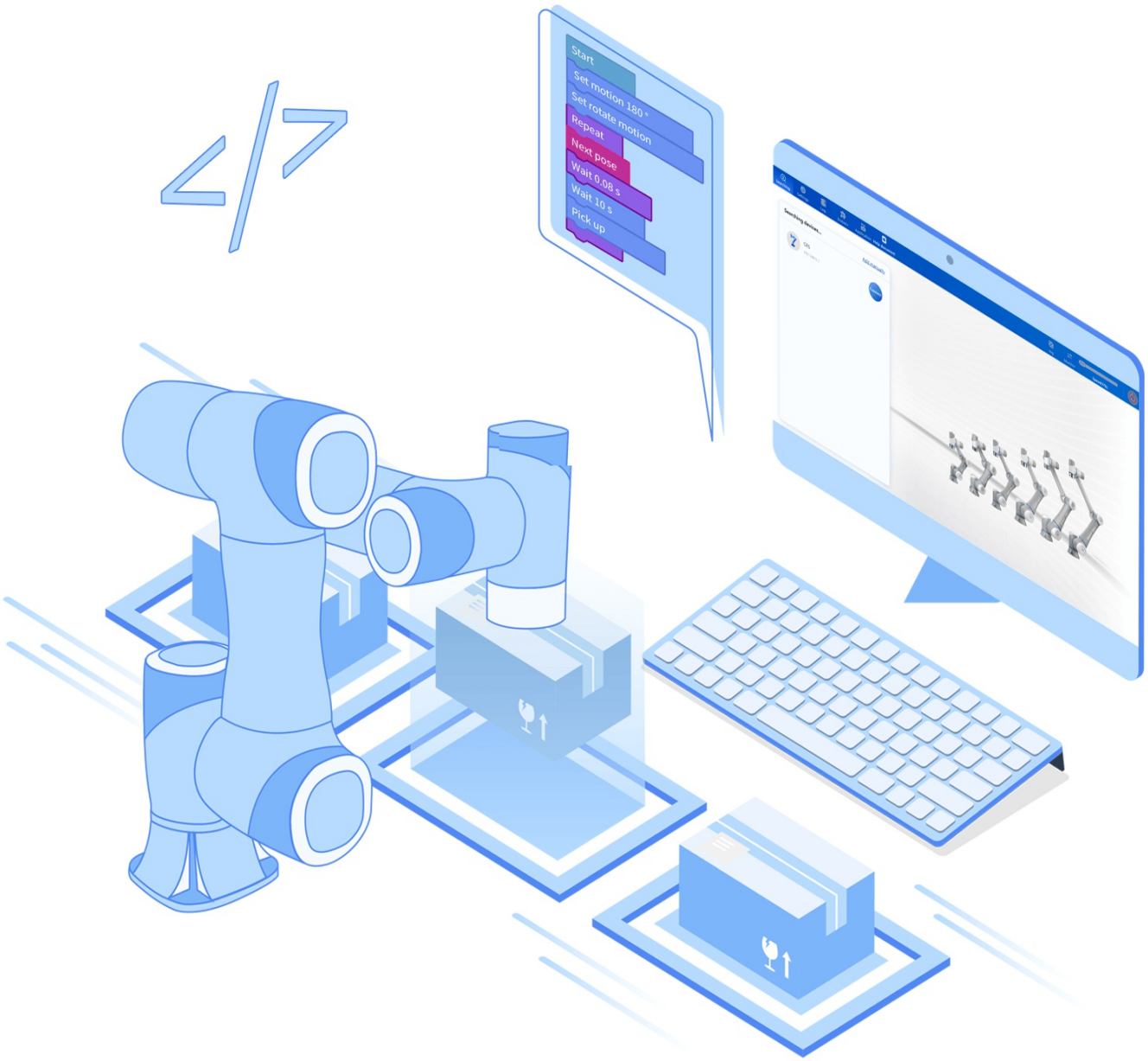
DobotStudio Proユーザーマニュアル
（6軸ロボット）



Shenzhen Yuejiang Technology Co.,Ltd.|China

目次

ElapsedTime

Systime

前書き

目的

本マニュアルでは、ロボット制御ソフトウェアDobotStudio ProによりCRAシリーズとMagician E6ロボットの機能および操作方法などを説明し、ユーザがDobotStudio Proをよく理解し、それを使用できるようにするためのものです。

対象読者

本マニュアルは下記の方を対象としています。

- 一般ユーザ
- セールスエンジニア
- 設置およびデバッグエンジニア
- テクニカルサポートエンジニア

改定記録

日時	バージョン番号	改定記録
2023/11/28	V4.5.0	DobotStudio Proバージョン4.5.0に更新しました
2023/09/13	V4.4.1	初版、対応ソフトウェアバージョンはDobotStudioPro 4.4.1

説明
このドキュメントは翻訳であり、翻訳ミスが含まれている可能性があります。オリジナルバージョン「DobotStudio Pro V4.5.0 User Guide_en」のみを基準にしてください。

標識の定義

本マニュアル中の下記標識は、それぞれ以下の意味を表しています。

標識	説明
危険	回避しなければ、人員の死亡または重傷を引き起こす可能性が高い潜在的危険性があることを示します
警告	回避しなければ、人員の軽傷やロボットの破損などを引き起こす可能性が中程度または低い潜在的危険性があることを示します
注意	潜在的危険性があることを示します。これらのテキスト内容を無視した場合、ロボットの破損やデータの損失、予期せぬ結果を招く可能性があります
説明	本文への強調または補足である追加情報を示します

1 ここから

DobotStudio Proへようこそ。DobotStudio Pro は、Dobotロボット向けにDobotが開発した制御ソフトウェアで、機能がシンプルで使いやすく、実用性が高く、インターフェースがシンプルで分かりやすいという特長があり、ユーザがDobotロボットの使い方を迅速にマスターするのに役立ちます。

本マニュアルでは、DobotStudio Proを使用してCRAシリーズのロボットを制御する方法について主に説明します。

DobotStudio Proでサポートされているオペレーティングシステムは次のとおりです。

PC端末

- Win7 64bit
- Win10
- Win11

移動側

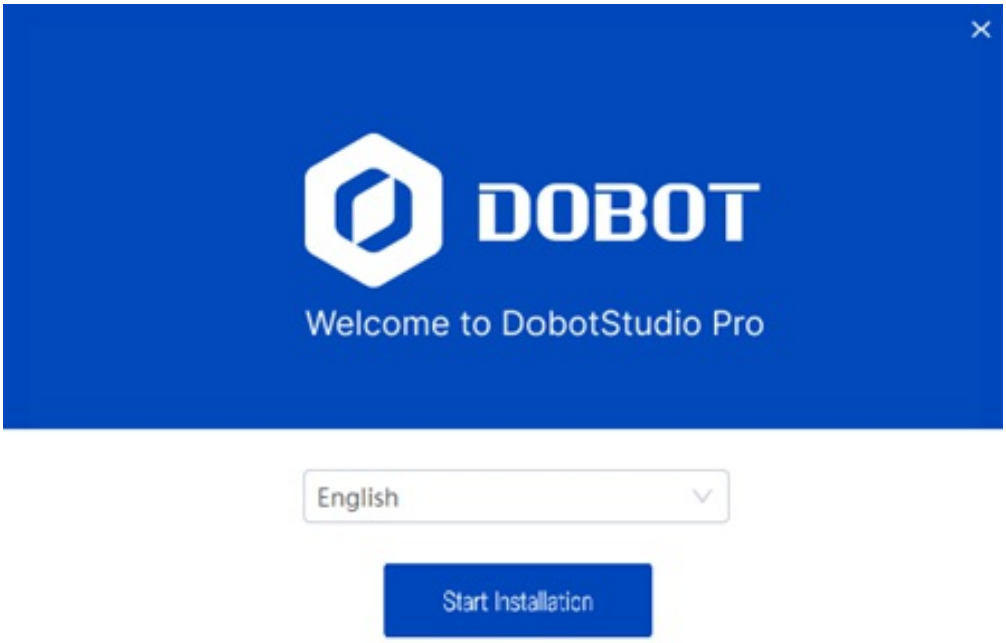
- Android 10以降

ソフトウェアのインストール

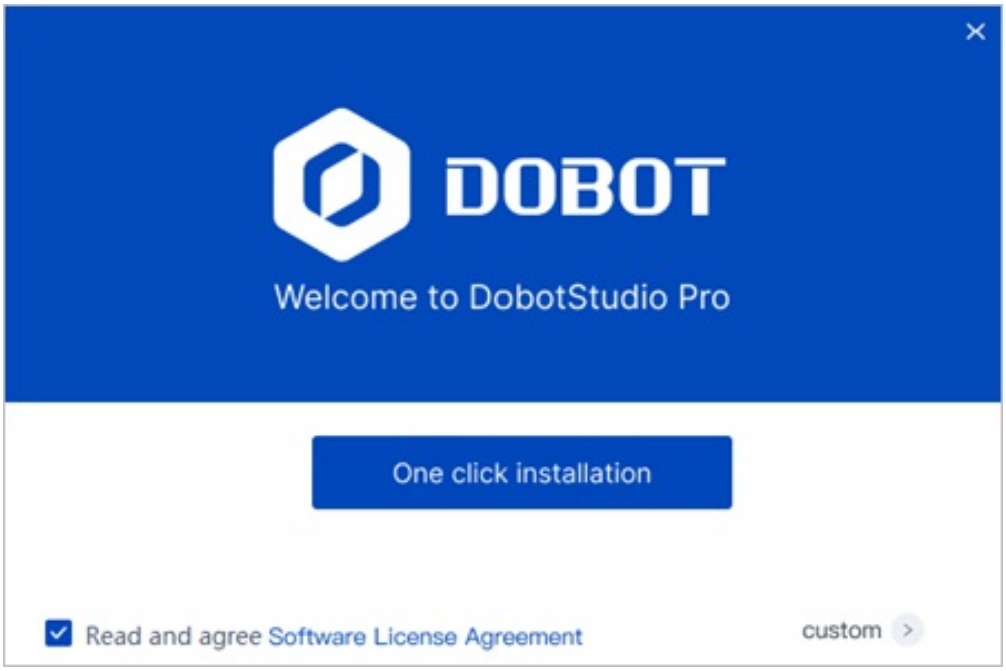
PC端末

公式サイトから最新のDobotStudio Proインストールパッケージをダウンロードしてインストールしてください。インストール手順は次のとおりです。

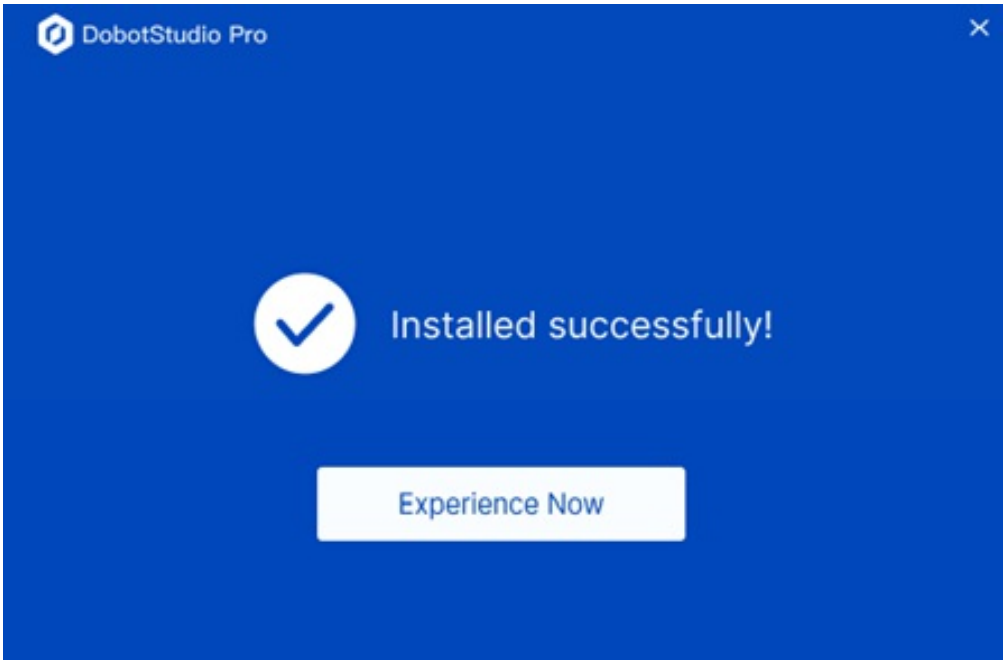
1. インストールパッケージをダブルクリックして開き、インストール言語を選択し、クリックします。次へ。



2. インストール画面でクリックします。ワンクリックインストール、またはカスタマイズオプションでインストールパスを設定してからインストールを開始します。



3. ソフトウェアのインストールが成功したら、プロンプト画面でクリックします。すぐに体験するソフトウェアを直接開くことができます。



移動側

- Android：公式ウェブサイトから最新のDobotStudio Proインストールパッケージをダウンロードしてインストールしてください。

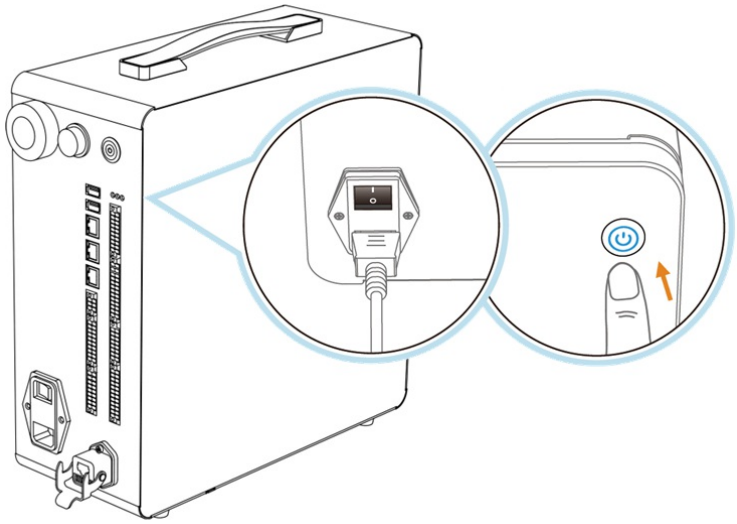
DobotStudio Proのほとんどの機能は、ロボットに接続した後でしか使用できません。ロボットに接続していない場合、システム設定で表示言語を設定したり、モバイル端末のクラッシュログをアップロードしたりすることができます。

2 電源を入れて接続する

2.1 電源を入れる

CRAシリーズ

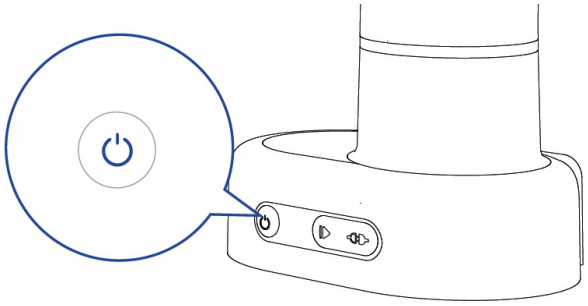
制御パネルのインストールと配線を行い、外部電源がオンになったら、電源接続口の上のスイッチを「|」に押し、次に制御パネルの上の丸いボタンを短く押します。指示灯が青色に長く点灯すると、制御パネルが起動したことを示します。



説明：
制御パネルの起動中、LAN2ポートのオレンジ色のライトが点滅し、制御パネル内蔵のWiFiルーターが初期化中であることを示します。LAN、WiFi、教示器はロボットに接続できません。初期化プロセス全体を通じて、ライトの点滅速度が徐々に速くなり、最終的に消灯します。これは初期化が完了したことを示し、この時点でユーザはロボットへの接続を試みることができます。

Magician E6

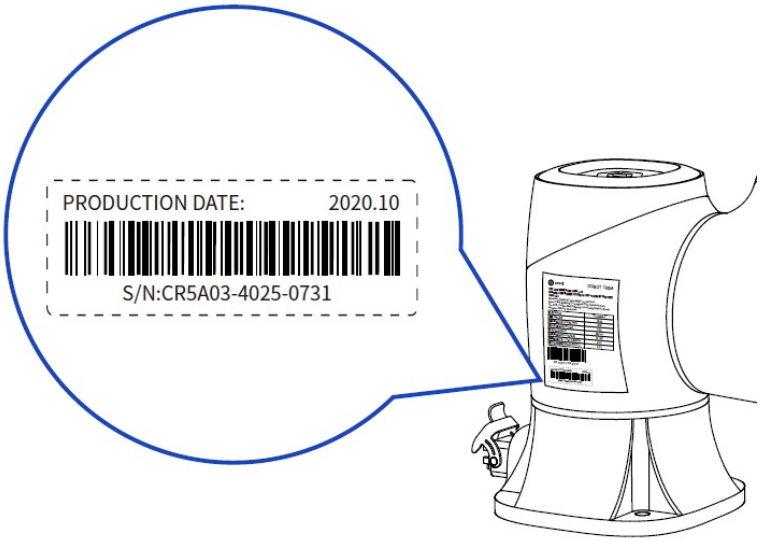
ロボットの設置と配線が完了し、外部電源がオンになったら、ロボットのベースの丸いボタンを短く押します。ロボットの指示灯が青色に長く点灯すると、ロボットが起動したことを示します。



2.2 ワイヤレス接続

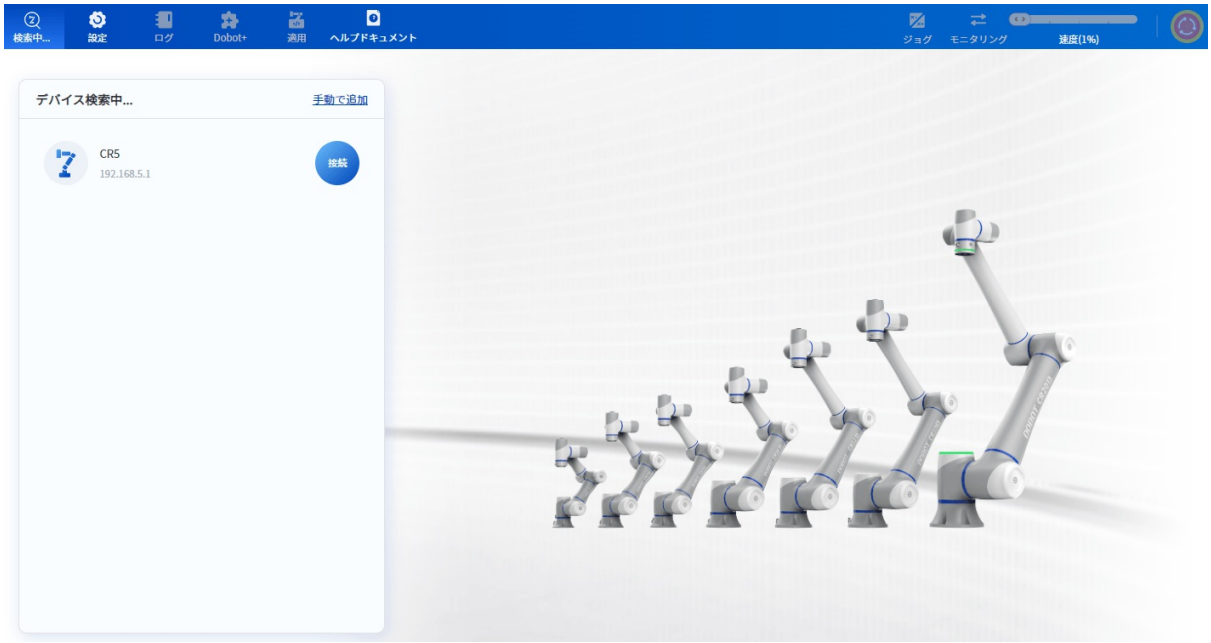
> 説明：
> Magician E6ロボットをワイヤレスで接続する場合は、別途で購入したワイヤレスモジュールをロボットのベースのUSBポートに挿入してください。

PCまたはタブレットでロボットのWiFiを検索し、接続します。WiFiのデフォルトのSSIDは「Dobot_WIFI*XXX*」です。*XXX*はロボットのS/N番号（機械アームのベースに貼られたプレートに記載されています。以下の図はCR5Aを例にしていますが、他のモデルも同様です）。初期WiFiパスワードは：1234567890です。WiFiのSSIDとパスワードは通信設定で変更可能です。



WiFi接続時のロボットのIPは192.168.201.1です。

PC端末



DobotStudio Proを開くと、ソフトウェアが自動的に接続可能なロボットを検索し、検索結果を画面に表示します。

接続したいロボットの右側の 

ボタンをクリックしてロボットに接続します。

移動側

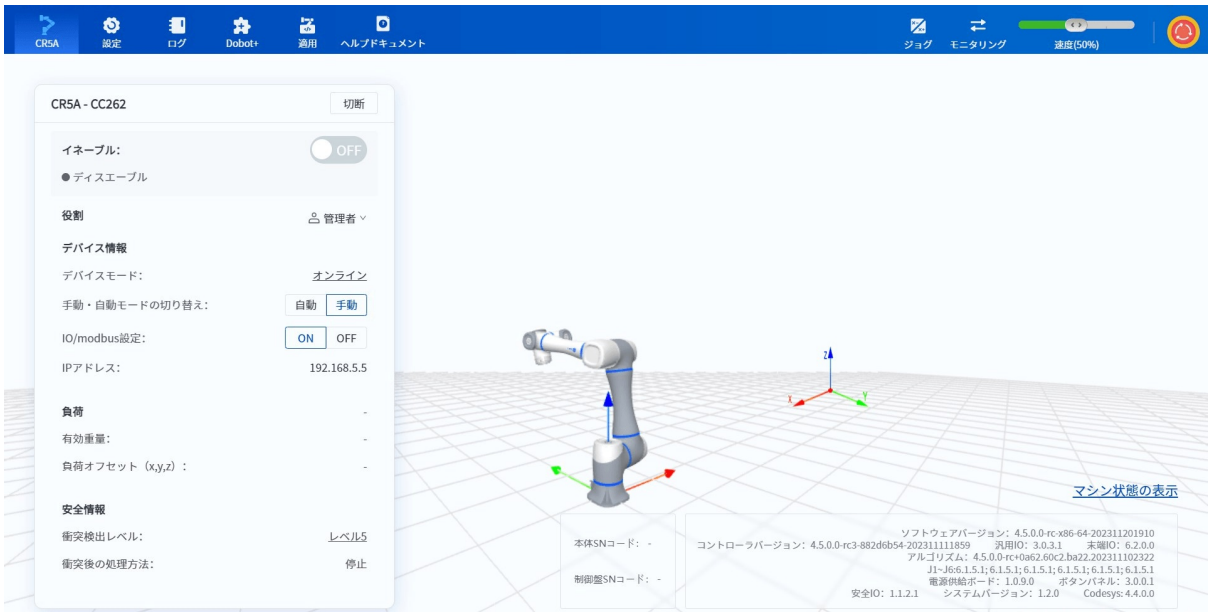
画面の左上の 

ボタンをクリックしてロボットに接続します。

説明：
ソフトウェアがロボットに接続する際、ソフトウェアのバージョンとコントローラのバージョンが一致するかどうかをチェックします。

- バージョン番号の最初の3つが一致していれば、ロボットに正常に接続します。
- バージョン番号の最初の2つが一致し、3つ目が一致しない場合、ソフトウェアは機能が互換性がない可能性があるというポップアップ警告を表示し、その後ロボットに接続します。
- バージョン番号の最初の2つが一致しない場合、ソフトウェアはバージョンが一致しないというポップアップ警告を表示し、自動的に接続を切断します。

ロボットの接続に成功すると、ソフトウェアの画面が更新され、接続されたロボットの関連情報と3Dモデルが表示されます。ロボットの名前の右側の切断ボタンをクリックすると、現在のロボットとの接続が切断されます。



2.3 有線接続

有線接続は**PC**側でのみサポートされています。

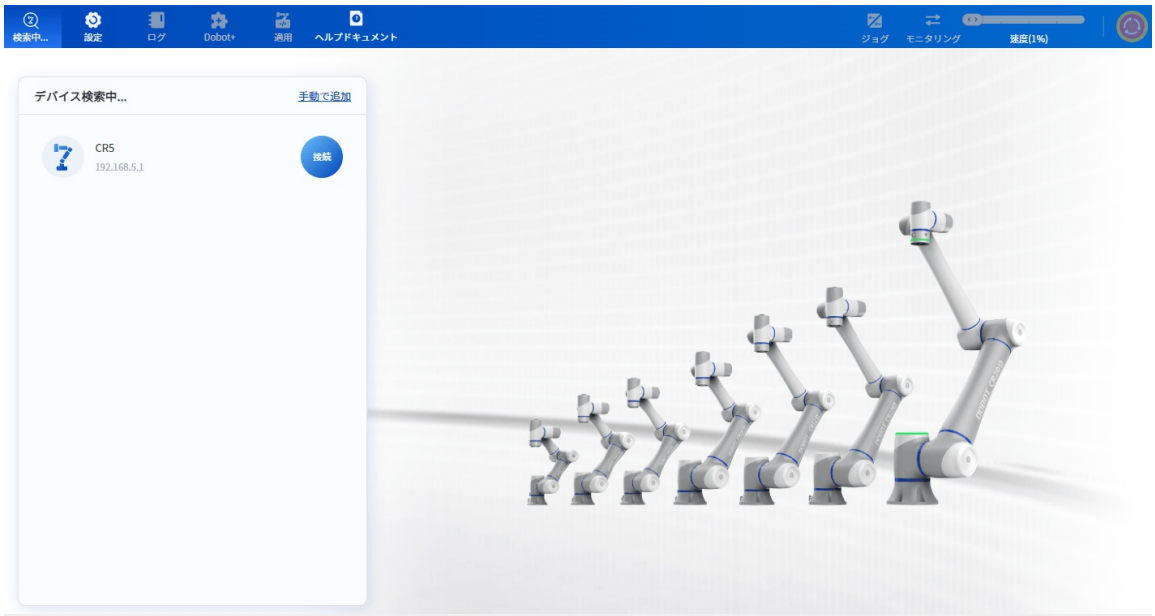
ネットワークケーブルの一方をコントローラのLANポートに、もう一方をPCに接続し、次にPCのIPアドレスを変更して、コントローラのIPアドレスと同じネットワークセグメントにします。LAN1ポートのデフォルトのIPは192.168.5.1で、LAN2ポートのデフォルトのIPは192.168.200.1です。LAN1ポートのIPアドレスは通信設定で変更可能ですが、LAN2ポートのIPアドレスは変更できません。

Windowsのバージョンによって PCのIPアドレスの変更方法がわずかに異なります。この記事では Windows 10を例に説明します。

1. タスクバーで「ネットワーク」を検索し、「ネットワーク接続を表示」を選択します。
2. 現在のネットワーク接続（例えばイーサネット）のアイコンを右クリックしプロパティを選択し、次に、ポップアップウィンドウで**Internet**プロトコルバージョン**4**（**TCP/IPv4**）をダブルクリックします。
3. **Internet**プロトコルバージョン**4**（**TCP/IPv4**）プロパティ画面で次の**IP**アドレスを使用し、PCのIPアドレス、サブネットマスク、デフォルトゲートウェイを変更します。PCのIPアドレスを、コントローラと同じネットワークセグメントで未使用の任意のIPアドレスに変更することができます。そのサブネットマスクとデフォルトゲートウェイはコントローラと同じです。例えば、コンピュータのIPアドレスを192.168.5.100、サブネットマスクを255.255.255.0に設定します。

4. 接続したいロボットの右側の 

ボタンをクリックしてロボットに接続します。

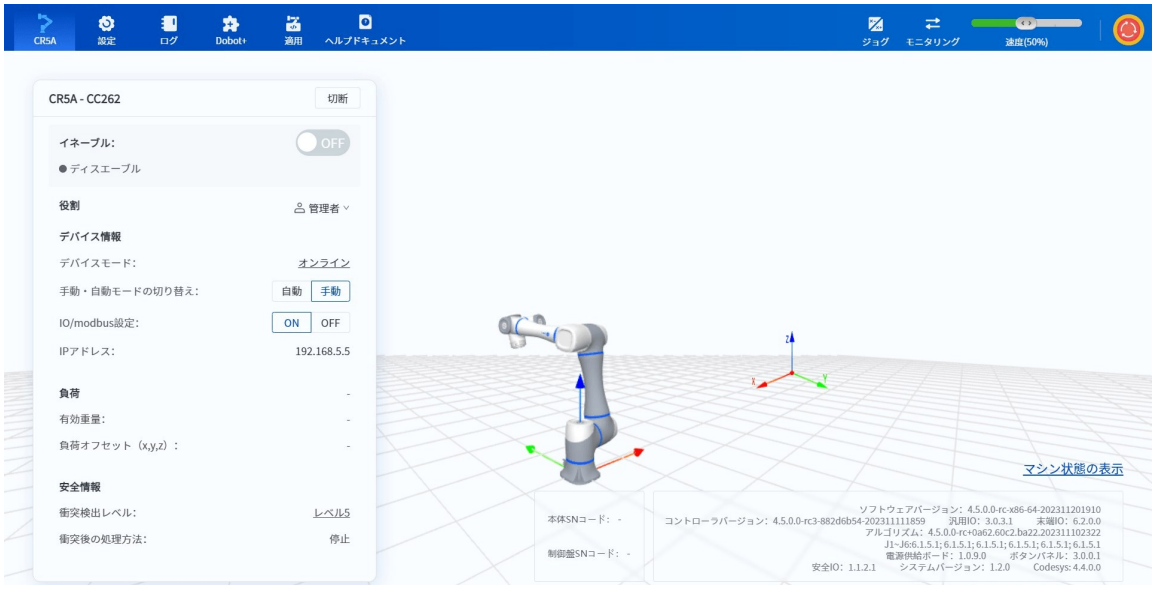


説明：

ソフトウェアがロボットに接続する際、ソフトウェアのバージョンとコントローラのバージョンが一致するかどうかをチェックします。

- バージョン番号の最初の3つが一致していれば、ロボットに正常に接続します。
- バージョン番号の最初の2つが一致し、3つ目が一致しない場合、ソフトウェアは機能が互換性がない可能性があるというポップアップ警告を表示し、その後ロボットに接続します。
- バージョン番号の最初の2つが一致しない場合、ソフトウェアはバージョンが一致しないというポップアップ警告を表示し、自動的に接続を切断します。

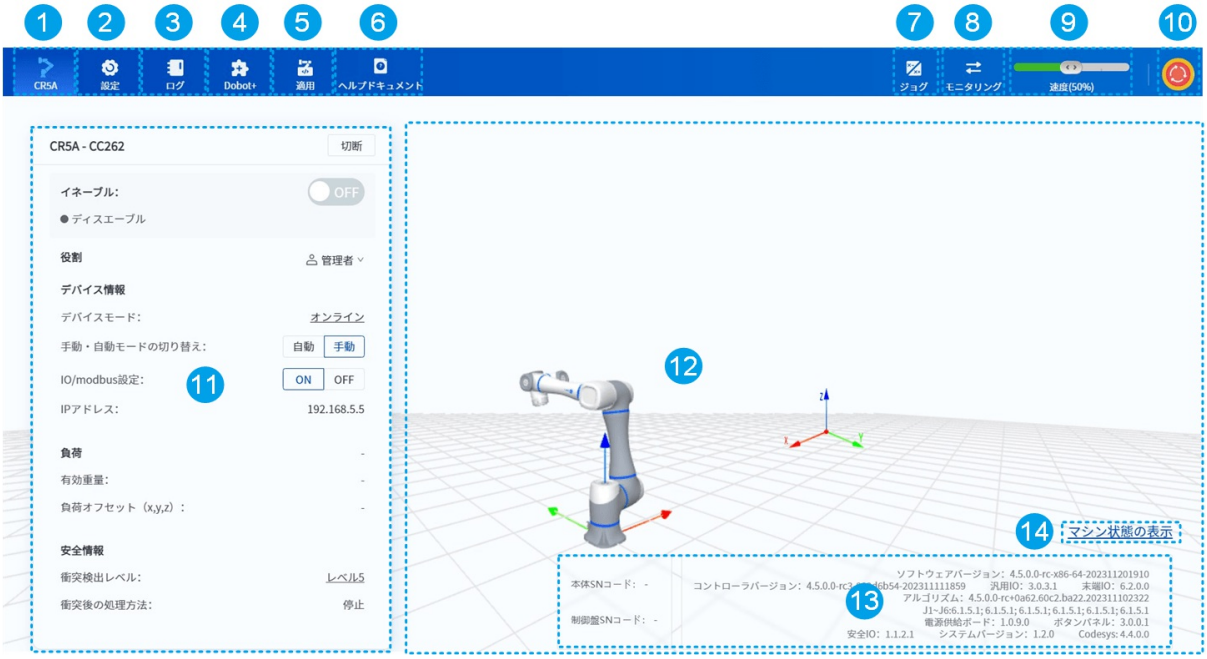
5. ロボットの接続に成功すると、ソフトウェアの画面が更新され、接続されたロボットの関連情報と3Dモデルが表示されます。ロボットの名前の右側の切断ボタンをクリックすると、現在のロボットとの接続が切断されます。



3 インターフェース概要

3.1 メインインターフェース

ソフトウェアがロボットに接続されると、以下のようなメイン画面が表示されます。

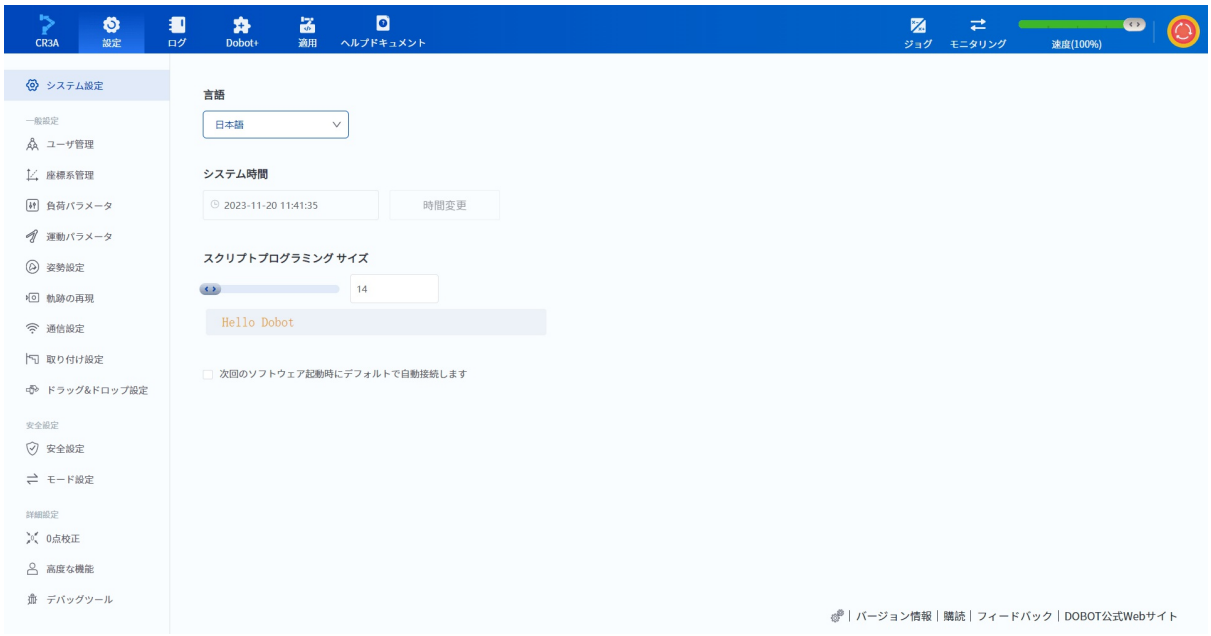


番号	説明
1	クリックするとメイン画面（現在の画面）が開きます。ロボットのアイコンの色は、ロボットのインジケータライトと同期し、ロボットの状態に応じて変化します。指示灯の色の定義については、ロボットのハードウェア取扱説明書をご参照ください。
2	クリックすると設定ページが開き、ロボットの設置、運動、安全などの関連パラメータを設定することができます。
3	クリックするとログページが開き、アラームとログを確認することができます。
4	クリックするとDobot+ページが開き、Dobot+エコシステムプラグインをインストールして使用することができます。
5	クリックするとアプリケーションページが開き、ブロックまたはスクリプトの方式でプログラミングを行うことができます。
6	クリックするとヘルプドキュメントが開きます。ソフトウェア画面での表示、スタンドアロンウィンドウの表示、ブラウザでの表示がサポートされています。
7	クリックするとジョグパネルが開き、ロボットをジョグまたはインチング動作させることができます。
8	クリックするとモニタリングパネルが開き、ロボットのIOとグローバル変数などを監視することができます。
9	グローバルの速度を表示し設定するためのもので、ロボットの動きの速度を制御します。

10	急停止ボタンは、ロボットが運行中に突発的な状況が発生した場合にこのボタンを押すと、ロボットは非常停止します。
11	このパネルは、ロボットのキーデータとモードの切り替えを表示するために使用されます。
12	ロボットの3Dモデルです。姿勢は実際のロボットと一致します。
13	ロボットのSNコード、ソフトウェアおよびファームウェアのバージョン情報。
14	クリックすると、ロボットの現在の電圧、電流、温度などの状態を確認できます。 

3.2 設定ページ

設定ページでは、ソフトウェアやロボットに関連する設定（例：表示言語、ユーザ/ツール座標系、セキュリティ設定など）をユーザが変更することができます。各設定ページの使用説明を参照してください。



3.3 ログページ

ログページには現在のアラームおよびログ記録2つのサブページがあります。

- 現在のアラームページはアラームの表示とクリアを行うためのもので、詳細はアラーム処理を参照してください。
- ログ記録ページはログの表示とエクスポートを行うためのもので、詳細はログを参照してください。

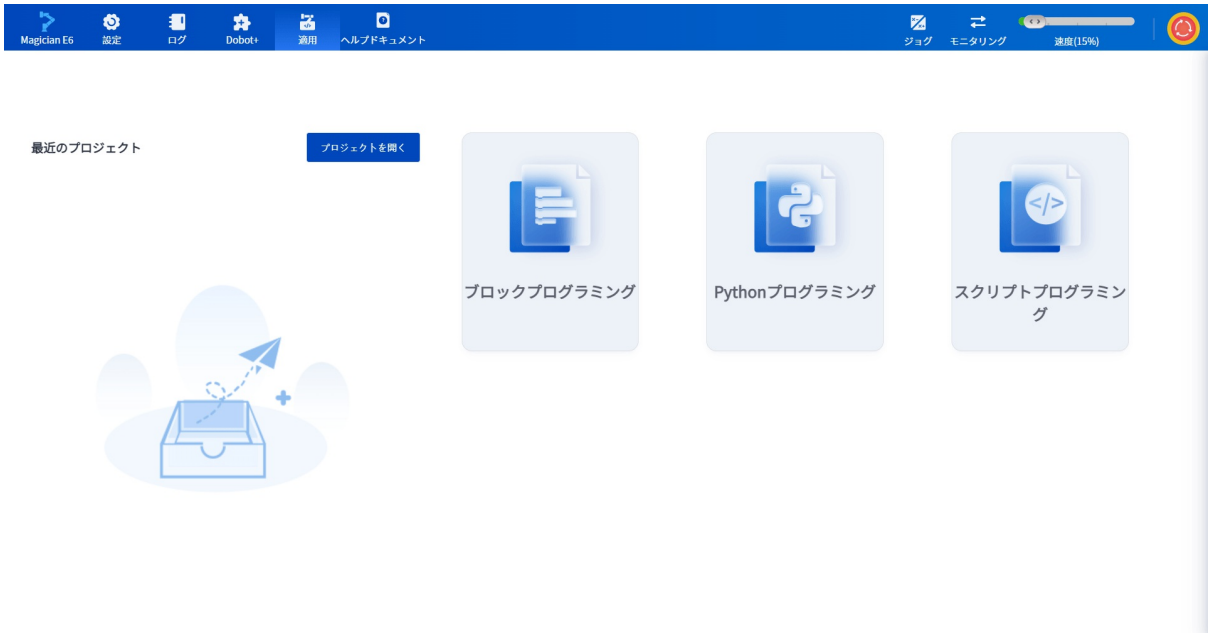
3.4 Dobot+ページ

Dobot+ページはDobot+プラグインの管理と使用を行うためのものです。詳細はDobot+機能説明をご参照ください。

Dobot+プラグインは、エコシステムアクセサリーに対応する専用プラグインで、ユーザは二次開発を行うことなく、プラグインでクローなどのエコシステムアクセサリーを直接設定・使用することができます。

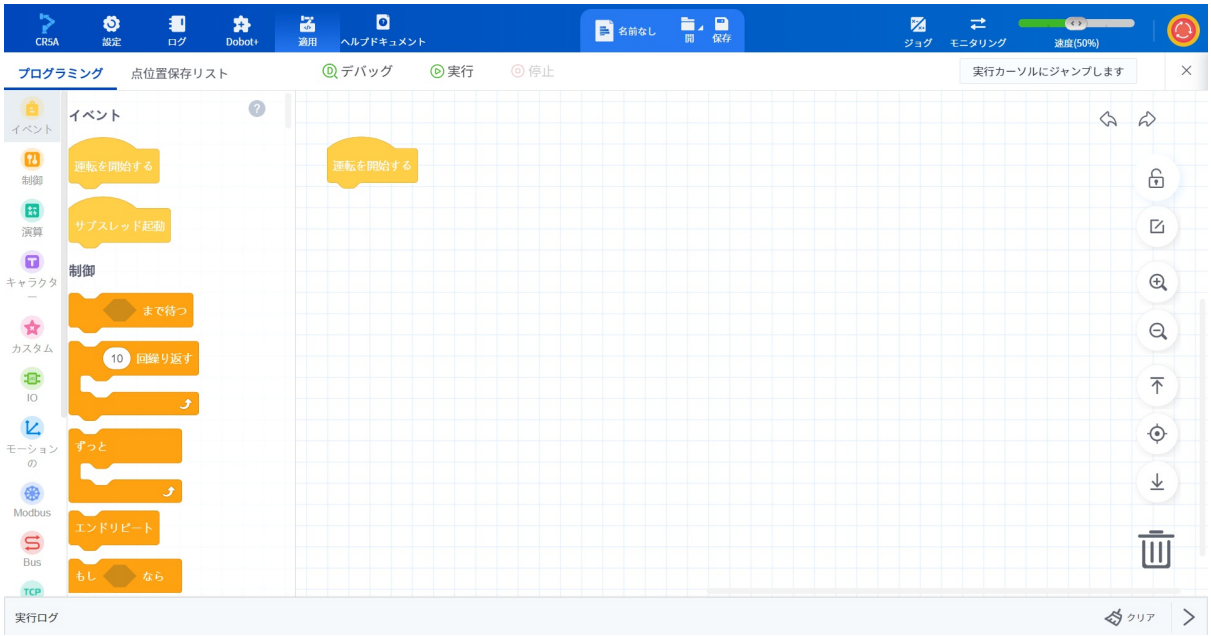
3.5 アプリケーションページ

アプリケーション画面では、ユーザが自分のプロジェクトを作成し、ロボットを自動実行させることができます。



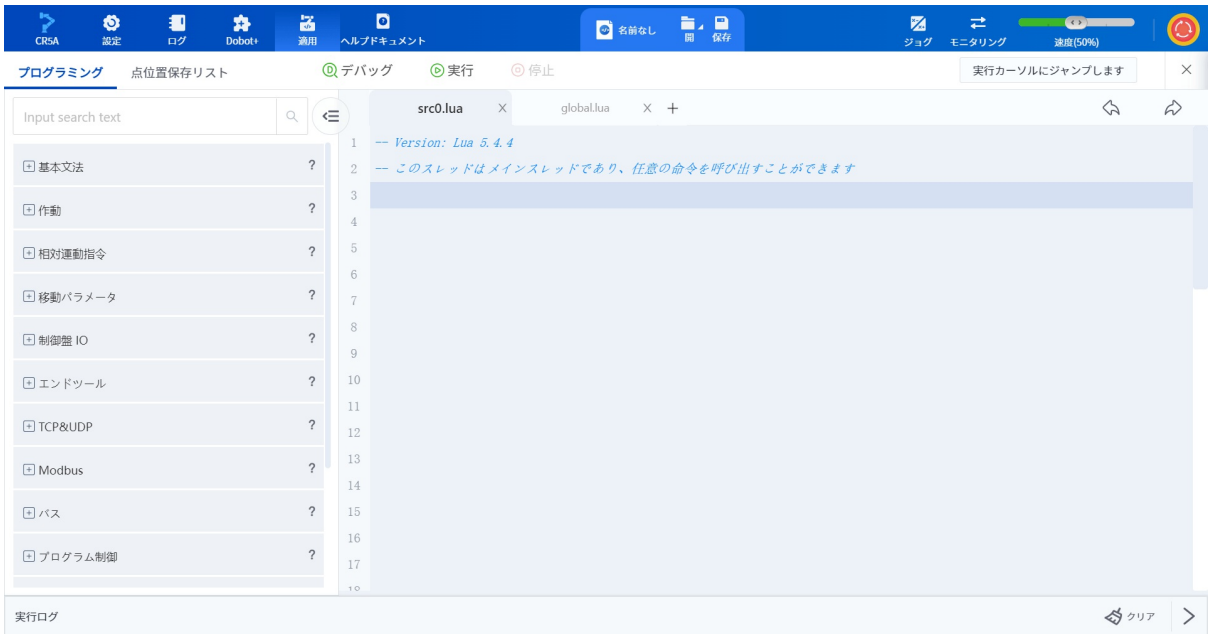
ブロックプログラミング

ブロックプログラミングはグラフィカルなプログラミング方式で、ユーザはプログラミングページの左側にあるブロックを右側のキャンバスにドラッグしてプログラミングすることができます。詳細はブロックプログラミングをご参照ください。



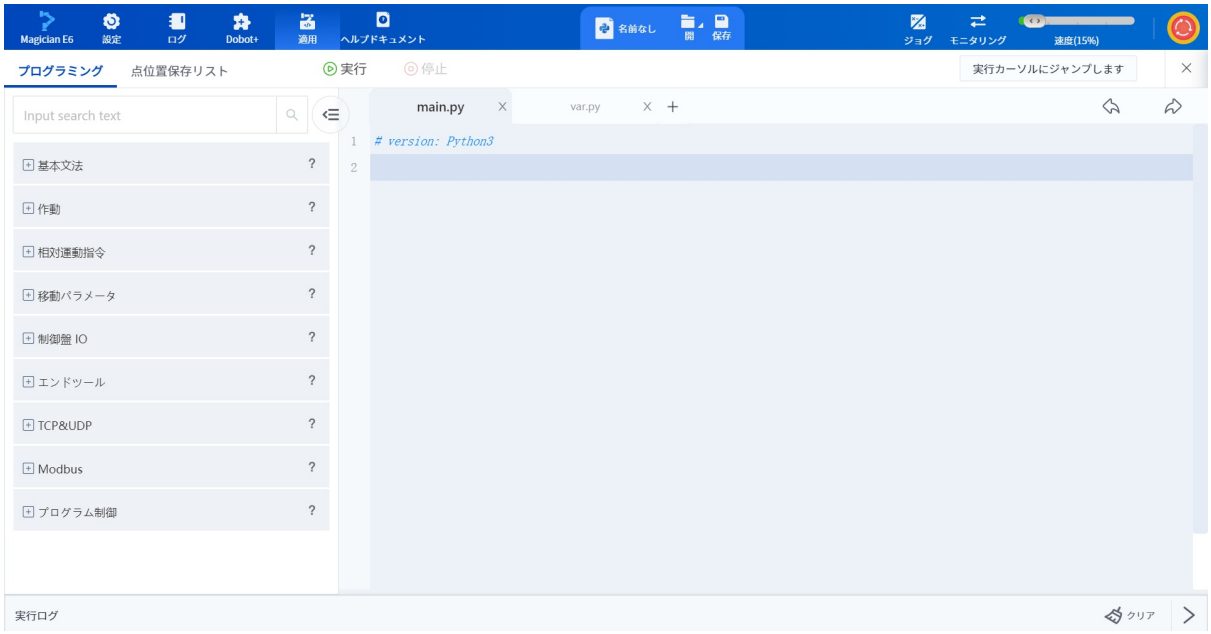
スクリプトプログラミング

スクリプトプログラミングはluaプログラミング言語に基づいたプログラミング方式で、ユーザはプログラミングページの左側で使用する命令を探し、パラメータを設定して右側のプログラミングエリアに追加したり、右側のプログラミングエリアで直接プログラムを書くことができます。詳細はスクリプトプログラミングをご参照ください。



Pythonプログラミング

Pythonプログラミングは、Pythonプログラミング言語に基づいたプログラミング手法です。**PC**上の**Magician E6**ロボットに接続されている場合にのみ利用可能、教育と研究に使用されます。オペレーティングインターフェースは基本的にスクリプトプログラミングと同じであり、デバッグには対応していません。詳細はPythonプログラミングを参照してください。



3.6 ジョグパネル

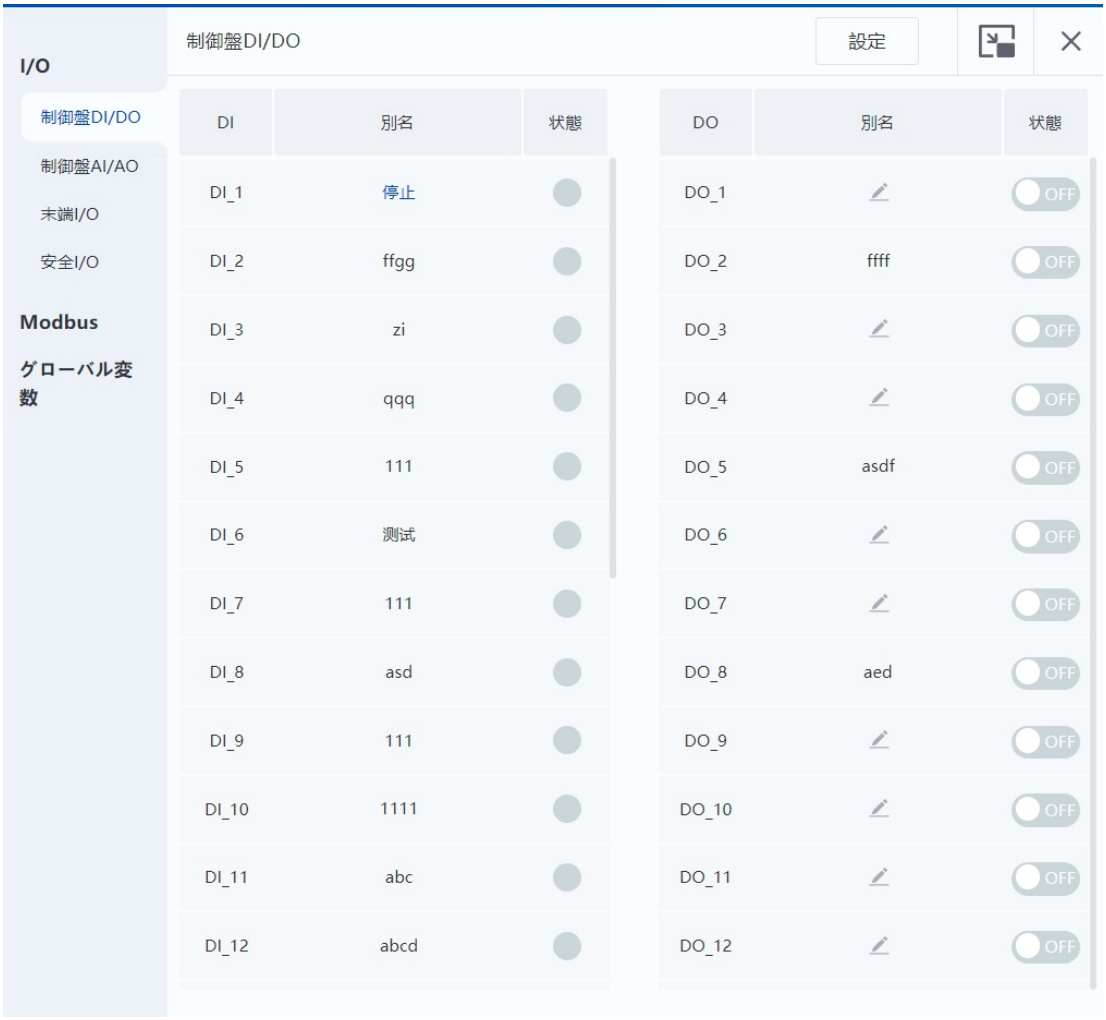
ジョグパネルは、ロボットをジョグまたはインチング動作させるためのもので、関節ジョグと座標系ジョグの2つの方式をサポートしています。詳細はジョグ機能説明を参照してください。

パネルの右上隅の はパネルを自由にドラッグできる独立したウィンドウにすることができ、独立したウィンドウを閉じて監視ページを再度開くと、監視ページは埋め込みパネルに戻ります。

3.7 監視パネル

監視パネルでは、ロボットの各IOの状態と機能を監視および設定することができ、また、グローバル変数の管理と表示も可能です。各監視ページの使用説明を参照してください。

パネルの右上隅の はパネルを自由にドラッグできる独立したウィンドウにすることができ、独立したウィンドウを閉じて監視ページを再度開くと、監視ページは埋め込みパネルに戻ります。



4 クイックスタート

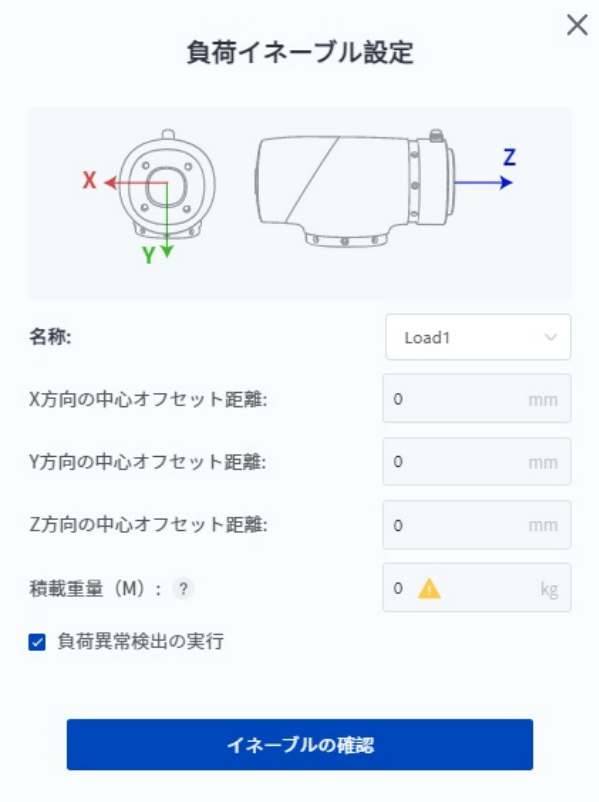
この章では、ロボットが2点間で循環運動するブロックプロジェクトを作成する方法を紹介し、ユーザがDobotロボットの機能を迅速に体験するのを助けます。

この章では、目標を達成するための最も簡単な操作手順のみを紹介します。各機能の詳細な説明は、本マニュアルの後続の章を参照してください。

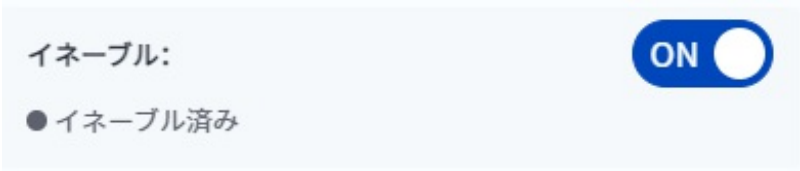
ロボットを有効にする

1. ロボット接続完了後、メインインターフェースの情報パネル上のイネーブルスイッチをクリックします。

2. ソフトウェアが負荷イネーブル設定ウィンドウを表示します。ロボットのエンドにツールが装着されていない場合は、イネーブルの確認ボタンを直接クリックします。ロボットのエンドにツールが装着されている場合は、有効にするページを参照して負荷パラメータを設定してください。

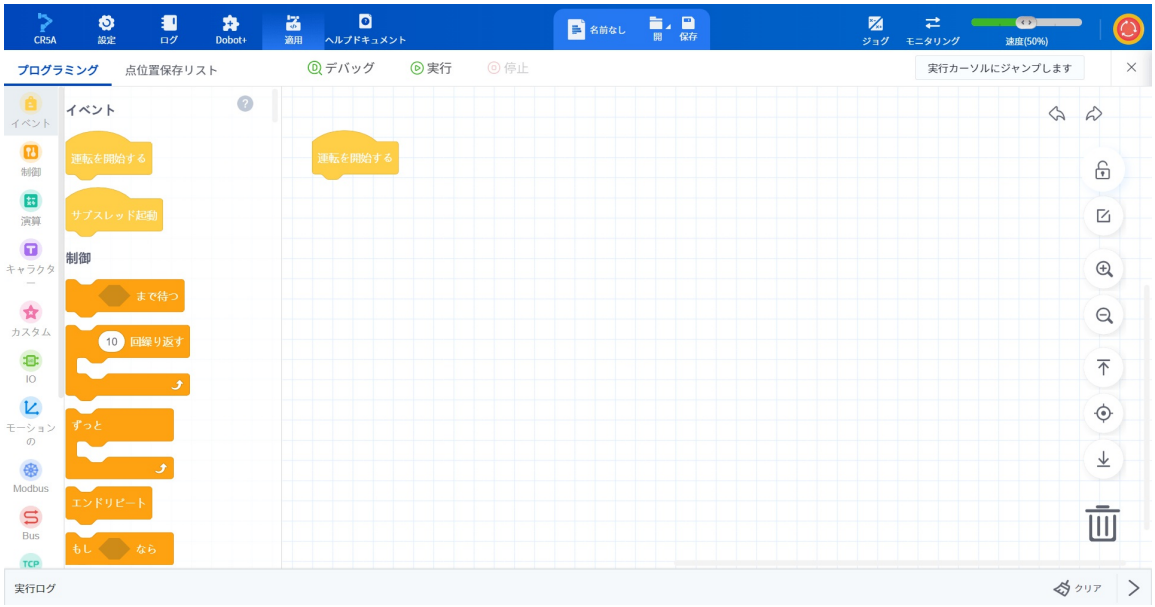


3. 有効化に成功すると、有効化ボタンは**ON**。

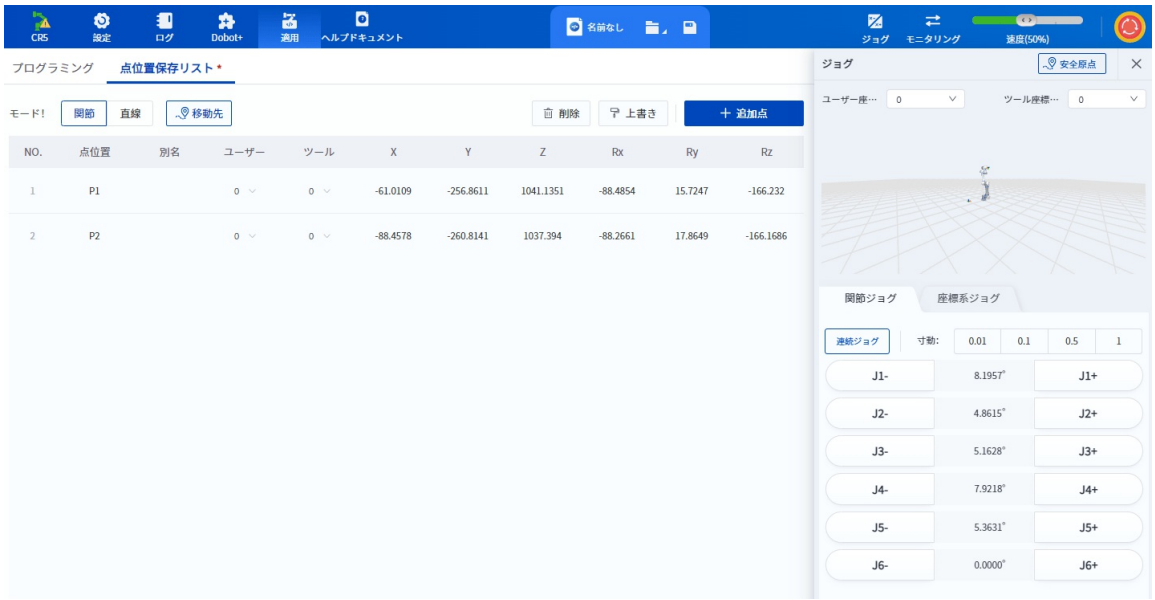


ブロックプロジェクトを作成する

1. 開くアプリケーション > ブロックプログラミングインターフェース。



2. 開く点位置保存リスト。



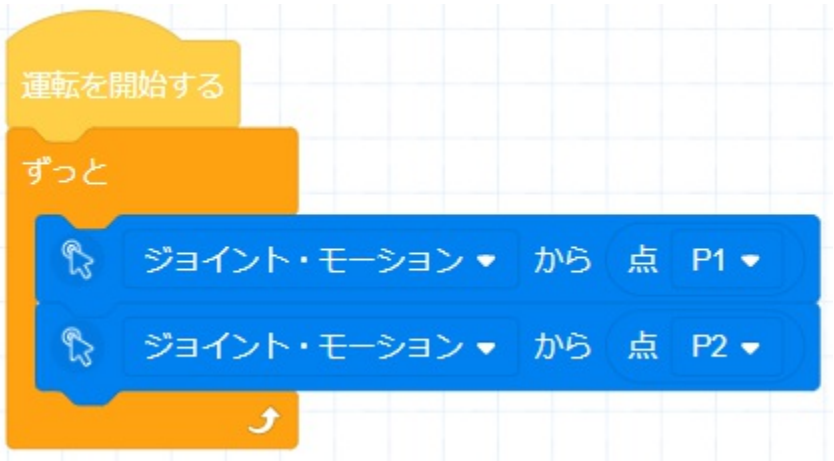
3. 押し続けますジョグパネル上の制御ボタン（例えば**J1+**、**J1-**）を使用して、ロボットを望む姿勢に移動させた後、点位置保存リスト右上隅の＋点を追加ボタンをクリックして、P1点を保存します。

4. 同様に、P2点を保存します。

5. に戻りますプロジェクトのプログラミングページ、左側のブロックエリアから制御ブロック群の繰り返し実行ブロックをキャンバス実行開始ブロックの下部にドラッグします。

6. 運動ブロックグループの点まで移動ブロックを繰り返し実行ブロックの中にドラッグし、ポイントのドロップダウンリストをクリックしてP1を選択します。

7. もう一つの点まで移動ブロックを前の点まで移動ブロックの下にドラッグし、ポイントのドロップダウンリストをクリックしてP2を選択します。最終的な効果は下図のようになります。



保存して実行

⚠注意：
ロボットを運転開始する前に、ロボットの作業範囲内に人や他の障害物がないことを確認してください。

プログラミングページの上部にある  をクリックし、プロジェクト名（例えば“test”）を入力して現在のプロジェクトを保存し、次に  実行をクリックします。すると、ロボットはまずP1点に移動し、その後P1とP2の間で循環運動を始めます。

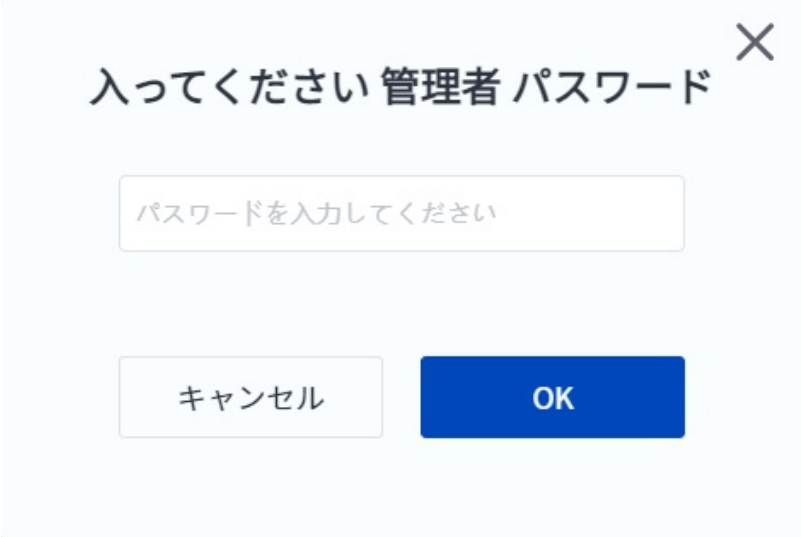
5 ロボットの基本操作

5.1 ユーザーログイン

DobotStudio Proは、デフォルトで管理者、技術者、操作者の3つの役割をサポートします。各役割の権限は変更できます。デフォルトの権限は次のように分類されています。管理者 > 技術者 > 操作者。詳細はデフォルトの役割の権限を参照してください。現在ログインしている機能に権限がない機能に対応するUIはグレーアウトされ、操作できません。

ユーザーは、ソフトウェアのメイン画面で現在ログインしている役割を切り換えることができます。ロボットの工場出荷時のデフォルトのログイン役割は技術者です。

ログインしたい役割にパスワードがない場合は直接切り換えることができますが、パスワードがある場合は切り換えるためにはパスワードを入力する必要があります。デフォルトの管理者パスワードは**888888**です。他の役割にはデフォルトでパスワードがありません。



ユーザーが管理者としてログインすると、カスタム役割の追加、各役割の権限とパスワードの表示/変更が可能になります。詳細はユーザー管理を参照してください。

5.2 イネーブル

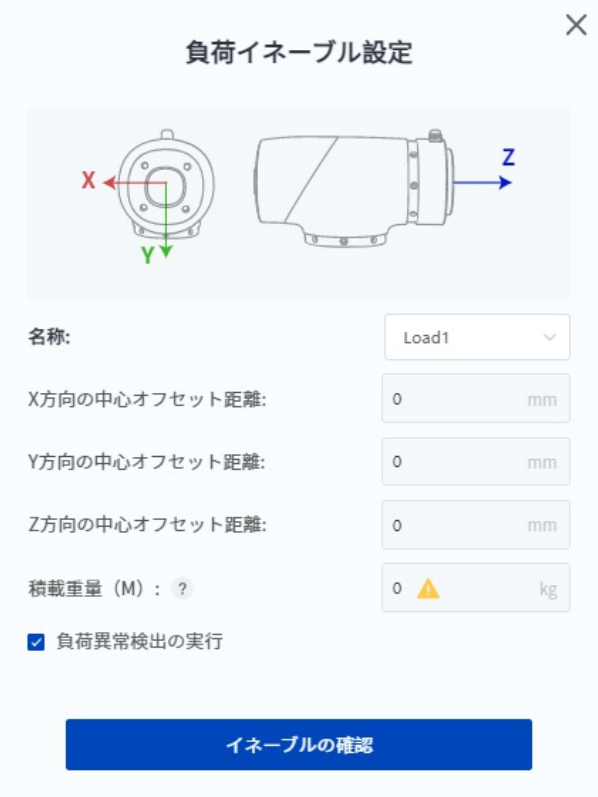
ロボットは電源を入れた後、デフォルトでは無効状態になります。この時、ロボットの設定やプログラミングは可能ですが、ロボットの動作やプロジェクトの実行はできません。ロボットの動作やプロジェクトの実行を制御するには、ロボットをイネーブルする必要があります。

ユーザはソフトウェアのメイン画面のイネーブルスイッチまたはロボット上のイネーブルボタンを使用してイネーブル状態を切り替えることができます。

ソフトウェアによるイネーブル

ソフトウェアのイネーブルスイッチはメイン画面の情報パネルにあります。

[イネーブル]ボタンが**OFF**の場合、クリックすると、[負荷設定]ウィンドウがポップアップ表示されます。



ドロップダウンボックスを使用して、負荷パラメータのセットを選択します。プリセットされるパラメータセットは、[負荷パラメータ]ページで設定した後、ここで選択する必要があります。「カスタム」を選択すると、次のパラメータを手動で変更できます。

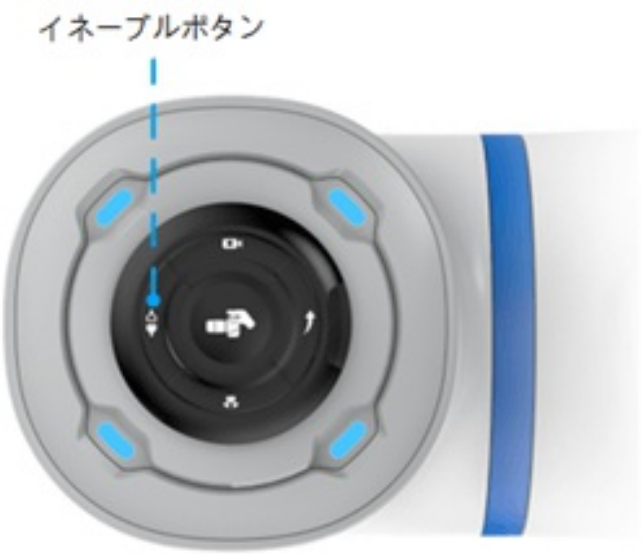
- **X/Y/Z**方向の中心オフセット距離：エンド負荷の重心が各方向にどれだけの距離をオフセットしているか。各座標軸の方向は画面上の図をご参照ください。
- 負荷重量：エンド治具の重量と工作物の重量の合計は、ロボットアームの許容最大荷重を超えないようにしてください。
- 負荷異常検出の実行：このオプションをチェックすると、ロボットはユーザが設定した負荷パラメータと実際の負荷との差が大きすぎるかどうかを検出するために小幅に動作します。差が大きすぎる場合、有効化に失敗し、負荷パラメータを再設定するか、このオプションのチェックを外す必要があります。

有効化したら、有効化ボタンは**ON**に変わり、もう一度クリックすると無効化します。

ロボットボタンによるイネーブル

- CRAシリーズ（CR20Aを除く）のロボットのイネーブルボタンはロボットの先端にあります。

説明：
CR20Aロボットにはイネーブルボタンがなく、ソフトウェアによるイネーブルのみが可能です。



電源がオンでロボットがイネーブルになっていない場合 (ロボットのインジケーターライトが青で点灯)、イネーブルボタンを長押し (1.5秒以上) してロボットをイネーブルにします (インジケーターライトが緑色で点灯)。イネーブル状態ではイネーブルボタンを短く押すと (1.5秒未満) ロボットをディセーブルにします。

- Magician E6ロボットのイネーブルボタンはロボットのベースにあります。

電源がオンでロボットがイネーブルになっていない場合 (ロボットのインジケーターライトが青で点灯)、イネーブルボタンを長押し (1.5秒以上) してロボットをイネーブルにします (インジケーターライトが緑色で点灯)。イネーブル状態ではイネーブルボタンを短く押すと (1.5秒未満) ロボットをディセーブルにします。

ロボットのボタンでイネーブルにした後、ロボットは最後にイネーブルにしたときの負荷設定を引き続き使用します。新しいロボットが初めてイネーブルにされる場合、負荷パラメータの値はすべて0です。

5.3 モード切り換え

5.3.1 デバイスモード

デバイスモードは、ロボットの現在の制御モードを示します：

- オンラインモードはデフォルトの制御モードです。このモードではDobotStudio Proを使用してロボットを制御することができます。また、I/OまたはModbusを使用してロボットをリモート制御することも可能です。
- **TCP**モデルはお客様がTCPに基づいて制御ソフトウェアを独自に開発する場合にのみ使用されます。制御ソフトウェアを独自に開発する場合は、「TCP_IPリモートコントロールインターフェース（V4）」の入手方法についてテクニカルサポートにご連絡ください。

ユーザはメイン画面の情報パネルで現在のデバイスモードを確認できます。下線付きのテキストをクリックするとモードを切り替えることができます。

> 説明：
> デバイスモードを切り替える時にロボットが動作中の場合、切り替え後にすぐに停止しますが、有効化状態は変わりません。

5.3.2 手動/自動モード

操作モードにより、ロボットの最大グローバル速度とリモートIO/Modbus/安全IOトリガー機能の有効範囲が異なります。現場における実際の必要性に基づいて操作モードを選択してください。

- デフォルトモード：ロボットは工場出荷時にこのモードがデフォルトに設定されており、このモードではグローバル速度とリモートIO/Modbus/安全IOトリガー機能は制限されません。ユーザがモード設定で操作モードの切り替え機能を有効にした場合、操作モードは手動/自動モードの二つから選択可能となります。その機能を無効にすると操作モードの切り替えデフォルトモードに戻ります。
- 手動モード：手動モードでは、ロボットが点動以外の動作を行う際のグローバル速度は50%を超えることはできません。リモートIO/Modbus/安全IOトリガー機能の有効範囲については下表を参照してください。
- 自動モード：自動モードでは、グローバル速度は制限されません。リモートIO/Modbus/安全IOトリガー機能の有効範囲については下表を参照してください。

リモート**IO/Modbus**制御

機能	手動モード	自動モード
開始	X	√
停止	X	√
一時停止	X	√
イネーブル	√	√
ディセーブル	√	√
アラームをクリア	√	√
ドラッグに入る	√	X
ドラッグを終了	√	X
代替プロジェクトの選択	X	√

安全**I/O**

機能	手動モード	自動モード
ユーザによる非常停止	√	√
防護停止	X	√
防護停止リセット	X	√
低減モード	X	√

モード設定で操作モードの切り替え機能を有効にすると、メイン画面の情報パネルに手動/自動モード切り替えスイッチが表示されます。デフォルトは手動モードです。

ロボットが以下の状態にない場合、スイッチをクリックして自動モードです。モード設定でパスワードを設定した場合、切り替える前にパスワードを入力する必要があります。

- プロジェクトデバッグ状態
- 軌跡再現状態
- ドラッグ状態
- 関節ブレーキリリース状態

自動モードでは、いつでもスイッチをクリックして手動モードに切り替えることができます。切り替え時にロボットが動作中であれば、切り替え後にすぐに停止しますが、有効状態は変わりません。

5.3.3 IO/Modbus設定スイッチ

IO/Modbus設定スイッチは、リモートIO/Modbusトリガ機能が有効になるかどうかを制御するためのものであり、具体的な有効範囲は手動/自動モードによっても影響を受けます。詳細は下表を参照してください。

操作モード	**IO/Modbus**を**ON**に設定	**IO/Modbus**を**OFF**に設定
デフォルトモード	すべてのIO/Modbus入力が有効	すべてのIO/Modbus入力が無効
自動モード	自動モードで使用可能なIO/Modbus入力が有効	すべてのIO/Modbus入力が無効
手動モード	手動モードで使用可能なIO/Modbus入力が有効	すべてのIO/Modbus入力が無効

メインインターフェースの情報パネル上の**IO/Modbus**設定スイッチをクリックしてON/OFFを切り替えることができます。

5.3.4 縮小モード

ロボットが縮小モードをトリガーすると、グローバル速度は10%に制限され、変更することはできません。これは、ロボットの高速動作による安全リスクを避けるためです。

ロボットは以下の方法で縮小モードをトリガーできます：

- ロボットは、縮小タイプのセキュリティウォールまたは安全領域をトリガーした。
- ユーザーが安全IOの縮小モード入力をトリガーした。

5.4 ジョグ

ユーザーは、ジョグパネルを使用して、ロボットを指定した関節軸またはデカルト座標軸に沿ってジョグまたはインチングさせることができます。

番号	説明
1	長押しすると、ロボットアームが安全原点に移動します。
2	ロボットの現在有効な座標系を表示します。ドロップダウンボックスをクリックして変更します。
3	ロボットシミュレーションパネルでは、ロボットの姿勢をリアルタイムに表示します。このパネルはモバイル版では利用できません。
4	ジョグの方法を選択し、関節のジョグと座標系のジョグをサポートします。
5	ジョグボタンの機能を選択します。 継続的なジョグ：ジョグボタンを押したままにするとロボットアームが動き続け、ボタンを離すとジョグが停止します。 インチング：ジョグボタンを押すたびに、ロボットアームが指定した距離だけ移動します。関節ジョグ時のジョグ距離の単位は°。座標系がジョグする場合、XYZ軸のインチング距離の単位はmm、RXRYRZ単位は°です。

6	関節ジョグボタン。**J1+**、**J1-**を例として、**J1+**、**J1-**ボタンをクリックして、ロボットアームがJ1軸に沿って正または負の方向に回転するように制御します。以降も同様です。
7	ロボットアームがユーザー座標系に沿ってジョグするか、工具座標系に沿ってジョグするかを選択します。ユーザー座標系を選択すると、ロボットアームの先端がユーザー座標系の対応する軸に沿ってジョグします。工具座標系を選択すると同じようになります。座標系に沿ってジョグすると、3Dモデルには現在の参照座標系が表示されます。
8	座標系のジョグボタン。**X+**、**X-**を例として、**X+**、**X-**ボタンをクリックして、ロボットアームが指定された座標系のX軸に沿って正または負の方向に移動するように制御します。以降も同様です。

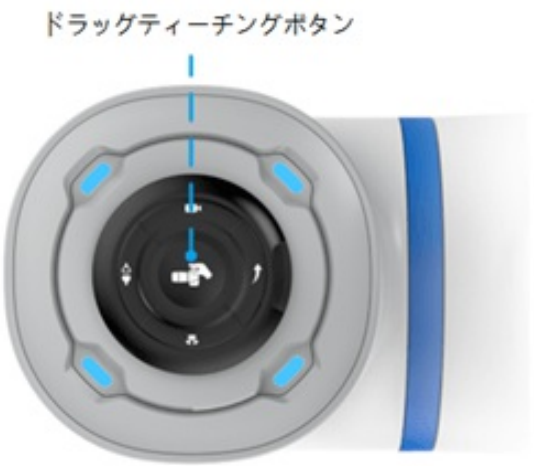
ジョグパネル以外にも、位置設定に関連するページでは  移動先ボタンが表示され、そのボタンを押したままにすると、対応する位置に移動し、その位置に到達すると、バブルプロンプトがポップアップ表示されます。移動中にこのボタンを放すとロボットアームの移動が停止します。

5.5 ドラッグ

ロボットは先端のドラッグ教示ボタンを通じてドラッグモードに入ります。ドラッグモードでは、ユーザーは手でロボットアームをドラッグして姿勢を変え、位置の教示を行うことができます。

ドラッグモードでの各関節の抵抗力は、ドラッグ設定ページで変更することができます

- CRAシリーズ（CR20Aを除く）のロボットのドラッグ教示ボタンの位置は下図の通りです。



  ロボットアームがイネーブルになっている場合 (インジケーターライトが緑色に点灯)、ドラッグ教示ボタンを押し続けると (1.5秒以上)、ロボットはドラッグモードに入ります (インジケーターライトが緑色で素早く点滅)。ユーザーはロボットを手でドラッグして姿勢を変えることができます。

