# Get Your Cyber-Physical Tests Done! Data-Driven Vulnerability Assessment of Robotic Aerial Vehicles

Aolin Ding, Matthew Chan*, Amin Hass, Nils Ole Tippenhauer†, Shiqing Ma*, Saman Zonouz‡
Accenture Labs, *Rutgers University, †CISPA Helmholtz Center for Information Security, ‡Georgia Tech

***Abstract*—The rapid growth of robotic aerial vehicles (RAVs) has attracted extensive interest in numerous public and civilian applications, from flying drones to quadrotors. Security of RAV systems is posting greater challenges as RAV controller software becomes more complex and exposes a growing attack surface. Memory isolation techniques, which virtually separate the memory space and conduct hardware-based memory access control, are believed to prevent the attacker from compromising the entire system by exploiting one memory vulnerability.**

**In this paper, we propose ARES, a new variable-level vulnerability assessment framework to explore deeper bugs from a combined cyber-physical perspective. We present a data-driven method to illustrate that, despite state-of-the-art memory isolation efforts, RAV systems are still vulnerable to physics-aware data manipulation attacks. We augment RAV control states with intermediate state variables by tracing accessible control parameters and vehicle dynamics within the same isolated memory region. With this expanded state variable space, we apply multivariate statistical analysis to investigate inter-variable quantitative data dependencies and search for vulnerable state variables. ARES utilizes a reinforcement learning-based method to show how an attacker can exploit memory bugs and parameter defects in a legitimate memory view and elaborately craft adversarial variable values to disrupt a RAV's safe operations. We demonstrate the feasibility and capability of ARES on the widely-used ArduPilot RAV framework. Our extensive empirical evaluation shows that the attacker can leverage these vulnerable state variables to achieve various RAV failures during real-time operation, and even evade existing defense solutions.**

***Index Terms*—Robotic Vehicle Security, System Testing, Cyber-Physical Systems, Vulnerability Assessment**

## I. INTRODUCTION

Robotic aerial vehicles (RAVs), such as quadrotors, are autonomous cyber-physical systems that are currently being adopted in a wide range of applications, such as package delivery [1], urban planning [2] and infrastructure inspection [3]. In the retail industry, Amazon's service Prime Air [4] proposes drone-based package deliveries to its customers, decreasing delivery times to less than 30 minutes with a cost reduction of 90% when compared to vehicle-based deliveries [5].

With the rapid growth of RAV applications, the security of these RAV systems has become increasingly important. RAVs are complex systems utilizing embedded hardware with real-time operational constraints. Typically, such RAV systems receive sensing inputs (e.g., accelerometers, gyroscopes) from the physical world, and calculate real-time vehicle dynamics (e.g., position and velocity) in controller software, then send control signals to actuators (e.g., propellers, motors) to accomplish tasks autonomously. Due to the growing sophistication of RAV features and increasingly diverse set of hardware implementations (e.g., number of motors, weights), the complexity of RAV firmware has trended upwards. For example, ArduPilot [6], a widely adopted RAV firmware, consists of six cascading controllers for every physical RAV specification (e.g., quadrotor, helicopter, etc), while each cascading controller is composed of three primitive sub-controllers for the position, velocity, and acceleration [7]. Frequent control transitions across the many software modules (e.g., libraries, drivers, kernels, built-in tools) expose vulnerable attack surfaces in a RAV's built-in software and programs [8]–[11]. For instance, memory corruption attacks in [11] expose the lack of memory protection in victim processes of real-time RAV controllers. These software attacks can invoke malicious payloads and trigger unsafe behaviors, putting RAVs at considerable risk [12].

However, assessing RAV security is challenging for many reasons. One of the main reasons is that the increasing complexity of RAV firmware makes it infeasible to analyze RAV systems using traditional static program analysis methods, such as control-flow integrity checking and symbolic execution [13], [14]. As one example, the firmware for the 3DR IRIS+ quadrotor has over 25 sensor readings and more than 2,150 parameters [15], with actuators that are updated 50 times each second. In addition, compounding the issue of software complexity of RAVs is their physical nature, which sees the vehicle state and dynamics being affected by a complex and hard-to-predict physical environment. This further challenges the coverage of static analyses, which may only cover a subset of program states in execution.

Dynamic monitoring techniques [16]–[20] can address the physical environment blind spot of static methods as they monitor the RAV's operations in real time. Dynamic monitoring typically uses estimation models of RAV control states to distinguish malicious behaviors from benign ones. However, they only use a small number of state variables and relatively simple controller templates to form the estimation models, which implies a lack of adequate completeness and soundness. Meanwhile, automation is another challenge for dynamic solutions, which require manual tuning [16]–[18] in the variable selection and detector configuration.

In this paper, we present ARES, a vulnerability assessment framework exploring vulnerable state variables within RAV controller software. ARES uses statistical analysis to identify concrete safety-critical variables and their numerical dependencies in an expanded search space which includes the intermediate variables within controller programs. Then, ARES leverages reinforcement learning to automatically assess the physical impact of manipulating each target variable in the victim memory region, identifying potential vulnerabilities. By exploring the state space over a series of time-dependent states, ARES identifies new types of longer-term vulnerabilities as compared to fuzzing works [7], [21], which focus on single-point modifications. This data-driven analysis shifts from the prior focus on the estimation model setup to a focus on the physical impact caused by influential-but-neglected state variables. With ARES, we show that carefully-crafted data manipulations can produce malicious behaviors and even evade detection by prior monitors [16]–[18].

Our main contributions in this paper are as follows:

- We propose a data-driven vulnerability assessment framework that leverages multivariate dependency analysis to identify vulnerable state variables within complex RAV controller software that previous defenses do not consider.
- We present a learning methodology to efficiently search for adversarial manipulations, providing concrete examples of exploiting discovered vulnerabilities.
- We evaluate ARES on the popular ArduPilot RAV suite and demonstrate its feasibility and effectiveness in searching for vulnerabilities and evaluating their impacts.

The structure of this paper is as follows: Section II provides background knowledge for ARES. In Section III, we discuss the limitations of existing works, threat model and challenges for our approach. Section IV illustrates the design of our target state variable selection and the learning-based vulnerability exploits. Section V presents the evaluation and case studies of our vulnerability assessment framework, and in Section VI, we discuss cross-platform extensibility and potential limitations and mitigations. We summarize the related works in Section VII and conclude in Section VIII.

## II. BACKGROUND

### A. RAV Physical Dynamics and Control Models

RAVs are equipped with autonomous control systems that take inputs from sensors such as accelerators and gyroscopes, responding with actuation commands based on controller calculations. In the quadrotor case, the vehicle dynamics include physical attributes like mass and shape and the attitude descriptor as six degrees of freedom (6DoF) including the three axes $(x, y, z)$ and the three rotation directions (roll: $\phi$, pitch: $\theta$ and yaw: $\psi$). Figure 1 shows these dynamics, as well as an abstract view of the ubiquitous Proportional-Integral-Differential (PID) controller that is usually used to control such dynamics. It measures the difference between the current state and the target state, providing feedback for the actuators (e.g., motor torques) to adjust the quadrotor's attitude and position in a cyclical loop. In an autonomous flight task such as a path following mission, RAV controller and data modules execute periodically and update RAV attitudes to arrive at each waypoint. To guarantee stable physical operations, a set of PID controllers must be customized for each RAV physical specification. Figure 1 shows that a RAV controls

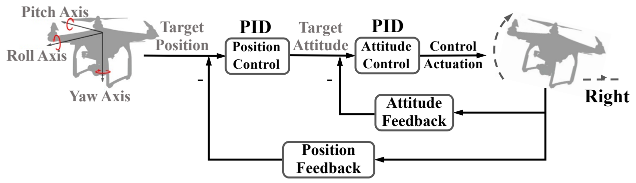


Fig. 1: Flight control block diagram of drone movement.

physical movements along the 6DoF, involving six cascading controllers. Each controller is responsible for a single DoF, such as the y-axis cascading controller [7]. For each DoF, there are three primitive PID controllers (denoted as *ctrl1 ctrl2*, and *ctrl3*), respectively computing the position, velocity, acceleration physicals.

### B. ARM Cortex-M Processor MPU Features

Most RAVs nowadays run their controllers on ARM processors such as ARM Cortex-M4. Memory management techniques has been adopted into resource-constrained systems for memory protection [11] and performance optimization [22]. One important security feature of ARM processors is that they can set up isolated memory regions via a memory protection unit (MPU) in internal SRAM and flash memory. The memory access to regions can be read and write, read-only or inaccessible, based on the time of the processor mode, management mode or user mode, and some additional permissions such as strategies related to cache and buffer operation. When the processor requests access to a region of main physical memory, the MPU will consider access attributes of the region and the processor mode. If the request meets the access requirement of the region, the MPU will allow the kernel to read and write into that memory section; if the request causes a memory access violation, the MPU will reject it and generate a corresponding abnormal signal based on the violation category.

## III. PROBLEM FORMULATION

In this section, we first discuss the limitations of existing RAV methods, then describe the threat model and assumptions behind ARES, in order to highlight the feasibility of an attacker exploiting the vulnerabilities that ARES finds. Next, we discuss the challenges of designing ARES.

