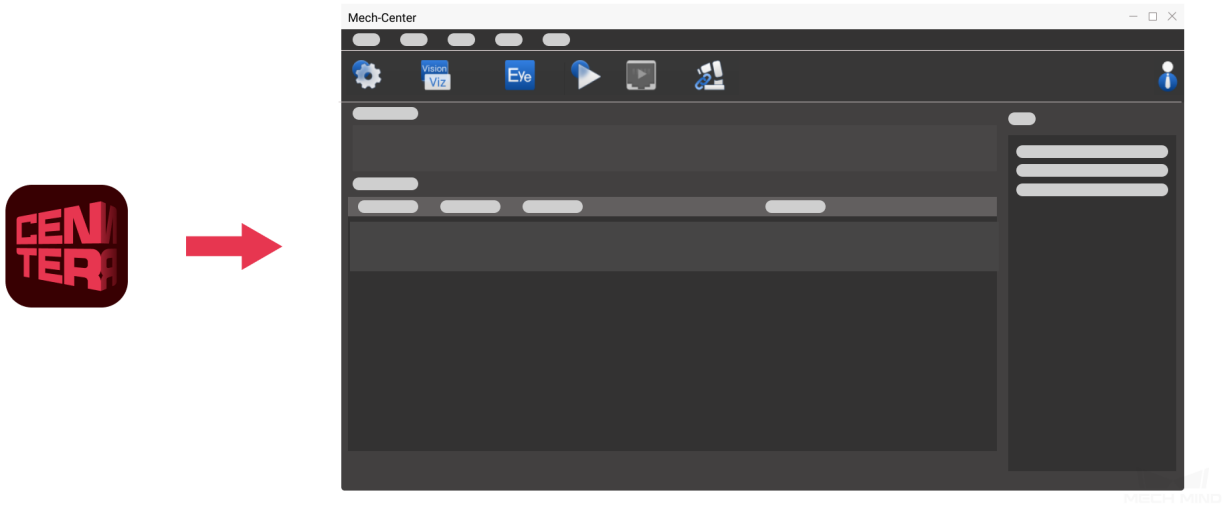
Mech-Center Manual

Mech-Mind

2023 年 08 月 02 日

Contents



Mech-Center 是本公司自主研发的 Mech-Mind 相关软件的中枢和控制中心，可完成相关软件的全局设置，工程的整体备份及还原，直观查看 Mech-Viz、Mech-Vision、Mech-Eye Viewer、机器人、标准接口、Adapter 的状态。并且作为对外接口 Mech-Interface 的启动及管理中心。

查看以下内容，了解 Mech-Center **界面组成及各部分功能**

快速了解 Mech-Center

查看以下内容，了解 Mech-Center 的 **基础使用流程**

开始使用 Mech-Center

查看以下内容，了解软件的 **对外接口服务**

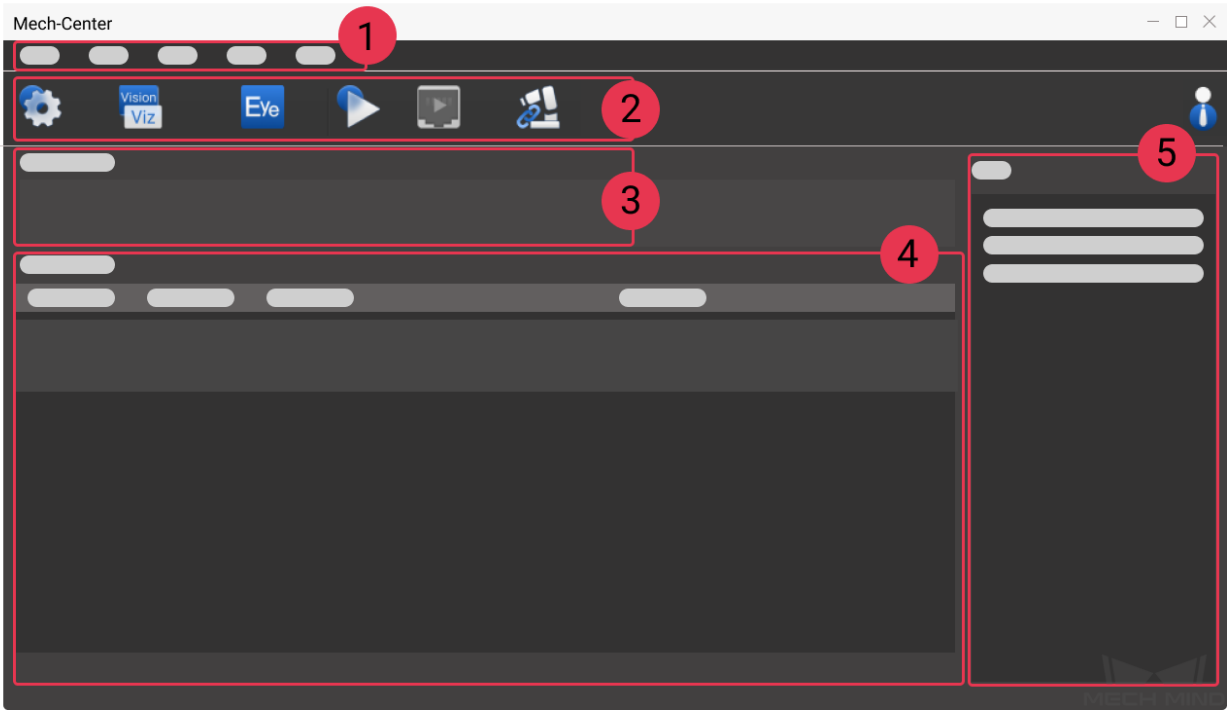
Mech-Interface

CHAPTER 1

快速了解 Mech-Center

Mech-Center 是本公司自主研发的 Mech-Mind 相关软件的中枢和控制中心，可完成相关软件的全局设置，工程的整体备份及还原，直观查看 Mech-Viz、Mech-Vision、Mech-Eye Viewer、机器人、标准接口、Adapter 的状态。并且作为对外接口 Mech-Interface 的启动及管理中心。

Mech-Center 主界面由五部分组成：



1. 菜单栏：工程操作、用户及视图管理、生成 Adapter 和软件信息查看等功能。
2. 工具栏：部署设置、软件及接口服务启动、连接机器人和运行等按钮。
3. 服务状态栏：显示注册的软件、相机、机器人等。
4. 工程状态栏：显示 Mech-Viz/Mech-Vision 工程状态、运行时间和详细信息。
5. 日志栏：实时显示当前工程及服务的日志信息。

1.1 菜单栏

菜单栏提供操作相关的基础功能，包含文件、工具、用户、视图和帮助。

1.1.1 文件

选项	描述	快捷键
备份工程	此选项用于生成 Mech-Mind 软件系统的工程备份。它将保存自动加载的 Mech-Viz / Mech-Vision 工程、Mech-Center 的适配器工程和 Mech-Center 的部署设置。	Ctrl+B
恢复工程	此选项用于从选择的备份中还原工程和设置，恢复后自动加载的工程会被替换，请确保工程已备份。	Ctrl+R
导入工程	此选项用于从其他路径导入工程的情况。	Ctrl+I
退出	退出 Mech-Mind 软件系统。	Ctrl+Q

注意： 在操作员模式下，仅提供备份工程和退出选项。

1.1.2 工具

选项	描述
打包调试数据	此选项用于打包 Mech-Mind 软件系统工程的调试数据，可选择时间段、打包路径、保存选项。
Adapter 生成器	此选项用于生成定制化 Adapter 程序。
日志查看器	此选项可查看 Mech-Viz / Mech-Vision 的日志信息，需手动选择日志文件，然后加载日志文件才可查看。另外，此选项支持根据日志级别查看相应日志，也支持输入关键字（可选正则表达式、区分大小写、通配符）过滤或查找日志，也可滑动时间轴线选择该时间之后的日志。
显示服务状态	此选项用于显示软件、机器人、各项接口等服务的连接状态。

注意：在操作员模式下，无 Adapter 生成器选项。

1.1.3 用户

修改密码：仅在管理员模式下，可修改管理员的密码。操作员无法使用该功能。

注意：若工控机首次安装 Mech-Center 软件，管理员的默认密码是 123456。另外，修改密码后，请妥善保管密码，切勿忘记。

1.1.4 视图

日志：可勾选是否在主界面中显示日志栏。

1.1.5 帮助

选项	描述	快捷键
用户手册	此选项用于快速打开接口使用文档目录。	F1
更新说明	此选项用于说明每个版本的新功能和修复信息。	无
关于	此选项显示软件版本及版权信息。	无

1.2 工具栏

Mech-Center 工具栏包括部署设置、启动 Viz/Vision 、启动相机查看器、运行、启动接口服务、主控机器人和管理员七个按钮。

1.2.1 部署设置

部署设置用于实现 Mech-Center 的基础设置、配置各软件的打开路径、查看自动加载的工程路径和选择外部服务等功能。

偏好设置

偏好设置选项用于个性化的全局设置，如下图所示。用户可根据自身使用习惯进行相关设置，如：切换软件主题色彩、切换 Mech-Center 软件界面语言（目前软件支持中、英、日、韩四种语言）、更改保存日志等。

- 勾选 运行时隐藏控制台后，启动 Mech-Center 软件后，控制台将隐藏。
- 勾选 开机时自动运行 *Mech-Center* 后，可以实现 Mech-Center 软件开机自启。
- 勾选 自动启动 *Mech-Viz/Mech-Vision/Mech-Interface*（如果有）后，可以在启动 Mech-Center 软件后，Mech-Viz/Mech-Vision/Mech-Interface 也随后自启。
- 勾选 启动 *Mech-Viz/Mech-Vision* 到托盘后，启动的 Mech-Viz/Mech-Vision 软件将隐藏至桌面状态栏的托盘处。

注意： 保存更改并重启 Mech-Center 后，设置才能生效。

Mech-Viz

Mech-Viz 选项用于设置 Mech-Viz 软件的打开路径并显示自动加载的工程路径，如下图所示。

单击左侧 *Mech-Viz* ，进入设置界面，默认勾选 *启用 Mech-Viz* 。

单击程序路径后的 …，选择 Mech-Viz 安装目录下的 mmind_viz.exe 文件，即可完成路径选择。

在 Mech-Viz 中勾选 自动加载当前工程，工程路径会自动加载，无需手动添加路径。

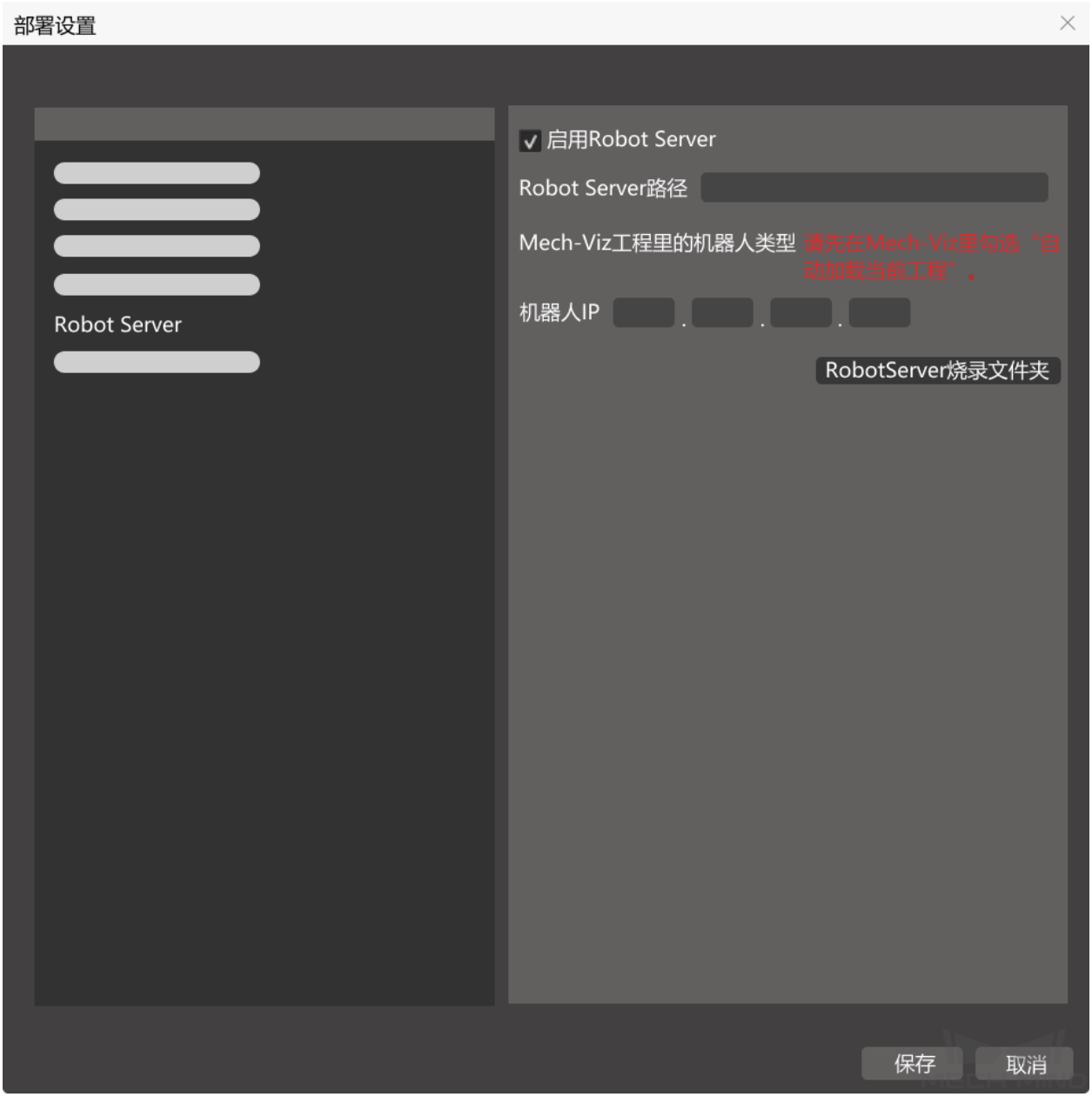
注解： 勾选 **仿真运行**后，Mech-Viz 软件将只进行仿真，不会移动真实机器人。

Mech-Vision 、Mech-Eye Viewer

Mech-Vision、Mech-Eye Viewer 部署设置方式与 Mech-Viz 类似。

Robot Server

Robot Server 选项用于适配机器人，以实现 Mech-Viz 软件的主控，如下图所示。



单击左侧 *Robot Server*，进入设置界面。默认勾选 *启用 Robot Server*，并自动加载随 Mech-Center 一起安装的 Robot Server 路径。

注意： 成功加载 Mech-Viz 工程后，Mech-Viz 工程里的机器人类型将自动显示，无需填写。机器人 IP 需按照实际工程中的设置填写，并确保正确。

Mech-Interface

Mech-Interface 是统一的外部接口，实现与第三方的通信。

1.2.2 启动 Mech-Viz、Mech-Vision

Mech-Viz/Mech-Vision 软件成功启动后，软件图标及其版本会显示在服务状态栏。

注意:

1. 运行工程时将会进行版本兼容性检测，建议搭配 V1.4.0 及以上版本的 Mech-Viz、Mech-Vision、Mech-Center 使用。
2. 当 Mech-Center 发现另外两款软件版本均低于 1.4.0 时，将提示版本必须高于 1.4.0 ，并且点击 运行 后弹出错误提示框。

1.2.3 启动相机查看器

Mech-Eye Viewer 软件成功启动后，软件图标会出现在桌面状态栏。

1.2.4 运行

运行 Mech-Viz 和 Mech-Vision 中加载的工程。

1.2.5 启动接口服务

启动*Mech-Interface* 。

1.2.6 主控机器人

成功连接真实机器人后，机器人图标及其型号会显示在服务状态栏。

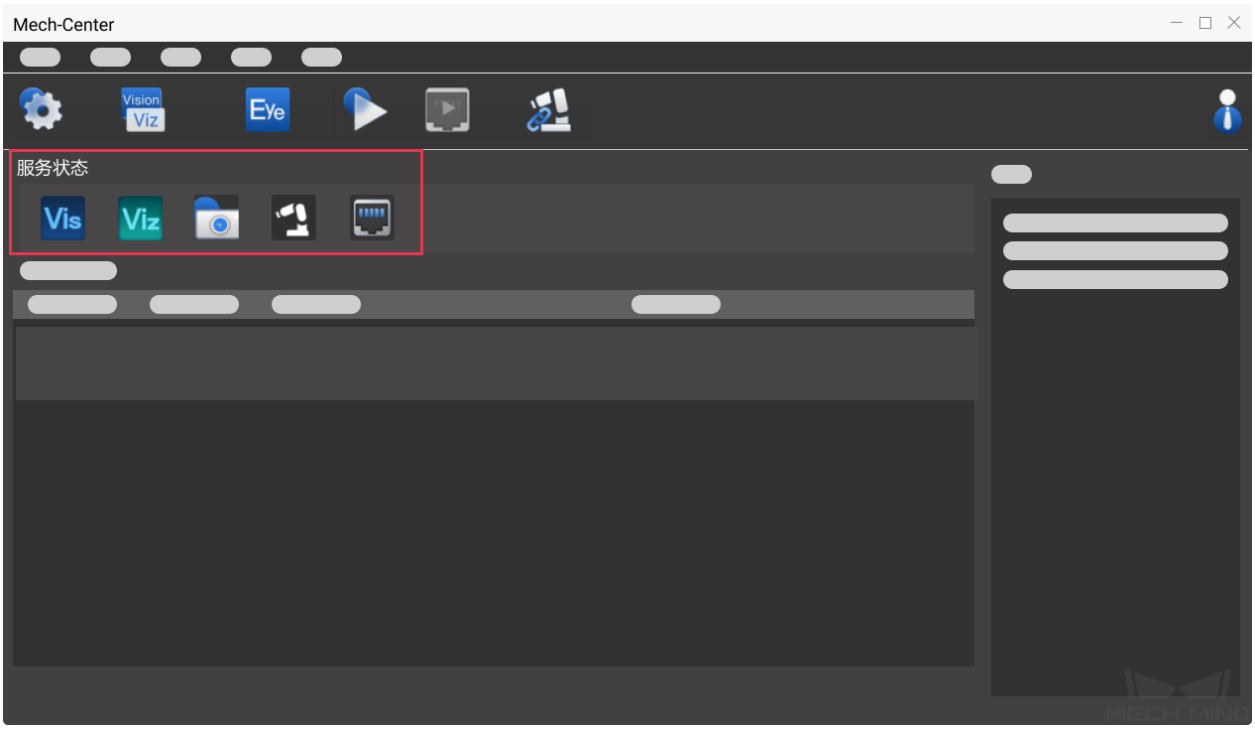
1.2.7 管理员

Mech-Center 中有管理员和操作员两种模式，默认为管理员模式。当使用操作员模式时，无法编辑工程和修改配置，图标显示为操作员模式。如需切换，可单击图标，在登录对话框选择用户类型后单击 登 录。

其中管理员模式需要登录密码，如需修改密码，可以在菜单栏的 用户 ▸ 修改密码处修改。

1.3 服务状态栏

在 Mech-Center 运行过程中，服务状态栏可以显示已启动的软件、相机、机器人和接口服务，并通过点击服务状态栏中的软件图标来打开对应的窗口。



1.4 工程状态栏

工程状态栏可以显示 Mech-Viz / Mech-Vision 工程名称、状态、执行时间和详细信息，通常与右侧日志栏相结合查看工程运行情况。

选项	描述
工程名称	Mech-Viz / Mech-Vision 工程名称。
状态	工程当前的状态（空闲或运行中）。
执行时间	工程从开始运行到执行结束所消耗的时间。
详细信息	记录工程运行时间和对应的状态，若工程运行错误，此处也将记录错误信息。

1.5 日志栏

实时显示当前运行工程及服务的日志信息。

选项	描述
D、i、W、E	表示日志等级，其中 D 表示调试信息，i 表示正常信息，W 表示警告信息，E 表示错误信息。单击某个等级选项便可对日志信息进行筛选，同时可选择多个等级选项。
清除	将上面日志信息全部清除。
打开文件夹	打开 Mech-Center 安装目录下的 `logs` 文件夹。

小技巧： 单击错误消息开头的错误码，便可跳转到在线文档查看详细信息。

CHAPTER 2

开始使用 Mech-Center

2.1 部署软件

打开 Mech-Center，单击工具栏 中的 **部署设置**  ，进入部署设置页面，如下图所示。在该界面中配置路径、参数等，完成后单击保存。具体内容请参考部署设置 介绍。

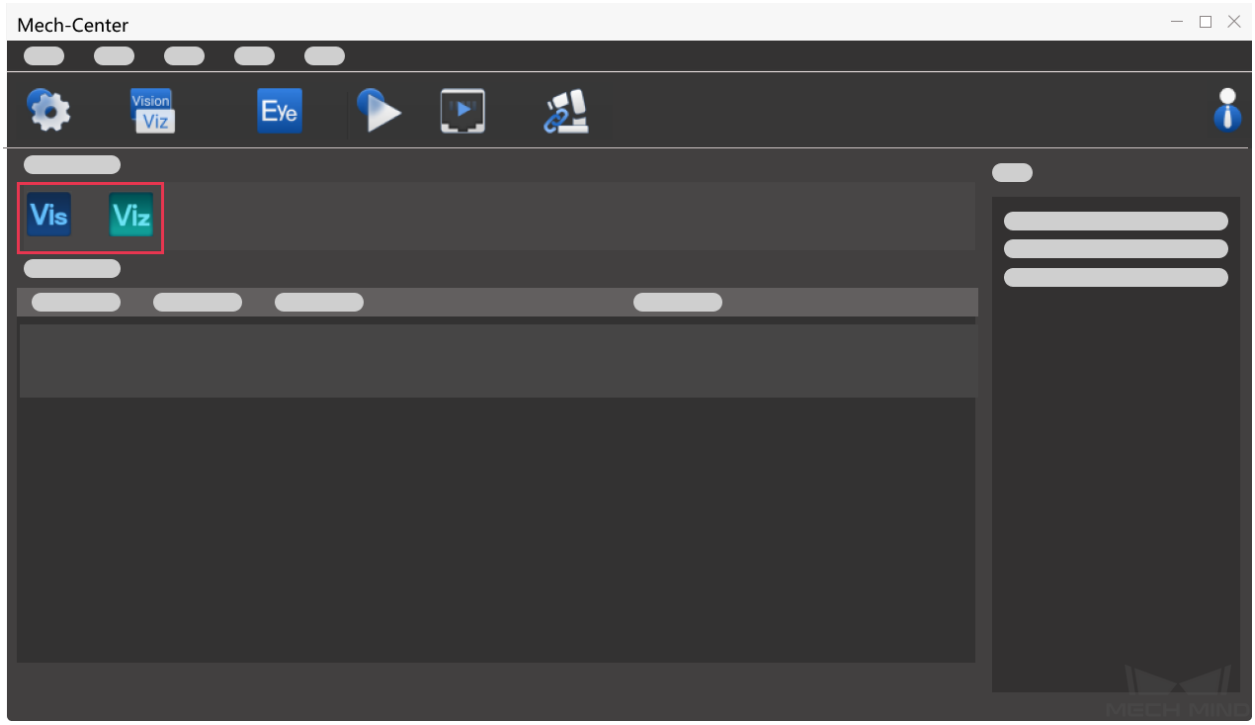
提示：

- 请根据实际使用情况进行部署设置。
- Mech-Vision、Mech-Viz 及 Mech-Eye Viewer 安装完成后，部署设置中的路径会自动添加，无需手动添加。

2.2 打开工程

单击工具栏 中的 **启动 Viz/Vision** 

，可打开 Mech-Viz 和 Mech-Vision，且在服务状态栏中显示图标。



2.2.1 打开 Mech-Viz 工程

1. 进入软件界面：单击 

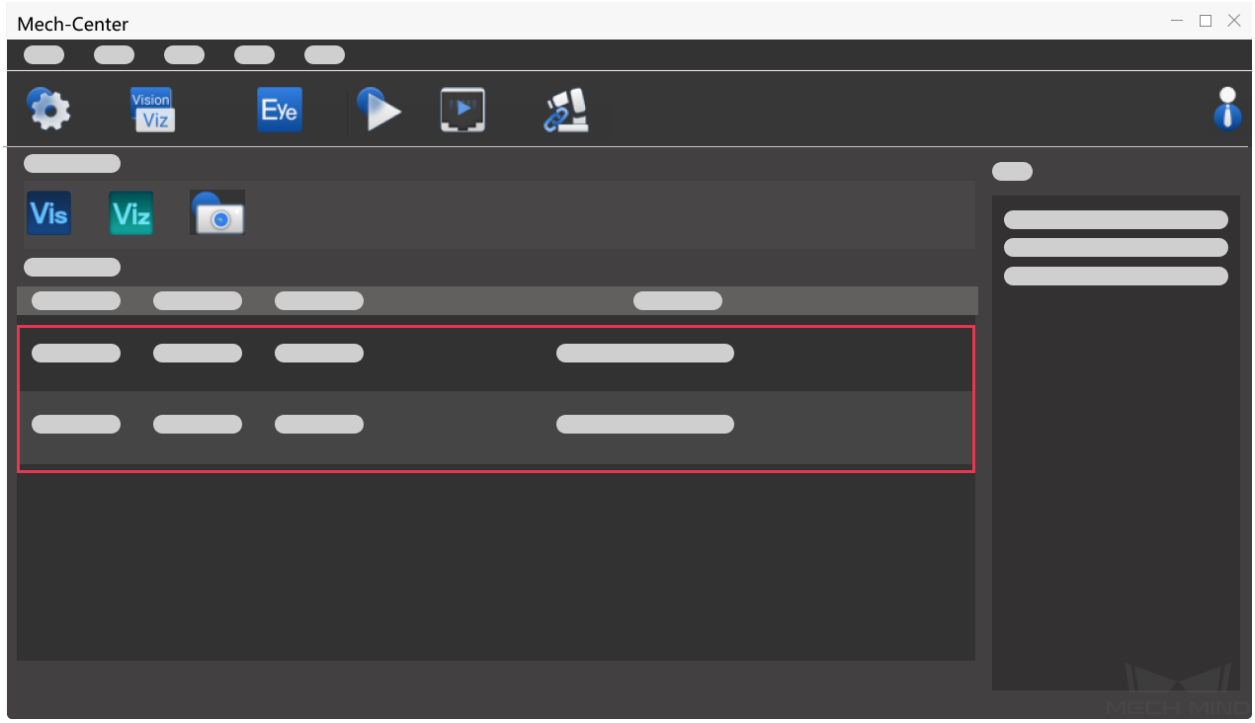
进入 Mech-Viz 主界面。
2. 搭建工程: 新建工程，或者打开已有工程。
3. 在 Mech-Viz 工程资源面板中，右键单击工程名称，勾选 设为自动加载。
4. 查看工程状态：工程状态栏中显示当前工程的运行状态。

提示:

- 勾选 设为自动加载后，Mech-Viz 工程的具体路径会自动添加至 Mech-Center 的 **部署设置 -> Mech-Viz -> 工程路径**中。

2.2.2 打开 Mech-Vision 工程

1. 进入软件界面：单击  进入 Mech-Vision 主界面。
2. 搭建方案或工程：新建方案，新建工程。
3. 自动加载当前方案或 自动加载当前工程。
4. 查看工程状态：工程状态栏中显示当前工程的运行状态，如下图所示。



2.2.3 修改相机设置

如运行过程中发现需查看或调整相机设置，可单击工具栏 中的 **启动相机查看器**  ，启动后在服务状态栏中显示图标。

2.3 连接机器人

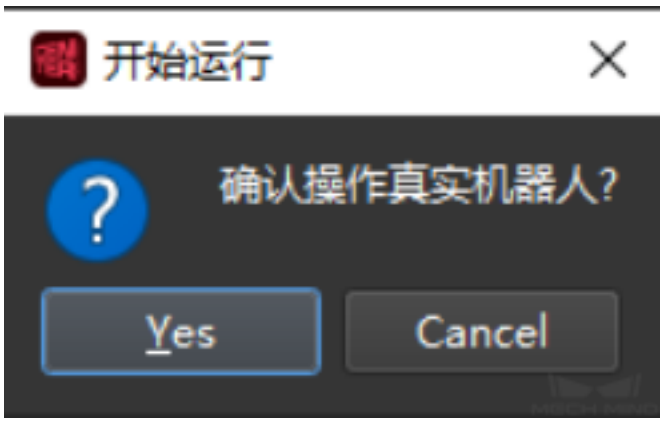
若用户现场真实机器人品牌不为 UR，需要进行机器人烧录操作，请参考 robot_integrations 。单击工具栏 中的 **主控机器人**  ，连接成功后，服务状态栏中显示机器人图标。

对于使用 Mech-Interface 的工程，单击工具栏 中的 **启动接口服务**  ，接口服务启动后，服务状态栏中显示对应图标。

2.4 运行工程

提示： 若运行真实机器人，请进入 **部署设置**中的 Mech-Viz 页面，取消勾选 *仿真运行*。

单击工具栏中的 **运行**  ，出现确认弹窗，如下图所示。单击 *Yes* ，运行已加载的工程，此时在工程状态栏可查看运行相关信息。



注意:

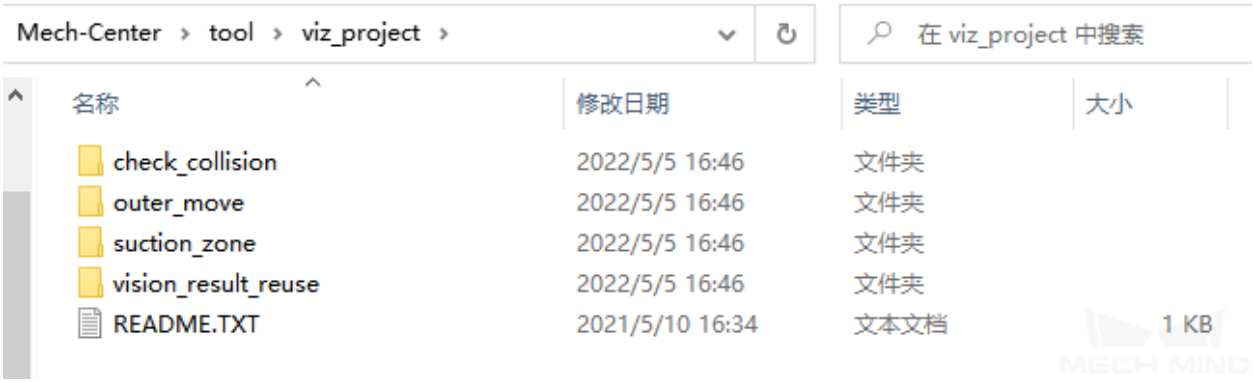
- 工程加载完成后再运行工程。

  若用户未处理打开 Mech-Viz 工程时的 **软件兼容性提示**，直接在 Mech-Center 中单击 运行，Mech-Center 的日志中将打印“Mech-Viz 正在加载工程，无法启动 Mech-Viz 工程：XXX”消息。此时用户需要处理软件兼容性提示，即单击软件兼容性提示中的 *Yes* ，并将 Mech-Center 运行键恢复后即可重新运行。

- 在运行机器人时，请确保人员安全。当发生紧急情况，请按下示教器上的急停按钮！

2.5 标准接口用 Mech-Viz 样例工程

Mech-Center 安装目录 (tool/viz_project) 文件夹内保存有一些用于标准接口的典型应用工程模板。



check_collision

用于视觉引导抓取中的路径规划和碰撞检测。

outer_move

用于机器人需要移动到从外部客户端传入的位姿的场景。

suction_zone

用于通过 DO 信号列表使用多个吸盘分区（或阵列夹具）。

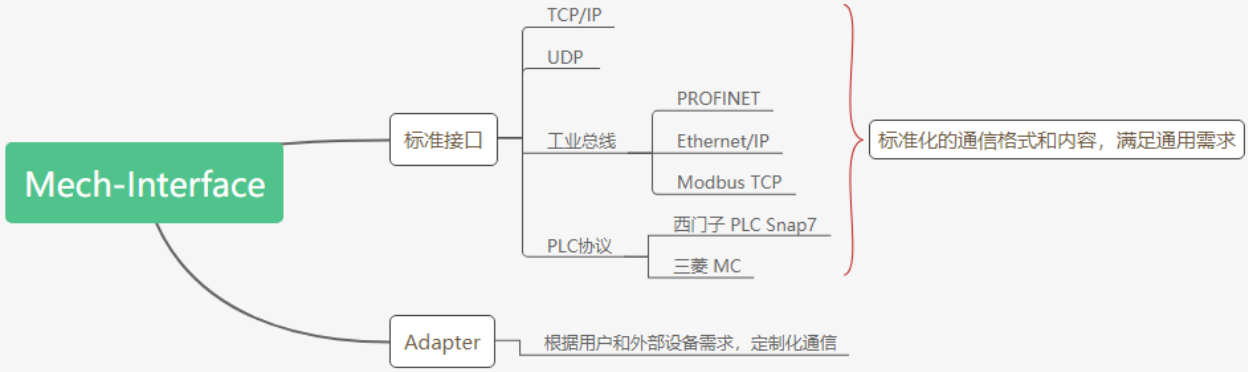
vision_result_reuse

用于视觉引导抓取中需要多次使用同一次返回的视觉结果进行路径规划和碰撞检测的场景。

CHAPTER 3

Mech-Interface

Mech-Interface 是 Mech-Mind 软件系统的对外接口服务，作为与外部通信的桥梁，其作用是接收外部信息、发送系统内部信息，包含 **标准接口**和 **Adapter** 。



本节主要包含如下内容：

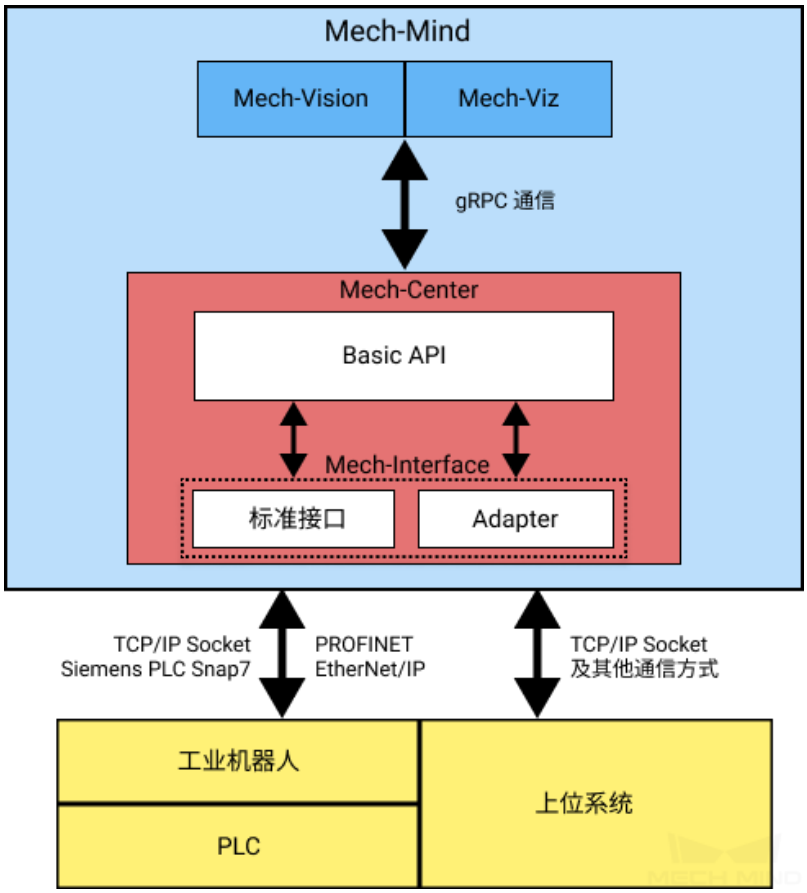
- 概览：介绍 Mech-Interface 两种通信方式的机制和应用场景。
- 标准接口：介绍标准接口的设置与部署。
- 标准接口开发者参考手册：介绍标准接口的通信协议、指令集、错误码以及错误排查。
- *Adapter*：介绍 Adapter 的快速了解、Adapter 生成器手册和 Adapter 编程指南。

3.1 概览

本节主要介绍两种 Mech-Interface 通信方式的机制、比较以及应用场景。

3.1.1 通信机制

两种 Mech-Interface 通信方式的机制如下图所示：



- Adapter 是连接外部通信设备（工业机器人、上位机、PLC）与 Mech-Vision 和 Mech-Viz 软件的 Python 适配程序，对内与 Mech-Vision 和 Mech-Viz 进行通信，对外通过任何 Python 能够实现的通信协议与外部设备进行通信。
- 标准接口是梅卡曼德提供的一套完整的 Adapter 程序，支持多种的通信协议，拥有强大的控制指令集和异常报警系统，可以满足大部分的用户需求。

3.1.2 标准接口和 Adapter 的比较

Adapter 和标准接口都是连接外部设备与 Mech-Vision 和 Mech-Viz 软件的适配程序。标准接口是梅卡曼德提供的一套固定的 Adapter 程序，不支持二次开发。

它们对内通过调用 Basic API 与 Mech-Vision 和 Mech-Viz 进行通信，对外通过特定的通信协议与外部设备进行通信。

标准接口和 Adapter 的详细对比如下表所示。

对比	标准接口	**Adapter**
对内通信	都通过调用 Basic API 与 Mech-Vision 和 Mech-Viz 进行通信。	
对外通信协议	标准接口仅支持使用如下通信协议与外部设备通信：- TCP/IP Socket - UDP - Siemens PLC Snap7 - PROFINET - EtherNet/IP - 三菱 MC - Modbus TCP	Adapter 支持使用任何 Python 支持的通信协议与外部设备通信。
功能	标准接口仅支持提供视觉结果。	Adapter 不仅提供视觉相关的功能，还可以提供 Python 支持的功能，例如界面、数据库和订单系统。
部署难度	标准接口，简单易用，可以快速完成部署。	Adapter 需要人工编写，时间和人工成本较高。
可扩展性	不支持功能扩展。	可以扩展支持更多的通信协议，支持更多的功能。

3.1.3 应用说明

在实际应用场景中，通常要根据外部通信对象、所使用的通信协议以及项目需要的通信功能等来确定使用何种类型的 Mech-Interface。

标准接口和 Adapter 支持的常见通信对象和通信协议如下表所示。

通信对象	通信协议	**Mech-Interface 类型**	说明
机器人	TCP/IP Socket	标准接口	Mech-Interface 作为 TCP/IP Socket 的服务端。
	UDP		Mech-Interface 作为 UDP 的服务端。
	PROFINET		Mech-Interface 作为 PROFINET 从站设备。
	EtherNet/IP		Mech-Interface 作为 EtherNet/IP 从站设备。
	Modbus TCP		Mech-Interface 作为 Modbus TCP 从站设备。
上位机	HTTP	Adapter	适用于集成项目，采用机器人主控方式。
	WebSocket		
	TCP/IP Socket	标准接口	Mech-Interface 作为 TCP/IP Socket 的服务端。
PLC	TCP/IP Socket	标准接口	Mech-Interface 作为 TCP/IP Socket 的服务端。
	Siemens PLC Snap7		Mech-Interface 作为西门子 PLC Snap7 客户端。
	PROFINET		Mech-Interface 作为 PROFINET 从站设备。
	EtherNet/IP		Mech-Interface 作为 EtherNet/IP 从站设备。
	Modbus TCP		Mech-Interface 作为 Modbus TCP 从站设备。
	三菱 MC		Mech-Interface 作为 MC 客户端。

提示：

- 需要使用 Mech-Interface 方式与外部设备通信时，在标准接口能够满足项目需求的情况下推荐使用标准接口；当标准接口不满足项目需求时（例如外部设备使用标准接口不支持的通信协议、项目需要使用标准接口不支持的功能），则需使用 Adapter。
- 标准接口支持的功能列表见“标准接口开发者参考手册”章节的内容。
- Adapter 支持的功能列表见“*Adapter 功能*”章节的内容。

关于标准接口和 Adapter 更详细的信息，您可以继续阅读：

- 标准接口
- *Adapter*

3.2 标准接口

当用户仅需要位姿，不需要用 Mech-Mind 软件系统控制机器人运动时，可以使用标准接口对数据进行传输。标准接口只提供最基础的接口功能，例如发送位姿和任务数据。若想要更多的接口功能，可以选择使用*Adapter*进行定制。标准接口集成在 Mech-Center 软件中，不需要额外生成。如需使用，直接在软件中开启服务即可。

