Context-Aware Input Selection Using Operational Design Domain

Christoph Torens*, Sebastian Schirmer†, Umut Durak‡,
German Aerospace Center (DLR), Institute of Flight Systems, Braunschweig, Germany

The Operational Design Domain (ODD) is an important concept for the safety of systems that utilize machine-learning (ML) components. It defines the operating conditions under which the ML component is expected to function correctly. This allows the system to monitor the component's ODD satisfaction and activate or deactivate the ML component accordingly. Similar concepts exist for other system components such as sensors and controllers, which also have defined conditions or operational envelopes within which they can safely operate. When multiple data producers such as sensors or ML components provide the same type of input but have different ODDs that may only partially overlap, a new challenge arises. During operation, the system and its environment may satisfy none, one, or several of these ODDs. However, it may not be allowed or safe to operate all data producers simultaneously. For instance, consider a set of ML components where each was trained to detect persons in camera images on training data at different flight altitudes. During operation, the system must pick the ML component that best fits the current conditions. In this paper, we formalize ODDs and present a method to infer an automatic selection mechanism that guarantees the system to choose a data producer that satisfies its ODD in the next step for all possible system executions. Therefore, the system will always receive valid inputs if they exist while minimizing the number of active data producers. We then discuss the automatic selection for an automatic landing where the selection is used to switch between different sensors and also between different specialized ML components. The main benefit of the selection mechanism is that it enables safe switching between data producers based on their ODDs, keeping the system within resource limits when not all data producers can be active at the same time, and improving overall resource consumption by only activating only the necessary data producers.

I. Introduction

Capturing the conditions under which a sensor, controller, or machine learning component can be trusted is essential to guaranteeing system safety. To this end, the European Union Aviation Safety Agency (EASA) introduced the notion of an operational design domain (ODD) for machine-learning (ML) components. An ODD refers to specific environmental and system conditions under which a component is designed to operate safely and effectively [1]. For example, consider an ML component that detects persons on the ground in camera images to support safe landing of a VTOL unmanned aircraft. If this ML component was trained only on training data using low altitude aerial images, then its ODD must be restricted to low altitude operations, as it cannot be assumed to generalize to detect persons in camera images at high altitudes. Vice versa, an ML component trained on high altitude aerial images should not be used for low altitudes. Further, there is no straight-forward way to compose or "fuse" two ML models into a single model that reliably handles low- and high altitude aerial images. If such a new ML component that is trained on both low altitude and high altitude is required, the certification process with its W-shaped development cycle must be repeated. While certain parts of the development cycle may be reused such as the data management, more demanding steps such as the learning assurance must be revisited.

In this paper, we consider an alternative idea: instead of developing a new ML component, we retain the two existing ones and derive an *automatic selection mechanism* based on their ODDs, even when the two ODDs only partially overlap. The overall architecture in which the automatic selection mechanism operates is given in Figure 1. In this architecture we refer to the components providing data to the automatic selection as *data producers*. In our example, the low altitude and high altitude ML component are the data producers. Each data producer is associated with an ODD,

*Research Scientist, Department Unmanned Aircraft, AIAA Associate Fellow.
†Research Scientist, Department Unmanned Aircraft.
‡Department Lead, Safety Critical Systems, AIAA Associate Fellow.

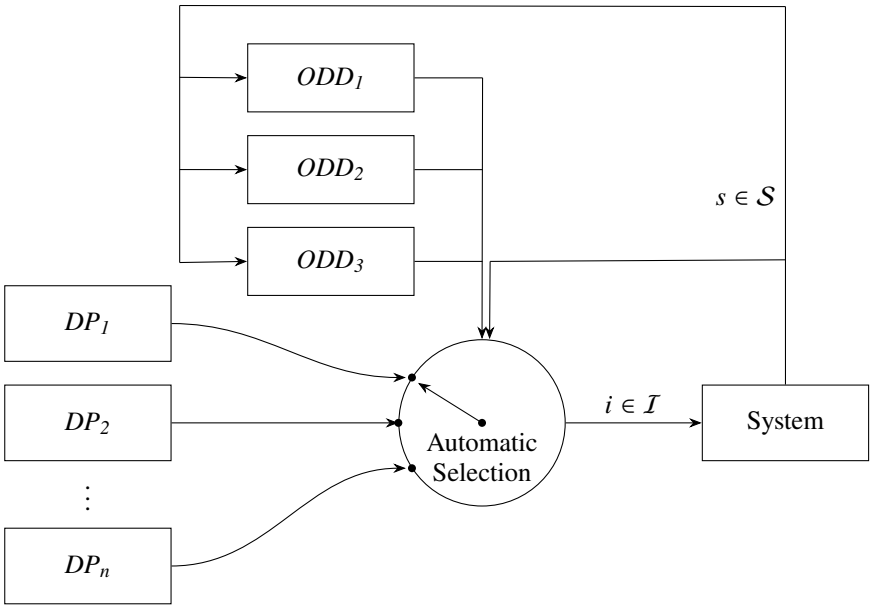

Fig. 1 Overview of the architecture showing multiple *data producers*(DPs) that forward their outputs to the *automatic switch*. The switch selects one of them based on information provided in each DP's operational design domain (ODD) and forwards it to the system. The ODDs evaluate their satisfaction based on the system's state s.