### A. Limitations of Existing Works

The complexity of RAVs and the system design of existing defense works raise the issues of completeness and soundness. Firstly, prior state monitoring approaches are incomplete in estimating the physical states and control behaviors of the RAVs due to scalability issues. For instance, the monitoring models that leverage linear equations [17], non-linear estimation [18] or machine learning (ML) models [16], [19] only cover a limited number of physical dynamics and sensor measurements, leaving the the majority of intermediate controller variables unexplored within a RAV's software.

Secondly, in real-time monitoring, prior defense works [16]–[19] typically measure the deviation (error) between the estimated state and the sensed state, raising an alarm if the error exceeds a predetermined threshold. The soundness of such error-threshold-comparison detection depends on the assumption that the *inflicted* error (i.e., the error caused by attacks) must significantly outweigh the *transient* error from imprecise estimation and environmental factors. However, this kind of detection model functions perfectly in the case of naive attacks (e.g., suddenly changing the roll angle to 30 degrees), but performance may degrade when dealing with adversarially-crafted mimicry attacks, as discussed in [16], [19]. Therefore, ARES aims to exploit these completeness and soundness issues and investigate the intermediate variables, showing how an attacker can launch various attacks from them.

### B. Threat Model

In this paper, the attacker's goal is to manipulate vulnerable control state variables with particularly-crafted adversarial values and thus exploit these vulnerabilities with the desired failures (e.g., a path deviation or crash). We assume that the RAV is equipped with an MPU as described in Section II and supports memory isolation via hardware-based privilege separation, a widespread feature in most ARM microcontrollers. As a result, RAV controller processes and variables are isolated in multiple protected memory regions.

We assume that the attacker utilizes existing attack surfaces within the RAV software based on realistic exploits (e.g., known bugs, semantic firmware bugs in specific versions, internal logic access to a certain type of physical quadrotor) and has successfully exploited one individual isolated memory region, thus can perform any data modifications and exploit any processes running in that single compromised memory region. We assume that the attacker also knows the communication channels between RAV and ground control station (GCS), and the attacker can concoct and issue malicious GCS commands to update the control parameters in the victim RAV.

Our attack model is realistic in comparison with prior works [7], [9], [16]–[19], [23]–[25] because (1) the attacker is able to exploit the memory corruption bugs and stack buffer overflow bugs [8], [11], [26] reported in the RAV firmware and perform data-oriented manipulation attacks through malicious code embedding [24] or malicious controller libraries [25]; (2) most RAVs are equipped with a remote control interface (e.g., MAVLink in ArduPilot [6], [7]) for operators to adjust or debug control parameters during its flights. Therefore, we assume the attacker has access to the aforementioned attack surfaces but cannot compromise other hardware components and configurations such as RAV's sensor devices, number of motors, weight, or make.

### C. Challenges

RAV firmware is increasingly complex and varies greatly across different hardware deployments, making it difficult to identify the potential vulnerabilities and assess their severity. As a real-world example, ArduPilot [6] consists of over 2670 configurable parameters [27], 110 library modules and 11 mathematical PID controllers for vehicle state updates. For the core controller modules, there are still hundreds of control parameters and intermediate controller variables that are nested with complicated mutual dependencies as all software modules (e.g., libraries, drivers) are highly interactive. Thus, it is almost impossible (and certainly computationally infeasible) to automatically analyze or fully model them. For instance, in PX4 [15], EKFNAVVELGAINSCALER, a intermediate scaler variable in controller function UPADTE_VEL_CONTROLLER_XY for RAV velocity control in x and y axis, is not visibly safety-critical as other variables describing vehicle dynamics (e.g., _ROLL_TARGET), but it has an indirect impact on an RAV's control behaviors since it scales PID outputs to compensate for the EFK noises. To identify such state variables, we propose a statistical dependency analysis (Section IV-B) to search a state space expanded with intermediate state variables.

Another main challenge is to study the **physical impact** on the vehicle caused by adversarial manipulations of the vulnerable state variables. Our framework identifies the state variable candidates and investigates the impact of potential attacks exploiting them with different desired malfunctions. For instance, an attacker may particularly craft and inject a set of adversarial values to deviate the RAV from its mission paths during flight. In other cases, the attacker may intentionally craft another set of adversarial values to force the RAV to hit a wall. Therefore, we study the capability of attackers exploiting our identified vulnerabilities in realistic physical environments by incorporating the data manipulating and desired malfunction in our optimization process (Section IV-C) to produce feasible attacks.

## IV. DESIGN

In this section, we describe the design choices we make to build a vulnerability assessment framework for the RAV control program by addressing the challenges in Section III.

### A. Overview

An overview of the ARES framework is shown in Figure 2. The initial vulnerability assessment begins with a RAV profiling step. First, ARES establishes a general list of state variables by automatically collecting data traces of the RAV in operation with the onboard dataflash memory logger. The state variables collected include sensor readings, control parameters, intermediate controller variables, and vehicle dynamics. Next, ARES identifies a set of candidate state variables with statistical methodologies including correlation analysis, clustering and multivariate regression. Then, ARES employs reinforcement learning (RL) to generate adversarial values for each selected state variable, leading to the RAV failures, such as path deviation or crashes.

To identify the target state variables, ARES needs the following information: (1) a list of configurable control parameters, (2) a list of RAV dynamics and sensor readings, (3) an RAV firmware and the memory layout of its platform board, and (4) memory regions configured with an MPU. We note that these

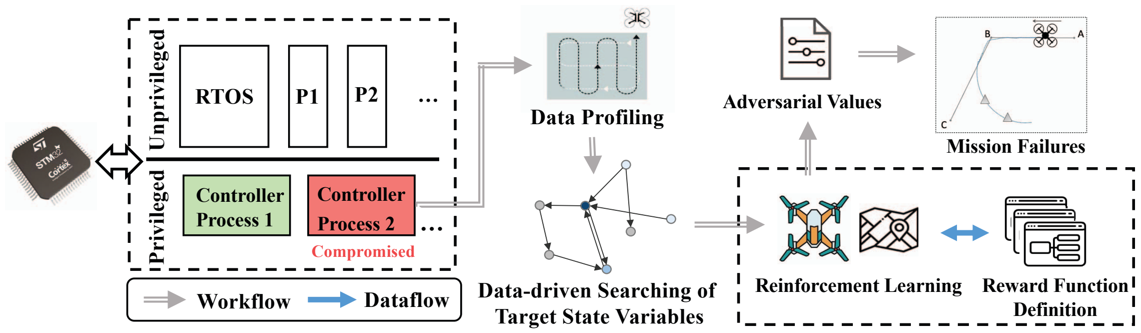


Fig. 2: The overall architecture of ARES.

information are accessible to an attacker under realistic settings. Many vendors (e.g., PX4 [15], ArduPilot [6]) provide the list of configurable control parameters or they can be collected by the command manual defined in the RAV communication protocol (e.g., MAVLink). An integrated logger within RAV software [6], [15], [28] is usually used to record the RAV's operation conditions, which includes the vehcile dynamics and sensor measurements. The memory layout information can be obtained by identifying the microcontroller unit (MCU) (e.g., STM32F427) on the RAV controller board [29]. As defined in III-B, the attacker is aware of the protected memory regions and can compromise an individual memory region to gain access to its intermediate variables.

## B. Target State Variable Identification

In this step, ARES identifies the set of state variable candidates to consider in the learning phase for data manipulation, known as the *target* state variable list (TSVL). Starting with a pre-determined *known* state variable list (KSVL), this state variable list is first expanded into an *expanded* state variable list (ESVL) by incorporating the value traces of the intermediate variables within the RAV controller program. Through statistical analyses of variable dependencies, the ESVL is then pared down into the final TSVL. The expansion and identification procedures are described below.

***State Variable List Expansion.*** ARES first identifies a starting KSVL through easily accessible means such as the onboard dataflash memory logger, which can be downloaded after an operational mission for debugging, or through online RAV specifications and documentation [27]. To obtain the initial state variables and trace intermediate variables, ARES leverages static analysis techniques that are similar with [13]. Specifically, through controller function identification techniques in [13], ARES identifies the memory locations of essential controller programs. ARES adds known controller variables to the KSVL including RAV dynamics (e.g., roll angle, velocity), sensor measurements and controller parameters (e.g., P parameter in PID controller), based on the prior knowledge of official documents and integrated logging information.

Figure 3 shows the combination of vehicle dynamics (*DesR, R, IR, IRErr, tv, DesP, P, DesY, Y*) and the IMU sensor measurements (*GyrX, GyrY, GyrZ, AccX, AccY, AccZ*) as the KSVL. Through memory instrumentation techniques and operational data traces in [13], ARES traces the local variables (*v1, v2, ..., v7*) (defined as private) in the memory region where the PID roll controller runs, as intermediate controller variables which are added into the ESVL as an expansion of KSVL. Note that we annotate these intermediate controller variables here only for readability since ARES does not require semantic disassembling of controller programs [13], [30], which reduces the assumptions for the vulnerability assessment of ARES. In other words, for the controller variable identification, the main difference is that existing works [13], [30] recover the variable semantics based on additional prior knowledge of standardized control templates (e.g., type of control algorithm, controller abstract syntax trees (ASTs)), while ARES, as a data-driven approach, does not relies on semantic disassembling of controller programs and does therefore not require these additional information to establish and instrument the ESVL.

