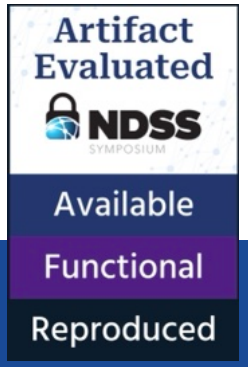
Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026 , San Diego, CA, USA



ADGFUZZ: Assignment Dependency-Guided Fuzzing for Robotic Vehicles

Yuncheng Wang*, **Yaowen Zheng***, **Puzhuo Liu**, **Dongliang Fang**✉, **Jiaxing Cheng**, **Dingyi Shi**, **Limin Sun**

University of Chinese Academy of Sciences
INSTITUTE OF INFORMATION ENGINEERING,CAS
ANT GROUP

*: Both authors contributed equally to this work. ✉: Corresponding Author

Robotic Vehicle (RV) and Control Software

- Robotic Vehicles (RVs) refer to unmanned autonomous or semi-autonomous platforms (e.g., UAVs, UGVs) designed to perform complex physical tasks in diverse environments.
  - Key Roles: commercial aerial photography, formation performances

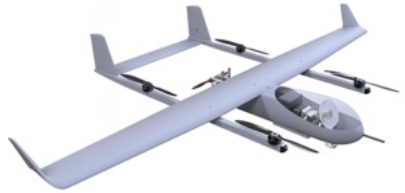
Unmanned Aerial Vehicle (UAV)

Unmanned Ground Vehicle (UGV)

Unmanned Surface Vehicle (USV)

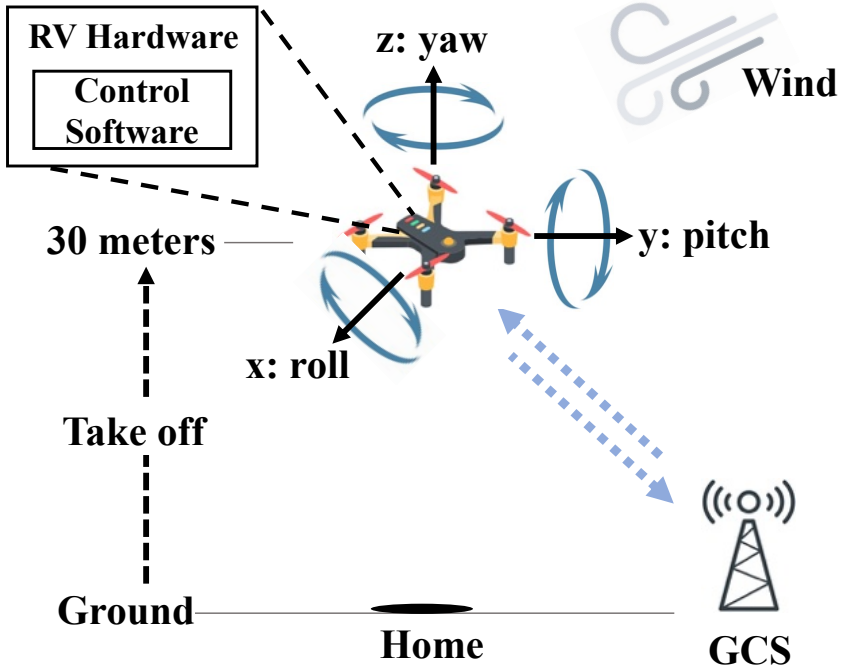
Robotic Vehicle (RV) and Control Software

- Robotic Vehicles (RVs) refer to unmanned autonomous or semi-autonomous platforms (e.g., UAVs, UGVs) designed to perform complex physical tasks in diverse environments.
  - Key Roles: commercial aerial photography, formation performances



- RV Control Software : The system 'brain'
  - Sensor Data Fusion
  - State Estimation & Computing
  - Stability & Attitude Control
  - ... ...



RVs - Security & Safety

- RVs are critical cyber-physical systems with widespread deployment
- Control software bugs can cause crashes, mission failures, or loss of control
- Security Landscape of RVs: The Complexity Challenge
  - Vast Input Space: Thousands of params, 164 commands, and fluctuating environment data.
  - State-Space Explosion: Exponentially increasing combinations make exhaustive testing impossible.
  - Undetected Deep Bugs: Conventional fuzzers fail to reach deep functional logic errors.

RVs - Security & Safety

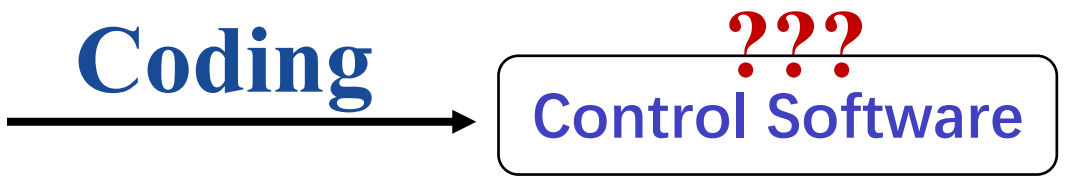
- RVs are critical cyber-physical systems with widespread deployment
- Control software bugs can cause crashes, mission failures, or loss of control
- Security Landscape of RVs: The Complexity Challenge
  - Vast Input Space: Thousands of params, 164 commands, and fluctuating environment data.
  - State-Space Explosion: Exponentially increasing combinations make exhaustive testing impossible.
  - Undetected Deep Bugs: Conventional fuzzers fail to reach deep functional logic errors.

Enabling focused testing by partitioning the massive input space into manageable RV-specific subspaces.

From Control Theory to Programmatic Realization

CPS + Control Theory

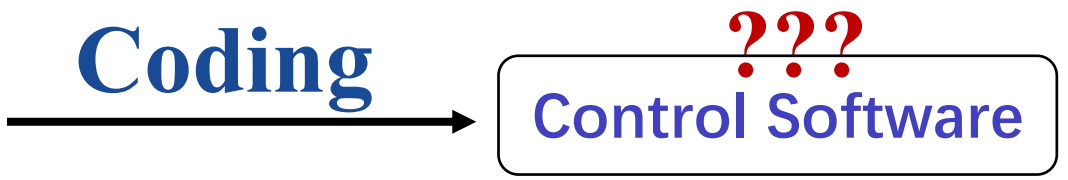
RV control software acts as the bridge between **digital logic** and **physical dynamics**.



From Control Theory to Programmatic Realization

CPS + Control Theory

RV control software acts as the bridge between **digital logic** and **physical dynamics**.



- The Gap in Formula Implementation
  - Complex theoretical formulas (e.g., PID control) cannot be perfectly replicated in code.
  - Requires numerical approximations, discretization, and indirect assignments through auxiliary variables.

