

PiPER Start Manual



WeGo

About PiPER

PiPER



Support

ROS2

ROS

python™

WeGo

1. Windows OS용 소프트웨어: ArmRobotUA

- 해당 프로그램을 통해 PiPER의 주요 기능을 대략적으로 살펴보고, 시범적으로 활용해보실 수 있습니다.

2. Python SDK

- Python으로 개발하고 통합할 수 있도록 지원하는 개발 도구입니다.

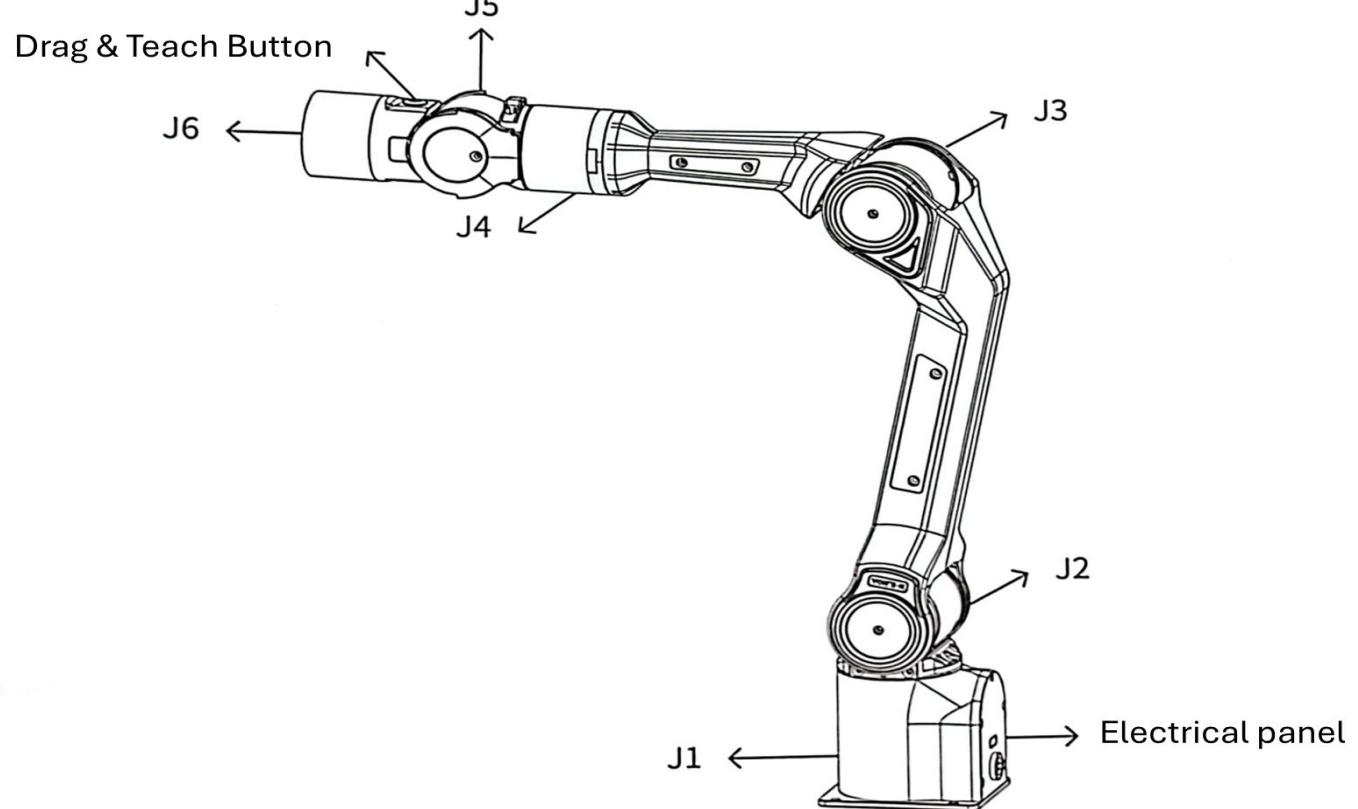
3. ROS1

- Ubuntu 20.04 이하 버전에서 사용 가능한 ROS로, Noetic 버전을 권장합니다.

4. ROS2

- Ubuntu 20.04 이상 버전에서 사용 가능한 ROS로, Humble 버전을 권장합니다.

About PiPER



파라미터 유형	항목	사양
구조적 사양	자유도	6
	유효 하중	1.5KG
	중량	4.2KG
	반복 정밀도	$\pm 0.1\text{mm}$
	작업 반경	626.75mm
	정격 전압	DC24V (최소: 24V, 최대: 26V)
	전력 소비	최대 전력 $\leq 120\text{W}$ 종합 소비 전력 $\leq 40\text{W}$
	재질	알루미늄 합금 프레임 플라스틱 외부
	컨트롤러	통합형
	통신 방식	CAN
	제어 방식	핸들링(끌어서 학습) / 오프라인 궤적 / API / 호스트 컴퓨터
	외부 인터페이스	전원 인터페이스 x1, CAN 인터페이스 x1
	베이스 설치 크기	70mm x 70mm x M5 x 4
	작동 환경	온도: -20 to 50°C 습도: 25%~85%
	소음	< 60dB
	관절 동작 범위	J1: $\pm 154^\circ$, J2: $0^\circ \sim 195^\circ$, J3: $-175^\circ \sim 0^\circ$ J4: $-106^\circ \sim 106^\circ$, J5: $-75^\circ \sim 75^\circ$, J6: $\pm 100^\circ$
	관절 최대 속도	J1: $180^\circ/\text{s}$, J2: $195^\circ/\text{s}$, J3: $180^\circ/\text{s}$ J4: $225^\circ/\text{s}$, J5: $225^\circ/\text{s}$, J6: $225^\circ/\text{s}$

주의: 관절 동작 범위 이상의 각도로 가동하면 모터가 e-stop모드로 들어가 갑작스럽게 멈출 수 있고, 설정된 원점이 어긋날 수 있으니 동작 범위 내에서 사용해 주시기 바랍니다.

About PiPER

PiPER 구성물품

①



②



③



④



⑤



⑥



No.	물품	개수
1	CAN to USB 모듈	1
2	전원 어댑터	1
3	마이크로 USB 케이블	1
4	전력 및 통신 케이블	1
5	베이스 육각 볼트	4
6	육각 렌치	1

About PiPER

1. 전기적 패널 구성



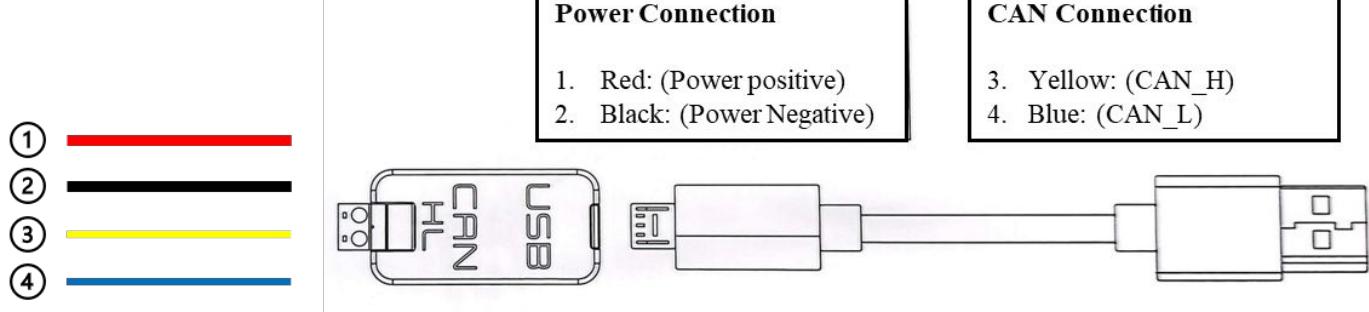
1. Power and communication port
2. Status indicator light
3. J1 and J2 connection port

