PID-Piper: Recovering Robotic Vehicles from Physical Attacks

Pritam Dash*, Guanpeng Li†, Zitao Chen*, Mehdi Karimibiuki*, Karthik Pattabiraman*
*University of British Columbia
{pdash, zitaoc, mkarimib, karthikp}@ece.ubc.ca
†University of Iowa
guanpeng-li@uiowa.edu

***Abstract*—Robotic Vehicles (RV) rely extensively on sensor inputs to operate autonomously. Physical attacks such as sensor tampering and spoofing can feed erroneous sensor measurements to deviate RVs from their course and result in mission failures. In this paper, we present *PID-Piper*, a novel framework for automatically recovering RVs from physical attacks. We use machine learning (ML) to design an attack resilient Feed-Forward Controller (FFC), which runs in tandem with the RV's primary controller and monitors it. Under attacks, the FFC takes over from the RV's primary controller to recover the RV, and allows the RV to complete its mission successfully. Our evaluation on 6 RV systems including 3 real RVs shows that *PID-Piper* achieves high accuracy in emulating the RV's controller, in the absence of attacks, with no false positives. Further, *PID-Piper* allows RVs to complete their missions successfully despite attacks in 83% of the cases, while incurring low performance overheads.**

***Index Terms*—Cyber Physical Systems (CPS), Robotic Vehicle Security, Attack, Detection, Resilience**

I. Introduction

Autonomous Robotic Vehicles (RVs) such as Unmanned Aerial Vehicles (UAVs, also known as drones) and Unmanned Ground Vehicles (UGVs, also known as rovers) are widely deployed in a variety of industrial sectors *e.g.*, agriculture, surveillance, package delivery, warehouse management, and space exploration [1]–[5]. RVs rely extensively on their on-board sensor measurements for autonomous operations.

Unfortunately, RVs have been shown to be vulnerable to *physical attacks*. These are attacks that maliciously perturb RV sensor measurements through physical means. Common physical attacks against RVs are GPS spoofing by transmitting false GPS signals [6], [7], and gyroscope [8], and accelerometer [9] tampering through acoustic noise injection. Physical attacks can have severe consequences such as crashing the RV, or significantly deviating it from its course and preventing it from reaching its destination, resulting in mission failure [10].

There have been many techniques proposed for detecting physical attacks in RVs [10]–[13]. However, upon detecting the attack, these techniques either raise an alarm and trigger the fail-safe modes of the RV (*e.g.*, forced landing for drones), or require manual remediation [14]. Unfortunately, this often leads to failure of the RV's mission (*i.e.*, RV does not reach its destination, or crashes). Because RVs are projected to be deployed in mission-critical tasks such as drug delivery [3], [15] and disaster relief [16], they need to recover from physical attacks (henceforth, by attacks we mean physical attacks), and complete their missions. Therefore, the recovery technique should steer the RV to its destination and allow it to *complete its mission*, without requiring any manual remediation.

In this paper, we propose *PID-Piper*[1], a framework for automated attack recovery in RVs by using a secondary controller in tandem with the RV's primary controller. PID (proportional-integral-derivative) control is the primary controller in autonomous RVs [17]. We design *PID-Piper* based on two observations we made. First, PID control is designed to handle faults such as sensor noise and environmental disturbances by compensating for the resulting errors (*e.g.*, increase thrust to minimize drift due to wind). However, it *over-compensates* for the sensor perturbations due to attacks, which are systematic in nature, causing the RV to deviate from its course and result in mission failures. Secondly, PID controllers perform a series of calculations based on the RV's inputs (*e.g.*, target position) and sensor measurements (*e.g.*, linear and angular position) to derive the actuator signals. However, these calculations are very sensitive to changes in the sensor's inputs, which often cause large variation in the outputs (*i.e.*, actuator signals). This is due to the *high collinearity* among the input parameters.

PID-Piper addresses the above weaknesses of PID controllers through two innovations. First, it uses a *Feed-Forward Controller (FFC)*, as opposed to a Feed-Back Controller (FBC) (used by all the prior work in this area [10], [13], [14]). Unlike an FBC, which relies on the primary PID controller to correct the RV's trajectory, an FFC predicts the potential disturbances (due to attacks) and directly rectifies the RV's trajectory to compensate for them. This allows it to avoid the over-compensation problem of PID. Second, we use Machine Learning (ML) to learn the appropriate model for predicting the PID controller's output, and we use *feature engineering* to avoid the high collinearity problem of PID. Thus, *PID-Piper* can accurately predict the behavior of the PID controller under normal (attack free) operation, while being resilient to attacks.

PID-Piper runs the ML-based FFC in tandem with the PID controller, and monitors the deviation between both the controllers. If this deviation exceeds a pre-defined threshold, it signals an attack, and switches the RV's output from the

[1] *PID-Piper* leads RV towards their mission targets despite attacks, and hence the name (pronounced Pi(e)d Piper).

PID controller to the ML-based FFC until the attack subsides. This allows the RV to recover from the attack, and complete the mission successfully. We call these attacks *overt attacks* as they cause explicit disruptions in RV missions.

However, attackers can also mount *stealthy attacks* that cause deviations in the RV's trajectory in a controlled manner, and over time can cause significant disruptions in RV missions [18]. Because such attacks cause controlled deviations under the threshold, they will not trigger detection, and hence recovery is not possible. Compared to prior work, *PID-Piper* derives a precise model of the RV using ML techniques, and hence sets tight detection thresholds. This allows it to significantly limit the effects of stealthy attacks.

To the best of our knowledge, PID-Piper *is the first technique that automatically recovers the RVs from overt attacks, and limits the effects of stealthy attacks, to achieve mission success.*

Our contributions in this paper are:

- We propose the use of FFCs for detecting and recovering from physical attacks against RVs, which directly rectifies the actuator output based on the model's predictions.
- We design a ML-based model for the FFC, and use feature engineering to make it resilient to attacks.
- We design *PID-Piper*, a framework to integrate the ML-based FFC controller with the PID controller. *PID-Piper* uses the latter's output in the absence of attacks. However, it monitors the deviation between them, and switches to the former's output if the deviation exceeds a specified threshold, thereby detecting an attack.
- We evaluate *PID-Piper* on 6 RVs - 3 simulated systems, and 3 real RVs (2 drones and a rover) on a wide range of missions, and subject them to overt and stealthy attacks.

We find that (1) *PID-Piper* achieves high accuracy in predicting the RV's runtime behaviour *i.e.*, comparable to the PID controller, during normal operation (in the absence of attacks), and incurs 0% false positives. (2) *PID-Piper* successfully recovers the RV under overt attacks, and completes the mission in about 83% of cases (on average). while prior work [14] does so in only 13% of the cases. Further, *PID-Piper* incurs neither crashes nor stalls of the RV, unlike prior work that does. (3) *PID-Piper* limits the impact of stealthy attacks on the RV by a factor of 7, compared to the next best technique [13], and ensures mission success in 100% of cases (prior work has 0% success rate), and (4) *PID-Piper* incurs less than 7% performance overhead, with no increase in mission times.

II. Background and Threat Model

In this section, we first discuss the various operations and modules of RV's autonomous control logic. Then we discuss overt and stealthy attacks against RVs, followed by the limitations of existing techniques. Finally, we present the differences between FFC and FBC, and the threat model.

A. Robotic Vehicle Control

In this paper, we focus on commodity RVs *e.g.*, delivery drones, warehouse rovers etc., operating autonomously. An RV uses many sensors (*e.g.*, GPS, barometer, gyroscope, accelerometer, and magnetometer) to capture its current physical state (*e.g.*, angular and linear position), which are used to estimate the actuator signals (*e.g.*, motor commands) for positioning the vehicle in the next state.

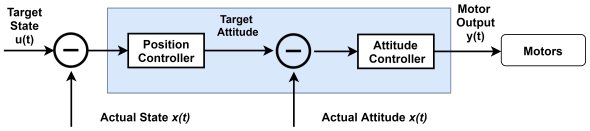


Fig. 1: RV's Autonomous Control Logic

RVs use PID control for estimating position (*e.g.*, altitude, latitude, longitude), and attitude (*e.g.*, yaw, roll, pitch), as well as correcting position and attitude errors during the RV mission. Figure 1 (based on ArduPilot [19]) shows an example of a RV controller. RV controllers consist of two parts. (i) The position controller takes the target states as input (*i.e.*, target position, target velocity, position error, and actual velocity), and calculates the corresponding velocity, acceleration and finally, the target angles (*i.e.*, yaw, roll, pitch). (ii) The attitude controller takes the target attitude as input (*i.e.*, target angles, target angular velocity, angular error, and actual angular velocity), and calculates the rotation rates and the corresponding high level motor commands. In addition to the inputs shown in the figure, the RV controller relies on the P, I, D coefficients that track the control signals and history of errors between target and actual state. The RV controller (*i.e.*, PID controller) then performs a series of calculations (*e.g.*, Target Position→Velocity→Acceleration→Target Angles. Target Angles→Angular Rotation→rotation rate) to derive the target angles and motor outputs.

