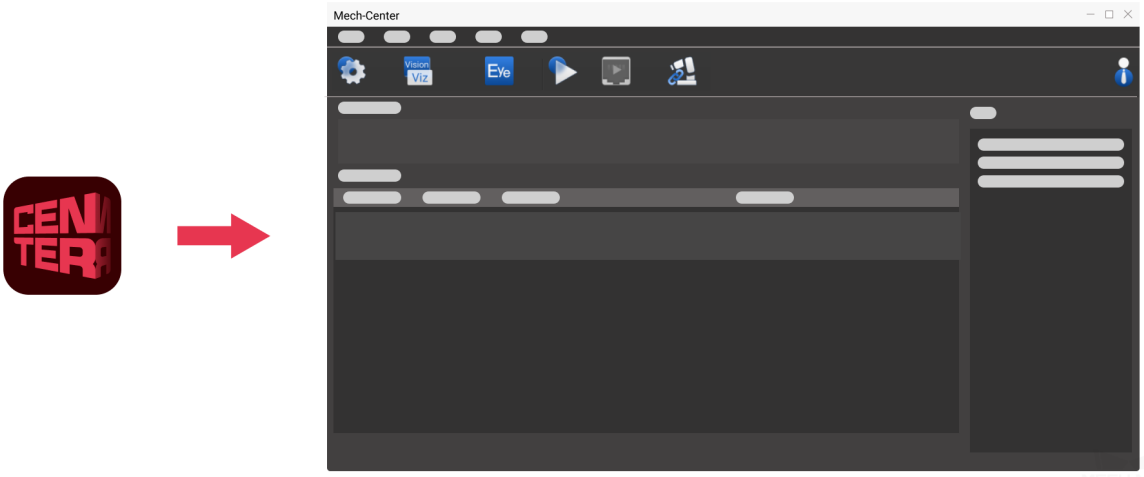
Mech-Center Manual

Mech-Mind

2022 년 11 월 04 일

Contents



Mech-Center 는 본사에서 자체 연구 & 개발한 Mech-Mind 소프트웨어 시스템의 중추와 컨트롤 센터로 소프트웨어의 전반 설정, 프로젝트의 전체 백업 및 복원을 실현할 수 있고 Mech-Viz, Mech-Vision, Mech-Eye Viewer, 로봇, 표준 인터페이스 및 Adapter 의 상태를 직관적으로 확인할 수 있으며 대외 인터페이스 Mech-Interface 의 작동 및 컨트롤 센터 역할을 수행할 수 있습니다.

다음 내용을 통해 Mech-Center **화면 구성 및 각 부분의 기능** 에 대해 알아보세요.

Mech-Center **퀵 스타트**

다음 내용을 통해 Mech-Center **기본 사용 가이드** 에 대해 알아보세요.

Mech-Center **시작하기**

다음 내용을 통해 Mech-Center **대외 인터페이스 서비스** 에 대해 알아보세요.

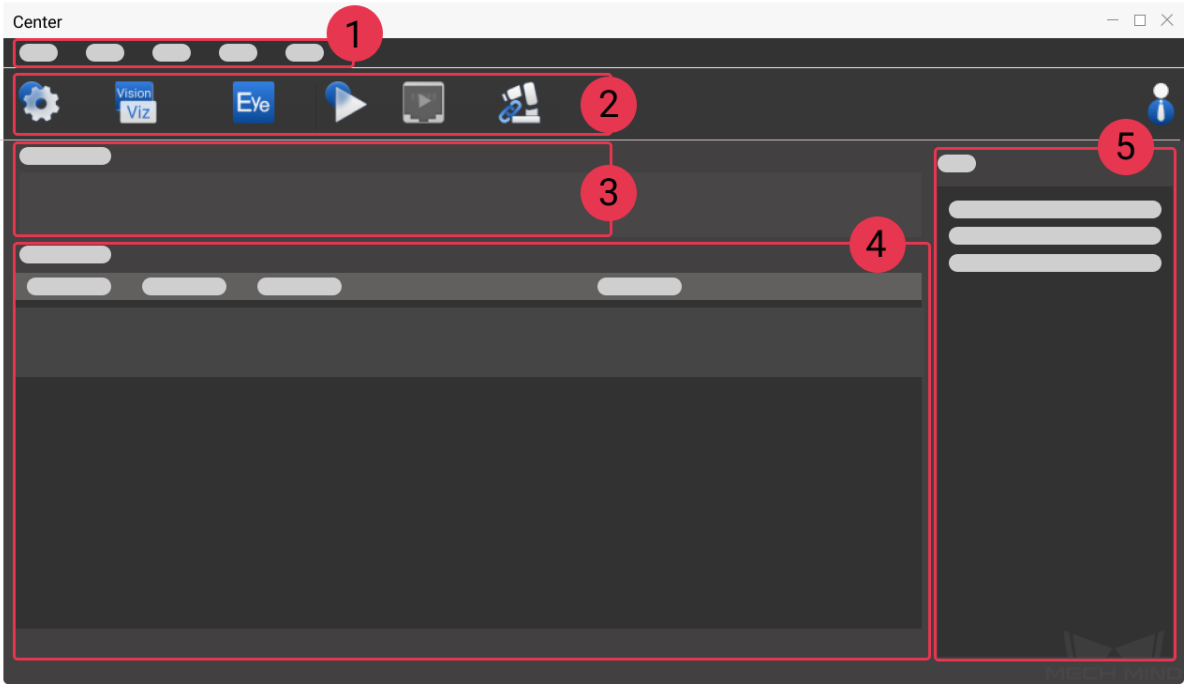
Mech-Interface

CHAPTER 1

Mech-Center 퀵 스타트

Mech-Center 는 당사에서 자체 연구 & 개발한 Mech-Mind 소프트웨어의 중추 및 컨트롤 센터로 소프트웨어들의 전반 설정, 프로젝트 백업 및 복원을 완성할 수 있고 Mech-Viz Mech-Vision Mech-Eye Viewer, 로봇, 표준 인터페이스와 Adapter 의 상태를 직관적으로 확인할 수 있으며 대외 인터페이스 Mech-Interface 의 부팅 및 관리 센터의 역할을 할 수 있습니다.

Mech-Center 의 메인 인터페이스가 주로 다음과 같이 다섯 부분으로 구성됩니다:

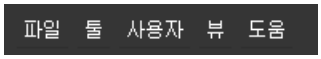


1. **메뉴바** 프로젝트 실행, 사용자 및 뷰 관리, Adapter 생성 및 소프트웨어 정보 확인 등 기능이 있습니다.
2. **툴바** 구성 설정, 소프트웨어 및 인터페이스 실행, 로봇 연결 및 실행 등 버튼이 포함됩니다.

3. **서비스 상태 표시줄** 등록된 소프트웨어, 카메라 및 로봇을 표시합니다.
4. **프로젝트 상태 표시줄** Mech-Viz/Mech-Vision 프로젝트의 실행 상태, 실행 시간 및 다른 상세 정보를 표시합니다.
5. **로그바** 현재 실행 중인 프로젝트 및 서비스의 로그 정보를 실시간으로 표시합니다.

1.1 메뉴바

메뉴바는 파일, 툴, 사용자, 뷰, 도움 등 작업과 관련된 기본 기능을 제공합니다.



1.1.1 파일

옵션	설명	단축키
프로젝트 백업	이 옵션은 Mech-Mind 소프트웨어 시스템의 프로젝트 백업을 생성하는 데 사용됩니다. 자동 로드된 Mech-Viz / Mech-Vision 프로젝트, Mech-Center의 어댑터 프로젝트 및 Mech-Center 의 구성 설정을 저장합니다.	Ctrl+B
복원하기	이 옵션은 선택한 백업에서 프로젝트 및 설정을 복원하는 데 사용됩니다. 자동으로 로드된 프로젝트는 복원 후 대체되므로 프로젝트가 백업되었는지 확인하십시오.	Ctrl+R
프로젝트를 도입하기	이 옵션은 다른 경로에서 프로젝트를 도입할 때 사용됩니다.	Ctrl+I
종료	Mech-Mind 소프트웨어 시스템을 종료합니다.	Ctrl+Q

주의: 오퍼레이터 모드에서는 프로젝트 백업 및 종료 옵션만 사용할 수 있습니다.

1.1.2 툴

옵션	설명
디버그 데이터를 패킹하기	Mech-Mind 소프트웨어 시스템 프로젝트의 디버깅 데이터를 패키징하는 옵션으로 시간, 패킹 경로, 저장 옵션을 선택할 수 있습니다.
Adapter 생성기	이 옵션은 사용자 정의 Adapter 프로그램을 생성하는 데 사용됩니다.
로그 뷰어	이 옵션은 Mech-Viz / Mech-Vision 의 로그 정보를 볼 수 있으며, 로그 파일을 수동으로 선택한 다음 로그 파일을 로드해야 볼 수 있습니다. 또한 이 옵션을 통해 로그 레벨에 따라 해당 로그 내용을 볼 수 있으며 로그를 필터링하거나 찾기 위해 키워드 (정규 표현식, 대소문자를 구분하기, 와일드카드 선택 가능) 입력을 지원하며 타임라인을 슬라이드 하여 이 시간 이후의 로그를 선택할 수도 있습니다.
서비스 상태 표시	이 옵션은 소프트웨어, 로봇 및 다양한 인터페이스와 같은 서비스의 연결 상태를 표시하는 데 사용됩니다.

주의: 오퍼레이더 모드에서는 Adapter 생성기 옵션이 없습니다.

1.1.3 사용자

비밀번호를 수정하기: 관리자 모드에서만 관리자 비밀번호를 수정할 수 있습니다. 오퍼레이더는 이 기능을 사용할 수 없습니다.

주의: IPC 에서 처음으로 Mech-Center 소프트웨어를 설치할 때 비밀번호가 필요합니다. 관리자 모드의 비밀번호는 123456 입니다. 비밀번호를 수정하면 잊어버리지 마시고 잘 기억하십시오.

1.1.4 뷰

로그: 메인 인터페이스에서 로그바 표시 여부를 선택할 수 있습니다.

1.1.5 도움

옵션	설명	단축키
사용자 안내서	이 옵션은 인터페이스 사용 안내서의 디렉터리를 빠르게 여는 데 쓰입니다.	F1
업데이트 설명	이 옵션은 각 버전의 새로운 기능과 수정 사항을 설명하는 데 사용됩니다.	없음
버전에 관하여	이 옵션은 소프트웨어 버전 및 저작권 정보를 표시합니다.	없음

1.2 툴바

Mech-Center 툴바에는 구성 설정, Mech-Viz/Vision 실행, 카메라 뷰어 실행, 실행, 인터페이스 실행, 마스터 컨트롤 로봇 및 관리자의 7 개 버튼이 있습니다.

1.2.1 구성 설정

구성 설정은 Mech-Center 의 기본 설정, 각 소프트웨어의 경로를 설정, 자동으로 로드된 프로젝트 경로 확인 및 외부 서비스 선택 등 기능을 실현할 수 있습니다.

선호 설정

선호 설정 옵션은 아래 이미지와 같이 소프트웨어의 전역 설정을 개인화하는 데 사용됩니다. 사용자는 선호에 따라 관련 설정을 할 수 있습니다. 예를 들면 소프트웨어 테마 색상 전환, Mech-Center 소프트웨어 인터페이스 언어 전환 (현재 소프트웨어는 중국어, 영어, 일본어, 한국어까지 4 개 언어 지원), 로그 변경 및 저장 등이 있습니다.

- **실행 시 콘솔을 숨기기** 를 선택하면 Mech-Center 소프트웨어가 실행될 때 콘솔이 숨겨집니다.
- *PC* **부팅 시** *Mech-Center* **자동으로 실행하기** 를 선택하면 Mech-Center 소프트웨어 자동 시작을 실현할 수 있습니다.
- *Mech-Viz/Mech-Vision/Mech-Interface* **자동으로 실행하기** *(***있으면***)* 를 선택하면, Mech-Viz/Mech-Vision/Mech-Interface 는 Mech-Center 소프트웨어를 시작한 후 자동으로 시작할 수도 있습니다.
- *Mech-Viz/Mech-Vision* **실행한 후 트레이로 최소화하기** 를 선택한 후 실행된 Mech-Viz/Mech-Vision 소프트웨어가 바탕 화면 상태 표시줄의 트레이에 숨겨집니다.

주의: 변경 사항을 저장하고 Mech-Center 를 리부팅해야 설정이 적용됩니다.

Mech-Viz

Mech-Viz 옵션은 Mech-Viz 소프트웨어의 열수 있는 경로를 설정하고 자동으로 로드된 프로젝트 경로를 표시하는 데 사용됩니다. 아래 그림과 같습니다.

왼쪽의 *Mech-Viz* 를 클릭하여 설정 인터페이스로 들어가면 *Mech-Viz* **실행** 이 기본적으로 선택되어 있을 것입니다.

프로그램 경로 뒤의 ⋯를 클릭하고 Mech-Viz 설치 디렉터리에서 mmind_viz.exe 파일을 선택하여 경로 선택을 완료합니다.

Mech-Viz 에서 **현재 프로젝트를 자동으로 로드하기** 를 선택하면 프로젝트 경로가 자동으로 로드되며 수동으로 경로를 추가할 필요가 없습니다.

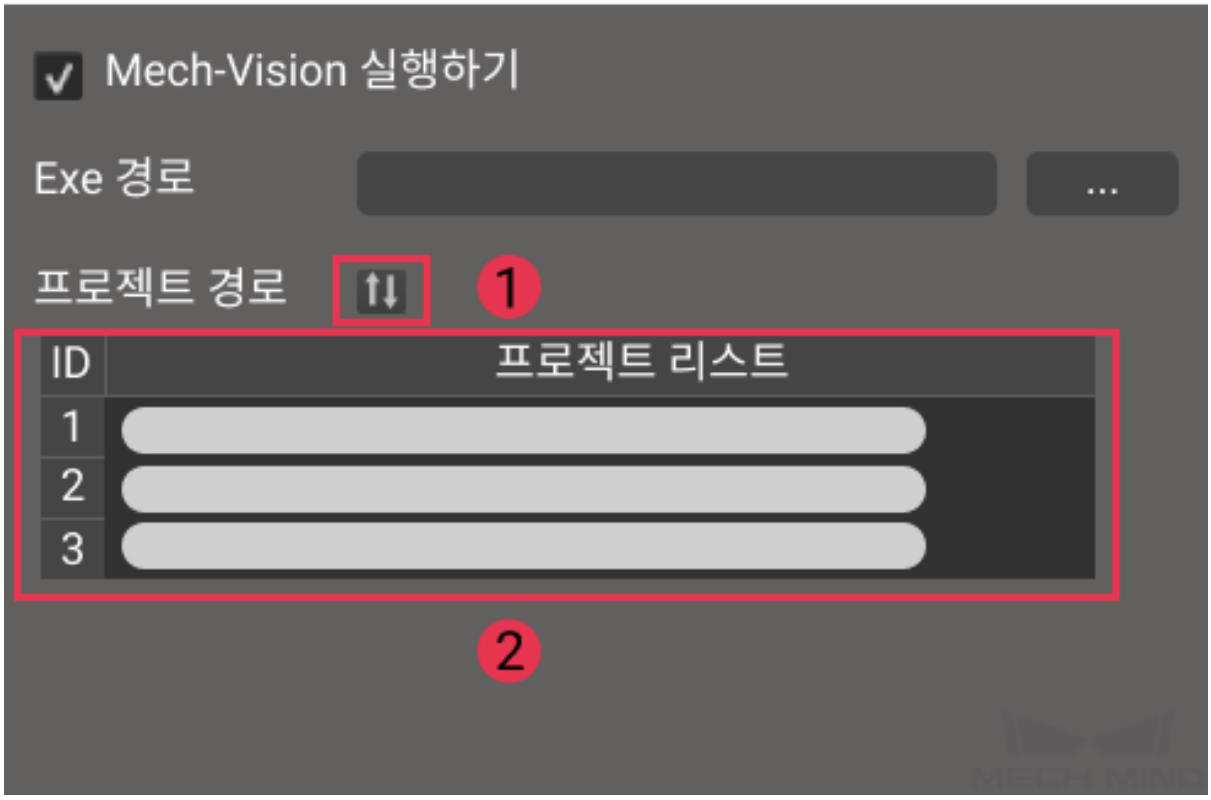
참고: **시뮬레이션** 을 선택하면 Mech-Viz 소프트웨어는 실제 로봇을 움직이지 않고 시뮬레이션만 합니다.

Mech-Vision 、Mech-Eye Viewer

Mech-Vision, Mech-Eye Viewer 구성 설정 방식은 Mech-Viz 와 유사합니다.

인터페이스에 해당하는 Mech-Vision 프로젝트 ID 를 조정하려면 구성 설정의 Mech-Vision 페이지에서 관련 설정을 할 수 있습니다.

먼저 Mech-Vision 소프트웨어에서 **현재 프로젝트를 자동으로 로드하기** 를 선택한 다음 Mech-Center 구성 설정의 Mech-Vision 페이지에서  를 클릭하여 Mech-Vision 프로젝트 경로 리스트를 동기화해야 합니다. 마지막으로 마우스 왼쪽 버튼으로 프로젝트를 길게 누르고 위 또는 아래로 끌어 프로젝트 경로의 순서를 조정합니다.



Robot Server

Robot Server 옵션은 Mech-Viz 소프트웨어의 마스터 컨트롤을 실현하도록 로봇에 적용하는 데 사용됩니다. 아래 그림과 같습니다.

왼쪽의 *Robot Server* 를 클릭하여 설정 인터페이스로 들어갑니다. *Robot Server* **실행** 은 기본적으로 선택되어 있으며 Mech-Center 와 함께 설치된 Robot Server 경로를 자동으로 로드합니다.

주의: Mech-Viz 프로젝트를 성공적으로 로드하면 Mech-Viz 프로젝트의 로봇 유형이 자동으로 표시되며 입력할 필요가 없습니다. 로봇 IP 는 실제 프로젝트의 설정에 따라 입력해야 하며 올바른지 확인해야 합니다.

Mech-Interface

Mech-Interface 는 외부 서비스와의 통신을 위한 통합 외부 인터페이스입니다.

1.2.2 Mech-Viz、Mech-Vision 실행하기

Mech-Viz/Mech-Vision 소프트웨어가 성공적으로 실행되면 소프트웨어 아이콘과 해당 버전이 서비스 상태 표시줄에 표시됩니다.

주의:

1. 버전 호환성 체크는 프로젝트 실행 시 진행되며, Mech-Viz, Mech-Vision, Mech-Center V1.4.0 및 이후 버전에서 사용할 것을 권장합니다.
2. Mech-Center 가 다른 두 소프트웨어 버전이 1.4.0 보다 낮은 것을 발견하면 버전이 1.4.0 보다 높아야 한다는 메시지가 표시되고 **실행** 을 클릭하면 오류 알림이 나타납니다.

1.2.3 카메라 뷰어 실행

Mech-Eye Viewer 소프트웨어가 성공적으로 실행되면 소프트웨어 아이콘이 바탕 화면 상태 표시줄에 나타납니다.

1.2.4 실행

Mech-Viz 및 Mech-Vision 에 로드된 프로젝트를 실행합니다.

1.2.5 인터페이스 실행

Mech-Interface 실행하기.

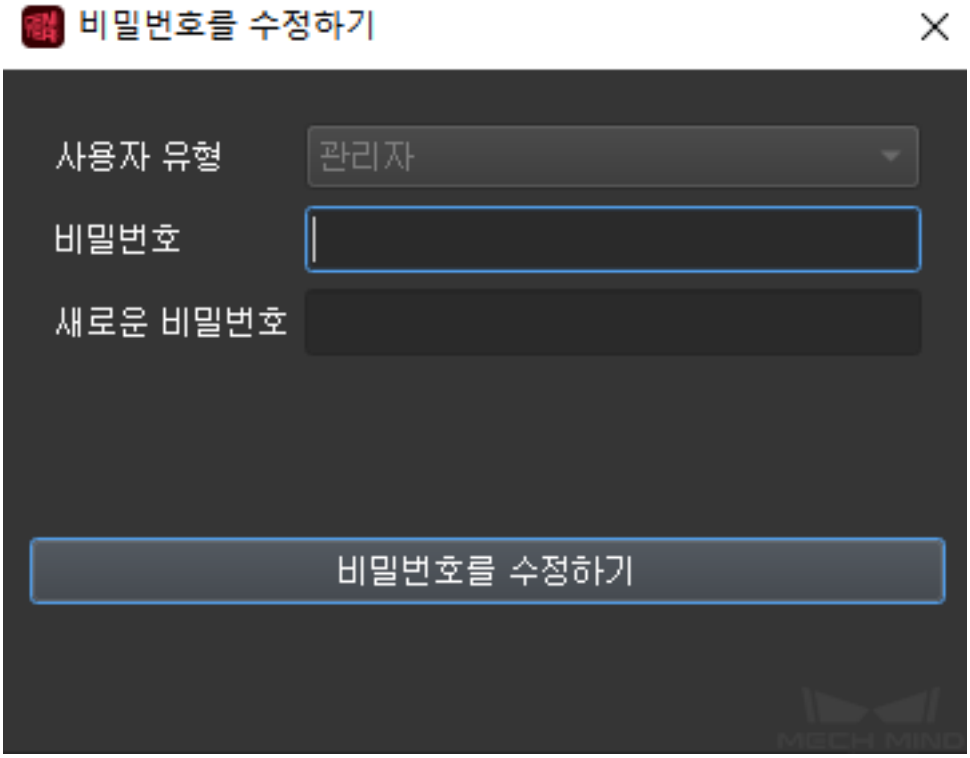
1.2.6 로봇 마스터 컨트롤

실제 로봇이 성공적으로 연결되면 로봇 아이콘과 해당 모델이 서비스 상태 표시줄에 표시됩니다.

1.2.7 관리자

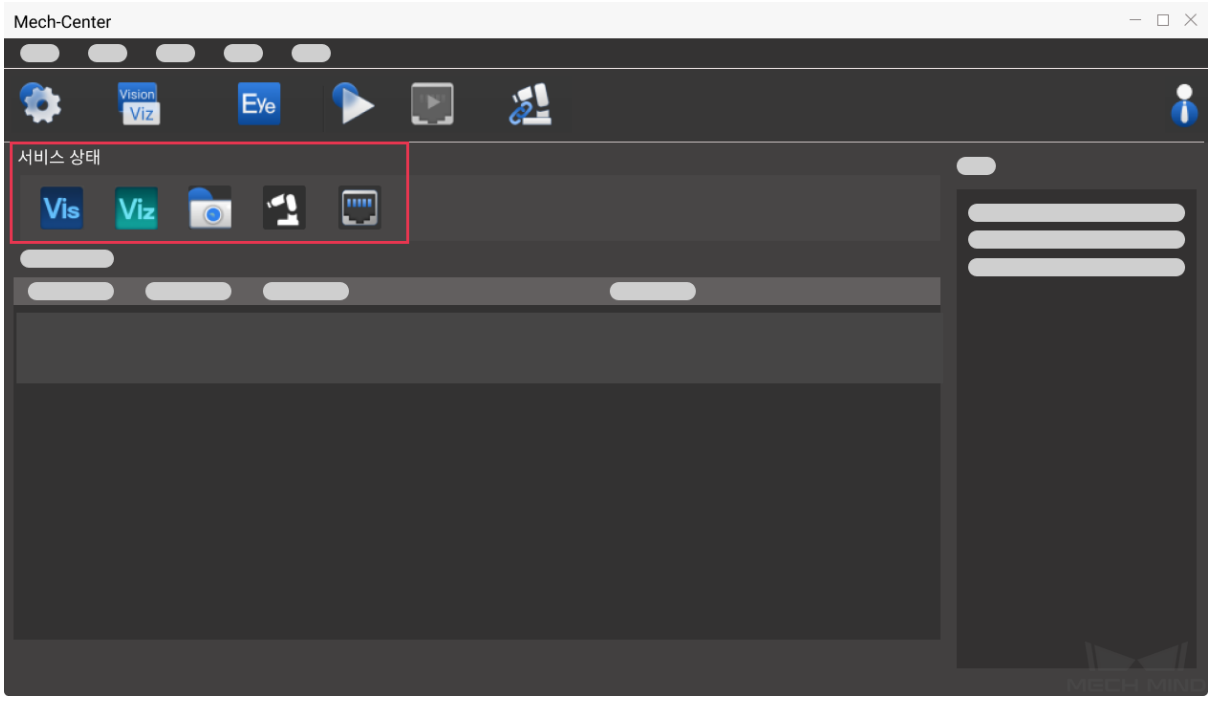
Mech-Center 에는 관리자와 오퍼레이터 이 두 가지 모드가 있으며 일반적으로 관리자 모드로 선택되어 있습니다. 오퍼레이터 모드를 사용하는 경우 프로젝트 편집 및 구성 수정이 불가능하며 아이콘이 오퍼레이터 모드로 표시됩니다. 모드를 전환하려면 아이콘을 클릭하고 로그인 대화 상자에서 사용자 유형을 선택한 다음 **로그인** 을 클릭하십시오.

관리자 모드는 로그인 비밀번호가 필요하며, 비밀번호 수정이 필요한 경우 메뉴바의 **사용자** → **비밀번호 변경** 에서 수정할 수 있습니다.



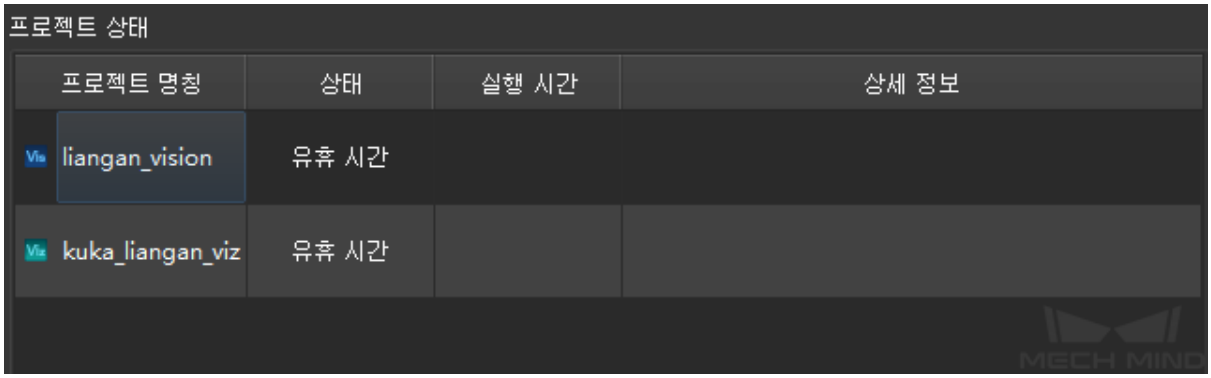
1.3 서비스 상태 표시줄

Mech-Center 가 실행되는 동안 서비스 상태 표시줄은 활성화된 소프트웨어, 카메라, 로봇 및 인터페이스 서비스를 표시하고 서비스 상태 표시줄의 소프트웨어 아이콘을 클릭하여 해당 창을 열 수 있습니다.



1.4 프로젝트 상태 표시줄

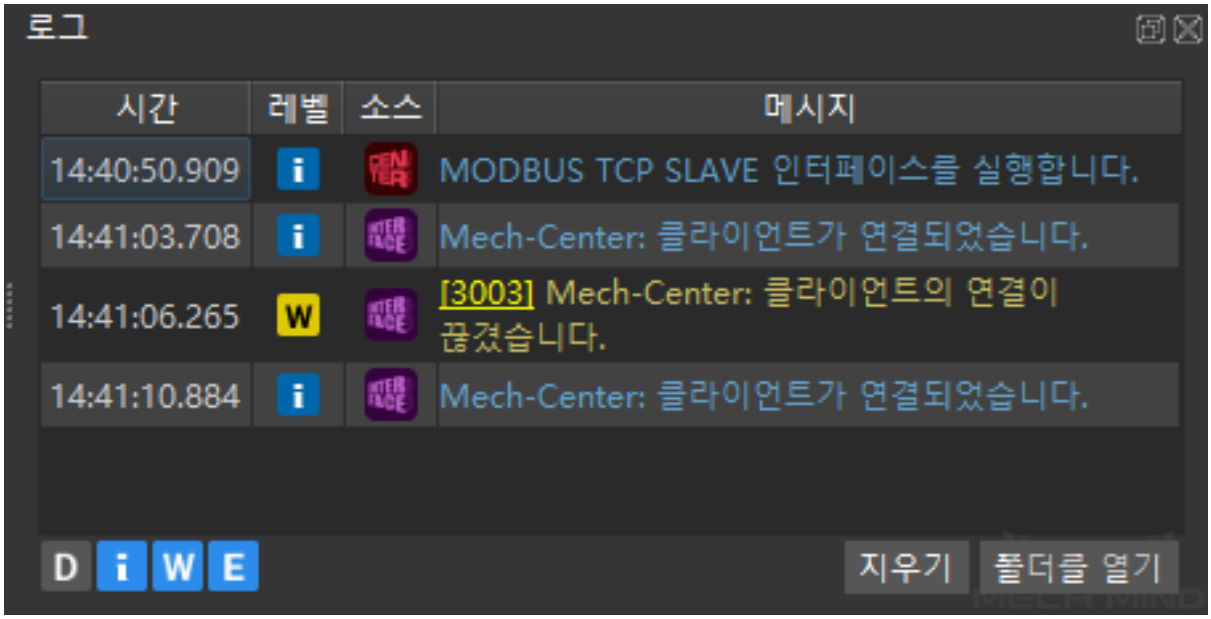
프로젝트 상태 표시줄은 Mech-Viz / Mech-Vision 프로젝트 명칭, 상태, 실행 시간 및 세부 정보를 표시할 수 있습니다. 일반적으로 오른쪽의 로그바와 결합되어 프로젝트 실행 상태를 봅니다.



옵션	설명
프로젝트 명칭	Mech-Viz / Mech-Vision 프로젝트 명칭.
상태	프로젝트의 현재 상태 (유휴 또는 실행 중).
실행 시간	프로젝트 실행 시작부터 실행 종료까지 걸리는 시간입니다.
세부 정보	프로젝트 실행 시간과 해당 상태를 기록하고 프로젝트 실행 과정에서 오류가 나타난 경우 오류 정보도 여기에 기록됩니다.

1.5 로그바

현재 실행 중인 프로젝트 및 서비스의 로그 정보를 실시간으로 표시합니다.



옵션	설명
D i W E	로그 레벨을 의미합니다. 그중에 D 는 디버깅 정보, I 는 정상 정보, W 는 경고, E 는 오류 정보입니다. 구체적인 레벨을 클릭하여 로그 내용에 대해 필터링할 수 있고 동시에 여러 가지 레벨을 선택할 수 도 있습니다.
클리어	모든 로그 내용을 지웁니다.
폴더를 열기	Mech-Center 설치 경로에 있는 `logs` 폴더를 엽니다.

팁: 오류 메시지 앞의 오류 코드를 클릭하면 자세한 정보를 볼 수 있는 온라인 매뉴얼로 이동합니다.

CHAPTER 2

Mech-Center 시작하기

2.1 소프트웨어 설정

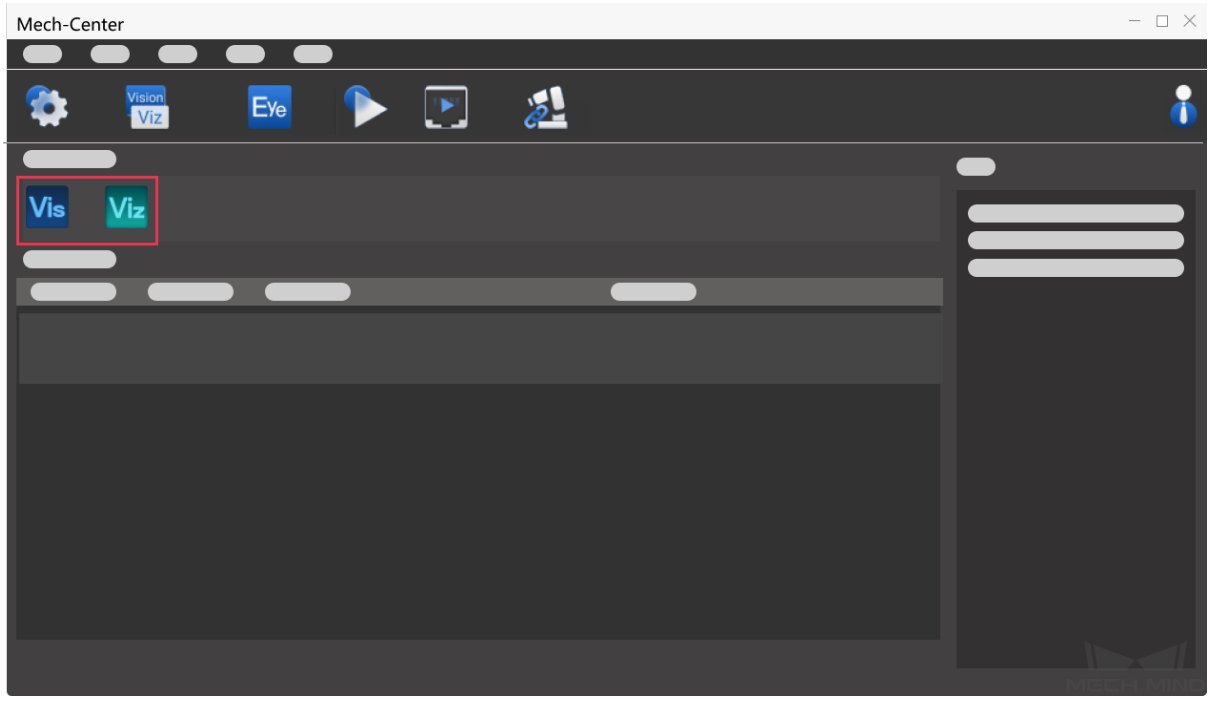
Mech-Center 를 열어**툴바** 의 **구성 설정**  을 클릭하여 아래 그림과 같이 구성 설정의 화면으로 들어갑니다. 이 화면에서 경로, 파라미터 등 사항을 설정하고 저장하면 됩니다. 상세한 내용은**구성 설정** 을 참조하세요.

힌트:

- 실제로 사용할 때의 상황에 따라 설정하십시오.
- Mech-Vision Mech-Viz 및 Mech-Eye Viewer 등 소프트웨어가 설치된 후 구성 설정에서의 경로는 자동으로 입력될 것이며 수동으로 입력할 필요가 없습니다.

2.2 프로젝트를 열기

툴바 에 있는 **Viz/Vision 실행** 을 클릭하여 Mech-Viz 및 Mech-Vision 소프트웨어를 부팅할 수 있으며 서비스 상태에서 해당 소프트웨어의 아이콘이 나타날 것입니다.



2.2.1 Mech-Viz 프로젝트를 열기

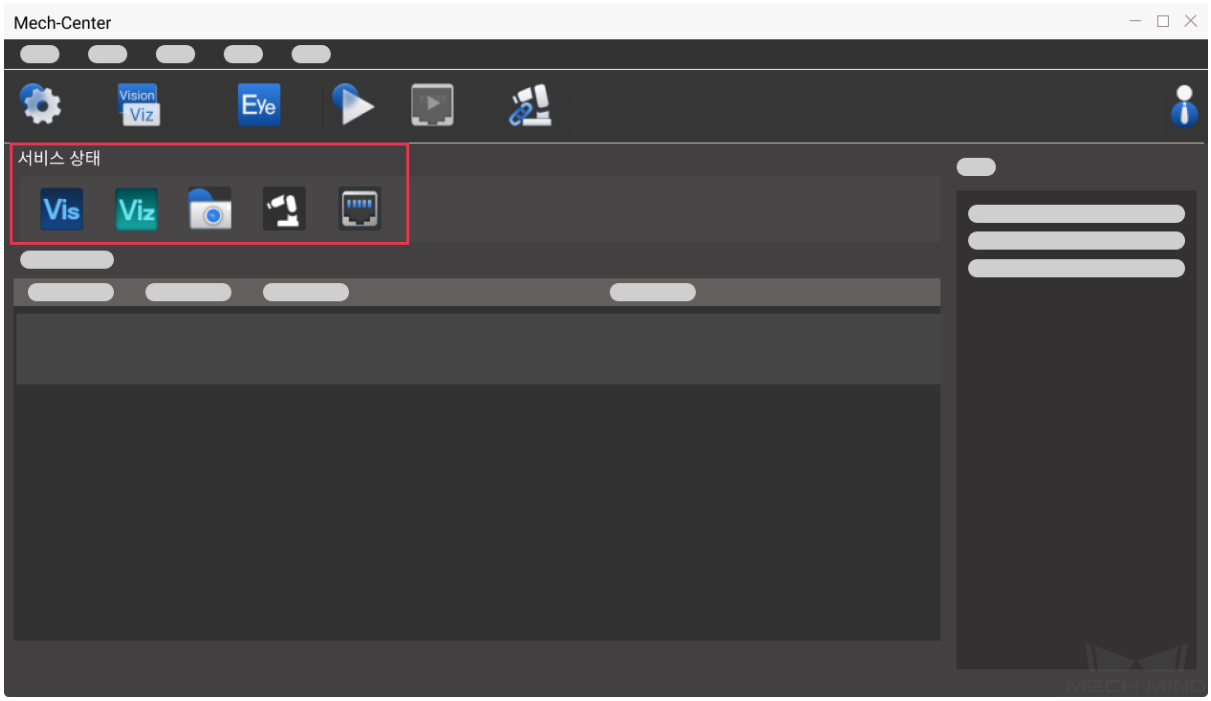
1. 소프트웨어 인터페이스로 들어가기:  를 클릭하여 Mech-Viz 의 메인 인터페이스로 들어갑니다.
2. 프로젝트 구축: 프로젝트를 새로 만들거나 기존 프로젝트를 엽니다.
3. **현재 프로젝트를 자동으로 로드하기** 를 선택하기: Mech-Viz 메인 인터페이스의 툴바에서 선택하십시오.
4. 프로젝트 상태를 확인하기: 프로젝트 상태 표시줄에서 현재 프로젝트의 실행 상태가 표시됩니다.

힌트:

- **현재 프로젝트를 자동으로 로드하기** 를 선택한 다음에 **프로젝트 경로** 가 자동으로 **구성 설정** 의 해당 위치에 입력될 것입니다.

2.2.2 Mech-Vision 프로젝트를 열기

1. 소프트웨어 인터페이스로 들어가기:  를 클릭하여 Mech-Vision 의 메인 인터페이스로 들어갑니다.
2. 프로젝트를 만들기: typical_applications 내용을 참조하여 새로운 프로젝트를 만들거나 기존 프로젝트를 엽니다.
3. **현재 프로젝트를 자동으로 로드하기** 를 선택하기: **프로젝트 리스트** 에서 원하는 프로젝트를 선택하여 마우스 오른쪽 버튼으로 클릭하며 **현재 프로젝트를 자동으로 로드하기** 를 선택합니다.
4. 프로젝트 경로를 로드하기: **구성 설정** → *Mech-Vision* 을 클릭하고  를 클릭한 다음에 **저장** 을 클릭하여 프로젝트의 경로를 로드할 수 있습니다.
5. 프로젝트 상태를 확인하기: 아래의 그림과 같이 프로젝트 상태 표시줄에서 현재 프로젝트의 실행 상태가 표시됩니다.



2.2.3 카메라 설정을 변경하기

프로젝트 실행 과정에서 카메라의 설정 사항을 확인하거나 변경할 수요가 있으면툴바 의 **Mech-Eye Viewer 실행**  을 클릭하십시오. 카메라 뷰어를 실행한 후 해당 소프트웨어의 아이콘이 서비스 표시줄에 나올 것입니다.

2.3 로봇 연결

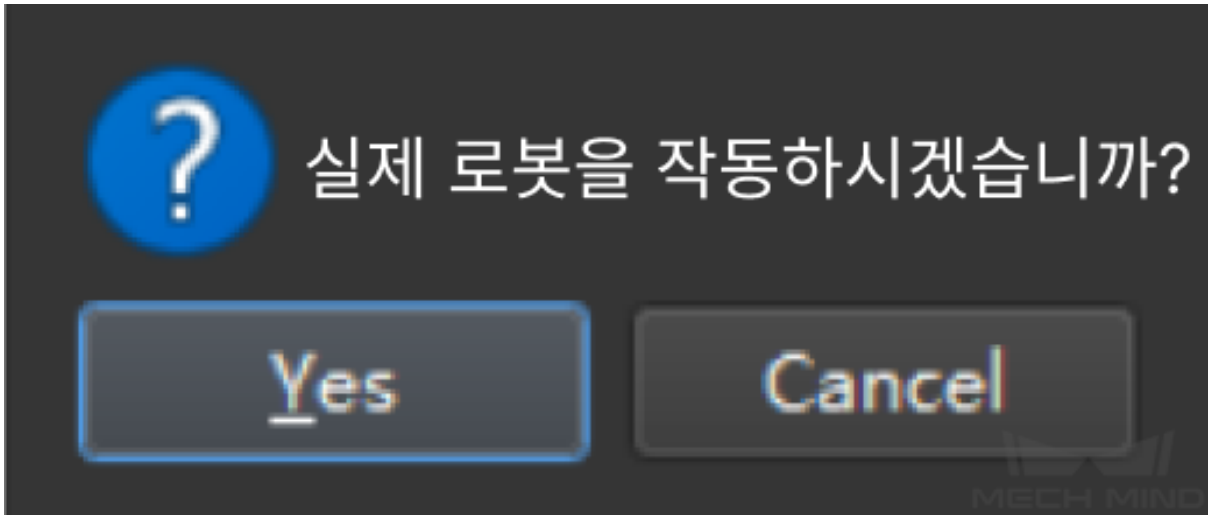
작업 현장에서 사용되는 실제 로봇이 UR 가 아닌 경우 로봇 프로그램 복제를 해야 합니다. 상세 정보는 robot_integrations 내용을 참조하세요. 툴바 에 있는 **로봇 컨트롤**  를 클릭하고 성공적으로 실행된 다음에 해당 로봇의 아이콘이 서비스 표시줄에 나올 것입니다.

Mech-Interface 를 사용하는 프로젝트에 대해서는툴바 에 있는 **인터페이스 실행**  를 클릭하십시오. 인터페이스가 실행되면 해당 아이콘이 서비스 표시줄에 나올 것입니다.

2.4 프로젝트 실행

힌트: 실제 로봇을 실행하려면 **구성 설정** 의 Mech-Viz 화면에서 **시뮬레이션 실행** 을 언체크하십시오.

툴바에 있는 **실행**  를 클릭하여 아래 그림과 같이 확인하기 위한 팝업창이 나올 것입니다 *Yes* 를 클릭하여 로드한 프로젝트를 실행할 수 있습니다. 이때 프로젝트 상태 표시줄에서 실행 상태를 확인할 수 있습니다.



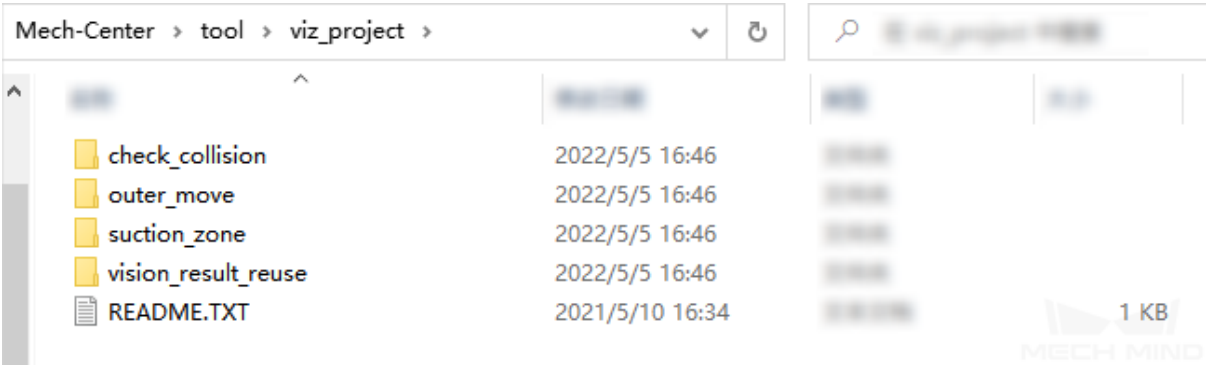
주의:

- 프로젝트 로드 과정이 끝난 후에야 실행할 수 있습니다.

  Mech-Viz 프로젝트를 열 때 사용자가 **소프트웨어 호환성 설명** 을 무시하고 Mech-Center 에서 직접 **실행** 버튼을 클릭하면 Mech-Center 로그에"Mech-Viz 에서 프로젝트를 로드하는 중입니다. Mech-Viz 프로젝트를 시작할 수 없습니다: XXX" 의 메시지를 출력합니다. 이 때 사용자는 소프트웨어 호환성 설명을 처리해야 합니다. 즉, 소프트웨어 호환성 설명에서 *Yes* 버튼을 클릭하고 Mech-Center 실행 버튼을 복원하면 실행할 수 있습니다.
- 로봇을 실행할 때 안전을 반드시 주의해야 합니다. 비상 시 티칭 머신에 있는 긴급 정지 버튼을 누르십시오.

2.5 표준 인터페이스는 Mech-Viz 샘플 프로젝트 사용

표준 인터페이스에 대한 일부 일반적인 응용 프로그램 프로젝트 템플릿은 Mech-Center 설치 디렉터리 (tool/viz_project) 폴더에 저장됩니다.



check_collision

비전 가이드 피킹의 경로 계획 및 충돌 감지에 사용됨

outer_move

로봇이 외부 클라이언트에서 전달된 포즈로 이동해야 하는 시나리오에 사용됨

suction_zone

DO 신호 리스트를 통해 여러 빨판 파티션 (또는 어레이 그리퍼) 사용에 사용됨

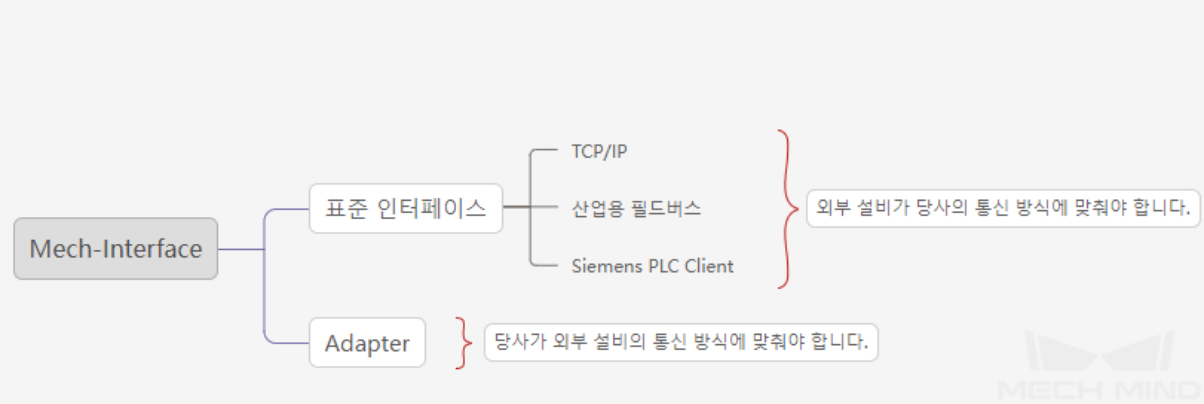
vision_result_reuse

비전 가이드 피킹에서 경로 계획 및 충돌 감지를 위해 동일한 반환된 비전 결과를 여러 번 사용해야 하는 시나리오에 사용됩니다.

CHAPTER 3

Mech-Interface

외부 통신과의 브릿지 역할을 하는 Mech-Interface 는 Mech-Mind 소프트웨어 시스템의 외부 인터페이스 서비스로서 **표준 인터페이스** 및 **Adapter** 를 포함하며 외부 정보를 수신하고 시스템 내부 정보를 보내는 기능을 실현할 수 있습니다.



이 부분에서 주로 다음과 같은 내용을 포함합니다.

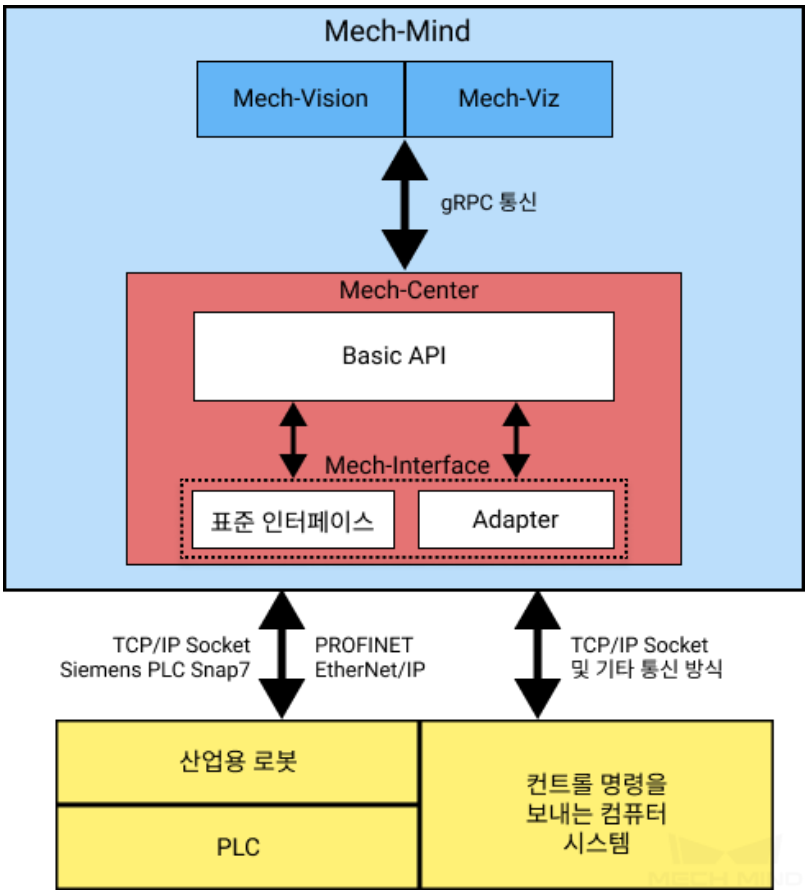
- **개요** Mech-Interface 두 가지 통신 방식의 매커니즘과 응용 시나리오에 관한 소개입니다.
- **표준 인터페이스** 표준 인터페이스의 설정과 배포에 관한 소개입니다.
- **표준 인터페이스 개발자 참고 매뉴얼** 표준 인터페이스의 통신 프로토콜, 명령어 세트, 오류 코드와 오류 원인 분석에 관한 소개입니다.
- *Adapter* Adapter 의 퀵 가이드, Adapter 생성기 사용 안내서와 Adapter 코딩 가이드에 관한 소개입니다.

3.1 개요

이 부분에서 주로 Mech-Interface 의 두 가지 통신 매커니즘, 표준 인터페이스와 Adapter 의 비교 및 응용 시나리오를 소개하고자 합니다.

3.1.1 통신 매커니즘

Mech-Interface 의 두 가지 통신 매커니즘은 아래 그림과 같습니다.



- Adapter 는 외부 통신 장치 (산업용 로봇, 컨트롤 명령을 전송하는 컴퓨터, PLC) 와 Mech-Vision, Mech-Viz 소프트웨어를 연결시키는 Python 적응 프로그램으로 내부적으로 Mech-Vision, Mech-Viz 소프트웨어와 통신하며 외부적으로 Python 을 통해 실현할 수 있는 모든 통신 프로토콜을 사용해 외부 통신 장치와 통신합니다.
- 표준 인터페이스는 메크마인드 로보틱스에서 제공한 완전한 Adapter 프로그램으로 다양한 통신 프로토콜을 지원하며 기능이 강한 컨트롤 명령어 세트와 오류 경보 시스템을 갖추어 대부분 고객의 수요를 충족할 수 있습니다.

3.1.2 표준 인터페이스와 Adapter 의 비교

표준 인터페이스와 Adapter 는 모두 외부 통신 장치와 Mech-Vision, Mech-Viz 소프트웨어를 연결하는 적응 프로그램입니다. 그 중에 표준 인터페이스는 메크마인드 로보틱스에서 제공한 고정된 Adapter 프로그램이고 이차개발을 지원하지 않습니다.

내부적으로 Basic API 를 호출함으로써 Mech-Vision, Mech-Viz 소프트웨어와 통신하고 외부적으로 특정한 통신 프로토콜을 통해 외부 장치와 통신합니다.

표준 인터페이스와 Adapter 의 상세한 비교 결과는 아래 도표를 참조할 수 있습니다.

비교	**표준 인터페이스**	**Adapter**
내부적 통신	모두 Basic API 를 호출함으로써 Mech-Vision, Mech-Viz 소프트웨어와 통신합니다.	
외부적 통신 프로토콜	표준 인터페이스는 아래 통신 프로토콜을 사용하여 외부 장치와 통신하는 것만 지원합니다. - TCP/IP Socket - Siemens PLC Snap7 - PROFINET - EtherNet/IP	Adapter 는 Python 을 통해 실현할 수 있는 모든 통신 프로토콜을 사용해 외부 통신 장치와 통신하는 것을 지원합니다.
기능	표준 인터페이스는 비전 결과를 제공하는 것만 지원합니다.	Adapter 는 비전과 관련된 기능을 뿐만 아니라 인터페이스, 데이터 베이스 와 주문 시스템 등 Python 의 관련 기능도 지원합니다.
배포 난이도	표준 인터페이스를 사용하기 손쉽고 신속하게 배포를 완성할 수 있습니다.	Adapter 를 사용하면 자체적으로 작성해야 하므로 긴 시간과 높은 인건비가 필요합니다.
확장 가능성	기능 확장을 지원하지 않습니다.	더 많은 통신 프로토콜을 확장할 수 있으며 더 다양한 기능을 실현할 수 있습니다.

3.1.3 사용 설명

실제 응용 시나리오에서 일반적으로 외부 통신 대상, 사용하는 통신 프로토콜과 프로젝트가 필요한 통신 기능 등 요소에 근거하여 어떤 유형의 Mech-Interface 를 사용할지를 결정해야 합니다.

표준 인터페이스와 Adapter 가 지원하는 일반적인 통신 대상 및 프로토콜은 아래 도표와 같습니다.

통신 대상	통신 프로토콜	Mech-Interface 유형	설명
로봇	TCP/IP Socket	표준 인터페이스	Mech-Interface 는 TCP/IP Socket 의 서버 역할을 합니다.
	PROFINET		Mech-Interface 는 PROFINET 의 수동국 역할을 합니다.
	EtherNet/IP		Mech-Interface 는 EtherNet/IP 의 어댑터 (수동국) 역할을 합니다.
	Modbus TCP	Adapter	현재 Adapter 만 해당 통신 프로토콜을 지원합니다.
컨트롤 명령을 전송하는 컴퓨터	HTTP	Adapter	통합 프로젝트에 적합하고 로봇 풀 컨트롤 (Full Control) 방식을 사용합니다.
	WebSocket		
	TCP/IP Socket	표준 인터페이스	Mech-Interface 는 TCP/IP Socket 의 서버 역할을 합니다.
PLC	TCP/IP Socket	표준 인터페이스	Mech-Interface 는 TCP/IP Socket 의 서버 역할을 합니다.
	Siemens PLC Snap7		Mech-Interface 는 Siemens PLC Snap7 의 마스터 지국 역할을 합니다.
	PROFINET		Mech-Interface 는 PROFINET 의 수동국 역할을 합니다.
	EtherNet/IP		Mech-Interface 는 EtherNet/IP 의 어댑터 (수동국) 역할을 합니다.
	Modbus TCP	Adapter	현재 Adapter 만 해당 통신 프로토콜을 지원합니다.
	미쓰비시 PLC MC	Adapter	현재 Adapter 만 해당 통신 프로토콜을 지원합니다.

힌트:

- Mech-Interface 를 사용하여 외부 장치와 통신해야 할 때 표준 인터페이스가 프로젝트의 수요를 충분히 충족할 수 있으면 표준 인터페이스를 사용하는 것을 권장하고 표준 인터페이스가 프로젝트의 수요를 충족할 수 없으면 (예를 들어 외부 통신 장치는 표준 인터페이스가 지원하지 않는 통신 프로토콜을 사용하거나 프로젝트는 표준 인터페이스가 지원하지 않는 기능이 필요한 경우) Adapter 를 사용하십시오.
- 표준 인터페이스가 지원하는 기능은“**표준 인터페이스 개발자 참고 매뉴얼**”내용을 참조하십시오.
- Adapter 가 지원하는 기능은“*Adapter* **기능**”내용을 참조하십시오.

표준 인터페이스와 Adapter 에 관한 더 많은 정보를 얻으려면 아래 내용을 읽으십시오.

- **표준 인터페이스**
- *Adapter*

3.2 표준 인터페이스

사용자가 포즈만 필요하고 Mech-Mind 소프트웨어 시스템을 사용하여 로봇의 움직임을 제어할 필요가 없는 경우 표준 인터페이스를 사용하여 데이터를 전송할 수 있습니다. 표준 인터페이스는 포즈 및 데이터 전송과 같은 가장 기본적인 기능만 제공합니다. 더 많은 기능을 원하시면*Adapter* 를 사용하여 사용자가 직접 지정할 수 있습니다. 표준 인터페이스는 Mech-Center 소프트웨어에 통합되며 생성할 필요가 없습니다. 사용을 원하시면 소프트웨어에서 직접 서비스를 열 수 있습니다.

3.2.1 Mech-Interface 실행

구성 설정 → *Mech-Interface* 에서 *Mech-Interface* **실행하기** 를 선택하고 인터페이스의 유형은 **표준 인터페이스** 를 선택합니다.

3.2.2 인터페이스 옵션 및 호스트 주소

사용자는 필요에 따라 외부 서비스 유형을 선택할 수 있습니다.

Siemens PLC Client Siemens PLC Client 는 호스트 IP 주소, PLC 슬롯 번호 및 DB 블록 번호를 설정해야 합니다.

TCP Server TCP Server 는 프로토콜 형식 (ASCII 또는 HEX) 을 선택해야 합니다. HEX 형식에서는 빅 엔디안 및 리틀 엔디안 을 선택해야 합니다.

TCP Server 는 실제 상황에 따라 포트 번호를 설정해야 하며 기본 포트는 50000 입니다.

사용자는 로봇에 따라 사용 가능한 통신 형식을 선택하여 인터페이스 프로그램을 작성할 수 있습니다.

로봇 유형		표준 인터페이스 프로그램 샘플
산업용 로봇	ABB	HEX 프로그램 패키지
	FANUC	HEX 프로그램 패키지
	KUKA	HEX 프로그램 패키지
	YASKAWA	ASCII 프로그램 패키지
	KAWASAKI	ASCII 프로그램 패키지
협동 로봇	FANUC CRX	Plugin 플러그인 (HEX)
	UR	URCap 플러그인 (ASCII)
	TM	plug and play 플로그인 (ASCII)
기타		Mech-Center 에서 샘플 프로그램을 제공하지 않았기 때문에 사용자는 로봇 측 프로그램을 작성해야 합니다.

팁: 로봇 인터페이스 샘플 프로그램 폴더와 실행 파일은 소프트웨어의 설치 디렉터리에 있는 Robot_Interface 폴더에 있습니다.

PROFINET PROFINET 로봇 모델을 설정해야 합니다.

EtherNet/IP EtherNet/IP 로봇의 모델을 설정해야 합니다.

MODBUS TCP SLAVE Modbus TCP SLAVE 는 슬레이브 IP, 포트 번호, 장치 주소 및 바이트 순서를 설정해야 합니다 (사용자는 일치하는 바이트 순서를 선택하기 위해 테스트할 수 있음).

3.2.3 로봇을 선택하기

작업 현장 상황에 따라  를 클릭하여 해당 로봇의 브랜드와 모델을 선택합니다.

설정이 끝난 후 변경 사항을 저장하고 Mech-Center 를 재부팅합니다. 재부팅 후 **인터페이스 실행** 을 클릭하면 표준 인터페이스를 실행할 수 있습니다.

3.2.4 고급 설정

고급 설정에서 한 번에 보낼 수 있는 최대 포즈 수, Mech-Viz 데이터 획득을 위한 타임아웃 기간, Mech-Vision 데이터 획득을 위한 타임아웃 기간을 설정할 수 있습니다.

3.2.5 표준 인터페이스 Mech-Viz 예시 프로젝트

Mech-Center 설치 폴더의 `tool/viz_project` 폴더에 표준 인터페이스에 사용되는 4 개 예시 프로젝트가 있습니다.

Check_collision

비전 가이드로 피킹 과정 속의 경로 계획 및 충돌 감지를 진행합니다.

Outer_move

로봇이 외부 클라이언트 측에서 전송된 포즈가 위치하는 시나리오로 이동해야 합니다.

Suction_zone

DO 신호를 통해 멀티 빨판 또는 어레이 그리퍼를 컨트롤합니다.

Vision_result_reuse

비전 가이드로 피킹 과정에서 한번 반환된 포즈를 여러 번 사용하여 경로를 계획하고 충돌을 감지해야 하는 시나리오에 적용됩니다.

3.3 표준 인터페이스 개발자 참고 매뉴얼

3.3.1 개요

프로토콜 소개

표준 인터페이스에서 외부 서비스는 다음 다섯 가지 유형으로 나눠지며 사용자는 실제 필요에 따라 해당 외부 서비스 유형을 선택할 수 있습니다.

1. TCP Server

   Mech-Center 는 TCP Server 를 외부 서비스 인터페이스로 제공합니다. 문자열 및 16 진법 바이트 데이터 전송을 지원합니다.

2. Siemens PLC Client

   Mech-Center 는 SNAP7 프로토콜을 기반으로 하는 PLC Client 및 Siemens S7 시리즈 PLC 통신을 제공합니다.

3. PROFINET

   Mech-Center 는 EtherNet/IP 슬레이브로 EtherNet/IP 산업용 네트워크에 연결할 수 있습니다. EtherNet/IP 산업용 버스를 사용하여 통신하려면 다음과 같은 조건이 충족되어야 합니다.

   - 산업용 컴퓨터 또는 호스트는 표준 PCI-e 보드 설치를 지원합니다.
   - HMS INpact 40 PIR 보드 및 Ixxat VCI 드라이버 소프트웨어를 설치합니다.
   - Mech-Center 1.5.0 이상을 설치하고 소프트웨어에서 제공하는 GSD 장치 설명 파일을 사용합니다.
   - EtherNet/IP 통신은 표준 빅 엔디안 데이터 형식을 사용합니다. 데이터에는 32-bit INT 포즈 데이터가 포함되어 있으며 EtherNet/IP 마스터 스테이션 (특히 로봇 컨트롤러) 은 32-bit 정수 송수신을 지원해야 합니다.