  ドラッグモードでは、教示ボタンを短押し (1.5秒未満) ドラッグしてドラッグモードを終了します。

- CR20Aロボットのドラッグ教示ボタンは先端の側面にあり、シルク印刷は**FREE**です。

  ロボットアームがイネーブルになっている場合（インジケーターライトが緑色に点灯）、このボタンを押し続けるとドラッグモード（インジケーターライトが緑色で素早く点滅）に入ります。ユーザーはロボットを手でドラッグして姿勢を変えることができます。

  ドラッグ教示ボタンを離すと、ドラッグモードを終了します。

- Magician E6ロボットのドラッグ教示ボタンの位置は下図の通りです。

ロボットアームがイネーブルになっている場合 (インジケーターライトが緑色に点灯)、ドラッグ教示ボタンを押し続けると (1.5秒以上)、ロボットはドラッグモードに入ります (インジケーターライトが緑色で素早く点滅)。ユーザーはロボットを手でドラッグして姿勢を変えることができます。ドラッグモードでもう一度長押しすると軌跡記録モードに入ります。

ドラッグモードでは、教示ボタンを短押し (1.5秒未満) ドラッグして、ドラッグモードと軌跡記録モードを終了します。

5.6 座標系の切り替え

ロボットがデカルト座標系に基づいて動作する場合、参照座標系としてユーザ座標系とツール座標系を指定する必要があります。

- ユーザ座標系：ユーザ座標系は、ユーザが自定義する作業台座標系またはワークピース座標系で、その原点と各軸の方向は実際の需要に応じて決定できます。そのため、作業範囲内の各点の位置を簡単に測定し、タスクを配置することができます。ユーザ座標系はユーザ座標系ページで管理します。
- ツール座標系：ツール座標系は、ツールの中心点TCP（Tool Center Point）の位置とツールの姿勢を定義する座標系で、その原点と方向はエンドエフェクタの位置と角度に応じて変化します。ツール座標系はツール座標系ページで管理します。

ロボットが異なるシーンで動作するとき、参照座標系の選択方法は異なります。

- ロボットのジョグ参照座標系は、ジョグパネルで選択します。ジョグの説明をご参照ください。
- ロボットのが指定点へ移動するときの参照座標系は、目標位置の示教時の参照座標系です。
- ロボットがプロジェクトを実行するとき、運動命令の参照座標系はパラメータで指定できます。詳細はスクリプト命令一般説明をご参照ください。

5.7 非常停止と復帰

ロボットの運転中に突発的な状況が発生した場合、非常停止機能を使用してロボットを非常停止することができます。

ユーザは次の方法で非常停止機能を起動できます：

- ロボットコントローラーに接続された非常停止ボタンを押します。
- 安全なIOのユーザ非常停止入力を起動します。
- ソフトウェアインターフェースの右上隅にある非常停止ボタンをクリックします。

非常停止機能が起動した後、ロボットは停止を計画します。

- 500ms以内に停止する場合、ロボットは電源を切らずに無効になります；
- 停止の計画が500ms後にロボットがまだ動いている場合、ロボットは強制的に無効になり、電源が切られます。

これらの二つの状況はそれぞれ対応する警告を生成します。

非常停止後、非常停止ボタンのアイコンは点滅状態になります。ロボットを再度有効にするには、まず非常停止ボタンを再度クリックしてリセットし、次に警告をクリアし、その後に有効にします（ロボットが電源を切っていた場合、まずポップアップの指示に従って電源を入れてください）。

⚠ 注意：

- ソフトウェアインターフェース上の非常停止ボタンは、ハードウェアの非常停止機能を補完するためのものであり、緊急時にはロボットを停止するためにまずハードウェアの非常停止機能を使用するべきです。
- 物理的な非常停止ボタンや安全なIOによる非常停止は、このボタンの状態を変更しますが、この時点ではソフトウェアインターフェースのリセットボタンを使用することはできず、対応する入力ソースでリセットする必要があります。

5.8 速度調整

ロボットの速度はジョギング速度および再現速度，ジョグにはジョギングパネルによって作動するジョギング/インチング、TCPジョギング命令、およびへ移動する機能を含み、再現にはプロジェクトの実行とTCP動作命令を含みます。

ジョグ速度の計算方法：

- ロボットの実際の最大ジョギング速度 = 動きパラメータ中の最大ジョギング速度 x グローバルレート

  例：関節速度は12°/sとし、グローバル速度は50%とすると、実際のジョギング速度は12°/s × 50% = 6°/sとなります。

- ロボットのジョギング加速度はグローバル速度から影響を受けません。

再現速度の計算方法：

- ロボットの実際の最大再現速度 = 動きパラメータ中の最大再現速度 x グローバルレート x プログラミング時命令パラメータが設定した速度比率

  例：デカルト速度が2000mm/s、グローバル速度が50%、プログラミング時の移動命令の速度設定が80%の場合、実際の移動速度は2000mm/s × 50% × 80% = 800mm/sです。

- ロボットの実際の最大再現加速度 = 動きパラメータ中の最大再現加速度 × プログラミング時命令パラメータが設定した加速度比率、グローバル速度から影響を受けません。

  例：デカルト加速度が10000mm/s²、プログラミング時の移動命令の加速度設定が50%の場合、実際の動作加速度は10000mm/s² × 50% = 5000mm/s²です。

- ロボットの再現加加速度はグローバル速度と命令パラメータから影響を受けません。

その中で、動きパラメータは動作パラメータ設定ページで閲覧および修正が可能です；

グローバル速度は上部ツールバーの右側で閲覧および修正が可能です；

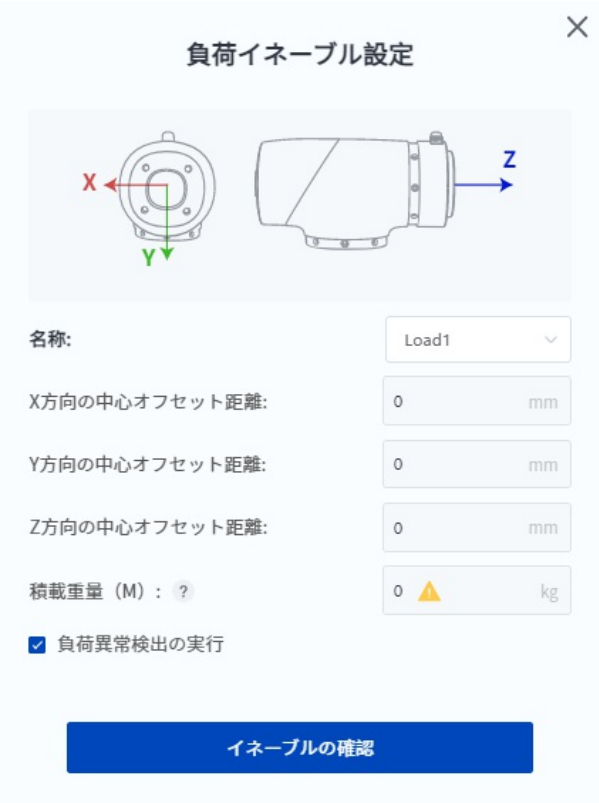
命令パラメータの使用方法はスクリプト命令の全般的な説明を参照してください。

5.9 負荷設定

負荷パラメータは、ロボットのエンド負荷（治具を含む）の重心と重量パラメータであり、実際の負荷に基づいて設定する必要があります。設定が正しくない場合、ロボットの性能が低下し、衝突検出が誤って発生したり、ドラッグが制御不能になったりする可能性があります。

ユーザは以下の方法でロボットの負荷パラメータを設定できます：

- ロボットをソフトウェアで有効にする際、ポップアップウィンドウで負荷パラメータを設定する必要があります。詳細は有効化の操作説明を参照してください。



- プログラミング時には、対応するブロックまたはスクリプト命令を呼び出して、プロジェクトの実行時に負荷パラメータを設定できます。詳細は付録の命令説明書を参照してください。この方法で設定した負荷パラメータは、プロジェクトが実行中のみ有効です。

  ブロック：

  スクリプト：

```
SetPayload(payload, {x, y, z}) -- 負荷パラメータのカスタマイズ
SetPayload(name) -- プリセットの負荷パラメータグループを使用
```

プリセットの負荷パラメータグループは負荷パラメータページで管理します。

5.10 衝突検出設定

ロボットが動作中に衝突を検出すると自動的に停止します。ユーザは衝突検出の感度と衝突後の具体的な処理方法を設定できます。衝突検出をご参照ください。

ユーザは該当するブロックまたはスクリプト命令を使用して、プロジェクトの実行時に衝突検出の感度を設定することもできます。詳細は付録の命令説明書をご参照ください。この方法で設定した衝突検出パラメータは、プロジェクト実行中のみ有効です。

ブロック：

スクリプト：

```
SetCollisionLevel(level)
```

5.11 アラーム処理

ロボットの位置制限や特異点に位置する場合など、インチングやポイント保存の方法が間違っていたり、ロボットが適切に使用されていなかったりすると、アラームがトリガーされます。ロボットの動作中にアラームが発生した場合、ログアイコンの右上に、現在のアラームの数を示す数値付き赤い点が表示されます。

現在のロボットにアラームがある場合、現在のアラームページにはアラームのレベル、エラーコード、説明および解決策が表示されます。説明と解決策を参照してアラームの問題を解決してください。

ロボットのアラームレベルの意味は以下のとおりです。

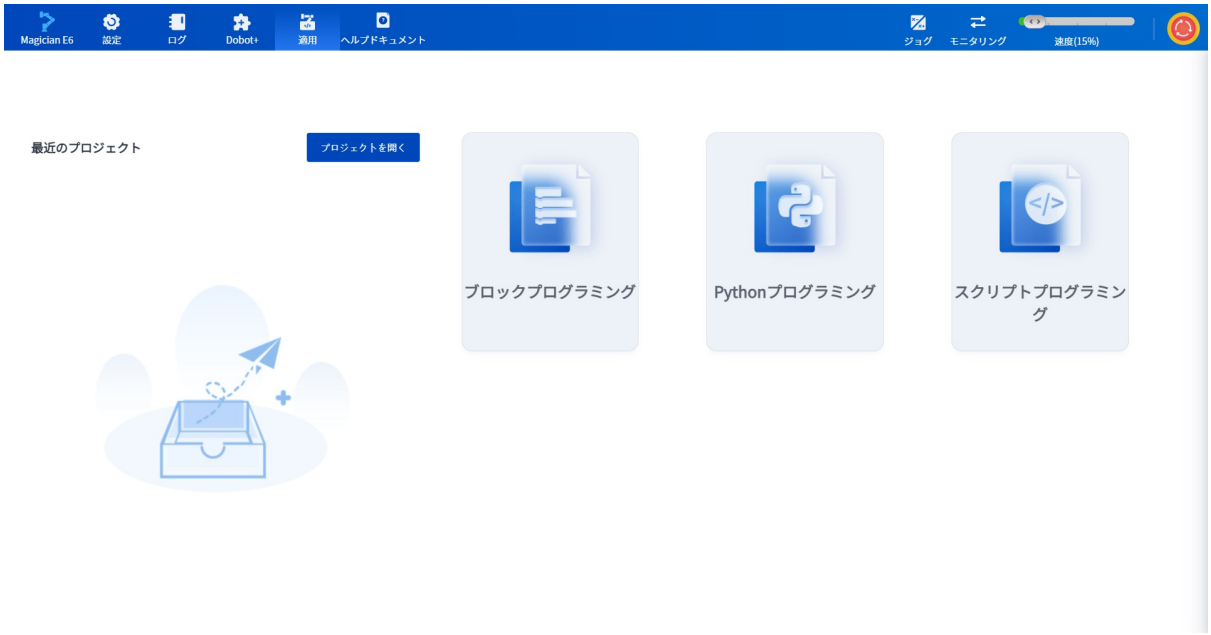
- 0：アラームが発生するとロボットの動作が停止し、有効化が解除され、電源が切られます。
- 1：アラームが発生するとロボットの動作が停止し、有効化が解除されます。
- 5：アラームが発生するとロボットの動作が停止します。

アラートのクリアをクリックすると現在のアラームをクリアできます。一部のアラームは直接クリアできない場合があります。詳細はアラームの解決策をご参照ください。

6 アプリケーション

6.1 アプリケーションタイプの選択

アプリケーション画面を開いた後、ユーザーはまずアプリケーションのタイプを選択する必要があります。



- ブロックプログラミング：グラフィカルなプログラミング方法、シンプルで使いやすいです。
- スクリプトプログラミング：Luaプログラミング言語に基づいたプログラミング方法で、ある程度のプログラミング基礎を備えたユーザーに適しています。
- **Python**プログラミング：Pythonプログラミング言語に基づいたプログラミング手法です。**PC**上の**Magician E6**ロボットに接続されている場合にのみ利用可能、教育と研究に使用されます。

最近のプロジェクトの下には最近開いたプロジェクトが表示されます。プロジェクトを開くボタンをクリックするとプロジェクト選択ボックスがポップアップ表示されます。プロジェクト名の左側にあるアイコンの意味は次のとおりです。

-  ：ブロックプログラミングプロジェクト
-  ：スクリプトプログラミングプロジェクト。
-  ：Pythonプログラミングプロジェクト

6.2 ブロックプログラミング

6.2.1 概要

DobotStudio Proはブロックプログラミングを提供し、ユーザーはブロックの形状に応じてサイドバーからブロックをプログラム領域にドラッグして組み合わせることでプログラミングが可能で、コードを書く必要はありません。

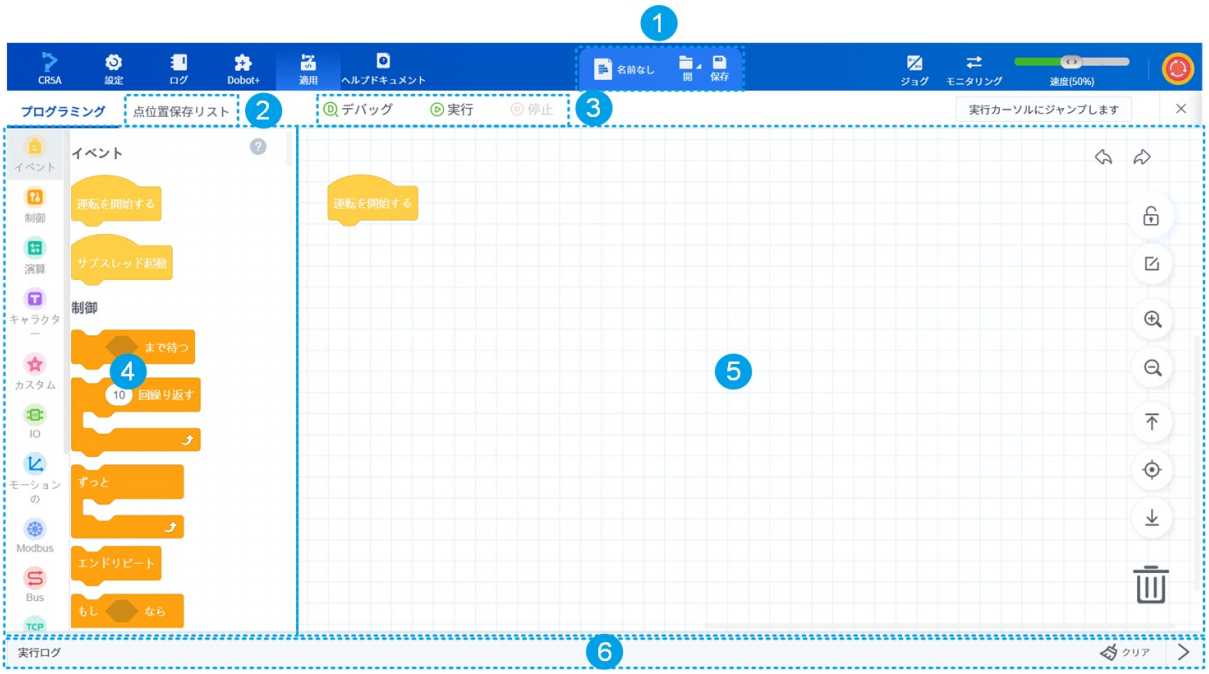
プロジェクト：ブロックプログラミングはプロジェクト単位で編集および実行し、デバッグをサポートします。アプリケーションホームページのアイコンからプログラミングページに進むと、新たに名前のないプロジェクトが自動的に作成されます。プログラミング後のデバッグや実行には名前を付けて保存する必要があります。プロジェクトはロボットアームコントローラーに保存され、インポートおよびエクスポートが可能です。

各プロジェクトでは、1つのメインスレッドとを最大4つのサブスレッドがサポートされています。

- メインスレッド：メインスレッドは、実行開始ブロックを開始ブロックとします。メインスレッドではすべてのブロックを使用できます。
- サブスレッド：サブスレッドは、サブスレッド開始ブロックを開始ブロックとします。サブスレッドは、メインスレッドとともに実行される並列スレッドで、IOや変数などの設定に使用されますが、移動ブロックは使用できません。

保存ポイント：プログラミング中に、ユーザーはいつでもロボットを所望のポーズに移動させて教示点として保存できます。教示点はプロジェクトに紐づけられ、ブロックのパラメータとして使用できます。

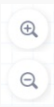
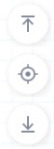
ブロックプログラミングのメイン画面は以下の図のようになります。



番号	説明
1	プロジェクトの名前を表示し、プロジェクトを管理します。
2	クリックすると保存ポイントの画面が開き、プロジェクト内の保存ポイントを管理できます。
3	現在のプロジェクトのデバッグ、実行、一時停止、停止を制御します。
4	プログラミングに必要なブロックを提供し、分類や色別に必要なブロックを探すことができます。右上の ? をクリックすると、ブロックの詳細説明を閲覧できます。
5	ブロックプログラミングのキャンバス、つまりブロックプロジェクトの作成エリアです。 変更が行われて保存されていないブロックの左端には アイコンが表示され、ユーザーにブロックが変更されたことを示します。
6	プロジェクトの実行ログを表示する実行ログバー。最右端のアイコンをクリックすると、ログ表示エリアを展開または折りたたむことができ、 クリアをクリックするとログ表示エリアを空にできます。

キャンバス右側のアイコンの意味は以下の通りです。

アイコン	説明
	プログラミング操作を元に戻す/やり直します。
	プログラム領域のロック/アンロックに使用されます。

	編集モードに入るために使用します。詳細は以下のプログラミング説明をご参照ください。
	プログラム領域をそれぞれ拡大/縮小するために使用されます。
	それぞれ、ブロックを上に戻す/中央揃え表示/ブロックを下に戻す操作に使用します。
	プログラム領域のブロックをここまでドラッグして削除します。ユーザーはブロックを右クリックして削除を選択することもできます。

説明：
ブロックをドラッグする際は、ブロック左側の幅の30%以内の部分を長押ししてください。さもないと、ブロックをドラッグすることはできません。

6.2.2 プロジェクト管理

ブロックプログラミング画面を開くと、新規の空白プロジェクトページがデフォルトで表示され、プロジェクト名が無題です。

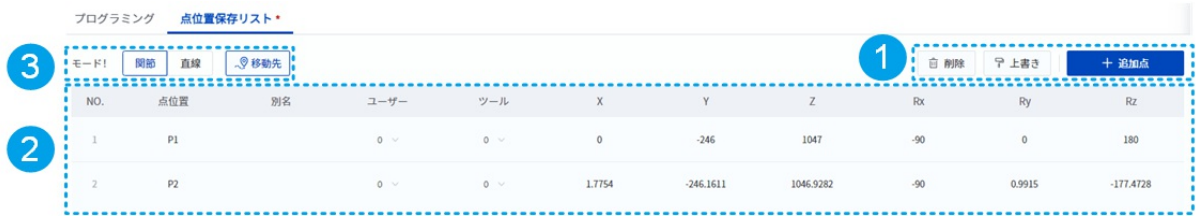
 をクリックすると、ファイルメニューが開き、プロジェクトの新規作成、開く、別名で保存、インポート、エクスポートが可能で、またブロックプロジェクトをスクリプトプロジェクトに変換することもできます。ブロックプロジェクトがスクリプトプロジェクトに変換された後、スクリプトプログラミングで開いて編集できます。

 をクリックすると、現在のプロジェクトを保存できます。プロジェクトが名称未設定の場合は、先にプロジェクト名を入力する必要があります。

6.2.3 保存ポイント

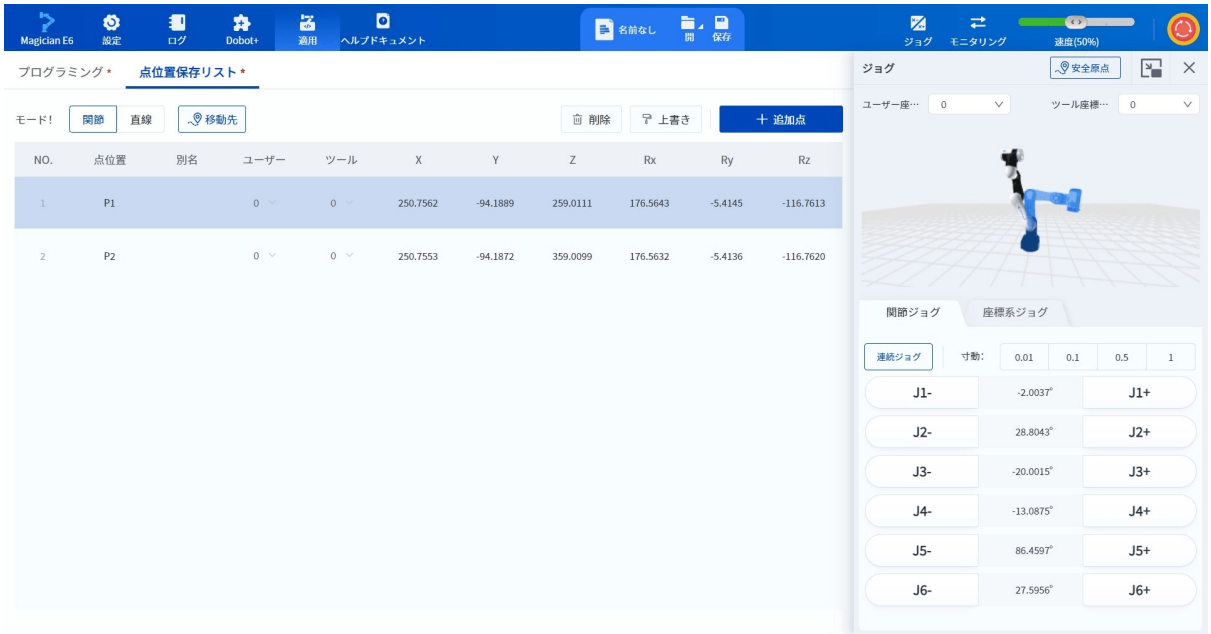
ユーザーはジョグまたはドラッグでロボットを所望のポーズに移動させ、その位置を保存ポイントリストに保存できます。

説明：
保存ポイントリストが開いているとき、CR20Aのエンドフランジ側の**POINT**ボタンを押すと、教示点を追加できます。



番号	説明
1	• ＋ポイント追加ボタンをクリックすると、ロボットの現在のポーズを新しい教示点として保存できます。 • 教示点を選択した後、上書きボタンをクリックすると、ロボットの現在のポーズを使用してその点を上書きできます。 • 教示点を選択した後、削除ボタンをクリックすると、その教示点を削除できます。
2	教示点のリスト。教示点を選択した後、その教示点**NO.**および位置以外の任意の値をクリックすると、その値を直接修正できます。
3	ロボットを現在選択されている位置まで指定されたモードで移動させるように制御します。

保存ポイントリストで選択されている位置に対応するポーズは、下図に示すように、ジョグパネルのシミュレーションエリアに青い輪郭の形で表示されます。

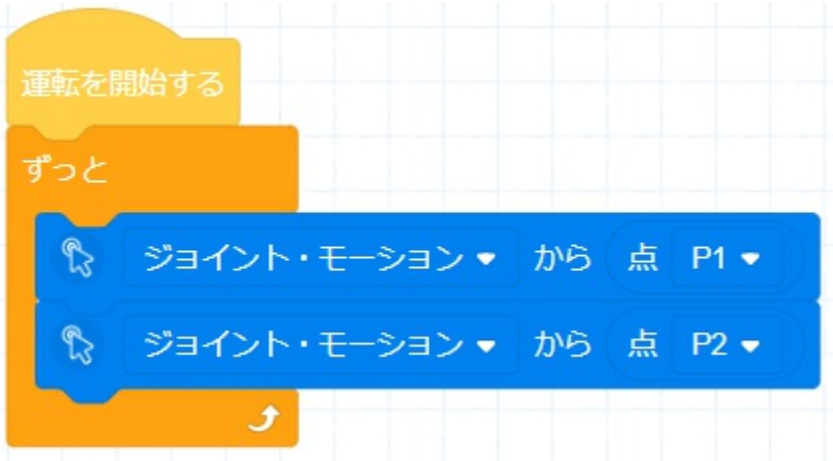


6.2.4 プログラミング

プログラミングを開始する前に、プログラムが実現する機能を明確にしてください。このセクションでは、ロボットを2つの点間でループ移動するように制御するプログラムの作成を例として、ブロックプロジェクトの作成方法を説明します。

> ⚠注意：
> デバッグやプロジェクトの実行を開始する前に、ロボットの動作範囲内に人やその他の障害物がないことを確認してください。

1. 左側のブロックエリアのコントロールブロックセットから繰り返すブロックをキャンバス中の操作を開始するブロックの下にドラッグします。
2. 移動ブロックセット中のポイントまで移動ブロックを繰り返すブロックにドラッグし、位置ドロップダウンボックスをクリックして、P1を選択します。
3. 別のポイントまで移動ブロックを前のポイントまで移動ブロックの下にドラッグし、位置ドロップダウンボックスをクリックして、P2を選択します。



これで、簡単な循環移動プログラムの作成が完了しました。

上記の基本的なプログラミング操作のほかに、DobotStudio Proは以下のプログラミング操作もサポートしています。

- コピー：キャンバスに配置されたブロック上で右クリック（モバイル端末では長押し、以下同じ）してメニューを開くと、選択したブロックとその下に接続されたすべてのブロックをコピーすることができます。または、選択した単一のブロックをコピーすることができます（選択したブロック内に他のブロックがネストされている場合は、一緒にコピーされます）。
- 削除：ユーザーは3つの方法でブロックを削除することができます。
  - ブロックを左側のブロックメニューにドラッグして放します。
  - ブロックをページの右下のゴミ箱アイコンにドラッグして放します。
  - ブロックを右クリックしてメニューを開き、削除を選択します（選択したブロック内に他のブロックがネストされている場合は、一緒に削除されます）。
- サブルーチンに変換：実行開始またはサブスレッド開始に接続されていないブロックのセットを右クリックしてメニューを開き、サブルーチンに変換を選択すると、このブロックのセットをサブルーチンに変換できます。サブルーチンに変換した後、ユーザーはこのブロ

ックセットを1つのサブルーチンブロックで置き換えることができます。これにより、再利用が容易になり、プログラミング効率が向上します。サブルーチン編集画面ではこの操作が無効です。
- ブロックを折りたたむ/展開：ネストされたブロックを右クリックすると、ネストされたブロックを折りたたんだり展開したりして、簡単に表示および編集できます。

キャンバスの右側の アイコンをクリックすると、編集状態になります。

編集状態で、ユーザーはブロックを複数選択またはすべてを選択し、コピーまたは削除できます。

編集を終了をクリックするか、プログラム領域で他の操作を行うと、編集状態が終了します。

その他のブロックに関する詳しい説明はここでは省略します。ブロックメニューの右上の をクリックして確認できます。内容は付属書Bと同じです。

6.2.5 デバッグと実行

> ⚠ 注意：
> デバッグやプロジェクトの実行を開始する前に、ロボットの動作範囲内に人やその他の障害物がないことを確認してください。

プロジェクトを保存した後、デバッグをクリックすると、デバッグモードに入ります。デバッグボタンは シングルステップボタンになります。このボタンをクリックするたびに、ロボットはブロックに対応する命令を実行します。実行が完了した後、 シングルステップボタンを再度クリックして次の命令を実行します。例えば、一つの移動命令をステップ実行した場合、ロボットの移動が終了するまで次の命令をステップ実行することはできません。

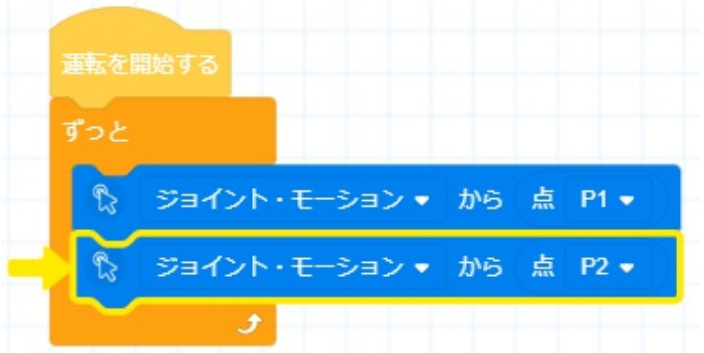
プロジェクトを保存してから 実行をクリックすると、実行モードに入り、 実行ボタンが 一時停止ボタンに変わり、ロボットはプロジェクトの実行を開始します。

- プロジェクトの実行を一時停止するには、 一時停止をクリックします。このとき、ボタ

ンが 続けるに変わり、クリックするとプロジェクトの実行を続けます。

- プロジェクトの実行を停止するには、 停止をクリックします。

デバッグ中または実行中は、次の図に示すように、画面に実行中のブロック命令がハイライト表示されます。ユーザーは実行カーソルへジャンプボタンをクリックしてハイライト表示された行をすばやく見つけることができます。



ユーザーは外部信号 (IOまたはModbus) で指定されたプロジェクトを実行することができます。ユーザーがプロジェクトを編集している場合は、ユーザーにバックアップを求めるポップアップが表示されます。ユーザーは、バックアップするかどうかを選択した後、ソフトウェアのホームページに戻ります。ユーザーが編集しているのは実行するプロジェクトである場合、ロボットは最後に保存されたプロジェクトを実行します。今回編集した内容を実行する必要がある場合は、外部信号によってトリガーされたプロジェクトを停止した後、バックアップファイルを開いて編集を完了し、保存してから実行してください。

外部信号によってトリガーされた実行中のプロジェクトは、アプリケーション画面を開くか、ロボットを再接続します。プロジェクトが実行中であることを示すポップアップが表示されます。ユーザーは、プロジェクトを停止するか、実行中のプロジェクトに入って実行ステータスを表示および制御するかを選択できます。

6.3 スクリプトプログラミング

6.3.1 概要

Dobotロボットは、移動命令、TCP/UDP命令など様々なDobot APIインターフェースを提供し、ユーザが二次開発時に容易に呼び出せるように、Lua言語を使用します。DobotStudio ProはLuaスクリプトのプログラミング環境を提供し、ユーザは自らLuaスクリプトを作成してロボットの運転を制御することができます。

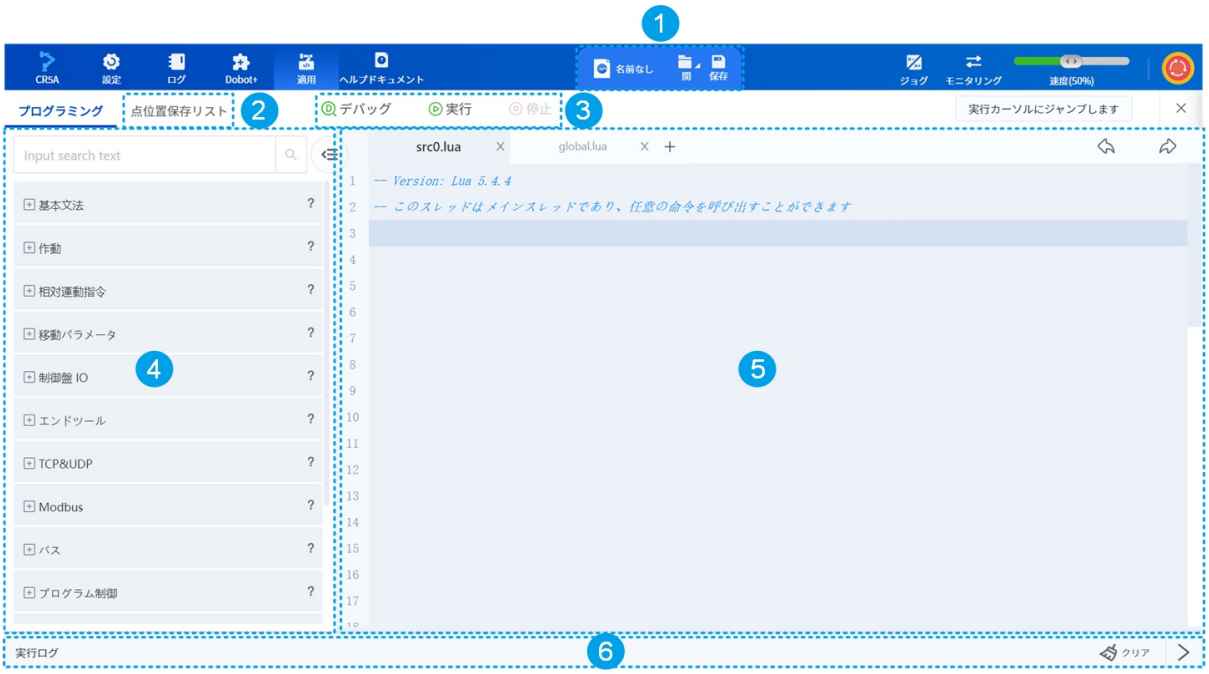
プロジェクト：スクリプトプログラミングはプロジェクト単位で編集および実行し、デバッグをサポートします。アプリケーションホームページのアイコンからプログラミングページに進むと、新たに名前のないプロジェクトが自動的に作成されます。プログラミング後のデバッグや実行には名前を付けて保存する必要があります。プロジェクトはロボットコントローラーに保存され、インポートおよびエクスポートが可能です。

一つのプロジェクトには以下のスクリプトファイルが含まれ、タブ形式で表示されます：

- **src0.lua**ファイルはメインスレッドであり、すべての命令を呼び出すことができます。
- **global.lua**ファイルは変数およびサブ関数の定義にのみ使用します。
- 名称が**src1.lua**～**src4.lua**の0～4個のサブスレッド。サブスレッドはメインプログラムと共に実行する並列プログラムで、最大4つまでです。I/O、変数などを設定することができますが、移動命令を呼び出すことはできません。

保存ポイント：プログラミング中に、ユーザはジョグまたはドラッグ＆ドロップでロボットを移動させ、その後、保存ポイントリストを開いて、ロボットの現在の姿勢をティーチング点として保存することができます。ティーチング点はプロジェクトにバインドされ、命令のパラメータとして使用することができます。

スクリプトプログラミングのメイン画面は下図に示す通りです。



番号	説明
1	プロジェクトの名前を表示し、プロジェクトを管理します。
2	クリックすると保存ポイントの画面が開き、プロジェクト内の保存ポイントを管理できます。
3	現在のプロジェクトのデバッグ、実行、一時停止、停止を制御します。
4	プログラミング関数の検索および使用に使用します。 をクリックすると関数説明ドキュメントを表示することができます。
5	プログラムの作成エリアで、タブをクリックしてスクリプトファイルを切り替えたり、+をクリックしてサブスレッドを追加することができます。 右上角の をクリックすると、プログラミング操作を元に戻す/やり直すことができます。
6	プロジェクトの実行ログを表示する実行ログバー。最右端のアイコンをクリックすると、ログ表示エリアを展開または折りたたむことができ、 クリアをクリックするとログ表示エリアを空にできます。

6.3.2 プロジェクト管理

ブロックプログラミング画面を開くと、新規の空白プロジェクトページがデフォルトで表示され、プロジェクト名が無題です。

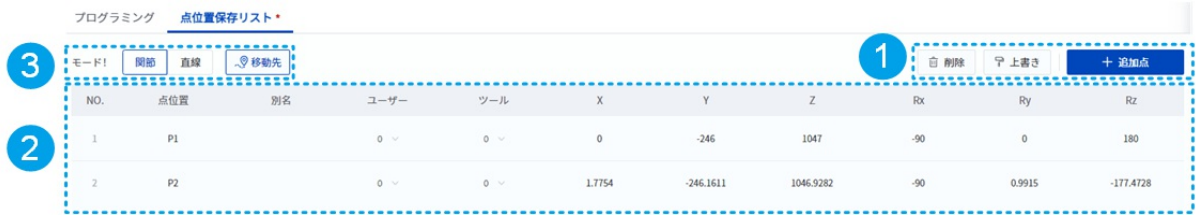
をクリックすると、ファイルメニューが開きます。プロジェクトの新規作成、開く、別名で保存、インポート、エクスポートがサポートされます。

をクリックすると、現在のプロジェクトを保存できます。プロジェクトが名称未設定の場合は、先にプロジェクト名を入力する必要があります。

6.3.3 点位置保存

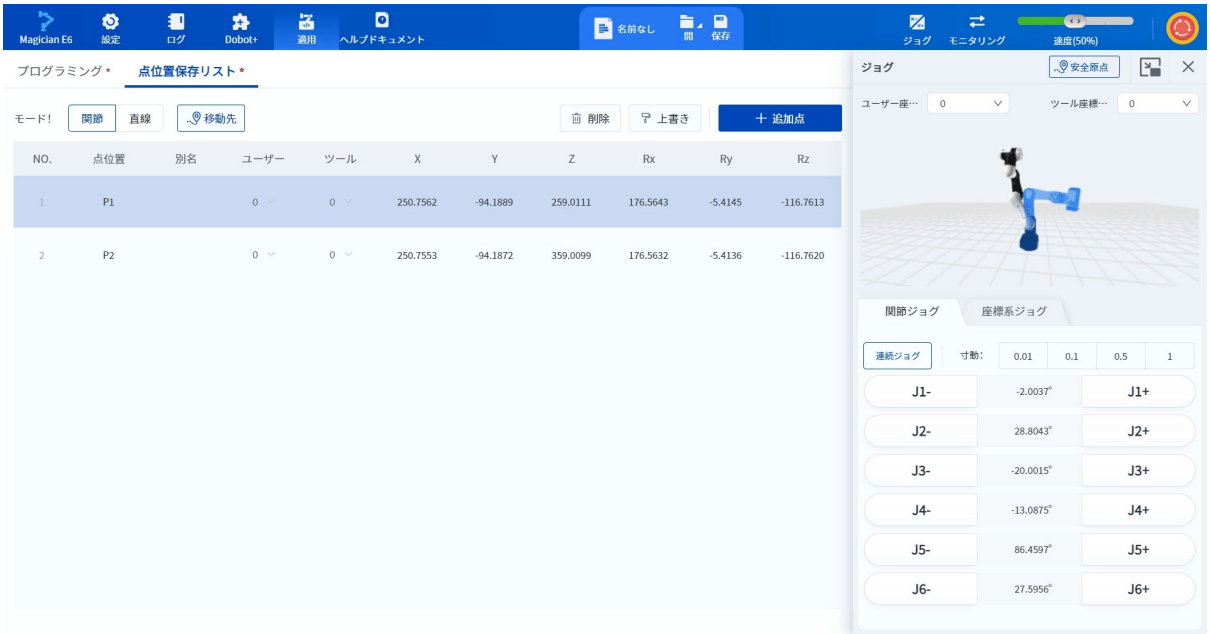
ユーザはジョグまたはドラッグでロボットを所望の位置に移動させ、その点を保存ポイントリストに保存できます。

> 説明：
> ポイントリストが開いているとき、CR20Aのエンドフランジ側の**POINT**ボタンを押すと、教示点を追加できます。



番号	説明
1	• ポイント追加ボタンをクリックすると、ロボットの現在の位置を新しい教示点として保存できます。 • 教示点を選択した後、上書きボタンをクリックすると、ロボットの現在の位置を使用してその点を上書きできます。 • 教示点を選択した後、削除ボタンをクリックすると、その教示点を削除できます。
2	教示ポイントのリスト。教示点を選択した後、その教示点**NO.**および位置以外の任意の値をクリックすると、その値を直接修正できます。
3	ロボットを現在選択されているポイント位置まで指定されたモードで移動させるように制御します。

保存ポイントリストで選択されている位置に対応するポーズは、下図に示すように、ジョグパネルのシミュレーションエリアに青い輪郭の形で表示されます。



6.3.4 プログラミング

ユーザは以下の二つの方法でプログラムを作成することができます：

- 左側の関数メニューから使用したい関数を見つけて、その右側の  をクリックします。パラメータ設定ウィンドウがポップアップ表示されます。ウィンドウでパラメータを設定し、**OK**をクリックすると、パラメータ付きの関数呼び出し命令がカーソルの位置にあるプログラム領域に追加されます。

- 左側の関数メニューで使用したい関数を見つけてダブルクリックし、デフォルトのパラメータを使用して命令をプログラム領域にすばやく挿入し、必要に応じてパラメータ値を変更します。
- 右側のプログラム領域にプログラムを直接入力して作成します。キーボードのTABキーによる命令の自動完了ができます。

プログラミングを開始する前に、プログラムが実現する機能を明確にしてください。このセクションでは、ロボットを2つの点間でループ移動するように制御するプログラムの作成を例として、スクリプトプロジェクトの作成方法を説明します。

1. ポイント保存ページから、ロボットの循環動作の始点P1と終点P2を順に追加します。
2. 上述のいずれの方法で、`while`ループ関数を作成し追加します。
3. ループコードの`end`の前に`MovJ`移動命令を追加し、目標点はP1とします。
4. さらに`MovJ`移動命令を追加し、目標点はP2とします。最終的なコードは以下の通りです。

```
while(true)
do
    MovJ(P1)
    MovJ(P2)
end
```

これで、簡単な循環動作プログラムの作成が完了しました。

スレッドを作成する必要がある場合は、プログラミングエリアの上部にあるタブの右側にある+をクリックしてスレッドを追加し、その後、スレッドタブに切り替えてプログラムを作成します。

スクリプトプログラミングに関するその他の説明はここでは省略します。関数メニュー中の をクリックして確認できます。内容は付属書Cと同じです。

6.3.5 デバッグと実行

> ⚠ 注意：
> ロボットのデバッグまたは実行を開始する前に、ロボットの作業範囲内に人や他の障害物がないことを確認してください。

プロジェクトを保存した後にデバッグをクリックすると、プロジェクトはデバッグモードに入ります。

- デバッグボタンはシングルステップボタンに変わり、このボタンをクリックするたびにロボットはスクリプト命令の行を実行します。実行が完了すると、シングルステップボタンを再度クリックして次の行の命令を実行します。例えば、一つの動作命令をステップ実行した場合、ロボットの動作が終了するまで次の行の命令をステップ実行することはできません。
- 実行ボタンがブレークポイントまで実行ボタンに変わり、ブレークポイントのデバッグが可能になります。コード行の左側にある番号をクリックするとブレークポイントを設定または解除することができます。そして、ブレークポイントまで実行ボタンをクリックすると、ロボットはブレークポイントがあるコード行まで命令を実行し続け、その後、中断状態になります。ユーザがブレークポイントまで実行ボタンを押した後、実行するコードにブレークポイントがない場合、プログラムは終了するまで実行されます。

プロジェクトを保存してから実行をクリックすると、実行モードに入り、実行ボタンが一時停止ボタンに変わり、ロボットはプロジェクトの実行を開始します。

- プロジェクトの実行を一時停止するには、一時停止をクリックします。このとき、ボタンが続けるに変わり、クリックするとプロジェクトの実行を続けます。
- プロジェクトの実行を停止するには、停止をクリックします。

デバッグ中または実行中は、次の図に示すように、画面に実行中のコード行がハイライト表示されます。ユーザーは実行カーソルへジャンプボタンをクリックしてハイライト表示された行をすばやく見つけることができます

```
1 -- Version: Lua 5.4.4
2 -- This thread is the main thread and can call any commands.
3 while(true)
4 do
5 MovJ(P1)
6 MovJ(P2)
7 end
```

ユーザーは外部信号 (IOまたはModbus) で指定されたプロジェクトを実行することができます。ユーザーがプロジェクトを編集している場合は、ユーザーにバックアップを求めるポップアップが表示されます。ユーザーは、バックアップするかどうかを選択した後、ソフトウェアのホームページに戻ります。ユーザーが編集しているのは実行するプロジェクトである場合、ロボットは最後に保存されたプロジェクトを実行します。今回編集した内容を実行する必要がある場合は、外部信号によってトリガーされたプロジェクトを停止した後、バックアップファイルを開いて編集を完了し、保存してから実行してください。

外部信号によってトリガーされた実行中のプロジェクトは、アプリケーション画面を開くか、ロボットを再接続します。プロジェクトが実行中であることを示すポップアップが表示されます。ユーザーは、プロジェクトを停止するか、実行中のプロジェクトに入って実行ステータスを表示および制御するかを選択できます。

6.4 Pythonプログラミング (Magician E6)

6.4.1 概要

Pythonプログラミングは、**PC**が**Magician E6**ロボットに接続されている場合にのみサポートされます

Dobotロボットは、Python言語を使用して、移動命令、TCP/UDP命令などの豊富なDobot APIインターフェースを提供します。これは、ユーザーが二次開発中に呼び出すのに便利です。
DobotStudio ProはPythonプログラミング環境を提供し、ユーザーは独自のPythonプログラムを作成してロボットの動作を制御できます。

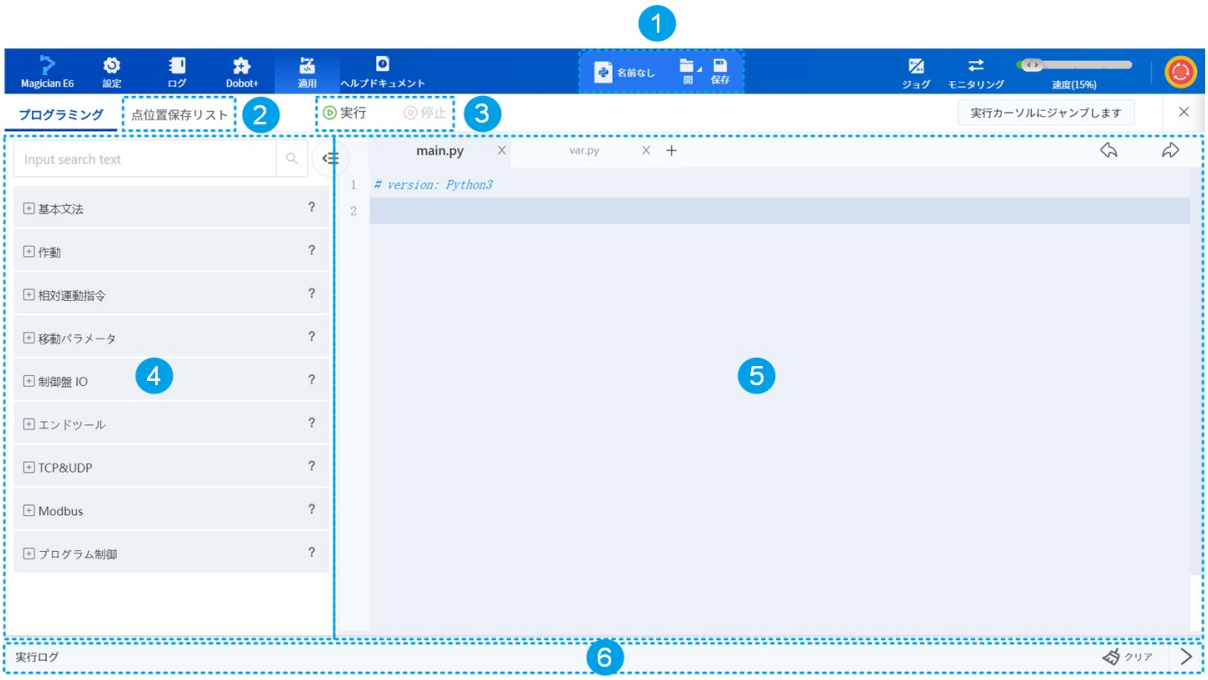
プロジェクト：Pythonプログラミングはプロジェクト単位で編集および実行し、デバッグをサポートします。アプリケーションホームページのアイコンからプログラミングページに進むと、新たに名前のないプロジェクトが自動的に作成されます。プログラミング後のデバッグや実行には名前を付けて保存する必要があります。プロジェクトはロボットアームコントローラーに保存され、インポートおよびエクスポートが可能です。

プロジェクトには、タブ形式で表示される次のPythonファイルが含まれます。

- **main.py**ファイルはメインスレッド、任意の命令を呼び出すことができます。
- **var.py**ファイルは変数を定義するためにのみ使用されます。
- 0〜4つのサブスレッド、名前は**script1.py**〜**script1.py**。サブスレッドはメインプログラムと共に実行する並列プログラムで、最大4つまでです。I/O、変数などを設定することができますが、移動命令を呼び出すことはできません。

保存ポイント：プログラミング中に、ユーザーはいつでもジョグまたはドラッグでロボットを移動させ、その後、保存ポイントリストを開いて、ロボットの現在のポーズを教示点として保存することができます。教示点はプロジェクトに紐づけられ、命令のパラメータとして使用できます。

Pythonプログラミングのメイン画面は以下の図のようになります。



番号	説明
1	プロジェクトの名前を表示し、プロジェクトを管理します。
2	クリックすると保存ポイントの画面が開き、プロジェクト内の保存ポイントを管理できます。
3	現在のプロジェクトの実行、一時停止、停止を制御します。
4	プログラミング関数の検索および使用に使用します。 をクリックすると関数説明ドキュメントを表示することができます。
5	プログラム作成領域では、タブをクリックしてPythonファイルを切り換えたり、+をクリックしてサブスレッドを追加したりできます。 右上の をクリックすると、プログラミング操作を元に戻す/やり直すことができます。
6	プロジェクトの実行ログを表示する実行ログバー。最右端のアイコンをクリックすると、ログ表示エリアを展開または折りたたむことができ、 クリアをクリックするとログ表示エリアを空にできます。

6.4.2 プロジェクト管理

ブロックプログラミング画面を開くと、新規の空白プロジェクトページがデフォルトで表示され、プロジェクト名が無題です。

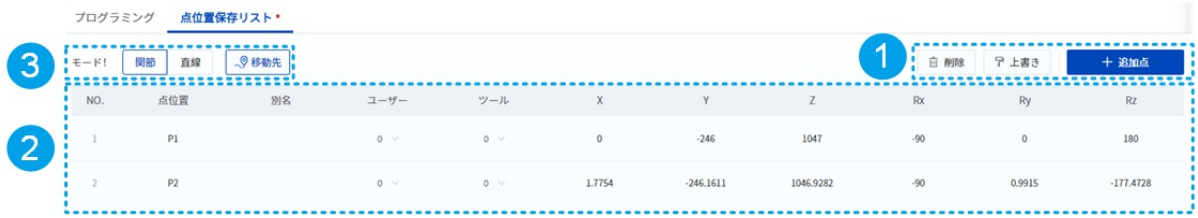
をクリックすると、ファイルメニューが開き、プロジェクトの新規作成、開く、別名で保存、インポート、エクスポートが可能です。

をクリックすると、現在のプロジェクトを保存できます。プロジェクトが名称未設定の場合は、先にプロジェクト名を入力する必要があります。

6.4.3 保存ポイント

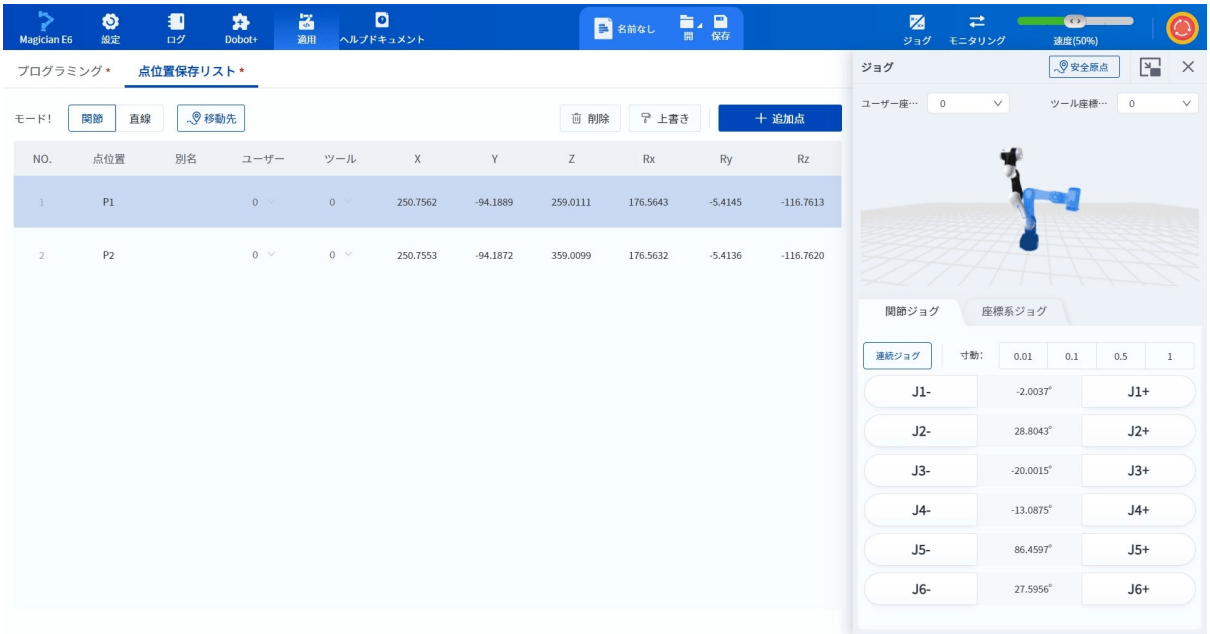
ユーザーはジョグまたはドラッグでロボットを所望のポーズに移動させ、その位置を保存ポイントリストに保存できます。

> 説明：
> 保存ポイントリストが開いているとき、CR20Aのエンドフランジ側の**POINT**ボタンを押すと、教示点を追加できます。



番号	説明
1	• ポイント追加ボタンをクリックすると、ロボットの現在のポーズを新しい教示点として保存できます。 • 教示点を選択した後、上書きボタンをクリックすると、ロボットの現在のポーズを使用してその点を上書きできます。 • 教示点を選択した後、削除ボタンをクリックすると、その教示点を削除できます。
2	教示点のリスト。教示点を選択した後、その教示点**NO.**および位置以外の任意の値をクリックすると、その値を直接修正できます。
3	ロボットを現在選択されている位置まで指定されたモードで移動させるように制御します。

保存ポイントリストで選択されている位置に対応するポーズは、下図に示すように、ジョグパネルのシミュレーションエリアに青い輪郭の形で表示されます。



6.4.4 プログラミング

ユーザーは次のような集中的な方法で命令を挿入できます。

- 左側の関数メニューから使用したい関数を見つけて、その右側の をクリックします。パラメータ設定ウィンドウがポップアップ表示されます。ウィンドウでパラメータを設定し、**OK**をクリックすると、パラメータ付きの関数呼び出し命令がカーソルの位置にあるプログラム領域に追加されます。

- 左側の関数メニューで使用したい関数を見つけてダブルクリックし、デフォルトのパラメータを使用して命令をプログラム領域にすばやく挿入し、必要に応じてパラメータ値を変更します。
- 右側のプログラム領域にプログラムを直接入力して作成します。キーボードのTABキーによる命令の自動完了ができます。

プログラミングを開始する前に、プログラムが実現する機能を明確にしてください。このセクションでは、ロボットを2つの点間でループ移動するように制御するプログラムの作成を例として、スクリプトプロジェクトの作成方法を説明します。

1. 保存ポイントページから、ロボットのループ移動の始点P1と終点P2を順に追加します。
2. `while` ループ関数を追加します。
3. ループコードに1つの `MovJ` 移動命令を追加し、目標点はP1です。
4. さらに1つの `MovJ` 移動命令を追加し、目標点はP2とします。最終的なコードは以下の通りです。

```
while True:
    MovJ(P1)
    MovJ(P2)
```

これで、簡単な循環移動プログラムの作成が完了しました。

サブスレッドを作成する必要がある場合は、プログラム領域の上部にあるタブの右側にある+をクリックしてサブスレッドを追加し、その後、サブスレッドタブに切り替えてプログラムを作成します。

スクリプトプログラミングに関するその他の説明はここでは省略します。関数メニュー中の ? をクリックして確認できます。内容は付属書Eと同じです。

6.4.5 実行

> ⚠ 注意：
> プロジェクトの実行を開始する前に、ロボットの動作範囲内に人やその他の障害物がないことを確認してください。

プロジェクトを保存してから ▷ 実行をクリックすると、実行モードに入り、▷ 実行ボタンが ⏸ 一時停止ボタンに変わり、ロボットはプロジェクトの実行を開始します。

- プロジェクトの実行を一時停止するには、⏸ 一時停止をクリックします。このとき、ボタ

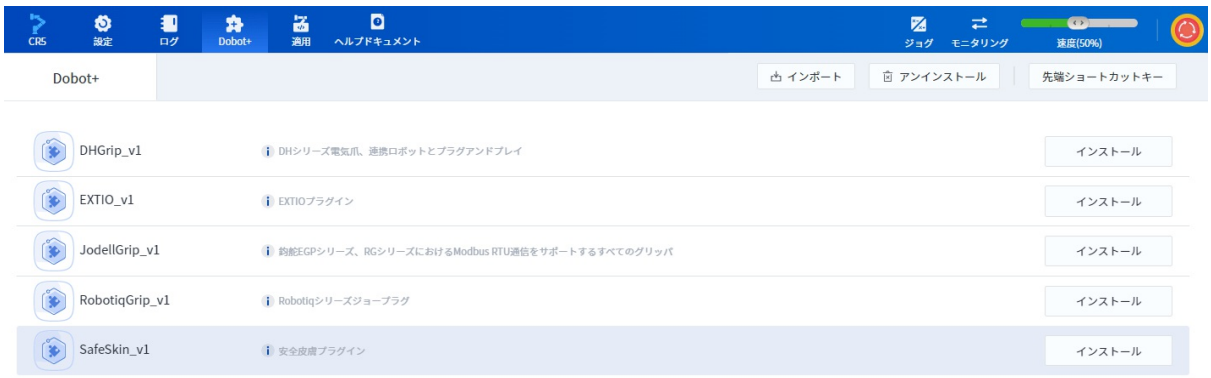
ンが  続けるに変わり、クリックするとプロジェクトの実行を続けます。

- プロジェクトの実行を停止するには、  停止をクリックします。

実行中は、画面に実行中のコード行がハイライト表示されます。

7 Dobot+

Dobot+プラグインを使用すると、Dobotのエコアクセサリを簡単に設定と使用でき、二次的な開発を省略します。



プラグインの右側のインストールボタンをクリックすると、そのプラグインをインストールできます。また、タイトルバーのインポートボタンをクリックしてプラグインをアップロードすることもできます。プラグインを選択した後、タイトルバーのアンインストールボタンをクリックすると、選択したプラグインをアンインストールできます。

プラグインがインストールされると、インストールボタンは開くボタンに変わり、クリックすると新しいタブでプラグインが開きます。複数のプラグインを同時に開くことができます。プラグインによって使い方が異なるので、このドキュメントでは詳しく説明しません。

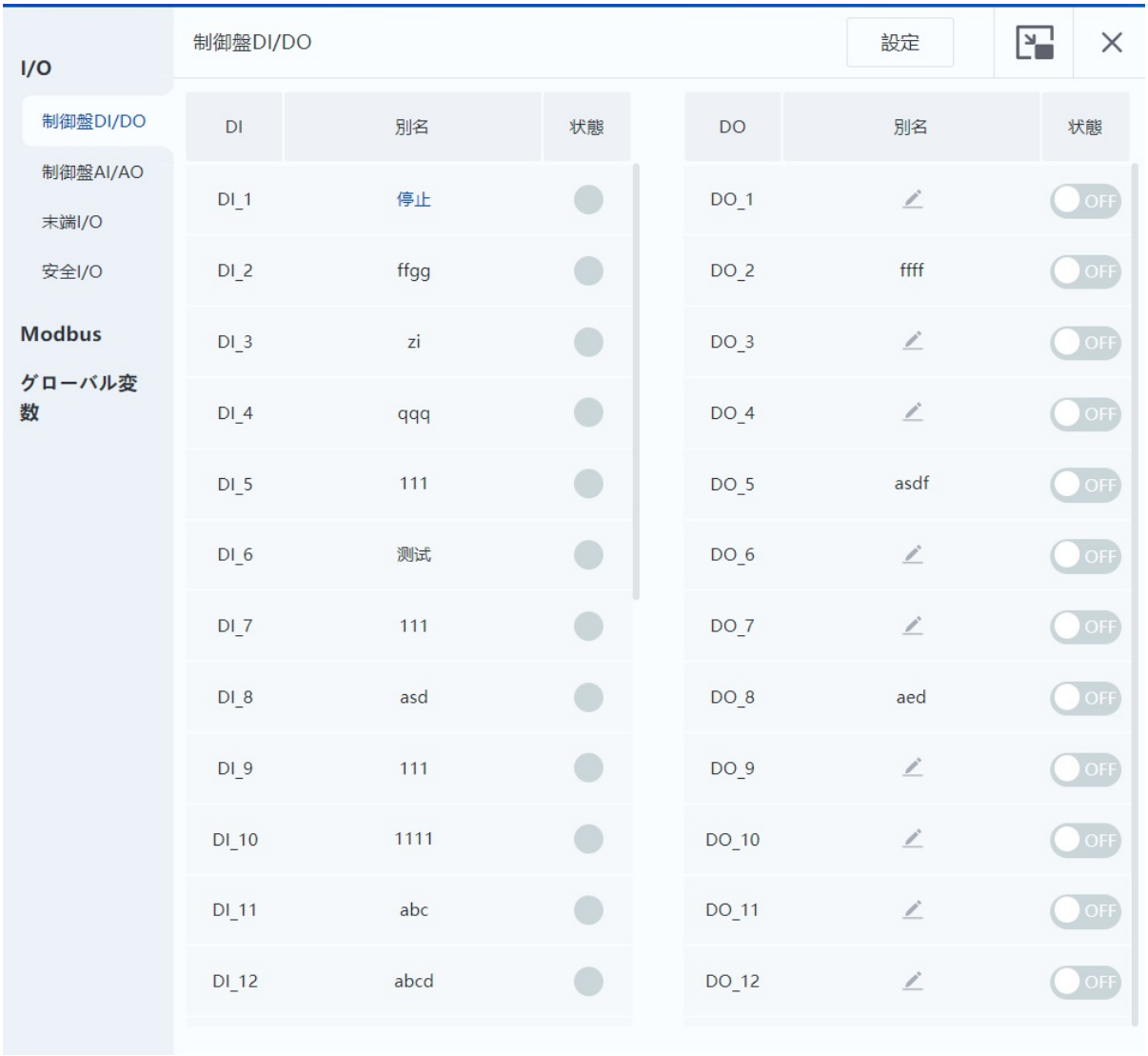
プラグインを追加すると、ブロックプログラミングおよびスクリプトプログラミングにもプラグイン関連のブロック/関数が新たに追加され、プロジェクト実行中にエコシステムアクセサリを制御するために使用されます。

8 モニタリング

8.1 コントロールキャビネットDI/DO

8.1.1 DI/DO監視

このページは、制御キャビネットのDIとDOの状態と機能を監視および設定するために使用します。



ページ中央の領域では、DI/DOの別名とステータスを表示/設定できます。

コントロールキャビネットDI/DOはデフォルトで共通**IO**であり、システム**IO**と設定されることができます。

- 共通**IO**の機能はユーザが定義します。エイリアスはデフォルトでは空です。をクリックすると変更することができます。ユーザはエイリアスを使ってIOの機能をメモすると、今後プログラミングなどの操作で確認することができます。変更された場合、下線付きの黒い

テキストで表示され、テキストをクリックすると再変更することができます。

- システム**IO**は、次の**IO**設定ページで特定の機能が設定されたDI/DOです。エイリアスは下線なしの青いテキストで表示されます。また、変更することはできません。

DIの右側にある円形の表示灯は、該当するDIの現在の状態を示します。灰色はOFF、緑はONを示します。

DOの右側にあるスイッチは該当するDOの現在の状態を示し、スイッチをクリックすることでON/OFFを切り替え、該当するDOの状態を制御します。システム**IO**は手動で状態を切り替えることはできません。

ページの上にある設定ボタンをクリックすると、設定ページに移動します。このページには3つのタブがあります。

8.1.2 I/O設定

仮想**DI**

DIの右側にある  をクリックすると、アイコンが  に変わり、保存すると対応するDIが仮想DIに切り替わります。DIが仮想DIに設定されると、監視ページのインジケータライトはDOと同じスイッチになり、クリックすると仮想DIのON/OFFを切り替えることができます。仮想DIを切り替えることで、ユーザは外部DIのデバイスの入力をシミュレートし、DI関連の機能をデバッグすることができます。例えば、プロジェクトの実行中にDI関連の判断条件を満たし、プロジェクトを続行することができます。

 注：
仮想DIを設定すると持続的に有効になり、プロジェクトが実行中に対応するDIを命令で読み取ると、仮想値を読み取り、そのDIの実際の値は取得できません。この状況を回避したい場合は、プロジェクトを実行する前にDIを実際のDIに切り替えてください。

システム**IO**

機能設定列のドロップダウンボックスを使用してDI/DOのリモート制御機能を設定できます。対応する機能の説明は以下の通りです。

DI機能	説明
開始	ロボットがアイドル状態になると、指定されたプロジェクトの実行が開始されます。詳細については以下の選択プロジェクトをご参照ください。 ロボットが停止しているとき、プロジェクト（またはその他の形式の命令シーケンス）の実行を続行します。
停止	実行中のプロジェクト（または他の形式の命令シーケンス）を停止します。
一時停止	実行中のプロジェクト（または他の形式の命令シーケンス）を一時停止します。
イネーブル	ロボットの電源が入っている場合、ロボットをイネーブルにするよう制御します
ディセーブル	ロボットがイネーブルになっている場合、ロボットをディセーブルにするよう制御します
アラームをクリア	ロボットの現在のアラームをクリアします。
ドラッグを開始する	ロボットがイネーブルになっている場合、ロボットがドラッグモードに入るよう制御します。
ドラッグを終了する	ロボットがドラッグ モードにある場合、ロボットがドラッグモードを終了するよう制御します。

DO機能	説明
動作状態	ロボットがプロジェクトを実行しているときには1を出力します。そうでない場合は0を出力します。
停止状態	ロボットが停止しているときには1を出力し、そうでない場合は0を出力します。
一時停止状態	ロボットが一時停止状態になると1を出力し、それ以外の場合は0を出力します。
安全原点状態	ロボットが安全原点にある場合は1を出力し、そうでない場合は0を出力します。
安全皮膚停止状態	ロボットがセーフティスキンでトリガされた一時停止状態になると1を出力し、そうでない場合は0を出力します。
アイドル状態	ロボットがアイドル状態 (イネーブル、停止中かつアラームなし) にある場合は1を出力し、それ以外の場合は0を出力します。
電源投入状態	ロボットの電源が入っている場合は1を出力します。そうでない場合は0を出力します。
イネーブル状態	ロボットをイネーブルするときに1を出力し、そうでない場合は0を出力します。
アラーム状態	ロボットにクリアされていないアラームがある場合は1を出力し、そうでなければ0を出力します。
衝突状態	ロボットが衝突を検出した場合は1、それ以外の場合は0を出力します。
ドラッグ状態	ロボットがドラッグ状態の場合は1を出力し、そうでない場合は0を出力します。
実行されていない時は低	プロジェクトまたはTCP命令シーケンスが実行されていない、一時停止または停止している場合は0を出力し、実行中は命令で対応するIO状態を設定できます。
未稼働時に高い	プロジェクトまたはTCP命令シーケンスが稼働していない、一時停止または停止しているときに1を出力し、稼働中に命令で対応するIO状態を設定できます。
異常停止時に低い	ロボットが異常停止したときに0を出力し、稼働中に命令で対応するIO状態を設定できます。 以下の状況が異常停止を引き起こします： • 安全機能停止（例：衝突検出、安全壁と安全領域、安全IO等） • ロボットのアラーム • プロジェクトの異常稼働

機能が候補プロジェクトのDIは、このページでは機能設定を行いません。最初に「プロジェクトの選択」ページでDIをリリースする必要があります。

設定を変更したら、ページ右上の保存ボタンをクリックして設定を完了します。

⚠️注意：

- 全てのリモートトリガ源が同時に有効になります。デバイスと生産の安全のため、ロボットは一つの制御源（制御ソフトウェア/DI/Modbus）からのみ起動するようにしてください。
- IOトリガの有効性は、手動/自動モードとIO/Modbus設定によっても影響を受けます。詳細は対応する機能の関連説明を参照してください。

プロジェクトを選択

デフォルトの実行プロジェクトを選択すると、開始DIがトリガーされたときに、ここで選択したプロジェクトが直接実行されます。

選択ボタンをクリックすると、プロジェクト選択ボックスが表示され。

削除をクリックすると、現在選択されているプロジェクトが空になります。

グループIO選択プロジェクトを選択すると、グループI/Oを使用して複数のプロジェクトを設定できます。

1. + または - をクリックすると、グループI/Oに割り当てられるアドレスの数を増減できます。割り当てられたアドレスの数（最大4個）が多いほど、設定可能なプロジェクトの数が増えます。

    - 1つのアドレス：2つのプロジェクトを設定可能
    - 2つのアドレス：4つのプロジェクトを設定可能
    - 3つのアドレス：8つのプロジェクトを設定可能
    - 4つのアドレス：16つのプロジェクトを設定可能

2. ドロップダウンボックスを使用して、割り当てられたアドレスを変更できます。割り当てられたグループI/Oのアドレスは、リモートI/Oまたは安全I/O（CCBOX）と重複してはなりません。

3. アドレスを割り当てた後、各グループのIO値に対してプロジェクトを設定できます。実際のニーズに合わせて設定すればよいので、空白のままにしておいても構いません。

    - プロジェクトを稼働する前に、対応する（緑色はON、灰色はOFF）グループIOの値を設定して、対応するプロジェクトを選択します。対応するグループIOプロジェクトでプロジェクトが選択されていない場合、開始DIがトリガされたときにロボットはプロジェクトを実行しません。また、コントローラがアラームを鳴らします。

    - 上記の図では、DI1～DI4の4つのアドレスを割り当てる例を示しています：

      DI1～DI4はOFFの場合、第1のプロジェクトを選択したことを意味します。

      DI1とDI2はON、DI3、DI4はOFFの場合、第3のプロジェクトを選択したことを意味します。

DI1～DI4がすべてONの場合、第16のプロジェクトを選択したことを意味します。

設定を変更したら、保存をクリックして設定を完了します。

詳細設定

トリガモード

DIの機能がどのようにトリガされるかを設定します。立ち上がりエッジは、DIがOFFからONに変化したときに設定された機能をトリガすることを示し、立ち下がりエッジは、DIがONからOFFに変化したときに設定された機能をトリガすることを示します。

モード選択

CRAシリーズは実際のハードウェア接続モードに応じてDOのモードを選択する必要があります。詳細は対応するハードウェア使用マニュアルをご参照ください。

設定を変更したら、保存をクリックして設定を完了します。

8.2 コントロールキャビネットAI/AO

このページは、制御キャビネットのAIとAOの状態とモードを監視および設定するために使用します。

> 注：
> Magician E6ロボットに接続するとこのページは表示されません。

アナログ出力はコントロールキャビネットAIインターフェースの実際の値を表示し、電圧と電流の2つの検出モードをサポートし、変更ボタンをクリックすると切り替えることができ、AOは手動で出力値を変更でき、設定後は変更確認ボタンをクリックすると有効になります。

8.3 エンドI/O

このページは、ロボットエンドのIO状態とモードを監視するために使用されます。

> 説明：
> 以下の図は、単一の航插のCRAシリーズを例にしています。複数の航插を持つ機種（CR20A）については、複数のタブが表示されます。
> Magician E6エンドには2ウェイDIと2ウェイDOのみがあり、RS485およびAIインターフェースはありません。

DI/DO

DI/DOエイリアス列の または テキストをクリックしてエイリアスを変更できます。

DIの右側にある円形の表示灯は、該当するDIの現在の状態を示します。灰色はOFF、緑はONを示します。

DOの右側にあるスイッチは該当するDOの現在の状態を示し、スイッチをクリックすることでON/OFFを切り替え、該当するDOの状態を制御します。

仮想**DI**

右上の設定ボタンで仮想DIを設定できます。DIが仮想DIとして設定された後、インジケータはスイッチに変わり、クリックすると仮想DIのON/OFFを切り替えることができます。仮想DIを切り替えることで、ユーザは外部DIのデバイスの入力をシミュレートし、DI関連の機能をデバッグすることができます。例えば、プロジェクトの実行中にDI関連の判断条件を満たし、プロジェクトを続行することができます。

注：
仮想DIを設定すると持続的に有効になり、プロジェクトが実行中に対応するDIを命令で読み取ると、仮想値を読み取り、そのDIの実際の値は取得できません。この状況を回避したい場合は、プロジェクトを実行する前にDIを実際のDIに切り替えてください。

末端作業モード

- 作業モードが**RS485**の場合、末端IOピン1と2の機能は485Aと485Bです；
- 作業モードがアナログ入力の場合、末端IOピン1と2の機能はAI_1とAI_2で、この画面で入力値をリアルタイムで監視できます。

8.4 安全I/O

このページは、安全なI/Oのステータスを表示し、安全なI/Oの機能を設定するために使用されます。

[安全I/Oのステータス]列の指示灯は、対応するI/Oのステータスを示します。緑色がハイレベル、灰色がローレベルです。

安全IOは2ウェイ設定です。2ウェイSIの2つのステータスライトは、それぞれ2ウェイのステータスを示します。トリガロジックは設定機能によって異なります。2ウェイSOの2ウェイ状態は同期であり、1つのステータスランプだけで表示されます。

> 説明：
> Magician E6の安全IOは制御キャビネットDI/DOと共有接続端子を使用しており、実際のニーズに応じて単路または2ウェイ設定を行うことができます。システムIOとして設定された端子は、安全IO機能を再設定することはできません。逆も同様です。

クリックして設定ボタンを使用して安全I/Oの機能設定を変更することができ、変更後はクリック保存ボタンをクリックして設定を完了します。

設定可能な**SI**機能

機能	説明
ユーザによる非常停止	ユーザエマージェンシー入力は、ユーザが使用するためのエマージェンシーインターフェースで、ユーザは外部エマージェンシー装置を接続できます。 ユーザエマージェンシー入力はデフォルトで高レベル常閉信号入力で、いずれかの路が低レベルになるとロボットが非常停止状態になります。この機能はデフォルトで急停止状態の出力をトリガーしますが、一部の使用シーンでは急停止が自己ロックする可能性があります。この状況を避けるために、制御ソフトウェアで設定可能なSIインターフェースをユーザの急停止入力（ステータスなしの出力）に変更し、対応するインターフェースをユーザの急停止入力として使用します。 SI_1とSI_2はこの機能に固定設定されています。
保護停止	保護停止入力は外部保護デバイス（例：安全扉、安全光幕など）のインターフェースです。 保護停止入力はデフォルトで高レベルの常閉信号入力であり、いずれかの低レベル信号がトリガーされるとロボットは保護停止状態（一時停止状態）になります。 • 保護停止リセット入力インターフェースを設定した場合、保護停止入力信号を復元し、保護停止リセット入力をトリガーした後で保護停止状態を解除できます。その後、上位マシンで運転を再開することを確認した後、ロボットの運転が再開されます。 • 保護停止リセット入力インターフェースを設定していない場合、保護停止入力信号を復元するだけで保護停止状態を解除し、ロボットの運転が再開されます。 SI_3とSI_4はこの機能に固定設定されています。
保護停止リセット	保護停止リセット入力は保護停止状態をリセットするためのインターフェースです。 保護停止リセット入力はデフォルトで高レベルの常開信号入力であり、両方の上昇エッジが同時にトリガーされると保護停止状態がリセットされます。
	縮小モード入力はユーザがロボットを縮小モードに制御するための

低減モード	インターフェースで、縮小モードでは、ロボットの運動パラメータ（関節速度、TCP速度）は縮小モードの範囲内に制限されます。 縮小モード入力はデフォルトで高レベルの常閉信号入力であり、いずれかの低レベル入力がトリガーされるとロボットは縮小モード状態になります。高レベル入力が復元されると、ロボットは縮小モードを終了し、通常モードになります。

⚠ 注意：
実際の使用時には、SI信号の変化間隔を150ms以上に保つようにしてください。そうしないと、SI信号のジッターがロボットの運転状態異常を引き起こす可能性があります。
例えば、保護停止リセット入力が設定されていない場合、保護停止入力信号のジッター（変化間隔が150ms未満）がロボットの一時停止後、自動的に運転を再開しない可能性があります。以下のいずれかの方法で解決できます：

