Journal of Korea Robotics Society (2021) 16(4):327-335
https://doi.org/10.7746/jkros.2021.16.4.327 ISSN: 1975-6291 / eISSN: 2287-3961

클라우드 연동을 위한 ROS2 on Yocto 기반의 Thin Client 로봇 개발

Development of ROS2-on-Yocto-based Thin Client Robot for Cloud Robotics

김 윤 성[1]·이 돈 근[2]·정 성 훈[2]·문 형 일[2]·유 창 승[2]·이 강 영[2]·최 준 열[2]·김 영 재[†]
Yunsung Kim[1], Dongoen Lee[2], Seonghoon Jeong[2], Hyeongil Moon[2],
Changseung Yu[2], Kangyoung Lee[2], Juneyoul Choi[2], Youngjae Kim[†]

Abstract: In this paper, we propose an embedded robot system based on "ROS2 on Yocto" that can support various robots. We developed a lightweight OS based on the Yocto Project as a next-generation robot platform targeting cloud robotics. Yocto Project was adopted for portability and scalability in both software and hardware, and ROS2 was adopted and optimized considering a low specification embedded hardware system. We developed SLAM, navigation, path planning, and motion for the proposed robot system validation. For verification of software packages, we applied it to home cleaning robot and indoor delivery robot that were already commercialized by LG Electronics and verified they can do autonomous driving, obstacle recognition, and avoidance driving. Memory usage and network I/O have been improved by applying the binary launch method based on shell and mmap application as opposed to the conventional Python method. Finally, we verified the possibility of mass production and commercialization of the proposed system through performance evaluation from CPU and memory perspective.

Keywords: Yocto, ROS, ROS2, Cleaning Robot, Delivery Robot, SLAM, Navigation, Autonomous Driving

1. 서 론

Yocto Project는 개발자가 하드웨어 아키텍처와 상관없이 임베디드 리눅스 기반 시스템을 개발할 수 있도록 템플릿, 툴 및 방법을 제공하는 오픈소스 협업 프로젝트로서, 어떠한 임베디드 하드웨어에서도 실행이 되는 맞춤 가능한 리눅스 배포판을 생성하는 것이 목적이다[1]. 여러 오픈 소스의 집합체이며, 하드웨어 독립적인 임베디드 제품설계를 위한 통합 빌드 시스템을 제공한다. 현재 ARM, ARM64, x86, x86-64, PowerPC, MIPS, MIPS64의 7개 processor를 지원하고 있으며, 빌드 환경, 유틸리티 및 개발도구를 제공함으로써 개발자 간 작업 환경의 의존성을 줄여준다.

ROS는 Robot Operating System의 약자로, 2007년에 등장한 오픈 소스 기반의 로봇 메타운영체제로서, 이기종 하드웨어 간의 데이터 송수신, 스케줄링, 에러 처리 등의 소프트웨어 프레임워크를 제공한다. 또한 각종 드라이버, 라이브러리, 개발도구를 공개 패키지로 제공하여 로보틱스 분야의 생태계를 구축하고 있다[2,3]. ABI Research에 따르면 2024년에는 전체 상용로봇의 55%가 ROS를 적용할 것으로 예측하였으며[4], 최근에는 iRobot, Toyota, Sony, Amazon, Microsoft, Intel, Qualcomm 등 로봇제조, 클라우드, OS 및 칩셋 업체들의 ROS2 적용 및 지원이 지속 확대되고 있다.

ROS2는 2017년 12월에 공개되었으며, ROS의 장점은 그대로 수용하면서, ROS가 가진 단점을 보완하였다[5].

ROS2는 미들웨어 통신 프레임워크로서 실시간 및 임베디드 시스템 지원을 위해 분산환경을 위한 데이터 중심의 통신 표준인 DDS (Data Distribution Service)를 채택하였다[6]. 멀티

Received : Jul. 31. 2021; Revised : Sep. 12. 2021; Accepted : Oct. 5. 2021
※ This project was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (No. 2020-0-00857)
1. Chief Researcher, LG Electronics, Seoul, Korea (yunsung.kim@lge.com)
2. Senior Researcher, LG Electronics, Seoul, Korea (dungeon.lee, seonghoon122.jeong, hyeongil.moon, changseung.yu, jude.lee, juneyoul.choi@lge.com)
† Research Fellow, Corresponding author: LG Electronics, Seoul, Korea (yjae.kiml@lge.com)

로봇 제어가 가능하고, 실시간 시스템을 지원하며, OS가 탑재되지 않은 마이크로 컨트롤러와 같은 작은 플랫폼과도 연동된다.

본 논문에서는 다양한 form factor 로봇을 지원할 수 있는 ROS2 on Yocto를 탑재한 임베디드 로봇시스템을 제안하였다. 제안된 시스템은 Yocto Project 적용으로 하드웨어 확장성, 소프트웨어 이식성을 확보하였으며, 저사양의 임베디드 하드웨어 시스템을 고려하여 ROS2를 최적화하여 탑재하였다.

제안한 로봇시스템 검증을 위해, LG전자에서 상용화한 가정용 청소로봇과 실내배송로봇에 적용하여 자율주행, 장애물 인식 및 회피 주행 검증을 진행하였다. 마지막으로 shell 기반의 binary launch 방식, mmap 적용을 통해 memory 사용량, network I/O를 개선하였으며, memory, CPU 관점에서의 성능평가를 통해 제안한 시스템의 양산 및 상용화 가능성을 검증하였다.