PID

$$u(x)=kp(err(t)+\frac{1}{T}\cdot\int err(t)dt+\frac{T_D derr(t)}{dt})$$

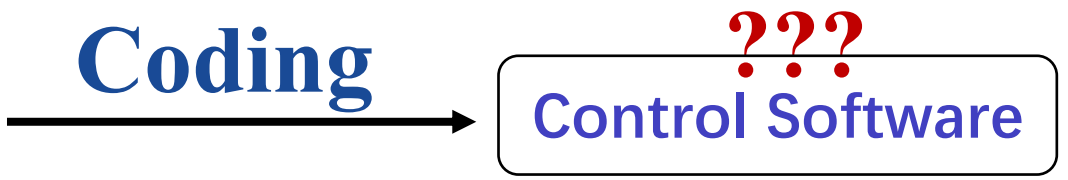
$$U(k) = K_p\,(err(k) + \frac{T}{T_I}\sum err(k) + \frac{T_D}{T}(err(k) - err(k-1)))$$

```
error = setpoint - current_value;
integral += error * dt;
derivative = (error - previous_error) / dt;
output = Kp * error + Ki * integral + Kd * derivative;
```

From Control Theory to Programmatic Realization

CPS + Control Theory

RV control software acts as the bridge between **digital logic** and **physical dynamics**.



- The Gap in Formula Implementation
  - Complex theoretical formulas (e.g., PID control) cannot be perfectly replicated in code.
  - Requires numerical approximations, discretization, and indirect assignments through auxiliary variables.

PID

$$u(x)=kp(err(t)+\frac{1}{T}\cdot\int err(t)dt+\frac{T_D derr(t)}{dt})$$

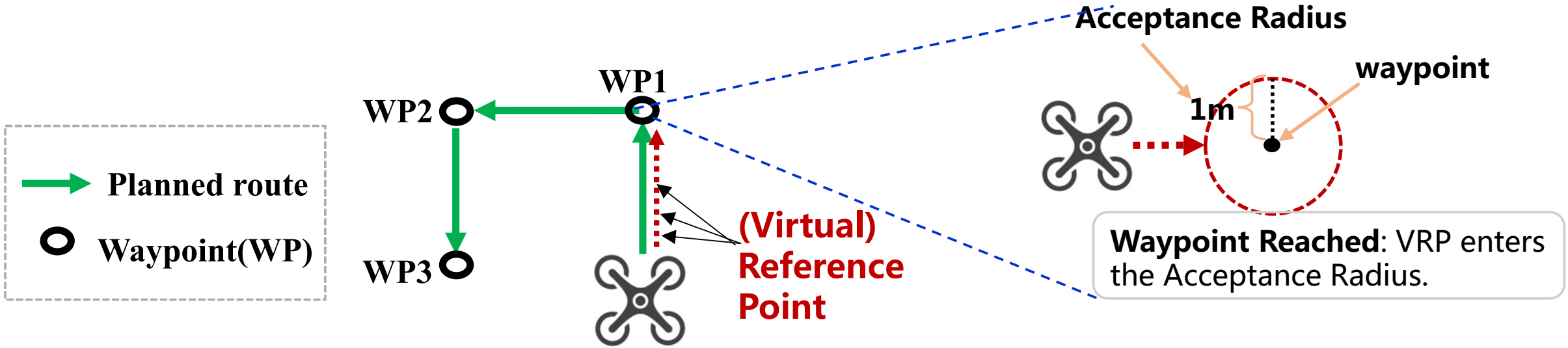
$$U(k) = K_p\,(err(k) + \frac{T}{T_I}\sum err(k) + \frac{T_D}{T}(err(k) - err(k-1)))$$

```
error = setpoint - current_value;
integral += error * dt;
derivative = (error - previous_error) / dt;
output = Kp * error + Ki * integral + Kd * derivative;
```

- Vulnerability
  - Human Errors: Memory corruption, missing validation logic.
  - Logic Flaws: Unintended interactions between conflicting inputs.
  - Translation Gap: Incomplete mapping from physical models to code.

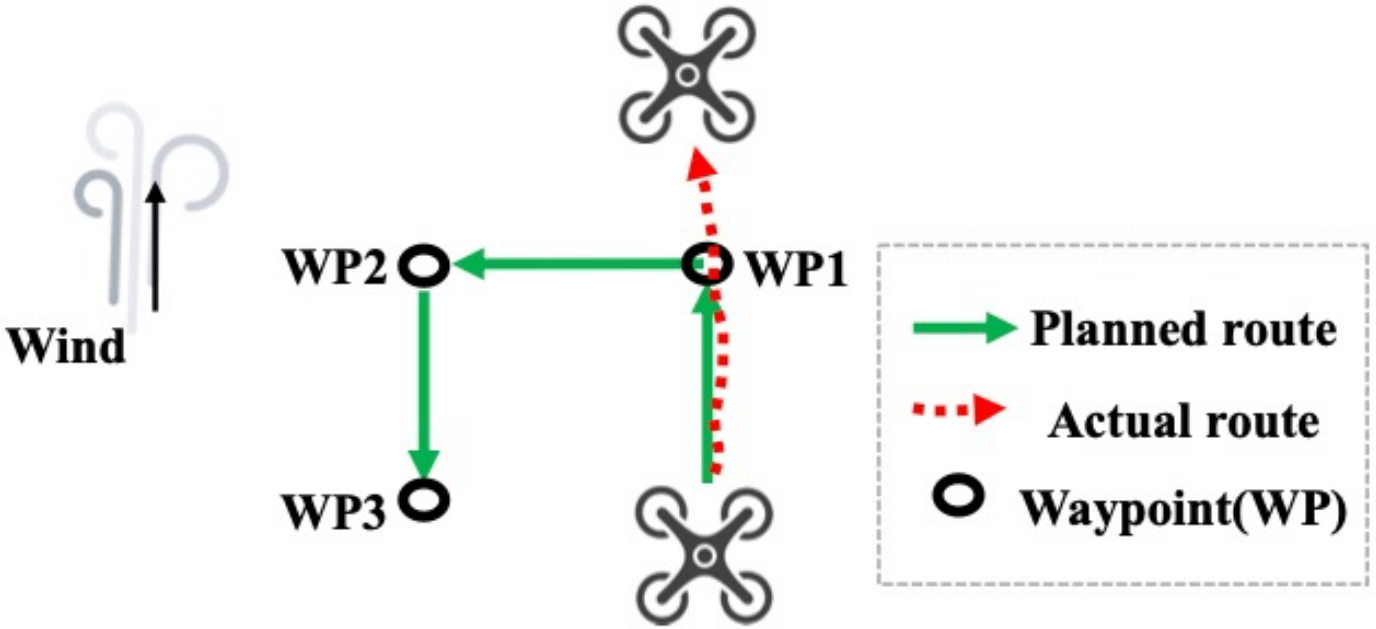
Motivating Example

- Waypoint Navigation Task: Autonomous waypoint-to-waypoint flight via a predefined mission path.
- Path Following via VRP
  - The vehicle tracks the path by "chasing" a Virtual Reference Point (VRP).
  - e.g. ArduPilot: L1 Controller (L1 Point)
- VRP represent the sequence of desired positions along the flight path.



