is faulty[‖]. This task is performed by the LC, DR, and R2U2 components of AOS and the root-cause failure mode is reported to DM.

Algorithm 2 Listing of schema for contingency action $\mathcal{A}$

```
1:  [G', F', C', P' ] = schema_A(G, X, F, C, P)
2:  C' = get_ac_capabilities(X, F, C)          ▷ what are the current capabilities of the aircraft?
3:  if is_applicable(A, X, C') then
4:      [Y, G1] = outcome(A, X, G)             ▷ what is the outcome of A if it would be executed?
5:                                             ▷ new AC location would be Y with new flight plan G1
6:      [X', F'] = predict_forward(X, F, Y)    ▷ predict state of UAS flying from X to point Y
7:      if is_safe(X', C') then                ▷ A is safe to be executed
8:          [G', P'] = schemas(G1, X', F', C', P.append(A))   ▷ now check the rest of the new flight plan G1
9:      end if
10: end if
```

With ambiguities resolved, the contingency planning is started. The DM employs a recursive schema-based planning algorithm, which uses backtracking search, constraint logic, and logic-based programming.

For each possible contingency action $\mathcal{A}$ (e.g., divert to emergency airport, pull parachute) we define a *schema*, a recursive piece of code, which represents a piece of a contingency plan with "holes", which need to be filled by calculations, constraint reasoning, or recursive invocation of other schemas (Figure 2). The goal of the UAS mission is to fully execute the flight plan F in a step-wise manner, i.e., fly each leg of the flight plan $w_i \rightarrow w_{i+1}$ in sequence given the current state of the aircraft X and failures F. Listing 2 shows an abstracted, high-level description of a schema for contingency action $\mathcal{A}$. The schema is called given the current flight plan G, aircraft state X, failure modes F, external constraints C, and an empty partial contingency plan $P = []$. In a first step, the current aircraft capabilities are calculated based upon the current state, failure modes, and given constraints. After consultation of the aircraft failure model and failure impact model, a new set of constraints is calculated, reflecting the effects of failure on the aircraft behavior. For example, a faulty LIDAR could result in a constraint $alt_{AGL} > 30$ft requiring that the UAS needs to stay at least 30ft above the ground, because barometric and GPS altimeter have larger error margins. Another typical constraint concerns strong climbs, which need to be restricted when engine or elevator failures occur or the battery is weak.

If the considered contingency action $\mathcal{A}$ is applicable in the current state under the current constraints, it is provisionally selected. Then, the implications of $\mathcal{A}$ on the aircraft state Y and the future flight-plan G_1 is calculated by the function `outcome`. Depending on the contingency action that might be $G_1 = G.rest()$ if we can fly to the next way-point, or G_1 might be set to the route to a suitable emergency airport. The code for `outcome` can include queries to aircraft and failure models, queries to the operational database (e.g., to obtain a route to an emergency landing spot), as well as calls to other schemas.

If the contingency action $\mathcal{A}$ is to be carried out successfully, the aircraft must be able to safely transition from the current state X to Y, e.g., be able fly to the first waypoint of the emergency route. This is checked by a simple forward prediction of the system state of the aircraft. This task involves both checking of constraints as well a state updates. Typical constraints concern limitations of altitude or speed, violations of reserved areas of the airspace, or limitations on maneuvers. State updates usually concern consumables. For example, the battery state of charge (SoC) is updated using a simple, deterministic model to reflect the drain of the battery while executing the flight from X to Y. Forward prediction can even change the failure modes if, for example, $\mathcal{A}$ is a failure mitigation action.

[‖] Here we assume a hovering climb of a copter to rule out changes in the terrain.

If the outcome of the forward prediction is deemed to be safe, action $\mathcal{A}$ is incorporated into the plan. The DM then needs to recursively needs to plan for the remaining parts of G' under the updated aircraft state X', failure modes F', and constraints C'.