- 制御ソフトウェアでプロジェクトを一時停止し、その後プロジェクトを再開します。
- 保護停止入力信号を再度トリガーし、150ms以上保持します。

設定可能な**SO**機能

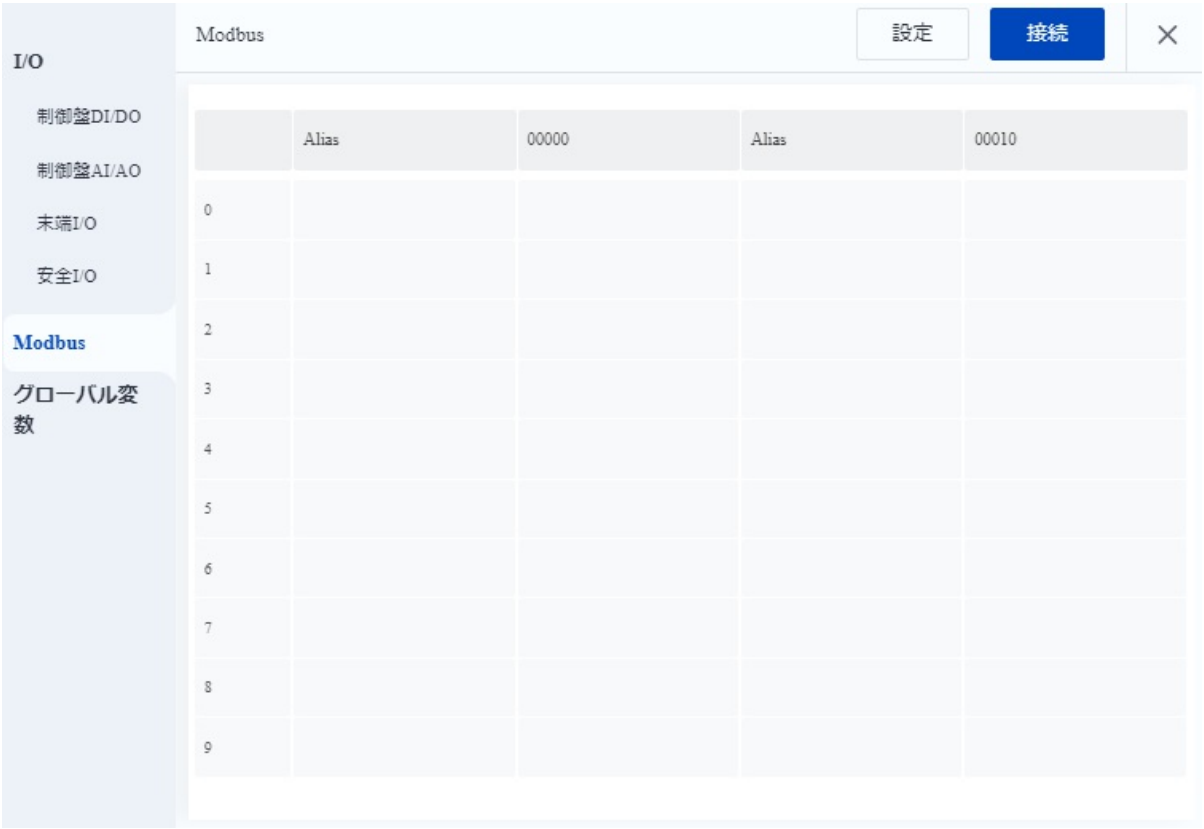
機能	説明
急停止状態出力	ロボットが非常停止状態にある場合、出力電圧は低レベルです；それ以外の場合は出力電圧は高レベルです。 どのような源からの急停止もこの出力をトリガーします。 SO_1とSO_2はこの機能に固定設定されています。
非停止状態出力	ロボットが自動運転状態の場合、非停止状態であり、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。 この状態の判断基準は、関節が動いているかどうかでなく、ロボットがプロジェクトを実行しているかどうかです。例えば、プロジェクトが実行中で、プログラムが指定DIがONになるのを待っていて、ロボットが動いていない場合、これは非停止状態であり、出力は低レベルです。プロジェクトが一時停止状態の場合、これは停止状態であり、出力は高レベルです。
縮小モード状態出力	ロボットが縮小モードの場合、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。
運動中の状態出力	ロボットの一つ以上の関節が**1°/s**以上移動している場合（ドラッグモードを除く）、運動中の状態となり、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。
安全原点状態出力	ロボットが安全原点の位置にある場合、出力電圧は高レベルです。それ以外の場合、出力電圧は低レベルです。 安全原点は、安全設定で変更することができます。
保護停止状態出力	ロボットが保護停止状態の場合、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。
システム非常停止	ロボットがシステム非常停止状態の場合、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。

状態出力	システム非常停止は、非常停止ボタンまたはソフトウェアによる非常停止がトリガーとなります。
ユーザ非常停止状態出力	ロボットがユーザ非常停止状態の場合、出力電圧は低レベルです。それ以外の場合、出力電圧は高レベルです。 ユーザ非常停止は、安全IOがトリガーとなる非常停止です。

8.5 Modbus

8.5.1 Modbus監視

このページは、上位機がModbusマスターとしてロボットコントローラー内蔵のModbusスレーブに接続し、レジスタの値を表示および変更し、コントローラーがスレーブとして動作する際のリモートModbus制御機能を設定するためのものです。



ページ上の接続ボタンをクリックして、接続するスレーブを設定します。

- スレーブ**IP**：Modbus機器のアドレス。制御キャビネットのModbusスレーブを接続する場合、制御キャビネットのIPアドレスを入力します。例：192.168.200.1。
- ポート：Modbus通信のポート番号。制御キャビネットのModbusスレーブを接続する場合に502を入力します。
- スレーブ**ID**：スレーブ機器のID。
- 機能コード：スレーブ機器の機能タイプを選択します。
- アドレス/数量：レジスタのアドレスと数量。制御キャビネットのModbusスレーブを接続する場合、付録A Modbusレジスタの定義を参考にしてください。
- スキャン頻度：ロボットがスレーブをスキャンする時間間隔。

接続完了後、ページ中央のテーブルには、スレーブの各アドレスのエイリアスと数量が表示され、エイリアスのセルをクリックしてエイリアスを修正します。レジスタタイプがコイルレジスタまたはホールドレジスタの場合、レジスタ値のセルをダブルクリック（PC端末）またはシングルクリック（モバイル端末）で値を変更できます。

ページ上の設定ボタンをクリックすると、設定画面に入ることができ、2つのタブがあります。

8.5.2 Modbus設定

トリガモードは、コイルレジスタ機能のトリガ方法を設定するためのものです。上昇は、コイルレジスタが0から1に変わるときに設定の機能をトリガし、下降はコイルレジスタが1から0に変わるときに設定の機能をトリガすることを示します。

コイルレジスタとコンタクトレジスタのアドレス情報は閲覧のみ可能で、変更はできません。対応する機能説明は以下の通りです。

コイルレジスタ機能	説明
開始	ロボットがアイドル状態になると、指定されたプロジェクトの実行が開始されます。詳細については以下の選択プロジェクトをご参照ください。 ロボットが停止しているとき、プロジェクト（またはその他の形式の命令シーケンス）の実行を続行します。
停止	実行中のプロジェクト（または他の形式の命令シーケンス）を停止します。
一時停止	実行中のプロジェクト（または他の形式の命令シーケンス）を一時停止します。
イネーブル	ロボットの電源が入っている場合、ロボットをイネーブルにするよう制御します
ディセーブル	ロボットがイネーブルになっている場合、ロボットをディセーブルにするよう制御します

アラームをクリア	ロボットの現在のアラームをクリアします。
ドラッグを開始する	ロボットがイネーブルになっている場合、ロボットがドラッグモードに入るよう制御します。
ドラッグを終了する	ロボットがドラッグ モードにある場合、ロボットがドラッグモードを終了するよう制御します。

DO機能	説明
動作状態	ロボットがプロジェクトを実行しているときには1を出力します。そうでない場合は0を出力します。
停止状態	ロボットが停止しているときには1を出力し、そうでない場合は0を出力します。
一時停止状態	ロボットが一時停止状態になると1を出力し、それ以外の場合は0を出力します。
安全原点状態	ロボットが安全原点にある場合は1を出力し、そうでない場合は0を出力します。
安全皮膚停止状態	ロボットがセーフティスキンでトリガされた一時停止状態になると1を出力し、そうでない場合は0を出力します。
アイドル状態	ロボットがアイドル状態 (イネーブル、停止中かつアラームなし) にある場合は1を出力し、それ以外の場合は0を出力します。
電源投入状態	ロボットの電源が入っている場合は1を出力します。そうでない場合は0を出力します。
イネーブル状態	ロボットをイネーブルするときに1を出力し、そうでない場合は0を出力します。
アラーム状態	ロボットにクリアされていないアラームがある場合は1を出力し、そうでなければ0を出力します。
衝突状態	ロボットが衝突を検出した場合は1、それ以外の場合は0を出力します。
ドラッグ状態	ロボットがドラッグ状態の場合は1を出力し、そうでない場合は0を出力します。
実行されていない時は低	プロジェクトまたはTCP命令シーケンスが実行されていない、一時停止または停止している場合は0を出力し、実行中は命令で対応するIO状態を設定できます。
未稼働時に高い	プロジェクトまたはTCP命令シーケンスが稼働していない、一時停止または停止しているときに1を出力し、稼働中に命令で対応するIO状態を設定できます。
異常停止時に低い	ロボットが異常停止したときに0を出力し、稼働中に命令で対応するIO状態を設定できます。 以下の状況が異常停止を引き起こします： • 安全機能停止（例：衝突検出、安全壁と安全領域、安全IO等） • ロボットのアラーム

	• プロジェクトの異常稼働

設定を変更したら、保存をクリックして設定を完了します。

⚠ 注意：

- 全てのリモートトリガー源が同時に有効になります。デバイスと生産の安全のため、ロボットは一つの制御源（制御ソフトウェア/DI/Modbus）からのみ起動するようにしてください。
- Modbusリモートコントロール信号はネットワークの影響を受けて遅延する可能性があります。現場での実測遅延を基に、遅延が許容範囲内かどうかを判断してください。
- Modbusのトリガーが有効になるかどうかは、手動/自動モードやIO/Modbusの設定にも影響を受けます。詳細は、対応する機能の関連説明を参照してください。
- ロボットの電源を投入して初期化する前に制御信号を送信しないでください。ロボットが異常動作するおそれがあります。

プロジェクトを選択

デフォルトで実行するプロジェクトを選択する場合、機能が開始であるコイルレジスタがトリガされると、この選択したプロジェクトを直接実行します。

選択をクリックすると、プロジェクト選択ボックスがポップアップ表示されます。

削除をクリックすると、現在選択されているプロジェクトがクリアされます。

保持レジスタ選択プロジェクトを選択した場合、複数のプロジェクトを選択できます。

+または-ボタンをクリックすると、設定するプロジェクト数を増減することができます（最大数99）。開始コイルレジスタがトリガーされた場合、保持レジスタの指定アドレス（3095）の値に基づいて起動するプロジェクトを決定します。

設定を変更したら、保存をクリックして設定を完了します。

8.6 グローバル変数

このページは、グローバル変数を設定するために使用します。

グローバル変数を設定した後、ブロックプログラミングではグローバル変数に関連するブロック呼び出しを使用できます。スクリプトプログラミングでは変数名を通じて直接呼び出すことができます。

追加ボタンをクリックしてグローバル変数を追加し、変数を選択して 変更ボタンをクリックして変数のプロパティを変更することができます。 削除ボタンをクリックして選択した変数を削除することができます。

次のタイプのグローバル変数がサポートされています。

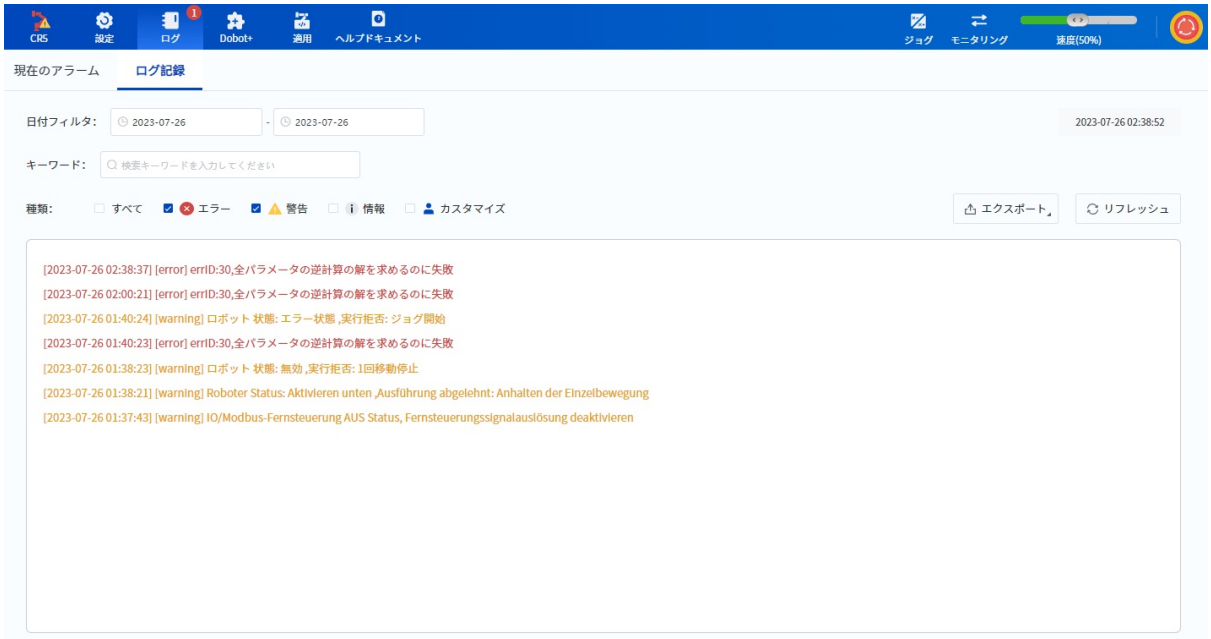
- number: 数値。値の範囲の設定をサポートしています。値の範囲を超える値は保存できません。値の範囲を超える値をプロジェクト内の数値変数に代入すると、ロボットがエラーを報

告して停止します。値の範囲を変更できるのは管理者のみです (この権限を割り当てることはできません)。
- bool：ブール値
- string：文字列
- table：テーブル（配列を含む）

グローバル保存：

- このオプションをチェックすると、変数の任意の変更が保存され、スクリプトを終了したり電源を再起動したりしても、変更後の値は保存されます。
- チェックしない場合、この変数の変更はスクリプトが実行されている間だけ有効で、スクリプトを終了すると初期設定値に戻ります。

9 ログ



ログ記録ページには、ロボットの動作ログが表示されます。日付、キーワード、ログタイプによるフィルタリングがサポートされています。

エクスポートログをパソコンのローカルパスまたは制御キャビネットのUSBインターフェースに接続されたストレージデバイスにエクスポートすることができます。

> 注意：
> USBポートに接続されたストレージデバイスが複数のパーティションを持っている場合、ログは最初のパーティションにエクスポートされます。一部のストレージデバイス（例えば、ブートディスクとして使用されるUSBメモリ）の最初のパーティションは隠しパーティションであり、Windowsではエクスポートされたログを直接見ることができない可能性があります。

一部のシーン（例えば、ロボットが自動で動作しているときに制御ソフトウェアがロボットに接続されている場合）では、ログは自動的に更新されず、手動でクリックする必要があります リフレッシュ最新のログレコードを取得します。

ログタイプの意味は次のとおりです。

- エラー：ソフトウェアが誤った動作をしたり、ロボットがアラーム状態になったりした場合のアラーム情報。
- 警告：ソフトウェアが異常な動作をしたり、ロボットが異常な状態になったりした場合の警告情報。異常状態は、後続のアクションの実行に影響しません。

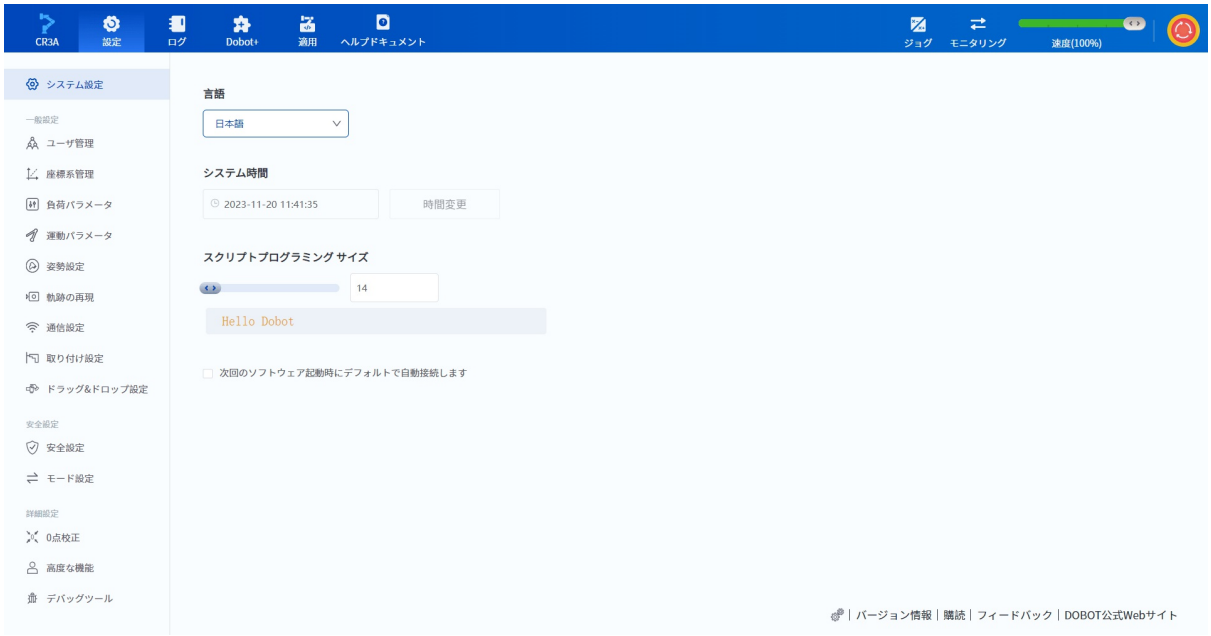
- メッセージ：ロボットの状態が変化したときに行うメッセージの記録。
- カスタム：ユーザが `Log(value)` 命令で出力したログ情報。

10 設定

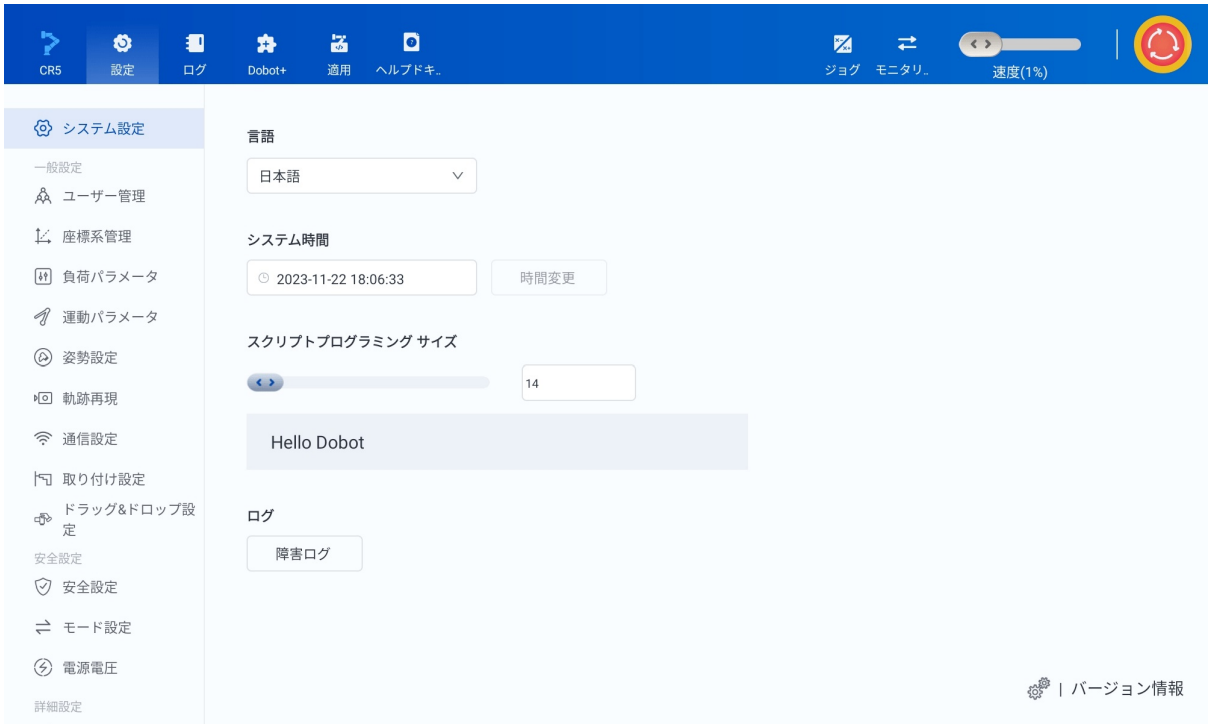
10.1 システム設定

システム設定ページはソフトウェア言語、管理者パスワード、システム時刻などを変更するために使用します。

PC側



モバイル側



言語：DobotStudio Proの表示言語を設定します。ロボットに接続していない状態でも変更可能です。

システム時刻：コントローラの現在のシステム時刻を表示します。デフォルトモードまたは手動モードでプロジェクトが実行されていないときに変更できます。現在のデバイスのシステム時刻がコントローラのシステム時刻と一致しない場合は、画面には「一致するように変更することをお勧めします」のメッセージが表示されます。

スクリプトプログラミングのフォントサイズ：スクリプトプログラミングのコードエリアのフォントサイズを設定します。デフォルトは14で、値の範囲は[12から50]です。

ログ：モバイル端末機能です。クラッシュログをアップロードします。インターネットへの接続が必要です。ロボットが接続されていない場合でも操作できます。

ソフトウェアが次に起動時にデフォルトで自動で接続：これをチェックした、次のソフトウェアの起動時に現在接続しているロボットアームの接続を自動で試みます。PC側のみ対応。

インターフェース右下角のアイコンおよび文字はクリック可能です。それらの機能は以下のとおりです。

 ：メーカー機能ページを開きます。付帯コードを入力する必要があります。本マニュアルでは説明は省略します。

バージョン情報：ソフトウェアのコンポーネントに関する説明を確認します。

以下の機能はPC側のみに対応します。

購読：ソフトウェア更新や製品ニュースに関する最新情報を購読します。

フィードバック：当社にソフトウェアの使用上の問題をフィードバックします。

DOBOT公式**WEB**サイト：クリックするとブラウザを開いてDOBOTオフィシャルサイトにアクセスすることができます。

10.2 ユーザー管理

ユーザーが管理者としてログインした場合、このページで役割の権限とパスワードを管理でき、カスタム役割の追加または削除がサポートされます。

デフォルトの役割を設定する

ページ左上にデフォルト役割（ロボットに接続するたびに自動的にログインする役割）を設定することができます。ただし、管理者としての設定はできません。デフォルト役割のパスワードが有効になっている場合、他の操作を実行する前に、ロボットに接続するたびにパスワードを入力するか、別の役割を選択してログインする必要があります。

カスタム役割を管理する

＋役割追加をクリックし、最大2つのカスタム役割を追加できます。

カスタム役割は、名前の変更や削除が可能ですが、その他の操作はデフォルト役割と同様ですので、以下をご参照ください。

役割パスワードを設定する

役割パスワードには複雑さや長さの要求はありません。20文字以内の英数字の組み合わせを使用できます。実際のニーズに応じて設定してください。

管理者のパスワードを有効にする必要があります。デフォルトのパスワードは**888888**で、変更することができますが。変更する場合は古いパスワードを入力する必要があります。

他の役割のパスワードはデフォルトで有効になっていません。パスワードを有効にするかどうか列のスイッチはデフォルトで**OFF**になっています。スイッチをクリックすると、パスワード設定ウィンドウが表示されます。設定後、パスワードが有効になり、スイッチが**ON**に変わります。

パスワードを設定したら、もう一度パスワード設定をクリックすると、同じパスワード設定画面が表示されます。このとき、古いパスワードでなく新しいパスワードを入力するだけで、該当役割のパスワードがリセットされます。

パスワードを有効にするかどうかスイッチをオンにすると、その役割に現在設定されているパスワードがクリアされ、次回オンにするときにパスワードをリセットする必要があります。

役割権限を設定する

権限の割り当てをクリックすると、次のページに進みます（カスタム役割を追加した場合は表示されます）。

すべての役割の権限は変更できます。デフォルト設定を復元をクリックすると権限の割り当て状態をデフォルト値に戻すことができます。

変更が完了したら、保存をクリックすると設定を有効にします。

デフォルトの役割権限

デフォルトの役割権限は次の表に示すとおりです。プロジェクトを開く/実行/停止機能の権限は変更できません。

種類	機能	管理者	技術者	オペレーター
基本操作	プロジェクトを開く/実行/停止	√	√	√
	イネーブル、アラーム解除、手動・自動モード切り替え、速度比例調整	√	√	√
	ロボットジョグ	√	√	√
	ログの確認・エクスポート	√	√	√
	オンライン/TCPモードの切り替え、IO/Modbus設定をオン・オフ	√	√	X
	IO（安全IOを含む）設定	√	√	X
	Modbus設定	√	√	X
	グローバル変数の設定	√	√	X
	DOBOT+プラグイン、エンドボタン設定	√	√	X
	プロジェクトとプログラム管理	√	√	X
	ポイントリスト操作	√	√	X

一般設定	システム時刻設定	√	X	X
	座標系管理	√	√	X
	負荷パラメータ設定	√	√	X
	キー設定	√	√	X
	軌跡の記録と再現	√	√	X
	通信設定	√	√	X
詳細設定	パッケージ・ゼロ点姿勢設定	√	X	X
	移動パラメータの設定	√	X	X
	設置角度設定	√	X	X
	ドラッグアンドドロップ設定	√	X	X
	安全設定	√	X	X
	手動・自動モード設定	√	X	X
	ゼロ点キャリブレーション	√	X	X
	高度機能設定	√	X	X

一括操作

一括操作 > 設定エクスポートをクリックすると現在のユーザー権限設定をファイルにエクスポートできます。一括操作 > 設定インポートをクリックするとユーザー権限設定をファイルからインポートできます。

10.3 座標系の管理

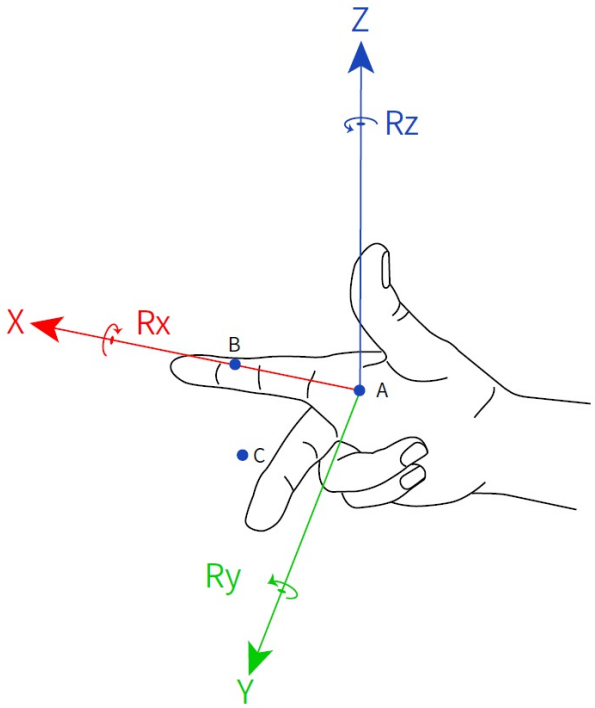
10.3.1 ユーザ座標系

ワークの位置が変化したとき、またはロボットの実行プログラムを複数の同じタイプの加工システム内で繰り返し使用する必要があるとき、ユーザ座標系を設定する必要があります。これにより、すべての経路がユーザ座標とともに同期更新されるので、ティーチングプログラミングを大幅に簡単化することができます。

現在、システムは最大50個のユーザ座標系に対応できます。ユーザ座標系**0**はベース座標系であり、変更することができません。

> ⚠注意：
> ユーザ座標系を確立するとき、点の位置を取得する際の基準座標系がユーザ座標系**0**であることを保証してください。

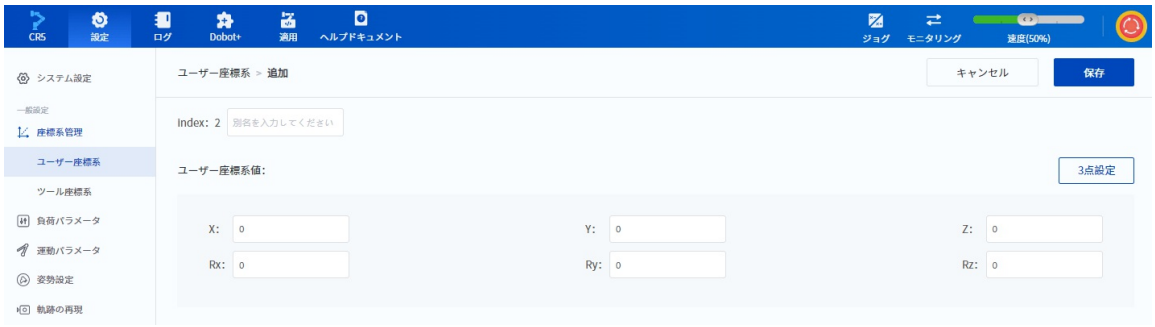
ユーザ座標系は3点ティーチング法で生成することをお勧めします。ロボットを任意の3つの点に移動させます。**A**，**B**および**C**。ここで、**A**点は原点とし、**AB**2点間の直線がユーザ座標系の**X**軸の正方向を決定し、**C**点が**X**軸に対して垂直線を作り、**Y**軸正方向を決定します。右手の法則に従って**Z**軸方向を決定します。



ユーザ座標系の作成

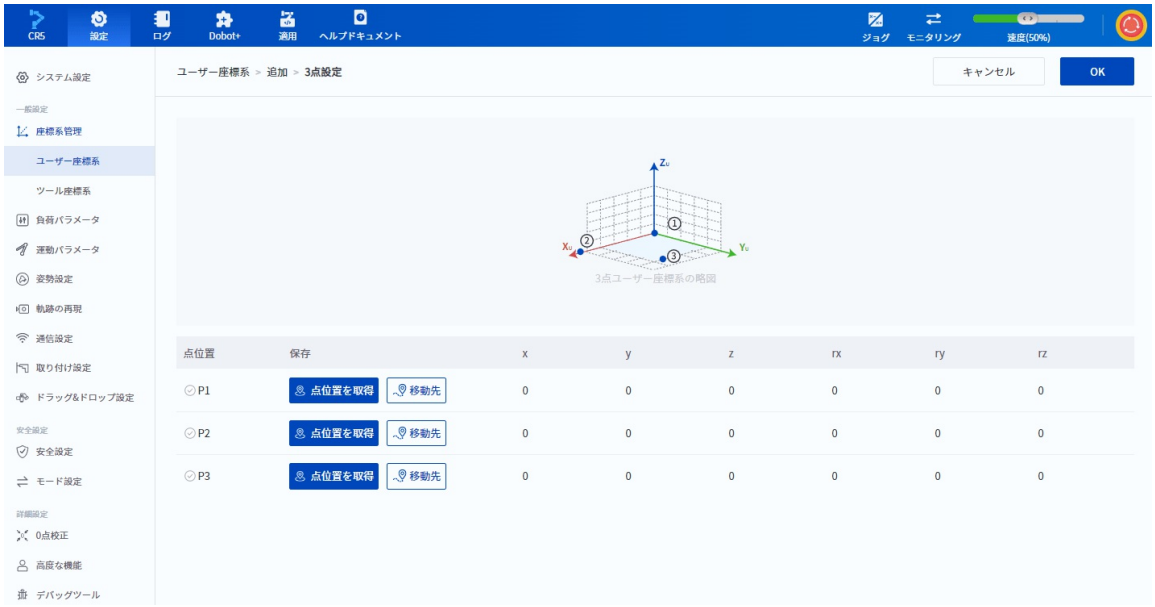
1. ユーザ座標系ページで＋追加をクリックします。下図のようになります。

2. ユーザ座標系追加ページで、座標系値の右上の**3**点設定。



> 説明：
> ユーザは直接X、Y、Z、Rx、Ry、Rzの値を変更し、保存をクリックすることもできます。X、Y、Zはユーザ座標系の原点がベース座標系中の位置を表し、Rx、Ry、Rzはユーザ座標系がベース座標系をX->Y->Zの順序で回転する角度を表します。

3. 概略図に示すように、ロボットを制御して対応する点に移動させた後、点位置を取得。



> 説明：

移動先を長押しすると、ロボットを取得した点位置に移動させることができます。

4. クリックして**OK**をクリックして、ユーザ座標系追加ページに戻ると、座標系値は校正した値に更新されます。該座標系を生成した3点設定を表示/変更するか、または座標系値を手動で変更することができます。詳細については次のユーザ座標系の変更手順を参照してください。

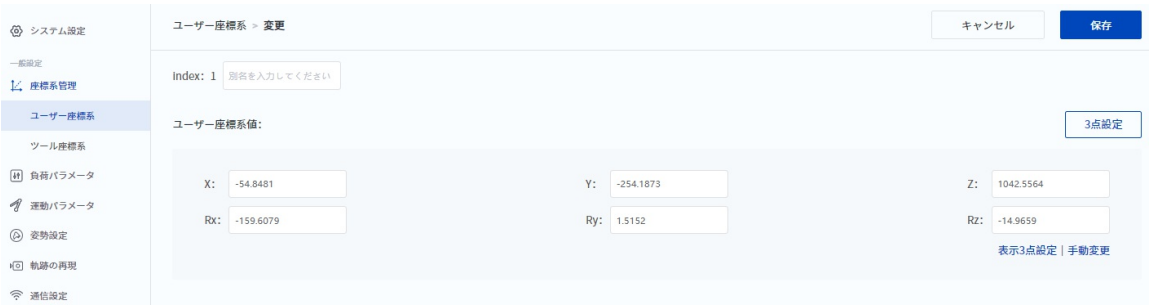
5. クリックして保存をクリックして、この座標系をユーザ座標系一覧に追加します。

ユーザ座標系の変更

1. ユーザ座標系ページで座標系を選択し、変更をクリックします。下図のようになります。

2. 選択したユーザ座標系の座標系値が、手動で入力し保存したものである場合、変更ページのUIは追加ページと同じであり、直接変更するか、または**3**点設定をクリックして校正することができます。

   選択したユーザ座標系の座標系値が、3点設定で生成したものである場合、変更ページのUIは下図に示すとおりです。



3. 座標系値の右下の**3**点設定を表示をクリックすると、該座標系値を生成した3点設定を表示し、変更が必要な点位置について再取得することができます。

4. クリックして変更をクリックすると、座標系値を直接変更することができます。

> 注意：
> 手動変更後、変更前の座標系値に対応する3点設定は再表示できなくなります。

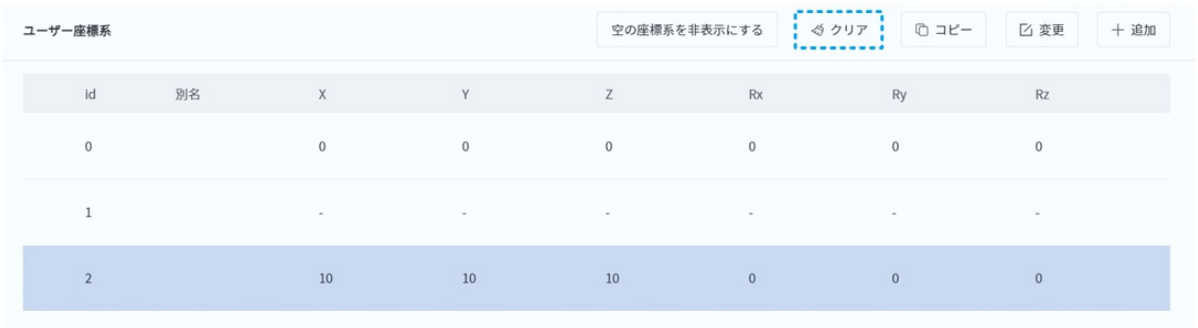
5. クリックして保存をクリックして、この座標系をユーザ座標系一覧で更新します。

ユーザ座標系のコピー

ユーザ座標系ページで座標系を選択し、コピーをクリックすると、選択した座標系と同じ新しい座標系を作成することができます。

ユーザー座標系をクリアする

「ユーザー座標系」ページで座標系を選択し、クリアと「OK」をクリックすれば、選択した座標系をクリアします。クリアされた座標系は引き続きそのIDを使用しているが、座標系データのみがクリアされます（例えば、下図の座標系2）。呼び出されるとエラーが発生します。



空の座標系を非表示にするをクリックした場合は、クリアされた座標系が座標系列リストに表示されないようにします。また、ボタンが空の座標系を表示に変わります。クリックすると、空の座標系の表示が復元されます。

空の座標系は変更でき、変更後に再割り当てされます。

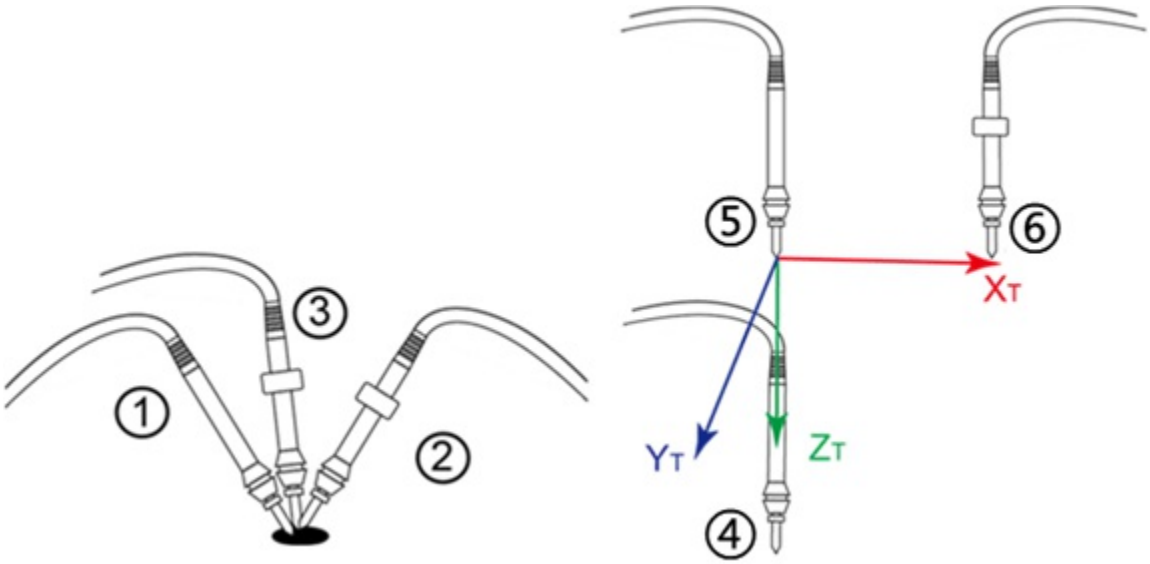
10.3.2 ツール座標系

ロボットのエンドにツール（例えば、溶接ガン、ノズル、グリッパなど）を取り付けた後、プログラミングおよびロボットの運転の必要上、この時点でツール座標系を設定する必要があります。例えば、複数のグリッパを使って複数のワークを同時に運搬する場合には、運搬効率が向上するように各グリッパを独立したツール座標系に設定することができます。

現在システムは最大50個のツール座標系に対応できます。ツール座標系**0**はフランジ座標系であり、ツールを使用することができず、変更することができません。

> ⚠ 注意：
> ツール座標系を確立するとき、点の取得時の基準座標系が工具座標系**0**であることを保証してください。

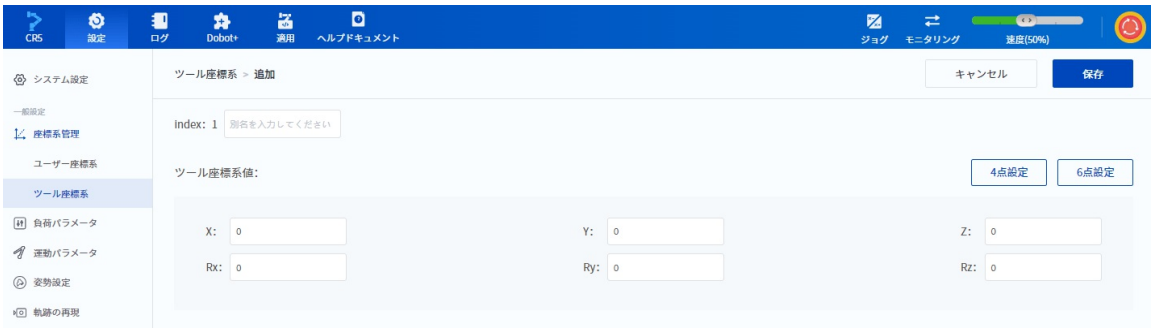
6軸ツール座標系は、6点ティーチング法「TCP+ZX」で生成することをお勧めします。ロボットのエンドにツールを取り付けた後、ツールの位置姿勢を調整して、TCP（Tool Center Point）を3つの異なる方向（①②③）で空間中の1つの点に合わせて、ツールの位置オフセットを取得します。さらに他の3つの点（④⑤⑥）によってツールの姿勢オフセットを取得します。ここで、④は、①②③と同じ点に合わせます。



ツール座標系の作成

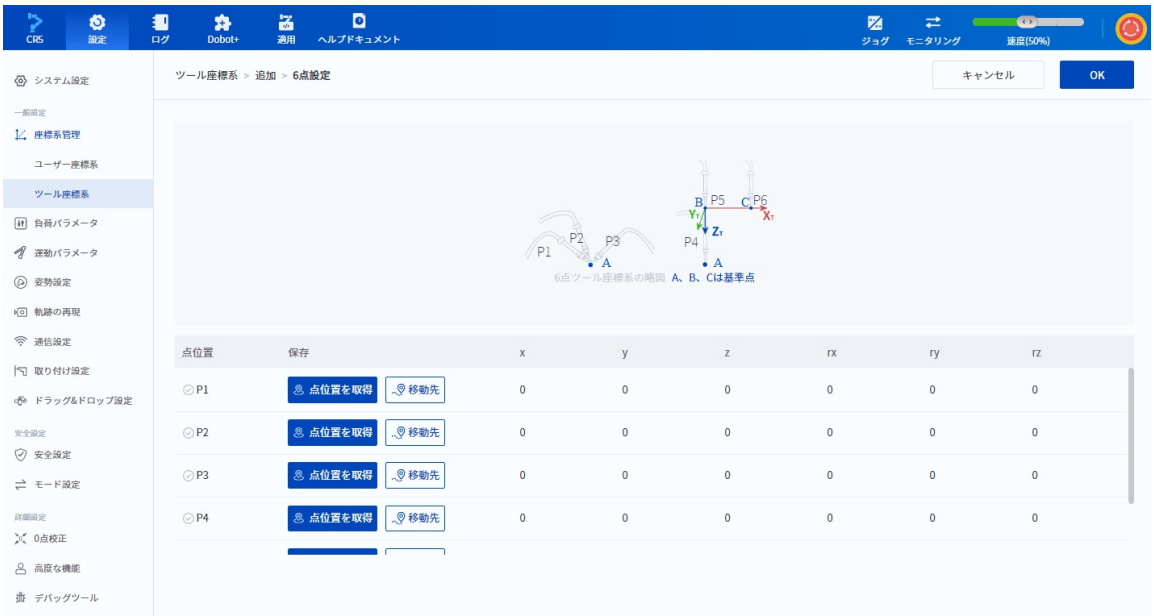
1. ツール座標系ページで＋追加をクリックします。下図のようになります。

2. ツール座標系追加ページで座標系値の右上の**6**点設定。



> 説明：
> ユーザは直接X、Y、Z、Rx、Ry、Rzの値を変更し、保存をクリックすることもできます。
> **4**点設定は**6**点設定と操作方法が似ており、本文章ではその説明は省略します。

3. 概略図に示すように、ロボットを制御して対応する点に移動させた後、点位置を取得。



> 説明：

移動先を長押しすると、ロボットを取得した点位置に移動させることができます。

4. クリックして**OK**、ツール座標系追加ページに戻ると、座標系値は校正した値に更新されます。該座標系を生成した3点設定を表示/変更するか、または座標系値を手動で変更することができます。詳細については次のツール座標系の変更手順を参照してください。

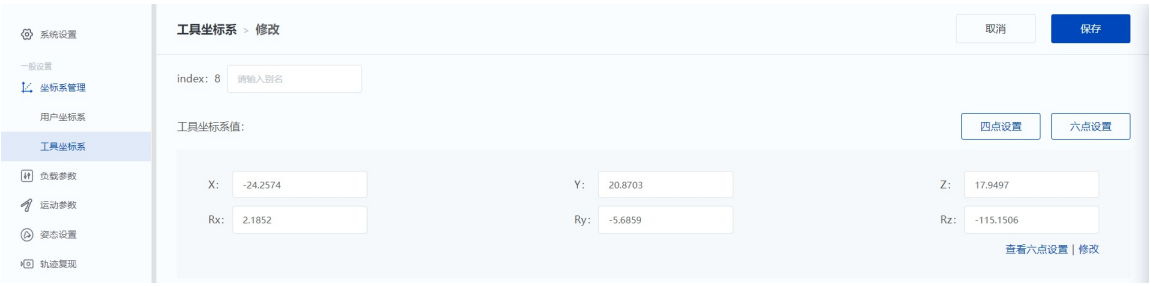
5. クリックして保存、この座標系をツール座標系一覧に追加します。

ツール座標系の変更

1. ツール座標系ページで座標系を選択してから  をクリックします。変更をクリックします。下図のようになります。

2. 選択したツール座標系の座標系値が、手動で入力し保存したものである場合、変更ページのUIは追加ページと同じであり、直接変更するか、または**4**点設定/**6**点設定をクリックして校正することができます。

   選択したユーザ座標系の座標系値が、4/6点設定で生成したものである場合、変更ページのUIは下図に示すとおりです。**6**点設定を表示は生成方法により、文字が変化します。



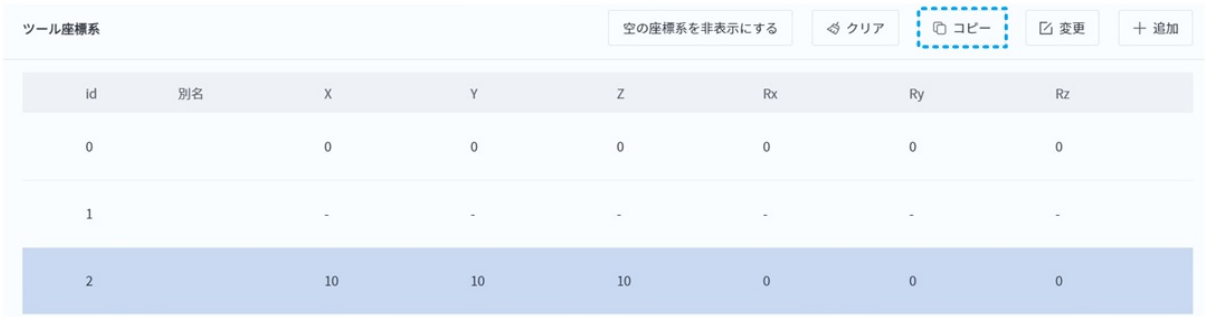
3. クリックして**6**点設定を表示をクリックすると、該座標系値を生成した6点設定を表示し、変更が必要な点位置について再取得することができます。4点設定も同様です。

4. クリックして変更をクリックすると、座標系値を直接変更することができます。手動変更後、変更前の座標系値に対応する6/4点設定は再表示できなくなりますので、注意してください。

5. クリックして保存をクリックすると、ツール座標系一覧でこの座標系を更新することができます。

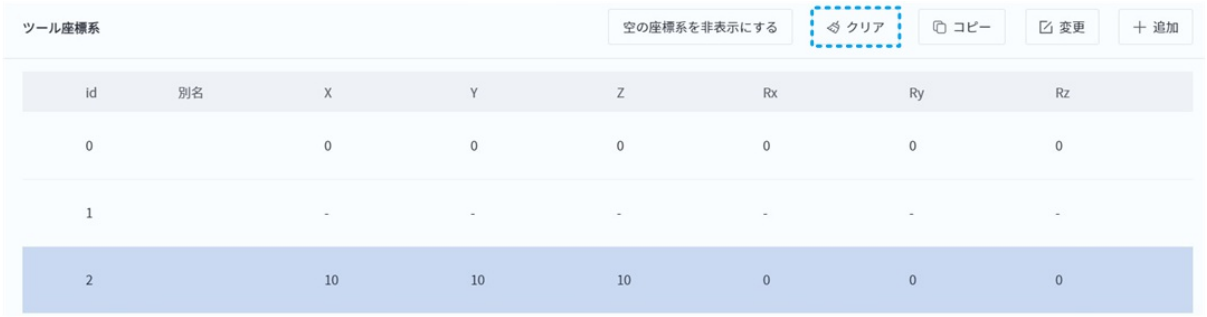
ツール座標系のコピー

ツール座標系ページで座標系を選択してから をクリックします。コピーをクリックすると、選択した座標系と同じ新しい座標系を作成することができます。

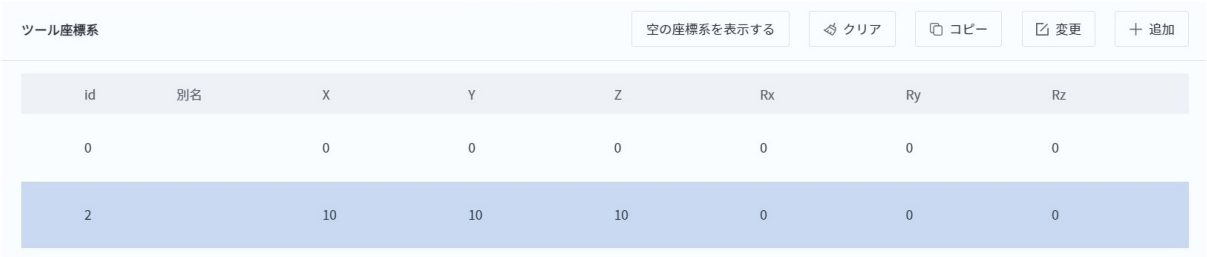


ツール座標系をクリアする

「ツール座標系」ページで、座標系を選択し、 クリアと「OK」をクリックすれば、選択した座標系をクリアします。クリアされた座標系は引き続きそのIDを使用していますが、座標系データのみがクリアされ（例えば、下図の座標系3）ます。呼び出されるとエラーが発生します。



空の座標系を非表示にするにチェックを入れた場合、クリアされた座標系が座標系列リストに表示されないようにします。また、ボタンが空の座標系を表示に変わります。クリックすると、空の座標系の表示が復元されます。

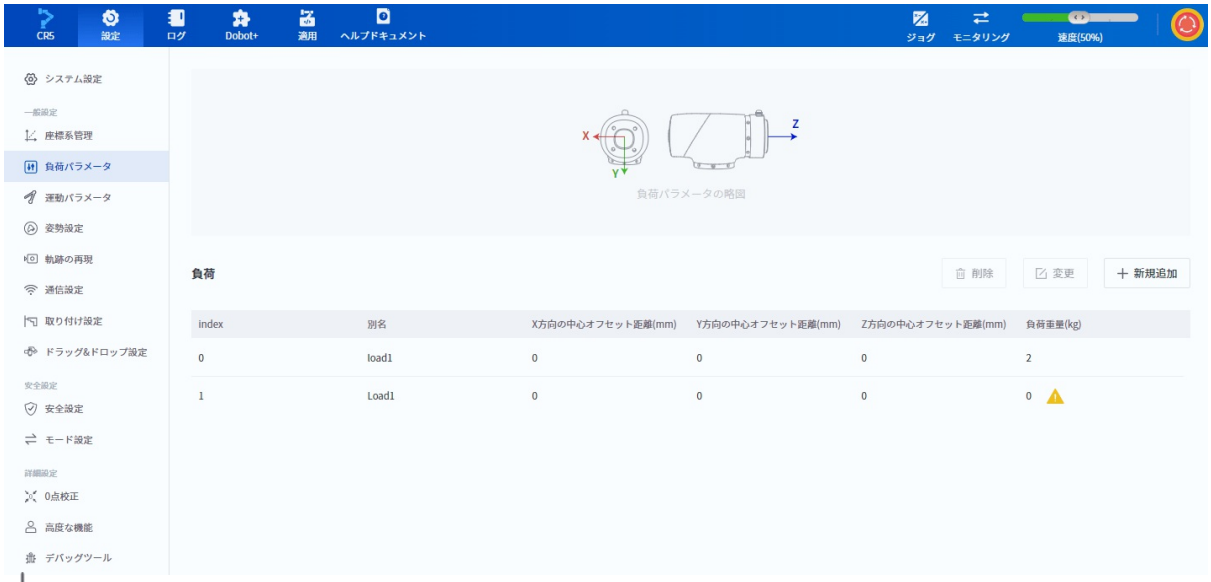


空の座標系は変更でき、変更後に再割り当てされます。

10.4 負荷パラメータ

負荷パラメータは、ロボットのエンドの負荷（治具含む）の重心と重量のパラメータです。実際の負荷に基づいて設定してください。設定が正しくない場合、ロボットの性能が低下し、衝突検出の誤動作やドラッグが制御不能になる可能性があります。

> 説明：
> このインターフェースは負荷パラメータグループの変更のみに使用され、変更後はすぐには有効になりません。変更後のパラメータを有効にするには、ロボットを有効にするときに表示される負荷設定ページで対応するパラメータグループを選択してください。



＋新規をクリックすると、一連の負荷パラメータが新規に追加されます。パラメータのセットをクリックして選択した後、変更をクリックして、選択したパラメータグループを変更し、削除をクリックして、選択したパラメータグループを削除します。

パラメータグループの追加/変更

パラメーターグループの別名は変更可能で、イネーブルとプログラミング時にパラメーターグループを参照するために使用します。

負荷パラメータの設定には、自動認識と手動変更の2つの方法があります。

自動認識

ロボットが有効であり、アラームがなく、動いていない場合、ロボットの現在の負荷のパラメーターが自動的に認識されます。

1. 自動認識をクリックすると、自動認識ウィンドウを開きます。
2. システムは7つのデフォルト点位置を自動的に生成します。点位置を選択すると、その姿勢図が表示されます。へ移動を長押しするとロボットをその点位置に移動させることができます。
3. 障害物などによって、ロボットがデフォルトの点位置に移動できない場合、その点位置図を参照して、条件を満たす別の点位置にロボットをジョグさせることができます。その後、それをデフォルトの点位置に上書きします。デフォルト点位置を復元をクリックすると、上書きされた点位置をデフォルトの点位置に戻すことができます。
4. ワンクリックで自動認識をクリックして確認すれば、ロボットアームが各点位置まで順次移動して、認識します。
   - 認識が成功すると、ソフトウェアは負荷追加/変更画面に戻り、負荷パラメータが識別されたパラメータに更新されます。実際の負荷の状況に基づいて、これらのパラメータが妥当であるかどうかを確認し、妥当でない場合は、負荷認識をやり直すか、パラメータを手動で変更してください。
   - 識別に失敗した場合、エラーメッセージがポップアップ表示され、自動認識ページに残ります。各点位置図を参照して、その点位置が条件を満たしているか確認してください。条件を満たしていない点位置を変更してからもう一度ワンクリックで自動認識を実行してください。

手動変更

次の負荷パラメータを手動で入力します。

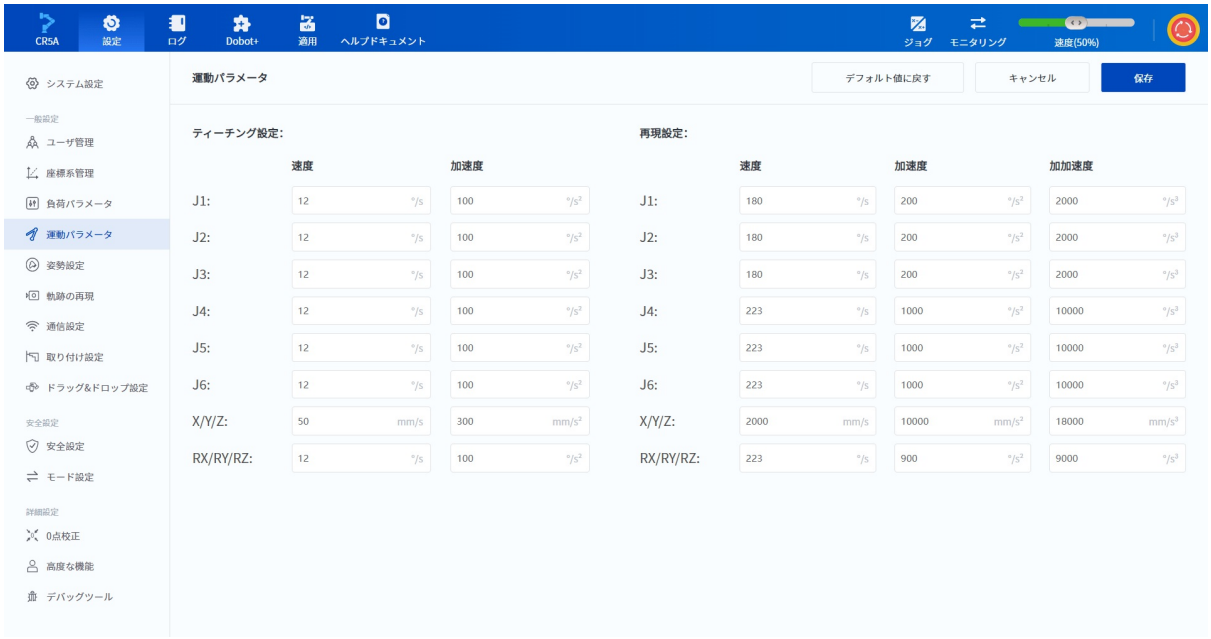
- **X/Y/Z**方向の中心オフセット距離：エンド負荷の重心が各方向にどれだけの距離をオフセットしているか、各座標軸の方向は画面上の図をご参照ください。
- エンドロード：エンド治具の重量と工作物の重量の合計がロボットの最大許容負荷を超えることはできません。

設定を変更した後、保存ボタンを押して設定を保存します。

⚠注意：
負荷の設定が正しくないと、衝突検出異常警告が発生する可能性があります、またはドラッグ時に本体が制御不能になる可能性があります。

10.5 移動パラメータ

ロボットは出荷時に一般的な作業条件に最適な移動パラメータが設定されています。特別な要求がない限り、変更は推奨されません。ユーザーがロボットの速度が高すぎると感じた場合は、実際のニーズに合わせて速度を下げることができます。高速化について、テクニカルサポートまでお問い合わせください。

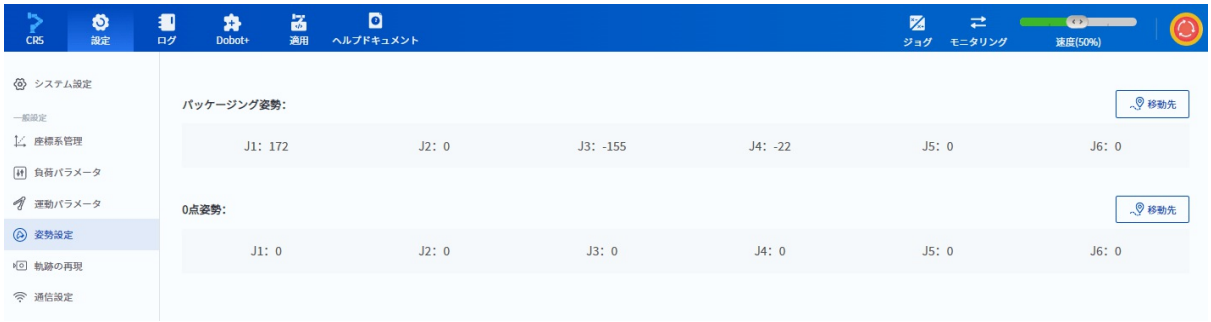


ロボットの速度はジョグ速度と再現速度があります。ジョグはジョグパネルによるジョグ・インチング、**TCP**ジョグ命令と へ移動する機能を含みます、再現はプロジェクト実行と**TCP**移動命令を含みます。

ジョグには各関節またはデカルト座標系の最大速度と最大加速度を設定することができます。再現移動には最大加速度を設定することができます。

インターフェースの右上隅のデフォルトに戻すをクリックすると、移動パラメータを出荷時パラメータにリセットできます。編集をクリックしてパラメータを変更できます。

10.6 姿勢設定

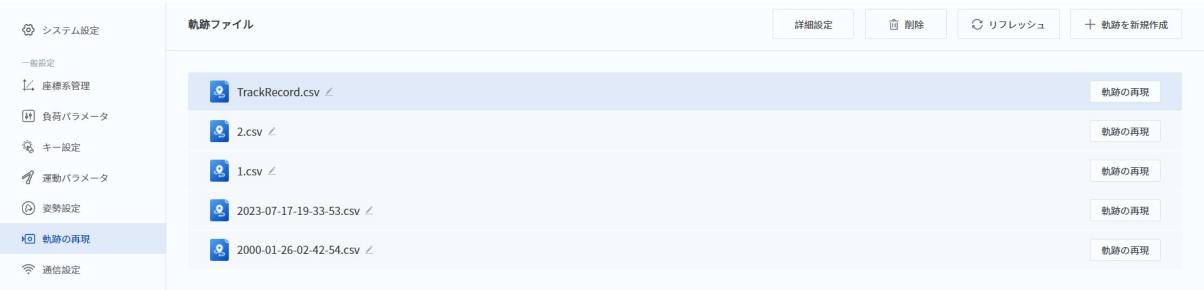


- パッケージング姿勢ロボットが占用する空間を小さくし、梱包搬送を容易にすることができます。
- **0**点姿勢各関節の角度はすべて0度です。

を長押ししますへ移動するロボットを対応する姿勢に移動させることができます。

10.7 軌跡再現

軌跡再現は、ロボットの移動軌跡の記録および再現に使用します。



右上角の＋軌跡を新規作成をクリックすると、ロボットはドラッグモードに入ります。このときユーザがロボットをドラッグすると、ドラッグ軌跡は記録されます。

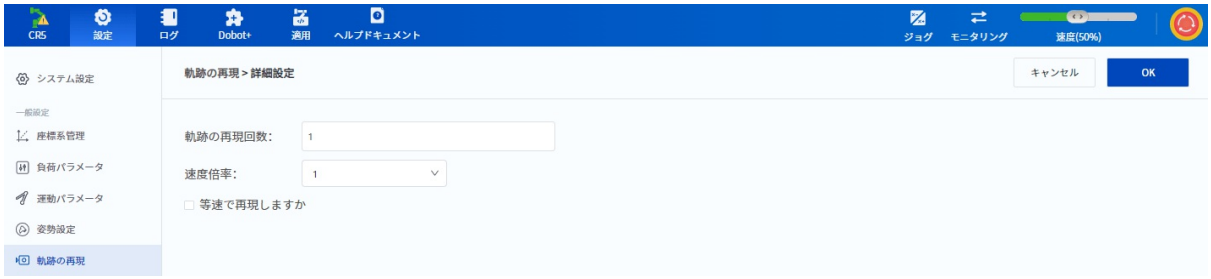
記録対象の軌跡が終了したら、保存をクリックすると、ロボットはドラッグモードを終了し、軌跡ファイル一覧に1つの記録が新規追加されます。ユーザはロボットのエンドのボタンを使って軌跡を記録することもできます（詳細は対応するハードウェアマニュアルをご参照ください）。エンドのボタンを押して記録を開始した軌跡は一つだけ保存され（ファイル名はTrackRecord.csv）、新たに記録した軌跡は以前の軌跡を上書きします。

- 軌跡ファイル名の隣のをクリックすると、軌跡ファイル名を変更することができます。
- 軌跡を再現をクリックすると、ロボットに記録した軌跡を再現させることができます。再現中にいつでも停止することができます。
- 軌跡ファイルを選択した後、削除をクリックするとこの軌跡を削除することができます。
- リフレッシュをクリックすると、コントローラの最新の軌跡ファイル一覧を表示することができます。

> 説明：
> 保存済みの軌跡ファイルは、グラフィカルプログラミングおよびスクリプトプログラミング中に軌跡再現に関連する命令で呼び出すこともできます。

詳細設定をクリックすると、軌跡再現方法を設定することができます。

- 軌跡再現回数は、軌跡を再現をクリックした後に行われる軌跡再現の回数を表しロボットます。回数が1より大きいとき、ロボットは1回の軌跡再現が完了した後に、自動関節移動で軌跡開始点に戻って、次の再現を開始します。
- 速度倍率は、等速で再現しますかチェックボックスが選択されていないときにのみ有効です。ロボットは、比例スケーリングされた原速度（グローバル速度から影響を受けない）で軌跡を再現します。
- 等速で再現しますかチェックボックスを選択すると、ロボットはグローバル速度で等速軌跡を再現します。



10.8 通信設定

IP設定

ロボットはLANインターフェースを介して外部デバイスと通信します。TCP、UDP、またはModbusプロトコルに対応しています。ユーザーはロボットのLAN1ポートのIPアドレス、サブネットマスク、ゲートウェイを変更することができます。外部デバイスを接続する場合は、IPアドレスが外部デバイスのIPアドレスと同じネットワークセグメントにあり、競合していないようにしなければなりません。

- ロボットが外部デバイスに直接接続されているか、スイッチを介して接続されている場合、ロボットと外部デバイスが同じネットワークセグメントにあるように、手動を選択して、IPアドレス、サブネットマスク（複数のネットワークセグメントを組み合わせたネットワークに接続する場合）およびデフォルトゲートウェイを変更します。
- ロボットがルーター経由で外部デバイスに接続されている場合、自動取得（IPアドレスはルーターによって自動的に割り当てられる）を選択します。

バス通信

バス通信機能にはオフ、プロフィネットまたはイーサネット/IPを設定することができます。

プロフィネットまたはイーサネット/IPを設定したら、バス通信が失われたときのロボットの動きを設定する必要があります。

- なし：バス通信が失われた場合、処理は実行されず、プロジェクトは継続して実行されます。
- 一時停止：バス通信が失われた場合、実行中のプロジェクトを一時停止します。
- 停止：バス通信が失われた場合、実行中のプロジェクトを停止します。

バス通信機能の使用方法については、「Dobotバス通信プロトコルドキュメント（EtherNet/IP、Profinet）」をご参照ください。

⚠️注意：
バス通信をProfinetに設定したら、ロボットのLAN1ポートはProfinet通信にのみ使用でき、IPは設定できません。LAN1を他の通信に使用する必要がある場合は、バス通信設定を再編集して保存してから、制御盤を再起動してください。

WiFi設定

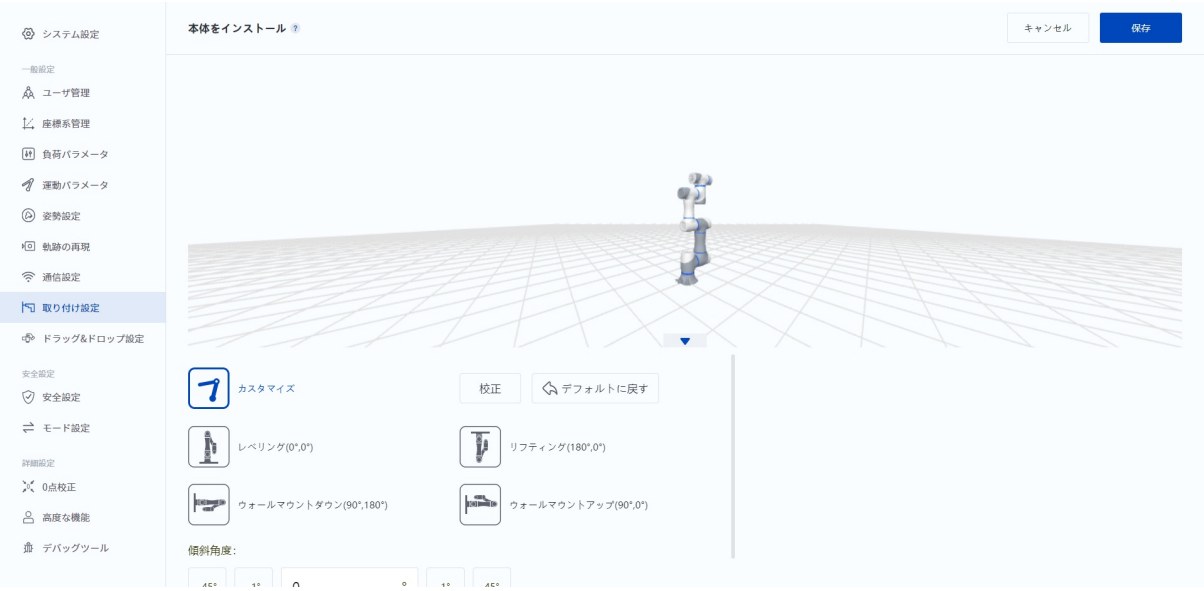
ロボットはWiFiを介して外部デバイスと通信できます。ユーザーはWiFiを有効または無効にしたり、WiFiの名前とパスワードを変更したりできます。現在のパスワードを確認するには、  をクリックします。

Magician E6のWiFiモジュールはユーザーが別途で購入して取り付ける必要があります。このページでオン/オフを切り替えることができません。

⚠️注意：
WiFi設定を変更すると、ソフトウェアとロボットの接続が切断される場合があります。切断された場合は、5秒後に再接続してください。

10.9 インストール設定

通常、ロボットアームは平らな作業台または地上に設置されています。このページでは操作を行う必要はありません。ロボットアームが天井または壁に取り付けられている場合、または角度を付けて取り付けられている場合は、ディセーブル状態で回転角度と傾斜角度を設定する必要があります。



ユーザーは手動および自動のいずれかでキャリブレーションを行うことができます。

手動キャリブレーション

実際の設置姿勢により、ページ左側の該当する設置の姿勢を選択するか、カスタマイズを選択して、下の傾斜角度と回転角度を調整します。各状態の詳細な説明は、ページタイトルの右側の ? をクリックして確認できます。

- 傾斜角度とは原点位置における本体のX軸回りの反時計回りの回転角度をいいます。
- 回転角度とは原点位置における本体のZ軸回りの反時計回りの回転角度をいいます。

自動キャリブレーション

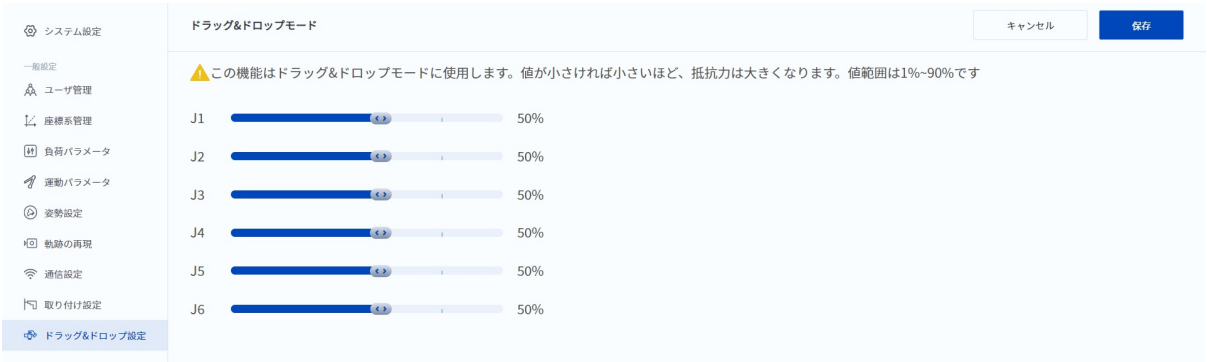
ロボットアームを設置して有効にしたら、キャリブレーションをクリックします。また、ポップアップダイアログボックスに従って操作し、傾斜角度と回転角度を取得します。

デフォルトに戻すをクリックすると、キャリブレーションされた角度はデフォルト値に戻ります。

設置角度を設定した後、ロボットアームのエンドボタンを押してドラッグモードに入り、ドラッグ機能が正常に動作するかどうかを確認してください。異常が発生した場合は、設置角度を再設定するか、テクニカルサポートにご連絡ください。

10.10 ドラッグアンドドロップ設定

ドラッグ設定は主に、ロボットの各関節のドラッグ操作時の感度を調節するために使用されます。感度が低いほど、ドラッグ中の抵抗が大きくなります。

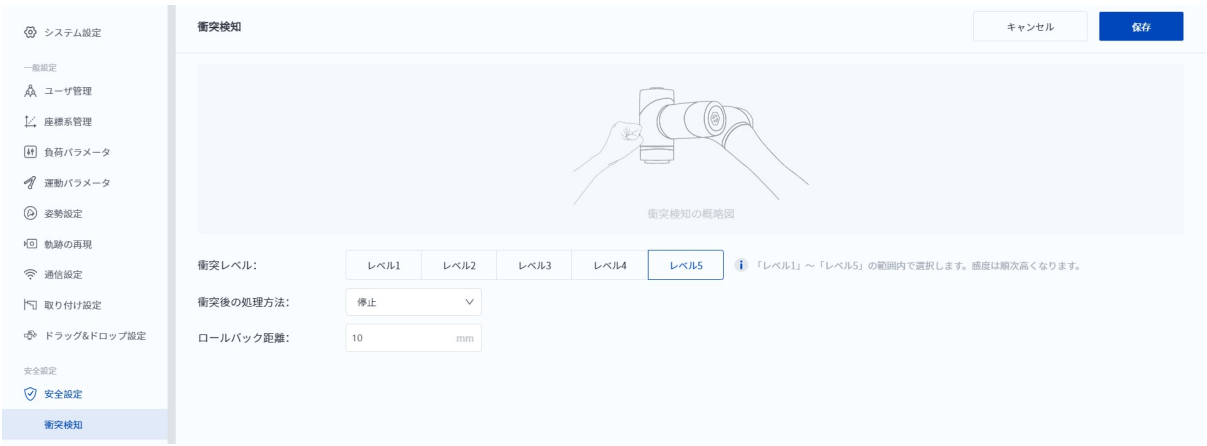


10.11 安全設定

10.11.1　衝突検出

ロボットの移動中に衝突を検知すると自動的に停止します。ユーザーは衝突検出の感度や衝突後の具体的な処理方法を設定できます。

> 説明：
> - このページでは衝突検出機能のオン/オフはサポートされていません。衝突レベルが最低でも誤報が頻繁に発生し、衝突検出をオフにしたい場合は、技術サポートにご連絡ください。
> - 衝突検出機能がオフのとき、このページの設定は無効になります。衝突検出を再度有効にするには、技術サポートにご連絡ください。



実際の要求に応じて、衝突レベルを高く設定するほど、衝突検出をトリガするのに必要な力は小さくなります。

ジョグと自動実行時、ロボットが衝突した後の処理方式に違いがあります。

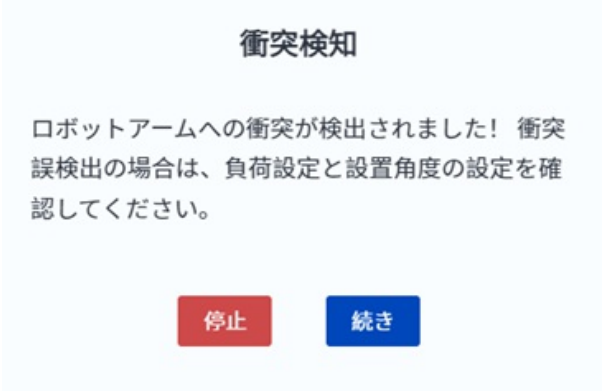
ジョグ中の衝突事故

ソフトウェアのポップアップウィンドウに、衝突が検出されたことが表示される場合は、衝突の原因を取り除く必要があります。リセットをクリックしてください。衝突の原因を取り除くには制御ソフトウェアを操作する場合、**1**分後に再通知するをクリックしてください。ポップアップウィンドウを一時的に閉じてもよい（ポップアップウィンドウは1分後に再び表示されます）。

自動実行中の衝突

ロボットは衝突後の処理方法に従って処理します。

- 停止：ロボットの動作が停止します。
- 一時停止：ロボットが一時停止し、図のようなポップアップウィンドウが表示されます。この場合、実際の状況に応じて「衝突の原因を取り除く」を選択したら、続行するか停止します。ロボットが衝突により一時停止状態のとき、ロボットのエンドのドラッグ教示ボタンを短く押すことで運転を続行することもできます。



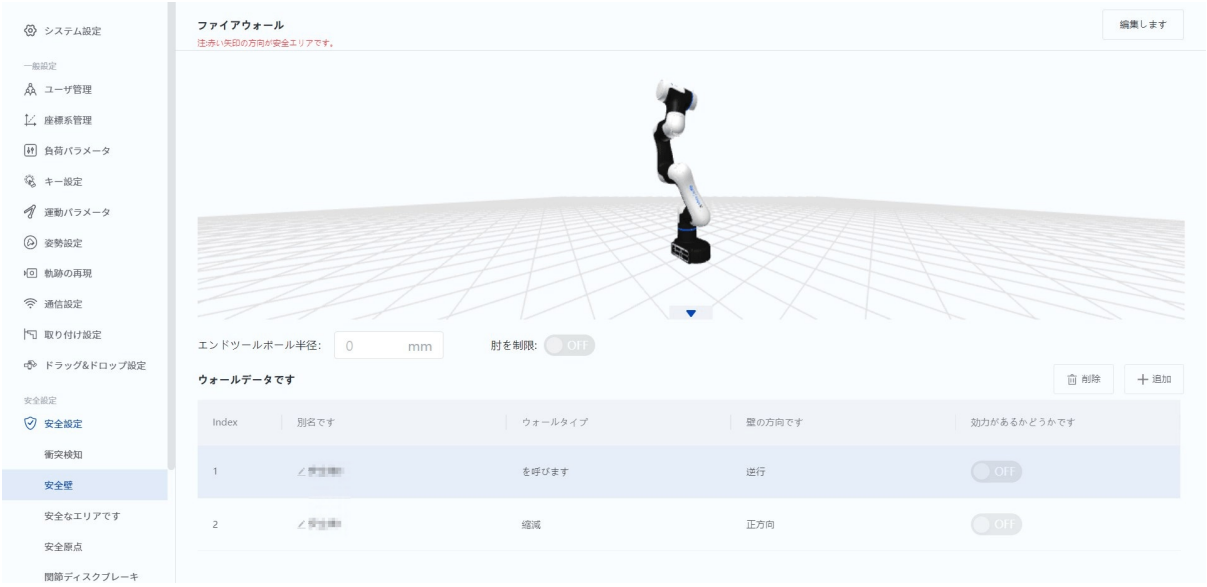
いずれの処理方式であっても、ロボットは衝突後、衝突前の軌跡に基づいて指定した距離だけ自動的に後退します。後退距離の設定範囲は0〜50mmで、デフォルト値は10mmです。

10.11.2 セーフウォール

DobotStudio Proでは最大8つのセーフウォールを設定することができ、ウォールの方向によりロボットの安全空間を指定します。ロボットエンドツールボール（TCPを中心とするカスタム半径の球状空間）またはJ3関節（オプション）がセーフウォールで定められた安全空間に近づくか超えると、壁のタイプによって異なるアクションがトリガーされます。

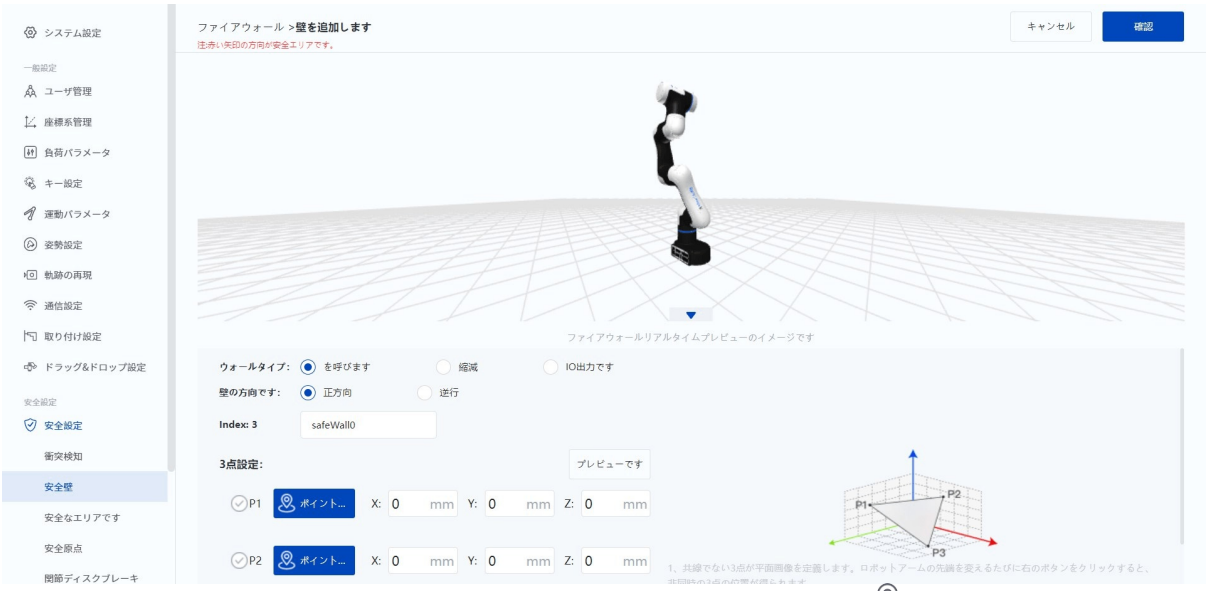
- アラーム：安全境界に近づいたとき、現在の動作計画によりロボットが安全範囲を超えた場合、アラームがトリガーされ、動作が停止します。
- 縮小：安全空間を超えると、ロボットは減速モードをどリガーし、速度が低下します。
- **IO**出力：安全領域を超えると、ロボットは指定されたDOをトリガーし、動作には影響しません。