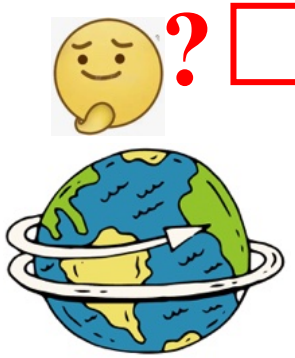
Motivating Example

- Wind-Induced Trajectory Deviation
  - Planned Mission (**Green Line**): A precise rectangular mission passing through three waypoints (WP1, WP2 and WP3).
  - Actual Trajectory (**Red Line**): the copter drifts away from the waypoints

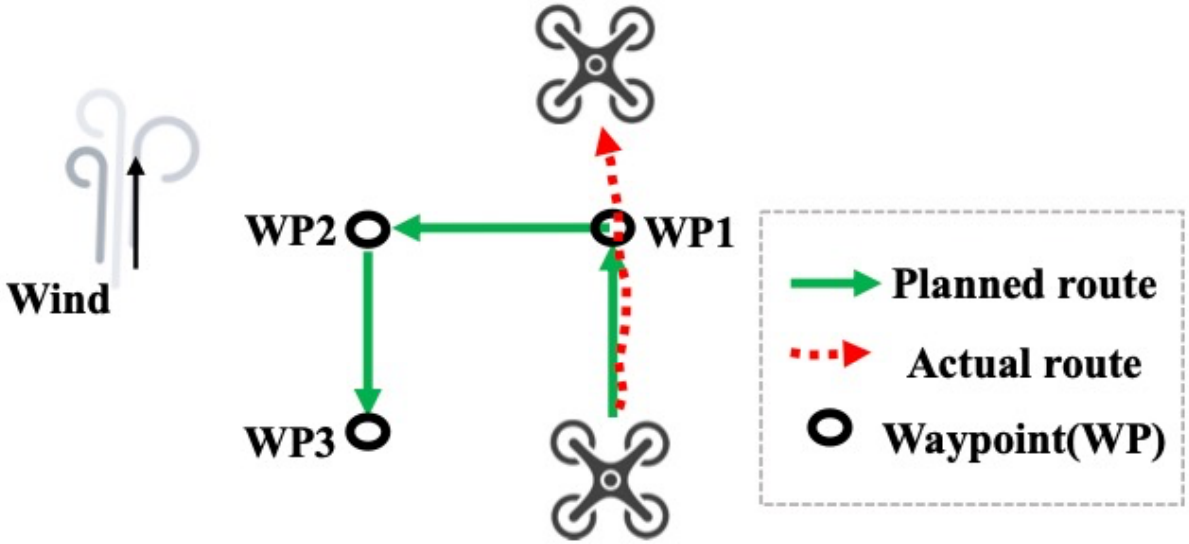


Motivating Example

- False Waypoint Reaching
  - Copter keeps drifting away....
  - But: waypoint reached???



Log:
[omitted]...
Flying to WP2...
Reached WP2!
Flying to WP3...



Motivating Example – Root Cause

- The trajectory tracking logic does not validate the Copter's actual position.
- The reference-point advancing speed is lower-bounded (≥ 0.1), preventing it from stopping.

```
1  bool AC_WPNav::advance_wp_target_along_track(float dt) {
2      float track_error = _pos_control_error.dot(track_direction);
3      float track_velocity = _inav_velocity_neu.dot(track_direction);
4  -   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.1f, 1.0f);
4  +   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.0f, 1.0f);
5      float track_scaler_tc = 0.01f * get_wp_acceleration()/_wp_jerk;
6      float _track_scalar_dt += (track_scaler_dt - _track_scalar_dt) * (dt / track_scaler_tc);
7      bool s_finished =  _s_leg.advance_target_along_track(_wp_radius_cm, get_corner_accel(),
                          _flags.fast_waypoint, _track_scalar_dt * vel_scaler_dt * dt,
                          target_pos, target_vel, target_accel);
8      if (!_flags.reached_destination) { // check whether reached the waypoint
9          if (s_finished) {
10             _flags.reached_destination = true;
11   //...(omitted). This means the change in angle is equivalent to the change in acceleration.
```

Assignment Statement Bugs

- Error Propagation in Control Logic
  - line 4 (track_scaler_dt) -> line 6 -> line 7 (s_finished) -> ...

```
1  bool AC_WPNav::advance_wp_target_along_track(float dt) {
2      float track_error = _pos_control_error.dot(track_direction);
3      float track_velocity = _inav_velocity_neu.dot(track_direction);
4  -   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.1f, 1.0f);
4  +   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.0f, 1.0f);
5      float track_scaler_tc = 0.01f * get_wp_acceleration()/_wp_jerk;
6      float _track_scalar_dt += (track_scaler_dt - _track_scalar_dt) * (dt / track_scaler_tc);
7      bool s_finished =  _s_leg.advance_target_along_track(_wp_radius_cm, get_corner_accel(),
                          _flags.fast_waypoint, _track_scalar_dt * vel_scaler_dt * dt,
                          target_pos, target_vel, target_accel);
8      if (!_flags.reached_destination) { // check whether reached the waypoint
9          if (s_finished) {
10             _flags.reached_destination = true;
11     //...(omitted). This means the change in angle is equivalent to the change in acceleration.
```

Assignment Statement Bugs

- Error Propagation in Control Logic
  - line 4 (track_scaler_dt) -> line 6 -> line 7 (s_finished) -> ...

```
1  bool AC_WPNav::advance_wp_target_along_track(float dt) {
2      float track_error = _pos_control_error.dot(track_direction);
3      float track_velocity = _inav_velocity_neu.dot(track_direction);
4  -   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.1f, 1.0f);
4  +   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                                track_error) / curr_target_vel.length(), 0.0f, 1.0f);
5      float track_scaler_tc = 0.01f * get_wp_acceleration()/_wp_jerk;
6      float _track_scalar_dt += (track_scaler_dt - _track_scalar_dt) * (dt / track_scaler_tc);
7      bool s_finished =  _s_leg.advance_target_along_track(_wp_radius_cm, get_corner_accel(),
                          _flags.fast_waypoint, _track_scalar_dt * vel_scaler_dt * dt,
                          target_pos, target_vel, target_accel);
8      if (!_flags.reached_destination) { // check whether reached the waypoint
9          if (s_finished) {
10             _flags.reached_destination = true;
11     //...(omitted). This means the change in angle is equivalent to the change in acceleration.
```

How common are assignment bugs in real RV software?

Exploratory Study

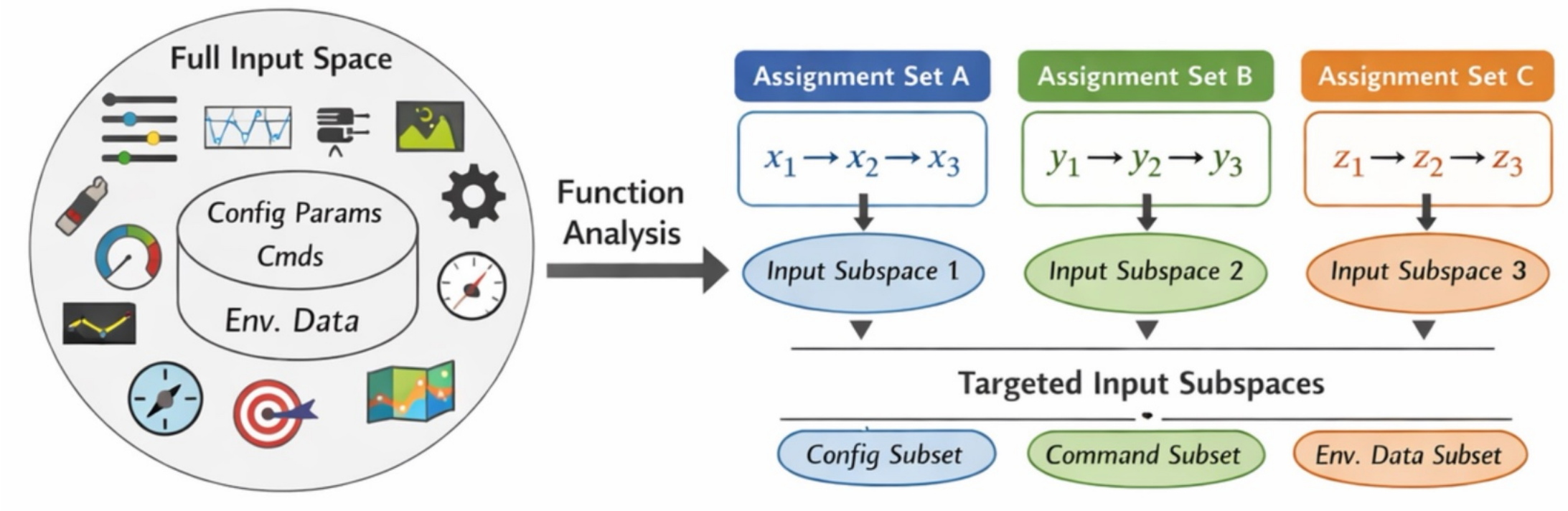
- ArduPilot issues (https://github.com/ArduPilot/ardupilot/issues?q=is:issue label:BUG)
  - From Jan. 2015 to Jan. 2025
  - 819 issues: label = 'BUG'