whose satisfaction depends on the current state s of the system, which may include internal system variables as well as sensor readings that capture environmental conditions, e.g., brightness.

The automatic selection receives (1) all outputs of the data producers, (2) their corresponding ODD satisfaction, and (3) the system's state, and selects the (minimal) set of data producers that must be activated to guarantee that at least one ODD will be satisfied at the next execution step using a transition system. This ensures that along the entire system execution, if at least one active DP exists whose ODD is satisfied, an output i that can be trusted is forwarded to the system. To provide this guarantee, we define the notion of an ODD, the ODD satisfaction along a system run, a safe selection, and the final output forwarding in cases where multiple data producers must be active due to ODD branching in the next system state. We provide two selection functions: sel is a conservative selection that activates all possible data producers that might be required in the next step and sel_o is an optimized selection that accounts for overlapping ODDs and activates only a sufficient subset of data producers. Note that originally, the concept of an ODD has only been discussed in the context of an aviation system utilizing an ML component. In this paper, we argue that the concept of an ODD is a general and universal tool for formalizing operational conditions for sensors, controllers, and most other components that have operational limits. To demonstrate this broader view, we present the use-case of an automatic landing of a VTOL unmanned aircraft where we used the selection to pick between ML components and also to pick between sensors.

A. Related Work

In the automotive domain, the concept of ODD was originally introduced and has since been formalized in several standards, including taxonomy and specification documents [2–4]. A formal ODD definition that enables the learning of ODDs from simulation runs was presented in [5]. In [6], another formal definition was provided that was automotive-centric and targeted towards operational monitoring and the identification of situations outside the intended domain. More recently, a comprehensive and consolidated standardization effort ended in the ASAM OpenODD framework [7]. This document provides both structured and textual specification formats and is expected to serve as a unifying reference for automotive-wide ODD descriptions.

In aviation, the concept of ODD has been adapted and was first formally introduced by EASA in [1], where it is applied to the assessment and certification of ML components. Within this guidance document, the ODD captures the operational constraints and environmental assumptions under which an ML component is expected to operate safely. Therefore, monitoring the ODD can be seen as a safeguard for the operation of an aircraft running ML components. In

[8], the authors presented an approach on how to design a data set compliant with an ML-based system ODD.

Many works invested ODD monitoring [9–12] as it is a crucial step for maintaining the safety and reliability of ML-enabled systems. ODD monitoring detects situation in which a machine-learning component may no longer be trustworthy, enabling fallback strategies or human intervention.

This paper builds on previous work in monitoring by employing *ODD* monitors as shown in Figure 1. However, in contrast to existing approaches, our work addresses the case where multiple partially overlapping ODDs are involved. Rather than monitoring for single ODD violations, we consider the problem of automatically selecting among multiple data producers and forwarding the correct data, if such exists, to the system. The proposed mechanism minimizes the number of active data producers while ensuring that all necessary producers are active. In this way, the automatic switch ensures that the system remains fully operational, yet activates only as many producers as are required at any given time, thereby adhering to resource constraints and even reducing overall resource consumption.

II. Preliminaries

In this section, we recap the notion of a transition system that is a common model in computer science to describe the behavior of systems. Our definition is similar to the one in [13]. However, instead of using actions, we use inputs and we do not include a labeling function that maps states to atomic propositions. Rather, we treat the states themselves as the relevant propositions. For example, a state $s = (v = slow, a = low)$ encodes the system's speed (v in $\frac{m}{s}$) and altitude (a in meters) where $slow := 0 \leq v \leq 10$ and $low := 0 \leq a \leq 10$. This allows to check constraints such as "if the system is flying below five meters" simply by evaluating the condition on the state itself. The example state $s = (v = slow, a = low)$ satisfies this constraint.

