ROS1과 ROS2의 구조적 차이와 자율주행 시스템 적용 관점에서의 비교 분석

박재원, 유경민, 남승우, 장윤성, 김주성, 백의준, 김명섭*

고려대학교

{2018270614, rudals2710, nam131119, brave1094, jsung0514, pb1069, *tmskim}@korea.ac.kr

Structural Differences Between ROS1 and ROS2 and Comparative Analysis in the Context of Autonomous Driving Systems

Jae-Won Park, Gyeong-Min Yu, Seung-Woo Nam, Ui-Jun Baek, Myung-Sup Kim*

Korea University

요 약

로봇 운영체제(ROS)는 로봇 분야에서 표준적인 프레임워크로 자리 잡았으며, 그 중 ROS1은 빠른 프로토타이핑에 적합한 구조로 널리 활용되었다. 그러나 중앙 마스터 노드에 대한 의존성, 실시간성 부족, 보안 미지원 등의 한계로 인해 자율주행 차량과 같은 안전이 중요한 시스템에는 적용에 제약이 있었다. 이러한 한계를 극복하기 위해 개발된 ROS2는 DDS(Data Distribution Service) 기반의 분산 아키텍처를 채택하여, 실시간성, 결함 내성, QoS 조정, 보안성 등 다양한 측면에서 기능이 향상되었다. 본 논문은 ROS1과 ROS2의 구조적 차이를 통신 구조, 프로토콜, 플랫폼 지원, 보안 측면에서 비교 분석하고, 자율주행 시스템 관점에서 각 버전의 장단점을 평가한다. 또한 Autoware.Auto와 Apex.OS 사례를 통해 ROS2의 산업적 실용성과 안정성을 입증하고, 차세대 로봇 시스템 도입을 고려하는 실무자들에게 ROS2 기반 시스템의 이해와 적용 전략 수립에 실질적인 인사이트를 제공한다.

Ⅰ. 서 론

로봇 운영체제(ROS, Robot Operating System)는 센서 처리, 제어, 시뮬레이션 등 로봇 개발에 필수적인 기능을 제공하는 오픈소스 프레임워크로, 전 세계적으로 학계와 산업계에서 널리 사용되고 있다[1]. 초기 버전인 ROS1은 빠른 프로토타입 개발과 연구에 적합한 환경을 제공하지만, 중앙 마스터 노드에 의존하는 구조적 특성으로 인해 몇 가지 중요한 한계를 지닌다. ROS1의 중앙 집중식 아키텍처는 단일 장애점(Single Point of Failure) 발생 가능성, 실시간성 부족, 보안 미지원 등의 문제를 일으킬 수 있다[1][3]. 이러한 문제들은 자율주행 차량과 같은 고신뢰성 및 실시간 처리 능력이 요구되는 시스템에 ROS1을 적용하는 데 큰 어려움을 초래한다.

이러한 한계를 극복하기 위해 개발된 ROS2는 DDS(Data Distribution Service)를 기반으로 한 분산형 통신 아키텍처를 채택하여, 시스템의 실시간성, 결함 내성, QoS(서비스 품질) 조정, 보안성 등에서 크게 개선된 기능을 제공한다[2][3]. ROS2는 중앙 마스터 노드의 의존성을 제거하고, 다양한 운영체제 및 임베디드 환경을 지원함으로써 산업 응용에 적합한 구조를 갖추고 있다. 이로 인해, ROS2는 자율주행 시스템을 포함한 고신뢰성, 고성능을 요구하는 로봇 시스템 설계에서 중요한 역할을 할 수 있게 되었다.

본 논문에서는 ROS1과 ROS2의 구조적 차이를 통신 구조, 프로토콜, 실시간성, 플랫폼 호환성, 보안성 측면에서 비교하고, 자율주행 시스템에서의 적용 사례를 통해 각 버전의 장단점을 분석한다. 이를 통해 ROS2의 기술적 우수성과 실용 가능성을 입증하고, 자율주행 시스템을 포함한 차세대 로봇 시스템 설계에 필요한 기초 자료를 제공하고자 한다.