Table: Bug fix patterns in 'BUG'-labeled ArduPilot issues and their categorization.

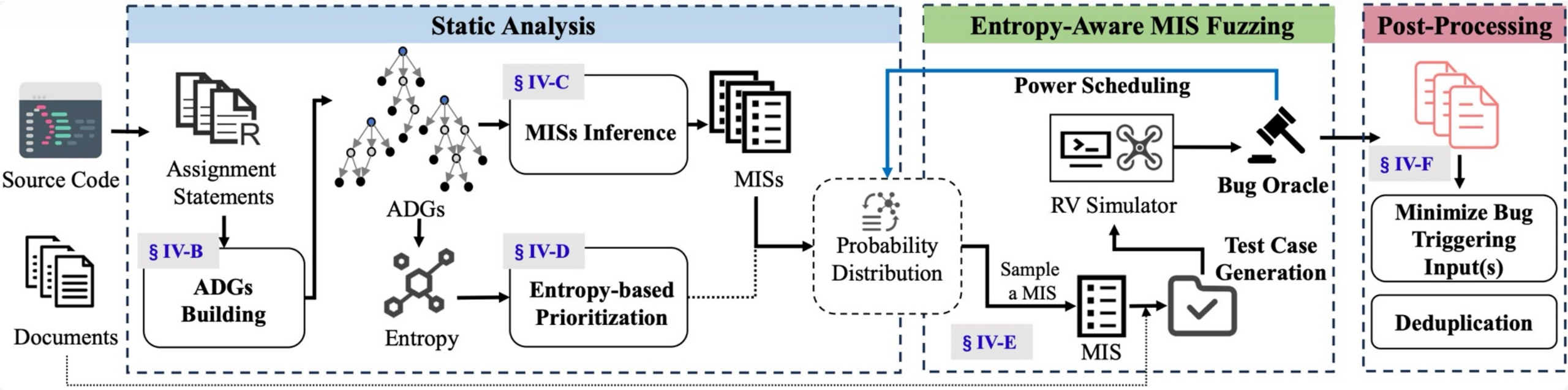
Fixing code types	# of issues	Proportion (%)
Assignment statement corrections	58	28.02
Logic additions or deletions	111	53.62
Function call modifications	16	7.74
Function return type or numerical unit changes	11	5.31
Default value adjustments	11	5.31

Main Idea: Focused input space exploration

- Parse each function into independent assignment sets.
- Leverage variable semantics to link to input space.
- Extract multiple dependent assignment sets.



Approach Overview - ADGFuzz



- Three main stages:
  - Static Analysis : (1) Extract dependency-aware assignment sets from source code and (2) map code variables to inputs (MIS).
  - Entropy-Aware MIS Fuzzing : Prioritize and fuzz MISs based on entropy.
  - Post-Processing : Analyze, minimize, and deduplicate fuzzing results to produce unique bug reports.

Definition

- Assignment Statement (AS)
  - Each AS is modeled as a **triple** $S = (y, X, O)$

$$y := x_1 \oplus_1 x_2 \oplus_2 \ldots \oplus_{n-1} x_n$$

$$\oplus_i \in \underbrace{\{+, -, \times, \div\}}_{arithmetic} \cup \underbrace{\{\wedge, \vee, \neg, =\}}_{logical}$$

- Assignment Dependency Graph (ADG)
  - A **Directed Acyclic Graph** $G = (V, E)$, constructed from interdependent AS within a specific function.
    - $V = V_{root} \cup V_{leaf} \cup V_{semi}$ *is a set of nodes where:*

$$V_{root} = \{v \mid \exists \mathbb{S}_1 : v \in \mathbb{S}_1.y \wedge \nexists \mathbb{S}_2 : v \in \mathbb{S}_2.X\}$$
$$V_{leaf} = \{v \mid \exists \mathbb{S}_1 : v \in \mathbb{S}_1.X \wedge \nexists \mathbb{S}_2 : v \in \mathbb{S}_2.y\}$$
$$V_{semi} = \{v \mid \exists \mathbb{S}_1 \neq \mathbb{S}_2 : v \in \mathbb{S}_1.y \wedge v \in \mathbb{S}_2.X\}$$