1. CAN 통신 연결부



1. CAN-L
2. Power Negative
3. CAN-H
4. Power positive

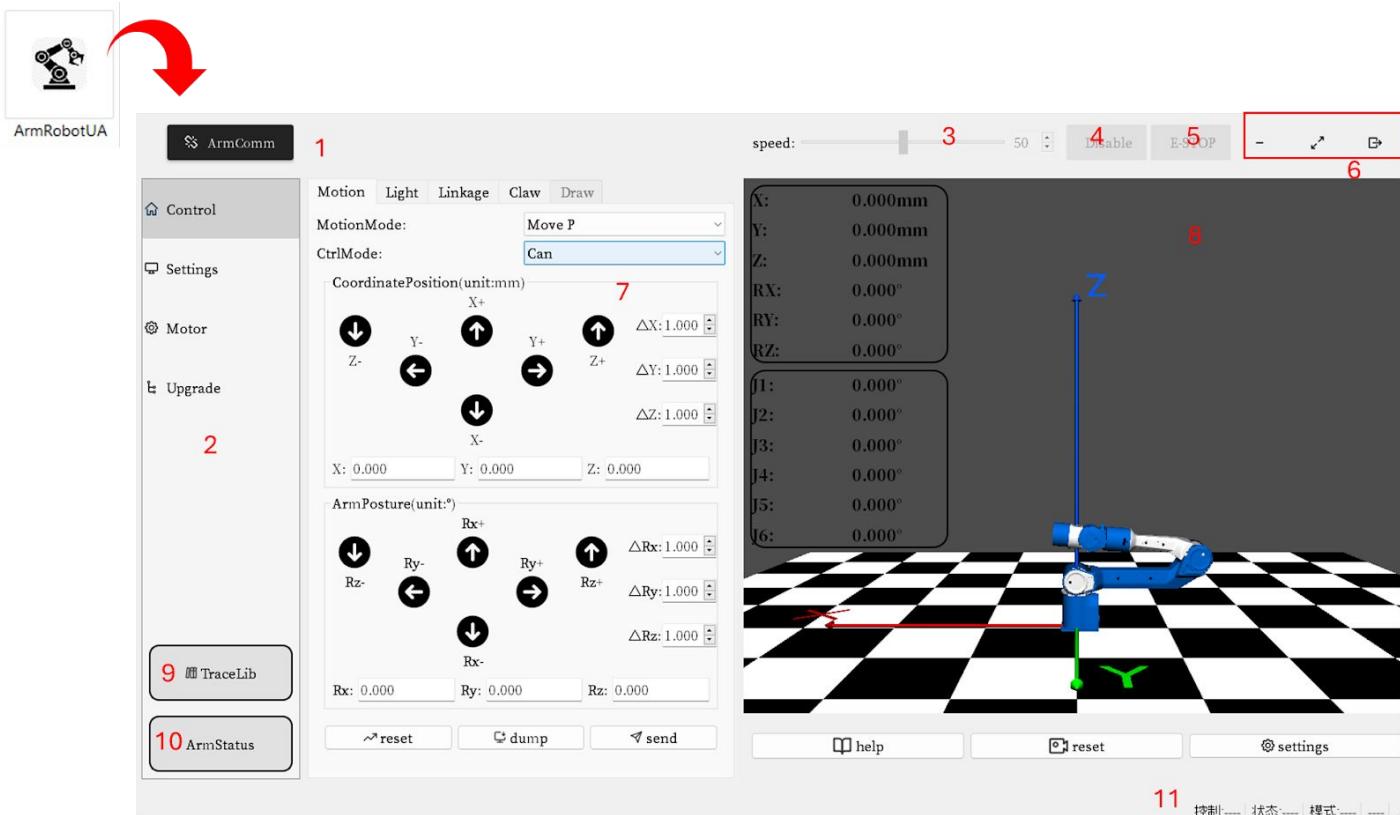
1. 전원 및 CAN 통신 연결 방법



ArmRobotUA

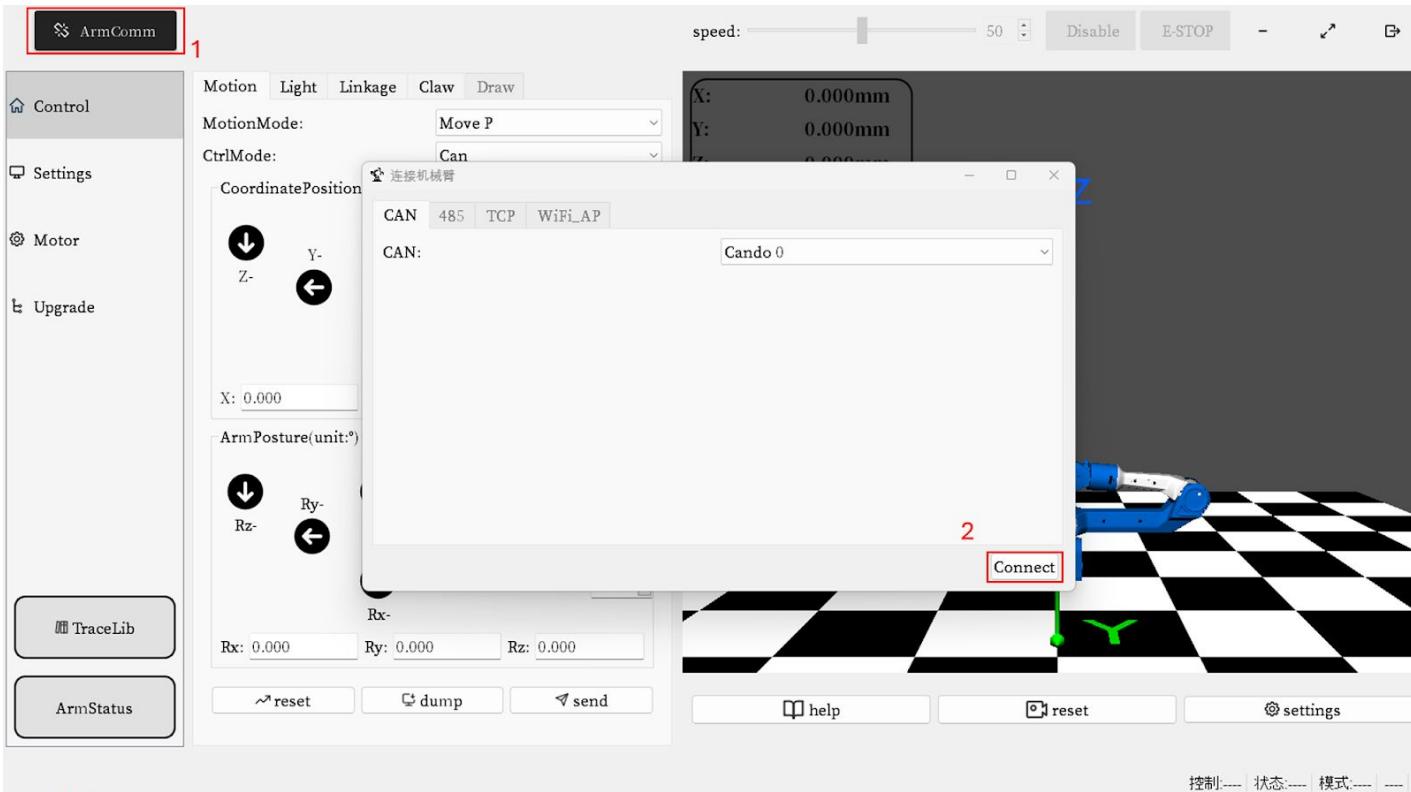
01

1. Windows OS용 프로그램(ArmRobotUA) 세부구성

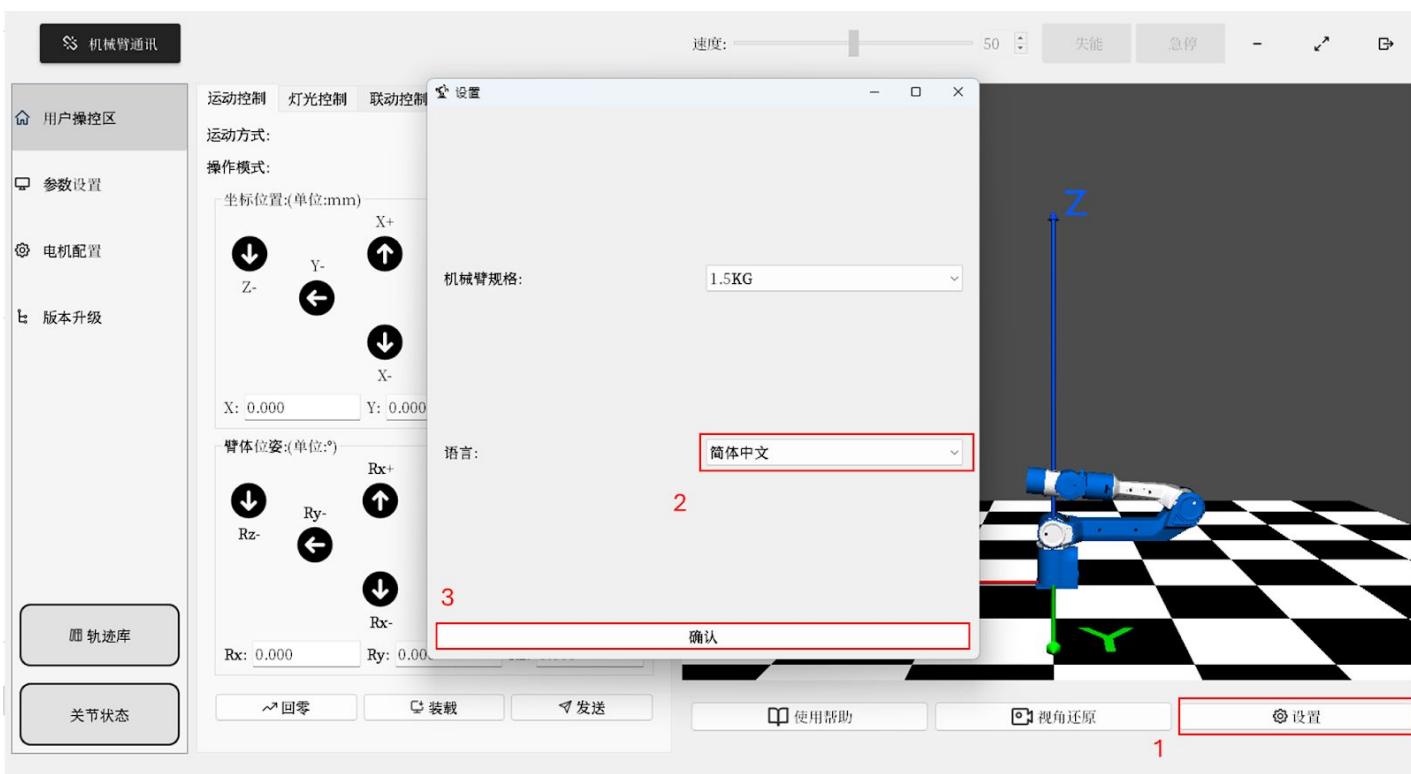


No.	항목
1	유/무선 통신 활성화 버튼
2	메뉴 사이드바
3	속도 비율(0-100) 설정
4	활성화 / 비활성화 버튼
5	비상 정지 / 복구 버튼
6	창 크기 조정 및 닫기 버튼
7	PiPER 구동 모드 및 함수
8	3D 시뮬레이션 모델
9	궤적 라이브러리 관리창
10	PiPER 세부 상태창
11	PiPER 기본 상태창

2. 좌측 상단 메뉴에서 CAN 통신을 활성화합니다. 'ArmComm'을 클릭한 후 'Connect' 버튼을 바로 누르시면 됩니다.

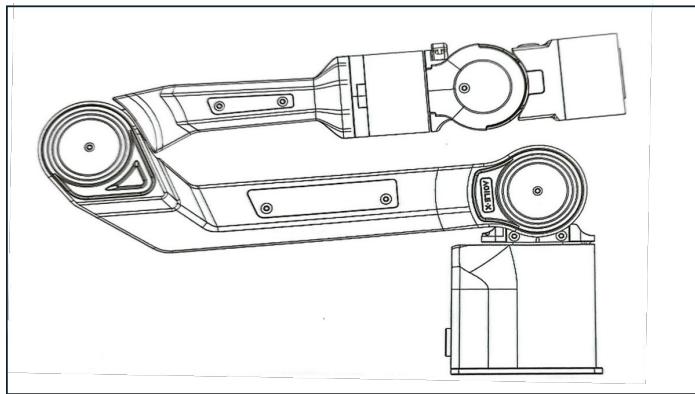


3. 언어 설정에서 현재 언어를 중국어에서 영어로 변경합니다.



01. ArmRobotUA

4. 현재 로봇팔을 원점에 가깝게 위치시킵니다.



- 좌측 사진처럼 로봇팔이 중력에 의해 넘어지지 않도록 위치시킵니다.
- 로봇팔에 전원 인가 후 처음 사용 시 우측 사진처럼 이동합니다. 해당 위치가 원점입니다.

5. 시작에 앞서 'E-STOP'을 누른 후, 다시 'Restore'를 눌러줍니다.

The screenshot shows the software interface for controlling the robotic arm. On the left, there are tabs for Control, Settings, Motor, and Upgrade. A green button labeled 'Connected' is at the top left. At the top right, there are buttons for 'Disable' (red border) and 'Restore'. Below these are two sets of coordinate input fields: 'CoordinatePosition(unit:mm)' and 'ArmPosture(unit:°)'. The 'CoordinatePosition' section includes buttons for X+, Y+, Z+, X-, Y-, Z-, and a 'Move P' dropdown. The 'ArmPosture' section includes buttons for Rx+, Ry+, Rz+, Rx-, Ry-, Rz-, and a 'Move P' dropdown. To the right, a 3D simulation window shows the blue robotic arm on a black-and-white checkered floor. A coordinate system (X red, Y green, Z blue) is overlaid on the arm. A data panel on the right lists current values:

X:	52.156mm
Y:	1.844mm
Z:	171.244mm
RX:	173.702°
RY:	78.662°
RZ:	174.792°
J1:	0.071°
J2:	-1.921°
J3:	1.205°
J4:	3.831°
J5:	17.021°
J6:	-2.627°

- E-STOP을 누르면 로봇팔이 그 자리에서 아래로 천천히 내려갑니다. 긴급한 상황에 사용해주시면 됩니다.

01. ArmRobotUA

6. 현재 로봇팔 그리퍼 부착 여부를 확인합니다.



a) Two Finger gripper



b) Teaching / Leader gripper

- 그리퍼가 부착되지 않는다면 a) Two Finger Gripper로 간주하면 됩니다.
- b) Leader Grippe는 Master로 구성해 Slave에게 모션을 학습시킬 때 사용됩니다.
- 하지만, 그리퍼 종류에 관계 없이 **해당 소프트웨어로 제어하려면 6번처럼 Input 모드로 변경**해야 합니다.

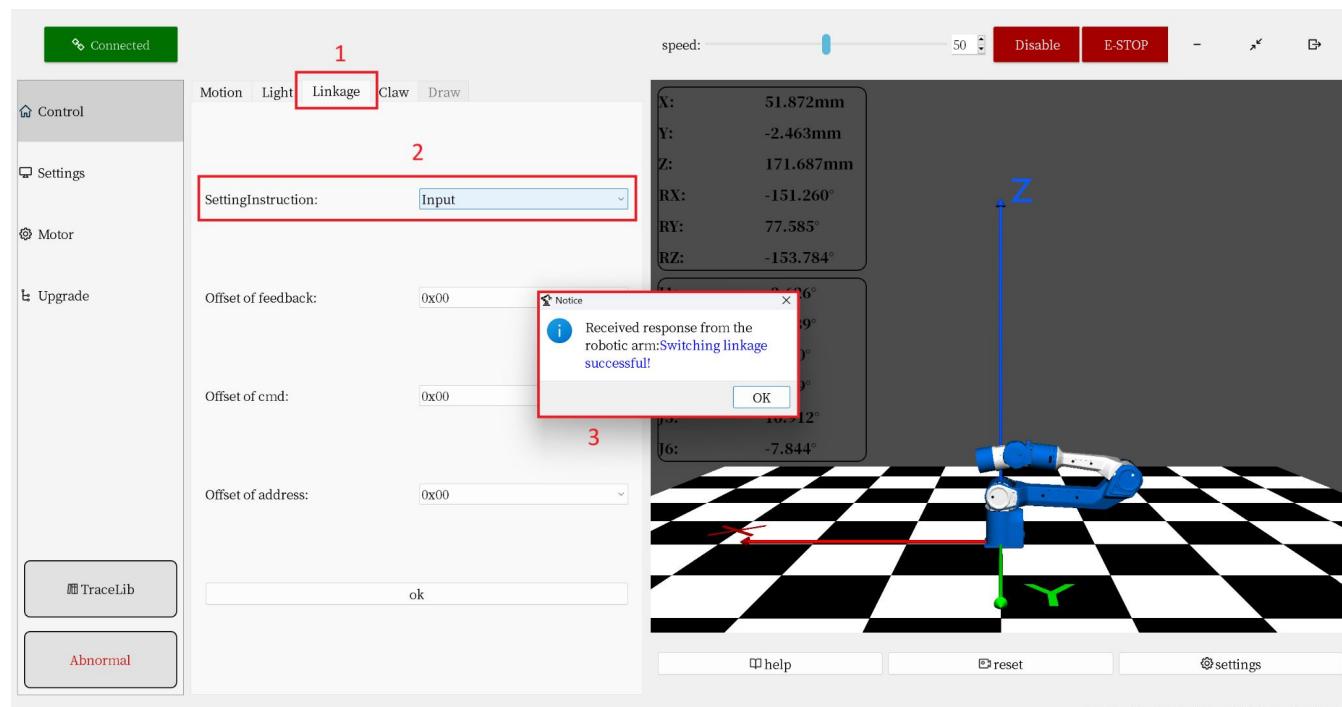
7. 그리퍼의 위치가 올바른지 확인합니다.



- 그리퍼의 카메라 부착 부분이 위를 향하도록 위치시켜야 합니다.
- 그렇지 않으면 **그리퍼 케이블이 끊어질 수 있습니다.**
- 이는 현재 위치에서 가장 가까운 방향으로 원점으로 이동하기 때문입니다.
- 따라서, 매번 'Reset'을 누르시기 전에는 그리퍼의 위치를 확인해주세요.

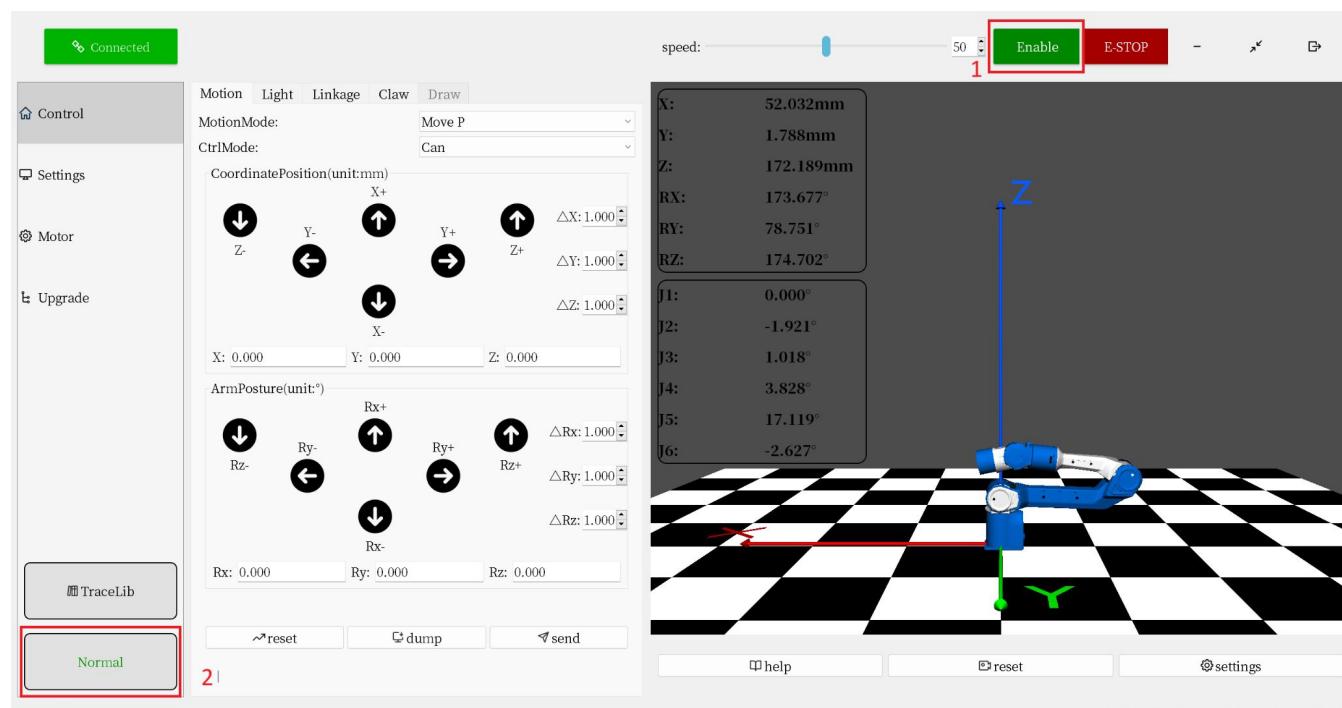
01. ArmRobotUA

8. 해당 프로그램 사용에 적합한 모드로 변경합니다.



