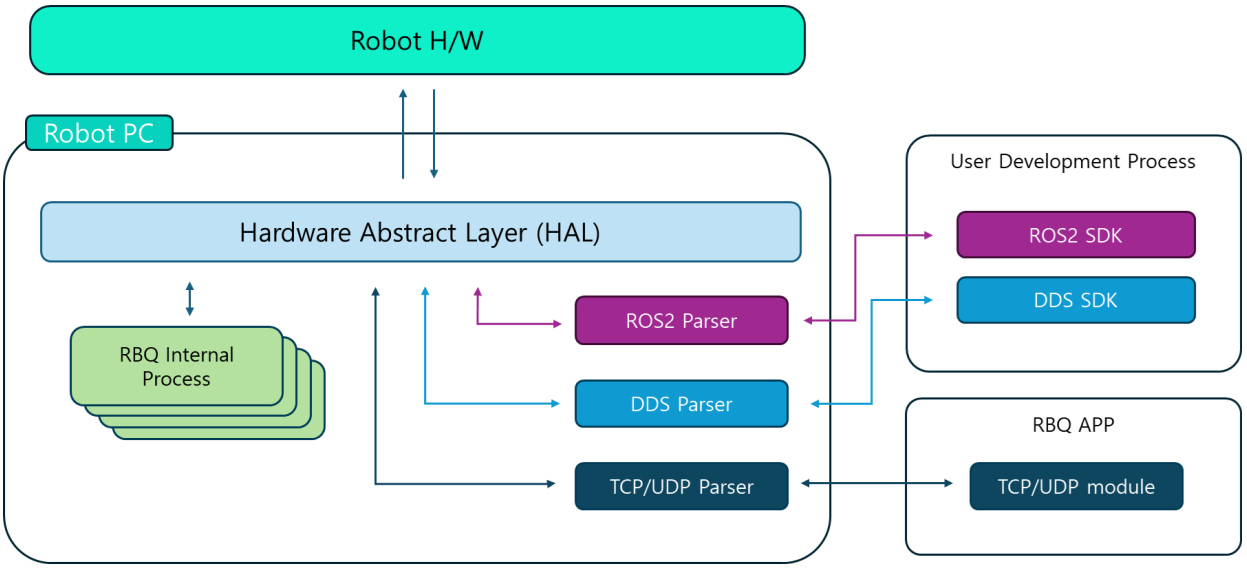
RAINBOW ROBOTICS

RBQ 소프트웨어 가이드

목차

1.1 시스템 다이어그램



SDK

개요

RBQ SDK는 개발자에게 **RBQ 사족보행 로봇**을 위한 유연하고 확장 가능한 인터페이스를 제공하기 위해 설계되었습니다.

Low Level 제어와 **High Level 명령**을 모두 지원하므로, 자체 컨트롤러를 개발하거나 로봇을 기존 시스템에 통합하는 경우 모두 적합한 진입점이 있습니다.

🔧 Low Level 접근

LV0 SDK는 로봇의 센서와 액추에이터에 직접 접근할 수 있습니다.

RAINBOW ROBOTICS

정밀한 제어와 하드웨어로부터의 **실시간 피드백**이 필요한 개발자를 위해 설계되었습니다.

✅ 주요 기능:

- 센서 데이터 읽기: IMU, 조인트 인코더 등
- 조인트 레벨 명령 전송: 위치, 속도, 토크
- 적합한 용도:
  - 커스텀 컨트롤러 개발
  - 저지연 실시간 실험
  - 보행 제어 연구

🎯 High Level 명령 인터페이스

LV1 SDK는 Low Level 세부 사항을 관리할 필요 없이 사전 정의된 동작과 명령을 사용하여 로봇을 제어할 수 있는 **간소화된 인터페이스**를 제공합니다.

✅ 주요 기능:

- 단일 명령으로 일반적인 모션 및 보행 동작 실행
- **RBQ APP** 명령 세트와 원활한 통합
- 사용 가능한 API:
  - C/C++
  - Python
  - ROS2
- 적합한 용도:
  - 애플리케이션 개발자
  - 시스템 통합
  - 최소한의 설정으로 빠른 프로토타이핑

📡 지원되는 통신 인터페이스

SDK는 사용 사례에 따라 여러 통신 프로토콜을 지원합니다:

인터페이스	설명
ROS2 SDK	ROS2 생태계의 로보틱스 개발자용
DDS SDK	DDS를 사용한 실시간 분산 통신용
TCP / UDP	커스텀 도구를 위한 경량 직접 패킷 통신

1.2 개발자 가이드

📚 소개

RBQ 시뮬레이션, 명령 및 배포/업데이트 매뉴얼에 오신 것을 환영합니다

이 매뉴얼은 RBQ 로봇을 **시뮬레이션**하고 명령하며 실제 로봇에 자신만의 알고리즘을 **배포**하는 방법에 대한 종합적인 가이드를 제공합니다.

✅ 🖥️ 개발자를 위한 시뮬레이터 및 GUI

- RBQ 로봇은 **MuJoCo**를 사용하여 정확하게 시뮬레이션할 수 있습니다.

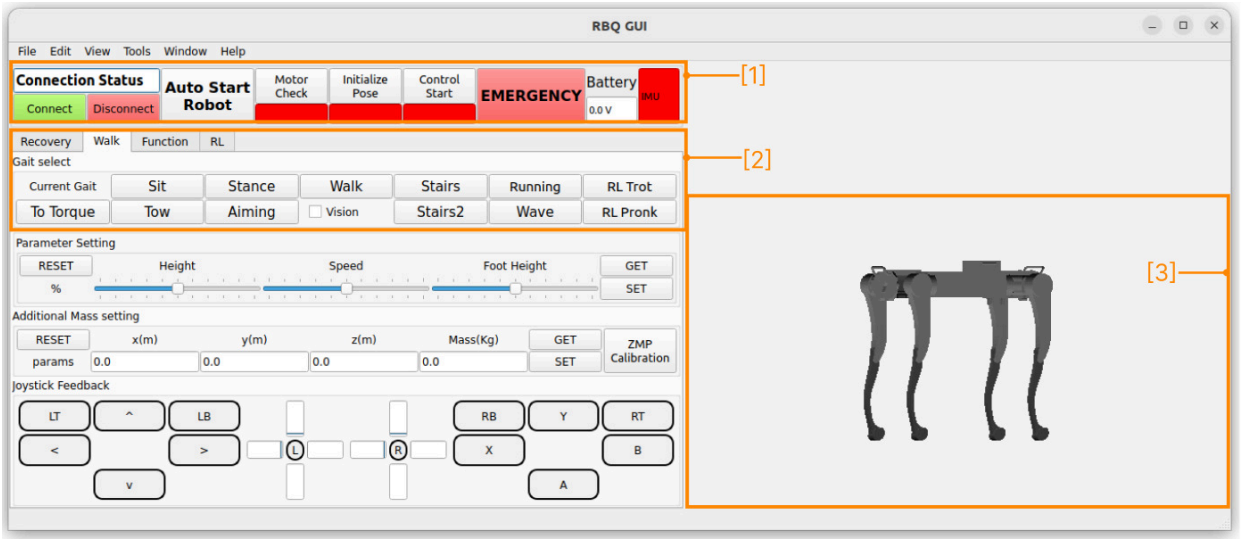
  📝 *개발자는 다음 파일을 편집하여 시뮬레이션 환경 "playground"를 수정할 수 있습니다:* `resources/model/environment.xml`

- **RBQ GUI**는 사용자가 로봇에 명령을 내리고 모니터링할 수 있는 직관적인 인터페이스를 제공하도록 설계되었습니다.
  - 로봇 상태, 센서 상태에 대한 시각적 피드백을 제공하며 키보드와 조이스틱 입력을 모두 지원합니다.

  ▶ YouTube에서 열기 (클릭)

GUI 개요

- 이 섹션에서는 **RBQ GUI** 사용 방법을 설명하며, 주요 기능과 인터페이스 레이아웃을 포함합니다.



- [1] : 상단 패널
- [2] : 중앙 패널
- [3] : 3D 패널

📌 상단 패널

상단 패널은 로봇 연결, 초기 자세, 모터 상태 확인 등 핵심 기능을 제공합니다. 비상 정지 버튼과 배터리 전압 및 IMU 센서 상태 표시기도 포함되어 있습니다.

- **Connection Status and Buttons:** 로봇 연결 여부를 표시합니다. 이 버튼을 사용하여 로봇과 연결하거나 끊을 수 있습니다.
- **Auto Start** 버튼: [Motor Check → Initialize Pose → Control Start] 순서로 로봇을 자동으로 초기화하고 제어 시스템을 준비합니다.
- **Motor Check / Initialize Pose / Control Start** 버튼: 각각의 시작 동작을 개별적으로 수행할 수 있습니다. 하지만 **Auto Start** 버튼을 사용하는 것이 권장됩니다.
  📝 *시뮬레이션: Control Start 버튼만 초록색으로 표시됩니다. 실제 로봇: 준비되면 세 버튼 표시등이 모두 초록색으로 변경됩니다.*
- EMERGENCY 버튼: 로봇의 모든 동작을 즉시 중단하여 로봇이 부드럽게 내려앉도록 합니다.
  - 비상 정지 후 동작을 재개하려면 **Gait select** 그룹 박스에서 **Stance** 버튼을 누르십시오.

> 📝 *시스템은 stance motion(초록색)으로 전환하기 전에 RECOVERY 모드(노란색)로 진입합니다.*

▶ YouTube에서 열기 (클릭)

📌 중앙 패널

- 중앙 패널에는 로봇에 다양한 명령을 보내기 위한 버튼이 있는 탭이 포함되어 있습니다.

  *실제 로봇에서는 추가적으로 **관절**, **펌웨어**, **전원** 탭이 있지만 사용하지 않는 것을 권장합니다.*

- **[보행 탭]** 로봇의 걷기 동작과 보행 방식 간 전환을 명령합니다.

✅ 버튼 동작	Sit	Stance	Walk	Stairs	Running	RL-trot	RL-pronk
⌨️ 키보드 키	1	2	3	4	5	R	F

3D 패널

- 이 패널은 연결된 로봇의 현재 자세를 3D로 렌더링한 디지털 트윈을 표시합니다.

📦 설치

- 이 섹션에서는 처음부터 RBQ 로봇 시뮬레이션 및 개발 환경을 설치하고 설정하는 방법을 설명합니다.

💻 시스템 요구사항

RBQ 시뮬레이션 및 배포 환경을 원활하게 실행하기 위해 다음 요구사항이 권장됩니다:

- 운영 체제: Ubuntu 22.04 (x86)

- 최소 PC 사양:
  - CPU: Intel Core i7 12세대
  - RAM: 16 GB
  - 저장 공간: 25 GB
- 제어 장치
  - 키보드: W, A, S, D, Q, E, R, F, [1-5] 키
  - 조이스틱: Logitech Wireless Gamepad F710 (권장)

📥 1. RBQ 저장소 다운로드

- 다음 명령어를 사용하여 저장소를 복제하세요:

```bash
git clone https://github.com/RainbowRobotics/RBQ.git
```

🛠️ 2. 환경 설정

- 다음 명령어를 사용하여 필요한 라이브러리를 설치하세요:

```bash
bash scripts/ex/install/setup.bash
```

▶️ 시뮬레이션

- 스크립트를 사용하여 **RBQ 제어 모듈, MuJoCo 시뮬레이터 및 GUI**를 함께 시작합니다.

참고: *인자:*

- `--help` : 도움말 메시지를 표시하고 종료합니다.
- `--vision` : 비전 모듈을 실행합니다 (예: 비전 센서, 계단 오르기 활성화).

```bash
bash scripts/sim.bash
```

- 프롬프트가 표시되면 새로 열린 CLI 터미널에서 `사용자 비밀번호`를 입력하여 프로세스를 시작합니다.

🔗 초기화 - 시뮬레이션 로봇 연결 및 시작

- `RBQ GUI`에서 **Connect** 버튼을 클릭하여 연결을 설정합니다.
  - 성공하면 `연결 표시등`이 **Simulator Connected** 텍스트와 **파란색**으로 변경됩니다.
- `RBQ GUI`에서 **Auto Start** 버튼을 클릭하여 제어를 시작합니다.
  - 성공하면 `연결 표시등`이 **초록색**으로 변경됩니다.

📌 Stance - 로봇이 서도록 명령하기

- 경로: `"Walk" tab --> "Gait select" group-box --> "Stance" 버튼` 또는 키보드 버튼 `2`를 누르세요
- 명령이 내려지면 로봇이 서 있는 자세로 이동하기 시작합니다. 완료되면 추가 명령에 응답합니다.
  - 📝 *현재 자세에 따라 서 있는 자세로 전환하는 데 몇 초가 걸릴 수 있습니다.*

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Nose up / down
L Stick	Left / Right	Q / E	Roll Left / Right
R Stick	Forward / Backward		Height up / down
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full range of motion

- 📝 *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Walk - 로봇이 걷도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Walk" 버튼` 또는 키보드 버튼 `3`을 누르세요
- 명령이 내려지면 로봇이 걷기 보행으로 전환됩니다. 완료되면 추가 명령에 응답합니다.

Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		Pitch Forward / Backward
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Stair - 로봇이 계단을 오르도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Stairs" 버튼` 또는 키보드 버튼 `4`를 누르세요
- 로봇은 비전 기능을 활용하여 계단을 올라갑니다.

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		

R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- 📝 *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Run - 로봇이 달리도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Running" 버튼` 또는 키보드 버튼 `5` 를 누르세요
- 로봇은 빠르고 민첩한 동작을 수행합니다.
    - *달리기로 전환하기 전에 현재 자세와 보행에 따라 STANCE 자세로 전환해야 할 수 있습니다.*

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Wave - 로봇이 부드럽게 걷도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Wave" 버튼`
- 로봇은 Wave 보행 패턴으로 부드럽게 걷습니다.

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		Pitch Forward / Backward
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- 📝 *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Aim - 로봇을 목표물에 향하도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Aiming" 버튼`
- 이 모드를 사용하여 로봇을 특정 목표물이나 방향으로 정확하게 조준합니다.

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Nose up / down
L Stick	Left / Right	Q / E	
R Stick	Forward / Backward		
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

RL-Trot - 로봇이 RL-Trot 하도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "RL-Trot" 버튼` 또는 키보드 버튼 `R` 를 누르세요
- 이 모드는 강화 학습 정책으로 생성된 Trot 보행을 활성화합니다. 로봇은 RL 기반 걷기 보행으로 전환되고 이후 사용자 명령에 응답합니다.

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		Pitch Forward / Backward
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 RL-Pronk - 로봇이 RL-Pronk 하도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "RL-Pronk" 버튼` 또는 키보드 버튼 `F` 를 누르세요
- 이 모드는 강화 학습 정책으로 생성된 Pronk 보행을 활성화합니다. 로봇은 RL 기반 Pronk 보행으로 전환되고 이후 사용자 명령에 응답합니다.

🕹️ Joystick	Command	Keyboard key	Robot Action
L Stick	Forward / Backward	W / S	Go Forward / Backward
L Stick	Left / Right	Q / E	Go Left / Right
R Stick	Forward / Backward		Pitch Forward / Backward
R Stick	Left / Right	A / D	Turn Left / Right
		Shift	Full speed of motion

- 📝 *조이스틱이 연결되면 키보드의 숫자 키만 바인딩됩니다.*

📌 Sit - 로봇을 앉히도록 명령하기

- 경로: `"Walk" 탭 --> "Gait select" 그룹 박스 --> "Sit" 버튼` 또는 키보드 버튼 `1` 를 누르세요
- 로봇이 부드럽게 앉은 자세로 전환하도록 명령합니다.
    - *현재 자세에 따라 앉은 자세로 전환하는 데 몇 초가 걸릴 수 있습니다.*

🤖 실제 로봇

이 섹션에서는 **실제 RBQ 로봇**을 작동하는 방법을 설명합니다.

📖 *자세한 작동 단계는 **작동 가이드**를 참조하세요*

1. 로봇에 연결하기

- PC를 로봇의 Wi-Fi에 연결합니다.

    Wi-Fi SSID: `RBQ_xxxx` .

2. GUI 실행 및 로봇 제어 설정

`bin` 디렉토리에서 `GUI`를 시작합니다:

```bash
cd <your workspace>/RBQ
```

```bash
./bin/GUI
```

- **Connect** 버튼을 클릭합니다.
  - 성공하면 `연결 표시등`이 **Robot Connected** 텍스트와 **초록색**으로 변경됩니다.
- **자동 시작** 버튼을 클릭합니다.
  - 성공하면 `제어 표시등`이 **초록색**으로 변경되어 로봇이 성공적으로 활성화되었고 이제 작동 준비가 되었음을 나타냅니다.

3. 명령은 시뮬레이션과 정확히 동일합니다

- 📝 *먼저 시뮬레이터에서 로봇을 조종 하는 연습을 하는 것을 강력히 권장합니다.*

🚀 로봇에 어플리케이션(바이너리 파일) 배포하기

- 개발 PC에서 다음 명령어를 실행하여 로봇에 바이너리를 배포합니다:

참고: `인자:`

- `--help` : 도움말 메시지를 표시하고 종료합니다.
- `--scripts` : 새 스크립트 배포

```bash
bash scripts/deploy.bash
```

SSH 비밀번호를 입력하라는 메시지가 표시됩니다.

- **비밀번호:** 담당자에게 SSH 비밀번호를 문의하세요.

- ⚠️ 중요: `deploy.bash`가 완료되면 로봇을 껐다 켜서 새로 배포된 바이너리로 실행합니다.

1.3 RBQ API 개요

RBQ API는 개발자가 RBQ 사족보행 로봇을 제어하고 상호작용할 수 있는 포괄적인 C++ 인터페이스를 제공합니다. 멀티프로세스 환경에서 저수준 실시간 제어와 고수준 행동 명령을 모두 지원하도록 설계되었습니다.

C++ 문서

C++ 문서 보기

프로세스 ID (모션 소유권)

`RBQ_API` 인스턴스를 생성할 때 **프로세스 ID**를 제공해야 합니다:

```cpp
RBQ_API api(20);  // 사용자 애플리케이션용 프로세스 ID (20-39 범위)
```

이 ID는 **모션 소유권 메커니즘**을 사용하여 안전한 멀티프로세스 제어를 보장하며, 각 관절은 소유자 프로세스에 의해서만 제어될 수 있습니다.

> ⚠️ 관절 명령을 보내기 전에 항상 setMotionOwner()를 호출하세요.

API 구조 요약

카테고리	구성요소	설명
제어	`Joint`	저수준 관절 제어: 위치, 토크, 게인 (Kp/Kd)
	`startMoveJoint()` / `stopMoveJoint()`	관절별 조그 모션 시작/중지
	`lockUnlockJoint()`	수동 조작을 위한 관절 잠금/해제

카테고리	구성요소	설명
센서	`IMU`	자세 (쿼터니언/RPY), 자이로, 가속도 접근
	`getBatteryVoltage()`	배터리 전압 읽기 (내부 전용)
게임패드	`Gamepad`	Logitech F710 게임패드 입력 접근: 조그, 트리거, 버튼
고수준 모션	`motionStaticReady()` / `motionDynamicWalk()` 등	미리 정의된 로봇 행동

관절 제어 API (`RBQ_API::Joint`)

센서 값 읽기

함수	설명
`getPos()`	관절 위치 읽기 (rad)
`getVel()`	관절 속도 읽기 (rad/s)
`getTorque()`	관절 토크 읽기 (Nm)
`getGainKp()` / `getGainKd()`	현재 Kp/Kd 게인 읽기

명령 전송

함수	설명
`setMotionOwner()`	관절 제어권 획득
`setPosRef()`	관절 위치 목표 설정
`setTorqueRef()`	토크 목표 설정 (Nm)
`setGainKp()` / `setGainKd()`	양자화를 통한 제어 게인 설정

> ✅ 각 관절은 제어 명령을 받기 위해 소유자 프로세스가 있어야 합니다.

IMU 센서 API (`RBQ_API::IMU`)

RAINBOW ROBOTICS

로봇의 자세와 운동 상태에 대한 접근을 제공합니다.

함수	출력
`getQuaternion()`	Eigen 쿼터니언으로 자세 (w, x, y, z)
`getRPY()`	롤, 피치, 요 (ZYX 오일러 각도, 라디안)
`getGyro()`	각속도 (rad/s) (X, Y, Z)
`getAcc()`	선형 가속도 (m/s^2) (X, Y, Z)

게임패드 API (`RBQ_API::Gamepad`)

원격 조작을 위한 **Logitech F710 (X 모드)** 지원.

타입	함수	설명
조이스틱	`getLeftJogX/Y()`, `getRightJogX/Y()`	범위: [-1.0, 1.0]
트리거	`getLeftTrigger()`, `getRightTrigger()`	범위: [0.0, 1.0]
버튼	`getButtonState(Button::X)`	눌렸을 때 `true` 반환

내장 모션 프로그램

미리 정의된 로봇 행동을 쉽게 트리거할 수 있습니다:

함수	설명
`motionStaticReady()`	준비 자세로 이동
`motionStaticGround()`	바닥에 눕는 자세
`motionDynamicWalk()`	동적 보행 시작
`motionDynamicRun()`	동적 달리기 시작
`motionDynamicAim()`	조작/비전을 위한 조준 자세
`motionDynamicStairs()`	계단 오르기 모션
`motionDynamicHealthCheck()`	진단 모션

ROS 2™

📘 RBQ ROS2 SDK 소개

RBQ **ROS2 SDK 메뉴얼**에 오신 것을 환영합니다.

- 이 가이드는 ROS2를 통해 RBQ 로봇을 **실제 하드웨어와 시뮬레이션**에서 제어하고 시각화하는 방법을 빠르게 시작할 수 있도록 도와줍니다.

이 SDK가 제공하는 기능

> 이 SDK는 다음과 같은 주요 기능을 제공합니다:

- **로봇 제어**: 자세 제어, 보행 전환, 위치 기반 내비게이션 실행
- **센서 피드백 스트리밍**: IMU, 발 접촉, 배터리 상태 등 지속적인 데이터 제공
- **비전 데이터 스트리밍**: 전방, 후방, 하방 및 좌우 카메라 데이터를 제공하여 인지 및 내비게이션 지원

- **RViz2 시각화**: 로봇 상태 및 LiDAR 등 센서를 실시간으로 시각화

💻 시스템 요구사항

구성 요소	필요 환경
운영체제	Ubuntu 22.04
ROS 2 배포판	ROS2 Humble
Python 버전	python 3.8|3.9|3.10

🛠️ 설치

📝 *아래의 모든 CLI 명령어는 **메인 디렉토리**에서 실행하는 것을 기준으로 합니다.*

- RBQ 저장소를 작업 공간에 클론합니다:

```bash
git clone https://github.com/RainbowRobotics/RBQ.git
```

시스템 설정

> 이 "시스템 설정" 과정은 '**개발자 PC**'에서 실행되어야 합니다. 이 패키지는 `Ubuntu 22.04 및 ROS 2 Humble`에서 테스트되었습니다.

- `ROS Humble` 설치

```bash
cd <your workspace>/RBQ
sudo bash scripts/ex/install/ros.bash
```

- `ROS Humble` 환경 불러오기

```bash
source /opt/ros/humble/setup.bash
```

- rosdep 초기화 및 의존성 업데이트

```bash
sudo rosdep init && rosdep update
```

패키지 설치

- `ros2` 디렉토리로 이동 후 ROS 의존성 설치

```bash
cd ros2
rosdep install --from-paths src -y --ignore-src
```

- `rbq_ros2` 패키지 빌드

```bash
colcon build --symlink-install
```

▶️ 노드 실행

먼저 작업 공간 디렉토리로 이동합니다:

```bash
cd <your workspace>/RBQ
```

- **'실제 로봇'**에서 `rbq_driver`를 실행하는 방법:

이 섹션은 **실제 RBQ 로봇**을 조작하는 방법을 설명합니다.

📖 *자세한 조작 단계는 **운영 가이드**를 참조하세요.*

---

1. PC를 로봇의 Wi-Fi에 연결

📝 *Wi-Fi SSID:* `RBQ_xxxx`.

---

2. SSH로 로봇 PC에 접속

```bash
ssh rbq@192.168.0.10
```

⚠️ **Simple-RL 실행 전에 반드시 로봇을 Auto Start 하세요**

---

3. 로봇 PC에서 프로세스 실행

로봇 PC에서 프로세스를 실행하려면 다음 순서로 진행합니다.

1. **C to LAN 허브**를 이용해 **로봇 PC**에서 네트워크를 연결합니다.
2. **SSH로 로봇 PC에 접속**합니다 (위 2번 단계 참조: `ssh rbq@192.168.0.10`).
3. 접속한 로봇 PC에서, 개발자 PC와 **동일한 과정**으로 저장소 클론 및 설치를 진행합니다.
   - 작업할 디렉토리(예: `cd ~`)에서 `git clone https://github.com/RainbowRobotics/RBQ.git` 후, 클론된 **RBQ** 디렉토리로 이동한 뒤, 시스템 설정(ROS Humble 설치·환경 불러오기, rosdep 초기화 등), 패키지 설치(`cd ros2` 후 `rosdep install`, `colcon build`)를 위 🛠️ **설치** 섹션과 같이 수행합니다.
4. 권한 문제가 발생하면 `sudo` 입력 후 해당 명령을 재시도하세요.

> **참고**: 아래 명령은 개발자 PC가 아닌 **실제 로봇(SSH로 접속한 로봇 PC)**에서 실행되어야 합니다.

- **'개발자 PC'**에서 **실제 로봇**으로 ROS를 사용하려면 다음 명령을 실행하세요:

```bash
cd <your workspace>/RBQ
source ros2/install/local_setup.bash
./scripts/start_ros_driver.bash
```

> **참고**: 아래 명령은 실제 로봇(SSH로 접속한 로봇 PC)이 아닌 **개발자 PC**에서 실행되어야 합니다.