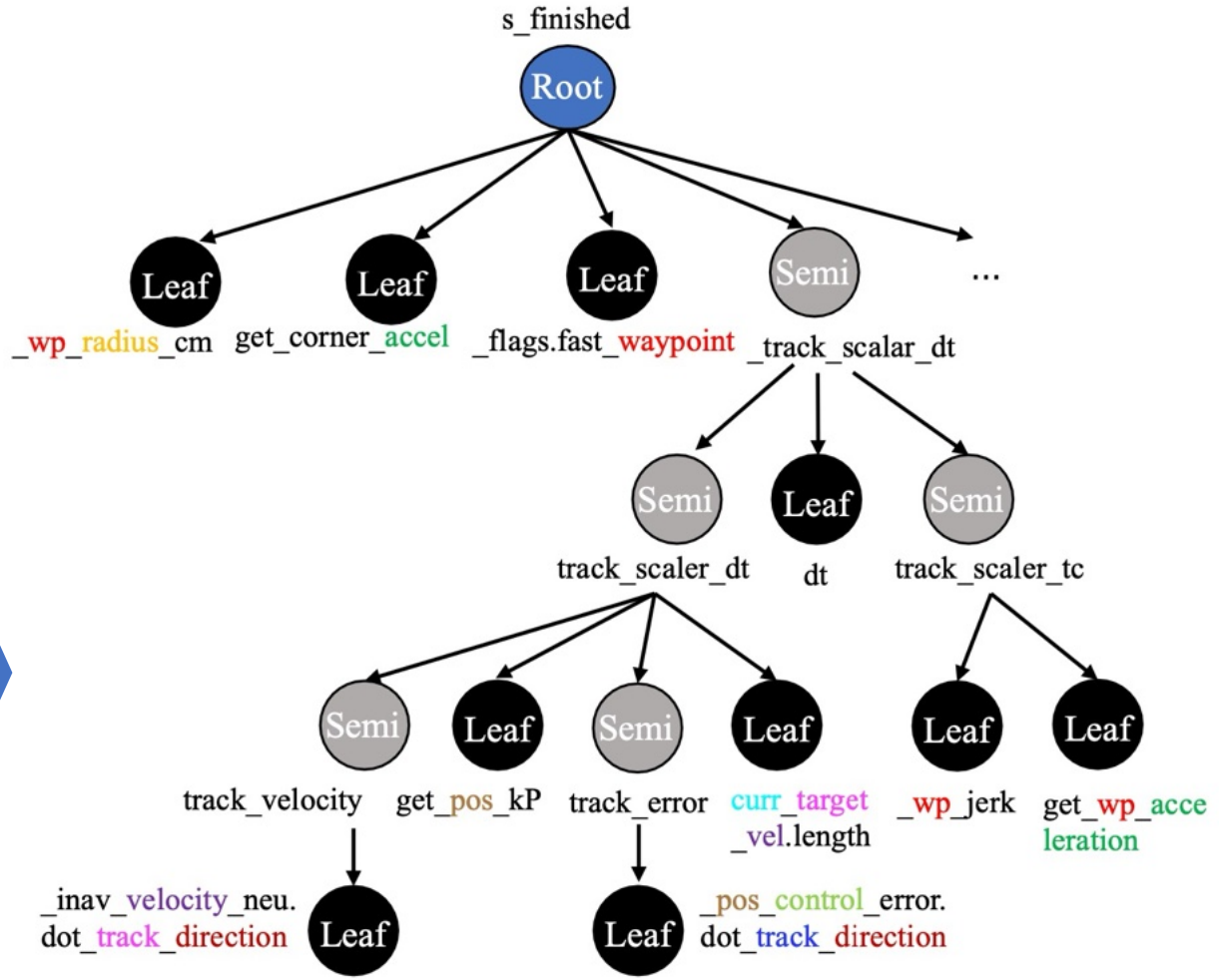
    - $E = E_{RL} \cup E_{RS} \cup E_{SL} \cup E_{SS}$

① ADGs Building

- Construct the ADG by performing syntax analysis on source code, mapping functional semantics to specific variable names.

```
1 bool AC_WPNav::advance_wp_target_along_track(float dt) {
2     float track_error = _pos_control_error.dot(track_direction);
3     float track_velocity = _inav_velocity_neu.dot(track_direction);
4 -   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                    track_error) / curr_target_vel.length(), 0.1f, 1.0f)
4 +   float track_scaler_dt = constrain_float(0.05f + (track_velocity - get_pos_kP() *
                                    track_error) / curr_target_vel.length(), 0.0f, 1.0f)
5     float track_scaler_tc = 0.01f * get_wp_acceleration()/_wp_jerk;
6     float _track_scalar_dt += (track_scaler_dt - _track_scalar_dt) * (dt / track_scaler_tc);
7     bool s_finished =  _s_leg.advance_target_along_track(_wp_radius_cm, get_corner_accel(),
                        _flags.fast_waypoint, _track_scalar_dt * vel_scaler_dt * dt,
                        target_pos, target_vel, target_accel);
8     if (!_flags.reached_destination) { // check whether reached the waypoint
9         if (s_finished) {
10            _flags.reached_destination = true;
11  //...(omitted). This means the change in angle is equivalent to the change in acceleratic
```

- 1-Node: Variable Extraction & Attribution
  - grammar filtering & attribute assignment
- 2-Edge: Dependency-Based Graph Linkage
  - root → semi, root → leaf
  - semi → semi, semi → leaf



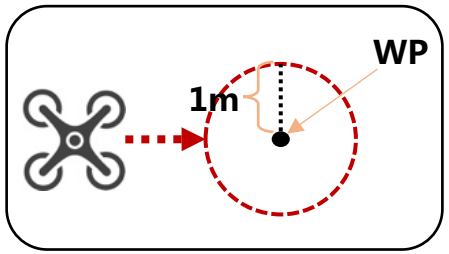
② MISs Inference

- Map ADG variables to multi-source inputs to construct **Matched Input Sets (MIS).**

```
bool s_finished =
_s_leg.advance_target_along_track(_wp_radius_cm,
get_corner_accel(), _flags.fast_waypoint,
_track_scalar_dt * vel_scaler_dt * dt,
target_pos, target_vel, target_accel);
```

_wp_radius_cm

(1) **Variable:** Represents the tolerance radius of the waypoint



Tolerance/Acceptance Radius

WPNAV_RADIUS: Waypoint Radius

Defines the distance from a waypoint, that when crossed indicates the wp has been hit.

Increment	Range	Units
1	5 to 1000	centimeters

Input param WPNAV_RADIUS : Used to set the tolerance radius for waypoints

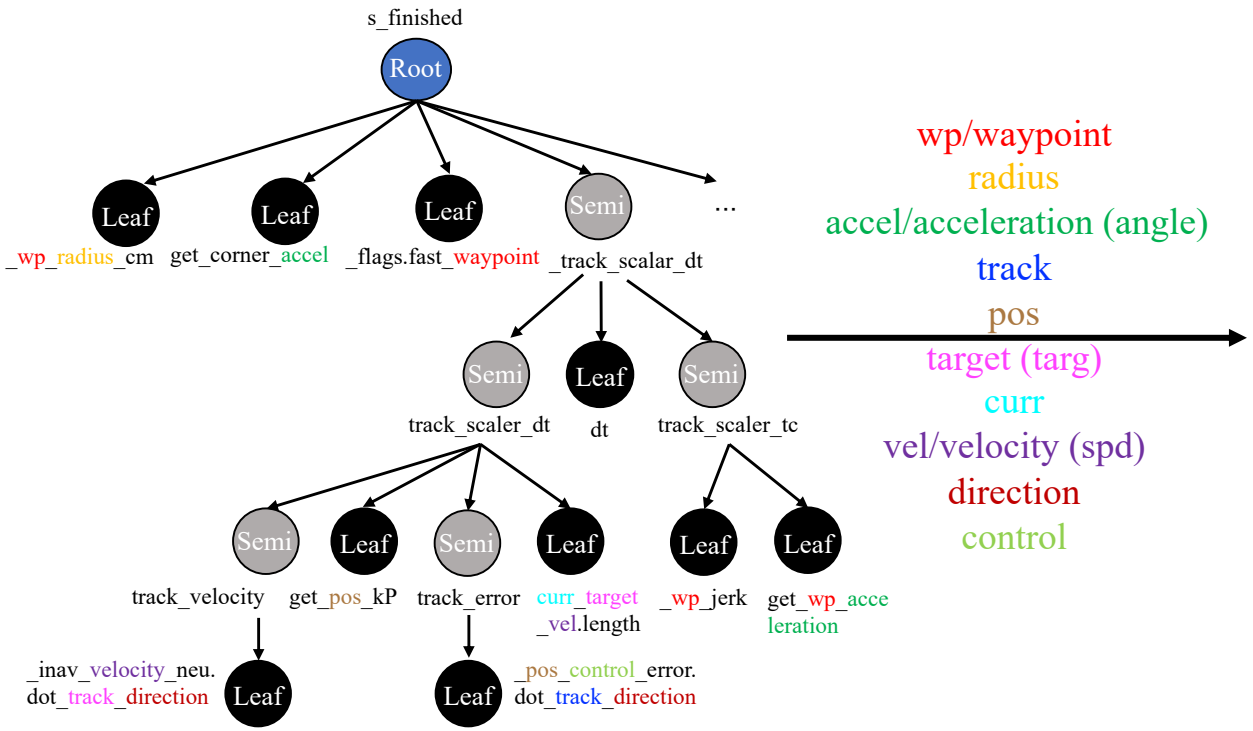
(2)

```
8     if (!_flags.reached_destination) { // check whether reached the waypoint
9         if (s_finished) {
10            _flags.reached_destination = true;
11    //...(omitted). This means the change in angle is equivalent to the change in acceleration.
```

(3) ...

