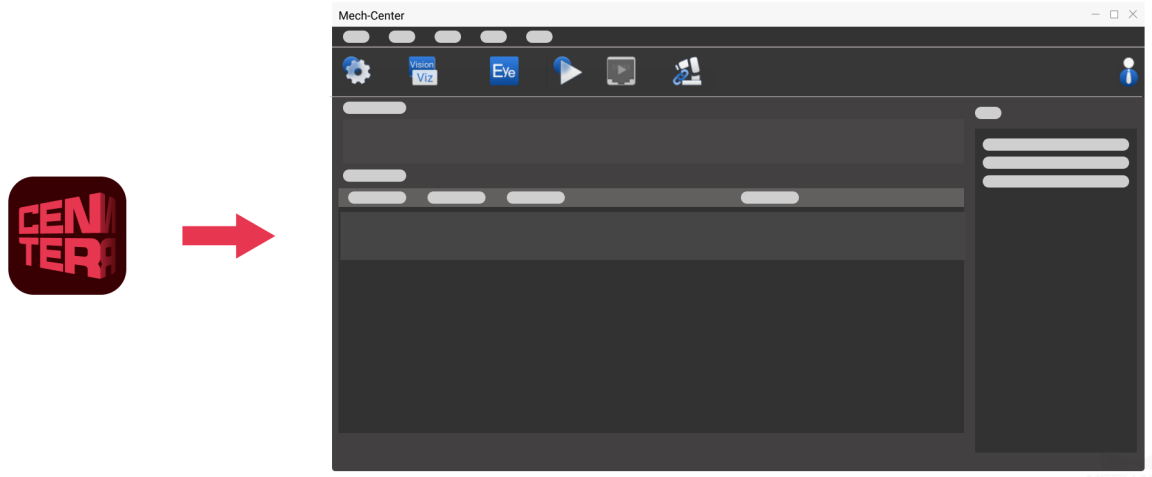
Mech-Center Manual

Mech-Mind

2023 年 08 月 02 日

CONTENTS



Mech-Center は、当社が独自開発した Mech-Mind ソフトウェアの中枢およびコントロールセンターであり、関連するソフトウェアのグローバル設定、プロジェクト全体のバックアップと復元を行うことができ、Mech-Viz、Mech-Vision、Mech-Eye Viewer、ロボット、標準インターフェース、およびアダプターの状態を直感的に確認できます。また、外部インターフェース Mech-Interface の起動および管理センターとしても機能します。

画面紹介および機能 については、以下の内容をお読みください。

クイックスタート

基本的な使用流れ については、以下の内容をお読みください。

Mech-Center を使ってみる

外部インターフェースサービス については、以下の内容をお読みください。

Mech-Interface

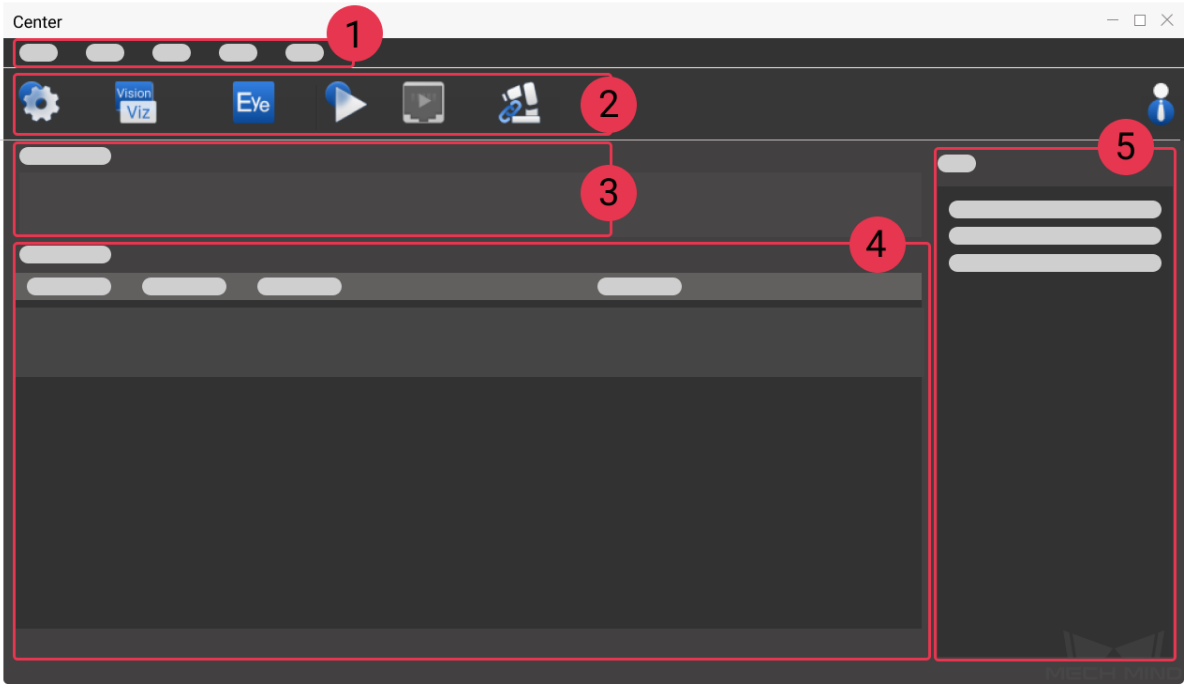
CHAPTER

ONE

クイックスタート

Mech-Center は、当社が独自開発した Mech-Mind ソフトウェアの中枢およびコントロールセンターであり、関連するソフトウェアのグローバル設定、プロジェクト全体のバックアップと復元を行うことができ、Mech-Viz、Mech-Vision、Mech-Eye Viewer、ロボット、標準インターフェイス、およびアダプターの状態を直感的に確認できます。また、外部インターフェイス Mech-Interface の起動および管理センターとしても機能します。

Mech-Center のメインインターフェイス、5 つ部分で構成されます。



1. **メニューバー**：プロジェクトへの操作、ユーザーおよびビューの管理、Adapter 生成および情報確認などの機能が備わっています。
2. **ツールバー**：設定、ソフトウェアおよびインターフェースサービスの起動、ロボット接続、実行などのボタンがあります。
3. **サービスステータスバー**：登録されたソフトウェア、カメラおよびロボットなどが表示されます。
4. **プロジェクトのステータスバー**：Mech-Viz/Mech-Vision プロジェクトのステータス、実行時間および詳細な情報が表示されます。
5. **ログバー**：現在のプロジェクトおよびサービスのログ情報がリアルタイムに表示されます。

1.1 メニューバー

メニューバーには、ファイル、ツール、ユーザー、ビュー、ヘルプなど、操作に関連する基本機能を提供します。

1.1.1 ファイル

オプション	ディスクリプション	ショートカット
プロジェクトをバックアップ	Mech-Mind ソフトウェアシステムのプロジェクトバックアップを生成するために使用されます。自動的に読み込まれた Mech-Viz / Mech-Vision プロジェクト、Mech-Center のアダプタープロジェクト、および Mech-Center の設定が保存されます。	Ctrl+B
プロジェクトを復元	選択したバックアップからプロジェクトと設定を復元するために使用されます。自動的に読み込まれたプロジェクトは復元後に置き換えられるので、プロジェクトをバックアップしておいてください。	Ctrl+R
インポート	その他のパスからプロジェクトをインポートするときに使用されます。	Ctrl+I
終了	Mech-Mind のソフトウェアシステムを終了します。	Ctrl+Q

注意: 操作員モードでは、「バックアップ」と「終了」の 2 つのオプションのみがあります。

1.1.2 ツール

オプション	ディスクリプション
デバッグデータをパッケージ化	Mech-Mind のソフトウェアシステムプロジェクトのデバッグデータをパッケージ化するために使用されます。期間、パッケージパスおよび保存オプションなどが選択可能です。
Adapter ジェネレーター	このオプションは、カスタマイズの Adapter プログラムを生成するために使用されます。
ログビューア	Mech-Viz / Mech-Vision のログメッセージを表示します。手動でログファイルを選択してから、ログファイルを読み込む必要があります。このオプションは、ログレベルに応じて対応するログを表示することができます。また、キーワード（正規表現、大文字と小文字を区別、ワイルドカードが選択可能）を入力してログをフィルタリング・検索することもできます。さらに、タイムラインをスライドして、指定した時間以降のログを選択することもできます。
サービスステータスを表示	ソフトウェア、ロボットおよび各インターフェースなどのサービスの接続ステータスを表示するために使用されます。

注意: 操作員モードで「Adapter ジェネレーター」のオプションはありません。

1.1.3 ユーザー

「パスワードを変更」のオプションがあります。管理者モードでのみ、パスワードを変更できます。操作員モードではこの機能を使用できません。

注意： Mech-Center を産業用コンピュータに初めてインストールする場合、管理者のデフォルトパスワードは「123456」です。また、パスワードを変更した後は、適切に保管してください。

1.1.4 ビュー

「ログ」にチェックを入れるかどうかを選択できます。チェックを入れれば、ログバーがメインインターフェイスで表示されます。

1.1.5 ヘルプ

オプション	ディスクリプション	ショートカット
ユーザーマニュアル	インターフェース関連のドキュメントディレクトリをすばやく開くために使用されます。	F1
更新説明	新たにリリースされたバージョンの新機能とエラー修正について説明します。	なし
バージョン情報	ソフトウェアのバージョンと著作権情報を表示します。	なし

1.2 ツールバー

ツールバーには、「設定」、「起動」、「Mech-Eye Viewer を起動」、「実行」、「インターフェースサービスを起動」、「ロボットを制御」、「管理員」という 7 つのボタンがあります。

1.2.1 設定

これは、Mech-Center の基本設定を実装し、各ソフトウェアのパスを設定し、自動的に読み込まれたプロジェクトパスを表示し、外部サービスを選択するために使用されます。

外観と動作

下図に示すように、「外観と動作」の設定画面は、ユーザーが自分の好みに合わせて設定することができます。例えば、ソフトウェアのテーマ、ソフトウェアの表示言語を切り替えることができます（現在は、中国語、英語、日本語、韓国語の 4 つの言語に対応します）。また、ログ保存の関連設定などを変更できます。

- **実行中にコンソールを非表示** のチェックを入れれば、Mech-Center を起動したら、コンソールが非表示になります。
- *PC* **起動時に** *Mech-Center* **を自動的に実行** にチェックを入れれば、コンピュータを起動すると、Mech-Center も自動的に起動します。
- *Mech-Viz/Mech-Vision/Mech-Interface* **を自動的に起動（もしあれば）** にチェックを入れれば、Mech-Center を起動すると、Mech-Viz/Mech-Vision/Mech-Interface も自動的に起動します。
- *Mech-Viz/Mech-Vision* **を起動するときにシステムトレイに最小化** にチェックを入れれば、起動した Mech-Viz/Mech-Vision がデスクトップステータスバーのトレイに非表示になります。

注意： 変更を保存して Mech-Center を再起動してから、設定が有効になります。

Mech-Viz

下図に示すように、これは Mech-Viz の exe パスを設定し、読み込まれたプロジェクトのパスを自動的に表示するために使用されます。

Mech-Viz をクリックして設定画面に入り、*Mech-Viz* **を使用** のチェックがデフォルトで入れられます。

パスの後にある ... をクリックし、Mech-Viz インストールディレクトリの下の mmind_viz.exe を選択したら、パス選択は完了します。

Mech-Viz ソフトウェアで **現在のプロジェクトを自動的に読み込む** のチェックを入れれば、プロジェクトパスは自動的に読み込まれるので、手動で入力する必要はありません。

注釈：実行をシミュレート にチェックを入れれば、Mech-Viz はロボット実機をシミュレートするだけで、それを移動させることはません。

Mech-Vision 、Mech-Eye Viewer

Mech-Vision、Mech-Eye Viewer の設定は、Mech-Viz とほぼ同じです。

Robot Server

下図に示すように、Mech-Viz ソフトウェアのフルコントロールを実現できるようにロボットを適合させるために使用されます。

Robot Server をクリックして設定画面に入ります。*Robot Server* **を使用** にチェックはデフォルトで入れられます。Mech-Center と共にインストールされた Robot Server パスも自動的に読み込まれます。

注意: Mech-Viz プロジェクトが正常に読み込まれた後、Mech-Viz プロジェクトのロボット型番は自動的に表示されるので、入力する必要はありません。実際のプロジェクトに応じて、ロボット IP アドレスを正しく入力する必要があります。

Mech-Interface

Mech-Interface は第三者との通信を実現するための統一された外部インターフェースです。

1.2.2 起動

Mech-Viz/Mech-Vision ソフトウェアが正常に起動すると、ソフトウェアアイコンがサービスステータスバーに表示されます。

注意:

1. プロジェクトの実行時にバージョン互換性チェックが実行されます。バージョン V1.4.0 以降の Mech-Viz、Mech-Vision および Mech-Center を併用することをお勧めします。
2. Mech-Center が他の 2 つのソフトウェアバージョンが 1.4.0 よりも低いことを検出すると、バージョンは 1.4.0 より高くなければならないことを表すプロンプトが表示されます。**実行** をクリックすると、エラーメッセージボックスがポップアップ表示されます。

1.2.3 Mech-Eye Viewer を起動

Mech-Eye Viewer ソフトウェアが正常に起動すると、ソフトウェアアイコンがデスクトップのステータスバーに表示されます。

1.2.4 実行

Mech-Viz および Mech-Vision で読み込まれたプロジェクトを実行します。

1.2.5 インターフェースサービスを起動

Mech-Interface を起動します。

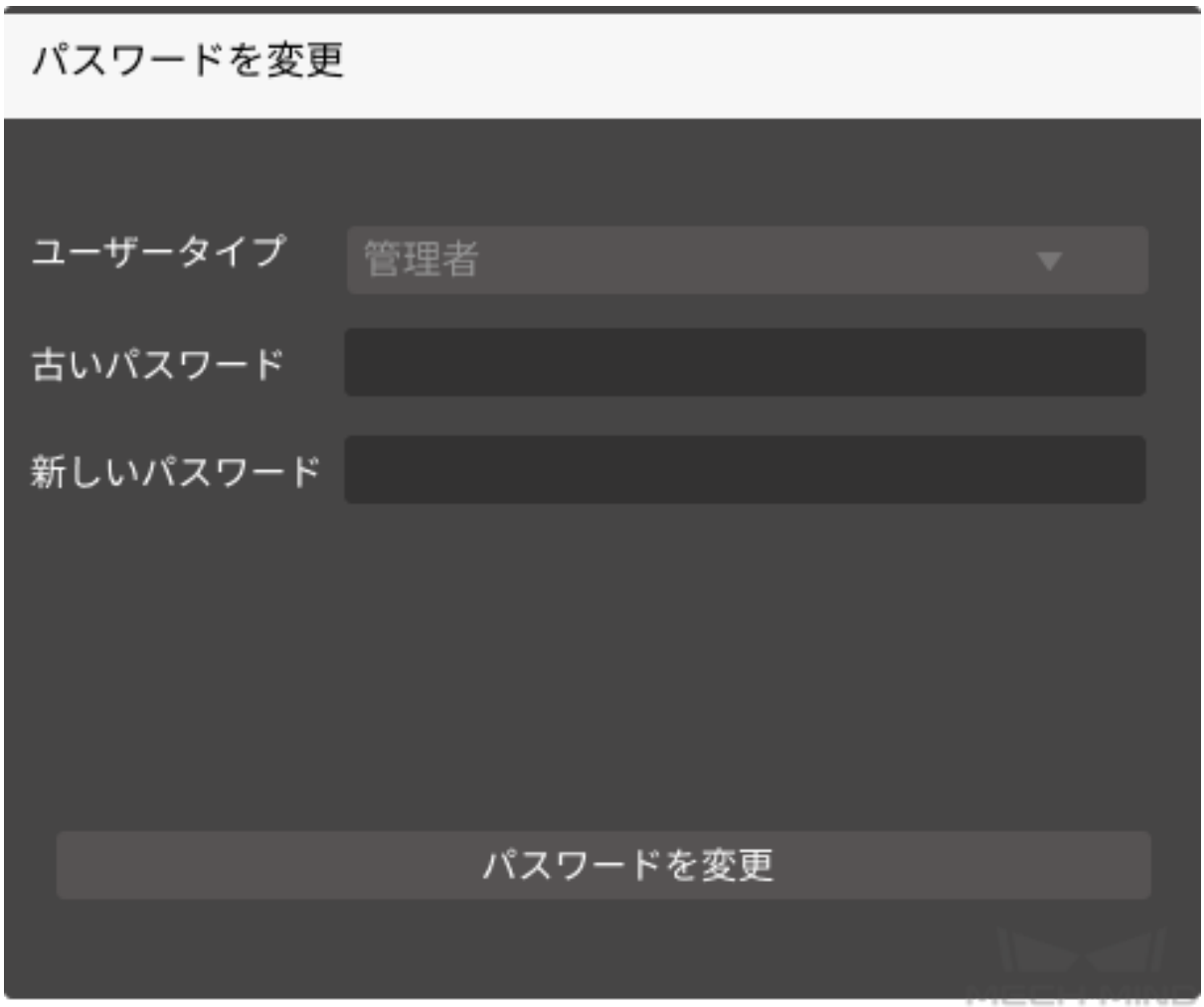
1.2.6 ロボットを制御

ロボット実機が正常に接続すると、ロボットのアイコンおよびその型番はサービスステータスバーに表示されます。

1.2.7 管理者

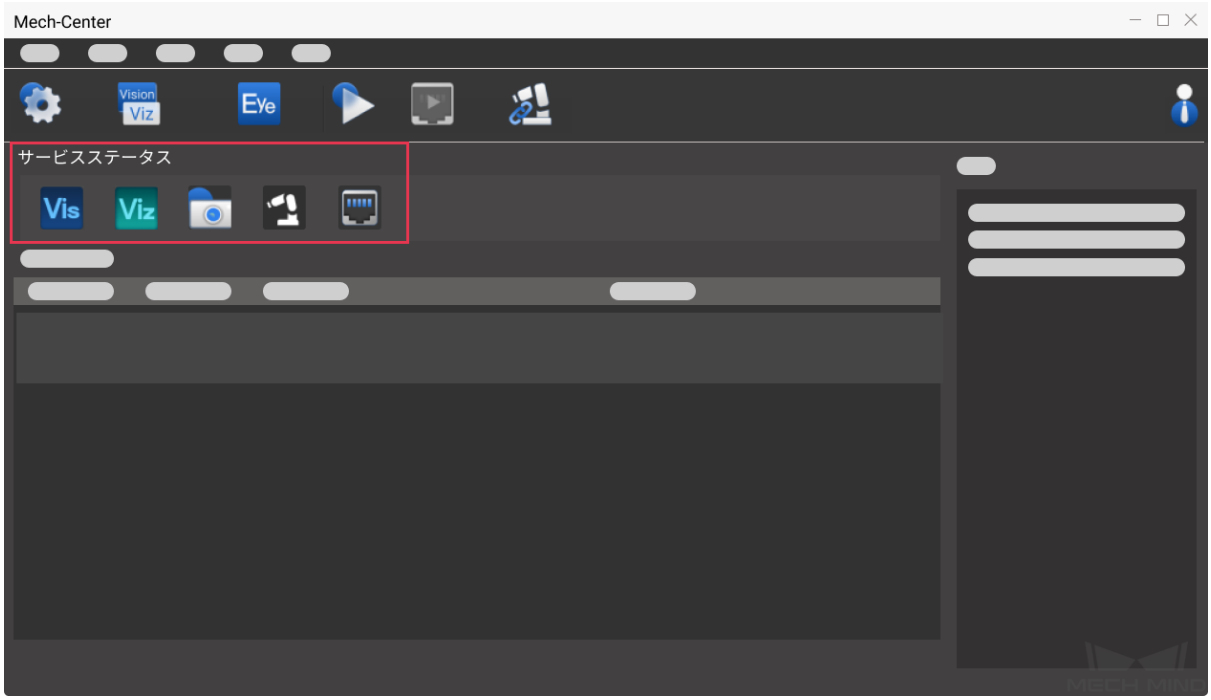
Mech-Center には、「管理者」と「操作員」の 2 つのモードがあり、デフォルトでは「管理者」になります。「操作員」モードに設定すれば、ユーザーがプロジェクトの編集や設定変更を行うことができません。切り替えたい場合、アイコンをクリックし、ダイアログボックスでユーザータイプを選択し、**サインイン** をクリックしてください。

「管理者」に切り替える場合、パスワードが必要です。パスワードを変更する場合はメニューバーで **ユーザー ‣ パスワードを変更** を選択し、パスワードを変更します。



1.3 サービスステータスバー

Mech-Center 実行中、サービスステータスバーに起動されたソフトウェア、カメラ、ロボット、およびインターフェースサービスを表示できます。また、サービスステータスバーでソフトウェアアイコンをクリックして対応する画面に入ることができます。



1.4 プロジェクトのステータスバー

Mech-Viz / Mech-Vision プロジェクト名、ステータス、実行時間および詳細情報が表示されます。通常は右側にあるログバーと組み合わせて、プロジェクトの実行状況を確認します。

オプション	ディスクリプション
プロジェクト名	Mech-Viz / Mech-Vision のプロジェクト名。
ステータス	プロジェクトの現在のステータス（アイドルまたは実行中）。
実行時間	プロジェクトが実行から終了するまでかかる時間。
詳細情報	プロジェクトの実行時間および対応するステータスを記録します。また、プロジェクトが正しく実行されない場合は、エラーメッセージがここに記録されます。

1.5 ログバー

実行中のプロジェクトとサービスのログ情報がリアルタイムに表示されます。

オプション	ディスクリプション
D、i、W、E	4 種類のログレベルです。その中、D はデバッグ情報、I は一般情報、W はワーニング、E はエラーを表します。レベルをクリックしてログメッセージをフィルタリングできます。また、レベルを同時に複数選択に対応します。
クリア	表示されたログをすべてクリアします。
フォルダを開く	Mech-Center インストールパスの `logs` フォルダを開きます。

ちなみに: エラーメッセージのエラーコードをクリックすると、オンラインドキュメントにジャンプして詳細情報を確認できます。

CHAPTER

TWO

MECH-CENTERを使ってみる

2.1 ソフトウェアを設定

下図に示すように、Mech-Center を起動し、ツールバー での  アイコンをクリックして「設定」画面に入ります。この画面で、パス、パラメータなどを設定してから「保存」をクリックします。詳しくは、設定をご参照ください。

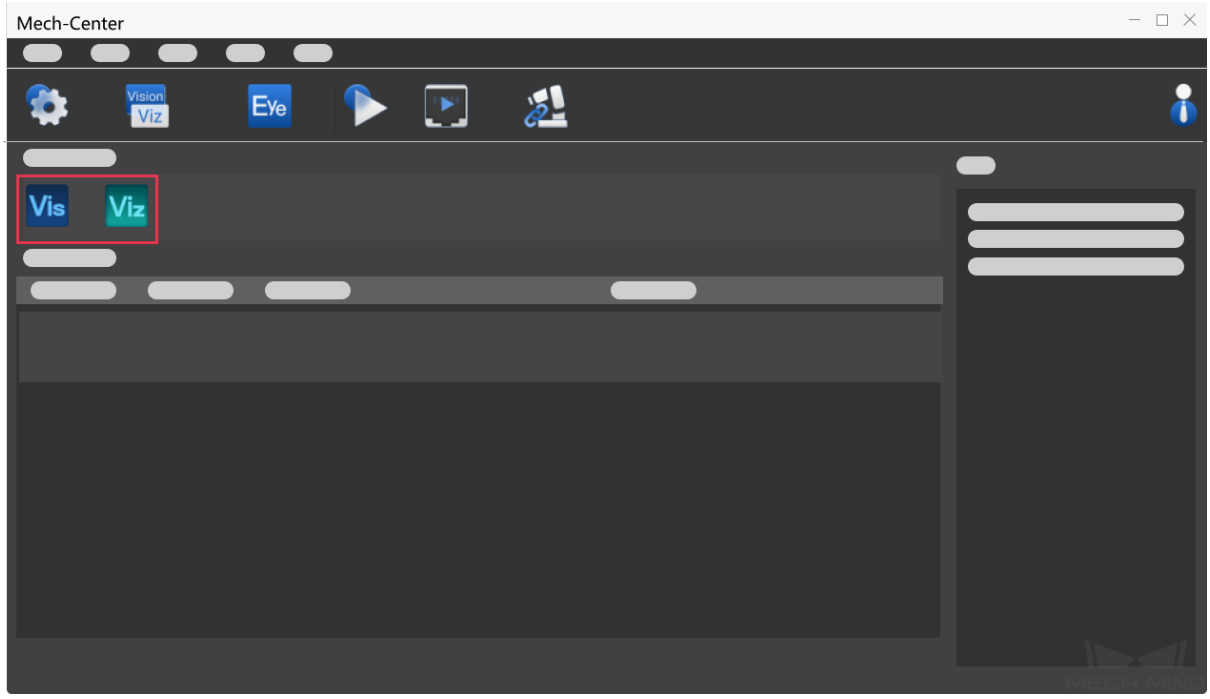
ヒント:

- 実際の状況に応じてソフトウェアを設定してください。
- Mech-Vision、Mech-Viz および Mech-Eye Viewer がインストールされた後、それらのパスが「設定」に自動的に読み込まれるので、手動で入力する必要はありません。

2.2 プロジェクトを開く

ツールバー の 

をクリックすれば、Mech-Viz および Mech-Vision が起動されます。また、それらのアイコンはサービスステータスバーに表示されます。



2.2.1 Mech-Viz プロジェクトを開く

1. ソフトウェアのインターフェイスに入る： 

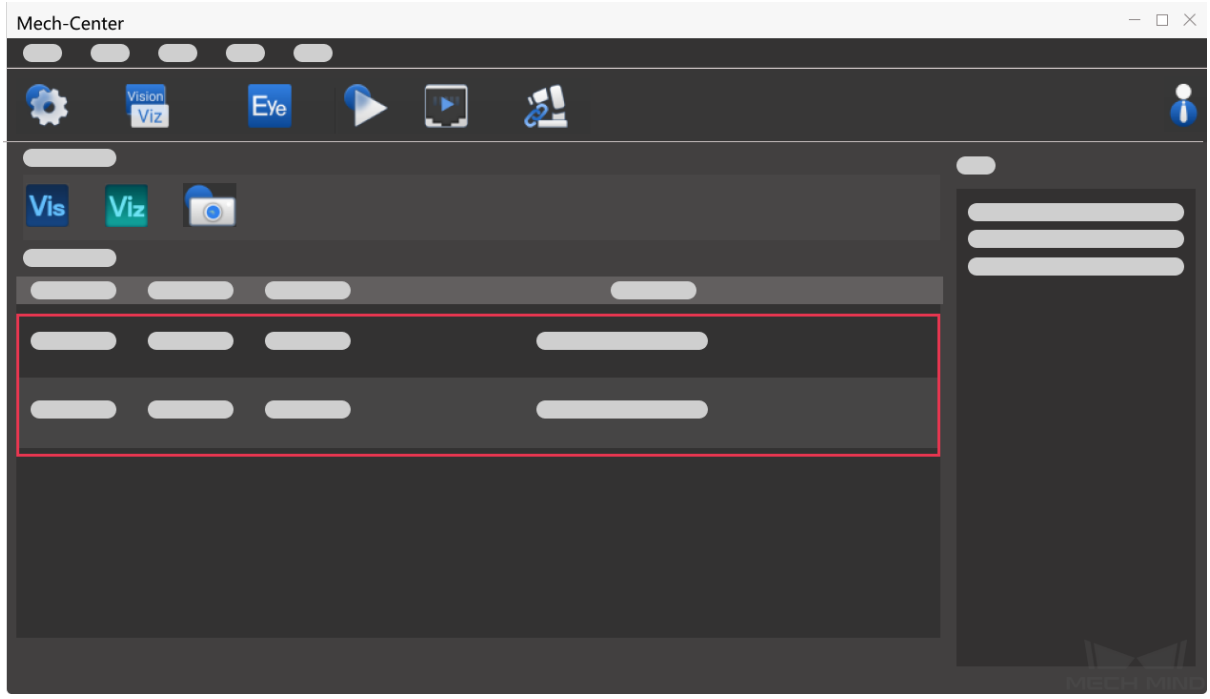
をクリックして Mech-Viz のメインインターフェイスに入ります。
2. プロジェクトを構築: プロジェクトを新規作成するか、既存のファイルを開きます。
3. Mech-Viz プロジェクトリソースのパネルで、プロジェクト名を右クリックして **自動的に読み込む** にチェックを入れます。
4. プロジェクトステータスを表示：現在のプロジェクトの実行ステータスがプロジェクトステータスバーに表示されます。

ヒント:

- **自動的に読み込む** にチェックを入れた後、Mech-Viz プロジェクトパスが Mech-Center の **設定** -> **Mech-Viz** -> **プロジェクトパス** に自動的に記入されます。

2.2.2 Mech-Vision プロジェクトを開く

1. ソフトウェアのインターフェイスに入る：  をクリックして Mech-Vision のメインインターフェイスに入ります。
2. ソリューションまたはプロジェクトを構築：詳細については、ソリューションを新規作成 、プロジェクトを新規作成 をご参照ください。
3. 自動読み込み設定：詳細については、ソリューションの自動読み込み または プロジェクトの自動読み込み をご参照ください。
4. プロジェクトステータスを表示：下図に示すように、現在のプロジェクトの実行ステータスがプロジェクトステータスバーに表示されます。



2.2.3 カメラ設定を変更

実行中にカメラの設定を表示または調整したい場合は、ツールバー での  をクリックしてカメラビューアが起動されると、そのアイコンはサービスステータスバーに表示されます。

2.3 ロボットを接続

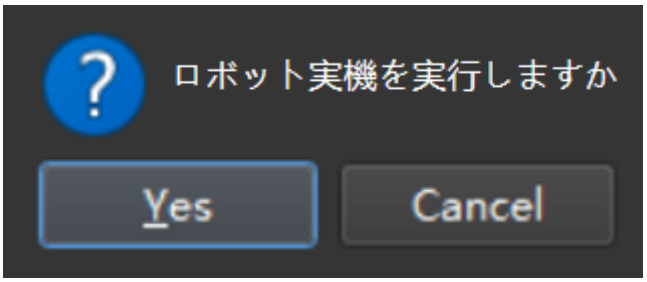
現場で使用するロボットが UR ロボットでない場合、ロボットプログラミングが必要です。ロボットプログラミングについては、robot_integrations をご参照ください。ツールバー での  をクリックしてロボットに正常に接続すると、ロボットのアイコンはサービスステータスバーに表示されます。

Mech-Interface を使用するプロジェクトの場合、ツールバー での  をクリックしてインターフェースサービスが起動されると、対応するアイコンはサービスステータスバーに表示されます。

2.4 プロジェクトを実行

ヒント: ロボット実機を動作させる場合、**「設定」**の「Mech-Viz」画面で **実行をシミュレート** のチェックを外してください。

下図に示すように、ツールバーでの  をクリックして確認ポップアップウィンドウが表示された後、*Yes* をクリックして読み込まれたプロジェクトが実行されます。また、プロジェクトの実行情報がプロジェクトステータスバーに表示されます。



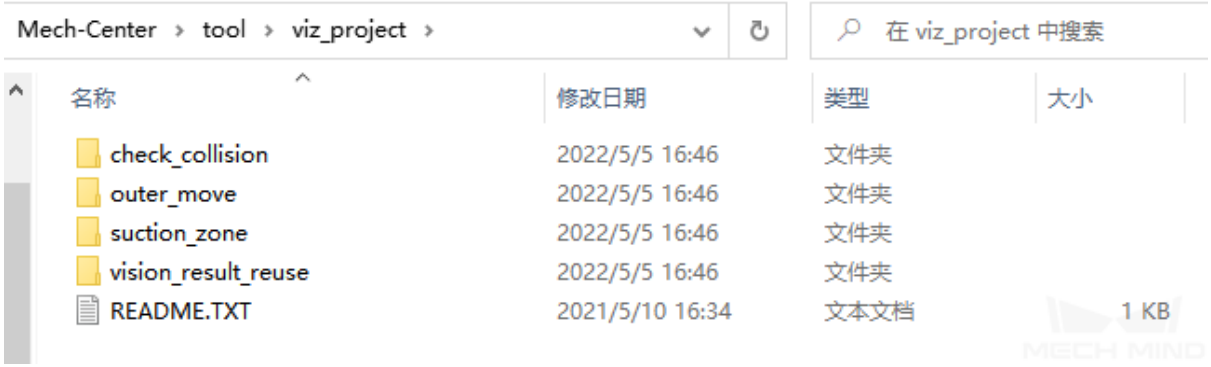
注意:

- プロジェクトが読み込まれた後、プロジェクトを実行します。

  Mech-Viz プロジェクトを開くときに **ソフトウェア互換性の説明** を処理しない場合は、Mech-Center で **実行** をクリックすると、Mech-Center のログに「Mech-Viz はプロジェクトを読み込んでいます。Mech-Viz プロジェクトを起動できません：XXX」というメッセージが表示されます。この時点で、ソフトウェア互換性プロンプトに対処する必要があります。つまり、ソフトウェア互換性プロンプトで *Yes* をクリックし、Mech-Center 実行ボタンを復元して再度実行する必要があります。
- ロボット動作中に安全を必ず確保してください。緊急事態が発生した場合は、ティーチペンダントの非常停止ボタンを押してください！

2.5 標準インターフェースに使用される Mech-Viz サンプルプロジェクト

標準インターフェースに使用されるサンプルプロジェクトは、Mech-Center のインストールディレクトリ (tool¥viz_project) に格納されています。



check_collision

ビジョンシステムによるピッキングの動作計画と衝突検出を行うために使用されます。

outer_move

ロボットが外部クライアントから入力された位置姿勢に移動する必要があるシーンに使用されます。

suction_zone

DO 信号リストを介して複数の吸盤パーティション（または配列ツール）を制御するために使用されます。

vision_result_reuse

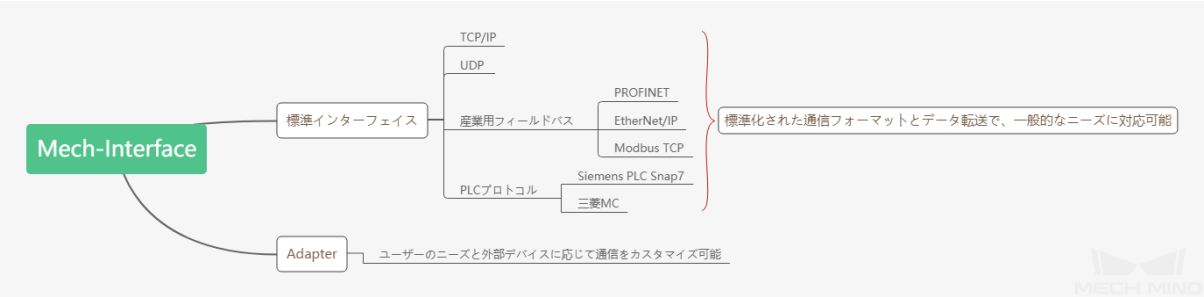
同じ返されたビジョン結果を複数回使用する必要があるビジョンガイドによるピッキングの経路計画と衝突検出を行うために使用されます。

CHAPTER

THREE

MECH-INTERFACE

Mech-Interface は Mech-Mind ソフトウェア システムの外部インターフェースサービスであり、外部通信の架け橋として機能します。外部情報の受信、システム内部情報の送信の機能を果たします。その中には、**標準インターフェース** と **Adapter** が含まれています。



本節では、以下の内容が含まれています。

- 概要：Mech-Interface の 2 種類の通信メカニズムと使用シーンについて説明します。
- 標準インターフェース：標準インターフェースの設定とデプロイ方法について説明します。
- 標準インターフェース開発者向けマニュアル：標準インターフェースの通信プロトコル、コマンド一覧、標準インターフェースのステータスコード一覧とトラブルシューティングについて説明します。
- *Adapter*：Adapter の概要、Adapter ジェネレーターのマニュアル、および Adapter プログラミングガイドについて説明します。

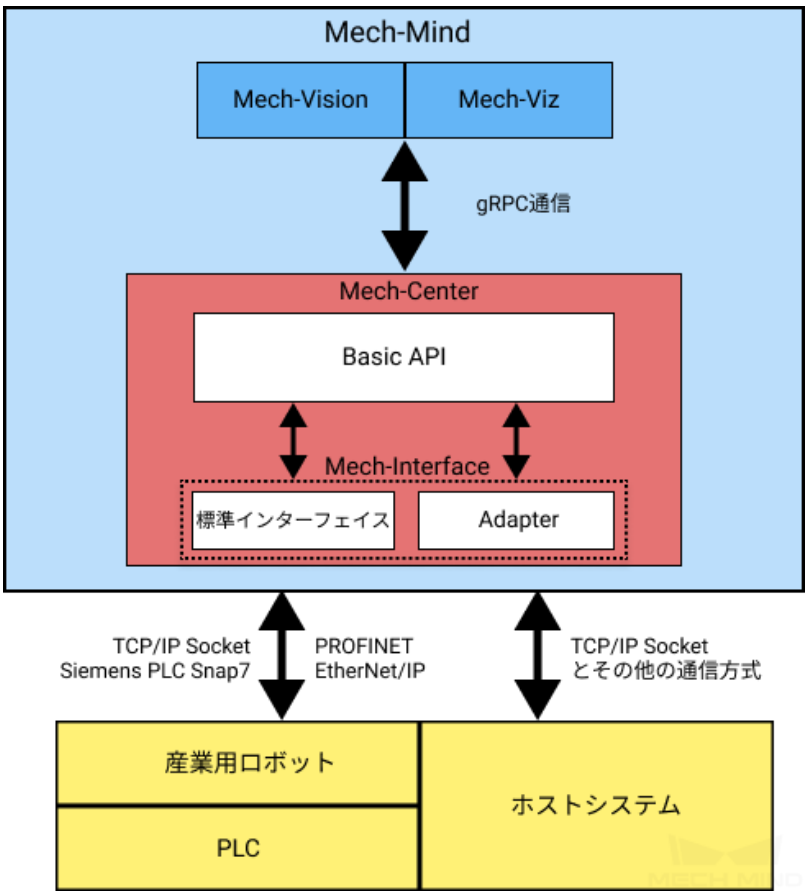
3.1 概要

本節では、Mech-Interface の 2 種類の通信メカニズム、違い、および使用シーンについて説明していきます。

- 通信メカニズム
- 標準インターフェースと *Adapter* の違い
- 使用シーン

3.1.1 通信メカニズム

Mech-Interface の 2 種類の通信メカニズムは下図に示します。



- Adapter は、外部通信デバイス（産業用ロボット、上位システムおよび PLC）と Mech-Vision および Mech-Viz ソフトウェアを接続する Python アダプタープログラムです。Mech-Vision および Mech-Viz と内部的に通信し、Python が実装できる任意の通信プロトコルを介して外部デバイスと通信します。
- 標準インターフェースは、Mech-Mind が提供する完全な Adapter プログラムです。これは、様々な通信プロトコルに対応し、強力な制御コマンドセットと異常警報システムを備えており、ほとんどのニーズを満たすことができます。

3.1.2 標準インターフェースと Adapter の違い

Adapter と標準インターフェースはどちらも、外部デバイスを Mech-Vision および Mech-Viz ソフトウェアと接続するためのアダプタープログラムです。標準インターフェースは Mech-Mind が提供する固定なアダプタープログラムであり、カスタマイズ開発に対応していません。

内部で Basic API を呼び出して Mech-Vision および Mech-Viz と通信し、外部で特定の通信プロトコルを介して外部デバイスと通信します。

標準インターフェースと Adapter の主な違いは次の表に示します。

違い	標準インターフェース	Adapter
内部通信	両方とも、Basic API の呼び出しによって Mech-Vision および Mech-Viz と通信します。	
外部通信プロトコル	標準インターフェースでは、次の通信プロトコルを使用した外部デバイスとの通信のみをサポートします：- TCP/IP Socket - Siemens PLC Snap7 - PROFINET - EtherNet/IP - 三菱 MC - Modbus TCP	Python が実装できる任意の通信プロトコルを介して外部デバイスと通信できます。
機能	視覚結果の出力のみに対応します。	視覚に関する機能だけでなく、ユーザーインターフェイス、データベース、注文システムなどの Python がサポートする機能も提供します。
デプロイメントの柔軟性	簡単で使いやすいであるため、迅速にデプロイできます。	手動作成が必要であり、時間と人件費が高いです。
拡張可能性	機能拡張はサポートされていません。	より多くの通信プロトコルと機能をサポートするように拡張できます。

3.1.3 使用シーン

実際の使用シーンでは、通常、外部通信デバイス、使用する通信プロトコル、およびプロジェクトで必要な通信機能に応じて、どの種類の Mech-Interface を使用するかを決定する必要があります。

標準インターフェースと Adapter がサポートする一般的な通信デバイスと通信プロトコルを次の表に示します。

通信先	通信プロトコル	Mech-Interface の種類	説明
ロボット	TCP/IP Socket	標準インターフェース	Mech-Interface は TCP/IP Socket のサーバーとして機能します。
	UDP		Mech-Interface は UDP のサーバーとして機能します。
	PROFINET		Mech-Interface は PROFINET のスレーブとして機能します。
	EtherNet/IP		Mech-Interface は EtherNet/IP のスレーブとして機能します。
	Modbus TCP		Mech-Interface は Modbus TCP のスレーブとして機能します。
上位システム	HTTP	Adapter	統合プロジェクトに適しており、Viz ティーチング通信を採用しています。
	WebSocket		
	TCP/IP Socket	標準インターフェース	Mech-Interface は TCP/IP Socket のサーバーとして機能します。
PLC	TCP/IP Socket	標準インターフェース	Mech-Interface は TCP/IP Socket のサーバーとして機能します。
	Siemens PLC Snap7		Mech-Interface は Siemens PLC Snap7 のクライアントとして機能します。
	PROFINET		Mech-Interface は PROFINET のスレーブとして機能します。
	EtherNet/IP		Mech-Interface は EtherNet/IP のスレーブとして機能します。
	Modbus TCP		Mech-Interface は Modbus TCP のスレーブとして機能します。
	三菱 MC		Mech-Interface は MC のクライアントとして機能します。

ヒント:

- Mech-Interface を使用して外部デバイスと通信する必要がある場合、標準インターフェースがプロジェクトの要件を満たすことができる場合は、標準インターフェースを使用することをお勧めします。標準インターフェイスがプロジェクトの要件を満たしていない場合（たとえば、外部デバイスが標準インターフェースでサポートされていない通信プロトコルを使用し、プロジェクトが標準インターフェースでサポートされていない機能を使用する必要がある場合）、Adapter が必要です。
- 標準インターフェースがサポートする機能一覧については、**標準インターフェース開発者向けマニュアル** をご参照ください。
- Adapter がサポートする機能一覧については、*Adapter* **の機能** をご参照ください。

標準インターフェースと Adapter の詳細については、以下の内容をお読みください。

- **標準インターフェース**
- *Adapter*

3.2 標準インターフェース

位置姿勢のみを必要とし、Mech-Mind ソフトウェアシステムでロボットの動作を制御する必要がない場合、標準インターフェースを使用してデータを転送することができます。標準インターフェースは、位置姿勢およびタスクデータの送信のような、最も基本的なインターフェース機能のみを実現します。より多くのインターフェース機能が必要な場合、*Adapter* を使用してカスタマイズすることができます。標準インターフェイスが Mech-Center に統合されるので、生成する必要はありません。Mech-Center でそれを起動するだけで使用できます。

3.2.1 Mech-Interface を使用

設定 ‣ *Mech-Interface* で *Mech-Interface* **を使用** にチェックを入れたら、インターフェースサービスのタイプを **標準インターフェイス** に設定します。

3.2.2 インターフェースオプションとホストアドレス

必要に応じて外部サービスタイプを選択することができます。

Siemens PLC Client Siemens PLC Client を選択する場合、ホスト IP アドレス、PLC スロット番号および DB ブロック番号の設定が必要です。

TCP Server TCP Server を選択する場合、ASCII か、HEX か、プロトコル形式を選択する必要があります。HEX のプロトコル形式を使用すれば ビッグエンディアンとリトルエンディアン を選択する必要があります。

また、実際の状況に応じてホストアドレスのポート番号を設定する必要があります。デフォルトのポート番号は 50000 です。

ロボットのサポートに応じて、使用可能な通信形式を選択してインターフェースプログラムを作成することができます。

ロボットタイプ		標準インターフェースのサンプル例
産業ロボット	ABB	HEX パッケージ
	FANUC	HEX パッケージ
	KUKA	HEX パッケージ
	YASKAWA	ASCII パッケージ
	KAWASAKI	ASCII パッケージ
	ROKAE	ASCII パッケージ
	NACHI	ASCII パッケージ
協働ロボット	FANUC CRX	Plugin プラグイン（HEX）
	UR	URCap プラグイン（ASCII）
	TM	Plug and Play プラグイン（ASCII）
	JAKA	ASCII プログラムパッケージ（Addon プラグイン付き）
	ELITE	ASCII パッケージ
その他		ユーザーはロボット側のプログラムを作成する必要があり、Mech-Center はサンプルを提供していません

ちなみに： ロボットインターフェースのサンプルプログラムパッケージと操作ドキュメントは、ソフトウェアインストールディレクトリの「Robot_Interface」フォルダにあります。

PROFINET PROFINET を使用する場合、ロボットの型番を設定する必要があります。

EtherNet/IP EtherNet/IP を使用する場合、ロボットの型番を設定する必要があります。

MODBUS TCP SLAVE Modbus TCP SLAVE は、スレーブ IP、ポート番号、デバイス アドレス、およびバイト順を設定する必要があります（ユーザーは、マッチングするバイト順を選択するためにテストできます）。

3.2.3 ロボットを選択

 をクリックして現場で使用するロボットのブランドと型番を選択します。

設定が完了したら、「保存」をクリックして Mech-Center を再起動します。そして、**インターフェースサービスを起動** をクリックしたら標準インターフェースが起動されます。

3.2.4 詳細設定

毎回送信する位置姿勢の最大数、Mech-Viz からデータ送信を待つタイムアウト時間および Mech-Vision からデータ送信を待つタイムアウト時間が設定可能です。

3.2.5 Mech-Viz のサンプルプロジェクト

Mech-Center のインストールフォルダの `tool¥viz_project` に、標準インターフェイスに使用されるサンプルプロジェクトは 4 つあります。

Check_collision

ビジョンガイドによる把持プロセスの経路計画および衝突検出のシーンに使用されます。

Outer_move

ロボットが外部クライアントから受信された位置姿勢に移動する必要があるシーンに使用されます。

Suction_zone

DO 信号で複数の吸盤（またはアレイグリッパー）を制御するシーンに使用されます。

Vision_result_reuse

経路計画と衝突検出のために一度返された視覚結果を複数回使用する必要があるビジョンガイドによる把持のシーンに使用されます。

3.3 標準インターフェース開発者向けマニュアル

3.3.1 概要

標準インターフェースの種類

標準インターフェースには、外部と接続するためのインターフェースは以下の 7 種類があります。実際のニーズに応じてインターフェースを選択できます。

1. TCP Server

   Mech-Center は、外部と接続するためのインターフェースとして TCP Server を提供します。ASCII や HEX のデータ送信に対応しています。

2. Siemens PLC Client

   Siemens S7 シリーズの PLC と通信するとき、Mech-Center から、SNAP7 プロトコルに基づいた PLC Client が提供され、通信インターフェースとして使用されます。