- **'개발자 PC'**에서 **시뮬레이션**으로 ROS를 사용하려면 다음 명령을 실행하세요:

```bash
cd <your workspace>/RBQ
source ros2/install/local_setup.bash
./scripts/sim.bash --ros
```

---

4. 개발자 PC에서 DDS 설정 (로봇 PC로 토픽 전송 시)

개발자 PC에서 로봇 PC로 DDS를 보내는 터미널에서는 아래 환경 변수를 설정해야 합니다. 이 터미널에서 `ros2 topic list`, `ros2 topic pub` 등을 실행할 때 사용합니다.

- **도메인 ID**는 `0`으로 고정합니다.
- **네트워크 인터페이스**는 본인 PC에 맞게 설정합니다. `wlp4s0`는 예시이며, `ifconfig`로 확인한 인터페이스 이름으로 바꿔 넣으세요.

```bash
export CYCLONEDDS_URI='<CycloneDDS><Domain><Id>0</Id><General>
<Interfaces><NetworkInterface name="wlp4s0"/></Interfaces></General>
</Domain></CycloneDDS>'
export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
```

아래 `ifconfig` 출력에서 사용할 인터페이스(예: Wi-Fi는 `wlp4s0`, 유선은 `eth0` 등)를 확인한 뒤, `CYCLONEDDS_URI`의 `name="wlp4s0"` 부분을 해당 이름으로 변경하면 됩니다.

인터페이스 확인:

```bash
ifconfig
```

5. 활성 토픽 목록 테스트

로봇 PC: 드라이버를 실행한 터미널에서 위 3번의 환경 변수가 설정된 상태라면, 같은 터미널에서 `ros2 topic list`로 확인할 수 있습니다.

개발자 PC: 로봇 PC로 토픽을 보내거나 확인하려면, **4번의 DDS 설정(도메인 0, Cyclone DDS)**을 적용한 터미널에서 아래를 실행하세요.

```bash
source ros2/install/local_setup.bash
ros2 topic list
```

- RBQ 드라이버는 ROS 2를 통해 RBQ를 제어하는 데 필요한 모든 토픽을 포함하고 있습니다.

rbq_driver 프로세스는 다음 노드들을 시작합니다:

- 로봇 상태 퍼블리셔

- 센서 피드백 퍼블리셔 (IMU, 배터리, 발 접촉)
- 명령 구독자
- 비전 데이터 퍼블리셔 (전방, 후방, 하방 카메라 스트림)

⚡ Quick Start

이 섹션에서는 ROS 2 토픽을 통해 로봇을 움직이는 데 필요한 핵심 명령어를 안내합니다.

1. HighLevel Command 모드로 전환

> **📍 중요 - 모드 전환 필수**
>
> `HighLevelCommand`를 통해 로봇을 실제로 움직이려면, **반드시** `rbq/motion/switchControlMode`를 `true`로 설정해야 합니다.

`true`	**HighLevel Mode** - 로봇이 `rbq/motion/cmd_highLevel`의 명령을 수신합니다
`false`	**JoyStick Mode** - 조이스틱으로 로봇을 제어합니다 (기본값)

```bash
ros2 topic pub --once /rbq/motion/switchControlMode std_msgs/msg/Bool "{data: true}"
```

2. Auto Start (관절 초기화 및 제어 시작)

> ⚠️ `autoStart` 실행 전, 로봇이 평평한 바닥 위에서 올바른 **초기 자세**인지 확인하세요. 테스트 중이라면 먼저 로봇을 앉히세요 (`switchGait gait_id: 0`).

```bash
ros2 topic pub --once /rbq/motion/autoStart std_msgs/msg/Bool "{data: true}"
```

3. 보행 전환 (Sit → Stand → Walk)

`rbq/motion/switchGait`에 목표 `gait_id`를 지정하여 로봇의 보행 상태를 변경합니다.

Sit (gait_id: 0)

```bash
ros2 topic pub --once /rbq/motion/switchGait std_msgs/msg/Int8 "{data: 0}"
```

Stand (gait_id: 1)

```bash
ros2 topic pub --once /rbq/motion/switchGait std_msgs/msg/Int8 "{data: 1}"
```

Walk (gait_id: 3)

```bash
ros2 topic pub --once /rbq/motion/switchGait std_msgs/msg/Int8 "{data: 3}"
```

> 사용 가능한 보행 상태 :

State ID	Name	Description
-2	Fall Mode	예기치 않게 균형을 잃었을 때 진입하는 모드
-1	Control Off	모든 구동기가 비활성화됨
0	Sitting	낮은 자세, 바닥에 앉아 있는 상태
1	Standing	기본 자세, 보행 준비 상태
2	Aim Mode	조준을 위한 조준 자세
3	Walk Mode	트롯 보행 모드
4	Stairs Mode	카메라 센서를 이용한 계단 적응 보행 모드
5	Wave Mode	워크 보행 모드
6	Run Mode	고속 보행 모드 (지원되는 경우)
30	RL Trot	강화학습 트롯 보행
31	RL Front Walk	강화학습 Front Walk 보행
33	RL Left Walk	강화학습 Left Walk 보행
34	RL Right Walk	강화학습 Right Walk 보행
35	RL Bound	강화학습 바운드 보행
36	RL Pace	강화학습 페이스 보행

State ID	Name	Description
37	RL Pronk	강화학습 프롱크 보행
38-41	RL 3Leg	강화학습 3다리 보행 (HR, HL, FR, FL)
42	RL Trot Vision	강화학습 비전 트롯 보행
45	RL Trot Run	강화학습 트롯 런 보행
46	RL Silent	강화학습 사일런트 보행

4. 속도 명령 전송

로봇이 Walk Mode (gait_id: 3)에 진입한 후, `rbq/motion/cmd_highLevel`로 속도 명령을 전송합니다.

0.3 m/s로 전진

```bash
ros2 topic pub --once /rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand \
'{header: {stamp: {sec: 0, nanosec: 0}, frame_id: "base"},
  identifier: "walk_fwd",
  roll: 0.0,
  pitch: 0.0,
  yaw: 0.0,
  vel_x: 0.3,
  vel_y: 0.0,
  omega_z: 0.0,
  delta_body_h: 0.0,
  delta_foot_h: 0.0,
  gait_state: 3,
  gait_transition: false}'
```

45° 대각선 이동 (전진 + 측방향)

```bash
ros2 topic pub --once /rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand \
'{header: {stamp: {sec: 0, nanosec: 0}, frame_id: "base"},
  identifier: "walk_diag",
  roll: 0.0,
  pitch: 0.0,
  yaw: 0.0,
  vel_x: 0.3,
  vel_y: 0.3,
  omega_z: 0.0,
  delta_body_h: 0.0,
  delta_foot_h: 0.0,
```

```
      gait_state: 3,
      gait_transition: false}'
```

> 💡 45° 대각선 이동은 동일한 `vel_x`와 `vel_y`를 설정하여 구현합니다 (0.3 × cos45° ≈ 0.212 m/s).

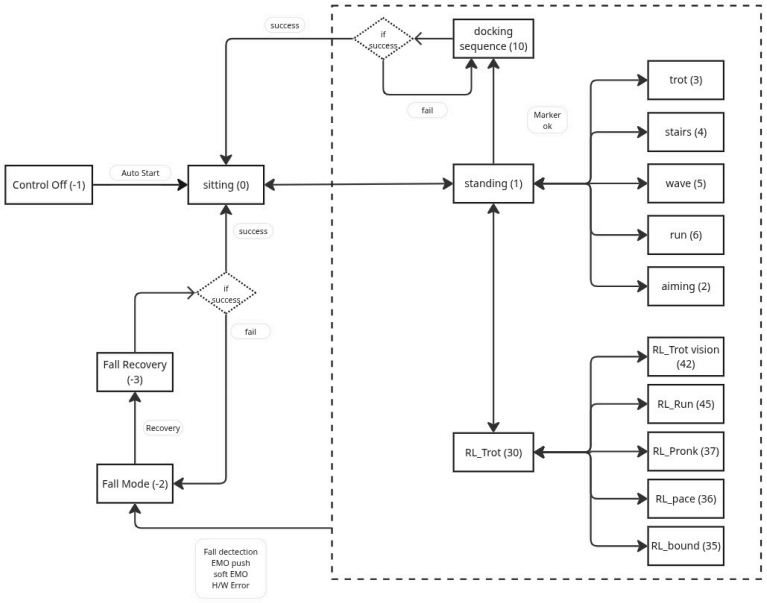
> HighLevelCommand 메시지 정의 :

```bash
std_msgs/Header header          # ROS 메시지 헤더 (타임스탬프와 frame_id 포함)
string identifier               # 명령 추적/로깅용 식별자
float64 roll                    # 몸체 roll 각도 (도 단위, ZYX 오일러 각)
float64 pitch                   # 몸체 pitch 각도 (도 단위, ZYX 오일러 각)
float64 yaw                     # 몸체 yaw 각도 (도 단위, ZYX 오일러 각)
float64 vel_x                   # X 방향 선형 속도 (m/s)
float64 vel_y                   # Y 방향 선형 속도 (m/s)
float64 omega_z                 # Z축 주위 각속도 (deg/s)
float64 delta_body_h            # 기본값에서 몸체 높이 조정 (m)
float64 delta_foot_h            # 기본값에서 발 높이 조정 (m)
int8    gait_state              # 목표 보행 상태 (위에 정의된 상수 사용)
bool    gait_transition         # 명령 실행 중 보행 전환 활성화(true)/비활성화
(false)
```

📋 토픽

📡 1. 보행 토픽 (Gait State)

Gait state는 로봇의 현재 보행 모드나 자세 모드를 의미합니다. 이 상태 전환은 상위 제어, 센서 조건, 내부 로직에 의해 발생합니다.



> 사용 가능한 보행 :

State ID	상태	설명
-2	Fall Mode	예기치 않게 균형을 잃었을 때 진입하는 모드
-1	Control Off	모든 구동기가 비활성화됨
0	Sitting	낮은 자세, 바닥에 앉아 있는 상태
1	Standing	기본 자세, 보행 준비 상태
2	Aim Mode	조준을 위한 조준 자세
3	Walk Mode	트롯 보행 모드
4	Stairs Mode	카메라 센서를 이용한 계단 적응 보행 모드
5	Wave Mode	워크 보행 모드
6	Run Mode	고속 보행 모드 (지원되는 경우)
30	RL Trot	강화학습 트롯 보행
31	RL Front Walk	강화학습 물구나무 보행
33	RL Left Walk	강화학습 레프트 워크 보행
34	RL Right Walk	강화학습 라이트 워크 보행
35	RL Bound	강화학습 바운드 보행
36	RL Pace	강화학습 페이스 보행

State ID	상태	설명
37	RL Pronk	강화학습 프롱크 보행
38-41	RL 3Leg	강화학습 3다리 보행 (HR, HL, FR, FL)
42	RL Trot Vision	강화학습 비전 트롯 보행
45	RL Trot Run	강화학습 트롯 런 보행
46	RL Silent	강화학습 사일런트 보행

> 두 번째 열에 있는 **사용 가능한 명령(Available Commands)**은 각 보행 상태에서 사용할 수 있는 명령을 보여줍니다. 반대로, **공통 명령(Common Commands)**과 **전원 제어(Power Control)**는 보행 상태와 관계없이 항상 사용할 수 있습니다.
> **참고**: 모든 보행 전환은 특정 `gait_id` 값을 사용하는 통합된 `rbq/motion/switchGait` 토픽을 사용합니다.
> 사용 가능한 gait_id 값은 위의 "사용 가능한 보행 상태(Available Gait States)" 표를 참조하세요.

보행 상태 (Gait State)	사용 가능한 명령(Available Commands)	공통 명령 (Common Commands)	전원 제어(Power Control)
Control Off	rbq/motion/autoStart	rbq/motion/switchGait (gait_id: 0-46) rbq/motion/cmd_highLevel rbq/motion/emergency rbq/motion/staticLock rbq/motion/staticReady rbq/motion/staticGround rbq/motion/switchControlMode	rbq/powerControl/setPortState # PDU Port ID int8 PDU_PORT_48V_LEG = 0x00 int8 PDU_PORT_48V_ADD = 0x01 int8 PDU_PORT_48V_EXT = 0x02 int8 PDU_PORT_12V_VisionPC = 0x10 int8 PDU_PORT_12V_COMM = 0x11 int8 PDU_PORT_12V_Lidar = 0x12 int8 PDU_PORT_12V_CCTV = 0x13 int8 PDU_PORT_12V_THER = 0x14 int8 PDU_PORT_12V_IRLed =
Sitting Mode	rbq/motion/canCheck		
Standing Mode	rbq/motion/cmd_navigateTo rbq/stateEstimation/comEstimationCompensation		
Walk Mode	-		
Stairs Mode	-		
Run Mode	-		

보행 상태 (Gait State)	사용 가능한 명령(Available Commands)	공통 명령 (Common Commands)	전원 제어(Power Control)
Wave Mode	rbq/motion/cmd_navigateTo	(true:HighLevel Mode, false:JoyStick Mode)	0x15 int8 PDU_PORT_12V_Speaker = 0x16 int8 PDU_PORT_5V_CAMERAS = 0x20 int8 PDU_PORT_5V_AUDIO_SIDE_CAM_USBHUB = 0x21
RL Mode	-		
Fall Mode	rbq/motion/recoveryFlex rbq/motion/switchGait (gait_id: 1)		

📡 2. 로우 레벨 토픽 (Low-level State Topics)

이 토픽들은 로봇의 내부 상태, 즉 자세, 센서값, 시스템 상태에 대한 실시간 피드백을 제공합니다.