異なるセーフウォールまたは安全領域は同時に有効であり、同じDOを設定することができます。有効なセーフウォールまたは安全領域をトリガーすると、そのアクションがトリガーされます。



セーフウォールの追加

＋新規をクリックするとセーフウォールを追加することができます。セーフウォールはユーザー定義の平面であり、該平面上に非共線の3つの点で決められます。ユーザーはまず以下P1、P2とP3と呼ぶ3つの点を決める必要があります。



1. ロボットアームをP1にジョギングさせるか、またはドラッグし、点位置を取得をクリックして、P1の座標を取得します。
2. 同じ方法でP2とP3の座標を取得し、プレビューをクリックすると上のシミュレーション領域で、生成されたセーフウォールを確認できます。

> 説明:
> - ユーザーは、点のデカルト座標を手動入力または変更することもできます。
> - 設定を変更したらプレビューをクリックするとシミュレーション領域の表示が更新されます。

3. ウォールのタイプを設定します。
4. ウォール方向の設定右側シミュレーションエリアでウォール方向を表示できます。矢印の方向がウォールの安全な側で、反対方向が制限側です。

> 説明:
> ウォールの正方向は、P1 -> P2 ->P3のベクトル方向に基づいて、右手の法則に従って算出した平面法線ベクトルとします。

5. **OK**をクリックすると、該セーフウォールを追加することができます。

セーフウォールの変更

セーフウォールは**Index**以外のプロパティが変更できます。信号出力はIO出力ウォールのみを設定できます。右側のスイッチは、セーフウォールを有効にするかどうかを制御します（ロボットの無効状態でのみ操作可能）。有効なセーフウォールがロボットアームと干渉し、右側のシミュレーション領域に表示されます。

セーフウォールを選択してから**X** 削除をクリックすると、選択したセーフウォールを削除することができます。

詳細設定

エンドツールボール半径

エンドツールボール（TCPを球心とするカスタム半径の球形空間）の半径を指定します。該球形空間がセーフウォール制限エリアと干渉すると、セーフウォールの動作をトリガーします。0に設定すると、TCPのみが制限エリアと干渉し得ることを表します。

肘部を制限

このオプションのチェックボックスを選択することは、J3関節も制限エリアと干渉し得ることをを表します。チェックボックスを選択していないとき、エンドツールボールのみが制限エリアと干渉することがあります。

復元モード

ロボットが非安全領域にある場合、ロボットを有効にすると、下の図に示すように、ポップアップウィンドウが表示され、「復元モードに入りました」をユーザーに通知します。

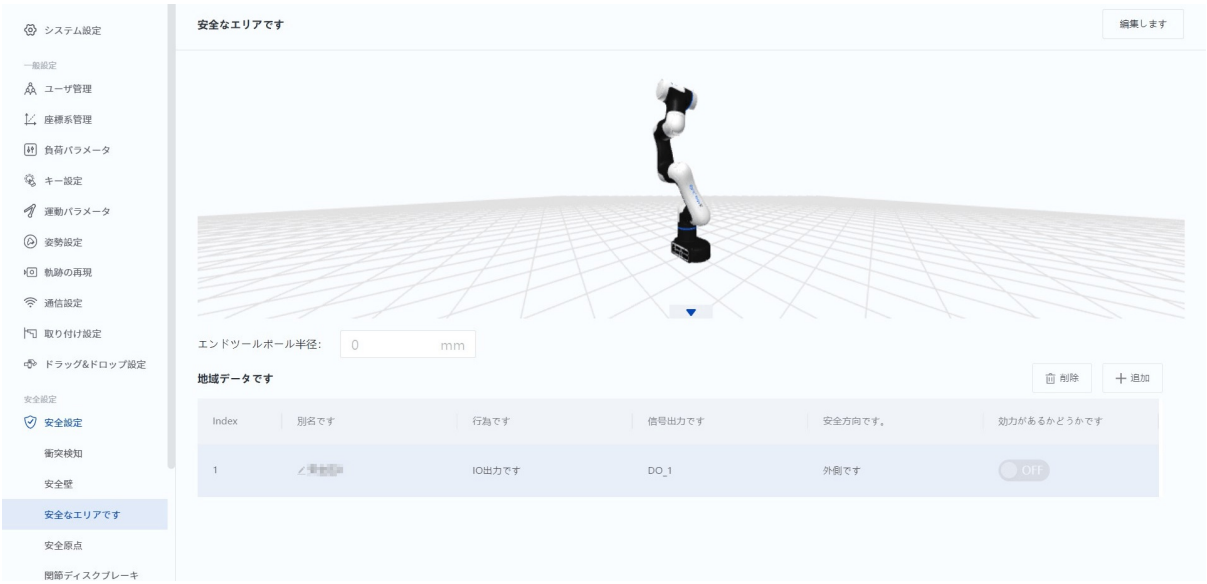
復元モードでは、ユーザーがロボットの動きを制御するにはジョグまたはドラッグ＆ドロップだけを使用できます。ロボットが安全領域に入ると、自動的に復元モードが終了し、ポップアップウィンドウが表示されます。

10.11.3 安全領域

DobotStudio Proは最大6つの安全領域を設定することができます。指定された領域の内側または外側はロボットの安全領域となります。ロボットエンドツールボール（TCPを中心とするカスタム半径の球状空間）が安全領域に近づくか超えると、安全領域の動作タイプによって異なるアクションがトリガーされます。

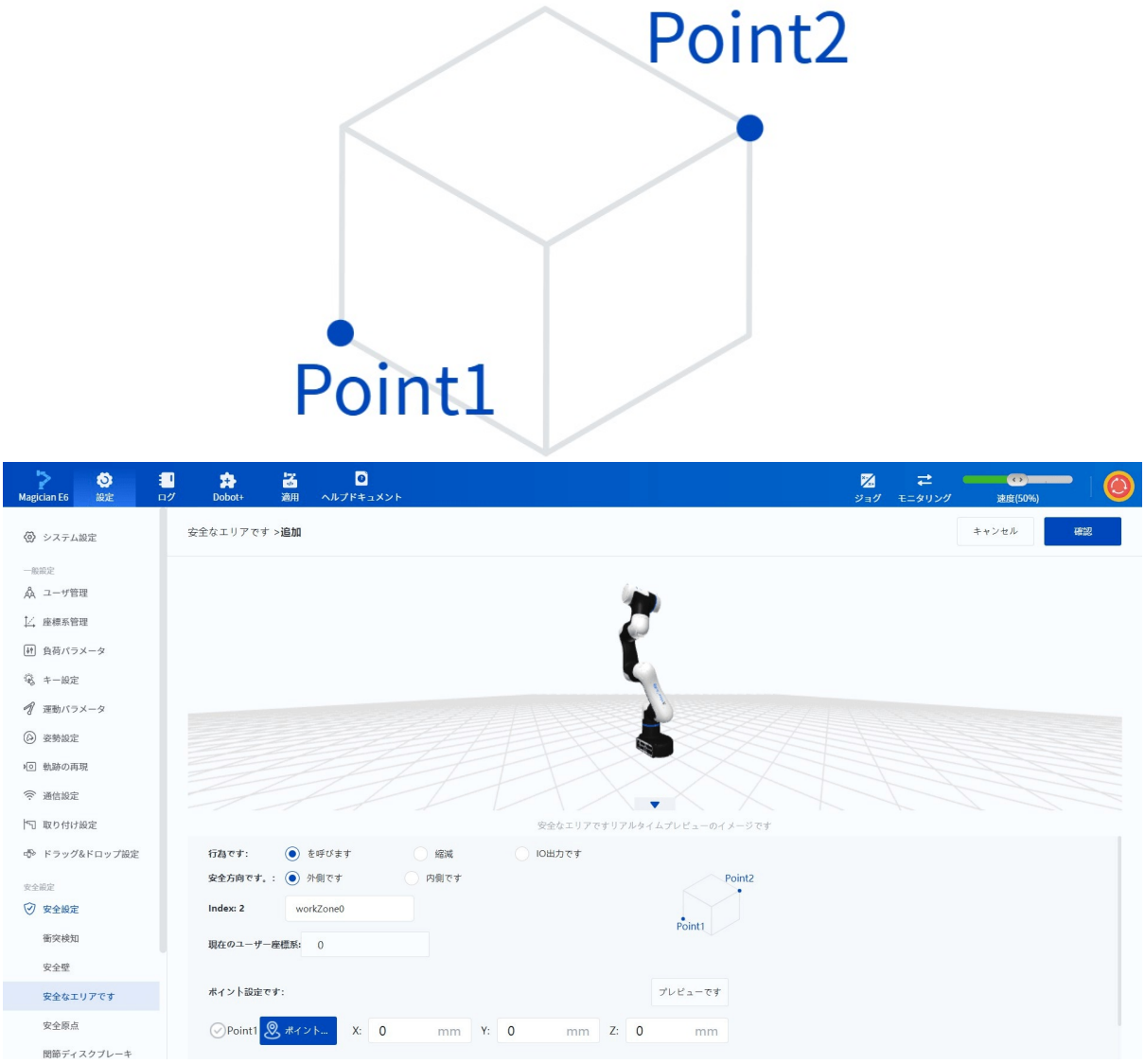
- アラーム：安全境界に近づいたとき、現在の動作計画によりロボットが安全範囲を超えた場合、アラームがトリガーされ、動作が停止します。
- 縮小：安全空間を超えると、ロボットは減速モードをどリガーし、速度が低下します。
- **IO**出力：安全領域を超えると、ロボットは指定されたDOをトリガーし、動作には影響しません。

異なる安全領域またはセーフウォールは同時に有効であり、同じDOを設定することができます。有効な安全領域またはセーフウォールをトリガーすると、そのアクションがトリガーされます。



安全領域を追加する

＋追加をクリックすると安全領域を追加することができます。安全領域は立方体です。ユーザーは安全領域の対角の2頂点（Point1とPoint2）を教示し、次の図に示すようにユーザー座標系を指定して立方体の側面の方向を決定する必要があります。



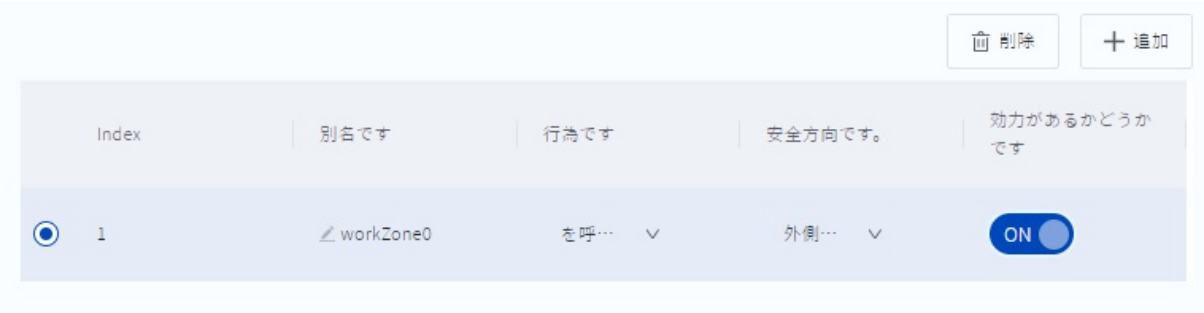
1. ロボットをPoint1にジョギングさせるか、またはドラッグし、点位置を取得をクリックして、Point1の座標を取得します。
2. 同じ方法を使用してPoint2 の座標を取得します。
3. 現在のユーザー座標系を選択し、プレビューをクリックすると上のシミュレーション領域で生成された安全領域を確認できます。

> 説明:
> - ユーザーは、点のデカルト座標を手動入力または変更することもできます。
> - 設定を変更したらプレビューをクリックするとシミュレーション領域の表示が更新されます。

4. ロボットのエンドツールボールが安全領域を超えた場合の動作を設定します。

5. 安全な方向を設定します。外側は立方体の外側の領域が安全領域であることを示します。内側は立方体の内部が安全領域であることを示します。
6. **OK**をクリックして、安全領域を追加します。

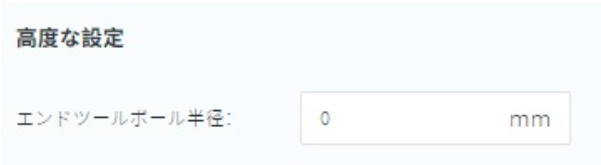
安全領域を変更する



安全領域は**Index**と領域形状以外のプロパティは変更できます。信号出力はアクションが**IO**出力である安全領域だけが設定可能です。右側のスイッチで安全領域を有効にするかどうかを制御できます（ロボットが無効状態でのみ操作可能）。有効な安全領域がロボットと干渉し、右側のシミュレーション領域に表示されます。

安全領域を選択したら、×削除をクリックすると選択した安全領域を削除できます。

詳細設定



エンドツールボール半径

エンドツール球の半径を指定します（TCPを中心とするカスタム半径の球状空間）。当該球状空間が非安全領域と干渉すると、セーフウォールがトリガーされます。0に設定すると、TCPのみが制限領域と干渉します。

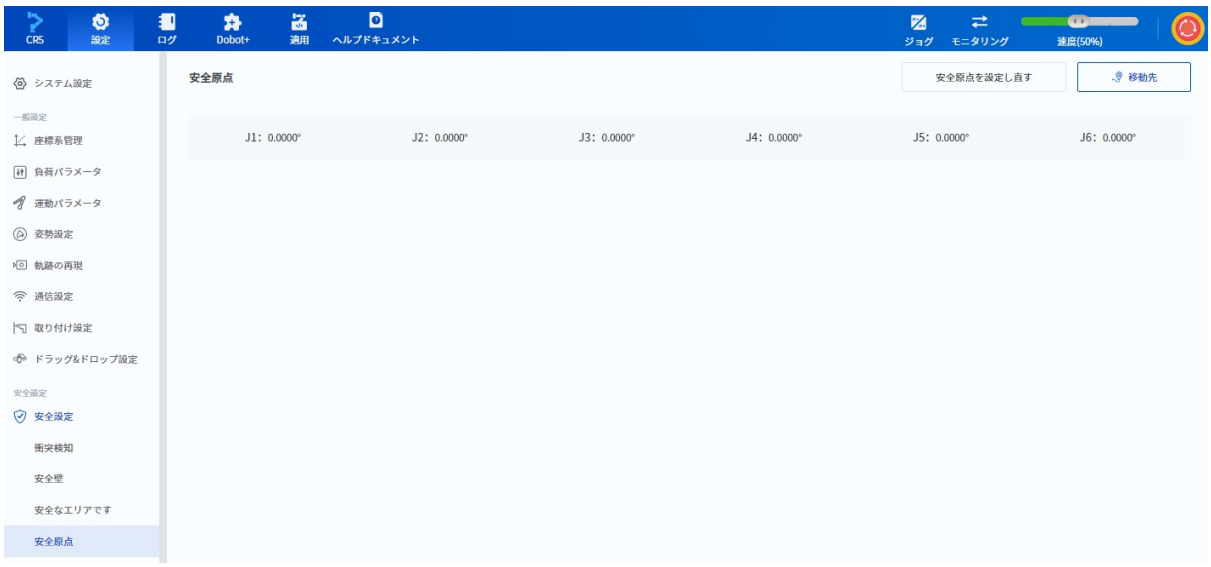
復元モード

ロボットが非安全領域にある場合、ロボットを有効にすると、下の図に示すように、ポップアップウィンドウが表示され、「復元モードに入りました」をユーザーに通知します。

復元モードでは、ユーザーがロボットの動きを制御するにはジョグまたはドラッグ＆ドロップだけを使用できます。ロボットが安全領域に入ると、自動的に復元モードが終了し、ポップアップウィンドウが表示されます。

10.11.4 安全原点

安全原点はカスタマイズ可能な姿勢で、デフォルトの姿勢はゼロ位置の姿勢です。つまり、各ジョイントの角度は0です。



- を長押しするとへ移動するロボットを安全原点に移動させることができます。
- をクリックすると安全原点を設定し直す安全原点を変更できます。

  各関節の角度を手動で入力するか、ロボットを指定の姿勢に移動させてからをクリック点位置を取得し、ロボットの現在の各関節の角度を読み取ることができます。クリックしてデフォルト点位置への復元安全原点をデフォルトの位置に戻すことができます。

  各関節の角度を確定した後、クリック保存すると安全原点が更新されます。

10.11.5　関節ブレーキ

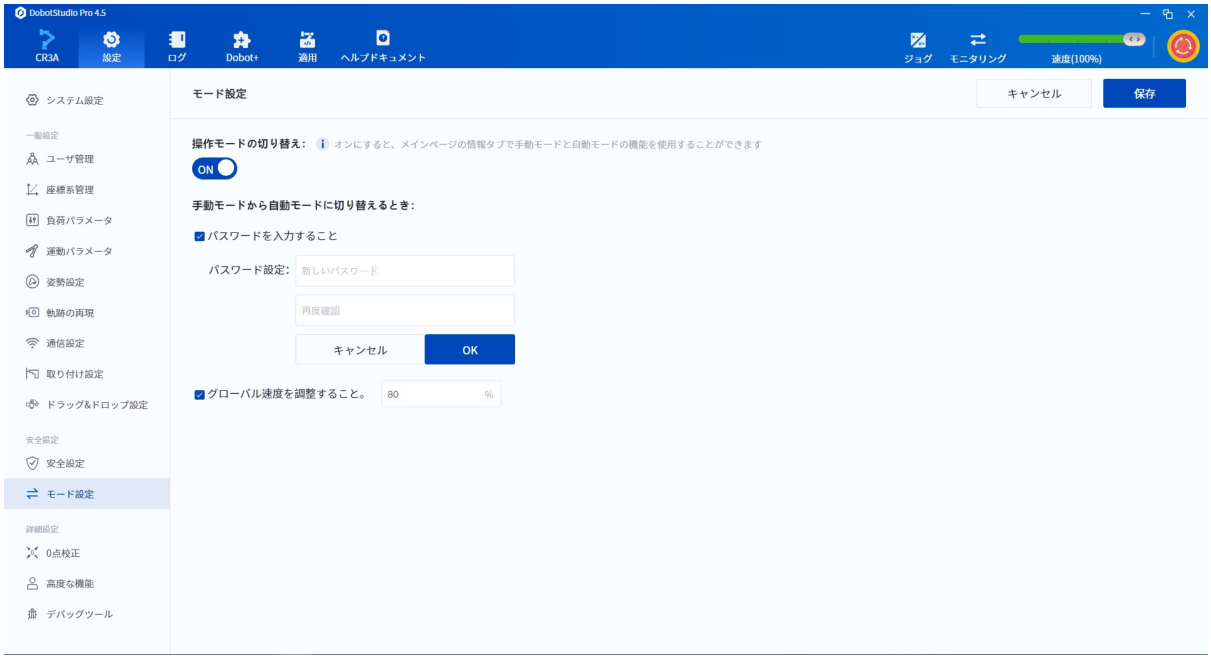
ロボットがディセーブル状態にある時、関節の移動を防止するため、関節は自動的にブレーキをかけてモータの位置をロックし、機械の移動部分が自重や外力で移動しないようにします。

ユーザーが関節のドラッグ操作を実行する場合、ブレーキをオンにすることができます。つまり、ロボットをディセーブルした後、手動で関節を保持し、動かしたい関節のスイッチをオンにして、保存をクリックします。

> ⚠警告：
> ブレーキを有効にするときは、手で関節を支えて、ロボットアームが自重で落下するのを防ぐ必要があります。

10.12 モード設定

ロボットの手動/自動モード切り替え機能は出荷時はデフォルトでオフになっています。このページでオンにすることができます。



各モードの違い

モード切り替え機能をオフにした場合、すべての機能を使用することができます。

手動モードでは、ロボットがジョグ以外の移動を実行する場合、グローバル速度は50%を超えることはできません。

自動モードのグローバル速度には制限がありません。手動モードから自動モードに切り替えるときのデフォルトのグローバル速度はこのページで設定できます。

速度を除き、手動と自動モードの主な違いはリモートIO/Modbus制御機能と安全I/O機能の入力が有効かどうかであり、詳細については下表をご参照ください。

リモート**IO/Modbus**制御

機能	手動モード	自動モード
開始	X	√
停止	X	√
一時停止	X	√
イネーブル	√	√

ディセーブル	√	√
アラームをクリア	√	√
ドラッグに入る	√	X
ドラッグを終了	√	X
プロジェクト選択	X	√

安全**I/O**

機能	手動モード	自動モード
ユーザーによる非常停止	√	√
防護停止	X	√
防護停止リセット	X	√
縮小モード	X	√

動作モード切替機能をオンにした後、手動モードから自動モードに切り替えるときにパスワードを入力するかどうかを選択できます。このオプションをチェックした場合、パスワードを設定する必要があります（初期パスワードはありません）。

この機能をオンにすると、トップページの情報パネルに手動･自動モード切替スイッチが表示されます。デフォルトでは手動モードです。

ロボットが次の状態にない場合は、スイッチをクリックして自動モードに切り替えることができます。自動モードパスワードを設定した場合、切り替えるのにパスワードを入力する必要があります。

- プロジェクトデバッグ状態
- 軌跡再現状態
- ドラッグ状態
- 関節ブレーキ解除状態

グローバル速度調整機能にチェックを入れた場合は、自動モードに切り替えると、グローバル速度が設定値に自動的に調整されます。

自動モードでは、いつでもスイッチをクリックして手動モードに切り替えることができます。
切り替えるときにロボットが移動している場合、切り替えた直後にロボットは停止しますが、有効状態は変わりません。

10.13　ゼロ点キャリブレーション

ロボットアームのモーターや減速機などのトランスミッション部品を交換したとき、またはワークと衝突したときなど状況で、ロボットアームのゼロ点位置に変化が生じたときは、ロボットアームに対してゼロ点キャリブレーションを行う必要があります。

> 説明：
> ゼロ点キャリブレーションはゼロ点の位置が変化する場合にのみ使用しますので、慎重に操作してください。

ページの指示に従って、ロボットアームをゼロ点の姿勢に移動します（ロボットアームの各関節のゼロ点シールをキャリブレーションに使用できます。詳細については、該当するロボットアームのハードウェアマニュアルをご参照ください）。その後、有効な状態でゼロ点キャリブレーションボタンをクリックします。

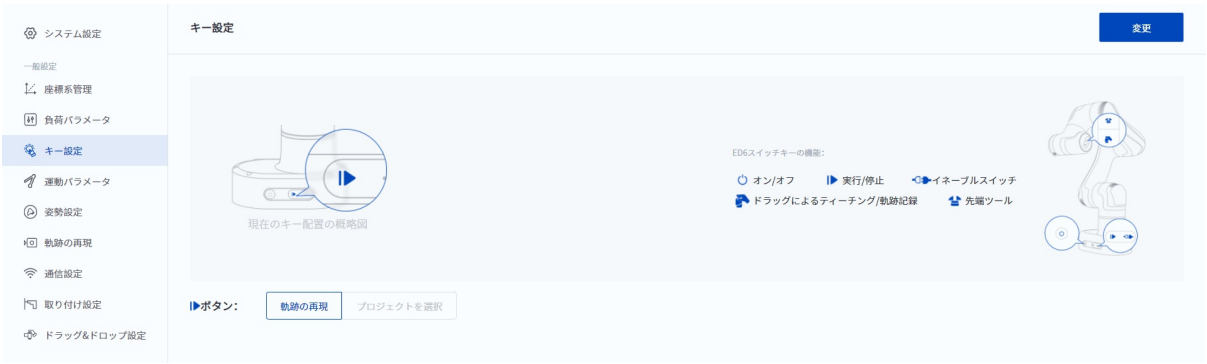
キャリブレーション完了後、一度ロボットをディセーブルにする必要があります。制御パネルに関節座標を表示することができます。このとき、J1～J6の値はいずれも0です。

10.14 高度な機能

現場により高度機能をオン/オフする場合は、この画面で設定できます。特別な要件がない場合は、デフォルト値を保持することをおすすめします。

10.15 ボタン設定（Magician E6）

ロボットのタイプがMagician E6の場合、ベースの運転/停止ボタンの機能を設定できます。



右上の変更ボタンを押した後に運転/停止ボタンの機能を変更できます。

- このボタンのデフォルトの機能はトラジェクトリ再現で、最近一度録音されたトラジェクトリを再現します。
- 機能をプロジェクトの実行に設定する場合、実行するプロジェクトを選択する必要があります。

このボタンがどの機能に設定されていても、ロボットのトラジェクトリ再現またはプロジェクトの実行中にこのボタンを押すと、運転が停止します。

付属書A Modbusレジスタの定義

1 Modbusの紹介

Modbusはシリアル通信プロトコルで、このプロトコルにより、ロボットは外部機器と通信することができます。外部機器（PLCなど）を通じてロボットを制御する場合、外部機器はModbusマスターとして、ロボットはModbusスレーブとなります。

Modbusプロトコルに基づき、Dobot 6軸協働ロボットに分配されるModbusアドレスは以下のとおりです。

- 00001-09999：コイルレジスタ、値は0または1のみをサポートし、ロボットを制御するために使用され、読み書きが可能です。
- 10001-19999：接点レジスタ、値は0または1のみをサポートし、ロボットの状態を取得するために使用され、読み取り専用です。
- 30001-39999：入力レジスタ、値は最大で64ビットの倍精度浮動小数点数をサポートし、ロボットのリアルタイムフィードバックデータを取得するために使用され、読み取り専用です。
- 40001-49999：保持レジスタ、値は最大で64ビットの倍精度浮動小数点数をサポートし、ロボットのPLC連携のために使用され、読み書きが可能です。

各種レジスタに対応するModbus機能コードは、標準のModbusプロトコルに従います。

レジスタタイプ	読み取りレジスタ	単一レジスタの書き込み	複数レジスタの書き込み
コイルレジスタ	01	05	0F
接点レジスタ	02	-	-
入力レジスタ	04	-	-
保持レジスタ	03	06	10

ロボットには、外部機器にアクセスできるようにするため、2つのテーブルが用意されています。map1は502 (Modbus-TCP)と503 (RTU-over-TCP)ポートを通じてアクセスし、map2は1502 (Modbus-TCP)と1503 (RTU-over-TCP)ポートを通じてアクセスできます。

そのうち、map1の一部のアドレスはロボットシステムにデフォルトで使用されています。詳細は下記のレジスタの定義を参照してください。map2は現在は空です。ユーザーは実際の必要性に基づいて使用してください。

2 コイルレジスタ（map1、ロボット制御）

PLCアドレス	スクリプトアドレス (Get/SetCoils)	レジスタタイプ	機能
00001	0	Bit	開始
00002	1	Bit	停止
00003	2	Bit	一時停止
00004	3	Bit	イネーブル
00005	4	Bit	ディセーブル
00006	5	Bit	アラームをクリア
00007	6	Bit	ドラッグに入る
00008	7	Bit	ドラッグを終了
00009	8	Bit	自動モードに切り替える
00010	9	Bit	手動モードに切り替える
00011~01025	10~1024	Bit	予約ビット
01025~09999	1024~9998	Bit	カスタム

3 接点レジスタ定義（map1、ロボット状態）

PLCアドレス	スクリプトアドレス (GetInBits)	レジスタタイプ	機能
10001	0	Bit	動作状態
10002	1	Bit	停止状態
10003	2	Bit	一時停止状態
10004	3	Bit	安全原点状態
10005	4	Bit	セーフティスキン一時停止状態
10006	5	Bit	アイドル状態
10007	6	Bit	電源投入状態
10008	7	Bit	イネーブル状態
10009	8	Bit	アラーム状態

10010	9	Bit	衝突状態
10011	10	Bit	ドラッグ状態に入る
10012~19999	11~9998	Bit	未定義

4 入力レジスタ定義（map1、ロボットのリアルタイムフィードバックデータ）

PLCアドレス30001～30999および31722～39999のレジスタ機能は未定義です (U16タイプコントローラのSNコードを格納する32001を除く)。エンディアンはリトルエンディアンを反転したエンディアンです。

PLCアドレス	データタイプ	数	バイトサイズ	スクリプトアドレス **(GetInRegs)**	タイプ	機能
31000	unsigned short	1	2	999	U16	データ有効性
31001	unsigned short	1	2	1000	U16	メッセージバイトの合計長さ
31002~31004	-	-	-	1001~1003	-	予約ビット
31005~31008	uint64	1	8	1004~1007	U64	デジタル入力の詳細はDI/DOの説明をご覧ください
31009~31012	uint64	1	8	1008~1011	U64	デジタル出力の詳細はDI/DOの説明をご覧ください
31013~31016	uint64	1	8	1012~1015	U64	ロボットモードの詳細はRobotModeの説明をご覧ください
31017~31020	uint64	1	8	1016~1019	U64	Unixタイムスタンプ（単位ms）
31021~31024	-	-	-	1020~1023	-	予約ビット
31025~31028	uint64	1	8	1024~1027	U64	メモリ構造試験標準値 0x0123 4567 89AB CDEF
31029~31032	-	-	-	1028~1031	-	予約ビット
31033~31036	double	1	8	1032~1035	F64	速度比率

31037~3104	-	-	-	1036~1039	-	予約ビット
31041~31044	double	1	8	1040~1043	F64	制御パネルの電圧
31045~31048	double	1	8	1044~1047	F64	ロボット電圧
31049~31052	double	1	8	1048~1051	F64	ロボット電流
31053~31096	-	-	-	1052~1095	-	予約ビット
31097~31120	double	6	48	1096~1119	F64	目標関節位置
31121~31144	double	6	48	1120~1143	F64	目標関節速度
31145~31168	double	6	48	1144~1167	F64	目標関節加速度
31169~31192	double	6	48	1168~1191	F64	目標関節電流
31193~31216	double	6	48	1192~1215	F64	目標関節トルク
31217~31240	double	6	48	1216~1239	F64	実際の関節位置
31241~31264	double	6	48	1240~1263	F64	実際の関節速度
31265~31288	double	6	48	1264~1287	F64	実際の関節電流
31289~31312	double	6	48	1288~1311	F64	TCPセンサ力値 （6次元力計算による）
31313~31336	double	6	48	1312~1335	F64	実際のTCPデカルト座標値
31337~31360	double	6	48	1336~1359	F64	実際のTCPデカルト速度値
31361~31384	double	6	48	1360~1383	F64	TCP力値 （関節電流により計算）
31384~31408	double	6	48	1384~1407	F64	目標のTCPデカルト座標値
31409~31432	double	6	48	1408~1431	F64	目標のTCPデカルト速度値
31433~31456	double	6	48	1432~1455	F64	関節温度
31456~31480	double	6	48	1456~1479	F64	関節制御モード。 8：位置モード。 10：トルクモード
31481~31504	double	6	48	1480~1503	F64	関節電圧
31505~31506	-	-	-	1504~1505	-	予約ビット
31507	char	1	1	1506下位バイト	I8	ユーザ座標系
31507	char	1	1	1506上位バイト	I8	工具座標系

31508	char	1	1	1507下位バイト	I8	アルゴリズムキュー運転フラグ
31508	char	1	1	1507上位バイト	I8	アルゴリズムキュー一時停止フラグ
31509	char	1	1	1508下位バイト	I8	関節速度比率
31509	char	1	1	1508上位バイト	I8	関節加速度比率
31510	char	1	1	1509下位バイト	I8	関節加加速度比率
31510	char	1	1	1509上位バイト	I8	デカルト位置速度比
31511	char	1	1	1510下位バイト	I8	デカルト姿勢速度比
31511	char	1	1	1510上位バイト	I8	デカルト位置加速度比
31512	char	1	1	1511下位バイト	I8	デカルト姿勢加速度比
31512	char	1	1	1511上位バイト	I8	デカルト位置加加速度比
31513	char	1	1	1512下位バイト	I8	デカルト姿勢加加速度比
31513	char	1	1	1512上位バイト	I8	ロボットのブレーキ状態の詳細はBrakeStatusの説明をご覧ください
31514	char	1	1	1513下位バイト	I8	ロボットのイネーブル状態
31514	char	1	1	1513上位バイト	I8	ロボットのドラッグ状態
31515	char	1	1	1514下位バイト	I8	ロボットの動作状態
31515	char	1	1	1514上位バイト	I8	ロボットのアラーム状態
31516	char	1	1	1515下位バイト	I8	ロボットのジョグ状態
31516	char	1	1	1515上位バイト	I8	ロボットタイプの詳細は、RobotTypeの説明をご覧ください
31517	char	1	1	1516下位バイト	I8	ボタンパネルドラッグ信号
31517	char	1	1	1516上位バイト	I8	ボタンパネルイネーブル信号

31518	char	1	1	1517下位バイト	I8	ボタンパネル記録信号
31518	char	1	1	1517上位バイト	I8	ボタンパネル再現信号
31519	char	1	1	1518下位バイト	I8	ボタンパネルグリッパ制御信号
31519	char	1	1	1518上位バイト	I8	6次元力オンライン状態
31520	char	1	1	1519下位バイト	I8	衝突状態
31520	char	1	1	1519上位バイト	I8	(セーフティスキン) 前腕が一時停止状態に近づいています
31521	char	1	1	1520下位バイト	I8	(セーフティスキン) J4は一時停止状態に近づいています
31521	char	1	1	1520上位バイト	I8	(セーフティスキン) J5は一時停止状態に近づいています
31522	char	1	1	1521下位バイト	I8	(セーフティスキン) J6は一時停止状態に近づいています
31522	char	1	1	1521上位バイト	I8	予約ビット
31523~31552	-	-	-	1522~1551	-	予約ビット
31553~31556	double	1	8	1552~1555	F64	予約ビット
31557~31560	uint64	1	8	1556~1559	U64	現在のアルゴリズムキューID
31561~31584	double	6	48	1560~1583	F64	実際のトルク
31585~31588	double	1	8	1584~1587	F64	負荷重量(kg)
31589~31592	double	1	8	1588~1591	F64	X方向偏心距離(mm)
31593~31596	double	1	8	1592~1595	F64	Y方向偏心距離(mm)
31597~31600	double	1	8	1596~1599	F64	Z方向偏心距離(mm)
31601~31624	double	6	48	1600~1623	F64	ユーザー座標値
31625~31648	double	6	48	1624~1647	F64	工具座標値
31649~31652	double	1	8	1648~1651	F64	軌跡再現運転インデックス
31653~31676	double	6	48	1652~1675	F64	現在の6次元力データの元の値
31677~31692	double	4	32	1676~1691	F64	[qw、qx、qy、qz] タ

31677~31692	double	4	32	1676~1691	F64	ーゲット四元数
31693~31708	double	4	32	1692~1707	F64	[qw、qx、qy、qz] 実際の四元数
31709	unsigned short	1	2	1708	U16	手動/自動状態。 1：手動 2：自動モード 0：モード切り換えが有効になっていません
31710~31721	double	1	24	1709~1720	F64	予約ビット
			1440			合計1440バイト

移動パラメータフィードバック値の説明

移動パラメータ（速度、加速度など）がプロジェクトで個別に設定されている場合、関連するフィードバック値はすぐには更新されませんが、ロボットが次の移動命令を実行するまで更新されます。

DI/DOの説明

DI/DOは各8バイトを占め、各バイトは8ビット（バイナリ）です。DI/DOはそれぞれ最大64のポート状態を示すことができます。各バイトは小さい方から高い方で端子の状態を示しており、1は対応する端子がON、0は対応する端子がOFFまたは対応する端子がないことを示しています。

例えば、DIの1つ目のバイトは0x01、バイナリ表示では00000001です。小さい方から大きい方でそれぞれD1_1～DI_8の状態を示しています。つまり、DI_1はONで、そのた7つはOFFです。

2つ目のバイトは0x02、バイナリ表示では00000010です。小さい方から大きい方でそれぞれD1_9～DI_16の状態を示しています。つまり、DI_10はONで、その他7つはOFFです。

後続のバイトはこれから類推でき、制御盤の種類とIO端子の数に基づいて、IO端子数を超えるバイナリのバイトはすべて0となります。

RobotModeの説明

取る値	定義	説明
1	ROBOT_MODE_INIT	初期化状態
2	ROBOT_MODE_BRAKE_OPEN	任意の関節のブレーキリリース
3	ROBOT_MODE_POWEROFF	ロボットアームのシャットダウン状態
4	ROBOT_MODE_DISABLED	無効（ブレーキがリリースされていない）
5	ROBOT_MODE_ENABLE	有効かつアイドル
6	ROBOT_MODE_BACKDRIVE	ドラッグモード
7	ROBOT_MODE_RUNNING	動作状態（プロジェクト、TCPキュー移動等）

8	ROBOT_MODE_SINGLE_MOVE	単一移動状態（ジョグ、RunTo等）
9	ROBOT_MODE_ERROR	クリアしていないアラームがあります。この状態の優先度は一番上であるため、ロボットアームがどのような状態であっても、アラーム時は9に戻ります。
10	ROBOT_MODE_PAUSE	プロジェクトの一時停止状態
11	ROBOT_MODE_COLLISION	衝突検出トリガ状態

BrakeStatusの説明

このバイトとは、各関節のブレーキ状態をビット単位で示し、対応するビットの1は関節のブレーキがリリースされていることを示します。ビット数と関節の対応関係は下表のとおりです。

ビット数	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
意味	予約ビット	予約ビット	関節1	関節2	関節3	関節4	関節5	関節6

例：

- 0x01（00000001）：関節6のブレーキがリリース
- 0x02（00000010）：関節5のブレーキがリリース
- 0x03（00000011）：関節5と関節6のブレーキがリリース
- 0x03（00000100）：関節4のブレーキがリリース

RobotTypeの説明

取る値	代表的な機種
3	CR3
5	CR5
7	CR7
10	CR10
12	CR12
16	CR16
101	Nova 2
103	Nova 5
113	CR3A
115	CR5A
117	CR7A
120	CR10A
122	CR12A

126	CR16A
130	CR20A
150	Magician E6

5 保持レジスタ定義（**map1**、ロボット**PLC**連携）

PLCアドレス	スクリプトアドレス **(Get/SetHoldRegs)**	レジスタタイプ	機能
40001~41024	0~1023	-	予約ビット
41025~49999	1024~9998	-	カスタム

付属書B ブロックプログラミングブロック説明

全般的な説明

ブロックの種類

ブロックプログラミングでは、形状によりブロックは4つの大きなカテゴリに分けられます。

丸いブロック

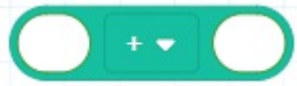
この形状のブロックはブロック列の先頭に使用され、実行するすべてのブロックはこの種のブロックの下に接続する必要があります。イベントブロックグループのみがこの種のブロックを含みます。

四角いブロック

この形状のブロックは命令を実行するために使用され、ブロックプログラムの主要な部分を構成します。ブロック内の楕円形と菱形のパラメータスロットには、対応する形状のブロックを使用できます。

四角いブロックには一種の変種があり、他のブロックをブロック内にネストすることができ、プロセス制御に使用します。

楕円形ブロック



楕円形ブロックは実行後に変数（数値/文字列/配列など）を返し、四角いブロックの対応する形状のパラメータスロットに入れてパラメータとして使用します。

菱形ブロック

菱形ブロックは条件判断に使用され、実行後にtrueまたはfalseを返し、四角いブロックの対応する形状のパラメータスロットに入れてパラメータとして使用します。

動作方法

ロボットがサポートする動作方法は次のいくつかに分けられます。

関節移動

ロボットは現在の各関節角度と目標点の各関節角度の差に基づいて各関節の動きを計画し、すべての関節が同時に動きを完了します。関節の動きはTCP（Tool Center Point）の動きの軌跡を制約せず、通常この軌跡は直線ではありません。

関節の動きは特異位置の制限を受けません（特異点の位置についてはロボットの対応するハードウェアマニュアルを参照）。そのため、動きの軌跡に要求がない場合や目標位置が特異位置近くにある場合は、関節の動きを使用することをお勧めします。

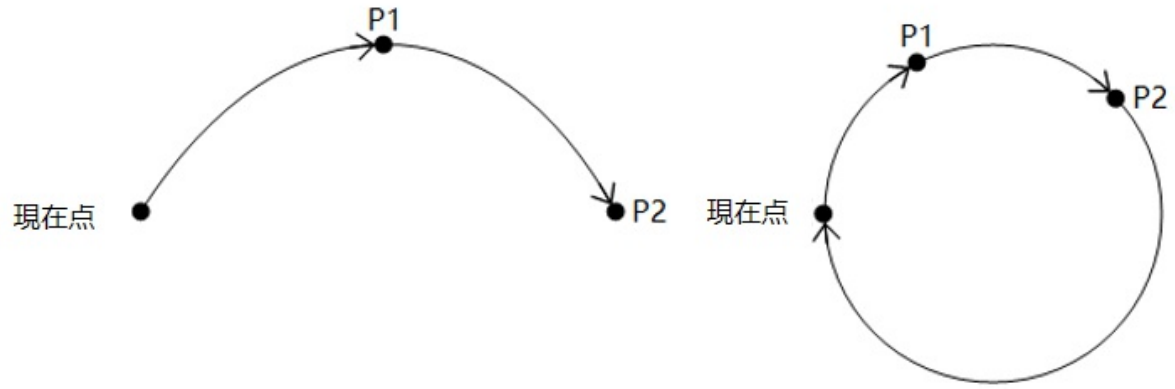
直線移動

ロボットは現在の姿勢と目標点の姿勢に基づいて動きの軌跡を計画し、TCPの動きの軌跡が直線になり、末端の姿勢が動きの過程で一定の速度で変化します。

動きの軌跡が特異位置を通る場合、直線の動きの命令をロボットに送信するとエラーが発生する可能性があります。その場合は点の位置を再計画するか、特異位置近くで関節の動きを使用することをお勧めします。

弧の動き

ロボットは現在の位置、P1、P2の3つの非共線点を使用して円弧または完全な円を確定します。動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。



動きの軌跡が特異位置を通る場合、弧の動きの命令をロボットに送信するとエラーが発生する可能性があります。その場合は点の位置を再計画するか、特異位置近くで関節の動きを使用することをお勧めします。

座標系パラメータ

モーション ビルディング ブロックは、詳細設定により、ポイントに対応するユーザ座標系と工具座標系を指定できます。パラメータの優先順位は次のとおりです。

1. 詳細設定で座標系が指定されている場合は、指定された座標系が使用されます。点パラメータがティーチング点である場合、ティーチング点の姿勢座標を指定座標系の値に換算し、使用することができます。
2. 詳細設定で座標系が指定されていない場合は、点パラメータがティーチング点であると、ティーチング点自体の座標系インデックスを使用します。
3. 詳細設定で座標系が指定されていない場合は、点パラメータが関節変数または姿勢変数であると、制御ブロック グループに設定されたグローバル座標系を使用します (「ユーザ座標系の設定とツール座標系の設定ビルディング ブロックをご参照ください。ブロックの設定が使用されていない場合、デフォルトの座標系は0です)

> 説明:
> プロジェクトの実行を開始すると、デフォルトのグローバル座標系はすべて0にリセットされ、プロジェクトの実行前にポイント動作パネルで設定した値とは無関係になります。

速度パラメータ

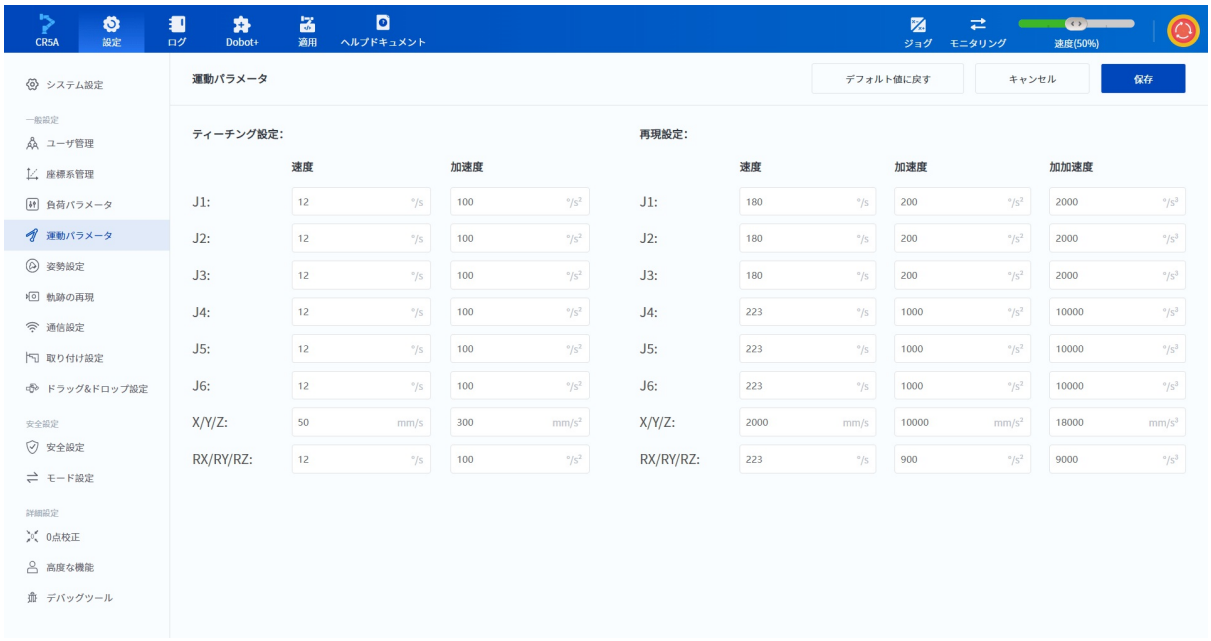
相対速度

動きのブロックは、高度な設定でロボットがその動きの命令を実行するときの加速度（Accel）と速度（V）を指定できます。

```
ロボットの実際の動きの速度 = 最大速度 × グローバルの速度比率 × 命令の速度比率
```

```
ロボットの実際の動きの加速度 = 最大加速度 × 命令の速度比率
```

そのうち、最大速度/加速度は再現設定によって制御され、制御ソフトウェアの運動パラメータページで確認および変更することができます。



グローバルの速度比率は制御ソフトウェア（上図の右上）またはグローバルの速度比率の設定ブロックによって設定することができます。

命令の速度は、運動ブロックの詳細設定で指定された速度比率で、詳細設定で運動加速度/速度が指定されていない場合、デフォルトでは対応する関節/直線速度/加速度ブロックで設定された値を使用し、ブロックの設定が使用されていない場合、デフォルトの値は100です。

絶対速度

直線および弧運動ブロックの高度な設定では、その運動命令を実行する際のロボットアームの絶対速度（Speed）を指定することができます。

絶対速度は、グローバルの速度比率から影響を受けないが、再現設定の最大速度の制限を受けます（ロボットが低減モードに入っている場合、低減後の最大速度の制限を受けます）。すなわち、絶対速度が再現設定の最大速度よりも大きい場合、最大速度に準じます。

例えば、直線運動の絶対速度を1000に設定し、再現設定の最大速度2000より小さい場合、ロボットは目標速度1000mm/sで運動します。この目標速度は、現時点のグローバル的な速度とは無関係です。ただし、ロボットが縮減モード（縮減率を10%で仮定）である場合、最大速度は200に変わり、1000より小さいので、この時のロボットは200mm/sを目標速度として移動します。

絶対速度と速度は同時に設定することはできません。

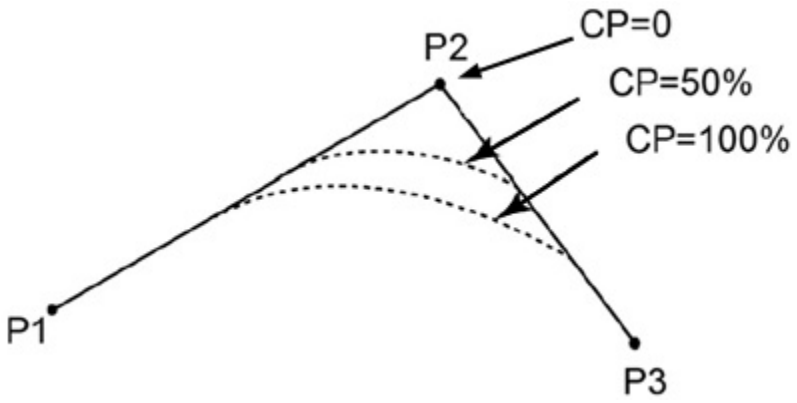
スムーズな遷移

ロボットが複数の点を通過して連続移動する場合、連続経路制御方式で中間点を通過し、ロボットを強く回転することを回避することができます。ユーザが指定する複数の経路点が、異なるツール座標系を基にしている場合、連続経路制御を行うことはできません。

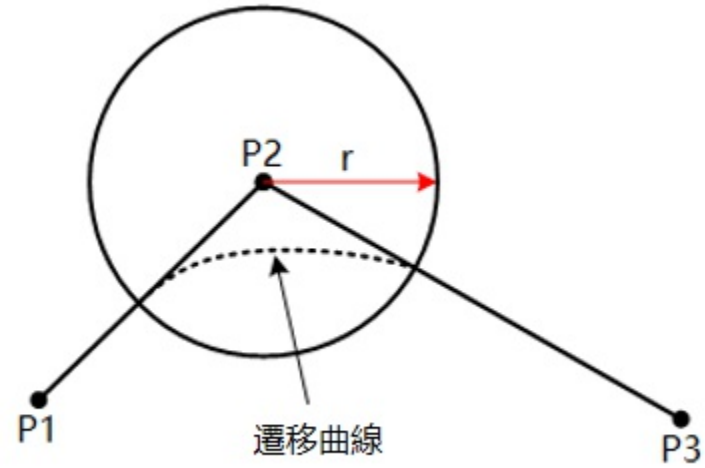
ユーザは運動命令の高度な設定で、その運動命令から次の運動命令へのスムーズな遷移比率（CP）またはスムーズな遷移半径（R）を指定することができます。

> 説明：
> 関節運動モードでは、スムーズな遷移半径（R）の設定はサポートされていません。

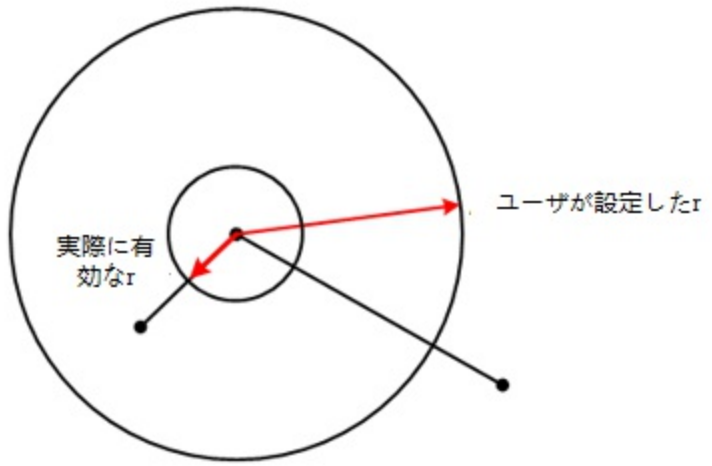
連続経路制御比率を設定する場合、システムは移動曲線の円弧を自動的に計算します。下図のとおり、CP値が大きい曲線になるほどスムーズになります。CP移動曲線は移動速度/加速度の影響を受けます。たとえ点およびCP値が同じであっても、移動速度/加速度が異なる場合の移動曲線の円弧は異なります。



連続経路制御の半径を設定する場合、システムは移動点を円の中心として、指定半径を基にして移動曲線を計算します。R移動曲線は移動速度/加速度の影響を受けず、点と移動半径のみで決定されます。



ユーザが設定する移動半径が大きすぎる場合（開始点/終了点と移動点との間の距離を超過）、システムは開始点/終了点と移動点との間の比較的短い距離の半分を移動半径を使用して、移動曲線を計算します。



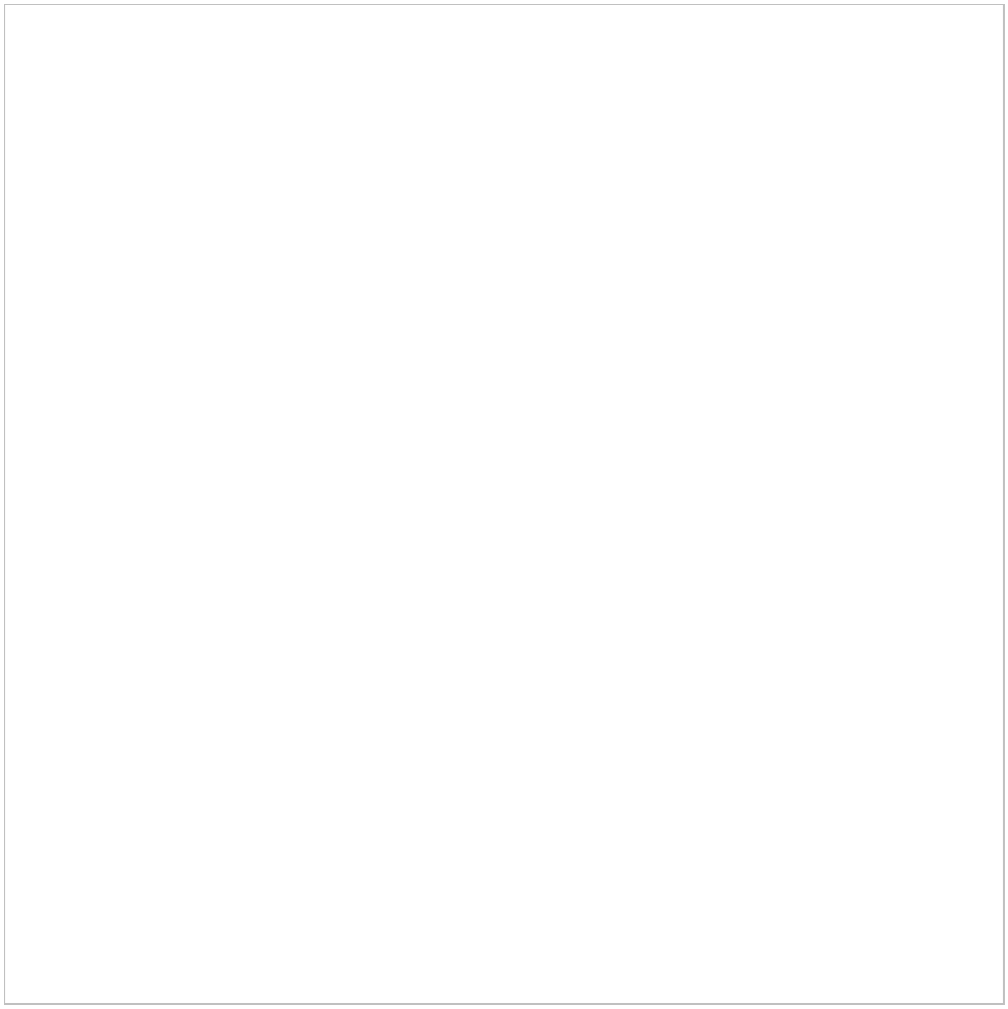
詳細設定でスムーズな遷移比率や半径が指定されていない場合、デフォルトではスムーズな遷移比率の設定ブロックで設定された値を使用し、ブロック設定が使用されていない場合のデフォルト値は0です。

> 説明：
> スムーズな遷移は、ロボットが中間点を通過せずに運動することを引き起こします。したがって、スムーズな遷移が設定されている場合、2つの運動命令の間のIO信号出力または機能設定（例えば、安全スキンのオン/オフ）命令は遷移過程で実行されます。
> ロボットが正確に中間点に到達したときに命令を実行したい場合は、前の命令のスムーズな遷移パラメータを0に設定してください。

停止条件

一部の移動ブロックは、高度な構成による停止条件の指定をサポートしており、ロボットによる移動命令の実行中に、指定された停止条件が満たされると、ロボットは現在の移動を終了し、次の命令を直接実行します。

- サポートされる判定条件には、IO、変数、およびLua構文に準拠した任意の式が含まれます。
- 判定条件は3つまで対応し、複数の条件の間ではキーワード**AND** (すべての条件が満たされた場合にのみ満たされる)/**OR** (いずれかの条件が満たされれば満たされる) で接続します。**AND**と**OR**の混合使用はサポートされていません。



イベントブロックグループ

イベントブロックグループは、プログラムが開始されるためのマーカーとして使用され、イベントブロックの下に接続されたブロックのみが実行されます。

実行開始

説明：プログラムのメインスレッドの識別。新しいプロジェクトを作成すると、プログラミング領域には初期設定で実行開始識別が表示されます。その下に他のイベント以外のブロックを接続してプログラミングしてください。

使用制限：「実行を開始」ブロックは、1つのプロジェクトに1つしか存在できません。

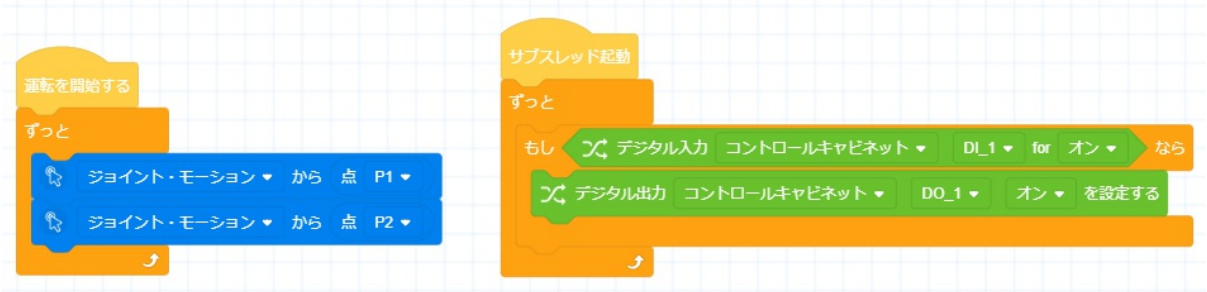
サブスレッドの起動

説明：プログラムのサブスレッドの識別。サブスレッドはメインスレッドと同期して実行されますが、ロボット制御コマンドを使用することはできません。また、変数演算やIO制御などを行うことができます。プログラムロジックの必要に応じて、サブスレッドを使用するかどうかを選択してください。

使用制限：一つのプロジェクトには最大4つのサブスレッドしか設定できません。

例：

以下の例では、プロジェクトを起動すると、メインスレッドとサブスレッドが同時に起動します。メインスレッドの動作命令とサブスレッドのIO命令は互いに干渉せず、それぞれのシーケンスに従って実行されます。



制御ブロックグループ

制御ブロックグループは、プログラムの実行経路を制御するために使用されます。

条件が満たされるまで待ちます

説明：プログラムは一時停止し、パラメータがtrueになると実行を続行します。

パラメータ：他の六角形ブロックをパラメータとして使用します。

例：

ロボットがP1に移動し、DI1がオンになるまで待ち、その後P2に移動します。

n回繰り返し実行します

説明：そのブロックの間に他のブロックを入れ子にすると、入れ子にされたブロック命令が指定された回数繰り返し実行されます。

パラメータ：繰り返し実行する回数。

継続的に繰り返し実行します

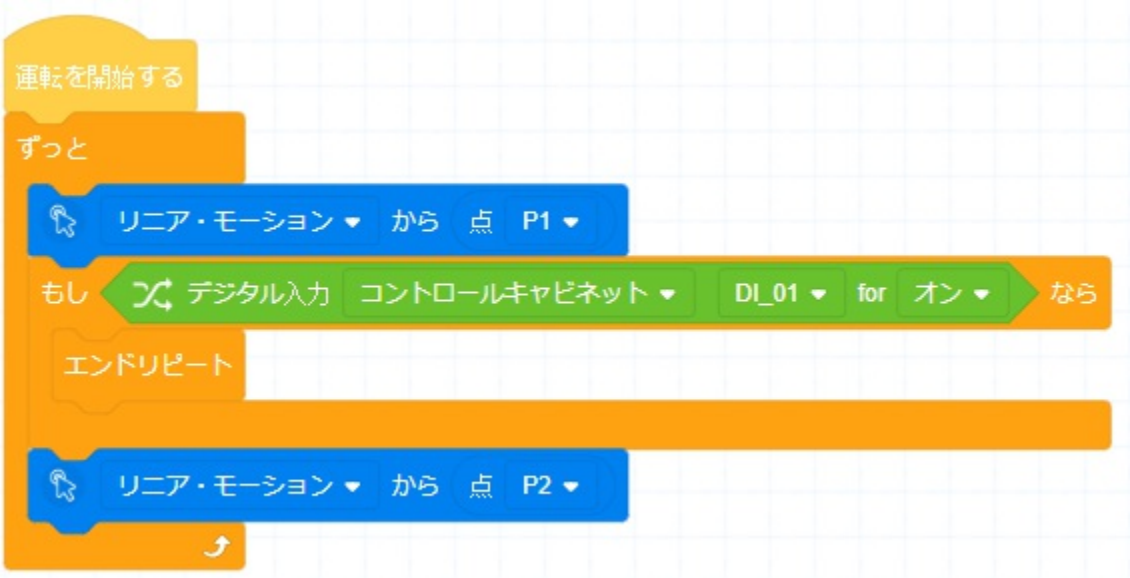
説明：他のブロックをそのブロックの間に入れ子にすると、ネストされたブロック命令は繰り返しブロックの終了に遭遇するまで繰り返し実行されます。このブロックの下には他のブロックを接続することはできません。

繰り返しを終了します

説明：下記の繰り返し実行クラスのブロックに埋め込むために使用し、プログラムがこのブロックに到達した時点で直ちに繰り返しを終了し、繰り返しブロックの後のブロック命令を実行します。

例：

P1とP2の往復運動を繰り返し実行しますが、ロボットがP1点に移動した後にDI1がオンになった場合、すぐに繰り返しを終了します。



条件が満たされた後に実行します

説明：パラメータがtrueであれば、ネストされたブロック命令を実行します。パラメータがfalseであれば、次のブロック命令に直接ジャンプします。

パラメータ：他の六角形ブロック（戻り値はブール値、つまりtrueまたはfalse）をパラメータとして使用します。

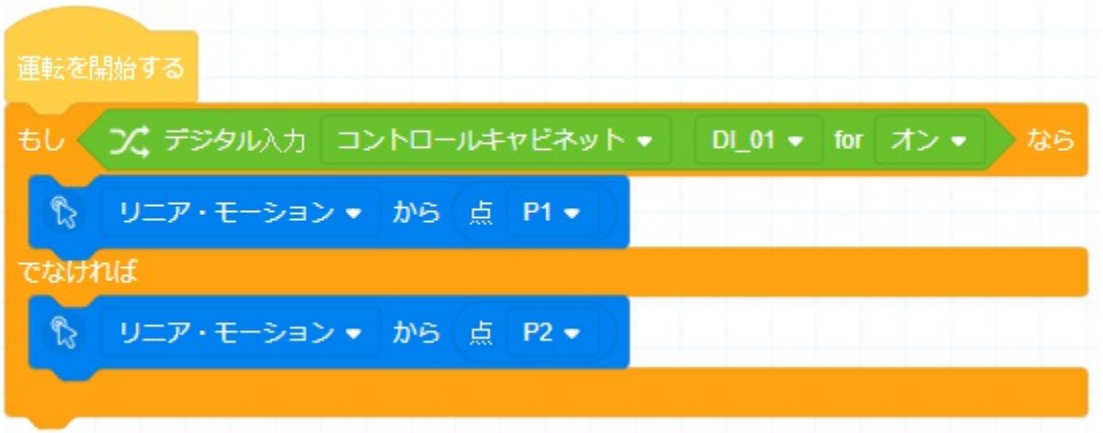
条件が満たされたか否かによってそれぞれ実行します

説明：パラメータがtrueであれば、「そうでない場合」の前にネストされたブロック命令が実行されます。パラメータがfalseであれば、「そうでない場合」の後にネストされたブロック命令が実行されます。

パラメータ：他の六角形ブロック（戻り値はブール値、つまりtrueまたはfalse）をパラメータとして使用します。

例：

DI1がオンであれば、ロボットはP1点に移動し、そうでなければP2点に移動します。



条件が満たされるまで繰り返し実行します

説明：ネストされたブロック命令は、パラメータがtrueになるまで繰り返し実行されます。

パラメータ：他の六角形ブロック（戻り値はブール値、つまりtrueまたはfalse）をパラメータとして使用します。

タグを設定します

説明：タグを設定し、設定後にタグジャンプブロックを用いてジャンプします。

パラメータ：タグ名の先頭は文字でなければなりません。スペースなどの特殊文字は使用できません。

タグジャンプ

説明：プログラムはそのブロックを実行すると、指定されたタグに直接ジャンプし、タグに続くブロック命令を実行します。

パラメータ：設定されたタグ名。

例：

DI1がオンであれば、ロボットは直接P1に移動します。そうでなければ、ロボットはまずP2に移動し、その後P1に移動します。

命令の折りたたみ

説明：ネストされたブロックを折りたたみ表示することができます。プログラムをより美しく、読みやすくするためだけに、制御するのではありません。ブロックの右下のダブルアローをクリックすると折りたたみ状態を切り替えることができます。

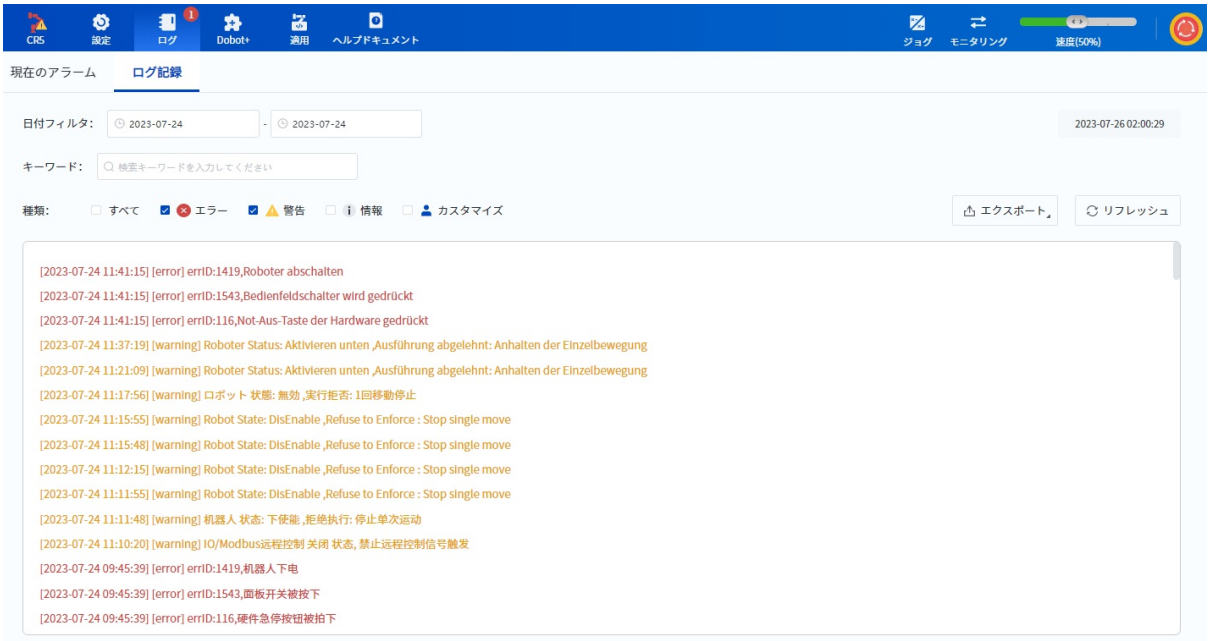
パラメータ：折りたたまれたブロックを説明します。間接的で直感的な名前を付けることをお勧めします。

一時停止

説明：プログラムはこのブロックまで実行されると自動的に停止し、制御ソフトや遠隔操作で実行を継続する必要があります。

カスタムログ

説明：カスタムレベルのログ情報を出力します。制御ソフトウェアログページに表示し、エクスポートすることができます。



衝突検出機能の設定

説明：衝突検出機能を設定します。このブロックで設定された衝突検出レベルは、プロジェクトの実行中にのみ有効になり、プロジェクトが停止すると変更前の値に戻ります。

パラメータ：衝突検出機能の感度を選択するには、オフまたはレベル1～5を選択できます。レベルの数字が大きいほど、衝突検出が敏感になります。

衝突時の後退距離の設定

説明：ロボットが衝突を検出してから元のルートに戻るまでの距離を設定します。このブロックで設定した値は、現在のプロジェクトが実行中にのみ有効で、プロジェクトが停止すると元の値に戻ります。

パラメータ：衝突後退の距離を設定します。取りうる値の範囲：[0,50]、単位：mm

ユーザ座標系の修正

説明：指定したユーザ座標系を修正します。この変更は現在のプロジェクトの実行中にのみ有効になり、プロジェクトが停止すると座標系は変更前の値に戻ります。

パラメータ：

- 変更するユーザ座標系の番号を指定します。
- 変更したユーザ座標系パラメータを指定します。

ツール座標系の修正

説明：指定したツール座標系を修正します。この変更は現在のプロジェクトの実行中にのみ有効になり、プロジェクトが停止すると座標系は変更前の値に戻ります。

パラメータ：

- 変更するツール座標系の番号を指定します。
- 変更したツール座標系パラメータを指定します。

ユーザ座標系の計算と更新

説明：指定されたユーザ座標系の計算と更新を行います。この変更は現在のプロジェクトの実行中にのみ有効になり、プロジェクトが停止すると座標系は変更前の値に戻ります。

パラメータ：

- 計算基準として使用するユーザ座標系の番号を指定します。ユーザ座標系0の初期値はベース座標系です。
- 計算方向を指定します。
  - 左からかける：前のパラメータで指定した座標系がベース座標系に沿って偏向します。
  - 右からかける：前のパラメータで指定した座標系が自分自身に沿って偏向します。
- 座標系のオフセット値を指定します。
- システムは上述のパラメータを使用して新しいユーザ座標系を計算し、ユーザは計算結果をどのユーザ座標系に更新するかを指定する必要があります。

例：

- ユーザ座標系の計算**1** 左からかけるオフセット値：X **10** Y **10** Z **10** RX **10** RY **10** RZ **10** 結果更新先 **1**

上記の命令は、初期姿勢がユーザ座標系1と同じ座標系が、ベース座標系に沿って平行移動{x=10, y=10, z=10}および回転{rx=10, ry=10, rz=10}した後、得られた新しい座標系をユーザ座標系1に割り当てることを示しています。

- ユーザ座標系の計算**1** 右からかけるオフセット値：X **10** Y **10** Z **10** RX **10** RY **10** RZ **10** 結果更新先 **1**

  上記の命令は、初期姿勢がユーザ座標系1と同じ座標系が、ユーザ座標系1に沿って{x=10, y=10, z=10}移動し、{rx=10, ry=10, rz=10}回転した後に得られる新しい座標系をユーザ座標系1に割り当てることを示しています。

ツール座標系を計算し更新する

説明：指定されたツール座標系を計算し更新します。この変更は現在のプロジェクトの実行中にのみ有効になり、プロジェクトが停止すると座標系は変更前の値に戻ります。

パラメータ：

- 計算の基準となるツール座標系の番号を指定します。ツール座標系0の初期値はフランジ座標系です。
- 計算方向を指定します。
  - 左からかける：前のパラメータで指定した座標系がフランジ座標系に沿って偏移することを示します。
  - 右からかける：前のパラメータで指定した座標系が自分自身に沿って偏向します。
- 座標系のオフセット値を指定します。
- システムは上記のパラメータを使用して新しいツール座標系を計算し、ユーザは計算結果をどのツール座標系に更新するかを指定する必要があります。

例：

- ツール座標系を計算する**1** 左からかけるオフセット値：X**10** Y **10** Z **10** RX **10** RY **10** RZ **10**結果更新先**1**

  上記の命令は、初期姿勢がツール座標系1と同じ座標系が、フランジ座標系に沿って{x=10, y=10, z=10}移動し、{rx=10, ry=10, rz=10}回転した後に得られる新しい座標系をツール座標系1に割り当てることを示しています。

- ツール座標系を計算する**1** 右からかけるオフセット値：X**10** Y **10** Z **10** RX **10** RY **10** RZ **10**結果更新先**1**

  上記の命令は、初期姿勢がツール座標系1と同じ座標系が、ツール座標系1に沿って{x=10, y=10, z=10}移動し、{rx=10, ry=10, rz=10}回転した後に得られる新しい座標系をツール座標系1に割り当てることを示しています。

ユーザ座標系を設定する

説明：現在のプロジェクトで使用するユーザ座標系を設定します。プロジェクトが停止すると元の値に戻ります。

パラメータ：ユーザ座標系のインデックス。

ツール座標系を設定する

説明：現在のプロジェクトで使用するツール座標系を設定します。プロジェクトが停止すると元の値に戻ります。

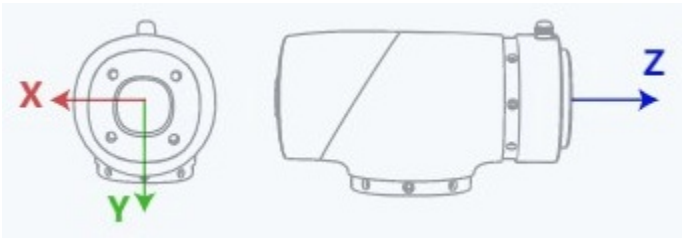
パラメータ：ツール座標系のインデックス。

負荷パラメータを設定する

説明：エンド負荷の重量と偏心座標を設定します。プロジェクトが停止すると元の値に戻ります。

パラメータ：

- 現在の負荷の重量を入力します。単位：kg
- 現在の負荷の偏心座標を入力します。座標軸の方向は下図をご参照ください。単位：mm



指定時間を待つ

説明：指定した時間が経過した後に次の命令を実行します。

パラメータ：命令の発行を遅らせる時間。待ち時間の最大値は2147483647msであり、最大値を超えるパラメータを設定すると命令が無効になります。

安全壁のスイッチを設定する

説明：個々の安全壁のスイッチを設定します。プロジェクトが停止すると元の値に戻ります。

パラメータ：

- 安全壁のインデックスを選択します。
- 安全壁のスイッチ状態を選択します。

干渉区域のスイッチを設定する

説明：個々の干渉区域のスイッチを設定します。プロジェクトが停止すると元の値に戻ります。

パラメータ：

- 干渉区域のインデックスを選択します。
- 干渉区域のスイッチ状態を選択します。

システム時間を取得する

説明：システムの現在の時刻を取得します。

戻り値：システムの現在の時間はUnixタイムスタンプで、単位はミリ秒に変換されています。つまり、グリニッジ標準時間1970年1月1日0時から現在までのミリ秒数で、通常は時間差を計算するために使用されます。

例：

取得した値が1686304295963の場合、それは北京時間で2023-06-09 17:51:35（963ミリ秒を加えたもの）に変換されます；

取得した値が1686304421968の場合、それは北京時間で2023-06-09 17:53:41（968ミリ秒を加えたもの）に変換されます；

何度か取得した値を引き算することで時間差を計算することができます。

タイマーを開始する

説明：プロジェクトがこのブロックに到達した時点でタイマーを開始します。ただし、計時結果の取得ブロックに合わせて使用する必要があります。

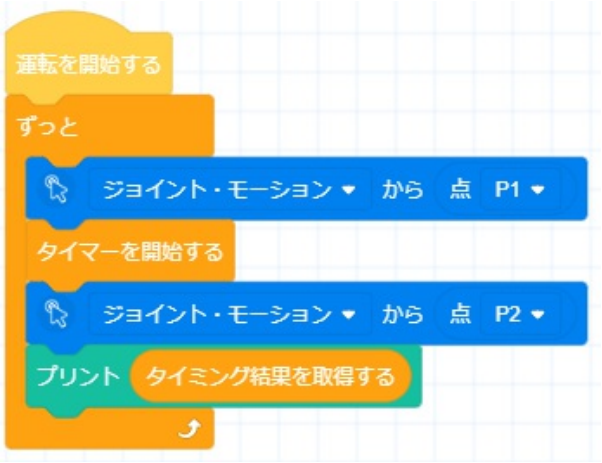
計時結果の取得

説明：計時を終了し、時間差を返します。

戻り値：計時開始から計時終了までの時間差（ミリ秒）。最大で4294967295ms（約49.7日）まで統計が可能で、それを超えると0から再カウントします。

例：

ロボットがP1ジョイントからP2へ移動するのにかかった時間を印刷します。



エンドエフェクタモードの設定

説明：機械アームの末端AI1とAI2インターフェースと485インターフェースが共有されているモデル（CRとCR Aシリーズ）では、このインターフェースを通じて末端の共有端子のモードを設定できます。デフォルトモードは485モードです。

> 説明：
> エンドエフェクタモードの切り替えをサポートしていないロボットでは、このインターフェースの呼び出しは効果がありません。

パラメータ：

- エンドエフェクタモードの選択。485モードとアナログ入力モードをサポートしています。485モードでは、以下の2つのパラメータを設定する必要はありません。
- AI1のアナログ入力モードの選択。0～10V電圧入力モード、電流モード、0～5V電圧入力モードをサポートしています。
- AI2のアナログ入力モードの選択。サポートされているモードはAI1と同じです。

エンドエフェクタの**485**データ形式の設定

説明：エンドツールのRS485インターフェースに対応するデータフォーマットを設定します。

パラメータ：

- RS485インターフェースのボーレートを入力します。
- 奇数または偶数のパリティビットがあるかどうかを選択します。
- ストップビットの長さを選択します。

エンドエフェクタの電源スイッチの設定

説明：エンドエフェクタの電源供給状態を設定します。通常、エンドエフェクタの電源を再起動するために使用されます。例えば、エンドエフェクタのクローを再電源化して初期化します。このインターフェースを連続して呼び出す場合は、少なくとも4ms以上の間隔をお勧めします。

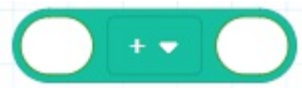
> 説明：
> Magician E6ロボットはこの命令をサポートしていません。呼び出しは効果がありません。