4. EtherNet/IP

   Mech-Center 는 EtherNet/IP 슬레이브로 EtherNet/IP 산업용 네트워크에 연결할 수 있습니다. EtherNet/IP 산업용 버스를 사용하여 통신하려면 다음과 같은 조건이 충족되어야 합니다.

   - 산업용 컴퓨터 또는 호스트는 표준 PCI-e 보드 설치를 지원합니다.
   - HMS INpact 40 EIP 보드 및 Ixxat VCI 드라이버 소프트웨어를 설치합니다.
   - Mech-Center 1.5.1 이상을 설치하고 소프트웨어에서 제공하는 EDS 장치 설명 파일을 사용하십시오.
   - EtherNet/IP 통신은 표준 빅 엔디안 데이터 형식을 사용합니다. 데이터에는 32-bit INT 포즈 데이터가 포함되어 있으며 EtherNet/IP 마스터 스테이션 (특히 로봇 컨트롤러) 은 32-bit 정수 송수신을 지원해야 합니다.

5. Modbus TCP SLAVE

   Mech-Center 는 마스터 장치와의 데이터 통신을 위한 표준 인터페이스 옵션 MODBUS TCP SLAVE 를 제공하여 슬레이브 장치로 사용할 수 있습니다. 이 기능을 사용하려면 Mech-Center 1.6.1 및 이상 버전을 설치해야 합니다.

명령어 기능 소개

Mech-Vision 관련

101 : Mech-Vision 프로젝트 시작 Mech-Vision 프로젝트의 실행을 트리거하여 이미지를 캡처하고 비전 데이터를 처리합니다. Mech-Vision 만 사용하고 Mech-Viz 를 사용하지 않는 시나리오에서 사용합니다.

102 : 비전 목표점을 획득하기 비전 인식 결과, 즉 대상 물체의 픽 포인트를 읽어냅니다. Mech-Vision 만 사용하고 Mech-Viz 를 사용하지 않는 시나리오에서 사용합니다.

103 : Mech-Vision 레시피 전환 Mech-Vision 프로젝트에 저장된 레시피를 전환합니다. 다종의 작업물을 식별할 때 인식 템플릿, 딥 러닝 모델 파일 등과 같은 다양한 프로젝트 파라미터를 전환하는 데 사용됩니다.

110 : Mech-Vision 에서 자체 정의 프트의 출력 데이터를 획득하기 Mech-Vision 의 스 텝 procedure_out

Mech-Viz 관련

201 : Mech-Viz 프로젝트 시작 Mech-Viz 프로젝트를 시작하고 해당 Mech-Vision 프로젝트를 호출하고 이동 경로를 계획합니다. Mech-Vision 과 Mech-Viz 가 모두 있는 시나리오에서 사용됩니다.

202 : Mech-Viz 프로젝트 정지 Mech-Viz 프로젝트 실행을 수동으로 종료합니다.

203 : Mech-Viz 분기 선택 Mech-Viz 프로젝트에 branch_by_msg 태스크가 있는 경우 지정된 포트를 통해 데이터를 출력하도록 이 태스크를 컨트롤합니다.

204 : 이동 인덱스 설정 Mech-Viz 프로젝트에서 이동 태스크에 대한 인덱스 파라미터를 설정하는 데 사용됩니다. 인덱스 파라미터를 포함하는 이동 태스크에는 [순서대로 이동], [배열대로 이동], [자체 정의한 파렛트 패턴] 및 [미리 설정된 파렛트 패턴] 이 포함됩니다.

205 : 계획된 경로 가져오기 Mech-Viz 프로젝트에서 계획한 로봇이 이동 경로를 가져오기 위해 사용합니다.

206 : DO 신호 리스트 가져오기 Mech-Viz 에서 계산한 빨판 컨트롤 신호 리스트를 가져옵니다. 종이 박스가 여러 개 잡힐 때 빨판 파티션 컨트롤 시나리오에 사용됩니다.

207 : Mech-Viz 태스크 파라미터를 읽기 지정된 태스크의 지정된 파라미터의 수치를 읽어내는 데 사용됩니다. Mech-Center **구성 설정** → *Mech-Interface* → **고급 설정** 의 **속성 구성 파일** 에서 구체적인 태스크와 파라미터를 지정합니다.

207 : Mech-Viz 태스크 파라미터를 설정하기 지정된 태스크의 지정된 파라미터의 수치를 설정하는 데 사용됩니다. Mech-Center **구성 설정** → *Mech-Interface* → **고급 설정** 의 **속성 구성 파일** 에서 구체적인 태스크와 파라미터, 그리고 수치를 지정합니다.

210 : 이동 목표점과 비전 계획 결과를 획득하기 Mech-Viz 에서 계획된 단일한 목표점을 획득하는 데 사용됩니다. 목표점은 비전 이동 목표점일 수도 있고 다른 이동 태스크의 목표점일 수도 있습니다. 목표점에 포즈, 속도, 공구와 작업물의 정보 등이 포함될 수 있습니다.

자동으로 데이터 전송하기

501 : Mech-Vision 에 물체 크기 전달 이 명령은 물체의 3D 크기를 설정하는 데 사용됩니다. Mech-Vision 프로젝트에서 read_object_dimensions 스텝이 있고 물체 크기 (예를 들면 상자 크기: 길이, 너비, 높이) 를 외부에서 입력해야 하는 시나리오에 사용됩니다.

502 : Mech-Viz 에 TCP 발송 Mech-Viz 프로젝트에서 동적으로 변환하는 이동 목표점을 설정하는 데 사용되며 Mech-Viz 프로젝트에 outer_move 태스크가 반드시 있어야 합니다.

맞춤 알림 메시지

601: 알림 사용자가 이 명령어를 보낼 필요가 없습니다. Mech-Viz / Mech-Vision 프로젝트가 태스크 notify_viz 또는 스텝 notify_vision 까지 실행될 때 Mech-Center 는 태스크 또는 스텝에서 정의한 메시지를 클라이언트로 보낼 것입니다.

캘리브레이션

701:; 캘리브레이션 카메라를 위한 핸드-아이 캘리브레이션하는 데 사용됩니다. 로봇은 캘리브레이션 포인트를 요청하고 카메라가 사진을 찍도록 트리거하여 전체 캘리브레이션 프로세스를 완료합니다. 캘리브레이션 포인트는 Mech-Vision 에서 제공합니다.

시스템 상태 조회

901 : 소프트웨어 상태 획득 Mech-Mind 의 소프트웨어 실행 상태를 가져오는 데 사용되며 검사 역할을 합니다.

샘플 소개

표준 인터페이스의 샘플에 관한 내용은 표준 인터페이스 를 참조하십시오.

3.3.2 TCP/IP 명령어 설명

메크마인드 소프트웨어 시스템은 TCP/IP 프로토콜을 기반으로 하는 표준 인터페이스를 통해 ABB, FANUC, Kawasaki, KUKA, YASKAWA 로봇과 통신할 수 있습니다.

다음은 TCP/IP 명령입니다.

- *101* **명령어——*Mech-Vision* 프로젝트를 시작하기**
- *102* **명령어——비전 목표점을 획득하기**
- *103* **명령어——*Mech-Vision* 레시피를 전환하기**
- *110* **명령어——*Mech-Vision* 에서 사용자 정의 출력 데이터를 획득하기**
- *201* **명령어——*Mech-Viz* 프로젝트를 시작하기**
- *202* **명령어——*Mech-Viz* 프로젝트를 정지하기**
- *203* **명령어——*Mech-Viz* 분기를 선택하기**
- *204* **명령어——이동 인덱스를 설정하기**
- *205* **명령어——계획된 경로를 획득하기**
- *206* **명령어——*DO* 신호 리스트를 획득하기**

101 명령어——Mech-Vision 프로젝트를 시작하기

이 명령어는 Mech-Vision 프로젝트를 시작하고 사진 캡처 및 비전 인식을 수행하는 데 사용됩니다.

프로젝트가 Eye In Hand 모드인 경우 이 명령은 로봇이 이미지를 캡처했을 때의 포즈를 프로젝트로 전송합니다.

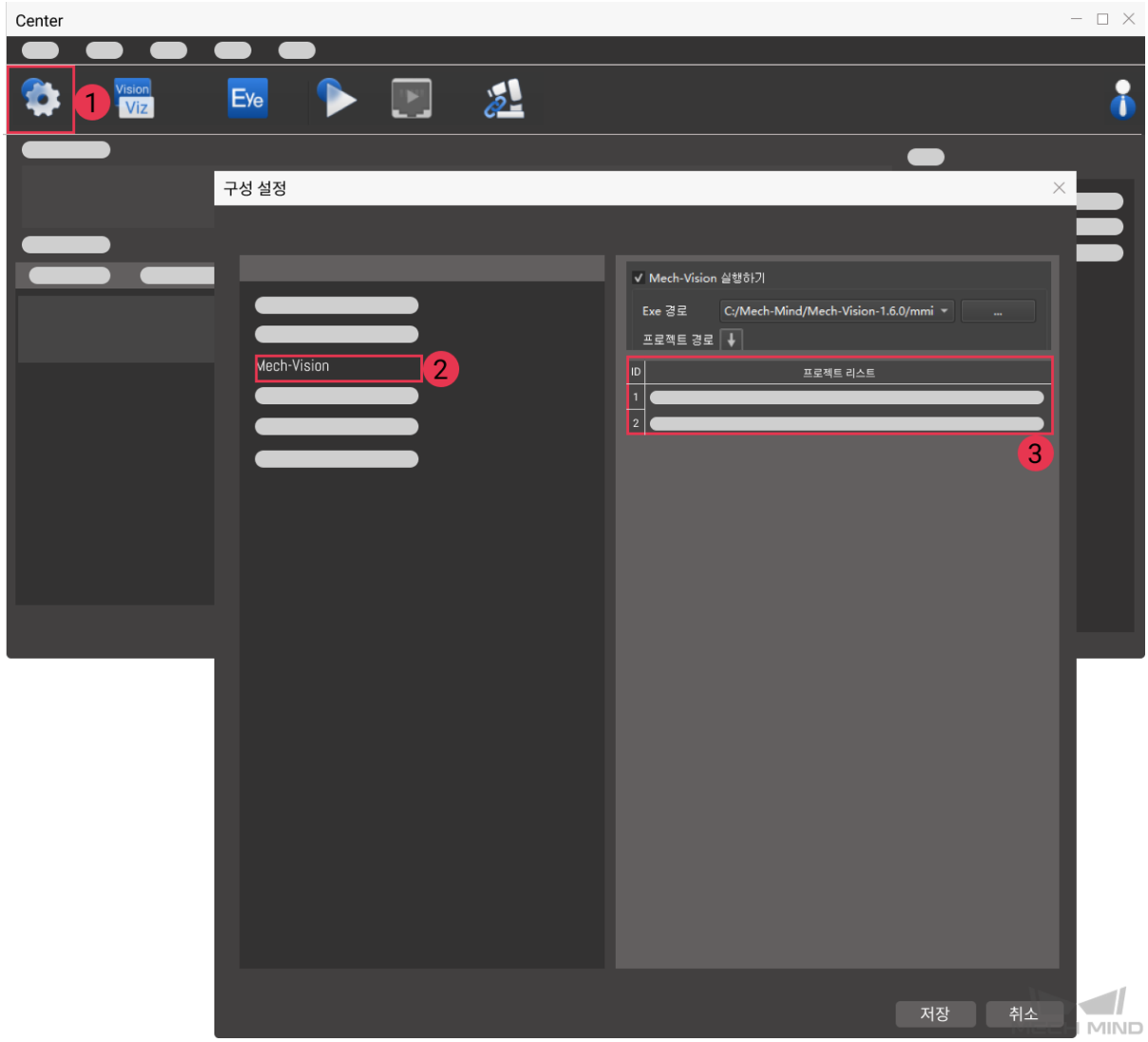
이 명령은 Mech-Vision 만 사용하는 시나리오에서 사용됩니다.

전송한 명령어 파라미터

101, Mech-Vision **프로젝트 번호**, **비전 포인트 수량의 목표값**, **로봇 포즈 유형**, **로봇 포즈**

Mech-Vision 프로젝트 번호

> Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



비전 포인트 수량의 목표값

Mech-Vision 에서 획득되길 바라는 비전 포인트의 수량입니다. 비전 포인트 정보에는 비전 포즈 및 포즈와 대응하는 포인트 클라우드, 레이블, 크기 조정 등이 포함됩니다.

- 0 Mech-Vision 프로젝트의 인식 결과에서 모든 비전 포인트를 획득합니다.
- 0 보다 큰 정수 ( ) Mech-Vision 프로젝트의 인식 결과에서 지정된 수의 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 작으면 인식 결과의 모든 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 크거나 같으면 이 파라미터에 지정된 비전 포인트 수를 획득합니다.

힌트: 비전 포인트를 획득하는 명령은 102 명령어입니다. TCP/IP 에서는 102 명령을 한번 실행하여 최대 20 개의 비전 포인트를 획득할 수 있습니다. 처음으로 102 명령어를 실행한 후 반환된 파라미터 중 하나는 요청된 모든 비전 포인트가 모두 반환되었는지 여부를 반영합니다. 그렇지 않은 경우 102 명령어를 반복 실행하십시오.

로봇 포즈 유형

이 파라미터는 실제 로봇의 포즈가 Mech-Vision 에 전달되는 형식을 지정합니다. 파라미터 범위: 0~2.

- 0 이 명령은 로봇 포즈를 비전 시스템에 전달할 필요가 없습니다. 프로젝트가 Eye to Hand 모드가 아닌 경우 이미지를 촬영하는 것은 로봇의 포즈와 관련이 없으며 Mech-Vision 은 로봇의 포즈가 필요하지 않습니다.
- 1 로봇 포즈는 JPs 관절 각도의 형식으로 전달됩니다.
- 2 로봇 포즈는 플랜지 포즈로 전달됩니다.

로봇 포즈

이 파라미터는 Eye In Hand 모드에서 필요한 로봇 포즈이며 로봇 포즈는 JPs 관절 각도 또는 플랜지 포즈의 형식입니다. 포즈의 형식은 **로봇 포즈 유형** 에 의해 결정됩니다.

반환한 데이터 파라미터

101, **상태 코드**

상태 코드

명령이 정상적으로 실행되면 **1102** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 101, 1, 10, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 101, 1102
```

명령이 비정상적으로 실행됨.

```
TCP send string = 101, 2, 10, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 101, 1011, 1
```

102 명령어——비전 목표점을 획득하기

101 **명령어——*Mech-Vision* 프로젝트를 시작하기** 이후에 사용되며, 이 명령어를 사용하여 Mech-Vision 의 인식 결과 (비전 포인트) 과 대응하는 로봇 포즈 및 레이블을 획득하며 로봇 포즈의 형식은 TCP 입니다.

Mech-Center 가 자동으로 비전 포인트의 정보를 해당한 로봇 TCP 로 전환할 것입니다. 구체적인 과정은 아래와 같습니다.

- 비전 포인트 정보에 포함한 포즈를 Y 축을 중심으로 180° 를 회전시킵니다.
- 해당 로봇 모델의 기준 좌표계 정의에 로봇 베이스의 높이가 포함된지를 인식하고 수직 방향에서의 옵셋을 추가합니다.

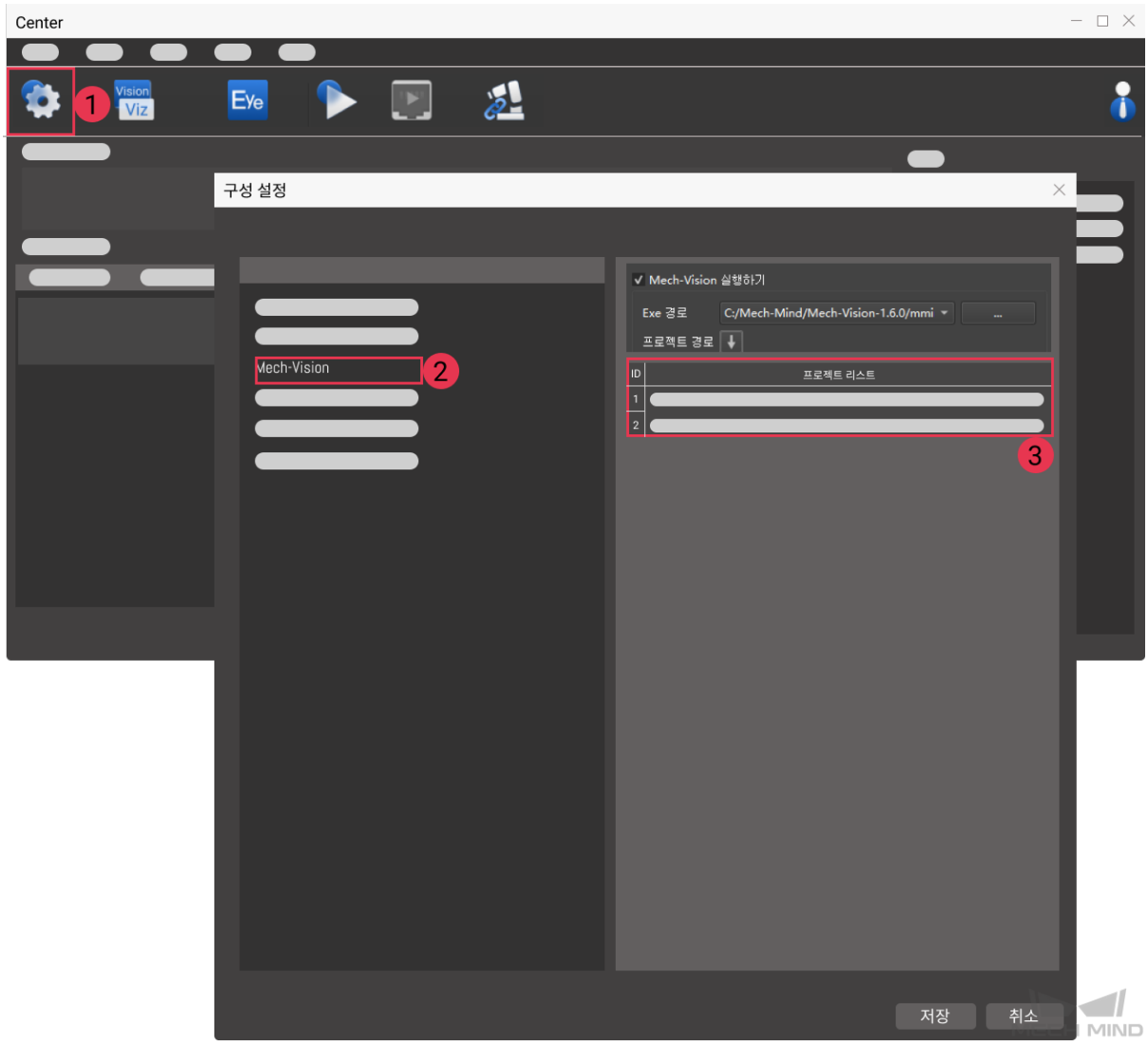
힌트: 102 명령어를 한번 실행하면 수신되는 TCP 수의 상한의 기본값은 20 입니다. 만약 획득할 TCP 의 수가 20 보다 크면 이 명령을 여러 번 실행하여 필요한 모든 TCP 를 획득합니다.

전송한 명령어 파라미터

102, Mech-Vision **프로젝트 번호**

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



반환한 데이터 파라미터

102, **상태 코드**, **전송 완료 여부**, *TCP* **수량**, **보류 필드**, **목표점**, **목표점**, **…**, **목표점**

참고: 목표점 데이터는 반환된 데이터 파라미터의 끝에 있으며 (한 번에 최대 20 개의 목표점 반환) 목표점에 TCP, 레이블, 속도 (값:0) 등 정보가 포함합니다.

상태 코드

명령어가 정상적으로 실행되면 **1100** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

이 명령이 호출될 때 Mech-Vision 결과가 반환되지 않은 경우 Mech-Center 는 로봇에 반환하기 전에 Mech-Vision 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초이며 제한 시간을 초과하면 타임아웃 오류 상태 코드를 반환합니다.

전송 완료 여부

이 파라미터는 필요한 모든 TCP 를 획득했는지 여부를 나타냅니다. 값은 0 또는 1 입니다.

- `0` 필요한 모든 TCP 를 획득하지 못했습니다. 파라미터 값이 1 이 될 때까지 102 명령을 반복적으로 실행하십시오.
- `1` 필요한 모든 TCP 를 획득 못했습니다.

101 **명령** 이 지정한 TCP 수량의 목표값이 20(송수신 데이터 길이의 기본값) 보다 크며 이 파라미터를 통해 전송되지 않은 TCP 가 있는지 판단할 수 있습니다. 데이터가 모두 전송되지 않은 경우 102 명령을 반복적으로 호출하여 계속 수신할 수 있습니다.

힌트: 모든 TCP 를 획득하지 못한 경우 이 때 다시 이미지를 캡처하라는 101 명령어를 호출하면 획득하지 못한 TCP 가 지워집니다.

TCP 수량

이 명령을 실행하여 얻은 TCP 의 수량입니다.

- 요청한 TCP 수가 Mech-Vision 에서 인식한 비전 포인트 수와 같거나 이상인 경우 Mech-Vision 에서 인식한 비전 포인트 수에 따라 전송됩니다.
- 요청한 TCP 수가 Mech-Vision 이 인식하는 비전 포인트 수보다 적을 경우 요청한 수만큼 전송됩니다.

기본 범위 0~20.

보류 필드

이 필드는 사용되지 않으며 기본값은 0 입니다.

목표점

```
            TCP,
  ,
```

- **물체 포즈와 대응하는 TCP** : TCP 에 3D 좌표 (XYZ) 및 오일러 각 (ABC) 이 포함됩니다.
- **레이블** 포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다. 프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

- **속도** 이 | 파라미터의 기본값은 0 입니다. 일반적으로 Mech-Vision 에서 출력한 비전 결과에 로봇 목표점의 속도 정보가 포함되지 않습니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 102, 1
TCP received string = 102, 1100, 1, 1, 0, 95.7806085592122, 644.5677779910724, 401.
↪1013614123109, 91.12068316085427, -171.13014981284968, 180.0, 0, 0
```

명령이 비정상적으로 실행되고 비전 결과가 없습니다.

```
TCP send string = 102, 1
TCP received string = 102, 1002, 1
```

비전 포인트를 요청하는 샘플 예시

다음 샘플 예시는 101, 102, 102 명령을 순서대로 전송하여 22 개의 비전 포인트와 대응하는 TCP 를 획득하는 과정입니다. 세부사항은 다음과 같습니다.

- TCP/IP 는 101 명령을 보내고 내용은 `101, 1, 0, 1, ...`이며 모든 TCP 를 얻기를 희망합니다.
- TCP/IP 는 102 명령을 보내 비전 목표점의 일부 TCP 를 획득합니다.
- TCP/IP 는 102 명령에서 반환된 데이터를 수신하고 내용이 `102, 1100, 0, 20, ...`이면 필요한 모든 TCP 를 얻지 못했음을 나타냅니다. 반환된 데이터에는 20 개의 TCP 가 포함됩니다.
- TCP/IP 는 102 명령을 다시 보내 나머지 TCP 를 획득합니다.
- TCP/IP 는 102 명령에서 반환된 데이터를 수신하며 내용은 `102, 1100, 1, 2, ...` 입니다. 반환된 데이터에는 2 개의 TCP 가 포함되어 있으며 필요한 모든 TCP 가 전송되었음을 나타냅니다.

101 명령어를 전송합니다.

```
TCP send string = 101, 1, 0, 1, -0, -20.63239, -107.81205, -0, -92.81818, 0.0016
TCP received strins = 101, 1102
```

처음으로 102 명령을 보내 20 개 TCP 를 얻습니다.

```
TCP send string = 102, 1

TCP received string = 102, 1100, 0, 20, 0, 95.7806085592122, 644.5677779910724, 401.
↪1013614123108, 31.12068316085427, ...
TCP received string = 78549940546, -179.99999999999991.0.0, 329.228345202334.712.7061697180302.
↪400.9702665047771, ...
TCP received string =39546, -83.62567351596952, -170.87955974536686, -179.99999999999937, 0, 0,
↪ 223.37118373658322, ...
TCP received string = 005627, 710.1004355953408, 400.82227273918835, -43.89328326393665, -171.
↪30845207792612, ...
TCP received string = 20.86318821742358, 838.7634193547805, 400.79807564314797, -102.
↪03947940869523, -171.149261231 ...
TCP received string = 390299920645, -179.99999999999994, 0, 0, 303.0722145720921, 785.
↪3254917220695, 400.75827437080, ...
TCP received string = 99668287.77.78291612041707, -171.53941633937786, 179.99999999899997, 0.0,
↪ 171.47819668864432, ...
TCP received string = 332193785, 400.6472716208158, -94.3418019038759, -171.10001228964776, -
↪179.39999999999994, ...
TCP received string = 92388542936, 807.5641001485708, 400.6021999602664, - 167.9834797197932.-
↪171.39671274951826, ...
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
TCP received string = 278.3198007132188, 780.5325992145735, 400.4924381003066, -174.
↪72728396633053, -171.422604771 ...
TCP received string = 3.99999999999994, 0, 0, 183.82195326381233, 862.5171519967056.400.
↪422966515846.-154. 17801945 ...
TCP received string = 173.34301974982765, -180.0, 0, 0
```

두번째로 102 명령을 다시 보내 나머지 두개 TCP 를 얻습니다.

```
TCP send string = 102, 1

TCP received string = 102, 1100, 1, 2, 0, 315.2017788478321, 592.1261793743445, 399.
↪60526335590957, 126.19602189220371, ...
TCP received string = 686127, -171.44430002882129, -1.3381805753922965e-15, 0, 0
```

103 명령어——Mech-Vision 레시피를 전환하기

Mech-Vision 프로젝트 내에서 파라미터 레시피를 전환합니다.

Mech-Vision 의 스텝에 대한 파라미터 설정은 파라미터 레시피를 전환하여 조정할 수 있습니다.

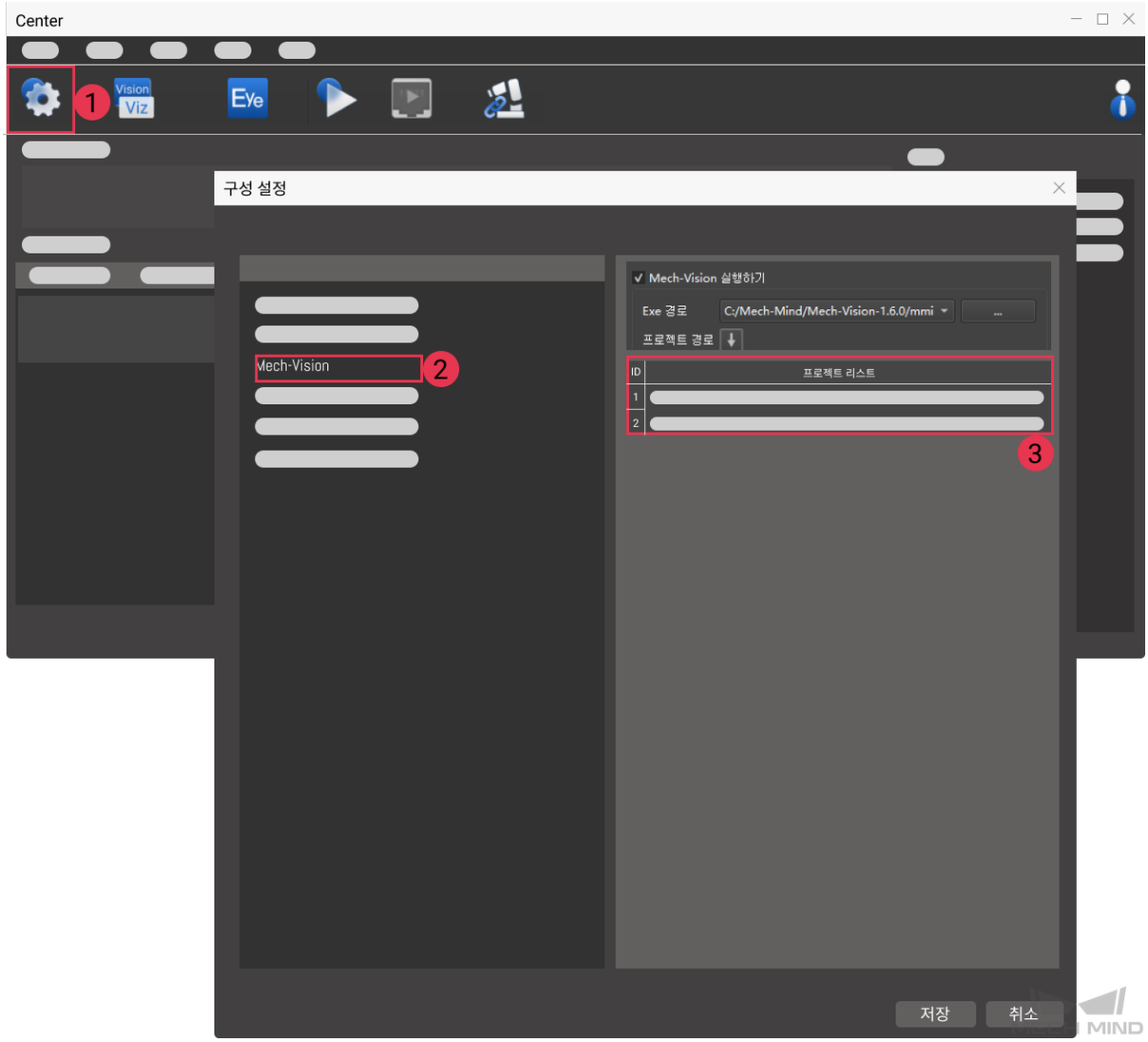
파라미터 레시피 조정과 관련된 파라미터에는 일반적으로 포인트 클라우드 매칭 모델, 이미지 매칭 모델, ROI, 믿음도 역치 등이 포함됩니다.

전송한 명령어 파라미터

103, Mech-Vision **프로젝트 번호**, **레시피 번호**

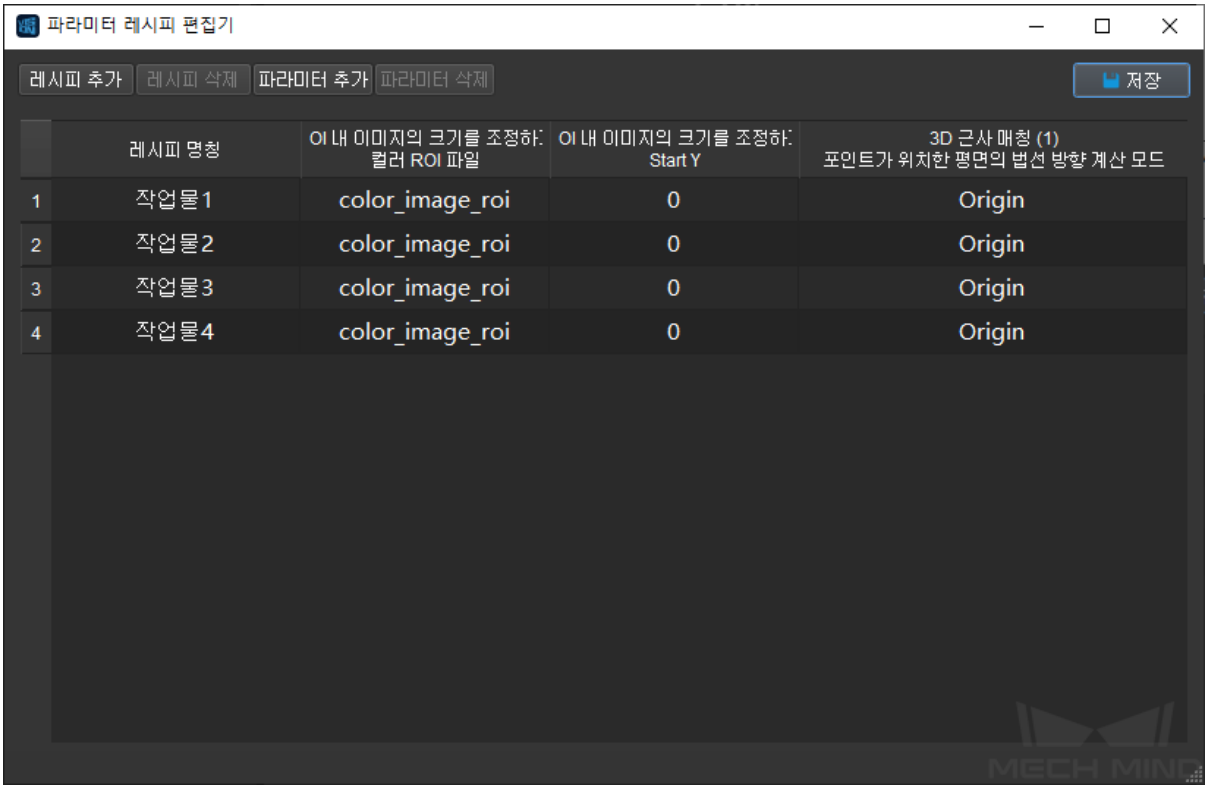
Mech-Vision 프로젝트 번호

> Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



레시피 번호

Mech-Vision 프로젝트의 레시피 템플릿 번호는 양의 정수 ( ) 입니다. **프로젝트 도우미** → **파라미터 레시피** 를 클릭하여 파라미터 레시피 편집기로 들어갑니다. 번호의 범위: 1~99.



반환한 데이터 파라미터

103, **상태 코드**

상태 코드

명령이 정상적으로 실행되면 **1107** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 103, 1, 2
TCP received string = 103, 1107
```

명령이 비정상적으로 실행됨.

```
TCP send string = 103, 1, 2
TCP received string = 103, 1102
```

110 명령어——Mech-Vision 에서 사용자 정의 출력 데이터를 획득하기

이 명령은 Mech-Vision 의 procedure_out 스텝에서 사용자 지정 유형 데이터, 즉 포즈 및 레이블 이외의 데이터 (스텝 파라미터《포트 유형"은"Dynamic"로 설정함) 를 수신하는 데 사용됩니다.

이 명령이 실행될 때마다 비전 결과에서 하나의 포즈와 대응하는 레이블, 점수 등 (있는 경우) 만 가져옵니다. 여러 포즈를 수신해야 하는 경우 이 명령을 여러 번 실행하십시오.

전송한 명령어 파라미터

110, Mech-Vision **프로젝트 번호**

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.

반환한 데이터 파라미터

110, **상태 코드**, **포즈 전송 완료 여부**, **자체 정의 데이터 항목 개수**, **대상 물체 포즈에 대응하는 로봇** *TCP* **포즈**, **레이블**, **자체 정의 데이터 항목**, **…**, **자체 정의 데이터 항목**

상태 코드

오류가 없으면 상태 코드 1100 이 반환됩니다. 그렇지 않으면 해당 오류 코드가 반환됩니다.

이 명령을 실행한 후, 만약 Mech-Vision 에서 결과가 반환되지 않은 경우 Mech-Center 는 로봇에 반환하기 전에 Mech-Vision 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초이며, 제한 시간을 초과하면 타임아웃 오류 상태 코드를 반환합니다.

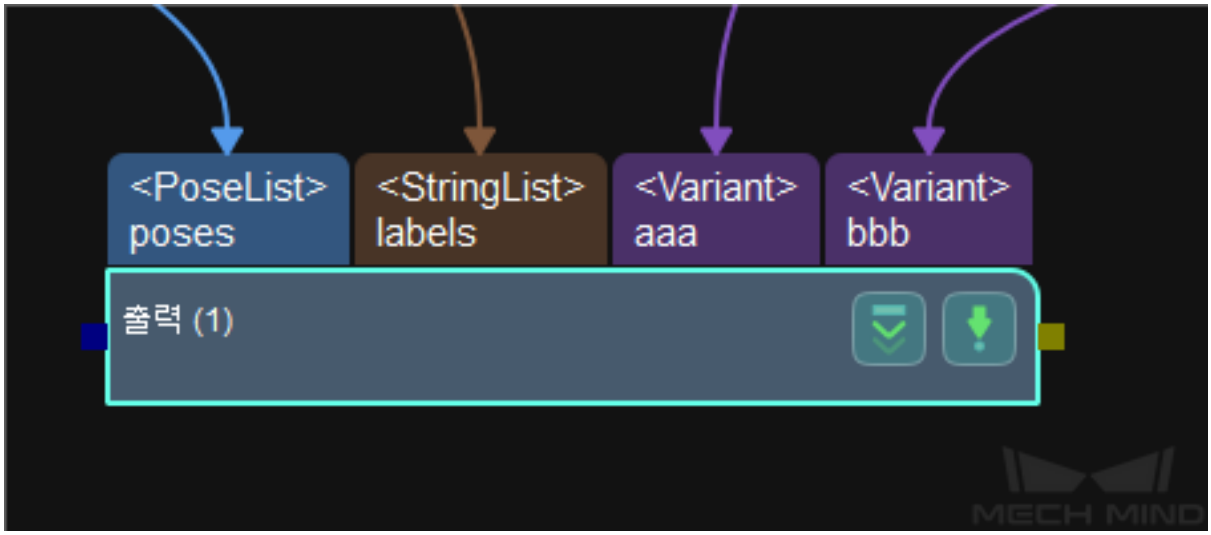
포즈 전송 완료 여부

- `0` 비전 결과에 아직 전달되지 않은 포즈가 있습니다.
- `1` 비전 결과의 모든 포즈가 이미 전송되었습니다.

자체 정의 데이터 항목 개수

포즈 및 레이블 이외의 데이터 유형의 사용자 정의 데이터 항목의 총 수입니다.

예를 들어, aaa 가 3 개 항목의 리스트이고 bbb 가 3 개 항목의 리스트인 경우 각 리스트에서 항목을 가져오기 위해 명령이 실행될 때 파라미터 값은 1+1=2 입니다. 만약 aaa 와 bbb 가 모두 단일 데이터인 경우 파라미터 값은 여전히 1+1=2 입니다.



대상 물체의 포즈에 대응하는 로봇의 TCP 포즈

로봇 좌표계의 TCP 포즈. Mech-Vision 은 대상 물체의 포즈를 출력하고 Mech-Center 는 포즈를 로봇의 TCP 포즈로 전환합니다. procedure_out 스텝에는 포즈 포트가 있어야 합니다.

대상 물체 포즈에 대응하는 TCP 포즈는 일반적으로 대상 물체 포즈의 Z 축을 회전하여 생성됩니다.

레이블

포즈와 대응하는 물체 정보 레이블입니다. 레이블은 양의 정수여야 합니다. 그렇지 않으면 Mech-Vision 프로젝트에서 label_mapping 스텝을 사용하여 양의 정수로 매핑해야 합니다. 만약 스텝 procedure_out 에 레이블 포트가 없으면 이 필드는 0 으로 채워집니다.

자체 정의한 데이터 항목

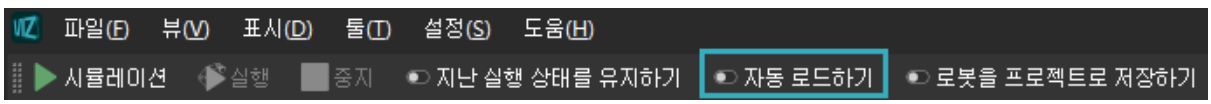
스텝 procedure_out 의 사용자 정의 데이터 유형에 대응하는 포트의 사용자 정의 데이터입니다.

사용자 정의 데이터의 파라미터는 포트 명칭의 알파벳 순서 A–Z 로 배열됩니다.

201 명령어——Mech-Viz 프로젝트를 시작하기

Mech-Vision 과 Mech-Viz 를 모두 사용하는 시나리오에 사용됩니다. 이 명령은 Mech-Viz 프로젝트를 실행하고 해당 Mech-Vision 프로젝트를 시작하는 데 사용되며 Mech-Viz 는 Mech-Vision 의 비전 결과를 기반으로 로봇의 이동 경로를 계획합니다.

시작할 Mech-Viz 프로젝트에서 **자동 로드하기** 를 선택해야 합니다.



Mech-Center 설치 디렉터리 (tool/viz_project) 폴더에 일부 전형적인 응용 프로그램 프로젝트 모델이 저장되어 있으며 사용자는 모델을 기반으로 수정할 수 있습니다.

표준 인터페이스용 Mech-Viz 샘플 프로젝트는**표준 인터페이스는 *Mech-Viz* 샘플 프로젝트 사용** 에 자세히 설명되어 있습니다.

전송한 명령어 파라미터

201, **포즈 유형**, **로봇 포즈**

포즈 유형

로봇의 포즈 유형입니다. 파라미터 범위: 0~1.

0

- Mech-Viz 는 현재 실제 로봇의 포즈를 읽을 필요가 없습니다. 이 명령은 포즈를 보내지 않습니다. 즉, 만약 프로젝트가 Eye to Hand 모드인 경우 이미지 캡처는 로봇 포즈와 아무 관련이 없으며 프로젝트는 로봇의 이미지 캡처 포즈를 읽을 필요가 없습니다.
- 포즈 유형이 0 으로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동합니다.

1

- Mech-Viz 에 전송된 포즈는 JPs 관절 각도 및 플랜지 포즈의 형식입니다. Mech-Viz 에서 가상 로봇은 초기 포즈 (즉, 이 명령에 의해 전송된 포즈) 에서 계획된 경로의 첫 번째 목표점으로 이동합니다.
- 포즈 유형이 1 로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs= 입력한 JPs 에서 첫 번째 이동 목표점으로 이동합니다.

힌트: 시나리오에 충돌 모델이 있고 로봇이 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동하는 것을 간섭할 때 포즈 유형을 1 로 설정해야 합니다.

로봇 포즈

로봇의 현재 JPs 관절 각도와 플랜지 포즈입니다 (파라미터“포즈 유형”이 1 인 경우).

반환한 데이터 파라미터

201, **상태 코드**

상태 코드

명령어가 정상적으로 실행되면 **2103** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 201, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 201, 2103
```

명령 실행 비정상 (Mech-Viz 는 TCP 유형을 지원하지 않음)

```
TCP send string = 201, 2, -0, 682.70355, 665.22266, 90, 179.99785, -89.99693
TCP received string = 201, 2015, 1
```

202 명령어——Mech-Viz 프로젝트를 정지하기

Mech-Viz 프로젝트 실행을 정지합니다. Mech-Viz 프로젝트가 무한 루프에 빠지지 않았거나 정상적으로 정지될 수 있는 경우 이 명령이 필요하지 않습니다.

전송한 명령어 파라미터

202

반환한 데이터 파라미터

202, **상태 코드**

상태 코드

명령이 정상적으로 실행되면 **2104** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 202
TCP received string = 202, 2104
```

203 명령어——Mech-Viz 분기를 선택하기

이 명령은 프로젝트가 따를 분기를 지정하는 데 사용됩니다. 분기 메커니즘은 branch_by_msg 에 의해 설정되며, 이 명령은 이 태스크의 아웃 포트를 지정함으로써 분기를 지정합니다.

이 명령을 실행하기 전에*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 을 실행하십시오.

Mech-Viz 프로젝트가 branch_by_msg 태스크까지 실행되면 이 명령으로 지정된 아웃 포트를 기다립니다.

전송한 명령어 파라미터

203, **분기 태스크 명칭**, **아웃 포트 번호**

분기 태스크 ID

이 파라미터는 어느 branch_by_msg 태스크에서 분기를 선택할지를 지정하는 데 사용됩니다.

이 파라미터는 양의 정수여야 합니다. 즉 branch_by_msg 의 태스크 ID 입니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

범위 1~99.

아웃 포트 번호

이 파라미터는 프로젝트가 branch_by_msg 태스크의 어느 아웃 포트를 따를지 지정하는 데 사용됩니다. Mech-Viz 프로그램은 이 아웃 포트를 따라 계속 실행될 것입니다. 파라미터는 양의 정수 ( ) 입니다.

힌트:

- 아웃 포트 번호는 Mech-Viz 에서 표시되는 포트 번호 + 1 입니다. 만약 포트 번호가 0 이면 아웃 포트 번호는 1 입니다.
- 아웃 포트 번호는 1 부터 시작하는 포트 인덱스 번호입니다. 예를 들어 지정된 아웃 포트가 왼쪽에서 오른쪽으로 두 번째 포트인 경우 아웃 포트 번호는 2 입니다.

반환한 데이터 파라미터

203, **상태 코드**

상태 코드

명령어가 정상적으로 실행되면 **2105** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 203, 1, 1
TCP received string = 203, 2105
```

명령이 비정상적으로 실행됨.

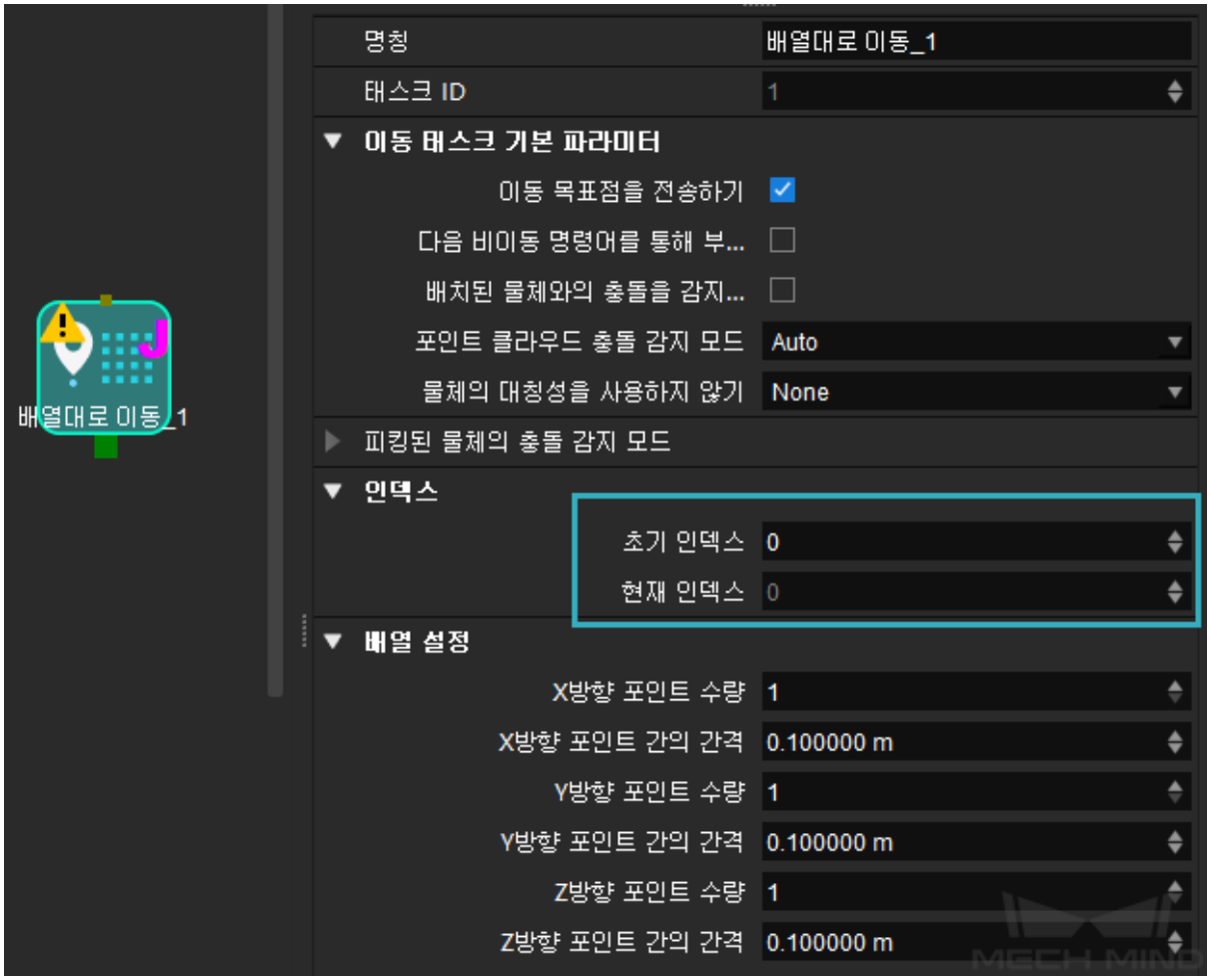
```
TCP send string = 203, 1, 3
TCP received string = 203, 2018, 1
```

204 명령어——이동 인덱스를 설정하기

태스크의 인덱스 파라미터 값을 설정하는 명령어입니다. 이러한 유형의 태스크는 일반적으로 연속적이거나 개별적으로 지정된 이동 또는 기타 작업에 사용됩니다.

인덱스 파라미터가 있는 태스크에는” 순서대로 이동”, ” 배열대로 이동”, ” 자체 정의한 파렛트 패턴” 및” 미리 설정된 파렛트 패턴” 등이 있습니다.

이 명령을 실행하기 전에*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 를 실행하십시오.



전송한 명령어 파라미터

204, **태스크 번호, 인덱스 값**

태스크 ID

이 파라미터는 인덱스 파라미터를 설정해야 하는 태스크를 지정합니다.

이 파라미터의 값은 양의 정수 ( ), 즉 인덱스 대기 중인 태스크의 태스크 ID 여야 합니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

범위 1~99.

인덱스 값

여기서 인덱스 파라미터의 값은 Mech-Viz 에 표시된 현재 인덱스 값에 1 을 더한 값입니다.

반환한 데이터 파라미터

204, **상태 코드**

상태 코드

명령어가 정상적으로 실행되면 **2106** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 204, 2, 6
TCP received string = 204, 2106
```

명령이 비정상적으로 실행됨.

```
TCP send string = 204, 3, 6
TCP received string = 204, 2028, 1
```

205 명령어——계획된 경로를 획득하기

이 명령은 Mech-Viz 에서 계획한 경로를 가져오는 데 사용됩니다. 계획된 경로는*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 실행 후 가져와야 합니다.

제한 사항: TCP/IP 에서 기본적으로 이 명령은 한 번에 계획된 경로의 최대 20 개의 목표점만 얻을 수 있으므로 이 명령을 여러 번 호출해야 할 수도 있습니다.

> **참고:** 프로젝트에서 어떤 이동 태스크의 목표점을 로봇으로 보내지 않아야 하는 경우 태스크 파라미터에서" 이동 목표점을 전송하기" 옵션을 선택 취소하십시오.

전송한 명령어 파라미터

205 **, 목표점 유형**

목표점 유형

이 파라미터는 Mech-Viz 가 반환할 목표점의 형식을 지정하는 데 사용됩니다.

- 1 목표점은 로봇 관절 각도 (JPs) 의 형식으로 반환됩니다.
- 2 목표점은 TCP 포즈의 형식으로 반환됩니다.

반환한 데이터 파라미터

205, **상태 코드**, **전송 완료 여부**, **목표점 수량**, **"비전 이동"태스크의 위치**, **목표점**, **목표점**, **…**, **목표점**

상태 코드

명령이 정상적으로 실행되면 **2100** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

힌트: 이 명령이 호출될 때 Mech-Viz 결과가 반환되지 않은 경우 (실행 중) Mech-Center 는 로봇에 반환하기 전에 Mech-Viz 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초입니다. 제한 시간을 초과하면 로봇에 타임아웃 오류를 반환합니다.

전송 완료 여부

- `0` 경로의 모든 목표점이 전송되지 않았습니다. 파라미터 값이 1 이 될 때까지 명령을 반복적으로 실행하십시오.
- `1` 경로의 모든 목표점이 전송되었습니다.

목표점 수량

이 파라미터는 이 명령에 의해 반환되는 경로의 이동 목표점 ([포즈, 레이블, 속도]) 의 수를 표시하는 데 사용됩니다.

경로에 20 개 이상의 목표점이 포함된 경우 이 명령을 여러 번 실행하십시오.

기본 범위 0~20.

" 비전 이동" 태스크 위치

" 비전 이동" 태스크의 목표점이 프로젝트 전체에서의 위치. " 비전 이동" 태스크는 비전 포인트 (물체를 피킹하는 포인트) 로 이동하는 이동 태스크입니다.

예를 들어 계획된 경로가" 이동 _1", " 이동 _2", " 비전 이동" " 이동 _3" 태스크로 구성된 경우" 비전 이동" 태스크 위치는 3 입니다.

경로에" 비전 이동" 태스크가 없는 경우 이 파라미터의 값은 0 입니다.

목표점

[포즈, 레이블, 속도]

- **포즈** 3D 좌표 및 오일러 각 또는 JPs 관절 각도. 유형은 명령어 205 중의 포즈 유형에 의해 결정됩니다.
- **레이블** 포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다. 프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.
- **속도** vision_move 태스크 파라미터의 0 이 아닌 속도 파라미터 백분율 값입니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string = 205, 1

TCP received string =205, 2100, 1. 2, 2, 8.307755332057372, 15.163476541700463, -142.
↪1778810972881, -2.7756047848536745, -31.44046012182799, -96.94907235126934, 0, 64, 8.2␣
↪42574265592342, 12.130080796661591, -141.75872288706663-2.513533225987894, -34.8905853␣
↪039525, -97.19108378871277, 0, 32
```

명령이 비정상적으로 실행됨.

```
TCP send string = 205, 1
TCP received strins = 205, 2008, 1
```

206 명령어——DO 신호 리스트를 획득하기

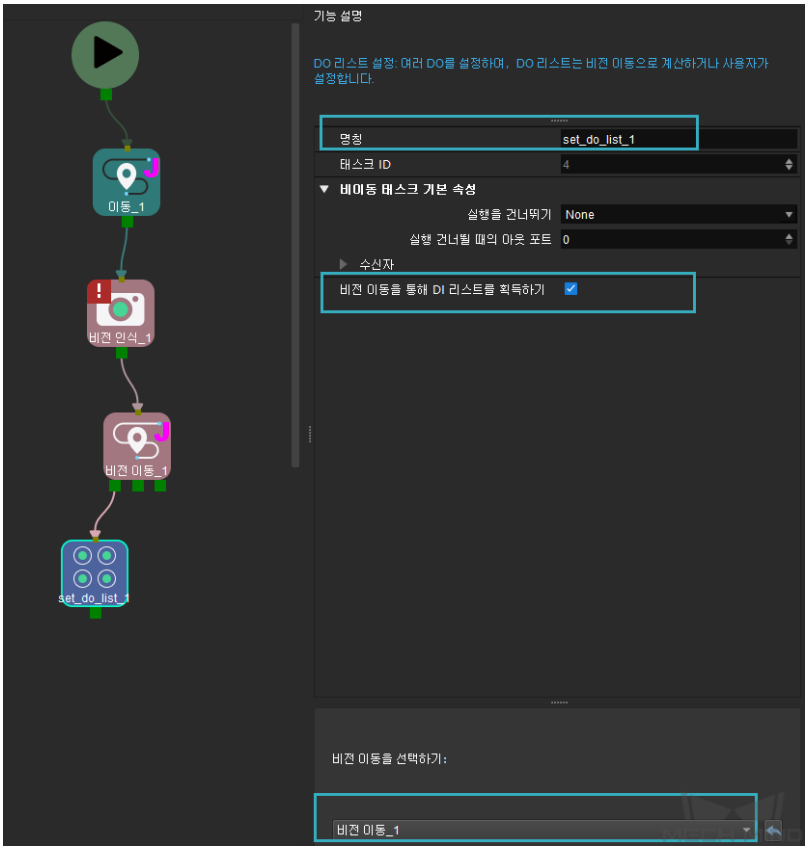
이 명령은 계획된 DO 신호 리스트를 획득하는 데 사용됩니다. DO 신호 리스트는 여러 도구 또는 빨판 파티션을 컨트롤하는 데 사용됩니다.

이 명령을 실행하기 전에 Mech-Viz 계획 경로를 얻기 위해*205* **명령** 을 실행해야 합니다.

템플릿 프로젝트에 따라 Mech-Viz 프로젝트를 구축하고 프로젝트에서 해당 빨판 구성 파일을 설정하십시오. 템플릿 프로젝트는 Mech-Center 설치 디렉터리 (tool/viz_project) 아래의 **suction_zone** 프로젝트입니다.

프로젝트의 set_do_list 태스크 파라미터에서

- ” 수신자” 에서” 표준 인터페이스” 를 선택하십시오.
- ” 비전 이동에서 DO 리스트를 획득하기” 를 선택하십시오.
- 파라미터 표시줄 하단에 DO 신호 리스트가 필요한 비전 이동 태스크를 선택하십시오.



전송한 명령어 파라미터

206

반환한 데이터 파라미터

206, **상태 코드**, *DO* **신호 값**, *DO* **신호 값**, ⋯, *DO* **신호 값**

상태 코드

명령어가 정상적으로 실행되면 **2102** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

DO 값

반환된 데이터의 파라미터 끝에 정수인 64 개의 DO 신호 값이 있습니다.

DO 신호 값 범위 0~999.

플레이스홀더 값 -1.

샘플 예시

명령어는 정상적으로 실행되며 가져온 DO 신호 값은 11 과 12 입니다.

```
TCP send string: 206
TCP receive string: 206, 2102, 11, 12, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,␣
↪-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
↪1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
↪1, -1
```

명령어가 비정상적으로 실행되어 DO 신호를 가져오지 못했습니다.

```
TCP send string: 206
TCP receive string: 206, 2011, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,␣
↪-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
↪1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
↪1, -1
```

207 명령어——Mech-Viz 태스크의 파라미터를 읽기

이 명령어는 지정된 태스크의 지정된 파라미터 값을 읽습니다.

Mech-Center 의 **배포 설정** → *Mech-Interface* → **고급 설정** 의 **속성 구성** 구성 파일에서 읽을 태스크의 파라미터를 지정하십시오.

구성 파일 내용 형식

read, config ID, **태스크** *ID,* **파라미터 툴팁**

구성 파일에서《#》으로 시작하는 내용은 주석 행으로 이 명령이 실행될 때 영향을 주지 않습니다.

read

문자열"read" 는 이 명령어가 파라미터 값을 **읽는데** 사용됨을 나타냅니다.

config ID

읽기 작업 ID 를 지정하는 양의 정수입니다. 하나의 config ID 는 하나의 파라미터를 읽는 데만 사용할 수 있으며 여러 파라미터를 읽어야 하는 경우 다른 config ID 를 사용하십시오.

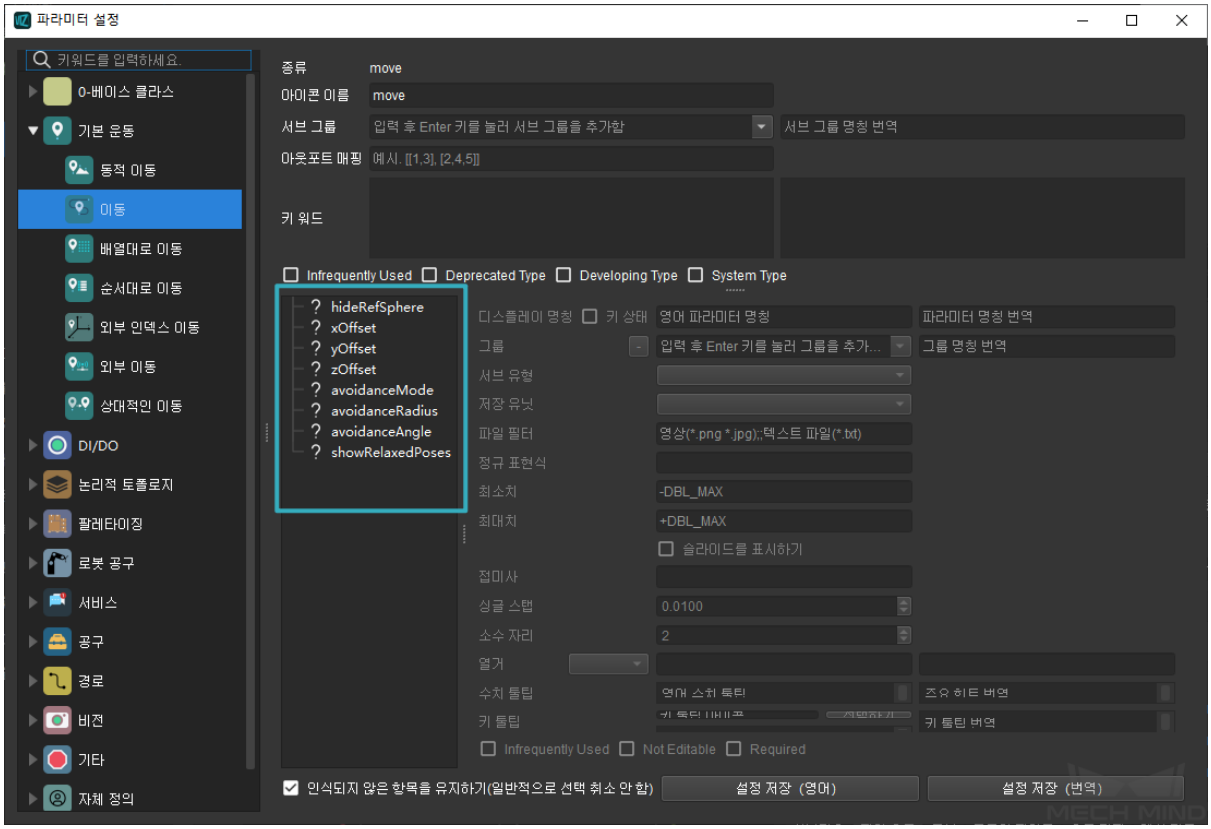
태스크 ID

읽을 파라미터가 속한 태스크 ID 입니다.

태스크 ID 는 태스크 파라미터에서 조회하고 설정할 수 있습니다.

파라미터 툴팁

읽을 파라미터 툴팁은 아래 그림과 같습니다. Mech-Viz 상단 메뉴 표시줄 **설정** → **기능 파라미터 설정** 을 통해 다음 창을 엽니다.



구성 파일 내용 샘플

```
read, 1, 10, digitalOutValue
read, 2, 2, xCount
read, 3, 2, yCount
read, 4, 5, allowVisionResultUnused
write, 1, 3, xOffset, 0.000000
write, 1, 3, yOffset, 0.000000
write, 1, 3, zOffset, 0.000030
write, 2, 7, delayTime, 0.1
```

반환한 데이터 파라미터

207, **상태 코드**, **태스크 파라미터 값**

상태 코드

오류가 발생하지 않으면 상태 코드 2109 가 반환됩니다. 그렇지 않으면 해당 오류 코드가 반환됩니다.