Definition 1 (Guarded Transition System) *A guarded transition system GTS is a tuple* $(\mathcal{S}, S_0, \mathcal{I}, \mathcal{G}, \mathcal{T})$ *where*

- $\mathcal{S}$ *is the set of states of the transition system,*
- $S_0 \subseteq \mathcal{S}$ *is a set of initial states,*
- $\mathcal{I}$ *is the set of inputs,*
- $\mathcal{G}$ *is the set of guard predicates,*
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{I} \times \mathcal{G} \times \mathcal{S}$ *is the guarded transition relation.*

Intuitively, the behavior starts in some initial state $s_0 \in S_0$ and evolves upon receiving inputs according to the guarded transition relation. That is, if s is the current state and input a is received, then the successor state s' is chosen according to $(s, a, g, s') \in \mathcal{T}$ where $g(a)$ must be true. This procedure is repeated in state s' and only finishes once a state is encountered that has no outgoing transitions. Formally, the behavior of a transition system is a finite or infinite *run* π with $\pi = s_0 \xrightarrow{g_0(a_0)} s_1 \xrightarrow{g_1(a_1)} s_2 \rightarrow \dots$ such that $(s_i, a_i, g_i, s_{i+1}) \in \mathcal{T}$ for all $i \geq 0$ where each $s_i \in \mathcal{S}$ and each $a_i \in \mathcal{I}$. For convenience, if the guards and inputs along the run are not relevant, we simply write $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$.

As an example, consider the transition system given in Figure 2 where each state consists of speed and altitude as before, and the guards are depicted on the transitions with the inputs in the middle. Each state's subscript indicates the speed category (slow < medium < fast) and its superscript denotes the altitude category (low < high). The initial state is s^{low}_{slow}. If the speed increases such that v satisfies the guard $10 \leq v \leq 20$, the system transitions to s^{low}_{med}. Then, when the altitude increase such that it satisfies the guard $10 \leq a \leq 20$, it transitions to s^{high}_{med}. The corresponding finite run is therefore: $s^{low}_{slow} \rightarrow s^{low}_{med} \rightarrow s^{high}_{med}$. Note that Figure 2 is highly simplified and chosen only for illustrative purposes. In particular, the GTS does not allow to process changes in speed and altitude simultaneous, nor does it allow large accelerations that could enable transition, e.g., from s^{low}_{slow} to s^{low}_{high}. In general, we assume that a GTS allows infinite runs, i.e., inputs can always be processed, and a run only stops when the execution is terminated.

III. Automatic Selection

In this section, we present the automatic selection of *data producers* (DPs). A DP is a component that produces inputs for the system, i.e., an input $i \in \mathcal{I}$. For brevity, we omit the inputs to the DPs. A typical example for a DP is a sensor such as a LiDAR altimeter. The concept also extends to others such as machine-learning networks whose outputs such as bounding boxes are used by others. In the following, we assume that DPs can be switched on or off instantaneously at discrete steps.

A DP operates under *operational constraints* that specify the conditions under which its outputs are valid. For example, a LiDAR altimeter has a limited range based on factors such as signal strength. During a landing approach, a downward-facing LiDAR should therefore only be used when the landing site is within its valid operating range.

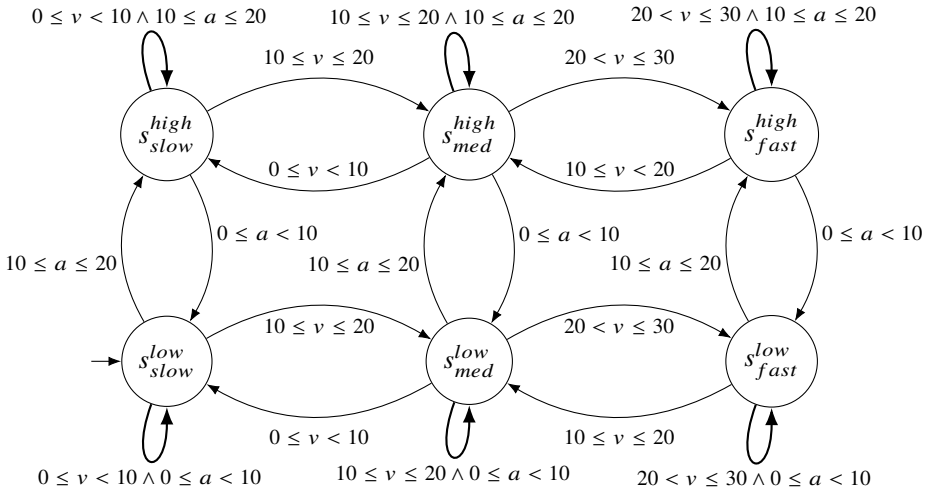

Fig. 2 Example of a guarded transition system with states defined by speed (slow < medium < fast) and altitude (low < high). The guards and inputs of the relation are depicted on the transition. The initial state is s^{low}_{slow}.