Ⅱ. 본론

2.1.1 통신 아키텍처

ROS1은 중앙집중식 아키텍처를 채택하고 있으며, 모든 노드는 ROS 마스터(ROS Master) 노드에 등록한 후 다른 노드와 통신을 수행한다. 마스터 노드는 각 노드의 토픽, 서비스 정보 등을 관리하고, 연결을 설정하는 역할을 한다. 또한, XML-RPC 기반의 프로토콜을 통해 이름 해석(name resolution)과 통신 경로 설정을 지원한다[1]. 그러나 이러한 구조는 마스터 노드의 장애[1] 시 전체 시스템이 동작을 멈추는 단일 장애점(Single Point of Failure)의 문제를 초래하며, 시스템의 확장성에 제한을 두어 분산 노드의 확장 시 통신 지연이나 병목현상이 발생할 수 있다[1].

반면, ROS2는 DDS(Data Distribution Service)를 기반으로 한 분산형 통신 아키텍처를 채택하여, 별도의 마스터 노드 없이 각 노드가 네트워크 상에서 동적으로 발견되고 통신을 설정하는 구조를 구현하였다[1]. DDS의 discovery 메커니즘은 네트워크 내에서 노드들이 서로를 자동으로 인식하고 연결을 설정할 수 있도록 한다. 각 노드는 퍼블리셔와 서브스크라이버 관계를 설정하여 데이터를 주고받으며, 이를 통해 시스템의 확장성과 결함 내성을 크게 향상시킬 수 있다. 또한, 네트워크 구성 변화에도 유연하게 대응할 수 있어, 시스템 안정성을 높이고 분산 환경에서 더 효과적으로 운영될 수 있다[2].

이러한 구조적 차이는 로봇 시스템이 소형 단일 장비에서 동작할 때보다, 자율주행 차량과 같은 복잡한 분산 시스템에서 더욱 중요한 영향을 미친다. ROS2는 여러 ECU(Electronic Control Unit) 간의 분산 통신을 지원하며, 자율주행 시스템의 안정성과 유연성을 동시에 확보할 수 있도록 설계되었다. 이는 자율주행 차량의 다양한 센서와 제어 시스템들이 실시간

으로 데이터를 주고받고, 통합된 환경에서 원활하게 동작할 수 있도록 도와준다.

2.1.2 통신 프로토콜 및 QoS

ROS1은 기본적으로 TCPROS 및 UDPROS라는 자체 정의 프로토콜을 사용하여 통신을 수행한다[1]. TCPROS는 신뢰성 있는 통신을 보장하지만, 대역폭이 크거나 지연에 민감한 환경에서는 성능 저하가 발생할 수 있다. UDPROS는 신뢰성을 포기하고 멀티캐스트 전송을 가능하게 하지만, 실질적으로 활용되는 빈도는 상대적으로 낮다. 또한, ROS1에서는 QoS(서비스 품질)를 세부적으로 설정할 수 없어, 센서 데이터와 제어 명령 간의 우선순위를 반영하거나 지연 허용 범위를 조절하는 데 제한이 있다. 이러한 점은 실시간성과 우선순위 제어가 중요한 자율주행 시스템에서 큰 제약으로 작용할 수 있다.

반면, ROS2는 DDS의 표준 통신 프로토콜인 RTPS(Real-Time Publish-Subscribe)를 기반으로 하여, 다양한 QoS 정책을 세밀하게 설정할 수 있다[2][3]. DDS는 Reliability(신뢰성), Durability(지속성), Deadline(데드라인), History(히스토리) 등 다양한 QoS 항목을 제공하여, 애플리케이션의 요구 사항에 맞춰 통신 조건을 정밀하게 조정할 수 있다. 예를 들어, 자율주행 시스템에서 차량 제어 명령은 반드시 손실 없이 정확히 전달되어야 하므로, '높은 신뢰성 + 짧은 데드라인' QoS 설정이 가능하다. 반면, 일부 센서 데이터 스트림은 네트워크 상태에 따라 샘플 손실을 허용할 수 있도록 '베스트 에포트(Best Effort)' 방식으로 전송할 수 있다.