3. PROFINET

   Mech-Center は、PROFINET スレーブとして PROFINET 産業用ネットワークに接続することができます。PROFINET 通信するために、いくつかの条件があります：

   - IPC（産業用 PC）は PCI-e ボードの搭載に対応しているかどうか、事前にご確認ください。
   - HMS INpact 40 PIR ボードの搭載および Ixxat VCI ドライバのインストールが必要です。
   - 1.5.0 以降のバージョンの Mech-Center を使ってください。ソフトウェアから提供した GSD デバイス記述ファイルを使用してください。

- PROFINET 通信は、標準的なビッグエンドデータフォーマットを使用しています。データは 32 ビットの DINT 位置姿勢データを含んであります。PROFINET マスター（特にロボットコントローラ）は 32 ビット整数の送受信をサポートする必要があります。

4. EtherNet/IP

   Mech-Center は、EtherNet/IP スレーブとして EtherNet/IP 産業用ネットワークに接続することができます。EtherNet/IP 通信するために、いくつかの条件があります：

   - IPC（産業用 PC）は PCI-e ボードの搭載に対応しているかどうか、事前にご確認ください。
   - HMS INpact 40 EIP ボードの搭載および Ixxat VCI ドライバのインストールが必要です。
   - 1.5.1 以降のバージョンの Mech-Center を使ってください。ソフトウェアから提供した EDS デバイス記述ファイルを使用してください。
   - EtherNet/IP 通信は、標準的なビッグエンドデータフォーマットを使用しています。データは 32 ビットの DINT 位置姿勢データを含んであります。EtherNet/IP マスター（特にロボットコントローラ）は 32 ビット整数の送受信をサポートする必要があります。

5. Modbus TCP SLAVE

   Mech-Center はスレーブデバイスとして使用でき、マスターデバイスと通信するための標準インターフェースオプション MODBUS TCP SLAVE を提供します。この機能を使用するには、1.6.1 バージョン以降の Mech-Center が必要です。

6. Mitsubishi MC Client

   Mech-Center はスレーブデバイスとして使用でき、MC（MELSEC 通信）プロトコルでマスターデバイスと通信するための標準インターフェースオプション Mitsubishi MC Client を提供します。この機能を使用するには、1.7.2 バージョン以降の Mech-Center が必要です。

7. UDP Server

   Mech-Center はサーバーとして使用でき、UDP プロトコルでクライアントと通信するための標準インターフェースオプション「UDP Server」を提供します。この機能を使用するには、1.7.2 バージョン以降の Mech-Center が必要です。

標準インターフェースの機能

Mech-Vision 関連

コマンド 101：Mech-Vision を起動 このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使います。対応する Mech-Vision プロジェクトの実行をトリガーして、カメラ撮影と画像処理を行うために使用されます。

コマンド 102：Mech-Vision からビジョン目標点を取得 このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使います。ビジョン認識結果を読み取るために使用されます。

コマンド 103：Mech-Vision のパラメータレシピを切り替える このコマンドは、Vision プロジェクトに保存されたパラメータレシピ（パラメータの設定）を切り替えるために使用されます。複数の部品を認識する際に、モデル認識や DL モデルファイルなど、異なるプロジェクトパラメータを切り替えるために使用されます。

コマンド 105：Mech-Vision の「経路計画」ステップの結果を取得 Mech-Vision でステップ「経路計画」からの衝突ないの動作経路を取得します。

コマンド 110：Mech-Vision からカスタマイズされたデータを取得 このコマンドは、Mech-Vision のステップ procedure_out からカスタマイズされたデータ型のデータを受け取るために使用されます。このコマンドが実行されるたびに、1 つの位置姿勢とそれに対応するラベル、スコアなど (存在する場合) のみがビジョン結果から取得されます。複数の位置姿勢を受け取る必要がある場合は、このコマンドを複数回実行してください。

Mech-Viz 関連

コマンド 201：Mech-Viz を起動 このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使います。Mech-Viz プロジェクトを起動し、対応する Mech-Vision プロジェクトを呼び出し、ロボットの動作経路を計画するために使用されます。

コマンド 202：Mech-Viz を停止 手動で Mech-Viz を終了させます。

コマンド 203：Mech-Viz の分岐を選択 Mech-Viz プロジェクト内に branch_by_msg ステップがある場合に、指定された出口に行くように制御することができます。

コマンド 204：移動インデックスを設定 Mech-Viz プロジェクト内の移動ステップのインデックスを設定するために使用されます。インデックスパラメータ付きの移動ステップは、リストによる移動、グリッドによる移動、カスタマイズのパレットパターンおよび事前計画したパレットパターンが含まれています。

コマンド 205：Mech-Viz の計画経路を取得 Mech-Viz のプロジェクトの計画経路を取得するために使用されます。

コマンド 206：DO リストを取得 吸盤を制御するデジタル出力リストを取得します。複数把持における吸盤の制御に使用するために使用されます。

コマンド 207：Mech-Viz のステップパラメータを読み取る このコマンドは、指定したステップの指定したパラメータの値を読み取るために使用されます。Mech-Center で **設定** ‣ *Mech-Interface* ‣ **詳細設定** ‣ **プロパティ構成ファイル** をクリックして、どのステップのどのパラメータを読み取るかを指定します。

コマンド 208：Mech-Viz のステップパラメータを設定 このコマンドは、指定したステップの指定したパラメータの値を設定するために使用されます。Mech-Center で **設定** ‣ *Mech-Interface* ‣ **詳細設定** ‣ **プロパティ構成ファイル** をクリックして、どのステップのどのパラメータを設定し、どの値を設定するかを指定します。

コマンド 210：移動目標点と視覚処理による計画結果を取得 このコマンドは、Mech-Viz によって計画された単一の目標点を取得するために使用されます。目標点は、視覚移動目標点または他の移動ステップの目標点にすることができます。目標点には、位置姿勢、速度、ツール情報、ワーク情報などを含めることができます。

動的なデータ受信

コマンド 501：Mech-Vision へ対象物の寸法を送信 このコマンドは、対象物の 3D サイズを設定するために使用されます。Mech-Vision プロジェクトで、read_object_dimensions ステップがあり、対象物の寸法（例：箱の寸法：長さ、幅、高さ）を外部から入力する必要がある場合、このコマンドが使われます。

コマンド 502：Mech-Viz へ TCP を送信 Mech-Viz プロジェクトで動的に変化する移動目標点を設定します。Mech-Viz プロジェクトに outer_move ステップが必要です。

カスタマイズ通知

コマンド 601：通知メッセージ ユーザーがこのコマンドを呼び出す必要はありません。Mech-Viz/Mech-Vision プロジェクトが notify_viz または notify_vision に実行されると、Mech-Center は定義されたメッセージをクライアントに送信します。

キャリブレーション

コマンド 701：キャリブレーション カメラとロボットのキャリブレーションに使用されます。ロボットからキャリブレーションポイントが請求されると、カメラは撮影し始め、キャリブレーションのプロセスを完成させます。キャリブレーションポイントは Mech-Vision から提供されます。

システムステータスチェック

コマンド 901：ソフトウェアのステータスを取得 Mech-Mind ソフトウェアシステムの状態を取得します。

> **注意:** 通信には、統一されたデータ単位が必要です。
>
> - 関節角度とオイラー角の単位は度（°）です。
> - フランジ位置姿勢の XYZ 座標の単位はミリメートル（mm）です。

サンプル紹介

標準インターフェースのサンプルについては、標準インターフェース をご参照ください。

3.3.2 TCP/IP コマンド

本節では、TCP/IP の通信プロトコルを基づいた標準インターフェースのコマンドについて説明します。

コマンド 101——Mech-Vision プロジェクトを起動

このコマンドは、Mech-Vision のプロジェクトを起動し、カメラの撮影および視覚認識を行う場合に使われます。

プロジェクトは Eye In Hand モードである場合、このコマンドを用いてロボット撮影の位置姿勢をプロジェクトへ送信します。

このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使われます。

送信コマンド

101, Mech-Vision **プロジェクト番号, ビジョン目標点の期待数, ロボット位置姿勢のタイプ, ロボット位置姿勢**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

ビジョン目標点の期待数

Mech-Vision から取得したいビジョン目標点の数です。ビジョン目標点情報に、ビジョン目標点及びそれに対応する点群、ラベル、スケーリングの情報が含まれています。

- 0：Mech-Vision プロジェクトで認識できたすべてのビジョン目標点を取得します。
- 0 より大きな整数：Mech-Vision プロジェクトで認識できた指定数のビジョン目標点を取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョン目標点の合計数より大きい場合、認識結果にあるすべてのビジョン目標点を取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョン目標点の合計数より小さい場合、このパラメータで指定された数のビジョン目標点を取得します。

ヒント: ビジョン目標点を取得するコマンドは 102 コマンドです。TCP/IP では、102 コマンドを一度実行して最大 20 個のビジョン目標点を取得できます。102 コマンドを一回実行した後、返された 1 つのパラメータに、リクエストされたすべてのビジョン目標点が返されたかどうかが反映されます。そうでなければ、102 コマンドを繰り返し実行してください。

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Vision に送信するタイプを設定します。パラメータ範囲は 0~3 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Vision にロボットの位置姿勢を送信する必要がありません。	プロジェクトは、Eye To Hand モードです。Mech-Vision プロジェクトで「経路計画」ステップを使用する場合、経路計画の開始位置は、経路計画ツールで設定した初期位置です。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Vision にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。この設定は、直行ロボット以外のほとんどのロボットで利用可能です。
2	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在のフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。ロボットは関節角度のデータを持たず、フランジ位置姿勢データのみを持ちます（直行ロボットの場合など）。
3	ロボット経路計画の開始位置の関節角度	Mech-Vision にロボット経路計画の開始位置の関節角度を送信する必要があります。	プロジェクトは、Eye To Hand モードです。また、Mech-Vision プロジェクトに「経路計画」ステップがあり、ロボット側から「経路計画」ステップの開始位置を設定する必要があります。

ヒント： 位置姿勢は位置と姿勢からなり、位置の単位はミリメートル（mm）です。姿勢はオイラー角で表し、単位は度（°）です。関節角度の単位は度（°）です。

返されたデータ

101, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**1102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

Eye In Hand の場合、コマンドはロボットの現在の関節角度とフランジ位置姿勢を送信し、正常実行例を以下に示します。

```
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
```

Eye To Hand の場合、コマンドはティーチングされた関節角度を送信し、正常実行例を以下に示します。

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

次の例は、番号 2 のビジョンプロジェクトが登録されていないことを示すコマンド実行例外です。

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

コマンド 102——Mech-Vision のビジョン目標点を取得

コマンド 101——Mech-Vision プロジェクトを起動 の後に使用します。このコマンドは、Mech-Vision からビジョン目標点を取得してビジョン目標点に変換するために使用されます。

以下に、ビジョン目標点に含まれる位置姿勢をロボット TCP に変換する処理を示します。

- ビジョン目標点に含まれる位置姿勢を Y 軸を中心に 180° 回転させます。
- 対応するロボット型番の基準座標系にロボットベースの高さが含まれているかどうかを認識し、それに応じて垂直方向のオフセットを増やします。

ヒント: デフォルトでは、102 コマンドは毎回最大 20 個までのビジョン目標点を取得することができます。20 個以上のビジョン目標点を取得するには、すべてのビジョン目標点を得るまで、102 コマンドを繰り返し実行してください。

送信コマンド

102, Mech-Vision **プロジェクト番号**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

102, **ステータスコード, 送信が完了しているかどうか, ビジョン目標点の数, 予約語, ビジョン目標点, ビジョン目標点, ..., ビジョン目標点**

注釈: ビジョン目標点データは、返されたデータパラメータの最後に位置します（毎回 20 個まで返されます）。ビジョン目標点には、TCP、ラベルおよび速度（速度値はゼロ）が含まれます。

ステータスコード

コマンドが正常に実行された場合、**1100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

送信が完了しているかどうか

このパラメータは、取得したいすべてのビジョン目標点が得られたかどうかを表します。値は 0 か 1 となります。

- `0`：取得したいすべてのビジョン目標点が得られていないことを意味します。このパラメータの値が 1 になるまで 102 コマンドを繰り返し実行してください。
- `1`：取得したいすべてのビジョン目標点が得られたことを意味します。

101 **コマンド** で、指定したビジョン目標点の期待数が 20 個以上ある場合、このパラメータ値によって、送信されなかったビジョン目標点があるかどうかを確認することができます。送信されなかったビジョン目標点がある場合、102 コマンドを繰り返し実行して、受信を継続することができます。

ヒント： すべてのビジョン目標点を取得していなくて、101 コマンドを再実行すると、取得しなかったビジョン目標点のデータがなくなります。

ビジョン目標点の数

このコマンドを実行して、取得したビジョン目標点の数です。

- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョン目標点の数よりも多い場合、Mech-Vision によって認識されたビジョン目標点の数に従って送信されます。
- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョン目標点の数よりも少ない場合、リクエストした数に従って送信されます。

デフォルトの範囲は 0~20 です。

予約語

この予約語が使われていないため、初期値は 0 です。

ビジョン目標点

1 つのビジョン目標点は 8 つのデータで構成され、最初の 6 個が TCP、7 番目がラベル、最後が速度を示します。

- **TCP**：TCP には 3 次元座標（XYZ、ミリメートル単位）とオイラー角（ABC、単位は度）が含まれます。
- **ラベル**：位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。
- **速度**：このパラメータの初期値はゼロです。通常、Mech-Vision によって出力されるビジョン結果には、目標点の速度情報は含まれていません。

実行例

コマンドが正常に実行された場合

```
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
```

コマンドの実行でエラーが発生したため、ビジョン結果が出てきませんでした。

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

ビジョン目標点をリクエストする実行例

次の実行例に従って、コマンド 101、102、102 の順でコマンドを実行し、22 個のビジョン目標点を取得します。

- TCP/IP はコマンド 101 を送信します。コンテンツは `101, 1, 0, 1, …` で、すべてのビジョン目標点の取得をリクエストします。

- TCP/IP はコマンド 102 を送信し、20 個のビジョン目標点を取得します。
- TCP/IP はコマンド 102 からのデータを受信します。コンテンツは 102, 1100, 0, 20, … で、すべてのビジョン目標点が取得していません。20 個のビジョン目標点を含んだデータが返されます。
- TCP/IP はコマンド 102 を再送信します。残りのビジョン目標点を取得します。
- TCP/IP はコマンド 102 からのデータを受信します。コンテンツは 102, 1100, 1, 2, … で、2 個のビジョン目標点を含んだデータが返され、すべての必要なビジョン目標点が送信されたことを表します。

コマンド 101 を送信します。

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

コマンド 102 を送信し、20 個のビジョン目標点を取得します。

```
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
```

もう一度コマンド 102 を送信し、残りの 2 つのビジョン目標点を取得します。

```
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
```

コマンド 103——Mech-Vision のパラメータレシピを切り替える

Mech-Vision プロジェクトに使われるパラメータレシピを切り替える際にこのコマンドを実行してください。

パラメータレシピを切り替えることで、Mech-Vision プロジェクトの各ステップのパラメータを変更することができます。

パラメータレシピには点群マッチングモデル、画像マッチングテンプレート、ROI、信頼度のしきい値などのパラメータの設定が含まれています。

注意： **コマンド** *101——Mech-Vision* **プロジェクトを起動** を実行する前に、このコマンドを使用する必要があります。

送信コマンド

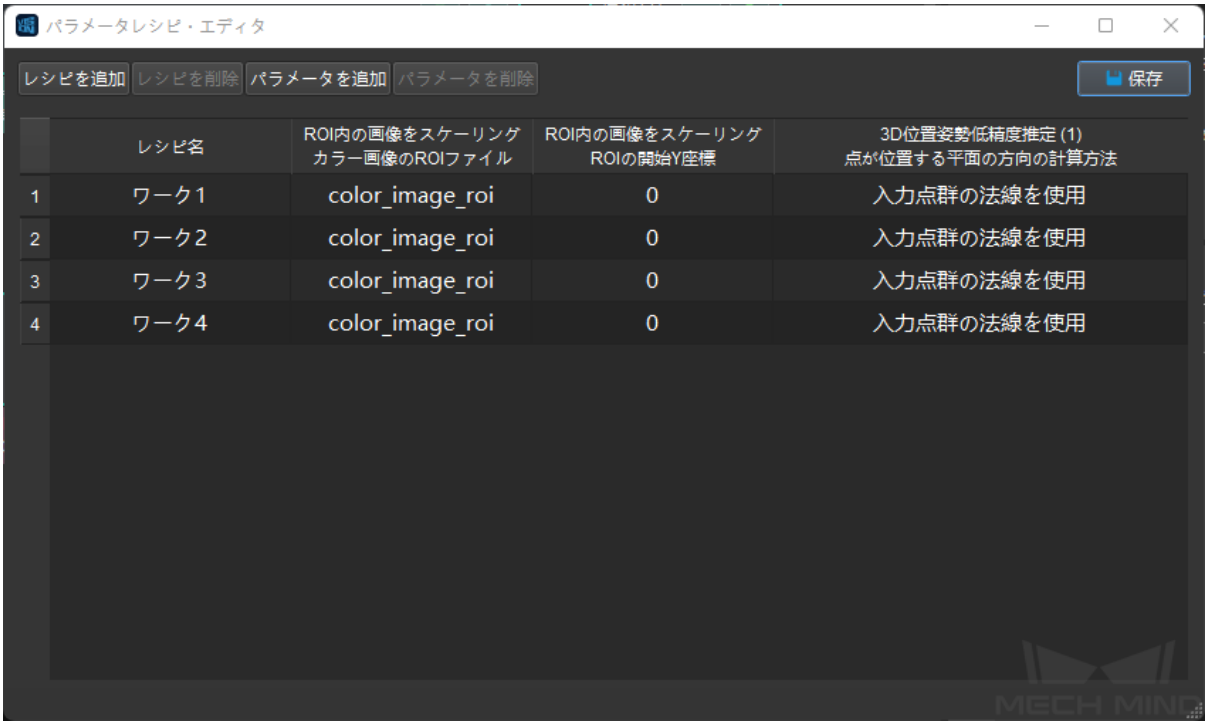
103, Mech-Vision **プロジェクト番号, レシピ番号**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

レシピ番号

Mech-Vision プロジェクトのレシピテンプレートの番号（正の整数）です。**プロジェクトアシスタント‣パラメータレシピ** をクリックして、パラメータレシピエディタに入ります。番号の有効範囲は 1~99 です。



返されたデータ

103, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**1107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

エラーが発生した場合

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

コマンド 105——Mech-Vision の「経路計画」ステップの結果を取得

101 コマンドを呼び出した後、このコマンドを使用して Mech-Vision の「経路計画」ステップから出力された衝突ないの把持経路を取得します。

このコマンドを使用する時、Mech-Vision の「出力」ステップの **ポートタイプ** を「事前定義済み（ロボット経路）」に設定する必要があります。

ヒント： 105 コマンドを呼び出す前に、105 コマンドの呼び出し回数を減らすように 101 コマンドの **ビジョン目標点の期待数** を 0 に設定する必要があります。101 コマンドの **ビジョン目標点の期待数** を 1 に設定すると、105 コマンドの呼び出しごとに 1 つの経路点のみが返され、105 コマンドを複数回呼び出した場合にのみすべての経路点が返されます。

送信コマンド

105, Mech-Vision **プロジェクト番号, 経路点の位置姿勢タイプ**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

経路点の位置姿勢タイプ

このパラメータは、「経路計画」ステップから返された経路点の位置姿勢タイプを指定するために使用されます。

- `1`：経路点の位置姿勢は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点の位置姿勢は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

***105*，ステータスコード，送信が完了したかどうか，経路点の数，「ビジョン処理による移動」の位置，経路点，経路点，...，経路点**

ステータスコード

コマンドが正常に実行された場合、**1103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

送信が完了しているかどうか

- `0`：経路にあるすべての経路点が送信されいないため、このパラメータの値が 1 になるまでに、このコマンドを繰り返して実行してください。
- `1`：経路にあるすべての経路点が送信されました。

経路点の数

このパラメータは、このコマンドを実行した後に返された経路点の数を表します。範囲は 0~20 です。20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

経路計画設定ツールで設定された「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

経路点

1 つの経路点は 8 つのデータで構成され、最初の 6 個は位置姿勢、7 番目はラベル、最後は速度を表します。

- **位置姿勢**：3 次元座標（ミリメートル単位）、オイラー角（度）、または関節角度（度単位）。位置姿勢の形式は、105 コマンドの **経路点のタイプ** で決定されます。
- **ラベル**：位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に ラベルマッピング ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。
- **速度**：経路計画設定ツールで設定された速度値です。

実行例

コマンドが正常に実行された場合

```
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
```

エラーが発生した場合

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

コマンド 110 —— Mech-Vision からカスタマイズされたデータを取得

このコマンドは、Mech-Vision の procedure_out ステップからカスタマイズされたデータを受け取るために使用されます。つまり、poses と labels 以外のポートのデータを受け取ります（ステップパラメータ「ポートタイプ」を「カスタム」に設定する場合）。

このコマンドが実行されるたびに、1 つの位置姿勢とそれに対応するラベル、スコアなど (存在する場合) のみがビジョン結果から取得されます。複数の位置姿勢を受け取る必要がある場合は、このコマンドを繰り返し実行してください。

送信コマンド

110, Mech-Vision **プロジェクト番号**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

110, **ステータスコード, 位置姿勢の送信が完了するかどうか, カスタマイズされたデータ項目数, 対象物の位置姿勢が対応するロボットツール位置姿勢（*TCP*）, ラベル, カスタマイズされたデータ項目**, ..., **カスタマイズされたデータ項目**

ステータスコード

コマンドが正常に実行された場合、ステータスコード 1100 が返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

位置姿勢の送信が完了するかどうか

- `0`：ビジョン結果にまだ送信されていない位置姿勢があります。
- `1`：ビジョン結果のすべての位置姿勢が送信されました。

カスタマイズされたデータ項目の数

位置姿勢とラベル以外のデータ型のカスタマイズされたデータ項目に合計数です。

対象物の位置姿勢が対応するロボットツール位置姿勢（TCP）

ロボット座標系でのツール位置姿勢（TCP、3 次元座標はミリメートル、オイラー角は度単位）です。Mech-Vision は対象物の位置姿勢を出力し、その位置姿勢をロボットツール位置姿勢に変換します。したがって、ステップ procedure_out には 1 つの位置姿勢ポートが必要です。

通常、対象物の位置姿勢に対応するツール位置姿勢は、対象物の位置姿勢の Z 軸を反転させることによって生成されます。

ラベル

位置姿勢に対応する対象物情報ラベルです。ラベルは正の整数である必要があります。それ以外の場合は、Mech-Vision プロジェクトの label_mapping ステップを使用して正の整数にマッピングする必要があります。procedure_out ステップにラベルポートがない場合、このフィールドは 0 で埋められます。

カスタマイズされたデータ項目

これは、Mech-Vision プロジェクトの procedure_out ステップでポートタイプが「カスタム」に設定された場合に出力された位置姿勢とラベル以外のデータです。

カスタマイズされたデータは、ポート名の A – Z の順に並べられています。

コマンド 201——Mech-Viz プロジェクトを起動

このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使われます。Mech-Viz プロジェクトを実行し、対応する Mech-Vision プロジェクトを呼び出し、Mech-Viz が Mech-Vision のビジョン結果に基づいて経路を計画する時に使用されます。

Mech-Viz では、**自動的に読み込む** にチェックを入れる必要があります。

Mech-Mind ソフトウェアシステムのインストールディレクトリ `Mech-Center\tool\viz_project` フォルダには、サンプルプロジェクトが格納され、それら基づいて修正することが可能です。

標準インターフェースに使用される Mech-Viz サンプルプロジェクトの詳細な説明については、**標準インターフェースに使用される *Mech-Viz* サンプルプロジェクト** をご参照ください。

送信コマンド

201, **ロボット位置姿勢のタイプ**, **ロボット位置姿勢**

ロボット位置姿勢のタイプ 、**ロボット位置姿勢**

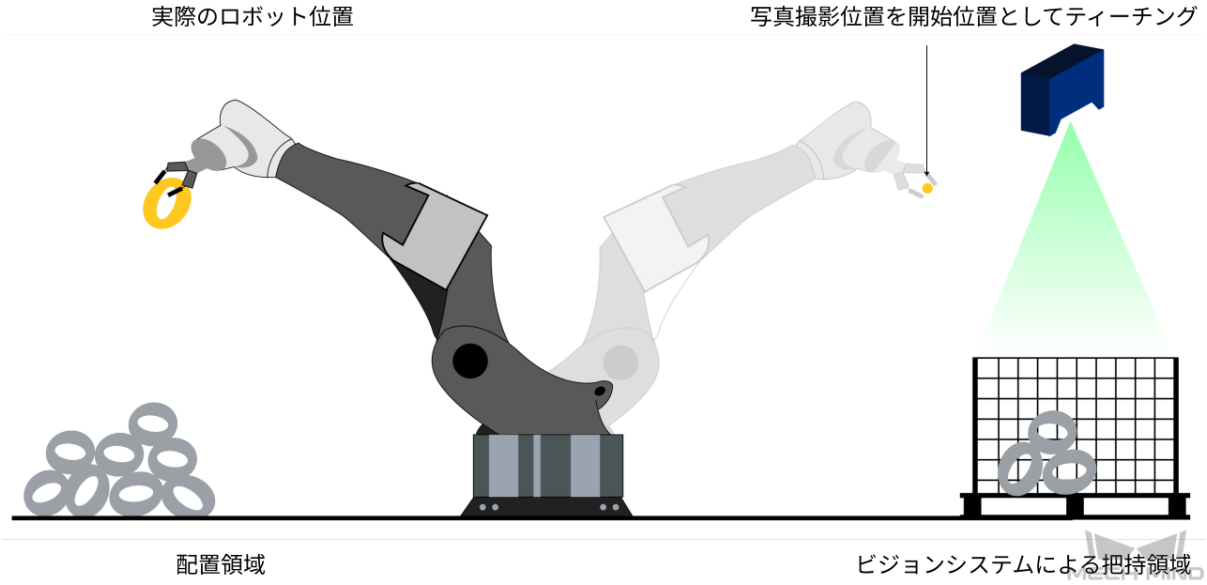
- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Viz に送信するタイプを設定します。パラメータ範囲は 0~2 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Viz にロボットの位置姿勢を送信する必要がありません。Mech-Viz での仮想ロボットは初期位置姿勢（JPs = [0, 0, 0, 0, 0, 0] ）から最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨しません。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Viz にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があり、Mech-Viz での仮想ロボットは受信された位置姿勢から最初の経路点に移動します。	プロジェクトは Eye In Hand モードである場合は、この設定は推奨します。
2	ロボット側でカスタマイズされた関節角度	Mech-Viz にロボットのティーチポイント（現在の関節角度ではない）を送信する必要があります。これは、ロボットが画像撮影領域の外にいるとき（下図に示す）、Mech-Viz プロジェクトが次回の経路を事前に計画することをトリガーするために使用されます。Mech-Viz での仮想ロボットは受信された最初のティーチポイントから最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨します。

Eye To Hand モードでは、**ロボット位置姿勢のタイプ** が 2 に設定されている理由は以下の通りです。

> Eye To Hand モードでは、カメラはロボットが画像撮影領域と把持領域に戻る前に撮影し、次回の把持経路を計画することができます。これにより、タクトタイムの向上を実現します。
>
> この時、**ロボット位置姿勢のタイプ** が 1 に設定され、つまり現在の位置姿勢を Mech-Viz に送信すれば、仮想ロボットがロボット実機の経路と一致しない可能性があります。また、未知の衝突が発生する可能性もあります。
>
> 仮想ロボットは現在の位置姿勢から Mech-Viz での最初の移動ステップで設定された位置姿勢に移動しますが、ロボット実機は上記の位置姿勢に移動する前に別の位置姿勢に移動する可能性があるということです。
>
> したがって、**ロボット位置姿勢のタイプ** パラメータを 2 に設定する必要があります。



ヒント： 位置姿勢は位置と姿勢からなり、位置の単位はミリメートル（mm）です。姿勢はオイラー角で表し、単位は度（°）です。関節角度の単位は度（°）です。

返されたデータ

201, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**2103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

Eye In Hand の場合、コマンドはロボットの現在の関節角度とフランジ位置姿勢を送信し、正常実行例を以下に示します。

```
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
```

Eye To Hand の場合、コマンドはティーチングされた関節角度を送信し、正常実行例を以下に示します。

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

以下のサンプルでは、コマンドは必要な位置姿勢データを送信しないため、実行例外となります。

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

コマンド 202——Mech-Viz プロジェクトを停止

Mech-Viz プロジェクトの実行を停止します。Mech-Viz プロジェクトは無限ループになっていない場合や、正常に停止できる場合は、このコマンドを使用する必要がありません。

送信コマンド

202

返されたデータ

202, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**2104** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

コマンド 203——Mech-Viz 分岐を選択

このコマンドは、Mech-Viz プロジェクトの分岐を指定する場合に使われます。分岐メカニズムは branch_by_msg によって作成されたら、このコマンドがステップの出口を指定することで実現します。

このコマンドを実行する前に、**コマンド *201——Mech-Viz* プロジェクトを起動** を実行してください。

Mech-Viz プロジェクトが branch_by_msg に実行すると、このコマンドによって出口を指定するのを待ちます。

送信コマンド

203, **分岐ステップ *ID*, 出口番号**

分岐ステップ ID

このパラメータは、分岐選択が行われる branch_by_msg を指定するために使用されます。

このパラメータ、つまり、branch_by_msg のステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

分岐の出口番号

このパラメータは、プロジェクトが branch_by_msg ステップに沿って実行される出口を指定します。Mech-Viz プロジェクトはこの出口に従って実行し続けます。パラメータの値は正の整数です。

ヒント：

- 分岐の出口番号は Mech-Viz で表される番号 + 1 になります。例えば、番号が 0 の場合、出口番号は 1 です。

- 出口番号は、1 から始まるポートのインデックス番号です。たとえば、指定された出口が左から右に 2 番目のポートである場合、出口番号は 2 です。

返されたデータ

203, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**2105** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

エラーが発生した場合

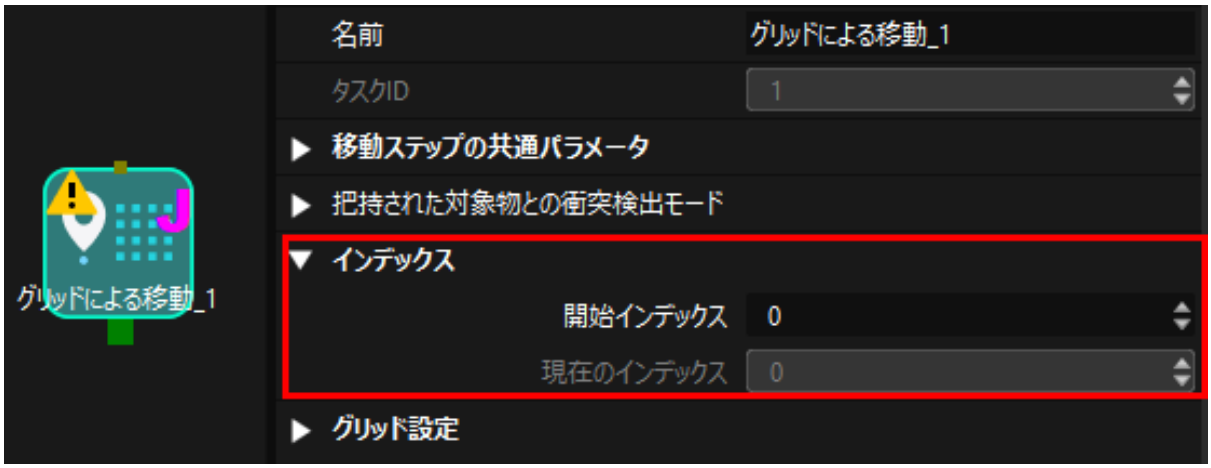
```
TCP send string = 203, 1, 3
TCP received string = 203, 2018
```

コマンド 204——移動インデックスを設定

このコマンドはステップのインデックスパラメータを設定するために使用されます。通常、連続した、あるいは個別に指定された移動などの操作に使用されます。

インデックスパラメータが付いたステップは「リストによる移動」、「グリッドによる移動」、「カスタマイズのパレットパターン」、「事前計画したパレットパターン」などです。

このコマンドを実行する前に、**コマンド** *201——Mech-Viz* **プロジェクトを起動** を実行してください。



送信コマンド

204, **ステップ** *ID,* **インデックス値**

ステップ番号

このパラメータはどのステップがインデックスを設定する必要かを指定します。

このパラメータ、つまり、インデックス付きのステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

インデックス値

次にこのステップが実行されたときに設定されるべきインデックス値です。

このコマンドを送信すると、Mech-Viz の現在のインデックス値がこのパラメータの値から 1 を引いた値に変更されます。

このコマンドで指定したステップに Mech-Viz プロジェクトが実行されると、Mech-Viz の現在のインデックス値が、このパラメータの値まで 1 つずつ増加します。

返されたデータ

204, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**2106** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

エラーが発生した場合

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

コマンド 205——Mech-Viz の計画経路を取得

コマンド *201——Mech-Viz* **プロジェクトを起動** を実行した後、このコマンドは Mech-Viz が計画した経路を取得するために使用されます。

初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

注釈: プロジェクト内の移動ステップの経路点をロボットに送信しない場合は、ステップパラメータで「移動目標点を送信」のチェックを外してください。

送信コマンド

205，**経路点タイプ**

経路点タイプ

このパラメータは Mech-Viz からどのような形式で経路点を返すかを指定します。

- `1`：経路点は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

205，**ステータスコード，送信が完了したかどうか，経路点の数，「ビジョン処理による移動」の位置，経路点，経路点，...，経路点**

ステータスコード

コマンドが正常に実行された場合、**2100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

ヒント： このコマンドを呼び出す時、Mech-Viz の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

送信が完了しているかどうか

- `0`：経路にあるすべての経路点が送信されいないため、このパラメータの値が 1 になるまでに、このコマンドを繰り返して実行してください。
- `1`：経路にあるすべての経路点が送信されました。

経路点の数

このパラメータは、返された経路点の数を表します。範囲は 0~20 です。初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は定点移動_1 -> 定点移動_2 -> ビジョン処理による移動 -> 定点移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

経路点

1 つの経路点は 8 つのデータで構成され、最初の 6 個は位置姿勢、7 番目はラベル、最後は速度を表します。

- **位置姿勢**：3 次元座標（ミリメートル単位）、オイラー角（単位は度）、または関節角度（単位は度）。位置姿勢の形式は、205 コマンドの位置姿勢のタイプによって決定されます。
- **ラベル**：位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。
- **速度**：移動ステップパラメータに設定されたゼロでないパーセンテージ値です。

実行例

コマンドが正常に実行された場合

```
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
```

エラーが発生した場合

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

コマンド 206——DO リストを取得

複数の治具または吸盤パーティションを制御する場合、このコマンドを用いて DO リストを取得します。

このコマンドを実行する前に、コマンド *205* を実行して Mech-Viz の計画経路を取得する必要があります。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定し、プロジェクトに対応する吸盤のコンフィグファイルを設定します。サンプルプロジェクトのパスは `Mech-Center\tool\viz_project\suction_zone` です。

プロジェクトの set_do_list のステップパラメータで以下の設定を行います。

- 「受信者」を「標準インターフェース」に設定します
- 「ビジョン処理による移動から DO リストを取得」にチェックを入れます
- パラメータバーの下部にあるエリアに DO リストが必要な「ビジョン処理による移動」ステップを選択します

送信コマンド

206

返されたデータ

206, **ステータスコード**, *DO* **信号値**, *DO* **信号値**, ..., *DO* **信号値**

ステータスコード

コマンドが正常に実行された場合、**2102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

DO 値

返されたデータの最後に 64 個の DO 値が受信され、すべては整数です。

DO 値の範囲は 0~999 です。

-1 はプレースホルダーとなります。

実行例

コマンドが正常に実行されると、11 と 12 の DO 信号値が得られます。

```
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
```

コマンドの実行中にエラーが発生した場合、DO 信号が得られませんでした。

```
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
```

コマンド 207——Mech-Viz ステップパラメータを読み取る

このコマンドは、指定されたステップの指定されたパラメータの値を読み取るために使用されます。

Mech-Center で **設定** ▸ *Mech-Interface* ▸ **詳細設定** ▸ **プロパティ構成ファイル** をクリックして、どのステップのどのパラメータを読み取るかを指定します。

送信コマンド

207, **構成** *ID*（*config ID*）

構成ファイルのコンテンツ

read, config ID, **ステップ** *ID,* **パラメータのキーの名前**

構成ファイルの「#」で始まる内容はコメント行であり、このコマンドを実行しても影響はありません。

read

文字列「read」は、コマンドがパラメータ値を読み取るために使用されることを示します。

config ID

読み取り操作 ID を指定する正の整数です。1 つの構成 ID を使用して、複数のパラメータを設定できます。複数のパラメータを読み取るには、異なる構成 ID を使用する必要があります。

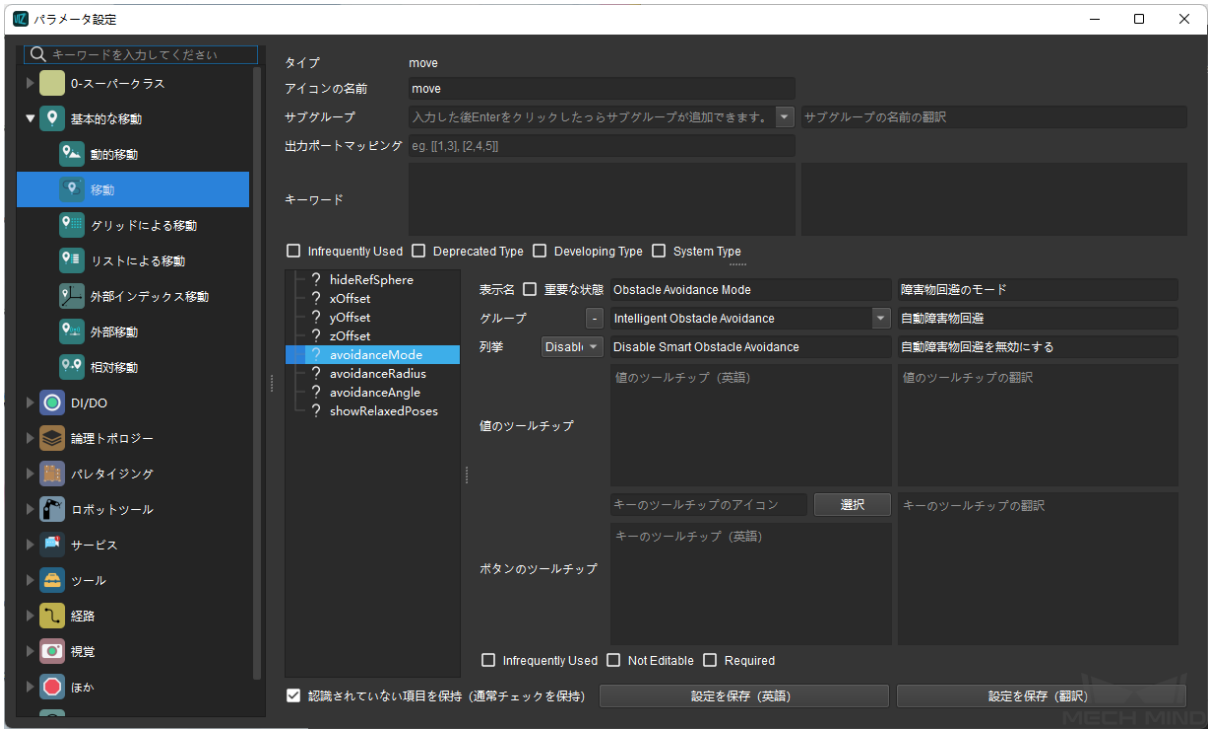
ステップ番号

読み取るパラメータのステップが属するステップのステップ ID です。

ステップパラメータでステップ ID を検索および設定できます。

パラメータのキーの名前

読み取るパラメータのキーの名前です。下図に示すように、Mech-Viz のメニューバーで **設定** ▸ **ステップ設定** をクリックしてその画面に入ります。



構成ファイルのコンテンツの例

```
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
```

返されたデータ

207, **ステータスコード**, **ステップパラメータ値**

ステータスコード

コマンドが正常に実行された場合、ステータスコード 2109 が返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

ステップパラメータ値

Mech-Viz プロジェクトに指定されたステップの指定されたパラメータの値です。

INT、FLOAT、STRING のデータ型に対応します。

実行例

コマンドが正常に実行されれば、パラメータ値が正常に読み込まれました。

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

コマンド実行例外であれば、対応するパラメータ名が見つかりませんでした。

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

コマンド 208 ——Mech-Viz のステップパラメータを設定

このコマンドは、指定されたステップの指定されたパラメータの値を設定するために使用されます。

Mech-Center で **設定** ▸ *Mech-Interface* ▸ **詳細設定** ▸ **プロパティ構成ファイル** をクリックして、どのステップのどのパラメータをどの値に設定するかを指定します。

送信コマンド

208, **構成** *ID*（*config ID*）

構成ファイルのコンテンツ

write, config ID, **ステップ** *ID,* **パラメータのキーの名前, 値**

write

文字列「write」は、コマンドがパラメータ値を設定するために使用されることを示します。

config ID

設定操作 ID です。1 つの構成 ID を使用して、複数のパラメータを設定できます。

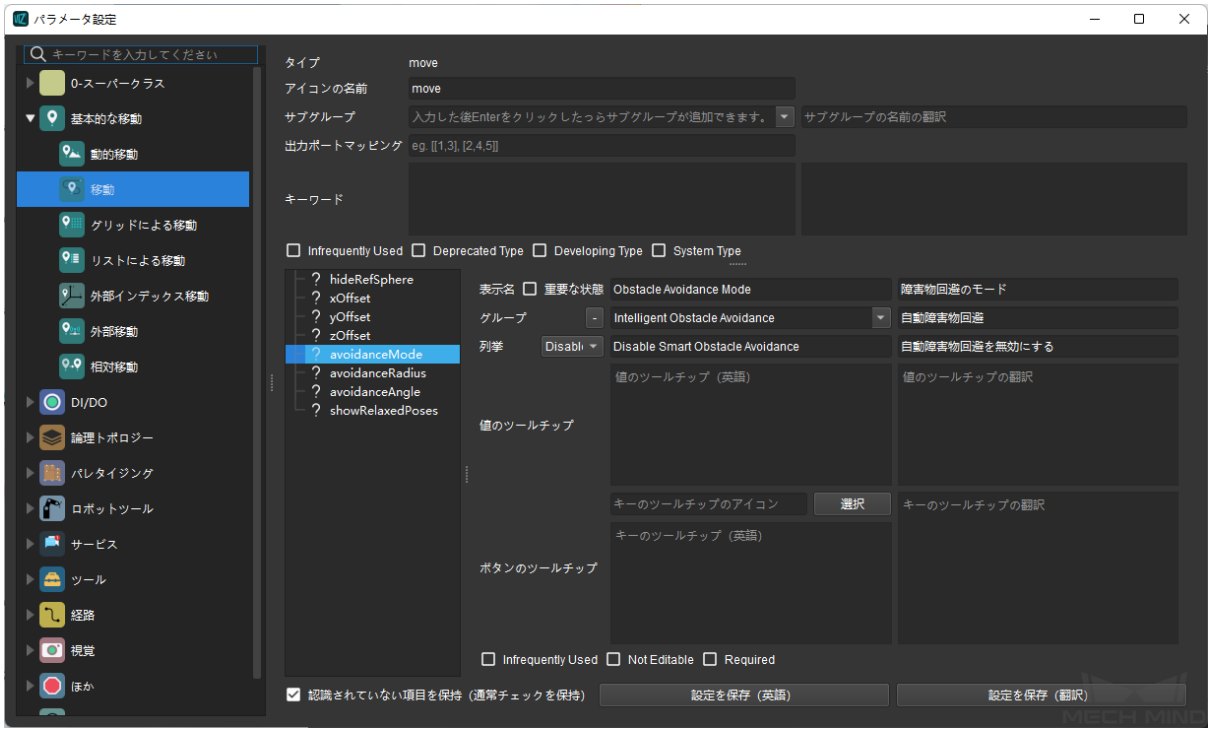
ステップ番号

読み取るパラメータが属するステップのステップ ID です。

ステップ ID はステップパラメータで検索および設定できます。

パラメータのキーの名前

設定するパラメータのキーの名前です。下図に示すように、Mech-Viz のメニューバーで **設定** ▸ **ステップのパラメータを設定** をクリックしてその画面に入ります。



値

パラメータに設定する値を指定します。

数値に加えて、文字列値もサポートされています。

構成ファイルのコンテンツの例

```
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
```

実行例

コマンドが正常に実行された場合、パラメータが正常に設定されます。

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

コマンド実行例外であれば、対応する番号が見つかりませんでした。

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

コマンド 210——単一経路点と視覚計画結果を取得

このコマンドは、Mech-Viz によって計画された単一の経路点を取得するために使用されます。経路点は、一般的な移動経路点または視覚移動経路点にすることができます。経路点には、位置姿勢、速度、ツール情報、ワーク情報などを含めることができます。

このコマンドを実行して得られる経路点は、次の 3 つのいずれかになります。

1. visual_move 以外の移動経路点に加えて、その情報には、運動タイプ（関節運動または直線運動）、ツール ID、および速度が含まれます。
2. visual_move の経路点です。その情報には、ラベル、把持されたワークの合計数、今回把持されたワークの数、吸盤のエッジコーナー番号、TCP オフセット、ワークの向き、およびワークグループのサイズが含まれます。
3. カスタマイズされたデータを含む、visual_move の経路点です。この場合、Mech-Vision プロジェクトのステップ procedure_out のポートタイプを「カスタム」に設定する必要があります。

注意:

- 通常、このコマンドは段ボール箱をワークとしたプロジェクトに使用されます。
- このコマンドを実行して一度に取得できる経路点は 1 つだけです。複数の経路点を取得したい場合は、このコマンドを繰り返し実行する必要があります。

送信コマンド

210, **返されたデータの予期形式**

返されたデータの予期形式

以下は、4 つの返されたデータの予期形式についての説明です。

返されたデータの予期形式 パラメータ	**返されたデータの予期形式** 説明
1	位置姿勢（JPs 形式）, 運動タイプ, ツール ID, 速度, カスタマイズされたデータの数, カスタマイズされたデータ 1, ..., カスタマイズされたデータ N
2	位置姿勢（TCP 形式）, 運動タイプ, ツール ID, 速度, カスタマイズされたデータの数, カスタマイズされたデータ 1, ..., カスタマイズされたデータ N
3	位置姿勢（JPs 形式）, 運動タイプ, ツール ID, 速度, ビジョン計画結果, カスタマイズされたデータの数, カスタマイズされたデータ 1, ..., カスタマイズされたデータ N
4	位置姿勢（TCP 形式）, 運動タイプ, ツール ID, 速度, ビジョン計画結果, カスタマイズされたデータの数, カスタマイズされたデータ 1, ..., カスタマイズされたデータ N

返されたデータ

210, **ステータスコード**, **経路点の送信が完了しているかどうか**, **経路点のタイプ**, **位置姿勢**, **運動タイプ**, **ツール** *ID,* **速度**, **ビジョン計画結果** **,* **カスタマイズされたデータの数**, **カスタマイズされたデータ** ***