태스크 파라미터 값

Mech-Viz 프로젝트에서 지정된 태스크의 지정된 파라미터 값입니다.

지원되는 데이터 유형 INT FLOAT STRING.

208 명령어——Mech-Viz 태스크의 파라미터 값을 설정하기

이 명령어는 지정된 태스크의 지정된 파라미터 값을 설정합니다.

Mech-Center 의 **구성 설정** → *Mech-Interface* → **고급 설정** 의:guilabel:`속성 구성 파일`에서 어느 태스크의 어느 파라미터를 설정하고 어떤 값으로 설정할지 지정합니다.

구성 파일 내용 형식

write, *config ID*, **태스크** *ID*, **파라미터 툴팁**, **값**

write

문자열"write" 는 이 명령어가 파라미터 값을 **설정** 하는 데 사용됨을 나타냅니다.

config ID

작업 ID. 하나의 config ID 를 사용하여 여러 파라미터를 설정할 수 있습니다.

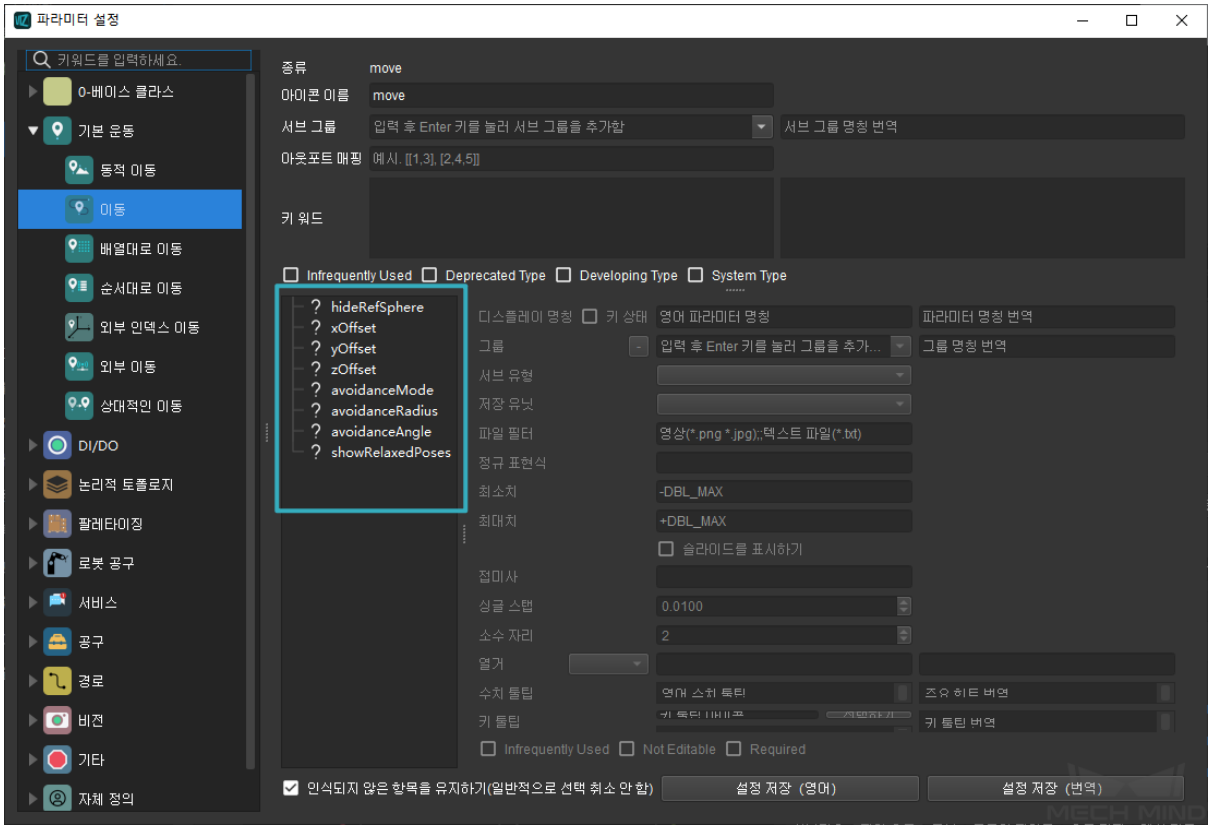
태스크 ID

설정할 파라미터가 속한 태스크의 ID 입니다.

태스크 ID 는 태스크 파라미터에서 조회하고 설정할 수 있습니다.

파라미터 툴팁

설정할 파라미터 툴팁입니다. 다음 그림과 같습니다. Mech-Viz 상단 메뉴 표시줄 **설정** → **기능 파라미터 설정** 을 통해 다음 창을 엽니다.



값

파라미터를 설정해야 하는 값을 지정합니다.

숫자 뿐만 아니라 문자열 값도 지원됩니다.

구성 파일 내용 샘플

```
read, 1, 10, digitalOutValue
read, 2, 2, xCount
read, 3, 2, yCount
read, 4, 5, allowVisionResultUnused
write, 1, 3, xOffset, 0.000000
write, 1, 3, yOffset, 0.000000
write, 1, 3, zOffset, 0.000030
write, 2, 7, delayTime, 0.1
```

210 명령어——이동 목표점과 비전 계획 결과를 획득하기

이 명령은 Mech-Viz 에서 계획한 단일한 목표점을 가져오는 데 사용됩니다. 목표점은 이동 목표점이나 비전 목표점이 될 수 있습니다. 목표점에는 포즈, 속도, 공구의 정보, 작업물의 정보 등이 포함될 수 있습니다.

이 명령을 실행하여 얻은 목표점은 다음 세 가지 중 하나가 될 수 있습니다.

1. vision_move 이외의 이동 태스크의 목표점으로 운동 유형 (관절 운동 또는 직선 운동), 공구 번호, 속도 등의 정보가 포함됩니다.
2. vision_move 태스크의 목표점, 그 정보에는 레이블, 피킹된 총 작업물 수, 다음 다수 피킹을 위해 계획된 작업물 수, 빨판의 에지 코너 번호, TCP 옵셋, 작업물 방향 및 작업물 조합의 크기를 포함합니다.
3. vision_move 태스크의 목표점은 Mech-Vision 프로젝트의 procedure_out 스텝의 출력을 포함합니다. 스텝의 포트 유형이“Dynamic”으로 설정된 경우 목표점에도 사용자 정의 데이터 유형이 포함됩니다.

전송한 명령어 파라미터

210, **예상한 반환 데이터 형식**

예상한 반환 데이터 형식

다음은 이동 목표점과 비전 계획 결과에 일반적인 네 가지 예상한 반환 데이터 형식입니다.

- JPs, 운동 유형, 공구 번호, 속도, 자체 정의 비전 출력 1, ..., 자체 정의 비전 출력 N
- TCP, 운동 유형, 공구 번호, 속도, 자체 정의 비전 출력 1, ..., 자체 정의 비전 출력 N
- JPs, 운동 유형, 공구 번호, 속도, 비전 계획 결과, 자체 정의 비전 출력 1, ..., 자체 정의 비전 출력 N
- TCP, 운동 유형, 공구 번호, 속도, 비전 계획 결과, 자체 정의 비전 출력 1, ..., 자체 정의 비전 출력 N

자체 정의한 비전 출력

자체 정의 비전 출력은 Mech-Vision 프로젝트의 procedure_out 스텝의 포즈와 레이블을 제외한 데이터입니다.

비전 계획 결과

vision_move 태스크의 계획 결과는 다음 정보를 포함합니다.

- 레이블: 10 개의 정수로 구성된 필드입니다.
- 이미 피킹된 작업물의 총 수입니다.
- 이번에 다수 피킹된 작업물의 수입니다.
- 에지 코너 번호: 빨판 에지 코너 번호는 vision_move 태스크의 **빨판 구성기** 에서 볼 수 있습니다.
- TCP 포즈 옵셋
- 작업물 방향
- 작업물 조합의 크기.

반환한 데이터 파라미터

210, **상태 코드**, **목표점 전송 완료 상태**, **목표점 유형**, **포즈**, **운동 유형**, **공구 번호**, **속도**, **비전 계획 결과** ***, **사용자 정의 비전 출력** ***

* 비전 계획 결과 및/또는 자체 정의 비전 출력이 있는지 여부는 명령을 실행할 때 전송되는 예상 데이터 형식에 따라 다릅니다.

상태 코드

오류가 발생하지 않으면 상태 코드 **2100** 이 반환됩니다. 그렇지 않으면 해당 오류 코드가 반환됩니다.

목표점 전송 완료 상태

- 0: 아직 전송되지 않은 목표점이 있습니다.
- 1: 모든 대상이 전송되었습니다.

목표점 유형

- 0: 목표점이 vision_move 태스크의 목표점이 아닙니다.
- 1: 목표점은 vision_move 태스크의 목표점입니다.

포즈

로봇 관절 각도 (JPs) 또는 TCP 포즈 (TCP) 형식인 목표점의 포즈, 형식은 전송된 명령 파라미터에 달렸습니다.

이동 유형

- 1 관절 운동 MOVEJ
- 2 직선 운동 MOVEL

공구 번호

목표점에서 사용할 공구 번호입니다. -1 은 공구가 사용되지 않음을 의미합니다.

속도

목표점에서 속도의 백분율 값, 즉 Mech-Viz 프로젝트의 목표점에 대응하는 이동 클래스 태스크의 파라미터에 설정된 속도입니다.

비전 계획 결과

목표점의 계획 결과 정보 (목표점이 이동 태스크 vision_move 에 대응하는 경우). 일반적으로 종이 상자의 다수 피킹 및 디팔레타이징에 사용됩니다. 정보에는 다음이 포함됩니다.

- 레이블: 최대 10 개의 양의 정수 레이블이 지원되며 기본값은 0 입니다.

- 이미 피킹된 작업물의 총 수입니다.
- 이번에 다수 피킹된 작업물의 수입니다.
- 에지 코너 번호: 작업물의 어느 에지 코너가 빨판에 가까운지 지정하는 데 사용됩니다. 빨판의 에지 코너 번호는 vision_move 태스크의 **빨판 구성기** 에서 볼 수 있습니다.
- TCP 포즈 옵셋: 작업물의 중심에 대응하는 TCP 에서 실제 TCP 까지의 옵셋입니다.
- 작업물 방향: TCP X 축을 기준으로 하는 작업물 좌표계 X 축의 방향입니다.
- 작업물 조합의 크기.

자체 정의한 비전 출력

스텝 procedure_out 의 사용자 정의 데이터 유형에 대응하는 포트의 사용자 정의 데이터입니다.

각 사용자 정의 비전 출력 파라미터는 포트 명칭에 따라 알파벳 A-Z 로 정렬됩니다.

포즈와 레이블을 제외한 다른 모든 포트 데이터는 사용자 정의 데이터로 처리됩니다.

501 명령어——Mech-Vision 에 물체 치수를 입력하기

이 파라미터는 물체 크기를 Mech-Vision 프로젝트로 동적으로 입력하는 데 사용됩니다. Mech-Vision 프로젝트를 시작하기 전에 물체 크기를 확인하십시오.

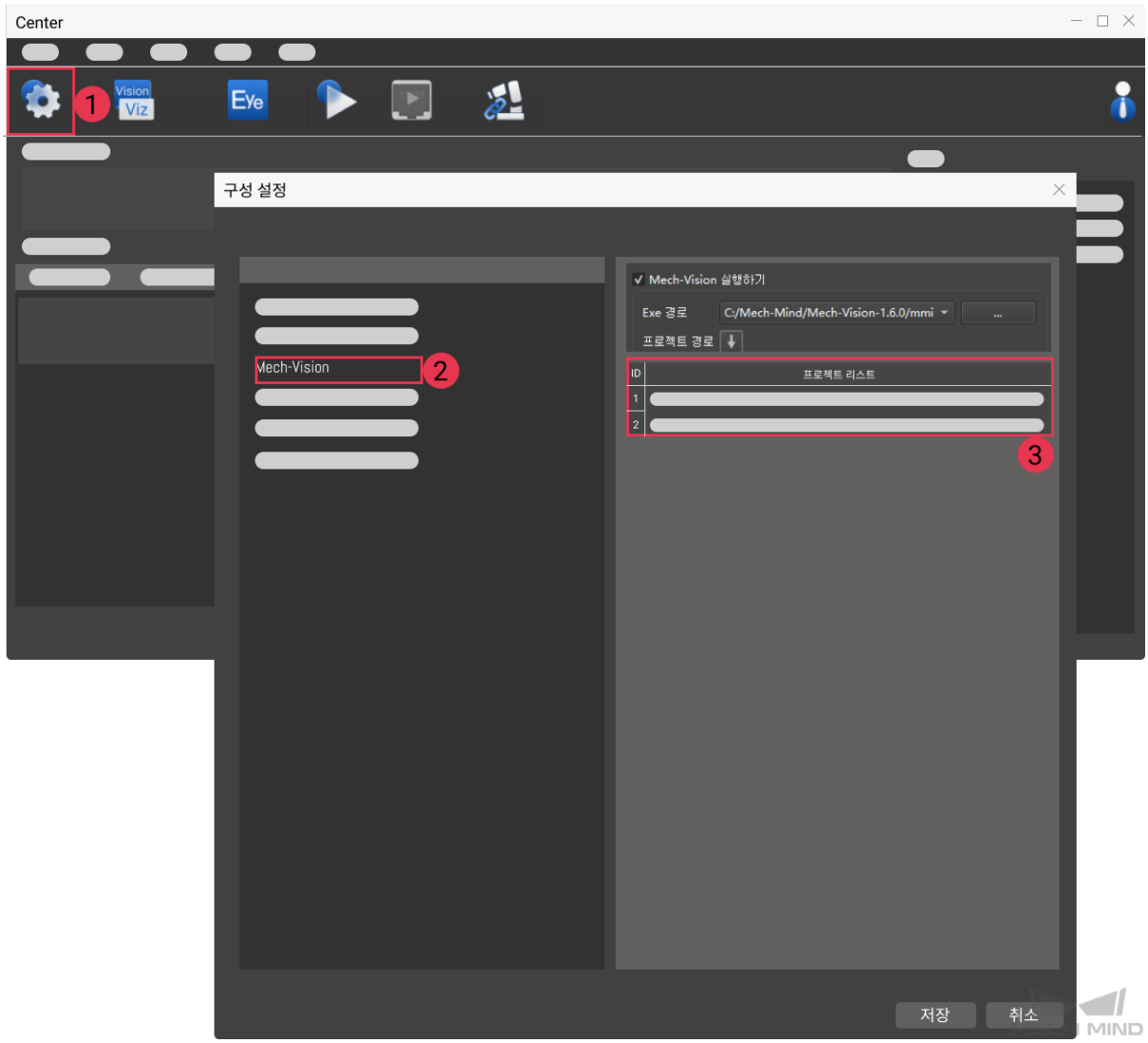
Mech-Vision 프로젝트에는 read_object_dimensions 스텝이 있어야 합니다. 이 스텝 파라미터 **파라미터에서 사이즈를 읽기** 는``True``로 설정해야 합니다.

전송한 명령어 파라미터

501, Mech-Vision **프로젝트 번호, 길이 , 너비 , 높이**

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



길이 , 너비 , 높이

Mech-Vision 프로젝트에 전달된 물체 사이즈입니다. 사이즈 값은 read_object_dimensions 스텝에서 읽습니다.

단위: 밀리미터 (mm)

반환한 데이터 파라미터

501, **상태 코드**

상태 코드

명령이 정상적으로 실행되면 **1108** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string: 501, 1, 100, 200, 300
TCP receive string: 501, 1108
```

명령 실행 비정상, 오류 코드 3002, " 높이" 값이 없습니다.

```
TCP send string: 501, 1, 100, 200
TCP receive string: 501, 3002
```

502 명령어——Mech-Viz 에 TCP 를 전송하기

이 명령어는 Mech-Viz 프로젝트에 로봇 TCP 를 동적으로 입력하는 데 사용됩니다. 로봇의 TCP 를 읽는 태스크는 outer_move 입니다.

템플릿 프로젝트를 기반으로 Mech-Viz 프로젝트를 구축하세요. 템플릿 프로젝트 경로는 Mech-Center 설치 디렉터리 아래의 `tool/viz_project/outer_move` 입니다.

outer_move 태스크를 작업 흐름의 올바른 위치에 배치합니다.

이 명령은*201* **명령어——***Mech-Viz* **프로젝트를 시작하기** 를 실행하기 전에 실행해야 합니다.

전송한 명령어 파라미터

502, **로봇** *TCP*

로봇 TCP

outer_move 태스크 목표점의 로봇 TCP 데이터를 설정하는 데 사용됩니다.

반환한 데이터 파라미터

502 **, 상태 코드**

상태 코드

명령어가 정상적으로 실행되면 **2107** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

샘플 예시

명령이 정상적으로 실행됨.

```
TCP send string: 502, 0, 10, 10, 20, 0, 0
TCP received string: 502, 2107
```

601 명령어——알림

사용자는 이 명령을 보낼 필요가 없습니다.

Mech-Viz / Mech-Vision 프로젝트가 notify_viz 태스크/ notify_vision 스텝까지 실행되면 Mech-Center는“알림”에 정의된 메시지를 클라이언트에게 보냅니다.

Mech-Vision 에서 notify_vision 스텝은“Standard Interface Notify”로 명명해야 합니다.

Mech-Viz 의 notify_viz 태스크 파라미터에서“접수 대상”아래의“표준 인터페이스”를 선택합니다.

전송한 명령어 파라미터

없음.

반환한 데이터 파라미터

601 **, 자체 정의한 알림 메시지**

자체 정의한 알림 메시지

notify_viz 태스크/ notify_vision 스텝에서 정의된 알림 메시지입니다. 메시지는 정수여야 합니다.

샘플 예시

만약 notify_viz 태스크/ notify_vision 스텝에 의해 정의된 알림 메시지가“1000”이면 프로젝트가 이 태스크/스텝으로 실행될 때 알림이 발송됩니다.

```
TCP receive string = 601, 1000
```

701 명령어——캘리브레이션

이 명령은 Hand-Eye 캘리브레이션 (카메라 외부 파라미터 캘리브레이션) 에 사용됩니다.

이 명령은 캘리브레이션 상태를 Mech-Vision 과 동기화하고 Mech-Vision 에서 로봇이 도달해야 하는 캘리브레이션 포인트를 얻습니다.

캘리브레이션을 완료하려면 이 명령을 여러 번 실행해야 합니다.

전송한 명령어 파라미터

701 **, 캘리브레이션 상태 , 플랜지 포즈 ,** *JPs* **관절 각도**

캘리브레이션 상태

파라미터 범위 0~2.

- 0 캘리브레이션 프로세스를 시작하도록 Mech-Vision 에 알립니다.
- 1 앞의 캘리브레이션 포인트가 수신되어 로봇으로 전송되었습니다.
- 2 앞의 캘리브레이션 포인트를 수신하지 못했습니다.

플랜지 포즈

로봇의 현재 플랜지 포즈입니다.

JPs 관절 각도

로봇의 현재 관절 각도입니다.

힌트: JPs 관절 각도 및 플랜지 포즈 중 하나를 선택하면 됩니다.

반환한 데이터 파라미터

701, **상태 코드**, **캘리브레이션 상태**, **다음 캘리브레이션 포인트의 플랜지 포즈**, **다음 캘리브레이션의 포인트** *JPs*

상태 코드

명령이 정상적으로 실행되면 **7101** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

캘리브레이션 상태

- `0` 캘리브레이션 진행 중.
- `1` 캘리브레이션 완료.

다음 캘리브레이션 포인트의 플랜지 포즈

로봇 이동 목표점의 포즈 데이터. 로봇 프로그램 측은 플랜지 포즈 또는 JPs 포즈를 사용하도록 선택할 수 있습니다.

다음 캘리브레이션의 포인트 관절 각도

로봇이 이동해야 할 다음 캘리브레이션 포인트의 JPs 입니다.

힌트: JPs 와 플랜지 포즈 중 하나를 선택하면 됩니다.

샘플 예시

캘리브레이션 프로세스 시작

```
TCP send string = 701, 0, 1371.62147, 25.6, 1334.3529, 148.58471, -179.24347, 88.75702, 88.
↪86102, -7.11107, -28.82309, -0.44014, -67.6509, 31.4764
TCP received string = 701, 7101, 0, 1271.6969, -743374, 1334.34094, -3128422, 1792412, -91.
↪11236, 93.28109, -12.0273, -32.8811, -0.37183, -68.41364, 27.02411
```

Mech-Vision 에서 캘리브레이션 포인트를 획득하기 (모든 캘리브레이션 포인트를 얻기 위해 이 과정을 여러 번 반복해야 합니다.)

```
TCP send string = 701, 1, 1271.6969. -74.3374. 1334.34094. -3128422, 1792412 -91.11236, 93.
↪28109, -12.0273, -32.8811, -0.37183, -68.41364, 27.02411
TCP received string = 701, 7101, 0, 1471.62226, -74.40452, 1334.34235. 148.56924, -179.24432,␣
↪88.74148, 92.8367, -2.14999, -24.25433, -0.39222, -67.23261, 27.485225
```

힌트: 여러 캘리브레이션 포인트를 얻기 위해 이 프로세스를 여러 번 반복해야 합니다.

캘리브레이션 종료

```
TCP send string = 701, 1, 1371.60876, 25.53615, 1384.45532. -20.82704. 179.22026, -72.77879,␣
↪88.88467, -7.42242.-26.68142, -0.2991, -69.95593, 39.26262
TCP received string = 701 7101, 1, 1371.62147.25.6, 1334.3529. 148, 58471. -179 24347, 88.
↪75702, 88.86102, -7.11107, -28.82309. -0.44014, -67.6509, 31.4764
```

901 명령어——소프트웨어 상태를 획득하기

이 명령은 소프트웨어의 실행 상태 (Mech-Vision, Mech-Viz, Mech-Center) 를 확인하는 데 사용됩니다. 현재 이 명령은 Mech-Vision 이 프로젝트 실행을 시작할 수 있는지 여부만 확인하는 것을 지원합니다.

전송한 명령어 파라미터

901

반환한 데이터 파라미터

901, **상태 코드**

상태 코드

시스템 자체 점검 상태. **1101** 상태 코드는"Mech-Vision 프로젝트가 준비됨"이고 다른 상태 코드는"Mech-Vision 프로젝트가 준비되지 않음"입니다. 현재 이 명령은 Mech-Vision 프로젝트가 준비되었는지 확인하는 데만 사용할 수 있습니다.

샘플 예시

Mech-Vision 프로젝트가 준비됨

```
TCP send string = 901
TCP received string = 901, 1101
```

Mech-Vision 프로젝트가 준비되지 않음