パラメータ：電源スイッチの状態を選択します。

演算ブロックグループ

演算ブロックグループは変数または定数を演算するために使用されます。

四則演算



説明：パラメータを四則演算します。

パラメータ：

- 両側に演算にかかわる変数または定数を記入すると、戻り値が数値である楕円形ブロックを使用するか直接記入できます。
- 中間で四則演算子を選択します。

戻り値：演算の結果の数値。

比較演算

説明：パラメータを比較演算します。

パラメータ：

- 両側に演算にかかわる変数または定数を記入すると、戻り値が数値である楕円形ブロックを使用するか直接記入できます。
- 中間で比較演算子を選択します。

戻り値：比較結果が真の時trueを返し、偽の時falseを返します。

AND演算

説明：パラメータをAND演算します。

パラメータ：両側に演算にかかわる変数を記入し、その他のヘキサゴンブロックを使用します。

戻り値：2つのパラメータが真の時trueを返し、いずれか1つが偽の時falseを返します。

OR演算

説明：パラメータをOR演算します。

パラメータ：両側に演算にかかわる変数を記入し、その他のヘキサゴンブロックを使用します。

戻り値：2つのパラメータのいずれか1つが真の時trueを返し、いずれも偽の時はfalseを返します。

NOT演算

説明：パラメータをNOT演算します。

パラメータ：演算にかかわる変数を記入し、その他のヘキサゴンブロックを使用します。

戻り値：パラメータが真の時falseを返し、偽の時trueを返します。

剰余演算

説明：パラメータを剰余演算します。

パラメータ：両側に演算にかかわる変数または定数を記入すると、戻り値が数値である楕円形ブロックを使用するか直接記入できます。

戻り値：演算の結果の数値。

四捨五入演算

説明：パラメータを四捨五入演算します。

パラメータ：演算にかかわる変数または定数を記入すると、戻り値が数値である楕円形ブロックを使用するか直接記入できます。

戻り値：演算の結果の数値。

単項演算

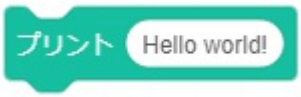
説明：パラメータについてさまざまな単項演算をします。

パラメータ：

- 演算の方式を選択します。
    - 絶対値
    - 切り捨て
    - 切り上げ
    - 平方根
    - sin
    - cos
    - tan
    - asin
    - acos
    - atan
    - ln
    - loh
    - e^
    - 10^
- 演算にかかわる変数または定数を記入すると、戻り値が数値である楕円形ブロックを使用するか直接記入できます。

戻り値：演算の結果の数値。

プリント



説明：パラメータをコンソールに出力して確認し、主にデバッグに使用されます。

パラメータ：

- 同期プリントまたは非同期プリントを選択します。同期プリントは既に発行されたすべてのインストラクションの実行が完了してから情報をプリントし、非同期プリントはプログラムがこのブロックまで実行された時すぐに情報をプリントします。
- コンソールまで出力する変数または定数で、その他の楕円形ブロックを使用するか直接記入できます。

文字ブロックグループ

文字ブロックグループには、文字列とアレイの一般機能が含まれています。

文字列の**n**番目の文字の取得

説明：変数とする文字列のn番目の文字を取得します。

パラメータ：

- 1つ目のパラメータは文字列を記入し、その他の楕円形ブロックを使用するか直接記入できます。
- 2つ目のパラメータは返す文字列の何番目かの文字を指定します。

戻り値：文字列の指定位置の文字。

文字列**1**が文字列**2**を含むか否かの判断

説明：1つ目のパラメータの文字列に2つ目のパラメータの文字列が含まれるか否かを判断します。

パラメータ：判断する2つの文字列には、戻り値が文字列の楕円形ブロックを使用するか直接記入できます。

戻り値：文字列1には文字列2が含まれている場合はtrueを返し、そうでない場合falseを返します。

2つの文字列の連結

説明：2つの文字列を1つの文字列に連結し、2つ目の文字列は1つ目の文字列の後ろにつきます。

パラメータ：連結する2つの文字列は、戻り値が文字列の楕円形ブロックを使用するか直接記入できます。

戻り値：連結後の文字列。

文字列またはアレーの長さの取得

説明：指定する文字列またはアレーの長さを取得します。文字列の長さとはこの文字列にいくつの文字があるかを指し、アレーの長さとはこのアレーにいくつの要素があるかを指します。

パラメータ：長さを計算する文字列またはアレーは、戻り値が文字列またはアレーの楕円形ブロックを使用できます。

戻り値：文字列またはアレーの長さの数値。

2つの文字列の比較

説明：ACSIIコードに基づき、2つの文字列の大きさを比較します。

パラメータ：比較する2つの文字列は、戻り値が文字列の楕円形ブロックを使用するか直接記入できます。

戻り値：文字列1と文字列2が同じ時、0を返し、文字列1が文字列2よりも小さい時、-1を返し、文字列1が文字列2よりも大きい時、1を返します。

アレーの文字列への変換

説明：指定するアレーを文字列に変換し、文字列中のさまざまなアレー要素は指定する区切り記号で区切られます。例えば、アレーが{1,2,3}で、区切り記号が|，の場合、変換後の文字列は「1|2|3」です。

パラメータ：

- 文字列に変換するアレーでは、戻り値がアレーの楕円形ブロックを使用します。
- 変換に使用する区切り記号。

戻り値：変換後の文字列。

文字列のアレーへの変換

説明：指定する文字列をアレーに変換し、指定する区切り記号に基づき文字列を区切ります。例えば、文字列が「1|2|3」で、区切り記号が|，の場合、変換後のアレーは{[1]=1,[2]=2,[3]=3}です。

パラメータ：

- アレーに変換する文字列は、戻り値が文字列の楕円形ブロックを使用するか直接入力できます。
- 変換に使用する区切り記号。

戻り値：変換後のアレー。

アレーのインデックスを指定する要素の取得

説明：指定するアレーにおけるインデックス位置を指定する要素を取得します。インデックスとは要素のアレーにおける位置を表し、例えば、アレー{7,8,9}における8のインデックスは2です。

パラメータ：

- ターゲットアレーは、戻り値がアレーの楕円形ブロックを使用します。
- 指定する要素のインデックス。

戻り値：アレーの位置を指定する要素の値。

アレー内で指定した複数の要素の取得

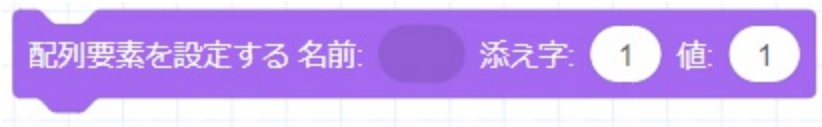
説明：指定するアレー内の複数のインデックス位置を指定する要素を取得します。最初のインデックスと最後のインデックスの範囲内で、ステップ値に基づき要素を取得します。

パラメータ：

- ターゲットアレーは、戻り値がアレーの楕円形ブロックを使用します。
- 最初のインデックスと最後のインデックスにより取得する要素の範囲を指定します。
- ステップ値は要素を取得する頻度を確定するために使用され、1はすべてを取得し、2は要素を1つおきに取得し、以下同様です。

戻り値：指定する元素が構成する新しいアレー。

アレー中の指定された要素の設定



説明：アレー中のインデックス位置を指定する要素の値を設定します。

パラメータ：

- ターゲットアレーは、戻り値がアレーの楕円形ブロックを使用します。
- 指定する要素のインデックス。
- 指定する要素の値。

カスタムブロックグループ

カスタムブロックはカスタムブロックの新規作成と管理、およびグローバル変数の呼び出しに使用されます。

グローバル変数の呼び出し

説明：制御ソフトウェアで設定されるグローバル変数を呼び出します。

パラメータ：グローバル変数の名称を選択します。

戻り値：グローバル変数の値。

グローバル変数の設定

説明：指定された変数を設定します。グローバル変数を設定するブロックとカスタム変数を設定するブロックの外観は同じですが、機能がやや異なることに注意してください。

パラメータ：

- 変更する変数の名称を選択します。
- 変更後の値は直接記入してもその他楕円形のブロックを使用しても構いません。

カスタム変数の新規作成

クリックするとカスタム変数を新規作成できます。変数タイプは数値または文字列を選択でき、変数名称はアルファベットから始まらなければならず、スペースなどの特殊文字を使用できません。少なくとも1つの変数を作成した後、ブロックリストに下記のカスタム変数関連のブロックが現れます。

カスタム数値変数

説明：新規作成したカスタム数値変数で、デフォルト値はnilです。値を割り当てた後に使用することをお勧めします。変数選択のドロップダウンリストにより、現在選択した変数の名称を変更したり、この変数を削除したりできます。

戻り値：変数の値。

カスタム数値の変数の値設定

説明：指定された数値の変数を設定します。グローバル変数を設定するブロックとカスタム変数を設定するブロックの外観は同じですが、機能がやや異なることに注意してください。

パラメータ：

- 変更する変数の名称を選択します。
- 変更後の値は直接記入してもその他楕円形のブロックを使用しても構いません。

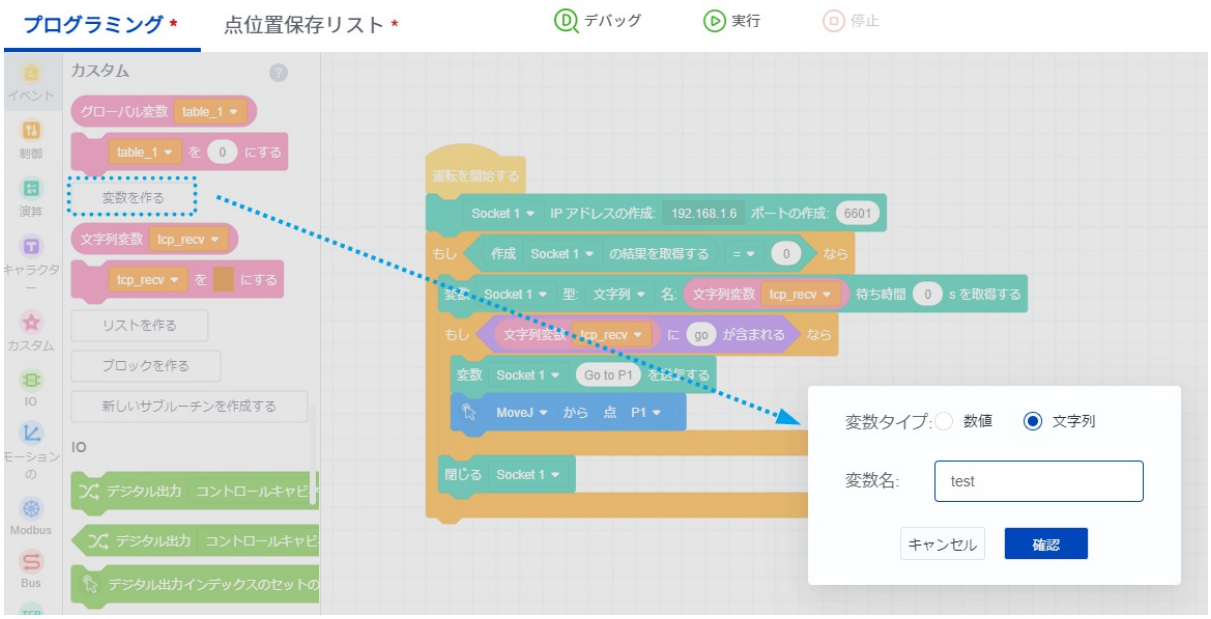
カスタム数値の変数の値増減

説明：指定した数値の変数を、指定した値分で増やします。

パラメータ：

- 変更する変数の名称を選択します。
- 増やす値は直接記入してもその他楕円形のブロックを使用しても構いません。負の数を設定すると、値を減らすことができます。

カスタム文字列変数



説明：新規作成したカスタム文字列変数で、デフォルト値はnilです。値を割り当てた後に使用することをお勧めします。変数選択のドロップダウンリストにより、現在選択した変数の名称を変更したり、この変数を削除したりできます。

戻り値：変数の値。

カスタム文字列の変数の値設定

説明：指定された文字列の変数を設定します。

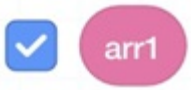
パラメータ：

- 変更する変数の名称を選択します。
- 変更後の値で、文字列を直接入力します。

アレーの作成

クリックするとカスタムアレーを新規作成できます。アレー名称はアルファベットから始まらなければならず、スペースなどの特殊文字を使用できません。少なくとも1つのアレーを作成した後、ブロックリストに下記のアレー関連のブロックが現れます。

カスタムアレー



説明：新規作成したカスタムアレーで、デフォルトは空のアレーです。値を割り当てた後に使用することをお勧めします。ブロックリストでブロックを右クリック（PC端末）/長押し（モバイル端末）すると、アレーの名称を変更したりアレーを削除したりできます。その他のアレーブロック中のアレー選択ドロップダウンリストにより、現在選択したアレーの名称を変更したり、このアレーを削除したりできます。アレーブロック前のチェックボックスは現在無効です。無視してください。

戻り値：アレーの値。

変数のアレーへの追加

説明：変数を指定されたアレーに追加します。新たに追加する変数は、アレーの最後の1アイテムになります。

パラメータ：

- 追加する変数で、直接記入してもその他楕円形のブロックを使用しても構いません。
- 変更するアレーを選択します。

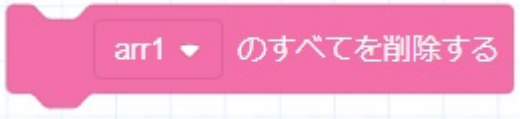
アレーの指定されたアイテムの削除

説明：指定されたアレーの指定されたアイテムを削除します。

パラメータ：

- 変更するアレーを選択します。
- 削除するアイテムの番号で、直接記入してもその他戻り値が数値の楕円形のブロックを使用しても構いません。

アレーのすべてのアイテムの削除



説明：指定されたアレーのすべてのアイテムを削除します。

パラメータ：変更するアレーを選択します。

アレーへのアイテムの挿入

説明：アレー中の指定された位置に変数を挿入します。

パラメータ：

- 変更するアレーを選択します。
- 挿入の位置で、直接記入してもその他戻り値が数値の楕円形のブロックを使用しても構いません。
- 追加する変数で、直接記入してもその他楕円形のブロックを使用しても構いません。

アレーにおける指定されたアイテムの置換

説明：アレー中の指定されたアイテムを指定された変数に置換します。

パラメータ：

- 変更するアレーを選択します。
- 置換するアイテムの番号で、直接記入してもその他戻り値が数値の楕円形のブロックを使用しても構いません。
- 置換後の変数は、直接記入してもその他楕円形のブロックを使用しても構いません。

アレーにおける指定されたアイテムの取得

説明：アレー中の指定されたアイテムの値を取得します。

パラメータ：

- アイテムを取得するアレーを選択します。
- 取得するアイテムの番号で、直接記入してもその他戻り値が数値の楕円形のブロックを使用しても構いません。

戻り値：指定されたアイテムの値。

アレーにおける総アイテム数の取得

説明：アレー中のアイテムの総数を取得します。

パラメータ：取得するアイテムのアレーを選択します。

戻り値：指定するアレーのアイテム総数。

1つの関数の新規作成

クリックすると1つの関数を新規作成できます。関数は1つの固定されたプログラムセグメントで、特定の一般機能を実装したブロックグループを1つの関数と定義できます。その後、この機能を使用するたびに、この関数を呼び出すだけでよく、同じブロックグループを繰り返し構築する必要がありません。関数を新規作成するには、この関数を宣言および定義する必要があります。関数が正常に新規作成できた後、ブロックリストに対応する関数ブロックが現れます。

1. 関数の宣言

この画面では、関数の名称を定義し、（パラメータの）タイプ、数量と名称を入力する必要があります。関数とパラメータの名称は、スペースなど特殊文字を含めることができません。さらに関数にタグを追加することができ、タグは注釈として使用され、関数または入力に補足説明を行うことができます。

1. 関数の定義

関数の宣言が完了した後、プログラミングエリア内にこの関数の定義ヘッダーブロックが現れます。

このブロック下にブロックを接続してプログラミングを行い、この関数の機能を定義する必要があります。

定義ヘッダーにおける入力は下のブロックにドラッグして使用でき、これは実際にこの関数を呼び出す時の入力をパラメータとして使用することを表します。

カスタム関数

説明：ユーザーがカスタマイズする関数ブロックで、名称と入力パラメータはユーザーによりカスタマイズし、定義済みの関数を呼び出すために使用されます。ブロックリストでブロックを右クリック（PC端末）/長押し（モバイル端末）すると、この関数の宣言を変更できます。この関数を削除する必要がある場合、この関数の定義ヘッダーブロックを削除してください。

サブルーチンの新規作成

クリックすると、サブルーチンを新規作成できます。グラフィカルプログラミングはサブルーチンのネストおよび呼び出しに対応し、サブルーチンはグラフィカルプログラミングとスクリプトプログラミングに対応し、最大で2層ネストします。サブルーチンが正常に新規作成できた後、ブロックリストに対応するサブルーチンブロックが現れます。

- ブロックプログラミングを選択後、ページがサブルーチンブロックプログラミングページに変わると、サブルーチンの説明を設定し、サブルーチンを作成できます。

- スクリプトプログラミングを選択後、サブルーチンのスクリプトプログラミングウィンドウが表示されると、サブルーチンの説明を設定し、サブルーチンを作成できます。

サブルーチン

- グラフィカルプログラミングのサブルーチン

- スクリプトプログラミングのサブルーチン

説明：サブルーチンブロックで、ユーザーがサブルーチンを作成する時に説明が定義され、保存済みのサブルーチンを呼び出すために使用されます。ブロックリストでブロックを右クリック（PC端末）/長押し（モバイル端末）すると、このサブルーチンを修正または削除できます。

IOブロックグループ

IOブロックグループは、ロボットのIO端子の入出力を管理するためのものです。入出力ポートの値の範囲は、ロボットの対応する端子の数によって決まります。詳細は対応するロボットのハードウェアマニュアルを参照してください。

デジタル出力の設定

説明：指定したDOのオン/オフを設定します。

パラメータ：

- DO端子の位置を選選択します。これには、制御キャビネットと端子が含まれます。
- DO端子の番号を選択します。
- 出力する状態を選択します（オンまたはオフ）。

デジタル出力状態の判定

説明：指定したDOの現在の状態が条件を満たしているかどうかを判断します。

パラメータ：

- DO端子の位置を選選択します。これには、制御キャビネットと端子が含まれます。
- DO端子の番号を選択します。
- 真と判断する状態を選択します。

戻り値：指定したDOの現在の状態が条件を満たしている場合はtrueを返し、そうでない場合はfalseを返します。

一連のデジタル出力の設定

説明：DOセットを設定します。

パラメータ：ブロックをプログラミングエリアにドラッグしてから、DOと状態を設定をクリックします。

- + あるいは - により、設定するDOの数を増減できます。
- 左側のドロップダウンメニューからDO端子の番号を選択します。
- 右側のドロップダウンメニューから出力する状態（オンまたはオフ）を選択します。

一連の数値出力を待つ

説明：一連のDOの状態がすべて条件を満たした後に、次の実行に進みます。

パラメータ：

- 待機する状態(オンまたはオフ)を選択します。
- タイムアウト時間を待ちます。0に設定すると、条件が満たされるまで待ちます。
- ブロックをプログラミングエリアにドラッグした後、待機するDOを設定するためにクリックします。

- + あるいは - 待機するDOの数を増減ができます。
- ドロップダウンメニューからDO端子の番号を選択します。

数字入力を待つ

説明：指定したDIが条件を満たすか、タイムアウトを待ってから、後続のブロック命令を実行します。

パラメータ：

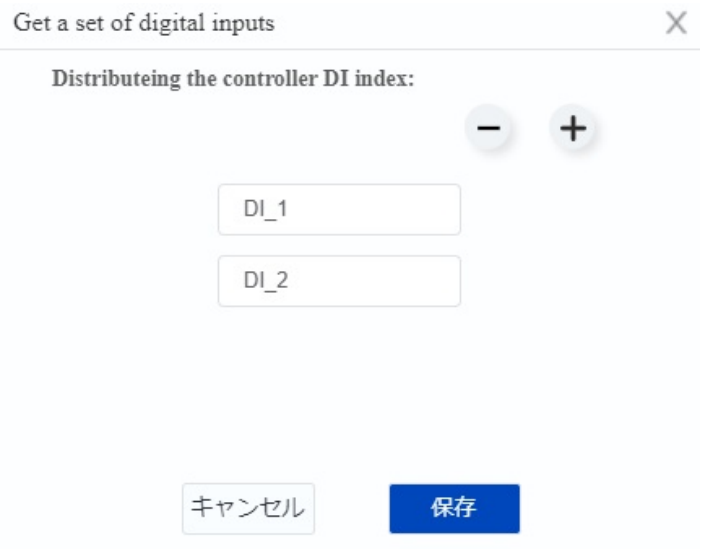
- DI端子の位置を選択します。これには、制御キャビネットと端子が含まれます。
- DI端子の番号を選択します。
- 待機する状態(オンまたはオフ)を選択します。
- タイムアウト時間を待ちます。0に設定すると、条件が満たされるまで待ちます。

一連の数字入力を待つ

説明：一連のDIの状態がすべて条件を満たした後に、次の実行に進みます。

パラメータ：

- 待機する状態(オンまたはオフ)を選択します。
- タイムアウト時間を待ちます。0に設定すると、条件が満たされるまで待ちます。
- ブロックをプログラミングエリアにドラッグした後、待機するDIを設定するためにクリックします。



- + あるいは - 待機するDIの数を増減ができます。
- ドロップダウンメニューからDI端子の番号を選択します。

アナログ出力を設定する

説明：指定したアナログ出力の値を設定します。

パラメータ：

- アナログ出力端子の番号を選択します。
- 出力する値は、直接入力するか、戻り値が数値の他の楕円形ブロックを使用します。

アナログ出力を取得する

説明：指定されたアナログ出力の値を取得します。

パラメータ：アナログ出力端子の番号を選択します。

デジタル入力状態を判断する

説明：指定したDIの現在の状態が条件を満たすか否かを判定します。

パラメータ：

- DI端子の位置を選択します。これには、制御キャビネットと端子が含まれます。
- DI端子の番号を選択します。
- 真と判断する状態を選択します。

戻り値：指定したDIの現在の状態が条件を満たす場合はtrueを返し、それ以外の場合はfalseを返します。

アナログ入力を取得する

説明：指定したアナログ入力の値を取得します。

パラメータ：

- アナログ入力端子の位置を選択します。それには制御キャビネットと端子が含まれます。
- アナログ入力端子の番号を選択します。

戻り値：アナログ入力の値を指定します。

移動ブロックグループ

移動ブロックグループは、ロボット移動の制御および移動に関連する設定を行うために使用します。

点位置パラメータは、該プロジェクトの点位置保存リストで追加した後にここで選択することができます。移動を制御するブロックの中では、デフォルトの変数ブロックをドラッグアウトし、他の戻り値が点位置である楕円形ブロックで入れ替えることができます。

目標点への移動

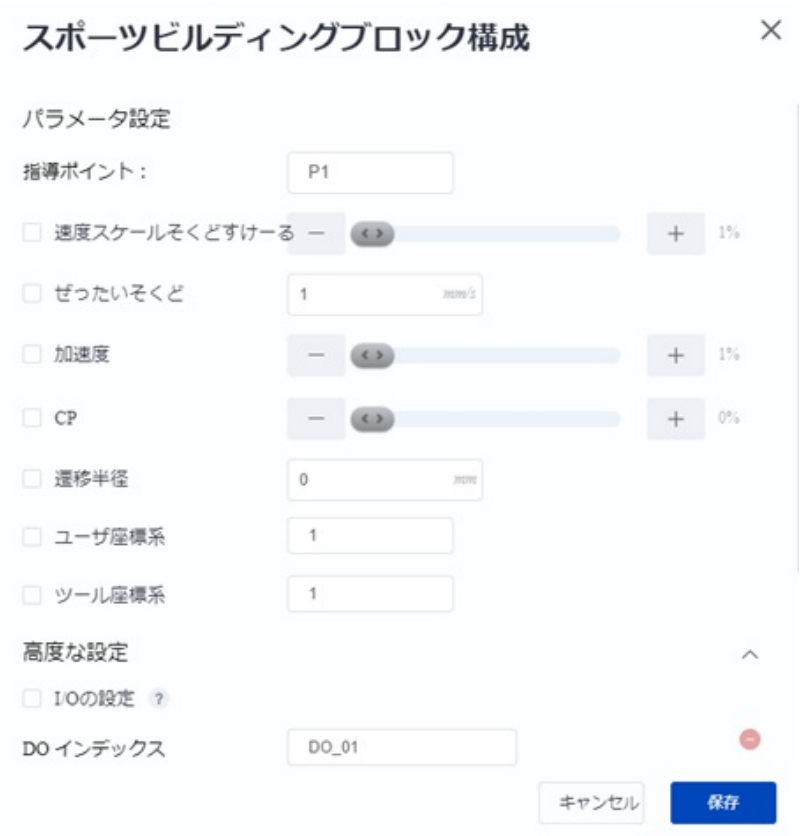
説明：ロボットが現在の位置から指定された点まで移動することを制御します。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。

パラメータ：

- 移動方式を選択します。関節移動および直線移動に対応します。
- 目標点。

詳細設定

関節動作の設定可能なパラメータは直線移動より少なく、以下の図は直線移動のポップアップを例にしています。詳細な違いは以下の説明を参照してください。



- ティーチング点：移動標的点位置

以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は全般的な説明を参照してください。

- 速度比率(V)：移動速度比率。値の範囲：1~100。
- 絶対速度(Speed)：直線移動のみに対応します。移動の絶対速度値。Vとは排他的です。
- 加速度(Accel)：移動加速度比率。値範囲：1~100。
- 連続経路制御(CP)：連続経路制御比率。値の範囲：0~100。
- 移動半径(R)：直線移動のみに対応します。移動曲線の半径。CPとは排他的です。
- ユーザ座標系：点パラメータのユーザ座標系インデックス。
- 工具座標系：点パラメータの工具座標系インデックス。
- 過程I/O設定：

ロボットが指定した距離または百分率までに移動すると、指定したDOの状態を設定するために使用します。

距離が正であるとき、開始点までの距離を表し、距離が負であるとき、標的点までの距離を表します。連続経路制御が設定されている場合、ロボットは目標点に到達せず、DO出力のタイミングに影響を与える可能性があります。

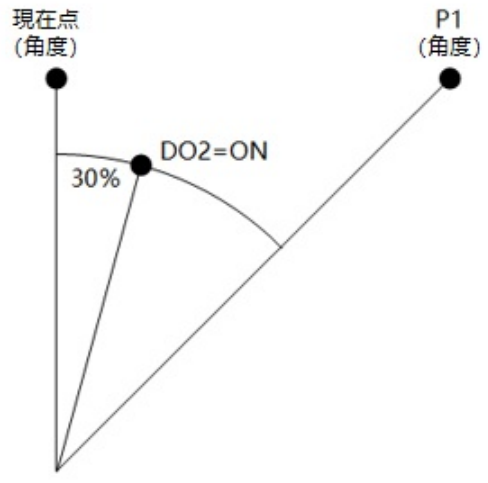
移動方式が関節移動の場合、距離は各関節角の合成角ベクトルを表し、計算は複雑になります。そのため、百分率モードを使用することを推奨し、効果はより直感的です。

下の + をクリックすると、過程I/Oを追加できます。右側の - をクリックすると、対応する過程I/Oを削除できます。

以下は、いくつかの過程IO設定の例です。

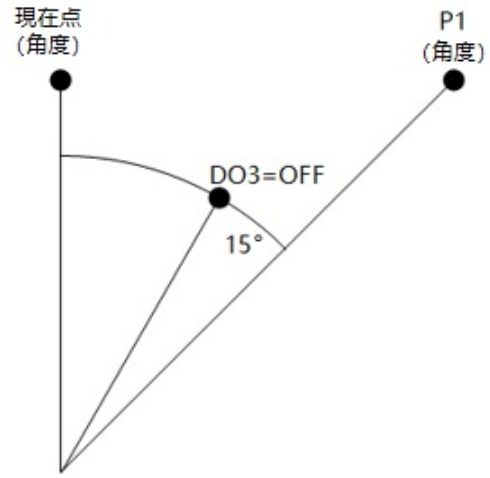
例**1**：P1まで関節移動、DOインデックス：DO_02、DO状態：ON、トリガモード：百分率、距離：30%

開始点から30%の位置に関節移動すると、DO2がONになるように設定します。



例**2**：P1まで関節移動、DOインデックス：DO_03、DO状態：OFF、トリガモード：距離、距離：-15°

終点まであと15°の位置に関節移動すると、DO3がOFFになるように設定します。



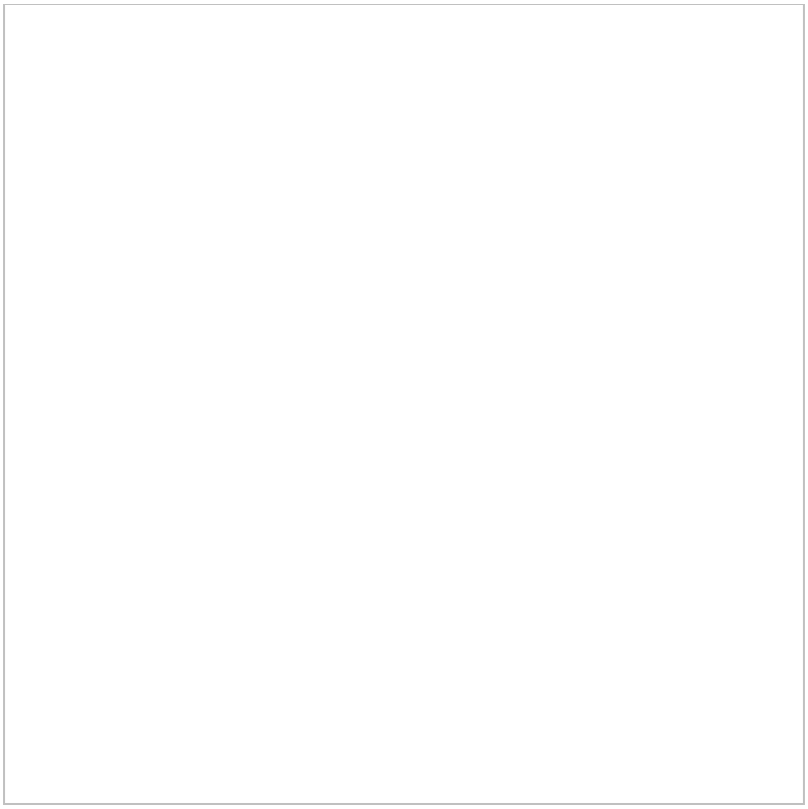
例**3**：P1まで直線移動、DOインデックス：DO_02、DO状態：ON、トリガモード：百分率、距離：30%

開始点から30%の位置に直線移動すると、DO2がONになるように設定します。

例**4**：P1まで直線移動、DOインデックス：DO_03、DO状態：OFF、トリガモード：距離、距離：-15mm

終点まであと15mmの位置に直線移動すると、DO3がOFFになるように設定します。

- 停止条件：



移動の停止条件を設定し、条件が満たされるとロボットは現在の移動を終了し、次の命令を直接実行します。

例**1**：「DI1==ONの場合」の1行の条件だけ設定します。これは、この移動命令の実行中、DI1がONであることを検出した場合に現在の移動をスキップすることを意味します。

例**2**：「DI1==ON、DI2～=ONの場合」の2行の条件を設定します。これは、この移動命令の実行中、DI1がONでDI2がONでないであることを検出した場合に現在の移動をスキップすることを意味します。

例**3**：「DI1==ON、またはグローバル変数var<=10の場合」の2行の条件を設定します。これは、この移動命令の実行中に、DI1がONまたはグローバル変数varが10以下であることを検出した場合に現在の移動をスキップすることを意味します。

座標系に沿ったオフセット移動

説明：ロボットを制御して、現在位置から座標系に沿って指定した距離だけオフセットさせます。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。

パラメータ：

- 移動方式を選択します。相対関節移動と相対直線移動に対応します。

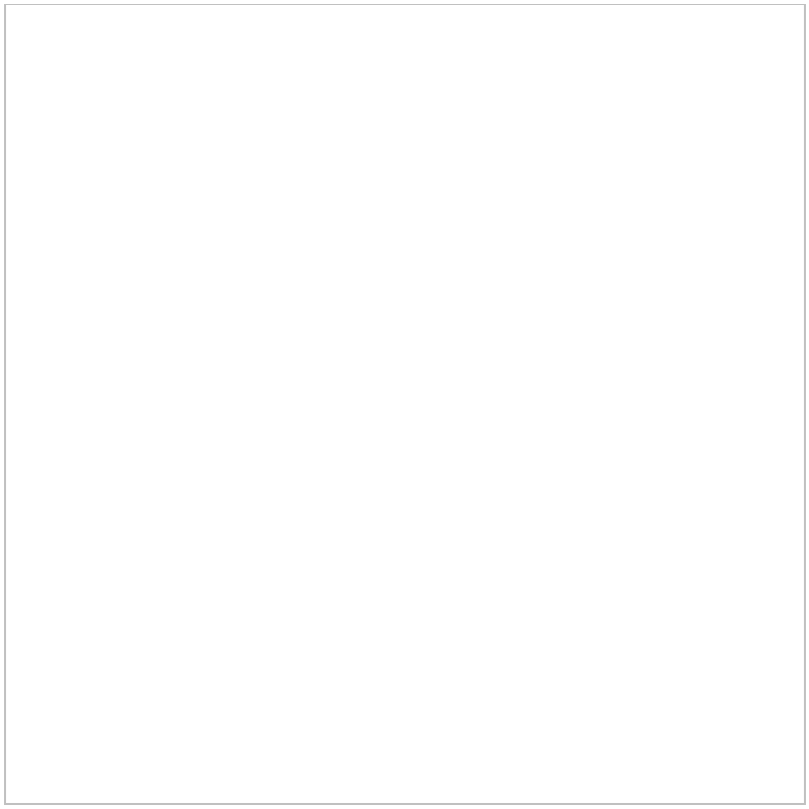
- ロボット アームがユーザ座標系か、または工具座標系に沿ってオフセットするように指定します。
- 指定した座標系での偏移量。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。

詳細設定

関節動作の設定可能なパラメータは直線移動より少なく、以下の図は直線移動のポップアップを例にしています。詳細な違いは以下の説明を参照してください。

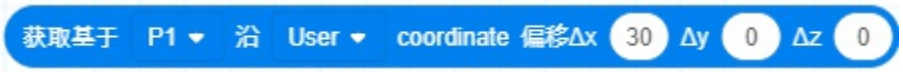
以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は全般的な説明を参照してください。

- 速度比率(V)：移動速度比率。値の範囲：1~100。
- 絶対速度(Speed)：直線移動のみに対応します。移動の絶対速度値。Vとは排他的です。
- 加速度(Accel)：移動加速度比率。値範囲：1~100。
- 連続経路制御(CP)：連続経路制御比率。値の範囲：0~100。
- 移動半径(R)：直線移動のみに対応します。移動曲線の半径。CPとは排他的です。
- ユーザ座標系：オフセット時に参照したユーザ座標系のインデックス。
- 工具座標系：オフセット時に参照した工具座標系のインデックス。
- 停止条件：



移動の停止条件を設定し、条件が満たされるとロボットは現在の移動を終了し、次の命令を直接実行します。

座標系に沿ったオフセット後の点の取得



説明：指定した点を指定した座標系に沿って指定した距離だけオフセットした後、オフセット後の新しい位置を返します。

パラメータ：

- オフセットする点を選択します。
- にオフセットがティーチング点のどの座標系に基づくかを選択します。ユーザ座標系または工具座標系を選択できます。
- 各方向のオフセット距離を入力します。

戻り値：オフセット後の新しい位置。

関節オフセット移動

説明：ロボット関節が現在の位置から指定するオフセット量まで移動することを制御します。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。

パラメータ：各関節のオフセット量（度）。

詳細設定

以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は全般的な説明を参照してください。

- 速度比率(V)：移動速度比率。値の範囲：1~100。
- 加速度(Accel)：移動加速度比率。値範囲：1~100。
- 連続経路制御(CP)：連続経路制御比率。値の範囲：0~100。
- ユーザ座標系/工具座標系：このブロックのこれら2つのパラメータは無効です。

関節オフセット後の点の取得

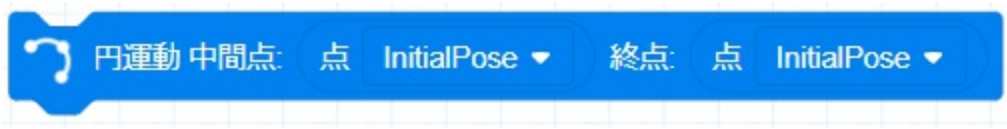
説明：指定した点を各関節に沿って指定した角度だけオフセットした後、オフセット後の新しい位置を返します。

パラメータ：

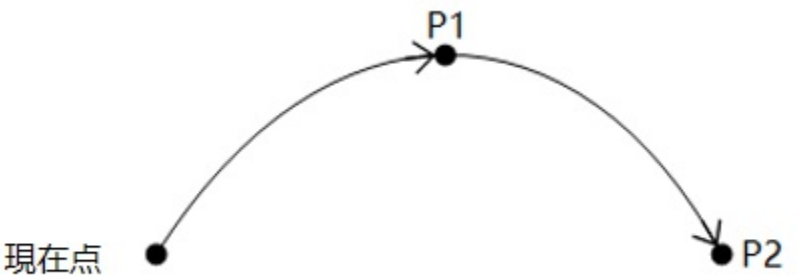
- オフセットする点を選択します。
- 各関節のオフセット角度を入力します。

戻り値：オフセット後の新しい位置。

円弧移動の実行



説明：ロボットが現在の位置から円弧補間の方式で、デカルト座標系の指定された点まで移動することを制御します。現在の位置の座標は、中間点または終了点で決まる直線上にないようにしてください。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。



動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

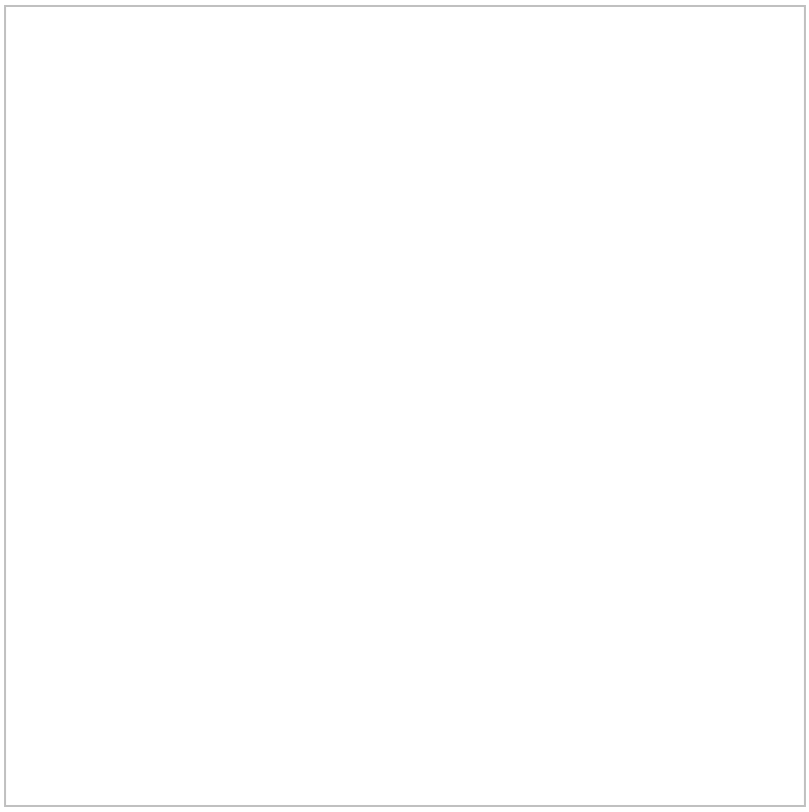
パラメータ：

- 中間点とは、円弧を決める中間点のことです。
- 終了点は標的点です。

詳細設定

以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は全般的な説明を参照してください。

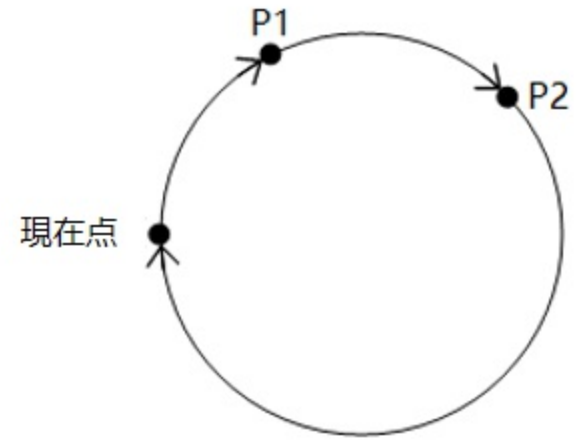
- 速度比率(V)：移動速度比率。値の範囲：1~100。
- 絶対速度(Speed)：移動の絶対速度値。Vとは排他的です。
- 加速度(Accel)：移動加速度比率。値範囲：1~100。
- 連続経路制御(CP)：連続経路制御比率。値の範囲：0~100。
- 移動半径(R)：移動曲線の半径。CPとは排他的です。
- ユーザ座標系：点パラメータのユーザ座標系インデックス。
- 工具座標系：点パラメータの工具座標系インデックス。
- 停止条件：



移動の停止条件を設定し、条件が満たされるとロボットは現在の移動を終了し、次の命令を直接実行します。

完全円形移動の実行

説明：ロボットが現在の位置から完全円形補間移動を行い、指定したターン数で移動した後、現在の位置に戻ることを制御します。現在の位置の座標は、中間点または終了点で決まる直線上にないようにしてください。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。



動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

パラメータ：

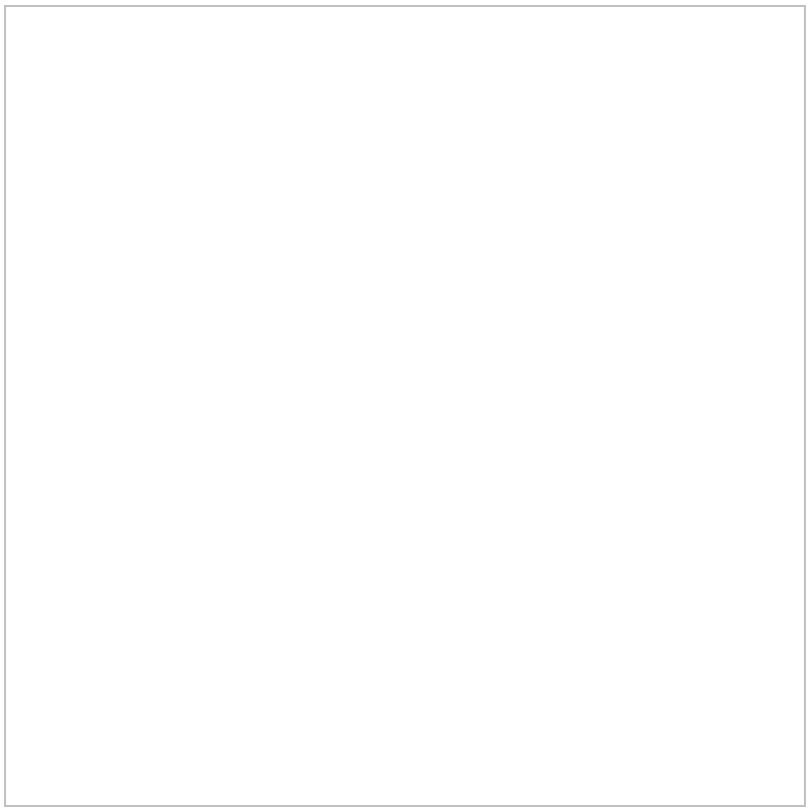
- 中間点は、全円を決めるために使用される位置1のことです。
- 終了点は、全円を決めるために使用される位置2のことです。
- 全円移動を行う周回数を入力します。値範囲：1~999。

詳細設定

以下のパラメータは選択した後にのみ有効となります。パラメータの詳細な説明は全般的な説明を参照してください。

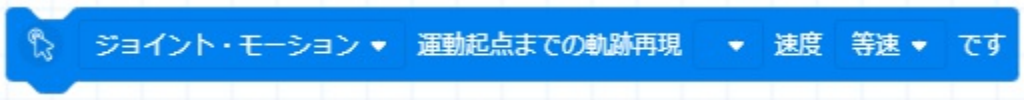
- 速度比率(V)：移動速度比率。値の範囲：1~100。
- 絶対速度(Speed)：移動の絶対速度値。Vとは排他的です。
- 加速度(Accel)：移動加速度比率。値範囲：1~100。

- 連続経路制御(CP)：連続経路制御比率。値の範囲：0~100。
- 移動半径(R)：移動曲線の半径。CPとは排他的です。
- ユーザ座標系：点パラメータのユーザ座標系インデックス。
- 工具座標系：点パラメータの工具座標系インデックス。
- 停止条件：



移動の停止条件を設定し、条件が満たされるとロボットは現在の移動を終了し、次の命令を直接実行します。

軌跡の再現



説明：ロボットが軌跡の開始点まで移動した後に軌跡再現を行います。再現した軌跡のファイルは、軌跡再現プロセス中で記録します。ブロックをプログラミングエリア内にドラッグすると、クリックして詳細設定を行うことができます。

パラメータ：

- 軌跡の開始点まで移動した際の移動方式を選択します。関節移動および直線移動に対応します。
- 再現する軌跡ファイルを選択します。

- 再現時の速度を選択します。
  - 等速。ロボットは、グローバル速度に従って軌跡を等速で再現します。
  - 0.25倍速。軌跡に記録した原速度に基づいて比例スケーリング処理を行います。このとき、ロボットの移動速度はグローバル速度から影響を受けません。以下同様です。
  - 0.5倍速
  - 1倍速
  - 2倍速

詳細設定

以下のパラメータは選択した後にのみ有効となります。

- 再現間隔：軌跡点位置サンプリング間隔。すなわち、軌跡ファイル生成時、隣接する2つの点位置間のサンプリング時間差。値の範囲：[8, 1000]、単位はms、デフォルト値は50です（コントローラが軌跡ファイルを記録する場合のサンプリング間隔）。
- リップル係数：該パラメータが小さければ小さいほど、再現される軌跡曲線がより平滑になりますが、原軌跡に対する変形も著しく大きくなります。原軌跡の平滑度に基づいて適切なフィルタ係数を設定してください。値の範囲：（0,1]、1はフィルタリング終了を示し、デフォルト値は0.2です。
- ユーザ座標系：軌跡点位置に対応するユーザ座標系インデックスを指定します。指定していないとき、軌跡ファイルに記録されたユーザ座標系インデックスを使用します。
- ツール座標系：軌跡点位置に対応するツール座標系インデックスを指定します。指定していないとき、軌跡ファイルに記録されたツール座標系インデックスを使用します。

連続経路制御の比率の設定

説明：移動中の連続経路制御の比率を設定します。設定は現在のプロジェクトの実行中にのみ有効です。連続経路制御の詳細な説明は全般的な説明を参照してください。

パラメータ：連続経路制御率で、値範囲は0~100です。

関節速度比率の設定

説明：関節が移動する時の速度比率を設定します。設定は、現在のプロジェクトの実行中にのみ有効です。速度計算の詳細な説明は全般的な説明を参照してください。

パラメータ：関節の速度比率で、値範囲は0~100です。

関節加速度比率の設定

説明：関節移動の加速度比率を設定します。設定は、現在のプロジェクトの実行中にのみ有効です。加速度計算の詳細な説明は全般的な説明を参照してください。

パラメータ：関節加速度比率で、値範囲は0~100です。

直線速度比率の設定

説明：直線および円弧移動の速度比率を設定します。設定は、現在のプロジェクトの実行中にのみ有効です。速度計算の詳細な説明は全般的な説明を参照してください。

パラメータ：直線および円弧の速度比率で、値範囲は0~100です。

直線加速度比率の設定

説明：直線および円弧移動の加速度比率を設定します。設定は、現在のプロジェクトの実行中にのみ有効です。加速度計算の詳細な説明は全般的な説明を参照してください。

パラメータ：直線および円弧の加速度比率で、値範囲は0~100です。

グローバル速度比率の設定

説明：ロボットの移動のグローバル速度比率を設定します。設定は、現在のプロジェクトの実行中にのみ有効です。速度計算の詳細な説明は全般的な説明を参照してください。

パラメータ：グローバル速度比率、値範囲は0~100です。

指定した点の座標値の変更

説明：指定した点の指定デカルト座標の次元の値を変更します。

パラメータ：

- 変更する点を選択します。
- 変更対象の座標次元を選択します。
- 変更後の値を選択します。

指定した点の座標値の取得

説明：指定した点の座標値を取得します。

パラメータ：取得する座標値の点を選択します。

戻り値：指定した点の座標値。

移動可能性の確認

説明：ロボットが現在点から指定した移動方式で指定した点まで移動する可能性を確認します。システムは全体の運動経路を計算し、経路中に到達不能な点がないかを確認します。

パラメータ：

- 移動方式を選択します。関節移動および直線移動に対応します。
- 目標点を選択します。

戻り値：チェック結果。

- 0：エラーなし
- 16：終点がショルダーの特異点に近接している
- 17：終点逆計算が解なし
- 18：終点逆計算が位置制限される
- 22：ジェスチャー切り替えエラー
- 26：終点が腕部の特異点に近接している
- 27：終点が肘部の特異点に近接している
- 29：速度パラメータエラー
- 32：軌跡にショルダーの特異点がある
- 33：軌跡に逆計算解なし点が存在する
- 34：軌跡に逆計算位置制限点が存在する
- 35：軌跡に腕部の特異点がある
- 36：軌跡に肘部の特異点がある
- 37：軌跡に関節ジャンピング点が存在する

指定した点の指定座標次元の値の取得

説明：指定した点の指定デカルト座標の次元の値を取得します。

パラメータ：

- 座標値を取得する点を選択します。
- 取得対象の座標次元を選択します

戻り値：指定した点の指定デカルト座標の次元の値。

指定した点の指定関節の角度の取得

説明：指定した点の指定関節の角度を取得します。

パラメータ：

- 関節角を取得する点を選択します。
- 角度を取得する関節を選択します。

戻り値：指定した点の指定関節の角度。

姿勢に対する関節角度の順解法

説明：ロボットの各関節角度を与え、ロボット末端の与えられたデカルト座標系中の姿勢を計算します。

パラメータ：

- 点を選択すると、この点の関節角度が順解の計算に使用されます。
- ユーザ座標系のインデックス。
- 工具座標系のインデックス。

戻り値：順解法で得られた姿勢変数。形式は `{pose = {x, y, z, rx, ry, rz} }`

関節角度に対する姿勢の逆解法

説明：ロボット末端の与えられたデカルト座標系中の座標値が与えられ、ロボットの各関節の角度を計算します。デカルト座標はTCPの空間座標と傾斜角のみを定義するので、ロボットは多種の異なるポーズを通じて同一のポーズに達し、1つの姿勢が複数の関節角度に対応でき、このブロックがロボットの現在の姿勢に最も近い関節角度に戻ることを意味します。

パラメータ：

- 点を選択すると、この点の姿勢が逆計算に使用されます。
- ユーザ座標系のインデックス。
- 工具座標系のインデックス。

戻り値：2つの変数が返され、表示用に印刷できます。最初の変数はエラーコードで、0は逆解が成功したことを意味し、-1は逆解が失敗した (解なし) ことを意味し、2番目の変数は逆解から得られた関節角度であり、形式は `{joint = {j1, j2, j3, j4, j5, j6} }` 。逆解が失敗すると、j1~j6はすべて0になります。

エンコーダーの値の取得

説明：ABZエンコーダーの現在の値を取得します。

戻り値：エンコーダーの現在の値。

Modbusブロック

Modbusブロックは Modbus通信関連の操作を行うためのものです。対応するDEMOを参考に、関連命令の使用を迅速に体験できます。

各種レジスタに対応するModbus機能コードは、標準のModbusプロトコルに従います。

レジスタタイプ	読み取りレジスタ	単一レジスタの書き込み	複数レジスタの書き込み
コイルレジスタ	01	05	0F
接触点レジスタ	02	-	-
入力レジスタ	04	-	-
ホールディングレジスタ	03	06	10

Modbusマスターステーションを作成する

説明：Modbusのマスターを作成し、スレーブとの接続を確立します。同時に最多で15のスレーブをサポートしています。

パラメータ：

- Modbusスレーブステーションの IPアドレスを入力します。
- Modbusスレーブステーションのポートを入力します。
- Modbusスレーブステーションの IDを選択します。

ロボットが搭載しているスレーブステーションに接続する場合は、IPをロボットのIP（デフォルトは192.168.5.1、変更可能）に設定し、ポートを502（map1）または1502（map2）に設定します。詳細は付録A Modbusレジスタの定義を参照してください。

サードパーティのスレーブステーションに接続する場合、レジスタの読み書き時のレジスタアドレスの取り扱い範囲と定義は、対応するスレーブステーションのModbusレジスタアドレス定義説明を参照してください。

RS485を基にしたModbusマスターステーションを作成する

説明：RS485インターフェースを基にしたModbusマスターを作成し、スレーブとの接続を確立します。最大で15の機器との接続を同時にサポートします。

パラメータ：

- RS485インターフェースのボーレートを入力します。
- ModbusスレーブステーションのIDを入力します。
- 奇数または偶数のパリティビットがあるかどうかを選択します。
- データビット長を入力します。現在のバージョンは8のみをサポートしています。
- ストップビット長。

マスターステーション作成結果を取得する

説明：Modbusマスター作成の結果を取得します。

戻り値：

- 0：Modbusマスターステーションの作成に成功しました
- 1：作成したマスターステーションが既に15個あり、新しいマスターステーションの作成に失敗しました
- 2：マスターステーションの初期化に失敗しました。IP、ポート、ネットワークの状態などを確認することをお勧めします
- 3：スレーブステーションへの接続に失敗しました。スレーブステーションが正常に設立されているか、ネットワークが正常であるかなどを確認することをお勧めします

入力レジスタを待つ

説明：入力レジスタが指定するアドレスの値が条件を満たすと、次の命令を引き続き実行します。

パラメータ：

- 入力レジスタの開始アドレスを入力します。
- 読み取るデータタイプを選択する。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）
- 入力レジスタが指定するアドレスの値が満足する条件。

保持レジスタを待つ

説明：保持レジスタが指定するアドレスが条件を満たすと、次の命令を引き続き実行します。

パラメータ：

- 保持レジスタの開始アドレスを入力します。
- 読み取るデータタイプを選択する。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）
- 入力レジスタが指定するアドレスの値が満足する条件。

接点レジスタを待つ

説明：接点アドレスが指定するアドレスの値が条件を満たすと、次の命令を引き続き実行します。

パラメータ：

- 接点レジスタの開始アドレスを入力します。
- 接点レジスタが指定するアドレスの値を満足する条件。

コイルレジスタを待つ

説明：コイルレジスタが指定するアドレスの値が条件を満たすと、次の命令を引き続き実行します。

パラメータ：

- コイルレジスタの開始アドレスを入力します。
- コイルレジスタが指定するアドレスの値を満足する条件。

入力レジスタを読み取る

説明：入力アドレスが指定する値を取得します。

パラメータ：

- 入力レジスタの開始アドレスを入力します。
- 読み取るデータタイプを選択する。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

戻り値：入力レジスタが指定するアドレスの値。

保持レジスタを読み取る

説明：保持レジスタが指定するアドレスの値を取得します。

パラメータ：

- 保持レジスタの開始アドレスを入力します。
- 読み取るデータタイプを選択する。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

戻り値：入力レジスタが指定するアドレスの値。

接点レジスタを読み取る

説明：接点レジスタが指定するアドレスの値を取得します。

パラメータ：接点レジスタの開始アドレスを入力します。

戻り値：接点レジスタが指定する値。

コイルレジスタを読み取る

説明：コイルレジスタが指定するアドレスの値を取得します。

パラメータ：コイルレジスタの開始アドレスを入入力します。

戻り値：コイルレジスタが指定するアドレスの値。

コイルレジスタを連続して読み取る

説明：コイルレジスタが指定するアドレスの値を連続で読み取ります。

パラメータ：

- コイルレジスタの開始アドレスを入力します。
- 連続で読み取ったレジスタのビット数。

戻り値：コイルレジスタが指定する値は、tableに保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

保持レジスタを連続して読み取る

説明：保持レジスタが指定するアドレスの値を連続で読み取ります。

パラメータ：

- 保持レジスタの開始アドレスを入力します。
- 連続で読み取った値の数。
- 読み取るデータタイプを選択する。
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

戻り値：コイルレジスタが指定する値は、tableに保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

コイルレジスタに書き込む

説明：指定した値をコイルレジスタが指定するアドレスに書き込みます。

パラメータ：

- コイルレジスタの開始アドレスを入力します。
- 書き込む値を選択しますが、0または1のみです。

コイルレジスタに連続して書き込む

説明：指定した値をコルレジスタが指定するアドレスに連続で書き込みます。

パラメータ：

- コイルレジスタの開始アドレスを入力します。
- 書き込む値を入力します。複数の数値はコンマで区切り、各ビットは0または1のみが可能です。

保持レジスタに書き込む

説明：指定した値を保持レジスタが指定するアドレスに書き込みます。

パラメータ：

- 保持レジスタの開始アドレスを入力します。
- 選択して書き込んだ値で、選択したデータタイプに対応していなければなりません。
- 選択して書き込むデータタイプ：
  - U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
  - U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
  - F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
  - F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

マスターステーションを閉じる

説明：Modbusマスターを閉じ、すべてのスレーブとの接続を遮断します。

バスブロックグループ

バスブロックグループはProfinetまたはEthernet/IPバスレジスタの読み書きに使用されます。バス通信機能の使用方法については、「Dobotバス通信プロトコルドキュメント（EtherNet/IP、Profinet）」をご参照ください。

> 説明：
> Magician E6は、この命令セットをサポートしていません。

バスレジスタの値を取得する

説明：指定されたバスレジスタの値を取得します。

パラメータ：

- レジスタのタイプを選択し、バス入力レジスタとバス出力レジスタをサポートします。
- レジスタのデータタイプを選択し、bool、int、floatをサポートします。
- レジスタのアドレスを選択します。

戻り値：レジスタの値を指定します。

バスレジスタの設定

説明：指定されたバスレジスタの値を設定します。

パラメータ：

- レジスタのタイプを選択し、現在はバス出力レジスタのみをサポートします。
- レジスタのデータタイプを選択し、bool、int、floatをサポートします。
- レジスタのアドレスを選択します。
- 設定する値を入力します。

TCPブロックグループ

TCPブロックグループは、TCP通信関連の操作を行うために使用します。対応するDEMOを参考に、関連命令の使用を迅速に体験できます。

SOCKETの接続

説明：TCPクライアントを作成し、指定したTCPサーバーと通信を行います。

パラメータ：

- SOCKET番号を選択して、最大4本のTCP通信リンクを確立します。
- TCPサーバーのIPアドレス。
- TCPサーバーのポート。

SOCKET接続結果の取得

説明：TCP通信接続結果を取得します。

パラメータ：SOCKET番号を選択します。

戻り値：接続が成功した場合0が返され、接続が失敗した場合1が返されます。

SOCKETの作成

説明：TCPサーバーを作成し、クライアントが接続するまで待機します。

パラメータ：

- SOCKET番号を選択して、最大4本のTCP通信リンクを確立します。
- TCPサーバーのIPアドレス。
- TCPサーバーのポート。システムによって占用されている下記ポートは使用しないでください。これらのポートを使用すると、サーバー作成が失敗する恐れがあります。

7, 13, 22, 37,

139, 445, 502, 503,

1501, 1502, 1503, 4840, 8172, 9527,

11740, 22000, 22001, 29999, 30004, 30005, 30006,

60000~65504, 65506, 65511~65515, 65521, 65522

SOCKET作成の結果の取得

説明：TCPサーバー作成結果を取得します。

パラメータ：SOCKET番号を選択します。

戻り値：作成できている場合は0が、作成できなかった場合は1が返されます。

SOCKETのオフ

説明：指定したSOCKETをオフにし、該通信リンクを切断します。

パラメータ：SOCKET番号を選択します。

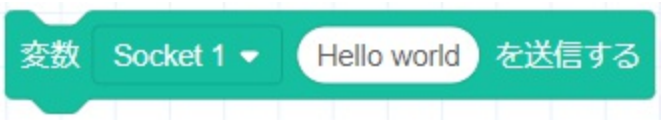
変数の取得

説明：TCP通信を介して変数を取得し、保存します。

パラメータ：

- SOCKET番号を選択します。
- 受信対象の変数のタイプを選択します。文字列、値に対応します。
- 受信したデータの変数の保存に使用されます。作成済みの変数ブロックを使用します。
- 待機タイムオーバー時間を記入します。0に設定されている場合には、タイムオーバーすることはなく、変数を受信するまでずっと待機します。

変数の送信



説明：TCP通信を介して変数を送信します。

パラメータ：

- SOCKET番号を選択します。
- 送信対象のデータ。直接記入するか、または他の戻り値が文字列または値である楕円形ブロックを使用してもよい。

変数送信結果の取得

説明：TCPを介した変数送信の結果を取得します。

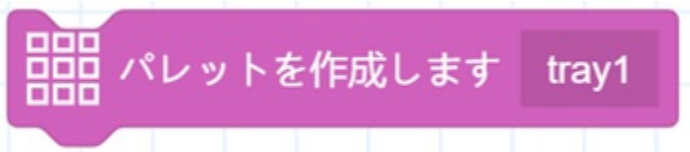
パラメータ：SOCKET番号を選択します。

戻り値：送信が成功した場合0が返され、送信が失敗した場合1が返されます。

パレットブロックセット

パレットは、バッチ材料を規則的に配置するための搬送装置であり、自動積み降ろしプロセスでよく使用されます。通常、パレットには多数の溝がアレイ状に配置されており、それぞれの溝に材料を配置できます。パレット命令を使用すると、少数の点を教示するだけで完全なパレット点配列を作成でき、作成したパレット内の特定の点を取得して、ロボットの自動ロードおよびアンロードを迅速に実現できます。

パレットを作成



説明：パレットを作成します。1次元、2次元、3次元パレットの作成をサポートします。パレットは最大20個まで作成できますが、同名のパレットを作成すると元のパレットが上書きされ、パレット数は増加しません。

パラメータ：

ブロックにパレット名を入力し、ブロックをクリックして設定ウィンドウがポップアップ表示され、パレットパラメータを設定します。

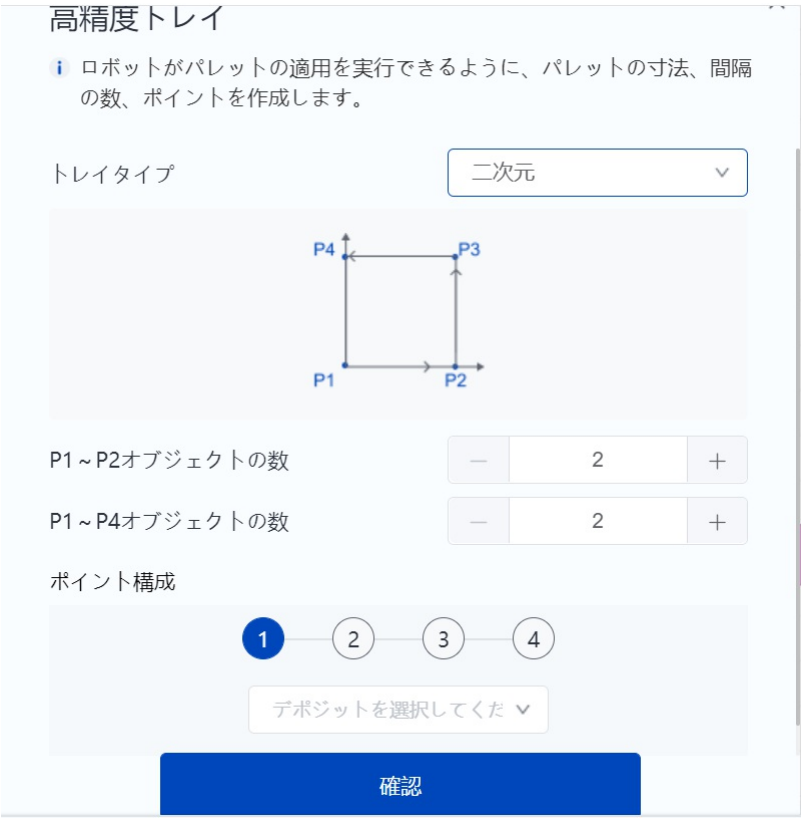
まずパレットの次元を選択します。パレットの次元が異なると、異なるパラメータを設定する必要があります。

- 1次元パレット

1次元パレットは、1直線に沿って等間隔に配置された点のセットです。

- まず、P1～P2のオブジェクトの数、つまり1次元パレット上の点の総数を入力します。
- 次に、P1点とP2点をそれぞれ設定し、保存ポイントリストからの選択のみをサポートします。

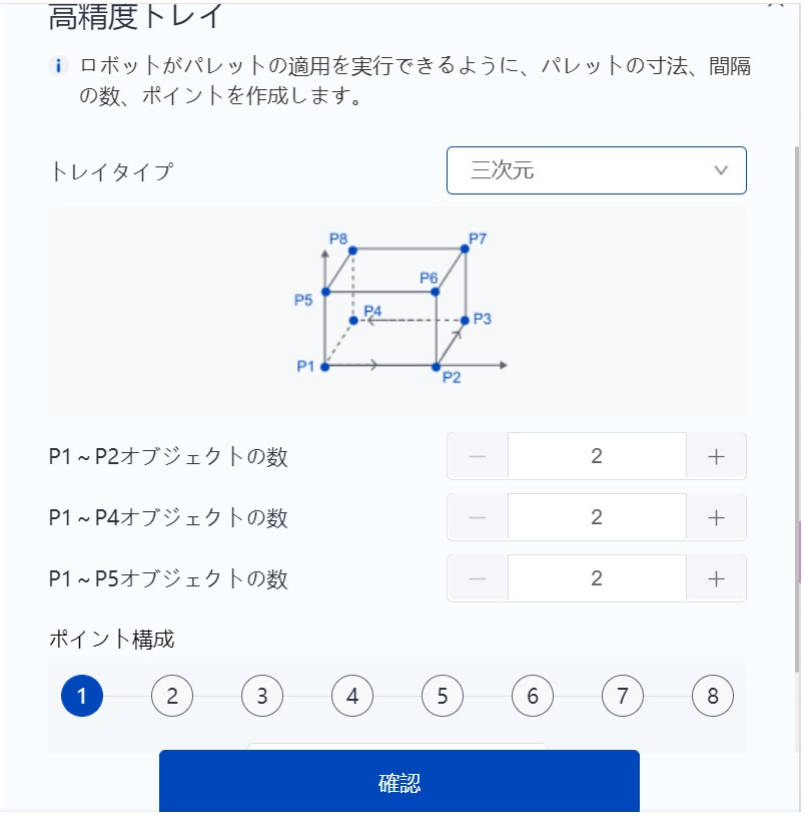
- 2次元パレット



2次元パレットは、平面上に配列された点のセットです。

- オブジェクトの数P1～P2 (行)とP1～P4 (列)をそれぞれ入力し、両者の積が2次元パレット上の点の総数となります。
- 次に、点P1～P4をそれぞれ設定します (保存ポイントリストからの選択のみをサポートします)。

- 3次元パレット



3次元パレットは、空間内に3次元的に分散された点のセットであり、垂直に配置された複数の2次元パレットとみなすことができます。

- オブジェクトの数P1～P2 (行)、P1～P4 (列)、P1～ P5 (層)をそれぞれ入力し、その3つの積が3次元パレット上の点の総数となります。
- 次に、点P1～P8をそれぞれ設定します (保存ポイントリストからの選択のみをサポートします)。

⚠注意：
先端ツールを使用する場合は、点の教示時に必ず先端ツールに対応した工具座標系を選択してください。

パレット点の総数を取得

説明：作成したパレットの点の総数を取得します。

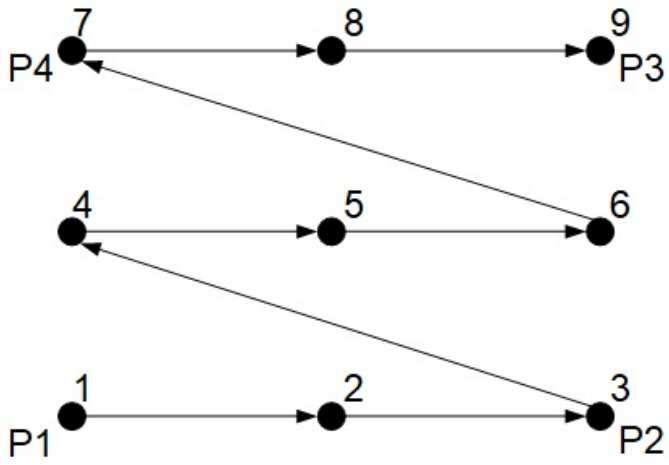
パラメータ：作成したパレットの名前を選択します。

戻る：パレット上の点の総数を指定します。

パレット点を取得

説明：指定したパレットの指定番号の点を取得します。点の番号は、パレットの作成時に渡された点のシーケンスに関連しています

- 1次元パレット：P1点の番号は1、P2点の番号は点の数と同じです。以降も同様です。
- 2次元パレット：次の図は、3x3パレットを例として、教示点と点の番号の関係を示しています。



- 3次元パレット：2次元パレットを参照すると、2番目の層の最初の点の番号は、1番目の層の最後の点の番号に1を加えたものになります。以降も同様です。

パラメータ：

- 作成したパレットの名前を選択します。
- 取得する点の番号を入力します。

戻る：番号に対応する点の座標。

クイックエクスペリエンス

- **Modbus**レジスタデータの読み取り/書き込み
- **TCP**通信を介したデータ伝送

Modbusレジスタデータの読み取り/書き込み

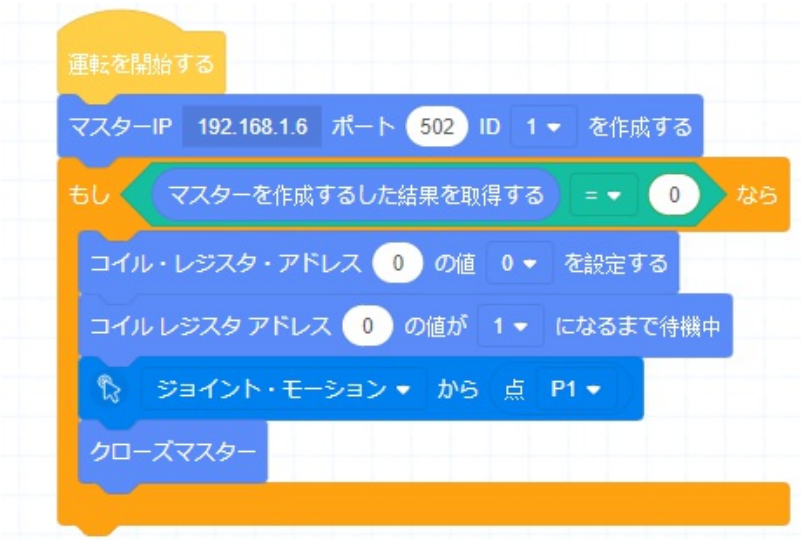
シーンの説明

グラフィカルプログラミングでModbusデータの読み取り/書き込みを行う方法を体験するために、まず以下のシーンを実現すると仮定します。

ロボットが1つのModbusマスターを作成する場合、外部スレーブに接続し、指定コイルレジスタのアドレスを読み取ります。該当するアドレスの値が1である場合、ロボットはP1点まで移動します。

プログラミング手順

上記シーンを実現するため、作成するプログラムは下図の通りです。



1. マスターを作成し、IPアドレスはスレーブアドレスで、ポートとIDはデフォルト値です。ここで迅速に検証するため、ロボット自身のスレーブを使用します。そのため、IPアドレスはロボットのアドレス（デフォルトは192.168.5.1、変更可能）となります。
2. マスターの作成が完了したかを判断し、作成完了していると後続のステップを実行できます。そうでない場合、プログラムは終了します。
3. ロボットコイルレジスタ9の値が修正されると、後続のプログラムロジックに影響を及ぼす可能性があります。そのためまず、コイルレジスタ0の値を0に設定します。
4. コイルレジスタ9の値が1になるまで待機します。
5. ロボットを制御してP1点まで移動します。P1はユーザ定義の点です。
6. マスターを閉じます。

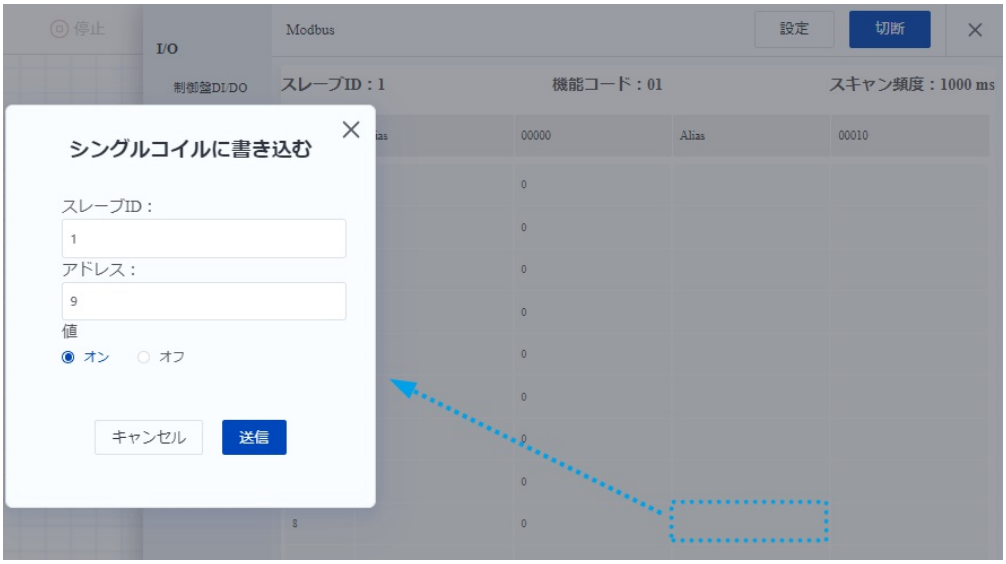
第三者のスレーブに接続する場合、マスター作成ブロックのIPとポートを第三者のスレーブのアドレスに変更し、レジスタの読み書き時のレジスタアドレスの値の範囲と定義は対応するスレーブのModbusレジスタアドレス定義説明を参照してください。

プログラムの実行

迅速にこのプログラムを実行する必要がある場合、Modbus監視ツールを使用してコイルレジスタの値を変更することができます。

1. 開く監視 > **Modbus**ページを開き、右上角の接続。
2. デフォルトの接続設定は下図の通りで、変更する必要はありません。直接接続。

3. Modbusマスターが正常に作成された後、ダブルクリック（PC端末）またはシングルクリック（モバイル端末）でコイルレジスタ9の値を表示するセルを開きます。単一のコイルを書き込むウィンドウを表示します。
4. コイルの値をオンに変更し、送信。



5. ロボットがP1まで移動するかを確認します。

TCP通信によるデータの転送

シーンの説明

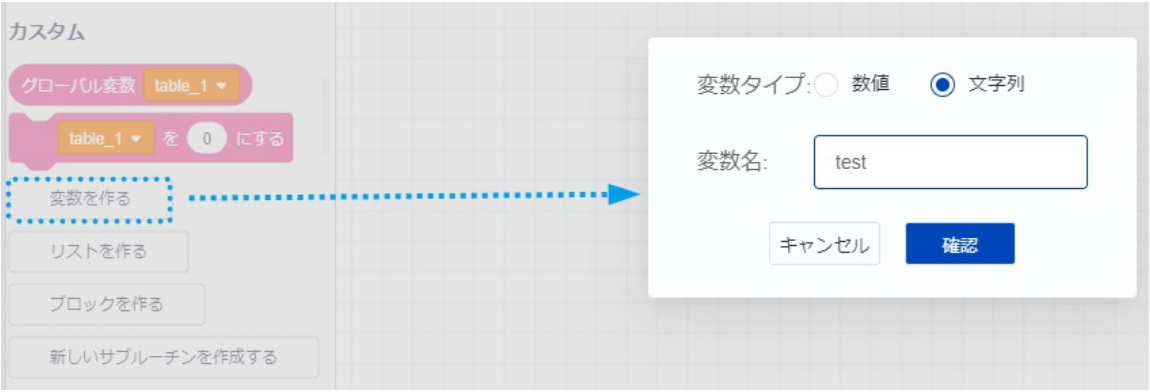
グラフィカルプログラミングでTCP通信を行う方法を体験するために、まず以下のシーンを実現すると仮定します。

ロボットは1つのTCPサーバを作成し、クライアントが接続されて「go」コマンドを送信すると、「Go to P1」情報が返され、P1点への移動が開始されます。

プログラミング手順

上記シーンを実現するため、作成するプログラムは下図のとおりです。

1. TCPサーバ(Socket 1)を作成する場合、IPアドレスはロボットのアドレスで、ポートはカスタムです。
2. サーバの作成が完了したかを判断し、作成完了していると後続のステップを実行できます。そうでない場合、プログラムは終了します。
3. クライアントが接続され、文字列を送信したら、受信した文字列を文字列変数tcp_recvに保存します。文字列変数はユーザ自身で作成してください。



4. 受信した文字列が「go」を含んでいるかを判断します。含んでいる場合、ステップ5と6を実行します。そうでない場合、直接ステップ7を実行します。

5. 文字列「Go to P1」をクライアントに送信します。
6. ロボットを制御してP1点まで移動します。P1はユーザ定義の点です。
7. TPCサーバを終了します。

付属書C スクリプトプログラミング関数の説明

Luaの基本的な文法

変数とデータ型

説明：
Luaプログラミングの知識を体系的に学びたい場合は、インターネットでLuaのチュートリアルを検索してください。このマニュアルでは、Luaの基本的な文法の一部を列挙して、素早く参照できるようにしています。

Lua言語では大文字と小文字が区別され、プログラム内のすべての識別子 **(**変数名、関数名など**)** の大文字と小文字は、このマニュアルで提供されている例と一致している必要があります。

変数は、値を格納したり、パラメータとして渡したり、結果として返したりするために使用します。変数には「=」を使用して値を指定します。

Luaでは、localを使用して変数を明示的にローカル変数（現在の関数内で有効）と宣言し、ローカル変数のスコープは宣言位置から所在関数の終了までとなります。

明示的宣言用localを使用していない変数は、デフォルトでスクリプトレベルの変数（現在のスクリプトファイル内で有効）となり、スクリプトレベルの変数のスコープは単一のスクリプトファイルとなります。