In contrast to previous defense works which monitor either control invariants [17], [18] or specifically-selected control parameters [7], [16], [19], ARES explores a larger state variable space by expanding the KSVL to cover not only plain control parameters but also intermediate controller variables within the compromised memory region, which contains partial control functions (e.g., state error used for updating PID outputs, $P = k_p * error$). This latter group of variables is usually not considered in prior works due to accessibility issues and the assumption that their effects are encompassed by other more important variables. By using statistical tests to determine the magnitudes of these effects, ARES removes the need for this assumption, which may not be true in all cases.

***State Variable Dependency Analysis.*** In this step, ARES investigates the aggregated ESVL using correlation analysis to shortlist the candidate state variables for further assessment. Only a small amount of data in real missions is needed to derive the mutual dependencies (i.e., correlation analysis) within our ESVL, which is practical to gather by employing Valgrind [16], [17] and tracing the memory readings and writings of the identified intermediate state variables. As shown in Algorithm 1, ARES determines the state variable selection strategy to prune the ESVL, refining it into the TSVL

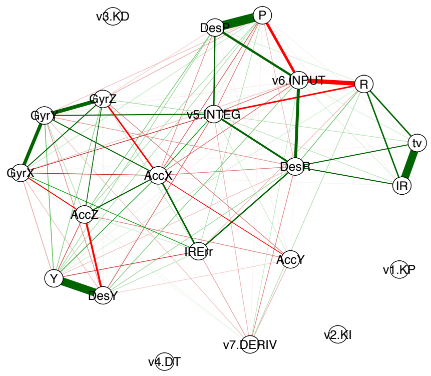


Fig. 3: ESVL example including the PID roll controller and the results of correlation-based dependency analysis. The green line and red line respectively, represent the positive and negative correlation, while the line width reflects the significance (i.e., strength) of the correlation.

**Algorithm 1** Target State Variable List Generation

**Input:** Expanded state variable list $ESVL$
**Output:** Target state variable list $TSVL$

```
1: function PruneStateVarList(ESVL)
2:     for $sv \in ESVL$ do                                  ▷ Check each state variable (sv)
3:         if $sv$ is not $iid$ OR $sv$ is not $NormDist$ then   ▷ Statistical assumptions
4:             remove $sv$ from $ESVL$                       ▷ Prune ESVL list
5:     return ESVL
6: function CheckSignificanceLevel($optimalModel$)
7:     Initialize $StateVariableList$;
8:     for $stateVariable \in optimalModel.features$ do
9:         if $stateVariable.p\_value < 0.05$ then          ▷ Statistical significant
10:            Add $stateVariable$ into $StateVariableList$
11:    return $stateVariableList$
12: function GenerateTargetStateVarList($ESVL$)          ▷ Main function
13:    Initialize $TSVL$;
14:    for $SV_i, SV_j \in ESVL$ do                       ▷ Check every pair of state variables
15:        $CorrMatrix[i, j] \leftarrow$ CalculateCorrelation($SV_i, SV_j$)
16:    $prunedESVL \leftarrow$ PruneStateVarList($ESVL$)   ▷ Assumption Check
17:    $identifiedSubsets \leftarrow$ HIE-Cluster($prunedESVL, CorrMatrix$)
18:    for $subset \in identifiedSubsets$ do
19:        $optimalModel \leftarrow$ StepwiseAIC($subset$)   ▷ Model selection
20:        $StateVariableList \leftarrow$ CheckSignificanceLevel($optimalModel$)
21:        $TSVL \leftarrow TSVL \cup StateVariableList$     ▷ Add identified variables
22:    return $TSVL$
```

to represent a potentially vulnerable surface, which is not sufficiently studied in previous works [16]–[19].

As an example, Figure 3 shows the correlation-based dependency results for ESVL examples including the PID roll controller. Our goal is to look for intermediate controller variables indirectly associated with essential vehicle dynamics (e.g., roll angle) but do have a significant impact on the completion of the RAV's safe operation, which are the candidates for our learning protocol to exploit through particularly crafting the adversarial values. Given the fact that the operation profile data and the value changes of the ESVL are represented in time series, correlation analysis is employed here to explore the state-by-state ESVL updates in the sequential cycles of the RAV. In our case, each state is a multivariate form. ARES quantifies the significance of the pairwise correlation of all continuing state variables (Line 14-15) in the ESVL using the Pearson correlation coefficient [31], which can be mathematically described as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1}$$

where $n$ is the number of recorded iterative cycles for the RAV control, $x_i$ and $y_i$ are the individual state variable indexed with $i$, $\bar{x}$ and $\bar{y}$ are the mean values for each state variable. Generally, the Pearson correlation coefficient ranges from -1 to 1 (i.e., -1 means the perfect negative linear correlation and 1 means the perfect positive linear correlation). For instance, the intermediate controller variable $v5$ (i.e., PID input error) in Figure 3 shows a significant correlation with variable R (i.e., roll angle) because the input error changes will impact the PID outputs and further propagate to the roll control.

By calculating the pairwise correlation coefficients for ESVL, we have a quick and simple summary of the direction and strength of the linear relationship between every two state variables. However, it is insufficient to apply the correlation analysis alone since it assumes that both variables in each pair play the same role. Basically, we are more interested in how control parameter updates influence vehicle dynamics, rather than the opposite. Therefore, we utilize linear regression analysis to further explore the numerical relationships (i.e., more detailed than correlation alone) between the state variables and how they can be modelled. Before that, there are few statistical assumptions needed to be initially checked regarding the dataset, in order to use the regression model. Specifically, we prune the ESVL list (Line 1-4) with two additional prerequisites: (1) the state variables must be normally distributed, and (2) the state variables must be independent and identically distributed (IID) to each other (Line 3). Therefore, the intermediate state variable with constant values such as `v1(KP)`, `v2(KI)`,`v3(KD)` in Figure 3 will not be considered in the correlation analysis.

***Target State Variable Selection.*** After obtaining the ESVL, ARES automatically selects the target state variables in following procedures: (1) To accelerate the state variable selection, ARES applies the hierarchical clustering technique [32] to create subsets of the ESVL (Line 17) based on the correlation coefficients matrix (Line 15). Compared to other clustering techniques like K-means, it does not require a pre-specified number of clusters [33]; (2) For each subset, ARES performs a stepwise feature selection (Line 19) using the Akaike information criterion (AIC) [34], a widely-used statistical method for model comparison. AIC method produces the optimal set of state variables by examining the model's significance level after adding or removing different state variables. In the example of Figure 3, ARES searches the optimal regression model for the response variable *R* (the roll angle) and explanatory variable set (*P, DesP, INPUT, DesR, tv, INTEG, IR*); (3) For the optimal regression model, ARES further examines the *p-value* (i.e., a common statistical number ranging from 0 to 1 for null hypothesis test) of its contained state variables (Line 8-9). ARES adds the state variables whose is smaller than 0.05 into TSVL (Line 9-10), which indicates there is a statistically significant relationship between this response variable and the predictor variable. Note that we do not consider more complicated scenarios such as multicollinearity tests, and therefore conservatively keep the identified target state variables. In summary, we employ the correlation analysis and regression analysis to automatically and efficiently select target state variables, where we create adversarial values using a learning protocol presented in the following section.

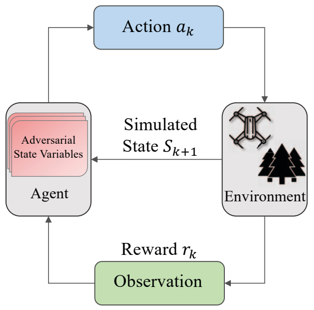
### *C. Generation of Adversarial Values*

After determining a set of target state variables, we generate adversarial patterns of values that can result in unstable or exploitable RAV behaviors. Even after selecting a small set of target parameters, several challenges remain in identifying a state variable manipulation vulnerability that could disrupt RAV operations. Despite having pruned many variables, the search space remains intractable when considering: (1) which concrete state variables need to be manipulated; (2) the adversarial sequences to be injected and their timing; (3) the overall impact on the RAV's control state.

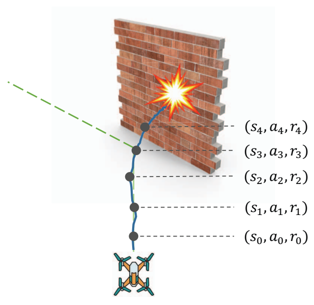
Because of the reasons enumerated above, it is still infeasible at this point to exhaustively search the state space of the selected variables. Instead, we formulate the generation of adversarial state variable values as a reinforcement learning (RL) problem to efficiently explore the state space, exploiting learned information to discover potential vulnerabilities.