이러한 QoS 설정의 유연성은 자율주행 시스템과 같은 복잡한 환경에서 통신 요구 사항이 다양한 경우 강력한 확장성을 발휘한다. 예를 들어, 자율주행 차량에서는 다양한 센서들이 실시간으로 데이터를 송수신하는데, 각 데이터의 중요도와 긴급성에 맞춰 QoS를 조정함으로써 통신의 효율성을 극대화하고, 시스템의 안정성을 보장할 수 있다. 또한, ROS2의 QoS 설정은 시스템의 요구 사항에 맞는 최적화된 통신을 가능하게 하여, 통신 지연을 최소화하고, 데이터 손실을 방지하는 데 중요한 역할을 한다.

2.1.3 실시간성과 성능

ROS1은 일반적인 리눅스 환경에서 실행되며, 실시간 스케줄링과 디터미니즘을 보장하는데 어려움이 있다. 노드 간 통신 시 불필요한 데이터 복사 및 컨텍스트 전환 오버헤드가 발생하며, 시스템 부하가 증가할수록 메시지 처리 지연이 발생할 수 있다[3]. 특히 실시간성이 중요한 제어 루프에서 이러한 문제는 시스템의 정확한 동작을 방해하고, 제어 시스템의 신뢰성에 큰 영향을 미칠 수 있다.

ROS2는 실시간 임베디드 시스템의 요구를 반영하여 설계되었으며, RTOS(Real-Time Operating System)와의 호환성을 확보하였다. DDS 미들웨어는 항공, 산업 자동화 등 고신뢰성 시스템에 널리 활용되어 왔으며, 이를 바탕으로 ROS2는 낮은 레이턴시와 예측 가능한 응답 지연을 실현할 수 있다[2][3]. 또한, ROS2는 intra-process communication 기능을 활용하여 퍼블리셔와 서브스크라이버 간의 데이터 복사를 제거함으로써 성능을 최적화하고, 시스템의 처리 속도를 개선할 수 있다. 추가적으로, Lifecycle Management 기능을 통해 노드의 상태를 체계적으로 관리함으로써 시스템 초기화 시간을 단축시키고, 자원의 효율적 활용을 가능하게 한다. 이러한 특성들은 자율주행 차량처럼 실시간 제어가 중요한 시스템에서 성능을 최적화하는 데 중요한 역할을 한다.

2.1.4 멀티플랫폼 및 보안

ROS1은 주로 Ubuntu Linux에서만 안정적으로 지원되며, Windows나 기타 운영체제에서의 지원은 제한적이다. 보안 측면에서도 기본적으로 암호화, 인증, 접근 제어 기능이 제공되지 않아, 네트워크 보안에 취약한 구조를 가지고 있다[3]. SROS1이라는 실험적인 보안 계층이 존재하지만, 이는 안정성이나 적용 범위에 한계가 있어 실제 운영 환경에서의 활용에 제약이 있었다.

반면, ROS2는 초기 설계 단계부터 멀티플랫폼 지원을 고려하여, Linux 뿐만 아니라 Windows, macOS, RTOS 등 다양한 운영체제에서 실행할 수 있다. 이러한 특성 덕분에 임베디드 시스템이나 소형 보드에서도 유연하게 배포가 가능하다. 또한, ROS2는 DDS 보안 스펙에 기반한 SROS2를 통해 메시지 암호화, 노드 인증, 권한 제어 등의 보안 기능을 제공한다. 이를 통해 자율주행 차량과 같이 보안이 중요한 시스템에서 안전하고 신뢰할 수 있는 환경을 구축할 수 있다[2][6]. 특히 SROS2는 ROS2의 보안 기능을 강화하여, 민감한 데이터와 명령들이 안전하게 전달될 수 있도록 보장하며, 보안성이 중요한 산업 및 자율주행 시스템에 적합한 솔루션을 제공한다.