> **注意：ビジョン計画結果** または **カスタマイズされたデータ** は、コマンドの実行時に送信される **予期されるデータ形式** によって異なります。

ステータスコード

コマンドが正常に実行された場合、ステータスコード 2100 が返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

経路点の送信が完了しているかどうか

- `0`：まだ送信されていない経路点があります。
- `1`：すべての経路点が送信されました。

経路点タイプ

- `0`：これは「ビジョン処理による移動」の経路点でないことを意味します。
- `1`：これは「ビジョン処理による移動」の経路点であるを意味します。

位置姿勢

経路点の位置姿勢は、ロボット関節角度（JPs、単位は度）、ツール位置姿勢（TCP、3 次元座標はミリメートル、オイラー角は度）のいずれかであり、その形式は送信するコマンドパラメータによって異なります。

運動タイプ

- `1`：関節運動（MOVEJ）
- `2`：直線運動（MOVEL）

ツール ID

経路点で使用するツール番号です。-1 は、ツールが使用されていないことを意味します。

速度

経路点での速度のパーセンテージ値（単位は%）、つまり、Mech-Viz プロジェクトでこの経路点に対応する移動ステップのパラメータに設定した速度に、Mech-Viz で設定したグローバル速度を掛けたものです。

視覚計画結果

経路内の計画結果の情報です（経路点が移動ステップ:ref:*visual_move* に対応する場合）。通常、段ボール箱の複数把持とデパレタイジングなどのシーンに使用されます。情報は次のとおりです。

- ラベル：10 個の正の整数で構成され、デフォルトは 10 個の 0 です。
- 把持されたワークの合計数。
- 今回把持されたワークの数。
- 吸盤エッジコーナー番号：ワークが吸盤のどのエッジに近いかを指定するために使用されます。吸盤のエッジコーナー番号は、Mech-Viz のプロジェクトリソースで対応するエンドツール名をダブルクリックし、**制御ロジック設定** をクリックすると表示されるようになります。
- ツール位置姿勢（TCP）のオフセット（ワークの中心に対応する TCP から実際の TCP までのオフセットです）。
- ワークの向き（TCP の X 軸に対するワーク座標系の X 軸の方向です）。
- ワークグループのサイズ。

カスタマイズされたデータ項目の数

これは、Mech-Vision プロジェクトの procedure_out ステップでポートタイプが「カスタム」に設定された場合に出力された poses と labels 以外のポートのデータの数です。

カスタマイズされたデータ項目

これは、Mech-Vision プロジェクトの procedure_out ステップでポートタイプが「カスタム」に設定された場合に出力された位置姿勢とラベル以外のデータです。

カスタマイズされたデータは、ポート名の A – Z の順に並べられています。

コマンド 501——Mech-Vision プロジェクトへ対象物の寸法を送信

このパラメータは Mech-Vision プロジェクトに、対象物の寸法を動的に送信すること場合に使われます。Mech-Vision プロジェクトを実行する前に対象物の寸法を確認する必要があります。

Mech-Vision プロジェクトに、read_object_dimensions ステップを入れておきます。このステップのパラメータ **パラメータから対象物の寸法を読み取る** にチェックを入れる必要があります。

送信コマンド

501, Mech-Vision **プロジェクト番号, 長さ, 幅, 高さ**

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

長さ，幅，高さ

Mech-Vision プロジェクトへ送信する対象物の寸法です。寸法の値は、read_object_dimensions ステップによって読み取られます。

単位はミリメートル（mm）です。

返されたデータ

501, **ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**1108** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

コマンド実行時にエラーが発生する場合、3002 のエラーコードが表示されます。これは「高さ」情報が欠落していることを意味します。

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

コマンド 502——Mech-Viz へ TCP を送信

このコマンドは、Mech-Viz プロジェクトにロボット TCP を動的に送信するためによく使われます。ロボット TCP を読み取るためのステップは outer_move です。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定します。サンプルプロジェクトのパスは `Mech-Center\tool\viz_project\outer_move` です。

outer_move をワークフローの適切な場所に配置します。

このコマンドは、**コマンド *201——Mech-Viz* プロジェクトを起動** を実行する前に実行してください。

送信コマンド

502, **ロボット** *TCP*

ロボット TCP

outer_move の目標点のロボット TCP データを設定するために使用されます。

返されたデータ

502，**ステータスコード**

ステータスコード

コマンドが正常に実行された場合、**2107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

実行例

コマンドが正常に実行された場合

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

コマンド 601——通知メッセージ

ユーザーはこのコマンドを実行する必要がありません。

Mech-Viz/Mech-Vision のプロジェクトは notify または notify_vision まで実行する場合、Mech-Center は「通知」にカスタマイズされたメッセージをクライアントに送信します。

Mech-Vision で notify_vision ステップの名前を「Standard Interface Notify」に変更します。

Mech-Viz で notify ステップパラメータ「受信者」を「標準インターフェース」に設定します。

送信コマンド

なし。

返されたデータ

601，**カスタマイズされた通知メッセージ**

カスタマイズされた通知メッセージ

notify または notify_vision でカスタマイズされた通知メッセージです。メッセージは整数である必要があります。

実行例

notify または notify_vision でカスタマイズされた通知メッセージは「1000」の場合、プロジェクトはこれらのステップまで実行すると、通知メッセージが送信されます。

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

コマンド 701——自動キャリブレーション

このコマンドは、ハンド・アイ・キャリブレーションを行う場合に使われます（カメラ外部パラメータのキャリブレーション）。

このコマンドを実行すると、Mech-Vision とキャリブレーションの状態を同期し、Mech-Vision からキャリブレーションポイントを取得します。

キャリブレーションを完成するために、このコマンドを繰り返して実行する必要があります。

送信コマンド

701，**キャリブレーション状態，フランジ位置姿勢，関節角度（*JPs*）**

キャリブレーション状態

パラメータ範囲は 0~2 です。

- 0：Mech-Vision と繋がり、キャリブレーションプロセスを開始します。
- 1：前回のキャリブレーションポイントが受信され、ロボットへ送信します。
- 2：前回のキャリブレーションポイントの受信が失敗しました。

フランジ位置姿勢

ロボットの現在のフランジ位置姿勢です。位置姿勢は位置と姿勢からなり、位置の単位はミリメートル（mm）です。姿勢はオイラー角で表し、単位は度（°）です。

関節角度（JPs）

ロボットの現在の関節角度（単位は度）です。

ヒント：フランジ位置姿勢 と **関節角度** は、どちらか一方のパラメータ値を指定するだけで、もう一方は 6 つのゼロで埋められます。

返されたデータ

701，**ステータスコード，キャリブレーション状態，次回キャリブレーションポイントのフランジ位置姿勢，次回キャリブレーションポイントの関節角度（*JPs*）**

ステータスコード

コマンドが正常に実行された場合、**7101** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

キャリブレーション状態

- 0：キャリブレーション進行中。
- 1：キャリブレーション終了。

次回キャリブレーションポイントのフランジ位置姿勢

移動先のキャリブレーションポイント位置姿勢データです。ロボットのプログラムから、フランジ位置姿勢や JPs 位置姿勢は選択可能です。

次回キャリブレーションポイントの関節角度

移動先のロボットの JPs データです。

ヒント： 関節角度（JPs）とフランジ位置姿勢のどちらが選べばよいです。

実行例

キャリブレーションプロセスを開始

```
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
```

Mech-Vision からキャリブレーションポイントを取得（すべてのキャリブレーションポイントを取得するには、このプロセスを数回繰り返す必要があります）

```
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
```

ヒント： このプロセスを繰り返して複数のキャリブレーションポイントを取得する必要があります。

キャリブレーションを終了

```
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
```

コマンド 901——ソフトウェアの起動状態を取得

このコマンドは、Mech-Vision、Mech-Viz、Mech-Center の起動状態を取得するために使用されます。

現在、このコマンドは Mech-Vision のみ対応できます。

送信コマンド

901

返されたデータ

901, **ステータスコード**

ステータスコード

システムのセルフチェックステータスです。ステータスコードが **1101** になると「Mech-Vision が準備できました」です。他のステータスコードが出てきた場合「Mech-Vision プロジェクトがまだ準備できていません」ということです。現在、このコマンドは Mech-Vision プロジェクトの準備ができているかどうかを確認するためにのみ使用できます。

実行例

Mech-Vision プロジェクトが準備できました

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

Mech-Vision プロジェクトがまだ準備できていません

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

ヒント: このコマンドを実行する前に、ソリューションの自動読み込み または プロジェクトの自動読み込み に設定されていることを確認してください。

3.3.3 Siemens PLC コマンド

本節では、Siemens PLC Snap 7 通信プロトコルに基づいた標準インターフェースのコマンドについて説明します。

コマンド 101——Mech-Vision プロジェクトを起動

このコマンドは、Mech-Vision のプロジェクトを起動し、カメラの撮影および視覚認識を行う場合に使われます。

プロジェクトは Eye In Hand モードである場合、このコマンドを用いて、ロボット撮影の位置姿勢をプロジェクトへ送信します。

このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使われます。

送信コマンド

パラメータ	DB オフセット
コマンド 101	2.0
Mech-Vision プロジェクト番号	8.0
ビジョン位置姿勢の期待数	6.0
ロボット位置姿勢のタイプ	4.0
ロボット位置姿勢	12.0（JPs）または 36.0（フランジ位置姿勢）

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

ビジョン位置姿勢の期待数

Mech-Vision から取得したいビジョン位置姿勢の数です。ビジョン位置姿勢情報に、ビジョン位置姿勢及びそれに対応する点群、ラベル、スケーリングの情報が含まれています。

- 0：Mech-Vision プロジェクトで認識できたすべてのビジョン結果を取得します。
- 0 より大きな整数：Mech-Vision プロジェクトで認識できた指定数のビジョン結果を取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョン位置姿勢の合計数より大きい場合、認識結果にあるすべてのビジョン位置姿勢を取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョン位置姿勢の合計数より小さい場合、このパラメータで指定された数のビジョン位置姿勢を取得します。

ヒント： 102 コマンドでビジョン結果を取得できます。デフォルトでは、102 コマンドの 1 回実行で最大 20 点のビジョン結果を取得することができます。最初の実行後、返されるパラメータの 1 つは、リクエストされたすべてのビジョン結果が返されたかどうかを示します。そうでない場合は、102 コマンドを繰り返してください。

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Vision に送信するタイプを設定します。パラメータ範囲は 0~3 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	ロボット位置姿勢 パラメータ	設定の説明	適用シーン
0	0, 0, 0, 0, 0, 0	Mech-Vision にロボットの位置姿勢を送信する必要がありません	プロジェクトは、Eye To Hand モードです。Mech-Vision プロジェクトで「経路計画」ステップを使用する場合、経路計画の開始位置は、経路計画ツールで設定した原点です。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Vision にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があります	プロジェクトは、Eye In Hand モードです。この設定は、直行ロボット以外のほとんどのロボットで利用可能です。
2	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在のフランジ位置姿勢を送信する必要があります	プロジェクトは、Eye In Hand モードです。ロボットは関節角度のデータを持たず、フランジい位置姿勢データのみを持ちます（直行ロボットの場合など）。
3	ロボット経路計画の開始位置の関節角度	Mech-Vision にロボット経路計画の開始位置の関節角度を送信する必要があります	プロジェクトは、Eye To Hand モードです。また、Mech-Vision プロジェクトに「経路計画」ステップがあり、ロボット側から「経路計画」ステップの開始位置を設定する必要があります。

ロボットの位置姿勢は Real タイプの数字です。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**1102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 102——Mech-Vision のビジョン目標点を取得

コマンド *101——Mech-Vision* プロジェクトを起動 の後に使用します。このコマンドは、Mech-Vision からビジョン位置姿勢を取得してビジョン目標点に変換するために使用されます。

以下に、ビジョン位置姿勢に含まれる位置姿勢をロボット TCP に変換する処理を示します。

- ビジョン位置姿勢に含まれる位置姿勢を Y 軸を中心に 180° 回転させます。
- 対応するロボット型番の基準座標系定義にロボットベースの高さが含まれているかどうかを認識し、それに応じて垂直方向のオフセットを増やします。

ヒント： デフォルトでは、102 コマンドは毎回最大 20 個までのビジョン目標点を取得することができます。20 個以上のビジョン目標点を取得するには、すべてのビジョン目標点を得るまで、102 コマンドを繰り返

し実行してください。

送信コマンド

パラメータ	DB オフセット
コマンド 102	2.0
Mech-Vision プロジェクト番号	8.0

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0
データ転送状態	202.0
ビジョン目標点の数	204.0
予約語	/
今回取得したすべての TCP	208.0
今回取得したすべてのラベル	1168.0

ステータスコード

コマンドが正常に実行された場合、**1100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しいビジョン目標点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョン目標点であることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

ビジョン目標点の数

このコマンドを実行して、取得したビジョン目標点の数です。

- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョン位置姿勢の数よりも多い場合、Mech-Vision によって認識されたビジョン位置姿勢の数に従って送信されます。
- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョン位置姿勢の数よりも少ない場合、リクエストした数に従って送信されます。

予約語

この予約語が使われていないため、初期値は 0 です。

今回取得したすべての TCP

1 つの TCP には、空間座標 (XYZ) とオイラー角 (ABC) の情報が含まれます。

今回取得したすべてのラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

コマンド 103——Mech-Vision のパラメータレシピを切り替える

Mech-Vision プロジェクトのパラメータレシピを切り替えます。

パラメータレシピを切り替えることで、Mech-Vision プロジェクトの各ステップのパラメータを変更することができます。

パラメータレシピには画像マッチングテンプレート、ROI、信頼度のしきい値などのパラメータの設定が含まれています。

注意： コマンド *101——Mech-Vision* プロジェクトを起動 を実行する前に、このコマンドを使用する必要があります。

送信コマンド

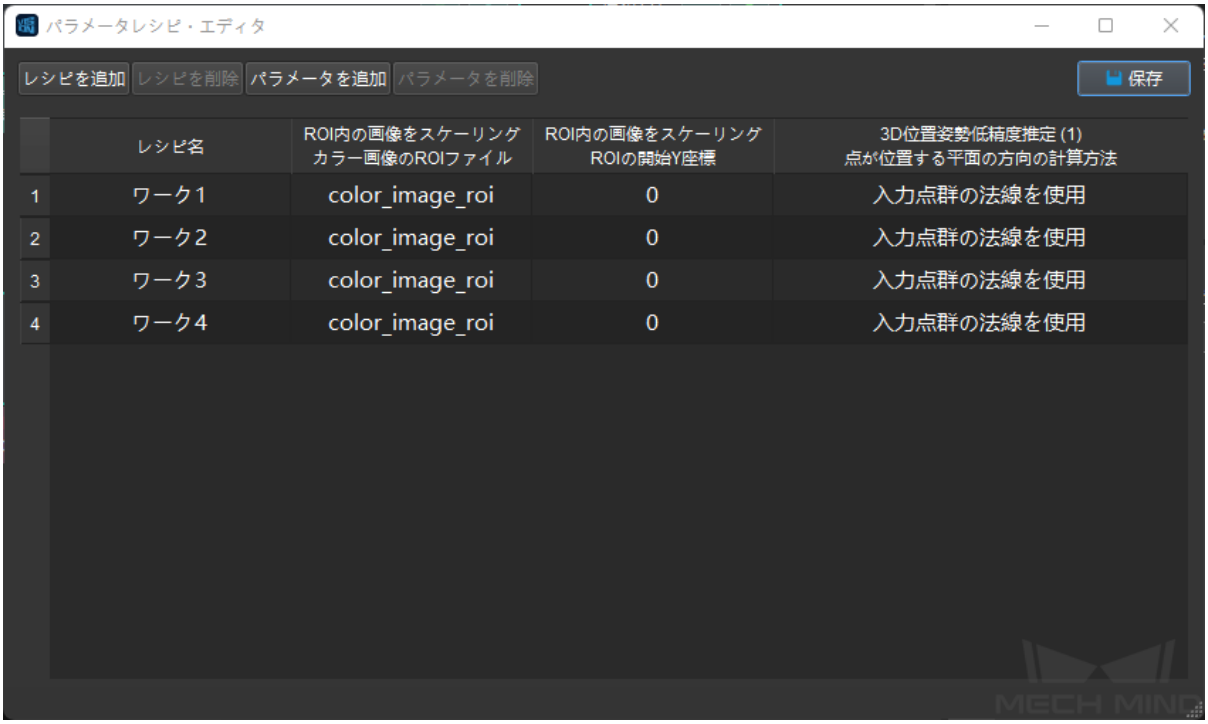
パラメータ	DB オフセット
コマンド 103	2.0
Mech-Vision プロジェクト番号	8.0
レシピ番号	10.0

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

レシピ番号

Mech-Vision プロジェクトのレシピテンプレートの番号（正の整数）です。**プロジェクトアシスタント ‣ パラメータレシピ** をクリックして、パラメータレシピエディタに入ります。番号の有効範囲は 1~99 です。



返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**1107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 105——Mech-Vision の「経路計画」ステップの結果を取得

101 コマンドを呼び出した後、このコマンドを使用して Mech-Vision の「経路計画」ステップから出力された衝突ないの把持経路を取得します。

このコマンドを使用する時、Mech-Vision の「出力」ステップの **ポートタイプ** を「事前定義済み（ロボット経路）」に設定する必要があります。

ヒント： 105 コマンドを呼び出す前に、105 コマンドの呼び出し回数を減らすように 101 コマンドの **ビジョン位置姿勢の期待数** を 0 に設定する必要があります。101 コマンドの **ビジョン位置姿勢の期待数** を 1 に設定すると、105 コマンドの呼び出しごとに 1 つの経路点のみが返され、105 コマンドを複数回呼び出した場合にのみすべての経路点が返されます。

送信コマンド

パラメータ	DB オフセット
コマンド 105	2.0
Mech-Vision プロジェクト番号	8.0
経路点の位置姿勢タイプ	4.0

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

経路点の位置姿勢タイプ

このパラメータは、「経路計画」ステップから返された経路点の位置姿勢タイプを指定するために使用されます。

- `1`：経路点の位置姿勢は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点の位置姿勢は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0
データ転送状態	202.0
経路点の数	204.0
「ビジョン処理による移動」の位置	206.0
今回送信したすべての経路点の位置姿勢	208.0
今回送信したすべての経路点のラベル	1168.0
今回送信したすべての経路点の速度	1248.0

ステータスコード

コマンドが正常に実行された場合、**1103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

`1`：返されたデータが新しい経路点であることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、このコマンドを実行した後に返された経路点の数を表します。範囲は 0~20 です。20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

経路計画ツールで設定された「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 105 送信時の **経路点の位置姿勢タイプ** によって、三次元座標、XYZ オイラー角、あるいは関節角度（JPs）で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に ラベルマッピング ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

経路計画設定ツールで設定された速度値です。

コマンド 110 —— Mech-Vision からカスタマイズされたデータを取得

このコマンドは、Mech-Vision の procedure_out ステップでカスタマイズされたデータ（poses と labels 以外のポート）を受信するために使用されます（ procedure_out ステップの「ポートタイプ」を「カスタム」に設定する必要があります）。

送信コマンド

パラメータ	DB オフセット
コマンド 110	2.0
Mech-Vision プロジェクト番号	8.0

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0
データ転送状態	202.0
今回取得したすべての TCP	208.0
今回取得したすべてのラベル	1168.0
カスタマイズされたポートデータ	1456.0

ステータスコード

コマンドが正常に実行された場合、ステータスコード 1100 が返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい TCP であるかどうかを表示するために使用されます。

1：返されたデータが新しい TCP であることを示し、それを読み取ります。

今回取得したすべての TCP

1 つの TCP には、空間座標 (XYZ) とオイラー角 (ABC) の情報が含まれます。

今回取得したすべてのラベル

位置姿勢に対応する対象物情報ラベルです。ラベルは正の整数である必要があります。それ以外の場合は、Mech-Vision プロジェクトの label_mapping ステップを使用して正の整数にマッピングする必要があります。procedure_out ステップにラベルポートがない場合、このフィールドは 0 で埋められます。

カスタマイズされたデータ項目

ステップ procedure_out でカスタマイズされたデータタイプが対応するポートのカスタマイズされたデータです。

カスタマイズされたデータパラメータは、ポート名の A – Z の順に並べられています。

コマンド 201——Mech-Viz プロジェクトを起動

このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使われます。Mech-Viz プロジェクトを開始し、対応する Mech-Vision プロジェクトを呼び出し、Mech-Viz が Mech-Vision のビジョン結果に基づいて経路を計画する時に使用されます。

Mech-Viz では、**自動的に読み込む** にチェックを入れる必要があります。

Mech-Center のインストールディレクトリ（tool¥viz_project）フォルダには、サンプルプロジェクトがあり、それら基づいて修正することが可能です。

標準インターフェースに使用される Mech-Viz サンプルプロジェクトの詳細な説明については、**標準インターフェースに使用される *Mech-Viz* サンプルプロジェクト** をご参照ください。

送信コマンド

パラメータ	DB オフセット
コマンド 201	2.0
ロボット位置姿勢のタイプ	4.0
ロボット位置姿勢	12.0（JPs）または 36.0（フランジ位置姿勢）

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Viz に送信するタイプを設定します。パラメータ範囲は 0~2 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**設定の説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Viz にロボットの位置姿勢を送信する必要がありません。Mech-Viz での仮想ロボットは初期位置姿勢（JPs = [0, 0, 0, 0, 0, 0] ）から最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨しません。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Viz にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があり、Mech-Viz での仮想ロボットは受信された位置姿勢から最初の経路点に移動します。	プロジェクトは Eye In Hand モードである場合は、この設定は推奨します。
2	ロボット側でカスタマイズされた関節角度	Mech-Viz にロボットのティーチポイント（現在の関節角度ではない）を送信する必要があります。これは、ロボットが画像撮影領域の外にいるとき（下図に示す）、Mech-Viz プロジェクトが次回の経路を事前に計画することをトリガーするために使用されます。Mech-Viz での仮想ロボットは受信された最初のティーチポイントから最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨します。

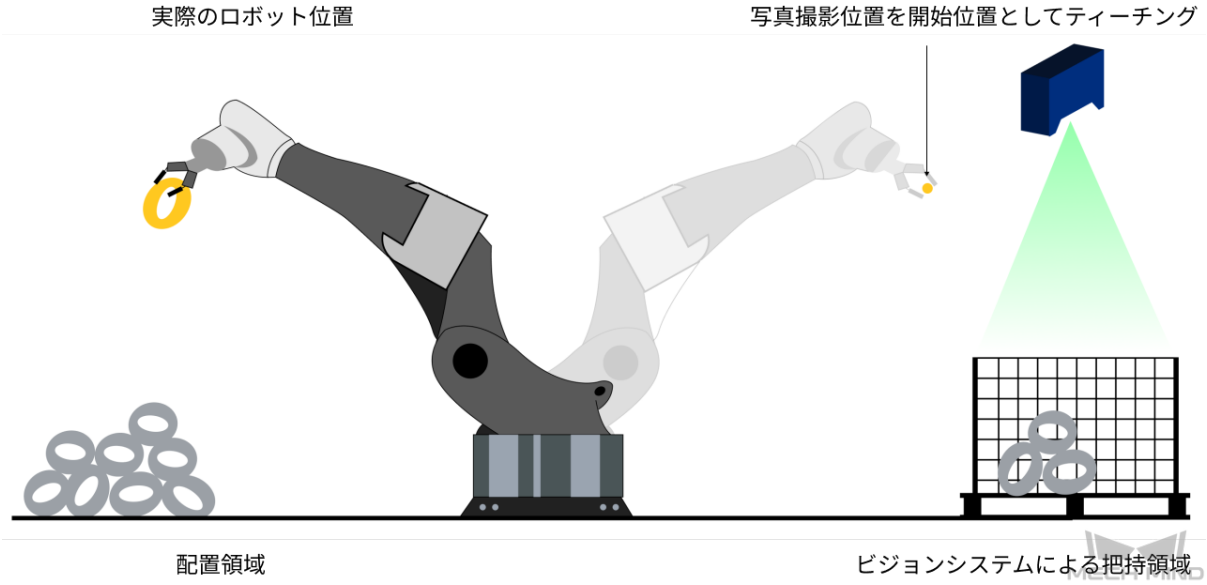
Eye To Hand モードでは、**ロボット位置姿勢のタイプ** が 2 に設定されている理由は以下の通りです。

Eye To Hand モードでは、カメラはロボットが画像撮影領域と把持領域に戻る前に撮影し、次回の把持経路を計画することができます。これにより、タクトタイムの向上が可能です。

この時、**ロボット位置姿勢のタイプ** が 1 に設定され、つまり現在の位置姿勢を Mech-Viz に送信すれば、仮想ロボットがロボット実機の経路と一致しない可能性があります。また、未知の衝突が発生する可能性もあります。

仮想ロボットは現在の位置姿勢から Mech-Viz での最初の移動ステップで設定された位置姿勢に移動しますが、ロボット実機は上記の位置姿勢に移動する前に別の位置姿勢に移動する可能性があるということです。

したがって、**ロボット位置姿勢のタイプ** パラメータを 2 に設定する必要があります。



返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**2103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 202—— Mech-Viz プロジェクトを停止

Mech-Viz プロジェクトの実行を停止します。Mech-Viz プロジェクトは無限ループになっていない場合や、正常に停止できる場合は、このコマンドを使用する必要がありません。

送信コマンド

パラメータ	DB オフセット
コマンド 202	2.0

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**2104** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 203——Mech-Viz 分岐を選択

このコマンドは、Mech-Viz プロジェクトの分岐を指定する場合に使われます。分岐メカニズムは branch_by_msg ステップによって作成されたら、このコマンドがステップの出口を指定することで実現します。

このコマンドを実行する前に、**コマンド *201——Mech-Viz* プロジェクトを起動** を実行してください。

Mech-Viz プロジェクトが branch_by_msg ステップに実行すると、このコマンドによって出口を指定するのを待ちます。

送信コマンド

パラメータ	DB オフセット
コマンド 203	2.0
分岐ステップ ID	60.0
分岐の出口番号	62.0

分岐ステップ ID

このパラメータは、分岐選択が行われる branch_by_msg ステップを指定するために使用されます。

このパラメータ、つまり、branch_by_msg のステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

出口番号

このパラメータは、プロジェクトが branch_by_msg ステップに沿って実行される出口を指定します。Mech-Viz プロジェクトはこの出口に従って実行し続けます。パラメータの値は正の整数です。

出口番号の範囲は 1~99 です。

ヒント:

- 出口番号は、Mech-Viz で表示されるポート番号に 1 を加えたもので、ポート 0 が出口 1 です。
- 出口番号は、1 から始まるポートのインデックス番号です。たとえば、指定された出口が左から右に 2 番目のポートである場合、番号は 2 です。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

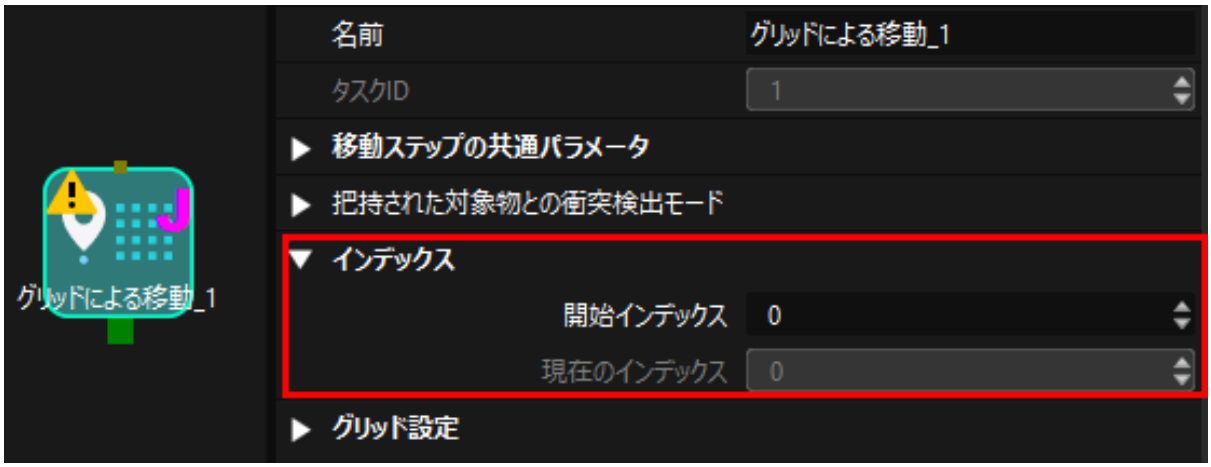
コマンドが正常に実行された場合、**2105** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 204——移動インデックスを設定

このコマンドはステップのインデックスパラメータを設定する時に使われます。一般的には、連続な移動か指定された移動ステップやパレタイジングなどに使用されます。

インデックスパラメータが付いたステップは「リストによる移動」、「グリッドによる移動」、「カスタマイズのパレットパターン」、「事前計画したパレットパターン」などです。

このコマンドを実行する前に、コマンド *201——Mech-Viz* プロジェクトを起動 を実行してください。



送信コマンド

パラメータ	DB オフセット
コマンド 204	2.0
ステップ ID	64.0
インデックス☒	66.0

ステップ ID

このパラメータはどのステップがインデックスを設定する必要かを指定します。

このパラメータ、つまり、インデックス付きのステップのステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

インデックス☒

次にこのステップが実行されたときに設定されるべきインデックス値です。

このコマンドを送信すると、Mech-Viz の現在のインデックス値がこのパラメータの値から 1 を引いた値に変更されます。

このコマンドで指定したステップに Mech-Viz プロジェクトが実行されると、Mech-Viz の現在のインデックス値が、このパラメータの値まで 1 つずつ増加します。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**2106** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 205——Mech-Viz の計画経路を取得

ref:*plc_start_mech_viz_project* を実行した後、このコマンドは Mech-Viz が計画した経路を取得するために使用されます。

初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

注釈: プロジェクト内の移動ステップの経路点をロボットに送信しない場合は、ステップパラメータで「移動目標点を送信」のチェックを外してください。

送信コマンド

パラメータ	DB オフセット
コマンド 205	2.0
経路点タイプ	4.0

経路点タイプ

このパラメータは Mech-Viz からどのような形式で経路点を返すかを指定します。

- `1`：経路点は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0
データ転送状態	202.0
経路点の数	204.0
「ビジョン処理による移動」の位置	206.0
今回送信したすべての経路点の位置姿勢	208.0
今回送信したすべての経路点のラベル	1168.0
今回送信したすべての経路点の速度	1248.0

ステータスコード

コマンドが正常に実行された場合、**2100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

ヒント: このコマンドを呼び出す時、Mech-Viz の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョン位置姿勢であることを示し、それを読み取ります。

注意: 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、返された経路点の数を表します。範囲は 0~20 です。初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。従って、20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は定点移動_1 -> 定点移動_2 -> ビジョン処理による移動 -> 定点移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 205 発送時のパラメータによって、三次元座標、オイラー角、あるいは関節角度で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

移動ステップパラメータに設定されたゼロでないパーセンテージ値です。

コマンド 206——DO リストを取得

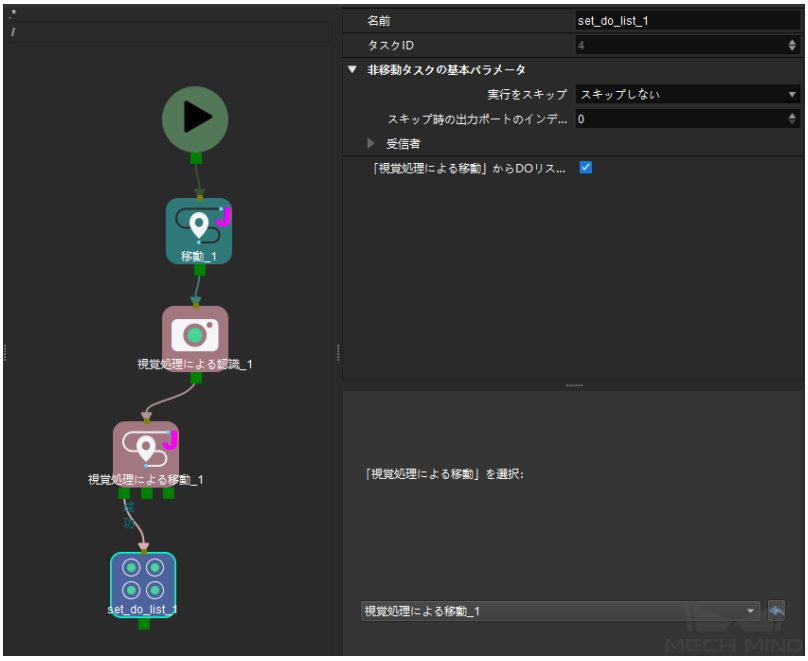
複数の治具または吸盤パーティションを制御する場合、このコマンドを用いて DO リストを取得します。

このコマンドを実行する前に、コマンド *205* を実行して Mech-Viz の計画経路を取得する必要があります。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定し、プロジェクトに対応する吸盤のコンフィグファイルを設定します。サンプルプロジェクトのパスは Mech-Center のインストールディレクトリ (tool¥viz_project) の **suction_zone** です。

プロジェクトの set_do_list ステップのパラメータで以下の設定を行います。

- 「受信者」を「標準インターフェース」に設定します
- 「ビジョン処理による移動から DO リストを取得」にチェックを入れます
- パラメータバーの下部にあるエリアに DO リストが必要な「ビジョン処理による移動」ステップを選択します



送信コマンド

パラメータ	DB オフセット
コマンド 206	2.0

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0
DO 信号リスト	1408.0

ステータスコード

コマンドが正常に実行された場合、**2102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

DO 値

64 個の DO 値が受信され、すべては整数です。

DO 値の範囲は 0~999 です。

-1 はプレースホルダーとなります。

例えば、DO 信号値は 1、3、5、6 です。

1	3	5	6	-1	-1	-1	-1	...	-1	-1
1 番目の整数	2 番目の整数	3 番目の整数	4 番目の整数	5 番目の整数	6 番目の整数	7 番目の整数	8 番目の整数	...	63 番目の整数	64 番目の整数

コマンド 501——Mech-Vision プロジェクトへ対象物の寸法を送信

このパラメータは Mech-Vision プロジェクトに、対象物の寸法を動的に送信すること場合に使われます。Mech-Vision プロジェクトを実行する前に対象物の寸法を確認する必要があります。

Mech-Vision プロジェクトに、read_object_dimensions ステップを入れておきます。このステップのパラメータ **パラメータから対象物の寸法を読み取る** にチェックを入れる必要があります。

送信コマンド

パラメータ	DB オフセット
コマンド 501	2.0
Mech-Vision プロジェクト番号	8.0
[長さ、幅、高さ]	68.0

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

長さ、幅、高さ

Mech-Vision プロジェクトへ送信する対象物の寸法です。寸法の値は、read_object_dimensions ステップによって読み取られます。

単位はミリメートル（mm）です。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**1108** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 502——Mech-Viz へ TCP を送信

このコマンドは、Mech-Viz プロジェクトにロボット TCP を動的に送信するためによく使われます。ロボット TCP を読み取るためのステップは outer_move です。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定します。サンプルプロジェクトのパスは Mech-Center のインストールディレクトリ `tool¥viz_project¥outer_move` です。

outer_move ステップをワークフローの適切な場所に配置します。

このコマンドは、**コマンド *201*——*Mech-Viz* プロジェクトを起動** を実行する前に実行してください。

送信コマンド

パラメータ	DB オフセット
コマンド 502	2.0
TCP	80.0

TCP

outer_move ステップの経路点の TCP データを設定するために使用されます。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

コマンドが正常に実行された場合、**2107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 901——ソフトウェアの起動状態を取得

このコマンドは、Mech-Vision、Mech-Viz、Mech-Center の起動状態を取得するために使用されます。

現在、このコマンドは Mech-Vision のみ対応できます。

送信コマンド

パラメータ	DB オフセット
コマンド 901	2.0

パラメータの説明：パラメータはありません。

返されたデータ

パラメータ	DB オフセット
ステータスコード	200.0

ステータスコード

システムのセルフチェックステータスです。ステータスコードが **1101** になると「Mech-Vision が準備できました」です。他のステータスコードが出てきた場合「Mech-Vision プロジェクトがまだ準備できていません」ということです。現在、このコマンドは Mech-Vision プロジェクトの準備ができているかどうかを確認するためにのみ使用できます。

3.3.4 PROFINET

PROFINET プロトコルに基づいた標準インターフェースを介して、Mech-Mind ソフトウェアシステムは、TIA Portal ソフトウェアを用いて、Siemens SIMATIC S7 PLC と通信することができます。

詳しい内容について、PROFINET - Siemens SIMATIC S7 PLC をご参照ください。

通信プロトコル

- *Mech-Center から PLC へ*
  - *Control_Output*
  - *Status_Code*
  - *Calib_Cam_Status*
  - *Send_Pose_Num*
  - *Visual_Point_Index*
  - *DO_List*
  - *Notify_Message*
  - *Send_Pose_Type*
  - *Target_Pose*
  - *Target_Label*
  - *Target_Speed*
  - *Ext_Output_Data*
- *PLC から Mech-Center へ*
  - *Control_Input*
  - *Command*
  - *Calib_Rob_Status*
  - *Vision_Proj_Num*
  - *Vision_Recipe_Num*
  - *Viz_Task_Name*
  - *Viz_Task_Value*

- *Req_Pose_Num*
- *Robot_Pose_Type*
- *Req_Pose_Type*
- *Robot_Pose_JPS*
- *Robot_Pose_TCP*
- *Ext_Input_Data*

Mech-Center から PLC へ

Control_Output

Bit	データ
7	/
6	/
5	/
4	コマンドの実行が完了した（ブール値）
3	データの更新が完了した（ブール値）
2	カメラの露出が完了した（ブール値）
1	システムの起動が成功した（ブール値）
0	ハートビート（ブール値）

Command_Complete

コマンドの実行が完了してから、ユーザーは返されたステータスコードや他のデータを読むことができるように、この信号が使われます。102 および 205 コマンドでは、送信された最後の位置姿勢データの受信が完了した後、この信号の値が 1 になります。

Data_Ready

この信号は、位置姿勢データが読めるように使用されます。102 または 205 コマンドが複数のロボットの位置姿勢データを受信する場合のみ使用されます。

Exposure_Complete

この信号の値はカメラの露出が完了すると 1 に設定されます。撮影された対象物または EIH ロボットが撮影した位置から移動可能という指示を下します。

Trigger_Acknowledge

Trigger_Acknowledge の値は 1 の場合、ビジョンシステムがトリガ信号によって起動されたことを意味します。Trigger_Acknowledge の値は、トリガ信号がリセットするまで、そのまま維持します。

Heartbeat

システムのハートビート、1 秒に 1 回反転します。

Status_Code

ステータスコード、INT32。

ビジョンシステムから返される実行ステータスコードであり、正常な状態に対応するステータスコードまたはエラーに対応するエラーコードが返されます。

Calib_Cam_Status

キャリブレーションのステータス、INT8。

キャリブレーション 701 コマンド専用の信号です。0：キャリブレーション実行中 。1：キャリブレーション終了。

Send_Pose_Num

送信される位置姿勢の数、INT8。

Visual_Point_Index

プロジェクトにおけるステップ「ビジョン処理による移動」の位置です。「ビジョン処理による移動」は、ビジョンポイントへ移動する移動ステップです。

例えば、定点移動_1 -> 定点移動_2 -> ビジョン処理による移動 -> 定点移動_3 のロジックにおいて、ステップ「ビジョン処理による移動」の位置は 3 です。

ステップ「ビジョン処理による移動」がなければ、このパラメータは 0 です。

データ型：INT8

DO_List

複数の吸盤パーティションまたは配列 グリッパーを制御するための 64 INT8 DO 信号です。

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

Notify_Message

Mech-Viz/Mech-Vision の通知ステップによって送信されるカスタマイズの整数メッセージです。

整数のメッセージ、INT32。

Send_Pose_Type

送信した位置姿勢のタイプ、INT8。

- 1：JPs 関節角度のタイプです。
- 2：ツールセンターポイント（TCP）のタイプです。

Target_Pose

ロボット TCP または JPs のタイプのロボット位置姿勢です。

> **注意：** このモジュールから読み取ったデータは、使用前に 10000 で除算する必要があります。

三次元座標とオイラー角で表示された位置姿勢データ構造は次のとおりです。

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

ロボットの関節角度 JPs で表現される位置姿勢には、最大 6 つの関節角度が含まれます。

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

Byte	Bit 0~7
0~3	目標点の X 座標または J1 関節角度、INT32
4~7	目標点の Y 座標または J2 関節角度、INT32
8~11	目標点の Z 座標または J3 関節角度、INT32
12~15	目標点の A 角度または J4 関節角度、INT32
16~19	目標点の B 角度または J5 関節角度、INT32
20~23	目標点の C 角度または J6 関節角度、INT32

Target_Label

送信された位置姿勢に対応するラベルです。値は非負の整数である。

データ型：INT32

Target_Speed

目標点に対応する移動ステップの速度パラメータのパーセンテージ値で、範囲は 0〜100 です。

データ型：INT32

Ext_Output_Data

予約モジュールで、他のデータを転送するために使用されます。

このモジュールは 40 バイト (INT32[1:10]、合計 10 個の INT32 整数) を占有します。

PLC から Mech-Center へ

Control_Input

Bit	データ
7	/
6	/
5	/
4	メッセージの通知をリセットする（ブール値）
3	データ確認（ブール値）
2	「露出完了」をリセットする（ブール値）
1	トリガー信号（ブール値）
0	通信を有効にする（ブール値）

Reset_Exposure

「露出完了」をリセットする（ブール値）

Reset_Exposure = 1 の場合は、Exposure Complete は 0 に設定されます。

Data_Acknowledge

データ確認（ブール値）はコマンド 102 または 205 が返されたデータを確認するために使用されます。

Data_Acknowledge = 0 は、PLC が Mech-Center からデータを取得していないことを示し、データはポートに保持されます。

Data_Acknowledge = 1 は、PLC が Mech-Center からデータを取得したことを示し、Mech-Center は次のサイクルのデータに書き込むことができます。

Data_Acknowledge は、Heartbeat 反転または Data_Ready = 0 の場合にリセットすることができます。

Reset_Notify

メッセージの通知をリセットする（ブール値）

Reset_Notify = 1 の場合は Notify_Message のコンテンツはクリアされます。

Trigger

トリガー信号（ブール値）

Trigger = 1 の場合、Mech-Center は送信されたコマンドを読み取り、実行します。

Mech-Center がトリガー信号を受信すると、Trigger_Acknowledge をリセットすることができます。

信号のアップストリームは 1 と見なされます。

Comm_Enable

通信を有効にする（ブール値）

`0`：通信を無効にします。Mech-Center はトリガー信号を無視します。

`1`：通信を有効にします。トリガー信号が機能し、Mech-Center はコマンドを受信します。

Command

コマンドコード、INT32。

Calib_Rob_Status

- `0`：キャリブレーションが開始します。
- `1`：ロボットは、送信された最新のキャリブレーションポイントに正常に移動しました。
- `2`：ロボットは、送信された最新のキャリブレーションポイントに移動できませんでした。

データ型：INT8

Vision_Proj_Num

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

データ型：INT8

Vision_Recipe_Num

Mech-Vision プロジェクトのレシピテンプレートの番号（正の整数）です。**プロジェクトアシスタント ‣ パラメータレシピ** をクリックして、パラメータレシピエディタに入ります。番号の有効範囲は 1~99 です。

データ型：INT8

Viz_Task_Name

コマンド関連の Mech-Viz ステップのステップ ID です。ステップのパラメータで読み取りと設定を行うことができます。

データ型：INT8

Viz_Task_Value

Mech-Viz のブランチステップ出口番号、または Mech-Viz のステップインデックスの設定値です。

データ型：INT8

Req_Pose_Num

Mech-Vision からリクエストした視覚位置姿勢の数です。

`0`：Mech-Vision の視覚位置姿勢からすべての利用可能な視覚位置姿勢をリクエストします。

データ型：INT8

Robot_Pose_Type

ロボット位置姿勢のタイプです。

データ型：INT8

Req_Pose_Type

Mech-Viz から返された位置姿勢の予期形式です。

- `1`：JPs タイプ。
- `2`：TCP タイプ。

データ型：INT8

Robot_Pose_JPS

画像撮影時のロボットの関節角度 JPs です。

モジュールを設定する前に JPs を 10000 倍にすることが必要です。

JPs には最大 6 つの関節角度データ（6 つの INT32 整数）が含まれます。

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

Byte	Bit 0~7
0~3	ロボット現時点での J1 関節角度 INT32
4~7	ロボット現時点での J2 関節角度 INT32
8~11	ロボット現時点での J3 関節角度 INT32
12~15	ロボット現時点での J4 関節角度 INT32
16~19	ロボット現時点での J5 関節角度 INT32
20~23	ロボット現時点での J6 関節角度 INT32

Robot_Pose_TCP

画像撮影ためのロボットフランジ位置姿勢です。

このモジュールを設定する前に位置姿勢データを 10000 倍にすることが必要です。

フランジには、3 次元座標（X、Y、Z）およびオイラー角（A、B、C）が含まれ、6 つの INT32 整数で構成されます。

Byte	Bit 0~7
0~3	ロボット現時点での X 座標 INT32
4~7	ロボット現時点での Y 座標 INT32
8~11	ロボット現時点での Z 座標 INT32
12~15	ロボット現時点での A 角度 INT32
16~19	ロボット現時点での B 座標 INT32
20~23	ロボット現時点での C 角度 INT32

Ext_Input_Data

予約モジュールで、他のデータを転送するために使用されます。

このモジュールは 40 バイト (INT32[1:10]、合計 10 個の INT32 整数) を占有します。

Profinet コマンド

コマンド 101—— Mech-Vision プロジェクトを起動

機能紹介

このコマンドは、Mech-Vision のプロジェクトを起動し、カメラの撮影および視覚認識を行う場合に使われます。

プロジェクトは Eye In Hand モードである場合、このコマンドを用いて、ロボット撮影の位置姿勢をプロジェクトへ送信します。

このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使われます。

送信コマンド

パラメータ	ディスクリプション
Vision_Proj_Num	Mech-Vision プロジェクト番号
Req_Pose_Num	ビジョンポイントの期待数
Robot_Pose_Type	ロボット位置姿勢のタイプ
Robot_Pose_JPS / Robot_Pose_TCP	ロボット関節角度/フランジ位置姿勢

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

ビジョンポイントの期待数

Mech-Vision から取得したいビジョンポイントの数です。ビジョンポイント情報に、ビジョンポイント及びそれに対応する点群、ラベル、スケーリングの情報が含まれています。

- 0：Mech-Vision プロジェクトで認識できたすべての視覚結果を取得します。
- 0 より大きな整数：Mech-Vision プロジェクトで認識できた指定数の視覚結果を取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より大きい場合、認識結果にあるすべてのビジョンポイントを取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より小さい場合、このパラメータで指定された数のビジョンポイントを取得します。

ロボット位置姿勢のタイプ 、ロボットの関節角度/フランジ位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Vision に送信するタイプを設定します。パラメータ範囲は 0~3 です。
- **ロボットの関節角度/フランジ位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって決定されます。