RL problems share a common basic structure, as presented in Figure 4a, consisting of an agent taking actions within an environment under specific policies. After each action, the agent observes state changes in the environment and receives a reward for that action. Over multiple episodes, the agent learns a policy of optimal actions to maximize the total reward. The RL agent is trained offline in simulation, manipulating the target state variables over repeated mission runs. Its goal is to learn a sequence of state variable manipulations resulting in unsafe RAV behavior. In learning adversarial state variable sequences, ARES demonstrates how these variables can be explored to probe for vulnerabilities.

***Defining RL Components.*** In our scenario, we consider a RAV flying a mission to achieve an objective. As we only manipulate specific internal variables of the RAV, the RAV itself as well as the mission it follows are a part of the environment. The agent takes the role of an external adversary, which at each time step can choose whether or not to modify one of the target state variables by a certain amount. At each state $s_t \in \mathbf{S}$, the agent chooses what it believes to be the best action $a_t \in \mathbf{A}$ according to a policy $\pi(a_t|s_t)$. The agent receives a reward $r_t(s_t, a_t)$ and observes the next state $s_{t+1}$. For each training episode, the return $R$ is equal to the sum of total discounted future rewards: $R = \sum_{t=0}^{\infty} \gamma^{\ t} r_t \quad , \quad 0 < \gamma < 1$. This process can be seen in Figure 4b. In the case of ARES, the state and action spaces are defined as follows. The state space is represented by a collection of data variables (position, velocity, rotation, etc.) and control variables (controller states, configuration parameters, etc.), representing the RAV state.



(a) A typical RL workflow



(b) RV exploit example with RL

Fig. 4: Reinforcement Learning (RL) setup for RAV Exploits.

The action space is limited to variables within the TSVL, and parameterized by the amount that the variable is changed.

***Considerations for Data Manipulation.*** In a realistic RAV controller implementation, a RAV's position and attitudes are commonly stored in vectors and matrices, leading to heavy use of pointer operations in their calculation. Leveraging this fact, ARES can probe for both syntactic bugs [7] and traditional memory corruption bugs (e.g. buffer overflows), while accounting for MPU protections. The first step of data manipulation is variable choice. We can initially determine this using the TSVL, then modify the selected state variables by analyzing changes from previous parameter states. For example, if the RAV is in a region of high dependency between two variables, then it is sufficient to only select one. For simplicity, we limit our evaluation to a single variable. The next step is manipulation amount. Manipulations can be either random or bounded (gradual changes relative to the current value), and this choice is partially dependent on the type of the variable. For example, physical data is usually bounded, while other parameters can change either way. Finally, we can further incentivize impactful variable choices through the definition of RL reward functions.

***Reward Function Formulations.*** The reward function itself can reflect a variety of attacker goals, such as crashing the RAV, deviating it from its intended mission path, or even adversarial control. Over many episodes, the agent will learn an adversarial policy satisfying the constraints of the attack as defined by the reward function. In general, we define two categories of vulnerabilities based on the definition of their attack goals in the RL reward function: ***controlled*** and ***uncontrolled*** failure vulnerabilities. For uncontrolled failures, the only necessary result is RAV failure, while controlled failures lead to specific failure modes. This distinction defines how we approach the RL formulation of the problem for each category. More importantly, this categorization also simplifies our reward function design.

***An Uncontrolled Failure Formulation.*** In the case of an uncontrolled failure, we consider a RAV following a predefined mission path. The agent attempts to steer the RAV away from the mission path, resulting in mission delays and failures.

However, there is no specific failure outcome (e.g., attitude angle, velocity or directions during the deviation).

The reward function for an uncontrolled failure is as follows:

$$r_t(s,a) = \begin{cases} +\mathrm{C}_1 & \text{deviating from the path,} \\ -\mathrm{C}_2 & \text{not deviating from the path.} \end{cases} \tag{2}$$

In each time step, we reward the agent based on whether or not it has increased its overall distance from the mission path. The agent will keep accumulating rewards the further it can get from the path.

***A Controlled Failure Formulation.*** In this scenario, we consider a RAV moving through a space containing forbidden navigation zones. These could range from simple obstacles like a wall to restricted airspace in the case of a drone. Therefore, its path planning needs to avoid these zones. In the controlled failure case, the RAV is specifically steered toward these obstacles, resulting in negative consequences. In the simplest formulation, we define our reward function as follows, where the goal is within the forbidden zone:

$$r_t(s,a) = \begin{cases} +\mathrm{C}_1 & \text{moving toward the goal,} \\ -\mathrm{C}_2 & \text{moving away from the goal,} \\ +\infty & \text{having reached the goal.} \end{cases} \tag{3}$$

In each time step, we reward the agent some positive or negative reward C depending on whether it has moved closer or further from the obstacle, with exceptions for when a crash has occurred (case 3 in Equation 3). In the controlled failure case, the RL agent has a goal that signifies success (e.g., a collision with an obstacle) when reached. A controlled failure can represent several adversary goals such as a crash (i.e., the goal is the wall) or even adversarial control (moving the RAV to a specific point). Each goal requires further specification of reward function terms to guide the agent effectively.

## V. Evaluation

In this section, we present the experimental setup of evaluating ARES. We present the search results of ARES in identifying target state variables. We also present the effectiveness of the learning strategies that exploit the vulnerabilities, which might even bypass the detectors.

### A. Experimental Setup

***System Setup.*** For our evaluation, we use two virtual vehicles, IRIS+ (a quadrotor) and Pixhawk4 (PX4), equipped with the ArduCopter firmware of ArduPilot 3.6 for our RAV control software due to its popularity and open-source nature. We execute the vehicle model dynamics, runtime behaviors, and real-world environment using the ArduPilot suite and with the 3D robotics framework Gazebo [35]. This allows for the production of state variable data in response to control actuation, as well as the inclusion of mission planning and external environmental factors in our experiments.

To handle the continuous action space of the RAV, we opt for a policy gradient method [36] over the conventional Q-learning algorithm [37] using OpenAI Gym [38] for the training of our agents. We `initialize` the Gym environment and set up the drone's initial position. In the episodic task, the agent takes a single action in each `step` function every 0.3 seconds and injects a variable manipulation of the target state variable through MAVProxy commands. We set the maximum number of episodes to 5000 with each episode having a maximum length of 300 steps. Rewards are calculated based on the observations of the new state. After the completion of each episode, the Gym environment will `reset` the IRIS+ model by landing, disarming the vehicle, and resetting it back into its initial position. For the next episode, the vehicle will again be armed and take off to the desired mission starting point to begin the observation-action training for the new task.

***Research Questions.*** We evaluate ARES in order to answer the following research questions:

- **Q1**: How practical is ARES for data-driven exploration of state variable vulnerabilities?
- **Q2**: Can ARES find exploits that bypass existing defenses deployed on RAVs?
- **Q3**: How effective is learning-based data manipulation for finding vulnerabilities leading to uncontrolled and controlled RAV failures?

### B. Data-Driven Search of Target State Variables

For the ArduCopter, we start with the KSVL as shown in Table I; the detailed description is available at [27]. We describe the three steps to identify state variable candidates: (1) data tracing to expand the state variable search space, including intermediate controller variables within the various controller algorithms; (2) statistical clustering based on the correlation coefficients derived from the collected dataset; (3) and selecting the target state variables to efficiently search the vulnerable ones in the learning framework. We present the results for each step in Table II.

TABLE I: The available variable list (total of 342) in ArduCopter built-in dataflash logger as the known state variable list (KSVL). ALV: available log variables

| Name | # of ALV | Name | # of ALV | Name | # of ALV | Name | # of ALV | Name | # of ALV | Name | # of ALV |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AHR2 | 7 | ATT | 12 | BARO | 5 | CMD | 6 | CTUN | 6 | CURR | 7 |
| DU32 | 3 | EKF1 | 14 | EKF2 | 12 | EKF3 | 11 | EKF4 | 14 | EV | 2 |
| FMT | 6 | GPA | 5 | GPS | 14 | IMU | 12 | IMU2 | 12 | MAG | 11 |
| MAG2 | 11 | MAV | 2 | MODE | 3 | MOTB | 5 | MSG | 1 | NKF1 | 14 |
| NKF2 | 13 | NKF3 | 12 | NKF4 | 13 | NTUN | 11 | PARM | 3 | PIDA | 7 |
| PIDR | 7 | PIDY | 7 | PIDP | 7 | PM | 7 | POS | 5 | RATE | 13 |
| RCIN | 15 | RCOU | 13 | SIM | 7 | VIBE | 7 | | | | |

***State Variable List Expansion.*** ARES extracts and classifies the KSVL from the dataflash logger in ArduCopter with core controller algorithms. Specifically, we focus on the PID controller, the square root controller (e.g., for position estimation) and the strapdown inertial navigation system (SNIS) (e.g., for velocity and position correction). For a RAV system equipped with protected memory regions, PID controllers executed by the stabilizer process usually run in the same memory region.