Definition 2 (Operational Constraint) *An Operational Constraint C_{OP} is a predicate. Formally, $C_{OP} : \mathcal{S} \to \mathbb{B}$ evaluates to true if the system is allowed to operate and false otherwise.*

An Operational Constraint that ensures the operating range of a LiDAR altimeter is *altitude* < 15 *meters*, which requires the aircraft to be below 15 meters for the predicate to be satisfied. All operational constraints for a DP are collected in its Operational Design Domain (ODD).

Definition 3 (Satisfaction of the Operational Design Domain) *An Operational Design Domain (ODD) is a set of operational constraints. We say that a state $s \in S$ satisfies an ODD O, denoted $s \models O$, if and only if every constraint $c \in O$ evaluates to true in state s, i.e., $s \models O \iff \forall c \in O.\, c(s) = true$. Similarly, a run $\pi = s_0 \to s_1 \to s_2 \to \ldots$ satisfies an ODD O, denoted $\pi \models O$, if and only if every state of the run satisfies O, i.e., $\pi \models O \iff \forall i \geq 0. s_i \models O$.*

As an example consider an ODD o for the LiDAR altimeter that consists of *altitude* < 15 (in meters) and *speed* < *10* (in $\frac{m}{s}$). Let a state consist of a tuple where the first value represents the altitude and the second the speed, e.g., $s_0 = (10, 5)$ represents that at state s_0 the *altitude* is ten meters and the speed is five $\frac{m}{s}$. It follows that $s_0 \models o$. If the system further produces $s_1 = (12, 6)$, $s_2 = (14, 8)$, and $s_3 = (15, 9)$, then the run π that visits s_0, s_1, s_2, and s_3 does not satisfy o, i.e., $\pi \not\models o$, because in state s_3 the altitude violates the operational altitude constraint.

Next, we consider the case where multiple DPs provide the same type of input. Each DP has its own ODD, which may differ and partially overlap with the other ODDs. We assume that the operational constraints of each ODD are defined over the same set of parameters, which makes them directly comparable.

As an example, consider the three DPs: LiDAR altimeter, GPS-based altimeter, and a barometric altimeter. Lets say the parameters of the ODDs are *altitude* (in meters), number of satellites in range (*num_sat*), and rate of pressure change (*pressure_change*, in hPA/s). Let the ODD of the LiDAR altimeter be $ODD_{LiDAR} = \{0 \le altitude \le 30, num\_sat \ge 0, pressure\_change \ge 0\}$, the ODD of the GPS-based altimeter be $ODD_{GPS} = \{15 \le altitude \le 1000, num\_sat \ge 0, pressure\_change \ge 0\}$, and the ODD of the barometric altimeter be $ODD_{Baro} = \{800 \le altitude \le 1000, num\_sat \ge 0, pressure\_change \ge 2\}$. The ODDs partially overlap in their parameter ranges. For an unmanned aircraft that operates using these altimeters, the system should *switch* between the sensors. For instance, for low altitude it should use the LiDAR altimeter, for medium altitudes it should rely on the GPS-based altimeter, and for high altitude it should use the Barometric altimeter. When flying between 15 to 30 meters, the system is free to chose which sensor to use or add some prioritization scheme that helps to pick. This redundancy is not always given, as for example when flying above 800 meters. Failures in such cases are beyond this work, typically they are handled by contingency or emergency procedures.

In the following, we present an automatic selection system that forwards exactly one of the DPs outputs. The system ensures that only outputs whose corresponding ODD is satisfied are forwarded and it requires no additional information beyond the ODD specification. This is different to sensor fusion that combines the outputs of multiple DPs into a

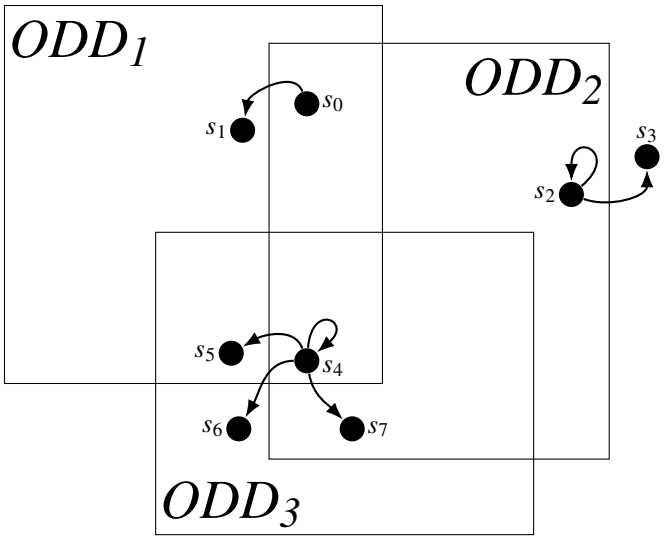

Fig. 3 Three different *ODD*s are depicted, showing some example state and their transitions of the underlying FSM. The state s_0 can transition to s_1 where only ODD_1 is satisfied. The state s_2 has an unsafe transition to s_3 after which no *ODD* is satisfied. The state s_4 has multiple transitions to all *ODD*s. What all these successor states s_5, s_6, and s_7 have in common is that they satisfy ODD_3.