Robot_Pose_Type、Robot_Pose_JPS と Robot_Pose_TCP の説明とパラメータ値についての詳細は下表に示します。

Robot_Pose_Type	Robot_Pose_JPS	Robot_Pose_TCP	説明	適用シーン
0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	Mech-Vision にロボットの位置姿勢を送信する必要がありません	プロジェクトは、Eye To Hand モードです。Mech-Vision プロジェクトで「経路計画」ステップを使用する場合、経路計画の開始位置は、経路計画ツールで設定した原点です。
1	ロボットの現在の関節角度	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があります	プロジェクトは、Eye In Hand モードです。この設定は、直行ロボット以外のほとんどのロボットで利用可能です。
2	0, 0, 0, 0, 0, 0	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在のフランジ位置姿勢を送信する必要があります	プロジェクトは、Eye In Hand モードです。ロボットは関節角度のデータを持たず、フランジい位置姿勢データのみを持ちます（直行ロボットの場合など）。
3	ロボット経路計画の開始位置の関節角度	0, 0, 0, 0, 0, 0	Mech-Vision にロボット経路計画の開始位置の関節角度を送信する必要があります	プロジェクトは、Eye To Hand モードです。また、Mech-Vision プロジェクトに「経路計画」ステップがあり、ロボット側から「経路計画」ステップの開始位置を設定する必要があります。

注意： 関節角度とフランジ位置姿勢の浮動小数点数のデータを 10000 倍して、Robot_Pose_JPS または Robot_Pose_TCP モジュールに設定する必要があります。

返されたデータ

`101, ステータスコード`

ステータスコード

コマンドが正常に実行された場合、**1102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 102——Mech-Vision の視覚目標点を取得

コマンド 101—— Mech-Vision プロジェクトを起動 の後に使用します。このコマンドは、Mech-Vision からビジョンポイントを取得して視覚目標点に変換するために使用されます。

以下に、ビジョンポイントに含まれる位置姿勢をロボット TCP に変換する処理を示します。

- ビジョンポイントに含まれる位置姿勢を Y 軸を中心に 180° 回転させます。
- 対応するロボット型番の基準座標系定義にロボットベースの高さが含まれているかどうかを認識し、それに応じて垂直方向のオフセットを増やします。

ヒント： デフォルトでは、102 コマンドは毎回最大 20 個までの視覚目標点を取得することができます。20 個以上の視覚目標点を取得するには、すべての視覚目標点を得るまで、102 コマンドを繰り返し実行してください。

送信コマンド

パラメータ	ディスクリプション
Command	102
Vision_Proj_Num	Mech-Vision プロジェクト番号

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

パラメータ	ディスクリプション
ステータスコード	ステータスコード
Send_Pose_Num	視覚目標点の数
Send_Pose_Type	位置姿勢のタイプ
Target_Pose	今回取得したすべての TCP
Target_Label	今回取得したすべてのラベル

ステータスコード

コマンドが正常に実行された場合、**1100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

視覚目標点の数

このコマンドを実行して、取得した視覚目標点の数です。

位置姿勢のタイプ

このパラメータは、Mech-Vision から返されたビジョンポイントでの対象物の位置姿勢を、対応する TCP に自動的に変換します。

このパラメータの初期値は 2 であり、位置姿勢のタイプが TCP であることを表します。

今回取得したすべての TCP

1 つの TCP には、姿勢座標 (XYZ) と方向オイラー角 (ABC) の情報が含まれます。

今回取得したすべてのラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

注意： 位置姿勢の送受信について、**通信制御プロセス** をご参照ください。Data_ready、Data_Acknowledge、Command_Complete 信号を用いて、プロセスのコントロールを行います。

コマンド 103—— Mech-Vision のパラメータレシピを切り替える

Mech-Vision プロジェクトのパラメータレシピを切り替えます。

パラメータレシピを切り替えることで、Mech-Vision プロジェクトの各ステップのパラメータを変更することができます。

パラメータレシピには画像マッチングテンプレート、ROI、信頼度のしきい値などのパラメータの設定が含まれています。

注意： **コマンド *101—— Mech-Vision* プロジェクトを起動** を実行する前に、このコマンドを使用する必要があります。

送信コマンド

パラメータ	ディスクリプション
Command	103
Vision_Proj_Num	Mech-Vision プロジェクト番号
Vision_Recipe_Num	Mech-Vision レシピ番号

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

Mech-Vision レシピ番号

Mech-Vision プロジェクトのレシピテンプレートの番号（正の整数）です。**プロジェクトアシスタント‣パラメータレシピ** をクリックして、パラメータレシピエディタに入ります。番号の有効範囲は 1~99 です。

返されたデータ

ステータスコード

コマンドが正常に実行された場合、**1107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 105——Mech-Vision の「経路計画」ステップの結果を取得

101 コマンドを呼び出した後、このコマンドを使用して Mech-Vision の「経路計画」ステップから出力された衝突ないの把持経路を取得します。

このコマンドを使用する時、Mech-Vision の「出力」ステップの **ポートタイプ** を「事前定義済み（ロボット経路）」に設定する必要があります。

ヒント： 105 コマンドを呼び出す前に、105 コマンドの呼び出し回数を減らすように 101 コマンドの **ビジョンポイントの期待数** を 0 に設定する必要があります。101 コマンドの **ビジョンポイントの期待数** を 1 に設定すると、105 コマンドの呼び出しごとに 1 つの経路点のみが返され、105 コマンドを複数回呼び出した場合にのみすべての経路点が返されます。

送信コマンド

パラメータ	ディスクリプション
Command	105
Vision_Proj_Num	Mech-Vision プロジェクト番号
Req_Pose_Type	経路点の位置姿勢タイプ

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

経路点の位置姿勢タイプ

このパラメータは、「経路計画」ステップから返された経路点の位置姿勢タイプを指定するために使用されます。

- `1`：経路点の位置姿勢は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点の位置姿勢は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	ディスクリプション
ステータスコード	ステータスコード
Send_Pose_Num	経路点の数
Send_Pose_Type	位置姿勢のタイプ
Visual_Point_Index	「ビジョン処理による移動」の位置
Target_Pose	今回送信したすべての経路点の位置姿勢
Target_Label	今回送信したすべての経路点のラベル
Target_Speed	今回送信したすべての経路点の速度

ステータスコード

コマンドが正常に実行された場合、**1103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

経路点の数

このパラメータは、このコマンドを実行した後に返された経路点の数を表します。範囲は 0~20 です。20 以上の経路点を取得するには、このコマンドを繰り返してください。

位置姿勢のタイプ

このコマンドで送信される「Req_Pose_Type」の値が同じです。

- `1`: JPs
- `2`: TCP

「ビジョン処理による移動」の位置

経路計画ツールで設定された「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 105 送信時の **経路点の位置姿勢タイプ** によって、三次元座標、XYZ オイラー角、あるいは関節角度（JPs）で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に ラベルマッピング ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

経路計画設定ツールで設定された速度値です。

コマンド 201——Mech-Viz プロジェクトを起動

このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使われます。Mech-Viz プロジェクトを開始し、対応する Mech-Vision プロジェクトを呼び出し、Mech-Viz が Mech-Vision の視覚結果に基づいて経路を計画する時に使用されます。

Mech-Viz では、**自動的に読み込む** にチェックを入れる必要があります。

Mech-Center のインストールディレクトリ（tool¥viz_project）フォルダには、サンプルプロジェクトがあり、それら基づいて修正することが可能です。

標準インターフェースに使用される Mech-Viz サンプルプロジェクトの詳細な説明については、**標準インターフェースに使用される *Mech-Viz* サンプルプロジェクト** をご参照ください。

送信コマンド

パラメータ	ディスクリプション
Command	201
Robot_Pose_Type	ロボット位置姿勢のタイプ
Robot_Pose_JPS / Robot_Pose_TCP	ロボット関節角度/フランジ位置姿勢

ロボット位置姿勢のタイプ 、ロボットの関節角度/フランジ位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Viz に送信するタイプを設定します。パラメータ範囲は 0~2 です。
- **ロボットの関節角度/フランジ位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって決定されます。

Robot_Pose_Type、Robot_Pose_JPS と Robot_Pose_TCP の説明とパラメータ値についての詳細は下表に示します。

Robot_Pose_Type	Robot_Pose_JPS	Robot_Pose_TCP	説明	適用シーン
0	0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0	Mech-Viz にロボットの位置姿勢を送信する必要がありません。Mech-Viz での仮想ロボットは初期位置姿勢（JPs = [0, 0, 0, 0, 0, 0] ）から最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨しません。
1	ロボットの現在の関節角度	ロボットの現在のフランジ位置姿勢	Mech-Viz にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があり、Mech-Viz での仮想ロボットは受信された位置姿勢から最初の経路点に移動します。	プロジェクトは Eye In Hand モードである場合は、この設定は推奨します。
2	ロボット側でカスタマイズされた関節角度	0, 0, 0, 0, 0, 0	Mech-Viz にロボットのティーチポイント（現在の関節角度ではない）を送信する必要があります。これは、ロボットが画像撮影領域の外にいるとき（下図に示す）、Mech-Viz プロジェクトが次回の経路を事前に計画することをトリガーするために使用されます。Mech-Viz での仮想ロボットは受信された最初のティーチポイントから最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨します。

Eye To Hand モードでは **Robot_Pose_Type** を 2 に設定する理由は以下の通りです。

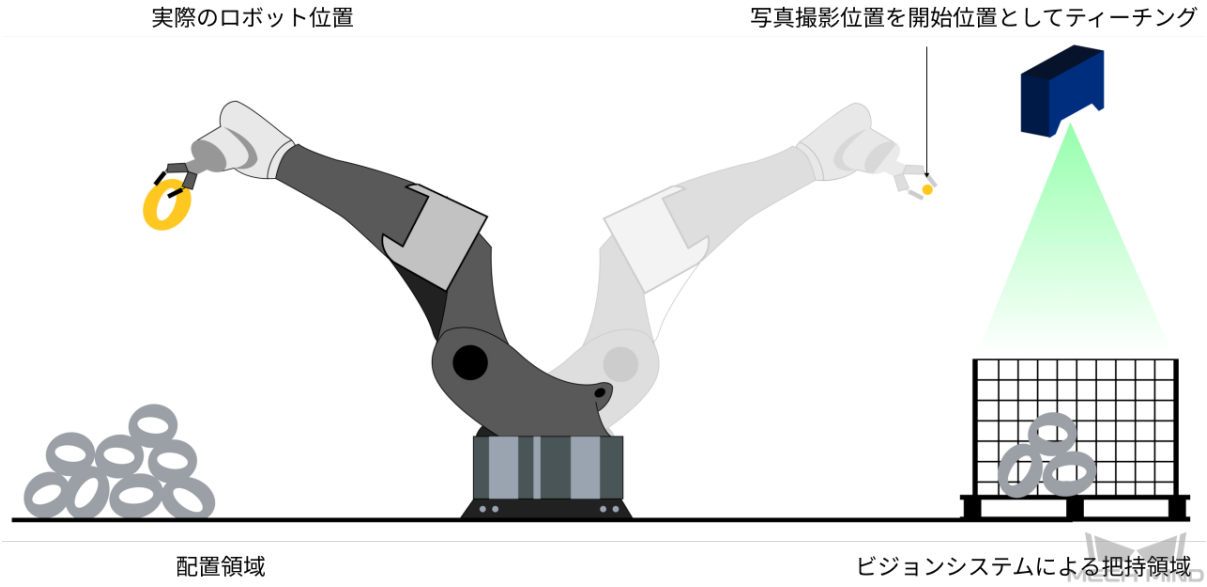
Eye To Hand モードでは、カメラはロボットが画像撮影領域と把持領域に戻る前に撮影し、次回の把持経路を計画することができます。これにより、タクトタイムの向上

が可能です。

この場合、**Robot_Pose_Type** を 1 に設定すると、ロボットの現在の位置姿勢が Mech-Viz の仮想ロボットに送信され、シミュレーションと実際のロボットの経路が予測不可能な衝突を起こす可能性があります。

仮想ロボットは現在の位置姿勢から Mech-Viz での最初の移動ステップで設定された位置姿勢に移動しますが、ロボット実機は上記の位置姿勢に移動する前に別の位置姿勢に移動する可能性があるということです。

したがって、**Robot_Pose_Type** パラメータを 2 に設定する必要があります。



返されたデータ

ステータスコード

コマンドが正常に実行された場合、**2103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 202—— Mech-Viz プロジェクトを停止

Mech-Viz の実行を停止させます。Mech-Viz プロジェクトがデッドループでない場合、または正常に停止できる場合は、このコマンドを使う必要がありません。

送信コマンド

パラメータ	ディスクリプション
Command	202

返されたデータ

ステータスコード

コマンドが正常に実行された場合、**2104** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 203——Mech-Viz 分岐を選択

このコマンドは、Mech-Viz プロジェクトの分岐を指定する場合に使われます。分岐メカニズムは branch_by_msg ステップによって作成されたら、このコマンドがステップの出口を指定することで実現します。

このコマンドを実行する前に、**コマンド** *201——Mech-Viz* **プロジェクトを起動** を実行してください。

Mech-Viz プロジェクトが branch_by_msg ステップに実行すると、このコマンドによって出口を指定するのを待ちます。

送信コマンド

パラメータ	ディスクリプション
Command	203
Viz_Task_ID	分岐ステップ ID
Viz_Task_Value	分岐の出口番号

分岐ステップ ID

このパラメータは、分岐選択が行われる branch_by_msg ステップを指定するために使用されます。

このパラメータ、つまり、branch_by_msg のステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

出口番号

このパラメータは、プロジェクトが branch_by_msg ステップに沿って実行される出口を指定します。Mech-Viz プロジェクトはこの出口に従って実行し続けます。パラメータの値は正の整数です。

ヒント:

- 出口番号は、Mech-Viz で表示されるポート番号に 1 を加えたもので、ポート 0 が出口 1 です。
- 出口番号は、1 から始まるポートのインデックス番号です。たとえば、指定された出口が左から右に 2 番目のポートである場合、番号は 2 です。

返されたデータ

ステータスコード

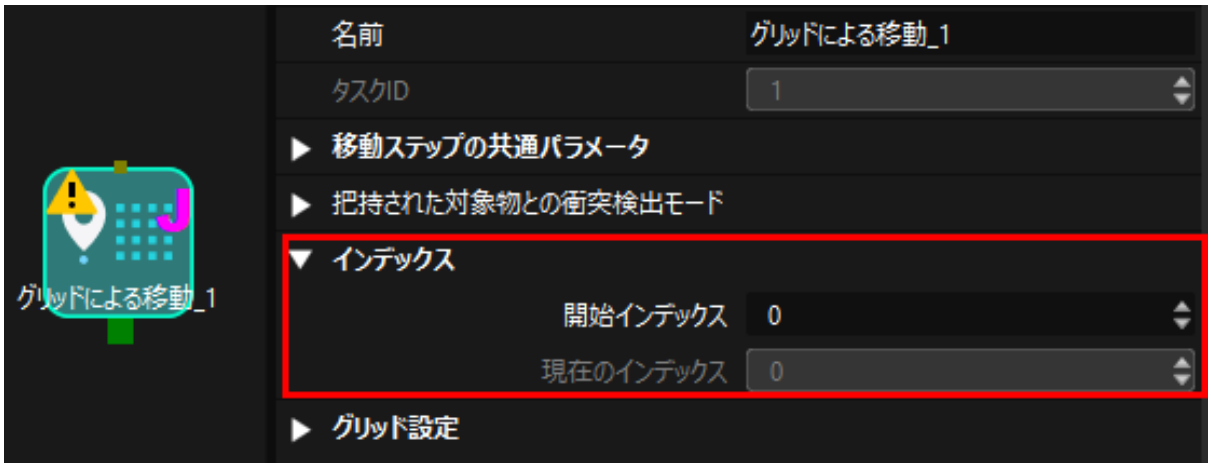
コマンドが正常に実行された場合、**2105** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 204——移動インデックスを設定

このコマンドはステップのインデックスパラメータを設定する時に使われます。一般的には、連続な移動か指定された移動ステップやパレタイジングなどに使用されます。

インデックスパラメータが付いたステップは「リストによる移動」、「グリッドによる移動」、「カスタマイズのパレットパターン」、「事前計画したパレットパターン」などです。

このコマンドを実行する前に、**コマンド *201——Mech-Viz* プロジェクトを起動** を実行してください。



送信コマンド

パラメータ	ディスクリプション
Command	204
Viz_Task_ID	ステップ ID
Viz_Task_Value	インデックス

ステップ ID

このパラメータはどのステップがインデックスを設定する必要かを指定します。

このパラメータ、つまり、インデックス付きのステップのステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

インデックス

次にこのステップが実行されたときに設定されるべきインデックス値です。

このコマンドを送信すると、Mech-Viz の現在のインデックス値がこのパラメータの値から 1 を引いた値に変更されます。

このコマンドで指定したステップに Mech-Viz プロジェクトが実行されると、Mech-Viz の現在のインデックス値が、このパラメータの値まで 1 つずつ増加します。

返されたデータ

ステータスコード

コマンドが正常に実行された場合、**2106** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 205——Mech-Viz の計画経路を取得

コマンド *201——Mech-Viz* プロジェクトを起動 を実行した後、このコマンドは Mech-Viz が計画した経路を取得するために使用されます。

初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

注釈: プロジェクト内の移動ステップの経路点をロボットに送信しない場合は、ステップパラメータで「移動目標点を送信」のチェックを外してください。

送信コマンド

パラメータ	ディスクリプション
Command	205
Req_Pose_Type	経路点タイプ

経路点タイプ

このパラメータは Mech-Viz からどのような形式で経路点を返すかを指定します。

- `1`：経路点は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	ディスクリプション
ステータスコード	ステータスコード
Send_Pose_Num	経路点の数
Send_Pose_Type	位置姿勢のタイプ
Visual_Point_Index	「ビジョン処理による移動」の位置
Target_Pose	今回送信したすべての経路点の位置姿勢
Target_Label	今回送信したすべての経路点のラベル
Target_Speed	今回送信したすべての経路点の速度

ステータスコード

コマンドが正常に実行された場合、**2100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

ヒント: このコマンドを実行する時、Mech-Viz から結果が返されていない場合（まだ実行中）、Mech-Center は Mech-Viz からの結果を待って、それをロボットに送信します。デフォルトの場

合、タイムアウト時間は 10 秒であり、待ち時間が超えるとタイムアウトエラーをロボットに返します。

経路点の数

このパラメータは、返された経路点の数を表します。範囲は 0~20 です。初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。従って、20 以上の経路点を取得するには、このコマンドを繰り返してください。

位置姿勢のタイプ

このコマンドで送信される「Req_Pose_Type」の値が同じです。

- `1`: JPs
- `2`: TCP

「ビジョン処理による移動」の位置

「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 205 発送時のパラメータによって、三次元座標、オイラー角、あるいは関節角度で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

移動ステップパラメータに設定されたゼロでないパーセンテージ値です。

注意: 位置姿勢の送受信について、**通信制御プロセス** をご参照ください。Data_ready、Data_Acknowledge、Command_Complete 信号を用いて、プロセスのコントロールを行います。

コマンド 206——DO リストを取得

複数の治具または吸盤パーティションを制御する場合、このコマンドを用いて DO リストを取得します。

このコマンドを実行する前に、**コマンド** *205* を実行して Mech-Viz の計画経路を取得する必要があります。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定し、プロジェクトに対応する吸盤のコンフィグファイルを設定します。サンプルプロジェクトのパスは Mech-Center のインストールディレクトリ (tool¥viz_project) の **suction_zone** です。

プロジェクトの set_do_list ステップのパラメータで以下の設定を行います。

- 「受信者」を「標準インターフェース」に設定します
- 「ビジョン処理による移動から DO リストを取得」にチェックを入れます
- パラメータバーの下部にあるエリアに DO リストが必要な「ビジョン処理による移動」ステップを選択します

送信コマンド

パラメータ	ディスクリプション
Command	206

返されたデータ

パラメータ	ディスクリプション
Status Code	ステータスコード
DO リスト	DO 信号リスト

ステータスコード

コマンドが正常に実行された場合、**2102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

DO 信号リスト

返されたデータの最後に 64 個の DO 値が受信され、すべては整数です。

DO 値の範囲は 0~999 です。

-1 はプレースホルダーとなります。

コマンド 501——Mech-Vision プロジェクトへ対象物の寸法を送信

このパラメータは Mech-Vision プロジェクトに、対象物の寸法を動的に送信すること場合に使われます。Mech-Vision プロジェクトを実行する前に対象物の寸法を確認する必要があります。

Mech-Vision プロジェクトに、read_object_dimensions ステップを入れておきます。このステップのパラメータ **パラメータから対象物の寸法を読み取る** にチェックを入れる必要があります。

送信コマンド

パラメータ	ディスクリプション
Command	501
Vision_Proj_Num	Mech-Vision プロジェクト番号
Ext_Input_Data	対象物の寸法

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

対象物の寸法

Mech-Vision プロジェクトに送信した対象物の寸法（長さ、幅、高さ）の値は、ステップ read_object_dimensions によって読み取られます。

単位はミリメートル（mm）です。

返されたデータ

ステータスコード

コマンドが正常に実行された場合、**1108** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 502——Mech-Viz へ TCP を送信

このコマンドは、Mech-Viz プロジェクトにロボット TCP を動的に送信するためによく使われます。ロボット TCP を読み取るためのステップは outer_move です。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定します。サンプルプロジェクトのパスは Mech-Center のインストールディレクトリ `tool¥viz_project¥outer_move` です。

outer_move ステップをワークフローの適切な場所に配置します。

このコマンドは、**コマンド *201*——*Mech-Viz* プロジェクトを起動** を実行する前に実行してください。

送信コマンド

パラメータ	ディスクリプション
Command	502
Ext_Input_Data	ロボット TCP

ロボット TCP

outer_move ステップの経路点の TCP データを設定するために使用されます。

> **注意:** ロボットの TCP データ（ミリメートル単位）を 10000 倍して、Ext_Input_Data の値を設定する必要があります。

返されたデータ

ステータスコード

コマンドが正常に実行された場合、**2107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 901——ソフトウェアの起動状態を取得

このコマンドは、Mech-Vision、Mech-Viz、Mech-Center の起動状態を取得するために使用されます。

現在、このコマンドは Mech-Vision のみ対応できます。

送信コマンド

パラメータ	ディスクリプション
Command	901

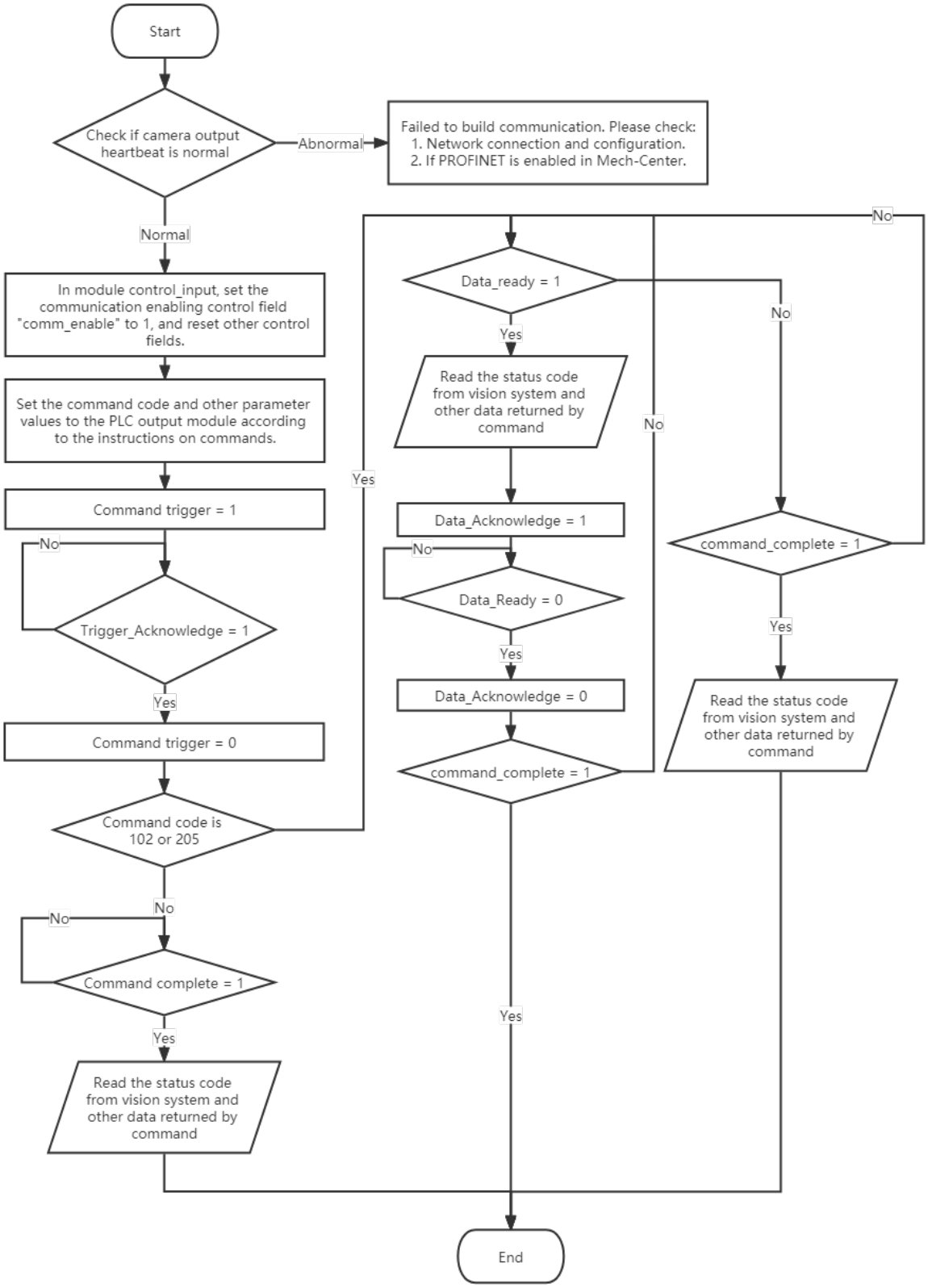
返されたデータ

ステータスコード

システムのセルフチェックステータスです。ステータスコードが **1101** になると「Mech-Vision が準備できました」です。他のステータスコードが出てきた場合「Mech-Vision プロジェクトがまだ準備できていません」ということです。現在、このコマンドは Mech-Vision プロジェクトの準備ができているかどうかを確認するためにのみ使用できます。

通信制御プロセス

Mech-Mind ビジョンシステムは、下図のように PROFINET 制御フローを使用しています。


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

3.3.5 EtherNet/IP

EtherNet/IP プロトコルに基づく標準インターフェースを介して、Mech-Mind ソフトウェア システムは Omron PLC および Keyence PLC と通信できます。

上記についての詳しい内容は、standard_interface_robot_and_plc をご参照ください。コマンドの説明については、*Profinet* コマンド をご参照ください。

3.3.6 Modbus TCP

Mech-Mind ソフトウェアシステムは、Modbus TCP 通信プロトコルに基づいた標準インターフェースを通じてマスター（PLC またはロボットコントローラ）と通信することが可能です。

詳細については、Modbus TCP - Siemens SIMATIC S7 PLC と《Modbus TCP - 三菱 Q シリーズ PLC》をご参照ください。

レジスタマッピング

詳細は下表のどおりです。

アドレス（10進数）	アドレス（16進数）	意味	長さ（文字単位）	読み取り/書込み	そのほか	保持レジスタ（4x）
0	0x0000	コマンドを実行させる	1	書込み	0：コマンドを実行させない 1：コマンドを実行させる	40001 – 40053：Mech-Center に書き込む
1	0x0001	コマンド	1	書込み	機能コード	
2	0x0002	位置姿勢タイプ	1	書込み		
3	0x0003	位置姿勢の数	1	書込み		
4	0x0004	Mech-Vision プロジェクト番号	1	書込み		
5	0x0005	レシピ番号	1	書込み		
6	0x0006	関節角度	12	書込み	単位は度	
18	0x0012	TCP	12	書込み	単位はミリメートルとオイラー角の度	
30	0x001E	分岐ステップ ID	1	書込み		
31	0x001F	分岐出口	1	書込み		
32	0x0020	インデックス名	1	書込み		
33	0x0021	インデックス⊠	1	書込み		
34	0x0022	外部からの箱の寸法	6	書込み	単位はミリメートル	
40	0x0028	外部から入力した位置姿勢	12	書込み	単位はミリメートルとオイラー角の度	
52	0x0034	ロボットの動作状態	1	書込み		
53	0x0035	予約	44	読み取り		40054 – 40728：Mech-Center を読み取る
97	0x0061	実行確認	1	読み取り	0：コマンド実行を受け入れない 1：コマンド実行を受け入れる	
98	0x0062	通知	1	読み取り		
99	0x0063	ハートビート	1	読み取り	1 Hz	
100	0x0064	ステータスコード	1	読み取り		
101	0x0065	位置姿勢データのステータス	1	読み取り	0：999 コマンド（レジスタデータを削除）が受信された 1：新しい位置姿勢データ	
102	0x0066	送信した位置姿勢の数	1	読み取り		
103	0x0067	ビジョンポイントが計画した経路における位置	1	読み取り		
104	0x0068	目標点	480	読み取り		
584	0x0248	目標ラベル	40	読み取り		
624	0x0270	速度のパーセンテージ値	40	読み取り		
664	0x0298	DO 信号リスト	64	読み取り		

注釈： Modbus TCP は簡単なバスプロトコルであり、1 回の読み取りまたは書込みの通信周期は 70ms 程度で、100 ワード程度の読み取りまたは書込みが推奨されています。可能な限り、PROFINET や EtherNet/IP などのリアルタイム EtherNet プロトコル、シーメンス独自の Snap7 通信プロトコル（Mech-Center Siemens PLC Client 標準インタフェースに対応）、三菱独自の MELSEC 通信プロトコル（MC プロトコル）を選択してください。

保持レジスタのコマンド

コマンド 101——Mech-Vision プロジェクトを起動

このコマンドは、Mech-Vision のプロジェクトを起動し、カメラの撮影およびビジョン認識を行う場合に使われます。

プロジェクトは Eye In Hand モードである場合、このコマンドを用いて、ロボット撮影の位置姿勢をプロジェクトへ送信します。

このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使われます。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 101	1
Mech-Vision プロジェクト番号	4
ビジョン位置姿勢の期待数	3
ロボット位置姿勢のタイプ	2
ロボット位置姿勢	6-17（JPs）または 18-29（フランジ位置姿勢）

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

ビジョン位置姿勢の期待数

Mech-Vision から取得したいビジョンポイントの数です。ビジョンポイントの情報に、ビジョン位置姿勢及びそれに対応する点群、ラベル、スケーリングなどの情報が含まれています。

- 0：Mech-Vision プロジェクトで認識できたすべてのビジョンポイントを取得します。
- 0 より大きな整数：Mech-Vision プロジェクトで認識できた指定数のビジョンポイントを取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より大きい場合、認識結果にあるすべてのビジョンポイントを取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より小さい場合、このパラメータで指定された数のビジョンポイントを取得します。

ヒント： ビジョンポイントを取得するためのコマンドは 102 コマンドです。デフォルトの設定環境において、このコマンドで一度に最大 20 個のビジョンポイントしか取得できないため、必要なビジョンポイントの数が 20 個より多い場合、このコマンド繰り返して実行する必要があります。

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Vision に送信するタイプを設定します。パラメータ範囲は 0~3 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	ロボット位置姿勢 パラメータ	説明	適用シーン
0	0, 0, 0, 0, 0, 0	Mech-Vision にロボットの位置姿勢を送信する必要がありません。	プロジェクトは、Eye To Hand モードです。Mech-Vision プロジェクトで「経路計画」ステップを使用する場合、経路計画の開始位置は、経路計画設定ツールで設定した原点です。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Vision にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。この設定は、直行ロボット以外のほとんどのロボットで利用可能です。
2	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在のフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。ロボットは関節角度のデータを持たず、フランジ位置姿勢データのみを持ちます（直行ロボットの場合など）。
3	ロボット経路計画の開始位置の関節角度	Mech-Vision にロボット経路計画の開始位置の関節角度を送信する必要があります。	プロジェクトは、Eye To Hand モードです。また、Mech-Vision プロジェクトに「経路計画」ステップがあり、ロボット側から「経路計画」ステップの開始位置を設定する必要があります。

ロボットの位置姿勢は Float タイプの数字です。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 102——Mech-Vision のビジョン目標点を取得

コマンド *101——Mech-Vision* プロジェクトを起動 の後に使用します。このコマンドは、Mech-Vision からビジョンポイントを取得し、それをビジョン目標点に変換するために使用されます。

以下に、ビジョンポイントに含まれる位置姿勢をロボット TCP に変換する処理を示します。

- ビジョンポイントに含まれる位置姿勢を Y 軸を中心に 180° 回転させます。
- 対応するロボット型番の基準座標系定義にロボットベースの高さが含まれているかどうかを認識し、それに応じて垂直方向のオフセットを増やします。

ヒント: デフォルトでは、102 コマンドは毎回最大 20 個までのビジョン目標点を取得することができます。20 個以上のビジョン目標点を取得するには、すべてのビジョン目標点を得るまで、102 コマンドを繰り返

し実行してください。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 102	1
Mech-Vision プロジェクト番号	4

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100
データ転送状態	101
ビジョン目標点の数	102
予約語	/
今回取得したすべての TCP	104
今回取得したすべてのラベル	584

ステータスコード

コマンドが正常に実行された場合、**1100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しいビジョン目標点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョン目標点であることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

ビジョン目標点の数

このコマンドを実行して、取得したビジョン目標点の数です。

- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョンポイントの数よりも多い場合、Mech-Vision によって認識されたビジョンポイントの数に従って送信されます。
- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョンポイントの数よりも少ない場合、リクエストした数に従って送信されます。

予約語

この予約語が使われていないため、初期値は 0 です。

今回取得したすべての TCP

1 つの TCP には、空間座標 (XYZ) とオイラー角 (ABC) の情報が含まれます。

今回取得したすべてのラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

コマンド 103——Mech-Vision のパラメータレシピの切り替え

Mech-Vision プロジェクトのパラメータレシピを切り替えます。

パラメータレシピを切り替えることで、Mech-Vision プロジェクトの各ステップのパラメータを変更することができます。

パラメータレシピには画像マッチングテンプレート、ROI、信頼度のしきい値などのパラメータの設定が含まれています。

注意： コマンド *101——Mech-Vision* プロジェクトを起動 を実行する前に、このコマンドを使用する必要があります。

送信コマンド

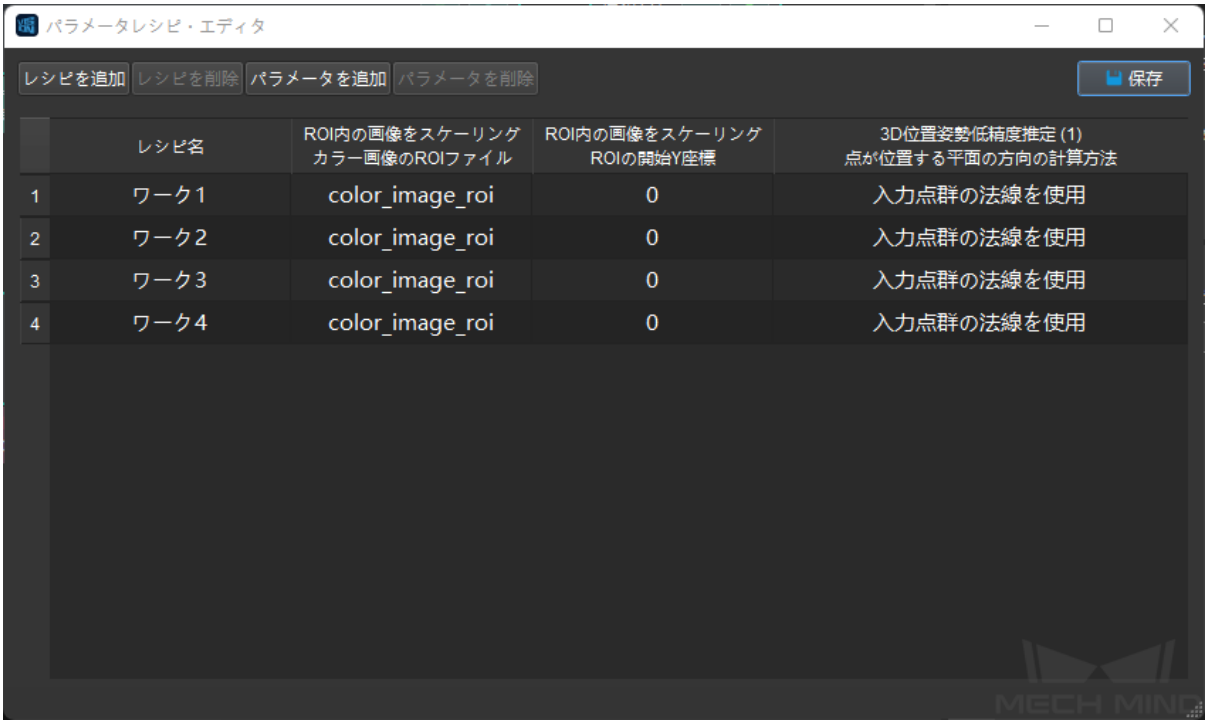
パラメータ	アドレス (10 進数)
コマンド 103	1
Mech-Vision プロジェクト番号	4
レシピ番号	5

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

レシピ番号

Mech-Vision プロジェクトのレシピテンプレートの番号（正の整数）です。**プロジェクトアシスタント ‣ パラメータレシピ** をクリックして、パラメータレシピエディタに入ります。番号の有効範囲は 1~99 です。



返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 105——Mech-Vision の「経路計画」ステップの結果を取得

101 コマンドを呼び出した後、このコマンドを使用して Mech-Vision の「経路計画」ステップから出力された衝突のない把持経路を取得します。

このコマンドを使用する時、Mech-Vision の「出力」ステップの **ポートタイプ** を「事前定義済み（ロボット経路）」に設定する必要があります。

ヒント： 105 コマンドを呼び出す前に、105 コマンドの呼び出し回数を減らすように 101 コマンドの **ビジョンポイントの期待数** を 0 に設定する必要があります。101 コマンドの **ビジョンポイントの期待数** を 1 に設定すると、105 コマンドの呼び出しごとに 1 つの経路点のみが返され、105 コマンドを複数回呼び出した場合にのみすべての経路点が返されます。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 105	1
Mech-Vision プロジェクト番号	4
経路点の位置姿勢タイプ	2

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

経路点の位置姿勢タイプ

このパラメータは、「経路計画」ステップから返された経路点の位置姿勢タイプを指定するために使用されます。

- 1：経路点の位置姿勢は、ロボットの関節角度（JPs）の形式で返されます。
- 2：経路点の位置姿勢は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100
データ転送状態	101
経路点の数	102
「ビジョン処理による移動」の位置	103
今回送信したすべての経路点の位置姿勢	104
今回送信したすべての経路点のラベル	584
今回送信したすべての経路点の速度	624

ステータスコード

コマンドが正常に実行された場合、**1103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

1：返されたデータが新しい経路点であることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、このコマンドを実行した後に返された経路点の数を表します。範囲は 0~20 です。20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

経路計画ツールで設定された「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は定点移動_1 -> 定点移動_2 -> ビジョン処理による移動 -> 定点移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 105 送信時の **経路点の位置姿勢タイプ** によって、三次元座標、XYZ オイラー角、あるいは関節角度（JPs）で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に ラベルマッピング ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

経路計画設定ツールで設定された速度値です。

コマンド 201——Mech-Viz プロジェクトを起動

このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使われます。Mech-Viz プロジェクトを開始し、対応する Mech-Vision プロジェクトを呼び出し、Mech-Viz が Mech-Vision のビジョン結果に基づいて経路を計画する時に使用されます。

Mech-Viz では、**自動的に読み込む** にチェックを入れる必要があります。

Mech-Center のインストールディレクトリ（tool¥viz_project）フォルダには、サンプルプロジェクトがあり、それら基づいて修正することが可能です。

標準インターフェースに使用される Mech-Viz サンプルプロジェクトの詳細な説明については、**標準インターフェースに使用される *Mech-Viz* サンプルプロジェクト** をご参照ください。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 201	1
ロボット位置姿勢のタイプ	2
ロボット位置姿勢	6-17（JPs）または 18-29（フランジ位置姿勢）

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Viz に送信するタイプを設定します。パラメータ範囲は 0~2 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Viz にロボットの位置姿勢を送信する必要がありません。Mech-Viz での仮想ロボットは初期位置姿勢（JPs = [0, 0, 0, 0, 0, 0] ）から最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨しません。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Viz にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があり、Mech-Viz での仮想ロボットは受信された位置姿勢から最初の経路点に移動します。	プロジェクトは Eye In Hand モードである場合は、この設定は推奨します。
2	ロボット側でカスタマイズされた関節角度	Mech-Viz にロボットのティーチポイント（現在の関節角度ではない）を送信する必要があります。これは、ロボットが画像撮影領域の外にいるとき（下図に示す）、Mech-Viz プロジェクトが次回の経路を事前に計画することをトリガーするために使用されます。Mech-Viz での仮想ロボットは受信された最初のティーチポイントから最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨します。

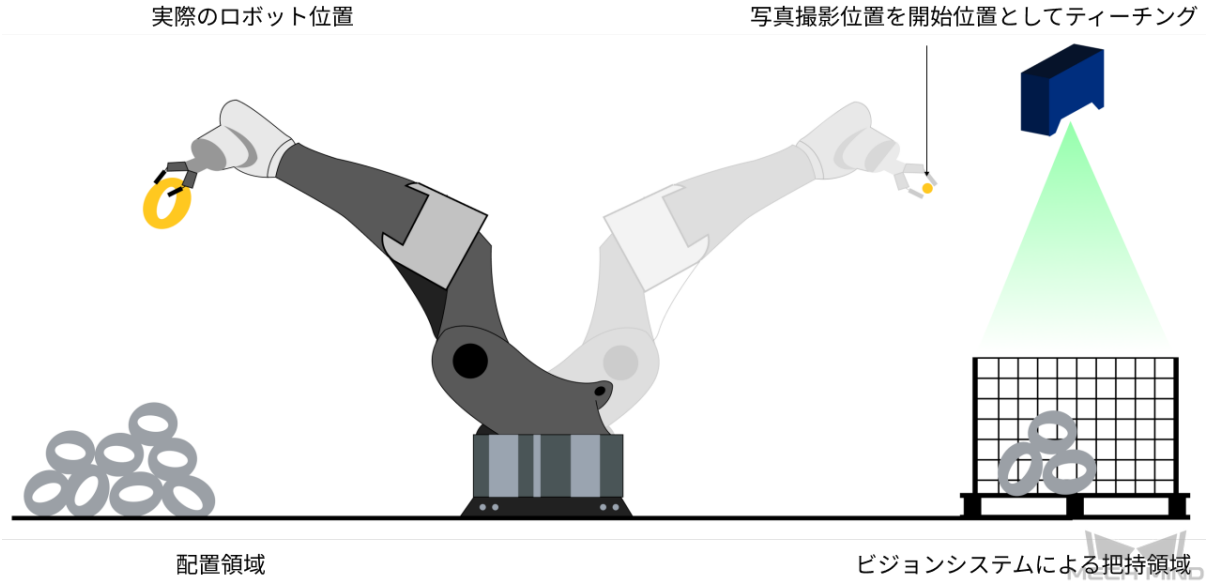
Eye To Hand モードでは、**ロボット位置姿勢のタイプ** が 2 に設定されている理由は以下の通りです。

Eye To Hand モードでは、カメラはロボットが画像撮影領域と把持領域に戻る前に撮影し、次回の把持経路を計画することができます。これにより、タクトタイムの向上が可能です。

この時、**ロボット位置姿勢のタイプ** が 1 に設定され、つまり現在の位置姿勢を Mech-Viz に送信すれば、仮想ロボットがロボット実機の経路と一致しない可能性があります。また、未知の衝突が発生する可能性もあります。

仮想ロボットは現在の位置姿勢から Mech-Viz での最初の移動ステップで設定された位置姿勢に移動しますが、ロボット実機は上記の位置姿勢に移動する前に別の位置姿勢に移動する可能性があるということです。

したがって、**ロボット位置姿勢のタイプ** パラメータを 2 に設定する必要があります。



返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 202—— Mech-Viz プロジェクトを停止

Mech-Viz プロジェクトの実行を停止します。Mech-Viz プロジェクトは無限ループになっていない場合や、正常に停止できる場合は、このコマンドを使用する必要がありません。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 202	1

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2104** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 203——Mech-Viz ブランチを選択

このコマンドは、Mech-Viz プロジェクトの分岐を指定する場合に使われます。分岐メカニズムは branch_by_msg ステップによって作成されたら、このコマンドがステップの出口を指定することで実現します。

このコマンドを実行する前に、**コマンド *201——Mech-Viz* プロジェクトを起動** を実行してください。

Mech-Viz プロジェクトが branch_by_msg ステップに実行すると、このコマンドによって出口を指定するのを待ちます。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 203	1
分岐ステップ ID	30
分岐の出口番号	31

分岐ステップ ID

このパラメータは、分岐選択が行われる branch_by_msg ステップを指定するために使用されます。

このパラメータ、つまり、branch_by_msg のステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

分岐の出口番号

このパラメータは、プロジェクトが branch_by_msg ステップに沿って実行される出口を指定します。Mech-Viz プロジェクトはこの出口に従って実行し続けます。パラメータの値は正の整数です。

出口番号の範囲は 1~99 です。

ヒント:

- 出口番号は、Mech-Viz で表示されるポート番号に 1 を加えたものです。例えば、番号が 0 の場合、出口番号は 1 です。
- 出口番号は、1 から始まるポートのインデックス番号です。例えば、指定された出口が左から右に 2 番目のポートである場合、番号は 2 です。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

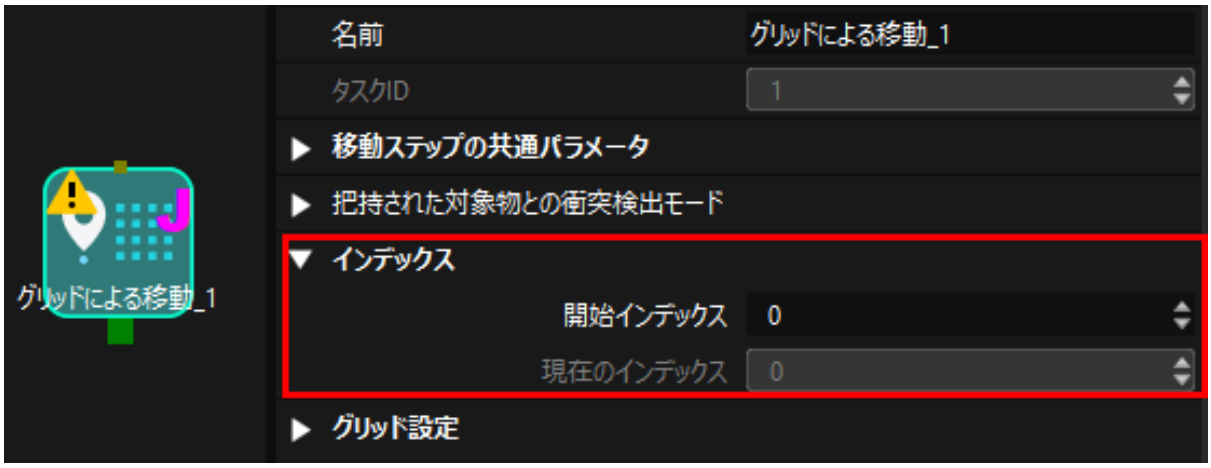
コマンドが正常に実行された場合、**2105** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 204——移動インデックスを設定