- 먼저 상단 바에서 **Linkage**로 이동합니다.
- 해당 프로그램으로 제어하려면, SettingInstruction에서 Output으로 설정합니다.**
- Input은 두 개 이상의 로봇팔을 Master & Slave로 구성할 때만 사용됩니다.

9. 'Enable'을 눌러 활성화시켜줍니다.



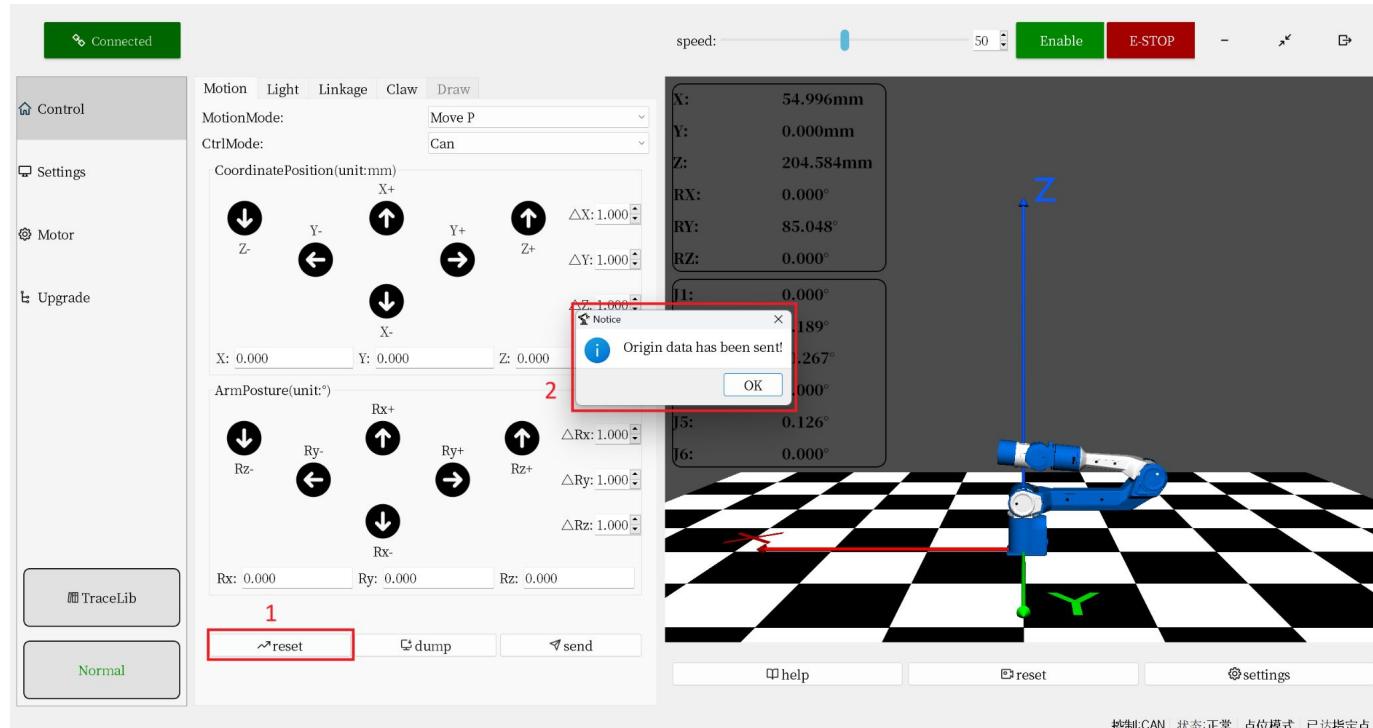
- 'Enable/Disable'은 전력 공급을 제어합니다. 'Disable'시 로봇팔이 수직으로 떨어지므로 주의해주세요.

10. 좌측 하단의 'Normal'로 조인트별 상세 정보를 확인할 수 있습니다.

关节参数窗口						
	J1	J2	J3	J4	J5	J6
StatueCode:	0x40	0x40	0x40	0x40	0x40	0x40
motorSpeed(rpm):	-3	-3	3	17	51	17
Degree(°):	3.192	-0.783	0.951	3.723	8.893	-11.541
Voltage(V):	23	23	23	23	23	23
Current(mA):	98	498	601	21	1102	57
motorTemperature(°C):	26	24	27	25	28	25
driverTemperature(°C):	35	31	32	34	33	33
collision:	normal	normal	normal	normal	normal	normal
blockage:	normal	normal	normal	normal	normal	normal
errorState:	normal	normal	normal	normal	normal	normal
EnableState:	Enable	Enable	Enable	Enable	Enable	Enable
Comm:	normal	normal	normal	normal	normal	normal
AngleState:	normal	normal	normal	normal	normal	normal

- 더 자세히로는 각 모터의 속도(RPM), 각도, 전압, 전류값, 모터 온도 등을 확인할 수 있습니다.

11. 'Reset'으로 로봇팔을 원점으로 위치시킵니다.



- 'Reset' 버튼을 누르기 전, 모드 변경은 잘 되었는지 그리퍼 위치는 올바른지 재확인합니다.

01. ArmRobotUA

12. ‘MotionMode’에서 4개의 모션 명령어로 로봇팔을 제어합니다.

① MoveL: 직선 이동

- 직선 경로로 이동하며, 물체를 집거나 정밀하고 안정적인 작업이 필요할 때 사용됩니다.

② MoveJ: 조인트 기반 이동

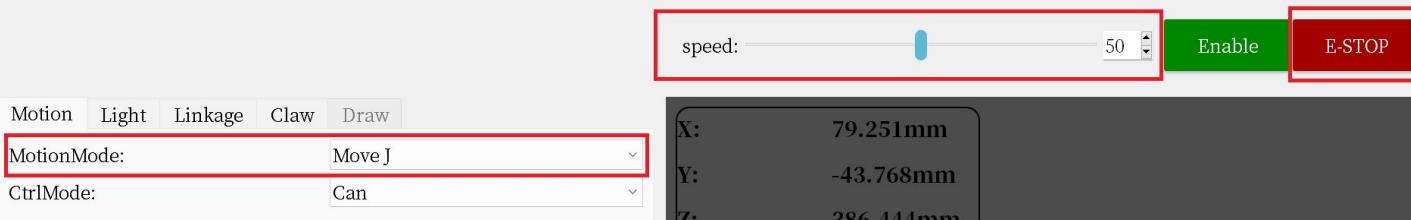
- 각 관절(조인트)을 제어해 목표 위치로 이동할 때 사용됩니다.

③ MoveP: 포인트 기반 이동

- 가장 빠른 경로로 목표 위치에 도달하는 모션 명령어로, 일반적으로 가장 많이 사용됩니다.

④ MoveC: 원호(곡선) 이동

- 곡선을 따라 부드럽게 이동할 때 사용됩니다. 시작점, 경유점, 도착점 총 3개의 위치값이 필요합니다.



Motion 상단 탭의 ‘MotionMode’에서 위의 4가지 모션 명령어를 설정할 수 있습니다. 또한, 속도도 비율(%)로 제어가 가능합니다. 처음에는 속도 20을 추천드립니다. 그리고 위급상황을 대비하기 위해 마우스를 ‘E-STOP’ 근처에 위치시킵니다.

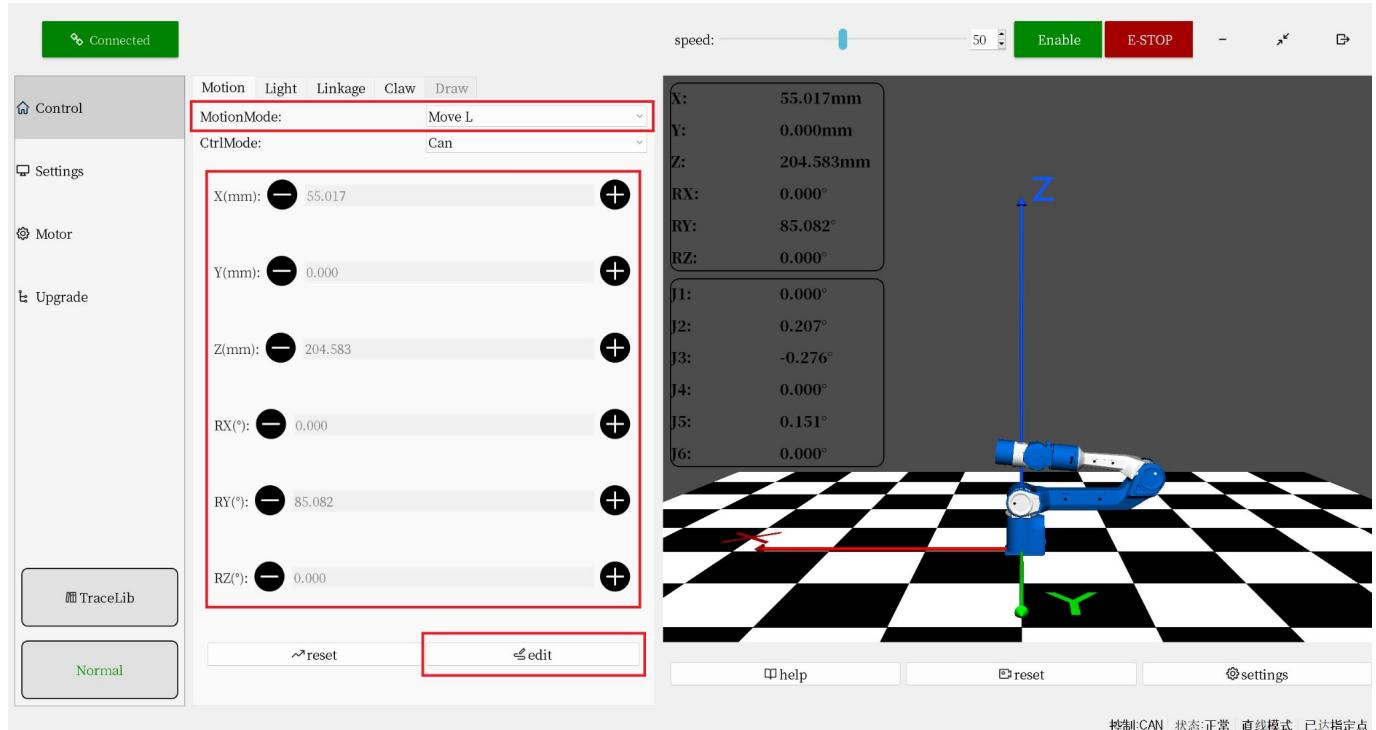
해당 프로그램으로 제어할 때 **CtrlMode는 반드시 CAN 모드로 설정해야 합니다.**

만약 Drag and Teach(6축에 위치한 LED 버튼을 통해 수동으로 로봇팔 조종)를 실행했다면, 다시 로봇팔을 3번 과정처럼 위치시킨 뒤 전원선을 뽑았다가 다시 연결하고 이전 과정을 진행해 주세요.

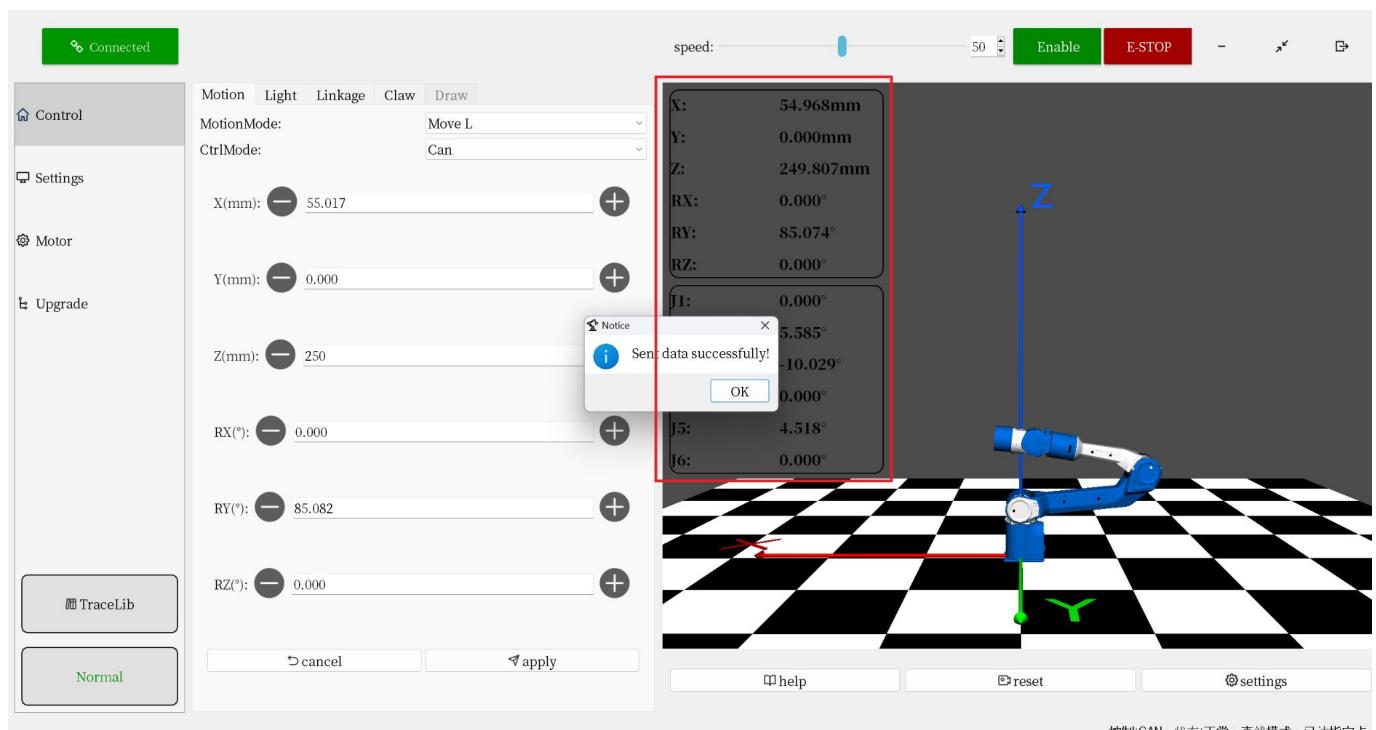
이는 실행 시 Standby 모드로 전환되기 때문이며, 이 상태에서 ArmRobotUA 프로그램을 사용하면 로봇팔이 임의의 위치로 틀 수 있습니다. CtrlMode 변환은 아직 개선 중입니다.

