ROS Fuzzing 기술 비교 분석 및 동향 연구

박종현 [1], 오현영 [2*]
[1] 가천대학교 인공지능학과 석사과정
[2] 가천대학교 인공지능학과 교수

hynmail@naver.com, hyoh@gachon.ac.kr

Comparative Analysis and Trends in ROS Fuzzing

Jong-Hyun Park, Hyunyoung Oh
Dept. of AI, Gachon University

요 약

로봇 기술의 고도화로 정밀 제어가 가능해졌고, ROS(Robot Operating System)는 분산 노드 구조와 DDS 기반 통신을 제공하는 개방형 미들웨어로 사실상 표준 개발 기반으로 자리 잡았다. 그러나 로봇 응용의 다양성과 복잡성으로 인해 ROS 기반 시스템의 결함을 수작업으로 찾아내는 일은 비효율적이다. 이에 자동 입력 생성과 실행 피드백을 활용하는 퍼징(fuzzing) 이 ROS 환경에도 적용되고 있다. 본 논문은 대표적 ROS 퍼저의 설계 축(대상 계층, mutation 모델, feedback 원천)과 플랫폼 축(ROS 1/ROS2)을 기준으로 SMACH, RoboFuzz, R2D2, ROFER 를 비교, 분석하고, 각 접근의 강점과 한계, 향후 연구 방향을 논의한다.

1. 서론

최근 로봇 산업 분야는 기술의 발전을 통해 더욱 정밀 제어가 가능해졌다. 이를 통해 산업, 물류, 군수 등의 많은 분야로 대중화가 확산되었다. 로봇 분야를 더욱 정밀하게 제어하기 위해서는 ROS (Robot Operating System)을 많이 사용한다. ROS 는 분산 노드 구조, DDS 기반 통신 등의 기능을 제공하는 오픈 소스 미들웨어이며 드론, 자율주행, 제조, 물류 분야에서 사실상 표준적인 개발 기반으로 자리 잡고 있다. 특히 원격으로 조정이 가능한 점과 무인으로 사용이 가능하다는 점에서 현재도 많은 연구가 진행 중이다. 원격과 무인 시스템이라는 점에서 ROS 의 신뢰성과 안정성은 문제는 필수적으로 해결해야 할 문제이다.

그러나 ROS 도 다른 SW 시스템과 마찬가지로 몇가지의 결함을 가지고 있다. 이들의 결함의 종류는 입력 값의 범위 초과, 행동 논리 오류, 순서 타이밍 오류, Correctness 오류. 메모리 버그 오류 등의 오류가 있다[1,2,3]. 하지만 ROS 는 각 로봇 시스템에 따라 맞춰 설계된 시스템이기 때문에 각각의 오류를 사람이 직접 찾아내는 것은 많은 비용과 시간이 요구된다.

이를 해결하기 위하여 ROS 시스템에 맞춰 제작된 ROS Fuzzing 이 주목받고 있다. 이들은 기존의 Fuzzing 과 마찬가지로 Fuzzing 대상인 ROS 에 오류를 유발하는 코드를 주입하여 오류를 탐지하고 이를 수정할 수 있도록 돕는 역할을 한다. 기존의 Fuzzing 은 메모리 안전 중심, 순서 타이밍 결합 등의 내용에는 부적절하여 이에 더욱 특화된 ROS Fuzzing 이 제작되었다. 본 논문에서는 이러한 ROS Fuzzing 들의 기술적인 특징과 Fuzzing 결과를 정리하고 이를 비교 분석한 후 각각의 한계점 지적, 향후 발전 방향성을 제시하는 것으로 마무리한다.

2. ROS Fuzzing 의 기술 및 결과 정리

ROS Fuzzing 은 두 분야에서 두가지로 나눌 수 있다. 1)플랫폼 축: Fuzzing 대상 ROS 가 ROS1 인지 ROS2 인지를 통한 분류이다. 2)설계 축: 대상 계층, Mutation 모델, feedback 원천 등으로 구분할 수 있다.

