Analysis and Prevention of MCAS-Induced Crashes

Noah T. Curran, Thomas Kennings, Kang G. Shin
The University of Michigan
Department of Computer Science & Engineering
{*ntcurran,kennings,kgshin*}*@umich.edu*

***Abstract*—Semi-autonomous (SA) systems face the challenge of determining which source to prioritize for control, whether it's from the human operator or the autonomous controller, especially when they conflict with each other. While one may design an SA system to default to accepting control from one or the other, such design choices can have catastrophic consequences in safety-critical settings. For instance, the sensors an autonomous controller relies upon may provide incorrect information about the environment due to tampering or natural fault. On the other hand, the human operator may also provide erroneous input.**

To better understand the consequences and resolution of this safety-critical design choice, we investigate a specific application of an SA system that failed due to a static assignment of control authority: the well-publicized Boeing 737-MAX Maneuvering Characteristics Augmentation System (MCAS) that caused the crashes of Lion Air Flight 610 and Ethiopian Airlines Flight 302. First, using a representative simulation, we analyze and demonstrate the ease by which the original MCAS design could fail. Our analysis reveals the most robust public analysis of aircraft recoverability under MCAS faults, offering bounds for those scenarios beyond the original crashes. We also analyze Boeing's updated MCAS and show how it falls short of its intended goals and continues to rely upon on a fault-prone static assignment of control priority. Using these insights, we present Semi-Autonomous MCAS (SA-MCAS), a new MCAS that *both* meets the intended goals of MCAS *and* avoids the failure cases that plague both MCAS designs. We demonstrate SA-MCAS's ability to make safer and timely control decisions of the aircraft, even when the human and autonomous operators provide conflicting control inputs.

I. Introduction

Semi-autonomous (SA) systems—those that take both autonomous and manual inputs to control their actions—are ubiquitous in the modern world, presenting applications in factories, hospitals, transportation, and more. Often, the purpose of these systems is to improve the safety and efficiency of tasks that take substantial manual effort. Airplanes, for example, use SA control to maintain safe flight while pilots perform other tasks. As a consequence of SA systems' close coupling with safety-critical applications, there is a complicated trade-off between trusting human and autonomous inputs. SA functionality is often included in a system because humans are prone to making mistakes, but autonomous systems are also imperfect. These autonomous controllers serve as a safety-critical component of embedded systems, and as such, their design should be closely scrutinized.

In order to handle these cases where there is conflict in control, it is necessary for an SA embedded system to incorporate a routine for resolving the conflict. Oftentimes, such a routine is hard-coded to always "trust" the input from one entity over the other. For instance, an autonomous system in the Boeing 737-MAX, the *Maneuvering Characteristics Augmentation System* (MCAS), an embedded system controller, was originally given static priority to manually override the pilot's control of the aircraft during aircraft stall events. While this decision was motivated by a lack of trust in pilot control during these safety-critical stall events, it had dire consequences. In 2019, MCAS mistakenly identified a stall event due to faulty sensor data, which ultimately caused the crash of two Boeing 737-MAX aircraft: Lion Air Flight 610 (JT610) and Ethiopian Airlines Flight 302 (ET302). These crashes prompted regulators to mandate Boeing redesign the MCAS before the 737-MAX could fly again [1]. Among the numerous changes Boeing made, one removed the static control priority from MCAS and gave it to the pilots [2].

In practical settings, the consequences from static assignment of control priority in embedded systems is not well documented. Leveraging the circumstances leading to Boeing creating two versions of MCAS that have opposing static priority controllers, we study the differences between their control failures under various fault types (§ II-D). From our analysis, we conclude that the redesign of MCAS takes an incorrect approach. We find that giving one entity the capability to override control of a safety-critical application creates a single point-of-failure, even in the case of the current MCAS version. Rather than defaulting the control to one entity, we argue for a dynamic control conflict arbiter that chooses which entity to allow control based on the aircraft's situation. This removes the single point-of-failure from MCAS, making the aircraft more tolerant of erroneous input from either autonomous or manual control.

Following this embedded system controller design philosophy, we propose a version of MCAS with a dynamic control arbiter, which we call Semi-Autonomous MCAS (SA-MCAS). Unlike the prior implementations of MCAS, SA-MCAS is capable of providing safer control of the aircraft's pitch in the presence of erroneous input from *both* autonomous *or* manual control. Through our investigation, we demonstrate that SA-MCAS can select which operator to control the aircraft in the presence of erroneous sensor readings or erroneous pilot control. We test the robustness of SA-MCAS under a dataset of representative flight scenarios under which MCAS may activate. Under these flight scenarios, we subject SA-MCAS to numerous data, control, and timing errors, summarized in § III.

Previously, there have been a few publicized analyses and reports of MCAS [3], [4]. Mainly, these were conducted by aeronautic enthusiasts and reporters who wished to demonstrate *how* the crash occurred. They sparsely discuss the choices made in the design of the embedded control logic without considering or mentioning alternative designs. In contrast, we present the first in-depth analysis of the control logic of MCAS, with the goal of finding a design alternative with better overall safety.

Prior work on the resolution of control conflict of SA systems is scarce. In the context of cars, there is some prior work on con-

Acknowledgements: The work in this paper was supported in part by ONR under Grant No. N00014-22-1-2622.

flict resolution between autonomous controllers and drivers [5]. Alongside this line of work, there is research in sensor anomaly detection [6]–[8], estimation [8], [9], and fault injection [10], [11]. While this work is somewhat relevant to our research, we focus on arbitration of conflicting autonomous and manual control.

This paper makes the following novel contributions:

1) We build a MATLAB/Simulink template[1] for simulating control input for aircraft modeled in JSBSim [12]. We provide the building blocks for easily creating and evaluating new aircraft control systems. (§ V)
2) We model timing and control constraints on the aircraft, determining the parameter boundaries for the recoverability of safe control of the aircraft. We conduct this for Boeing's original (MCAS_{old}), new (MCAS_{new}), and our (SA-MCAS) versions of MCAS. Our analysis tweaks the MCAS response time, the MCAS duration, the fixed time interval between MCAS events, and the pilot's reaction delay to MCAS. (§ III and § VI)
3) For the first time, we model a comprehensive failure analysis for both MCAS_{old} and MCAS_{new} in the Boeing 737-MAX. This model incorporates incorrect sensor data that MCAS relies upon and dangerous flight control that leads to stalls. Our analysis uncovers previously unseen flight scenarios resulting in aircraft crashes due to erroneous control input, in addition to the scenarios that occurred in the original Boeing 737-MAX crashes. (§ III and § VI)
4) We propose SA-MCAS, which makes control decisions that account for erroneous inputs from the pilot or autonomous system. SA-MCAS is shown to improve the state-of-the-art, loosening the constraints for recoverability of flights to safe control in comparison to MCAS_{old} and MCAS_{new}. (§ VII)

II. Background & Motivation

For better comprehension of the problem, we provide the necessary knowledge for understanding the Boeing 737-MAX crashes. We start with a basic explanation of longitudinal flight dynamics. Such information is essential for understanding why Boeing chose to include MCAS in the 737-MAX and for understanding how MCAS impacts the aircraft's control. We then discuss the brief history of the MCAS implementation and how its design flaws caused the crashes of JT610 and ET302. Since these crashes, both the FAA and Boeing have introduced new requirements for MCAS to avoid future accidents. We analyze these new requirements to motivate our investigation.

A. System Components

The components of the aircraft system that are of concern to this work are summarized in Fig. 1. The *Air Data Inertial Reference Unit* (ADIRU) is at the core of the flight control system. It is information supplied from various sensors, which enables the calculation of the airspeed, angle-of-attack (AoA), altitude, position, and other inertial and environmental information. As seen in Fig. 1, there is a redundant ADIRU available in the Boeing 737-MAX.

The information from the ADIRU is passed along to the flight control computer, which is responsible for actuating the aircraft control surfaces. In the case of pitch control, it uses the Mach information from the ADIRU to trim the *horizontal stabilizer* (HS). The flight dynamics and control that govern this process are described in the following subsection. Moreover, the pilots can manually trim the HS and compete for control of the aircraft with the flight control computer.

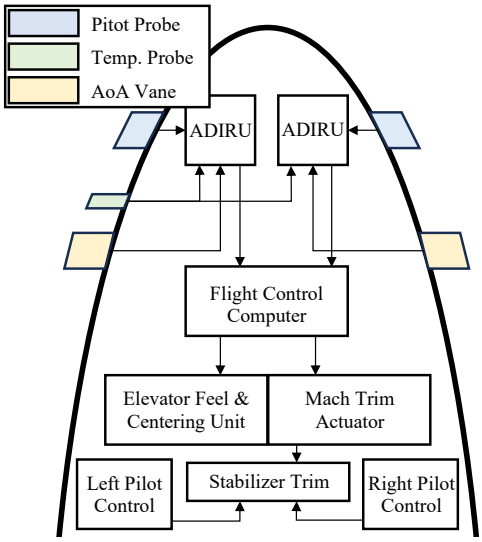


Fig. 1: The components of the Boeing 737-MAX aircraft pitch control stability system. This was adapted from those found in [13].