B. Attacks Against RVs and Defense

As RVs rely on sensor measurements for autonomous operation, attacks on sensors can be debilitating for the RV. *Physical attacks* manipulate sensor measurements through physical means, *e.g.*, gyroscope measurements can be manipulated through acoustic noise injection [8], and GPS measurements can be manipulated by transmitting malicious GPS signals [6]. Physical attacks are often launched by injecting false data (a bias value) to raw sensor measurements [10], [13], [18]. There are two kinds of physical attacks: (1) Overt, and (2) Stealthy.

Overt attacks inject large bias f into sensor measurements $x : x = x + f$ to cause an immediate disruption in the RV's mission (*e.g.*, drastically deviate a drone from its trajectory and/or crash it) [8], [9]. Real-time invariant analysis have been proposed for detecting overt attacks against RVs [10], [11], [13]. Typically, invariant analysis techniques derive a model correlating the RV's sensor inputs with the actuator outputs. Based on the RV's current state (*e.g.*, linear and angular position), the model estimates the system's real-time behaviour (*i.e.*, next state). At runtime, if the error between observed values V_o and model estimated output V_p is $>$ a predefined threshold τ, an alarm is raised. To prevent false alarms due to transient effects, the comparison is performed in a time window as shown in the following equation.

$$D(t) = true, \quad if \sum_{t_i}^{t_j} |V_p - V_o|_n > \tau. \tag{1}$$

Unfortunately, attackers can exploit the above attack detection strategy to mount *stealthy attacks* [18]. Assuming the attacker is able to determine τ, she can trigger stealthy attacks in a controlled manner where $|V_p - V_o|$ never exceeds τ. Because the error remains under the threshold τ at any time, the attack remains undetected.

In general, the higher the threshold, the greater the disruption that a stealthy attack can cause. Model based techniques heavily rely on the model's accuracy to determine a precise estimate of the RV's real-time behaviour. If the model fails to accurately estimate the RV's dynamics (*e.g.*, position, attitude), the system will have to tolerate significant errors to avoid false positives, and hence set a high detection threshold. Prior work [18] has demonstrated that stealthy attacks on such models, when performed over a long time, can cause substantial deviations in the RV's trajectory.

C. Limitations in Existing Techniques and Motivation

Attack detection techniques for RVs such as Control Invariants (CI) [10] and Savior [13], enable fail-safe mode (*e.g.*, landing for drones) once an attack is detected. However, they cause the mission to be aborted upon detection of an attack. As RVs are often deployed in critical missions, it is important to recover from the attacks and complete the mission successfully. *We define a successful mission as one in which the RV reaches the planned destination without crashing*.

A recent technique for recovering RVs from attacks is software sensor based recovery (SRR) technique [10]. SRR uses system identification to construct a model of the RV that considers controller, actuators and vehicle dynamics. The system model predicts the RV's next state given the current state and target position. The software sensors (*i.e.*, set of programs that emulate the real sensors such as GPS, gyroscope, accelerometer) take the system model's predictions as input, and derive the measurements similar to the real sensors. A recovery monitor observes the real sensors and switches to the software sensors if the difference between the software sensors and real sensor measurements increases above a predefined threshold, thereby signifying an attack (Equation 1).

However, there are two issues with SRR. First, SRR detects only abrupt fluctuations in the RV's trajectory due to overt attacks, and prevents crashes by transitioning the RV to an emergency state for a short time. This means that upon an attack detection, the RV will be placed in a holding state, and will require *manual intervention* to successfully complete the mission. As we show later (Section VI), SRR achieves low mission success rates in RVs without any manual intervention.

Second, SRR uses a linear state-space model to estimate the real-time behaviour of RVs. As RVs are non-linear systems [13], SRR fails to accurately estimate the RV's dynamics, as a result of which, it is vulnerable to stealthy attacks. Such stealthy attacks can result in significant deviations of the RV, and also lead to mission failures (Section II-B).

D. Feed-Forward vs Feed-Back Control

In non-linear control systems such as RVs, there are two ways to handle errors due to sensor or environmental noise, and predict actuator outputs [20]. Feed-Back control (FBC), or reactive control, measures the error between the target and actual parameters (Figure 1) and derives appropriate control inputs to minimize the error. Since all of the noise affecting an RV is not known apriori (*e.g.*, friction, wind, payload etc.), FBC measures the error during operation and minimizes its impact. Therefore, it does not require a precise system model.

Feed-Forward control (FFC), or predictive control, requires a precise model of the system and the noise affecting it. FFC uses a noise model to predict the error between the target and actual parameters, and estimates the system's response. FFC directly modifies the control input to account for the error predicted by the noise model, and prevents the modeled errors from causing large fluctuations in the actuator output.

Prior techniques that detect attacks against RVs [10], [13], and the only other recovery technique for RVs, SRR [14], use an FBC alongside the RV's PID controller. To the best of our knowledge, no prior work has evaluated the appropriate controller design for recovery from attacks in RVs. Therefore, we evaluate which of the 2 techniques is better for minimizing the effects of sensor perturbation due to attacks, and recover the RVs (Section IV). Though it has been observed that an FFC is better at rejecting sensor noise and correcting it than an FBC [21], it is not clear how it performs under attacks.

E. Threat Model

We focus on physical attacks that maliciously perturb RV's sensor measurements to cause deviations or disruptions in its mission. These can be either overt or stealthy attacks. We assume that the attacker has the following capabilities: (1) Perform sensor spoofing attacks on GPS or manipulate IMU (gyroscope, accelerometer, magnetometer etc.) sensors of the RV. (2) Snoop on the communication channel between the RV and the ground control station (GCS), as well as on sensor measurements, control inputs and outputs of the RV. The attacker can also arbitrarily manipulate sensor readings to her desired values, and at any time during the RV's mission.

However, we exclude attacks that result in persistent drastic sensor manipulations, as well as attacks that target all the sensors simultaneously. These attacks have been shown to be hard to mount in real world RV mission [8]. Similar assumptions have been made in prior work [13]. We also exclude cyber attacks that target software components or firmware, as they can be handled by existing techniques [22].

Further, we assume that the attacker does not have the following capabilities: (1) write access to the firmware, (2) root access to the Operating System (OS). Hence, she cannot bypass *PID-Piper*'s checking. Finally, we assume that the attacker may not poison the training data for the ML models.

III. Initial Study

Our goal is to provide recovery for RVs by predicting the RV's actuator signals under attacks, and make them

complete their mission successfully. We present a preliminary experimental study to understand how attack induced sensor perturbations cause fluctuations in the PID controller's outputs.

In the experiment, we performed a mission on a Pixhawk based drone (Section VI has details), with the following mission command Arm $\rightarrow$ Takeoff $\rightarrow$ Waypoint $\rightarrow$ Land. Once the drone attains $5m$ elevation and cruises towards the destination, we launched a GPS spoofing (overt) attack. The attack is triggered intermittently for 3-5 seconds throughout the mission, which lasts for a total of 60 seconds.

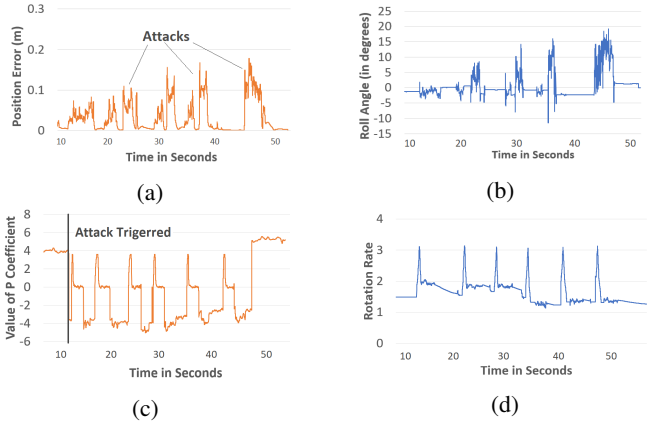


Fig. 2: Parameters of Pixhawk drone under GPS manipulation attack (a) Position error, (b) Fluctuations in Roll angle, (c) P coefficient adjustment, (d) Change in rotation rate

High Collinearity in PID parameters Figure 2 shows the changes in various parameters in the Pixhawk drone under the attack. Figure 2a shows the position error incurred by the PID controller due to the attack, while Figure 2b shows the corresponding output *i.e.*, roll angle estimation. The figures show only the straight line part (*i.e.*, the steady state) of the mission, where the change in roll angle should be near 0 if there is no position error (*i.e.*, without any attacks), as the drone is not making any turns. However, as can be seen in Figure 2a, though the position error pE due to the attack is small (*i.e.*, between 0 to 0.2 m), the corresponding fluctuations in the roll angle y are large (*i.e.*, between -10 to 20 degrees), which deviates the drone significantly from its course.