In case any of the checks fail, the contingency action $\mathcal{A}$ cannot be considered at this point. Our logic-based search algorithm therefore performs a backtracking step and tries the next available schema. The backtracking search guarantees that all possible alternatives can be tried. The schemas are ordered in such a way that the least mission-intrusive contingency actions are tried first. Table 3 lists a selection of relevant schemas.

Table 3 Schemas for on-board contingency planning

Action $\mathcal{A}$	Severity	intended flight plan G	Description
empty	0	[]	mission concluded, no action to be taken
flight-plan	0	$\langle w_i, ... \rangle$	follow flight plan to next waypoint w_i
switch-battery	1	$\langle w_i, w_{i+1}, ... \rangle$	switch to backup battery and plan with new battery capacity
shortcut	1	$\langle w_i, w_{i+1}, ... \rangle$	skip waypoint w_i. Fly directly to w_{i+1}
deviate-airport	2	$\langle w_i, w_{i+1}, ... \rangle$	deviate to emergency airport at waypoint $w'_\perp$ using emergency flight plan $\langle w'_1, w'_2, \ldots, w'_\perp \rangle$. Original mission G is terminated
land-immediate	3	$\langle w_i, w_{i+1}, ... \rangle$	land at nearest safe waypoint w'. G is terminated
parachute	4	any	pull parachute at current location X. G is terminated

With this search-based contingency planning algorithm, the DM will always come up with a contingency plan that

- tries to perform the original mission as long as it is possible in a safe manner,
- tries to execute mitigation actions that change the configuration of the UAS, e.g., to switch to a backup battery or to turn of unnecessary subsystems to save battery power,
- tries to select contingency actions that are as little disruptive to the mission as possible. This includes
  - a shortened flight plan that sacrifices not so important goals, for example, a package delivery,
  - diverting the UAS to a suitable and safe emergency airport, and
  - pulling the parachute as a last resort to minimize crash impact.

Numerous additional constraints must be obeyed during the search for a suitable contingency plan. For example, certain contingency actions must not be repeated (e.g., switching of batteries), others might be tried in certain intervals. Our reasoning-based framework furthermore allows us to use constraint satisfaction techniques to calculate important parameters of the contingency procedures. For example, a safe all-engines-out glide into an emergency airport requires a minimal altitude at the preceding waypoints. This minimal altitude is automatically propagated backwards to the current AC position and will cause a backtracking step if the AC is not high enough for the glide.

VIII. Case Studies

A. Case Study 1

This case study is based upon a small fixed-wing UAS owned by a sub-team of AOS at NASA'S Langley Research Center. This fixed-wing RC class aircraft model (see Figure 10a) is a model of an Extra 260, given a tail number of "R5". The LaRC group modified the airframe for electric propulsion and included extra sensors for research purposes. In this case study, we are concerned with an actuator failure occurring during the flight. We first describe the relevant details of the actuator model and then the steps comprising the specific failure scenario demonstrated in simulation.

1. General system and failure information

The LaRC R5 has a flight control system in which one current sensor measures the current going to all of the left-side actuator servos: left aileron, left flap, left elevator, and the rudder. A second current sensor measures the current going to the right-side actuators: right aileron, right flap, right elevator. Figure 10b shows the two circuits measuring the servo currents I_1 and I_2.

a) 

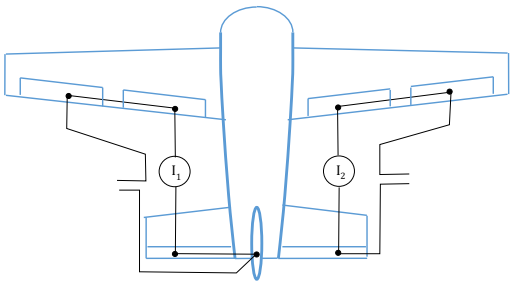
b) 

Fig. 10 a) Langley R5 (68 in. wingspan) from [15]. b) Schematic of servo current sensors on the R5.

As with the real electrical servos used on the R5, in our simulation, the actuators are modeled as having an increased and static current when they are stuck, even at a neutral position. By looking at the current sensor data, it is then possible to tell, on which side of the aircraft an actuator became stuck, but not which of the actuators is stuck. This results in an ambiguous initial diagnosis.