このコマンドはステップのインデックスパラメータを設定する時に使われます。一般的には、連続な移動か指定された移動ステップやパレタイジングなどに使用されます。

インデックスパラメータが付いたステップは「リストによる移動」、「グリッドによる移動」、「カスタマイズのパレットパターン」、「事前計画したパレットパターン」などです。

このコマンドを実行する前に、**コマンド *201*——*Mech-Viz* プロジェクトを起動** を実行してください。



送信コマンド

パラメータ	アドレス (10 進数)
コマンド 204	1
ステップ ID	32
インデックス☒	33

ステップ ID

このパラメータはどのステップがインデックスを設定する必要かを指定します。

このパラメータ、つまり、インデックス付きのステップのステップ ID は正の整数である必要があります。ステップ ID は、ステップパラメータで読み取りを行います。

インデックス☒

次にこのステップが実行されたときに設定されるべきインデックス値です。

このコマンドを送信すると、Mech-Viz の現在のインデックス値がこのパラメータの値から 1 を引いた値に変更されます。

このコマンドで指定したステップに Mech-Viz プロジェクトが実行されると、Mech-Viz の現在のインデックス値が、このパラメータの値まで 1 つずつ増加します。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2106** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 205——Mech-Viz の計画経路を取得

コマンド 201——Mech-Viz プロジェクトを起動 を実行した後、このコマンドは Mech-Viz が計画した経路を取得するために使用されます。

初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

注釈: プロジェクト内の移動ステップの経路点をロボットに送信しない場合は、ステップパラメータで「移動目標点を送信」のチェックを外してください。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 205	1
経路点タイプ	2

経路点タイプ

このパラメータは Mech-Viz からどのような形式で経路点を返すかを指定します。

- `1`：経路点は、ロボットの関節角度（JPs）の形式で返されます。
- `2`：経路点は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100
データ転送状態	101
経路点の数	102
「ビジョン処理による移動」の位置	103
今回送信したすべての経路点の位置姿勢	104
今回送信したすべての経路点のラベル	584
今回送信したすべての経路点の速度	624

ステータスコード

コマンドが正常に実行された場合、**2100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

ヒント: このコマンドを呼び出す時、Mech-Viz の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョンポイントであることを示し、それを読み取ります。

注意: 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、返された経路点の数を表します。範囲は 0~20 です。初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。従って、20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 205 送信時のパラメータによって、三次元座標、オイラー角、あるいは関節角度で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に label_mapping ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

移動ステップパラメータに設定されたゼロでないパーセンテージ値です。

コマンド 206——DO リストを取得

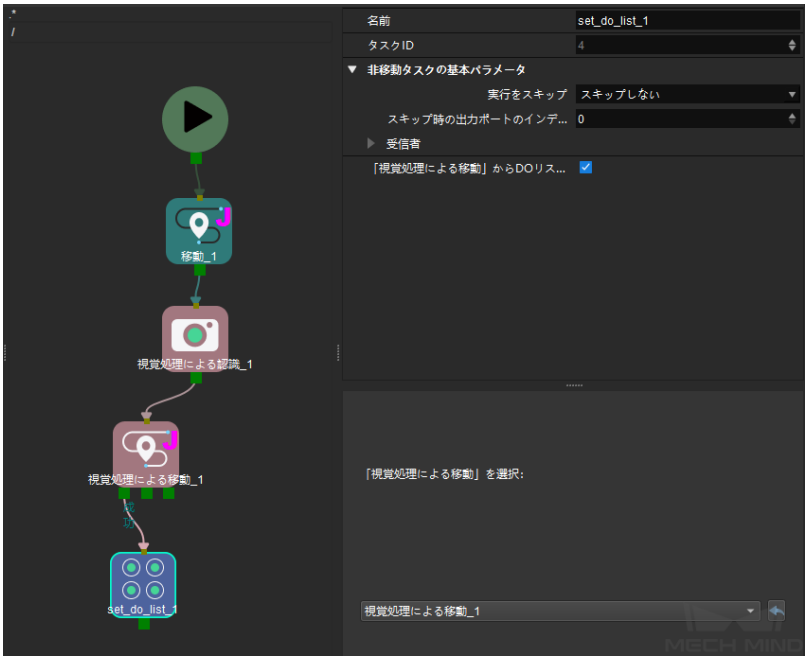
複数の治具または吸盤パーティションを制御する場合、このコマンドを用いて DO リストを取得します。

このコマンドを実行する前に、**コマンド** *205* を実行して Mech-Viz の計画経路を取得する必要があります。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定し、プロジェクトに対応する吸盤のコンフィグファイルを設定します。サンプルプロジェクトのパスは `Mech-Center\tool\viz_project\suction_zone` です。

プロジェクトの set_do_list ステップのパラメータで以下の設定を行います。

- 「受信者」を「標準インターフェース」に設定します
- 「ビジョン処理による移動から DO リストを取得」にチェックを入れます
- パラメータバーの下部にあるエリアに DO リストが必要な「ビジョン処理による移動」ステップを選択します



送信コマンド

パラメータ	アドレス (10 進数)
コマンド 206	1

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100
DO 信号リスト	664

ステータスコード

コマンドが正常に実行された場合、**2102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

DO 値

64 個の DO 値が受信され、すべては整数です。

DO 値の範囲は 0~999 です。

-1 はプレースホルダーとなります。

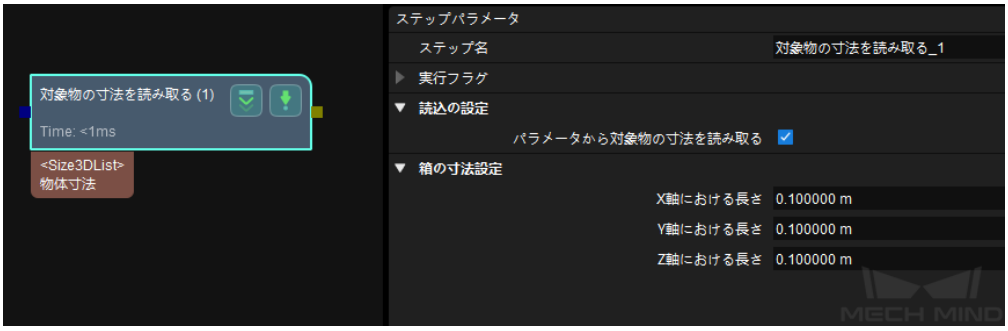
例えば、DO 信号値は 1、3、5、6 です。

1	3	5	6	-1	-1	-1	-1	...	-1	-1
1 番目の整数	2 番目の整数	3 番目の整数	4 番目の整数	5 番目の整数	6 番目の整数	7 番目の整数	8 番目の整数	...	63 番目の整数	64 番目の整数

コマンド 501——Mech-Vision プロジェクトへ対象物の寸法を送信

このパラメータは Mech-Vision プロジェクトに、対象物の寸法を動的に送信すること場合に使われます。Mech-Vision プロジェクトを実行する前に対象物の寸法を確認する必要があります。

Mech-Vision プロジェクトに、read_object_dimensions ステップを入れておきます。このステップのパラメータ **パラメータから対象物の寸法を読み込む** にチェックを入れる必要があります。



送信コマンド

パラメータ	アドレス (10 進数)
コマンド 501	1
Mech-Vision プロジェクト番号	4
[長さ、幅、高さ]	34

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

長さ、幅、高さ

Mech-Vision プロジェクトへ送信する対象物の寸法です。寸法の値は、read_object_dimensions ステップによって読み取られます。

単位はミリメートル（mm）です。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1108** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 502——Mech-Viz へ TCP を送信

このコマンドは、Mech-Viz プロジェクトにロボット TCP を動的に送信するためによく使われます。ロボット TCP を読み取るためのステップは outer_move です。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定します。サンプルプロジェクトのパスは Mech-Center のインストールディレクトリ `tool¥viz_project¥outer_move` です。

outer_move ステップをワークフローの適切な場所に配置します。

コマンド *201——Mech-Viz* プロジェクトを起動 を実行する前に、このコマンドを実行してください。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 502	1
TCP	40

TCP

outer_move ステップの経路点の TCP データを設定するために使用されます。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 901——ソフトウェアの起動状態を取得

このコマンドは、Mech-Vision、Mech-Viz、Mech-Center の起動状態を取得するために使用されます。

現在、このコマンドは Mech-Vision のみ対応できます。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 901	1

パラメータの説明：パラメータはありません。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

システムのセルフチェックステータスです。ステータスコードが **1101** になると「Mech-Vision が準備できました」です。他のステータスコードが出てきた場合「Mech-Vision プロジェクトがまだ準備できていません」ということです。現在、このコマンドは Mech-Vision プロジェクトの準備ができているかどうかを確認するためにのみ使用できます。

コマンド 999——レジスタデータを削除

このコマンドは、レジスタデータに保存されたデータを削除するために使用されます。

送信コマンド

パラメータ	アドレス (10 進数)
コマンド 999	1

パラメータの説明：パラメータはありません。

返されたデータ

パラメータ	アドレス (10 進数)
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**3103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

3.3.7 三菱 MC

Mech-Mind ソフトウェアシステムは、MC プロトコルに基づく標準インターフェースを介して三菱 PLC と通信することができます。

その他の関連マニュアルは現在作成中であり、後日公開する予定です。

レジスタの使用説明

PLC が使用するデータ型は MM_Interface 構造体で、728 個の D レジスタを占有しています。この構造体変数のベースアドレスは、Mech-Vision で設定されているベースアドレスと同じにする必要があります。PLC と Mech-Vision の両方で使用するベースアドレスが 10000 の場合、各変数のレジスタアドレスは下図のようになります。

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]		

下表は、各変数のレジスタのベースアドレスに対するオフセットとその説明です。

レジスタアドレスのオフセット	名称	データ型	説明
0	Command_Trigger	Word[符号付き]	トリガー信号
1	Command	Word[符号付き]	コマンドコード
2	Pose_Type	Word[符号付き]	位置姿勢タイプ
3	Pose_Number	Word[符号付き]	ビジョンポイントの期待数
4	Vision_Project_Num	Word[符号付き]	Mech-Vision プロジェクト番号
5	Recipe_Num	Word[符号付き]	Mech-Vision レシピ番号
6	Joint_Position	Float[単精度][6]	関節角度データ
18	TCP_Pose	Float[単精度][6]	フランジ位置姿勢データ
30	Branch_Name	Word[符号付き]	Mech-Viz の「メッセージによって異なる分岐を実行」ステップ ID
31	Branch_Exit_Port	Word[符号付き]	Mech-Viz の「メッセージによって異なる分岐を実行」ステップの出口番号
32	Index_Name	Word[符号付き]	Mech-Viz のインデックス付きステップのステップ ID
33	Index_Counter	Word[符号付き]	次にこのステップが実行されたときに設定されるべきインデックス値
34	Ext_Input-BoxDim	Float[単精度][3]	Mech-Vision プロジェクトに入力される対象物寸法（長さ、幅、高さ、ミリメートル単位）
40	Ext_Input_Pose	Float[単精度][6]	Mech-Viz プロジェクトに入力される外部 TCP データ
52	Robot_Move_Status	Word[符号付き]	ロボットの動作状態
53	Reserved	Word[符号付き][44]	予約語
97	Trigger_Acknowledge	Word[符号付き]	実行確認
98	Notify	Word[符号付き]	カスタマイズされたメッセージ
99	Heartbeat	Word[符号付き]	ハートビート
100	Status_Code	Word[符号付き]	ステータスコード
101	Status_of_Pose_Sent	Word[符号付き]	位置姿勢の送信状態
102	Number_of_Pose_Sent	Word[符号付き]	送信した位置姿勢の数
103	Index_of_Vision_Picking_Point	Word[符号付き]	「ビジョン処理による移動」の位置
104	Target_Pose	Float[単精度][240]	目標位置姿勢
584	Target_Label	Word[符号付き][40]	ラベル
624	Speed_Percentage	Word[符号付き][40]	速度
664	Digital_Output	Word[符号付き][40]	デジタル出力信号

コマンド説明

本節では、MC プロトコルを基づいた標準インターフェースのコマンドについて説明します。

コマンド 101——Mech-Vision プロジェクトを起動

このコマンドは、Mech-Vision のみ使用し、Mech-Viz を使用しない場合に使われます。このコマンドは、Mech-Vision のプロジェクトを起動し、カメラの撮影およびビジョン認識を行う場合に使われます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 101	1
Mech-Vision プロジェクト番号	4
ビジョンポイントの期待数	3
ロボット位置姿勢のタイプ	2
ロボット位置姿勢	6-17（JPs）または 18-29（フランジ位置姿勢）

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

ビジョンポイントの期待数

Mech-Vision から取得したいビジョンポイントの数です。ビジョンポイントの情報に、ビジョン位置姿勢及びそれに対応する点群、ラベル、スケーリングなどの情報が含まれています。

- `0`：Mech-Vision プロジェクトで認識できたすべてのビジョンポイントを取得します。
- `0` より大きな整数：Mech-Vision プロジェクトで認識できた指定数のビジョンポイントを取得します。
  - このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より大きい場合、認識結果にあるすべてのビジョンポイントを取得します。

– このパラメータの値が Mech-Vision で認識されたビジョンポイントの合計数より小さい場合、このパラメータで指定された数のビジョンポイントを取得します。

ヒント: ビジョンポイントを取得するためのコマンドは 102 コマンドです。デフォルトの設定環境において、このコマンドで一度に最大 20 個のビジョンポイントしか取得できないため、必要なビジョンポイントの数が 20 個より多い場合、このコマンド繰り返して実行する必要があります。

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Vision に送信するタイプを設定します。パラメータ範囲は 0~3 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**設定の説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Vision にロボットの位置姿勢を送信する必要がありません。	プロジェクトは、Eye To Hand モードです。Mech-Vision プロジェクトで「経路計画」ステップを使用する場合、経路計画の開始位置は、経路計画ツールで設定した初期位置です。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Vision にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。この設定は、直行ロボット以外のほとんどのロボットで利用可能です。
2	ロボットの現在のフランジ位置姿勢	Mech-Vision にロボットの現在のフランジ位置姿勢を送信する必要があります。	プロジェクトは、Eye In Hand モードです。ロボットは関節角度のデータを持たず、フランジ位置姿勢データのみを持ちます（直行ロボットの場合など）。
3	ロボット経路計画の開始位置の関節角度	Mech-Vision にロボット経路計画の開始位置の関節角度を送信する必要があります。	プロジェクトは、Eye To Hand モードです。また、Mech-Vision プロジェクトに「経路計画」ステップがあり、ロボット側から「経路計画」ステップの開始位置を設定する必要があります。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 102——Mech-Vision のビジョン目標点を取得

コマンド「Mech-Vision プロジェクトを起動」の後に使用します。このコマンドは、Mech-Vision からビジョンポイントを取得し、それをビジョン目標点に変換するために使用されます。

以下に、ビジョンポイントに含まれる位置姿勢をロボット TCP に変換する処理を示します。

- ビジョンポイントに含まれる位置姿勢を Y 軸を中心に 180° 回転させます。
- ロボット型番の基準座標系の定義によって、ロボットベースの高さが含まれているかどうかを判断し、それに応じて垂直方向のオフセットを増やします。

ヒント: デフォルトでは、102 コマンドは毎回最大 20 個までのビジョン目標点を取得することができます。20 個以上のビジョン目標点を取得するには、すべてのビジョン目標点を得るまで、102 コマンドを繰り返し実行してください。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 102	1
Mech-Vision プロジェクト番号	4

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100
データ転送状態	101
ビジョン目標点の数	102
予約語	/
今回取得したすべての TCP	104
今回取得したすべてのラベル	584

ステータスコード

コマンドが正常に実行された場合、**1100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

このコマンドを呼び出す時、Mech-Vision の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しいビジョン目標点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョン目標点であることを示し、それを読み取ります。

> **注意:** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

ビジョン目標点の数

このコマンドを実行して、取得したビジョン目標点の数です。

- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョンポイントの数よりも多い場合、Mech-Vision によって認識されたビジョンポイントの数に従って送信されます。
- リクエストしたビジョン目標点の数は Mech-Vision によって認識されたビジョンポイントの数よりも少ない場合、リクエストした数に従って送信されます。

予約語

この予約語が使われていないため、初期値は 0 です。

今回取得したすべての TCP

1 つの TCP には、空間座標 (XYZ) とオイラー角 (ABC) の情報が含まれます。

今回取得したすべてのラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に「ラベルマッピング」ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

103 コマンド—— Mech-Vision のパラメータレシピの切り替え

このコマンドは、Mech-Vision プロジェクトに使われるパラメータレシピを切り替える際に使用されます。コマンド「Mech-Vision プロジェクトを起動」を実行する前に、このコマンドを使用する必要があります。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 103	1
Mech-Vision プロジェクト番号	4
レシピ番号	5

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

レシピ番号

Mech-Vision プロジェクトの整数のレシピ番号です。**プロジェクトアシスタント・パラメータレシピ** をクリックしてレシピエディタに入り、レシピ番号を確認できます。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 105——Mech-Vision の「経路計画」ステップの結果を取得

コマンド「Mech-Vision プロジェクトを起動」を呼び出した後、このコマンドを使用して、Mech-Vision の「経路計画」ステップから出力される衝突のない経路を取得します。

このコマンドを使用する時、Mech-Vision の「出力」ステップの **ポートタイプ** を「事前定義済み（ロボット経路）」に設定する必要があります。

ヒント： 105 コマンドを呼び出す前に、105 コマンドの呼び出し回数を減らすように 101 コマンドの **ビジョンポイントの期待数** を 0 に設定する必要があります。101 コマンドの **ビジョンポイントの期待数** を 1 に設定すると、105 コマンドの呼び出しごとに 1 つの経路点のみが返され、105 コマンドを複数回呼び出した場合にのみすべての経路点が返されます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 105	1
Mech-Vision プロジェクト番号	4
経路点の位置姿勢タイプ	2

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

経路点の位置姿勢タイプ

このパラメータは、「経路計画」ステップから返された経路点の位置姿勢タイプを指定するために使用されます。

- 1：経路点の位置姿勢は、ロボットの関節角度（JPs）の形式で返されます。
- 2：経路点の位置姿勢は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100
データ転送状態	101
経路点の数	102
「ビジョン処理による移動」の位置	103
今回送信したすべての経路点の位置姿勢	104
今回送信したすべての経路点のラベル	584
今回送信したすべての経路点の速度	624

ステータスコード

コマンドが正常に実行された場合、**1103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

1：返されたデータが新しい経路点であることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、このコマンドを実行した後に返された経路点の数を表します。範囲は 0~20 です。20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

経路計画設定ツールで設定された「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 ->移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 105 送信時の **経路点の位置姿勢タイプ** によって、三次元座標、XYZ オイラー角、あるいは関節角度（JPs）で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に ラベルマッピング ステップを使用してラベルを整数にマッピングする必要があります。Mech-Vision のプロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

経路計画設定ツールで設定された速度値です。

コマンド 201——Mech-Viz プロジェクトを起動

このコマンドは、Mech-Vision と Mech-Viz の両方を使用する場合に使われます。Mech-Viz プロジェクトを実行し、対応する Mech-Vision プロジェクトを呼び出し、ロボットの動作経路を計画する時に使用されます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 201	1
ロボット位置姿勢のタイプ	2
ロボット位置姿勢	6-17（JPs）または 18-29（フランジ位置姿勢）

ロボット位置姿勢のタイプ 、ロボット位置姿勢

- **ロボット位置姿勢のタイプ** パラメータは、ロボット実機の位置姿勢を Mech-Viz に送信するタイプを設定します。パラメータ範囲は 0~2 です。
- **ロボット位置姿勢** のパラメータ値は、**ロボット位置姿勢のタイプ** のパラメータ値によって異なります。

2 つのパラメータの値と関係と説明は以下の通りです。

ロボット位置姿勢のタイプ パラメータ	**ロボット位置姿勢** パラメータ	**設定の説明**	**適用シーン**
0	0, 0, 0, 0, 0, 0	Mech-Viz にロボットの位置姿勢を送信する必要がありません。Mech-Viz での仮想ロボットは初期位置姿勢（JPs = [0, 0, 0, 0, 0, 0] ）から最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨しません。
1	ロボットの現在の関節角度とフランジ位置姿勢	Mech-Viz にロボットの現在の関節角度とフランジ位置姿勢を送信する必要があり、Mech-Viz での仮想ロボットは受信された位置姿勢から最初の経路点に移動します。	プロジェクトは Eye In Hand モードである場合は、この設定は推奨します。
2	ロボット側でカスタマイズされた関節角度	Mech-Viz にロボットのティーチポイント（現在の関節角度ではない）を送信する必要があります。これは、ロボットが画像撮影領域の外にいるとき（下図に示す）、Mech-Viz プロジェクトが次回の経路を事前に計画することをトリガーするために使用されます。Mech-Viz での仮想ロボットは受信された最初のティーチポイントから最初の経路点に移動します。	プロジェクトは Eye To Hand モードである場合は、この設定は推奨します。

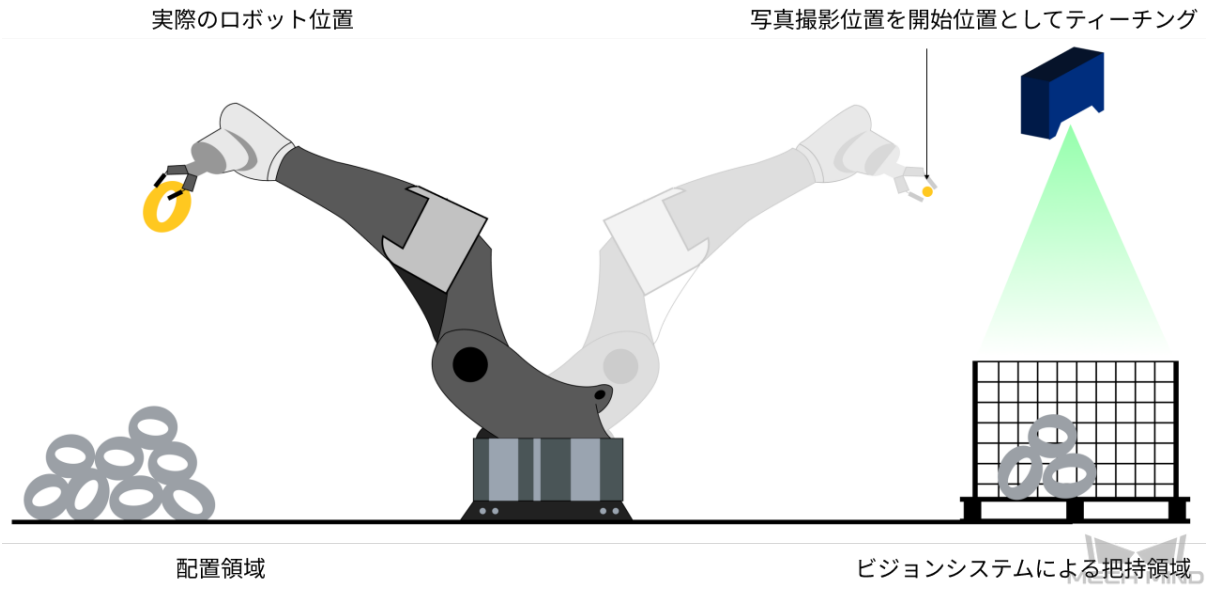
Eye To Hand モードでは、**ロボット位置姿勢のタイプ** が 2 に設定されている理由は以下の通りです。

> Eye To Hand モードでは、カメラはロボットが画像撮影領域と把持領域に戻る前に撮影し、次回の把持経路を計画することができます。これにより、タクトタイムの向上を実現します。

この時、**ロボット位置姿勢のタイプ** が 1 に設定され、つまり現在の位置姿勢を Mech-Viz に送信すれば、仮想ロボットがロボット実機の経路と一致しない可能性があります。また、未知の衝突が発生する可能性もあります。

仮想ロボットは現在の位置姿勢から Mech-Viz での最初の移動ステップで設定された位置姿勢に移動しますが、ロボット実機は上記の位置姿勢に移動する前に別の位置姿勢に移動する可能性があるということです。

したがって、**ロボット位置姿勢のタイプ** パラメータを 2 に設定する必要があります。



返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2103** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 202——Mech-Viz プロジェクトを停止

Mech-Viz プロジェクトの実行を停止します。Mech-Viz プロジェクトは無限ループになっていない場合や、正常に停止できる場合は、このコマンドを使用する必要がありません。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 202	1

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2104** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 203——Mech-Viz 分岐を選択

Mech-Viz プロジェクトに「メッセージによって異なる分岐を実行」ステップがある場合、このコマンドは Mech-Viz プロジェクトにおける「メッセージによって異なる分岐を実行」ステップの出口番号を指定します。このコマンドを実行する前に、コマンド「Mech-Viz プロジェクトを起動」を実行する必要があります。Mech-Viz は「メッセージによって異なる分岐を実行」ステップまで実行すると、このコマンドで送られる分岐出口番号を待ちます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 203	1
分岐ステップ ID	30
分岐の出口番号	31

分岐ステップ ID

「メッセージによって異なる分岐を実行」のステップ ID で、正の整数です。

出口番号

「メッセージによって異なる分岐を実行」の出口番号で、正の整数です。

出口番号は、1 から始まるポートのインデックス番号です。例えば、指定された出口が左から右に 2 番目のポートである場合、番号は 2 です。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2105** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 204——移動インデックスを設定

このコマンドはステップのインデックスパラメータを設定するために使用されます。インデックスパラメータが付いたステップは「リストによる移動」、「グリッドによる移動」、「カスタマイズのパレットパターン」、「事前計画したパレットパターン」などです。通常、連続した、あるいは個別に指定された移動などの操作に使用されます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 204	1
ステップ ID	32
インデックス☒	33

ステップ ID

インデックス付きステップのステップ ID で、正の整数である必要があります。

インデックス値

次にこのステップが実行されたときに設定されるべきインデックス値です。

このコマンドを送信すると、Mech-Viz の現在のインデックス値がこのパラメータの値から 1 を引いた値に変更されます。

このコマンドで指定したステップまで Mech-Viz プロジェクトが実行されると、Mech-Viz の現在のインデックス値がこのパラメータの値まで 1 ずつ増加します。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2106** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 205——Mech-Viz の計画経路を取得

コマンド「Mech-Viz プロジェクトを起動」を実行した後、このコマンドは Mech-Viz が計画した経路を取得するために使用されます。

初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

注意： プロジェクト内の移動ステップの経路点をロボットに送信しない場合は、ステップパラメータで「移動目標点を送信」のチェックを外してください。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 205	1
経路点タイプ	2

経路点タイプ

このパラメータは、Mech-Viz から返された経路点の位置姿勢タイプを指定するために使用されます。

- 1：経路点は、ロボットの関節角度（JPs）の形式で返されます。
- 2：経路点は、ロボットのツール位置姿勢（TCP）の形式で返されます。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100
データ転送状態	101
経路点の数	102
「ビジョン処理による移動」の位置	103
今回送信したすべての経路点の位置姿勢	104
今回送信したすべての経路点のラベル	584
今回送信したすべての経路点の速度	624

ステータスコード

コマンドが正常に実行された場合、**2100** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

注意： このコマンドを呼び出す時、Mech-Viz の結果が返されていない場合、デフォルトで 10 秒間待機します。タイムアウトになった場合、タイムアウトエラーを表すステータスコードが返されます。

データ転送状態

このパラメータは、返されたデータは新しい経路点であるかどうかを表示するために使用されます。

1：返されたデータが新しいビジョンポイントであることを示し、それを読み取ります。

> **注意：** 新しく返されたデータを読み取った後、このパラメータを 0 にリセットします。

経路点の数

このパラメータは、返された経路点の数を表します。範囲は 0~20 です。初期設定を使用する場合、このコマンドを一回実行すると最大 20 個の計画された経路点を取得できます。したがって、20 以上の経路点を取得するには、このコマンドを繰り返してください。

「ビジョン処理による移動」の位置

「ビジョン処理による移動」の経路点が経路全体における位置です。

例えば、経路計画は移動_1 -> 移動_2 -> ビジョン処理による移動 -> 移動_3 のステップで構成されている場合、「ビジョン処理による移動」の位置は 3 です。

「ビジョン処理による移動」がなければ、このパラメータは 0 です。

今回送信したすべての経路点の位置姿勢

フォーマットはコマンド 205 送信時のパラメータによって、三次元座標、オイラー角、あるいは関節角度で記述されます。

今回送信したすべての経路点のラベル

位置姿勢に対応する整数のラベルです。Mech-Vision プロジェクトでラベルは文字列タイプであり、出力する前に「ラベルマッピング」ステップを使用してラベルを整数にマッピングする必要があります。プロジェクトにラベルが含まれていない場合、このパラメータの初期値は 0 です。

今回送信したすべての経路点の速度

移動ステップパラメータに設定されたゼロでないパーセンテージ値です。

コマンド 206——DO リストを取得

複数のロボットハンドまたは吸盤パーティションを制御する場合、このコマンドを用いて DO リストを取得します。このコマンドを実行する前に、コマンド「Mech-Viz によって計画された経路を取得」を実行する必要があります。

プロジェクトの「DO リストを設定」のステップパラメータで以下の設定を行います。

- 「受信者」を「標準インターフェース」に設定します
- 「ビジョン処理による移動から DO リストを取得」にチェックを入れます
- DO リストが必要な「ビジョン処理による移動」ステップを選択します。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 206	1

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100
DO 信号リスト	664

ステータスコード

コマンドが正常に実行された場合、**2102** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

DO 信号リスト

このコマンドは 64 個の DO ポート番号を返します。有効なポート番号は 0 から 999 の範囲の正の整数で、無効なポート番号は-1（プレースホルダーとして）です。

例えば、下表の場合、返される有効な DO ポート番号は 1、3、5、6 であり、上記ポート番号に対応する値が 1 に設定されることを意味します。

1	3	5	6	-1	-1	-1	-1	...	-1	-1
1 番目の整数	2 番目の整数	3 番目の整数	4 番目の整数	5 番目の整数	6 番目の整数	7 番目の整数	8 番目の整数	...	63 番目の整数	64 番目の整数

コマンド 501——Mech-Vision プロジェクトへ対象物の寸法を送信

このパラメータは Mech-Vision プロジェクトに、対象物の寸法を動的に送信すること場合に使われます。Mech-Vision プロジェクトを実行する前に対象物の寸法を確認する必要があります。

Mech-Vision プロジェクトに「対象物の寸法を読み込む」ステップがあり、**パラメータから対象物の寸法を読み込む** にチェックを入れる必要があります。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 501	1
Mech-Vision プロジェクト番号	4
[長さ、幅、高さ]	34

Mech-Vision プロジェクト番号

Mech-Vision のプロジェクト番号は、Mech-Vision のプロジェクトリストで確認できます。プロジェクト名の前の数字は、プロジェクト番号を表します。

長さ，幅，高さ

Mech-Vision プロジェクトへ送信された対象物の寸法が「対象物の寸法を読み取る」ステップで読み取られます。

単位はミリメートル（mm）です。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**1108** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 502——Mech-Viz へ TCP を送信

このコマンドは、Mech-Viz プロジェクトにロボット TCP を動的に送信するために使用されます。ロボット TCP を読み取るためのステップは outer_move です。このコマンドは、コマンド「Mech-Viz プロジェクトを起動」を実行する前に実行する必要があります。

サンプルプロジェクトに基づいて Mech-Viz プロジェクトを設定してください。サンプルプロジェクトは、Mech-Mind ソフトウェアシステムのインストールディレクトリ `tool/viz_project/outer_move` に格納されています。「外部移動」ステップをワークフロー内の適切な場所に配置してください。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 502	1
TCP	40

TCP

「外部移動」ステップの経路点を設定するための TCP データ。

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

コマンドが正常に実行された場合、**2107** のステータスコードが返されます。エラーが発生した場合、エラーを表すエラーコードが返されます。

コマンド 901——ソフトウェアの起動状態を取得

このコマンドは、Mech-Vision、Mech-Viz、Mech-Center の起動状態を取得するために使用されます。現在、このコマンドは Mech-Vision プロジェクトの準備ができているかどうかを確認するためにのみ使用できます。

送信コマンド

パラメータ	レジスタアドレスのオフセット
コマンド 901	1

返されたデータ

パラメータ	レジスタアドレスのオフセット
ステータスコード	100

ステータスコード

システムのセルフチェックステータスです。ステータスコードが **1101** になると「Mech-Vision が準備できました」です。他のステータスコードが出てきた場合「Mech-Vision プロジェクトがまだ準備できていません」ということです。

3.3.8 付録

Mech-Viz による衝突検出

`XXX/Mech-Center/tool/viz_project/check_collision` フォルダの check_collision.viz プロジェクトを使用します。以下のことに注意してください。

1. check_collision はサンプルプロジェクトです。移動に関するステップを除くすべてのステップは必須であり、削除や位置の変更はできません。ロボットモデルで実際に使用されている型番を選択してください。
2. 移動関連のステップクは、追加や削除が可能です。送信される位置姿勢の数は、移動に関連するステップの数と同じです。
3. 初期位置が必要な場合は、ロボット側からカメラで撮像するコマンドを呼び出す前に、位置姿勢を設定するコマンドを一度呼び出します。

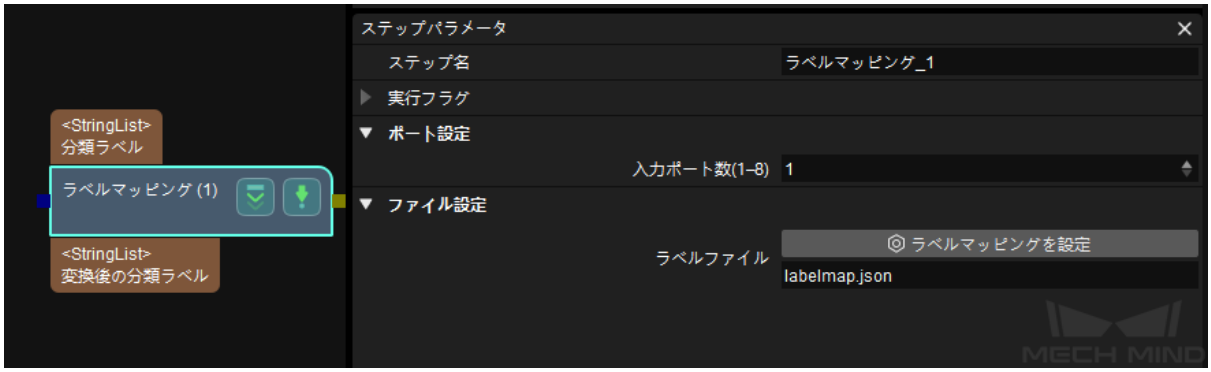
Mech-Viz の吸盤コンフィグレータ

`XXX/Mech-Center/tool/viz_project/suction_zone` フォルダの suction_zone.viz プロジェクトを使用します。以下のことに注意してください。

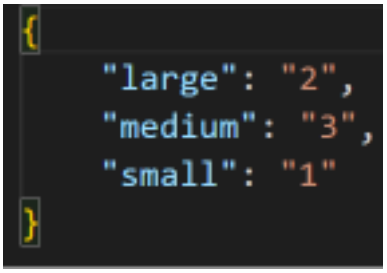
1. suction_zone.viz はサンプルプロジェクトです。移動に関するステップを除くすべてのステップは必須であり、削除や位置の変更はできません。ロボットモデルで実際に使用されている型番を選択してください。
2. 移動関連のステップクは、追加や削除が可能です。送信される位置姿勢の数は、移動に関連するステップの数と同じです。
3. 初期位置が必要な場合は、ロボット側からカメラで撮像するコマンドを呼び出す前に、位置姿勢を設定するコマンドを一度呼び出します。
4. 実行する前に吸盤コンフィグレータを設定してください。
5. ロボット側では、まずカメラで撮像するコマンドを呼び出し、次に DO 信号を取得するコマンドを呼び出す必要があります。

対象物のラベルを送信

ロボットに送信されるラベルは整数のラベルコードです。ラベルのマッピングは、Mech-Vision プロジェクトで作成する必要があります。

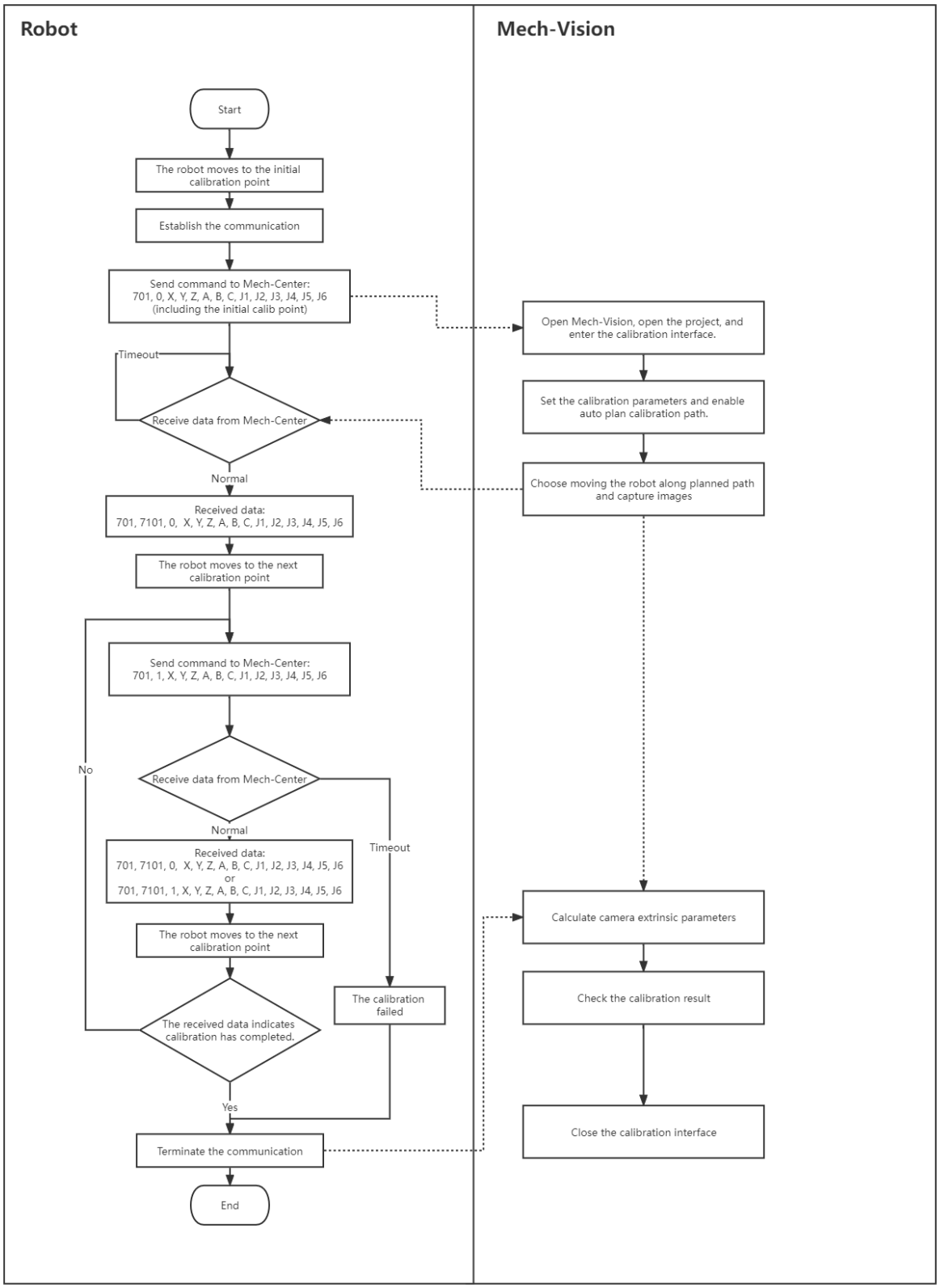


ラベルマッピングのフォーマットは以下の通りです。



- large 、small、medium はラベルの文字列です。
- 1、2、3 はラベルコードです。

ロボット自動キャリブレーションの流れ



カメラの露出完了の信号を追加

PROFINET および EtherNet/IP プロトコルの標準インターフェースには、タクトタイムを改善するために使用される **カメラ露出完了** 信号があります。Mech-Vision プロジェクトの実行時間が長い場合、カメラ露光終了後すぐにロボットの位置を移動させる必要があります。**カメラ露出完了** 信号を作成するには、Mech-Vision プロジェクトにいくつかの設定が必要です。

1. Mech-Vision プロジェクトに notify_vision を追加し、capture_images_from_camera の制御フローにつなぎます。capture_images_from_camera の **実行フラグ ‣ 出力がある場合に制御フローをトリガー** にチェックを入れます。

2. notify_vision のサービス名を **Standard Interface Notify** に、メッセージ内容を **1001** に設定します（1001 のメッセージ内容は変更不可）。

3. Mech-Vision プロジェクトを実行すると、カメラの露光が終了すると、PLC/ロボット側に Exposure_Complete 信号が送られます。Exposure_Complete 信号を受信後、Reset_Exposure 信号で Exposure_Complete 信号をリセットしてください。10 秒以内にリセット信号を受信しないと、**Mech-Center データ確認信号タイムアウト** というエラーが報告されます。

3.3.9 標準インターフェースのステータスコード一覧とトラブルシューティング

概要

標準インターフェースを利用する場合、コマンドのリターンメッセージにはコマンドの実行状況（ステータスコード）が含まれます。ステータスコードには、正常に実行完了のコードとエラーコードがあります。

Mech-Vision：

- 1001~1099：Mech-Vision 関連のエラーコード
- 1100~1199：Mech-Vision 関連の正常実行完了のコード

Mech-Viz：

- 2001~2099：Mech-Viz 関連のエラーコード
- 2100~2199：Mech-Viz 関連の正常実行完了のコード

Mech-Center：

- 3001~3099：Mech-Center 関連のエラーコード
- 3100~3199：Mech-Center 関連の正常実行完了のコード

ロボット：

- 4001~4099：ロボット関連のエラーコード
- 4100~4199：ロボット関連の正常に実行完了のステータスコード

外部パラメータのキャリブレーション：

- 7001~7099：外部パラメータのキャリブレーション関連のエラーコード
- 7100~7199：外部パラメータのキャリブレーション関連の正常実行完了のコード

Mech-Vision

Mech-Vision 関連のエラーコード

エラーコード	意味
1001	Mech-Vision：プロジェクトが登録されていない、または自動読み込みに設定されていない
1002	Mech-Vision：ビジョン結果がない
1003	Mech-Vision：ROI 内に点群がない
1004	Mech-Vision：パラメータの設定に失敗
1005	Mech-Vision：Mech-Vision の起動時のコマンドパラメータが無効
1006	Mech-Vision：無効な位置姿勢データ
1007	Mech-Vision：計算中
1008	エラーコードが使用されていない
1009	Mech-Vision：位置姿勢の数は移動パラメータの数と一致しない
1010	Mech-Vision：位置姿勢の数はラベル数と一致しない
1011	Mech-Vision：プロジェクト番号が存在しない
1012	Mech-Vision：パラメータレシピの番号が上限を超えた
1013	Mech-Vision：レシピ名が存在しない
1014	Mech-Vision：パラメータの設定に失敗
1015	Mech-Vision：プロジェクト実行エラー
1016	Mech-Vision：ディープラーニングサーバーの起動に失敗
1017	Mech-Vision：無効なラベルマッピング
1018	Mech-Vision：ビジョン位置姿勢の数がエラー

次の

表 1 – 前のページからの続き

1019	Mech-Vision：実行時間タイムアウト
1020	Mech-Vision：実行していない
1021	Mech-Vision：対象物寸法の設定に失敗（プロジェクトにステップ read_object_dimensions があるかを確認
1022	Mech-Vision：無効な対象物寸法
1023	Mech-Vision：カメラの接続に失敗
1024	Mech-Vision：位置姿勢の数はカスタマイズされた出力データの数と一致しない
1025	Mech-Vision：仮想カメラが使用中で、データは放棄された
1026	Mech-Vision：無効な位置姿勢タイプ
1028	Mech-Viz：経路計画に失敗
1030	Mech-Vision：ロボットがビジョン位置姿勢に到達不能
1031	Mech-Viz：ロボット関節角度の計算に失敗
1032	Mech-Vision：経路に到達不能
1033	Mech-Vision：移動特異点
1034	Mech-Vision：直線運動の計画に失敗
1035	Mech-Vision：無効な把持位置姿勢
1036	Mech-Vision：ロボット本体が衝突
1037	Mech-Vision：ロボットと対象物との衝突
1038	Mech-Vision: 点群と衝突する点数がしきい値より大きく、衝突が検出された
1039	Mech-Vision: 点群と衝突する面積がしきい値より大きく、衝突が検出された
1040	Mech-Vision: 点群と衝突する体積がしきい値より大きく、衝突が検出された
1044	Mech-Vision：「ビジョン処理による移動」ステップがビジョン位置姿勢を送信していない
1046	Mech-Vision：無効なエンドツール

Mech-Vision 関連の正常実行完了のコード

正常実行完了のコード	意味
1100	Mech-Vision：ビジョン位置姿勢の取得に成功
1101	Mech-Vision：準備完了
1102	Mech-Vision：プロジェクトのトリガーに成功
1103	Mech-Vision：計画経路の取得成功（105 コマンド実行成功後の戻り値）
1107	Mech-Vision：レシピの切替に成功
1108	Mech-Vision：対象物寸法の設定に成功

Mech-Viz

Mech-Viz 関連のエラーコード

エラーコード	意味
2001	Mech-Viz：ソフトウェアが登録されていない
2002	Mech-Viz：プロジェクト実行中
2003	Mech-Viz：Mech-Vision からのビジョン結果を受信していない
2004	Mech-Viz：Mech-Vision からのビジョン結果に到達できない
2005	Mech-Viz：ロボット関節角度の計算に失敗
2006	Mech-Viz：Mech-Viz の起動時のコマンドパラメータが無効
2007	Mech-Viz：動作計画に失敗
2008	Mech-Viz：プロジェクト実行エラー
2009	Mech-Viz：ツール位置姿勢（TCP）が提供されていない
2010	Mech-Viz：経路に到達できない

表 2 – 前のページからの続き

2011	Mech-Viz：DO リストが提供されていない
2012	Mech-Viz：無効な位置姿勢タイプ
2013	Mech-Viz：無効な位置姿勢データ
2014	Mech-Viz：プロジェクトが設定されていない
2015	エラーコードが使用されていない
2016	Mech-Viz：ステップパラメータの設定に失敗
2017	Mech-Viz：停止に失敗
2018	Mech-Viz：無効な分岐の出口番号
2019	Mech-Viz：分岐の設定に失敗（分岐のステップ ID を確認してください）
2020	Mech-Viz：特異点のため実行に失敗
2021	Mech-Viz：直線運動の計画に失敗
2022	Mech-Viz：実行していない
2023	Mech-Viz：プロジェクトファイルが異常
2024	Mech-Viz：無効な分岐名
2025	Mech-Viz：実行時間タイムアウト
2026	Mech-Viz：無効なインデックス付きのステップ名
2027	Mech-Viz：無効なインデックス
2028	Mech-Viz：インデックスの設定に失敗（インデックパラメータ付きのステップがプロジェクトに入れるか
2029	Mech-Viz：外部移動の設定に失敗
2030	Mech-Viz：無効な把持位置姿勢
2031	Mech-Viz：ロボット本体が衝突
2032	Mech-Viz：シーンとの衝突
2033	Mech-Viz：点群と衝突する点の数が上限を超えた
2034	Mech-Viz：点群と衝突する面積が上限を超えた
2035	Mech-Viz：点群と衝突するボリュームが上限を超えた
2036	Mech-Viz：撮影されていない
2037	Mech-Viz：ビジョン結果がない
2038	Mech-Viz：ビジョン結果の ROI に点群がない
2039	Mech-Viz：計画するビジョン位置姿勢がない
2040	Mech-Viz：ビジョン結果を再利用するための計画経路が失敗
2041	Mech-Viz：ステップパラメータの取得に失敗
2042	Mech-Viz：「ビジョン処理による移動」からの動作計画の取得に失敗
2043	Mech-Viz：ビジョン結果からのカスタマイズされたデータの取得に失敗
2044	Mech-Viz：ビジョンサービスが登録されていない
2045	Mech-Viz：無効なエンドツール