B. Longitudinal Flight Dynamics & Control

The longitudinal flight dynamics involve moments about the aircraft's Y-axis. The longitudinal equations[2] describe the physics of the angular velocity of the pitch ($\dot{\theta}$), the angular acceleration of the pitch ($\dot{q}$), the acceleration ($\dot{U}$) in the X-direction, and the acceleration ($\dot{W}$) in the Z-direction. These equations can be reduced to state-space form as

$$\begin{bmatrix}\dot{U} & \dot{W} & \dot{q} & \dot{\theta}\end{bmatrix}^\intercal = A\begin{bmatrix}U & W & q & \theta\end{bmatrix}^\intercal + Bu, \quad (1)$$

where A is the state matrix, B is the input matrix, and $u = \begin{bmatrix}\delta_e & \delta_p\end{bmatrix}^\intercal$ is the control vector. This is a linear system that is controlled by the deflection of the elevator (δ_e) and the change in the thrust (δ_p). To examine control design, we can approximate the longitudinal dynamics using simple models. For our purposes, we can focus on the short period approximation [14]. For the short period approximation, the response from θ and w are in the same phase, and the response from u and q are very small. The resulting equation is

$$\dot{x}_{sp} = \begin{bmatrix}\dot{w} & \dot{q}\end{bmatrix}^\intercal = A_{sp}x_{sp} + B_{sp}\delta_e. \quad (2)$$

Here, we assume that the thrust remains constant in steady flight. The control input of interest is the deflection of the elevator, as this is what the pilot uses to directly impact the pitch of the aircraft. The elevator is attached to the HS, which is what MCAS controls in order to automatically trim the aircraft's pitch.

C. MCAS

1) Why Was MCAS Necessary? During the design of an aircraft, regulatory bodies will provide a type certificate once it is deemed safe for flying in the air. If a newly designed aircraft is the same type as a previously certified aircraft, the regulation is expedited since certain preliminary prototypes are unnecessary. Instead, just the parts of the aircraft that are changed need to

[1]Available on GitHub: https://github.com/noah-curran/SA-MCAS.

[2]See [14] for the full longitudinal equations.

be tested. An additional benefit of this practice is the reduction in the amount of training the pilots of the previous aircraft require in order to operate the new aircraft. The type certificate is amended in order to include the updates made.

During the design of the 737-MAX line of aircraft, Boeing sought to certify it as a 737 variant to take advantage of this certification amendment process. One such change was made to the engines, in which the 737-MAX moved from the CFM56 engine to the LEAP-1B engine, which is larger and placed farther forward on the wings of the aircraft. Testing revealed that when the 737-MAX encountered high-pitch scenarios at low airspeeds, the weight distribution of the new engines would push the nose of the 737-MAX further upward and cause the aircraft to gain a high Angle-of-Attack (AoA) and subsequently stall.[3] To address this issue, Boeing would either need to (A) redesign the entire aircraft type to accommodate the engines and follow a long and expensive type certification process, or (B) use some flight control mechanism to counter the problematic stall behavior. Since the former option would compromise Boeing's goals to get the 737-MAX on the market quickly and reduce the costs of pilot training, they provided the aircraft with MCAS$_{old}$, a flight stabilization program that automatically pitches the aircraft down to prevent a stall during high-AoA maneuvering.

2) How Does MCAS Work? In order to determine whether a stall event is about to occur, MCAS observes the AoA of the aircraft through a sensor. The AoA sensor is a swept vane that is aerodynamically aligned with the aircraft in order to measure the angle of the airflow passing the wing. While the 737-MAX is equipped with two AoA sensors (one on both sides of the nose of the plane), MCAS$_{old}$ used just one of the sensors. In response to a high AoA (defined as $\sim$17°), MCAS$_{old}$ provides a nose-down control input to the HS to avoid a stall. At low speeds, this nose-down deflection is 2.5° and at high speeds it is 0.6° [3]. During the high-speed stall events, MCAS$_{old}$ checks for a high g-force in addition to the AoA. The g-force check is omitted during low-speed flight.

3) What are the Functional Requirements of MCAS? For the functional safety of any part aboard an aircraft, the FAA follows ARP4761 and ARP4754 to provide an assurance level for the design [15], [16]. In the case of MCAS, the FAA designated it as a "hazardous failure" system, which should have a probability of occurring at $<10^{-7}$ per flight hour. In this case, "failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers." These safety requirements assume an undistracted pilot can respond to an issue within 3 seconds [3].

4) How Did MCAS Cause Deadly Accidents? The issues with MCAS$_{old}$ occurred due to design choices of its activation during low-speed stall events. Because there was no g-force check and only one AoA sensor was checked, a single-point-of-failure was present. The g-force check is primarily to determine whether pilots need assistance during high-speed events, which cause the pitch control column to become too heavy and cumbersome to control. Boeing's initial disclosure of MCAS to the FAA accounted for its necessity to activate during high-speed stall events. The failure analysis from Boeing's disclosure demonstrated that MCAS$_{old}$ was less intrusive to the flight controls (and deemed a low risk) due to the improbability of its failure and due to the redundancy the g-force check provided during the high-speed stall events. While the issues experienced during high-speed stall events were a non-issue during low-speed stall events, a high g-force could still be useful as a source of redundancy to detect the stall event. The original authority MCAS$_{old}$ was granted to provide nose-down deflection to the HS was limited to 0.6°. However, after discovering issues during low-speed tests, Boeing provided MCAS$_{old}$ additional authority. In short, during low speeds the pitch control surface of an aircraft requires more deflection in order to yield the same response as during high speeds, so Boeing increased the nose-down deflection amount to 2.5° for low-speed events. After making this substantial change to MCAS$_{old}$, Boeing failed to notify the FAA and made no mention of MCAS$_{old}$ in the 737-MAX's pilot manuals.

The increased likelihood of a dangerous event occurring compounded with pilots inadequately prepared to handle an erroneously engaged MCAS$_{old}$ created a recipe for disaster. Two deadly crashes followed: ET302 and JT610. During these flights, the AoA sensor delivered faulty readings that made MCAS$_{old}$ believe the airplane's AoA was too high. Consequently, the nose of the aircraft was pushed down by MCAS$_{old}$. To counteract MCAS$_{old}$, the pilot manually trimmed the HS and pulled back on the column to actuate the elevator to undo the nose-down deflection. MCAS$_{old}$ again displaced the HS due to the sensor's incorrect readings, entering a state known as a "runaway stabilizer". After back-and-forth between MCAS$_{old}$ and the pilot, the HS was eventually displaced so much that elevator deflection could not counter the effects of the much larger HS. Also, due to aerodynamic factors, the manual HS hand-crank available in the cockpit eventually would not budge. In both catastrophic cases, the aircraft entered a steep nosedive and crashed. The two crashes killed all 346 people onboard and resulted in the grounding of all Boeing 737-MAX aircraft globally. While skilled pilots were sometimes capable of landing aircraft that MCAS$_{old}$ negatively impacted, these instances were not reported to any regulatory agencies until after the deadly crashes [17].

5) How Did MCAS Change After the Crashes? In response to these crashes, Boeing proposed a redesigned MCAS$_{new}$ with several changes [2]. First, MCAS will now check both the left and right AoA sensors. If the AoA sensors exceed 17° when the flaps are not up or if they disagree with one another more than $>5.5°$, MCAS$_{new}$ will not activate. Additionally, Boeing introduced Mid-Value Select (MVS) to pick an AoA value when they disagree within the acceptable range [4]. MCAS$_{new}$ will store the previously selected AoA value and during the next iteration it will pick the median between the stored, left, and right AoA values. Second, MCAS$_{new}$ will only activate once per sensed event rather than an unconstrained number of times, preventing a runaway stabilizer. Lastly, when MCAS$_{new}$ does engage, pilots can now override it and perform manual flight at any time since it will not provide more input on the HS than the pilot can put on the elevator. The final revision approved by the FAA included an additional requirement to the flight control computer, requiring an integrity monitor in order to stop erroneously generated trim commands from MCAS$_{new}$ [18].

[3]A stall is a flight event where there is not enough lift under the wing of the aircraft, causing the aircraft to lose altitude. These normally occur when the AoA is $\gtrsim$17°.

D. Analysis of the Revised MCAS Requirements

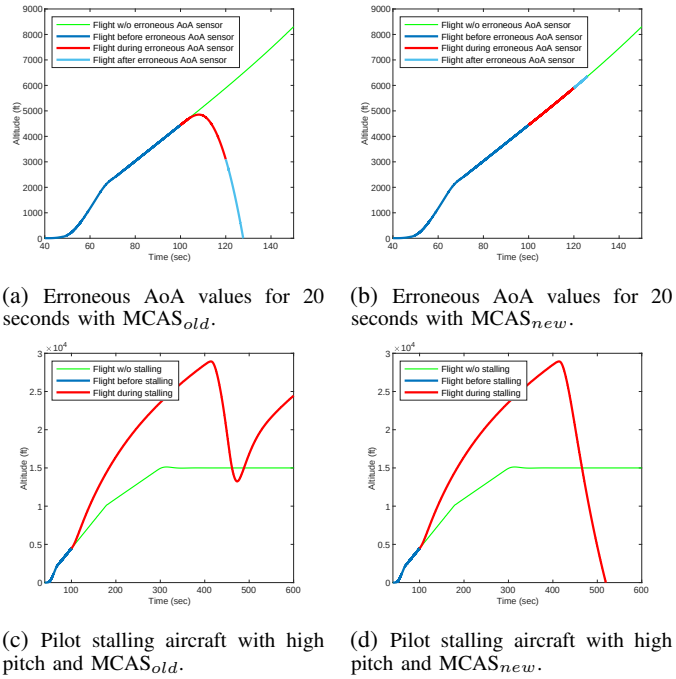
Boeing's revised requirements for MCAS$_{new}$ made a major pivot in control authority. MCAS$_{old}$ was originally designed with absolute authority; in fact, there was not even a switch for cutting off its control of the HS in the event of a runaway stabilizer. However, in MCAS$_{new}$, there is a clear lack of trust for autonomous control reversing control authority in favor of the pilot. This change is notable, as it is contrary to Boeing's original goals of providing mechanisms to emulate the feel of the 737-NG and reducing the amount of training required for the pilot. Now, the pilot must learn how to safely counter the MCAS in the event of additional problems occurring.