Our failure model (Section VI) contains information about the setup of the actuators, the available sensors, and the failure modes. Ambiguity-groups are comprised of sets of failure modes that cannot be distinguished, here, for example: left aileron, left elevator, rudder, or left flap. Different active diagnosis procedures have been modeled that can help in the disambiguation of the failure modes. Furthermore, the model contains information about the operational impact of the failure modes. For example, a stuck elevator limits the capability of the aircraft to execute steep climbs, while a stuck rudder limits sharp turns.

2. Scenario Details

For our scenario, we assume that the aircraft's mission is to fly along the waypoints $A - B - C - D - L$, where L corresponds to the intended target airport. Figure 11a shows a stylized map of the flight plan. This map also contains two emergency airports E_1 and E_2 with the possible routes to the emergency airports shown as dashed lines. There is a large South-North mountain range, which requires a steep climb to fly over the mountain range.

At the beginning of the scenario that flight plan is active and the aircraft approaches waypoint A. At that point, the failure, a stuck elevator, occurs by command injection into the simulator.

3. Diagnosis and Contingency Management

The failure causes an increase in servo current I_1, which is detected by LC and causes DR to produce an initial ambiguous diagnosis containing several failure modes. This ambiguous initial diagnosis will cause DM to command an active diagnosis action. In this case, the active diagnosis action, which is read from the failure model, is to perform one or more doublets. A doublet is a small rotation back and forth about one of the aircraft axes. For example, a pitch doublet is executed by commanding the aircraft to pitch up for a short time, followed by a short pitch down. Similarly, a roll doublet consists of a short left banking followed by a right banking. The data from the aircraft angular rate sensors (rate gyros) are monitored during the doublet to check if the aircraft angular response is normal or degraded. If the response is degraded about a particular axis, the ambiguous diagnosis is resolved to the actuator, which controls that axis. In our scenario, a degraded performance in the pitch axis is detected, disambiguating the failure to "elevator stuck".

After the failure has been fully isolated, the decision-maker module checks the current flight plan to see if it is impacted by this failure. If it is, DM may change the flight plan, or perform other mitigation actions. More specifically, during the scenario, AOS carries out the following actions caused by the data flow shown in Figure 11b:

1) The failure occurs, and the data monitored by DR shows an increased current on the left side of the aircraft.
2) This diagnosis is ambiguous and is reported to DM. The possibilities for failure modes are: {left-aileron-stuck, left-flap-stuck, left-elevator-stuck, rudder-stuck}.
3) DM determines that active diagnosis is needed to disambiguate the failure. It commands R2U2 to begin monitoring the higher-rate angular velocity sensors from the aircraft, and then commands the aircraft to perform a pitch doublet.
4) R2U2 analyzes the aircraft's angular velocity response to the doublet and determines that the response was

