CiLab
Creative Idea Lab

자율주행 플랫폼 CILAB X3
사용메뉴얼

CiLab Co.,Ltd

본 매뉴얼은 CiLab X3 자율주행 플랫폼 시리즈 ROS 로봇 사용 매뉴얼로 사용 전 자세히 읽어 주세요.

하드웨어의 구성 및 주요 규격부터 ROS 로봇의 작동 환경과 소프트웨어 설명 등 다양한 내용을 통해 유저에게 ROS 로봇의 작동 방법, 운행 원리, 그리고 소프트웨어 프레임워크에 대해 소개합니다.

본 매뉴얼을 통해 유저는 로봇에 대한 전반적인 지식을 획득할 수 있을 것이며 보다 상세한 로봇 동작 방법, 코드 해독 및 개발 지도는 문의메일 혹은 홈페이지에 등록되어 있는 별도 자료를 참조해주시기 바랍니다.

이용 약관

CILAB 자율주행 플랫폼을 구매해주신 고객님께 감사 인사를 전합니다. 고객님께서 본 제품을 이용하시는 데 있어 본 약관에 동의해 주실 필요가 있습니다. 본 약관은 고객님과 ㈜CILAB 사이에 합의되는 것으로, 고객님에게 본 로봇 차량의 이용과 관련해 중요한 정보를 제공하는 것입니다. 본 제품을 이용하시기에 앞서 19세 미만의 고객님의 경우 법정대리인의 동의가 필요합니다. 본 약관의 조건은 본 제품을 이용하는 분 모두에게 적용되며, 본 제품을 사용함으로써 본 규정에 동의한 것으로 간주합니다.

제 1조 (효력의 발생)

본 제품을 사용함으로써 귀하께서는 본 고지사항에 대해 안내를 받았으며, 이를 이해하고 준수하기로 동의한 것으로 간주합니다. CILAB 자율주행 플랫폼을 사용함으로써 발생할 수 있는 모든 파손이나 사고 에 대해서 제조사는 법적으로 어떠한 책임이나 의무를 지지 않습니다.

제 2조 (주의사항)

ㄱ. 사용자가 음주, 약물 복용, 마취, 사용중 어지럼증 호소, 피로, 메스꺼움 및 기타 신체적 혹은 정신적으로 능력을 손상시킬 수 있는 상황에서 상해 혹은 피해를 입었을 때

ㄴ. 의도적인 조작에 의해 상해 혹은 피해를 입었을 때

ㄷ. 조립 혹은 작동과 관련해 매뉴얼의 지침을 따르지 않아 발생한 사고에 의한 정신적 피해를 입었을 때

ㄹ. 제조사에서 제공하는 부품이 아닌 타사의 부품을 이용하였거나, 무단으로 개조, 혹은 본 매뉴얼 상의 규격 외의 커스텀을 통해 오작동 혹은 문제가 발생했을 때

ㅁ. 제조사에서 제공하는 제품 외의 사용자 임의 부착 제품에 의해 상해 혹은 피해를 입었을 때

ㅂ. 잘못된 조작 혹은 주관이 들어간 오판으로 인해 상해 혹은 피해를 입었을 때

ㅅ. 노후로 인해 기계적 고장으로 상해나 피해를 입었을 때

ㅇ. 배터리 부족 경고를 무시하고 동작 중 상해 혹은 피해를 입었을 때

ㅈ. 비정상적인 상태 (ex: 조립이 완전하지 않거나, 주요 부품에 이상이 있거나, 내외부 탈거된 부품이 있음)에서 임의로 제품을 이용해 상해나 피해를 입었을 때

ㅊ. 민간인 통제 구역 혹은 개인 사유지 등 민감 지역에서 사용하다가 법적 피해를 입었을 때

ㅋ. 조작 환경이 좋지 않거나 천재지변 등 걷잡을 수 없는 외부 요인에 의해 상해 또는 피해를 입었을 때

ㅌ. 사용자 과실 혹은 올바르지 않은 충전으로 인해 **화재**로 인해 재산적 혹은 신체적 상해를 입었을 때

ㅍ. 과충전으로 인한 화재 혹은 주변 화재로 인한 2차 **화재**의 원인으로 신체적 재산적 피해를 입었을 때

ㅎ. 상기 외의 본사의 책임 범위 내에 포함되지 않는 피해 혹은 상해를 입을 경우

목차

1. 제품 소개

CiLab 시리즈의 ROS 로봇은 X1(프로토타입), X2(메인 모델), X3(부분 업그레이드 모델) 등의 시리즈를 포함하고 있습니다. X3은 CiLab의 소형 ROS 교육 로봇 제품으로 2WD AKM 모델입니다.

라즈베리파이 4B, SUNRIZE X3, 젯슨 시리즈, X86 개인 PC 등 다양한 ROS 메인컨트롤러를 지원하며 라이다(Li-dar), 스테레오 카메라, 음성 컨트롤 등 고성능 하드웨어를 탑재해 맵 내비게이션, 딥러닝, 3D 시각화, 음성 인터랙션, 군집 주행, 자율주행 등의 응용이 가능하다는 장점을 가지고 있습니다.

CiLab X3

1.1. 제품 리스트 소개

CiLab X3 시리즈는 아래와 같은 모델이 포함되어 있습니다. 스테레오 카메라와 라이다를 포함하고 있는 기본 모델을 베이스로 하여, 스크린 모델은 7인치 터치스크린이 추가되며 스크린을 더 편리하게 사용할 수 있도록 미니 무선 키보드를 함께 제공합니다.

Model	Li-Dar		Depth Stereo Cam	7inch Display	미니 키보드	각종 예제코드	전용 케이스
	Slam A1	LanHai E300					
Basic		√	√			√	√
Option	선택 가능			선택 옵션	선택 옵션		

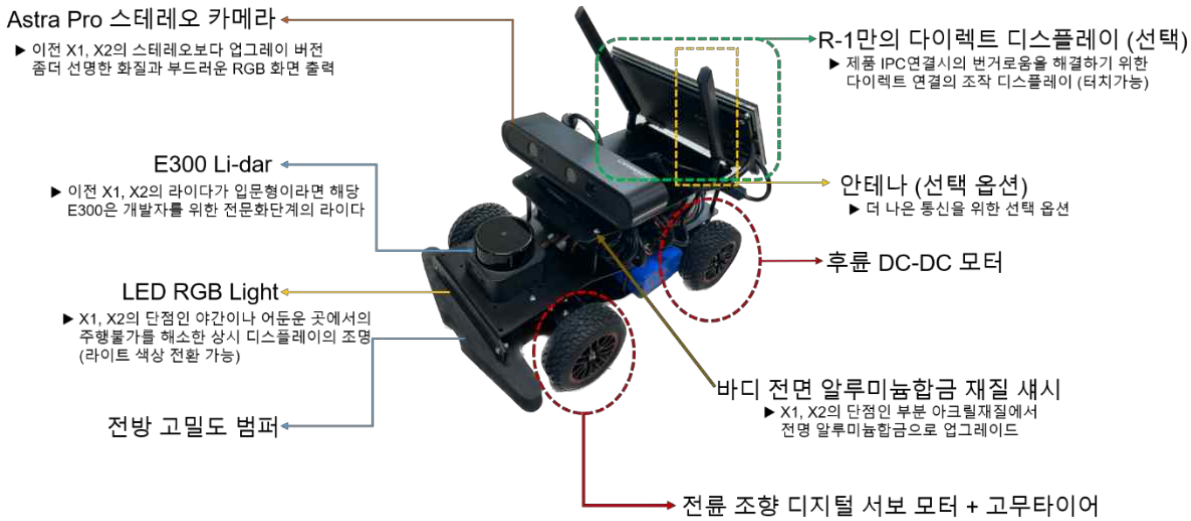
X3 시리즈 ROS 로봇 제품은 조립 및 테스트 완료 기기뿐만 아니라 아래와 같은 부품도 포함하고 있습니다. 모델에 따라 차이가 있을 수 있으며 개인 PC형은 카드 리더기와 TF 카드를 포함하지 않습니다. 무선 핸들은 로봇 조종에 사용될 수 있으며 충전기는 차량 전용입니다. USB 케이블은 OpenCTR 소프트웨어 업데이트와 STM 32 디버그에 사용되고 카드 리더기는 라즈베리파이 미러링 복사에 사용되며 TF카드는 라즈베리파이 미러링 화면을 저장할 수 있습니다.

전용 도구에는 육각 렌치(M2)가 포함되어 있어 차량의 분해 및 보수에 사용할 수 있습니다. 디스플레이에 연결할 수 있도록 HDMI 케이블도 함께 제공해드리며 이 또한 제품에 따라 차이가 있을 수 있습니다.

1.2 제품 특징

본 제품은 라즈베리파이 4B, SUNRIZE X3, 젯슨 시리즈, X86 개인 PC 등 다양한 ROS 메인보드를 이용한 컨트롤을 지원하며 Open CTR 보드는 로봇 퍼포먼스를 전담하는 모션 컨트롤러입니다.

로봇은 라이다, 카메라, IMU 등 센서를 통합하여 실내 SLAM 지도 구축, 내비게이션 장애물 회피, 이미지 인식 등의 로봇 응용 개발과 학습에 사용할 수 있습니다. 2WD AKM 모델의 핵심 부품 구성은 아래와 같으며, 모델에 따라 차이가 있을 수 있습니다.



X3 로봇은 다음과 같은 특징이 있습니다.

오픈소스 컨트롤 보드가 사용하는 STM32F407 메인 컨트롤러는 작지만 강력하다는 특징이 있습니다. 4포트 인코딩 모터, 80A 전원스위치, 듀얼 포트 5V5A 전원, IMU센서, 듀얼 포트 USB 시리얼 인터페이스, CAN, 6port PWM 스티어링 기어, RGB램프, 32개의 확장 IO가 구비되어 있어 확장 및 개발하는데에 더욱 편리합니다.

아크만형과 4륜형은 팬듈럼 서스펜디드 시스템을 개발하고 두랄루민 CNC정밀 가공방식을 채택하였으며, 고급 흑연 슬리브와 다중 정밀 베어링이 장착되었습니다. 정교함이 이전 모델인 X1, X2에 비해 업그레이드되었으며, 레이디얼(Radial) 흔들림과 축방향 가상위치 문제를 완전히 해결하여 지상 적응성이 우수하고 로봇 주행거리(미터기)가 더욱 정밀해졌습니다.

정밀도가 높은 1024라인 인코딩 모터로 업그레이드가 진행되었으며 그 성능은 일반적인 홀 인코더의 78배에 달합니다. 뿐만 아니라 주행거리 정밀도가 더 높으며 저속 내비게이션 이용 시 우수한 성능을 실현 가능해 고정밀도 SLAM 연구에도 적합합니다.

인코딩이 가능한 RGB 컬러 헤드램프로 다양한 라이팅 모드를 지원합니다.

기존보다 더 다양한 ROS 메인 컨트롤러를 지원하며 SUNRIZE X3, 라즈베리파이, 젯슨 시리즈, X86 개인 PC 등을 비롯해 ROS1 Melodic/Noetic, ROS2 Foxy/Humble 등 더 많은 ROS 버전을 지원할 수 있게 업그레이드되었습니다.

제공해드리는 오픈 소스로 ROS C++/Python 인코딩은 물론 STM32 로봇 인코딩 학습, FreeRTOS 운영 시스템 로봇 기능 개발까지 가능합니다.

라이다(Li-dar)는 산업용 TOF 라이다로 업그레이드되어 빛에 강하며 실내외에서 모두 사용할 수

있습니다. 입문용 모델을 기준으로, 상위 모델 E300은 스캔 주파수가 25HZ에 달할 정도로 레이스급 성능의 가성비의 제품입니다. (옵션 선택 사항)

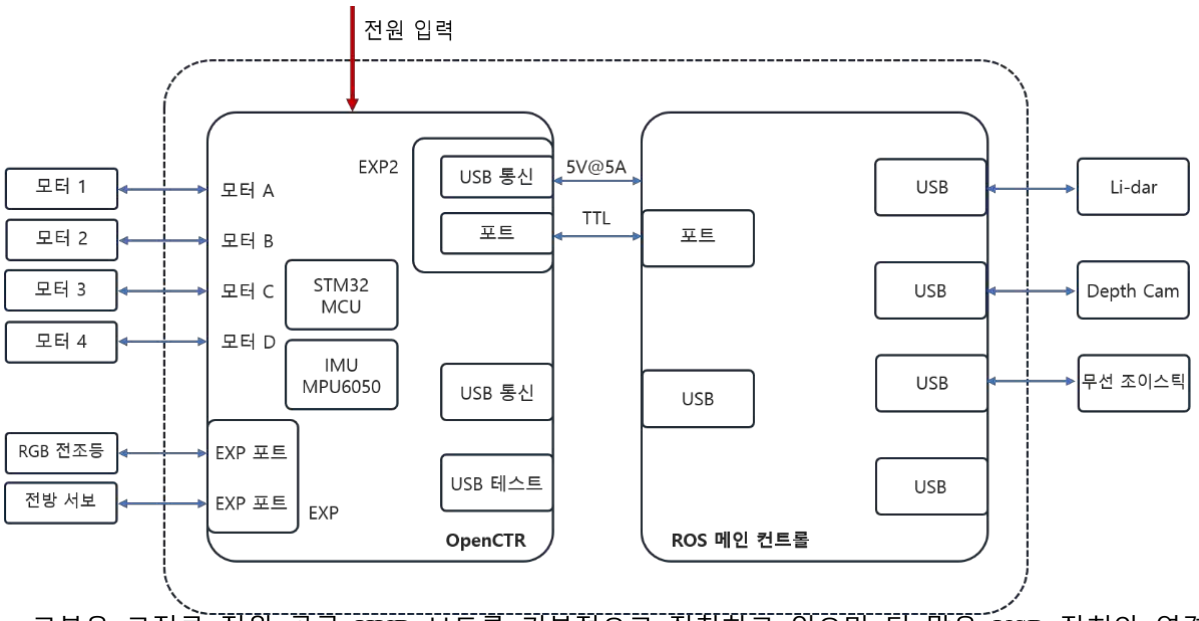
차량은 13,400mah 대용량의 국산 리튬 이온 배터리를 탑재하여 경쟁사 제품보다 2배 가까이 큰 용량과 안전성을 겸비하였습니다.

2. 하드웨어 소개

X3 시리즈 로봇 하드웨어에 대해 소개합니다.

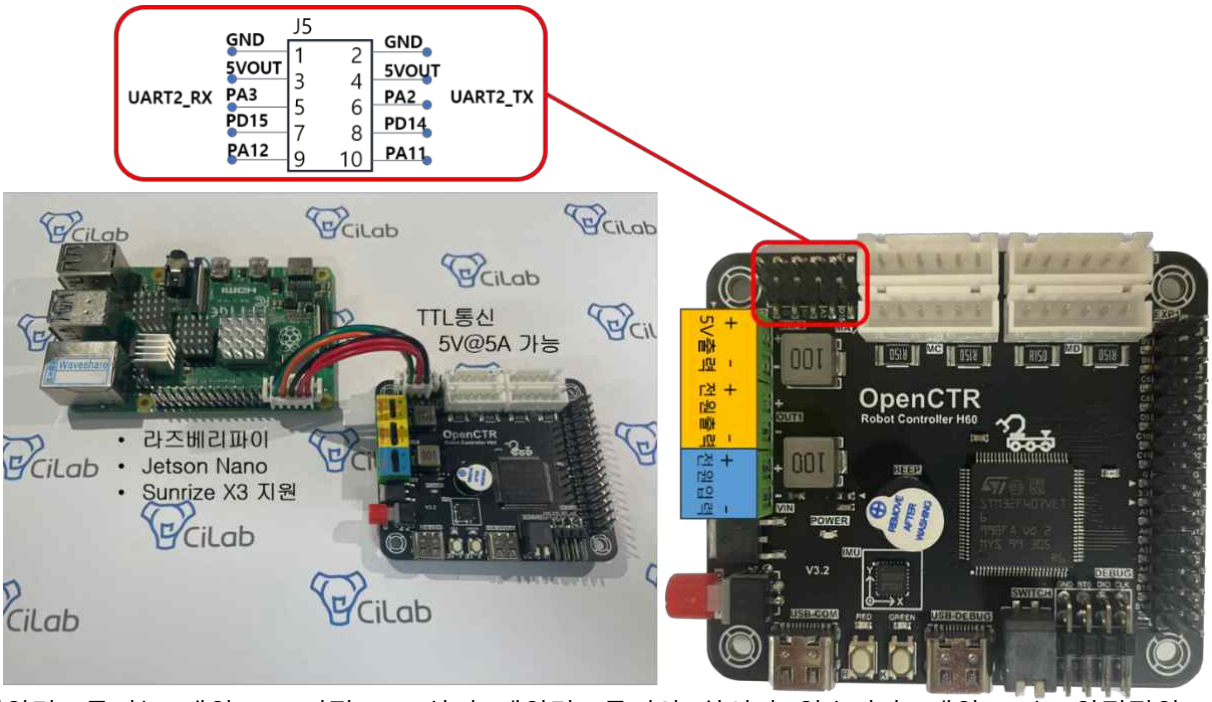
2.1. 하드웨어 시스템

X3 시리즈 로봇은 SUNRIZE X3, 라즈베리파이, Nano 메인컨트롤러를 사용합니다. 하드웨어 구성은 다음과 같이 인터페이스를 통해 메인컨트롤러와 연결합니다. 전력 공급 및 TTL 시리얼 인터페이스 통신이 가능합니다. 메인컨트롤러는 USB 인터페이스를 통해 라이다, 카메라, 무선 핸들, 그리고 키보드 등 센서 및 주변 장치와 연결할 수 있습니다.



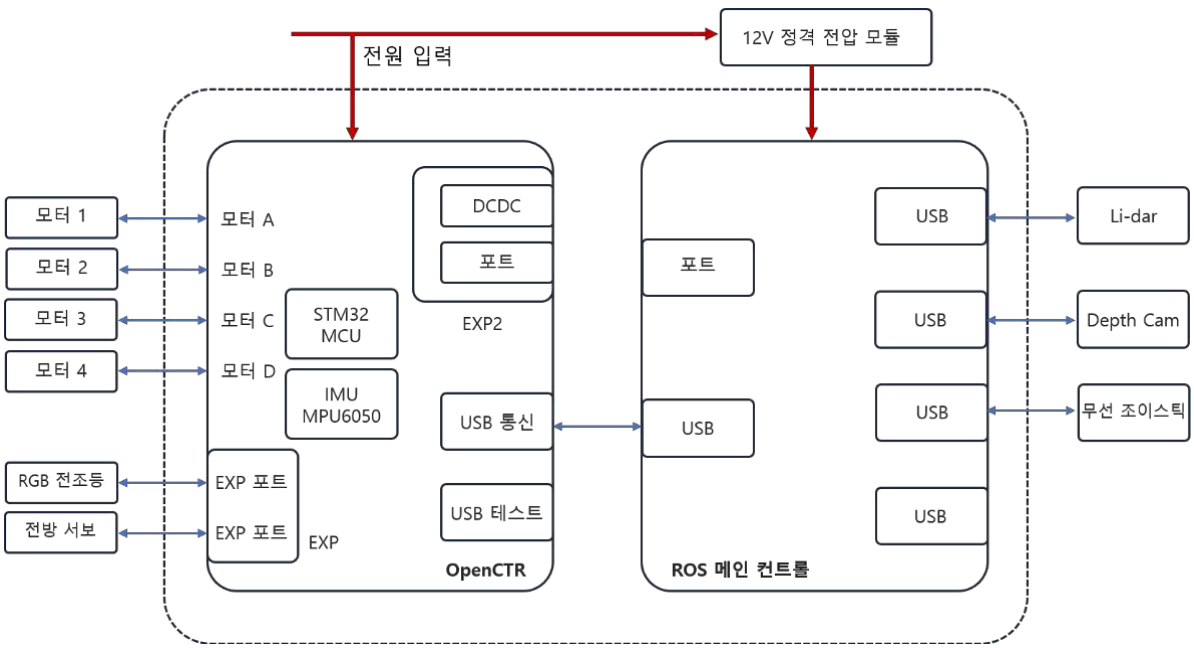
로봇은 고전류 전원 공급 HUB 보드를 기본적으로 장착하고 있으며 더 많은 USB 장치와 연결할 수 있습니다. OpenCTR 컨트롤러는 바닥부 운동 컨트롤러로서 직류 인코더가 감속 모터와 IMU 가속도 자이로스코프 센서와 연결하여 로봇 지지층 운동 컨트롤과 센서기 데이터 수집을 가능케 할 수 있습니다.

OpenCTR은 EXP2 인터페이스로 연결되며 전용 와이어를 사용하며 인터페이스 정의와 연결 방식은 다음의 사진과 같습니다. 5A의 전력 공급을 진행하기에 메인 컨트롤러 전력 공급과 시리얼 인터페이스 통신이 가능하게 되었습니다. 시리얼 인터페이스 통신은 RX, TX 두개의 핀을 통해 진행됩니다.



메인컨트롤러는 개인 PC 버전으로 상기 메인컨트롤러와 차이가 있습니다. 개인 PC는 안정적인 12V 전력 공급이 필요하며 외부 12V 전압 안정화 모듈을 사용하게 됩니다. 구체적으로는 아래의 블록도에서 볼 수 있듯이 USB 시리얼 인터페이스를 통해 개인 PC와 연결하여 통신을 진행합니다.

아래는 연결도입니다.



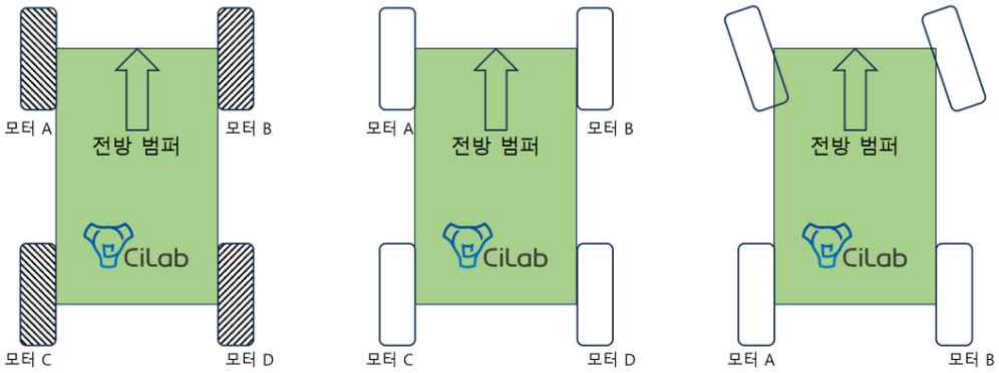
OpenCTR 계열의 H60컨트롤러는 하부 운동 컨트롤의 역할을 진행하며 직접 4포트 직류인코더 감속 모터에 연결이 가능합니다. IMU 가속도 자이로스코프가 내장되어 있어, 차량 하판 모션 컨트롤과 센서 데이터 수집을 실현할 수 있습니다. X3계열은 대효율 HUB보드와 블루투스 디스플레이 확장보드는 아래 그림과 같이 3가지를 적층하여 사용할 수 있습니다.

2.2. 섀시 종류

X3 시리즈는 모델에 따라 다른 특징을 가지고 있습니다. X3 MEC휠 로봇은 360도 전방향 이동

이 가능 모델로 전진 및 방향 전환은 물론 평면 360도 모든 방향으로의 이동이 가능합니다. 4륜과 2륜 차동(differential) 로봇은 좌우 양쪽의 바퀴의 속도차를 통해 방향 전환이 가능합니다. AKM 로봇은 시중의 자동차처럼 스티어링 기어를 통해 전륜 바퀴의 방향 전환을 컨트롤하여 이동합니다.

각 모델의 구동 관련 이미지는 아래와 같습니다.



특히 X3 시리즈 MEC휠 로봇은 트위스트 서스펜디드 장치가 설계되어 있어 바퀴와 지면의 마찰을 최적화함으로써 차량의 모션 정밀도를 이전보다 업그레이드 하였습니다. 반면에 팬듈럼 서스펜디드 장치가 탑재되어 있지 않는 MEC휠은 고르지 않은 지면에서 주행할 경우 미끄러지기 쉽고 고정밀도의 방향 이동이 어려우며 주행거리 계산 및 해석의 오차가 크다는 단점이 있습니다.

CiLab X3의 프로토 타입 이미지

2.3. ROS 메인컨트롤러

CiLab X3 시리즈는 전면 ROS 메인컨트롤러를 지지하며 각 메인컨트롤러의 사용 방법은 매뉴얼과 동일합니다. 모두 Ubuntu 시스템을 사용하며 기본 작동, SLAM 매핑 내비게이션, 이미지 처리 기능이 동일하고, AI 관련 기능에서 약간의 차이가 있을 수 있습니다.

ROS메인컨트롤러의 규격은 아래와 같습니다.

ROS 메인보드	라즈베리파이 4B	Jetson Nano	Jetson Nano NX
CPU	ARM Cortex-A72 64bit@1.5GHz 4Core	ARM Cortex-A57 64bit@1.5GHz 4Core	NVIDA Carmel ARMV8.2 64bit 6MB L2 4MB L3 6core
GPU	Broadcom Videocore IV (32-bt)	128-core NVIDIA Maxwell GPU	384-core NVIDIA Volta GPU 48 Tensor Cores
내장 메모리	4G/8G	4G LPDDR4	8G LPDDR4x
효율	15W	10W	15W
WIFI	2.4G/5.8G	2.4G/5.8G	2.4G/5.8G
전원	5V	5V	12V
처리능력	0.2T (FP16)	0.5T (FP16)	6.8T (FP16)
시스템	Ubuntu18.04 Ubuntu20.04 Ubuntu22.04	Ubuntu18.04	Ubuntu20.04
ROS 버전	ROS: Melodic/Noetic ROS2: Foxy/Humble	ROS: Melodic ROS2: Eloquent	ROS: Noetic ROS2: Foxy

Ubuntu 시스템에서 지원하는 ROS 버전의 경우 Jetson Nano는 제조사가 제공하는 것으로 업데이트가 비교적 느릴 수 있고, 특정 ROS 버전만 지원될 수 있습니다. 라즈베리파이와 개인 PC 모델은 Ubuntu 공식 시스템으로 비교적 많은 최신 버전의 ROS 지원이 가능합니다.

AI 딥러닝은 하드웨어 계산 가속 특성이 있는 Nano를 권장합니다. 더 강력한 AI 성능을 갖춘 모델로의 업그레이드가 필요할 경우 매뉴얼 마지막 고객센터로 문의해주시면 커스텀과 관련된 논의가 가능합니다.

라즈베리파이 4B 모델

Jetson Nano / NX 모델

2.3.1 라즈베리 4B

라즈베리 4B는 간단한 ROS 입문 학습에 사용될 수 있습니다. CPU 성능은 Nano와 비슷하다. 다양한 Ubuntu 시스템을 지지하며 ROS 버전 호환성이 높기 때문에 ROS 입문 학습에 적합할 뿐만 아니라 경량급 AI 딥러닝 알고리즘도 실행할 수 있다는 장점이 있습니다.

라즈베리 4B는 ARM사의 고성능 쿼드코어 Cortex-A72(ARM v8) x64를 포함하고, 2개의 micro-HDMI 포트, 최대 4K 해상도의 듀얼 스크린, 4Kp 60에 달하는 하드웨어 영상 디코딩, 8GB RAM, 듀얼 밴드 2.4/5.0 GHz 와이파이, 블루투스 5.0, 기가비트 이더넷, USB 3.0과 PoE 기능을 지원합니다.

CiLab X 시리즈 로봇은 기본적으로 32GB/64GB TF 메모리 카드가 탑재되어 있으며 라즈베리파이 RAM 크기는 4GB로 제공되었습니다. 실제 개발 텍스트에 따르면 4GB RAM 정도면 대부분의 사용 환경에 적용이 가능합니다.

대규모 편집을 해야하는 경우에는 8GB 버전으로 업데이트하여 진행하시는 것을 추천합니다. (CiLab에서는 4GB까지 기본 제공됩니다)

RTAB 기능파일집을 예로 들자면, Catographer 알고리즘 매핑 기능을 편집할 경우 CPU 메모리가 4GB를 초과할 수 있습니다.

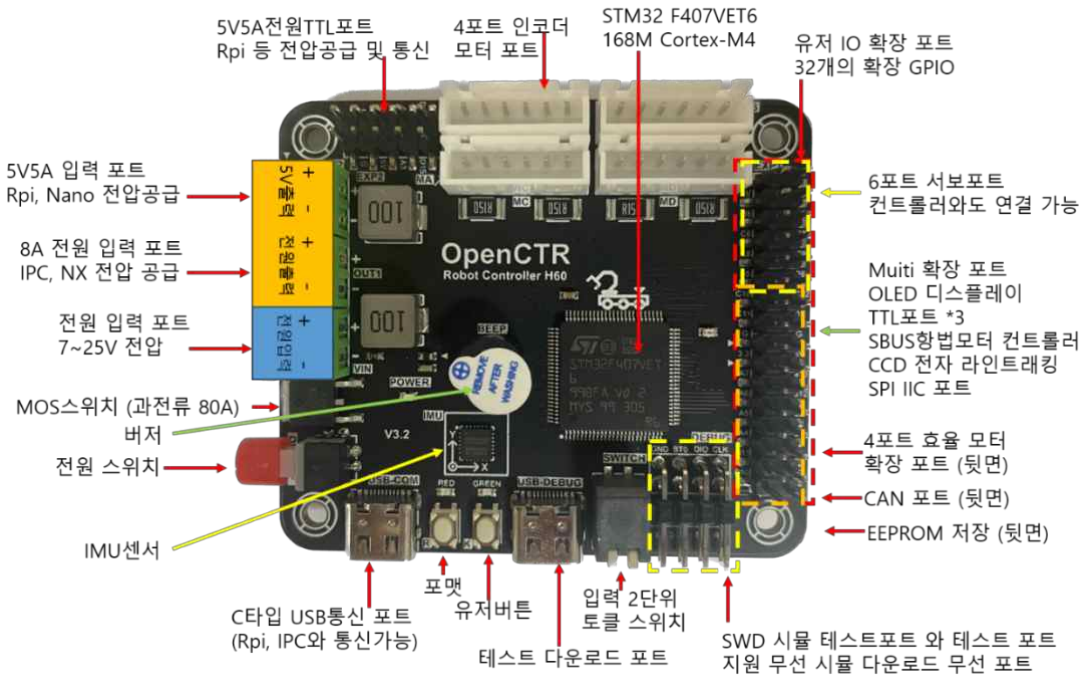
2.3.2. Jetson Nano

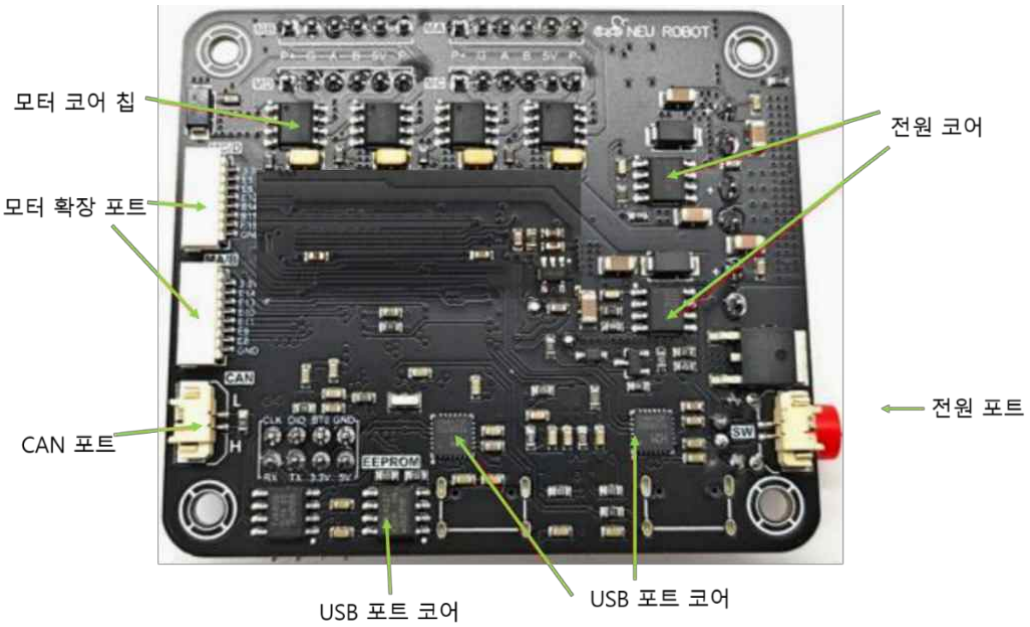
Jetson Nano는 특징적인 GPU 연산 가속 능력을 갖추고 있으며 NVIDIA TensorRT와 딥러닝 관련 자원을 사용할 수 있습니다. 라즈베리파이 4B보다 딥러닝 알고리즘을 더 원활하게 실행할 수 있습니다.

2.4 OpenCTR 컨트롤러

X3 시리즈 ROS 로봇 운동 컨트롤러는 CiLab 자체 개발 OpenCTR 로봇 운동 컨트롤러를 사용하고 있습니다. OpenCTR (Open-source Controller Module for Robot)은 로봇 개발 전용 오픈 소스 컨트롤러로서 외소한 규격과 세련된 외관이 장점입니다.

메인컨트롤러로 STM32F407를 사용하며 IMU 가속도 자이로스코프 센서가 장착되어 있는데 이는 4개의 인코더 모터 포트와 2포트 5V 전원을 지지하여 더 풍부한 온보드(on-board) 자원과 확장 인터페이스를 갖추므로 ROS 로봇 개발에 적합하다는 장점이 있습니다. 라즈베리파이, 젯슨 시리즈 등 메인컨트롤러와 함께 ROS 로봇을 구성할 수 있으며 컨트롤보드의 구성은 아래와 같습니다.



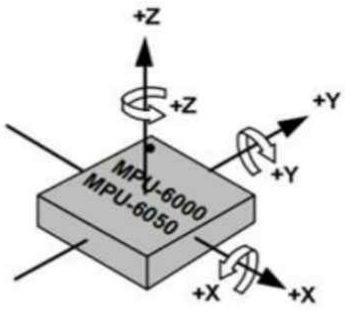


해당 보드의 인터페이스는 위 이미지를 참고하시면 됩니다.

해당 보드는 SWD 디버깅 인터페이스를 갖추고 있으며 USB 시리얼 인터페이스로 프로그램을 설치 가능합니다. JobbSTM32 인코딩 2차 개발이 가능할 뿐더러 온보드 및 외부 장치의 샘플 예제 코드도 제공하고 있기에 유저입장에서는 더 편리하게 학습할 수 있도록 서포트 가능합니다.

OpenCTR 메인컨트롤러는 ST 회사의 STM32F407VET6을 채택하였는데, ARM의 Cortex-M4 코어와 168M 메인 주파수를 사용하고 있습니다. 또한, TF 카드 내의 FLASH와 SRAM 메모리는 각각 512k와 192k이며 FPU와 DSP 지령을 통합합니다.

OpenCTR 컨트롤러는 3축 가속도와 3축 자이로스코프를 포함한 6축 IMU 센서 MPU6050을 탑재하였습니다. 센서의 경우 12C 인터페이스를 통해 컨트롤러와 연결하는데 IMU 가속도 자이로스코프 6축 정의는 아래와 같습니다.



Open CTR 컨트롤러는 확장 인터페이스 EXP1과 EXP2의 두 그룹이 존재하며, ESP2는 주로 부분 ROS 마스터 5V 전원 공급 및 TTL 직렬 통신에 사용되며, EXP1의 경우에는 확장, X3 시리즈의 스티어링 기어, RGB 램프 및 상부 확장, 보드 확장에 사용됩니다. 인터페이스 정의 및 이해 관계는

아래 그림과 같습니다.

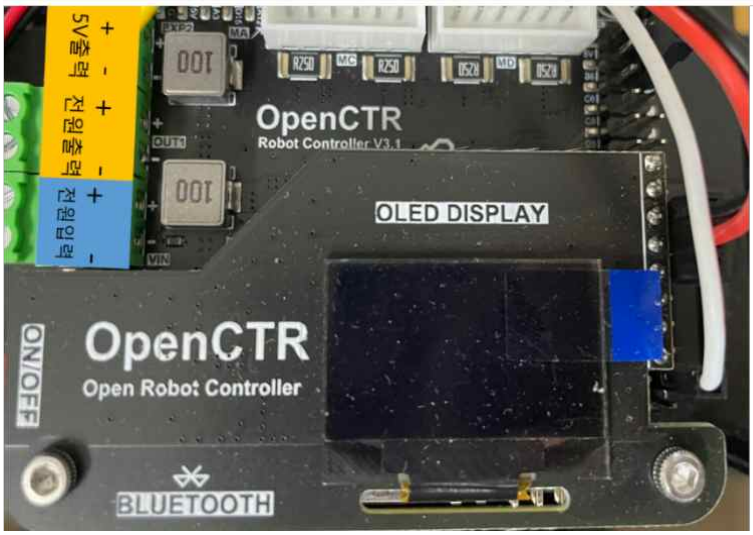
2.5. 블루투스 디스플레이 확장 보드

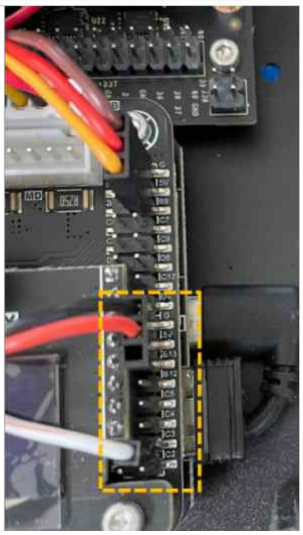
X3시리즈의 블루투스 디스플레이 확장보드가 부착되어 있습니다.

OLED디스플레이는 차량의 정보를 알려주는데에 사용되게 됩니다.

확장보드는 OpenCTR컨트롤러에 다이렉트로 연결되어 있습니다.

확장 보드의 연결 방식은 아래와 같습니다.

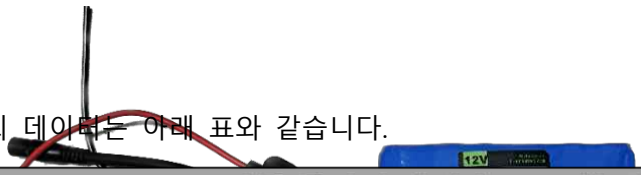




* 우측 사진에 보이는 것처럼 1번이 아닌 2번부터 시작하여 비어있는 공간이 있으니 개발 및 수리 후에 주의하여 꽂아주시면 됩니다.

2.6. 전원 설명

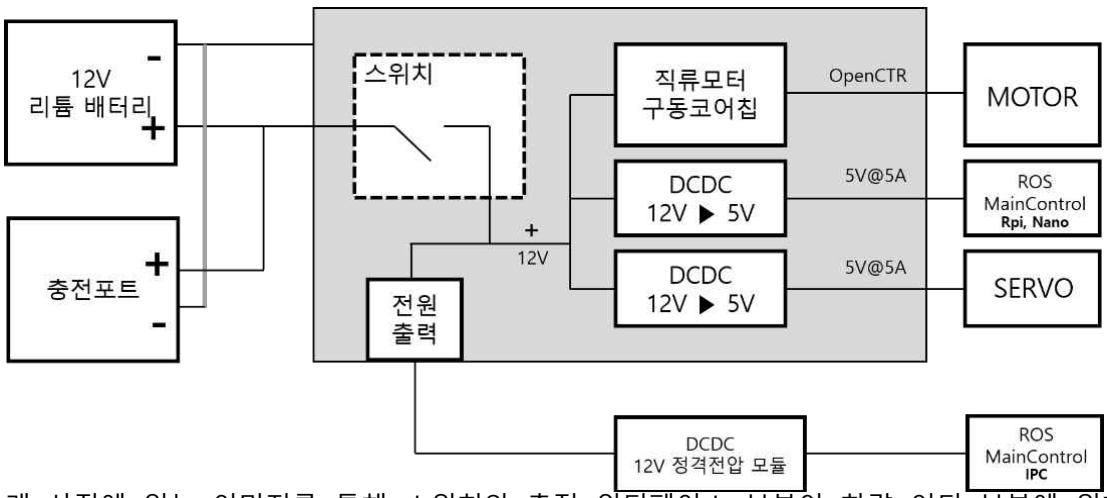
로봇은 10.8V, 13,400 mAh의 리튬 폴리머 배터리와 BMS 보호 보드를 탑재하여 과방전, 과충전, 과전압 등 다중 보호 시스템을 가지고 있습니다. 항공모듈급 T 플러그 인터페이스로 로봇에 강력한 동력을 제공하고 더 긴 항속 시간을 보장합니다.



배터리의 데이터는 아래 표와 같습니다.

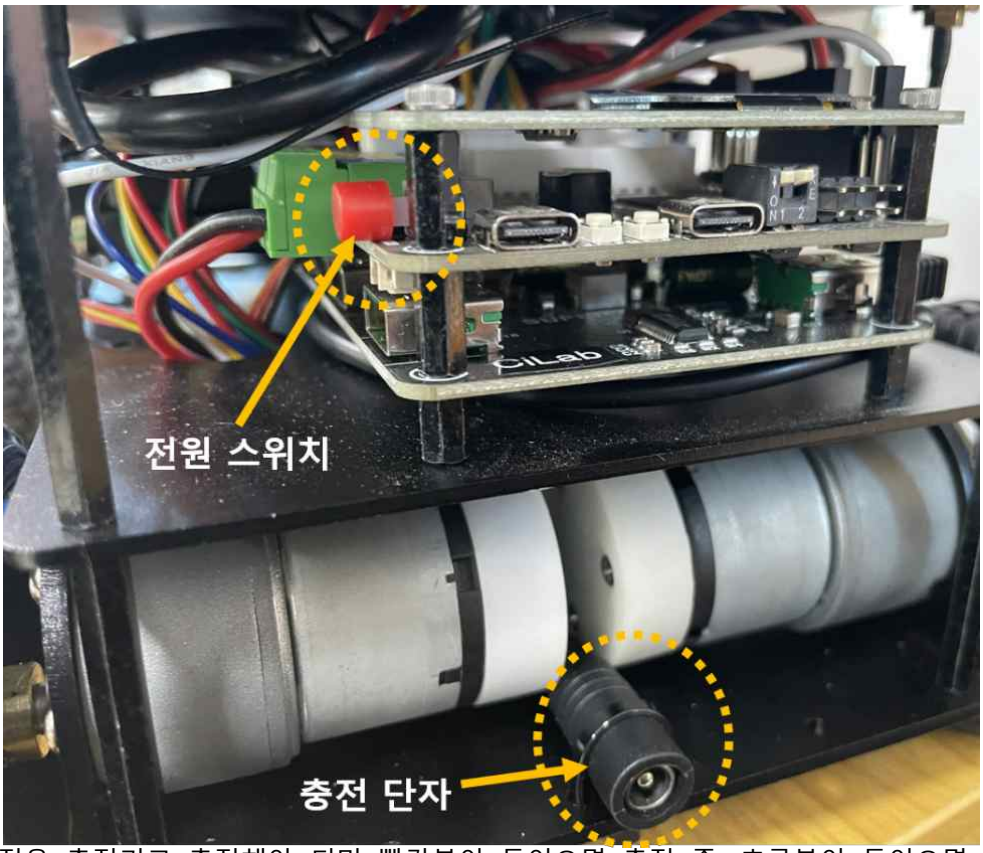
리튬 폴리머 배터리 (BMS형) 상세 정보			
용량	10.8V 13,400mah	커넥터	T형 방전 커넥터, DC 충전 커넥터
방전 전류	15A	최대 전류	25A
충전	12.6V / Max 4A	셀구성	3S4P SDI 35E

로봇 시스템 전원 블록도는 다음과 같습니다. 배터리의 10.8V를 OpenCTR에 입력하면 직류 모터 구동에 직접적으로 전력을 공급합니다. 보드에서는 DC - DC를 거쳐 5V의 전압으로 전환되어 라즈베리파이, Nano 메인컨트롤러에 전력을 공급합니다. 최대 전류는 25A에 달하기 때문에 사용에 있어 원활한 전력공급이 가능합니다. 그 외에 PWM 스티어링 기어에 작동 전원을 공급하는 DC - DC 5A5A 전원도 있습니다.



아래 사진에 있는 이미지를 통해 스위치와 충전 인터페이스 부분이 차량 어디 부분에 위치해 있는지를 확인할 수 있습니다. 그 중에서 스위치는 전자 스위치로서 최대 80A의 전류까지 가능합니다. 구체적인 회로는 OpenCTR 원리도를 참고하길 바랍니다.

주의: 로봇 작동 중에는 충전기로 충전할 수 없으며 전원 차단 후에만 충전기 충전 가능.



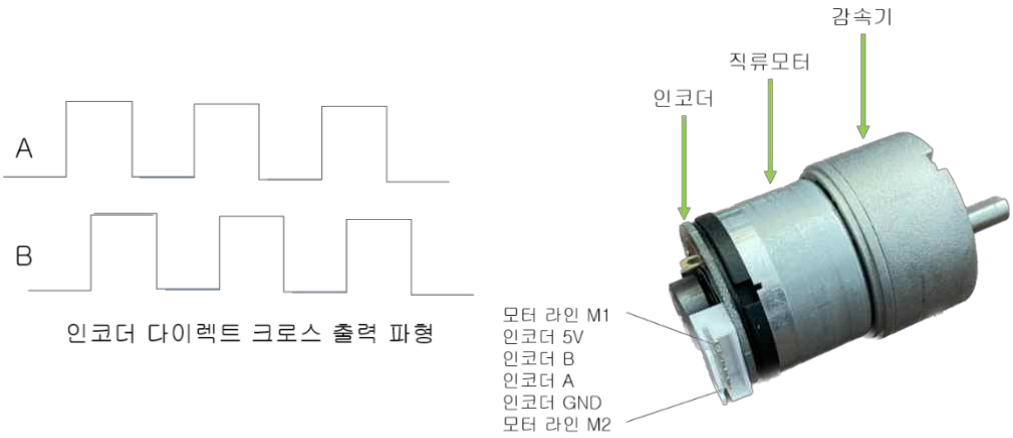
차량은 전용 충전기로 충전해야 되며 빨간불이 들어오면 충전 중, 초록불이 들어오면 충전 완료이거나 전원이 연결되어 있지 않음을 나타내는 것입니다.

차량에는 배터리 저압 경보와 보호 기능이 있으며 잔량은 OpenCTR 컨트롤보드의 빨간 LED 램프에 표시되는 신호 의미는 아래와 같습니다.

RED LED 상태	배터리 잔량	상태 정보
라이트_OFF	40~100%	배터리 전압이 10.65V 이상이며, 잔량이 충분하고 필요에 따라 충전하면 됩니다.
라이트_점멸	20~40%	배터리 전압이 10.65V 이하이며, 잔량이 40% 이하인 상태로 충전이 필요할 수 있습니다.
상시 라이트	10~20%	배터리 전압이 10.12V 이하이며, 잔량이 20%이하인 상태로 충전을 권장합니다.
알림음	0~10%	배터리 전압이 9.84V 이하이며, 부저에서 알림음이 나옵니다. 차량 자체적으로 전량보호모드로 돌입합니다. 배터리 방전으로 차량이 동작하지 않을 경우 충전을 진행하면 정상적으로 동작하는 것을 확인할 수 있습니다.

2.7. 직류 감속 모터

X3 시리즈 로봇은 CiLab 자체 개발 라지 토크(large torque) 직류 감속 인코더 모터 M250을 사용합니다. 모터는 구리 코일로 고품질 탄소 브러시, 메탈 기어, 고강도 세라믹 타일을 갖추고 있으며 우수한 성능과 내구성을 가지고 있습니다. 인코더는 1024선에 30:1 감속기를 사용하며 STM32 인코더 4배 주파수 기술과 함께 122,880 펄스/바퀴를 실현할 수 있습니다. 즉 바퀴가 한 번 돌리면 최대 122,880에 달하는 펄스를 발생시킬 수 있습니다. 모터 및 인터페이스는 아래와 같습니다.



모터 연결선은 아래 이미지를 참고하시고 각각의 위치에 있는 모터 번호에 주의하여 연결해주시기 바랍니다. 하기 사진 속 표시는 이해를 돕기 위한 표시이며 출고되는 제품에는 별도로 표기되어 있지 않으니 주의하시기 바랍니다.

2.8. RGB 램프

로봇에는 코딩이 가능한 Full Color RGB 헤드라이트가 설계되어 있으며 24비트 컬러에 1600만 가지의 다양한 색상을 표현할 수 있습니다. 뿐만 아니라 단일, 변환, 조합, 경보등, 오프로드용라이트 등 다양한 디스플레이 효과를 지지하며 ROS 토픽과 rqt 이미지 GUI 화면으로 컨트롤할 수 있습니다

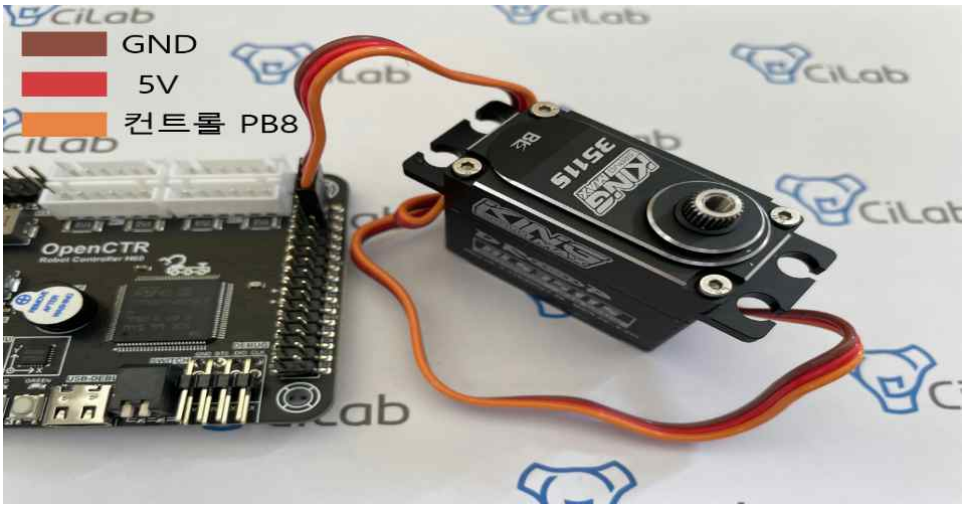
2.9. 스티어링 기어

AKM 로봇은 스티어링 기어를 통해 방향을 전환하는데 CiLab에서 설계한 맞춤형 스티어링 기어를 사용하고 있습니다. 시중의 플라스틱 스티어링 기어에 비해 정밀도와 수명이 눈에 띄게 향상되었으며 유저분들께서는 걱정없이 사용할 수 있습니다.