```
TCP send string = 901
TCP received string = 901, 1001, 1
```

힌트: Mech-Vision 에서 프로젝트를 열고 왼쪽의 프로젝트 리스트에서 프로젝트를 우클릭하고 **현재 프로젝트를 자동으로 로드하기** 를 선택하십시오.

3.3.3 Siemens PLC 명령어 설명

메크마인드 소프트웨어 시스템은 Siemens PLC Client 를 기반으로 하는 표준 인터페이스를 통해 Snap7 프로토콜을 지원하는 PLC 와 통신할 수 있습니다.

standard_interface_robot_and_plc 내용을 참조할 수 있습니다.

- *101* **명령어——*Mech-Vision* 프로젝트를 시작하기**
- *102* **명령어——비전 목표점을 획득하기**
- *103* **명령어——*Mech-Vision* 레시피를 전환하기**

101 명령어——Mech-Vision 프로젝트를 시작하기

이 명령어는 Mech-Vision 프로젝트를 시작하고 사진 캡처 및 비전 인식을 수행하는 데 사용됩니다.

프로젝트가 Eye In Hand 모드인 경우 이 명령은 로봇이 이미지를 캡처했을 때의 포즈를 프로젝트로 전송합니다.

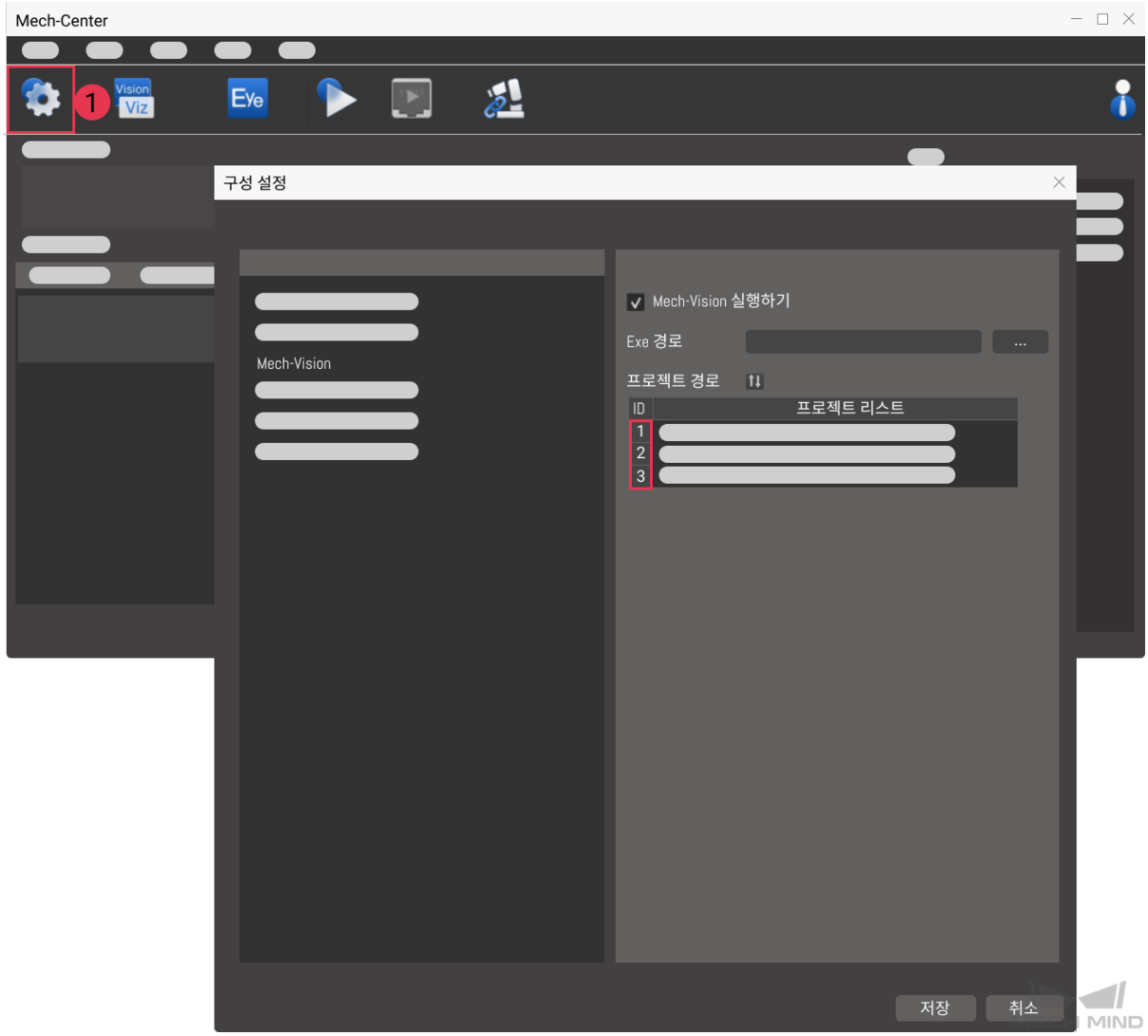
이 명령은 Mech-Vision 만 사용하는 시나리오에서 사용됩니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 101	2.0
Mech-Vision 프로젝트 번호	8.0
비전 포인트 수량의 목표값	6.0
로봇 포즈 유형	4.0
로봇 포즈	12.0(JPs) 또는 36.0(플랜지 포즈)

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



비전 포인트 수량의 목표값

Mech-Vision 에서 획득되길 바라는 비전 포인트의 수량입니다. 비전 포인트 정보에는 비전 포즈 및 포즈와 대응하는 포인트 클라우드, 레이블, 크기 조정 등이 포함됩니다.

- 0 Mech-Vision 프로젝트의 인식 결과에서 모든 비전 포인트를 획득합니다.
- 0 보다 큰 정수 ( ) Mech-Vision 프로젝트의 인식 결과에서 지정된 수의 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 작으면 인식 결과의 모든 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 크거나 같으면 이 파라미터에 지정된 비전 포인트 수를 획득합니다.

힌트: 비전 포인트를 획득하는 명령은 102 명령어입니다. PLC 의 기본 설정에서는 102 명령을 한번 실행하여 최대 20 개의 비전 포인트를 획득할 수 있습니다. 획득할 비전 포인트 수가 40 보다 크면 102 명령어를 반복적으로 호출해야 합니다.

로봇 포즈 유형

이 파라미터는 실제 로봇의 포즈가 Mech-Vision 에 전달되는 형식을 지정합니다. 파라미터 범위: 0~2.

- 0 이 명령은 로봇 포즈를 비전 시스템에 전달할 필요가 없습니다. 프로젝트가 Eye to Hand 모드가 아닌 경우 이미지를 촬영하는 것은 로봇의 포즈와 관련이 없으며 Mech-Vision 은 로봇의 포즈가 필요하지 않습니다.
- 1 로봇 포즈는 JPs 관절 각도의 형식으로 전달됩니다.
- 2 로봇 포즈는 플랜지 포즈로 전달됩니다.

로봇 포즈

이 파라미터는 Eye In Hand 모드에서 필요한 로봇 포즈이며 로봇 포즈는 JPs 관절 각도 또는 플랜지 포즈의 형식입니다. 포즈의 형식은 **로봇 포즈 유형** 에 의해 결정됩니다.

로봇 포즈는 6 개 Real 유형의 숫자로 구성됩니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령이 정상적으로 실행되면 **1102** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

102 명령어——비전 목표점을 획득하기

101 **명령어——*Mech-Vision* 프로젝트를 시작하기** 이후에 사용되며, 이 명령어를 사용하여 Mech-Vision 의 인식 결과 (비전 포인트) 와 대응하는 로봇 포즈 및 레이블을 획득하며 로봇 포즈의 형식은 TCP 입니다.

Mech-Center 가 자동으로 비전 포인트의 정보를 해당한 로봇 TCP 로 전환할 것입니다. 구체적인 과정은 아래와 같습니다.

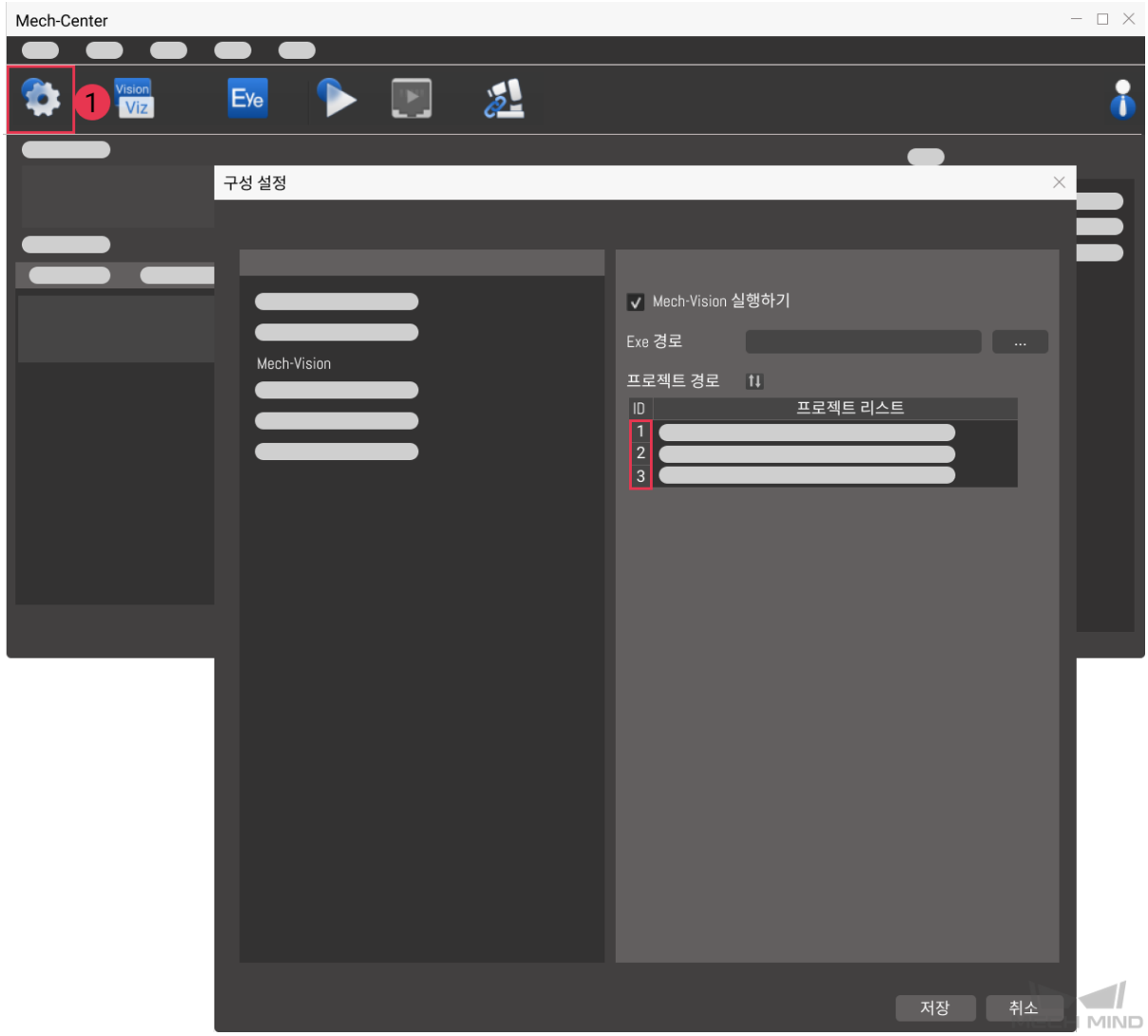
- 비전 포인트 정보에 포함한 포즈를 Y 축을 중심으로 180° 를 회전시킵니다.
- 해당 로봇 모델의 기준 좌표계 정의에 로봇 베이스의 높이가 포함된지를 인식하고 수직 방향에서의 옵셋을 추가합니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 102	2.0
Mech-Vision 프로젝트 번호	8.0

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0
데이터 전송 상태	202.0
TCP 수량	204.0
보류된 필드	/
이번에 획득한 모든 TCP	208.0
이번에 획득한 모든 레이블	1168.0

상태 코드

명령이 정상적으로 실행되면 **1100** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

이 명령이 호출될 때 Mech-Vision 결과가 반환되지 않은 경우 Mech-Center 는 로봇에 반환하기 전에 Mech-Vision 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초이며 제한 시간을 초과하면 타임아웃 오류 상태 코드를 반환합니다.

데이터 전송 상태

이 파라미터는 반환된 데이터가 새 TCP 인지를 표시하는 데 사용됩니다.

1 반환된 데이터는 새 TCP 이며 읽을 수 있습니다.

TCP 수량

이 명령을 실행하여 얻은 TCP 의 수량입니다.

- 요청한 TCP 수가 Mech-Vision 에서 인식한 비전 포인트 수와 같거나 이상인 경우 Mech-Vision 에서 인식한 비전 포인트 수에 따라 전송됩니다.
- 요청한 TCP 수가 Mech-Vision 이 인식하는 비전 포인트 수보다 적을 경우 요청한 수만큼 전송됩니다.

보류 필드

이 필드는 사용되지 않으며 기본값은 0 입니다.

이번에 획득한 모든 TCP

단일한 TCP 에 3D 좌표 (XYZ) 및 오일러 각 (ABC) 이 포함됩니다.

이번에 획득한 모든 레이블

포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다. 프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

103 명령어——Mech-Vision 레시피를 전환하기

Mech-Vision 프로젝트 내에서 파라미터 레시피를 전환합니다.

Mech-Vision 의 스텝에 대한 파라미터 설정은 파라미터 레시피를 전환하여 조정할 수 있습니다.

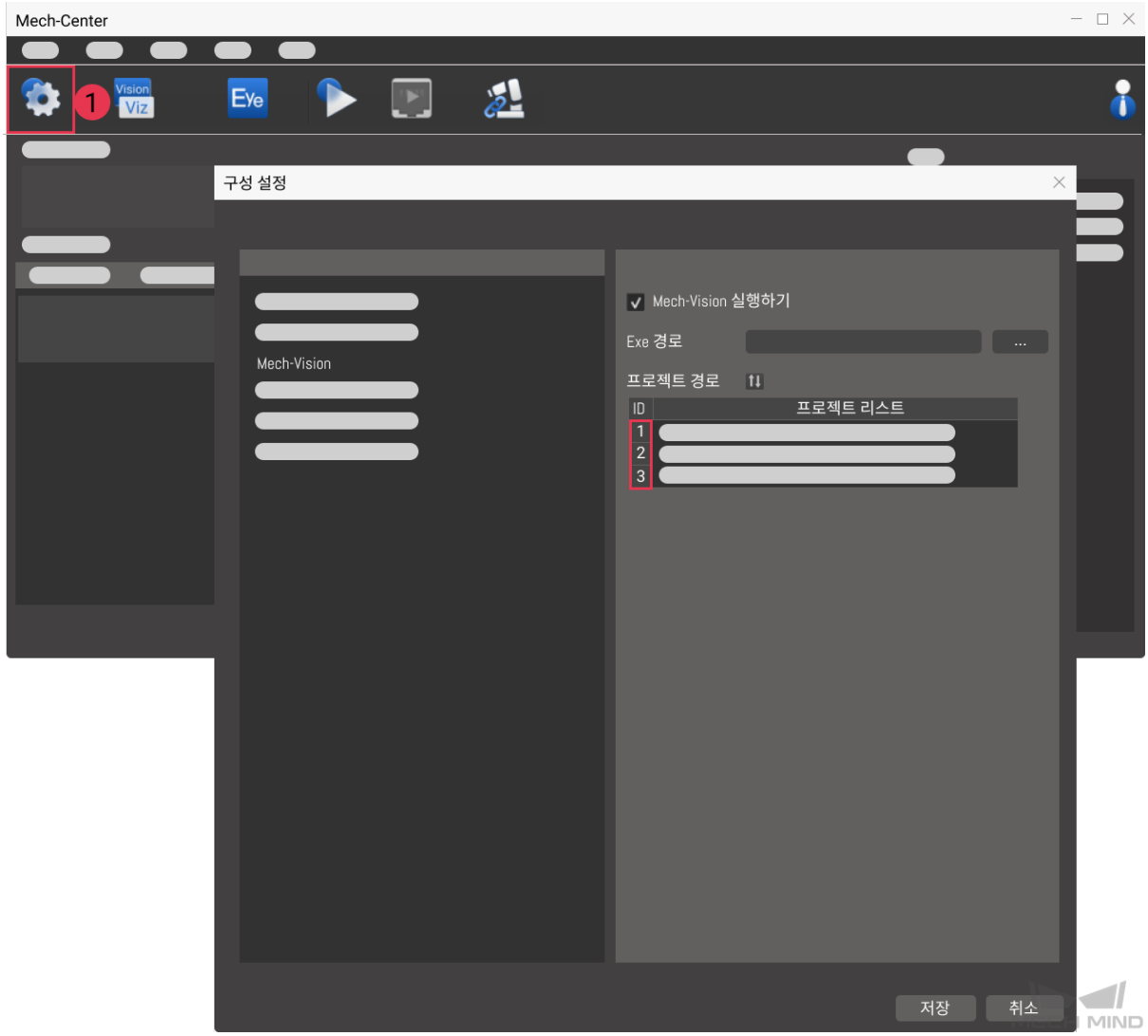
파라미터 레시피 조정과 관련된 파라미터에는 일반적으로 포인트 클라우드 매칭 모델, 이미지 매칭 모델, ROI, 믿음도 역치 등이 포함됩니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 103	2.0
Mech-Vision 프로젝트 번호	8.0
레시피 번호	10.0

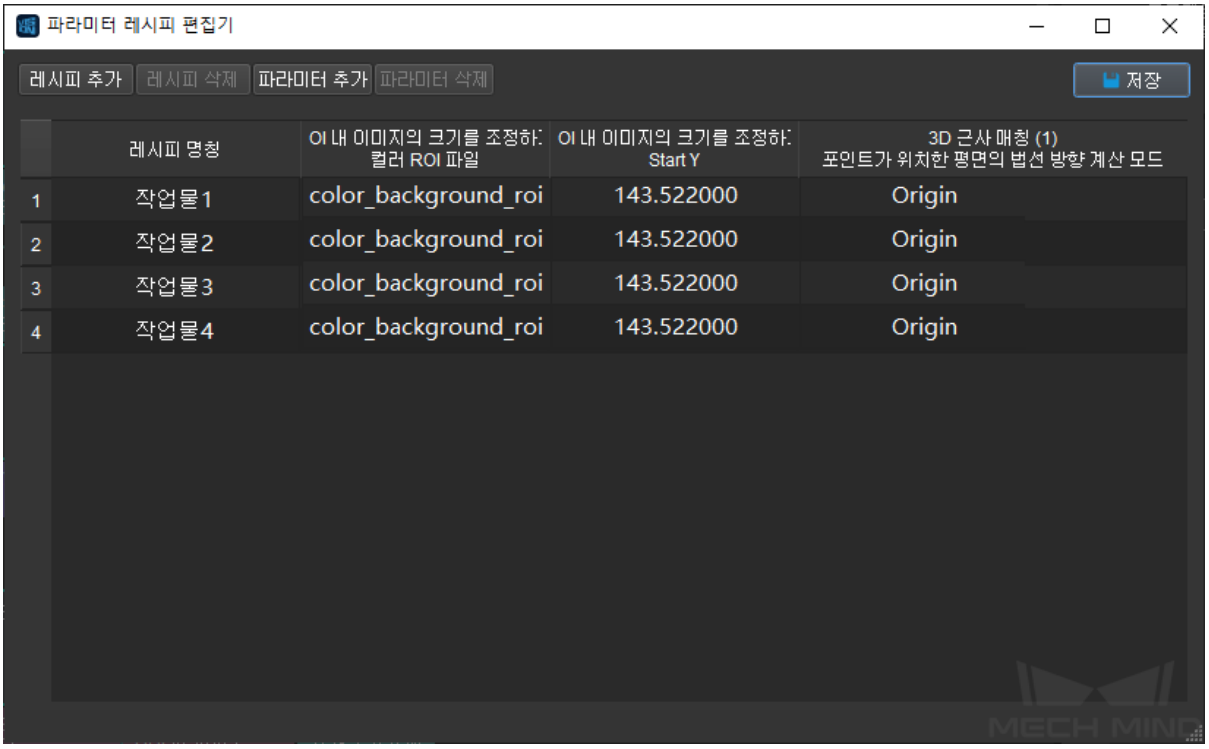
프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



레시피 번호

Mech-Vision 프로젝트의 레시피 템플릿 번호는 양의 정수 ( ) 입니다. **프로젝트 도우미** → **파라미터 레시피** 를 클릭하여 파라미터 레시피 편집기로 들어갑니다. 번호의 범위: 1~99.



반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

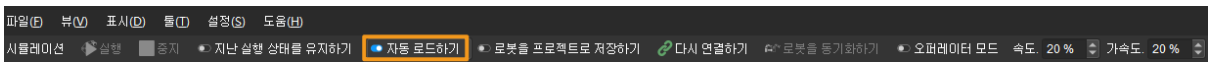
상태 코드

명령이 정상적으로 실행되면 **1107** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

201 명령어——Mech-Viz 프로젝트를 시작하기

Mech-Vision 과 Mech-Viz 를 모두 사용하는 시나리오에 사용됩니다. 이 명령은 Mech-Viz 프로젝트를 실행하고 해당 Mech-Vision 프로젝트를 시작하는 데 사용되며 Mech-Viz 는 Mech-Vision 의 비전 결과를 기반으로 로봇의 이동 경로를 계획합니다.

시작할 Mech-Viz 프로젝트에서 **자동 로드하기** 를 선택해야 합니다.



Mech-Center 설치 디렉터리 (tool/viz_project) 폴더에 일부 전형적인 응용 프로그램 프로젝트 모델이 저장되어 있으며 사용자는 모델을 기반으로 수정할 수 있습니다.

표준 인터페이스용 Mech-Viz 샘플 프로젝트는**표준 인터페이스는 *Mech-Viz* 샘플 프로젝트 사용** 에 자세히 설명되어 있습니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 201	2.0
포즈 유형	4.0
로봇 포즈	12.0

포즈 유형

로봇의 포즈 유형입니다. 파라미터 범위: 0~1.

0

- Mech-Viz 는 현재 실제 로봇의 포즈를 읽을 필요가 없습니다. 이 명령은 포즈를 보내지 않습니다. 즉, 만약 프로젝트가 Eye to Hand 모드인 경우 이미지 캡처는 로봇 포즈와 아무 관련이 없으며 프로젝트는 로봇의 이미지 캡처 포즈를 읽을 필요가 없습니다.
- 포즈 유형이 0 으로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동합니다.

1

- Mech-Viz 에 전송된 포즈는 JPs 관절 각도 및 플랜지 포즈의 형식입니다. Mech-Viz 에서 가상 로봇은 초기 포즈 (즉, 이 명령에 의해 전송된 포즈) 에서 계획된 경로의 첫 번째 목표점으로 이동합니다. TCP 형식으로 포즈를 전송할 수 없습니다.
- 포즈 유형이 1 로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs= 입력한 JPs 에서 첫 번째 이동 목표점으로 이동합니다.

힌트: 시나리오에 충돌 모델이 있고 로봇이 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동하는 것을 간섭할 때 포즈 유형을 1 로 설정해야 합니다.

로봇 포즈

로봇의 현재 JPs 관절 각도입니다 (파라미터“포즈 유형”이 1 인 경우).

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령어가 정상적으로 실행되면 **2103** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

202 명령어——Mech-Viz 프로젝트를 정지하기

Mech-Viz 프로젝트 실행을 정지합니다. Mech-Viz 프로젝트가 무한 루프에 빠지지 않았거나 정상적으로 정지될 수 있는 경우 이 명령이 필요하지 않습니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 202	2.0

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령이 정상적으로 실행되면 **2104** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

203 명령어——Mech-Viz 분기를 선택하기

이 명령은 프로젝트가 따를 분기를 지정하는 데 사용됩니다. 분기 메커니즘은 branch_by_msg 에 의해 설정되며, 이 명령은 이 태스크의 아웃 포트를 지정함으로써 분기를 지정합니다.

이 명령을 실행하기 전에*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 을 실행하십시오.

Mech-Viz 프로젝트가 branch_by_msg 태스크까지 실행되면 이 명령으로 지정된 아웃 포트를 기다립니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 203	2.0
분기 태스크 ID	60.0
아웃 포트 번호	62.0

분기 태스크 ID

이 파라미터는 어느 branch_by_msg 태스크에서 분기를 선택할지를 지정하는 데 사용됩니다.

이 파라미터는 양의 정수여야 합니다. 즉 branch_by_msg 의 태스크 ID 입니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

태스크 ID 범위: 1~99.

아웃 포트 번호

이 파라미터는 프로젝트가 branch_by_msg 태스크의 어느 아웃 포트를 따를지 지정하는 데 사용됩니다. Mech-Viz 프로그램은 이 아웃 포트를 따라 계속 실행될 것입니다. 파라미터는 양의 정수 ( ) 입니다.

아웃 포트 번호 범위: 1~99.

힌트:

- 아웃 포트 번호는 Mech-Viz 에서 표시되는 포트 번호 + 1 입니다. 만약 포트 번호가 0 이면 아웃 포트 번호는 1 입니다.
- 아웃 포트 번호는 1 부터 시작하는 포트 인덱스 번호입니다. 예를 들어 지정된 아웃 포트가 왼쪽에서 오른쪽으로 두 번째 포트인 경우 아웃 포트 번호는 2 입니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

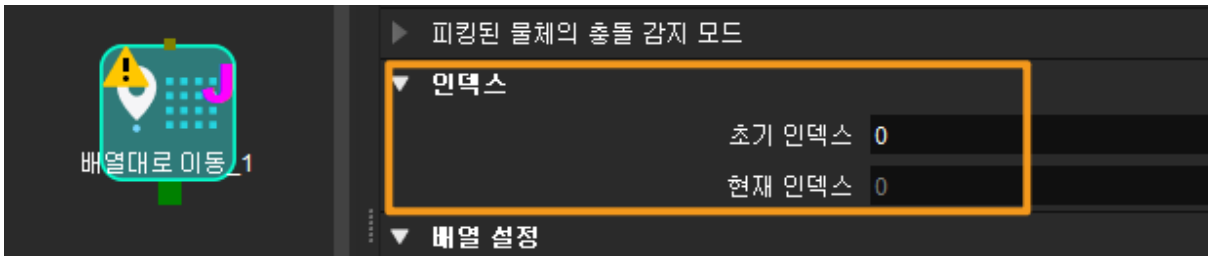
명령어가 정상적으로 실행되면 **2105** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

204 명령어——이동 인덱스를 설정하기

태스크의 인덱스 파라미터 값을 설정하는 명령어입니다. 이러한 유형의 태스크는 일반적으로 연속적이거나 개별적으로 지정된 이동 또는 기타 작업에 사용됩니다.

인덱스 파라미터가 있는 태스크에는” 순서대로 이동”, ” 배열대로 이동”, ” 자체 정의한 파렛트 패턴” 및” 미리 설정된 파렛트 패턴” 등이 있습니다.

이 명령을 실행하기 전에*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 를 실행하십시오.



전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 204	2.0
태스크 ID	64.0
인덱스 값	66.0

태스크 ID

이 파라미터는 인덱스 파라미터를 설정해야 하는 태스크를 지정합니다.

이 파라미터의 값은 양의 정수 ( ), 즉 인덱스 대기 중인 태스크의 태스크 ID 여야 합니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

인덱스 값

여기 인덱스 파라미터 값은 Mech-Viz 에 표시된 현재 인덱스 값에 1 을 더한 값입니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령어가 정상적으로 실행되면 **2106** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

205 명령어——계획된 경로를 획득하기

이 명령은 Mech-Viz 에서 계획한 경로를 가져오는 데 사용됩니다. 계획된 경로는*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 실행 후 가져와야 합니다.

팁: 프로젝트에서 어떤 이동 태스크의 목표점을 로봇으로 보내지 않아야 하는 경우 태스크 파라미터에서" 이동 목표점을 전송하기" 옵션을 선택 취소하십시오.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 205	2.0
목표점 유형	4.0

목표점 유형

이 파라미터는 Mech-Viz 가 반환할 목표점의 형식을 지정하는 데 사용됩니다.

- 1 목표점은 로봇 관절 각도 (JPs) 의 형식으로 반환됩니다.
- 2 목표점은 TCP 포즈의 형식으로 반환됩니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0
데이터 전송 상태	202.0
목표점 수량	204.0
" 비전 이동" 태스크의 위치	206.0
이번에 전송한 모든 목표점의 포즈	208.0
이번에 전송한 모든 목표점의 레이블	1168.0
이번에 전송한 모든 목표점의 속도	1248.0

상태 코드

오류가 발생하지 않으면 상태 코드 **2100** 이 반환됩니다. 그렇지 않으면 해당 오류 코드가 반환됩니다.

힌트: 이 명령이 호출될 때 Mech-Viz 결과가 반환되지 않은 경우 (실행 중) Mech-Center 는 로봇에 반환하기 전에 Mech-Viz 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초입니다. 제한 시간을 초과하면 로봇에 타임아웃 오류를 반환합니다.

데이터 전송 상태

이 파라미터는 반환된 데이터가 새 목표점인지를 표시하는 데 사용됩니다.

1 반환된 데이터는 새 목표점이며 읽을 수 있습니다.

목표점 수량

이 파라미터는 이 명령에 의해 반환되는 경로의 이동 목표점 ([포즈, 레이블, 속도]) 의 수를 표시하는 데 사용됩니다.

경로에 20 개 이상의 목표점이 포함된 경우 이 명령을 여러 번 실행하십시오.

범위: 0~20.

" 비전 이동" 태스크의 위치

" 비전 이동" 태스크의 목표점이 프로젝트 전체에서의 위치. " 비전 이동" 태스크는 비전 포인트 (물체를 피킹하는 포인트) 로 이동하는 이동 태스크입니다.

예를 들어 계획된 경로가" 이동 _1", " 이동 _2", " 비전 이동" " 이동 _3" 태스크로 구성된 경우" 비전 이동" 태스크 위치는 3 입니다.

경로에" 비전 이동" 태스크가 없는 경우 이 파라미터의 값은 0 입니다.

포즈

3D 좌표 및 오일러 각 또는 JPs 관절 각도. 유형은 명령어 205 중의 포즈 유형에 의해 결정됩니다.

레이블

포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다.

프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

속도

vision_move 태스크 파라미터의 0 이 아닌 속도 파라미터 백분율 값입니다.

206 명령어——DO 신호 리스트를 획득하기

이 명령은 계획된 DO 신호 리스트를 획득하는 데 사용됩니다. DO 신호 리스트는 여러 도구 또는 빨판 파티션을 컨트롤하는 데 사용됩니다.

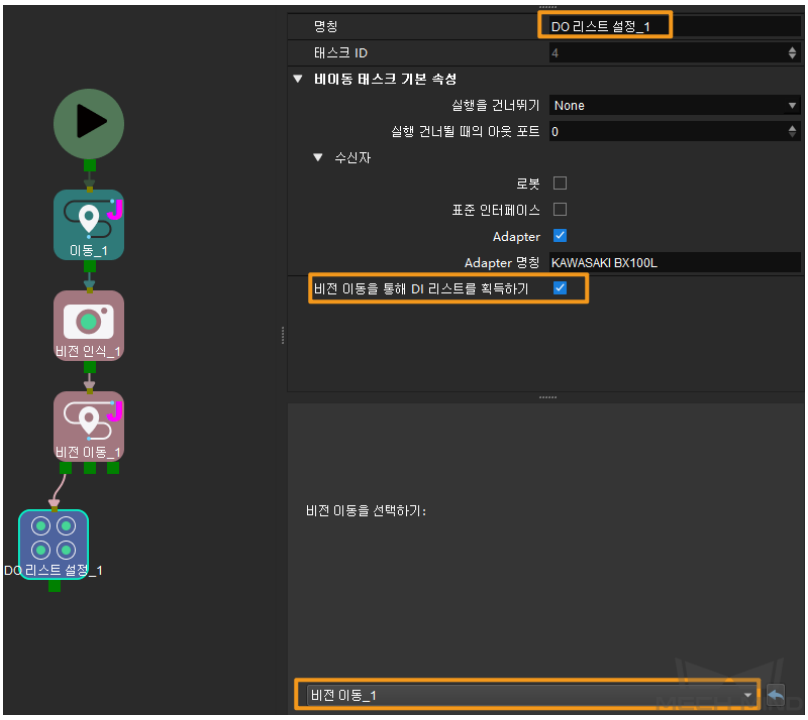
이 명령을 실행하기 전에 Mech-Viz 계획 경로를 얻기 위해*205* **명령** 을 실행해야 합니다.

템플릿 프로젝트에 따라 Mech-Viz 프로젝트를 구축하고 프로젝트에서 해당 빨판 구성 파일을 설정하십시오. 템플릿 프로젝트는 Mech-Center 설치 디렉터리 (tool/viz_project) 아래의 **suction_zone** 프로젝트입니다.

프로젝트의 set_do_list 태스크 파라미터에서

- " 수신자" 에서" 표준 인터페이스" 를 선택하십시오.
- " 비전 이동에서 DO 리스트를 획득하기" 를 선택하십시오.

- 파라미터 표시줄 하단에 DO 신호 리스트가 필요한 비전 이동 태스크를 선택하십시오.



전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 206	2.0

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0
DO 리스트	1408.0

상태 코드

명령어가 정상적으로 실행되면 **2102** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

DO 포트 값

정수 ( ) 인 64 개의 DO 신호 값이 있습니다.

DO 신호 값 범위 0~999.

플레이스홀더 값 -1.

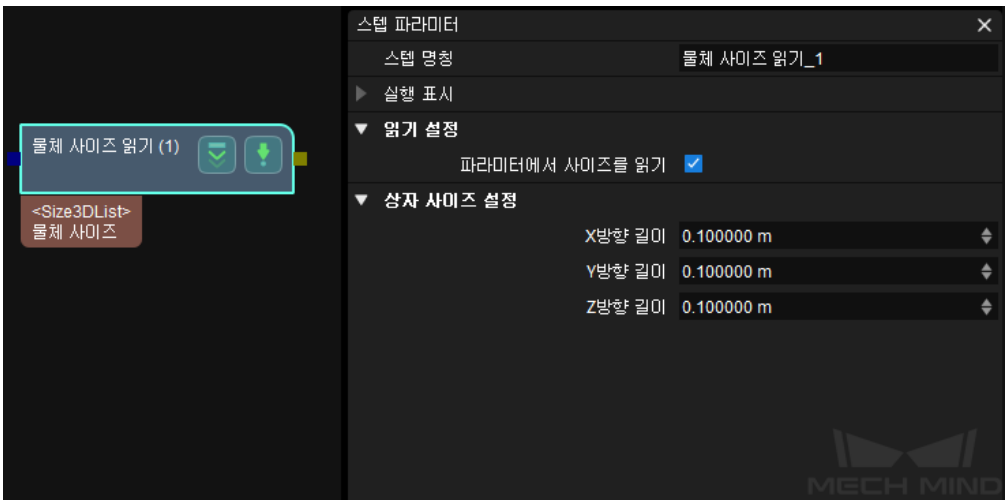
예: DO 신호 값은 각각 1, 3, 5, 6 입니다.

1	3	5	6	-1	-1	-1	-1	...	-1	-1
1 위	2 위	3 위	4 위	5 위	6 위	7 위	8 위	...	63 위	64 위

501 명령어——Mech-Vision 에 물체 치수를 입력하기

이 파라미터는 물체 크기를 Mech-Vision 프로젝트로 동적으로 입력하는 데 사용됩니다. Mech-Vision 프로젝트를 시작하기 전에 물체 크기를 확인하십시오.

Mech-Vision 프로젝트에는 read_object_dimensions 스텝이 있어야 합니다. 이 스텝 파라미터 **파라미터에서 사이즈를 읽기** 는``True``로 설정해야 합니다.

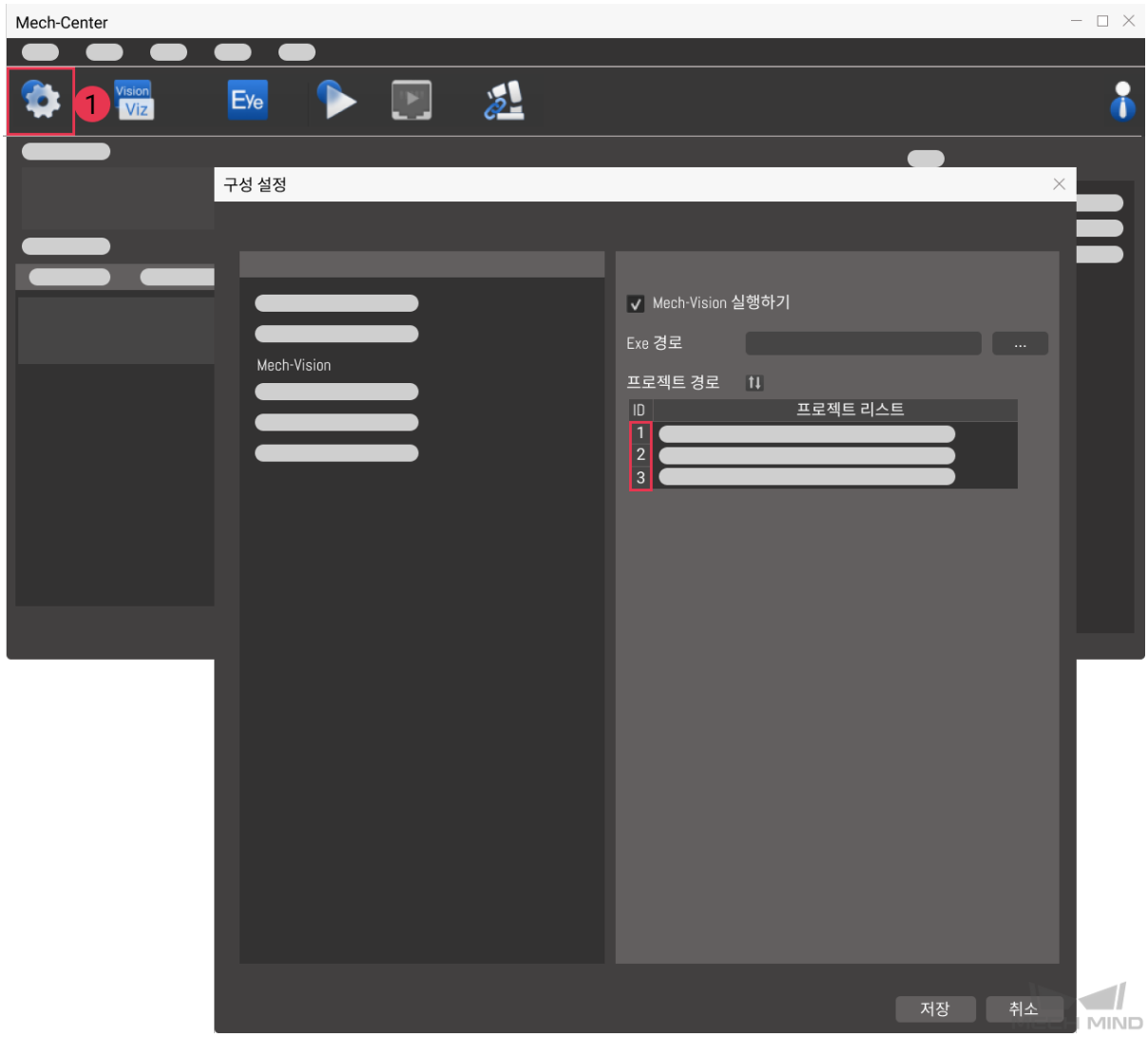


전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 501	2.0
Mech-Vision 프로젝트 번호	8.0
[길이, 너비, 높이]	68.0

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



길이, 너비, 높이

Mech-Vision 프로젝트에 전달된 물체 사이즈입니다. 사이즈 값은 read_object_dimensions 스텝에서 읽습니다.

딘위: 밀리미터 (mm)

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령이 정상적으로 실행되면 **1108** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

502 명령어——Mech-Viz 에 TCP 를 전송하기

이 명령어는 Mech-Viz 프로젝트에 로봇 TCP 를 동적으로 입력하는 데 사용됩니다. 로봇의 TCP 를 읽는 태스크는 outer_move 입니다.

템플릿 프로젝트를 기반으로 Mech-Viz 프로젝트를 구축하세요. 템플릿 프로젝트 경로는 Mech-Center 설치 디렉터리 아래의 `tool/viz_project/outer_move` 입니다.

outer_move 태스크를 작업 흐름의 올바른 위치에 배치합니다.

이 명령은*201* **명령어——***Mech-Viz* **프로젝트를 시작하기** 를 실행하기 전에 실행해야 합니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 502	2.0
TCP	80.0

TCP

outer_move 태스크 목표점의 로봇 TCP 데이터를 설정하는 데 사용됩니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

명령어가 정상적으로 실행되면 **2107** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

901 명령어——소프트웨어 상태를 획득하기

이 명령은 소프트웨어의 실행 상태 (Mech-Vision, Mech-Viz, Mech-Center) 를 확인하는 데 사용됩니다.

현재 이 명령은 Mech-Vision 이 프로젝트 실행을 시작할 수 있는지 여부만 확인하는 것을 지원합니다.

전송한 명령어 파라미터

파라미터	DB 옵셋
명령어 901	2.0

파라미터 설명: 파라미터가 없습니다.

반환한 데이터 파라미터

파라미터	DB 옵셋
상태 코드	200.0

상태 코드

시스템 자체 점검 상태. **1101** 상태 코드는“Mech-Vision 프로젝트가 준비됨”이고 다른 상태 코드는“Mech-Vision 프로젝트가 준비되지 않음”입니다. 현재 이 명령은 Mech-Vision 프로젝트가 준비되었는지 확인하는 데만 사용할 수 있습니다.

3.3.4 PROFINET

PROFINET 프로토콜을 기반으로 하는 표준 인터페이스를 통해 메크마인드 소프트웨어 시스템은 TIA Portal 소프트웨어를 사용하여 Siemens SIMATIC S7 PLC 와 통신할 수 있습니다.

자세한 내용은 PROFINET - Siemens SIMATIC S7 PLC 을 참조하십시오.

통신 프로토콜

Mech-Center **에서** *PLC* **로**

- *Control_Output*
  - *Command_Complete*
  - *Data_Ready*
  - *Exposure_Complete*
  - *Trigger_Acknowledge*
- *Heartbeat*
- *Status_Code*
- *Calib_Cam_Status*
- *Send_Pose_Num*
- *Visual_Point_Index*
- *DO_List*
- *Notify_Message*
- *Send_Pose_Type*
- *Target_Pose*
- *Target_Label*
- *Target_Speed*
- *Ext_Output_Data*

PLC **에서** *Mech-Center* **로**

- *Control_Input*
  - *Reset_Exposure*
  - *Data_Acknowledge*

  - *Reset_Notify*
  - *Trigger*
  - *Comm_Enable*
- *Command*
- *Calib_Rob_Status*
- *Vision_Proj_Num*
- *Vision_Recipe_Num*
- *Viz_Task_Name*
- *Viz_Task_Value*
- *Req_Pose_Num*
- *Req_Pose_Type*
- *Robot_Pose_JPS*
- *Robot_Pose_TCP* (**이름은** *Robot_Pose_Flange* **로 수정되었음**)
- *Ext_Input_Data*

Mech-Center 에서 PLC 로

Control_Output

Bit(비트)	데이터
7	/
6	/
5	/
4	명령 실행 완료 (부울 값)
3	데이터가 업데이트됨 (부울 값)
2	카메라 노출 완료 (부울 값)
1	시스템이 성공적으로 트리거됨 (부울 값)
0	하트비트 (부울 값)

Command_Complete

이 신호는 명령 실행이 완료되었음을 나타내는 데 사용되며 사용자는 포트에서 반환된 상태 코드 및 기타 데이터를 읽을 수 있습니다. 102 및 205 명령은 포즈 데이터 전송의 마지막 세트가 완료된 경우에만 이 모듈의 신호가 높게 설정됩니다.

Data_Ready

이 신호는 포즈 데이터를 읽을 수 있음을 나타내는 데 사용됩니다. 102 또는 205 명령이 여러 세트의 로봇 포즈 데이터를 수신할 때 특별히 사용됩니다.

Exposure_Complete

카메라 노출이 완료되면 이 신호가 높게 설정됩니다. 캡처한 작업물 또는 EIH 로봇이 촬영 위치에서 이동할 수 있음을 나타내는 데 사용됩니다.

Trigger_Acknowledge

Trigger_Acknowledge =1 의 경우, 비전 시스템이 Trigger 신호에 의해 성공적으로 트리거되었음을 의미합니다. 이 신호는 트리거 신호가 재설정될 때까지 높은 상태로 유지됩니다.

Heartbeat

시스템의 하트비트는 1 초마다 한번 반전됩니다.

Status_Code

상태 코드, INT32.

비전 시스템에서 반환된 실행 상태 코드이며 정상 상태 및 오류 코드를 포함합니다.

Calib_Cam_Status

캘리브레이션 진행 상태, INT8.

701 명령어 캘리브레이션 전용. 0: 캘리브레이션 진행 중. 1: 캘리브레이션 완료.

Send_Pose_Num

전송하는 포즈의 수량, INT8. 이번에 명령어를 실행함으로써 전송하는 포즈의 수량.

Visual_Point_Index

" 비전 이동" 태스크의 목표점이 경로 전반에서의 위치. " 비전 이동" 는 비전 포인트 (물체를 피킹하는 포인트) 로 이동하는 이동 태스크입니다.

예를 들어 계획된 경로는" 이동 _1" " 이동 _2" " 비전 이동" " 이동 _3" 태스크들로 구성되면" 비전 이동" 태스크의 위치는 3 입니다.

경로에" 비전 이동" 태스크가 없으면 이 파라미터의 값은 0 입니다.

데이터 유형: INT8.

DO_List

여러 개 빨판 파티션 또는 어레이 그리퍼를 컨트롤하는 INT8 DO 신호 (총 64 개).

Byte(바이트)	Bit 0-7
0	DO 리스트 0　신호 0-7
1	DO 리스트 1　신호 8-15
2	DO 리스트 2　신호 16-23
3	DO 리스트 3　신호 24-31
4	DO 리스트 4　신호 32-39
5	DO 리스트 5　신호 40-47
6	DO 리스트 6　신호 48-55
7	DO 리스트 7　신호 56-63

Notify_Message

Mech-Viz/Mech-Vision 의" 알림" 스텝/태스크에서 보내 온 자체 정의한 정수인 메시지.
정수를 형식으로 하는 메시지, INT32.

Send_Pose_Type

전송한 포즈 유형, INT8.

- 1　로봇 관절 각도 JPs.
- 2　로봇 공구중심점 TCP.

Target_Pose

로봇 TCP 또는 JPs 형식의 목표점 로봇 포즈.
3D 좌표와 오일러 각으로 표시되는 포즈의 데이터 구조는 아래와 같습니다.

[X, Y, Z, A, B, C]

로봇 관절 각도 JPs 로 표시되는 포즈에 최대 6 개의 관절 각도를 포함할 수 있습니다.

[J1, J2, J3, J4, J5, J6]

Byte(바이트)	Bit 0-7
0-3	목표점의 X 좌표 또는 관절 각도 J1, INT32.
4-7	목표점의 Y 좌표 또는 관절 각도 J2, INT32.
8-11	목표점의 Z 좌표 또는 관절 각도 J3, INT32.
12-15	목표점의 각도 A 또는 관절 각도 J4, INT32.
16-19	목표점의 각도 B 또는 관절 각도 J5, INT32.
20-23	목표점의 각도 C 또는 관절 각도 J6, INT32.

Target_Label

보낸 포즈에 해당하는 레이블입니다. 값은 음이 아닌 정수입니다.

데이터 유형: INT32

Target_Speed

목표점과 대응하는 이동 태스크의 속도 파라미터의 백분율. 범위: 0-100.

데이터 유형: INT32

Ext_Output_Data

다른 데이터를 전송하는 데 사용되는 보류 모듈.

이 모듈은 40 바이트 (INT32[1:10], 총 10 개의 INT32 정수) 를 차지합니다.

PLC 에서 Mech-Center 로

Control_Input

Bit(비트)	데이터
7	/
6	/
5	/
4	메시지 알림 재설정 (부울 값)
3	데이터 획인 (부울 값)
2	Exposure_Complete ” 노출 완료” 를 리셋하기 (부울 값)
1	신호 트리거 (부울 값)
0	통신 활성화 (부울 값)

Reset_Exposure

Exposure_Complete ” 노출 완료” 를 리셋하기 (부울 값)

Reset_Exposure = 1 의 경우 Exposure Complete 의 값은 0 으로 설정될 것입니다.

Data_Acknowledge

데이터 확인 (부울 값) 은 102 또는 205 명령어를 실행한 후 반환된 데이터를 이미 읽어낸지 여부를 확인하는 데 사용됩니다.

Data_Acknowledge = 0, PLC 가 Mech-Center 에서 데이터를 읽어내지 않았으며 데이터는 아직 포트에 저장되어 있음을 나타냅니다.

Data_Acknowledge = 1, PLC 가 Mech-Center 에서 데이터를 읽어냈고 Mech-Center 는 다음 라운드에서 데이터를 쓸 수 있음을 나타냅니다.

Data_Acknowledge 는 Heartbeat 뒤집힐 때 또는 Data_Ready = 0 의 경우 리셋될 수 있습니다.

Reset_Notify

메시지 알림 재설정 (부울 값)

Reset_Notify = 1 인 경우 Notify Message 의 내용이 지우게 될 것입니다.

Trigger

신호 트리거 (부울 값)

Trigger = 1 인 경우 Mech-Center 는 전송한 명령어를 읽어내어 실행할 것입니다.

Mech-Center 가 트리거 신호를 수신하자마자 Trigger_Acknowledge 를 리셋할 수 있습니다.

신호의 업스트림 부분은 1 로 간주됩니다.

Comm_Enable

통신 활성화 (부울 값)

`0` 통신 불가. Mech-Center 가 트리거 신호를 무시할 것입니다.

`1` 통신 활성화. 트리거 신호가 적용되며 Mech-Center 는 명령어를 송/수신할 수 있습니다.

Command

명령어 코드, INT32.

Calib_Rob_Status

- `0` 캘리브레이션 시작.
- `1` 로봇은 이미 전송된 가장 새로운 캘리브레이션 포인트 위치에 이동했습니다.
- `2` 로봇은 전송된 가장 새로운 캘리브레이션 포인트 위치에 도달하지 못했습니다.

데이터 유형: INT8.

Vision_Proj_Num

Mech-Center 에 등록된 Mech-Vision 의 프로젝트 번호. 즉 Mech-Center **구성 설정** → *Mech-Vision* 에서 해당 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그함으로써 조정할 수 있습니다.

데이터 유형: INT8.

Vision_Recipe_Num

Mech-Vision 프로젝트에 있는 레시피 템플릿의 번호 (양의 정수여야 함). **프로젝트 도우미** → **파라미터 레시피** 버튼을 클릭하여 파라미터 레시피 편집기 화면으로 들어갑니다. 번호 범위: 1~99.

데이터 유형: INT8.

Viz_Task_Name

명령어와 관련된 Mech-Viz 태스크의 ID. 태스크의 파라미터에서 읽어내거나 설정할 수 있습니다.

데이터 유형: INT8.

Viz_Task_Value

Mech-Viz 분기 태스크의 아웃 포트의 번호 또는 Mech-Viz 태스크의 index 파라미터를 위해 설정한 수치.

데이터 유형: INT8.

Req_Pose_Num

Mech-Vision 에서 요청한 비전 포인트 수량

`0` Mech-Vision 의 비전 결과에서 사용할 수 있는 모든 비전 포인트를 요청합니다.

데이터 유형: INT8.

Req_Pose_Type

요청한 로봇 포즈 유형.

- `0` 이미지를 캡처할 필요가 없을 때의 로봇 포즈 (Eye-To-Hand 모드)
- `1` 전송한 이미지를 캡처할 때의 로봇 포즈는 JPs 관절 각도 형식입니다.
- `2` 전송한 이미지를 캡처할 때의 로봇 포즈는 플랜지 포즈 형식입니다.

데이터 유형: INT8.

Robot_Pose_JPS

이미지를 캡처할 때의 로봇 관절 각도 JPs.

모듈을 설정하기 전에 JPs 데이터에 10000 을 곱하십시오.

JPs 에 최대 6 개의 관절 각도 데이터 (6 개의 INT32 정수) 가 포함됩니다.

[J1, J2, J3, J4, J5, J6]

Byte(바이트)	Bit 0-7
0-3	로봇의 현재 J1 관절 각도 INT32
4-7	로봇의 현재 J2 관절 각도 INT32
8-11	로봇의 현재 J3 관절 각도 INT32
12-15	로봇의 현재 J4 관절 각도 INT32
16-19	로봇의 현재 J5 관절 각도 INT32
20-23	로봇의 현재 J6 관절 각도 INT32

Robot_Pose_TCP (이름은 Robot_Pose_Flange 로 수정되었음)

이미지를 캡처하는 데 사용되는 로봇 플랜지 포즈.

모듈을 설정하기 전에 포즈 데이터에 10000 을 곱하십시오.

플랜지 포즈에는 3D 좌표 (X　Y　Z) 및 오일러 각 (A　B　C) 총 6 개의 INT32 정수가 포함됩니다.

Byte(바이트)	Bit 0-7
0-3	로봇의 현재 X 좌표 INT32
4-7	로봇의 현재 Y 좌표 INT32
8-11	로봇의 현재 Z 좌표 INT32
12-15	로봇의 현재 A 각도 INT32
16-19	로봇의 현재 B 각도 INT32
20-23	로봇의 현재 C 각도 INT32

Ext_Input_Data

다른 데이터를 전송하는 데 사용되는 보류 모듈.

이 모듈은 40 바이트 (INT32[1:10], 총 10 개의 INT32 정수) 를 차지합니다.

Profinet 명령어 설명

- *101* 명령어——*Mech-Vision* 프로젝트를 시작하기
- *102* 명령어——비전 목표점을 획득하기
- *103* 명령어——*Mech-Vision* 레시피를 전환하기
- *201* 명령어——*Mech-Viz* 프로젝트를 시작하기
- *202* 명령어——*Mech-Viz* 프로젝트를 정지하기
- *203* 명령어——*Mech-Viz* 분기를 선택하기
- *204* 명령어——이동 인덱스를 설정하기
- *205* 명령어——계획된 경로를 획득하기
- *206* 명령어——*DO* 신호 리스트를 획득하기
- *501* 명령어——*Mech-Vision* 에 물체 치수를 입력하기
- *502* 명령어——*Mech-Viz* 에 *TCP* 를 전송하기
- *901* 명령어——소프트웨어 상태를 획득하기

101 명령어——Mech-Vision 프로젝트를 시작하기

기능 소개

이 명령어는 Mech-Vision 프로젝트를 시작하고 사진 캡처 및 비전 인식을 수행하는 데 사용됩니다.

프로젝트가 Eye In Hand 모드인 경우 이 명령은 로봇이 이미지를 캡처했을 때의 포즈를 프로젝트로 전송합니다.

이 명령은 Mech-Vision 만 사용하는 시나리오에서 사용됩니다.

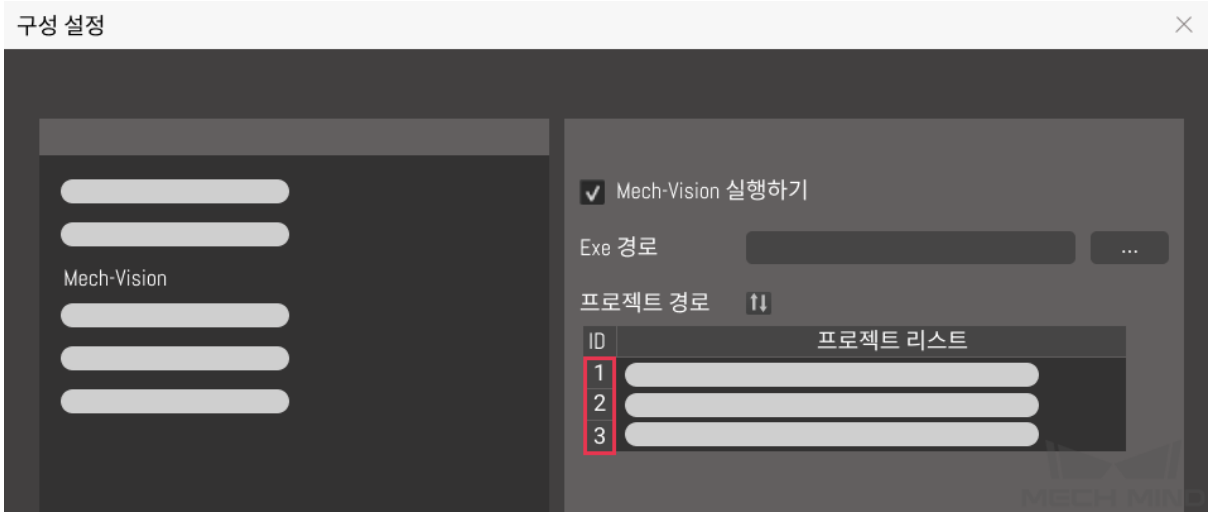
전송한 명령어 파라미터

파라미터	설명
Vision_Proj_Num	Mech-Vision 프로젝트 번호
Req_Pose_Num	비전 포인트 수량의 목표값
Robot_Pose_Type	로봇 포즈 유형
Robot_Pose_JPS / Robot_Pose_TCP *	로봇 포즈

* Robot_Pose_TCP 의 이름은 Robot_Pose_Flange 로 바꾸게 될 것입니다. 이미지를 캡처할 때 로봇 포즈는 TCP 의 형식으로 표시되지 못하기 때문입니다.

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



비전 포인트 수량의 목표값

Mech-Vision 에서 획득되길 바라는 비전 포인트의 수량입니다. 비전 포인트 정보에는 비전 포즈 및 포즈와 대응하는 포인트 클라우드, 레이블, 크기 조정 등이 포함됩니다.

- 0 Mech-Vision 프로젝트의 인식 결과에서 모든 비전 포인트를 획득합니다.
- 0 보다 큰 정수 ( ) Mech-Vision 프로젝트의 인식 결과에서 지정된 수의 비전 포인트를 획득합니다.

  - 총 비전 포인트 수가 이 파라미터의 값보다 작으면 인식 결과의 모든 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 크거나 같으면 이 파라미터에 지정된 비전 포인트 수를 획득합니다.

힌트: 비전 포인트를 획득하는 명령은 102 명령어입니다.

로봇 포즈 유형

이 파라미터는 실제 로봇의 포즈가 Mech-Vision 에 전달되는 형식을 지정합니다. 파라미터 범위: 0~2.

- `0` 이 명령은 로봇 포즈를 비전 시스템에 전달할 필요가 없습니다. 프로젝트가 Eye to Hand 모드가 아닌 경우 이미지를 촬영하는 것은 로봇의 포즈와 관련이 없으며 Mech-Vision 은 로봇의 포즈가 필요하지 않습니다.
- `1` 로봇 포즈는 JPs 관절 각도의 형식으로 전달됩니다.
- `2` 로봇 포즈는 플랜지 포즈로 전달됩니다.

로봇 포즈

이 파라미터는 Eye In Hand 모드에서 필요한 로봇 포즈이며 로봇 포즈는 JPs 관절 각도 또는 플랜지 포즈의 형식입니다. 포즈의 형태는 **로봇 포즈 유형** 에 의해 결정됩니다.

주의: JPs 및 플랜지 포즈의 부동 소수점 데이터는 10000 을 곱한 다음 Robot_Pose_JPS 또는 Robot_Pose_TCP 모듈 (나중에 Robot_Pose_Flange 로 이름이 바뀔 여정) 로 설정해야 합니다.

반환된 데이터 파라미터

`101, 상태 코드`

상태 코드

명령이 정상적으로 실행되면 **1102** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

102 명령어——비전 목표점을 획득하기

101 **명령어**——*Mech-Vision* **프로젝트를 시작하기** 뒤에 사용되며, 이 명령어를 사용하여 Mech-Vision 의 인식 결과 (비전 포인트) 과 대응하는 로봇 포즈 및 레이블을 획득하며 로봇 포즈의 형식은 TCP 입니다.

Mech-Center 가 자동으로 비전 포인트의 정보를 해당한 로봇 TCP 로 전환할 것입니다. 구체적인 과정은 아래와 같습니다.

- 비전 포인트 정보에 포함한 포즈를 Y 축을 중심으로 180° 를 회전시킵니다.
- 해당 로봇 모델의 기준 좌표계 정의에 로봇 베이스의 높이가 포함된지를 인식하고 수직 방향에서의 옵셋을 추가합니다.

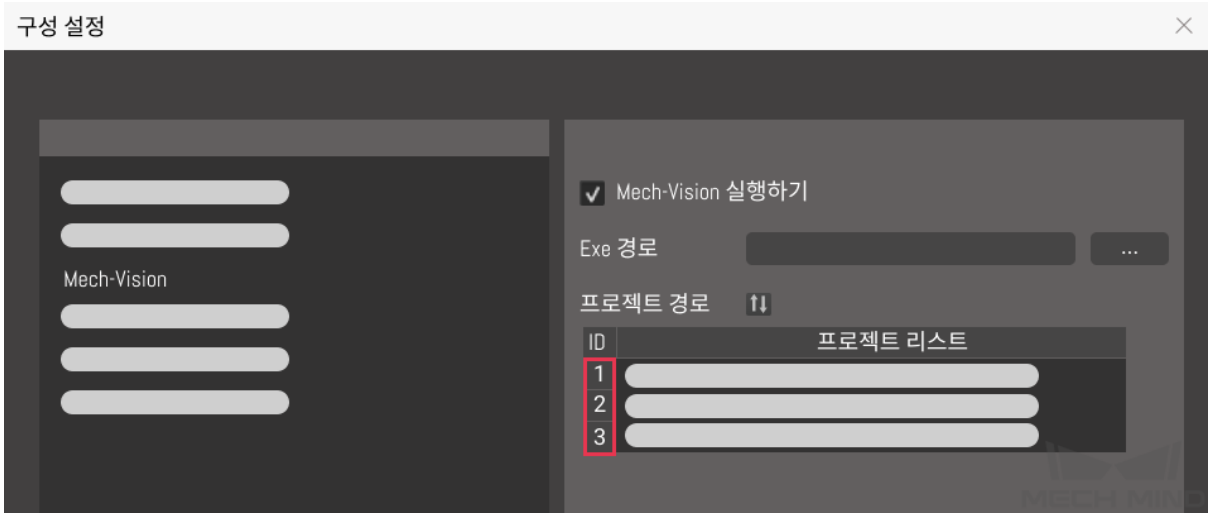
힌트: Profinet 102 명령어를 한번 실행하면 수신되는 TCP 수의 상한의 기본값은 20 입니다. 만약 획득할 TCP 의 수가 20 보다 크면 이 명령을 여러 번 실행하여 필요한 모든 TCP 를 획득합니다.

전송한 명령어 파라미터

파라미터	설명
Command	102
Vision_Proj_Num	Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



반환된 데이터 파라미터

파라미터	설명
Status Code	상태 코드
Send_Pose_Num	TCP 수량
Send_Pose_Type	포즈 유형
Target_Pose	이번에 획득한 모든 TCP
Target_Label	이번에 획득한 모든 레이블

상태 코드

명령어가 정상적으로 실행되면 **1100** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

이 명령이 호출될 때 Mech-Vision 결과가 반환되지 않은 경우 Mech-Center 는 로봇에 반환하기 전에 Mech-Vision 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초이며 제한 시간을 초과하면 타임아웃 오류 상태 코드를 반환합니다.

TCP 수량

이 명령을 실행하여 얻은 TCP 의 수량입니다.

포즈 유형

Mech-Center 가 Mech-Vision 에서 반환된 비전 포인트에 있는 대상 물체의 포즈를 해당한 TCP 로 자동으로 전환합니다.

이 파라미터의 기본값이 2 이며 포즈의 유형은 TCP 임을 나타냅니다.

이번에 획득한 모든 TCP

단일한 포즈에 포함된 정보는 포즈의 좌표 (XYZ) 및 방향을 나타내는 오일러 각 (ABC) 입니다.

이번에 획득한 모든 레이블

포즈에 해당하는 정수 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다. 프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

주의: 포즈의 송수신은통신 제어 프로세스 를 참조하십시오. Data_ready Data_Acknowledge Command_Complete 신호를 사용하여 프로세스 제어를 합니다.

103 명령어——Mech-Vision 레시피를 전환하기

Mech-Vision 프로젝트의 파라미터 레시피를 전환합니다.

Mech-Vision 의 스텝에 대한 파라미터 설정은 파라미터 레시피를 전환하여 조정할 수 있습니다.

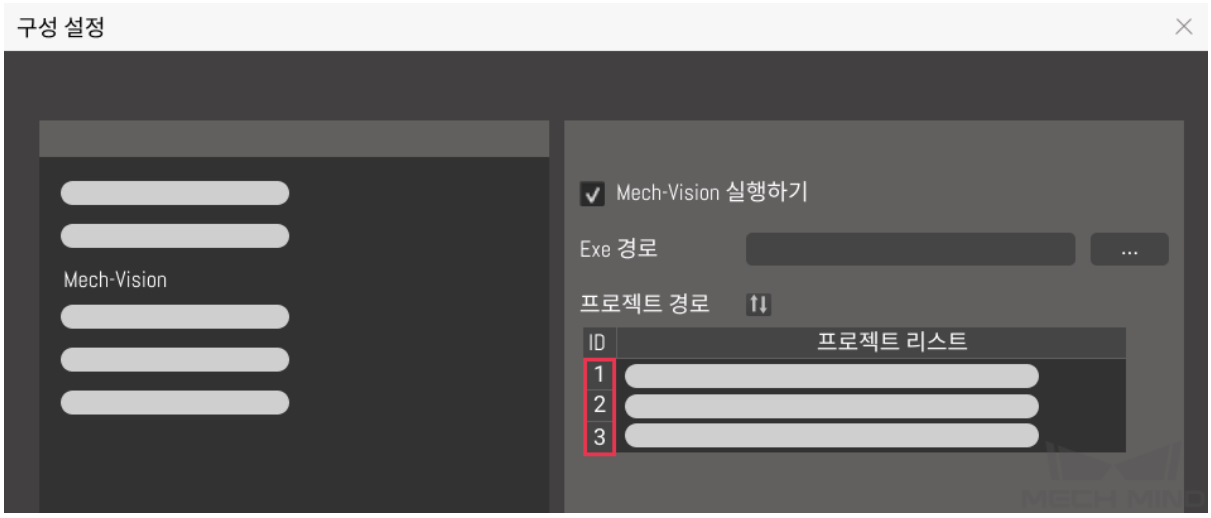
파라미터 레시피 조정과 관련된 파라미터에는 일반적으로 포인트 클라우드 매칭 모델, 이미지 매칭 모델, ROI, 믿음도 역치 등이 포함됩니다.

전송한 명령어 파라미터

파라미터	설명
Command	103
Vision_Proj_Num	Mech-Vision 프로젝트 번호
Vision_Recipe_Num	Mech-Vision 레시피 번호

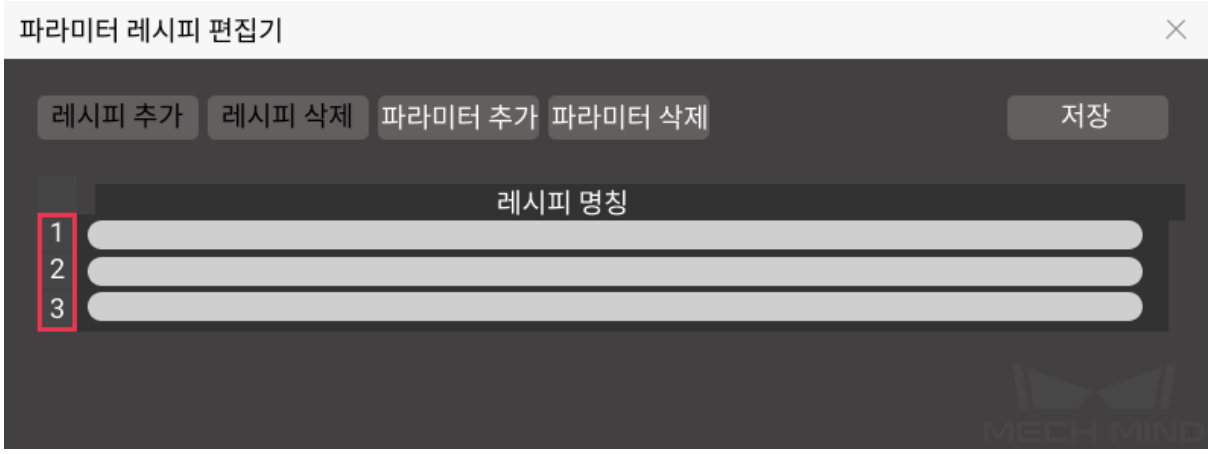
Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



Mech-Vision 레시피 번호

Mech-Vision 프로젝트의 레시피 템플릿 번호는 양의 정수 ( ) 입니다. **프로젝트 도우미** → **파라미터 레시피** 를 클릭하여 파라미터 레시피 편집기로 들어갑니다. 번호의 범위: 1~99.



반환된 데이터 파라미터

상태 코드

명령이 정상적으로 실행되면 **1107** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

201 명령어——Mech-Viz 프로젝트를 시작하기

Mech-Vision 과 Mech-Viz 를 모두 사용하는 시나리오에 사용됩니다. 이 명령은 Mech-Viz 프로젝트를 실행하고 해당 Mech-Vision 프로젝트를 시작하는 데 사용되며 Mech-Viz 는 Mech-Vision 의 비전 결과를 기반으로 로봇의 이동 경로를 계획합니다.

시작할 Mech-Viz 프로젝트에서 **자동 로드하기** 를 선택해야 합니다.

Mech-Center 설치 디렉터리 (tool/viz_project) 폴더에 일부 전형적인 응용 프로그램 프로젝트 모델이 저장되어 있으며 사용자는 모델을 기반으로 수정할 수 있습니다.

표준 인터페이스용 Mech-Viz 샘플 프로젝트는**표준 인터페이스는 *Mech-Viz* 샘플 프로젝트 사용** 에 자세히 설명되어 있습니다.

전송한 명령어 파라미터

파라미터	설명
Command	201
Robot_Pose_Type	포즈 유형
Robot_Pose_JPS	로봇 관절 각도

포즈 유형

로봇의 포즈 유형. 파라미터 범위: 0~1.

0

- Mech-Viz 는 현재 실제 로봇의 포즈를 읽을 필요가 없습니다. 이 명령은 포즈를 보내지 않습니다. 즉, 만약 프로젝트가 Eye to Hand 모드인 경우 이미지 캡처는 로봇 포즈와 아무 관련이 없으며 프로젝트는 로봇의 이미지 캡처 포즈를 읽을 필요가 없습니다.
- 포즈 유형이 0 으로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동합니다.

1

- Mech-Viz 에 전송된 포즈는 JPs 관절 각도 형식입니다. Mech-Viz 에서 가상 로봇은 초기 포즈 (즉, 이 명령에 의해 전송된 포즈) 에서 계획된 경로의 첫 번째 목표점으로 이동합니다. TCP 형식으로 포즈를 전송할 수 없습니다.
- 포즈 유형이 1 로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs= 입력한 JPs 에서 첫 번째 이동 목표점으로 이동합니다.

힌트: 시나리오에 충돌 모델이 있고 로봇이 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동하는 것을 간섭할 때 포즈 유형을 1 로 설정해야 합니다.

로봇 관절 각도

로봇의 현재 JPs 관절 각도입니다 (파라미터“포즈 유형”이 1 인 경우).

주의: JPs 의 실제 데이터는 10000 을 곱하여 정수 ( ) 로 전환한 후 Robot_Pose_JPS 모듈에 보내야 합니다.

반환된 데이터 파라미터

상태 코드

명령어가 정상적으로 실행되면 **2103** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

202 명령어——Mech-Viz 프로젝트를 정지하기

Mech-Viz 실행을 중지합니다. 만약 Mech-Viz 프로젝트가 무한 루프가 아니거나 정상적으로 중지될 수 있는 경우에는 이 명령어를 사용할 필요가 없습니다.

전송한 명령어 파라미터

파라미터	설명
Command	202

반환된 데이터 파라미터

상태 코드

명령이 정상적으로 실행되면 **2104** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

203 명령어——Mech-Viz 분기를 선택하기

이 명령은 프로젝트가 따를 분기를 지정하는 데 사용됩니다. 분기 메커니즘은 branch_by_msg 에 의해 설정되며, 이 명령은 이 태스크의 아웃 포트를 지정함으로써 분기를 지정합니다.

이 명령을 실행하기 전에*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 명령을 먼저 실행하십시오.

Mech-Viz 프로젝트가 branch_by_msg 태스크까지 실행되면 이 명령으로 지정된 아웃 포트를 기다립니다.

전송한 명령어 파라미터

파라미터	설명
Command	203
Viz_Task_ID	분기 태스크 ID
Viz_Task_Value	아웃 포트 번호

분기 태스크 ID

이 파라미터는 어느 branch_by_msg 태스크에서 분기를 선택할지를 지정하는 데 사용됩니다.

이 파라미터는 양의 정수여야 합니다. 즉 branch_by_msg 의 태스크 ID 입니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

아웃 포트 번호

이 파라미터는 프로젝트가 branch_by_msg 태스크의 어느 아웃 포트를 따를지 지정하는 데 사용됩니다. Mech-Viz 프로그램은 이 아웃 포트를 따라 계속 실행될 것입니다. 파라미터는 양의 정수입니다.

힌트:

- 아웃 포트 번호는 Mech-Viz 에서 표시되는 포트 번호 + 1 입니다. 만약 포트 번호가 0 이면 아웃 포트 번호는 1 입니다.

- 아웃 포트 번호는 1 부터 시작하는 포트 인덱스 번호입니다. 예를 들어 지정된 아웃 포트가 왼쪽에서 오른쪽으로 두 번째 포트인 경우 아웃 포트 번호는 2 입니다.

반환된 데이터 파라미터

상태 코드

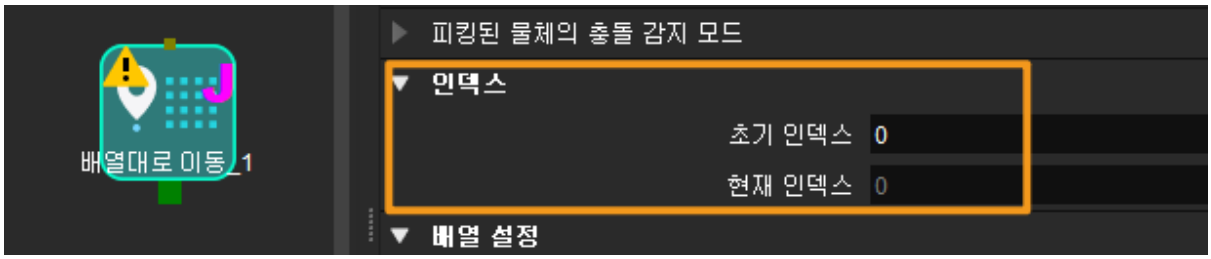
명령어가 정상적으로 실행되면 **2105** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

204 명령어——이동 인덱스를 설정하기

태스크의 인덱스 파라미터 값을 설정하는 명령어입니다. 이러한 유형의 태스크는 일반적으로 연속적이거나 개별적으로 지정된 이동 또는 기타 작업에 사용됩니다.

인덱스 파라미터가 있는 태스크에는 [순서대로 이동], [배열대로 이동], [자체 정의한 파렛트 패턴] 및 [미리 설정된 파렛트 패턴] 등이 있습니다.

이 명령을 실행하기 전에*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 명령을 실행하십시오.



전송한 명령어 파라미터

파라미터	설명
Command	204
Viz_Task_ID	태스크 ID
Viz_Task_Value	인덱스 값

태스크 ID

이 파라미터는 인덱스 파라미터를 설정해야 하는 태스크를 지정합니다.

이 파라미터의 값은 양의 정수, 즉 인덱스 대기 중인 태스크의 태스크 ID 여야 합니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다. 범위: 1~99.

인덱스 값

여기 인덱스 파라미터 값은 Mech-Viz 에 표시된 현재 인덱스 값에 1 을 더한 값입니다.

반환된 데이터 파라미터

상태 코드

명령어가 정상적으로 실행되면 **2106** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

205 명령어——계획된 경로를 획득하기

이 명령은 Mech-Viz 에서 계획한 경로를 가져오는 데 사용됩니다. 계획된 경로는*201* **명령어——***Mech-Viz* **프로젝트를 시작하기** 실행 후 가져와야 합니다.

전송한 명령어 파라미터

파라미터	설명
Command	205
Req_Pose_Type	목표점 유형

목표점 유형

이 파라미터는 Mech-Viz 가 반환할 목표점의 형식을 지정하는 데 사용됩니다.

- 1 목표점은 로봇 관절 각도 (JPs) 의 형식으로 반환됩니다.
- 2 목표점은 TCP 포즈의 형식으로 반환됩니다.

반환된 데이터 파라미터

파라미터	설명
Status Code	상태 코드
Send_Pose_Num	목표점 수량
Send_Pose_Type	포즈 유형
Visual_Point_Index	**" 비전 이동" 태스크 위치**
Target_Pose	이번에 전송한 모든 목표점의 포즈
Target_Label	이번에 전송한 모든 목표점의 레이블
Target_Speed	이번에 전송한 모든 목표점의 속도

상태 코드

명령이 정상적으로 실행되면 **2100** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

힌트: 이 명령이 호출될 때 Mech-Viz 결과가 반환되지 않은 경우 (실행 중) Mech-Center 는 로봇에 반환하기 전에 Mech-Viz 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초입니다. 제한 시간을 초과하면 로봇에 타임아웃 오류를 반환합니다.

목표점 수량

이 파라미터는 이 명령에 의해 반환되는 경로의 이동 목표점 ([포즈, 레이블, 속도]) 의 수를 표시하는 데 사용됩니다.

경로에 20 개 이상의 목표점이 포함된 경우 이 명령을 여러 번 실행하십시오.

기본적인 범위: 0~20

포즈 유형

이 명령어가 보낸“Req_Pose_Type”수치와 같습니다.

- 1: JPs
- 2: TCP

” 비전 이동” 태스크 위치

” 비전 이동” 태스크의 목표점이 프로젝트 전체에서의 위치. ” 비전 이동” 태스크는 비전 포인트 (물체를 피킹하는 포인트) 로 이동하는 이동 태스크입니다.

예를 들어 계획된 경로가” 이동 _1”, ” 이동 _2”, ” 비전 이동” ” 이동 _3” 태스크로 구성된 경우” 비전 이동” 태스크 위치는 3 입니다.

경로에” 비전 이동” 태스크가 없는 경우 이 파라미터의 값은 0 입니다.

이번에 전송한 모든 목표점의 포즈

3D 좌표 및 오일러 각 또는 JPs 관절 각도. 유형은 명령어 205 중의 포즈 유형에 의해 결정됩니다.

이번에 전송한 모든 목표점의 레이블

포즈에 해당하는 정수 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력ㅎ기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다.

프로젝트에 레이블이 없는 경우 이 파라미터는 기본값이 0 입니다.

이번에 전송한 모든 목표점의 속도

vision_move 태스크 파라미터의 0 이 아닌 속도 파라미터 백분율 값입니다.

> **주의:** 포즈의 송수신은**통신 제어 프로세스** 를 참조하십시오. Data_ready Data_Acknowledge Command_Complete 신호를 사용하여 프로세스 제어를 합니다.

206 명령어——DO 신호 리스트를 획득하기

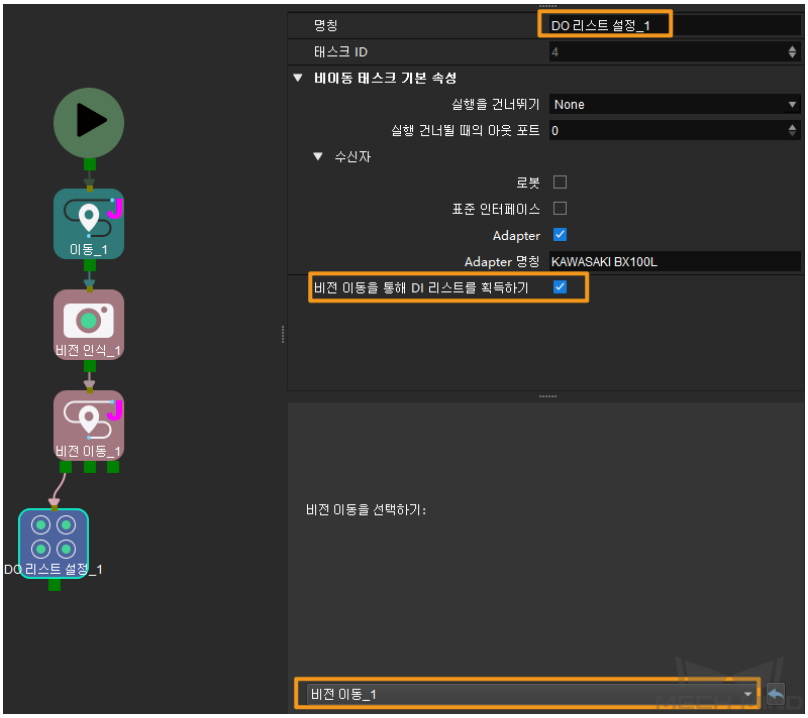
이 명령은 계획된 DO 신호 리스트를 획득하는 데 사용됩니다. DO 신호 리스트는 여러 도구 또는 빨판 파티션을 컨트롤하는 데 사용됩니다.

이 명령을 실행하기 전에 Mech-Viz 계획 경로를 얻기 위해*205* **명령** 을 실행해야 합니다.

템플릿 프로젝트에 따라 Mech-Viz 프로젝트를 구축하고 프로젝트에서 해당 빨판 구성 파일을 설정하십시오. 템플릿 프로젝트는 Mech-Center 설치 디렉터리 (tool/viz_project) 아래의 **suction_zone** 프로젝트입니다.

프로젝트의 set_do_list 태스크 파라미터에서

- ” 수신자” 에서” 표준 인터페이스” 를 선택하십시오.
- ” 비전 이동에서 DO 리스트를 획득하기” 를 선택하십시오.
- 파라미터 표시줄 하단에 DO 신호 리스트가 필요한 비전 이동 태스크를 선택하십시오.



전송한 명령어 파라미터

파라미터	설명
Command	206

반환된 데이터 파라미터

파라미터	설명