예시 1: MoveL (직선 운동)

- ① 먼저 Z축을 기준으로 +를 눌러봅니다. 혹은, ‘Edit’을 통해 Z값을 250으로 설정 후 ‘Apply’ 버튼을 눌러줍니다.

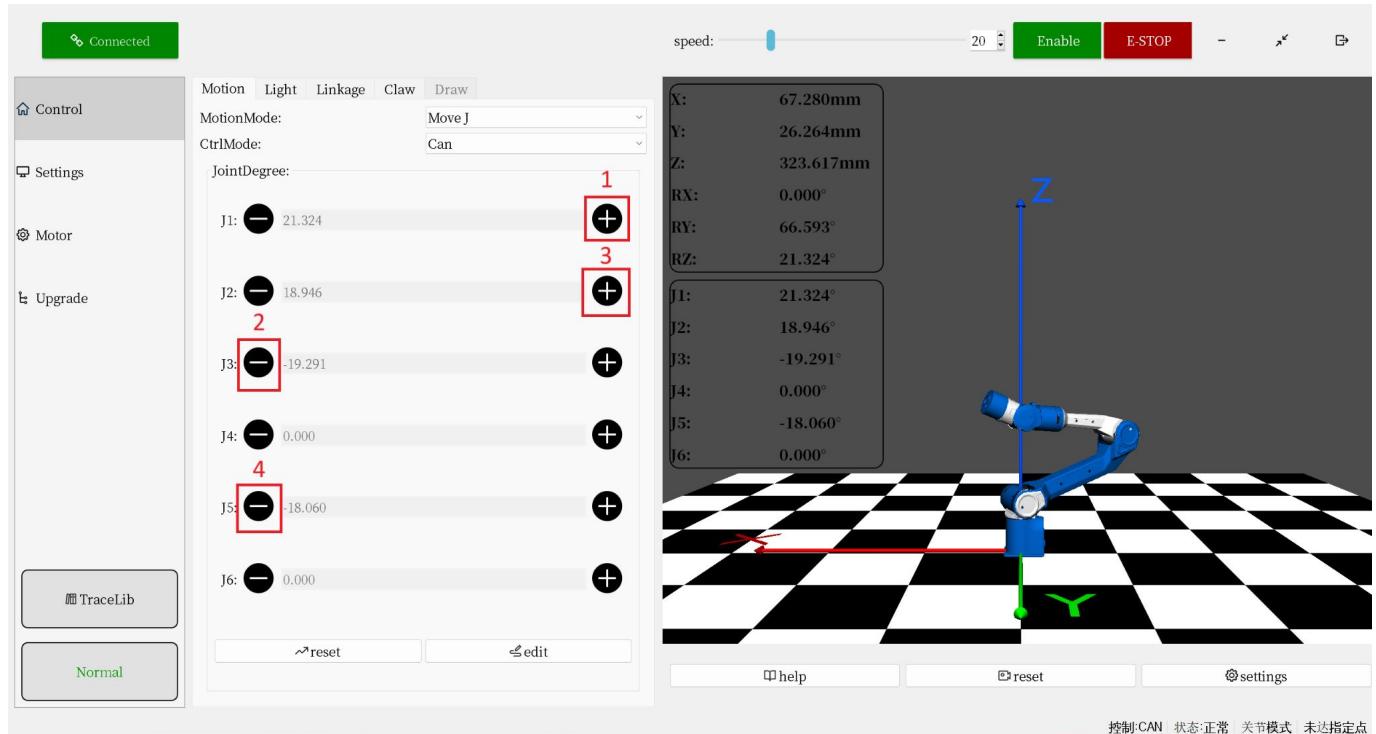


- ② 해당 위치로 잘 이동했는지를 우측 시뮬레이션 창에서 확인하실 수 있습니다.

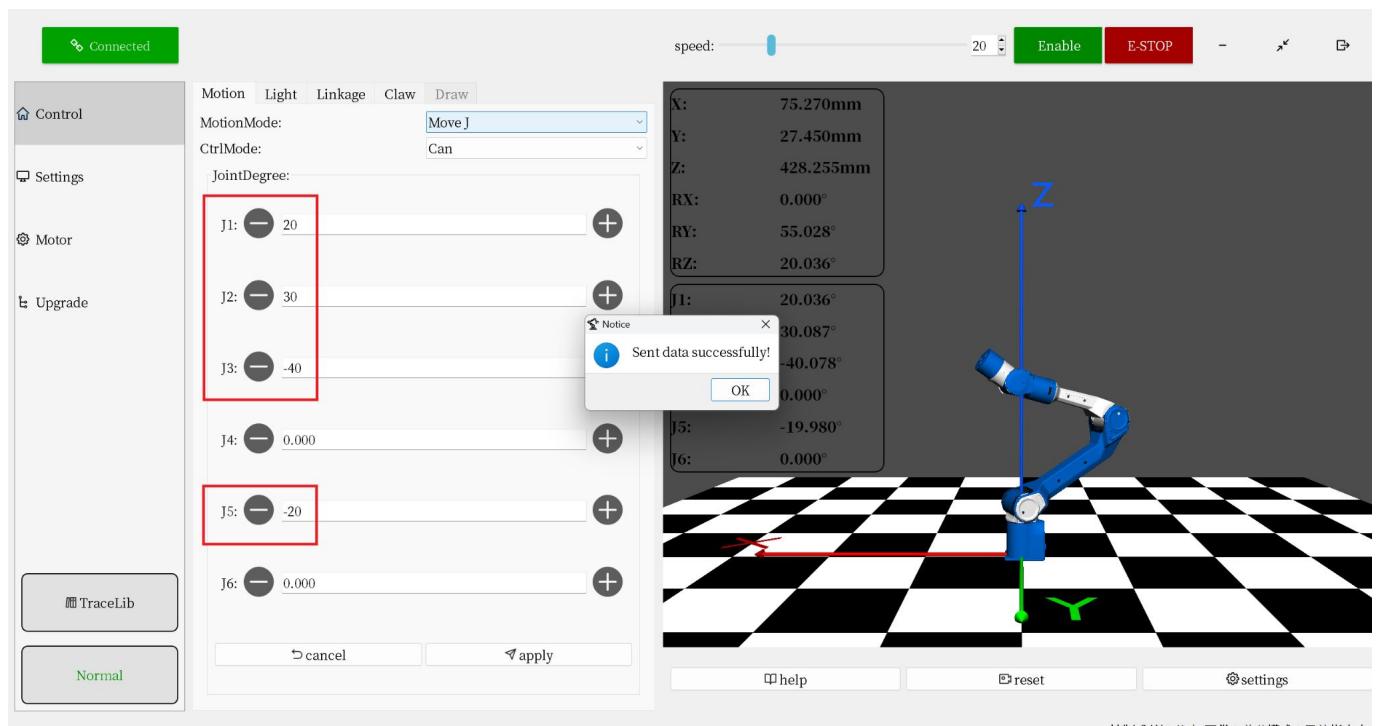


예시 2: MoveJ (조인트 기반 운동)

① J1(+), J3(-), J2(+), J5(-) 순으로 로봇팔의 관절 각도를 조절해 봅시다. 특히 J3와 J5는 (-)가 원점을 기준으로 위를 향하고, J2는 (+)로 위로 이동하니 주의하세요.



② Edit을 통해 더 정확한 관절 각도를 입력할 수 있으며, 이미 알고 있는 각도를 기반으로 6가지 입력값을 입력하여 동시에 해당 각도로 이동시킬 수도 있습니다



01. ArmRobotUA

예시 3: MoveP (포인트 기반 운동)

① 이번에도 Z값을 먼저 조절하겠습니다. 상단 부분은 Cartesian 좌표계를 기준으로 포지션을 조절합니다.

우측의 Δ 는 각 버튼을 눌렀을 때의 이동 단위(mm)로, 0.1mm부터 5mm까지 조절이 가능합니다. 처음에는 3mm를 단위 거리로 설정하시는 걸 추천드립니다.

CoordinatePosition(unit:mm)

The interface shows a grid of movement buttons and their corresponding coordinate labels:

- Z-**: Down arrow button (highlighted with a red box).
- X-**: Left arrow button.
- X+**: Right arrow button.
- Y-**: Up arrow button.
- Y+**: Down arrow button (highlighted with a red box).
- Z+**: Up arrow button (highlighted with a red box).

Position values at the bottom:

- X: 75.040
- Y: 27.261
- Z: 410.176

Delta values on the right:

- $\Delta X: 1.000$
- $\Delta Y: 1.000$
- $\Delta Z: 3.000$ (highlighted with a red box)

② 하단 부분은 Euler Transformation을 기준으로 회전 각도를 조절합니다. 이번에는 Ry를 기준으로 회전 각도를 조절해봅시다.

우측의 Δ 는 각 버튼을 눌렀을 때의 각도(Degree)로, 0.1°부터 5°까지 조절이 가능합니다. 처음에는 0.1°로 단위 각도로 설정하시는 걸 추천드립니다.

ArmPosture(unit:°)

The interface shows a grid of movement buttons and their corresponding coordinate labels:

- Rz-**: Down arrow button.
- Ry-**: Left arrow button (highlighted with a red box).
- Rx+**: Up arrow button.
- Ry+**: Right arrow button (highlighted with a red box).
- Rx-**: Down arrow button.

Posture values at the bottom:

- Rx: 0.000
- Ry: 59.767
- Rz: 19.865

Delta values on the right:

- $\Delta Rx: 0.100$
- $\Delta Ry: 0.100$
- $\Delta Rz: 0.100$ (highlighted with a red box)

③ 이외에 각 X, Y, Z, RX ,RY, RZ 값을 모두 변경시켜 'send'를 통해 동시에 입력이 가능합니다. 또한, 'dump'를 통해 현재 위치를 반환할 수도 있습니다.

01. ArmRobotUA

예시 4: MoveC (곡선 운동)

곡선 운동은 총 3가지 점(시작점, 경유점, 도착점)을 필요로 하며, 각 위치를 반영하기 위해 ‘dump’ 기능을 활용할 수 있습니다.

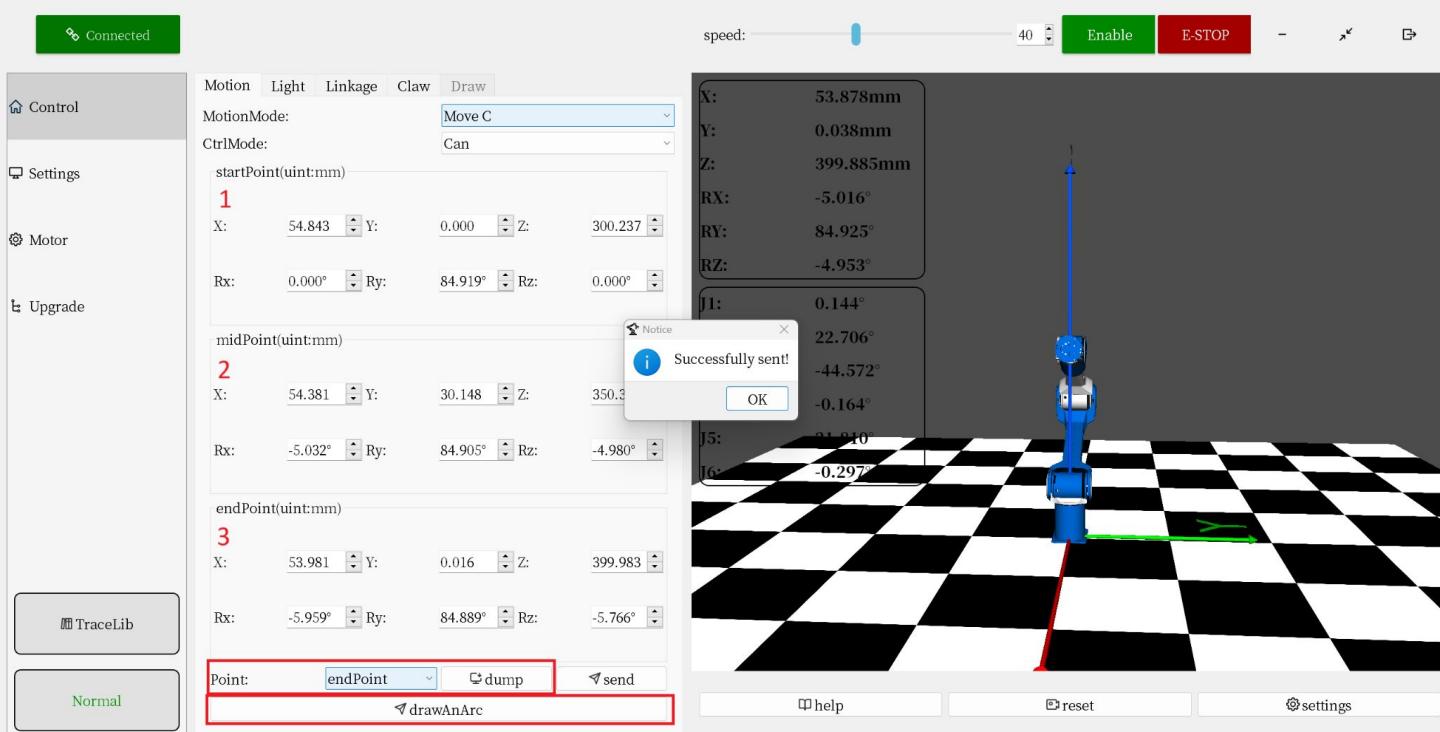
① 먼저, ‘reset’을 눌러 원위치로 이동시킨 뒤 MoveJ 및 MoveL을 통해 목표 위치값을 입력해봅니다.