2. Ubuntu와 Yocto의 비교

2.1 Ubuntu와 Yocto

리눅스는 전체 시스템을 구성하는 각 개별 파트가 아닌 OS 커널을 의미하며, 리눅스 배포판은 부트로더, 리눅스 커널, 루트 파일시스템으로 구성된다. Yocto는 리눅스 배포판을 만들 수 있는 툴이며, 리눅스 배포판 자체를 의미하지 않는다. 반면 Ubuntu는 Debian을 기반으로 오픈 소스 소프트웨어로 구성된 리눅스 배포판이며, 리눅스 배포판을 만들 수 있는 툴이 아니다. 즉, Yocto와 Ubuntu는 성격자체가 다르기 때문에 임베디드 기반의 리눅스 시스템을 개발하고 유지보수 하는 데 있어서, 각각의 장단점을 가지고 있다.

2.2 Ubuntu와 Yocto의 장단점

Ubuntu는 리눅스 배포판을 사용하여, 시스템을 구성하기 때문에 빠른 프로토타이핑, Proof of Concept에 장점을 가지고 있다. 그러나 Ubuntu를 지원하는 하드웨어가 있어야 하고, 많은 메모리가 필요하다. 또한 하드웨어 이식성, 이력관리를 통한 유지보수가 어려운 단점을 가지고 있다. 반면 Yocto는 다양한 하드웨어를 지원하는 맞춤형 임베디드 환경에 적합하며, 하드웨어 이식성, 유지보수, 이력관리에 장점이 있다. Ubuntu와 Yocto의 장단점에 대해서는 [Table 1]에 상세히 기술하였다[7].

결론적으로 Ubuntu는 빠른 프로토타이핑, Proof of Concept에 장점을 가지고 있다. 그러나 다양한 form factor의 로봇지원, 하드웨어 이식성, 부팅시간, 메모리, 라이센스 관리 및 유지보수 측면을 고려하여, bottom-up 방식의 맞춤형 설계로 OS의 경량화가 필요하며, 클라우드와의 연동을 통해 로봇의 자원을 효율화하고, 성능을 극대화할 수 있는 ROS2 on Yocto 기반의 Thin Client 로봇 개발이 요구된다.

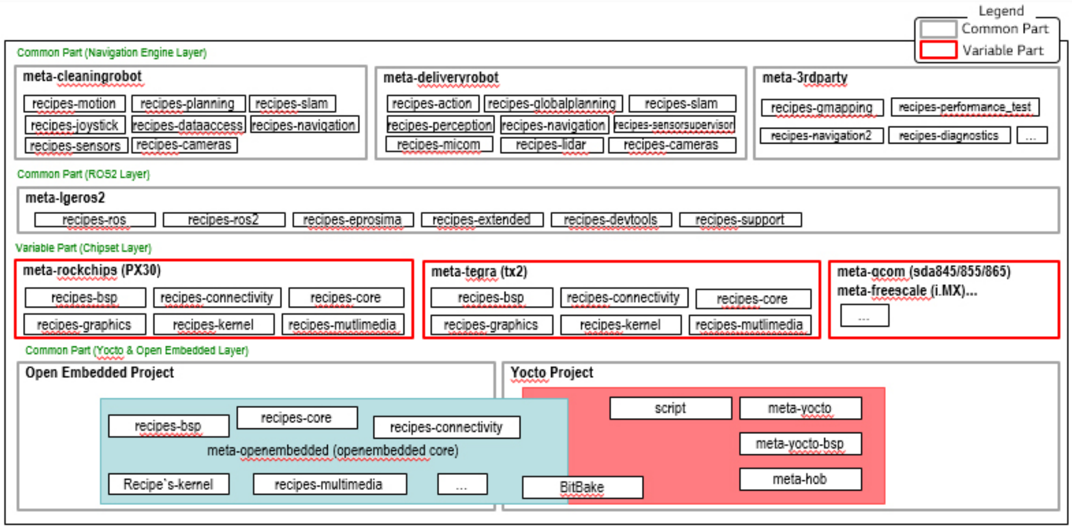
3. Software Architecture

본 논문에서 제안한 ROS2 on Yocto 기반 Thin Client 로봇의 software architecture는 [Fig. 1]과 같으며, meta-lge, meta-3rd party, meta-lgeros2, meta-chipset, Yocto Project (open embedded 포함)의 총 5개 layer로 구성된다.

Yocto Project layer는 Yocto Project에서 제공되는 layer 이며, meta-3rdparty layer는 gmapping, navigation2, diagnostics, performance test와 같은 ROS2의 오픈 패키지를 탑재할 수 있는 layer이다. Meta-chipset layer는 meta-rockchips, meta-tegra와 같이 칩셋 업체에서 제공되는 layer로서, 본 논문에서는 청소로봇으로는 Rockchip사의 PX30K, 실내배송로봇은 Nvidia사의 TX2를 적용하였으며, 로봇들의 부품 구동을 위해 device driver를 구현하였다. Meta-lgeros2는 ROS2 탑재를 위한 layer로서, 임베디드 환경(CPU, memory)을 고려하여 memory 및 network I/O 최적화를 진행하였다. Meta-lge는 LG전자의 주행 engine (ROS2 node의 집합) 탑재를 위한 layer로서 가정용 청

[Table 1] Comparison of Yocto and Ubuntu Debian

Issue	Yocto	Ubuntu
Compile	Fully cross compile	Target board or cross compile
Configuration, Customization	Ease in hardware portability and maintenance When configuring the same platform based on x86 and arm64, only bootloader and kernel can be changed	Difficulty in hardware portability and maintenance When configuring the same platform based on x86 and arm64, needs to be developed separately
Reproducibility	Fully automated build system can be built	Manually select a package to create an image Create with a script tool made by the developer
Patching	A patch for the target chipset is required	Impossible to predict and debug the impact of the patch on the system
License	Automatically included when creating a distro	Manual extraction required
Boot time	Possible to optimize boot-time with bottom-up development method	Increase of boot-time by including unnecessary services
Memory	Minimization and optimization possible with the required package configuration	Growing with unnecessary packages and patches
Learning curve	Difficult to learn, so developers shun it.	Develop like using a Linux desktop