TABLE II: The number of potential state variables as attack surface for essential controller software examples using data-driven searching strategy at each step. Sqrt: square root controller, SINS: Strapdown inertial navigation system.

| Controller Function | # of KSVL | # of Added SVs | # of ESVL | # of TSVL | Ratio of SV Selection |
|---|---|---|---|---|---|
| PID | 28 | 36 | 64 | 6 | 9.4% |
| Sqrt | 9 | 12 | 21 | 3 | 14.3% |
| SINS | 14 | 19 | 33 | 3 | 9.1% |

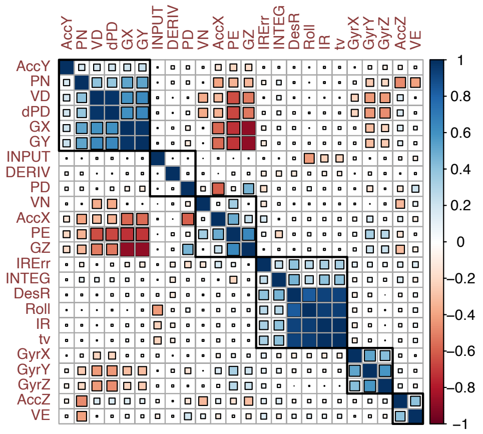


Fig. 5: Correlation coefficients heat map with hierarchical clustering for the ESVL of RAV's roll control.

Table II combines the results of all PID controllers including their sub-controllers (e.g., PIDR, PIDY, PIDP controller) and variants (e.g., P, PI, PIDFF controller). We trace the updates of the attitude state variables (e.g., roll, pitch angle) in the 28 available log variables and identify the memory locations to profile 9 intermediate variables (e.g., P, I, D gains) for each of their PID controllers. For this ESVL containing 64 state variables, we categorize them as different subsets associated with specific vehicle dynamics we are interested in. After pruning the ESVL and removing unchanged state variables (e.g., having a constant value), we determine the total of 24 state variables shown in Figure 5 as the ESVL specifically for the roll angle and its associated controller variables.

***Dependency Analysis and Target State Variable Selection.*** To identify those state variables highly associated with the RAV operations (e.g., vehicle dynamics), ARES utilizes the pairwise correlation analysis. We log the dataset at a frequency of 16HZ for the ESVL in 5 benign missions and each of them takes about 40 to 70 seconds to complete, as a result collecting over 3000 value vectors for candidate state variable in time series. We calculate the pairwise correlation coefficients for all state variables in the ESVL, obtaining the correlation matrix shown in Figure 5. This heat map of correlation coefficients clusters the vehicle roll angle and other variables in the ESVL, including sensor measurements (e.g., accelerator and gyroscope), vehicle dynamics and intermediate controller variables (e.g., the PID input error). With the results of the correlation-based dependency analysis, ARES ranks the state variables by their significance level in regression analysis of subsets and selects 4 candidates (i.e., INTEG, DesR, IR, tv) as target state variables in Figure 5. They specifically correspond to the roll angle by using correlation-based dependency analysis and regression significance check for state variable sets (see Section IV-B).

## C. Vulnerability Assessment with Existing Defenses

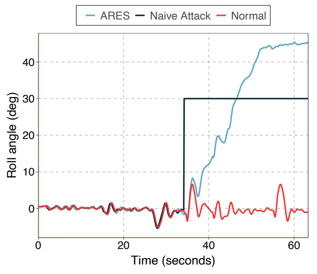
Control invariants [17], ML-based monitoring [16], and sensor estimation [18] have been identified as the most common forms of RAV monitors these days. These techniques inspect the integrity of control invariants, actuator outputs and sensor measurements by comparing the error between the expected and perceived values with the deduced threshold. By exploiting their limitations, as we pointed out in Section III-A, we demonstrate that ARES can exploit the vulnerable state variables and even evade these defenses.

To allow the RL agent to learn vulnerabilities and edge cases in existing defenses, we first include a control invariants [17] monitor within the simulation, which is then incorporated into the reward function. This incentivizes the RL agent to explore areas of the state space which do not trigger an alarm, but still lead the RAV toward the desired attacker goal. To validate the effectiveness and stealthiness of adversarial vectors generated by ARES, we compare with a naive attack strategy as the baseline which generates rapid changes in the RAV's vehicle dynamics (e.g., roll angle).

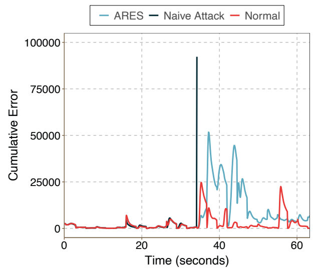
***Effectiveness.*** In our observations, a mild manipulation of yaw and pitch control (e.g., left/right, forward/backward) over time will have less impact on the drone, since the drone has active control of these degrees of freedom compared to the roll axis in a path following mission consisting of a couple of straight lines (e.g., the drone turns direction towards the next waypoint after reaching a waypoint and thus always moves forward without tilting much in the roll axis). In order to achieve the mission deviation, ARES takes the actions of individually manipulating the values of each state variable in ESVL to learn the optimal attack surfaces identified in Section V-B within the PIDR controller (i.e., the PID controlling the roll angle rate).

Through manipulating the input error values in PIDR controller, we successfully increase the roll angles for 2.5 degrees every second (i.e., on average 0.00625 degrees at each step when accounting for noise) amidst the mission until it reaches 45 degrees. This attack slowly deviates the drone from its mission paths and the RAV gets drifting away and fails to complete the mission (eventually crashes after draining the battery). Our adversarial data manipulations prove to be undetected by control invariants during the mission. Figure 6a shows the roll angle changes, and we also comparatively implement a simple attack strategy which naively sets the roll angle to 30 degrees, quickly crashing the drone but also getting detected by the control invariants immediately.

***Stealthiness.*** After successfully compromising the drone, we analyze the stealthiness of our adversarial manipulations by

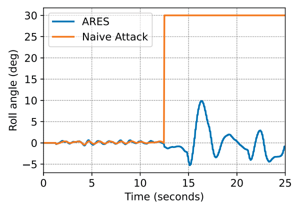


(a) Roll angle

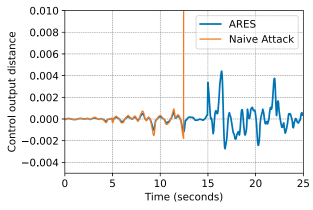


(b) Cumulative error

Fig. 6: The control invariants detection results for the ARES's adversarial manipulations and a naive attack example, with the attack beginning at the vertical line of the naive attack.



(a) Roll angle

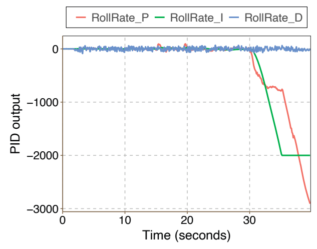


(b) Control output distance

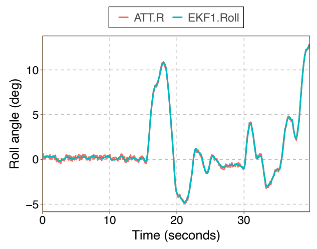
Fig. 7: The ML detection results of ARES's adversarial attack.

assuming that the control invariants model is accessible for us to take a closer and transparent look at the difference between our manipulations and naive attack under the monitoring of the control invariants detector. The control invariants protocol in our setup has a checking frequency of 400Hz, a predetermined threshold of 400,000 and a monitoring window of 1024 time steps (i.e., about 2.5 seconds). Figure 6b shows the control invariants model's cumulative estimation error during a benign mission, the same mission with a naive attack, and with our manipulations. When our manipulation starts, the cumulative error generally increases but still fluctuates within the "safe" ranges of the detection boundary, which will be classified as the `attack-free` condition and not trigger an alarm. For the naive attack, the cumulative error sharply increases to and stays over 1,000,000, which is significantly greater than the predetermined threshold, and immediately triggers an alarm. Therefore, the learning protocol of ARES produces a more sophisticated attack strategy, demonstrating much greater stealthiness in comparison to the naive attack on RAV systems in the presence of a control invariants detector.

***Assessment of Machine Learning-based Monitors.*** For real-time RAVs, malicious modifications of control parameters can cause inconsistency issues of controller outputs and eventually disrupt the safe operation. Compare to control invariant, ML-based monitors [16], [19] estimate the numerical calculations within mathematical controllers and detect the malicious control output deviations. In our experiments, we assume an IRIS+ drone whose PID controller outputs are monitored by [16],



(a) PID output



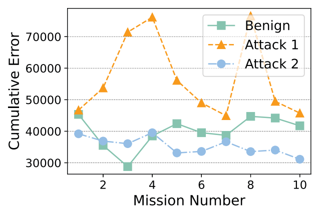
(b) EKF estimation

Fig. 8: The detection results of ARES's adversarial attack by sensor estimation techniques such as EKF.