② MISs Inference

- Leveraging **Terminology Similarity** to bridge the gap between **Program Variables** (internal) and **Multi-source Inputs** (external).



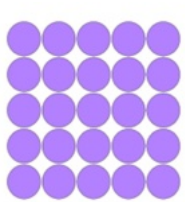
wp/waypoint
radius
accel/acceleration (angle)
track
pos
target (targ)
curr
vel/velocity (spd)
direction
control

WP_YAW_BEHAVIOR, WP_NAVALT_MIN, MAV_CMD_NAV_WAYPOINT, ...
ADSB_LIST_RADIUS, CIRCLE_RADIUS, FENCE_RADIUS, WPNAV_RADIUS, ...
AVOID_ACCEL_MAX, ATC_ACCEL_{R/P/Y}_MAX, INS_ACCEL_FILTER, WPNAV_ACCEL, ANGLE_MAX, AVOID_ANGLE_MAX, ATC_ANGLE_BOOST, ...
MAV_CMD_CAMERA_TRACK_POINT, MAV_CMD_CAMERA_TRACK_RECTANGLE
FLOW_POS_X, RNGFND{*}_POS_{X/Y/Z}, SIM_SONAR_POS_X, ...
AROT_FWD_SP_TARG, SIM_SB_ALT_TARG
MOT_BAT_CURR_TC, MOT_BAT_CURR_MAX, BATT_CURR_PIN, ...
EK2_VEL_I_GATE, EK3_VEL_I_GATE, AVOID_BACKUP_SPD, SIM_WIND_SPD, ...
ZIGZ_DIRECTION
MAV_CMD_DO_CONTROL_VIDEO, MAV_CMD_DO_MOUNT_CONTROL, ...

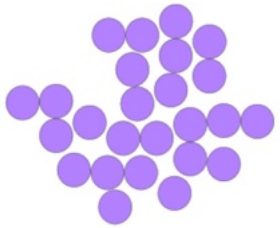
(a) ADG **(b) MIS**

③ Entropy-based Prioritization

- Randomly testing massive MIS is inefficient.
- Shannon Entropy [1]: High Uncertainty → High Unpredictability. ? → Higher Bug Potential



Low entropy
Highly ordered



Entropy increase
The degree of disorder increases

- **Metric: Semantic Information Richness Entropy** measures each MIS.

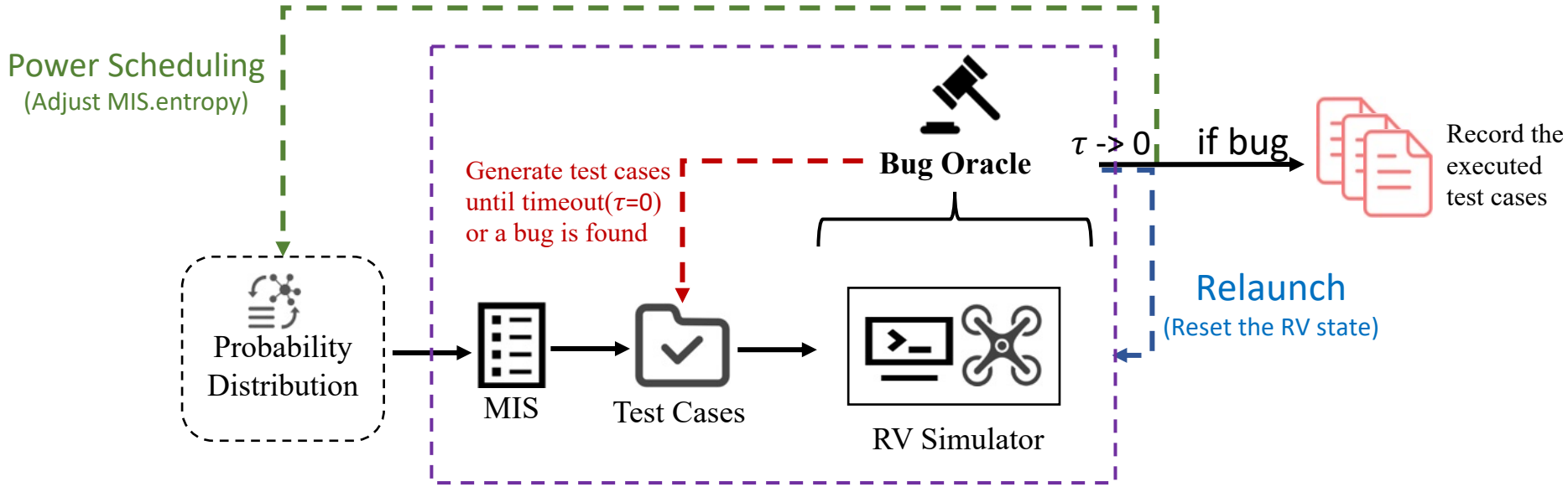
$$\underbrace{E(\mathrm{MIS}) = \log_2(|N|+1) + \log_2\left(1 + \sum_{v \in V_{\mathrm{leaf}}} \sum_{i=1}^{k} \frac{i}{T_i}\right)}$$

$$p(\mathrm{MIS}) = \frac{E(\mathrm{MIS})}{\sum_{L \in \mathrm{MISs}} E(L)}$$

[1] Shannon CE. A mathematical theory of communication[J/OL]. SIGMOBILEMob.Comput. Commun. Rev., 2001, 5(1): 3–55.

④ Entropy-Aware MIS Fuzzing

- Parallel Fuzzing Workflow
  - **Simulation Module**: Spawns simulator instances and automates multi-stage navigation tasks.
  - **Execution Module**: Samples MIS via entropy distribution and injects test cases during flight states.
  - **Oracle Module**: Monitors anomalies in real-time, updates entropy weights, and triggers post-processing analysis.



Evaluation Setup

- Evaluated on the three most widely used RV types: Copter, Plane, and Rover.