Status code	상태 코드
DO List	DO 신호 리스트

DO 신호 리스트

반환된 데이터의 파라미터 끝에 정수인 64 개의 DO 신호 값이 있습니다.

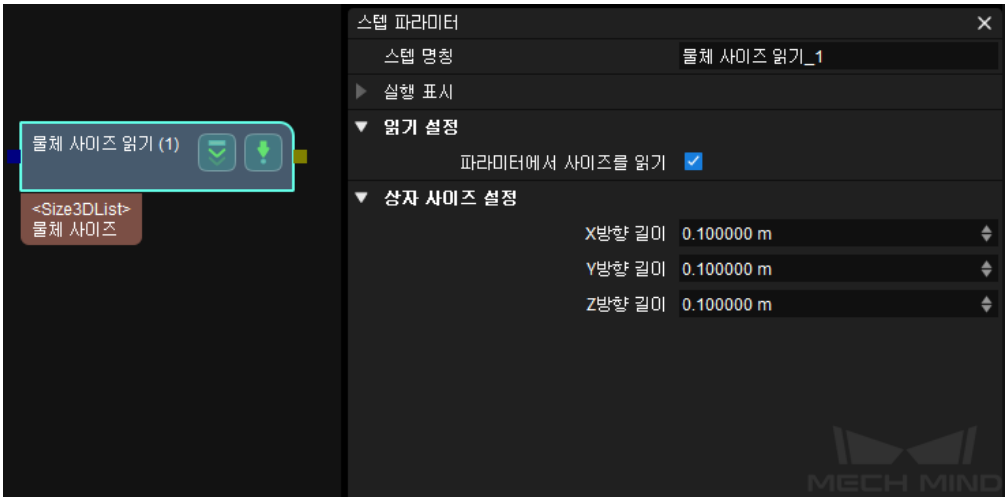
DO 신호 값 범위: 0~999.

플레이스홀더 값: -1.

501 명령어——Mech-Vision 에 물체 치수를 입력하기

이 파라미터는 물체 크기를 Mech-Vision 프로젝트로 동적으로 입력하는 데 사용됩니다. Mech-Vision 프로젝트를 시작하기 전에 물체 크기를 확인하십시오.

Mech-Vision 프로젝트에는 read_object_dimensions 스텝이 있어야 합니다. 이 스텝 파라미터 **파라미터에서 사이즈를 읽기** 는``True``로 설정해야 합니다.

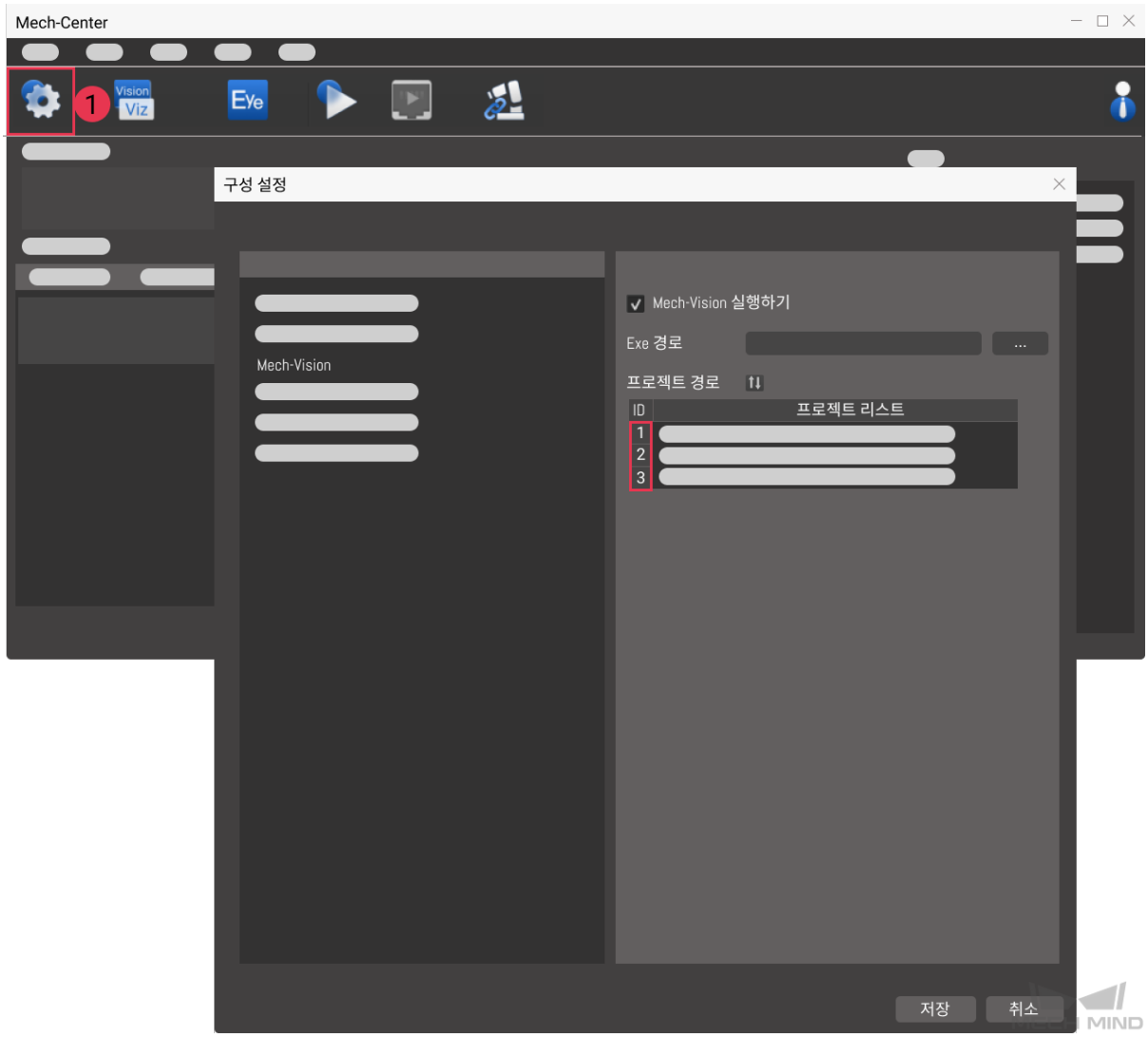


전송한 명령어 파라미터

파라미터	설명
Command	501
Vision_Proj_Num	Mech-Vision 프로젝트 번호
Ext_Input_Data	물체 치수

Mech-Vision 프로젝트 번호

Mech-Center 에서 Mech-Vision 프로젝트 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 프로젝트 경로 왼쪽에 표시되는 번호입니다. 드래그 앤 드롭하여 조정할 수 있습니다.



물체 치수

Mech-Vision 프로젝트에 전달된 물체 사이즈입니다. 사이즈 값은 read_object_dimensions 스텝에서 읽게 됩니다.

단위: 밀리미터 (mm)

반환된 데이터 파라미터

상태 코드

명령이 정상적으로 실행되면 **1108** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

502 명령어——Mech-Viz 에 TCP 를 전송하기

이 명령어는 Mech-Viz 프로젝트에 로봇 TCP 를 동적으로 입력하는 데 사용됩니다. 로봇의 TCP 를 읽는 태스크는 outer_move 입니다.

템플릿 프로젝트를 기반으로 Mech-Viz 프로젝트를 구축하세요. 템플릿 프로젝트 경로는 Mech-Center 설치 디렉터리 아래의 `tool/viz_project/outer_move` 입니다.

outer_move 태스크를 작업 흐름의 올바른 위치에 배치합니다.

이 명령은*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 명령을 실행하기 전에 실행해야 합니다.

전송한 명령어 파라미터

파라미터	설명
Command	502
Ext_Input_Data	로봇 TCP

로봇 TCP

outer_move 태스크 목표점의 TCP 데이터를 설정하는 데 사용됩니다.

반환된 데이터 파라미터

상태 코드

명령어가 정상적으로 실행되면 **2107** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

901 명령어——소프트웨어 상태를 획득하기

이 명령은 소프트웨어의 실행 상태 (Mech-Vision, Mech-Viz, Mech-Center) 를 확인하는 데 사용됩니다.

현재 이 명령은 Mech-Vision 이 프로젝트 실행을 시작할 수 있는지 여부만 확인하는 것을 지원합니다.

전송한 명령어 파라미터

파라미터	설명
Command	901

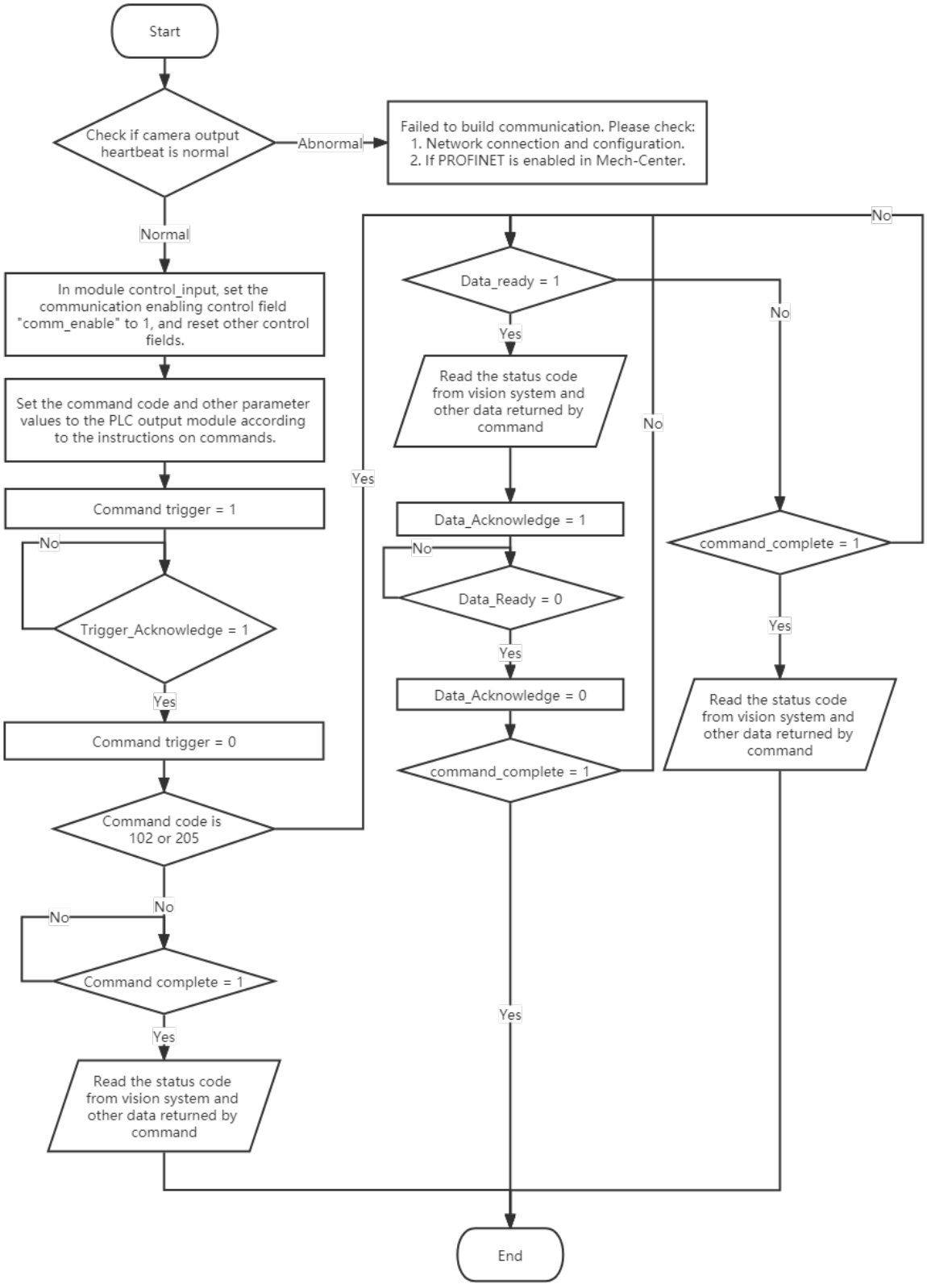
반환된 데이터 파라미터

상태 코드

시스템 자체 점검 상태. **1101** 상태 코드는“Mech-Vision 프로젝트가 준비됨”이고 다른 상태 코드는“Mech-Vision 프로젝트가 준비되지 않음”입니다. 현재 이 명령은 Mech-Vision 프로젝트가 준비되었는지 확인하는 데만 사용할 수 있습니다.

통신 제어 프로세스

Mech-Mind 비전 시스템이 PROFINET 을 이용한 제어 프로세스는 아래 그림과 같습니다.


Start
Check if camera output heartbeat is normal
Abnormal
Failed to build communication. Please check:
1. Network connection and configuration.
2. If PROFINET is enabled in Mech-Center.
Normal
In module control_input, set the communication enabling control field "comm_enable" to 1, and reset other control fields.
Set the command code and other parameter values to the PLC output module according to the instructions on commands.
Command trigger = 1
No
Trigger_Acknowledge = 1
Yes
Command trigger = 0
Command code is 102 or 205
No
No
Command complete = 1
Yes
Read the status code from vision system and other data returned by command
Yes
Data_ready = 1
No
Yes
Read the status code from vision system and other data returned by command
No
Data_Acknowledge = 1
No
Data_Ready = 0
Yes
Data_Acknowledge = 0
command_complete = 1
No
Yes
command_complete = 1
Yes
Read the status code from vision system and other data returned by command
End

3.3.5 Ethernet/IP

메크마인드 소프트웨어 시스템은 Ethernet/IP 프로토콜 기반의 표준 인터페이스를 통해 오므론 (Omron) PLC 및 키엔스 (Keyence) PLC 와 통신할 수 있습니다.

자세한 내용은 standard_interface_robot_and_plc 를 참조할수 있으며, 명령어 설명은*Profinet* **명령어 설명** 을 참조하십시오.

3.3.6 Modbus TCP

메크마인드 소프트웨어 시스템은 Modbus TCP 프로토콜을 기반으로 하는 표준 인터페이스를 통해 마스터 장치 (PLC 또는 로봇 컨트롤러) 와 통신할 수 있습니다.

구체적인 샘플 프로젝트에 대해서는 <Modbus TCP - Siemens SIMATIC S7 PLC> 및 <Modbus TCP - Mitsubishi Q 시리즈 PLC> 를 참조하십시오.

통신 구성

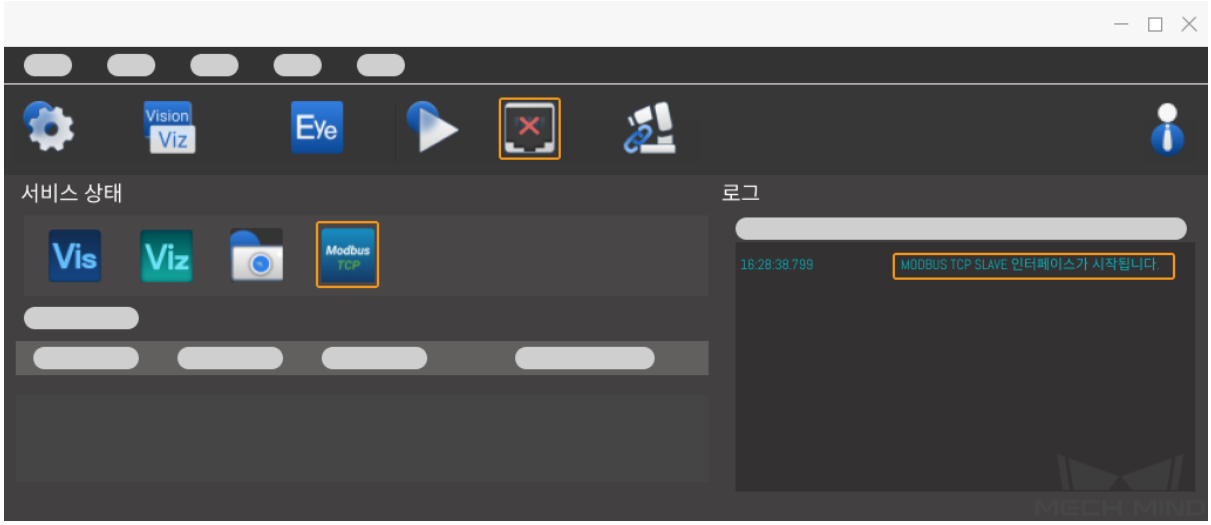
Mech-Center 는 Modbus TCP 통신을 지원하며 slave 장치로서 Poll 마스터 장치와의 데이터 통신을 위한 표준 인터페이스 옵션인 Modbus TCP SLAVE 를 제공합니다. 구체적인 구성 단계는 다음과 같습니다.

1. 마스터의 IP 주소와 동일한 네트워크 세그먼트에 있어야 하는 IPC 의 IP 주소를 구성합니다. IPC 의 **cmd 명령 프롬프트** 를 열고 (cmd 를 검색하여 입력할 수 있음) **ping xxx.xxx.xxx.xxx** (마스터의 IP 주소) 를 입력하고 IPC 와 마스터 간의 연결이 정상인지 테스트합니다.
2. Mech-Center 를 실행하여 **배포 설정** 창을 엽니다.

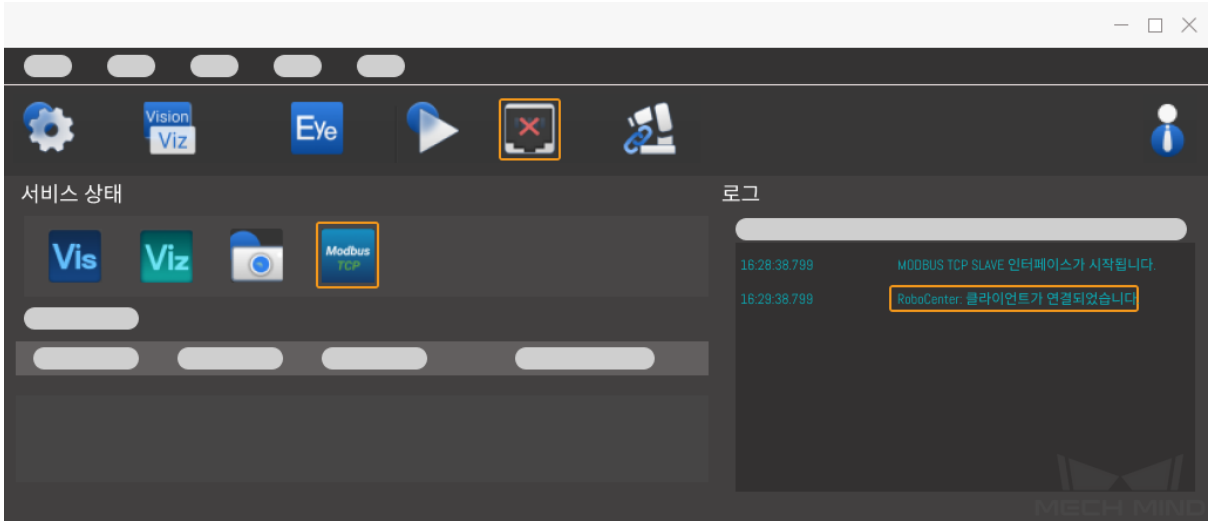
3. 왼쪽:guilabel:*Mech-Interface* 를 클릭하고 *Mech-Interface* **실행하기** → **표준 인터페이스** → *Modbus TCP SLAVE* 차례로 선택, 슬레이브 IP 는 0.0.0.0(즉, 로컬 컴퓨터의 IPv4 주소), 슬레이브 장치 주소를 설정하고 바이트 선택을 선택한 다음 마지막으로 **저장** 을 클릭합니다.

팁: 마스터의 부동 소수점 바이트 순서 요구 사항에 따라 바이트 순서를 설정합니다. 예를 들어 표준 빅 엔디안 데이터의 경우 DCBA 를 선택하고 표준 리틀 엔디안 데이터의 경우 ABCD 를 선택합니다.

4. Mech-Center 메인 창에서 **인터페이스 실행** 을 클릭하면 Modbus TCP SLAVE 서비스 아이콘이 서비스 상태 표시줄에 나타납니다.



5. Mech-Center 와 마스터 간의 통신이 성공하면 인터페이스 오른쪽의 로그 열에" 클라이언트가 연결되었습니다." 가 표시되고, Mech-Center 와 마스터 간의 통신이 실패하면 물리적 연결이 제대로 되어 있는지 확인하십시오.



레지스터 매핑 리스트

Mech-Center Modbus TCP Register Map-table(레지스터 매핑 리스트) 은 아래 표와 같습니다.

주소	내용	길이 (글자를 단위로 함)	읽기/쓰기	비고	레지스터 보유 (4x)
0x0000	명령어 트리거	1	쓰기	0: 명령어를 트리거하지 않음. 1: 명령어를 트리거 함.	40001-40053: Mech-Center 에 쓰기
0x0001	명령어	1	쓰기	기능 코드	
0x0002	포즈 유형	1	쓰기		
0x0003	포즈 개수	1	쓰기		
0x0004	Mech-Vision 프로젝트 번호	1	쓰기		
0x0005	레시피 번호	1	쓰기		
0x0006	관절 각도	12	쓰기	단위: 각도	
0x0012	TCP	12	쓰기	단위: 밀리미터 및 오일러 각	
0x001E	분기 명칭	1	쓰기		
0x001F	분기 아웃 포트	1	쓰기		
0x0020	인덱스 명칭	1	쓰기		
0x0021	인덱스 번호	1	쓰기		
0x0022	외부 상자 크기	6	쓰기	단위: 밀리미터	
0x0028	외부 입력 포즈	12	쓰기	단위: 밀리미터 및 오일러 각	
0x0033	로봇 이동 상태	1	쓰기	카메라 캘리브레이션 전용	
0x0034	보류	44	읽기		40054-40728: Mech-Center 읽기
0x0061	트리거 확인	1	읽기	0: 명령어 트리거를 수락하지 않음. 1: 명령어를 트리거를 수락함.	
0x0062	통지	1	읽기		
0x0063	하트비트	1	읽기	1Hz	
0x0064	상태 코드	1	읽기		
0x0065	포즈 상태 보내기	1	읽기	0: 999 기능 코드 수신 1: 새로운 포즈 데이터	
0x0066	포즈 수량 보내기	1	읽기		
0x0067	계획된 경로에서의 비전 포인트 위치	1	읽기		
0x0068	목표점	480	읽기		
0x0248	대상 레이블	40	읽기		
0x0270	속도 백분율	40	읽기		
0x0298	디지털 출력 신호	64	읽기		

참고: Modbus TCP 는 간단한 버스 프로토콜로 한 번에 약 100 자 정도의 읽기 또는 쓰기를 권장하며 단일 읽기 또는 쓰기의 통신 주기는 약 70ms 입니다. 가능하면 PROFINET 및 Ethernet/IP 와 같은 실시간 이더넷 프로토콜이나 지멘스 (Siemens) 에서 개발한 Snap7 통신 프로토콜 (Mech-Center Siemens PLC Client 의 표준 인터페이스에 해당) 또는 미쓰비시 (Mitsubishi) 에서 개발한 MELSEC 통신 프로토콜 (약칭: MC 프로토콜) 을 선택하십시오.

레지스터 명령어 보유

101 명령어——Mech-Vision 프로젝트 시작

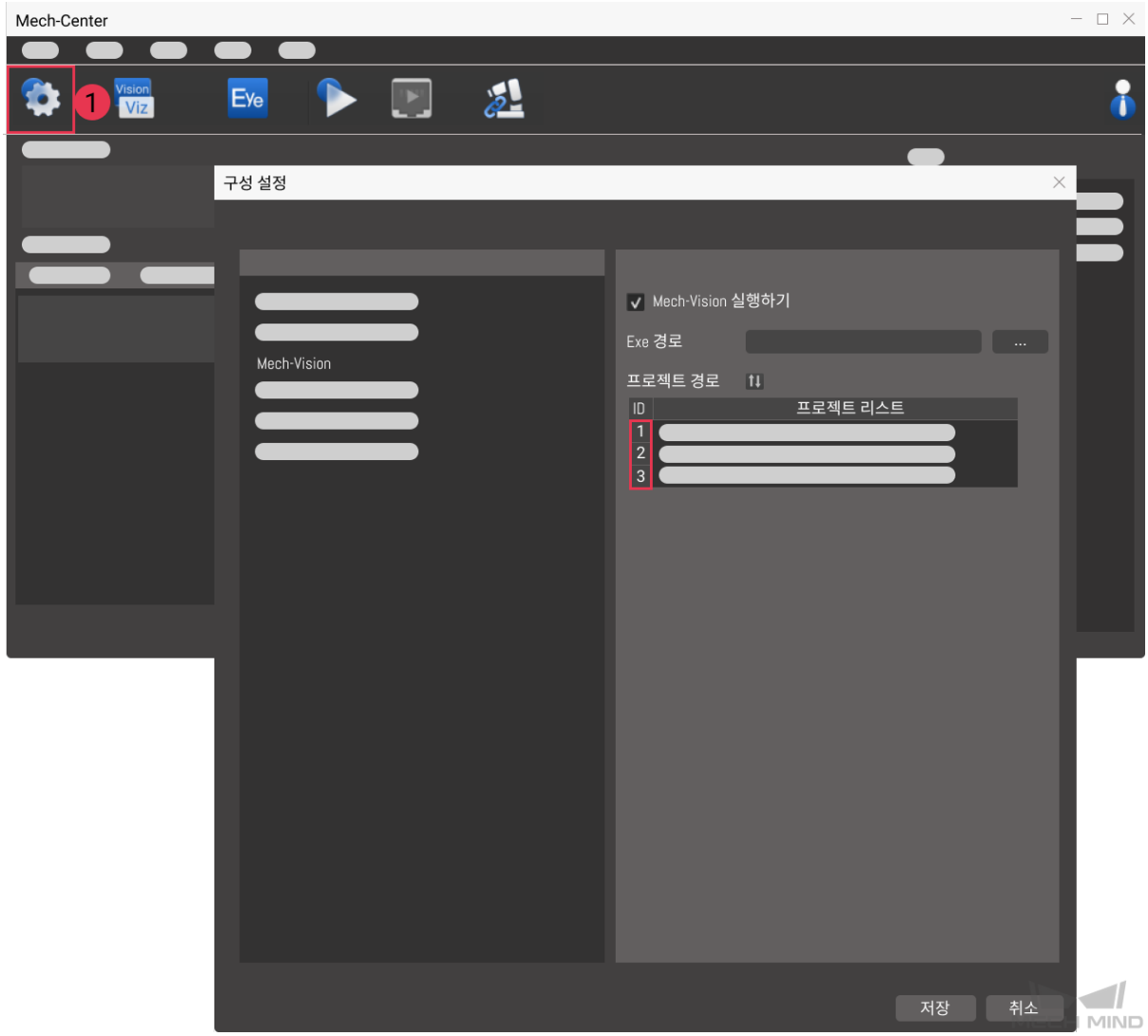
이 명령은 카메라 사진 캡처 및 비전 인식을 수행하는 데 사용되는 Mech-Vision 프로젝트를 시작합니다. 프로젝트가 Eye In Hand 모드인 경우 이 명령은 로봇 카메라 사진 캡처 포즈를 프로젝트로 전송합니다. 이 명령은 Mech-Vision 만 사용하는 시나리오에서 사용됩니다.

전송된 명령 파라미터

파라미터	주소
명령어 101	1
Mech-Vision 프로젝트 번호	4
비전 포인트 수량의 목표값	3
로봇 포즈 유형	2
로봇 포즈	6-11(JPs) 또는 12-17(플랜지 포즈)

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



비전 포인트 수량의 목표값

Mech-Vision 에서 획득되길 바라는 비전 포인트의 수량입니다. 비전 포인트 정보에는 비전 포즈 및 포즈와 대응하는 포인트 클라우드, 레이블, 크기 조정 등이 포함됩니다.

- 0 Mech-Vision 프로젝트의 인식 결과에서 모든 비전 포인트를 획득합니다.
- 0 보다 큰 정수 ( ) Mech-Vision 프로젝트의 인식 결과에서 지정된 수의 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 작으면 인식 결과의 모든 비전 포인트를 획득합니다.
  - 총 비전 포인트 수가 이 파라미터의 값보다 크거나 같으면 이 파라미터에 지정된 비전 포인트 수를 획득합니다.

힌트: 비전 포인트를 획득하는 명령은 102 명령어입니다. PLC 의 기본 설정에서는 102 명령을 한번 실행하여 최대 20 개의 비전 포인트를 획득할 수 있습니다. 획득할 비전 포인트 수가 40 보다 크면 102 명령어를 반복적으로 호출해야 합니다.

로봇 포즈 유형

이 파라미터는 실제 로봇의 포즈가 Mech-Vision 에 전달되는 형식을 지정합니다. 파라미터 범위: 0~2.

- 0 이 명령은 로봇 포즈를 비전 시스템에 전달할 필요가 없습니다. 프로젝트가 Eye to Hand 모드가 아닌 경우 이미지를 촬영하는 것은 로봇의 포즈와 관련이 없으며 Mech-Vision 은 로봇의 포즈가 필요하지 않습니다.
- 1 로봇 포즈는 JPs 관절 각도의 형식으로 전달됩니다.
- 2 로봇 포즈는 플랜지 포즈로 전달됩니다.

로봇 포즈

이 파라미터는 Eye In Hand 모드에서 필요한 로봇 포즈이며 로봇 포즈는 JPs 관절 각도 또는 플랜지 포즈의 형식입니다. 포즈의 형식은 **로봇 포즈 유형** 에 의해 결정됩니다.

로봇 포즈는 6 개 Float 유형의 숫자로 구성됩니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령이 정상적으로 실행되면 **1102** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

102 명령어——비전 목표점을 획득하기

101 **명령어**——*Mech-Vision* **프로젝트 시작** 이후에 사용되며, 이 명령어를 사용하여 Mech-Vision 의 인식 결과 (비전 포인트) 와 대응하는 로봇 포즈 및 레이블을 획득하며 로봇 포즈의 형식은 TCP 입니다.

Mech-Center 가 자동으로 비전 포인트의 정보를 해당한 로봇 TCP 로 전환할 것입니다. 구체적인 과정은 아래와 같습니다.

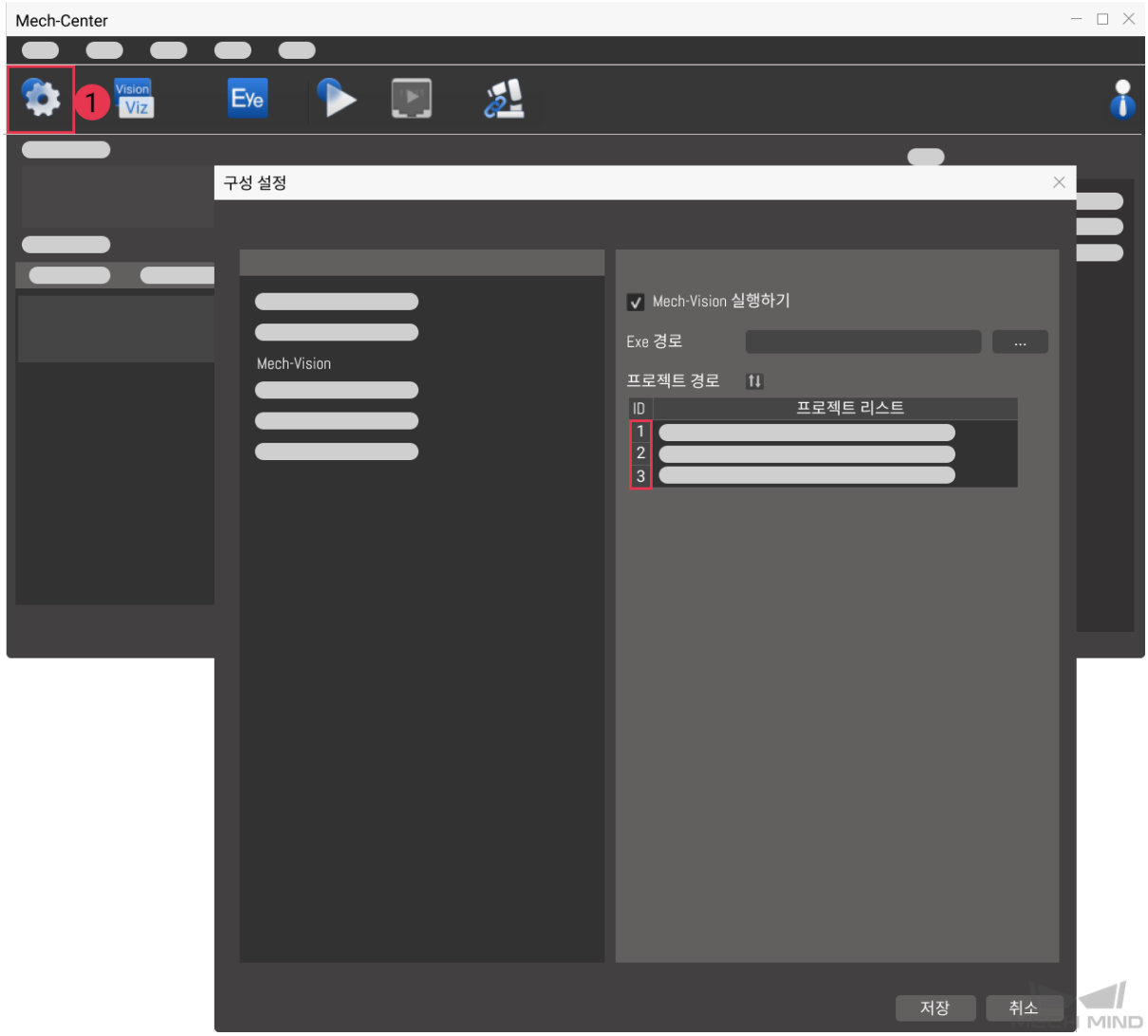
- 비전 포인트 정보에 포함한 포즈를 Y 축을 중심으로 180° 를 회전시킵니다.
- 해당 로봇 모델의 기준 좌표계 정의에 로봇 베이스의 높이가 포함된지를 인식하고 수직 방향에서의 옵셋을 추가합니다.

전송된 명령 파라미터

파라미터	주소
명령어 102	1
Mech-Vision 프로젝트 번호	4

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



반환한 데이터 파라미터

파라미터	주소
상태 코드	100
데이터 전송 상태	101
TCP 수량	102
보류 필드	/
이번에 획득한 모든 TCP	104
이번에 획득한 모든 레이블	584

상태 코드

명령이 정상적으로 실행되면 **1100** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

이 명령이 호출될 때 Mech-Vision 결과가 반환되지 않은 경우 Mech-Center 는 로봇에 반환하기 전에 Mech-Vision 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초이며 제한 시간을 초과하면 타임아웃 오류 상태 코드를 반환합니다.

데이터 전송 상태

이 파라미터는 반환된 데이터가 새로운 TCP 인지를 표시하는 데 사용됩니다.

1 반환된 데이터는 새 TCP 이며 읽을 수 있습니다.

TCP 수량

이 명령을 실행하여 얻은 TCP 의 수량입니다.

- 요청한 TCP 수가 Mech-Vision 에서 인식한 비전 포인트 수와 같거나 이상인 경우 Mech-Vision 에서 인식한 비전 포인트 수에 따라 전송됩니다.
- 요청한 TCP 수가 Mech-Vision 이 인식하는 비전 포인트 수보다 적을 경우 요청한 수만큼 전송됩니다.

보류 필드

이 필드는 사용되지 않으며 기본값은 0 입니다.

이번에 획득한 모든 TCP

단일한 TCP 에 3D 좌표 (XYZ) 및 오일러 각 (ABC) 이 포함됩니다.

이번에 획득한 모든 레이블

포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다. 프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

103 명령어——Mech-Vision 레시피를 전환하기

Mech-Vision 프로젝트 내에서 파라미터 레시피를 전환합니다.

Mech-Vision 의 스텝에 대한 파라미터 설정은 파라미터 레시피를 전환하여 조정할 수 있습니다.

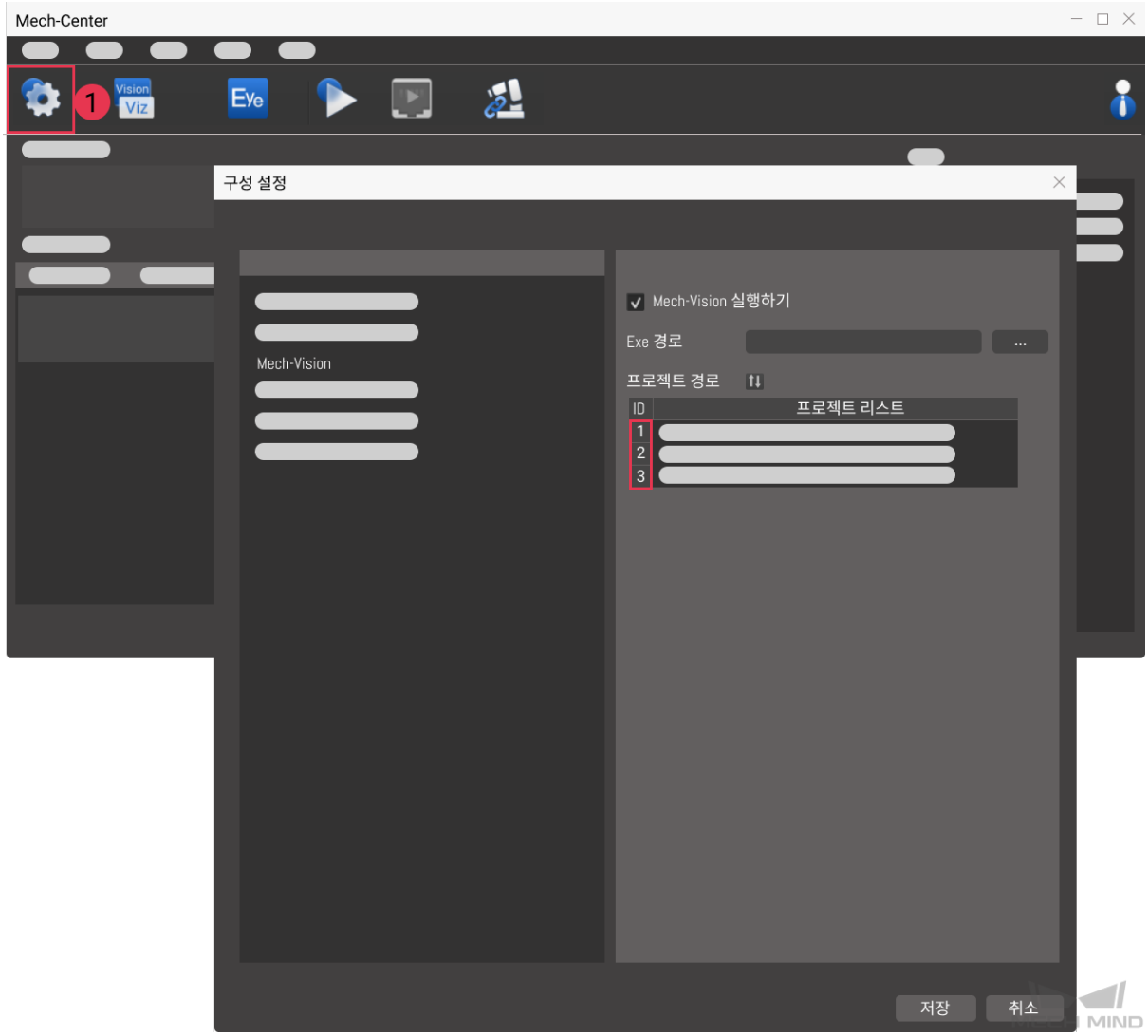
파라미터 레시피 조정과 관련된 파라미터에는 일반적으로 포인트 클라우드 매칭 모델, 이미지 매칭 모델, ROI, 믿음도 역치 등이 포함됩니다.

전송된 명령 파라미터

파라미터	주소
명령어 103	1
Mech-Vision 프로젝트 번호	4
레시피 번호	5

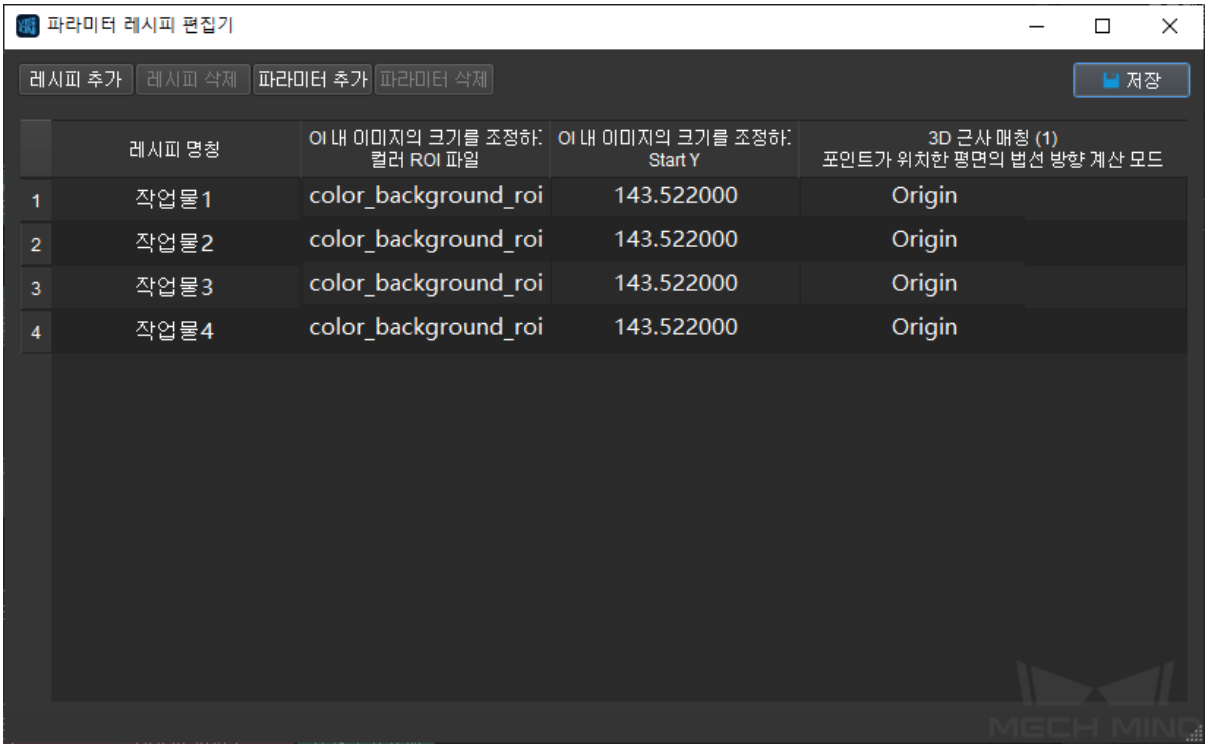
프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



레시피 번호

Mech-Vision 프로젝트의 레시피 템플릿 번호는 양의 정수 ( ) 입니다. **프로젝트 도우미** → **파라미터 레시피** 를 클릭하여 파라미터 레시피 편집기로 들어갑니다. 번호의 범위: 1~99.



반환한 데이터 파라미터

파라미터	주소
상태 코드	100

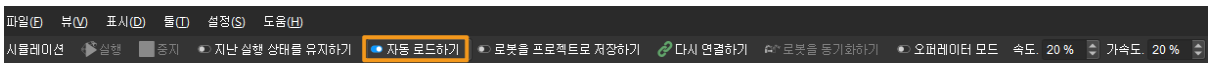
상태 코드

명령이 정상적으로 실행되면 **1107** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

201 명령어——Mech-Viz 프로젝트를 시작하기

Mech-Vision 과 Mech-Viz 를 모두 사용하는 시나리오에 사용됩니다. 이 명령은 Mech-Viz 프로젝트를 실행하고 해당 Mech-Vision 프로젝트를 시작하는 데 사용되며 Mech-Viz 는 Mech-Vision 의 비전 결과를 기반으로 로봇의 이동 경로를 계획합니다.

시작할 Mech-Viz 프로젝트에서 **자동 로드하기** 를 선택해야 합니다.



Mech-Center 설치 디렉터리의 `tool/viz_project` 폴더에 일부 전형적인 응용 프로그램 프로젝트 모델이 저장되어 있으며 사용자는 모델을 기반으로 수정할 수 있습니다.

표준 인터페이스용 Mech-Viz 샘플 프로젝트는**표준 인터페이스는 *Mech-Viz* 샘플 프로젝트 사용** 에 자세히 설명되어 있습니다.

전송된 명령 파라미터

파라미터	주소
명령어 201	1
포즈 유형	2
로봇 포즈	6

포즈 유형

로봇의 포즈 유형입니다. 파라미터 범위: 0~1.

0

- Mech-Viz 는 현재 실제 로봇의 포즈를 읽을 필요가 없습니다. 이 명령은 포즈를 보내지 않습니다. 즉, 만약 프로젝트가 Eye to Hand 모드인 경우 이미지 캡처는 로봇 포즈와 아무 관련이 없으며 프로젝트는 로봇의 이미지 캡처 포즈를 읽을 필요가 없습니다.
- 포즈 유형이 0 으로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동합니다.

1

- Mech-Viz 에 전송된 포즈는 JPs 관절 각도 및 플랜지 포즈의 형식입니다. Mech-Viz 에서 가상 로봇은 초기 포즈 (즉, 이 명령에 의해 전송된 포즈) 에서 계획된 경로의 첫 번째 목표점으로 이동합니다. TCP 형식으로 포즈를 전송할 수 없습니다.
- 포즈 유형이 1 로 설정되면 Mech-Viz 에서 가상 로봇은 초기 포즈 JPs= 입력한 JPs 에서 첫 번째 이동 목표점으로 이동합니다.

힌트: 시나리오에 충돌 모델이 있고 로봇이 초기 포즈 JPs=[0,0,0,0,0,0] 에서 첫 번째 이동 목표점으로 이동하는 것을 간섭할 때 포즈 유형을 1 로 설정해야 합니다.

로봇 포즈

로봇의 현재 JPs 관절 각도입니다 (파라미터“포즈 유형”이 1 인 경우).

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령어가 정상적으로 실행되면 **2103** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

202 명령어——Mech-Viz 프로젝트를 정지하기

Mech-Viz 프로젝트 실행을 정지합니다. Mech-Viz 프로젝트가 무한 루프에 빠지지 않았거나 정상적으로 정지될 수 있는 경우 이 명령이 필요하지 않습니다.

전송된 명령 파라미터

파라미터	주소
명령어 202	1

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령이 정상적으로 실행되면 **2104** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

203 명령어——Mech-Viz 분기를 선택하기

이 명령은 프로젝트가 따를 분기를 지정하는 데 사용됩니다. 분기 메커니즘은 branch_by_msg 에 의해 설정되며, 이 명령은 이 태스크의 아웃 포트를 지정함으로써 분기를 지정합니다.

이 명령을 실행하기 전에*201* **명령어——*Mech-Viz* 프로젝트를 시작하기** 을 실행하십시오.

Mech-Viz 프로젝트가 branch_by_msg 태스크까지 실행되면 이 명령으로 지정된 아웃 포트를 기다립니다.

전송된 명령 파라미터

파라미터	주소
명령어 203	1
분기 태스크 ID	30
아웃 포트 번호	31

분기 태스크 ID

이 파라미터는 어느 branch_by_msg 태스크에서 분기를 선택할지를 지정하는 데 사용됩니다.

이 파라미터는 양의 정수여야 합니다. 즉 branch_by_msg 의 태스크 ID 입니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

태스크 ID 범위: 1~99.

아웃 포트 번호

이 파라미터는 프로젝트가 branch_by_msg 태스크의 어느 아웃 포트를 따를지 지정하는 데 사용됩니다. Mech-Viz 프로그램은 이 아웃 포트를 따라 계속 실행될 것입니다. 파라미터는 양의 정수 ( ) 입니다.

아웃 포트 번호 범위: 1~99.

힌트:

- 아웃 포트 번호는 Mech-Viz 에서 표시되는 포트 번호 + 1 입니다. 만약 포트 번호가 0 이면 아웃 포트 번호는 1 입니다.
- 아웃 포트 번호는 1 부터 시작하는 포트 인덱스 번호입니다. 예를 들어 지정된 아웃 포트가 왼쪽에서 오른쪽으로 두 번째 포트인 경우 아웃 포트 번호는 2 입니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

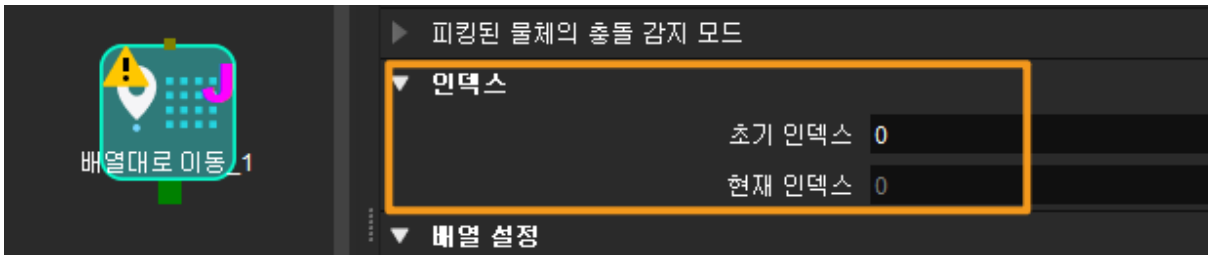
명령어가 정상적으로 실행되면 **2105** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

204 명령어——이동 인덱스를 설정하기

태스크의 인덱스 파라미터 값을 설정하는 명령어입니다. 이러한 유형의 태스크는 일반적으로 연속적이거나 개별적으로 지정된 이동 또는 기타 작업에 사용됩니다.

인덱스 파라미터가 있는 태스크에는" 순서대로 이동", " 배열대로 이동", " 자체 정의한 파렛트 패턴" 및" 미리 설정된 파렛트 패턴" 등이 있습니다.

이 명령을 실행하기 전에*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 를 실행하십시오.



전송된 명령 파라미터

파라미터	주소
명령어 204	1
태스크 ID	32
인덱스 값	33

태스크 ID

이 파라미터는 인덱스 파라미터를 설정해야 하는 태스크를 지정합니다.

이 파라미터의 값은 양의 정수 ( ), 즉 인덱스 대기 중인 태스크의 태스크 ID 여야 합니다. 태스크 ID 는 태스크 파라미터에서 조회 및 설정할 수 있습니다.

인덱스 값

여기 인덱스 파라미터 값은 Mech-Viz 에 표시된 현재 인덱스 값에 1 을 더한 값입니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령어가 정상적으로 실행되면 **2106** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

205 명령어——계획된 경로를 획득하기

이 명령은 Mech-Viz 에서 계획한 경로를 가져오는 데 사용됩니다. 계획된 경로는*201* **명령어**——*Mech-Viz* **프로젝트를 시작하기** 실행 후 가져와야 합니다.

팁: 프로젝트에서 어떤 이동 태스크의 목표점을 로봇으로 보내지 않아야 하는 경우 태스크 파라미터에서" 이동 목표점을 전송하기" 옵션을 선택 취소하십시오.

전송된 명령 파라미터

파라미터	주소
명령어 205	1
목표점 유형	2

목표점 유형

이 파라미터는 Mech-Viz 가 반환할 목표점의 형식을 지정하는 데 사용됩니다.

- 1 목표점은 로봇 관절 각도 (JPs) 의 형식으로 반환됩니다.
- 2 목표점은 TCP 포즈의 형식으로 반환됩니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100
데이터 전송 상태	101
목표점 수량	102
" 비전 이동" 태스크의 위치	103
이번에 전송한 모든 목표점의 포즈	104
이번에 전송한 모든 목표점의 레이블	584
이번에 전송한 모든 목표점의 속도	624

상태 코드

오류가 발생하지 않으면 상태 코드 **2100** 이 반환됩니다. 그렇지 않으면 해당 오류 코드가 반환됩니다.

힌트: 이 명령이 호출될 때 Mech-Viz 결과가 반환되지 않은 경우 (실행 중) Mech-Center 는 로봇에 반환하기 전에 Mech-Viz 결과가 반환될 때까지 기다립니다. 기본 제한 시간은 10 초입니다. 제한 시간을 초과하면 로봇에 타임아웃 오류를 반환합니다.

데이터 전송 상태

이 파라미터는 반환된 데이터가 새 목표점인지를 표시하는 데 사용됩니다.

1 반환된 데이터는 새 목표점이며 읽을 수 있습니다.

목표점 수량

이 파라미터는 이 명령에 의해 반환되는 경로의 이동 목표점 ([포즈, 레이블, 속도]) 의 수를 표시하는 데 사용됩니다.

경로에 20 개 이상의 목표점이 포함된 경우 이 명령을 여러 번 실행하십시오.

범위: 0~20.

" 비전 이동" 태스크의 위치

" 비전 이동" 태스크의 목표점이 프로젝트 전체에서의 위치. " 비전 이동" 태스크는 비전 포인트 (물체를 피킹하는 포인트) 로 이동하는 이동 태스크입니다.

예를 들어 계획된 경로가" 이동 _1", " 이동 _2", " 비전 이동" " 이동 _3" 태스크로 구성된 경우" 비전 이동" 태스크 위치는 3 입니다.

경로에" 비전 이동" 태스크가 없는 경우 이 파라미터의 값은 0 입니다.

포즈

3D 좌표 및 오일러 각 또는 JPs 관절 각도. 유형은 명령어 205 중의 포즈 유형에 의해 결정됩니다.

레이블

포즈에 해당하는 정수 ( ) 레이블입니다. Mech-Vision 프로젝트에서 레이블이 문자열인 경우 출력하기 전에 label_mapping 스텝을 사용하여 레이블을 정수로 매핑합니다.

프로젝트에 레이블이 없는 경우 이 파라미터의 기본값이 0 입니다.

속도

vision_move 태스크 파라미터의 0 이 아닌 속도 파라미터 백분율 값입니다.

206 명령어——DO 신호 리스트를 획득하기

이 명령은 계획된 DO 신호 리스트를 획득하는 데 사용됩니다. DO 신호 리스트는 여러 도구 또는 빨판 파티션을 컨트롤하는 데 사용됩니다.

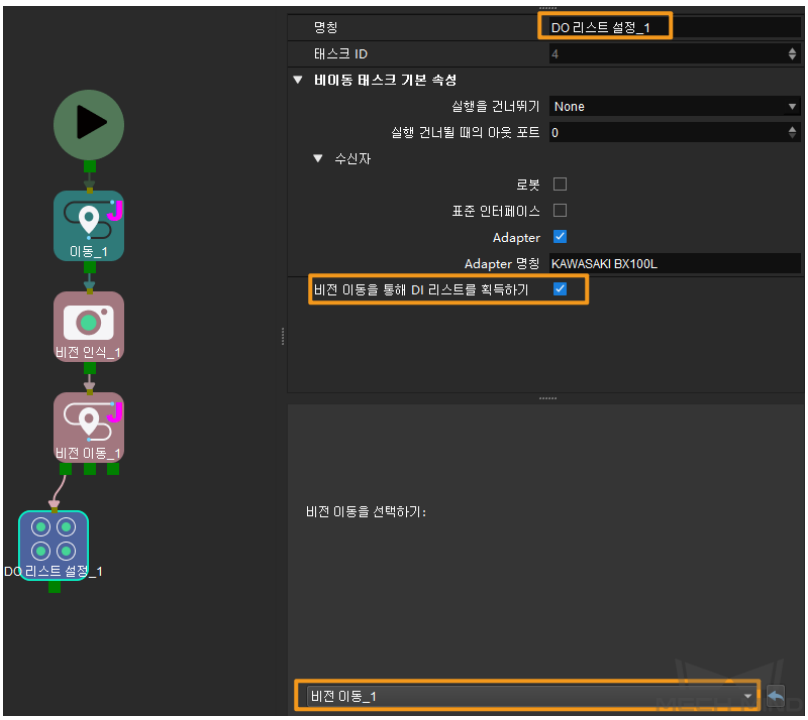
이 명령을 실행하기 전에 Mech-Viz 계획 경로를 얻기 위해*205* **명령** 을 실행해야 합니다.

템플릿 프로젝트에 따라 Mech-Viz 프로젝트를 구축하고 프로젝트에서 해당 빨판 구성 파일을 설정하십시오. 템플릿 프로젝트는 Mech-Center 설치 디렉터리 `tool/viz_project` 아래의 **suction_zone** 프로젝트입니다.

프로젝트의 set_do_list 태스크 파라미터에서

- " 수신자" 에서" 표준 인터페이스" 를 선택하십시오.

- ” 비전 이동에서 DO 리스트를 획득하기” 를 선택하십시오.
- 파라미터 표시줄 하단에 DO 신호 리스트가 필요한 비전 이동 태스크를 선택하십시오.



전송된 명령 파라미터

파라미터	주소
명령어 206	1

반환한 데이터 파라미터

파라미터	주소
상태 코드	100
DO 신호 리스트	704

상태 코드

명령어가 정상적으로 실행되면 **2102** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

DO 포트 값

정수 ( ) 인 64 개의 DO 신호 값이 있습니다.

DO 신호 값 범위 0~999.

플레이스홀더 값 -1.

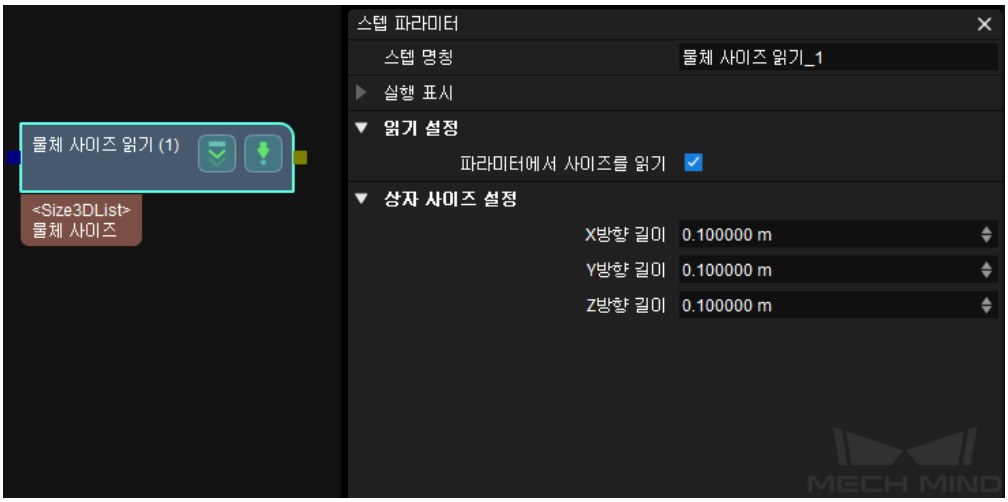
예: DO 신호 값은 각각 1, 3, 5, 6 입니다.

1	3	5	6	-1	-1	-1	-1	...	-1	-1
1 위	2 위	3 위	4 위	5 위	6 위	7 위	8 위	...	63 위	64 위

501 명령어——Mech-Vision 에 물체 치수를 입력하기

이 명령은 물체 크기를 Mech-Vision 프로젝트로 동적으로 입력하는 데 사용됩니다. Mech-Vision 프로젝트를 시작하기 전에 물체 크기를 확인하십시오.

Mech-Vision 프로젝트에는 read_object_dimensions 스텝이 있어야 합니다. 이 스텝 파라미터 **파라미터에서 사이즈를 읽기** 는``True``로 설정해야 합니다.

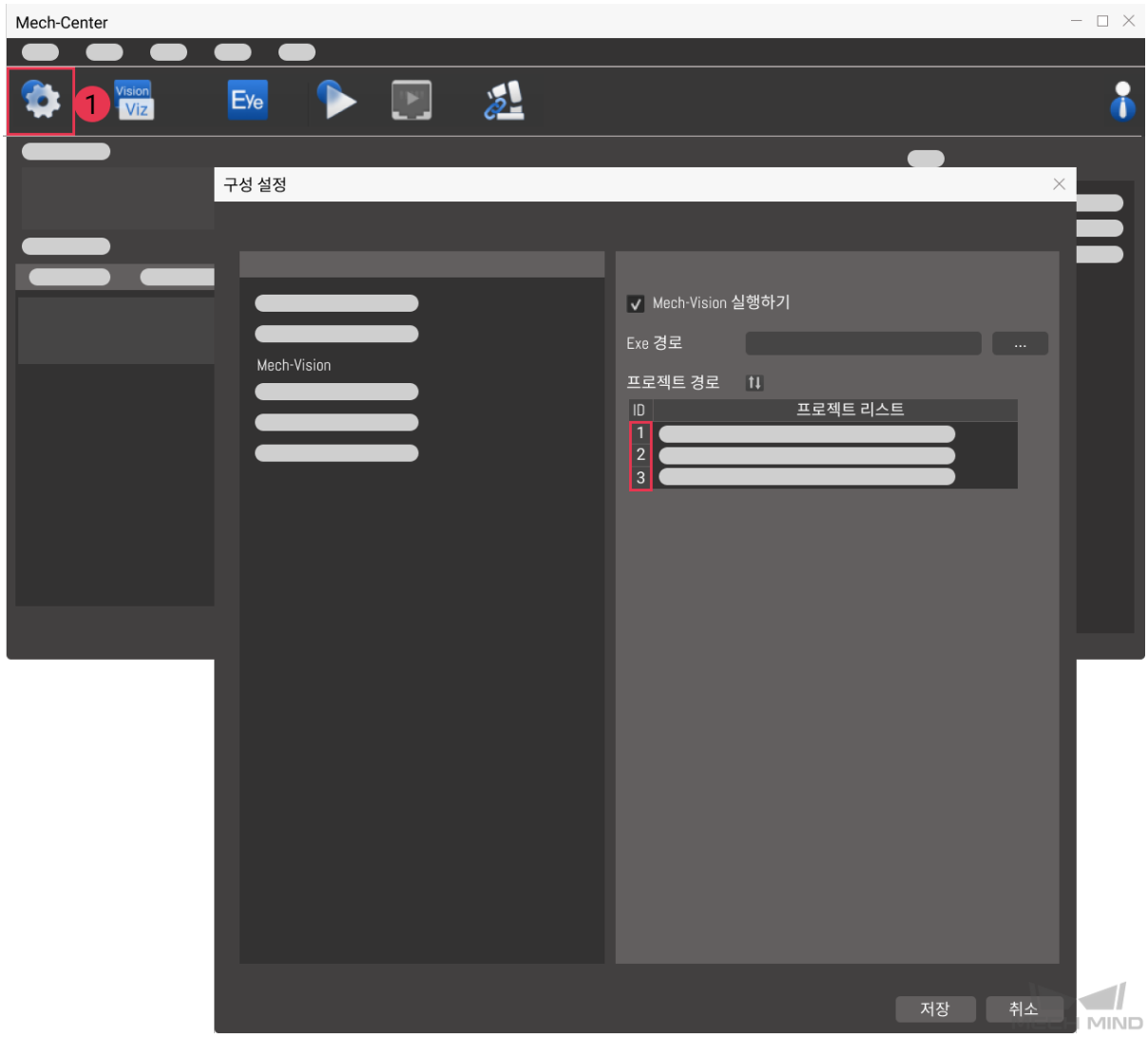


전송된 명령 파라미터

파라미터	주소
명령어 501	1
Mech-Vision 프로젝트 번호	4
[길이, 너비, 높이]	34

Mech-Vision 프로젝트 번호

Mech-Vision 프로젝트가 Mech-Center 에서의 등록 번호, 즉 Mech-Center 에서 **구성 설정** → *Mech-Vision* 의 프로젝트 경로 왼쪽에 표시되는 숫자입니다. 드래그 앤 드롭하여 조정합니다.



길이, 너비, 높이

Mech-Vision 프로젝트에 전달된 물체 사이즈입니다. 사이즈 값은 read_object_dimensions 스텝에서 읽습니다.

딘위: 밀리미터 (mm)

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령이 정상적으로 실행되면 **1108** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

502 명령어——Mech-Viz 에 TCP 를 전송하기

이 명령어는 Mech-Viz 프로젝트에 로봇 TCP 를 동적으로 입력하는 데 사용됩니다. 로봇의 TCP 를 읽는 태스크는 outer_move 입니다.

템플릿 프로젝트를 기반으로 Mech-Viz 프로젝트를 구축하십시오. 템플릿 프로젝트 경로는 Mech-Center 설치 디렉터리 아래의 `tool/viz_project/outer_move` 입니다.

outer_move 태스크를 작업 흐름의 올바른 위치에 배치합니다.

이 명령은*201* **명령어——***Mech-Viz* **프로젝트를 시작하기** 를 실행하기 전에 실행해야 합니다.

전송된 명령 파라미터

파라미터	주소
명령어 502	1
TCP	40

TCP

outer_move 태스크 목표점의 로봇 TCP 데이터를 설정하는 데 사용됩니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령어가 정상적으로 실행되면 **2107** 상태 코드를 반환하고, 그렇지 않으면 해당 오류 코드를 반환합니다.

901 명령어——소프트웨어 상태를 획득하기

이 명령은 소프트웨어의 실행 상태 (Mech-Vision, Mech-Viz, Mech-Center) 를 확인하는 데 사용됩니다.

현재 이 명령은 Mech-Vision 이 프로젝트 실행을 시작할 수 있는지 여부만 확인하는 것을 지원합니다.

전송된 명령 파라미터

파라미터	주소
명령어 901	1

파라미터 설명: 파라미터가 없습니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

시스템 자체 점검 상태. **1101** 상태 코드는“Mech-Vision 프로젝트가 준비됨”이고 다른 상태 코드는“Mech-Vision 프로젝트가 준비되지 않음”입니다. 현재 이 명령은 Mech-Vision 프로젝트가 준비되었는지 확인하는 데만 사용할 수 있습니다.

999 명령어——레지스터 데이터 지우기

이 명령어는 레지스터의 데이터를 지우는 데 사용됩니다.

전송된 명령 파라미터

파라미터	주소
명령어 909	1

파라미터 설명: 파라미터가 없습니다.

반환한 데이터 파라미터

파라미터	주소
상태 코드	100

상태 코드

명령이 정상적으로 실행되면 **3103** 상태 코드가 반환되고, 그렇지 않으면 해당 오류 코드가 반환됩니다.

3.3.7 부록

Mech-Viz 를 사용한 충돌 감지

`XXX/Mech-Center/tool/viz_project/check_collision` 파일의 check_collision.viz 프로젝트와 함께 사용하려면 다음 사항에 주의하십시오.

1. check_collision 프로젝트는 예시 프로젝트일 뿐이며, 프로젝트에 이동 태스크와 관련된 태스크를 제외하고는 다 필요하며 해당 프로젝트의 상대 위치를 삭제하거나 변경할 수 없습니다. 로봇 모델 중 실제 사용하는 모델을 선택하십시오.
2. 이동 태스크와 관련된 태스크는 실제 상황에 따라 추가 또는 삭제될 수 있습니다. 전송된 포즈의 수는 이동 태스크와 관련된 태스크의 수와 일치합니다.
3. Home 위치가 필요한 경우 로봇 측에서 카메라 트리거 명령을 호출하기 전에 포즈 설정 명령을 한 번 호출할 수 있습니다.

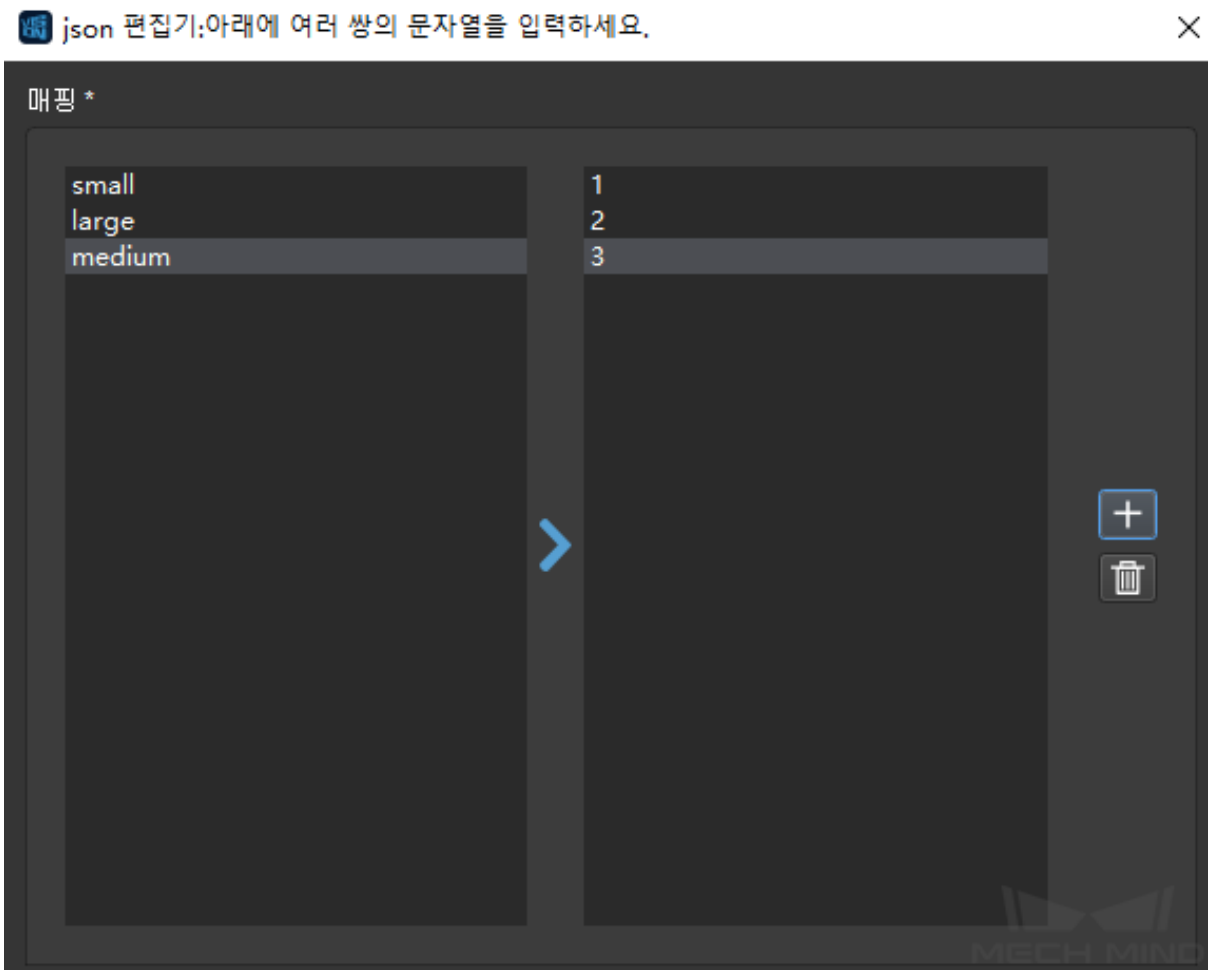
Mech-Viz 빨판 파티션 기능 사용

`XXX/Mech-Center/tool/viz_project/suction_zone` 파일의 Suction_zone.viz 프로젝트와 함께 사용하려면 다음 사항에 주의하십시오.

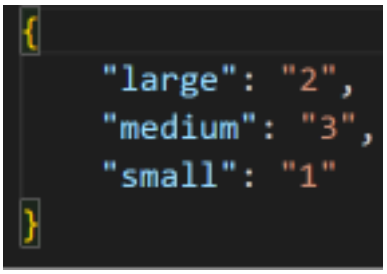
1. suction_zone.viz 프로젝트는 샘플 프로젝트로, 프로젝트에 이동 태스크와 관련된 태스크를 제외하고는 다 필요하며, 해당 프로젝트의 상대 위치를 삭제하거나 변경할 수 없습니다. 로봇 모델 중 실제 사용하는 모델을 선택하십시오.
2. 이동 태스크와 관련된 태스크는 실제 상황에 따라 추가 또는 삭제될 수 있습니다. 전송된 포즈의 수는 이동 태스크와 관련된 태스크의 수와 일치합니다.
3. Home 위치가 필요한 경우 로봇 측에서 카메라 트리거 명령을 호출하기 전에 포즈 설정 명령을 한 번 호출할 수 있습니다.
4. 사용하기 전에 빨판 파일을 구성해야 합니다.
5. 로봇 측에서는 카메라 트리거 명령을 먼저 호출한 다음 DO 신호 리스트 획득하기 명령을 호출해야 합니다.

물체 레이블 식별 기능 보내기

로봇에 전송되는 레이블 필드는 레이블 코드 (정수 표시) 입니다. Mech-Vision 프로젝트에서 레이블 매핑을 설정해야 합니다.

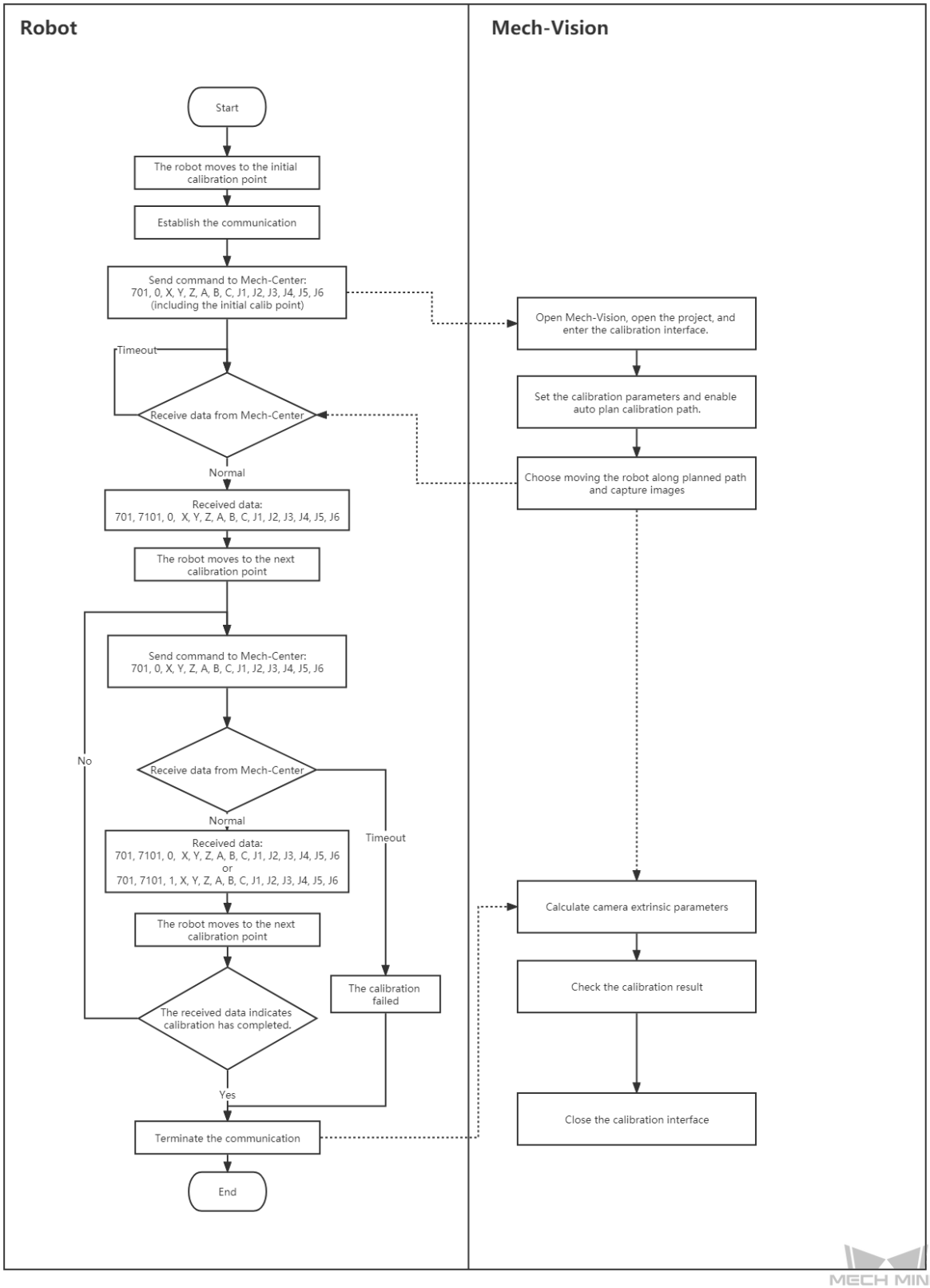


레이블 매핑 파일의 형식은 다음과 같습니다.



- large small medium 은 레이블 문자열입니다.
- 1, 2, 3 은 레이블 코드입니다.

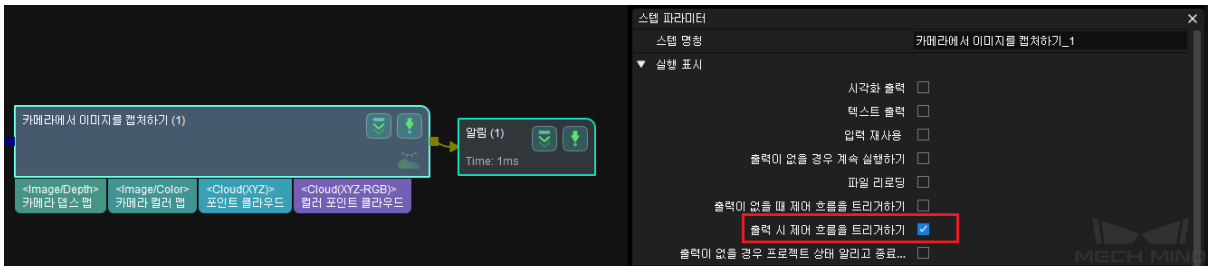
로봇 자동 캘리브레이션 프로그램 흐름



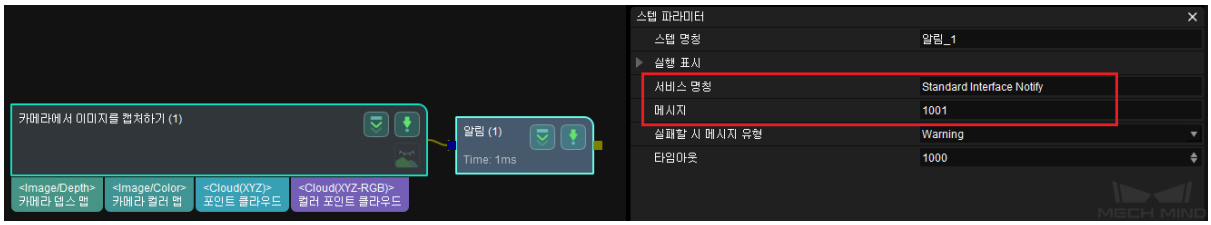
카메라 노출 추가 완료

PROFINET 및 EtherNet/IP 프로토콜의 표준 인터페이스는 시스템 비트를 최적화하는 데 주로 사용되는 **카메라 노출 완료** 신호를 제공합니다. Mech-Vision 프로젝트 계산 시간이 긴 경우 로봇은 카메라 노출이 완료된 직후 위치를 이동해야 합니다. **카메라 노출 완료** 신호 생성에는 Mech-Vision 프로젝트에서 몇 가지 필요한 설정이 필요합니다.

1. Mech-Vision 프로젝트에 notify_vision 을 추가해 capture_images_from_camera 의 제어 흐름에 연결합니다. capture_images_from_camera 의 **실행 플래그** → **출력이 있을 때 트리거 제어 흐름** 을 **True** 로 설정합니다.



2. notify_vision 의 서비스 이름을 **Standard Interface Notify** 로, 메시지 내용을 **1001** 으로 설정합니다 (1001 의 메시지 내용은 변경할 수 없습니다).



3. Mech-Vision 프로젝트를 실행하고 카메라 노출이 완료되면 PLC/로봇 측에서 Exposure_Complete 신호를 수신합니다. Exposure_Complete 신호를 수신한 후 Reset_Exposure 신호를 사용하여 Exposure_Complete 신호를 재설정합니다. 시스템이 10 초 이내에 재설정 신호를 수신하지 않으면 **Mech-Center 데이터 확인 신호 시간 초과** 오류 메시지가 표시됩니다.

3.3.8 표준 인터페이스 상태 코드 및 오류 분석

개요

메크마인드 로보틱스 표준 인터페이스 서비스를 사용할 때 반환된 명령어 정보에 해당 명령어의 실행 상태 (상태 코드) 가 포함됩니다. 상태 코드에는 정상적으로 실행이 완료되는 경우 및 오류가 발생하여 알림이 나타나는 경우가 포함되어 있으며 사용자가 상태 코드에 근거하여 오류를 해결할 수 있습니다.

Mech-Vision

- 1001-1099 Mech-Vision 과 관련된 오류 코드
- 1100-1199 Mech-Vision 과 관련된 정상 상태 코드

Mech-Viz

- 2001-2099 Mech-Viz 와 관련된 오류 코드
- 2100-2199 Mech-Viz 와 관련된 정상 상태 코드

Mech-Center

- 3001-3099 Mech-Center 와 관련된 오류 코드
- 3100-3199 Mech-Center 와 관련된 정상 상태 코드

로봇:

- 4000-4099 로봇과 관련된 오류 코드
- 4100-4199 외부 파라미터 캘리브레이션과 관련된 정상 상태 코드

외부 파라미터 캘리브레이션:

- 7001-7099 외부 파라미터 캘리브레이션과 관련된 오류 코드
- 7100-7199 외부 파라미터 캘리브레이션과 관련된 정상 상태 코드

Mech-Vision

Mech-Vision 의 오류 코드

오류 코드	설명
1001	Mech-Vision 프로젝트가 등록되지 않음
1002	Mech-Vision 비전 결과가 없음
1003	Mech-Vision ROI 내 포인트 클라우드가 없음
1004	Mech-Vision 파라미터 설정 실패
1005	Mech-Vision 무효한 포즈 유형
1006	Mech-Vision 무효한 포즈 데이터
1007	Mech-Vision 계산 중
1008	오류 코드가 사용되지 않았음
1009	Mech-Vision 포즈와 이동 파라미터 수량이 일치하지 않음
1010	Mech-Vision 포즈와 레이블 수량이 일치하지 않음
1011	Mech-Vision 프로젝트 번호가 존재하지 않음
1012	Mech-Vision 파라미터 레시피 번호가 제한을 초과함
1013	Mech-Vision 레시피 명칭이 존재하지 않음
1014	Mech-Vision 파라미터 설정 실패
1015	Mech-Vision 프로젝트 실행 오류
1016	Mech-Vision 딥 러닝 서버 부팅 실패
1017	Mech-Vision 무효한 레이블 매핑
1018	Mech-Vision 포즈 수량 오류
1019	Mech-Vision 실행 시 타임아웃
1020	Mech-Vision 실행되지 않음
1021	Mech-Vision 물체의 사이즈를 설정하지 못했습니다. 스텝 read_object_dimensions 가 프로젝트에 있는지 확인하세요.
1022	Mech-Vision 설정된 물체 사이즈가 무효함
1023	Mech-Vision 카메라 연결 실패
1024	Mech-Vision 포즈와 이동 파라미터 수량이 일치하지 않음
1025	Mech-Vision: 가상 카메라를 사용하고 있으며 데이터를 폐기합니다.

Mech-Vision 의 정상 상태 코드

정상 코드	설명
1100	Mech-Vision 포즈 획득 성공
1101	Mech-Vision 준비 완료
1102	Mech-Vision 프로젝트 트리거 성공
1107	Mech-Vision 레시피 전환 성공
1108	Mech-Vision 상자 사이즈 설정 성공

Mech-Viz

Mech-Viz 의 오류 코드

오류 코드	설명
2001	Mech-Viz 프로젝트가 등록되지 않음
2002	Mech-Viz 프로젝트 실행 중
2003	Mech-Viz Mech-Vision 의 비전 결과를 획득하지 않음
2004	Mech-Viz Mech-Vision 에서 전송된 비전 포즈에 도달하지 못함
2005	Mech-Viz 로봇 관절 각도 계산 실패
2006	오류 코드가 사용되지 않았음
2007	Mech-Viz 이동 계획 실패
2008	Mech-Viz 프로젝트 실행 오류
2009	Mech-Viz TCP 포즈를 제공하지 않음
2010	Mech-Viz 경로에 도달하지 못함
2011	Mech-Viz DO 리스트를 제공하지 않음
2012	Mech-Viz 무효한 포즈 유형
2013	Mech-Viz 무효한 포즈 데이터
2014	Mech-Viz 프로젝트가 설정되지 않음
2015	Mech-Viz TCP 유형을 지원하지 않음
2016	Mech-Viz 파라미터 설정 실패
2017	Mech-Viz 실행 정지 실패
2018	Mech-Viz 무효한 분기 아웃 포트 번호
2019	Mech-Viz 분기 설정 실패. 분기 명칭이 존재한지 확인하기
2020	Mech-Viz 싱귤래리티 운동 실패
2021	Mech-Viz 이동 계획 실패
2022	Mech-Viz 실행되지 않음
2023	Mech-Viz 프로젝트 파일 오류
2024	Mech-Viz 무효한 분기 명칭
2025	Mech-Viz 실행 시 타임아웃
2026	Mech-Viz 무효한 인덱스 유형의 태스크 명칭
2027	Mech-Viz 무효한 인덱스 수치
2028	Mech-Viz 인덱스 설정 실패. 인덱스 명칭이 존재한지 확인하기
2029	Mech-Viz 외부 이동 설정 실패
2030	Mech-Viz 무효한 포즈
2031	Mech-Viz 로봇 관절 사이의 충돌
2032	Mech-Viz 시나리오 물체와의 충돌
2033	Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 포인트 수가 역치를 초과함
2034	Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 면적이 역치를 초과함
2035	Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 부피가 역치를 초과함

다음 페이지에 계속

표 1 – 이전 페이지에서 계속

2036	Mech-Viz	[비전 인식] 에 이미지를 캡처하지 않음
2037	Mech-Viz	[비전 인식] 에 결과가 없음
2038	Mech-Viz	[비전 인식] 에 ROI 내 포인트 클라우드가 없음
2039	Mech-Viz	계획에 사용될 수 있는 포즈가 없음
2040	Mech-Viz	[비전 결과를 재사용하기] 에 계획이 실패한 경로가 있음
2041	Mech-Viz	파라미터 설정 실패
2042	Mech-Viz	이동 계획 실패
2043	Mech-Viz	비전 목표점의 자체 정의한 데이터를 획득 실패
2044	Mech-Viz	비전 서비스가 등록되지 않음

Mech-Viz 의 정상 상태 코드

정상 코드	설명	
2100	Mech-Viz	실행 성공
2101	Mech-Viz	실행 정지 성공
2102	Mech-Viz	DO 리스트를 성공적으로 보냄
2103	Mech-Viz	부팅 성공
2104	Mech-Viz	정지 성공
2105	Mech-Viz	분기 설정 성공
2106	Mech-Viz	인덱스 설정 성공
2107	Mech-Viz	외부에서 전송된 포즈에 대한 설정 성공

Mech-Center

Mech-Center 의 오류 코드

오류 코드	설명	
3001	Mech-Center	무효한 명령어
3002	Mech-Center	인터페이스 명령어 길이 또는 포맷 오류
3003	Mech-Center	클라이언트 연결 끊김
3004	Mech-Center	서버 연결 끊김
3005	Mech-Center	Mech-Vision 을 호출할 때 타임아웃
3006	Mech-Center	알수 없는 오류
3007	Mech-Center	데이터 확인 신호 타임아웃
3008	Mech-Center	구성 ID 가 없어서 파라미터를 읽어내거나 설정할 수 없습니다.

Mech-Center 의 정상 상태 코드

정상 코드	설명	
3100	Mech-Center	클라이언트 연결 정상
3101	Mech-Center	서버 접속 정상
3102	Mech-Center	클라이언트 연결 대기 중
3103	Mech-Center	데이터 캐시 삭제 성공

로봇

로봇 오류 코드

오류 코드	설명
4001	무효한 로봇 유형
4002	지원되지 않은 오일러 각 유형
4003	로봇 서비스가 등록되지 않음
4004	로봇 파라미터 누락됨
4005	로봇을 연결할 수 없습니다. 로봇 IP 및 네트워크 구성을 확인하십시오.
4006	로봇 프로그램 버전이 일치하지 않습니다. 다시 다운로드하십시오.

로봇 정상 상태 코드

정상 코드	설명
4100	로봇 서비스가 등록됨
4101	로봇 연결 성공
4102	로봇 연결 끊김
4103	사용자가 로봇 서비스를 껐음

외부 파라미터 캘리브레이션

외부 파라미터 캘리브레이션의 오류 코드

오류 코드	설명
7001	캘리브레이션: 파라미터 오류
7002	캘리브레이션: Mech-Vision 이 캘리브레이션된 포즈를 출력하지 않음
7003	캘리브레이션: 로봇이 캘리브레이션 포인트에 도달하지 못함

외부 파라미터 캘리브레이션의 정상 상태 코드

정상 코드	설명
7100	캘리브레이션: 로봇이 캘리브레이션 포인트에 성공적으로 도달함
7101	캘리브레이션: Mech-Vision 이 캘리브레이션된 포즈를 출력함

Mech-Vision 오류 분석 및 문제 해결

1001

Mech-Vision 프로젝트가 등록되지 않음

오류 원인:

- 프로젝트가 Mech-Vision 에서 열리지 않았습니다.
- 해당 프로젝트가 **자동 로드하기** 옵션을 선택하지 않았습니다.

해결 방법:

- Mech-Vision 에서 해당 프로젝트를 열어야 합니다.
- 해당 프로젝트가 **자동 로드하기** 옵션을 선택해야 합니다.

1002

Mech-Vision 비전 결과가 없음

오류 원인:

- 호출한 Mech-Vision 프로젝트가 성공적으로 실행되었지만 출력 결과가 없습니다. 이런 문제가 발생할 수 있는 원인은 다음과 같습니다. 인스턴스 세그먼테이션의 믿음도 역치가 과하게 높음, 현재 시나리오에 매칭할 수 있는 물체가 없음, ROI 설정이 잘못됐음, 포인트 클라우드의 효과가 좋지 않음, 필터링 설정이 합리적이지 않음 등입니다.
- 102 명령어를 호출함으로써 비전 목표점에 있는 모든 포즈를 획득했고 캐시도 비어 있지만 102 명령어를 계속 호출합니다.

해결 방법:

- 스텝 procedure_out 의 PoseList 포트부터 위로 Mech-Vision 프로젝트의 데이터 스티림을 체크합니다.
- 클라이언트 인터페이스 프로그램을 체크합니다. 102 명령어가 반환된 데이터의 파라미터 [포즈 전송 상태] 가 1 이면 모든 포즈가 전송되었다는 뜻이고 이 때 102 명령어를 계속 호출하면 해당 오류를 초래할 수 있습니다.
  - 102 명령어가 반환한 데이터 포맷: *102,* **상태 코드**, **포즈 전송 상태**, **포즈의 수**, **예약된 필드**, *[***포즈**, **레이블**, **속도***]*.

1003

Mech-Vision ROI 내 포인트 클라우드가 없음

오류 원인:

- 호출한 Mech-Vision 프로젝트가 성공적으로 실행되었지만 3D ROI 내에 포인트 클라우드가 없습니다.

해결 방법:

- 포인트 클라우드에서 ROI 를 설정하는 것과 관련된 스텝의 ROI 설정을 체크합니다. 일부 프로젝트에서 이 오류를 통해 상자가 제자리에 있는지 또는 상자가 비어 있는지를 확인하는 데 도움이 될 수도 있기 때문에 이 오류는 반드시 프로젝트 오류가 아닙니다.

1004

Mech-Vision 파라미터 설정 실패

1005

Mech-Vision 무효한 포즈 유형

오류 원인:

- 101 명령어에서 포즈 유형을 설정하는 파라미터 중에 무효한 값이 존재합니다.
  - 101 명령어의 포맷: *101,* **프로젝트 번호**, **포즈의 목표 수량**, **로봇 포즈 유형**, **로봇 포즈**.

    파라미터 [로봇 포즈 유형] 의 수치:

    * `0` 이미지 캡처 포즈가 필요하지 않습니다. 예를 들어 Eye to Hand 모드에서 이 수치가 0 이 될 수 있습니다.
    * `1` 이미지 캡처 포즈의 포맷은 JPs 관절 각도입니다.
    * `2` 이미지 캡처 포즈의 포맷은 플랜지 포즈입니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크합니다. 파라미터 [포즈 유형] 의 수치는 [0, 2] 범위 내에 있어야 합니다.

1006

Mech-Vision 무효한 포즈 데이터

오류 원인:

- 로봇이 101 명령어를 전송하고 설정한 포즈 데이터가 여섯자리 미만입니다. 로봇 포즈는 기본적으로 6 축 로봇의 포즈인데 4 축 또는 5 축 로봇인 경우 나머지 필드에 0 을 입력하세요.
  - 101 명령어의 포맷: *101,* **프로젝트 번호**, **포즈의 목표 수량**, **로봇 포즈 유형**, **로봇 포즈**.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크합니다. 101 명령어가 전송한 포즈 데이터는 여섯 자리로 구성되어야 합니다.

1007

Mech-Vision 계산 중

오류 원인:

- Mech-Vision 프로젝트가 실행될 때
- Mech-Vision 에서 여러 프로젝트를 동시에 실행할 수 있지만 하나의 Mech-Vision 프로젝트가 실행될 때 반복적으로 시작될 수 없습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크합니다. 프로그램에서 설정한 Mech-Vision 프로젝트 번호가 맞는지 확인합니다.
- 프로그램에서 같은 Mech-Vision 프로젝트가 짧은 시간 내에 반복적으로 호출된 경우가 있는지 체크합니다.

1008

오류 코드가 사용되지 않았음

1009

Mech-Vision 포즈와 이동 파라미터 수량이 일치하지 않음

오류 원인:

- 일반적으로 이런 오류는 비전 결과를 통해 로봇의 이동 경로를 계획하는 프로젝트 (예: 접착제 도포) 에 사용됩니다. Mech-Vision 프로젝트에서 출력한 이동 파라미터의 수가 경로의 이동 목표점의 수와 일치하지 않습니다.

해결 방법:

아직 없음

1010

Mech-Vision 포즈와 레이블 수량이 일치하지 않음

오류 원인:

- Mech-Vision 프로젝트에서 출력한 레이블의 수가 포즈와 일치하지 않습니다.

해결 방법:

- Mech-Vision 프로젝트의 데이터 스트림 중의 포즈와 레이블을 체크하여 포즈와 레이블의 수가 서로 다른 원인을 찾으세요.

1011

Mech-Vision 프로젝트 번호가 존재하지 않음

오류 원인:

- Mech-Center 의 구성 설정에 101 명령어 파라미터에서 설정한 프로젝트 번호가 없습니다.
  - 예를 들어 구성 설정에서 하나의 프로젝트 경로만 있는 경우 101 명령어가 2 번 프로젝트를 호출하면 이 오류를 초래할 것입니다.

해결 방법:

- 해당 Mech-Vision 프로젝트에서 **현재 프로젝트를 자동으로 로드하기** 옵션을 선택한지 체크합니다.
- 클라이언트 인터페이스 프로그램에서 트리거한 Mech-Vision 프로젝트 번호가 맞는지 확인합니다.
  - 101 명령어의 포맷: *101*, **프로젝트 번호**, **포즈의 목표 수량**, **로봇 포즈 유형**, **로봇 포즈**.

1012

Mech-Vision 파라미터 레시피 번호가 제한을 초과함

오류 원인:

- 클라이언트 인터페이스 프로그램은 103 명령어를 호출하여 Mech-Vision 프로젝트 레시피를 설정했지만 호출한 레시피 번호와 대응하는 레시피가 없습니다.
  - 예를 들어 Mech-Vision 프로젝트에 설정한 레시피가 두개만 있는데 클라이언트 인터패이스 프로그램은 3 번 레시피를 호출했을 때 이런 오류 코드가 나타날 것입니다.

해결 방법:

- Mech-Vision 프로젝트의 레시피가 올바르게 설정된지를 체크하십시오.
- 클라이언트 인터페이스 프로그램의 103 명령어에 설정한 레시피 번호가 맞는지 체크하십시오.

1013

Mech-Vision 레시피 명칭이 존재하지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램은 103 명령어를 호출하여 Mech-Vision 프로젝트 레시피를 설정했지만 Mech-Vision 프로젝트에서 아무 레시피도 설정하지 않았을 때 이런 오류 코드가 나타날 것입니다.

해결 방법:

- 프로젝트가 필요한 레시피 및 대응하는 번호가 모두 올바르게 설정되도록 Mech-Vision 프로젝트의 레시피 설정을 체크하십시오.
- 해당 프로젝트에서 레시피를 바꿀 수요가 없으면 클라이언트 인터페이스 프로그램은 레시피를 전환하는 기능을 사용하지 않습니다.

1014

Mech-Vision 파라미터 설정 실패

오류 원인:

- Mech-Center 가 Mech-Vision 과 통신하는 데 이상이 발생하여 Mech-Vision 레시피를 성공적으로 설정하지 못했습니다.

해결 방법:

- Mech-Center 및 Mech-Vision 소프트웨어를 다시 시작하십시오.

1015

Mech-Vision 프로젝트 실행 오류

오류 원인:

- Mech-Vision 프로젝트가 실행되는 동안 오류가 발생하여 Mech-Vision 의 오류 코드 [CV-Exxxx] 또는 다른 Mech-Center 와 관련된 분석되지 않은 오류 알림이 나타납니다. 이런 경우 Mech-Vision 프로젝트가 끝까지 실행되지 못해 비정상적으로 중지됩니다.

해결 방법:

- Mech-Vision 의 오류 메시지를 확인하고 오류 메시지 내용에 근거하여 Mech-Vision 프로젝트의 문제를 분석하고 해결하십시오.

1016

Mech-Vision 딥 러닝 서버 부팅 실패