Mech-Viz 関連の正常実行完了のコード

正常実行完了のコード	意味
2100	Mech-Viz：実行に成功
2101	Mech-Viz：実行停止に成功
2102	Mech-Viz：DO リスト送信に成功
2103	Mech-Viz：起動に成功
2104	Mech-Viz：停止に成功
2105	Mech-Viz：分岐の設定に成功
2106	Mech-Viz：インデックスの設定に成功
2107	Mech-Viz：外部からの位置姿勢の設定に成功
21072108	Mech-Viz：Mech-Viz ステップのパラメータの設定に成功
2109	Mech-Viz：Mech-Viz ステップのパラメータの読み取りに成功

Mech-Center

Mech-Center 関連のエラーコード

エラーコード	意味
3001	Mech-Center：無効なコマンド
3002	Mech-Center：インターフェースコマンド長さやフォーマットがエラー
3003	Mech-Center：クライアントが切断された
3004	Mech-Center：サーバーが切断された
3005	Mech-Center：Mech-Vision を呼び出す時間がタイムアウト
3006	Mech-Center：未知エラー
3007	Mech-Center：データ確認応答信号がタイムアウト
3008	Mech-Center：構成 ID が存在しないため、パラメータの読み取り/設定ができない

Mech-Center 関連の正常実行完了のコード

正常実行完了のコード	意味
3100	Mech-Center：クライアントとの接続は正常
3101	Mech-Center：サーバーとの接続は正常
3102	Mech-Center：クライアントからの接続を待つ
3103	Mech-Center：データキャッシュのクリアに成功

ロボット

ロボット関連のエラーコード

エラーコード	意味
4001	無効なロボットタイプ
4002	ロボットのオイラー角の形式に対応しない
4003	ロボットサービスが登録されていない
4004	ロボットパラメータが不完全
4005	ロボットに接続できませんでした（ロボットの IP アドレスとネットワーク構成を確認してください）
4006	ロボットのロードプログラムのバージョンが不一致（再ダウンロードしてください）

ロボット関連の正常実行完了のコード

正常実行完了のコード	意味
4100	ロボットサービスの登録に成功
4101	ロボットの接続に成功
4102	ロボットの接続が切断した
4103	ユーザーによってロボットサービスが終了された

外部パラメータのキャリブレーション

外部パラメータのキャリブレーション関連のエラーコード

エラーコード	意味
7001	キャリブレーション：パラメータがエラー
7002	キャリブレーション：Mech-Vision から提供されるキャリブレーションポイントがない
7003	キャリブレーション：ロボットはキャリブレーションポイントに到達することに失敗

外部パラメータのキャリブレーション関連の正常実行完了のコード

正常実行完了のコード	意味
7100	キャリブレーション：ロボットはキャリブレーションポイントに到達することに成功
7101	キャリブレーション：Mech-Vision から提供されるキャリブレーションポイントが正常

Mech-Vision 関連のエラーのトラブルシューティング

1001

Mech-Vision：プロジェクトが登録されていない、または自動読み込みに設定されていない

エラーの原因：

- Mech-Vision が起動していないか、プロジェクトが開かれません
- **現在のプロジェクトを自動的に読み込む** にチェックを入れていないため、プロジェクト ID はありません。
- 内部通信の異常、または同名のプロジェクトの存在により、Mech-Vision プロジェクトのサービス登録が失敗しました。

トラブルシューティング手順：

- Mech-Vision ソフトウェアが起動していること、呼び出し中のプロジェクトが開いていることを確認します。
- **現在のプロジェクトを自動的に読み込む** にチェックを入れていることを確認します。
- 同じ名前のプロジェクトが存在しないことを確認し、Mech-Vision ソフトウェアを再起動してください。

1002

Mech-Vision：ビジョン結果がない

エラーの原因：

- 呼び出された Mech-Vision が正常に実行されましたが、poses のポートにはデータがありません。考えられる理由は、インスタンスセグメンテーションの信頼度しきい値が高すぎる、シーンにマッチングする物体がない、不適切な ROI 設定、点群精度が出ない、不適切なフィルタリング設定などがあります。

- Mech-Vision プロジェクトの出力ステップで、poses ポートが存在しません。考えられる理由は、出力ステップのポートタイプがカスタムに設定されているが、poses ポート名が作成されていません。

トラブルシューティング手順：

- ステップ procedure_out の poses ポートから上に Mech-Vision プロジェクトのデータフローを確認します。
- procedure_out ステップのポートタイプとポート名を確認します。

1003

Mech-Vision：ROI 内に点群がない

エラーの原因：

- 呼び出された Mech-Vision プロジェクトが正常に位実行されましたが、3D ROI に点群がありません。

トラブルシューティング手順：

- 点群に 3D ROI を設定するステップの ROI 設定を確認します。一部のプロジェクトには、このエラーによって箱が位置に到達しているか、箱は空であるかなどが判断できるので、必ずしもプロジェクトエラーとは限りません。

1004

Mech-Vision：パラメータの設定に失敗

1005

Mech-Vision：Mech-Vision の起動時のコマンドパラメータが無効

エラーの原因：

- クライアントが 101 コマンドを呼び出して Mech-Vision プロジェクトをトリガーするとき、ロボット位置姿勢のタイプのパラメータ値が正しく設定されていませんでした。ロボット位置姿勢のタイプのパラメータ値は 0〜3 の範囲です。

トラブルシューティング手順：

- クライアントのインターフェースプログラムで、101 コマンドで設定したロボット位置姿勢のタイプのパラメータ値が正しいかを確認します。

1006

Mech-Vision：無効な位置姿勢データ

エラーの原因：

- 101 コマンドよって設定されたロボットの位置姿勢データは、6 桁未満です。ロボットの位置姿勢はデフォルトで 6 軸ロボットの位置姿勢になります。4 軸または 5 軸ロボットの場合、残りのフィールドに 0 を入力します。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認します。101 コマンドで送信されるロボットの位置姿勢データは、6 つの数字で構成されている必要があります。

1007

Mech-Vision：計算中

エラーの原因：

- Mech-Vision プロジェクトがまだ実行されている間、クライアントのインターフェースプログラムは 101 コマンドを再度呼び出して、同じ Mech-Vision プロジェクトをトリガーしようとします。
- Mech-Vision では複数のプロジェクトを同時に実行できますが、実行中に同じ Mech-Vision プロジェクトを繰り返しトリガーすることはできません。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、設定された Mech-Vision プロジェクト番号が正しいかを確認します。
- プログラムに同じ Mech-Vision プロジェクトが短期間で再度呼び出されるかを確認します。

1008

エラーコードが使用されていない

1009

Mech-Vision：位置姿勢の数は移動パラメータの数と一致しない

エラーの原因：

- このエラーは通常、ビジョン結果がロボットの移動経路（接着など）であり、Mech-Vision プロジェクトによって出力される移動 パラメータが経路上の目標点の数と一致しないプロジェクトで発生します。

トラブルシューティング手順：

まだありません。

1010

Mech-Vision：位置姿勢の数はラベル数と一致しない

エラーの原因：

- Mech-Vision プロジェクトによって出力されたラベルがビジョン位置姿勢の数と一致しません。

トラブルシューティング手順：

- Mech-Vision プロジェクトにデータフローの位置姿勢とラベルを確認し、位置姿勢とラベル数の不一致の原因をトラブルシューティングします。

1011

Mech-Vision：プロジェクト番号が存在しない

エラーの原因：

- 101 コマンドに設定したプロジェクト番号が Mech-Vision のプロジェクトリストに存在しません。
  - 例えば、プロジェクトリストに 1 つのプロジェクトしかない場合、101 コマンドが 2 番目のプロジェクトを呼び出すと、このエラーが発生します。

トラブルシューティング手順：

- 項目で使用されるすべての Mech-Vision プロジェクトで **現在のプロジェクトを自動的に読み込む** が設定されていることを確認します。
- クライアントのインターフェースプログラムでトリガーされた Mech-Vision プロジェクト番号が正しいかどうかを確認します。

1012

Mech-Vision：パラメータレシピの番号が上限を超えた

エラーの原因：

- クライアントのインターフェースプログラムは 103 コマンドを呼び出して Mech-Vision プロジェクトレシピを設定しますが、呼び出されたレシピ番号には対応するレシピがありません。
  - 例えば、Mech-Vision プロジェクトで設定されたレシピが 2 つしかない場合、クライアントが 3 番目のレシピを呼び出すと、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Vision プロジェクトのレシピ設定を確認します。
- クライアントのインターフェースプログラムの 103 コマンドで設定したレシピ番号が正しいかを確認します。

1013

Mech-Vision：レシピ名が存在しない

エラーの原因：

- クライアントのインターフェースプログラムが 103 コマンドを呼び出して Mech-Vision プロジェクトのレシピを設定したが、Mech-Vision プロジェクトにレシピが設定されていない場合、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Vision プロジェクトのレシピ設定を確認し、プロジェクトで必要なレシピと番号が正しく設定されていることを確認します。
- プロジェクトがレシピを切り替える必要がない場合、クライアントのインターフェースプログラムはレシピを切り替える機能を使用しません。

1014

Mech-Vision：パラメータの設定に失敗

エラーの原因：

- Mech-Center と Mech-Vision ソフトウェア間の通信が異常で、Mech-Vision レシピが正常に設定されません。

トラブルシューティング手順：

- Mech-Vision のログで詳細情報を確認してください。

1015

Mech-Vision：プロジェクト実行エラー

エラーの原因：

- Mech-Vision プロジェクトが正常に実行されず、Mech-Vision エラーコード CV-Exxxx または Mech-Center によって解析されないその他のエラーが表示されます。このエラーが発生すると、Mech-Vision プロジェクトは実行を完了せずに中止されます。

トラブルシューティング手順：

- Mech-Vision でエラーメッセージを確認し、エラーメッセージに基づいて Mech-Vision プロジェクトのトラブルシューティングを行います。

1016

Mech-Vision：ディープラーニングサーバーの起動に失敗

エラーの原因：

- トリガーされた Mech-Vision プロジェクトにはディープラーニングモジュールが含まれていますが、ディープラーニングサーバーは起動されていません。対応する Mech-Vision エラーコードは「DL-E0201 ディープラーニングサーバーが起動されていません」です。

トラブルシューティング手順：

- Mech-Vision プロジェクトが初めて実行されていること、およびモデルの読み込みに時間がかかりすぎていないことを確認します。モデルのプリロードを設定する必要がある場合は、ディープラーニングステップのパラメータで **プロジェクトを開くとモデルを自動プリロード** にチェックを入れてください。
- ディープラーニング環境が正しくインストールされていることを確認します。

1017

Mech-Vision：無効なラベルマッピング

エラーの原因：

- Mech-Vision プロジェクトによって出力されたラベルは、INT データ型ではありません。標準インターフェースを使用する場合、Mech-Vision プロジェクトによって出力されたラベルは INT データ型にマッピングする必要があります。そうしないと、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Vision プロジェクトのデータフローを確認します。ステップ procedure_out への入力ラベルが正の整数でない場合は、ステップ label_mapping を使用してラベルを正の整数にマッピングします。

1018

Mech-Vision：ビジョン位置姿勢の数がエラー

エラーの原因：

- クライアントのインターフェースプログラムは 101 コマンドを呼び出して Mech-Vision プロジェクトをトリガーし、設定されたビジョン位置姿勢の期待数がインターフェースのデータ長さ制限を超えています。

トラブルシューティング手順：

- データ長さが Mech-Vision のツールバーで **ロボット通信設定** で設定可能です。通信画面の **詳細設定** で一度に送信するビジョン位置姿勢の最大数を設定します。設定可能な範囲は [1, 30] です。
- パラメータ「ビジョン位置姿勢の期待数」が上記で設定した値よりも小さいことを確認します。

1019

Mech-Vision：実行時間タイムアウト

エラーの原因：

- 102 コマンドを呼び出して Mech-Vision からビジョン結果を取得することからタイマーを開始し、指定された時間内に Mech-Vision プロジェクトの実行が完了しなかった場合、このエラーが報告されます。
- Mech-Vision のツールバーで **ロボット通信設定** をクリックし、タイムアウト時間が **詳細設定** で設定可能です。初期値は 10s です。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、102 コマンドを呼び出して Mech-Vision のビジョン結果を取得する前にタイムアウト時間を適切に追加できます。
- Mech-Vision プロジェクトの実行時間が長いプロジェクトの場合、上記のタイムアウト制限時間の設定を適切に変更できます。

1020

Mech-Vision：実行していない

エラーの原因：

- クライアントのインターフェースプログラムが最初に 101 コマンドを呼び出して Mech-Vision プロジェクトをトリガーせず、このプロジェクトのビジョン結果を取得しようとして 102 コマンドを直接呼び出すと、このエラーが報告されます。
  - 例えば、2 つの Mech-Vision プロジェクトが Mech-Center に登録されている場合、クライアントのインターフェースプログラムがプロジェクト 1 を開始してからプロジェクト 2 のビジョン結果を取得しようとすると、このエラーが報告されます。
- コマンド 102 が呼び出されてビジョン結果のすべての位置姿勢が取得されました。キャッシュは空ですが、コマンド 102 の呼び出しを続けます。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、102 コマンドによって指定されたプロジェクト番号が正しいであることを確認します。
- クライアントインターフェースプログラムを確認します。102 コマンドによって返されたデータのパラメータ「位置姿勢の送信が完了するかどうか」が 1 の場合、全ての位置姿勢送信が完了したことを意味し、102 コマンドを呼び出し続けると本エラーとなります。

1021

Mech-Vision：対象物寸法の設定に失敗（プロジェクトにステップ read_object_dimensions があるかを確認してください）

エラーの原因：

- クライアントのインターフェースプログラムは 501 コマンドを呼び出して Mech-Vision プロジェクトの対象物寸法を動的に設定しますが、Mech-Vision プロジェクトには read_object_dimensions ステップがなく、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Vision プロジェクトにステップ read_object_dimensions があることを確認します。

1022

Mech-Vision：無効な対象物寸法

エラーの原因：

- クライアントのインターフェースプログラムは 501 コマンドを呼び出して Mech-Vision プロジェクトに対象物の寸法を動的に設定しますが、設定された寸法の値は無効であり、0 たは負の数値があります。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、設定された対象物の寸法（長さ/幅/高さ）は正の実数である必要があります。

1023

Mech-Vision：カメラの接続に失敗

エラーの原因：

- クライアントインターフェースプログラムは Mech-Vision プロジェクトをトリガーして画像を撮影し、カメラが切断されます。対応するエラーコードは CV-E0201 です。

トラブルシューティング手順：

- ネットワーク接続を確認します。
- カメラと IPC の IP アドレスが同じネットワークセグメントにあるかどうかを確認します。

  （上記の *2* つは、コマンドプロンプトで「*ping*」とカメラの *IP* アドレスを入力することで実行できます。）
- カメラの電源、IPC のファイアウォールの設定などを確認します。
- 問題をトラブルシューティングできない場合は、Mech-Mind サポートにお問い合わせください。

1024

Mech-Vision：位置姿勢の数はカスタマイズされた出力データの数と一致しない

エラーの原因：

- クライアントのインターフェースプログラムは 110 コマンドを呼び出した後、返されたデータに位置姿勢の数とカスタマイズされたデータ要素の数と一致しません。カスタマイズされたデータは Mech-Vision プロジェクトの procedure_out ステップの位置姿勢とラベル以外のポートへの入力データです。考えられる状況は次のとおりです。
  - カスタマイズされたポートのデータが空です。
  - カスタマイズされたポートのデータ長さが位置姿勢のデータ長さと等しくありません。
- 標準インターフェースにステップ procedure_out を使用する場合、位置姿勢の数が N であれば、カスタマイズされたポートのデータ（位置姿勢とラベル以外のデータ）は、1 つまたは N 個のデータ項目を入力する必要があります。それ以外の場合は、このエラーが報告されます。

  - 例えば、カスタマイズされたポートのデータがインスタンスセグメンテーションで得られた対象物の数である場合、1 つのデータ項目を入力するだけでよく、このコマンドを呼び出すたびに同じ数値を受け取ります。カスタマイズされたポートのデータが箱の寸法である場合、N 個のデータ項目が必要であり、このコマンドを呼び出すたびに位置姿勢が対応する「箱の寸法」データを受け取ります。

トラブルシューティング手順：

- Mech-Vision プロジェクトを確認して、出力ポートのデータが上記の要件を満たしていることを確認します。

1025

Mech-Vision：仮想カメラが使用中で、データは放棄された

1026

Mech-Vision：無効な位置姿勢タイプ

エラーの原因：

- クライアントが 105 コマンドを呼び出して Mech-Vision の「経路計画」ステップの結果を取得するとき、経路点タイプのパラメータ値が正しく設定されていませんでした。経路点のタイプパラメータは、1 または 2 のみ設定可能です。

トラブルシューティング手順：

- クライアントインターフェースプログラムで、105 コマンドで設定した経路点のタイプのパラメータ値が正しいかを確認します。

1028

Mech-Viz：経路計画に失敗

エラーの原因：

- 経路計画設定ツールは、ビジョン結果のいずれかの位置姿勢の経路を計画できませんでした。そして、すべての位置姿勢が同じ理由で計画に失敗したわけではありません。

  （同じ理由でビジョン位置姿勢で計画が失敗した場合、対応するエラーコードが生成されます）

  - 例えば、ビジョン結果に 40 個のビジョン位置姿勢があり、ロボットがビジョン位置姿勢に到達できなかったために計画に失敗した 20 個と、衝突が検出されたために計画に失敗した 20 個がある場合、このエラーが報告されます。

トラブルシューティング手順：

- 対応する Mech-Vision プロジェクトに経路計画設定ツールを開き、計画履歴を確認して失敗原因を見つけます。

1030

Mech-Vision：ロボットがビジョン位置姿勢に到達不能

エラーの原因：

- 経路計画設定ツールに送信したツールのビジョン位置姿勢がロボットの到達可能な範囲を超えました。

トラブルシューティング手順：

- カメラの外部パラメータデータを確認します。
- 経路計画設定ツールで、ロボットハンドの TCP が正しく設定されているかどうかを確認します。
- ワークはロボットの到達可能な範囲にあるかどうかを確認します。

1031

Mech-Viz：ロボット関節角度の計算に失敗

エラーの原因：

- 経路計画設定ツールは、経路点に対するロボットの関節角度を計算することができません。

トラブルシューティング手順：

まだありません。

1032

Mech-Vision：経路に到達不能

エラーの原因：

- 経路計画設定ツールの実行中にエラーが発生され、経路計画ステップがエラーコード MP-E0007 を報告しました。

トラブルシューティング手順：

まだありません。

1033

Mech-Vision：移動特異点

エラーの原因：

- ロボット動作計画中に特異点エラーが発生し、計画が失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールで経路点を確認し、経路点の位置姿勢またはパラメータを調整して、ロボットの特異点エラーを回避します。

1034

Mech-Vision：直線運動の計画に失敗
エラーの原因：

- 経路計画設定ツールは、直線運動で目標点に到達できないことを検出しました。

トラブルシューティング手順：

- 経路計画設定ツールでこの目標点の位置姿勢を適切に調整するか、中間点を追加します。
- 関節運動でロボットを移動します。

1035

Mech-Vision：無効な把持位置姿勢
エラーの原因：

- 経路計画設定ツールからエラーコード MP-E0010（原因不明）が報告されました。

トラブルシューティング手順：

まだありません。

1036

Mech-Vision：ロボット本体が衝突
エラーの原因：

- 経路計画設定ツールはロボット本体の衝突を検出しますので、経路計画に失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールで経路点を検査します。

1037

Mech-Vision：ロボットと対象物との衝突
エラーの原因：

- 経路計画設定ツールはロボットがシーンの物体との衝突を検出しますので、経路計画に失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールでシーンの物体と経路点を検査します。

1038

Mech-Vision: 点群と衝突する点数がしきい値より大きく、衝突が検出されました。

エラーの原因：

- 経路計画設定ツールはロボットが物体の点群との衝突を検出し、衝突した点の数がしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールで経路点を検査します。
- 必要に応じて衝突検出設定で衝突点数のしきい値を設定します。

1039

Mech-Vision: 点群と衝突する面積がしきい値より大きく、衝突が検出されました。

エラーの原因：

- 経路計画設定ツールはロボットが物体の点群との衝突を検出し、衝突した面積がしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールで経路点を検査します。
- 必要に応じて衝突検出設定で衝突面積のしきい値を設定します。

1040

Mech-Vision: 点群と衝突する体積がしきい値より大きく、衝突が検出されました。

エラーの原因：

- 経路計画設定ツールはロボットが物体の点群との衝突を検出し、衝突した体積がしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- 経路計画設定ツールで経路点を検査します。
- 必要に応じて衝突検出設定で衝突体積のしきい値を設定します。

1044

Mech-Vision：「ビジョン処理による移動」ステップがビジョン位置姿勢を送信していません。

エラーの原因：

- 「経路計画」ステップの入力ポートがビジョンポイントを送信していません。

トラブルシューティング手順：

- 「経路計画」ステップのビジョンポイントのポートから上に Mech-Vision プロジェクトのデータフローを確認します。

1046

Mech-Vision：無効なエンドツール

エラーの原因：

- 経路計画設定ツールで、エンドツールを設定します。

トラブルシューティング手順：

- 経路計画設定ツールで使用するエンドツールを設定してください。エンドツールを使用しない場合、空のエンドツールを作成してください。

Mech-Viz 関連のエラーのトラブルシューティング

2001

Mech-Viz：ソフトウェアが登録されていない

エラーの原因：

- Mech-Viz が起動されていません。
- 開発者モードで、複数の Mech-Viz を同時に起動します。

トラブルシューティング手順：

- Mech-Viz が起動されているかどうかを確認します。
- Mech-Viz で開発者モードを無効にしてから、再起動します。

2002

Mech-Viz：プロジェクト実行中

エラーの原因：

- クライアントのインターフェースプログラムは Mech-Viz プロジェクトがまだ実行中に、同じ Mech-Viz プロジェクトをトリガーしようとして 201 コマンドを再度呼び出すと、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Viz プロジェクトの実行をトリガーするために、クライアントのインターフェースプログラムが短時間に 201 コマンドを繰り返し呼び出すかどうかを確認します。

2003

Mech-Viz：Mech-Vision からのビジョン結果を受信していない

エラーの原因：

- Mech-Viz プロジェクトは check_look ステップの 2 番目の出口（結果なし）から実行したので、プロジェクトは中止されました。

トラブルシューティング手順：

- Mech-Viz プロジェクトの check_look ステップの「結果なし」出口につなぎを追加する必要があるか、他の分岐から実行する必要があるかを確認します。
- ROI 内の認識するワークが把持されたかどうかを確認します。
- Mech-Vision プロジェクトの ROI とインスタンスセグメンテーションのしきい値が正しく設定されていることを確認します。

2004

Mech-Viz：Mech-Vision からのビジョン結果に到達できない

エラーの原因：

- Mech-Viz によって呼び出されるビジョンサービスが計算したビジョン位置姿勢は、ロボットが到達不能です。

トラブルシューティング手順：

- Mech-Viz でワークの点群の位置を確認します。
- カメラの外部キャリブレーションデータを確認します。
- Mech-Viz でロボット治具の TCP 設定、およびワークはロボットが到達できる範囲にあるかどうかを確認します。

2005

Mech-Viz：ロボット関節角度の計算に失敗

エラーの原因：

- Mech-Viz は、目標点のロボットの JPs 関節角度を計算できません。

トラブルシューティング手順：

まだありません。

2006

Mech-Viz：Mech-Viz の起動時のコマンドパラメータが無効

エラーの原因：

- クライアントインターフェースプログラムは 201 コマンドを呼び出して Mech-Viz プロジェクトの実行をトリガーするときに、ロボット位置姿勢のタイプのパラメータ値が正しく設定されていません。ロボット位置姿勢タイプのパラメータは、0〜2 の範囲です。

トラブルシューティング手順：

- クライアントインターフェースプログラムで、201 コマンドで設定したロボット位置姿勢のタイプのパラメータ値が正しいかどうかを確認します。

2007

Mech-Viz：動作計画に失敗

エラーの原因：

- Mech-Viz は、ビジョン結果のいずれかのビジョン位置姿勢の経路を計画できませんでした。そして、すべてのビジョン位置姿勢が同じ理由で計画に失敗したわけではありません。

  （同じ理由でビジョン位置姿勢で計画が失敗した場合、対応するエラーコードが生成されます）

  - 例えば、ビジョン結果に 40 個のビジョン位置姿勢があり、ロボットがビジョン位置姿勢に到達できなかったために計画に失敗した 20 個と、衝突が検出されたために計画に失敗した 20 個がある場合、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Viz で計画履歴を確認して、計画が失敗した理由を見つけます。

2008

Mech-Viz：プロジェクト実行エラー

エラーの原因：

- Mech-Viz プロジェクトが正常に実行されず、Mech-Viz エラーコード MP-Exxxx または Mech-Center によって解析されないその他のエラーが表示されます。このエラーが発生すると、Mech-Viz プロジェクトは実行を完了せずに中止されます。

トラブルシューティング手順：

- Mech-Viz でエラー情報とログ情報を確認します。

2009

Mech-Viz：ツール位置姿勢（TCP）が提供されていない

エラーの原因：

- クライアントのインターフェースプログラムは 205 コマンドを呼び出して Mech-Viz から計画された経路を取得するとき、返されたデータにはロボットのツール（TCP）位置姿勢が必要ですが、Mech-Viz の出力にはツール（TCP）位置姿勢データは含まれません。

トラブルシューティング手順：

- Mech-Viz の「その他」パネルで「TCP 位置姿勢を送信」にチェックを入れていることを確認します。

2010

Mech-Viz：経路に到達できない

エラーの原因：

- Mech-Viz プロジェクトの実行エラーで、エラーコード「MP-E0007」が報告されます。

トラブルシューティング手順：

まだありません。

2011

Mech-Viz：DO リストが提供されていない

エラーの原因：

- クライアントのインターフェースプログラムは 206 コマンドを呼び出して、ツールを制御する DO 信号リスト（パーティション吸盤、配列治具など）を取得しますが、Mech-Viz プロジェクトでは DO リストが設定されていません。

トラブルシューティング手順：

- Mech-Viz プロジェクトで「ビジョン処理による移動」ステップの後に set_do_list ステップがあるかどうかを確認し、ステップパラメーターの「受信者」を「標準インターフェース」に設定します。

2012

Mech-Viz：無効な位置姿勢タイプ

エラーの原因：

- クライアントのインターフェースプログラムは 205 コマンドを呼び出して Mech-Viz から計画された経路を取得するときに、設定した経路点タイプのパラメータ値が不正になります。

トラブルシューティング手順：

- クライアントインタフェースプログラムで、205 コマンドで設定した経路点タイプのパラメータの値が正しいかどうか確認します。

2013

Mech-Viz：無効な位置姿勢データ

エラーの原因：

- クライアントのインターフェースプログラムが 201 コマンドを呼び出して Mech-Viz プロジェクトの実行をトリガーするとき、設定されたロボット位置姿勢パラメータ値は 6 桁未満です。Mech-Viz プロジェクトは 6 軸ロボットの位置姿勢データのみをサポートし、4 軸または 5 軸ロボットの 6 番目の軸データは 0 で埋める必要があります。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、201 コマンドを呼び出して Mech-Viz プロジェクトをトリガーするときに、ロボットの位置姿勢 データが 6 ビットの JPs 関節角度データであるかどうかを確認します。Mech-Viz が現在のロボットの位置姿勢情報を必要としない場合は、位置姿勢タイプを 0 に設定できます。つまり、ロボットの位置姿勢を入力する必要はありません。

2014

Mech-Viz：プロジェクトが設定されていない

エラーの原因：

- プロジェクトが Mech-Viz で開かれていません。
- Mech-Viz プロジェクトは **自動的に読み込む** にチェックを入れていません。

トラブルシューティング手順：

- Mech-Viz でエラーメッセージを確認し、正しいプロジェクトを開いて **自動的に読み込む** にチェックを入れます。

2015

エラーコードが使用されていない

2016

Mech-Viz：ステップパラメータの設定に失敗

エラーの原因：

- クライアントのインターフェースプログラムは 208 コマンドを呼び出して Mech-Viz ステップパラメータを設定するとき、エラーが報告されました。

トラブルシューティング手順：

- 構成ファイルのステップ ID とパラメータのキーの名前を確認します。
- Mech-Center で **設定** ▸ *Mech-Interface* ▸ **詳細設定** をクリックして、構成ファイルが表示されます。

2017

Mech-Viz：停止に失敗

エラーの原因：

- クライアントのインターフェースプログラムは 202 コマンドを呼び出して Mech-Viz の実行を停止します。Mech-Viz が 5 秒以内に正常に停止しない場合、このエラーが報告されます。

トラブルシューティング手順：

まだありません。

2018

Mech-Viz：無効な分岐の出口番号

エラーの原因：

- クライアントのインターフェースプログラムはコマンド 203 を呼び出して、Mech-Viz の分岐出口を設定します。出口番号が 0 以下の場合、または出口番号が分岐ステップの出口数を超えた場合、このエラーが報告されます。

トラブルシューティング手順：

- Mech-Viz プロジェクトの分岐ステップの各出口を確認します。
- 203 コマンドで設定された出口番号を確認します。

2019

Mech-Viz：分岐の設定に失敗（分岐のステップ ID を確認してください）

エラーの原因：

- クライアントのインターフェースプログラムは 203 コマンドを呼び出して Mech-Viz の分岐を設定します。分岐ステップのステップ ID は整数である必要があります。Mech-Viz プロジェクトに指定された ID の分岐ステップがない場合、このエラーが報告されます。
- Mech-Viz では、ステップ ID の読み取りと設定は対応するステップパラメータで行います。範囲は 1~99 です。

トラブルシューティング手順：

- コマンドによって送信されたステップ ID に対応するステップが Mech-Viz にあることを確認します。
- コマンドと Mech-Viz プロジェクトで設定されたステップ ID が正の整数であることを確認します。

2020

Mech-Viz：特異点のため実行に失敗

エラーの原因：

- Mech-Viz はロボット動作計画中に特異点エラーが発生し、計画が失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトで目標点を確認し、目標点の位置姿勢またはパラメータを調整して、ロボットの特異点エラーを回避します。

2021

Mech-Viz：直線運動の計画に失敗

エラーの原因：

- Mech-Viz は、直線運動で目標点に到達できないことを検出しました。

トラブルシューティング手順：

- Mech-Viz でこの移動点の位置姿勢を適切に調整するか、中間点を追加します。
- 関節運動でロボットを移動します。

2022

Mech-Viz：実行していない

エラーの原因：

- クライアントのインターフェースプログラムは 203 コマンドを呼び出して Mech-Viz の分岐を設定するとき、プロジェクトが実行されていません。
- クライアントのインターフェースプログラムは 205 コマンドを呼び出して Mech-Viz から計画された経路を取得する前に、プロジェクトの実行がトリガーされていません。
- クライアントのインターフェースプログラムは 205 コマンドを呼び出したとき、Mech-Viz が計画結果を出力しませんでした。
- クライアントインタフェースプログラムは、すべての位置姿勢を取得し、キャッシュが空であるにもかかわらず、コマンド 205 の呼び出しを続けます。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、Mech-Viz の分岐を設定する前にプロジェクトを実行する必要があります。
- クライアントのインターフェースプログラムを確認し、Mech-Viz の実行をトリガーしてから計画された経路を取得する必要があります。
- クライアントインターフェースプログラムを確認します。205 コマンドによって返されたデータのパラメータ「位置姿勢の送信が完了するかどうか」が 1 の場合、全ての位置姿勢送信が完了したことを意味し、205 コマンドを呼び出し続けると本エラーとなります。

2023

Mech-Viz：プロジェクトファイルが異常

2024

Mech-Viz：無効な分岐名

エラーの原因：

- クライアントのインターフェースプログラムは 203 コマンドを呼び出して Mech-Viz の分岐を設定します。分岐ステップのステップ ID は正の整数である必要があります。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、設定された Mech-Viz 分岐ステップのステップ ID は正の整数である必要があります。
- Mech-Viz では、ステップ ID の読み取りと設定は対応するステップパラメータで行います。範囲は 1~99 です。

2025

Mech-Viz：実行時間タイムアウト

エラーの原因：

- クライアントのインターフェースプログラムは 205 コマンドを呼び出して Mech-Viz の計画された経路を取得しますが、プロジェクトは指定された時間（デフォルトは 10 秒）内に実行を終了しません。実行時間タイムアウトためエラーが報告されます。

トラブルシューティング手順：

- Mech-Viz プロジェクトが正常に実行された時間を確認します。10 秒以上かかる場合は、Mech-Center で **設定** ▸ *Mech-Interface* ▸ **詳細設定** をクリックして対応する設定を行います。
- クライアントのインターフェースプログラムは 205 コマンドを呼び出す前に遅延時間を設定できます。

2026

Mech-Viz：無効なインデックス付きのステップ名

エラーの原因：

- クライアントのインターフェースプログラムは 204 コマンドを呼び出して Mech-Viz のインデックスパラメータ付きのステップのインデックス値を設定する場合、設定されたステップ ID は正の整数である必要があります。
  - 204 コマンドのフォーマット：*204,* **インデックスパラメータ付きのステップ** *ID,* **インデックス値**。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認します。204 コマンドを呼び出して Mech-Viz のインデックス値を設定する場合、インデックスパラメータ付きのステップ ID は正の整数である必要があります。
- Mech-Viz では、ステップ ID の読み取りと設定は対応するステップパラメータで行います。範囲は 1~99 です。

2027

Mech-Viz：無効なインデックス

エラーの原因：

- クライアントのインターフェースプログラムは 204 コマンドを呼び出して Mech-Viz のインデックス値を設定します。設定されたインデックス値は正の整数である必要があります。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、204 コマンドを呼び出して Mech-Viz のインデックスを設定する場合、インデックス値は正の整数である必要があります。

2028

Mech-Viz：インデックスの設定に失敗（インデックパラメータ付きのステップがプロジェクトに入れるかどうかを確認してください）

エラーの原因：

- クライアントのインターフェースプログラムは 204 コマンドを呼び出して Mech-Viz の移動ステップのインデックスパラメータを設定します。移動ステップののステップ ID は正の整数である必要があります。Mech-Viz プロジェクトにステップ ID に対応する移動ステップがない場合、このエラーが報告されます。

トラブルシューティング手順：

- クライアントのインターフェースプログラムで設定した移動ステップのステップ ID を確認します。
- Mech-Viz プロジェクトの移動ステップのステップ ID を確認します。
- Mech-Viz では、ステップ ID の読み取りと設定は対応するステップパラメータで行います。範囲は 1~99 です。

2029

Mech-Viz：外部移動の設定に失敗

2030

Mech-Viz：無効な把持位置姿勢

エラーの原因：

- Mech-Viz プロジェクトはエラー MP-E0010 を報告します。理由は不明です。

トラブルシューティング手順：

まだありません。

2031

Mech-Viz：ロボット本体が衝突

エラーの原因：

- Mech-Viz はロボット本体の衝突を検出しますので、経路計画に失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトの目標点を確認します。

2032

Mech-Viz：シーンとの衝突

エラーの原因：

- Mech-Viz はロボットがシーンの物体との衝突を検出しますので、経路計画に失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトのシーンのモデルおよび目標点を確認します。

2033

Mech-Viz：点群と衝突する点の数が上限を超えた

エラーの原因：

- Mech-Viz はロボットが物体の点群との衝突を検出し、衝突した点の数がしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトの目標点を確認します。
- 必要に応じて Mech-Viz の衝突検出設定で衝突点数のしきい値を設定します。

2034

Mech-Viz：点群と衝突する面積が上限を超えた

エラーの原因：

- Mech-Viz はロボットが物体の点群との衝突を検出し、衝突した面積がしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトの目標点を確認します。
- 必要に応じて Mech-Viz の衝突検出設定で衝突面積のしきい値を設定します。

2035

Mech-Viz：点群と衝突するボリュームが上限を超えた

エラーの原因：

- Mech-Viz はロボットが物体の点群との衝突を検出し、衝突したボリュームがしきい値を超えたので、経路計画に失敗しました。

トラブルシューティング手順：

- Mech-Viz プロジェクトの目標点を確認します。
- 必要に応じて Mech-Viz の衝突検出設定で衝突ボリュームのしきい値を設定します。

2036

Mech-Viz：撮影されていない

エラーの原因：

- Mech-Viz プロジェクトの実行中に check_look ステップの 4 番目の出口（未撮影）から実行し、この出口は他のステップにつながらないため、Mech-Viz プロジェクトの実行は中止されています。

トラブルシューティング手順：

- Mech-Viz プロジェクトの visual_look および check_look ステップは設定されたビジョンサービスと一致するかを確認します。

2037

Mech-Viz：ビジョン結果がない

エラーの原因：

- Mech-Viz プロジェクトの実行中に check_look ステップの 2 番目の出口（結果なし）から実行し、この出口は他のステップにつながらないため、Mech-Viz プロジェクトの実行は中止されています。また、対応する Mech-Vision プロジェクトはビジョン結果を出力しません。

トラブルシューティング手順：

- *1002* を参照して問題をトラブルシューティングします。

2038

Mech-Viz：ビジョン結果の ROI に点群がない

エラーの原因：

- Mech-Viz プロジェクトの実行中に check_look ステップの 5 番目の出口（点群なし）から実行し、この出口は他のステップにつながらないため、Mech-Viz プロジェクトの実行は中止されています。また、対応する Mech-Vision プロジェクトはビジョン結果を出力しません。

トラブルシューティング手順：

- *1003* を参照して問題をトラブルシューティングします。

2039

Mech-Viz：計画するビジョン位置姿勢がない

エラーの原因：

- Mech-Viz プロジェクトは visual_move によって計画されるビジョン位置姿勢がありません。check_look ステップが使用されていない場合、または check_look ステップが 2 番目の出口（結果なし）から実行し、その後に「ビジョン処理による移動」ステップがある場合、このエラーが報告されます。

トラブルシューティング手順：

- *1002* を参照して問題をトラブルシューティングします。
- Mech-Viz プロジェクトに check_look ステップが正しく設定されているかを確認します。

2040

Mech-Viz：ビジョン結果を再利用するための計画経路が失敗

エラーの原因：

- Mech-Viz プロジェクトの visual_move ステップは、「ビジョン結果を再使用」のパラメータにチェックを入れ、返された同じビジョン結果を使用して、複数のワークを把持します。すべてのビジョン位置姿勢に対して正常に把持できる動作経路が計画されているわけではなく、一部の計画失敗があります。
- このエラーが発生した場合でも、成功した計画経路が出力されます。

トラブルシューティング手順：

- Mech-Viz で計画履歴を確認して、計画が失敗した理由を見つけます。

2041

Mech-Viz：ステップパラメータの取得に失敗

エラーの原因：

- クライアントのインターフェースプログラムは 207 コマンドを呼び出して、Mech-Viz のステップパラメータの取得中にエラーが発生しました。

トラブルシューティング手順：

- 構成ファイルの内容を確認します。Mech-Center で **設定** ▸ *Mech-Interface* ▸ **詳細設定** をクリックして、構成ファイルが表示されます。
  - 特に、ステップ ID とパラメータのキーの名前が正しいことを確認します。

2042

Mech-Viz：「ビジョン処理による移動」からの動作計画の取得に失敗

エラーの原因：

- クライアントのインターフェースプログラムは 210 コマンドを呼び出して、Mech-Viz から visual_move ステップの計画結果の取得中にエラーが発生しました。

トラブルシューティング手順：

まだありません。

2043

Mech-Viz：ビジョン結果からのカスタマイズされたデータの取得に失敗

エラーの原因：

- クライアントのインターフェースプログラムは 210 コマンドを呼び出して、Mech-Viz から visual_move の計画結果とビジョン結果のカスタマイズされたデータを取得しましたが、カスタマイズされたデータの取得中にエラーが発生しました。

トラブルシューティング手順：

- *1024* を参照して問題をトラブルシューティングします。

2044

Mech-Viz：ビジョンサービスが登録されていない

エラーの原因：

- Mech-Viz プロジェクトの visual_look ステップにビジョンサービス名が正しく設定されていません。

トラブルシューティング手順：

- Mech-Viz プロジェクトの visual_look ステップにビジョンサービス名が正しく設定されているかを確認します。

2045

Mech-Viz：無効なエンドツール

エラーの原因：

- Mech-Viz プロジェクトに「ツールを設定」ステップがありますが、エンドツールが設定されていません。

トラブルシューティング手順：

- Mech-Viz プロジェクトの「ツールを設定」ステップにツールを設定してください。ツールを使用していない場合、ステップを削除するか、空のツールを作成してください。

Mech-Center 関連のエラーのトラブルシューティング

3001

Mech-Center：無効なコマンド

エラーの原因：

- システムは、クライアントのインターフェースプログラムによって送信されたコマンドコードをサポートしていません。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認します。

3002

Mech-Center：インターフェースコマンド長さやフォーマットがエラー

エラーの原因：

- クライアントのインターフェースプログラムから送信されたコマンド情報の長さが異常です。例えば、ロボットの位置姿勢データの桁数が 6 桁未満です。
- クライアントのインターフェースプログラムから送信されたコマンド情報のフォーマットが異常です。例えば、カンマ（英半角カンマ）区切りが使用されていません。

トラブルシューティング手順：

- クライアントのインターフェースプログラムから送信されたコマンド情報がインターフェースの要件を満たしているかどうかを確認します。

3003

Mech-Center：クライアントが切断された

3004

Mech-Center：サーバーが切断された

3005

Mech-Center：Mech-Vision を呼び出す時間がタイムアウト

3006

Mech-Center：未知エラー

3007

Mech-Center：データ確認応答信号がタイムアウト

エラーの原因：

PROFINET/EtherNet/IP を使用して通信する場合：

- Mech-Center が新しい位置姿勢データをポートに送信する前に、クライアントのインターフェースプログラムの Data_Acknowledge 信号が 0 であることを確認する必要があります。指定されたタイムアウト期間内にクライアントが Data_Acknowledge 信号をリセットしない場合、このエラーが報告されます。
- Mech-Center が位置姿勢データをポートに送信した後、クライアントのインターフェースプログラムはポートデータが読み取られたことを示す 1 の Data_Acknowledge 信号を指定されたタイムアウト期間内に送信しない場合、このエラーが報告されます。

トラブルシューティング手順：

- クライアントのインターフェースプログラムを確認し、102 または 205 コマンドを呼び出す前に、Data_Acknowledge 信号が 0 であることを確認します。
- クライアントのインターフェースプログラムを確認し、ポートの位置姿勢データを読み取った後、Data_Acknowledge を 1 に設定します。

3008

Mech-Center：構成 ID が存在しないため、パラメータの読み取り/設定ができない

エラーの原因：

- クライアントのインターフェースプログラムがコマンド 207 を呼び出してステップパラメータを取得するか、コマンド 208 を呼び出してステップパラメータを設定するときは、対応する構成 ID と内容が Mech-Center の構成ファイルに見つかりませんでした。
- Mech-Center で **設定** ‣ *Mech-Interface* ‣ **詳細設定** をクリックして、構成ファイルが表示されます。

トラブルシューティング手順：

- 構成ファイル内の構成 ID は正の整数であることを確認します。
- 構成ファイル内の対応する構成 ID と内容が存在するかどうかを確認します。

キャリブレーション関連のエラーのトラブルシューティング

7001

キャリブレーション：パラメータがエラー

エラーの原因：

- クライアントのインターフェースプログラムがキャリブレーション中に Mech-Center に送信されるロボットの位置姿勢データは 6 ビット未満です。JPs とフランジ位置姿勢データは両方とも 6 ビットである必要があります。4 軸または 5 軸ロボットの場合、残りのデータは 0 で埋める必要があります。

トラブルシューティング手順：

- クライアントのインターフェースプログラムから送信されたロボット位置姿勢データの長さが正しいか確認します。

7002

キャリブレーション：Mech-Vision から提供されるキャリブレーションポイントがない

エラーの原因：

- Mech-Vision はキャリブレーション実行中にロボット位置姿勢を Mech-Center に送信していません。

トラブルシューティング手順：

まだありません。

7003

キャリブレーション：ロボットはキャリブレーションポイントに到達することに失敗

3.4 Adapter

標準インターフェースの機能がプロジェクトの要件を満たすことができない場合、Adapter プロジェクトを作成することにより、複雑なプロセス制御を実現できます。

本節では、以下の内容が含まれています。

- *Adapter* の概要
- *Adapter* ジェネレーターのマニュアル
- *Adapter* プログラミングのガイド
- *Adapter* のプログラミング例

ちなみに: Adapter プロジェクトで追加の Python ライブラリを使用する必要がある場合は、それを Mech-Center ソフトウェアの「python」ディレクトリにインストールしてください。Python ライブラリのインストール方法は次のとおりです。

1. 「コマンドプロンプト」または「PowerShell」プログラムを起動します。
2. Mech-Center ソフトウェアの「python」ディレクトリに切り替えます（例えば：C:¥Mech-Mind¥Mech-Center-1.6.x¥python）。

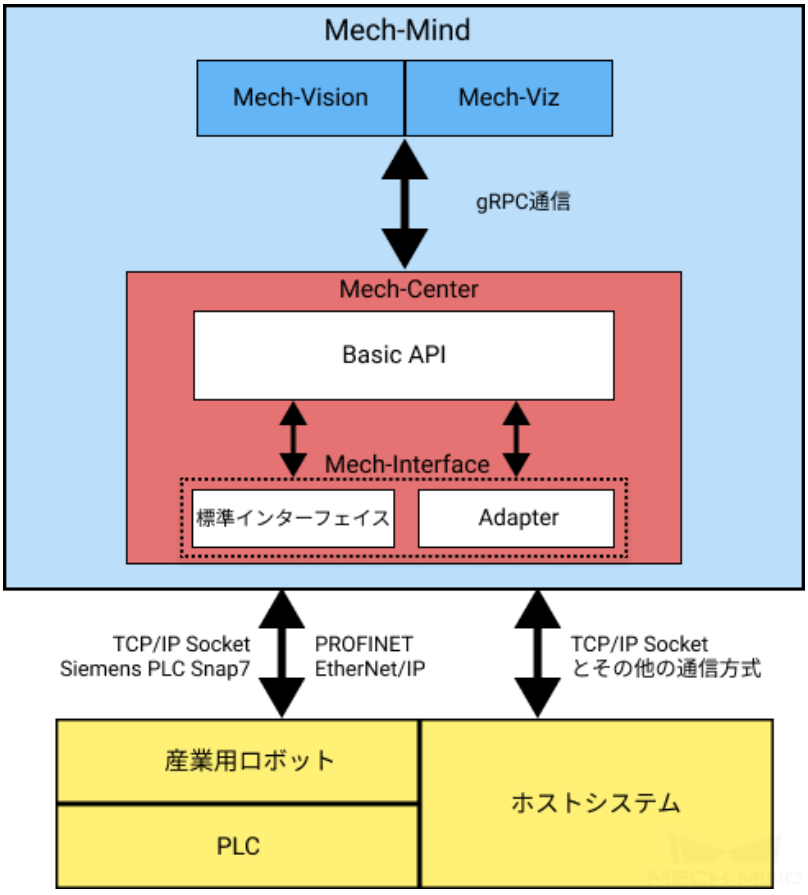
3. 「**./python -m pip install** *library_name* 」コマンドを実行します。