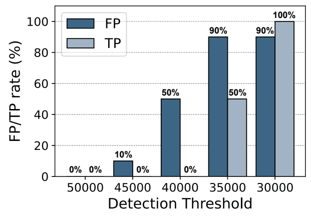
is hovering at 5 feet above a fixed point on the ground. As shown in Figure 7a, we launch the attack at time instance 12.0s by gradually manipulating the PID scaler ratio of the RAV's roll controller and also compare with the naive attack. The manipulations of ARES disturb the stabilizing of the RAV's roll angle and cause the RAV drift away from the original location, while the naive attack forcibly sets the roll angle to 30 degrees. Figure 7b shows that the control output distance of the naive attack sharply increases and exceeds the upper bound of the benign error range (i.e., threshold) 0.01, while the fluctuations of control output distance for ARES remains in the benign error range, and thus ARES can evade the ML-based detection.

***Assessment of Sensor Estimation Detectors.*** In addition, ARES can also evade sensor-based state estimation detectors like SAVIOR [18], since ARES manipulates controller variables instead of compromising the sensor measurements. For instance, we identify the PID controller outputs as the attack surface and manipulate its value by adding a gradual perturbation; the manipulation of the controller output is directly fed to the motor controllers, leading to modified actuation. This causes the PID to attempt to compensate in response to the perturbed control as shown in Figure 8, eventually failing to do so and crashing the drone. This manipulation is successful since the PID controller output (the sum of P, I and D) has an oversized safety range of $\pm 5000$, exposing it to the range validation bugs reported in [7]. When the attack starts at $t = 30s$, the roll angle of the RAV enters into an unstable and aggressive stabilization as shown in Figure 8a. However, the residual between sensor measurements `ATT.R` and EKF estimated state `EKF1.Roll` remains close to 0 in Figure 8b because the drone's motion is affected by the variable manipulation, and as a result will not raise an alarm.

***Robustness to Various Detection Thresholds.*** ARES exploits the circumstance that the detector is already set up, but the monitoring configuration (e.g., threshold and window size) can be adjusted by re-configuring the RAV model. In this section, we look at the control invariants work as an example that this vehicle-specific protection can not prevent the exploits generated by ARES through simply reducing the threshold or adjusting the detection window size, which are commonly used

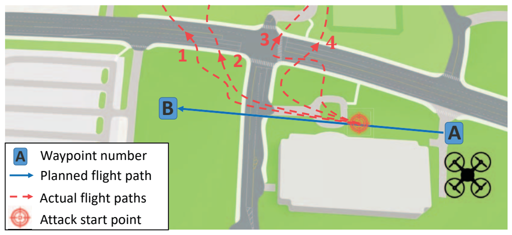


(a) Maximum invariant cumulative error

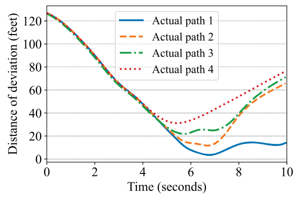


(b) False Positive and True Positive rate

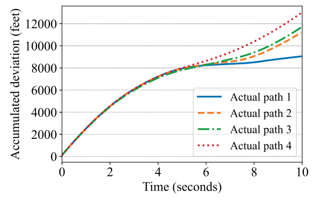
Fig. 9: Control invariants detection on two ARES attacks.



(a) Uncontrolled failures of the roll controller deviating an RAV from the mission path and prevent it reaching the next waypoint B



(b) Distance of deviation from next waypoint B under different exploit scenarios



(c) Accumulated distance of deviation from next waypoint B under different exploit scenarios

Fig. 10: RL-based uncontrolled failure as path deviation.

to increase the detection precision and sensitivity.

For the control invariants detector, decreasing the size of the sliding window is less applicable and may easily cause a high false positive rate [17]. Therefore, we evaluate two additional attacks to show that an improved control invariants model with a varying threshold is still insufficient to ARES. The only difference is that Attack 1 increases the roll angle on average by 0.0125 degrees at each step when accounting for noise, while Attack 2 increases the roll angle on average by 0.000625 degrees. We launch the two attacks and also collect the benign data for 10 trials on various missions. Figure 9a presents the maximum cumulative error observed during flight for each mission. We assume that the control invariants model will sound an alarm once the cumulative error exceeds the predetermined threshold. We observe that the larger scale of Attack 1 results in a larger cumulative error, while Attack 2 is difficult to distinguish from the benign runs since their maximum cumulative errors are very close, meaning that the control invariants detector can not easily identify our attack even by setting a more sensitive threshold value. Figure 9b presents the false positive and true positive rates in these experiments by adjusting the threshold. Although decreasing threshold values from 50,000 to 30,000 improves the true positive rate, it also results in higher false positive rates, which significantly weakens the detection capability of the control invariants model (e.g., 90% false positive rate is unacceptable in realistic applications).

### *D. Learning-based Vulnerability Assessment Strategies*

Here we present the two exploit categories, uncontrolled and controlled failures, in ArduPilot using reinforcement learning algorithms (Section IV-C). We experimentally train and validate ARES to generate adversarial manipulations that can lead to uncontrolled and controlled failures while successfully evading the detectors. We also evaluate the effectiveness of ARES in finding state variable vulnerabilities, as well as its efficiency in learning such vulnerabilities.

*1) Case Study I: Uncontrolled Failure:* In this case, we consider an uncontrolled failure for the path following mission for the IRIS+ quadrotor equipped with ArduCopter control firmware. The goal of our agent is to learn how to deviate the drone from its predetermined path by manipulating a vulnerable state variable. However, we do not have the precise definition of the desired malicious behaviors (e.g., angle, velocity of the deviation) but only an intuitive adversarial plan (e.g., deviate the drone as far as possible) reflected by the reward function. Therefore, we do not consider or require the actual direction the RAV will deviate towards during the failure. We explore the RL learning process and show how ARES can evade detectors.

To deviate the RAV from its mission path, we define our observations as the minimum distance between the current position $P_{RV}$ and the paths $P_{th}$. As shown in Figure 10, we start the exploit between waypoint A and waypoint B, sending adversarial state variable values of $PIDR.INTEG$ (i.e., the internal integrator in the PID roll controller) through $MAVProxy$ commands to the ArduCopter controller. When the exploit starts, the agent uses the current position $(x_{rv}, y_{rv}, z_{rv})$ to calculate the minimum Euclidean distance to mission paths as the observation

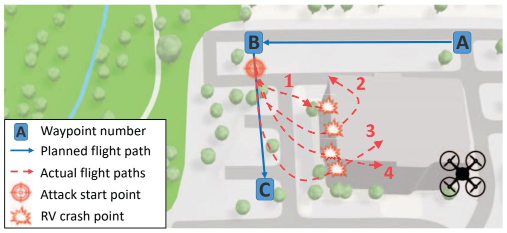
$$d = min(\sqrt{(x_{rv} - x)^2 + (y_{rv} - y)^2 + (z_{rv} - z)^2})$$

in which $x$, $y$, and $z$ are points defined on the original paths. For instance, the path from waypoint A to waypoint B can be defined as a line segment in the 3D space

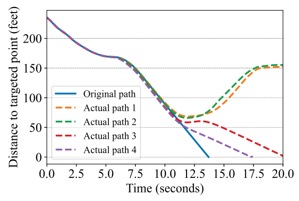
$$P_{th}(A, B) : z = \alpha_1 * x + \alpha_2 * y + \beta$$

where $x \in [x_A, x_B]$, $y \in [y_A, y_B]$ and $z \in [z_A, z_B]$ and $\alpha_1$, $\alpha_2$, and $\beta$ are parameters of the line equation. Having calculated the distance, and for a small $\varepsilon = 0.01$ representing the radius of the drone, we define the reward function at time step $t$ as follows:

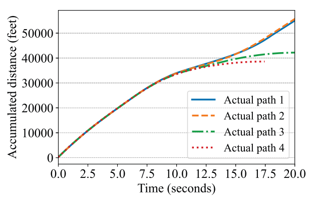
$$r_t(s, a) = \begin{cases} +\Delta d & d_t > d_{t-1} \text{ and } d_t > \varepsilon \\ -\Delta d & d_t < d_{t-1} \text{ or } d_t < \varepsilon \\ -\infty & \text{if an anomaly is detected.} \end{cases} \quad (4)$$



(a) Controlled failures crashing RAV to targeted point caused by the low roll rate parameter in roll velocity controller



(b) Distance of path deviation for different attack scenarios via RL



(c) Distance of path deviation for different attack scenarios via RL

Fig. 11: RL-based targeted attack strategy for hitting obstacles.

In this case, the agent will gain relatively small rewards unless it can create a larger deviation ($\Delta d = ||d_t - d_{t-1}||$) by adversarially manipulating the targeted state variable values. In Figure 10 we demonstrate a less successful exploit, path 4, where the RAV deviates from the path but the reward function does not constantly increase on the presented attack routes passing through positions A and B. In Figure 10 path 4, the distance observations of the routes are strictly increasing between the two steps from position A to position B. Therefore, the agent moving along more successful routes will learn from these observations and continue to manipulate the state variable and deviate the RAV even further from its mission paths with the aim of maximizing the cumulative reward, which turns to be a more successful attack strategy compared to path 1.

*2) Case Study II: Controlled Failure:* In the controlled failure case, the goal of the agent is to learn parameter manipulations that lead the drone toward a forbidden area or obstacle. We use the same drone model and control software as in the uncontrolled failure, and similarly, explore one target state variable at a time.