These additional problems are not completely unimaginable. While the choice to use an agreement between the left and right AoA sensors to validate their use is an improvement over the single-point-of-failure, the AoA sensors are subject to the same environmental factors and hence the two sensors may possibly agree with each other on an incorrect value. This differs from the case of MCAS activation at high-speed, which is contingent on a high g-force value as well. Rather than seeking a source of redundancy within an entirely separate system, Boeing chose to consider just the unused AoA sensor.

Such scenarios where both the autonomous entity and a human compete for control of a vehicle is called a *semi-autonomous* (SA) system. Defaulting control authority to one entity in the case of disagreement is a common trend in SA system design. While Boeing designed MCAS$_{old}$ and MCAS$_{new}$ with this default behavior, one can see cases where the pilot is more trustworthy and others where the MCAS is: there are instances in flight where either the pilot or the autonomous system could be incorrect. On one hand, sensor failures have and will continue to occur, and on the other hand, pilots may not respond quickly or correctly enough during chaotic flight scenarios. Thus, we argue that neither the original design nor the redesign of MCAS is the right choice.

To back this claim, we reconstruct the behavior of MCAS$_{old}$ and MCAS$_{new}$ using information *The Seattle Times* [3] and the FAA [18] reported to the public and conduct a preliminary analysis (Fig. 2). Using the open-source flight dynamic simulator JSBSim [12], we built a toolkit for running MCAS experiments (see § V for more details of how it works). The preliminary analysis is our first step to investigate the safety of the MCAS$_{new}$. The simple experiment runs a takeoff maneuver with either an injection of an erroneous AoA sensor at $t = 100$s or a pilot beginning to stall the aircraft with a high pitch at $t = 100$s. While MCAS$_{new}$ mends the original single-point-of-failure issue, it introduces a new hazard related to pilot stalling that was not present with MCAS$_{old}$. With an aggressive pitch-up control from the pilot, the MCAS system will not start recovery until *after* it detects a stall is occurring, i.e., the AoA exceeds $\sim 17°$. As a result, the flight will still lose some altitude such as in Fig. 2c while MCAS$_{old}$ recovers the aircraft. On the contrary, MCAS$_{new}$ can only adjust the HS once, which is not enough for recovery (*cf.* Fig. 2d).

However, this evidence just proves the *possibility* and does not explicitly answer *why* or *how* the MCAS implementations directly contribute to the crash of the aircraft. To formally investigate these questions, we provide a framework for defining the error model in § III, which incorporates various modes of sensor failure as well as the timing deadlines that pilots must meet for aircraft recoverability.



(a) Erroneous AoA values for 20 seconds with MCAS$_{old}$.

(b) Erroneous AoA values for 20 seconds with MCAS$_{new}$.

(c) Pilot stalling aircraft with high pitch and MCAS$_{old}$.

(d) Pilot stalling aircraft with high pitch and MCAS$_{new}$.

Fig. 2: Simulation of the fault-tolerance of MCAS$_{old}$ and MCAS$_{new}$. For each, we simulated 737-MAX takeoff using our custom toolkit (§ V) built on top of JSBSim [12].

III. Error Model

From our previous demonstration, we conclude that pitch control of the aircraft is fallible through the input of either the autonomous MCAS or the human pilot. In this section, we provide a more general model for the behaviors that can cause these control failures. We do not consider errors that are innate to the flight compute platform itself, such as software bugs or attack vectors that adversaries may exploit. We consider these types of errors beyond the scope of our work, but individual components of the flight platform may utilize a trusted platform modules to verify the integrity of the controller on boot [19] or mathematical verification [20] to ensure the correctness of the estimators used in Alg. 1. Moreover, the estimators used in Alg. 1 will undergo certification by the FAA, so such safety-critical implementation concerns should be resolved during this process.

A. Erroneous Sensor Data

Like any sensing system, those utilized by an aircraft may incorrectly measure the environment, and these sensors can present the erroneous data in a number of different ways. Here, the models for how these measurement errors manifest are discussed in mathematical terms. We use $\boldsymbol{x}_s(t)$ to denote the ground-truth airplane sensor data at time t, and $\overline{\boldsymbol{x}}_s(t)$ to denote the erroneous airplane sensor data at the same time t. Since we are considering the particular case of MCAS, we can further simplify $\boldsymbol{x}_s(t)$ to include the left and right AoA sensor readings. Our model incorporates Gaussian noise that is typically present in sensor measurements as part of $\boldsymbol{x}_s(t)$.

Our selection of sensor errors is representative of those errors that would occur in the real-world; we do not consider errors that are theoretically possible but have no known way of

occurring in the real world. Moreover, we do not consider any combination of the errors mentioned in this section. In practice, the more dominant error will have its effects impact the system, so we consider our evaluation of the failures in isolation.

1) Sudden Error. A *sudden error* is defined as

$$\overline{\boldsymbol{x}}_s(t) = \delta, \tag{3}$$

where δ is a constant value. This error is agnostic of the current sensor value, making it the most simple sensor error. The erroneous data from the AoA sensor in ET302 was due to a sudden error, which may occur due to a jam caused by a bird strike.

2) Delta Error. For a *delta error*, we again assume a constant value δ and characterize the error as

$$\overline{\boldsymbol{x}}_s(t) = \boldsymbol{x}_s(t) + \delta. \tag{4}$$

The delta error is simply an offset to the actual sensor value, in which the constant value δ is added to the existing sensor data. The erroneous data from the AoA sensor in JT610 was due to a delta error, akin to a miscalibrated sensor.

3) Gradual Error. The *gradual error* is the most sophisticated of the three, incorporating a function $f(t)$ as part of error. The gradual error is generalization of the delta error, replacing δ with $f(t)$:

$$\overline{\boldsymbol{x}}_s(t) = \boldsymbol{x}_s(t_0) + f(t). \tag{5}$$

Furthermore, in contrast to the delta error, the x-intercept of the function is replaced with the sensor value at the start of the error, t_0. While the function $f(t)$ can be any function, we choose a few standard functions: the linear ($f(t) = at$), quadratic ($f(t) = at^2 + bt$), and logarithmic ($f(t) = a\log(t)$) functions, where a and b are predefined coefficients. This error replicates how a drifting sensor failure may occur over time.

B. Dangerous Pilot Behavior

The scope of MCAS's authority for counteracting pilot control is exclusively within the aircraft's pitch axis in the downward direction. The pilot controls the pitch by either manually cranking the HS or moving the control column to adjust the elevator. See § II-B for a brief introduction to longitudinal flight dynamics and control of the aircraft.

For a pilot to provide dangerous control to the aircraft, we consider two avenues through which this may occur. First, the pilot could continuously pitch the aircraft up. S/he does it by pulling back the control column, which in turn commands a consistent input to δ_e in Eq. (2). Eventually, the aircraft's AoA will exceed $\sim 17^\circ$, causing the aircraft to stall and experience a significant decline in altitude before entering a nosedive.

Second, the aircraft may have a major failure that demands the pilot to respond quickly. During such an event, the FAA guidance states that a pilot should respond within 3s [3]. However, if the pilot were to take longer to recognize and respond to the occurrence of such an issue, they may risk losing control of the aircraft and cause an accident. In fact, FAA flight handbooks refer to reaction times of 4s as common [21]. A major failure could refer to a correct MCAS activation that requires the pilot to re-adjust the aircraft accordingly, or an incorrect MCAS activation that the pilot must counteract.

1) Modeling the Timing Constraint of Aircraft Recovery. To characterize whether a pilot is performing a timely control of an aircraft, we use

$$\tau = \tau_{sensing} + \tau_{action}, \tag{6}$$

and constrain the success of evading failure with

$$t_{start\ bad\ event} + \tau \leq \tau_{deadline}. \tag{7}$$

FAA mandates that a well-trained pilot's sensing of a major failure, $\tau_{sensing}$, should be < 3 s [3]. However, we cannot assume the pilot will always be able to have such quick sensing of a failure, especially if they are overwhelmed with other tasks/warnings to attend to. As previously mentioned, the FAA accepts that it is common for pilots to require >4s to react [21].

The time it takes a pilot to complete the aircraft recovery action from a perceived failure is modeled with

$$\tau_{action} = x_{stab\ offset} * \frac{RPD}{RPS} \tag{8}$$

where we model how long it would take for the pilot to move the HS $x_{stab\ offset}$ degrees. This movement is dictated by the number of rotations per second (RPS) the pilot can turn the hand crank and the number of rotations per degree (RPD) required to move the HS (which is 18 in a 737 [22]). We note that while the pilot in the physical world may apply a *variable* RPS, our investigation in § VI models a *constant* RPS during pilot response to MCAS events. After recovering from the event, the simulated pilot will stop rotating the hand crank.