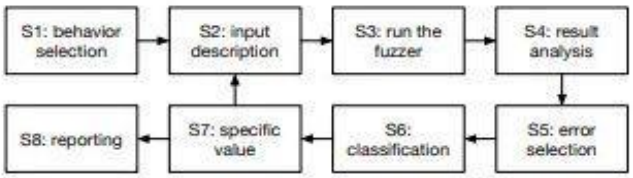
또한 ROS Fuzzing 은 모두 (1)초기 입력 갑 생성→(2)입력 합성 및 변이→(3) 시스템 실행→(4) 관측 및 오라클/버그 평가 →(5)피드백 계산 → (6)피드백 값을 통한 입력 갑 재생성이라는 파이프라인을 가지고 있다. 대부분의 ROS Fuzzing 이 주목한 부분은 Mutation 과 Feedback 두가지 분야를 집중적으로 개발하여 ROS Fuzzing 의 기능을 향상시켰다.

2.1 SMACH

SMACH[4]는 로봇의 행동 논리/스키마 오류 주목해

* 교신저자

제작한 Fuzzing 이다. SMACH 는 로봇의 행동을 기술하는 SMACH 상태(사용자가 정의하는 작업의 단위) 기계를 대상으로 한 grammar-based Fuzzing 방식이다. Fuzzing 의 대상은 ROS1 이다. 상태에 투입할 입력을 문법에 맞게 자동 생성하여 white-box(개별 상태), black-box(전체 상태) 수준에서 시험하며 Python 동적 타이핑으로 인한 입력 타입 불분명 문제는 seed 실행에서 전달 된 userdata 구조를 로깅해 타입을 추정하는 state-monitoring 으로 해결한다



(그림 1) smach 프로세스 구조도[4]

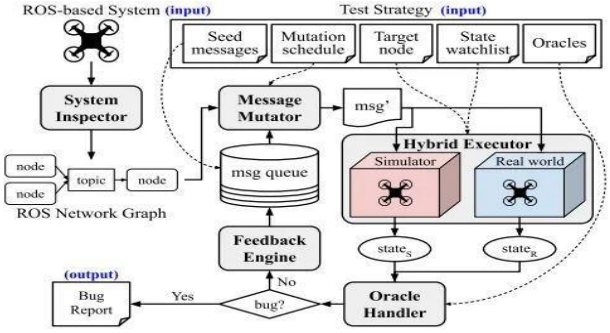
SMACH 흐름 구조는 (그림 1)의 방식으로 진행한다 행동 선정→입력 기술→퍼저 실행→결과 분석→오류 선별·분류→리포팅의 S1–S8 절차로 진행된다. Fuzzer 는 grammar 로 생성한 입력을 각 상태에 여러 차례 주입하여 로그(입력 값·outcome·오류·실행시간)를 축적하고, 잘못된 타입/부적절 값이 탐지되면 grammar 를 정제해 탐색을 이어간다. black-box 모드에서는 외부 센서 값을 모사하는 입력을 상태 기계에 주입해 잡음, 교란에 대한 강건성을 점검한다.

SMACH 의 평가는 SoftBank Pepper 기반 UChile Peppers 팀의 실제 SW 적용되었고, 6 건의 오류를 발견하여 실무자 패널 검증을 받았다. 오류 유형은 문법 오류, 데이터 처리 오류, 로직 오류, 아키텍처 설정 오류로 분류되며, 패널은 이 상황들을 대체로 중요한 문제로 인식했다. 그러나 논문은 단일 로봇 사례에 대한 결과라는 점을 한계로 명시한다.

2.2 RoboFuzz

RoboFuzz[3]는 시뮬레이션과 현실의 괴리에 주목했다. 따라서 시뮬레이션을 통한 Fuzzing(SW)과 실제 물리적인 로봇의 행동을 직접 보고 파악할 수 있는 Fuzzing(HW)이 모두 가능하다는 장점이 있다. [RoboFuzz 는 로봇을 데이터-흐름 관점에서 테스트하도록 설계되며 전체적인 구조는 (그림 2)를 통해 볼 수 있다.