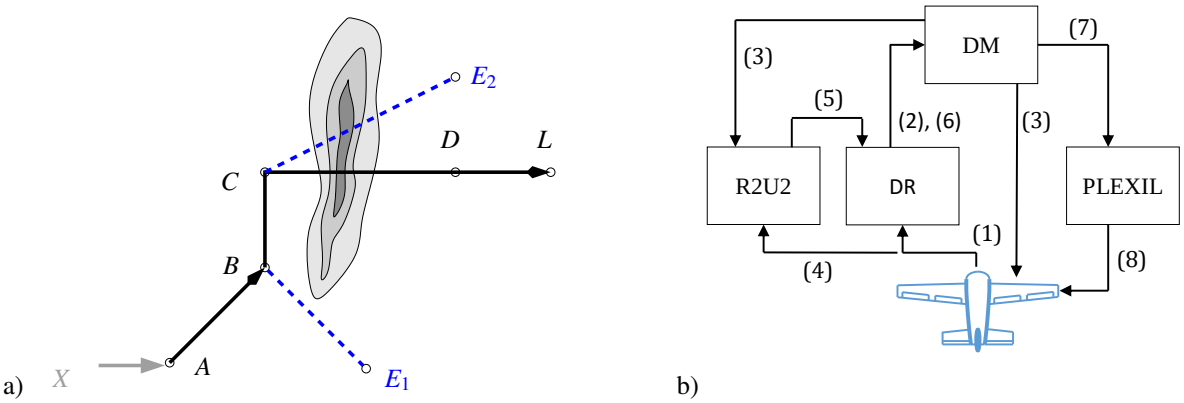


Fig. 11 a) Diagram of waypoints with possible divergence routes (dashed) and emergency airports E_1, E_2. b) Diagnosis and Contingency Planning Steps.

degraded: only half the control authority is remaining about the pitch axis. If, on the other hand, R2U2 had found full control authority about the pitch axis, these steps would be repeated for the roll and yaw axis.

5) R2U2 sends "degraded control authority" as additional diagnostic information to DR.
6) Given the additional diagnostic information, DR is able to resolve the diagnostic ambiguity. It reports that the left elevator stuck as the only possible failure mode.
7) DM takes the reported failure mode, and analyzes the aircraft's flight plan to determine if it is still feasible in light of the failure or if a contingency action has to be performed. This analysis is described in Section VIII.A.4 below. In this scenario, DM constructs, an alternate flight plan that diverts the aircraft to emergency airport E_1. This alternate flight plan is sent to PLEXIL.
8) PLEXIL finally executes the new flight plan.

Table 4 DM reasoning steps after left-elevator-stuck failure

Step	schema_$\mathcal{A}$	X	Goals G	Plan P	Action
1	fltp	X	$\langle A, B, C, D, L\rangle$	$\langle\rangle$	ok
2	fltp	A	$\langle B, C, D, L\rangle$	$\langle A\rangle$	ok
2	fltp	B	$\langle C, D, L\rangle$	$\langle A, B\rangle$	ok
3	fltp	C	$\langle D, L\rangle$	$\langle A, B, C\rangle$	fail(climb needed)
4	shortcut(C)	B	$\langle D, L\rangle$	$\langle A, B\rangle$	fail(climb needed)
5	divert(E_2)	C	$\langle D, L\rangle$	$\langle A, B, C\rangle$	fail(climb needed)
6	-	B	$\langle C, D, L\rangle$	$\langle A, B\rangle$	fail(no solution for C)
7	divert(E_1)	B	$\langle C, D, L\rangle$	$\langle A, B, E_1\rangle$	ok(solution)

4. Decision-making for the elevator stuck scenario

After determination of the exact failure modes, DM is provided in Step 7) with the following information: intended flight-plan $G = \langle A, B, C, D, L\rangle$, failure mode F = "left-elevator-stuck", and the current position of the aircraft X, which is approaching A. Table 4 shows, which reasoning steps are carried out for this scenario. In a first step, the schema to follow the flight-plan ($\mathcal{A}$ = "fltp") is tried. Flying from X to A and then B and C is still safe and possible even under the constraints of degraded performance caused by the failure. However, the leg $C \rightarrow D$ is not possible, because a strong climb is required to fly over the mountain range, which violates our constraints.

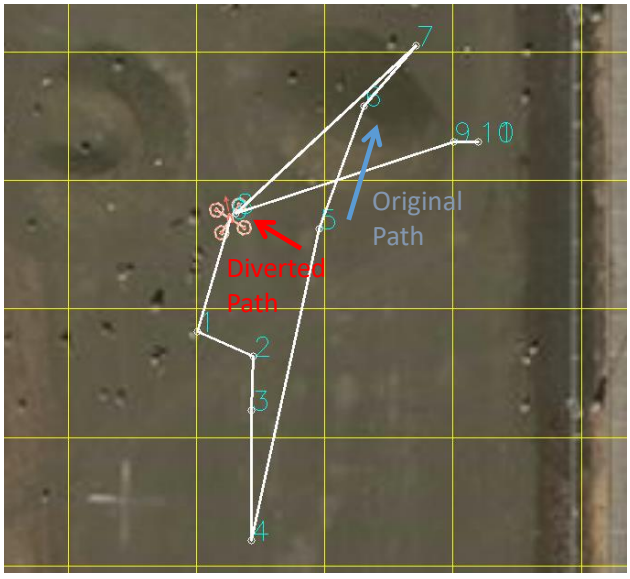
DM then backtracks and searches for alternate possibilities that do not violate any flight constraints. This means that other schemas are tried in sequence. It finds and rejects two alternates, one trying to take a shortcut from B to D