new artificial output using probabilistic models, covariance information, or assumptions about noise characteristics to *improve* the output. Instead, our approach does not improve the selected output but guarantees to pick a valid one w.r.t. the ODDs. Formally, let $\mathbb{O}$ be the set of all ODDs and let $\mathbb{O}_a \subseteq \mathbb{O}$ be the set of ODDs that correspond to the currently active DPs. Given the run $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$, our objective is to activate a set of DPs such that, for every state along the run, at least one DP with a satisfied ODD is active, if such a DP exists. To realize this at runtime, we define a *selection* function $sel : S \rightarrow 2^{\mathbb{O}}$ which, given the current state $s \in S$, returns $\mathbb{O}'_a$, i.e., the set of active ODDs for the next state. The set O' is obtained by first considering all possible successor states s' such that $(s, \_, s') \in T$, where _ denotes any possible input, and returning exactly those ODDs $o \in \mathbb{O}$ for which a state s' satisfies the ODD o, i.e., $sel(s) = \{o \in \mathbb{O} \mid \exists s' \in S.(s, \_, s') \in T \wedge s' \models o\} = \mathbb{O}'_a$.

Definition 4 (Safe Selection) *A selection of DPs along a run $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ is safe if and only if $\forall i \geq 0.\ \exists O_i \in sel(s_i).\ s_i \models O_i$*

As an example consider Figure 3, which illustrates how the selection function chooses the active DP in three scenarios. Note that only the relevant states are depicted in the figure and that states that are within an ODD satisfy it. The first scenario begins in state s_0. Given the successor states s_1 which satisfies ODD_1, the *sel*-function returns $\{ODD_1\}$, resulting in activating the corresponding IP. The second scenario begins in state s_4. In this case, the *sel*-function returns $\{ODD_1, ODD_2, ODD_3\}$. Therefore, all DPs are active to guarantee that a valid output can be forwarded to the system. As a counter-example, assume that *sel* returns only $\{ODD_1, ODD_2\}$ but the system transitions to state s_6 in the next state. Since s_6 violates ODD_1 and ODD_2 both outputs of the corresponding DP are invalid and the only DP that would be valid did not produce an output. The last scenario begins in state s_2. Here, *sel* returns only $\{ODD_2\}$ due to the self-loop transition. Note that the transition to s_3 is *unsafe* as no DP exists for which its output is valid. This can be avoid by adding constraints that remove s_3 or by adding a DP that covers s_3.

Definition 5 (DP Output Forwarding) *Let the selection function sel that returns $\mathbb{O}_a$ at time i, that guarantees that for any successor state s' at least one DP has a satisfying ODD. At time $i + 1$, the output of the DP is forwarded to the system whose ODD $O \in \mathbb{O}_a$ is satisfied by the current state. If multiple such DPs exist, one can either select any of them arbitrarily or apply a predefined prioritization scheme.*

Next, we optimize the selection. Note that for an initial state such as s_4 in Figure 3, it is sufficient to activate only ODD_3 as it covers all next states, but that *sel* did return $\{ODD_1, ODD_2, ODD_3\}$. In Algorithm 1, we define an optimized selection function sel_o that iteratively computes the minimal set of required OOD. The algorithm iterates

over all successor states given the current state s (Line 2). It initializes a set that captures all variants of ODD combinations to satisfy all seen successor states (Line 1) and computes all ODD options that satisfy the successor state (Line 3). It then consists of two stages. The first stage (Lines 4 to 7) computes all combinations for ODDs. The second stage (Lines 8 to 9) minimizes these combinations using the current lowest cardinality within the variants. In the last line, we pick one variant of the minimized variants as $\mathbb{O}_a$. Using sel_o on the initial set s_4 in Figure 3 returns only $\{ODD_3\}$. Let s_5, s_4, s_7, s_6 be the order used by the for loop. After the first stage, the set *Variants* is $\{\{ODD_1\}, \{ODD_3\}\}$ and after the second stage the set remains unchanged. Next, for state s_4 the *Variants* set is $\{\{ODD_1\}, \{ODD_1, ODD_2\}, \{ODD_1, ODD_3\}, \{ODD_3, ODD_2\}, \{ODD_3\}\}$ after the first stage and $\{\{ODD_1\}, \{ODD_3\}\}$ after the second stage. Similar, happens for state s_7 and s_6, i.e., the set expands but reduces to $\{\{ODD_1\}, \{ODD_3\}\}$ for state s_7 and to $\{\{ODD_3\}\}$ for s_6 afterwards.