② MoveL로 Z값을 300으로 맞춘 후, MoveC에서 ‘dump’를 클릭합니다.

③ 이후 MoveL로 Y를 30, Z를 350으로 버튼을 통해 위치시킵니다. 그 후 MoveC에서 Point 부분을 변경하여 ‘midpoint’로 설정하고 ‘dump’를 클릭합니다.

④ 다시 MoveL에서 Y를 0, Z를 400으로 버튼을 통해 위치시킨 후, ‘end point’로 변경하고 ‘dump’를 클릭합니다.

⑤ 최종적으로 속도를 40에 맞춘 뒤, ‘draw an arc’를 클릭하여 실행시킵니다.

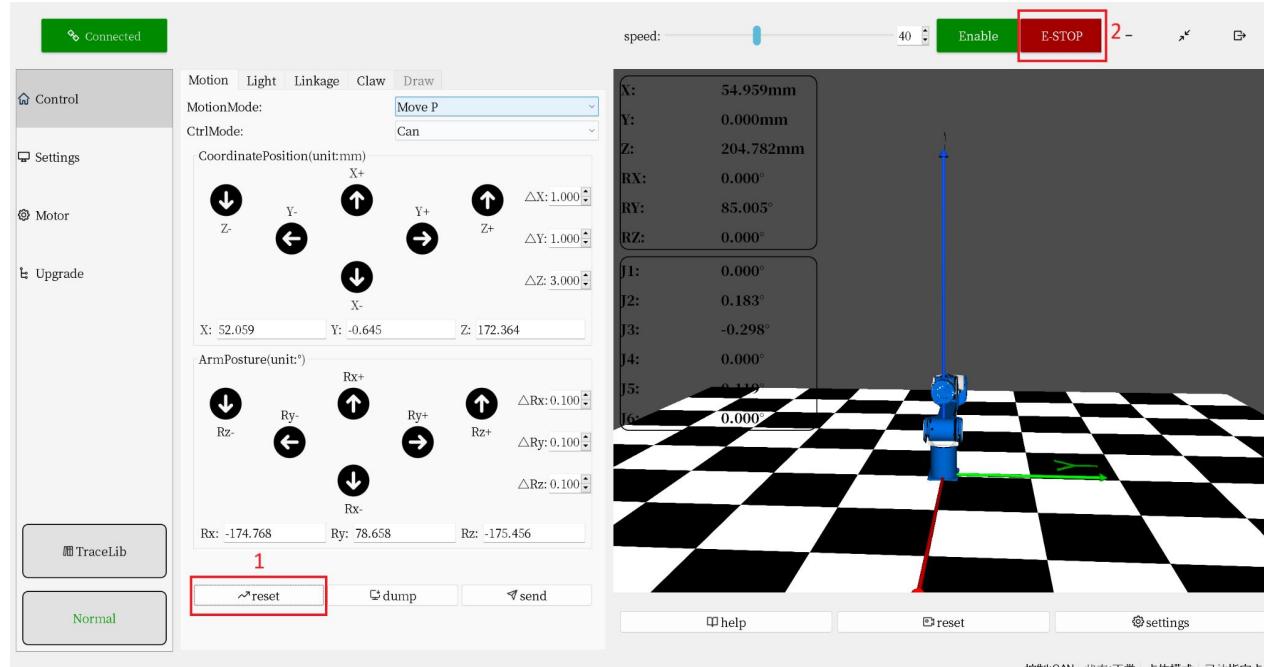


14. 로봇팔을 안전하게 종료시키기 위한 절차

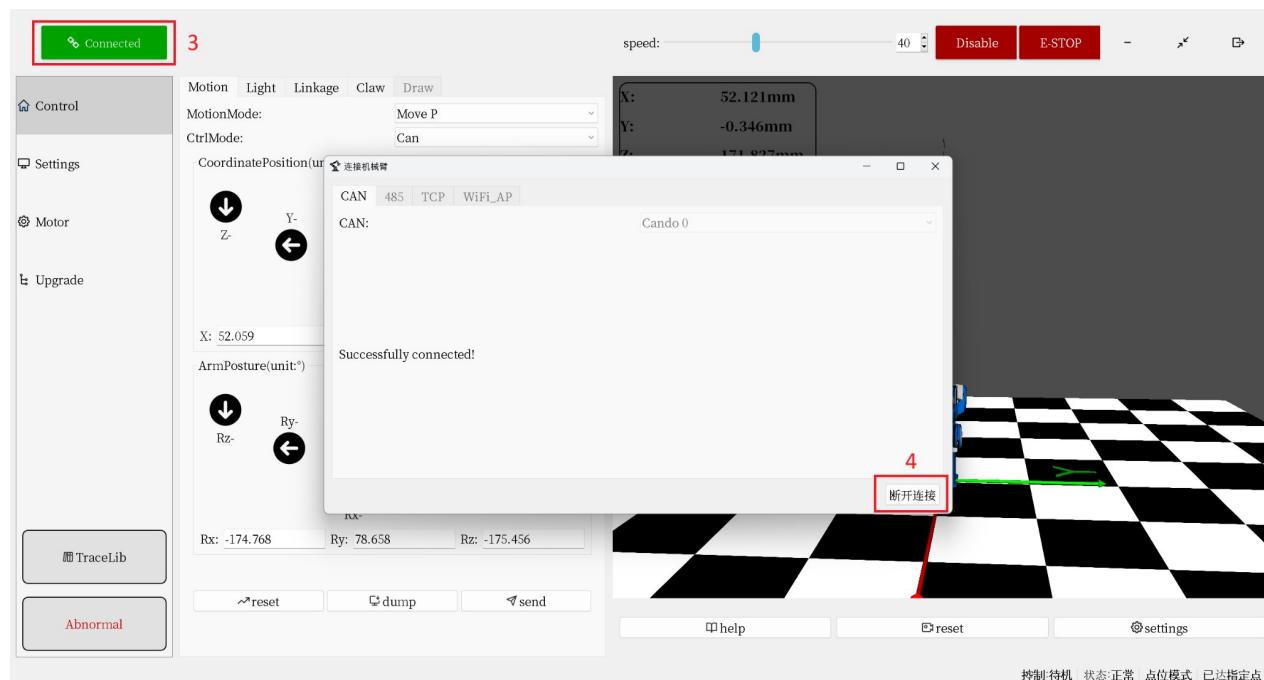
① ‘reset’ 버튼을 눌러 원위치로 복귀시킵니다.

② ‘E-STOP’ 버튼으로 안전하게 종료시킵니다.

- 이때, 로봇팔 그리퍼 부분을 손으로 받쳐주면 스크래치 방지를 할 수 있습니다.
- 이후 로봇팔이 자연스럽게 Disable 상태로 전환됩니다.



③ ‘ArmComm’을 클릭 후 Disconnect를 눌러 통신을 종료시킵니다. 이후 전원을 차단하시면 됩니다.



PythonSDK

02

02. Python SDK

[1] 설치 및 설정 가이드

Python SDK를 사용하면 CAN 인터페이스로 외부 컴퓨터와 통신하고 PiPER에 더 구체적이고 커스텀이 가능한 명령을 보낼 수 있습니다.

자세한 내용은 링크를 참고하세요: [GitHub - PiPER SDK](#)

- 인터페이스 세부 정보: [Interface README](#)
- 프로토콜 파싱: [Protocol README](#)
- 메시지: [Msgs README](#)

1. Python 버전 및 의존성 설치

Python SDK 설치를 위해 Python 3.8 또는 3.10이 필요합니다. 버전을 확인합니다.

```
python3 –version
```

이후 아래 명령어로 pip 패키지를 설치하세요:

```
sudo apt install pip3
```

추가로 net-tools 패키지가 없을 경우 아래 명령어로 설치하세요:

```
sudo apt install net-tools
```

2. PiPER Workspace 클론 및 설정

터미널에 아래 명령어를 입력하여 소스 코드를 다운로드합니다.

```
git clone https://github.com/WeGo-Robotics/piper\_ws.git
```

WeGo Robotics에서 제공하는 ID와 PW를 입력해야 합니다.

ID : WeGo-Robotics

PW : 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

다운로드 받은 piper_ws 폴더로 이동합니다.

```
cd piper_ws
```

02. Python SDK

원클릭 셋팅 파일 권한 설정 터미널에 아래 명령어를 입력하여, 원클릭 셋팅 파일 (setup_dependencies.sh)의 실행 권한을 부여합니다.

```
chmod +x setup_dependencies.sh
```

원클릭 셋팅 파일을 실행하여 필요한 의존성을 설치합니다.

```
./setup_dependencies.sh
```

```

CMake 최신 버전 업데이트 중...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
cmake is already the newest version (3.22.1-1ubuntu1.22.04.2).
The following packages were automatically installed and are no longer required:
  libnvidia-cfg1-550 libnvidia-common-550 libnvidia-decode-550 libnvidia-decode-550:i386
  libnvidia-fbc1-550 libnvidia-fbc1-550:i386 libnvidia-gl-550 libnvidia-gl-550:i386
  xserver-xorg-video-nvidia-550
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

Python 패키지 설치 중...
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pip in /home/wego/.local/lib/python3.10/site-packages
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: python-can in /home/wego/.local/lib/python3.10/site-packages
Requirement already satisfied: piper_sdk in /home/wego/.local/lib/python3.10/site-packages
Requirement already satisfied: scipy in /usr/lib/python3/dist-packages (from -r requirements.txt)
Requirement already satisfied: wrapt~=1.10 in /usr/lib/python3/dist-packages (from -r requirements.txt)
Requirement already satisfied: packaging>=23.1 in /home/wego/.local/lib/python3.10/site-packages
Requirement already satisfied: typing_extensions>=3.10.0.0 in /home/wego/.local/lib/python3.10/site-packages
Requirement already satisfied: msgpack~=1.1.0 in /home/wego/.local/lib/python3.10/site-packages
✓ 모든 의존성이 설치되었습니다!

```

빌드 및 실행 준비 필요한 의존성 설치가 완료되면, colcon build 명령어로 빌드를 실행하여 준비를 마칩니다.

```
colcon build
```

* colcon build시 moveit-visual-tools 오류가 발생한다면 터미널에 아래 명령어를 입력하면 설치됩니다.

```
sudo apt install ros-humble-moveit-visual-tools
```

* PIPER ROS2 환경까지 모두 설치됩니다.

02. Python SDK

PC가 단일 CAN-to-USB 모듈에 연결된 경우, 아래 스크립트를 실행하여 설정합니다.

```
bash can_activate.sh can0 1000000
```

--START--

```
Both ethtool and can-utils are installed.
[sudo] password for wego:
Expected to configure a single CAN module, detected interface can0 with corresponding USB address 3-1.4.4:1.0.
Interface can0 is not activated or bitrate is not set.
Interface can0 has been reset to bitrate 1000000 and activated.
```

--OVER--

CAN 네트워크 인터페이스(can0)에 대한 버스(bus) 정보를 확인합니다.

```
sudo ethtool -i can0 | grep bus
```

```
bus-info: 3-1.4.4:1.0
```

CAN 모듈 여러 개 사용 시, 모든 다른 모듈을 제거한 뒤 Piper CAN 통신을 먼저 설정하세요. 아래와 같이 USB 주소와 CAN 이름을 지정하여 다시 실행할 수 있습니다.

```
bash can_activate.sh can_piper 1000000 "3-1.4.4:1.0"
```

- USB 주소: 실제로 확인된 값을 사용하세요.
- 기본 CAN 인터페이스 이름(can0)은 사용자 지정 이름(can_piper)으로 변경됩니다.

--START--

```
Both ethtool and can-utils are installed.
Detected USB hardware address parameter: 3-1.4.4:1.0
Found the interface corresponding to USB hardware address 3-1.4.4:1.0: can0.
Interface can0 is already activated with a bitrate of 1000000.
Rename interface can0 to can_piper.
The interface has been renamed to can_piper and reactivated.
```

--OVER--

4. CAN 설정 확인

CAN 모듈 설정 여부를 확인합니다.

```
ifconfig
```

--OVER--

```
wego@wego-GS66-Stealth-10SD:~/piper_ws$ ifconfig
can0: flags=193<UP,RUNNING,NOARP> mtu 16
      unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
              txqueuelen 10  (UNSPEC)
      RX packets 167615 bytes 1340920 (1.3 MB)
      RX errors 0 dropped 167615 overruns 0 frame 0
      TX packets 0 bytes 0 (0.0 B)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

02. Python SDK

CAN 버스에서 데이터 프레임을 실시간 수신 및 출력을 모니터링 합니다.