[Fig. 1] Software Architecture for Thin Client Robots

소로봇 및 상업용 실내배송로봇을 위한 navigation engine을 모두 탑재하였다. 또한 각 layer들을 공통부(Yocto, ROS2, navigation layer)와 변동부(chipset layer)로 분리하여, 공통부의 신속한 소프트웨어 버전 업데이트가 가능하게 설계하였으며, 공통부와 변동부의 조합을 통해 로봇 제품별, 칩셋별 다양한 form factor를 지원하는 소프트웨어 버전을 손쉽게 확장할 수 있다.

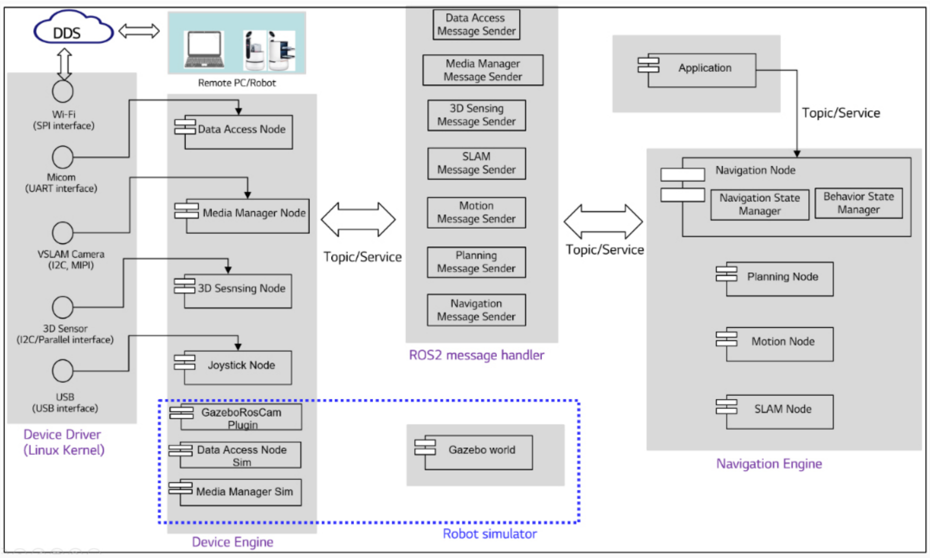
4. Functional Block Diagram

4.1 가정용 청소로봇

ROS2 on Yocto 기반의 가정용 청소로봇을 위한 functional block diagram은 [Fig. 2]와 같으며[8], Rockchip PX30K board의 device 제어를 위한 BSP device driver, device를 운용하기 위한 device engine, 자율주행을 위한 navigation engine, Gazebo 기반의 simulator 구동을 위한 robot simulator 및 ROS2 message handler로 구성된다. 모든 engine들은 ROS2 node로 구현되었으며, ROS2 message handler는 ROS2의 node간 통신을 위한 topic, service의 interface를 API로 제공한다. 각 node의 역할은 아래 [Table 2]에 상세히 기술하였다.

4.2 상업용 실내배송로봇

ROS2 on Yocto 기반의 상업용 실내배송로봇의 functional block diagram은 [Fig. 3]과 같으며, Nvidia TX2 board의 device 제어를 위한 BSP device driver, device를 운용하기 위한 device



[Fig. 2] Functional Block Diagram for Cleaning Robots

[Table 2] Nodes and Responsibility of Cleaning Robots

Node	Responsibility
Media manager	Operation of front-top camera module
3D sensing	Operation of 3D sensing module and detection obstacles
Data access	Processing micom's sensor data and control the motor
Joystick	Manual driving of the robot via Joystick
Navigation	Request operation related to driving such as cleaning start, stop, homing, docking, etc..
Motion	Processing the current robot motion after receiving obstacle information and robot position/state information
SLAM	Recognizing the robot position o and making a map
Planning	Planning a map-based route and judging the completion of cleaning
Application	Responsible for interface with navigation for driving
Data access sim	The role of data access node for simulator
GazeboRosCam	Providing raw image data on the simulator
Media manager sim	The role of media manager for simulator

engine, 자율주행을 위한 navigation engine, 장애물 인식 및 docking/undocking을 위한 perception engine, Unity 기반의 simulator 구동을 위한 robot simulator 및 ROS2 message handler로 구성된다. 청소로봇과 동일하게 모든 engine들은 ROS2 node로 구현되었으며, ROS2 message handler는 ROS2의 node 간 통신을 위한 topic, service의 interface를 API로 제공한다. 마지막으로 cloud bridge는 클라우드 연동을 위한 bridge node로서 ZeroMQ를 통해 로봇과 서버를 연결한다. 각 node의 역할은 아래 [Table 3]에 상세히 기술하였다.