토픽	데이터 구조	설명
rbq/status/robot_status	std_msgs/Header header bool con_start bool ready_pos bool ground_pos bool force_con bool ext_joy bool is_standing bool can_check bool find_home int8 gait_id bool is_fall int8 docking_state bool imu_success	이 메시지는 로봇 상태 정보를 포함하며 50Hz로 발행됩니다. 모터 제어 상태, 위치 상태, 통신 상태, 보행 정보, 도킹 상태를 포함합니다. 사용 가능한 gait_id 값은 위의 "사용 가능한 보행" 표를 참조하세요. # Docking States int8 DOCKING_MAX_FAIL_CNT_REACHED = -6 int8 DOCKING_MARKER_POS_INVALID_ROTATION = -5 int8 DOCKING_MARKER_POS_INVALID_TOO_FAR = -4 int8 DOCKING_MARKER_POS_INVALID_WRONG_DIR = -3 int8 DOCKING_MARKER_NOT_FOUND = -2 int8 DOCKING_FAILED = -1 int8 DOCKING_OPERATION_MODE = 0 int8 DOCKING_APPROACH_OFFSET = 1 int8 DOCKING_APPROACH = 2 int8 DOCKING_APPROACH_WIDE = 3 int8 DOCKING_SIT_DOWN = 4 int8 DOCKING_SUCCESS = 5 int8 DOCKING_SUCCESS_CHARGING = 6 int8 DOCKING_SUCCESS_NO_CHARGING = 7

토픽	데이터 구조	설명
rbq/powerControl/battery_status	std_msgs/Header header string identifier float64 charge_percentage float64 current float64 voltage float64[] temperatures uint8 status	# Status uint8 STATUS_UNKNOWN = 0 uint8 STATUS_MISSING = 1 uint8 STATUS_CHARGING = 2 uint8 STATUS_DISCHARGING = 3 uint8 STATUS_BOOTING = 4
rbq/stateEstimation/footStates	std_msgs/Header header geometry_msgs/Point[] foot_position_rt_body geometry_msgs/Point[] foot_velocity_rt_body uint8[] contact	로컬 바디 프레임 기준의 발 위치와 속도입니다. 바디 프레임의 원점은 로봇 몸체의 중심에 있으며, 앞쪽 방향은 +x, 왼쪽 방향은 +y, 위쪽 방향은 +z로 정의됩니다. 모든 배열은 4개의 요소를 포함합니다 (각 다리당 하나). # Contact uint8 CONTACT_UNKNOWN = 0 uint8 CONTACT_MADE = 1 uint8 CONTACT_LOST = 2 # Leg number Leg num 0 : Right-Hind leg (RH) Leg num 1 : Left-Hind Leg (LH) Leg num 2 : Right-Front Leg (RF) Leg num 3 : Left-Front Leg (LF)
rbq/joint/joint_status	std_msgs/Header header bool[] connected int8[] temperature int8[] motor_temp bool[] status_fet bool[] status_run bool[] status_init bool[] status_mod bool[] status_non_ctr bool[] status_bat bool[] status_calib bool[] status_mt_err bool[] status_jam	이 메시지는 연결 상태, 온도, 모터 상태 플래그, 제어 데이터를 포함한 상세한 관절 정보를 포함합니다. 각 관절에 대해 50Hz로 발행됩니다.

토픽	데이터 구조	설명
	bool[] status_cur bool[] status_big bool[] status_inp bool[] status_flt bool[] status_tmp bool[] status_ps1 bool[] status_ps2 bool[] status_rsvd float32[] position_ref float32[] position_enc float32[] velocity float32[] torque_ref float32[] current float32[] kp float32[] kd int32[] owner	

> 💡 **예시** : 각 상태 토픽 구독

로봇 상태

```bash
ros2 topic echo --once rbq/status/robot_status
```

관절 상태

```bash
ros2 topic echo --once rbq/joint/joint_status
```

배터리 상태

```bash
ros2 topic echo --once rbq/powerControl/battery_status
```

발 상태

```bash
ros2 topic echo --once rbq/stateEstimation/footStates
```

> 상태(State) 메시지는 다음과 같이 정의됩니다:

- RobotStatus

```bash
# DOCKING_STATE
int8 DOCKING_MAX_FAIL_CNT_REACHED = -6           # (중단 시퀀스) 최대 도킹 재시도 횟수 도달 (10회)
int8 DOCKING_MARKER_POS_INVALID_ROTATION  = -5   # (중단 시퀀스) 마커 회전 각도가 +-40도 초과
int8 DOCKING_MARKER_POS_INVALID_TOO_FAR   = -4   # (중단 시퀀스) 마커가 5m 이상 너무 멀리 있음
int8 DOCKING_MARKER_POS_INVALID_WRONG_DIR = -3   # (중단 시퀀스) 마커가 로봇 전면에서 감지됨
int8 DOCKING_MARKER_NOT_FOUND = -2               # (중단 시퀀스) 마커를 찾을 수 없음
int8 DOCKING_FAILED           = -1               # 도킹 실패 --> 자동 재시도
int8 DOCKING_OPERATION_MODE   = 0                # 로봇이 정상 작동 중
int8 DOCKING_APPROACH_OFFSET  = 1                # 1차 오프셋으로 충전기에 도달
int8 DOCKING_APPROACH         = 2                # 2차 충전기에 도달
int8 DOCKING_APPROACH_WIDE    = 3                # 3차 넓은 자세로 충전기에 도달
int8 DOCKING_SIT_DOWN         = 4                # 충전기에 앉기
int8 DOCKING_SUCCESS          = 5                # 도킹 성공 : 충전기 연결됨
int8 DOCKING_SUCCESS_CHARGING = 6                # 도킹 성공 : 충전 중
int8 DOCKING_SUCCESS_NO_CHARGING = 7             # 도킹 성공 : 충전 안됨

# gait_state constants
int8 STATE_FALL                = -2    # Fall Mode - 예상치 못한 균형 상실 시 트리거됨
int8 STATE_OFF                 = -1    # Control Off - 모든 액추에이터 비활성화
int8 STATE_SIT                 = 0     # Sitting - 낮은 자세로 바닥에 휴식
int8 STATE_STAND               = 1     # Standing - 중립 자세로 보행
int8 STATE_AIM                 = 2     # Aim Mode - 타겟팅을 위한 조준 자세 준비
int8 STATE_WALK                = 3     # Walk Mode - walking Trot 보행
int8 STATE_STAIRS              = 4     # Stairs Mode - 카메라 센서를 사용한 계단 적응 보행
int8 STATE_WAVE                = 5     # Wave Mode - walking Walk 보행
int8 STATE_RUN                 = 6     # Run Mode - 고속 보행 (지원되는 경우)
int8 STATE_RL_TROT             = 30    # RL Trot - 강화 학습 Trot 보행
int8 STATE_RL_FRONT_WALK       = 31    # RL Front Walk - 강화 학습 Front Walk 보행
int8 STATE_RL_LEFT_WALK        = 33    # RL Left Walk - 강화 학습 Left Walk 보행
```

```
int8 STATE_RL_RIGHT_WALK       = 34    # RL Right Walk - 강화 학습 Right
Walk 보행
int8 STATE_RL_BOUND            = 35    # RL Bound - 강화 학습 Bound 보행
int8 STATE_RL_PACE             = 36    # RL Pace - 강화 학습 Pace 보행
int8 STATE_RL_PRONK            = 37    # RL Pronk - 강화 학습 Pronk 보행
int8 STATE_RL_3LEG_HR          = 38    # RL 3Leg HR - 강화 학습 3-Leg 보행
(Hind Right)
int8 STATE_RL_3LEG_HL          = 39    # RL 3Leg HL - 강화 학습 3-Leg 보행
(Hind Left)
int8 STATE_RL_3LEG_FR          = 40    # RL 3Leg FR - 강화 학습 3-Leg 보행
(Front Right)
int8 STATE_RL_3LEG_FL          = 41    # RL 3Leg FL - 강화 학습 3-Leg 보행
(Front Left)
int8 STATE_RL_TROT_VISION      = 42    # RL Trot Vision - 강화 학습
Vision을 사용한 Trot 보행
int8 STATE_RL_TROT_RUN         = 45    # RL Trot Run - 강화 학습 Trot Run
보행
int8 STATE_RL_SILENT           = 46    # RL Silent - 강화 학습 Silent 보행

std_msgs/Header header
bool con_start          # 모터 제어 활성화 (1 = 활성화, 0 = 비활성화)
bool ready_pos          # 로봇이 준비 위치에 있음 (1 = 준비됨, 0 = 준비 안됨)
bool ground_pos         # 로봇이 바닥/앉은 위치에 있음 (1 = 앉음, 0 = 앉지 않
음)
bool force_con          # 힘 제어 모드 활성화 (1 = 활성화, 0 = 비활성화)
bool ext_joy            # 외부 조이스틱 연결됨 (1 = 연결됨, 0 = 연결 안됨)
bool is_standing        # 로봇이 서 있음 (1 = 서있음, 0 = 서있지 않음)
bool can_check          # CAN 통신 확인 (1 = 성공, 0 = 실패)
bool find_home          # 엔코더 홈 찾기 상태 (1 = 성공, 0 = 실패)
int8 gait_id            # 현재 보행 모드 식별자 (GAIT_STATE 열거형에 정의된 값
반환)
bool is_fall            # 낙하 감지 상태 (1 = 로봇이 넘어짐, 0 = 정상)
int8 docking_state      # 도킹 프로세스 상태 (DOCKING_STATE 열거형에 정의된 값
반환)
bool imu_success        # IMU 연결 상태 (1 = 성공, 0 = 실패)
```

- JointStatus

```bash
std_msgs/Header header

# 관절 상세 정보
# 연결 및 온도 상태
bool[] connected          # 관절 연결 상태 (true = 연결됨, false = 연결 안됨)
int8[] temperature        # 보드 온도 (섭씨)
int8[] motor_temp         # 모터 온도 (섭씨)

# 모터 상태 플래그
bool[] status_fet         # FET (전계 효과 트랜지스터) 상태 (true = ON,
false = OFF)
bool[] status_run         # 모터 실행 상태 (true = 실행 중, false = 정지)
```

```
    bool[] status_init          # 초기화 상태 (true = 초기화됨, false = 초기화 안
    됨)
    bool[] status_mod           # 제어 모드 상태 (true = 위치 제어, false = 토크 제
    어)
    bool[] status_non_ctr       # non 카운트 오류 (true = 오류, false = 정상)
    bool[] status_bat           # 배터리 상태 (true = 배터리 부족, false = 정상)
    bool[] status_calib         # 보정 상태 (true = 보정 모드, false = 정상)
    bool[] status_mt_err        # 모터 오류 상태 (true = 오류, false = 정상)
    bool[] status_jam           # JAM 오류 상태 (true = 걸림, false = 정상)
    bool[] status_cur           # 과전류 오류 (true = 과전류, false = 정상)
    bool[] status_big           # 큰 위치 오류 (true = 큰 오류, false = 정상)
    bool[] status_inp           # 입력 오류 (true = 입력 오류, false = 정상)
    bool[] status_flt           # FET 드라이버 결함 오류 (true = 결함, false = 정
    상)
    bool[] status_tmp           # 온도 오류 (true = 과온, false = 정상)
    bool[] status_ps1           # 위치 제한 오류 - 하한 (true = 제한 도달, false =
    정상)
    bool[] status_ps2           # 위치 제한 오류 - 상한 (true = 제한 도달, false =
    정상)
    bool[] status_rsvd          # 예약된 상태 비트

    # 제어 데이터
    float32[] position_ref      # 기준 위치 (라디안)
    float32[] position_enc      # 엔코더 위치 (라디안)
    float32[] velocity          # 관절 속도 (deg/s)
    float32[] torque_ref        # 기준 토크 (Nm)
    float32[] current           # 모터 전류 (암페어)
    float32[] kp                # 비례 이득 (P 이득)
    float32[] kd                # 미분 이득 (D 이득)
    int32[] owner               # 관절 소유자 ID (어떤 컨트롤러가 이 관절을 소유하는지)
```

- BatteryState

```bash
    # Status
    uint8 STATUS_UNKNOWN = 0
    uint8 STATUS_MISSING = 1
    uint8 STATUS_CHARGING = 2
    uint8 STATUS_DISCHARGING = 3
    uint8 STATUS_BOOTING = 4

    std_msgs/Header header
    string identifier
    float64 charge_percentage
    float64 current
    float64 voltage
    float64[] temperatures
    uint8 status
```

- FootStates

```bash
# Contact constants
uint8 CONTACT_UNKNOWN = 0
uint8 CONTACT_MADE = 1
uint8 CONTACT_LOST = 2

std_msgs/Header header

# Arrays for 4 legs (FL, FR, RL, RR)
geometry_msgs/Point[] foot_position_rt_body    # Foot position relative
to body frame (4 elements)
geometry_msgs/Point[] foot_velocity_rt_body    # Foot velocity relative
to body frame (4 elements)
uint8[] contact                                 # Contact state for
each foot (4 elements)
```

🎮 3. 제어 토픽 (Command Topics)

다음 토픽들은 RBQ 로봇에 상위 제어 및 내비게이션 명령을 보내는 데 사용됩니다.