```
candump can0
candump can_piper(can_piper로 지정했을 시)
```

```
can0 254 [8] 00 11 00 00 00 00 00 00 00
can0 255 [8] 00 00 00 00 00 00 00 00 00
can0 256 [8] 00 33 00 00 00 00 00 00 00
can0 2A1 [8] 01 00 00 00 00 00 00 00 00
can0 2A2 [8] 00 00 B3 33 FF FF FE 37
can0 2A3 [8] 00 02 96 0B FF FD 58 77
can0 2A4 [8] 00 01 2B A1 FF FD 57 72
can0 2A5 [8] FF FF FE F3 FF FF E8 C6
can_piper 2A3 [8] 00 02 95 D3 FF FD 85 CD
can_piper 2A4 [8] 00 01 2D A1 FF FD 92 00
can_piper 2A5 [8] 00 00 0E 46 FF FF F8 8D
can_piper 2A6 [8] 00 00 05 16 FF FF F4 1E
can_piper 2A7 [8] 00 00 45 89 FF FF FC BC
can_piper 2A8 [8] 00 00 00 00 00 00 00 00 00
```

5. VS CODE로 코드 확인 및 수정

현재 디렉토리에서 VS Code가 실행되어 예제 코드를 확인 및 수정이 가능합니다.

```
cd src/piper_sdk/scripts
code .
```

```
1-5.piper_MoveAll.py
scripts > 1-5.piper_MoveAll.py > ...
1  #!/usr/bin/env python3
2  # -*- coding:utf8 -*-
3
4  from typing import Optional
5  from piper_sdk import *
6  from WeGo_MetaClass import WeGo
7
8 """
9 메타클래스를 활용하여 포지션 값만 입력하면 로봇팔이 자동으로 목표 위치로 이동 가능
10 """
11
12 class MoveAll:
13     def __init__(self, piper:C_PiperInterface):
14         # piper 인스턴스 생성 및 연결
15         self.piper = piper
16         self.piper.ConnectPort() # 포트 연결
17
18         # WeGo 인스턴스 생성 및 로봇팔 활성화
19         self.wego = WeGo(self.piper)
20         self.wego.enable.run()
21
22         self.position = [0,0,0,0,0,0,0]
23         self.positions = [
24             [0,0,0,0,0,0x01],
25             [0,0,0,0,0,0x02],
26             [0,0,0,0,0,0x03]
27         ]
28
29     def move(self):
30         # movej => 조인트 각도(Deg) : [조인트1, 조인트2, 조인트3, 조인트4, 조인트5, 조인트6, 그리퍼]
31         self.position = [0, 0, 0, 0, 0, 0, 0]
32         self.wego.movej.run(*self.position)
33
34         # movevp / l => 포지션(mm, Deg) : [X 좌표값, Y 좌표값, Z 좌표값, RX (회전) 좌표값, RY (회전) 좌표값, RZ (회전) 좌표값, 그리퍼]
35         self.position = [190, 0, 400, 0, 85, 0, 70]
36         self.wego.movevp.run(*self.position)
37
38         self.position = [190, 0, 250, 0, 85, 0, 0]
39         self.wego.movel.run(*self.position)
40
```

02. Python SDK

[2] PiPER SDK 예제

PiPER SDK를 활용하여 더 자세한 정보를 읽어오거나, 피드백 값을 수정하거나 PiPER를 제어하기 위한 명령어를 송신할 수 있습니다.

주의: 예제에서 사용한 CAN 포트가 can0이 아닌 경우, C_PiperInterface("can_piper")와 같이 CAN 포트를 수정해야 합니다.

1. PiPER 정보 읽기

PiPER의 여러 가지 정보를 읽기 위해 SDK의 scripts 폴더로 이동하여 아래의 Python 스크립트를 실행합니다.

```
cd piper_ws/src/piper_sdk/scripts  
python3 0-1. piper_arm_info.py
```

다음은 코드 예시입니다:

```
#!/usr/bin/env python3  
# -*-coding:utf8-*-  
  
"""  
PiPER에 관한 대부분의 피드백을 해당 파일에서 확인할 수 있습니다.  
한 개씩 주석을 해제하며 차근차근 확인하실 수 있습니다.  
"""  
  
from typing import Optional  
import time, math  
from piper_sdk import *  
  
if __name__ == "__main__":  
    piper = C_PiperInterface()  
    piper.ConnectPort()  
    while True:
```

02. Python SDK

```
# # [1] PiPER 슬레이브 암 관련 피드백 받는 함수
# print(piper.GetArmStatus())
# print(piper.GetArmEndPoseMsgs())
# print(piper.GetArmJointMsgs())
# print(piper.GetArmGripperMsgs())
# print(piper.GetArmHighSpdInfoMsgs())
# print(piper.GetArmLowSpdInfoMsgs())
# print(piper.GetCurrentEndVelAndAccParam())
# print(piper.GetCrashProtectionLevelFeedback())

## PiPER 특정 모터의 피드백 받는 함수
# piper.SearchMotorMaxAngleSpdAccLimit(1,0x02)
# print(piper.GetCurrentMotorMaxAccLimit())
# print(piper.GetCurrentMotorAngleLimitMaxVel())

## PiPER 전체 모터의 피드백 받는 함수
# piper.SearchAllMotorMaxAngleSpd()
# print(piper.GetAllMotorAngleLimitMaxSpd())
# piper.SearchAllMotorMaxAccLimit()
# print(piper.GetAllMotorMaxAccLimit())

## [2] PiPER 마스터 암 관련 피드백 받는 함수
# print(piper.GetArmJointCtrl())
# print(piper.GetArmGripperCtrl())
# print(piper.GetArmCtrlCode151())

## [3] PiPER 기타 피드백 받는 함수
# PiPER Forward Kinematics 피드백 받는 함수
# print(f"feedback: {piper.GetFK('feedback')}")
# print(f"control: {piper.GetFK('control')}")

# CAN통신 FPS 피드백 받는 함수
# print(f"can: {piper.GetCanFps()}")
# print(f"all_fps: {piper.GetCanFps()}")
# print(f"status: {piper.GetArmStatus().Hz}")
# print(f"end_pose: {piper.GetArmEndPoseMsgs().Hz}")
# print(f"joint_states: {piper.GetArmJointMsgs().Hz}")
# print(f"gripper_msg: {piper.GetArmGripperMsgs().Hz}")
# print(f"high_spd: {piper.GetArmHighSpdInfoMsgs().Hz}")
# print(f"low_spd: {piper.GetArmLowSpdInfoMsgs().Hz}")
# print(f"joint_ctrl: {piper.GetArmJointCtrl().Hz}")
# print(f"gripper_ctrl: {piper.GetArmGripperCtrl().Hz}")
# print(f"ctrl_151: {piper.GetArmCtrlCode151().Hz}")
time.sleep(0.005)
pass
```

스크립트 내부의 주석을 지우고 실행하면 파이퍼의 여러 가지 정보(로봇팔의 상태, 6개 모터의 각도, 그리퍼 상태 등)들을 확인할 수 있습니다.

02. Python SDK

2. PiPER 조인트 제어

PiPER 조인트를 SDK 기능을 통해 제어하려면 아래 Python 스크립트를 실행합니다.

```
python3 1-1. piper_MoveJ.py
```

다음은 코드 예시입니다:

```
#!/usr/bin/env python3
# -*-coding:utf8-*-
from typing import Optional
import time, math
from piper_sdk import *
from piper_sdk.interface.piper_interface_v2 import C_PiperInterface_V2

class Move_J:
    def __init__(self, piper: C_PiperInterface_V2):
        self.piper = piper # piper 인터페이스 객체 초기화
        self.factor = 1000 # 0.001 도를 기준으로 int값을 넘겨주기 때문에 설정 (단위 변환용)
        self.position = [0, 0, 0, 0, 0, 0, 0] # 초기 위치 (각 관절)
        self.gripper = 0 # 초기 위치 (그리퍼)
        self.count = 0 # 이동 횟수 카운트 초기화

    def enable(self):
        """
        로봇 팔을 활성화하고, 모터가 준비될 때까지 대기하는 함수입니다.
        """
        enable_flag = False # 팔이 활성화되었는지 확인하는 플래그
        elapsed_time_flag = False # 시간 초과 여부를 체크하는 플래그
        timeout = 5 # 타임아웃 시간 (초)
        start_time = time.time() # 시작 시간 기록
        while not enable_flag: # 팔이 활성화 될 때까지 반복
            elapsed_time = time.time() - start_time # 경과 시간 계산
            enable_flag = all( # 모든 모터가 활성화되었는지 확인
                getattr(self.piper.GetArmLowSpdInfoMsgs(), f"motor_{i}").foc_status.driver_enable_status
                for i in range(1, 7)
            )
            self.piper.EnableArm(7) # 로봇 팔 활성화
            self.piper.GripperCtrl(0, 1000, 0x01, 0) # 그리퍼 활성화 (초기 상태)

        if elapsed_time > timeout: # 타임아웃 시간 초과 시
            print("시간초과")
            elapsed_time_flag = True # 시간 초과 플래그 설정
            enable_flag = True # 루프 종료
            break
        time.sleep(1) # 1초 간격으로 체크

    if elapsed_time_flag: # 시간 초과 발생 시
        print("프로그램을 종료합니다")
        exit(0) # 프로그램 종료

    def convert(self):
        return [round(angle * self.factor) for angle in self.position] + [round(self.gripper * self.factor)]
```

02. Python SDK

```
def move(self):
    while True:
        print(self.piper.GetArmJointMsgs()) # 현재 로봇 팔 조인트 각도 출력
        self.count += 1 # 카운트 증가
        print(self.count) # 카운트 출력

    # 매번 조인트 별로 45도씩 돌아가는 예제 코드
    if self.count == 1:
        print("원 점")
        self.position = [0, 0, 0, 0, 0, 0]
        self.gripper = 0
    elif self.count == 200:
        print("웨이포인트 1")
        self.position = [45, 0, 0, 0, 0, 0]
        self.gripper = 0
    elif self.count == 400:
        print("웨이포인트 2")
        self.position = [0, 45, 0, 0, 0, 0]
        self.gripper = 0
    elif self.count == 600:
        print("웨이포인트 3")
        self.position = [0, 0, -45, 0, 0, 0]
        self.gripper = 0
    elif self.count == 800:
        print("웨이포인트 4")
        self.position = [0, 0, 0, 45, 0, 0]
        self.gripper = 0
    elif self.count == 1000:
        print("웨이포인트 5")
        self.position = [0, 0, 0, 0, -45, 0]
        self.gripper = 0
    elif self.count == 1200:
        print("웨이포인트 6")
        self.position = [0, 0, 0, 0, 0, 45]
        self.gripper = 0
    elif self.count == 1400:
        print("웨이포인트 7")
        self.position = [0, 0, 0, 0, 0, 0]
        self.gripper = 70
    elif self.count == 1600:
        print("웨이포인트 8")
        self.position = [45, 45, -45, 45, -45, 45]
        self.gripper = 0
    elif self.count == 1800:
        self.count = 0 # 카운트 초기화
```

02. Python SDK

```
# 변환된 값 적용
converted_values = self.convert()
joint_values = converted_values[:6] # 첫 6개는 조인트 값
gripper_value = converted_values[6] # 마지막 값은 그리퍼

# 로봇의 동작을 제어하는 명령
self.piper.MotionCtrl_2(0x01, 0x01, 50, 0x00) # 팔 모션 제어 (속도 및 제어 모드)
self.piper.JointCtrl(*joint_values) # 조인트 제어
self.piper.GripperCtrl(abs(gripper_value), 1000, 0x01, 0) # 그리퍼 제어
time.sleep(0.005) # 5ms 대기

# 메인 함수
def main():
    piper = C_PiperInterface("can0") # piper 인터페이스 생성
    piper.ConnectPort() # 포트 연결

    move_j = Move_J(piper) # Move_J 객체 생성
    move_j.enable() # 팔 활성화
    move_j.move() # 팔 이동

if __name__ == "__main__":
    main() # 메인 함수 실행
```

스크립트를 실행하면 count 변수에 따라 Joint가 45도씩 움직이며, 그리퍼가 열리고 닫힙니다. 각도는 degree 단위로 값을 제공하여 관절의 움직임을 관찰할 수 있습니다.

02. Python SDK

3. PiPER E-stop 및 초기화

PIPER를 사용하다 긴급하게 멈춰야 할 경우, 또는 PIPER의 작동을 종료해야 할 경우 아래의 스크립트를 실행합니다.

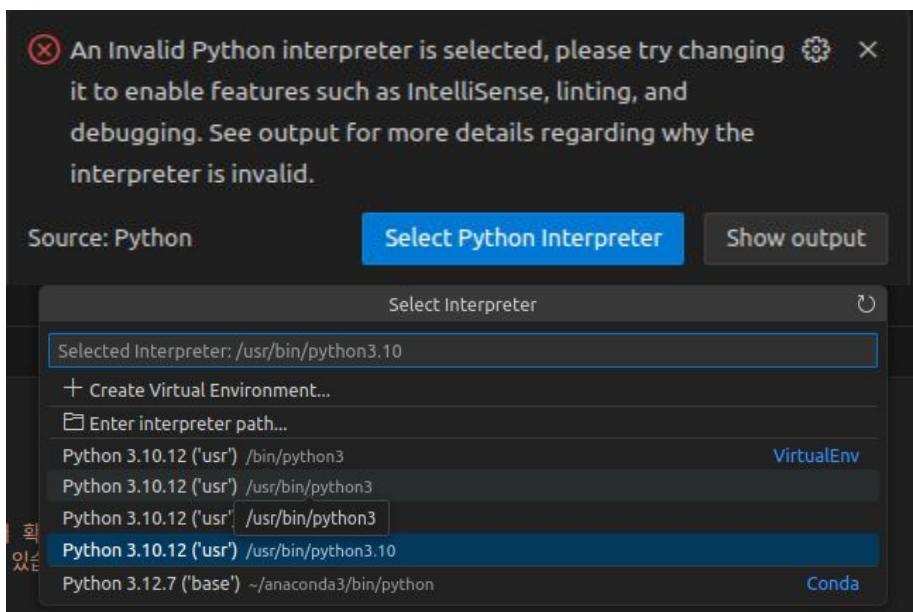
```
cd piper_ws/src/piper_sdk/scripts
python3 0-2. piper_emergency_stop.py
```

또는 visual studio code에서 실행합니다.