3.2.1 启用 Mech-Interface

在 *部署设置* ▸ *Mech-Interface* 选项卡下勾选 *启用 Mech-Interface*，接口服务类型勾选 *标准接口*。

3.2.2 接口选项、主机地址

用户可根据需要选择外部服务类型。

Siemens PLC Client Siemens PLC Client 需要设置主机 IP 地址及 PLC 的插槽编号、DB 块编号。

TCP Server TCP Server 需选择协议格式：ASCII 或 HEX 。HEX 格式下需要选择 大小端。

TCP Server 需要根据实际情况设置端口号，默认端口为 50000。

用户可以根据机器人端的支持情况，选择可用的通信格式编写接口程序。

机器人类型		标准接口程序样例
工业	ABB	HEX 程序包
	FANUC	HEX 程序包
	KUKA	HEX 程序包
	YASKAWA	ASCII 程序包
	KAWASAKI	ASCII 程序包
	ROKAE	ASCII 程序包
	NACHI	ASCII 程序包
协作	FANUC CRX	Plugin 插件（HEX）
	UR	URCap 插件（ASCII）
	TM	Plug and Play 插件（ASCII）
	JAKA	ASCII 程序包，同时提供 Addon 插件
	ELITE	ASCII 程序包
其他		用户需要编写机器人端程序，Mech-Center 未提供样例

小技巧： 机器人接口样例程序包、操作文档位于软件安装目录下的 Robot_Interface 文件夹中。

PROFINET PROFINET 需要设置机器人型号。

EtherNet/IP EtherNet/IP 需要设置机器人型号。

MODBUS TCP SLAVE Modbus TCP SLAVE 需要设置从站 IP、端口号、设备地址和字节顺序（用户可进行测试选择匹配的字节顺序）。

3.2.3 选择机器人

根据现场情况，单击  来选择对应的机器人品牌与型号。

设置完成后保存并重启 Mech-Center，重启后在界面上单击 *启动接口服务* 即可启用标准接口。

3.2.4 高级设置

高级设置中，可设置单次发送位姿的最大数量、获取 Mech-Viz 数据超时时间、获取 Mech-Vision 数据超时时间。

3.2.5 标准接口 Mech-Viz 示例工程

在 Mech-Center 安装文件夹下的 `tool/viz_project` 中，有四个用于标准接口的示例工程。

Check_collision

视觉引导抓取中的路径规划和碰撞检测。

Outer_move

机器人需要移动到从外部客户端传入的位姿的场景。

Suction_zone

通过 DO 信号控制多吸盘（或阵列夹具）。

Vision_result_reuse

对于视觉引导抓取中需要多次使用一次返回的视觉结果进行路径规划和碰撞检测的场景。

3.3 标准接口开发者参考手册

3.3.1 概览

协议介绍

在标准接口中，外部服务分为以下七类，用户可依据实际需求选择对应的外部服务类型。

1. TCP Server

   Mech-Center 提供一个 TCP Server 作为外部的服务接口。支持字符串和 Hex 字节数据的传输。

2. Siemens PLC Client

   Mech-Center 提供一个基于 SNAP7 协议的 PLC Client 与 Siemens S7 系列 PLC 通信。

3. PROFINET

   Mech-Center 可以作为 PROFINET 从站，连入 PROFINET 工业网络。使用 PROFINET 工业总线进行通信，需要满足的条件包括：

   - 工控机或主机支持安装标准 PCI-e 板卡；
   - 安装 HMS INpact 40 PIR 板卡及 Ixxat VCI 驱动软件；
   - 安装 Mech-Center 1.5.0 及以上，使用软件提供的 GSD 设备描述文件；
   - PROFINET 通信采用的是标准大端数据格式。数据包含 32-bit DINT 位姿数据，PROFINET 主站（特别是机器人控制器）需要支持 32-bit 整数收发。

4. EtherNet/IP

   Mech-Center 可以作为 EtherNet/IP 从站，连入 EtherNet/IP 工业网络。使用 EtherNet/IP 工业总线进行通信，需要满足的条件包括：

   - 工控机或主机支持安装标准 PCI-e 板卡；
   - 安装 HMS INpact 40 EIP 板卡及 Ixxat VCI 驱动软件；
   - 安装 Mech-Center 1.5.1 及以上，使用软件提供的 EDS 设备描述文件；
   - EtherNet/IP 通信采用的是标准大端数据格式。数据包含 32-bit DINT 位姿数据，EtherNet/IP 主站（特别是机器人控制器）需要支持 32-bit 整数收发。

5. Modbus TCP SLAVE

   Mech-Center 可以作为从站设备，提供标准接口选项 MODBUS TCP SLAVE，与主站设备进行数据通信。此功能需要安装 Mech-Center 1.6.1 及以上软件。

6. Mitsubishi MC Client

Mech-Center 可以作为从站设备，提供标准接口选项 Mitsubishi MC Client，与主站设备进行 MC （MELSEC communication）协议通信。此功能需要安装 Mech-Center 1.7.2 及以上软件。

7. UDP Server

   Mech-Center 可以作为服务端，提供标准接口选项 UDP Server，与客户端进行 UDP 协议通信。此功能需要安装 Mech-Center 1.7.2 及以上软件。

指令功能介绍

Mech-Vision 相关

101：启动 Mech-Vision 工程 触发运行 Mech-Vision 工程进行图像采集和视觉数据处理。用于只用 Mech-Vision，不用 Mech-Viz 的场景。

102：获取视觉目标点 读取视觉识别结果，即工件上的目标点。用于只用 Mech-Vision，不用 Mech-Viz 的场景。

103：切换 Mech-Vision 配方 切换 Mech-Vision 工程内保存的配方。用于多种工件识别时，切换不同的工程参数，如识别模板，深度学习模型文件等。

105：获取 Mech-Vision “路径规划”步骤的结果 获取 Mech-Vision 中“路径规划”步骤输出的免碰撞运动路径。

110：从 Mech-Vision 获取自定义输出数据 该指令用于从 Mech-Vision 中的步骤 procedure_out 接收自定义输出数据类型数据（步骤参数“端口类型”设置为“动态”）。每次执行该指令只会从视觉结果中获取一个位姿及其对应的标签、分数等（如有）。如果需要接收多个位姿，请多次执行该指令。

Mech-Viz 相关

201：启动 Mech-Viz 工程 触发运行 Mech-Viz 工程，调用相应的 Mech-Vision 工程，并规划机器人移动类路径。用于既有 Mech-Vision 又有 Mech-Viz 的场景。

202：停止 Mech-Viz 工程 手动终止 Mech-Viz 工程的运行。

203：选择 Mech-Viz 分支 当 Mech-Viz 工程中有 branch_by_msg 步骤时，控制该步骤由指定的端口输出数据。

204：设置移动索引 用于设置 Mech-Viz 工程中移动类步骤的索引参数。包含索引参数的移动类步骤有：按序列移动、按阵列移动、自定义垛型、预设垛型。

205：获取规划路径 用于获取 Mech-Viz 工程规划的机器人移动路径。

206：获取 DO 信号列表 获取 Mech-Viz 计算的吸盘控制信号列表。用于纸箱多抓时吸盘分区控制的场景。

207：读取 Mech-Viz 步骤参数 该指令读取指定步骤的指定参数的值。请在 Mech-Center 中 *部署设置 ▸ Mech-Interface ▸ 高级设置的 属性配置*配置文件指定读取哪些步骤的哪些参数。

208：设置 Mech-Viz 步骤参数 该指令设置指定步骤的指定参数的值。请在 Mech-Center 中 *部署设置 ▸ Mech-Interface ▸ 高级设置的 属性配置*配置文件指定设置哪些步骤的哪些参数，以及设置成什么值。

210：获取移动目标和视觉规划结果 该指令用于从 Mech-Viz 获取规划的单个目标点。目标点可以是视觉移动目标点也可以是其他移动类步骤目标点。目标点可能包含位姿、速度、工具信息、工件信息等。

动态传入数据

501：向 Mech-Vision 传入物体尺寸 该指令用来设置物体 3D 尺寸。用于当 Mech-Vision 工程中存在步骤 read_object_dimensions，需要外部输入物体尺寸（比如箱子长宽高）的场景。

502：向 Mech-Viz 传入 TCP 设置 Mech-Viz 工程中动态变化的移动目标点，Mech-Viz 工程中需要有 outer_move 步骤。

自定义通知消息

601：通知 用户不需要发起该指令。当 Mech-Viz / Mech-Vision 工程运行至“通知”步骤时，Mech-Center 会把步骤中定义的消息发给客户端。

标定

701：标定 用于相机的手眼标定。机器人请求标定点，并触发相机拍照，从而完成整个标定过程，标定点由 Mech-Vision 提供。

系统状态查询

901：获取软件状态 获取 Mech-Mind 软件系统的状态，起到检测作用。

注意： 通信需要统一的数据单位：

- 关节角和欧拉角的单位是度（°）。
- 法兰位姿的 XYZ 坐标单位为毫米（mm）。

样例介绍

有关标准接口的样例，请参考 标准接口。

3.3.2 TCP/IP 指令说明

本文介绍基于 TCP/IP 协议的标准接口指令。

101 指令——启动 Mech-Vision 工程

该指令启动 Mech-Vision 工程，用于执行相机拍照和视觉识别。

如果工程为 Eye In Hand 模式，该指令将把机器人拍照位姿传入工程。

该指令用于仅使用 Mech-Vision 的场景。

发送的指令参数

101, Mech-Vision 工程编号, 预期视觉点数量, 机器人位姿类型, 机器人位姿

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

预期视觉点数量

期望从 Mech-Vision 得到的视觉点数量。视觉点信息包括视觉位姿及对应点云、标签、缩放等。

- 0：从 Mech-Vision 工程取得识别结果中所有的视觉点。
- 大于 0 的整数：从 Mech-Vision 工程取得识别结果中指定数量的视觉点。
  - 如果视觉点总数小于该参数值，则取得识别结果中所有视觉点。
  - 如果视觉点总数大于等于该参数值，则取得该参数指定数量的视觉点。

提示：获取视觉点的指令是 102 指令。在 TCP/IP 中，默认每次执行 102 指令最多可以获取 20 个视觉点。在第一次执行 102 指令后，其中一个返回的参数将体现是否已返回所有请求的视觉点；

如果没有，请重复执行 102 指令。

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Vision，其取值范围为 0~3。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	说明	适用场景
0	0, 0, 0, 0, 0, 0	无需向 Mech-Vision 传入机器人位姿	工程为 Eye To Hand 模式。若 Mech-Vision 工程中使用“路径规划”步骤，则路径规划的起始点为路径规划工具中设置的 Home 点。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的关节角和法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，除桁架机器人外的大多数机器人适用该设定。
2	机器人的当前法兰位姿	需要将机器人的当前法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，机器人无关节角数据，仅有法兰位姿数据（如桁架机器人）。
3	机器人路径规划起始点的关节角	需要将机器人路径规划起始点的关节角传入 Mech-Vision	工程为 Eye To Hand 模式，并且 Mech-Vision 工程中存在“路径规划”步骤，且需要从机器人端设置“路径规划”步骤的起始点。

提示： 位姿由位置和姿态组成，其中位置单位为毫米（mm）；姿态使用欧拉角表示，单位为度（°）。关节角单位为度（°）。

返回的数据参数

101, 状态码

状态码

若指令执行正常，返回 **1102** 状态码；否则返回对应的错误码。

样例

在 Eye In Hand 场景下，指令发送机器人当前关节角和法兰位姿，执行正常的示例如下所示。

```
TCP send string = 101, 1, 0, 1, 5.18, 14.52, 4.03, 0.09, 72.44, 5.15, 549.56, 50.0,␣
↪647.01, 180.0, -1.0, 180.0
TCP received string = 101, 1102
```

在 Eye To Hand 场景下，指令发送示教点的关节角，执行正常的示例如下所示。

```
TCP send string = 101, 1, 0, 3, 5.18, 14.52, 4.03, 0.09, 72.44, 5.15
TCP received string = 101, 1102
```

下面示例为指令执行异常，表示工程编号为 2 的视觉工程未注册。

```
TCP send string = 101, 2, 10, 0
TCP received string = 101, 1011
```

102 指令——获取视觉目标点

该指令用在*101 指令——启动 Mech-Vision 工程* 之后，用于获取 Mech-Vision 输出的视觉点，然后将视觉点转换为视觉目标点。

具体转换过程如下所示，即将视觉点包含的位姿转换为对应机器人 TCP。

- 将视觉点包含的位姿绕 Y 轴旋转 180°。
- 识别对应机器人型号的参考坐标系定义是否涉及机器人基座高度，并相应增加垂直方向的偏置。

提示： 102 指令单次接收视觉目标点的数量上限默认为 20。如需要获取的视觉目标点的数量大于 20，需多次执行该指令来获取所有所需的视觉目标点。

发送的指令参数

102, Mech-Vision 工程编号

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

102, 状态码, 是否发送完成, 视觉目标点数量, 保留字段, 视觉目标点, 视觉目标点, …, 视觉目标点

注解： 视觉目标点数据位于返回的数据参数的结尾（单次最多返回 20 个视觉目标点）。视觉目标点包含 TCP、标签、速度，其中速度值为 0。

状态码

若指令执行正常，则返回 **1100** 状态码；否则返回对应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

是否发送完成

该参数表明是否已取得所有所需的视觉目标点。值为 0 或 1。

- `0`：未取得所有所需的视觉目标点。请重复执行 102 指令直到该参数值为 1。
- `1`：已取得所有所需的视觉目标点。

如果*101 指令* 指定的预期视觉点数量大于 20（收发数据长度的默认值），可通过该参数判断是否还有未发送的视觉目标点。若数据未发送完毕，则可以重复调用 102 指令，继续接收。

提示： 如未取得所有视觉目标点，若此时调用 101 指令重新拍照，则未取得的视觉目标点将被清除。

视觉目标点数量

执行该指令获得的视觉目标点数量。

- 如果请求的视觉目标点数量大于等于 Mech-Vision 识别的视觉点数量，按照 Mech-Vision 识别的视觉点数量发送。
- 如果请求的视觉目标点数量小于 Mech-Vision 识别视觉点数量，按照请求数量发送。

默认范围：0~20。

保留字段

该字段未使用，默认值为 0。

视觉目标点

一个视觉目标点由 8 个数据构成，前 6 个表示 TCP，第 7 个表示标签，第 8 个表示速度。

- **TCP**：TCP 包括三维坐标（XYZ，单位为毫米）及欧拉角（ABC，单位为度）。
- **标签**：位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前用步骤 label_mapping 将标签映射为整数。如工程中没有标签，则该参数为默认值 0。
- **速度**：该参数值默认为 0。Mech-Vision 输出的视觉结果中一般不带机器人目标点速度信息。

样例

指令执行正常

```
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
```

指令执行异常，无视觉结果。

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

请求视觉目标点的样例

以下样例为依次发送的指令参数 101、102、102 来获取 22 个视觉目标点的过程。细节如下：

- TCP/IP 发送 101 指令，内容为 `101, 1, 0, 1, …`，希望获取全部视觉目标点。
- TCP/IP 发送 102 指令，获取 20 个视觉目标点。
- TCP/IP 收到 102 指令返回的数据，内容为 `102, 1100, 0, 20, …`，表示未取得所有视觉目标点。返回的数据包含 20 个视觉目标点。
- TCP/IP 再次发送 102 指令，获取剩余视觉目标点。
- TCP/IP 收到 102 指令返回的数据，内容为 `102, 1100, 1, 2, ...`。返回的数据包含 2 个视觉目标点，且表明所有视觉目标点已发送完成。

发送 101 指令。

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

第一次发送 102 指令，并获取 20 个视觉目标点。

```
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, ...
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
```

第二次发送 102 指令并接收剩余的两个视觉目标点。

```
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 指令——切换 Mech-Vision 配方

切换 Mech-Vision 工程内的参数配方。

Mech-Vision 中步骤的参数设定可通过切换参数配方调整。

参数配方调整涉及的参数通常包括点云匹配模板、图像匹配模板、感兴趣区域、置信度阈值等。

> **注意：** 需要在执行*101 指令——启动 Mech-Vision 工程* 之前使用该指令。

发送的指令参数

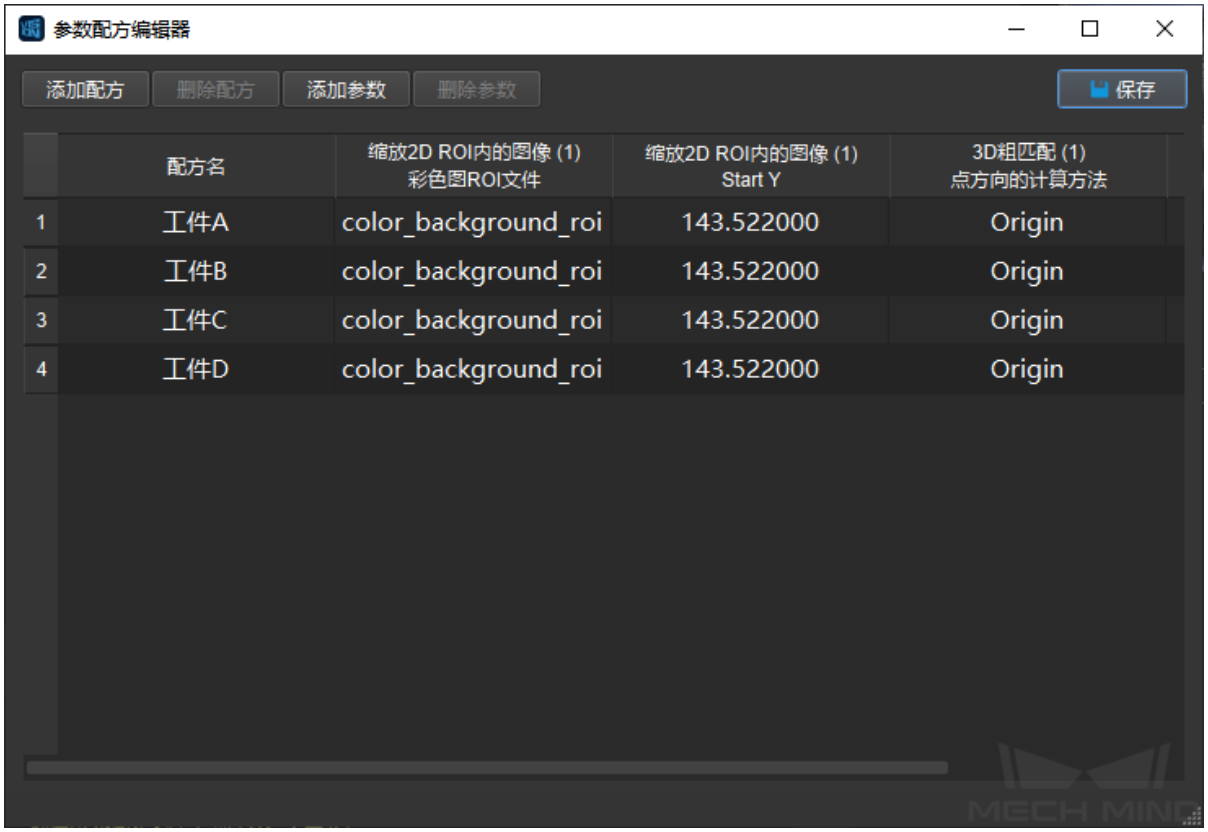
103, Mech-Vision 工程编号, 配方编号

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

配方编号

Mech-Vision 工程中配方模板的编号，为正整数。单击 `工程助手 ▸ 参数配方`，进入参数配方编辑器。编号范围：1~99。



返回的数据参数

103, 状态码

状态码

若指令执行正常，则返回 **1107** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

指令执行异常

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

105 指令——获取 Mech-Vision “路径规划”步骤的结果

在调用 101 指令之后，使用该指令获取 Mech-Vision 中“路径规划”步骤输出的免碰撞抓取路径。

在使用该指令时，Mech-Vision “输出”步骤的 **端口类型**参数需要设置为“预定义（机器人路径）”。

提示： 在调用 105 指令前，请务必将 101 指令的 **预期视觉点数量**设置为 0，以减少调用 105 指令的次数。若 101 指令的 **预期视觉点数量**设置为 1，则每次调用 105 指令只会返回一个路径点，只有多次调用 105 指令才能接收全部路径点。

发送的指令参数

105, Mech-Vision 工程编号, 路径点位姿类型

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

路径点位姿类型

该参数用于指定”路径规划“步骤返回的路径点的位姿类型。

- `1`：路径点的位姿将以机器人关节角（JPs）的形式返回。
- `2`：路径点的位姿将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

105, 状态码, 是否发送完成, 路径点数量, “视觉移动”位置, 路径点, 路径点, …, 路径点

状态码

若指令执行正常，则返回 **1103** 状态码；否则返回对应的错误码。

是否发送完成

- `0`：未发送完路径中的全部路径点。请重复执行该指令，直到该参数值为 1。
- `1`：已发送完路径中全部路径点。

路径点数量

该参数用于表示此次执行该指令返回的路径点个数，取值范围为 0~20。若获取的路径点数多于 20，请多次调用该指令。

“视觉移动”位置

路径规划工具中设置的“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下组成：“移动 _1”，“移动 _2”，“视觉移动”，“移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

路径点

一个路径点由 8 个数据构成，前 6 个表示位姿，第 7 个表示标签，第 8 个表示速度。

- **位姿**：三维坐标（单位为毫米）及欧拉角（单位为度），或 JPs 关节角（单位为度）。类型由 105 指令中的 **路径点类型**决定。
- **标签**：位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在“输出”步骤前使用 “标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。
- **速度**：路径规划工具中设置的速度值。

样例

指令执行正常

```
TCP send string = 105, 1, 2
TCP received string =105,1103,1,5,3,1030.0,0,1260.0,0.0,90.0,-0.0,0,7,1149.114,-298.
↪9656,274.9219,-0.0977,-1.3863,-175.9702,0,7,1149.8416,-296.8585,245.0048,-0.0977,-1.
↪3863,-175.9702,2,7,1149.114,-298.9656,274.9219,-0.0977,-1.3863,-175.9702,0,7,1030.0,
↪0,1260.0,0.0,90.0,-0.0,0,7
```

指令执行异常

```
TCP send string = 105, 1, 2
TCP received string = 105, 1020
```

110 指令——从 Mech-Vision 获取自定义数据

该指令用于从 Mech-Vision 中的 procedure_out 步骤接收自定义数据，即除 poses 和 labels 之外其他端口的数据（procedure_out 步骤参数“端口类型”设置为“自定义”）。

每次执行该指令只会从视觉结果中获取一个位姿及其对应的标签、分数等（如有）。如果需要接收多个位姿，请多次执行该指令。

发送的指令参数

110, Mech-Vision 工程编号

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

110, 状态码, 位姿是否传输完成, 自定义数据项个数, 目标物体位姿对应的机器人工具位姿（TCP）, 标签, 自定义数据项, …, 自定义数据项

状态码

如果没有错误，将返回状态码 1100。否则，将返回相应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

位姿是否传输完成

- `0`：视觉结果中仍有位姿未传输。
- `1`：视觉结果中的所有位姿都已传输。

自定义数据项个数

位姿和标签以外的数据类型的自定义数据项的总数。

目标物体位姿对应的机器人工具位姿（TCP）

机器人坐标系下的工具位姿（TCP，其中三维坐标单位为毫米，欧拉角单位为度）。Mech-Vision 输出目标物体位姿，然后将该位姿转换为机器人工具位姿。procedure_out 步骤必须有一个位姿端口。

目标物体位姿对应的工具位姿通常是通过将目标物体位姿的 Z 轴进行翻转来生成的。

标签

与位姿对应的物体信息标签。标签应该是一个正整数，否则应在 Mech-Vision 工程中使用 label_mapping 步骤将其映射为正整数。如果步骤 procedure_out 上没有标签端口，则该字段填充为 0。

自定义数据项

当 Mech-Vision 工程中 procedure_out 步骤的端口类型为“自定义”时，除 poses 和 labels 之外其他端口的数据。

各自定义数据按照端口名称的字母顺序（A—Z）排列。

201 指令——启动 Mech-Viz 工程

用于既使用 Mech-Vision 又使用 Mech-Viz 的场景。该指令用于运行 Mech-Viz 工程，启动对应的 Mech-Vision 工程，Mech-Viz 基于 Mech-Vision 的视觉结果规划机器人的运动路径。

请在需要启动 Mech-Viz 工程中勾选 *自动加载*。

梅卡曼德软件安装目录下 `Mech-Center\tool\viz_project` 文件夹内保存有一些典型应用工程模板，用户可以在模板的基础上进行修改。

用于标准接口的 Mech-Viz 样例工程描述请见标准接口用 *Mech-Viz* 样例工程。

发送的指令参数

201, 机器人位姿类型, 机器人位姿

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Viz，其取值范围为 0~2。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	机器人位姿参数值	说明	适用场景
0	0, 0, 0, 0, 0, 0	无需向 Mech-Viz 传入机器人位姿，Mech-Viz 中仿真机器人将从初始位姿 JPs = [0, 0, 0, 0, 0, 0] 开始移动到第一个路径点。	工程为 Eye To Hand 模式。不推荐使用该设定。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的当前关节角和法兰位姿传入 Mech-Viz，Mech-Viz 中仿真机器人将从传入的关节角开始移动到第一个路径点。	工程为 Eye In Hand 模式时，推荐使用该设定。
2	机器人端自定义的关节角	需要将机器人的一个示教点（非当前关节角）传入 Mech-Viz，用于在机器人处于拍照区域外时，提前触发 Mech-Viz 工程规划下一轮路径（如下图），Mech-Viz 中仿真机器人将从传入的示教点开始运动到第一个路径点。	工程为 Eye To Hand 模式时，推荐使用该设定。

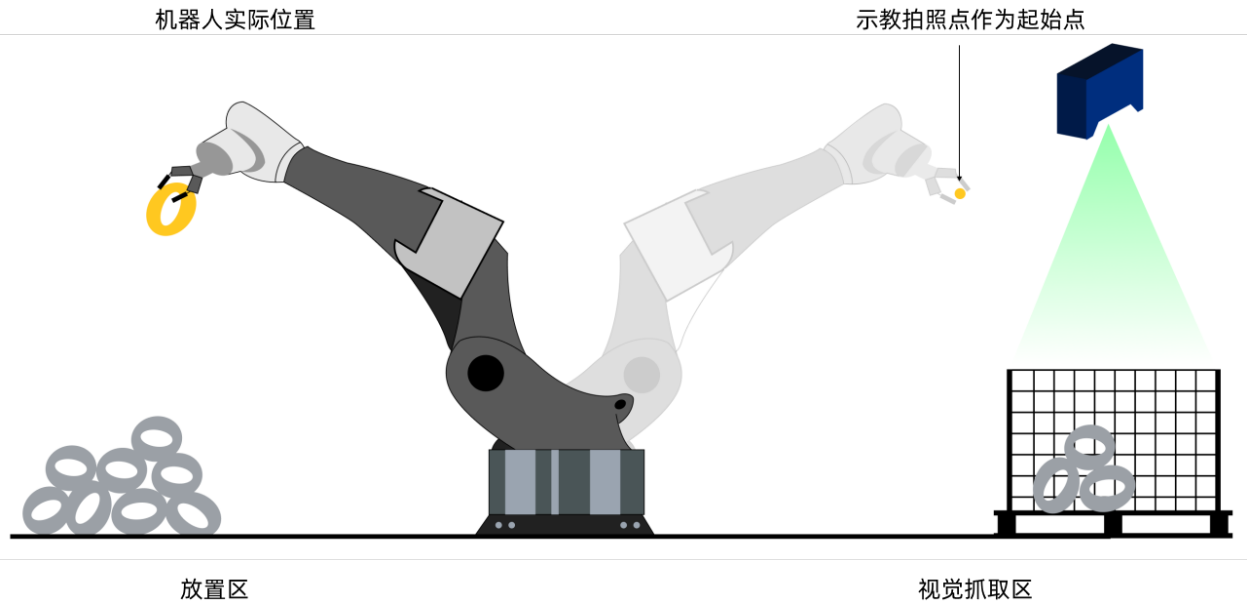
Eye To Hand 模式下应将 **机器人位姿类型**设为 2 的原因：

Eye To Hand 模式下，相机可在机器人回到拍照及抓取区域之前拍照，用于提前规划下一轮抓取路径，从而缩短节拍。

若此时将 **机器人位姿类型**设为 1，即将机器人当前位姿发送给 Mech-Viz 仿真机器人，可能导致仿真机器人与真实机器人轨迹不一致，发生未预知的碰撞，造成危险。

即仿真机器人将直接从当前位姿移动至 Mech-Viz 中第一个移动步骤中设置的位姿，而真实机器人可能在移动至其他位姿后才移动至上述位姿。

所以应将 **机器人位姿类型**参数设为 2。



提示： 位姿由位置和姿态组成，其中位置单位为毫米（mm）；姿态使用欧拉角表示，单位为度（°）。关节角单位为度（°）。

返回的数据参数

201, 状态码

状态码

若指令执行正常，则返回 **2103** 状态码；否则返回对应的错误码。

样例

在 Eye In Hand 场景下，指令发送机器人当前关节角和法兰位姿，执行正常的示例如下所示。

```
TCP send string = 201, 1, 5.18, 14.52, 4.03, 0.09, 72.44, 5.15, 549.56, 50.0, 647.01,␣
↪180.0, -1.0, 180.0
TCP received string = 201, 2103
```

在 Eye To Hand 场景下，指令发送示教点的关节角，执行正常的示例如下所示。

```
TCP send string = 201, 2, 5.18, 14.52, 4.03, 0.09, 72.44, 5.15
TCP received string = 201, 2103
```

在下面示例中，指令未发送所需的位姿数据，因此执行异常。

```
TCP send string = 201, 1
TCP received string = 201, 2013
```

202 指令——停止 Mech-Viz 工程

停止 Mech-Viz 工程的运行。若 Mech-Viz 工程未陷入无限循环或可正常停止，则不需要使用此指令。

发送的指令参数

202

返回的数据参数

202, 状态码

状态码

若指令执行正常，返回 **2104** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

203 指令——选择 Mech-Viz 分支

该指令用于指定工程将沿哪个分支运行。分支机制通过 branch_by_msg 步骤建立，该指令则通过指定该步骤的出口来指定分支。

在执行该指令前请先执行*201 指令——启动 Mech-Viz 工程* 。

Mech-Viz 工程运行至 branch_by_msg 步骤时，将等待该指令指定出口。

发送的指令参数

203, 分支步骤编号, 出口号

分支步骤编号

该参数用于指定分支选择将在哪个 branch_by_msg 步骤上进行。

该参数应为正整数，即 branch_by_msg 的步骤编号。步骤编号可在步骤参数中读取。

出口号

该参数用于指定工程将沿 branch_by_msg 步骤的哪个出口运行，Mech-Viz 程序将沿该出口继续执行。参数为正整数。

提示:

- 出口号为 Mech-Viz 显示的端口号 + 1。如端口号为 0，则出口号为 1。
- 出口号为从 1 开始的端口索引号。比如指定的出口为从左往右数第二个端口，那么出口号为 2。

返回的数据参数

203, 状态码

状态码

若指令执行正常，则返回 **2105** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

指令执行异常

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

204 指令——设置移动索引

该指令用于设定步骤的索引参数值。该类步骤一般用于连续或单独指定的移动或其他操作。

带有索引参数的步骤包括“按序列移动”、“按阵列移动”、“自定义垛型”、“预设垛型”等。

在执行该指令前，请先执行*201 指令——启动 Mech-Viz 工程* 。

发送的指令参数

204, 步骤编号, 索引值

步骤编号

该参数指定哪个步骤需要索引参数设定。

该参数值应为正整数，即待索引步骤的步骤编号。步骤编号可在步骤参数中读取。

索引值

下次执行此步骤时应设置的索引值。

发送该指令时，Mech-Viz 中的当前索引值将变为该参数值减 1。

当 Mech-Viz 项目运行到该指令指定的步骤时，Mech-Viz 中的当前索引值将增加 1，成为该参数的值。

返回的数据参数

204, 状态码

状态码

若指令执行正常，则返回 **2106** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

指令执行异常

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

205 指令——获取规划路径

在执行*201 指令——启动 Mech-Viz 工程* 后，该指令用于获取 Mech-Viz 规划的路径。

在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

> **注解：** 如果工程中某个移动类步骤的路径点不应发送给机器人，请取消勾选该步骤参数中的“发送路径点”选项。

发送的指令参数

205, 路径点类型

路径点类型

该参数用于指定 Mech-Viz 将返回何种形式的路径点。

- `1`：路径点将以机器人关节角（JPs）的形式返回。
- `2`：路径点将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

205, 状态码, 是否发送完成, 路径点数量, “视觉移动”步骤位置, 路径点, 路径点, …, 路径点

状态码

若指令执行正常，则返回 **2100** 状态码；否则返回对应的错误码。

提示： 调用该指令时，若 Mech-Viz 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

是否发送完成

- `0`：未发送完路径中的全部路径点。请重复执行该指令，直到该参数值为 1。
- `1`：已发送完路径中全部路径点。

路径点数量

该参数表示返回的路径点个数，取值范围为 0~20。在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

“视觉移动”位置

“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下步骤组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

路径点

一个路径点由 8 个数据构成，前 6 个表示位姿，第 7 个表示标签，第 8 个表示速度。

- **位姿：**三维坐标（单位为毫米）及欧拉角（单位为度），或 JPs 关节角（单位为度）。类型由 205 指令中的路径点位姿类型决定。
- **标签：**位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前使用步骤 label_mapping 将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。
- **速度：**移动类步骤参数中的非零速度参数百分数值。

样例

指令执行正常

```
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
```

指令执行异常

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

206 指令——获取 DO 信号列表

该指令用于获取规划的 DO 信号列表。DO 信号列表用于控制多个工具或吸盘分区。

执行该指令前，需要先执行*205 指令* 获取 Mech-Viz 规划路径。

请根据模板工程来部署 Mech-Viz 工程，并在工程中设置对应的吸盘配置文件。模板工程位于梅卡曼德软件安装目录下 `Mech-Center\tool\viz_project\suction_zone` 文件夹。

在工程的 set_do_list 步骤的参数中：

- 在“接收对象”下勾选“标准接口”
- 勾选“从视觉移动中获取 DO 列表”
- 在参数栏底部选择需要 DO 信号列表的视觉移动步骤

发送的指令参数

206

返回的数据参数

206, 状态码, DO 信号值, DO 信号值, ···, DO 信号值

状态码

若指令执行正常，则返回 **2102** 状态码；否则返回对应的错误码。

DO 信号值

返回的数据参数末尾共有 64 个 DO 信号值，为整数。

DO 信号值范围：0~999。

占位值：-1。

样例

指令执行正常，获得 DO 信号值为 11 和 12。

```
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, -1, -1, -1
```

指令执行异常，未获得任何 DO 信号。

```
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, -1, -1, -1
```

207 指令——读取 Mech-Viz 步骤参数

该指令读取指定步骤的指定参数的值。

请在 Mech-Center 中 部署设置 ▸ *Mech-Interface* ▸ 高级设置的 属性配置配置文件指定读取哪些步骤的哪些参数。

发送的指令格式

207, 配置编号（config ID）

配置文件内容格式

read, config ID, 步骤编号, 参数键名

配置文件中以“#”为开头的内容为批注行，在执行该指令时不起作用。

read

字符串“read”表示该指令用于 读取参数值。

config ID

用于指定读取操作 ID 的正整数。一个 config ID 只能用于读取一个参数。如果读取多个参数，请使用不同的 config ID。

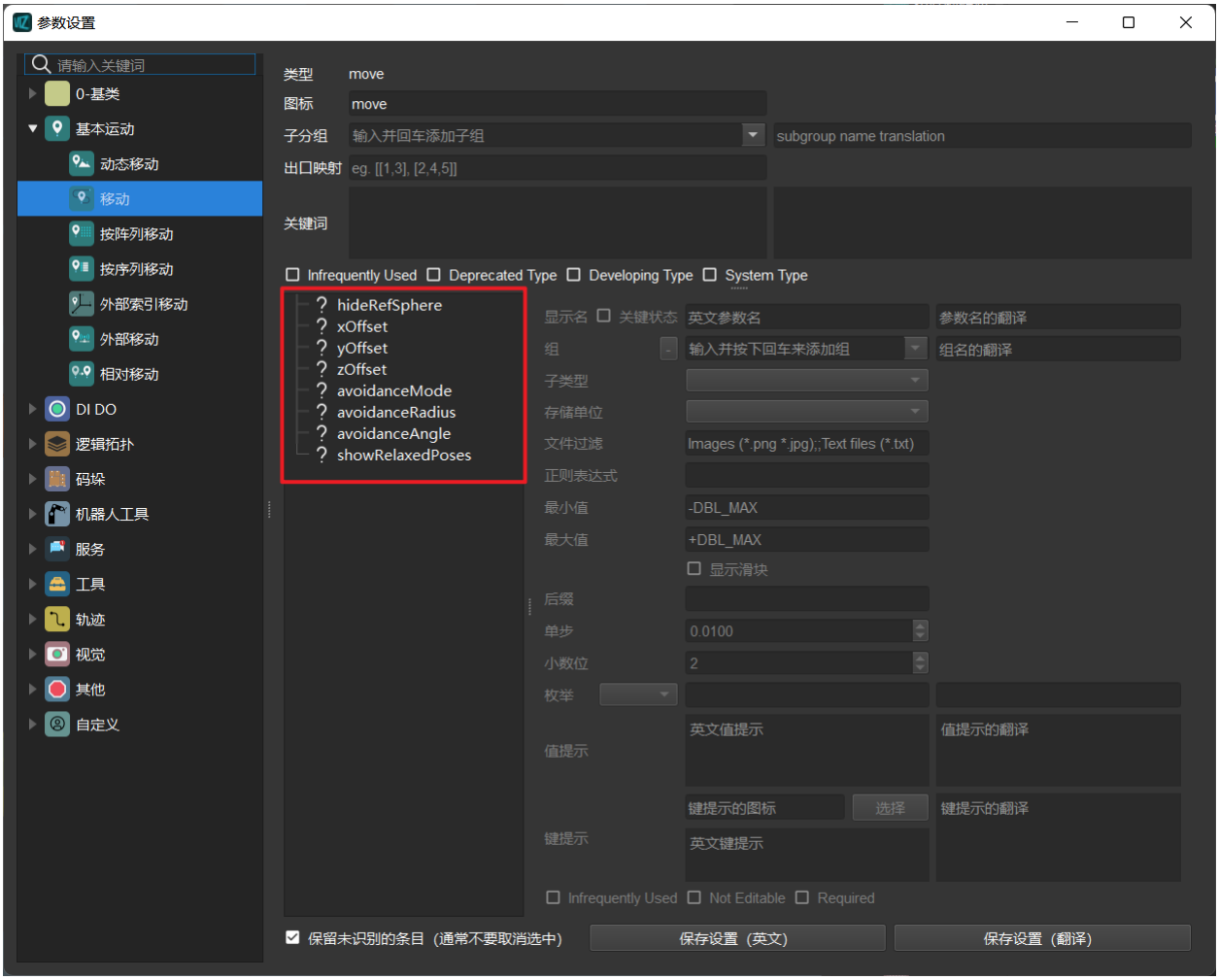
步骤编号

要读取的参数所属于的步骤的编号。

可以在步骤参数中找到步骤编号并设置。

参数键名

要读取的参数的键名，如下图所示。请通过 Mech-Viz 顶部菜单栏 设置 ▸ 组件属性设置打开以下窗口。



配置文件内容样例

```
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
```

返回的数据参数

207, 状态码, 步骤参数值

状态码

如果未发生错误，则返回状态码 2109。否则，将返回相应的错误码。

步骤参数值

Mech-Viz 工程中指定步骤的指定参数的值。

支持的数据类型：INT、FLOAT、STRING。

样例

指令执行正常，成功读取参数值。

```
TCP send string = 207, 1
TCP received string = 207, 2109, 10
```

指令执行异常，无法找到对应的参数名称。