Table. 1은 ROS1과 ROS2의 주요 구조적 차이점을 요약 비교한 것이다.

항목	ROS1 (1세대 ROS)	ROS2 (차세대 ROS)
아키텍처	ROS 마스터 노드 필요 (중앙 집중식)	DDS 기반 분산 아키텍처 (마스터 노드 없음)
통신 프로토콜	XML-RPC 기반 TCPROS/UDPROS(전용 프로토콜)	DDS/RTPS (표준 미들웨어 프로토콜 활용)
실시간 지원	비실시간 (일반 Linux 커널, 실시간 보장 어려움)	실시간성 고려 (RTOS 지원, DDS QoS 통한 튜닝)
플랫폼 지원	주로 Ubuntu Linux	Linux, Windows, Mac, RTOS 등 멀티플랫폼
보안	기본 보안 미제공 (SROS1 실험적)	DDS 보안 지원 (SROS2 공식 지원)
기타 특징	Nodelet으로 프로세스 내 통신 일부 지원	Lifecycle 등 노드 관리 기능, 다중 DDS 벤더 지원

Table. 1. Comparison of Key Features between ROS1 and ROS2

2.2 자율주행 시스템 적용 관점에서의 비교 분석

ROS의 구조적 특성은 자율주행 시스템의 설계와 운영에 중요한 영향을 미친다. 자율주행 차량은 카메라, 라이다(LiDAR), 레이더, IMU 등 다양한 센서로부터 초당 수백 MB 이상의 데이터를 수집하며, 이를 실시간으로 분석하여 경로 계획 및 제어에 반영해야 한다. 이 과정에서 통신 지연, 데이터 손실, 시스템 다운은 곧 안전 문제로 이어질 수 있기 때문에, 자율주행 시스템에서는 신뢰성과 실시간성이 특히 중요한 요소로 작용한다.

ROS1 기반 자율주행 플랫폼인 Autoware.AI는 연구 및 시제품 개발에 적합한 환경을 제공했으나, 시스템 확장 시 발생하는 여러 문제에 직면했다. 특히 노드 수가 증가하면서 마스터 노드의 과부하, 센서 데이터 동기화 지연, 통신 병목 현상 등으로 인한 성능 저하가 문제가 되었다[5]. 예를 들어, ROS1에서는 토픽 충돌이나 노드 재연결 실패 등이 종종 발생하며, 초기화 과정에서 전체 시스템의 지연을 초래하기도 한다.

이러한 한계를 해결하기 위해 Autoware 프로젝트는 ROS2 기반의 차세대 플랫폼인 Autoware.Auto를 개발하였다. ROS2는 DDS 기반의 분산 통신 구조를 채택하여, 실시간 분산 통신 환경을 구축하고, 노드 간 의존성을 줄여 모듈화를 강화하였다[5]. 또한, ROS2에서는 QoS 설정을 통해 센서 종류별로 데이터 우선순위를 조절함으로써, 차량 제어에 필수적인 핵심 정보의 지연 및 손실을 최소화할 수 있었다. 이로 인해 자율주행 시스템에서 발생할 수 있는 위험 요소를 줄이며, 실시간 처리 성능을 향상시

켰다.

또한, 미국의 Apex.ai는 ROS2를 기반으로 한 상용 자율주행 소프트웨어 프레임워크인 Apex.OS를 개발하고, ISO 26262 ASIL-D 등급의 기능 안전 인증을 목표로 하여 ROS2의 안정성과 실시간성을 산업적으로 입증하고 있다[4]. Apex.OS는 ROS2 API와 완벽하게 호환되며, 하드 실시간성 보장을 통해 고안정성 자율주행 플랫폼을 구축하고 있다. 이 시스템은 이미 일부 자동차 제조사에 채택되어 실제 자율주행 시스템에서 활용되고 있다.