```
0-2. piper_emergency_stop.py
1 #!/usr/bin/env python3
2 # -*- coding: utf8 -*-
3
4 """
5   긴하게 PiPER를 멈추고 싶을 경우, 해당 파일을 꼭 실행해야 합니다!
6 """
7
8 from typing import Optional
9 from piper_sdk import *
10
11 if __name__ == "__main__":
12     piper = C_PiperInterface()
13     piper.ConnectPort()
14
15     piper.MotionCtrl_1(0x01, 0, 0x00) # 긴급 정지버튼 (E-stop)
16
17     pass
```

*아래와 같은 Python Interpreter 오류가 발생할 경우, Python Interpreter - Python 3.10 버전을 선택하면 됩니다.



스크립트를 실행하면 PIPER의 동작이 멈추고, E-stop 동작을 실행해 로봇 암이 천천히 하강하게 됩니다.

02. Python SDK

E-stop이 실행 중인 PIPER를 초기화시켜 다시 사용하면, 아래의 스크립트를 실행합니다.

```
python3 0-3. piper_emergency_restore.py
```

또는 visual studio code에서 실행합니다.



```

EXPLORER ... 0-3. piper_emergency_restore.py ...
src > piper_sdk > scripts > 0-3. piper_emergency_restore.py > ...
1 #!/usr/bin/env python3
2 # -*- coding: utf8 -*-
3
4 """
5 해당 파일은 0-2. piper_emergency_stop.py 파일을 실행시켰을 경우에만 적용됩니다.
6 해당 파일은 PIPER 모드를 리셋하기 위해 차운으로 Disable를 적용시키기 때문에 PIPER를 안전한 위치에 놓고 실행시켜주세요.
7 단장 모드 적용 시, 현재는 리셋이 불가능합니다. 전원선을 뽑았다가 다시 재인가해주세요.
8 """
9
10 from typing import Optional
11 import time, math
12 from piper_sdk import *
13
14 if __name__ == "__main__":
15     piper = C_PiperInterface()
16     piper.ConnectPort()
17
18     piper.MotionCtrl_1(0x02, 0, 0x00)
19     time.sleep(1)
20
21     piper.MotionCtrl_1(0x00, 0, 0x00)

```

스크립트를 실행하면 초기화된 PIPER를 다시 사용할 수 있습니다.

주의: E-stop 스크립트를 실행하지 않고 PIPER 전원을 종료하거나,

emergency_restore 스크립트를 실행할 경우, PIPER가 빠른 속도로 하강해 **제품이 파손될** 수 있습니다.

ROS1

03

03. ROS1

설치 및 설정 가이드

[PiPER ROS 패키지](#)는 PiPER SDK를 기반으로 구축되어 있으며, 통합된 PC와 데이터를 읽고 전송하는 사용자 정의 ROS 메시지를 포함하고 있습니다.

해당 패키지는 Ubuntu 20.04와 ROS Noetic과 호환됩니다. ROS 패키지의 전제 조건은 PiPER Python SDK와 ROS이므로, 이미 설치되어 있는지 확인하십시오.

1. PiPER ROS1 패키지 설치

PiPER ROS1 패키지를 설치하려면 아래 명령어를 순차적으로 입력하세요.

```
mkdir -p ~/piper_ros_ws/src && cd ~/piper_ros_ws/src  
git clone https://github.com/agilexrobotics/piper_ros.git  
cd ..  
catkin_make
```

빌드가 성공적으로 완료되면, launch 파일을 사용하여 PiPER 제어 노드를 실행할 수 있습니다.

다만, launch 파일 실행 전에 CAN 통신이 활성화되어야 하며, 활성화 방법은 앞서 설명된 Python SDK 부분을 참조하십시오.

주의: Drag and Teach 모드가 사용된 경우 PiPER 전원을 차단하고 재공급해야 합니다.

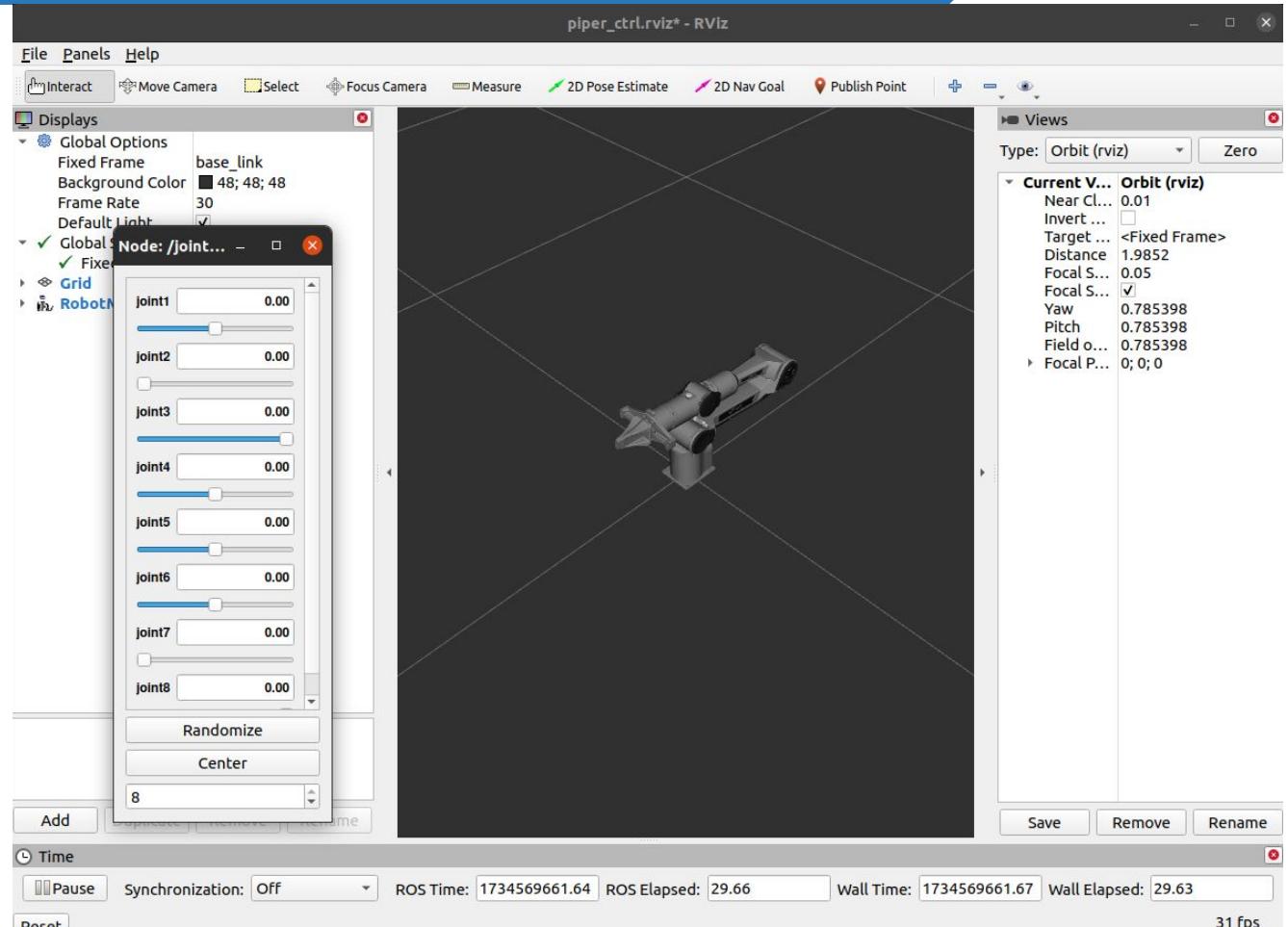
2. 소스 빌드 후 Rviz로 제어

launch 파일을 실행하려면 다음 명령어를 입력하세요.

```
source devel/setup.bash  
rosrun piper start_single_piper_rviz.launch
```

그러면 새로운 “Rviz” 창과 함께 조인트를 제어할 수 있는 QT 기반 GUI가 열립니다.

03. ROS1



슬라이더를 조작하여 조인트 각도를 변경할 수 있습니다.

Rviz 없이 로봇 팔을 제어하려면 다음 명령어를 입력하세요:

```
source devel/setup.bash
roslaunch piper start_single_piper.launch can_port:=can0
```

PiPER 제어 노드에서 사용할 수 있는 ROS 토픽 목록은 다음과 같습니다.

- /arm_status: 로봇 팔의 상태
- /enable_flag: 노드로 전송되는 활성화 플래그. "true"를 전송하여 활성화
- /end_pose: 로봇 팔의 엔드 이펙터 포즈에 대한 피드백
- /joint_states: 인트 위치를 전송하여 로봇 팔의 움직임을 제어
- /joint_states_single: 로봇 팔의 조인트 상태에 대한 피드백
- /pos_cmd: 엔드 이펙터 제어 메시지

ROS 토픽을 사용하여 PiPER를 제어하려면 sensor_msgs/JointState 메시지 형식으로 데이터를 게시할 수 있습니다.

03. ROS1

모션 관련 토픽을 게시하기 전에 먼저 "/enable_flag" 토픽을 "true"로 게시해야 합니다.

```
rostopic pub -1 /enable_flag std_msgs/Bool "data: true"
```

이제 조인트를 변경하려면 "/joint_states" 토픽을 아래와 같이 게시할 수 있습니다.

```
rostopic pub /joint_states sensor_msgs/JointState "header:  
seq: 0  
stamp: {secs: 0, nsecs: 0}  
frame_id: "  
name: []  
position: [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.04]  
velocity: [0,0,0,0,0,10]  
effort: [0,0,0,0,0,0.5]"
```

위 메시지를 살펴보면 position, velocity, effort가 7번째 인덱스에 0이 아닌 값이 들어있습니다. 이는 그리퍼를 열게 하는 메시지입니다. 이와 비슷한 방식으로 다른 조인트들도 제어할 수 있습니다.

Note:

- CAN 장치를 활성화하고 올바른 전송 속도를 설정해야만 로봇 팔을 읽거나 제어할 수 있습니다.
- "Enable Status: False" 또는 <class 'can.exceptions.CanError'> Message NOT sent라는 메시지가 표시되면 PiPER가 CAN 모듈에 연결되지 않았음을 나타냅니다. USB를 분리하고 다시 연결한 후 PiPER를 재시작하고 CAN 모듈을 활성화한 후, 노드를 다시 시작해주세요.
- 자동 활성화가 활성화된 상태에서 5초 이내에 활성화가 실패하면 프로그램이 자동으로 종료됩니다.
- 엔드 이펙터 포즈 제어가 필요하면 PiPER의 사용자 정의 메시지 "PoseCmd.msg"를 사용하십시오.
- 일부 특이점에는 접근할 수 없을 수 있습니다.

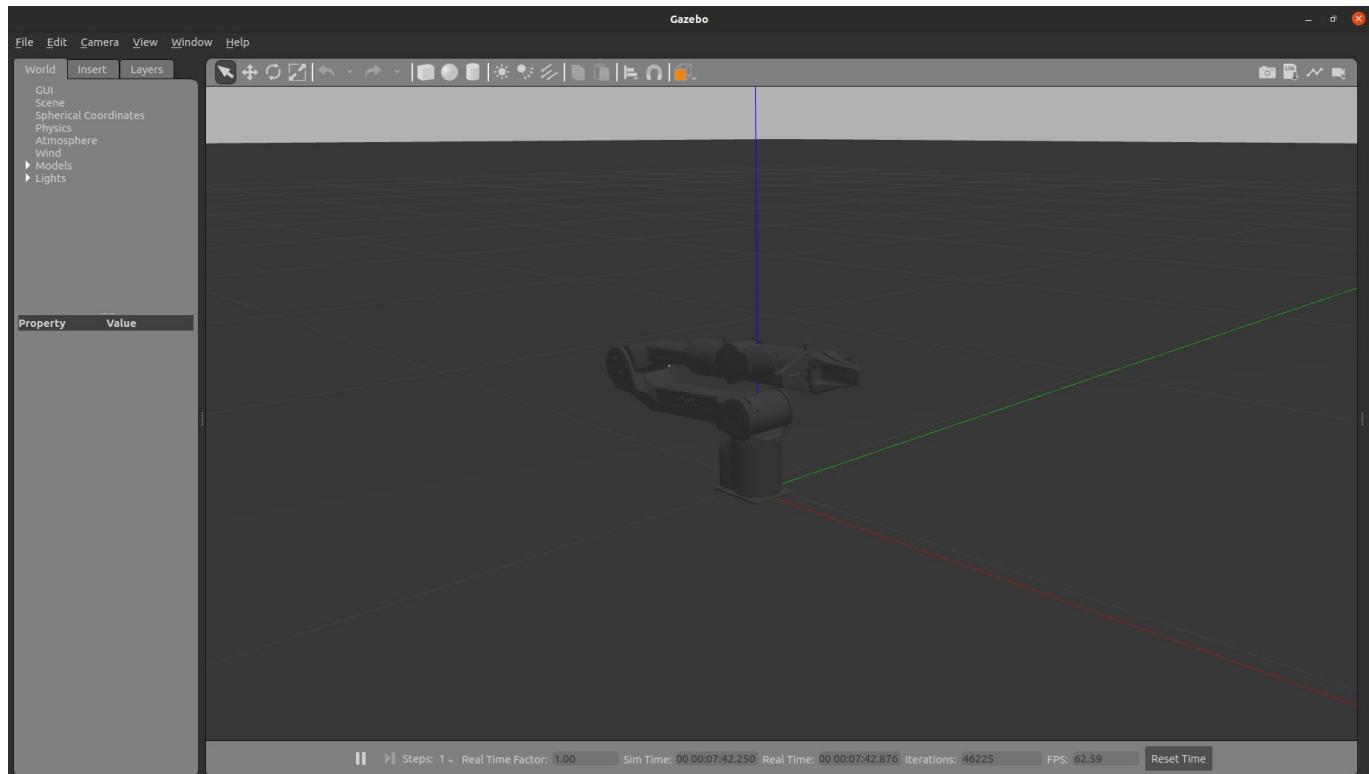
03. ROS1

3. Gazebo 시뮬레이션

Gazebo는 로봇의 동작과 센서를 3D 물리 시뮬레이션 환경에서 테스트할 수 있는 오픈소스 소프트웨어입니다. launch 파일을 실행하려면 다음 명령어를 입력하세요.