The deadline of aircraft recovery, $\tau_{deadline}$, is calculated:

$$\tau_{deadline} = \min \begin{cases} t_{current} + \frac{h}{v(t+\tau)} \\ t_{last\ MCAS\ fire} + \tau_{MCAS\ c.d.} \end{cases} \tag{9}$$

In the first term, we determine whether the altitude, h, of the aircraft will reduce to 0 before recovery is completed. Its catastrophic nature makes it a *hard deadline*. We model the velocity trajectory of the aircraft, $v(t)$, using the velocity of a falling object with initial velocity v_0:

$$\frac{dv}{dt} = \frac{1}{m} \sum F(v); \quad v_0 < v_t \tag{10}$$

mandated by the vertical forces of drag ($D = \frac{1}{2}\rho A C_d v^2$), lift ($L = \frac{1}{2}\rho A C_l v^2$), gravity ($G = mg$), and thrust ($T$) [23]:

$$\sum F(v) = G + D\sin(c) - L\cos(c) - T\sin(c), \tag{11}$$

with climb angle c, and the terminal velocity:

$$v_t = \sqrt{\frac{T\sin(c) - G}{\frac{1}{2}\rho A C_d \sin(c) - \frac{1}{2}\rho A C_l \cos(c)}}. \tag{12}$$

obtained by setting $\frac{dv}{dt} = 0$, $v = v_t$ and solving for v_t. In other words, we find the velocity when it is no longer changing. Finally, we integrate $\frac{dv}{dt}$ on the interval $[v_0, v(t)]$ and find the velocity at time t [24]:

$$v(t) = v_t \tanh\left(\tanh^{-1}\left(\frac{v_0}{v_t}\right) - \frac{t(T\sin(c) - G)}{v_t m}\right). \tag{13}$$

The second term of Eq. (9) relates to the next MCAS activation. For MCAS_{old} and MCAS_{new}, it offsets the HS 2.5° with a cooldown of 11s if the AoA remains higher than 17°; otherwise, MCAS will not activate again. During the case of successive MCAS triggers, the pilot must achieve higher than $RPD * \frac{x_{stab\ offset}}{\tau_{MCAS\ c.d.}} = 18 * \frac{2.5}{11} = 4.09$ RPS of the trim wheel in order to counter the MCAS. Sustaining a high RPS for a period of multiple seconds is impractical due to the physical strain it would cause. Alternatively, during MCAS activation, the pilot may attempt to physically halt the trim wheel when MCAS activates, but this similarly requires the pilot to hold a large amount of weight for a period of time.

Fortunately, the HS cooldown time is not a *hard deadline*. The pilot failing to undo the HS displacement once will not lead to catastrophic failure; it is the result of missing it multiple times that leads to the catastrophic failure of crashing the aircraft. This makes the task more similar to an (m,k)-firm guarantee. When evaluating this aspect of the timing constraints, we assume RPD and $x_{stab\ offset}$ are static values built into the design of the aircraft and thus are not free to change. This assumption is consistent with the implementation of MCAS in the Boeing 737-MAX.

IV. Semi-Autonomous MCAS (SA-MCAS)

Following our preliminary analysis of the differences between MCAS$_{old}$ and MCAS$_{new}$ (Fig. 2), we propose Semi-Autonomous MCAS (SA-MCAS), an MCAS that does not give static authority to one control input over the other. Unlike Boeing's MCAS, SA-MCAS uses a *Synthetic Air Data System (SADS) arbiter* to cross-validate the sensor readings to first determine whether the pilot or autonomous control input is correct and then decide which of the two is allowed to control the pitch of the aircraft.

The SADS is a mechanism that was not originally employed in the 737-MAX, but it has appeared in advanced commercial aircraft such as the Boeing 787 [25] and UAVs [26]. It precisely estimates air data that the ADIRU also supplies, making it an additional source of redundancy. In the wake of JT610 and ET302 crashes, a U.S. congressional committee outlined clear evidence that the use of a SADS in the 737-MAX may have improved its safety and reliability [27], but the benefits of its inclusion have neither been shown empirically nor has Boeing added one to the 737-MAX.

Moreover, while SADS estimates a sensor's measurement without directly using that sensor, it may use other sensors' measurements that may also have measurement inaccuracies. This is a mature tool, so there are many ways for a SADS to estimate air data [28]. Our novelty is to add an arbiter that is responsible for choosing the synthetic data that will be used in the previously mentioned cross-validation step. We show that, indeed, SADS presents itself as a promising tool in combination with an arbiter.

We note that while MCAS$_{new}$ similarly uses a cross-validation check to ensure consistency between both AoA sensors, it fails to consider instances where both may be incorrect at the same time. Functionally, it only checks to see if the measurements of two AoA sensors are different from one another by $>5.5°$. Therefore, if both sensor measurements are incorrect, yet still similar, the failure is never noticed. For SA-MCAS, the incorporation of a SADS arbiter and its multiple independent estimations of the AoA measurements ensures that this issue will never arise.

There are a few strategies that can be used after the cross-validation step determines that the sensor measurements are incorrect. This stage of the process is extremely important: we cannot employ a strategy that allows the arbiter to become a new single point of failure. Instead of directly using the estimated measurement from the SADS, the arbiter can use the previous data that was determined correct temporarily. For instance, we can just use the previous sample directly or extrapolate the correct data using flight models. However, these strategies may have unpredictable outcomes without rigorous validation or if the measurements fail for an extended period of time. A more predictable strategy is to drop the measurements completely. In doing so, the arbiter no longer becomes a single point of failure; instead, the arbiter prevents MCAS from making any choice since it has no data to use.

Algorithm 1: SA-MCAS activation using the SADS arbiter technique.

```
Function do_activate_SA_MCAS():
    S_l, S_r ← ADIRU_left(), ADIRU_right()
    S_SADS ← SADS(S_l, S_r)
    S_correct ← arbiter(S_l, S_r, S_SADS)
    return is_stall(S_correct)

Function SADS(S_l, S_r):
    w ← model_free_wind_triangle(S_l, S_r)
    m ← model_flight_dynamics(S_l, S_r)
    S_SADS ← ∅
    /* Internal SADS consistency check.          */
    for s_w ∈ w, s_m ∈ m same sensor do
        if |s_w − s_m| < ε then
            S_SADS ← S_SADS ∪ s_w
    return S_SADS

Function arbiter(S_l, S_r, S_SADS):
    S_correct ← ∅
    /* External SADS consistency check.          */
    for s_l ∈ S_l, s_r ∈ S_r, s_SADS ∈ S_SADS same sensor do
        if |s_l − s_SADS| < ε then
            S_correct ← S_correct ∪ s_l
        else if |s_r − s_SADS| < ε then
            S_correct ← S_correct ∪ s_r
    return S_correct
```

Algorithm and Deployment. The full algorithm for SA-MCAS activation is presented in Alg. 1. To ensure SA-MCAS does not become a new single-point-of-failure, it is built with two layers of consistency checking.

First, there is an internal consistency check in `SADS()`, which follows the estimation of air data using the model-free [29] and flight dynamic model-based [28], [30] methods, `model_free_wind_triangles` and `model_flight_dynamics`, respectively. Because there is a diversity in estimation methods, there are duplicate estimations of the same air data measurements from these two methods. For example, the AoA (α) can be estimated with the model-free method $\alpha = \tan^{-1}(\frac{u}{v})$ and with the model-based method $\alpha = f(C_L, M, h)$. The goal of the internal consistency check is to ensure that potentially erroneous ADIRU sensors involved in the estimation methods are not impacting the final estimation. The input to `SADS()` includes the left ADIRU sensor data, $\boldsymbol{S}_l$, and the right ADIRU sensor data, $\boldsymbol{S}_r$. Its output is the set of estimated sensor data, $\boldsymbol{S}_{SADS}$.

On the other hand, determining which air data measurements are incorrect is left to the second, external consistency check, which occurs during the `arbiter()` step. If the `SADS()` estimate is ε distance away from the physical measurement of the left or right ADIRU, the measurement is passed on to the final step. If the difference exceeds ε, the measurement is dropped. In this work, the value of ε is selected based on the typical Gaussian noise of each sensor measurement. A tight bound is selected to err in favor of false-negatives since the effects are in the spirit of the design choices of Boeing aircraft. For more conservative bounds, we would recommend utilizing conformal prediction for creating a dynamic uncertainty quantification of the estimated sensor states [31]. The input to `arbiter()` includes the left and right ADIRU sensor data, $\boldsymbol{S}_l$ and $\boldsymbol{S}_r$, respectively, and the `SADS()` estimated sensor data, $\boldsymbol{S}_{SADS}$. The output is the set of sensor data that passes the external consistency check, $\boldsymbol{S}_{correct}$,

meaning the sensor data that is similar enough to the estimates.

In the final step of the algorithm, the air data passed on, $\boldsymbol{S}_{correct}$, is used to check whether a stall is occurring or whether the airplane is at risk of stalling. If it is, SA-MCAS activates control on the HS. Similar to Boeing's implementation of MCAS, SA-MCAS is deployed on the flight control computer of the 737-MAX. This was previously shown in Fig. 1; as seen in the figure, no modification to the communication architecture is necessary to incorporate SA-MCAS.