さらに、DobotStudioProのモニター>グローバル変数ページでコントローラレベルのグローバル変数を設定し、同一コントローラ内の異なるスクリプトファイルで直接呼び出すこともできます。

以下の例では、2つのハイフン（--）の後の内容は一行コメントです。

例1では、ローカル変数とスクリプトレベル変数のスコープの違いを説明しています。

```
function func()      -- 関数を定義し、実行時にa、bを出力します
```

```
    local a = 1      -- ローカル変数
    b = 2            -- スクリプトレベルの変数
    print(a)
    print(b)
end

func()             -- 関数を実行し、aとbの値をそれぞれ1と2として出力します
print(a)           --> nil（変数のスコープ外から呼び出すと、出力はnilとなります。以下同様）
print(b)           --> 2
```

例2では、ローカル変数がスクリプトレベルの変数と同名でも、関数内でのみ有効であることを説明しています。do/endなどのフロー制御（ループ、条件判断）のコードブロックも関数に属します。

```
a = "a"

for i=10,1,-1 do
do
    local a = 6     -- ローカル変数
    print(a)        --> 6、ステートメントブロック内で呼び出されるのはローカル変数aです
end

print(a)        --> a、ステートメントブロック外で呼び出されるのはスクリプトレベルの変数aです
```

例3では、グローバル変数と他の2種類の変数のスコープの違い、およびグローバル変数のグローバル保持機能を説明しています。

```
-- プロジェクト1のsrc0.luaファイル
-- 2つのグローバル変数g1とg2をグローバル変数ページに追加したと仮定します
-- g1はグローバル保持ではなく、値は10です
-- g2はグローバル保持で、値は20です

local a = 1
b = 2
print(a)        --> 1
print(b)        --> 2

print(g1)           --> 10
print(g2)           --> 20
SetGlobalVariable("g1",11)  -- グローバル保持ではないグローバル変数に値を設定します
SetGlobalVariable("g2",22)  -- グローバル保持のグローバル変数に値を設定します
print(g1)           --> 11
print(g2)           --> 22

-- プロジェクト2のsrc0.luaファイル
-- プロジェクト2はプロジェクト1の後に実行します

print(a)        --> nil
print(b)        --> nil
print(g1)       --> 10（グローバル保持ではない変数は、プロジェクト1が変更する前の値に戻ります）
print(g2)       --> 22（グローバル保持の変数はプロジェクト1が変更した後の値になります）
```

変数名は、Luaが予約しているキーワードを除いて、アルファベット、アンダースコア、数字から構成された数字以外の文字から始まる任意の文字列で構いません。

Luaでは、変数に値を設定するだけで、Luaが自動的に変数の型を判断します。変数の型を定義する必要はありません。スクリプト内で既に値が設定された変数に異なる型の値を設定すると、変数の型が変わります。しかし、モニター>グローバル変数ページで設定した変数の型は変更できず、設定する値の型は変数を作成したときに選択した型と一致していなければならず、そうでないとコントローラがスクリプトを実行するときにエラーが発生します。

Luaは、数値（number）、ブール値（boolean）、文字列（string）、テーブル（table）など、多くのデータ型をサポートしています。Luaにおける配列はtableの一種です。

Luaにはnilという特殊なデータ型もあります。nilは空（有効な値がない）を表します。例えば、値が割り当てられていない変数をプリントすると、nil値が出力されます。

数字

Luaのnumberは倍精度型の実数浮動小数点数で、さまざまな演算をサポートします。次の表記はnumberとみなされます。

- 2
- 2.2
- 0.2
- 2e+1
- 0.2e-1
- 7.8263692594256e-06

ブール値

boolean型には、true（真）とfalse（偽）の2つのオプション値しかありません。Luaはfalseとnilをfalseと見なし、それ以外はすべてtrueで、0もtrueです。

文字列

stringは、数字、アルファベット、アンダースコアからなる文字列です。stringは、次の3つの方法で表現できます。

- 一重引用符で囲まれた文字列。
- 二重引用符で囲まれた文字列。
- **[[**との**]]**間の文字列。

Luaは、数値文字列を算術演算するときに、その数値文字列を数値に変換しようとします。

Luaには、文字列の操作をサポートする多くのメソッドが用意されています。

メソッド	説明
string.upper (argument)	文字列をすべて大文字に変換する

string.lower (argument)	文字列をすべて小文字に変換する
string.gsub(mainString, findString,replaceString,num)	文字列内の文字を置換します。mainStringは処理される文字列、findStringは置換対象である文字列、replaceStringは置換される文字列、numは置換回数（省略可能な場合はすべて置換）
string.find (str, substr, [init, [end]])	指定されたターゲット文字列strの中で指定されたコンテンツsubstrを検索し、一致する部分文字列が見つかった場合は、その部分文字列の開始インデックスと終了インデックスを返します。存在しない場合はnilを返します。
string.reverse(arg)	文字列の反転
string.format(...)	printfのような書式設定された文字列を返す
string.char(arg)およびstring.byte(arg[,int])	charは整数を文字に変換して連結し、byteは文字を整数値に変換します（特定の文字を指定できます。デフォルトでは最初の文字です）。
string.len(arg)	文字列の長さの計算
string.rep(string, n)	文字列stringのn個のコピーを返す
..	2つの文字列を連結する
string.gmatch(str, pattern)	反復関数を返します。この関数が呼び出されるたびに、文字列strで見つかったpatternの記述と一致する次の部分文字列を返します。パラメータpatternで記述された文字列が見つからなかった場合、反復関数はnilを返す
string.match(str, pattern, init)	string.match()は、ソース文字列strの最初のペアのみを検索します。パラメータinitはオプションとして、検索プロセスの開始点を指定します。デフォルトは1です。 ペアリングが成功した場合、関数はペアリング式から取得したすべての結果を返します。取得タグが設定されていない場合は、ペアリング文字列全体を返します。ペアリングが成功しなかった場合は、nilを返します。
string.sub(s, i [, j])	文字列のインターセプトに使用します。sはインターセプトする文字列、iはインターセプトの開始位置、jはインターセプトの終了位置です。デフォルトは-1、つまり最後の文字です。

例：

```
str = "Lua"
print(string.upper(str))       --文字列をすべて大文字に変換し、結果をプリントする：LUA
print(string.lower(str))       --文字列をすべて小文字に変換し、結果をプリントする：lua
print(string.reverse(str))    --文字列を反転し、結果をプリントする：aul
print(string.len("abc"))       --文字列abcの長さを計算し、結果をプリントする：3
print(string.format("the value is: %d",4))     --結果をプリントする：the value is: 4
print(string.rep(str,2))       --文字列を2回コピーし、結果をプリントする：LuaLua
string1 = "cn."
string2 = "dobot"
string3 = ".cc"
print("文字列の結合",string1..string2..string3)  -- ..を使用して文字列を結合し、出力結果：cn.dobot.
cc
```

```
string1 = [[aaaa]]
print(string.gsub(string1,"a","z",3))                --文字列内の置換を行い、結果をプリントする：zz
za

print(string.find("Hello Lua user", "Lua", 1))         --文字列内でLuaを検索し、部分文字列の開始イ
ンデックスと終了インデックスを返し、結果をプリントする：7，9

sourcestr = "prefix--runoobgoogletaobao--suffix"
sub = string.sub(sourcestr, 1, 8)                    --文字列の接頭辞（1から8番目）をとる
print("\n切り取り", string.format("%q", sub))             --出力結果：prefix--の切り取り
```

テーブル

table：インデックス付きデータのセットです。

- 最も単純なコンストラクタは{}で、空のテーブル作成に使用します。テーブルを直接初期化できます。
- tableは実際には連想配列であり、任意の型の値（nilを除く）を配列のインデックスとして使用します。
- tableはサイズが固定されず、必要に応じて拡張できます。

- 記号を使用してテーブルの長さを取得できます。

```
tbl = {[1] = 2, [2] = 6, [3] = 34, [4] =5}
print("tbl の長さ ", #tbl)  -- 出力結果は4です
```

Luaは、テーブルの操作をサポートする多くのメソッドが用意されています。

メソッド	説明
table.concat (table [, sep [, start [, end]]])	concatはconcatenate（連鎖、連結）の省略形です。table.concat()関数は、指定されたtableの配列部分のstart位置からend位置までのすべての要素を、指定された区切り文字（sep）で区切ってリストします。
table.insert (table, [pos,] value)	tableの配列部分で指定された位置（pos）に値valueの要素を挿入します。posパラメータはオプションで、デフォルトはテーブルの末尾です。
table.remove (table [, pos])	table配列の部分がposの位置にある要素を返します。その後の要素は前に移動します。posパラメータはオプションで、デフォルトはtableの長さ、つまり、最後の要素から削除されます。
table.sort (table [, comp])	指定されたテーブルを昇順にソートする

例1：

```
fruits = {}         -- 初期化テーブル
fruits = {"banana","orange","apple"}  -- テーブルへの値の割り当て

print("結合後の文字列 ",table.concat(fruits,", ", 2,3))    -- インデックスを指定してtableを結合し、結合後の文字列は orange, appleです

-- 末尾に挿入
table.insert(fruits,"mango")
print("インデックス4の要素は",fruits[4])      --印刷結果：インデックス4の要素はmangoです

-- インデックスが2のキーに挿入する
table.insert(fruits,2,"grapes")
print("インデックス2の要素は",fruits[2])      --印刷結果：インデックス2の要素はgrapesです

print("最後の要素は",fruits[5])         --印刷結果：最後の要素はmangoです
table.remove(fruits)
print("削除後の最後の要素は  ",fruits[5])    --結果をプリントする：削除後の最後の要素はnil
```

例2：

```
fruits = {"banana","orange","apple","grapes"}
print("ソート前")
for k,v in ipairs(fruits) do
    print(v)          --結果をプリントする：banana orange apple grapes
end
--昇順でソートする
table.sort(fruits)
print("ソート後")
for k,v in ipairs(fruits) do
    print(v)          --結果をプリントする：apple banana grapes orange
end
```

配列

配列とは、同じデータ型の要素が一定の順序で配列された集合であり、1次元配列と多次元配列のどちらでもよい。Lua配列のインデックスキー値は整数で表し、配列のサイズは固定ではありません。

- 1次元配列：最も単純な配列で、その論理構造は線形テーブルです。
- 多次元配列：配列または1次元配列を含む配列内のインデックスキーが、1つの配列に対応します。

例1：1次元配列では、forを使用して配列内の要素をループで取得します。配列要素にアクセスするには整数インデックスを使用します。インデックスに値がない場合はnilを返します。

```
array = {"Lua", "Tutorial"}  --1次元配列の新規作成
for i= 0, 2 do
```

```
   print(array[i])            --プリント結果: nil Lua Tutorial
end
```

Luaインデックスの値は1から始まりますが、0から始まるように指定することもできます。また、負の数を配列インデックス値として使用できます。

```
array = {}
for i= -2, 2 do
   array[i] = i*2+1            --1次元配列への値の割り当て
end
for i = -2,2 do
   print(array[i])            --プリント結果: -3-1 1 3 5
end
```

例2：3行3列の配列多次元配列

```
-- 配列の初期化
array = {}
for i=1,3 do
 array[i] = {}
    for j=1,3 do
      array[i][j] = i*j
    end
end

-- 配列へのアクセス
for i=1,3 do
  for j=1,3 do
     print(array[i][j])      --プリント結果: 1 2 3 2 4 6 3 6 9
  end
end
```

演算子

算術演算子

インストラクション記号	説明
+	加算
-	減算
*	乗算
/	浮動小数点除算
//	切り捨て除算
%	切り上げ除算
^	指数演算
&	ビットAND演算
\	ビットOR演算
~	ビットXOR演算
<<	ビット左シフト演算
>>	ビット右シフト演算

例：

```
a=20
b=5
print(a+b)            --a+bの結果をプリント：25
print(a-b)            --a-bの結果をプリント：15
print(a*b)            --axbの結果をプリント：100
print(a/b)            --a/bの結果をプリント：4
print(a//b)            --aがbで割り切れる結果をプリント：4
print(a%b)            --aをbで割った余りの結果をプリント：0
print(a^b)            --aのb乗の結果をプリント：3200000
print(a&b)            --aとbのビットANDの結果をプリント：4
print(a|b)            --aとbのビットORの結果をプリント：21
print(a~b)            --aとbのビットXORの結果をプリント：17
print(a<<b)            --aをb単位で左に移動した結果をプリント：640
print(a>>b)            --aをb単位で右に移動した結果をプリント：0
```

関係演算子

インストラクション記号	説明
==	等しい

~=	等しくない
<=	以下
>=	以上
<	未満
>	より大きい

例：

```
a=20                --変数aを作成する
b=5                   --変数bを作成する
print(a==b)           --aがbに等しい比較結果をプリント：false
print(a~=b)           --aがbに等しくない比較結果をプリント：true
print(a<=b)           --aがb以下である比較結果をプリント：false
print(a>=b)           --aがb以上である比較結果をプリント：true
print(a<b)          --aがb未満である比較結果をプリント：false
print(a>b)          --aがbより大きい比較結果をプリント：true
```

ロジック演算子

インストラクション記号	説明
and	論理積、両側がtrueである場合、その結果はtrue。片側がfalseである場合、その結果はfalse
or	論理和、片側の結果がtrueである場合、その結果はtrue。orの両側がfalseである場合、その結果はfalse
not	論理否定とは、判断結果を直接逆にする

```
a=true
b=false
print(a and b)            --trueとfalse、結果はfalse
print(a or b)           --trueとfalse、結果はtrue
print(20 > 5 not true)    --trueとuntrueは、trueとfalseに等しい、最後の結果はfalse
```

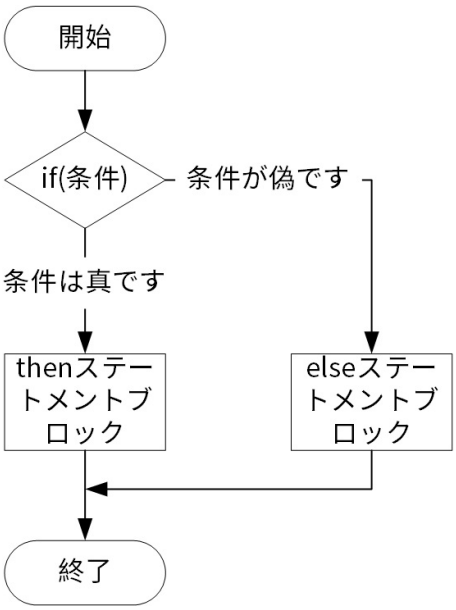
プロセス制御

命令記号	説明
if… then… else… end	if条件判定命令。上から順に条件が成立するか否かを判定し、ある判定がtrueであれば、対応するコードブロックを実行し終えると、後続の条件判定はそのまま無視し、実行されません。
while… do…end	whileループ制御命令。条件がtrueのときに、プログラムに特定の文を繰り返し実行させます。ステートメントを実行する前に条件がtrueであるかどうかをチェックします。
for… do…end	forループ制御命令は、指定した文を繰り返し実行し、繰り返し回数はfor文で設定されます。
repeat… until()	repeatループ制御命令。指定した条件が真になるまでループを繰り返します。

例：

1.if条件判定命令

ifの後の括弧は条件式であり、式の結果は任意の値です。Luaではfalseとnilをfalseとして扱い、それ以外はすべてtrue（数値0を含む）となります。式がtrueの場合は、thenステートメントブロックが実行され、式がfalse、かつelseステートメントがある場合は、elseステートメントブロックが実行され、そうでない場合はendの後のステートメントブロックが直接実行されます。



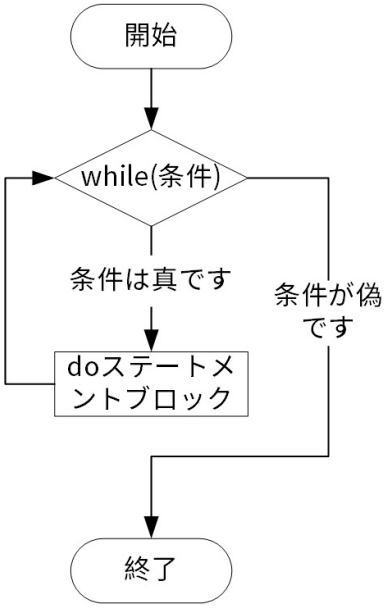
ifステートメントは入れ子にすることができます。典型的な例を次に示します。

```
a = 100;
b = 200;
```

```
--[ 条件の検査 --]
if(a == 100)
then
    --[if条件がtrueの場合は以下のif条件判定を実行します--]
    if(b == 200)
    then
        --[if条件がtrueの場合はこのステートメントブロックを実行します--]
        print("aの値は次のとおりです: ", a )  -- aの値は次のとおりです: 100
        print("bの値は:", b )  -- でbの値は200です。
    end
else
    --[最初のif条件がfalseの場合、次のステートメントが実行されます。--]
    print("aは100に等しくありません")
end
```

2.whileループ制御命令

whileの後の括弧内は条件式で、trueの場合はdoステートメントブロックが実行され、条件を再判定します。falseの場合は、endの後ろの文を直接実行します。



例：

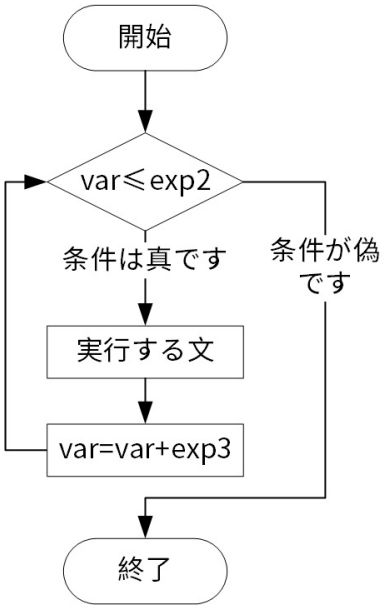
```
a=10
while( a < 20 )
do
    print("aの値は次のとおりです: ", a)  -- 10回実行すると出力値は10～19になります
    a = a+1
end
```

3.forループ制御命令

for文の構造は次のとおりです。

```
for var=exp1,exp2,exp3 do
    <実行する文>
end
```

変数varの開始値はexp1で、1回につき<実行する文>を1回繰り返し、実行後のvarの値にexp2（exp3は負の値を指定できます。指定されない場合、デフォルト値は1）ずつ加算し、exp2の値はvarの値未満であれば、処理を繰り返します。

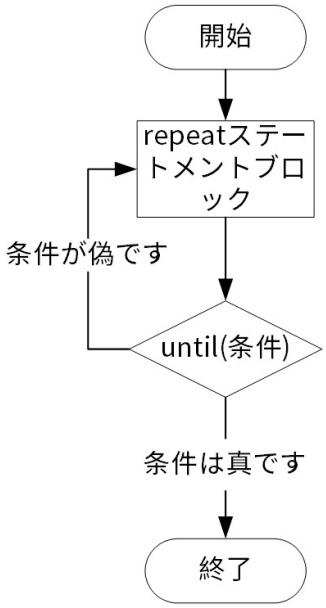


例：

```
for i=10,1,-1 do
    print(i) -- 10回実行すると、出力値は10から1までになります。
end
```

4.repeatループ制御命令

repeatループはwhile ループに似ています。主な違いは、whileはループするステートメントを実行する前に条件を判定し、条件がtrueの場合にループに入り、repeatはループするステートメントを実行した後に条件を判定し、条件が偽の場合にループすることです。



例：

```
a = 10
repeat
    print("aの値: ", a) -- 5回実行すると出力値は10から15までになります
    a = a + 1
until(a > 15)
```

全般的な説明

動作方法

ロボットがサポートする動作方法は次のいくつかに分けられます。

関節移動

ロボットは現在の各関節角度と目標点の各関節角度の差に基づいて各関節の動きを計画し、すべての関節が同時に動きを完了します。関節の動きはTCP（Tool Center Point）の動きの軌跡を制約せず、通常この軌跡は直線ではありません。

関節の動きは特異位置の制限を受けません（特異点の位置についてはロボットの対応するハードウェアマニュアルを参照）。そのため、動きの軌跡に要求がない場合や目標位置が特異位置近くにある場合は、関節の動きを使用することをお勧めします。

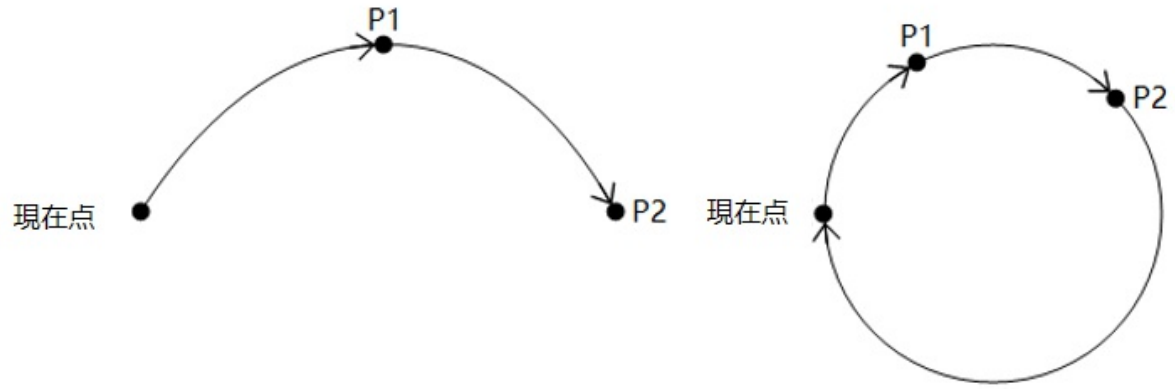
直線移動

ロボットは現在の姿勢と目標点の姿勢に基づいて動きの軌跡を計画し、TCPの動きの軌跡が直線になり、末端の姿勢が動きの過程で一定の速度で変化します。

動きの軌跡が特異位置を通る場合、直線の動きの命令をロボットに送信するとエラーが発生する可能性があります。その場合は点の位置を再計画するか、特異位置近くで関節の動きを使用することをお勧めします。

弧の動き

ロボットは現在の位置、P1、P2の3つの非共線点を使用して円弧または完全な円を確定します。動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。



動きの軌跡が特異位置を通る場合、弧の動きの命令をロボットに送信するとエラーが発生する可能性があります。その場合は点の位置を再計画するか、特異位置近くで関節の動きを使用することをお勧めします。

ポイントパラメータ

特別な説明がない場合、本ハンドブック中のすべての点パラメータは3種類の表現方法をサポートします。

- 関節変数：各ロボットの各関節の角度(j1-j6)を使用して目標点を表示します。

  ジョイント変数が直線または弧の運動パラメータとして使用される場合、システムは順運動学の数値演算によりこれを位相変数に変換しますが、アルゴリズムはロボットが目標点に到達したときのジョイント角度が設定値と一致することを保証します。

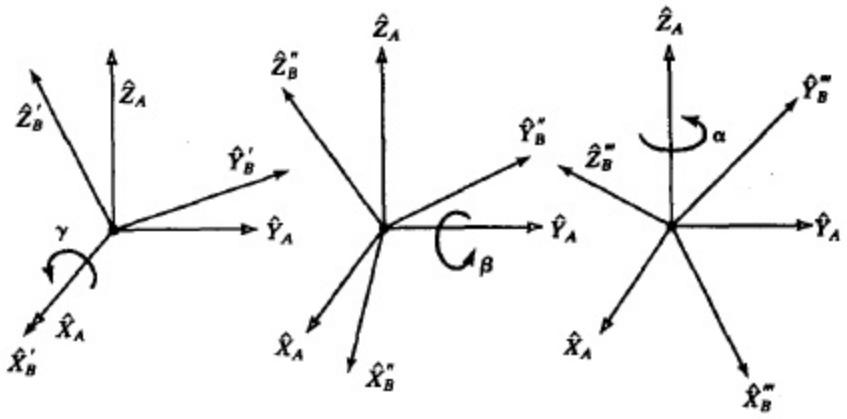
```
{joint = {j1, j2, j3, j4, j5, j6} }
```

- ポーズ変数：デカルト座標（x，y，z）を使用して、目標点のユーザ座標系における空間的位置を表示します。オイラー角（rx，ry，rz）を使用してTCP（Tool Center Point）がこの点に到達したときのツール座標系のユーザ座標系に対する回転角度を表示します。

  位相変数がジョイント運動のポイントパラメータとして使用される場合、システムは逆運動学の数値演算によりこれをジョイント変数に逆変換します（ロボットの現在のジョイント角度に最も近い解を採用）。

```
{pose = {x, y, z, rx, ry, rz} }
```

Dobotのロボットのオイラー角を計算する時の回転順序はX->Y->Zです。各軸は、下図に示されているように、固定軸（ユーザ座標系）を中心に旋回します（rx=γ，ry=β，rz=α）。



回転順序が確定すると、回転行列（cαはcosα、sαはsinαの略語である。以下同様）を

$$
\begin{aligned}
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
\end{aligned}
$$

方程式に導出します

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

この方程式により、ロボットエンドの姿勢を計算します。

- ティーチング点：制御ソフトウェアを使用して、ティーチングで取得した点は以下の形式の定数として保存されます。

```
--[[
    name：ティーチング点の名称。
    joint：ティーチング点の関節座標。
    tool：ティーチング時に使用する工具座標系のインデックス。
    user：ティーチング時に使用するユーザ座標系のインデックス。
    pose：ティーチング点の姿勢変数値。
--]]
{
    name = "name",
    joint = {j1, j2, j3, j4, j5, j6},
    tool = index,
    user = index,
    pose = {x, y, z, rx, ry, rz}
}
```

座標系パラメータ

運動命令のオプションパラメータuserとtoolは、目標点のユーザとツール座標系を指定するために使用され、現在はインデックス番号による指定のみをサポートしています。対応する座標系は先に制御ソフトウェアに追加する必要があります。

システムの選択点の座標系の優先順位は以下のとおりです。

1. モーション命令のオプションパラメータで座標系を指定した場合は、指定された座標系を使用します。点パラメータがティーチング点である場合、ティーチング点の姿勢座標を指定座標系の値に換算し、使用することができます。
2. オプションパラメータで座標系が指定されていない場合は、点がティーチング点であると、ティーチング点自体の座標系インデックスを使用します。
3. オプションパラメータで座標系が指定されていない場合は、点が関節変数または姿勢変数であると、移動パラメータ中で設定したグローバル座標系を使用します（詳細はUserとTool命令を参照。命令を呼び出して設定しない場合のデフォルト座標系は0）

説明：
- スクリプトの実行を開始すると、デフォルトのグローバル座標系はすべて0にリセットされ、スクリプトの実行前にポイント動作パネルで設定した値とは無関係になります。
- 関節運動命令（MovJ/MovJIO/RelMovJTool/RelMovJUser）を呼び出す際、ポイントが関節変数である場合、座標系パラメータは無効です。

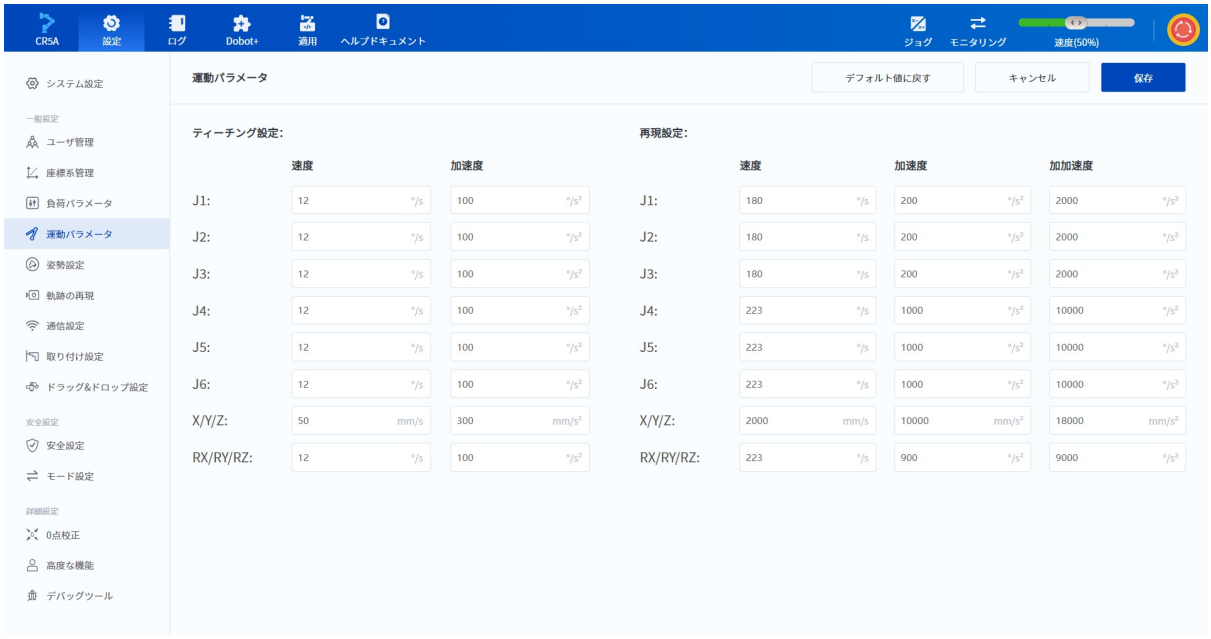
速度パラメータ

相対速度

オプションパラメータ中のaおよびvは、ロボットがこの移動命令を実行する場合の加速度と速度を指定するためのものです。

```
ロボットの実際の動きの速度 = 最大速度 × グローバルの速度比率 × 命令の速度比率
ロボットの実際の動きの加速度 = 最大加速度 × 命令の速度比率
```

そのうち、最大速度/加速度は再現設定によって制御され、制御ソフトウェアの運動パラメータページで確認および変更することができます。



グローバル速度比率は、コントロールソフトウェア（上図右上角）またはSpeedFactor命令で設定できます。

命令速度は、運動命令のオプションパラメータで指定された比率で、運動加速度/速度比率がオプションパラメータで指定されていない場合、デフォルトでは運動パラメータで設定された値（詳細はVelJ、AccJ、VelL、AccL命令を参照、命令が設定されていない場合のデフォルト値はすべて100）が使用されます。

例：

```
AccJ(50) -- 関節運動のデフォルト加速度を50%に設定
VelJ(60) -- 関節運動のデフォルト速度を60%に設定
AccL(70) -- 直線運動のデフォルト加速度を70%に設定
VelL(80) -- 直線運動のデフォルト速度を80%に設定

-- グローバル速度比率は20%;

MovJ(P1) -- （最大関節加速度 × 50%）の加速度と（最大関節速度 × 20% × 60%）の速度で関節をP1に移動
MovJ(P2,{a = 30, v = 80}) -- （最大関節加速度 × 30%）の加速度と（最大関節速度 × 20% × 80%）の速度で関節をP1に移動

MovL(P1) -- （最大デカルト加速度 × 70%）の加速度と（最大デカルト速度 × 20% × 80%）の速度でP1に直線移動

MovL(P1,{a = 40, v = 90}) -- （最大デカルト加速度 × 40%）の加速度と（最大デカルト速度 × 20% × 90%）の速度でP1に直線移動
```

絶対速度

直線および円弧移動命令オプションのspeedは、ロボットアームがこの移動を実行する場合の絶対速度を指定するためのものです。

絶対速度は、グローバル速度から影響を受けないが、再現設定の最大速度の制限を受けます（ロボットが低減モードに入っている場合、低減後の最大速度の制限を受けます）。すなわち、speedパラメータに設定した標的速度が再現設定の最大速度よりも大きい場合、最大速度に準じます。

例：

```
MovL(P1,{speed = 1000})  -- 絶対速度1000でP1に直線移動
```

MovLはspeedを1000に設定し、再現設定の最大速度2000を下回るため、ロボットは目標速度1000mm/sで運動します。この目標速度は現時点のグローバル速度比率とは無関係です。ただし、ロボットが縮減モード（縮減率を10%で仮定）である場合、最大速度は200に変わり、1000より小さいので、この時のロボットは200m/sを目標速度として移動します。

speedパラメータとvパラメータは排他的で、両方存在する場合はspeedが優先されます。

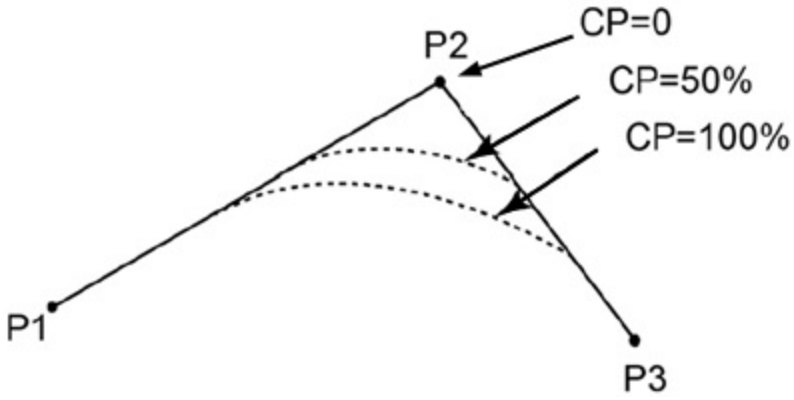
スムーズな遷移パラメータ

ロボットが複数の点を通過して連続移動する場合、連続経路制御方式で中間点を通過し、ロボットを強く回転することを回避することができます。ユーザが指定する複数の経路点が、異なるツール座標系を基にしている場合、連続経路制御を行うことはできません。

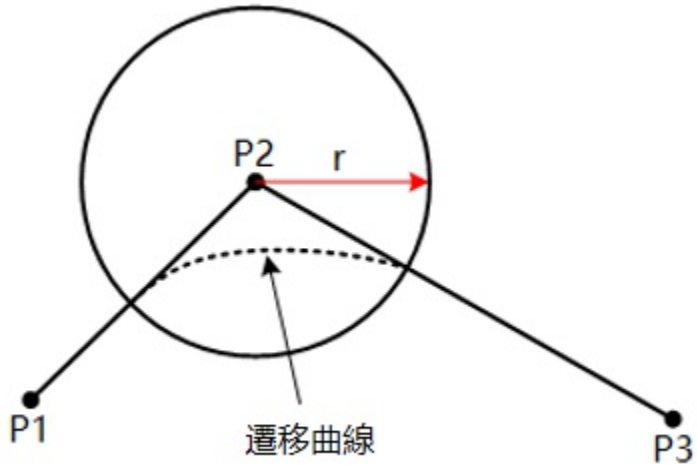
オプションパラメータのcpまたはrは、現在の移動命令から次の移動命令までの連続経路制御比率（cp）または連続経路制御半径（r）を指定するためのもので、両者は相互に排他的です。同時に存在する場合にはrを基準とします。

> 説明：
> 関節運動関連の命令はスムーズな遷移半径（r）の設定をサポートしていません。各命令のオプションパラメータをご参照ください。

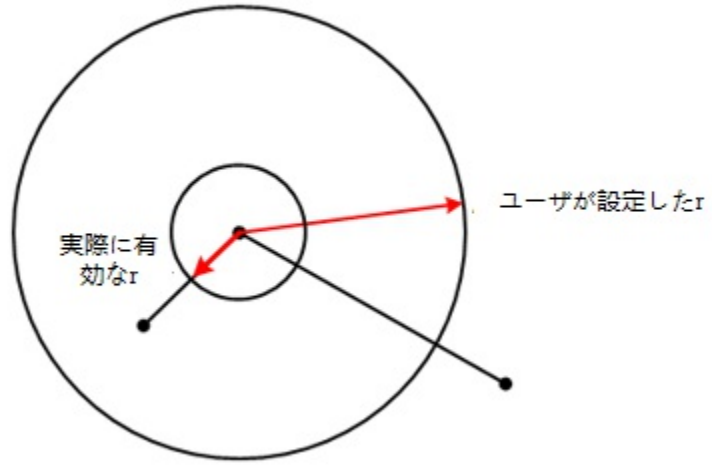
連続経路制御比率を設定する場合、システムは移動曲線の円弧を自動的に計算します。下図のとおり、CP値が大きい曲線になるほどスムーズになります。CP移動曲線は移動速度/加速度の影響を受けます。たとえ点およびCP値が同じであっても、移動速度/加速度が異なる場合の移動曲線の円弧は異なります。



連続経路制御の半径を設定する場合、システムは移動点を円の中心として、指定半径を基にして移動曲線を計算します。R移動曲線は移動速度/加速度の影響を受けず、点と移動半径のみで決定されます。



ユーザが設定する移動半径が大きすぎる場合（開始点/終了点と移動点との間の距離を超過）、システムは開始点/終了点と移動点との間の比較的短い距離の半分を移動半径を使用して、移動曲線を計算します。



オプションパラメータでスムーズな遷移比率や半径が指定されていない場合、デフォルトでは運動パラメータで設定されたスムーズな遷移比率（詳細はCP命令を参照、命令が設定されていない場合のデフォルト値は0）が使用されます。

> 説明：
> スムーズな遷移は、ロボットが中間点を通過せずに運動することを引き起こします。したがって、スムーズな遷移が設定されている場合、2つの運動命令の間のIO信号出力または機能設定（例えば、安全スキンのオン/オフ）命令は遷移過程で実行されます。
> ロボットが正確に中間点に到達したときに命令を実行したい場合は、前の命令のスムーズな遷移パラメータを0に設定してください。

cpとrの実装方法が異なるため、連続経路制御が必要な2つの移動命令の間には、移動に影響を与えない他の命令(条件判定、IO、機能設定など)が挿入される場合、cpとrの処理方法も違います。

- cpモードを使用して移行する場合、命令の処理時間が長すぎる場合 (Wait命令など) を除き、ほとんどの命令は移行に影響を与えません。

以下の例では、2つの移動命令の間にifステートメントを挿入しても、cpモードの連続経路制御には影響しません。

```
-- 正常に移動できます。ロボットはP1に到達する直前にDI1を判定し、ONの場合は連続経路制御比率50%で次の動作命令に遷移します。
MovL(P1,{cp=50})
if (DI(1)==ON)
then
    MovL(P2)
end
```

- rモードを使用して移行する場合、次のホワイトリストにある命令のみが連続経路制御に影響しません。他の命令を挿入すると、rモードでの連続経路制御が無効になります。

```
RelPointTool, RelPointUser, DOGroup, DO, AO, ToolDO,
SetUser, SetTool, User, Tool, CP, AccJ, AccL, VelL, VelJ,
SetPayload, SetCollisionLevel, SetBackDistance, EnableSafeSkin, SetSafeSkin
```

以下の例では、2つの移動命令の間にifステートメントを挿入すると、rモードでの連続経路制御の設定が無効になります。

```
-- 連続経路制御パラメータは無効であり、ロボットはP1に到達した後にDI1を判断します。
MovL(P1,{r=5})
if (DI(1)==ON)
then
    MovL(P2)
end
```

停止条件

オプションパラメータのstopcondは、移動停止条件を文字列式で指定するために使用されます。停止条件を指定すると、ロボットは動作命令の実行中にリアルタイムで停止条件を確認し、停止条件が成立すると現在の動作を終了し、次の命令を直接実行します。

停止条件はLua構文を満たす任意の式をサポートします。式がtrueの場合は、条件を満たしているとみなされます。具体例は次のとおりです。

```
-- DI1がONの場合は動作を停止します
MovJ(P1,{stopcond="DI(1) == ON"})
-- DI1がON、DO1がONの場合は動作を停止します
MovJ(P1,{stopcond="DI(1) == ON and GetDO(1) == ON"})
-- 保持レジスタアドレス100に格納されている値が10未満の場合に移動を停止します
MovJ(P1,{stopcond="GetHoldRegs(id, 100, 1)[1] < 10"})
-- 変数var1が5に等しくない場合は移動を停止します
MovJ(P1,{stopcond="var1 ~= 5"})
```

説明：

- 停止条件を指定すると、連続経路制御パラメータrが無効になります。
- 停止条件を指定すると、連続経路制御パラメータcpが有効になりますが、連続経路制御部（曲線部分）が停止条件の有効範囲に入りません。

IO信号の表現方法

システム内部では、ONとOFFの変数がデジタルIOの信号有無を表すために事前定義されています。s

- ON　=　1は、信号があることを意味します
- OFF　=　0は、信号がないことを意味します

パラメータの中のON|OFFは、パラメータの値がONまたはOFFであることを意味します。ユーザは1または0を入力として使用することもできます。

移動命令

命令リスト

移動命令は、ロボットを制御して移動させるためのものです。**使用する前に[全般的な説明](common.md)を読んでください**。 | 命令 | 機能 | | ---------- | ---------- | | [MovJ](#movj) | 関節移動 | | [MovL](#movl) | 直線移動 | | [Arc](#arc) | 円弧補間移動 | | [Circle](#circle) | 全円補間移動 | | [MovJIO](#movjio) | 関節移動と出力DO | | [MovLIO](#movlio) | 直線移動と出力DO | | [StartPath](#startpath) | 記録した移動軌跡を再生 | | [GetPathStartPose](#getpathstartpose) | 軌道の開始点を取得 | | [PositiveKin](#positivekin) | 姿勢に対する関節角度の順解法 | | [InverseKin](#inversekin) | 関節角度に対する姿勢の逆解法 |

MovJ

プロトタイプ： ```lua MovJ(point, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = "expression"}) ``` **説明：** 現在の位置から関節移動方式で目標点まで井戸します。 **必須パラメータ：** point：目標点。 **選択可能なパラメータ：** - user：目標点のユーザ座標系。 - tool：目標点の工具座標系。 - a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100] - v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100] - cp：連続経路制御の比率。値の範囲：[0,100] - stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。 詳細は[一般説明](common.md)を参照してください。 **例：** ```lua -- ロボットはデフォルト設定でP1点まで関節移動します。 MovJ(P1) ``` ```lua -- ロボットはデフォルト設定で指定の関節角度まで関節移動します。 MovJ({joint={0,0,90,0,90,0} }) ``` ```lua -- ロボットは指定の姿勢まで関節移動します。姿勢はユーザ座標系1と工具座標系1に対応し、移動加速度と速度は共に50%、連続経路制御比率は50%です。 MovJ({pose={300,200,300,180,0,0} }, {user=1,tool=1,a=50,v=50,cp=50}) ``` ```lua -- 先に点を定義し、その後移動命令で呼び出します。実行結果は上記の命令と同じです。 customPoint={pose={300,200,300,180,0,0} } MovJ(customPoint, {user=1,tool=1,a=50,v=50,cp=50}) ``` ```lua -- ロボットアームはP1に関節的に移動し、移動中にDI1がONになると、現在の移動は終了します。 MovJ(P1,{stopcond="DI(1) == ON"}) ```

MovL

プロトタイプ：

```
MovL(point, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "exp
ression"})
```

説明：

現在の位置から直線移動方式で目標点まで移動します。

必須パラメータ：

point：目標点。

選択可能なパラメータ：

- user：目標点のユーザ座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- tool：目標点の工具座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定でP1点まで直線移動します。
MovL(P1)
```

```
-- ロボットは500m/sの絶対速度でP1点まで直線移動します。
MovL(P1,{speed=500})
```

```
-- ロボットはデフォルト設定で指定の関節角度まで直線移動します。
MovL({joint={0,0,90,0,90,0} })
```

```
-- ロボットは指定の姿勢まで直線移動します。姿勢はユーザ座標系1と工具座標系1に対応し、移動加速度と速度は共に50%、連続経路制御半径は5mmです。
MovL({pose={300,200,300,180,0,0} },{user=1,tool=1,a=50,v=50,r=5})
```

```
-- 先に点を定義し、その後移動命令で呼び出します。実行結果は上記の命令と同じです。
customPoint={pose={300,200,300,180,0,0} }
MovL(customPoint,{user=1,tool=1,a=50,v=50,r=5})
```

```
-- speedとvを同時に持つと、speedが有効になり、同時にコントローラは対応する警告ログを出力します。
-- cpとrを同時に持つと、rが有効になり、同時にコントローラは対応する警告ログを出力します。
MovL(P1,{v=50,speed=500,cp=60,r=5}) -- 実行時に選択可能なパラメータはspeedとrのみ有効です
```

```
-- ロボットアームはP1まで直線的に移動し、移動中にDI1がONになると、現在の移動は終了します。
MovL(P1,{stopcond="DI(1) == ON"})
```

Arc

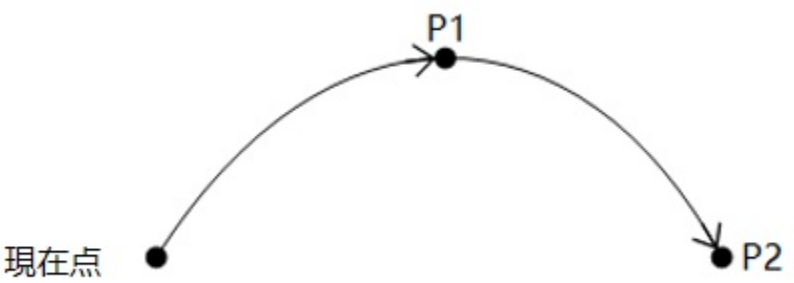
プロトタイプ：

```
Arc(P1, P2, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "exp
ression"})
```

説明：

現在の位置から円弧補間方式で目標点まで移動します。

現在の位置、P1、P2の3点により1つの円を確定します。そのため、現在の位置はP1とP2で決まる直線上にあってはなりません。



動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

必須パラメータ：

- P1：円弧の中間点。
- P2：目標点。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]

- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはP1まで移動し、さらにP2を経てデフォルト設定でP3まで円弧移動します。
MovJ(P1)
Arc(P2,P3)
```

```
-- ロボットはP1まで移動し、さらに点{300,200,300,180,0,0}を経てP3まで円弧移動します。ユーザ座標系と工具座標系はどちらも1で、移動加速度と速度は共に50%です。
MovJ(P1)
Arc({pose={300,200,300,180,0,0} },P3,{user=1,tool=1,a=50,v=50})
```

```
-- ロボットアームはP1に移動し、その後デフォルト設定でP2からP3までの円弧に沿って移動し、移動中にDI1がONになると、現在の移動は終了します。
MovJ(P1)
Arc(P2,P3,{stopcond="DI(1) == ON"})
```

Circle

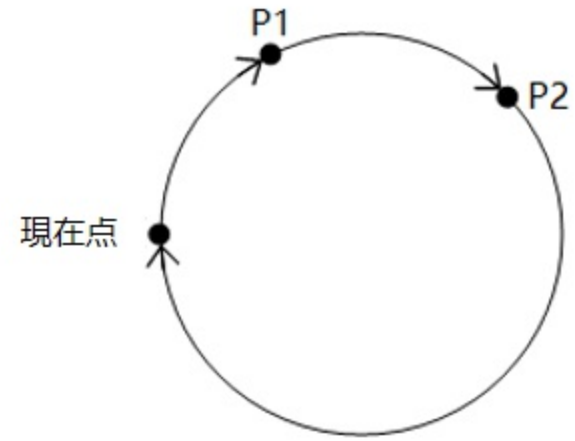
プロトタイプ：

```
Circle(P1, P2, Count, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100, r = 5, stopcond = "expression"})
```

説明：

現在位置で円形補完移動を行い、指定の回数を移動したら現在位置に戻ります。

現在の位置、P1、P2の3点により1つの円を確定します。そのため、現在の位置はP1とP2で決まる直線上にあってはなりません。さらに、3点で決まる円は、ロボットの移動範囲を超えてはなりません。



動きの過程でのロボットの末端の姿勢は、現在点とP2点の姿勢から補間計算され、P1点の姿勢は計算に使用されません（つまり、ロボットがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

必須パラメータ：

- P1：円の位置1。
- P2：円の位置2。
- Count：完全円形移動を行う回数ををを表します。値範囲は1~999とします。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはP1まで移動し、さらにP1、P2、P3で決まる円に沿って1周移動します。
MovJ(P1)
Circle(P2,P3,1)
```

```
-- ロボットはP1まで移動し、さらにP1、点{300,200,300,180,0,0}、P3で決まる円に沿って10周移動します。ユーザ座標系と工具座標系はどちらも1で、移動加速度と速度は共に50%です。
MovJ(P1)
Circle({pose={300,200,300,180,0,0} },P3,10,{user=1,tool=1,a=50,v=50})
```

```
-- ロボットアームはP1に移動し、その後P1、P2、P3で決められた円形に沿って一周移動し、移動中にDI1がONになると、現在の移動は終了します。
MovJ(P1)
Circle(P2,P3,1,{stopcond="DI(1) == ON"})
```

MovJIO

プロトタイプ：

```
MovJIO(P,{ {Mode,Distance,Index,Status},{Mode,Distance,Index,Status}...}, {user = 1, tool = 0,
 a = 20, v = 50, cp = 100})
```

説明：

現在の位置から、関節移動方式により目標点まで移動します。移動する場合にデジタル出力のポート状態を並行して設定します。

必須パラメータ：

- P：目標点。
- 並行デジタル出力パラメータ：ロボットが指定距離または百分率まで移動する場合に、指定のDOを起動することを設定します。複数グループを設定することができます。各グループは以下のパラメータを含みます。
  - Mode：トリガモード。0歯百分率の起動を示し、1は距離の起動を示します。システムは各関節角を1つの角度ベクトルを合成し、終点と起点の角度差を移動の総距離として計算します。
  - Distance：百分率/角度を指定します。角度計算は合成角ベクトルを使用しているため、百分率モードを使用することをお勧めします。その方の効果が直感的に理解しやすいです。
    - Distanceが正の数である場合、起点からの百分率/角度を表示します。
    - Distanceが負の数である場合、目標点からの百分率/角度を表示します。
    - Modeが0である場合、Distanceは合計角度との百分率を表示します。値の範囲：(0,100]
    - Modeが1である場合、Distanceは角度の値を表示します。単位：°
  - Index：DO端子の番号。
  - Status：設定するDO状態で、0とOFFは信号なし、1とONは信号ありを示します。

選択可能なパラメータ：

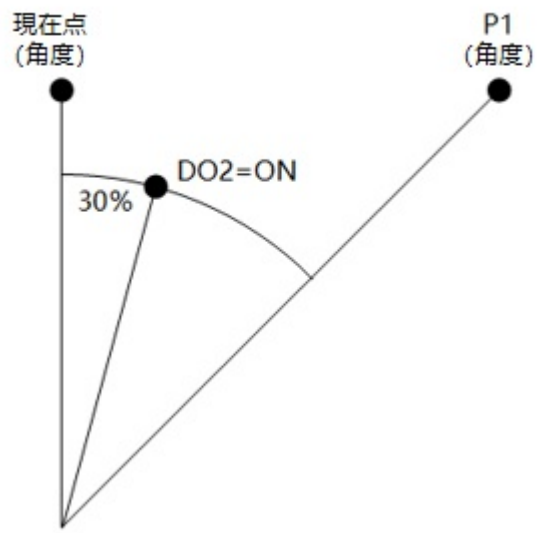
- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]

スムーズ移動はロボットの移動軌跡を変更し、DO出力のタイミングに影響を及ぼしますので、慎重に使用してください。

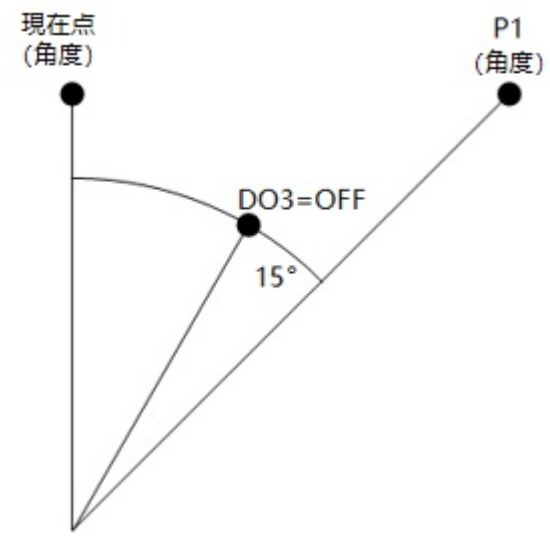
詳細は一般説明を参照してください。

例：

```
-- ロボットは、デフォルト設定ではP1点に向けて関節移動します。開始点から30%の位置に移動すると、DO2がオンになるように設定します。
MovJIO(P1, { {0, 30, 2, 1} })
```




```
-- ロボットは、デフォルト設定ではP1点に向けて関節移動します。終点まであと15°の位置に到達すると、DO3がオフになるように設定します。
MovJIO(P1, { {1, -15, 3, 0} })
```



MovLIO

プロトタイプ：

```
MovLIO(P,{ {Mode,Distance,Index,Status},{Mode,Distance,Index,Status}...},{user = 1, tool = 0,
a = 20, v = 50, cp = 100, r = 5})
```

説明：

現在の位置から直線移動方式で目標点まで移動し、移動する場合にはデジタル出力ポート状態を並行して設定します。

必須パラメータ：

- P：目標点。
- 並行デジタル出力パラメータ：ロボットが指定距離または百分率まで移動する場合に、指定のDOを起動することを設定します。複数グループを設定することができます。各グループは以下のパラメータを含みます。
  - Mode：トリガモード。0歯百分率の起動を示し、1は距離の起動を示します。
  - Distance：百分率/距離を指定します。
    - Distanceが正の数である場合、起点からの百分率/距離を表示します。
    - Distanceが負の数である場合、目標点からの百分率/距離を表示します。
    - Modeが0である場合、Distanceは合計距離との百分率を表示します。値の範囲：(0,100]
    - Modeが1である場合、Distanceは距離の値を表示します。単位：mm
  - Index：DO端子の番号。
  - Status：設定するDO状態で、0とOFFは信号なし、1とONは信号ありを示します。

選択可能なパラメータ：

- user：目標点のユーザ座標系。

- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

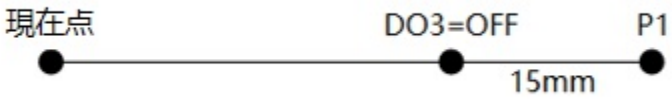
スムーズ移動はロボットの移動軌跡を変更し、DO出力のタイミングに影響を及ぼしますので、慎重に使用してください。

詳細は一般説明を参照してください。

例：

```
-- ロボットは、デフォルト設定ではP1点に向けて直線移動します。開始点から30%の位置に移動すると、DO2がオンになるように設定します。
MovLIO(P1, { {0, 30, 2, 1} })
```


```
-- ロボットは、デフォルト設定ではP1点に向けて直線移動します。終点まで15mmの位置に到達すると、DO3がオフになるように設定します。
MovLIO(P1, { {1, -15, 3, 0} })
```



StartPath

プロトタイプ：

```
StartPath(string, {multi = 1, isConst = 0, sample = 50,  freq = 0.2, user = 0,tool = 0})
```

説明：

指定した軌跡ファイル中で記録した軌跡を再現します。この命令を呼び出す前に、ユーザはロボットを軌跡の開始点まで移動する必要があります。

必須パラメータ：

string：軌跡ファイル名（サフィックスを含む）。

選択可能なパラメータ：

- multi：再現する場合の速度倍数で、isConst=0の場合のみ有効。値の範囲：[0.1, 2]、引数がない場合のデフォルト値は1です。
- isConst：均一速度で再現するかどうか。引数がない場合のデフォルト値は0です。
  - 1は均一速での再現を示し、ロボットは一定のグローバル速度で軌跡を再現します。
  - 0は軌跡記録時の元の速度で再現することを示します。multiパラメータを使用して移動速度を比例的に調整できます。この場合、ロボットの移動速度はグローバル速度の影響を受けません。
- sample：軌跡点のサンプリング間隔です。つまり、軌跡ファイルを作成した場合の、隣接する2つの点のサンプリング時間差です。値の範囲：[8, 1000]、単位：ms、引数がない場合のデフォルト値は50です（コントローラが軌跡ファイルを記録する場合のサンプリング間隔）。
- freq：フィルタリング係数で、このパラメータの値が小さくなるほど、再現する軌跡曲線はスムーズになります。ただし、元の軌跡に対する変形が厳しくなるほど、元の軌跡のスムーズ程度に基づいて適切なフィルタリング係数を設定してください。値の範囲：（0,1]、1はフィルタリング終了を示し、引数がない場合のデフォルト値は0.2です。
- user：軌跡点に対応するユーザ座標系インデックスを指定します。指定しない場合、軌跡ファイル中に記録されたユーザ座標系インデックスを使用します。
- tool：軌跡点に対応する工具座標系インデックスを指定します。指定しない場合、軌跡ファイル中に記録された工具座標系インデックスを使用します。

例：

```
-- track.csv軌跡ファイルの開始点まで関節移動した後、元の速度の2倍でファイル中に記録された軌跡を再現します。
local StartPoint = GetPathStartPose("track.csv")
MovJ(StartPoint)
StartPath("track.csv", {multi = 2, isConst = 0})
```

```
-- track.csv軌跡ファイルの開始点まで関節移動した後、ファイル中に記録された軌跡を均一速度で再現します。
local StartPoint = GetPathStartPose("track.csv")
MovJ(StartPoint)
StartPath("track.csv", {isConst = 1})
```

GetPathStartPose

プロトタイプ：

```
GetPathStartPose(string)
```

説明：

軌跡の開始点を取得します。軌跡ファイルにより、点データのタイプ（ティーチング点/関節/姿勢）も異なる場合があります。

必須パラメータ：

string：軌跡ファイル名（サフィックスを含む）。

例：

```
-- track.csv軌跡ファイルの開始点を取得し、印刷します。
local StartPoint = GetPathStartPose("track.csv")
print(StartPoint)
-- StartPoint = {joint={j1,j2,j3,j4,j5,j6},"name" = "",user = userIndex,tool = toolIndex,pose
= {x,y,z,a,b,c}}
-- StartPoint = {joint={j1,j2,j3,j4,j5,j6}}
-- StartPoint = {pose = {x,y,z,a,b,c}}
```

PositiveKin

プロトタイプ

```
PositiveKin(joint, {user = 1, tool = 0})
```

説明

正演算を実行する：ロボットの各関節角度を与え、ロボット末端の与えられたデカルト座標系中の座標値を計算します。

必須パラメータ

joint：関節変数。形式は `{joint = {j1, j2, j3, j4, j5, j6} }`

選択可能なパラメータ

- user：ユーザ座標系のインデックス。指定していない場合、グローバルユーザ座標系を使用します。
- tool：ツール座標系のインデックス。指定していない場合、グローバルツール座標系を使用します。

戻る

順解法で得られた姿勢変数。形式は `{pose = {x, y, z, rx, ry, rz} }`

例：

```
PositiveKin({joint={0,0,-90,0,90,0} },{user=1,tool=1})
```

関節座標が{0,0,-90,0,90,0}であり、ロボット末端のユーザ座標系1および関節座標系1のデカルト座標を計算します。

InverseKin

プロトタイプ

```
InverseKin(pose, {useJointNear = true, jointNear = joint, user = 1, tool = 0})
```

説明

逆演算を実行する：ロボット末端の与えられたデカルト座標系中の座標値が与えられ、ロボットの各関節の角度を計算します。

デカルト座標はTCPの空間座標と傾斜角のみを定義するので、ロボットは多種の異なるポーズを通じて同一のポーズに達し、1つのポーズ変数が複数の関節変数に対応できることを意味します。一意の解を得るため、システムは指定された関節座標を必要とし、当該関節座標に最も近い解を逆演算の結果として選択します。

必須パラメータ

pose：姿勢変数。形式は `{pose = {x, y, z, rx, ry, rz} }`

選択可能なパラメータ

- useJointNear：ブール値。JointNearパラメータが有効かどうかを設定するためのものです。
  - trueは、JointNearパラメータに応じて最も近い解を選択します。
  - falseまたはパラメータを指定していない場合は、JointNearパラメータが無効であることを意味し、システムはロボットの現在の関節角度に応じて最も近い解を選択します。
  - JointNearパラメータを指定せずにこのパラメータだけを指定した場合、このパラメータは無効になります。
- jointNear：最も近い解を選択するための関節変数。形式は `{joint = {j1, j2, j3, j4, j5, j6} }`
- user：ユーザ座標系のインデックス。指定していない場合、グローバルユーザ座標系を使用します。
- tool：ツール座標系のインデックス。指定していない場合、グローバルツール座標系を使用します。

戻る

- エラー コード。0は逆解が成功したことを意味し、-1は逆解が失敗した (解がない) ことを意味します。
- 逆解から得られる関節変数。形式は `{joint = {j1, j2, j3, j4, j5, j6} }`。逆解が失敗すると、j1~j6はすべて0になります。

例：

```
local errId, jointPoint = InverseKin({ pose = {300, 200, 300, 180, 0, 0} }, {useJointNear = tr
ue, jointNear = { joint = {90, 30, -90, 180, 30, 0} } })
```

ロボットの末端のグローバルユーザ座標系およびグローバル関節座標系でのデカルト座標は、{300, 200, 300, 180, 0, 0}です。関節座標を計算し、関節角度{90, 30, -90, 180, 30, 0}と最も近い解を選択します。

相対移動命令

命令リスト

相対移動命令関数は、ロボットを制御して偏移動作を行うためのものです。使用する前に一般的な説明を読んでください。

命令	機能
RelPointUser	点をユーザ座標系に沿って偏移
RelPointTool	点を工具座標系に沿って偏移
RelMovJTool	工具座標系に沿って相対関節移動を実行
RelMovLTool	工具座標系に沿って相対直線移動を実行
RelMovJUser	ユーザ座標系に沿って相対関節移動を実行
RelMovLUser	ユーザ座標系に沿って相対直線移動を実行
RelJointMovJ	関節は指定した偏移角度まで移動
RelJoint	点を指定した間接角度まで偏移

RelPointUser

プロトタイプ：

```
RelPointUser(P, {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz})
```

説明：

指定点に対して指定したユーザ座標系に沿って変位し、偏移後の姿勢変数を返します。

必須パラメータ：

- P：変位する指定点。
- {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}：ユーザ座標系での偏移量を指定します。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。
  - 指定点がティーチング点である場合、ティーチング点のユーザ座標系を基にしてオフセットを実行します。
  - 指定点が関節変数または姿勢変数である場合、グローバルユーザ座標系を基にしてオフセットを実行します。

戻る：

偏移後の姿勢変数：{pose = {x, y, z, rx, ry, rz}}

例：

```
-- P1を指定ユーザ座標系において一定距離オフセットさせ、さらにオフセット後の点に移動します。
local Offset={OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}
local p = RelPointUser(P1, Offset)
MovL(p)
```

RelPointTool

プロトタイプ：

```
RelPointTool(P, {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz})
```

説明：

指定点に対して指定工具座標系に沿って偏移し、偏移後の姿勢変数を返します。

必須パラメータ：

- P：変位する指定点。
- {OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}：工具座標系での偏移量を指定します。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。
  - 指定点がティーチング点である場合、ティーチング点の工具座標系を基にしてオフセットを実行します。
  - 指定点が関節変数または姿勢変数である場合、グローバルツール座標系を基にしてオフセットを実行します。

戻る：

偏移後の姿勢変数：{pose = {x, y, z, rx, ry, rz}}

例：

```
-- P1を指定工具座標系において一定距離オフセットさせ、さらにオフセット後の点に移動します。
local Offset={OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz}
local p = RelPointTool(P1, Offset)
MovL(p)
```

RelMovJTool

プロトタイプ：

```
RelMovJTool({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = "
expression"})
```

説明：

現在の位置から、指定ツール座標系に沿って、関節移動方式で相対移動を行います。関節移動の軌跡は直線ではなく、すべての関節は同時に移動を完了します。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定工具座標系における偏移量。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定では、グローバルツール座標系に沿って指定のオフセット点まで関節移動します。
RelMovJTool({10, 10, 10, 0, 0, 0})
```

RelMovLTool

プロトタイプ：

```
RelMovLTool({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100,
 r = 5, stopcond = "expression"})
```

説明：

現在の位置から、指定工具座標系に沿って、直線移動方式で相対移動を行います。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定工具座標系における偏移量。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範

囲：(0,100]

- speed：この命令を実行する場合のロボットの移動目標速度で、vと相互に排他的です。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定により、グローバルツール座標系に沿って指定のオフセット点まで直線移動します。
RelMovLTool({10, 10, 10, 0, 0, 0})
```

RelMovJUser

プロトタイプ：

```
RelMovJUser({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, cp = 100, stopcond = "
expression"})
```

説明：

現在の位置から、指定ユーザ座標系に沿って、関節移動方式で相対移動を行います。関節運動の軌跡は直線ではなく、すべての関節は同時に異動を完了します。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定ユーザ座標系における偏移量。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定により、グローバルユーザ座標系に沿って指定のオフセット点まで関節移動します。
RelMovJUser({10, 10, 10, 0, 0, 0})
```

RelMovLUser

プロトタイプ：

```
RelMovLUser({x, y, z, rx, ry, rz}, {user = 1, tool = 0, a = 20, v = 50, speed = 500, cp = 100,
 r = 5, stopcond = "expression"})
```

説明：

現在の位置から、指定ユーザ座標系に沿って、直線移動方式で相対移動を行います。

必須パラメータ：

{x, y, z, rx, ry, rz}：目標点の現在の位置に対して指定ユーザ座標系における偏移量。x，y，zは空間偏移量（mm）を示し、rx，ry，rzは角度偏移量（°）を示す。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する場合のロボットの移動目標速度で、vと相互に排他的です。値の範囲：[1、最大移動速度]、単位：mm/s
- cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm
- stopcond：停止条件式です。この条件が成立すると現在の動作を終了し、次の命令を実行します。

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定により、グローバルユーザ座標系に沿って指定のオフセット点まで直線移動します。
RelMovLUser({10, 10, 10, 0, 0, 0})
```

RelJointMovJ

プロトタイプ：

```
RelJointMovJ({Offset1, Offset2, Offset3, Offset4, Offset5, Offset6}, {a = 20, v = 50, cp = 100
})
```

説明：

現在の位置から、関節移動方式で指定された関節偏移角度まで移動します。

必須パラメータ：

{Offset1, Offset2, Offset3, Offset4, Offset5, Offset6}：関節座標系でのJ1軸～J6軸方向の偏移値（°）。

選択可能なパラメータ：

- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：スムーズ移動比率。値の範囲：[0,100]

詳細は一般説明を参照してください。

例：

```
-- ロボットはデフォルト設定に従ってオフセット角度に関節移動します。
RelJointMovJ({20,20,10,0,10,0})
```

RelJoint

プロトタイプ：

```
RelJoint(P, {Offset1, Offset2, Offset3, Offset4, offset5, offset6})
```

説明：

指定点に対して関節座標系でJ1~J6軸の変位量を増やし、偏移後の関節変数を返します。

必須パラメータ：

- P：変位する指定点。
- Offset1~Offset6：関節座標系でのJ1軸～J6軸方向における偏移値（°）です。

戻る：

偏移後の関節変数：{joint = {j1, j2, j3, j4, j5, j6}}。

例：

```
-- P1をJ1~J6軸上にそれぞれ一定角度オフセットさせ、さらにオフセット後の点に移動します。
local Offset = {Offset1, Offset2, Offset3, Offset4, offset5, offset6}
```

```
local p = RelJoint(P1, Offset)
MovJ(p)
```

移動パラメータ

命令リスト

運動パラメータ関数は、ロボットの運動に関連するパラメータを設定または取得するために使用します。使用する前に一般的な説明を読んでください。

命令	機能
CP	連続経路制御の比率の設定
VelJ	関節の移動方式の速度設定
AccJ	関節の移動方式の加速度設定
VelL	直線と弧の移動方式の速度比率の設定
AccL	直線と円弧の移動方式の加速度設定
SpeedFactor	ロボットのグローバル移動速度の設定
SetPayload	エンド負荷の設定
User	グローバルユーザ座標系の設定
SetUser	指定したユーザ座標系の修正
CalcUser	ユーザ座標系の計算
Tool	グローバルツール座標系の設定
SetTool	指定したツール座標系の修正
CalcTool	ツール座標系の計算
GetPose	ロボットのリアルタイム姿勢の取得
GetAngle	ロボットのリアルタイム間接角度の取得
GetABZ	エンコーダの現在の位置の取得
CheckMovJ	関節移動命令の移動可能性の確認
CheckMovL	直線または円弧移動命令の移動可能性の確認
SetSafeWallEnable	指定した安全壁の起動またはシャットダウン
SetWorkZoneEnable	指定の干渉区域を開くまたは閉じる
SetCollisionLevel	衝突レベルの設定
SetBackDistance	衝突戻り距離の設定

CP

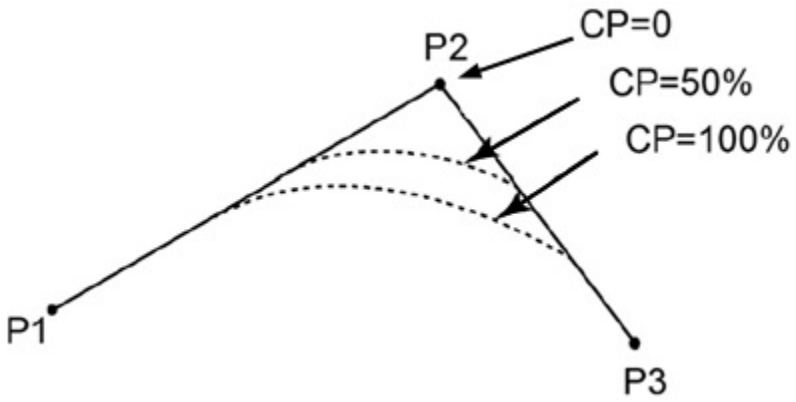
プロトタイプ：

```
CP(R)
```

説明：

連続経路制御の比率を設定します。すなわち、ロボットが複数の点を経由して連続的に移動するとき、直角方式でそれとも曲線方式で中間点を通過するのかを設定します。

このコマンドのが設定する平滑過渡の比率は、現在のプロジェクト実行においてのみ有効です。設定していない場合のデフォルト値は0です。



必須パラメータ：

R：連続経路制御の比率。値の範囲：[0, 100]

例：

```
-- ロボットはP1からP3への移動時にP2を通過し、50%スムーズに遷移します。
CP(50)
MovL(P1)
MovL(P2)
MovL(P3)
```

VelJ

プロトタイプ：

```
VelJ(R)
```

説明：

関節運動方式（MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ）の速度を設定します。

このコマンドが設定する速度は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：速度。値の範囲：[1,100]

例：

```
-- ロボットは速度20%でP1点に移動します。
VelJ(20)
MovJ(P1)
```

AccJ

プロトタイプ：

```
AccJ(R)
```

説明：

関節運動方式（MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ）の加速度を設定します。

このコマンドが設定する加速度は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：加速度。値の範囲：[1,100]

例：

```
-- ロボットは加速度50%でP1点に移動します。
AccJ(50)
MovJ(P1)
```

VelL

プロトタイプ：

```
VelL(R)
```

説明：

直線および曲線運動方式（MovL/Arc/Circle/MovLIO/RelMovLTool/RelMovLUser）の速度を設定します。

このコマンドが設定する速度は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：速度。値の範囲：[1,100]

例：

```
-- ロボットは速度20%でP1点に移動します。
VelL(20)
MovL(P1)
```

AccL

プロトタイプ：

```
AccL(R)
```

説明：

直線および曲線運動方式（MovL/Arc/Circle/MovLIO/RelMovLTool/RelMovLUser）の加速度を設定します。

このコマンドが設定する加速度は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：加速度。値の範囲：[1,100]

例：

```
-- ロボットは加速度50%でP1点に移動します。
AccL(50)
MovL(P1)
```

SpeedFactor

プロトタイプ：

```
SpeedFactor(ratio)
```

説明：

ロボットのグローバル的な運動速度を設定します。詳細は運動命令の一般的な説明を参照してください。

この命令で設定されたグローバル的な速度は現在のプロジェクトの実行中にのみ有効で、設定されていない場合は、スクリプトの実行前の制御ソフトウェア（制御ソフトウェアでスクリプトを実行する場合）またはTCP命令（TCP命令でスクリプトを実行する場合）の設定値を引き続き使用します。

必須パラメータ：

ratio：移動速度。値の範囲：(0,100)

例：

```
-- グローバルの動作速度を50%に設定します。
SpeedFactor(50)
```

SetPayload

プロトタイプ：

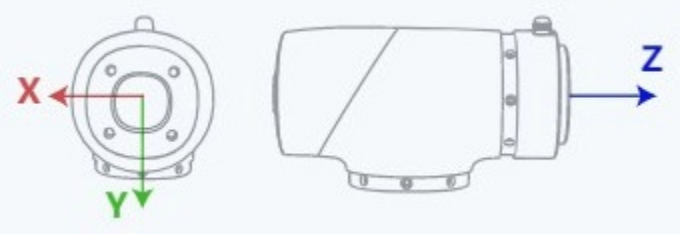
```
SetPayload(payload, {x, y, z})
SetPayload(name)
```

説明：

エンド負荷の重量と偏心座標を設定します。直接の設定と、事前設定パラメータセット方式の2種類の設定をサポートしています。このコマンドで設定したパラメータは、現在のプロジェクトの実行中のみ有効で、前の設定を上書きします。プロジェクトが停止すると、元の設定に戻ります。

必須選択パラメータ**1**：

- payload：エンドの負荷重量。単位：kg
- {x, y, z}：エンドエフェクタの負荷の偏心座標。座標軸の方向は以下の図を参照してください。単位：mm



例**1**：

```
-- エンドエフェクタの負荷の重量を1.5kg、偏心座標を{0, 0, 10}に設定します。
SetPayload(1.5, {0, 0, 10})
```

必須パラメータ**2**：

- name：事前設定パラメータセット名称で、まず制御ソフトウェアで対応するパラメータセットを保存しなければなりません。

例2：

```
-- load1という名前のプリセット負荷パラメータグループを使用して、エンドエフェクタの負荷を設定します。
SetPayload("load1")
```

User

プロトタイプ：

```
User(user)
```

説明：

グローバルユーザ座標系を設定します。ユーザがその他命令を呼び出す場合、オプションパラメータによりユーザ座標系を指定しない場合、システムはグローバルユーザ座標系を使用することができます。

このコマンドが設定するグローバルユーザ座標系は、現在のプロジェクト実行中でのみ有効です。設定していない場合には、デフォルトのグローバルユーザ座標系はユーザ座標系0です。

必須パラメータ：

user：切り換え後のユーザ座標系で、インデックス番号により指定します。まず制御ソフトウェアで対応する座標系を追加してください。指定した座標系のインデックスが存在しない場合、プロジェクトの実行が停止し、エラーが報告されます。

例：

```
-- グローバルのユーザ座標系をユーザ座標系1に切り替えます。
User(1)
```

SetUser

プロトタイプ：

```
SetUser(index,table)
```

説明：

指定したユーザ座標系を修正します。このコマンドで変更した座標系は、現在のプロジェクトの実行中のみ有効です。プロジェクトが停止すると、元の値に戻ります。

必須パラメータ：

- index：ユーザ座標系のインデックス。取りうる値：[0,50]。座標系0の初期値はベース座標系です。
- table：変更後のユーザ座標系。形式は{x, y, z, rx, ry, rz}です。CalcUser命令を使用して取得することを推奨します。

例：

```
-- ユーザ座標系1をCalcUserで計算された新しい座標系に変更します。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
SetUser(1,newUser)
```

CalcUser

プロトタイプ：

```
CalcUser(index,matrix_direction,table)
```

説明：

ユーザ座標系を計算します。

必須パラメータ：

- index：ユーザ座標系のインデックス。取りうる値：[0,50]。ユーザ座標系0の初期値はベース座標系です。
- matrix_direction：計算の方向。
  - 1：左からかけることです。indexで指定した座標系がベース座標系に沿ってtableで指定した値だけ偏向します。
  - 0：右からかけることです。indexで指定した座標系が自身に沿ってtableで指定した値だけ偏向します。
- table：ユーザ座標系の偏移値で、形式は{x, y, z, rx, ry, rz}。

戻る：

計算で得られたユーザ座標系で、形式は{x, y, z, rx, ry, rz}。

例：

```
-- 以下の計算過程は次のように等価です：初期姿勢がユーザ座標系1と同じ座標系が、ベース座標系に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newUserが得られます。
newUser = CalcUser(1,1,{10,10,10,10,10,10})
```

```
-- 以下の計算過程は次のように等価です：初期姿勢がユーザ座標系1と同じ座標系が、ユーザ座標系1に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newUserが得られます。
newUser = CalcUser(1,0,{10,10,10,10,10,10})
```

Tool

プロトタイプ：

```
Tool(tool)
```

説明：

グローバルツール座標系を切り替えます。ユーザがその他命令を呼び出す場合、オプションパラメータによりツール座標系を指定しない場合、システムはグローバルツール座標系を使用します。

このコマンドが設定するグローバルツール座標系は、現在のプロジェクト実行中でのみ有効です。設定していない場合には、デフォルトのグローバルツール座標系は、ツール座標系0です。

必須パラメータ：

tool：切り換え後の工具座標系で、インデックス番号により指定します。まず制御ソフトウェア中に対応する座標系を追加してください。指定した座標系のインデックスが存在しない場合、プロジェクトの実行が停止し、エラーが報告されます。

例：

```
-- グローバルのツール座標系をツール座標系1に切り替えます。
Tool(1)
```

SetTool

プロトタイプ：

```
SetTool(index,table)
```

説明：

指定したツール座標系を修正します。このコマンドで変更した座標系は、現在のプロジェクトの実行中のみ有効です。プロジェクトが停止すると、元の値に戻ります。

必須パラメータ：

- index：ツール座標系のインデックス。取りうる値：[0,50]。ここで、座標系0の初期値はフランジ座標系（TCP0）です。
- table：変更後のツール座標系。形式は{x, y, z, rx, ry, rz}です。CalcTool命令を使用して取得することを推奨します。

例：

```
-- ツール座標系1をCalcToolで計算された新しい座標系に変更します。
newTool = CalcTool(1,1,{10,10,10,10,10,10})
SetTool(1,newTool)
```

CalcTool

プロトタイプ：

```
CalcTool(index,matrix_direction,table)
```

説明：

工具座標系を計算します。

必須パラメータ：

- index：ツール座標系のインデックス。取りうる値：[0,50]。座標系0の初期値はフランジ座標系（TCP0）に基づいています。
- matrix_direction：計算の方向。
  - 1：左からかけることです。indexで指定した座標系がフランジ座標系（TCP0）に沿ってtableで指定した値だけ偏向します。
  - 0：右からかけることです。indexで指定した座標系が自身に沿ってtableで指定した値だけ偏向します。
- table：工具座標系で、形式は{x, y, z, rx, ry, rz}。

戻る：

計算で得られた工具座標系で、形式は{x, y, z, rx, ry, rz}。

例：

```
-- 以下の計算過程は次のように等価です：初期姿勢がツール座標系1と同じ座標系が、フランジ座標系（TCP0）に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newToolが得られます。
newTool = CalcTool(1,1,{10,10,10,10,10,10})
```