스티어링 기어 규격은 아래와 같습니다.

서보모터 상세 데이터			
모델명	XDS-820C 금속 토크 서보	최대 각도	180°
상대 속도	0.14sec/60°	정격전류 : 360mA	4.8~7.4V
토크	20.5kg.cm(5V) / 23kg.cm(7.4V)	피크 전류	2A (5V)

스티어링 기어 연결 위치는 아래와 같습니다.



2.10. 라이다

X3 시리즈 로봇은 두 가지 상용급 라이다 중에서 선택할 수 있다. 강렬한 빛이 비추는 실외, 실내 환경에도 모두 사용 가능하며 삼각 거리 측정 라이다에 비해 성능이 대폭 향상되었습니다. 유저는 입문용과 심화 버전에서 자유롭게 선택할 수 있습니다. 또한, ROS1/ROS2 모두 지원하는 모델이며, 두 가지 라이다의 규격 에 대한 비교는 아래의 표에서 확인할 수 있습니다.

Li-dar 명칭	SLAM A1 (고속Ver_입문자용)	PaceCat E300 (R2 채택)
최대거리	12m	25m
스캔 주파수	7~16Hz (조절가능)	10/15/25Hz (조절가능)
표본 추출 주파수	8,000Hz	18,000Hz
출력 데이터	각도, 거리	각도, 거리, 명도
측량 원리	삼각 거리 측정	TOF 거리측정
통신 방식	115,200bps	768,000bps
응용	실내환경	80KLux조도 저항 실내 실외 모두 가능
ROS 지원	ROS1 / ROS2	ROS1 / ROS2

SLAM A1은 고속 모델로 스캔주파수가 16Hz에 달하며, 8K의 표본 추출 주파수를 자랑하는 입문용의 고성능 라이다입니다. 대부분의 교육 현장에 적용가능하며, 가성비가 비교적 좋으며, 입문 유저에게 권장하는 모델입니다.

E300의 경우 고성능 TOF 라이다로 동급 라이다 중에서 가성비가 좋으며 빛에 강하다는 장점이 있기 때문에 실내외 모든 공간에서 제약없이 사용할 수 있다는 장점이 있습니다. 게다가 매핑 능력이 더 섬세하여 고속 주행이나 연구 개발, 대회 등에서 사용하기에 매우 적합한 모델입니다. 두 제품의 라이다 모두 차량에 탑재된 것으로 안정성이 좋으므로 유저께서는 안심하고 사용하실 수 있습니다.

2.10.1. 라이다 SLAM A1

A1 업그레이드 버전은 12M 거리 측정 범위, 초당 8,000회 측정 주파수, 5.5/16HZ 스캔 주파수 측정을 실현할 수 있는 CLAM 기반 Slam A1 표준 버전의 레이더 커스터마이징입니다. 표준 버전에 비해 수동 스위치를 추가하여 레이더를 끌 수 없는 사용자의 단점을 해결하고 사용하지 않을 때 레이더를 끌 수 있어 전력을 절약할 수 있습니다. 라이다의 최대 스캔 주파수를 16HZ로 증가시켜 레이더의 동적 성능을 향상시키고 맵 구축 및 항법 효과가 크게 향상되었습니다.

SLAM A1 고속 버전의 레이더 매개변수 지표는 다음 표에 나와 있습니다.

SLAM A1 상세 데이터			
거리 측정 원리	삼각 거리 측정	최대 측량 반경	12m
스캔 주파수	16Hz	표본 추출 주파수	8,000회 / 초(sec)
제품 규격	96.8*70.3*55	출력 데이터	[각도] [거리]
적용 환경	실내	ROS 지원	ROS1 / ROS2

2.10.2. PaceCat E300

E300은 TOF 기술의 2차원 펄스 레이저 레이더로 펄스 레이저를 사용하여 타겟에 단일 레이저 펄스를 방출하고 타겟에서 확산 반사된 레이저 펄스 에코를 수신하여 왕복 시간을 계산 및 처리하고 연산된 타겟의 거리 데이터 및 각도 정보를 직렬 포트를 통해 다른 장비로 전송합니다. 초당 최대 18,000회의 고속 레이저 거리 측정 및 샘플링 능력과 광자기 융합 기술을 결합하여 기존 라이다의 수명을 극복하고 장기간 안정적으로 작동할 수 있습니다.

레이더는 2차원 평면에서 최대 30m 범위 내에서 360도 전방위 레이저 거리 측정 스캔을 수행할 수 있습니다. 레이더 매개변수는 다음 표에 나와 있습니다.

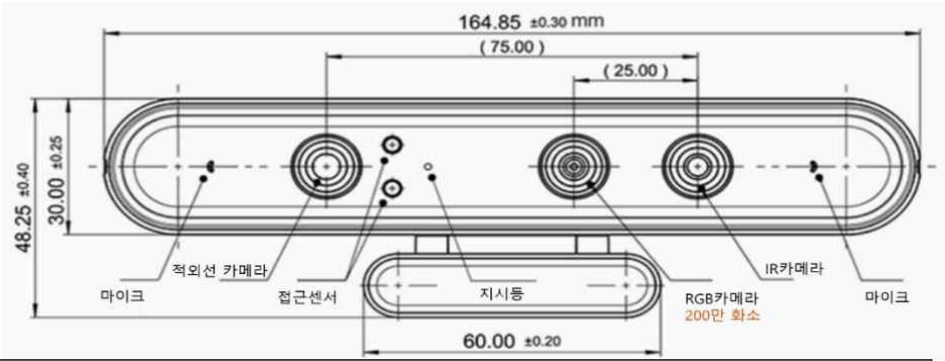
PaceCat E300 상세 데이터			
거리 측정 원리	TOF 거리 측정	최대 측량 반경	30m
스캔 주파수	10/15/25Hz	표본 추출 주파수	18,000회 / 초(sec)
각도 정밀도 (오차)	0.18°	거리 해상도	10mm
출력 데이터	[각도] [거리] [조도]		
제품 규격	57*56.7*44	구동 방식	내장 BLDC 모터
조도 환경 저항	80KLux	적용 환경	실내 · 외
포트 전송률	768,000bps	ROS 지원	ROS1 / ROS2

2.11. 카메라

2.11.1 Depth 카메라

로봇은 720P RGB 일반 카메라와 Depth 카메라의 기능을 모두 가지고 있는 오비벡(ORBBEC) Astra S Depth 카메라를 탑재하였습니다. 뿐만 아니라 듀얼 스테레오 마이크도 갖추고 있습니다. 주변 환경에 따라 자동으로 셔터 이미지를 조절할 수 있는 고성능 ISP 칩이 장착되어 있어 로봇 시각 이미지 처리에 적합화 되어 있습니다.

오비 미드라이트 Astra Pro Plus는 RGB 카메라를 200만 화소 카메라로 업그레이드하여 1080P@30 프레임을 지원하는 Pro의 업그레이드 버전입니다. 딥 프로세싱 칩은 MX6000으로 업그레이드되었습니다. 아래는 Depth 카메라의 상세 도면입니다.

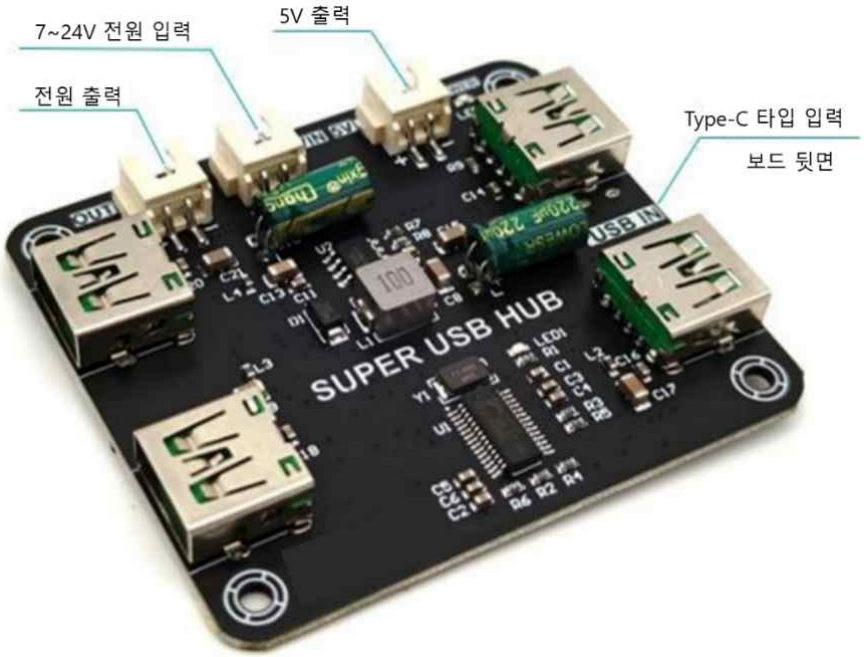


Astra Pro Plus Depth 카메라 상세 데이터			
3D 기술	단일 구조형광 + 단일 RGB	동작범위	0.6~8m
Depth 처리 칩셋	MX6000	RGB카메라 화소	200만
시각 (Depth)	H 58.4° * V 45.8°	시각 (Depth)	H 66.73° * V 46.81°
해상도 (Depth)	1280 * 1024 @ 7fps 640 * 480 @ 30fps 320 * 240 @ 30fps	해상도 (RGB)	1920 * 1080 @ 30fps 1280 * 720 @ 30fps 640 * 480 @ 30fps
제품 규격	166 * 48 * 40 mm	UVC (RGB)	지원
조작 시스템	Android / Linux / Win	데이터 포트	USB 2.0 이상

2.12. 기타

2.12.1. HUB 확장 보드

로봇에는 ROS 메인 컨트롤보드에 인터페이스 부족 및 고출력 외부 장치의 전력 공급 부족 문제를 보완하기 위해 4포트의 5A 고전류 USB HUB 확장 보드를 설치하였습니다.



초박형 디자인에 여러 겹으로 캐스케이드(cascade)될 수 있어 구동 없이 꽂으면 바로 쓸 수 있습니다.

HUB 보드는 독립적인 전원 공급이 요구됩니다. XH2.54 인터페이스로 입력되고 전원 공급 포트는 OpenCTR 컨트롤보드의 전원 출력 포트로 도입되게 됩니다. 출력은 HUB 보드와 다시 연결할 수 있어 캐스케이드를 실현하게 됩니다. 또한 HUB 보드 XH2.54 인터페이스의 5V 출력과 최대 5A의 전류가 가능합니다.

2.12.2. 무선 핸들

로봇은 무선 핸들이 기본적으로 배치되어 있으므로 더 세밀하게 로봇 운동을 컨트롤할 수 있습니다. 원격 감지를 통해 속도를 조절할 수 있으며 원격 속도 조정 거리는 10m 정도가 됩니다. 핸들과 본체를 연결하여 로봇을 컨트롤하는 것도 가능하고, 와이파이를 통해 더 넓은 범위의 원격 컨트롤을 실현할 수 있습니다. 핸들은 AAA 건전지로 전원 공급이 되며 라즈베리파이 USB 인터페이스에 꽂아서 사용하는 작은 USB 수신기를 가지고 있습니다.

핸들은 스마트 절전 기능도 있습니다. 켜진 상태에서 장시간 동안 버튼 사용이 없으면 자동적으로 수면 모드로 변경됩니다. START 버튼을 누르면 다시 작동됩니다.

2.12.3. 무선 키보드

CiLab에는 로봇에 무선 키보드를 설계해 주었는데 터치패드는 마우스 기능으로 사용할 수 있습니다. 무선 키보드는 일반 마우스로 로봇 바탕 화면을 컨트롤할 수 있을 뿐만 아니라 로봇 운동을 컨트롤하는 전용 컨트롤러로도 쓰일 수 있으며 응용 버튼 기능도 사용 가능합니다.

무선 키보드는 리튬 배터리가 내장되어 있고 USB 인터페이스를 통해 충전할 수 있습니다. 무선 수신기는 키보드 뒷면의 케이스에 있으며 아래 이미지를 참고하면 확인할 수 있습니다.

무선 키보드에는 백라이트가 있어서 꺼짐, 빨간색, 초록색, 파란색 등 네 가지 모드를 지원하며 왼쪽 하단의 FN과 F2 버튼을 통해 조절할 수 있습니다.

무선 키보드 역시 스마트 절전 기능이 있습니다. 켜진 상태에서 장시간 동안 버튼 모션이 없으면 자동으로 수면 모드로 변경되고 임의의 버튼을 누르면 다시 활성화 됩니다.

2.11.4 터치스크린

로봇은 스크린과 무선 마우스를 포함하는 터치스크린을 선택할 수 있습니다. 터치 패드는 콘텐츠 표시 및 로봇 조정에 편리하고 여러 기능 단축키가 내장되어 있어 터치만 하면 해당 기능이 작동됩니다. 사용 방법은 일반 모니터 스크린과 비슷하므로 자세한 설명은 생략합니다.

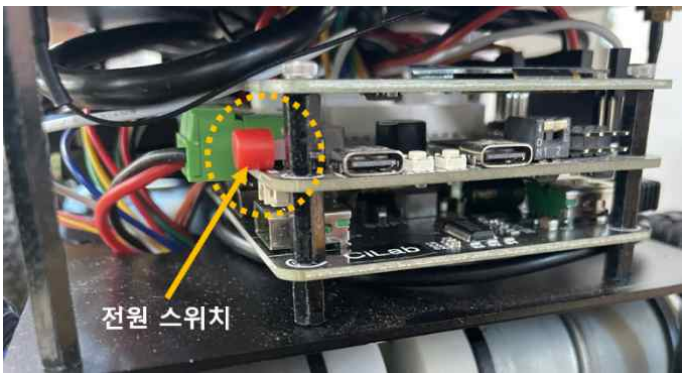

Ubuntu Desktop
Chromium Web Browser
Jetson
CiLab
Creative Idea Lab
해상도 1024*600
CILAB
THANKS
틸팅 조절 가능

3. 기본 사용

앞에서 로봇의 하드웨어 구성에 대해 이해했으며, 본 챕터에서는 로봇의 기본 사용에 대해 설명합니다.

3.1. 로봇 켜기 설명

로봇은 전원 스위치를 켜고 작동하며, 작동 시 부저 알람이 울립니다.



스위치가 켜진 후 약 2초 후에 로봇은 자이로스코프 교정을 수행하며 교정 시 OpenCTR 녹색 LED 램프가 빠르게 깜박이며 교정 기간은 약 0.5초 입니다. 교정중에는 로봇을 반드시 정지시켜 주십시오. 그렇지 않으면 교정 데이터가 정확하지 않습니다. 기계를 켤 때 로봇은 바닥에 놓고 정지하는 것이 좋습니다.

교정이 완료된 후 녹색 LED가 천천히 깜박이고 전면 RGB 램프도 약 35초 동안 호흡 효과가 있으며 ROS 마스터 컨트롤이 시작될 때까지 기다립니다. 교정이 끝나면 버저가 알람을 울립니다.

이때 OpenCTR의 소프트웨어는 이미 동작중이며, 잠시후 ROS 마스터 컨트롤의 Ubuntu 시스템이 동작하게 되며 이는 유저 컨트롤에 따라 상이할 수 있습니다.

3.2. 펌웨어 사용설명

OpenCTR 컨트롤러 펌웨어 프로그램은 ROS 제어 외에도 버튼 전환 조명 효과, OLED 화면 디스플레이, AKM 스티어링 기어 캘리브레이션, 원격 컨트롤러 제어 등을 풍부한 구성 기능을 지원하고 있습니다.

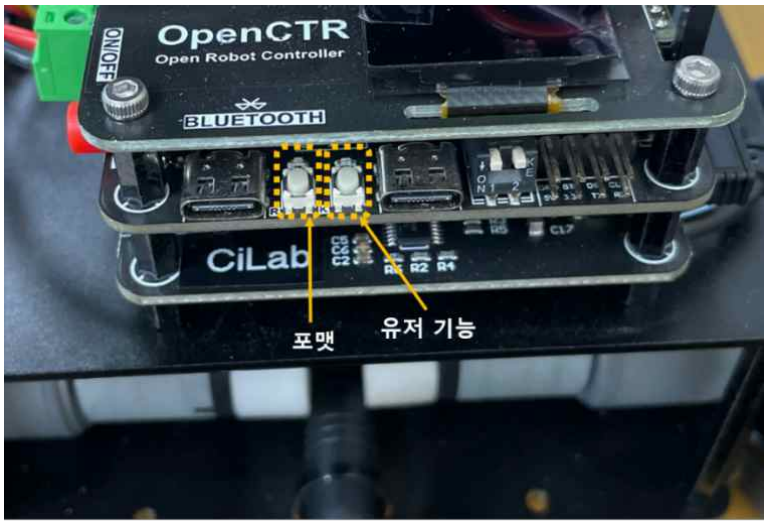
배터리 전압은 배터리의 전력을 직관적으로 이해하는 데 사용할 수 있고 로봇은 완전한 전력 관리 기능을 가지고 있으며 전력이 낮을 때 버저 또는 빨간색 LED 램프를 통해 사용자에게

알립니다.

화면에는 자이로스코프 내비게이션 각 속도 데이터도 표시됩니다. 내비게이션 각 속도 데이터는 캘리브레이션이나 자이로스코프에 이상 발생 여부를 확인하는 데 사용할 수 있습니다.

일반적으로 캘리브레이션 진행 후 데이터는 ±5 내에서 유동이 있을 수 있으며, 데이터가 너무 크면 자이로스코프가 제로로 편향되어 보정이 필요합니다.

해당 이슈에 대해서는 OpenCTR 컨트롤러를 재설정하거나 ROS rqt 도구를 통해 교정할 수 있습니다.



사용자 버튼은 3가지 조작을 수행할 수 있습니다

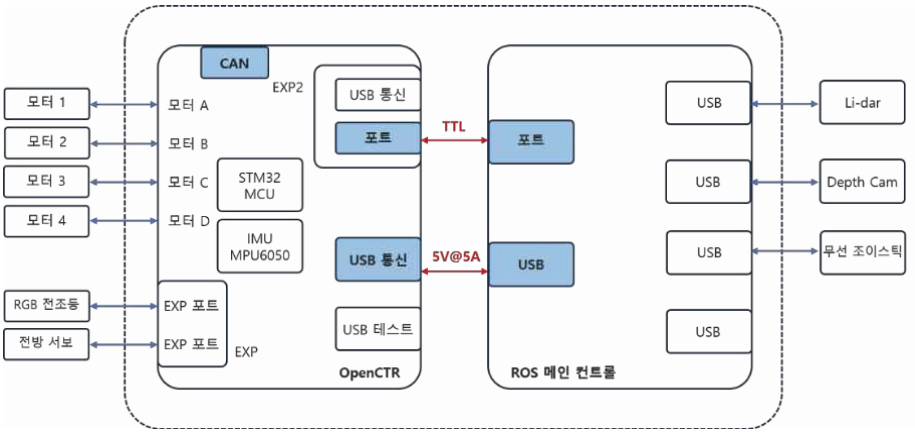
유저 버튼은 RGB 조명 효과를 전환할 수 있습니다.

오른쪽 유저버튼을 3초 이상 누르게 되면 AKM 스티어링 기어 바이어스 영점 보정 프로그램이 실행됩니다. (기타 유형은 유효하지 않음)

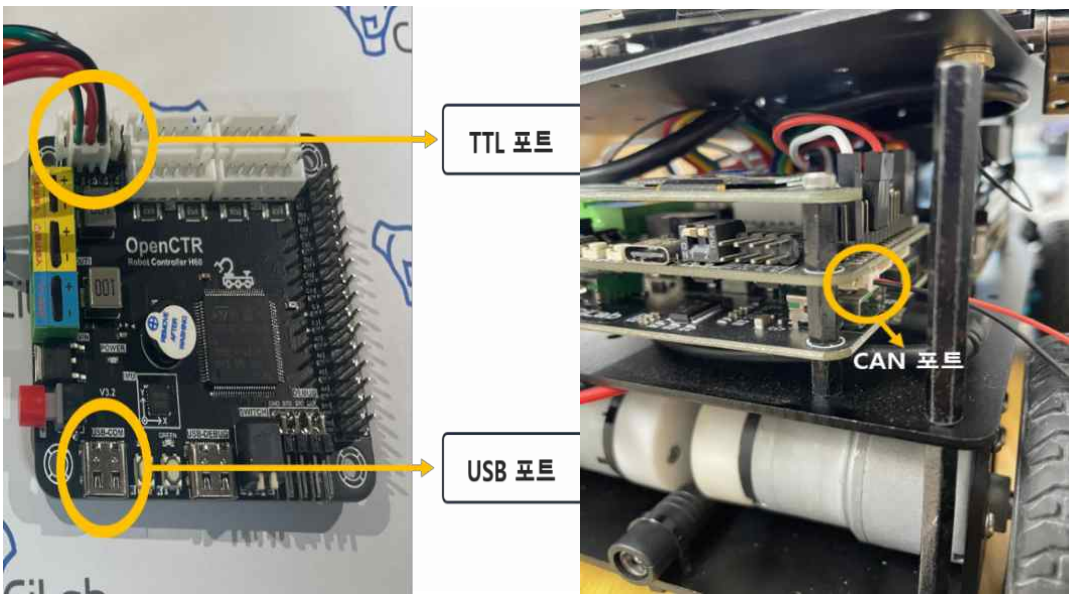
왼쪽 포맷 버튼을10초 정도 길게 누르게 되면 EEPROM에서 매개 변수를 지우고 초기화 설정으로 돌아갑니다.

3.3. 제어 인터페이스 설명

OpenCTR 컨트롤러 펌웨어 프로그램은 TTL 직렬 포트, USB 직렬 포트 및 CAN -bus 제어 등 모두 지원합니다. 기본 라즈베리 파이, 나노 및 일부 국내 마스터 컨트롤은 TTL 직렬 포트를 통해 제어되며 특수 케이블을 통해 전원 공급과 통신 제어가 가능하며 연결이 편리합니다.



작업 제어 장치는 12V 전원 공급 ROS 마스터가 필요한 경우 USB 직렬 포트를 통해 통신합니다. USB와 TTL의 두 개의 직렬 포트는 제어 명령을 수신하고 기본 마일리지, IMU, 전력 데이터를 전송할 수 있습니다. 라즈베리 파이 로봇과 같이 둘 다 동시에 사용할 수 있으며, 두 개의 마스터 컨트롤이 동시에 ROS의 CiLab_robot 하위 구동 드라이버를 시작하지 않는 한 하나의 로봇과 두 개의 마스터 컨트롤을 실현할 수 있습니다.



로봇은 CAN 인터페이스 제어도 지원하며, 인터페이스는 2P의 1.25 커넥터를 사용하여 사용자가 직접 확장할 수 있습니다.

3.4. RGB 조명 효과

X3 시리즈 로봇은 RGB의 화려한 헤드라이트를 지원하며 호흡, 랜턴, 경광등 등 다양한 디스플레이 효과를 지원하며, ROS의 rqt 도구를 통해 제어할 수 있습니다. (RGB_3가지 색상)

RQT에서 설정한 조명 효과가 계속 표시되도록 하려면 "설정 저장" 버튼을 누르면 라이트 이펙트의 파라미터가 OpenCRT EEPROM에 저장되며, 효과가 꺼진 후에도 저장됩니다.

현재 참고용으로 다음과 같은 몇 가지 조명 표시 모드가 설계되었습니다.

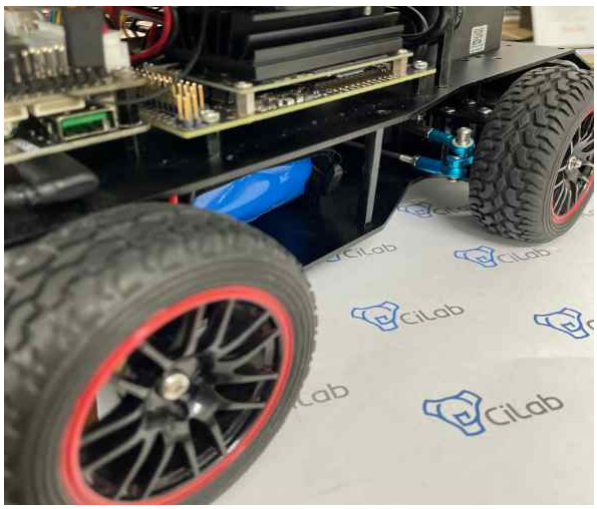
디스플레이 효과는 프로그램 코드를 기준으로 지속적으로 업데이트됩니다.

3.5. AKM 스티어링 기어 교정

아크만 로봇의 경우 조향은 스티어링 기어에 의해 제어되며 조립 시 필연적으로 특정 편차가 발생하여 제로 상태에서 앞바퀴에 좌우 편차가 발생합니다. 기계적 조립 편차를 제거하기 위해 AKM은 소프트웨어 영점 위치 보정 기능을 지원합니다. 로봇 전원을 켠 후 앞바퀴에 약간의 편차가 있는 것이 확인되면 소프트웨어에서 영점 위치를 보정할 수 있습니다.

사용자 버튼을 3초 이상 누르면 AKM 스티어링 기어의 보정 프로세스에 들어가 버저 알람이 울립니다.

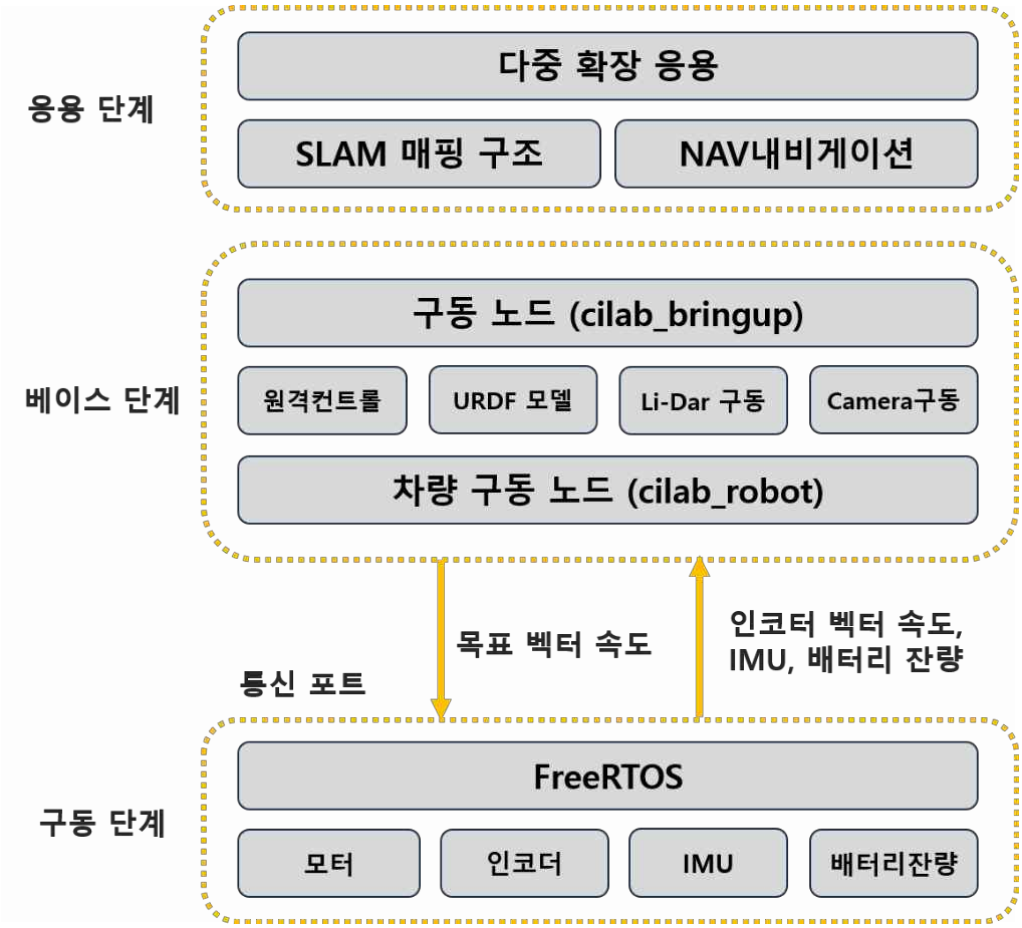
로봇을 높이 받쳐 네 바퀴를 공중에 띄웁니다. 아래와 같이 오른쪽 뒷바퀴를 돌리면 앞바퀴가 약간 좌우로 흔들리고, OLED 디스플레이에도 앞바퀴가 바로 설 때까지 보정값이 변합니다. K키를 짧게 누르면 교정이 완료되고 정상 작동 상태로 돌아갑니다. 이 시점에서 교정 매개변수는 EEPROM에 기록되며 로봇을 다시 시작하면 자동으로 새 교정 매개변수가 로드됩니다.



좌우 회전이 ±120을 초과하면 보정 최대값을 초과하고 보정 매개변수가 더 이상 증가하지 않으며 버저 프롬프트가 함께 표시됩니다

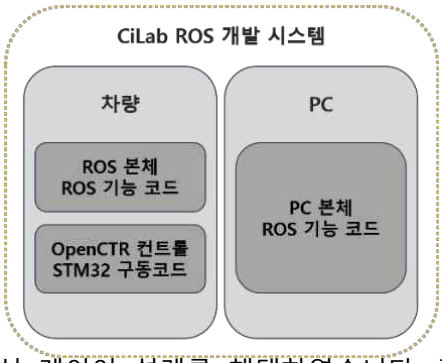
3.6. ROS 사용설명

로봇의 가장 핵심적인 기능은 ROS 제어와 ROS를 기반으로 개발된 많은 기능 루틴입니다. ROS 소프트웨어 아키텍처는 주로 드라이브 계층, 기본 계층 및 애플리케이션 계층을 포함하는 모듈식 계층화 설계를 채택하였습니다. 구동 계층은 주로 모터, 인코더, IMU 등과 같은 로봇 하부 하드웨어 제어를 담당하며 직렬 포트 프로토콜을 통해 ROS 기본 계층의 로봇 섀시 구동 드라이버와 통신하고 기본 계층에서 보내는 목표 벡터 속도를 수신하고 실시간 속도, IMU 데이터, 배터리 전압 데이터를 전송합니다. ROS 기본 계층에는 섀시 구동 기능 패키지, URDF 모델, 원격 제어 기능 패키지, 라이다 기능 패키지, 심도 카메라 기능 패키지 등이 포함되며, 부팅 드라이버(bringup)는 기본 하드웨어의 부팅 제어를 담당하고 있습니다. ROS 애플리케이션 계층에는 주로 SLAM 구축 내비게이션 기능 패키지와 더 많은 확장 애플리케이션 기능 패키지가 포함됩니다. 소프트웨어 프레임워크는 아래 그림과 같습니다.

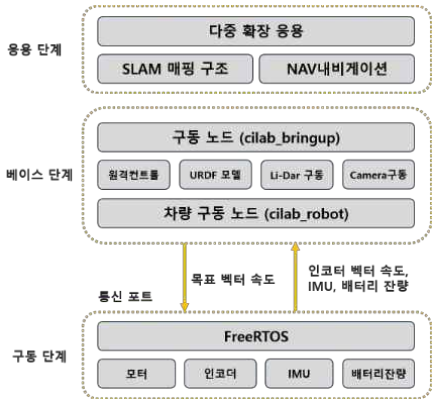

응용 단계
다중 확장 응용
SLAM 매핑 구조
NAV내비게이션
베이스 단계
구동 노드 (cilab_bringup)
원격컨트롤
URDF 모델
Li-Dar 구동
Camera구동
차량 구동 노드 (cilab_robot)
목표 벡터 속도
인코터 벡터 속도,
IMU, 배터리 잔량
통신 포트
구동 단계
FreeRTOS
모터
인코더
IMU
배터리잔량

4. 로봇 소프트웨어

본 파트는 사용자가 로봇 코드에 대해 대략적으로 이해하기 위해 작성된 파트입니다. CiLab ROS 로봇 개발 시스템은 주로 다음과 같은 세 가지 소프트웨어로 구성되며 구동에 있어서 세가지 소프트웨어가 함께 사용됩니다. 핵심 부분은 로봇 측 ROS 기능파일집 코드와 OpenCTR STM32 지지층 구동 코드입니다. 본체 측 코드는 주로 로봇 컨트롤을 보조하는 용도로 사용됩니다.



로봇 소프트웨어는 주로 모듈식 레이어 설계를 채택하였습니다. 주로 구동층, 베이스층과 응용층을 포함합니다. 구동층은 주로 모터, 인코더, IMU 등과 같은 로봇 지지층 하드웨어 컨트롤을 담당하며, 시리얼 인터페이스 협의를 통해 ROS 베이스층의 로봇 섀시 구동 드라이버와 통신합니다. 베이스층에서 보내는 목표 벡터 속도를 수신하고 마일리지 계산기가 계산한 실시간 속도, IMU 데이터, 배터리 전압 데이터를 전송합니다. ROS 베이스층에는 섀시 구동 기능파일집, URDF 모델, 원격 컨트롤 기능파일집, 라이다 기능파일집, DEEP 카메라 기능파일집 등이 포함하고 있습니다. 가동 드라이버(bringup)는 지지층 하드웨어의 가동을 담당합니다. ROS 응용층은 주로 SLAM 매핑 내비게이션 기능파일집과 더 많은 확장 응용 기능파일집을 포함합니다. 소프트웨어 프레임워크는 아래의 이미지와 같습니다.



4.1. 로봇 코드 설명

로봇 ROS 기능파일집 코드는 라즈베리파이 유저 목록 아래의 CiLab 폴더에 있습니다. 그 중 ros_ws 폴더는 ROS 작업 공간이고, src 폴더는 로봇 기능파일집입니다. 기능파일집 목록 및 설명은 아래 표와 같습니다.

명칭	설명
cilab_robot	차량 섀시 지지층 기능파일집. 차량 모션 컨트롤, IMU, 마일리지 계산지, 배터리 전압 데이터 수집 및 배포 등 기능 포함.
cilab_description	차량 URDP 모형 설명 기능파일집
cilab_teleop	차량 컨트롤 기능파일집. 키보드 및 핸들 컨트롤 기능 포함.
cilab_bringup	차량 가동 기능파일집. 로봇이 자주 사용하는 노드의 가종 및 설정 가능
cilab_slam	차량 SLAM 매핑 구축 기능파일집
cilab_nav	차량 SLAM 내비게이션 기능파일집
cilab_demo	차량 확장 응용 기능 데모 파일집
cilab_packages	제 3의 기능파일집, 레이더와 카메라 구동 기능 등 포함
cilab_doc	차량 각 기능의 상세 설명 문서 파일

이 부분은 로봇의 핵심 기능 코드입니다

4.2. Host 코드 설명

Host ROS 기능파일집 코드는 본체 유저 목록 아래의 CiLab 폴더에 있습니다. 그 중 ros_ws 폴더는 ROS 작업 공간이고, src 폴더는 주요 기능파일집입니다. 기능파일집 목록 및 설명은 다음의 표에서 제시합니다.

명칭	설명
cilab_description	로봇 URDF 모형 설명 기능파일집
cilab_teleop	로봇 컨트롤 기능파일집. 키보드 및 핸들 컨트롤 기능
cilab_rviz	차량 RVIZ 디스플레이 배치 파일
cilab_doc	각 기능 상세 설명 파일

4.3. STM32 코드 설명

OpenCTR 컨트롤 보드는 로봇 지지층 구동 보드로 주로 로봇 모터 PID 컨트롤, 인코더, IMU 데이터 수집, RGB 헤드램프 컨트롤 등을 담당하며 시리얼 인터페이스를 통해 ROS 지지층의 로봇 섀시 구동 드라이버와 통신하고 기본 계층에서 보내는 목표 벡터 속도를 수신하고 마일리지 실시간 속도와 IMU 데이터, 배터리 전압 데이터를 전송합니다.

위의 기능을 더 잘 실행하도록 OpenCTR은 FreeRTOS 임베디드 컨트롤 시스템을 사용하여 소프트웨어 설계를 하였습니다.

4.4. 코드 편집 설명

ROS 코드를 읽고 편집하기 위해 CiLab에서는 기본적으로 마이크로소프트 VSCode 편집기를 사용합니다. VSCode는 로컬 코드를 읽는 것 외에도 SSH Remote를 사용하여 로봇에 연결하여 코드를 편집할 수 있어 편리합니다.

또한 높은 코드, 자동 완성 등의 기능이 있으며 플러그인 또한 매우 풍부하여 현재 사용 편의성이 가장 뛰어난 편집기 소프트웨어라고 할 수 있습니다.
일반적인 ROS 개발 요구 사항을 충족시킬 수 있습니다.

4.5. OpenCTR 코드 업데이트

OpenCTR 컨트롤러는 STM32 코드를 실행하는 로봇의 하부 모션 컨트롤러로, 로봇이 출고될 때 해당 코드를 태워서 직접 사용할 수 있습니다. 일반 ROS 개발 학습 사용자는 업데이트할 필요가 없으며 기본 STM32 개발 학습이 필요한 경우 OpenCTR 코드를 업데이트하거나 디버깅할 수 있습니다.

OpenCTR은 USB 직렬 포트를 통해 ISP 코드를 업데이트하거나 SWD 인터페이스를 통해 코드를 업데이트하거나 디버깅할 수 있습니다.

4.6. 로봇 조립 해체

로봇은 기본적으로 CiLab에서 조립 및 테스트를 거쳐 출고되며, 사용자가 로봇의 하드웨어 구성에 대해 더 알고 싶다면 로봇을 분해하거나 스스로 기능을 확장할수도 있게끔 설계되어 있습니다. (자사 홈페이지에서 해당 파일도 다운로드 가능합니다)

5. 사용 환경 구축

본 챕터에서는 ROS 버전, 인터넷 토대 및 구성, 본체 환경 구축 등 ROS 로봇의 사용 환경에 대해서 설명합니다. 학습을 통해 유저는 ROS 로봇의 조작 방법과 운행에 대해서 알 수 있습니다.

사용 환경 구축은 다음의 단계를 포함하고 있습니다

1. 가상 메인컨트롤러 구축.
2. 로봇 인터넷 연결.
3. 본체 로봇 연결.

5.1. ROS 버전 설명

로봇은 다양한 ROS1와 ROS2 버전을 지지하며 메인컨트롤러에 따라 차이가 있습니다. 구체적으로는 다음의 표를 참조할 수 있습니다.

CILAB R20 차량 ROS 시스템 지원 현황				
ROS 본체	Sunrize X3 4G	Rpi 4B 4/8G	Jetson Nano 4G	IPC X86 8G
출고 세팅	Ubuntu20.04 LTS ROS: Noetic/Foxy	Ubuntu20.04 LTS ROS: Noetic/Foxy	Ubuntu18.04 LTS ROS: Melodic/Eloq	Ubuntu20.04 LTS ROS: Noetic/Foxy
확장 시스템 1		Ubuntu18.04 LTS ROS: Melodic		Ubuntu18.04 LTS ROS: Melodic
확장 시스템 2		Ubuntu22.04 LTS ROS2: Humble		Ubuntu22.04 LTS ROS2 : Humble

대부분 로봇은 메인컨트롤러의 출고 기본 시스템은 Ubuntu20.04 LTS 버전으로 ROS1Noetic 버전과 ROS2 Foxy 버전도 탑재되어 있습니다. ROS1와 ROS2는 같은 시스템에 설치되었습니다. Nano를 제외하고 현재 공식적으로는 Melodic와 Eloq를 탑재한 Ubuntu18.04 시스템만 가능합니다.

현재 CiLab Ubuntu20.04 시스템을 통해 개발할 수 있으며 ROS2의 Humble 버전을 사용할 수 있습니다. 뿐만 아니라 프로그램에 대해서는 5년 장기 유효 버전을 지원하고 있습니다.

현재 공식 추천 버전인 Noetic 버전은 ROS1의 마지막 버전이기도 하며 이 유효기간은 2025년까지입니다. Melodic 버전에 대한 지원은 2023까지이므로 Noetic 버전이 이미 점점 주류 버전이 되어가고 있는 추세이기 때문에 CiLab의 자율주행 플랫폼 튜토리얼도 Noetic 버전 위주로 설명이 제공됩니다.

Melodic 버전은 기능파일집과 미러링만 제공하며 튜토리얼은 Noetic 버전을 참고하면 되고 차이점에 대한 설명도 포함되어 있습니다.

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)

현재 공식 홈페이지 추천 ROS2 버전은 Foxy 버전이고 CiLab의 X3 시리즈 ROS 로봇의 미러링과 튜토리얼도 Foxy 버전을 기준으로 설명합니다. 일부 메인컨트롤러도 점차 Humble 버전을 지원합니다.

Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023

Noetic 버전과 Foxy 버전이 대응하는 Linux 버전은 모두 Ubuntu20.04 시스템을 사용하며 가상 본체 측은 역시 Ubuntu20.04 시스템을 사용합니다. Melodic 버전의 로봇 측은 Ubuntu18.04이고, 가상 본체 측은 기본적으로 Noetic 버전이 공용하는 Ubuntu20.04 버전을 사용한다. 두 버전은 호환됩니다.

Melodic 버전이 설치된 Ubuntu18.04 시스템도 같이 제공하므로 필요에 따라 선택할 수 있습니다.

다음은 Ubuntu20.04 , ROS1 Noetic버전 , ROS2 Foxy 버전 위주로 설명할 것이지만 Melodic 버전 역시 참조할 수 있습니다. 대부분의 내용은 비슷하고 차이점은 수업이나 동영상을 통해 설명할 것입니다.

Nano 미러링 차량 명칭 : cilab, 유저 아이디: cilab, 비밀번호: cilab.

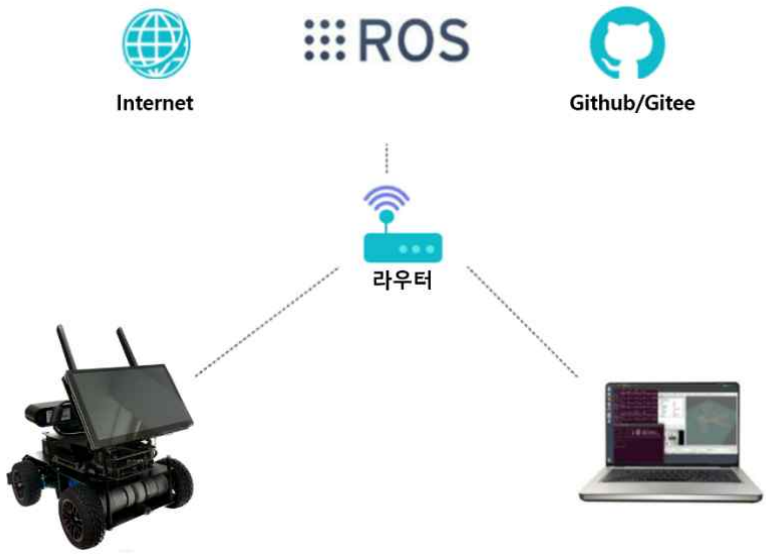
가상 PC 명칭 : cilab-pc, 유저 아이디: cilab, 비밀번호: cilab.

5.2. 네트워크 프레임워크 설명

ROS는 분산식 설계 프레임워크입니다. 서로 다른 본체는 ROS 네트워크를 통해 통신할 수 있습니다. 일반적으로 소형 혹은 마이크로 로봇 플랫폼의 컨트롤 시스템은 다중 프로세서 실현 전략을 선택할 수 있습니다. 구체적으로는 PC 본체와 임베디드 본체는 임베디드 시스템(예: 라즈베리파이)을 구성하며 로봇 본체의 시스템으로 작동합니다. PC 본체는 원격 모니터링을 담당합니다. 전자를 통해 데이터 수집과 섀시에 대한 직접적인 컨트롤을 실현합니다. 후자는 원격으로 이미지 디스플레이 및 기능 연산을 실현합니다.

로봇 개발 환경은 본체와 로봇을 포함하고 있습니다. 시스템은 Ubuntu20.04이고 ROS Noetic 버

전과 소프트웨어 기능파일집도 같이 설치되어 있습니다. 본체는 Ubuntu20.04이 설치되어 있는 가상 혹은 물리 PC 본체에서 작동되며 역시 ROS Noetic 버전과 관련 기능파일집이 설치됩니다.



라우터가 인터넷에 접속될 수 있으면 로봇이나 본체도 쉽게 ROS 소스 , GitHub와 Gitee의 ROS 소프트웨어 파일집 소스 코드를 포함한 인터넷 자원을 사용할 수 있습니다.

주의해야 할 것은 GitHub와 ROS 소스의 경우 국내 접속이 비교적 느린 편이므로 더 나은 개발감을 위해서, 특히 ROS 소스와 기능파일집을 자주 다운로드해야 하고 개발 수요가 높은 유저에게는 가능하면 과학적으로 인터넷을 접속하는 것을 권장합니다.

5.3. 본체 환경 구축

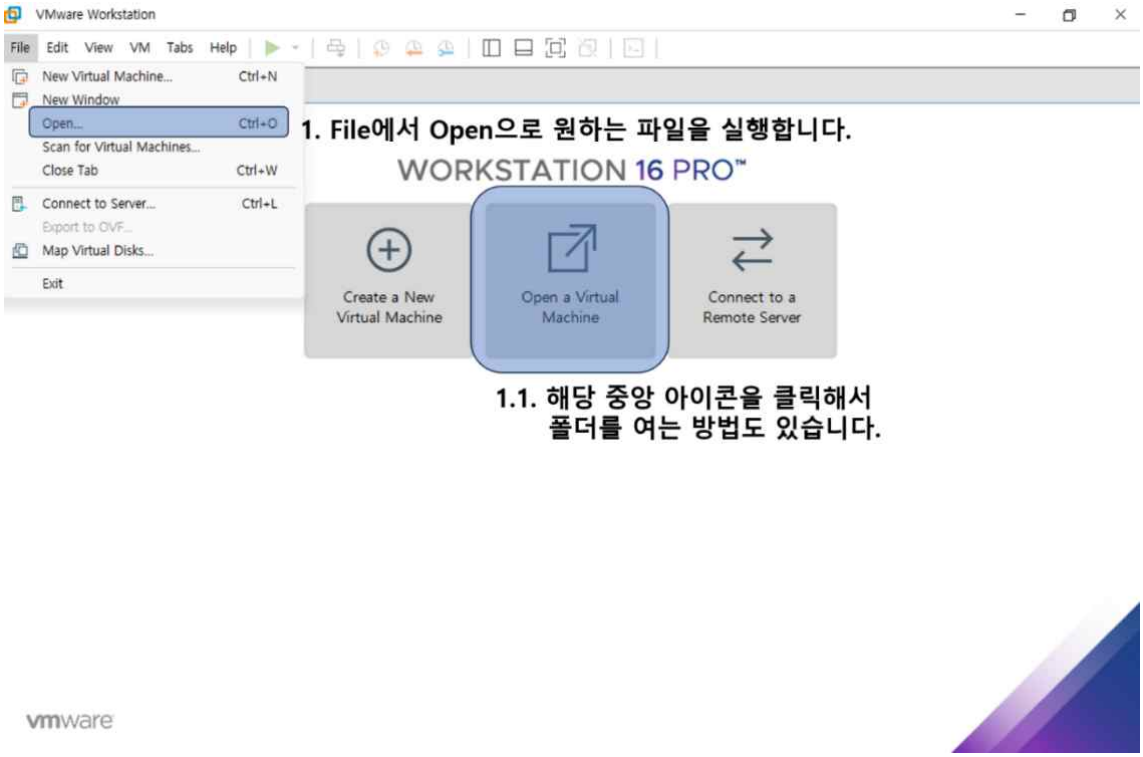
ROS 본체는 Ubuntu 시스템을 사용하는 컴퓨터이고 Ubuntu 시스템은 물리 PC 본체나 가상 프로그램에도 운용할 수 있습니다.

이 챕터에서는 CiLab에서 제공하는 VMware 가상 머신를 예로 들어 설명합니다.

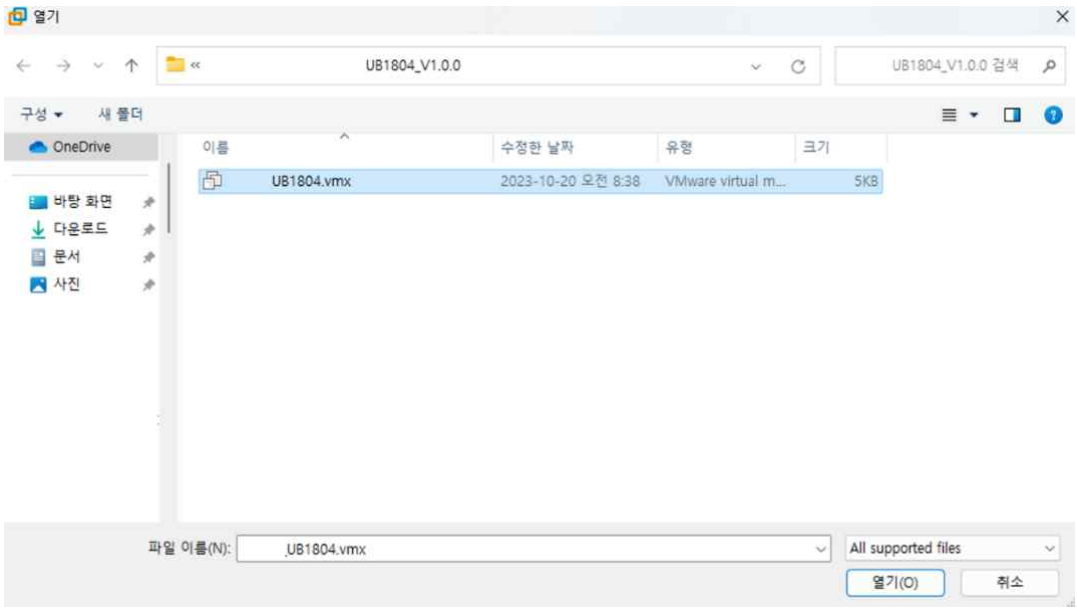
ROS 초보자는 우리가 제공하는 가상 프로그램을 통해 공부하고, 기본적인 GUI지식을 습득한 후에 별도로 구축하는 것을 권장합니다.