Challenges. This study is the first public exploration of the consequences of the designs of the MCAS's ability to recover under faults. As a result, during the development of SA-MCAS we encountered several challenges that lead to the primary contributions of this paper. We raise the following technical questions:

❶ How can we streamline the design and evaluation of MCAS without a physical aircraft? (§ V)
❷ Which control inputs from MCAS and the human pilot threaten the safety of the aircraft? (§ VI)
❸ Does SA-MCAS mitigate the issues present in MCAS$_{old}$ and MCAS$_{new}$, and does it satisfy the timing constraints for recovering the aircraft? (§ VII)

V. MCAS Simulation

This section addresses Challenge−❶. Using a real airplane as a testbed for evaluation of SA-MCAS is unrealistic/infeasible due to the high cost of purchasing the aircraft, renting or building a storage facility, and hiring pilots. Moreover, our analysis demands us to stress the limits of the aircraft and put it into hazardous situations that may ultimately crash it. Thus, the natural solution is to employ a widely-used flight simulation engine. Aerospace companies have custom flight simulators for testing their internal products, but they are usually unavailable to researchers. As a result, open-source flight simulators, such as JSBSim [12], are popular among academic researchers. In particular, JSBSim has been vetted by NASA, validating its accuracy of modeling real flight maneuvers [32]. The JSBSim flight simulator enables us to accurately model the Boeing 737-MAX's flight and control dynamics. Furthermore, because of the ease of modeling control loops in MATLAB Simulink, an integration of JSBSim to MATLAB was developed for this purpose [33]. However, its functionality was limited to just a few hard-coded control inputs and no account for pilot control or autonomous systems.

To overcome this inflexibility, we made an extension to the JSBSim Simulink module, which includes several user-definable features. Our extension enables the user to select any flight sensor input/output to/from JSBSim, provides a pilot simulation module with customizable scripts for controlling the aircraft, and an MCAS module for easy integration of new MCAS designs. Furthermore, switching between scripts and different MCAS designs is configurable before simulation execution, allowing for automated simulation runs without any additional manual effort. Where possible, we have merged features into JSBSim, while other features specific to our toolkit are provided as a separate GitHub repository (see Footnote 1).

In addition to these features, we provide a module for injecting sensor errors into the JSBSim sensor data. This module is capable of injecting the three different types of erroneous data in § III-A.

Maneuver	Performed in Crash	Ref.
Accelerate	•	N/A
Climb	•	[34]
Descend		[34]
Level-Turn		[35]
Climb-Turn		[34], [35]
Descend-Turn		[34], [35]
Holding Pattern		[36]
Takeoff	•	[34], [37]
Landing		[34]

TABLE I: Simulated flight maneuvers. We denote those that occur during the crashes of JT610 and ET302.

	Parameter	Range
Takeoff	Liftoff Indicated Airspeed (kts)	[160, 200]
	Transition to Cruise-Climb Altitude (ft)	[2000, 4000]
	Cruise-Climb Indicated Airspeed (kts)	[220, 300]
	Level-Off Altitude (ft)	[5000, 15000]
Landing	Initial Altitude (ft)	[1000, 5500]
	Descent Rate (ft/minute)	[600, 960]
	Final Approach Indicated Airspeed (kts)	[130, 150]

TABLE II: Parameter ranges used for validation of maneuvers.

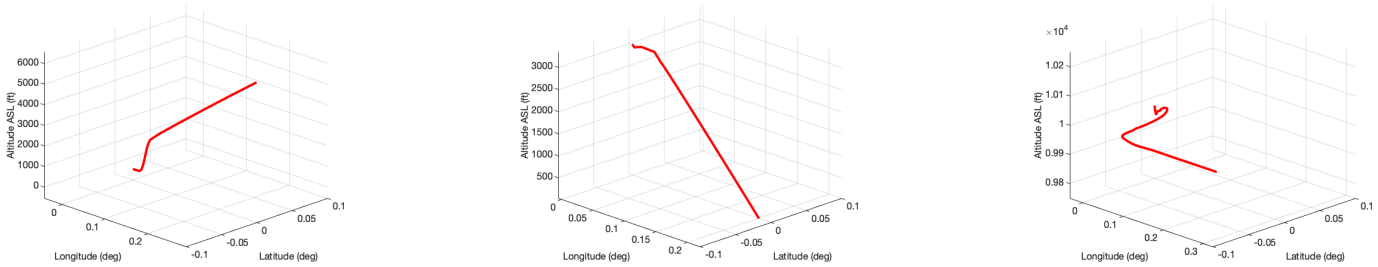
A. Simulation Creation Process

Next we describe how to create simulations in our toolkit.

Step 1: Initializing the input/output parameters. Before designing the rest of the simulation, the user must specify the air data they want provided throughout the simulation. This air data may be flexibly used and changed in other simulation components. To request the air data, the user provides an XML file with the simulator directory paths in which JSBSim stores each air data measurement. The user also initializes the erroneous sensor measurements with the time intervals they occur and the characteristics of the errors as defined in § III-A. This is to be defined within a JSON file.

Step 2: Building the MCAS module. We next integrate a specified MCAS design into the simulation. Using output parameters from the previous time step (which also may have been altered by the measurement error module), we define the specific conditions for MCAS activation behavior. For our implementations of Boeing's versions of MCAS, we refer to publicly available materials to develop a best-effort replica. For instance, we refer to [3] for MCAS$_{old}$ and [2] for MCAS$_{new}$. We additionally refer to sources such as [4], [18] to replicate finer details. Our implementation is without access or knowledge of proprietary information that has not been made available publicly.

Step 3: Scripting the pilot behavior. Finally, we provide several pilot flight maneuvers as part of the toolkit, such as takeoff, landing, and turning. To create a flight maneuver module, it takes the initialized sensor measurements as input and monitors them for user-defined conditions that trigger the next step for the maneuver. Since this is a simulated pilot, it lacks some of the finer feel and touch of a real pilot. However, we design these maneuvers using aircraft manuals that make suggestions for typical choices in flight (see the references in Tbl. I). While environmental conditions impact some aspects of each flight maneuver (such as the timing of a specific step), the steps pilots follow are usually the same. We validate our simulation of these flight maneuvers in the following subsection.



(a) Takeoff. (b) Landing. (c) Level turn.

Fig. 3: Simulated aircraft traces of typical Boeing 737-MAX maneuvers.

B. Example Simulation Scenarios

Internal states from the JSBSim flight simulator are fed into our pilot-emulation toolkit, which makes decisions and generates control commands based on the thus-provided states. This setup simulates the pilot controlling the plane towards a high-level goal while considering the flight conditions.

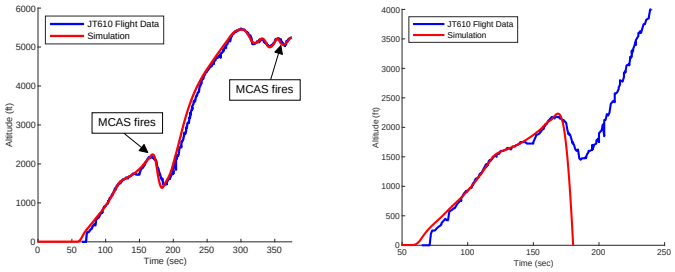
Several flight scenarios (Tbl. I) are used to verify that the overall simulation functions as expected. Takeoff and landing scenarios are targeted for closer study, as these were the scenarios in which real-world MCAS-related mishaps occurred. We demonstrate the trace of the flight path for a few of these scenarios in Fig. 3.

225 takeoff simulations and 100 landing simulations were conducted. Flight parameters were varied in each simulation (summarized in Tbl. II) to cover a broad range of possible takeoff and landing profiles. Following this process, we determine that our simulation of pilot behavior is sufficient for investigating dangerous flight scenarios. The flying traces are shown to accurately draw the path of the desired maneuver. Furthermore, we verify the effect of injected errors to the AoA sensor on the flight path, as seen in the preliminary study presented in Fig. 2.

C. Case Study: Simulation of JT610

In order to verify the accuracy of simulating flights impacted by incorrect sensor data *and* to understand how sensor failures impacted the real flight on JT610, we model the *delta error* that impacted the MCAS decision-making during JT610. The flight-data recorder (black box) for JT610 was successfully recovered by the Indonesian government, and while the raw data was never publicly released, detailed graphs of the data are available for analysis [17], [38]. We use the pilot simulation framework described in § V to model the decisions made by the pilots in JT610, following the same takeoff procedure. Likewise, we modeled the same AoA delta error that the left AoA sensor encountered, which had $\delta \approx 15^\circ$ for the entire flight from takeoff until crash. Before takeoff, the error in the left AoA sensor was more variable, but because it was before takeoff it did not impact the operation of the airplane. Thus, we do not model this in our simulation.

As mentioned before, the black box data from JT610 was never publicly released, but we were able to acquire the flight path of JT610 from Flightradar24 [39]. We overlay this recovered data with our simulation of JT610 in Fig. 4. The overlay on the simulation demonstrates the capability of our toolkit to accurately model MCAS misfires in the presence of incorrect sensor values. In Fig. 4a, the simulation is shown to closely overlap with the true flight path of JT610. Since we are capable of providing an accurate simulation of real piloting of aircraft experiencing sensor failures, we provide a deeper investigation of how our proposed error model from § III impacts the simulated aircraft in the following section.