Algorithm 1: Optimized selection function sel_o

Input: Current state s, transition system $TS = (S, s_0, \mathcal{I}, T)$
Output: Minimal $\mathbb{O}_a$

1 $Variants \leftarrow \emptyset$ ▷ Used to store the different ODD combinations
2 **foreach** s' *with* $(s, s') \in T$ **do**
3 $O_{s'} \leftarrow \{o \in \mathbb{O} \mid s' \models o\}$ ▷ ODD options for state s'
4 **if** $Variants = \emptyset$ **then**
5 $Variants \leftarrow O_{s'}$
6 **else**
7 $Variants \leftarrow Variants \times O_{s'}$ ▷ Cartesian product of both sets
8 $c \leftarrow \min_{v \in Variants} |v|$ ▷ Captures the smallest set size
9 $Variants \leftarrow \{v \in Variants \mid |v| = c\}$ ▷ Only the minimal sets that satisfy all seen successor states remain
10 **return** $\mathbb{O}'_a \in Variants$

Proposition 1 *For any safe selection, it is necessary that up to n-DPs can be activated simultaneously, where* $n = \max_{s \in S} |sel_o(s)|$ *is the maximum number of ODDs that the optimized selection function may return on any state.*

Figure 3 shows an example where $n = 1$. Note that removing ODD_3 results in $n = 2$ as s_5 requires the other remaining *ODDs* to be active.

IV. Automated Landing of a VTOL Unmanned Aircraft

As a use-case, we consider an automatic landing for an air taxi or cargo drone. Figure 4 depicts the different flight phases. In (1), the unmanned aircraft is in the takeoff-phase where it ascends. Then, it transition to (2) where it picks up speed. After reaching a certain altitude, the cruise phase (3) begins. When this phase ends, the aircraft decelerates and descends (4) and eventually starts with the landing phase (5). The different phases differ in their altitude and speed. The takeoff phase brings the drone from an attitude of zero meters to an altitude of 20 meters. The ascend phase will go from 20 meters to 100 meters. Cruise flight takes place between 100 and 1000 meters. The descent phase descends from 100 and 20 meters. Finally, the landing brings the drone back to zero meters. Next, we show how the developed automatic selection mechanism picks the correct sensors and ML components for each phase of the automatic landing.

A. Selection of Sensors

We assume our aircraft has an LiDAR altimeter as well as a GPS-based altimeter. To avoid a running LiDAR that cannot be used but drains battery, the system will not use the sensor data outside of their ODD. The LiDAR provides high accuracy measurements, better than GPS, but it works only when the altitude over ground is below 30 meters. This is captures in the LiDAR's ODD (ODD_{LiDAR}) and the GPS's ODD (ODD_{GPS}):

$ODD_{LiDAR} = \{0m <= altitude <= 30m, \Delta(angle) \leq 5°, speed \leq 15\frac{m}{s}, battery > 40\%\}$

$ODD_{GPS} = \{15m <= altitude <= 1000m, \Delta(angle) \leq 5°, speed \leq 10\frac{m}{s}, battery > 20\%\}$.

Both ODDs requires that the aircraft is stable using $\Delta(angle)$. The LiDAR allows low altitude and faster flights, while the GPS allows high altitude flights. As the LiDAR drains more battery, it requires a higher level of remaining battery. Note that the altitudes of the ODDs overlap.

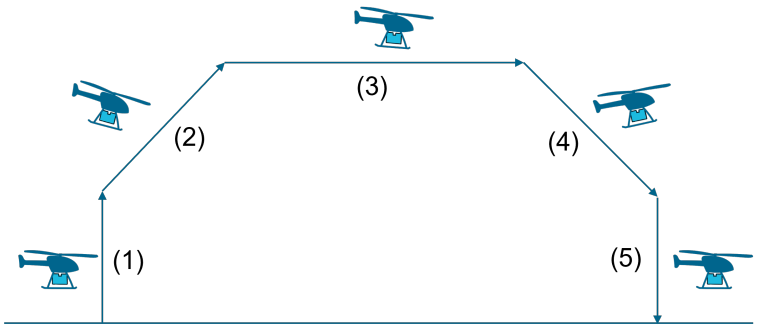

Fig. 4 The five flight phases of an air taxi or cargo drone are shown: take-off (1), ascent (2), cruise-flight (3), descent (4), and landing (5).