- SITL (Software-in-the-Loop) simulation
- MAVProxy: Ground Control Station
- MAVLink: Communication protocol
- Test Duration: 24-hour
- Cross-verified on real-world RV

Control Software: ArduPilot and PX4

(a) Copter (b) Plane (c) Rover

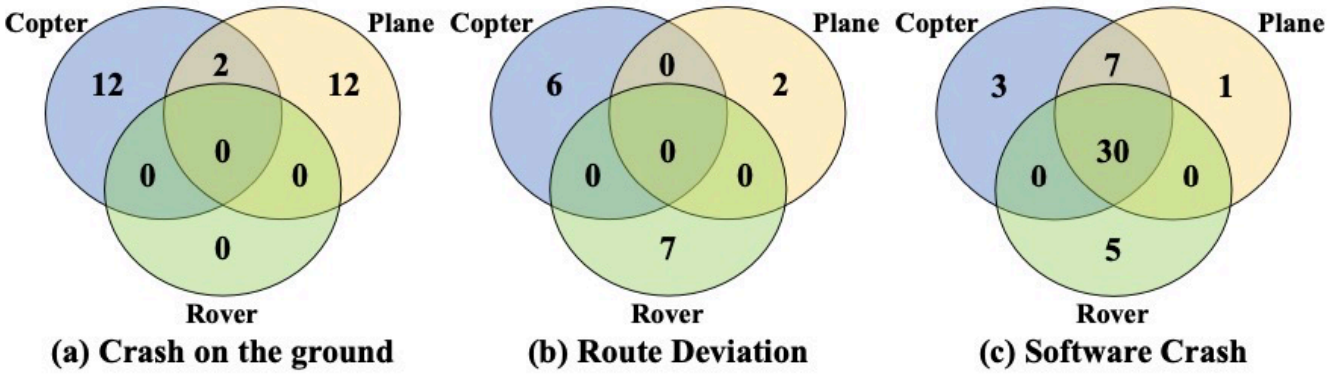
MAVLink

GCS
(MAVProxy)

Bugs Found

- **ArduPilot: 87 Unique Bugs** identified.

RV Type	# of bugs	Bug Invariants		
		Crash on the ground	Route deviation	Software crash
Copter	60	14	6	40
Plane	54	14	2	38
Rover	42	0	7	35
Total	87*	26*	15	46*



(a) Crash on the ground (b) Route Deviation (c) Software Crash

- 35 bugs were found in PX4 (not shown in the figure)