(a) Pilot attempts to counter the MCAS misfires, similar to the actual flight.

(b) Pilot does not intervene at the onset of MCAS falsely firing for the first time.

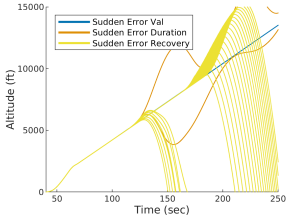
Fig. 4: Simulations of the flight JT610, alongside a delta injection with $\delta = 15^\circ$. The simulated flights are plotted against the flight path of JT610.

Before this deeper investigation, our case study reveals a more interesting pattern that warrants a closer inspection. The pilot of JT610 was capable of maintaining an altitude of ∼5250 ft. for ∼7 mins. If an immediate action was not taken by the pilot, JT610 would have entered a nosedive almost immediately (simulated in Fig. 4b). In fact, the pilot recovered the aircraft 21 times in a row before becoming overwhelmed and handing off responsibility to the co-pilot, who ultimately failed to recover the aircraft after the hand-off. Ultimately, the crash of JT610 was a combination of MCAS *repeatedly* activating and the pilot becoming too tired to manually fight against MCAS automatically trimming the HS. In other words, this case study underscores the pilot's capability of manually recovering an aircraft from *rare* false-positive activation of MCAS.

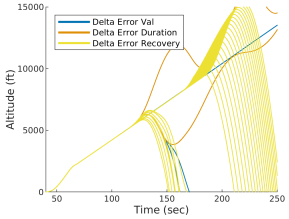
Conclusion for CHALLENGE−❶: *We provide an open-source MCAS toolkit built on the JSBSim flight simulator and MATLAB Simulink. We verify the correctness and usefulness of the simulations and include guidelines for using this toolkit.*

VI. STRESS TESTING BOEING MCAS

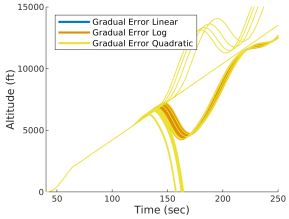
Using the erroneous sensor injection tool included as part of our simulation toolkit presented in § V, we try to cause failures on MCAS to reveal the precise conditions under which dangerous control of the aircraft is possible. Moreover, we have the pilot cause a stall in order to evaluate whether MCAS can mitigate the dangerous control. Such stress testing incorporates the error model from § III. Doing so leads to a conclusion for CHALLENGE−❷.



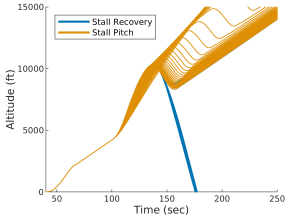
(a) Sudden error flight paths.



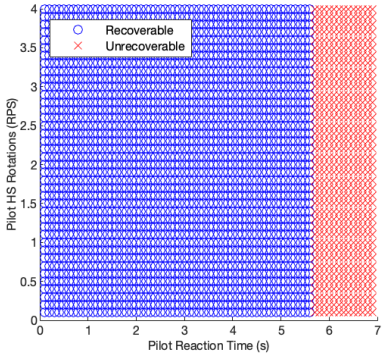
(b) Delta error flight paths.



(c) Gradual error flight paths.



(d) Pilot-induced stall flight paths.



(e) Pilot can have some variability in their response time and exerted effort on the HS hand-crank. This figure examines the impact of this variability on aircraft recovery. The recoverability of the flight *is not* dictated by the pilot's reaction speed and rotation of the HS.

Fig. 5: Summary of the stress test simulation for MCAS$_{old}$.

A. Methodology for Stress Tests

For the *sudden* and *delta errors*, we conduct three stress tests for each error: (1) $\delta \in [0,90]$ for time range $t \in [100,150]$, pilot reacts after 5s; (2) $\delta = 18°$ for time range $t \in [100, t_{end}]$, $t_{end} \in [110,180]$, pilot reacts after 5s; and (3) $\delta = 18°$ for time range $t \in [100,150]$, pilot reacts $\in [0,10]$s. For each stress test, we perform a parameter sweep using binary search in order to find the boundary for which the aircraft is no longer recoverable. The ranges for these test are chosen based on the physical limitations of the aircraft and pilot. For instance, while δ may theoretically be higher than 90°, the aircraft will never pitch high enough for this to be the case.

For the *gradual errors*, we conduct three experiments, one for each of linear, quadratic, and logarithmic functions. The controlled settings for these stres tests are, respectively, (1) $f(t) = at$ where $a \in [0,3]$ and pilot reaction after 5s; (2) $f(t) = a\log(t)$ where $a \in [0,500]$ and pilot reaction after 5s; and (3) $f(t) = at^2$ where $a \in [0,3]$ and pilot reaction after 5s.

The goal of these stress tests is to see whether MCAS will incorrectly activate (i.e., actives when no stall is occurring). When MCAS incorrectly activates, the pilot is providing a normalized elevator input of -0.1 and their reaction is to re-trim the HS at a rate of 3.5 RPS.

To incorporate *dangerous pilot behavior*, we perform two separate stress tests. (1) We simulate a pilot pitching up the aircraft $\in [20,90]°$ in order to cause a stall. 5s after the stall occurs (i.e., MCAS correctly activates), the pilot begins recovery of the aircraft, which follows a pitch-down of the aircraft to gain speed, then a pitch-up followed by trimming the HS. (2) We simulate a pilot's recovery time in order to test the timing aspect of the recovery. The pilot pitches up the aircraft 50° to cause a stall. After MCAS activates, we vary the recovery reaction time $\in [0,10]$s.

We provide further investigation into the impact that the variable pilot behavior may have on the timing analysis. Our analysis is shown in Fig. 5e. The x axis is the pilot reaction time, $\tau_{sensing}$, in the range of 0.1 to 7s. The y axis is the component of τ_{action} that is under the pilot's control, the RPS of the HS hand-crank, which is in the 0.1 to 4 RPS range.

Stress Test \ MCAS	MCAS$_{old}$	MCAS$_{new}$	SA-MCAS
Sudden Val	17°	No failure	No failure
Sudden Duration	140.5450s	No failure	No failure
Sudden Recovery	2.7991s	No failure	No failure
Delta Val	13.8750°	No failure	No failure
Delta Duration	140.5450s	No failure	No failure
Delta Recovery	2.7991s	No failure	No failure
Gradual Linear	1.5000	No failure	No failure
Gradual Log	222.5000	No failure	No failure
Gradual Quadratic	1.4999	No failure	No failure
Stall Pitch	51.5497°	46.2531°	51.5497°
Stall Recovery	5.6333s	3.9084s	5.6333s

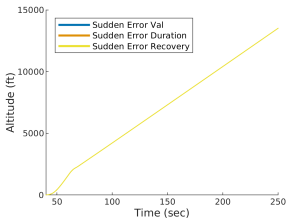
TABLE III: Summary table of the results from our stress test of each MCAS. Each row is associated with a particular stress test. Details on these tests are reported in § VI-A. Each entry is the lower bound for failure, and the higher numbers are better.

B. Stress Test of MCAS$_{old}$

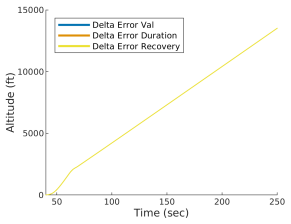
The summary of the results for the stress test of MCAS$_{old}$ may be found in Fig. 5 and the first column of Tbl. III.

1) Sudden & Delta Sensor Errors. For the variant of these tests that increase the measurement error incrementally, the plane was irrecoverable when the error is large enough to start triggering MCAS$_{old}$, i.e., the measurement error causes the AoA value to exceed 17°. This is primarily due to the simulated pilot responding too late. For the tests that stress the measurement error duration, the results are similar; after MCAS$_{old}$ activates a third time, the pilot is unable to recover the aircraft. This result is consistent with what was observed in the crashes of JT610 and ET302 — after the third activation, MCAS$_{old}$ has displaced the location of the HS substantially enough to make the aircraft irrecoverable. Finally, we observe that no matter the response time of the pilot, the flight cannot be recovered if a measurement error is sustained for a long enough period of time.

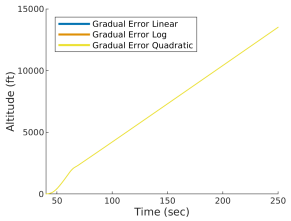
2) Gradual Sensor Errors. For the case of the gradual sensor measurement error, MCAS$_{old}$ is only capable of preventing a crash when the log function and the linear function are parameterized with $a < 222.5$ and $a < 1.5$, respectively. The quadratic case only has successful recovery when $a < 1.5$. This means that a moderately gradual drift in measurement error may incorrectly invoke MCAS$_{old}$ and cause the aircraft to be irrecoverable.



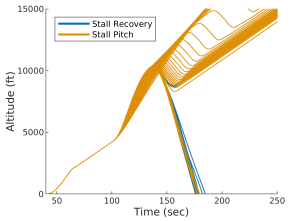
(a) Sudden error flight paths.