The observations indicate that the PID controller is highly sensitive to continuous variance inflation in sensor measurements. We identified the reason for this behaviour as *high collinearity in the PID controller's parameters*. The PID controller performs a series of calculations based on inputs such as sensor measurements and target state, to derive intermediary parameters such as velocity, acceleration, etc., and estimates the Euler angles (roll, pitch, yaw) (Section II-A). Collinearity means that one or more of these parameters are highly correlated with each other, and so a change in any one parameter leads to changes in all of them [23].

We assess the degree of collinearity between the PID's parameters using the Variance Inflation Factor (VIF) [23] metric, which measures variance increase in the output parameter due to collinear input parameters. VIF(x_1) is calculated by regressing parameter x_1 against every other parameter $x_2..x_n$. A VIF value close to 1 indicates low collinearity, while a VIF value greater than 10 indicates high collinearity [23].

We find that position variance, linear and angular position have the lowest VIF values (ranging from 1 to 1.6), while velocity, acceleration, angular rotation and angular speed have the highest VIF values (clustering between 22 to 29). The parameters with high VIF values experience large fluctuations due to small variance inflation in pE, consequently triggering a cascading variance inflation in the operations of the PID controller (Section II-A). *Therefore, high collinearity in the PID controller's parameters triggers large fluctuations in its outputs due to the attack induced sensor perturbations*

Overcompensation Issue in PID The PID controller derives the actuator output by attempting to reduce the error between the target state u and the actual state x (Figure 1). For example, when a drone is cruising in steady state, PID derives the rotation rate r : $r = P * (u - x)$, and ensures that r remains constant by adjusting the coefficient P. Figure 2c shows the P coefficient adjustments under the attack shown in Figure 2a which induced position error. Before the attack started ($t < 10$), the PID controller sets the P coefficient to 4 to derive r. As the position error increases starting from $t = 11$ due to the attack (Figure 2a), we can see in Figure 2c that the PID controller persistently adjusts the P coefficient in order to maintain r at a constant rate. As a result of the persistent adjustments to the P coefficient, as can be seen in Figure 2d, the rotation rate increases from 1.6 to 3.2 starting at $t = 11$. The increase in rotation rate results in increasing the thrust, which drifts the drone from its original trajectory.

The PID controller is designed to handle sensor noise and disturbances such as wind, by adjusting the set of coefficients (namely P, I, and D), and by modifying the controller's inputs to increase/decrease the thrust in order to compensate for the error. However, the errors between u and x due to sensor noise are typically transient in nature unlike attacks, which induce the error systematically. *Therefore, systematically induced sensor perturbations due to attacks cause overcompensation in the PID controller, resulting in erroneous actuator signals.*

IV. Design

In this section, we present the various design choices we make to build an attack resilient controller for RVs by addressing the issues discussed in Section III.

A. Modeling Robotic Vehicles

There are two options for building a resilient controller. The first option is to modify the original PID controller to address the high collinearity and over-compensation issues. The second option is to add a secondary controller in tandem with PID, and overcome the aforementioned issues.

We choose the second option, namely adding a secondary controller, for two reasons. First, modifying the PID controller may affect its fault tolerance capabilities because collinear parameters and compensation for errors is inherent in its

design. Second, PID control is a well-defined mathematical model containing differential equations, and radical changes to the PID controller may result in erroneous estimations.

Therefore, we design a secondary controller for RVs that monitors the PID controller's output at runtime, and takes over the autonomous control only when an attack is detected. When the attack subsides, the control is switched back to the PID controller (until another attack is detected or the mission ends).

We use ML to build the secondary controller, and train the model to emulate the PID controller (in the absence of attacks). We choose ML based models for the following reasons: First, the diversity and flexibility of ML techniques allows us to design an appropriate model of input/output relationships for non-linear RV systems. Second, we can leverage ML design principles to build a model that emulates PID controller while being resilient to malicious sensor perturbations. In particular, our ML model learns the temporal relationships in input parameters to achieve high accuracy, and we use feature engineering in the ML model design to eliminate the parameters that contribute to high collinearity.

Because the PID controller uses multiple, independent input parameters to derive the corresponding actuator signals, we formulate the ML model predictions as a regression problem. The PID controller's input is a multivariate time-series. Therefore, we use Long Short Term Memory (LSTM) architecture, which is well-suited to model temporal patterns and long-term dependencies respectively [24].

B. Controller Design

Recall that the secondary controller can be built either based on FFC or FBC designs (Section II-D). Prior work for attack detection and/or recovery in RVs [10], [13], [14] modeled a physical process *i.e.*, given the current physical state of the RV, what will be the next state. Because their goal was to predict physical states of the RV under attack, they use state space models and/or Kalman filters, which predicts the RV's physical dynamics in a feedback control loop (*i.e.*, FBC). However, ML techniques can model both the physical dynamics (*i.e.*, correlate previous state to next state) as well as the RV's control operations (*i.e.*, correlate target state to actuator output). Thus, this flexibility in ML models allow us to consider both FFC and FBC in our design.

We design FFC and FBC using the same ML technique *i.e.*, LSTM. The FFC design, learns a model $\mathcal{F}_\theta$ that predicts the actuator output $y'(t)$ (angular rotation, Euler angles) given the current state $x(t)$ (linear and angular position), and the target state $u(t)$ (target position) as inputs. The FBC design, on the other hand, learns a model $\mathcal{F}_\psi$ to predict the current state of the RV $x'(t)$ given the output of the previous state $y(t-1)$ and $u(t)$ as inputs. The FBC predicted $x'(t)$ is used by the PID controller to derive $y(t)$. *Therefore, FFC models the system to predict the output of the PID controller, while FBC models the system to predict one of the inputs of the PID controller.* Figure 3 shows difference between the two designs.

The steps involved in building FBC and FFC using ML are:

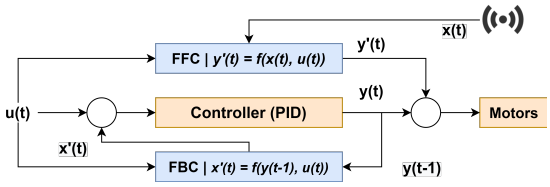


Fig. 3: FBC and FFC controller design

(1) *Data Collection*: We collect a diverse set of RV mission profile data from both simulated and real RVs. We collect sensor measurements (GPS, gyroscope, accelerometer, barometer), data related to RVs linear and angular state in X, Y, Z axis (linear position, position variance, angular position, angular speed, velocity, acceleration etc.), and the outputs of position and attitude controller (Figure 1). We also collect the PID specific coefficients that track the control signals.

(2) *Feature Selection*: In this step, we select a set of meaningful features for training ML models. Note that FBC takes $y(t-1)$ and $u(t)$ as inputs, while FFC takes $x(t)$ and $u(t)$ as inputs. Therefore, the feature sets for both the controller designs are different. The ML model for FBC contains 12 features (*e.g.*, target position, Euler angles, velocity, acceleration, angular rotation etc.), while the ML model for FFC contains 44 features (*e.g.*, target position, position variance, linear position, angular position, angular speed, IMU measurements etc.). We use a greedy approach for feature subset selection. We start with having a single feature in the model, and on every iteration we add a new feature, and measure the model accuracy. We stop when the accuracy saturates.

(3) *Model Training*: Finally, we train two LSTM models to learn $\mathcal{F}_\theta$ and $\mathcal{F}_\psi$ with the respective feature sets. Both the models have 2 layer stacked LSTM design, a Sigmoid neural net layer followed by 2 fully connected PRelu layers. We save the inputs of the previous steps in LSTM cells. This helps the model learn the temporal correlations between the inputs, which is necessary for accurate modeling of transition states in the RV (*e.g.*, steady state to landing in drones).

C. Analyzing FBC and FFC Designs

We perform three experiments to analyse the effectiveness of FBC and FFC design with and without attacks. We use the same Pixhawk drone as before (Section III) for all three experiments, navigating via 3 points, A, B and C. In the first experiment, no attacks are considered. The Pixhawk drone navigates from location A→B in steady state, makes a 150 degree turn, then navigates towards location C.