모든 명령은 표준 ROS2 메시지 타입 또는 사용자 정의 `rbq_msgs` 정의를 따릅니다.

> 제어 토픽은 다음과 같이 정의됩니다:

토픽	설명
rbq/motion/autoStart	관절 위치를 초기화하고 관절 제어를 시작합니다 AutoStart 프로세스는 'RobotStatus' 토픽을 통해 확인할 수 있습니다 CAN 확인 → 홈 찾기 → 제어 시작 플래그가 활성화됨
rbq/motion/emergency	비상 정지 - 모든 다리 관절을 high damping 모드로 전환합니다. 이 동작은 로봇에 충격을 줄 수 있습니다.
rbq/motion/switchGait	gait_id 매개변수를 사용하여 로봇 보행 모드를 전환합니다. 사용 가능한 gait_id 값은 위의 "사용 가능한 보행" 표를 참조하세요.
rbq/motion/recoveryFlex rbq/motion/switchGait (gait_id: 1)	'recoveryFlex' 토픽은 로봇이 Fall 모드에 있을 때만 적용됩니다. 특정 관절 동작을 통해 로봇을 앉은 모드로 복귀시키도록 명령합니다. 또는 'switchGait' 토픽에 gait_id: 1 (Standing)을 사용하여 Fall 모드에서 복구할 수 있습니다.
rbq/motion/switchControlMode	HighLevel Command 모드를 활성화하거나 비활성화합니다. `true`로 설정하면 HighLevel Command 모드가 활성화되어 `rbq/motion/cmd_highLevel` 토픽에서 명령을 수신합니다. `false`로 설정하면 Joystick 제어 모드가 활성화됩니다.

> 💡 **예시** : 각 제어 토픽 발행

Auto Start (관절 초기화 및 제어 시작)

> ⚠️ `autoStart` 실행 전, 로봇이 평평한 바닥 위에서 올바른 **초기 자세**인지 확인하세요. 테스트 중이라면 먼저 로봇을 앉히세요 (`switchGait gait_id: 0`).

```bash
ros2 topic pub --once /rbq/motion/autoStart std_msgs/msg/Bool "{data: true}"
```

비상정지 (Emergency Stop)

> ⚠️ 이 명령은 모든 다리 관절을 **high damping** 모드로 전환합니다 - 로봇이 즉시 주저앉습니다. 테스트 중이라면 이 명령을 사용하기 전에 먼저 로봇을 앉히세요 (`switchGait gait_id: 0`).

```bash
ros2 topic pub --once /rbq/motion/emergency  std_msgs/msg/Bool "{data: true}"
```

보행 전환 (예: Walk Mode, gait_id: 3)

```bash
ros2 topic pub --once /rbq/motion/switchGait std_msgs/msg/Int8 "{data: 3}"
```

Fall 모드에서 복구

```bash
ros2 topic pub --once /rbq/motion/recoveryFlex std_msgs/msg/Bool "{data: true}"
```

HighLevel Command 모드로 전환

```bash
ros2 topic pub --once rbq/motion/switchControlMode std_msgs/msg/Bool '{data: true}'
```

JoyStick 모드로 전환

```bash
ros2 topic pub --once rbq/motion/switchControlMode std_msgs/msg/Bool
'{data: false}'
```

Static Lock (현재 위치에서 관절 고정)

```bash
ros2 topic pub --once /rbq/motion/staticLock std_msgs/msg/Bool "{data:
true}"
```

Static Ready

```bash
ros2 topic pub --once /rbq/motion/staticReady std_msgs/msg/Bool "{data:
true}"
```

Static Ground

```bash
ros2 topic pub --once /rbq/motion/staticGround std_msgs/msg/Bool "
{data: true}"
```

> 상위 제어 명령 토픽은 다음과 같이 정의됩니다:

토픽	데이터 구조	설명
rbq/motion/cmd_highLevel	std_msgs/Header header string identifier float64 roll float64 pitch float64 yaw float64 vel_x float64 vel_y	현재 보행 상태에 따른 로봇 제어를 위한 상위 제어 명령 토픽. (사용 가능한 gait_id 값은 위의 "사용 가능한 보행" 표를 참조하세요) • **Standing Mode:** roll, pitch, yaw (ZYX 오일러 각)에 따라 로봇 몸체 자세를 조정합니다. body_H 매개변수는 로봇 몸체의 높이를 지정합니다 범위: Roll: -25~25°, Pitch: -20~20°, Yaw: -25~25°, Delta_body_h: -0.15~0.05m • **Walk Mode:** 트롯 보행 상태에서 로봇의 로컬 좌표계 기준 이동 속도를 설정합니다. Vx, Vy는 전진 속도, 측방향 속도를 나타내고 Wz는 회전 속도를 나타냅니다. 트롯 보행 중 delta_body_H는 기본 몸체 높이에서의 몸체 높이를 설정하는 데 사용되고, foot_H는 발 높이를 설정하는 데 사용되며, pitch는 몸체의 피치 각도를 설정하는 데 사용됩니다. 범위: Vx: -1.0~1.2m/s, Vy: -0.4~0.4m/s, Wz: -75~75°/s Delta_body_h: -0.15~+0.05m, Delta_foot_h: -0.06~+0.04m • **Stairs Mode:** 계단 내비게이션을 위한 전진, 측방향, 회전 속도

토픽	데이터 구조	설명
	float64 omega_z float64 delta_body_h float64 delta_foot_h int8 gait_state bool gait_transition	범위: Vx: -0.5~0.5m/s, Vy: -0.2~0.2m/s, Wz: -15~15°/s • **Run Mode:** 고속 이동 제어를 위한 전진, 측방향, 회전 속도 범위: Vx: -1.0~1.8m/s, Vy: -0.6~0.6m/s, Wz: -75~75°/s • **Wave Mode:** 저속 이동 제어를 위한 전진, 측방향, 회전 속도 범위: Vx: -0.3~0.3m/s, Vy: -0.2~0.2m/s, Wz: -20~20°/s • **RL TROT Mode:** 향상된 속도 제어를 갖춘 강화학습 트롯 보행 범위: Vx: -1.5~2.0m/s, Vy: -1.0~1.0m/s, Wz: -75~75°/s • **RL TROT VISION Mode:** 비전 통합을 갖춘 강화학습 트롯 범위: Vx: -1.5~2.0m/s, Vy: -1.0~1.0m/s, Wz: -75~75°/s
rbq/motion/cmd_navigateTo	geometry_msgs/Pose pose uint8 mode	로봇을 지정된 위치로 이동시키는 내비게이션 명령: **# 접근 모드:** 0: 목표로 회전 → 직선 보행 → 목표 yaw로 회전 1: 목표 yaw로 회전 → 목표로 대각선 보행 2: 목표로 대각선 보행 → 목표 yaw로 회전 3: 목표 yaw로 회전하면서 동시에 대각선 보행 4: 넓은 다리 보행을 사용한 접근 모드 3

> 💡 **예시** : 상위 제어 및 내비게이션 명령

0.5 m/s로 전진 (Walk Mode)

```bash
ros2 topic pub --once /rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand '{header: {stamp: {sec: 0, nanosec: 0},
frame_id: "base"},
  identifier: "walk_fwd",
  roll: 0.0,
  pitch: 0.0,
  yaw: 0.0,
  vel_x: 0.5,
  vel_y: 0.0,
  omega_z: 0.0,
  delta_body_h: 0.0,
  delta_foot_h: 0.0,
```

```
  gait_state: 3,
  gait_transition: true}'
```

30°/s로 회전 (Walk Mode)

```bash
ros2 topic pub --once /rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand '{header: {stamp: {sec: 0, nanosec: 0},
frame_id: "base"},
  identifier: "turn",
  roll: 0.0,
  pitch: 0.0,
  yaw: 0.0,
  vel_x: 0.0,
  vel_y: 0.0,
  omega_z: 30.0,
  delta_body_h: 0.0,
  delta_foot_h: 0.0,
  gait_state: 3,
  gait_transition: true}'
```

제자리 정지 (영속도)

```bash
ros2 topic pub --once /rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand '{header: {stamp: {sec: 0, nanosec: 0},
frame_id: "base"},
  identifier: "stop",
  roll: 0.0,
  pitch: 0.0,
  yaw: 0.0,
  vel_x: 0.0,
  vel_y: 0.0,
  omega_z: 0.0,
  delta_body_h: 0.0,
  delta_foot_h: 0.0,
  gait_state: 3,
  gait_transition: true}'
```

월드 좌표 위치로 내비게이션 (x: 20cm, y: 10cm)

```bash
ros2 topic pub --once /rbq/motion/cmd_navigateTo
geometry_msgs/msg/PoseStamped "
header:
  frame_id: map
pose:
  position:
    x: 0.2
```

```
      y: 0.1
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  "
```

PDU 포트 제어 매핑 (setPortState용 port_id)

`rbq/powerControl/setPortState` 토픽에서 사용하는 포트 ID는 아래 enum과 동일합니다.

```cpp
enum PDU_PORT_IDs_e : unsigned char {
    PDU_PORT_48V_LEG                    = 0x00,
    PDU_PORT_48V_ADD                    = 0x01,
    PDU_PORT_48V_EXT                    = 0x02,

    PDU_PORT_12V_VisionPC               = 0x10,
    PDU_PORT_12V_COMM                   = 0x11,
    PDU_PORT_12V_Lidar                  = 0x12,
    PDU_PORT_12V_CCTV                   = 0x13,
    PDU_PORT_12V_THER                   = 0x14,
    PDU_PORT_12V_IRLed                  = 0x15,
    PDU_PORT_12V_Speaker                = 0x16,

    PDU_PORT_5V_CAMERAS                 = 0x20,
    PDU_PORT_5V_AUDIO_SIDE_CAM_USBHUB   = 0x21,
};
```

PDU 포트 제어 (예: 12V LiDAR 포트 ON)

12V LiDAR는 `PDU_PORT_12V_Lidar = 0x12` (18)입니다.

```bash
ros2 topic pub --once /rbq/powerControl/setPortState
std_msgs/msg/Int8MultiArray "{data: [18, 1]}"
```

> 상위 제어 명령(High-Level Command)의 메시지는 다음과 같이 정의됩니다:

- HighLevelCommand

```bash
std_msgs/Header header          # ROS 메시지 헤더 (타임스탬프와 frame_id 포함)
string identifier               # 명령 추적/로깅용 식별자
float64 roll                    # 몸체 roll 각도 (도 단위, ZYX 오일러 각)
float64 pitch                   # 몸체 pitch 각도 (도 단위, ZYX 오일러 각)
float64 yaw                     # 몸체 yaw 각도 (도 단위, ZYX 오일러 각)
float64 vel_x                   # X 방향 선형 속도 (m/s)
float64 vel_y                   # Y 방향 선형 속도 (m/s)
float64 omega_z                 # Z축 주위 각속도 (deg/s)
float64 delta_body_h            # 기본값에서 몸체 높이 조정 (m)
float64 delta_foot_h            # 기본값에서 발 높이 조정 (m)
int8    gait_state              # 목표 보행 상태 (위에 정의된 상수 사용)
bool    gait_transition         # 명령 실행 중 보행 전환 활성화(true)/비활성화
(false)
```

> 💡 **예제**: 사용자 정의 자세로 Standing 모드 (roll: 15°, pitch: 30°, yaw: 20°)

```bash
ros2 topic pub --once rbq/motion/cmd_highLevel
rbq_msgs/msg/HighLevelCommand \
'{header: {stamp: {sec: 0, nanosec: 0}, frame_id: "base"}, identifier:
"standing_pose", roll: 15.0, pitch: 30.0, yaw: 20.0, vel_x: 0.0, vel_y:
0.0, omega_z: 0.0, delta_body_h: 0.0, delta_foot_h: 0.0, gait_state: 1,
gait_transition: true}'
```

📷 4. 비전 토픽 (Vision Topics)

- PTZ 카메라 제어 토픽 (PTZ Camera Control Topics)

토픽	설명
rbq/ptzCamera/setPanTiltZoom	Float32MultiArray를 사용한 PTZ 카메라 제어 명령 [pan, tilt, zoom]

> example:

```bash
ros2 topic pub --once rbq/ptzCamera/setPanTiltZoom
std_msgs/msg/Float32MultiArray "{data: [0.0, 0.0, 1.0]}"
```

- 카메라 센서 토픽 (Camera Sensor Topics)

> 카메라 센서는 로봇 주변의 여러 위치에서 RGB, IR, 깊이 이미지를 제공합니다. 이미지는 원시 및 압축 형식으로 발행되며 해당 카메라 보정 정보와 함께 제공됩니다.
> 📍 **하드웨어 정보**: 카메라 사양, FOV, 변환 행렬은 매뉴얼 **하드웨어**의 **전·후방 카메라**, **지면 뷰 카메라**, LiDAR (OS1-32), LiDAR (MID-360), PTZ 카메라 페이지를 참조하세요.