(skipping C), and another trying to divert to emergency airport E_2. Both of these alternatives also would require flying over the mountain range and thus violate the constraint prohibiting a strong climb. DM has to backtrack again (to B) and, finally, tries a route that diverts to E_1. This path does not violate any constraints. DM chooses this as the new flight plan, and sends it to be executed. AOS demonstrated this scenario successfully in simulation.

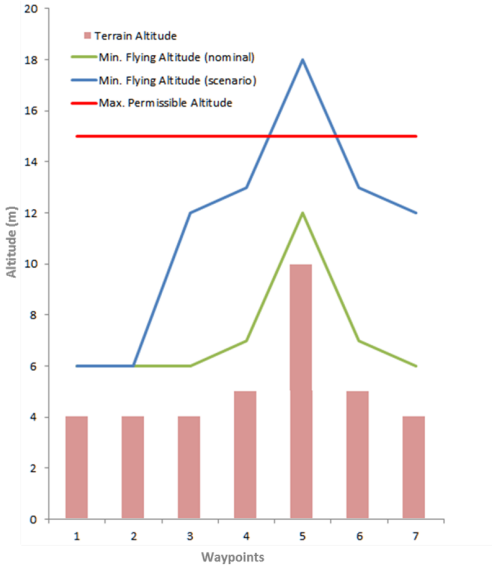
B. Case Study 2

The contingency management in this case study was demonstrated on an actual test flight at the NASA ARC Roverscape using a DJI S1000 octo-copter (Figure 12). This scenario involves a LIDAR sensor, which gives a precise measurement of the S1000's altitude above ground (AGL). The original flight plan (Figure 13a) requires accurate altitude estimation to fly over a tall hill (Figure 13b) as the UAS must also obey a ceiling altitude (similar to the FAA airspace ceiling for small UAS systems). During the flight, however, the LIDAR fails (by failure injection) and the remaining altitude sensors (GPS and barometric altitude) do not meet the accuracy requirements. At that point in time, the on-board DM determines that the original flight plan is not feasible and commands a diversion to a waypoint that does not require crossing the hill.

Fig. 12 DJI S1000+ octo-copter with on-board AOS (Photo courtesy Jonas Jonsson, SGT Inc/NASA ARC)



a)



b)

Fig. 13 Case Study with S1000 octo-copter. a) waypoints at NASA ARC Roverscape (Satellite Image ©Google Maps). b) terrain and minimal altitude AGL for nominal and failure condition

1. LIDAR failure scenario details

In the specific scenario demonstrated by AOS, the injected fault is the LIDAR sensor failure. The flow of information for this scenario is similar to the one in Section VIII.A. The data from the LIDAR and barometer were synthesized, and not obtained from from the flight vehicle, due to tight time constraints. The diagnostics done by DR was on that synthesized data, and R2U2 was not needed for additional data analysis. However, PLEXIL and DM were being exercised fully and using the position data from the on-board GPS sensor. A failure model for the S1000+ with the additional LIDAR sensor was developed and used for this case study.