We use the Mean Absolute Error (MAE) $MAE = \frac{1}{n}\sum_{t=1}^{n}|y_{PID} - y_{ML}|$ to compare ML predictions with the PID's predictions. The MAE measures the average magnitude of the errors between the ML predictions and the PID's. Note that FBC does not predict the actuator signals directly (Figure 3). Therefore, in the case of FBC, y_{ML} refers to PID output while using ML predicted $x'(t)$.

We compare the roll angle predictions of FFC and FBC with the roll angle estimation of the PID controller with and

without attacks. In the absence of attacks, the MAE for both FBC and FFC is under 1 degree, which is considered as good accuracy in time series predictions for non-linear systems [13], [14]. Therefore, in the absence of attacks, both FBC and FFC are equally effective in predicting the RV's realtime behaviour.

In the second experiment, we perform attacks similar to Figure 2a during a drone mission. We use the PID's roll angle estimations in the previous mission (without attacks) as the baseline, and analyse the errors in PID, FBC and FFC outputs under an attack. For the PID, the MAE under attack was found to be 8.09, while for FBC and FFC, the MAE values were 6.16 and 5.85 respectively. Thus, the FFC is slightly better than the FBC in predicting the RV's trajectory under attacks.

In the third experiment, we use *feature engineering* in the ML model design [25] to address high collinearity, and subject them to attacks. Recall that parameters with high VIF values experience high fluctuations under attacks even if the variance inflation is small (Section III). To reduce the collinearity in FBC, we eliminate the parameters in the set $y(t-1)$ (*e.g.*, inertial velocity and angular rotation) that have high VIF values, and learn a model (Equation 2), selecting the minimum number of $y(t-1)$ parameters to achieve high accuracy.

$$x(t) = \min_{y} \mathcal{F}_{\psi}(y(t-1), u(t)) \tag{2}$$

Similarly, to reduce collinearity in FFC, we eliminate parameters in the set $x(t)$ (*e.g.*, GPS values, raw IMU sensor values, IMU velocity and acceleration in X, Y, Z axis) that have high VIF values, and learn a model (Equation 3) selecting the minimum number of $x(t)$ parameters to achieve high accuracy.

$$y'(t) = \min_{x} \mathcal{F}_{\theta}(x(t), u(t)) \tag{3}$$

The pruned feature set for FBC contains 6 parameters such as target states, Euler angles, and angular states. On the other hand, the pruned feature set for FFC contains 24 parameters that capture the RV's linear and angular positions such as target position, position error, position variance, angular position, angular orientation, angular speed etc.

We again run the above mission under attacks. We observed that the MAE for both the techniques is lower after feature engineering. Further, we find that the MAE of FFC was 0.86, while that of FBC was 3.91. Therefore, *the FFC with feature engineering is much more accurate than FBC with feature engineering in predicting the RV's trajectory under attacks.*

D. Addressing Over-Compensation with FFC Design

The FFC is more accurate in predicting RV's trajectory under attacks than FBC because FFC is effective in preventing sensor perturbations from influencing its output. Recall that the LSTM model in both FBC and FFC designs uses memory cells to remember temporal relationships between inputs. We leverage the memory cells to build a noise model for both FFC and FBC which correlates past and present inputs and minimizes the effects of sensor perturbations. Because FFC and FBC have different inputs and outputs (Figure 3), the noise model in FFC minimizes the influence of attack induced fluctuations in $x(t)$ on model output $y'(t)$, whereas, the noise model in FBC minimizes the influence of attack induced fluctuations in $y(t-1)$ on model output $x'(t)$. Note that the noise model cannot eliminate the error completely, it can only minimize the error in both FFC and FBC's output.

Recall that in FBC design, the ML model predicts the RV's current state $x'(t)$, and finally the PID controller predicts the actuator signal $y(t)$ using the ML predicted $x'(t)$ along with $u(t)$ (Figure 3). As FBC relies on the PID controller for deriving the actuator signal, and because FBC cannot predict $x'(t)$ with 0 error under attacks, PID overcompensates for the error in $x'(t)$ while deriving $y(t)$ (Section III). On the other hand, in FFC design, the ML model predicts the actuator signal $y'(t)$ independently based on the given input set *i.e.*, $x(t)$ and $u(t)$. Thus, even if the FFC and FBC designs have the same LSTM design and a noise model, FBC still suffers from the overcompensation issue, whereas, FFC prevents overcompensation.

The steps involved in building the noise model leveraging memory cells in LSTM are as follows. We model the relationship between past and present input parameters in the first layer of the LSTM which uses a Sigmoid activation function (Sigmoid layer). At each instant t, the Sigmoid layer examines the past inputs $X(k)$ ($X(k) = x(t-1)..x(t-k)$) in the memory cell and the present inputs $x(t)$, and outputs a value between 0 and 1 for each feature (f_t). If the variance between the $X(k)$ and $x(t)$ is high (under an attack), the Sigmoid layer outputs a value close to 0. If the variance is low, the Sigmoid layer outputs a value close to 1. By modeling the relationship between $X(k)$ and $x(t)$, the Sigmoid layer controls the weight of each feature in deriving the actuator signals. Thus, it prevents sensor perturbations due to attacks from propagating to the next layer, and influencing the output.

In summary, the FFC design with feature engineering in the ML model, is much more effective in minimizing the output fluctuations under attacks. Thus, *PID-Piper* uses a secondary controller based on FFC design in tandem with PID controller.

V. *PID-Piper* Architecture and Algorithm

Figure 4 shows the architecture of *PID-Piper*. *PID-Piper* consists of 3 main components (labeled in the figure and shown in blue): ❶ a FFC-based ML model, ❷ a monitoring module, and ❸ a recovery module. The ML model runs in

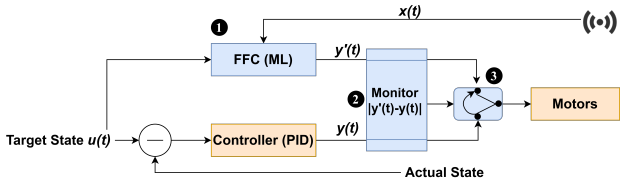


Fig. 4: *PID-Piper* architecture.

tandem with the PID controller, and predicts actuator signals $y'(t)$ based on the target state $u(t)$ and current state $x(t)$. In the absence of attacks, the PID controller's output is used to derive $y(t)$. If the error between the 2 controllers $\delta = |y'(t) - y(t)|$

exceeds a preset threshold τ, the monitoring module detects an attack and enters the recovery mode by switching RV's autonomous navigation from the PID controller to the ML model's output. When the attack stops, the recovery module switches the navigation back to the PID controller's output.

PID-Piper relies on a threshold based monitoring to detect attacks and trigger recovery. Recall that the ML model attempts to predict the PID's output. However, because the ML model is more complex than the PID, the model's predictions may incur a small latency, and lag behind the PID controller's operation. To account for the delay, we use dynamic time warping (DTW) [26] to calculate the temporal difference between the time series PID estimates and ML predictions, and calculate an optimal match between them [10]. We record the observed temporal deviations between the two time series (*i.e.*, PID estimations and ML predictions) in the absence of attacks over multiple missions. The error accumulated ($\tau = \sum_{t=i}^{t=j} |y_{PID} - y_{ML}|$) in the highest recorded temporal deviation across the validation sets is chosen as the threshold.

We consider the effect of overt and stealthy attacks below.

1) Overt Attacks: PID-Piper limits the impact of overt attacks by providing recovery, which minimizes the trajectory deviations as a result of the attacks (*e.g.*, navigates RVs back on course) and prevents undesirable outcomes such as crashes. To trigger recovery, the first step is to detect an overt attack based on the divergence between the PID and ML predictions.

As ML provides approximations of the PID's output, it is necessary to differentiate the divergence due to attacks from natural ones. For this purpose, we use the Cummulative Sum (CUSUM) technique. Choi *et al.* [14] used a time-window to keep track of the anomaly and raise an alert if the residuals during the time window exceed a given value. However, later work [13], [27] has shown that CUSUM outperforms the time-window strategy as it prevents the attacker from hiding her attack between windows. So we use CUSUM in our work.