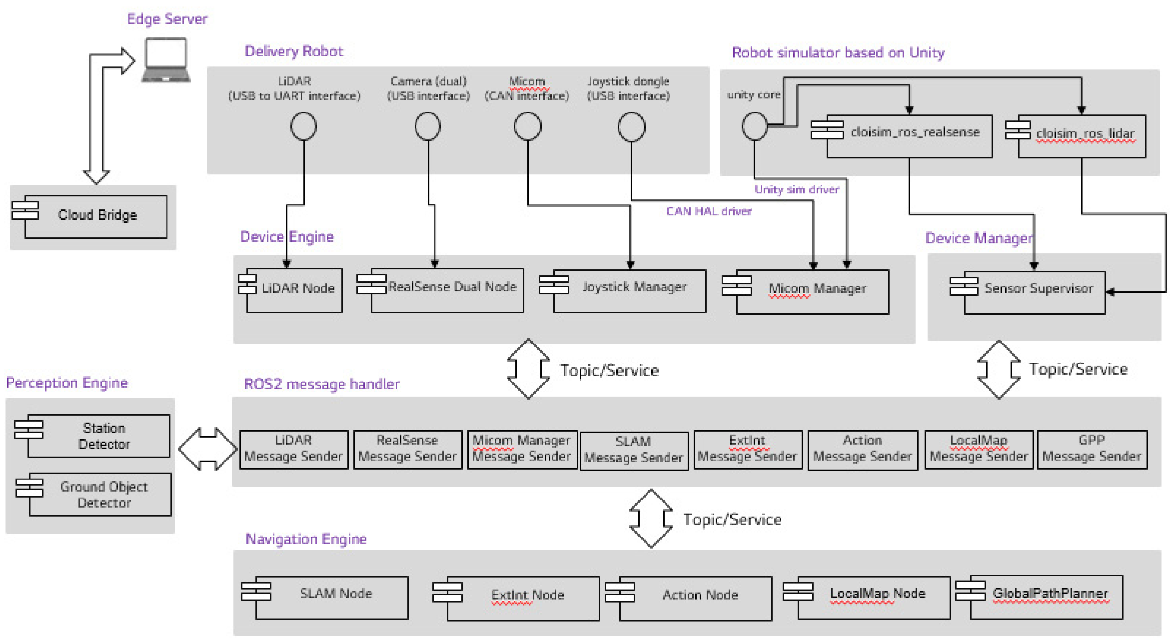
[Table 3] Nodes and Responsibility of Indoor Delivery Robots

Node	Responsibility
Realsense dual	Operation of front and bottom camera modules
LiDAR	Operation of LiDAR scan data processing
Micom manager	Processing micom's sensor data and controling the motor
Joystick	Manual driving of the robot via Joystick
Sensor supervisor	Delivery to the required node after collecting sensor and LiDAR data
ExtInt/Action	Performing driving functions such as destination driving, docking, and kidnap recovery
Local map	Creating and saving a map for driving
SLAM	Recognizing the robot position and making a map
Global path plan	Planning the global path based on map
Ground object detection	Recognizing obstacles up and down based on the road surface with the depth camera in front and behind.
Station detector	Creating a path using the position/angle of the charging station and the position/angle of the robot and perform docking
Cloisim_ros_realsense	Providing raw image data on the simulator
Cloisim_ros_lidar	Providing LiDAR data on the simulator

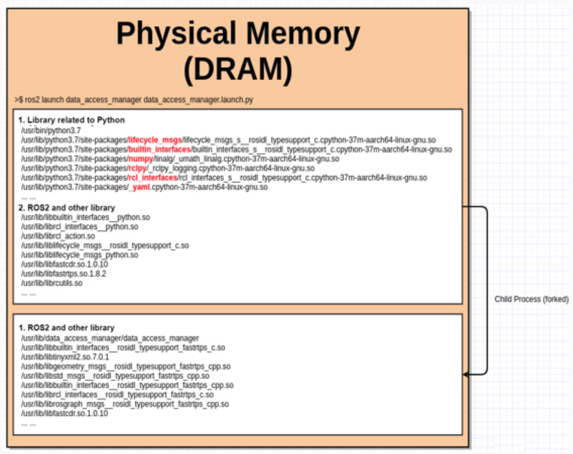
5. 성능개선

5.1 Memory 사용량 개선

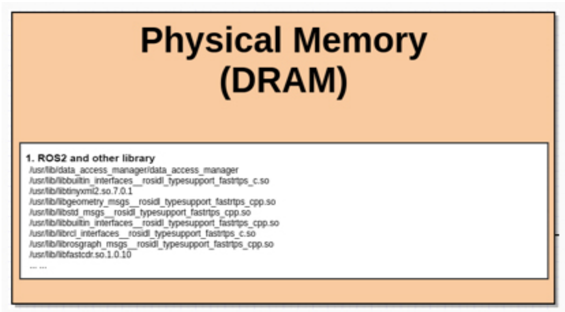
임베디드 환경에서의 RAM 사용량 개선을 위해 본 연구에서는 binary 기반 ROS2 node의 launch 방식을 적용하여, Python 기반 방식 대비 memory 사용량을 개선하였다[9]. [Fig. 4]는 ROS2



[Fig. 3] Functional Block Diagram for Delivery Robots



[Fig. 4] ROS2 launch method based on Python



[Fig. 5] ROS2 launch method based on Python

에서 제공하는 Python 기반 node 실행방식에서의 physical RAM 사용량을 보여준다. 그림과 같이 Python 관련 library, ROS2의 library process와 함께 folk를 통해 child process가 추가로 생성되어 실행되는 것을 확인할 수 있다. 즉, 2개의 process가 실행되 어 RAM 점유율이 증가되는 문제점을 가지고 있다.

이에 대한 개선을 위해 본 논문에서는 binary 기반의 ROS2 node 실행방식을 적용하여 RAM 사용량을 개선하였다. [Fig. 5]와 같이 shell 기반의 binary 실행방식 적용을 통해, RAM상에 상주하는 Python code들을 모두 제거하여 저용량의 RAM에서도 ROS2가 동작할 수 있는 가능성을 검증하였다.

아래는 Python, binary node 실행 방식 및 이에 대한 예로 motion node 실행 방법에 대해 각각 설명하였다.