카메라	이미지 타입	형식	토픽명
Bottom (0-3)	IR	Raw	rbq/vision/sensor_bottom_0/ir rbq/vision/sensor_bottom_1/ir rbq/vision/sensor_bottom_2/ir rbq/vision/sensor_bottom_3/ir
		Compressed	rbq/vision/sensor_bottom_0/ir/compressed rbq/vision/sensor_bottom_1/ir/compressed rbq/vision/sensor_bottom_2/ir/compressed rbq/vision/sensor_bottom_3/ir/compressed
		Camera Info	rbq/vision/sensor_bottom_0/ir/camera_info rbq/vision/sensor_bottom_1/ir/camera_info rbq/vision/sensor_bottom_2/ir/camera_info rbq/vision/sensor_bottom_3/ir/camera_info
	Depth	Raw	rbq/vision/sensor_bottom_0/depth rbq/vision/sensor_bottom_1/depth rbq/vision/sensor_bottom_2/depth rbq/vision/sensor_bottom_3/depth
		Compressed	rbq/vision/sensor_bottom_0/depth/compressed rbq/vision/sensor_bottom_1/depth/compressed rbq/vision/sensor_bottom_2/depth/compressed rbq/vision/sensor_bottom_3/depth/compressed
		Camera Info	rbq/vision/sensor_bottom_0/depth/camera_info rbq/vision/sensor_bottom_1/depth/camera_info rbq/vision/sensor_bottom_2/depth/camera_info rbq/vision/sensor_bottom_3/depth/camera_info
Front/Rear	RGB	Raw	rbq/vision/sensor_front/rgb rbq/vision/sensor_rear/rgb
		Compressed	rbq/vision/sensor_front/rgb/compressed rbq/vision/sensor_rear/rgb/compressed
		Camera Info	rbq/vision/sensor_front/rgb/camera_info rbq/vision/sensor_rear/rgb/camera_info
	IR	Raw	rbq/vision/sensor_front/ir rbq/vision/sensor_rear/ir
		Compressed	rbq/vision/sensor_front/ir/compressed rbq/vision/sensor_rear/ir/compressed

		Camera Info	rbq/vision/sensor_front/ir/camera_info rbq/vision/sensor_rear/ir/camera_info
	Depth	Raw	rbq/vision/sensor_front/depth rbq/vision/sensor_rear/depth
		Compressed	rbq/vision/sensor_front/depth/compressed rbq/vision/sensor_rear/depth/compressed
		Camera Info	rbq/vision/sensor_front/depth/camera_info rbq/vision/sensor_rear/depth/camera_info

---

🖼️ RViz 시각화

GUI와 함께 RViz2 실행

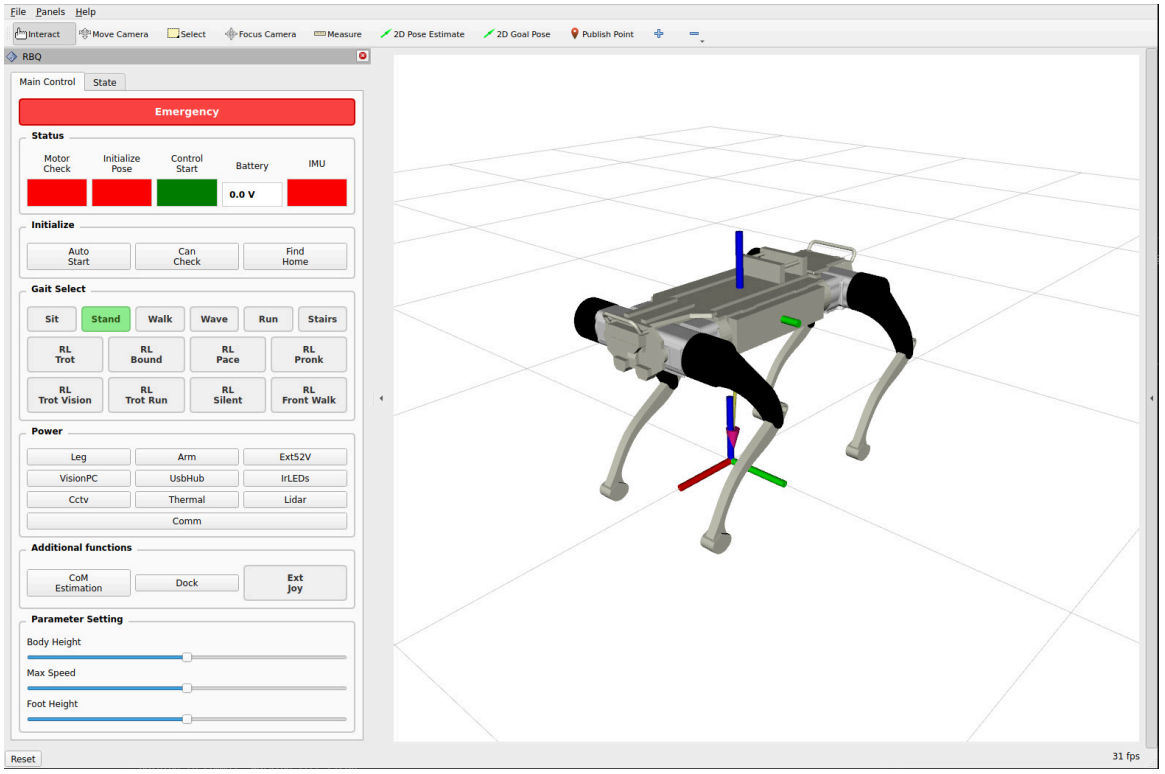
> RViz의 왼쪽 GUI를 통해 로봇을 동시에 시각화하고 명령을 전송할 수 있습니다.
> 먼저 워크스페이스 디렉토리로 이동하세요:

```bash
cd <your workspace>/RBQ
```

> 그런 다음 RViz를 실행하세요:

```bash
source ros2/install/local_setup.bash
./scripts/start_rviz.bash
```

RAINBOW ROBOTICS



📘 도움

이 저장소를 사용하는 동안 문제가 발생하면, 자유롭게 issue를 등록해 주세요.

Git Issue 등록하기

👥 기여자

이 프로젝트는 Rainbow-Robotics에서 제공합니다.

Rainbow-Robotics 기여자:

- Gurbann
- Jinwon Seo

RAINBOW ROBOTICS

1.5 RBQ GYM

📚 소개

RBQ gym 오픈소스 시뮬레이션 환경에 오신 것을 환영합니다

이 매뉴얼은 IsaacGym에서 RBQ 로봇을 **시뮬레이션**하여 강화학습으로 자체 보행 policy를 **학습**하고, **Play**로 평가한 후, 실제 로봇에 **배포**하는 방법에 대한 종합적인 가이드를 제공합니다.

💻 시스템 요구사항

RBQ gym 시뮬레이션 환경을 원활하게 실행하려면 다음 요구사항이 권장됩니다:

- OS: Ubuntu 22.04 (x86)
- 최소 PC 사양:
  - CPU: Intel Core i7 - 12세대
  - RAM: 16 GB
  - Storage: 25 GB
  - GPU: Nvidia RTX 4080
- 컨트롤 장치
  - 키보드

GPU 호환성

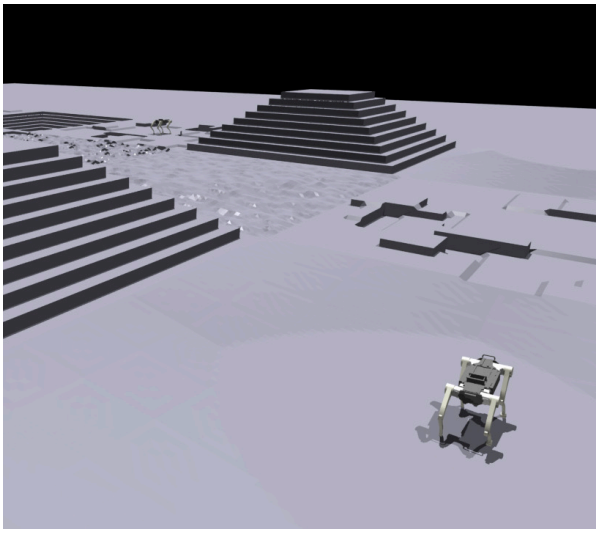
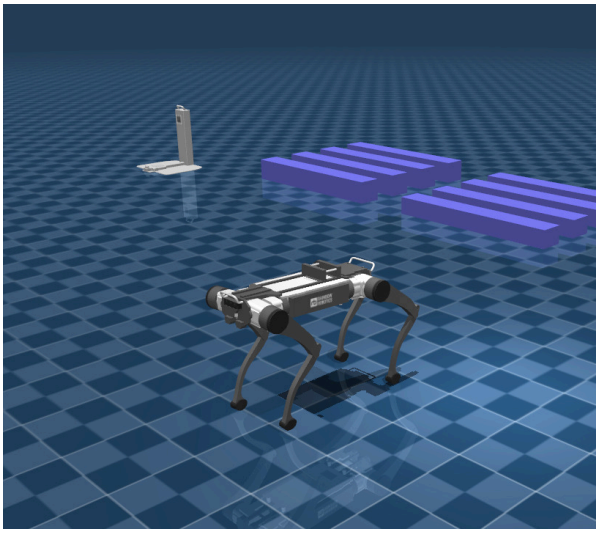
Isaac Gym은 **NVIDIA RTX 50 시리즈 (Blackwell)** GPU를 지원하지 않습니다. 학습에는 RTX 40 시리즈 이하의 GPU를 사용하세요.

🔁 프로세스 개요

강화학습을 통한 모션 제어의 기본 워크플로우는 다음과 같습니다:

Train → Play → Sim2Sim

- **Train**: IsaacGym 시뮬레이션 환경에서 로봇이 환경과 상호작용하며 PPO 알고리즘 기반 강화학습을 통해 설계된 보상을 최대화하는 Policy를 학습합니다.
- **Play**: Play 명령어로 IsaacGym 환경 내에서 학습된 Policy를 검증하고 기대한 대로 동작하는지 확인합니다.
- **Sim2Sim**: **Mujoco** 시뮬레이터에서 학습된 Policy를 평가하여 성능과 신뢰성을 확인합니다.
- **Deploy**: 평가된 Policy을 실제 로봇에 배포합니다.

Isaac Gym	Mujoco
	

📦 환경 설정

rbq_gym 디렉토리로 이동합니다:

```bash
cd <your workspace>/RBQ/rbq_gym
```

rbq_gym 디렉토리 내에서 다음 명령어를 실행하여 환경을 설정합니다:

```bash
bash scripts/setup.bash
```

🏋️ Train

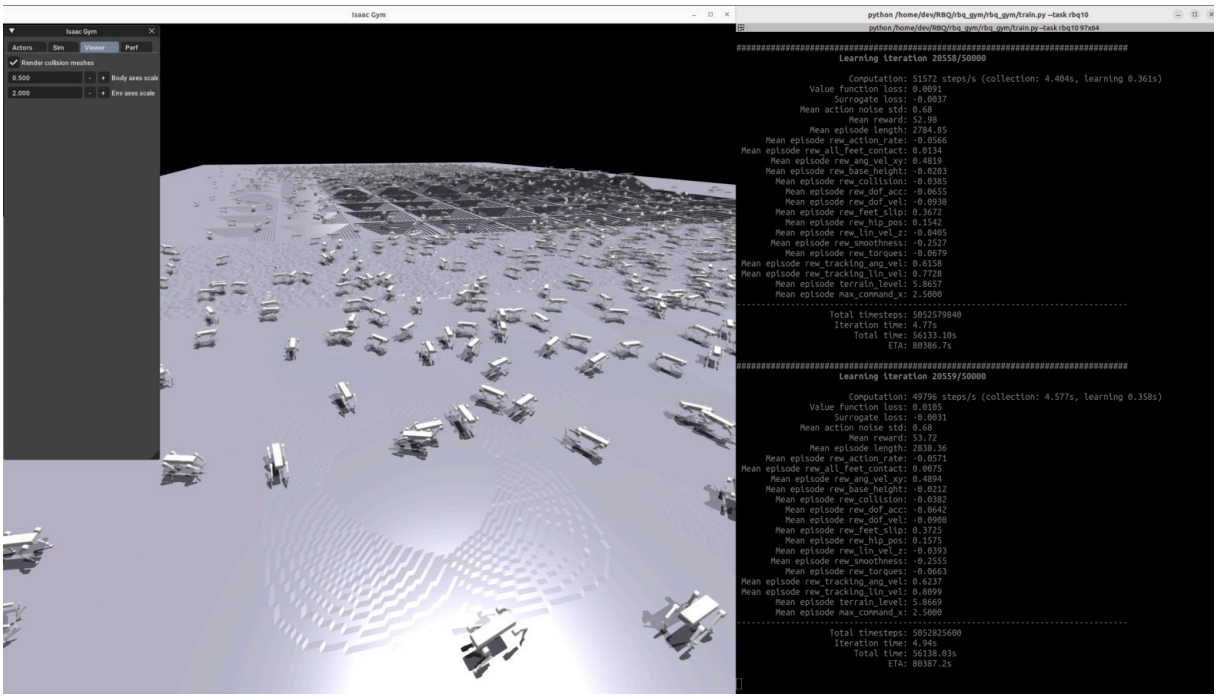
rbq_gym 디렉토리 내에서 다음 명령어를 실행하여 학습을 시작합니다:

```bash
bash scripts/train.bash
```

- 렌더링 없이 실행하려면 `--headless` 를 추가하세요

> **TIP**
>
> 성능 향상을 위해 학습이 시작되면 `v` 키를 눌러 렌더링을 중지할 수 있습니다. 나중에 진행 상황을 확인하기 위해 다시 활성화할 수 있습니다.