오류 원인:

- 트리거된 Mech-Vision 프로젝트에 딥 러닝 모듈이 포함되지만 딥 러닝 서버가 시작되지 않았습니다. 대응한 Mech-Vision 의 오류 코드: DL-E0201 딥 러닝 서버가 시작되지 않음.

해결 방법:

- Mech-Vision 프로젝트가 처음으로 실행된 것인지, 또는 모델을 로드하는 데 걸린 시간이 매우 긴 것인지를 체크하십시오. 모델을 미리 로드하려면 딥 러닝과 관련된 스텝의 파라미터에서 **프로젝트 오픈 시 모델 프리로드** 를 선택하십시오.
- 딥 러닝 환경이 설치되어 있는지를 확인하세요.

1017

Mech-Vision 무효한 레이블 매핑

오류 원인:

- Mech-Vision 프로젝트에서 출력한 레이블의 데이터 유형이 INT 가 아닙니다. 표준 인터페이스를 사용할 때 Mech-Vision 프로젝트에서 출력한 레이블은 INT 포맷으로 매핑되어야 합니다. 그렇지 않으면 이 오류 코드가 나타날 것입니다.

해결 방법:

- Mech-Vision 의 데이터 스트림을 확인하십시오. 스텝 procedure_out 에 입력한 레이블이 양의 정수가 아닌 경우 스텝 label_mapping 을 통해 레이블을 양의 정수로 매핑하십시오.

1018

Mech-Vision 포즈 수량 오류

오류 원인:

- 클라이언트 인터페이스 프로그램이 101 명령어를 호출하여 Mech-Vision 프로젝트를 호출했지만 설정한 비전 포인트의 목표 수량이 인터페이스의 데이터 길이 제한을 넘었습니다.
  - 101 명령어의 포맷: *101,* **프로젝트 번호**, **포즈의 목표 수량**, **로봇 포즈 유형**, **로봇 포즈**.

해결 방법:

- 데이터 길이는 Mech-Center 의 [구성 설정] 에서 설정할 수 있습니다. *Mech-Interface* → **고급 설정** 아래에서 단번에 포즈를 보낼 수 있는 최대 수량을 설정할 수 있습니다. 범위: [1, 30].
- 파라미터 [포즈의 목표 수량] 의 수치가 Mech-Center 의 [구성 설정] 에서 설정한 수치보다 작아야 합니다.

1019

Mech-Vision 실행 시 타임아웃

오류 원인:

- Mech-Vision 비전 결과를 획득하기 위해 102 명령어를 호출했을 때부터 타이머를 시작하고 규정한 시간 내에 Mech-Vision 프로젝트의 실행 과정이 완료되지 않았다면 이 오류 코드가 나타날 것입니다.
- 제한 시간은 Mech-Center [구성 설정] 의 *Mech-Interface* → **고급 설정** 아래에서 설정할 수 있으며 기본값은 10 초입니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 확인하고 Mech-Vision 비전 결과를 획득하기 위해 102 명령어를 호출하기 전에 지연 기간을 적절히 설정할 수 있습니다.
- 실행 시간이 긴 Mech-Vision 프로젝트에 대해 제한 시간을 절적하게 수정할 수 있습니다.

1020

Mech-Vision 실행되지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램은 101 명령어를 호출하여 Mech-Vision 프로젝트를 트리거하지 않고 직접 102 명령어를 통해 해당 프로젝트의 비전 결과를 획득하려고 했을 때 이 오류 코드가 나타납니다.
  - 예를 들어 Mech-Center 에서 두 가지 Mech-Vision 프로젝트를 동시에 등록하고 클라이언트 인터페이스 프로그램을 프로젝트 1 로 시작했지만 프로젝트 2 의 비전 결과를 획득하려 하면 이 오류가 발생할 것입니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 102 명령어가 지정한 프로젝트 번호가 맞는지를 확보하세요.

1021

Mech-Vision 물체의 사이즈를 설정하지 못했습니다. 스텝 read_object_dimensions 가 프로젝트에 있는지 확인하세요.

오류 원인:

- 클라이언트 인터페이스 프로그램이 501 명령어를 호출하여 Mech-Vision 프로젝트를 위해 동적으로 물체 사이즈를 설정했지만 Mech-Vision 프로젝트에 스텝 read_object_dimensions 가 없는 경우 이 오류가 발생합니다.

해결 방법:

- 프로젝트에 스텝 read_object_dimensions 가 있도록 Mech-Vision 프로젝트를 체크하십시오.

1022

Mech-Vision 설정된 물체 사이즈가 무효함

오류 원인:

- 클라이언트 인터페이스 프로그램이 501 명령어를 호출하여 Mech-Vision 프로젝트를 위해 동적으로 물체 사이즈를 설정했지만 설정한 물체의 사이즈가 무효 (0/음수가 있음) 합니다.

해결 방법:

- 설정한 물체의 사이즈 (길이/너비/높이) 가 양의 실수가 되도록 클라이언트 인터페이스 프로그램을 체크하십시오.

1023

Mech-Vision 카메라 연결 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 Mech-Vision 프로젝트를 트리거하여 이미지를 캡처하려고 했지만 카메라 연결이 끊겼습니다. 대응하는 Mech-Vision 의 오류 코드: CV-E0201

해결 방법:

- 네트워크 연결을 체크하십시오.
- 카메라와 IPC 의 IP 주소가 동일한 네트워크 세그먼트에 있는지 확인하세요.

  *(**위에서 언급한 두 항목은** CMD **에서“**ping**”+ 카메라** IP **주소를 입력하여 확인할 수 있습니다**.)*
- 카메라 전원, IPC 의 방화벽 설정 등 사항을 체크하십시오.
- 문제가 여전히 해결되지 못하면 저희 서포트팀에게 문의하십시오.

1024

Mech-Vision 포즈와 이동 파라미터 수량이 일치하지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램은 110 명령어를 호출한 후 반환된 데이터 중에 포즈의 수가 자체 정의한 데이터 요소의 수와 다릅니다. 자체 정의한 데이터는 Mech-Vision 프로젝트에 있는 스텝 procedure_out 의 포즈와 레이블을 제외한 다른 포트에 입력한 데이터를 가리킵니다. 가능한 상황은 다음과 같습니다.
  - 자체 정의한 포트의 데이터가 비어 있음
  - 자체 정의한 포트의 데이터의 길이가 포즈의 길이와 일치하지 않음
- 표준 인터페이스를 사용하는 경우 스텝 procedure_out 을 실행했을 때 포즈 수량이 N 이면 모든 자체 정의한 데이터 포트 (포즈와 레이블을 제외함) 에 1 혹은 N 개 데이터를 입력해야 합니다. 그렇지 않으면 이 오류 코드가 나타날 것입니다.
  - 예를 들어 자체 정의한 포트의 데이터가 인스턴스 세그먼테이션을 통해 출력한 물체 수량이면 1 개의 데이터 항목만 입력하고 이 명령어를 호출할 때마다 같은 물체 수량을 획득합니다. 자체 정의한 포트의 데이터 유형이 상자의 사이즈인 경우 N 개 데이터를 입력해야 하고 이 명령어를 호출할 때마다 포즈와 대응하는 [상자 사이즈] 데이터를 획득합니다.

해결 방법:

- Mech-Vision 프로젝트를 체크하고 출력 포트의 데이터가 위 요구에 부합해야 합니다.

1025

Mech-Vision: 가상 카메라를 사용하고 있으며 데이터를 폐기합니다.

Mech-Viz 오류 분석 및 문제 해결

2001

Mech-Viz 프로젝트가 등록되지 않음

오류 원인:

- 이 프로젝트가 Mech-Viz 에서 열리지 않았습니다.
- 201 명령어를 통해 시작된 Mech-Viz 프로젝트가 **자동 로드하기** 옵션을 선택하지 않았습니다.

해결 방법:

- 프로젝트가 Mech-Viz 에서 열려 있는지 확인하십시오.
- 해당 프로젝트가 **자동 로드하기** 옵션을 선택해야 합니다.

2002

Mech-Viz 프로젝트 실행 중

오류 원인:

- Mech-Viz 프로젝트가 실행되는 동안 클라이언트 인터페이스 프로그램이 201 명령어를 다시 한번 호출하여 같은 Mech-Viz 프로젝트를 트리거하려고 하기 때문에 이 오류가 발생했습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램이 짧은 시간 내에 201 명령어를 반복적으로 호출하여 Mech-Viz 프로젝트 실행을 트리거하는 경우가 있는지 체크하십시오.

2003

Mech-Viz Mech-Vision 의 비전 결과를 획득하지 않음

오류 원인:

- Mech-Viz 프로젝트에서 태스크 check_look 가“결과 없음”아웃포트 (즉 왼쪽부터 두번째 포트) 에서 실행되었으며 프로젝트가 중지되었습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 태스크 check_look 의“결과 없음”아웃포트가 다른 분기를 통해 실행되도록 연결선을 추가할 필요가 있는지 확인하세요.
- ROI 에 있는 대상물이 모두 피킹 되었는지 체크하십시오.
- Mech-Vision 프로젝트의 ROI 및 인스턴스 세그먼테이션의 역치와 관련된 설정이 맞는지 체크하십시오.

2004

Mech-Viz Mech-Vision 에서 전송된 비전 포즈에 도달하지 못함

오류 원인:

- Mech-Viz 가 비전 서비스를 호출하여 계산한 포즈가 로봇이 도달할 수 있는 작업 범위를 초과했습니다.

해결 방법:

- Mech-Viz 에서 작업물 포인트 클라우드의 위치가 정상인지를 체크하십시오.
- 카메라 외부 파라미터가 정상인지 체크하십시오.
- Mech-Viz 에서 로봇 클램프 TCP 의 설정이 정상인지, 또한 작업물이 로봇 작업 범위 이외에 위치한지를 체크하십시오.

2005

Mech-Viz 로봇 관절 각도 계산 실패

오류 원인:

- Mech-Viz 가 목표점을 위해 로봇 JPs 관절 각도를 계산할 수 없습니다.

해결 방법:

아직 없음

2006

오류 코드가 사용되지 않았음

2007

Mech-Viz 이동 계획 실패

오류 원인:

- Mech-Viz 는 비전 결과 중의 모든 비전 포인트에 대한 경로 계획에 실패했으며 모든 비전 포인트 경로 계획에 실패한 원인은 모두 같지 않습니다.

  *(**같은 원인으로 비전 포인트 경로 계획에 실패했다면 오류 코드도 같습니다**.)*

  - 예를 들어 비전 결과에 40 개의 비전 포인트가 있고, 그 중 20 개는 로봇이 비전 포인트에 도달하지 못해 계획에 실패하고, 20 개는 충돌이 감지되어 계획에 실패한 경우 이 오류가 보고됩니다.

해결 방법:

- Mech-Viz 경로 계획과 관련된 로그 내용을 확인하여 계획 실패의 원인을 찾습니다.

2008

Mech-Viz 프로젝트 실행 오류

오류 원인:

- Mech-Viz 프로젝트가 실행되는 동안 오류가 발생하여 Mech-Viz 의 오류 코드 [MP-Exxxx] 또는 다른 Mech-Center 와 관련된 분석되지 않은 오류 알림이 나타납니다. 이런 경우 Mech-Viz 프로젝트가 끝까지 실행되지 못해 비정상적으로 중지됩니다.

해결 방법:

- Mech-Viz 오류 알림과 로그 내용을 체크하십시오.

2009

Mech-Viz TCP 포즈를 제공하지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램이 205 명령어를 호출하여 Mech-Viz 가 계획한 경로를 획득하고 요구한 반환 데이터의 유형이 로봇 TCP 포즈이지만 실제 Mech-Viz 가 출력한 데이터에 TCP 포즈가 없습니다.

해결 방법:

- Mech-Viz 소프트웨어의 [기타] 패널 설정에서“TCP 포즈를 전송하기”옵션을 선택했는지 확인하세요.

2010

Mech-Viz 경로에 도달하지 못함

오류 원인:

- Mech-Viz 프로젝트가 실행될 때 오류가 발생함. 오류 코드: MP-E0007.

해결 방법:

아직 없음

2011

Mech-Viz DO 리스트를 제공하지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램이 206 명령어를 호출하여 공구 (파티션 빨판, 어레이 그리퍼 등)의 DO 신호 리스트를 획득하려고 했지만 Mech-Viz 프로젝트에서 DO 신호 리스트를 구성하지 않았습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 태스크 [비전 이동] 뒤에 태스크 set_do_list 가 있는지를 확인하고 태스크의 파라미터“수신자”를“표준 인터페이스”로 선택하십시오.

2012

Mech-Viz 무효한 포즈 유형

오류 원인:

- 클라이언트 인터페이스 프로그램이 201 명령어를 호출하여 Mech-Viz 프로젝트 실행을 트리거할 때 설정한 로봇 포즈 유형의 값이 무효입니다.
  - 201 명령어 형식: *201* , **포즈 유형** , **로봇 포즈**.
    - 파라미터“포즈 유형”의 값:

· 0 이미지 캡처 포즈가 필요하지 않습니다. 예를 들어 Eye to Hand 모드에서 로봇의 현재 포즈부터 Mech-Viz 첫 번째 목표점까지의 경로를 체크할 필요가 없을 때 0 으로 설정할 수 있습니다.

· 1 현재 로봇 포즈의 포맷은 JPs 관절 각도입니다.

- 클라이언트 인터페이스 프로그램이 205 명령어를 호출하여 Mech-Viz 가 계획한 경로를 획득했을 때 설정한 포즈의 반환값 유형이 무효입니다.
  - 201 명령어 형식: *201* , **반환값 유형**.
    * 파라미터"반환값 유형"의 수치:
      · 1 반환한 포즈 유형은 로봇 JPs 관절 각도입니다.
      · 2 반환한 포즈 유형은 로봇 TCP 입니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하여 Mech-Viz 프로젝트를 트리거할 때 설정한 포즈 유형이 유효한지 확인하십시오.
- 클라이언트 인터페이스 프로그램을 체크하고 205 명령어를 호출하여 Mech-Viz 프로젝트가 경로를 계획할 때 설정한 포즈 반환값 유형이 유효한지 확인하십시오.

2013

Mech-Viz 무효한 포즈 데이터

오류 원인:

- 클라이언트 인터페이스 프로그램이 201 명령어를 호출하여 Mech-Viz 프로젝트를 트리거했을 때 설정한 로봇 포즈 데이터의 길이가 여섯자리 미만입니다. Mech-Viz 프로젝트는 6 축 로봇의 포즈 데이터만 지원하며 4 축 또는 5 축 로봇인 경우 6 축 데이터에 0 을 입력하세요.
  - 201 명령어 형식: *201* , **포즈 유형** , **로봇 포즈**.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 201 명령어를 통해 Mech-Viz 프로젝트를 트리거할 때 로봇의 포즈 데이터가 여섯자리인 JPs 관절 각도인지 체크하십시오. Mech-Viz 는 현재 로봇 포즈가 필요하지 않으면 포즈 유형을 0(로봇 포즈를 입력할 필요가 없음) 으로 설정할 수 있습니다.

2014

Mech-Viz 프로젝트가 설정되지 않음

오류 원인:

- 해당 프로젝트가 Mech-Viz 에서 열리지 않았습니다.
- Mech-Viz 프로젝트에서 **자동 로드하기** 옵션을 선택하지 않았습니다.

해결 방법:

- Mech-Viz 소프트웨어를 열어 올바른 프로젝트를 시작하고 **자동 로드하기** 를 선택하십시오.

2015

Mech-Viz TCP 유형을 지원하지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램이 201 명령어를 호출하여 Mech-Viz 프로젝트를 트리거했을 때 설정한 포즈 유형은 TCP 인데 Mech-Viz 가 지원하지 않습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 201 명령어를 통해 Mech-Viz 프로젝트를 트리거할 때 설정한 포즈 유형이 0(필요하지 않음) 또는 1(JPs 관절 각도) 이 될 수 있는지 확인하세요.
  - 201 명령어 형식: *201* , **반환값 유형**.
    - 파라미터"반환값 유형"의 수치:
      - 1 반환한 포즈 유형은 로봇 JPs 관절 각도입니다.
      - 2 반환한 포즈 유형은 로봇 TCP 입니다.

2016

Mech-Viz 파라미터 설정 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 208 명령어를 호출하여 Mech-Viz 태스크 파라미터를 설정했을 때 오류가 발생했습니다.

해결 방법:

- 구성 파일에 있는 태스크 번호와 파라미터 툴팁이 맞는지 확인하세요.
- Mech-Center 의 **구성 설정** → *Mech-Interface* → **고급 설정** 에서 구성 파일 (*Property Config*) 을 열 수 있습니다.

2017

Mech-Viz 실행 정지 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 202 명령어를 호출하여 Mech-Viz 실행을 중지했지만 5 초 내에 Mech-Viz 가 정상적으로 중지되지 못한다면 이 오류 코드가 나타날 것입니다.

해결 방법:

아직 없음

2018

Mech-Viz 무효한 분기 아웃 포트 번호

오류 원인:

- 클라이언트 인터페이스 프로그램이 203 명령어를 호출하여 Mech-Viz 분기 아웃포트를 설정했을 때 아웃포트 번호가 0 인 경우, 또는 아웃포트 번호가 분기 태스크의 아웃포트 수보다 큰 경우 이 오류 코드가 나타납니다.
  - 203 명령어 형식: *203,* **분기 태스크 인덱스**, **아웃포트 번호**.

해결 방법:

- Mech-Viz 프로젝트에 있는 각 분기 태스크의 각 아웃포트를 체크하십시오.
- 203 명령어에서 설정한 아웃포트 번호를 체크하십시오.

2019

Mech-Viz 분기 설정 실패. 분기 명칭이 존재한지 확인하기

오류 원인:

- 클라이언트 인터페이스 프로그램이 203 명령어를 호출하여 Mech-Viz 분기를 설정했을 때 분기 태스크의 인덱스가 반드시 양의 정수가 되어야 합니다. Mech-Viz 프로젝트에 지정된 인덱스를 가진 분기 태스크가 없는 경우 이 오류 코드가 나타납니다.
- Mech-Viz 에서 태스크 인덱스는 해당 태스크의 파라미터에서 읽어내거나 설정할 수 있습니다. 범위: 1-99.
  - 203 명령어 형식: *203,* **분기 태스크 인덱스**, **아웃포트 번호**.

해결 방법:

- Mech-Viz 에 명령어가 보내 온 테스크 인덱스와 대응한 태스크가 있는지 확인하십시오.
- 명령어와 Mech-Viz 프로젝트에서 설정한 태스크 인덱스는 반드시 양의 정수가 되어야 합니다.

2020

Mech-Viz 싱귤래리티 운동 실패

오류 원인:

- Mech-Viz 에서 로봇 이동 경로를 계획했을 때 싱귤래리티 오류가 나타나 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트의 이동 목표점을 체크하고 목표점의 포즈 또는 파라미터를 수정하여 싱귤래리티 오류를 방지합니다.

2021

Mech-Viz 이동 계획 실패

오류 원인:

- Mech-Viz 는 로봇이 직선 운동을 통해 이동 목표점에 도달하지 못한다는 것을 감지합니다.

해결 방법:

- Mech-Viz 에서 해당 이동 목표점의 포즈를 적당히 조정하거나 전이점을 추가하십시오.
- 괄절 운동을 선택하세요.

2022

Mech-Viz 실행되지 않음

오류 원인:

- 클라이언트 인터페이스 프로그램이 203 명령어를 호출하여 Mech-Viz 분기를 설정했을 때 프로젝트가 실행되지 않았습니다.
- 클라이언트 인터페이스 프로그램이 205 명령어를 호출하여 Mech-Viz 가 계획한 경로를 획득하기 전에 프로젝트을 트리거하지 않았습니다.
- 클라이언트 인터페이스 프로그램이 205 명령어를 호출할 때 Mech-Viz 에서 계획 결과를 출력하지 않았습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 Mech-Viz 분기를 설정하기 전에 프로젝트를 먼저 실행해야 합니다.
- 클라이언트 인터페이스 프로그램을 체크하고 Mech-Viz 를 먼저 실행해야 계획한 경로를 획득할 수 있습니다.

2023

Mech-Viz 프로젝트 파일 오류

2024

Mech-Viz 무효한 분기 명칭

오류 원인:

- 클라이언트 인터페이스 프로그램이 203 명령어를 호출하여 Mech-Viz 분기를 설정할 때 분기 태스크의 번호는 반드시 양의 정수가 되어야 합니다.
  - 203 명령어 형식: *203,* **분기 태스크 인덱스**, **아웃포트 번호**.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 설정한 Mech-Viz 분기 태스크의 번호는 반드시 양의 정수가 되어야 합니다.
- Mech-Viz 에서 태스크 인덱스는 해당 태스크의 파라미터에서 읽어내거나 설정할 수 있습니다. 범위: 1-99.

2025

Mech-Viz 실행 시 타임아웃

오류 원인:

- 클라이언트 인터페이스 프로그램이 205 명령어를 호출하여 Mech-Viz 가 계획한 경로를 획득하려고 했지만 규정한 시간 (기본값:10 초) 내에 프로젝트의 실행이 완료되지 않았습니다. 실행 시간이 제한 시간을 넘었기 때문에 오류가 발생했습니다.

해결 방법:

- 정상적인 상황에서 Mech-Viz 의 실행 시간을 확인하세요. 10 초 보다 길면 Mech-Center **구성 설정** → *Mech-Interface* → **고급 설정** 을 조절하십시오.
- 클라이언트 인터페이스 프로그램은 205 명령어를 호출하기 전에 일정한 지연 기간을 설정할 수 있습니다.

2026

Mech-Viz 무효한 인덱스 유형의 태스크 명칭

오류 원인:

- 클라이언트 인터페이스 프로그램이 204 명령어를 호출하여 Mech-Viz 에서 인덱스 파라미터를 갖춘 태스크의 인덱스를 설정할 때 설정한 인덱스 수치는 반드시 양의 정수가 되어야 합니다.
  - 204 명령어 형식: *204*, **인덱스 파라미터를 갖춘 태스크 번로**, **인덱스 수치**.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 204 명령어를 통해 Mech-Viz 인덱스를 설정할 때 인덱스 파라미터를 갖춘 태스크 번호는 반드시 양의 정수가 되어야 합니다.
- Mech-Viz 에서 태스크 인덱스는 해당 태스크의 파라미터에서 읽어내거나 설정할 수 있습니다. 범위: 1-99.

2027

Mech-Viz 무효한 인덱스 수치

오류 원인:

- 클라이언트 인터페이스 프로그램은 204 명령어를 호출하여 Mech-Viz 에서 인덱스 파라미터를 갖춘 태스크의 인덱스를 설정할 때 설정한 인덱스 수치가 반드시 양의 정수가 되어야 합니다.
  - 204 명령어 형식: *204*, **인덱스 파라미터를 갖춘 태스크 번로**, **인덱스 수치**.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하고 204 명령어를 호출하여 Mech-Viz 인덱스를 설정할 때 인덱스 수치는 반드시 양의 정수가 되어야 합니다.

2028

Mech-Viz 인덱스 설정 실패. 인덱스 명칭이 존재한지 확인하기

오류 원인:

- 클라이언트 인터페이스 프로그램이 204 명령어를 호출하여 Mech-Viz 이동 태스크의 인덱스 파라미터를 설정할 때 이동 태스크의 번호는 반드시 양의 정수가 되어야 합니다. Mech-Viz 프로젝트에서 태스크 번호와 대응하는 이동 태스크가 없는 경우 이 오류가 발생합니다.
  - 204 명령어 형식: *204*, **인덱스 파라미터를 갖춘 태스크 번호**, **인덱스 수치**.

해결 방법:

- 클라이언트 인터페이스 프로그램이 설정한 이동 태스크의 번호가 맞는지 체크하십시오.
- Mech-Viz 프로젝트의 이동 태스크의 번호가 맞는지 체크하십시오.
- Mech-Viz 에서 태스크 인덱스는 해당 태스크의 파라미터에서 읽어내거나 설정할 수 있습니다. 범위: 1-99.

2029

Mech-Viz 외부 이동 설정 실패

2030

Mech-Viz 무효한 포즈

오류 원인:

- Mech-Viz 프로젝트에서 MP-E0010 이란 오류 메시지가 나타나지만 오류 원인은 알 수 없습니다.

해결 방법:

아직 없음

2031

Mech-Viz 로봇 관절 사이의 충돌

오류 원인:

- Mech-Viz 프로젝트에서 로봇 관절 사이의 충돌을 감지하여 경로 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 이동 목표점을 체크합니다.

2032

Mech-Viz 시나리오 물체와의 충돌

오류 원인:

- Mech-Viz 프로젝트에서 로봇과 시나리오 물체 사이의 충돌을 감지하여 경로 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 시나리오 모델과 목표점을 체크하십시오.

2033

Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 포인트 수가 역치를 초과함

오류 원인:

- Mech-Viz 프로젝트에서 로봇과 물체 포인트 클라우드 사이의 충돌을 감지하고 충돌 포인트 수가 역치를 초과했기 때문에 경로 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 이동 목표점을 체크합니다.
- 필요하면 Mech-Viz 충돌 감지의 충돌 포인트 역치를 조절할 수 있습니다.

2034

Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 면적이 역치를 초과함

오류 원인:

- Mech-Viz 프로젝트에서 로봇과 물체 포인트 클라우드 사이의 충돌을 감지하고 충돌 면적이 역치를 초과했기 때문에 경로 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 이동 목표점을 체크합니다.
- 필요하면 Mech-Viz 충돌 감지의 충돌 면적 역치를 조절할 수 있습니다.

2035

Mech-Viz 포인트 클라우드와 충돌할 때의 충돌 부피가 역치를 초과함

오류 원인:

- Mech-Viz 프로젝트에서 로봇과 물체 포인트 클라우드 사이의 충돌을 감지하고 충돌 부피가 역치를 초과했기 때문에 경로 계획에 실패했습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 이동 목표점을 체크합니다.
- 필요하면 Mech-Viz 충돌 감지의 충돌 부피 역치를 조절할 수 있습니다.

2036

Mech-Viz [비전 인식] 에 이미지를 캡처하지 않음

오류 원인:

- Mech-Viz 프로젝트가 실행되는 동안 태스크 check_look 가 네번째 아웃포트 (“사진 찍지 않음”) 에서 실행되고 이 아웃포트가 다른 태스크와 연결되지 않았기 때문에 Mech-Viz 실행 과정이 중지되었습니다.

해결 방법:

- Mech-Viz 프로젝트에 있는 태스크 vision_look 및 check_look 에서 설정한 비전 서비스가 일치한지 확인하세요.

2037

Mech-Viz [비전 인식] 에 결과가 없음

오류 원인:

- Mech-Viz 프로젝트가 실행되는 동안 태스크 check_look 가 두 번째 아웃포트 (“결과 없음”) 에서 실행되고 이 아웃포트가 다른 태스크와 연결되지 않았기 때문에 Mech-Viz 실행 과정이 중지되었습니다. 대응하는 Mech-Vision 프로젝트에서도 비전 결과를 출력하지 않았습니다.

해결 방법:

- *1002* 내용을 참조하여 문제를 해결해 보세요.

2038

Mech-Viz [비전 인식] 에 ROI 내 포인트 클라우드가 없음

오류 원인:

- Mech-Viz 프로젝트가 실행되는 동안 태스크 check_look 가 다섯 번째 아웃포트 (“포인트 클라우드 없음”) 에서 실행되고 이 아웃포트가 다른 태스크와 연결되지 않았기 때문에 Mech-Viz 실행 과정이 중지되었습니다. 대응하는 Mech-Vision 프로젝트에서도 비전 결과를 출력하지 않았습니다.

해결 방법:

- *1003* 내용을 참조하여 문제를 해결해 보세요.

2039

Mech-Viz 계획에 사용될 수 있는 포즈가 없음

오류 원인:

- Mech-Viz 프로젝트가 vision_move 에 사용될 수 있는 포즈를 제공하지 않았습니다. Mech-Viz 에서 태스크 check_look 를 사용하지 않거나 태스크 check_look 가 두 번째 아웃포트 (“결과 없음”) 에서 실행되며 뒤에 태스크 [비전 이동] 가 있을 때 이 오류가 발생합니다.

해결 방법:

- *1002* 내용을 참조하여 문제를 해결해 보세요.
- Mech-Viz 프로젝트에서 태스크 check_look 를 사용했는지 체크하십시오.

2040

Mech-Viz [비전 결과를 재사용하기] 에 계획이 실패한 경로가 있음

오류 원인:

- Mech-Viz 프로젝트의 태스크 vision_move 에서 비전 결과 재사용을 선택하며 동일한 반환된 포즈로 여러 개 작업물을 피킹했는데 모든 포즈에 대해 성공적으로 이동 경로를 계획하지 못했고 계획에 실패한 포즈가 존재합니다.
- 이 오류가 발생해도 성공적으로 계획된 경로는 여전히 출력될 것입니다.

해결 방법:

- Mech-Viz 의 계획 기록을 통해 실패한 원인을 찾으세요.

2041

Mech-Viz 파라미터 설정 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 207 명령어를 호출하여 Mech-Viz 태스크 파라미터를 획득했을 때 오류가 발생했습니다.

해결 방법:

- 구성 파일 내용이 맞는지 확인하세요. Mech-Center 의 **구성 설정** → *Mech-Interface* → **고급 설정** 아래에서 구성 파일 (*Property Config*) 을 열 수 있습니다.
  - 특별히 태스크 ID 와 파라미터 툴팁이 맞는지 확인해야 합니다.

2042

Mech-Viz 이동 계획 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 201 명령어를 호출하여 Mech-Viz 태스크 vision_move 의 계획 결과를 획득했을 때 오류가 발생했습니다.

해결 방법:

아직 없음

2043

Mech-Viz 비전 목표점의 자체 정의한 데이터를 획득 실패

오류 원인:

- 클라이언트 인터페이스 프로그램이 201 명령어를 호출하여 Mech-Viz 태스크 vision_move 의 계획 결과 및 비전 결과에 있는 자체 정의한 데이터를 획득했을 때 자체 정의한 데이터를 획득하는 데 오류가 발생했습니다.

해결 방법:

- *1024* 내용을 참조하십시오.

2044

Mech-Viz 비전 서비스가 등록되지 않음

오류 원인:

- Mech-Viz 프로젝트에서 태스크 vision_look 에 비전 서비스를 올바르게 설정하지 않았습니다.

해결 방법:

- Mech-Viz 프로젝트에서 태스크 vision_look 에 비전 서비스를 올바르게 설정했는지 체크하십시오.

Mech-Center 오류 분석 및 문제 해결

3001

Mech-Center 무효한 명령어

오류 원인:

- 시스템은 클라이언트 인터페이스 프로그램에서 보내는 명령어 코드를 지원하지 않습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크합니다.

3002

Mech-Center 인터페이스 명령어 길이 또는 포맷 오류

오류 원인:

- 클라이언트 인터페이스 프로그램에서 보내 온 명령어 정보의 길이가 비정상적입니다. 예를 들어 로봇 포즈 데이터의 길이는 여섯 자리 미만입니다.
- 클라이언트 인터페이스 프로그램에서 보내 온 명령어 정보의 포맷이 비정상적입니다. 예를 들어 쉼표 (영어 반각 쉼표) 구분 기호가 사용되지 않습니다.

해결 방법:

- 클라이언트 인터페이스 프로그램에서 보내 온 명령어가 인터페이스의 요구에 부합한지 체크합니다.

3003

Mech-Center 클라이언트 연결 끊김

3004

Mech-Center 서버 연결 끊김

3005

Mech-Center Mech-Vision 을 호출할 때 타임아웃

3006

Mech-Center 알수 없는 오류

3007

Mech-Center 데이터 확인 신호 타임아웃

오류 원인:

PROFINET / EthernetIP 를 사용하여 통신할 때,

- Mech-Center 가 새 포즈 데이터를 클라이언트 인터페이스에 보내기 전에 클라이언트 인터페이스 프로그램의 Data_Acknowledge 신호가 0 인지 확인해야 합니다. 제한 시간 내에 클라이언트 인터페이스 프로그램이 Data_Acknowledge 신호를 리셋하지 않았다면 이 오류 코드가 나타날 것입니다.
- Mech-Center 가 새 포즈 데이터를 클라이언트 인터페이스에 보낸 후 제한 시간 내에 클라이언트 인터페이스 프로그램이 데이터를 이미 읽었다는 의미를 가진 1 인 Data_Acknowledge 신호를 보내지 않았다면 이 오류 코드가 나타날 것입니다.

해결 방법:

- 클라이언트 인터페이스 프로그램을 체크하십시오. 102 혹은 105 명령어를 호출하기 전에 Data_Acknowledge 신호가 0 이 되어야 합니다.
- 클라이언트 인터페이스 프로그램을 체크하십시오. 인터페이스의 포즈 데이터를 읽은 후 즉시에 Data_Acknowledge 신호를 1 로 설정해야 합니다.

3008

Mech-Center 구성 ID 가 없어서 파라미터를 읽어내거나 설정할 수 없습니다.

오류 원인:

- 클라이언트 인터페이스 프로그램이 207 명령어를 호출하여 태스크 파라미터를 획득하거나 208 명령어를 통해 태스크 파라미터를 설정했을 때 Mech-Center 의 구성 파일에서 대응한 ID 와 내용을 찾을 수 없습니다.
- Mech-Center 의 **구성 설정** → *Mech-Interface* → **고급 설정** 에서 구성 파일 (*Property Config*) 을 열 수 있습니다.

해결 방법:

- 구성 파일에 있는 구성 ID 가 양의 정수인지 체크하십시오.
- 구성 파일과 대응하는 구성 ID 및 내용이 존재한지 체크합니다.

외부 파라미터 캘리브레이션의 오류 분석

7001

캘리브레이션: 파라미터 오류

오류 원인:

- 클라이언트 인터페이스 프로그램이 캘리브레이션 과정을 수행했을 때 Mech-Center 에 보낸 로봇 포즈의 데이터 길이가 여섯 자리 미만입니다. JPs 및 플랜지 포즈의 데이터 길이는 반드시 여섯 자리가 되어야 하며 4 축 또는 5 축 로봇을 사용하는 경우 나머지 필드에 0 을 입력해야 합니다.

해결 방법:

- 클라이언트 인터페이스 프로그램에서 보내 온 로봇의 포즈 데이터 길이가 맞는지 확인합니다.

7002

캘리브레이션: Mech-Vision 이 캘리브레이션된 포즈를 출력하지 않음

오류 원인:

- Mech-Vision 에서 캘리브레이션 과정을 진행했을 때 Mech-Vision 이 로봇의 포즈를 Mech-Center 에 보내지 않았습니다.

해결 방법:

아직 없음

7003

캘리브레이션: 로봇이 캘리브레이션 포인트에 도달하지 못함

3.4 Adapter

표준 인터페이스의 기능으로 프로젝트의 수요를 충족하지 못했을 때 Adapter 프로그램을 작성함으로써 복잡한 컨트롤 과정을 실현할 수 있습니다.

이 부분에는 주로 아래 내용을 포함합니다.

- *Adapter* **퀵 가이드**
- *Adapter* **생성기 매뉴얼**
- *Adapter* **프로그래밍 가이드**
- *Adapter* **프로그래밍 샘플**

팁: Adapter 프로젝트는 다른 Python 라이브러리를 사용하려면 Mech-Center 소프트웨어의"python" 디렉터리에 설치하십시오. python 라이브러리의 설치 방법은 아래와 같습니다.

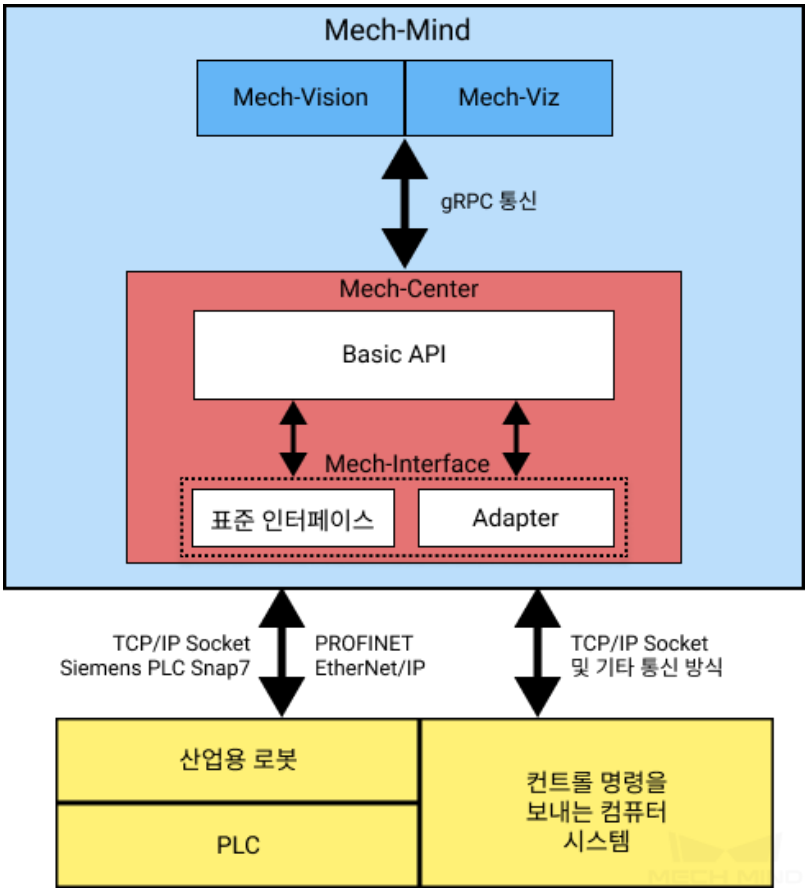
1. "CMD" 또는"PowerShell" 프로그램을 엽니다.
2. Mech-Center 소프트웨어의"python" 디렉터리로 이동합니다. 예: C:\Mech-Mind\Mech-Center-1.6.x\python.
3. " **./python -m pip install** *library_name* " 명령어를 실행합니다.

3.4.1 Adapter 퀵 가이드

- *Adapter* **에 관한 소개**
- *Adapter* **기능**
- *Adapter* **개발**
- *Adapter* **프로젝트 배포**

Adapter 에 관한 소개

Adapter 는 Mech-Center 소프트웨어의 통신 컴포넌트로 Basic API 인터페이스를 통해 Mech-Vision, Mech-Viz 와 gRPC 통신을 진행하며 외부 장치와 TCP/IP Socket, HTTP 또는 PLC 데이터 블록 전송 프로토콜 (예를 들어 PLC MC) 등과 같이 다양한 산업적 통식 방식으로 통신을 진행합니다.



Adapter 의 응용 시나리오와 관련된 정보는“**사용 설명**”내용을 참조하십시오.

Adapter 기능

Adapter 를 통해 다음과 같은 기능을 실현할 수 있습니다.

- **내부에서 Mech-Vision 과 Mech-Viz 소프트웨어를 제어할 수 있습니다.**

유형	기능
Mech-Vision 과 관련된 기능	소프트웨어를 실행하고 Mech-Vision 의 비전 처리 결과를 획득하기
	Mech-Vision 스텝의 파라미터를 설정하기
	Mech-Vision 스텝의 파라미터를 읽어내기
	Mech-Vision 파라미터 레시피
Mech-Viz 와 관련된 기능	Mech-Viz 를 시작하기
	Mech-Viz 를 중지하기
	Mech-Viz 태스크 파라미터를 설정하기
	Mech-Viz 태스크 파라미터를 읽어내기
	클램프 번호를 설정하기
	로봇의 실행 속도를 설정하기
	포인트 클라우드의 충돌 파라미터를 설정하기
	Mech-Viz 에서 반환된 실행 상태를 획득하기
기타	구체적인 기능은“*Adapter* **프로그래밍 가이드**”내용을 참조하십시오.

- **외부에서 사용자 인터페이스, 데이터 베이스, 파일 읽기 및 쓰기, Web 시스템과의 통신 등 비전 이외의 기능을 실현할 수 있습니다.**

외부적인 기능은 Python 프로그래밍을 통해 실현되어야 합니다.

Adapter 개발

TCP/IP Socket 통신 방식을 사용할 때 Mech-Center 에 내장되어 있는 **Adapter 생성기** 기능을 통해 Adapter 를 처음으로 사용하는 사용자도 Adapter 프로그램을 신속하게 생성하고 Adapter 프로젝트를 구축할 수 있습니다. 상세한 정보는"*Adapter* **생성기 매뉴얼**"내용을 참조하십시오.

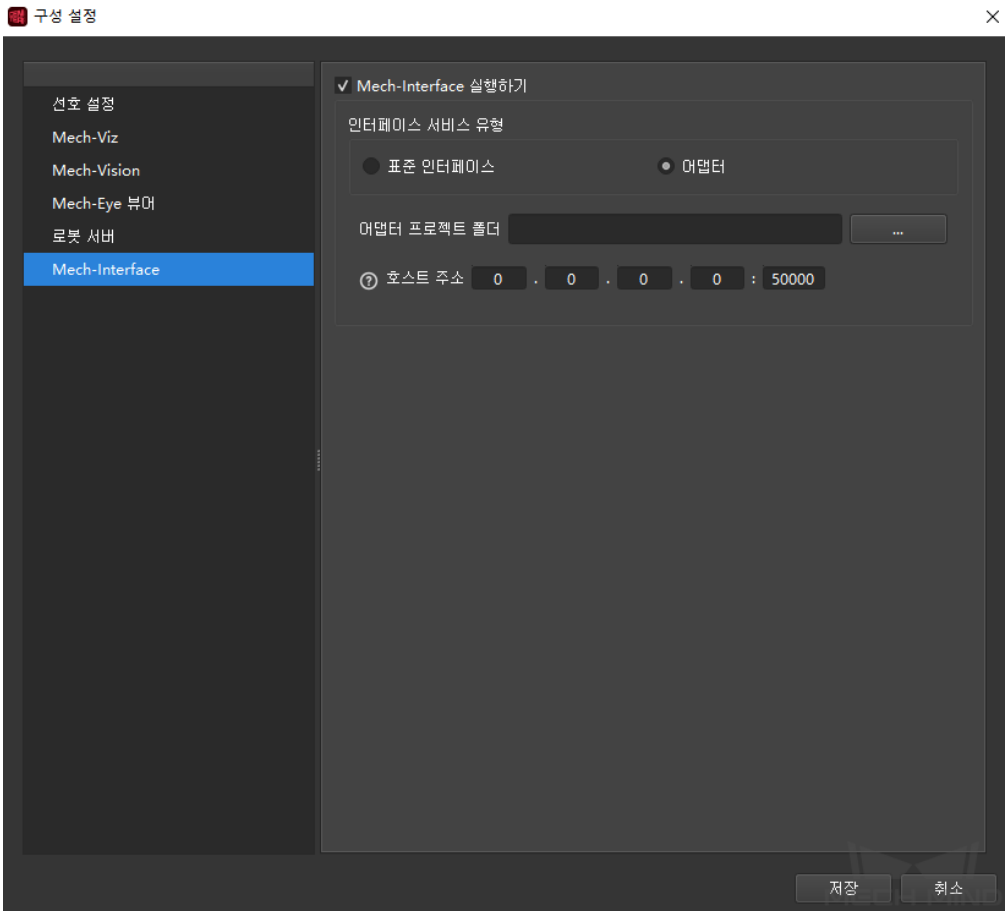
이미 생성된 Adapter 를 기반으로 하여 프로그램의 2 차 개발을 진행할 수 있습니다.

물론 처음부터 끝까지 완전한 어댑터 프로그램을 자체적으로 작성할 수도 있습니다. "*Adapter* **프로그래밍 가이드**"및"*Adapter* **프로그래밍 샘플**"내용을 참조하십시오.

Adapter 프로젝트 배포

Adapter 프로그램을 작성한 다음에 다음 단계를 참조하여 Adapter 프로젝트를 배포하세요.

1. Mech-Center 소프트웨어를 열어 툴바에 있는 **구성 설정** 버튼을 클릭하십시오.
2. **구성 설정** 창에서 **Mech-Interface** 를 선택하고 **Mech-Interface 실행하기** 옵션을 선택하며 **인터페이스 서비스 유형** 을 **Adapter** 로 설정하세요. 아래 그림과 같습니다.



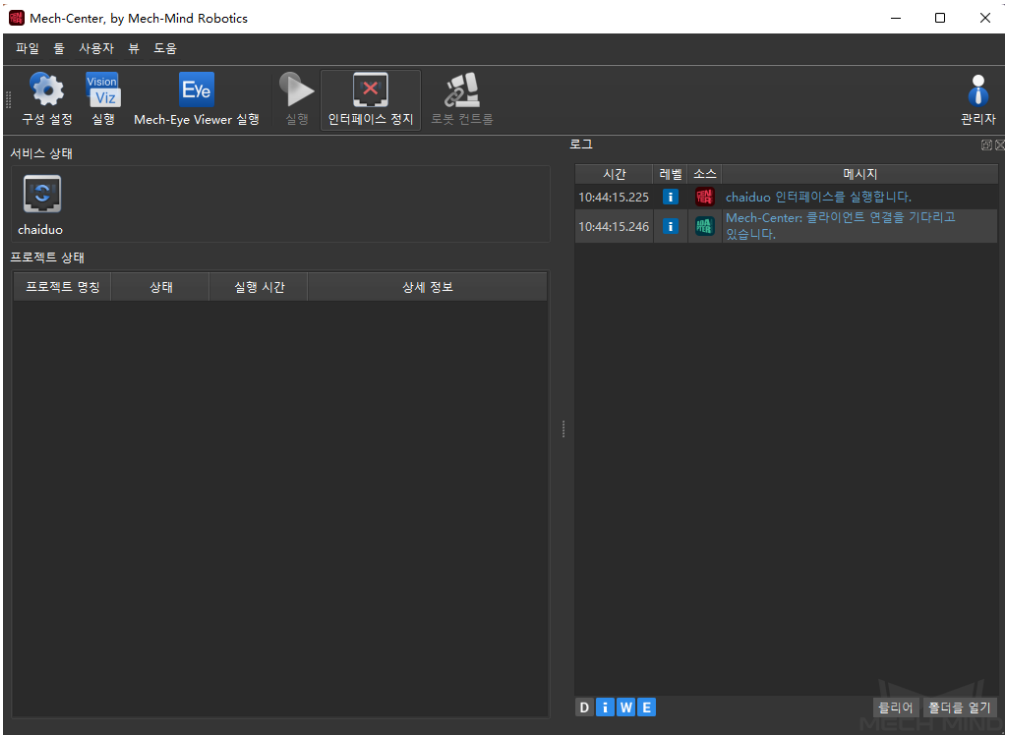
3. **어댑터 프로젝트 폴더** 를 Adapter 프로그램이 저장되는 디렉터리로 설정하세요.

4. 작업 현장에 실제 상황에 근거하여 **호스트 주소** 를 설정하세요. 포트는 상대 쪽의 포트와 일치해야 합니다.

- 상대 쪽이 통신의 서버 역할을 하면 **호스트 주소** 는 상대 쪽의 IP 주소로 설정해야 합니다.
- 상대 쪽이 통신의 클라이언트 역할을 하면 **호스트 주소** 는“0.0.0.0”로 설정해야 합니다.

5. **저장** 버튼을 클릭하고 Mech-Center 소프트웨어를 다시 시작하세요.
6. 툴바에 있는 **인터페이스 실행** 버튼을 클릭하여 Adapter 서비스를 시작하세요.

인터페이스 실행 버튼이 **인터페이스 정지** 버튼으로 변하고 서비스 표시줄에 이미 실행된 Adapter 프로그램이 표시되면 Adapter 서비스가 이미 성공적으로 시작되었다는 뜻입니다. 아래 그림과 같습니다.



Adapter 에 대해 이해한 후“*Adapter* **생성기 매뉴얼**”내용을 참조하여 첫 번째 Adapter 프로그램을 신속하게 생성할 수 있습니다.

3.4.2 Adapter 생성기 매뉴얼

이 부분에서는 Adapter 생성기와 Adapter 생성기를 사용하여 Adapter 프로그램을 빠르게 생성하는 방법에 대해 설명합니다.

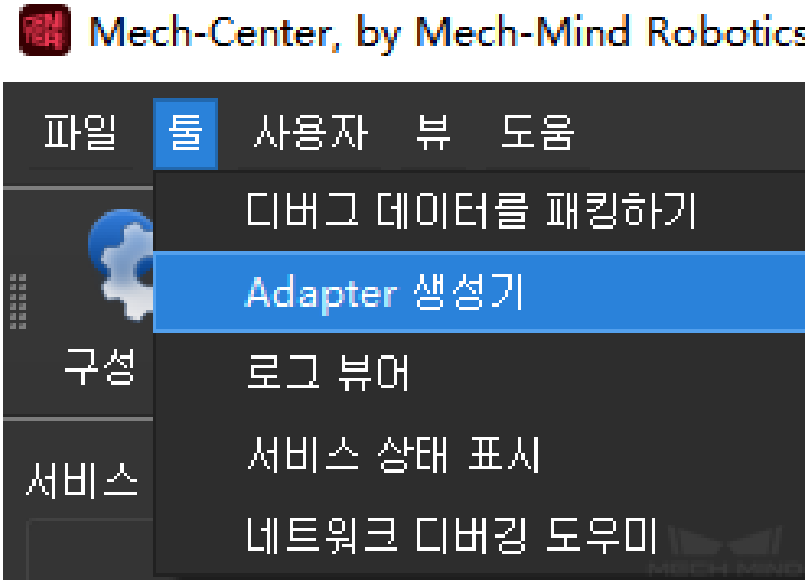
Adapter 생성기 소개

Adapter 생성기는 Mech-Center 에 통합된 작은 구성 요소입니다. Adapter 생성기는 TCP/IP Socket 통신 방식만 사용하고 Mech-Vision 만 사용하여 포즈를 제공하는 시나리오에만 적합합니다.

Adapter 생성기는 다음을 도와줄 수 있습니다.

- Adapter 프로그램을 빠르게 생성하고 어댑터 프로젝트를 구축합니다.
- Adapter 프로그래밍과 관련된 내용을 신속히 파악하여 복잡한 실제 요구 사항을 충족시킵니다.

Adapter 생성기를 시작하여 아래 이미지와 같이 **툴** → *Adapter* **생성기** 를 선택하여 관련 설정을 진행할 수 있습니다.



주의: Adapter 생성기는 관리자 모드에서만 사용할 수 있습니다.

Adapter 생성기를 사용하여 Adapter 프로그램을 빠르게 생성하기

힌트: 구성의 편의를 위해 해당 구성 요소에 대한 자세한 설명이 있으며 구성 요소 위에 마우스 커서를 올려 놓으면 해당 구성 요소를 볼 수 있습니다.

네트워크 구성 - 서버 또는 클라이언트

이 스텝에서 **Adapter 명칭**, **Adapter 의 역할**, **스트림 포맷** 파라미터를 설정하고 아래와 같이 **다음으로** 를 클릭합니다.

파라미터 설명 :

- **어댑터 명칭** 어댑터 프로그램의 명칭을 지정합니다.
- **Adapter 의 역할**: Adapter 를 TCP/IP Socket 통신을 위한 서버 (Server) 또는 클라이언트 (Client) 로 지정합니다. Adapter 가 클라이언트 역할을 하고 서버가 클라이언트에 대한 포트 제한이 있는 경우 **포트 바인딩** 확인란을 선택해야 합니다.

  > **힌트:** 정상적인 통신을 하려면 Adapter 프로젝트가 배포될 때 올바른 호스트 IP 주소와 포트를 지정해야 합니다. 자세한 내용은“*Adapter* **프로젝트 배포**”부분을 참조하십시오.

- **스트림 포맷**: 통신의 전송 형식을 지정하고 ASCII 문자열 또는 HEX(16 진법) 를 지원합니다. **스트림 포맷** 형식이“HEX”로 설정된 경우 바이트 순서 (endianness) 도“빅 엔디안”또는“리틀 엔디안”으로 지정해야 합니다.

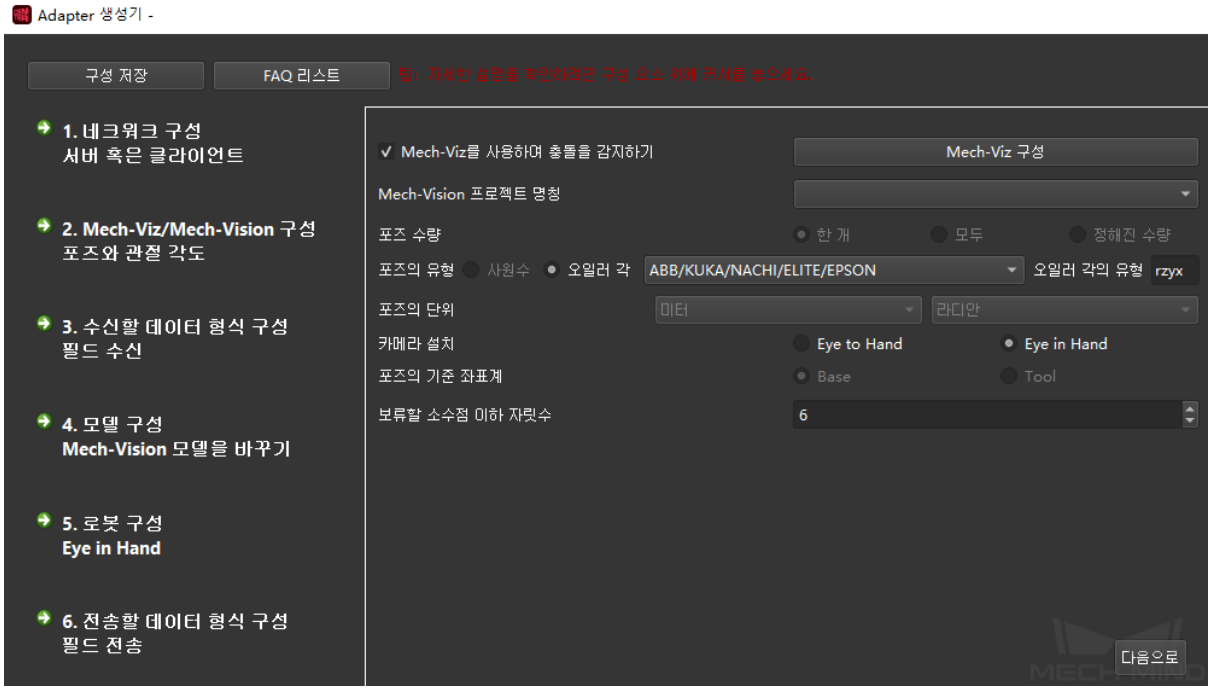
Mech-Viz/Mech-Vision 구성 - 포즈 및 관절 각도

힌트:

- 이 스텝을 수행하기 전에 Mech-Vision 프로젝트와 충돌 감지에 사용되는 Mech-Viz 프로젝트를 준비하고 자동 로드 옵션을 활성화하여 프로젝트가 Mech-Center 에 로드되었는지 확인합니다.
- Mech-Center 는 충돌 감지를 위한 Mech-Viz 샘플 프로젝트“check_collision”을 제공합니다. Mech-Center 설치 경로 아래의“tool\viz_project”디렉터리에서 획득하십시오.

주의: “check_collision”샘플 프로젝트에서 볼 수 있듯이 프로젝트는 사진을 찍기 위해 vision_look 에 의해 실행되며 비이동류 태스크를 포함해야 합니다. 작업 이름은 notify_1, notify_2 및 vision_look_1 을 포함하여 변경할 수 없습니다.

이 스텝에서 Mech-Viz 및 Mech-Vision 프로젝트와 관련된 파라미터를 설정하고 **다음으로** 를 클릭합니다.

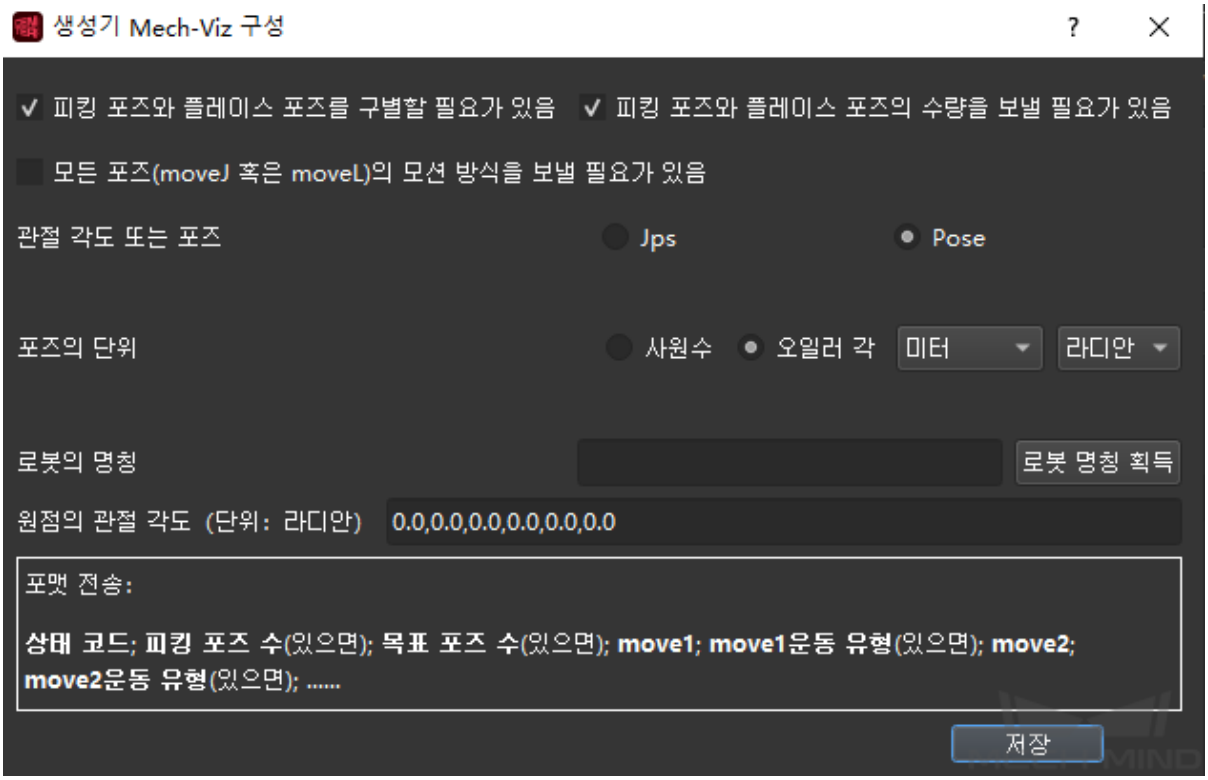


Mech-Vision 관련 파라미터 설명 :

- **Mech-Vision 프로젝트 명칭**: Mech-Vision 프로젝트의 이름을 지정합니다. 드롭다운 리스트에서 Adapter 와 상호 작용할 Mech-Vision 프로젝트를 선택합니다.
- **포즈 수**: 대상에게 보낼 포즈 수를 선택합니다.
- **포즈 형식**: 사원수 또는 오일러 각을 선택할 수 있습니다.
- **포즈 단위**: 일반적으로 밀리미터와 도를 사용하도록 선택합니다.
- **카메라**: 카메라 설치 방법을 선택합니다. ETH 와 EIH 의 두 가지 경우로 나뉩니다.
- **포즈 좌표계**: 전송된 포즈의 기반이 되는 좌표계를 결정합니다. 일반적으로 포즈는 로봇 기준 좌표계를 기반으로 합니다. EIH 시나리오에서 로봇이 로봇 말단 포즈를 제공할 수 없는 경우 포즈는 TCP 좌표계만 기반으로 할 수 있습니다.
- **소수점 이하 자릿수 유지**: 전송된 포즈의 소수점 이하 자릿수를 결정합니다. 최대 자릿수는 10 입니다.

Mech-Viz 관련 파라미터 설명:

- **Mech-Viz 를 사용하여 충돌 감지**: 선택 후 비전 포인트는 Mech-Viz 프로젝트의 충돌 감지를 거쳐 계획에 실패한 포인트를 필터링하고 프로세스에서 충돌이 없는 피킹 포즈를 골라냅니다.
- **Mech-Viz 구성**. 이 버튼은 *Mech-Viz* **를 사용하여 충돌 감지** 가 선택된 경우에 사용할 수 있습니다. 이 버튼을 클릭하면 **생성기 Mech-Viz 설정** 인터페이스가 나타납니다. 해당 파라미터를 설정한 후 **저장** 버튼을 클릭합니다. 아래 그림과 같습니다.

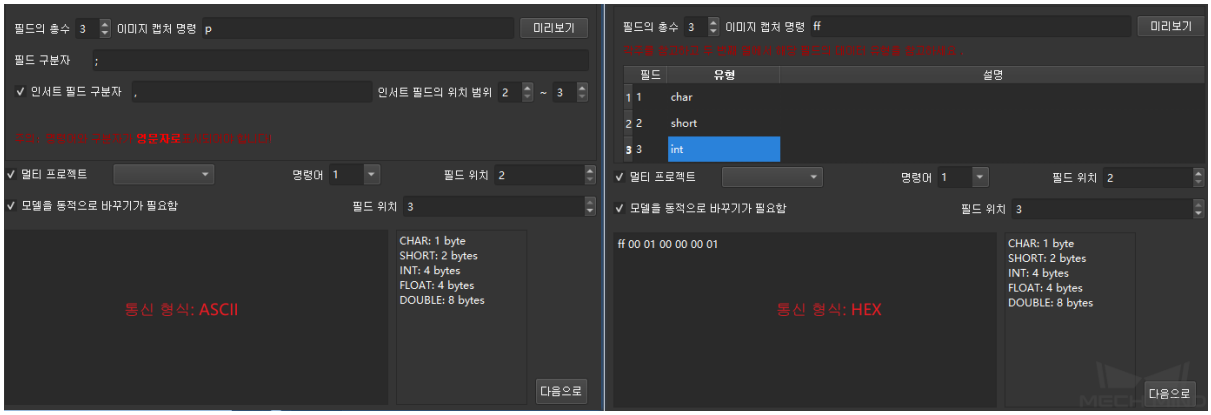


- **피킹 포인트 위치와 배치 포인트 위치를 구분해야 함**: 피킹 포인트 위치는 **비전 이동** (비전 이동 포함) 이전의 모든 포인트 위치이고 배치 포인트 위치는 **비전 이동** 이후의 모든 포인트 위치입니다. 일부 시나리오에서 로봇은 태스크에 따라 피킹과 배치 동작을 구별해야 할 수도 있습니다.
- **피킹 포인트 위치와 배치 포인트 위치를 전송해야 함**: 포인트 수가 많을 경우 수량 필드를 가져올 수 있습니다. 이 필드는 체크 후 기본 포인트 수가 1 보다 큰 경우에만 포함됩니다.
- **각 포인트 위치의 이동 방식을 전송해야 함**: Mech-Viz 에서 이동류 태스크의 이동 방식은 관절 이동 또는 직선 이동으로 구분됩니다.
- **수치 업데이트**: 기본 관절 이동 해당 코드는 1 이고 직선 이동 해당 코드는 2 입니다. 코드는 사용자 정의할 수 있습니다. 변경 후 **수치 업데이트** 버튼을 클릭하면 적용됩니다.
- **Jps 또는 포즈**: 포즈의 형식을 전송, 기본적으로 Jps 가 사용됩니다. 포즈를 선택하는 경우 Mech-Viz 의 **기타** 패널에서 *TCP* **포즈를 전송하기** 확인란을 선택해야 합니다. 아래 이미지와 같습니다.

- **Jps 단위/포즈 단위**: 보내는 포즈 형식이 **Jps** 인 경우 Jps 단위를 설정하고 일반적으로 도를 사용합니다. 전송된 포즈 형식이 **Pose** 인 경우 일반적으로 밀리미터를 사용하여 포즈 단위를 설정합니다.
- **로봇의 명칭**: 로봇 서비스의 명칭을 지정합니다. Mech-Viz 를 사용하여 로봇 동작을 시뮬레이션하려면 실제 로봇 서비스가 필요합니다. 생성된 어댑터는 이 서비스를 시뮬레이션합니다. 로봇 명칭은 이 서비스의 명칭이며 Mech-Viz 의 로봇 명칭과 동일해야 합니다. **로봇 명칭을 획득하기** 를 클릭하면 로봇 명칭을 자동으로 추가할 수 있습니다.
- **원점의 관절 각도 (단위: 라디안)** : Mech-Viz 에서 시작 동작을 위한 기준 원점을 설정합니다. 단위는 쉼표로 구분된 라디안입니다. Mech-Viz 에서 **이동** 을 원점으로 편집하여 해당 원점의 관절 위치를 복사할 수 있습니다.

접수할 데이터 형식 구성 - 접수할 필드

이 스텝에서는 사진 캡처 명령어, 여러 프로젝트 (명령어 코드), 총 필드 수, 필드 유형, 필드의 필드 구분자 및 하위 필드 구분자를 포함하여 접수된 필드의 형식을 설정한 다음 **다음으로** 를 클릭합니다. 아래 그림과 같습니다.



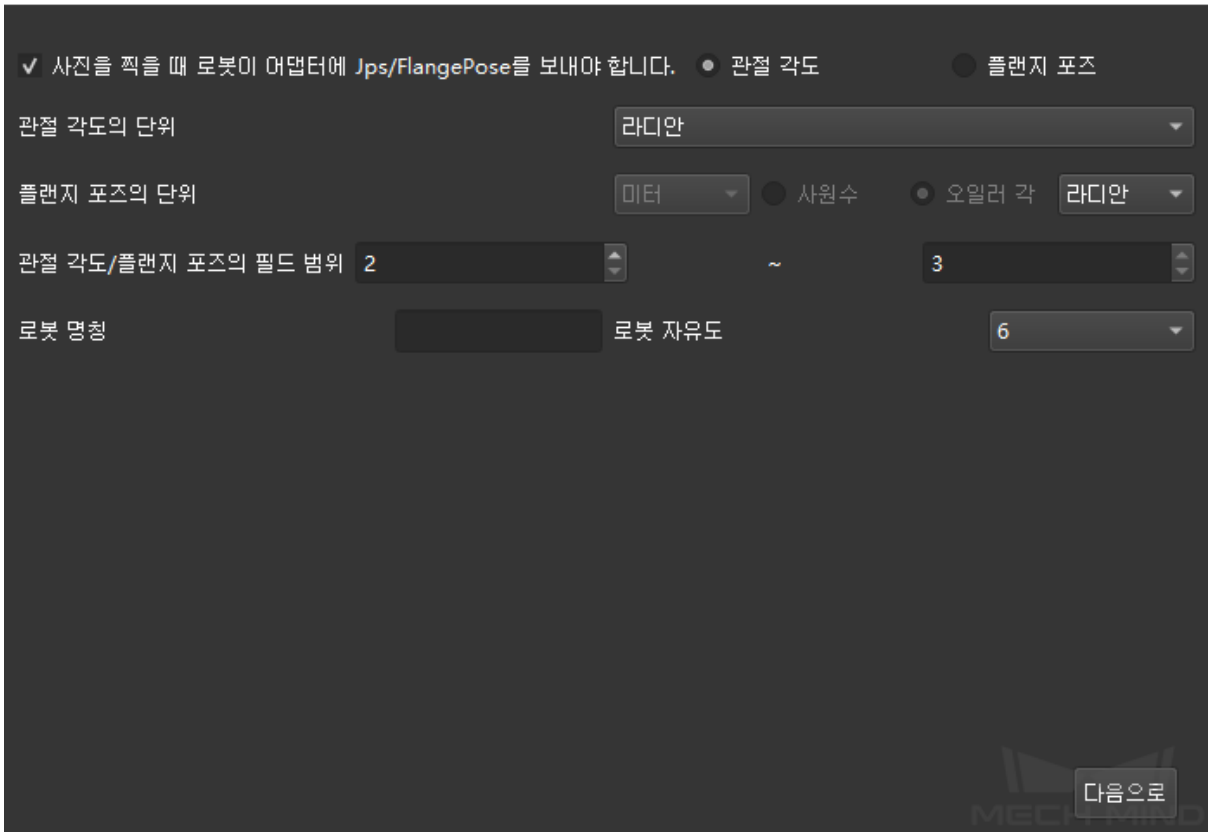
- **총 필드 수**: 설정하고자 하는 파라미터의 개수와 관련이 있으며, 값 범위는 1~10 입니다. 현장에 사진 캡처 명령어가 있어야 합니다.
- **이미지 캡처 명령어**: 카메라가 사진을 캡처하고 수집하도록 하기 위해 외부 통신 장치에서 Mech-Mind 소프트웨어 시스템으로 보내는 이미지 캡처 명령어입니다. 통신 형식이 ASCII 코드인 경우 *p* 와 같은 문자를 사용하는 것이 좋으며 필드 위치는 기본적으로 1 입니다. 통신 형식이 16 진법 (HEX) 인 경우 *0xff* 또는 *ff* 와 같이 16 진법 형식의 정수가 필요합니다.
- **필드 구분자** 및 **하위 필드 구분자**: 통신 형식이 ASCII 인 경우 설정해야 합니다. 2 개 이상의 필드가 있는 경우 필드 구분자를 써야 하며, 추가 정보에 다른 구분자가 필요한 경우 하위 필드 구분자도 필요하며 하위 필드의 시작 및 끝 범위를 지정할 수 있습니다.
- **필드 유형** 및 **필드 설명**: 통신 형식이 16 진법 (HEX) 인 경우 설정해야 합니다. 선택적 필드 유형은 CHAR, SHORT, INT, FLOAT 및 DOUBLE 입니다. **필드 설명** 은 필드의 기능이나 의미를 설명하는 데 사용됩니다.
- **여러 프로젝트 (명령어 코드)** : 선택 사항, 프로젝트에 여러 Mech-Vision 프로젝트가 있는 경우 외부 지침에 따라 다른 Mech-Vision 프로젝트를 호출해야 하며 명령어 코드를 구성할 수 있습니다.

> **주의:** 각 프로젝트의 해당 명령어 코드는 유일하고 필드 위치도 유일하며 다른 필드와 겹칠 수 없습니다.

로봇 구성 - Eye In Hand

힌트: "*Mech-Viz/Mech-Vision* 구성 - 포즈 및 관절 각도"스텝에서 **카메라** 파라미터가 **Eye In Hand** 로 설정된 경우 이 스텝을 사용할 수 있습니다.

이 스텝에서는 사진 캡처 시 로봇의 포즈 형식을 설정하고 **다음으로** 버튼을 클릭합니다. 아래 그림과 같습니다.



파라미터 설명 :

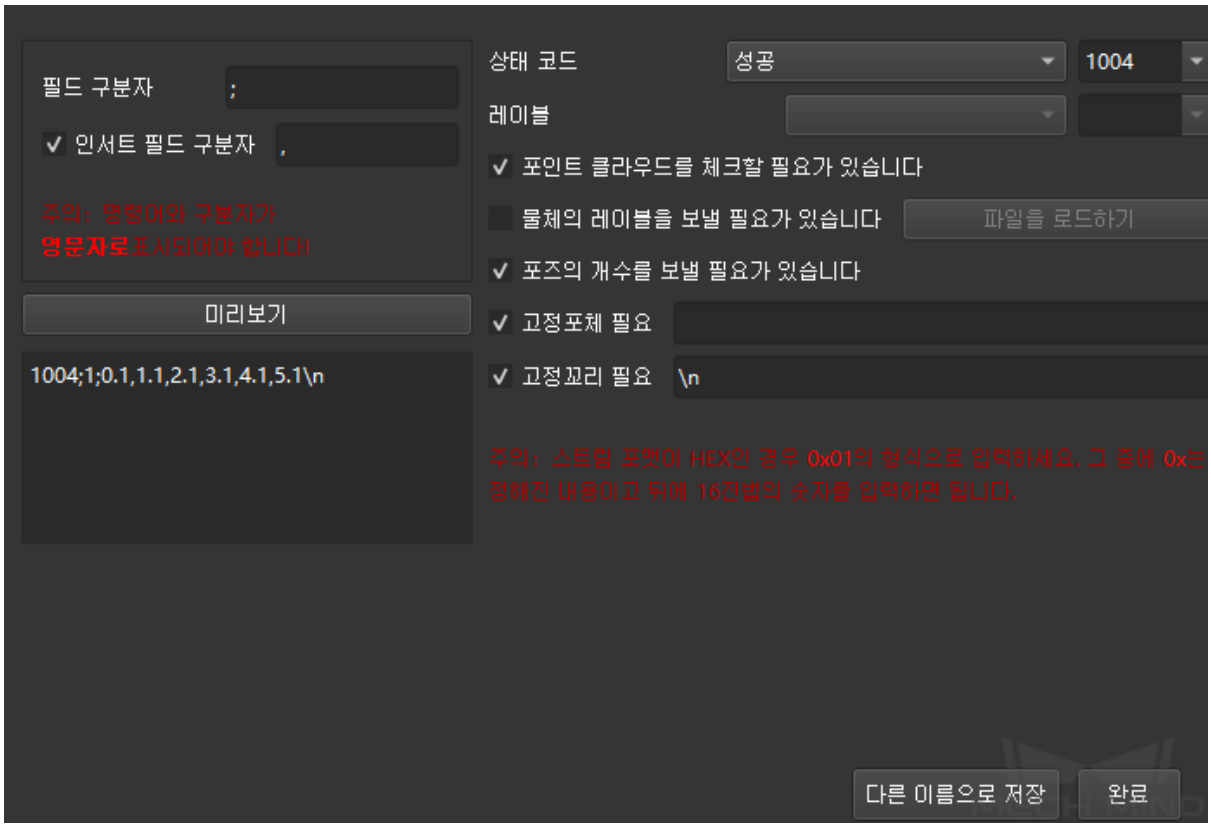
- **로봇은 Adapter 에게 사진을 캡처할 때의 관절 각도 또는 플랜지 포즈를 피드백 해야 합니다**: 상대방이 로봇 기본 좌표계의 물체 포즈를 기반으로 해야 하는 경우 로봇이 사진을 캡처할 때의 관절 각도 또는 플랜지 포즈를 제공해야 합니다. 이때 확인란을 선택하십시오. 체크한 후 포즈의 형태를 지정하고 관절 각도 또는 플랜지 포즈를 선택할 수 있습니다.
- **관절 각도 단위**: 관절 각도 단위를 지정합니다. 각도 또는 라디안을 선택할 수 있습니다.
- **플랜지 포즈 단위**: 플랜지 포즈 단위를 지정합니다. 사원수 또는 오일러 각을 선택할 수 있습니다. 사원수를 사용하는 경우 단위는 미터 또는 밀리미터를 선택할 수 있고, 오일러 각을 사용하는 경우 단위는 도 또는 라디안을 선택할 수 있습니다.
- **관절 각도/플랜지 포즈 필드 위치**: 관절 각도 또는 포즈의 시작 및 끝 필드가 전체 필드에 있는 위치를 지정합니다.

  > **주의:** 인덱스 위치는 1 부터 시작하며 다른 필드와 겹칠 수 없습니다. 예를 들어 인덱스 위치 1 은 사진 캡처 명령어입니다!

- **로봇 이름**: 로봇 서비스를 식별하는 데 사용되는 이름으로 Mech-Viz 의 로봇 이름과 동일해야 합니다.
- **로봇 자유도**: 현재 4 축, 6 축 로봇을 지원하며, 프로젝트의 실제 상황에 따라 해당 로봇의 자유도를 선택할 수 있습니다.

전송할 데이터 형식 구성 - 전송할 필드

이 스텝에서 전송할 포즈의 형식을 지정하고 **다른 이름으로 저장** 을 클릭합니다. 아래 그림과 같습니다.



파라미터 설명 :

- **필드 구분자** 및 **하위 필드 구분자**: 구분자의 형식을 설정합니다. 통신 형식이 ASCII 인 경우 설정해야 합니다. 2 개 이상의 필드가 있는 경우 필드 구분자를 써야 하며, 추가 정보에 다른 구분자가 필요한 경우 하위 필드 구분자도 필요하며 하위 필드의 시작 및 끝 범위를 지정할 수 있습니다.
- **상태 코드**: 전송 상태를 설정합니다. 각 상태에 해당하는 상태 코드는 유일합니다.
- **포인트 클라우드 확인 필요**: 확인 후 어댑터는 포인트 클라우드를 확인하고, 포인트 클라우드가 존재하지 않는 경우 해당 상태 코드를 출력합니다.
- **물체 레이블을 보내야 함**: 물체 레이블을 보낸다는 것은 Mech-Vision 에서 식별한 레이블을 상대방에게 보내는 것을 의미하며, 각 레이블은 포즈의 뒤에 연결되어 있습니다. 상대방이 레이블 문자열을 파싱하기 불편한 경우 해당 레이블의 코드도 지정할 수 있으며 모든 레이블 문자열을 포함하는 레이블 파일을 로드해야 합니다 (주의: 레이블 파일 형식은 json 배열 형식이어야 합니다).
- **수량을 전송해야 함**: 전송할 때 현재 전송된 포즈의 수량을 포함합니다.
- **바디를 고정시켜야 함**: 비전 인식 실패 시 상대방에게 메시지를 보냅니다 (오류 코드 뒤의 메시지).
- **꼬리를 고정시켜야 함**: 체크하면 전송한 데이터 뒤에 꼬리 고정 표시가 추가됩니다.

> **주의:** 통신 형식이 16 진법 (HEX) 인 경우 상태 코드, 포즈 수, 포즈 필드의 값 유형을 설정해야 합니다.

위의 모든 구성을 완료한 후 **완료** 또는 **다른 이름으로 저장** 을 클릭하여 Adapter 프로젝트를 저장합니다. Adapter 프로젝트를 저장한 후"*Adapter* **프로젝트 배포**"내용을 참조하여 Adapter 프로젝트를 배포 & 사용할 수 있습니다.

Adapter 의 프로그래밍과 관련된 더 많은 정보는 아래 내용을 읽으십시오.

- *Adapter* **프로그래밍 가이드**
- *Adapter* **프로그래밍 샘플**

3.4.3 Adapter 프로그래밍 가이드

이 부분에서는 Adapter 프로그래밍 스타일 규범과 프로그래밍 구문 지식을 소개합니다.

타겟 독자

이 부분 내용은 주로 Adapter 개발자를 위한 것입니다.

Adapter 개발자는 다음 지식을 구비해야 합니다.

- Python 기초 문법
- JSON 데이터 형식

먼저 **Adapter 프로그래밍 스타일 규범** 을 읽는 것을 권장합니다.

Adapter 프로그래밍 스타일 규범

Adapter 는 Python 언어로 작성되었으므로 Adapter 개발자는 Python 프로그래밍 규범 을 엄격히 준수해야 합니다.

또한 Adapter 프로그래밍의 품질과 가독성을 향상시키기 위해 Adapter 개발자는 이 부분의 규칙도 따라야 합니다.

명명 규범

1. 클래스 이름은 카멜 표기법을 사용하십시오. 예를 들면 다음과 같습니다.