Python 기반 실행방식

ros2 launch [NODE_NAME] [NODE_NAME].launch.py

ros2 launch motion motion.launch.py

Binary (Shell) 기반 실행방식

[BIN_PATH] __node:=[NODE_NAME] __ns:=[NAME_SPACE] __params:=[YAML_PATH]

/usr/lib/motion/motion __node:=motion __ns:=/manager_ns __params:=/usr/share/motion/launch/motion.yaml

[Table 4] RAM usage of python and binary launch method

	Total	Used	Free
Memory (Python)	471,188 KB	436,852 KB	9,960 KB
Memory (Binary)	471,188 KB	257,932 KB	149,632 KB

[Table 4]는 각각 Python 기반의 실행방식, binary 기반의 실행방식을 적용하여, 청소로봇으로 zigzag 주행검증 시의 RAM 점유율을 보여주고 있다.

Python 기반 실행 방식의 경우, 7개의 node (data access, navigation, motion, SLAM, planning, media, 3D sensing) 실행 시, free RAM이 9,960 KB (9.7 MB) 수준으로 측정되었다. 반면 Binary 기반 실행방식 적용 시에는 free RAM이 149,632 KB (146 MB) 수준으로 Python 실행방식 대비, 약 140 MB 이상의 RAM 사용량을 절감하였으며, 충분한 free RAM 용량을 확보하였다고 판단할 수 있다.

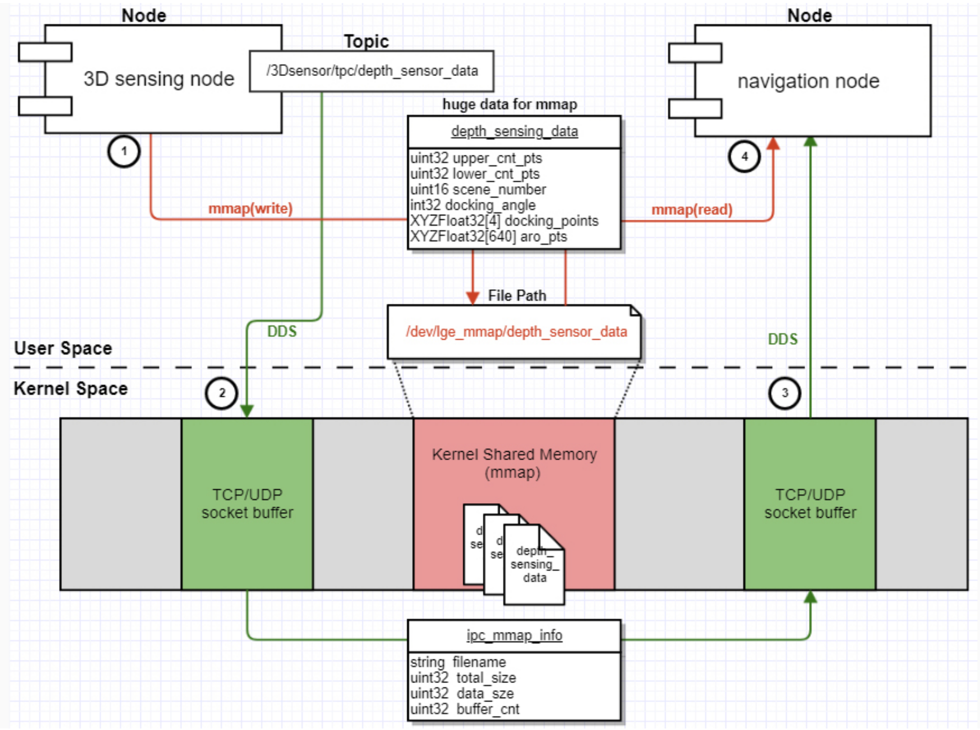
5.2 Network I/O 개선

ROS2의 node간 통신은 bus가 아닌, TCP/UDP를 통한 network 통신으로 이루어진다. 1 KB 이상의 huge data topic의 빈번한 통신은 network 부하 및 message loss를 유발할 수 있으며, 이에 대한 처리를 위해 IPC (Inter-Process Communication) 적용이 필요하다. 본 논문에서는 huge data 처리를 위한 IPC를 구현함으로써 network I/O에 대한 개선을 진행하였다. 빠른 속도처리를 위해서 일반적인 파일 R/W의 storage를 사용하지 않고, DRAM에만 임시 저장하는 mmap 방식을 적용하였다.

huge data topic으로는 3D sensing, image data를 선정하였으며, [Fig. 6]은 3D sensing에 적용한 mmap의 동작방식을 나타내고 있다. Mmap의 정보(ipc_mmap_info)는 기존과 같이 DDS 통신으로 data가 처리되며, 3D sensing data인 depth_sensing_data 구조체는 mmap을 이용한 shared memory방식으로 huge data를 전달하게 된다. 이때 동기화 문제를 최소화 하기 위해, data를 mmap으로 생성한 file path에 write하고, physical memory에 바로 sync 하도록 구현하였다.

Mmap 적용 전후의 개선효과를 확인하기 위해 free memory와 network I/O 성능검증을 진행하였다. free memory 검증에는 'free' shell command를, network I/O 검증에는 'cat /proc/net/dev' shell command로 사용하였으며, packet에 대한 transmit, receive의 bytes값으로 계산하였다.

[Table 5]는 mmap 전용 전후의 free memory, network I/O의 성능검증 결과를 보여주고 있다. Free memory는 mmap 적용 전후의 유의차가 없으나, network I/O는 mmap 적용으로 80 KB~100 KB의 개선효과가 확인되었다.