이러한 사례들은 ROS2가 ROS1의 구조적 한계를 극복하고, 자율주행 시스템에 요구되는 성능, 안정성, 보안성을 충족할 수 있는 기술적 기반을 제공하고 있음을 잘 보여준다. 특히 DDS 기반의 QoS 설정, 분산 discovery 메커니즘, 암호화 통신 등의 기능은 실질적인 시스템 구현에서 중요한 역할을 하며, ROS2의 확산과 산업 적용을 가속화하는 데 중요한 요소로 작용할 것이다[2][6].

Ⅲ. 결론

본 논문에서는 ROS1과 ROS2의 구조적 차이를 비교하고, 자율주행 시스템 적용 관점에서 두 프레임워크의 장단점을 심도 있게 분석하였다. ROS1은 중앙 마스터 노드에 의존하는 구조로 인해 실시간성 부족, 단일 장애점 발생 가능성, 보안 미지원 등의 문제를 가지고 있으며, 이러한 문제들은 안전성과 확장성이 중요한 자율주행 시스템에 적용하는 데 큰 제약을 주었다.

반면, ROS2는 DDS 기반의 분산 아키텍처를 채택하여 노드 간 통신 구조를 근본적으로 개선하고, QoS 조절, 실시간성 보장, 멀티플랫폼 지원, 보안 강화 등 다양한 측면에서 기술적 진보를 이루었다. 특히 Autoware.Auto와 Apex.OS와 같은 사례들은 ROS2가 자율주행 시스템에서 요구되는 실시간성, 안정성, 기능 안전성 등을 충족시키며 실제로 활용되고 있음을 증명한다. ROS2는 자율주행 분야뿐만 아니라 다양한 로봇 시스템에 실질적인 기술적 기반을 제공하고 있으며, 이는 ROS1이 해결하지 못한 문제들을 효과적으로 개선한 결과이다.

다만, ROS2는 아직 완전한 실시간성 확보나 DDS 설정의 복잡성 등의 문제를 안고 있으며, 이러한 과제를 해결하기 위한 추가 연구가 필요하다. 그러나 ROS2는 ROS1의 한계를 극복하며 로보틱스 산업, 특히 자율주행 시스템의 요구를 충족하는 방향으로 발전하고 있다. 향후 ROS2의 성능 최적화, 사용자 친화적인 개발 환경 개선, 그리고 ROS1에서 ROS2로의 전환을 위한 마이그레이션 도구 및 전략 개발이 중요한 과제가 될 것이다. 본 연구는 ROS를 활용하는 로봇 개발자 및 자율주행 시스템 설계자들에게 실질적인 기술적 통찰을 제공하며, ROS2의 도입 및 적용 전략 수립에 중요한 참고자료가 될 것으로 기대된다. 또한, ROS2의 지속적인 발전과 보급이 로봇 기술과 자율주행 분야에서 중요한 혁신을 이끌어갈 것으로 전망된다.

ACKNOWLEDGMENT

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(00235509, ICT융합 공공 서비스● 인프라의 암호화 사이버위협에 대한 네트워크 행위기반 보안관제 기술 개발)과 2023년도 정부(산업통상자원부)의 재원으로 한국산업기술진흥원의 지원을 받아 수행된 연구임 (P0024177, 2023년 지역혁신클러스터육성)의 지원을 받아 수행된 연구임.

참 고 문 헌

[1] ROS.org, “Robot Operating System,” https://www.ros.org

[2] OMG, “Data Distribution Service (DDS),” https://www.dds-foundation.org

[3] M. Macenski et al., “Robot Operating System 2: Design, Architecture, and Uses in the Wild,” Science Robotics, 2022.

[4] Apex.AI, “ROS 2 on Self-Driving Cars,” Apex.ai Blog, 2020. [Online]. Available: https://www.apex.ai/post/ros-2-on-self-driving-cars

[5] K. Tokuda, et al., “Autoware on ROS 2: Design, Architecture, and Lessons Learned,” Autoware Foundation, 2021.

[6] OMG, “DDS Security Specification,” Version 1.1, Object Management Group (OMG), 2020.