Algorithm 1 shows the algorithm used by *PID-Piper* to detect and recover from overt attacks. *PID-Piper* monitors the PID and ML model's outputs based on the RV's current state throughout the mission (Line 11 to 30). *PID-Piper* tracks the difference between both the predictions based on the RV's current state (Line 12 to 13) following CUSUM statistics, which is described as: $S_{t+1} = S_t + |y_{ML}(t) - y_{PID}(t)| - b(t)$, where $S(0) = 0$ and $b(t) > 0$ are selected to prevent S_t from increasing due to transient difference between y_{ML} and y_{PID}. When an overt attack occurs, the S_{t+1} value (Line 15), will exceed the threshold τ; such deviations will be flagged as attacks, and *PID-Piper* will trigger the recovery mode (Line 16). When the recovery mode is triggered, the ML predicted output (Line 20) will be used to derived motor thrust (Line 29), instead of the PID's outputs. If the attack stops, the error between y_{PID} and y_{ML} will go back to near 0 (Line 21 to 22). In that case, *PID-Piper* will turn the recovery mode off (Line 23), and switch back to the PID's outputs (Line 24).

2) Stealthy Attacks: In a stealthy attack, the attacker's goal is to inject controlled false sensor measurements x^a to deviate the RV from its trajectory while maintaining the δ below τ to avoid detection. *PID-Piper* limits the impact of stealthy attacks due to 2 reasons. First, the ML model used by *PID-Piper* predicts the RV's behaviour with high accuracy (Section VI), and hence *PID-Piper* can employ a lower detection threshold than prior work [14]. Second, because we keep track of the historic δ using CUSUM, and compare it with τ, the attacker cannot persistently inject the false data, as it will result in higher fluctuation, and hence be detected.

Algorithm 1 Algorithm for Recovery

```
1: u ← target state
2: x ← current state based on sensor measurements
3: y ← actuator signal to calculate motor thrust
4: procedure RecoveryMonitor
5:     u_t ← autonomousLogic()
6:     x_t ← AHRS()
7:     y_PID ← AttitudeControl()
8:     y_ML ← PID_Piper(u_t, x_t)                    ▷ ML model output
9:     S(t) = 0
10:    b(t) > 0
11:    while !mission_end do
12:        δ ← |y_ML − y_PID|
13:        S(t+1) = S(t) + δ − b(t)                  ▷ monitoring using CUSUM
14:        recovery_mode ← false
15:        if S(t+1) > τ then
16:            recovery_mode ← true                  ▷ recovery mode activated
17:            S = 0
18:        end if
19:        if recovery_mode then
20:            y ← y_ML                              ▷ switching control to ML
21:            error ← |y_ML − y_PID| − b(t)
22:            if error → 0 then                     ▷ recovery mode deactivated
23:                recovery_mode ← false
24:                y ← y_PID
25:            end if
26:        else
27:            y ← y_PID
28:        end if
29:        thrustHeading(y)
30:    end while
31: end procedure
```

VI. Results and Evaluation

We first present the experimental setup for evaluating *PID-Piper* (Section VI-A). Then, we evaluate the accuracy of *PID-Piper* in emulating the PID controller, and its false-positive rate. We also measure its effectiveness in handling both overt and stealthy attacks, and finally, its performance overhead.

(a) Pixhawk Drone (b) Aion R1 Rover (c) Sky-viper Drone

Fig. 5: Real RV Systems used for Experiments.

A. Experimental Setup

To evaluate *PID-Piper*, we use 6 RV systems, 3 of which are real RVs. They are (1) Pixhawk based drone [28] (Pixhawk drone), (2) an Aion R1 ground rover [29] (Aion rover), and (3) Sky Viper Journey drone [30] (Sky-viper drone). These are

all commodity RV systems, each costing less than $2000. The other 3 systems are simulated RVs, (4) Ardupilot's quadcopter (ArduCopter), (5) Ardupilot's ground rover [19] (ArduRover), (6) PX4 software in the loop (PX4) [31]. We use the APM SITL [32], and Gazebo [33] platforms for vehicle simulations.

RV Hardware The Pixhawk drone and Aion rover are based on the Pixhawk platform [28]. Pixhawk is an ARM Cortex based platform, consisting of a flight management unit (FMU) controller, I/O, sensors and memory. The Sky-viper drone is based on an STM32 processor, and uses an IMU with 3-axis accelerometer, gyroscope, and barometer.

TABLE I: Mission Profiles, and the empirically derived thresholds (roll, pitch, yaw) for each subject RVs. SL: Straight Line, MW: Multiple Waypoints, CP: Circular Paths, HE: Hover at fixed Elevation, PP: Polygonal Path. '-' means no threshold

RV Systems	Number of Missions					Thresholds
	SL	MW	CP	HE	PP	
ArduCopter	7	10	3	3	7	18, 18.09, 18.6
PX4 Solo	7	10	3	3	7	18.4, 18.4, 18.9
ArduRover	8	12		-	10	– , – , 20
Pixhawk	8	8	3	2	9	18.5, 18.62, 19.1
Sky-viper	8	8	3	2	9	23, 23.6, 24.05
Aion R1	15	5			10	– , – , 21.25

Dataset As there is no standard dataset to test models for RVs, we use a diverse set of missions for each RV with varying mission durations, mission distances, environmental conditions (*e.g.*, noisy sensor data), and covering a variety of mission paths *i.e.*, straight line, circular paths, flights in polygonal paths, and flights with multiple destinations (or waypoints). These emulate real-world RV missions: (1) a last mile delivery drone [34] (straight line path), (2) those used for surveillance or agriculture [35] (circular or polygonal path), and (3) rovers deployed in warehouse management [5] (polygonal paths). Because RVs transition through a known set of operational modes (e.g., takeoff, waypoint/loiter/circle/RTL, and land [19]), the LSTM model has to learn a relatively small set of patterns in input vectors (Section IV-C), and keep track of dependencies between current sensor inputs and past sensor inputs to cover the typical RV missions.

We collect a total of 30 mission profiles for each subject RV, and randomly select 24 (80%) for training, and 6 (20%) for testing. Note that these mission profiles do not include any attacks. We derive the thresholds for attack detection and recovery (Algorithm 1) based on the training set, and validate them using the testing set. Table I summarizes the missions used for the different RVs, and the empirically derived thresholds. We derive only the yaw angle threshold for rovers, because rovers can only control the Z-axis rotation.

Due to physical space restrictions in our environment, our real RV missions were limited to short distances (of 50m). Therefore, we used simulations for long distance RV missions.

Implementation[2] The LSTM model is implemented using Tensorflow 1.10 [36] and Keras 2.2 library. Through offline training, we learn the LSTM model, and implement the *PID-Piper*'s online inferencing module in C++, which is then plugged in to the RV's autopilot software. The autopilot software includes modules for control operations, sensor fusion, and other autonomy related functionalities. We modified fewer than 100 lines of code in each RV's autopilot software, which typically consist of thousands of lines of code. Most of these changes consisted of routing the sensor measurements to *PID-Piper*'s ML model. The ML model runs in tandem with the PID controller for each input/output cycle.

PID-Piper's implementation consists of less than 600 lines, which results in an 1.5% increase in the firmware image size.

Comparison We quantitatively compare our results with SRR [14], which is a recovery framework for RVs based on an FBC design. CI [10] and Savior [13] are also FBC based approaches for detecting attacks against RVs. However, they do not incorporate any recovery. Therefore, for a fair comparison, we extended CI and Savior to provide recovery when an attack is detected, by switching the control to the models used for attack detection (similar to *PID-Piper*'s recovery approach).

Because SRR is not publicly available, we implemented it using Matlab's system identification (SI) tool [38] with our best effort. We derived the system model (containing sets of matrices) using our training data, and we validated the system model's accuracy with our test data. For CI and Savior, we used the publicly available code released by their authors.

Prior work uses different hardware platforms, e.g, Savior used Intel Aero, while SRR and CI used 3DR Iris - both of these are currently discontinued. Because we use different RV hardware, for a fair comparison, we adopt the original simulation platforms used in the respective prior work *i.e.*, ArduCopter for SRR and CI, and PX4 for Savior.

Attacks As we did not have access to special equipment for mounting physical attacks (*e.g.*, noise emitters), we emulated the attacks through targeted software modifications, similar to what previous work has done [10], [13], [14], [18]. We launch three kinds of overt attacks, similar to prior work, namely (i) Attack-1: manipulates gyroscope readings $gyro+f, f$ resulting in > 20 degrees error. (ii) Attack 2: manipulates GPS readings $GPS + f, f$ resulting in $>$ 20m position error, (iii) Attack 3: manipulates gyroscope readings during vulnerable states of the RV (*e.g.*, landing in drones), often resulting in crashes [18].

To mount stealthy attacks, we used the attack code used in prior work [18]. We inject false data based on the selected thresholds for SRR, CI, Savior and *PID-Piper*. To maximize the impact of stealthy attacks, we set the false-data to the maximum value that can escape detection by each technique.