[Fig. 6] The operation flow of IPC (mmap)

[Table 5] Free RAM and network I/O before and after mmap

Patch	Test Items	#1	#2	#3	#4	#5
Before applying mmap	Free RAM (MB)	69	60	66	65	63
	Network I/O (KB/s)	355	375	354	360	357
After applying mmap	Free RAM (MB)	65	68	65	65	62
	Network I/O (KB/s)	275	274	274	275	275

6. 자율주행 검증 및 성능평가

6.1 소프트웨어 및 하드웨어 사양

ROS2 on Yocto 적용에 따른 성능개선 및 memory, network I/O 개선효과를 실증하기 위해 LG전자 청소로봇 및 실내배송로봇을 활용하였다. 적용한 청소로봇 및 실내배송로봇의 Yocto, ROS2 및 kernel의 소프트웨어 버전은 [Table 6]와 같다.

실기 검증을 위해 제작한 청소로봇 및 실내배송로봇의 하드웨어 사양은 [Table 7], [Table 8]과 같다.

6.2 청소로봇의 자율주행 검증 및 성능평가

개선된 사항을 청소로봇 실기에 적용하여, [Fig. 7]과 같이 zigzag 주행기반으로 자율주행 및 회피주행을 검증하였다[10].

[Table 6] Software version of the cleaning and delivery robots

	Software version
ROS2	Foxy Fitzroy
Yocto	Code name : Dunfell (version 3.1)
Kernel	Linux version 4.4 (Cleaning Robot) Linux version 4.9 (Delivery Robot)

[Table 7] Hardware specification of the cleaning robots

	Specification
AP	Rockchip PX30K
CPU	Cortex-A35 X 4 (64bit-ARMv8 @1.296 GHz)
Flash Memory	1GB eMMC (Code size limited to 512 MB)
RAM	DDR3 512 MB, 800 Hz
Micom	STM32F(Coretex-M3)

[Table 8] Hardware specification of the delivery robots

	Specification
AP	NVIDIA Jetson TX2
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM® Cortex®-A57 MPCore
Flash Memory	32 GB eMMC 5.1
RAM	8 GB 128-bit LPDDR4 1866 MHx - 59.7 GB/s
Micom	TI Microcontroller(Coretex-R4F)

[Fig. 7] Autonomous driving verification of cleaning robots

[Table 9] The performance verification result of cleaning robots

Verification Items	Verification results
CPU load	64%
Memory usage	374 MB (Based on 512 MB RAM)

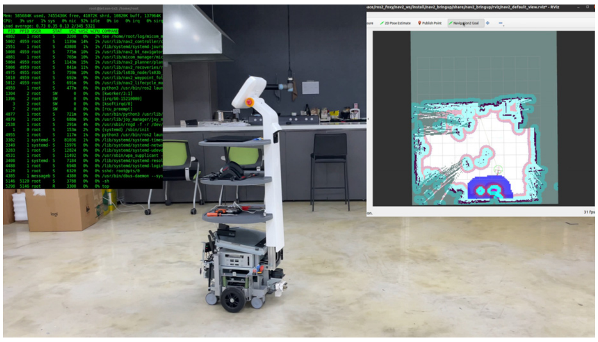
로봇은 전방에 위치한 3D sensing 하드웨어 모듈에 의해 장애물을 인식한다. 해당 모듈은 상방 laser, 하방 laser 및 camera로 구성되며, camera를 통해 적외선을 detection하여 장애물을 감지한다. 해당 모듈의 운용을 위한 3D sensing ROS2 node는 IR laser line을 상방과 하방에 주사하여 camera에 맺히는 영상으로 추출한 2차원의 IR line 좌표를 3차원의 실제 좌표로 변환하여 거리데이터를 추출하여 장애물과의 거리를 산출하여 navigation node로 전달한다. [Fig. 7]은 zigzag 주행 중 3D sensing에 의한 장애물 감지 및 이에 따른 회피주행 동작을 보여주고 있다.

또한 로봇의 현재 주행상태를 파악하기 위해 [Fig. 7]의 좌우측 상단과 같이 Rviz를 활용한 visualization tool을 구현하였다. 로봇은 주행 중 corrected pose, SLAM node 좌표, coverage map을 publish 하는데 동일 network domain내의 proxy server (remote PC)를 활용하여 해당정보를 subscribe후 Rviz로 출력할 수 있도록 구현하였다.

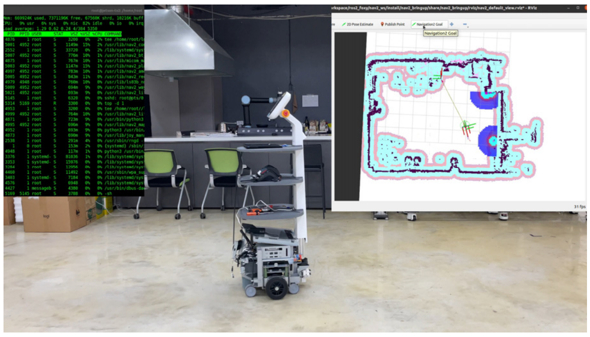
청소시작, docking, 청소재시작, kidnap recovery, docking의 시나리오로 10분 동안 반복주행을 하면서, 1초주기로 CPU사용량, memory 사용량을 측정한 결과는 [Table 9]과 같다. 측정결과 CPU 사용량은 64%, RAM 사용량은 378 MB로 저사양의 임베디드 환경에서도 추가적인 기능들을 탑재할 수 있는 성능을 확보하였다고 판단할 수 있다.