```
TCP send string = 207, 1
TCP received string = 207, 2041
```

208 指令——设置 Mech-Viz 步骤参数

该指令设置指定步骤的指定参数的值。

请在 Mech-Center 中 部署设置 ▸ *Mech-Interface* ▸ 高级设置的 属性配置配置文件指定设置哪些步骤的哪些参数，以及设置成什么值。

发送的指令格式

208, 配置编号（config ID）

配置文件内容格式

write, config ID, 步骤编号, 参数键名, 值

write

字符串“write”表示该指令用于 设置参数值。

config ID

操作 ID。一个 config ID 可用于设置多个参数。

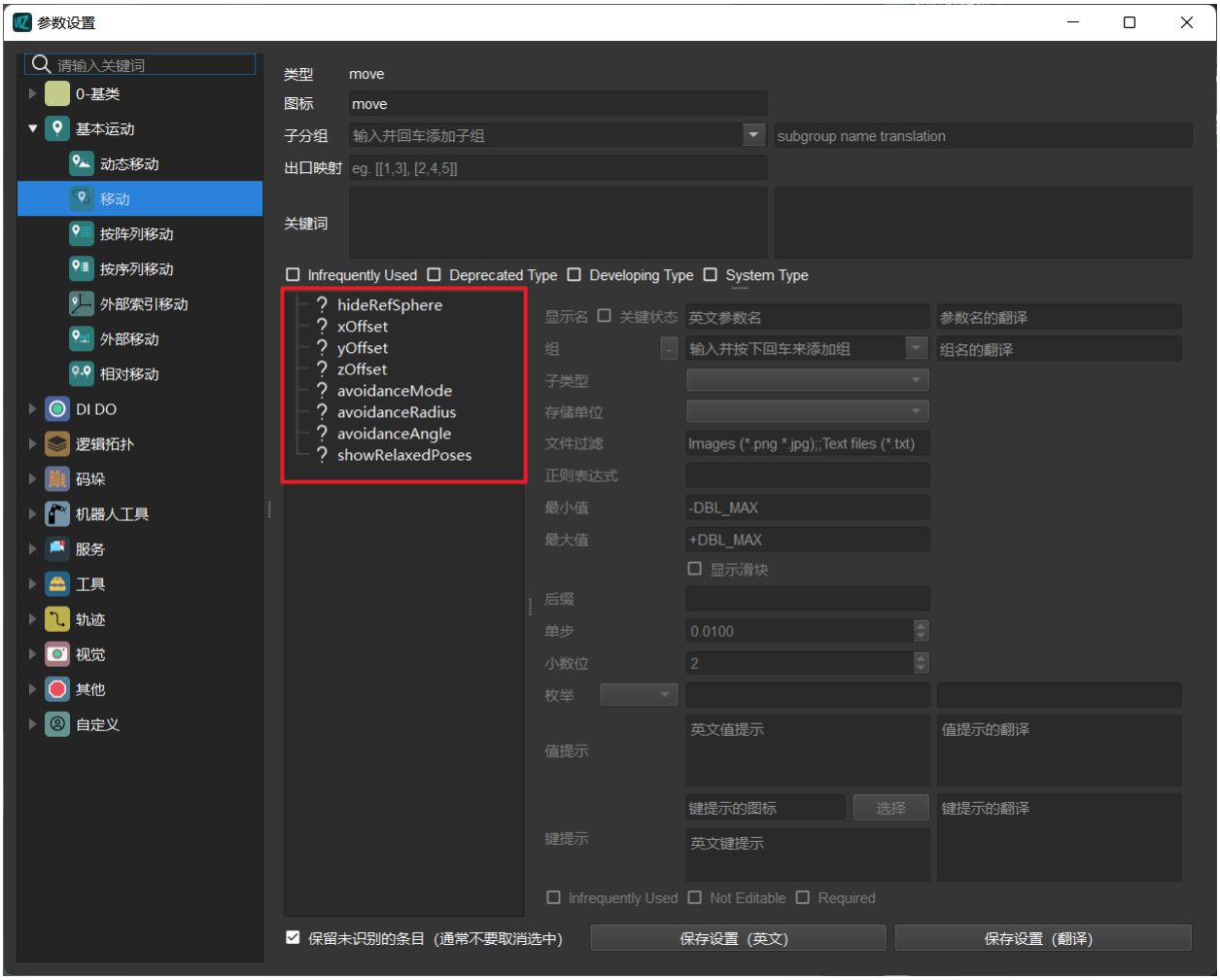
步骤编号

设置的参数所属于的步骤的编号。

步骤编号可以在步骤参数中找到并设置。

参数键名

要设置的参数的键名，如下图所示。请通过 Mech-Viz 顶部菜单栏 设置 ▸ 组件属性设置打开以下窗口。



值

指定参数应设置为的值。

除了数字，还支持字符串值。

配置文件内容样例

```
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
```

样例

指令执行正常，成功设置参数。

```
TCP send string = 208, 1
TCP received string = 208, 2108
```

指令执行异常，无法找到对应的配置编号。

```
TCP send string = 208, 10
TCP received string = 208, 3008
```

210 指令——获取单个路径点和视觉规划结果

该指令用于从 Mech-Viz 获取单个路径点。路径点可以是一般移动路径点，也可以是视觉移动路径点。路径点可能包含位姿、速度、工具信息、工件信息等。

执行该指令得到的路径点可以是以下三种之一。

1. 除了 visual_move 之外的一般移动路径点，其信息包括运动类型（关节运动或直线运动）、工具编号、速度。
2. visual_move 路径点，其信息包括标签、已抓取的工件总数、本次抓取的工件数量、吸盘边角号、TCP 偏移量、工件朝向、工件组尺寸。
3. 包含自定义数据的 visual_move 路径点，此时 Mech-Vision 工程 procedure_out 步骤的端口类型需设置为“自定义”。

注意:

- 通常该指令用于工件为箱子的项目。
- 执行该指令一次只能得到一个路径点。如果希望获取多个路径点，请多次执行该指令。

发送的指令参数

210, 预期的返回数据格式

预期的返回数据格式

以下是四种预期的返回数据格式说明。

预期的返回数据格式参数值	**预期的返回数据**说明（每个字段的具体解释参见下文）
1	位姿（JPs 形式）, 运动类型, 工具编号, 速度, 自定义数据项个数, 自定义数据项 1, ⋯, 自定义数据项 N
2	位姿（TCP 形式）, 运动类型, 工具编号, 速度, 自定义数据项个数, 自定义数据项 1, ⋯, 自定义数据项 N
3	位姿（JPs 形式）, 运动类型, 工具编号, 速度, 视觉规划结果, 自定义数据项个数, 自定义数据项 1, ⋯, 自定义数据项 N
4	位姿（TCP 形式）, 运动类型, 工具编号, 速度, 视觉规划结果, 自定义数据项个数, 自定义数据项 1, ⋯, 自定义数据项 N

返回的数据参数

*210, 状态码, 路径点传输完成状态, 路径点类型, 位姿, 运动类型, 工具编号, 速度, 视觉规划结果 *, 自定义数据项个数, 自定义数据项 **

> **注意：** 是否有 **视觉规划结果**和/或 **自定义数据项**取决于执行指令时发送的 **预期数据格式**参数值。

状态码

如未发生错误，将返回状态码 2100。否则，将返回相应的错误码。

路径点传输完成状态

- `0`：仍有路径点未传输。
- `1`：所有路径点都已传输。

路径点类型

- `0`：不是 visual_move 路径点。
- `1`：是 visual_move 路径点。

位姿

路径点的位姿可以是机器人关节角（JPs，单位为度）或工具位姿（TCP，其中三维坐标单位为毫米，欧拉角单位为度），其形式取决于发送的指令参数。

运动类型

- `1`：关节运动，MOVEJ
- `2`：直线运动，MOVEL

工具编号

路径点的工具编号。-1 表示不使用工具。

速度

路径点的速度百分比值，其值为 Mech-Viz 工程中该路径点对应移动类步骤的参数中设置的速度乘以 Mech-Viz 中设置的全局速度，单位为%。

视觉规划结果

路径中的规划结果信息（如果路径点对应移动步骤为 visual_move）。通常用于纸箱多拣、卸垛等。信息包括：

- 标签：由 10 个正整数组成，默认为 10 个 0。
- 已抓取的工件总数。
- 本次抓取的工件数量。
- 吸盘边角号：用于指定工件靠近吸盘的哪个边角。在 Mech-Viz 的“工程资源”中双击对应的末端工具名称，然后单击 *控制逻辑配置窗口*，便可查看吸盘边角号。
- 工具位姿（TCP）偏移量：从对应工件中心的 TCP 到实际 TCP 的偏移量。
- 工件朝向：工件坐标系 X 轴相对于 TCP X 轴的方向。
- 工件组尺寸。

自定义数据项个数

当 Mech-Vision 工程中 procedure_out 步骤的端口类型为“自定义”时，除 poses 和 labels 之外其他端口输出的数据个数。

自定义数据项

当 Mech-Vision 工程中 procedure_out 步骤的端口类型为“自定义”时，除 poses 和 labels 之外其他端口的数据。

各自定义数据项按照端口名称的字母顺序（A—Z）排列。

501 指令——向 Mech-Vision 中传入物体尺寸

该参数用于往 Mech-Vision 工程中动态传入物体尺寸。启动 Mech-Vision 工程前请先确认物体尺寸。

Mech-Vision 工程中应有 read_object_dimensions 步骤。该步骤参数 **从参数读取物体尺寸**应设定为 `True`。

发送的指令参数

501, Mech-Vision 工程编号, 长, 宽, 高

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

长，宽，高

传入 Mech-Vision 工程的物体尺寸。尺寸值将被 read_object_dimensions 步骤读取。

单位：毫米（mm）

返回的数据参数

501, 状态码

状态码

若指令执行正常，则返回 **1108** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

指令执行异常，错误码 3002，缺少“高”的值。

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

502 指令——向 Mech-Viz 中传入 TCP

该指令用于往 Mech-Viz 工程中动态传入机器人 TCP。读取机器人 TCP 的步骤为 outer_move。

请基于模板工程部署 Mech-Viz 工程。模板工程位于梅卡曼德软件安装目录下 `Mech-Center\tool\viz_project\outer_move` 文件夹。

请将 outer_move 步骤放在工作流程中合适的位置。

该指令需在执行*201 指令——启动 Mech-Viz 工程* 之前执行。

发送的指令参数

502, 机器人 TCP

机器人 TCP

用于设定 outer_move 步骤的机器人 TCP 数据。

返回的数据参数

502，状态码

状态码

若指令执行正常，则返回 **2107** 状态码；否则返回对应的错误码。

样例

指令执行正常

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

601 指令——通知

用户不需要发起该指令。

当 Mech-Viz / Mech-Vision 工程运行至 notify 步骤/ notify_vision 步骤时，Mech-Center 会把“通知”中定义的消息发给客户端。

在 Mech-Vision 中，notify_vision 步骤需命名为“Standard Interface Notify”。

在 Mech-Viz 中，在 notify 步骤的参数中，请勾选“接收对象”下的“标准接口”。

发送的指令参数

无。

返回的数据参数

601, 自定义通知内容

自定义通知内容

notify 步骤/ notify_vision 步骤中定义的通知消息。该消息需为整数。

样例

若 notify 步骤/ notify_vision 步骤定义的通知消息为“1000”，工程运行至该步骤/步骤时会发出该通知。

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

701 指令——标定

该指令用于手眼标定（相机外参标定）。

该指令会与 Mech-Vision 同步标定状态，并从 Mech-Vision 处获得机器人需要到达的标定点。

该指令需要运行多次以完成标定。

发送的指令参数

701, 标定状态, 法兰位姿, JPs 关节角

标定状态

参数范围：0~2。

- 0：通知 Mech-Vision 开始标定流程。
- 1：上一标定点已接收，并发给机器人。
- 2：上一标定点接收失败。

法兰位姿

机器人当前法兰位姿。位姿由位置和姿态组成，其中位置单位为毫米（mm）；姿态使用欧拉角表示，单位为度（°）。

JPs 关节角

机器人当前关节角，单位为度（°）。

提示：法兰位姿与 **JPs 关节角**只需指定其中一个参数值即可，另一个参数值使用 6 个 0 填充。

返回的数据参数

701, 状态码, 标定状态, 下一标定点法兰位姿, 下一标定点 JPs

状态码

若指令执行正常，返回 **7101** 状态码，否则返回对应的错误码。

标定状态

- 0：标定进行中。
- 1：标定完成。

下一标定点法兰位姿

机器人移动的下一标定点位姿数据。机器人程序端可以选择使用法兰位姿或者 JPs 位姿。

下一标定点关节角

机器人需移动至的下一标定点 JPs。

提示： JPs 与法兰位姿选其一即可。

样例

开始标定流程

```
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, -74.3374, 1334.34094, -3128.422, 179.
↪2412, -91.11236, 93.28109, -12.0273, -32.8811, -0.37183, -68.41364, 27.02411
```

从 Mech-Vision 获取标定点（该过程需重复多次以获得所有标定点）

```
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
```

提示： 该过程需重复多次以获得多个标定点。

结束标定

```
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 指令——获取软件状态

该指令用于检查软件运行状态（Mech-Vision、Mech-Viz、Mech-Center）。

目前该指令只支持检查 Mech-Vision 是否可以开始运行工程。

发送的指令参数

901

返回的数据参数

901, 状态码

状态码

> 系统自检状态。**1101** 状态码为“Mech-Vision 工程已就绪”；其他状态码为“Mech-Vision 工程未就绪”。目前该指令只能用于检查 Mech-Vision 工程是否已就绪。

样例

Mech-Vision 工程已就绪

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

Mech-Vision 工程未就绪

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

提示： 执行该指令前，请确认已 自动加载当前方案或 自动加载当前工程。

3.3.3 Siemens PLC 指令说明

本文介绍基于 Siemens PLC Snap 7 协议的标准接口指令。

101 指令——启动 Mech-Vision 工程

该指令启动 Mech-Vision 工程，用于执行相机拍照和视觉识别。

如果工程为 Eye In Hand 模式，该指令将把机器人拍照位姿传入工程。

该指令用于仅使用 Mech-Vision 的场景。

发送的指令参数

参数	DB 偏移量
指令码 101	2.0
Mech-Vision 工程编号	8.0
预期视觉点数量	6.0
机器人位姿类型	4.0
机器人位姿	12.0（JPs）或 36.0（法兰位姿）

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

预期视觉点数量

期望从 Mech-Vision 得到的视觉点数量。视觉点信息包括视觉位姿及对应点云、标签、缩放等。

- 0：从 Mech-Vision 工程取得识别结果中所有的视觉点。
- 大于 0 的整数：从 Mech-Vision 工程取得识别结果中指定数量的视觉点。
  - 如果视觉点总数小于该参数值，则取得识别结果中所有视觉点。
  - 如果视觉点总数大于等于该参数值，则取得该参数指定数量的视觉点。

提示： 获取视觉点的指令是 102 指令。默认每次执行 102 指令最多可以获取 20 个视觉点。在第一次执行 102 指令后，其中一个返回的参数将体现是否已返回所有请求的视觉点；如果没有，请重复执行 102 指令。

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Vision，其取值范围为 0~3。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	说明	适用场景
0	0, 0, 0, 0, 0, 0	无需向 Mech-Vision 传入机器人位姿	工程为 Eye To Hand 模式。若 Mech-Vision 工程中使用“路径规划”步骤，则路径规划的起始点为路径规划工具中设置的 Home 点。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的关节角和法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，除桁架机器人外的大多数机器人适用该设定。
2	机器人的当前法兰位姿	需要将机器人的当前法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，机器人无关节角数据，仅有法兰位姿数据（如桁架机器人）。
3	机器人路径规划起始点的关节角	需要将机器人路径规划起始点的关节角传入 Mech-Vision	工程为 Eye To Hand 模式，并且 Mech-Vision 工程中存在“路径规划”步骤，且需要从机器人端设置“路径规划”步骤的起始点。

机器人位姿为 Real 类型数字。

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，返回 **1102** 状态码；否则返回对应的错误码。

102 指令——获取视觉目标点

该指令用在*101 指令——启动 Mech-Vision 工程* 之后，用于获取 Mech-Vision 输出的视觉点，然后将视觉点转换为视觉目标点。

具体转换过程如下所示，即将视觉点包含的位姿转换为对应机器人 TCP。

- 将视觉点包含的位姿绕 Y 轴旋转 180°。
- 识别对应机器人型号的参考坐标系定义是否涉及机器人基座高度，并相应增加垂直方向的偏置。

提示： 102 指令单次接收视觉目标点的数量上限默认为 20。如需要获取的视觉目标点的数量大于 20，需多次执行该指令来获取所有所需的视觉目标点。

发送的指令参数

参数	DB 偏移量
指令码 102	2.0
Mech-Vision 工程编号	8.0

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

参数	DB 偏移量
状态码	200.0
数据传输状态	202.0
视觉目标点数量	204.0
保留字段	/
此次获取的所有 TCP	208.0
此次获取的所有标签	1168.0

状态码

若指令执行正常，则返回 **1100** 状态码；否则返回对应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的视觉目标点。

`1`：表示返回的数据是新的视觉目标点，请读取。

> **注意：** 在读取新返回的数据后，请将该参数重置为 0。

视觉目标点数量

执行该指令获得的视觉目标点数量。

- 如果请求的视觉目标点数量大于等于 Mech-Vision 识别的视觉点数量，按照 Mech-Vision 识别的视觉点数量发送。
- 如果请求的视觉目标点数量小于 Mech-Vision 识别视觉点数量，按照请求数量发送。

保留字段

该字段未使用，默认值为 0。

此次获取的所有 TCP

单个 TCP 所含信息包括空间坐标（XYZ）和欧拉角（ABC）。

此次获取的所有标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前用步骤 label_mapping 将标签映射为整数。如工程中没有标签，则该参数为默认值 0。

103 指令——切换 Mech-Vision 配方

切换 Mech-Vision 工程内的参数配方。

Mech-Vision 中步骤的参数设定可通过切换参数配方调整。

参数配方调整涉及的参数通常包括点云匹配模板、图像匹配模板、感兴趣区域、置信度阈值等。

> **注意：** 需要在执行*101 指令——启动 Mech-Vision 工程* 之前使用该指令。

发送的指令参数

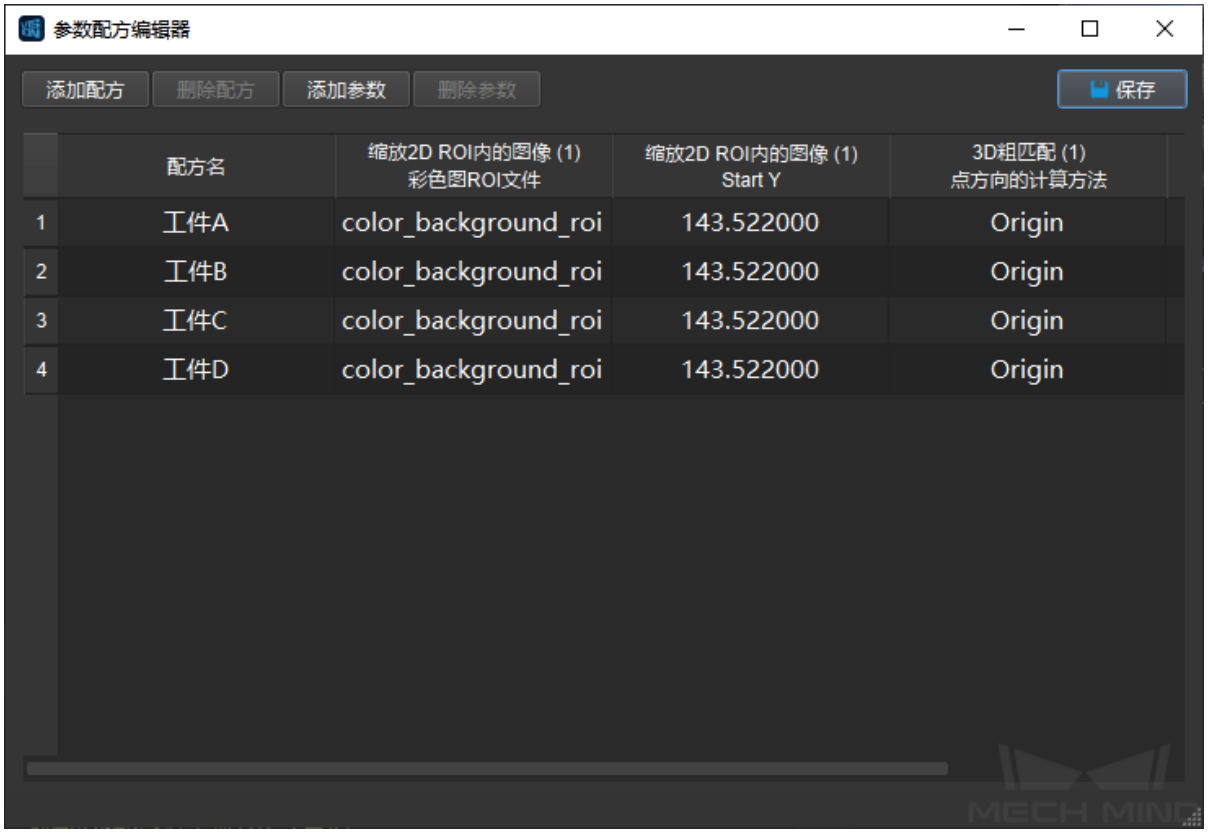
参数	DB 偏移量
指令码 103	2.0
Mech-Vision 工程编号	8.0
配方编号	10.0

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

配方编号

Mech-Vision 工程中配方模板的编号，为正整数。单击 `工程助手 ▸ 参数配方`，进入参数配方编辑器。编号范围：1~99。



返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **1107** 状态码；否则返回对应的错误码。

105 指令——获取 Mech-Vision “路径规划”步骤的结果

在调用 101 指令之后，使用该指令获取 Mech-Vision 中“路径规划”步骤输出的免碰撞抓取路径。

在使用该指令时，Mech-Vision “输出”步骤的 **端口类型**参数需要设置为“预定义（机器人路径）”。

提示：在调用 105 指令前，请务必将 101 指令的 **预期视觉点数量**设置为 0，以减少调用 105 指令的次数。若 101 指令的 **预期视觉点数量**设置为 1，则每次调用 105 指令只会返回一个路径点，只有多次调用 105 指令才能接收全部路径点。

发送的指令参数

参数	DB 偏移量
指令码 105	2.0
Mech-Vision 工程编号	8.0
路径点位姿类型	4.0

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

路径点位姿类型

该参数用于指定”路径规划“步骤返回的路径点的位姿类型。

- `1`：路径点的位姿将以机器人关节角（JPs）的形式返回。
- `2`：路径点的位姿将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	DB 偏移量
状态码	200.0
数据传输状态	202.0
路径点数量	204.0
“视觉移动”位置	206.0
此次发送的所有路径点的位姿	208.0
此次发送的所有路径点的标签	1168.0
此次发送的所有路径点的速度	1248.0

状态码

若指令执行正常，则返回 **1103** 状态码；否则返回对应的错误码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

`1`：表示返回的数据是新的路径点，请读取。

> **注意：**在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数用于表示此次执行该指令返回的路径点个数，取值范围为 0~20。若获取的路径点数多于 20，请多次调用该指令。

“视觉移动”位置

路径规划工具中设置的“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下组成：“移动 _1”，“移动 _2”，“视觉移动”，“移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 105 指令中的路径点位姿类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在“输出”步骤前使用“标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

路径规划工具中设置的速度值。

110 指令——从 Mech-Vision 获取自定义输出数据

该指令用于从 Mech-Vision 中的 procedure_out 步骤接收自定义类型数据，即除 poses 端口和 labels 端口之外的其他端口输出的数据（procedure_out 步骤的“端口类型”参数需设置为“自定义”）。

发送的指令参数

参数	DB 偏移量
指令码 110	2.0
Mech-Vision 工程编号	8.0

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

参数	DB 偏移量
状态码	200.0
数据传输状态	202.0
此次获取的所有 TCP	208.0
此次获取的所有标签	1168.0
自定义端口数据	1456.0

状态码

如果没有错误，将返回状态码 1100。否则，将返回相应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的 TCP。

1：表示返回的数据是新的 TCP，请读取。

此次获取的所有 TCP

单个 TCP 所含信息包括空间坐标（XYZ）和欧拉角（ABC）。

此次获取的所有标签

与位姿对应的物体信息标签。标签应该是一个正整数，否则应在 Mech-Vision 工程中使用 label_mapping 步骤将其映射为正整数。如果步骤 procedure_out 上没有标签端口，则该字段填充为 0。

自定义数据项

procedure_out 步骤上自定义数据类型对应端口的自定义数据。

自定义数据参数按端口名称的字母顺序 A-Z 排列。

201 指令——启动 Mech-Viz 工程

用于既使用 Mech-Vision 又使用 Mech-Viz 的场景。该指令用于运行 Mech-Viz 工程，启动对应的 Mech-Vision 工程，Mech-Viz 基于 Mech-Vision 的视觉结果规划机器人的运动路径。

请在需要启动 Mech-Viz 工程中勾选 *自动加载*。

Mech-Center 安装目录 (tool/viz_project) 文件夹内保存有一些典型应用工程模板，用户可以在模板的基础上进行修改。

用于标准接口的 Mech-Viz 样例工程描述请见*标准接口用 Mech-Viz 样例工程*。

发送的指令参数

参数	DB 偏移量
指令码 201	2.0
机器人位姿类型	4.0
机器人位姿	12.0（JPs）或 36.0（法兰位姿）

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Viz，其取值范围为 0~2。

- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	**说明**	**适用场景**
0	0, 0, 0, 0, 0, 0	无需向 Mech-Viz 传入机器人位姿，Mech-Viz 中仿真机器人将从初始位姿 JPs = [0, 0, 0, 0, 0, 0] 开始移动到第一个路径点。	工程为 Eye To Hand 模式。不推荐使用该设定。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的当前关节角和法兰位姿传入 Mech-Viz，Mech-Viz 中仿真机器人将从传入的关节角开始移动到第一个路径点。	工程为 Eye In Hand 模式时，推荐使用该设定。
2	机器人端自定义的关节角	需要将机器人的一个示教点（非当前关节角）传入 Mech-Viz，用于在机器人处于拍照区域外时，提前触发 Mech-Viz 工程规划下一轮路径（如下图），Mech-Viz 中仿真机器人将从传入的示教点开始运动到第一个路径点。	工程为 Eye To Hand 模式时，推荐使用该设定。

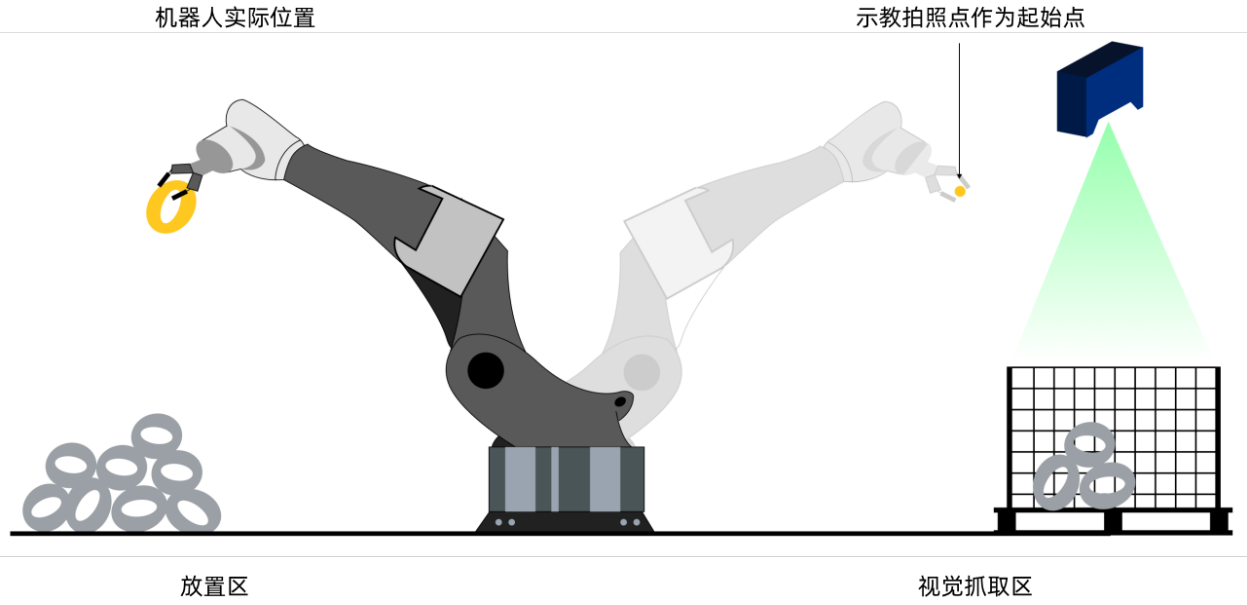
Eye To Hand 模式下应将 **机器人位姿类型**设为 2 的原因：

Eye To Hand 模式下，相机可在机器人回到拍照及抓取区域之前拍照，用于提前规划下一轮抓取路径，从而缩短节拍。

若此时将 **机器人位姿类型**设为 1，即将机器人当前位姿发送给 Mech-Viz 仿真机器人，可能导致仿真机器人与真实机器人轨迹不一致，发生未预知的碰撞，造成危险。

即仿真机器人将直接从当前位姿移动至 Mech-Viz 中第一个移动步骤中设置的位姿，而真实机器人可能在移动至其他位姿后才移动至上述位姿。

所以应将 **机器人位姿类型**参数设为 2。



返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **2103** 状态码；否则返回对应的错误码。

202 指令——停止 Mech-Viz 工程

停止 Mech-Viz 工程的运行。若 Mech-Viz 工程未陷入无限循环或可正常停止，则不需要使用此指令。

发送的指令参数

参数	DB 偏移量
指令码 202	2.0

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，返回 **2104** 状态码；否则返回对应的错误码。

203 指令——选择 Mech-Viz 分支

该指令用于指定工程将沿哪个分支运行。分支机制通过 branch_by_msg 步骤建立，该指令则通过指定该步骤的出口来指定分支。

在执行该指令前请先执行*201 指令——启动 Mech-Viz 工程* 。

Mech-Viz 工程运行至 branch_by_msg 步骤时，将等待该指令指定出口。

发送的指令参数

参数	DB 偏移量
指令码 203	2.0
分支步骤编号	60.0
出口号	62.0

分支步骤编号

该参数用于指定分支选择将在哪个 branch_by_msg 步骤上进行。

该参数应为正整数，即 branch_by_msg 的步骤编号。步骤编号可在步骤参数中读取。

出口号

该参数用于指定工程将沿 branch_by_msg 步骤的哪个出口运行，Mech-Viz 程序将沿该出口继续执行。参数为正整数。

出口号范围：1~99。

提示：

- 出口号为 Mech-Viz 显示的端口号 + 1。如端口号为 0，则出口号为 1。
- 出口号为从 1 开始的端口索引号。比如指定的出口为从左往右数第二个端口，那么出口号为 2。

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **2105** 状态码；否则返回对应的错误码。

204 指令——设置移动索引

该指令用于设定步骤的索引参数值。该类步骤一般用于连续或单独指定的移动或其他操作。

带有索引参数的步骤包括“按序列移动”、“按阵列移动”、“自定义垛型”、“预设垛型”等。

在执行该指令前，请先执行*201 指令——启动 Mech-Viz 工程* 。

发送的指令参数

参数	DB 偏移量
指令码 204	2.0
步骤编号	64.0
索引值	66.0

步骤编号

该参数指定哪个步骤需要索引参数设定。

该参数值应为正整数，即待索引步骤的步骤编号。步骤编号可在步骤参数中读取。

索引值

下次执行此步骤时应设置的索引值。

发送该指令时，Mech-Viz 中的当前索引值将变为该参数值减 1。

当 Mech-Viz 项目运行到该指令指定的步骤时，Mech-Viz 中的当前索引值将增加 1，成为该参数的值。

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **2106** 状态码；否则返回对应的错误码。

205 指令——获取规划路径

在执行*201 指令——启动 Mech-Viz 工程* 后，该指令用于获取 Mech-Viz 规划的路径。

在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

注解： 如果工程中某个移动类步骤的路径点不应发送给机器人，请取消勾选该步骤参数中的“发送路径点”选项。

发送的指令参数

参数	DB 偏移量
指令码 205	2.0
路径点类型	4.0

路径点类型

该参数用于指定 Mech-Viz 将返回何种形式的路径点。

- 1：路径点将以机器人关节角（JPs）的形式返回。
- 2：路径点将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	DB 偏移量
状态码	200.0
数据传输状态	202.0
路径点数量	204.0
“视觉移动”位置	206.0
此次发送的所有路径点的位姿	208.0
此次发送的所有路径点的标签	1168.0
此次发送的所有路径点的速度	1248.0

状态码

若指令执行正常，则返回 **2100** 状态码；否则返回对应的错误码。

提示： 调用该指令时，若 Mech-Viz 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

1：表示返回的数据是新的视觉点，请读取。

注意： 在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数表示返回的路径点个数，取值范围为 0~20。在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

“视觉移动”位置

“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下步骤组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 205 指令中的路径点类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前使用步骤 label_mapping 将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

移动类步骤参数中的非零速度参数百分数值。

206 指令——获取 DO 信号列表

该指令用于获取规划的 DO 信号列表。DO 信号列表用于控制多个工具或吸盘分区。

执行该指令前，需要先执行*205 指令* 获取 Mech-Viz 规划路径。

请根据模板工程来部署 Mech-Viz 工程，并在工程中设置对应的吸盘配置文件。模板工程为 Mech-Center 安装目录 (tool/viz_project) 下的 **suction_zone** 工程。

在工程的 set_do_list 步骤的参数中：

- 在“接收对象”下勾选“标准接口”
- 勾选“从视觉移动中获取 DO 列表”
- 在参数栏底部选择需要 DO 信号列表的视觉移动步骤

发送的指令参数

参数	DB 偏移量
指令码 206	2.0

返回的数据参数

参数	DB 偏移量
状态码	200.0
DO 信号列表	1408.0

状态码

若指令执行正常，则返回 **2102** 状态码；否则返回对应的错误码。

DO 端口值

共有 64 个 DO 信号值，为整数。

DO 信号值范围：0~999。

占位值：-1。

例如：DO 信号值为 1、3、5、6 。

1	3	5	6	-1	-1	-1	-1	…	-1	-1
第 1 位	第 2 位	第 3 位	第 4 位	第 5 位	第 6 位	第 7 位	第 8 位	…	第 63 位	第 64 位

501 指令——向 Mech-Vision 传入物体尺寸

该参数用于往 Mech-Vision 工程中动态传入物体尺寸。启动 Mech-Vision 工程前请先确认物体尺寸。

Mech-Vision 工程中应有 read_object_dimensions 步骤。该步骤参数 **从参数读取物体尺寸**应设定为 `True`。

发送的指令参数

参数	DB 偏移量
指令码 501	2.0
Mech-Vision 工程编号	8.0
[长, 宽, 高]	68.0

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

长，宽，高

传入 Mech-Vision 工程的物体尺寸。尺寸值将被 read_object_dimensions 步骤读取。

单位：毫米（mm）

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **1108** 状态码；否则返回对应的错误码。

502 指令——向 Mech-Viz 传入 TCP

该指令用于往 Mech-Viz 工程中动态传入机器人 TCP。读取机器人 TCP 的步骤为 outer_move。

请基于模板工程部署 Mech-Viz 工程。模板工程路径为 Mech-Center 安装目录下 `tool/viz_project/outer_move` 。

请将 outer_move 步骤放在工作流程中合适的位置。

该指令需在执行*201 指令——启动 Mech-Viz 工程* 之前执行。

发送的指令参数

参数	DB 偏移量
指令码 502	2.0
TCP	80.0

TCP

用于设定 outer_move 步骤路径点的 TCP 数据。

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

若指令执行正常，则返回 **2107** 状态码；否则返回对应的错误码。

901 指令——获取软件状态

该指令用于检查软件运行状态（Mech-Vision、Mech-Viz、Mech-Center）。

目前该指令只支持检查 Mech-Vision 是否可以开始运行工程。

发送的指令参数

参数	DB 偏移量
指令码 901	2.0

参数说明：无参数。

返回的数据参数

参数	DB 偏移量
状态码	200.0

状态码

系统自检状态。**1101** 状态码为“Mech-Vision 工程已就绪”；其他状态码为“Mech-Vision 工程未就绪”。目前该指令只能用于检查 Mech-Vision 工程是否已就绪。

3.3.4 PROFINET

通过基于 PROFINET 协议的标准接口，梅卡曼德软件系统可以使用 TIA Portal 软件与 Siemens SIMATIC S7 PLC 进行通信。

详细内容可参考 PROFINET - Siemens SIMATIC S7 PLC 。

通信协议

从 Mech-Center 到 PLC

Control_Output

Bit 位	数据
7	/
6	/
5	/
4	指令执行完成（布尔值）
3	数据已更新（布尔值）
2	相机曝光完成（布尔值）
1	系统触发成功（布尔值）
0	心跳（布尔值）

Command_Complete

该信号用于指示指令执行完成，用户可以读取端口返回的状态码和其他数据。对于 102 和 205 指令，只有当最后一组位姿数据传输完成时，该信号才会被置高。

Data_Ready

该信号用于指示位姿数据可读。专用于 102 或 205 指令接收多组机器人位姿数据时。

Exposure_Complete

当相机曝光完成时，该信号被置高。用于指示被拍目标物体或 EIH 机器人可以从拍照位置移动。

Trigger_Acknowledge

Trigger_Acknowledge 等于 1 表示视觉系统被 Trigger 信号成功触发。该信号会保持高位直至 Trigger 信号复位。

Heartbeat

系统心跳，1 秒反转一次。

Status_Code

状态码，INT32。

视觉系统返回的执行状态码，包括正常状态与错误码。

Calib_Cam_Status

标定进行状态，INT8。

专用于 701 指令标定。0：标定中。1：标定结束。

Send_Pose_Num

发送的位姿数量，INT8。通过此次执行指令传输的位姿数。

Visual_Point_Index

“视觉移动”步骤目标点在整个路径中的位置。“视觉移动”步骤即移动至视觉点（抓取物体的点）的移动步骤。

比如，如果规划路径由以下步骤组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”步骤位置为 3。

如果路径中无“视觉移动”步骤，则该参数值为 0。

数据类型：INT8

DO_List

用于控制多个吸盘分区或阵列夹具的 64 个 INT8 DO 信号。

Byte	Bit 0~7
0	DO 列表 0，信号 0~7
1	DO 列表 1，信号 8~15
2	DO 列表 2，信号 16~23
3	DO 列表 3，信号 24~31
4	DO 列表 4，信号 32~39
5	DO 列表 5，信号 40~47
6	DO 列表 6，信号 48~55
7	DO 列表 7，信号 56~63

Notify_Message

Mech-Viz/Mech-Vision 的“通知”步骤发送的自定义整数消息。

以整数为格式的消息，INT32。

Send_Pose_Type

发送的位姿类型，INT8。

- `1`：机器人关节角 JPs。
- `2`：机器人工具中心点 TCP。

Target_Pose

机器人 TCP 或 JPs 形式的目标点机器人位姿。

> **注意：** 从此模块的读取的数据在使用前应除以 10000。

以三维坐标和欧拉角表示的位姿数据结构如下：

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

以机器人关节角 JPs 表示的位姿最多包含 6 个关节角度：

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

Byte	Bit 0~7
0~3	目标点 X 坐标或 J1 关节角，INT32
4~7	目标点 Y 坐标或 J2 关节角，INT32
8~11	目标点 Z 坐标或 J3 关节角，INT32
12~15	目标点 A 角度或 J4 关节角，INT32
16~19	目标点 B 角度或 J5 关节角，INT32
20~23	目标点 C 角度或 J6 关节角，INT32

Target_Label

发送的位姿对应的标签。该值为非负整数。

数据类型：INT32

Target_Speed

目标点对应的移动类步骤的速度参数百分比值。范围：0 到 100。

数据类型：INT32

Ext_Output_Data

保留模块，用于传输其他数据。

该模块占用 40 个字节（INT32[1:10]，总共 10 个 INT32 整数）。

从 PLC 到 Mech-Center

Control_Input

Bit 位	数据
7	/
6	/
5	/
4	重置消息通知（布尔值）
3	数据确认（布尔值）
2	复位 Exposure_Complete “曝光完成”（布尔值）
1	触发信号（布尔值）
0	通信使能（布尔值）

Reset_Exposure

复位 Exposure_Complete “曝光完成”（布尔值）

如果 Reset_Exposure = 1，Exposure Complete 将被设置为 0。

Data_Acknowledge

数据确认（布尔值）用于确认已读取执行 102 指令或 205 指令返回的数据。

Data_Acknowledge = 0，表示 PLC 还没有从 Mech-Center 读取数据，数据保留在端口。

Data_Acknowledge = 1，表示 PLC 已从 Mech-Center 读取数据，Mech-Center 可以写入下一轮的数据。

Data_Acknowledge 可以在 Heartbeat 翻转或 Data_Ready = 0 时重置。

Reset_Notify

重置消息通知（布尔值）

如果 Reset_Notify = 1，Notify_Message 的内容将被清除。

Trigger

触发信号（布尔值）

如果 Trigger = 1，Mech-Center 将读取发送的指令并执行该指令。

一旦 Mech-Center 接收到触发信号，就可以重置 Trigger_Acknowledge。

信号的上行部分认为是 1。

Comm_Enable

通信使能（布尔值）

`0`：通信禁用。Mech-Center 将忽略触发信号。

`1`：通信使能。触发信号将起作用，Mech-Center 将接收指令。

Command

指令码，INT32。

Calib_Rob_Status

- `0`：标定开始。
- `1`：机器人已正常移动到发送的最新标定点。
- `2`：机器人未能移动到发送的最新标定点。

数据类型：INT8

Vision_Proj_Num

Mech-Vision 工程编号，可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

数据类型：INT8

Vision_Recipe_Num

Mech-Vision 工程中配方模板的编号，为正整数。单击 *工程助手 ▸ 参数配方*，进入参数配方编辑器。编号范围：1~99。

数据类型：INT8

Viz_Task_Name

指令涉及的 Mech-Viz 步骤的步骤编号。可以在 Task 的参数中读取和设置。

数据类型：INT8

Viz_Task_Value

Mech-Viz 分支步骤出口端口号，或给 Mech-Viz 步骤的 index 参数设置的值。

数据类型：INT8

Req_Pose_Num

从 Mech-Vision 请求的视觉点数。

`0`：从 Mech-Vision 的视觉结果中请求所有可用的视觉点。

数据类型：INT8

Robot_Pose_Type

机器人的位姿类型。

数据类型：INT8

Req_Pose_Type

希望 Mech-Viz 返回的机器人位姿类型。

- `1`：JPs 类型。
- `2`：TCP 类型。

数据类型：INT8

Robot_Pose_JPS

拍照时的机器人关节角 JPs。

设置模块前请先将 JPs 数据乘 10000。

JPs 包含最多 6 个关节角数据（6 个 INT32 整数）：

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

Byte	Bit 0~7
0~3	机器人当前 J1 关节角 INT32
4~7	机器人当前 J2 关节角 INT32
8~11	机器人当前 J3 关节角 INT32
12~15	机器人当前 J4 关节角 INT32
16~19	机器人当前 J5 关节角 INT32
20~23	机器人当前 J6 关节角 INT32