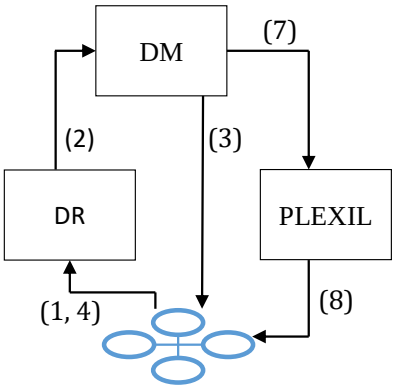


Fig. 14 Information flow and involved components for the S1000+ case study

The individual steps of health monitoring and contingency planning (shown in Figure 14) are

1) The failure occurs and the (synthetic) data analyzed by DR shows that the LIDAR and the barometer altitude sensors disagree.
2) DR reports the ambiguous diagnosis to DM, consisting of the two possibilities: {LIDAR-sensor-broken, baro-altimeter-broken}.
3) DM determines that an active diagnosis is needed to disambiguate the failure. It commands the copter to do perform a climb.
4) DR analyzes the new LIDAR and barometer sensor information, and determines that the LIDAR measurements did not respond to the altitude increase, and therefore LIDAR is the faulty sensor.
5) DM takes the reported failure mode and analyzes the aircraft's flight plan to determine if it is still feasible in light of the failure (see below). In this scenario, the flight plan is found to be infeasible, and an alternate flight plan is determined. The alternate flight plan diverts the S1000+ to a different waypoint. This alternate flight plan is sent to PLEXIL as the new flight plan to execute.
6) PLEXIL executes the new flight plan, sending waypoints to the S1000+ when needed.

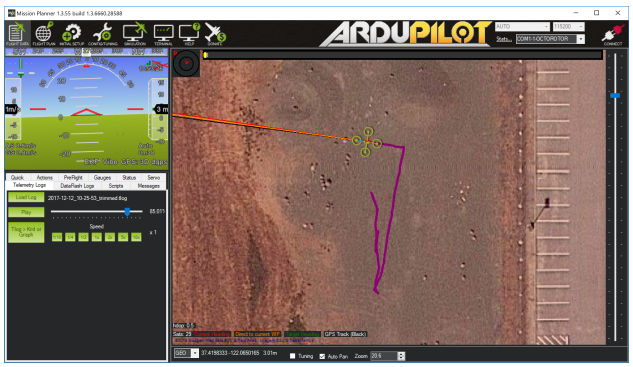
2. Decision-making for the LIDAR-failed scenario

The decision-making for the LIDAR-failed flight scenario is based on the set of waypoints shown in Figure 13b. The original flight plan requires going over a hill, building, or other obstacle at Waypoint 5. With LIDAR working and providing more accurate altitude measurements, the S1000+ can fly at a lower altitude over the obstacle without risk of crashing. This is shown by the green line in the plot above. When the LIDAR is reported as failed, the estimate of altitude above ground from the barometer is not as good, and thus the S1000+ needs to be given more of a cushion and distance above the obstacle. This is shown by the blue line in the plot above. However, DM determines that the increased altitude required will violate the maximum permissible altitude constraint for a small UAS. The original flight path is therefore deemed infeasible, and DM determines a new path with a process similar to that in the fixed-wing scenario. In the image above, the new path is shown with a blue line, which avoids the tall obstacle.

3. Flight data from the S1000+

This scenario was demonstrated successfully during an S1000+ flight with failure injection. Figure 15a shows the ground track of the S1000+ during the flight. During the experiment, the S1000+ flew the route described above, going to Waypoints 1, 2, 3, and 4, then diverting instead of proceeding to Waypoint 5. The active diagnosis (climb) occurred between Waypoints 2 and 3. Figure 15B shows the estimated altitude of the S1000+ during the flight, based upon GPS

and barometric altitude. The climb, which was carried out during active diagnosis is clear to see.

a) 

b) 

Fig. 15 Case Study with S1000 octo-copter. a) ground track of S1000+. (Image ©Google Maps). b) estimated altitude

IX. Related Work

Due to restricted on-board computing capabilities, contingency management for autonomous spacecraft have been kept very simple: typically, when a failure occurs during a mission, the vehicle enters a defined "safe" mode and waits for commands from the ground station. Popular UAS autopilot systems, like ArduPilot** or PX4 [16] provide a simple method for contingency management in the form of "fail-safe" actions: if a failure or a specific predefined condition occurs, then immediately a specific emergency action is triggered. Typical actions include: return to home plate, loiter at the current location, or land immediately.