3.4.1 Adapter の概要

- *Adapter* の紹介
- *Adapter* の機能
- *Adapter* の開発
- *Adapter* プロジェクトの設定

Adapter の紹介

Adapter は Mech-Center ソフトウェア内の通信コンポーネントです。Basic API インターフェースを介して Mech-Vision および Mech-Viz との gRPC 通信を行い、TCP/IP Socket、HTTP および PLC プロトコル（例えば：三菱 PLC MC プロトコル）などの様々な一般的な産業用通信方式を介して外部デバイスと通信します。



Adapter の使用シーンについては、使用シーン をご参照ください。

Adapter の機能

Adapter は次の機能に対応します。

- **内部で Mech-Vision および Mech-Viz ソフトウェアの制御を実現できます。**

タイプ	機能
Mech-Vision 関連	Mech-Vision を実行して視覚結果を取得します
	Mech-Vision のステップパラメータを設定します
	Mech-Vision のステップパラメータを読み取ります
	Mech-Vision のパラメータレシピを切り替えます
Mech-Viz 関連	Mech-Viz を起動します
	Mech-Viz を停止します
	Mech-Viz のステップパラメータを設定します
	Mech-Viz のステップパラメータを読み取ります
	治具の番号を設定します
	ロボットの動作速度を設定します
	点群衝突検出のパラメータを設定します
	Mech-Viz の実行状態を取得して返します
その他	詳細については、*Adapter* **プログラミングのガイド** をご参照ください。

- **外部でユーザーインターフェース、データベース、ファイルの読み書き、Web システムとの通信などの視覚に関連しない機能を実現できます。**

  外部通信機能は、Python プログラミングによって開発および実装する必要があります。

Adapter の開発

TCP/IP Socket プロトコルの場合、Mech-Center は **Adapter ジェネレーター** を提供し、Adapter を初めて使用するユーザーが Adapter プロジェクトをすばやく生成することに役立ちます。詳細については、*Adapter* **ジェネレーターのマニュアル** をご参照ください。

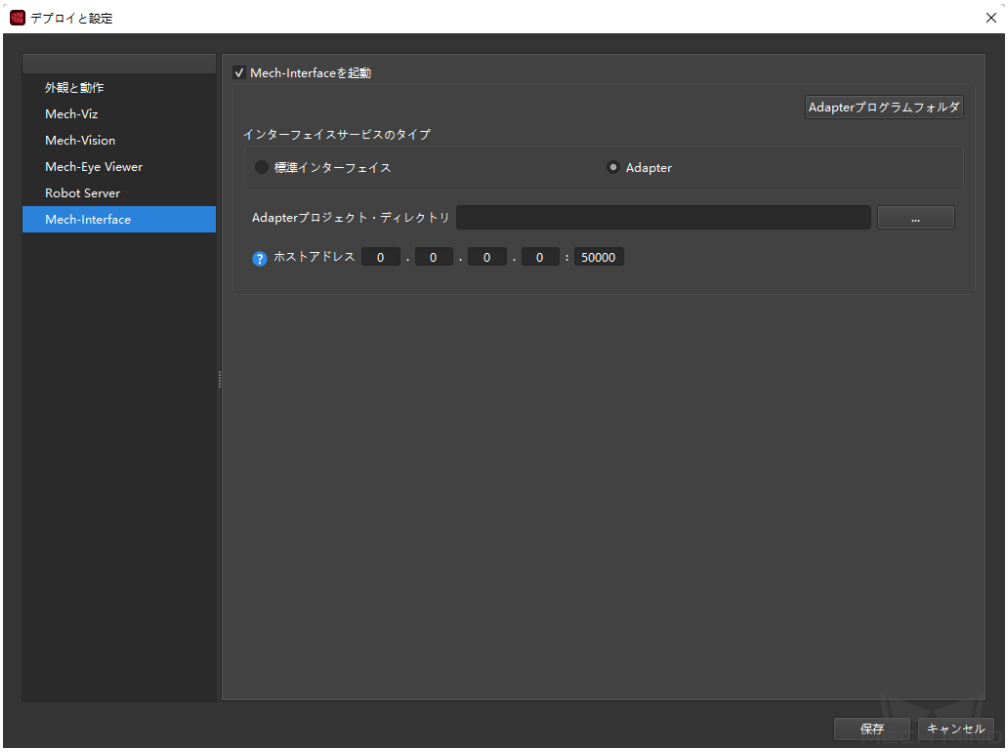
既存の Adapter に基づいて、プロジェクトのカスタマイズ開発を行うことができます。

Adapter プロジェクトを最初から作成することもできます。詳細については、*Adapter* **プログラミングのガイド** と *Adapter* **のプログラミング例** をご参照ください。

Adapter プロジェクトの設定

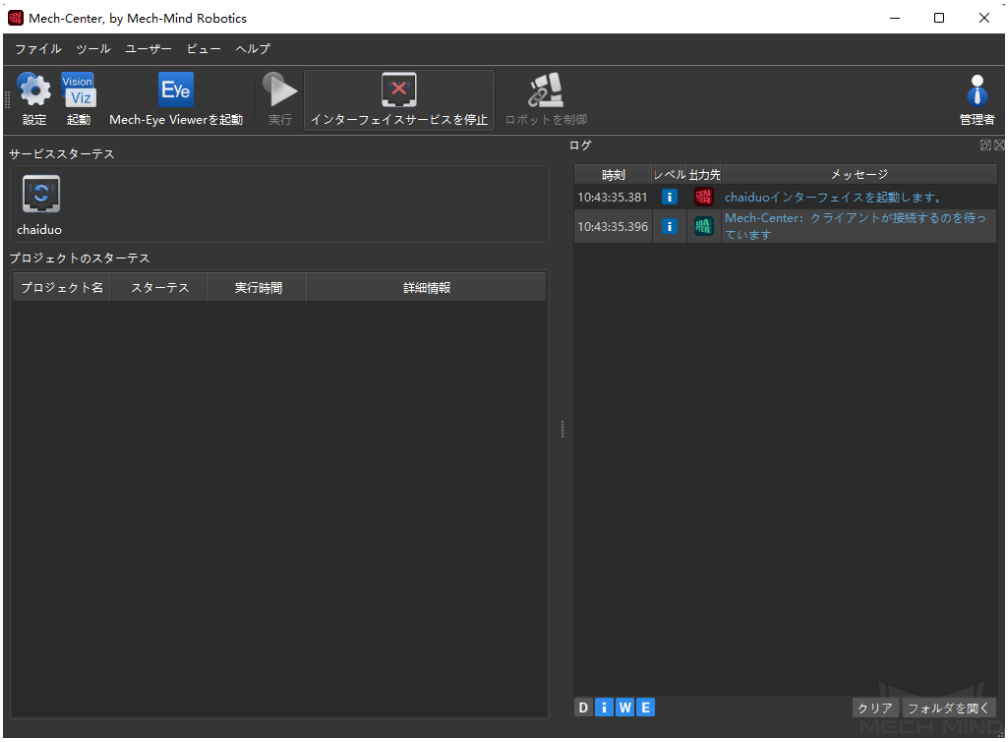
Adapter プロジェクトを作成したら、以下の手順に従って Adapter プロジェクトを設定できます。

1. Mech-Center ソフトウェアを起動し、ツールバーの **設定** をクリックします。
2. 下図に示すように、**設定** 画面の左側で **Mech-Interface** をクリックし、**Mech-Interface を使用** にチェックを入れて、**インターフェイスサービスのタイプ** を **Adapter** に設定します。



3. **Adapter プロジェクト・ディレクトリ** を、Adapter プログラムが保存されているディレクトリに設定します。
4. 実際の状況に応じて **ホストアドレス** を設定します。ポートが通信先と一致する必要があります。
   - 通信先がサーバーの場合、**ホストアドレス** は相手側の IP アドレスに設定する必要があります。
   - 通信先がクライアントの場合、**ホストアドレス** は「0.0.0.0」に設定する必要があります。
5. **保存** をクリックしその画面を閉じます。その後、Mech-Center ソフトウェアを再起動します。
6. ツールバーの **インターフェースサービスを起動** をクリックして Adapter サービスを起動します。

下図に示すように、**インターフェースサービスを起動** が **インターフェースサービスを停止** になって、サービスステータス内に起動された Adapter プロジェクトが表示されると Adapter サービスが起動されている状態になります。



Adapter を簡単に理解したら、*Adapter* ジェネレーターのマニュアル を参照して、最初の Adapter プロジェクトをすばやく生成できます。

3.4.2 Adapter ジェネレーターのマニュアル

本節では、Adapter ジェネレーターと、Adapter ジェネレーターを使用して Adapter プログラムをすばやく生成する方法について説明しています。

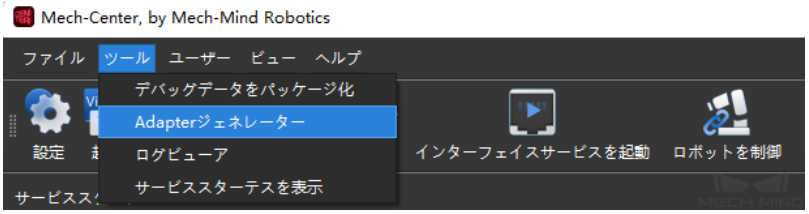
Adapter ジェネレーターの紹介

Adapter ジェネレーターは、Mech-Center に統合されたコンポーネントです。Adapter ジェネレーターは、TCP/IP Socket 通信を使用し、Mech-Vision のみを使用して視覚位置姿勢を提供するシーンにのみ適しています。

Adapter ジェネレーターは、次のことに役立ちます。

- Adapter プロジェクトをすばやく生成すること。
- 複雑なニーズを満たすために、Adapter プログラミングの開発をすばやく学ぶこと。

下図に示すように、**ツール** ‣ *Adapter* **ジェネレーター** をクリックして Adapter ジェネレーターを起動して設定を行います。



注意： Adapter ジェネレーターは、管理者モードでのみ使用できます。

Adapter ジェネレーターを使用して Adapter プログラムをすばやく生成する

ヒント： コンポーネントにマウスを置くと詳細な説明が表示されます。

ネットワーク設定 - サーバー/クライアント

下図のような画面で **Adapter 名**、**クライアント/サーバー**、**形式** を設定してから、**次へ** をクリックします。

パラメータの説明：

- **Adapter 名**：Adapter プロジェクトの名前を設定します。
- **Adapter の役割**：Adapter を TCP/IP Socket 通信用のサーバー (Server) またはクライアント (Client) として設定します。Adapter がクライアントとして機能し、サーバーがクラ イアントに対してポート制限を持っている場合は、**ポートバインド** にチェックを入れるする必要があります。

  ヒント： 通信先と正常に通信するために、Adapter プロジェクトを設定するときはホスト IP アドレスとポートを正しく設定する必要があります。詳細については、*Adapter* **プロジェクトの設定** をご参照ください。

- **形式**：通信時のデータ送信フォーマットを設定し、ASCII 文字列または HEX（16 進数）をサポートします。「HEX」に設定されている場合、バイトオーダ（エンディアン）も「ビッグエンディアン」または「リトルエンディアン」で指定する必要があります。

Mech-Viz/Mech-Vision 設定ー位置姿勢と関節角度

ヒント:

- この設定を行う前に、Mech-Vision プロジェクトと衝突検出のための Mech-Viz プロジェクトを開いて自動読み込みに設定し、プロジェクトが Mech-Center に読み込まれていることを確認します。
- Mech-Center は、衝突検出のための Mech-Viz サンプルプロジェクト「check_collision」を提供しています。それは、Mech-Center のインストールパスの「tool\viz_project」フォルダに格納されています。

注意:「check_collision」サンプルプロジェクトに示すように、プロジェクトは visual_look によってトリガーされて撮影し、非移動ステップを含める必要があります。また、notify_1、notify_2 および vision_look_1 などのステップ名を変更することはできません。

下図のような画面で Mech-Viz および Mech-Vision プロジェクトのパラメータを設定してから、**次へ** をクリックします。

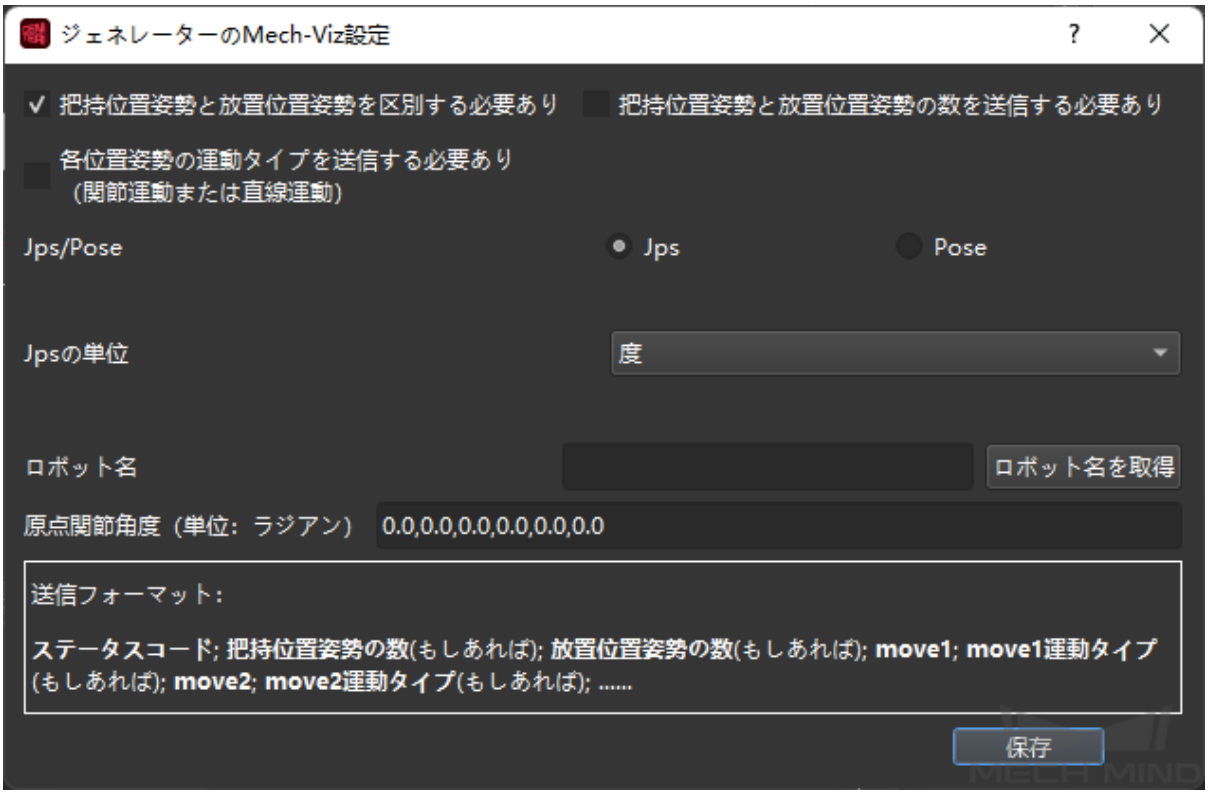
Mech-Vision に関するパラメータの説明：

- **Mech-Vision プロジェクト名**：Mech-Vision プロジェクト名を選択します。ドロップダウンリストから Adapter と通信する Mech-Vision プロジェクトを選択します。
- **位置姿勢の数**：通信先に送信する位置姿勢の数を設定します。
- **位置姿勢の形式**：四元数またはオイラー角を選択可能です。
- **位置姿勢の単位**：一般的に、ミリメートルおよび度が使用されます。
- **カメラ**：ETH および EIH の 2 つのカメラ取付方式があります。
- **位置姿勢の基準座標系**：送信する位置姿勢が基づく座標系を選択します。一般に、位置姿勢はベース座標系に基づいています。位置姿勢は、EIH モードでロボットがロボットエンドの位置姿勢を提供できない場合にのみ、ツール座標系に基づきます。

- **小数点以下の桁数を保持**：送信する位置姿勢の小数点以下の桁数を設定します。最大の桁数は 10 です。

Mech-Viz に関するパラメータの説明：

- **Mech-Viz を使って衝突を検出**：チェックを入れたら、視覚位置姿勢が Mech-Viz によって検出・計算されます。それにより、計画に失敗した位置姿勢がフィルタリングされ、衝突のない把持位置姿勢が選出されます。
- **Mech-Viz の設定**：この画面で *Mech-Viz* **を使って衝突を検出** にチェックを入れた後、*Mech-Viz* **の設定** をクリックすれば下図のような画面が表示されます。設定が完了したら、右下隅にある **保存** をクリックします。

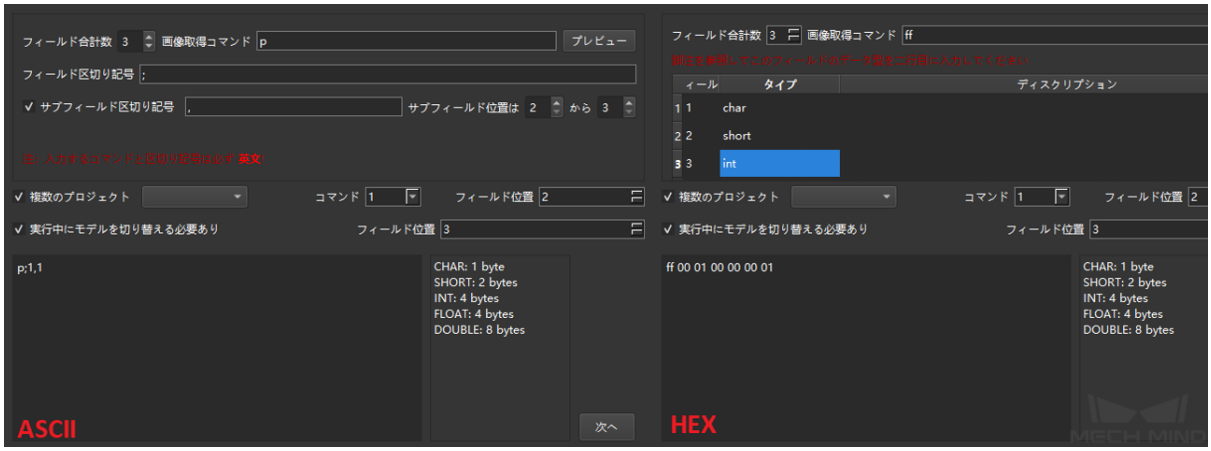


  - **把持位置姿勢と配置位置姿勢を区別する必要あり**：把持位置姿勢は「ビジョン処理による移動」の前のすべての位置姿勢（視覚認識による移動を含む）であり、配置位置姿勢は「ビジョン処理による移動」の後のすべての位置姿勢です。シーンによっては、ロボットがステップによって把持動作と配置動作を区別する必要があります。
  - **把持位置姿勢と放置位置姿勢の数を送信する必要あり**：把持位置姿勢の数が多い場合、数量フィールドをつけることができます。チェックを入れた後、デフォルトの位置姿勢の数が 1 以上であればこのフィールドをつけます。
  - **各位置姿勢の運動タイプを送信する必要あり**：Mech-Viz での移動ステップは、関節運動または直線運動と 2 種類があります。
  - **コードをアップデート**：デフォルトで、関節運動の対応コードは 1 で、直線運動の対応コードは 2 です。コードはカスタマイズ可能で、それを変更して **コードをアップデート** をクリックすれば、新しいコードが有効になります。
  - **Jps/位置姿勢**：位置姿勢の送信形式で、デフォルトで Jps を使用します。
  - **Jps の単位／位置姿勢の単位**：通常、送信する位置姿勢の形式は **Jps** の場合、Jps の単位が度に設定され、送信する位置姿勢の形式は **Pose** の場合、位置姿勢の単位はメートルに設定されます。

- **ロボット名**：ロボットサービスの名前を設定します。Mech-Viz でロボットの動作をシミュレートするには、ロボット実機のサービスが必要です。このサービスは、生成された Adapter によってシミュレートされます。ロボット名はこのサービスの名前であり、Mech-Viz でのロボット名と一致している必要があります。この画面で **ロボット名を取得** をクリックすれば、ロボット名が正常に追加されます。
- **原点関節角度（ラジアン単位）**：これは、Mech-Viz でロボット動作の基準原点を指します。単位はラジアンで、コンマで区切られます。Mech-Viz の「移動」ステップを編集して原点とし、原点の関節位置をコピーすることができます。

ストリームの受信設定 - 受信するフィールド

下図のような画面で受信フィールドのフォーマットを設定する必要があります。画像取得コマンド、複数のプロジェクト、フィールド合計数、フィールドタイプ、フィールド区切り記号とサブフィールド区切り記号を設定する必要があります。設定が完了したら、右下隅にある **次へ** をクリックします。



- **フィールド合計数**：設定するパラメータの数によって異なります。その範囲を「1~10」に設定できます。また、フィールドに画像取得コマンドがなければなりません。
- **画像取得コマンド**：外部から Mech-Mind ソフトウェアシステムに画像取得コマンドを送信し、カメラで撮影します。ASCII の場合、*p* のような文字を入力し、フィールド位置をデフォルトの 1 にすることをお勧めします。HEX の場合、*0xff* または *ff* などの 16 進数形式の整数を使用する必要があります。
- **フィールド区切り記号** と **サブフィールド区切り記号**：ASCII の場合は設定が必要です。2 つ以上のフィールドがある場合は、フィールド区切り記号を入力する必要があります。追加情報で別のフィールド区切り記号が必要な場合は、サブフィールド区切り記号も必要です。また、サブフィールド区切り記号の開始範囲と終了範囲を指定できます。
- **フィールドタイプ** と **ディスクリプション**：HEX の場合は設定が必要です。CHAR、SHORT、INT、FLOAT および DOUBLE が選択可能です。**ディスクリプション** はフィールドの機能または意味を説明するために使用されます。
- **複数のプロジェクト**：この設定は選択可能です。1 つの項目に複数の Mech-Vision プロジェクトがある場合、外部からのコマンドによって異なる Mech-Vision プロジェクトを呼び出す必要があります。その中、コマンドを設定することができます。

> **注意：** プロジェクトごとに唯一のコマンドが対応します。また、フィールドの位置も唯一で、他のフィールドと重複してはいけません。

ヒント： *Mech-Viz/Mech-Vision* 設定ー位置姿勢と関節角度 の画面で **カメラ** が **Eye In Hand** に設定されている場合に使用できます。

下図のような画面で、画像撮影時のロボットの位置姿勢の形式を設定してから、**次へ** をクリックします。

パラメータの説明：

- **ロボットから画像取得時の関節角度またはフランジ位置姿勢を Adapter に送信する必要あり**：通信先はロボットベース座標系に基づいた位置姿勢を提供する必要がある場合、画像取得時のロボットからの関節角度またはフランジ姿勢が必要であるため、これにチェックを入れてください。チェックを入れたら、位置姿勢を関節角度またはオイラー角に設定することができます。
- **関節角度の単位**：ラジアンまたは度を選択可能です。
- **フランジ位置姿勢の単位**：フランジ位置姿勢の単位（四元数またはオイラー角）を設定します。四元数の場合はメートルとミリメートルを選択可能で、オイラー角の場合は度とラジアンを選択可能です。
- **関節角度/フランジ位置姿勢のフィールド位置**：全体フィールド内の関節角度または位置姿勢の開始フィールドと終了フィールドの位置を設定します。

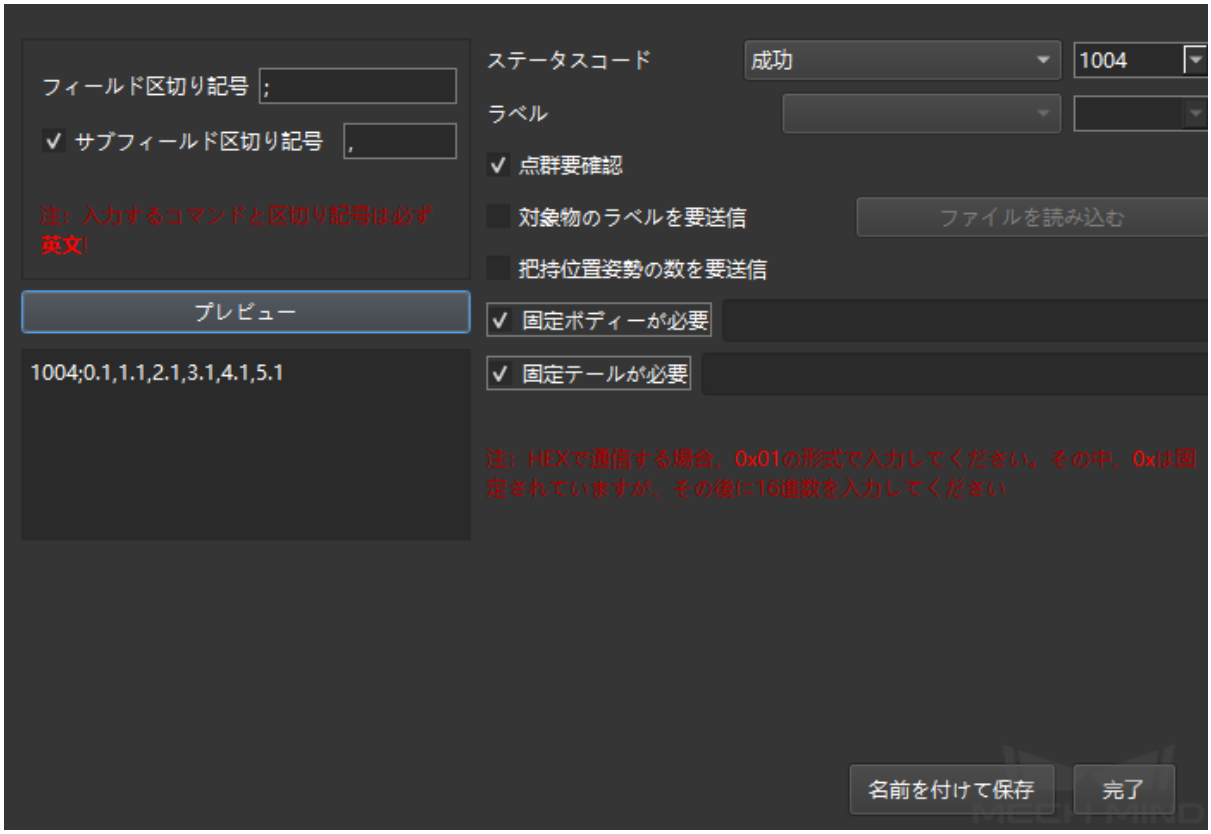
  > **注意：** インデックス位置は 1 からカウントされ、他のフィールドと重複することはできません。例えば、インデックス位置「1」は画像取得コマンドです。

- **ロボット名**：ロボットサービスをマークするための名前です。それを Mech-Viz 中のロボット名と一致させる必要があります。

- **ロボットの自由度**：プロジェクトによってロボットの自由度を選択してください。現在は、4 軸および 6 軸ロボットに対応しています。

送信するストリームの設定-フィールド送信

下図に示すように、この画面で送信する位置姿勢の形式を設定してから、**次へ** をクリックします。



パラメータの説明：

- **フィールド区切り記号** と **サブフィールド区切り記号**：区切り記号の形式を設定します。ASCII の場合は設定が必要です。2 つ以上のフィールドがある場合は、フィールド区切り記号を入力する必要があります。追加情報で別のフィールド区切り記号が必要な場合は、サブフィールド区切り記号も必要です。また、サブフィールド区切り記号の開始範囲と終了範囲を指定できます。
- **ステータスコード**：送信形式を設定します。各ステータスが唯一のステータスコードに対応します。
- **点群要確認**：チェックを入れれば、点群が Adapter によって確認されます。点群がない場合は、対応するステータスコードが出力されます。
- **対象物のラベルを要送信**：対象物のラベルの送信は、Mech-Vision で認識されたラベルを通信先に送信することです。各ラベルは位置姿勢の後に付きます。通信先にとってラベルの文字列を解析するのが不便な場合は、対応のラベルのコードを指定することもできます。その中、すべてのラベルの文字列を含むラベルファイルを読み込む必要があります。また、ラベルファイルの形式は json 配列形式である必要があります。
- **把持位置姿勢の数を要送信**：今回送信する位置姿勢の数を送信します。
- **固定ボディーが必要**：視覚認識に失敗した場合、通信先にメッセージ（エラーコード後のメッセージ）を送信します。

- **固定テールが必要**：チェックを入れれば、送信するデータの後にテールを固定する標識が追加されます。

> **注意:** 通信形式を 16 進数（HEX）に設定する場合、ステータスコード、位置姿勢の数、位置姿勢のタイプを設定する必要があります。

上記の設定をすべて完了したら、**完了** または **名前を付けて保存** をクリックして Adapter プロジェクトを保存します。Adapter プロジェクトを保存した後、*Adapter* **プロジェクトの設定** を参照してそれをデプロイして実行することができます。

Adapter プログラミング開発については、以下の内容をお読みください。

- *Adapter* **プログラミングのガイド**
- *Adapter* **のプログラミング例**

3.4.3 Adapter プログラミングのガイド

本節では、Adapter のプログラミングスタイル規則とプログラミング文法の知識をについて説明していきます。

対象となる読者

本節は、主に Adapter 開発者向けです。

Adapter 開発者は、以下のことを事前に把握しておくことが必要です。

- Python の基本文法
- JSON データ形式

最初に **Adapter プログラミングスタイル規則** を把握しておくことが必要です。詳細については、以下の内容をお読みください。

Adapter プログラミングスタイル規則

Adapter は Python 言語で記述されているため、Adapter 開発者は Python プログラミング規則 を厳守する必要があります。

また、Adapter プログラミングのメンテナンス性と読みやすさを向上させるために、Adapter は本節で記述される規則も厳守する必要があります。

- **命名規則**
- **コメントの規則**
- **ログ規則**

命名規則

1. クラス名にはキャメルケースを使用します。次に例を示します。

```
class AdapterWidget
```

2. クラスのメンバー変数とメンバー関数は、アンダースコアで結合された小文字の単語を使用します。次に例を示します。

```
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. クラスのメンバー変数とメンバー関数がクラス内でのみ使用される場合は、名前の前にアンダースコアを追加して、その名前をクラス外で使用することは推奨されていませんが、クラス外で使用できます。次に例を示します。

```
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. 定数はアンダースコアで結合されたア大文字の単語を使用します。次に例を示します。

```
ADAPTER_DIR = "D:/adapter_for_communication"
```

コメントの規則

コメントを適切に使用する必要があります。表現が複雑な場合、または表現された意味が重要な場合にのみ追加する必要があります。

- 一行コメント

  一行コメントは#で始めます。

- 複数行コメント

  複数行コメントは"""と"""の間に記述します。

- 関数、クラス、またはパッケージ紹介のコメント

  コメントは、関数の下、クラスの下、またはパッケージの先頭に配置されます。次に例を示します。

```
def viz_is_running(self):

    """
    Will be called when find viz is running during starting viz.
    """

class Adapter(QObject):

    """
    Base class which encapsulates Viz/Vision/Hub/Robserver inter faces.
    """

"""
Service base classes.
```

(次のページに続く)

(前のページからの続き)

```
"""

import logging
```

ログ規則

ログは、エラーが発生したときに問題を分析するのに役立ちます。主要関数を呼び出す場合、関数が参照可能なデータを返す場合など、適切な場合にログ情報を出力するのが最適です。

Adapter は、次の 2 つのログ出力方法をサポートしています。

- print：この方法は、ログをコンソールに出力するだけです。実行時のリアルタイム監視には便利ですが、プログラムはエラーが発生するとメッセージが失われます。したがって、実際の生産で使用することはお勧めしません。
- logging：この方法は、ログ表示形式をフォーマットすることをサポートし、ログをログファイルに保存することもサポートします（選択可能な機能）。したがって、この方法を使用することをお勧めします。

次に、Adapter プログラミングに関連する **抽象親クラスインターフェース** と **Util** パッケージを学び、Adapter の最も基本的な親クラスと一般的な関数を理解します。詳細については、以下の内容をお読みください。

抽象親クラスインターフェース

抽象親クラスインターフェースとは、子クラスが親クラスを継承する際に、実際のニーズに応じて書き換えることができる関数を指します。本節では、次の抽象親クラスについて説明していきます。

Communication

通信関連のクラスのソースファイルは、Mech-Center ソフトウェアのインストールパスの ¥src¥interface¥communication.py ファイルに格納されています。

Communication クラス

Communication クラスは通信を司る基本的なクラスであり、一連のインターフェースを提供します。サーバーやクライアントは、このクラスのインターフェース関数を書き換える必要があります。

クラス関数	説明
is_connected()	現在の接続が切断されているかどうかを確認します
set_recv_size()	受信データの長さを設定します（デフォルトは 1024 バイト）
send()	インターフェース関数で、データを送信します
recv()	インターフェース関数で、データを受信します
close()	インターフェース関数で、接続を閉じます
before_recv()	インターフェース関数で、データを受信する前に実際の状況に応じてロジックを追加できます（この関数を書き換えることができます）
after_recv()	インターフェース関数で、データを受信した後に実際の状況に応じてロジックを追加できます（この関数を書き換えることができます）
after_handle()	インターフェース関数で、データを処理した後に実際の状況に応じてロジックを追加できます（この関数を書き換えることができます）

TcpServer クラス

TcpServer クラスは、TCP/IP Socket のサーバーをパッケージ化します。

クラス関数	説明
bind_and_listen()	ポートをバインドします
local_socket()	ネイティブソケット情報を提供します
remote_socket()	リモート ソケット情報を提供します
accept()	クライアント接続を受け入れます
send()	データを送信します
recv()	データを受信します
close()	Socket 接続を終了します
close_client()	クライアント接続を閉じます

TcpClinet クラス

TcpClient クラスは、TCP/IP Socket のクライアントをパッケージ化します。

クラスのプロパティ	説明
is_bind_port	ポートをバインドするかどうか（接続されたクライアントのポートがサーバーによって制限されている場合、この変数は True に設定する必要があります）

クラス関数	説明
send()	データを送信します
recv()	データを受信します
close()	接続を閉じます
set_timeout()	タイムアウトを設定します（単位は秒）
reconnect_server()	サーバーを再接続します
after_connect_server()	インターフェース関数で、サーバーへの最初の接続が成功した後の操作です
after_reconnect_server()	インターフェース関数で、サーバーへの再接続が成功した後の操作です
after_timeout()	インターフェース関数で、タイムアウト後の操作です

Adapter

Adapter 関連のクラスのソースファイルは、Mech-Center ソフトウェアのインストールパスの `¥src¥interface¥adapter.py` ファイルに格納されています。

Adapter 基本クラス

Adapter クラスは、Mech-Viz、Mech-Vision、Mech-Center および Robserver に関連する呼び出しをパッケージ化します。その中には、Mech-Viz の起動、Mech-Viz の停止、Mech-Vision または Mech-Viz ステップパラメータの設定、Mech-Vision 認識の実行などの関数が含まれます。Adapter プログラムが Mech-Viz または Mech-Vision を呼び出す限り、Adapter クラスを継承する必要があります。

Adapter クラスのプロパティを次の表に示します。

クラスのプロパティ	説明
viz_project_dir	現在の Mech-Viz プロジェクトパス
vision_project_name	現在の Mech-Vision プロジェクト名
is_simulate	Mech-Viz をシミュレーションで実行するかどうか
is_keep_viz_state	Mech-Viz を最後に停止した時の状態を保持するかどうか
is_save_executor_data	Mech-Viz アクチュエータのデータを保存するかどうか
is_force_simulate	Mech-Viz を強制的にシミュレーションで実行するかどうか
is_force_real_run	Mech-Viz を強制的に実行するかどうか
code_signal	Mech-Center のメインインターフェースで Adapter メッセージを表示するための信号（エラーコード付きのメッセージ）
msg_signal	Mech-Center のメインインターフェースで Adapter メッセージを表示するための信号（エラーコードのないメッセージ）
i_code_signal	Mech-Center のメインインターフェースで Mech-Interface メッセージを表示するための信号（エラーコード付きのメッセージ）
i_msg_signal	Mech-Center のメインインターフェースで Mech-Interface メッセージを表示するための信号（エラーコードのないメッセージ）
viz_finished_signal	Mech-Viz 実行終了信号（正常終了または異常終了）
connect_robot_signal	ロボット接続/切断の信号
start_adapter_signal	Adapter の起動信号
service_name_changed	Mech-Center メインインターフェースで Mech-Viz および Mech-Vision ステータを表示するための信号
setting_infos	Mech-Center の構成情報
service_name	登録サービス名

Adapter クラスのプロパティを次の表に示します。

クラス関数	説明
on_exec_status_changed()	Mech-Viz および Mech-Vision のステータス情報を受信します
register_self_service()	Adapter サービスを登録します
vision_project_dirs(self):	Mech-Vision プロジェクトのフォルダパスを探します
vision_project_names()	すべての Mech-Vision プロジェクト名を探します
vision_project_names_in_center()	Mech-Center で登録されたすべての Mech-Vision プロジェクト名を探します
is_viz_registered()	Mech-Viz プロジェクトが登録されているかどうかを確認します
is_viz_in_running()	Mech-Viz が登録しているかどうかを確認します
is_vision_started()	Mech-Vision プロジェクトが登録されているかどうかを確認します
find_services()	ビジョンサービスを探します
before_start_viz()	Mech-Viz の起動前に呼び出される関数
after_start_viz()	Mech-Viz の起動後に呼び出される関数
viz_not_registerd()	Mech-Viz の起動後、Mech-Viz プロジェクトが登録されていない場合はこの関数が呼び
viz_is_running()	Mech-Viz の起動後、Mech-Viz が実行している場合はこの関数が呼び出されます
viz_run_error()	Mech-Viz の起動後、Mech-Viz 実行中にエラーが発生した場合はこの関数が呼び出され
viz_run_finished()	Mech-Viz の実行が終了したときに呼び出される関数
viz_plan_failed()	Mech-Viz の計画が失敗したときに呼び出される関数
viz_no_targets()	Mech-Viz の計画に移動点がない場合に呼び出される関数

次のペ

表 3 – 前のページからの続き

viz_unreachable_targets()	Mech-Viz の計画に到達不能な移動点がある場合に呼び出される関数
viz_collision_checked()	Mech-Viz の計画に衝突が検出された場合に呼び出される関数
viz_collision_checked()	Mech-Viz の返信を解析します
wait_viz_result()	Mech-Viz の返信を待ちます
start_viz()	Mech-Viz を起動します
stop_viz()	Mech-Viz を停止します
pause_viz()	Mech-Viz を一時停止します
find_vision_pose()	Mech-Vision プロジェクトをトリガーして撮影します
async_call_vision_run()	Mech-Vision プロジェクトを非同期にトリガーして撮影します
async_get_vision_callback()	Mech-Vision からの結果を非同期的に受信します
deal_vision_result()	Mech-Vision からの結果を処理します
set_step_property()	Mech-Vision のステップパラメータを設定します
set_step_property()	Mech-Vision のステップパラメータを読み取ります
select_parameter_group()	Mech-Vision プロジェクトのレシピテンプレートを選択します
set_task_property()	Mech-Viz のステップパラメータを設定します
read_task_property()	Mech-Viz のステップパラメータを読み取ります
get_digital_in()	DI を取得します
set_digital_out()	DO を設定します
before_start_adapter()	Adapter の起動前に呼び出される関数
start()	Adapter を起動します
close()	Adapter を閉じます
handle_command()	受信した外部コマンドを処理します

TcpServerAdapter クラス

TcpServerAdapter クラスは Adapter を継承し、TcpServer の機能をパッケージ化します。詳細は次の通りです。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
                    self._server.close_client()
                    self.accept()
                    continue
                self._server.after_recv()
            except socket.error:
                logging.warning("Adapter client is closed!")
                self.code_signal.emit(logging.WARNING, CENTER_CLIENT_DISCONNECTED)
                self._server.close_client()
                self.accept()
            except Exception as e:
                logging.exception("Exception occurred when receiving data from␣
↪client: {}.".format(e))
            else:
                try:
                    self.handle_command(cmds)
                    self._server.after_handle()
                except Exception as e:
                    self.msg_signal.emit(logging.ERROR, _translate("messages",
↪"Handle command exception: {}".format(e)))
                    logging.exception("Adapter exception in handle_command(): {}".
↪format(e))

    def close(self):
        super().close()
        self._server.close()

    def before_start_adapter(self):
        super().before_start_adapter()
        self.accept()

    def accept(self):
        if self.is_stop_adapter:
            return
        self.code_signal.emit(logging.INFO, CENTER_WAIT_FOR_CLIENT)
        self._server.accept()
        if self._server.is_connected():
            self.code_signal.emit(logging.INFO, CENTER_CLIENT_CONNECTED)
            self.msg_signal.emit(logging.INFO, _translate("messages", "Client␣
↪address is") + " {}".format(self._server.remote_socket()[1]))
```

TcpClientAdapter クラス

TcpClientAdapter クラスは Adapter を継承し、TcpClient の機能をパッケージ化します。詳細は次の通りです。

```
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
```

(次のページに続く)

(前のページからの続き)

```
    def set_recv_size(self, size):
        self._client.set_recv_size(size)

    def send(self, msg, is_logging=True):
        self._client.send(msg, is_logging)

    def recv(self):
        return self._client.recv()

    def start(self):
        self.reconnect_server(False)
        while not self.is_stop_adapter:
            try:
                self._client.before_recv()
                cmds = self._client.recv()
                if not cmds:
                    self.reconnect_server()
                    continue
                logging.info("Received command from server:{}".format(cmds))
                self._client.after_recv()
            except socket.timeout:
                logging.warning("Socket timeout")
                self._client.after_timeout()
            except socket.error:
                sleep(5)
                self.reconnect_server()
            except Exception as e:
                logging.exception("Exception occurred when receiving from server:
↪{}".format(e))
            else:
                try:
                    self.handle_command(cmds)
                except Exception as e:
                    self.msg_signal.emit(logging.ERROR, _translate("messages",
↪"Handle command exception: {}".format(e)))
                    logging.exception("Adapter exception in handle_command(): {}".
↪format(e))

    def close(self):
        super().close()
        self._client.close()

    def reconnect_server(self, is_reconnect=True):
        self._client.reconnect_server()
        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 クラス

TcpMultiplexingServerAdapter クラスは Adapter を継承しており、主に複数のクライアントの接続に使用されます。詳細は次の通りです。

```
class TcpMultiThreadingServerAdapter(Adapter):
    def __init__(self, address):
        super().__init__()
        self._servers = {}
        self.add_server(address)
        self.sockets = {}
        self.clients_ip = {}
        self.thread_pool = ThreadPoolExecutor(max_workers=4, thread_name_prefix=
↪"tcp_multi_server_thread")
        self.thread_id_socket_dict = {}
        self.set_recv_size()

    def set_recv_size(self, size=1024):
        self.recv_size = size

    def _find_client_ip(self, sock):
        for k, v in self.sockets.items():
            if v == sock:
                return k

    def _find_server(self, sock):
        for k, v in self._servers.items():
            if v == sock:
                return k

    def add_server(self, host_address):
        server = TcpServer(host_address)
        server.bind_and_listen()
        self._servers[server] = server.local_socket()

    def set_clients_ip(self, clients_ip):
        """
            Must be called before start().
            `clients_ip` is a dict(key is client ip, value is client description).
        """
        self.clients_ip = clients_ip

    def add_connection(self, ip_port, sock):
        self.sockets[ip_port] = sock
        logging.info("Add {}, connections: {}".format(ip_port, self.sockets))
        self.msg_signal.emit(logging.INFO, _translate("messages", "The client {}␣
↪gets online.").format(ip_port))

    def del_connection(self, ip):
        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)
```

(次のページに続く)

(前のページからの続き)

```
    def check_read_events(self, rs):
        for s in rs:
            if s in self._servers.values():  # recv connection
                server = self._find_server(s)
                if self.is_stop_adapter:
                    return
                server.accept()
                client_socket, client_addr = server.remote_socket()
                ip_port = "{}:{}".format(str(client_addr[0]), str(client_addr[1]))
                self.add_connection(ip_port, client_socket)
            elif s in self.sockets.values():  # recv data
                client_ip = self._find_client_ip(s)
                if not client_ip:
                    continue
                msg = self.recv_by_s(s)
                if not msg:
                    self.del_connection(client_ip)
                    return
                try:
                    future = self.thread_pool.submit(self.handle_command_thread, s,
↪ msg)
                except Exception as e:
                    logging.exception("Adapter exception in handle_command(): {}".
↪format(e))

    def handle_command_thread(self, s, msg):
        thread_id = threading.get_ident()
        self.thread_id_socket_dict[thread_id] = s
        self.handle_command(msg)
        # del self.thread_id_socket_dict[thread_id]

    def send(self, msg, is_logging=True):
        thread_id = threading.get_ident()
        sock = self.thread_id_socket_dict.get(thread_id)
        len_total = len(msg)
        while msg:
            if sock:
                len_sent = sock.send(msg)
            else:
                for v in self.sockets.values():
                    try:
                        len_sent = v.send(msg)
                    except Exception as e:
                        logging.warning(e)
            if not len_sent:
                logging.warning("Connection lost, close the client connection.")
                return len_sent
            if is_logging:
                logging.info("Server send: {}, len_sent: {}".format(msg, len_sent))
            msg = msg[len_sent:]
        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):
```

(次のページに続く)

(前のページからの続き)

```
        msg = b""
        try:
            msg = sock.recv(self.recv_size)
        except socket.error:
            logging.error("The client is closed!")
        if msg:
            logging.info("Received message: {}".format(msg))
        return msg

    def check_task(self):
        """
            Interface.
        """

    def close(self):
        super().close()
        for server in self._servers.keys():
            server.close()
        for client_ip in self.sockets.keys():
            try:
                self.client_connection(client_ip).close()
                logging.info("Close socket :{}".format(client_ip))
            except Exception as e:
                logging.warning("Close socket error:{}, exception:{}".
↪format(client_ip, e))
        self.sockets = {}

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            avalible_sockets = list(self.sockets.values()) + list(self._servers.
↪values())
            rs, _, _ = select(avalible_sockets, [], [], 0.1)
            self.check_read_events(rs)
            try:
                self.check_task()
            except Exception as e:
                self.msg_signal.emit(logging.ERROR,
                                     _translate("messages", "Handle command␣
↪exception: {}".format(e)))
                logging.exception("Exception when check task:{}".format(e))
                sleep(5)
```

IOAdapter クラス

IOAdapter クラスは Adapter を継承し、DI を周期的に取得する操作をパッケージ化します。詳細は次の通りです。

```
class IOAdapter(Adapter):
    robot_name = None
    check_rate = 0.5

    def __init__(self, host_address):
        super().__init__()
        self.last_gi = 0
```

(次のページに続く)