Robot_Pose_TCP

用于拍照的机器人法兰位姿。

在设置到模块之前，请将位姿数据乘 10000。

法兰包括三维坐标（X、Y、Z）和欧拉角（A、B、C），总共 6 个 INT32 整数。

Byte	Bit 0~7
0~3	机器人当前 X 坐标 INT32
4~7	机器人当前 Y 坐标 INT32
8~11	机器人当前 Z 坐标 INT32
12~15	机器人当前 A 角度 INT32
16~19	机器人当前 B 角度 INT32
20~23	机器人当前 C 角度 INT32

Ext_Input_Data

保留模块，用于传输其他数据。

该模块占用 40 个字节（INT32[1:10]，总共 10 个 INT32 整数）。

Profinet 指令说明

101 指令——启动 Mech-Vision 工程

功能介绍

该指令启动 Mech-Vision 工程，用于执行相机拍照和视觉识别。

如果工程为 Eye In Hand 模式，该指令将把机器人拍照位姿传入工程。

该指令用于仅使用 Mech-Vision 的场景。

发送的指令参数

参数	描述
Vision_Proj_Num	Mech-Vision 工程编号
Req_Pose_Num	预期视觉点数量
Robot_Pose_Type	机器人位姿类型
Robot_Pose_JPS / Robot_Pose_TCP	机器人 JPs / 法兰位姿

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

预期视觉点数量

期望从 Mech-Vision 得到的视觉点数量。视觉点信息包括视觉位姿及对应点云、标签、缩放等。

- `0`：从 Mech-Vision 工程取得识别结果中所有的视觉点。
- `大于 0 的整数`：从 Mech-Vision 工程取得识别结果中指定数量的视觉点。
  - 如果视觉点总数小于该参数值，则取得识别结果中所有视觉点。
  - 如果视觉点总数大于等于该参数值，则取得该参数指定数量的视觉点。

机器人位姿类型、机器人 JPs / 法兰位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Vision，其取值范围为 0~3。
- **机器人 JPs / 法兰位姿**参数值取决于 **机器人位姿类型**参数值。

下表为 Robot_Pose_Type 与 Robot_Pose_JPS 和 Robot_Pose_TCP 参数取值的关系及说明。

Robot_Pose_Type	Robot_Pose_JPS	Robot_Pose_TCP	说明	适用场景
0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	无需向 Mech-Vision 传入机器人位姿	工程为 Eye To Hand 模式。若 Mech-Vision 工程中使用“路径规划”步骤，则路径规划的起始点为路径规划工具中设置的 Home 点。
1	机器人当前关节角	机器人的当前法兰位姿	需要将机器人的关节角和法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，除桁架机器人外的大多数机器人适用该设定。
2	0, 0, 0, 0, 0, 0	机器人的当前法兰位姿	需要将机器人的当前法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，机器人无关节角数据，仅有法兰位姿数据（如桁架机器人）。
3	机器人路径规划起始点的关节角	0, 0, 0, 0, 0, 0	需要将机器人路径规划起始点的关节角传入 Mech-Vision	工程为 Eye To Hand 模式，并且 Mech-Vision 工程中存在“路径规划”步骤，且需要从机器人端设置“路径规划”步骤的起始点。

注意： JPs 和法兰位姿的浮点数数据需乘 10000，再设置到 Robot_Pose_JPS 或 Robot_Pose_TCP 模块中。

返回的数据参数

`101, 状态码`

状态码

若指令执行正常，返回 **1102** 状态码；否则返回对应的错误码。

102 指令——获取视觉目标点

该指令用在*101 指令——启动 Mech-Vision 工程* 之后，用于获取 Mech-Vision 输出的视觉点，然后将视觉点转换为视觉目标点。

具体转换过程如下所示，即将视觉点包含的位姿转换为对应机器人 TCP。

- 将视觉点包含的位姿绕 Y 轴旋转 180°。
- 识别对应机器人型号的参考坐标系定义是否涉及机器人基座高度，并相应增加垂直方向的偏置。

提示： 102 指令单次接收视觉目标点的数量上限默认为 20。如需要获取的视觉目标点的数量大于 20，需多次执行该指令来获取所有所需的视觉目标点。

发送的指令参数

参数	描述
Command	102
Vision_Proj_Num	Mech-Vision 工程编号

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

参数	描述
Status Code	状态码
Send_Pose_Num	视觉目标点数量
Send_Pose_Type	位姿类型
Target_Pose	此次获取的所有 TCP
Target_Label	此次获取的所有标签

状态码

若指令执行正常，则返回 **1100** 状态码；否则返回对应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

视觉目标点数量

执行该指令获得的视觉目标点数量。

位姿类型

该参数表示将 Mech-Vision 返回的视觉点中的目标物体位姿转换为相应的 TCP。

该参数的值默认为 2，表示位姿类型为 TCP。

此次获取的所有 TCP

单个 TCP 所含信息包括空间坐标（XYZ）和方向欧拉角（ABC）。

此次获取的所有标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前用步骤 label_mapping 将标签映射为整数。如工程中没有标签，则该参数为默认值 0。

注意： 具体收发操作请参考通信控制流程 。使用 Data_ready、Data_Acknowledge、Command_Complete 信号进行过程控制。

103 指令——切换 Mech-Vision 配方

切换 Mech-Vision 工程内的参数配方。

Mech-Vision 中步骤的参数设定可通过切换参数配方调整。

参数配方调整涉及的参数通常包括点云匹配模板、图像匹配模板、感兴趣区域、置信度阈值等。

> **注意：** 需要在执行*101 指令——启动 Mech-Vision 工程* 之前使用该指令。

发送的指令参数

参数	描述
Command	103
Vision_Proj_Num	Mech-Vision 工程编号
Vision_Recipe_Num	Mech-Vision 配方编号

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

Mech-Vision 配方编号

Mech-Vision 工程中配方模板的编号，为正整数。单击 `工程助手 ▸ 参数配方`，进入参数配方编辑器。编号范围：1~99。

返回的数据参数

状态码

若指令执行正常，则返回 **1107** 状态码；否则返回对应的错误码。

105 指令——获取 Mech-Vision “路径规划”步骤的结果

在调用 101 指令之后，使用该指令获取 Mech-Vision 中“路径规划”步骤输出的免碰撞抓取路径。

在使用该指令时，Mech-Vision “输出”步骤的 **端口类型**参数需要设置为“预定义（机器人路径）”。

提示：在调用 105 指令前，请务必将 101 指令的 **预期视觉点数量**设置为 0，以减少调用 105 指令的次数。若 101 指令的 **预期视觉点数量**设置为 1，则每次调用 105 指令只会返回一个路径点，只有多次调用 105 指令才能接收全部路径点。

发送的指令参数

参数	描述
Command	105
Vision_Proj_Num	Mech-Vision 工程编号
Req_Pose_Type	路径点位姿类型

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

路径点位姿类型

该参数用于指定”路径规划“步骤返回的路径点的位姿类型。

- `1`：路径点的位姿将以机器人关节角（JPs）的形式返回。
- `2`：路径点的位姿将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	描述
Status Code	状态码
Send_Pose_Num	路径点数量
Send_Pose_Type	位姿类型
Visual_Point_Index	“视觉移动”位置
Target_Pose	此次发送的所有路径点的位姿
Target_Label	此次发送的所有路径点的标签
Target_Speed	此次发送的所有路径点的速度

状态码

若指令执行正常，则返回 **1103** 状态码；否则返回对应的错误码。

路径点数量

该参数用于表示此次执行该指令返回的路径点个数，取值范围为 0~20。若获取的路径点数多于 20，请多次调用该指令。

位姿类型

与该指令发出的“Req_Pose_Type”值相同。

- 1: JPs
- 2: TCP

“视觉移动”位置

路径规划工具中设置的“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下组成：“移动 _1”，“移动 _2”，“视觉移动”，“移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 105 指令中的路径点位姿类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在“输出”步骤前使用“标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

路径规划工具中设置的速度值。

201 指令——启动 Mech-Viz 工程

用于既使用 Mech-Vision 又使用 Mech-Viz 的场景。该指令用于运行 Mech-Viz 工程，启动对应的 Mech-Vision 工程，Mech-Viz 基于 Mech-Vision 的视觉结果规划机器人的运动路径。

请在需要启动 Mech-Viz 工程中勾选 *自动加载*。

Mech-Center 安装目录 (tool/viz_project) 文件夹内保存有一些典型应用工程模板，用户可以在模板的基础上进行修改。

用于标准接口的 Mech-Viz 样例工程描述请见*标准接口用 Mech-Viz 样例工程*。

发送的指令参数

参数	描述
Command	201
Robot_Pose_Type	机器人位姿类型
Robot_Pose_JPS / Robot_Pose_TCP	机器人 JPs / 法兰位姿

机器人位姿类型、机器人 JPs / 法兰位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Viz，其取值范围为 0~2。
- **机器人 JPs / 法兰位姿**参数值取决于 **机器人位姿类型**参数值。

下表为 Robot_Pose_Type 与 Robot_Pose_JPS 和 Robot_Pose_TCP 参数取值的关系及说明。

Robot_Pose_Type	Robot_Pose_JPS	Robot_Pose_TCP	说明	适用场景
0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	无需向 Mech-Viz 传入机器人位姿，Mech-Viz 中仿真机器人将从初始位姿 JPs = [0, 0, 0, 0, 0, 0] 开始移动到第一个路径点。	工程为 Eye To Hand 模式。不推荐使用该设定。
1	机器人当前关节角	机器人的当前法兰位姿	需要将机器人的当前关节角和法兰位姿传入 Mech-Viz，Mech-Viz 中仿真机器人将从传入的关节角开始移动到第一个路径点。	工程为 Eye In Hand 模式时，推荐使用该设定。
2	机器人端自定义的关节角	0, 0, 0, 0, 0, 0	需要将机器人的一个示教点（非当前关节角）传入 Mech-Viz，用于在机器人处于拍照区域外时，提前触发 Mech-Viz 工程规划下一轮路径（如下图），Mech-Viz 中仿真机器人将从传入的示教点开始运动到第一个路径点。	工程为 Eye To Hand 模式时，推荐使用该设定。

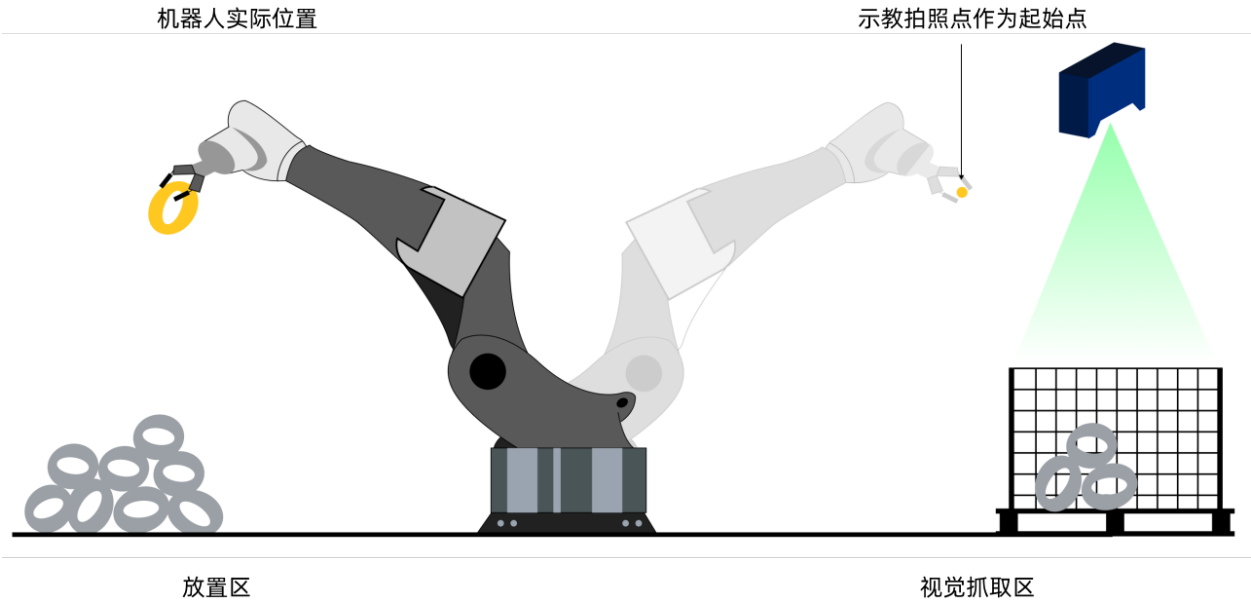
Eye To Hand 模式下应将 **Robot_Pose_Type** 设为 2 的原因：

Eye To Hand 模式下，相机可在机器人回到拍照及抓取区域之前拍照，用于提前规划下一轮抓取路径，从而缩短节拍。

若此时将 **Robot_Pose_Type** 设为 1，即将机器人当前位姿发送给 Mech-Viz 仿真机器人，可能导致仿真机器人与真实机器人轨迹不一致，发生未预知的碰撞，造成危险。

即仿真机器人将直接从当前位姿移动至 Mech-Viz 中第一个移动步骤中设置的位姿，而真实机器人可能在移动至其他位姿后才移动至上述位姿。

所以应将 **Robot_Pose_Type** 参数设为 2。



返回的数据参数

状态码

若指令执行正常，则返回 **2103** 状态码；否则返回对应的错误码。

202 指令——停止 Mech-Viz 工程

停止 Mech-Viz 运行。如果 Mech-Viz 工程不是死循环，或可以正常停止，则不需要使用该指令。

发送的指令参数

参数	描述
Command	202

返回的数据参数

状态码

若指令执行正常，返回 **2104** 状态码；否则返回对应的错误码。

203 指令——选择 Mech-Viz 分支

该指令用于指定工程将沿哪个分支运行。分支机制通过 branch_by_msg 步骤建立，该指令则通过指定该步骤的出口来指定分支。

在执行该指令前请先执行*201 指令——启动 Mech-Viz 工程* 。

Mech-Viz 工程运行至 branch_by_msg 步骤时，将等待该指令指定出口。

发送的指令参数

参数	描述
Command	203
Viz_Task_ID	分支步骤编号
Viz_Task_Value	出口号

分支步骤编号

该参数用于指定分支选择将在哪个 branch_by_msg 步骤上进行。

该参数应为正整数，即 branch_by_msg 的步骤编号。步骤编号可在步骤参数中读取。

出口号

该参数用于指定工程将沿 branch_by_msg 步骤的哪个出口运行，Mech-Viz 程序将沿该出口继续执行。参数为正整数。

提示:

- 出口号为 Mech-Viz 显示的端口号 + 1。如端口号为 0，则出口号为 1。
- 出口号为从 1 开始的端口索引号。比如指定的出口为从左往右数第二个端口，那么出口号为 2。

返回的数据参数

状态码

若指令执行正常，则返回 **2105** 状态码；否则返回对应的错误码。

204 指令——设置移动索引

该指令用于设定步骤的索引参数值。该类步骤一般用于连续或单独指定的移动或其他操作。

带有索引参数的步骤包括“按序列移动”、“按阵列移动”、“自定义垛型”、“预设垛型”等。

在执行该指令前，请先执行*201 指令——启动 Mech-Viz 工程*。

发送的指令参数

参数	描述
Command	204
Viz_Task_ID	步骤编号
Viz_Task_Value	索引值

步骤编号

该参数指定哪个步骤需要索引参数设定。

该参数值应为正整数，即待索引步骤的步骤编号。步骤编号可在步骤参数中读取。

索引值

下次执行此步骤时应设置的索引值。

发送该指令时，Mech-Viz 中的当前索引值将变为该参数值减 1。

当 Mech-Viz 项目运行到该指令指定的步骤时，Mech-Viz 中的当前索引值将增加 1，成为该参数的值。

返回的数据参数

状态码

若指令执行正常，则返回 **2106** 状态码；否则返回对应的错误码。

205 指令——获取规划路径

在执行*201 指令——启动 Mech-Viz 工程* 后，该指令用于获取 Mech-Viz 规划的路径。

在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

注解： 如果工程中某个移动类步骤的路径点不应发送给机器人，请取消勾选该步骤参数中的“发送路径点”选项。

发送的指令参数

参数	描述
Command	205
Req_Pose_Type	路径点类型

路径点类型

该参数用于指定 Mech-Viz 将返回何种形式的路径点。

- `1`：路径点将以机器人关节角（JPs）的形式返回。
- `2`：路径点将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	描述
Status Code	状态码
Send_Pose_Num	路径点数量
Send_Pose_Type	位姿类型
Visual_Point_Index	“视觉移动”位置
Target_Pose	此次发送的所有路径点的位姿
Target_Label	此次发送的所有路径点的标签
Target_Speed	此次发送的所有路径点的速度

状态码

若指令执行正常，则返回 **2100** 状态码；否则返回对应的错误码。

提示： 调用该指令时，若 Mech-Viz 结果还未返回（正在运行中），Mech-Center 会等待 Mech-Viz 结果返回后再给机器人返回，默认等待 10s 超时，若发生超时，则给机器人返回超时错误。

路径点数量

该参数表示返回的路径点个数，取值范围为 0~20。在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

位姿类型

与该指令发出的“Req_Pose_Type”值相同。

- 1: JPs
- 2: TCP

“视觉移动”位置

“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下步骤组成：“移动 _1”，“移动 _2”，“视觉移动”，“移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 205 指令中的路径点类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前使用步骤 label_mapping 将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

移动类步骤参数中的非零速度参数百分数值。

注意： 位姿的详细收发操作请参考通信控制流程 。使用 Data_ready、Data_Acknowledge、Command_Complete 信号进行过程控制。

206 指令——获取 DO 信号列表

该指令用于获取规划的 DO 信号列表。DO 信号列表用于控制多个工具或吸盘分区。

执行该指令前，需要先执行*205 指令* 获取 Mech-Viz 规划路径。

请根据模板工程来部署 Mech-Viz 工程，并在工程中设置对应的吸盘配置文件。模板工程为 Mech-Center 安装目录 (tool/viz_project) 下的 **suction_zone** 工程。

在工程的 set_do_list 步骤的参数中：

- 在“接收对象”下勾选“标准接口”
- 勾选“从视觉移动中获取 DO 列表”

- 在参数栏底部选择需要 DO 信号列表的视觉移动步骤

发送的指令参数

参数	描述
Command	206

返回的数据参数

参数	描述
Status code	状态码
DO List	DO 信号列表

状态码

若指令执行正常，则返回 **2102** 状态码；否则返回对应的错误码。

DO 信号列表

返回的数据参数末尾共有 64 个 DO 信号值，为整数。

DO 信号值范围：0~999。

占位值：-1。

501 指令——向 Mech-Vision 中传入物体尺寸

该参数用于往 Mech-Vision 工程中动态传入物体尺寸。启动 Mech-Vision 工程前请先确认物体尺寸。Mech-Vision 工程中应有 read_object_dimensions 步骤。该步骤参数 **从参数读取物体尺寸**应设定为 `True`。

发送的指令参数

参数	描述
Command	501
Vision_Proj_Num	Mech-Vision 工程编号
Ext_Input_Data	物体尺寸

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

物体尺寸

传入 Mech-Vision 工程的物体长、宽、高尺寸。尺寸值将被 read_object_dimensions 步骤读取。

单位：毫米（mm）

返回的数据参数

状态码

若指令执行正常，则返回 **1108** 状态码；否则返回对应的错误码。

502 指令——向 Mech-Viz 中传入 TCP

该指令用于往 Mech-Viz 工程中动态传入机器人 TCP。读取机器人 TCP 的步骤为 outer_move。

请基于模板工程部署 Mech-Viz 工程。模板工程路径为 Mech-Center 安装目录下 `tool/viz_project/outer_move` 。

请将 outer_move 步骤放在工作流程中合适的位置。

该指令需在执行*201 指令——启动 Mech-Viz 工程* 之前执行。

发送的指令参数

参数	描述
Command	502
Ext_Input_Data	机器人 TCP

机器人 **TCP**

用于设定 outer_move 步骤路径点的 TCP 数据。

> **注意：** 需要将机器人的 TCP（单位 mm）数据乘 10000，再设定 Ext_Input_Data 的值。

返回的数据参数

状态码

若指令执行正常，则返回 **2107** 状态码；否则返回对应的错误码。

901 指令——获取软件状态

该指令用于检查软件运行状态（Mech-Vision、Mech-Viz、Mech-Center）。

目前该指令只支持检查 Mech-Vision 是否可以开始运行工程。

发送的指令参数

参数	描述
Command	901

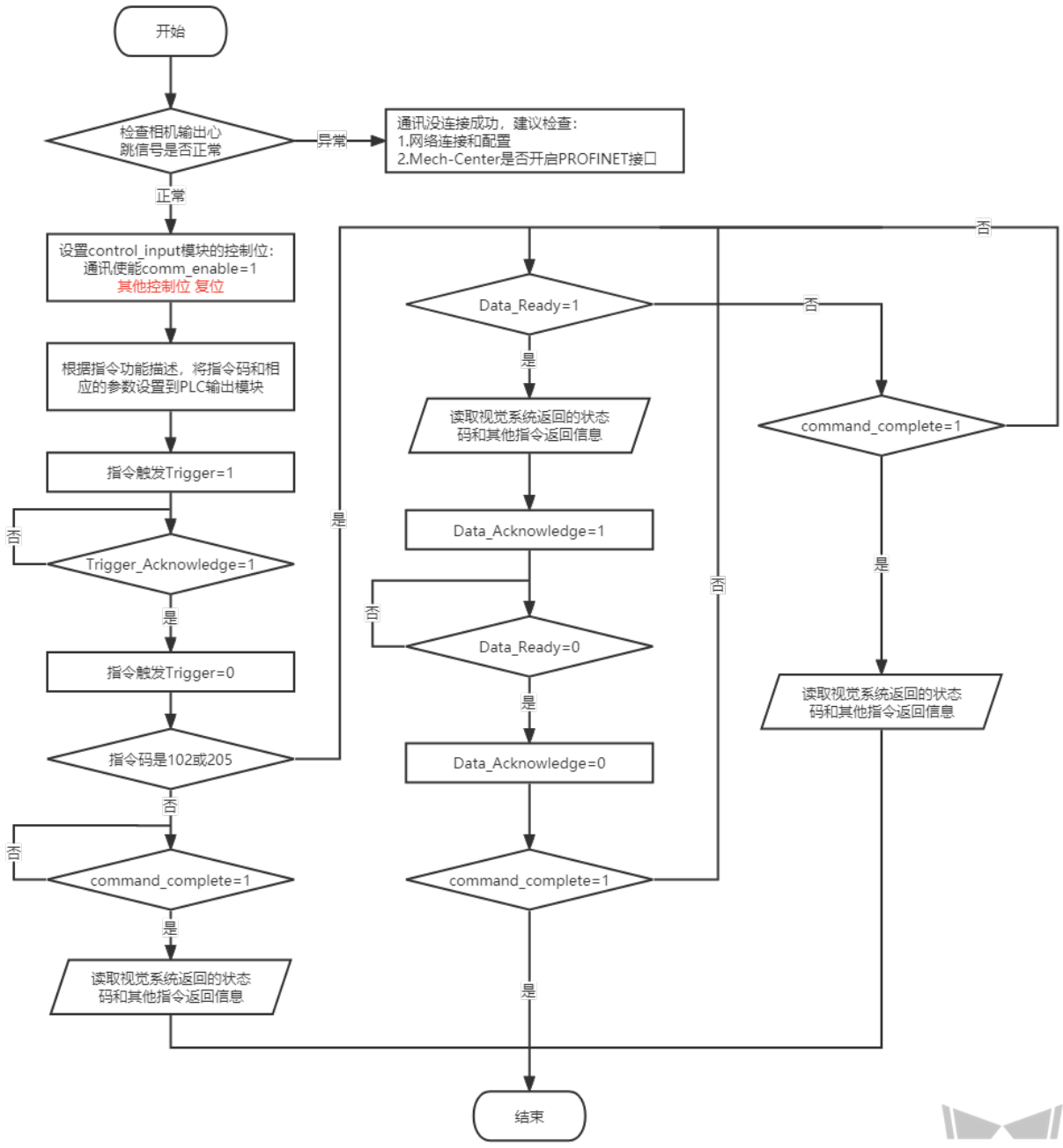
返回的数据参数

状态码

系统自检状态。**1101** 状态码为“Mech-Vision 工程已就绪”；其他状态码为“Mech-Vision 工程未就绪”。目前该指令只能用于检查 Mech-Vision 工程是否已就绪。

通信控制流程

Mech-Mind 视觉系统使用 PROFINET 控制流程如下图所示。


开始
检查相机输出心跳信号是否正常
异常
通讯没连接成功，建议检查：
1.网络连接和配置
2.Mech-Center是否开启PROFINET接口
正常
设置control_input模块的控制位：
通讯使能comm_enable=1
其他控制位 复位
根据指令功能描述，将指令码和相应的参数设置到PLC输出模块
指令触发Trigger=1
否
Trigger_Acknowledge=1
是
指令触发Trigger=0
指令码是102或205
是
否
否
command_complete=1
是
读取视觉系统返回的状态码和其他指令返回信息
Data_Ready=1
否
是
读取视觉系统返回的状态码和其他指令返回信息
Data_Acknowledge=1
否
Data_Ready=0
是
Data_Acknowledge=0
command_complete=1
否
是
否
command_complete=1
是
读取视觉系统返回的状态码和其他指令返回信息
结束

3.3.5 Ethernet/IP

通过基于 Ethernet/IP 协议的标准接口，梅卡曼德软件系统可以与欧姆龙 PLC 、基恩士 PLC 进行通信。

具体内容可参考 standard_interface_robot_and_plc ，指令说明可参考*Profinet 指令说明* 。

3.3.6 Modbus TCP

通过基于 Modbus TCP 协议的标准接口，梅卡曼德软件系统可以与主站设备（PLC 或机器人控制器）进行通信。

具体例程可以参考 《Modbus TCP - 西门子 SIMATIC S7 PLC》和 《Modbus TCP - 三菱 Q 系列 PLC》。

寄存器映射表

Mech-Center Modbus TCP Register Map Table（寄存器映射表）如下表所示。

地址（十进制）	地址（十六进制）	内容	长度（以字为单位）	读取/写入	备注	保持寄存器（4x）
0	0x0000	指令触发	1	写入	0：不触发指令 1：触发指令	40001–40053：写入 Mech-Center
1	0x0001	指令	1	写入	功能码	
2	0x0002	位姿类型	1	写入		
3	0x0003	位姿个数	1	写入		
4	0x0004	Mech-Vision 工程编号	1	写入		
5	0x0005	配方编号	1	写入		
6	0x0006	关节角	12	写入	单位：度	
18	0x0012	TCP	12	写入	单位：毫米和欧拉角度数	
30	0x001E	分支步骤编号	1	写入		
31	0x001F	分支出口	1	写入		
32	0x0020	索引名称	1	写入		
33	0x0021	索引值	1	写入		
34	0x0022	外部箱子尺寸	6	写入	单位：毫米	
40	0x0028	外部输入位姿	12	写入	单位：毫米和欧拉角度数	
52	0x0034	机器人移动状态	1	写入		
53	0x0035	预留	44	读取		40054–40728：读取 Mech-Center
97	0x0061	触发确认	1	读取	0：未接受指令触发 1：已接受指令触发	
98	0x0062	通知	1	读取		
99	0x0063	心跳	1	读取	1 Hz	
100	0x0064	状态码	1	读取		
101	0x0065	位姿数据状态	1	读取	0：已收到 999 指令（清除寄存器数据）1：新位姿数据	
102	0x0066	发送位姿数量	1	读取		
103	0x0067	视觉点在规划路径中的位置	1	读取		
104	0x0068	目标点	480	读取		
584	0x0248	目标标签	40	读取		
624	0x0270	速度百分比	40	读取		
664	0x0298	DO 信号列表	64	读取		

注解： Modbus TCP 是一种简单总线协议，建议单次读取或者写入 100 个字左右，单次读取或者写入的通信周期约为 70 ms。在可选择的情况下，尽可能选取 PROFINET、Ethernet/IP 等实时以太网协议，或者西门子自研的 Snap7 通信协议（对应 Mech-Center Siemens PLC Client 标准接口），或者三菱自研的 MELSEC 通信协议（简称 MC 协议）。

保持寄存器指令

101 指令——启动 Mech-Vision 工程

该指令启动 Mech-Vision 工程，用于执行相机拍照和视觉识别。

如果工程为 Eye In Hand 模式，该指令将把机器人拍照位姿传入工程。

该指令用于仅使用 Mech-Vision 的场景。

发送的指令参数

参数	地址（十进制）
指令码 101	1
Mech-Vision 工程编号	4
预期视觉点数量	3
机器人位姿类型	2
机器人位姿	6-17（JPs）或 18-29（法兰位姿）

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

预期视觉点数量

期望从 Mech-Vision 得到的视觉点数量。视觉点信息包括视觉位姿及对应点云、标签、缩放等。

- 0：从 Mech-Vision 工程取得识别结果中所有的视觉点。
- 大于 0 的整数：从 Mech-Vision 工程取得识别结果中指定数量的视觉点。
  - 如果视觉点总数小于该参数值，则取得识别结果中所有视觉点。
  - 如果视觉点总数大于等于该参数值，则取得该参数指定数量的视觉点。

提示：获取视觉点的指令是 102 指令。默认设置下，102 指令每次最多能获取 20 个视觉点，若需要获取的视觉点数量大于 20，则需重复调用 102 指令。

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Vision，其取值范围为 0~3。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	**说明**	**适用场景**
0	0, 0, 0, 0, 0, 0	无需向 Mech-Vision 传入机器人位姿	工程为 Eye To Hand 模式。若 Mech-Vision 工程中使用“路径规划”步骤，则路径规划的起始点为路径规划工具中设置的 Home 点。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的关节角和法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，除桁架机器人外的大多数机器人适用该设定。
2	机器人的当前法兰位姿	需要将机器人的当前法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，机器人无关节角数据，仅有法兰位姿数据（如桁架机器人）。
3	机器人路径规划起始点的关节角	需要将机器人路径规划起始点的关节角传入 Mech-Vision	工程为 Eye To Hand 模式，并且 Mech-Vision 工程中存在“路径规划”步骤，且需要从机器人端设置“路径规划”步骤的起始点。

机器人位姿为 Float 类型数字。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，返回 **1102** 状态码；否则返回对应的错误码。

102 指令——获取视觉目标点

该指令用在*101 指令——启动 Mech-Vision 工程* 之后，用于获取 Mech-Vision 输出的视觉点，然后将视觉点转换为视觉目标点。

具体转换过程如下所示，即将视觉点包含的位姿转换为对应机器人 TCP。

- 将视觉点包含的位姿绕 Y 轴旋转 180°。
- 识别对应机器人型号的参考坐标系定义是否涉及机器人基座高度，并相应增加垂直方向的偏置。

提示： 102 指令单次接收视觉目标点的数量上限默认为 20。如需要获取的视觉目标点的数量大于 20，需多次执行该指令来获取所有所需的视觉目标点。

发送的指令参数

参数	地址（十进制）
指令码 102	1
Mech-Vision 工程编号	4

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

参数	地址（十进制）
状态码	100
数据传输状态	101
视觉目标点数量	102
保留字段	/
此次获取的所有 TCP	104
此次获取的所有标签	584

状态码

若指令执行正常，则返回 **1100** 状态码；否则返回对应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的视觉目标点。

`1`：表示返回的数据是新的视觉目标点，请读取。

注意： 在读取新返回的数据后，请将该参数重置为 0。

视觉目标点数量

执行该指令获得的视觉目标点数量。

- 如果请求的视觉目标点数量大于等于 Mech-Vision 识别的视觉点数量，按照 Mech-Vision 识别的视觉点数量发送。
- 如果请求的视觉目标点数量小于 Mech-Vision 识别视觉点数量，按照请求数量发送。

保留字段

该字段未使用，默认值为 0。

此次获取的所有 TCP

单个 TCP 所含信息包括空间坐标（XYZ）和欧拉角（ABC）。

此次获取的所有标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前用步骤 label_mapping 将标签映射为整数。如工程中没有标签，则该参数为默认值 0。

103 指令——切换 Mech-Vision 配方

切换 Mech-Vision 工程内的参数配方。

Mech-Vision 中步骤的参数设定可通过切换参数配方调整。

参数配方调整涉及的参数通常包括点云匹配模板、图像匹配模板、感兴趣区域、置信度阈值等。

注意： 需要在执行*101 指令——启动 Mech-Vision 工程* 之前使用该指令。

发送的指令参数

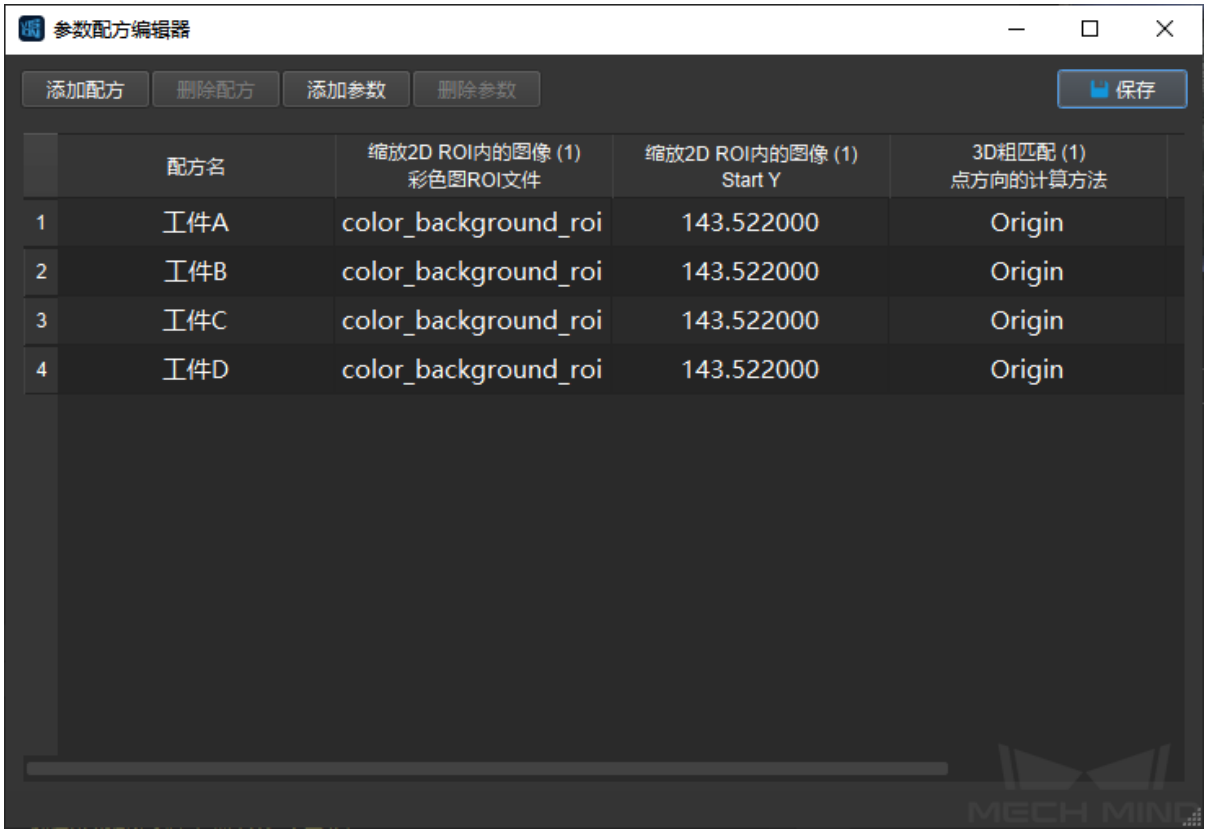
参数	地址（十进制）
指令码 103	1
Mech-Vision 工程编号	4
配方编号	5

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

配方编号

Mech-Vision 工程中配方模板的编号，为正整数。单击 *工程助手 ▸ 参数配方*，进入参数配方编辑器。编号范围：1~99。



返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **1107** 状态码；否则返回对应的错误码。

105 指令——获取 Mech-Vision “路径规划”步骤的结果

在调用 101 指令之后，使用该指令获取 Mech-Vision 中“路径规划”步骤输出的免碰撞抓取路径。

在使用该指令时，Mech-Vision “输出”步骤的 **端口类型**参数需要设置为“预定义（机器人路径）”。

提示： 在调用 105 指令前，请务必将 101 指令的 **预期视觉点数量**设置为 0，以减少调用 105 指令的次数。若 101 指令的 **预期视觉点数量**设置为 1，则每次调用 105 指令只会返回一个路径点，只有多次调用 105 指令才能接收全部路径点。

发送的指令参数

参数	地址（十进制）
指令码 105	1
Mech-Vision 工程编号	4
路径点位姿类型	2

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

路径点位姿类型

该参数用于指定”路径规划“步骤返回的路径点的位姿类型。

- `1`：路径点的位姿将以机器人关节角（JPs）的形式返回。
- `2`：路径点的位姿将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	地址（十进制）
状态码	100
数据传输状态	101
路径点数量	102
“视觉移动”位置	103
此次发送的所有路径点的位姿	104
此次发送的所有路径点的标签	584
此次发送的所有路径点的速度	624

状态码

若指令执行正常，则返回 **1103** 状态码；否则返回对应的错误码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

`1`：表示返回的数据是新的路径点，请读取。

> **注意：**在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数用于表示此次执行该指令返回的路径点个数，取值范围为 0~20。若获取的路径点数多于 20，请多次调用该指令。

“视觉移动”位置

路径规划工具中设置的“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 105 指令中的路径点位姿类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在“输出”步骤前使用“标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

路径规划工具中设置的速度值。

201 指令——启动 Mech-Viz 工程

用于既使用 Mech-Vision 又使用 Mech-Viz 的场景。该指令用于运行 Mech-Viz 工程，启动对应的 Mech-Vision 工程，Mech-Viz 基于 Mech-Vision 的视觉结果规划机器人的运动路径。

请在需要启动 Mech-Viz 工程中勾选 *自动加载*。

Mech-Center 安装目录下的 `tool/viz_project` 文件夹内保存有一些典型应用工程模板，用户可以在模板的基础上进行修改。

用于标准接口的 Mech-Viz 样例工程描述请见*标准接口用 Mech-Viz 样例工程*。

发送的指令参数

参数	地址（十进制）
指令码 201	1
机器人位姿类型	2
机器人位姿	6-17（JPs）或 18-29（法兰位姿）

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Viz，其取值范围为 0~2。

- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	**说明**	**适用场景**
0	0, 0, 0, 0, 0, 0	无需向 Mech-Viz 传入机器人位姿，Mech-Viz 中仿真机器人将从初始位姿 JPs = [0, 0, 0, 0, 0, 0] 开始移动到第一个路径点。	工程为 Eye To Hand 模式。不推荐使用该设定。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的当前关节角和法兰位姿传入 Mech-Viz，Mech-Viz 中仿真机器人将从传入的关节角开始移动到第一个路径点。	工程为 Eye In Hand 模式时，推荐使用该设定。
2	机器人端自定义的关节角	需要将机器人的一个示教点（非当前关节角）传入 Mech-Viz，用于在机器人处于拍照区域外时，提前触发 Mech-Viz 工程规划下一轮路径（如下图），Mech-Viz 中仿真机器人将从传入的示教点开始运动到第一个路径点。	工程为 Eye To Hand 模式时，推荐使用该设定。