Performance Overheads Similar to SRR [14] and Savior [13], we measure the CPU time incurred by the autopilot modules, with and without *PID-Piper*, at periodic intervals, and report the average overhead. We also measure the overall mission time with and without *PID-Piper*, averaged across 5 missions.

Success Metric We consider a mission to be successful, if after the mission is complete, the total deviation from the original destination is *less than* $10m$. Most GPS sensors used

[2] *PID-Piper* code and dataset used in this work is available at https://github.com/DependableSystemsLab/pid-piper

in commodity RVs have an offset of 5 meters [39], and we consider $2X$ of the GPS offset as our threshold (5m offset from its position, and 5m offset from the destination), as this deviation is indistinguishable from the standard GPS error in RVs. We consider the mission to be unsuccessful, if the RV crashes, stalls, or if the deviation from the destination is greater than 10m. *Crash* means the RV could potentially be physically damaged (*e.g.*, drone falls to the ground), and *stall* means the RV freezes and stops making progress towards the destination.

B. Prediction Accuracy

In these experiments, we evaluate the prediction accuracy of *PID-Piper* with respect to the PID controller. We conducted 5 missions in each of the real RVs. No attacks were considered.

We use Mean Absolute Error (MAE) as the metric to measure accuracy (Section IV). For each mission, we sample the PID and ML outputs at 400 Hz (RV's control logic operates at this frequency), and the MAE is calculated as before.

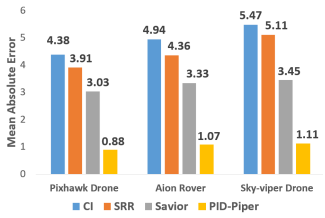


Fig. 6: Comparison of MAE in normal operation.

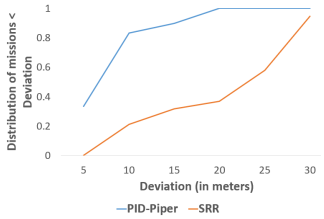


Fig. 7: CDF of deviations from target during recovery.

We also compare *PID-Piper*'s accuracy with CI, Savior and SRR. We record the real RVs' mission data (online) and run the profiled mission data through the models (offline). Recall that all the above techniques used an FBC design, while *PID-Piper* uses an FFC design. Hence, we test their models' effectiveness in predicting the RV's current state. RVs use Extended Kalman Filter (EKF) to translate sensor measurements into the current state (*e.g.*, linear and angular position). Hence, the MAE for the above models is calculated as $MAE = \frac{1}{n}\sum_{t=1}^{n}|y_{EKF} - y_M|$.

Figure 6, shows the average MAE for 5 missions for the 3 real RVs. As can be seen in the figure, *PID-Piper* achieves the lowest MAE among all the techniques, ranging between 0.88 - 1.11 degrees. This is because the LSTM effectively learns temporal sequences in a time series, even in complex missions. Both CI and SRR use a linear model to emulate non-linear RVs, which does not model the RV's behaviour. Though Savior uses a non-linear model, it does not model the RV's transitions (*e.g.*, steady state $\rightarrow$ LAND), and hence its MAE is higher than *PID-Piper*, but lower than CI and SRR.

To test the robustness of *PID-Piper*, we also measured its MAE against noisy sensor inputs due to environmental conditions such as wind. We found that the MAE for ML model's estimations did not change much under environmental disturbances. For example, under variable wind speed (15 km/h to 35 km/h), the MAE ranged from 0.96 to 1.38, which is in the same ballpark as the MAE in normal conditions.

C. False Positives

We define the false positive rate (FPR) as the percentage of times the mission fails in the absence of attacks. For measuring the FPR, we run 30 different missions for each technique without any attacks, and record an FP if a recovery is activated, *and the mission fails as a result.* Even without attacks, it is possible for a gratuitous recovery to get activated when the model estimates deviate from the observed state. However, we record a false positive only if the mission fails as a result.

On average each mission lasts approximately 3 minutes, and so the total time for all missions is 360 minutes $(4 * 30 * 3)$.

TABLE II: Comparison of Gratuitous Recovery in absence of attacks, and FPR across the techniques

Analysis Type	**CI**	**Savior**	**SRR**	***PID-Piper***
Total Missions	30	30	30	30
Recovery activated	7	4	6	3
Mission Successful	0	0	3	3
Mission Failed	7	4	3	0
$FPR = Failed/Total * 100$	23.33%	13.33%	10%	0%

Table II shows the results on the simulated RVs for each technique. We find that *PID-Piper* performs gratuitous recovery in 3 of the 30 missions, but completes the mission successfully in all of them (100% recovery rate), achieving 0% FPR overall. In comparison, CI, Savior and SRR activated gratuitous recovery in 7, 4, and 6 of the missions respectively. Of these cases, CI and Savior recover in none of them (0% recovery rate), while SRR recovers successfully in half of them (50% recovery rate). *Thus, the overall FPR for CI, Savior and SRR was 23.33%, 13.33% and 10% respectively, unlike* PID-Piper*, which had 0% FPR.* We also tested *PID-Piper* on the 3 real RVs and obtained a similar result, namely 0% FPRs.

D. Recovery from Overt Attacks

To test *PID-Piper* 's effectiveness in detecting and recovering from overt attacks, we launched 3 overt attacks targeting the gyroscope and GPS sensors. We then measured the mission success rate under overt attacks. All the techniques, including *PID-Piper*, successfully detected all the attacks, and activated recovery. Therefore, we measure how often the mission is successful *after recovery is activated* by each technique.

As before, we performed the same 30 missions for the simulated RVs with the techniques CI, Savior, and SRR in addition to *PID-Piper*, and mounted the same overt attacks in each mission. Without any recovery, all the missions crashed.

Table III shows the mission outcomes of *PID-Piper* and the prior work after recovery from overt attacks. As can be seen, PID-Piper *successfully recovered the RVs from attacks in 25 out of the 30 missions (mission success rate of* 83%*).*

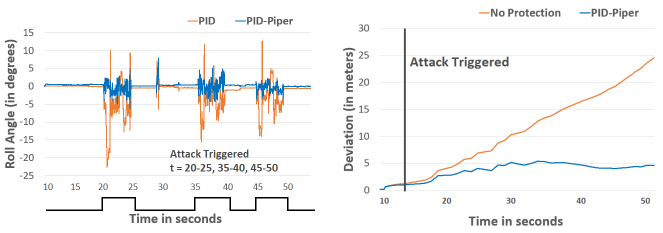
In comparison, *neither CI nor Savior could successfully complete any of the missions (success rate of 0%)*, and *SRR could complete successfully only 4 out of the 30 missions (success rate of 13%).* Moreover, *PID-Piper* did not experience any crashes/stalls (0% crash rate), while SRR experienced crashes/stalls in 11 out of 30 missions (35% crash rate). Finally, CI and Savior incurred crash/stalls in 80% of missions.

TABLE III: Mission Outcomes under Overt Attacks

Analysis Type	CI	Savior	SRR	*PID-Piper*
Total missions	30	30	30	30
Misson Successful	0	0	4	25
Mission Failed (no crash)	4	5	15	5
Crash/Stall	26	25	11	0

We also measured the deviations from the target destination in the non-crash missions for SRR and *PID-Piper*. We exclude CI and Savior as they had very few non-crashing missions. In *PID-Piper*, the average deviation for the 5 failed missions was 14.5m, while for the 25 successful missions, it was 5.7m, for an overall average deviation of 7.35m. In contrast, for SRR, the average deviation was 24m in the 15 failed missions, and 8.2m for the 4 successful missions, for an overall average deviation of 20.67m. *Thus,* PID-Piper *has 2.8X smaller average deviation than SRR for non-crashing missions.*

Figure 7 shows the cumulative distribution function (CDF) of the non-crashing missions for *PID-Piper* and SRR as a function of the deviation incurred. This is normalized to the number of non-crash missions in each technique (30 for *PID-Piper*, 19 for SRR). As can be seen, *PID-Piper*'s CDF increases sharply unlike SRR's CDF, which increases gradually with number of missions. Thus, *PID-Piper* incurs much less deviation than SRR for the non-crashing missions.