(前のページからの続き)

```
    def get_digital_in(self, timeout=None):
        return super().get_digital_in(self.robot_name, timeout)

    def set_digital_out(self, port, value, timeout=None):
        super().set_digital_out(self.robot_name, port, value, timeout)

    def _check_gi(self):
        gi_js = self.get_digital_in()
        gi = int(json.loads(gi_js.decode())["value"])
        if self.last_gi != gi:
            self.last_gi = gi
            logging.info("Check GI signal status: {}".format(gi))
        self.handle_gi(gi)

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            try:
                self._check_gi()
            except Exception as e:
                logging.exception(e)
                self.check_gi_failed()
            sleep(self.check_rate)

    def handle_gi(self, gi):
        """
            Interface.
        """

    def check_gi_failed(self):
        """
            Interface.
        """
```

AdapterWidget クラス

AdapterWidget クラスは、Adapter UI をカスタマイズするための親クラスであり、UI をカスタマイズする機能はそれから継承する必要があります。詳細は次の通りです。

```
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

サービス関連のクラスのソースファイルは、Mech-Center インストールパスの ¥src¥interface¥services.py ファイルに格納されています。

NotifyService クラス

NotifyService クラスは次の通りです。

```
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)
```

デフォルトのサービス名は adapter です。プロジェクトで複数の通知サービスが必要な場合は、子クラスで service_name を書き換えて、異なるサービスを区別できます。クラス関数の説明を次の表に示します。

クラス関数	説明
handle_message()	インターフェース関数で、この関数にロジックを実装できるために子クラスはこの関数を書き換えることができます
notify()	メッセージの解析を提供します（通常、子クラスは書き換える必要はありません）

VisionResultSelectedAtService クラス

VisionResultSelectedAtService クラスは次の通りです。

```
class VisionResultSelectedAtService(JsonService):
    service_type = "vision_watcher"
    service_name = "vision_watcher_adapter"

    def __init__(self):
        self.poses = None

    def poses_found(self, result):
        """
            Interface.
        """

    def posesFound(self, request, _):
        logging.info("{} result:{}".format(jk.mech_vision, request))
        self.poses_found(request)

    def poses_planned(self, result):
        """
            Interface.
        """
```

(次のページに続く)

(前のページからの続き)

```
    def posesPlanned(self, request, _):
        logging.info("Plan result:{}".format(request))
        self.poses_planned(request)

    def multiPickCombination(self, request, _):
        logging.info("multiPickCombination:{}".format(request))
```

デフォルトのサービスタイプは vision_watcher で、タイプを変更することはできません。デフォルトの名前は vision_watcher_adapter です。プロジェクトで複数の vision_watcher サービスが必要な場合は、子クラスで service_name を書き換えて、異なるサービスを区別できます。クラス関数の説明を次の表に示します。

クラス関数	説明
poses_found()	インターフェース関数で、パラメータは Mech-Vision からの認識結果です（この関数にロジックを実装できるために子クラスはこの関数を書き換えることができます）
posesFound()	Mech-Vision によって認識されたメッセージを解析します（通常、子クラスを書き換える必要はありません）
poses_planned()	インターフェース関数で、パラメータは Mech-Viz 計画によって選択された視覚位置姿勢です
posesPlanned()	Mech-Viz からの計画メッセージの解析を提供します

RobotService クラス

RobotService クラスは次の通りです。

```
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
```

(次のページに続く)

(前のページからの続き)

```
    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
```

デフォルトのサービスタイプは robot で、タイプを変更することはできません。デフォルトの名前は robot で、子クラスに対応するロボット名に変更する必要があります。また、Mech-Viz の実行中に 1 つの位置姿勢を固定するために、子クラスに jps または pose 値を設定する必要があります。この位置姿勢が経路全体でシーンと衝突しないようにすることに注意してください。クラス関数の説明を次の表に示します。

クラス関数	説明
getJ()	Mech-Viz/Mech-Vision へ関節角度を返します
setJ()	外部で関節角度を設定します（ラジアン単位）
getL()	Mech-Viz/Mech-Vision へツール位置姿勢を返します
getFL()	Mech-Viz/Mech-Vision へフランジ位置姿勢を返します
setL()	外部でフランジ位置姿勢を設定します（四元数の形式、メートル単位）
moveXs()	Mech-Viz が経路を計画した後にこの関数が呼び出され、パラメータには移動点のプロパティが含まれます（Mech-Viz プロジェクトに「DI をチェック」、ブランチ関連のステップなど、事前計画を中断するステップがある場合は、Mech-Viz はこの関数を複数回呼び出します）
stop()	ロボットを停止します（通常は使用されません）
stop()	TCP を設定します（通常は使用されません）
setDigitalOut()	DO を設定します（通常は使用されません）
getDigitalIn()	DI を取得します（通常は使用されません）
switchPauseContinue()	ロボットを一時停止/再開します（通常は使用されません）

OuterMoveService クラス

OuterMoveService クラスは次の通りです。

```
class OuterMoveService(JsonService):
    service_type = "outer_move"
    service_name = "outer_move"
    move_target_type = TCP_POSE
    velocity = 0.25
    acceleration = 0.25
    blend_radius = 0.05
    motion_type = MOVEJ
    is_tcp_pose = False
    pick_or_place = 0
```

(次のページに続く)

(前のページからの続き)

```
    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}
```

デフォルトのサービスタイプと名前は outer_move です。プロジェクトで複数の outer_move サービスが必要な場合は、子クラスで service_name を書き換えて、異なるサービスを区別できます。クラス関数の説明を次の表に示します。

クラス関数	説明
move_target_type()	移動点のタイプ（0:jps、1:tcp_pose、2:obj_pose）
velocity()	移動点の速度（デフォルトは 0.25）
acceleration()	移動点の加速度（デフォルトは 0.25）
blend_radius()	移動点のブレンド半径（デフォルトは 0.05m）
motion_type()	移動点の運動タイプ（'J':moveJ、'L':moveL）
is_tcp_pose()	移動点は TCP であるかどうかを示します
gather_targets()	インターフェース関数で、すべての移動点を取得します（パラメータは現時点でのロボットの関節角度、フランジ位置姿勢および DI 値で、必要に応じて子クラスを変更できます）
add_target()	単一の移動点を追加します（子クラスでこの関数を呼び出して移動点を追加できます）
getMoveTargets()	Mech-Viz は外部移動ステップに実行すると、この関数が呼び出されます（パラメータには現時点でのロボットの関節角度、フランジ位置姿勢、および DI 値が含まれます）

サービスを登録

上記 4 つのカテゴリーに対応するサービスは、登録後のみ使用できます。登録サービス関数は次の通りです。

```
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 パッケージ

Adapter util パッケージは、Mech-Center インストールパスの `¥src¥util` フォルダに格納されています。その中には数多くのモジュールが含まれており、いくつかの一般的な関数を提供します。プログラミングプロセスでは、最初に util パッケージに関数が実装されているかどうかを確認します。関数が実装されている場合は直接使用できます。関数が実装されておらず、より一般的な場合は、それを小さな関数に抽象化して util パッケージに追加できます。

以下は、各モジュールの簡単な紹介です。

database モジュール

database モジュールはデータベースに対応します。Mech-Center は、実行時にデフォルトで mechmind.db のデータベース ファイルを作成します。これは、実行中にログを保存するために使用されます。database モジュールは、SQL ステートメントを実行して 1 つまたはすべてのレコードをクエリする関数を提供します。

json_keys モジュール

json_keys モジュールは、Mech-Center で使用される json キーまたは値の文字列を保存します。これを他のモジュールにインポートして直接使用できます。

message_box モジュール

message_box モジュールは、ポップアップ プロンプトの機能を提供します。ポップアッププロンプトのタイプには information（一般情報）、warning（ワーニング）および critical（重大エラー）が含まれます。

timestamp モジュール

timestamp モジュールは、現在のタイムスタンプを返す関数を提供します。

transforms モジュール

transforms モジュールは、オイラー角から四元数に変換する、四元数からオイラー角に変換する、位置姿勢乗算、対象物の位置姿勢から TCP に変換する、TCP から対象物の位置姿勢に変換する、対象物回転計算などの機能を提供します。サードパーティのライブラリ transforms3d もオイラー角から四元数に変換する、四元数からオイラー角に変換する機能を提供していますが、実際の使用では、transforms3d によって変換された値が間違っている場合があります。実際の計算では、最初に transforms3d ライブラリを使用できますが、結果が正しくない場合は、transforms モジュールが提供するカスタマイズの変換関数を使用できます。

util_file モジュール

util_file モジュールは、一般的に使用される json ファイルなど、読み取りおよび書き込み関数を提供します。

timer モジュール

timer モジュールは、便利な Timer クラスを提供します。タイミング関数が必要な場合は、Timer オブジェクトを生成し、コールバック関数を渡し、start() を呼び出すことができます。使用後、Timer オブジェクトを破棄する必要はありません。プログラムの終了時に自動的に破棄されます。

pose モジュール

pose モジュールは、Mech-Viz の位置姿勢と同じクラスを提供します。これには、並進 (メートル単位) と回転 (四元数の形式)、反転と乗算の操作、および list 間の相互変換が含まれます。さらに、pose モジュールは、ミリメートルからメートル、メートルからミリメートル、ラジアンから度、度からラジアン、四元数からオイラー角、オイラー角から四元数など、位置姿勢の単位変換関数もいくつか提供します。

最後に、要件に従って抽象親クラスインターフェイスを実装し、内部通信（Mech-Vision および Mech-Viz）と外部通信（外部デバイス）を可能にします。詳細については、以下の内容をお読みください。

インターフェースの取得

現在の Mech-Viz プロジェクトで使用されているステップを取得する

現在の Mech-Viz プロジェクトで使用されているステップを取得する関数を以下に示します。

```
def get_viz_task_names(self, msg={}, timeout=None):
    result = self.call_viz("getAllTaskNames", msg, timeout)
    logging.info("Property result: {}".format(json.loads(result)))
    return result
```

get_viz_task_names() を呼び出した後、json 形式の文字列を返します。これは、取得したすべてのステップを示します。

Mech-Viz または Mech-Vision プロジェクトのパラメータを取得する

Mech-Viz または Mech-Vision プロジェクトのパラメータを取得する関数を以下に示します。

```
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
```

msg パラメータに「type」を指定せずに呼び出した場合、すべてのパラメータを取得することを意味します。指定すると、対応するパラメータのみが取得されます。たとえば、get_property_info(msg={"type": "move"}) を呼び出した後、json 形式の文字列を返します。これは、取得された移動ステップパラメータを示します。

Mech-Vision インターフェース

本節では、Mech-Vision を使用するインターフェースについて紹介していきます。次のインターフェースが含まれています。

- 視覚目標点を取得する
- ステップパラメータを設定する
- ステップパラメータを読み取る
- パラメータレシピを切り替える

視覚目標点を取得する

以下に示すように、adapter.py の find_vision_pose() 関数で Mech-Vision の視覚結果を取得します。

```
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 の視覚位置姿勢は通常、四元数の形式で表示される対象物の位置姿勢（obj_pose）で出力されます。ツール位置姿勢を出力することもできますが、Mech-Vision プロジェクトで変換する必要があります。

例

Adapter は対象物の位置姿勢をツール位置姿勢（tcp_pose）に変換する関数を構築して、ロボット側に送信します。場合によっては四元数をオイラー角、ラジアンを度などに変換する必要もあります（変換することは、ロボット側と一致するかどうかによって決定されます）。さらに、Adapter は Mech-Vision によって出力された次のような視覚位置姿勢を確認することもできます。

```
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
```

その中で、vision_result は find_vision_pose() から取得され、呼び出しステートメントは次のとおりです。

```
self.check_vision_result(json.loads(self.find_vision_pose().decode()))
```

vision_result を関数に渡した後、プロジェクトに ROI が設定された場合、まずは空の箱かどうかを判定してから視覚位置姿勢を取得します。視覚位置姿勢が正常であれば、視覚位置姿勢を変換関数（pack_pose()）に渡して送信します。

ステップパラメータを設定する

通常は adapter.py で set_step_property() を呼び出して、Mech-Vision でステップパラメータを動的に設定します。

```
def set_step_property(self, msg, project_name=None, timeout=None):
    return self.call_vision("setStepProperties", msg, project_name, timeout)
```

その中で、msg は特定のステップ名と設定が必要なパラメータを決定します。

例

Mech-Vision のマッチングモデルをワークの種類に応じて動的に設定する必要がある場合は、次の関数を作成して msg を設定できます。

```
def _step_matching_model_cell(step_name, model_type):
    msg = {"name": step_name,
           "values":
          {"modelFile": model_type["ply"],
        "pickPointFilePath": model_type["json"]}}
return msg
```

その中で、step_name は Mech-Vision の設定するステップ名です。model_type は、ワークのタイプに対応するファイルパスです。「ply」と「json」を介して、「.ply」と「.json」で終わるモデルファイルパスを取得し、Mech-Vision の対応するステップパラメータにそれぞれ入力します。

step_name は「Local Matching」の場合、呼び出しステートメントは次のようになります。

```
msg = _step_matching_model_cell("Local Matching", model_type)
self.set_step_property(msg)
```

効果を下図に示します。

ステップパラメータを読み取る

adapter.py で read_step_property() を呼び出して、Mech-Vision のステップパラメータを取得します。

```
def read_step_property(self, msg):
     result = self.call_vision("readStepProperties", msg)
     logging.info("Property result: {}".format(result))
     return result
```

その中で、msg は取得するステップ名とパラメータ値を決定します。関数を作成して msg を書き換えることができます。

例

カメラ IP を取得する場合のサンプルコードを以下に示します。

```
def read_camera_property(self):
     msg = {"type": "Camera",
            "properties": ["MechEye"]}
     property_results = json.loads(self.read_step_property(msg).decode())
     camera_ip = property_results["MechEye"]["NetCamIp"]
```

Mech-Vision プロジェクトにカメラが 1 つしかない場合、タイプ（type）のみに基づいてステップを見つけることができます。Mech-Vision プロジェクトに同じステップが複数あり、特定のステップパラメータのみを取得または設定する場合は、名前（name）でステップを見つけることができます。以下に示すように、read_step_property() を呼び出して json 形式に変換することにより、カメラのすべてのパラメータ値（json 形式) を取得します。

```
Property result:
{
  "MechEye": {
    "NetCamIp": "127.0.0.1",
    "TimeOut": "10",
    "configGroup": "",
  }
}
```

本例では、特定のパラメータフィールド（「MechEye」および「NetCamIp」）に従って、カメラ IP（127.0.0.1）を取得します。

パラメータレシピを切り替える

一部の Mech-Vision プロジェクトのプロセスが同じであるが、特定のパラメータが異なる場合、パラメータレシピを設定する、つまり、adapter.py で select_parameter_group() を呼び出すことにより、異なるプロジェクトに対応するパラメータを切り替える機能を実現できます。

```
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
```

その中で、project_name は Mech-Vision のプロジェクト名で、group_index はレシピ番号です。

例

プロジェクトでレシピを切り替える必要がある場合は、次のサンプルコードを使用して select_parameter_group() 関数を呼び出し、例外を処理します。

```
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
```

その中で、self.vision_project_name は受信された Mech-Vision のプロジェクト名で、model_code-1 は受信されたレシピ番号です。

Mech-Viz インターフェース

本節では、Mech-Viz を使用するインターフェースについて説明していきます。次のインターフェースが含まれています。

Mech-Viz を起動

Mech-Viz を起動する関数は、adapter.py ファイルの Adapter クラスで定義されているため、コード内で start_viz() を直接呼び出すことができます。また、プロジェクトのニーズに応じて self.before_start_viz() と self.after_start_viz() を書き換えることで、Mech-Viz 起動前後のカスタマイズの動作を実現できます。

関数定義

```
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()
```

(次のページに続く)

(前のページからの続き)

```
    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() は、デフォルトで新しいスレッドで Mech-Viz が実行を終了するのを待ちます。この目的は、Mech-Viz の起動以外の他の操作に影響を与えないようにすることです。

次にステップパラメータを動的に設定することを例として、self.before_start_viz() を書き換える方法を示します。

```
def before_start_viz(self):
     self.set_move_offset(x, y, z)
```

Mech-Viz を起動する前に、読み取んだデータに従って、ある移動点の x、y、z 方向のオフセットを設定します。

Mech-Viz を停止

Mech-Viz を停止する関数は、adapter.py ファイルの Adapter クラスで定義されているため、コード内で stop_viz() を直接呼び出すことができます。

関数定義

```
def stop_viz(self, timeout=None):
    if not self.is_viz_registered():
        self.code_signal.emit(WARNING, VIZ_NOT_REGISTERED)
        return False
    self.call_viz("stop", timeout=timeout)
    self.code_signal.emit(INFO, VIZ_STOP_OK)
    return True
```

Mech-Viz を一時停止・再開

Mech-Viz を一時停止・再開する関数は、adapter.py ファイルの Adapter クラスで定義されています。Mech-Viz ソフトウェアの停止ボタンと同じように機能し、シミュレーションのみに使用できます。

関数定義

```
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)
```

ステップパラメータを設定

通常、Adapter クラスの set_task_property() を呼び出して、Mech-Viz でステップパラメータを動的に設定します。

関数定義

```
def set_task_property(self, msg, timeout=None):
    return self.call_viz("setTaskProperties", msg, timeout)
```

その中で、msg は様々なステップに対して様々なパラメータを設定することを決定します。

定点移動

Mech-Viz の実行中に、移動ステップで X、Y、Z のオフセットを微調整する必要がある場合があります。Mech-Viz を制御するメインプログラムには、次の関数を記述できます。

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)
```

その中で、name は移動ステップの名前です。UNIT_PER_METER=1000。通常、x_offset、y_offset および z_offset のデータ単位は mm で、Mech-Viz のデータ単位は m であるため、単位変換には UNIT_PER_METER を使用します。

以下の方法で set_move_offset() 関数を呼び出すと、Mech-Viz の対応する「移動_1」ステップの X、Y、Z オフセットがそれに応じて変更されます。

```
self.set_move_offset("move_1", 100, 200, 300)
```

リストによる移動/グリッドによる移動

通常、事前に Mech-Viz で編集する必要があります。その後、Adapter がロジックに従ってインデックスを変更します。使用方法は、**定点移動** 、**パレタイジング** ステップと同じです。

外部移動

制御を計画するために複数の外部目標位置姿勢を Mech-Viz に送信する必要がある場合、それらは外部移動ステップによって実現できます。外部移動ステップは、JPs、TCP、対象物の位置姿勢の設定に対応しています。使用方法は次の通りです。

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 は外部移動ステップに実行すると、getMoveTargets() が呼び出されます。様々な外部移動ステップは、サービス名によって区別されます。

```
def _register_service(self):
   self.outer_move_service = CustomOuterMoveService()
   self._outer_move_server, port = register_service(outer_move_service, port)
```

Mech-Viz では、外部移動ステップのパラメータを下図に示します。

注意: 外部移動ステップの前に、把持することを示すステップが必要です。そうしないと、「- 対象物が把持されてない場合、対象物の位置姿勢は無効です。」というエラーが報告されます。

パレタイジング

Mech-Viz の実行中に、様々なパレタイジングステップに応じて様々なパラメータを設定する必要がある場合があります。変更が必要なステップは、パレタイジングステップの名前で見つけることができます。Mech-Viz のプロジェクト編集エリアでステップを選択すると、パラメータバーに表示されるすべてのパラメータを変更できます。

例

例えば、カスタマイズのパレットパターンの場合、通常、変更が必要なパラメータは **開始インデックス** および **ファイル名**（**動的に読み込む** にチェックを入れる場合のみに表示）です。したがって、メインプログラムで次の関数を定義できます。

```
def set_stack_pallet(self, name, startIndex, fileName):
    msg = {
        "name": name,
        "values": {
            "startIndex": startIndex,
            "fileName": fileName,
        }
    }
    self.set_task_property(msg)
```

その中で、「startIndex」は **開始インデックス** を示し、「fileName」は **ファイル名** に示します。「curIndex」および「filename」のパラメータ名は、Mech-Viz で定義されています。

以下の方法を使用して set_stack_pallet() 関数を呼び出します。

```
self.set_stack_pallet("common_pallet_1", 2, "re.json")
```

事前計画パレットパターンの場合、通常、変更が必要なパラメータは **開始インデックス** 、**パレットパターン** 、**箱の長さ** 、**箱の幅**、**箱の高さ**、**行の数**、**列の数** および **段数** です。したがって、メインプログラムで次の関数を定義できます。

```
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)
```

設定するパラメータが多いため、通常はパラメータを excel ファイルに書き込み、excel ァイルのデータを読み取って、self.box_data_info に記録します。これは、後で stack_type の値によってインデックス化できます。「startIndex」、「palletType」および「cartonLength」などの名パラメータは Mech-Viz で定義されています。

カスタマイズのパレットパターン クラスと **事前設定したパレットパターン** クラスの msg 値を比較すると、違いが「values」の値にあることがわかります。ステップパラメータを設定する必要がある場合は、対応するパラメータ名と値を「values」に追加するだけで設定できます。他のパレットパターンのステップパラメータ設定については、上記の例をご参照ください。

メッセージ分岐

Mech-Viz は分岐ステップに実行すると、外部信号が出口を設定するのを待ちます。分岐ステップの場合、分岐制御を行うために次の関数を定義できます。

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)
```

その中で、name は分岐ステップの名前を示し、area は出口を示します。出口には、左から右に 0、1、2、… の番号が付けられています。このステップが左端の出口から実行する場合、area=0 になります。Mech-Viz を起動せずに分岐ステップを直接呼び出すと、Mech-Viz は「エグゼキュータはありません」というエラーが報告されます。

カウンター

カウンターステップを使用する場合、通常、計数の合計数と現在の計数を設定する必要があります。このステップを定義して設定するコードは次の通りです。

例

```
def set_counter_property(self, name, count, curCount):
    msg = {"name": name,
           "values": {"count": count, "currentCount": curCount}}
    self.set_task_property(msg)
```

この関数の呼び出し例を以下に示します。

```
self.set_counter_property("counter_1", 5, self.success_stack_num)
```

その中で、self.success_stack_num は正常にパレタイジングされた対象物の数を示します。パレタイジング中に箱が下した場合、人間の介入により、Mech-Viz は停止します。この時点で、Mech-Viz の「カウンター_1」というカウンターステップは、「currentCount」の値を保存しません。Mech-Viz を再起動した後、カウンターステップの現在の計数は、self.success_stack_num によってリセットできます。

ステップパラメータを読み取る

Mech-Viz の実行中に、ステップパラメータを読み取る必要がある場合は、メインプログラムで次の関数を定義できます。

例

```
def read_move_pallet(self, name):
     msg = {"name": name,
            "properties": ["xOffset","yOffset","zOffset", ]}
     return read_task_property(msg)
```

その中で、「name」によって読み取るステップ名を見つけることができます。「properties」の後のリストの値は、必要に応じてパラメータを追加または削除できます。本例では、「xOffset」、「yOffset」、および「zOffset」の値が読み取られるので、3 つの値を「properties」に追加します。呼び出す方法を以下に示します。

```
self.read_move_pallet("move_3")
```

呼び出した後、次の結果が得られます。

```
{'zOffset': -0.23, 'xOffset': -0.12, 'yOffset': -0.15}
```

また、Mech-Viz のステップから得られるパラメータ値は、計画進行時の値であり、実際の実行時の値ではないことにご注意ください。計画は、実行前に実行します（つまり、Mech-Viz は以後の動作を実際実行より前に計画します）。ここで、パレタイジングステップの開始インデックスを例として説明します。メインプログラムで以下のような関数を定義できます。

```
def read_pallet_current_index(self, name):
     msg = {"name": name,
            "properties": ["curIndex"]}
     return read_task_property(msg)
```

開始インデックスの値を呼び出す方法は以下の通りです。

```
self.read_pallet_current_index("common_pallet_1")
```

呼び出すと以下の結果が得られます。

```
{'curIndex': 5}
```

ここで表示される「5」は、5 回の計画が実行されたことを意味します。例えば、箱配置のプロジェクトでは、5 は計画実行中に 5 つの箱が配置されましたが、実際にロボットは 5 箱より少ない箱しか配置しない可能性があります。したがって、このパラメータは実際実行時の値ではなく、計画された値を示します。

TCP を設定

TCP の設定は、Mech-Viz のエンドツールリストで対応するインデックスを指定するだけです。Mech-Viz のエンドツールリストのインデックスは、上から下まで 0、1、2、3、... です。範囲を超えないように注意してください。TCP 関数の設定は次の通りです。

例

```
def set_tcp(self, index):
    msg = {"function": "setTcp", "index": index}
    self.call("executor", msg)
```

実行中のグローバル速度を設定

Mech-Viz の実行中に、ロボットの実行速度を動的に調整する必要がある場合は、メインプログラムに次の関数を定義できます。

例

```
def set_vel(self, vel_scale):
    msg = {"function": "setConfig",
           "velScale": vel_scale / 100, "accScale": vel_scale / 100}
    self.call("executor", msg)
```

速度が 80%に設定されている場合、関数は次のように呼び出すことができます。

```
self.set_vel(80)
```

注意: この関数は、Mech-Viz の起動後に呼び出す必要があります。そうしないと、エラーが報告されます。つまり、このモジュールの呼び出し条件は分岐ステップと一致しています。したがって、分岐ステップで set_vel() を呼び出し、新しいスレッドで set_vel() を呼び出さないようにします。例を以下に示します。

```
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)
```

点群衝突パラメータを設定

点群衝突パラメータを設定します。Mech-Viz に対応するインターフェースは setConfig() であり、これは実行時にグローバル速度を設定するインターフェースと同じですが、設定する情報が異なります。例を以下に示します。

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 の返された値

Mech-Viz の返された値を次の表に示します。

返された値	意味
Finished	正常に実行しました（Mech-Viz プロジェクトに「ビジョン処理による移動」ステップの右の分岐がステップにつなぐ場合は正常に戻ります）
Command Stop	Mech-Viz で停止ボタンを押すか、stop_viz() 関数を呼び出します
No targets	視覚位置姿勢はありません（Mech-Vision からの視覚結果は空であることを意味します）
No proper vision poses	視覚位置姿勢に到達できません（ロボット自体が現在認識された対象物に到達できない、またはそれと衝突したことを意味します）
PlanFail	Mech-Viz の動作計画に失敗しました
SceneCollision	衝突が発生しました

Mech-Viz の返された値に応じて、Adapter 基本クラスは対応するインターフェース関数を提供します。

RobotService インターフェース

本節では、RobotService を使用するインターフェースについて説明していきます。次のインターフェースが含まれています。

RobotServer サービスをシミュレート

RobotServer サービスのシミュレーションは通常、ロボットを主動制御するために使用されますが、RobotServer を介してロボットを制御することはできません。代わりに、RobotService を作成して RobotServer の機能をシミュレートします。RobotService が Mech-Center に登録されると、Mech-Viz はロボット実機と同じように RobotService と通信します。

RobotServer サービスのシミュレーションは、次のシーンで使用されます。

- Eye In Hand モードで、Mech-Vision 計算用のロボット関節角度を取得するシーン。
- ロボット実機からの位置姿勢を RobotService サービスに送信します。

例

Adapter の子クラスの呼び出し方法は次の通りです。

```
def _register_service(self):
        """
        register_service
        :return:
        """
        if self.robot_service:
            return
```

(次のページに続く)

(前のページからの続き)

```
        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 クラスを以下に示します。

```
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() は Mech-Viz および Mech-Vision で現在の関節角度を取得するために使用されます。通常、setJ() によって現在の関節角度を設定してから、getJ() を呼び出します。詳細は次の通りです。

例

```
def getJ(self, *_):
     pose = {"joint_positions": self._jps}
     return pose
```

1. Eye In Hand：ロボットから送信された関節角度を setJ() に書き込みます。

   ```
   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))
   ```

   その中、jps はロボットから送信された関節角度データで、getJ() によって呼び出されるように self._jps に割り当てられます。getJ() はラジアン形式のデータを取得する必要があるため、ここでの単位変換には注意が必要です。

2. Eye To Hand：ロボットの現在の関節角度を設定する必要はありませんが、ロボット実機の状態をシミュレートするには、最初のポイント (通常は初期位置) として安全な目標点を設定する必要があります。そうしないと、乱数が割り当てられますので、エラーが発生しやすくなります。

   ```
   def getJ(self, *_):
        return {"joint_positions": [1.246689,-0.525868,-0.789761,-1.330814, 0.
   ↪922581, 4.364021]}
   ```

getFL

getFL() は、画像撮影時のロボットのフランジ位置姿勢を提供するために使用されます。Eye In Hand モードでフランジとカメラの位置関係はキャリブレーションされますが、最終的に使用されるデータはベース座標系のデータです。したがって、画像撮影時のロボットのフランジ位置姿勢が必要です。

```
def getFL(self, *_):
    return {"flange_pose": self.pose}
```

Eye In Hand 方式で視覚位置姿勢を提供する場合は、次の点に注意してください。

1. ロボットが画像撮影時の JPs を返す場合、RobotService で setJ() を直接呼び出すことができます（ラジアン単位）。この関数は [] を返します。
2. ロボットが位置姿勢を返す場合、次の操作を行う必要があります。
   - Mech-Vision プロジェクトの「extri_param.json」外部パラメータファイルの「is_eye_in_hand」が true であることを確認します。
   - RobotService の setFL() を呼び出します（メートル単位、四元数形式の位置姿勢）。

moveXs

Adapter によって作成された RobotService サービスは、moveXs() を使用して、Mech-Viz によって計画された経路点データを受け取ります。詳細は次の通りです。

関数定義

```
def moveXs(self, params, _):
     with self.lock:
         for move in params["moves"]:
             self.targets.append(move)
     return {"finished": True}
```

その中、params は Mech-Viz プロジェクトのすべてのパラメータを渡します。params["moves"] を介して、ビジョン処理による移動、相対移動など、すべての移動点の位置姿勢を取得できます。位置姿勢はデフォルトで関節角度の形式で返され、その後の呼び出しのために位置姿勢が self.targets に渡されます。

例

通常、この関数は通知と併用する必要があります。Adapter は通知メッセージを受け取ると、self.targets を関数に送信して変換し、パッケージ化してロボットに送信します。例を以下に示します。

```
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)
```

通知メッセージが「started」の場合、目標点リストをクリアします（Mech-Viz のエラー中断により、前後の 2 つの移動点が重なってしまうのを防ぐため）。通知メッセージが「finished」の場合、目標点リストを pack_move 関数に渡します（データ統合送信のため）。

```
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)
```

現場の必要に応じて、Mech-Viz のすべての移動点を送信するか、index に従っていくつかの点を選択して送信することができます。pack_move() は通常、ロボットの各ブランドが必要とするフォーマットに従ってデータを統合します（通常、通信プロトコルで事前に設定される）。

その他のインターフェース

本節では、Adapter を使用するための他のインターフェースについて説明していきます。次のインターフェースが含まれています。

通知サービス

Mech-Viz プロジェクトが特定の分岐または特定のステップに実行されたときに、Adapter プログラムの対応する関数を呼び出したい場合は、Mech-Viz に通知ステップを追加できます。

例

たとえば、Adapter に把持された対象物の数を 1 ずつ増やす関数を記述した場合、デパレタイジングプロセスの最後のステップの後に通知ステップを追加できます。ここで到達すると、Adapter をトリガーして、対応する関数を呼び出すことができます。この機能の実装例を以下に示します。

1. NotifyService を継承するクラスを作成します。

```
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()
```

   この通知は、次の機能を実装できます。Mech-Viz は「Success」を送信する通知ステップに実行すると、Adapter は update_success_num() 関数を呼び出します。「Fail」を送信する通知ステップに実行すると、Adapter は update_fail_num() 関数を呼び出します。

2. Mech-Viz メインプログラムを制御するクラスに NotifyService クラスをインスタンス化してこのサービスを登録します。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
    def update_success_num(self):
        # the num of unstack successfully plus 1
        self.success_num += 1

    def update_fail_num(self):
        # the num of unstack fiplus 1
        self.fail_num += 1
```

3. Mech-Viz で必要に応じて、対応する通知ステップを追加します。

   通知ステップで **Adapter 名** および **メッセージ** を入力する必要があります。これら 2 つの値は、上記の NotifyService クラスの service_name および msg の値と一致する必要があります。

   プログラムが実行されると、service_type と service_name が Mech-Center に表示されれば、通知サービスが正常に登録されました。

VisionWatcher サービス

Mech-Vision の実行が終了すると、vision result:{'noCloudInRoi': False, 'function': 'posesFound', 'vision_name': 'TJTvision-3'}などのいくつかの結果が出力されます。一部の異常な状況では、Adapter は VisionWatcher サービスを介してエラー情報を送信することができます。

例

1. VisionResultSelectedAtService を継承するクラスを作成します。

```
from interface.services import VisionResultSelectedAtService, register_service

class VisionWatcher(VisionResultSelectedAtService):
```

(次のページに続く)

(前のページからの続き)

```
    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()
```

子クラス VisionWatcher は、親クラスの poses_found() 関数を書き変える必要があるため、Adapter で send_err_no_cloud()(エラー情報を送信する関数) を呼び出して poses_found() で書き換えられます。実行中に、Mech-Vision によって返された視覚位置姿勢の値は、poses_found() の result パラメータに渡されます。

2. Mech-Viz メインプログラムを制御するクラスに VisionWatcher クラスをインスタンス化します。

```
class MyClient(TcpClientAdapter):
    def __init__(self, host_address):
        super().__init__(host_address)
        self._register_service()
    def _register_service(self):
        self.robot_service = VisionWatcher(self.send_err_no_cloud)
        self.server, port = register_service(self.hub_caller, self.robot_
↪service)

    def send_err_no_cloud(self):
        # send no cloud error message
        self.send("12,NoCloudErr,done".encode())
```

VisionWatcher クラスのインスタンス中に、send_err_no_cloud() 関数をパラメータとして VisionWatcher() に渡します。点群のない場合は、poses_found() のロジックに従ってエラーメッセージを送信する関数が呼び出されます。

プログラムが実行されると、登録されたサービスが Mech-Center に表示されれば、VisionWatcher サービスが Adapter によって正常に登録されました。

3.4.4 Adapter のプログラミング例

Adapter のプログラミング文法とプログラミングロジックを理解したら、本節で提供されているサンプルプログラムを参考にして Adapter プログラムを作成することができます。

- *Mech-Vision* を使用して視覚位置姿勢を送信する

Mech-Vision を使用して視覚位置姿勢を送信する

本節では、Mech-Vision のみを使用してし視覚位置姿勢を送信する Adapter サンプルプログラムについて詳しく説明していきます。主に以下の内容が含まれます。

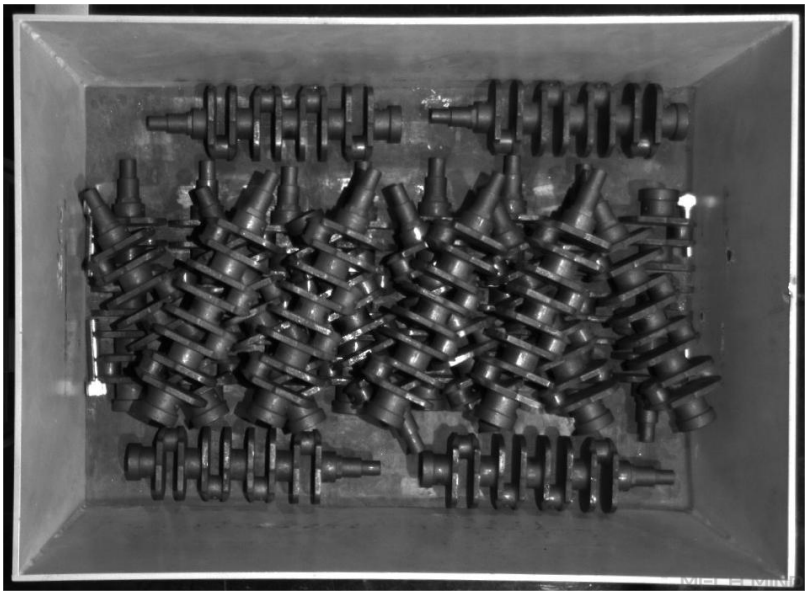
- 背景の紹介
- 通信ソリューション
- 通信メッセージのフォーマット

背景の紹介

本例は、クランクシャフト供給の応用シーンに適しています。カメラは箱の上方のスタンドに取り付けられ、Mech-Vision は画像を撮影して把持可能な部品の座標をロボット側に出力します。

本例では、Mech-Vision に組み込みの「大型の非平面形状部品」サンプルプロジェクトを使用して説明していきます。Mech-Vision で **ファイル ‣ サンプルプロジェクトを閲覧 ‣ 部品のロード・アンロード** をクリックして、このプロジェクトが表示されます。

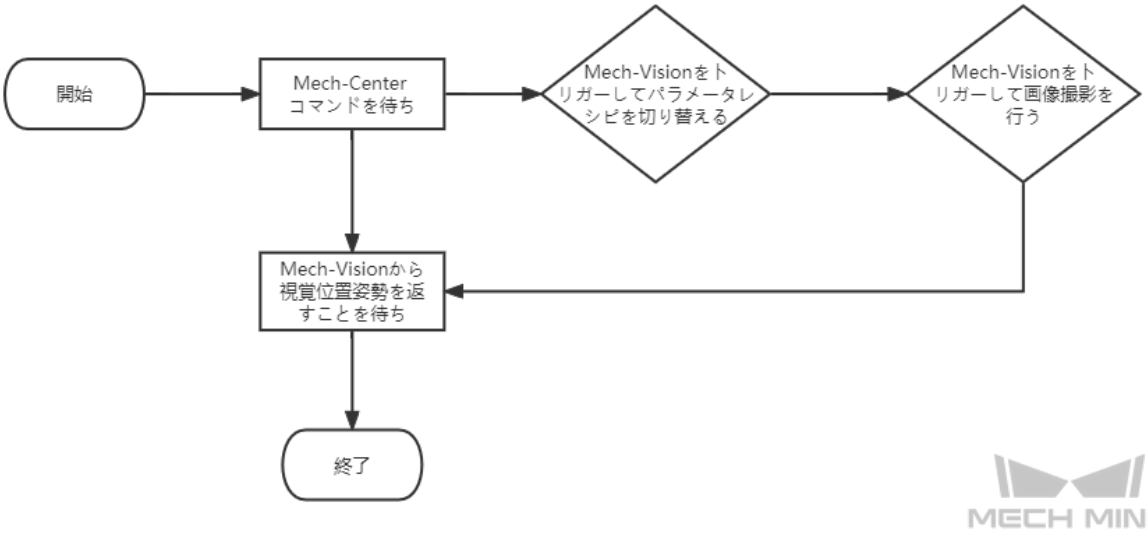
このプロジェクトは 3D モデルマッチングのアルゴリズムを使用し、異なるワークに対して異なるモデルファイルと把持位置姿勢を設定する必要があります。したがって、Mech-Vision でパラメータレシピを設定する必要があり、ロボット側から画像撮影コマンドを送信する際にレシピ番号（ワーク番号）を設定します。レシピ番号の設定や表示方法については、パラメータレシピ をご参照ください。



通信ソリューション

ロボットは、TCP/IP Socket プロトコルを使用して Mech-Mind ビジョンシリーズソフトウェアがインストールされた産業用 PC と通信します。通信フォーマットは ASCII 文字列で、英語のコンマ（,）をデータ区切りとして使用されます。その中、ビジョン関連ソフトウェアはサーバーとして使用され、ロボットはクライアントとして使用されます。

通信のプロセスは下図に示します。



通信プロセスは、次のように詳細に説明されています。

1. Mech-Center は、ロボットが画像撮影コマンド `P` とレシピ番号を送信するのを待ちます。
2. Mech-Center は、Mech-Vision をトリガーしてパラメータレシピを切り替えます。
3. Mech-Center は、Mech-Vision をトリガーして画像撮影と認識を行います。
4. Mech-Vision が画像を撮影して認識に成功すると、ステータスコードと視覚位置姿勢を Mech-Center に返します。
5. Mech-Center は、ステータスコードと視覚位置姿勢をロボットに返します。

注釈: ロボットの把持を容易にするために、Mech-Center は把持する部品の座標をロボットの TCP 座標に変換します。

通信メッセージのフォーマット

詳細は下記の通りです。

	画像撮影コマンド	部品番号
送信（ロボット -> 産業用 PC）	P	「1~100」範囲にある整数です
	ステータスコード	**視覚位置姿勢（TCP 座標）**
受信（産業用 PC -> ロボット）	「0~4」範囲内にある整数です。0-正常認識；1-エラーのコマンドコード；2-Mech-Vision プロジェクトが登録されない；3-視覚位置姿勢なし；4-点群なし	カンマ「,」で区切られた 6 つの浮動小数点データ (x,y,z,a,b,c の形式)

注釈: 応答メッセージの長さは固定です。応答メッセージのステータスコードが異常コード（1~4）の場合、視覚位置姿勢データは 0 で埋める必要があります。

通信メッセージの例

リクエストメッセージ

```
P,1
```

通常応答時のメッセージ

```
0,1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517
```

> **注釈:** 本例では、Mech-Vision は正常に認識し、「1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517」の TCP 座標を返します。

異常応答時のメッセージ：エラーのコマンドコード

```
1,0,0,0,0,0,0
```

異常応答時のメッセージ：Mech-Vision プロジェクトが登録されない

```
2,0,0,0,0,0,0
```

異常応答時のメッセージ：視覚位置姿勢なし

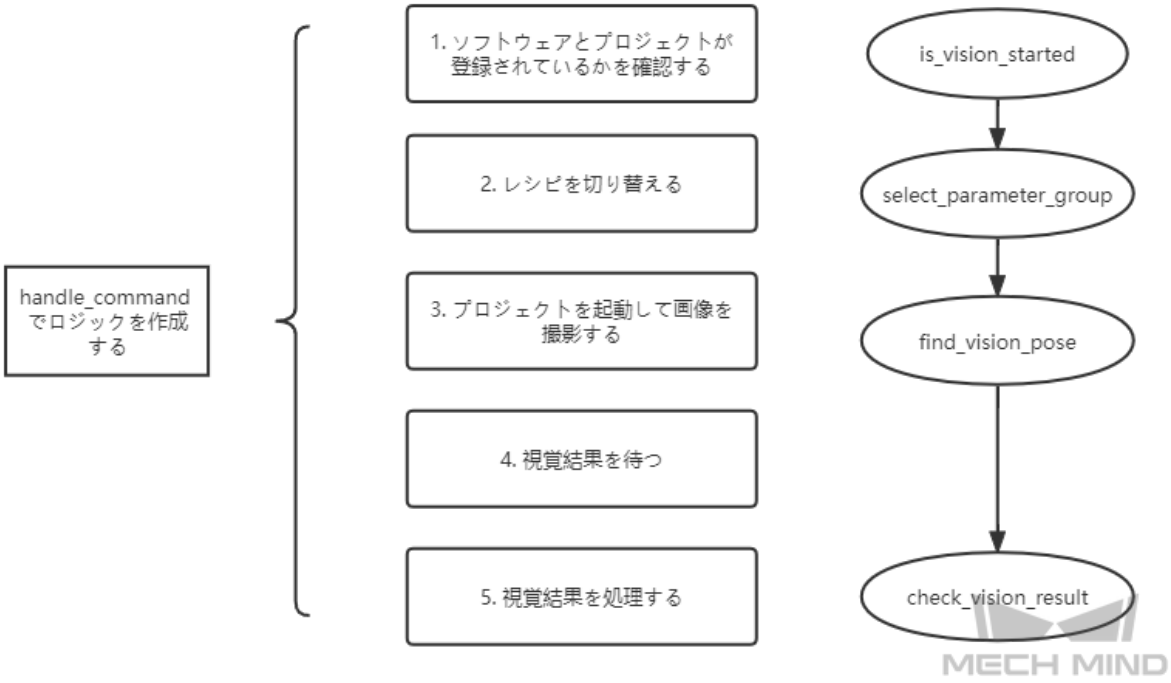
```
3,0,0,0,0,0,0
```

異常応答時のメッセージ：点群なし

```
4,0,0,0,0,0,0
```

プログラミングのアイデア

プロジェクトの要件を満たすために、本例では、下図に示すようなアイデアに従って Adapter プログラムを作成します。



上記の図は、サンプルプログラムのメッセージ処理ロジックのみを示しています。以下では、サンプルプログラムについて詳しく説明します。

サンプルプログラムの詳細な説明

注釈: クリックして Adpater サンプルプログラム をダウンロードします。

Python パッケージをインポートする

Adapter プログラムが依存するすべてのモジュールをインポートします。

```
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
```

クラスを定義する

「TcpServerAdapter」親クラスを継承する「TestAdapter」サブクラスを定義します。

```
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)
```

注釈: 本例では、Adapter プログラムを TCP/IP Socket 通信用のサーバーとして定義します。

受信コマンドと処理ロジックを設定する

リクエスト（画像撮影コマンド、パラメータレシピ含む）を受け取る処理ロジックを設定します。

```
# Receive command _create_received_section
def handle_command(self, cmds):
    photo_cmd, *extra_cmds = cmds.decode().split(',')
    recipe = extra_cmds[0]
    # Check command validity _check_cmd_validity_section
    if photo_cmd != 'P':
        self.msg_signal.emit(logging.ERROR, 'Illegal command: {}'.
↪format(photo_cmd))
        self.send(('1' + '' + '').encode())
        return
    # Check whether vision is registered _check_vision_service_section
    if not self.is_vision_started():
        self.msg_signal.emit(logging.ERROR, 'Vision not registered: {}'.
↪format(self.vision_project_name))
        self.send(('2' + '' + '').encode())
        return
    # Change TODO parameter "extra_cmds" according to actual conditions
    sleep(0.1)  # wait for a cycle of getting in Vision
    # _check_vision_result_function_section

    try:
        result = self.select_parameter_group(self.vision_project_name,␣
↪int(recipe) - 1)
        if result:
            result = result.decode()
            if result.startswith("CV-E0401"):
                self.send(('5' + '' + '').encode())
                return
            elif result.startswith("CV-E0403"):
                self.send(('5' + '' + '').encode())
                return
            raise RuntimeError(result)
    except Exception as e:
        logging.exception('Exception happened when switching model: {}'.
↪format(e))
        self.send(('5' + '' + '').encode())
        return
    self.show_custom_message(logging.INFO, "Switched model for project␣
↪successfully")
```

(次のページに続く)

(前のページからの続き)

```
    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())
```

注釈: 「handle_command」関数は、TCP/IP Socket サーバーがメッセージを受信するための処理エントリとして使用されます。

Mech-Vision からの視覚結果のチェックを定義する

Mech-Vision によって出力された視覚結果を確認するように Adapter を設定します。

```
# 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')
```

視覚位置姿勢の出力形式を設定する

視覚位置姿勢が外部への出力形式を設定します。

```
# Pack pose _pack_pose_section
def pack_pose(self, poses, labels, at=None):

    pack_count = min(len(poses), 1)
    msg_body = ''
    for i in range(pack_count):
        pose = poses[i]
        object2tcp(pose)
        t = [p * 1000 for p in pose[:3]]
        r = [math.degrees(p) for p in euler.quat2euler(pose[3:], 'rzyx')]
        p = t + r
        self.msg_signal.emit(logging.INFO, 'Sent pose: {}'.format(p))
        msg_body += ('{:.4f},' * (len(p) - 1) + '{:.4f}').format(*p)

        if i != (pack_count - 1):
            msg_body += ','
```

(次のページに続く)

(前のページからの続き)

```
    return '{},'.format(0) + msg_body + ''
```

Adapter の閉じる操作を定義する

Adapter を閉じる方法を定義します。

```
def close(self):
    super().close()
```