PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles

Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z. Berkay Celik, and Dongyan Xu
Purdue University

NDSS 2021
Workflow of robotic vehicles (RV)

- Physical space
  - Attitude, altitude, speed, etc.
- Cyber space
  - Measuring the RV’s current states
  - Adjusting actuators to reach target states

Three types of inputs:
- User commands
- Environment factors
- Parameters for control algorithm

Physical space:
- Vehicle
- Current altitude: 20 m
- Target altitude: 10 m

Cyber space:
- Sensor data gathering
- Control algorithm
- Commands to actuators
- Decreasing motors’ speed to lower altitude
Fuzzers for robotic vehicles (RV)

• Rule:
  • “Fail-safe mode must be triggered when the engine temperature is higher than 100 °C (212 °F)”

// Developers forget to convert F° to C° scale
if (temperature >= 100) {
  Fail-Safe -> execute();
}

Fail-safe is triggered under 100 F° (37 C°).

Can traditional fuzzers (AFL, libFuzzer) discover such a design flaw? No
- Mutation: Code coverage
- Bug oracle: Memory access violation
Fuzzers for robotic vehicles (RV)

- Can fuzzers specialized for RVs discover the design flaw?
  - RVFUZZER, CPI, etc.

```java
// Developers forget to convert F° to C° scale
if (temperature >= 100) {
    Fail-Safe -> execute();
}
```

Fail-safe is triggered under 100 F° (37 C°).

What about fuzzers for RVs? No
- Mutation & Bug oracle: unstable attitude


Overview of PGFUZZ

• Previous works do not
  • Know the RV's correct behaviors
  • Consider entire input space

• PGFUZZ

1. Creating formulas
2. Reducing fuzzing space
3. Building distance metrics
4. Mutating inputs

- Behavior-aware bug oracle
- Policy-guided mutation

Discover bugs
A vehicle must not deploy a parachute when the vehicle is:

1) In FLIP or ACRO flight modes
2) Climbing

\[ \neg \Box \{(Parachute=on)\} \land \{(Mode_t = FLIP/ACRO) \lor (ALT_t > ALT_{t-1})\} \]

The formula is created in the form of Metric temporal logic (MTL).
Finding inputs for mutation

• Huge fuzzing space
  • 1,140 configuration parameters
  • 58 user commands
  • 168 environmental factors

• Only mutating inputs relevant to the policy
Finding inputs for mutation

- Policy consists of terms (physical states)
  - Only mutating inputs related to the terms

- Decompose the formula into terms (states)

\[ \neg \diamond \{ \text{Parachute} = \text{on} \} \land \{ \text{Mode}_t = \text{FLIP/ACRO} \} \lor (\text{ALT}_t > \text{ALT}_{t-1}) \]

<table>
<thead>
<tr>
<th>Policy</th>
<th>Related terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parachute</td>
<td>Parachute</td>
</tr>
<tr>
<td></td>
<td>Flight mode</td>
</tr>
<tr>
<td></td>
<td>Altitude</td>
</tr>
</tbody>
</table>

<Policy-term map>
Mapping parameters to each term

- Static analysis to identify which states are affected by each parameter

![Diagram showing mapping of parameters to states and source code snippet]

- Configuration parameter
- Starting point for a def-use chain of TEMP parameter

A list of states:
- Altitude
- Roll
- Pitch
- Yaw

Source code snippet:
```
1: AP_GROUPINFO("TEMP", ..., ground_temp);
2: _user_temp = ground_temp + 273.15f;
3: temp = _user_temp
3: altitude = 153.8462f * temp * ...
```
Mapping other types of inputs to each term

- How to map environmental factors and user commands to each term from source code? Use an RV simulator!

1) Change motors’ speed

2) Log changed states

3) Change flight mode

<Changed states according to motors’ speed>

- Heading
- Throttle
- Altitude
- Climb
Two types of distances to mutate inputs

- Propositional distance
  - Goal: efficiently mutating inputs
  - Quantifies how close a proposition to the policy violation

\[ \neg \lozenge \{(\text{Parachute}=\text{on})\} \land \{(\text{Mode}_t = \text{FLIP/ACRO}) \lor (\text{ALT}_t > \text{ALT}_{t-1})\} \]