(그림 2)는 그래프 → 변이 → 하이브리드 실행 → 오라클 → 피드백 루프를 두 줄로 요약한다. 먼저 System Inspector 가 ROS 네트워크 그래프 (node-topic-node)를 추출하고, 상단의 Test Strategy 를 기준으로 Message Mutator 가 타입 인지 변이로 형식적으로 유효한 msg′ 를 만든다. 생성된 입력은 Hybrid Executor 로 들어가 시뮬레이터와 현실에 동일 주입되고, 각각의 상태(state_s, state_r)가 수집된다. Oracle Handler 는 안전 불



(그림 2) RoboFuzz 구조도[3]

변식·물리 제약·명세 규칙 및 시뮬–현실 일치성까지 확인해 위반을 Bug Report 로 기록한다. 오라클 위반이 없더라도 Feedback Engine 이 실행의 흥미도(interestingness)를 갱신해 유망한 시드를 승격하고, 변이 방향을 가이드하며 다음 라운드를 반복한다. 시뮬레이터–현실 비교는 이 루프의 핵심 강점이다.

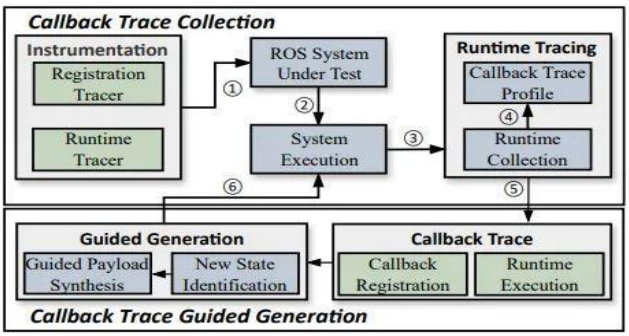
RoboFuzz 의 결과는 시뮬 전용(S)·물리 부정합(P)·사이버-피지컬 괴리(D)·ROS 소프트웨어(R)를 아우르는 신규 정확성 버그 30 건을 보고하고, 이 중 13 건은 ROS 내부 계층에서 확인되며 입력 리플레이로 오탐 아님을 검증한다. 또한 RoboFuzz 는 SMACH 와 마찬가지로 Github 저장소에 공개되어 있기 때문에 재현 및 확장이 가능하다.

2.3 R2D2

R2D2[5]는 callback trace 를 실시간으로 수집·프로파일해 상태 기반으로 가이드하는 Fuzzing 을 수행하는 ROS 전용 Fuzzer 이다. R2D2 의 ROS 는 분산 메시지·콜백 중심으로 동작해 코드 커버리지만으로는 상태 변화를 포착하기 어렵다는 기존의 한계점에 주목하여 코드 커버리지 의존을 아예 제거한 뒤 callback trace 상태 전이를 식별해 입력을 진화시키는 접근을 제안한다. 이를 위해 ROS 런타임에 커스텀 트레이서를 삽입해 콜백 등록·스케줄·시작/종료와 메시지 발행/수신 시각·버퍼 크기 등을 수집하고, 이를 callback latency/message latency 벡터로 구성된 trace 로 요약한다. 이렇게 얻은 상태 식별자를 피드백으로 사용해 새로운 상태를 유도하는 페이로드에 가중치를 주어 다음 변이를 안내한다.

(그림 3)에 보이는 것처럼 R2D2 의 워크플로는 두 단계다. 조금 더 세부적인 절차를 살펴보면 다음과 같다. 1) rclcpp_callback_init/ rcl_callback_init 로 시작하고 2) executor_execute/ callback_start / callback_end / rcl_tack (런타임) 등으로 다층(ROS 런타임)에서 상호작용을 추적하며 3) 해당 이벤트를 공유 메모리에 실시간 기록한다. 4) 수집한 trace 로 callback 그래프(순서)와 전역 지연/처리량 기준선을 유지하고 5) 새 시퀀스 출현, 특정 callback 지연의 통계적 이탈, 메시지 처리량 급감을 새

상태로 판정한다. 6) 입력 합성은 인터페이스 추출기가 토픽/서비스의 타입·포맷을 자동 수집해 스퀘 준수 페이로드를 만들고, 이전에 새 상태를 만든 seed 를 우선 변이한다. 이 설계는 ROS 는 정해진 제어흐름으로 초기 커버리지가 금방 포화된다는 것을 현실을 반영하며, 순서·타이밍·스케줄 차이를 직접 가이드 신호로 삼는 것이 핵심이다.