```
-- 以下の計算過程は次のように等価です：初期姿勢がツール座標系1と同じ座標系が、ツール座標系1に沿って{x=10, y=10, z=10}だけ平行移動し、{rx=10, ry=10, rz=10}だけ回転すると、新しい座標系newToolが得られます。
newTool = CalcTool(1,0,{10,10,10,10,10,10})
```

GetPose

プロトタイプ：

```
GetPose(user_index, tool_index)
```

説明：

ロボットのリアルタイムの姿勢を取得します。

2つの移動命令間でこの命令を呼び出す場合、得られるポイントはスムーズ内移動設定の影響を受けます。

- スムーズ移動機能（cpまたはrは0）が閉じている場合、目標点を正確に取得することができます。
- スムーズ移動機能が起動している場合、移動曲線上のポイントを得ることができます。

選択可能なパラメータ：

- user_index：姿勢に対応するユーザ座標系インデックスで、まず制御ソフトウェア中で対応する座標系を追加してください。設定していない場合、グローバルユーザ座標系を使用します。
- tool_index：姿勢に対応する工具座標系インデックスで、まず制御ソフトウェア中で対応する座標系を追加してください。設定していない場合、グローバルユーザ座標系を使用します。

戻る：

ロボットの現在の姿勢：{pose = {x, y, z, rx, ry, rz}}

例：

```
-- ロボットはまずP1に移動し、その後現在の姿勢に戻ります。
local currentPose = GetPose()
MovJ(P1)
MovJ(currentPose)
```

```
-- ロボットがP1に移動した後、P2に移動し、P1の姿勢を取得します。
MovJ(P1,{cp=0})
local currentPose = GetPose() --P1の姿勢を取得
MovJ(P2)
```

GetAngle

プロトタイプ：

```
GetAngle()
```

説明：

ロボットのリアルタイムの関節角度を取得します。

2つの移動命令間でこの命令を呼び出す場合、得られる間接角度はスムーズ移動設定の影響を受ける場合があります。

- スムーズ移動機能（cpまたはrは0）が閉じている場合、目標点に対応する関節角度を正確に取得することができます。
- スムーズ移動機能が起動している場合、移動曲線上のポイントに対応する関節角度を得ることができます。

戻る：

ロボットの現在の間接角度：{joint = {j1, j2, j3, j4, j5, j6} }

例：

```
-- ロボットはまずP1に移動し、その後現在の姿勢に戻ります。
local currentAngle = GetAngle()
MovJ(P1)
MovJ(currentAngle)
```

```
-- ロボットがP1に移動した後、P2に移動し、P1の関節角度を取得します。
MovJ(P1,{cp=0})
local currentAngle = GetAngle()  --P1の関節角度を取得する
MovJ(P2)
```

GetABZ

プロトタイプ：

```
GetABZ()
```

説明：

エンコーダの現在の位置を取得します。

戻り値：

エンコーダの現在の位置。

例：

```
-- 設定されたABZエンコーダーの現在位置を取得し、変数abzに割り当てます。
local abz = GetABZ()
```

CheckMovJ

プロトタイプ：

```
CheckMovJ(P, {user = 1, tool = 0, a = 20, v = 50, cp = 100})
```

説明：

現在点から目標点までの関節運動の実行可能性を確認します。システムは全体の運動経路を計算し、経路中に到達不能な点がないかを確認します。

必須パラメータ：

P：目標点。

選択可能なパラメータ：

- user：目標点のユーザ座標系。
- tool：目標点の工具座標系。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度。値の範囲：(0,100]
- cp：スムーズ移動比率。値の範囲：[0,100]

詳細は一般説明を参照してください。

戻る：

チェック結果。

- 0：エラーなし
- 16：終点がショルダーの特異点に近接している
- 17：終点逆計算が解なし
- 18：終点逆計算が位置制限される
- 22：ジェスチャー切り替えエラー
- 26：終点が腕部の特異点に近接している
- 27：終点が肘部の特異点に近接している
- 29：速度パラメータエラー
- 32：軌跡にショルダーの特異点がある
- 33：軌跡に逆計算解なし点が存在する
- 34：軌跡に逆計算位置制限点が存在する
- 35：軌跡に腕部の特異点がある

- 36：軌跡に肘部の特異点がある
- 37：軌跡に関節ジャンピング点が存在する

例：

```
-- 関節運動のデフォルト設定を使用してP1に到達可能かどうかを確認します。到達可能な場合、関節運動でP1に移動します。
local status=CheckMovJ(P1)
if(status==0)
then
    MovJ(P1)
    status=CheckMovJ(P2) -- P1からP2への実行可能性の確認は、ロボットがP1に移動した後に実行する必要があります。
    if(status==0)
    then
        MovJ(P2)
    end
end
```

CheckMovL

プロトタイプ：

```
CheckMovL(P, {user = 1, tool = 0, a = 20, v = 50, cp = 100, r = 5})
```

説明：

現在点から目標点までの直線運動の実行可能性を確認します。システムは全体の運動経路を計算し、経路中に到達不能な点がないかを確認します。

必須パラメータ：

P：目標点。

選択可能なパラメータ：

- user：目標点のユーザ座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- tool：目標点の工具座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- a：この命令を実行する場合のロボットの移動加速度。値の範囲：(0,100]
- v：この命令を実行する場合のロボットの移動速度で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する際のロボットの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：スムーズ移動比率で、rと相互に排他的です。値の範囲：[0,100]
- r：スムーズ移動半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲ア：[0,100]、単位：mm

詳細は一般説明を参照してください。

戻る：

チェック結果。

- 0：エラーなし
- 16：終点がショルダーの特異点に近接している
- 17：終点逆計算が解なし
- 18：終点逆計算が位置制限される
- 22：ジェスチャー切り替えエラー
- 26：終点が腕部の特異点に近接している
- 27：終点が肘部の特異点に近接している
- 29：速度パラメータエラー
- 32：軌跡にショルダーの特異点がある
- 33：軌跡に逆計算解なし点が存在する
- 34：軌跡に逆計算位置制限点が存在する
- 35：軌跡に腕部の特異点がある
- 36：軌跡に肘部の特異点がある
- 37：軌跡に関節ジャンピング点が存在する

例：

```
-- 直線運動のデフォルト設定を使用してP1に到達可能かどうかを確認します。到達可能な場合、直線運動でP1に移動します。
local status=CheckMovL(P1)
if(status==0)
then
    MovL(P1)
    status=CheckMovL(P2) -- P1からP2への実行可能性の確認は、ロボットがP1に移動した後に実行する必要があります。
    if(status==0)
    then
        MovL(P2)
    end
end
```

SetSafeWallEnable

プロトタイプ：

```
SetSafeWallEnable(index,value)
```

説明：

指定された安全壁を起動または終了します。このコマンドで設定した安全壁の状態は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

- index：設定する安全壁インデックスで、まず制御ソフトウェア中で対応する安全壁を追加してください。値の範囲：[1,8]
- value：
  - true：安全壁を開く
  - false：安全壁を閉じる

例：

```
-- インデックス1の安全壁を開きます。
SetSafeWallEnable(1,true)
```

SetWorkZoneEnable

プロトタイプ：

```
SetWorkZoneEnable(index,value)
```

説明：

指定した干渉区を開くまたは閉じます。このコマンドで設定した干渉区域の状態は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

- index：設定する干渉区域のインデックス。事前に制御ソフトウェアで対応する干渉区域を追加する必要があります。取りうる値の範囲：[1,6]
- value：
  - true：干渉区域を開く
  - false：干渉区域を閉じる

例：

```
-- インデックス1の干渉区域を開きます。
SetWorkZoneEnable(1,true)
```

SetCollisionLevel

プロトタイプ：

```
SetCollisionLevel(level)
```

説明：

衝突検出レベルを設定します。このコマンドで設定した値は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

level：衝突検出等級で、0は衝突検出終了、1~5の数字では、大きくなるほど感度が高くなります

例：

```
-- 衝突検出レベルを3に設定します。
SetCollisionLevel(3)
```

SetBackDistance

プロトタイプ：

```
SetBackDistance(distance)
```

説明：

ロボットが衝突を検出してから元のルートに戻るまでの距離を設定します。このコマンドで設定した値は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

distance：衝突して戻る距離で、値の範囲は：[0,50]、単位：mm

例：

```
-- 衝突後退距離を20mmに設定します。
SetBackDistance(20)
```

IO

命令リスト

IO命令は、ロボットアームシステムのIOの読み書きおよび関連パラメータの設定を行うために使用されます。

命令	機能
DI	DIポートのステータスの取得
DIGroup	複数のDIポートの状態の取得
DO	デジタル出力ポートの状態を設定
DOGroup	複数のデジタル出力ポートの状態を設定
GetDO	デジタル出力ポートの現在の状態の取得
GetDOGroup	複数のデジタル出力ポートの現在の状態の取得
AI	AIポートの値の取得
AO	アナログ出力ポートの値を設定
GetAO	アナログ出力ポートの現在の値の取得

DI

プロトタイプ：

```
DI(index)
```

説明：

デジタル入力ポートの状態を読み込みます。

必須パラメータ：

index：DI端子の番号。

戻る：

対応するDI端子の状態（ON/OFF）。

例：

```
-- DI1がONの場合、ロボットアームは直線移動でP1に移動します。
if (DI(1)==ON)
then
```

```
    MovL(P1)
end
```

DIGroup

プロトタイプ：

```
DIGroup(index1,...,indexN)
```

説明：

複数のデジタル入力ポートの状態を読み取ります。

必須パラメータ：

index：DI端子の番号。コンマで区切って複数入力できます。

戻る：

対応するDI端子の状態（ON/OFF）を配列として返します。

例：

```
-- DI1とDI2の両方がONの場合、ロボットアームは直線移動でP1に移動します。
local digroup = DIGroup(1,2)
if (digroup[1]&digroup[2]==ON)
then
    MovL(P1)
end
```

DO

プロトタイプ：

```
DO(index,ON|OFF,time_ms)
```

説明：

デジタル出力ポートの状態を設定します。

必須パラメータ：

- index：DO端子の番号。
- ON|OFF：設定するDOポートの状態。

選択可能なパラメータ：

time_ms：[25, 60000]の範囲で、継続的な出力時間（ms）。このパラメータを設定すると、指定した時間後にDOが自動的に反転されます。反転は非同期動作であり、命令キューをブロックすることはありません。DO出力が実行されると次の命令が実行されます。

例：

```
-- DO1をONに設定します。
DO(1,ON)
```

```
-- DO1をONに設定し、50ms後に自動的にOFFに設定します。
DO(1,ON,50)
```

DOGroup

プロトタイプ：

```
DOGroup({index1,ON|OFF},..,{indexN,ON|OFF})
```

説明：

複数のデジタル出力ポートの状態を設定します。

必須パラメータ：

- index：DO端子の番号。
- ON|OFF：設定するDOポートの状態。

複数のグループを設定できます。各グループは中括弧で囲まれ、グループの間はコンマで区切られます。

例：

```
-- DO1とDO2をONに設定します。
DOGroup({1,ON},{2,ON})
```

GetDO

プロトタイプ：

```
GetDO(index)
```

説明：

デジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：DO端子の番号。

戻る

対応するDI端子の状態（ON/OFF）。

例：

```
-- DO1の現在の状態を取得します。
GetDO(1)
```

GetDOGroup

プロトタイプ：

```
GetDOGroup(index1,...,indexN)
```

説明：

複数のデジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：DO端子の番号。コンマで区切って複数入力できます。

戻る：

対応DO端子の状態（ON/OFF）を配列として返します。

例：

```
-- DO1とDO2の現在の状態を取得します。
GetDOGroup(1,2)
```

AI

プロトタイプ：

```
AI(index)
```

説明：

アナログ入力ポートの値を読み込みます。

必須パラメータ：

index：AI端子の番号。

戻る：

対応するAI端子の値。

例：

```
-- AI1の値を読み取り、変数testに代入します。
test = AI(1)
```

AO

プロトタイプ：

```
AO(index,value)
```

説明：

アナログ出力ポートの値を設定します。

必須パラメータ：

- index：AO端子の番号。
- value：設定する値。電圧範囲：[0,10]、単位：V、電流範囲：[4,20]、単位：mA

例：

```
-- AO1の出力値を2に設定します。
AO(1,2)
```

GetAO

プロトタイプ：

```
GetAO(index)
```

説明：

アナログ出力ポートの現在の値を取得します。

必須パラメータ：

index：AO端子の番号。

戻る

対応するAO端子の値。

例：

```
-- AO1の現在の状態を取得します。
GetAO(1)
```

エンドツール

命令リスト

エンドツール命令は、ロボットシステムのエンドIOの読み取り/書き込みおよび関連パラメータの設定に使用します。

命令	機能
ToolDI	エンドデジタル入力ポートの状態を読み取ります
ToolDO	エンドデジタル出力ポートの状態を設定します(シーケンス命令)
GetToolDO	エンドデジタル出力ポートの現在の状態を取得します
ToolAI	エンドアナログ入力ポートの値を読み取ります
SetToolMode	エンド端子多重化通信モードを設定します
SetToolPower	エンドツールの電源状態を設定します
SetTool485	エンドツールのRS485インターフェースに対応するデータフォーマットを設定します

ToolDI

プロトタイプ：

```
ToolDI(index)
```

説明：

エンドデジタル入力ポートの状態を読み取ります。

必須パラメータ：

index：エンドDI端子の番号。

戻る：

対応するDI端子の状態（ON/OFF）。

例：

```
-- エンドDI1がONであるとき、ロボットが直線移動方式でP1点に移動します。
if (ToolDI(1)==ON)
then
    MovL(P1)
end
```

ToolDO

プロトタイプ：

```
ToolDO(index,ON|OFF)
```

説明：

エンドデジタル出力ポートの状態を設定します。

必須パラメータ：

- index：エンドDO端子の番号。
- ON|OF：設定するDOポートの状態。

例：

```
-- エンドDO1をONに設定します。
ToolDO(1,ON)
```

GetToolDO

プロトタイプ：

```
GetToolDO(index)
```

説明：

エンドデジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：エンドDO端子の番号。

戻る

対応するエンドDO端子の状態（ON/OFF）。

例：

```
-- エンドDO1の現在の状態を取得します。
GetToolDO(1)
```

ToolAI

プロトタイプ：

```
ToolAI(index)
```

説明：

エンドアナログ入力ポートの値を読み取ります。使用前に、SetToolModeを通じて端子をアナログ入力モードに設定する必要があります。

> 説明：
> エンドAIインターフェースのないロボットアームの場合は、このインターフェースを呼び出しても効果はありません。

必須パラメータ：

index：エンドAI端子の番号。

戻る：

対応するAI端子の値。

例：

```
-- エンドAI1の値を読み取り、その値を変数testに割り当てます。
test = ToolAI(1)
```

SetToolMode

プロトタイプ：

```
SetToolMode(mode,type,identify)
```

説明：

機械アームの末端AI1とAI2インターフェースと485インターフェースが共有されているモデル（CRとCR Aシリーズ）では、このインターフェースを通じて末端の共有端子のモードを設定できます。デフォルトモードは485モードです。

> 説明：
> エンドエフェクタモードの切り替えをサポートしていないロボットでは、このインターフェースの呼び出しは効果がありません。

必須パラメータ：

- mode：端子多重化のモード

    - 1：485モード。
    - 2：アナログ入力モード。
- type：modeが1であるとき、該パラメータは無効です。Modeが2であるとき、該パラメータはアナログ入力のモードの設定に使用します。一の位はAI1のモードを表し、十の位はAI2のモードを表します。十の位が0であるとき、一の位のみを入力できます。

  モード:

    - 0：0～10Vの電圧入力モード
    - 1：電流収集モード
    - 2：0~5V電圧入力モード 見本：
    - 0：AI1とAI2はいずれも0~10V電圧入力モード
    - 1：AI2は0~10V電圧入力モード、AI1は電流収集モード
    - 11：AI2とAI1はいずれも電流収集モード
    - 12：AI2は電流収集モード、AI1は0~5V電圧入力モード
    - 20：AI2は0~5V電圧入力モード、AI1は0~10V電圧入力モード

選択可能なパラメータ

identify：ロボットに複数のエンド航空コネクタがあるとき、航空コネクタの指定に使用します。1は航空コネクタ1を表し、2は航空コネクタ2を表します。

例：

```
-- エンド端子多重化をアナログ入力に設定し、2つの入力パスはいずれも0~10V電圧入力モードです。
SetToolMode(2,0)
```

SetToolPower

プロトタイプ：

```
SetToolPower(status)
```

説明：

エンドエフェクタの電源供給状態を設定します。通常、エンドエフェクタの電源を再起動するために使用されます。例えば、エンドエフェクタのクローを再電源化して初期化します。このインターフェースを連続して呼び出す場合は、少なくとも4ms以上の間隔をお勧めします。

必須パラメータ：

status：エンドツールの電源状態。0：電源オフ、1：電源オン

例：

```
-- エンドツールの電源を再起動します。
```

```
SetToolPower(0)
Wait(5)
SetToolPower(1)
```

SetTool485

プロトタイプ：

```
SetTool485(baud,parity,stopbit,identify)
```

説明：

エンドツールのRS485インターフェースに対応するデータフォーマットを設定します。

> 説明：
> エンド485インターフェースのないロボットアームの場合は、このインターフェースを呼び出しても効果はありません。

必須パラメータ：

baud：RS485インターフェースのビットレート

選択可能なパラメータ：

- parity：パリティビットかどうか。「O」は奇数パリティを、「E」は偶数パリティを、「N」はパリティビットなしを示します。デフォルト値は「N」です。
- stopbit：ストップビット長さ。値の範囲は1と2です。デフォルト値は1とします。
- identify：ロボットに複数のエンド航空コネクタがあるとき、航空コネクタの指定に使用します。1は航空コネクタ1を表し、2は航空コネクタ2を表します。

例：

```
-- エンドツールのRS485インターフェースのボーレートを115200Hzに設定します。パリティビットはなく、ストップビットの長さは1にします。
SetTool485(115200,"N",1)
```

TCP&UDP

命令リスト

TCP&UDP関数は、TCPやUDP通信を行うために使用します。

命令	機能
TCPCreate	TCPネットワークオブジェクトを作成します
TCPStart	TCP接続を確立します
TCPRead	TCP相手側が送信したデータを受信します
TCPWrite	TCP相手側にデータを送信します
TCPDestroy	TCP接続を切断し、socketオブジェクトを廃棄します
UDPCreate	UDPネットワークオブジェクトを作成します
UDPRead	UDP相手側が送信したデータを受信します
UDPWrite	UDP相手側にデータを送信します

TCPCreate

プロトタイプ：

```
TCPCreate(isServer, IP, port)
```

説明：

TCPネットワークオブジェクトは、1つのみ作成することができます。

必須パラメータ：

- isServer：サーバーを作成するかどうか。true：サーバーを作成することを表します。false：クライアントを作成することを表します。
- IP：サーバーIPアドレス。クライアントIPアドレスと同じネットワークセグメント上にあり、かつ競合しない必要があります。サーバー作成時はロボットのIPアドレスとし、クライアント作成時は相手側のアドレスとします。
- port：サーバーポート。システムによって占用されている下記ポートは使用しないでください。これらのポートを使用すると、サーバー作成が失敗する恐れがあります。

  7、13、22、37、139、445、502、503（0~1024の間のポートは、linuxシステムのカスタマポートであり、占用されている可能性が高いため、できるだけ使用しないでください）。

1501, 1502, 1503, 4840, 8172, 9527,

11740, 22000, 22001, 29999, 30004, 30005, 30006,

60000~65504, 65506, 65511~65515, 65521, 65522

戻る：

- err：0はTCPネットワークオブジェクト作成が成功したことを表し、1はTCPネットワークオブジェクト作成が失敗したことを表します。
- socket：作成したsocketオブジェクト。

例**1**：

```
-- TCPサーバーを作成します。
local ip="192.168.5.1" -- ロボットのIPアドレスをサーバーのIPアドレスとします
local port=6001 -- サーバーポート
local err=0
local socket=0
err, socket = TCPCreate(true, ip, port)
```

例**2**：

```
-- TCPクライアントを作成します。
local ip="192.168.5.25" -- カメラなどの外部設備のIPアドレスをサーバーのIPアドレスとします
local port=6001 -- サーバーポート
local err=0
local socket=0
err, socket = TCPCreate(false, ip, port)
```

TCPStart

プロトタイプ：

```
TCPStart(socket, timeout)
```

説明：

TCP接続を確立します。ロボットをサーバーとするとき、クライアントが接続するのを待機します。ロボットをクライアントとするとき、サーバーに自発的に接続します

必須パラメータ：

- socket：作成したsocketオブジェクト。
- timeout：待機タイムアウト時間（秒）。0に設定した場合、接続の確立が成功するまで待機します。0ではない場合、設定した時間を超えると接続に失敗したと返されます。

戻る：

接続結果。

- 0：接続は成功しました
- 1：入力パラメータエラー
- 2：socketオブジェクトが存在しません
- 3：設定タイムアウト時間エラー
- 4：接続に失敗しました

例：

```
-- TCP接続の確立を開始し、接続の確立が成功するまで待機します。
err = TCPStart(socket, 0) -- socketは、TCPCreate成功で返されるsocketオブジェクトです
```

TCPRead

プロトタイプ：

```
TCPRead(socket, timeout, type)
```

説明：

TCP相手側が送信したデータを受信します。

必須パラメータ：

socket：作成したsocketオブジェクト。

選択可能なパラメータ：

- timeout：待機タイムアウト時間（秒）。設定しないか0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定していない場合、RecBufバッファ形式はtable形式とします。「string」に設定した場合、RecBufバッファは文字列です。

戻る：

- err：0はデータの受信が成功したことを表し、1はデータの受信が失敗したことを表します。
- Recbuf：受信データバッファエリア。

例：

```
-- TCPデータを受信します。受信したデータをそれぞれ文字列とtable形式で保存します。
-- socketは、TCPCreate成功で返されるsocketオブジェクトです
err, RecBuf = TCPRead(socket,0,"string") -- RecBufデータタイプは文字列です
err, RecBuf = TCPRead(socket, 0) -- RecBufデータタイプはtableです
```

TCPWrite

プロトタイプ：

```
TCPWrite(socket, buf, timeout)
```

説明：

TCP相手側にデータを送信します。

必須パラメータ：

- socket：作成したsocketオブジェクト。
- buf：送信対象のデータ。

選択可能なパラメータ：

timeout：待機タイムアウト時間（秒）。設定しないか0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

結果を送信します。

- 0：送信は成功しました。
- 1：送信は失敗しました

例：

```
-- TCPデータを送信します。データコンテンツは「test」とします。
TCPWrite(socket, "test") -- socketは、TCPCreate成功で返されるsocketオブジェクトです
```

TCPDestroy

プロトタイプ：

```
TCPDestroy(socket)
```

説明：

TCP接続を切断し、socketオブジェクトを廃棄します。

必須パラメータ：

socket：作成したsocketオブジェクト。

戻る：

実行結果。

- 0：実行は成功しました
- 1：実行は失敗しました

例：

```
-- TCP相手側との接続を切断します。
TCPDestroy(socket) -- socketは、TCPCreate成功で返されるsocketオブジェクトです
```

UDPCreate

プロトタイプ：

```
UDPCreate(isServer, IP, port)
```

説明：

UDPネットワークオブジェクトは、1つのみ作成することができます。

必須パラメータ：

- isServer：固定的にfalseを記入します。
- IP：相手側IPアドレス。ロボットIPアドレスと同じネットワークセグメント上にあり、かつ競合しない必要があります。
- port：相手側ポート。

戻る：

- err：0はUDPネットワークオブジェクト作成が成功したことを表し、1はTCPネットワークオブジェクト作成が失敗したことを表します。
- socket：作成したsocketオブジェクト。

例：

```
-- UDPネットワークオブジェクトを作成します。
local ip="192.168.5.25" -- カメラなどの外部設備のIPアドレスを相手側のIPアドレスとします
local port=6001 -- 相手側ポート
local err=0
local socket=0
err, socket = UDPCreate(false, ip, port)
```

UDPRead

プロトタイプ：

```
UDPRead(socket, timeout, type)
```

説明：

UDP相手側が送信したデータを受信します。

必須パラメータ：

socket：作成したsocketオブジェクト。

選択可能なパラメータ：

- timeout：待機タイムアウト時間（秒）。設定しないか0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定していない場合、RecBufバッファ形式はtable形式とします。「string」に設定した場合、RecBufバッファは文字列です。

戻る：

- err：0はデータの受信が成功したことを表し、1はデータの受信が失敗したことを表します。
- Recbuf：受信データバッファエリア。

例：

```
-- UDPデータを受信します。受信したデータをそれぞれ文字列とtable形式で保存します。
-- socketは、UDPCreate成功で返されるsocketオブジェクトです
err, RecBuf = UDPRead(socket,0,"string") -- RecBufデータタイプは文字列です
err, RecBuf = UDPRead(socket, 0) -- RecBufデータタイプはtableです
```

UDPWrite

プロトタイプ：

```
UDPWrite(socket, buf, timeout)
```

説明：

UDP相手側にデータを送信します。

必須パラメータ：

- socket：作成したsocketオブジェクト。
- buf：送信対象のデータ。

選択可能なパラメータ：

timeout：待機タイムアウト時間（秒）。設定しないか0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

結果を送信します。

- 0：送信は成功しました。
- 1：送信は失敗しました

例：

```
-- UDPデータを送信します。データコンテンツは「test」とします。
UDPWrite(socket, "test") -- socketは、UDPCreate成功で返されるsocketオブジェクトです
```

Modbus

命令リスト

Modbus関数は、Modbusマスタとスレーブ間の通信を確立するために使用されます。レジスタアドレスの取得範囲と定義については、対応するスレーブのModbusレジスタアドレス定義説明をご参照ください。

命令	機能
ModbusCreate	Modbusマスターを作成
ModbusRTUCreate	RS485インターフェースを基にしたModbusマスターを作成
ModbusClose	Modbusスレーブとの接続を解除
GetInBits	Modbusマスターを作成
GetInRegs	入力レジスタの読み取り
GetCoils	コイルレジスタの読み取り
SetCoils	コイルレジスタの書き込み
GetHoldRegs	保持レジスタの読み取り
SetHoldRegs	保持レジスタの書き込み

各種レジスタに対応するModbus機能コードは、標準のModbusプロトコルに従います。

レジスタタイプ	読み取りレジスタ	単一レジスタの書き込み	複数レジスタの書き込み
コイルレジスタ	01	05	0F
接触点レジスタ	02	-	-
入力レジスタ	04	-	-
ホールディングレジスタ	03	06	10

ModbusCreate

プロトタイプ：

```
ModbusCreate(IP, port, slave_id, isRTU)
```

説明：

Modbusマスターを作成し、スレーブとの接続を確立します。最大で15の機器との接続を同時にサポートします。

ロボットが搭載しているスレーブステーションに接続する場合は、IPをロボットのIP（デフォルトは192.168.5.1、変更可能）に設定し、ポートを502（map1）または1502（map2）に設定します。詳細は付録A Modbusレジスタの定義を参照してください。

第三者のスレーブに接続する場合、IPとポートは第三者のスレーブのアドレスであり、レジスタを読み書きする際のレジスタアドレスの範囲と定義は、対応するスレーブのModbusレジスタアドレス定義説明を参照してください。

必須パラメータ：

- IP：スレーブのIPアドレス。
- port：スレーブのポート。

選択可能なパラメータ：

- slave_id：スレーブID。
- isRTU：ブール値。携帯していない、またはfalseである場合、ModbusTCP通信を確立します。trueである場合、ModbusRTU通信を確立します。

> ⚠ 注意：
> このパラメータは、接続確立後のデータ転送に使用するプロトコル形式を決定しますが、接続結果には影響しません。したがって、マスタを作成する際にこのパラメータを誤って設定した場合でも、作成は成功しますが、その後の通信では異常が発生する可能性があります。

戻る：

- err：
  - 0：Modbusマスターステーションの作成に成功しました
  - 1：作成したマスターステーションが最大数に達しているため、新しいマスターステーションの作成に失敗しました
  - 2：マスターステーションの初期化に失敗しました。IP、ポート、ネットワークの状態などを確認することをお勧めします
  - 3：スレーブステーションへの接続に失敗しました。スレーブステーションが正常に設立されているか、ネットワークが正常であるかなどを確認することをお勧めします
- id：返されたマスターインデックスで、その他Modbus命令を後で呼び出すときに使用します。値の範囲は[0, 14]

例：

```
-- Modbusマスターステーションを作成し、指定したスレーブと接続します。
local ip="192.168.5.123"
```

```
local port=503
local err=0
local id=0
err, id = ModbusCreate(ip, port, 1)
```

```
-- Modbusマスターステーションを作成し、ロボットの内蔵スレーブと接続します。
local ip="192.168.5.1"
local port=502
local err=0
local id=0
err, id = ModbusCreate(ip, port)
```

ModbusRTUCreate

プロトタイプ：

```
ModbusRTUCreate(slave_id, baud, parity, data_bit, stop_bit)
```

説明：

RS485インターフェースを基にしたModbusマスターを作成し、スレーブとの接続を確立します。最大で15の機器との接続を同時にサポートします。

必須パラメータ：

- slave_id：スレーブID。
- baud：RS485インターフェースのボーレート。

選択可能なパラメータ：

- parity：パリティビットかどうか。「O」は奇数パリティを、「E」は偶数パリティを、「N」はパリティビットなしを示します。パラメータなしの場合、デフォルト値は“E”です。
- data_bit：データビット長さ。値の範囲は8。パラメータなしの場合、デフォルト値は8です。
- stopbit：ストップビット長さ。値の範囲は1と2です。パラメータなしの場合、デフォルト値は1です。

戻る：

- err：0はModbusマスターの作成に成功したことを示し、1はModbusマスターの作成に失敗したことを示します。
- id：返されたマスターインデックスで、その他Modbus命令を後で呼び出すときに使用します。

例：

```
-- Modbusマスターステーションを作成し、RS485インターフェースに接続されたスレーブと接続します。スレーブIDは1、ボーレートは115200です。
err, id = ModbusRTUCreate(1, 115200)
```

ModbusClose

プロトタイプ：

```
ModbusClose(id)
```

説明：

Modbusスレーブとの接続を解除し、マスターを解放します。

必須パラメータ：

id：既に作成されているマスターインデックス。

戻る：

捜査結果

- 0：接続解除成功。
- 1：接続解除失敗。

例：

```
-- Modbusスレーブとの接続を切断します。
ModbusClose(id)
```

GetInBits

プロトタイプ：

```
GetInBits(id, addr, count)
```

説明：

Modbusスレーブ接点レジスタアドレスの値を読み取ります。Modbus機能コード02に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：接点レジスタの開始アドレス。
- count：接点レジスタの数を連続で読み取る数。値の範囲：1~2000（ModbusTCPプロトコル

の制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

戻る：

接点レジスタアドレスの値で、table注に保存されます。table注の1つ目の値は、接点レジスタ開始アドレスの値に対応します。

例：

```
-- アドレス0から連続して5つのアドレスの値を読み取ります。
inBits = GetInBits(id,0,5)
```

GetInRegs

プロトタイプ：

```
GetInRegs(id, addr, count, type)
```

説明：

指定されたデータタイプに基づき、Modbusスレーブ入力レジスタアドレスの値を読み取ります。Modbus機能コード04に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：入力レジスタの開始アドレス。
- count：入力レジスタの値を連続で読み取る数。値の範囲：1~125（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です。
- U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

戻る：

入力レジスタアドレスの値は、table中に保存されます。table中の1つ目の値は入力レジスタ開始アドレスの値に対応します。

例：

```
-- アドレス2048から32ビットの符号なし整数を1つ読み取ります。
data = GetInRegs(id, 2048, 2, "U32")
```

GetCoils

プロトタイプ：

```
GetCoils(id, addr, count)
```

説明：

Modbusスレーブコイルレジスタアドレスの値を読み取ります。Modbus機能コード01に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：コイルレジスタ開始アドレスのアドレス。
- count：コイルレジスタの値を連続で読み取る数。値の範囲：1~2000（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

戻る：

コイルレジスタのアドレスの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

例：

```
-- アドレス0から連続して5つのアドレスの値を読み取ります。
Coils = GetCoils(id,0,5)
```

SetCoils

プロトタイプ：

```
SetCoils(id, addr, count, table)
```

説明：

指定した値をコイルレジスタが指定するアドレスに書き込みます。Modbus機能コード05（1つ書き込み）と0F（複数書き込み）に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：コイルレジスタの開始アドレス。
- count：コイルレジスタの値を連続で書き込む数。値の範囲：1~1968（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）
- table：書き込むコイルレジスタの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

例：

```
-- アドレス1024から始めて、連続して5つのコイルを書き込みます。
local Coils = {0,1,1,1,0}
SetCoils(id, 1024, #coils, Coils)
```

GetHoldRegs

プロトタイプ：

```
GetHoldRegs(id, addr, count, type)
```

説明：

指定したデータタイプに基づき、Modbusスレーブ保持レジスタアドレスの値を読み取ります。Modbus機能コード03に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：保持レジスタの開始アドレス。
- count：保持レジスタの値を連続で読み取る数。値の範囲：1~125（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です。
- U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

戻る：

保持レジスタアドレスの値は、table中に保存されます。table中の1つ目の値は、保持レジスタの開始アドレスの値に対応します。

例：

```
-- アドレス2048から32ビットの符号なし整数を1つ読み取ります。
data = GetHoldRegs(id, 2048, 2, "U32")
```

SetHoldRegs

プロトタイプ：

```
SetHoldRegs(id, addr, count, table, type)
```

説明：

指定したデータタイプに基づいて、指定した値を保持レジスタが指定するアドレスに書き込みます。Modbus機能コード06（1つ書き込み）と10（複数書き込み）に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：保持レジスタの開始アドレス。
- count：保持レジスタの値を連続で書き込む数。値の範囲：1~123（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）
- table：書き込むコイルレジスタの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です。
- U16：16ビットの符号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占有）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占有）

例：

```
-- アドレス2048から始めて、64ビットの倍精度浮動小数点数を1つ書き込みます。
local data = {95.32105}
SetHoldRegs(id, 2048, 4, data, "F64")
```

バスレジスタ

命令リスト

バスレジスタ命令は、ProfinetまたはEthernet/IPバスレジスタを読み書きするためのものです。

> 説明：
> Magician E6は、この命令セットをサポートしていません。

命令	機能
GetInputBool	入力レジスタが指定するアドレスのbool値を取得します
GetInputInt	入力レジスタが指定するアドレスのint値を取得します
GetInputFloat	入力レジスタが指定するアドレスのfloat値を取得します
GetOutputBool	出力レジスタが指定するアドレスのbool値を取得します
GetOutputInt	出力レジスタが指定するアドレスのint値を取得します
GetOutputFloat	出力レジスタが指定するアドレスのfloat値を取得します
SetOutputBool	出力レジスタが指定するアドレスのbool値を設定します
SetOutputInt	出力レジスタが指定するアドレスのint値を設定します
SetOutputFloat	出力レジスタが指定するアドレスのfloat値を設定します

GetInputBool

プロトタイプ：

```
GetInputBool(address)
```

説明：

入力レジスタが指定するアドレスのboolタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～63]

戻る：

指定されたレジスタアドレスの値は0または1です

例：

```
-- 入力レジスタ0の値が真の場合、後続の操作を実行します。
if(GetInputBool(0))
then
    -- 後続の操作を実行します
end
```

GetInputInt

プロトタイプ：

```
GetInputInt(address)
```

説明：

入力レジスタが指定するアドレスのintタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～23]

戻る：

指定されたレジスタアドレスの値は整数（int32）です

例：

```
-- 入力レジスタ1の値を読み取り、変数regIntに代入します。
local regInt = GetInputInt(1)
```

GetInputFloat

プロトタイプ：

```
GetInputFloat(address)
```

説明：

入力レジスタが指定するアドレスのfloatタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～23]

戻る：

指定されたレジスタアドレスの値は単精度浮動小数点数（float）です

例：

```
-- 入力レジスタ2の値を読み取り、変数regFloatに代入します。
local regFloat = GetInputFloat(2)
```

GetOutputBool

プロトタイプ：

```
GetOutputBool(address)
```

説明：

出力レジスタが指定するアドレスのboolタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～63]

戻る：

指定されたレジスタアドレスの値は0または1です

例：

```
-- 出力レジスタ0の値が真の場合、後続の操作を実行します。
if(GetOutputBool(0))
then
    -- 後続の操作を実行します
end
```

GetOutputInt

プロトタイプ：

```
GetOutputInt(address)
```

説明：

出力レジスタが指定するアドレスのintタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～23]

戻る：

指定されたレジスタアドレスの値は整数（int32）です

例：

```
-- 出力レジスタ1の値を読み取り、変数regIntに代入します。
local regInt = GetOutputInt(1)
```

GetOutputFloat

プロトタイプ：

```
GetOutputFloat(address)
```

説明：

出力レジスタが指定するアドレスのfloatタイプの数値を取得します。

必須パラメータ：

address：レジスタアドレスです。値の範囲[0～23]

戻る：

指定されたレジスタアドレスの値は単精度浮動小数点数（float）です

例：

```
-- 出力レジスタ2の値を読み取り、変数regFloatに代入します。
local regFloat = GetOutputFloat(2)
```

SetOutputBool

プロトタイプ：

```
SetOutputBool(address, value)
```

説明：

出力レジスタが指定するアドレスのboolタイプの数値を設定します。

必須パラメータ：

- address：レジスタアドレスです。値の範囲[0～63]
- value：設定する値です。ブール値または0/1をサポートします。

例：

```
-- 出力レジスタ0の値を偽に設定します。
SetOutputBool(0,0)
```

SetOutputInt

プロトタイプ：

```
SetOutputInt(address, value)
```

説明：

出力レジスタが指定するアドレスのintタイプの数値を設定します。

必須パラメータ：

- address：レジスタアドレスです。値の範囲[0～23]
- value：設定する値です。整数（int32）をサポートします。

例：

```
-- 出力レジスタ1の値を123に設定します。
SetOutputInt(1,123)
```

SetOutputFloat

プロトタイプ：

```
SetOutputFloat(address, value)
```

説明：

出力レジスタが指定するアドレスのfloatタイプの数値を設定します。

必須パラメータ：

- address：レジスタアドレスです。値の範囲[0～23]
- value：設定する値です。単精度浮動小数点数（float）をサポートします。

例：

```
-- 出力レジスタ2の値を12.3に設定します。
SetOutputFloat(2,12.3)
```

プログラム制御

命令リスト

プログラム制御関数は、プログラム実行の制御に関連する汎用関数です。

命令	機能
Print	デバッグ情報を制御コンソールにプリントアウトします。
Log	カスタムログを出力します。
Wait	指定時間を待機するか、または指定条件が満たされると、次のコマンドの実行に進みます。
Pause	スクリプトの実行を一時停止
ResetElapsedTime	計時を開始
ElapsedTime	計時を終了
Systime	現在のシステム時刻の取得
SetGlobalVariable	グローバル変数の設定

Print

プロトタイプ：

```
Print(value)
```

説明：

デバッグ情報をコンソールにプリントアウトします（命令名は `print` と書くこともできます）。

> 説明：
> 変数のプリントアウトフォーマットは、本文書でに記載のフォーマットと多少異なることがありますが、同じデータフォーマットを表すものです。本文書に記載のフォーマットを参照して把握し、使用してください。

```
-- 例えば、変数のフォーマットが{pose={x,y,z,rx,ry,rz}}の場合、プリントアウトフォーマットは以下のようなものです。
table:0x123abc{
[pose] => table:0x123abc{
[1] => x
[2] => y
```

```
[3] => z
[4] => rx
[5] => ry
[6] => rz
}
}
```

必須パラメータ：

value：プリント待ちのデータ

例：

```
-- コンソールに文字列Successをプリントアウトします。
Print('Success')
```

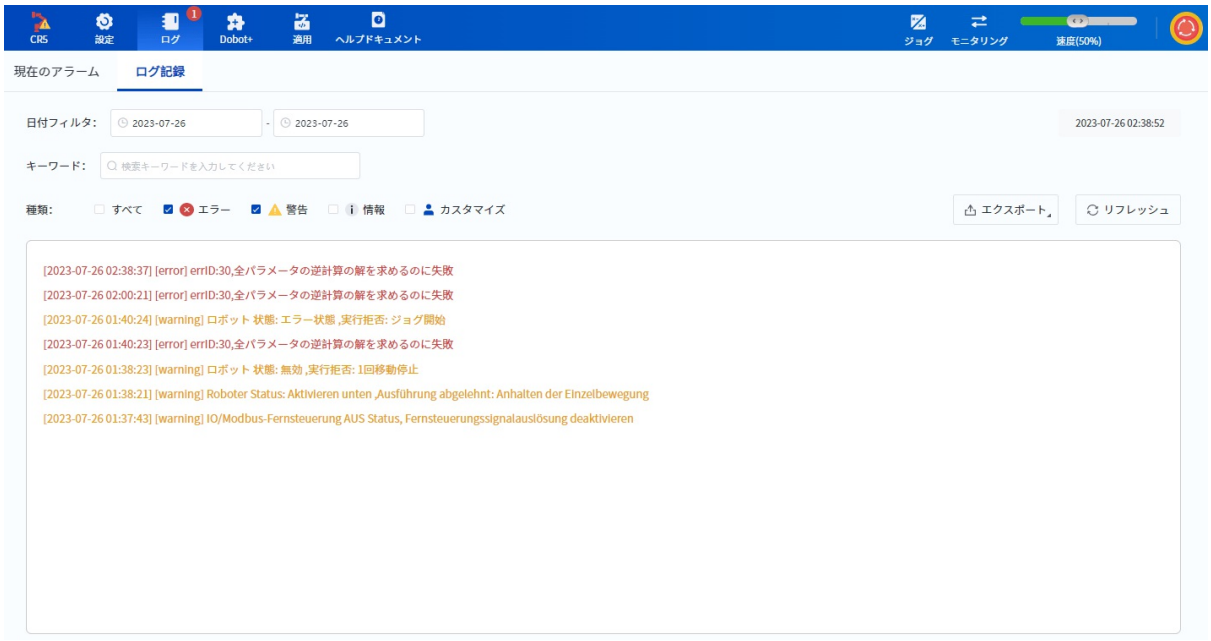
Log

プロトタイプ：

```
Log(value)
```

説明：

カスタムレベルのログ情報を出力します。制御ソフトウェアログページに表示し、エクスポートすることができます。



必須パラメータ：

value：ログ情報

例：

```
-- 内容がtestであるログ情報を出力します。
Log('test')
```

Wait

プロトタイプ：

```
Wait(time_ms)
Wait(check_str)
Wait(check_str, timeout_ms)
```

説明：

ロボットが前の命令を完了すると、指定時間を待機するか、または指定条件が満たされると、次の命令の実行に進みます。待ち時間の最大値は2147483647msであり、設定されたパラメータが最大値を超えると命令が無効になります。

必須パラメータ：

- time_ms：パラメータ値がintegerタイプであるとき、指定待機時間を表し、0以下であるときは待機しないことを表します。単位：ms
- check_str：パラメータ値がstringタイプであるとき、ロジック判断を表し、ロジックがtrueであれは次の命令の実行に進みます。

選択可能なパラメータ：

timeout_ms：タイムアウト時間。ロジック判断がずっとfalseであり、かつ該時間を超えるまで待機すると、システムは次の命令の実行に進み、「false」が返されます。0以下であるときは、すぐにタイムアウトし、待機しないことを表します。該パラメータを設定しないとき、タイムアウトがなく、ロジック判断がtrueになるまでずっと待機します。単位：ms

戻る：

条件が満たされて実行を続行するとき、trueが返されます。条件が満たされず、タイムアウトに起因して実行を続行するときは、falseが返されます。

例：

```
-- 300ms待機します。
Wait(300)
```

```
-- DI1がONであるとき、実行を続行します。
Wait("DI(1) == ON")
```

```
-- DO1がONで、AI(1)が7未満のとき、実行を続行します。
Wait("GetDO(1) == ON and AI(1) < 7")
```

```
-- 1s内のDI1の状態に応じて異なるサービスロジックを実行します。
if (Wait("DI(1) == ON", 1000))
then
    -- DI1状態がONである
else
    -- DI1状態がOFFであり、かつ待機した時間が1sを上回る
end
```

Pause

プロトタイプ：

```
Pause()
```

説明：

スクリプトの実行を一時停止します。実行を続行するには、制御ソフトウェアまたはリモート制御で操作する必要があります。

例：

```
-- ロボットがP1点に移動した後に移動を一時停止し、外部制御によって実行を続行してP2点に移動します。
MovJ(P1)
Pause()
MovJ(P2)
```

ResetElapsedTime

プロトタイプ：

```
ResetElapsedTime()
```

説明：

この命令の前のすべての命令の実行が完了するのを待機してから計時を開始します。ElapsedTime()と合わせて使用する必要があります。実行時間の計算に使用します。

例：

ElapsedTimeの例を参照してください。

ElapsedTime

プロトタイプ：

```
ElapsedTime()
```

説明：

計時が終了し、時間差が返されます。ResetElapsedTime()と合わせて使用する必要があります。

戻る：

計時開始から計時終了までの時間差（ミリ秒）。最大で4294967295ms（約49.7日）まで統計が可能で、それを超えると0から再カウントします。

例：

```
-- ロボットがP1とP2の間を直線移動で10回往復する時間を計算し、コンソールにプリントアウトします。
MovJ(P2)
ResetElapsedTime()
for i=1,10 do
    MovL(P1)
    MovL(P2)
end
print (ElapsedTime())
```

Systime

プロトタイプ：

```
Systime()
```

説明：

現在のシステム時間を取得します。

戻る：

システムの現在時刻のUnixタイムスタンプ、単位はミリ秒に変換されています。すなわち、グリニッジ標準時1970年1月1日0時から現在までのミリ秒数で、通常は時間差を計算するために使用されます。

例：

```
-- システムの現在の時刻を取得します。
local time1 = Systime()
print(time1)  -- > 1686304295963を北京時間に変換すると、2023年6月9日17時51分35秒（963ミリ秒を追加）
となります。
local time2 = Systime()
print(time2)  -- > 1686304421968を北京時間に変換すると、2023年6月9日17時53分41秒（968ミリ秒を追加）
```

```
となります。

-- ロボットがP1に移動するのにかかる時間を計算します。単位はミリ秒です。
local time1 = Systime()
MovL(P1)
local time2 = Systime()
print(time2-time1)
```

SetGlobalVariable

プロトタイプ：

```
SetGlobalVariable(key,val)
```

説明：

グローバル変数を設定します。グローバル変数に値を割り当てる場合は、この関数を使用することをお勧めします。「=」の使用は推奨されません。

必須パラメータ：

- key：設定するグローバル変数の名前。
- val：設定するグローバル変数の値。サポートされるデータ型はbool、table、string、numberを含みます。

例：

```
-- グローバル変数g1の値を10に設定します。
SetGlobalVariable("g1",10)
```

パレット

命令リスト

パレットは、バッチ材料を規則的に配置するための搬送装置であり、自動積み降ろしプロセスでよく使用されます。通常、パレットには多数の溝がアレイ状に配置されており、それぞれの溝に材料を配置できます。パレット命令を使用すると、少数の点を教示するだけで完全なパレット点配列を作成でき、作成したパレット内の特定の点を取得して、ロボットの自動ロードおよびアンロードを迅速に実現できます。

命令	機能
CreateTray	パレットの作成
GetTrayPoint	パレット点を取得する

CreateTray

プロトタイプ：

```
CreateTray(Trayname, {Count}, {P1,P2})  -- 1次元パレット
CreateTray(Trayname, {row,col}, {P1,P2,P3,P4})  -- 2次元パレット
CreateTray(Trayname, {row,col,layer}, {P1,P2,P3,P4,P5,P6,P7,P8})  -- 3次元パレット
```

説明：

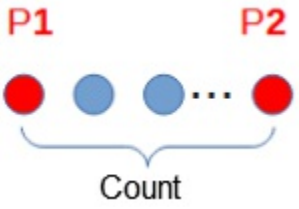
パレットを作成し、1次元、2次元、3次元のパレットを作成することができます。パレットは最大20個まで作成できますが、同名のパレットを作成すると元のパレットが上書きされ、パレット数は増加しません。

必須パラメータ：

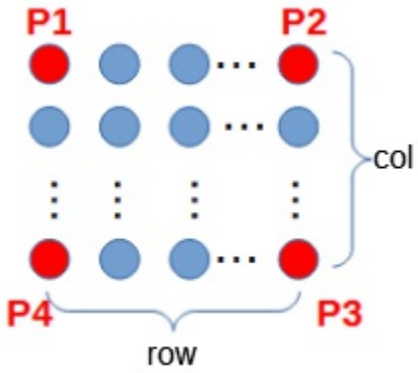
- Trayname：パレット名。純粋な数字やスペースを使用できない最大32バイトの文字列。

最後の2つのパラメータはtable変数です。table内の値の数は作成されるパレットの次元によって異なります。以下にそれぞれ紹介します。

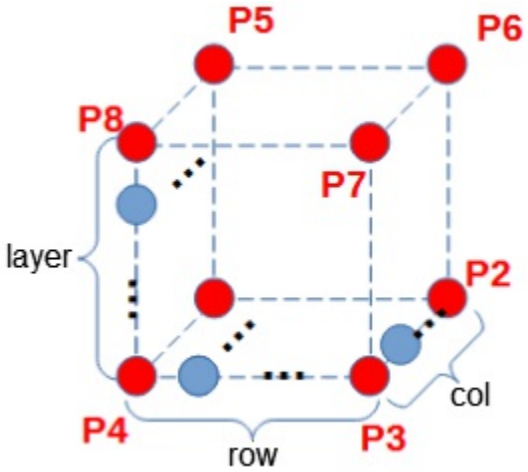
- 1次元パレットの作成：1次元パレットは1直線に沿って等間隔に配置された点のセットです。
    - {Count}：Countはポイントの数を表し、値の範囲は[2,50]で、整数以外の数値を入力すると、自動的に切り捨てられます。
    - {P1、P2}：P1とP2はそれぞれ1ビットパレットの2つのエンドポイントであり、教示点と姿勢変数をサポートします。



- 2次元パレットの作成：2次元パレットは平面上に配列された点のセットです。
    - {row,col}：rowは行方向（P1～P2方向）のポイント数、colは列方向（P1～P4方向）のポイント数を表し、値の範囲は1次元パレットのCountと同じです。
    - {P1、P2、P3、P4}：P1、P2、P3、P4はそれぞれ2次元パレットの4つの頂点であり、教示点と姿勢変数をサポートします。



- 3次元パレットの作成：3次元パレットは、空間内に立体的に分散された点の集合であり、複数の2次元パレットが垂直に配置されたものとみなすことができます。
    - {row,col,layer}：rowは行方向（P1～P2方向）のポイント数、colは列方向（P2～P4方向）のポイント数、layerはレイヤ数（P1～P5方向）を表します。
    - {P1、P2、P3、P4、P5、P6、P7、P8}：P1～P8は3次元パレットの8つの頂点であり、教示点と姿勢変数をサポートします。



⚠注意：
先端ツールを使用する場合は、点の教示時に必ず先端ツールに対応した工具座標系を選択してください。

例：

```
-- t1という名前の5点の1次元パレットを作成します。
CreateTray("t1", {5}, {P1,P2})
-- t2という名前の4x5の2Dパレットを作成します。
CreateTray("t2", {4,5}, {P1,P2,P3,P4})
-- t3という名前の4x5x6の3次元パレットを作成します。
CreateTray("t2", {4,5,6}, {P1,P2,P3,P4,P5,P6,P7,P8})
```

GetTrayPoint

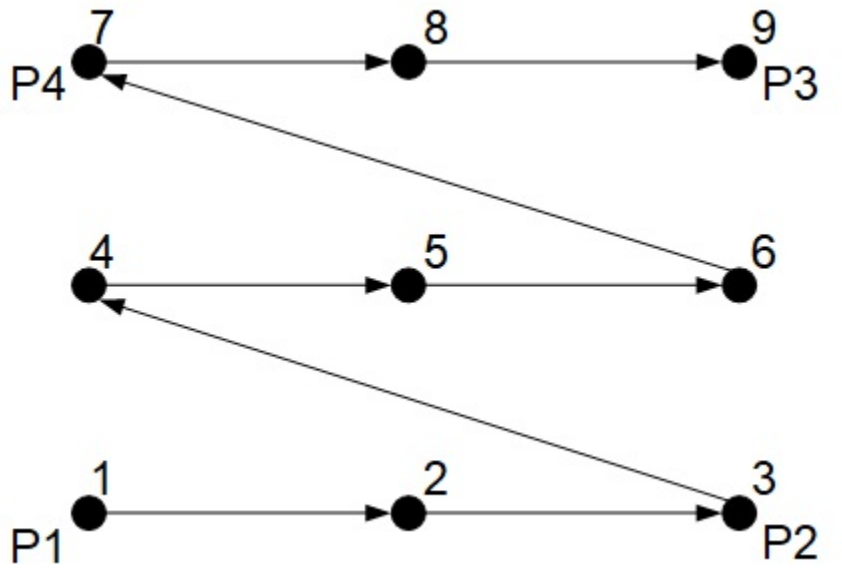
プロトタイプ：

```
GetTrayPoint(Trayname, index)
```

説明：

指定パレットの指定された番号の点位置を取得します。点の番号はパレットの作成時に渡された点の順番に関わります。

- 1次元パレット：P1点の番号は1、P2点の番号は点の数と同じです。以降も同様です。
- 2次元パレット：次の図は、3x3パレットを例として、教示点と点の番号の関係を示しています。



- 3次元パレット：2次元パレットを参照すると、2番目の層の最初の点の番号は、1番目の層の最後の点の番号に1を加えたものになります。以降も同様です。

必須パラメータ：

- Trayname：最大32バイトの文字列を含む作成されたパレット名。
- Index：取得する点位置の番号。

戻る：

番号に対応する点位置の座標。パレット作成時に使用した点が教示点の場合、返される点の形式も教示点となります。パレットの作成時に使用された点が姿勢変数の場合、返される点の形式も姿勢変数です。

例：

```
-- t1という名前のパレットの番号が3のて点位置を取得します。
GetTrayPoint("t1",3)
```

付属書E Pythonプログラミング関数の説明

基本的な構文

> 説明：
> このマニュアルの読者は、Pythonプログラミングの基本的な知識をすでにある程度知っている必要があります。このマニュアルには、クイックリファレンスとしていくつかの基本的なPython構文のみがリストされています。

Python言語では大文字と小文字が区別され、プログラム内のすべての識別子 **(**変数名、関数名など**)** の大文字と小文字は、このマニュアルで提供されている例と一致している必要があります。

変数

変数は、値を格納したり、パラメータとして渡したり、結果として返したりするために使用します。Pythonの変数は宣言する必要がなく、値が設定された後に使用できます。

変数には等号を介して値が設定され、二重等号は左側と右側の式が等しいかどうかを判断するために使用されます。

Python独自のスコープルールに加えて、DobotStudio Proのモニター > グローバル変数ページでコントローラーレベルのグローバル変数を設定することができます。この変数は、同じコントローラー内の異なるプロジェクトで直接呼び出すことができます。

次の例は、コントローラーグローバル変数のスコープとグローバル永続化機能を示しています。

```
'''
プロジェクト1のmain.pyファイル
2つのグローバル変数g1とg2がグローバル変数ページに追加されたとします
g1はグローバル保持されず、値は10です
g2はグローバル保持され、値は20です
```

```
'''

a = 1
print(a)        --> 1

print(g1)           --> 10
print(g2)           --> 20
g1=11  # グローバル保持ではないグローバル変数に値を設定します
g2=22  # グローバル保持のグローバル変数に値を設定します
print(g1)           --> 11
print(g2)           --> 22

'''
プロジェクト2のmain.pyファイル
プロジェクト2はプロジェクト1の後に実行されます
'''

print(a)        # 変数が存在しません
print(b)        # 変数が存在しません
print(g1)       # 10（グローバル保持ではない変数は、プロジェクト1が変更する前の値に戻ります）
print(g2)       # 22（グローバル保持の変数は、プロジェクト1が変更した後の値になります）
```

Pythonはさまざまなデータタイプをサポートしており、DobotはAPIを設計するときに通常、数字（Number）、ブール値（bool）、文字列（String）、リスト（List）、辞書（Dictionary）を使用します。

数字

Pythonの数字は、int（長整数）、float（倍精度浮動小数点数）、bool（ブール値）、およびcomplex（複素数）をサポートします。

```
var1 = 1
var2 = 10
```

ブール値

ブールタイプには、TrueとFalseの2つのオプションのみがあります。Pythonでは、Trueは数字1と同じ、Falseは数字0と同じであり、数値演算に使用できます。さらに、ゼロ以外の数字、ヌルでない文字列、リスト、辞書、その他のデータタイプはすべてTrueとみなされ、0、ヌルの文字列、ヌルのリスト、ヌルの辞書などのみがFalseとみなされます。

```
print(2 < 3)   # True
print(2 == 3)  # False

a = True
b = False
print(a and b)  # False
print(a or b)   # True
print(not a)    # False
```

文字列

Pythonの文字列は一重引用符または二重引用符で囲まれ、プラス記号で連結できます。

```
str1 = 'Dobot'
str2 = " Robot"
print(str1 + str2)  # Dobot Robot
```

文字列には添字を使用してインデックスを付けることができ、インデックス値は0から始まり、最後は-1から始まります。文字列をインターセプトするための構文形式は次のとおりです。`変数[先頭の添え字:末尾の添え字]`。文字列は変更できません。新しい文字列を変数に割り当てることはできますが、インデックスを作成して文字列内の文字を変更することはできません。

```
str1 = 'Dobot'
print(str1[0])    # D
print(str1[-1])   # t
print(str1[0:2])  # Dob
```

リスト

リストは、角括弧の間に書かれたカンマ区切りの要素のリストです。リストは文字列と同様に、プラス記号で接続し、添え字でインデックスを付けることができます。文字列とは異なり、リストは変更することができ、インデックスによってリストの要素を個別に変更できます。

```
list1 = [1, 2, 3]
list2 = ["a", "b", "c"]
print(list1 + list2)     # [1, 2, 3, "a", "b", "c"]
print(list1[0])          # 1
list1[0] = "a"
print(list1)             # ["a", 2, 3]
del list[2]              # リストから3番目の要素を削除します
```

辞書

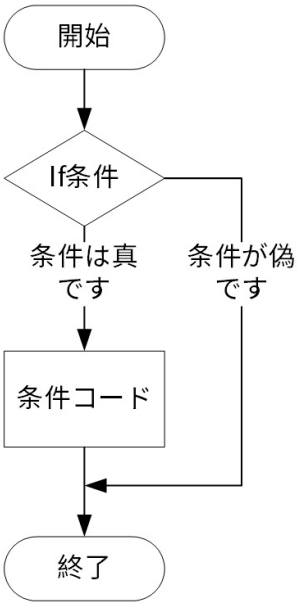
辞書は、中括弧で囲まれ、カンマで区切られた「キー(key) : 値(value)」の集合です。辞書はキーによってインデックスが付けられるため、キーは一意である必要があります。辞書は、キーを使用して追加、削除、変更、検索できます。

```
dict1 = {"user" : 1, "tool" : 0, "a" : 20, "v" : 50, "cp" : 100}
print(dict1["user"])      # 1
dict1["tool"] = 1         # 辞書内のtoolの値を1に変更します
dict1["r"] = 5            # キーが辞書に存在しない場合は、新しいキーと値のペアが辞書に追加されます。
del dict1["r"]            # 指定されたキーと値のペアを削除します
```

プロセス制御

if条件判定命令

If命令は、条件判定の結果に基づいて異なるコードブロックを実行できます。



Pythonのifステートメントの標準形式は次のとおりです。elifとelseは必要ありません。

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

- 「condition_1」がTrueの場合、「statement_block_1」ステートメントブロックが実行されます
- 「condition_1」がFalseの場合、「condition_2」が判定されます
- 「condition_2」がTrueの場合、「statement_block_2」ステートメントブロックが実行されます
- 「condition_2」がFalseの場合、「statement_block_3」ステートメントブロックが実行されます

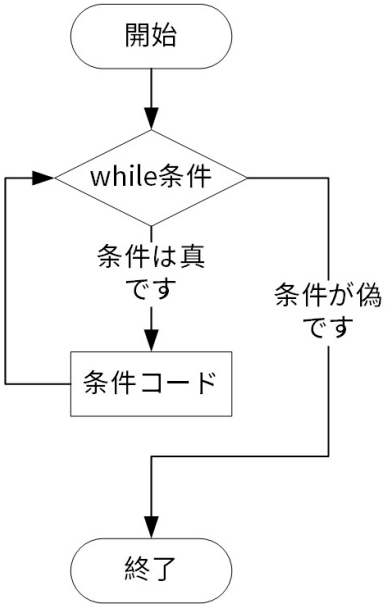
ifステートメントは入れ子にすることができます。典型的な例を次に示します。

```
a = 100;
b = 200;
# 条件を確認する
if a == 100:
    # if条件がTrueの場合、以下のif条件判定が実行されます
    if b == 200:
        # if条件がTrueの場合、このステートメントブロックが実行されます
```

```
        print("aの値は次のとおりです：", a )  # aの値は次のとおりです：100
        print("bの値は次のとおりです：", b )  # bの値は次のとおりです：200
else:
    # 最初のif条件がFalseの場合、次のステートメントブロックが実行されます
    print("aは100に等しくありません")
```

whileループ命令

while命令は、条件に基づいてコードブロックをループできます。



Pythonのwhileステートメントの標準形式は次のとおりです。

```
while condition:
    statement_block
```

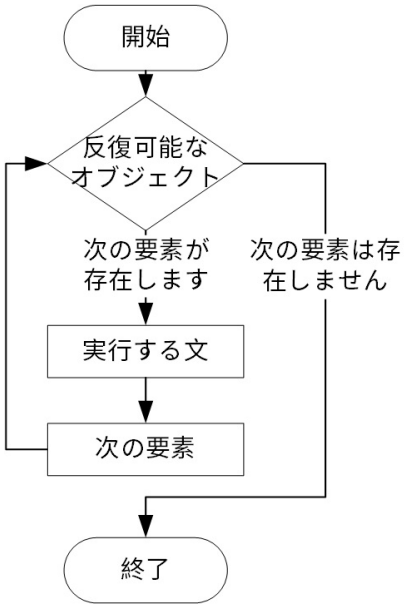
- 「condition」がTrueの場合、「statement_block」ステートメントブロックは実行され、その後「condition」が再判定されます
- 「condition」がFalseの場合、「statement_block」ステートメントブロックはスキップされ、後続のステートメントが直接実行されます

例：

```
a=10
while a < 20
    print("aの値は次のとおりです：", a)  # 10回実行すると出力値は10～19になります
    a = a+1
```

forループ命令

for命令は、リストや文字列などの反復可能なオブジェクトを走査し、走査回数に応じてコードブロックをループできます。



Pythonのforステートメントの標準形式は次のとおりです。

```
for variable in sequence:
    statement_block
```

sequenceは反復可能なオブジェクトであり、variableは反復可能なオブジェクトを走査するために使用される変数です。

1. まず「sequence」の最初の要素は「variable」に代入され、次に「statement_block」ステートメントブロックが1回実行されます。
2. 「sequence」内に別の要素がある場合、その要素は「variable」に代入され、「statement_block」ステートメントブロックが再度実行されます。
3. 「sequence」内のすべての要素が走査されたまで、上記の手順を繰り返します。

例：

```
joint = [j1,j2,j3,j4,j5,j6]
for angles in joint:
    print(angles)   # 6回実行してj1～j6の値を出力します
```

全般的な説明

移動方法

ロボットアームがサポートする移動方法は次のいくつかに分けられます。

関節移動

ロボットアームは現在の各関節角度と目標点の各関節角度の差に基づいて各関節の移動を計画し、すべての関節が同時に移動を完了します。関節移動はTCP（Tool Center Point）の移動の軌跡を制約せず、通常この軌跡は直線ではありません。

関節移動は特異位置の制限を受けません（特異位置についてはロボットアームの対応するハードウェアマニュアルを参照）。そのため、移動の軌跡に要求がない場合や目標位置が特異位置近くにある場合は、関節移動を使用することをお勧めします。

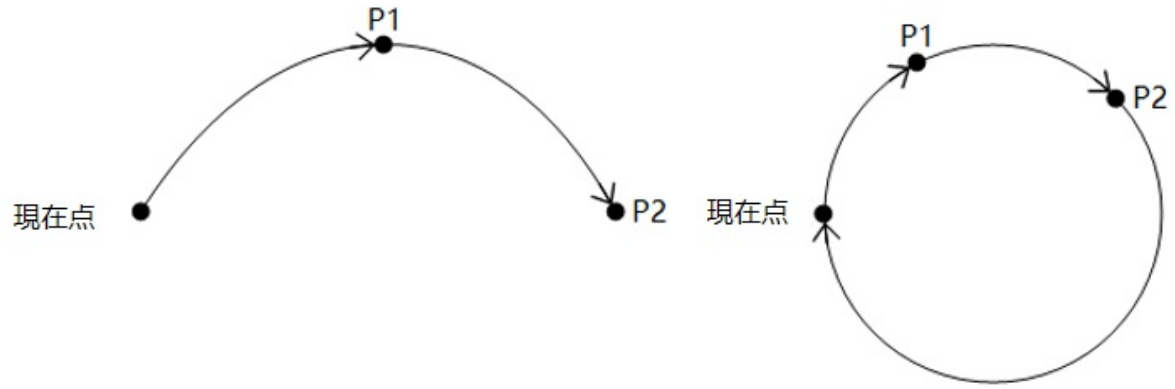
直線移動

ロボットアームは現在のポーズと目標点のポーズに基づいて移動の軌跡を計画し、TCPの移動の軌跡が直線になり、先端の姿勢が移動の過程で一定の速度で変化します。

移動の軌跡が特異位置を通る場合、直線移動の命令をロボットアームに送信するとエラーが発生する可能性があります。その場合は位置を再計画するか、特異位置近くで関節移動を使用することをお勧めします。

弧線移動

ロボットアームは現在の位置、P1、P2の3つの非共線点を使用して円弧または完全な円を確定します。移動中のロボットアームの先端の姿勢は、現在の点とP2点の姿勢から補間計算され、P1点の姿勢は計算に参加しません（つまり、ロボットアームがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。



移動の軌跡が特異位置を通る場合、弧線移動の命令をロボットアームに送信するとエラーが発生する可能性があります。その場合は位置を再計画するか、特異位置近くで関節移動を使用することをお勧めします。

位置パラメータ

特別な説明がない場合、本ハンドブック中のすべての位置パラメータは3種類の表現方法をサポートします。

- 関節変数：各ロボットアームの各関節の角度(j1-j6)を使用して目標位置を表示します。

  ジョイント変数が直線または弧線移動パラメータとして使用される場合、システムは順運動学の数値演算によりこれをポーズ変数に変換しますが、アルゴリズムはロボットアームが目標点に到達したときの関節角度が設定値と一致することを保証します。

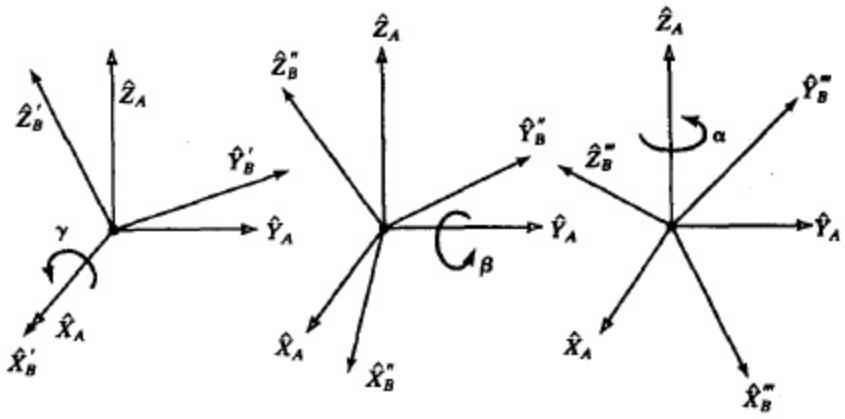
```
{"joint" : [j1, j2, j3, j4, j5, j6]}
```

- ポーズ変数：デカルト座標（x，y，z）を使用して、目標位置のユーザー座標系における空間的位置を表示します。オイラー角（rx，ry，rz）を使用してTCP（Tool Center Point）がこの点に到達したときの工具座標系のユーザー座標系に対する回転角度を表示します。

  ポーズ変数が関節移動の位置パラメータとして使用される場合、システムは逆運動学の数値演算によりこれを関節変数に逆変換します（ロボットアームの現在の関節角度に最も近い解を採用）。

```
{"pose" : [x, y, z, rx, ry, rz]}
```

越疆のロボットアームのオイラー角を計算する時の回転順序はX->Y->Zです。各軸は、下図に示されているように、固定軸（ユーザー座標系）を中心に旋回します（rx=γ，ry=β，rz=α）。



回転順序が確定すると、回転行列（cαはcosα、sαはsinαの略語である。以下同様）を

$$
\begin{aligned}
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)R_Y(\beta)R_X(\gamma) \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
\end{aligned}
$$

方程式に導出します

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

この方程式により、ロボットアーム先端の姿勢を計算します。

- 教示点：制御ソフトウェアを使用して、教示で取得した位置は以下の形式の定数として保存されます。

```
'''
name：教示点の名称。
joint：教示点の関節座標。
tool：教示時に使用する工具座標系のインデックス。
user：教示時に使用するユーザー座標系のインデックス。
pose：教示点のポーズ変数値。
'''
{
    "name" : "name",
    "tool" : index,
    "user" : index,
    "joint" : [j1, j2, j3, j4, j5, j6],
    "pose" : [x, y, z, rx, ry, rz]
}
```

座標系パラメータ

移動命令のオプションパラメータ中のuserとtoolは、目標点のユーザーと工具座標系を指定するために使用され、現在はインデックス番号による指定のみをサポートしています。対応する座標系は先に制御ソフトウェアに追加する必要があります。

システムの位置座標系の選択の優先順位は以下のとおりです。

1. 移動命令のオプションパラメータで座標系を指定した場合は、指定された座標系を使用します。位置パラメータが教示点である場合、教示点のポーズ座標を指定座標系の値に換算し、使用することができます。
2. オプションパラメータで座標系が指定されていない場合は、位置が教示点であると、教示点自体の座標系インデックスを使用します。
3. オプションパラメータで座標系が指定されていない場合は、位置が関節変数またはポーズ変数であると、移動パラメータ中で設定したグローバル座標系を使用します（詳細はUserとTool命令を参照。命令を呼び出して設定しない場合のデフォルト座標系は0）

説明：
- スクリプトの実行を開始すると、デフォルトのグローバル座標系はすべて0にリセットされ、スクリプトの実行前にジョグパネルで設定した値とは無関係になります。
- 関節移動命令（MovJ/MovJIO/RelMovJTool/RelMovJUser）を呼び出す際、位置が関節変数である場合、座標系パラメータは無効です。

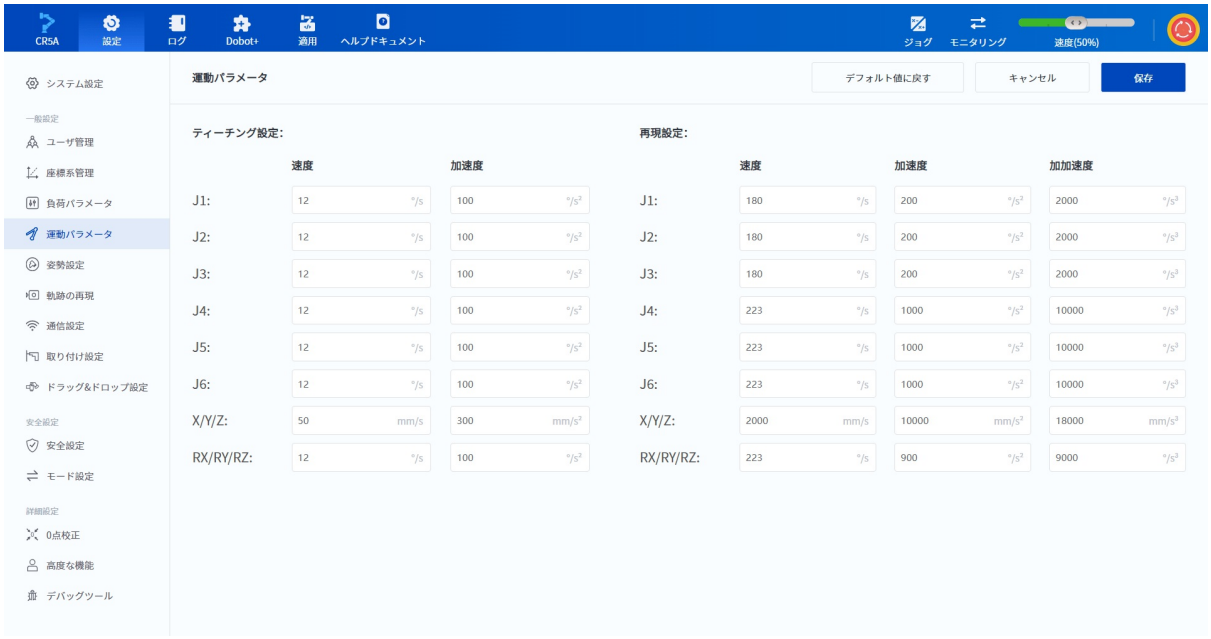
速度パラメータ

相対速度

オプションパラメータ中のaおよびvは、ロボットアームがこの移動命令を実行する場合の加速度と速度の比率を指定するためのものです。

```
ロボットアームの実際の移動の速度 = 最大速度 x グローバル速度 x 命令の速度
ロボットアームの実際の移動の加速度 = 最大加速度 x 命令の速度
```

そのうち、最大速度/加速度は再現設定によって制御され、制御ソフトウェアの移動パラメータページで確認および変更することができます。



グローバル速度は、制御ソフトウェア（上図右上隅）またはSpeedFactor命令で設定できます。

指令の速度は、移動命令のオプションパラメータで指定された比率で、移動加速度/速度比率がオプションパラメータで指定されていない場合、デフォルトでは移動パラメータで設定された値が使用されます（詳細はVelJ、AccJ、VelL、AccL命令を参照、命令を呼び出して設定しない場合のデフォルト値はすべて100）。

例：

```
AccJ(50) # 関節移動のデフォルト加速度を50%に設定
VelJ(60) # 関節移動のデフォルト速度を60%に設定
AccL(70) # 直線移動のデフォルト加速度を70%に設定
VelL(80) # 直線移動のデフォルト速度を80%に設定

# グローバル速度は20%です。

MovJ(P1) # 加速度（最大関節加速度 x 50%）および速度（最大関節速度 x 20% x 60%）でP1に関節移動します
MovJ(P2,{"a":30, "v":80}) # 加速度（最大関節加速度 x 30%）と速度（最大関節速度 x 20% x 80%）でP1に関節移動します

MovL(P1) # 加速度（最大デカルト加速度 x 70%）と速度（最大デカルト速度 x 20% x 80%）でP1に直線移動します
MovL(P1,{"a":40, "v":90}) # 加速度（最大デカルト加速度 x 40%）と速度（最大デカルト速度 x 20% x 90%）でP1に直線移動します
```

絶対速度

直線および円弧移動命令のオプションパラメータ中のspeedは、ロボットアームがこの移動命令を実行する場合の絶対速度を指定するためのものです。

絶対速度は、グローバル速度から影響を受けないが、再現設定の最大速度の制限を受けます（ロボットアームが低減モードに入っている場合、低減後の最大速度の制限を受けます）。すなわち、speedパラメータに設定した標的速度が再現設定の最大速度よりも大きい場合、最大速度に準じます。

例：

```
MovL(P1,{"speed":1000})  # 絶対速度1000でP1に直線移動
```

MovLはspeedを1000に設定し、再現設定の最大速度2000を下回るため、ロボットアームは目標速度1000mm/sで移動します。この目標速度は現時点のグローバル速度とは無関係です。ただし、ロボットアームが縮減モード（縮減率を10%で仮定）である場合、最大速度は200に変わり、1000より小さいので、この時のロボットアームは200m/sを目標速度として移動します。

speedパラメータとvパラメータは排他的で、両方存在する場合はspeedが優先されます。

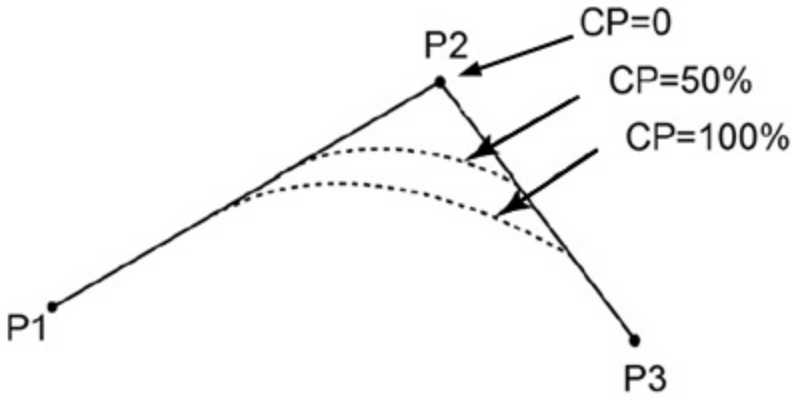
連続経路制御パラメータ

ロボットアームが複数の点を通過して連続移動する場合、連続経路制御方式で中間点を通過し、ロボットアームを強く回転することを回避することができます。ユーザーが指定する複数の経路点が、異なる工具座標系を基にしている場合、連続経路制御を行うことはできません。

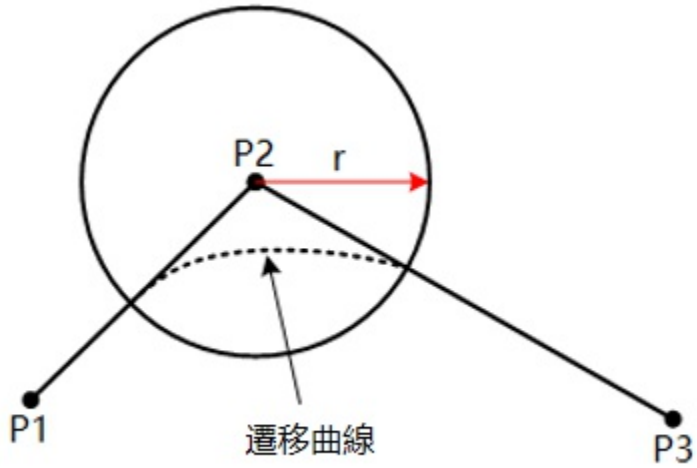
オプションパラメータのcpまたはrは、現在の移動命令から次の移動命令までの連続経路制御比率（cp）または連続経路制御半径（r）を指定するためのもので、両者は相互に排他的です。同時に存在する場合にはrを基準とします。

> 説明：
> 関節移動関連の命令は連続経路制御半径（r）の設定をサポートしていません。各命令のオプションパラメータをご参照ください。

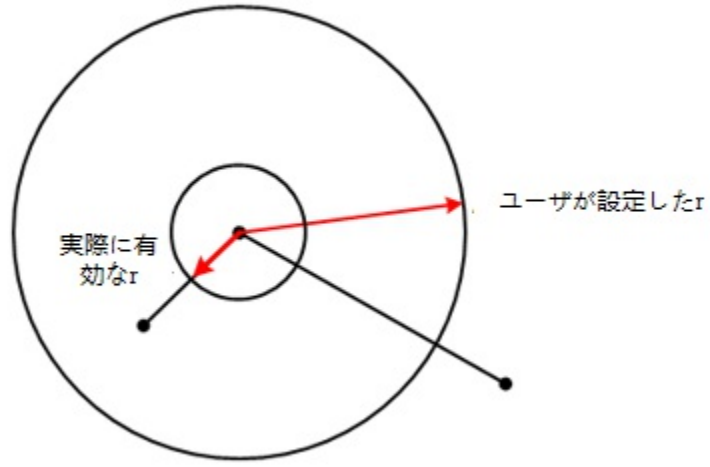
連続経路制御比率を設定する場合は、移動曲線の円弧が自動的に計算されます。下図のとおり、CP値が大きいほど曲線がスムーズになります。CP移動曲線は移動速度/加速度の影響を受けます。たとえ位置およびCP値が同じであっても、移動速度/加速度が異なる場合の移動曲線の円弧は異なります。



連続経路制御の半径を設定する場合は、移動点を円の中心として、指定半径を基にして移動曲線が計算されます。R移動曲線は移動速度/加速度の影響を受けず、位置と移動半径のみで決定されます。



ユーザーが設定する移動半径が大きすぎる場合は（開始点/終了点と移動点との間の距離を超過）、開始点/終了点と移動点との間の比較的短い距離の半分を移動半径を使用して、移動曲線が計算されます。



オプションパラメータで連続経路制御比率や半径が指定されていない場合、デフォルトでは移動パラメータで設定された連続経路制御比率が使用されます（詳細はCP命令を参照、命令を呼び出して設定しない場合のデフォルト値は0）。

> 説明：
> 連続経路制御により、ロボットアームが中間点を通過せずに移動することができます。したがって、連続経路制御が設定されている場合、2つの移動命令の間のIO信号出力または機能設定（例えば、安全スキンのオン/オフ）命令は移動中に実行されます。
> ロボットアームが正確に中間点に到達したときに命令を実行したい場合は、前の命令の連続経路制御パラメータを0に設定してください。

cpとrの実装方法が異なるため、連続経路制御が必要な2つの移動命令の間には、移動に影響を与えない他の命令(条件判定、IO、機能設定など)が挿入される場合、cpとrの処理方法も違います。

- cpモードを使用して移行する場合、命令の処理時間が長すぎる場合 (Wait命令など) を除き、ほとんどの命令は移行に影響を与えません。

以下の例では、2つの移動命令の間にifステートメントを挿入しても、cpモードの連続経路制御には影響しません。