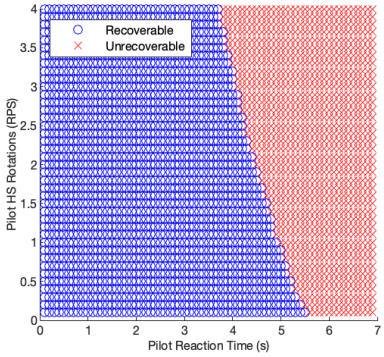
(b) Delta error flight paths.



(c) Gradual error flight paths.



(d) Pilot-induced stall flight paths.



(e) Pilot can have some variability in their response time and exerted effort on the HS hand-crank. This figure examines the impact of this variability on aircraft recovery. The recoverability of the flight *is* dictated by the pilot's reaction speed and rotation of the HS.

Fig. 6: Summary of the stress test simulation for MCAS$_{new}$.

3) Pilot Stalling. Our investigation into pilot stalling reveals that with the assistance of MCAS$_{old}$ the factor most important is the pilot's reaction time. When the pilot has a reaction time of 5s, all pitch angles from 20° to 50° have recoverable stall events. On the other hand, with a pitch angle of 50°, the stall event is recoverable when the pilot reacts in at most 5.63s. After 5.63s, none of the flights are recoverable.

4) Deadline & Timing Analysis. The stall recovery test demonstrates that an MCAS$_{old}$-assisted recovery of the stall is only possible when the pilot reacts within ∼5.63s. We investigate this further in Fig. 5e. We find that the pilot reaction time is the main contributor to aircraft recovery. Interestingly, it is inconsequential how much effort the pilot exerts toward recovery, i.e., the number of rotations per second on the hand-crank that adjusts the HS. Also, notably, the sudden and delta errors clearly are irrecoverable when the pilot reacts to the incorrect MCAS$_{old}$ activation too late (e.g., our set 5s for the experiments). If the pilot responds quickly, recovery is still possible. The pilot, therefore, has stricter reaction constraints for safe recovery of the aircraft when MCAS$_{old}$ falsely activates.

C. Stress Test of MCAS$_{new}$

The summary of the results for the stress test of MCAS$_{new}$ may be found in Fig. 6 and the second column of Tbl. III.

1) Sensor Errors. For MCAS$_{new}$, all flight paths are recovered for the sensor measurement error tests. This is consistent with Boeing's claims and is unsurprising since our stress tests only cause measurement error in one of the two AoA sensors. In other words, just comparing the difference between the two sensor measurements is sufficient. However, if both AoA sensors have a similar measurement error, MCAS$_{new}$ is not sufficient. Trivially, this is not detectable nor recoverable by MCAS$_{old}$ since it only uses one of the AoA sensors. In § VII we show how SA-MCAS is capable of recovery against this class of measurement error.

2) Pilot Stalling. Consistent with our initial investigation in Fig. 2, we find that because MCAS$_{new}$ is functionally only allowed to activate once, there are stall events during a high pitch-up that render the flight unrecoverable. With MCAS$_{old}$, these flights were recoverable because of its repeated assistance. Moreover, with the fixed pitch angle set to 50°, the stall event is only recoverable if the pilot responds in ∼ 3.9s. This is ∼31% decrease in the total time the pilot had previously to recover from a similar stall event with MCAS$_{old}$.

3) Deadline & Timing Analysis. For MCAS$_{new}$, the amount of time that the pilot has to respond to a stall event is reduced by nearly 2s. While there are clear gains from eliminating the pathway for these errors to occur, enforcing such restrictions on the MCAS activation places burdens on the pilot. We conduct an in-depth analysis of the pilot's influence toward recovery in Fig. 6e. The results are consistent with those we observe in Fig. 5e. It also demonstrates that the time the pilot can respond to recover the aircraft is consistently reduced.

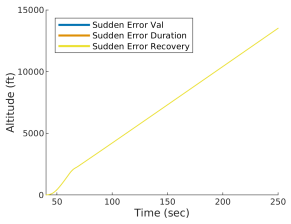
Conclusion for CHALLENGE−❷: *We demonstrate a series of control threats outside of those that caused the original 737-MAX crashes. We also demonstrate that the new Boeing MCAS design is susceptible to the newly identified control threats from the pilot. Our analysis unveils precise upper bounds for aircraft recoverability during erroneous MCAS events.*

VII. Evaluation of SA-MCAS

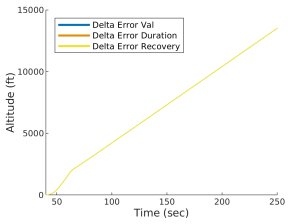
With an available simulator for streamlining the design and evaluation of MCAS programs (§ V) and well-defined failure scenarios (§ VI), we must consider a solution that merges the strengths of both MCAS$_{old}$ and MCAS$_{new}$. Unlike the prior versions of MCAS, our solution, SA-MCAS, does not incorporate a static assignment of control authority during control conflicts. Here, we evaluate our implementation of SA-MCAS. By doing so, we seek a conclusive answer to CHALLENGE−❸. The summary of the results for the stress test of SA-MCAS may be found in Fig. 7 and the third column of Tbl. III. SA-MCAS is implemented using Alg. 1, which was detailed in § IV.

A. Sensor Errors.

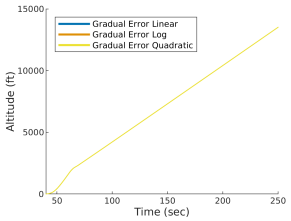
For sensor errors of any type (i.e., *sudden*, *delta*, or *gradual errors*), SA-MCAS provides the correct control of the aircraft's



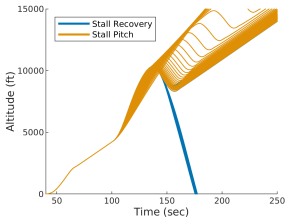
(a) Sudden error flight paths.



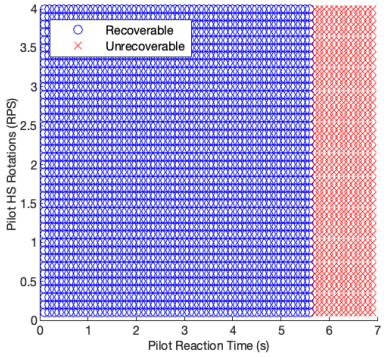
(b) Delta error flight paths.



(c) Gradual error flight paths.



(d) Pilot-induced stall flight paths.



(e) Pilot can have some variability in their response time and exerted effort on the HS hand-crank. This figure examines the impact of this variability on aircraft recovery. The recoverability of the flight *is not* dictated by the pilot's reaction speed and rotation of the HS.

Fig. 7: Summary of the stress test simulation for SA-MCAS.

HS in *every single modeled sensor measurement error scenario* that we outlined in § VI-A. Unlike MCAS$_{old}$, SA-MCAS is capable of preventing sensor measurement errors from causing dangerous MCAS control. Furthermore, because Alg. 1 has two layers of internal and external redundancy, SA-MCAS can prevent erroneous activation when both AoA sensors have measurement error. This was not possible with either MCAS$_{old}$ or MCAS$_{new}$.

B. Dangerous Pilot Behavior.

To evaluate SA-MCAS on dangerous pilot behavior, we similarly use our simulated pilot maneuvers from § VI-A. Comparing the results in Fig. 7e to MCAS$_{new}$, SA-MCAS does not reduce the amount of time the pilot has to respond in order to successfully recover the aircraft. Here, the results mirror the strengths of MCAS$_{old}$. These simulations demonstrate that SA-MCAS's ability of utilizing the strengths of the prior MCAS versions, and hence enabling an overall safer autonomous control system.

C. Deadline & Timing Analysis.

Reverting the decision to only allow MCAS to activate once, SA-MCAS returns to the stall-prevention capabilities of MCAS$_{old}$. Thus, the pilot has nearly 2s more time than MCAS$_{new}$ to react to a stall event with SA-MCAS, a $\sim$44% increase. The FAA guidelines say that the pilot should react to a major failure within 3s, but their flight training materials say that it is likely for a pilot to take even longer to respond. For instance, in the crashes of JT610 and ET302 the pilots were distracted and could not respond within 3s. The benefit of providing the pilot with more time to respond cannot be understated: in realistic scenarios for a safety-critical system it can be between life and death.

Conclusion for CHALLENGE–❸: *We present the evaluation of SA-MCAS, an MCAS that is capable of resolving control conflicts between the manual and automatic input. It is less susceptible to the previously identified control threats, increasing the upper bounds on the conditions for aircraft recoverability.*

VIII. DISCUSSION

Below we discuss limitations of our work and potential directions for addressing them. For the cases where iterative improvements to SA-MCAS may be available, we leave these as future work.

Prevention of Dangerous Pilot Input is Effective in a Limited Scope of Conditions. As we alluded to in § VII-B, there is a limited scope of conditions where SA-MCAS is incapable of recovering an aircraft from dangerous input. Generally, these simulations involve a pilot reaction time greater than 6s. In order to overcome this limitation, MCAS would require greater control authority, which the FAA would need to first approve. Given the constraint of how MCAS is currently allowed to operate, such a drawback may be acceptable. As mentioned in § VII-C, SA-MCAS improves the state-of-the-art by improving the maximum pilot reaction time by nearly 2s while avoiding sensor failures and being capable of handling multiple sensors failing at once.