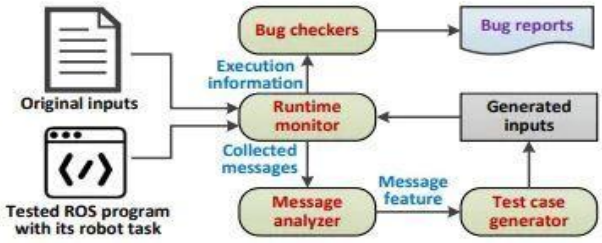


(그림 3) R2D2 전체 구조도[5]

R2D2 논문의 자체 평가에서 R2D2 는 Nav2, Turtlebot3, Turtlesim, Autoware 에 적용되어 미확인 버그 39 건을 찾아냈고, 이 중 6 건이 실제 수정되었다. 코드 커버리지는 RoboFuzz 대비 2.56× 향상되었으며, 계측 오버헤드는 지연 10.4%, 메모리 1.0% 수준으로 보고된다. 논문은 또한 커버리지 지표가 동일해도 callback 순서/타이밍 차이로 물리적 결과가 달라질 수 있음을 동기 예제로 제시하고, callback trace 이런 내부 상태 전이를 가늠하는 더 민감한 지표임을 보인다. 요약하면, R2D2 는 런타임 상태 인지 피드백으로 ROS 의 방대한 상태 공간을 효율적으로 탐색하도록 설계된 callback-tracing guided Fuzzing 의 레퍼런스 구현이다.

2.4 ROFER

ROFER[6]는 ROS 의 분산 상태 메시지 동학과 다차원 조합 결함에 주목하여 제작하였다. ROFER 의 특징은 다차원 조합의 오류에 주목한 만큼 다차원 입력을 하여 4 가지 차원의 서로 다른 입력 값이 생긴다는 것이다. ROFER 는 (그림 4)를 통해서 구조를 살펴보면 ROS 프로그램을 Runtime monitor → Message analyzer → Test case generator → Bug checkers 로 잇는 루프로 Fuzzing 한다. Runtime monitor 가 생성 입력으로 로봇 태스크를 실행하며 실행 정보와 메시지를 수집하고, Message analyzer 가 이를 message feature 로 요약하여 프로그램 피드백으로 제공한다. Test case generator 는 dimension-level mutation (유저 명령·설정, 파라미터·센서, 데이터·메시지 순서 등 차원별 기여도를 반영한 변이)로 다음 입력을 합성한다. 이 과정은 rosbag 을 사용해 코드 계측 없이 C++/Python 등 언어에 무관하게 적용되며, 필요시 ASan/Msan 등 외부 bug checker 와 결합한다. 논문은 메시지 데이터의 변동성에서 상태 전이를 반영하는 피크 특성을 쓰고, 랜덤 교란을통계적 피크 검출과 이산화로 제거하여 ROS Fuzzing 의 효율을 높였음을 설명한다.



(그림 4) ROFER 다차원 입력 방식 구조도[6]

자체 평가에서 ROFER 는 ROS2 실제 패키지 13 종을 Gazebo 가상 로봇으로 24 시간씩 Fuzzing 하여 실제버그 88 건을 보고한다. 동등 조건 비교에서 ROFER 는 Ros2-fuzz/ASTAA-like/RoboFuzz/ROZZ 대비 분기 커버리지 +687%/+15%/ +7% / +3%, message-feature 커버리지 +308% / +68% / +67%로 가장 넓은 탐색을 달성하고, 버그를 84/70/81/45 건 더 발견한다. 또한 49/88 은 두 개 이상 입력 차원의 특수 조합이 트리거한 복합 결함으로, 차원 선택을 학습하는 설계의 효과를 뒷받침한다. 한편 저자들은 현 버전이 메모리/예외 계열 중심이고, 실험이 시뮬레이션 환경에서 수행되었음을 명시한다. 요약하면, 커버리지·버그 수·언어 일반성 기준에서 ROFER 는 현시점 가장 설득력 있는 ROS Fuzzer 로 평가된다.