6.3 배송로봇의 Nav2 자율주행 검증 및 성능평가

Nav2 Project는 ROS2에서 공개한 Navigation Stack으로서[11], 모바일 로봇이 A지점에서 B지점으로 안전하게 이동할 수 있는 방법을 제공하는 ROS2 node들의 집합이다. Navigation Stack의 목적은 로봇의 현재위치, 목표위치, 경로계획을 완료하고, odometry 데이터, 센서 데이터를 입력 받아, 모터의 속도를 계산하고, 장애물을 회피하며, 길을 잃지 않게 목적지로 이동하는 것이다. 본 논문에서는 [Fig. 8], [Fig. 9]과 같이 실내배송로봇에 Nav2를 적용하여, ROS2 on Ubuntu와 ROS2 on Yocto 기반의 자율주행을 검증하고, 각각의 성능평가를 진행하였다. [Fig. 8]은 SLAM toolbox[12]를 이용한 mapping, [Fig. 9]는 Nav2를 이용한 목적지 자율주행을 나타내고 있으며, 각 그림의 좌측상단은 리눅스에서 지원하는 ‘htop’ 명령어에 따른 memory 사용량 및 CPU 점유율을, 우측상단은 Rviz를 활용한 현재 로봇의 위치, 맵과 장애물 정보, 주행상태를 보여주고 있다.



[Fig. 8] Mapping of delivery robot using SLAM toolbox



[Fig. 9] Autonomous driving using Nav2

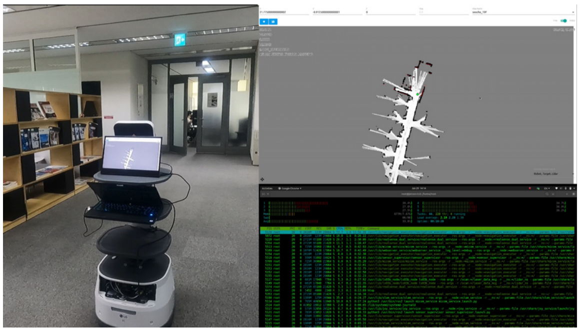
[Table 10] The performance verification result of delivery robots after applying Nav2 open source package

Verification items	Verification results	
	Ubuntu	Yocto
CPU load	46~54%	15%~19%
Memory usage	1.7 GB	600 MB

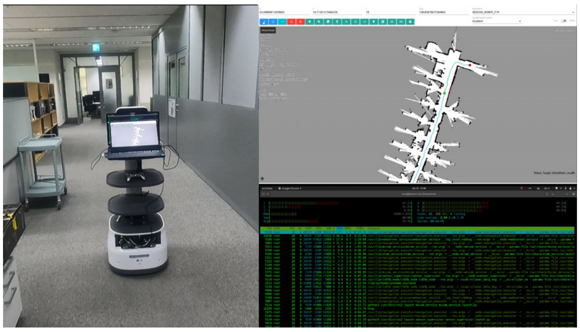
Ubuntu 및 Yocto 기반의 실내배송로봇으로 move to goal(목적지 주행)을 반복하면서, 1초주기로 CPU사용량, memory 사용량을 측정한 결과는 [Table 10]과 같다. 측정결과 Yocto 적용시, Ubuntu 대비 CPU 점유율을 약 30%, memory 사용량은 약 1.1 GB의 성능개선이 확인되었다.

6.4 배송로봇의 자사 Navigation 적용에 따른 자율주행 검증 및 성능평가

본 논문에서는 [Fig. 10], [Fig. 11]과 같이 실내배송로봇에 자사 navigation을 적용하여 mapping 및 목적지 주행을 통해



[Fig. 10] Mapping of delivery robot using LG navigation



[Fig. 11] Autonomous driving using LG navigation

[Table 11] The performance verification result of delivery robots

Verification items	Verification results	
	Ubuntu	Yocto
CPU load	Over 90%	50%~55%
Memory usage	2.4 GB	1.25 GB

자율주행 및 성능을 검증하였다.

로봇은 위치인식을 위해 LiDAR node에서 주기적으로 scan data를 입력받고, micom으로부터 dead reckoning pose (odometry)를 받는다. 로봇이 일정 거리 이동하거나 일정 각도 이상 회전하였을 경우 realsense node에 영상을 요청하여 영상을 수신한다. 전방 영상을 이용하여 특징점을 추출하며 3D point를 생성하고 pose-graph 통하여 최적화를 수행하며 global pose를 보정한다.

로봇은 목적지 주행을 위해 DWA (Dynamic Window Approach) 알고리즘을 기반으로 목적지까지 자율주행을 한다. 현재 위치와 주변 장애물 정보를 고려하여 목적지에 도달하기 위한 경로를 생성하고, 생성된 경로를 기반으로 속도 명령을 생성한다. 주행 간 사람 및 장애물을 만나면 회피하며, 별도의 종료 명령이 없으면 최종 목적지까지 주행을 완료한다.

Ubuntu 및 Yocto 기반의 실내배송로봇으로 move to goal을 반복하면서, 1초주기로 CPU사용량, memory 사용량을 측정한 결과는 [Table 11]과 같다. 측정결과 Yocto 적용 시, Ubuntu 대비 CPU 점유율을 약 35%, memory 사용량은 약 1.15 GB의 성능개선이 확인되었다.

7. 결론 및 향후연구

본 논문에서는 다양한 form factor 로봇을 지원할 수 있는 ROS2 on Yocto를 탑재한 임베디드 로봇시스템을 제안하였다. 제안된 시스템은 Yocto Project 적용으로 하드웨어 확장성, 소프트웨어 이식성을 확보하였다. 그리고 저사양의 임베디드 시스템을 고려하여 ROS2를 최적화하고, 성능 개선을 진행하였다. 또한 제안한 로봇시스템 검증을 위해, LG전자에서 상용화한 가정용 청소로봇과 실내배송로봇에 적용하여 자율주행, 장애물 인식 및 회피 주행 검증 및 성능평가를 진행하였다.