```
class AdapterWidget
```

2. 클래스의 맴버 변수와 맴버 함수는 밑줄로 연결된 소문자 단어를 사용합니다. 예를 들면 다음과 같습니다.

```
self.pick_count # Note that variables are generally nouns
def start_viz(self): # Note that the function name is generally a verb + object
    pass
```

3. 클래스의 맴버 변수와 맴버 함수가 클래스 내부에서만 사용되는 경우 이름 앞에 밑줄을 추가하여 이름이 클래스 외부에서는 사용하지 않는 것이 좋지만 클래스 외부에서는 계속 사용할 수 있음을 나타냅니다. 예를 들면 다음과 같습니다.

```
self._socket = socket() # Indicates that the _socket variable is only used inside the␣
↪class
def _init_widget(self): # Indicates that the _init_widget function is only used inside␣
↪the class
    pass
```

4. 상수는 밑줄로 연결된 대문자를 사용합니다. 예를 들면 다음과 같습니다.

```
ADAPTER_DIR = "D:/adapter_for_communication"
```

주석 규범

주석은 너무 많으면 안 되며 표현이 복잡하거나 표현된 의미가 비교적 중요할 때만 추가하면 됩니다.

- 한 줄 주석

  # 으로 한 줄 주석을 시작합니다.

- 여러 줄 주석

  """및"""사이에 여러 줄 주석을 추가하십시오.

- 함수, 클래스 또는 패키지에 대한 소개 주석

  주석은 함수 아래, 클래스 아래 또는 패키지 상단에 있습니다. 예를 들면 다음과 같습니다.

```
def viz_is_running(self):

    """
    Will be called when find viz is running during starting viz.
    """

class Adapter(QObject):

    """
    Base class which encapsulates Viz/Vision/Hub/Robserver inter faces.
    """

"""
Service base classes.
"""

import logging
```

로그 규범

로그는 오류 발생 시 문제를 분석하는 데 도움이 될 수 있습니다. 로그 정보는 적절한 경우에 출력하는 것이 좋습니다. 예를 들면 핵심 함수를 호출하고 함수가 참조할 수 있는 데이터를 반환하는 경우입니다.

Adapter 는 두 가지 로그 프린트 방법을 지원합니다.

- print: 이 방법은 로그만 콘솔에 출력합니다. 이 방법은 런타임에 실시간 관찰에 편리하지만 프로그램이 실패하면 메시지가 손실됩니다. 따라서 실제 생산 환경에서는 사용하지 않는 것이 좋습니다.
- logging: 이 방법은 형식화된 로그 표시 형식을 지원하고 로그 파일에 로그 저장도 지원합니다 (옵션 기능). 따라서 이 방법을 사용하는 것이 좋습니다.

다음으로 Adapter 프로그래밍에 관련된 **추상적인 상위 클래스 인터페이스** 와 **Util** 패키지에 익숙해지고 Adapter 의 가장 기본적인 상위 클래스와 자주 사용하는 함수를 파악합니다.

추상적인 상위 클래스 인터페이스

추상적인 상위 클래스 인터페이스는 하위 클래스가 상위 클래스를 상속받을 때 실제 필요에 따라 다시 작성할 수 있는 함수를 말합니다. 이 부분에서는 다음과 같은 추상적인 상위 클래스를 소개합니다.

Communication

통신 관련 클래스의 소스 파일은 Mech-Center 소프트웨어 설치 경로 아래의 `/src/interface/communication.py` 파일에 있습니다.

Communication 클래스

Communication 클래스는 통신을 담당하는 기본 클래스로 일련의 인터페이스를 제공하며, 서버나 클라이언트는 이 클래스의 인터페이스 함수를 다시 작성해야 합니다.

클래스 함수	설명
is_connected()	현재 연결이 끊어졌는지 확인
set_recv_size()	수신된 데이터의 길이를 설정합니다. 기본값은 1024 바이트입니다.
send()	인터페이스 함수, 데이터 전송
recv()	인터페이스 함수, 데이터 수신
close()	인터페이스 함수, 연결 닫기
before_recv()	인터페이스 함수, 데이터를 수신하기 전에 실제 상황에 따라 논리를 추가할 수 있으며 이 함수를 다시 작성할 수 있습니다.
after_recv()	인터페이스 함수, 데이터를 수신한 후 실제 상황에 따라 논리를 추가할 수 있으며 이 함수를 다시 작성할 수 있습니다.
after_handle()	인터페이스 함수, 데이터를 처리한 후 실제 상황에 따라 논리를 추가할 수 있으며 이 함수를 다시 작성할 수 있습니다.

TcpServer 클래스

TcpServer 클래스는 TCP/IP Socket 서버를 캡슐화합니다.

클래스 함수	설명
bind_and_listen()	바인드 포트
local_socket()	기본 로컬 Socket 정보 제공
remote_socket()	원격 Socket 정보 제공
accept()	클라이언트 연결 수락
send()	데이터 전송
recv()	데이터 수신
close()	Socket 연결 닫기
close_client()	클라이언트 연결 닫기

TcpClinet 클래스

TcpClient 클래스는 TCP/IP Socket 클라이언트를 캡슐화합니다.

클래스 속성	설명
is_bind_port	포트 바인딩 여부, 서버가 연결된 클라이언트의 포트를 제한하는 경우 이 변수는 True 여야 합니다.

클래스 함수	설명
send()	데이터 전송
recv()	데이터 수신
close()	연결 닫기
set_timeout()	제한 시간을 설정합니다. 파라미터의 단위는 초입니다.
reconnect_server()	서버 다시 연결
after_connect_server()	인터페이스 함수, 처음으로 서버 연결 성공 후의 작업
after_reconnect_server()	인터페이스 함수, 서버 재접속 성공 후의 작업
after_timeout()	인터페이스 함수, 타임아웃 후의 작업

Adapter

Adapter 관련 클래스의 소스 파일은 Mech-Center 소프트웨어 설치 경로 아래의 `/src/interface/adapter.py` 파일에 있습니다.

Adapter 상위 클래스

Adapter 는 Mech-Viz 시작, Mech-Viz 중지, 태스크 파라미터 설정, 스텝 파라미터 설정 및 Mech-Vision 인식 시작과 같은 기능을 포함하여 Mech-Viz, Mech-Vision, Mech-Center 및 Robserver 와 관련된 호출을 캡슐화합니다. Adapter 프로그램이 Mech-Viz 또는 Mech-Vision 을 호출할 때마다 Adapter 클래스는 Adapter 의 하위 클래스여야 합니다.

Adapter 클래스 속성은 아래와 같습니다.

클래스 속성	설명
viz_project_dir	현재 Mech-Viz 프로젝트 경로
vision_project_name	현재 Mech-Vision 프로젝트 경로
is_simulate	Mech-Viz 시뮬레이션 실행 여부
is_keep_viz_state	Mech-Viz 가 마지막으로 중지되었을 때의 상태 그대로 계속 유지할지 여부
is_save_executor_data	Mech-Viz 이그제큐터의 데이터 저장 여부
is_force_simulate	시뮬레이션을 강제로 Mech-Viz 를 실행할지 여부
is_force_real_run	강제로 Mech-Viz 를 실제 실행할지 여부
code_signal	Mech-Center 의 메인 인터페이스에서 Adapter 정보를 표시하는 신호 (오류 코드가 있는 정보)
msg_signal	Mech-Center 의 메인 인터페이스에서 Adapter 정보를 표시하는 신호 (오류 코드가 없는 정보)
i_code_signal	Mech-Center 의 메인 인터페이스에서 Mech-Interface 정보를 표시하는 신호 (오류 코드가 있는 정보)
i_msg_signal	Mech-Center 의 메인 인터페이스에서 Mech-Interface 정보를 표시하는 신호 (오류 코드가 없는 정보)
viz_finished_signal	Mech-Viz 실행 종료 신호 (정상 종료 또는 비정상 종료)
connect_robot_signal	로봇 신호 연결/분리
start_adapter_signal	Adapter 신호 실행
service_name_changed	Mech-Center 메인 인터페이스는 Mech-Viz 및 Mech-Vision 상태 신호를 표시합니다.
setting_infos	Mech-Center 의 구성 정보
service_name	등록된 서비스 이름

Adapter 클래스 함수는 다음 표에 나와 있습니다.

클래스 함수	설명
on_exec_status_changed()	Mech-Viz 및 Mech-Vision 에서 보낸 상태 정보 수신
register_self_service()	Adapter 서비스 등록
vision_project_dirs(self):	Mech-Vision 프로젝트 폴더 경로 조회
vision_project_names()	모든 Mech-Vision 프로젝트 명칭 조회
vision_project_names_in_center()	Mech-Center 에서 모든 Mech-Vision 프로젝트 명칭 조회
is_viz_registered()	Mech-Viz 가 등록되었는지 확인
is_viz_in_running()	Mech-Viz 가 실행 중인지 확인
is_vision_started()	Mech-Vision 프로젝트 등록 여부 확인
find_services()	서비스 찾기
before_start_viz()	Mech-Viz 가 시작되기 전에 호출할 함수
after_start_viz()	Mech-Viz 가 시작된 후 호출할 함수
viz_not_registerd()	Mech-Viz 를 시작한 후 Mech-Viz 가 등록되지 않은 것으로 확인되면 이 함수가 호
viz_is_running()	이 함수는 Mech-Viz 를 시작한 후 Mech-Viz 가 실행 중인 것으로 확인되면 호출됨
viz_run_error()	Mech-Viz 시작 후 Mech-Viz 운행 중 에러 발생 시 호출되는 함수
viz_run_finished()	Mech-Viz 실행이 완료되면 호출되는 함수
viz_plan_failed()	Mech-Viz 계획이 실패할 때 호출되는 함수
viz_no_targets()	Mech-Viz 계획에 이동 포인트가 없을 때 호출되는 함수
viz_unreachable_targets()	Mech-Viz 계획에 도달할 수 없는 포인트가 있을 때 호출되는 함수
viz_collision_checked()	Mech-Viz 계획이 충돌을 감지할 때 호출되는 함수
parse_viz_reply()	Mech-Viz 의 응답 구문 분석
wait_viz_result()	Mech-Viz 의 회신을 기다림
start_viz()	Mech-Viz 실행
stop_viz()	Mech-Viz 중지
pause_viz()	Mech-Viz 정지
find_vision_pose()	Mech-Vision 프로젝트가 사진을 캡처하도록 트리거하기
async_call_vision_run()	Mech-Vision 프로젝트가 사진을 캡처하도록 비동기식으로 트리거하기
async_get_vision_callback()	Mech-Vision 에서 비동기식으로 결과 수신
deal_vision_result()	Mech-Vision 의 결과 처리
set_step_property()	Mech-Vision 에서 스텝 파라미터 설정
read_step_property()	Mech-Vision 에서 스텝 파라미터 읽어내기
select_parameter_group()	Mech-Vision 프로젝트에서 레시피 템플릿 선택
set_task_property()	Mech-Viz 에서 태스크 파라미터 설정
read_task_property()	Mech-Viz 에서 태스크 파라미터 읽어내기
get_digital_in()	DI 획득
set_digital_out()	DO 설정
before_start_adapter()	이 함수는 Adapter 를 시작하기 전에 호출됩니다.
start()	Adapter 시작
close()	Adapter 닫기
handle_command()	수신된 외부 명령 처리

TcpServerAdapter 클래스

TcpServerAdapter 클래스는 Adapter 를 상속받아 아래와 같이 TcpServer 의 기능을 캡슐화합니다.

```
class TcpServerAdapter(Adapter):
    def __init__(self, host_address, server=TcpServer):
        super(TcpServerAdapter, self).__init__()
        self.init_server(host_address, server)

    def init_server(self, host_address, server=TcpServer):
        self._server = server(host_address)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    def set_recv_size(self, size):
        self._server.set_recv_size(size)

    def send(self, msg, is_logging=True):
        return self._server.send(msg, is_logging)

    def recv(self):
        return self._server.recv()

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            try:
                self._server.before_recv()
                cmds = self._server.recv()
                logging.info("Received raw data from client:{}".format(cmds))
                if not cmds:
                    logging.warning("Adapter client is disconnected!")
                    self.code_signal.emit(logging.WARNING, CENTER_CLIENT_DISCONNECTED)
                    self._server.close_client()
                    self.accept()
                    continue
                self._server.after_recv()
            except socket.error:
                logging.warning("Adapter client is closed!")
                self.code_signal.emit(logging.WARNING, CENTER_CLIENT_DISCONNECTED)
                self._server.close_client()
                self.accept()
            except Exception as e:
                logging.exception("Exception occurred when receiving data from client: {}.".
↪format(e))
            else:
                try:
                    self.handle_command(cmds)
                    self._server.after_handle()
                except Exception as e:
                    self.msg_signal.emit(logging.ERROR, _translate("messages", "Handle command␣
↪exception: {}".format(e)))
                    logging.exception("Adapter exception in handle_command(): {}".format(e))

    def close(self):
        super().close()
        self._server.close()

    def before_start_adapter(self):
        super().before_start_adapter()
        self.accept()

    def accept(self):
        if self.is_stop_adapter:
            return
        self.code_signal.emit(logging.INFO, CENTER_WAIT_FOR_CLIENT)
        self._server.accept()
        if self._server.is_connected():
            self.code_signal.emit(logging.INFO, CENTER_CLIENT_CONNECTED)
            self.msg_signal.emit(logging.INFO, _translate("messages", "Client address is") + "
↪{}".format(self._server.remote_socket()[1]))
```

TcpClientAdapter 클래스

TcpClientAdapter 클래스는 Adapter 를 상속받아 아래와 같이 TcpClient 의 기능을 캡슐화합니다.

```
class TcpClientAdapter(Adapter):

    def __init__(self, host_address):
        super().__init__()
        self.init_client(host_address)

    def init_client(self, host_address, client=TcpClient):
        self._client = client(host_address)

    def set_bind_port(self, is_bind=True):
        self._client.is_bind_port = is_bind

    def set_recv_size(self, size):
        self._client.set_recv_size(size)

    def send(self, msg, is_logging=True):
        self._client.send(msg, is_logging)

    def recv(self):
        return self._client.recv()

    def start(self):
        self.reconnect_server(False)
        while not self.is_stop_adapter:
            try:
                self._client.before_recv()
                cmds = self._client.recv()
                if not cmds:
                    self.reconnect_server()
                    continue
                logging.info("Received command from server:{}".format(cmds))
                self._client.after_recv()
            except socket.timeout:
                logging.warning("Socket timeout")
                self._client.after_timeout()
            except socket.error:
                sleep(5)
                self.reconnect_server()
            except Exception as e:
                logging.exception("Exception occurred when receiving from server: {}".
↪format(e))
            else:
                try:
                    self.handle_command(cmds)
                except Exception as e:
                    self.msg_signal.emit(logging.ERROR, _translate("messages", "Handle command␣
↪exception: {}".format(e)))
                    logging.exception("Adapter exception in handle_command(): {}".format(e))

    def close(self):
        super().close()
        self._client.close()

    def reconnect_server(self, is_reconnect=True):
        self._client.reconnect_server()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        if self.is_stop_adapter:
            return
        if self._client.is_connected():
            self.code_signal.emit(logging.INFO, CENTER_CONNECT_TO_SERVER)
        else:
            self.code_signal.emit(logging.WARNING, CENTER_SERVER_DISCONNECTED)
```

TcpMultiplexingServerAdapter 클래스

TcpMultiplexingServerAdapter 클래스는 Adapter 를 상속받아 아래와 같이 다중 클라이언트 연결에 주로 사용됩니다.

```
class TcpMultiThreadingServerAdapter(Adapter):
    def __init__(self, address):
        super().__init__()
        self._servers = {}
        self.add_server(address)
        self.sockets = {}
        self.clients_ip = {}
        self.thread_pool = ThreadPoolExecutor(max_workers=4, thread_name_prefix="tcp_multi_
↪server_thread")
        self.thread_id_socket_dict = {}
        self.set_recv_size()

    def set_recv_size(self, size=1024):
        self.recv_size = size

    def _find_client_ip(self, sock):
        for k, v in self.sockets.items():
            if v == sock:
                return k

    def _find_server(self, sock):
        for k, v in self._servers.items():
            if v == sock:
                return k

    def add_server(self, host_address):
        server = TcpServer(host_address)
        server.bind_and_listen()
        self._servers[server] = server.local_socket()

    def set_clients_ip(self, clients_ip):
        """
            Must be called before start().
            `clients_ip` is a dict(key is client ip, value is client description).
        """
        self.clients_ip = clients_ip

    def add_connection(self, ip_port, sock):
        self.sockets[ip_port] = sock
        logging.info("Add {}, connections: {}".format(ip_port, self.sockets))
        self.msg_signal.emit(logging.INFO, _translate("messages", "The client {} gets online.
↪").format(ip_port))

    def del_connection(self, ip):
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        logging.info("Del {}, connections: {}".format(ip, self.sockets))
        if self.client_connection(ip):
            self.client_connection(ip).close()
            self.sockets.pop(ip)
        self.msg_signal.emit(logging.WARNING, _translate("messages", "The client {} gets␣
↪offline.").format(ip))

    def client_connection(self, client_ip):
        return self.sockets.get(client_ip)

    def check_read_events(self, rs):
        for s in rs:
            if s in self._servers.values():  # recv connection
                server = self._find_server(s)
                if self.is_stop_adapter:
                    return
                server.accept()
                client_socket, client_addr = server.remote_socket()
                ip_port = "{}:{}".format(str(client_addr[0]), str(client_addr[1]))
                self.add_connection(ip_port, client_socket)
            elif s in self.sockets.values():  # recv data
                client_ip = self._find_client_ip(s)
                if not client_ip:
                    continue
                msg = self.recv_by_s(s)
                if not msg:
                    self.del_connection(client_ip)
                    return
                try:
                    future = self.thread_pool.submit(self.handle_command_thread, s, msg)
                except Exception as e:
                    logging.exception("Adapter exception in handle_command(): {}".format(e))

    def handle_command_thread(self, s, msg):
        thread_id = threading.get_ident()
        self.thread_id_socket_dict[thread_id] = s
        self.handle_command(msg)
        # del self.thread_id_socket_dict[thread_id]

    def send(self, msg, is_logging=True):
        thread_id = threading.get_ident()
        sock = self.thread_id_socket_dict.get(thread_id)
        len_total = len(msg)
        while msg:
            if sock:
                len_sent = sock.send(msg)
            else:
                for v in self.sockets.values():
                    try:
                        len_sent = v.send(msg)
                    except Exception as e:
                        logging.warning(e)
            if not len_sent:
                logging.warning("Connection lost, close the client connection.")
                return len_sent
            if is_logging:
                logging.info("Server send: {}, len_sent: {}".format(msg, len_sent))
            msg = msg[len_sent:]
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        return len_total

    def recv(self):
        thread_id = threading.get_ident()
        sock = self.thread_id_socket_dict.get(thread_id)
        return self.recv_by_s(sock)

    def recv_by_s(self, sock):
        msg = b""
        try:
            msg = sock.recv(self.recv_size)
        except socket.error:
            logging.error("The client is closed!")
        if msg:
            logging.info("Received message: {}".format(msg))
        return msg

    def check_task(self):
        """
            Interface.
        """

    def close(self):
        super().close()
        for server in self._servers.keys():
            server.close()
        for client_ip in self.sockets.keys():
            try:
                self.client_connection(client_ip).close()
                logging.info("Close socket :{}".format(client_ip))
            except Exception as e:
                logging.warning("Close socket error:{}, exception:{}".format(client_ip, e))
        self.sockets = {}

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            avalible_sockets = list(self.sockets.values()) + list(self._servers.values())
            rs, _, _ = select(avalible_sockets, [], [], 0.1)
            self.check_read_events(rs)
            try:
                self.check_task()
            except Exception as e:
                self.msg_signal.emit(logging.ERROR,
                                     _translate("messages", "Handle command exception: {}".
↪format(e)))
                logging.exception("Exception when check task:{}".format(e))
                sleep(5)
```

IOAdapter 클래스

IOAdapter 클래스는 Adapter 에서 상속되며 아래와 같이 주기적으로 DI 를 얻는 작업을 캡슐화합니다.

```
class IOAdapter(Adapter):
    robot_name = None
    check_rate = 0.5

    def __init__(self, host_address):
        super().__init__()
        self.last_gi = 0

    def get_digital_in(self, timeout=None):
        return super().get_digital_in(self.robot_name, timeout)

    def set_digital_out(self, port, value, timeout=None):
        super().set_digital_out(self.robot_name, port, value, timeout)

    def _check_gi(self):
        gi_js = self.get_digital_in()
        gi = int(json.loads(gi_js.decode())["value"])
        if self.last_gi != gi:
            self.last_gi = gi
            logging.info("Check GI signal status: {}".format(gi))
        self.handle_gi(gi)

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            try:
                self._check_gi()
            except Exception as e:
                logging.exception(e)
                self.check_gi_failed()
            sleep(self.check_rate)

    def handle_gi(self, gi):
        """
            Interface.
        """

    def check_gi_failed(self):
        """
            Interface.
        """
```

AdapterWidget 클래스

AdapterWidget 클래스는 Adapter 사용자 인터페이스의 상위 클래스를 정의하여 모든 사용자 인터페이스 정의의 기능은 반드시 아래와 같이 상속되어야 합니다.

```
class AdapterWidget(QWidget):

    def set_adapter(self, adapter):
        self.adapter = adapter
        self.after_set_adapter()
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    def after_set_adapter(self):
        """
            Interface.
        """

    def close(self):
        super().close()
        """
            Interface.
        """
```

Service

서비스 관련 클래스의 소스 파일은 Mech-Center 소프트웨어 설치 경로 아래의 `/src/interface/services.py` 파일에 있습니다.

NotifyService 클래스

NotifyService 클래스는 아래와 같습니다.

```
class NotifyService(JsonService):
    service_type = "notify"
    service_name = "adapter"

    def handle_message(self, msg):
        """
            Interface.
        """

    def notify(self, request, _):
        msg = request["notify_message"]
        logging.info("notify message:{}".format(msg))
        return self.handle_message(msg)
```

기본 서비스 이름은 adapter 이며 프로젝트에서 여러 알림 서비스가 필요한 경우 하위 클래스에서 service_name 을 다시 작성하여 다른 서비스를 구별할 수 있습니다. 클래스 함수 설명은 다음 표와 같습니다.

클래스 함수	설명
handle_message()	인터페이스 함수, 서브클래스는 이 함수를 재정의하고 이 함수에서 논리를 구현할 수 있습니다.
notify()	메시지 구문 분석을 제공하며 일반적으로 하위 클래스를 다시 작성할 필요가 없습니다.

VisionResultSelectedAtService 클래스

VisionResultSelectedAtService 클래스는 아래와 같습니다.

```
class VisionResultSelectedAtService(JsonService):
    service_type = "vision_watcher"
    service_name = "vision_watcher_adapter"

    def __init__(self):
        self.poses = None

    def poses_found(self, result):
        """
            Interface.
        """

    def posesFound(self, request, _):
        logging.info("{} result:{}".format(jk.mech_vision, request))
        self.poses_found(request)

    def poses_planned(self, result):
        """
            Interface.
        """

    def posesPlanned(self, request, _):
        logging.info("Plan result:{}".format(request))
        self.poses_planned(request)

    def multiPickCombination(self, request, _):
        logging.info("multiPickCombination:{}".format(request))
```

기본 서비스 유형은 vision_watcher 이며 다른 유형으로 변경할 수 없으며 기본 이름은 vision_watcher_adapter 입니다. 프로젝트에서 여러 vision_watcher 서비스가 필요한 경우 서브 클래스에서 service_name 을 다시 작성하여 다른 서비스를 구분할 수 있습니다. 클래스 함수 설명은 다음 표와 같습니다.

클래스 함수	설명
poses_found()	인터페이스 함수, 하위 클래스는 이 함수를 재정의하고 이 함수에서 논리를 구현할 수 있으며 파라미터는 Mech-Vision 의 인식 결과입니다.
posesFound()	Mech-Vision 에서 식별한 메시지를 구문 분석하며 일반적으로 하위 클래스를 다시 작성할 필요가 없습니다.
poses_planned()	인터페이스 함수, 파라미터는 Mech-Viz 계획에 의해 선택된 비전 포인트입니다.
posesPlanned()	Mech-Viz 계획 메시지 구문 분석 제공

RobotService 클래스

RobotService 클래스는 아래와 같습니다.

```
class RobotService(JsonService):
    service_type = "robot"
    service_name = "robot"
    jps = [0, 0, 0, 0, 0, 0]
    pose = [0, 0, 0, 1, 0, 0, 0]

    def getJ(self, *_):
        return {"joint_positions": self.jps}

    def setJ(self, jps):
        logging.info("setJ:{}".format(jps))
        self.jps = jps

    def getL(self, *_):
        return {"tcp_pose": self.pose}

    def getFL(self, *_):
        return {"flange_pose": self.pose}

    def setL(self, pose):
        logging.info("setL:{}".format(pose))
        self.pose = pose

    def moveXs(self, params, _):
        pass

    def stop(self, *_):
        pass

    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
```

기본 서비스 유형은 robot 이며 다른 유형으로 변경할 수 없으며 기본 이름은 robot 이며 하위 클래스는 해당 로봇 이름으로 변경해야 합니다. jps 또는 pose 값은 하위 클래스에서 설정해야 합니다. 기능은 Mech-Viz 작동 중 포즈를 수정하는 것입니다. 여기에서 이 포즈가 전체 경로에서 시나리오와 충돌하지 않아야 한다는 점에 유의해야 합니다. 클래스 함수 설명은 다음 표와 같습니다.

클 래 스 함수	설명
getJ()	Mech-Viz/Mech-Vision 의 관절 각도 반환
setJ()	외부에서 관절 각도를 설정합니다. 단위는 라디안입니다.
getL()	Mech-Viz/Mech-Vision 에 TCP 포즈 반환
getFL()	Mech-Viz/Mech-Vision 에 플랜지 포즈 반환
setL()	외부적으로 플랜지 포즈 (사원수 형식) 를 설정합니다. 단위는 미터입니다.
moveXs()	Mech-Viz 가 경로를 계획한 후 이 함수를 호출합니다. 파라미터에는 이동 포인트의 속성이 포함되어 있습니다. 주의: Mech-Viz 프로젝트에 DI 체크, 분기 등 태스크가 있으면 사전 계획된 경로를 방해합니다. 이때 Mech-Viz 는 함수를 여러 번 호출합니다.
stop()	로봇 중지, 일반적으로 사용하지 않음
setTcp()	TCP 설정, 일반적으로 사용하지 않음
setDigitalOut()	DO 설정, 일반적으로 사용하지 않음
getDigitalIn()	DI 획득, 일반적으로 사용하지 않음
switchPauseContinue()	로봇 일시 중지/계속 실행하기, 일반적으로 사용하지 않음

OuterMoveService 클래스

OuterMoveService 클래스는 아래와 같습니다.

```
class OuterMoveService(JsonService):
    service_type = "outer_move"
    service_name = "outer_move"
    move_target_type = TCP_POSE
    velocity = 0.25
    acceleration = 0.25
    blend_radius = 0.05
    motion_type = MOVEJ
    is_tcp_pose = False
    pick_or_place = 0

    def __init__(self):
        self.targets = []

    def gather_targets(self, di, jps, flange_pose):
        """
            Interface.

            Please add targets to `self.targets` here if needed.
        """

    def add_target(self, move_target_type, target):
        self.targets.append({"move_target_type": move_target_type, "target": target})

    def getMoveTargets(self, params, *_):
        """
        @return: targets(move_target_type  0:jps, 1:tcp_pose, 2:obj_pose)
                 velocity(default 0.25)
                 acceleration(default 0.25)
                 blend_radius(default 0.05)
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
                motion_type(default moveJ  'J':moveJ, 'L':moveL)
                is_tcp_pose(default False)
        """
        di = params["di"]
        jps = params["joint_positions"]
        flange_pose = params["pose"]
        logging.info("getMoveTargets: di={}, jps={}, flange_pose={}".format(di, jps, flange_
↪pose))

        self.gather_targets(di, jps, flange_pose)
        targets = self.targets[:]
        self.targets.clear()
        logging.info("Targets: {}".format(targets))
        return {"targets": targets, "velocity": self.velocity, "acceleration": self.
↪acceleration, "blend_radius": self.blend_radius,
                "motion_type": self.motion_type, "is_tcp_pose": self.is_tcp_pose, "pick_or_
↪place": self.pick_or_place}
```

기본 서비스 유형 및 명칭은 outer_move 이며, 프로젝트에 여러 outer_move 서비스가 필요한 경우 하위 클래스에서 service_name 을 다시 작성하여 다른 서비스를 구분할 수 있습니다. 클래스 함수 설명은 다음 표와 같습니다.

클래스 함수	설명
move_target_type()	이동 포인트 유형 0:jps, 1:tcp_pose, 2:obj_pose
velocity()	이동 포인트 속도, 기본값은 0.25 입니다.
acceleration()	이동 포인트 가속도, 기본값은 0.25 입니다.
blend_radius()	이동 포인트의 회전 반경, 기본값은 0.05m 입니다.
motion_type()	이동 포인트 운동 유형 'J':moveJ, 'L':moveL
is_tcp_pose()	이동 포인트가 TCP 인지 여부
gather_targets()	인터페이스 함수는 모든 이동 포인트를 수집하는데 이때 로봇의 관절 각도, 플랜지 포즈, DI 값을 파라미터로 하며, 서브 클래스는 필요에 따라 판단 및 수정할 수 있습니다.
add_target()	단일 이동 포인트를 추가합니다. 이 함수는 하위 클래스에서 호출하여 이동을 추가할 수 있습니다.
getMoveTargets()	이 함수는 Mech-Viz 가 외부 이동을 수행할 때 호출되며, 이때 파라미터에 로봇의 관절 각도, 플랜지 포즈 및 DI 값이 포함됩니다.

서비스 등록

위 네 가지 클래스에 해당하는 서비스는 회원가입 후 이용이 가능하며, 서비스 등록 함수는 다음과 같습니다.