- Positive value: If the proposition is true
- Negative value: If the proposition is false

\[ P_1 = \begin{cases} 1 & \text{if parachute = on} \\ -1 & \text{if parachute = off} \end{cases} \]

\[ P_2 = \begin{cases} 1 & \text{if mode = FLIP/ACRO} \\ -1 & \text{if mode \neq FLIP/ACRO} \end{cases} \]

\[ P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t} \]

If the term is numeric, we use normalized difference.
Two types of distances to mutate inputs

• Global distance
  • Goal: detecting a policy violation

-1 × [Min\{P_1, \max(P_2, P_3)\}]

Positive value if there is no policy violation

Negative value if the RV violates the policy
Working example (time $T = 1$)

$$P_1 = \begin{cases} 
1 & \text{If parachute = on} \\
-1 & \text{If parachute = off} 
\end{cases}$$

$$P_2 = \begin{cases} 
1 & \text{If mode = FLIP/ACRO} \\
-1 & \text{If mode ≠ FLIP/ACRO} 
\end{cases}$$

$$P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t}$$

Randomly select an input and assign a random value to the selected input

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Parachute (on/off)</th>
<th>FLIP/ACRO mode (T/F)</th>
<th>Altitude (m)</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>Global distance</th>
<th>Next input for Time T+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>false</td>
<td>90</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Motor speed = 1,800$^1$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^1$ (Motor speed > 1,500) → increasing RV's altitude
(Motor speed < 1,500) → decreasing RV's altitude

$P_1$ = $\begin{cases} 
1 & \text{If parachute = on} \\
-1 & \text{If parachute = off} 
\end{cases}$

$P_2$ = $\begin{cases} 
1 & \text{If mode = FLIP/ACRO} \\
-1 & \text{If mode ≠ FLIP/ACRO} 
\end{cases}$

$P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t}$

Randomly select an input and assign a random value to the selected input

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Parachute (on/off)</th>
<th>FLIP/ACRO mode (T/F)</th>
<th>Altitude (m)</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>Global distance</th>
<th>Next input for Time T+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>false</td>
<td>90</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Motor speed = 1,800$^1$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^1$ (Motor speed > 1,500) → increasing RV's altitude
(Motor speed < 1,500) → decreasing RV's altitude
Working example (time $T = 2$)

$P_1 = \begin{cases} 
1 & \text{If parachute = on} \\
-1 & \text{If parachute = off} 
\end{cases}$

$P_2 = \begin{cases} 
1 & \text{If mode = FLIP/ACRO} \\
-1 & \text{If mode ≠ FLIP/ACRO} 
\end{cases}$

$P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t}$

$-1 \times [\text{Min} (P_1, \text{Max} (P_2, P_3))]$

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Parachute (on/off)</th>
<th>FLIP/ACRO mode (T/F)</th>
<th>Altitude (m)</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>Global distance</th>
<th>Next input for Time T+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>false</td>
<td>90</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Motor speed = 1,800$^1$</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>false</td>
<td>100</td>
<td>-1</td>
<td>-1</td>
<td>0.1</td>
<td>1</td>
<td>Motor speed = 1,800$^1$</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1) We log (motor, 1,800) because the input increases P3.

2) PGFUZZ selects an input and assigns a random value to the selected input.

3) When the selected input increased a distance before, we reuse the input and value pair (motor, 1,800)

1) (Motor speed > 1,500) $\rightarrow$ increasing RV’s altitude
(Motor speed < 1,500) $\rightarrow$ decreasing RV’s altitude
Working example (time T = 3)

\[ P_1 = \begin{cases} 
1 & \text{If parachute = on} \\
-1 & \text{If parachute = off} 
\end{cases} \]

\[ P_2 = \begin{cases} 
1 & \text{If mode = FLIP/ACRO} \\
-1 & \text{If mode ≠ FLIP/ACRO} 
\end{cases} \]

\[ P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t} \]

\[ -1 \times [\text{Min}\{P_1, \text{Max}(P_2, P_3)\}] \]