```
source devel/setup.bash
roslaunch piper_description gazebo_xacro.launch
```

그러면 새로운 “Gazebo” 창이 열립니다.



마찬가지로 ROS 토픽으로 로봇팔 제어가 가능합니다. 아래의 명령어를 입력하여 조인트1을 90도 회전 시킵니다.

```
rostopic pub /piper_description/joint1_position_controller/command std_msgs/Float64 "data: 1.57"
```

입력 값은 도(degree)가 아닌 라디안(radian)으로 입력해야 합니다. 우측 표는 관절의 작동 범위를 나타냅니다.

단, 7번과 8번 관절은 그리퍼의 작동을 의미하며, 미터(meter) 단위로 입력해야 합니다.

joint_name	limit
joint1	[-2.618,2.618]
joint2	[0,3.14]
joint3	[-2.697,0]
joint4	[-1.832,1.832]
joint5	[-1.22,1.22]
joint6	[-3.14,3.14]
joint7	[0,0.038]
joint8	[-0.038,0]

ROS2

04

04. ROS2

설치 및 설정 가이드

PiPER ROS 2 패키지 또한 PiPER SDK를 기반으로 구축되어 있으며, ROS 패키지와 동일한 기능을 포함하고 있습니다. ROS 2 패키지는 Ubuntu 22.04와 ROS Foxy/Humble과 호환됩니다.

1. PiPER ROS2 패키지 설치

PiPER Workspace 설치 과정에서 ROS2 패키지도 설치를 완료했기 때문에, 빌드가 성공적으로 완료됐다면 launch 파일을 사용하여 PiPER 제어 노드를 실행할 수 있습니다.

다만, launch 파일 실행 전에 CAN 통신이 활성화되어야 하며, 활성화 방법은 앞서 설명된 Python SDK 부분을 참조하십시오.

주의: Drag and Teach 모드가 사용된 경우 PiPER 전원을 차단하고 재공급해야 합니다.

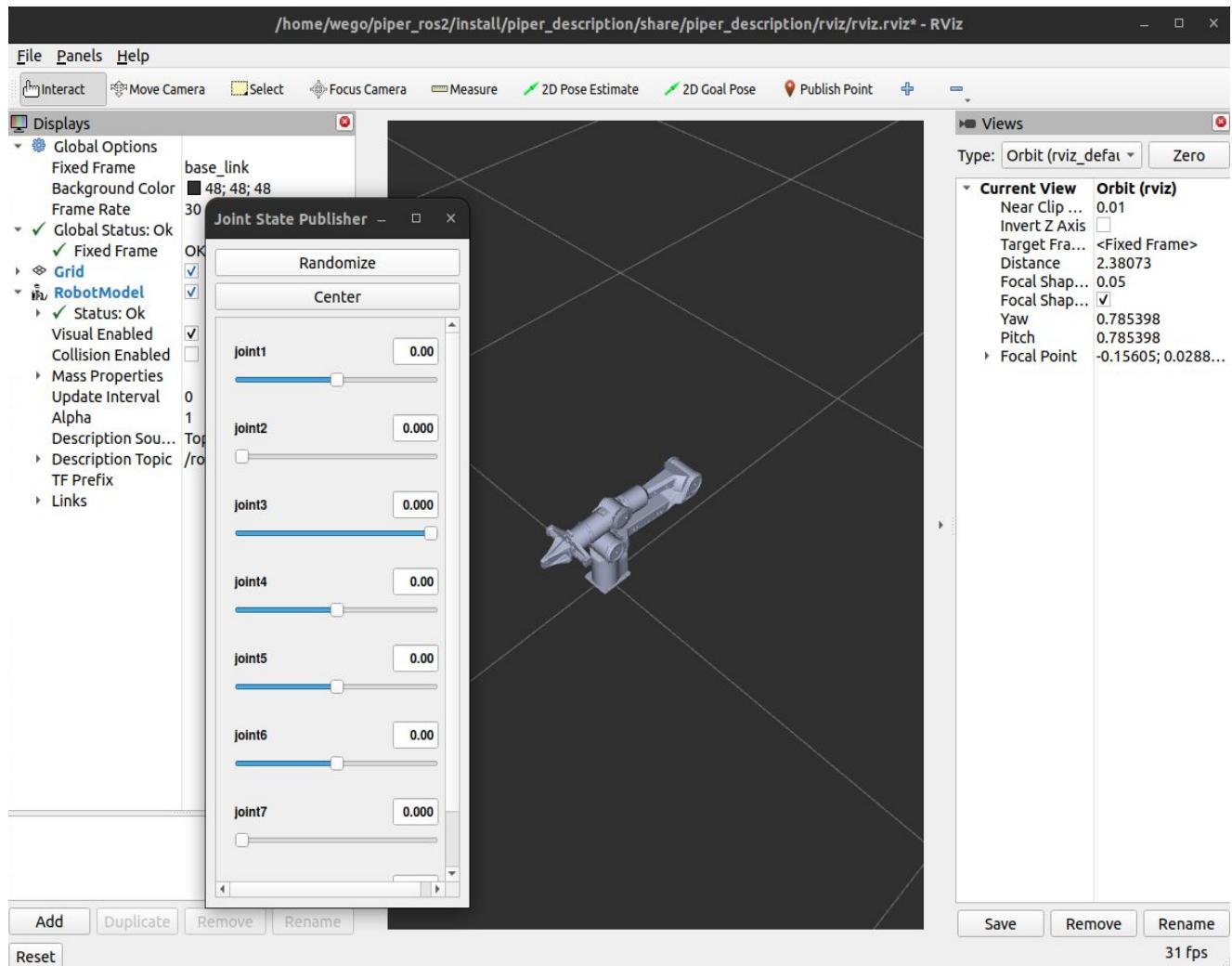
2. 환경 설정 후 Rviz로 제어

launch 파일을 실행하려면 다음 명령어를 입력하세요.

```
cd piper_ws  
source install/setup.bash  
ros2 launch piper start_single_piper_rviz.launch.py
```

04. ROS2

새로운 “Rviz 2” 창과 함께 조인트를 제어할 수 있는 QT 기반 GUI가 열립니다.



Rviz 없이 로봇 팔을 제어하려면 다음 명령어를 입력하세요:

```
source install/setup.bash
ros2 launch piper start_single_piper.launch
```

PiPER 제어 노드에서 사용할 수 있는 ROS 토픽 목록은 ROS1과 같습니다.

- /arm_status: 로봇 팔의 상태
- /enable_flag: 노드로 전송되는 활성화 플래그. "true"를 전송하여 활성화
- /end_pose: 로봇 팔의 엔드 이펙터 포즈에 대한 피드백
- /joint_states: 인트 위치를 전송하여 로봇 팔의 움직임을 제어
- /joint_states_single: 로봇 팔의 조인트 상태에 대한 피드백
- /pos_cmd: 엔드 이펙터 제어 메시지

04. ROS2

모션 관련 토픽을 게시하기 전에 먼저 "/enable_flag" 토픽을 "true"로 게시해야 합니다.

```
ros2 topic pub /enable_flag std_msgs/msg/Bool "data: true"
```

이제 조인트를 변경하려면 "/joint_states" 토픽을 아래와 같이 게시할 수 있습니다.

```
ros2 topic pub /joint_states sensor_msgs/msg/JointState "{header: {stamp: {sec: 0, nanosec: 0}, frame_id: 'piper_single'}, name: ['joint1', 'joint2','joint3','joint4','joint5','joint6','joint7'], position: [0.3,0,0,0,0,0,0.3], velocity: [0.5,0,0,0,0,0,0.5], effort: [0,0,0,0,0,0,0.5]}"
```

해당 명령어를 살펴보면 position에서 첫 번째 인덱스를 제외한 나머지 인덱스에는 0.0 아닌 값이 들어있습니다. 이는 첫 번째 조인트를 약 17.2° 회전 시키는 명령어임을 확인할 수 있습니다.

```
[piper_single_ctrl-1] [INFO] [1734573619.781584780] [piper_ctrl_single_node]: Received Joint States:  
[piper_single_ctrl-1] [INFO] [1734573619.781979765] [piper_ctrl_single_node]: joint_0: 0.3  
[piper_single_ctrl-1] [INFO] [1734573619.782340394] [piper_ctrl_single_node]: joint_1: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.782693544] [piper_ctrl_single_node]: joint_2: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.783050461] [piper_ctrl_single_node]: joint_3: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.783388768] [piper_ctrl_single_node]: joint_4: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.783729231] [piper_ctrl_single_node]: joint_5: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.784729379] [piper_ctrl_single_node]: joint_6: 0.0  
[piper_single_ctrl-1] [INFO] [1734573619.786034047] [piper_ctrl_single_node]: vel_all: 5  
[piper_single_ctrl-1] [INFO] [1734573619.786690692] [piper_ctrl_single_node]: gripper_effort: 0.5
```

실제로 조인트1을 제외한 나머지에서는 원위치에서 변화가 없음을 확인할 수 있습니다.



MoveIt2

05

05. MoveIt2

MoveIt2 사용법 가이드

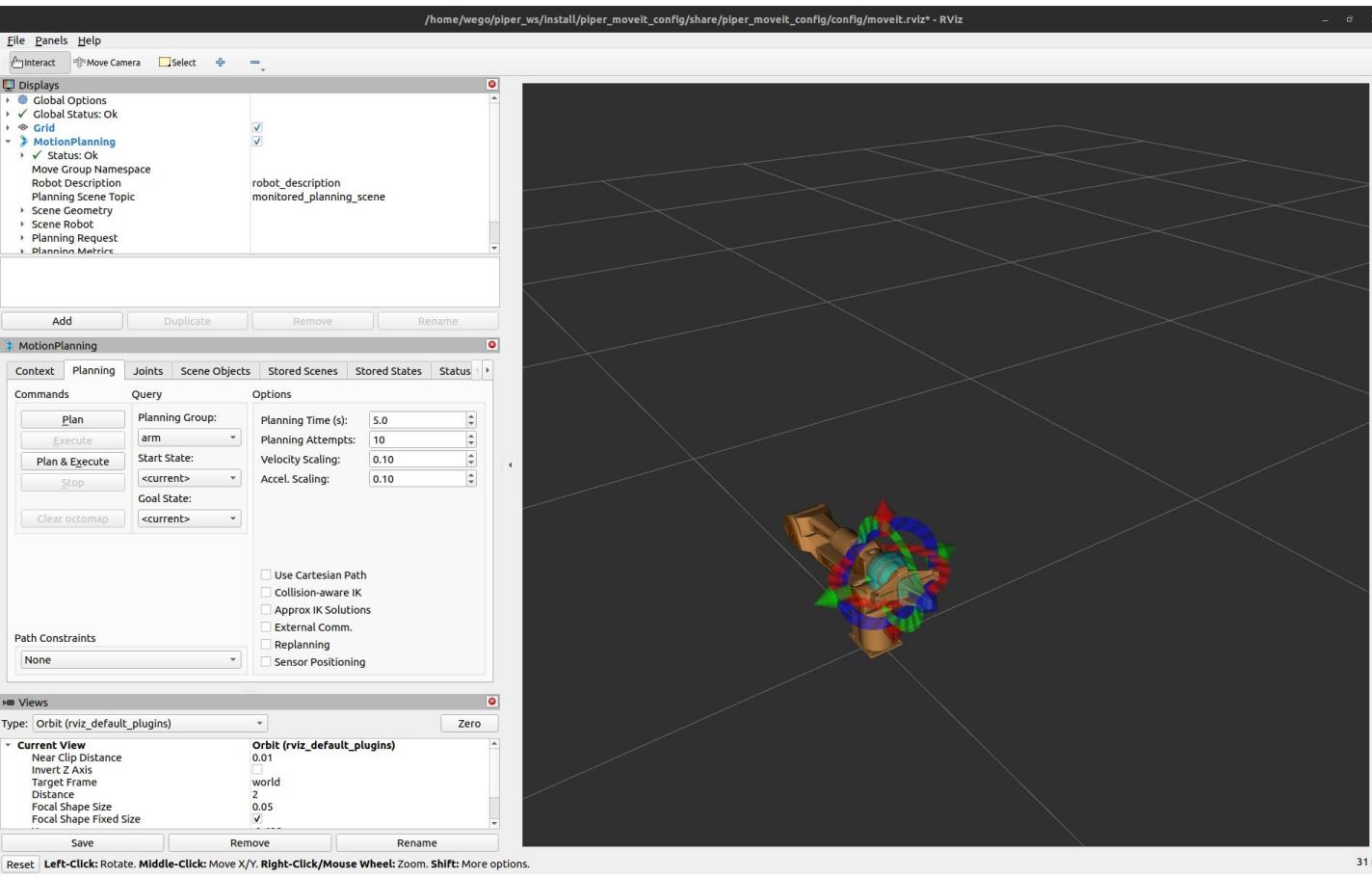
MoveIt을 사용하게 되면 로봇의 경로 계획, 충돌 검사, 역기구학 연산 등을 효율적으로 수행이 가능합니다. 이는 ROS 버전에 따라 MoveIt의 버전도 달라집니다. ROS2를 기반으로 설명을 드리겠습니다.

PiPER MoveIt2 패키지 사용법

`start_single_piper.launch.py` 파일을 먼저 실행시키는 이유는 MoveIt2에서 계산한 조인트 위치값을 받아 실제 PiPER를 구동하기 위함입니다. 이후, 실제 MoveIt을 사용할 때 필요한, `demo.launch.py`로 모션 플래닝을 실행시킵니다.

launch 파일을 실행하려면 다음 명령어를 입력하세요.

```
cd piper_ws
source install/setup.bash
ros2 launch piper start_single_piper.launch.py
ros2 launch piper_moveit demo.launch.py
```





WeGo

본사(기술연구소 및 사무실) : 16914 경기도 용인시 기흥구 구성로 357(청덕동)
용인테크노밸리 B동 513호

기술연구소(서울) : 04799 서울특별시 성동구 성수동2가 280-13, 삼환디지털벤처타워 401호

대표전화 : 031 – 229 – 3553
팩스 : 031 – 229 – 3554

제품문의: go.sales@wego-robotics.com
기술문의: go.support@wego-robotics.com