3 ROS Fuzzing 비교 분석

	대상 ROS 버전	Fuzzing 대상	Mutation 방식	오픈소스화
SMACH	ROS1	행동/논리 오류	Grammar-based	Git-hub 존재[7]
RoboFuzz	ROS2	Correctness 오류	타입 인지 변이	Git-hub 존재[8]
R2D2	ROS2	순서/타이밍/동시성 오류	callback 등록/실행	공개 코드 없음
ROFER	ROS2	메모리 오류	다차원 입력	공개 코드 없음

(표 1) ROS Fuzzing 특징 분석

앞에서 언급된 4 가지의 방식을 비교한 (표 1)을 보면 각 방식은 서로 다른 주요 목적을 지닌다. ROFER 는 ROS2 에서 메모리/예외 계열 버그를 넓게 캐치하도록 설계되어 있고, 다차원 입력선택으로 탐색 폭을 키운다. R2D2 는 같은 ROS2 이지만 표적이 전혀 다르다. 콜백 순서·지연·처리량을 추적해 타이밍/동시성 결함을 드러내는 런타임 중심 접근이다. RoboFuzz 는 타입 인지 변이 + 의미론 오라클로 정확성(correctness) 위반을 잡는 데 강하고, SMACH 는 ROS1/SMACH 에서 행동(상태·전이) 논리의 설계·코딩 오류를 찾는 데

특화되어 있다. 즉, “어떤 결함을 우선 가정하느냐”에 따라 선택지가 달라진다: 메모리/예외(ROFER) ↔ 타이밍/동시성(R2D2) ↔ 의미론/Sim2Real(RoboFuzz) ↔ 행동 로직(SMACH). 실용성과 이식성을 보면 SMACH RoboFuzz 는 공개 저장소가 있어 재현성이 높은 편이고, R2D2·ROFER 는 코드 접근성이 낮아 외부 재현·적용 장벽이 있다. 또한 ROS1 ↔ ROS2 불일치가 있어 행동 로직 점검은 SMACH (ROS1), 시스템 전역 결함은 ROFER/R2D2/RoboFuzz (ROS2)로 스택 분리가 불가피하다. 따라서 ROS2 실무 환경이면 ROFER (메모리/예외) + R2D2 (타이밍/동시성)를 기본 조합으로, 정확성·Sim2Real 리스크가 크면 RoboFuzz 를 보완 투입, 초기 설계 단계에서 행동 논리 안정화가 필요하면 SMACH 로 선제 검증하는 구성이 합리적이다.

4.ROS Fuzzing 의 한계점 정리

SMACH 는 ROS1/SMACH(Python) 생태계를 전제로 하여 ROS2 및 BehaviorTree.CPP 기반 워크플로와의 상호운용성이 낮고, 분산 callback, QoS, 타이밍 등 시스템 수준 결함에 대한 관측 범위가 제한적이다. RoboFuzz 는 의미론 오라클 기반 정확성 점검과 시뮬레이터·실기 하이브리드 실행을 강점으로 하나, 오라클 설계·튜닝의 도메인 의존성과 맵/센서 정합화 등 실행 환경 준비 비용으로 인해 재현성·이식성 측면의 부담이 크다. R2D2 는 callback 추적을 피드백으로 활용하여 순서·지연·동시성 결함 탐지에는 효과적이나, 물리 제약 및 업무 규칙과 같은 의미론적 정합성 위반을 직접 판별하지 못한다. ROFER 는 메시지 기반 피드백과 차원 선택 변이를 통해 탐색 효율을 높이지만, 시뮬레이션 중심 평가와 메모리/예외 계열 편중으로 Sim-to-Real 및 의미론적 정확성 측면의 포괄성이 제한된다.

기법에 공통적으로는 다음의 구조적 한계가 관찰된다. 첫째, ROS1↔ROS2 및 배포판/버전 차이로 인한 표준화·이식성 부족으로 도구 적용 범위가 협소하다. 둘째, 대용량 Docker 이미지와 특정 빌드 체인 의존으로 실행 환경 장벽이 높아 초기 도입 비용이 크다. 셋째, 오픈소스 코드·실험 아티팩트 공개 수준의 불균일로 재현성 및 비교 가능성이 저하된다. 넷째, 자동 오라클 부재로 도메인 지식 기반의 수작업 설정 비용이 상당하다. 다섯째, HIL/실기 기반 벤치마크의 부족으로 시뮬레이션 성과의 외적 타당성 확보가 어렵다.