Focusing on the altitude, the selection process is depicted in Table 1. The selection analyzes the possible successor states defined by the guarded transition system. For simplicity, we assume a GTS that switches to the next state (represented by its altitude) in steps of five meters with zero being the lowest altitude, i.e., our initial state is "0" and its successor is "5". The column "Satisfied ODDs" illustrates which sensors are satisfied at the current altitude. The column "*sel*" and "sel_o" show which sensors are selected to be active in the next state using the basic selection and the optimized selection, respectively. Hence, there is a relation between the "Satisfied ODDs" and the selected sensors. For example, consider altitude 30 where *sel* returns both sensor which matches the satisfied ODDs at altitude 25, 30, and 35, whereas sel_o returns only GPS at altitude 30, but GPS is sufficient for altitudes 25, 30, and 35. Note that sel_o is optimize to minimize the number of active data producers. This might be suboptimal w.r.t. the used sensors as a LiDAR might be preferable at altitude 25. At altitude 30, using *sel* forwards the LiDAR's output, while using sel_o forwards the GPS's altitude. When transitioning from altitude 30 to 35 while using *sel* unnecessarily activates the LiDAR.

Altitude [m]	Successors	Satisfied ODDs	$sel(h)$	$sel_o(h)$	Forwarding	Rationale
0	0, 5	LiDAR	LiDAR	LiDAR	LiDAR	GPS invalid; LiDAR covers all successors.
5	0, 5, 10	LiDAR	LiDAR	LiDAR	LiDAR	GPS invalid; LiDAR covers all successors.
10	5, 10, 15	LiDAR	LiDAR, GPS	LiDAR	LiDAR	LiDAR or GPS possible, LiDAR prioritized.
15	10, 15, 20	LiDAR, GPS	LiDAR, GPS	LiDAR	LiDAR	LiDAR or GPS possible, LiDAR prioritized.
20	15, 20, 25	LiDAR, GPS	LiDAR, GPS	LiDAR	LiDAR	LiDAR or GPS possible, LiDAR prioritized.
25	20, 25, 30	LiDAR, GPS	LiDAR, GPS	LiDAR	LiDAR	LiDAR or GPS possible, LiDAR prioritized.
30	25, 30, 35	LiDAR, GPS	LiDAR, GPS	GPS	LiDAR / GPS	Successor 35 m invalidates LiDAR for sel_o.
35	30, 35, 40	GPS	LiDAR, GPS	GPS	GPS	LiDAR invalid; covers all successors.
			...			
1000	995, 1000	GPS	GPS	GPS	GPS	Cruise fully inside GPS ODD.

Table 1 Outputs of the selection functions *sel* and sel_o and the respective forwarded input.

B. Selection of ML Components

We assume that there are multiple ML components that detect persons on the ground running onboard of the drone. The safety of the operation and ODD of the ML component is dependent on the training data. Specifically, one ML component is trained for low altitude operation and landing, e.g. , below 20 meters, and another ML component is trained for medium altitude during the descent phase between 20 and 50 meters. Let their ODDs be partially overlapping in altitude and speed, similar to the sensor's ODDs discussed in the previous subsection. Each model is trained to detect persons on the ground and is specialized for a certain altitude and speed. Running all ML components simultaneously is typically infeasible due to limited computing resources and power budget, especially on small drones. Instead, we use the automatic selection mechanism to decide which ML component must be active in the next execution step.

$ODD_{MLDetectLow} = \{0 \leq altitude \leq 20, 0 \leq velocity \leq 5\}$

$ODD_{MLDetectMid} = \{20 \leq altitude \leq 50, 0 \leq velocity \leq 15\}$

Certification Aspects of Multiple Scoped ODDs From a certification perspective, this architecture offers a practical advantage. Each ML component is developed, verified, and certified only with respect to its own ODD, according to the existing guidance for ML components in aviation. Let an additional ML component operate at high altitudes in the range 40 to 60 meters, allowing the drone to select a landing site depending on an early assessment of the risk of detected persons.

$ODD_{MLDetectHigh} = \{40 \leq altitude \leq 60, 0 \leq velocity \leq 15\}$

With the automatic selection mechanism in place, the existing ML components do not need to be changed, re-trained, or re-certified. Moreover, the utilization of the ML component depending on the ODD has not changed. The new ML component can be developed, verified, and certified based on its own limited ODD with reduced efforts. The new component can be integrated, as its ODD can be directly used by the selection mechanism. The automatic selection mechanism itself is a small, deterministic software artifact that can be verified using conventional assurance techniques, where most of the computations involved in the selection can be performed offline, which allows to store the results in a lookup table. This modularization enables incremental upgrades of the ML stack while keeping the certification burden minimal and clearly scoped to limited ODDs of individual components.