Train RBQ10



🏅 Play

rbq_gym 디렉토리 내에서 다음 명령어를 실행하여 학습 결과를 평가합니다:

```bash
bash scripts/play.bash
```

- CPU를 사용하여 실행하려면 다음 인자를 추가하세요: `--sim_device=cpu`

- 기본적으로 실험 폴더의 마지막 실행에서 가장 최근 모델을 로드합니다.
- `load_run`과 `checkpoint`를 설정하여 다른 횟수로 학습된 모델을 선택할 수 있습니다.

Play RBQ10
▶ YouTube에서 열기 (클릭)

🎯 Sim2Sim & Deploy

학습된 정책을 평가하고 실제 로봇에 배포하려면 Simple-RL에서 자세한 내용을 참고하세요.

📂 디렉토리 구조

```
rbq_gym/
├── 3rdparty
├── policy
│   └── rbq10
│       ├── info.json
│       └── policy.onnx
├── rbq_gym
│   ├── envs
│   │   ├── base
│   │   │   ├── base_config.py
│   │   │   ├── base_task.py
│   │   │   ├── rbquad_config.py
│   │   │   └── rbquad_env.py
│   │   ├── __init__.py
│   │   └── rbq10
│   │       ├── rbq10_config.py
│   │       ├── rbq10_env.py
│   │       └── rewards.py
│   ├── __init__.py
│   ├── model_test.py
│   ├── play.py
│   ├── train.py
│   └── utils
│       ├── helpers.py
│       ├── __init__.py
│       ├── keyboard.py
│       ├── logger.py
```

```
    │       ├── math.py
    │       ├── task_registry.py
    │       └── terrain.py
    ├── scripts
    │   ├── activate.bash
    │   ├── clear.bash
    │   ├── configure.bash
    │   ├── play.bash
    │   ├── python.bash
    │   ├── setup.bash
    │   └── train.bash
    ├── dependencies.yaml
    └── setup.py
```

- `setup.bash`, `train.bash`, `play.bash` : 환경 설정, 학습, 실행을 위한 스크립트.
- `envs/base/` : RBQ 로봇을 위한 기본 환경 클래스 및 설정.
- `envs/rbq10/` : RBQ10 환경, 설정, 보상 함수.
- `utils/` : 헬퍼, 키보드 입력, 로깅, 수학 함수, 태스크 레지스트리, 지형 생성 등 유틸리티 모듈.
- `3rdparty/` : 서드파티 의존성 및 라이브러리.
- `policy/` : ONNX 형식의 학습된 정책 및 info 파일.

🔧 새로운 GYM 환경 추가하기

기본 환경 `rbquad_env` 는 거친 지형 보행 태스크를 구현합니다. 새로운 환경을 추가하려면:

1. `envs/` 에 새 폴더를 추가하고 기존 환경 설정을 상속하는 `<your_env>_config.py` 를 만듭니다.
2. 새 로봇을 추가하는 경우:
   - `resources/` 디렉토리에 해당 에셋을 추가합니다.
   - 설정에서 에셋 경로를 설정하고, body 이름, `default_joint_positions`, PD 게인을 정의합니다.
   - 원하는 `train_cfg` 와 환경 클래스 이름을 지정합니다.
   - `train_cfg` 에서 `experiment_name` 과 `run_name` 을 설정합니다.

3. (필요한 경우) `<your_env>_env.py`에서 기존 환경을 상속하여 원하는 함수를 오버라이드하거나 보상 함수를 추가합니다.
4. `rbq_gym/envs/__init__.py`에 환경을 등록합니다.
5. 필요에 따라 다른 매개변수를 수정/조정합니다. 보상을 제거하려면 해당 스케일을 0으로 설정하세요. 다른 환경의 매개변수는 수정하지 마세요.

1.6 RBQ-Lab

📚 소개

RBQ-Lab 사용 가이드입니다

이 문서는 **Isaac Sim / Isaac Lab** 환경에서 **RBQ10** 사족 로봇 보행 policy를 학습하고, 재생(Play) 후 배포에 활용하는 기본 절차를 설명합니다.

💻 시스템 요구사항

RBQ-Lab 사용 시 아래 환경을 권장합니다.

- OS: Ubuntu 22.04 (x86)
- GPU: CUDA 가능한 NVIDIA GPU
- 디스크: 최소 **80 GB**, 권장 **120 GB 이상** (Isaac Sim / Isaac Lab / Conda / 캐시 포함)
- 네트워크: 최초 설치 시 `wget`, `git clone`, `pip` 다운로드 필요

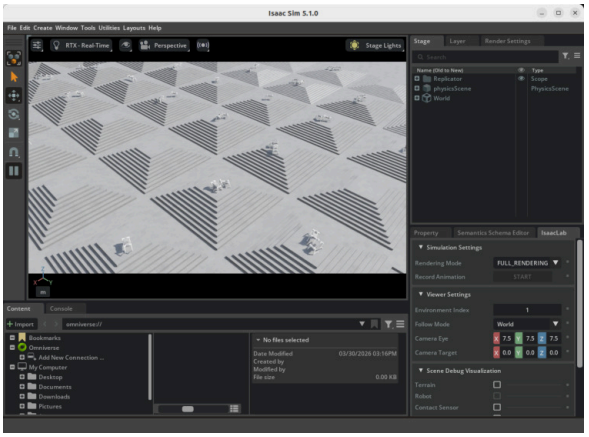
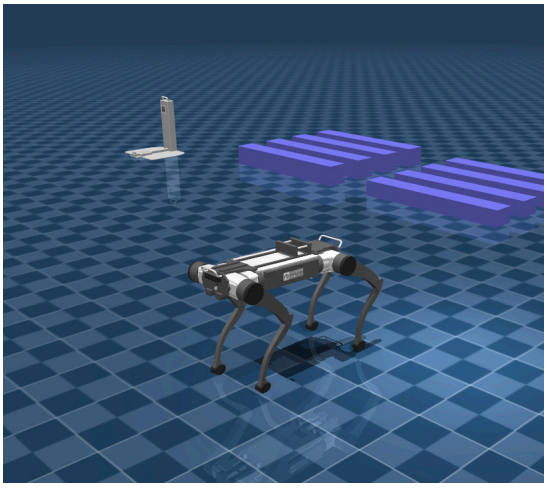
> **버전 확인**
>
> 실제 운영 버전은 `dependencies.yaml` **과** `scripts/setup.bash` **의 핀**을 우선 확인하세요.

🔁 프로세스 개요

강화학습을 통한 모션 제어의 기본 워크플로우는 다음과 같습니다:

Setup → Train → Play → Deploy

- **Setup**: `setup.bash` 로 의존성/환경 설치
- **Train**: 태스크 `rbq10` 으로 학습
- **Play**: 태스크 `rbq10_play` 로 재생 검증
- **Deploy**: 산출 정책(`policy.onnx` , `info.json` 등) 활용

Isaac Sim	Mujoco
	

📦 환경 설정

rbq_lab 디렉토리로 이동합니다:

```bash
cd <your workspace>/RBQ/rbq_lab
```

rbq_lab 디렉토리 내에서 다음 명령어를 실행하여 환경을 설정합니다:

```bash
bash scripts/setup.bash
```

🏋️ Train

기본 학습 실행:

```bash
bash scripts/train.bash
```

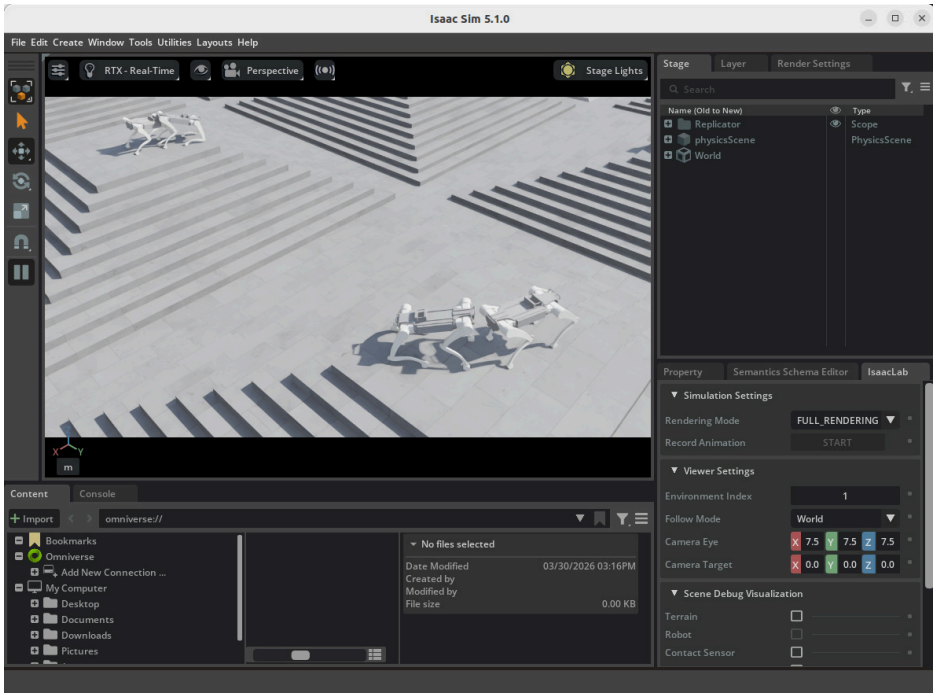
기본 태스크는 `rbq10` 입니다. 학습 로그는 보통 `logs/rbq10/<experiment_name>/...` 에 쌓입니다.

`train.py` 에서 쓸 수 있는 옵션 (참고)

`--task`, `--num_envs`, `--device`, `--headless`, `--resume`, `--checkpoint`, `--max_iterations`, `--seed`, `--video` 등. 전체 목록은 `bash scripts/python.bash rbq_lab/train.py --help` 로 확인합니다.

인자를 고정해서 쓰고 싶다면 **`scripts/train.bash` 안에서 `train.py` 호출 부분을 수정**해 원하는 옵션을 넘기면 됩니다. headless 중심 동작이라 학습 창이 뜨지 않아도 정상입니다.

Train RBQ-Lab



학습 로그 확인: `logs/rbq10/<experiment_name>/...`

🏅 Play

기본 재생 실행:

```bash
bash scripts/play.bash
```

기본 태스크는 **`rbq10_play`** 입니다. 학습 체크포인트(.pt)는 보통 `logs/<experiment_name>/.../model_*.pt` 에 있습니다. play 실행 시에는 원하는 체크

포인트를 `--checkpoint`로 직접 지정하는 것이 가장 확실합니다.

`play.py`에서 쓸 수 있는 옵션 (참고)

`--use_pretrained_checkpoint`, `--no_keyboard`, `--real-time`, `--video`, `--checkpoint` 등. 전체 목록은 `bash scripts/python.bash rbq_lab/play.py --help`로 확인합니다.

인자를 고정해서 쓰고 싶다면 **`scripts/play.bash` 안에서 `play.py` 호출 부분을 수정**해 원하는 옵션을 넘기면 됩니다.

Play RBQ-Lab
▶ YouTube에서 열기 (클릭)

🎯 Deploy

학습된 정책을 평가하고 실제 로봇에 배포하려면 Simple-RL에서 자세한 내용을 참고하세요.

Play 종료 후 `logs/<experiment_name>/exported/` 아래에 `policy.jit`, `policy.onnx`, `info.json`이 생성됩니다.

📂 디렉토리 구조

```
rbq_lab/
├── 3rdparty
├── policy
│   └── rbq10
│       ├── info.json
│       └── policy.onnx
├── rbq_lab
│   ├── envs
│   │   ├── base
│   │   │   ├── base_task.py
│   │   │   └── base_task_cfg.py
│   │   ├── __init__.py
│   │   └── rbq10
```

```
│   │       ├── __init__.py
│   │       ├── env.py
│   │       ├── env_cfg.py
│   │       ├── env_mdp.py
│   │       ├── rbq10.py
│   │       └── rsl_rl_ppo_cfg.py
│   ├── __init__.py
│   ├── play.py
│   ├── train.py
│   └── utils
│       ├── camera.py
│       ├── cli_args.py
│       ├── keyboard.py
│       ├── marker.py
│       ├── math.py
│       └── rough.py
├── scripts
│   ├── activate.bash
│   ├── clear.bash
│   ├── configure.bash
│   ├── isaacsim.bash
│   ├── play.bash
│   ├── python.bash
│   ├── setup.bash
│   └── train.bash
├── dependencies.yaml
└── setup.py
```

- USD 등 에셋은 별도 `resources/` 등에 두고, 코드·`LAB_ASSET_DIR` 설정에 따릅니다.
- 학습 로그: `logs/rsl_rl/<experiment_name>/...`
- 재생·체크포인트 탐색: `logs/<experiment_name>/...`
- 정리: `bash scripts/clear.bash` (`*.egg-info`, `__pycache__`, 필요 시 `logs/` 등)

🔧 새로운 Lab 환경 추가하기

기본 흐름은 기존 `rbq10` 환경을 복사·수정해 새 태스크를 등록하는 방식입니다.