```
# 正常に移行できます。ロボットはP1点に到達する直前にDI1を判定し、ONの場合は連続経路制御率50%で次の移動命令に移行します。
MovL(P1,{"cp":50})
if DI(1)==ON:
    MovL(P2)
```

- rモードを使用して移行する場合、次のホワイトリストにある命令のみが連続経路制御に影響しません。他の命令を挿入すると、rモードでの連続経路制御が無効になります。

```
RelPointTool, RelPointUser, DOGroup, DO, AO, ToolDO,
SetUser, SetTool, User, Tool, CP, AccJ, AccL, VelL, VelJ,
SetPayload, SetCollisionLevel, SetBackDistance, EnableSafeSkin, SetSafeSkin
```

以下の例では、2つの移動命令の間にifステートメントを挿入すると、rモードでの連続経路制御の設定が無効になります。

```
# 連続経路制御パラメータは無効であり、ロボットは点P1に到達した後にDI1を判定します。
MovL(P1,{"r":5})
if DI(1)==ON:
    MovL(P2)
```

IO信号の表現方法

システム内部では、ONとOFFの変数がデジタルIOの信号有無を表すために事前定義されています。

- ON　=　1は、信号があることを意味します
- OFF　=　0は、信号がないことを意味します

パラメータのON|OFFは、パラメータの値がONまたはOFFであることを意味します。ユーザーは1または0を入力として使用することもできます。

移動命令

命令リスト

移動命令は、ロボットアームを制御して移動させるためのものです。使用する前に全般的な説明を読んでください。

命令	機能
MovJ	関節移動
MovL	直線移動
Arc	円弧補間移動
Circle	円補間移動
MovJIO	関節的に移動し、DOを出力する
MovLIO	直線移動と出力DO
StartPath	記録を再現する移動軌跡
GetPathStartPose	軌跡の開始点を取得
PositiveKin	関節角度から姿勢を導出
InverseKin	姿勢から関節角度を導出

MovJ

プロトタイプ：

```
MovJ(point, {"user":1, "tool":0, "a":20, "v":50, "cp":100})
```

説明：

現在の位置から関節移動方式で目標点まで移動します。

必須パラメータ：

point：目標点。

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]

- cp：連続経路制御の比率。値の範囲：[0,100]

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定でP1まで関節的に移動します。
MovJ(P1)
```

```
# ロボットアームはデフォルト設定で指定された関節角度に関節的に移動します。
MovJ({"joint":[0,0,90,0,90,0]})
```

```
# ロボットアームはユーザー座標系1とツール座標系1に対応する指定された姿勢に関節的に移動します。移動加速度と速度は両方とも50%、連続経路制御の比率は50%です。
MovJ({"pose":[300,200,300,180,0,0]},{"user":1, "tool":1, "a":50, "v":50, "cp":50})
```

```
# 最初に点位置を定義してから移動命令で呼び出します。動作効果は前の命令と同じです。
customPoint={"pose":[300,200,300,180,0,0]}
MovJ(customPoint,{"user":1, "tool":1, "a":50, "v":50, "cp":50})
```

MovL

プロトタイプ：

```
MovL(point, {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100, "r":5})
```

説明：

現在の位置から直線移動方式で目標点まで移動します。

必須パラメータ：

point：目標点。

選択可能なパラメータ：

- user：目標点のユーザー座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- tool：目標点のツール座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この指令を実行する際のロボットアームの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]

- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定でP1まで直線的に移動します。
MovL(P1)
```

```
# ロボットアームは絶対速度500m/sでP1まで直線的に移動します。
MovL(P1,{"speed":500})
```

```
# ロボットアームは指定された関節角度までデフォルト設定で直線的に移動します。
MovL({"joint":[0,0,90,0,90,0]})
```

```
# ロボットアームはユーザー座標系1とツール座標系1に対応する指定された位置に直線的に移動します。移動加速度と速度は両方とも50%、連続経路制御半径は5mmです。
MovL({"pose":[300,200,300,180,0,0]},{"user":1, "tool":1, "a":50, "v":50, "r":5})
```

```
# 最初に点位置を定義してから移動命令で呼び出します。動作効果は前の命令と同じです。
customPoint={"pose":[300,200,300,180,0,0]}
MovL(customPoint,{"user":1, "tool":1, "a":50, "v":50, "r":5})
```

```
# Speedとvが同時に含まれると、speedが有効になり、コントローラは警告ログを出力します。
# cpとrが同時に含まれると、rが有効になり、コントローラは警告ログを出力します。
MovL(P1,{"v":50,"speed":500,"cp":60,"r":5}) # 実行時に選択可能なパラメータはspeedとrのみ有効です
```

Arc

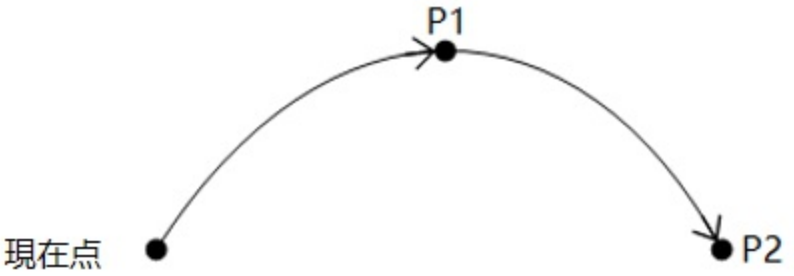
プロトタイプ：

```
Arc(P1, P2, {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100, "r":5})
```

説明：

現在の位置から円弧補間方式で目標点まで移動します。

現在の位置P1、P2、P3の3点により1つの円弧を確定します。これにより、現在の位置はP1およびP2で決まる直線上にあってはなりません。



移動中のロボットアームの先端の姿勢は、現在の点とP2点の姿勢から補間計算され、P1点の姿勢は計算に参加しません（つまり、ロボットアームがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

必須パラメータ：

- P1：円弧の中間点。
- P2：目標点。

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この指令を実行する際のロボットアームの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはP1に移動し、その後デフォルト設定でP2からP3までの円弧に沿って移動します。
MovJ(P1)
Arc(P2,P3)
```

```
# ロボットアームがP1に移動し、点[300,200,300,180,0,0]を経由した円弧に沿ってP3に移動します。ユーザー座標系とツール座標系は両方とも1、移動加速度と速度は両方とも50%です。
MovJ(P1)
Arc({pose:[300,200,300,180,0,0]},P3,{"user":1, "tool":1, "a":50, "v":50})
```

Circle

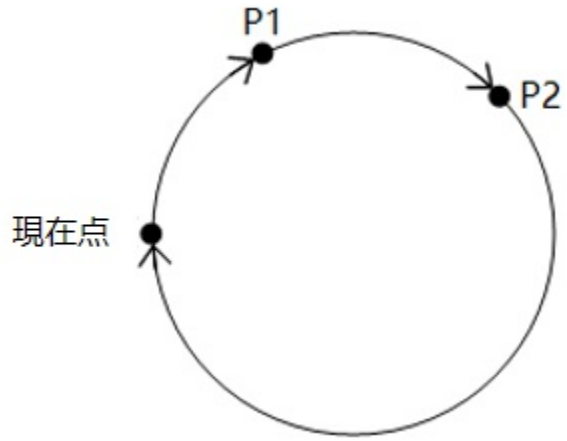
プロトタイプ：

```
Circle(P1, P2, Count, {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100, "r":5})
```

説明：

現在位置で円形補完移動を行い、指定の回数を移動したら現在位置に戻ります。

現在の位置、P1、P2の3点により1つの円を確定します。そのため、現在の位置はP1とP2で決まる直線上にあってはなりません。さらに、3点で決まる円は、ロボットアームの移動範囲を超えてはなりません。



移動中のロボットアームの先端の姿勢は、現在の点とP2点の姿勢から補間計算され、P1点の姿勢は計算に参加しません（つまり、ロボットアームがP1点に到達するときの姿勢は示教姿勢と異なる可能性があります）。

必須パラメータ：

- P1：円の位置1。
- P2：円の位置2。
- Count：完全円形移動を行う回数ををを表します。値範囲は1～999とします。

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この指令を実行する際のロボットアームの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはP1まで移動し、さらにP1、P2、P3で決まる円に沿って1周移動します。
MovJ(P1)
Circle(P2,P3,1)
```

```
# ロボットアームはP1に移動し、その後、P1、点[300,200,300,180,0,0]、P3によって決定された円形に沿って10回り移動します。ユーザー座標系とツール座標系は両方とも1、移動加速度と速度は両方とも50%です。
MovJ(P1)
Circle({pose:[300,200,300,180,0,0]},P3,10,{"user":1, "tool":1, "a":50, "v":50})
```

MovJIO

プロトタイプ：

```
MovJIO(P,[[Mode,Distance,Index,Status],[Mode,Distance,Index,Status]...], {"user":1, "tool":0, "a":20, "v":50, "cp":100})
```

説明：

現在の位置から、関節移動方式により目標点まで移動します。移動する場合にデジタル出力のポート状態を並行して設定します。

必須パラメータ：

- P：目標点。
- 並行デジタル出力パラメータ：ロボットアームが指定距離または百分率まで移動する場合に、指定のDOを起動することを設定します。複数グループを設定することができます。各グループは以下のパラメータを含みます。
  - Mode：トリガモード。0は百分率の起動を示し、1は距離の起動を示します。システムは各関節角を1つの角度ベクトルを合成し、終点と起点の角度差を移動の総距離として計算します。
  - Distance：百分率/角度を指定します。角度計算は合成角ベクトルを使用しているため、百分率モードを使用することをお勧めします。その方の効果が直感的に理解しやすいです。
    - Distanceが正の数である場合、開始点からの百分率/角度を表示します。
    - Distanceが負の数である場合、目標点からの百分率/角度を表示します。
    - Modeが0である場合、Distanceは合計角度との百分率を表示します。値の範囲：(0,100]
    - Modeが1である場合、Distanceは角度の値を表示します。単位：°
  - Index：DO端子の番号。
  - Status：設定するDO状態で、0とOFFは信号なし、1とONは信号ありを示します。

選択可能なパラメータ：

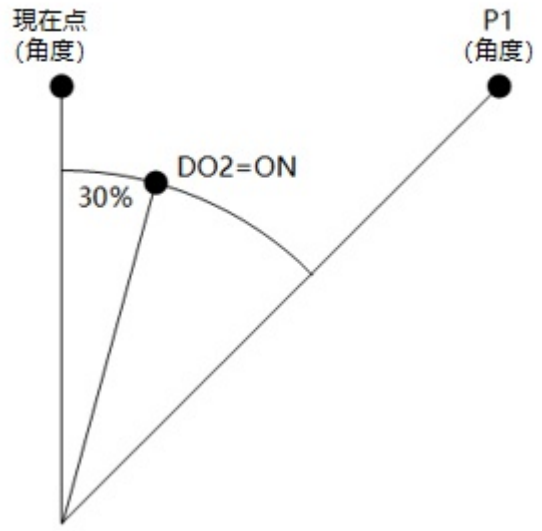
- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]

連続経路制御はロボットアームの移動軌跡を変更し、DO出力のタイミングに影響を及ぼしますので、慎重に使用してください。

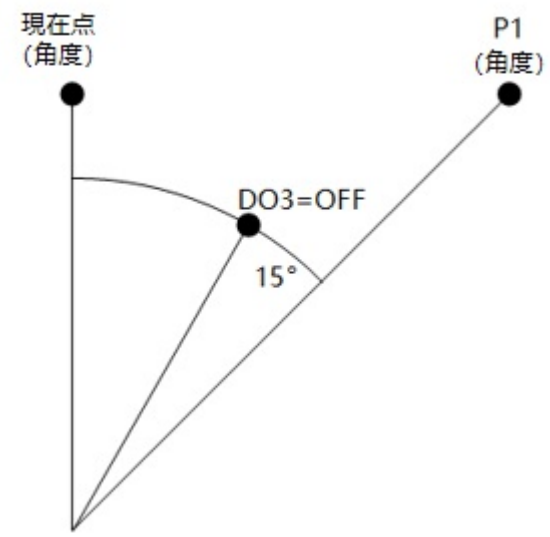
詳細は一般説明をご参照ください。

例：

```
# ロボットアームは、デフォルト設定ではP1に向けて関節的に移動します。起点から30%の位置に移動した場合、DO2を開として設定します。
MovJIO(P1, [[0, 30, 2, 1]])
```




```
# ロボットアームはデフォルト設定でP1に向けて関節的に移動し、終点から15°の位置に移動したときに、DO3をオフに設定します。
MovJIO(P1,[[1, -15, 3, 0]])
```



MovLIO

プロトタイプ：

```
MovLIO(P,[[Mode,Distance,Index,Status],[Mode,Distance,Index,Status]...],{"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100, "r":5})
```

説明：

現在の位置から直線移動方式で目標点まで移動し、移動する場合にはデジタル出力ポート状態を並行して設定します。

必須パラメータ：

- P：目標点。
- 並行デジタル出力パラメータ：ロボットアームが指定距離または百分率まで移動する場合に、指定のDOを起動することを設定します。複数グループを設定することができます。各グループは以下のパラメータを含みます。
  - Mode：トリガモード。0は百分率の起動を示し、1は距離の起動を示します。
  - Distance：百分率/距離を指定します。
    - Distanceが正の数である場合、開始点からの百分率/距離を表示します。
    - Distanceが負の数である場合、目標点からの百分率/距離を表示します。
    - Modeが0である場合、Distanceは合計距離との百分率を表示します。値の範囲：(0,100]
    - Modeが1である場合、Distanceは距離の値を表示します。単位：mm
  - Index：DO端子の番号。
  - Status：設定するDO状態で、0とOFFは信号なし、1とONは信号ありを示します。

選択可能なパラメータ：

- user：目標点のユーザー座標系。

- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この指令を実行する際のロボットアームの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

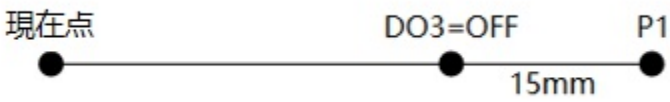
連続経路制御はロボットアームの移動軌跡を変更し、DO出力のタイミングに影響を及ぼしますので、慎重に使用してください。

詳細は一般説明をご参照ください。

例：

```
# ロボットアームは、デフォルト設定ではP1に向けて直線的に移動します。起点から30%の位置に移動した場合、DO2を開として設定します。
MovLIO(P1, [[0, 30, 2, 1]])
```


```
# ロボットアームはデフォルト設定でP1に向けて直線的に移動し、終点から15mmの位置に移動したときに、DO3を閉として設定します。
MovLIO(P1, [[1, -15, 3, 0]])
```



StartPath

プロトタイプ：

```
StartPath(string, {"multi":1, "isConst":0, "sample":50,  "freq":0.2, "user":0, "tool":0})
```

説明：

指定した軌跡ファイル中で記録した軌跡を再現します。この命令を呼び出す前に、ユーザーはロボットアームを軌跡の開始点まで移動する必要があります。

必須パラメータ：

string：軌跡ファイル名（サフィックスを含む）。

選択可能なパラメータ：

- multi：再現する場合の速度倍数で、isConst=0の場合のみ有効。値の範囲：[0.1, 2]、パラメータがない場合のデフォルト値は1です。
- isConst：均一速度で再現するかどうか。パラメータがない場合のデフォルト値は0です。
  - 1は均一速度での再現を示し、ロボットアームは一定のグローバル速度率で軌跡を再現します。
  - 0は軌跡記録時の元の速度で再現することを示します。multiパラメータを使用して移動速度を比例的に調整できます。この場合、ロボットアームの移動速度はグローバル速度率から影響を受けません。
- sample：軌跡点のサンプリング間隔です。つまり、軌跡ファイルを作成した場合の、隣接する2つの点のサンプリング時間差です。値の範囲：[8, 1000]、単位：ms、引数がない場合のデフォルト値は50です（コントローラが軌跡ファイルを記録する場合のサンプリング間隔）。
- freq：フィルタリング係数で、このパラメータの値が小さくなるほど、再現する軌跡曲線はスムーズになります。ただし、元の軌跡に対する変形が厳しくなるほど、元の軌跡の連続経路制御程度に基づいて適切なフィルタリング係数を設定してください。値の範囲：（0,1]、1はフィルタリング終了を示し、引数がない場合のデフォルト値は0.2です。
- user：軌跡点に対応するユーザー座標系インデックスを指定します。指定しない場合、軌跡ファイル中に記録されたユーザー座標系インデックスを使用します。
- tool：軌跡点に対応するツール座標系インデックスを指定します。指定しない場合、軌跡ファイル中に記録されたツール座標系インデックスを使用します。

例：

```
# track.csv軌跡ファイルの開始点まで関節的に移動した後、元の速度の2倍でファイル中に記録された軌跡を再現します。
StartPoint = GetPathStartPose("track.csv")
MovJ(StartPoint)
StartPath("track.csv", {"multi":2, "isConst":0})
```

```
# track.csv軌跡ファイルの開始点まで関節的に移動した後、ファイル中に記録された軌跡を一定速度で再現します。

StartPoint = GetPathStartPose("track.csv")
MovJ(StartPoint)
StartPath("track.csv", {"isConst":1})
```

GetPathStartPose

プロトタイプ：

```
GetPathStartPose(string)
```

説明：

軌跡の開始点を取得します。軌跡ファイルにより、点データのタイプ（ティーチング点/関節/姿勢）も異なる場合があります。

必須パラメータ：

string：軌跡ファイル名（サフィックスを含む）。

例：

```
# track.csv軌跡ファイルの開始点を取得し、印刷します。
StartPoint = GetPathStartPose("track.csv")
print(StartPoint)
```

PositiveKin

プロトタイプ

```
PositiveKin(joint, {"user":1, "tool":0})
```

説明

順運動学の数値演算を実行する：ロボットアームの各関節角度を与え、ロボットアーム末端の与えられたデカルト座標系中の座標値を計算します。

必須パラメータ

Joint：`{"joint":[j1、j2、j3、j4、j5、j6]}` 形式の関節変数です。

選択可能なパラメータ

- user：ユーザー座標系のインデックス。指定していない場合、グローバルユーザー座標系を使用します。
- tool：ツール座標系のインデックス。指定していない場合、グローバルツール座標系を使用します。

戻る

`{"pose":[x、y、z、rx、ry、rz]}` 形式の順運動学法で得られた姿勢変数です。

例

```
PositiveKin({"joint":[0,0,-90,0,90,0]},{"user":1, "tool":1})
```

関節座標は[0,0,-90,0,90,0]であり、ユーザー座標系1と関節座標系1でのロボットアームのエンドのデカルト座標を計算します。

InverseKin

プロトタイプ

```
InverseKin(pose, {"useJointNear":True, "jointNear":joint, "user":1, "tool":0})
```

説明

逆運動学の数値演算を実行する：ロボットアーム末端の与えられたデカルト座標系中の座標値が与えられ、ロボットアームの各関節の角度を計算します。

デカルト座標はTCPの空間座標と傾斜角のみを定義するので、ロボットアームは多種の異なるポーズを通じて同一のポーズに達し、1つのポーズ変数が複数の関節変数に対応できることを意味します。一意の解を得るため、システムは指定された関節座標を必要とし、当該関節座標に最も近い解を逆演算の結果として選択します。

必須パラメータ

pose： `{"pose"：[x、y、z、rx、ry、rz]}` 形式の姿勢変数です。

選択可能なパラメータ

- useJointNear：ブール値。JointNearパラメータが有効かどうかを設定するためのものです。
  - Trueは、JointNearパラメーターに基づいて最も近い解を選択することを意味します。
  - falseまたはパラメータがない場合は、JointNearパラメータが無効であることを意味します。システムはロボットアームの現在の関節に基づいて最も近い解を選択します。
  - JointNearパラメータでなくこのパラメータだけを含む場合は、このパラメータは無効になります。
- JointNear： `{"joint"：[j1、j2、j3、j4、j5、j6]}` 形式で、最も近い解を選択するために使用される関節変数です。
- user：ユーザー座標系のインデックス。指定していない場合、グローバルユーザー座標系を使用します。
- tool：ツール座標系のインデックス。指定していない場合、グローバルツール座標系を使用します。

戻る

- エラー コード。0は逆演算が成功したことを意味し、-1は逆演算が失敗した (解がない) ことを意味します。
- `{"joint"：[j1、j2、j3、j4、j5、j6]}` 形式の逆移動学法で得られた関節変数です。演算が失敗するとj1～j6は全て0になります。

例

```
errId, jointPoint = InverseKin({"pose":[300, 200, 300, 180, 0, 0]}, {"useJointNear" :True, "jo
intNear":{"joint":[90, 30, -90, 180, 30, 0]} })
```

グローバルユーザー座標系とグローバル関節座標系でのロボットアームのエンドのデカルト座標は[300、200、300、180、0、0]であり、関節座標を計算し、関節角度[90、30、-90、180、30、0]から最も近いの解を選択します。

相対移動命令

命令リスト

相対移動命令関数はロボットアームのオフセット移動を制御するために使用されます。ご使用前は共通説明をお読みください。

命令	機能
RelPointUser	点をユーザー座標系に沿って偏移
RelPointTool	点をツール座標系に沿って偏移
RelMovJTool	ツール座標系に沿って相対関節移動を実行
RelMovLTool	ツール座標系に沿って相対直線移動を実行
RelMovJUser	ユーザー座標系に沿って相対関節移動を実行
RelMovLUser	ユーザー座標系に沿って相対直線移動を実行
RelJointMovJ	指定した偏移角度まで関節的に移動
RelJoint	点を指定した関節角度まで偏移

RelPointUser

プロトタイプ：

```
RelPointUser(P, [OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz])
```

説明：

指定点に対して指定したユーザー座標系に沿って偏移し、偏移後の姿勢変数を返します。

必須パラメータ：

- P：変位する指定点。
- [OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz]：ユーザー座標系でのオフセットを指定します。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°
  - 指定点がティーチング点である場合、ティーチング点のユーザー座標系を基にしてオフセットを実行します。
  - 指定点が関節変数または姿勢変数である場合、グローバルユーザー座標系を基にしてオフセットを実行します。

戻る：

オフセット後の姿勢変数：`{"pose":[x, y, z, rx, ry, rz]}`

例：

```
# P1を指定ユーザー座標系において一定距離偏移させ、さらに偏移後の点に移動します。
Offset=[OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz]
p = RelPointUser(P1, Offset)
MovL(p)
```

RelPointTool

プロトタイプ：

```
RelPointTool(P, [OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz])
```

説明：

指定点に対して指定ツール座標系に沿って偏移し、偏移後の姿勢変数を返します。

必須パラメータ：

- P：変位する指定点。
- [OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz]：ツール座標系でのオフセットを指定します。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°
  - 指定点がティーチング点である場合、ティーチング点のツール座標系を基にしてオフセットを実行します。
  - 指定点が関節変数または姿勢変数である場合、グローバルツール座標系を基にしてオフセットを実行します。

戻る：

オフセット後の姿勢変数：`{"pose" :[x, y, z, rx, ry, rz]}`

例：

```
# P1を指定ツール座標系において一定距離偏移させ、さらに偏移後の点に移動します。
Offset=[OffsetX, OffsetY, OffsetZ, OffsetRx, OffsetRy, OffsetRz]
p = RelPointTool(P1, Offset)
MovL(p)
```

RelMovJTool

プロトタイプ：

```
RelMovJTool([x, y, z, rx, ry, rz], {"user":1, "tool":0, "a":20, "v":50, "cp":100})
```

説明：

現在の位置から、指定ツール座標系に沿って、関節移動方式で相対移動を行います。関節移動の軌跡は直線ではなく、すべての関節は同時に移動を完了します。

必須パラメータ：

[x、y、z、rx、ry、rz]：指定されたツール座標系での現在位置を基準とした目標点のオフセット。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定で、グローバルツール座標系に沿って指定の偏移点まで関節的に移動します。
RelMovJTool([10, 10, 10, 0, 0, 0])
```

RelMovLTool

プロトタイプ：

```
RelMovLTool([x, y, z, rx, ry, rz], {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100,
"r":5})
```

説明：

現在の位置から、指定ツール座標系に沿って、直線移動方式で相対移動を行います。

必須パラメータ：

[x、y、z、rx、ry、rz]：指定されたツール座標系での現在位置を基準とした目標点のオフセット。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的で

す。値の範囲：(0,100]
- speed：この命令を実行する場合のロボットアームの移動目標速度で、vと相互に排他的です。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定で、グローバルツール座標系に沿って指定の偏移点まで直線的に移動します。
RelMovLTool([10, 10, 10, 0, 0, 0])
```

RelMovJUser

プロトタイプ：

```
RelMovJUser([x, y, z, rx, ry, rz], {"user":1, "tool":0, "a":20, "v":50, "cp":100})
```

説明：

現在の位置から、指定ユーザー座標系に沿って、関節移動方式で相対移動を行います。関節移動の軌跡は直線ではなく、すべての関節は同時に移動を完了します。

必須パラメータ：

[x、y、z、rx、ry、rz]：指定されたユーザー座標系での現在位置を基準とした目標点のオフセット。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定で、グローバルユーザー座標系に沿って指定の偏移点まで関節的に移動します。
RelMovJUser([10, 10, 10, 0, 0, 0])
```

RelMovLUser

プロトタイプ：

```
RelMovLUser([x, y, z, rx, ry, rz], {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100,
"r":5})
```

説明：

現在の位置から、指定ユーザー座標系に沿って、直線移動方式で相対移動を行います。

必須パラメータ：

[x、y、z、rx、ry、rz]：指定されたユーザー座標系での現在位置を基準とした目標点のオフセット。x，y，zは空間偏移量を示し、単位はmm。rx，ry，rzはオフセット角度を示し、単位は°

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]
- speed：この命令を実行する場合のロボットアームの移動目標速度で、vと相互に排他的です。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定で、グローバルユーザー座標系に沿って指定の偏移点まで直線的に移動します。
RelMovLUser([10, 10, 10, 0, 0, 0])
```

RelJointMovJ

プロトタイプ：

```
RelJointMovJ([Offset1, Offset2, Offset3, Offset4, Offset5, Offset6], {"a":20, "v":50, "cp":100
})
```

説明：

現在の位置から、関節移動方式で指定された関節偏移角度まで移動します。

必須パラメータ：

[Offset1, Offset2, Offset3, Offset4, Offset5, Offset6]：関節座標系におけるJ1軸～J6軸方向のオフセット値（°）。

選択可能なパラメータ：

- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]

詳細は一般説明をご参照ください。

例：

```
# ロボットアームはデフォルト設定に従ってオフセット角度に関節的に移動します。
RelJointMovJ([20,20,10,0,10,0])
```

RelJoint

プロトタイプ：

```
RelJoint(P, [Offset1, Offset2, Offset3, Offset4, offset5, offset6])
```

説明：

指定点に対して関節座標系でJ1～J6軸のオフセット量を増やし、偏移後の関節変数を返します。

必須パラメータ：

- P：変位する指定点。
- Offset1～Offset6：関節座標系でのJ1軸～J6軸方向におけるオフセット値で、単位は°

戻る：

オフセット後の関節変数：`{"joint": [j1, j2, j3, j4, j5, j6]}`

例：

```
# P1をJ1～J6軸上にそれぞれ一定角度偏移させ、さらに偏移後の点に移動します。
Offset = [Offset1, Offset2, Offset3, Offset4, offset5, offset6]
p = RelJoint(P1, Offset)
MovJ(p)
```

移動パラメータ

命令リスト

移動パラメータ関数はロボットアームの移動に関するパラメータを設定または取得するために使用されます。ご使用前に共通説明をお読みください。

命令	機能
CP	連続経路制御の比率の設定
VelJ	関節移動方式の速度比率の設定
AccJ	関節移動方式の加速度比率の設定
VelL	直線と円弧の移動方式の速度比率の設定
AccL	直線と円弧の移動方式の加速度比率の設定
SpeedFactor	ロボットアームのグローバル移動速度の設定
SetPayload	エンド負荷の設定
User	グローバルユーザー座標系の設定
SetUser	指定したユーザー座標系の変更
CalcUser	ユーザー座標系の計算
Tool	グローバルツール座標系の設定
SetTool	指定したツール座標系の変更
CalcTool	ツール座標系の計算
GetPose	ロボットアームのリアルタイム姿勢の取得
GetAngle	ロボットアームのリアルタイム関節角度の取得
GetABZ	エンコーダの現在の位置の取得
CheckMovJ	関節移動命令の実行可能性の確認
CheckMovL	直線または円弧移動命令の実行可能性の確認
SetSafeWallEnable	指定したセーフウォールのオンまたはオフ
SetWorkZoneEnable	指定の干渉領域のオン・オフ
SetCollisionLevel	衝突レベルの設定
SetBackDistance	衝突戻り距離の設定

CP

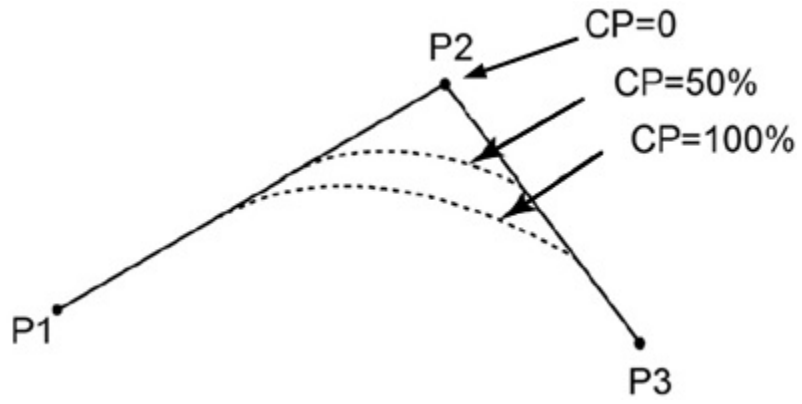
プロトタイプ：

```
CP(R)
```

説明：

連続経路制御の比率を設定します。すなわち、ロボットアームが複数の点を経由して連続的に移動するとき、直角方式でそれとも曲線方式で中間点を通過するのかを設定します。

このコマンドのが設定する連続経路制御の比率は、現在のプロジェクト実行においてのみ有効です。設定していない場合のデフォルト値は0です。



必須パラメータ：

R：連続経路制御の比率。値の範囲：[0, 100]

例：

```
# ロボットアームはP1からP2を経由してP3まで移動する場合、50%で滑らかに移動します。
CP(50)
MovL(P1)
MovL(P2)
MovL(P3)
```

VelJ

プロトタイプ：

```
VelJ(R)
```

説明：

関節移動方式（MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ）の速度比率を設定します。

このコマンドが設定する速度比率は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：速度比率。値の範囲：[1, 100]

例：

```
# ロボットアームは20%の速度比率でP1まで移動します。
VelJ(20)
MovJ(P1)
```

AccJ

プロトタイプ：

```
AccJ(R)
```

説明：

関節移動方式（MovJ/MovJIO/RelMovJTool/RelMovJUser/RelJointMovJ）の加速度比率を設定します。

このコマンドが設定する加速度比率は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：加速度比率。値の範囲：[1, 100]

例：

```
# ロボットアームは50%の速度比率でP1まで移動します。
AccJ(50)
MovJ(P1)
```

VelL

プロトタイプ：

```
VelL(R)
```

説明：

直線および曲線移動方式（MovL/Arc/Circle/MovLIO/RelMovLTool/RelMovLUser）の速度比率を設定します。

このコマンドが設定する速度比率は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：速度比率。値の範囲：[1, 100]

例：

```
# ロボットアームは20%の速度比率でP1まで移動します。
VelL(20)
MovL(P1)
```

AccL

プロトタイプ：

```
AccL(R)
```

説明：

直線および曲線移動方式（MovL/Arc/Circle/MovLIO/RelMovLTool/RelMovLUser）の加速度比率を設定します。

このコマンドが設定する加速度比率は、現在のプロジェクト実行中のみで有効です。設定していない場合のデフォルト値は100です。

必須パラメータ：

R：加速度比率。値の範囲：[1, 100]

例：

```
# ロボットアームは50%の速度比率でP1まで移動します。
AccL(50)
MovL(P1)
```

SpeedFactor

プロトタイプ：

```
SpeedFactor(ratio)
```

説明：

ロボットアームの全体的な移動速度を設定します。詳細は移動指令の一般的な説明をご参照ください。

このコマンドで設定された全体的な速度は現在のプロジェクトの実行中にのみ有効で、設定されていない場合は、スクリプトの実行前の制御ソフトウェア（制御ソフトウェアでスクリプトを実行する場合）またはTCP指令（TCP指令でスクリプトを実行する場合）の設定値を引き続き使用します。

必須パラメータ：

ratio：移動速度比率。値の範囲：(0,100)

例：

```
# グローバル移動速度の設定は50%です。
SpeedFactor(50)
```

SetPayload

プロトタイプ：

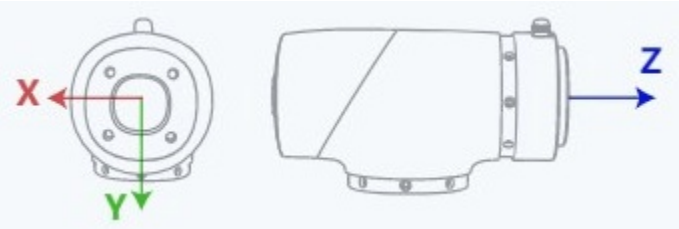
```
SetPayload(payload, [x, y, z])
SetPayload(name)
```

説明：

エンド負荷の重量と偏心座標を設定します。直接の設定と、事前設定パラメータセット方式の2種類の設定をサポートしています。このコマンドで設定したパラメータは、現在のプロジェクトの実行中のみ有効で、前の設定を上書きします。プロジェクトが停止すると、元の設定に戻ります。

必須パラメータ**1**：

- payload：エンドの負荷重量。単位：kg
- [x、y、z]：エンド負荷の偏心座標です。座標軸の方向は下図をご参照ください。単位：mm



例**1**：

```
# エンド負荷重量は1.5kgで、偏心座標は[0、0、10]です。
```

```
SetPayload(1.5, [0, 0, 10])
```

必須パラメータ**2**：

- name：事前設定パラメータセット名称で、まず制御ソフトウェアで対応するパラメータセットを保存しなければなりません。

例**2**：

```
# load1という名前の負荷パラメータグループをエンド負荷に設定します。
SetPayload("load1")
```

User

プロトタイプ：

```
User(user)
```

説明：

グローバルユーザー座標系を設定します。ユーザーがその他命令を呼び出す場合、オプションパラメータによりユーザー座標系を指定しない場合、システムはグローバルユーザー座標系を使用することができます。

このコマンドが設定するグローバルユーザー座標系は、現在のプロジェクト実行中でのみ有効です。設定していない場合には、デフォルトのグローバルユーザー座標系はユーザー座標系0です。

必須パラメータ：

user：切り換え後のユーザー座標系で、インデックス番号により指定します。まず制御ソフトウェアで対応する座標系を追加してください。指定した座標系のインデックスが存在しない場合、プロジェクトの実行が停止し、エラーが報告されます。

例：

```
# グローバルユーザー座標系をユーザー座標系1に切り替えます。
User(1)
```

SetUser

プロトタイプ：

```
SetUser(index,table)
```

説明：

指定したユーザー座標系を変更します。このコマンドで変更した座標系は、現在のプロジェクトの実行中のみ有効です。プロジェクトが停止すると、元の値に戻ります。

必須パラメータ：

- index：ユーザー座標系インデックスです。値の範囲は[0,50]、座標系0の初期値はベース座標系です。
- table：変更されたユーザー座標系です。形式は[x、y、z、rx、ry、rz]で、CalcUser命令で取得することをお勧めします。

例：

```
# ユーザー座標系1をCalcUserによって計算された新しい座標系に変更します。
newUser = CalcUser(1,1,[10,10,10,10,10,10])
SetUser(1,newUser)
```

CalcUser

プロトタイプ：

```
CalcUser(index,matrix_direction,table)
```

説明：

ユーザー座標系を計算します。

必須パラメータ：

- index：ユーザー座標系インデックスです。値の範囲は[0,50]で、ユーザー座標系0の初期値はベース座標系です。
- matrix_direction：計算の方向。
    - 1：左乗算。indexで指定した座標系が基座標系に沿ってtableで指定した値だけ偏向しま

す。
    - 0：右乗算。indexで指定した座標系が自身に沿ってtableで指定した値だけ偏向します。
- table：ユーザー座標系オフセット値です。形式は[x、y、z、rx、ry、rz]です。

戻る：

計算された[x、y、z、rx、ry、rz]形式のユーザー座標系です。

例：

```
# 計算過程は次のように等価と考えることができます：初期位置姿勢がユーザー座標系1と同じ座標系が、ベース座標系に沿って[x=10、y=10、z=10]だけ平行移動し、[rx=10、ry=10、rz=10]だけ回転した後に得られた新しい座標系はnewUserです。
newUser = CalcUser(1,1,[10,10,10,10,10,10])
```

```
# 計算過程は次のように等価と考えることができます：初期位置姿勢がユーザー座標系1と同じ座標系が、ユーザー座標系1に沿って[x=10、y=10、z=10]だけ平行移動し、[rx=10、ry=10、rz=10]だけ回転した後に得られた新しい座標系はnewUserになります。
newUser = CalcUser(1,0,[10,10,10,10,10,10])
```

Tool

プロトタイプ：

```
Tool(tool)
```

説明：

グローバルツール座標系を切り替えます。ユーザーがその他命令を呼び出す場合、オプションパラメータによりツール座標系を指定しない場合、システムはグローバルツール座標系を使用します。

このコマンドが設定するグローバルツール座標系は、現在のプロジェクト実行中でのみ有効です。設定していない場合には、デフォルトのグローバルツール座標系は、ツール座標系0です。

必須パラメータ：

tool：切り換え後のツール座標系で、インデックス番号により指定します。まず制御ソフトウェア中に対応する座標系を追加してください。指定した座標系のインデックスが存在しない場合、プロジェクトの実行が停止し、エラーが報告されます。

例：

```
# グローバルツール座標系をツール座標系1に切り替えます。
Tool(1)
```

SetTool

プロトタイプ：

```
SetTool(index,table)
```

説明：

指定したツール座標系を変更します。このコマンドで変更した座標系は、現在のプロジェクトの実行中のみ有効です。プロジェクトが停止すると、元の値に戻ります。

必須パラメータ：

- index：ツール座標系インデックスです。値の範囲は[0,50]、ここで座標系0の初期値はフランジ座標系（TCP0）です。
- table：変更されたツール座標系です。形式は[x、y、z、rx、ry、rz]で、CalcTool命令で取得することをお勧めします。

例：

```
# ツール座標系1をCalcToolで計算された新しい座標系に変更します。
newTool = CalcTool(1,1,[10,10,10,10,10,10])
SetTool(1,newTool)
```

CalcTool

プロトタイプ：

```
CalcTool(index,matrix_direction,table)
```

説明：

ツール座標系を計算します。

必須パラメータ：

- index：ツール座標系インデックスです。値の範囲は[0,50]で、座標系0の初期値によれば、フランジ座標系（TCP0）になります。
- matrix_direction：計算の方向。
  - 1：左乗算。indexで指定した座標系がフランジ座標系（TCP0）に沿ってtableで指定した値だけ偏向します。
  - 0：右乗算。indexで指定した座標系が自身に沿ってtableで指定した値だけ偏向します。
- table：ツール座標系です。形式は[x、y、z、rx、ry、rz]です。

戻る：

[x、y、z、rx、ry、rz]形式の計算されたツール座標系です。

例：

```
# 計算過程は次のように等価と考えることができます：初期位置姿勢がツール座標系1と同じ座標系が、フランジ座標系（TCP0）に沿って[x=10、y=10、z=10]だけ平行移動し、[rx=10、ry=10、rz=10]だけ回転した後に得られた新しい座標系はnewToolになります。
newTool = CalcTool(1,1,[10,10,10,10,10,10])
```

```
# 計算過程は次のように等価と考えることができます：初期位置姿勢がツール座標系1と同じ座標系が、ツール座標系1に沿って[x=10、y=10、z=10]だけ平行移動し、[rx=10、ry=10、rz=10]だけ回転した後に得られた新しい座標系はnewToolになります。
newTool = CalcTool(1,0,[10,10,10,10,10,10])
```

GetPose

プロトタイプ：

```
GetPose(user_index, tool_index)
```

説明：

ロボットアームのリアルタイムの姿勢を取得します。

2つの移動命令間でこの命令を呼び出す場合、得られるポイントは連続経路制御設定の影響を受けます。

- 連続経路制御機能（cpまたはrは0）が閉じている場合、目標点を正確に取得することができます。
- 連続経路制御機能が起動している場合、移動曲線上のポイントを得ることができます。

選択可能なパラメータ：

- user_index：姿勢に対応するユーザー座標系インデックスで、まず制御ソフトウェア中で対応する座標系を追加してください。設定していない場合、グローバルユーザー座標系を使用します。
- tool_index：姿勢に対応するツール座標系インデックスで、まず制御ソフトウェア中で対応する座標系を追加してください。設定していない場合、グローバルツール座標系を使用します。

戻る：

ロボットアームの現在の姿勢：`{"pose"：[x、y、z、rx、ry、rz]}`

例：

```
# ロボットアームはまずP1まで移動し、さらに現在の姿勢に戻ります。
```

```
currentPose = GetPose()
MovJ(P1)
MovJ(currentPose)
```

```
# ロボットアームはP1まで移動した後に、さらにP2に移動し、P1の姿勢を取得します。
MovJ(P1,{"cp":0})
currentPose = GetPose() #P1の姿勢を取得
MovJ(P2)
```

GetAngle

プロトタイプ：

```
GetAngle()
```

説明：

ロボットアームのリアルタイムの関節角度を取得します。

2つの移動命令間でこの命令を呼び出す場合、得られる間接角度は連続経路制御設定の影響を受ける場合があります。

- 連続経路制御機能（cpまたはrは0）が閉じている場合、目標点に対応する関節角度を正確に取得することができます。
- 連続経路制御機能が起動している場合、移動曲線上のポイントに対応する関節角度を得ることができます。

戻る：

ロボットアームの現在の関節角度: `{"joint": [j1、j2、j3、j4、j5、j6]}`

例：

```
# ロボットアームはまずP1まで移動し、さらに現在の姿勢に戻ります。
currentAngle = GetAngle()
MovJ(P1)
MovJ(currentAngle)
```

```
# ロボットアームはP1に移動した後、さらにP2に移動し、P1の関節角度を取得します。
MovJ(P1,{"cp":0})
currentAngle = GetAngle()  #P1の関節角度を取得する
MovJ(P2)
```

GetABZ

プロトタイプ：

```
GetABZ()
```

説明：

エンコーダの現在の位置を取得します。

戻り値：

エンコーダの現在の位置。

例：

```
# 設定済みのABZエンコーダの現在の位置を取得し、変数abzに値を与えます。
abz = GetABZ()
```

CheckMovJ

プロトタイプ：

```
CheckMovJ(P, {"user":1, "tool":0, "a":20, "v":50, "cp":100})
```

説明：

現在の点から目標点までの関節移動の実行可能性を確認します。システムは全体の移動経路を計算し、経路中に到達不能な点がないかを確認します。

必須パラメータ：

P：目標点。

選択可能なパラメータ：

- user：目標点のユーザー座標系。
- tool：目標点のツール座標系。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率。値の範囲：(0,100]
- cp：連続経路制御の比率。値の範囲：[0,100]

詳細は一般説明をご参照ください。

戻る：

チェック結果。

- 0：エラーなし
- 16：終点がショルダーの特異点に近接している

- 17：終点逆演算が解なし
- 18：終点逆演算が位置制限される
- 22：ジェスチャー切り替えエラー
- 26：終点が腕部の特異点に近接している
- 27：終点が肘部の特異点に近接している
- 29：速度パラメータエラー
- 32：軌跡にショルダーの特異点がある
- 33：軌跡に逆演算解なし点が存在する
- 34：軌跡に逆演算位置制限点が存在する
- 35：軌跡に腕部の特異点がある
- 36：軌跡に肘部の特異点がある
- 37：軌跡に関節ジャンピング点が存在する

例：

```
# 関節移動のデフォルト設定でP1に達するかを確認します。達する場合、関節はP1まで移動します。
status=CheckMovJ(P1)
if status==0:
    MovJ(P1)
    status=CheckMovJ(P2) #P1からP2への実行可能性の確認はロボットアームがP1に移動した後に実行する必要があります。
    if(status==0)
        MovJ(P2)
```

CheckMovL

プロトタイプ：

```
CheckMovL(P, {"user":1, "tool":0, "a":20, "v":50, "speed":500, "cp":100, "r":5})
```

説明：

現在の点から目標点までの直線移動の実行可能性を確認します。システムは全体の移動経路を計算し、経路中に到達不能な点がないかを確認します。

必須パラメータ：

P：目標点。

選択可能なパラメータ：

- user：目標点のユーザー座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- tool：目標点のツール座標系。目標点が関節変数の場合、座標系パラメータは無効です。
- a：この命令を実行する場合のロボットアームの移動加速度比率。値の範囲：(0,100]
- v：この命令を実行する場合のロボットアームの移動速度比率で、speedと相互に排他的です。値の範囲：(0,100]

- speed：この指令を実行する際のロボットアームの目標速度。vとは互いに排他的で、両方が存在する場合はspeedを優先します。値の範囲：[1、最大移動速度]、単位：mm/s
- Cp：連続経路制御の比率で、rと相互に排他的です。値の範囲：[0,100]
- r：連続経路制御半径で、cpと相互に排他的です。同時に存在する場合にはrを基準とします。値の範囲：[0,100]、単位：mm

詳細は一般説明をご参照ください。

戻る：

チェック結果。

- 0：エラーなし
- 16：終点がショルダーの特異点に近接している
- 17：終点逆演算が解なし
- 18：終点逆演算が位置制限される
- 22：ジェスチャー切り替えエラー
- 26：終点が腕部の特異点に近接している
- 27：終点が肘部の特異点に近接している
- 29：速度パラメータエラー
- 32：軌跡にショルダーの特異点がある
- 33：軌跡に逆演算解なし点が存在する
- 34：軌跡に逆演算位置制限点が存在する
- 35：軌跡に腕部の特異点がある
- 36：軌跡に肘部の特異点がある
- 37：軌跡に関節ジャンピング点が存在する

例：

```
# 直線移動を使用してデフォルト設定でP1まで達することができるかを確認します。達する場合、P1まで直線的に移動します。
status=CheckMovL(P1)
if(status==0)
    MovL(P1)
    status=CheckMovL(P2) # P1からP2への実行可能性の確認は、ロボットアームがP1に移動した後に実行する必要があります。
    if(status==0)
        MovL(P2)
```

SetSafeWallEnable

プロトタイプ：

```
SetSafeWallEnable(index,value)
```

説明：

指定されたセーフウォールをオンまたはオフにします。このコマンドで設定したセーフウォールの状態は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

- index：設定するセーフウォールインデックスで、まず制御ソフトウェア中で対応するセーフウォールを追加してください。値の範囲：[1,8]
- value：
  - True：セーフウォールをオンにします
  - False：セーフウォールをオフにします

例：

```
# インデックスが1のセーフウォールをオンにします。
SetSafeWallEnable(1,True)
```

SetWorkZoneEnable

プロトタイプ：

```
SetWorkZoneEnable(index,value)
```

説明：

指定した干渉領域をオンまたはオフにします。このコマンドで設定した干渉干渉領域の状態は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

- index：設定する干渉領域のインデックス。事前に制御ソフトウェアで対応する干渉領域を追加する必要があります。値の範囲：[1,6]
- value：
  - True：干渉領域をオンにします
  - False：干渉領域をオフにします

例：

```
# インデックス1の干渉領域を開きます。
SetWorkZoneEnable(1,True)
```

SetCollisionLevel

プロトタイプ：

```
SetCollisionLevel(level)
```

説明：

衝突検出レベルを設定します。このコマンドで設定した値は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

level：衝突検出等級で、0は衝突検出終了、1～5の数字では、大きくなるほど感度が高くなります

例：

```
# 衝突検出レベルを3で設定します。
SetCollisionLevel(3)
```

SetBackDistance

プロトタイプ：

```
SetBackDistance(distance)
```

説明：

ロボットアームが衝突を検出してから元のルートに戻るまでの距離を設定します。このコマンドで設定した値は、現在のプロジェクトが実行中であるだけで有効で、プロジェクトが停止すると元の値に戻ります。

必須パラメータ：

distance：衝突して戻る距離で、値の範囲は：[0,50]、単位：mm

例：

```
# 衝突して戻る距離を20mmで設定します。
SetBackDistance(20)
```

IO

命令リスト

IO命令は、ロボットアームシステムのIOの読み書きおよび関連パラメータの設定を行うために使用されます。

命令	機能
DI	DIポートのステータスの取得
DIGroup	複数のDIポートの状態の取得
DO	デジタル出力ポートの状態を設定
DOGroup	複数のデジタル出力ポートの状態を設定
GetDO	デジタル出力ポートの現在の状態の取得
GetDOGroup	複数のデジタル出力ポートの現在の状態の取得
AI	AIポートの値の取得
AO	アナログ出力ポートの値を設定
GetAO	アナログ出力ポートの現在の値の取得

DI

プロトタイプ：

```
DI(index)
```

説明：

デジタル入力ポートの状態を読み込みます。

必須パラメータ：

index：DI端子の番号。

戻る：

対応するDI端子の状態（ON/OFF）。

例：

```
# DI1がONの場合、ロボットアームはP1まで直線移動します。
if DI(1)==ON:
    MovL(P1)
```

DIGroup

プロトタイプ：

```
DIGroup(index1,...,indexn)
```

説明：

複数のデジタル入力ポートの状態を読み取ります。

必須パラメータ：

index：DI端子の番号。コンマで区切って複数入力できます。

戻る：

対応するDI端子の状態（ON/OFF）を配列として返します。

例：

```
# DI1とDI2の両方がONの場合、ロボットアームはP1まで直線移動します。
digroup = DIGroup(1,2)
if digroup[1]&digroup[2]==ON:
    MovL(P1)
```

DO

プロトタイプ：

```
DO(index,ON|OFF,time_ms)
```

説明：

デジタル出力ポートの状態を設定します。

必須パラメータ：

- index：DO端子の番号。
- ON|OFF：設定するDOポートの状態。

選択可能なパラメータ：

time_ms：[25, 60000]の範囲で、継続的な出力時間（ms）。このパラメータを設定すると、指定した時間後にDOが自動的に反転されます。反転は非同期動作であり、命令キューをブロックすることはありません。DO出力が実行されると次の命令が実行されます。

例：

```
# DO1をONに設定します。
DO(1,ON)
```

```
# DO1をONに設定し、50ms後に自動的にOFFに設定します。
DO(1,ON,50)
```

DOGroup

プロトタイプ：

```
DOGroup([index1,ON|OFF],..,[indexN,ON|OFF])
```

説明：

複数のデジタル出力ポートの状態を設定します。

必須パラメータ：

- index：DO端子の番号。
- ON|OFF：設定するDOポートの状態。

複数のグループを設定できます。各グループは中括弧で囲まれ、グループの間はコンマで区切られます。

例：

```
# DO1とDO2をONに設定します。
DOGroup([1,ON],[2,ON])
```

GetDO

プロトタイプ：

```
GetDO(index)
```

説明：

デジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：DO端子の番号。

戻る

対応するDI端子の状態（ON/OFF）。

例：

```
# DO1の現在の状態を取得します。
GetDO(1)
```

GetDOGroup

プロトタイプ：

```
GetDOGroup(index1,...,indexN)
```

説明：

複数のデジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：DO端子の番号。コンマで区切って複数入力できます。

戻る：

対応DO端子の状態（ON/OFF）を配列として返します。

例：

```
# DO1とDO2の現在の状態を取得します。
GetDOGroup(1,2)
```

AI

プロトタイプ：

```
AI(index)
```

説明：

アナログ入力ポートの値を読み込みます。

必須パラメータ：

index：AI端子の番号。

戻る：

対応するAI端子の値。

例：

```
# AI1の値を読み取り、変数testに代入します。
test = AI(1)
```

AO

プロトタイプ：

```
AO(index,value)
```

説明：

アナログ出力ポートの値を設定します。

必須パラメータ：

- index：AO端子の番号。
- value：設定する値。電圧範囲：[0,10]、単位：V、電流範囲：[4,20]、単位：mA

例：

```
# AO1の出力値を2に設定します。
AO(1,2)
```

GetAO

プロトタイプ：

```
GetAO(index)
```

説明：

アナログ出力ポートの現在の値を取得します。

必須パラメータ：

index：AO端子の番号。

戻る

対応するAO端子の値。

例：

```
# AO1の現在の状態を取得します。
GetAO(1)
```

エンドツール

命令リスト

エンドツール命令は、ロボットアームシステムのエンドIOの読み取り/書き込みおよび関連パラメータの設定に使用します。

命令	機能
ToolDI	エンドデジタル入力ポートの状態を読み取ります
ToolDO	エンドデジタル出力ポートの状態を設定します(キュー命令)
GetToolDO	エンドデジタル出力ポートの現在の状態を取得します
SetToolPower	エンドツールの電源状態を設定します

ToolDI

プロトタイプ：

```
ToolDI(index)
```

説明：

エンドデジタル入力ポートの状態を読み取ります。

必須パラメータ：

index：エンドDI端子の番号。

戻る：

対応するDI端子の状態（ON/OFF）。

例：

```
# エンドDI1が0であるとき、ロボットアームが直線移動方式でP1に移動します。
if ToolDI(1)==ON:
    MovL(P1)
```

ToolDO

プロトタイプ：

```
ToolDO(index,ON|OFF)
```

説明：

エンドデジタル出力ポートの状態を設定します。

必須パラメータ：

- index：エンドDO端子の番号。
- ON|OF：設定するDOポートの状態。

例：

```
# エンドDO1をONに設定します。
ToolDO(1,ON)
```

GetToolDO

プロトタイプ：

```
GetToolDO(index)
```

説明：

エンドデジタル出力ポートの現在の状態を取得します。

必須パラメータ：

index：エンドDO端子の番号。

戻る

対応するエンドDO端子の状態（ON/OFF）。

例：

```
# エンドDO1の現在の状態を取得します。
GetToolDO(1)
```

SetToolPower

プロトタイプ：

```
SetToolPower(status)
```

説明：

エンドツールの電源供給状態を設定します。通常、エンド電源を再起動するために使用されます。例えば、エンドエフェクタの電源を再投入して初期化します。このインターフェースを連続して呼び出す場合は、少なくとも4ms以上の間隔をお勧めします。

必須パラメータ：

status：エンドツールの電源状態。0：電源オフ、1：電源オン

例：

```
# エンドツールの電源を入れ直します。
SetToolPower(0)
Wait(5)
SetToolPower(1)
```

TCP&UDP

命令リスト

TCP&UDP関数は、TCPやUDP通信を行うために使用します。

命令	機能
TCPCreate	TCPネットワークオブジェクトを作成
TCPStart	TCP接続を確立
TCPRead	TCP相手側が送信したデータを受信
TCPWrite	TCP相手側にデータを送信
TCPDestroy	TCP接続を切断し、socketオブジェクトを廃棄
UDPCreate	UDPネットワークオブジェクトを作成
UDPRead	UDP相手側が送信したデータを受信
UDPWrite	UDP相手側にデータを送信

TCPCreate

プロトタイプ：

```
TCPCreate(isServer, IP, port)
```

説明：

TCPネットワークオブジェクトは、1つのみ作成することができます。

必須パラメータ：

- isServer：サーバーを作成するかどうか。true：サーバーを作成することを表します。false：クライアントを作成することを表します。
- IP：サーバーIPアドレス。クライアントIPアドレスと同じネットワークセグメント上にあり、かつ競合しないひょうにしなければなりません。サーバー作成時はロボットアームのIPアドレスとし、クライアント作成時は相手側のアドレスとします。
- port：サーバーポート。システムによって占用されている下記ポートは使用しないでください。これらのポートを使用すると、サーバー作成が失敗する恐れがあります。

  7、13、22、37、139、445、502、503（0～1024の間のポートは、linuxシステムのカスタマポートであり、占用されている可能性が高いため、できるだけ使用しないでください）。

1501, 1502, 1503, 4840, 8172, 9527,

11740, 22000, 22001, 29999, 30004, 30005, 30006,

60000~65504, 65506, 65511~65515, 65521, 65522

戻る：

- err：0はTCPネットワークオブジェクト作成が成功したことを表し、1はTCPネットワークオブジェクト作成が失敗したことを表します。
- socket：作成したsocketオブジェクト。

例**1**：

```
# TCPサーバーを作成します。
ip="192.168.5.1" # ロボットアームのIPアドレスをサーバーのIPアドレスとします
port=6001 # サーバーポート
err=0
socket=0
err, socket = TCPCreate(true, ip, port)
```

例**2**：

```
# TCPクライアントを作成します。
ip="192.168.5.25" # カメラなどの外部設備のIPアドレスをサーバーのIPアドレスとします
port=6001 # サーバーポート
err=0
socket=0
err, socket = TCPCreate(false, ip, port)
```

TCPStart

プロトタイプ：

```
TCPStart(socket, timeout)
```

説明：

TCP接続を確立します。ロボットアームをサーバーとするとき、クライアントが接続するのを待機します。ロボットアームをクライアントとするとき、サーバーに自発的に接続します。

必須パラメータ：

- socket：作成したsocketオブジェクト。
- timeout：待機タイムアウト時間。単位：秒。0に設定した場合、接続の確立が成功するまで待機します。0ではない場合、設定した時間を超えると接続に失敗したと返されます。

戻る：

接続結果。

- 0：接続は成功しました
- 1：入力パラメータエラー
- 2：socketオブジェクトが存在しません
- 3：設定タイムアウト時間エラー
- 4：接続に失敗しました

例：

```
# TCP接続の確立を開始し、接続の確立が成功するまで待機します。
err = TCPStart(socket, 0) # socketは、TCPCreateから正常に返されるsocketオブジェクトです
```

TCPRead

プロトタイプ：

```
TCPRead(socket, timeout, type)
```

説明：

TCP相手側が送信したデータを受信します。

必須パラメータ：

socket：作成したsocketオブジェクト。

選択可能なパラメータ：

- timeout：待機タイムアウト時間。単位：秒。設定しないか0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定されていない場合、RecBufキャッシュの形式はリストになります。「string」に設定されている場合、RecBufキャッシュの形式は文字列になります。

戻る：

- err：0はデータの受信が成功したことを表し、1はデータの受信が失敗したことを表します。
- Recbuf：受信データバッファエリア。

例：

```
# TCPデータを受信します。受信したデータをそれぞれ文字列とtable形式で保存します。
# socketはTCPCreateから正常に返されたsocketオブジェクトです。
err, RecBuf = TCPRead(socket,0,"string") # RecBufデータタイプは文字列です
err, RecBuf = TCPRead(socket,0) # RecBufデータ型はリストです
```

TCPWrite

プロトタイプ：

```
TCPWrite(socket, buf, timeout)
```

説明：

TCP相手側にデータを送信します。

必須パラメータ：

- socket：作成したsocketオブジェクト。
- buf：送信対象のデータ。

選択可能なパラメータ：

timeout：待機タイムアウト時間。単位：秒。設定しないか0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

結果を送信します。

- 0：送信は成功しました。
- 1：送信は失敗しました

例：

```
# TCPデータを送信します。データコンテンツは「test」とします。
TCPWrite(socket, "test") # socketは、TCPCreateから正常に返されるsocketオブジェクトです
```

TCPDestroy

プロトタイプ：

```
TCPDestroy(socket)
```

説明：

TCP接続を切断し、socketオブジェクトを廃棄します。

必須パラメータ：

socket：作成したsocketオブジェクト。

戻る：

実行結果。

- 0：実行は成功しました
- 1：実行は失敗しました

例：

```
# TCP相手側との接続を切断します。
TCPDestroy(socket) # socketは、TCPCreateから正常に返されるsocketオブジェクトです
```

UDPCreate

プロトタイプ：

```
UDPCreate(isServer, IP, port)
```

説明：

UDPネットワークオブジェクトは、1つのみ作成することができます。

必須パラメータ：

- isServer：固定的にfalseを記入します。
- IP：相手側IPアドレス。ロボットアームIPアドレスと同じネットワークセグメント上にあり、かつ競合しない必要があります。
- port：相手側ポート。

戻る：

- err：0はUDPネットワークオブジェクト作成が成功したことを表し、1はTCPネットワークオブジェクト作成が失敗したことを表します。
- socket：作成したsocketオブジェクト。

例：

```
# UDPネットワークオブジェクトを作成します。
ip="192.168.5.25" # カメラなどの外部設備のIPアドレスを相手側のIPアドレスとします
port=6001 # 相手側ポート
err=0
socket=0
err, socket = UDPCreate(false, ip, port)
```

UDPRead

プロトタイプ：

```
UDPRead(socket, timeout, type)
```

説明：

UDP相手側が送信したデータを受信します。

必須パラメータ：

socket：作成したsocketオブジェクト。

選択可能なパラメータ：

- timeout：待機タイムアウト時間。単位：秒。設定しないか0に設定した場合、データ読み取りが完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。
- type：戻り値のタイプ。設定されていない場合、RecBufキャッシュの形式はリストになります。「string」に設定されている場合、RecBufキャッシュの形式は文字列になります。

戻る：

- err：0はデータの受信が成功したことを表し、1はデータの受信が失敗したことを表します。
- Recbuf：受信データバッファエリア。

例：

```
# UDPデータを受信します。受信したデータをそれぞれ文字列とtable形式で保存します。
# socketはUDPCreateから正常に返されたsocketオブジェクトです。
err, RecBuf = UDPRead(socket,0,"string") # RecBufデータタイプは文字列です
err, RecBuf = UDPRead(socket, 0) # RecBufデータ型はリストです
```

UDPWrite

プロトタイプ：

```
UDPWrite(socket, buf, timeout)
```

説明：

UDP相手側にデータを送信します。

必須パラメータ：

- socket：作成したsocketオブジェクト。
- buf：送信対象のデータ。

選択可能なパラメータ：

timeout：待機タイムアウト時間。単位：秒。設定しないか0に設定した場合、相手側がデータ受信を完了するまで待機します。0ではない場合、設定した時間を超えると直接次の実行に進みます。

戻る：

結果を送信します。

- 0：送信は成功しました。
- 1：送信は失敗しました

例：

```
# UDPデータを送信します。データコンテンツは「test」とします。
UDPWrite(socket, "test") # socketは、UDPCreateから正常に返されるsocketオブジェクトです
```

Modbus

命令リスト

Modbus関数は、Modbusマスターとスレーブ間の通信を確立するために使用されます。レジスタアドレスの取得範囲と定義については、対応するスレーブのModbusレジスタアドレス定義説明をご参照ください。

命令	機能
ModbusCreate	Modbusマスターを作成
ModbusRTUCreate	RS485インターフェースを基にしたModbusマスターを作成
ModbusClose	Modbusスレーブとの接続を解除
GetInBits	接点レジスタの読み取り
GetInRegs	入力レジスタの読み取り
GetCoils	コイルレジスタの読み取り
SetCoils	コイルレジスタの書き込み
GetHoldRegs	保持レジスタの読み取り
SetHoldRegs	保持レジスタの書き込み

各種レジスタに対応するModbus機能コードは、標準のModbusプロトコルに従います。

レジスタタイプ	読み取りレジスタ	単一レジスタの書き込み	複数レジスタの書き込み
コイルレジスタ	01	05	0F
接点レジスタ	02	-	-
入力レジスタ	04	-	-
保持レジスタ	03	06	10

ModbusCreate

プロトタイプ：

```
ModbusCreate(IP, port, slave_id, isRTU)
```

説明：

Modbusマスターを作成し、スレーブとの接続を確立します。最大で15の機器との接続を同時にサポートします。

ロボットが搭載しているスレーブに接続する場合は、IPをロボットのIP（デフォルトは192.168.5.1、変更可能）に設定し、ポートを502（map1）または1502（map2）に設定します。詳細は付属書A Modbusレジスタの定義を参照してください。

第三者のスレーブに接続する場合、IPとポートは第三者のスレーブのアドレスであり、レジスタを読み書きする際のレジスタアドレスの範囲と定義は、対応するスレーブのModbusレジスタアドレス定義説明を参照してください。

必須パラメータ：

- IP：スレーブのIPアドレス。
- port：スレーブのポート。

選択可能なパラメータ：

- slave_id：スレーブID。
- isRTU：ブール値。非搭載またはFasleの場合はModbusTCP通信を確立し、Trueの場合はModbusRTU通信を確立します。

> ⚠ 注意：
> このパラメータは、接続確立後のデータ転送に使用するプロトコル形式を決定しますが、接続結果には影響しません。したがって、マスタを作成する際にこのパラメータを誤って設定した場合でも、作成は成功しますが、その後の通信では異常が発生する可能性があります。

戻る：

- err：
  - 0：Modbusマスターの作成に成功しました
  - 1：作成したマスターが最大数に達しているため、新しいマスターの作成に失敗しました
  - 2：マスターの初期化に失敗しました。IP、ポート、ネットワークの状態などを確認することをお勧めします
  - 3：スレーブへの接続に失敗しました。スレーブが正常に設立されているか、ネットワークが正常であるかなどを確認することをお勧めします
- id：返されたマスターインデックスで、その他Modbus命令を後で呼び出すときに使用します。値の範囲は[0, 14]です

例：

```
# Modbusマスターを作成し、指定したスレーブと接続を確立します。
ip="192.168.5.123"
```

```
port=503
err=0
id=0
err, id = ModbusCreate(ip, port, 1)
```

```
# Modbusマスターを作成し、ロボット自身のスレーブと接続を確立します。
ip="192.168.5.1"
port=502
err=0
id=0
err, id = ModbusCreate(ip, port)
```

ModbusRTUCreate

プロトタイプ：

```
ModbusRTUCreate(slave_id, baud, parity, data_bit, stop_bit)
```

説明：

RS485インターフェースを基にしたModbusマスターを作成し、スレーブとの接続を確立します。最大で15の機器との接続を同時にサポートします。

必須パラメータ：

- slave_id：スレーブID。
- baud：RS485インターフェースのボーレート。

選択可能なパラメータ：

- parity：パリティビットかどうか。「O」は奇数パリティを、「E」は偶数パリティを、「N」はパリティビットなしを示します。パラメータなしの場合、デフォルト値は「E」です。
- data_bit：データビット長さ。値の範囲は8です。パラメータなしの場合、デフォルト値は8です。
- stopbit：ストップビット長さ。値の範囲は1と2です。パラメータなしの場合、デフォルト値は1です。

戻る：

- err：0はModbusマスターの作成に成功したことを示し、1はModbusマスターの作成に失敗したことを示します。
- id：返されたマスターインデックスで、その他Modbus命令を後で呼び出すときに使用します。

例：

```
# Modbusマスターを作成し、RS485インターフェースに接続されたスレーブとの接続を確立します（スレーブIDは1、ボーレートは115200。）
err, id = ModbusRTUCreate(1, 115200)
```

ModbusClose

プロトタイプ：

```
ModbusClose(id)
```

説明：

Modbusスレーブとの接続を解除し、マスターを解放します。

必須パラメータ：

id：既に作成されているマスターインデックス。

戻る：

操作結果

- 0：接続解除成功。
- 1：接続解除失敗。

例：

```
# Modbusスレーブとの接続を解除します。
ModbusClose(id)
```

GetInBits

プロトタイプ：

```
GetInBits(id, addr, count)
```

説明：

Modbusスレーブ接点レジスタアドレスの値を読み取ります。Modbus機能コード02に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：接点レジスタの開始アドレス。
- count：連続して読み出した接点レジスタの数。値の範囲：1~2000（ModbusTCPプロトコル

の制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

戻る：

接点レジスタアドレスの値は、table中に保存されます。table中の1つ目の値は、接点レジスタ開始アドレスの値に対応します。

例：

```
# アドレス0から開始して、5つのアドレスの値を連続で読み取ります。
inBits = GetInBits(id,0,5)
```

GetInRegs

プロトタイプ：

```
GetInRegs(id, addr, count, type)
```

説明：

指定されたデータタイプに基づき、Modbusスレーブ入力レジスタアドレスの値を読み取ります。Modbus機能コード04に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：入力レジスタの開始アドレス。
- count：連続して読み出した入力レジスタの数。値の範囲：1~125（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの記号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占用）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占用）

戻る：

入力レジスタアドレスの値は、table中に保存されます。table中の1つ目の値は、入力レジスタ開始アドレスの値に対応します。

例：

```
# アドレス2048から開始して、32ビットの記号なし整数を読み取ります。
data = GetInRegs(id, 2048, 2, "U32")
```

GetCoils

プロトタイプ：

```
GetCoils(id, addr, count)
```

説明：

Modbusスレーブコイルレジスタアドレスの値を読み取ります。Modbus機能コード01に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：コイルレジスタの開始アドレス。
- count：連続して読み出したコイルレジスタの数。値の範囲：1~2000（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

戻る：

コイルレジスタアドレスの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

例：

```
# アドレス0から開始して、5つのアドレスの値を連続で読み取ります。
Coils = GetCoils(id,0,5)
```

SetCoils

プロトタイプ：

```
SetCoils(id, addr, count, table)
```

説明：

指定した値をコイルレジスタが指定するアドレスに書き込みます。Modbus機能コード05（1つ書き込み）と0F（複数書き込み）に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：コイルレジスタの開始アドレス。
- count：連続して書き込んだコイルレジスタの数。値の範囲：1~1968（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）
- table：書き込むコイルレジスタの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

例：

```
# アドレス1024から開始して、5つのコイルを連続で書き込みます。
local Coils = {0,1,1,1,0}
SetCoils(id, 1024, #coils, Coils)
```

GetHoldRegs

プロトタイプ：

```
GetHoldRegs(id, addr, count, type)
```

説明：

指定したデータタイプに基づき、Modbusスレーブ保持レジスタアドレスの値を読み取ります。Modbus機能コード03に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：保持レジスタの開始アドレス。
- count：連続して読み出した保持レジスタの数。値の範囲：1~125（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの記号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占用）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占用）

戻る：

保持レジスタアドレスの値は、table中に保存されます。table中の1つ目の値は、保持レジスタの開始アドレスの値に対応します。

例：

```
# アドレス2048から開始して、32ビットの記号なし整数を読み取ります。
data = GetHoldRegs(id, 2048, 2, "U32")
```

SetHoldRegs

プロトタイプ：

```
SetHoldRegs(id, addr, count, table, type)
```

説明：

指定したデータタイプに基づいて、指定した値を保持レジスタが指定するアドレスに書き込みます。Modbus機能コード06（1つ書き込み）と10（複数書き込み）に対応します。

必須パラメータ：

- id：作成したマスターインデックス。
- addr：保持レジスタの開始アドレス。
- count：連続して書き込んだ保持レジスタの数。値の範囲：1~123（ModbusTCPプロトコルの制限により、実際の値の範囲はスレーブレジスタ数またはプロトコルの規定に基づいて確定してください）
- table：書き込むコイルレジスタの値は、table中に保存されます。table中の1つ目の値は、コイルレジスタ開始アドレスの値に対応します。

選択可能なパラメータ：

type：データタイプ。

- 空である場合、デフォルトはU16です
- U16：16ビットの記号なし整数（2バイト、1レジスタを占用）
- U32：32ビットの記号なし整数（4バイト、2レジスタを占用）
- F32：32ビット単精度浮動小数点数（4バイト、2レジスタを占用）
- F64：64ビット倍精度浮動小数点数（8バイト、4レジスタを占用）

例：

```
# アドレス2048から開始して、1つの64ビット倍精度浮動小数点数を書き込みます。
local data = {95.32105}
SetHoldRegs(id, 2048, 4, data, "F64")
```

プログラム制御

命令リスト

プログラム制御関数は、プログラム実行の制御に関連する汎用関数です。

命令	機能
Print	デバッグ情報を制御コンソールにプリントアウトします。
Wait	指定時間を待機するか、または指定条件が満たされると、次のコマンドの実行に進みます。
Pause	スクリプトの実行を一時停止
ResetElapsedTime	計時を開始
ElapsedTime	計時を終了
Systime	現在のシステム時刻の取得

Print

プロトタイプ：

```
Print(value)
```

説明：

デバッグ情報をコンソールにプリントアウトします（命令名は `print` と書くこともできます）。

必須パラメータ：

value：プリント待ちのデータ

例：

```
# 制御コンソールに文字列Successをプリントアウトします。
Print('Success')
```

Wait

プロトタイプ：

```
Wait(time_ms)
Wait(check_str)
```

```
Wait(check_str, timeout_ms)
```

説明：

ロボットアームが前の命令を完了すると、指定時間を待機するか、または指定条件が満たされると、次のコマンドの実行に進みます。

必須パラメータ：

- time_ms：パラメータ値がintegerタイプであるとき、指定待機時間を表し、0以下であるときは待機しないことを表します。単位：ms
- check_str：パラメータ値がstringタイプであるとき、ロジック判断を表し、ロジックがtrueであれは次の命令の実行に進みます。

選択可能なパラメータ：

timeout_ms：タイムアウト時間。ロジック判断がずっとfalseであり、かつ該時間を超えるまで待機すると、システムは次の命令の実行に進み、「false」が返されます。0以下であるときは、すぐにタイムアウトし、待機しないことを表します。該パラメータを設定しないとき、タイムアウトがなく、ロジック判断がtrueになるまでずっと待機します。単位：ms

戻る：

条件が満たされて実行を続行するとき、trueが返されます。条件が満たされず、タイムアウトに起因して実行を続行するときは、falseが返されます。

例：

```
# 300ms待機します。
Wait(300)
```

```
# DI1がONであるとき、実行を続行します。
Wait("DI(1) == ON")
```

```
# DO1がONでAI(1)が7未満の場合は動作を継続します。
Wait("GetDO(1) == ON and AI(1) < 7")
```

```
# 1s内のDI1の状態に応じて異なるサービスロジックを実行します。
if Wait("DI(1) == ON", 1000):
    # DI1状態がON
else:
    # DI1状態がOFFで1秒以上待機します
```

Pause

プロトタイプ：

```
Pause()
```

説明：

スクリプトの実行を一時停止します。実行を続行するには、制御ソフトウェアまたはリモート制御で操作する必要があります。

例：

```
-- ロボットアームがP1に移動した後に移動を一時停止し、外部制御によって実行を続行してP2に移動します。
MovJ(P1)
Pause()
MovJ(P2)
```

ResetElapsedTime

プロトタイプ：

```
ResetElapsedTime()
```

説明：

この命令の前のすべての命令の実行が完了するのを待機してから計時を開始します。ElapsedTime()と合わせて使用する必要があります。実行時間の計算に使用します。

例：

ElapsedTimeの例をご参照ください。

ElapsedTime

プロトタイプ：

```
ElapsedTime()
```

説明：

計時が終了し、時間差が返されます。ResetElapsedTime()と合わせて使用する必要があります。

戻り：

計時開始から計時終了までの時間差、単位はミリ秒です。最大で4294967295ms（約49.7日）まで統計が可能で、それを超えると0から再カウントします。

例：

```
# ロボットアームが現在位置からP1を経由してP2まで移動するのにかかる時間を計算し、コンソールに出力します。

ResetElapsedTime()
MovL(P1)
MovL(P2)
print(ElapsedTime())
```

Systime

プロトタイプ：

```
Systime()
```

説明：

現在のシステム時刻を取得します。

戻り：

システムの現在時刻のUnixタイムスタンプ、単位はミリ秒に変換されています。すなわち、グリニッジ標準時1970年1月1日0時から現在までのミリ秒数で、通常は時間差を計算するために使用されます。

例：

```
# 現在のシステム時刻を取得します。
time1 = Systime()
print(time1)  # 1686304295963を北京時間に変換すると、2023年6月9日17時51分35秒（963ミリ秒を追加）となります。
time2 = Systime()
print(time2)  # 1686304421968を北京時間に変換すると、2023年6月9日17時53分41秒（968ミリ秒を追加）となります。

# ロボットアームがP1に移動するのにかかる時間（ミリ秒）を計算します。
time1 = Systime()
MovL(P1)
time2 = Systime()
print(time2-time1)
```