Approaches to more complex contingency planning for UASs have been developed for the planning and pre-flight assessment [17] or concern a "holistic" multi-level contingency management system that spans UAS, communications, weather, and battle teams [18]. Here, however, the actual planning uses the Lockheed Martin tool TeamWorks [18] and only has limited control or monitoring capabilities on-board. On-board path planning is described, for example in [19]. This approach performs on-board dynamic probabilistic reconfiguration of the trajectories, but does not incorporate diagnosis or failure-based contingency management. A system for flight mission planning, which addresses the specific problem of autonomous recharging is presented in [20]. It uses elaborate graph optimization algorithms, but is not being run on-board.

Current UAS autopilot systems continuously monitor the UAS, but do not perform on-board diagnosis. Numerous approaches and tools for model-based diagnosis have been made. The workshop series on principles of diagnosis [21], or the conference series on Prognostics and Health Management (PHM)†† provides a good overview of current research.

In the past, the Timed Failure Propagation Graph (TFPG) models and the diagnosis and anomaly detection algorithms in the associated tool set (Fault Adaptive Control Technology or FACT) have been used for real-time diagnosis, reconfiguration, and fault management of a generic aircraft fuel management system [14, 22], actuator faults in manned and unmanned rotor crafts [22, 23], and real-time software health management [24]. In these cases the focus was on real-time diagnosis. A prescribed reconfiguration was followed in most cases. A model based diagnosis and deliberative reasoning scheme [25], wherein the reconfiguration decision was based on the solution of a Boolean satisfiability problem, was demonstrated for reliable software health management.

Our current work is different in that it uses and extends the TFPG models to capture the impact of faults on system performance and expected functionality. Furthermore, our approach models the performance impact of possible solutions for refining the diagnosis results and reconfiguring the system. The decision maker explores viable alternatives presented in the model taking into account current mission conditions and additional constraints imposed by the faults and the associated reconfiguration.

**http://ardupilot.org/ardupilot/

††https://www.phmsociety.org/

X. Conclusion

We have described the Autonomy Operating System (AOS), and the model-based diagnosis and decision-making capabilities it contains. The capabilities are implemented in several submodules, DR, R2U2, and DM. They were exercised in two scenarios, each with a different target system: a simulated fixed-wing electric aircraft, and a real flight experiment with a DJI S1000+ octo-copter. In each scenario, AOS successfully detected a failure, performed active diagnosis to resolve the failure ambiguity, and exercised decision-making in order to mitigate the failure and ensure the safety of the vehicle.

Acknowledgments

The CAS (Convergent Aeronautics Solutions) AOS project was funded under NASA ARMD.

References

[1] Lowry, M., Bajwa, A. R., Pressburger, T., Sweet, A., Dalal, M., Fry, C., Schumann, J., Dahl, D., Karsai, G., and Mahadevan, N., "Design Considerations for a Variable Autonomy Exeuctive for UAS in the NAS," *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, 2018.

[2] McComas, D., "NASA/GSFC's Flight Software Core Flight System," *Flight Software Workshop*, 2012.

[3] Gundy-Burlet, K., "Validation and Verification of LADEE Models and Software," *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2013.

[4] Lowry, M., Pressburger, T., Dahl, D., and Dalal, M., "Towards Autonomous Piloting: Communicating with Air Traffic Control," *Scitech 2019*, 2019.

[5] Verma, V., Jonsson, A., Pasareanu, C., and Iatauro, M., "Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations," *Spacecraft Control and Operations, American Institute of Aeronautics and Astronautics Space 2006 Conference*, 2006.

[6] Luo, J., Tu, H., Pattipati, K., Qiao, L., and Chigusa, S., "Graphical models for diagnosis knowledge representation and inference," *Autotestcon, 2005. IEEE*, 2005, pp. 483–489. doi:10.1109/AUTEST.2005.1609185.