우선, VMware 가상 머신를 설치합니다. VMware Workstation 소프트웨어는 VMware 공식 사이트에서 다운받을 수 있으며 최신 버전인 VMware Workstation 16버전을 권장합니다.

VMware 설치가 끝난 후 관리자 권한으로 실행하는 것을 권장합니다. 제공하는 Ubuntu20.04 가상 프로그램으로 로딩할 수 있는데, 해당 프로그램은 자사에서 제공하는 링크를 통해 다운로드가 가능합니다. 가상 프로그램의 압축을 풀어서 SSD에 저장해 놓는 것을 권장합니다. SSD 크기는 50G 정도이므로 일반 USB에 저장하면 대기 시간이 비교적 길어지므로 사용감이 떨어질 수 있습니다. 반면에 SSD에 저장할 경우에는 읽기 및 쓰기 속도가 크게 향상되고 본체에 가까운 사용감을 느낄 수 있습니다.

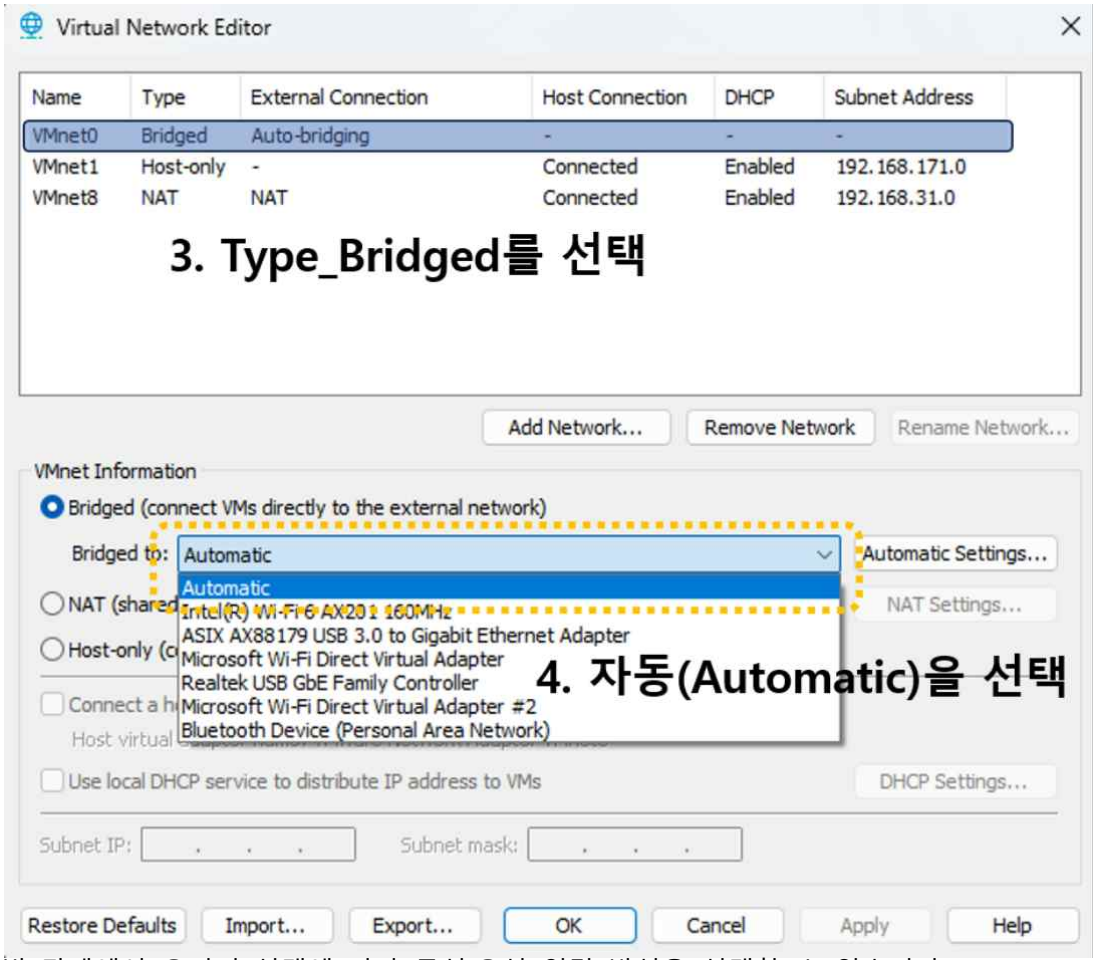


경로 선택 화면에 앞에서 압축을 풀어 놓은 폴더를 선택하고 안에 있는 .vmx 파일을 을 실행합니다.



그런 다음에 아래와 같은 화면이 뜨는데, 저희가 제공한 가상 머신은 RAM 6GB, 디스크 40GB, 그리고 프로세서 4코어에 해당합니다. 가상 프로그램을 시작하기 전에 인터넷을 먼저 설정해야 합니다. 아래와 같이 'Edit' 메뉴의 'Vertual Network Editor'를 클릭합니다.

만약 회색 표기가 나타나면 관리자 권한이 아님을 나타내므로 '설치 변경' 아이콘을 클릭해 관리자 권한으로 변환해줍니다.

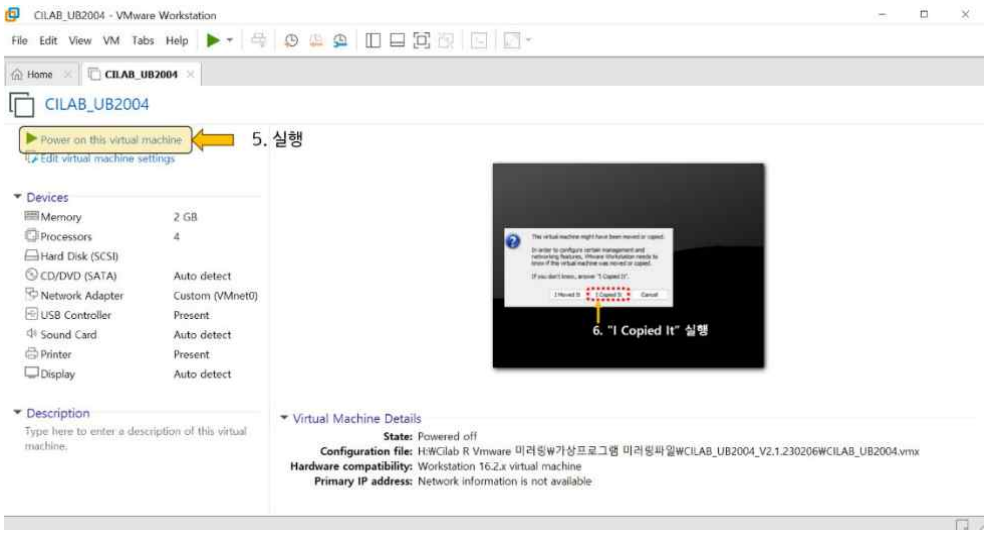


*4번 단계에서 유저의 선택에 따라 무선/유선 연결 방식을 선택할 수 있습니다.

해당 단계에서는 네트워크를 추가합니다. VMware에는 3가지 네트워크 연결 방식이 있는데, 그

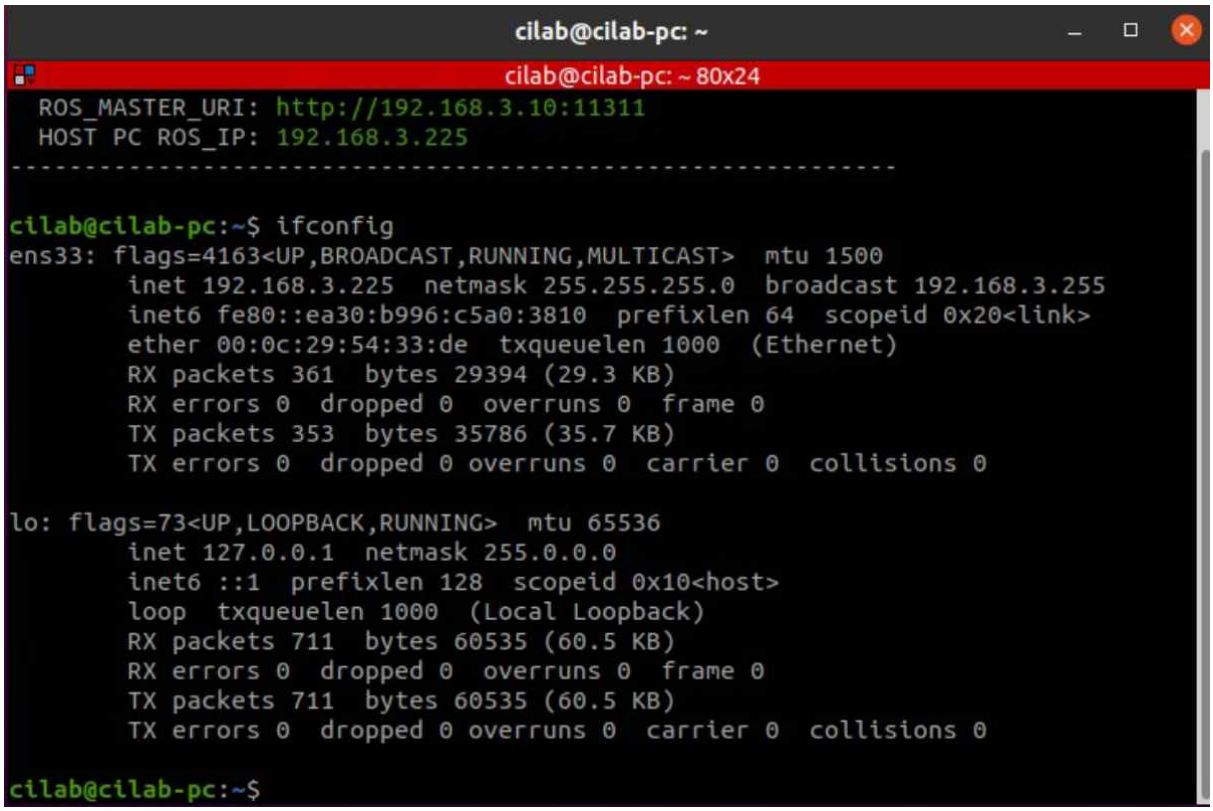
중에서 브릿지(bridge) 모드를 선택합니다. 컴퓨터가 실제로 사용하는 네트워크 카드 장치를 선택해야 연결할 수 있음을 주의해야 합니다. 보통 데스크톱은 일반 유선 네트워크 카드를, 노트북은 무선 네트워크 카드를 사용하는데 유저는 본인의 상황에 따라 선택할 수 있습니다. 자동 선택이 있는 경우에는 자동 옵션을 우선적으로 선택해도 좋습니다.

이상 네트워크 설치가 완료된 후에 'Power on the virtual machine'를 클릭하면 가상 시스템을 실행할 수 있습니다. 아래의 알림이 뜨면 'I Copied It'을 클릭합니다.



CiLab 가상 머신이 실행된 후 화면은 다음과 같습니다. 유저의 더 편리한 사용을 위해 CiLab 가상 머신에는 공식 Ubuntu20.04를 기초로 해서 터미널 소프트웨어, VS Code 등 소프트웨어도 설치되어 있습니다. 뿐만 아니라 ROS Noetic 버전 및 필수 ROS 기능파일집도 같이 설치하여 직접적으로 로봇을 컨트롤할 수 있습니다.

실행 후 ifconfig 명령을 통해 가상머신의 IP 주소를 확인할 수 있습니다.



이렇게 가상 머신이 유저의 컴퓨터에 연결되었으며 컴퓨터가 인터넷에 접속 가능할 경우 가상 프로그램에서도 인터넷에 접속할 수 있습니다.

로봇 연결과 본체 환경의 편리한 사용을 위해 CiLab는 본체의 환경 파라미터 파일 .bashrc에 아래와 같은 항목을 추가할 수 있습니다. 유저 목록에서 gedit 편집기로 확인 및 수정할 수 있습니다. 파일을 여는 방식은 다음과 같습니다.

```
cilab@cilab-pc:~$ gedit .bashrc
```

cilab@cilab-pc:~$ gedit .bashrc

```
#-------------CILAB HOSTPC CONFIG-------------------------------------
# ROS_Setting Environment Variables
source /opt/ros/noetic/setup.bash
source ~/cilab/ros_ws/devel/setup.bash

# ROS2_Setting Environment Variables
#source /opt/ros/foxy/setup.bash
#source ~/cilab/ros2_ws/install/setup.bash

# ROS_Setting Main PC Network
interface=ens33
export IPAddress=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
export ROS_IP=$IPAddress

# ROS_Robot Network Addrees
export ROBOT_IP=192.168.3.10
export ROS_MASTER_URI=http://$ROBOT_IP:11311

# alias_Pass Quickslot key
alias cw='cd ~/cilab/ros_ws'
alias cs='cd ~/cilab/ros_ws/src'
alias cm='cd ~/cilab/ros_ws && catkin_make'

# alias_Robot Connection Quickslot
alias nfsrobot='sudo mount -t nfs -o nolock $ROBOT_IP:/home/cilab/cilab /home/cilab/Robot'
alias unfsrobot='sudo umount -lf /home/cilab/Robot'
alias sshrobot='ssh -X cilab@$ROBOT_IP'

# BASH Terminal Display
echo ""
echo "-------------------- CILAB HOST PC INFO --------------------"
echo -e "  ROS_MASTER_URI: \033[32m$ROS_MASTER_URI\033[0m"
echo -e "  HOST PC ROS_IP: \033[32m$ROS_IP\033[0m"
echo "------------------------------------------------------------"
echo ""

#-------------CILAB HOSTPC CONFIG-------------------------------------
```

① ROS의 환경 파라미터. ROS 설치 위치와 cilab ros_ws 작업 환경 파라미터, ROS1과 ROS2를 포함합니다.

② 로봇 측 네트워크 설정. 빨간색은 설정이 필요한 로봇 IP 주소임을 나타내며 다음 챕터의 내용에 따라 로봇 IP를 확보한 다음 새로 확보한 IP로 바꾸고 해당 주소로 ROS_MASTER_URI를 설정합니다.

③ 로컬 네트워크 설정. 자동으로 본체의 IP 주소를 얻을 수 있으며 ROS_IP로 설정합니다.

④ 단축 명령 이동. ros 작업 공간에서 주요 파일 사이에 빠르게 이동할 수 있습니다.

⑤ 로봇 단축 명령 연결. ssh 명령, nfs 기능 로딩 및 삭제 명령을 포함하는데 사용 방법은 동영상 참조할 수 있습니다.

⑥ 터미널 디스플레이. 현재 환경에서 ROS_MASTER_URI와 ROS_IP는 터미널에 디스플레이 될 수 있습니다. 다음과 같이 유저는 두 IP를 통해 로봇이 같은 네트워크에 있는지를 판단할 수 있습니다.

```
# ROS_Robot Network Addrees
export ROBOT_IP=192.168.3.10
export ROS_MASTER_URI=http://$ROBOT_IP:11311

# alias_Pass Quickslot key
alias cw='cd ~/cilab/ros_ws'
alias cs='cd ~/cilab/ros_ws/src'
alias cm='cd ~/cilab/ros_ws && catkin_make'

# alias_Robot Connection Quickslot
alias nfsrobot='sudo mount -t nfs -o nolock $ROBOT_IP:/home/cilab/cilab /home/cilab/Robot'
alias unfsrobot='sudo umount -lf /home/cilab/Robot'
alias sshrobot='ssh -X cilab@$ROBOT_IP'

# BASH Terminal Display
echo ""
echo "-------------------- CILAB HOST PC INFO --------------------"
echo -e "  ROS_MASTER_URI: \033[32m$ROS_MASTER_URI\033[0m"
echo -e "  HOST PC ROS_IP: \033[32m$ROS_IP\033[0m"
echo "------------------------------------------------------------"
echo ""

#-------------CILAB HOSTPC CONFIG-------------------------------------
```

5.4. 로봇 구동 설명

전원 버튼을 누르면 로봇이 가동되며 버저가 울립니다. 가동 후 약 2초 정도 지나면 자이로스코프 캘리브레이션이 진행됩니다. 캘리브레이션 시 OpenCTR 푸른 불이 빠른 속도로 깜빡이게 됩니다. 캘리브레이션 시간은 0.5 초 정도 지속되며 캘리브레이션이 진행되는 동안 반드시 로봇을 정지시켜야 합니다. 그렇지 않으면 정확한 캘리브레이션 데이터를 얻을 수 없을 수 없습니다. 따라서 캘리브레이션 작업 시 로봇을 평평한 바닥에 위치해 두는 것을 권장합니다.

캘리브레이션이 완료된 뒤 본체가 가동될 때까지 LED가 10 초 정도 천천히 깜빡입니다. 가동이 완료되면 버저가 울리게 되고 이때 OpenCTR 소프트웨어가 모두 사용 가능한 상태로 전환되게 됩니다.

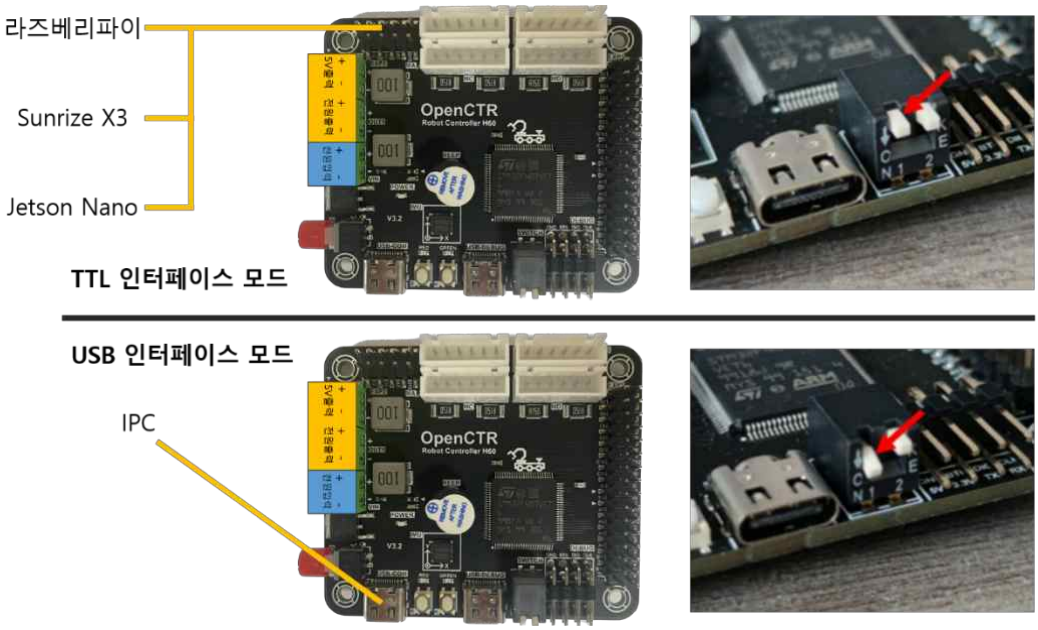
5~10 초 정도 경과하면 ROS 메인컨트롤러의 Ubuntu 시스템도 가동될 수 있습니다.

5.5. 모드 선택 설명

OpenCTR의 다이얼 스위치를 통해 다른 작업 모드를 선택할 수 있습니다.

5.5.1. 시리얼 인터페이스 출력 선택

OpenCTR 컨트롤러는 두 가지 방식으로 ROS 메인컨트롤러와 연결할 수 있습니다. 첫 번째 방식은 EXP2 확장 인터페이스의 TTL 시리얼 인터페이스를 사용하는 것인데, UART2, 라즈베리파이, SUNRIZE X3, Nano도 기본적으로 이 방식을 사용합니다. 전선 하나만으로 전원 공급 및 통신할 수 있습니다.



OpenCTR는 시리얼 인터페이스로 ROS 메인컨트롤러에 데이터를 내보내며 인코더 스위치 1을 통해 전환하게 됩니다. 스위치는 위에서 EXP2 인터페이스로 데이터를 전송합니다.

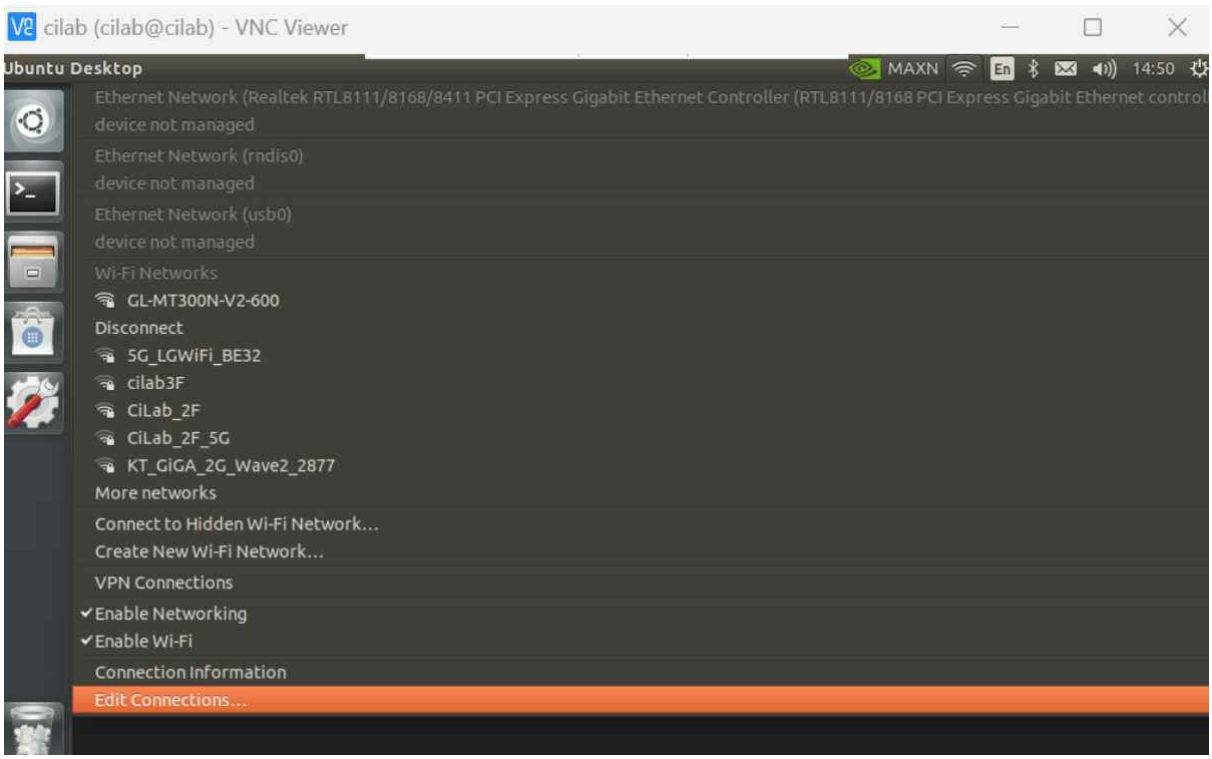
5.6. 로봇 네트워크 연결

모든 ROS 메인 컨트롤러는 와이파이 기능을 내포하고 있으며 와이파이를 통해 LAN에 연결됩니다.

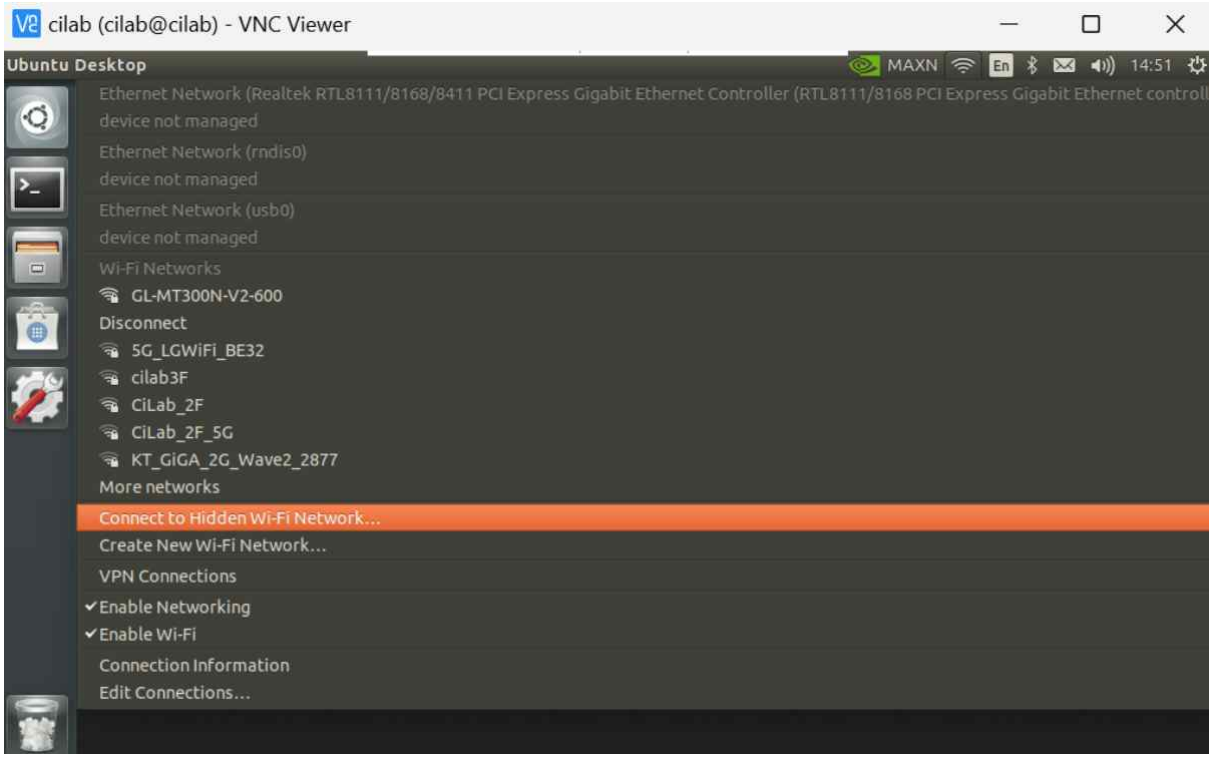
로봇과 본체는 하나의 네트워크 세그먼트, 즉 동일한 라우터에 연결되어야 하며 만일 그렇지 않을 경우 동일한 LAN으로 출력이 되더라도 실질적으로는 로봇과 프로그램이 통신할 수 없습니다.

로봇이 와이파이 네트워크에 연결하는 방법은 일반 컴퓨터와 동일하며, 화면을 통해 컨트롤할 수 있습니다. 만약 디스플레이의 사용이 불가한 경우에는 HDMI 케이블을 이용해 모니터에 연결하여 컨트롤할 수 있습니다. 로봇에는 Micro HDMI 전환 포트가 탑재되어 있으며 유저는 아래 이미지와 같이 일반 HDMI 케이블을 연결해 모니터에 연결할 수 있습니다.

CiLab X3을 구동한 후 Ubuntu 바탕 화면이 나오면 연결된 키보드나 터치스크린을 통해 컨트롤할 수 있습니다.



와이파이가 연결된 상태에서 터미널이나 ifconfig 명령을 통해서 로봇의 IP 주소를 확인할 수 있습니다. 해당 IP 주소는 가상 머신을 연결해줄 때도 동일합니다.



이 주소는 DHCP를 통해 자동으로 얻어진 것으로 인터넷 환경에 따라 바뀔 수도 있습니다. 바뀌게 되면 다시 연결하여 해당 절차대로 다시 확인해야 하는 번거로움이 있을 수 있으며 IP 주소가 바뀌지 않기를 원하면 고정 IP 주소를 설정하는 방법도 있습니다. 구체적인 설정 방법은 다음과 같습니다.

차량 AP 명 : CILAB-X3

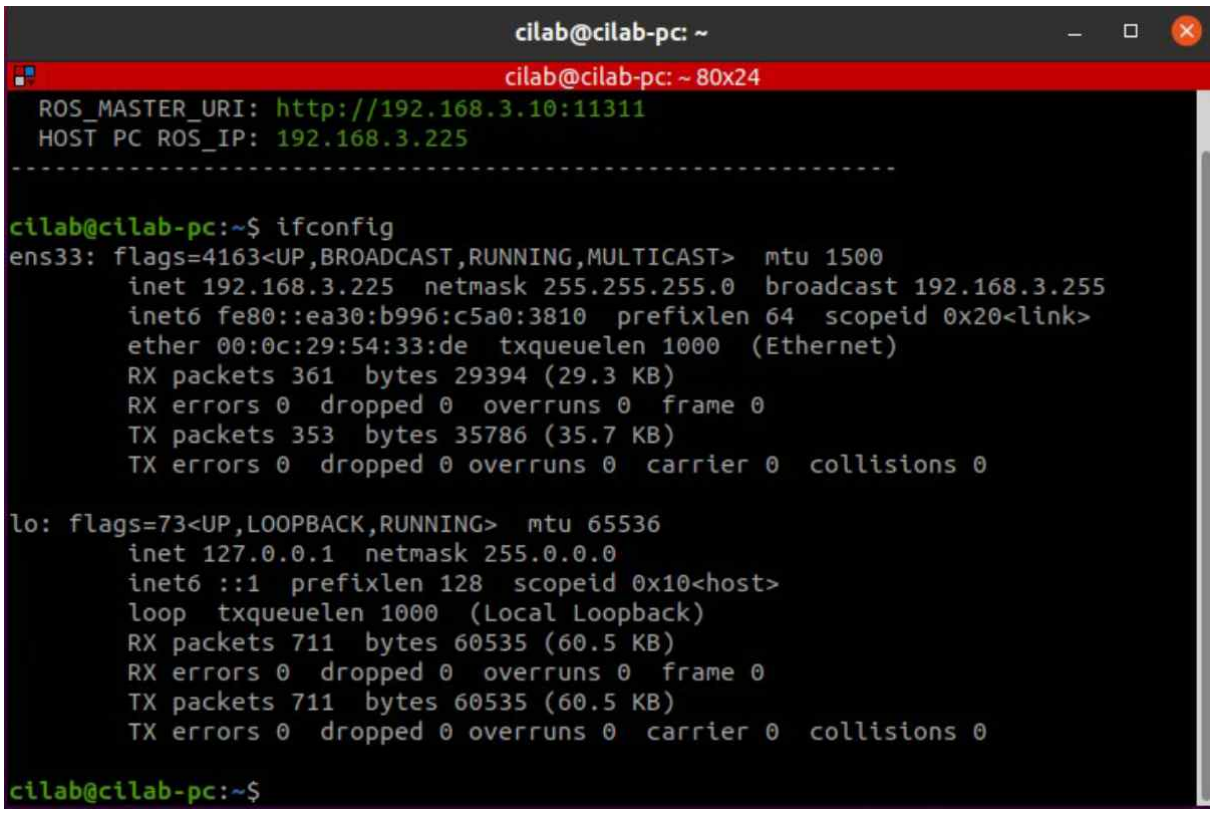
PC 네트워크 연결 비밀번호 : cilabrobot

차량과 본체는 하나의 네트워크 세그먼트, 즉 동일한 네트워크에 연결되어야 하며, 만일 그렇지 않을 경우 동일한 LAN으로 출력은 되더라도 실질적으로는 로봇과 프로그램이 통신할 수 없습니다.

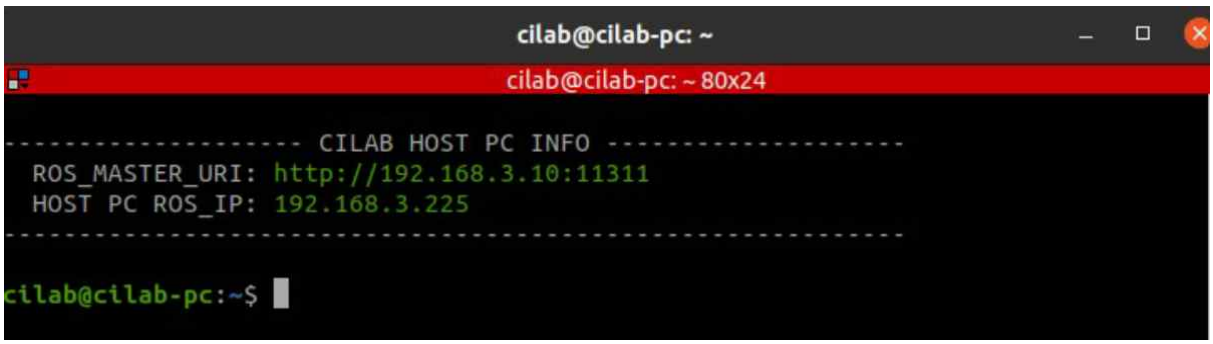
5.6.1. 본체와 로봇 연결

본체 IP 주소와 로봇 IP 주소를 파악하여 두 주소가 하나의 네트워크 세그먼트에 있는지 확인합니다. 일반적으로 LAN은 IP 주소의 앞쪽 세 자리만 확인하면 되는데, 동일하게 출력된는 것을 확인함으로써 하나의 LAN에 위치하고 있음을 알 수 있습니다. 그렇지 않으면 본체 또는 로봇의 네트워크를 설정하여 하나의 네트워크 세그먼트에 있게 해야 합니다.

먼저 gedit으로 가상 본체에 있는 .bashrc 파일의 ROBOT_IP 파라미터를 실제로 파악한 로봇 IP 주소로 변경합니다. 변경을 저장하고 닫아주면 프로그램에서 환경 파라미터 ROBOT_IP를 통해 로봇의 IP 주소를 얻을 수 있게 됩니다.



이때 본체에서 터미널을 실행하면 이봇과 본체의 IP 주소를 볼 수 있으며, 같은 네트워크 세그먼트 내에 있는지 다시 한 번 확인합니다.

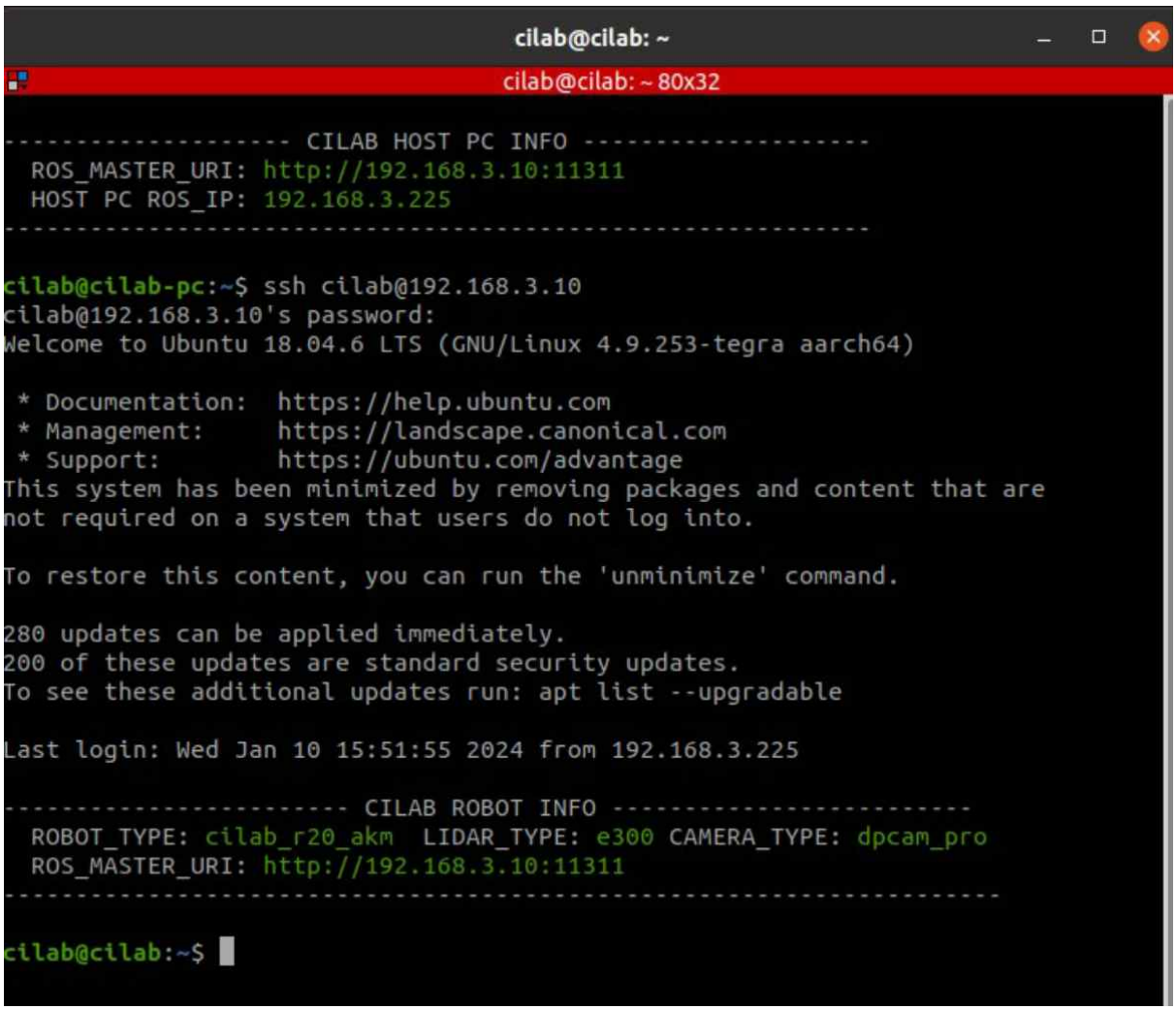


ssh 명령을 통해 로봇에 접속할 수 있습니다. 알림에 따라 로봇 비밀번호(CiLab)를 입력하면 로봇에 로그인할 수 있으며, 로봇 본체의 시스템 정보도 볼 수 있습니다.

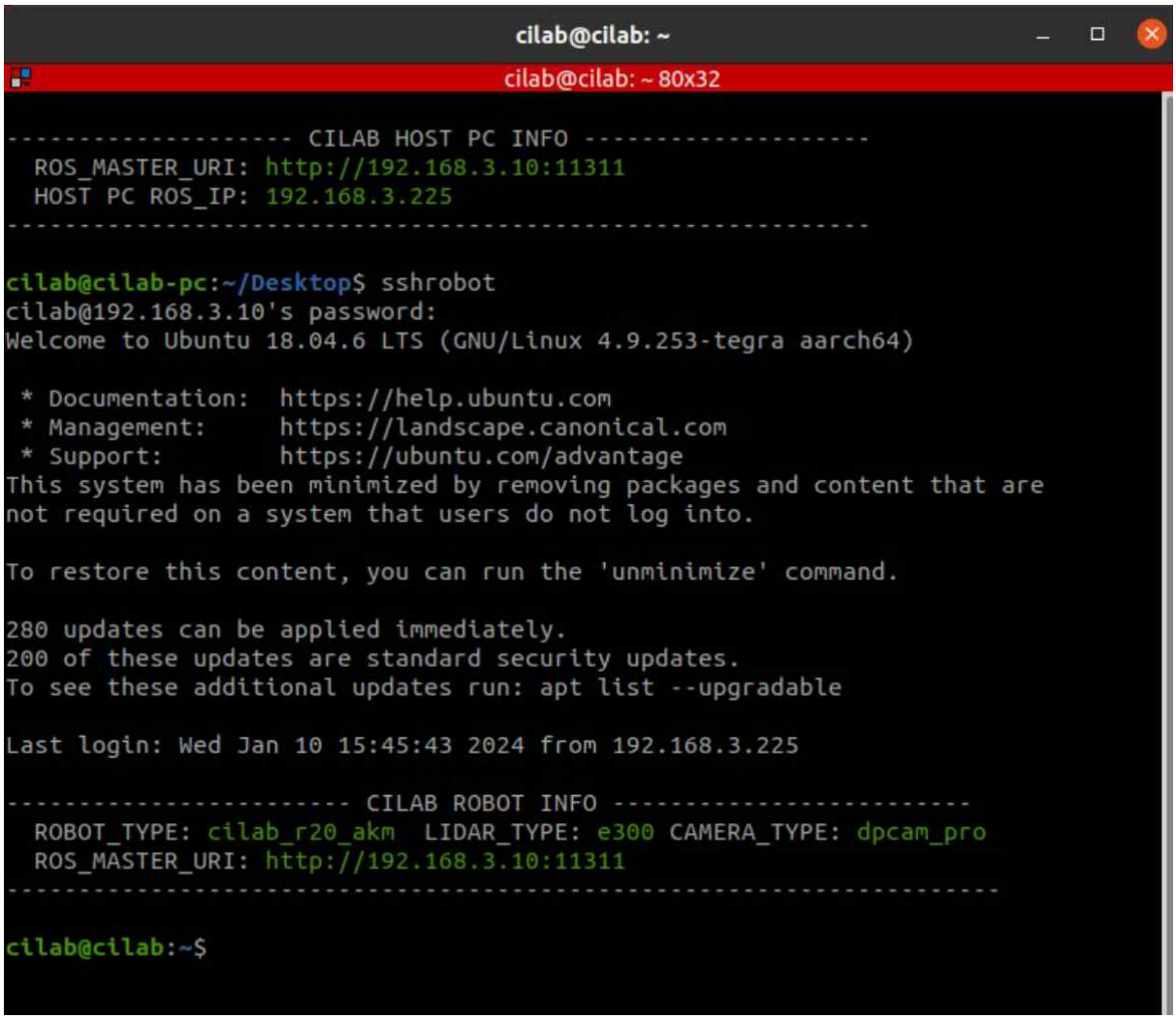
CILAB R 차량에 출력되는 IP : 192.168.1.164

cilab@cilab-pc:~$ ssh cilab@192.168.1.164

password : cilab



ssh 명령 입력은 비교적 길기 때문에 편의를 위해 sshrobot 명령을 사용할 수 있습니다. sshrobot 명령을 입력해도 로봇에 로그인할 수 있습니다. 이 명령은 .bashrc 파일에서 alias 명령을 통한 매핑이 설정되어 있습니다.

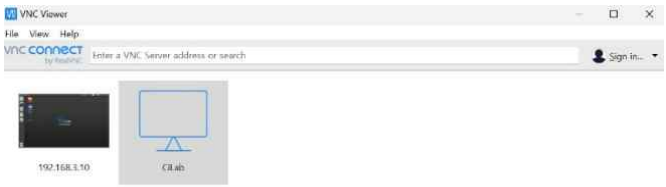


본체 ssh가 로봇에 연결되면 명령을 통해 로봇을 조정할 수 있습니다. 대부분의 ROS 기능은 주어진 명령어를 통해 작동됩니다.

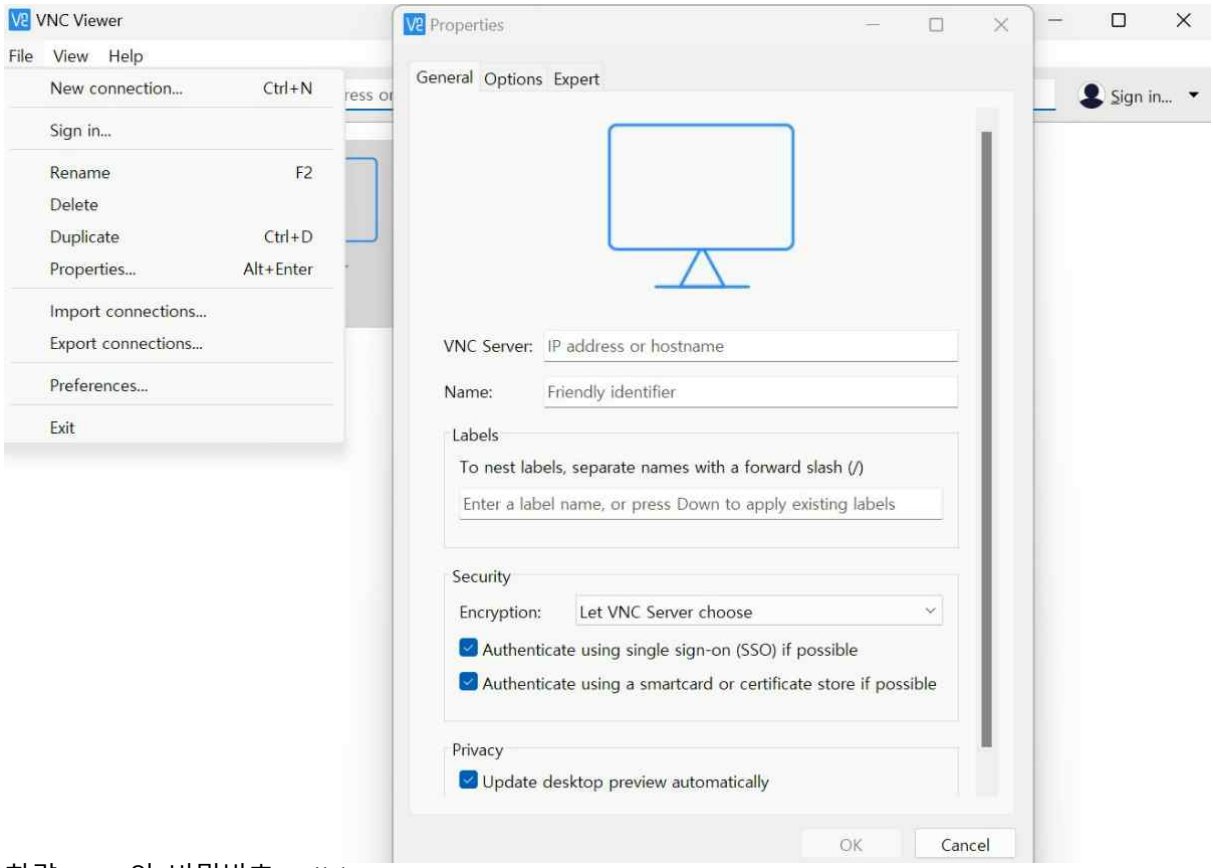
로봇 파일을 읽고 편집하는 기능을 용이하게 하기 위해 로봇에는 NFS 인터넷 파일 서비스 시스템이 설치되어 있습니다. 로봇 바탕화면을 쉽게 컨트롤할 수 있도록 VNC 원격 바탕화면 서비스가 구동 가능합니다.

VNC 원격 바탕화면 기능

로봇은 VNC 원격 데스크탑 기능을 실행하여 Windows 버전의 VNC Viewer 또는 MobaXterm 소프트웨어를 통한 원격 연결을 할 수 있습니다. 연결 방법은 VNC Viewer 소프트웨어를 예로 들어 설명합니다.

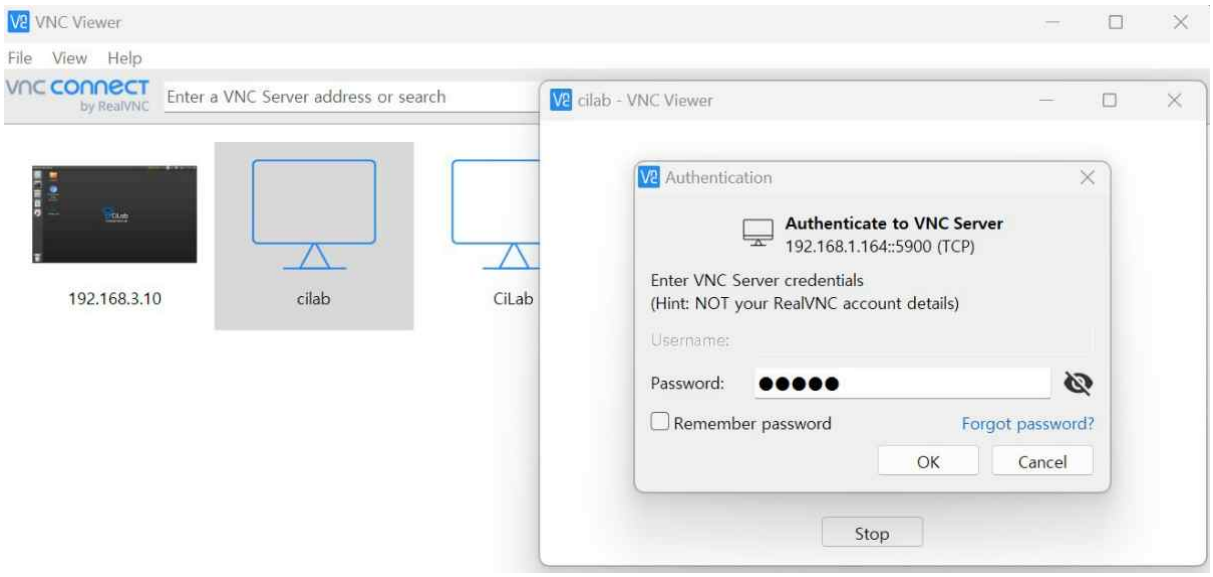


원격 기능을 작동하기 전에 로봇이 개발된 로컬 네트워크에 연결되어 있는지, PC와 로봇이 같은 LAN에 있는지를 확인합니다.