Passenger Trust. In the aftermath of the Boeing 737-MAX crashes, passenger trust towards the 737-MAX aircraft has slowly recovered. The main contributor to this revival of trust has been from dropping MCAS as a tool that is capable of having authority to autonomously control the pitch of the aircraft. However, Boeing continues fall under scrutiny due to arising safety concerns for the 737-MAX. One drawback of our work is its assumption of regaining the trust of such passengers with a dynamic authority SA-MCAS. However, we note that the airline industry is not the only one facing this challenge—the autonomous vehicle industry has faced several controversies due to issues in the self-driving algorithms that lead to deadly accidents.

While restoring public trust in autonomous systems is outside the scope of this paper, we acknowledge the drawback that this issue presents to SA-MCAS. In order to regain this trust, we propose that a system such as SA-MCAS should be introduced in such a way that would (1) educate the pilot on its autonomous functions and limitations so the pilot will not over-trust MCAS, and (2) give the pilot the capability to disable SA-MCAS in the event that issues with the algorithm arise. Before ever being

put into the air, SA-MCAS should also go through hundreds of simulated flight hours with real pilots in order to establish trust with regulation agencies such as the FAA. While this may be seen as contradictory to our original motivation, it may still be a necessary measure. In fact, our argument is about the controller capabilities and conflict, e.g., with respect to MCAS itself. If the pilot chooses to disable MCAS altogether, then this is outside the scope of our goals.

IX. Conclusion

In this paper, we introduced SA-MCAS, a system for deciding who to trust when a human pilot and the autonomous MCAS module of the Boeing 737-MAX are in disagreement. Our analysis of the control risks of MCAS_{old} and MCAS_{new} shows the need for an MCAS that can make such dynamic control arbitration. We demonstrate SA-MCAS's capability of providing the correct control input in all cases of injected erroneous sensor values as well as many instances of dangerous pilot behavior, matching the best performance of both MCAS_{old} and MCAS_{new}. Our results suggest that it would be beneficial for the flight control computer of a Boeing 737-MAX to include a system like SA-MCAS. It would serve as an integrity checker in order to achieve the FAA's flight directive in [18].

References

[1] Federal Aviation Administration. (2019) Emergency Order of Prohibition. [Online]. Available: https://web.archive.org/web/20230417205843/https://www.faa.gov/news/updates/media/Emergency_Order.pdf

[2] Boeing. (2019) 737 MAX Software Update. [Online]. Available: https://www.boeing.com/commercial/737max/737-max-software-updates.page

[3] D. Gates and M. Baker, "The Inside Story of MCAS: How Boeing's 737 MAX System Gained Power and Lost Safeguards," *The Seattle Times*, 2019.

[4] P. Lemme. (2021) Mid-Value Select (MVS): Goldilocks in the House of MCAS. [Online]. Available: https://www.satcom.guru/2021/01/mid-value-select-mvs-goldilocks-in.html

[5] C.-Y. Chen, "Context-Aware Detection and Resolution of Data Anomalies for Semi-Autonomous Cyber-Physical Systems," Ph.D. dissertation, University of Michigan, 2022.

[6] L. Xue, Y. Liu, T. Li, K. Zhao, J. Li, L. Yu, X. Luo, Y. Zhou, and G. Gu, "SAID: State-Aware Defense Against Injection Attacks on In-Vehicle Network," in *USENIX Security Symposium*, 2022.

[7] F. Guo, Z. Wang, S. Du, H. Li, H. Zhu, Q. Pei, Z. Cao, and J. Zhao, "Detecting Vehicle Anomaly in the Edge via Sensor Consistency and Frequency Characteristic," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, 2019.

[8] N. T. Curran, A. Ganesan, M. D. Pesé, and K. G. Shin, "Using Phone Sensors to Augment Vehicle Reliability," in *IEEE Conference on Communications and Network Security (CNS)*, 2023.

[9] A. Ganesan, J. Rao, and K. G. Shin, "Exploiting Consistency Among Heterogeneous Sensors for Vehicle Anomaly Detection," SAE Technical Paper, Tech. Rep., 2017.

[10] K. Koscher, S. Checkoway, et al., "Experimental Analysis of a Modern Automobile," in *IEEE Symposium on Security & Privacy (S&P)*, 2010.

[11] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," in *Black Hat USA*, 2015.

[12] J. Berndt, "JSBSim: An Open Source Flight Dynamics Model in C++," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*, 2004.

[13] Boeing, "Boeing 737-600/-700/-800/-900 Operations Manual," 2018.

[14] J. P. How, "Lecture notes in 16.333: Aircraft Stability and Control," 2004.

[15] Aerospace Recommended Practice, "ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," SAE International, Standard, Dec. 1996.

[16] ——, "ARP 4754: Guidelines For Development Of Civil Aircraft and Systems," SAE International, Standard, Dec. 2010.

[17] S. Tjahjono, "Preliminary KNKT.18.10.35.04 Aircraft Accident Investigation Report," National Transportation Safety Committee of Indonesia, Tech. Rep., 2018. [Online]. Available: https://downloads.regulations.gov/FAA-2024-0159-0002/attachment_11.pdf

[18] Federal Aviation Administration, "Airworthiness Directives; The Boeing Company Airplanes," *Federal Register*, vol. 85, no. 225, pp. 74 560–74 593, 2020. [Online]. Available: https://www.federalregister.gov/d/2020-25844

[19] M. Technology. (2024) Trusted Platform Module: Complete Security for PCs and Embedded Systems. [Online]. Available: https://www.microchip.com/en-us/products/security/security-ics/tpm

[20] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "Formal verification of ACAS X, an industrial airborne collision avoidance system," in *International Conference on Embedded Software (EMSOFT)*, 2015.

[21] Federal Aviation Administration, "Airplane Flying Handbook Chapter 17," 2021. [Online]. Available: https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/airplane_handbook/media/19_afh_ch17.pdf

[22] Bianfable. (2019) How many turns of a 737 trim wheel equal 2.5 degrees stabilizer deflection? [Online]. Available: https://aviation.stackexchange.com/questions/70184/how-many-turns-of-a-737-trim-wheel-equal-2-5-degrees-stabilizer-deflection

[23] NASA. (2023) Forces in a Climb. [Online]. Available: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/forces-in-a-climb/

[24] Wikipedia. (2024) Drag (physics): Velocity of a falling object. [Online]. Available: https://en.wikipedia.org/wiki/Drag_(physics)#Velocity_of_a_falling_object

[25] T. Dodt, "Introducing the 787: Effect on Major Investigations and Interesting Tidbits," 2011.

[26] K. Sun, "Reliable Air Data Solutions For Small Unmanned Aircraft Systems," Ph.D. dissertation, University of Minnesota, 2020.

[27] P. A. Defazio and R. Larson. (2020) Final Committee Report: The Design, Development, and Certification of the Boeing 737 MAX. [Online]. Available: https://web.archive.org/web/20221207103614/https://transportation.house.gov/imo/media/doc/2020.09.15%20FINAL%20737%20MAX%20Report%20for%20Public%20Release.pdf

[28] F. A. P. Lie and D. Gebre-Egziabher, "Synthetic Air Data System," *Journal of Aircraft*, vol. 50, no. 4, pp. 1234–1249, 2013.

[29] V. Klein and E. A. Morelli, "Aircraft System Identification: Theory and Practice," *AIAA Education Series*, vol. 213, 2006.

[30] J. E. Zeis, "Angle of Attack and Slideslip Estimation Using an Inertial Reference Platform," Master's thesis, Airforce Institute of Technology, 1988.

[31] S. Yang, G. J. Pappas, R. Mangharam, and L. Lindemann, "Safe Perception-Based Control under Stochastic Sensor Uncertainty using Conformal Prediction," in *IEEE Conference on Decision and Control (CDC)*, 2023.

[32] E. B. Jackson, et al., "Further Development of Verification Check-Cases for Six-Degree-of-Freedom Flight Vehicle Simulations," in *AIAA Modeling and Simulation Technologies Conference*, 2015.

[33] T. Sikström, "Flight Simulator Integration in Test Rig," Master's thesis, Royal Institute of Technology, 2021.

[34] M. Křepelka. (2022) Boeing 737–800 Flight Notes. [Online]. Available: https://krepelka.com/fsweb/learningcenter/aircraft/flightnotesboeing737-800.htm

[35] Badmachine. (2010) Built-In Bank Angle Limits. [Online]. Available: https://www.pprune.org/tech-log/416502-built-bank-angle-limits.html

[36] Federal Aviation Administration. (2022) ENR 1.5 Holding, Approach, and Departure Procedures. [Online]. Available: https://www.faa.gov/air_traffic/publications/atpubs/aip_html/part2_enr_section_1.5.html

[37] I. D. Williams. (2021) Boeing 737-800 Takeoff Procedure (simplified). [Online]. Available: https://www.flaps2approach.com/journal/2014/8/4/boeing-737-800-takeoff-procedure-simplified.html

[38] National Transportation Safety Committee of Indonesia, "Accident Boeing 737 MAX 8, PK-LQP (LNI610)," 2018. [Online]. Available: https://ngamotu.nz/images/20181122-jt610-knkt.pdf

[39] I. Petchenik. (2018) JT610 Granular ADS-B Data. Flightradar24. [Online]. Available: https://www.flightradar24.com/blog/wp-content/uploads/2018/10/JT610_Granular_ADSB_Data.csv