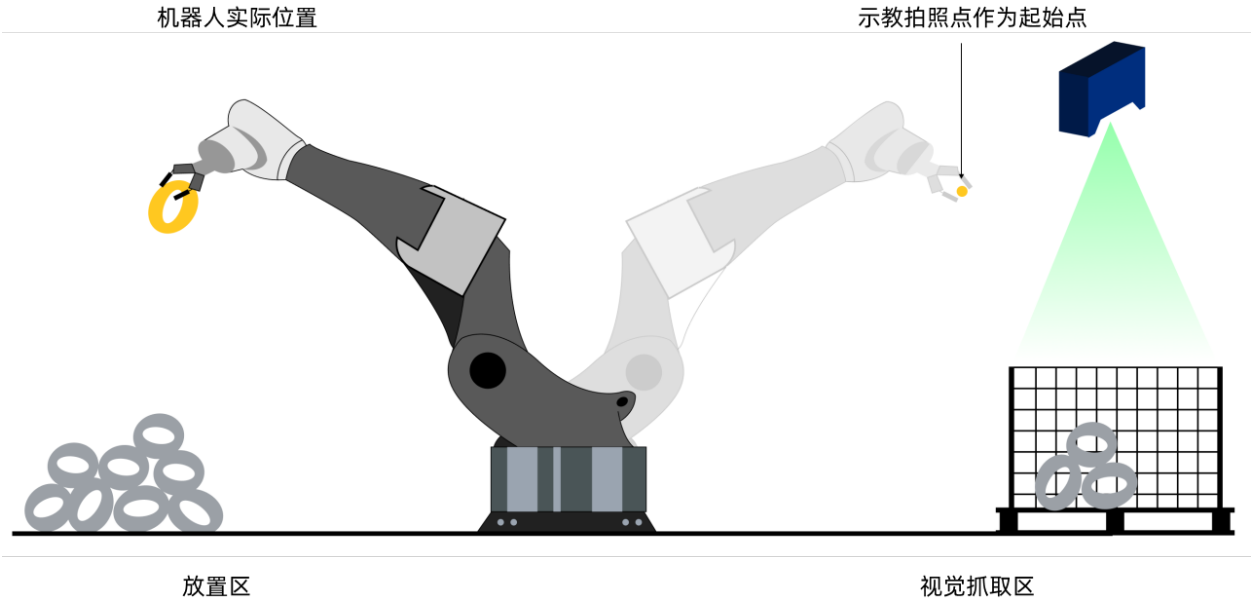
Eye To Hand 模式下应将 **机器人位姿类型**设为 2 的原因：

Eye To Hand 模式下，相机可在机器人回到拍照及抓取区域之前拍照，用于提前规划下一轮抓取路径，从而缩短节拍。

若此时将 **机器人位姿类型**设为 1，即将机器人当前位姿发送给 Mech-Viz 仿真机器人，可能导致仿真机器人与真实机器人轨迹不一致，发生未预知的碰撞，造成危险。

即仿真机器人将直接从当前位姿移动至 Mech-Viz 中第一个移动步骤中设置的位姿，而真实机器人可能在移动至其他位姿后才移动至上述位姿。

所以应将 **机器人位姿类型**参数设为 2。



返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **2103** 状态码；否则返回对应的错误码。

202 指令——停止 Mech-Viz 工程

停止 Mech-Viz 工程的运行。若 Mech-Viz 工程未陷入无限循环或可正常停止，则不需要使用此指令。

发送的指令参数

参数	地址（十进制）
指令码 202	1

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，返回 **2104** 状态码；否则返回对应的错误码。

203 指令——选择 Mech-Viz 分支

该指令用于指定工程将沿哪个分支运行。分支机制通过 branch_by_msg 任务建立，该指令则通过指定该步骤的出口来指定分支。

在执行该指令前请先执行*201 指令——启动 Mech-Viz 工程* 。

Mech-Viz 工程运行至 branch_by_msg 步骤时，将等待该指令指定出口。

发送的指令参数

参数	地址（十进制）
指令码 203	1
分支步骤编号	30
出口号	31

分支步骤编号

该参数用于指定分支选择将在哪个 branch_by_msg 步骤上进行。

该参数应为正整数，即 branch_by_msg 的步骤编号。步骤编号可在步骤参数中读取。

出口号

该参数用于指定工程将沿 branch_by_msg 步骤的哪个出口运行，Mech-Viz 程序将沿该出口继续执行。参数为正整数。

出口号范围：1~99。

提示：

- 出口号为 Mech-Viz 显示的端口号加 1。如端口号为 0，则出口号为 1。
- 出口号为从 1 开始的端口索引号。比如指定的出口为从左往右数第二个端口，那么出口号为 2。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **2105** 状态码；否则返回对应的错误码。

204 指令——设置移动索引

该指令用于设定步骤的索引参数值。该类步骤一般用于连续或单独指定的移动或其他操作。

带有索引参数的步骤包括“按序列移动”、“按阵列移动”、“自定义垛型”、“预设垛型”等。

在执行该指令前，请先执行*201 指令——启动 Mech-Viz 工程* 。

发送的指令参数

参数	地址（十进制）
指令码 204	1
步骤编号	32
索引值	33

步骤编号

该参数指定哪个步骤需要索引参数设定。

该参数值应为正整数，即待索引步骤的步骤编号。步骤编号可在步骤参数中读取。

索引值

下次执行此步骤时应设置的索引值。

发送该指令时，Mech-Viz 中的当前索引值将变为该参数值减 1。

当 Mech-Viz 项目运行到该指令指定的步骤时，Mech-Viz 中的当前索引值将增加 1，成为该参数的值。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **2106** 状态码；否则返回对应的错误码。

205 指令——获取规划路径

在执行*201 指令——启动 Mech-Viz 工程* 后，该指令用于获取 Mech-Viz 规划的路径。

在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

注解： 如果工程中某个移动类步骤的路径点不应发送给机器人，请取消勾选该步骤参数中的“发送路径点”选项。

发送的指令参数

参数	地址（十进制）
指令码 205	1
路径点类型	2

路径点类型

该参数用于指定 Mech-Viz 将返回何种形式的路径点。

- `1`：路径点将以机器人关节角（JPs）的形式返回。
- `2`：路径点将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	地址（十进制）
状态码	100
数据传输状态	101
路径点数量	102
“视觉移动”位置	103
此次发送的所有路径点的位姿	104
此次发送的所有路径点的标签	584
此次发送的所有路径点的速度	624

状态码

若指令执行正常，则返回 **2100** 状态码；否则返回对应的错误码。

提示： 调用该指令时，若 Mech-Viz 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

`1`：表示返回的数据是新的视觉点，请读取。

注意：在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数表示返回的路径点个数，取值范围为 0~20。在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

“视觉移动”位置

“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下步骤组成：“移动 _1”，“移动 _2”，“视觉移动”，“移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 205 指令中的路径点类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前使用步骤 label_mapping 将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

移动类步骤参数中的非零速度参数百分数值。

206 指令——获取 DO 信号列表

该指令用于获取规划的 DO 信号列表。DO 信号列表用于控制多个工具或吸盘分区。

执行该指令前，需要先执行*205 指令* 获取 Mech-Viz 规划路径。

请根据模板工程来部署 Mech-Viz 工程，并在工程中设置对应的吸盘配置文件。模板工程为 Mech-Center 安装目录下的 `tool/viz_project` 文件夹中的 **suction_zone** 工程。

在工程的 set_do_list 步骤的参数中：

- 在“接收对象”下勾选“标准接口”
- 勾选“从视觉移动中获取 DO 列表”
- 在参数栏底部选择需要 DO 信号列表的视觉移动步骤

发送的指令参数

参数	地址（十进制）
指令码 206	1

返回的数据参数

参数	地址（十进制）
状态码	100
DO 信号列表	664

状态码

若指令执行正常，则返回 **2102** 状态码；否则返回对应的错误码。

DO 端口值

共有 64 个 DO 信号值，为整数。

DO 信号值范围：0~999。

占位值：-1。

例如：DO 信号值为 1、3、5、6 。

1	3	5	6	-1	-1	-1	-1	…	-1	-1
第 1 位	第 2 位	第 3 位	第 4 位	第 5 位	第 6 位	第 7 位	第 8 位	…	第 63 位	第 64 位

501 指令——向 Mech-Vision 传入物体尺寸

该指令用于往 Mech-Vision 工程中动态传入物体尺寸。启动 Mech-Vision 工程前请先确认物体尺寸。

Mech-Vision 工程中应有 read_object_dimensions 步骤。该步骤参数 **从参数读取物体尺寸**应设定为 `True`。

发送的指令参数

参数	地址（十进制）
指令码 501	1
Mech-Vision 工程编号	4
[长, 宽, 高]	34

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

长，宽，高

传入 Mech-Vision 工程的物体尺寸。尺寸值将被 read_object_dimensions 步骤读取。

单位：毫米（mm）

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **1108** 状态码；否则返回对应的错误码。

502 指令——向 Mech-Viz 传入 TCP

该指令用于往 Mech-Viz 工程中动态传入机器人 TCP。读取机器人 TCP 的步骤为 outer_move。

请基于模板工程部署 Mech-Viz 工程。模板工程路径为 Mech-Center 安装目录下 `tool/viz_project/outer_move` 。

请将 outer_move 步骤放在工作流程中合适的位置。

该指令需在执行*201 指令——启动 Mech-Viz 工程* 之前执行。

发送的指令参数

参数	地址（十进制）
指令码 502	1
TCP	40

TCP

用于设定 outer_move 步骤路径点的 TCP 数据。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **2107** 状态码；否则返回对应的错误码。

901 指令——获取软件状态

该指令用于检查软件运行状态（Mech-Vision、Mech-Viz、Mech-Center）。

目前该指令只支持检查 Mech-Vision 是否可以开始运行工程。

发送的指令参数

参数	地址（十进制）
指令码 901	1

参数说明：无参数。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

系统自检状态。**1101** 状态码为“Mech-Vision 工程已就绪”；其他状态码为“Mech-Vision 工程未就绪”。目前该指令只能用于检查 Mech-Vision 工程是否已就绪。

999 指令——清除寄存器数据

该指令用于清除寄存器中的数据。

发送的指令参数

参数	地址（十进制）
指令码 999	1

参数说明：无参数。

返回的数据参数

参数	地址（十进制）
状态码	100

状态码

若指令执行正常，则返回 **3103** 状态码；否则返回对应的错误码。

3.3.7 Mitsubishi MC

通过基于 MC 协议的标准接口，梅卡曼德软件系统可以与三菱 PLC 进行通信。

具体示例可以参考 《Mitsubishi MC ——三菱 Q 系列 PLC》。

寄存器使用说明

PLC 使用的数据类型为 MM_Interface 结构体，该结构体需要占用 728 个 D 寄存器。该结构体变量的起始地址需要与 Mech-Vision 中配置的起始地址相同。当 PLC 和 Mech-Vision 使用的起始地址都为 10000 时，各变量的寄存器地址如下图所示。

软元件/标签	当前值	数据类型	类	软元件
⊟ MM_Camera		MM_Interface	VAR_GLOBAL	
Command_Trigger	0	Word[Signed]		D10000
Command	0	Word[Signed]		D10001
Pose_Type	1	Word[Signed]		D10002
Pose_Number	0	Word[Signed]		D10003
Vision_Project_Num	1	Word[Signed]		D10004
Recipe_Num	1	Word[Signed]		D10005
⊞ Joint_Position		FLOAT (Single Precision) [6]		
⊞ TCP_Pose		FLOAT (Single Precision) [6]		
Branch_Name	1	Word[Signed]		D10030
Branch_Exit_Port	1	Word[Signed]		D10031
Index_Name	10	Word[Signed]		D10032
Index_Counter	1	Word[Signed]		D10033
⊞ Ext_InputBoxDim		FLOAT (Single Precision) [3]		
⊞ Ext_Input_Pose		FLOAT (Single Precision) [6]		
Robot_Move_Status	0	Word[Signed]		D10052
⊞ Reserved		Word[Signed] [44]		
Trigger_Acknowledge	0	Word[Signed]		D10097
Notify	0	Word[Signed]		D10098
Heartbeat	1	Word[Signed]		D10099
Status_Code	2100	Word[Signed]		D10100
Status_of_Pose_Sent	1	Word[Signed]		D10101
Number_of_Pose_Sent	6	Word[Signed]		D10102
Index_of_Vision_Picking_Point	3	Word[Signed]		D10103
⊞ Target_Pose		FLOAT (Single Precision) [240]		
⊞ Target_Label		Word[Signed] [40]		
⊞ Speed_Percentage		Word[Signed] [40]		
⊞ Digital_Output		Word[Signed] [64]		

下表为各变量的寄存器相对于起始地址的偏移量及说明。

寄存器地址偏移量	名称	数据类型	说明
0	Command_Trigger	Word[有符号]	触发信号
1	Command	Word[有符号]	指令码
2	Pose_Type	Word[有符号]	位姿类型
3	Pose_Number	Word[有符号]	预期视觉点数量
4	Vision_Project_Num	Word[有符号]	Mech-Vision 工程编号
5	Recipe_Num	Word[有符号]	Mech-Vision 配方编号
6	Joint_Position	Float[单精度][6]	关节角数据
18	TCP_Pose	Float[单精度][6]	法兰位姿数据
30	Branch_Name	Word[有符号]	Mech-Viz 消息分支的步骤编号
31	Branch_Exit_Port	Word[有符号]	Mech-Viz 消息分支步骤的出口号
32	Index_Name	Word[有符号]	Mech-Viz 索引类步骤的步骤编号
33	Index_Counter	Word[有符号]	下次执行此步骤时应设置的索引值
34	Ext_InputBoxDim	Float[单精度][3]	传入 Mech-Vision 工程的物体尺寸（长宽高，单位 mm）
40	Ext_Input_Pose	Float[单精度][6]	传入 Mech-Viz 工程的外部 TCP 数据
52	Robot_Move_Status	Word[有符号]	机器人移动状态
53	Reserved	Word[有符号][44]	预留
97	Trigger_Acknowledge	Word[有符号]	触发确认
98	Notify	Word[有符号]	用户消息
99	Heartbeat	Word[有符号]	心跳值
100	Status_Code	Word[有符号]	状态码
101	Status_of_Pose_Sent	Word[有符号]	发送位姿的状态
102	Number_of_Pose_Sent	Word[有符号]	发送位姿的数量
103	Index_of_Vision_Picking_Point	Word[有符号]	“视觉移动”位置
104	Target_Pose	Float[单精度][240]	目标位姿
584	Target_Label	Word[有符号][40]	标签
624	Speed_Percentage	Word[有符号][40]	速度
664	Digital_Output	Word[有符号][40]	数字输出信号

MC 指令说明

本节介绍基于 MC 协议的标准接口指令。

101 指令——启动 Mech-Vision 工程

在仅使用 Mech-Vision 的场景下，该指令用于启动 Mech-Vision 工程，并执行相机拍照和视觉识别。

发送的指令参数

参数	寄存器地址偏移量
指令码 101	1
Mech-Vision 工程编号	4
预期视觉点数量	3
机器人位姿类型	2
机器人位姿	6-17（JPs）或 18-29（法兰位姿）

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

预期视觉点数量

期望从 Mech-Vision 得到的视觉点数量。视觉点信息包括视觉位姿及对应点云、标签、缩放等。

- 0：从 Mech-Vision 工程取得识别结果中所有的视觉点。
- 大于 0 的整数：从 Mech-Vision 工程取得识别结果中指定数量的视觉点。

- 如果视觉点总数小于该参数值，则取得识别结果中所有视觉点。
- 如果视觉点总数大于等于该参数值，则取得该参数指定数量的视觉点。

提示：获取视觉点的指令是 102 指令。默认设置下，102 指令每次最多能获取 20 个视觉点，若需要获取的视觉点数量大于 20，则需重复调用 102 指令。

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Vision，其取值范围为 0~3。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	**说明**	**适用场景**
0	0, 0, 0, 0, 0, 0	无需向 Mech-Vision 传入机器人位姿	工程为 Eye To Hand 模式。若 Mech-Vision 工程中使用“路径规划”步骤，则路径规划的起始点为路径规划工具中设置的 Home 点。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的关节角和法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，除桁架机器人外的大多数机器人适用该设定。
2	机器人的当前法兰位姿	需要将机器人的当前法兰位姿传入 Mech-Vision	工程为 Eye In Hand 模式，机器人无关节角数据，仅有法兰位姿数据（如桁架机器人）。
3	机器人路径规划起始点的关节角	需要将机器人路径规划起始点的关节角传入 Mech-Vision	工程为 Eye To Hand 模式，并且 Mech-Vision 工程中存在“路径规划”步骤，且需要从机器人端设置“路径规划”步骤的起始点。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，返回 **1102** 状态码；否则返回对应的错误码。

102 指令——获取视觉目标点

该指令用在*101 指令——启动 Mech-Vision 工程* 之后，用于获取 Mech-Vision 输出的视觉点，然后将视觉点转换为视觉目标点。

具体转换过程如下所示，即将视觉点包含的位姿转换为对应机器人 TCP。

- 将视觉点包含的位姿绕 Y 轴旋转 180°。
- 根据机器人型号的参考坐标系定义，判断是否涉及机器人基座高度，从而确定是否增加相应垂直方向的偏置。

提示： 102 指令单次接收视觉目标点的数量上限默认为 20。如需要获取的视觉目标点的数量大于 20，需多次执行该指令来获取所有所需的视觉目标点。

发送的指令参数

参数	寄存器地址偏移量
指令码 102	1
Mech-Vision 工程编号	4

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

返回的数据参数

参数	寄存器地址偏移量
状态码	100
数据传输状态	101
视觉目标点数量	102
保留字段	/
此次获取的所有 TCP	104
此次获取的所有标签	584

状态码

若指令执行正常，则返回 **1100** 状态码；否则返回对应的错误码。

调用该指令时，若 Mech-Vision 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的视觉目标点。

`1`：表示返回的数据是新的视觉目标点，请读取。

> **注意：** 在读取新返回的数据后，请将该参数重置为 0。

视觉目标点数量

执行该指令获得的视觉目标点数量。

- 如果请求的视觉目标点数量大于等于 Mech-Vision 识别的视觉点数量，按照 Mech-Vision 识别的视觉点数量发送。
- 如果请求的视觉目标点数量小于 Mech-Vision 识别视觉点数量，按照请求数量发送。

保留字段

该字段未使用，默认值为 0。

此次获取的所有 TCP

单个 TCP 所含信息包括空间坐标（XYZ）和欧拉角（ABC）。

此次获取的所有标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前用“标签映射”步骤将标签映射为整数。如工程中没有标签，则该参数为默认值 0。

103 指令——切换 Mech-Vision 配方

该指令用于切换 Mech-Vision 工程内的参数配方，需要在执行*101 指令——启动 Mech-Vision 工程* 之前使用。

发送的指令参数

参数	寄存器地址偏移量
指令码 103	1
Mech-Vision 工程编号	4
配方编号	5

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

配方编号

Mech-Vision 工程中配方参数的编号，为正整数。单击 `工程助手 ▸ 参数配方`，进入参数配方编辑器，可查看配方编号。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **1107** 状态码；否则返回对应的错误码。

105 指令——获取 Mech-Vision “路径规划”步骤的结果

在调用*101 指令——启动 Mech-Vision 工程* 之后，使用该指令获取 Mech-Vision 中“路径规划”步骤输出的免碰撞规划路径。

在使用该指令时，Mech-Vision “输出”步骤的 **端口类型**参数需要设置为“预定义（机器人路径）”。

提示：在调用 105 指令前，请务必将 101 指令的 **预期视觉点数量**设置为 0，以减少调用 105 指令的次数。若 101 指令的 **预期视觉点数量**设置为 1，则每次调用 105 指令只会返回一个路径点，只有多次调用 105 指令才能接收全部路径点。

发送的指令参数

参数	寄存器地址偏移量
指令码 105	1
Mech-Vision 工程编号	4
路径点位姿类型	2

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

路径点位姿类型

该参数用于指定”路径规划“步骤返回的路径点的位姿类型。

- `1`：路径点的位姿将以机器人关节角（JPs）的形式返回。
- `2`：路径点的位姿将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	寄存器地址偏移量
状态码	100
数据传输状态	101
路径点数量	102
“视觉移动”位置	103
此次发送的所有路径点的位姿	104
此次发送的所有路径点的标签	584
此次发送的所有路径点的速度	624

状态码

若指令执行正常，则返回 **1103** 状态码；否则返回对应的错误码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

`1`：表示返回的数据是新的路径点，请读取。

> **注意：**在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数用于表示此次执行该指令返回的路径点个数，取值范围为 0~20。若获取的路径点数多于 20，请多次调用该指令。

“视觉移动”位置

路径规划工具中设置的“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 105 指令中的路径点位姿类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在“输出”步骤前使用“标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

路径规划工具中设置的速度值。

201 指令——启动 Mech-Viz 工程

该指令用于既有 Mech-Vision 又有 Mech-Viz 的场景，用于启动 Mech-Viz 工程，调用相应的 Mech-Vision 工程，并规划移动路径。

发送的指令参数

参数	寄存器地址偏移量
指令码 201	1
机器人位姿类型	2
机器人位姿	6-17（JPs）或 18-29（法兰位姿）

机器人位姿类型、机器人位姿

- **机器人位姿类型**参数指定真实机器人的位姿将以何种形式传入 Mech-Viz，其取值范围为 0~2。
- **机器人位姿**参数值取决于 **机器人位姿类型**参数值。

下表为两参数取值的关系及说明。

机器人位姿类型参数值	**机器人位姿**参数值	说明	适用场景
0	0, 0, 0, 0, 0, 0	无需向 Mech-Viz 传入机器人位姿，Mech-Viz 中仿真机器人将从初始位姿 JPs = [0, 0, 0, 0, 0, 0] 开始移动到第一个路径点。	工程为 Eye To Hand 模式。不推荐使用该设定。
1	机器人当前关节角 + 当前法兰位姿	需要将机器人的当前关节角和法兰位姿传入 Mech-Viz，Mech-Viz 中仿真机器人将从传入的关节角开始移动到第一个路径点。	工程为 Eye In Hand 模式时，推荐使用该设定。
2	机器人端自定义的关节角	需要将机器人的一个示教点（非当前关节角）传入 Mech-Viz，用于在机器人处于拍照区域外时，提前触发 Mech-Viz 工程规划下一轮路径（如下图），Mech-Viz 中仿真机器人将从传入的示教点开始运动到第一个路径点。	工程为 Eye To Hand 模式时，推荐使用该设定。

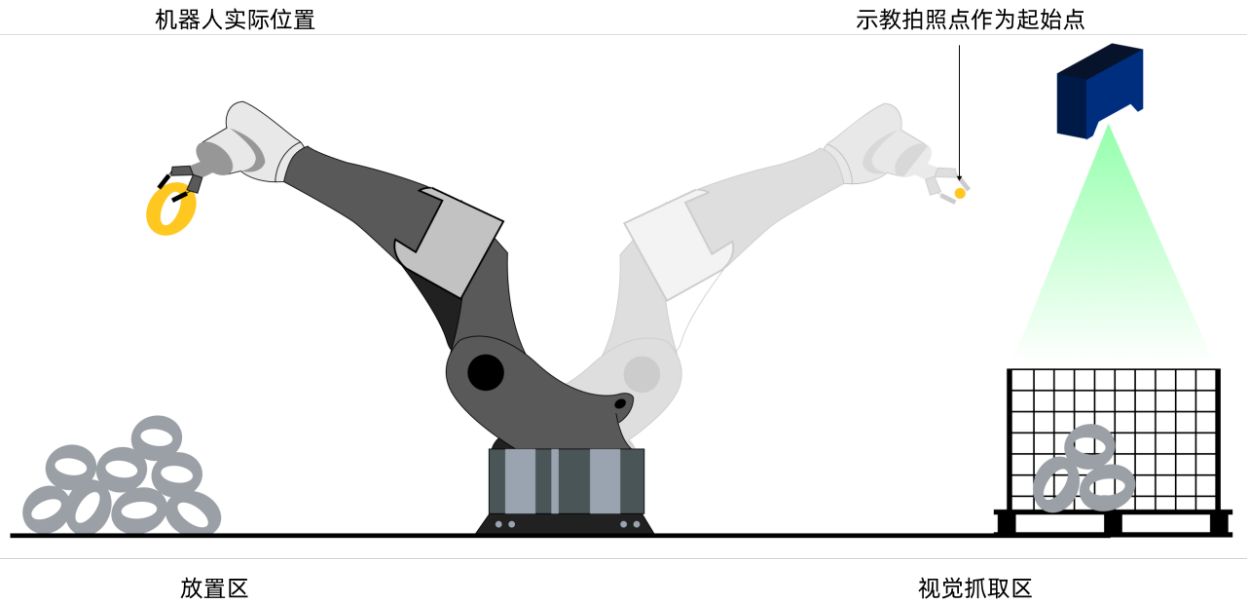
Eye To Hand 模式下应将 **机器人位姿类型**设为 2 的原因：

Eye To Hand 模式下，相机可在机器人回到拍照及抓取区域之前拍照，用于提前规划下一轮抓取路径，从而缩短节拍。

若此时将 **机器人位姿类型**设为 1，即将机器人当前位姿发送给 Mech-Viz 仿真机器人，可能导致仿真机器人与真实机器人轨迹不一致，发生未预知的碰撞，造成危险。

即仿真机器人将直接从当前位姿移动至 Mech-Viz 中第一个移动步骤中设置的位姿，而真实机器人可能在移动至其他位姿后才移动至上述位姿。

所以应将 **机器人位姿类型**参数设为 2。



返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **2103** 状态码；否则返回对应的错误码。

202 指令——停止 Mech-Viz 工程

该指令用于停止运行 Mech-Viz 工程。如果 Mech-Viz 工程不是死循环，或可以正常停止，则不需要使用该指令。

发送的指令参数

参数	寄存器地址偏移量
指令码 202	1

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，返回 **2104** 状态码；否则返回对应的错误码。

203 指令——选择 Mech-Viz 分支

当 Mech-Viz 工程中有“消息分支”步骤时，该指令可控制 Mech-Viz 工程中的“消息分支”步骤走指定的出口。在执行该指令前，请先执行*201 指令——启动 Mech-Viz 工程* 。Mech-Viz 运行到分支步骤时会等待该指令发送的分支出口号。

发送的指令参数

参数	寄存器地址偏移量
指令码 203	1
分支步骤编号	30
出口号	31

分支步骤编号

消息分支的步骤编号，为正整数。

出口号

消息分支的出口号，为正整数。

出口号为从 1 开始的端口索引号。例如指定的出口为从左往右数第二个端口，那么出口号为 2。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **2105** 状态码；否则返回对应的错误码。

204 指令——设置移动索引

该指令用于设定步骤的索引参数值。带有索引参数的步骤包括“按序列移动”、“按阵列移动”、“自定义垛型”、“预设垛型”等，一般用于连续或单独指定的移动或其他操作。

发送的指令参数

参数	寄存器地址偏移量
指令码 204	1
步骤编号	32
索引值	33

步骤编号

索引类步骤的步骤编号，为正整数。

索引值

下次执行此步骤时应设置的索引值。

发送该指令时，Mech-Viz 中的当前索引值将变为该参数值减 1。

当 Mech-Viz 工程运行到该指令指定的步骤时，Mech-Viz 中的当前索引值将增加 1，成为该参数的值。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **2106** 状态码；否则返回对应的错误码。

205 指令——获取规划路径

该指令用在*201 指令——启动 Mech-Viz 工程* 后，用于获取 Mech-Viz 规划的路径。

在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

注意： 如果工程中某个移动类步骤的路径点不应发送给机器人，请取消勾选该步骤参数中的“发送路径点”选项。

发送的指令参数

参数	寄存器地址偏移量
指令码 205	1
路径点类型	2

路径点类型

该参数用于指定 Mech-Viz 将返回路径点的位姿类型。

- `1`：路径点将以机器人关节角（JPs）的形式返回。
- `2`：路径点将以机器人工具位姿（TCP）的形式返回。

返回的数据参数

参数	寄存器地址偏移量
状态码	100
数据传输状态	101
路径点数量	102
“视觉移动”位置	103
此次发送的所有路径点的位姿	104
此次发送的所有路径点的标签	584
此次发送的所有路径点的速度	624

状态码

若指令执行正常，则返回 **2100** 状态码；否则返回对应的错误码。

> **注意：** 调用该指令时，若 Mech-Viz 未返回结果，则默认等待 10 秒。若发生超时，则返回超时错误状态码。

数据传输状态

该参数用于显示返回的数据是否是新的路径点。

`1`：表示返回的数据是新的视觉点，请读取。

> **注意：** 在读取新返回的数据后，请将该参数重置为 0。

路径点数量

该参数表示返回的路径点个数，取值范围为 0~20。在使用默认设置时，该指令每次最多只能获取 20 个规划的路径点，因此，若获取的路径点数多于 20，则可以多次调用该指令。

“视觉移动”位置

“视觉移动”路径点在整个路径中的位置。

例如，如果规划路径由以下步骤组成：“定点移动 _1”，“定点移动 _2”，“视觉移动”，“定点移动 _3”，则“视觉移动”位置为 3。

如果路径中无“视觉移动”，则该参数值为 0。

此次发送的所有路径点的位姿

三维坐标及欧拉角，或 JPs 关节角。类型由 205 指令中的路径点类型决定。

此次发送的所有路径点的标签

位姿对应的整数标签。如果在 Mech-Vision 工程中，标签为字符串，请在输出前使用“标签映射”步骤将标签映射为整数。如果工程中没有标签，则该参数为默认值 0。

此次发送的所有路径点的速度

移动类步骤参数中的非零速度参数百分数值。

206 指令——获取 DO 信号列表

该指令用于获取规划的 DO 信号列表。DO 信号列表用于控制多个工具或吸盘分区。在执行该指令前，请先执行*205 指令——获取规划路径* 。

在工程的“设置多个 DO”步骤的参数中：

- 在“接收对象”下勾选“标准接口”。
- 勾选“从视觉移动中获取 DO 列表”。
- 在参数栏底部选择需要 DO 信号列表的视觉移动步骤。

发送的指令参数

参数	寄存器地址偏移量
指令码 206	1

返回的数据参数

参数	寄存器地址偏移量
状态码	100
DO 信号列表	664

状态码

若指令执行正常，则返回 **2102** 状态码；否则返回对应的错误码。

DO 信号列表

该指令返回 64 个 DO 端口号，其中有效的端口号为正整数，范围为 0~999；无效的端口号为 -1（作为占位值）。

例如，在下表中，返回的有效 DO 端口号为 1、3、5、6，即表示将上述端口号对应的值置为 1。

1	3	5	6	-1	-1	-1	-1	…	-1	-1
第 1 位	第 2 位	第 3 位	第 4 位	第 5 位	第 6 位	第 7 位	第 8 位	…	第 63 位	第 64 位

501 指令——向 Mech-Vision 传入物体尺寸

该指令用于往 Mech-Vision 工程中动态传入物体尺寸。启动 Mech-Vision 工程前请先确认物体尺寸。

Mech-Vision 工程中应有“读取物体尺寸”步骤，且勾选 **从参数中读取物体尺寸**参数。

发送的指令参数

参数	寄存器地址偏移量
指令码 501	1
Mech-Vision 工程编号	4
[长, 宽, 高]	34

Mech-Vision 工程编号

Mech-Vision 工程编号可在 Mech-Vision 工程列表窗口中查看，工程名称前的数字表示工程编号。

长，宽，高

传入 Mech-Vision 工程的物体尺寸。尺寸值将被“读取物体尺寸”步骤读取。

单位：毫米（mm）。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **1108** 状态码；否则返回对应的错误码。

502 指令——向 Mech-Viz 传入 TCP

该指令用于往 Mech-Viz 工程中动态传入机器人 TCP，需在执行*201 指令——启动 Mech-Viz 工程* 之前执行。读取机器人 TCP 的步骤为 outer_move。

请基于模板工程部署 Mech-Viz 工程。模板工程路径为梅卡曼德系统软件安装目录下 `tool/viz_project/outer_move` 。请将“外部移动”步骤放在工作流程中合适的位置。

发送的指令参数

参数	寄存器地址偏移量
指令码 502	1
TCP	40

TCP

用于设定“外部移动”步骤路径点的 TCP 数据。

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

若指令执行正常，则返回 **2107** 状态码；否则返回对应的错误码。

901 指令——获取软件状态

该指令用于检查软件（Mech-Vision、Mech-Viz、Mech-Center）运行状态。目前该指令只能用于检查 Mech-Vision 工程是否已就绪。

发送的指令参数

参数	寄存器地址偏移量
指令码 901	1

返回的数据参数

参数	寄存器地址偏移量
状态码	100

状态码

系统自检状态。**1101** 状态码为“Mech-Vision 工程已就绪”；其他状态码为“Mech-Vision 工程未就绪”。

3.3.8 附录

使用 Mech-Viz 检测碰撞

配合 `XXX/Mech-Center/tool/viz_project/check_collision` 文件中的 check_collision.viz 工程使用，注意以下几点：

1. check_collision 工程只是一个示例工程，工程里除了移动类相关的步骤，都是必须的，不可删除和不可更改其相对位置。其中的机器人模型请选择实际使用的型号。
2. 可根据实际增删其中的移动类相关的步骤。发送的位姿数量与移动类相关的步骤数量一致。
3. 若需要 Home 位置，在机器人端调用触发拍照指令之前调用一次设置位姿命令即可。

使用 Mech-Viz 吸盘分区功能

配合 `XXX/Mech-Center/tool/viz_project/suction_zone` 文件中的 suction_zone.viz 工程使用，注意以下几点：

1. suction_zone.viz 工程只是一个示例工程，工程里除了移动类相关的步骤，都是必须的，不可删除和不可更改其相对位置。其中的机器人模型请选择实际使用的型号。
2. 可根据实际增删其中的移动类相关的步骤。发送的位姿数量与移动类相关的步骤数量一致。
3. 若需要 Home 位置，在机器人端调用触发拍照指令之前调用一次设置位姿命令即可。
4. 使用前需要配置好吸盘文件。
5. 机器人端需先调用触发拍照命令，再调用获取 DO 信号列表命令。

发送识别物体标签功能

给机器人发送的标签字段是一个标签码（整数表示）。需要在 Mech-Vision 工程中建立标签映射。

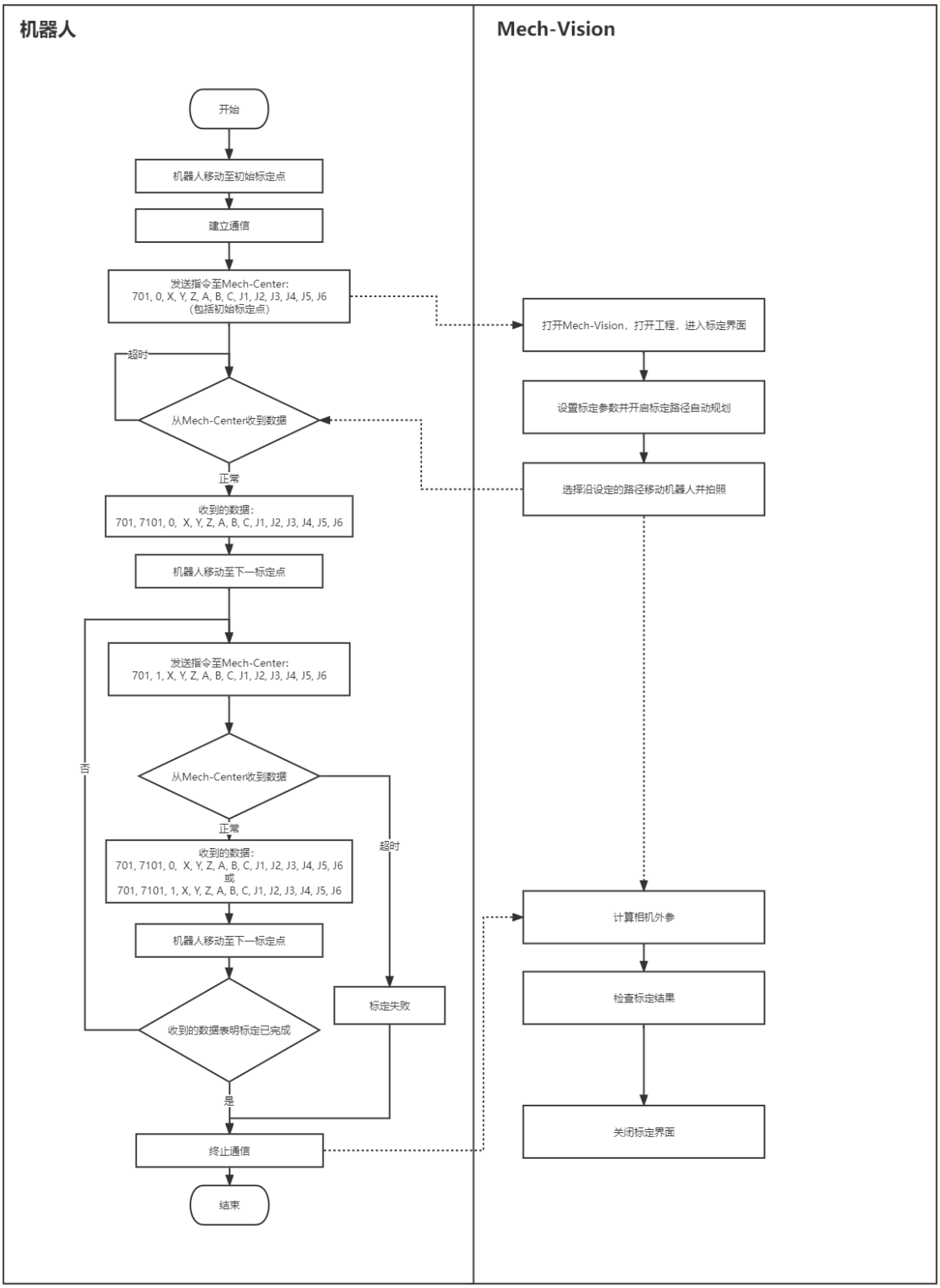

JSON编辑器:请在下方输入数对字符串。
映射 *
small
large
medium
1
2
3

标签映射文件格式如下：


{
"large": "2",
"medium": "3",
"small": "1"
}

- large 、small、medium 是标签字符串。
- 1、2、3 是标签码。

机器人自动标定程序流程



添加相机曝光完成

PROFINET 和 EtherNet/IP 协议的标准接口提供 **相机曝光完成**信号，主要用于优化系统节拍。当 Mech-Vision 工程运算时间较长，机器人需要在相机曝光完成之后立即移动位置。**相机曝光完成**信号的产生需要在 Mech-Vision 工程中进行一些必要的设置。

1. 在 Mech-Vision 工程中添加一个 notify_vision ，连接在 capture_images_from_camera 的控制流上。设置 capture_images_from_camera 的 运行标志 ▸ 有输出时触发控制流为 **True** 。

2. 设置 notify_vision 的服务名称为 **Standard Interface Notify** ，消息内容为 **1001** （1001 的消息内容不允许更改）。

3. 运行 Mech-Vision 工程，当相机曝光完成时，PLC/机器人端会收到 Exposure_Complete 信号。收到 Exposure_Complete 信号后，请使用 Reset_Exposure 信号将 Exposure_Complete 信号复位。若 10s 内系统没有收到复位信号，将提示报错 **Mech-Center 数据确认信号超时**。

3.3.9 标准接口状态码及错误排查

概览

使用梅卡曼德标准接口服务时，指令的返回信息中包含该指令的执行情况（状态码），状态码包括正常执行完成情况和异常报警情况，用户可以根据状态码进行错误处理。

Mech-Vision：

- 1001~1099：Mech-Vision 相关错误码
- 1100~1199：Mech-Vision 相关正常状态码

Mech-Viz：

- 2001~2099：Mech-Viz 相关错误码
- 2100~2199：Mech-Viz 相关正常状态码

Mech-Center：

- 3001~3099：Mech-Center 相关错误码
- 3100~3199：Mech-Center 相关正常状态码

机器人：

- 4001~4099：机器人相关错误码
- 4100~4199：机器人相关正常状态码

外参标定：

- 7001~7099：外参标定相关错误码
- 7100~7199：外参标定相关正常状态码

Mech-Vision

Mech-Vision 错误码

错误码	含义
1001	Mech-Vision：工程未注册或工程未设置自动加载
1002	Mech-Vision：无视觉结果
1003	Mech-Vision：ROI 内无点云
1004	Mech-Vision：参数设置失败
1005	Mech-Vision：启动 Mech-Vision 时命令参数无效
1006	Mech-Vision：无效的位姿数据
1007	Mech-Vision：正在计算中
1008	错误码未使用
1009	Mech-Vision：位姿和运动参数数量不一致
1010	Mech-Vision：位姿和标签数量不一致
1011	Mech-Vision：工程号不存在
1012	Mech-Vision：参数配方编号超限
1013	Mech-Vision：参数配方不存在

下页继续

表 1 – 续上页

1014	Mech-Vision：配方设置失败
1015	Mech-Vision：工程运行错误
1016	Mech-Vision：深度学习服务器启动失败
1017	Mech-Vision：无效的标签映射
1018	Mech-Vision：视觉点数量错误
1019	Mech-Vision：执行超时
1020	Mech-Vision：未执行
1021	Mech-Vision：设置物体尺寸失败，请确认步骤 read_object_dimensions 是否存在于工程中
1022	Mech-Vision：设置物体尺寸数值无效
1023	Mech-Vision：相机连接失败
1024	Mech-Vision：位姿数量与自定义输出数据数量不匹配
1025	Mech-Vision：正在使用虚拟相机，数据已弃用
1026	Mech-Vision：无效位姿类型
1028	Mech-Vision：路径规划失败
1030	Mech-Vision：机器人无法到达视觉位姿点
1031	Mech-Vision：机器人关节角计算失败
1032	Mech-Vision：轨迹不可达
1033	Mech-Vision：移动奇异点错误
1034	Mech-Vision：直线运动规划失败
1035	Mech-Vision：无效抓取点
1036	Mech-Vision：检测到机器人自体碰撞
1037	Mech-Vision：检测到机器人与物体间碰撞
1038	Mech-Vision：点云碰撞点数大于阈值，检测到碰撞
1039	Mech-Vision：点云碰撞面积大于阈值，检测到碰撞
1040	Mech-Vision：点云碰撞体积大于阈值，检测到碰撞
1044	Mech-Vision：视觉移动步骤未收到视觉位姿
1046	Mech-Vision：无效的末端工具