5. 결론 및 향후 연구 방향성 제시

ROFER 는 ROS2 프로그램에서 message feature 를 피드백으로, 차원 선택 변이로 입력을 합성하여 코드 계측 없이 rosbag 만으로 C++/Python 모두에 적용 가능하고, 커버리지·결함 발견 수에서 동시대 도구를 상회해 현시점 가장 효과적인 ROS Fuzzing 으로 평가된다. 대비해 RoboFuzz 는 type-aware 변이 + 시뮬레이터·실기 하이브리드 실행 + 의미론 오라클로 정확성/Sim2Real 괴리 판정에 강점이 있고, R2D2 는 callback 추적을 피드백으로 활용해 순서·지연·동시성 결함을 효과적으로 탐지하며, SMACH 는 grammar-based 로 상태기계의 행동 논리/전이 오류를 조기에 포착한다. 즉, ROFER 는 분산 메시지 동학·다차원 조합에서 폭넓은 탐색과 메모리/예외 결함 탐지에 유리하고, 의미론·Sim2Real·정밀 타이밍/행동 논리 영역은 각각 RoboFuzz, R2D2, SMACH 가 상호 보완한다.

ROFER 의 공개구현, 아티팩트가 제한되고 ROS1↔ROS2 불일치·컨테이너 의존성 등으로 재현성·이식성이 저하된다. 향후에는 (1) 표준 어댑터 레이어 (Topic/Service/Action/Parameter, QoS/Executor 이벤트)와 플러그형 피드백 (coverage, message-feature, callback-trace, semantic-oracle, sanitizer) 통합, (2) Sim→HIL→Real 공개 벤치마크·리플레이 데이터셋, (3) 자동 오라클(정책·물리 제약 명세/학습)과 LLM-보조 입력·시퀀스 생성 결합, (4) 경량 Docker/colcon 템플릿 기반 원 클릭 재현 환경을 제안한다. 이는 ROFER 의 탐색 효율을 유지하면서 의미론, 실세계 타당성·재현성을 균형 있게 확장하는 실질적 경로다.

사사문구

이 논문은 2025 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. RS-2024-00337414, SW 공급망 운영환경에서 역공학 한계를 넘어서는 자동화된 마이크로 보안 패치 기술 개발)과 한국산업기술기획평가원의 지원(No. RS-2024-00406121,
자동차보안취약점기반위협분석시스템개발(R&D))을 받아 수행된 연구 결과임.

참고문헌

[1] C. S. Timperley et al., “ROBUST: 221 Bugs in the Robot Operating System,” Empirical Software Engineering, vol. 29, no. 3, Art. 57, 2024.
[2] J.-P. Ore et al., “Dimensional Inconsistencies in Code and ROS Messages: A Study of 5.9M Lines of Code,” in Proc. IEEE/RSJ IROS 2017, Vancouver, Canada, 2017, pp. 712–718.
[3] S. Kim and T. Kim, “RoboFuzz: Fuzzing Robotic Systems over ROS for Finding Correctness Bugs,” in Proc. ESEC/FSE ’22, Singapore, 2022
[4] R. Delgado et al., “Fuzz Testing in Behavior-Based Robotics,” in Proc. IEEE ICRA 2021, Xi’an, China, 2021, pp. 9375–9381.
[5] Y. Shen et al., “Enhancing ROS System Fuzzing through Callback Tracing,” in Proc. ACM ISSTA 2024, Vienna, Austria, 2024, pp. 76–87.
[6] J.-J. Bai et al., “Multi-Dimensional and Message-Guided Fuzzing for Robotic Programs in ROS (ROFER),” in Proc. ACM ASPLOS ’24, San Diego (La Jolla), USA, 2024.
[7] R. Delgado, fuzz_testing, GitHub repository. Available: https://github.com/rdelgadov/fuzz_testing
[8] S. Kim and T. Kim, RoboFuzz, GitHub repository. Available: https://github.com/sslab-gatech/RoboFuzz