We again define our observations as the minimum distance between the current RAV position $P_{RV}$ and the closest point of the forbidden area $P_f$. We define the forbidden area as a set of geometric surfaces in a similar manner to the path segments of the untargeted attack. Once again, using the calculated distance and $\varepsilon$, we define the targeted attack reward function at time step $t$ as follows:

$$r_t(s,a) = \begin{cases} +\Delta d & d_t < d_{t-1} \text{ and } d_t > \varepsilon \\ -\Delta d & d_t > d_{t-1} \\ +\infty & d_t \leq \varepsilon \\ -\infty & \text{if an anomaly is detected.} \end{cases} \quad (5)$$

As presented in Figure 11 path 3, as the distance of the drone from the obstacle increases from $d_1$ at waypoint B to $d_2$ at waypoint C, the rewards will be negative. Likewise, in Figure 11 path 2, as the drone approaches the obstacle, the rewards are positive and increase with increasing velocity. Unlike the uncontrolled failure mode, the episode ends if the agent is able to steer the drone sufficiently close to the goal (i.e., make contact with the obstacle and crash).

## VI. Discussion

***Generalizability and Scalability.*** We evaluate ARES on the ArduPilot platform, which is a well-renowned robotic vehicle platform with over 15 years history. The ArduPilot code base consists of more than 700k lines of code, whose scale is comparable with many other commercial RAV firmware (e.g., MantisQ [39], Crazyflie [28]) and also much larger than many small embedded IoT device firmware. ARES assesses the vulnerabilities in an expanded state variable space including the configurable control parameters, sensor measurements, vehicle dynamics, and intermediate controller variables, which are widely spread and implemented in many RAV applications (e.g., drone, fixed-wing plane, helicopter). The static analysis techniques [13] that ARES leverages is agnostic to the RAV's firmware since [13] is demonstrated on both open-source firmware ArduPilot and closed-source (stripped) firmware MantisQ [39]. The methodologies we involved, such as target state variable selection based on multivariate correlation analysis and reinforcement learning-based data manipulations, are agnostic to the RAV's controller software, which can be generalized to more realistic RAVs (e.g., Crazyflie [28], MantisQ [39]). Despite the different implementation details and physical configurations, RAV firmware implementations generally follow a similar pattern such as universal controller software (e.g., PID, Extended Kalman Filter) in a cycle-based paradigm which can be assessed by our learning framework. In addition, most RAVs nowadays run their controllers on ARM processors, which our evaluations are also based on.

***Platform Applicability for Real RAVs.*** Regarding ARES 's applicability for real RAV models, ARES aims to explore the non-obvious vulnerabilities within the intermediate state variables of RAV controller software. We evaluate ARES on ArduPilot, whose controller software is running on many real-world commercial RAVs such as 3DR Solo, AeroDrone MR4 and Xunwing X4. We leverage its simulation toolkit to simulate the RAV physics and test the effectiveness of ARES. The training process of RL-based control methods [40]–[43] is usually limited to software simulation instead of real drones or vehicles because of the difficulties in various environment settings, high financial and time cost of many potential crashes. Further research on more consumer-grade RAVs with closed source firmware will be our future work.

***Limitations of* ARES.** The components that ARES relies on have two limitations in finding target state variables and crafting manipulations. One limitation is inherited from static analysis [13] that ARES may mistakenly identify different controller variables as identical aliases (i.e., false positives) or miss the alias of state variables (i.e., false negatives) when identifying the state variables. This is a common limitation of points-to analysis [44] in alias tracking, which impacts the accuracy and coverage of state variables we find. To address this limitation, further research of advanced static analysis and dynamic analysis is required, as pointed out by [29].

The other limitation of ARES is that the RL-based search for potential exploitable vulnerabilities is sometimes incomplete. Apart from scalability issues, checking for RAV vulnerabilities using data manipulations is subject to firmware protection restrictions, as mentioned in our threat model, and the design robustness of the RAV: (1) the controller program can reject obviously illegitimate parameter values; (2) mild variable manipulations can be discarded by the RAV controllers as an environmental disturbance. Moreover, RL process may take longer time to search a large size of ESVL or TSVL, although we rarely see this condition in our experiments. These restrictions further limit the evaluation comprehensiveness for the RL search space, especially when considering that malicious behaviors may occur over time. Therefore, ARES focuses on demonstrating that an attacker with limited capabilities of accessing only several state variables in a single compromised memory region can still craft various attacks to bypass existing defense mechanisms [16]–[18].

***Countermeasures.*** The root cause of state variable vulnerabilities in our work is the lack of completeness and soundness in the system design of RAV monitors. We foresee that there are two directions to improve them. Firstly, RAVs need better memory management (e.g., hybrid design [45]) and fine-grained monitors in the variable level [13] rather than the system level. Secondly, RAV monitors can enlarge monitoring objectives by combining control invariants or control parameters with essential state variables we identified within controller functions. This requires enhancing the reverse engineering techniques and the runtime support of RAV's software.

## VII. Related Work

***RAV Attacks.*** Physical attacks targeting on RAV sensors including accelerators [46], [47], optical sensor flow [46], GPS signals [48] and gyroscope [23] has been proposed. Compared to sensor spoofing attacks, ARES focuses on state variables and control parameters within the RAV controller software. RAV is also facing the challenges of various parameter-wise attacks such as memory corruption [8], [11], vulnerable control parameters [7], [26], logic bugs [49] and control logic manipulation [25]. By leveraging these cyber-physical attack surfaces, the attackers may exploit the vulnerabilities found by ARES with particularly-crafted adversarial values. However, ARES investigates a different input space including those intermediate state variables within the RAV controller software, allowing it to find new types of vulnerabilities.

***RAV Monitoring.*** Learning-based approaches [16], [50]–[52] detect malicious behaviors by training a predictive model with RAV operational data collected from benign and malicious runs. However, obtaining large enough training data and encompassing all possible control states are expensive and laborious. State estimation defenses [17], [24], [53], [54] aim to extract and represent the safety-critical RAV state updates and sensor measurements using a state estimator. However, these works mainly focus on addressing the threats of physical channels (i.e., sensors). ARES provides a data-driven search to identify and assess the vulnerable state variables, exposing a wider range of vulnerabilities and sophisticated attack strategies within RAV controller software. Other reference monitor work [55] leverages the relatively stable access patterns of memory-mapped I/O in control algorithms to establish baselines for normal operation and detect anomalies. However, ARES does not modify controller semantics or spoof sensors, which make up the majority of attacks studied in this work. We also find that [56] uses short-term variable correlations for anomaly detection in automated driving contexts. ARES has a different purpose of correlation analysis that we investigate longer-term correlations to identify important variable relationships for further vulnerability analysis in simulation.

***RAV Testing via Fuzzing and Fault Injection.*** Fuzzing-based approaches for control algorithms probe for vulnerabilities by testing a wide variety of inputs. To control for the vast input space, the process is guided with feedback [7] or predefined policies [21]. Unlike these approaches, which focus on single-point modifications, ARES examines longer-term vulnerabilities which may occur over a series of time-dependent states. This results in coverage of different parts of the overall input space, allowing it to find new vulnerabilities. Other approaches leverage intelligent fault injection [57]–[59] to discover vulnerabilities. While similar in methodology, ARES differs at the source of the fault injection. These works typically inject faults directly, manipulating sensors and actuators to disrupt controller operation. In contrast, ARES looks at manipulations further upstream (within controller state variables) which may not be as easy to detect using defense mechanisms.

## VIII. Conclusion

In this work, we present a novel RAV vulnerability assessment framework ARES, which identifies the vulnerable state variables and assesses their impact on the safe operation of the RAV system. We use statistical methods to explore an expanded parameter space. By applying the reinforcement learning-based search approach, we explore the physical impact caused by these vulnerability exploitations in a data-driven way, and show that current memory isolation and control state estimation defenses may struggle against an adaptive adversary with the knowledge that such a system is in use.

## Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) Cyber-Physical (CPS) and Secure and Trustworthy Computing (SaTC) programs, as well as the Department of Energy under Award Number DE-OE0000780, Cyber Resilient Energy Delivery Consortium (CREDC).

## References

[1] DHL parcelcopter launches initial operations for research purposes, Retrieved July 1, 2021, https://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html.

[2] J. Green, Drones Will Elevate Urban Design, Retrieved July 15, 2021, https://www.smartcitiesdive.com/ex/sustainablecitiescollective/drones-will-elevate-urban-design/1053491/.

[3] T. G. 2016, First passenger drone makes its debut at CES, Retrieved July 1, 2021, https://www.theguardian.com/technology/2016/jan/07/first-passenger-drone-makes-world-debut.

[4] Amazon, First Prime Air Delievery, Retrieved July 1, 2021, https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011.

[5] Forbes, Drone Delivery Is Live Today, and It's 90% cheaper than Car-based services, August 18, 2021, https://www.forbes.com/sites/johnkoetsier/2021/08/18/drone-delivery-is-live-today-and-its-90-cheaper-than-car-based-services/?sh=42e7a5204d02.