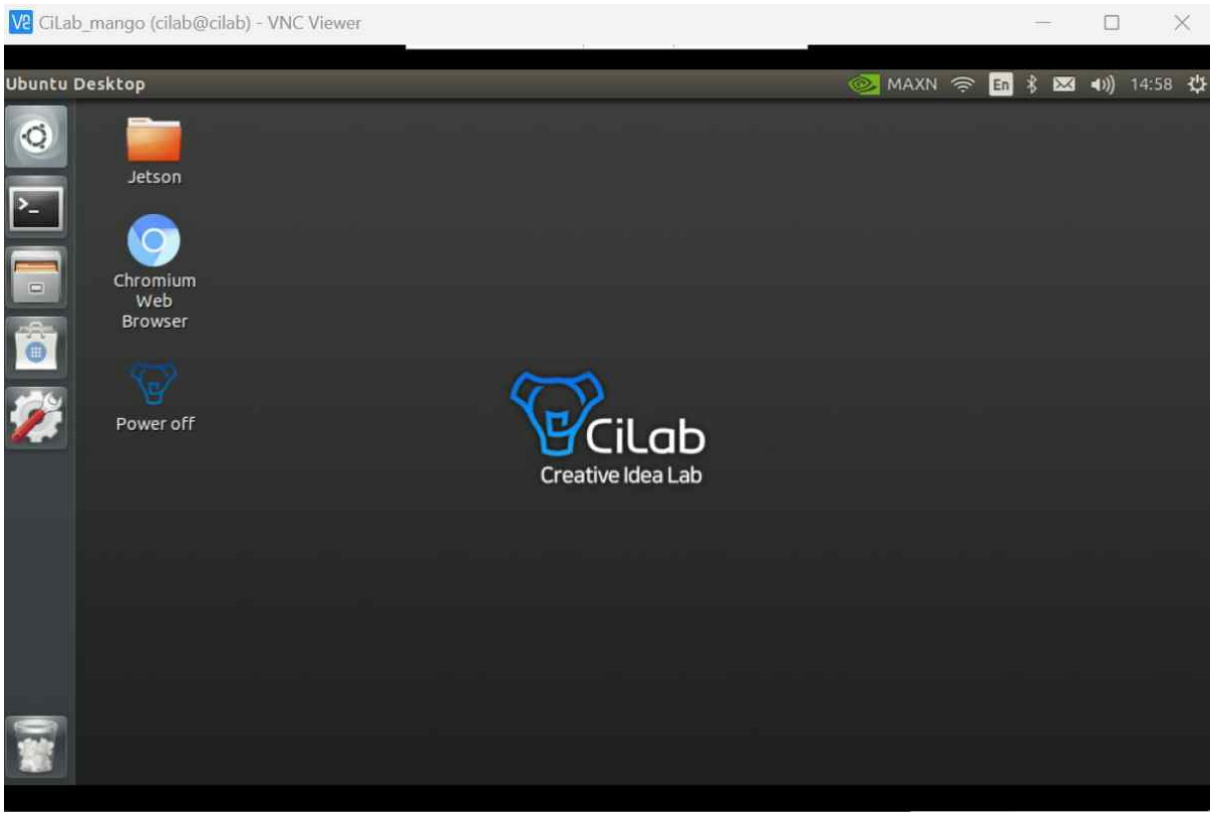
차량 VNC의 비밀번호 : cilab

먼저 VNC 소프트웨어를 시작하여 새 연결을 만들고 IP 주소와 VNC 본체 이름(유저 정의)을 입력한 다음에 OK 버튼을 클릭합니다. 다음으로 새로 만든 연결을 선택하고 로봇 VNC 암호를 입력하면 로봇 원격 바탕화면에 연결할 수 있습니다.



연결된 뒤 로봇 바탕화면은 아래 이미지와 같습니다. 마우스와 키보드로는 로봇 바탕화면을 직접 컨트롤할 수 있으며 바탕화면 디폴드 해상도는 720P로 화면이 선명하고 컨트롤이 수월하다는 장점이 있습니다.

VNC 접속 비밀번호 : cilab

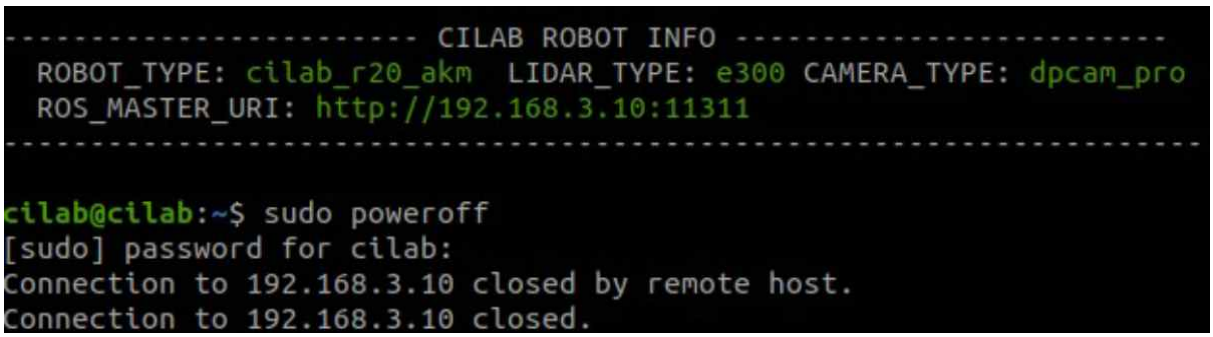


위와 같은 작업을 통해 유저 PC의 VNC 프로그램 화면에 정상적으로 접속할 수 있게 됩니다.

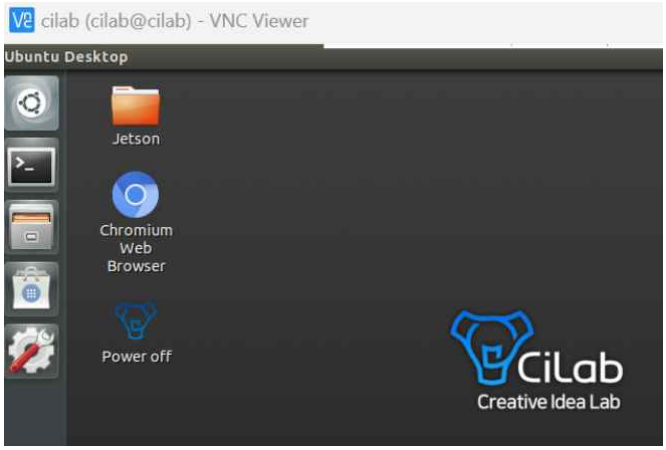
5.7. 로봇 종료 방법

로봇을 사용 종료시 바로 전원을 끊는 유저가 많은데, 일반적으로 시스템에 이상이 없겠지만 이것은 결코 안전한 전원오프 방법이 아닙니다. 유저의 원활한 사용을 위해서는 안전한 종료를 권장합니다.

본체 측 SSH는 원격으로 로봇에 연결하여 sudo power off 명령을 실행하면 Ubuntu 시스템을 종료할 수 있습니다. (혹은 바탕화면에 CiLab 아이콘을 통해 해당 명령을 대신할 수 있습니다.)

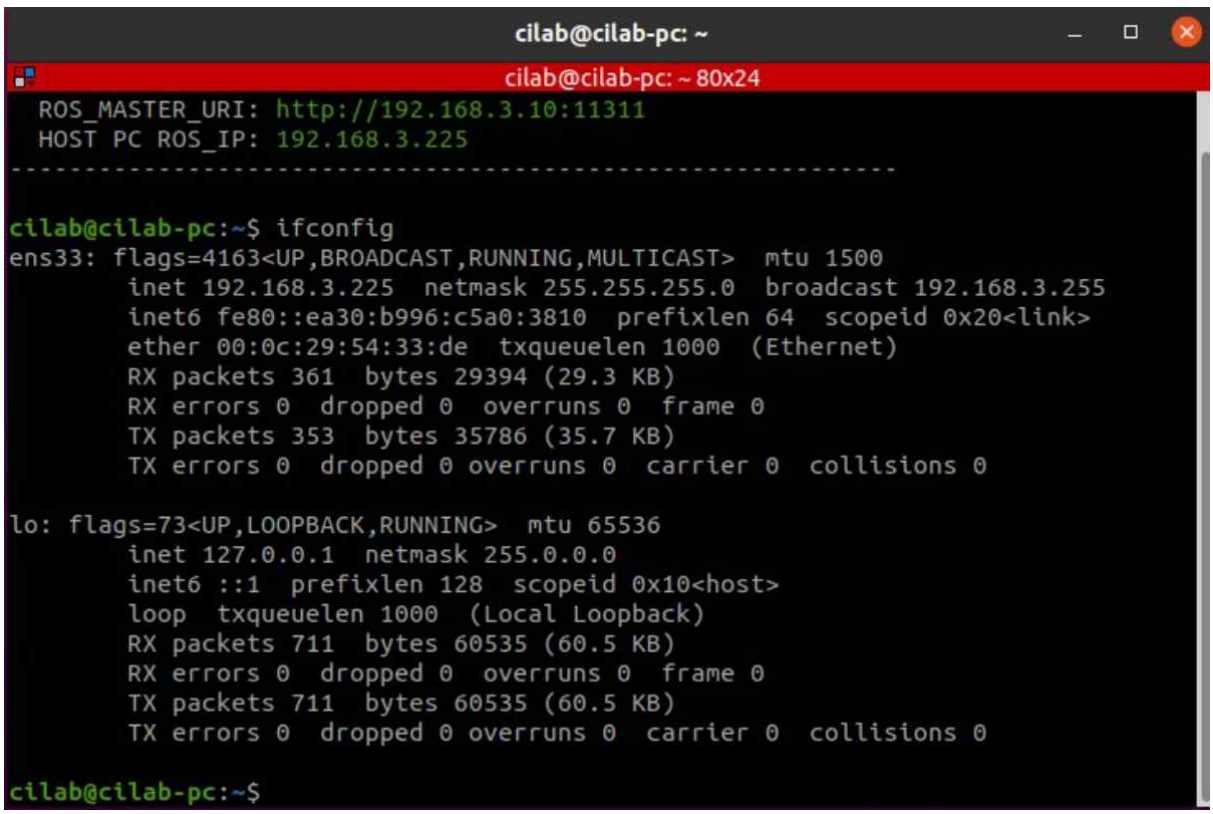


시스템을 끄는 동안 라즈베리파이는 빨간불이 먼저 꺼지고 초록불이 깜박이게됩니다. 초록불이 꺼지고 나서 빨간불이 다시 켜지면 시스템이 종료됩니다. 이때 시스템 전원 스위치를 오프해야 안전하게 종료할 수 있습니다.



5.8. 로봇 구성 설명

X3 시리즈 로봇은 다음과 같이 정의됩니다.



본체 측 ssh는 원격으로 로봇에 연결되며 gedit 이미지 편집기를 사용하여 .bashrc 파일을 실행하여 수정할 수 있습니다. nano와 vim 텍스트 편집기로 파일을 열어 취향에 따라 선택 수정할 수

도 있습니다.

cilab@cilab-robot:~$ gedit .bashrc

```
#-------------CILAB HOSTPC CONFIG-------------------------------------
# ROS_Setting Environment Variables
source /opt/ros/noetic/setup.bash
source ~/cilab/ros_ws/devel/setup.bash

# ROS2_Setting Environment Variables
#source /opt/ros/foxy/setup.bash
#source ~/cilab/ros2_ws/install/setup.bash

# ROS_Setting Main PC Network
interface=ens33
export IPAddress=`ifconfig $interface | grep -o 'inet [^ ]*' | cut -d " " -f2`
export ROS_IP=$IPAddress

# ROS_Robot Network Addrees
export ROBOT_IP=192.168.3.10
export ROS_MASTER_URI=http://$ROBOT_IP:11311

# alias_Pass Quickslot key
alias cw='cd ~/cilab/ros_ws'
alias cs='cd ~/cilab/ros_ws/src'
alias cm='cd ~/cilab/ros_ws && catkin_make'

# alias_Robot Connection Quickslot
alias nfsrobot='sudo mount -t nfs -o nolock $ROBOT_IP:/home/cilab/cilab /home/cilab/Robot'
alias unfsrobot='sudo umount -lf /home/cilab/Robot'
alias sshrobot='ssh -X cilab@$ROBOT_IP'

# BASH Terminal Display
echo ""
echo "-------------------- CILAB HOST PC INFO --------------------"
echo -e "  ROS_MASTER_URI: \033[32m$ROS_MASTER_URI\033[0m"
echo -e "  HOST PC ROS_IP: \033[32m$ROS_IP\033[0m"
echo "------------------------------------------------------------"
echo ""

#-------------CILAB HOSTPC CONFIG-------------------------------------
```

이 중 ①⑤⑥⑦은 본체 구성과 유사하므로 여기에서는 더 이상 설명하지 않습니다.

② 로봇 모델 선택은 본인의 로봇 모델에 따라 대응하는 모델을 선택할 수 있습니다.

③ 라이다 모델의 선택은 자신의 로봇 라이다 모델에 따라 대응하는 라이다 모델을 선택할 수 있습니다.

④ 라이다 모델 선택은 자신의 로봇 카메라 모델에 따라 대응하는 카메라 모델을 선택할 수 있습니다.

로봇은 출고 시 이미 해당 데이터가 수정되었으며 유저가 미러링 파일을 교체하거나 다시 복사하였다면 스스로 해당 모델로 수정해야 합니다.

6. 로봇 조작 가이드

6.1. 로봇 구동하기

먼저 ssh 명령어를 입력해 로봇으로 원격으로 연결해 아래 launch파일을 실행합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

해당 드라이버 파일집은 로봇 구동 베이스 단계에 해당하며, ROS 휴머노이드와 하부 제어 로봇 간의 불일치를 예방하기 위해 휴머노이드 검사 기능이 추가되었습니다.

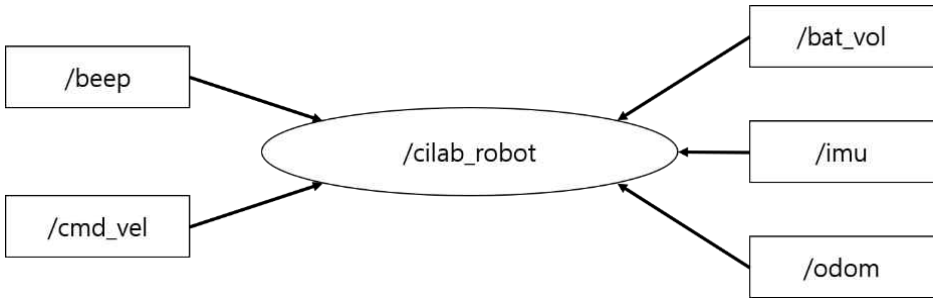
ROS 드라이브 패키지를 부팅하게 되면 로봇 기반 컨트롤러 OpenCTR에서 로봇 유형 매개변수를 보내게 됩니다.

일치할 경우 부저에서 짧은 알림음(Beep)이 나게되며, 일치하지 않을 경우 긴 알림음(Beeeeep)이 나게됩니다.

만일 이러한 상황이 발생한 경우에는 OpenCTR 디스플레이와 로봇에서의 연결을 확인해야 합니다. bahsrc 파일에 기재되어 있는 파일과 일치하는지 확인할 필요가 있습니다.

부팅이 되었다면 가상 프로그램 호스트 포트(cilab-pc)에서 다음 그림과 같이 rqt_graph 도구를 통해 드라이버 토픽 게시 상황을 열람할 수 있습니다.

그 중 /cmd_vel은 속도, /odom은 오도메트리 마일리지, /imu는 IMU 가속 자이로 센서, /bat_vol은 배터리 전압입니다.



각각의 토픽에 대한 정의는 아래와 같습니다.

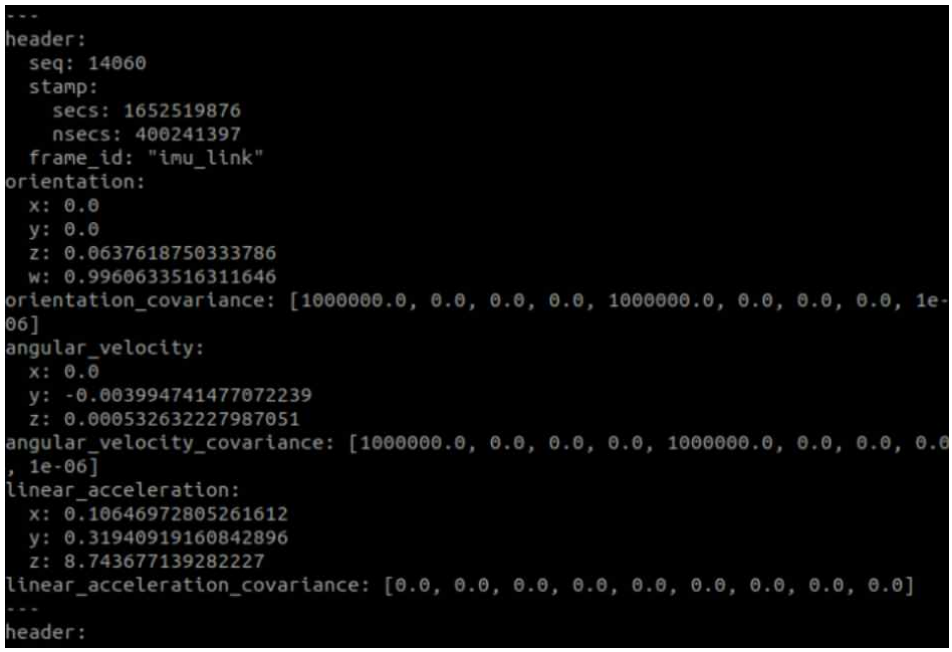
Topic	구독/발행	상세설명
/cmd_vel	구독	로봇에서 확인되는 속도 토픽
/beep	구독	로봇에서 확인되는 알림음 토픽
/odom	발행	로봇에서 송출하는 미터기 토픽
/imu	발행	로봇에서 송출하는 IMU 자이로스코프 센서 토픽
/bat_vol	발행	로봇에서 송출하는 배터리 전압 토픽

로봇 혹은 가상 프로그램에서 rostopic명령어를 통해 송출 드라이버의 토픽 데이터를 확인할 수 있습니다.

IMU토픽을 예로 들어 설명합니다.

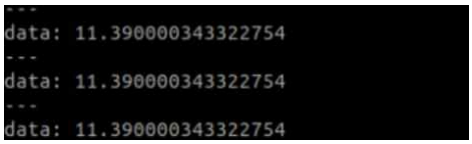
cilab@cilab:~$ rostopic echo /imu

IMU토픽패키지의 경우 3축 가속도데이터와 3축 각속도를 비롯한 4가지의 데이터(x,y,z,w)를 포함하고 있으며, 아래 이미지를 통해서 확인할 수 있습니다.



배터리 전압의 경우 /bat_vol 명령어를 통해 확인 가능합니다.

cilab@cilab:~$ rostopic echo /bat_vol



6.2. 속도제어

구동 드라이버에서는 /cmd_vel명령어를 통해 표준 세팅 속도 데이터에 연결 가능합니다.
본 챕터에서는 /cmd_vel토픽 제어를 통해 로봇을 조작하는 방법에 대해 설명합니다.

먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

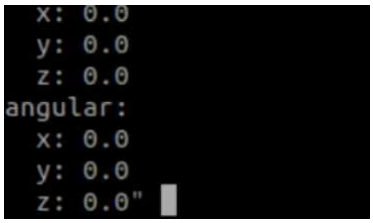
그리고 새로운 터미널(cmd)을 생성해 위와 동일하게 ssh명령어로 로봇에 연결한 후에 속도 토

픽을 수정하면 차량은 우리가 설정한 속도를 표준으로 하여 주행을 하게됩니다.

cilab@cilab:~$ rostopic pub /cmd_vel

실제 동작시 위 명령어를 입력하게 되면 tab 키를 입력해주면 자동으로 해당하는 명령어가 입력되게 됩니다. ex. rost → [tab] → rostopic) 속도 데이터를 수정할 때에는 키보드의 ↑↓버튼으로 수정할 수 없으며 반드시 ←→버튼만으로 변환이 가능합니다.

cilab@cilab:~$ rostopic pub /cmd_vel geometry_msgs/Twist "linear:



위의 예시와 같이 해당 토픽에서는 linear & angular 2개의 내용이 확인 가능합니다.

linear는 선형속도 즉 직선 구간 속도를 의미하며, x는 앞 뒤, y는 좌 우, z는 위 아래를 나타내게 됩니다. (CiLab R의 경우에는 RWD 모델로 y,z의 값은 무의미합니다.)

angular는 각 속도를 의미하며 현재단계에서는 z축값만으로 해당 각을 컨트롤할 수 있습니다.

토픽을 통한 컨트롤은 프로그램 제어에 주로 사용되며, 수동컨트롤로 제어하기에는 다소 까다로울 수 있습니다.

6.2.1. 알림음 제어

로봇의 전원 자체적으로 부저의 알림을 제어할 수 있도록 설계되어 있으며 토픽을 사용하여 제어할 수 있습니다.

먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

그리고 새로운 터미널(cmd)을 생성해 위와 동일하게 ssh명령어로 로봇에 연결한 후에 알림음 토픽을 실행하여 해당 기능을 수정할 수 있습니다.

다시 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ rostopic pub /beep std_msgs/Int8 "data: 1"

그리고 아래 명령어를 통해 알림음을 온오프 설정이 가능합니다.

```
cilab@cilab:~$ rostopic pub /beep std_msgs/Int8 "data: 1"
publishing and latching message. Press ctrl-C to terminate
```

cilab@cilab:~$ rostopic pub /beep std_msgs/Int8 “data: 0”

```
cilab@cilab:~$ rostopic pub /beep std_msgs/Int8 "data: 0"
publishing and latching message. Press ctrl-C to terminate
```

6.3. 전조등 제어

차량 전방의 RGB 전조등 퍼포먼스는 조이스틱, ROS Service, rqt_reconfigure 그래픽 인터페이스를 통해 제어할 수 있습니다.

ROS에서 직접 제어하기 (ROS Service)

먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

그리고 새로운 터미널(cmd)을 생성해 위와 동일하게 ssh명령어로 로봇에 연결한 후에 아래 명령어를 통해 전조등 관련 파라미터를 수정할 수 있습니다.

cilab@cilab:~$ rosservice call /Light_Server “{RGB_M_:1, RGB_S_:0, RGB_T_:0, RGB_R_:255, RGB_G_:0, RGB_B_:0,}”

해당 서비스 메시지는 사용자 정의를 의미합니다.

RGB_M: 1-10 범위의 조명 제어 마스터 모드
RGB_S: 조명 슬레이브 모드 (범위 1~10)

RGB_T: 시간(time) 매개변수 (범위 0~255)
RGB_R, G, B : 각각 붉은색, 초록색, 파란색의 세 가지 색상 값 (범위: 0~255)

주의 : 현재 로봇 예제코드에는 메인 6가지의 모드가 내포되어 있으나 해당 모델에서는 모드나 시간 매개변수는 아직 개발단계로 추후 개발 로봇에 새로운 모드가 적용될 예정입니다.

rqt_reconfigure 이미지화 도구 컨트롤

위에서 설명한 서비스 제어 외에 ROS의 rqt_reconfigure gui도구를 통해서도 이를 제어할 수 있습니다.

먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

그리고 새로운 터미널(cmd)을 생성해 위와 동일하게 ssh명령어로 로봇에 연결한 후에 아래 명령어를 통해 rqt_reconfigure 도구를 실행합니다.

cilab@cilab:~$rosrun rqt_reconfigure rqt_reconfigure

팝업 페이지를 보면 각 변수가 위에서 설명한 ROS 서비스 메시지와 일치하도록 정의되어 있습니다.

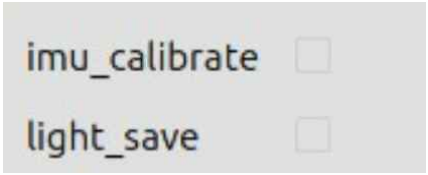
현 시점에서 유저가 선택 및 조절가능한 내용은 아래와 같습니다.

RGB_M : Master mode 1~6
RGB_S : 현재 개발 단계에 있습니다.
RGB_T : 현재 개발 단계에 있습니다.
RGB_R,G,B : 각각 빨간색(red), 녹색(green), 파란색(blue)을 의미
0-255의 범위 내에서 포인터를 직접 이동시켜 변경 가능

해당 값을 수정하면서 실시간으로 퍼포먼스를 관찰할 수 있습니다.

주의 : 모드 2는 변조 모드이며, 해당 모드에서 모든 색상이 절반 미만으로 출력이 되면 램프가 더 이상 켜지지 않습니다.

로봇의 전원을 켰을때 원하는 조명 모드로 직접 변경하려면 디버깅된 조명 매개변수를 하위 OpenCTR 매개변수에도 동일하게 변경해주어야 합니다. rqt_reconfigure를 선택하여 다음 구성 매개변수 항목을 변경하는 과정을 통해 이를 제어할 수 있습니다.



light_save 체크박스를 체크한 후 Open_CTR 컨트롤러는 EEPROM의 매개변수를 업데이트하게 되면 변경 완료 알림이 울립니다. 이후 로봇이 전원을 켜면 자동으로 설정된 모드의 색상이 들어오게 됩니다.

6.4. IMU 캘리브레이션

로봇 IMU 센서의 자이로스코프는 영점을 기준으로 동작하며, 로봇의 장시간 움직임에 따라 해당 영점 조절 기능의 오차가 커지게 됩니다. 앞서 소개한 바와 동일하게 시스템을 켜면 자동으로 조절 동작을 수행하게되고, 사용 중 자이로스코프가 동작하면 OpenCTR 제어 리셋 버튼을 통해 영점 조절 캘리브레이션을 다시 수행하거나 rqt_reconfigure 도구를 사용하여 직접 조절할 수도 있습니다.

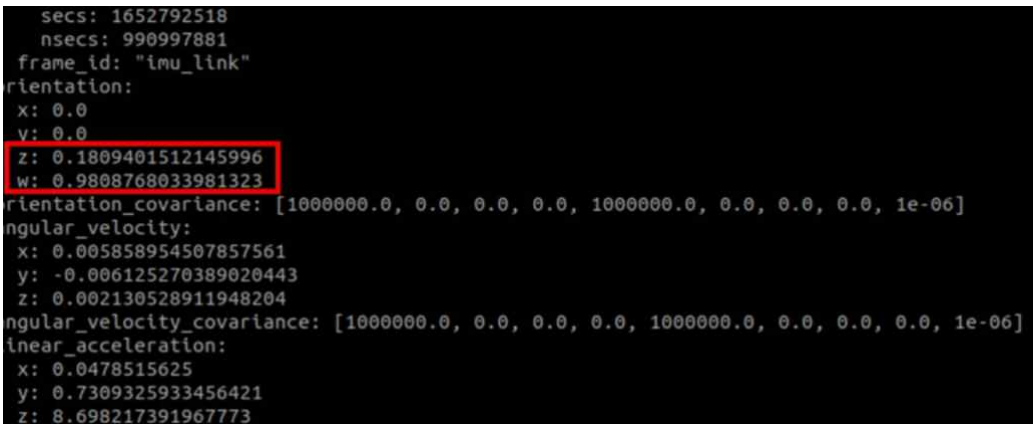
먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

그리고 새로운 터미널(cmd)을 생성해 위와 동일하게 ssh명령어로 로봇에 연결한 후에 아래 명령어를 통해 IMU 데이터 토픽을 확인합니다.

cilab@cilab:~$ rostopic echo /imu

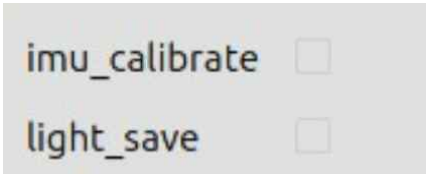
해당 출력되는 데이터에서 아래 박스로 표시된 부분을 주의합니다.



해당 매개변수는 로봇의 자세 방향 정보를 담당하며 z 또는 w가 급격히 변화하는 것이 확인되면(소수점 세 자리부터 빠르게 변화함), 캘리브레이션 과정이 필요하다는 것을 의미합니다. 방법은 비교적 간단하며 절차는 다음과 같습니다.

cilab@cilab-pc:~$ rosrun rqt_reconfigure rqt_reconfigure

인터페이스에서 아래와 같은 팝업창이 나타나면 왼쪽 상단의 cilab_robot을 클릭하면 오른쪽에 옵션바가 나타납니다.



imu_calibrate는 IMU의 캘리브레이션(보정) 기능 버튼으로, 체크하면 IMU 캘리브레이션 프로그램을 실행하고 부저에서 알림음이 두 차례 나게되면 해당 작업이 정상적으로 진행되었음을 의미합니다. 그리고 IMU 데이터는 정상 상태로 돌아가게 됩니다.

주의, 캘리브레이션 과정시 반드시 로봇을 수평 상태로 유지해야합니다.

7. 원격 컨트롤 가이드

구동 기능 패키지가 설치되어 있는 경우 cmd_vel 속도 토픽을 통해 로봇의 움직임을 제어할 수 있습니다.

cilab_teleop 기능 패키지는 키보드, 조이스틱 또는 USB 핸들로 로봇을 제어할 수 있습니다. 로봇 또는 가상 프로그램에서 이를 제어할 수 있습니다. 본 챕터에서는 주로 키보드 제어 방법을 소개합니다.

추가 옵션인 조이스틱 및 USB 핸들 제어는 매뉴얼 2페이지의 QR코드를 통해 연락주시면 상세한 설명이 가능합니다.

7.1. 기능 패키지 설명

원격 제어 기능 패키지 명칭은 cilab_teleop이며, 기능 패키지의 파일은 다음과 같습니다.

7.2. 원격 컨트롤

키보드로 로봇을 제어하여 로봇 또는 가상 프로그램에서 동작할 수 있습니다.

키보드 컨트롤

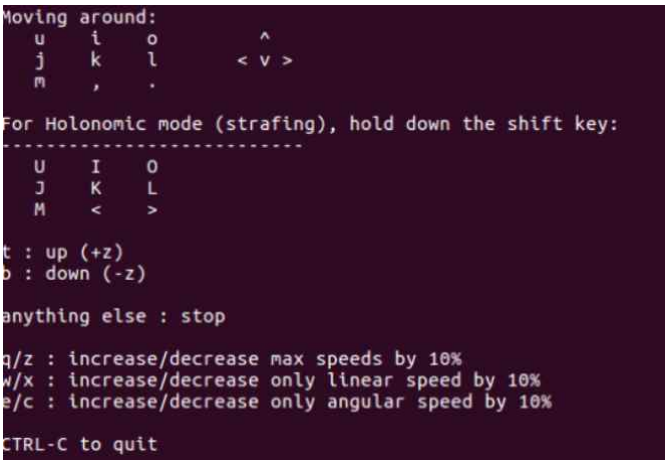
먼저 ssh 명령어를 이용하여 로봇에 연결합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

새로운 터미널을 생성해 ssh 명령어를 이용해 로봇과 연결합니다.

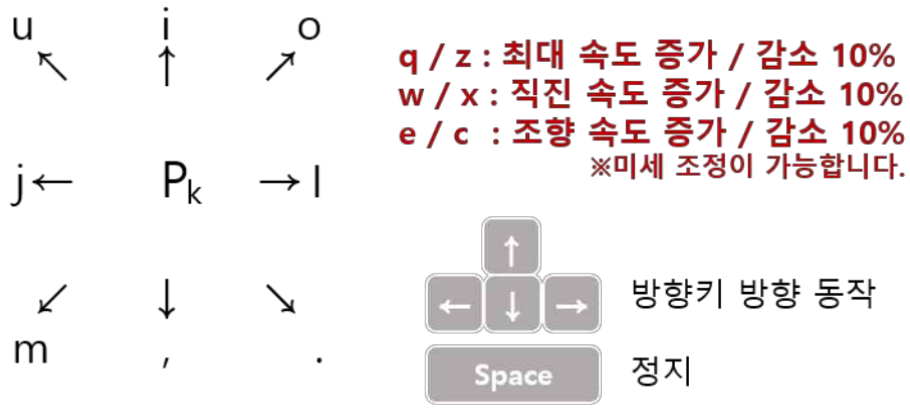
cilab@cilab:~$ roslaunch cilab_teleop keyboard.launch

위 명령어를 입력하면 아래와 같은 창이 나타나게 되고 해당하는 버튼을 눌러 동작할 수 있습니다.



키보드 컨트롤시 동작 키는 아래 설명과 같습니다.

키보드 컨트롤의 상세 설명은 아래와 같습니다.



가상 프로그램에서 키보드 컨트롤

키보드뿐만 아닌 가상 머신에서도 키보드를 사용한 컨트롤이 가능합니다.

먼저 SSH 명령을 통해 로봇에 연결하고 로봇 섀시 구동 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

새로운 터미널을 생성해 가상 프로그램 cmd에서 아래 명령어를 입력합니다.

cilab@cilab-pc:~$ roslaunch cilab_teleop keyboard.launch

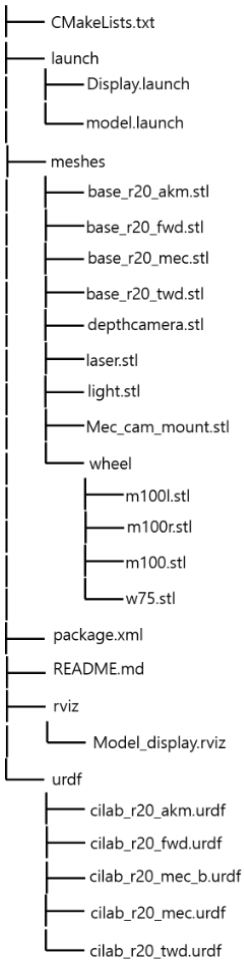
이하 동작에 대한 설명은 상기 키보드와 동일하기에 추가 설명은 생략합니다.

8. URDF 모델링

URDF 모델은 rviz gui에서 로봇 3차원 모델을 디스플레이할 수 있습니다.

8.1. 기능 패키지 설명

로봇 URDF 모델 기능 패키지에는 cilab_description으로 여러 모델의 차량이 포함되어 있습니다. 부팅 시 로봇에 연결된 환경 변수에 따라 자동으로 자체 모델 옵션이 선택됩니다.
X3의 기능 패키지에 있는 파일은 아래와 같습니다.



가상 머신 측 URDF 모델 표시 기능을 구현하려면 가상 머신 측에도 동일한 모델 파일이 있어야 하며 cilab에서 제공하는 가상 프로그램은 기본적으로 X3 시리즈 URDF 모델 기능 패키지로 설정되어 있습니다.

8.2. 모델링 파일 확인 가이드

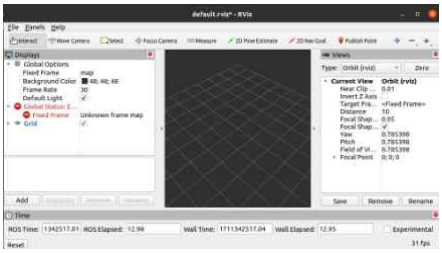
먼저 ssh 명령어로 로봇에 연결하고 model.launch 파일을 실행해 본인의 로봇 모델을 설정합니다. 기본 세팅은 cilab_X3_akm 모델입니다.

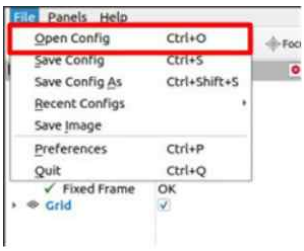
cilab@cilab:~$ roslaunch cilab_description model.launch robot_type:=X3_akm

가상 프로그램에서 아래 명령어로 rviz 프로그램을 실행하면 모델링 파일을 확인할 수 있습니다.

cilab@cilab-pc:~$ rosrun rviz rviz

아래 이미지와 같이 토픽을 추가해줍니다.





조이스틱이나 키보드로 차량의 이동 또한 가능합니다. 이동시 로봇 모션 상태는 rviz로 실시간 디스플레이되며 아래 이미지 예시를 통해 참고하실 수 있습니다.

로봇 model.launch파일은 아래와 같습니다. 아래 매개변수 중 akm에 해당하는 차량 모델이 디스플레이됩니다.

#로봇 모델명, 시리즈 및 유형 정보 포함
#유형 mec / fwd / twd / akm / tak

로봇 모델은 향후 지속적으로 업데이트될 예정입니다.

8.3. 모델링 파일 업그레이드

일반적으로 ROS 로봇의 모델은 cilab이 제공하는 가상 머신의 기본 모델인 X3 시리즈 URDF 모델 기능 패키지로 설정되어 출고됩니다. X3 시리즈 외 다른 시리즈의 로봇으로 디스플레이를 구현하기 위해서는 로봇의 URDF 모델을 가상 시스템에 동일하게 업데이트해주어야 합니다.

작동 방법은 다음과 같습니다. 먼저 가상 시스템 src 디렉토리에 들어가 기본 X3 시리즈 cilab_description 기능 패키지를 삭제합니다.

cilab@cilab-pc:~$ cd cilab/ros_ws/src

제거 기능인 rm 명령어를 이용해 URDF패키지를 삭제합니다.

cilab@cilab-pc:~/cilab/ros_ws/src $ rm -r cilab_description

그런 다음 ssh 명령어를 통해 로봇에 연결하고 scp 명령을 사용하여 로봇의 URDF 모델 패키지를 가상 시스템의 ~/cilab/ros_ws/src 디렉토리에 복사합니다.

주의 : IP 주소_가상 프로그램의 IP 주소

cilab@cilab:~$ scp -r cilab/ros_ws/src/cilab_description cilab@192.168.1.164:~/cilab/ros_ws/src

명령어 입력 후 나타나는 프롬프트는 yes를 입력한 후 가상 프로그램 비밀번호 cilab을 입력합니다.

```
cilab@cilab:~$ scp -r cilab/ros_ws/src/cilab_description cilab@192.168.3.10:~/ci
lab/ros_ws/src
The authenticity of host '192.168.3.10 (192.168.3.10)' can't be established.
ECDSA key fingerprint is SHA256:h46If4qiaC7b3AnM0oU0GKtFe+/qjEwtOA9/DXuTLAI.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '192.168.3.10' (ECDSA) to the list of known hosts.
cilab@192.168.3.10's password:
```

로봇의 URDF 모델을 가상 프로그램으로 업데이트할 수 있습니다.

```
model.launch                                  100%  573    232.7KB/s   00:00
display.launch                                100%  270    223.3KB/s   00:00
package.xml                                   100% 2581      2.5MB/s   00:00
CMakeLists.txt                                100% 7102      5.3MB/s   00:00
cilab_r20_mec.urdf                            100% 5894      4.5MB/s   00:00
cilab_r20_fwd.urdf                            100% 5890      4.5MB/s   00:00
cilab_r20_mec_b.urdf                          100% 6573      5.2MB/s   00:00
cilab_r20_akm.urdf                            100% 5890      4.0MB/s   00:00
cilab_r20_twd.urdf                            100% 4755      3.3MB/s   00:00
model_display.rviz                            100% 6179      4.9MB/s   00:00
base_r20_twd.stl                              100%  999KB   45.4MB/s   00:00
depthcamera.stl                               100%  103KB   27.3MB/s   00:00
base_r20_mec.stl                              100%  753KB   44.9MB/s   00:00
light.stl                                     100% 2684      2.8MB/s   00:00
m100r.stl                                     100% 2992KB   54.2MB/s   00:00
w100.stl                                      100%  123KB   24.6MB/s   00:00
w75.stl                                       100%  139KB   35.1MB/s   00:00
m100l.stl                                     100% 2980KB   55.2MB/s   00:00
base_r20_akm.stl                              100% 1708KB   50.1MB/s   00:00
laser.stl                                     100%   62KB   21.0MB/s   00:00
base_r20_fwd.stl                              100%  753KB   45.9MB/s   00:00
mec_cam_mount.stl                             100%   40KB   17.4MB/s   00:00
```

9. 구동 패키지

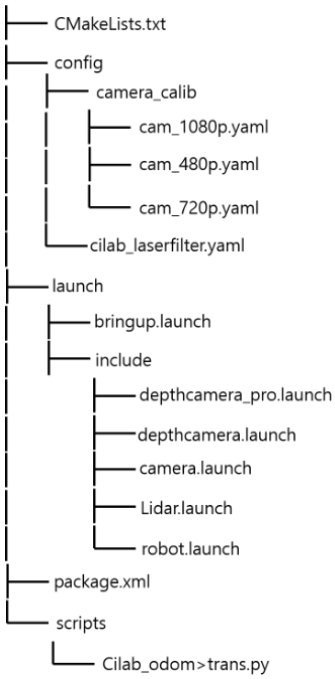
CILAB 로봇의 기본 통합 기능 패키지 cilab_bringup은 각 드라이브 패키지의 동작을 담당합니다.

전원, depth 카메라, 일반 카메라, Li-Dar, URDF 모델, WEB 이미지 디스플레이 등 로봇의 기본적인 기능들을 정리한 것입니다.

9.1. 구동 패키지 설명

로봇 기능 패키지 cilab_bringup에 대한 소개입니다.

패키지의 파일은 아래와 같으며, 핵심 구동 드라이버는 bringup.launch입니다



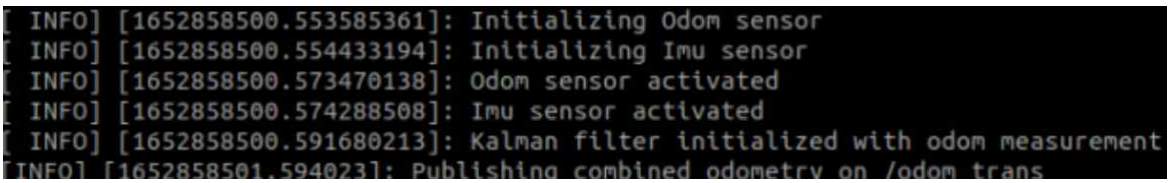
본 챕터에서는 기능 패키지의 파일 프레임워크에 대해서는 위 그림에 소개된 순서대로 설명합니다.

9.2. 로봇 드라이버 (Driver)

cilab_bringup 기능 패키지는 로봇 전원 드라이버를 포함하고 있으며 일부 구성 코드를 추가하여 기능을 더욱 풍부하게 할 수 있습니다. 방식은 다음과 같습니다.

cilab@cilab:~$ roslaunch cilab_bringup robot.launch

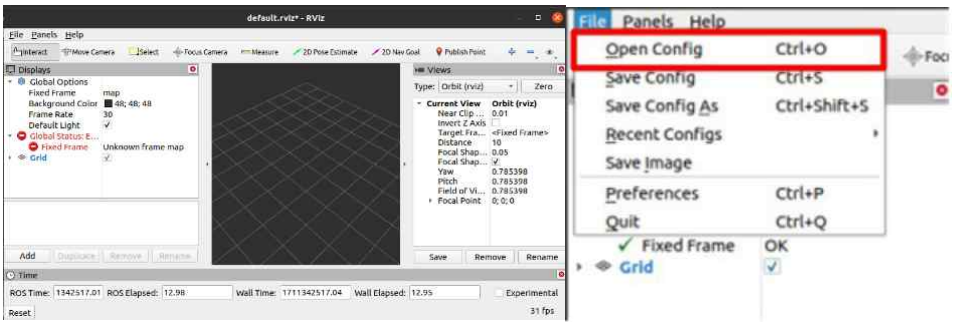
로봇에서 위 명령어를 입력하면 아래와 같은 프롬프트를 확인할 수 있습니다.

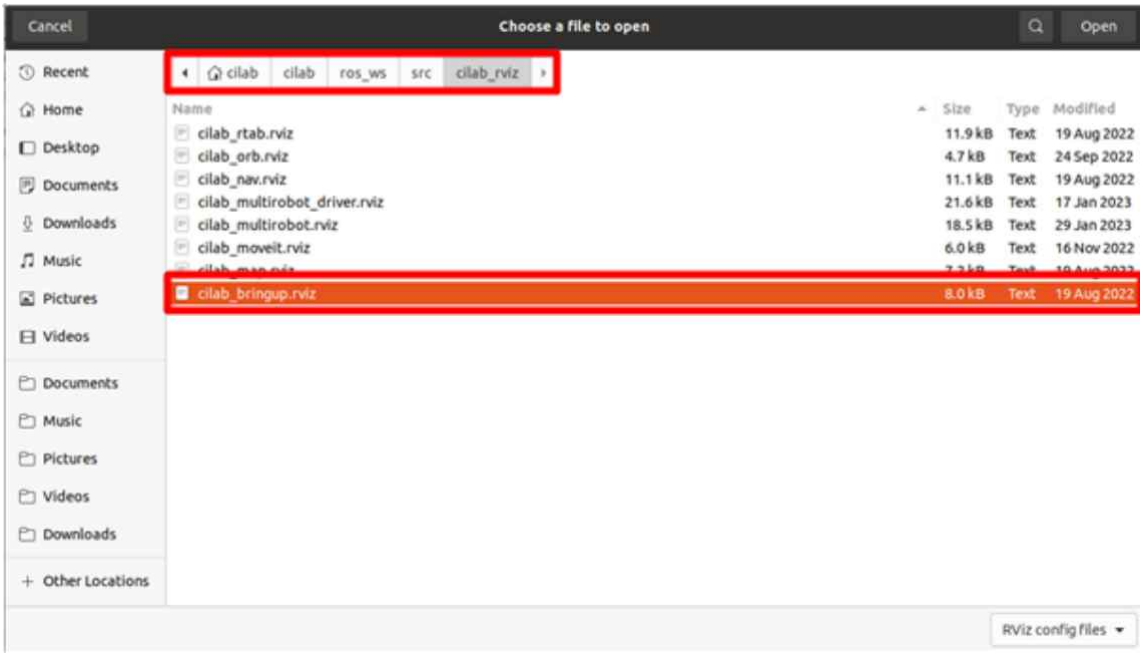


드라이버 실행후 rviz 도구를 사용하여 URDF 파일을 불러와 rviz 인터페이스에서 로봇의 움직임을 확인할 수 있습니다.

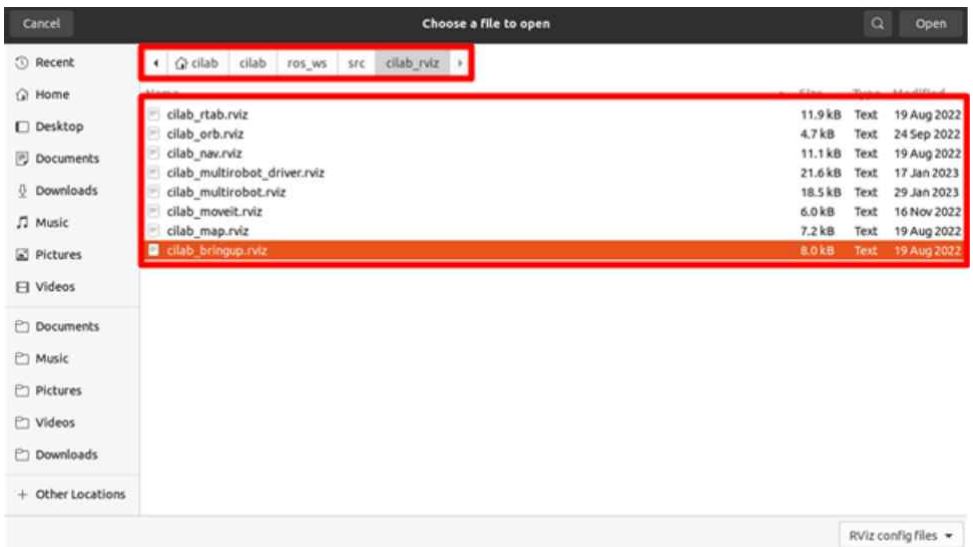
cilab@cilab-pc:~$ rosrun rviz rviz

절차는 아래와 같습니다.

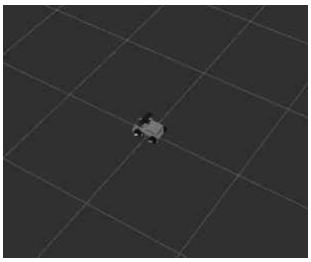


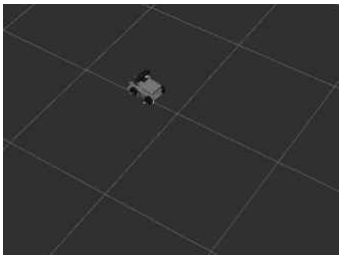


추가 옵션에 선택 가능한 URDF모델을 확인할 수 있습니다.



실제 로봇을 이동할 때 로봇의 URDF모델은 실제 로봇의 이동방향과 동일하게 움직이게 됩니다.



9.3. Li-Dar 드라이버 (Driver)

라이다 드라이버는 로봇 매핑 및 탐색에 필요한 2차원 평면 스캔 데이터를 제공합니다. ssh 명령어를 통해 로봇에 연결한 후 cilab_bringup 기능 패키지의 lidar.launch 파일을 실행하여 라이다 드라이버를 시작합니다.