1. `rbq_lab/envs/` 아래에 새 환경 폴더를 추가합니다.
2. 새 폴더에 `env.py`, `env_cfg.py`, `env_mdp.py`, `rsl_rl_ppo_cfg.py`를 구성합니다.

3. 필요 시 `resources/`에 로봇/환경 에셋(USD 등)을 추가하고, 환경 설정에서 경로를 연결합니다.
4. `rbq_lab/envs/__init__.py`에 새 환경(태스크 ID)을 등록합니다.
5. `train.py`에서 새 태스크 ID가 정상 선택되는지 확인 후 학습·재생으로 검증합니다.

태스크 추가 후에는 작은 `num_envs`로 먼저 실행해 초기 오류(에셋 경로, 관측/행동 차원, 보상 정의)를 확인하는 것을 권장합니다.

2.1 🚀 Simple-motion

🛠️ Simple-motion 코드 빌드

- 다음 명령어를 실행하여 프로젝트를 빌드하세요:

```bash
bash scripts/ex/build.bash
```

  - *인자 설명:*
    - `--help` : 도움말 메시지를 표시하고 종료합니다.
    - `--no-cache` : 캐시를 사용하지 않고 새로 빌드합니다.

🧪 테스트

- `simulator` 스크립트 실행

```bash
bash scripts/sim.bash
```

- `Simple-motion` 프로세스 실행

```bash
sudo ./examples/bin/Simple-motion
```

✅ 로봇 동작	앉기	서기
⌨️ 키보드 키	x	z

▶ YouTube에서 열기 (클릭)

🚀 실제 로봇에 배포 및 실행

- PC를 로봇의 Wi-Fi에 연결하세요.

  📝 *Wi-Fi SSID:* `RBQ_xxxx` *(로봇의 식별자에 따라* `xxxx` *부분이 다르니 참고해 주세요.)*

- 빌드된 실행 파일을 로봇 PC로 복사합니다.

```bash
scp examples/bin/Simple-motion rbq@192.168.0.10:~/rbq_ws/examples/bin/.
```

- 로봇 PC에 SSH로 접속합니다.

```bash
ssh rbq@192.168.0.10
```

- 로봇 PC에서 프로세스를 실행

```bash
cd ~/rbq_ws && sudo ./examples/bin/Simple-motion
```

✅ 로봇 동작	앉기	서기
⌨️ 키보드 키	x	z

⚠️ 소프트웨어 버전 호환성

로봇의 온보드 소프트웨어 버전이 **개발 PC에 설치된 소프트웨어 버전**과 일치하는지 확인하세요.

- 배포 전에 두 환경 모두 최신 공식 릴리스로 업데이트하는 것을 권장합니다. **공식 RBQ 소프트웨어 저장소**에서 최신 릴리스를 다운로드할 수 있습니다.

2.2 🚀 Simple-command

🛠️ Simple-command 코드 빌드

- 다음 명령어를 실행하여 프로젝트를 빌드하세요:

```bash
bash scripts/ex/build.bash
```

  - *인자 설명:*
    - `--help` : 도움말 메시지를 표시하고 종료합니다.
    - `--no-cache` : 캐시를 사용하지 않고 새로 빌드합니다.

🧪 테스트

- `simulator` 스크립트 실행

```bash
bash scripts/sim.bash
```

- `Simple-command` 프로세스 실행

```bash
sudo ./examples/bin/Simple-command
```

✅ 버튼 동작	앉기	서기	걷기	계단	달리기
⌨️ 키보드 키	1	2	3	4	5

▶ YouTube에서 열기 (클릭)

🚀 실제 로봇에 배포 및 실행

- PC를 로봇의 Wi-Fi에 연결하세요.

  📝 *Wi-Fi SSID:* `RBQ_xxxx` *(로봇의 식별자에 따라* `xxxx` *부분이 다르니 참고해 주세요.)*

- 빌드된 실행 파일을 로봇 PC로 복사합니다.

```bash
scp examples/bin/Simple-command rbq@192.168.0.10:~/rbq_ws/examples/bin/.
```

- 로봇 PC에 SSH로 접속합니다.

```bash
ssh rbq@192.168.0.10
```

- 로봇 PC에서 프로세스를 실행

```bash
cd ~/rbq_ws && sudo ./examples/bin/Simple-command
```

✅ 버튼 동작	앉기	서기	걷기	계단	달리기
⌨️ 키보드 키	1	2	3	4	5

⚠️ 소프트웨어 버전 호환성

로봇의 온보드 소프트웨어 버전이 **개발 PC에 설치된 소프트웨어 버전**과 일치하는지 확인하세요.

- 배포 전에 두 환경 모두 최신 공식 릴리스로 업데이트하는 것을 권장합니다. **공식 RBQ 소프트웨어 저장소**에서 최신 릴리스를 다운로드할 수 있습니다.

2.3 🚀 Simple-RL

🛠️ 예제 빌드

- 다음 명령어를 실행하여 프로젝트를 빌드합니다:

```bash
bash scripts/ex/docker/run.bash --no-cache
```

  - *인자:*
    - `--help` : 도움말 메시지를 표시하고 종료합니다.
    - `--no-cache` : 재빌드합니다.

🧪 테스트

- `simulator` 스크립트를 실행합니다

```bash
bash scripts/sim.bash
```

⚠️ Simple-RL을 실행하기 전에 반드시 로봇을 Auto Start 해주세요

- `Simple-RL` 프로세스를 실행합니다

```bash
sudo ./examples/bin/Simple-RL
```

✅ 로봇 동작	앉기	서기	제어 모드	명령
키보드 키	Z	X	C	W - 🔼 전진 S - 🔽 후진 A - ◀️ 좌측이동 D - ▶️ 우측이동 Q - 🔄 좌측 Yaw E - 🔃 우측 Yaw

RAINBOW ROBOTICS

▶ YouTube에서 열기 (클릭)

🚀 실제 로봇에 배포 및 실행

- PC를 로봇의 Wi-Fi에 연결합니다.

  📝 *Wi-Fi SSID:* `RBQ_xxxx` *(*`xxxx`*는 로봇 Wi-Fi의 고유 식별자로 교체하세요).*

- Robot PC에 연결합니다

```bash
ssh rbq@192.168.0.10
```

- 파일 디렉토리를 생성합니다

```bash
mkdir -p /home/rbq/rbq_ws/rbq_gym/policy/rbq10
mkdir -p /home/rbq/rbq_ws/examples/bin
```

- 개발 PC로 돌아와서 바이너리, 정책 파일을 로봇 PC로 복사합니다

```bash
scp -r examples/bin/ rbq@192.168.0.10:~/rbq_ws/examples/
scp -r rbq_gym/policy/rbq10/ rbq@192.168.0.10:~/rbq_ws/rbq_gym/policy/
```

- 로봇 PC에 SSH로 접속합니다

```bash
ssh rbq@192.168.0.10
```

⚠️ Simple-RL을 실행하기 전에 반드시 로봇을 Auto Start 해주세요

- 로봇 PC에서 프로세스를 실행합니다

```bash
sudo ~/rbq_ws/examples/bin/Simple-RL
```

✅ 로봇 동작	앉기	서기	제어 모드	명령

키보드 키	Z	X	C	W - 🔼 전진 S - 🔽 후진 A - ◀️ 좌측 이동 D - ▶️ 우측 이동 Q - 🔄 좌측 Yaw E - 🔃 우측 Yaw

---

⚠️ 소프트웨어 버전 호환성

로봇의 온보드 소프트웨어 버전이 **개발 PC에 설치된 소프트웨어 버전**과 일치하는지 확인하세요.

- ✅ 배포 전에 두 환경을 모두 최신 공식 릴리즈로 업데이트하는 것을 권장합니다. **공식 RBQ 소프트웨어 저장소**에서 최신 릴리즈를 다운로드할 수 있습니다.

3.1 소프트웨어 업데이트 방법

🚀 1단계. 최신 릴리스 다운로드

공식 RBQ GitHub 릴리스 페이지에서 최신 소스 코드를 다운로드하세요:

🔗 https://github.com/RainbowRobotics/RBQ/releases

> 📝 로봇 플랫폼에 맞는 올바른 버전을 선택했는지 반드시 확인하세요.

📦 2단계. 애플리케이션(바이너리)을 로봇에 배포

`소스 코드` 파일을 다운로드한 후 아래 단계를 따라 로봇에 배포합니다.:

1. **개발 PC를 로봇과 같은 네트워크에 연결하고 SSH 접근을 활성화 합니다.**
2. 개발 PC에서 다음 명령어를 실행하세요:

```bash
bash scripts/deploy.bash
```

SSH 비밀번호를 입력하라는 메시지가 나타납니다.

- **비밀번호:** 담당자에게 SSH 비밀번호를 문의하세요.
- ⚠️ 중요: `deploy.bash`가 완료된 후, 새로 배포된 바이너리로 실행하기 위해 로봇을 반드시 **재시작**하세요 (전원을 끄고 다시 켜면 됩니다).

📋 추가 업데이트 정보

사전 요구사항

- 로봇에 대한 SSH 접근 권한이 있는지 확인하세요
- 개발 환경이 올바르게 설정되어 있는지 확인하세요
- PC와 로봇 간의 네트워크 연결을 확인하세요

업데이트 프로세스 단계

1. **바이너리 준비**: 배포할 애플리케이션이 모두 정상적으로 컴파일되어 있는지 확인합니다
2. **배포 스크립트 실행**: 개발 PC에서 `deploy.bash` 스크립트를 실행하세요
3. **자격 증명 제공**: 메시지가 나타나면 SSH 비밀번호를 입력하세요
4. **로봇 재시작**: 새로운 바이너리를 로드하기 위해 로봇의 전원을 껐다 켜세요

문제 해결

- 배포가 실패하면 네트워크 연결을 확인하세요
- 올바른 SSH 자격 증명이 있는지 확인하세요
- 로봇이 접근 가능하고 전원이 켜져 있는지 확인하세요

📜 히스토리

- 2025.07.18 : "RCL : Robot Control Library" 첫 번째 릴리스
- 2025.07.29 : "RCL" 상태 추정기 추가

3.2 🖥️ RBQ Development PC 연결 방법

이 가이드는 원격 데스크톱 접속을 위해 **NoMachine**을 사용하여 **RBQ Dev PC**에 연결하는 방법을 설명합니다.

🚀 1단계. NoMachine 다운로드

1. 공식 웹사이트에서 NoMachine 애플리케이션을 다운로드합니다: 🔗 **NoMachine 다운로드 페이지**
2. 플랫폼(Linux)에 맞는 올바른 버전을 선택하세요.

📦 2단계. RBQ Dev PC에 NoMachine 설치

터미널을 열고 다음 명령어를 실행하여 NoMachine을 설치합니다:

```bash
cd /usr
sudo tar zxvf ~/Downloads/nomachine_latest_version.tar.gz
sudo /usr/NX/nxserver --install
sudo /usr/NX/bin/nxserver --status
```

> ✅ `nxserver --status`를 사용하여 서버가 정상적으로 실행 중인지 확인하세요.

🔌 3단계. 개발 PC에서 연결

1. **개발 PC가 RBQ와 동일한 네트워크에 연결**되어 있는지 확인하세요 (이더넷 또는 SSH 연결 가능).
2. PC에서 **NoMachine 앱**을 실행합니다.
3. **"Add"**를 클릭하여 새 연결을 생성합니다.

4. 다음 정보를 입력합니다:

```
Name : rbq
Host : 192.168.0.10
```

> 💡 권장 NoMachine 디스플레이 해상도 설정:
>
> - ✅ 활성화: **"Scale the remote desktop to fit the window"**
> - 🚫 비활성화: **"Resize the remote display"**

✅ 연결 완료!

이제 사용자 친화적인 데스크톱 인터페이스를 통해 **RBQ Dev PC**를 원격으로 제어하고 개발할 수 있습니다.

개발 워크플로우를 즐기세요!

4.1 문의하기

사용자가 경험하는 다양한 런타임 오류 및 문제를 지속적으로 수집하고 해결하고 있습니다. 문제가 발생하면 아래 채널을 통해 연락해 주시기 바랍니다:

- **GitHub Issues**: 기술적인 문제나 버그가 발생하면 GitHub Issues 페이지에 등록해 주세요. 문제를 효율적으로 추적하고 해결하는 데 도움이 됩니다. **GitHub Issue 제출하기**
- **GitHub Discussions**: 일반적인 질문, 피드백, 또는 다른 사용자 및 개발자와 런타임 오류에 대해 논의하려면 GitHub Discussions 포럼에 참여해 주세요. **GitHub Discussions 참여하기**
- **이메일**: 비공개적이거나 민감한 문의가 있으시면 이메일로 직접 연락해 주세요. rbq.support@rainbow-robotics.com