```
def register_service(hub_caller, service, other_info=None):
    server, port = start_server(service)
    if service.service_type == "robot":
        other_info["from_adapter"] = True
        other_info["simulate"] = False
    hub_caller.register_service(service.service_type, service.service_name, port, other_info)
    return server, port
```

Adapter util 패키지

Adapter util 패키지는 Mech-Center 소프트웨어 설치 경로 아래의 `/src/util` 폴더에 있으며, 많은 모듈을 포함하고 몇 가지 공통 함수를 제공합니다. 프로그래밍 과정에서 먼저 util 패키지에 어느 함수의 기능이 구현되었는지 확인합니다. 기능이 구현되어 있으면 직접 사용할 수 있고, 구현하지 않고 보다 일반적이면 작은 함수로 추상화하여 util 패키지에 추가할 수 있습니다.

다음은 각 모듈에 대한 간략한 소개입니다.

database 모듈

database 모듈은 데이터베이스에 대한 작업을 제공합니다. Mech-Center 는 실행 시 기본적으로 mech-mind.db 데이터베이스 파일을 생성하며, 이는 실행 중인 로그를 저장하는 데 사용됩니다. database 모듈은 하나 또는 모든 기록을 조회하기 위해 SQL 문을 실행하는 기능을 제공합니다.

json_keys 모듈

json_keys 모듈은 Mech-Center 에서 사용되는 json 키/값 문자열을 저장하며, 다른 모듈에서 직접 가져와서 사용할 수 있습니다.

message_box 모듈

message_box 모듈은 팝업 프롬프트의 기능을 제공하며 팝업 프롬프트의 종류에는 정보 (information), 경고 (warning), 위험 (critical) 이 있습니다.

timestamp 모듈

timestamp 모듈은 현재 타임스탬프를 반환하는 기능을 제공합니다.

transforms 모듈

transforms 모듈은 오일러 각에서 사원수로/사원수에서 오일러 각으로 전환, 포즈 곱셈, 물체 포즈에서 TCP 포즈로/TCP 포즈에서 물체 포즈로 전환, 물체 회전 계산 등과 같은 기능을 제공합니다. transforms3d 도 오일러 각에서 사원수로, 사원수에서 오일러 각으로 전환하는 기능을 제공하지만 실제 사용에서는 transforms3d 로 변환된 값이 잘못된 경우가 있습니다. 실제 계산에서는 transforms3d 라이브러리를 먼저 사용할 수 있으며, 결과가 틀리면 transforms 모듈에서 제공하는 사용자 정의 변환 함수를 사용할 수 있습니다.

util_file 모듈

util_file 모듈은 파일 읽기 및 쓰기 기능을 제공합니다. 일반적으로 사용되는 json 파일 읽기 및 쓰기를 포함합니다.

timer 모듈

timer 모듈은 편리한 타이머 클래스를 제공합니다. 타이밍 기능이 필요한 경우 Timer 대상을 생성하고 콜백 함수를 전달하고 start() 를 호출할 수 있습니다. 사용 후 Timer 대상은 소멸될 필요가 없으며 프로그램이 종료될 때 자동으로 소멸됩니다.

pose 모듈

pose 모듈은 Mech-Viz 의 포즈 표현과 동일한 클래스를 제공합니다. 예를 들면 평행이동 (미터 단위) 및 회전 (사원수 형식) 을 포함하며 역계산 및 곱셈 연산을 할 수 있고 list 에서 전환 또는 list 로 전환을 할 수 있습니다. 또한 pose 모듈은 pose 의 단위 전환 함수를 제공합니다. 예를 들면 밀리미터에서 미터로, 미터에서 밀리미터, 라디안에서 도, 도에서 라디안, 사원수에서 오일러 각으로, 오일러 각에서 수원수 등 전환입니다.

마지막으로 요구 사항에 따라 추상적인 상위 클래스 인터페이스를 구현하여 내부 (Mech-Vision 및 Mech-Viz) 와 외부 (외부 장치) 간의 통신을 실현합니다.

인터페이스를 획득하기

현재 Mech-Viz 프로젝트에서 사용하는 태스크를 획득하기

현재 Mech-Viz 프로젝트에서 사용하는 태스크를 가져오는 함수는 다음과 같습니다.

```
def get_viz_task_names(self, msg={}, timeout=None):
    result = self.call_viz("getAllTaskNames", msg, timeout)
    logging.info("Property result: {}".format(json.loads(result)))
    return result
```

get_viz_task_names() 를 호출한 후 json 형식의 문자열을 반환해 획득한 모든 태스크를 나타냅니다.

Mech-Viz 또는 Mech-Vision 프로젝트에서 파라미터를 획득하기

Mech-Viz 또는 Mech-Vision 프로젝트에서 파라미터를 획득하는 함수는 다음과 같습니다.

```
def get_property_info(self, msg={}, get_viz=True, timeout=None):
    result = (self.call_viz if get_viz else self.call_vision)("getPropertyInfo", msg, timeout)
    logging.info("{0} property result: {1}".format("Viz" if get_viz else "Vision", json.
↪loads(result)))
    return result
```

호출할 때 msg 파라미터에"type" 을 지정하지 않으면 모든 파라미터를 가져오는 것을 의미합니다. 만약 지정된 경우 해당 파라미터만 가져옵니다. 예를 들어 get_property_info(msg={"type": "move"}) 를 호출한 후 json 형식의 문자열을 반환해 가져온 이동 태스크 파라미터를 나타냅니다.

Mech-Vision 인터페이스

이 부분에서는 Mech-Vision 과 관련된 인터페이스 사용에 대해 소개합니다. 구체적으로 다음 인터페이스를 포함합니다.

비전 목표점을 획득하기

아래와 같이 adapter.py 의 find_vision_pose() 함수를 통해 Mech-Vision 비전 결과를 가져옵니다.

```
def find_vision_pose(self, project_name=None, timeout=default_vision_timeout):
    vision_result = self.call_vision("findPoses", project_name=project_name, timeout=timeout)
    logging.info("Find vision result: {}".format(vision_result))
    return vision_result
```

Mech-Vision 의 포즈는 일반적으로 물체 포즈 (obj_pose) 와 사원수 형식으로 출력됩니다 (도구 포즈로도 출력될 수 있으며 Mech-Vision 프로젝트에서 변환해야 함).

예시

Adapter 는 함수를 생성해 그것을 TCP 포즈 (tcp_pose) 로 전환해야 하며 때로는 사원수를 오일러 각도, 라디안을 각도 등으로 전환해야 합니다 (전환 여부는 로봇 측과 일치해야 함), 마지막으로 패키지 하여 로봇 측으로 보냅니다. 또한 Adapter 는 다음과 같이 Mech-Vision 이 출력하는 비전 포인트를 확인할 수도 있습니다.

```
def check_vision_result(self, vision_result):
    if vision_result["noCloudInRoi"]:                    # Determine whether it is an empty␣
↪bin
    logging.info("Layer has no objects")
    self.send(pack('>2B6i', CODE_NO_CLOUD, vision_num, *EMPTY_PLACEHOLDER))
    return
    poses = vision_result.get("poses", [])               # Get pose
    if len(poses) == 0:                                 # Determine whether there is a vision␣
↪point
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        logging.warning("No pose from vision")
        self.send(pack('>2B6i', CODE_NO_POSE, vision_num, *EMPTY_PLACEHOLDER))
        return
    self.send(pack_pose(poses[0], vision_num))       # Send after format conversion
```

이 중 find_vision_pose() 에서 vision_result 를 가져오며, 호출문은 다음과 같습니다.

```
self.check_vision_result(json.loads(self.find_vision_pose().decode()))
```

vision_result 를 함수에 입력한 후 ROI 와 관련된 프로젝트인 경우 먼저 빈 상자인지 확인한 후 포즈를 획득하고, 포즈가 정상이면 포즈를 전환 함수 (pack_pose()) 에 보낸 다음에 전송합니다.

스텝 파라미터를 설정하기

Mech-Vision 스텝의 파라미터를 동적으로 설정할 때 일반적으로 adapter.py 에서 set_step_property() 만 호출하면 됩니다.

```
def set_step_property(self, msg, project_name=None, timeout=None):
    return self.call_vision("setStepProperties", msg, project_name, timeout)
```

그 중 msg 는 구체적인 스텝 명칭과 구성해야 하는 파라미터를 결정합니다.

예시

Mech-Vision 매칭 모델이 다양한 작업물 유형에 따라 동적으로 설정되어야 하는 경우 다음 함수를 생성하여 msg 를 설정할 수 있습니다.

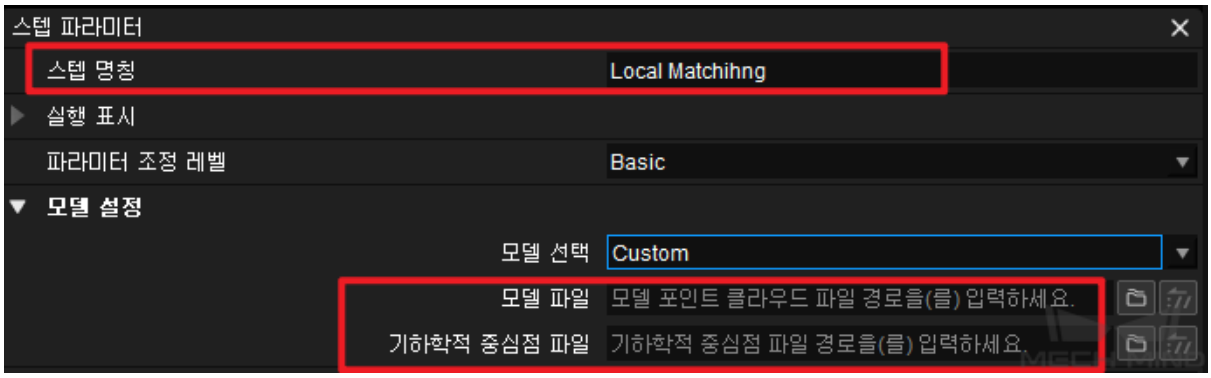
```
def _step_matching_model_cell(step_name, model_type):
    msg = {"name": step_name,
           "values":
          {"modelFile": model_type["ply"],
        "pickPointFilePath": model_type["json"]}}
return msg
```

이 중 step_name 은 설정해야 하는 Mech-Vision 스텝의 이름이고 model_type 은 해당 작업물 유형에 해당하는 파일 경로이며"ply" 와"json" 을 통해.ply 와.json 로 끝나는 모델 파일 경로를 얻을 수 있으며 Mech-Vision 의 해당 스텝의 파라미터를 각각 채울 수 있습니다.

step_name 이"Local Matching" 인 경우 호출 문은 다음과 같습니다.

```
msg = _step_matching_model_cell("Local Matching", model_type)
self.set_step_property(msg)
```

효과는 아래 그림과 같습니다.



스텝 파라미터를 읽어내기

Mech-Vision 에서 어느 스텝의 파라미터를 가져오려면 adapter.py 중의 read_step_property() 를 호출하면 됩니다.

```
def read_step_property(self, msg):
    result = self.call_vision("readStepProperties", msg)
    logging.info("Property result: {}".format(result))
    return result
```

이 중 msg 는 스텝명칭과 획득할 파라미터 값을 결정하고 함수를 생성하여 msg 를 다시 작성한면 됩니다.

예시

카메라 IP 를 얻으려면 샘플 코드가 아래와 같습니다.

```
def read_camera_property(self):
    msg = {"type": "Camera",
           "properties": ["MechEye"]}
    property_results = json.loads(self.read_step_property(msg).decode())
    camera_ip = property_results["MechEye"]["NetCamIp"]
```

Mech-Vision 프로젝트에 카메라가 하나만 있는 경우 유형 (type) 에 따라 스텝을 찾을 수 있습니다. 만약 Mech-Vision 에 동일한 스텝이 여러 개 있고 특정 스텝 파라미터만 가져오거나 설정하려는 경우 명칭 (name) 으로 스텝을 찾을 수 있습니다. 다음과 같이 read_step_property() 함수를 호출하고 json 형식으로 변환하여 카메라의 모든 파라미터 값 (json 형식) 을 가져옵니다.

```
Property result:
{
  "MechEye": {
    "NetCamIp": "127.0.0.1",
    "TimeOut": "10",
    "configGroup": "",
  }
}
```

이 예시에서 카메라의 IP 주소 (127.0.0.1) 는 특정 파라미터 필드 (”MechEye” 및”NetCamIp”) 에 따라 최종적으로 획득됩니다.

파라미터 레시피를 전환하기

Mech-Vision 을 사용할 때 일부 프로젝트의 프로세스는 동일하지만 일부 파라미터가 다른 경우 구성 파라미터 레시피를 사용하여 다른 프로젝트에 대한 해당 파라미터를 전환하는 기능을 실현할 수 있습니다. 즉 adapter.py 에서 select_parameter_group() 을 호출하는 것입니다.

```
def select_parameter_group(self, project_name, group_index, timeout=None):
    msg = {"parameter_group_idx": group_index}
    result = self.call_vision("selectParameterGroup", msg, project_name, timeout)
    logging.info("selectParameterGroup result: {}".format(result))
    return result
```

이 중 project_name 파라미터는 Mech-Vision 의 프로젝트 명칭이고 group_index 파라미터는 레시피 번호입니다.

예시

프로젝트에서 레시피를 전환해야 하는 경우 다음 샘플 코드를 사용하여 select_parameter_group() 함수를 호출하고 예외를 처리합니다.

```
try:
   result = self.select_parameter_group(self.vision_project_name, model_code-1)
   if result:
      result = result.decode()
      if result.startswith("CV-E0401"):
            return -1
      elif result.startswith("CV-E0403"):
            return -1
      raise RuntimeError(result)
   except Exception as e:
      logging.exception('Exception when switch model: {}'.format(e))
      return -1
   return 0
```

그 중 self.vision_project_name 은 전송한 Mech-Vision 프로젝트 명칭이고 model_code-1 은 보내 온 레시피 번호입니다.

Mech-Viz 인터페이스

이 부분에는 Mech-Viz 와 관련된 인터페이스를 소개하고자 합니다. 구체적으로 다음과 같은 인터페이스을 포함합니다.

- *Mech-Viz* 를 시작하기
- *Mech-Viz* 를 정지하기
- *Mech-Viz* 를 중지하기/계속 실행하기
- 태스크의 파라미터를 설정하기
  - 이동
  - 순서대로 이동/ 배열대로 이동
  - 외부 이동
  - 팔레타이징
  - 분기
  - 카운터
- 태스크의 파라미터를 읽어내기
- *TCP* 를 설정하기
- 실행 전반 속도를 설정하기
- 포인트 클라우드의 충돌 파라미터를 설정하기
- *Mech-Viz* 반환값

Mech-Viz 를 시작하기

adapter.py 파일의 Adapter 클래스에서 Mech-Viz 를 시작하는 함수를 정의했기 때문에 코드에서 직접 start_viz() 를 호출하면 됩니다. 또한 Mech-Viz 를 시작하기 전/시작한 후의 작업을 자체 정의하기 위해 프로젝트의 기능에 근거하여 self.before_start_viz() 및 self.after_start_viz() 을 다시 작성할 수 있습니다.

함수 정의

```
def start_viz(self, in_new_thread=True, timeout=None):
    if not self.is_viz_registered():
        logging.error("{} has not registered in {}.".format(jk.mech_viz, jk.mech_center))
        self.code_signal.emit(ERROR, VIZ_NOT_REGISTERED)
        self.viz_finished_signal.emit(True)
        self.viz_not_registerd()
        return False
    if self.is_viz_in_running():
        logging.info("{} is already running.".format(jk.mech_viz))
        self.code_signal.emit(WARNING, VIZ_IS_RUNNING)
        self.viz_finished_signal.emit(False)
        self.viz_is_running()
        return False
    self._read_viz_settings()
    if not self.viz_project_dir:
        self.msg_signal.emit(ERROR, _translate("messages", "The project of {0} is not␣
↪registered. Please make sure Autoload Project is selected in {0}.").format(jk.mech_viz))
        self.viz_finished_signal.emit(True)
        return False
    msg = {"simulate": self.is_simulate, "project_dir": self.viz_project_dir}
    if self.is_keep_viz_state:
        msg["keep_exec_state"] = self.is_keep_viz_state
    if self.is_save_executor_data:
        msg["save_executor_data"] = self.is_save_executor_data
    self.before_start_viz()
    self.viz_finished_signal.emit(False)
    if in_new_thread:
        threading.Thread(target=self.wait_viz_result, args=(msg, timeout)).start()
    else:
        self.wait_viz_result(msg, timeout)
    self.after_start_viz()
    return True
```

Mech-Viz 를 시작하는 작업 이외의 다른 작업에 영향을 미치지 않도록 start_viz() 는 기본적으로 새 스레드에서 Mech-Viz 실행이 완료될 때까지 기다립니다.

다음으로 스텝의 파라미터를 동적으로 설정하는 것을 예시로 self.before_start_viz() 를 다시 작성하는 방법을 설명해 보겠습니다. 아래와 같습니다.

```
def before_start_viz(self):
     self.set_move_offset(x, y, z)
```

Mech-Viz 를 시작하기 전에 읽어낸 데이터에 따라 어떤 이동 포인트 x,y,z 방향에서의 옵셋을 구성합니다.

Mech-Viz 를 정지하기

adapter.py 파일의 Adapter 클래스에서 Mech-Viz 를 정지하는 함수를 정의했기 때문에 코드에서 직접 stop_viz() 를 호출하면 됩니다.

함수 정의

```
def stop_viz(self, timeout=None):
    if not self.is_viz_registered():
        self.code_signal.emit(WARNING, VIZ_NOT_REGISTERED)
        return False
    self.call_viz("stop", timeout=timeout)
    self.code_signal.emit(INFO, VIZ_STOP_OK)
    return True
```

Mech-Viz 를 중지하기/계속 실행하기

adapter.py 파일의 Adapter 클래스에서 Mech-Viz 를 중지하거나 계속 실행하는 함수를 정의했고 이 함수의 기능은 Mech-Viz 소프트웨어의 [중지] 버튼과 같으며 시뮬레이션 과정에서만 사용될 수 있습니다.

함수 정의

```
def pause_viz(self, msg, timeout=None):
       if not self.is_viz_registered():
           self.code_signal.emit(WARNING, ADAPTER_CANCEL_PAUSE)
           return
       self.call_viz("switchPauseContinue", msg, timeout)
       self.code_signal.emit(INFO, ADAPTER_PAUSE_VIZ if msg.get(
           "to_pause") else ADAPTER_CONTINUE_VIZ)
```

태스크의 파라미터를 설정하기

Mech-Viz 에서 태스크의 파라미터를 동적으로 설정하려면 일반적으로 Adapter 클래스의 set_task_property() 를 호출하면 됩니다.

함수 정의

```
def set_task_property(self, msg, timeout=None):
    return self.call_viz("setTaskProperties", msg, timeout)
```

그 중에 msg 가 다른 태스크에 대해 다른 파라미터를 설정하는 것을 정합니다.

이동

Mech-Viz 가 실행될 때 이동 태스크의 X, Y, Z 의 옵셋 값을 조금 올려주거나 낮춰야 할 경우가 종종 있습니다. 이런 경우에 Mech-Viz 를 컨트롤하는 메인 프로그램에서 다음과 같은 함수를 작성할 수 있습니다.

예시

```
def set_move_offset(self, name, x_offset, y_offset, z_offset):
    msg = {"name": name,
           "values": {"xOffset": x_offset / UNIT_PER_METER,
                      "yOffset": y_offset / UNIT_PER_METER,
                      "zOffset": z_offset / UNIT_PER_METER}}
    self.set_task_property(msg)
```

name: 모듈의 명칭. UNIT_PER_METER=1000. 일반적으로 x_offset, y_offset 및 z_offset 데이터의 단위는 mm 이고 Mech-Viz 의 데이터 단위는 m 이기 때문에 UNIT_PER_METER 를 통해 단위를 전환해야 합니다.

다음 방식으로 set_move_offset() 함수를 호출하면 Mech-Viz 에 있는 move_1 태스크의 X, Y, Z 의 옵셋 값이 변할 것입니다.

```
self.set_move_offset("move_1", 100, 200, 300)
```

순서대로 이동/ 배열대로 이동

일반적으로 순서대로 이동/ 배열대로 이동 태스크를 사용하기 전에 Mech-Viz 에서 미리 편집해야 하고 Adapter 가 논리에 따라 인덱스를 수정할 것입니다. 사용 방법은 이동과 팔레타이징 태스크와 같습니다.

외부 이동

계획과 컨트롤를 위해 여러 외부 대상 물체의 포즈를 Mech-Viz 로 보내야 하는 경우" 외부 이동" 을 통해 실현할 수 있습니다. 외부 이동 태스크에서는 JPs, TCP 및 물체 포즈를 설정할 수 있습니다. 사용 방법은 아래와 같습니다.

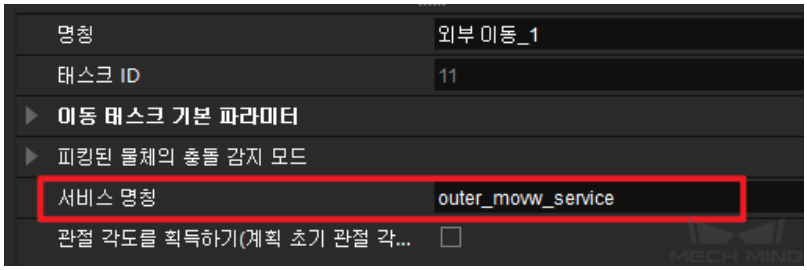
예시

```
class CustomOuterMoveService(OuterMoveService):
    def gather_targets(self, di, jps, flange_pose):
        self.add_target(self.move_target_type, [0.189430,-0.455540,0.529460,-0.079367,0.294292,
↪-0.952178,0.021236])
```

Mech-Viz 가 외부 이동 태스크를 실행할 때 getMoveTargets() 함수를 호출할 것입니다. 서비스 명칭을 통해 다른 외부 이동 태스크를 구분합니다.

```
def _register_service(self):
   self.outer_move_service = CustomOuterMoveService()
   self._outer_move_server, port = register_service(outer_move_service, port)
```

Mech-Viz 에서 외부 이동 태스크의 파라미터는 아래 그림과 같습니다.



주의: 외부 이동 태스크 전에 피킹 태스크가 있어야 합니다. 그렇지 않으면"- 물체를 잡고 있지 않은 경우에 물체 포즈가 무효합니다!" 라는 오류 메시지가 나옵니다.

팔레타이징

Mech-Viz 가 실행될 때 다른 팔레타이징 태스크에 따라 다른 파라미터를 설정해야 할 경우가 종종 있습니다. 이런 경우에 팔레타이징 태스크의 명칭을 통해 수정할 태스크를 찾을 수 있습니다. 프로젝트 편집 구역에 있는 태스크를 선택하고 Mech-Viz 파라미터 편집 구역에 나온 모든 파라미터를 수정할 수 있습니다.

예시

예를 들어 [자체 정의한 파렛트 패턴] 태스크에 대해 일반적으로 파라미터 **현재 인덱스** 및 **파일 명칭** 을 수정해야 하며 (**동적 로딩** 을 선택해야 **폴더 경로** 를 볼 수 있음) 메인 프로그램에서 다음과 같은 함수를 정의할 수 있습니다.

```
def set_stack_pallet(self, name, curIndex, fileName):
    msg = {
        "name": name,
        "values": {
            "curIndex": curIndex,
            "fileName": fileName,
        }
    }
    self.set_task_property(msg)
```

그 중에"curIndex" 는 파라미터 **현재 인덱스** 와 대응하며"fileName" 은 파라미터 **파일 명칭** 과 대응합니다. "curIndex" 및"fileName" 의 파라미터 명칭은 모두 Mech-Viz 에서 정의됩니다.

다음 방법을 통해 set_stack_pallet() 함수를 호출하십시오.

```
self.set_stack_pallet("common_pallet_1", 2, "re.json")
```

[미리 설정된 파렛트 패턴] 태스크에 대해 일반적으로 파라미터 **현재 인덱스**, **파렛트 패턴**, **상자 너비**, **상자 높이**, **층수**, **행수**, **열수** 등을 수정해야 하며 메인 프로그램에서 다음과 같은 함수를 정의할 수 있습니다.

```
def set_stack_pallet(self, name, curIndex, stack_type):
    pallet_info = self.box_data_info[stack_type]
    """
        pallet_info: Length(mm),Width(mm),Height(mm),pallet type,rows,columns,layers
    """
    msg = {
        "function": "setTaskProperties",
        "name": name,
        "values": {
            "curIndex": curIndex,
            "palletType": pallet_info[3],
            "cartonLength": pallet_info[0] / UNIT_PER_METER,
            "cartonWidth": pallet_info[1] / UNIT_PER_METER,
            "cartonHeight": pallet_info[2] / UNIT_PER_METER,
            "cylinderRows": pallet_info[4],
            "cylinderCols": pallet_info[5],
            "layerNum": pallet_info[6]
        }
    }
    self.set_task_property(msg)
```

설정해야 할 파라미터가 많기 때문에 보통 파라미터들을 하나의 excel 파일에 입력하고 excel 파일의 데이터를 읽어내 self.box_data_info 에 기록합니다. 나중에 stack_type 의 수치를 통해 찾아볼 수 있습니다. Mech-Viz 에서"curIndex", "palletType" 및"cartonLength" 등 명칭은 정해진 것입니다.

자체 정의한 파렛트 패턴 과 **미리 설정된 파렛트 패턴** 클래스의 msg 수치를 비교해 보면 둘의"values" 수치가 다릅니다. 어떤 태스크의 파라미터를 설정하려면"values" 에서 해당 파라미터의 명칭과 수치를 추가

하면 바로 설정할 수 있습니다. 다른 [팔레타이징] 태스크의 파라미터를 설정할 때 위 예시를 참조하십시오.

분기

Mech-Viz 가 분기 태스크를 수행할 때 외부 신호 (Adapter) 가 아웃포트를 지정하는 것을 기다립니다. 분기 태스크에 대해 다음과 같은 함수를 정의하여 분기를 컨트롤할 수 있습니다.

예시

```
def set_branch(self, name, area):
    time.sleep(1) # The delay of 1s here is to wait for the Mech-Viz executor to fully start
    try:
        info = {"out_port": area, "other_info": []}
        msg = {"name": name,
               "values": {"info": json.dumps(info)}}
        self.set_task_property(msg)
    except Exception as e:
        logging.exception(e)
```

name: 분기 태스크의 명칭. area: 아웃포트 번호이고 왼쪽에서 오른쪽으로 0 1 2 ...입니다. 예를 들어 이번에 해당 태스크가 가장 왼쪽 아웃포트를 통해 실행되면 area 의 수치는 0 입니다. Mech-Viz 를 시작하지 않고 바로 분기 태스크를 호출하면“이그제큐터가 없습니다!”라는 오류 메시지가 나옵니다.

카운터

카운터 태스크를 사용할 때 일반적으로 카운터 총수와 현재 카운터를 설정해야 합니다. 해당 태스크를 정의하는 코드가 다음과 같습니다.

예시

```
def set_counter_property(self, name, count, curCount):
    msg = {"name": name,
           "values": {"count": count, "currentCount": curCount}}
    self.set_task_property(msg)
```

이 함수를 호출하는 예시가 다음과 같습니다.

```
self.set_counter_property("counter_1", 5, self.success_stack_num)
```

self.success_stack_num 은 성공적으로 배치한 파렛트의 수량을 뜻합니다. 팔레타이징이 진행될 때 만약에 상자가 떨어지는 경우 수동으로 Mech-Viz 를 정지시킵니다. 이때 Mech-Viz 에 있는”counter_1” 카운터 태스크가”currentCount” 의 수치를 저장하지 않을 것입니다. Mech-Viz 를 다시 시작한 후 self.success_stack_num 를 통해 현재 카운터를 다시 설정할 수 있습니다.

태스크의 파라미터를 읽어내기

Mech-Viz 가 실행되는 동안 어떤 태스크의 파라미터를 읽어내려면 메인 프로그램에서 다음과 같은 함수를 정의할 수 있습니다.

예시

```
def read_move_pallet(self, name):
     msg = {"name": name,
            "properties": ["xOffset","yOffset","zOffset", ]}
     return read_task_property(msg)
```

그중에"name" 을 통해 읽어낼 태스크의 명칭을 찾습니다. "properties" 뒤에 있는 리스트의 수치는 수요에 따라 파라미터를 추가하거나 삭제할 수 있습니다. 예시에서"xOffset", "yOffset" 및"zOffset" 의 수치를 읽어내려고 하기 때문에 이 세 가지 수치를"properties" 에 입력하면 됩니다. 구체적인 방법은 아래와 같습니다.

```
self.read_move_pallet("move_3")
```

호출한 후 다음과 같은 결과를 얻었습니다.

```
{'zOffset': -0.23, 'xOffset': -0.12, 'yOffset': -0.15}
```

TCP 를 설정하기

TCP 를 설정하려면 Mech-Viz 엔드 이펙터 리스트에서 해당 인덱스를 수정하면 됩니다. Mech-Viz 엔드 이펙터 리스트의 인덱스는 위에서 아래로 0 1 2 3 .....입니다. 주의해야 할 것은 한계를 넘지 마세요. TCP 를 설정하는 함수가 아래와 같습니다.

예시

```
def set_tcp(self, index):
    msg = {"function": "setTcp", "index": index}
    self.call("executor", msg)
```

실행 전반 속도를 설정하기

Mech-Viz 가 실행될 때 로봇 작업 속도의 백분율을 동적으로 조절하려면 메인 프로그램에서 아래와 같은 함수를 작성하면 됩니다.

예시

```
def set_vel(self, vel_scale):
    msg = {"function": "setConfig",
           "velScale": vel_scale / 100, "accScale": vel_scale / 100}
    self.call("executor", msg)
```

속도를 80% 로 설정하면 아래와 같이 해당 함수를 호출하십시오.

```
self.set_vel(80)
```

주의: Mech-Viz 를 먼저 시작해야 이 함수를 호출할 수 있습니다. 즉 해당 모듈을 호출하는 조건은 분기 태스크와 일치합니다. 따라서 일반적으로 분기 태스크에서 set_vel() 함수를 호출하고 새 스레드에서 set_vel() 함수를 호출하지 마세요. 예시는 아래와 같습니다.

```
def set_branch(self, name, area):
    time.sleep(1)
    if self.box_data_info[int(self.pallet_info)][7] <= 10:
        self.set_vel(100)
    else:
        self.set_vel(80)
    try:
        info = {"out_port": area, "other_info": []}
        msg = {"function": "setTaskProperties",
               "name": name,
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
                "values": {"info": json.dumps(info)}}
        self.set_task("executor", msg)
    except Exception as e:
        logging.exception(e)
```

포인트 클라우드의 충돌 파라미터를 설정하기

포인트 클라우드의 충돌 파라미터를 설정할 때 대응한 Mech-Viz 인터페이스는 setConfig() 입니다. 인터페이스는 실행 전반 속도를 설정할 때의 인터페이스와 같으며 설정 내용만 다릅니다. 아래와 같습니다.

예시

```
msg = {}
msg["function"] = "setConfig"
msg["check_pcl_collision"] = True
msg["collided_point_thre"] = 5000
msg["collide_area_thre"] = 20
msg["pcl_resolution_mm"] = 2
self.call("executor", msg)
```

Mech-Viz 반환값

Mech-Viz 의 반환값은 아래와 같습니다.

반환값	정의
Finished	실행 과정이 정상적으로 완료되었습니다. 주의할 것은 Mech-Viz 프로젝트에서 [비전 이동] 태스크의 오른쪽 분기가 다른 태스크와 연결될 때도 Mech-Viz 가 데이터를 정상적으로 반환할 것입니다.
Command Stop	[정지] 버튼을 누르거나 stop_viz() 함수를 호출합니다.
No targets	비전 포즈가 없습니다. 즉 Mech-Vision 에서 반환된 데이터가 비어 있습니다.
No proper vision poses	비전 포즈에 도달하지 못합니다. 즉 로봇은 현재 대상 물체의 위치에 도달하지 못하거나 대상 물체와 충돌했습니다.
PlanFail	Mech-Viz 경로 계획 실패
SceneCollision	충돌 발생

Mech-Viz 에서 반환된 데이터에 대해 Adapter 상위 클래스가 대응한 인터페이스 함수를 제공합니다.

RobotService 인터페이스

이 부분에서 아래와 같이 RobotService 와 관련된 인터페이스를 소개하고자 합니다.

- *moveXs*

RobotServer 서비스 시뮬레이션

RobotServer 시뮬레이션 서비스는 보통 풀 컨트롤 (full control) 인 경우 실제 RobotServer 를 통해 로봇을 컨트롤하는 대신 RobotService 를 만들어 RobotServer 의 기능을 시뮬레이션하는 것입니다. RobotService 가 Mech-Center 에서 등록된 후 Mech-Viz 가 리얼 로봇과 통신하는 것처럼 RobotService 와 상호적으로 통신할 것입니다.

RobotServer 서비스 시뮬레이션의 주요 응용 시나리오가 아래와 같습니다.

- Eye In Hand 경우에 Mech-Vision 계산에 사용되는 로봇 관절 각도를 획득합니다.
- 리얼 로봇에서 포즈를 획득하여 RobotService 에 보냅니다.

예시

Adapter 하위 클래스에서 호출하는 방식은 아래와 같습니다.

```
def _register_service(self):
        """
        register_service
        :return:
        """
        if self.robot_service:
            return

        self.robot_service = RobotService(self)
        other_info = {'robot_type': self.robot_service.service_name}
        self.server, _ = register_service(self.hub_caller, self.robot_service, other_info)

        self.robot_service.setJ([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
```

RobotService 의 클래스가 아래와 같습니다.

```
class RobotService(JsonService):
    service_type = "robot"
    service_name = "robot"
    jps = [0, 0, 0, 0, 0, 0]
    pose = [0, 0, 0, 1, 0, 0, 0]

    def getJ(self, *_):
        return {"joint_positions": self.jps}

    def setJ(self, jps):
        logging.info("setJ:{}".format(jps))
        self.jps = jps

    def getL(self, *_):
        return {"tcp_pose": self.pose}

    def getFL(self, *_):
        return {"flange_pose": self.pose}

    def setL(self, pose):
        logging.info("setL:{}".format(pose))
        self.pose = pose
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    def moveXs(self, params, _):
        pass

    def stop(self, *_):
        pass

    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
```

getJ

getJ() 함수는 Mech-Viz 와 Mech-Vision 가 현재 관절 각도를 획득하는 데 사용됩니다. 구체적으로 아래와 같습니다. 보통 먼저 setJ() 함수를 통해 현재 관절 각도를 설정한 후 getJ() 를 호출합니다.

예시

```
def getJ(self, *_):
     pose = {"joint_positions": self._jps}
     return pose
```

1. Eye In Hand 로봇에서 보내 온 관절 각도를 setJ() 에 입력합니다.

   ```
   def setJ(self, jps):
     assert len(jps) == 6
     for i in range(6):
         jps[i] = deg2rad(float(jps[i]))
     self._jps = jps
     logging.info("SetJ:{}".format(self._jps))
   ```

   그 중 jps 는 로봇에서 보내 온 관절 각도 데이터이고 getJ() 에게 호출되도록 self._jps 에 수치를 할당합니다. getJ() 가 라디안 포맷의 데이터를 획득하기 때문에 단위의 전환를 주의해야 합니다.

2. Eye To Hand: 로봇의 현재 관절 각도에 따라 설정할 필요는 없지만 실제 로봇 상태를 시뮬레이션 하려면 안전한 목표점을 시작점 (보통 Home 포인트) 으로 설정해야 합니다. 그렇지 않으면 임의의 숫자가 할당되며 오류가 발생하기 쉽습니다.

   ```
   def getJ(self, *_):
        return {"joint_positions": [1.246689,-0.525868,-0.789761,-1.330814, 0.922581, 4.
   ↪364021]}
   ```

getFL

getFL() 함수는 로봇이 이미지를 캡처할 때의 플랜지 포즈를 제공하는 데 사용됩니다. Eye In Hand 모드에서 진행되는 캘리브레이션은 플랜지와 카메라의 상대적인 위치 관계를 설명하지만 결국 베이스 좌표계 아래의 데이터를 사용하기 때문에 이미지를 캡처했을 때의 플랜지 포즈를 제공해야 합니다.

```
def getFL(self, *_):
    return {"flange_pose": self.pose}
```

Eye In Hand 방식을 통해 포즈를 제공할 때 다음과 같은 사항을 주의해야 합니다.

1. 로봇이 이미지를 캡처했을 때의 JPs 를 반환하면 RobotService 에서 직접 setJ() 함수를 호출하여 (단위: 라디안) 이 함수를 [] 에 반환하십시오.
2. 로봇이 플랜지 포즈를 반환하면 다음과 같이 처리하십시오.
   - Mech-Vision 프로젝트 외부 파라미터 파일 extri_param.json 중 파라미터"is_eye_in_hand" 의 값은 true 로 설정하십시오.
   - RobotService 의 setFL() 를 호출하십시오 (단위: 미터, 포즈 유형: 사원수).

moveXs

Adapter 를 통해 생성한 RobotService 서비스는 moveXs() 를 사용하여 Mech-Viz 가 계획한 경로 이동 목표점을 수신합니다. 구체적으로 아래와 같습니다.

함수 정의

```
def moveXs(self, params, _):
     with self.lock:
         for move in params["moves"]:
             self.targets.append(move)
     return {"finished": True}
```

그중에 params 가 Mech-Viz 프로젝트의 모든 파라미터를 입력하고 params[《moves》] 를 통해 모든 이동 포인트 (비전 이동, 상대적인 이동 등 포함) 의 포즈를 획득할 수 있습니다. 포즈는 기본적으로 관절 각도의 형식으로 반환되고 포즈를 self.targets 에 입력한 후 나중에 파라미터를 조절하는 데 사용될 것입니다.

예시

일반적으로 이 기능은 [알림 [태스크와 함께 사용되어야 합니다. Adapter 가 알림에서 보내 온 메시지를 수신했을 때 self.targets 를 함수에 입력하고 전환 및 압축한 후 로봇에 전송합니다. 예를 들면 아래와 같습니다.

```
def notify(self, request, _):
    msg = request["notify_message"]
    logging.info("{} notify message:{}".format(self.service_name, msg))
    if msg == "started":
        with self.lock:
            self.move_targets.clear()
    elif msg == "finished":
        with self.lock:
            targets = self.move_targets[:]
            self.move_targets.clear()
        self.send_moves(targets)
```

메시지"started" 를 수신했을 때 Mech-Viz 오류로 인해 중단되어 이동 전/후 두 개의 이동 포인트가 서로 겹치는 것을 방지하기 위해 목표점 리스트를 지워야 합니다. 메시지"finished" 를 수신했을 때 목표점 리스트를 pack_move 함수에 입력해 데이터를 정리하고 보내는 데 사용됩니다.

```
def pack_move(self, move_param):
    move_list = []
    for i, move in enumerate(move_param):
        target = move["target"]
        move_list.append(target)
    logging.info("move list num:{}".format(len(move_list)))
    logging.info("move list:{}".format(*move_list))
    motion_cmd = pack('>24f', *move_list)
    self.send(motion_cmd)
```

작업 현장의 실제 수요에 따라 Mech-Viz 의 모든 이동 목표점을 전송해도 되고 index 에 근거하여 몇 개를 선택해서 전송해도 됩니다. pack_move() 함수는 각 로봇 브랜드가 필요한 데이터 포맷에 따라 데이터를 정리합니다 (일반적으로 통신 프로토콜에서 미리 설정됨).

기타 인터페이스

이 부분에서는 Adapter 의 기타 인터페이스 사용에 대해 소개합니다. 구체적으로 다음 인터페이스를 포함합니다.

알림 서비스

Mech-Viz 프로젝트가 특정 분기 또는 특정 스텝으로 실행될 때 Adapter 프로그램의 해당 함수를 호출하려는 경우 Mech-Viz 에 알림 태스크를 추가할 수 있습니다.

예시

예를 들어, Adapter 에 철거 횟수를 1 씩 늘리는 기능이 작성되어 있으면 디팔레타이징 프로세스의 마지막 태스크 후에 알림 태스크를 추가할 수 있습니다. 이 지점에 도달하면 Adapter 가 트리거될 수 있고 해당 함수를 호출할 수 있습니다. 이 기능을 구현하는 예시는 아래와 같습니다.

1. NotifyService 를 상속하는 클래스를 작성하십시오.

   ```
   from interface.services import NotifyService, register_service

   class NotifyService(NotifyService):
       service_type = "notify"
       service_name = "FANUC_M410IC_185_COMPACT"

       def __init__(self, update_success_num, update_fail_num):
           self.update_success_num = update_success_num
           self.update_fail_num = update_fail_num

       def handle_message(self, msg):
           if msg == "Success":
               self.update_success_num()
           elif msg == "Fail":
               self.update_fail_num()
   ```

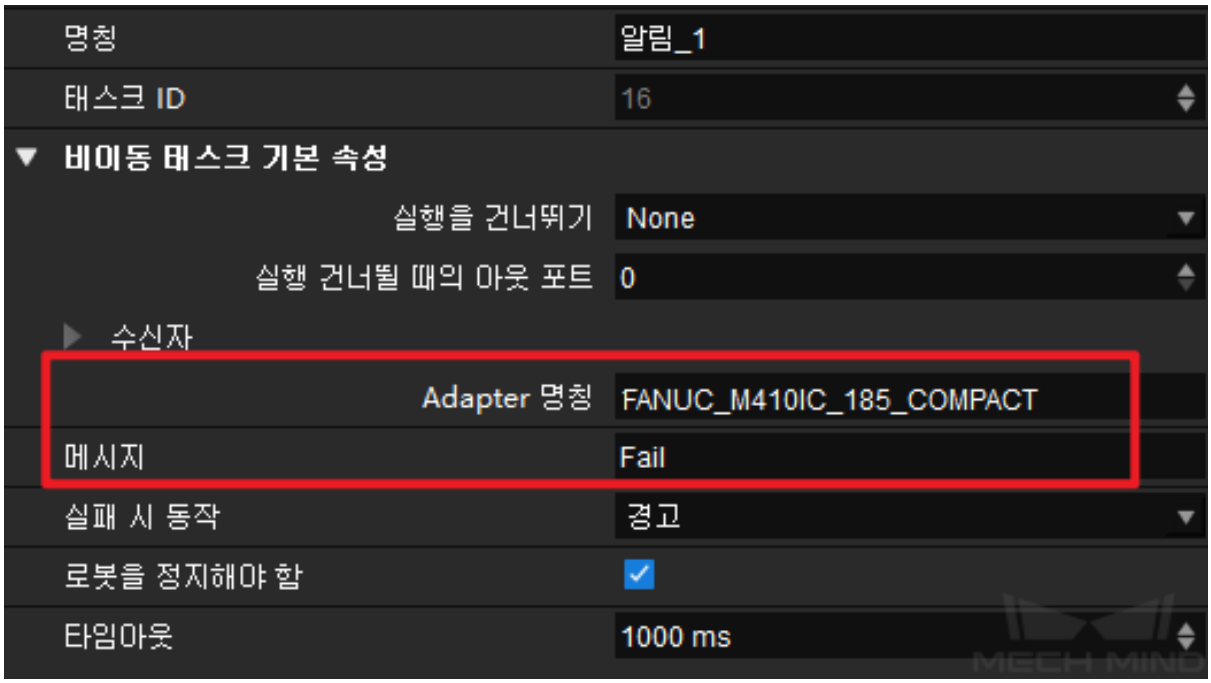
   이 알림은 다음 기능을 수행할 수 있습니다. Mech-Viz 가"Success" 을 보내는 알림 태스크까지 실행되면 Adapter 는 update_success_num() 함수를 호출할 수 있고"Fail" 에 도달하면 Adapter 는 update_fail_num() 함수를 호출합니다.

2. Mech-Viz 메인 프로그램을 제어하는 클래스에서 NotifyService 클래스를 인스턴스화하고 이 서비스를 등록합니다.

```
class MyClient(TcpClientAdapter):

    def __init__(self, host_address):
        super().__init__(host_address)
        self._register_service()

    def _register_service(self):
        self.robot_service = NotifyService(self.update_success_num, self.update_fail_num)
        self.server, port = register_service(self.hub_caller, self.robot_service)

    def update_success_num(self):
        # the num of unstack successfully plus 1
        self.success_num += 1

    def update_fail_num(self):
        # the num of unstack fiplus 1
        self.fail_num += 1
```

3. Mech-Viz 에서 필요한 곳에 해당 알림 태스크를 추가합니다.

   알림 태스크에서 가장 중요한 것은 **Adapter 명칭** 과 **메시지** 를 채우는 것입니다. 이 두 곳의 값은 위의 NotifyService 클래스의 service_name 및 msg 값과 일치해야 합니다.



   프로그램이 실행되면 Mech-Center 인터페이스에 service_type 및 service_name 이 나타납니다. 이는 알림 서비스 등록이 성공했음을 나타냅니다.

VisionWatcher 서비스

Mech-Vision 이 실행을 마치면 일부 결과가 출력됩니다 (예: vision result:{'noCloudInRoi': False, 'function': 'posesFound', 'vision_name': 'TJTvision-3'}). 일부 비정상적인 상황의 경우 Adapter 는 VisionWatcher 서비스를 통해 상기시키기 위해 오류 정보를 보낼 수 있습니다.

예시

1. VisionResultSelectedAtService 를 상속하는 클래스를 작성하십시오.

```
from interface.services import VisionResultSelectedAtService, register_service

class VisionWatcher(VisionResultSelectedAtService):
    def __init__(self, send_err_no_cloud):
        super().__init__()
        self.send_err_no_cloud = send_err_no_cloud

    def poses_found(self,result):
        has_cloud_in_roi = not result.get("noCloudInRoi", False)

        if not has_cloud_in_roi:
            time.sleep(2)
            self.send_err_no_cloud()
```

   하위 클래스 VisionWatcher 는 상위 클래스의 poses_found() 함수를 다시 작성해야 하므로 Adapter 에서 send_err_no_cloud()(오류 정보를 전송하는 함수) 호출하는 논리가 poses_found() 에서 변할 것입니다. 실행 과정 동안 Mech-Vision 이 반환한 포즈의 값은 poses_found() 의 result 파라미터로 전송됩니다.

2. Mech-Viz 메인 프로그램을 제어하는 클래스에서 VisionWatcher 를 인스턴스화합니다.

```
class MyClient(TcpClientAdapter):
    def __init__(self, host_address):
        super().__init__(host_address)
        self._register_service()
    def _register_service(self):
        self.robot_service = VisionWatcher(self.send_err_no_cloud)
        self.server, port = register_service(self.hub_caller, self.robot_service)

    def send_err_no_cloud(self):
        # send no cloud error message
        self.send("12,NoCloudErr,done".encode())
```

   VisionWatcher 클래스를 인스턴스화하는 과정에 send_err_no_cloud() 함수를 파라미터로 VisionWatcher() 에 전송합니다. 포인트 클라우드가 표시되지 않으면 poses_found() 의 논리에 따라 오류 메시지를 보내는 함수가 호출됩니다.

   프로그램이 실행되면 등록된 서비스가 Mech-Center 인터페이스에 나타납니다. 그러면 Adapter 가 VisionWatcher 서비스를 성공적으로 등록했음을 나타냅니다.

3.4.4 Adapter 프로그래밍 샘플

Adapter 프로그래밍 구문 및 프로그래밍 논리에 익숙해지면 이 부분에서 제공하는 샘플 프로그램을 참조하여 자신의 Adapter 프로그램을 작성할 수 있습니다.

- *Mech-Vision* 만 사용하여 포즈를 보내기

Mech-Vision 만 사용하여 포즈를 보내기

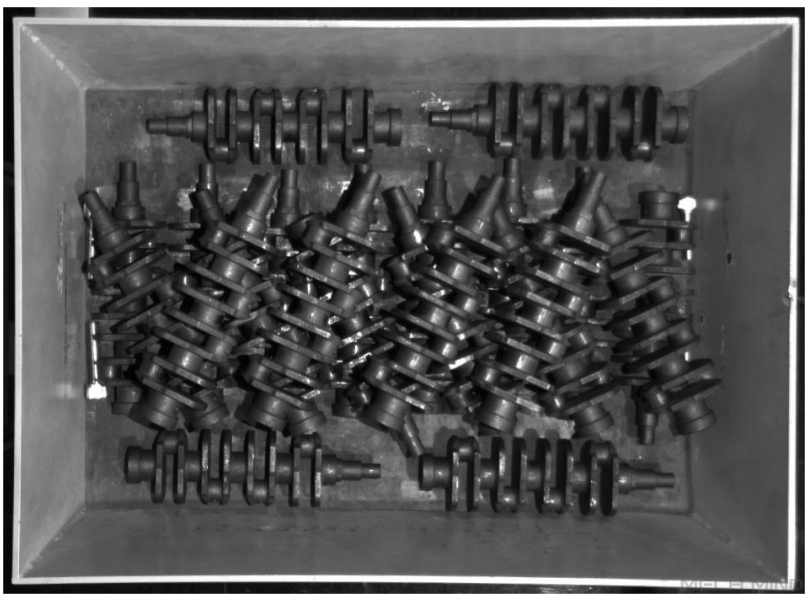
이 부분에서는 Mech-Vision 만 사용하여 포즈를 보내는 Adapter 샘플 프로그램에 대해 자세히 설명하고자 합니다. 이 부분의 내용에는 다음이 포함됩니다.

배경 소개

이 샘플은 크랭크축 텐딩의 응용 시나리오에 관한 것입니다. 카메라는 상자 위의 브래킷에 고정됩니다. Mech-Vision 은 사진을 캡처하고 피킹할 수 있는 작업물의 좌표를 출력해 로봇 측에 보냅니다.

이 샘플은 Mech-Vision 의 내장 예시 프로젝트“대형 비 평면 부품”(**파일** → **샘플 프로젝트를 보기** → **부품 로드언 로드** → **대형 비 평면 부품**) 을 사용합니다.

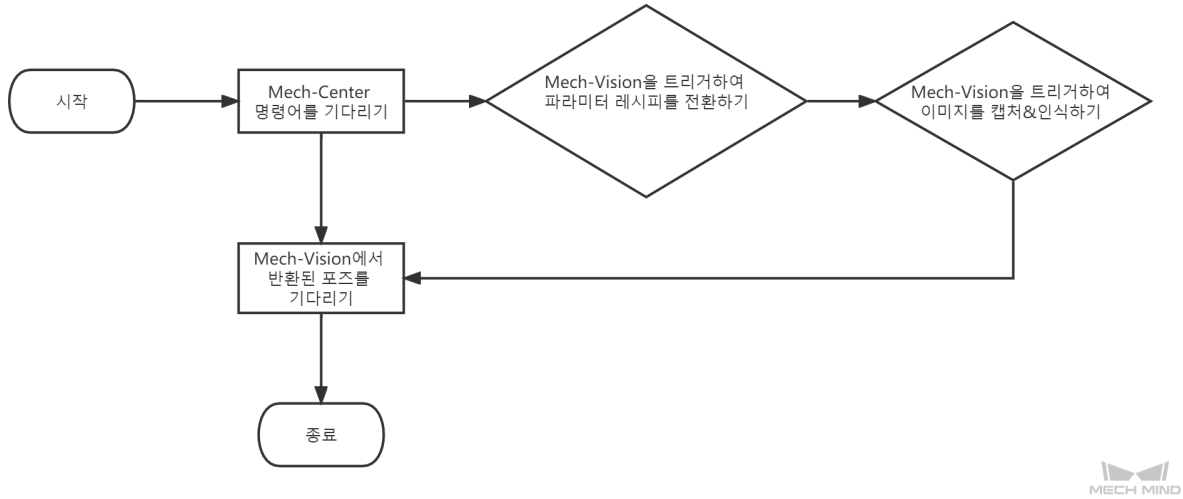
이 프로젝트는 3D 모델 매칭 알고리즘을 사용하고 다른 작업물에 대해 다른 모델 파일과 픽 포인트를 설정해야 합니다. 따라서 Mech-Vision 에서 파라미터 레시피를 설정해야 하며, 로봇 측에서 사진 캡처 명령을 보낼 때 레시피 번호 (작업물 번호) 를 설정해야 합니다.



통신 방안

로봇과 Mech-Mind 비전 시리즈 소프트웨어를 설치하는 IPC 는 TCP/IP Socket 프로토콜을 사용하여 통신하며 통신 형식은 ASCII 문자열이며 데이터 구분 기호는 영어 쉼표 (,) 를 사용합니다. 그 중 비전 시리즈 소프트웨어는 통신 서버로, 로봇은 클라이언트로 사용됩니다.

통신 프로세스는 아래 그림과 같습니다.



통신 프로세스의 상세한 설명은 아래와 같습니다.

1. Mech-Center 는 로봇이 이미지 캡처 명령``P``와 레시피 번호를 보낼 때까지 기다립니다.
2. Mech-Center 는 Mech-Vision 을 트리거하여 파라미터 레시피를 전환합니다.
3. Mech-Center 는 Mech-Vision 을 트리거하여 이미지를 캡처하고 대상물을 인식합니다.
4. Mech-Vision 이 이미지를 캡처하고 성공적으로 인식한 후 상태 코드와 포즈를 Mech-Center 에 반환합니다.
5. Mech-Center 는 상태 코드와 포즈를 로봇에 반환합니다.

참고: 로봇 피킹를 용이하게 하기 위해 Mech-Center 는 피킹할 작업물의 좌표를 로봇 TCP 좌표로 변환합니다.

통신 메시지 형식

자세한 통신 메시지 형식은 아래와 같습니다.

	이미지 캡처 명령	**작업물 번호**
전송 로봇 -> IPC	P	정수, 값 범위: 1~100
	상태 코드	**포즈 (TCP 좌표)**
수신 IPC -> 로봇	정수, 값 범위: 0~4 (0-정상 인식, 1-잘못된 명령 코드, 2-Vision 프로젝트가 등록되지 않음, 3-포즈가 없음, 4-포인트 클라우드가 없음)	x,y,z,a,b,c 형식의 쉼표 (,) 로 구분된 6 개의 부동 소수점 데이터입니다.

참고: 응답 메시지의 길이는 고정되어 있습니다. 응답 메시지의 상태 코드가 비정상 코드 (1~4) 인 경우 비전 포인트 데이터는 0 으로 채워집니다.

통신 메시지 샘플

요청 메시지

```
P,1
```

정상적인 응답 메시지

```
0,1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517
```

참고: 이 샘플에서 Mech-Vision 은 정상적으로 인식하며 반환된 TCP 좌표는 1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517 입니다.

비정상적인 응답 메시지: 잘못된 명령 코드

```
1,0,0,0,0,0,0
```

비정상 응답 메시지: 비전 프로젝트가 등록되지 않음

```
2,0,0,0,0,0,0
```

비정상적인 응답 메시지: 포즈가 없음

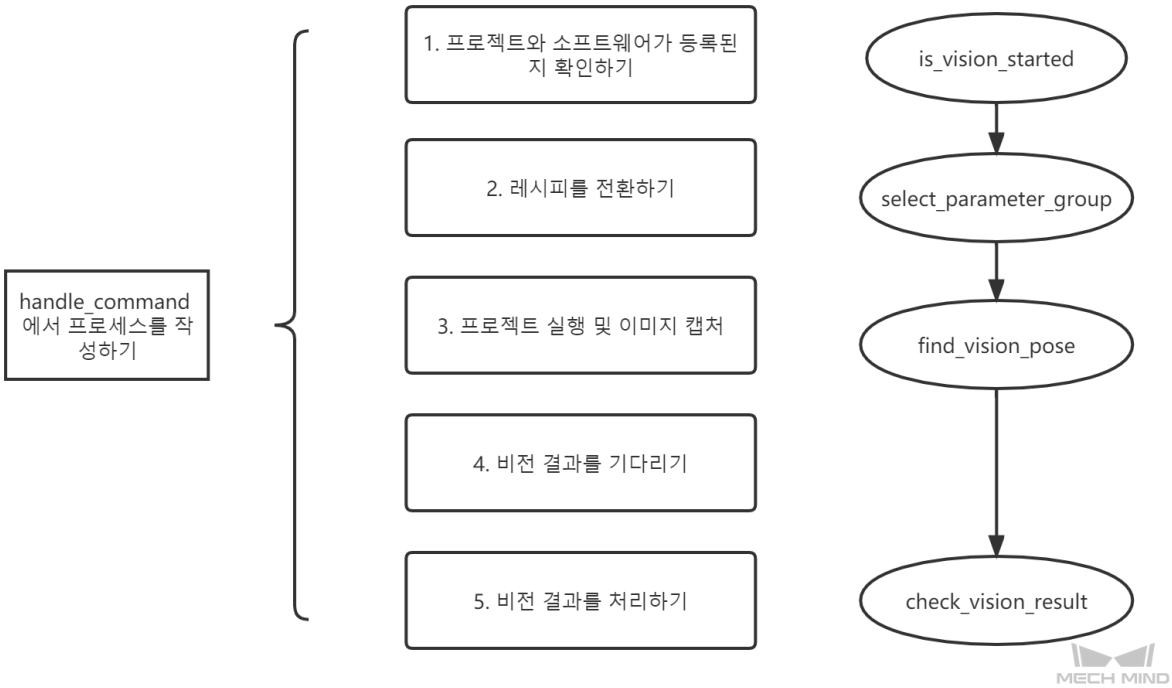
```
3,0,0,0,0,0,0
```

비정상적인 응답 메시지: 포인트 클라우드가 없음

```
4,0,0,0,0,0,0
```

프로그래밍 맥락

이 샘플은 프로젝트 목표를 달성하기 위해 다음 그림과 같은 맥락에 따라 Adapter 프로그램을 작성합니다.



위의 그림은 샘플 프로그램의 메시지 처리 논리만을 나열한 것입니다. 다음 부분에서는 샘플 프로그램에 대해 자세히 설명합니다.

샘플 프로그램 상세 소개

참고: Adpater 샘플 프로그램 을 클릭하여 다운로드하십시오.

Python 패키지를 가져오기

Adapter 프로그램이 의존하는 모든 모듈을 도입합니다.

```
import json
import logging
import math
import sys
from time import sleep
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..", "..")))
from transforms3d import euler
from interface.adapter import TcpServerAdapter, TcpClientAdapter
from util.transforms import object2tcp
```

클래스 정의

“TcpServerAdapter”상위 클래스를 상속하는“TestAdapter”하위 클래스를 정의하십시오.

```
class TestAdapter(TcpServerAdapter):
    vision_project_name = "Large_Non_Planar_Workpieces"
    # vision_project_name = 'Vis-2StationR7-WorkobjectRecognition-L1'
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
    is_force_real_run = True
    service_name = "test Adapter"

    def __init__(self, address):
        super().__init__(address)
        self.robot_service = None
        self.set_recv_size(1024)
```

참고: 이 샘플은 Adapter 프로그램을 TCP/IP 소켓 통신용 서버로 정의합니다.

명령 수신 및 논리 처리 설정

수신을 받기 위한 처리 논리를 설정합니다 (이미지 캡처 및 파라미터 레시피 포함).

```
# Receive command _create_received_section
def handle_command(self, cmds):
    photo_cmd, *extra_cmds = cmds.decode().split(',')
    recipe = extra_cmds[0]
    # Check command validity _check_cmd_validity_section
    if photo_cmd != 'P':
        self.msg_signal.emit(logging.ERROR, 'Illegal command: {}'.format(photo_cmd))
        self.send(('1' + '' + '').encode())
        return
    # Check whether vision is registered _check_vision_service_section
    if not self.is_vision_started():
        self.msg_signal.emit(logging.ERROR, 'Vision not registered: {}'.format(self.
↪vision_project_name))
        self.send(('2' + '' + '').encode())
        return
    # Change TODO parameter "extra_cmds" according to actual conditions
    sleep(0.1)  # wait for a cycle of getting in Vision
    # _check_vision_result_function_section

    try:
        result = self.select_parameter_group(self.vision_project_name, int(recipe) -␣
↪1)
        if result:
            result = result.decode()
            if result.startswith("CV-E0401"):
                self.send(('5' + '' + '').encode())
                return
            elif result.startswith("CV-E0403"):
                self.send(('5' + '' + '').encode())
                return
            raise RuntimeError(result)
    except Exception as e:
        logging.exception('Exception happened when switching model: {}'.format(e))
        self.send(('5' + '' + '').encode())
        return
    self.show_custom_message(logging.INFO, "Switched model for project successfully")

    self.msg_signal.emit(logging.WARNING, 'Started capturing image')
    try:
        self.check_vision_result(json.loads(self.find_vision_pose().decode()))

    except Exception as e:
```

(다음 페이지에 계속)

(이전 페이지에서 계속)

```
        self.msg_signal.emit(logging.ERROR, 'Calling project timed out. Please check␣
↪whether the project is correct: {}'.format(e))
        self.send(('2' + '' + '').encode())
```

참고: “handle_command”함수는 TCP/IP 소켓 서버가 수신한 메시지에 대한 처리 인포트로 사용됩니다.

Mech-Vision 비전 결과의 검사를 정의하기

Mech-Vision 에서 출력한 비전 결과를 검사하도록 Adapter 를 설정합니다.

```
# Check vision results
def check_vision_result(self, vision_result, at=None):

    noCloudInRoi = vision_result.get('noCloudInRoi', True)
    if noCloudInRoi:
        self.msg_signal.emit(logging.ERROR, 'No point clouds')
        self.send(('4' + '' + '').encode())
        return

    poses = vision_result.get('poses')
    labels = vision_result.get('labels')
    if not poses or not poses[0]:
        self.msg_signal.emit(logging.ERROR, 'No visual points')
        self.send(('3' + '' + '').encode())
        return

    self.send(self.pack_pose(poses, labels).encode())
    self.msg_signal.emit(logging.INFO, 'Sent TCP successfully')
```

포즈의 출력 형식을 설정하기

비전 포인트의 외부로 출력하는 형식을 설정합니다.

```
# Pack pose _pack_pose_section
def pack_pose(self, poses, labels, at=None):

    pack_count = min(len(poses), 1)
    msg_body = ''
    for i in range(pack_count):
        pose = poses[i]
        object2tcp(pose)
        t = [p * 1000 for p in pose[:3]]
        r = [math.degrees(p) for p in euler.quat2euler(pose[3:], 'rzyx')]
        p = t + r
        self.msg_signal.emit(logging.INFO, 'Sent pose: {}'.format(p))
        msg_body += ('{:.4f},' * (len(p) - 1) + '{:.4f}').format(*p)

        if i != (pack_count - 1):
            msg_body += ','
    return '{},'.format(0) + msg_body + ''
```

Adapter 를 닫는 작업을 정의하기

Adapter 를 닫는 방법을 정의합니다.

```
def close(self):
    super().close()
```