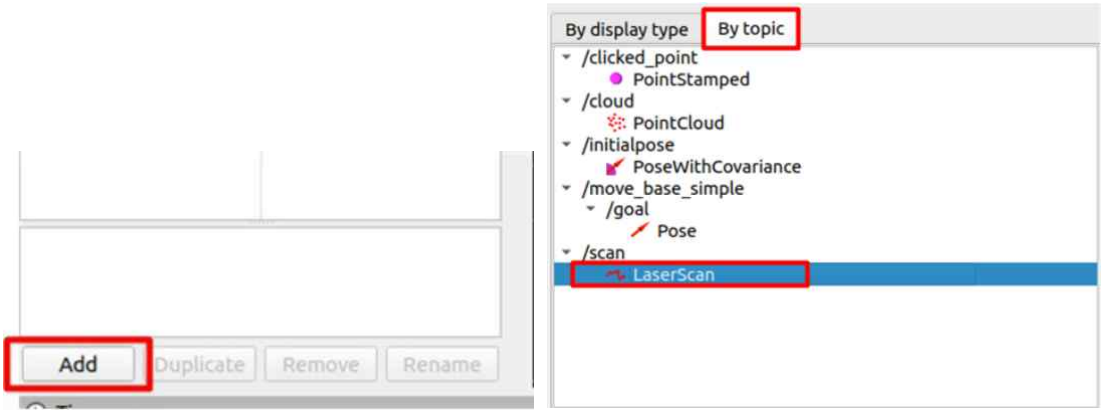
cilab@cilab:~$ roslaunch cilab_bringup lidar.launch

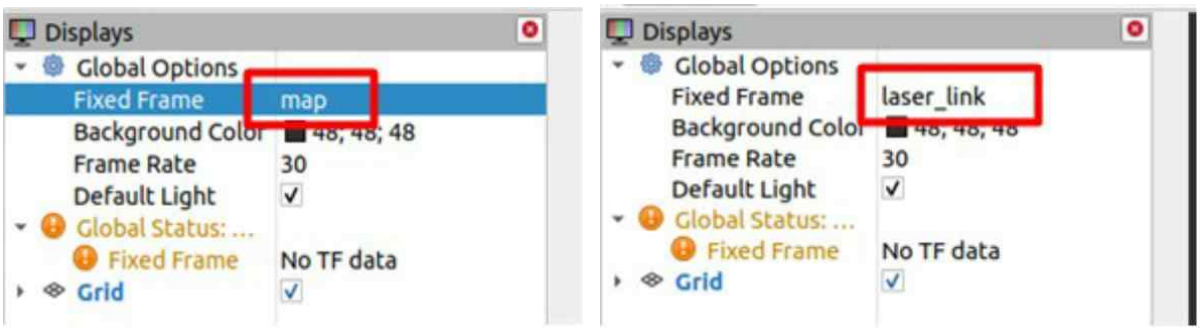
정상적으로 구동되었다면 아래 이미지를 확인할 수 있습니다.

```
send command : 'LFFF1H'
set LiDAR shadow filter to set f
send command : 'LSSS1H'
set LiDAR smooth to set s
send command : 'LSRPM:900H'
set RPM to : OK!
unknown 170 bytes : 70 57 06 01 00 00 01 00 00 01 00 00 01 00 00 01
2340 drop 13 fans
chksum cf errorunknown 132 bytes : 00 18 00 00 19 00 00 24 00 00 26 00 00 35 00
00
3240 drop 18 fans
```

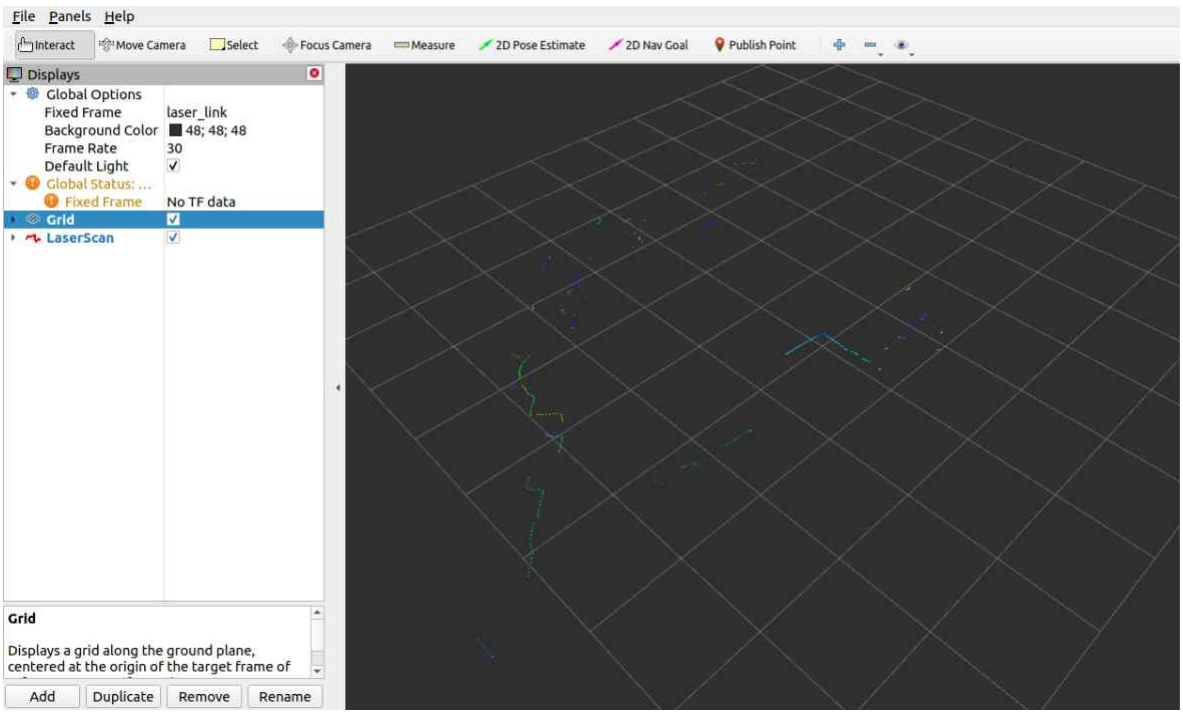
가상 프로그램에서 RVIZ 도구를 실행하여 그림과 같이 라이다 데이터를 확인합니다. 그리고 Fixed Frame 항목을 laser_link로 수정합니다.

cilab@cilab-pc:~$ rosrun rviz rviz



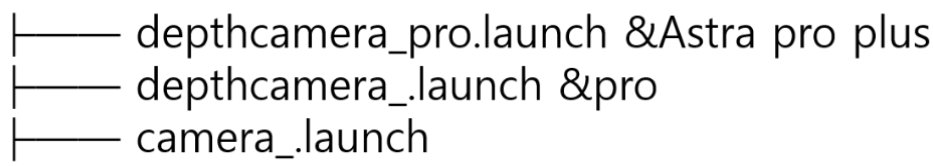


라이다 토픽 데이터는 주변 환경 정보를 보여줍니다.



9.4. 카메라 드라이버 (Driver)

CILAB 시리즈 로봇에는 일반 RGB 카메라, Astra Depth 카메라, Astra Pro Plus Depth 카메라 등 세 가지 카메라 모델이 있으며 카메라마다 부팅 파일이 상이합니다.



다음은 Astra Pro Plus depth 카메라를 예로 들어 설명하며 다른 유형의 카메라는 참고만 해주시기 바랍니다. 해당 카메라는 depth 이미지와 컬러 이미지를 획득할 수 있으며, ssh 명령어를 통해 로봇에 연결한 후 cilab_bringup 기능 패키지의 depthcamera_pro.launch 파일을 실행합니다.

cilab@cilab:~$ roslaunch cilab_bringup depthcamera_pro.launch

Depth 카메라 드라이버를 시작하면 컬러 이미지, 심도 이미지, 포인트 클라우드 이미지 등을 포함한 다양한 이미지 정보를 확인 가능하며, 모두 RVIZ 도구를 통해 확인할 수 있습니다. rostopic list 명령을 통해 내부의 모든 토픽을 확인할 수 있습니다.

cilab@cilab:~$ rostopic list

```
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/camera/rgb/image_rect_color
/camera/rgb/image_rect_color/compressed
/camera/rgb/image_rect_color/compressed/parameter_descriptions
/camera/rgb/image_rect_color/compressed/parameter_updates
/camera/rgb/image_rect_color/compressedDepth
/camera/rgb/image_rect_color/compressedDepth/parameter_descriptions
/camera/rgb/image_rect_color/compressedDepth/parameter_updates
/camera/rgb/image_rect_color/theora
/camera/rgb/image_rect_color/theora/parameter_descriptions
/camera/rgb/image_rect_color/theora/parameter_updates
/camera/rgb_rectify_color/parameter_descriptions
/camera/rgb_rectify_color/parameter_updates
```

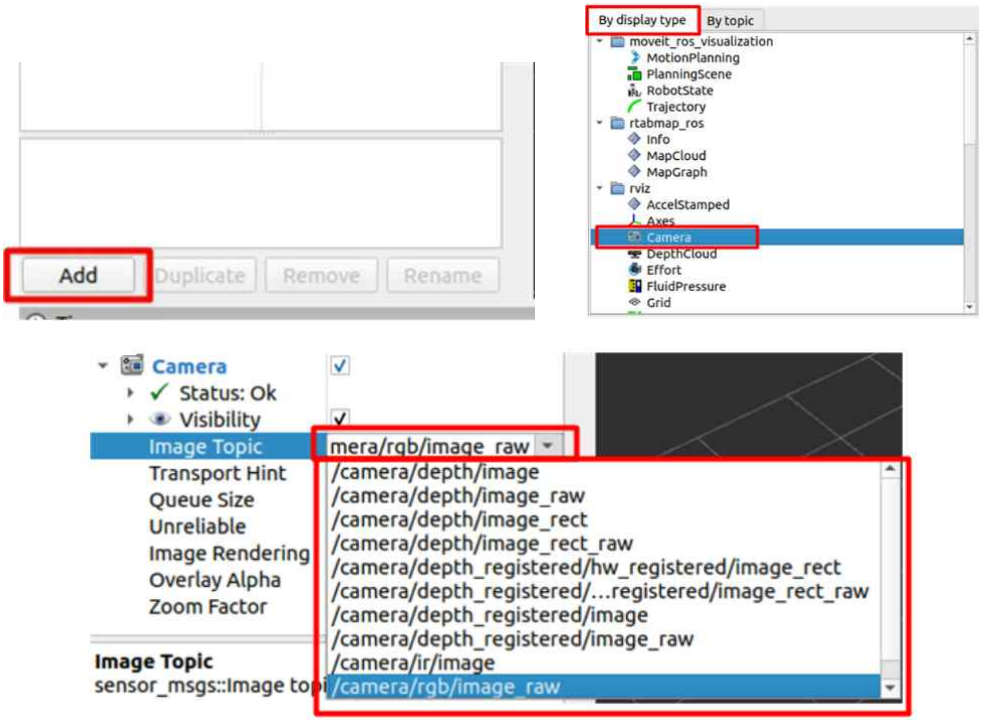
Depth 카메라는 다양한 토픽을 송출하며 주로 사용되는 토픽 몇 가지에 대해 설명합니다. 해당 퍼포먼스는 RVIZ 또는 rqt_image_view 도구를 통해 확인할 수 있습니다.

RVIZ 도구

아래 명령어를 통해 rviz 프로그램을 실행합니다.

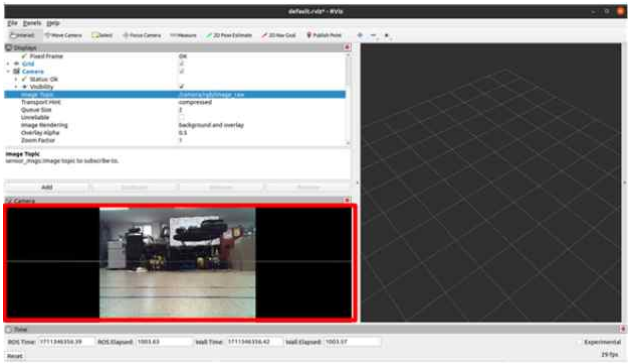
cilab@cilab-pc:~$ rosrun rviz rviz

아래 방법과 동일하게 현재 필요로 하는 토픽을 선택합니다.

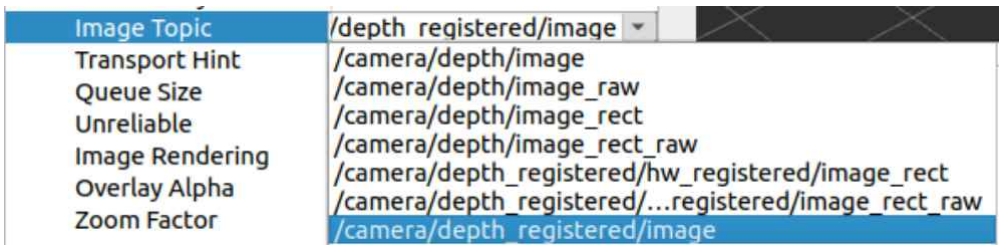


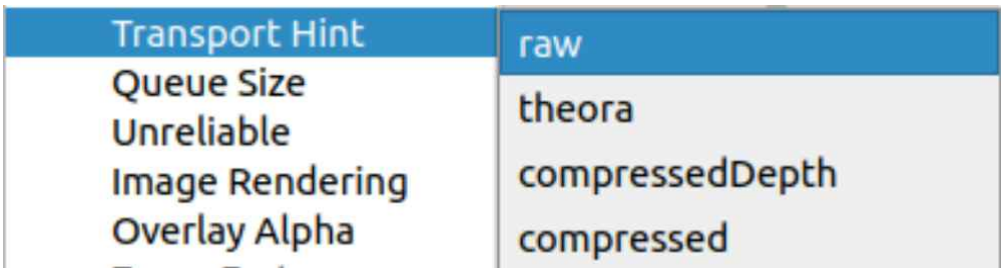
일반 RGB이미지도 선택해줍니다.

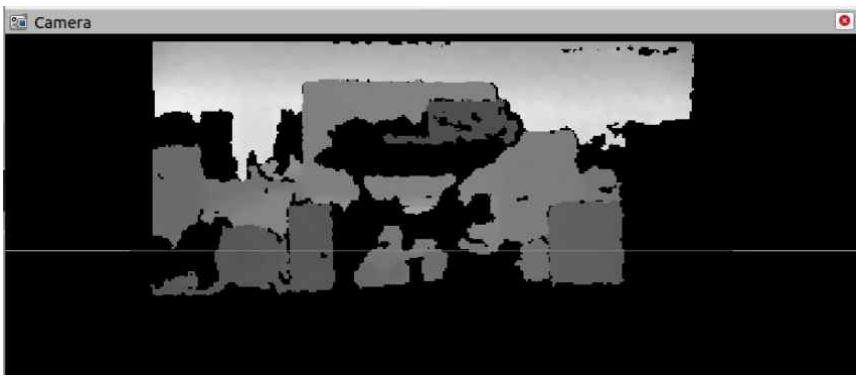
추가 완료한 후에 해당 GUI의 이미지는 아래와 같습니다.



디스플레이 퍼포먼스는 아래와 같으며, 다른 유형의 이미지는 토픽 수정을 통해 확인할 수 있습니다. 이상이 감지되면 사용자는 선택한 토픽이 올바른지 확인할 수 있습니다. Depth 카메라는 raw 원본 데이터만 선택할 수 있으며 압축 데이터를 지원하지 않습니다.





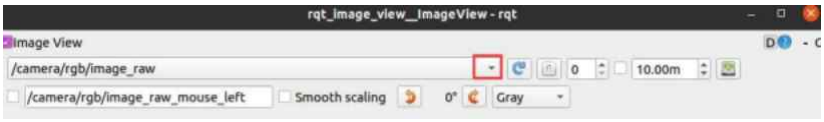


rqt_image_view 도구

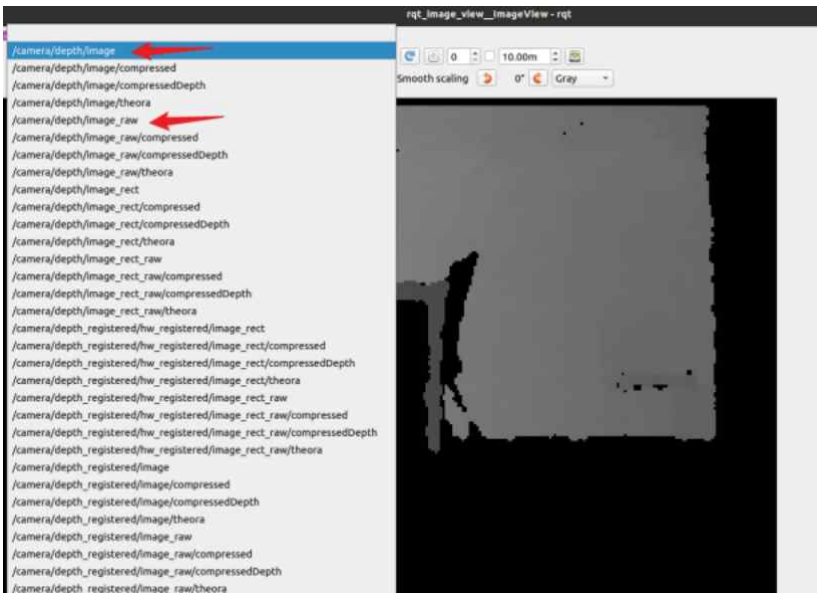
이미지 토픽을 확인할 때 rqt_image_view 도구를 사용합니다. 가상 프로그램에서 아래 명령을 실행하여 rqt_image_view 도구를 실행합니다.

cilab@cilab-pc:~$ rosrun rqt_image_view rqt_image_view

위 명령어를 입력하면 아래 팝업창이 나타납니다.



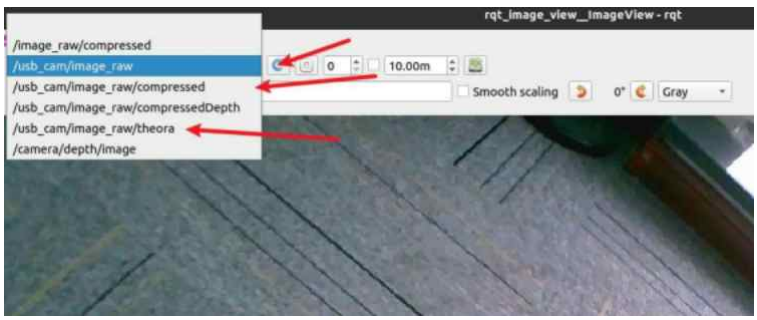
아래 리스트 확인 버튼을 클릭하면 현재 열람할 수 있는 이미지 토픽들을 확인할 수 있습니다. 확인하고자하는 토픽을 선택합니다.



RGB 일반 카메라의 경우에는 선택가능한 토픽으로 rgb 관련 이미지들로 구성이되며 Depth 데이터는 없습니다. ssh명령어를 통해 로봇에 연결하여 cilab_bringup 기능 패키지의 camera를 실행해 depth 카메라 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_bringup camera.launch

일반 RGB 카메라 드라이버를 구동하면 RGB 이미지 정보가 전송되며 이는 rviz나 rqt_image_view gui를 통해 확인하실 수 있습니다.



9.5. WEB 이미지 디스플레이

카메라 드라이버가 실행된 후 WEB 기반 이미지 서버를 시작할 수 있으며 사용자는 웹 브라우저에 웹 주소를 입력하여 로봇 카메라 데이터의 브라우저를 표시할 수 있습니다. 먼저 ssh 명령을 통해 로봇에 연결하고 아래 depth카메라 드라이버를 실행합니다. (본 매뉴얼에서는 depth카메라를 기준으로 설명합니다)

cilab@cilab:~$ roslaunch cilab_bringup depthcamera_pro.launch

실행 완료 후 아래 명령어를 입력해 웹브라우저 서비스와 연동을 시작합니다.

cilab@cilab:~$ rosrun web_video_server web_video_server

웹 브라우저 주소창에 아래 IP주소를 입력해줍니다. (로봇에서의 주소)
http://192.168.3.10:8080 을 입력하면 옵션이 나타나고 원하는 토픽을 선택하면 영상 확인이 가능합니다.

9.6. 구동 드라이버 설명

9.6.1. bringup 구동 드라이버 조작

로봇에서 bringup.launch 파일을 실행합니다.

cilab@cilab:~$ roslaunch cilab_bringup bringup.launch

아래 프롬프트가 나타나면 정상적으로 구동되었음을 나타냅니다.

```
process[camera_base_link1-21]: started with pid [3082]
process[camera_base_link2-22]: started with pid [3097]
process[camera_base_link3-23]: started with pid [3105]
process[web_video_server-24]: started with pid [3116]
[ INFO] [1652852138.675105477]: Device "2bc5/0402@1/5" found.
Warning: USB events thread - failed to set priority. This might cause loss of da
ta...
[ INFO] [1652852138.877191034]: device name: Orbbec Astra S
[ INFO] [1652852139.306244674]: Waiting For connections on 0.0.0.0:8080
[INFO] [1652852139.927082]: Publishing combined odometry on /odom_trans
```

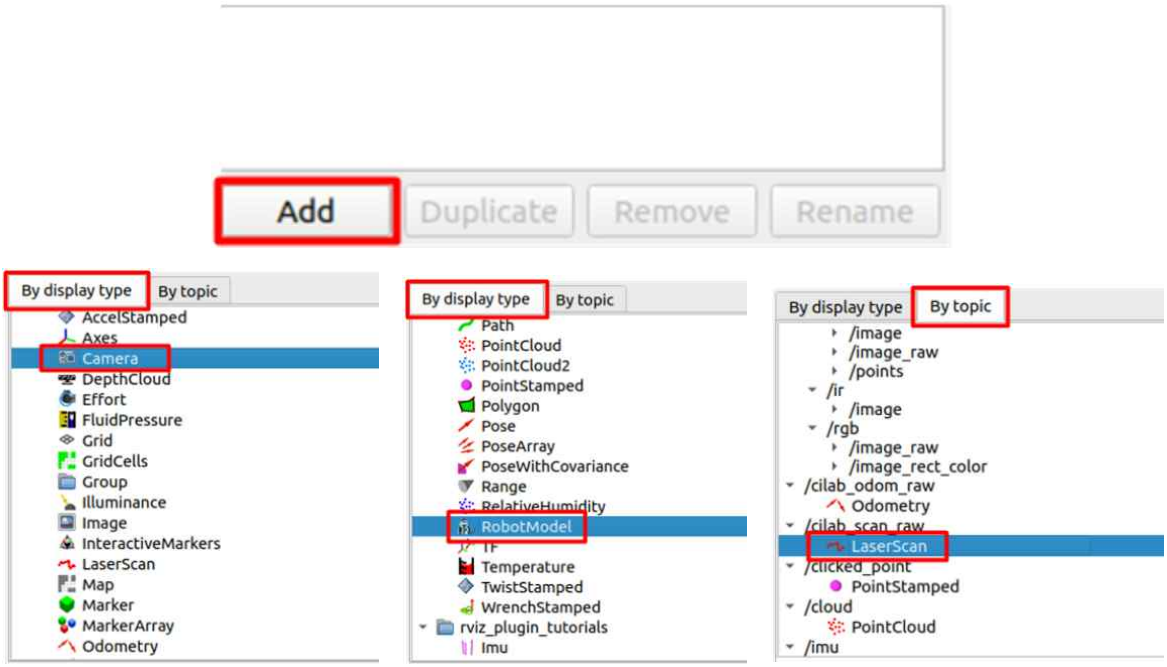
로봇은 기본적으로 다음과 같이 작동합니다.

1. 드라이브(urdf포함)
2. 라이다
3. depth 카메라
4. 웹 비디오 서비스

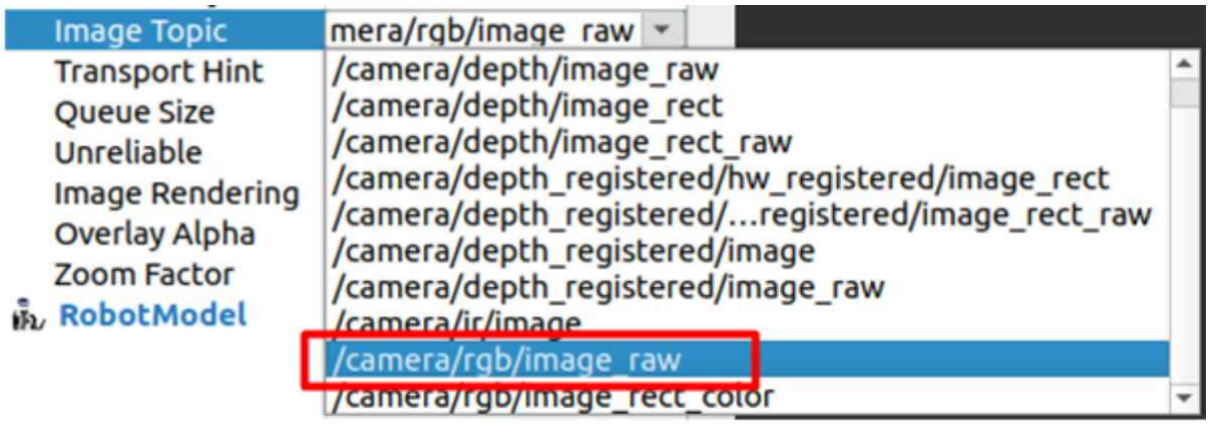
위 내용 모두 rviz 시각화 인터페이스를 통해 확인할 수 있으며 가상 프로그램에서 명령어를 입력해줍니다.

cilab@cilab-pc:~$ rosrun rviz rviz

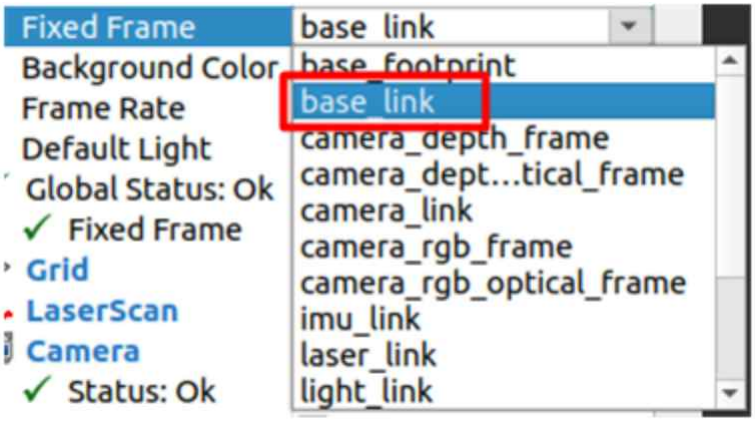
rviz에 로봇 모델링, 라이다 스캔 데이터, 카메라 이미지 토픽을 추가하여 로봇 대응 드라이버가 정확하게 동작하는지 확인할 수 있습니다.



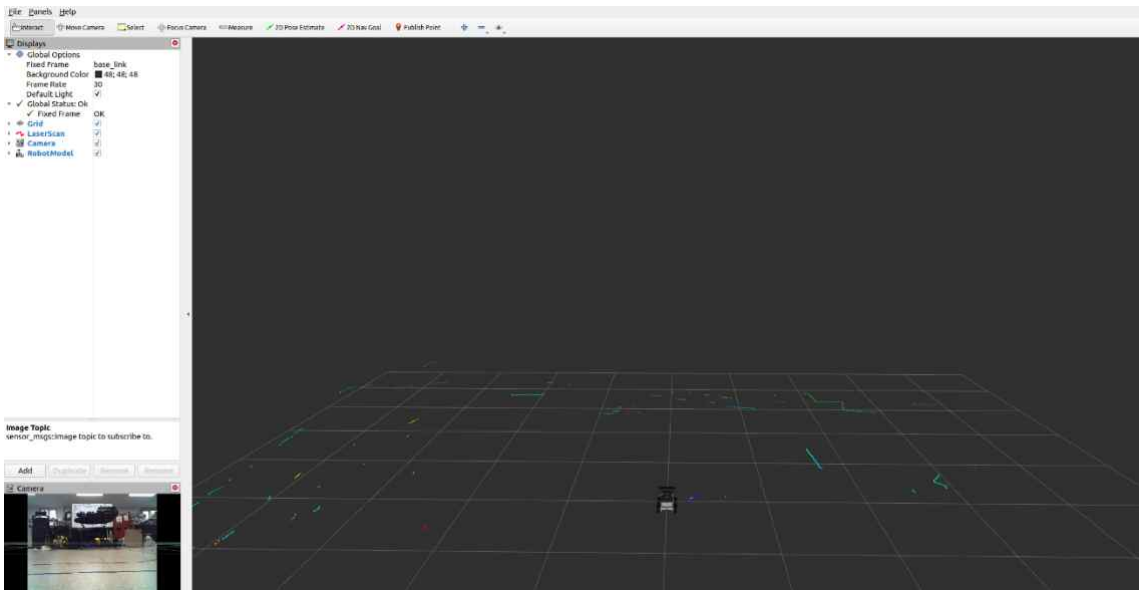
작업 완료 후에는 rviz 인터페이스에서 해당 로봇 모델링을 확인 가능하게 됩니다. 그리고 카메라 이미지도 함께 추가합니다.



라이다 스캔 이미지를 추가합니다.



완료된 퍼포먼스는 아래와 같습니다.

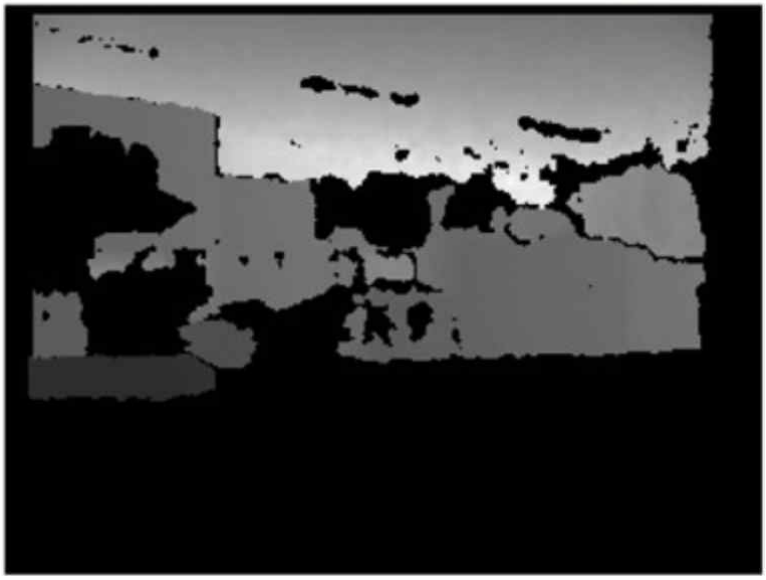


동영상 서비스 기능은 컴퓨터의 브라우저를 사용해야 하며 (동일한 네트워크에 연결되어 있는지 확인), 브라우저에 아래 URL을 입력합니다.

http://192.168.3.10:8080/stream_viewer/topic = /camera/rgb/image_raw

IP 주소를 로봇에 해당하는 IP로 변경하면 다음과 같은 퍼포먼스가 가능합니다.

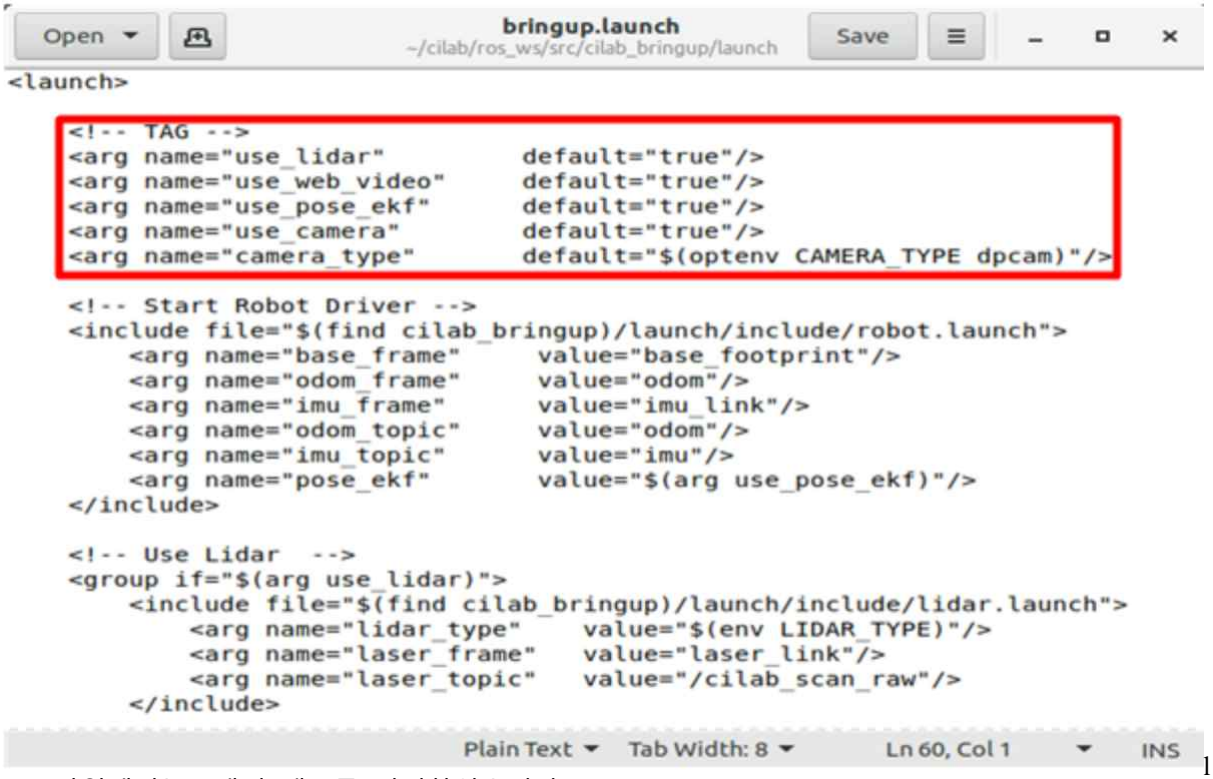
/camera/depth_registered/hw_registered/image_rect_raw



이제 원격 제어 방법을 사용하여 로봇을 제어할 수 있으며, 로봇 모델의 이미지 정보가 로봇의 움직임에 따라 실시간으로 이동하는 것을 볼 수 있습니다.

9.6.2. bringup 구동 드라이버 나열

부팅 파일 bringup.launch의 주요 코드의 내용은 다음과 같습니다.

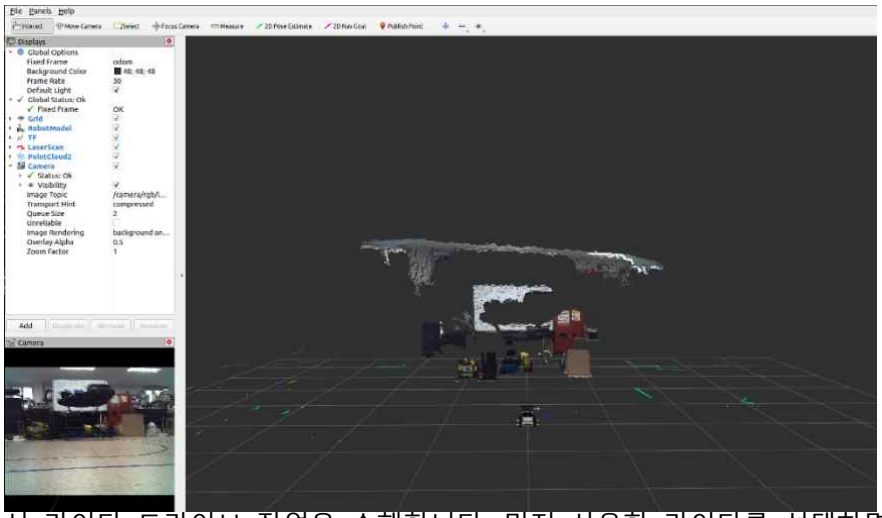


launch 파일에서는 5개의 태그를 정의하였습니다.

use_lidar : 레이더 사용 여부
use_camera : 카메라 사용 여부
use_web_video: 비디오 서버의 사용 여부
use_pose_ekf: ekf 융합 드라이버 사용 여부
camera_type: 어떤 종류의 카메라를 사용하는지

bringup을 시작할 때 상기 라벨의 활성화 여부를 선택할 수 있으며 선택하지 않을 경우 기본 매개변수가 로드되기 때문에 라벨을 선택한 후 기능을 활성화해야 합니다. robot.launch에서 구동 드라이브를 시작하고 다음과 같은 작업을 진행합니다.

기본 구동 드라이브를 시작하고 use_pose_ekf를 선택하여 구동 드라이브에서 ekf 융합을 사용할지 여부를 결정하며, ekf 융합 전에는 로봇이 odom 데이터를 메인 데이터로 사용하고 ekf 융합 후에는 IMU 데이터를 통합하여 마일리지 오류를 교정하여 데이터를 보다 정확하게 만듭니다. 그런 다음 urdf 모델을 시작하여 로봇 모델을 rviz에 로드하고 로봇 센서 간의 TF 변환을 송출합니다.



lidar.launch에서 라이다 드라이브 작업을 수행합니다. 먼저 사용할 라이다를 선택하면 드라이버가 자동으로 읽힙니다. bashrc 파일의 LIDAR_TYPE 매개변수 파트에서 사용할 레이더를 선택 구동합니다. 간섭이 발생할 수 있는 위치 좌표를 제거하기 위해 라이다 필터 드라이버가 실행됩니다.

```
# LAIDAR Model, PaceCAT：e300  SLAM A1：a1
export LIDAR_TYPE=e300
```

카메라를 작동시키고 사용할 카메라를 선택하면 드라이버가 자동으로 불러옵니다. bashrc 파일의 camera_type 매개변수에서 사용할 카메라 유형을 결정, 구동합니다.

드라이브 패키지는 아래 파일주소를 참고합니다.

공식 주소 : http://wiki.ros.org/astra_camera

```
# CAMERA MODEL,  PRO Depth CAM：dpcam_pro  DEPTH CAM：dpcam  Ordinary CAM：cam
export CAMERA_TYPE=dpcam_pro
```

공식 주소 : http://wiki.ros.org/usb_cam

마지막으로 웹 서비스가 시작되었으며, 이 서비스를 호출하면 브라우저에서 이미지를 호출할 수 있습니다.

공식 주소 : http://wiki.ros.org/web_video_server

10. 기타설명

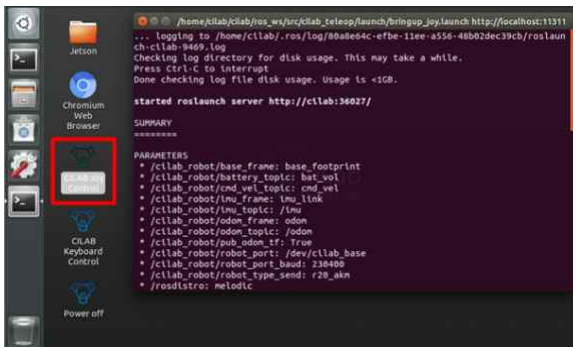
본 챕터에서는 ROS 개발 단계에 있어 필요한 내용을 설명합니다.

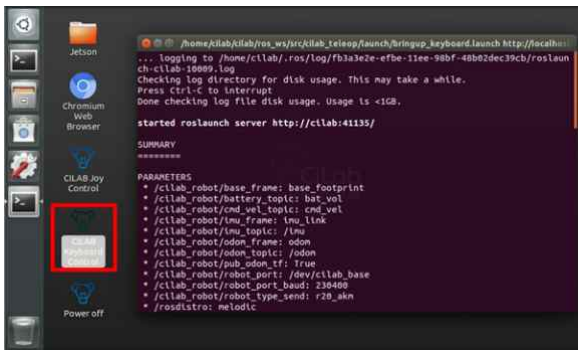
10.1. 단축키(Shortcut) 아이콘

X3의 경우 원활한 로봇의 제어를 위해 터치디스플레이를 탑재하고 있습니다. 해당 디스플레이

바탕화면에는 총 4가지의 단축키 아이콘이 마련되어 있습니다. 해당 아이콘을 더블 클릭하게 되면 정상적으로 해당 기능이 실행되게 됩니다. (Joy 컨트롤, Keyboard 컨트롤, 전원 오프, 재시작)

CILAB Keyboard Control 아이콘은 로봇에서의 키보드 컨트롤을 위한 단축 아이콘으로 작동방식은 앞서 설명되었던 내용과 동일합니다. (사용된 예제 드라이버는 구동 드라이버와 키보드 컨트롤 드라이버입니다)





CILAB Power off 아이콘의 경우 이전보다 편리하고 신속하게 안전한 방식으로 전원을 종료하는 기능입니다. 두개의 파일 모두 ~/Desktop경로에 저장되어 있으며, 유저는 해당 예제코드를 참조하여 본인만의 코드를 생성할 수도 있습니다.

10.2. Nano 메인 확장

Nano 메인 컨트롤은 미러링을 만들 때 전달이 용이하도록 디스크를 약 30GB 정도까지 압축하였으며 2차 개발에 더 큰 저장 공간이 필요한 경우 다음 명령을 사용하여 디스크 용량을 확장할 수도 있습니다.

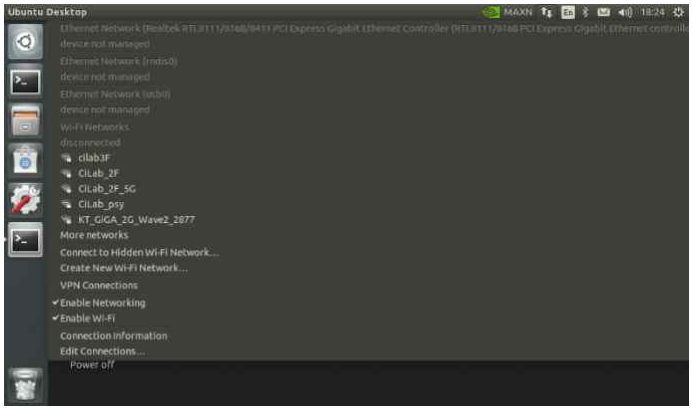
cilab@cilab:~$ sudo /usr/share/cilab/cilab_resizefs.sh

10.3. Wi-Fi ↔ AP 변환하기

로봇이 Wi-Fi 모드에 연결되어 있을 때 유저입장에서는 AP모드로 다시 전환해야 하는 경우가 생길 수 있는데 그럴 경우 아래 매뉴얼을 따라 변환이 가능합니다. 동일하지 않은 ROS 메인 컨트롤간에는 화면상 차이가 있을 수 있으니 양해바랍니다. X3의 경우에는 Ubuntu 20.04버전을 사용하고 있습니다.

Jetson Nano (Ubuntu20.04)

WIFI 모드에서는 언제든지 AP모드로의 변환이 가능합니다. 아래 설명을 참조합니다.



아래 팝업 이미지가 나타나면 [Wi-Fi connetion 1]을 선택합니다.

그리고 난 후 오른쪽 아래 connect 버튼을 클릭하고 잠시 기다리면 현재 Wi-Fi 모드에서 AP모드로 자동 변환하게 됩니다.

개인 AP로 변환 연결되었음을 확인할 수 있습니다.

10.4. 조이스틱 컨트롤

앞에서 키보드 컨트롤 로봇을 사용하는 것에 대해 자세히 설명했는데, 사용자가 USB 핸들이나 핸들 컨트롤을 확장하고 싶다면 이 절의 내용을 참고하시거나 매뉴얼 2페이지의 연락처로 문의 남겨주시면 언제든 회신 가능합니다.

10.4.1. 조이스틱 컨트롤

키보드로는 고정 속도 제어만 가능하지만 무선 조이스틱은 무빙 스틱으로 속도 제어가 가능하고 보다 정교한 제어를 실현할 수 있습니다.

조이스틱으로 로봇을 제어하면서 로봇 또는 가상 프로그램에서 프로그램을 제어할 수 있습니다.

특별 설명 : 본 챕터에서의 사용 손잡이는 표준 PC 게임 핸들(무선 핸들 포함)이며 일부 핸들은 안드로이드 모드, PS3 모드 등과 같은 다양한 모드를 지원하며 사용 시 PC 모드로 전환됩니다. (빨간불이 항상 켜져 있는 상태)

조이스틱 다이렉트 컨트롤

잿슨 나노에 연결한 컨트롤은 근거리 제어에 적합하며 제어 거리는 10미터 미만이며, 사용 전 전용 동글을 차량 잿슨 나노 컨트롤러에 삽입해야합니다.

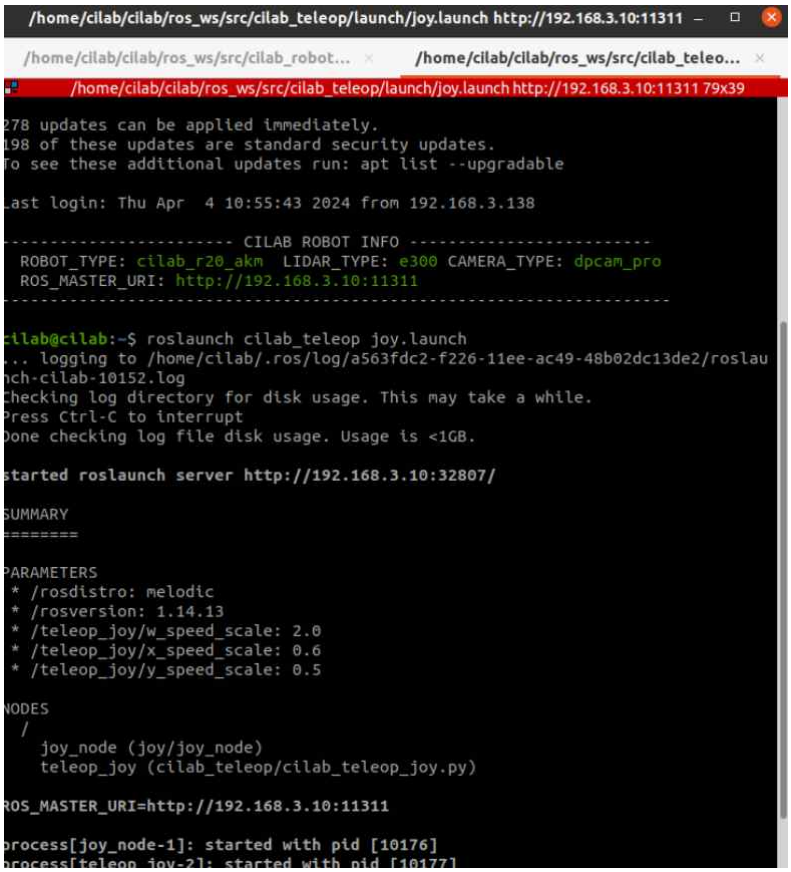
먼저 ssh 명령어를 이용해 로봇에 연결하고 차량 구동 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

차량이 정상적으로 구동이 되었다면 새로운 터미널을 실행해 ssh 명령어를 이용해 차량으로 연결하고 조이스틱 컨트롤 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_teleop joy.launch

해당 드라이버가 정상적으로 실행되면 아래 프롬프트가 나타납니다.



조이스틱 컨트롤 방식은 L1 버튼을 누른 상태에서 무빙 스틱의 조작이 이루어져야 합니다. 오른쪽 무빙 스틱은 전방향 이동이 가능하며 왼쪽은 차량의 회전방향에 해당합니다. 차동기어 플랫폼인 X3_akm의 경우에는 오른쪽 스틱으로 전진 후진, 왼쪽으로 방향 조절만 가능한 점 주의바랍니다.

PC조이스틱을 사용하실 경우 조이스틱을 너무 빠르게 동작할 경우 운행이 원활하지 않을 수 있습니다. 원격 제어 속도는 필요에 따라 조정할 수 있기에 본인이 컨트롤할 수 있는 적정 속도로 운행하는 것을 권장합니다.

가상 프로그램 조이스틱 컨트롤

로봇에서 거리가 비교적 떨어져 있는 경우 (10m를 초과하는 거리) 조이스틱을 개발 호스트에 삽입하여 제어할 수 있습니다. 연결 후 새로운 터미널을 생성해 ssh 명령어를 이용하여 로봇으로 연결해 구동 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_robot robot.launch

구동 드라이버가 동작한 후에는 새로운 터미널을 생성해 가상프로그램에서의 조이스틱 컨트롤 드라이버를 실행합니다.

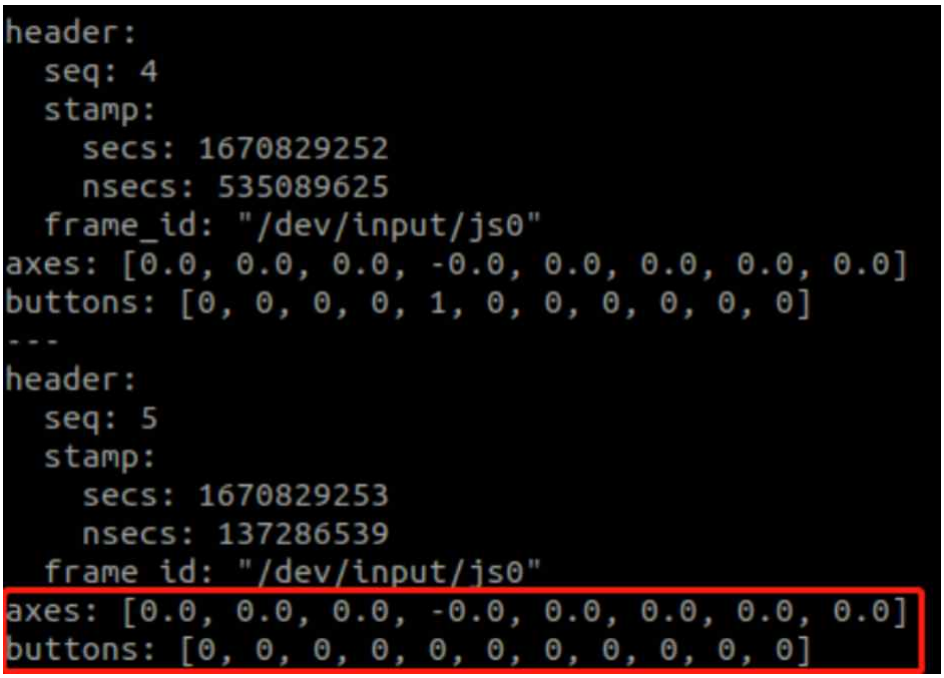
cilab@cilab:~$ roslaunch cilab_teleop joy.launch

드라이버가 실행되면 바로 조이스틱의 동작이 가능합니다. 이때 버튼 정의는 해당 핸들에 나오

는 규격을 참고하시기 바랍니다. 이렇게 하면 핸들을 이용하여 더 먼 거리에서도 컨트롤이 가능하며, 신호가 원활한 경우 로봇을 Wi-Fi에 연결하여 조이스틱으로 제어할 수도 있습니다.

조이스틱 토픽

ssh 명령어를 이용해 로봇에 연결하고 rostopic list 명령어를 입력하면 현재 ROS에서 실행 중인 드라이버를 확인 가능합니다. /joy 토픽을 확인했다면 rostopic echo /joy 명령어를 입력해 다음과 같은 프롬프트 창을 확인할 수 있습니다.



조이스틱의 버튼이나 스틱을 움직이면 위 사진에 표시된 데이터 값이 변동되는 것을 확인할 수 있으며, 해당 연관 관계를 찾아 원격 제어 드라이버를 직접 수정할 수도 있습니다.