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Parachute (on/off)</th>
<th>FLIP/ACRO mode (T/F)</th>
<th>Altitude (m)</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>Global distance</th>
<th>Next input for Time T+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>false</td>
<td>90</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Motor speed = 1,800^1)</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>false</td>
<td>100</td>
<td>-1</td>
<td>-1</td>
<td>0.1</td>
<td>1</td>
<td>Motor speed = 1,800</td>
</tr>
<tr>
<td>3</td>
<td>off</td>
<td>false</td>
<td>110</td>
<td>-1</td>
<td>-1</td>
<td>0.09</td>
<td>1</td>
<td>Parachute = on</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1) (Motor speed > 1,500) → increasing RV’s altitude
(Motor speed < 1,500) → decreasing RV’s altitude
Working example (time $T = 4$)

$$P_1 = \begin{cases} 
1 & \text{If parachute = on} \\
-1 & \text{If parachute = off}
\end{cases}$$

$$P_2 = \begin{cases} 
1 & \text{If mode = FLIP/ACRO} \\
-1 & \text{If mode ≠ FLIP/ACRO}
\end{cases}$$

$$P_3 = \frac{\text{ALT}_t - \text{ALT}_{t-1}}{\text{ALT}_t}$$

-1 X [Min{$P_1$, Max{$P_2$, $P_3$}}]

<table>
<thead>
<tr>
<th>Time (T)</th>
<th>Parachute (on/off)</th>
<th>FLIP/ACRO mode (T/F)</th>
<th>Altitude (m)</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>Global distance</th>
<th>Next input for Time T+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>false</td>
<td>90</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>Motor speed = 1,800$^1$</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>false</td>
<td>100</td>
<td>-1</td>
<td>-1</td>
<td>0.1</td>
<td>1</td>
<td>Motor speed = 1,800</td>
</tr>
<tr>
<td>3</td>
<td>off</td>
<td>false</td>
<td>110</td>
<td>-1</td>
<td>-1</td>
<td>0.09</td>
<td>1</td>
<td>Parachute = on</td>
</tr>
<tr>
<td>4</td>
<td>on</td>
<td>false</td>
<td>112</td>
<td>1</td>
<td>-1</td>
<td>0.02</td>
<td>-0.02</td>
<td>Policy violation!</td>
</tr>
</tbody>
</table>

1) (Motor speed $>$ 1,500) $\rightarrow$ increasing RV’s altitude
(Motor speed $<$ 1,500) $\rightarrow$ decreasing RV’s altitude

Vehicle must not increase its altitude
Evaluation

• RV control software
  • ArduPilot, PX4, and Paparazzi

• 56 extracted policies
  • Fuzzing 48 hours per each control software
  • Violating 14 policies in the three-control software

• Found 156 bugs
Case study

• Policy
  • “If time exceeds COM_POS_FS_DELAY seconds after GPS loss is detected, the GPS fail-safe must be triggered”
Case study

• Fail to trigger the GPS fail-safe under
  • COM_POS_FS_DELAY = -1

PX4 maintains the ORBIT flight mode under GPS signal loss.
Conclusion

- Novel fuzzing approach to find logic bugs
  - Behavior-aware bug oracle
    - Leverage policies (MTL formulas)
  - Policy-guided mutation
    - Propositional and global distances

- 156 previously unknown bugs
  - 128 out of 156 found bugs can only be discovered by PGFUZZ.
  - 106 bugs have been acknowledged
  - 9 bugs have been patched
Thank you! Questions?

kim2956@purdue.edu
Backup slides
Safety bug in real world

- Boeing-737 Max airplanes
  - Crashed due to a design flaw
  - Lowered its altitude based on only one broken sensor

How can we find such a critical bug in flight control software? Um… fuzzing?

The plane’s sensors took different readings

- Incorrectly measured sensor values

Source: Ethiopian Aircraft Accident Investigation Bureau

Threat model (1)

• Developers are benign
  • Incorrectly design or make buggy code

• Users are also benign
  • Unintentionally trigger the buggy code
Threat model (2)

- Attackers control three types of inputs
  - Further, they can wait until suitable conditions

- Attackers’ goal
  - Stealthily triggering buggy code via sending inputs that looks innocent

- The followings are out of scope
  - Physical sensor attacks
  - Malicious code injections
Evaluation

• RV control software
  • ArduPilot, PX4, and Paparazzi

• 56 extracted policies
  • Fuzzing 48 hours per each control software
  • Violating 14 policies in the three-control software

• Found 156 bugs

<table>
<thead>
<tr>
<th>Physical effect</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unstable attitude</td>
<td>Software</td>
<td>Unexpected</td>
</tr>
<tr>
<td></td>
<td>crash</td>
<td>behavior</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>Total: 156</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, failing to trigger GPS fail-safe mode
Outline

• Defining RV’s correct behaviors as formulas
• Reducing fuzzing space
• Building distance metrics
• Evaluation