V. Conclusion

In this paper, we formally defined ODDs that capture operational constraints on sensors, ML components, and other system components. We then focused on the problem of handling multiple data producers with partially overlapping ODDs, where at runtime none, one, or several ODDs are satisfied, but not all data producers can be activated simultaneously. To address this problem, we introduced the automatic selection mechanism that uses ODDs and a guarded transition system to infer which data producer to activate such that there is an satisfying ODD in the next execution step. This guarantees the safe continuation of the operation while dynamically activating and deactivating input producers during flight to satisfy resource constraints and optimize resource usage. For an example use-case, we showed how the selection mechanism activates and deactivates a LiDAR and a GPS-based altimeter, thereby reducing battery consumption for an automatic landing. We further showed how the selection mechanism enables switching between multiple ML component specialized for different flight altitudes, even though running all ML components simultaneously is infeasible due to computational limitations. In future, we plan to validate the automatic selection mechanism through flight test and to relax some of our current assumptions. So far, we assume that components can be activated and deactivated instantaneously and that they require no ramp-up nor ramp-down time when activated or deactivated, respectively.

References

[1] EASA, "EASA Concept Paper: guidance for Level 1 & 2 machine learning applications Issue 02," , Mar. 2024. URL https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-concept-paper-issue-2.

[2] SAE International, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. Surface Vehicle Recommended Practice J3016," , 2016. URL https://www.sae.org/standards/j3016_202104-taxonomy-definitions-terms-related-driving-automation-systems-road-motor-vehicles.

[3] The British Standards Institution, Center for Connected and Autonomous Vehicles, "PAS 1883:2021 Operational Design Domain (ODD) Taxonomy for an Automated Driving System (ADS) – Specification," , 2021. URL https://www.bsigroup.com/globalassets/localfiles/en-th/cav/bsi-cav-safety-benchmarking-report-2021-th.pdf.

[4] 33, I. S., "Road Vehicles – Test scenarios for automated driving systems – Specification for operational design domain," Standard, International Organization for Standardization, Aug. 2023.

[5] Torfah, H., Xie, C., Junges, S., Vazquez-Chanlatte, M., and Seshia, S. A., "Learning Monitorable Operational Design Domains for Assured Autonomy," *Automated Technology for Verification and Analysis*, edited by A. Bouajjani, L. Holík, and Z. Wu, Springer International Publishing, Cham, 2022, pp. 3–22.

[6] Shakeri, A., "Defining Operational Domain and Specifying Operational Design Domains: Current Practices, Standards, and a Systematic Approach," *10th Symposium Driving Simulation*, 2024. URL https://elib.dlr.de/206836/.

[7] ASAM e.V., "ASAM OpenODD: Operational Design Domain (ODD) Standard, v1.0.0," https://www.asam.net/standards/detail/openodd/, Apr. 2025. Release version 1.0.0.

[8] Cappi, C., Cohen, N., Ducoffe, M., Gabreau, C., Gardes, L., Gauffriau, A., Ginestet, J.-B., Mamalet, F., Mussot, V., Pagetti, C., et al., "How to design a dataset compliant with an ML-based system ODD?" *arXiv:2406.14027*, 2024.

[9] Torfah, H., and Seshia, S. A., "Runtime monitors for operational design domains of black-box ml-models," *NeurIPS ML Safety Workshop*, 2022.

[10] Torens, C., Juenger, F., Schirmer, S., Schopferer, S., Zhukov, D., and Dauer, J. C., *Ensuring Safety of Machine Learning Components Using Operational Design Domain*, AIAA, 2023. https://doi.org/10.2514/6.2023-1124, URL https://arc.aiaa.org/doi/abs/10.2514/6.2023-1124.

[11] Yu, W., Li, J., Peng, L.-M., Xiong, X., Yang, K., and Wang, H., "SOTIF risk mitigation based on unified ODD monitoring for autonomous vehicles," *Journal of intelligent and connected vehicles*, Vol. 5, No. 3, 2022, pp. 157–166.

[12] Cofer, D., Amundson, I., Sattigeri, R., Passi, A., Boggs, C., Smith, E., Gilham, L., Byun, T., and Rayadurgam, S., "Run-Time Assurance for Learning-Based Aircraft Taxiing," *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–9. https://doi.org/10.1109/DASC50938.2020.9256581.

[13] Baier, C., and Katoen, J.-P., *Principles of model checking*, MIT press, 2008.