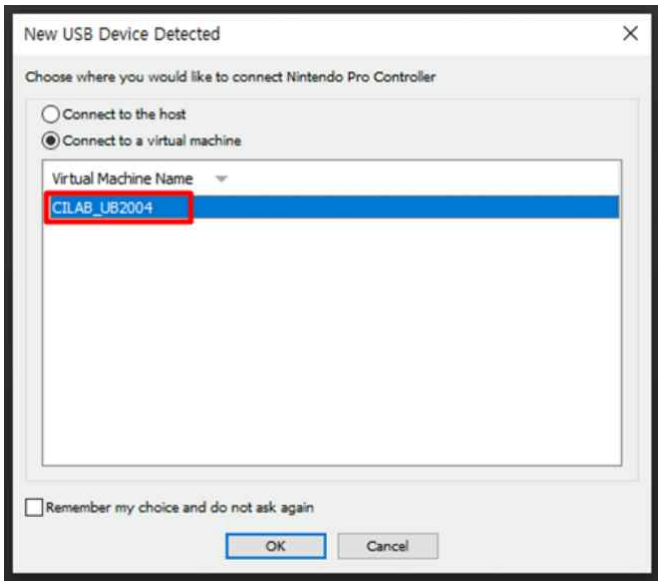
10.4.2. 방향 핸들 조이스틱 (옵션)

X3부터는 전용 핸들링 옵션이 추가되었으며, 해당 기능을 통해 실제 운전과 유사한 방식으로 로봇을 제어할 수 있습니다. 해당 방식으로 사람들에게 더 나은 게임 경험을 제공할 수 있으며, 연결 방법은 아래와 같습니다. (하드웨어와 유선 연결되어야 합니다)

가상 프로그램 핸들링 제어

제어 전에 먼저 핸들을 가상 프로그램에 연결해주어야 하며 핸들을 가상 프로그램 호스트에 삽입하면 어떤 시스템을 연결할 것인지를 팝업창이 나타나게 되고 아래 그림과 같이 가상 프로그램으로 연결을 선택합니다.

cilab@cilab:~$ ros launch cilab_robot robot.launch



구동 드라이버가 실행된 후 터미널을 실행하여 가상 프로그램에서 핸들 컨트롤 드라이버를 실행합니다. 핸들링 기능을 시작하기 전에 로봇 타이어를 평평하고 트여진 곳에 위치시켜 드라이버 동작시 차량이 급발진하는 것을 방지해야 합니다. (이때 발생할 수 있는 파손에 대해서는 책임지지 않습니다)

cilab@cilab-pc:~$ roslaunch cilab_teleop wheel.launch

해당 드라이버가 정상적으로 실행되면 핸들링 기능을 사용할 수 있으며 동작 중에는 다음과 같이 컨트롤이 가능합니다.

핸들링 토픽

유저는 핸들링 토픽을 보고 각각의 버튼에 해당하는 채널을 확인 가능합니다.

ssh 명령어를 이용해 로봇에 연결하고 rostopic list 명령어를 입력하면 현재 ROS에서 실행 중인 드라이버를 확인 가능합니다. /joy 토픽을 확인했다면 rostopic echo /joy 명령어를 입력해 다음과 같은 프롬프트 창을 확인할 수 있습니다.

```
header:
  seq: 377
  stamp:
    secs: 1711934251
    nsecs: 266685039
  frame_id: "/dev/input/js0"
axes: [-0.18503372371196747, -0.0, 1.0, -0.0, -0.0, 0.5512253046035767, -0.0, -
0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
header:
  seq: 378
  stamp:
    secs: 1711934251
    nsecs: 714903914
  frame_id: "/dev/input/js0"
axes: [-0.19290749728679657, -0.0, 1.0, -0.0, -0.0, 0.5512253046035767, -0.0, -
0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
```

11. SLAM 매핑

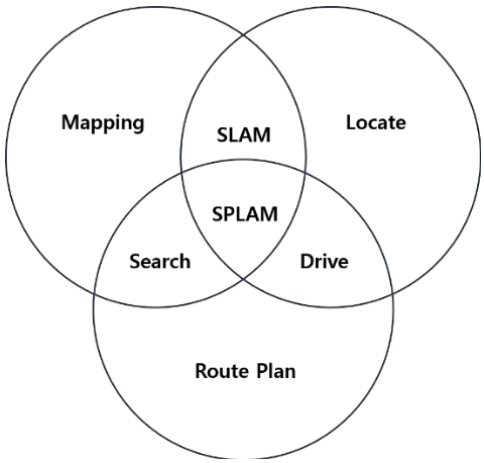
11.1. SLAM 매핑 개요

SLAM은 영어로 ‘simultaneous localization and mapping’로 ‘실시간 위치 측정 및 지도 구축 또는 동시 매핑 및 포지션 지도화’라고 정의할 수 있습니다. SLAM은 로봇이 임의의 알 수 없는 환경으로부터 이동하면서 위치를 측정하고 동시에 지도를 활용하여 위치 측정을 진행하며 데이터를 쌓는 식의 지도를 구축함으로써 위치 측정 및 내비게이션을 실현하는 것을 뜻합니다.

초창기 연구 단계에서는 로봇 위치 측정 문제와 매핑 문제는 별개의 것으로 간주되었습니다. 로봇 위치 측정은 기지의 전역 맵을 기반으로 하여 로봇 센서를 활용해 측량을 진행하였습니다. 측정된 정보와 지도 사이의 관계를 활용하여 지도 내 로봇의 위치를 확보하는 것입니다.
매핑은 로봇의 전반적인 위치를 알고 있는 상태에서 센서를 통해 주변 환경을 측정하고, 맵 상에서 좌표를 측정할 때 로봇 위치 및 측정 거리, 방향 정보를 활용하여 수집한 맵 좌표를 계산하는 것입니다. 그러나 이처럼 사전의 환경 검증을 기반한 위치 추정 및 매핑은 임의의 알 수 없는 환경에서 작동하기 어렵다는 한계가 있습니다.

SLAM 기술은 이 두가지 문제를 하나로 통합시켰습니다. 로봇이 복잡한 미지의 밀폐 환경에 위치할 때, 센서를 활용하여 주변 환경 데이터를 수집합니다. 로봇이 처한 미지의 환경을 자체적으로 관찰하고 측정하여 공간 위치 파악과 환경 매핑을 실현하는 것입니다.

SLAM 자율 내비게이션 솔루션은 매핑, 위치 측정 및 경로 계획 3가지 기본 이슈로 구성되어 있습니다. 이 세 가지 이슈는 서로 중첩되어 새로운 이슈, 즉 SLAM 이슈, 내비게이션 이슈, 탐색 이슈 등으로 다시 구성합니다. 이를 구성하는 핵심 요소는 지도, 목적지, 위치 측정, 장애물, 경로 계획 등이 있으며 상호간의 관계는 아래와 같습니다. 본 챕터에서는 주로 SLAM 매핑 이슈에 대해 설명합니다.



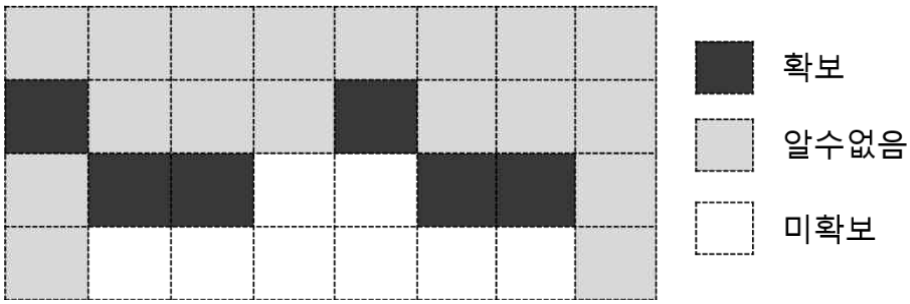
로봇은 SLAM과 자율 내비게이션을 수행하기 위해 먼저 주어진 주변 환경 데이터를 탐지할 수 있는 능력이 필요합니다. 특히 주변 환경의 상세한 정보를 인지할 수 있어야 하는데 이를 위해서 라이다(Li-Dar) 센서가 필요합니다.

ROS 로봇은 GMapping, hector, karto, cartographer 등 다양한 매핑 알고리즘을 지원하며 관련하여 후반부에서 자동 탐색과 같은 매핑 알고리즘에 대해서도 설명이 이어지니, 유저는 필요한 파트를 직접 선정하여 테스트를 진행할 수 있습니다.

11.1.1. 그리드 지도 (Geurideu MAP)

"환경 모델링"이란 주변 환경 상태를 설명하는 것인데, 다른 말로 하면 "환경 지도를 구축하는 것"이라고 할 수 있습니다. 스캔한 지도는 위치 측정과 장애물 회피에 사용될 수 있습니다. 따라서 위치 측정과 장애물 회피에 사용되는 지도는 반드시 동일하지 않을 수도 있습니다. 환경 지도에는 여러 가지 종류가 있으며 Li-Dar를 이용한 SLAM은 주로 그리드 지도를 구축하는 데 중점을 두고 있습니다.

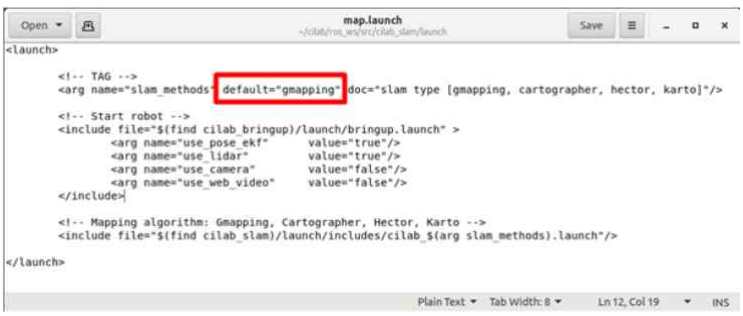
2차원 그리드 지도는 2차원 연속 공간을 그리드 형식으로 분할하는 것으로 설명할 수 있습니다. 로봇은 일반적으로 2차원 점유 그리드 지도를 사용하는데, 이는 각 그리드를 점유 확률 값에 기반하여 수치화하는 것입니다. 아래 이미지에서 볼 수 있듯, 확률이 1인 그리드는 확보 상태로 표시되고, 확률이 0인 그리드는 미확보 상태로 표시됩니다. 0과 1 사이의 확률 값을 갖는 그리드는 미지의 상태 즉 알수없음으로 표시됩니다. 로봇은 내비게이션 과정에서 확보 상태인 그리드를 피하고, 미확보 상태인 그리드 사이를 통과합니다. 로봇은 센서를 사용하여 이 알수없음 상태 그리드를 탐색합니다.



11.1.2. 매핑 패키지

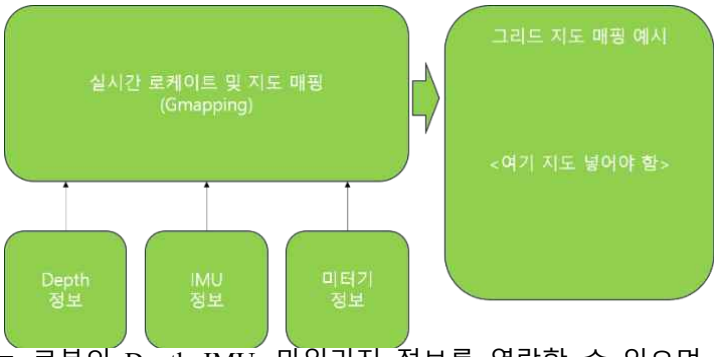
로봇 매핑 패키지는 cilab_slam이며, 일반적으로 사용되는 네 가지 SLAM 매핑 알고리즘이 기본적으로 설치되어 있습니다. 패키지에 있는 cilab_map.launch 파일을 통해 다양한 매핑 알고리즘을 실행할 수 있습니다.

먼저 로봇에서 bringup.launch 파일을 실행하여 구동, 모델링 파일, TF 변환, 라이다 등 드라이버를 가동한 다음, 파라미터에 따라 다른 알고리즘 파일을 실행합니다. 기본 디폴트 알고리즘은 Gampping입니다.



11.2. GMapping 매핑

GMapping 패키지는 Rao-Blackwellized 파티클 필터 알고리즘을 통합한 것으로 개발자를 위해 복잡한 내부 구현을 생략하였습니다. GMapping 패키지의 전체적인 프레임은 아래와 같습니다.



GMapping 패키지는 로봇의 Depth, IMU, 마일리지 정보를 열람할 수 있으며 몇 가지 필수 파라미터만 설정하면 확률 기반의 2D 그리드 지도를 구축하고 출력할 수 있습니다. GMapping 패키지는 Open Slam 커뮤니티의 오픈 소스 SLAM 알고리즘을 기반으로 한 것입니다. (자세한 사항은 SLAM 홈페이지를 참고바랍니다)

GMapping은 협소한 환경의 지도를 구축할 때 측정량이 적어지는 대신 정확도가 높아집니다. Hector SLAM과 비교했을 때 라이다에 대한 저주파 요구사항과 견고함이 향상되었습니다. GMapping은 지면이 평탄하고 공간이 좁은 환경에서 매핑 효과가 좋습니다. 마일리지 데이터를 필요로 하기에 광활한 곳이나 지면이 평탄하지 않은 지역, 파티클(노이즈)이 많은 경우에는 적합하지 않을 수 있습니다.

11.2.1. 알고리즘 개념

GMapping의 이론적 개념은 비교적 간단합니다. Rao-Blackwellized 파티클 필터(RBPF) 알고리즘을 기반으로 한 오픈 소스 SLAM 알고리즘입니다. GMapping 알고리즘은 위치 측정과 매핑 과정을 분리하여 진행합니다. 먼저 파티클 필터 알고리즘을 통해 위치 측정을 수행하고, 도출된 파티클을 생성된 지도와 스캔 매칭(scan matching)하여 지속적으로 마일리지 오차를 수정하고 지도에 새로운 스캔을 추가하는 방식입니다.

GMapping은 RBPF 알고리즘을 기초로 두 가지 부분이 개선되었습니다 (제안 분포와 선택적 리샘플링)

GMapping 알고리즘의 장점은 실시간으로 실내 지도를 구축할 수 있다는 것입니다. 또한 넓지 않은 환경에서 지도를 구축하는 데 필요한 계산량이 적고 이에 대한 결과의 정확도가 높다는 것입니다.

Hector SLAM에 비해 라이다 주파수에 대한 요구가 낮고 결과 값이 견고한 편입니다. (Hector는 로봇이 빠르게 회전할 때 매칭 오류가 생기기 쉽고 구축한 지도에 위치가 오표기되는 경우가 있을 수 있습니다. 이는 최적화 알고리즘에 에러가 발생하기 쉽기 때문입니다)

Cartographer와 비교했을 때, 좁은 환경에서 매핑할 때 GMapping은 많은 파티클을 필요로 하지 않으며, 루프 탐지가 없기 때문에 계산량이 적고 정확도에도 큰 차이가 없습니다.

GMapping은 차량의 미터기(mileage) 정보를 효과적으로 사용합니다. 이것은 GMapping가 라이다 주파수에 대한 요구가 낮은 이유이기도 합니다. 주행거리 기록계는 로봇의 최초 위치 정보를 제공할 수 있습니다.

Hector와 Cartographer의 설계 의도는 로봇의 평면이동 위치 측정과 매핑 이슈를 해결하기 위한 것이 아니기 때문입니다. Hector는 주로 재난 구조 등과 같이 지면이 평평하지 않은 상황에 사용되기 때문에 주행거리 기록계를 사용할 수 없습니다. Cartographer는 휴대용 라이다로 SLAM 프로세스를 수행하기 위해 설계되었기 때문에 사용 가능한 주행거리 기록계가 없습니다.

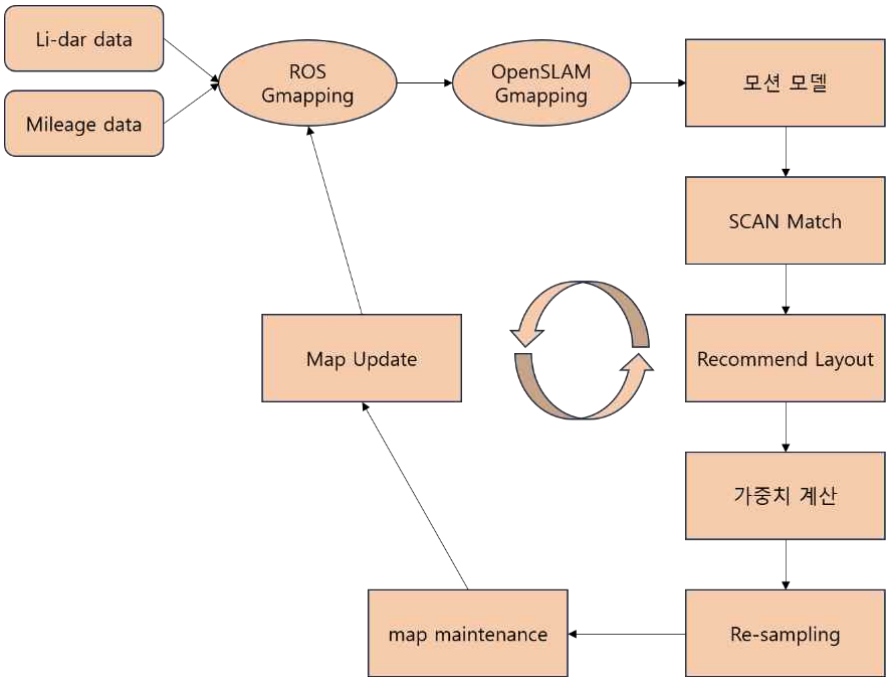
GMapping 알고리즘의 단점은 환경이 커질수록 필요한 파티클의 수가 증가한다는 점입니다. 각 파티클 지도를 가지고 있기 때문에 대규모 지도를 구축할 때는 메모리와 계산량도 증가할 것입니다. 따라서 대규모 환경의 매핑에는 적합하지 않습니다. 또한 루프 탐색 기능이 없기 때문에 루프를 닫을 때 지도의 위치가 잘못될 수 있습니다. 파티클을 추가 생성하여 지도를 닫을 수는 있지만, 이는 계산량과 메모리 사용량의 증가에 대비한 것입니다.

따라서 Cartographer처럼 대규모 지도를 구축할 수 없습니다. 이론상으로는 수만 제곱미터 범주의 지도를 생성할 수 있다고 설명되어 있지만, 실제로는 수천 제곱미터 범주부터 오류가 발생할 수 있음이 증명되었습니다.

GMapping과 Cartographer는 각각 필터 기반 SLAM과 최적화 기반 SLAM으로 두 알고리즘 모두 시간과 공간 복잡성의 균형과 관련이 있습니다. GMapping은 시간 복잡성을 보장하기 위해 공간 복잡성을 포기하기에 이는 대규모 환경의 지도 구축에는 적합하지 않다. 예를 들어, 200m x 200m의 지도를 구축한다고 가정했을때, 5cm 정도의 그리드 해상도를 선택합니다. 그렇다면 한 개의 그리드가 16MB의 메모리를 필요로 합니다. 100개의 파티클이 있다면 1,600MB 즉, 1.6GB의 메모리를 필요로 하게됩니다. 만일 500m x 500m의 사이즈일 때 파티클 수가 200개라는 계산이 나오는데 컴퓨터는 당연히 과부하가 올 수 있습니다. Cartographer 알고리즘을 살펴보면 최적화를 위해 지도에서 하나의 파티클만 사용하는 것과 마찬가지이므로 저장 공간은 GMapping보다 훨씬 작지만 계산량은 비교적으로 더 많습니다.

즉 일반적인 노트북으로는 좋은 지도를 생성하기는 어렵거나 실행 자체가 어려울 수 있습니다. 최적화 그래프에는 복잡한 행렬 연산이 필요하기 때문에 Google에서는 ceres 라이브러리를 개발한 것입니다.

GMapping의 알고리즘은 아래와 같습니다.



위 그림에서 확인할 수 있듯이, ROS의 slam_GMapping 소프트웨어 패키지는 오픈소스 openslam_GMapping 알고리즘 패키지에 속하게됩니다. 즉, ROS의 GMapping 소프트웨어 패키지는 openslam_GMapping 알고리즘 파일집을 ROS로 캡슐화한 것입니다.

GMapping 관련 소스 코드 및 WIKI 주소:

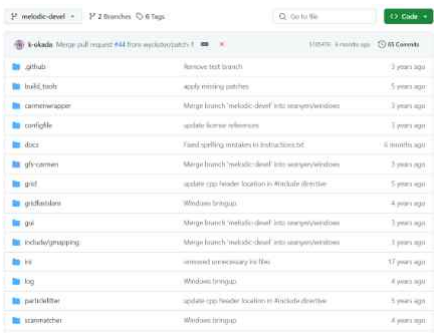
• GMapping ROS WIKI: http://wiki.ros.org/GMapping

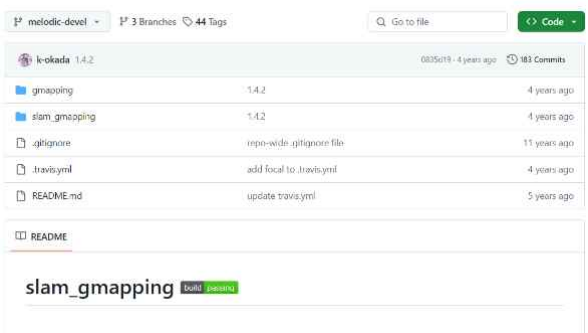
• slam_GMapping 소프트웨어 패키지:

https://github.com/ros-perception/slam_GMapping

• openslam_GMapping 오픈소스 알고리즘:

https://github.com/ros-perception/openslam_GMapping





11.2.2. 조작 가이드

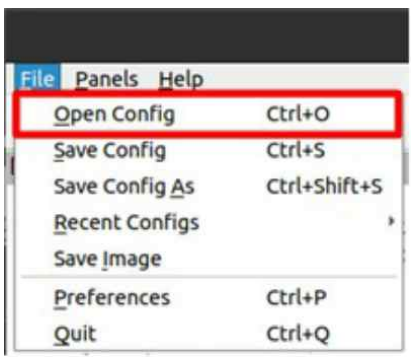
먼저 ssh명령어로 로봇에 연결하고 cilab_slam 패키지의 map.launch 드라이버 파일을 실행하여 GMapping 매핑 드라이버를 실행합니다.

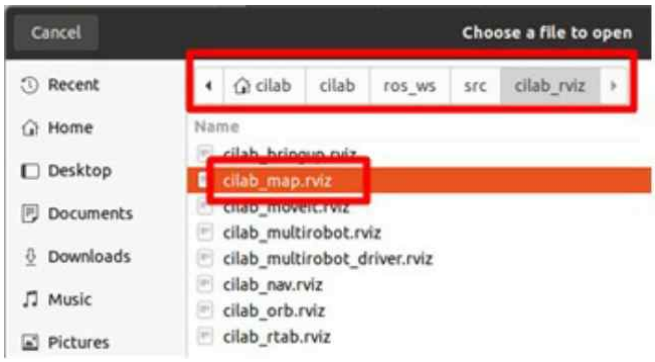
$ roslaunch cilab_slam map.launch slam_method:=GMapping

가상 프로그램에서 rviz 도구를 실행합니다.

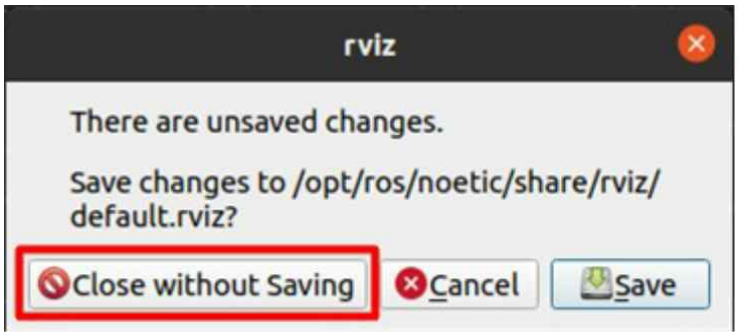
$ rosrun rviz rviz

rviz 도구에 가상 프로그램의 설정 파일을 추가하고, rviz 좌측 상단의 "file/Open Config"를 클릭합니다. 이미지에서 보이는 것처럼 폴더 내 rviz 파일을 찾아 더블 클릭하여 불러오면 아래 화면이 나타납니다.





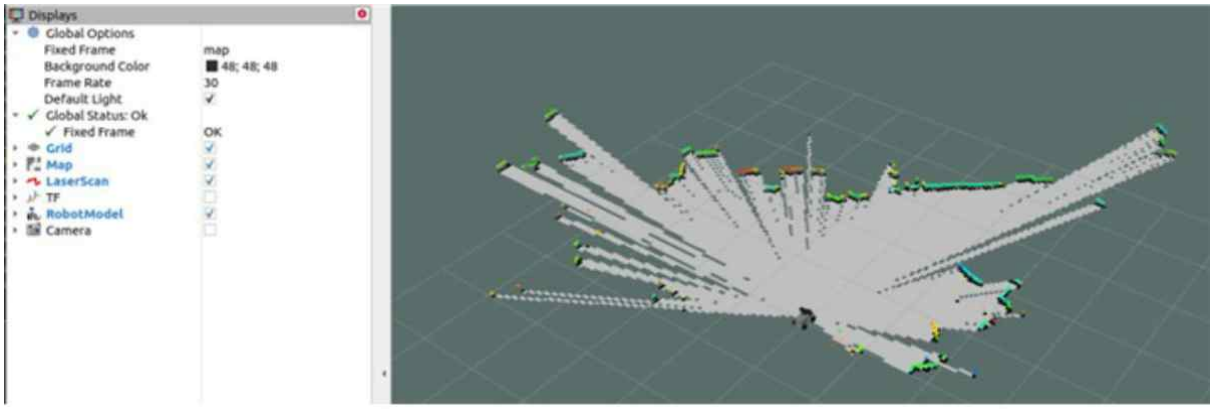
"Close without Saving"을 선택하면 매핑 화면이 나타납니다.

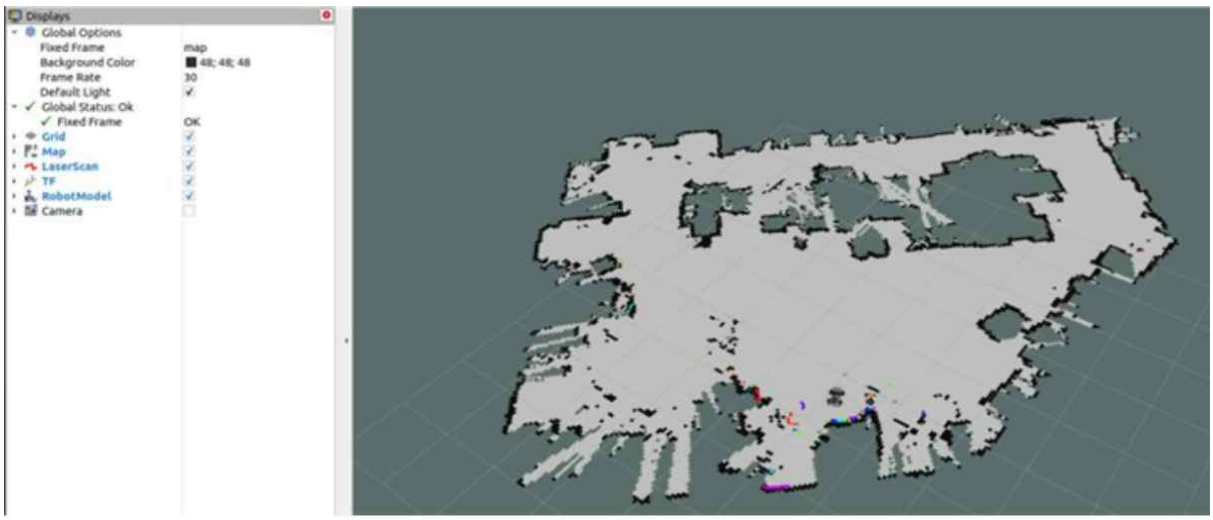


키보드 제어 노드를 실행하여 로봇을 컨트롤하면서 매핑을 실행합니다.

$ roslaunch cilab_teleop keyboard.launch

로봇을 이동하면서 매핑을 진행할 때 로봇의 이동 속도를 적당하게 조절하는 것이 좋습니다. 로봇의 이동 속도가 느릴수록 이동거리 오차가 상대적으로 적으며 더 좋은 매핑 효과를 기대할 수 있습니다. RVIZ에 표시되는 지도는 로봇의 이동에 따라 지속적으로 증가되며 전체 환경에 대한 매핑이 진행됩니다.





매핑이 완료되면 지도 데이터는 차량 내 메모리에 저장되며 드라이버가 종료될 때 데이터가 동시에 연결이 해제됩니다. 로봇의 내비게이션에 사용하기 위해 그리드 지도를 디스크에 저장해야 합니다. map_server 패키지를 통해 지도를 저장할 수 있습니다.

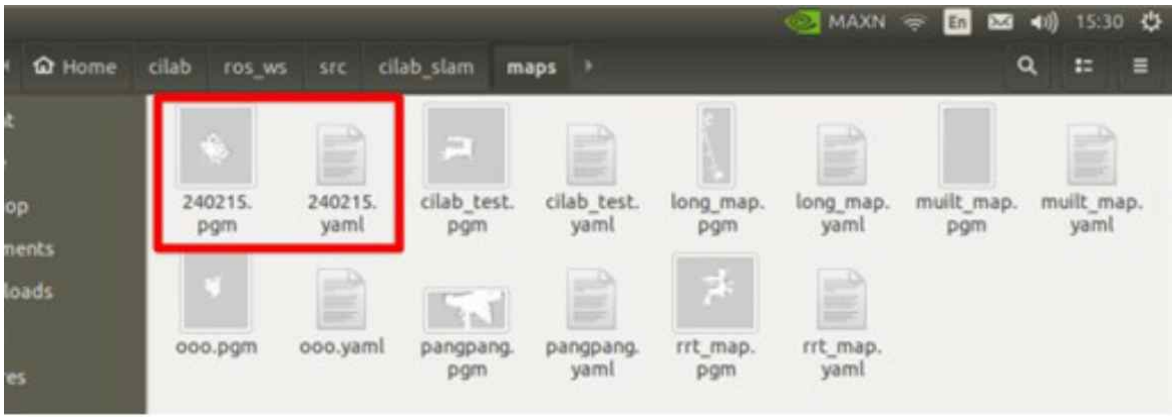
먼저 ssh 명령어로 로봇에 연결하고 아래 launch 파일을 실행합니다.

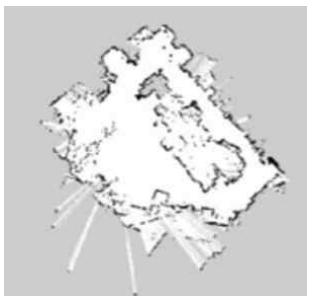
지도는 차량의 cilab_slam 패키지의 maps 폴더에 저장되며 파일 이름은 사용자 임의로 설정할 수 있습니다.

$ roslaunch cilab_slam save.launch name:=xxx (사용자정의)

성공적으로 저장된 후에maps 폴더에 들어가면 xxx.pgm 및 xxx.yaml 두 개의 파일로 저장된 지도 파일을 확인할 수 있습니다. 지도 파일에 대한 상세한 설명은 아래를 참조하길 바랍니다.

```
cilab@cilab:~$ cd cilab/ros_ws/src/cilab_slam/maps/
cilab@cilab:~/cilab/ros_ws/src/cilab_slam/maps$ ls
cartographer.pgm    hector.yaml         long_map.pgm     pangpang.yaml
cartographer.yaml   kartomapping.pgm    long_map.yaml    rrt_map.pgm
cilab_test.pgm      kartomapping.yaml   muilt_map.pgm    rrt_map.yaml
cilab_test.yaml     karto.pgm           muilt_map.yaml
hector.pgm          karto.yaml          pangpang.pgm
cilab@cilab:~/cilab/ros_ws/src/cilab_slam/maps$
```