Index	RV Type	Inputs	Description	Impact Type	Reproducibility	Status
1	C	AHRS_EKF_TYPE	Assigning special values (e.g., 0 or 11) to this parameter results in the copter crashing to the ground.	Safety	Yes	Pending
2	C	SERVO3_FUNCTION	Setting this parameter to special values (e.g., 141) leads to thrust loss and the copter crashes.	Safety	Yes	Pending
3	C	MOT_BAT_CURR_MAX, MOT_PWM_MIN	Setting either parameter individually does not induce issues; however, concurrently assigning both parameters to small values results in complete loss of thrust and a crash.	Safety	Yes	Yes
4	C	FLTMODE_CH	Assigning a special value (e.g., 8) to this parameter leads to thrust loss and a subsequent crash.	Safety	Yes	Pending
5	C	AHRS_EKF_TYPE, AHRS_TRIM_X	Assigning these two parameters to 2 and the minimum legal value (-0.1745), respectively, causes the copter to crash to the ground.	Safety	Yes	Pending
6	C	SIM_GPS1_LAG_MS	Setting to a large value leads to unstable flight attitude,leading to a crash after a short period.	Usability	No	No
7	C	SIM_TIME_JITTER,MAV_CMD_DO_SET_MODE	Causes the copter to crash to the ground.	Safety	No	No
8	C	MOT_BAT_VOLT_MAX, MOT_PWM_MIN	Setting either parameter independently does not cause adverse effects; however, assigning both parameters simultaneously results in thrust loss and a crash.	Safety	Yes	Yes
9	C	FS_THR_VALUE	Setting to the maximum legal value (1100) prevents the safety mechanism from triggering, causes the copter to crash.	Safety	Yes	Pending
10	C	SIM_RATE_HZ	Assigning values within a specific range (e.g., 85) causes the copter to flip unstably and fall to the ground.	Usability	No	No
11	C	MAV_CMD_NAV_LOITER_UNLIM	Normal use of this command unexpectedly changes the copter's flight mode, causing it to hit the ground.	Safety	Yes	Yes
12	C	Multiple MOT_SPIN_MIN	During waypoint navigation, setting MOT_SPIN_MIN to 0.6 causes the copter to climb after reaching the waypoint. Restoring it to the nominal 0.15 corrects the altitude. Setting it to 0.6 again results in a brief climb followed by a stall and crash.	Safety	Yes	Yes
13	P	SIM_SERVO_SPEED	Inadequate validation of parameter ranges allows a value of 100, which results in the plane crashing.	Usability	No	No
14	P	TECS_PITCH_MIN, PTCH_RATE_D_FF	When the mission waypoint altitude is set to 30 meters, assigning large values to these parameters causes the plane to first climb to approximately 50 meters, then rapidly descend and crash.	Safety	Yes	Pending
15	P	RCMAP_PITCH	Assigning special values (e.g., 10 or 16) causes the plane to gradually lose altitude and ultimately crash.	Safety	Yes	Pending
16	P	AIRSPEED_MIN	If the plane cannot reach the target speed under current parameter settings, it lowers its pitch and altitude to accelerate, which may lead to gradual descent and eventual crash.	Safety	Yes	Pending
17	P	ARSPD_OFFSET	No proper range checking; large values cause the plane to crash to the ground.	Safety	Yes	Pending
18	P	TECS_SPD_OMEGA	Large values cause linear speed reduction and eventual crash.	Safety	Yes	Pending
19	P	SIM_SERVO_DELAY	Small values cause the plane to flip in flight and rapidly crash.	Usability	No	No
20	P	PTCH2SRV_RMAX_UP	Assigning small values causes the plane to crash.	Safety	Yes	Pending
21	P	ICE_ENABLE	Out-of-range values incorrectly cause the plane to crash to the ground.	Safety	Yes	Pending
22	P	TECS_TIME_CONST	Assigning large values leads to deceleration and eventual crash.	Safety	Yes	Pending
23	P	PTCH2SRV_RLL	Assigning large values causes the plane to overshoot the prescribed waypoint altitude and oscillate at high altitudes; if the waypoint altitude is below 20 meters, the plane crashes.	Safety	Yes	Pending
24	P	AIRSPEED_STALL	Setting to large values causes the plane to crash to the ground.	Safety	Yes	Pending
25	C&P	SIM_CAN_SRV_MSK	Assigning special values (e.g., 10 or 30) causes the plane to lose power and crash.	Usability	No	Pending
26	C&P	BARO_ALT_OFFSET	Assigning large values results in the plane descending during flight and crashing.	Safety	Yes	Pending
27	C	MOT_PWM_MIN	When set to the maximum value, the copter rapidly ascends and deviates from waypoints. After some distance, it erroneously marks waypoints as reached, eventually recording all waypoints as completed despite not having actually reached them.	Safety	Yes	Yes
28	C	MOT_SPIN_MIN	Same as above.	Safety	Yes	Yes
29	C	SIM_GPS1_LAG_MS	Lack of proper value constraints allows large values to cause the copter to deviate from the planned flight path.	Usability	No	No
30	C	ANGLE_MAX, SIM_WIND_SPD	With a small ANGLE_MAX (e.g., 1600) and high wind speed, the copter cannot complete turns due to tilt angle limitations.	Safety	Yes	Yes
31	C	SIM_WIND_TURB	Values above 10 cause attitude oscillation and speed fluctuations, leading to flight path deviation.	Usability	No	No
32	C	AHRS_EKF_TYPE, AHRS_TRIM_X	Changing both parameters simultaneously causes the copter to deviate from its planned route and eventually crash.	Safety	Yes	Pending
33	P	RCMAP_ROLL	Assigning a special value (e.g., 6) to this parameter causes the plane to abort the mission and immediately enter a loitering state.	Safety	Yes	Pending
34	P	SIM_SERVO_SPEED	Assigning a low value to this parameter results in the plane lacking a fixed trajectory and deviating from the planned route.	Usability	No	No
35	R	ATC_SPEED_FF	Assigning the maximum value to this parameter causes the rover to deviate from the planned route and, after a period, incorrectly report waypoint completion even when distant from the waypoint (Similar to No.27).	Safety	Yes	Yes
36	R	SIM_CAN_SRV_MSK	Setting the third bit to 1 (e.g., 0100 or 1110) causes the rover to rapidly move in the direction opposite to the intended route.	Usability	No	Pending
37	R	BATT_ARM_MAH, SIM_TIME_JITTER	When both parameters are assigned large values, the rover fails to recognize arrival at the target waypoint, instead lingering nearby.	Usability	No	No
38	R	ATC_STR_RAT_FF	Assigning large values stalls the rover near the target, ending the navigation task.	Safety	Yes	Pending
39	R	FRAME_CLASS	Assigning large values stalls the rover near the target, ending the navigation task.	Safety	Yes	Pending
40	R	FENCE_RADIUS, FENCE_ENABLE	When these parameters are assigned after the rover has moved, it detects a fence violation but fails to trigger RTL, causing the vehicle to freeze instead of returning to the home position.	Safety	Yes	Pending
41	R	ATC_SPEED_D_FF	Assigning large values causes the rover to slow to zero speed near a waypoint, then quickly reverse course, repeatedly approaching and departing the waypoint in a loop.	Safety	Yes	Pending
42	C	SIM_BATT_VOLTAGE	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
43	C	SIM_TIME_JITTER, SIM_RATE_HZ	Setting SIM_TIME_JITTER to a large value with a low SIM_RATE_HZ causes a software crash after approximately two seconds of delay.	Usability	No	No
44	C	MAV_CMD_NAV_LOITER_TO_ALT	Excessive x/y/z values cause overflow and crash the flight control software.	Security	Yes	Yes
45	P	SIM_ACC_FILE_RW	When set to the special value 3, a delay of a few seconds results in "EOF on TCP stack", after which SITL becomes unresponsive.	Usability	No	Pending
46	R	AHRS_EKF_TYPE	Assigning the special value 2 triggers a floating-point exception, causing a rover software crash.	Security	Yes	Pending
47	R	INS_LOG_BAT_LGCT	Assigning the special value 0 triggers a floating-point exception, resulting in a rover software crash.	Security	Yes	Pending
48	R	SIM_WIND_T_ALT	Assigning the special value 0 triggers a floating-point exception, resulting in a rover software crash.	Usability	No	No
49	R	SIM_MAG_ALY_HGT	Assigning the special value 0 triggers a floating-point exception, resulting in a rover software crash.	Usability	No	No
50	R	SIM_CAN_SRV_MSK	Assigning special values 7 or 21 causes the rover to spin erratically, followed by a software crash due to floating-point overflow after one to two seconds.	Usability	No	No
51	C&P	SIM_RATE_HZ	Setting this value too low (0-22) triggers a floating-point exception and results in a software crash.	Usability	No	No
52	C&P	GPS2_TYPE	Assigning the special value 11 results in a segmentation fault and software crash.	Security	Yes	Pending
53	C&P	SIM_WIND_TURB	Assigning a large value to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
54	C&P	TERRAIN_SPACING	Setting this parameter to a value less than 1 (sub-meter accuracy) leads to floating-point overflow and software crash.	Security	Yes	Yes
55-57	C&P	SIM_IMU_POS_X/SIM_IMU_POS_Y/ SIM_IMU_POS_Z	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
58-62	C&P&R	INS_GYR0FFS_X/INS_GYR0FFS_Y/ INS_GYR2OFFS_X/INS_GYR2OFFS_Y/ INS_GYR2OFFS_Z	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Security	Yes	No
63	C&P&R	SIM_DRIFT_SPEED	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
64-69	C&P&R	SIM_ACC1_BIAS_X/SIM_ACC1_BIAS_Y/ SIM_ACC1_BIAS_Z/SIM_ACC2_BIAS_X/ SIM_ACC2_BIAS_Y/SIM_ACC2_BIAS_Z	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
70-75	C&P&R	SIM_GYR1_BIAS_X/SIM_GYR1_BIAS_Y/ SIM_GYR1_BIAS_Z/SIM_GYR2_BIAS_X/ SIM_GYR2_BIAS_Y/SIM_GYR2_BIAS_Z	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
76,77	C&P&R	SIM_GPS1_GLTCH_X/SIM_GPS1_GLTCH_Y	Assigning large values to this parameter results in a floating-point exception and subsequent software crash.	Usability	No	No
78,79	C&P&R	SIM_GPS1_HZ/ SIM_GPS2_HZ	Assigning the special value 0 triggers a floating-point exception and results in a software crash.	Usability	No	No
80	C&P&R	SIM_PLD_ENABLE	Assigning the special value 1 triggers a "PANIC" event, rendering the flight control software unresponsive.	Usability	No	Yes
81	C&P&R	SIM_SONAR_SCALE	Assigning the special value 0 triggers a floating-point exception and results in a software crash.	Usability	No	No
82	C&P&R	SERIAL_PASS2	Assigning special values 0 or 3 causes the software to become unresponsive and the UAV to display a "no link" message.	Security	Yes	Pending
83	C&P&R	SIM_GYR_FILE_RW	Assigning the special value 1 or 3 triggers a "PANIC" event, resulting in unresponsive flight control software.	Usability	No	Yes
84	C&P&R	MAV_CMD_CAN_FORWARD	Setting a large first value (e.g., [498, 0, 0, 0, 0, 0, 0]) triggers a segmentation fault and crashes the software.	Security	Yes	Yes
85	C&P&R	MAV_CMD_DO_SET_HOME	Supplying excessively large values in the x/y/z data fields (e.g., [x, x, x, x, 399, 300, 699]) causes overflow.	Security	Yes	Yes
86	C&P&R	MAV_CMD_DO_SET_ROI_LOCATION	Supplying excessively large values for x/y/z position data leads to overflow and subsequently causes software crashes.	Security	Yes	Yes
87	C&P&R	MAV_CMD_EXTERNAL_POSITION_ESTIMATE	Supplying excessively large values for x/y/z position data leads to overflow and software crashes.	Security	Yes	Yes

Conclusion

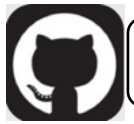
- Leverage Assignment Dependencies and Semantic Mapping to create targeted MIS for focused, high-efficiency testing.
- ADGFUZZ proves that Assignment Dependency is critical for uncovering RV logic bugs and is adaptable to other CPS.
- Open-source code and a pre-configured Ubuntu VM image are provided for reproducibility.

Thank You!

Q&A

Contact: Yuncheng Wang
Email: wangyuncheng@iie.ac.cn

Open-source: https://github.com/wyunc/ADGFuzz https://doi.org/10.5281/zenodo.16956667


Source Code

Virtual Machine