Mech-Vision 正常状态码

正常码	含义
1100	Mech-Vision：成功获取视觉点
1101	Mech-Vision：已就绪
1102	Mech-Vision：触发工程成功
1103	Mech-Vision: 成功获取规划路径（执行 105 指令成功后的返回值）
1107	Mech-Vision：配方切换成功
1108	Mech-Vision：设置物体尺寸成功

Mech-Viz

Mech-Viz 错误码

错误码	含义
2001	Mech-Viz：软件未注册
2002	Mech-Viz：工程运行中
2003	Mech-Viz：未收到 Mech-Vision 视觉结果
2004	Mech-Viz：无法到达 Mech-Vision 传入的视觉点
2005	Mech-Viz：机器人关节角计算失败
2006	Mech-Viz：启动 Mech-Viz 时命令参数无效
2007	Mech-Viz：运动路径规划失败
2008	Mech-Viz：工程运行错误
2009	Mech-Viz：未提供工具位姿（TCP）
2010	Mech-Viz：路径不可达
2011	Mech-Viz：未提供 DO 列表
2012	Mech-Viz：无效的位姿类型
2013	Mech-Viz：无效的位姿数据
2014	Mech-Viz：工程未设置
2015	错误码未使用
2016	Mech-Viz：设置步骤参数失败
2017	Mech-Viz：停止执行失败
2018	Mech-Viz：无效的分支出口号
2019	Mech-Viz：设置分支失败，请确认该分支步骤编号是否存在
2020	Mech-Viz：奇异点运动失败
2021	Mech-Viz：直线运动规划失败
2022	Mech-Viz：未执行
2023	Mech-Viz：工程文件异常
2024	Mech-Viz：无效的分支名
2025	Mech-Viz：执行超时
2026	Mech-Viz：无效的索引类步骤名
2027	Mech-Viz：无效的索引值
2028	Mech-Viz：设置索引失败，请检查带索引参数的步骤是否存在于工程中
2029	Mech-Viz：外部移动设置失败
2030	Mech-Viz：无效的视觉点
2031	Mech-Viz：机器人本体碰撞
2032	Mech-Viz：场景碰撞
2033	Mech-Viz：点云碰撞点数超限
2034	Mech-Viz：点云碰撞面积超限
2035	Mech-Viz：点云碰撞体积超限
2036	Mech-Viz：视觉识别未拍照
2037	Mech-Viz：视觉识别无结果
2038	Mech-Viz：视觉识别 ROI 内无点云
2039	Mech-Viz：无可供规划的视觉点
2040	Mech-Viz：复用视觉结果存在规划失败路径
2041	Mech-Viz：获取步骤参数失败
2042	Mech-Viz：获取视觉移动规划结果失败

下页继续

表 2 – 续上页

2043	Mech-Viz：获取视觉目标点自定义数据失败
2044	Mech-Viz：视觉服务未注册
2045	Mech-Viz：无效的末端工具

Mech-Viz 正常状态码

正常码	含义
2100	Mech-Viz：执行成功
2101	Mech-Viz：停止执行成功
2102	Mech-Viz：发送 DO 列表成功
2103	Mech-Viz：启动成功
2104	Mech-Viz：已成功停止
2105	Mech-Viz：分支设置成功
2106	Mech-Viz：索引设置成功
2107	Mech-Viz：外部传入位姿设置成功
2108	Mech-Viz：成功设置 Mech-Viz 步骤的参数值
2109	Mech-Viz：成功读取 Mech-Viz 步骤的参数值

Mech-Center

Mech-Center 错误码

错误码	含义
3001	Mech-Center：非法指令
3002	Mech-Center：接口指令长度或格式错误
3003	Mech-Center：客户端断连
3004	Mech-Center：服务端断连
3005	Mech-Center：调用 Mech-Vision 超时
3006	Mech-Center：未知错误
3007	Mech-Center：数据确认信号超时
3008	Mech-Center：配置 ID 不存在，参数无法被读取/设置

Mech-Center 正常状态码

正常码	含义
3100	Mech-Center：客户端连接正常
3101	Mech-Center：服务端连接正常
3102	Mech-Center：等待客户端连接
3103	Mech-Center：数据缓存清零成功

机器人

机器人错误码

错误码	含义
4001	无效的机器人类型
4002	机器人欧拉角类型不支持
4003	机器人服务未注册
4004	机器人参数缺失
4005	连接机器人失败，请检查机器人 IP 地址和网络配置
4006	机器人烧录程序版本不匹配，请重新下载

机器人正常状态码

正常码	含义
4100	机器人服务注册成功
4101	连接机器人成功
4102	机器人断连
4103	用户关闭机器人服务

外参标定

外参标定错误码

错误码	含义
7001	标定：参数错误
7002	标定：Mech-Vision 没有输出标定位姿
7003	标定：机器人到达标定点失败

外参标定正常状态码

正常码	含义
7100	标定：机器人到达标定点成功
7101	标定：Mech-Vision 输出标定位姿正常

Mech-Vision 错误排查

1001

Mech-Vision：工程未注册或工程未设置自动加载

错误原因：

- Mech-Vision 软件未打开，或工程未打开
- 工程未勾选 *自动加载*，导致没有工程编号。
- 内部通信异常或存在同名工程，导致注册 Mech-Vision 工程服务失败。

排查步骤：

- 确保 Mech-Vision 软件已打开，调用的工程已打开。
- 确保工程已勾选 *自动加载*。
- 确保不存在同名工程，并尝试重启 Mech-Vision 软件。

1002

Mech-Vision：无视觉结果

错误原因：

- 调用的 Mech-Vision 工程已成功执行，但输出中 poses 端口数据为空。可能原因包括实例分割的置信度阈值过高、场景中没有匹配的物体、ROI 设置不当、点云质量低、过滤设置不当等。
- 在 Mech-Vision 工程的输出步骤中，poses 端口不存在，可能输出步骤的端口类型设置为自定义，但没有创建 poses 端口名称。

排查步骤：

- 从 procedure_out 步骤的 poses 端口向上检查 Mech-Vision 工程数据流。
- 检查 procedure_out 步骤的端口类型参数和端口名称。

1003

Mech-Vision：ROI 内无点云

错误原因：

- 调用的 Mech-Vision 工程已成功执行，但 3D ROI 中没有点云。

排查步骤：

- 请检查在点云上设置 3D ROI 的步骤中的 ROI 设置。部分项目中，可以根据该错误判断料框是否到位，或料框是否为空等，所以该报错不一定是工程错误。

1004

Mech-Vision：参数设置失败

1005

Mech-Vision：启动 Mech-Vision 时命令参数无效

错误原因：

- 客户端调用 101 指令触发 Mech-Vision 工程时，机器人位姿类型参数值设置错误。机器人位姿类型参数取值范围为 0~3。

排查步骤：

- 在客户端接口程序中，检查 101 指令设置的机器人位姿类型参数取值是否正确。

1006

Mech-Vision：无效的位姿数据

错误原因：

- 机器人发送 101 指令设置的机器人位姿参数值不足六位。机器人位姿默认为 6 轴机器人位姿。如果机器人是 4 轴或 5 轴机器人，请用 0 填写剩余字段。

排查步骤：

- 检查客户端接口程序。101 指令发送的机器人位姿数据应由 6 个数字组成。

1007

Mech-Vision：正在计算中

错误原因：

- 当 Mech-Vision 工程仍在运行时，客户端接口程序再次调用 101 指令以尝试触发相同的 Mech-Vision 工程。
- Mech-Vision 允许同时运行多个工程，但同个 Mech-Vision 工程在运行时不能被重复启动。

排查步骤：

- 检查客户端接口程序，检查程序中设置的 Mech-Vision 工程编号是否正确。
- 检查程序中是否存在同一 Mech-Vision 工程在短时间内被重新调用的情况。

1008

错误码未使用

1009

Mech-Vision：位姿和运动参数数量不一致

错误原因：

- 此错误通常发生在视觉结果为机器人运动路径的工程中（如涂胶），Mech-Vision 工程输出的运动参数与路径上目标点数量不匹配。

排查步骤：

暂无

1010

Mech-Vision：位姿和标签数量不一致

错误原因：

- Mech-Vision 工程输出的标签与视觉点数量不匹配。

排查步骤：

- 检查 Mech-Vision 工程数据流中的位姿和标签。排查导致位姿和标签数量不一致的原因。

1011

Mech-Vision：工程号不存在

错误原因：

- Mech-Vision 的工程列表中不存在 101 指令参数中设置的工程编号。
  - 例如工程列表只有一个工程时，如果 101 指令调用 2 号工程，将引发该错误。

排查步骤：

- 检查项目使用的 Mech-Vision 工程是否都选择了 *自动加载*。
- 检查客户端接口程序中触发的 Mech-Vision 工程编号是否正确。

1012

Mech-Vision：参数配方编号超限

错误原因：

- 客户端接口程序调用 103 指令设置 Mech-Vision 工程配方，调用的配方编号没有对应的配方。
  - 例如：Mech-Vision 工程配置的配方只有两个，此时若客户端接口程序调用编号为 3 的配方，则报该错误。

排查步骤：

- 检查 Mech-Vision 工程配方设置是否正确。
- 检查客户端接口程序 103 指令设置的配方编号是否正确。

1013

Mech-Vision：参数配方不存在

错误原因：

- 客户端接口程序调用 103 指令设置 Mech-Vision 工程配方，但 Mech-Vision 工程未配置任何配方，则报该错误。

排查步骤：

- 检查 Mech-Vision 工程的配方设置，确保项目需要的配方及编号都设置正确。
- 若项目无需切换配方，则客户端接口程序不使用切换配方的功能。

1014

Mech-Vision：配方设置失败

错误原因：

- Mech-Center 与 Mech-Vision 软件通信异常，未成功设置 Mech-Vision 配方。

排查步骤：

- 在 Mech-Vision 软件的日志窗口中查看具体错误原因。

1015

Mech-Vision：工程运行错误

错误原因：

- Mech-Vision 工程运行出错，出现 Mech-Vision 错误码 CV-Exxxx 或其他 Mech-Center 未具体解析的报错。发生该错误时，Mech-Vision 工程没有执行完毕即被中止。

排查步骤：

- 检查 Mech-Vision 界面的报错信息，根据报错信息排查 Mech-Vision 工程问题。

1016

Mech-Vision：深度学习服务器启动失败

错误原因：

- 被触发的 Mech-Vision 工程包含深度学习模块，但深度学习服务器未启动。对应的 Mech-Vision 错误码为：DL-E0201 深度学习服务器未启动。

排查步骤：

- 检查 Mech-Vision 工程是否为首次运行，检查模型的加载时间是否过长。如果需要设置模型预加载，请在深度学习步骤的参数中勾选 **自动预加载模型**。
- 检查深度学习环境是否已正确安装。

1017

Mech-Vision：无效的标签映射

错误原因：

- Mech-Vision 工程输出的标签不是 INT 数据类型。使用标准接口时，Mech-Vision 工程输出的标签需要映射为 INT 数据格式，否则报该错误。

排查步骤：

- 检查 Mech-Vision 中的数据流，如果输入到步骤 procedure_out 的标签不是正整数，请使用步骤 label_mapping 将标签映射为正整数。

1018

Mech-Vision：视觉点数量错误

错误原因：

- 客户端接口程序调用 101 指令启动 Mech-Vision 工程，设置的预期视觉点数量参数值超过了接口数据长度限制。

排查步骤：

- 数据长度可以在 Mech-Vision 工具栏的 机器人与接口配置中设置。在通信配置界面的 **高级设置**下设置单次发送的最大视觉点（位姿）数。范围：[1, 30]。
- 确保参数“预期视觉点数”小于上述设置的值。

1019

Mech-Vision：执行超时

错误原因：

- 从调用 102 指令获取 Mech-Vision 视觉结果开始计时，Mech-Vision 工程在规定时间内未执行结束，则报该错误。
- 超时时限可以在 Mech-Vision 工具栏的 机器人与接口配置的 **高级设置**下进行设置，默认为 10s。

排查步骤：

- 检查客户端接口程序，在调用 102 指令获取 Mech-Vision 视觉结果之前，可以适当加一个延时操作。
- 针对 Mech-Vision 工程执行时间较长的项目，可适当修改上述超时时限设置。

1020

Mech-Vision：未执行

错误原因：

- 客户端接口程序未首先调用 101 指令启动 Mech-Vision 工程，而是直接调用 102 指令企图获取该工程视觉结果，则报该错误。
  - 例如，如果在 Mech-Center 中注册了两个 Mech-Vision 工程，如果客户端接口程序启动工程 1，然后尝试获取工程 2 的视觉结果，则报该错误。
- 已调用过 102 指令获取视觉目标点中的所有位姿，缓存为空，但仍继续调用 102 指令。

排查步骤：

- 检查客户端接口程序，确保 102 指令指定的工程号正确。
- 检查客户端接口程序。如果 102 指令返回数据中的“是否发送完成”参数值为 1，则表示所有位姿都已传输完毕，继续调用 102 指令会导致该错误。

1021

Mech-Vision：设置物体尺寸失败，请确认步骤 read_object_dimensions 是否存在于工程中

错误原因：

- 客户端接口程序调用 501 指令给 Mech-Vision 工程动态设置物体尺寸，但 Mech-Vision 工程中没有 read_object_dimensions 步骤，则报该错误。

排查步骤：

- 检查 Mech-Vision 工程，确保工程有 read_object_dimensions 步骤。

1022

Mech-Vision：设置物体尺寸数值无效

错误原因：

- 客户端接口程序调用 501 指令给 Mech-Vision 工程动态设置物体大小，但设置的物体尺寸数值无效，存在零或负数。

排查步骤：

- 检查客户端接口程序，设置的物体尺寸长/宽/高都必须为正实数。

1023

Mech-Vision：相机连接失败

错误原因：

- 客户端接口程序触发 Mech-Vision 工程拍照，相机断连。对应的 Mech-Vision 错误码为 CV-E0201。

排查步骤：

- 检查网络连接。
- 检查相机和 IPC 的 IP 是否在同一网段

  （以上两项可以通过在命令提示符中键入“*ping*”加相机的 *IP* 地址来完成。）
- 检查相机电源、IPC 防火墙设置等。
- 如无法排查问题，请联系梅卡曼德技术支持。

1024

Mech-Vision：位姿数量与自定义输出数据数量不匹配

错误原因：

- 客户端接口程序调用 110 指令后，返回的数据中，位姿数和自定义数据元素数量不一致。自定义数据即 Mech-Vision 工程中 procedure_out 步骤的位姿和标签以外的端口输入的数据。可能的情况包括：
  - 自定义端口数据为空
  - 自定义端口数据的长度没有和位姿数据的长度相等。
- 对标准接口使用步骤 procedure_out 时，若位姿数量为 N，要求任何自定义数据端口（位姿和标签以外的数据）需要具有 1 个或 N 个数据项的输入，否则报该错误。
  - 例如，若自定义端口数据为实例分割得出的物体数量，则仅需输入 1 个数据项，每次调用该指令，会收到同一物体数量值；若自定义端口数据为箱子尺寸，则需 N 个数据项，每次调用该指令，会收到与位姿对应的“箱子尺寸”数据。

排查步骤：

- 请检查 Mech-Vision 工程，确保输出端口数据符合上述数据数据的要求。

1025

Mech-Vision：正在使用虚拟相机，数据已弃用

1026

Mech-Vision：无效位姿类型

错误原因：

- 客户端调用 105 指令获取 Mech-Vision “路径规划”步骤的结果时，路径点类型参数值设置错误。路径点类型参数仅可设置为 1 或 2 。

排查步骤：

- 在客户端接口程序中，检查 105 指令设置的路径点类型参数取值是否正确。

1028

Mech-Vision：路径规划失败

错误原因：

- 路径规划工具未能为视觉结果中的任何一个视觉点规划路径，并且并非所有视觉点规划失败的原因都相同。

  （如果因为同样的原因在视觉点上规划失败，会产生相应的错误码）

  - 例如，视觉结果中有 40 个视觉点，其中 20 个因机器人无法到达视觉点而规划失败，而其中 20 个因检测到碰撞而规划失败，则报该错误。

排查步骤：

- 在相应的 Mech-Vision 工程中，打开路径规划工具，查看规划日志，查明规划失败原因。

1030

Mech-Vision：机器人无法到达视觉位姿点

错误原因：

- 传入路径规划工具的视觉点超出机器人的可达范围。

排查步骤：

- 检查相机外参数据是否正常。
- 检查路径规划工具中机器人夹具 TCP 设置是否正常。
- 检查工件是否放置在机器人可达范围之外。

1031

Mech-Vision：机器人关节角计算失败

错误原因：

- 路径规划工具无法为路径点计算机器人 JPs 关节角。

排查步骤：

暂无。

1032

Mech-Vision：轨迹不可达

错误原因：

- 路径规划工具运行出错，路径规划步骤报错误码 MP-E0007。

排查步骤：

暂无。

1033

Mech-Vision：移动奇异点错误

错误原因：

- 机器人在路径规划时遇到奇异点错误，引起规划失败。

排查步骤：

- 检查路径规划工具中的路径点，修改路径点中的位姿或参数，避免机器人奇异点报错。

1034

Mech-Vision：直线运动规划失败

错误原因：

- 路径规划工具检测到无法以直线运动移动到目标点。

排查步骤：

- 适当调整路径规划工具中该移动点的位姿，或添加过渡点。
- 选择使用关节移动的方式进行移动。

1035

Mech-Vision：无效抓取点

错误原因：

- 路径规划工具报错误码 MP-E0010, 原因未知。

排查步骤：

暂无。

1036

Mech-Vision：检测到机器人自体碰撞

错误原因：

- 路径规划工具检测到机器人本体碰撞，路径规划失败。

排查步骤：

- 检查路径规划工具中的路径点。

1037

Mech-Vision：检测到机器人与物体间碰撞

错误原因：

- 路径规划工具检测到机器人与场景物体发生碰撞，路径规划失败。

排查步骤：

- 检查路径规划工具中的场景模型和路径点。

1038

Mech-Vision：点云碰撞点数大于阈值，检测到碰撞。

错误原因：

- 路径规划工具检测到机器人与物体点云发生碰撞，碰撞点数超过阈值。路径规划失败。

排查步骤：

- 检查路径规划工具中的路径点。
- 若有需要，可调节碰撞检测中的碰撞点数阈值。

1039

Mech-Vision：点云碰撞面积大于阈值，检测到碰撞。

错误原因：

- 路径规划工具检测到机器人与物体点云发生碰撞，碰撞面积超过阈值。路径规划失败。

排查步骤：

- 检查路径规划工具中的路径点。
- 若有需要，可调节碰撞检测中的碰撞面积阈值。

1040

Mech-Vision：点云碰撞体积大于阈值，检测到碰撞。

错误原因：

- 路径规划工具检测到机器人与物体点云发生碰撞，碰撞体积超过阈值。路径规划失败。

排查步骤：

- 检查路径规划工具中的路径点。
- 若有需要，可调节碰撞检测中的碰撞体积阈值。

1044

Mech-Vision：视觉移动步骤未收到视觉位姿。

错误原因：

- “路径规划”步骤的输入端口没有收到视觉点。

排查步骤：

- 从“路径规划”步骤的视觉点端口向上检查 Mech-Vision 工程的数据流。

1046

Mech-Vision：无效的末端工具

错误原因：

- 路径规划工具中未对末端工具进行设置。

排查步骤：

- 请在路径规划工具中设置所使用的夹具。如果不使用任何夹具，请创建一个空夹具。

Mech-Viz 错误排查

2001

Mech-Viz：软件未注册

错误原因：

- Mech-Viz 软件未打开。
- 在开发者模式下，并行开启多个 Mech-Viz 软件。

排查步骤：

- 检查 Mech-Viz 软件是否打开。
- 退出 Mech-Viz 的开发者模式，重启 Mech-Viz 软件。

2002

Mech-Viz：工程运行中

错误原因：

- 客户端接口程序在 Mech-Viz 工程还在运行的过程中，重新调用 201 指令企图触发同一 Mech-Viz 工程，引发该错误。

排查步骤：

- 检查客户端接口程序是否存在短时间内重复调用 201 指令触发 Mech-Viz 工程运行的情况。

2003

Mech-Viz：未收到 Mech-Vision 视觉结果

错误原因：

- Mech-Viz 工程中 check_look 步骤走了“无结果”出口（即左起第二个出口）。工程中止。

排查步骤：

- 确认 Mech-Viz 工程 check_look 的“无结果”出口是否需要添加连线走其他分支。
- 检查 ROI 内待识别工件是否已抓完。
- 检查 Mech-Vision 工程 ROI 设置和实例分割阈值等设置是否正确。

2004

Mech-Viz：无法到达 Mech-Vision 传入的视觉点

错误原因：

- Mech-Viz 调用的视觉服务计算出的视觉点超出机器人的可达范围。

排查步骤：

- 检查 Mech-Viz 中工件点云的位置是否正常。
- 检查相机外参数据是否正常
- 检查 Mech-Viz 中机器人夹具 TCP 设置是否正常，工件是否放置在机器人可达范围之外。

2005

Mech-Viz：机器人关节角计算失败

错误原因：

- Mech-Viz 无法为目标点计算机器人 JPs 关节角。

排查步骤：

暂无

2006

Mech-Viz：启动 Mech-Viz 时命令参数无效

错误原因：

- 客户端调用 201 指令触发 Mech-Viz 工程时，机器人位姿类型参数值设置错误。机器人位姿类型参数取值范围为 0~2。

排查步骤：

- 在客户端接口程序中，检查 201 指令中设置的机器人位姿类型参数取值是否正确。

2007

Mech-Viz：运动路径规划失败

错误原因：

- Mech-Viz 未能为视觉结果中的任何一个视觉点规划路径，并且并非所有视觉点规划失败的原因都相同。

  （如果因为同样的原因在视觉点上规划失败，会产生相应的错误码）

  - 例如，视觉结果中有 40 个视觉点，其中 20 个因机器人无法到达视觉点而规划失败，而其中 20 个因检测到碰撞而规划失败，则报该错误。

排查步骤：

- 查看 Mech-Viz 的规划日志，查明规划失败原因。

2008

Mech-Viz：工程运行错误

错误原因：

- Mech-Viz 工程运行出错，报 Mech-Viz 错误码 MP-Exxxx 或其他 Mech-Center 未具体解析的错误。发生该错误时，Mech-Viz 工程没有执行完毕即被中止。

排查步骤：

- 检查 Mech-Viz 界面报错和日志信息。

2009

Mech-Viz：未提供工具位姿（TCP）

错误原因：

- 客户端接口程序触发 205 指令获取 Mech-Viz 规划路径，返回数据要求是机器人的工具位姿（TCP）。但 Mech-Viz 的输出中并没有包含工具位姿数据。

排查步骤：

- 检查 Mech-Viz 软件的“其他”标签页设置，确认已勾选“发送工具位姿”。

2010

Mech-Viz：路径不可达

错误原因：

- Mech-Viz 工程运行出错，Mech-Viz 错误码 MP-E0007。

排查步骤：

暂无

2011

Mech-Viz：未提供 DO 列表

错误原因：

- 客户端接口程序调用 206 指令获取控制工具的 DO 信号列表（如分区吸盘、阵列夹具等），但在 Mech-Viz 工程中，没有配置 DO 信号列表。

排查步骤：

- 检查 Mech-Viz 工程中视觉移动步骤后是否连有 set_do_list 步骤，步骤参数中“接收对象”下勾选“标准接口”。

2012

Mech-Viz：无效的位姿类型

错误原因：

- 客户端接口程序调用 205 指令获取 Mech-Viz 规划的路径时，设置的路径点类型参数值（仅为 1 或 2）非法。

排查步骤：

- 在客户端接口程序中，检查 205 指令设置的路径点类型参数值是否正确。

2013

Mech-Viz：无效的位姿数据

错误原因：

- 客户端接口程序调用 201 指令触发 Mech-Viz 工程运行时，设置的机器人位姿参数值不足六位。Mech-Viz 工程仅支持 6 轴机器人位姿数据，4 轴或 5 轴机器人第 6 轴数据请补零。

排查步骤：

- 检查客户端接口程序，调用 201 指令触发 Mech-Viz 工程时机器人的位姿数据是否是 6 位 JPs 关节角数据。若 Mech-Viz 无需当前机器人位姿信息时，可将位姿类型设置为 0，即无需输入机器人位姿。

2014

Mech-Viz：工程未设置

错误原因：

- 工程未在 Mech-Viz 中打开。
- Mech-Viz 工程没有设置 *自动加载*。

排查步骤：

- 检查 Mech-Viz 软件界面，打开正确的工程，并勾选 *自动加载*。

2015

错误码未使用

2016

Mech-Viz：设置步骤参数失败

错误原因：

- 客户端接口程序调用 208 指令设置 Mech-Viz 步骤参数时，发生错误。

排查步骤：

- 检查配置文件中的步骤编号和参数键名是否正确。
- 配置文件入口（*Property Config*）位于 Mech-Center 的 部署设置 ▸ *Mech-Interface* ▸ 高级设置下。

2017

Mech-Viz：停止执行失败

错误原因：

- 客户端接口程序调用 202 指令停止 Mech-Viz 运行，若 5 秒内 Mech-Viz 未正常停止，则报该错误。

排查步骤：

暂无

2018

Mech-Viz：无效的分支出口号

错误原因：

- 客户端接口程序调用 203 指令设置 Mech-Viz 分支出口，出口号小于或等于零，或出口号超过分支步骤出口数量，则报该错误。

排查步骤：

- 检查 Mech-Viz 工程中分支步骤各出口。
- 检查 203 指令中设置的出口号。

2019

Mech-Viz：设置分支失败，请确认该分支步骤编号是否存在

错误原因：

- 客户端接口程序调用 203 指令设置 Mech-Viz 分支，分支步骤编号必须为正整数，若 Mech-Viz 工程中无指定编号的分支步骤，则报该错误。
- Mech-Viz 中，步骤编号可在对应步骤的参数中读取和设置。范围：1~99。

排查步骤：

- 确保指令发送的步骤编号在 Mech-Viz 中有对应的步骤。
- 确保在指令和 Mech-Viz 工程中设置的步骤编号是正整数。

2020

Mech-Viz：奇异点运动失败

错误原因：

- Mech-Viz 中机器人路径规划遇到奇异点错误，引起规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的目标点，修改目标点中的位姿或参数，避免机器人奇异点报错。

2021

Mech-Viz：直线运动规划失败

错误原因：

- Mech-Viz 检测到无法以直线运动移动到目标点。

排查步骤：

- 适当调整 Mech-Viz 中该移动点的位姿，或添加过渡点。
- 选择使用关节移动的方式进行移动。

2022

Mech-Viz：未执行

错误原因：

- 客户端接口程序调用 203 指令设置 Mech-Viz 分支时，工程未运行。
- 客户端接口程序调用 205 指令获取 Mech-Viz 规划的路径之前，没有触发运行工程。
- 客户端接口程序调用 205 指令时，Mech-Viz 未输出规划结果。
- 客户端接口程序调用 205 指令已获取所有位姿，缓存为空，但仍继续调用 205 指令。

排查步骤：

- 检查客户端接口程序，设置 Mech-Viz 分支时，工程需要先运行起来。
- 检查客户端接口程序，需要先触发 Mech-Viz 运行，然后获取规划路径。

- 检查客户端接口程序，如果 205 指令返回数据中的“是否发送完成”参数值为 1，则表示所有位姿都已传输完毕，继续调用 205 指令会导致该错误。

2023

Mech-Viz：工程文件异常

2024

Mech-Viz：无效的分支名

错误原因：

- 客户端接口程序调用 203 指令设置 Mech-Viz 分支，分支步骤编号必须是正整数。

排查步骤：

- 检查客户端接口程序，设置的 Mech-Viz 分支步骤编号必须是正整数。
- Mech-Viz 中，步骤编号可在对应步骤的参数中读取和设置。范围：1~99。

2025

Mech-Viz：执行超时

错误原因：

- 客户端接口程序调用 205 指令获取 Mech-Viz 规划路径，在规定时间内（默认 10s）工程没有运行结束。执行时间超时报错。

排查步骤：

- 检查 Mech-Viz 工程正常的执行时间。如果耗时超过 10 秒，请调整 Mech-Center 部署设置 ▸ *Mech-Interface* ▸ *Advanced Settings* 下的设置。
- 客户端接口程序可在调用 205 指令之前设置一定的延时时间。

2026

Mech-Viz：无效的索引类步骤名

错误原因：

- 客户端接口程序调用 204 指令设置 Mech-Viz 的具有索引参数的步骤索引值，设置的步骤编号必须是正整数。

  - 204 指令格式: *204, 带索引参数的步骤编号, 索引值。*

排查步骤：

- 检查客户端接口程序，调用 204 指令设置 Mech-Viz 索引时，带索引参数的步骤编号必须是正整数。
- Mech-Viz 中，步骤编号可在对应步骤的参数中读取和设置。范围：1~99。

2027

Mech-Viz：无效的索引值

错误原因：

- 客户端接口程序调用 204 指令设置 Mech-Viz 的索引值，设置的索引值必须是正整数。

排查步骤：

- 检查客户端接口程序，调用 204 指令设置 Mech-Viz 索引时，索引值必须是正整数。

2028

Mech-Viz：设置索引失败，请检查带索引参数的步骤是否存在于工程中

错误原因：

- 客户端接口程序调用 204 指令设置 Mech-Viz 的移动类步骤的索引参数，设置移动类步骤的步骤编号必须是正整数。若 Mech-Viz 工程中没有步骤编号对应的移动步骤，则报该错误。

排查步骤：

- 检查客户端接口程序设置的移动类步骤编号是否正确。
- 检查 Mech-Viz 工程的移动类步骤编号是否正确。
- Mech-Viz 中，步骤编号可在对应步骤的参数中读取和设置。范围：1~99。

2029

Mech-Viz：外部移动设置失败

2030

Mech-Viz：无效的视觉点

错误原因：

- Mech-Viz 工程报错 MP-E0010, 原因未知。

排查步骤：

暂无

2031

Mech-Viz：机器人本体碰撞

错误原因：

- Mech-Viz 工程检测到机器人本体碰撞，路径规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的目标点。

2032

Mech-Viz：场景碰撞

错误原因：

- Mech-Viz 工程检测到机器人与场景物体发生碰撞，路径规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的场景模型和目标点。

2033

Mech-Viz：点云碰撞点数超限

错误原因：

- Mech-Viz 工程检测到机器人与物体点云发生碰撞，碰撞点数超过阈值。路径规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的目标点。
- 若有需要，可调节 Mech-Viz 碰撞检测中的碰撞点数阈值。

2034

Mech-Viz：点云碰撞面积超限

错误原因：

- Mech-Viz 工程检测到机器人与物体点云发生碰撞，碰撞面积超过阈值。路径规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的目标点。
- 若有需要，可调节 Mech-Viz 碰撞检测中的碰撞面积阈值。

2035

Mech-Viz：点云碰撞体积超限

错误原因：

- Mech-Viz 工程检测到机器人与物体点云发生碰撞，碰撞体积超过阈值。路径规划失败。

排查步骤：

- 检查 Mech-Viz 工程中的目标点。
- 若有需要，可调节 Mech-Viz 碰撞检测中的碰撞体积阈值。

2036

Mech-Viz：视觉识别未拍照

错误原因：

- Mech-Viz 工程执行过程中 check_look 步骤走了第四个出口（“未拍照”），该出口没有连接其他步骤，Mech-Viz 运行中止。

排查步骤：

- 检查 Mech-Viz 工程中 visual_look 和 check_look 步骤中设置的视觉服务是否一致。

2037

Mech-Viz：视觉识别无结果

错误原因：

- Mech-Viz 工程执行中 check_look 步骤走了第二个出口（“无结果”），该出口没有连接其他步骤，Mech-Viz 运行中止。对应的 Mech-Vision 工程没有输出视觉结果。

排查步骤：

- 参考*1002* 进行故障排查。

2038

Mech-Viz：视觉识别 ROI 内无点云

错误原因：

- Mech-Viz 工程执行中 check_look 步骤走了第五个出口（“无点云”），该出口没有连接其他步骤，Mech-Viz 运行中止。对应的 Mech-Vision 工程没有输出视觉结果。

排查步骤：

- 参考*1003* 进行故障排查。

2039

Mech-Viz：无可供规划的视觉点

错误原因：

- Mech-Viz 工程无可供 visual_move 规划的视觉点。若 Mech-Viz 中没有使用 check_look 步骤，或 check_look 步骤走了第二个出口（“无结果”），且后连接有视觉移动步骤，则报该错误。

排查步骤：

- 参考*1002* 进行故障排查。
- 检查 Mech-Viz 工程中是否正确使用了 check_look 步骤。

2040

Mech-Viz：复用视觉结果存在规划失败路径

错误原因：

- Mech-Viz 工程 visual_move 步骤中选择复用视觉结果，使用同一次返回的视觉结果进行多个工件抓取，没有对所有视觉点都成功规划出了移动路径，存在部分规划失败的视觉点。
- 该错误发生时，成功的规划路径仍会输出。

排查步骤：

- 查看 Mech-Viz 的规划历史，查明规划失败原因。

2041

Mech-Viz：获取步骤参数失败

错误原因：

- 客户端接口程序调用 207 指令获取 Mech-Viz 步骤参数指令时，发生错误。

排查步骤：

- 检查配置文件内容是否正确。配置文件入口（*Property Config*）位于 Mech-Center 的 部署设置 ▸ *Mech-Interface* ▸ 高级设置下。
  - 尤其是需要检查步骤编号和参数键名是否正确。

2042

Mech-Viz：获取视觉移动规划结果失败

错误原因：

- 客户端接口程序在调用 210 指令获取 Mech-Viz 中 visual_move 步骤的规划结果时，发生错误。

排查步骤：

暂无

2043

Mech-Viz：获取视觉目标点自定义数据失败

错误原因：

- 客户端接口程序调用 210 指令从 Mech-Viz 获取 visual_move 的规划结果和视觉结果中的自定义数据，但在获取自定义数据部分时遇到错误。

排查步骤：

- 请参考*1024*.

2044

Mech-Viz：视觉服务未注册

错误原因：

- Mech-Viz 工程中 visual_look 步骤未正确设置视觉服务。

排查步骤：

- 检查 Mech-Viz 工程的 visual_look 步骤是否设置了正确的视觉服务名称。

2045

Mech-Viz：无效的末端工具

错误原因：

- Mech-Viz 工程中存在“切换工具”步骤，但未对末端工具进行设置。

排查步骤：

- 请在 Mech-Viz 工程中给“切换工具”步骤设置末端工具。如果不使用任何末端工具，请删除该步骤，或创建一个空末端工具。

Mech-Center 错误排查

3001

Mech-Center：非法指令

错误原因：

- 系统不支持客户端接口程序发送的指令代码。

排查步骤：

- 检查客户端接口程序。

3002

Mech-Center：接口指令长度或格式错误

错误原因：

- 客户端接口程序发送的指令信息长度异常，比如机器人位姿数据的位数不足 6 位。
- 客户端接口程序发送的指令信息格式异常，比如未使用逗号（英文半角逗号）分隔符。

排查步骤：

- 检查客户端接口程序发送的指令信息是否符合接口要求。

3003

Mech-Center：客户端断连

3004

Mech-Center：服务端断连

3005

Mech-Center：调用 Mech-Vision 超时

3006

Mech-Center：未知错误

3007

Mech-Center：数据确认信号超时

错误原因：

在使用 PROFINET / EthernetIP 通信时：

- Mech-Center 将新的位姿数据发送到端口前，需要确认客户端接口程序的 Data_Acknowledge 信号为 0，若在规定的超时时间内，客户端接口程序未将 Data_Acknowledge 信号重置，则报该错误。
- Mech-Center 将位姿数据发送到端口后，若在规定的超时时间内，客户端接口程序未发送为 1 的 Data_Acknowledge 信号表示端口数据已读取，则报该错误。

排查步骤：

- 检查客户端接口程序，确保在调用 102 或 205 指令前，Data_Acknowledge 信号为 0。
- 检查客户端接口程序，确保在读取端口位姿数据后，将 Data_Acknowledge 及时置为 1。

3008

Mech-Center：配置 ID 不存在，参数无法被读取/设置

错误原因：

- 客户端接口程序调用 207 指令获取步骤参数或 208 指令设置步骤参数时，在 Mech-Center 的配置文件中未找到对应的配置 ID 和内容。
- 配置文件入口（*Property Config*）位于 Mech-Center 的 部署设置 ▸ *Mech-Interface* ▸ 高级设置下。

排查步骤：

- 检查配置文件中的的配置 ID 是否是正整数。
- 检查配置文件中对应的配置 ID 和内容是否存在。

外参标定错误排查

7001

标定：参数错误

错误原因：

- 客户端接口程序在执行标定流程时，发送给 Mech-Center 的机器人位姿数据位数不够 6 位。JPs 和法兰位姿数据都必须是 6 位。对于 4 轴或 5 轴机器人，剩余数据需要补零。

排查步骤：

- 检查客户端接口程序发送的机器人位姿数据长度是否正确。

7002

标定：Mech-Vision 没有输出标定位姿

错误原因：

- Mech-Vision 标定流程中，Mech-Vision 未将机器人位姿发送给 Mech-Center。

排查步骤：

暂无

7003

标定：机器人到达标定点失败

3.4 Adapter

当标准接口的功能无法满足项目需求的时候，可以通过编写 Adapter 适配程序实现复杂的过程控制。

本节主要包含如下内容：

- *快速了解 Adapter*
- *Adapter 生成器手册*
- *Adapter 编程指南*
- *Adapter 编程样例*

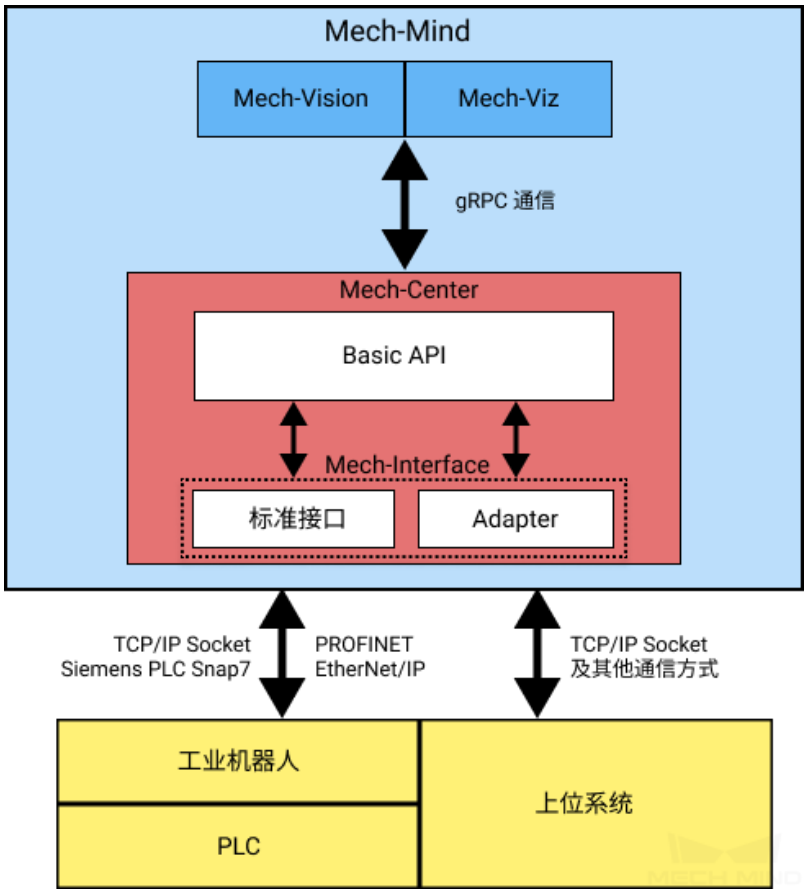
小技巧：如果 Adapter 工程需要使用额外的 Python 库，请安装至 Mech-Center 软件的“python”目录下。Python 库的安装方法如下：

1. 打开“命令提示符”或“PowerShell”程序。
2. 切换到 Mech-Center 软件的“python”目录，例如：C:\Mech-Mind\Mech-Center-1.6.x\python。
3. 执行“**./python -m pip install** *library_name* ”命令。

3.4.1 快速了解 Adapter

Adapter 介绍

Adapter 是 Mech-Center 软件下的通信组件，通过 Basic API 接口与 Mech-Vision 和 Mech-Viz 进行 gRPC 通信，与外部设备之间通过常见的各种工业通信方式进行通信，例如 TCP/IP Socket、HTTP 以及 PLC 数据块传输协议（例如三菱 PLC MC 协议）。