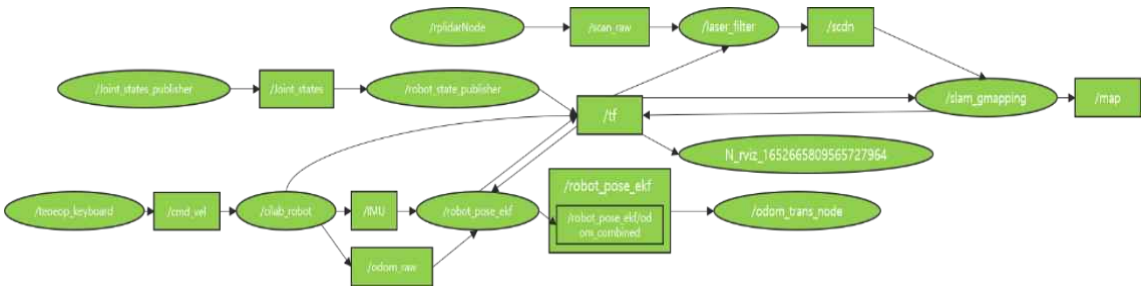


Open
240215.yaml

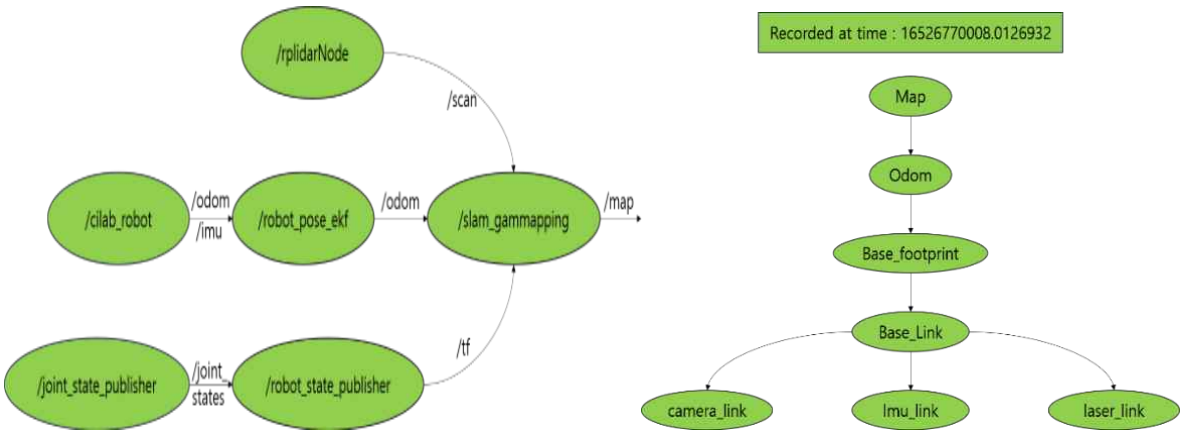
```
image: /home/cilab/cilab/ros_ws/src/cilab_slam/maps/240215.pgm
resolution: 0.050000
origin: [-12.200000, -13.800000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

11.2.3. 알고리즘 설명

먼저 rqt_graph 도구로 계산 그래프를 확보합니다. 그런 다음 아래 이미지와 같이 핸들링 드라이버는 /cmd_vel 토픽을 통해 컨트롤 드라이버와 통신하며, 베이스 컨트롤 드라이버는 인코딩 주행거리 기록계와 IMU 데이터를 robot_pose_ekf 드라이버로 전달합니다. EKF로 융합된 TF 정보 /odom은 GMapping 매핑 드라이버로 입력되고 라이다 드라이버는 /scan 토픽을 통해 데이터를 GMapping 매핑 드라이버에 입력하고, urdf 해석 데이터인 /joint_state_publisher는 정적 /tf_static을 통해 로봇의 각 센서 좌표계 관계를 GMapping 매핑 드라이버로 입력합니다. GMapping 매핑은 이러한 데이터를 사용하여 매핑을 진행하고, 해당 토픽에 지도를 나타냅니다.



아래는 주요 메인 드라이버의 데이터 흐름 및 각 토픽의 연결상황입니다.



매핑 과정에서의 tf 상태는 아래와 같습니다. 여기서 라이다와 로봇 간의 정적 tf 데이커 관계는 base_footprint -> base_link->base_laser_link이며, urdf 해석 드라이버 /robot_state_publisher에 의해 유지됩니다. 휠 주행 거리 기록계 및 IMU 센서 데이터는 /robot_pose_ekf에서 융합된 뒤 /odom에서 제공된 동적 tf 관계 base_footprint를 보여줍니다. 이는 /robot_pose_ekf에 의해 유지되고, 지도와 휠

주행거리 기록계 사이의 동적 tf 관계 map -> odom은 GMapping 드라이버에 의해 유지됩니다. GMapping에서 유지하는 map -> odom의 tf 관계는 사실상 휠 주행거리 기록계의 누적 오차를 동적으로 수정한 것과 같다고 볼 수 있습니다.

11.2.4. 파라미터 설명

파라미터 설명은 다음을 참고하길 바랍니다

throttle_scans: 알고리즘이 데이터를 한 번 처리하는 데 필요한 스캔 횟수.

(숫자가 높게 설정될수록 더 많은 스캔을 건너뛸 수 있음)

base_frame: 로봇 기준 좌표계 명칭
map_frame: 지도 좌표계 명칭.
odom_frame: 주행거리 기록계 좌표계 명칭
map_update_interval: 두 번 업데이트 사이의 시간 간격(초 단위)_이 숫자를 낮추면 더 빈번하게 업데이트되어 계산 부하를 증가시키는 결과를 초래할 수 있음.
maxUrange: 라이다의 최대 사용 가능 범위. (라이다 최대 스캔 반경 값)
maxRange: 센서의 최대 범위. 센서 범위 내에 장애물이 없는 영역이 지도의 자유 공간으로 표시되어야 함.
(maxUrange < 실제 센서의 최대 범위 <= maxRange로 설정)
Sigma: 그리드 앤드 포인트 매칭에 사용되는 Sigma 값.
kernelSize: 라이다 매칭 관계를 찾기 위한 메인코어 사이즈.
Lstep: 평행 이동의 최적화 스텝 사이즈.
minimumScore: 스캔 매칭 결과를 좋음으로 간주하는 최소 범위. 제한된 범위(ex. 5m)에서 라이다 스캐너를 사용할 때 넓은 공간에서 패스 측정하는 것을 피할 수 있음. 점수가 600 이상인 경우에 패스 측정 이슈를 해결하기 위해 50으로 조절해볼 수 있음.
srr: 선형 방정식에서 주행거리 기록계 선형 오차.
srt: 회전 방정식에서 주행거리 기록계 선형 오차.
str: 선형 방정식에서 주행거리 기록계 회전 오차.
srr: 마일리지 리코더 회전 오차.
linearUpdate: 로봇이 해당 거리를 평행 이동할 때마다 스캔 업데이트를 수행
angularUpdate: 로봇이 해당 각도를 회전할 때마다 스캔 업데이트를 수행
temporalUpdate: 마지막으로 처리된 스캔이 업데이트 시간 이전(초 단위)이면 스캔을 처리함. 음수 값(-1)일 때 시간 기반 업데이트를 비활성화
resampleThreshold: Neff에 기반한 리샘플링 threshold value.
particles: 필터 내의 샘플 파티클 수
xmin: 디폴트 지도의 x축 최소 크기(m)
ymin: 디폴트 지도의 y축 최소 크기(m)
xmax: 디폴트 지도의 x축 최대 크기(m)
ymax: 디폴트 지도의 y축 최대 크기(m)
delta: 지도 해상도(각 점유 그리드의 길이(m))
llsamplerange: 평행 이동 샘플 범위 영역.
llsamplestep: 평행 이동 샘플 단계의 영역.

transform_publish_period: 전환 표시 전의 시간 간격(초 단위). 전환주파수를 비활성화는 0으로 설정.
occ_thresh: 영사 점유 비율 임계값. 높은 비율값은 확보된 것으로 간주 (즉 결과 sensor_msgs/LaserScan에서 100으로 설정)

11.3. 지도 설명

앞서 GMapping 알고리즘을 통해 지도를 구축하였고, 본 챕터에서는 지도 서비스 기능과 지도 데이터에 대해서 설명합니다.

11.3.1. 지도 서비스

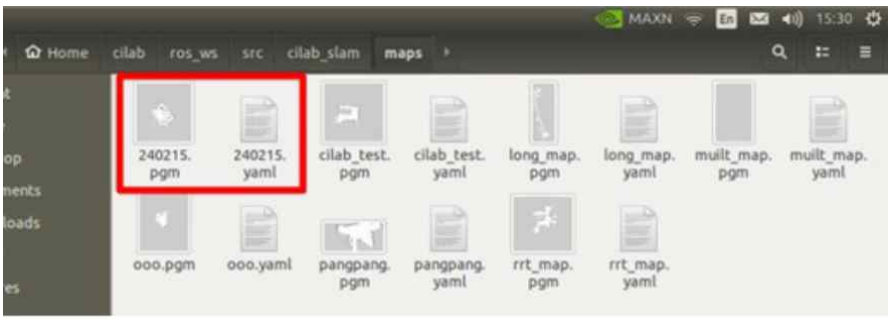
map_server 패키지에는 map_saver와 map_server라는 두 개의 드라이버가 세팅되어 있습니다. map_saver는 그리드 지도를 디스크에 저장하는 역할을 하고, map_server는 디스크에 저장되어 있는 그리드 지도를 읽고 서비스 형태로 제공하는 데 사용됩니다. map_saver 드라이버는 map(nav_msgs/OccupancyGrid) 토픽을 열람하여 지도 파일을 생성하는 데 사용됩니다. map_server 드라이버가 게시한 map_metadata (nav_msgs/MapMetaData) 토픽은 지도의 원형(Origin) 데이터이며, map(nav_msgs/OccupancyGrid) 토픽은 지도 데이터입니다. static_map(nav_msgs/GetMap) 서비스를 통해 맵에 대한 데이터를 얻을 수 있습니다.

해당 지도 서비스 패키지에 대한 자세한 정보는 아래 wiki를 참조해주시기 바랍니다.

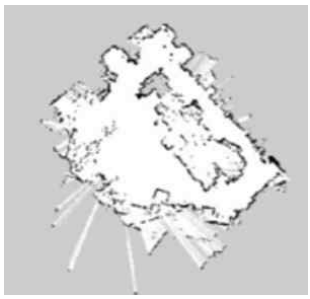
링크 : http://wiki.ros.org/map_server

11.3.2. 지도 정보

map_saver 드라이버를 실행해 지도를 저장하면 지정된 경로에 xxx.pgm 및 xxx.yaml 형식의 파일이 생성됩니다.



xxx.pgm 파일은 이미지 파일입니다. 지도의 해상도는 0에서 255 사이의 그레이스케일(grayscale) 값으로 나타낼 수 있습니다. 다음과 같이 일반적인 이미지 형식인 jpg, png 형식으로 열람이 가능하며, 그래픽 편집 도구로 지도를 수정하거나 다시 그릴 수 있습니다.



xxx.yaml 파일은 지도의 원형 데이터 정보를 저장합니다. 아래와 같은 형식으로 이미지 정보를 작성하는데 사용됩니다.

image: 데이터를 이미지 파일 경로. 기술된 이미지 경로는 절대 경로일 수도 있고, 상대 경로일 수도 있음.

resolution: 지도의 해상도 (meter/pixel)

origin: 지도의 시작 상태 (x, y, yaw), ,yaw는 편항각.

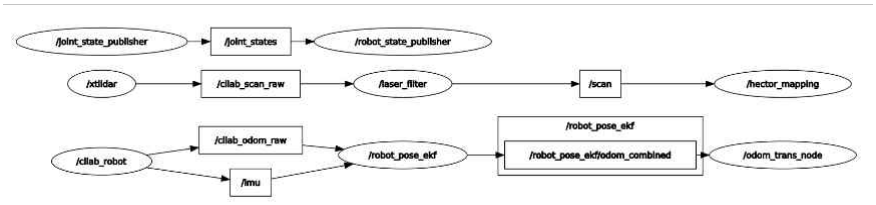
negate: 흰색/검은색의 미확보/확보 지역 반전 인식 여부

occupied_thresh: 점유 확률이 보다 큰 픽셀은 완전히 확보된 것으로 간주 (값 [0,1])

free_thresh: 점유 확률이 보다 작은 픽셀은 미확인으로 간주 (값 [0,1])

11.4. Hector 매핑

Hector 패키지는 가우스 뉴턴 방법을 사용하며, 주행거리 기록계 데이터가 아닌 라이다의 정보만으로도 지도를 구축할 수 있습니다. 본 패키지는 공중 드론이나 휴대용 매핑 장치 및 특수 로봇 등에서 효과적으로 사용됩니다.



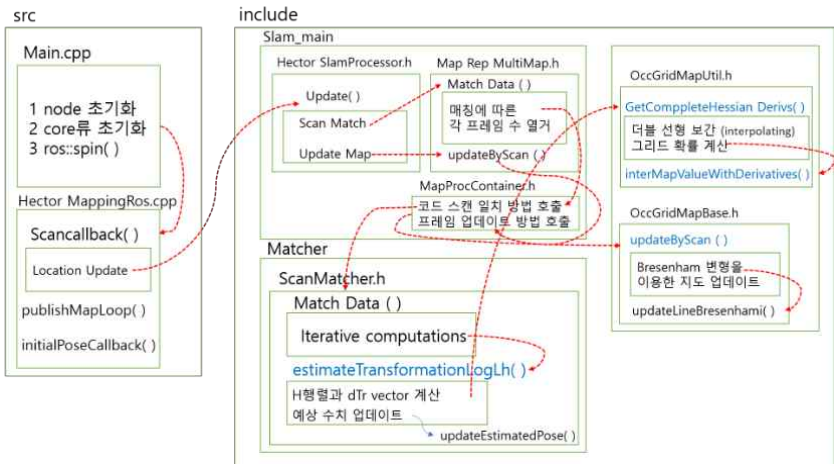
11.4.1. 알고리즘 개요

Hector SLAM은 가우스 뉴턴 방법을 사용하여 스캔 매칭 문제를 해결하기에, 센서에 대한 요구가 비교적 높습니다.

Hector SLAM의 장점은 주행거리 기록계가 필요하지 않다는 것이며, 이를 통해 공중 드론에 의한 지도 구축이나 지상 차량이 평탄하지 않은 지역에서 지도 구축이 가능합니다. 수집한 지도를 통해 라이다 포인트 배열을 최적화하고, 지도 내에서 라이다 포인트의 표현과 점유 그리드의 확률을 측정합니다. 가우스 뉴턴 방법을 사용하여 스캔 매칭 문제를 해결하여 라이다 포인트를 기존 지도로 영사한 변환 데이터(x, y, theta)를 작성할 수 있게 되면서 다중 해상도의 지도를 사용합니다. 내비게이션 동작 중에는 관성 측정 시스템(IMU)을 추가하여 EKF 필터를 사용할 수도 있습니다.

Hector SLAM은 라이다의 업데이트 주파수가 비교적으로 높고 측정 소음이 적어야 합니다. 따라서 매핑 과정에서 로봇 속도를 비교적 완만한 상태로 유지했을때 가장 이상적인 매핑 효과를 얻을 수 있습니다. 또한 주행거리 기록계 데이터가 정확한 경우에 주행거리 정보를 효과적으로 활용할 수 없게 됩니다. 교육용 제품 중 가장 널리 사용되는 A1 라이다이며 실제 데이터 주파수를 5.5Hz에서 15Hz로 증가시켜 Hector 등 매핑 알고리즘의 효과와 관련해 더 자세한 데이터를 획득할 수 있습니다.

Hector의 알고리즘은 아래와 같습니다.



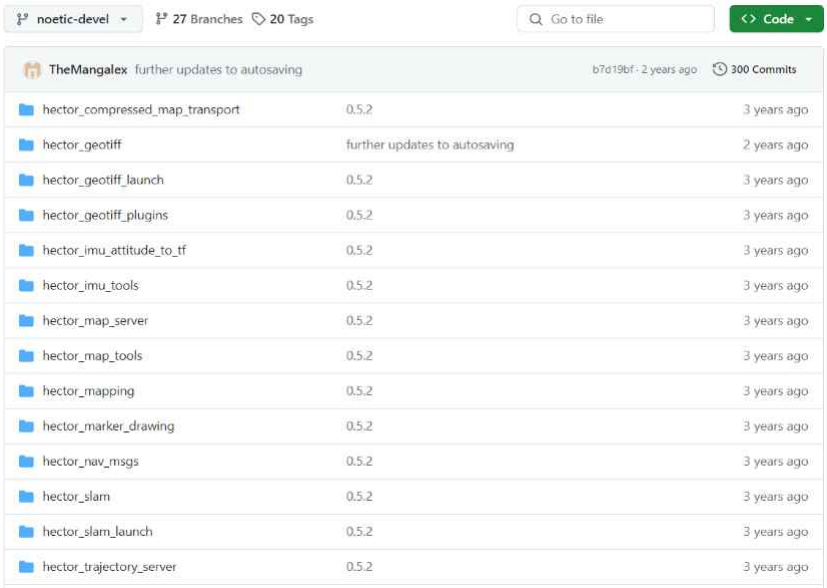
위의 그림은 Hector SLAM의 알고리즘 시스템 구성뿐만 아니라 Hector의 예제코드 구조도이기도 합니다. Hector SLAM 알고리즘 소스 코드를 공부할 때도 해당 구조를 참고하여 예제코드 정리 및 학습을 진행할 수 있습니다.

HectorSLAM 관련 소스 코드 및 WIKI 주소:

• Hector Mapping ROS Wiki: http://wiki.ros.org/hector_mapping

• Hector_slam 소프트웨어 파일집:

https://github.com/tu-darmstadt-ros-pkg/hector_slam



11.4.2. 조작 가이드

작동 방법은 GMAPPING과 유사합니다. 먼저 SSHssh 명령어를 이용해 로봇에 연결하고 로봇에서 cilab_slam 구동 패키지 내의 map 드라이버를 실행하고 상기 설명에 유의하여 아래 명령어로 Hector를 구동합니다.

$ roslaunch cilab_slam map.launch slam_methods:=hector

이 외 작업은 GMapping과 동일하므로 자세한 설명은 GMapping을 참고해주시기 바랍니다.

11.4.3. 데이터 설명

• base_frame: 로봇 기준 좌표계 명칭

• map_frame: 지도 좌표계 명칭

• odom_frame: 주행거리 기록계 좌표 명칭

• map_resolution: 지도의 해상도 [m], 격자 가장자리 길이

• map_size: 지도의 크기 [각 축의 픽셀 수]. 지도는 정사각형이며 (map_size * map_size) 개의 격자 셀을 가지고 있습니다.

• map_start_x: 지도 프레임의 원점[0.0, 1.0]의 x축이 그리드에 대응되는 위치를 나타냅니다. (0.5는 중앙을 의미)

• map_start_y: 지도 프레임의 원점[0.0, 1.0]의 y축이 그리드에 대응되는 위치를 나타냅니다. (0.5는 중앙을 의미)

• map_update_distance_thresh: 지도 업데이트의 threshold value [m]. 지도의 최근 업데이트 이후 플랫폼이 반드시 미터 단위로 해당 거리를 이동시켜야 합니다. 혹은 map_update_angle_thresh 파라미터가 정의된 각도로 변화해야 합니다.

• map_update_angle_thresh: 지도 업데이트의 각도 threshold value [rad]. 지도의 최근 업데이트 이후에 map_update_distance_thresh 파라미터의 지정 각도에 이르기까지 플랫폼이 반드시 파라미터가 나타내는 각도만큼 이동해야 합니다.

• map_pub_period: 지도 업로드 주기 [s]

• map_multi_res_levels: 다중 해상도 그리드급 수량

• update_factor_free: [0.0, 1.0] .범위 내 비어 있는 단원을 업데이트하기 위한 영사 업데이트 팩터 값. (0.5는 변화가 없음을 의미)

• update_factor_occupied: [0.0, 1.0] .범위 내 점유 단원을 업데이트하기 위한 영사 업데이트 팩터 값. (0.5는 변화가 없음을 의미)

• laser_min_dist: 시스템이 사용할 laser scan 끝점의 최소 거리 [m]. 이 값보다 가까운 스캔 포인트는 무시

• laser_max_dist: 시스템이 사용할 laser scan 끝점의 최대 거리 [m]. 이 값보다 먼 스캔 포인트는 무시

• laser_z_min_value: 시스템이 사용할 laser scan점이 laser scan 기기에 대한 최소 높이[m], 이 값보다 낮은 스캔 포인트는 무시

• laser_z_max_value: 시스템이 사용할 laser scan점이 laser scan 기기에 대한 최대 높이 [m]. 이 값보다 높은 스캔 포인트는 무시

• pub_map_odom_transform: 시스템에서 map -> odom 변환을 게시해야 하는지 여부를 결정

• output_timing: 각 laser scan을 처리하기 위해 ROS_INFO를 통해 타이밍 정보 출력.

• scan_subscriber_queue_size: 스캔 구독 서버의 진열 크기. 로그 파일이 실시간 속도보다 빠르게 hector_mapping으로 재생되는 경우 더 높은 값(예: 50)으로 설정

• pub_map_scanmatch_transform: 스캔매처 영사 변환을 tf로 게시해야 하는지 결정. 프레임 이름은 "tf_map_scanmatch_transform_frame_name" 파라미터에 의해 결정

• tf_map_scanmatch_transform_frame_name: 앞서 언급한 파라미터에 따라 스캔매처를 영사 변환에 게시할 때의 프레임 이름

• occ_thresh: 영사의 점유율 threshold value. 높은 점유율을 가진 단원은 점유된 것으로 간주 (즉, sensor_msgs/LaserScan 결과에 100으로 설정됨)

더 상세한 설명은 공식 사이트를 참조하길 바랍니다.

http://wiki.ros.org/hector mapping.

hector mapping.

This page does not exist yet. What type of page are you trying to create?

Please see EditingTheWiki for guidelines on how our wiki is organized and tips on creating new pages.

1. ROS package or Stack

If you're creating the initial page for a stack or package, please give it the page the same name as the stack or package itself, and use one of the following templates:

- StackTemplate - Template this new page to hold usage documentation for a stack
- PackageTemplate - Template this new page to hold usage documentation for a package

If you're creating additional documentation for a package or stack, feel free to structure them however you like, but please keep them within the namespace of your package (e.g. "ros.org/wiki/my_package/more_details") Create new empty page

2. Tutorials

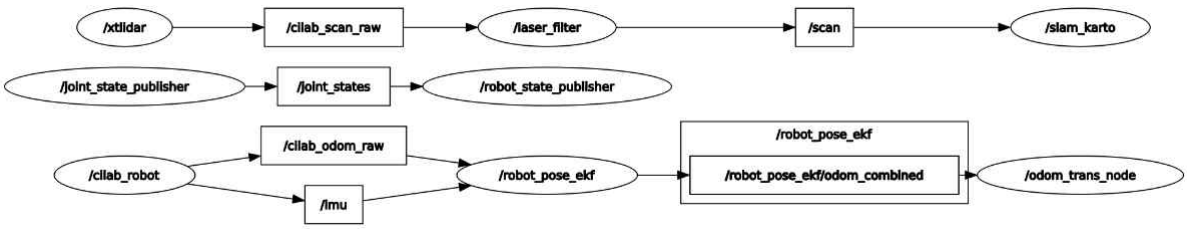
To keep tutorials organized, each stack and package has a link to tutorials at the bottom of the page. If you got here via that link please start a list of tutorials with this template:

- TutorialIndexTemplate

11.5. Karto 매핑

Karto SLAM은 효과적인 로봇 매핑을 위해 Karto 알고리즘을 사용합니다. karto salm은 그래프 최적화를 기반으로 한 알고리즘의 방법으로, 그래프의 평균치를 이용하여 지도를 표현합니다. 각 드라이버는 로봇 경로의 위치와 센서 측정 데이터 그룹을 나타냅니다. 화살표로 표시된 연결은 연속적인 로봇 위치 간의 움직임을 의미합니다. 새로운 데이터가 추가될 때마다 지도는 공간상의 노드 화살표 제약 조건에 따라 계산을 업데이트합니다. Karto SLAM은 주행거리 및 라이다 데이터가 요구됩니다.

11.5.1. 알고리즘 개요



Karto SLAM은 그래프 최적화를 기반으로 한 알고리즘으로, 고도로 최적화된 비-반복 Cholesky 분해를 사용하여 sparse 시스템을 통한 decoupling을 하는 방법입니다. 그래프 최적화 방법은 그래프의 평균치를 이용하여 지도를 표현하며, 각 드라이버는 로봇 경로의 위치와 센서 측정 데이터 그룹을 나타냅니다. 새로운 데이터가 추가될 때마다 해당 계산 절차를 업데이트합니다.

Karto SLAM의 ROS는 공간 포즈 조정(Spare Pose Adjustment, SPA)이 스캔 매칭 및 폐쇄 루프 점검과 연관이 있습니다. 즉 매핑상 랜드마크가 많을수록 메모리 수요량이 증가하지만, 그래프 최적화 방법은 다른 방법에 비해 넓은 환경에서의 더 우수한 효과를 자랑합니다. 왜냐하면 그래프 최적화 방법은 포인트 그림(로봇 포즈)만 포함되며 자세가 수정된 후에 지도를 구축하기 때문입니다.

Karto SLAM의 알고리즘 프레임은 아래와 같습니다.

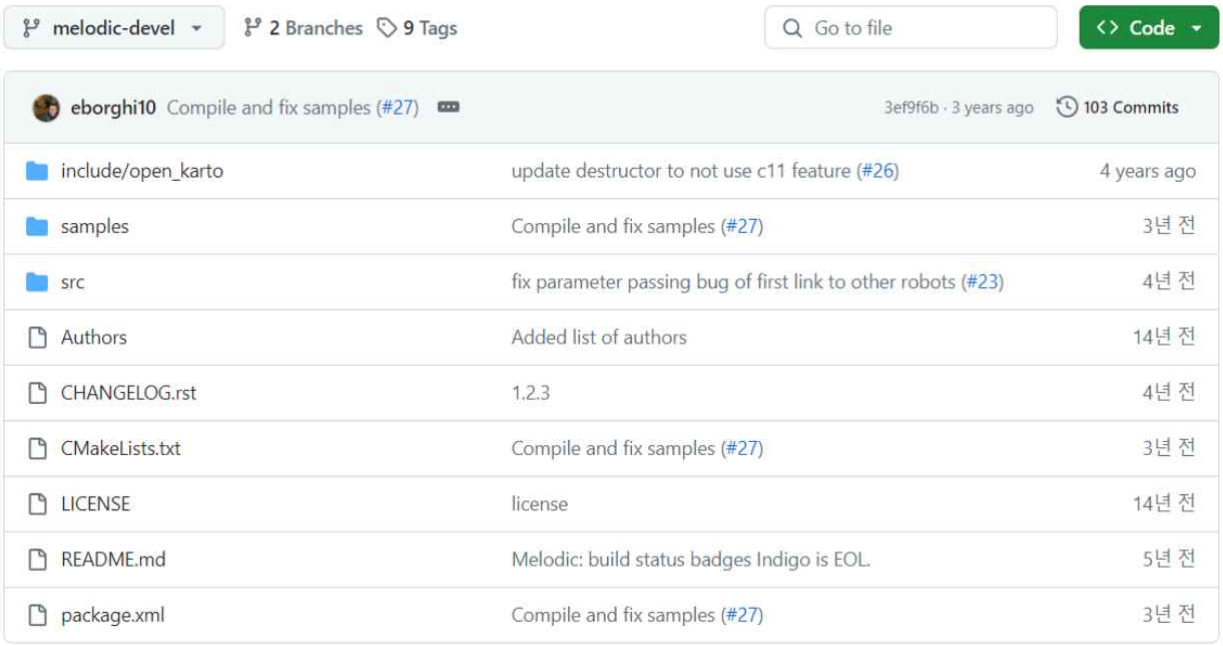
상기 내용을 통해서 알 수 있듯이, 해당 알고리즘의 프로세스는 상당히 간단합니다. 기본적인 실시간 실행 메커니즘으로서 데이터가 들어갈 때마다 처리하며 완료되면 다시 원점으로 돌아갑니다.

Karto SLAM 관련 소스 코드 및 위키 주소:

Karto SLAM ROS Wiki: http://wiki.ros.org/slam_karto

slam_karto 파일집: https://github.com/ros-per

open_karto 오픈소스 알고리즘 https://github.com/ros-perception/open_karto



11.5.2. 조작 가이드

작동 방법은 GMapping과 유사하며 먼저 ssh 명령어를 이용해 로봇에 연결하고 로봇에서 cilab_slam 구동 패키지의 map 노드를 실행하고 아래 명령어를 사용해 karto를 실행합니다.

$ roslaunch cilab_slam map.launch slam_methods:=karto

그 외 작업은 GMapping과 동일하므로 GMapping의 내용을 참고해주시기 바랍니다.

11.5.3. 파라미터 설명

파라미터의 설명은 아래와 같습니다.

• odom_frame: 주행거리 기록계 좌표 명칭
• map_frame: 지도 좌표계 명칭.
• base_frame: 로봇 기본 좌표계 명칭.

• throttle_scans: 알고리즘이 데이터를 한 번 처리하는 데 필요한 스캔 수 (더 높은 숫자로 설정하여 더 많은 스캔을 건너뛸 수 있습니다).
• map_update_interval: 두 번의 업데이트 사이의 시간 간격(초 단위). 이 숫자를 낮추면 점유 그리드를 더 자주 업데이트하게 되어 계산 부담을 증가시킬 수 있습니다.
• resolution: 지도의 해상도(점유 그리드 당 미터 수).
• delta: 지도의 해상도(점유 그리드 당 미터 수). 해상도와 동일하게 정의됩니다. GMapping의 파라미터 명칭과 호환되도록 정의됩니다.
• transform_publish_period: TF를 게시하는 시간 간격(초 단위). TF 라디오를 사용하지 않으려면 0으로 설정해야 합니다.
• use_scan_matching: true로 설정하면 매핑 알고리즘이 스캔 매칭 알고리즘을 사용합니다. 대부분의 실제 상황에서 true로 설정하는 것이 좋습니다. 이로써 매핑 알고리즘이 거리 측정 방법, 스캔 데이터의 소음 및 오류에 대한 캘리브레이션을 진행할 수 있습니다. 그러나 일부 시뮬레이션 환경에서는 스캔 및 거리 측정 데이터가 매우 정확한 경우 스캔 매칭 알고리즘이 나쁜 결과를 가져올 수 있습니다. 이 경우 결과를 개선하기 위해 false로 설정할 수 있습니다.
• use_scan_barycenter: 스캔 끝점의 무게 중심을 사용하여 스캔 간의 거리를 정의합니다.
• minimum_travel_distance: 두 스캔 사이의 최소 거리를 설정합니다. 새로운 스캔의 위치가 이전 스캔의 minimumTravelDistance보다 크면 알고리즘은 새로운 스캔 데이터를 사용합니다. 그렇지 않은 경우, 최소 이동 요구 사항을 충족시킬 수 없습니다. 새로운 면 스캔도 폐기됩니다. 성능상의 이유로, 일반적으로 로봇이 합리적인 거리를 이동한 경우에만 과정 스캔을 수행합니다..
• minimum_travel_heading: 두 스캔 사이의 최소 방향 변경을 설정함. 새로운 스캔 데이터가 이전 스캔의minimum_travel_heading보다 크면 새로운 스캔 데이터를 사용합니다. 그렇지 않은 경우, 최소 이동 거리 요구에 충족되지 못하면 새로운 스캔이 폐기됩니다. 성능을 최적화하기 위해 로봇이 합리적인 거리를 이동한 경우에만 과정 스캔을 수행합니다.
• scan_buffer_size: 매칭을 위해 저장되는 스캔 체인의 길이. scan_buffer_size는 대략 scan_buffer_maximum_scan_distance / minimum_travel_distance로 설정해야 함. 이 아이디어는 스캔 매칭에 약 20m 길이의 영역을 확보하여 스캔 매칭에 사용합니다. 예를 들어, minimum_travel_distance == 0.3m마다 스캔을 추가합니다. 그렇다면 scan_buffer_size는 20 / 0.3 = 67이어야 합니다..
• scan_buffer_maximum_scan_distance: 첫 번째 스캔과 마지막 스캔 사이의 최대 거리로 매칭·저장되는 스캔 체인으로 설정합니다.
• link_match_minimum_response_fine: 해당 응답 값보다 큰 경우에만 스캔을 연결합니다.
• link_scan_maximum_distance: 연결된 스캔 사이의 최대 거리. 관련 반응 수치와 상관없이 거리가 먼 스캔이 연결되지 않습니다.
• loop_search_maximum_distance: 현재 위치와의 거리가 이 값보다 작은 스캔은 폐쇄 루프 매칭으로 간주합니다.
• do_loop_closing: 루프 클로징 매칭 활성화/비활성화.
• loop_match_minimum_chain_size: 폐쇄 루프 탐지가 후보 객체를 찾을 때, 그것은 많은 연결 스캔 중의 일부여야 합니다. 스캔 체인이 이 값보다 작으면 폐쇄 루프를 시도하지 않습니다.
• loop_match_maximum_variance_coarse: 가능한 해결책을 고려할 때, 가능한 루프 클로징의 공분산 값이 이 값보다 작아야 합니다. 이는 대략적인 검색에 적용됩니다.
• loop_match_minimum_response_coarse: 응답이 이 값보다 크면 대략적인 해상도로 순환을 활성화하며 검색을 비활성화 합니다.
• loop_match_minimum_response_fine: 응답이 이 값보다 크면 더 높은 해상도로 순환을 활성화하고 검색을 비활성화 합니다.
• correlation_search_space_dimension: 매처가 사용하는 검색 그리드의 크기를 설정합니다. 검색 그리드 크기:

correlation_search_space_dimension x correlation_search_space_dimension.
• correlation_search_space_resolution: 관련 그리드의 해상도(그리드 크기)를 설정합니다.
• correlation_search_space_smear_deviation: X 및 Y의 값을 사용하여 Smooth point 해독을 하여 더 부드러운 응답을 구축합니다.
• loop_search_space_dimension: 매처가 사용하는 검색 그리드 크기.
• loop_search_space_resolution: 관련 그리드의 해상도(그리드 크기)

• loop_search_space_smear_deviation: X 및 Y의 값을 사용하여 Smooth point 해독을 하여 더 부드러운 응답을 구축합니다.
• distance_variance_penalty: 스캔 매칭 시 마일리지에서 벗어나는 차이에 대한 벌칙. 벌칙은 측정 스캔위치와 측정 거리 자세의 증량 함수입니다. (<1.0).
• angle_variance_penalty: distance_variance_penalty와 동일하게 정의됩니다.
• fine_search_angle_offset: 세밀한 검색 과정에 검색할 각도 범위.
• coarse_search_angle_offset: 대략적인 검색 과정에 검색할 각도 범위
• coarse_angle_resolution: 대략적인 검색 과정에 검색할 각도 해상도
• minimum_angle_penalty: 각도 벌칙의 최소값으로, 점수가 너무 작아지지 않도록 합니다.
• minimum_distance_penalty: 거리 벌칙의 최소값으로, 점수가 너무 작아지지 않도록 합니다.
• use_response_expansion: 초기에 적합한 매칭 항목을 찾지 못한 경우에 검색 공간을 확장할지 여부

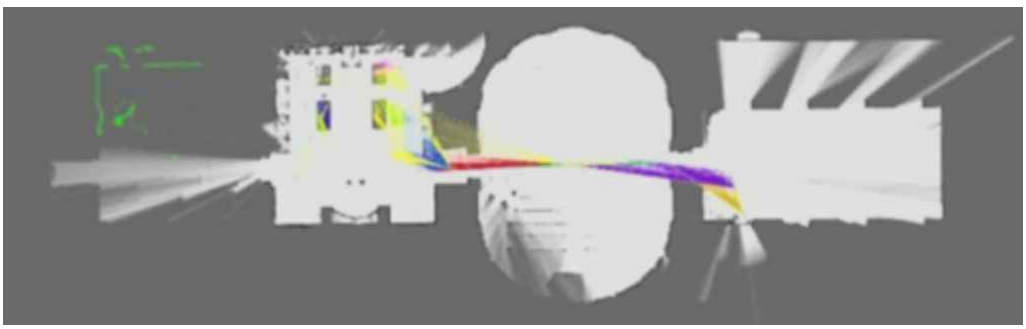
11.6. Cartographer 매핑

본 챕터에서는 Google Cartographer 알고리즘을 사용하여 로봇 매핑을 진행합니다. Cartographer는 그래프 최적화를 기반으로 한 매핑 알고리즘의 일환이며, Karto와 마찬가지로 그래프 최적화 프레임에 속합니다. 그러나 두 알고리즘은 여러 가지 차이점이 있습니다.

예를 들어, Karto는 spa그래프 최적화 방법을 채택하고, Cartographer는 Google의 ceres로 문제 최적화를 구축합니다. Karto의 경우 단일 스레드로 수행되지만, Cartographer는 멀티 스레드 최적화 데이터를 채택합니다. Cartographer도 다중 센서를 활용한 융합 매핑을 지원하며, 라이다, IMU, 주행거리 기록계 등 센서에서 제공되는 데이터를 활용하여 지도를 구축할 수 있습니다.

또한, Cartographer는 그래프 최적화 알고리즘 중 가장 평가가 높은 알고리즘을 기반으로 한 가장 최신의 알고리즘이라고 할 수 있습니다. 코드의 안정성이 높다는 장점이 있으며 매핑과 리로컬라이제이션(relocalization) 기능을 함께 제공하는 소수의 알고리즘 중 하나입니다.

11.6.1. 알고리즘 개요



Cartographer의 주요 이론은 폐쇄 루프 검증을 통해 매핑 과정에서 발생하는 누적 오차를 제거하는 것입니다. 폐쇄 루프 점검에 사용되는 기본 단위는 서브지도(submap)입니다. 서브지도는 일정 수량의 laser scan 데이터로 구성됩니다. laser scan이 해당하는 서브지도에 삽입될 때, 기존 서브지도의 laser scan과 다른 센서 데이터를 기반으로 해당 laser scan이 지도에 있을 최적 위치를 측정합니다.

서브지도는 짧은 시간 동안의 오차 누적이 충분히 작다고 가정하고 시간이 지남에 따라 점점 더 많은 서브지도기 생성되면, 서브지도 간의 누적 오차가 커지게 됩니다. 따라서 이러한 누적 오차를 적절히 제거하기 위해 서브지도의 포지션을 최적화하는 폐쇄 루프 점검이 필요하며 이는 포지션 최적화 문제로 변환됩니다.

서브지도의 구성이 완료되면, 해당 서브지도에 새로운 laser scan이 더 이상 삽입되지 않게 되는 시점에서 해당 서브지도는 폐쇄 루프 점검 단계에 추가되게 됩니다. 해당 점검은 모든 작성 완료된 서브지도를 점검합니다.

지도에 새로운 laser scan이 추가되고 난 후 해당 laser scan의 측정 포지션이 서브지도의 특정 laser scan의 자세와 유사합니다, 하나의 스캔 매치 전략을 통해 폐쇄 루프가 검출됩니다.

Cartographer의 스캔 매칭은 지도에 새롭게 추가된 laser scan의 측정 포지션에서 윈도우를 확보하고, 이 윈도우 내에서 해당 laser scan의 가능한 매칭을 찾게 됩니다. 적합한 매칭을 찾는다면 해당 매칭의 폐쇄 루프 제약을 포지션 최적화 문제에 추가합니다. Cartographer의 핵심은 다중 센서 데이터의 국부 서브지도 생성 및 폐쇄 루프 점검을 위한 스캔 매치 전략의 실현에 있습니다.

Cartographer는 크게 두 부분으로 나눌 수 있습니다.

첫째, 로컬 SLAM(Local SLAM)입니다. Frontend라고도 불리기도 합니다. 주요 미션은 서브지도(Submap)을 구축하고 유지하는 것입니다. 그러나 해당 이슈는 매핑 오차가 시간이 지남에 따라 누적되고, 관련된 파라미터는 /src/cartographer/configuration_files/trajectory_builder_2d.lua 및

/src/cartographer/configuration_files/trajectory_builder_3d.lua에 정의되게 됩니다.

둘째, 글로벌 SLAM(Global SLAM)입니다. Backend라고도 불리기도 합니다. 주요 미션은 Loop Closure를 수행하는 것입니다. 앞에서 설명한 것처럼, 폐쇄 루프 검점은 최적화 이슈이며, 본 매뉴얼에서는 pixel-accurate match 형식으로 표현되었습니다. Branch-and-Bound 접근법(BBA)으로 해결이 가능합니다.

전반적으로 Local Slam은 양질의 서브지도를 생성하고, 글로벌 SLAM 부분은 전반적인 최적화를 수행하며 서로 다른 서브지도를 가장 잘 맞는 자세로 분류합니다.

레이저 데이터는 센서에서 출발해 두 개의 필터를 거쳐 스캔 매칭을 진행하여 서브지도를 구축합니다. 새로운 laser scan이 들어오면 이미 유지 중인 서브지도에 적절한 위치로 삽입되기도 합니다.

삽입되는 최적의 자세를 결정하는 것은 Ceres Scan Matching을 통해 실현되고 측정된 데이터는 주행거리 기록계 및 IMU 데이터와 통합해 다음 시점의 포지션을 측정하는 데 사용됩니다.

11.6.2. 조작 가이드

Cartographer의 작업 방법은 GMapping과 유사합니다. 먼저 ssh 명령어를 이용하여 로봇에 연결하고 로봇 터미널에서 cilab_slam 매핑 구동 패키지 내의 매핑 드라이버를 실행합니다.

$ roslaunch cilab_slam map.launch slam_methods:=cartographer

다른 작업도 GMapping과 동일하며 자세한 설명은 GMapping 파트를 참고해 주시기 바랍니다.

11.6.3. 파라미터 설명

Cartographer의 파라미터는 주로 cilab_cartographer.lua 파일을 통해 로딩됩니다. 파일 위치는 아래와 같습니다.

~/cilab/ros_ws/src/cilab_slam/config.

주요 파라미터에 대한 설명은 아래와 같습니다.

• map_frame: 지도 좌표계의 명칭

• tracking_frame: 경로 추적에 사용되는 좌표계 명칭

• published_frame: 좌표계 변환을 게시하는 좌표계의 명칭

• odom_frame: 주행거리 기록계 좌표 명칭

• provide_odom_frame: 마일리지 리코더 좌표계 변환 게시 여부

• publish_frame_projected_to_2d: 활성화된 경우 게시된 pose가 순수한 2D 자세(롤, 피치 또는 z 변위 없음)로 제한됩니다.

• use_odometry: odom 데이터 사용 여부

• use_nav_sat: 활성화된 경우 토픽 'fix'에서 sensor_msgs/NavSatFix를 구독해야 합니다. Navigation 데이터를 제공하며 전역 SLAM에 정보가 포함해야 합니다.

• use_landmarks: 활성화된 경우 Landmarks 토픽에서

cartographer_ros_msgs/LandmarkList 데이터를 구독해야 합니다. 이 경우 Landmarks를 반드시 제공하고 SLAM에 정보를 포함해야 합니다.

• num_laser_scan: 구독하는 레이저입니다. 토픽의 수

• num_multi_echo_laser_scans: 구독하는 다중 에코 레이저입니다. 토픽의 수

• lookup_transform_timeout_sec: TF로 변환 응답 최대 대기 시간

• submap_publish_period_sec: 서브지도를 게시하는 주기(초 단위)

• pose_publish_period_sec: pose를 게시하는 주기(예: 5e-3의 주파수는 200Hz)

• trajectory_publish_period_sec: 초 단위로 경로 표시를 게시하는 주기 (예: 30e-3는 30밀리초 지속)샘플링 비율 관련 파라미터:

• rangefinder_sampling_ratio: 메시지의 고정된 샘플링 비율

• odometry_sampling_ratio: 마일리지 리코더 메시지의 고정 샘플링 비율

• fixed_frame_sampling_ratio: 고정 좌표계 메시지의 고정 샘플링 비율

• imu_sampling_ratio: IMU 메시지의 고정 샘플링 비율

• landmarks_sampling_ratio: 랜드마크 메시지의 고정 샘플링 비율

• use_response_expansion: 초기에 적합한 매칭 항목을 찾지 못할 경우에 검색 공간을 확장할지 여부.

Cartographer의 구성 가능한 파라미터가 많으며, 여기에서는 하나하나 나열하지 않기로 합니다. Cartographer 알고리즘을 자세히 학습하려면 다음 링크를 참조하여 공식 파일을 참고할 수 있습니다.

• Cartographer GitHub: https://github.com/cartographer-project/cartographer

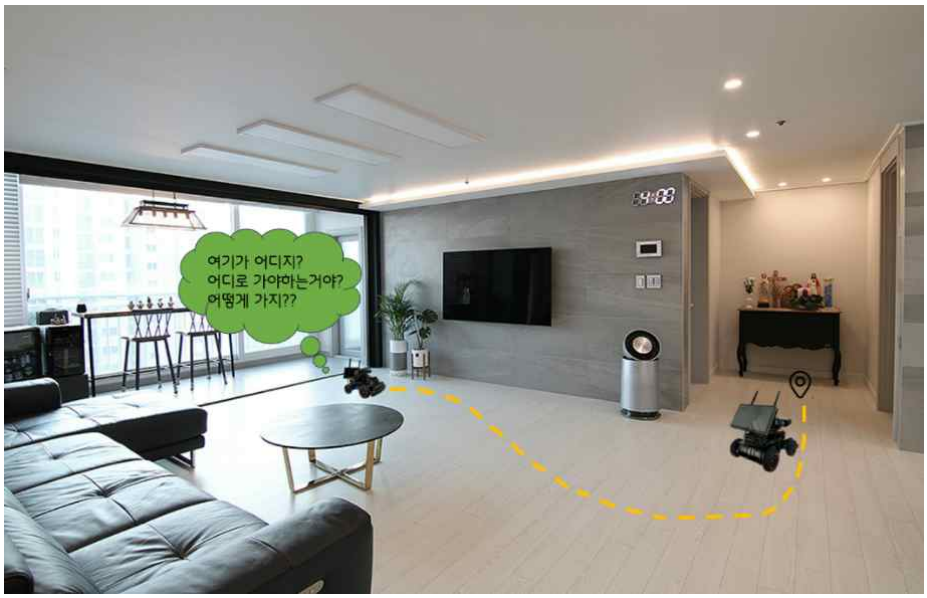
• Cartographer 공식 Doc:

https://google-cartographer.readthedocs.io/en/latest/configuration.html

12. SLAM 내비게이션

12.1. 내비게이션 개요

자율 내비게이션 기능은 A 포인트에서 지정된 B 포인트로 이동하는 이슈를 해결하는 것입니다. 일반적으로 자율 주행이라 함은 로봇 입장에서 “내가 어디에 있는걸까?”, “내가 어디로 가야할까?”, “어떻게 가야할까?”라는 주어진 이슈를 풀어내는 것입니다.



해당 퍼포먼스는 전문적인 수준에서 개발되기 때문에 일반 개발자의 입장에서는 탐색 알고리즘 및 하드웨어 상호작용과 같이 복잡하고 낮은 수준의 구현에 치중하지 않아도 무관합니다. 단지 유지 및 관리만 꾸준히 해주어도 상위수준의 기능에 더 포커스를 맞출 수 있습니다. 각 모듈의 구성파일에서 본체의 관련 매개변수를 적절하게 수정해주는 것만으로 기존 패키지의 2차 개발이 가능해지기도 합니다.

자율 주행 로봇의 내비게이션 기능은 ROS 내비게이션 기능 패키지인 ros-navigation에 기반하여 동작하기 때문에 ROS 시스템에서 가장 중요한 요소라고 볼 수 있습니다. 본 챕터에서는 해당 기능의 동작, 원리분석, 사용에 대해 설명합니다.

12.2. 내비게이션 컨트롤

챕터 11에서 로봇이 매핑을 진행한 후에 2차원의 그리드 맵을 작성에 대해 설명했습니다. 본 챕터에서는 그 이후 구성된 맵을 기반으로 한 매핑, 위치 지정, 경로 계획, 모션 제어, 환경 제어 등의 관련 지식에 대해 설명합니다. SLAM 자율항법 프로세스는 먼저 로봇을 수동으로 제어하여 SLAM 환경을 스캔하고 구축된 지도를 저장합니다. 그리고 사전에 구축된 오프라인 지도를 불러와 SLAM 재배치 모드를 진행하여 로봇의 실시간 모션을 확인합니다.

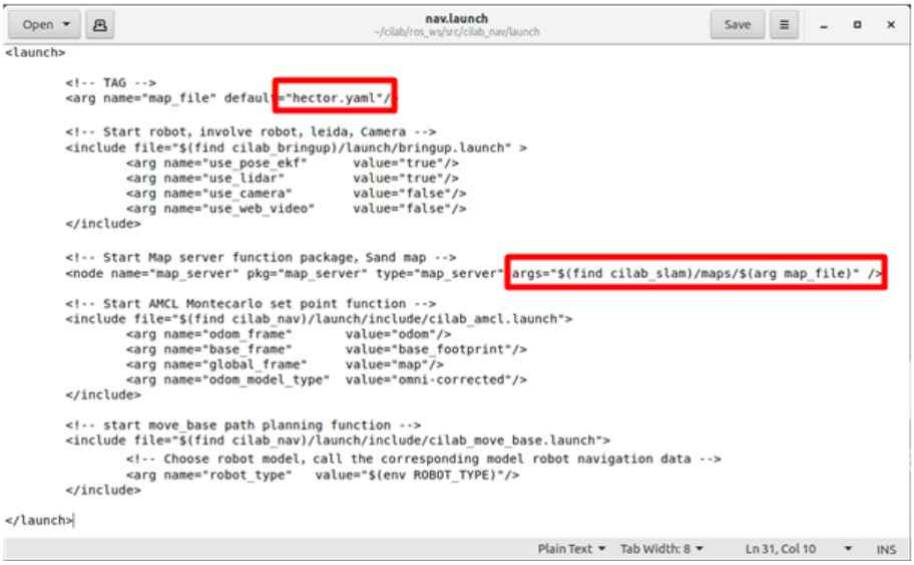
로봇은 라이다와 기타 센서들을 이용해 위치 및 환경을 감지하고, 계획된 경로를 따라 장애물을 회피하며 지정된 포인트로 이동을 하게 됩니다.

로봇 자율주행을 진행하기에 앞서 지도를 생성하고 저장해야합니다. 해당 절차는 wizard를 시작하기 위해 이전에 생성된 2차원 지도를 사용합니다.

cilab_nav 패키지 드라이버를 실행하기 전에 ssh 명령어를 이용해 로봇에 연결하고 아래 명령어를 통해 nav.launch 드라이버 내의 지도파일 경로와 이름을 수정합니다.

cilab@cilab:~$ roscd cilab_nav/launch/
cilab@cilab:~$ vi nav.launch

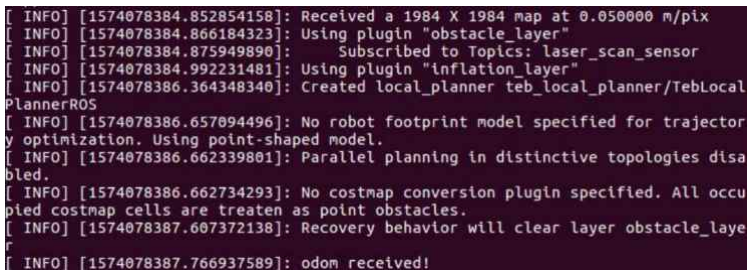
아래 그림과 같이 현재 저장된 지도파일의 경로를 확인할 수 있습니다.



저장 후에 launch파일을 실행하면 수정한 지도로 변경됩니다.
다음으로 로봇에 ssh 명령어를 이용하여 연결하고 cilab_nav 탐색 드라이버 패키지를 실행합니다.

cilab@cilab:~$ roslaunch cilab_nav nav.launch

아래와 같은 프롬프트가 나타나게 되면 정상적인 로딩을 위해 기다려줍니다.

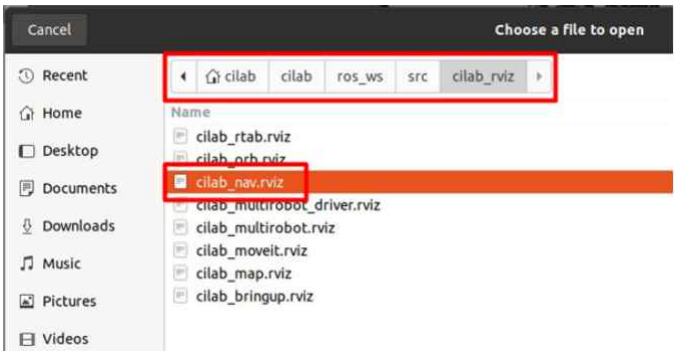


가상 프로그램에서 rviz gui를 실행합니다.

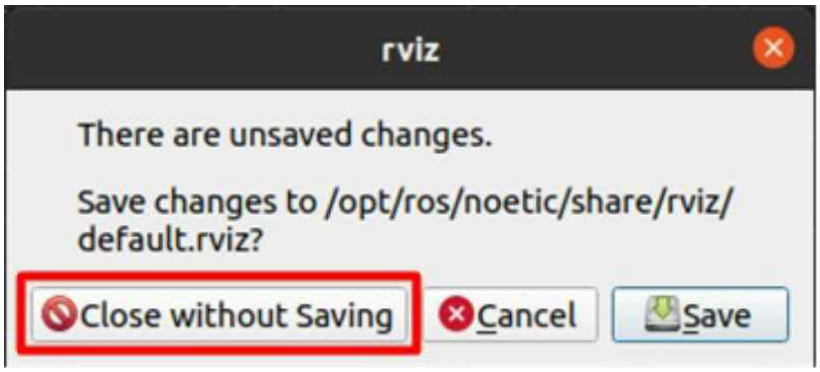
cilab@cilab-pc:~$ rosrun rviz rviz

해당 gui를 실행한 후에 가상 프로그램의 구성 파일을 추가해주고, 왼쪽 상단의 File-Open Config를 클릭합니다.

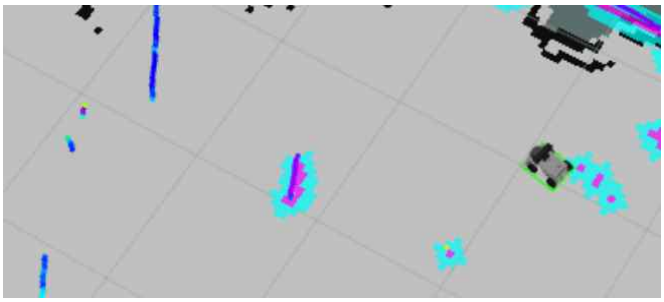
아래 경로로 이동하면 rviz파일이 저장되어 있음을 확인할 수 있습니다.



해당 파일을 더블클릭해 불러오면 아래 팝업팡이 나타나게 되며 Close without Saving을 선택해줍니다.



탐색을 진행하기에 앞서 아래 그림과 같이 실제 장면에서의 로봇의 위치와 방향을 기반으로 하여 로봇의 최초 위치를 보정해주어야 합니다. 상단바에 위치한 도구(tool)중 "2D Pose Estimate"는 로봇 보정을 진행합니다. 화살표 꼬리부분이 로봇의 위치가 되고 머리 부분이 이동 방향이 되게

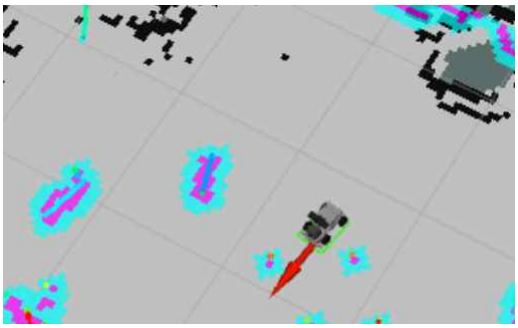


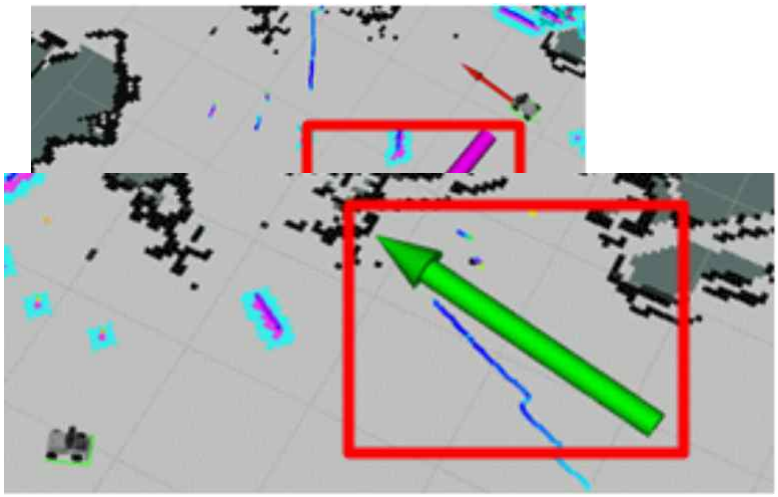
됩니다.

최초 위치를 바로잡은 후에 탐색을 진행합니다. 해당 최초 세팅위치는 일반적으로 매우 정교하며 약간의 편차는 AMCL 알고리즘을 통해 조율하게 됩니다.

Rviz의 "2D Nav Goal"기능을 이용해 목표 포인트를 설정해주면 ROS 로봇이 자동으로 경로상의 장애물을 탐색, 회피하면서 주행을 진행하게 됩니다.

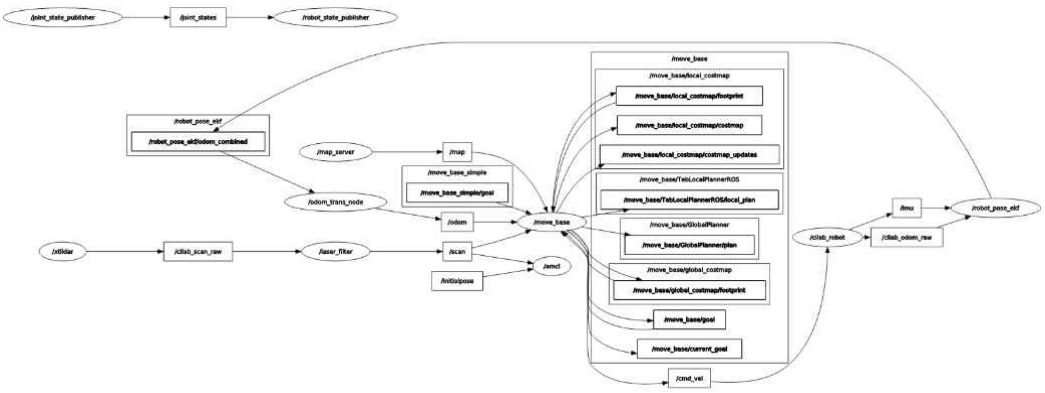
해당 과정중에 새로운 장애물이 나타나 경로를 방해할 경우 로봇은 경로를 다시 계획하고 해당 장애물을 회피하게 됩니다. 지도에 나타나는 초록색은 직접적인 경로이며, 붉은색은 실제 이동 경로에 해당합니다.





이때 아래 명령어를 이용하여 rqt도구를 활용하면서 탐색중의 ROS 데이터 흐름과 tf상태를 확인할 수 있습니다.

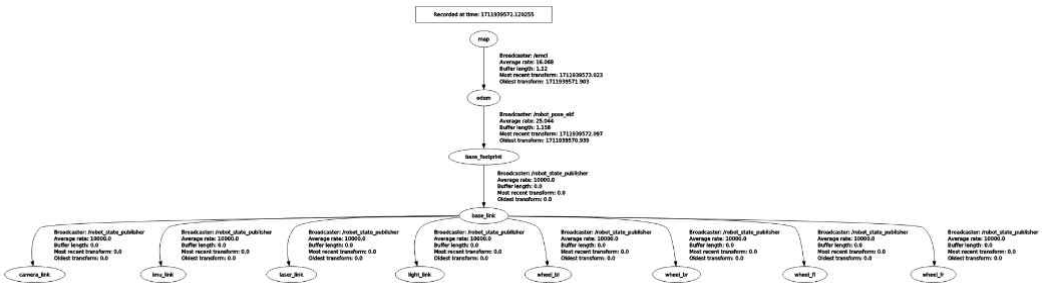
cilab@cilab-pc:~$ rosrun rqt_graph rqt_graph



move_base는 rviz가 표시한 목표 포인트 정보를 수신한 후 속도 토픽인 /cmd_vel를 표시합니다. 제어 드라이버는 /cmd_vel 정보를 받은 후 로봇을 목표 포인트로 이동시킵니다. 인코더 주행거리 기록계와 IMU 데이터는 /robot_pose_ekf 로 전송되어 융합된 tf 정보인/odom_combined을 /move_base에 입력합니다. 라이다는 /scan 토픽을 통해 데이터를 move_base 내비게이션으로 입력하며, /map_server 노드는 현재 내비게이션에 사용되는 정적 지도 /map을 제공합니다. 라이다 노드 /rplidarNode와 지도 서버 /map_server는 수집한 정보를 /amcl에 전송하여 실시간으로 로봇의 위치를 교정하고, urdf 해석 노드는 차량의 각 센서 좌표계 관계를 정적 /tf_static을 통해 내비게이션 노드에 입력합니다. 내비게이션 드라이버에서는 이러한 입력 데이터를 활용하여 목표 포인트로 이동하고 있는 속도 토픽 /cmd_vel을 표시하면서 amcl과 함께 위치 측정을 진행합니다.

아래 명령어를 통해 해당 내비게이션 과정의 tf상태를 확인할 수 있습니다.

cilab@cilab-pc:~$ rosrun rqt_tf_tree rqt_tf_tree



여기서 라이다와 로봇 사이의 정적 tf 관계는 base_footprint->base_link->base_laser_link인데, urdf 해석 노드 /robot_state_publisher에 의해 관리됩니다.

휠 주행거리 기록계와 IMU 센서 데이터는 /robot_pose_ekf를 통해 융합된 뒤 /odom_combined를 표시합니다. 동적인 tf 관계인 odom_combined->base_footprint를 제공하며 /robot_pose_ekf에 의해 유지됩니다. 또한 지도와 휠 주행거리 기록계 사이의 동적 tf 관계인 map->odom은 amcl 매핑에서 유지되게 됩니다.

12.3. 내비게이션 패키지 상세 설명

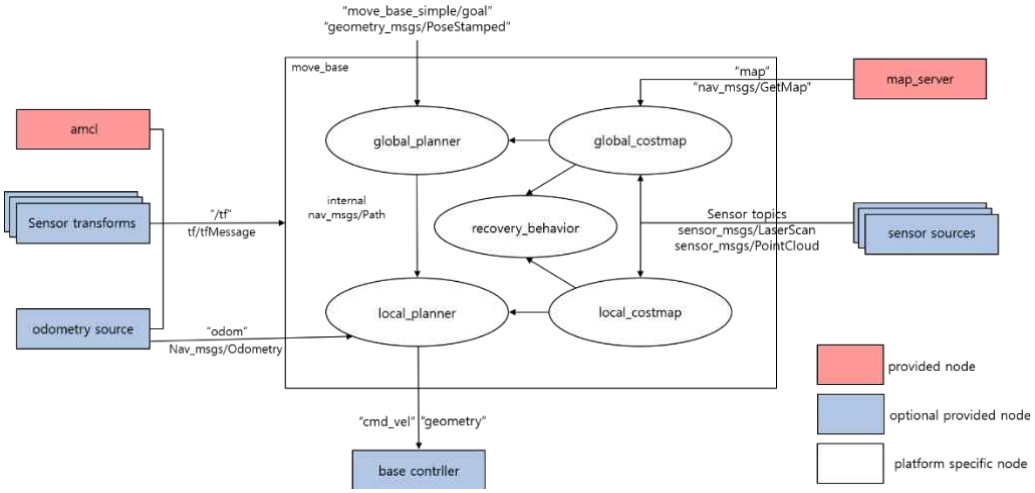
본 챕터에서는 ros-navigation 내비게이션 기능 패키지에 대해 자세히 설명합니다. (원리, 설치 및 사용 방법)

12.3.1. 원리 프레임 설명

내비게이션 시스템은 내비게이션 목표, 로봇 위치 및 지도 정보를 입력하고, 로봇을 조정하는 실제 조정량을 출력합니다. 우선 로봇의 위치와 도달해야 할 목표점의 위치를 알아낸 뒤에 경로를 찾고 내비게이션을 시작합니다.

지도 정보는 내비게이션 시작지점과 도착 지점 사이의 장애물 정보를 제공하며, 로봇은 경로 계획 알고리즘을 통해 경로를 탐색, 조율을 통해 실제 선속도와 각속도를 출력해 내비게이션을 진행한다.

ros-navigation 내비게이션 시스템은 다양한 ROS 기능 패키지와 알고리즘의 구체적인 실현 노드가 포함되어 있는 패키지입니다. 아래 이미지와 같이, 필수 노드, 로봇 플랫폼과 같은 선택 노드로 분류할 수 있습니다. 여기서, 흰색 부분은 ROS에서 필수적으로 사용해야 하는 모듈이며, 회색 부분은 ROS에서 선택적으로 사용할 수 있는 모듈입니다. 파란색 부분은 직접 구현해야 하는 부분입니다.



move_base는 필수 노드이고, amcl과 map_server는 선택 노드이며 sensor transforms, odometry sourc, sensor sources, basecontroller는 로봇 플랫폼 관련 드라이버입니다. 그 중 가장 핵심인 move_base는 플러그인 메커니즘을 통해 코드를 구성하는데, 이를 통해 global_planner, local_planner, global_costmap, local_costmap, recovery_behaviors 등의 알고리즘을 쉽게 수정할 수 있게 됩니다.

map_server는 지도를 제공하고, amcl은 위치 정보를 제공하며, 센서는 장애물 정보 피드백을, 주행거리 기록계는 운동 정보 피드백을 제공합니다. 섀시 제어 노드는 운동을 담당하게 됩니다. 아래 이미지를 통해 위에서의 설명을 한눈에 확인할 수 있습니다.

move_base: 핵심 노드로서, 다른 노드들의 토픽 메시지를 수신하여 이동 조정 명령으로 변환시켜 전송합니다.

sensor sources: 센서 데이터 예제코드이며 라이다 센서 또는 Depth 카메라 센서로부터 데이터를 수신하고 주변 환경을 실시간으로 팀지하여 장애물 회피 및 내비게이션 기능을 수행합니다.

sensor transforms: 센서 좌표 변환으로, 센서의 좌표계를 다른 좌표계와의 위치 환산을 통해 센서 데이터가 로봇 본체와 장애물 사이의 실제 거리를 표시.

odometry source: 로봇 주행거리 기록계 정보로서, 로봇 지도 내에서의 운동 정보를 기록하며, 로봇이 지도 내 위치 측정 데이터의 주요 소스

amcl: 몬테카를로 위치 측정. 라이다 데이터를 통해 로봇의 현재 자세를 보조 측정. 해당 노드는 회색(필수가 아님을 나타냄)으로 표시되어 있지만 실제로 사용하면 효과가 좋으므로 일반적으로 많이 사용하는 노드

map server: 지도 서비스로서, 이미 구축된 지도를 사용하는지 여부를 결정한다. 매핑 섹션에서 생성한 지도를 사용하려면 map server 기능을 통해 도입 필요

base controller: move_base에서 출력하는 유일한 토픽. move_base는 메시지를 받으면 메시지에 따라 로봇을 제어하여 목표 지점으로 이동

"move_base_simple/goal": 지도에 표시되는 목표 지점으로, rviz 도구를 사용하거나 토픽을 통해 직접 표시가능

위 내용은 내비게이션 프레임의 각 구성 요소에 대한 설명입니다. 지금부터 위치 측정, Cost-Map, 경로 계획 및 복구 계획을 중심으로 ros-navigation 시스템 프레임워크를 분석합니다.

12.3.2. 기능 패키지 설명

내비게이션 시스템은 내비게이션 목표, 위치 정보 및 지도 정보를 입력하고, 로봇을 조종하는 신호를 출력합니다. 먼저 로봇 위치와 도달해야 할 목표 지점을 알아야 합니다.

ros-navigation 기능 패키지의 설치 방법은 두 가지가 있습니다. 첫 번째는 apt-get을 통해 편집된 ros-navigation 폴더를 시스템에 설치하는 것입니다.

cilab@cilab:~$ sudo apt-get install ros-*-navigation

두 번째는 ros-navigation 소스 코드를 다운로드하여 수동으로 편집하고 설치하는 것이다. 내비게이션 기능 패키지를 학습용으로만 사용한다면 1번째 방법으로 설치를 하는 것이 편리할 것입니다. 그러나 알고리즘 개선을 위해 ros-navigation의 코드를 수정해야 한다면 소스 코드를 수동으로 설치해야 할 것이다.

내비게이션 기능 위키 주소는 다음과 같습니다.

링크: http://wiki.ros.org/navigation

12.3.3. AMCL 위치 추정

위치 추정은 로봇이 전체 지도에서의 위치를 계산하는 것을 의미합니다.

SLAM에는 위치 측정 알고리즘도 포함되어 있지만, SLAM의 위치 측정은 전체 지도를 구축하기 위한 것으로, 내비게이션 시작 전 단계에 해당합니다.

현재 위치 측정은 내비게이션 과정에 사용되는데, 로봇은 설정된 경로를 따라 이동하고 위치 측정을 통해 로봇의 실제 경로와 예상이 일치하는지 확인합니다. ROS 내비게이션 기능 패키지 ros-navigation에는 내비게이션 로봇 위치 측정을 위한 AMCL(Adaptive Monte Carlo Localization) 파일이 제공됩니다.

위치 측정은 로봇과 장애물 간의 관계에 대한 이슈와 연관이 있습니다. 경로 계획은 본질적으로 로봇 주변의 장애물을 기반으로 결정을 내리는 과정입니다. 이론적으로는 로봇의 전반적인 위치를 측정하고, 라이다 등의 센서를 통해 스캔 정보를 이용하면 실시간으로 장애물을 회피하고 내비게이션 임무를 완수할 수 있습니다.

그러나 전반적인 위치 측정은 일반적으로 시간적으로도 정확도 방면으로도 효과가 높지 않기에, 이를 경로 추적의 피드백 신호로 사용하는 것은 효과적이지 못합니다. 대신, 인접한 두 위치 사이의 세밀한 위치는 휠 주행거리 기록계나 IMU 등의 실 위치 측정을 통해 확인할 수 있으므로 경로 추적에 있어서 시간적으로나 정확도 면에서 신뢰도를 보장할 수 있습니다.

내비게이션 패키지의 amcl 노드는 map→odom의 tf 관계를 표시하여 전반적인 위치 측정 서비스를 제공합니다. amcl 위치 측정은 필수적인 것은 아니며, 유저는 map→odom tf 관계를 제공할 수 있는 다른 위치 측정 방법으로 교체할 수 있습니다. SLAM, UWB, QR 코드 위치 추정 등을 예로 들 수 있는데 내비게이션 기능 패키지는 로봇 플랫폼의 주행거리 노드에서 게시된 odom→base_footprint의 tf 관계를 통해 실제이동 위치 측정 서비스를 제공합니다. 구체적으로 사용되는 주행거리 기록계 데이터와 실제 로봇 플랫폼에 따라 구체적인 방법이 결정됩니다.

전체 위치 측정과 실 위치 측정은 이미 map→odom→base_footprint와 같은 동적 tf 관계를 구축해두었습니다. 로봇 내 각 센서 사이의 정적 tf 관계는 로봇 URDF 모델을 통해 제공되며 이 tf 관계는 로봇과 장애물 간의 연관 관계를 해결합니다. 예를 들어, 라이다에 전방 3m 위치의 장애물이 탐지된다면, 로봇 플랫폼과 라이다 사이의 tf 관계, 즉 base_link에서 laser_link로의 좌표 변환을 통해 해당 장애물과 로봇 플랫폼 간의 관계를 파악할 수 있습니다.

12.3.4. 셀프 몬테카를로 위치 측정

AMCL은 몬테카를로 위치 측정 알고리즘의 업그레이드 버전으로, 실시간 성능을 향상시키기 위해 몬테카를로 위치 측정 알고리즘의 소량 샘플만을 사용합니다. 이는 셀프 적응 또는 KLD 샘플링 몬테카를로 위치 추정 방법을 실현한 것으로, 기존 지도에 따라 입자 필터를 사용하여 로봇의 자세를 추정합니다.

셀프 몬테카를로 위치 측정 노드는 대부분 라이다 스캔 및 라이다 지도를 사용하여 메시지를 전달하고 자세값을 계산합니다. 과정은 ROS 시스템에서 제공하는 각 초기화 파라미터를 대상으로 하여 셀프 몬테카를로 위치 측정 알고리즘의 입자 필터를 초기화합니다. 최초 세팅 자세값이 설정되지 않은 경우, 셀프 몬테카를로 위치 측정 알고리즘은 로봇이 좌표계 원점에서 시작하는 것으로 가정하고 동작하게 되는데 이렇게 될 경우 계산이 상대적으로 복잡해지게 됩니다. 따라서, 최초 세팅값을 설정하기 위해 rviz에서 2D Pose Estimate 버튼을 통해 자세를 설정해 주는 것을 권장합니다.

로봇 위치 측정의 세 가지 주요 이슈는 전체 위치 측정, 자세 측정 및 납치 이슈입니다. "납치"는 로봇이 자신의 위치를 알고 있지만 잘못된 위치 정보를 받거나 외부의 제 3의 외압에 의해 다른 위치로 물리적으로 이동시키는 경우를 의미합니다. 이때 주행거리 기록계에서는 잘못된 정보를 제공하게 됩니다.

12.4. Cost-Map

내비게이션의 주요 업무는 장애물 회피이므로 장애물에 대한 인식이 핵심 이슈입니다. SLAM 그리드 지도는 장애물 측정 능력을 어느 정도 향상시킬 수 있지만, 환경 내 정적인 장애물 측정만 제공할 수 있습니다. 로봇 내비게이션이 장애물을 회피할 때 SLAM 지도에서 제공하는 정적 장애물 정보만이 아니라 센서로 탐지된 실시간 이동하는 동적 장애물 정보와 특수 장애물 정보도 고려해야 합니다. 예를 들자면 장애물 팽창, 인위적으로 지정된 위험 지역, 보행자 또는 일부 동

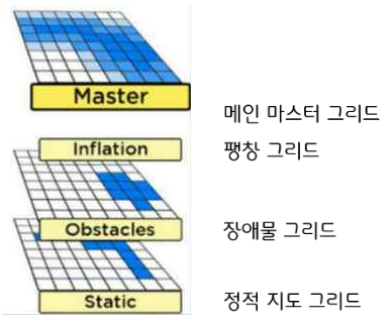
적 정보 등이 있습니다.

여러 가지 복잡한 장애물 인식 이슈를 해결하기 위해 내비게이션 패키지은 Costmap를 이용해 장애물을 통합시켜 해당 과정을 진행합니다. Costmap은 여러 독립 그리드 레이어를 통해 장애물 정보를 처리할 수 있습니다. 각 레이어는 특정 소스의 장애물 정보를 독립적으로 처리하는 것이 가능하며, 이러한 레이어는 필요에 따라 중첩되어 특정 장애물 설명 레이어를 형성할 수 있게 되는 것입니다.

Costmap은 자연스럽게 장애물을 측정할 수 있으며, 필요에 따라 특정 레이어를 생성하고 해당 레이어에서 주요 장애물 정보를 처리할 수 있게 됩니다. 로봇에 라이다만 설치된 경우, 라이다로 탐지된 장애물 정보를 처리하기 위해 장애물(Obstacles) 레이어를 생성해 주어야 합니다. 만일 로봇에 초음파 센서가 추가되었다면, 초음파 센서로 스캔된 장애물 정보를 유지하기 위해 새로운 초음파(Sonar) 레이어를 생성해야 하는 것입니다. 각 레이어는 자체적으로 장애물에 대한 업데이트 규칙을 지니며, 장애물 추가, 삭제, 위치 신뢰도 등 기능이 가능하기에 내비게이션 시스템의 확장성을 크게 향상시키게 되었습니다.

위 레이어들은 costmap_2d에서 플러그인 형식으로 구성되어 있습니다. costmap_2d는 기본적으로 InflationLayer, ObstacleLayer, StaticLayer 및 VoxelLayer를 지원합니다. 유저는 제3의 costmap_2d 레이어 플러그인을 사용하거나 costmap_2d 레이어 플러그인의 인터페이스 규격에 따라 본인이 필요한 레이어 플러그를 별도로 직접 개발할 수도 있습니다.

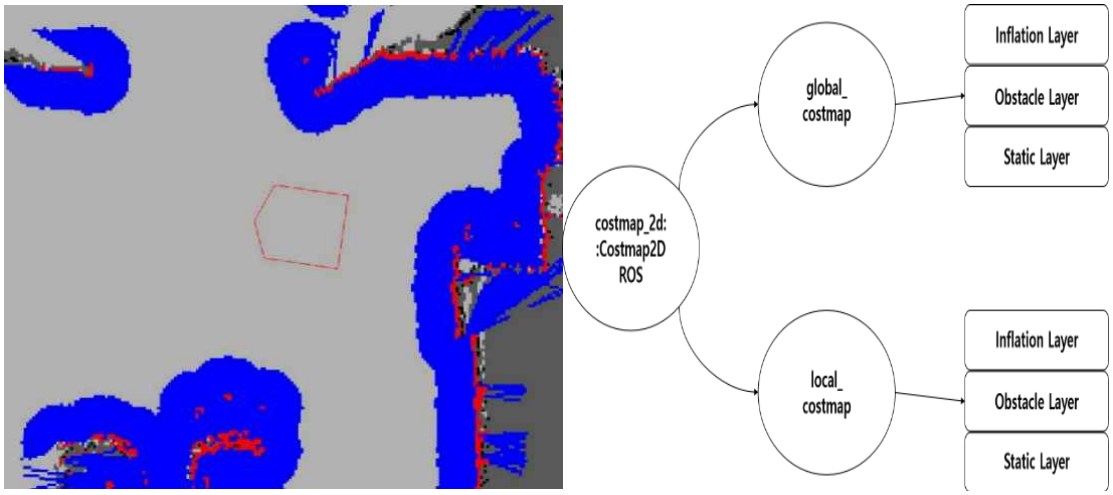
일반적으로 로봇 내비게이션 지도는 아래와 같이 정적 레이어, 장애물 레이어, 팽창 레이어로 구성됩니다.



정적 지도 레이어(Static Layer)는 /map 토픽 정보를 받아 지도 파일을 로딩합니다.

장애물 레이어(Obstacle Layer)는 라이다 등의 정보를 받아 환경 장애물을 실시간으로 탐지하고 관측 예제코드는 라이다, Depth 카메라 또는 초음파 센서 등의 데이터를 수집하며, 토픽은 LaserScan, PointCloud, PointCloud2 등이 포함됩니다.

팽창 레이어(Inflation Layer). 팽창 반경 정보에 따라 장애물을 팽창시켜 로봇의 안전한 운행을 확보할 수 있게 됩니다. 아래 이미지에서는 팽창 후의 효과를 보여줍니다.

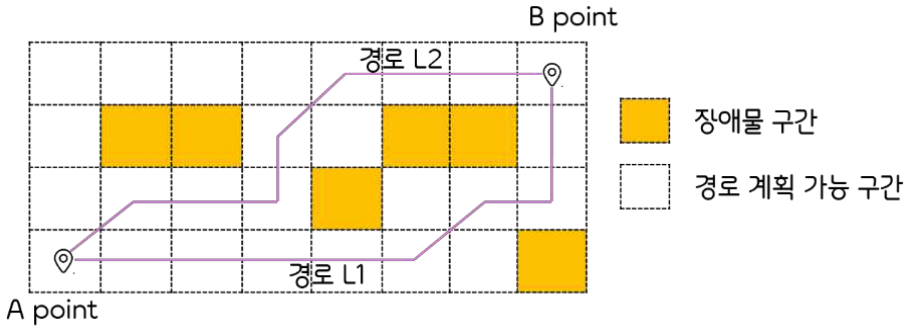


move_base 프레임에는 두 개의 costmap 지도가 생성되게 되는데 global_costmap과 local_costmap이 바로 그 결과입니다.

12.5. 전체 경로 계획

경로 계획과 관련하여 로봇은 “어던 루트로 이동해야 하나요?”라는 세 번째 이슈에 답해야합니다. 이미 지도가 완성된 상황, 미지의 환경에서 지도를 탐색해야 하는 상황 모두 사실상 지도에서 출발점부터 목표점까지 이동할 수 있는 경로를 찾는 것으로 이해하게 됩니다.

로봇의 자율 내비게이션에서 일반적으로 주어진 그리드 지도에서 경로 계획을 수행합니다. 아래 이미지처럼 전체 그리드 지도를 탐색하여 A에서 B로 이동하는 경로를 찾게됩니다. 실제 로봇 내비게이션 과정은 단순히 통행 가능한 경로 하나만 찾으면 되는 것이 아닙니다. 경로의 거리, 원활한 주행, 충돌 위험성, 다양한 제약 조건 및 로봇 자체의 성능도 고려해야 합니다.



로봇 자율 내비게이션에 흔히 사용되는 경로 계획 알고리즘은 Dijkstra, A*, D*, PRM, RRT, 유전 알고리즘, 군집 알고리즘, 퍼지 알고리즘 등이 있습니다. 로봇에서 사용되는 Dijkstra, A*는 그래프 기반의 경로 탐색 알고리즘에 속합니다.

내비게이션 기능패키지에는 navfn, global_planner 및 carrot_planner 전체 공간 경로 계획 플러그가 통합되어 있습니다. 유저는 하나를 선택하여 move_base에 로딩할 수 있습니다. 그 중에서 SBPL_Lattice_Planner, srl_global_planner를 예로 들 수 있는데 번외로 nav_core 인터페이스 규범에 따라 필요한 전체 공간 경로 계획 플러그인을 직접 개발할 수도 있습니다.

navfn은 내비게이션 기능 패키지에 초기에 통합된 전체 공간 경로 계획 플러그인으로, Dijkstra 알고리즘을 기반으로 합니다. global_planner 전체 공간 경로 계획 플러그인은 navfn의 개선된 버전으로 추가적으로 A* 알고리즘도 지원하고 있습니다.

carrot_planner 전체 공간 경로 계획 플러그인은 더 유연한 플래너로, 목표 지점에 대한 처리를 반영합니다. 목표 지점이 장애물 구간과 겹칠 경우, 플래너는 계획이 실패하는 것을 피하기 위해 목표점 주변의 빈 공간을 목표 지점으로 수정하여 계획을 수행합니다.

12.5.1. Dijkstra 알고리즘

Dijkstra 알고리즘은 최단 경로 알고리즘이며, 시작 지점을 중심으로 외부를 향해 확장하며 진행합니다. 너비를 우선적으로 탐색(BFS)하는 방식을 적용하는 것입니다. 최종 지점까지의 데이터를 확보하기 전까지 계속 탐색하며, 가중치를 고려한 너비 우선 알고리즘으로, 전체 경로 계획 문제를 처리하는 데 가장 일반적으로 사용되는 알고리즘 중 하나입니다.

Dijkstra 알고리즘에 대한 자세한 내용과 사용 방법은 아래 링크를 참조해주시기 바랍니다.

링크: https://wiki.ros.org/navfn

12.5.2. A* 알고리즘

A* 알고리즘은 A Star라고 읽기도 하며, 경로 탐색 및 그래프 탐색에서 최단 경로를 찾기 위한 일반적인 방법 중 하나입니다. Dijkstra 알고리즘과 최적 우선 탐색(BFS) 알고리즘의 장점을 종합하여 알고리즘 효율성을 향상시키는 동시에 예상 비용이 가장 낮은 경로를 따라 탐색할 수 있습니다.

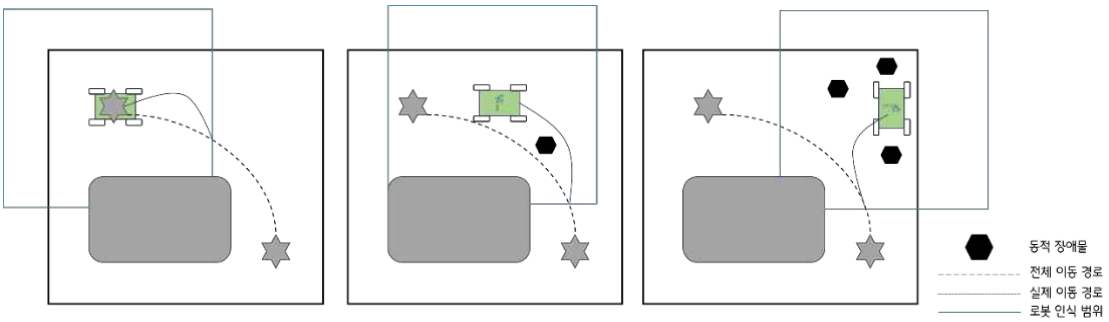
A* 알고리즘에 대한 자세한 내용과 사용 방법은 아래 링크를 참조해주시기 바랍니다.

링크: https://wiki.ros.org/global_planner

12.6. 실제 경로 계획

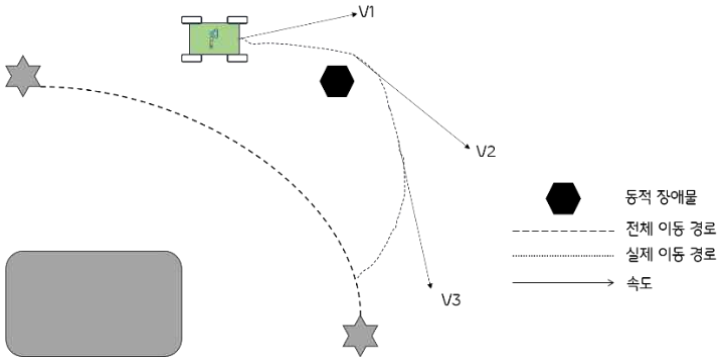
전체 경로 계획은 시작 지점부터 목표 지점을 입력하고 생성된 전역 지도에 표시된 장애물 정보를 기반으로 유효한 경로를 계획합니다. 전체 경로는 산화된 포인트들로 구성되어 있으며, 정적 장애물 정보만 고려하기 때문에 직접적인 내비게이션 제어에 사용되기 난해하고, 참조용도로 사용됩니다.

실제 경로 계획은 전체 경로 계획의 세부 과정이라고 할 수 있습니다. 로봇이 인식 가능한 경계에 있는 경로 포인트를 실제 목표 지점으로 하고, 로봇이 인식한 실제 경로의 동적 장애물 정보를 기반으로 현재 위치에서 실제 목표 지점까지의 경로를 계획합니다. 아래 이미지를 참조해주시기 바랍니다.



실제 경로 계획은 직접적인 내비게이션 제어에 사용되지 않으며, 동적 장애물의 갑작스러운 출현과 같은 예상치 못한 이슈에 노출될 수 있습니다. 실제 로봇의 제어 오차로 인해 실제 이동 경로가 전체 경로에서 벗어날 수도 있습니다.

경로 계획과 측정은 실제 경로 이동 계획의 세부 과정입니다. 이는 실제 이동 경로 계획에 기하학적 제약만 고려하며, 경로 계획은 운동학, 동력학 제약을 추가합니다. 각 경로에서의 포인트 속도 정보를 운동량의 형식으로 입력합니다. 모터 보정 오차, 지면 평탄 여부, 타이어 공회전 등의 문제로 인해 개방 루프 제어를 직접 사용할 수 있습니다. 로봇은 이 경로 추적을 위해 폐쇄 루프 제어를 진행하게 되고 일정 시간 동안 이동하고 참조 경로에서 벗어난 것을 발견하면 운동 방향을 조정하여 수정된 경로로 최대 오차범위 내로 복귀하게 됩니다.



내비게이션 기능 패키지에는 base_local_planner와 dwa_local_planner라는 실제 이동 경로 계획 플러그인이 통합되어 있습니다. 이 중 하나를 선택하여 move_base에 로딩하거나 제3의 실제 경로 경로 계획 플러그인을 사용할 수 있습니다. nav_core 인터페이스 규범에 따라 필요한 실제 경로 계획 플러그인을 개발하여 사용할 수도 있습니다.

base_local_planner는 동적인 궤적 탐색 기반의 실제 이동 경로 계획 프로세스로서, 완전한 구동을 보정하는 데에는 제한이 있을 수 있습니다. 실제 이동 경계 계획 프로세스 dwa_local_planner는 base_local_planner의 개선된 버전으로서, 구동에 있어 완전 또는 비완전 제약 모두 적용할 수 있습니다.

12.6.1. DWA 알고리즘

DWA 알고리즘은 Dynamic Window Approach의 약자로, 동적 윈도우 접근법이라고 불리기도 합니다. 로봇이 장애물을 피하면서 가능한 빠르게 목표지점에 도달하는 것을 목표로 하는 프로세스이며, DWA 알고리즘은 로봇의 속도를 최적화하여 충돌을 피하면서, 최대한 짧은 시간 내에 목표지점에 도달할 수 있도록 하는 계산법이다. 내비게이션 과정에서 DWA 알고리즘은 현재 로봇의 최적 속도를 계산해 명령의 형식으로 로봇으로 전송합니다.

DWA이론의 장애물 회피 원리는 다음과 같습니다. 먼저, 로봇이 실시간 속도로 샘플링을 진행합니다. 각 샘플 속도에 따라 로봇 이동 시뮬레이션을 진행하고 각 샘플 속도에 대해 궤적을 탐색해 시뮬레이션 궤적을 얻게 됩니다. 마지막으로, 장애물과의 거리, 충돌 발생 여부, 목표지점과의 접근 정도, 전체 경로와의 일치성, 속도 등을 기준으로 각 시뮬레이션 궤적을 계산하게 됩니다. 그리고 가장 하이 스코어로 책정된 가상 루트를 실제 이동 계획 경로로 선택합니다.

DWA 알고리즘에 대한 자세한 설명과 사용 방법은 아래 링크를 참조하시기 바랍니다.

링크: http://wiki.ros.org/dwa_local_planner

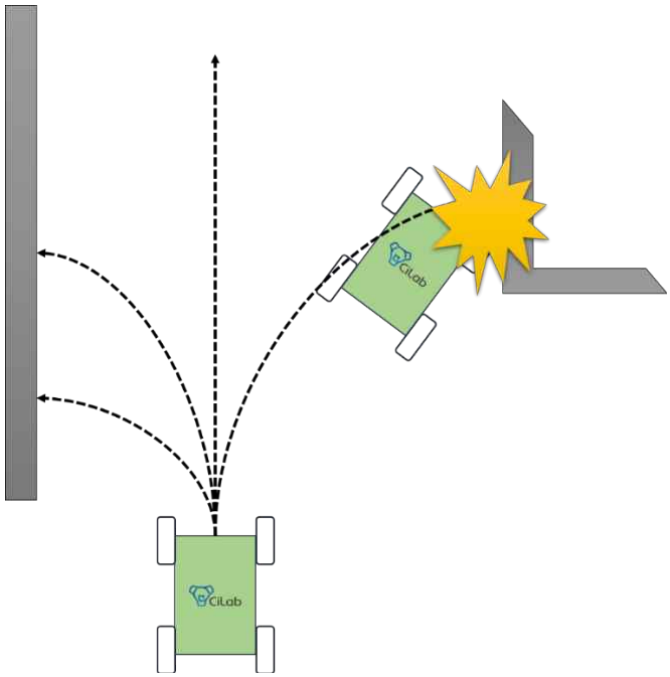
12.7. Recovery 전략

실제 내비게이션 탐색 과정은 손쉽게 곤경에 빠질 수 있습니다. 예를 들어, 장애물에 충돌할 경우, 장애물이 이미 로봇의 기계 모델 내부 즉 동일 그리드 내에 존재하게 되므로 해당 경로 계획은 실패로 돌아갈 수밖에 없습니다. 또한 로봇이 협소한 통로를 극복해야 하는 경우, 센서 블라인드(死角) 영역 및 측정 정확도 등으로 인해 로봇의 후진이 불가한 상황이 있을 수도 있습니다. 로봇이 곤경에 빠지는 요인은 그 외에도 다양합니다. 예를 들어 타이어 공회전, 센서 블라인드 영역, 센서 정확도 등이 있습니다.

따라서 복구 전략은 위와 같은 이슈에서 벗어나는 해결책이라고 할 수 있습니다. 예를 들어, 제자리 회전을 실행(CiLab_X3_MEC 가능)하여 지도 상 장애물을 제거하거나, 매우 느린 속도로 전진이나 후진을 반복 시도하여 장애물의 범위에서 빠져나가는 방법이 있을 수 있습니다.

내비게이션 기능 패키지에는 전체 공간 지도와 실제 이동 Cost-Map의 복구 프로세스가 포함되어 있으며, rotate_recovery, move_slow_and_clear, clear_costmap_recovery 복구 전략 플러그인이 통합되어 있습니다. 유저는 제3의 복구 전략 플러그인을 선택하거나 인터페이스에 따라 직접 개발하여 사용할 수도 있습니다.

내비게이션 작업 중에 유저가 지정한 허용 오차 범위 내에서 목표 자세를 완성할 수 있습니다. 동적인 장애물이 없는 경우, 내비게이션 드라이버는 로봇을 오차 허용 범위 내에서 목표 지점에 도달할 수 있게 진행합니다. 만일 그럴 수 없을 경우 유저에게 오류 신호를 송출하게 됩니다. 로봇이 물리적 이슈에 처하게 된 것이 감지되면, 내비게이션 드라이버에서는 자동으로 복구 전략을 실행하게 되는 것입니다.



recovery_behaviors 파라미터를 사용하여 리커버리 동작을 구성할 수 있으며, 복구 플러그인에 대한 자세한 설명은 아래 링크를 참조해주시기 바랍니다.

링크:

http://wiki.ros.org/rotate_recovery

http://wiki.ros.org/move_slow_and_clear

http://wiki.ros.org/clear_costmap_recovery

12.8. 내비게이션 기능 고급

내비게이션 시스템은 기본 로봇 자율 내비게이션 인터페이스를 제공하며 1번 포인트에서 2번 포인트까지의 단일 지점 내비게이션이 실현될 수 있습니다. 실제로 로봇은 종종 복잡한 작업을 수행해야 하는 상황에 맞딱드릴 수 있습니다. 이러한 작업은 일반적인 기본 작업으로 기계의 형식으로 결합된 형태라고 볼 수 있습니다. 본 챕터에서는 로봇의 멀티 구간 순환 탐색 기능에 대해 설명합니다.

12.8.1. 다중 구간 연속 내비게이션

본 챕터에서는 다중 목표 지점을 설정하고 로봇이 설정된 목표 지점에 차례로 이동하는 기능을 예로 들어 설명합니다. 음식 배달이나 서비스 로봇 같은 응용과 비슷하게 다중의 특정 목표 지점으로 이동을 하는 것입니다.

12.8.1.1. 조작 가이드

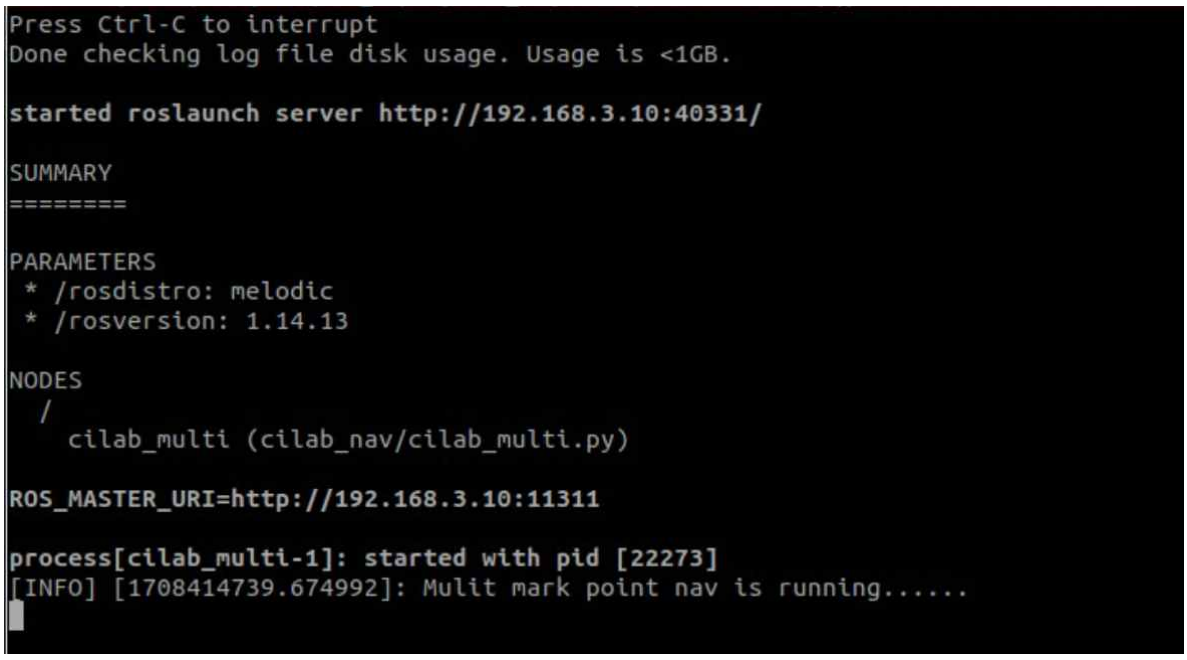
로봇에 ssh 명령어를 이용하여 직접 연결한 후 로봇 cilab_nav 내비게이션 구동 패키지의 내비게이션 드라이버를 실행합니다.

cilab@cilab:~$ roslaunch cilab_nav nav.launch

새로운 터미널(ctrl+shift+t)을 열고 로봇에서 multidrop 목표 지점 구동 프로세스를 실행합니다.

cilab@cilab:~$ roslaunch cilab_nav multi.launch

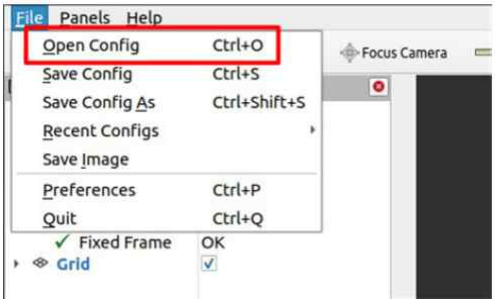
아래 화면이 나타나면 프로그램이 정상 실행되었음을 알 수 있습니다.



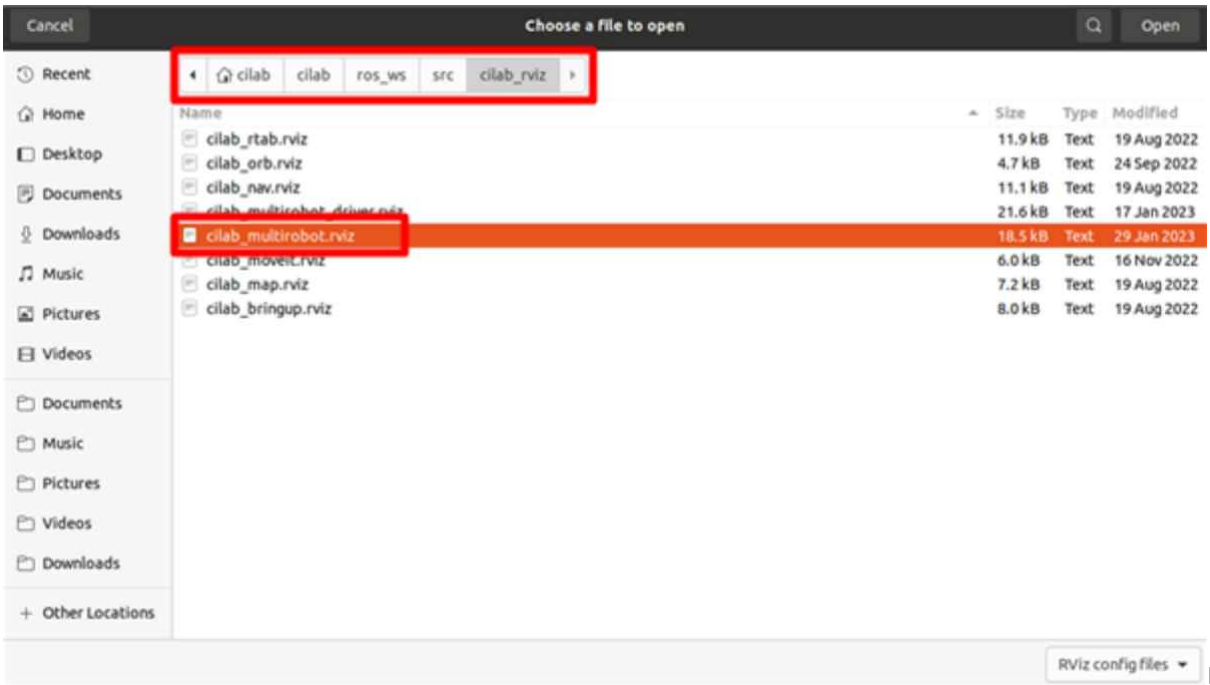
가상 프로그램에서 rviz 도구를 실행합니다.

cilab@cilab:~$ rosrun rviz rviz

rviz 도구에서 가상 프로그램의 구성 파일을 추가하여 rviz 왼쪽 상단 file/Open Config를 클릭합니다.

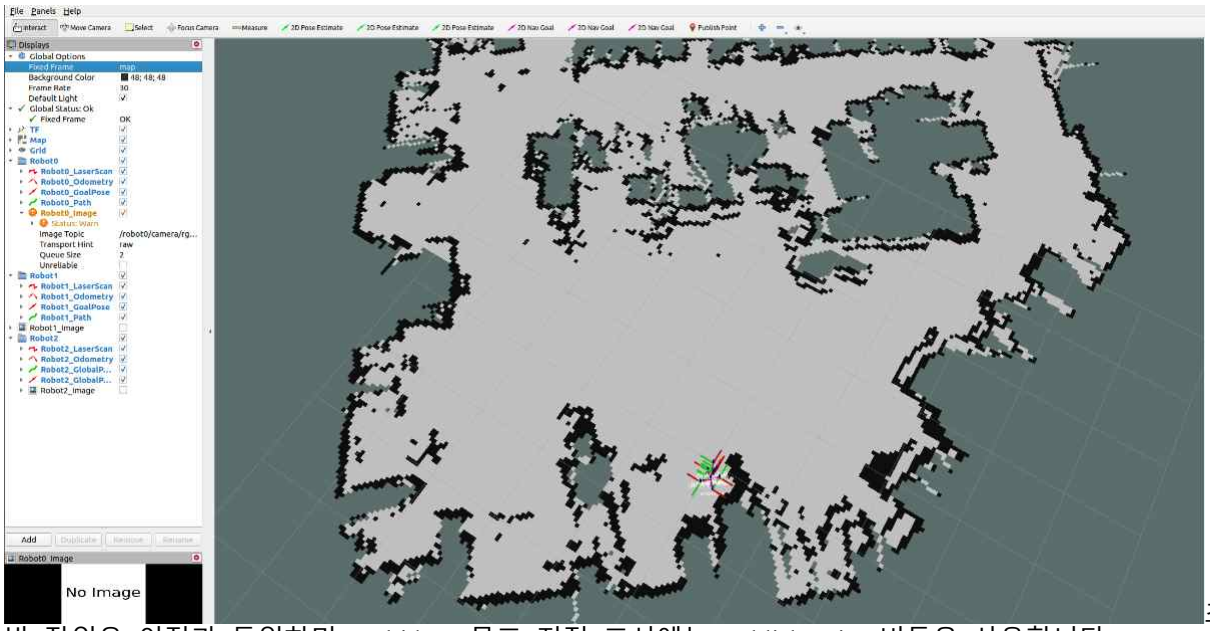


아래 이미지의 경로로 들어가 rviz 파일을 실행합니다.



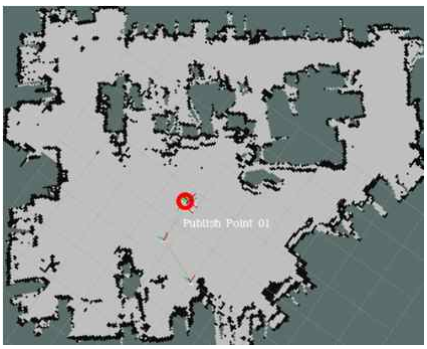
더블 클릭하면 다음과 같은 화면이 나타나게 됩니다.

Close without Saving을 선택하면 매핑 화면이 나타나게 됩니다.



초반 작업은 이전과 동일하며, multidrop 목표 지점 표시에는 Publish Point 버튼을 사용합니다.

Publish Point를 클릭하고 지도에서 임의의 지점을 클릭하면 로봇이 해당 목표 지점으로 이동하게 됩니다. 두 개의 목표점이 동시에 표시된 경우, 먼저 표시된 목표 지점으로 이동하게 됩니다. 효과는 다음과 같습니다.