(a) (b)

Fig. 8: *PID-Piper*'s recovery under overt attacks. (a) Gyroscope attack in Sky-viper, (b) GPS attack in Pixhawk drone

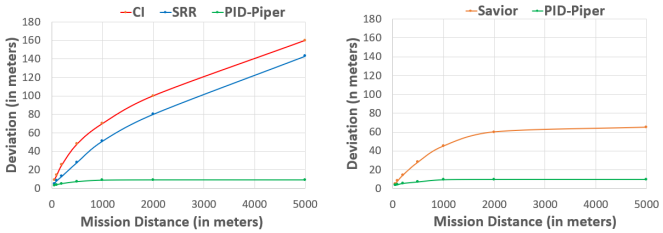
Finally, we tested *PID-Piper*'s recovery on the 3 real RVs. Table IV shows the results. *PID-Piper* performs successful recovery in 87% of the missions on average (videos showing *PID-Piper*'s recovery are available here [40])). To get a deeper understanding, we study its operation on 2 real RVs.

Figure 8a shows an example of *PID-Piper*'s recovery from the gyroscope attack on the Sky-viper drone. The top portion shows the outputs of the PID controller and *PID-Piper* under attacks, and the bottom portion shows the three instances when the attack was launched. As can be seen in the figure, the PID's roll angle estimations experienced heavy fluctuations under the attack (between 12 and -20 degrees), while the ML model's predictions limited the fluctuations between -5 and 5 degrees. *PID-Piper* detected the attack, triggered recovery, and completed the mission successfully. However, without *PID-Piper*, the drone crashed - thus, *PID-Piper* prevented the crash.

Figure 8b shows the deviations due to *PID-Piper* recovery after a GPS attack on the Pixhawk drone. The figure shows the deviations during a 50m mission both with and without *PID-Piper* 's recovery. As can be seen, the deviation without *PID-Piper* is about 25m, while with *PID-Piper*, the deviation is about 5m. Further, the deviations with *PID-Piper* are bounded as the mission distance increases, unlike the deviations without *PID-Piper*. Thus, *PID-Piper* significantly bounds the deviations due to the attack, and makes the mission successful.

E. Mitigating Stealthy Attacks

Recall that stealthy attacks are triggered by controlled sensor manipulations, which over time, result in large deviations in RV missions (Section II-C). We compared *PID-Piper*'s ability to mitigate stealthy attacks with the other systems. We used the ArduCopter for comparison with SRR and CI, and the PX4 for comparison with Savior. We varied the mission distances from 50m to 5000m, for linear paths *i.e.*, straight line missions. Straight line missions represent the best case for the prior techniques as they predominantly use linear models. The resulting deviations are shown in Figures 9a and 9b.



(a) (b)

Fig. 9: Deviation due to stealthy attacks. (a) Comparison between *PID-Piper*, SRR, and CI on ArduCopter. (b) Comparison between *PID-Piper* and Savior on PX4 Solo.

As can be seen, on the Arducopter, PID-Piper *significantly limits the deviations due to stealthy attacks, incurring less than* 10*m deviation even for a* 5000*m mission distance.* In comparison, CI and SRR incur more than 160m and 140m deviation respectively. This is because CI and SRR use a monitoring window to track the error between the observed value and model predicted values. and they set a very high threshold (91 degrees in CI [10] for a 3s window, 22 degrees in SRR [14] for 1s window). This allows stealthy attacks that are launched persistently at each time window to cause a large deviation. In contrast, *PID-Piper* sets the threshold to around 18 degrees, allowing it to limit the effect of stealthy attacks.

On the PX4 Solo system, both *PID-Piper* and Savior cap the deviations, as they use CUSUM [27] to track the error throughout the RV mission (Algorithm 1) rather than a window, which allows them to control the deviations. However, *PID-Piper* caps the deviations due to stealthy attacks at 10m, while Savior caps it at 70m (regardless of the mission distance). This is because Savior again sets a high threshold for comparison, 60 degrees, to avoid false positives. *Thus, the deviation due to stealthy attacks in* PID-Piper *is 7X smaller than that of Savior.*

Overall, the success rate of PID-Piper *is 100%, under stealthy attacks, while for the other 3 techniques, it is 0%.*

We also validated *PID-Piper*'s efficacy by performing stealthy attacks on the 3 real RVs for 50m missions. Table IV shows the results. As can be seen, with *PID-Piper*, the deviations due to stealthy attacks on real RVs were 1 to 3.5m, while without *PID-Piper*, the deviations were 10 to 14m.

TABLE IV: *PID-Piper*'s Overt attack recovery rate, and deviations due to stealthy attacks on 50 m real RV missions.

RV Systems	Overt Attacks Success Rate	Deviation due to Stealthy Attacks(m)	
		No Protection	With PID-Piper
Pixhawk	87.5%	10	1
Sky-viper	88%	13	3.5
Aion R1	86.6%	14	1.23

F. Performance and Power Overhead

As the performance overheads in simulated RVs depend on the computing platform, we use only the three real RVs for measuring the overheads. *PID-Piper* incurs 6.91%, 6.78% and 5.36% performance overhead for the Pixhawk drone, Sky-viper drone and the Aion rover respectively, for an average overhead of 6.35%. In comparison, the average performance overhead for SRR is 6.9% [14]. Therefore, *PID-Piper* incurs a similar performance overhead compared to SRR, while achieving a much higher mission success rate (Table III). We exclude CI and Savior from this comparison due to their poor mission success rates.

We estimate the power consumption based on the runtime overhead of *PID-Piper*. The Pixhawk drone and Aion rover both use an ARM processor and 5000 mAH battery. The processor typically accounts for 12% of the total power consumption [41]. Because *PID-Piper* adds an overhead of 7%, the total power consumption of the RV increases by 0.84%. Finally, we find that *PID-Piper* does *not increase* the RV's overall mission time, both with and without attacks.

VII. Discussion

Missing Attacks As seen from the results, *PID-Piper* achieves over 80% mission success rates under overt attacks, compared to other techniques, which achieve less than 15% success. However, *PID-Piper* does not achieve mission success in about 17% of cases. This is because the attacks cause significant deviation in the RV's trajectory towards the end of the mission, and the ML model is unable to correct the deviation in time. To handle such attacks, *PID-Piper* can be improved using ensemble learning e.g., two LSTM models, one tailored for attacks towards the end of the mission, and use boosting algorithms to correlate the predictions of both networks to obtain the actuator signals of the RV.

Attacks Targeting Other Sensors We only tested *PID-Piper* with attacks on the GPS and gyroscope sensors. However, *PID-Piper* is not limited to these attacks. Attacks on other sensors, such as the accelerometer, magnetometer or optical sensor of the RV, will also result in fluctuations of the PID controller's output. Hence, *PID-Piper* will detect this as an attack and activate the recovery.

Adversarial Attacks It is possible to craft adversarial inputs against *PID-Piper* if the attacker has knowledge of the ML model itself, or has access to the training data (white-box) [42]. However, it is possible to mitigate such attacks by certifying the robustness of each prediction [43], or by utilizing decision boundaries [44] which guarantees that the model does not provide incorrect predictions. On the other hand, attackers can query the system to construct a surrogate model to craft adversarial samples (black-box) [45], [46]. It is difficult to launch such attacks against RVs as they are closed systems. Further, it is possible to defend against such attacks by monitoring the similarity index in queries [47].

VIII. Related Work

Physical invariants have been used to detect attacks against different CPS [48], [49]. These invariants are specific to the CPS it is designed for. For example, BRUIDS, [50] uses domain knowledge to derive rules for detecting different attack types against UAVs. However, this technique is limited to known attack signatures, and does not work for previously unseen attacks. ML techniques have been developed to derive invariants [51]–[55] and physical dynamics based monitoring [56] techniques has been proposed to detect attacks against CPS. None of these have been used for RVs, to our knowledge.

In the domain of RVs, physical attacks targeting the RV's gyroscope [8], accelerometer [9], GPS [7], and optical sensor [57] have been proposed. Only three techniques have been used for attack detection and recovery [10], [13], [14]. We have studied these in detail earlier. CORGIDS [11] finds correlations among system parameters in drones using Hidden Markov Models to detect attacks. However, it does not predict or correct the RV's behaviour, and cannot be used for recovery. In recent work, Fei *et al*. proposed the use of reinforcement learning (RL) in RV controllers to recover from faults and attacks [58]. However, this technique requires the policy to be trained with representative faults and attacks, in order to distinguish them from normal operation. Such representative faults/attacks are difficult to obtain in practice. Further, the RL-based controller cannot control the altitude drop due to gyroscope attacks which will ultimately resulting in a crash. Therefore, unlike *PID-Piper*, this technique will require manual remediation to handle attacks targeting RV's gyroscope.

IX. Conclusion

We presented *PID-Piper*, a framework that uses a Feed-Forward Controller (FFC) for recovering Robotic Vehicles (RVs) from attacks. The FFC in *PID-Piper* uses a Machine Learning (ML) model to emulate the RV's PID controller. The FFC-based ML model prevents over-compensation and high collinearity, making it resilient to attacks.