ROS2 on Yocto 적용으로 ROS2 on Ubuntu 대비 CPU 점유율은 30% 이상, RAM 사용량은 약 1 GB 이상의 개선이 확인되었다. 또한 저사양의 임베디드 시스템을 고려하여 가정용 청소로봇으로 추가적인 성능개선을 진행하였으며, 청소로봇의 RAM 사용량은 422 MB 수준(512 MB RAM 기준), CPU 점유율은 평균 64%로 검증되었다. 결론적으로 ROS2 on Yocto 탑재에 따른 LG전자 로봇들의 성능평가 결과, memory 및 CPU 측면에서 충분한 성능을 확보하였다고 평가할 수 있다. LG전자는 제안한 ROS2 on Yocto 기반으로 클라우드와 연동한 다양한 로봇개발을 위한 선행연구를 지속적으로 진행해 나갈 예정이다.

References

[1] *Linux Foundation, Yocto Project Overview*, [Online], https://yoctoproject.org/software-overview/, Accessed: Aug 19, 2019.

[2] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E.Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," *International Conference on Robotics and Automation workshop on open source software*, 2009, [Online], http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[3] *Open Robotics, ROS Wiki*, [online], http://wiki.ros.org/, Accessed: Jan 21, 2019.

[4] *Business Wire*, "The Rise of ROS: Nearly 55% of total commercial robots shipped in 2024 Will Have at Least One Robot Operating System package Installed," [Online], https://www.businesswire.com/news/home/20190516005135/en/The-Rise-of-ROS-Nearly-55-of-total-commercial-robots-shipped-in-2024-Will-Have-at-Least-One-Robot-Operating-System-package-Installed, Accessed: Sep 17, 2019.

[5] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," *International Conference on Embedded Software*, Pittsburgh, USA, pp. 1-10, 2016, DOI: 10.1145/2968478.2968502.

[6] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," *23rd International Conference on Distributed Computing Systems Workshops*, Rhode Island, USA, pp. 200-206, Providence, RI, USA, 2003, DOI: 10.1109/ICDCSW.2003.1203555.

[7] *Prevas Inc*., "*Yocto or Debian for Embedded Systems*," [Online], https://www.prevas.dk/download/18.58aaa49815ce6321a327da/1506087244328/Yocto_Debian_Whitepaper.pdf
[8] Y.-S. Kim, D.-G. Lee, S.-H. Jeong, H.-I. Moon, C.-S. Yu, K.-Y. Lee, and J.-Y. Choi, "Development of an embedded cleaning robot system applied with ROS2 based on Yocto Project," *2020 35th ICROS Annual Conference 2020*, Sokcho, Gangwon, pp. 281-283, 2020, [Online], http://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE09410439.
[9] Y.-S. Kim, D.-G. Lee, S.-H. Jeong, H.-I. Moon, C.-S. Yu, K.-Y. Lee, and J.-Y. Choi, "Development of a Cleaning Robot System Equipped with ROS2 Based on Yocto Project," *Journal of Institute of Control, Robotics and Systems*, vol. 15, no. 1, pp. 54-60, 2021, DOI: 10.5302/J.ICROS.2021.20.0142.
[10] D.-G. Lee, Y.-S. Kim, S.-H. Jeong, H.-I. Moon, C.-S. Yu, K.-Y. Lee, and J.-Y. Choi, "Multi-Robot Control through Map Sharing Based on ROS2," *Proc. of 2021 36th ICROS Annual Conference 2021*, Yeosu, Jeolla, pp. 158-160, 2021, [Online], https://www.dbpia.co.kr/pdf/pdfView.do?nodeId=NODE10609203
[11] *Steve Macenski, NAV2*, [Online], http://navigation.ros.org/, Accessed: Feb 16, 2021.
[12] *Steve Macenski, slam toolbox*, [Online], https://github.com/SteveMacenski/slam_toolbox, Accessed: Feb 16, 2021.

김 윤 성

2000 고려대학교 전기전자전파공학부(학사)
2002 고려대학교 전자공학과(석사)
2004~현재 LG전자 책임연구원

관심분야: Embedded System, Robot Operating System, System Performance & Stability

이 돈 근

2011 성균관대학교 컴퓨터공학과(학사)
2011~현재 LG전자 선임연구원

관심분야: Sensor Device, Robot Operating System

정 성 훈

2011 충남대학교 컴퓨터공학부(학사)
2013 충남대학교 컴퓨터시스템(석사)
2013~현재 LG전자 선임연구원

관심분야: Embedded System, Robot Operating System

문 형 일

2013 부산대학교 컴퓨터공학부(학사)
2014~현재 LG전자 선임연구원

관심분야: Embedded System, Robot Operating System, SLAM

유 창 승

2010 아주대학교 전자공학부(학사)
2013 아주대학교 전자공학부(석사)
2013~현재 LG전자 선임연구원

관심분야: Embedded System, Robot Operating System

이 강 영

2013 인하대학교 정보통신공학(학사)
2013~현재 LG전자 선임연구원

관심분야: Embedded System, Robot Operating System, 로봇제어

최 준 열

2013 홍익대학교 컴퓨터공학부(학사)
2013~현재 LG전자 선임연구원

관심분야: Yocto, Embedded System, System Performance & Stability

김 영 재

1999 서울대학교 전기공학부(공학사)
2003 Stanford Univ. 전기공학(석사)
2007 Stanford Unv. 전기공학(박사)
2009~2017 Apple Inc., 시니어 SW엔지니어
2017~2018 Velodyne LiDAR, 수석엔지니어
2019~현재 LG전자 연구위원

관심분야: Cloud Robotics, Network, Robot Operating System