```
[ INFO] [1711949754.094383309]: Recovery behavio
[ INFO] [1711949754.248163840]: odom received!
[ INFO] [1711949912.521072880]: GOAL Reached!
```

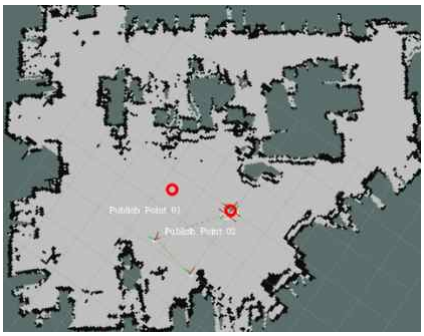
```
process[cilab_multi-1]: started with pid [25719]
[INFO] [1711949760.909445]: Mulit mark point nav is running......
Add a new target  1.
x:1.5448205471, y:1.22861373425, z:0, w:1
Reach the last target point 1, Please input new target point......
```

<단일 지점>

```
[ INFO] [1711949754.094383309]: Recovery behavior will clear layer 'obstacles'
[ INFO] [1711949754.248163840]: odom received!
[ INFO] [1711949912.521072880]: GOAL Reached!
chksum cf errorunknown 169 bytes : ca 18 68 bf 18 67 b7 18 66 ab 18 67 9a 18 50
00
3240 drop 18 fans
[ INFO] [1711950276.619384399]: GOAL Reached!
Add a new target  1.
x:1.5448205471, y:1.22861373425, z:0, w:1
Reach the last target point 1, Please input new target point......

Add a new target  2.
x:1.81036293507, y:0.0602852106094, z:0, w:1
Reach the target point 1, going to next point 2
x:1.81036293507, y:0.0602852106094, z:0, w:1
Reach the last target point 2, Please input new target point......
```

<복수 지점>



12.8.1.2. 기능 패키지 설명

Launch 파일

SLAM 매핑 launch 파일 경로는 다음과 같습니다.

~/cilab/ros_ws/src/cilab_nav

roscd 명령을 사용하여 해당 기능 패키지를 빠르게 이동할 수 있습니다.

cilab@cilab:~$ roscd cilab_nav/launch/

여기서 multi.launch는 multidrop 내비게이션 패키지입니다.

내용은 다음과 같습니다.

Open | multi.launch ~/cilab/ros_ws/src/cilab_nav/launch | Save

```
<launch>

    <!-- Multi-navigation, through the RVIZ gui send multi-navigate points, robot can follow that -->
    <node pkg="cilab_nav" type="cilab_multi.py" name="cilab_multi" respawn="false" output="screen">
    </node>

</launch>
```

Plain Text | Tab Width: 8 | Ln 7, Col 10 | INS

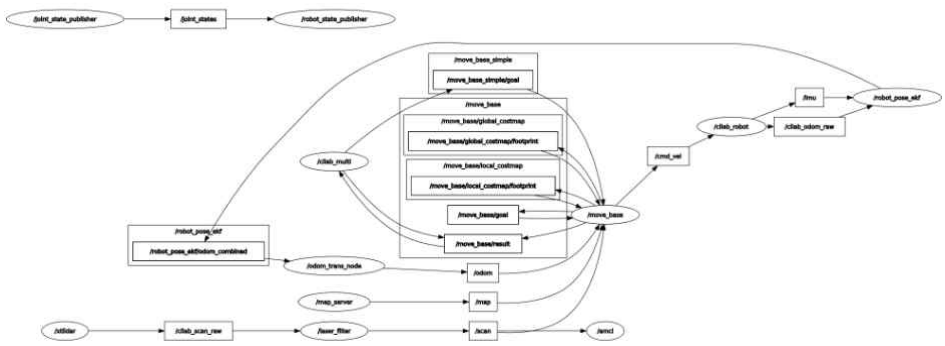
이 launch 파일의 주요 역할은 cilab_multi.py을 실행하여 multidrop 내비게이션을 시작하는 것입니다.

노드 토픽 설명

로봇에서 cilab_multi.launch에서 매핑 launch 파일을 실행한 후 rqt_graph를 사용해 현재의 토픽 메시지 연결 상태를 확인할 수 있습니다.

cilab@cilab:~$ rosrun rqt_graph rqt_graph

아래 이미지와 같이, /move_base에서 cilab_multi가 표시하는 /move_base_simple/goal 토픽을 통해서 /cilab_multi와 /move_base 내비게이션 프레임을 연결합니다.



12.8.2. 다중 지점 순환 내비게이션

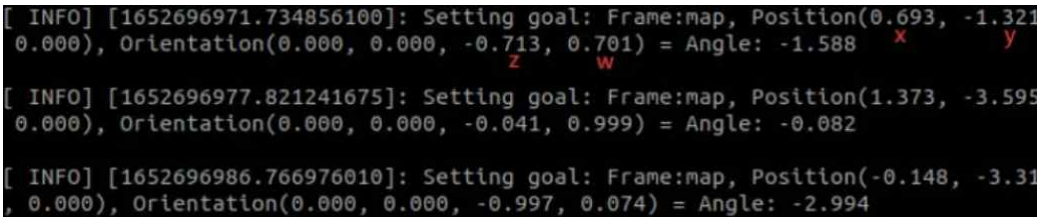
본 챕터에서 내비게이션 패키지 내 제공하는 단일 포인트 내비게이션 인터페이스를 기반으로 간단한 응용 프로그램을 실현하여 다중 포인트 순환 내비게이션을 실현하게 됩니다. 실제 소독용 로봇과 유사한 방식이며, 다중(multi) 목표 지점을 순환하여 이동하는 것이 특징입니다.

12.8.2.1. 조작 가이드

ssh 명령어를 이용하여 로봇에 직접 연결하고 로봇에 cilab_nav 내비게이션 패키지의 내비게이션 노드를 실행합니다.

cilab@cilab:~$ roslaunch cilab_nav nav.launch

rviz를 열고 지도에서 이동하고자 하는 다중 목표 지점을 선정합니다. 일반 내비게이션 2D Nav Goal을 통해 목표 지점을 표시할 때 rviz에서 목표 지점 정보가 표시되게 됩니다.



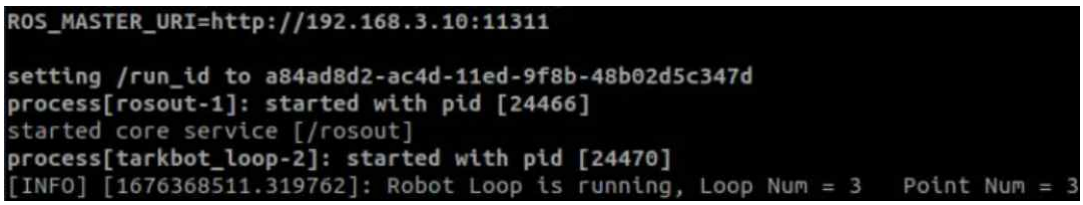
아래 지도를 보게 되면 세 개의 목표 지점을 표시해두었는데, 이 포인트들을 순환 내비게이션의 launch 파일로 수정해 줍니다.

파일 경로: ~/cilab/ros_ws/src/cilab_nav/launch, 파일명: loop.launch

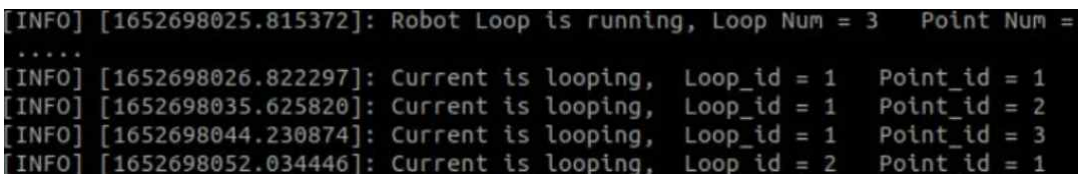
새로운 터미널을 실행하고 ssh명령어를 통해 로봇에서 multidrop 구동 파일을 실행해 줍니다.

cilab@cilab:~$ roslaunch cilab_nav loop.launch

아래 프롬프트가 실행되면 정상적인 가동이 완료되었음을 확인ㅊㅇ 할 수 있습니다.



화면에 내비게이션 정보를 순환으로 표시합니다.



로봇이 움직이기 시작한 후 목표 지점을 추가해야 할 때는 2D 데이터에 새로운 위치 정보를 직접 추가할 수도 있습니다. 순환 횟수는 loop_num 파라미터를 통해 변경할 수 있는데 예를 들어, 4번 순환하면 파라미터를 4로 변경하면 됩니다.

Open ▾ loop.launch ~/cilab/ros_ws/src/cilab_nav/launch Save

```
<launch>

    <!-- Multi-point navigation, Self dedicate point, for example service robot at restaurant -->
    <node pkg="cilab_nav" type="cilab_loop.py" name="cilab_loop" respawn="false" output="screen">

        <!-- MAP Point  -->
        <param name="map_frame" value="map" />

        <!-- Cycle NUM  -->
        <param name="loop_num"  value="1" />

        <!-- Dedicate cycle point, the robot can setting multi point automaticaly  -->
        <param name="point"  value="1.177,  0.024, -0.005,  1.000,
                                    0.711,  1.160, 0.313,  0.950,
                                    0.054,  0.058, 0.027,  1.000  " />

    </node>

</launch>
```

Plain Text ▾ Tab Width: 8 ▾ Ln 9, Col 2 ▾ INS

12.8.2.2. 기능 패키지 설명

Launch 파일

SLAM 매핑 launch 파일 경로는 다음과 같습니다.

~/cilab/ros_ws/src/cilab_nav

roscd 명령을 통해 해당 기능 패키지를 빠르게 이동할 수 있습니다.

cilab@cilab:~$ roscd cilab_nav/launch/

loop.launch는 순환 내비게이션 파일로 구체적인 내용은 다음과 같습니다.

Open ▾ loop.launch ~/cilab/ros_ws/src/cilab_nav/launch Save

```
<launch>

    <!-- Multi-point navigation, Self dedicate point, for example service robot at restaurant -->
    <node pkg="cilab_nav" type="cilab_loop.py" name="cilab_loop" respawn="false" output="screen">

        <!-- MAP Point  -->
        <param name="map_frame" value="map" />

        <!-- Cycle NUM  -->
        <param name="loop_num"  value="1" />

        <!-- Dedicate cycle point, the robot can setting multi point automaticaly  -->
        <param name="point"  value=" 1.177,  0.024, -0.005,  1.000,
                                     0.711,  1.160, 0.313,  0.950,
                                     0.054,  0.058, 0.027,  1.000  " />

    </node>

</launch>
```

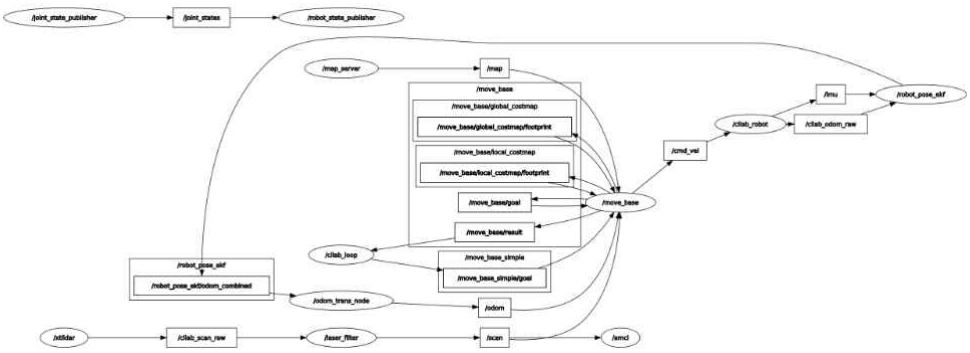
Plain Text ▾ Tab Width: 8 ▾ Ln 9, Col 2 ▾ INS

해당 launch 파일은 cilab_loop.py를 실행하여 multidrop 순환 내비게이션 기능을 구동합니다.

노드 토픽 소개

ssh 명령어로 로봇 연결 후 cilab_loop.launch를 실행하면 rqt_graph를 통해 토픽 메시지를 통해 연결 상태를 확인할 수 있습니다.

cilab@cilab:~$ rosrun rqt_graph rqt_graph



아래 이미지와 같이, /move_base에서 cilab_multi가 게시하는 /move_base_simple/goal 토픽과 /move_base/result을 통해 /cilab_loop와 /move_base 내비게이션 프레임을 연결합니다.

13. 프로그래밍 개발 환경

13.1. 프로그래밍 개발 환경 소개

ROS의 개발은 Ubuntu의 gedit, vi 등 다양한 편집기를 사용해 예제코드의 수정이 가능합니다. 프로젝트 학습 및 개발을 용이하게 하기 위해서 많은 개발자들이 Eclipse, Vim, Qt Creater Pycharm, VS Code등의 IDE를 사용하고 있습니다.

IDE와 관련하여 자세한 내용은 아래 링크를 확인해주세요

http://wiki.ros.org/IDEs

13.1.1. Eclipse

Eclipsesms 만능 통합 개발 환경으로 다양한 프로그래밍 언어의 프로젝트 개발에 적합하며 풍부한 플러그인을 통해 IDE의 기능을 확장할 수 있습니다.

13.1.2. VS Code

VS Code는 강대한 커넥터 저장소와 확장 기능을 제공하기에 사용자가 ROS 학습 및 개발에 있어 매우 유용한 프로그램입니다.

14. ROS 학습법

ROS는 상대적으로 복잡한 구조이기 때문에 다음과 같은 방법으로 학습하시는 것을 권장합니다.

1. Linux의 이해, ROS는 Linux 환경에서 사용하기 때문에 Linux의 구조 및 기초 조작법 등을 학습할 수 있습니다.

2. ROS 로봇 조작 시에는 ssh 명령어를 통해 많은 파일을 작업할 수 있기에 텍스트(.txt)기반의 에디터 사용법을 학습하여 사용하시는 것을 비교적 권장합니다.

3. ROS의 기본 개념을 학습하시는 것을 권장합니다.

4. 루틴 소스 예제 코드 결합, ROS 관련 요소 및 편집 방법을 학습합니다.