关于 Adapter 的应用场景，请参考“应用说明”章节的内容。

Adapter 功能

Adapter 可以提供如下功能：

- 对内实现对 **Mech-Vision** 和 **Mech-Viz** 软件的控制

类别	功能
Mech-Vision 相关	运行并获取 Mech-Vision 视觉结果
	设置 Mech-Vision 步骤参数
	读取 Mech-Vision 步骤参数
	切换 Mech-Vision 参数配方
Mech-Viz 相关	启动 Mech-Viz
	停止 Mech-Viz
	设置 Mech-Viz 步骤参数
	读取 Mech-Viz 步骤参数
	设置夹具编号
	设置机器人运行速度
	设置点云碰撞参数
	获取 Mech-Viz 运行状态返回
其他	具体功能请参见“*Adapter 编程指南*”

- **对外实现比如用户界面、数据库、文件读写、与 Web 系统通信等非视觉功能**

  对外功能，需要通过 Python 编程开发实现。

Adapter 的开发

针对 TCP/IP Socket 通信方式，Mech-Center 提供 **Adapter 生成器**功能，帮助初次接触 Adapter 的用户快速生成 Adapter 程序并搭建 Adapter 工程。详细信息，请参考“*Adapter 生成器手册*”章节的内容。

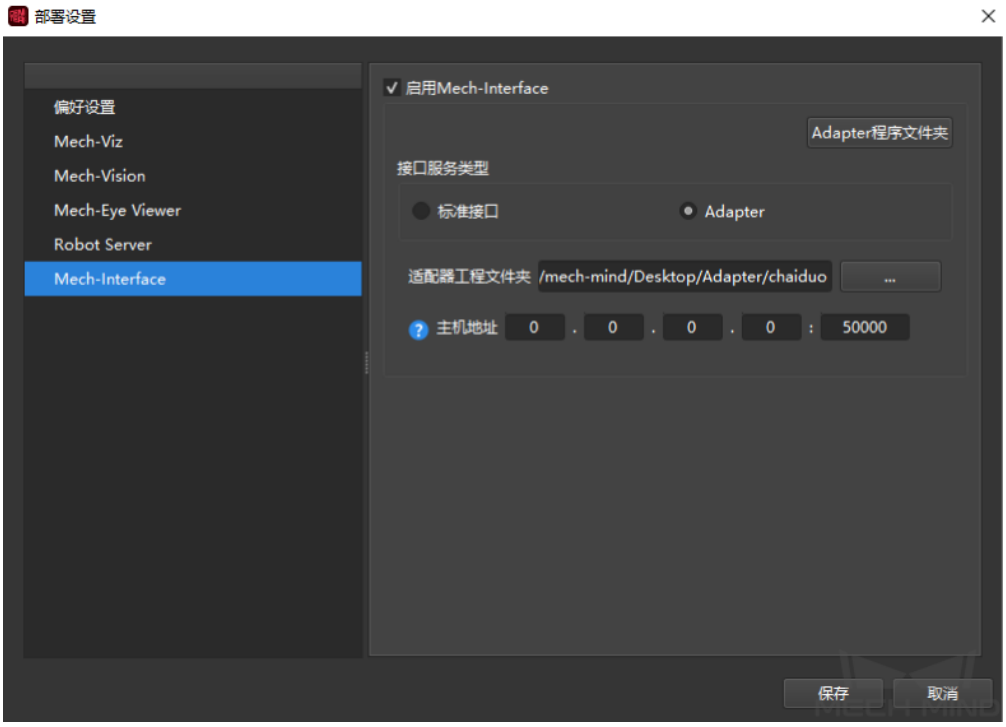
您可以在生成的 Adapter 基础上进行程序的二次开发。

当然，您也可以从零开始编写完整的 Adapter 程序。详细信息，请参考“*Adapter 编程指南*”和“*Adapter 编程样例*”章节的内容。

Adapter 工程的部署

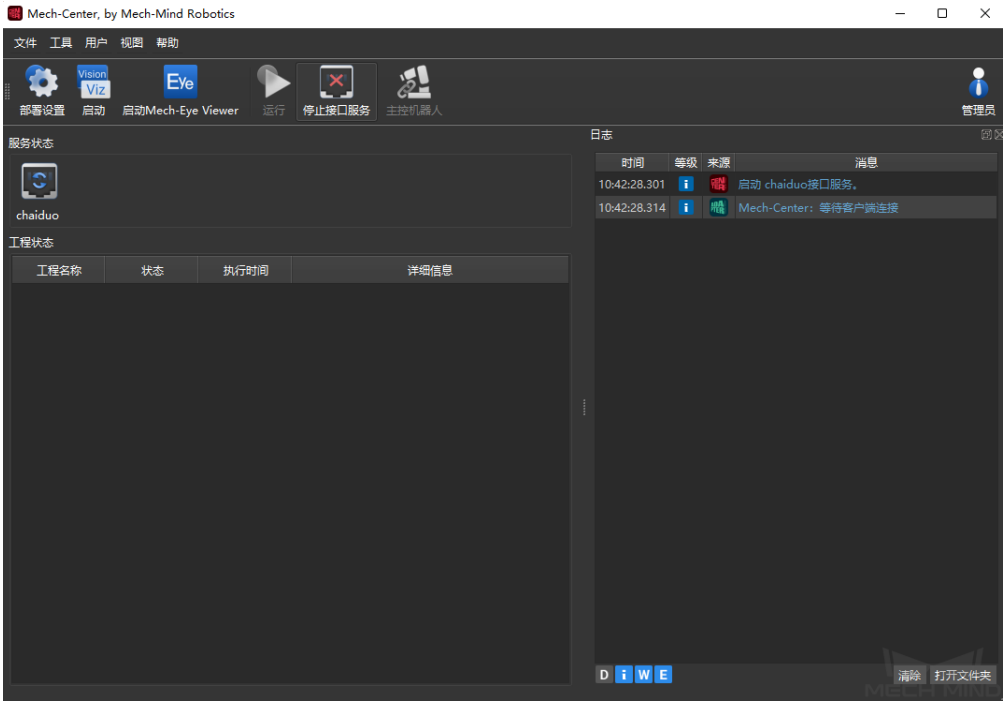
完成 Adapter 程序的编写后，按照如下步骤部署 Adapter 工程：

1. 打开 Mech-Center 软件，点击工具栏中的 部署设置按钮。
2. 在 **部署设置**窗口，选择 **Mech-Interface**，勾选 **启用 Mech-Interface** 复选框，并将 **接口服务类型**设置为 **Adapter**，如下图所示。



3. 将 **适配器工程文件夹**设置为保存 Adapter 程序的目录。
4. 根据现场实际，设置 **主机地址**。端口与对端保持一致。
   - 如果对端作为通信的服务端，**主机地址**应设为对端的 IP 地址。
   - 如果对端作为通信的客户端，**主机地址**应设为“0.0.0.0”。
5. 点击 **保存**按钮，然后重启 Mech-Center 软件。
6. 点击工具栏中的 *启动接口服务*按钮启用 Adapter 服务。

当 *启动接口服务*按钮变为 *停止接口服务*并且服务状态栏中显示启用的 Adapter 程序时，Adapter 服务启用成功，如下图所示。



快速了解 Adapter 后，您可以参照“*Adapter 生成器手册*”章节快速生成第一个 Adapter 程序。

3.4.2 Adapter 生成器手册

本节介绍 Adapter 生成器以及如何使用 Adapter 生成器快速生成 Adapter 程序。

Adapter 生成器介绍

Adapter 生成器是集成在 Mech-Center 中的一个小组件。Adapter 生成器仅适用于采用 TCP/IP Socket 通信方式且只使用 Mech-Vision 提供视觉点的场景。

Adapter 生成器可以帮助您：

- 快速生成 Adapter 程序, 并搭建 Adapter 工程。
- 快速学习 Adapter 的编程开发，以满足复杂的需求。

您可以通过选择 *工具 ▸ Adapter 生成器*，启动 Adapter 生成器以进行相关设置，如下图所示。

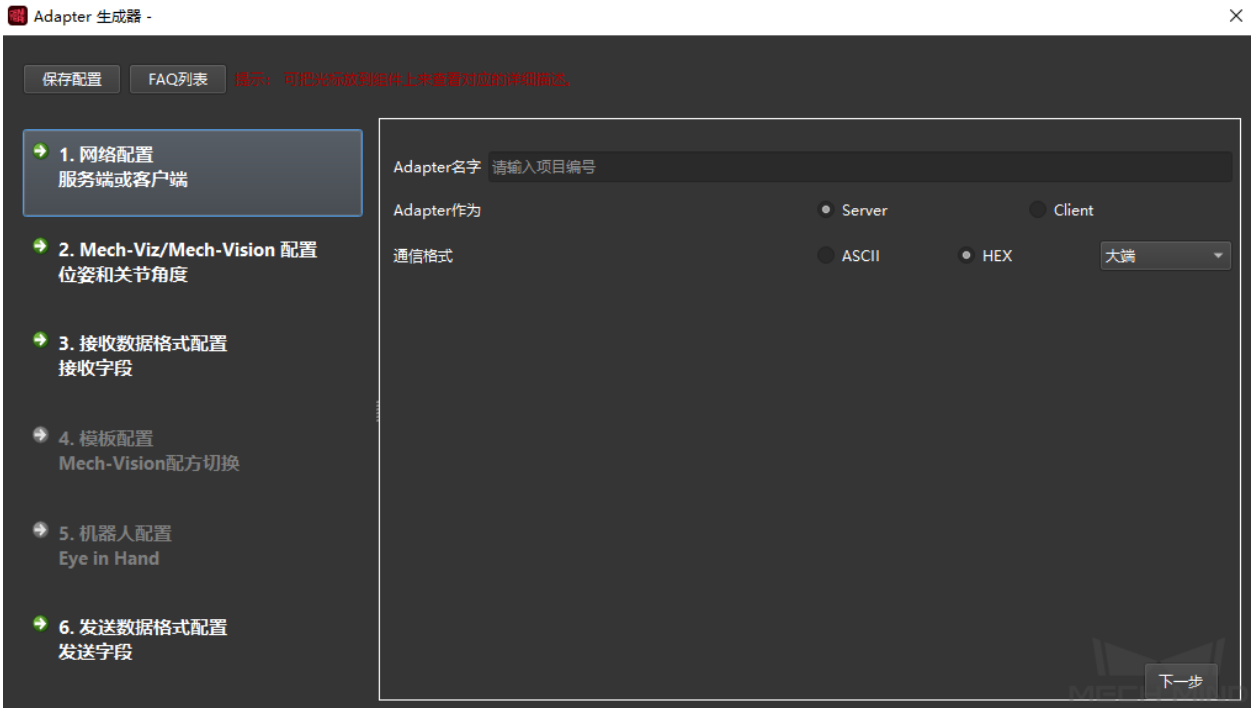
注意： Adapter 生成器仅在管理员模式下可用。

使用 Adapter 生成器快速生成 Adapter 程序

提示： 为方便配置，对应组件处有其详细描述，可把鼠标光标悬浮到组件处查看。

网络配置 - 服务端或客户端

在此步骤，设置参数 **Adapter 名字**、**Adapter 作为**、**通信格式**，然后点击 下一步，如下图所示。



参数说明：

- **Adapter 名字**：指定 Adapter 程序的名称。
- **Adapter 作为**：指定 Adapter 作为 TCP/IP Socket 通信的服务端（Server）或客户端（Client）。如果 Adapter 作为客户端并且服务端对客户端有端口限制，需要勾选 **绑定端口**复选框。

  提示： 为了保证与对端的正常通信，在 Adapter 工程部署时需要指定正确的主机 IP 地址和端口。详细信息，请参见“*Adapter 工程的部署*”章节的内容。

- **通信格式**：指定通信的传输格式，支持 ASCII 字符串或 HEX (十六进制)。当 **通信格式**格式设为“HEX”时，还需指定字节顺序（endianness）为“大端”或“小端”。

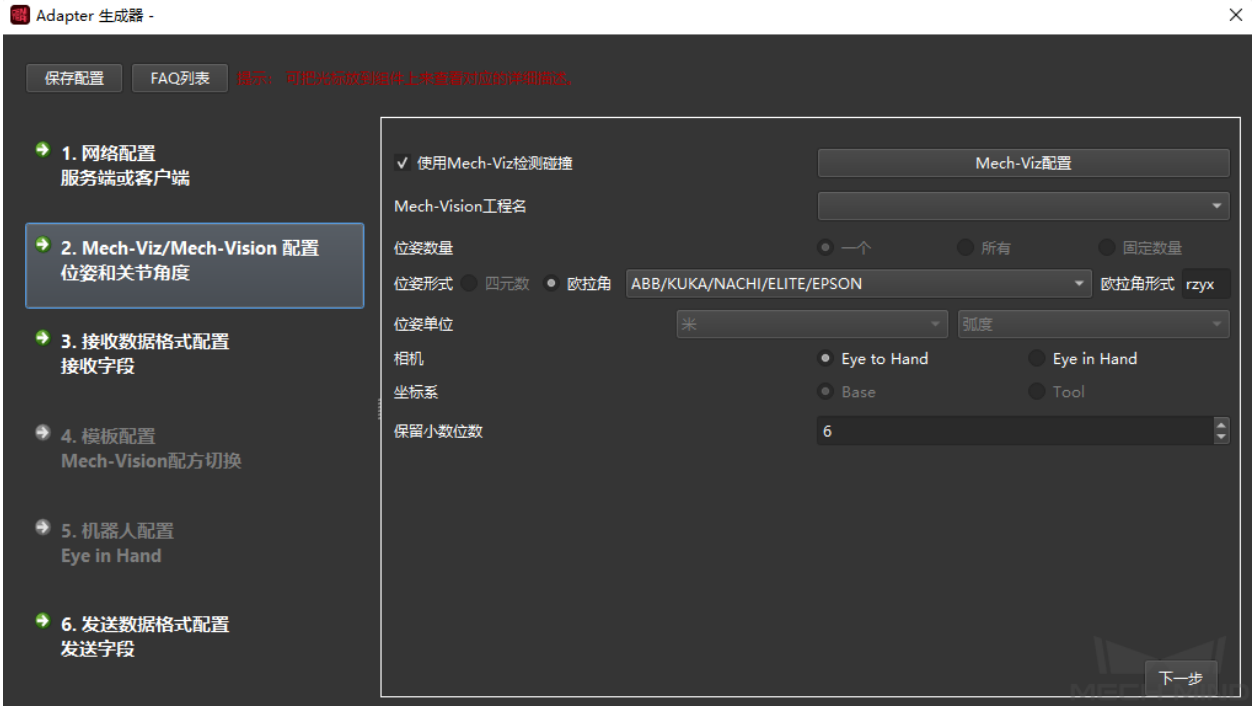
Mech-Viz/Mech-Vision 配置 - 位姿和关节角度

提示：

- 在执行此步骤前，请先准备好 Mech-Vision 工程和用于进行碰撞检测的 Mech-Viz 工程，并启用自动加载选项，确保工程已经在 Mech-Center 中加载。
- Mech-Center 提供了碰撞检测的 Mech-Viz 示例工程“check_collision”，请从 Mech-Center 安装路径下的“tool\viz_project”目录下获取。

注意：如“check_collision”示例工程所示，工程由 visual_look 触发拍照，且必须包含非移动类的任务，任务名称不可更改，包括 notify_1、notify_2 和 visual_look_1。

在此步骤，设置 Mech-Viz 和 Mech-Vision 工程相关的参数，然后点击 下一步。



Mech-Vision 相关参数说明：

- **Mech-Vision 工程名**：指定 Mech-Vision 工程的名称。从下拉列表中选择与 Adapter 交互的的 Mech-Vision 工程。
- **位姿数量**：选择发送给对端的位姿个数。

- **位姿形式**：可以选择四元数或欧拉角。
- **位姿单位**：一般情况下选择使用毫米和度。
- **相机**：选择相机安装方式，分为 ETH 和 EIH 两种情况。
- **位姿坐标系**：确定发送的位姿基于的坐标系。一般情况下，位姿都是基于机器人基坐标系。在 EIH 场景下，当机器人不能给出机器人末端位姿的情况下，位姿只能基于工具坐标系。
- **保留小数位数**：确定发送的位姿的小数位数。最大位数为 10。

Mech-Viz 相关参数说明：

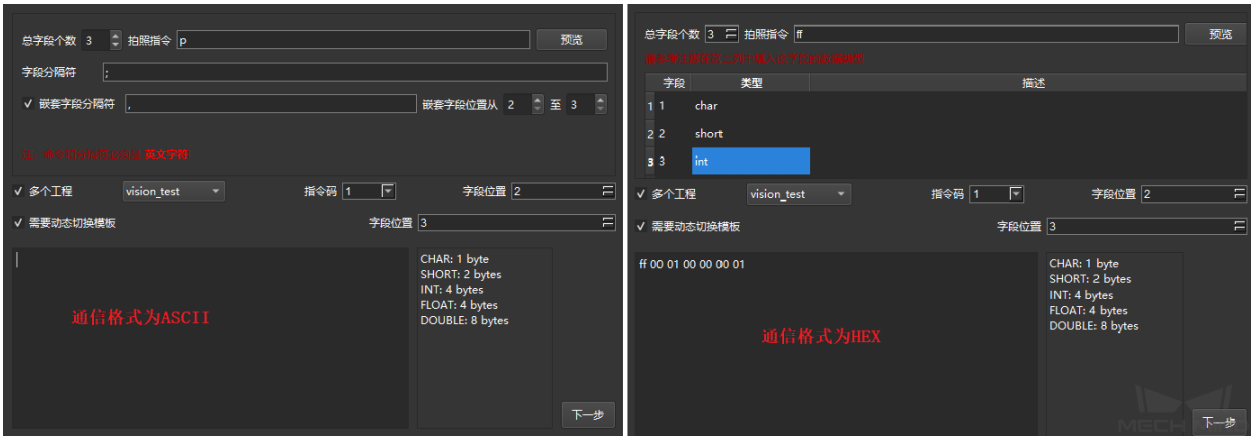
- **使用 Mech-Viz 检测碰撞**：勾选后，视觉点经过 Mech-Viz 工程的碰撞检测，过滤规划失败的点，筛选出过程无碰撞的抓取位姿。
- **Mech-Viz 配置**。勾选 *使用 Mech-Viz 检测碰撞*后，该按钮可用。点击该按钮弹出 **生成器 Mech-Viz 配置** 界面，设置相应参数后，点击 *保存*按钮，如下图所示。*保存*。

  - **需要区分抓取点位和放置点位**：抓取点位是 *视觉移动*之前的所有点位（包含视觉移动），放置点位是 *视觉移动*之后的所有点位。在某些场景下，机器人根据任务可能需要区分抓取动作和放置动作。
  - **需要发送抓取点位和放置点位数量**：若点位数量比较多，可带上数量字段，勾选后默认点位数量大于 1 个才会带上此字段。
  - **需要发送每个点位的运动方式**：Mech-Viz 中移动类任务的运动方式，分为关节运动或直线运动。
  - **更新数值**：默认关节运动对应码是 1，直线运动对应码是 2，可自定义码，更改后点击 *更新数值*按钮生效。
  - **Jps 或位姿**：发送位姿的形式，默认使用 Jps。

- **Jps 单位/位姿单位**：当发送位姿形式为 **Jps** 时，设置 Jps 单位，一般使用度。当发送位姿形式为 **Pose** 时，设置位姿单位，一般使用毫米。
- **机器人名字**：指定机器人服务的名称。使用 Mech-Viz 模拟机器人运动需要一个真实的机器人服务，生成的 Adapter 会模拟这个服务，机器人名字即该服务的名字，需要和 Mech-Viz 中机器人名字一致。单击 *获取机器人名称* 即可自动添加机器人名称。
- **原点关节角度（单位：弧度）**：设置 Mech-Viz 中开始运动的参考原点，单位是弧度，以逗号分隔。可以从 Mech-Viz 中编辑一个 *移动* 作为原点，复制该原点的关节位置。

接收数据格式配置 - 接收字段

在此步骤，设置接收字段的格式，包括拍照指令、多个工程（指令码）以及总字段个数、字段类型、字段中的分隔符和嵌套分隔符，然后点击 *下一步*，如下图所示。



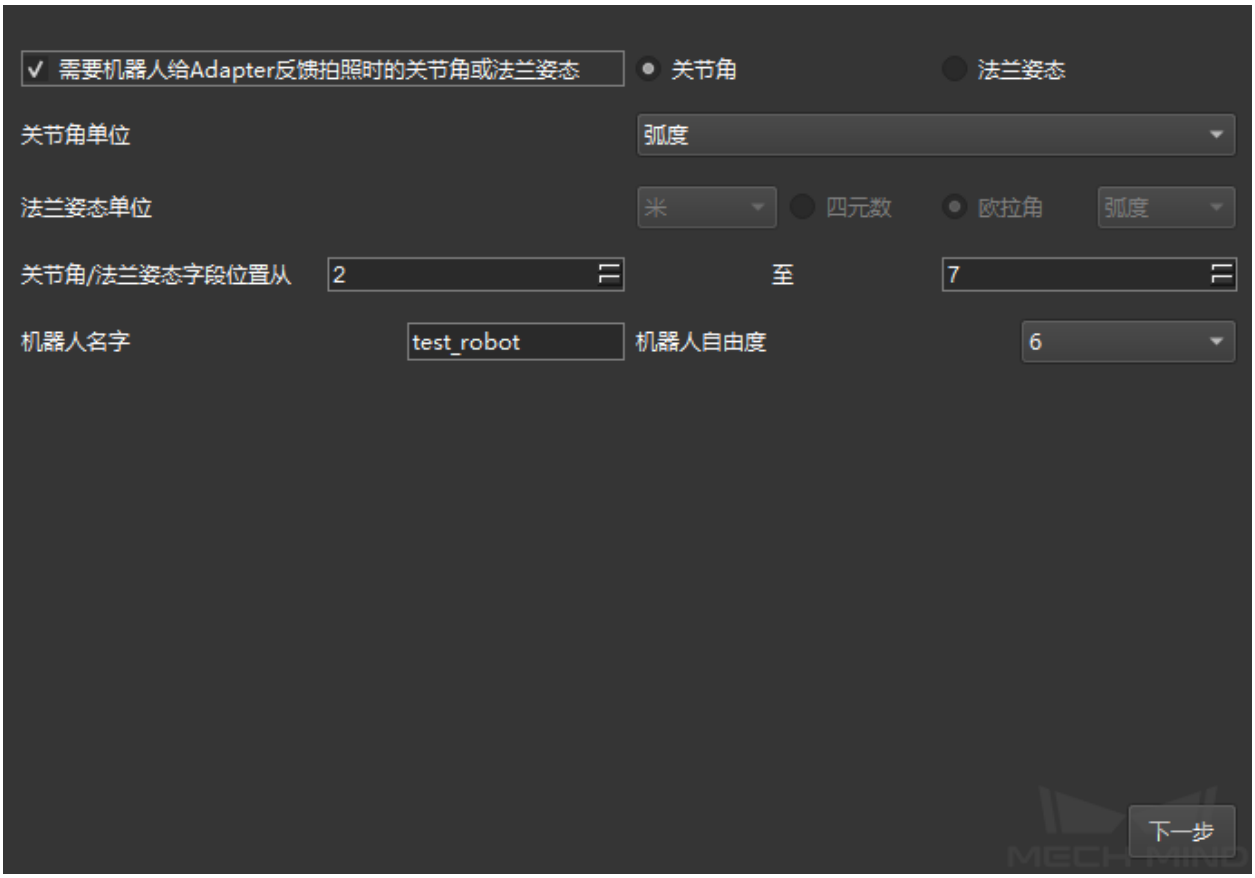
- **总字段个数**：与需要设置的参数个数有关，其取值范围为 1~10。字段中必须存在拍照指令。
- **拍照指令**：外部通信设备向 Mech-Mind 软件系统发送的拍照指令，使相机进行拍照采集。当通信格式为 ASCII 码时，建议使用字母表示，例如字母 *p* 并且字段位置默认为 1。当通信格式为十六进制时（HEX），需要使用十六进制形式的整数，例如 *0xff* 或 *ff* 。
- **字段分隔符**和 **嵌套字段分隔符**：当通信格式为 ASCII 时需要设置。如果字段超过 2 个，需填写字段分隔符；如果额外信息里又需要另外的分隔符，嵌套字段分隔符也是必须的，可指定嵌套字段的起止范围。
- **字段类型**和 **字段描述**：当通信格式为十六进制（HEX）时需要设置。可选的字段类型有 CHAR、SHORT、INT、FLOAT 和 DOUBLE。**字段描述**用于描述字段的功能或含义。
- **多个工程（指令码）**：可选项，当一个项目中存在多个 Mech-Vision 工程，需要根据外部的指令来调用不同的 Mech-Vision 工程，指令码可配置。

> **注意：**每个工程对应的指令码唯一，并且字段位置唯一，不可与其他字段重叠。

机器人配置 - Eye In Hand

提示： 在“*Mech-Viz/Mech-Vision 配置 - 位姿和关节角度*”步骤中，如果 **相机**参数设置为 **Eye In Hand** 时，该步骤可用。

在此步骤，设置机器人拍照时的位姿格式，然后点击 下一步，如下图所示。



参数说明：

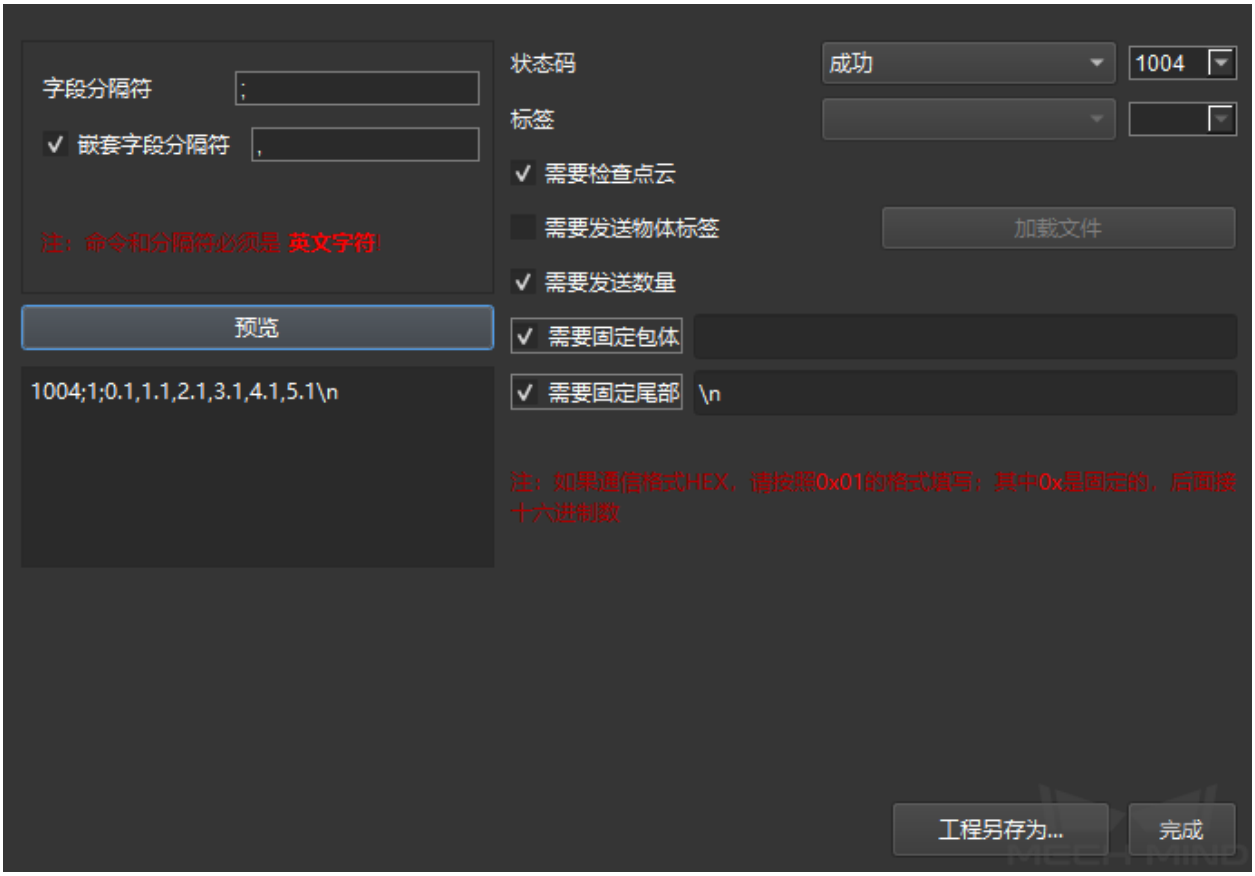
- **需要机器人给 Adapter 反馈拍照时的关节角或法兰位姿：**如果对端需要基于机器人基坐标系下的物体位姿，则需要机器人提供拍照时的关节角或法兰位姿，请勾选该复选框。勾选后，指定位姿的形式，可以选择关节角或法兰位姿。
- **关节角单位：**指定关节角单位，可以选择度或弧度。
- **法兰位姿单位：**指定法兰位姿单位，可以选择四元数或欧拉角。当使用四元数时，单位可以选择米或毫米；当使用欧拉角时，单位可以选择度或弧度。
- **关节角/法兰姿态字段位置：**指定关节角或位姿的起始和结束字段在总字段中的位置。

> **注意：**索引位置从 1 开始计数，不可与其他字段重叠，例如索引位置 1 为拍照指令！

- **机器人名字**：用来标识机器人服务的名字，需要和 Mech-Viz 中机器人名字一致。
- **机器人自由度**：当前支持 4 轴和 6 轴机器人，按工程实际情况选择对应的机器人自由度。

发送数据格式配置 - 发送字段

在此步骤，设置发送位姿的格式，然后点击 下一步，如下图所示。



参数说明：

- **字段分隔符**和 **嵌套字段分隔符**：设置分隔符形式。当通信格式为 ASCII 时需要设置。如果字段超过 2 个，需填写字段分隔符；如果额外信息里又需要另外的分隔符，嵌套字段分隔符也是必须的，可指定嵌套字段的起止范围。
- **状态码**：设置发送状态，每一种状态对应的状态码唯一。
- **需要检查点云**：勾选后，Adapter 会检查点云，若点云不存在则会输出对应的状态码。
- **需要发送物体标签**：发送物体标签是指根据 Mech-Vision 识别的标签发给对端，每个标签是接在位姿后面；当对端不方便解析标签字符串时，也可以指定对应标签的码，需要加载包含所有标签字符串的标签

文件。注意：标签文件格式必须是 json 数组格式。

- **需要发送数量**：发送时包含当次发送的位姿数量。
- **需要固定包体**：当视觉识别失败时，给对端发送消息（错误码后面的消息）。
- **需要固定尾部**：勾选后会在发送的数据后面添加固定尾部标识。

注意： 当通信格式为十六进制（HEX）时，需要设置状态码、位姿个数和位姿字段的数值类型。

完成以上全部配置后，单击 *完成* 或 *工程另存为* 保存 Adapter 工程。保存 Adapter 工程后，您可以参照“*Adapter 工程的部署*”章节部署和使用 Adapter 工程。

关于 Adapter 编程开发的更多信息，请阅读：

- *Adapter 编程指南*
- *Adapter 编程样例*

3.4.3 Adapter 编程指南

本节介绍 Adapter 编程风格规范以及编程语法知识。

目标读者

本节内容主要面向 Adapter 开发者。

Adapter 开发者应具备以下知识：

- Python 基础语法
- JSON 数据格式

建议先阅读 **Adapter 编程风格规范**。

Adapter 编程风格规范

Adapter 采用 Python 语言编写，因此，Adapter 开发者应严格遵守 Python 编程规范 。

此外，为了增强 Adapter 编程的质量和可读性，Adapter 开发者还应遵守本节的约定。

- 命名规范
- 注释规范
- 日志规范

命名规范

1. 类名使用驼峰命名格式，例如：

```
class AdapterWidget
```

2. 类的成员变量和成员函数使用以下划线连接的小写单词，例如：

```
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. 类的成员变量和成员函数若只在类内部使用，可在名字前面加下划线，表示此名字不建议类外部使用，但类外部仍可使用。例如：

```
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. 常量使用以下划线连接的大写单词，例如：

```
ADAPTER_DIR = "D:/adapter_for_communication"
```

注释规范

注释不宜过多，只需要在表达式复杂或所表达意思较为关键时添加。

- 单行注释

  单行注释请以 # 开头。

- 多行注释

  多行注释请包含在“”” 与 “”” 之间。

- 函数、类或包的介绍注释

  注释放在函数的下面、类的下面或包的最上面，例如：

```
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.
    """

"""
```

(下页继续)

(续上页)

```
Service base classes.
"""

import logging
```

日志规范

日志可帮助出现故障时候分析问题。日志信息最好在适当的时候输出，例如调用关键函数以及函数返回了可参考的数据。

Adapter 支持两种日志打印方式：

- print：该方式只将日志输出到控制台上。该方式方便运行时实时观察，但程序故障时消息将会丢失。因此不建议在实际生产环境使用。
- logging：该方式支持格式化日志显示格式，也支持将日志保存到日志文件中（可选功能）。因此建议使用该方式。

接下来熟悉 Adapter 编程中涉及到的 **抽象父类接口**以及 **Util** 包，对 Adapter 最基本父类和常用函数有所掌握。

抽象父类接口

抽象父类接口是指当子类继承父类时可以根据实际需求进行重写的函数。本节介绍以下几个抽象父类：

Communication

通信相关类的源文件位于 Mech-Center 软件安装路径下 `/src/interface/communication.py` 文件内。

Communication 类

Communication 类是负责通信的基础类，提供了一系列接口，服务端或客户端需要重写此类的接口函数。

类函数	说明
is_connected()	判断当前连接是否断开
set_recv_size()	设置接收数据的长度，默认是 1024 字节
send()	接口函数，发送数据
recv()	接口函数，接收数据
close()	接口函数，关闭连接
before_recv()	接口函数，在接收数据之前可根据实际添加逻辑，可重写此函数
after_recv()	接口函数，在接收到数据之后可根据实际添加逻辑，可重写此函数
after_handle()	接口函数，在处理数据之后可根据实际添加逻辑，可重写此函数

TcpServer 类

TcpServer 类封装了一个 TCP/IP Socket 服务端。

类函数	说明
bind_and_listen()	绑定端口
local_socket()	提供本机 Socket 信息
remote_socket()	提供远程 Socket 信息
accept()	接受客户端连接
send()	发送数据
recv()	接收数据
close()	关闭 Socket 连接
close_client()	关闭客户端连接

TcpClinet 类

TcpClient 类封装了一个 TCP/IP Socket 客户端。

类属性	说明
is_bind_port	是否绑定端口，如果服务端限制了所连接的客户端的端口，此变量需要为 True

类函数	说明
send()	发送数据
recv()	接收数据
close()	关闭连接
set_timeout()	设置超时，参数的单位为秒
reconnect_server()	重连服务端
after_connect_server()	接口函数，首次连接服务端成功后的操作
after_reconnect_server()	接口函数，重连服务端成功后的操作
after_timeout()	接口函数，超时之后的操作

Adapter

Adapter 相关类的源文件位于 Mech-Center 软件安装路径下 `/src/interface/adapter.py` 文件内。

Adapter 基类

Adapter 封装了 Mech-Viz、Mech-Vision、Mech-Center、Robserver 相关的调用，包括启动 Mech-Viz、停止 Mech-Viz、设置 Mech-Vision 或 Mech-Viz 步骤参数、启动 Mech-Vision 识别等功能。Adapter 程序只要调用 Mech-Viz 或 Mech-Vision，则必然需继承 Adapter 类。

Adapter 类属性如下表所示。

类属性	说明
viz_project_dir	当前 Mech-Viz 工程路径
vision_project_name	当前 Mech-Vision 工程名称
is_simulate	是否仿真运行 Mech-Viz
is_keep_viz_state	是否保持 Mech-Viz 上次停止时的运行状态
is_save_executor_data	是否保存 Mech-Viz 执行器的数据
is_force_simulate	是否强制仿真运行 Mech-Viz
is_force_real_run	是否强制真实运行 Mech-Viz
code_signal	在 Mech-Center 主界面显示 Adapter 信息的信号（信息带错误码）
msg_signal	在 Mech-Center 主界面显示 Adapter 信息的信号（信息不带错误码）
i_code_signal	在 Mech-Center 主界面显示 Mech-Interface 信息的信号（信息带错误码）
i_msg_signal	在 Mech-Center 主界面显示 Mech-Interface 信息的信号（信息不带错误码）
viz_finished_signal	Mech-Viz 运行结束信号（正常结束或异常结束）
connect_robot_signal	连接/断开机器人信号
start_adapter_signal	启动 Adapter 信号
service_name_changed	Mech-Center 主界面显示 Mech-Viz 和 Mech-Vision 状态的信号
setting_infos	Mech-Center 的配置信息
service_name	注册的服务名称

Adapter 类函数如下表所示。

类函数	说明
on_exec_status_changed()	接收来自 Mech-Viz 和 Mech-Vision 的状态信息
register_self_service()	注册 Adapter 服务
vision_project_dirs(self):	查询 Mech-Vision 工程文件夹路径
vision_project_names()	查询所有的 Mech-Vision 工程名称
vision_project_names_in_center()	查询在 Mech-Center 中已加载的所有 Mech-Vision 工程名称
is_viz_registered()	判断 Mech-Viz 是否已注册
is_viz_in_running()	判断 Mech-Viz 是否在运行
is_vision_started()	判断 Mech-Vision 工程是否已注册
find_services()	查找服务
before_start_viz()	在 Mech-Viz 启动前调用的函数
after_start_viz()	在 Mech-Viz 启动后调用的函数
viz_not_registerd()	启动 Mech-Viz 之后发现 Mech-Viz 未注册，则会调用此函数
viz_is_running()	启动 Mech-Viz 之后发现 Mech-Viz 正在运行，则会调用此函数
viz_run_error()	启动 Mech-Viz 之后，Mech-Viz 运行过程中发生错误，则会调用此函数
viz_run_finished()	当 Mech-Viz 运行结束后调用的函数
viz_plan_failed()	当 Mech-Viz 规划失败后调用的函数
viz_no_targets()	当 Mech-Viz 规划没有移动点位后调用的函数
viz_unreachable_targets()	当 Mech-Viz 规划出现不可达点位后调用的函数
viz_collision_checked()	当 Mech-Viz 规划检测到碰撞后调用的函数
parse_viz_reply()	解析 Mech-Viz 的回复
wait_viz_result()	等待 Mech-Viz 的回复
start_viz()	启动 Mech-Viz
stop_viz()	停止 Mech-Viz
pause_viz()	暂停 Mech-Viz
find_vision_pose()	触发 Mech-Vision 工程拍照
async_call_vision_run()	异步触发 Mech-Vision 工程拍照
async_get_vision_callback()	异步接收 Mech-Vision 的结果
deal_vision_result()	处理 Mech-Vision 的结果
set_step_property()	设置 Mech-Vision 中步骤参数
read_step_property()	读取 Mech-Vision 中步骤参数
select_parameter_group()	选择 Mech-Vision 工程中的配方模板
set_task_property()	设置 Mech-Viz 中的步骤参数
read_task_property()	读取 Mech-Viz 中的步骤参数
get_digital_in()	获取 DI
set_digital_out()	设置 DO
before_start_adapter()	在启动 Adapter 之前会调用此函数
start()	启动 Adapter
close()	关闭 Adapter
handle_command()	处理接收到的外部命令

TcpServerAdapter 类

TcpServerAdapter 类继承自 Adapter，并封装了 TcpServer 的功能，具体如下所示。

```
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)

    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):
```

(下页继续)

(续上页)

```
        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 类

TcpClientAdapter 类继承自 Adapter，封装了 TcpClient 的功能，具体如下所示。

```
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
```

(下页继续)

(续上页)

```
                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()
        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 类

TcpMultiplexingServerAdapter 类继承自 Adapter，主要用于多个客户端的连接，具体如下所示。

```
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
```

(下页继续)

(续上页)

```
    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):
        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:
```

(下页继续)

(续上页)

```
                    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:]
        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
```

(下页继续)

(续上页)

```
    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 类

IOAdapter 类继承自 Adapter，封装了循环获取 DI 的操作，具体如下所示。

```
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):
```

(下页继续)

(续上页)

```
        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 类

AdapterWidget 类是定制 Adapter 用户界面的父类，任何定制用户界面的功能必须继承它，具体如下所示。

```
class AdapterWidget(QWidget):

    def set_adapter(self, adapter):
        self.adapter = adapter
        self.after_set_adapter()

    def after_set_adapter(self):
        """
            Interface.
        """

    def close(self):
        super().close()
        """
            Interface.
        """
```

Service

服务相关类的源文件位于 Mech-Center 软件安装路径下 `/src/interface/services.py` 文件内。

NotifyService 类

NotifyService 类如下所示。

```
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)
```

默认服务名称为 adapter，若项目中需要多个通知服务，可在子类中重写 service_name 来区分不同的服务。类函数说明如下表所示。

类函数	说明
handle_message()	接口函数，子类可重写此函数，在此函数中实现逻辑
notify()	提供了对消息的解析，子类一般不需要重写

VisionResultSelectedAtService 类

VisionResultSelectedAtService 类如下所示。

```
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):
```