[7] Reinbacher, T., Rozier, K. Y., and Schumann, J., "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems," *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science (LNCS), Vol. 8413, Springer-Verlag, 2014, pp. 357–372.

[8] Geist, J., Rozier, K. Y., and Schumann, J., "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems," *Proceedings of the 14th International Conference on Runtime Verification (RV14)*, Vol. 8734, Springer-Verlag, 2014, pp. 215–230.

[9] Schumann, J., Roychoudhury, I., and Kulkarni, C., "Diagnostic reasoning using prognostic information for unmanned aerial systems," *PHM15*, 2015.

[10] Rozier, K. Y., and Schumann, J., "R2U2: Tool Overview," *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, 2017, pp. 138–156. URL `http://www.easychair.org/publications/paper/Vncw`.

[11] Maroti, M., Kecskes, T., Kereskenyi, R., Broll, B., Volgyesi, P., Juracz, L., Levendovszky, T., and Ledeczi, A., "Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure," *8th Multi-Paradigm Modeling Workshop*, Valencia, Spain, 2014.

[12] Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: Systems Modeling Language*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[13] Abdelwahed, S., Karsai, G., and Biswas, G., "A consistency-based robust diagnosis approach for temporal causal systems," *16th International Workshop on Principles of Diagnosis*, 2005, pp. 73–79.

[14] Abdelwahed, S., Karsai, G., Mahadevan, N., and Ofsthun, S. C., "Practical considerations in systems diagnosis using timed failure propagation graph models," *Instrumentation and Measurement, IEEE Transactions on*, Vol. 58, No. 2, 2009, pp. 240–247.

[15] McSwain, R., and Grosveld, F., "Development of a Heterogeneous sUAS High-Accuracy Positional Flight Data Acquisition System," Tech. Rep. NASA/TM-2016-219335, NASA, 2016.

[16] Meier, L., Honegger, D., and Pollefeys, M., "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240. doi:10.1109/ICRA.2015.7140074.

[17] DiFelici, J., and Wargo, C., "UAS safety planning and contingency assessment and advisory reseach," *2016 Integrated Communications Navigation and Surveillance (ICNS)*, 2016, pp. 8E3–1–8E3–16. doi:10.1109/ICNSURV.2016.7486388.

[18] Franke, J. L., Hughes, A., and Jameson, S. C., "Holistic Contingency Management for Autonomous Unmanned Systems," 2006.

[19] Wzorek, M., and Doherty, P., "Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle," *2006 International Conference on Hybrid Information Technology*, Vol. 2, 2006, pp. 242–249. doi:10.1109/ICHIT.2006.253618.

[20] Tseng, C., Chau, C., Elbassioni, K., and Khonji, M., "Autonomous Recharging and Flight Mission Planning for Battery-operated Autonomous Drones," , 2017. URL arXiv:1703.10049v2.

[21] Zanella, M., Pill, I., and Cimatti, A. (eds.), *28th International Workshop on Principles of Diagnosis (DX'17)*, Kalpa Publications in Computing, Vol. 4, EasyChair, 2018.

[22] Karsai, G., Biswas, G., Abdelwahed, S., Mahadevan, N., and Manders, E., "Model-based software tools for integrated vehicle health management," *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*, 2006, pp. 438–442. doi:10.1109/SMC-IT.2006.51.

[23] Drozeski, G. R., "A Fault-Tolerant Control Architecture for Unmanned Aerial Vehicles," Ph.D. thesis, Georgia Tech, 2005.

[24] Dubey, A., Karsai, G., and Mahadevan, N., "Model-based software health management for real-time systems," *2011 Aerospace Conference*, 2011, pp. 1–18. doi:10.1109/AERO.2011.5747559.

[25] Mahadevan, N., Dubey, A., Balasubramanian, D., and Karsai, G., "Deliberative, Search-based Mitigation Strategies for Model-based Software Health Management," *Innov. Syst. Softw. Eng.*, Vol. 9, No. 4, 2013, pp. 293–318. doi:10.1007/s11334-013-0215-x, URL http://dx.doi.org/10.1007/s11334-013-0215-x.