[6] ArduPilot: versatile, Trusted, Open Autopilot software for drones and other autonomous systems, Retrieved July, 2021, https://ardupilot.org/about.

[7] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "Rvfuzzer: finding input validation bugs in robotic vehicles through control-guided testing," in 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 425–442.

[8] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in 2013 IEEE Symposium on Security and Privacy. IEEE, 2013, pp. 48–62.

[9] T. Kim, C. H. Kim, A. Ozen, F. Fei, Z. Tu, X. Zhang, X. Deng, D. J. Tian, and D. Xu, "From control model to program: Investigating robotic aerial vehicle accidents with {MAYDAY}," in 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020, pp. 913–930.

[10] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! attacking plcs with physical model aware rootkit," in Proceedings of the Network & Distributed System Security Symposium, San Diego, CA, USA, 2017, pp. 26–28.

[11] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing real-time microcontroller systems through customized memory view switching." in NDSS, 2018.

[12] B. Nassi, R. Bitton, R. Masuoka, A. Shabtai, and Y. Elovici, "Sok: Security and privacy in the age of commercial drones," in 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021, pp. 1434–1451.

[13] T. Kim, A. Ding, S. Etigowni, P. Sun, J. Chen, L. Garcia, S. Zonouz, D. Xu, and D. Tian, "Reverse engineering and retrofitting robotic aerial vehicle control firmware using dispatch," in Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys), 2022, pp. 69–83.

[14] S. Etigowni, S. Hossain-McKenzie, M. Kazerooni, K. Davis, and S. Zonouz, "Crystal (ball) i look at physics and predict control flow! just-ahead-of-time controller recovery," in Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 553–565.

[15] PX4 Open Source Autopilot - User Guide, 2021, https://docs.px4.io/master/en/getting_started/sensor_selection.html.

[16] A. Ding, P. Murthy, L. Garcia, P. Sun, M. Chan, and S. Zonouz, "Mini-me, you complete me! data-driven drone security via dnn-based approximate computing," in 24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2021, pp. 428–441.

[17] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 801–816. [Online]. Available: https://doi.org/10.1145/3243734.3243752

[18] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "{SAVIOR}: Securing autonomous vehicles with robust physical invariants," in 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020, pp. 895–912.

[19] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Pattabiraman, "Pidpiper: Recovering robotic vehicles from physical attacks," in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2021, pp. 26–38.

[20] X. Zhou, B. Ahmed, J. H. Aylor, P. Asare, and H. Alemzadeh, "Hybrid knowledge and data driven synthesis of runtime monitors for cyber-physical systems," IEEE Transactions on Dependable and Secure Computing, 2023.

[21] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles," in Network and Distributed System Security Symposium, 2021.

[22] F. Wen, M. Qin, P. V. Gratz, and A. N. Reddy, "Hardware memory management for future mobile hybrid memory systems," IEEE Transactions on computer-aided design of integrated circuits and systems, vol. 39, no. 11, pp. 3627–3637, 2020.

[23] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in 24th {USENIX} Security Symposium ({USENIX} Security 15), 2015, pp. 881–896.

[24] F. Fei, Z. Tu, R. Yu, T. Kim, X. Zhang, D. Xu, and X. Deng, "Cross-layer retrofitting of uavs against cyber-physical attacks," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 550–557.

[25] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Out of control: stealthy attacks against robotic vehicles protected by control-based techniques," in Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 660–672.

[26] Y. Xia, Y. Liu, H. Chen, and B. Zang, "Cfimon: Detecting violation of control flow integrity using performance counters," in IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). IEEE, 2012, pp. 1–12.

[27] ArduPilot, Complete parameter list of ArduCopter, 2021, https://ardupilot.org/copter/docs/parameters.html.

[28] Bitcraze, Crazyflie 2.1, 2021, https://www.bitcraze.io/products/crazyflie-2-1.

[29] T. Kim, V. Kumar, J. Rhee, J. Chen, K. Kim, C. H. Kim, D. Xu, and D. J. Tian, "Pasan: Detecting peripheral access concurrency bugs within bare-metal embedded applications." in USENIX Security Symposium, 2021, pp. 249–266.

[30] P. Sun, L. Garcia, and S. Zonouz, "Tell me more than just assembly! reversing cyber-physical execution semantics of embedded iot controller software binaries," in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2019, pp. 349–361.

[31] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in Noise reduction in speech processing. Springer, 2009, pp. 37–40.

[32] S. C. Johnson, "Hierarchical clustering schemes," Psychometrika, vol. 32, no. 3, pp. 241–254, 1967.

[33] D. Pelleg, A. W. Moore et al., "X-means: Extending k-means with efficient estimation of the number of clusters." in Icml, vol. 1, 2000, pp. 727–734.

[34] H. Akaike, "A new look at the statistical model identification," IEEE transactions on automatic control, vol. 19, no. 6, pp. 716–723, 1974.

[35] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," arXiv preprint arXiv:1608.05742, 2016.

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[37] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[38] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," arXiv preprint arXiv:1606.01540, 2016.

[39] Mantis Q, 2022, https://us.yuneec.com/mantis-q-robust-travel-drone-with-voice-control.

[40] R. Ourari, K. Cui, A. Elshamanhory, and H. Koeppl, "Nearest-neighbor-based collision avoidance for quadrotors via reinforcement learning," in 2022 International Conference on Robotics and Automation (ICRA). IEEE, 2022, pp. 293–300.

[41] L. He, N. Aouf, J. F. Whidborne, and B. Song, "Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance," in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 7491–7497.

[42] D. Wang, T. Fan, T. Han, and J. Pan, "A two-stage reinforcement learning approach for multi-uav collision avoidance under imperfect sensing,"

IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 3098–3105, 2020.

[43] S. He, Y. Wang, S. Han, S. Zou, and F. Miao, “A robust and constrained multi-agent reinforcement learning framework for electric vehicle amod systems,” arXiv preprint arXiv:2209.08230, 2022.

[44] A. Machiry, C. Spensky, J. Corina, N. Stephens, C. Kruegel, and G. Vigna, “Dr. checker: A soundy analysis for linux kernel drivers.” in USENIX Security Symposium, 2017, pp. 1007–1024.

[45] F. Wen, M. Qin, P. Gratz, and N. Reddy, “Openmem: Hardware/software cooperative management for mobile memory system,” in 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021, pp. 109–114.

[46] D. Davidson, H. Wu, R. Jellinek, V. Singh, and T. Ristenpart, “Controlling uavs with sensor input spoofing attacks.” in WOOT, 2016.

[47] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, “Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks,” in 2017 IEEE European symposium on security and privacy (EuroS&P). IEEE, 2017, pp. 3–18.

[48] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, “On the requirements for successful gps spoofing attacks,” in Proceedings of the 18th ACM conference on Computer and communications security, 2011, pp. 75–86.

[49] H. Kim, M. O. Ozmen, Z. B. Celik, A. Bianchi, and D. Xu, “Pgpatch: Policy-guided logic bug patching for robotic vehicles,” in 2022 IEEE Symposium on Security and Privacy (SP). IEEE, 2022, pp. 1826–1844.

[50] Y. Chen, C. M. Poskitt, and J. Sun, “Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system,” in 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018, pp. 648–660.

[51] F. Fei, Z. Tu, D. Xu, and X. Deng, “Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks,” in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 7358–7364.

[52] K. N. Junejo and J. Goh, “Behaviour-based attack detection and classification in cyber physical systems using machine learning,” in Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security, 2016, pp. 34–43.

[53] P.-J. Bristeau, E. Dorveaux, D. Vissière, and N. Petit, “Hardware and software architecture for state estimation on an experimental low-cost small-scaled helicopter,” Control Engineering Practice, vol. 18, no. 7, pp. 733–746, 2010.

[54] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, and L. Sha, “Securecore: A multicore-based intrusion detection architecture for real-time embedded systems,” in 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2013, pp. 21–32.

[55] A. Khan, H. Kim, B. Lee, D. Xu, A. Bianchi, and D. J. Tian, “M2mon: Building an mmio-based security reference monitor for unmanned vehicles.” in USENIX Security Symposium, 2021, pp. 285–302.

[56] Z. Zhang, S. S. V. Singapuram, Q. Zhang, D. K. Hong, B. Nguyen, Z. M. Mao, S. Mahlke, and Q. A. Chen, “Avmaestro: A centralized policy enforcement framework for safe autonomous-driving environments,” in 2022 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2022, pp. 1333–1339.

[57] S. Jha, S. Banerjee, T. Tsai, S. K. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, “Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection,” in 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 2019, pp. 112–124.

[58] M. Moradi, B. J. Oakes, M. Saraoglu, A. Morozov, K. Janschek, and J. Denil, “Exploring fault parameter space using reinforcement learning-based fault injection,” in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2020, pp. 102–109.

[59] X. Zhou, A. Schmedding, H. Ren, L. Yang, P. Schowitz, E. Smirni, and H. Alemzadeh, “Strategic safety-critical attacks against an advanced driver assistance system,” in 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2022, pp. 79–87.