(下页继续)

(续上页)

```
        """
            Interface.
        """

    def posesPlanned(self, request, _):
        logging.info("Plan result:{}".format(request))
        self.poses_planned(request)

    def multiPickCombination(self, request, _):
        logging.info("multiPickCombination:{}".format(request))
```

默认服务类型为 vision_watcher，类型不可修改为其他；默认名称为 vision_watcher_adapter ，若项目中需要多个 vision_watcher 服务，可在子类中重写 service_name 以区分不同的服务。类函数说明如下表所示。

类函数	说明
poses_found()	接口函数，子类可重写此函数，在此函数中实现逻辑，参数是 Mech-Vision 的识别结果
posesFound()	解析 Mech-Vision 识别的消息，子类一般不需要重写
poses_planned()	接口函数，参数是 Mech-Viz 规划选择的视觉点
posesPlanned()	提供了对 Mech-Viz 规划消息解析

RobotService 类

RobotService 类如下所示。

```
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
```

(下页继续)

(续上页)

```
    def stop(self, *_):
        pass

    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
```

默认服务类型为 robot，类型不可修改为其他；默认名称为 robot，子类中需要修改为对应的机器人名称。子类中需要设置 jps 或 pose 值，作用是在 Mech-Viz 运行过程中固定一个位姿，此处需注意，此位姿需要在整个轨迹中不会和场景发生碰撞。类函数说明如下表所示。

类函数	说明
getJ()	给 Mech-Viz/Mech-Vision 返回关节角
setJ()	外部设置关节角，注意单位是弧度
getL()	给 Mech-Viz/Mech-Vision 返回工具位姿
getFL()	给 Mech-Viz/Mech-Vision 返回法兰位姿
setL()	外部设置法兰位姿（四元数形式），注意单位是米
moveXs()	Mech-Viz 在规划完路径后，会调用此函数，参数里包含了移动点的属性，注意：若 Mech-Viz 工程中存在检查 DI、分支等会打断预规划的步骤时，Mech-Viz 会调用该函数多次
stop()	停止机器人，一般不用
setTcp()	设置 TCP，一般不用
setDigitalOut()	设置 DO，一般不用
getDigitalIn()	获取 DI，一般不用
switchPauseContinue()	暂停/继续机器人，一般不用

OuterMoveService 类

OuterMoveService 类如下所示。

```
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
```

(下页继续)

(续上页)

```
    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)
                 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}
```

默认服务类型和名称为 outer_move，若项目中需要多个 outer_move 服务，可在子类中重写 service_name 以区分不同的服务。类函数说明如下表所示。

类函数	说明
move_target_type()	移动点类型，0:jps, 1:tcp_pose, 2:obj_pose
velocity()	移动点速度，默认为 0.25
acceleration()	移动点加速度，默认为 0.25
blend_radius()	移动点转弯半径，默认为 0.05m
motion_type()	移动点运动类型，’J’:moveJ, ‘L’:moveL
is_tcp_pose()	移动点是否为 TCP
gather_targets()	接口函数，收集所有移动点，参数是此时机器人的关节角、法兰位姿和 DI 值，子类可根据需要进行判断和修改
add_target()	添加单个移动点，子类中可调用此函数来添加移动点
getMoveTargets()	Mech-Viz 执行到外部移动时，会调用此函数，参数包含了此时机器人的关节角、法兰位姿和 DI 值

注册服务

以上四种类对应的服务只有在注册后才能使用，注册服务函数如下所示。

```
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 包

Adapter util 包位于 Mech-Center 软件安装路径下 `/src/util` 文件夹内，其中包含许多模块，并提供了一些通用函数。在编程过程中，先查看 util 包中是否已实现某个功能。若功能已实现，则直接使用即可；若功能未实现，且比较通用，可以将其抽象成一个小函数，添加至 util 包中。

下面分别对各个模块做简单介绍。

- *database* 模块
- *json_keys* 模块
- *message_box* 模块
- *timestamp* 模块
- *transforms* 模块
- *util_file* 模块
- *timer* 模块
- *pose* 模块

database 模块

database 模块提供对数据库的操作。Mech-Center 在运行时会默认创建一个 mechmind.db 数据库文件，此文件用于存储运行过程中的日志。database 模块提供了执行 SQL 语句，查询一条或所有记录的函数。

json_keys 模块

json_keys 模块保存了 Mech-Center 中所使用到的 json 键/值字符串，其他模块中可直接导入使用。

message_box 模块

message_box 模块提供了弹窗提示的功能，弹窗提示类型包括 information、warning 和 critical。

timestamp 模块

timestamp 模块提供了返回当前时间戳的功能。

transforms 模块

transforms 模块提供了欧拉角转四元数、四元数转欧拉角、位姿相乘、物体位姿转 TCP 、TCP 转物体位姿、计算物体旋转等函数。第三方库 transforms3d 中也有提供欧拉角转四元数和四元数转欧拉角，但在实际使用过程中，某些情况下 transforms3d 转换的值是错误的。在实际计算时，可优先使用 transforms3d 库，若结果不正确，可使用 transforms 模块提供的自定义转换函数。

util_file 模块

util_file 模块提供了文件读写功能，包括常用的读写 json 文件。

timer 模块

timer 模块提供了一个便捷的定时器类。在需要定时功能时，可生成 Timer 对象，传入回调函数，调用 start() 即可。使用完成后，不需要销毁 Timer 对象，程序退出会自动销毁。

pose 模块

pose 模块提供了一个与 Mech-Viz 表示位姿相同的类，包含平移（单位是米）和旋转（四元数形式），可进行取逆和相乘操作，可从 list 转换或转换到 list 中。另外，pose 模块也提供了几个关于 pose 的单位转换函数，包括毫米转米、米转毫米、弧度转度、度转弧度、四元数转欧拉角、欧拉角转四元数。

最后，根据需求实现抽象父类接口，从而实现对内（Mech-Vision 和 Mech-Viz）与对外（外部设备）的通信。

获取接口

获取当前 Mech-Viz 工程所用的步骤

获取当前 Mech-Viz 工程所用步骤的函数如下所示。

```
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
```

调用 get_viz_task_names() 后，返回 json 格式的字符串，表示获取的所有步骤。

获取 Mech-Viz 或 Mech-Vision 工程中的参数

获取 Mech-Viz 或 Mech-Vision 工程中参数的函数如下所示。

```
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
```

在调用时，未指定 msg 参数中的“type”，则表示获取所有参数。若指定，则仅获取相应的参数。例如，调用 get_property_info(msg={“type”：“move”}) 后，返回 json 格式的字符串，表示获取的移动步骤参数。

Mech-Vision 接口

本节介绍使用 Mech-Vision 相关的接口，具体包含以下接口：

获取视觉目标点

通过 adapter.py 中的 find_vision_pose() 函数获取 Mech-Vision 视觉结果，如下所示。

```
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 中的视觉点通常是以物体位姿（obj_pose）且以四元数格式输出（也可以工具位姿输出，需要在 Mech-Vision 工程中做转换）。

示例

Adapter 需要构建函数将其转换为工具位姿（tcp_pose），有时也需要将四元数转换为欧拉角、弧度转换为角度等（是否转换，需与机器人端达成一致），最后打包发送给机器人端。此外，Adapter 也可以对 Mech-Vision 输出的视觉点进行检查，例如：

```
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
        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
```

其中，vision_result 从 find_vision_pose() 中获取，调用语句如下所示。

```
self.check_vision_result(json.loads(self.find_vision_pose().decode()))
```

vision_result 传入函数后，对于涉及到 ROI 的项目，先判断是否为空料框，再获取视觉点，若视觉点正常，将视觉点传入转换函数（pack_pose()）并发送。

设置步骤参数

在为 Mech-Vision 中的步骤动态设置参数时，通常只需调用 adapter.py 中的 set_step_property()。

```
def set_step_property(self, msg, project_name=None, timeout=None):
    return self.call_vision("setStepProperties", msg, project_name, timeout)
```

其中，msg 决定具体步骤名称及其需要配置的参数。

示例

当需要根据不同类型工件动态设置 Mech-Vision 匹配模板时，可以创建如下函数来设置 msg。

```
def _step_matching_model_cell(step_name, model_type):
    msg = {"name": step_name,
             "values":
            {"modelFile": model_type["ply"],
          "pickPointFilePath": model_type["json"]}}
return msg
```

其中，step_name 为需要配置的 Mech-Vision 步骤名称；model_type 为相应类型工件所对应的文件路径，通过“ply”和“json”可获取以.ply 和.json 结尾的模板文件路径，分别填入 Mech-Vision 对应步骤的参数中。

如果 step_name 为“Local Matching”，则调用语句如下所示。

```
msg = _step_matching_model_cell("Local Matching", model_type)
self.set_step_property(msg)
```

效果如下图所示。

读取步骤参数

获取 Mech-Vision 中某个步骤的参数可通过调用 adapter.py 中的 read_step_property()。

```
def read_step_property(self, msg):
     result = self.call_vision("readStepProperties", msg)
     logging.info("Property result: {}".format(result))
     return result
```

其中，msg 决定步骤名称和需要获取的参数值，创建函数改写 msg 即可。

示例

如果要获取相机 IP，示例代码如下所示。

```
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"]
```

若 Mech-Vision 工程只有一个相机，则仅根据类型（type）即可定位步骤。若 Mech-Vision 中有多个相同步骤，而只想获取或设置某个步骤参数，则可根据名称（name）定位步骤。通过调用 read_step_property() 函数并进行 json 格式转换获取相机的所有参数值（json 格式），如下所示：

```
Property result:
{
  "MechEye": {
    "NetCamIp": "127.0.0.1",
    "TimeOut": "10",
    "configGroup": "",
  }
}
```

本示例中，根据具体参数字段（“MechEye”和“NetCamIp”）最终获取相机的 IP 地址（127.0.0.1）。

切换参数配方

在使用 Mech-Vision 时，如果某些工程的流程一致，只是某些参数不同，此时使用配置参数配方可以实现对不同工程切换相应参数的功能，即调用 adapter.py 中的 select_parameter_group()。

```
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
```

其中，project_name 参数为 Mech-Vision 的工程名称，group_index 参数为配方编号。

示例

当工程需要切换配方时，使用以下示例代码调用 select_parameter_group() 函数并做异常处理。

```
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
```

其中，self.vision_project_name 传入的 Mech-Vision 工程名称，model_code-1 是传入的配方编号。

Mech-Viz 接口

本节介绍使用 Mech-Viz 相关的接口，具体包含以下接口：

启动 Mech-Viz

在 adapter.py 文件的 Adapter 类中已定义了启动 Mech-Viz 的函数，因此代码中可直接调用 start_viz()。另外，可根据项目功能需要重写 self.before_start_viz() 和 self.after_start_viz() 以实现自定义 Mech-Viz 启动前后的操作。

函数定义

```
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()
```

(下页继续)

(续上页)

```
    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
```

start_viz() 默认在新线程中等待 Mech-Viz 运行完成，此目的是避免影响启动 Mech-Viz 之外的其他操作。

接下来以动态设置步骤参数为例，演示如何重写 self.before_start_viz()，如下所示。

```
def before_start_viz(self):
     self.set_move_offset(x, y, z)
```

在 Mech-Viz 启动前，根据读取的数据，配置某个移动点 x、y、z 方向的偏置。

停止 Mech-Viz

在 adapter.py 文件的 Adapter 类中已定义了停止 Mech-Viz 的函数，因此代码中可直接调用 stop_viz()。

函数定义

```
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
```

暂停与继续 Mech-Viz

在 adapter.py 文件的 Adapter 类中已定义了暂停与继续 Mech-Viz 的函数，其作用与 Mech-Viz 中的暂停按钮相同，只能在仿真时使用。

函数定义

```
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)
```

设置步骤参数

在 Mech-Viz 中，若需动态设置步骤参数，通常只需要调用 Adapter 类中的 set_task_property()。

函数定义

```
def set_task_property(self, msg, timeout=None):
    return self.call_viz("setTaskProperties", msg, timeout)
```

其中，msg 决定对不同步骤配置不同的参数。

定点移动

在 Mech-Viz 运行过程中，有时候需要微调定点移动步骤中 X、Y、Z 的偏移值，则在控制 Mech-Viz 的主程序中可以编写如下函数。

示例

```
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)
```

其中，name 表示定点移动步骤的名称；UNIT_PER_METER=1000，通常 x_offset、y_offset 与 z_offset 数据的单位为 mm，而 Mech-Viz 中的数据单位为 m，所以使用 UNIT_PER_METER 进行单位转化。

若按照如下方式调用 set_move_offset() 函数，则 Mech-Viz 中相应的 move_1 步骤的 X、Y、Z 偏移量就会发生相应变化。

```
self.set_move_offset("move_1", 100, 200, 300)
```

按序列移动/按阵列移动

按序列移动或按阵列移动通常需在 Mech-Viz 中提前编辑，然后 Adapter 根据逻辑来修改开始索引，用法同定点移动 、码垛。

外部移动

如果若干个外部目标位姿需要发给 Mech-Viz 进行规划控制，则可以通过外部移动步骤来实现。外部移动步骤支持设置 JPs、TCP、物体位姿，用法如下所示。

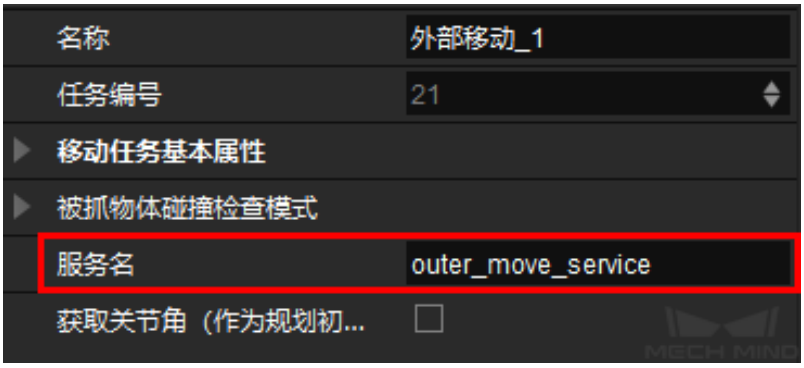
示例

```
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 运行到外部移动步骤时就会调用 getMoveTargets()。不同外部移动步骤之间通过服务名称进行区分。

```
def _register_service(self):
   self.outer_move_service = CustomOuterMoveService()
   self._outer_move_server, port = register_service(outer_move_service, port)
```

在 Mech-Viz 中，外部移动步骤参数如下图所示。



注意： 在外部移动步骤前面需要有一个步骤表示抓取，否则会报错误：“- 未持有物体时物体姿态无效！”。

码垛

在 Mech-Viz 运行过程中，有时候需要根据不同的码垛步骤设置不同的参数，则通过码垛步骤的名称可以定位到需要修改的步骤。选中工程编辑区中的步骤，在 Mech-Viz 参数编辑区出现的参数都可修改。

示例

例如，对于自定义垛型，通常需要改变的参数值是 **开始索引**与 **文件名**（需勾选 **动态加载**，才会出现 **文件夹路径**），因此可以在主程序中定义如下的函数。

```
def set_stack_pallet(self, name, startIndex, fileName):
    msg = {
        "name": name,
        "values": {
            "startIndex": startIndex,
            "fileName": fileName,
        }
    }
    self.set_task_property(msg)
```

其中，“startIndex”对应 **开始索引**，“fileName”对应 **文件名**。“startIndex”与“fileName”参数名称是在 Mech-Viz 中定义的。

使用以下方法调用 set_stack_pallet() 函数。

```
self.set_stack_pallet("common_pallet_1", 2, "re.json")
```

对于预设垛型，通常需要改变的参数值是 **开始索引**、**垛型**、**箱子长度**、**箱子宽度**、**箱子高度**、**垛型行数**、**垛型列数**和 **垛型层数**，因此可以在主程序中定义如下的函数。

```
def set_stack_pallet(self, name, startIndex, 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": {
            "startIndex": startIndex,
            "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)
```

由于需要设置的参数比较多，因此通常会将参数写入一个 excel 文件，然后读取 excel 文件中的数据并记录到 self.box_data_info 中，后续可以通过 stack_type 的值进行索引。“startIndex”、“palletType”、“cartonLength”等名称在 Mech-Viz 中是固定的。

比较 **自定义垛型**和 **预设垛型**类中的 msg 值，不难发现，不同之处就在“values”值。若需要设置某个步骤参数，只需在“values”中添加相应的参数名与值便可进行设置。对于其他垛型步骤的参数设置，参考上述讲解的示例。

消息分支

Mech-Viz 在执行到消息分支步骤时，会一直等待外部信号（指 Adapter）设置出口。对于消息分支步骤，定义如下函数可以进行分支控制。

示例

```
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)
```

其中，name 表示步骤的名称，area 表示出口号，出口从左到右编号为 0、1、2、…。如果此次步骤走最左侧出口，则 area=0。如果没有启动 Mech-Viz 就直接调用分支步骤，Mech-Viz 会返回“没有执行器”的错误。

计数器

在使用计数器步骤时，通常需要设置计数器步骤的总计数次数与当前计数次数，定义设置该步骤的代码如下所示。

示例

```
def set_counter_property(self, name, count, curCount):
    msg = {"name": name,
           "values": {"count": count, "currentCount": curCount}}
    self.set_task_property(msg)
```

调用该函数的示例如下所示。

```
self.set_counter_property("counter_1", 5, self.success_stack_num)
```

其中，self.success_stack_num 表示成功码垛的数量。如果在一次码垛过程中，箱子发生掉落，然后在人为干预下，Mech-Viz 停止。此时 Mech-Viz 中名为“counter_1”的计数器步骤不会保存“currentCount”的值。重启 Mech-Viz 后，通过 self.success_stack_num 可以重新设置计数器步骤的当前计数值。

读取步骤参数

在 Mech-Viz 运行过程中，如果需要读取某个步骤的参数，可以在主程序中定义如下函数。

示例

```
def read_move_pallet(self, name):
     msg = {"name": name,
            "properties": ["xOffset","yOffset","zOffset", ]}
     return read_task_property(msg)
```

其中，通过“name”定位需要读取的步骤名称；“properties”后面的列表中的值可以根据需要添加或者删除参数，示例中表示读取“xOffset”、“yOffset”和“zOffset”的值，所以将该三个值添加到“properties”，调用方法如下所示。

```
self.read_move_pallet("move_3")
```

调用后得到如下结果。

```
{'zOffset': -0.23, 'xOffset': -0.12, 'yOffset': -0.15}
```

另外，需注意的是，从 Mech-Viz 步骤中获取的一些属性值，是规划进度下的数值，而非实际执行进度下的数值。规划进度往往超前于执行进度（即 Mech-Viz 会尽量提前规划未来的移动）。此处以码垛类步骤的当前索引（curIndex）属性值为例进行说明，用户可在主程序中定义如下函数。

```
def read_pallet_current_index(self, name):
     msg = {"name": name,
            "properties": ["curIndex"]}
     return read_task_property(msg)
```

通过如下调用方法获取 curIndex 属性值。

```
self.read_pallet_current_index("common_pallet_1")
```

调用上述方法后可能得到如下结果。

```
{'curIndex': 5}
```

此处的 5 表示当前规划已经完成了 5 轮。例如，对于码放箱子工程而言，5 表示在规划进度下已码放了 5 个箱子，而机器人实际码放的箱子很可能少于 5 个。因此，诸如“curIndex”这类属性值表示已规划的值，而非执行时的值。

设置 TCP

设置 TCP 只需指定 Mech-Viz 中末端工具列表中对应的索引即可。Mech-Viz 中末端工具列表的索引从上往下依次是 0、1、2、3、……，注意不可越界。设置 TCP 函数如下所示。

示例

```
def set_tcp(self, index):
    msg = {"function": "setTcp", "index": index}
    self.call("executor", msg)
```

设置运行时全局速度

在 Mech-Viz 运行过程中，如果需要动态调整机器人运行速度百分比，则可以在主程序中编写如下函数。

示例

```
def set_vel(self, vel_scale):
    msg = {"function": "setConfig",
           "velScale": vel_scale / 100, "accScale": vel_scale / 100}
    self.call("executor", msg)
```

如果将速度设置为 80%，则可以如以下调用该函数。

```
self.set_vel(80)
```

注意： 该函数必须在 Mech-Viz 启动后才能进行调用，否则会报错，即该模块的调用条件与消息分支步骤一致。因此，通常会在消息分支步骤中调用 set_vel() 函数，避免在一个新线程中调用 set_vel() 函数，示例如下所示。

```
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,
               "values": {"info": json.dumps(info)}}
        self.set_task("executor", msg)
    except Exception as e:
        logging.exception(e)
```

设置点云碰撞参数

设置点云碰撞参数，Mech-Viz 对应的接口是 setConfig()，和设置运行时全局速度接口一样，只是设置信息不同，示例如下所示。

示例

```
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 返回值

Mech-Viz 的返回值如下表所示。

返回值	含义
Finished	正常执行完毕，注意：当 Mech-Viz 工程中视觉移动的右分支连接步骤时，Mech-Viz 也会正常返回
Command Stop	按下停止或调用 stop_viz() 函数
No targets	没有视觉点，指 Mech-Vision 返回空
No proper vision poses	视觉点不可达，指机器人自身不可达当前识别物体或与其发生碰撞
PlanFail	Mech-Viz 规划路径失败
SceneCollision	发生碰撞

针对 Mech-Viz 的返回情况，Adapter 基类提供了对应的接口函数。

RobotService 接口

本节介绍使用 RobotService 相关的接口，具体包含以下接口：

模拟 RobotServer 服务

模拟 RobotServer 服务通常用于主控机器人，但无法通过 RobotServer 控制机器人，而是创建 RobotService 以模拟 RobotServer 的作用。当 RobotService 注册在 Mech-Center 后，Mech-Viz 会以与真实机器人连接的同等方式与 RobotService 进行信息交互。

模拟 RobotServer 服务应用于以下场景：

- 获取机器人关节角用于 Mech-Vision 计算（Eye In Hand）；
- 从真实机器人获取位姿传输给 RobotService 服务。

示例

在 Adapter 子类中调用方式如下所示。

```
def _register_service(self):
        """
        register_service
        :return:
        """
        if self.robot_service:
```

(下页继续)

(续上页)

```
            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 类如下所示。

```
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
```

getJ

getJ() 用于 Mech-Viz 和 Mech-Vision 获取当前关节角，具体如下所示。通常先通过 setJ() 设置当前关节角，然后再调用 getJ()。

示例

```
def getJ(self, *_):
     pose = {"joint_positions": self._jps}
     return pose
```

1. Eye In Hand：将机器人发送的关节角写入 setJ()。

   ```
   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))
   ```

   其中，jps 是机器人发送过来的关节角数据，将其赋值给 self._jps，以被 getJ() 调用。由于 getJ() 需要获取弧度格式的数据，此处需注意单位转化。

2. Eye To Hand：无需根据机器人当前关节角设置，但模拟真实机器人状态，需要设置一个安全的目标点用作初始点（通常为 Home 点），否则会以随机数赋值，易出错。

   ```
   def getJ(self, *_):
        return {"joint_positions": [1.246689,-0.525868,-0.789761,-1.330814, 0.922581,
   ↪ 4.364021]}
   ```

getFL

getFL() 用于提供机器人拍照时的法兰位姿，由于在 Eye In Hand 方式下标定的是法兰和相机的位置关系，但最终使用的数据是基坐标系下的数据，因此需要提供拍照时的法兰位姿。

```
def getFL(self, *_):
    return {"flange_pose": self.pose}
```

使用 Eye In Hand 方式提供视觉点时，需注意以下几点：

1. 若机器人返回拍照时的 JPs，则直接在 RobotService 中调用 setJ() 即可（单位是弧度），此函数将返回 []。
2. 若机器人返回法兰位姿，则需进行以下操作：
   - 确保 Mech-Vision 工程中外参文件 extri_param.json 中“is_eye_in_hand”为 true；
   - 调用 RobotService 的 setFL()（单位是米，位姿类型是四元数）。

moveXs

Adapter 创建的 RobotService 服务使用 moveXs() 接收 Mech-Viz 规划后的路径点数据，具体如下所示。

函数定义

```
def moveXs(self, params, _):
    with self.lock:
        for move in params["moves"]:
            self.targets.append(move)
    return {"finished": True}
```

其中，params 传入了 Mech-Viz 工程中全部参数，通过 params[“moves”] 可获取全部运动点的位姿，包括视觉移动、相对移动等，位姿默认以关节角方式返回，将位姿传入 self.targets 便于后续调用。

示例

通常，此功能需要搭配通知共同使用，当 Adapter 获取到通知的消息时，将 self.targets 传入函数转化并打包后发送给机器人，示例如下所示。

```
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)
```

当收到通知消息为“started”时，将目标点列表清零（防止 Mech-Viz 错误中断导致前后两次移动点叠加）；当收到通知消息为“finished”时，将目标点列表传入 pack_move 函数，用于数据整合发送。

```
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)
```

根据现场需要，可发送 Mech-Viz 中全部运动点，也可按照 index 选取其中几点发送；pack_move() 一般根据各品牌机器人所需格式对数据进行整合（通常会在通信协议中提前设定）。

其他接口

本节介绍使用 Adapter 的其他接口，具体包含以下接口：

通知服务

在 Mech-Viz 工程运行到某个分支或者某一步时，若希望调用 Adapter 程序相应的函数，则可以在 Mech-Viz 中添加一个通知步骤。

示例

例如，在 Adapter 中编写了一个使已拆数量加 1 的函数，那么就可以在拆垛过程的最后一个步骤的后面添加一个通知步骤，当运动到此处时，就可以触发 Adapter 并调用相应函数。实现该功能的示例如下所示。

1. 编写一个继承 NotifyService 的类。

```
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()
```

   此通知可以实现如下功能：当 Mech-Viz 运行到发送“Success”的通知步骤时，可以使 Adapter 调用 update_success_num() 函数，当到“Fail”时，Adapter 调用 update_fail_num() 函数。

2. 在控制 Mech-Viz 主程序的类中对 NotifyService 类进行实例化并注册该服务。

```
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)
```

(下页继续)

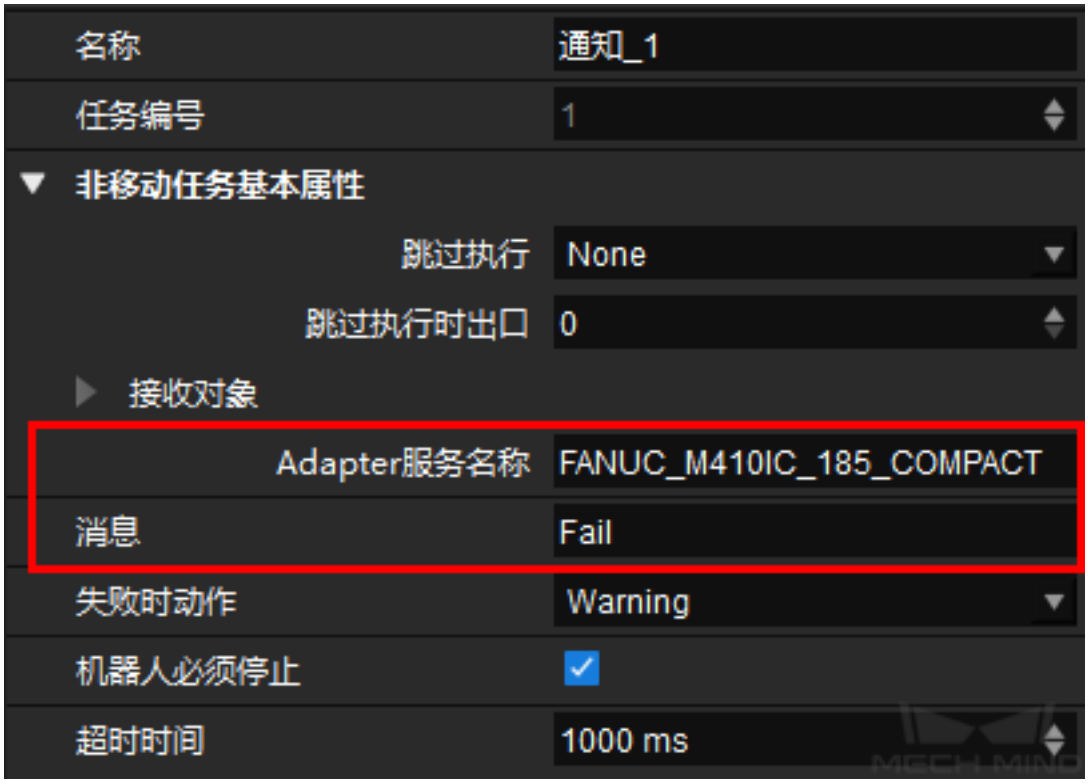
(续上页)

```
    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. 在 Mech-Viz 中需要的位置添加相应的通知步骤。

   在通知步骤中，最主要的是填写 **Adapter 服务名**和 **消息**。这两处的值需要和上面 NotifyService 类中的 service_name 和 msg 值一致。



一旦运行了程序，则在 Mech-Center 界面中出现 service_type 和 service_name，表明注册通知服务成功。

VisionWatcher 服务

当 Mech-Vision 运行结束后，会输出一些结果，例如，vision result:{‘noCloudInRoi’: False, ‘function’: ‘posesFound’, ‘vision_name’: ‘TJTvision-3’}。对于一些异常情况，Adapter 可以通过 VisionWatcher 服务发送错误信息进行提醒。

示例

1. 编写继承 VisionResultSelectedAtService 的类。

```
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()
```

子类 VisionWatcher 需要重写父类的 poses_found() 函数，所以调用 Adapter 中 send_err_no_cloud()（发送错误信息的函数）的逻辑会在 poses_found() 中发生变化。在运行过程中，Mech-Vision 返回视觉点的值会传递给 poses_found() 中的 result 参数。

2. 在控制 Mech-Viz 主程序的类中实例化 VisionWatcher。

```
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())
```

在进行 VisionWatcher 类实列化的过程中，将 send_err_no_cloud() 函数作为参数传递给 VisionWatcher()。当无点云出现时，按照 poses_found() 中的逻辑，则会调用发送错误信息的函数。

一旦程序运行以后，在 Mech-Center 界面中出现已注册的服务，则表明 Adapter 注册 VisionWatcher 服务成功。

3.4.4 Adapter 编程样例

在熟悉 Adapter 编程语法和编程逻辑后，您可以参照本节提供的样例程序编写自己的 Adapter 程序。

仅使用 Mech-Vision 发送视觉点

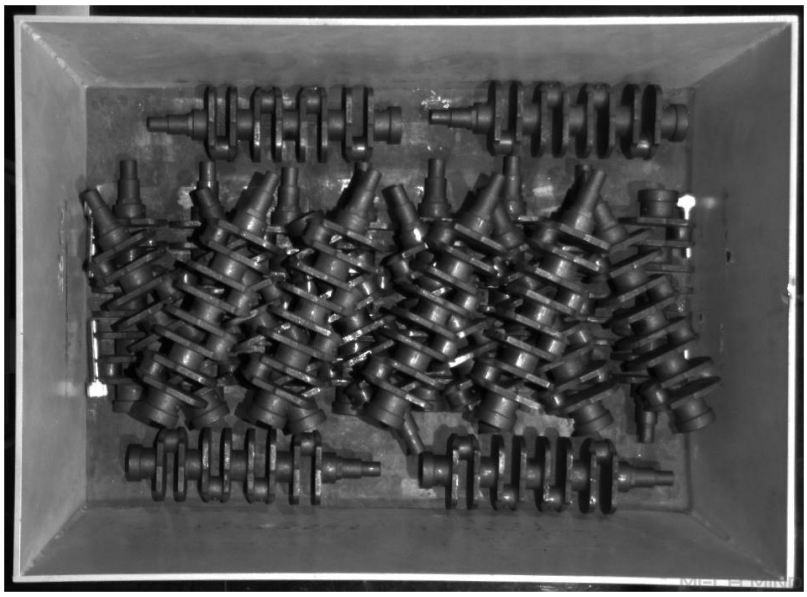
本节详细介绍一个仅使用 Mech-Vision 发送视觉点的 Adapter 样例程序。本节包含以下内容：

背景介绍

本样例针对曲轴上料的应用场景，相机固定安装在料框上方支架上，Mech-Vision 进行拍照并输出一个可抓工件的坐标给机器人端。

本样例使用 Mech-Vision 内置的示例工程“大尺寸非平面工件”（文件 ▸ 浏览示例工程 ▸ 上下料 ▸ 大尺寸非平面工件）。

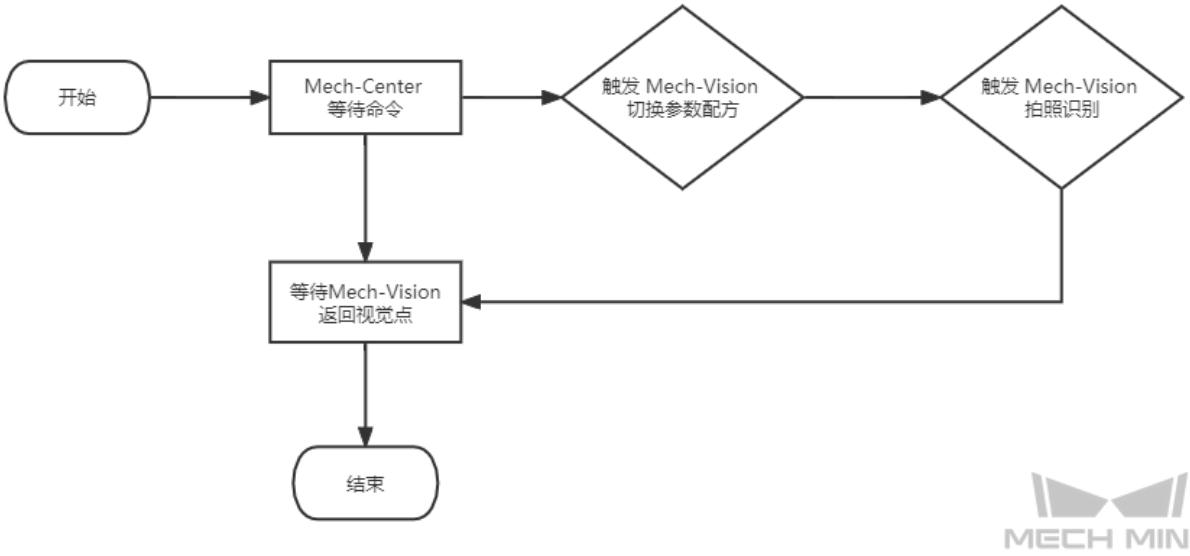
该工程采用 3D 模板匹配的算法，对不同的工件需要设置不同的模板文件和抓取点。因此，需要在 Mech-Vision 中设置参数配方，配方编号（工件编号）在机器人端发送拍照指令时进行设置。关于如何配置参数配方以及查看配方编号，请参考 参数配方章节。



通信方案

机器人与安装 Mech-Mind 视觉系列软件的工控机采用 TCP/IP Socket 协议进行通信，通信格式为 ASCII 字符串，使用英文逗号（,）作为数据分隔符。其中，视觉系列软件作为通信的服务端，机器人作为客户端。

通信流程如下图所示。



通信流程详细描述如下：

1. Mech-Center 等待机器人发送拍照指令 `P` 和配方编号。
2. Mech-Center 触发 Mech-Vision 切换参数配方。
3. Mech-Center 触发 Mech-Vision 拍照识别。
4. Mech-Vision 拍照识别成功后，返回状态码和视觉点给 Mech-Center。
5. Mech-Center 将状态码和视觉点返回给机器人。

注解： 为了方便机器人抓取，Mech-Center 将待抓工件的坐标转换为机器人 TCP 坐标。

通信报文格式

具体通信报文格式如下表所示。

	请求命令	工件编号
发送（机器人 -> 工控机）	P	整数，取值范围: 1~100
	状态码	**视觉点（TCP 坐标）**
接收（工控机 -> 机器人）	整数，取值范围：0~4，0-正常识别；1-错误的命令码；2-Vision 工程未注册；3-无视觉点；4-无点云	6 个浮点型数据,以逗号(,)隔开，格式为：x,y,z,a,b,c

注解： 响应报文的长度固定。如果响应报文的状态码为异常码（1~4），视觉点数据用 0 补齐。

通信报文样例

请求报文

```
P,1
```

正常响应报文

```
0,1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517
```

注解： 本样例中，Mech-Vision 识别正常，返回的 TCP 坐标为：1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517。

异常响应报文：错误的命令码

```
1,0,0,0,0,0,0
```

异常响应报文：Vision 工程未注册

```
2,0,0,0,0,0,0
```

异常响应报文：无视觉点

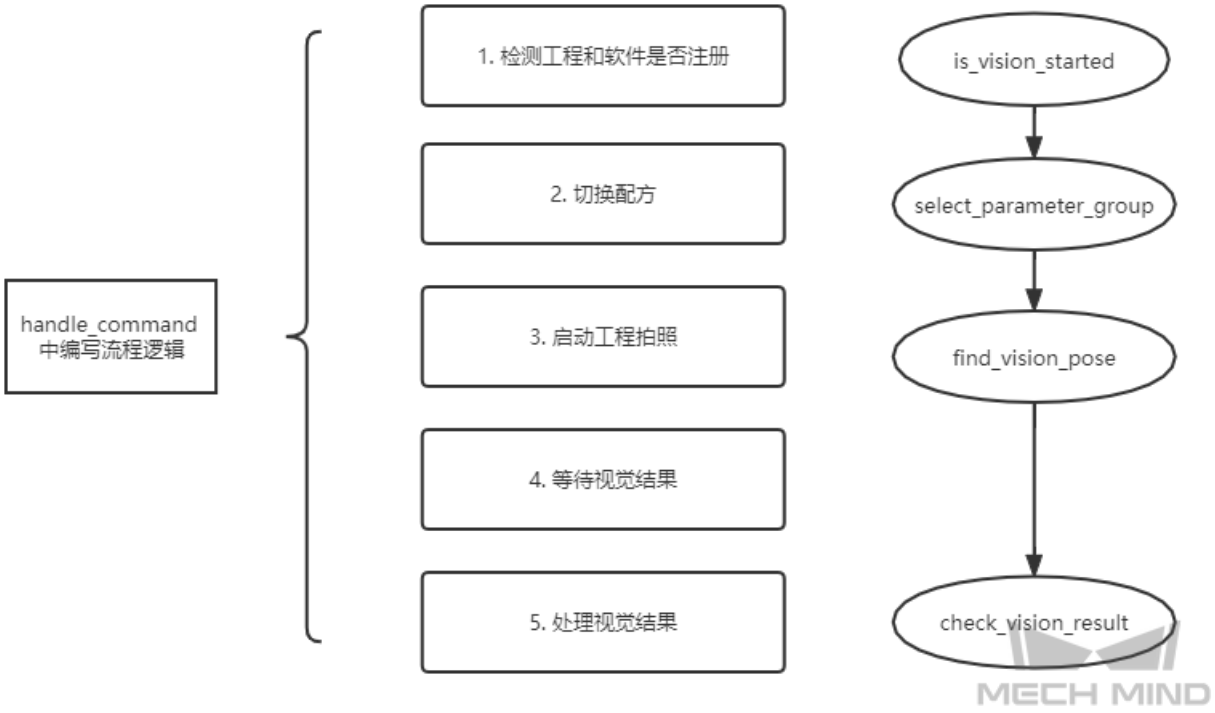
```
3,0,0,0,0,0,0
```

异常响应报文：无点云

```
4,0,0,0,0,0,0
```

编程思路

为了实现项目目标，本样例按照下图所示的思路编写 Adapter 程序。



上图仅列出了样例程序的报文处理逻辑。下节将详细解释样例程序。

样例程序详解

注解： 点击下载 Adpater 样例程序 。

引入 Python 包

导入 Adapter 程序依赖的所有模块。

```
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
```

定义类

定义继承“TcpServerAdapter”父类的“TestAdapter”子类。

```
class TestAdapter(TcpServerAdapter):
    vision_project_name = "Large_Non_Planar_Workpieces"
    # vision_project_name = 'Vis-2StationR7-WorkobjectRecognition-L1'
    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)
```

注解： 本样例将 Adapter 程序定义为 TCP/IP Socket 通信的服务端。

设置接收指令与处理逻辑

设置接收请求（包括拍照指令和参数配方）的处理逻辑。

```
# 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))
```

(下页继续)

(续上页)

```
        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:
        self.msg_signal.emit(logging.ERROR, 'Calling project timed out.␣
↪Please check whether the project is correct: {}'.format(e))
        self.send(('2' + '' + '').encode())
```

注解: “handle_command”函数作为 TCP/IP Socket 服务端接收报文的处理入口。

定义 Mech-Vision 视觉结果检查

设置 Adapter 检查 Mech-Vision 输出的视觉结果。

```
# 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')
```

设置视觉点输出格式

设置视觉点的对外输出格式。

```
# 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)
```

(下页继续)

(续上页)

```
        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 + ''
```

定义 Adapter 的关闭操作

定义如何关闭 Adapter。

```
def close(self):
    super().close()
```