We evaluate *PID-Piper* on 3 real RVs and 3 simulated RVs. We find that *PID-Piper* (1) achieves 0% false positives in the absence of attacks, (2) recovers RVs from overt attacks and achieves mission success in over 83% of the cases, (3) significantly limits impacts of stealthy attacks to achieve 100% mission success, and (4) incurs low performance overheads.

ACKNOWLEDGEMENT

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and a research gift from Intel. We thank 'Reviewer A' of our ACSAC'20 submission, whose comments helped us rethink the way to position our paper. We also thank the anonymous reviewers of DSN'21 for their helpful comments about this work, and our shepherd, Dr. Mohamed Kaaniche.

REFERENCES

[1] Amazon prime delivery. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011

[2] N. Sheilds. Walmart drone delivery. [Online]. Available: https://www.businessinsider.com/walmart-blockchain-drone-delivery-patent-2018-9

[3] A. Baker. Zipline drone delivery. [Online]. Available: http://www.flyzipline.com/

[4] M. . Mission. Mars exploration rover. [Online]. Available: https://mars.nasa.gov/mer/mission/rover/

[5] C. Steiner. Bot-in-time delivery. [Online]. Available: https://www.forbes.com/forbes/2009/0316/040_bot_time_saves_nine.html

[6] T. E. Humphreys, "Assessing the spoofing threat: Development of a portable gps civilian spoofer," in *In Proceedings of the Institute of Navigation GNSS (ION GNSS*, 2008.

[7] 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*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 75–86. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046719

[8] 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)*. Washington, D.C.: USENIX Association, 2015, pp. 881–896. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/son

[9] 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)*, April 2017, pp. 3–18.

[10] 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: ACM, 2018, pp. 801–816. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243752

[11] E. Aggarwal, M. Karimibiuki, K. Pattabiraman, and A. Ivanov, "Corgids: A correlation-based generic intrusion detection system," in *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, ser. CPS-SPC '18. New York, NY, USA: ACM, 2018, pp. 24–35. [Online]. Available: http://doi.acm.org/10.1145/3264888.3264893

[12] A. Bezemskij, G. Loukas, R. J. Anthony, and D. Gan, "Behaviour-based anomaly detection of cyber-physical attacks on a robotic vehicle," in *2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS)*, Dec 2016, pp. 61–68.

[13] 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)*. Boston, MA: USENIX Association, Aug. 2020.

[14] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. San Sebastian: USENIX Association, Oct. 2020, pp. 349–364.

[15] J. C. Rosser, V. Vignesh, B. A. Terwilliger, and B. C. Parker, "Surgical and medical applications of drones: A comprehensive review," 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/30356360

[16] H. Hildmann and E. Kovacs, "Using unmanned aerial vehicles (uavs) as mobile sensing platforms (msps) for disaster response, civil security and public safety," *Drones*, vol. 3, no. 3, p. 59, 2019.

[17] J. Li and Y. Li, "Dynamic analysis and pid control for a quadrotor," in *2011 IEEE International Conference on Mechatronics and Automation*, Aug 2011, pp. 573–578.

[18] 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*, ser. ACSAC '19. New York, NY, USA: ACM, 2019, pp. 660–672. [Online]. Available: http://doi.acm.org/10.1145/3359789.3359847

[19] Ardupilot - software in the loop. [Online]. Available: http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[20] E. Rusli, T. O. Drews, D. L. Ma, R. C. Alkirc, and R. D. Braatz, "Robust nonlinear feedback-feedforward control of a coupled kinetic monte carlo-finite difference simulation," in *Proceedings of the 2005, American Control Conference, 2005.*, 2005, pp. 2548–2553 vol. 4.

[21] J. Juang and K. Eure, "Predictive feedback and feedforward control for systems with unknown disturbances," *The Journal of the Acoustical Society of America*, vol. 105, no. 2, pp. 971–971, 1999.

[22] A. A. Clements, N. Almakhdhub, K. S. Saab, P. Srivastava, J. Koo, S. Bagchi, and M. Payer, "Protecting bare-metal embedded systems with privilege overlays," in *2017 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2017, pp. 289–303. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2017.37

[23] C. F. Dormann, J. Elith, S. Bacher, C. Buchmann, G. Carl, G. Carré, J. R. G. Marquéz, B. Gruber, B. Lafourcade, P. J. Leitao *et al.*, "Collinearity: a review of methods to deal with it and a simulation study evaluating their performance," *Ecography*, vol. 36, no. 1, pp. 27–46, 2013.

[24] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, Jan 2017, pp. 140–145.

[25] D. L. Poole and A. K. Mackworth, "Artificial intelligence : Foundations of computational agents," Feb 2018. [Online]. Available: https://www.bookdepository.com/Artificial-Intelligence-David-L-Poole/9781107195394

[26] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," *Pattern Recognition*, vol. 44, no. 9, pp. 2231 – 2240, 2011, computer Analysis of Images and Patterns.

[27] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1092–1105. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978388

[28] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2992–2997.

[29] A. Robotics. R1 ardupilot edition. [Online]. Available: https://docs.aionrobotics.com/en/latest/r1-ugv.html

[30] S. Rocket. Sky viper journey. [Online]. Available: https://sky-viper.com/journey/

[31] P. D. Team. Pixhawk autopilot. [Online]. Available: https://docs.px4.io/en/flight_controller/pixhawk_series.html

[32] A. D. Team. Building ardupilot. [Online]. Available: https://github.com/ArduPilot/ardupilot/blob/master/BUILD.md

[33] G. Project. Gazebo robot simulation. [Online]. Available: http://gazebosim.org/

[34] A. J. Hawkins. Ups will use drones to deliver medical supplies in north carolina. [Online]. Available: https://www.theverge.com/2019/3/26/18282291/ups-drone-delivery-hospital-nc-matternet

[35] DJI. Agras t16 - agriculture spraying drone. [Online]. Available: https://www.dji.com/ca/t16

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems,"

2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] MATLAB, "System identification overview." [Online]. Available: https://www.mathworks.com/help/ident/gs/about-system-identification.html

[39] GPS.GOV. Official u.s. government information about the global positioning system (gps) and related topics. [Online]. Available: https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf

[40] PID-Piper. Recovering from attacks against rvs. [Online]. Available: https://bit.ly/3oswuTc

[41] Pixhawk (and apm) power consumption. [Online]. Available: https://diydrones.com/profiles/blogs/pixhawk-and-apm-power-consumption

[42] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *CoRR*, vol. abs/1511.07528, 2015.

[43] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 656–672.

[44] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017.

[45] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016.

[46] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–26.

[47] H. Li, S. Shan, E. Wenger, J. Zhang, H. Zheng, and B. Y. Zhao, "Blacklight: Defending black-box adversarial attacks on deep neural networks," 2020.

[48] S. Adepu and A. Mathur, "Using process invariants to detect cyber attacks on a water treatment system," in *IFIP International Information Security and Privacy Conference*, 2016, pp. 91–104.

[49] T. Paul, J. W. Kimball, M. Zawodniok, T. P. Roth, B. McMillin, and S. Chellappan, "Unified invariants for cyber-physical switched system stability," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, p. 2014, 2014.

[50] R. Mitchell and I.-R. Chen, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, p. 2014, 2014.

[51] 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)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2018, pp. 648–660. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2018.00016

[52] C. M. Ahmed, J. Zhou, and A. P. Mathur, "Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in cps," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. New York, NY, USA: ACM, 2018, pp. 566–581. [Online]. Available: http://doi.acm.org/10.1145/3274694.3274748

[53] J. Inoue, Y. Yamagata, Y. Chen, C. M. Poskitt, and J. Sun, "Anomaly detection for a water treatment system using unsupervised machine learning," *CoRR*, vol. abs/1709.05342, 2017. [Online]. Available: http://arxiv.org/abs/1709.05342

[54] 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*, ser. CPSS '16. New York, NY, USA: ACM, 2016, pp. 34–43. [Online]. Available: http://doi.acm.org/10.1145/2899015.2899016

[55] R. C. Borges Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, "Machine learning for power system disturbance and cyber-attack discrimination," in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*, Aug 2014, pp. 1–8.

[56] P. Guo, H. Kim, N. Virani, J. Xu, M. Zhu, and P. Liu, "Roboads: Anomaly detection against sensor and actuator misbehaviors in mobile robots," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 574–585.

[57] D. Davidson, H. Wu, R. Jellinek, V. Singh, and T. Ristenpart, "Controlling uavs with sensor input spoofing attacks," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/davidson

[58] 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)*, 2020, pp. 7358–7364.